

SphereEx-DBPlusSuite

V1.0

2023.04

目录

1 产品简介	1
2 内核原理	23
3 产品白皮书	61
4 技术白皮书	64
5 性能白皮书	71
6 快速入门	145
7 插件指南	174
8 开发指南	297
9 运维指南	358
10 最佳实践	462
11 Driver 使用手册	484
12 参考引用	532
13 相关协议	663
14 FAQ	671

1.1 行业背景

伴随数字化转型步入“深水区”，企业的业务需求迭代迅速，数据量和并发访问量呈指数级增长，传统关系型数据库遇到捉襟见肘的扩展能力。数据对于企业的重要性越来越高，一方面企业对于以数据库为核心的专业服务需求也愈发迫切。另一方面企业也翘首以待新型数据库来满足不同的数据场景应用需求。

数据库在传统关系型流派的基础上迅速裂变，进入百花齐放、百家争鸣的时代。但由于企业数据库技术体系庞杂，各个数据库之间间隙明显，管理困难，尤其是数据库在企业中的角色向来都是成本中心，控制成本、提高效能，是企业对底层数据库的长期不变的诉求。因此在多方位的作用下，企业需要服务提供商能够提供横向主流数据库产品和纵向多版本技术服务覆盖能力，以面对来自无感解决用户侧多元化需求以及抽离行业侧共性问题并形成解决方案的诉求。

在上述情况下，对于数据的利用和底层架构的转型，就显得尤为迫切。想办法提升数据库的灵活性与扩展性，打造组件化数据库，或许是一种可行的方式。SphereEx 公司产品正是迎合了上述发展趋势，打造以连接、增强、可插拔为核心理念的数据库增强计算平台。

1.2 产品族说明



Fig. 1: SphereEx 产品族

SphereEx-DBPlusSuite 构建在数据库生态之上，在存储端将各种数据库作为存储节点，在接入端满足数据库协议及 SQL 方言，并在这层为数据库应用架构提供增量能力。

SphereEx-DBPlusSuite 商业产品套件包含三款产品：SphereEx-Boot、SphereEx-Console、SphereEx-DBPlusEngine。

■ SphereEx-Boot

SphereEx-Boot 简称（Boot）用于方便管理 SphereEx DBPlusEngine-Proxy 集群和 SphereEx-Console 的命令行工具。

■ SphereEx-Console

SphereEx-Console 简称（Console）是应用于对 SphereEx-DBPlusEngine 进行管理控制的可视化操作平台，提供更为易用的使用体验；同时，构建起以 SphereEx-DBPlusEngine 为核心的整体解决方案，包括针对资源、实例、插件等多种功能封装，为用户带来一站式的使用体验。

■ SphereEx-DBPlusEngine

SphereEx-DBPlusEngine 是在开源内核 ShardingSphere 基础上，增强部分企业级功能后封装而成，可为企业提供增强数据服务能力，包括但不限于数据分片、数据安全、数据库替换、弹性伸缩等。它由 SphereEx-DBPlusEngine-Driver、SphereEx-DBPlusEngine-Proxy 这 2 款既能够独立部署，又支持混合部署配合使用的产品组成。它们均提供标准化的数据水平扩展、分布式事务和分布式治理等功能，可适用于如 Java 同构、异构语言、云原生等各种多样化的应用场景。

- SphereEx-DBPlusEngine-Proxy
独立部署形态，已数据库“代理”的方式工作，应用可通过原生“数据库”的方式访问使用。
- SphereEx-DBPlusEngine-Driver
以 SDK 的形式工作，可与 Java 应用无缝集成。

本文档主要以 SphereEx-DBPlusEngine-Proxy 介绍为主。SphereEx-DBPlusEngine-Driver 的介绍，可参考独立章节[Driver 版使用手册](#)。

1.3 产品发展史

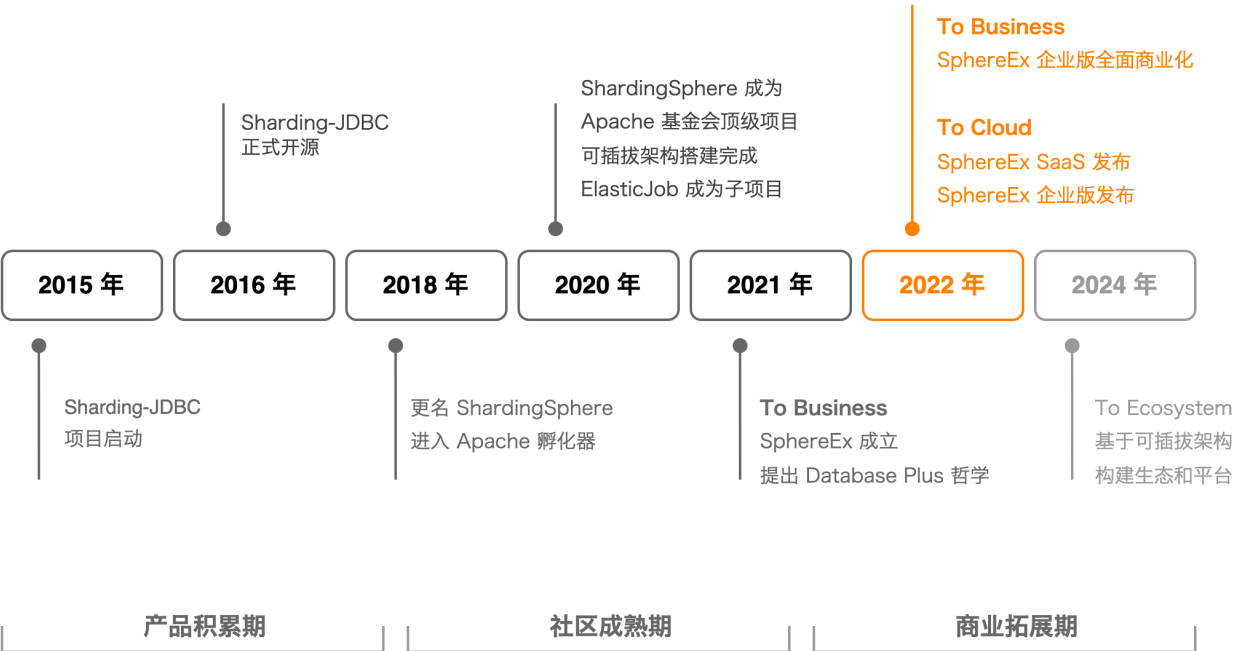


Fig. 2: 发展历程

1.4 技术架构

SphereEx-DBPlusSuite 商业套件的技术架构图：

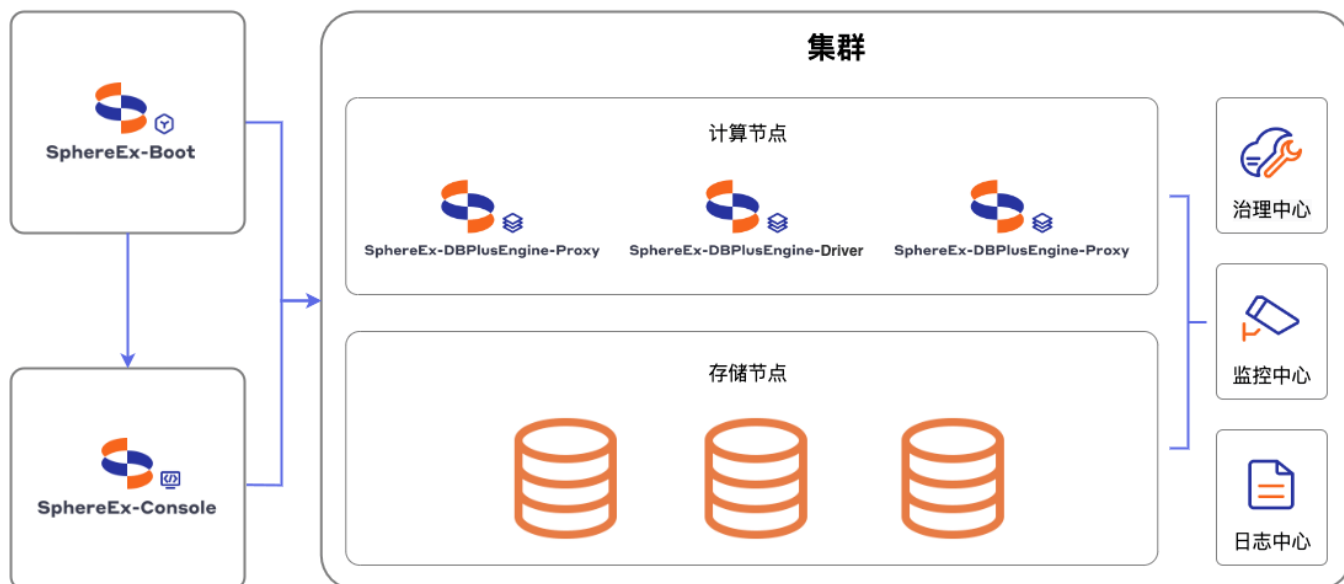


Fig. 3: 技术架构图

1.4.1 集群生态工具

- SphereEx-Boot：字符界面集群管理工具
- SphereEx-Console：图形化集群管理工具

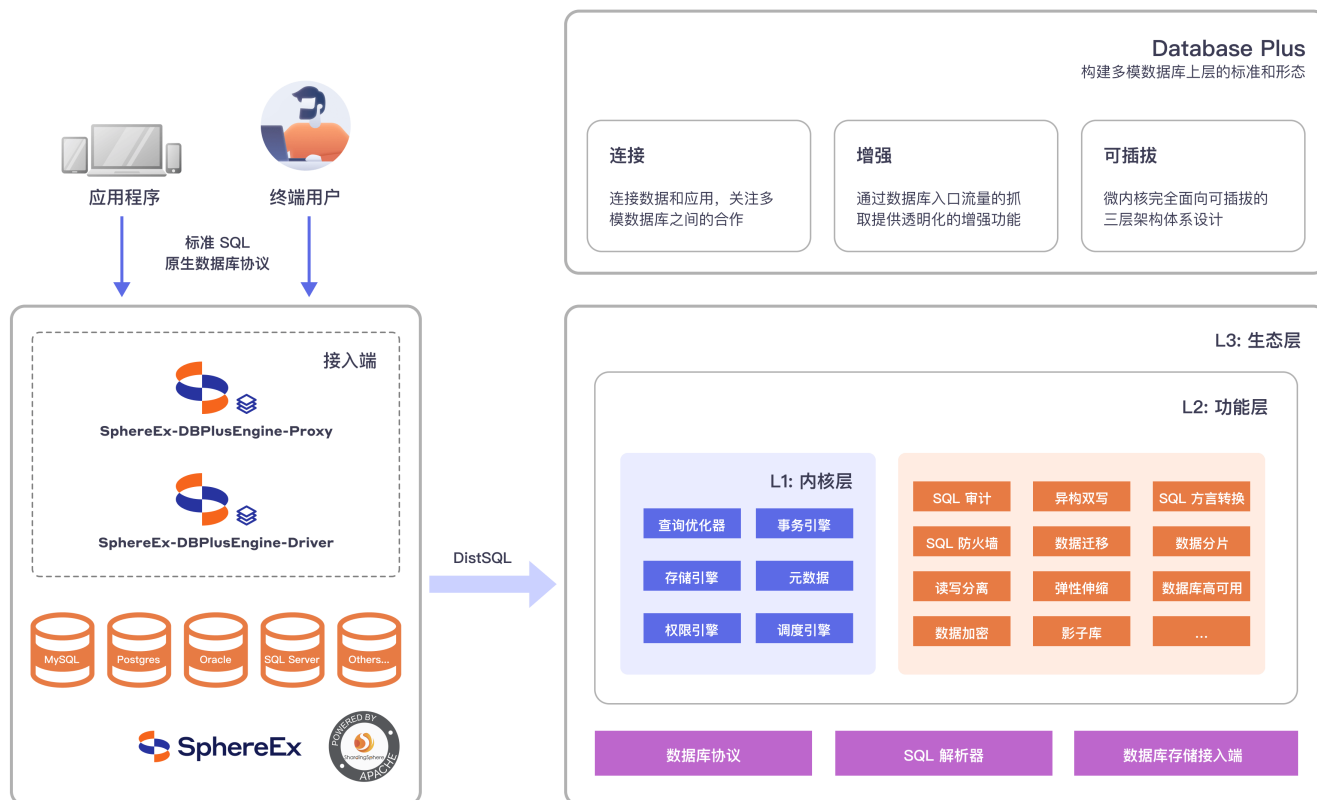
1.4.2 集群组件

- 计算节点 (Compute Node)，即 SphereEx-DBPlusEngine-Proxy 和 SphereEx-DBPlusEngine-Driver，本文档中特指 SphereEx-DBPlusEngine-Proxy
- 存储节点 (Storage Node)，底层数据库，例如：MySQL、PostgreSQL 等。
- 治理中心 (Governance Center)，底层使用 ZooKeeper、ETCD 或自研组件实现，主要提供分布式的元数据，分布式事务管理、维护账号、权限等安全信息。
- 监控中心 (Monitor Center)，底层使用 Prometheus 实现对集群的监控。
- 日志中心 (Log Center)，底层使用 Elasticsearch + Logstash + Filebeat 实现对计算节点日志的管理。

1.4.3 SphereEx-DBPlusEngine 整体架构

DBPlusEngine 的可插拔架构划分为 3 层，分别是 L1 内核层、L2 功能层、L3 生态层。

DBPlusEngine 整体架构如下图所示。



L1 内核层

是数据库基本能力的抽象，其所有组件均必须存在，但具体实现方式可通过可插拔的方式更换。主要包括查询优化器、分布式事务引擎、分布式执行引擎、权限引擎和调度引擎等。

L2 功能层

用于提供增量能力，其所有组件均是可选的，可以包含零至多个组件。组件之间完全隔离，互无感知，多组件可通过叠加的方式相互配合使用。主要包括数据分片、读写分离、数据库高可用、数据加密、影子库等。用户自定义功能可完全面向 DBPlusEngine 定义的顶层接口进行定制化扩展，而无需改动内核代码。

L3 生态层

用于对接和融入现有数据库生态，包括数据库协议、SQL 解析器和存储适配器，分别对应于 DBPlusEngine 以数据库协议提供服务的方式、SQL 方言操作数据的方式以及对接存储节点的数据库类型。

DBPlusEngine 的接入形态包括驱动端和代理端两种形态，即 DBPlusEngine-Driver 和 DBPlusEngine-Proxy。

DBPlusEngine-Driver

轻量级的 Java 框架，在 Java 的 JDBC 层提供的额外服务。它使用客户端直连数据库，以 jar 包形式提供服务，无需额外部署和依赖，可理解为增强版的 JDBC 驱动，完全兼容 JDBC 和各种 ORM 框架。

- 适用于任何基于 JDBC 的 ORM 框架，如：JPA、Hibernate、Mybatis、Spring JDBC Template 或直接使用 JDBC；
- 支持任何第三方的数据库连接池，如：DBCP，C3P0，BoneCP，HikariCP 等；
- 支持任意实现 JDBC 规范的数据库，目前支持 MySQL，PostgreSQL，Oracle，SQLServer 以及任何可使用 JDBC 访问的数据库。

DBPlusEngine-Proxy

透明化的数据库代理端，提供封装了数据库二进制协议的服务端版本，用于完成对异构语言的支持。目前提供 MySQL 和 PostgreSQL（兼容 openGauss 等基于 PostgreSQL 的数据库）版本，它可以使用任何兼容 MySQL/PostgreSQL 协议的访问客户端（如：MySQL Command Client、MySQL Workbench、Navicat 及 DBeaver 等）操作数据，对 DBA 更加友好。

- 向应用程序完全透明，可直接当做 MySQL/PostgreSQL 使用；
- 适用于任何兼容 MySQL/PostgreSQL 协议的客户端。

1.4.4 安全体系

DBPlusEngine-Proxy（下称 Proxy）提供了完善的安全体系，兼顾 Engine 安全和数据安全两大部分。

登录认证

Proxy 拥有严格的登录认证机制，只有经过身份认证的用户才可以成功建立连接。

密码认证

默认情况下，Proxy 使用密码认证方式，登录用户需提供正确的用户名和密码。

特别的，由于 Proxy 支持多种数据库协议（如 MySQL、PostgreSQL 等），当用户应用不同的数据库客户端时，Proxy 能够自动适配密码通信协议，在复杂场景下为用户提供一致的安全体验。

主机限制

管理员可以为 Proxy 的用户限制登录主机地址，以提高安全等级。如：

```
- !AUTHORITY
users:
- user: root@127.0.0.1
password: root
```

以上配置指定了 root 用户只能从 127.0.0.1 这个地址访问 Proxy，从其他地址登录时，即使密码正确也将被拒绝。

LDAP 认证

为方便企业用户进行统一认证管理，Proxy 也提供了 LDAP (Lightweight Directory Access Protocol) 认证方式，LDAP 认证现已支持 MySQL 和 PostgreSQL 客户端。同时，Proxy 允许用户以非常灵活的方式来接入 LDAP，例如：

1. 可以配置为默认使用 LDAP，这样所有用户都通过 LDAP 认证；
2. 支持为用户配置 auth 属性，指定用户使用密码或 LDAP 认证，各个用户可以使用不同方式；
3. 各个用户可以使用不同的 LDAP 认证器，即对接不同的 LDAP 服务；
4. 支持为用户指定 DN 模板，满足复杂场景的需要；
5. 支持 LDAPS 协议，可进一步提升安全等级。

管理安全

在 DBPlusEngine-Proxy 中，用户能够通过 DistSQL 执行多种维度的管理操作，包括但不限于：1. Proxy 配置管理，如事务类型、日志开关等；2. 逻辑库管理；3. 存储资源管理；4. 数据分片规则管理；5. 读写分离规则管理；6. 加解密规则管理；7. 数据库发现规则管理；8. 影子库规则管理；9. 元数据查看等。

由于 DistSQL 功能强大，数据库管理员可为不同用户分配不同的 DistSQL 权限，以实现 Proxy 管理安全。例如：

```
GRANT DIST SHOW SHARDING ON sharding_db.* TO 'sharding'@'%';
```

通过以上授权语句，将逻辑库 sharding_db 中的“查看分片规则”的权限授予 'sharding@%'，则该用户能够在 sharding_db 中执行 SHOW SHARDING TABLE RULES、SHOW SHARDING BINDING TABLE RULES 等 SHARDING 相关的 RQL，但无法执行其他未授权的 DistSQL。

例如 'sharding@%' 用户此时执行 CREATE SHARDING TABLE RULE 语句，将会得到异常提示：

```
Access denied for operation [CREATE] of subject sharding_db.table_name:SHARDING.]
```

若要向 'sharding@%' 用户授予所有 DistSQL 权限，可以通过如下方式：

```
GRANT DIST RDL,RQL,RAL ON sharding_db.* TO 'sharding'@'%';
```

访问安全

数据访问安全，是企业级数据库必备的能力之一。DBPlusEngine-Proxy 作为分布式数据库集群的入口，为用户提供了全面的访问控制能力。与传统集中式数据库或单协议数据库不同，DBPlusEngine 拥有管理多类型底层数据库和对接多协议客户端的能力，在进行访问控制时就会面临诸多挑战：

- 不同数据库类型拥有不同的逻辑概念；
- 不同数据库类型使用不同的方言；
- 不同数据库提供了不同的存储结构。

为了给用户提供一致的安全体验，Proxy 屏蔽了底层数据库的差异，提供统一易用的安全体系，具有如下特点：

- 细粒度权限管理：支持库级、表级、列级访问控制；
- 统一的交互语言：使用 DistSQL 进行用户和权限的管理，适用于不同的数据库协议；
- 存储独立：权限的信息保存在 DBPlusEngine 的治理中心，不依赖于底层数据库；
- 实时生效：对用户和权限的管控实时生效，无需重启或手动刷新。

例如，若管理员向 'sharding@%' 授予 sharding_db 中 t_order 表的查询和写入权限，可以执行这样的 DistSQL：

```
GRANT DIST SELECT, INSERT ON sharding_db.t_order TO 'sharding'@'%';
```

操作完成之后，'sharding@%' 用户立即获得相应授权。若用户执行未授权的操作，如 DELETE，将会收到拒绝提示：

```
=> DELETE FROM sharding_db.t_order WHERE id = 1;  
Access denied for operation [DELETE] of subject sharding_db.t_order]
```

存储安全

近年来，数据安全和隐私保护越来越受到重视，处理数据加密和数据脱敏也成为了许多企业的一项重要任务。在这样的趋势下，技术团队面临了新的挑战：

- 加密的过程，在应用端还是在数据库？
- 如果是应用端，各个项目团队都会有编码任务，后期修改算法影响的范围也极大；
- 如果在数据库，不同数据库的加密方式不同，企业中将存在多种加密手段，不能很好的复用。

为此，DBPlusEngine 提供了崭新的思路，通过在数据引擎中支持可配置的加密规则，在简化开发团队工作量的同时，提供了更通用的数据安全服务。DBPlusEngine 中的数据加密功能具有众多优势：

- 对应用无侵入，不需修改源码；
- 广泛可用，适用于任何通过 DBPlusEngine 接入的数据库；
- 多种加密算法可选，包括 SHA、SM3、SM4 等内置算法支持；
- 定制灵活：用户可根据 SPI 自定义加密算法，满足个性化需求。

以用户手机号加密场景为例，在应用接入 DBPlusEngine 后，只需要通过 YAML 或 DistSQL 定义一个 ENCRYPT RULE，如：

```
CREATE ENCRYPT RULE t_user (  
  COLUMNS(  
    (NAME=mobile,CIPHER=mobile,TYPE(NAME="AES",PROPERTIES("aes-key-value"="123456abc"))));
```

这样，当应用向 t_user 表写入新记录时，DBPlusEngine 会自动对 mobile 字段进行加密，存储密文内容。反之，查询时也能将密文自动解密，应用得到的查询结果将会恢复为明文。

这样，既保证了应用对数据加密过程的无感，又保证了存储数据的安全。如果发生“脱库”一类的黑客事件，泄露的也仅仅是密文内容，安全风险大大降低。

1.4.5 架构优势

- 高性能

驱动程序端历经多年打磨，效率接近原生 JDBC，性能极致。主流竞品中，目前只有 DBPlusEngine 提供驱动端接入形态。

- 高兼容

代理端适用于任何兼容 MySQL/PostgreSQL 协议的客户端，驱动程序端支持任意实现 JDBC 规范的数据库。

- 高扩展

面对数据库替换场景，DBPlusEngine 可满足业务平滑迁移，对业务零侵入。

- 低成本

保留原有数据库技术栈，对 DBA 友好，学习及管理成本低。

- 安全稳定

不干预数据库内核，基于成熟的数据库底座提供增能力，兼顾安全性及稳定性。

- 弹性扩展

具备弹性的计算和存储能力，可满足计算及存储层不断变化的需求，在线完成数据库拆分和迁移；

- 开放生态

可插拔模型，使内核、功能组件以及生态对接完全能够灵活的方式进行插拔式扩展，用户可以像使用积木一样定制适合业务的独特系统。

1.5 部署架构

集群部署架构图：

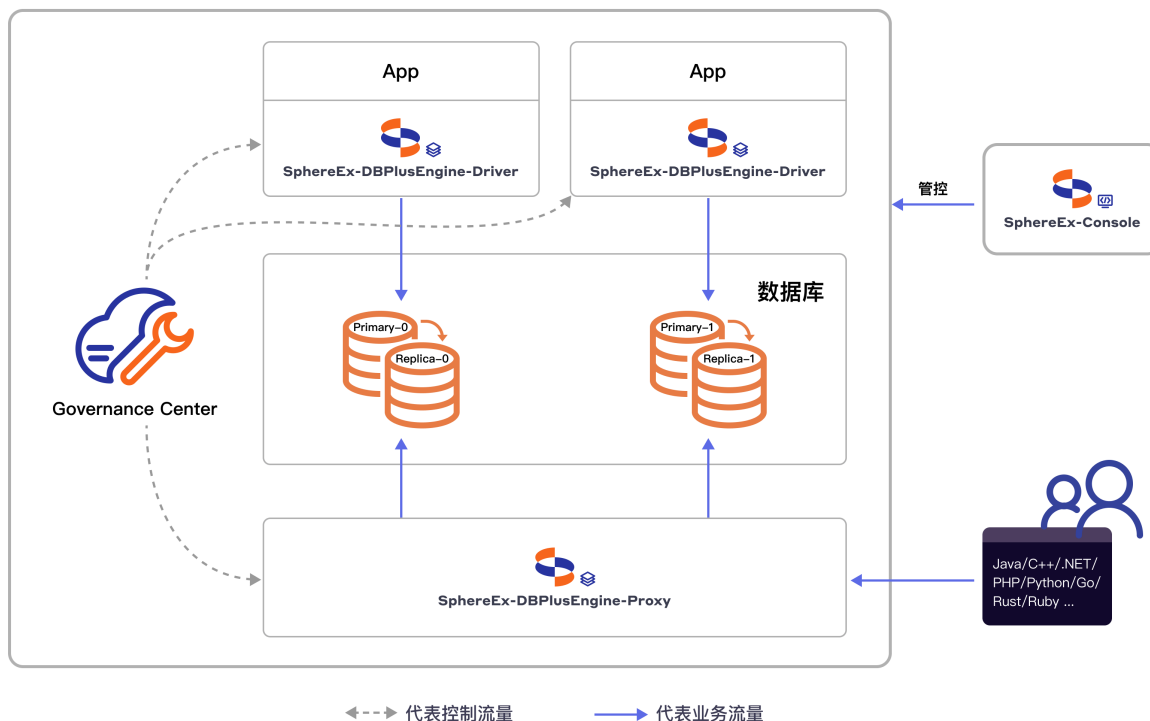


Fig. 4: 部署架构图

1.5.1 DBPlusEngine-Driver 产品独立部署

定位为轻量级 Java 框架，在 Java 的 Driver 层提供的额外服务。它使用客户端直连数据库，以 jar 包形式提供服务，无需额外部署和依赖，可理解为增强版的 JDBC 驱动，完全兼容 JDBC 和各种 ORM 框架。

- 适用于任何基于 JDBC 的 ORM 框架，如：JPA、Hibernate、Mybatis、Spring JDBC Template 或直接使用 JDBC；
- 支持任何第三方的数据库连接池，如：DBCP、C3P0、BoneCP、HikariCP 等；
- 支持任意实现 JDBC 规范的数据库，目前支持 MySQL、PostgreSQL、Oracle、SQLServer 以及任何可使用 JDBC 访问的数据库。

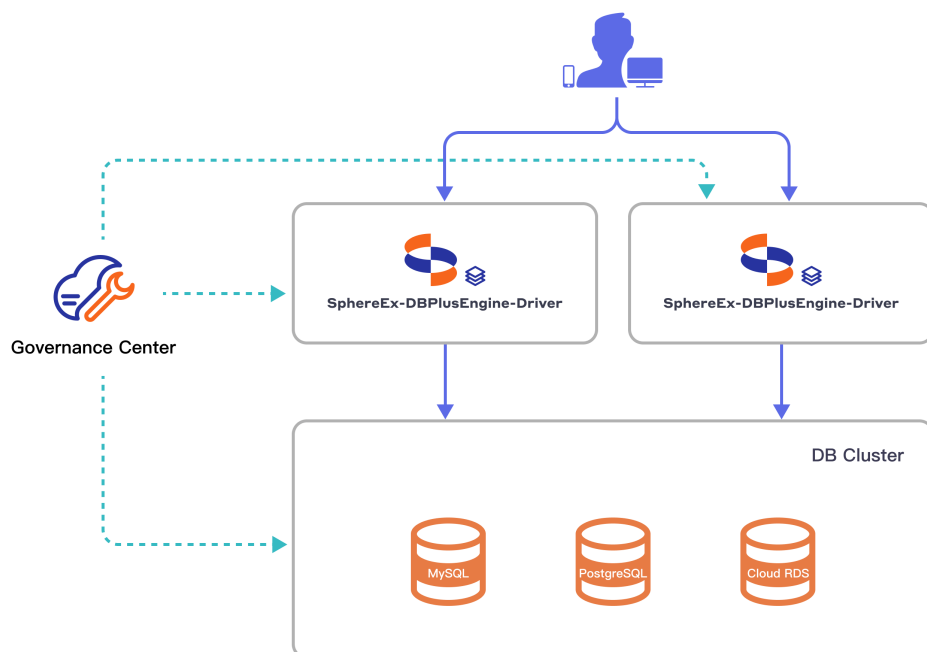


Fig. 5: DBPlusEngine-Driver Architecture

	DBPlusEngine-Driver	DBPlusEngine-Proxy
数据库	任意	MySQL/PostgreSQL
连接消耗数	高	低
异构语言	仅 Java	任意
性能	损耗低	损耗略高
无中心化	是	否
静态入口	无	有

DBPlusEngine-Driver 的优势在于对 Java 应用的友好度。

1.5.2 DBPlusEngine-Proxy 产品独立部署

定位为透明化的数据库代理端，提供封装了数据库二进制协议的服务端版本，用于完成对异构语言的支持。目前提供 MySQL 和 PostgreSQL（兼容 openGauss 等基于 PostgreSQL 的数据库）版本，它可以使用任何兼容 MySQL/PostgreSQL 协议的访问客户端（如：MySQL Command Client, MySQL Workbench, Navicat 等）操作数据，对 DBA 更加友好。

- 向应用程序完全透明，可直接当做 MySQL/PostgreSQL 使用；
- 适用于任何兼容 MySQL/PostgreSQL 协议的客户端。

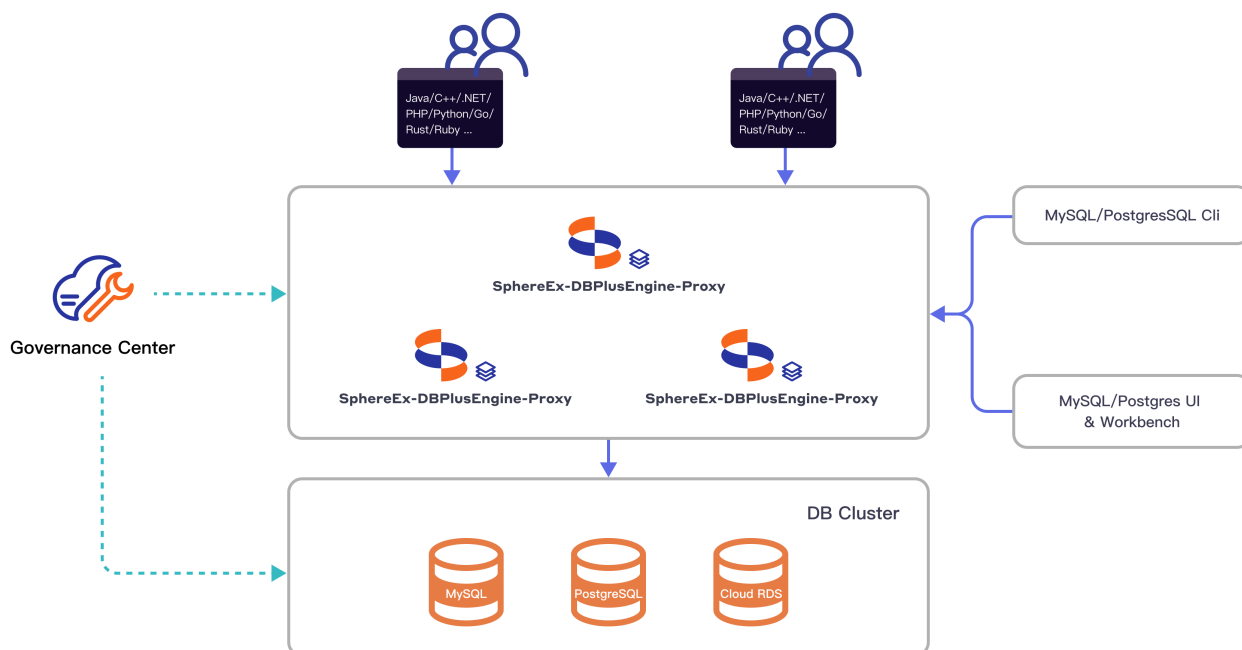


Fig. 6: DBPlusEngine-Proxy Architecture

DBPlusEngine-Proxy 的优势在于对异构语言的支持，以及为 DBA 提供可操作入口。

1.5.3 DBPlusEngine-Driver 与 DBPlusEngine-Proxy 产品混合部署

DBPlusEngine 是多接入端共同组成的生态圈。通过混合使用 DBPlusEngine-Driver 和 DBPlusEngine-Proxy，并采用同一注册中心统一配置分片策略，能够灵活的搭建适用于各种场景的应用系统，使得架构师更加自由地调整适合于当前业务的最佳系统架构。

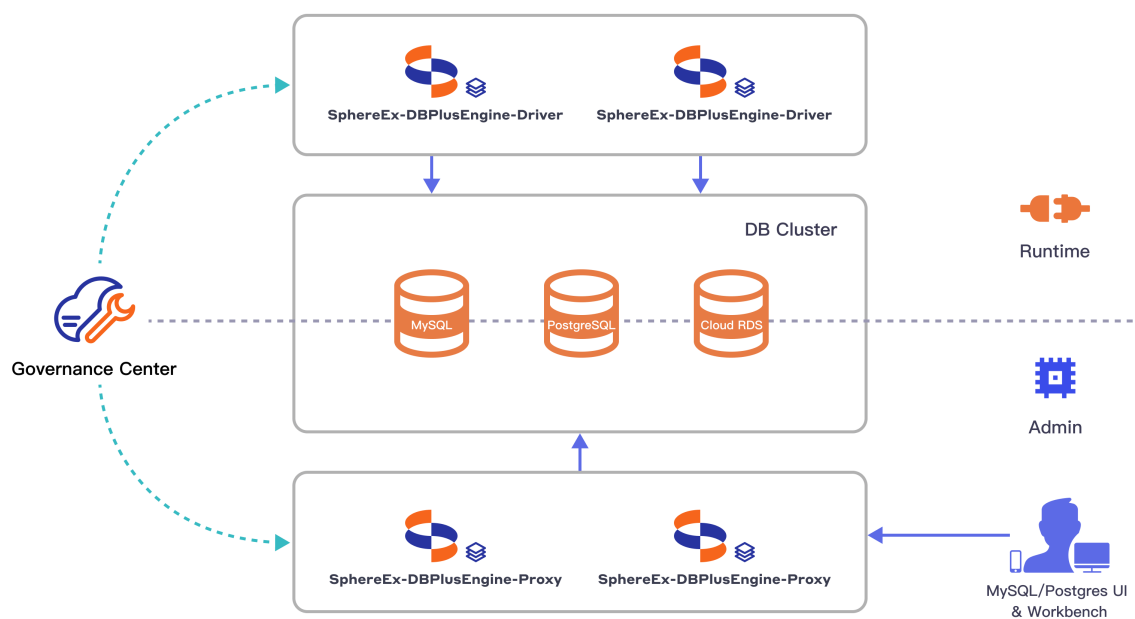


Fig. 7: DBPlusEngine Hybrid Architecture

1.6 功能架构

SphereEx-Boot 功能架构图:

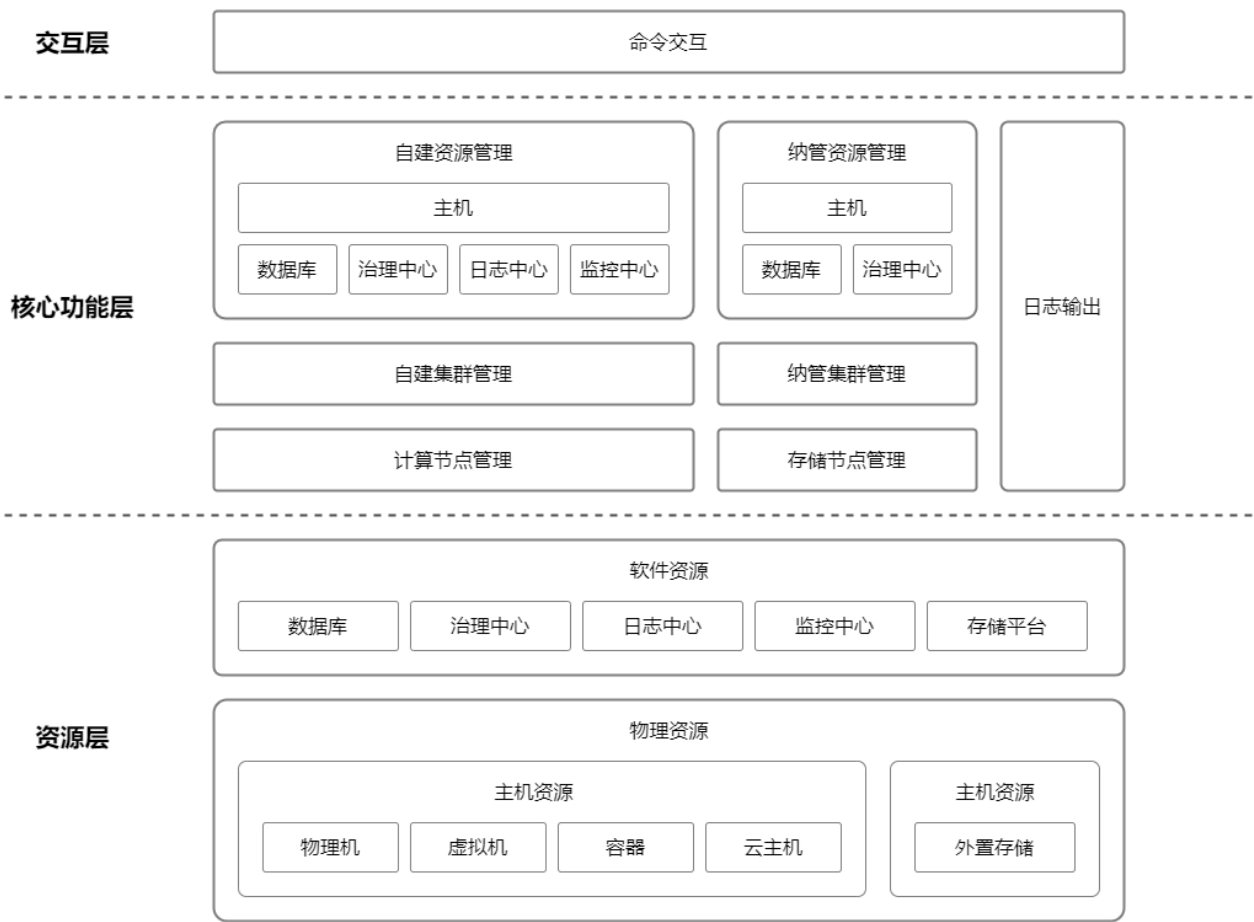


Fig. 8: SphereEx-Boot 功能架构图

SphereEx-Console 功能架构图：

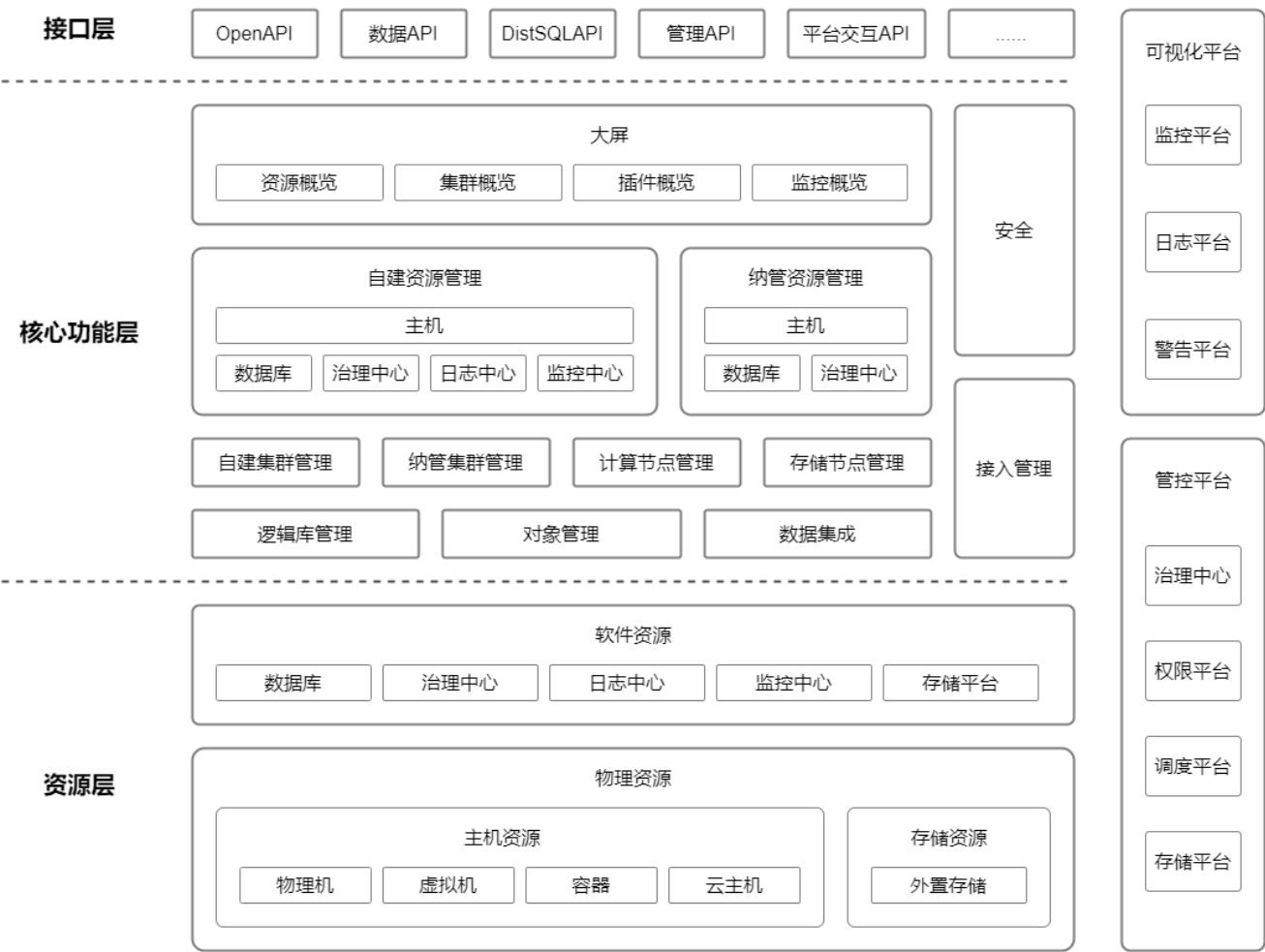


Fig. 9: SphereEx-Console 功能架构图

SphereEx-DBPlusEngine 功能架构图：

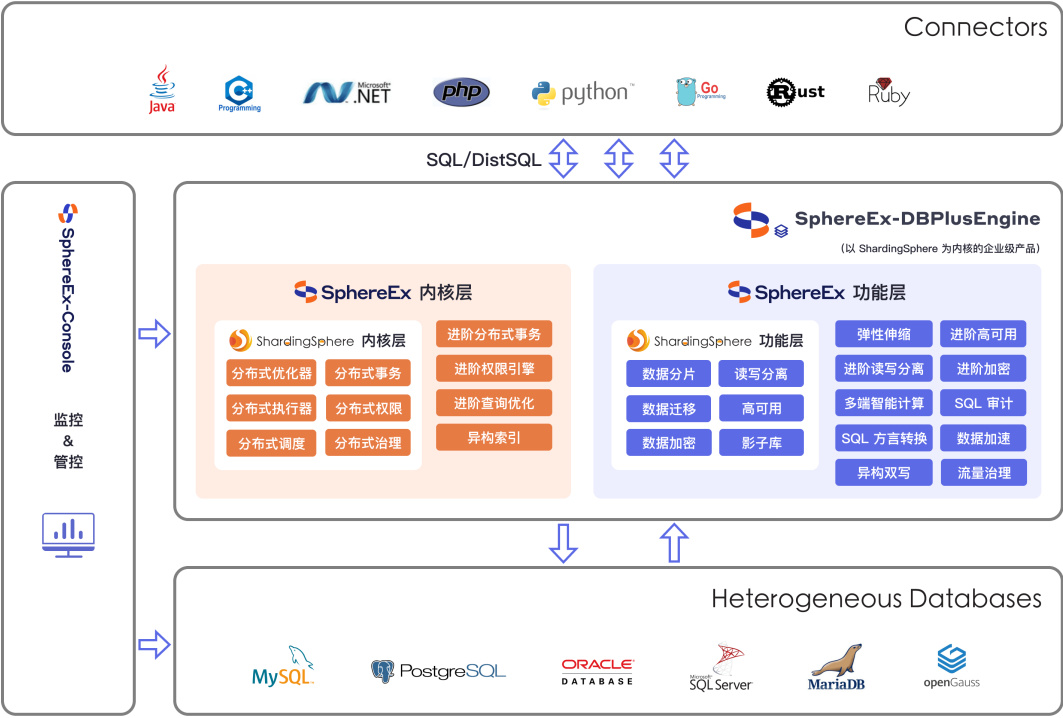


Fig. 10: DBPlusEngine 功能架构图

1.7 功能列表

如下列表是 SphereEx-DBPlusSuite 支持的所有功能，其中终端管控部分为 SphereEx-Boot 产品所具备功能，可视化管控和监控诊断部分为 SphereEx-Console 产品所具备功能，其余为 SphereEx-DBPlusEnginge 产品所具备功能。

一级功能	二级功能
方言兼容性	MySQL
	PostgreSQL
	Oracle
	SQL Server
	openGauss
	SQL 92/2003 标准
	国产库
数据库协议支持	MySQL
	PostgreSQL
SQL 扩展	SQL Hint
	SQL 方言转换
	标准 DistSQL
	增强 DistSQL
分布式事务	XA 事务
	增强 XA 事务
	柔性事务
	全局事务及 MVCC
查询优化器	计算下推
	联邦查询
数据集成	全局 CDC
数据分片	基础分片策略

continues on next page

Table 1 – continued from previous page

一级功能	二级功能
	定制化分片策略
	自动化分片管理
	预分片能力
	自动数据重分布
	异构二级索引
	一键分布式
数据安全（存储加密）	数据存储加密
	开源加密算法
	商业加密算法
	自定义数据类型及长
	查询辅助列简单密态计算
	复杂密态计算
	加密洗数
	本地密钥管理
	云端密钥管理
	敏感数据识别
数据安全（权限）	用户管理
	角色管理
	语句级别权限控制
	库级别权限控制
	表级别权限控制
	列级别权限控制
	行级别权限控制
	在线变更权限
	LDAP 对接
	第三方权限对接
行为安全（审计）	用户行为审计（事后）
	主动式审计（事前）
影子库压测	测试生产数据隔离
读写分离	读写访问分离
	延迟探测及一致性访问
	内置精准延迟探测能力
数据迁移	全量数据迁移
	分布式高性能迁移
	单实例一致性校验
	分布式高性能校验
弹性伸缩	单实例数据重分布
	分布式数据重分布
	单实例数据重分布
	单实例一致性校验
	分布式高性能校验
	语句流控
	负载流控
元数据管理	流量切换与控制
	在线强一致变更
	云上去中心化
终端管控	管理集群组件
	管理集群
可视化管控	管理主机
	管理集群组件
	管理集群
	管理 SQL
	管理日志
高可用	计算节点高可用
	存储节点高可用
	存储节点故障感知与自愈

continues on next page

Table 1 – continued from previous page

一级功能	二级功能
监控诊断	主机级监控
	组件级监控
	语句级监控（慢查询）
	增强监控指标
	监控报警对接
	链路跟踪（APM）

1.8 开源版与商业版的区别

模块	一级功能	二级功能	开源	商业
内核功能	方言兼容性	MySQL	支持	支持
		PostgreSQL	支持	支持
		Oracle	支持	支持
		SQL Server	支持	支持
		openGauss	支持	支持
		SQL 92/2003 标准	支持	支持
		国产库	不支持	支持
	数据库协议支持	MySQL	支持	支持
		PostgreSQL	支持	支持
	SQL 扩展	SQL Hint	支持	支持
		SQL 方言转换	实验功能	开发中
		标准 DistSQL	支持	支持
		增强 DistSQL	不支持	支持
	分布式事务	XA 事务	支持	支持
		增强 XA 事务	不支持	支持
		柔性事务	支持	支持
		全局事务及 MVCC	不支持	规划中
	查询优化器	计算下推	支持	支持
		联邦查询	不支持	开发中
	流量管理	智能流量路由	不支持	支持
		流量复制双写	不支持	不支持
	数据集成	全局 CDC	实验功能	开发中
插件功能	数据分片	基础分片策略	支持	支持
		定制化分片策略	支持	支持
		自动化分片管理	不支持	开发中
		预分片能力	不支持	开发中
		自动数据重分布	不支持	支持
		异构二级索引	不支持	规划中
		一键分布式	不支持	规划中
	数据安全（存储加密）	数据存储加密	支持	支持
		开源加密算法	支持	支持
		商业加密算法	不支持	支持
		自定义数据类型及长度	不支持	支持
		查询辅助列	支持	支持
		简单密态计算	支持	支持
		复杂密态计算	不支持	开发中
		加密洗数	不支持	支持
		本地密钥管理	支持	支持
		云端密钥管理	不支持	支持
		敏感数据识别	不支持	开发中
	数据安全（权限）	用户管理	支持	支持
		角色管理	不支持	支持
		语句级别权限控制	不支持	支持

continues on next page

Table 2 – continued from previous page

模块	一级功能	二级功能	开源	商业
		库级别权限控制	支持	支持
		表级别权限控制	不支持	支持
		列级别权限控制	不支持	支持
		行级别权限控制	不支持	不支持
		在线变更权限	不支持	支持
		LDAP 对接	不支持	支持
		第三方权限对接	不支持	支持
		用户行为审计（事后）	不支持	支持
	行为安全（审计）	主动式审计（事前）	不支持	支持
	影子库压测	测试生产数据隔离	支持	支持
	读写分离	读写访问分离	支持	支持
		延迟探测及一致性访问	不支持	支持
		内置精准延迟探测能力	不支持	开发中
	数据迁移	全量数据迁移	支持	支持
		分布式高性能迁移	不支持	开发中
		单实例一致性校验	支持	支持
		分布式高性能校验	不支持	开发中
	弹性伸缩	单实例数据重分布	不支持	支持
		分布式数据重分布	不支持	支持
		单实例数据重分布	不支持	开发中
		单实例一致性校验	不支持	支持
		分布式高性能校验	不支持	开发中
		语句流控	不支持	支持
		负载流控	不支持	开发中
		流量切换与控制	不支持	支持
管控功能	元数据管理	在线强一致变更	不支持	开发中
		云上去中心化	不支持	支持
	终端管控	管理集群组件	支持	支持
		管理集群	支持	支持
	可视化管控	管理主机	不支持	支持
		管理集群组件	不支持	支持
		管理集群	不支持	支持
		管理 SQL	不支持	支持
		管理日志	不支持	支持
		计算节点高可用	不支持	规划中
	高可用	存储节点高可用	不支持	规划中
		存储节点故障感知与自愈	支持	支持
	监控诊断	主机级监控	不支持	支持
		组件级监控	不支持	支持
		语句级监控（慢查询）	不支持	支持
		增强监控指标	不支持	支持
		监控报警对接	不支持	支持
		链路跟踪（APM）	支持	支持
服务	服务内容	问题解答	支持	支持
		架构咨询	不支持	支持
		培训	不支持	支持
		远程故障支持	不支持	支持
		现场故障支持	不支持	支持
		安装及升级服务	不支持	支持
		现场保障服务	不支持	支持

1.9 产品优势

1.9.1 SphereEx-DBPlusEngine

- 旨在构建异构数据库上层的标准和生态

SphereEx-DBPlusEngine 基于 Database Plus 产品理念，旨在构建异构数据库上层标准和生态。它关注如何合理利用数据库计算和存储能力，而并非实现一个全新的数据库。其定位是站在数据库上层视角，更多关注于数据库与应用间协作。

- 在原有关系型数据库基础上提供扩展和增强

SphereEx-DBPlusEngine 旨在充分合理地在分布式的场景下利用关系型数据库的计算和存储能力，并非实现一个全新的关系型数据库。关系型数据库当今依然占有巨大市场份额，是企业核心系统的基石，未来也难于撼动，我们更加注重在原有基础上提供增量，而非颠覆。

- SphereEx-DBPlusEngine 支持多态接入

SphereEx-DBPlusEngine-Driver 采用无中心化架构与应用程序共享资源，SphereEx-DBPlusEngine-Proxy 独立于应用程序部署。SphereEx-DBPlusEngine 适用于标准数据库接入方式，提供对多种语言支持，适合于 OLTP 以及部分 OLAP 类场景。

1.9.2 SphereEx-Boot

- 快速上手

只需简单命令，即可运行 SphereEx-DBPlusEngine 集群及包含组件。

- 易运维

通过图形化方式对 SphereEx-DBPlusEngine 集群进行管理，大幅降低运维成本。

- 易扩展

提供标准的水平扩展功能，可通过增加服务器数量动态对集群进行算力的扩容。

1.9.3 SphereEx-Console

- 功能易用

提高了用户体验，避免了配置错误，用户不需要通过配置和命令操作 SphereEx-DBPlusEngine，易用性得到极大提升。

- 功能全面

提供了对 SphereEx-DBPlusEngine 从基础资源到插件能力的管控，形成了统一的解决方案。

- 监控可视化

原生提供可视化的监控数据面板，可以实时在线查看 SphereEx-DBPlusEngine 的集群、组件、主机、数据库的监控数据。

1.10 版本说明

说明：分别从产品维度和组件维度介绍 SphereEx-DBPlusSuite 包含的产品版本和产品部署的环境和组件依赖的情况。

1.10.1 产品版本

产品	版本
SphereEx-DBPlusSuite	1.1.0+
SphereEx-DBPlusEngine	1.1.0、1.1.2、1.2.0、1.3.0、1.4.0
SphereEx-Boot	1.1.x
SphereEx-Console	1.1.x

1.10.2 集群组件版本

组件/资源	组件	版本
主机	node-exporter	1.3.1
监控中心	Prometheus	2.32.0
治理中心	ZooKeeper	3.7.0+
	ZK-export	无
日志中心	Elasticsearch	7.5.2
	Logstash	7.5.2
	Filebeat	7.5.2
存储节点	MySQL	5.7+
	MySQL-exporter	0.14.0

1.10.3 环境依赖

SphereEx-DBPlusEngine 环境依赖

依赖	版本
OS	Linux、Windows、MacOS
Java 环境	1.8+

1.10.4 SphereEx-Boot 环境依赖

依赖	版本
OS	Linux
Python 环境	2.7+ 3.5+

1.10.5 SphereEx-Console 环境依赖

依赖	版本
OS	Linux、Windows
Java 环境	1.8+
数据库 (MySQL)	8.+ 5.7+

1.11 应用场景

SphereEx-DBPlusSuite 的应用场景：

■ 数据分片

数据库容量达到单体瓶颈场景下，通过横向水平扩展，可以有效解决单体数据存储容量问题，并可实现平滑数据迁移。

■ 数据安全

面临日益严重数据安全问题，通过多维度的数据加密能力，解决数据存储安全问题；并通过完善的数据行为的审计能力。

■ 信创改造

提升开发效率，降低开发成本，缩短开发周期，降低运行风险。统一数据库访问管理，降低使用风险，便于使用追踪。降低应用访问及数据安全风险。兼容多款数据库产品（MySQL、PG、SQLServer、Oracle、openGauss、KingBase…）

1.12 术语表

术语	说明
资源	集群运行所需的硬件软件资源（主机、数据库）。
集群	由治理中心、监控中心、日志中心、计算节点、存储节点组成，其中监控中心和日志中心为可选项，且计算节点属于治理中心的同一 namespace。
集群生态工具	SphereEx-Boot 和 SphereEx-Console 为生态组件。
集群组件	组成集群的各个组件，即（治理中心、监控中心、日志中心、计算节点、存储节点）。
最小化集群	仅包含治理中心和计算节点。
治理中心	使用 ZooKeeper 实现用于存储计算节点的元数据信息。
监控中心	使用 Prometheus 实现用于监控集群。
监控插件	部署在客户端，采集监控信息至监控中心。
日志中心	使用 Elasticsearch+Logstash 实现，用于存储计算节点的日志。
日志中心插件	使用 Filebeat 部署在计算节点，采集日志通过 logstash 传输至 Elasticsearch 存储。
计算节点	Compute Node（别名：代理节点 Proxy）一个计算节点，对应于 SphereEx-DBPlusEngine 部署单元，由 IP、端口确定的一组资源。单个计算节点，承担 SQL 路由、数据合并、数据聚合等功能。一个 SphereEx-DBPlusEngine 集群，对应多个计算节点。本文档中仅涉及 SphereEx-DBPlusEngine-Proxy
存储节点	Storage Node 物理对应一个数据库实例下的某个 Schema，同之前的“数据源”概念。即一组 IP、端口、用户名、密码、模式（Schema）约定下的对象资源集合。注意：对应模式下面，而非实例下面；即可对应到物理数据库实例下面的多个模式。针对 SphereEx-DBPlusSuite 来说，这一角色就是承担数据存储的职能。
逻辑库	Logical Database（别名：Schema）一个逻辑库，是由一或多个存储节点组成的虚拟库。一个 SphereEx-DBPlusEngine 集群，可对应多个逻辑数据库。

术语	说明
对象	逻辑库下管理的逻辑对象，可类比数据库中的对象。
插件	可插拔架构上的数据应用功能，包括但不限于数据分片、读写分离等，可以作用于逻辑库、对象等。
分片表	逻辑表相同结构的水平拆分数据库（表）的逻辑名称，是 SQL 中表的逻辑标识。
广播表	指所有的分片数据源中都存在的表，表结构及其数据在每个数据库中均完全一致。
表组	指分片规则一致的主表和子表。使用绑定表进行多表关联查询时，必须使用分片键进行关联，否则会出现笛卡尔积关联或跨库关联，从而影响查询效率。例如：t_order 表和 t_order_item 表，均按照 order_id 分片，并且使用 order_id 进行关联，则此两张表互为绑定表关系。被绑定表之间的多表关联查询不会出现笛卡尔积关联，关联查询效率将大大提升。
单表	指所有的存储节点中仅唯一存在的表，适用于数据量不大且无需分片的表。
Dist-SQL	SphereEx-DBPlusEngine 特有的操作语言。它与标准 SQL 的使用方式完全一致，用于提供增量功能的 SQL 级别操作能力

针对若干内核层面的技术要点进行说明，原则上是对用户可感知的内核功能，是需要写的，其他可不需要。重点说明原理。

2.1 架构理念: Database Plus

Database Plus 是一种设计哲学理念，该理念致力于构建数据库上层的标准和生态，在生态中补充数据库所缺失的能力。其与 Apache ShardingSphere 及 SphereEx-DBPlus Engine 的关系如下。- Database Plus：是 ShardingSphere 的设计哲学；- ShardingSphere：是 Database Plus 的工程化实现；- SphereEx-DBPlusEngine：是在 ShardingSphere 基础上面向企业用户的增强产品。

作为创建分布式数据库系统的指导性概念，Database Plus 在已经呈现出碎片化趋势的数据库之上，去建立全新的连接关系。这个过程将在无入侵上层应用前提下，在数据库上层提供数据增强服务，如分布式、数据控制和流量控制等。对于存储层，需要为用户保留了原有数据库技术栈，稳定性、兼容性及运维习惯等均无变化，可靠性也有更好的保证。这样，应用程序只需要与标准服务层对话，而不需要再关心底层数据库间的差异和能力，形成一个闭环生态环境。

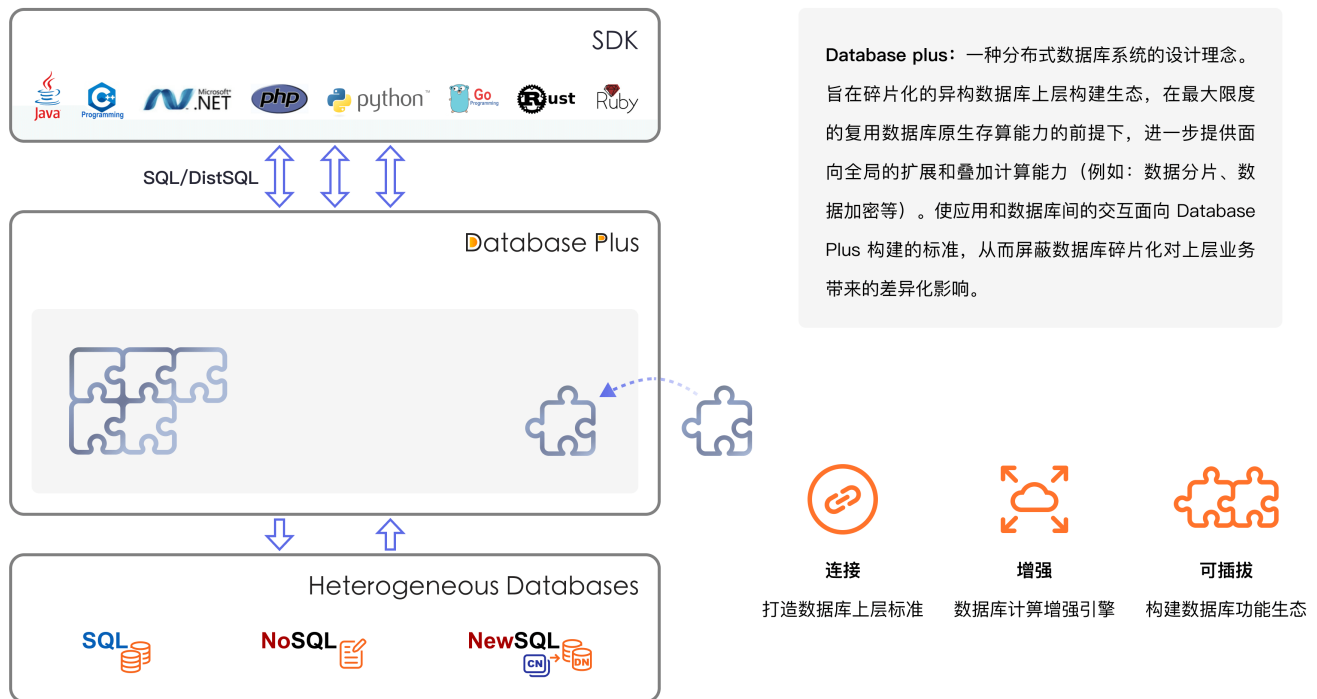


Fig. 1: 设计哲学

Database Plus 的三个核心特性分别是连接、增强和可插拔。由“点”到“线”，最后形成一个生态“面”。

■ 连接：打造数据库上层标准

连接是 Database Plus 能力的基础“点”。即以『数据库网关』的形态为应用系统提供统一数据库入口，提供一个可以适配于各种数据库 SQL 方言和访问协议的中间层，Database Plus 将 SQL 解析而成的 AST（抽象语法树）根据其他数据库方言的规则重新生成 SQL。它可屏蔽异构数据库方言差异，也可屏蔽业务中异构开发语言的差异。向上，以标准数据库协议支持多种开发语言，可做到应用无感平滑使用；向下，支持多种数据库产品既满足北向多种开发语言，同时还支持南向多种数据库产品。目前已支持 MySQL、PostgreSQL、openGauss 等数据库协议，以及 MySQL、PostgreSQL、openGauss、SQL Server、Oracle 和所有支持 SQL92 标准的 SQL 方言。

■ 增强：数据库计算增强引擎

基于连接的“点”，可以进一步延伸到这一条增强功能“线”。传统数据库的实践经验是时间和多场景所沉淀的精华，Database Plus 在复用数据库的存储和原生计算能力基础上，再对其算力、容量及功能进行突破。当流量进入到了网关后，Database Plus 通过全局化的能力在分布式、数据控制和流量控制三个方面进行增强。数据分片、弹性伸缩、高可用、读写分离、分布式事务及基于垂直拆分的异构数据库联邦查询等功能，都是 Database Plus 在分布式的异构数据库全局层面下所能够提供的增强能力，以排列组合的方式提供给用户使用。

■ 可插拔：构建数据库功能生态

面向用户，Database Plus 以一个“面”的形态去呈现，即生态面。我们可尝试把这个“面”理解为所熟知的应用商店，你只需在众多的插件中选择所需要的功能即可，它可以单独使用，也可以组合使用，甚至你可以在应用商店中打造自己的定制化插件。通过可插拔体系，Database Plus 将能够真正的构建面向数据库的功能生态，将异构数据库的全局能力统一纳管。它不仅面向集中式数据库的分布式化，也同时面向分布式数据库的竖井功能一体化。可插拔的特性可在提供更好扩展的同时，还能做到收敛，只提供用户所需要的内容。生态无边界，能力可插拔。

2.2 内核处理流程

SphereEx-DBPlusEngine 内核处理流程包括标准的 SQL Parser 和 SQL Binder 模块，这两个模块用于 SQL 具体特征的识别，并根据其结果，将 SQL 执行流程分为 Simple Push Down Engine 和 SQL Federation Engine。

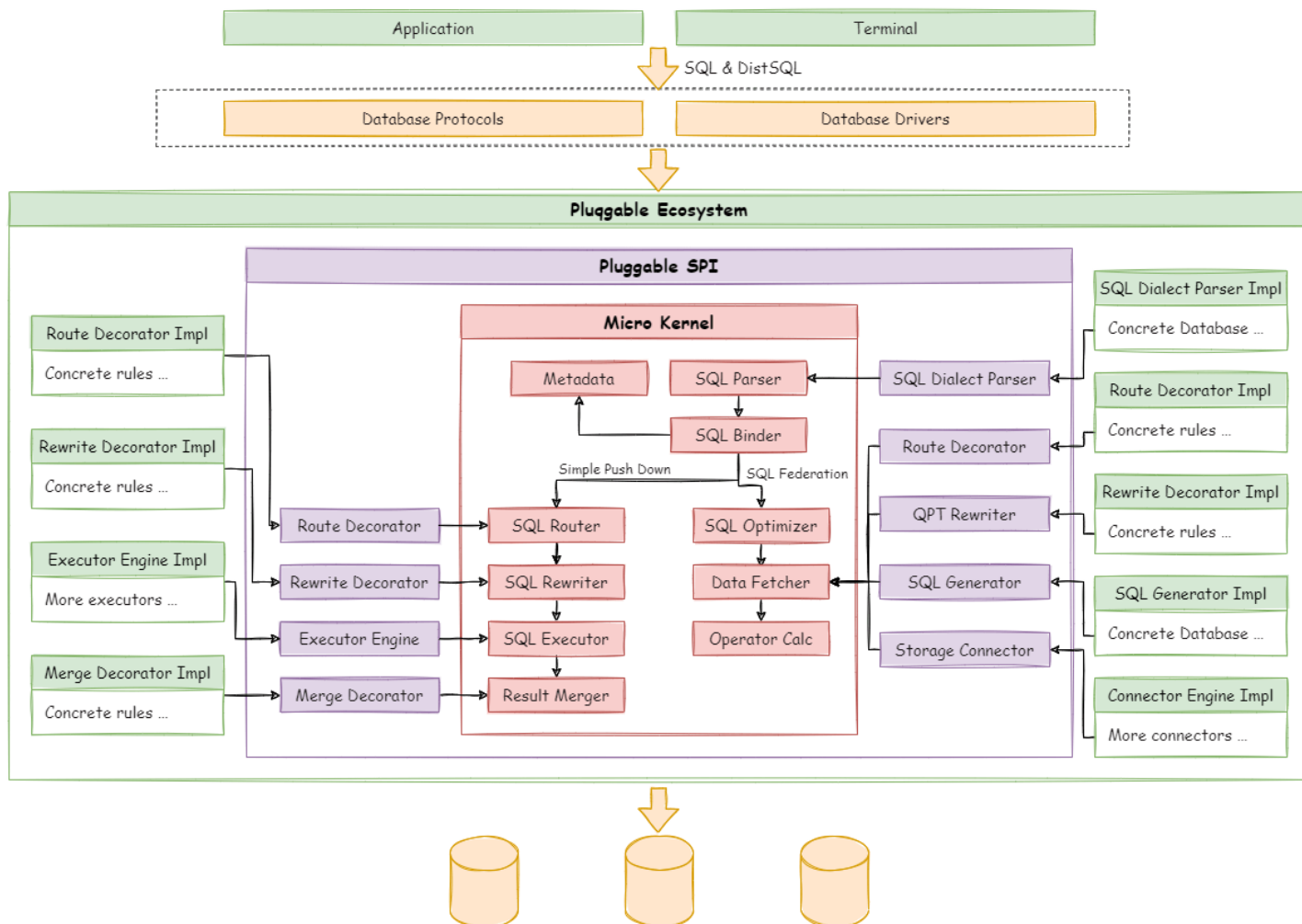


Fig. 2: 内核处理流程

在微内核的流程中，SQL Parser 负责将用户输入的 SQL 通过 Lexer 和 Parser 的标准流程，转化为 AST (Abstract Syntax Tree) Node，并最终转化为提取必要特征的 SQLStatement，它是 Apache ShardingSphere 内核处理的核心输入。

SQLStatement 是 SQL 原义的体现，而 SQL Binder 则结合 Metadata 与 SQLStatement，补充 SQL 中的通配符和缺失部分，生成完整的符合数据库表结构的 AST Node。SQL Parser 通过分析 SQL 是否包含关联查询和子查询等基本信息，SQL Binder 则通过分析逻辑表和物理数据库的从属关系，最终确定 SQL 请求是否具备跨越多数据源操作的可能性。当 SQL 可以经过逻辑表修改、执行信息补全等操作即可以全文下推至数据库存储节点时，采用 Simple Push Down Engine，用于保证 SQL 的最大兼容度；反之，当 SQL 涉及跨库关联和跨库子查询时，则采用 SQL Federation Engine，用于在分布式表关联的操作中获取更加的性能。

2.2.1 Simple Push Down 下推流程

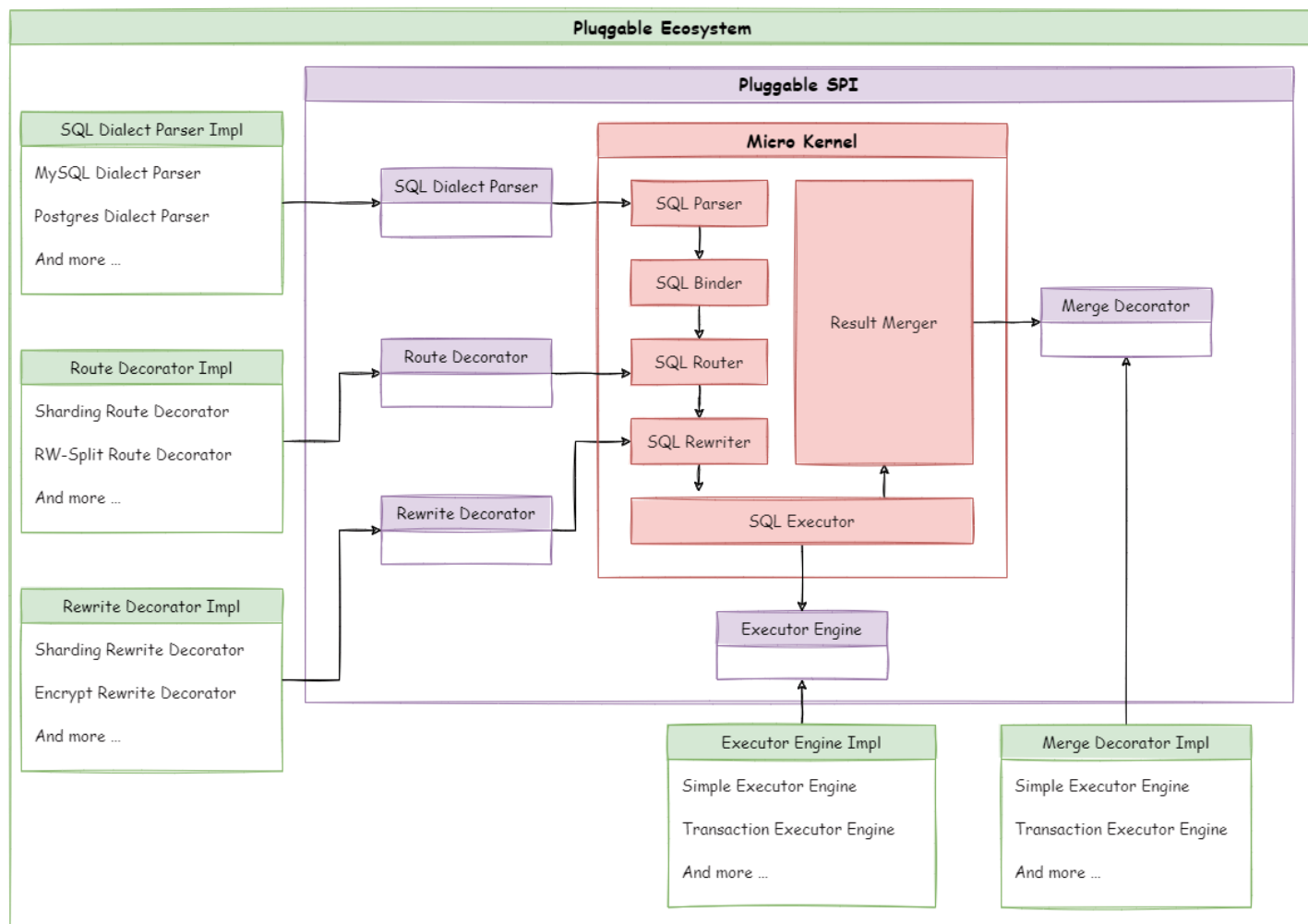


Fig. 3: Simple Push Down 下推流程

Simple Push Down 下推流程由 SQL 解析 => SQL 绑定 => SQL 路由 => SQL 改写 => SQL 执行 => 结果归并组成，主要用于处理标准分片场景下的 SQL 执行。

在经过了标准的 SQL Parser 和 SQL Binder 前置流程后，SQL Router 会提取 SQLStatement 中的关键字段（如：分片键），并与用户通过 DistSQL 配置的具体规则相匹配，计算最终路由的数据源。

在知晓数据所在的数据源之后，SQL Rewriter 负责将 SQL 改写为可以在分布式场景下直接下推到数据库执行的 SQL，比如逻辑表名替换、补列、聚合函数修正等。

SQL Executor 则根据当前请求的事务状态选择适合的执行引擎，并将改写完成后的 SQL 以并发和分组的方式，发送至路由结果指向的数据源。最终，当 SQL 的执行结果全部完成后，Result Merger 会自动将多个结果集进行聚合归并或进一步改写。

2.2.2 SQL Federation 执行引擎流程

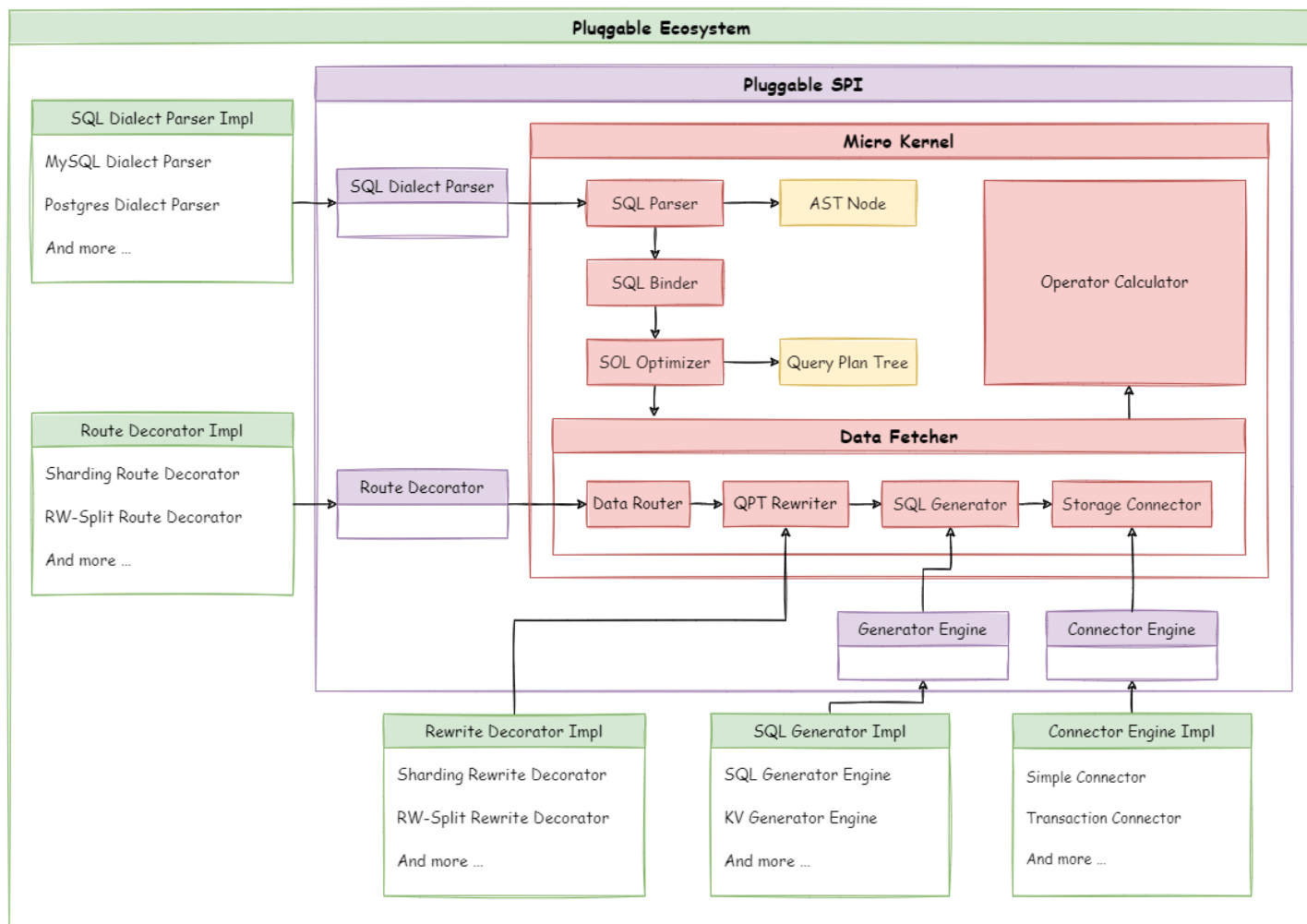


Fig. 4: SQL Federation 执行引擎流程

SQL Federation Engine 与 Simple Push Down Engine 最大的不同在于 SQL Optimizer，它将 AST Node 通过 RBO (Rule Based Optimizer) 和 CBO (Cost Based Optimizer) 两种方式进行优化，生成 Query Plan Tree。

从存储节点中获取数据，并非通过原始 SQL，而是根据 Query Plan Tree 重新生成能够在单一数据节点执行的 SQL，并根据路由结果，发送至存储节点。在生成最终 SQL 之前，Query Plan Tree Rewriter 将根据用户通过 DistSQL 配置的规则对其中的表或列进行修改。值得注意的是，目标数据源并非必须与用户输入的 SQL 方言保持一致，可以通过 Query Plan Tree 生成适用于存储节点的全新 SQL 方言，比如：其他数据库方言，甚至 Key-Value 等。

执行结果返回至 SphereEx-DBPlusEngine 的计算节点，并通过 Operator Calculator 在内存中进行最后的聚合计算，直至结果集最终返回。

2.3 分布式事务

2.3.1 背景

数据库事务需要满足 ACID（原子性、一致性、隔离性、持久性）四个特性。- 原子性（Atomicity）指事务作为整体来执行，要么全部执行，要么全不执行；- 一致性（Consistency）指事务应确保数据从一个一致的状态转变为另一个一致的状态；- 隔离性（Isolation）指多个事务并发执行时，一个事务的执行不应影响其他事务的执行；- 持久性（Durability）指已提交的事务修改数据会被持久保存。在单一数据节点中，事务仅限于对单一数据库资源的访问控制，称之为本地事务。几乎所有的成熟的关系型数据库都提供了对本地事务的原生支持。但是在基于微服务的分布式应用环境下，越来越多的应用场景要求对多个服务的访问及其相对应的多个数据库资源能纳入到同一个事务当中，分布式事务应运而生。关系型数据库虽然对本地事务提供了完美的 ACID 原生支持。但在分布式的场景下，它却成为系统性能的桎梏。如何让数据库在分布式场景下满足 ACID 的特性或找寻相应的替代方案，是分布式事务的重点工作。

2.3.2 事务分类

本地事务

在不开启任何分布式事务管理器的前提下，让每个数据节点各自管理自己的事务。它们之间没有协调以及通信的能力，也并不互相知晓其他数据节点事务的成功与否。本地事务在性能方面无任何损耗，但在强一致性以及最终一致性方面则力不从心。

两阶段提交

XA 协议最早的分布式事务模型是由 X/Open 国际联盟提出的 X/Open Distributed Transaction Processing (DTP) 模型，简称 XA 协议。基于 XA 协议实现的分布式事务对业务侵入很小。它最大的优势就是对使用方透明，用户可以像使用本地事务一样使用基于 XA 协议的分布式事务。XA 协议能够严格保障事务 ACID 特性。严格保障事务 ACID 特性是一把双刃剑。事务执行在过程中需要将所需资源全部锁定，它更加适用于执行时间确定的短事务。对于长事务来说，整个事务进行期间对数据的独占，将导致对热点数据依赖的业务系统并发性能衰退明显。因此，在高并发的性能至上场景中，基于 XA 协议的分布式事务并不是最佳选择。

柔性事务

如果将实现了 ACID 的事务要素的事务称为刚性事务的话，那么基于 BASE 事务要素的事务则称为柔性事务。BASE 是基本可用、柔性状态和最终一致性这三个要素的缩写。- 基本可用（Basically Available）保证分布式事务参与方不一定同时在线；- 柔性状态（Soft state）则允许系统状态更新有一定的延时，这个延时对客户来说不一定能够察觉；- 最终一致性（Eventually consistent）通常是通过消息传递的方式保证系统的最终一致性。

在 ACID 事务中对隔离性的要求很高，在事务执行过程中，必须将所有的资源锁定。柔性事务的理念则是通过业务逻辑将互斥锁操作从资源层面上移至业务层面。通过放宽对强一致性要求，来换取系统吞吐量的提升。基于 ACID 的强一致性事务和基于 BASE 的最终一致性事务都不是银弹，只有在最适合的场景中才能发挥它们的最大长处。可通过下表详细对比它们之间的区别，以帮助开发者进行技术选型。

	本地事务	两(三)阶段事务	柔性事务
业务改造	无	无	实现相关接口
一致性	不支持	支持	最终一致
隔离性	不支持	支持	业务方保证
并发性能	无影响	严重衰退	略微衰退
适合场景	业务方处理不一致	短事务 & 低并发	长事务 & 高并发

2.3.3 核心概念

XA 事务

两阶段事务提交采用的是 X/OPEN 组织所定义的 DTP 模型所抽象的 AP（应用程序），TM（事务管理器）和 RM（资源管理器）概念来保证分布式事务的强一致性。其中 TM 与 RM 间采用 XA 的协议进行双向通信。与传统的本地事务相比，XA 事务增加了准备阶段，数据库除了被动接受提交指令外，还可以反向通知调用方事务是否可以被提交。TM 可以收集所有分支事务的准备结果，并于最后进行原子提交，以保证事务的强一致性。下图为两阶段提交模型

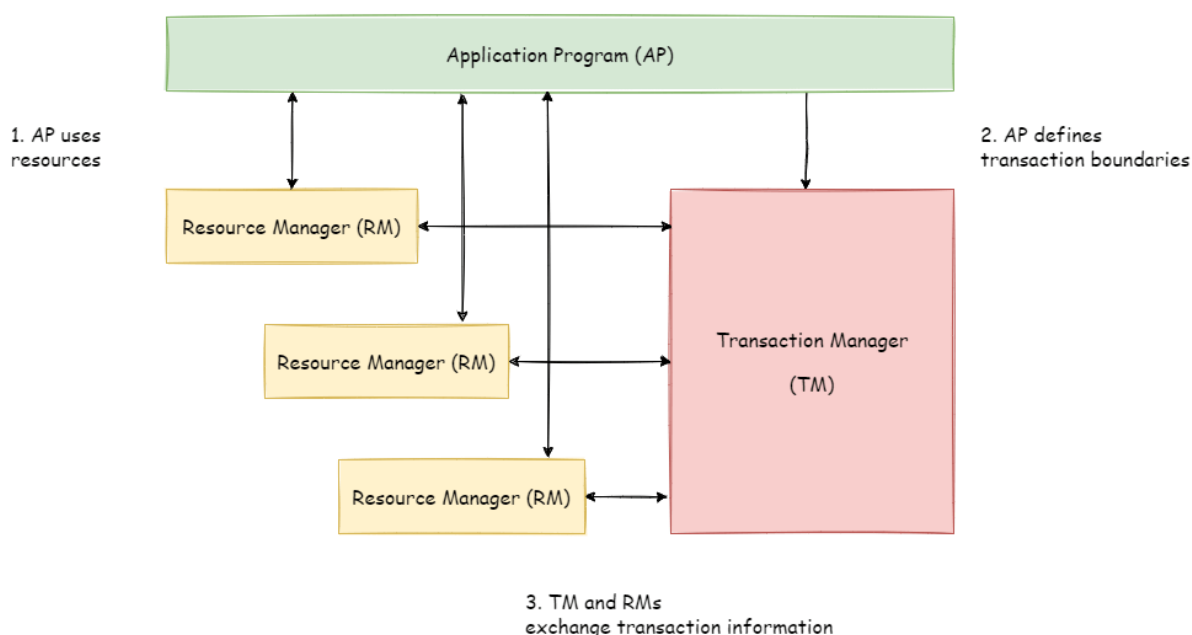


Fig. 5: 两阶段提交模型

Java 通过定义 JTA 接口实现了 XA 模型，JTA 接口中的 Resource Manager 需要数据库厂商提供 XA 驱动实现，Transaction-Manager 则需要事务管理器的厂商实现，传统的事务管理器需要同应用服务器绑定，因此使用的成本很高。而嵌入式的事务管理器可以通过 jar 形式提供服务，同 DBPlusEngine 集成后，可保证分片后跨库事务强一致性。通常，只有使用了事务管理器厂商所提供的 XA 事务连接池，才能支持 XA 的事务。DBPlusEngine 在整合 XA 事务时，采用分离 XA 事务管理和连接池管理的方式，做到对应用程序的零侵入。

柔性事务

柔性事务在 2008 年发表的一篇论文中被最早提到，它提倡采用最终一致性放宽对强一致性的要求，以达到事务处理并发度的提升。TCC 和 Saga 是两种常见实现方案。他们主张开发者自行实现对数据库的反向操作，来达到数据在回滚时仍能够保证最终一致性。SEATA 实现了 SQL 反向操作的自动生成，可以使柔性事务不再必须由开发者介入才能使用。DBPlusEngine 集成了 SEATA 作为柔性事务的使用方案。

2.3.4 XA 事务

XAShardingSphereTransactionManager 为 DBPlusEngine 的分布式事务的 XA 实现类。它主要负责对多数据源进行管理和适配，并且将相应事务的开启、提交和回滚操作委托给具体的 XA 事务管理器。

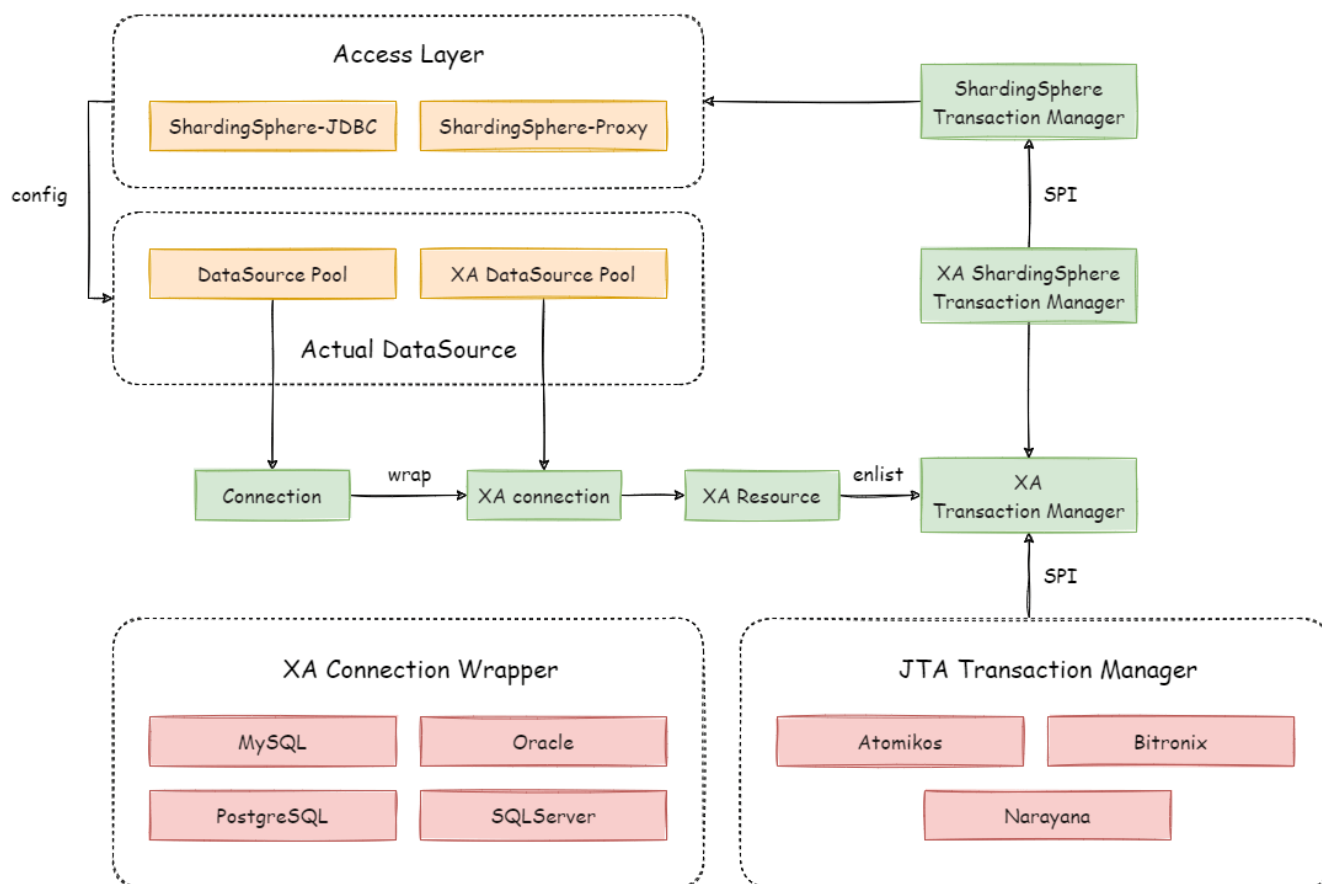


Fig. 6: XA 事务

开启全局事务

收到接入端的 `set autoCommit=0` 时，`XAShardingSphereTransactionManager` 将调用具体的 XA 事务管理器开启 XA 全局事务，以 `XID` 的形式进行标记。

执行真实分片 SQL

`XAShardingSphereTransactionManager` 将数据库连接所对应的 `XAResource` 注册到当前 XA 事务中之后，事务管理器会在此阶段发送 `XAResource.start` 命令至数据库。数据库在收到 `XAResource.end` 命令之前的所有 SQL 操作，会被标记为 XA 事务。

例如：

```
XAResource1.start ## Enlist 阶段执行
statement.execute("sql1"); ## 模拟执行一个分片 SQL1
statement.execute("sql2"); ## 模拟执行一个分片 SQL2
XAResource1.end ## 提交阶段执行
```

示例中的 `sql1` 和 `sql2` 将会被标记为 XA 事务。

提交或回滚事务

`XAShardingSphereTransactionManager` 在接收到接入端的提交命令后，会委托实际的 XA 事务管理进行提交动作，事务管理器将收集到的当前线程中所有注册的 `XAResource`，并发送 `XAResource.end` 指令，用以标记此 XA 事务边界。接着会依次发送 `prepare` 指令，收集所有参与 `XAResource` 投票。若所有 `XAResource` 的反馈结果均为正确，则调用 `commit` 指令进行最终提交；若有任意 `XAResource` 的反馈结果不正确，则调用 `rollback` 指令进行回滚。在事务管理器发出提交指令后，任何 `XAResource` 产生的异常都会通过恢复日志进行重试，以保证提交阶段的操作原子性，和数据强一致性。

```
XAResource1.prepare ## ack: yes
XAResource2.prepare ## ack: yes
XAResource1.commit
XAResource2.commit
XAResource1.prepare ## ack: yes
XAResource2.prepare ## ack: no
XAResource1.rollback
XAResource2.rollback
```

2.3.5 Seata 柔性事务

柔性事务在 2008 年发表的一篇[论文](#)中被最早提到，它提倡采用最终一致性放宽对强一致性的要求，以达到事务处理并发度的提升。

TCC 和 Saga 是两种常见实现方案。他们主张开发者自行实现对数据库的反向操作，来达到数据在回滚时仍能够保证最终一致性。[SEATA](#) 实现了 SQL 反向操作的自动生成，可以使柔性事务不再必须由开发者介入才能使用。

DBPlusEngine 集成了 SEATA 作为柔性事务的使用方案。

整合 Seata AT 事务时，需要将 TM、RM 和 TC 的模型融入 DBPlusEngine 的分布式事务生态中。在数据库资源上，Seata 通过对接 `DataSource` 接口，让 JDBC 操作可以同 TC 进行远程通信。同样，DBPlusEngine 也是面向 `DataSource` 接口，对用户配置的数据源进行聚合。因此，将 `DataSource` 封装为基于 Seata 的 `DataSource` 后，就可以将 Seata AT 事务融入到 DBPlusEngine 的分片生态中。

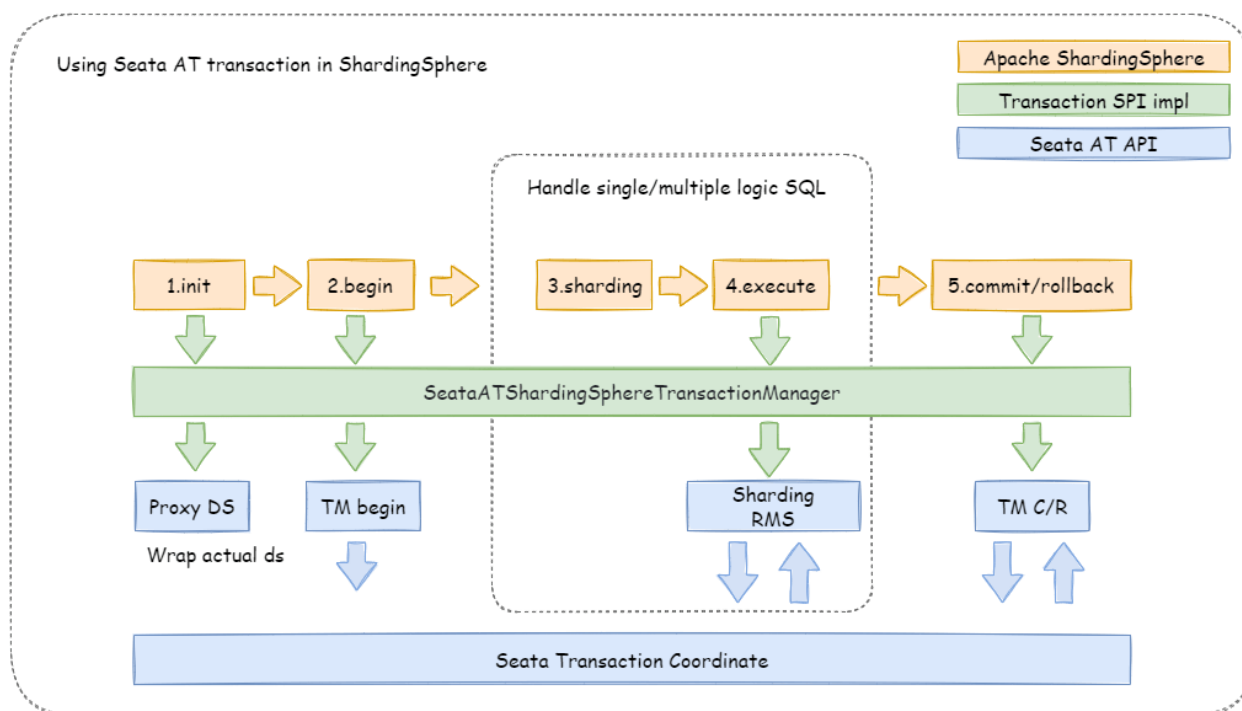


Fig. 7: Seata 柔性事务

引擎初始化

包含 Seata 柔性事务的应用启动时，用户配置的数据源会根据 seata.conf 的配置，适配为 Seata 事务所需的 DataSource-Proxy，并且注册至 RM 中。

开启全局事务

TM 控制全局事务的边界，TM 通过向 TC 发送 Begin 指令，获取全局事务 ID，所有分支事务通过此全局事务 ID，参与到全局事务中；全局事务 ID 的上下文存放在当前线程变量中。

执行真实分片 SQL

处于 Seata 全局事务中的分片 SQL 通过 RM 生成 undo 快照，并且发送 participate 指令至 TC，加入到全局事务中。由于 DBPlusEngine 的分片物理 SQL 采取多线程方式执行，因此整合 Seata AT 事务时，需要在主线程和子线程间进行全局事务 ID 的上下文传递。

提交或回滚事务

提交 Seata 事务时，TM 会向 TC 发送全局事务的提交或回滚指令，TC 根据全局事务 ID 协调所有分支事务进行提交或回滚。

2.4 可观察性

2.4.1 背景

如何观测集群的运行状态，使运维人员可以快速掌握当前系统现状，并进行进一步的维护工作，是分布式系统的全新挑战。登录到具体服务器的点对点运维方式，无法适用于面向大量分布式服务器的场景。通过对可系统观察性数据的遥测是分布式系统推荐的运维方式。Tracing（链路跟踪）、Metrics（指标监控）和 Logging（日志）是系统运行状况的可观察性数据重要的获取手段。

2.4.2 挑战

Tracing 和 Metrics 需要通过埋点来收集系统信息。大量的埋点使项目核心代码支离破碎，难于维护，且不易定制化统计指标。

2.4.3 核心概念

代理

基于字节码增强和插件化设计，以提供 tracing 和 metrics 埋点，以及日志输出功能。需要开启代理的插件功能后，才能将监控指标数据输出至第三方 APM 中展示。

APM

APM 是应用性能监控的缩写。着眼于分布式系统的性能诊断，其主要功能包括调用链展示，应用拓扑分析等。

Tracing

链路跟踪，通过探针收集调用链数据，并发送到第三方 APM 系统

Metrics

系统统计指标，通过探针收集，并且写入到时序数据库，供第三方应用展示。

2.4.4 系统实现

Agent 模块为 SphereEx-DBPlusEngine 提供了可观察性的框架，它是基于 Java Agent 技术实现的。Metrics、Tracing 和 Logging 等功能均通过插件的方式集成在 Agent 中。- Metrics 插件用于收集和展示整个集群的统计指标。SphereEx-DBPlusEngine 默认提供了对 Prometheus 的支持。- Tracing 插件用于获取 SQL 解析与 SQL 执行的链路跟踪信息。SphereEx-DBPlusEngine 默认提供了对 Jaeger、OpenTelemetry、OpenTracing (SkyWalking) 和 Zipkin 的支持，也支持用户通过插件化的方式开发自定义的 Tracing 组件。- 默认的 Logging 插件展示了如何在 SphereEx-DBPlusEngine 中记录额外的日志，实际应用中需要用户根据自己的需求进行探索。

APM

APM（应用性能监控）是通过对系统可观察性数据进行采集、存储和分析，进行系统的性能监控与诊断，主要功能包括性能指标监控、调用链分析，应用拓扑图等。

Metrics

DBPlusEngine 并不负责如何采集、存储以及展示应用性能监控的相关数据，而是为应用监控系统提供必要的指标数据。换句话说，DBPlusEngine 仅负责产生具有价值的数 据，并通过标准协议或插件化的方式递交给相关系统。

Tracing

Tracing 用于获取 SQL 解析与 SQL 执行的链路跟踪信息。DBPlusEngine 默认提供了对 SkyWalking, Zipkin, Jaeger 和 OpenTelemetry 的支持，也支持用户通过插件化的方式开发自定义的 Tracing 组件。

- 使用 Zipkin 和 Jaeger

通过在 agent 配置文件中开启对应的插件，并配置好 Zipkin 或者 Jaeger 服务器信息即可。

- 使用 OpenTelemetry

OpenTelemetry 在 2019 年由 OpenTracing 和 OpenCensus 合并而来。使用这种方式，只需要在 agent 配置文件中，根据 OpenTelemetry SDK 自动配置说明，填写合适的配置即可。

- 使用 SkyWalking

需要在 agent 配置中配置启用对应插件，并且需要同时配置使用 SkyWalking 的 apm-toolkit 工具。

- 使用 SkyWalking 的内置自动探针

DBPlusEngine 团队与 Apache SkyWalking 团队共同合作，在 SkyWalking 中实现了 DBPlusEngine 自动探针，可以将相关的应用性能数据自动发送到 SkyWalking 中。注意这种方式的自动探针不能与 DBPlusEngine 插件探针同时使用。Metrics 则用于收集和展示整个集群的统计指标。DBPlusEngine 默认提供了对 Prometheus 的支持。

2.5 弹性伸缩

2.5.1 原理说明

考虑到 DBPlusEngine 的弹性伸缩模块的几个挑战，目前的弹性伸缩解决方案为：临时地使用两个数据库集群，伸缩完成后切换的方式实现。

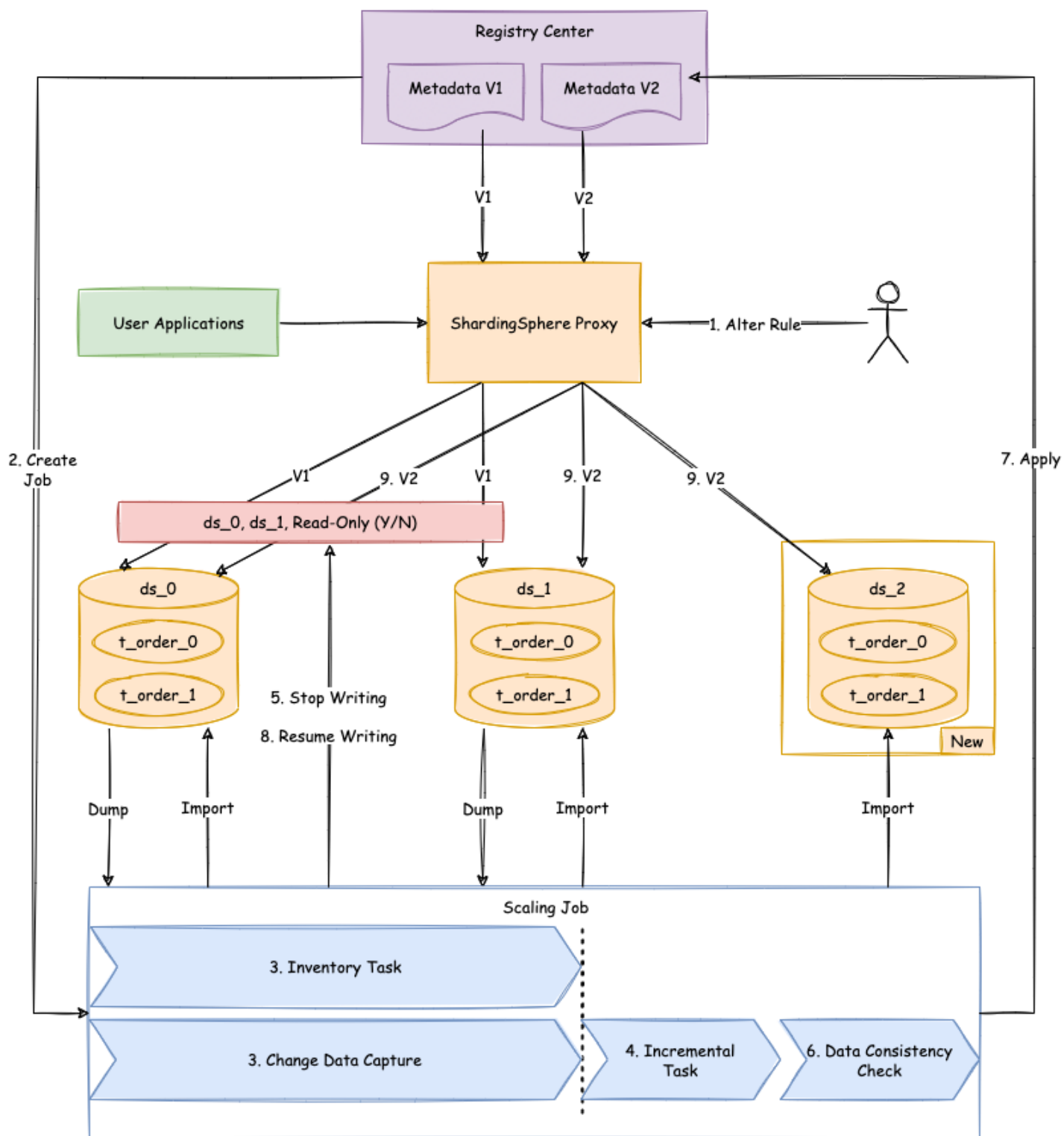


Fig. 8: 伸缩总揽

这种实现方式有以下优点：

1. 伸缩过程中，原始数据没有任何影响
2. 伸缩失败无风险
3. 不受分片策略限制

同时也存在一定的缺点：

1. 在一定时间内存在冗余服务器
2. 所有数据都需要移动

弹性伸缩模块会通过解析旧分片规则，提取配置中的数据源、数据节点等信息，之后创建伸缩作业工作流，将一次弹性伸缩拆解为 4 个主要阶段

1. 准备阶段
2. 存量数据迁移阶段
3. 增量数据同步阶段
4. 规则切换阶段

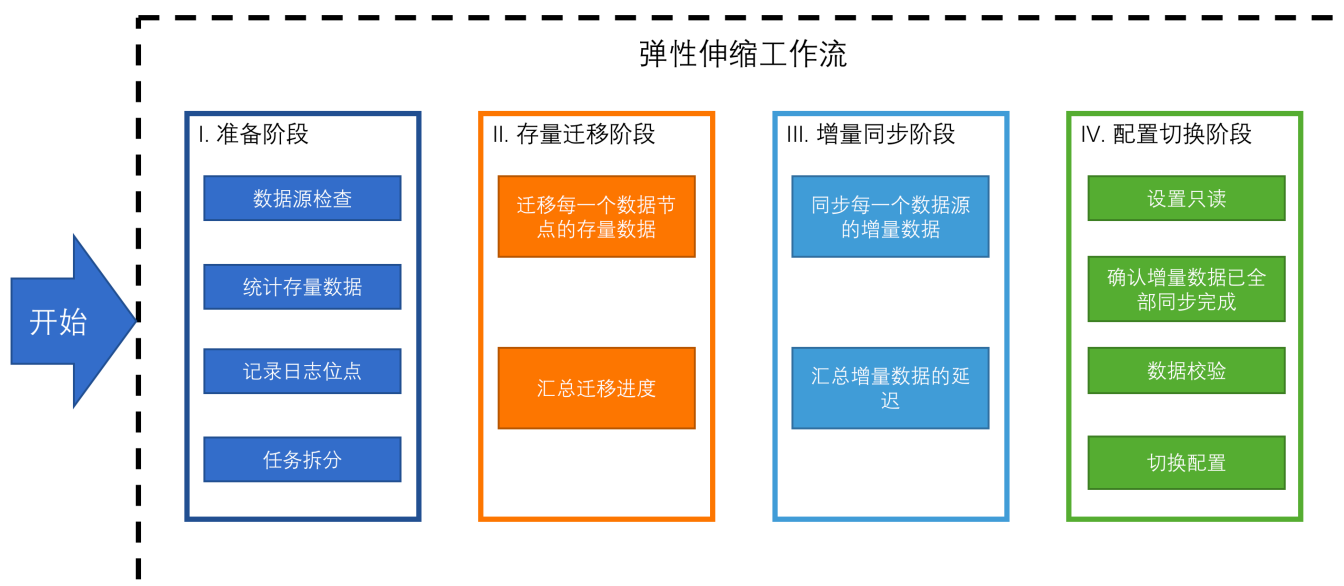


Fig. 9: 伸缩工作流

2.5.2 执行阶段说明

准备阶段

在准备阶段，弹性伸缩模块会进行数据源连通性及权限的校验，同时进行存量数据的统计、日志位点的记录，最后根据数据量和用户设置的并行度，对任务进行分片。

存量数据迁移阶段

执行在准备阶段拆分好的存量数据迁移作业，存量迁移阶段采用 JDBC 查询的方式，直接从数据节点中读取数据，并使用新规则写入到新集群中。

增量数据同步阶段

由于存量数据迁移耗费的时间受到数据量和并行度等因素影响，此时需要对这段时间内业务新增的数据进行同步。不同的数据库使用的技术细节不同，但总体上均为基于复制协议或 WAL 日志实现的变更数据捕获功能。

- MySQL：订阅并解析 binlog
- PostgreSQL：采用官方逻辑复制 `test_decoding`

这些捕获的增量数据，同样会由弹性伸缩模块根据新规则写入到新数据节点中。当增量数据基本同步完成时（由于业务系统未停止，增量数据是不断的），则进入规则切换阶段。

规则切换阶段

在此阶段，可能存在一定时间的业务只读窗口期，通过设置数据库只读或 DBPlusEngine 的熔断机制，让旧数据节点中的数据短暂静态，确保增量同步已完全完成。

这个窗口期时间短则数秒，长则数分钟，取决于数据量和用户是否需要对数据进行强校验。确认完成后，DBPlusEngine 可通过配置中心修改配置，将业务导向新规则的集群，弹性伸缩完成。

2.6 数据分片

SphereEx-DBPlusEngine 数据分片优化流程，按照是否进行查询优化，可以分为 Standard 内核流程和 Federation 执行引擎流程。- Standard 内核流程由 SQL 解析 => SQL 路由 => SQL 改写 => SQL 执行 => 结果归并组成，主要用于处理标准分片场景下的 SQL 执行。- Federation 执行引擎流程由 SQL 解析 => 逻辑优化 => 物理优化 => 优化执行 => Standard 内核流程组成，Federation 执行引擎内部进行逻辑优化和物理优化，在优化执行阶段依赖 Standard 内核流程，对优化后的逻辑 SQL 进行路由、改写、执行和归并。

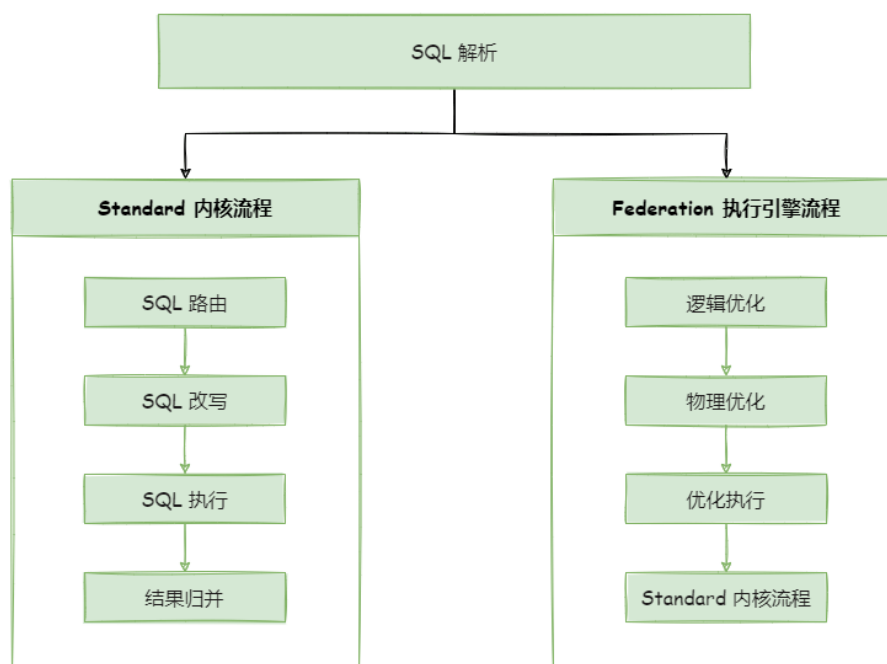


Fig. 10: 数据分片主要流程图

2.6.1 SQL 解析

分为词法解析和语法解析。先通过词法解析器将 SQL 拆分为一个个不可再分的单词。再使用语法解析器对 SQL 进行理解，并最终提炼出解析上下文。解析上下文包括表、选择项、排序项、分组项、聚合函数、分页信息、查询条件以及可能需要修改的占位符的标记。

2.6.2 SQL 路由

根据解析上下文匹配用户配置的分片策略，并生成路由路径。目前支持分片路由和广播路由。

2.6.3 SQL 改写

将 SQL 改写为在真实数据库中 can 正确执行的语句。SQL 改写分为正确性改写和优化改写。

2.6.4 SQL 执行

通过多线程执行器异步执行。

2.6.5 结果归并

将多个执行结果集归并以便于通过统一的 JDBC 接口输出。结果归并包括流式归并、内存归并和使用装饰者模式的追加归并这几种方式。

2.6.6 查询优化

由 Federation 执行引擎（开发中）提供支持，对关联查询、子查询等复杂查询进行优化，同时支持跨多个数据库实例的分布式查询，内部使用关系代数优化查询计划，通过最优计划查询出结果。

2.6.7 解析引擎

相对于其他编程语言，SQL 是比较简单的。不过，它依然是一门完善的编程语言，因此对 SQL 的语法进行解析，与解析其他编程语言（如：Java 语言、C 语言、Go 语言等）并无本质区别。

抽象语法树

解析过程分为词法解析和语法解析。词法解析器用于将 SQL 拆解为不可再分的原子符号，称为 Token。并根据不同数据库方言所提供的字典，将其归类为关键字，表达式，字面量和操作符。再使用语法解析器将词法解析器的输出转换为抽象语法树。

例如，以下 SQL：

```
sql> SELECT id, name FROM t_user WHERE status = 'ACTIVE' AND age > 18
```

解析之后的为抽象语法树见下图。

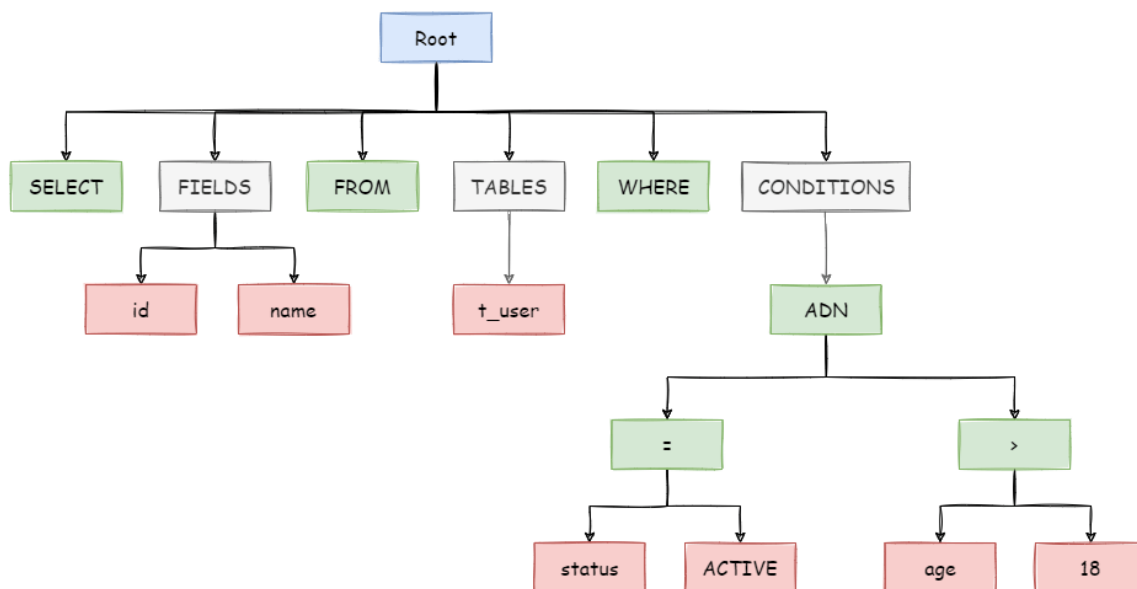


Fig. 11: 抽象语法树

为了便于理解，抽象语法树中的关键字的 Token 用绿色表示，变量的 Token 用红色表示，灰色表示需要进一步拆分。最后，通过 visitor 对抽象语法树遍历构造域模型，通过域模型（SQLStatement）去提炼分片所需的上下文，并标记有可能需要改写的位置。供分片使用的解析上下文包含查询选择项（Select Items）、表信息（Table）、分片条件（ShardingCondition）、自增主键信息（Auto increment Primary Key）、排序信息（Order By）、分组信息（Group By）以及分页信息（Limit、Rownum、Top）。SQL 的一次解析过程是不可逆的，一个个 Token 按 SQL 原本的顺序依次进行解析，性能很高。考虑到各种数据库 SQL 方言的异同，在解析模块提供了各类数据库的 SQL 方言字典。

SQL 解析引擎

■ 功能点

- 提供独立的 SQL 解析功能
- 可以非常方便的对语法规则进行扩充和修改（使用了 ANTLR）
- 支持多种方言的 SQL 解析

数据库	支持状态
MySQL	支持，完善
PostgreSQL	支持，完善
SQLServer	支持
Oracle	支持
SQL92	支持
openGauss	支持

■ 开发中功能

- 提供 SQL 格式化功能
- 提供 SQL 模板化功能

2.6.8 路由引擎

根据解析上下文匹配数据库和表的分片策略，并生成路由路径。对于携带分片键的 SQL，根据分片键的不同可以分为单片路由（分片键的操作符是等号）、多片路由（分片键的操作符是 IN）和范围路由（分片键的操作符是 BETWEEN）。不携带分片键的 SQL 则采用广播路由。

分片策略通常可以采用由数据库内置或由用户方配置。数据库内置的方案较为简单，内置的分片策略大致可分为尾数取模、哈希、范围、标签、时间等。由用户方配置的分片策略则更加灵活，可以根据使用方需求定制复合分片策略。如果配合数据自动迁移来使用，可以做到无需用户关注分片策略，自动由数据库中间层分片和平衡数据即可，进而做到使分布式数据库具有的弹性伸缩的能力。

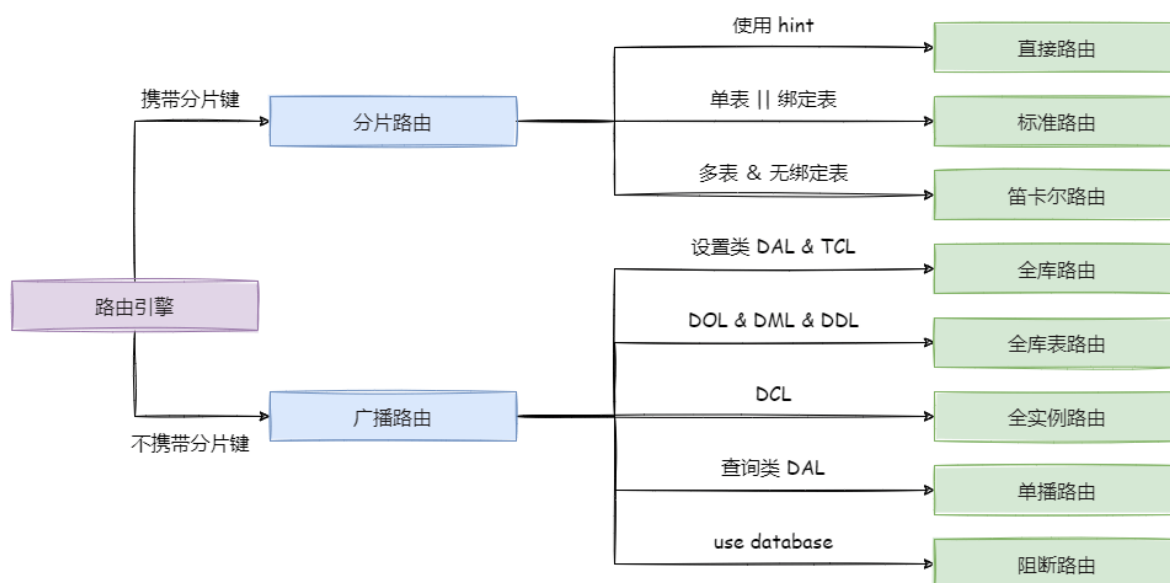


Fig. 12: 路由引擎原理图

分片路由

用于根据分片键进行路由的场景，又细分为直接路由、标准路由和笛卡尔积路由这 3 种类型。

直接路由

满足直接路由的条件相对苛刻，它需要通过 Hint（使用 HintAPI 直接指定路由至库表）方式分片，并且是只分库不分表的前提下，则可以避免 SQL 解析和之后的结果归并。因此它的兼容性最好，可以执行包括子查询、自定义函数等复杂情况的任意 SQL。直接路由还可以用于分片键不在 SQL 中的场景。

标准路由

标准路由是 DBPlusEngine 最为推荐使用的分片方式，它的适用范围是不包含关联查询或仅包含绑定表之间关联查询的 SQL。当分片运算符是等于号时，路由结果将落入单库（表），当分片运算符是 BETWEEN 或 IN 时，则路由结果不一定落入唯一的库（表），因此一条逻辑 SQL 最终可能被拆分为多条用于执行的真实 SQL。举例说明，如果按照 order_id 的奇数和偶数进行数据分片，一个单表查询的 SQL 如下：

```
sql> SELECT * FROM t_order WHERE order_id IN (1, 2);
```

那么路由的结果应为：

```
sql> SELECT * FROM t_order_0 WHERE order_id IN (1, 2);  
sql> SELECT * FROM t_order_1 WHERE order_id IN (1, 2);
```

绑定表的关联查询与单表查询复杂度和性能相当。举例说明，如果一个包含绑定表的关联查询的 SQL 如下：

```
sql> SELECT * FROM t_order o JOIN t_order_item i ON o.order_id=i.order_id WHERE order_id IN (1, 2);
```

那么路由的结果应为：

```
sql> SELECT * FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE order_id IN (1, 2);  
sql> SELECT * FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE order_id IN (1, 2);
```

可以看到，SQL 拆分的数目与单表是一致的。

笛卡尔路由

笛卡尔路由是最复杂的情况，它无法根据绑定表的关系定位分片规则，因此非绑定表之间的关联查询需要拆解为笛卡尔积组合执行。如果上个示例中的 SQL 并未配置绑定表关系，那么路由的结果应为：

```
sql> SELECT * FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE order_id IN (1, 2);  
sql> SELECT * FROM t_order_0 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE order_id IN (1, 2);  
sql> SELECT * FROM t_order_1 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE order_id IN (1, 2);  
sql> SELECT * FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE order_id IN (1, 2);
```

笛卡尔路由查询性能较低，需谨慎使用。

广播路由

对于不携带分片键的 SQL，则采取广播路由的方式。根据 SQL 类型又可以划分为全库表路由、全库路由、全实例路由、单播路由和阻断路由这 5 种类型。

全库表路由

全库表路由用于处理对数据库中与其逻辑表相关的所有真实表的操作，主要包括不带分片键的 DQL 和 DML，以及 DDL 等。

例如：

```
sql> SELECT * FROM t_order WHERE good_prority IN (1, 10);
```

则会遍历所有数据库中的所有表，逐一匹配逻辑表和真实表名，能够匹配得上则执行。路由后成为

```
sql> SELECT * FROM t_order_0 WHERE good_prority IN (1, 10);
sql> SELECT * FROM t_order_1 WHERE good_prority IN (1, 10);
sql> SELECT * FROM t_order_2 WHERE good_prority IN (1, 10);
sql> SELECT * FROM t_order_3 WHERE good_prority IN (1, 10);
```

全库路由

全库路由用于处理对数据库的操作，包括用于库设置的 SET 类型的数据库管理命令，以及 TCL 这样的事务控制语句。在这种情况下，会根据逻辑库的名字遍历所有符合名字匹配的真实库，并在真实库中执行该命令，例如：

```
sql> SET autocommit=0;
```

在 t_order 中执行，t_order 有 2 个真实库。则实际会在 t_order_0 和 t_order_1 上都执行这个命令。

全实例路由

全实例路由用于 DCL 操作，授权语句针对的是数据库的实例。无论一个实例中包含多少个 Schema，每个数据库的实例只执行一次。例如：

```
sql> CREATE USER customer@127.0.0.1 identified BY '123';
```

这个命令将在所有的真实数据库实例中执行，以确保 customer 用户可以访问每一个实例。

单播路由

单播路由用于获取某一真实表信息的场景，它仅需要从任意库中的任意真实表中获取数据即可。例如：

```
sql> DESCRIBE t_order;
```

t_order 的两个真实表 t_order_0，t_order_1 的描述结构相同，所以这个命令在任意真实表上选择执行一次。

阻断路由

阻断路由用于屏蔽 SQL 对数据库的操作，例如：

```
sql> USE order_db;
```

这个命令不会在真实数据库中执行，因为 DBPlusEngine 采用的是逻辑 Schema 的方式，无需将切换数据库 Schema 的命令发送至数据库中。

2.6.9 改写引擎

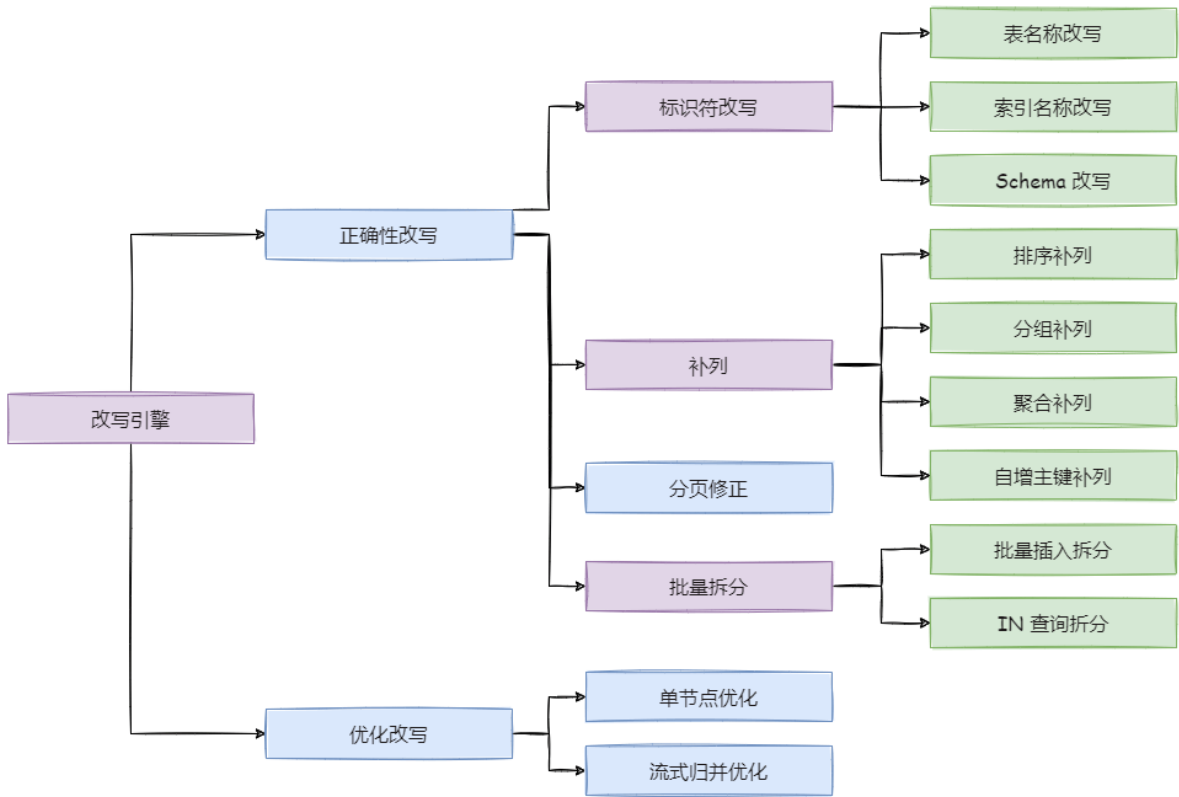


Fig. 13: 改写引擎原理图

工程师面向逻辑库与逻辑表书写的 SQL，并不能够直接在真实的数据库中执行，SQL 改写用于将逻辑 SQL 改写为在真实数据库中正确执行的 SQL。它包括正确性改写和优化改写两部分。

正确性改写

在包含分表的场景中，需要将分表配置中的逻辑表名称改写为路由之后所获取的真实表名称。仅分库则不需要表名称的改写。除此之外，还包括补列和分页信息修正等内容。

标识符改写

需要改写的标识符包括表名称、索引名称以及 Schema 名称。

表名称改写是指将找到逻辑表在原始 SQL 中的位置，并将其改写为真实表的过程。表名称改写是一个典型的需要对 SQL 进行解析的场景。从一个最简单的例子开始，若逻辑 SQL 为：

```
sql> SELECT order_id FROM t_order WHERE order_id=1;
```

假设该 SQL 配置分片键 order_id，并且 order_id=1 的情况，将路由至分片表 1。那么改写之后的 SQL 应该为：

```
sql> SELECT order_id FROM t_order_1 WHERE order_id=1;
```

在这种最简单的 SQL 场景中，是否将 SQL 解析为抽象语法树似乎无关紧要，只要通过字符串查找和替换就可以达到 SQL 改写的效果。但是下面的场景，就无法仅仅通过字符串的查找替换来正确的改写 SQL 了：

```
sql> SELECT order_id FROM t_order WHERE order_id=1 AND remarks='t_order xxx';
```

正确改写的 SQL 应该是：

```
sql> SELECT order_id FROM t_order_1 WHERE order_id=1 AND remarks='t_order xxx';
```

而非：

```
sql> SELECT order_id FROM t_order_1 WHERE order_id=1 AND remarks='t_order_1 xxx';
```

由于表名之外可能含有表名称的类似字符，因此不能通过简单的字符串替换的方式去改写 SQL。下面再来看一个更加复杂的 SQL 改写场景：

```
sql> SELECT t_order.order_id FROM t_order WHERE t_order.order_id=1 AND remarks='t_order xxx';
```

上面的 SQL 将表名作为字段的标识符，因此在 SQL 改写时需要一并修改：

```
sql> SELECT t_order_1.order_id FROM t_order_1 WHERE t_order_1.order_id=1 AND remarks='t_order xxx';
```

而如果 SQL 中定义了表的别名，则无需连同别名一起修改，即使别名与表名相同亦是如此。例如：

```
sql> SELECT t_order.order_id FROM t_order AS t_order WHERE t_order.order_id=1 AND remarks='t_order xxx';
```

SQL 改写则仅需要改写表名称就可以了：

```
sql> SELECT t_order.order_id FROM t_order_1 AS t_order WHERE t_order.order_id=1 AND remarks='t_order xxx';
```

索引名称是另一个有可能改写的标识符。在某些数据库中（如 MySQL、SQLServer），索引是以表为维度创建的，在不同的表中的索引是可以重名的；而在另外的一些数据库中（如 PostgreSQL、Oracle），索引是以数据库为维度创建的，即使是作用在不同表上的索引，它们也要求其名称的唯一性。

在 SphereEx-DBPlusEngine 中，管理 Schema 的方式与管理表如出一辙，它采用逻辑 Schema 去管理一组数据源。因此，DBPlusEngine 需要将用户在 SQL 中书写的逻辑 Schema 替换为真实的数据库 Schema。DBPlusEngine 目前还不支持在 DQL 和 DML 语句中使用 Schema。它目前仅支持在数据库管理语句中使用 Schema，例如：

```
sql> SHOW COLUMNS FROM t_order FROM order_ds;
```

Schema 的改写指的是将逻辑 Schema 采用单播路由的方式，改写为随机查找到的一个正确的真实 Schema。

补列

需要在查询语句中补列通常由两种情况导致。第一种情况是 SphereEx-DBPlusEngine 需要在结果归并时获取相应数据，但该数据并未通过查询的 SQL 返回。这种情况主要是针对 GROUP BY 和 ORDER BY。结果归并时，需要根据 GROUP BY 和 ORDER BY 的字段项进行分组和排序，但如果原始 SQL 的选择项中若并未包含分组项或排序项，则需要对原始 SQL 进行改写。先看一下原始 SQL 中带有结果归并所需信息的场景：

```
sql> SELECT order_id, user_id FROM t_order ORDER BY user_id;
```

由于使用 user_id 进行排序，在结果归并中需要能够获取到 user_id 的数据，而上面的 SQL 是能够获取到 user_id 数据的，因此无需补列。

如果选择项中不包含结果归并时所需的列，则需要进行补列，如以下 SQL：

```
sql> SELECT order_id FROM t_order ORDER BY user_id;
```

由于原始 SQL 中并不包含需要在结果归并中需要获取的 user_id，因此需要对 SQL 进行补列改写。补列之后的 SQL 是：

```
sql> SELECT order_id, user_id AS ORDER_BY_DERIVED_0 FROM t_order ORDER BY user_id;
```

值得一提的是，补列只会补充缺失的列，不会全部补充，而且，在 SELECT 语句中包含 * 的 SQL，也会根据表的元数据信息选择性补列。

下面是一个较为复杂的 SQL 补列场景：

```
sql> SELECT o.* FROM t_order o, t_order_item i WHERE o.order_id=i.order_id ORDER BY user_id, order_item_id;
```

我们假设只有 t_order_item 表中包含 order_item_id 列，那么根据表的元数据信息可知，在结果归并时，排序项中的 user_id 是存在于 t_order 表中的，无需补列；order_item_id 并不在 t_order 中，因此需要补列。补列之后的 SQL 是：

```
sql> SELECT o.*, order_item_id AS ORDER_BY_DERIVED_0 FROM t_order o, t_order_item i WHERE o.order_id=i.order_id ORDER BY user_id, order_item_id;
```

补列的另一种情况是使用 AVG 聚合函数。在分布式的场景中，使用 avg1 + avg2 + avg3 / 3 计算平均值并不正确，需要改写为 (sum1 + sum2 + sum3) / (count1 + count2 + count3)。这就需要包含 AVG 的 SQL 改写为 SUM 和 COUNT，并在结果归并时重新计算平均值。例如以下 SQL：

```
sql> SELECT AVG(price) FROM t_order WHERE user_id=1;
```

需要改写为：

```
sql> SELECT COUNT(price) AS AVG_DERIVED_COUNT_0, SUM(price) AS AVG_DERIVED_SUM_0 FROM t_order WHERE user_id=1;
```

然后才能够通过结果归并正确的计算平均值。

最后一种补列是在执行 INSERT 的 SQL 语句时，如果使用数据库自增主键，是无需写入主键字段的。但数据库的自增主键是无法满足分布式场景下的主键唯一的，因此 SphereEx-DBPlusEngine 提供了分布式自增主键的生成策略，并且可以通过补列，让使用方无需改动现有代码，即可将分布式自增主键透明的替换数据库现有的自增主键。分布式自增主键的生成策略将在下文中详述，这里只阐述与 SQL 改写相关的内容。举例说明，假设表 t_order 的主键是 order_id，原始的 SQL 为：

```
sql> INSERT INTO t_order (`field1`, `field2`) VALUES (10, 1);
```

可以看到，上述 SQL 中并未包含自增主键，是需要数据库自行填充的。SphereEx-DBPlusEngine 配置自增主键后，SQL 将改写为：

```
sql> INSERT INTO t_order (`field1`, `field2`, order_id) VALUES (10, 1, xxxxx);
```

改写后的 SQL 将在 INSERT FIELD 和 INSERT VALUE 的最后部分增加主键列名称以及自动生成的自增主键值。上述 SQL 中的 xxxxx 表示自动生成的自增主键值。如果 INSERT 的 SQL 中并未包含表的列名称，DBPlusEngine 也可以根据判断参数个数以及表元信息中的列数量对比，并自动生成自增主键。例如，原始的 SQL 为：

```
sql> INSERT INTO t_order VALUES (10, 1);
```

改写的 SQL 将只在主键所在的列顺序处增加自增主键即可：

```
sql> INSERT INTO t_order VALUES (xxxxx, 10, 1);
```

自增主键补列时，如果使用占位符的方式书写 SQL，则只需要改写参数列表即可，无需改写 SQL 本身。

分页修正

从多个数据库获取分页数据与单数据库的场景是不同的。假设每 10 条数据为一页，取第 2 页数据。在分片环境下获取 LIMIT 10, 10，归并之后再根据排序条件取出前 10 条数据是不正确的。举例说明，若 SQL 为：

```
sql> SELECT score FROM t_score ORDER BY score DESC LIMIT 1, 2;
```

下图展示了不进行 SQL 的改写的分页执行结果。

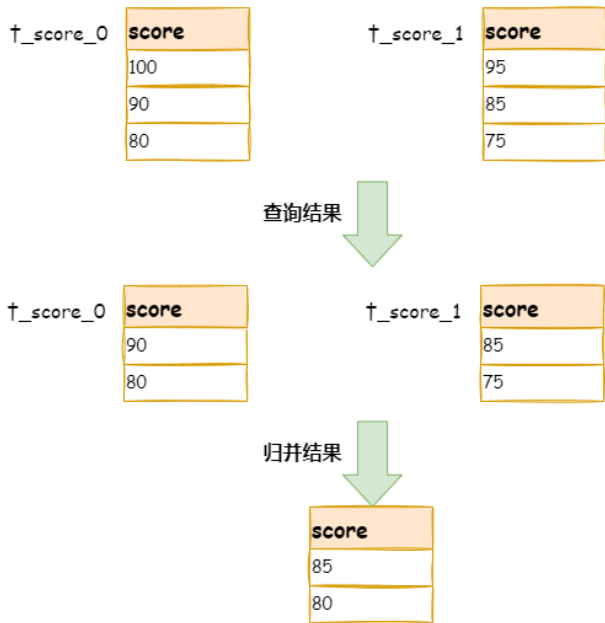


Fig. 14: 分页 1

通过图中所示，想要取得两个表中共同的按照分数排序的第 2 条和第 3 条数据，应该是 95 和 90。由于执行的 SQL 只能从每个表中获取第 2 条和第 3 条数据，即从 `t_score_0` 表中获取的是 90 和 80；从 `t_score_1` 表中获取的是 85 和 75。因此进行结果归并时，只能从获取的 90，80，85 和 75 之中进行归并，那么结果归并无论怎么实现，都不可能获得正确的结果。

正确的做法是将分页条件改写为 `LIMIT 0, 3`，取出所有前两页数据，再结合排序条件计算出正确的数据。下图展示了进行 SQL 改写之后的分页执行结果。

SELECT score FROM t_score ORDER BY score DESC LIMIT 0 , 3

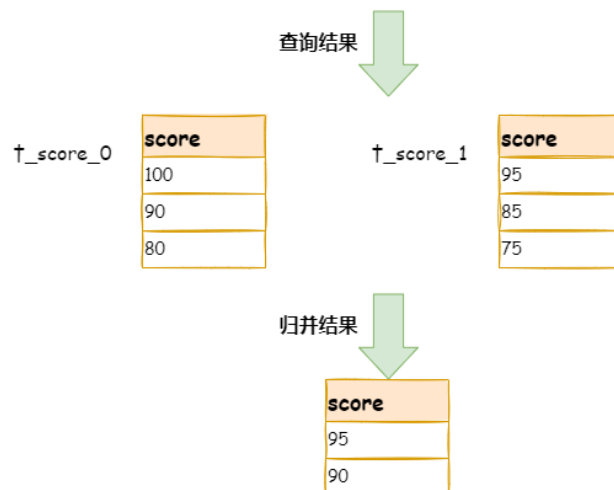


Fig. 15: 分页 2

越获取偏移量位置靠后数据，使用 LIMIT 分页方式的效率就越低。有很多方法可以避免使用 LIMIT 进行分页。比如构建行记录数量与行偏移量的二级索引，或使用上次分页数据结尾 ID 作为下次查询条件的分页方式等。分页信息修正时，如果使用占位符的方式书写 SQL，则只需要改写参数列表即可，无需改写 SQL 本身。

批量拆分

在使用批量插入的 SQL 时，如果插入的数据是跨分片的，那么需要对 SQL 进行改写来防止将多余的数据写入到数据库中。插入操作与查询操作的不同之处在于，查询语句中即使用了不存在于当前分片的分片键，也不会对数据产生影响；而插入操作则必须将多余的分片键删除。举例说明，如下 SQL：

```
sql> INSERT INTO t_order (order_id, xxx) VALUES (1, 'xxx'), (2, 'xxx'), (3, 'xxx');
```

假设数据库仍然是按照 order_id 的奇偶值分为两片的，仅将这条 SQL 中的表名进行修改，然后发送至数据库完成 SQL 的执行，则两个分片都会写入相同的记录。虽然只有符合分片查询条件的数据才能够被查询语句取出，但存在冗余数据的实现方案并不合理。因此需要将 SQL 改写为：

```
sql> INSERT INTO t_order_0 (order_id, xxx) VALUES (2, 'xxx');  
sql> INSERT INTO t_order_1 (order_id, xxx) VALUES (1, 'xxx'), (3, 'xxx');
```

使用 IN 的查询与批量插入的情况相似，不过 IN 操作并不会导致数据查询结果错误。通过对 IN 查询的改写，可以进一步的提升查询性能。如以下 SQL：


```
sql> SELECT * FROM t_order WHERE order_id IN (1, 2, 3);
```

改写为：

```
sql> SELECT * FROM t_order_0 WHERE order_id IN (2);  
sql> SELECT * FROM t_order_1 WHERE order_id IN (1, 3);
```

可以进一步的提升查询性能。DBPlusEngine 暂时还未实现此改写策略，目前的改写结果是：

```
sql> SELECT * FROM t_order_0 WHERE order_id IN (1, 2, 3);  
sql> SELECT * FROM t_order_1 WHERE order_id IN (1, 2, 3);
```

虽然 SQL 的执行结果是正确的，但并未达到最优的查询效率。

优化改写

优化改写的目的是在不影响查询正确性的情况下，对性能进行提升的有效手段。它分为单节点优化和流式归并优化。

单节点优化

路由至单节点的 SQL，则无需优化改写。当获得一次查询的路由结果后，如果是路由至唯一的数据节点，则无需涉及到结果归并。因此补列和分页信息等改写都没有必要进行。尤其是分页信息的改写，无需将数据从第 1 条开始取，大量的降低了对数据库的压力，并且节省了网络带宽的无谓消耗。

流式归并优化

它仅为包含 GROUP BY 的 SQL 增加 ORDER BY 以及和分组项相同的排序项和排序顺序，用于将内存归并转化为流式归并。在结果归并的部分中，将对流式归并和内存归并进行详细说明。

2.6.10 执行引擎

SphereEx-DBPlusEngine 采用一套自动化的执行引擎，负责将路由和改写完成之后的真实 SQL 安全且高效发送到底层数据源执行。它不是简单地将 SQL 通过 Driver 直接发送至数据源执行；也并非直接将执行请求放入线程池去并发执行。它更关注平衡数据源连接创建以及内存占用所产生的消耗，以及最大限度地合理利用并发等问题。执行引擎的目标是自动化的平衡资源控制与执行效率。

连接模式

从资源控制的角度看，业务方访问数据库的连接数量应当有所限制。它能够有效地防止某一业务操作过多的占用资源，从而将数据库连接的资源耗尽，以致于影响其他业务的正常访问。特别是在一个数据库实例中存在较多分表的情况下，一条不包含分片键的逻辑 SQL 将产生落在同库不同表的大量真实 SQL，如果每条真实 SQL 都占用一个独立的连接，那么一次查询无疑将会占用过多的资源。

从执行效率的角度看，为每个分片查询维持一个独立的数据库连接，可以更加有效的利用多线程来提升执行效率。为每个数据库连接开启独立的线程，可以将 I/O 所产生的消耗并行处理。为每个分片维持一个独立的数据库连接，还能够避免过早的将查询结果数据加载至内存。独立的数据库连接，能够持有查询结果集游标位置的引用，在需要获取相应数据时移动游标即可。

以结果集游标下移进行结果归并的方式，称之为流式归并，它无需将结果数据全数加载至内存，可以有效的节省内存资源，进而减少垃圾回收的频次。当无法保证每个分片查询持有一个独立数据库连接时，则需要在复用该数据库连接获取下一张分表的查询结果集之前，将当前的查询结果集全数加载至内存。因此，即使可以采用流式归并，在此场景下也将退化为内存归并。

一方面是对数据库连接资源的控制保护，一方面是采用更优的归并模式达到对中间件内存资源的节省，如何处理好两者之间的关系，是 DBPlusEngine 执行引擎需要解决的问题。具体来说，如果一条 SQL 在经过 DBPlusEngine 的分片后，需要操作

某数据库实例下的 200 张表。那么，是选择创建 200 个连接并行执行，还是选择创建一个连接串行执行呢？效率与资源控制又应该如何抉择呢？针对上述场景，DBPlusEngine 提供了一种解决思路。它提出了连接模式（Connection Mode）的概念，将其划分为内存限制模式（MEMORY_STRICTLY）和连接限制模式（CONNECTION_STRICTLY）这两种类型。

内存限制模式

使用此模式的前提是，SphereEx-DBPlusEngine 对一次操作所耗费的数据库连接数量不做限制。如果实际执行的 SQL 需要对某数据库实例中的 200 张表做操作，则对每张表创建一个新的数据库连接，并通过多线程的方式并发处理，以达成执行效率最大化。并且在 SQL 满足条件情况下，优先选择流式归并，以防止出现内存溢出或避免频繁垃圾回收情况。

连接限制模式

使用此模式的前提是，SphereEx-DBPlusEngine 严格控制对一次操作所耗费的数据库连接数量。如果实际执行的 SQL 需要对某数据库实例中的 200 张表做操作，那么只会创建唯一的数据库连接，并对其 200 张表串行处理。如果一次操作中的分片散落在不同的数据库，仍然采用多线程处理对不同库的操作，但每个库的每次操作仍然只创建一个唯一的数据库连接。这样就可以防止对一次请求对数据库连接占用过多所带来的问题。该模式始终选择内存归并。

注意：内存限制模式适用于 OLAP 操作，可以通过放宽对数据库连接的限制提升系统吞吐量；连接限制模式适用于 OLTP 操作，OLTP 通常带有分片键，会路由到单一的分片，因此严格控制数据库连接，以保证在线系统数据库资源能够被更多的应用所使用，是明智的选择。

自动化执行引擎

SphereEx-DBPlusEngine 最初将使用何种模式的决定权交由用户配置，让开发者依据自己业务的实际场景需求选择使用内存限制模式或连接限制模式。这种解决方案将两难的选择的决定权交由用户，使得用户必须要了解这两种模式的利弊，并依据业务场景需求进行选择。这无疑增加了用户对 SphereEx-DBPlusEngine 的学习和使用的成本，并非最优方案。这种一分为二的处理方案，将两种模式的切换交由静态的初始化配置，是缺乏灵活应对能力的。在实际的使用场景中，面对不同 SQL 以及占位符参数，每次的路由结果是不同的。这就意味着某些操作可能需要使用内存归并，而某些操作则可能选择流式归并更优，具体采用哪种方式不应该由用户在 SphereEx-DBPlusEngine 启动之前配置好，而是应该根据 SQL 和占位符参数的场景，来动态的决定连接模式。为了降低用户的使用成本以及连接模式动态化这两个问题，SphereEx-DBPlusEngine 提炼出自动化执行引擎的思路，在其内部消化了连接模式概念。用户无需了解所谓的内存限制模式和连接限制模式是什么，而是交由执行引擎根据当前场景自动选择最优的执行方案。自动化执行引擎将连接模式的选择粒度细化至每一次 SQL 的操作。针对每次 SQL 请求，自动化执行引擎都将根据其路由结果，进行实时的演算和权衡，并自主地采用恰当的连接模式执行，以达到资源控制和效率的最优平衡。针对自动化的执行引擎，用户只需配置 maxConnectionSizePerQuery 即可，该参数表示一次查询时每个数据库所允许使用的最大连接数。

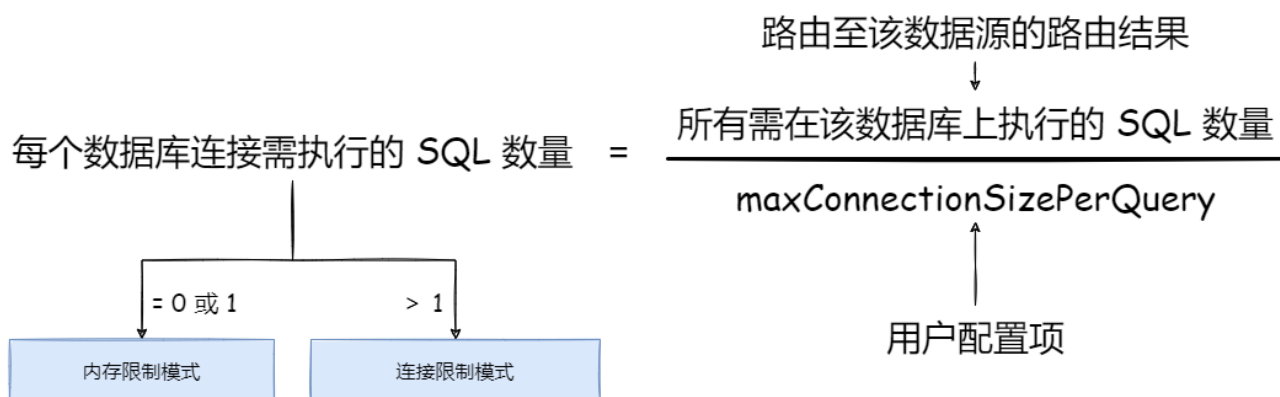
执行引擎分为准备和执行两个阶段。

准备阶段

顾名思义，此阶段用于准备执行的数据。它分为结果集分组和执行单元创建两个步骤。

结果集分组是实现内化连接模式概念的关键。执行引擎根据 maxConnectionSizePerQuery 配置项，结合当前路由结果，选择恰当的连接模式。具体步骤如下：

1. 将 SQL 的路由结果按照数据源的名称进行分组。
2. 通过下图的公式，可以获得每个数据库实例在 maxConnectionSizePerQuery 的允许范围内，每个连接需要执行的 SQL 路由结果组，并计算出本次请求的最优连接模式。



连接模式计算公式

Fig. 16: 连接模式计算公式

在 maxConnectionSizePerQuery 允许的范围内，当一个连接需要执行的请求数量大于 1 时，意味着当前的数据库连接无法持有相应的数据结果集，则必须采用内存归并；反之，当一个连接需要执行的请求数量等于 1 时，意味着当前的数据库连接可以持有相应的数据结果集，则可以采用流式归并。每一次的连接模式的选择，是针对每一个物理数据库的。也就是说，在同一次查询中，如果路由至一个以上的数据库，每个数据库的连接模式不一定一样，它们可能是混合存在的形态。

通过上一步骤获得的路由分组结果创建执行的单元。当数据源使用数据库连接池等控制数据库连接数量的技术时，在获取数据库连接时，如果不妥善处理并发，则有一定几率发生死锁。在多个请求相互等待对方释放数据库连接资源时，将会产生饥饿等待，造成交叉的死锁问题。

举例说明，假设一次查询需要在某一数据源上获取两个数据库连接，并路由至同一个数据库的两个分表查询。则有可能出现查询 A 已获取到该数据源的 1 个数据库连接，并等待获取另一个数据库连接；而查询 B 也已经在该数据源上获取到的一个数据库连接，并同样等待另一个数据库连接的获取。如果数据库连接池的允许最大连接数是 2，那么这 2 个查询请求将永久的等待下去。下图描绘了死锁的情况。

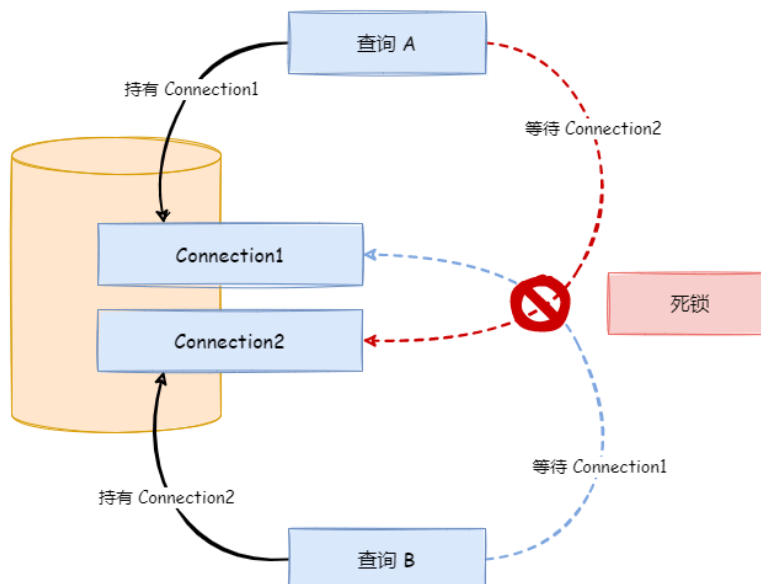


Fig. 17: 死锁

SphereEx-DBPlusEngine 为了避免死锁的出现，在获取数据库连接时进行了同步处理。它在创建执行单元时，以原子性的方式一次性获取本次 SQL 请求所需的全部数据库连接，杜绝了每次查询请求获取到部分资源的可能。由于对数据库的操作非常频繁，每次获取数据库连接时时都进行锁定，会降低 DBPlusEngine 的并发。因此，SphereEx-DBPlusEngine 在这里进行了 2 点优化：

1. 避免锁定一次性只需要获取 1 个数据库连接的操作。因为每次仅需要获取 1 个连接，则不会发生两个请求相互等待的场景，无需锁定。对于大部分 OLTP 的操作，都是使用分片键路由至唯一的数据节点，这会使得系统变为完全无锁的状态，进一步提升了并发效率。除了路由至单分片的情况，读写分离也在此范畴之内。
2. 仅针对内存限制模式时才进行资源锁定。在使用连接限制模式时，所有的查询结果集将在装载至内存之后释放掉数据库连接资源，因此不会产生死锁等待的问题。

执行阶段

该阶段用于真正的执行 SQL，它分为分组执行和归并结果集生成两个步骤。

分组执行将准备执行阶段生成的执行单元分组下发至底层并发执行引擎，并针对执行过程中的每个关键步骤发送事件。如：执行开始事件、执行成功事件以及执行失败事件。执行引擎仅关注事件的发送，它并不关心事件的订阅者。DBPlusEngine 的其他模块，如：分布式事务、调用链路追踪等，会订阅感兴趣的事件，并进行相应的处理。DBPlusEngine 通过在执行准备阶段的获取的连接模式，生成内存归并结果集或流式归并结果集，并将其传递至结果归并引擎，以进行下一步的工作。

执行引擎的整体结构划分如下图所示。

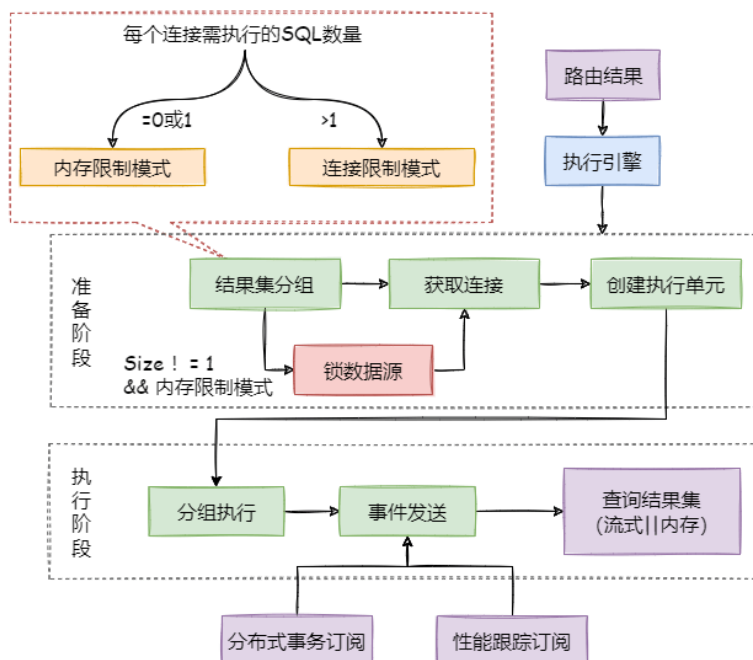


Fig. 18: 执行引擎的整体结构划分图

2.6.11 归并引擎

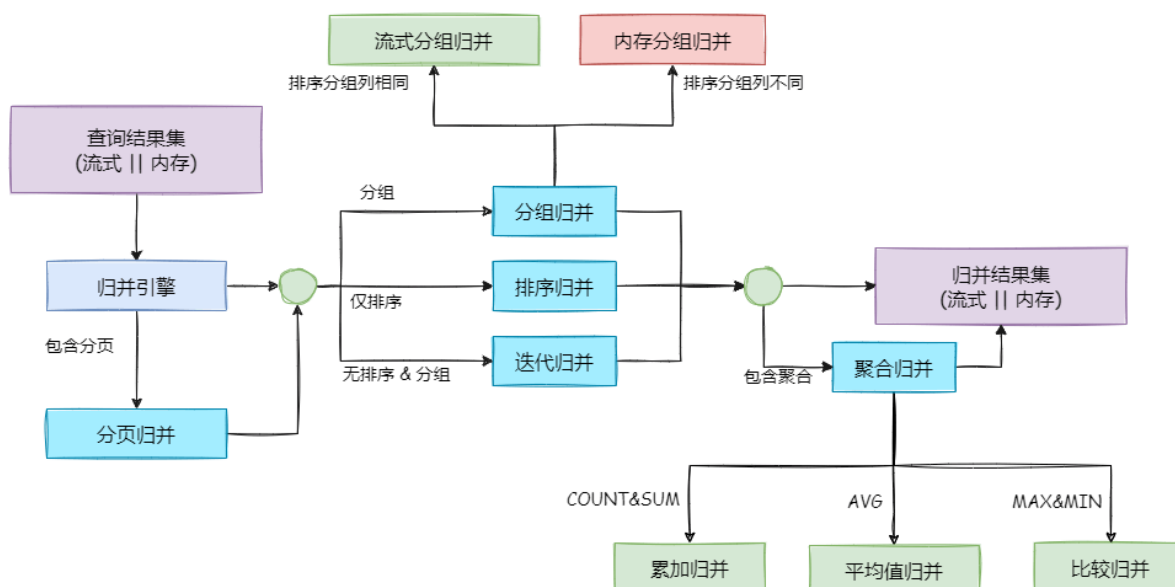


Fig. 19: 归并流程图

将从各个数据节点获取的多数据结果集，组合成为一个结果集并正确的返回至请求客户端，称为结果归并。DBPlusEngine 支持的结果归并从功能上分为遍历、排序、分组、分页和聚合 5 种类型，它们是组合而非互斥的关系。从结构划分，可分为流式归并、内存归并和装饰者归并。流式归并和内存归并是互斥的，装饰者归并可以在流式归并和内存归并之上做进一步的处理。由于从数据库中返回的结果集是逐条返回的，并不需要将所有数据一次性加载至内存中，因此，在进行结果归并时，沿用数据库返回结果集的方式进行归并，能够极大减少内存的消耗，是归并方式的优先选择。

流式归并是指每一次从结果集中获取到的数据，都能够通过逐条获取的方式返回正确的单条数据，它与数据库原生的返回结果集的方式最为契合。遍历、排序以及流式分组都属于流式归并的一种。

内存归并则是需要将结果集的所有数据都遍历并存储在内存中，再通过统一的分组、排序以及聚合等计算之后，再将其封装成为逐条访问的数据结果集返回。装饰者归并是对所有的结果集归并进行统一的功能增强，目前装饰者归并有分页归并和聚合归并这 2 种类型。

遍历归并

它是最为简单的归并方式。只需将多个数据结果集合并为一个单向链表即可。在遍历完成链表中当前数据结果集之后，将链表元素后移一位，继续遍历下一个数据结果集即可。

排序归并

由于在 SQL 中存在 ORDER BY 语句，因此每个数据结果集自身是有序的，因此只需要将数据结果集当前游标指向的数据值进行排序即可。这相当于对多个有序的数组进行排序，归并排序是最适合此场景的排序算法。

SphereEx-DBPlusEngine 在对排序的查询进行归并时，将每个结果集的当前数据值进行比较（通过实现 Java 的 Comparable 接口完成），并将其放入优先级队列。每次获取下一条数据时，只需将队列顶端结果集的游标下移，并根据新游标重新进入优先级排序队列找到自己的位置即可。

通过一个例子来说明 DBPlusEngine 的排序归并，下图是一个通过分数进行排序的示例图。图中展示了 3 张表返回的数据结果集，每个数据结果集已经根据分数排序完毕，但是 3 个数据结果集之间是无序的。将 3 个数据结果集的当前游标指向的数据值进行排序，并放入优先级队列，t_score_0 的第一个数据值最大，t_score_2 的第一个数据值次之，t_score_1 的第一个数据值最小，因此优先级队列根据 t_score_0, t_score_2 和 t_score_1 的方式排序队列。

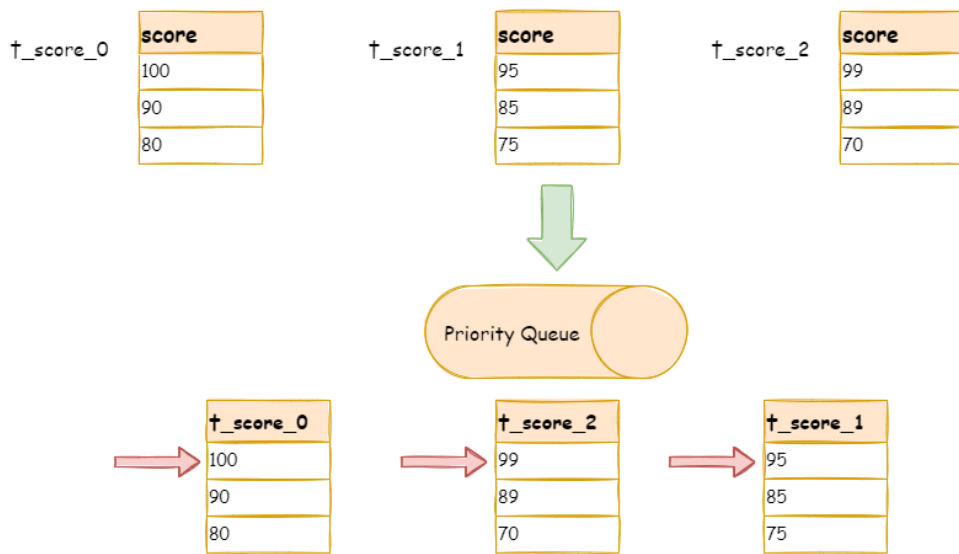


Fig. 20: 排序归并示例图 1

下图则展现了进行 next 调用的时候，排序归并是如何进行的。通过图中我们可以看到，当进行第一次 next 调用时，排在队列首位的 t_score_0 将会被弹出队列，并且将当前游标指向的数据值（也就是 100）返回至查询客户端，并且将游标下移一位之后，重新放入优先级队列。而优先级队列也会根据 t_score_0 的当前数据结果集指向游标的数据值（这里是 90）进行排序，根据当前数值，t_score_0 排列在队列的最后一位。之前队列中排名第二的 t_score_2 的数据结果集则自动排在了队列

首位。在进行第二次 next 时，只需要将目前排列在队列首位的 t_score_2 弹出队列，并且将其数据结果集游标指向的值返回至客户端，并下移游标，继续加入队列排队，以此类推。当一个结果集中已经没有数据了，则无需再次加入队列。

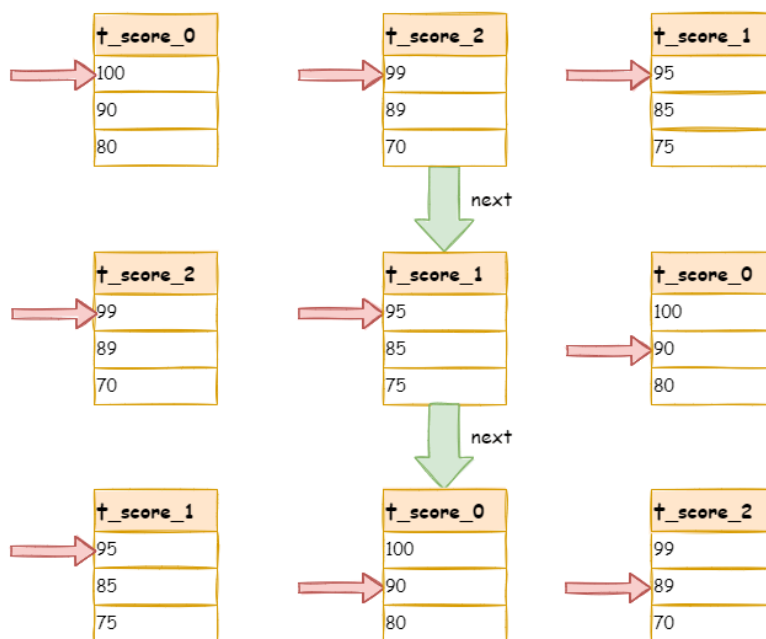


Fig. 21: 排序归并示例图 2

可以看到，对于每个数据结果集中的数据有序，而多数据结果集整体无序的情况下，ShardingSphere 无需将所有数据都加载至内存即可排序。它使用的是流式归并的方式，每次 next 仅获取唯一正确的一条数据，极大的节省了内存的消耗。从另一个角度来说，DBPlusEngine 的排序归并，是在维护数据结果集的纵轴和横轴这两个维度的有序性。纵轴是指每个数据结果集本身，它是天然有序的，它通过包含 ORDER BY 的 SQL 所获取。横轴是指每个数据结果集当前游标所指向的值，它需要通过优先级队列来维护其正确顺序。每一次数据结果集当前游标的下移，都需要将该数据结果集重新放入优先级队列排序，而只有排列在队列首位的数据结果集才可能发生游标下移的操作。

分组归并

分组归并的情况最为复杂，它分为流式分组归并和内存分组归并。流式分组归并要求 SQL 的排序项与分组项的字段以及排序类型（ASC 或 DESC）必须保持一致，否则只能通过内存归并才能保证数据正确性。

举例说明，假设根据科目分片，表结构中包含考生的姓名（为了简单起见，不考虑重名的情况）和分数。通过 SQL 获取每位考生的总分，可通过如下 SQL：

```
sql> SELECT name, SUM(score) FROM t_score GROUP BY name ORDER BY name;
```

在分组项与排序项完全一致的情况下，取得的数据是连续的，分组所需的数据全数存在于各个数据结果集的当前游标所指向的数据值，因此可以采用流式归并。如下图所示。

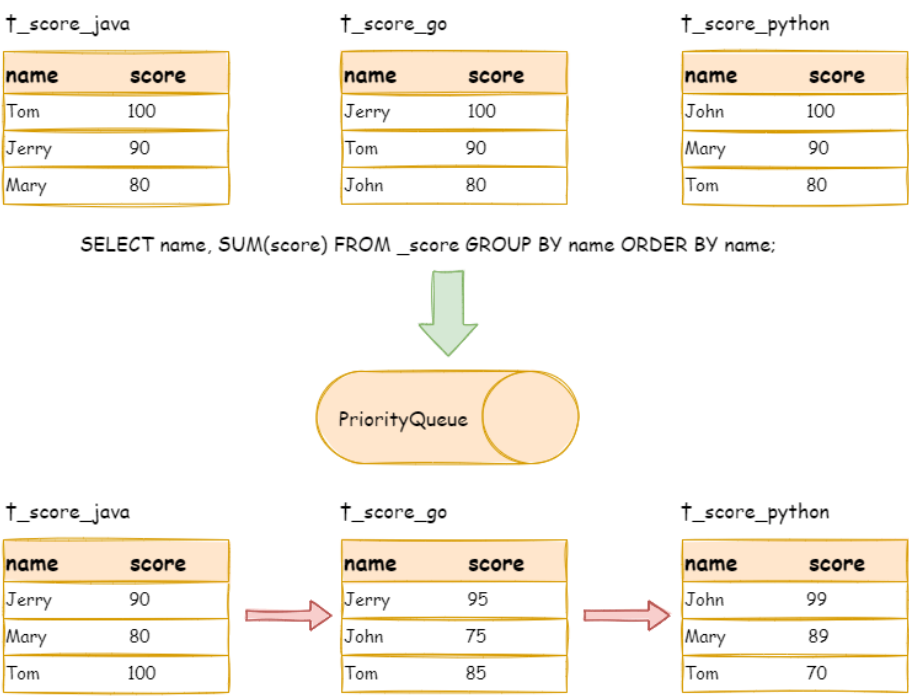


Fig. 22: 流式分组归并示例图 1

进行归并时，逻辑与排序归并类似。下图展现了进行 next 调用的时候，流式分组归并是如何进行的。

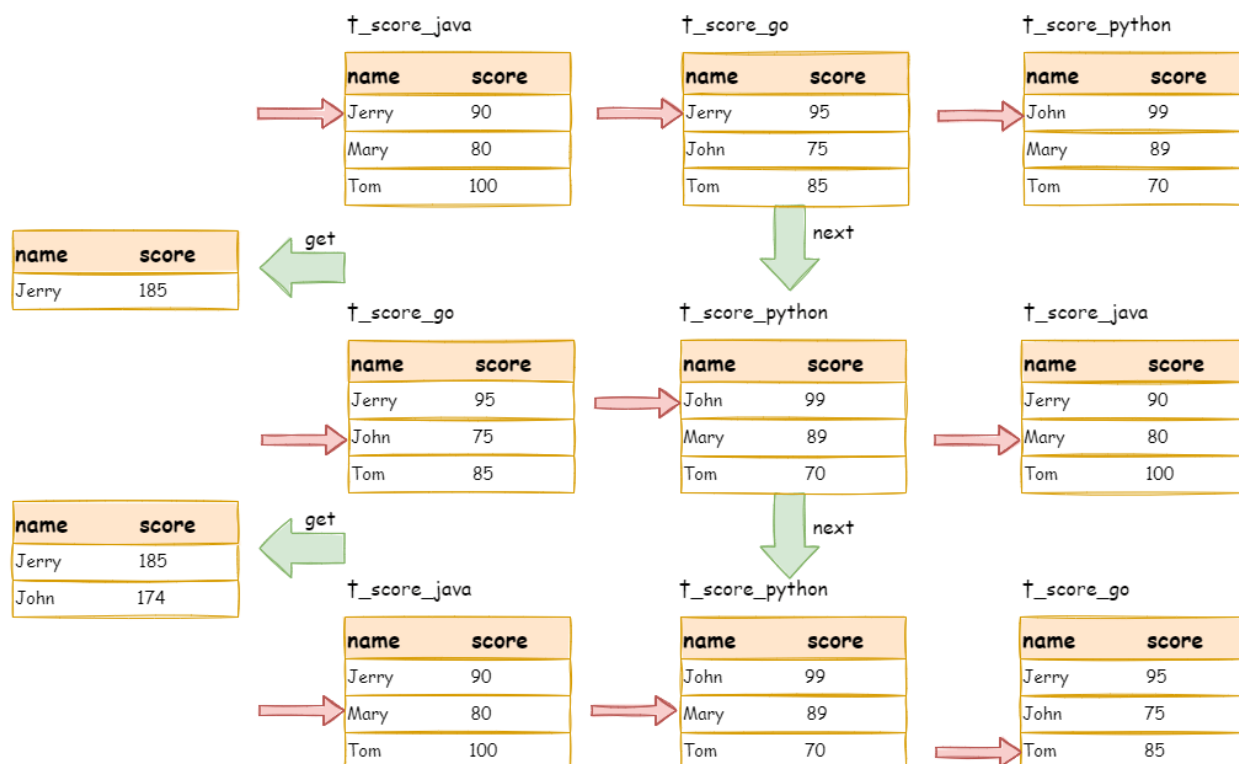


Fig. 23: 流式分组归并示例图 2

通过图中我们可以看到，当进行第一次 next 调用时，排在队列首位的 `t_score_java` 将会被弹出队列，并且将分组值同为“Jerry”的其他结果集中的数据一同弹出队列。在获取了所有的姓名为“Jerry”的同学的分数之后，进行累加操作，那么在第一次 next 调用结束后，取出的结果集是“Jerry”的分数总和。与此同时，所有的数据结果集中的游标都将下移至数据值“Jerry”的下一个不同的数据值，并且根据数据结果集当前游标指向的值进行重排序。因此，包含名字顺着第二位的“John”的相关数据结果集则排在队列的前列。

流式分组归并与排序归并的区别仅仅在于两点：

1. 它会一次性的将多个数据结果集中的分组项相同的数据全数取出。
2. 它需要根据聚合函数的类型进行聚合计算。

对于分组项与排序项不一致的情况，由于需要获取分组的相关的数据值并非连续的，因此无法使用流式归并，需要将所有的结果集数据加载至内存中进行分组和聚合。例如，若通过以下 SQL 获取每位考生的总分并按照分数从高至低排序：

```
sql> SELECT name, SUM(score) FROM t_score GROUP BY name ORDER BY score DESC;
```

那么各个数据结果集中取出的数据与排序归并那张图的上半部分的表结构的原始数据一致，是无法进行流式归并的。当 SQL 中只包含分组语句时，根据不同数据库的实现，其排序的顺序不一定与分组顺序一致。但由于排序语句的缺失，则表示此 SQL 并不在意排序顺序。因此，DBPlusEngine 通过 SQL 优化的改写，自动增加与分组项一致的排序项，使其能够从消耗内存的内存分组归并方式转化为流式分组归并方案。

聚合归并

无论是流式分组归并还是内存分组归并，对聚合函数的处理都是一致的。除了分组的 SQL 之外，不进行分组的 SQL 也可以使用聚合函数。因此，聚合归并是在之前介绍的归并类的之上追加的归并能力，即装饰者模式。聚合函数可以归类为比较、累加和求平均值这 3 种类型。比较类型的聚合函数是指 MAX 和 MIN。它们需要对每一个同组的结果集数据进行比较，并且直接返回其最大或最小值即可。累加类型的聚合函数是指 SUM 和 COUNT。它们需要将每一个同组的结果集数据进行累加。求平均值的聚合函数只有 AVG。它必须通过 SQL 改写的 SUM 和 COUNT 进行计算，相关内容已在 SQL 改写的内容中涵盖，不再赘述。

分页归并

上文所述的所有归并类型都可能进行分页。分页也是追加在其他归并类型之上的装饰器，SphereEx-DBPlusEngine 通过装饰者模式来增加对数据结果集进行分页的能力。分页归并负责将无需获取的数据过滤掉。SphereEx-DBPlusEngine 的分页功能比较容易让使用者误解，用户通常认为分页归并会占用大量内存。在分布式的场景中，将 LIMIT 10000000, 10 改写为 LIMIT 0, 10000010，才能保证其数据的正确性。用户非常容易产生 DBPlusEngine 会将大量无意义的的数据加载至内存中，造成内存溢出风险的错觉。其实，通过流式归并的原理可知，会将数据全部加载到内存中的只有内存分组归并这一种情况。

而通常来说，进行 OLAP 的分组 SQL，不会产生大量的结果数据，它更多的用于大量的计算，以及少量结果产出的场景。除了内存分组归并这种情况之外，其他情况都通过流式归并获取数据结果集，因此 SphereEx-DBPlusEngine 会通过结果集的 next 方法将无需取出的数据全部跳过，并不会将其存入内存。但同时需要注意的是，由于排序的需要，大量的数据仍然需要传输到 DBPlusEngine 的内存空间。因此，采用 LIMIT 这种方式分页，并非最佳实践。由于 LIMIT 并不能通过索引查询数据，因此如果可以保证 ID 的连续性，通过 ID 进行分页是比较好的解决方案，例如：

```
sql> SELECT * FROM t_order WHERE id > 100000 AND id <= 100010 ORDER BY id;
```

或通过记录上次查询结果的最后一条记录的 ID 进行下一页的查询，例如：

```
sql> SELECT * FROM t_order WHERE id > 10000000 LIMIT 10;
```

3.1 解决某互联网金融客户千亿数据存储和扩容的问题

3.1.1 客户痛点

为了应对快速发展的业务和激增的海量数据，数据架构进行了数次演进，但随着产品升级迭代，早期的解决方案演变成为了眼前的问题，通过业务框架实现的数据分片方案导致业务代码复杂度增加、维护成本不断攀升，紧耦合的弊端原形毕露，应用每次升级都需要投入较多的精力对分片做相应调整，研发人员难以专注于业务本身。技术团队经权衡后开始考虑使用成熟的分库分表组件来承担这部分工作，让业务系统升级和架构调整不再复杂。

3.1.2 客户调研

基于自研框架分片和基于 DBPlusEngine 分片的对比如下：

	基于自研框架分片	基于 DBPlusEngine 分片
性能	高	高
代码耦合度	高	低
业务入侵程度	高	低
升级难度	高	低
扩展性	一般	良好

DBPlusEngine 优势 1. 产品成熟稳定 2. 极致性能表现 3. 处理海量数据 4. 架构灵活扩展

3.1.3 客户改造与架构

SphereEx-DBPlusEngine 在主力客户互联网金融业务落地过程中对产品的功能和性能方面进行了更多的支持和提升，产品再一次经历典型案例的打磨。

1. 升级 SQL 引擎
2. 分布式主键
3. 业务分片键值注入
4. SQL 解析结果缓存
5. JDBC 元数据信息缓存
6. Bind 表 & 广播表的使用

7. 自动化执行引擎 & 流式归并

经两团队通力合作，客户的互联网金融业务与 SphereEx-DBPlusEngine 相结合的各项指标满足预期，性能与原生 JDBC 几乎一致。改造后的架构

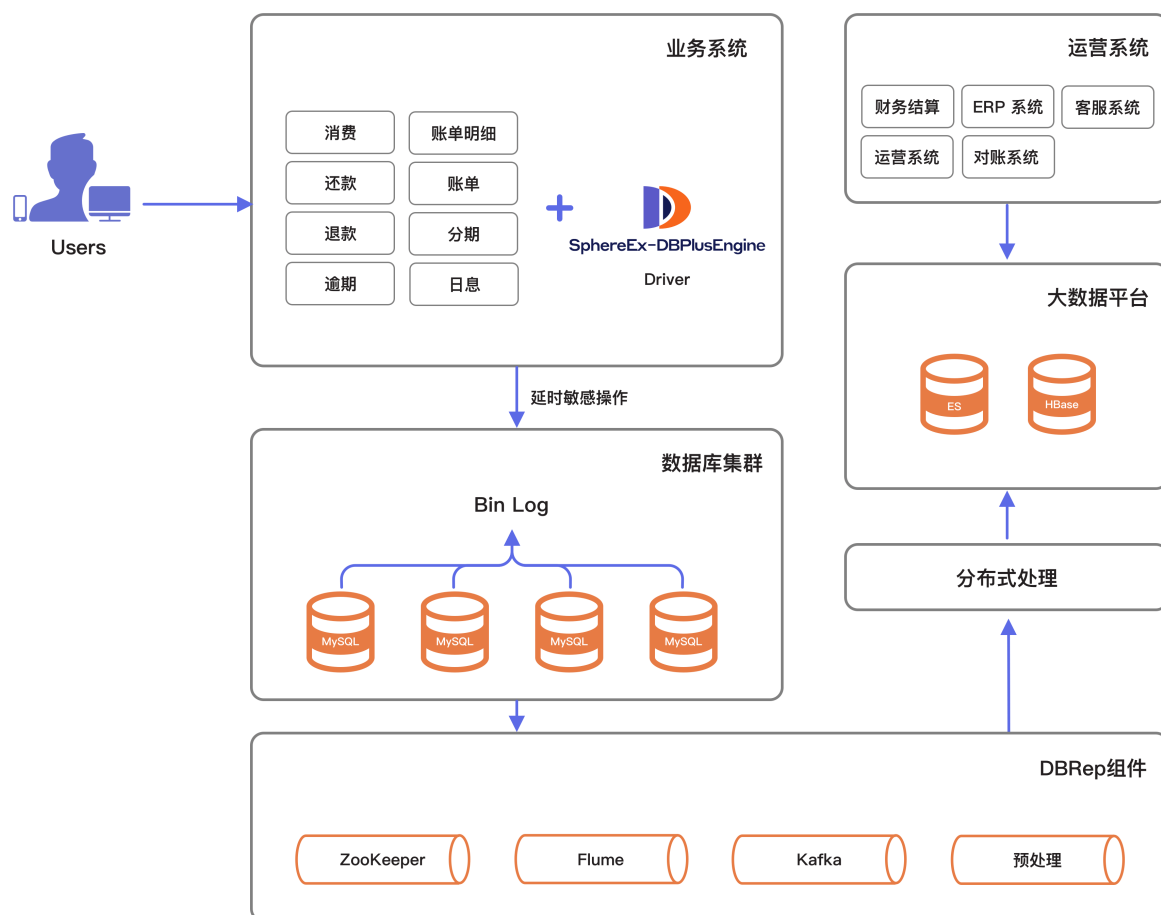


Fig. 1: Architecture

3.2 服务保障

3.2.1 等级服务协议 SLA

具体可参考SLA。

3.2.2 服务反馈

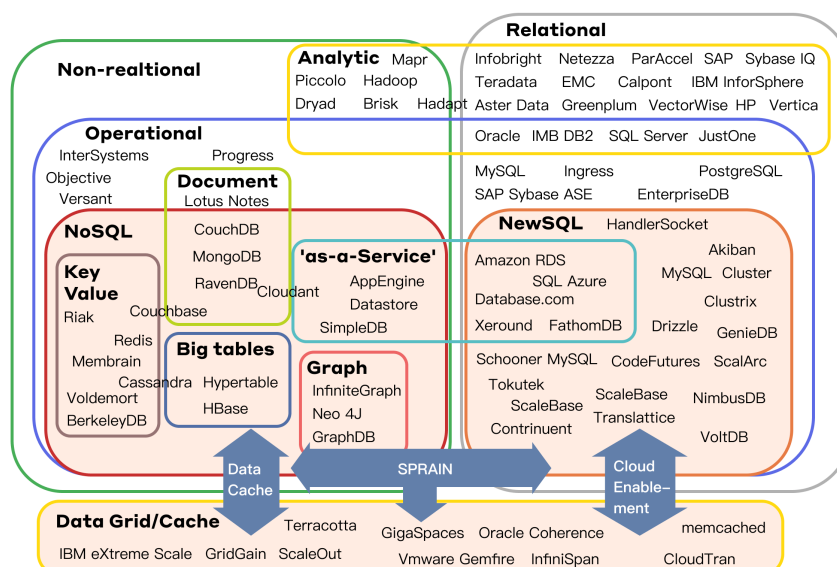
SphereEx 有专业的售后服务团队为您提供售后星级服务。联系方式为[联系我们](#)。

4.1 行业趋势

在数据库碎片化的趋势中，DBPlusEngine 让数据与应用的连接，变得更简单。

近年来，在新兴应用场景驱动下，数据库技术发展飞快，众多优秀产品如雨后春笋般涌现。数据库作为数字基础设施的核心，角色也愈发重要。

企业的数据库架构逐步向开源、分布式和云原生发展。同时在数据库品类方面，企业从原来的“大一统”架构，过渡到现今的多种数据库各尽所能，数据库的多元并存已是常态，也是不可逆的趋势。



在二十年前，结合当时的技术发展、业务复杂度以及体量，企业可以选择一款商用数据库即可满足所有的业务场景，并不是一件难事。

现如今业务发展驱动着架构转型，“大一统”的数据架构已经逐渐被时代所摒弃。

在 IT 架构中数据库碎片化的趋势下，我们不仅要应对来自业务侧与日俱增的压力，还要满足企业技术路线转型中多元化的需求。

结合数据库所面临的多种新挑战，可以考虑用一种高效且低成本的方式来增强现有数据库的性能，同时结合业务需求灵活补充数据库的功能，并尽量使异构数据库对上层业务透明化，这是最佳答案之一。

4.2 技术原理

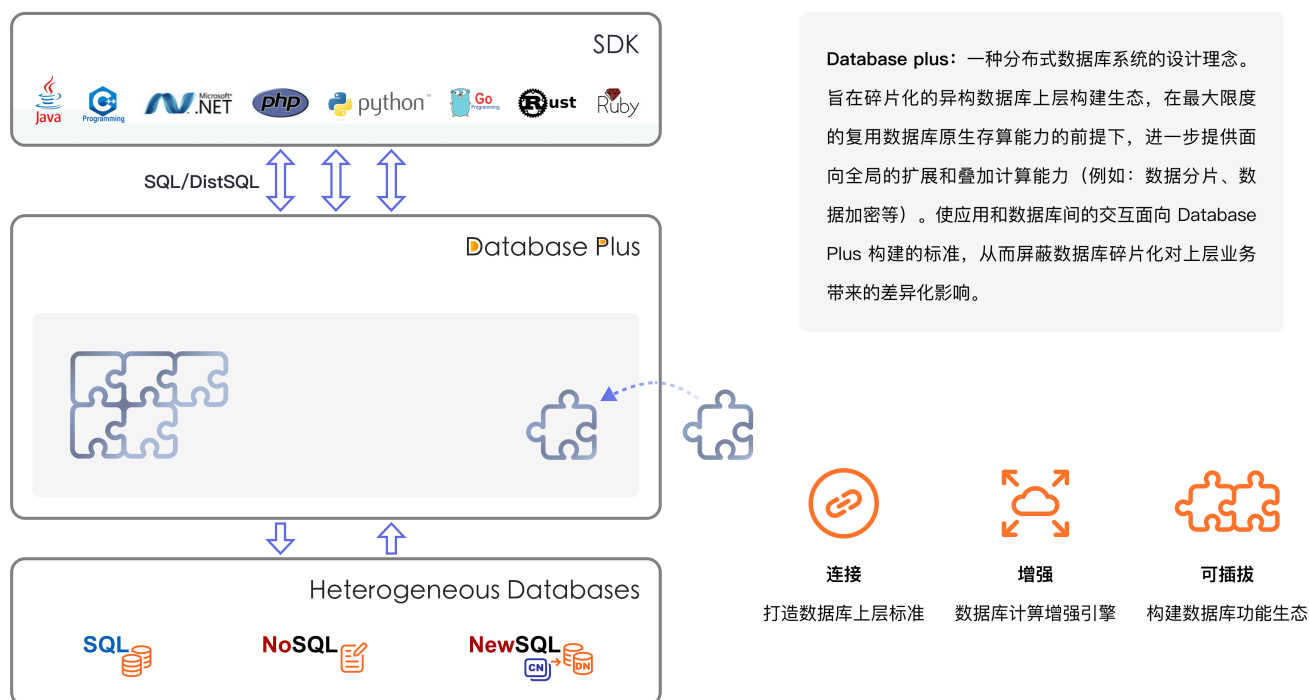
DBPlusEngine 采用了 Database Plus 设计哲学，该理念致力于构建数据库上层的标准和生态，在生态中补充数据库所缺失的能力。

- Database Plus：是 ShardingSphere 的设计哲学；
- ShardingSphere：是 Database Plus 的工程化实现；
- DBPlusEngine：是在 ShardingSphere 基础上面向企业用户的增强产品。

作为创建分布式数据库系统的指导性概念，Database Plus 在已经呈现出碎片化趋势的数据库之上，去建立全新的连接关系。这个过程将在无入侵上层应用前提下，在数据库上层提供数据增强服务，如分布式、数据控制和流量控制等。

对于存储层，需要为用户保留了原有数据库技术栈，稳定性、兼容性及运维习惯等均无变化，可靠性也有更好的保证。

这样，应用程序只需要与标准服务层对话，而不需要再关心底层数据库间的差异和能力，形成一个闭环生态环境。



Database Plus 的三个核心特性分别是连接、增强和可插拔。由“点”到“线”，最后形成一个生态“面”。

■ 连接：打造数据库上层标准

连接是 Database Plus 能力的基础“点”。即以『数据库网关』的形态为应用系统提供统一数据库入口，提供一个可以适配于各种数据库 SQL 方言和访问协议的中间层，Database Plus 将 SQL 解析而成的 AST（抽象语法树）根据其他数据库方言的规则重新生成 SQL。它可屏蔽异构数据库方言差异，也可屏蔽业务中异构开发语言的差异。

向上，以标准数据库协议支持多种开发语言，可做到应用无感平滑使用；向下，支持多种数据库产品既满足北向多种开发语言，同时还支持南向多种数据库产品。目前已支持 MySQL、PostgreSQL、openGauss 等数据库协议，以及 MySQL、PostgreSQL、openGauss、SQL Server、Oracle 和所有支持 SQL92 标准的 SQL 方言。

■ 增强：数据库计算增强引擎

基于连接的“点”，可以进一步延伸到这一条增强功能“线”。

传统数据库的实践经验是时间和多场景所沉淀的精华，Database Plus 在复用数据库的存储和原生计算能力基础上，再对其算力、容量及功能进行突破。

当流量进入到了网关后，Database Plus 通过全局化的能力在分布式、数据控制和流量控制三个方面进行增强。

数据分片、弹性伸缩、高可用、读写分离、分布式事务及基于垂直拆分的异构数据库联邦查询等功能，都是 Database Plus 在分布式的异构数据库全局层面下所能提供的增强能力，以排列组合的方式提供给用户使用。

■ 可插拔：构建数据库功能生态

面向用户，Database Plus 以一个“面”的形态去呈现，即生态面。

我们可尝试把这个“面”理解为所熟知的应用商店，你只需在众多的插件中选择所需要的功能即可，它可以单独使用，也可以组合使用，甚至你可以在应用商店中打造自己的定制化插件。

通过可插拔体系，Database Plus 将能够真正的构建面向数据库的功能生态，将异构数据库的全局能力统一纳管。它不仅面向集中式数据库的分布式化，也同时面向分布式数据库的竖井功能一体化。

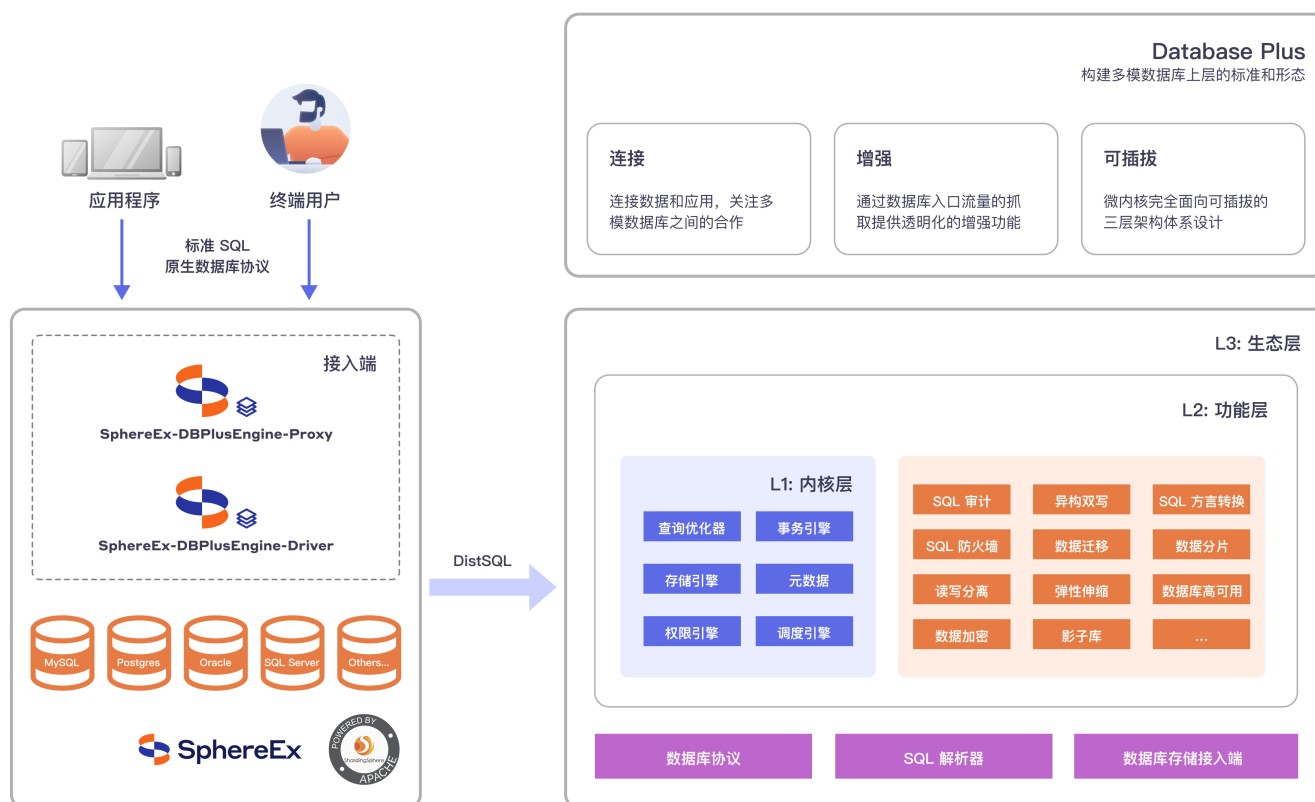
可插拔的特性可在提供更好扩展的同时，还能做到收敛，只提供用户所需要的内容。生态无边界，能力可插拔。

4.3 技术架构

4.3.1 整体架构

DBPlusEngine 的可插拔架构划分为 3 层，分别是 L1 内核层、L2 功能层、L3 生态层。

DBPlusEngine 整体架构如下图所示。



L1 内核层

是数据库基本能力的抽象，其所有组件均必须存在，但具体实现方式可通过可插拔的方式更换。主要包括查询优化器、分布式事务引擎、分布式执行引擎、权限引擎和调度引擎等。

L2 功能层

用于提供增量能力，其所有组件均是可选的，可以包含零至多个组件。组件之间完全隔离，互无感知，多组件可通过叠加的方式相互配合使用。主要包括数据分片、读写分离、数据库高可用、数据加密、影子库等。用户自定义功能可完全面向 DBPlusEngine 定义的顶层接口进行定制化扩展，而无需改动内核代码。

L3 生态层

用于对接和融入现有数据库生态，包括数据库协议、SQL 解析器和存储适配器，分别对应于 DBPlusEngine 以数据库协议提供服务的方式、SQL 方言操作数据的方式以及对接存储节点的数据库类型。

DBPlusEngine 的接入形态包括驱动端和代理端两种形态，即 DBPlusEngine-Driver 和 DBPlusEngine-Proxy。

DBPlusEngine-Driver

轻量级的 Java 框架，在 Java 的 JDBC 层提供的额外服务。它使用客户端直连数据库，以 jar 包形式提供服务，无需额外部署和依赖，可理解为增强版的 JDBC 驱动，完全兼容 JDBC 和各种 ORM 框架。

- 适用于任何基于 JDBC 的 ORM 框架，如：JPA、Hibernate、Mybatis、Spring JDBC Template 或直接使用 JDBC；
- 支持任何第三方的数据库连接池，如：DBCP，C3P0，BoneCP，HikariCP 等；
- 支持任意实现 JDBC 规范的数据库，目前支持 MySQL，PostgreSQL，Oracle，SQLServer 以及任何可使用 JDBC 访问的数据库。

DBPlusEngine-Proxy

透明化的数据库代理端，提供封装了数据库二进制协议的服务端版本，用于完成对异构语言的支持。目前提供 MySQL 和 PostgreSQL（兼容 openGauss 等基于 PostgreSQL 的数据库）版本，它可以使用任何兼容 MySQL/PostgreSQL 协议的访问客户端（如：MySQL Command Client、MySQL Workbench、Navicat 及 DBeaver 等）操作数据，对 DBA 更加友好。

- 向应用程序完全透明，可直接当做 MySQL/PostgreSQL 使用；
- 适用于任何兼容 MySQL/PostgreSQL 协议的客户端。

4.3.2 架构优势

- 高性能

驱动程序端历经多年打磨，效率接近原生 JDBC，性能极致。主流竞品中，目前只有 ShardingSphere 提供驱动端接入形态。

- 高兼容

代理端适用于任何兼容 MySQL/PostgreSQL 协议的客户端，驱动程序端支持任意实现 JDBC 规范的数据库。

- 高扩展

面对数据库替换场景，DBPlusEngine 可满足业务平滑迁移，对业务零侵入。

- 低成本

保留原有数据库技术栈，对 DBA 友好，学习及管理成本低。

- 安全稳定

不干预数据库内核，基于成熟的数据库底座提供增能力，兼顾安全性及稳定性。

■ 弹性扩展

具备弹性的计算和存储能力，可满足计算及存储层不断变化的需求，在线完成数据库拆分和迁移；

■ 开放生态

可插拔模型，使内核、功能组件以及生态对接完全能够灵活的方式进行插拔式扩展，用户可以像使用积木一样定制适合业务的独特系统。

4.4 部署与运维

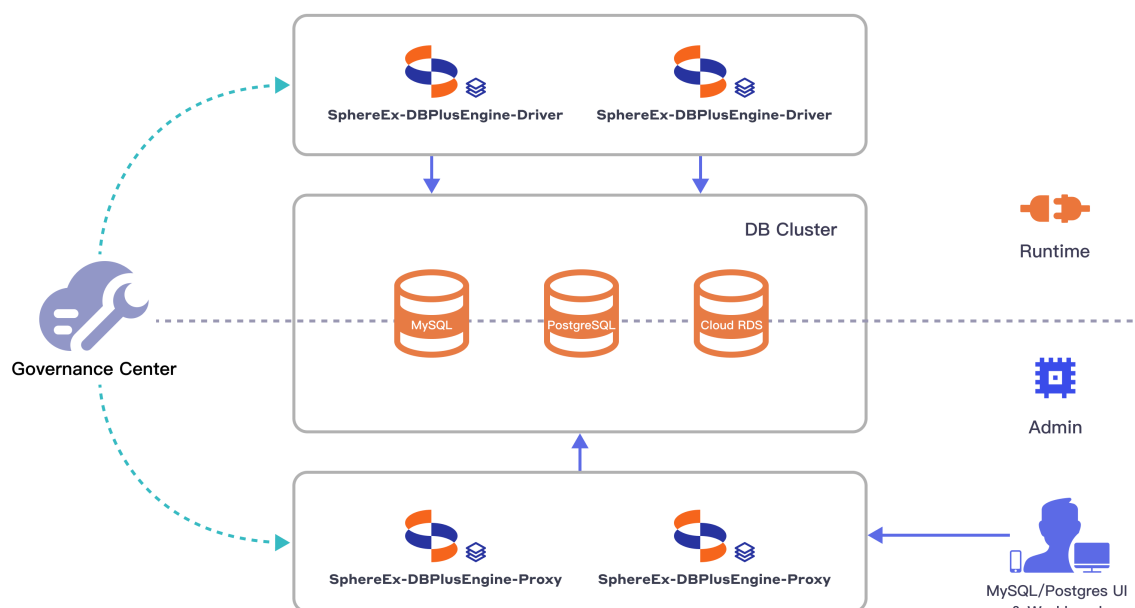
4.4.1 部署形态

驱动端注重性能，而代理端对运维管理更友好，在部署方式上，混合使用的方式是更好的选择。

无论是 Apache ShardingSphere，还是基于其进行增强的 DBPlusEngine，都是多接入端共同组成的生态圈。

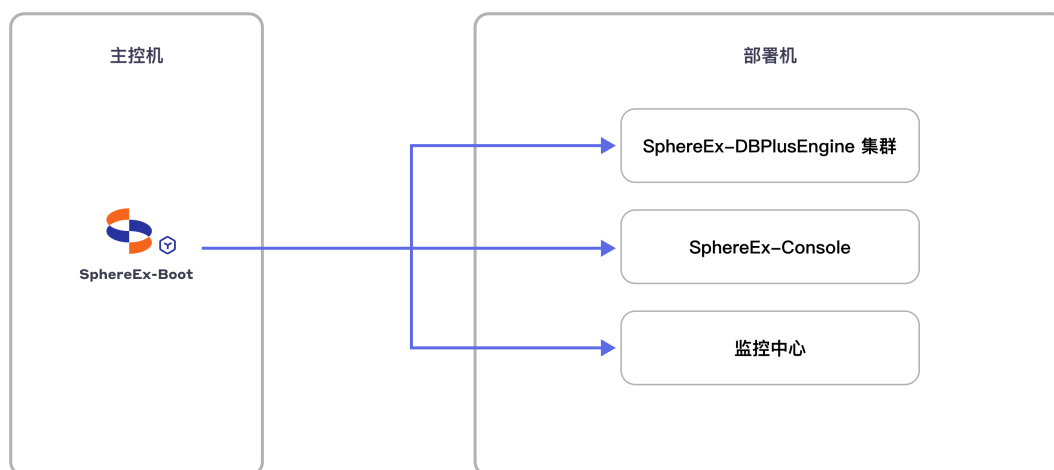
- DBPlusEngine-Driver 采用无中心化架构，与应用程序共享资源，适用于 Java 开发的高性能的轻量级 OLTP 应用；
- DBPlusEngine-Proxy 提供静态入口以及异构语言的支持，独立于应用程序部署，适用于 OLAP 应用以及对分片数据库进行管理和运维的场景。

通过混合使用 DBPlusEngine-Driver 和 DBPlusEngine-Proxy，并采用同一注册中心统一配置分片策略，能够灵活的搭建适用于各种场景的应用系统，使得架构师更加自由地调整适合于当前业务的最佳系统架构。



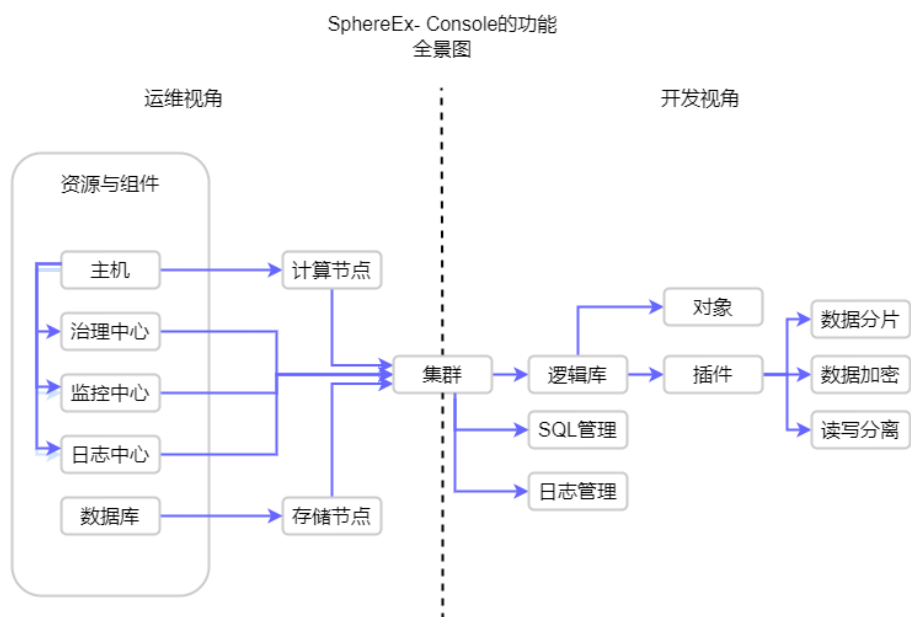
4.4.2 运维工具 SphereEx-Boot

SphereEx-Boot 是用于方便管理 DBPlusEngine-Proxy 集群的命令行工具，主要功能是对 DBPlusEngine-Proxy 进行安装、启动、停止、查看运行状态、卸载等操作。



4.4.3 管控工具 SphereEx-Console

SphereEx-Console 是对 DBPlusEngine 进行管理控制的可视化操作平台，提供更为易用的使用体验；同时，构建起以 DBPlusEngine 为核心的整体解决方案，包括针对资源、实例、插件等多种功能封装，为用户带来一站式的使用体验。



本文主要针对我司产品在基准测试和场景测试中性能结果的展示，提供用户产品性能方面参考，方便用户选型。随着内核和功能场景代码的不断优化，基准测试结果并非最优性能，文档也会同步更新。如果你需要评估我司产品性能，可依照本文提供的一些测试方法进行测试。

5.1 TPC-C 基准测试

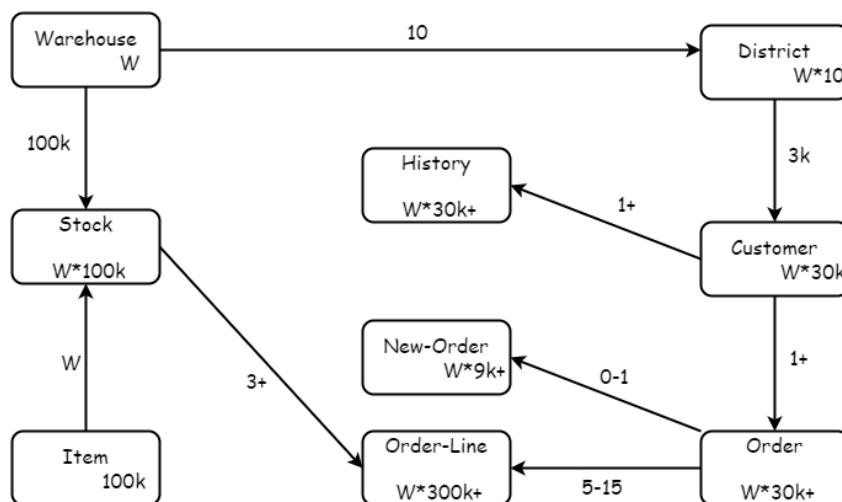
TPC-C 的测试场景模拟了在线电商的交易情况：

有一个大型商品批发商，在地理分布的多个 **区域 (district)** 有业务，并且使用仓库管理。当业务扩展的时候，公司将添加新的 **仓库 (warehouse)**。每个仓库负责为 10 个区域供货；每个区域为 3000 个 **客户 (customer)** 提供服务。所有仓库维护公司销售的 100,000 种 **商品 (item)** 的 **库存 (stock)** 记录。平均每个客户的一个 **订单 (order)** 有 10 个订单项 (order line)；所有订单中约 1% 的订单项在其直接所属的仓库中没有存货，需要由其他区域的仓库来供货。

客户向公司的系统发出新的订单或者查询其订单状态。

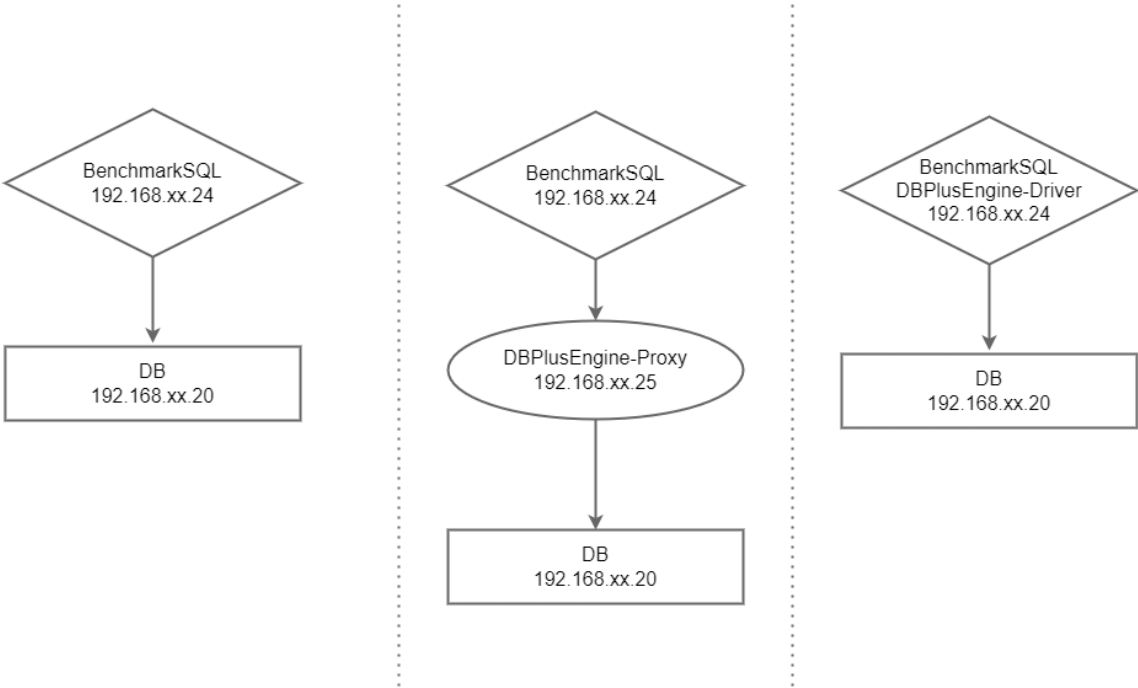
公司的系统也用于处理客户的付款，处理发货的订单，检查库存状态以便发现潜在的供货短缺问题。

下图显示了 **仓库 (warehouse)**，**区域 (district)** 和 **客户 (customer)** 间的关系。



5.1.1 测试方案一：单存储节点、单计算节点性能测试

选取 BenchmarkSQL 测试单存储节点下的 DBPlusEngine-Proxy、DBPlusEngine-Driver 与 MySQL 的性能对比。



测试目的

在相同数据量场景下，使用 BenchmarkSQL 压测模型分别对 DBPlusEngine-Proxy、DBPlusEngine-Driver 与 MySQL 进行性能对比，从而使用户对 DBPlusEngine-Proxy、DBPlusEngine-Driver 性能有详细的了解认识。

测试工具

BenchmarkSQL 是一款经典的开源数据库测试工具，内嵌了 TPC-C 测试脚本，可以对 PostgreSQL、MySQL、Oracle 以及 SQL Server 等数据库进行测试。

测试环境

应用	IP 地址	端口	版本
DBPlusEngine-Proxy	192.168.xx.25	3307	1.2
DBPlusEngine-Driver	192.168.xx.24	■	1.2
MySQL	192.168.xx.20	13306	8.0.29

服务器配置

条目	配置
CPU	48 C
内存	96 G
硬盘	SSD 820 G
JDK	17.0.2

测试步骤

1. 创建测试库。

```
mysql -utest -h192.168.xx.20 -P13306 -p
```

```
DROP DATABASE IF EXISTS test_tpcc;
```

```
CREATE DATABASE IF NOT EXISTS test_tpcc;
```

2. 安装测试工具。

```
wget https://udomain.dl.sourceforge.net/project/benchmarksql/benchmarksql-5.0.zip
yum -y install ant
unzip benchmarksql-5.0.zip -d /usr/local
cd /usr/local/benchmarksql-5.0
ant
cp mysql-connector-java-8.0.24.jar benchmarksql-5.0/run/lib
```

创建 props.proxy 文件。

```
db=postgres
driver=com.mysql.jdbc.Driver
conn=jdbc:mysql://192.168.xx.25:3307/sharding_db?useSSL=false&useServerPrepStmts=true&cachePrepStmts=true&
prepStmtCacheSize=8192&prepStmtCacheSqlLimit=8000
user=root
password=root
warehouses=200
loadWorkers=100
terminals=200
runTxnsPerTerminal=0
runMins=10
limitTxnsPerMin=0
terminalWarehouseFixed=true
newOrderWeight=45
paymentWeight=43
orderStatusWeight=4
deliveryWeight=4
stockLevelWeight=4
resultDirectory=my_result_%tY-%tm-%td_%tH%M%S
shardingNumber=1
```

3. YAML

```
rules:
- !SHARDING
  bindingTables:
  - bmsql_warehouse, bmsql_customer
  - bmsql_stock, bmsql_district, bmsql_order_line
  defaultDatabaseStrategy:
  none:
  defaultTableStrategy:
  none:
  keyGenerators:
  snowflake:
```



```
type: SNOWFLAKE
tables:
  bmsql_config:
    actualDataNodes: ds_0.bmsql_config

  bmsql_warehouse:
    actualDataNodes: ds_${0..0}.bmsql_warehouse
    databaseStrategy:
      standard:
        shardingColumn: w_id
        shardingAlgorithmName: mod_1

  bmsql_district:
    actualDataNodes: ds_${0..0}.bmsql_district
    databaseStrategy:
      standard:
        shardingColumn: d_w_id
        shardingAlgorithmName: mod_1

  bmsql_customer:
    actualDataNodes: ds_${0..0}.bmsql_customer
    databaseStrategy:
      standard:
        shardingColumn: c_w_id
        shardingAlgorithmName: mod_1

  bmsql_item:
    actualDataNodes: ds_${0..0}.bmsql_item
    databaseStrategy:
      standard:
        shardingColumn: i_id
        shardingAlgorithmName: mod_1

  bmsql_history:
    actualDataNodes: ds_${0..0}.bmsql_history
    databaseStrategy:
      standard:
        shardingColumn: h_w_id
        shardingAlgorithmName: mod_1

  bmsql_oorder:
    actualDataNodes: ds_${0..0}.bmsql_oorder
    databaseStrategy:
      standard:
        shardingColumn: o_w_id
        shardingAlgorithmName: mod_1

  bmsql_stock:
    actualDataNodes: ds_${0..0}.bmsql_stock
    databaseStrategy:
      standard:
        shardingColumn: s_w_id
        shardingAlgorithmName: mod_1

  bmsql_new_order:
    actualDataNodes: ds_${0..0}.bmsql_new_order
    databaseStrategy:
      standard:
        shardingColumn: no_w_id
        shardingAlgorithmName: mod_1

  bmsql_order_line:
    actualDataNodes: ds_${0..0}.bmsql_order_line
    databaseStrategy:
```

```
standard:
  shardingColumn: ol_w_id
  shardingAlgorithmName: mod_1
```

```
shardingAlgorithms:
  mod_1:
    type: MOD
    props:
      sharding-count: 1
```

4. 数据准备。

数据库初始化，通过 Proxy 建表并插入数据。

```
cd /usr/local/benchmarksql/run
./runDatabaseDestroy.sh props.proxy
./runDatabaseBuild.sh props.proxy
```

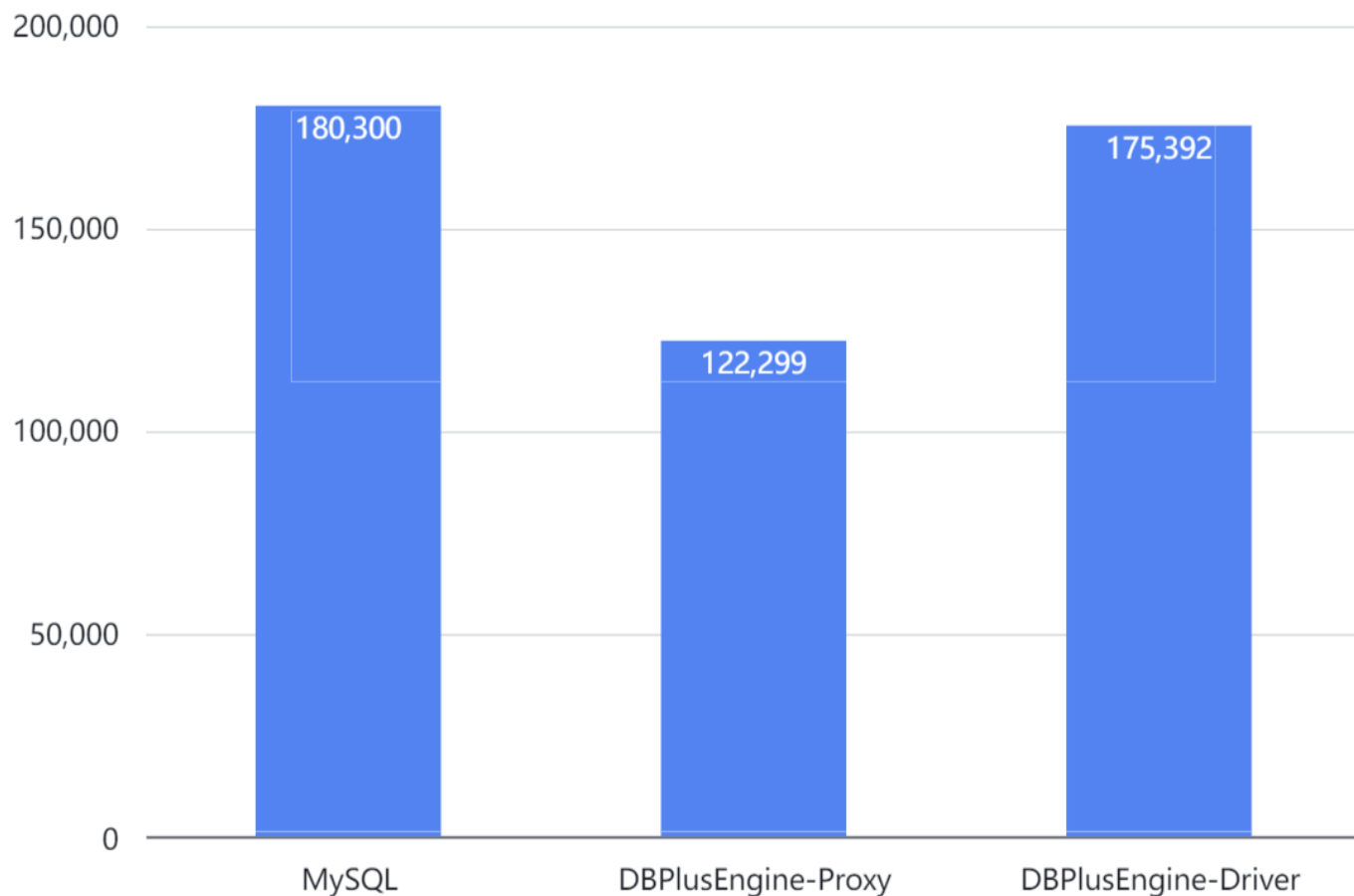
5. 压测阶段。

```
./runBenchmark.sh props.proxy
```

测试结果

测试对象	tpmC
MySQL	180,300
DBPlusEngine-Proxy	122,299
DBPlusEngine-Driver	175,392

单存储节点性能测试 tpmC

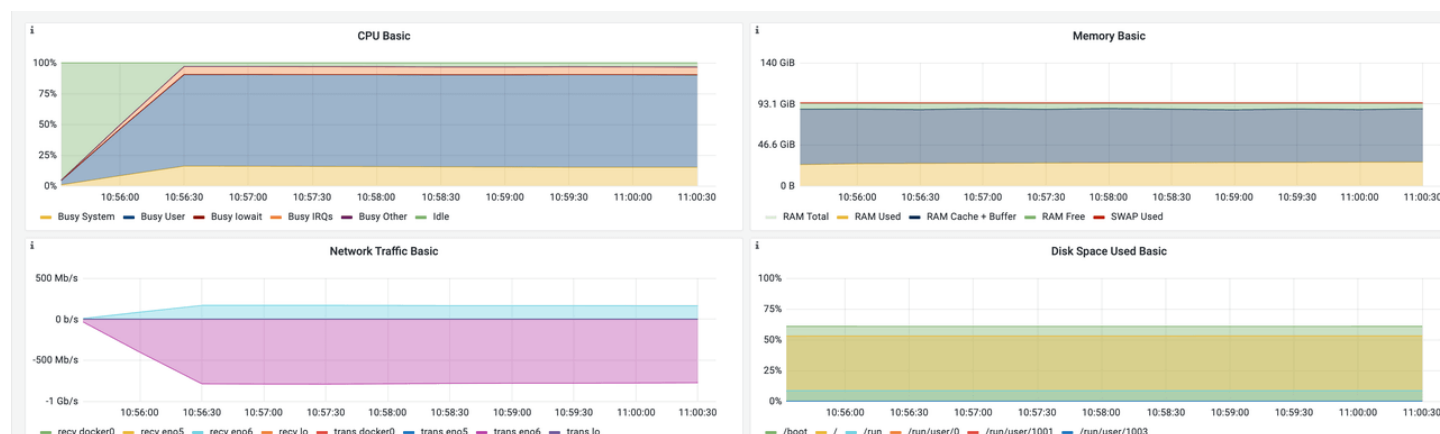


结论：在单存储节点场景下，DBPlusEngine-Driver 略低于 MySQL，DBPlusEngine-Proxy 低于 DBPlusEngine-Driver。

监控信息

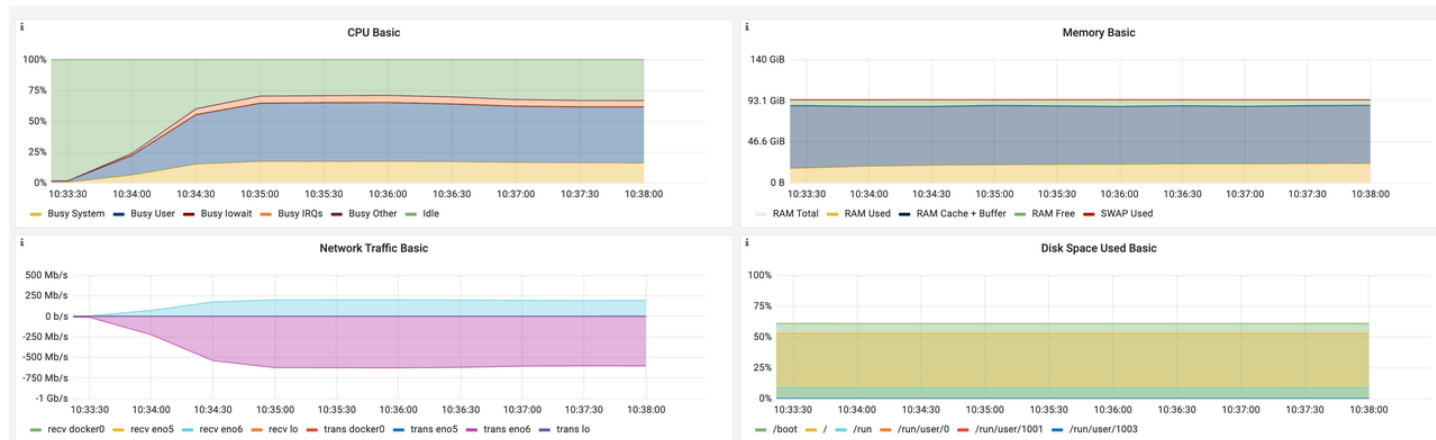
BenchmarkSQL ——> MySQL

■ MySQL

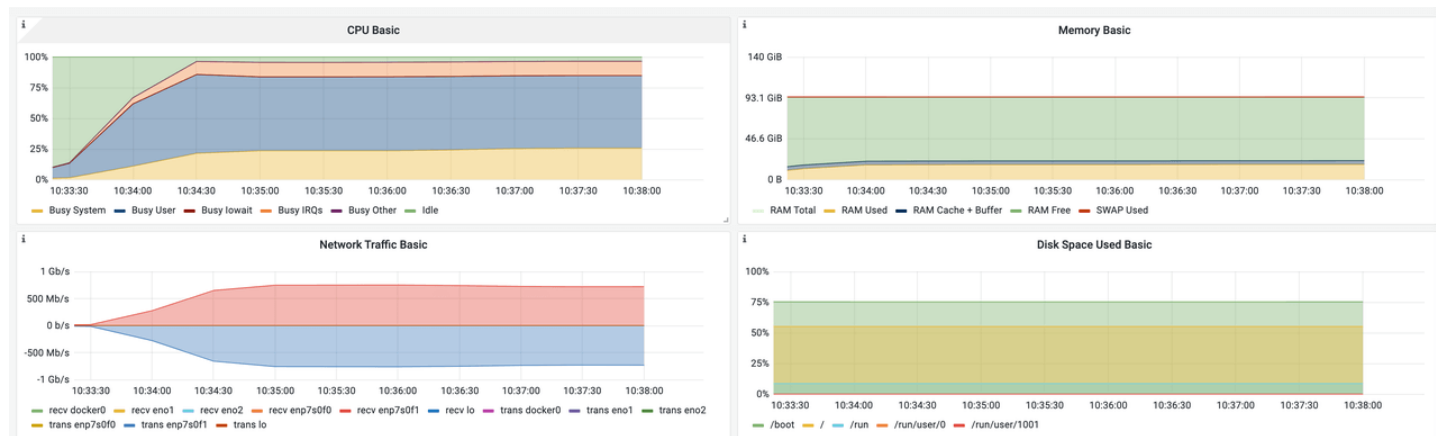


BenchmarkSQL —> Proxy —> MySQL

MySQL



Proxy



BenchmarkSQL —> Driver —> MySQL

MySQL



Driver



名词解释

warehouses: 仓库数，决定数据量

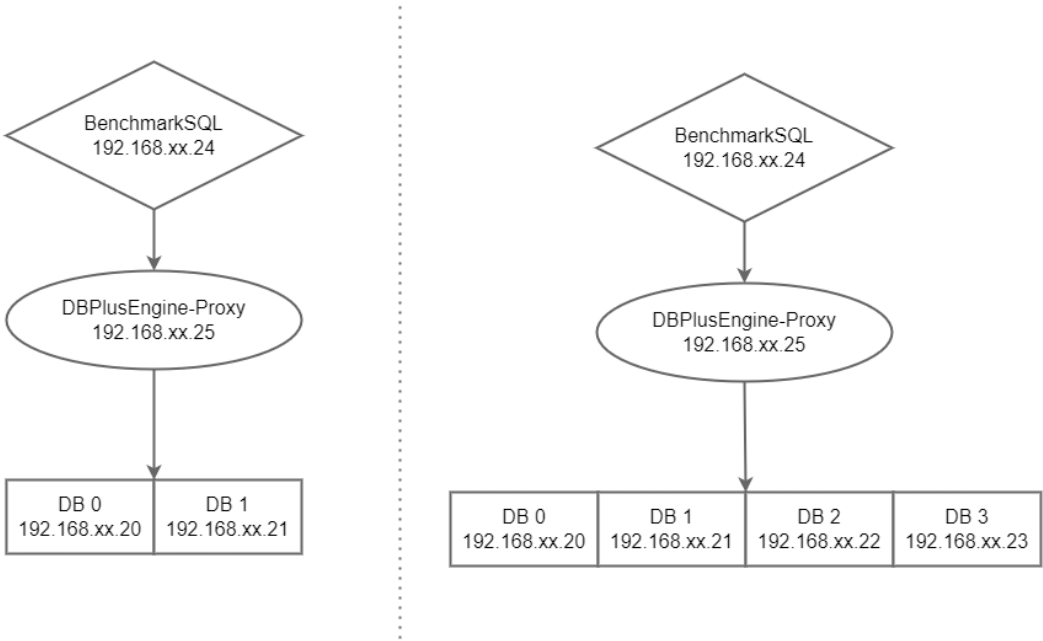
terminals: 线程数

runMins: 测试执行时长

tpmC: 每分钟处理事务数

5.1.2 测试方案二：存储节点扩容

使用 BenchmarkSQL 工具对 2/4 存储节点场景进行压测。



测试目的

在相同数据量下，使用 BenchmarkSQL 压测模型分别对 DBPlusEngine-Proxy 2/4 存储节点场景下进行对比，通过压测结果分析，使用户更加理解分片场景下优势。

测试工具

BenchmarkSQL

测试环境

应用	IP 地址	端口	版本
DBPlusEngine-Proxy	192.168.xx.25	3307	1.2
MySQL	192.168.xx.20	13306	8.0.29
MySQL	192.168.xx.21	13306	8.0.29
MySQL	192.168.xx.22	13306	8.0.29
MySQL	192.168.xx.23	13306	8.0.29

服务器配置

条目	配置
CPU	48 C
内存	96 G
硬盘	SSD 820 G
JDK	17.0.2

测试步骤

1. 创建测试表。

```
mysql -utest -h192.168.xx.20 -P13306 -p
```

```
DROP DATABASE IF EXISTS test_tpcc;
```

```
CREATE DATABASE IF NOT EXISTS test_tpcc;
```

```
mysql -utest -h192.168.xx.21 -P13306 -p
```

```
DROP DATABASE IF EXISTS test_tpcc;
```

```
CREATE DATABASE IF NOT EXISTS test_tpcc;
```

```
mysql -utest -h192.168.xx.22 -P13306 -p
```

```
DROP DATABASE IF EXISTS test_tpcc;
```

```
CREATE DATABASE IF NOT EXISTS test_tpcc;
```

```
mysql -utest -h192.168.xx.23 -P13306 -p
```

```
DROP DATABASE IF EXISTS test_tpcc;
```

```
CREATE DATABASE IF NOT EXISTS test_tpcc;
```

2. 安装测试工具。

```
wget https://udomain.dl.sourceforge.net/project/benchmarksql/benchmarksql-5.0.zip
```

```
yum -y install ant
```

```
unzip benchmarksql-5.0.zip -d /usr/local
```

```
cd /usr/local/benchmarksql-5.0
```

```
ant
```

```
cp mysql-connector-java-8.0.24.jar benchmarksql-5.0/run/lib
```

创建 props.proxy 文件。

```
db=postgres
driver=com.mysql.jdbc.Driver
conn=jdbc:mysql://192.168.xx.25:3307/sharding_db?useSSL=false&useServerPrepStmts=true&cachePrepStmts=true&
prepStmtCacheSize=8192&prepStmtCacheSqlLimit=8000
user=root
password=root
warehouses=200
loadWorkers=100
terminals=200
runTxnsPerTerminal=0
runMins=10
limitTxnsPerMin=0
terminalWarehouseFixed=true
newOrderWeight=45
```

```
paymentWeight=43
orderStatusWeight=4
deliveryWeight=4
stockLevelWeight=4
resultDirectory=my_result_%tY-%tm-%td_%tH%M%S
shardingNumber=2
```

3. YAML

```
rules:
- !SHARDING
  bindingTables:
  - bmsql_warehouse, bmsql_customer
  - bmsql_stock, bmsql_district, bmsql_order_line
  defaultDatabaseStrategy:
  none:
  defaultTableStrategy:
  none:
  keyGenerators:
  snowflake:
    type: SNOWFLAKE
  tables:
  bmsql_config:
    actualDataNodes: ds_0.bmsql_config

  bmsql_warehouse:
    actualDataNodes: ds_${0..1}.bmsql_warehouse
    databaseStrategy:
    standard:
      shardingColumn: w_id
      shardingAlgorithmName: mod_2

  bmsql_district:
    actualDataNodes: ds_${0..1}.bmsql_district
    databaseStrategy:
    standard:
      shardingColumn: d_w_id
      shardingAlgorithmName: mod_2

  bmsql_customer:
    actualDataNodes: ds_${0..1}.bmsql_customer
    databaseStrategy:
    standard:
      shardingColumn: c_w_id
      shardingAlgorithmName: mod_2

  bmsql_item:
    actualDataNodes: ds_${0..1}.bmsql_item
    databaseStrategy:
    standard:
      shardingColumn: i_id
      shardingAlgorithmName: mod_2

  bmsql_history:
    actualDataNodes: ds_${0..1}.bmsql_history
    databaseStrategy:
    standard:
      shardingColumn: h_w_id
      shardingAlgorithmName: mod_2

  bmsql_oorder:
    actualDataNodes: ds_${0..1}.bmsql_oorder
    databaseStrategy:
    standard:
```



```
shardingColumn: o_w_id
shardingAlgorithmName: mod_2
```

```
bmsql_stock:
actualDataNodes: ds_${0..1}.bmsql_stock
databaseStrategy:
standard:
shardingColumn: s_w_id
shardingAlgorithmName: mod_2
```

```
bmsql_new_order:
actualDataNodes: ds_${0..1}.bmsql_new_order
databaseStrategy:
standard:
shardingColumn: no_w_id
shardingAlgorithmName: mod_2
```

```
bmsql_order_line:
actualDataNodes: ds_${0..1}.bmsql_order_line
databaseStrategy:
standard:
shardingColumn: ol_w_id
shardingAlgorithmName: mod_2
```

```
shardingAlgorithms:
mod_2:
type: MOD
props:
sharding-count: 2
```

4. 数据准备。

数据库初始化，通过 Proxy 建表并插入数据。

```
cd /usr/local/benchmarksql/run
./runDatabaseDestroy.sh props.proxy
./runDatabaseBuild.sh props.proxy
```

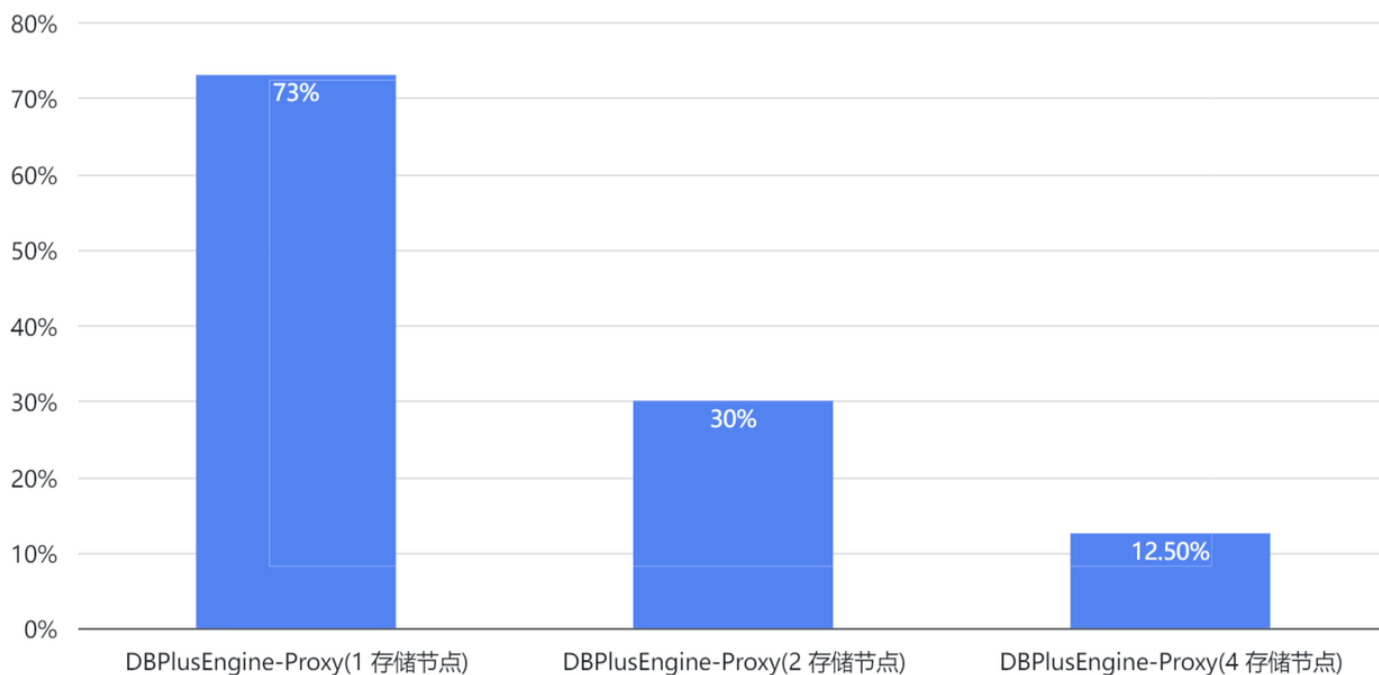
5. 压测阶段。

```
./runBenchmark.sh props.proxy
```

测试结果

测试对象	存储节点 AVG(CPU Load)	tpmC
DBPlusEngine-Proxy(1 存储节点)	73%	122,299
DBPlusEngine-Proxy(2 存储节点)	30%	122,580
DBPlusEngine-Proxy(4 存储节点)	12.5%	128,035

存储节点 AVG(CPU Load)



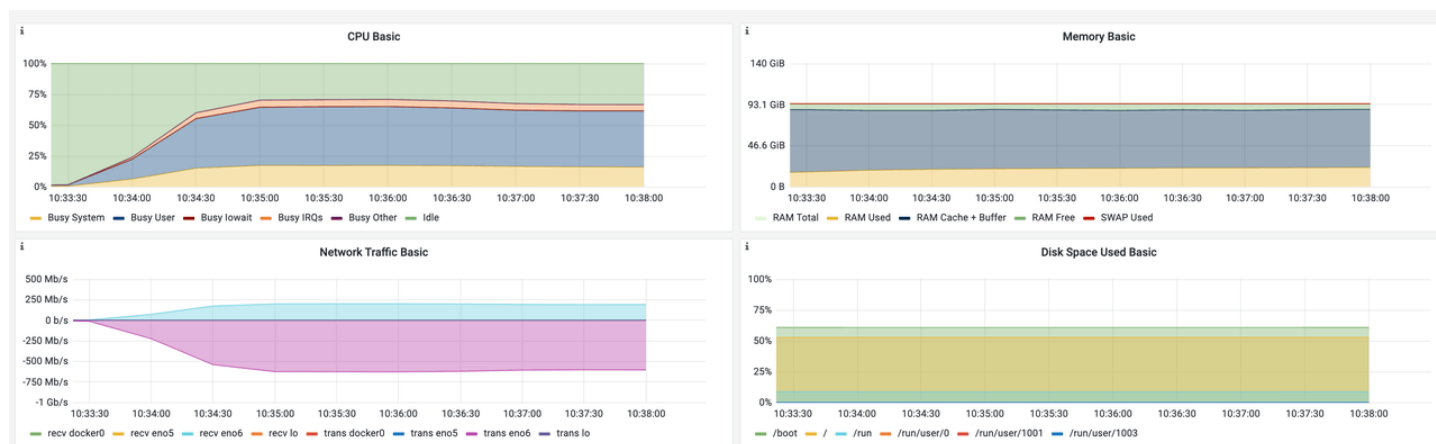
结论：在「1 Proxy 节点 + 1 存储节点」的场景下，可以看到性能压测瓶颈在 Proxy 所在服务器 CPU。因此扩容存储节点带来的性能提升不明显；本实验中扩容存储节点带来的优势在于各个存储节点的 CPU 利用率下降。

该结论仅供参考，用户可以根据实际业务情况，实际场景进行测试。

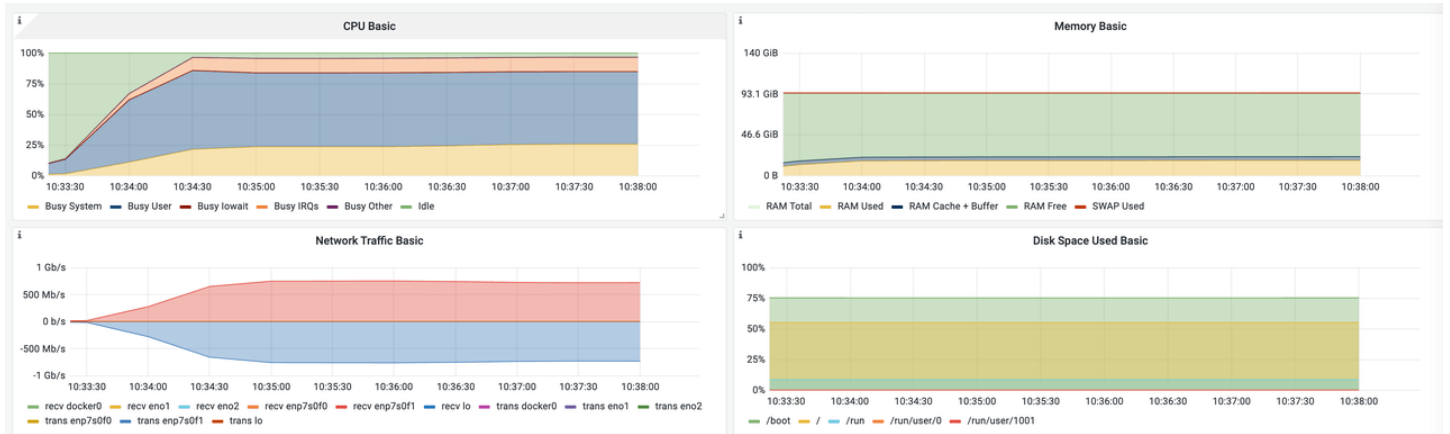
监控信息

BenchmarkSQL ——> Proxy ——> MySQL(1)

■ MySQL

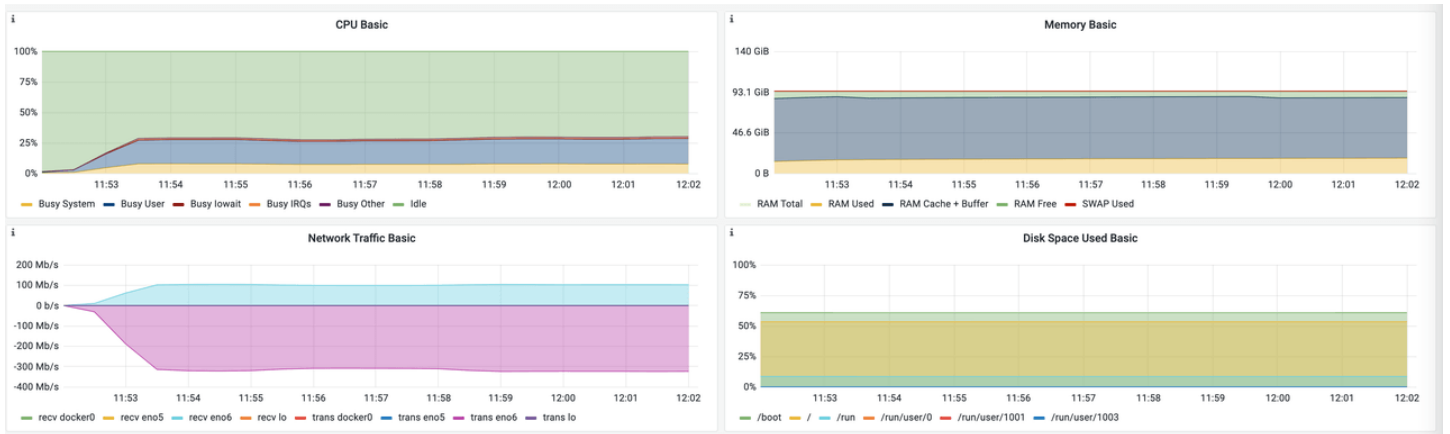


■ Proxy

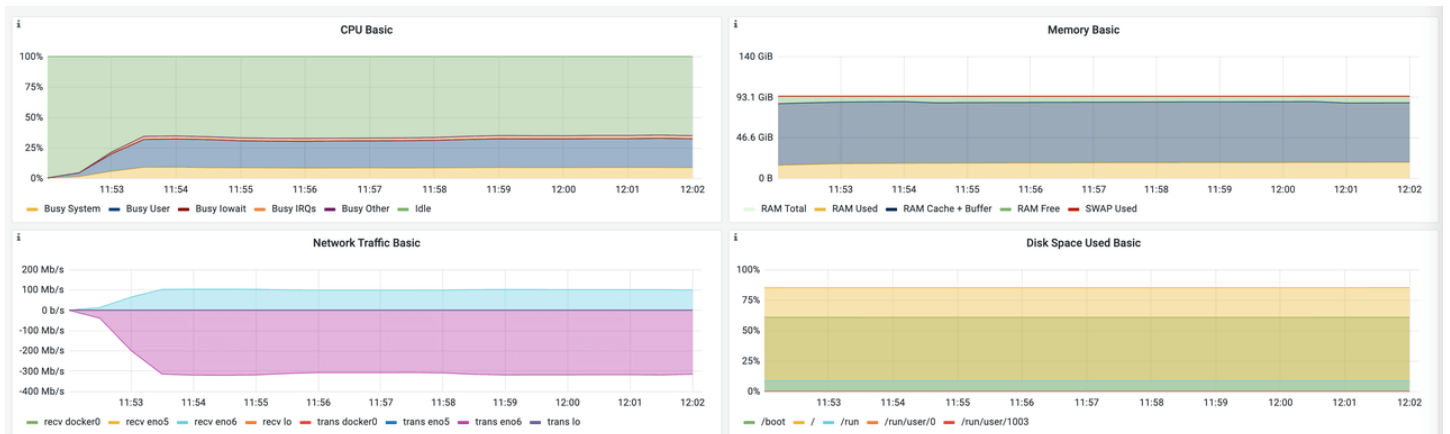


BenchmarkSQL —> Proxy —> MySQL(2)

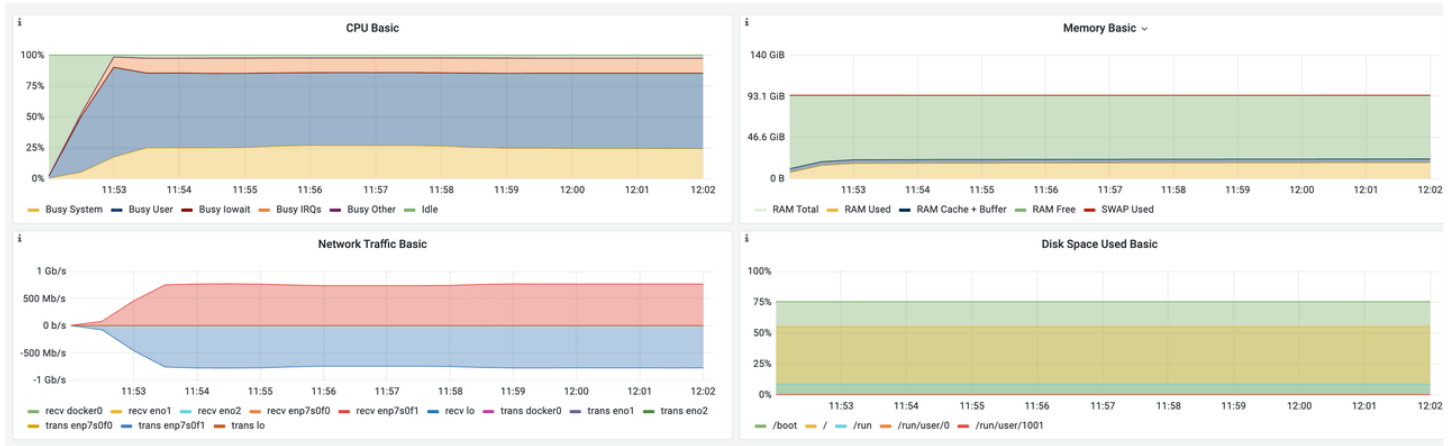
MySQL(1)



MySQL(2)

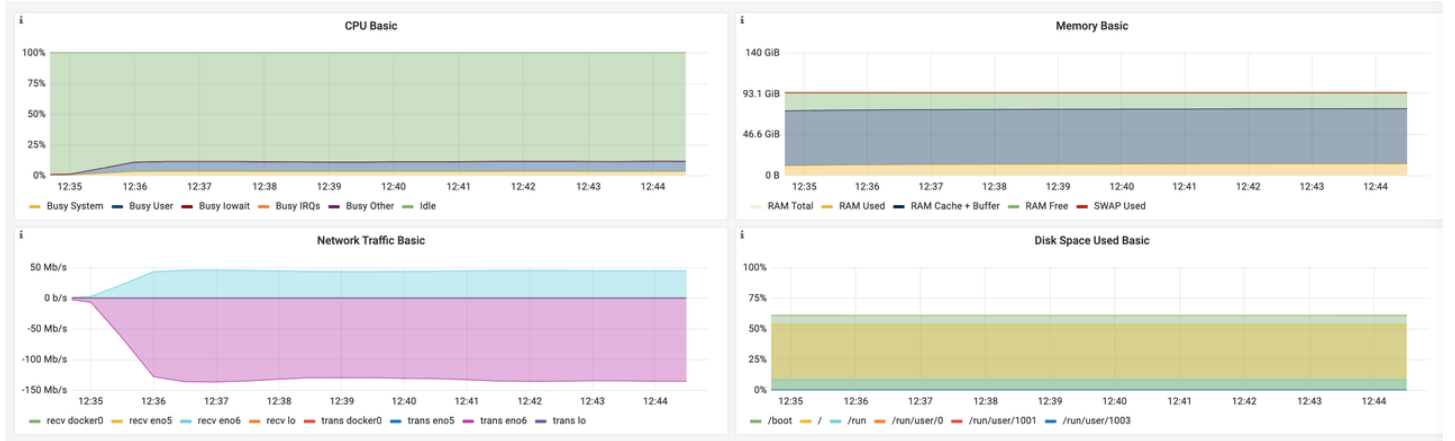


Proxy

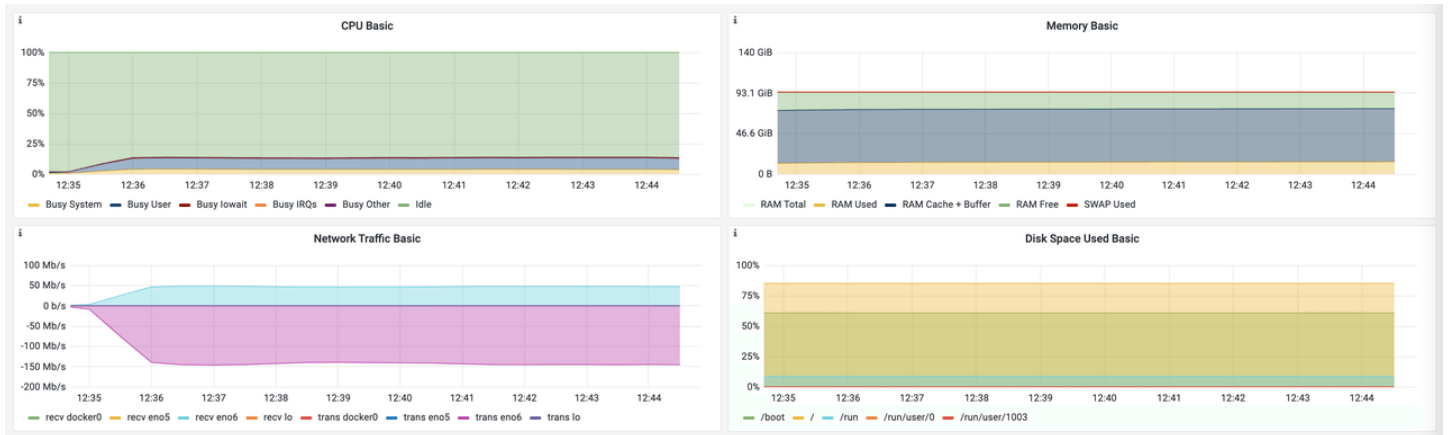


BenchmarkSQL → Proxy → MySQL(4)

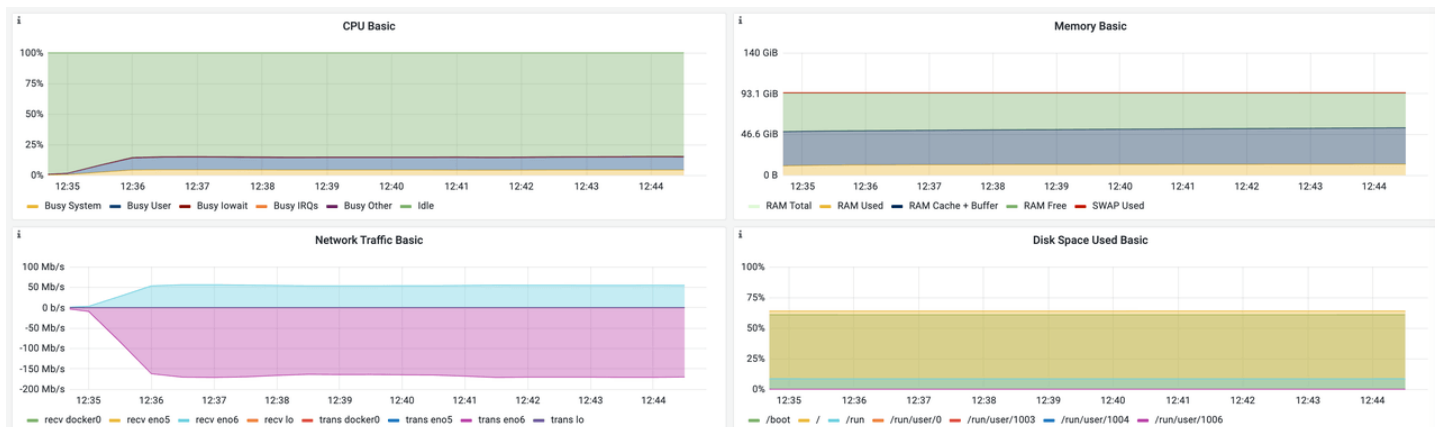
■ MySQL(1)



■ MySQL(2)



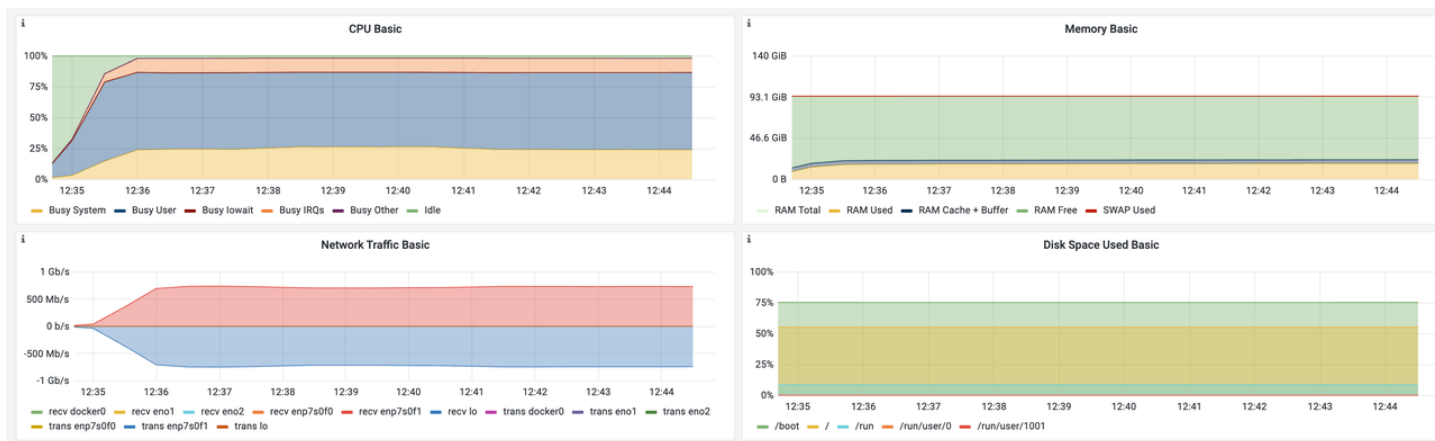
■ MySQL(3)



MySQL(4)



Proxy



名词解释

warehouses: 仓库数，决定数据量。

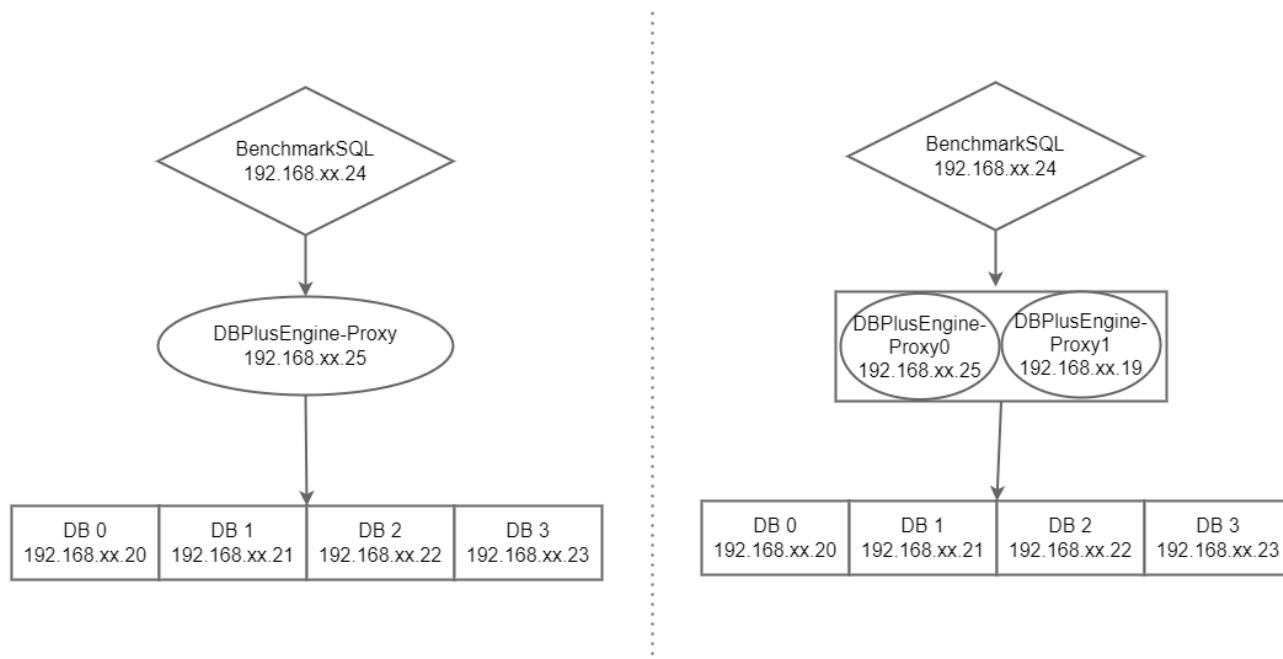
terminals: 线程数。

runMins: 测试执行时长。

tpmC: 每分钟处理事务数。

5.1.3 测试方案三：计算节点扩容

使用 BenchmarkSQL 工具对 4 存储节点 (1 计算节点、2 计算节点) 场景进行压测。



测试目的

在相同数据量下，使用 BenchmarkSQL 压测模型分别对 1/2 计算节点 DBPlusEngine-Proxy 在 4 存储节点场景下进行对比，通过压测结果分析，使用户更加了解计算节点扩容的优势。

测试工具

BenchmarkSQL

测试环境

应用	IP 地址	端口	版本
DBPlusEngine-Proxy	192.168.xx.25	3307	1.2
DBPlusEngine-Proxy	192.168.xx.19	3307	1.2
MySQL	192.168.xx.20	13306	8.0.29
MySQL	192.168.xx.21	13306	8.0.29
MySQL	192.168.xx.22	13306	8.0.29
MySQL	192.168.xx.23	13306	8.0.29

服务器配置

条目	配置
CPU	48 C
内存	96 G
硬盘	SSD 820 G
JDK	17.0.2

测试步骤

1. 创建测试表。

```
mysql -utest -h192.168.xx.20 -P13306 -p
```

```
DROP DATABASE IF EXISTS test_tpcc;
CREATE DATABASE IF NOT EXISTS test_tpcc;
```

```
mysql -utest -h192.168.xx.21 -P13306 -p
```

```
DROP DATABASE IF EXISTS test_tpcc;
CREATE DATABASE IF NOT EXISTS test_tpcc;
```

```
mysql -utest -h192.168.xx.22 -P13306 -p
```

```
DROP DATABASE IF EXISTS test_tpcc;
CREATE DATABASE IF NOT EXISTS test_tpcc;
```

```
mysql -utest -h192.168.xx.23 -P13306 -p
```

```
DROP DATABASE IF EXISTS test_tpcc;
CREATE DATABASE IF NOT EXISTS test_tpcc;
```

2. 安装测试工具。

```
wget https://udomain.dl.sourceforge.net/project/benchmarksql/benchmarksql-5.0.zip
yum -y install ant
unzip benchmarksql-5.0.zip -d /usr/local
```

```
cd /usr/local/benchmarksql-5.0
ant
cp mysql-connector-java-8.0.24.jar benchmarksql-5.0/run/lib
```

创建 props.proxy 文件。

```
db=postgres
driver=com.mysql.jdbc.Driver
conn=jdbc:mysql://192.168.xx.25:3307/sharding_db?useSSL=false&useServerPrepStmts=true&cachePrepStmts=true&
prepStmtCacheSize=8192&prepStmtCacheSqlLimit=8000
user=root
password=root
warehouses=200
loadWorkers=80
terminals=200
runTxnsPerTerminal=0
runMins=10
limitTxnsPerMin=0
terminalWarehouseFixed=true
newOrderWeight=45
paymentWeight=43
orderStatusWeight=4
deliveryWeight=4
stockLevelWeight=4
resultDirectory=my_result_%tY-%tm-%td_%tH%M%S
shardingNumber=4
connBalance=192.168.xx.25:3333,192.168.xx.19:3333
```

3. YAML

```
rules:
- !SHARDING
  bindingTables:
  - bmsql_warehouse, bmsql_customer
  - bmsql_stock, bmsql_district, bmsql_order_line
  defaultDatabaseStrategy:
    none:
  defaultTableStrategy:
    none:
  keyGenerators:
    snowflake:
      type: SNOWFLAKE
  tables:
    bmsql_config:
      actualDataNodes: ds_0.bmsql_config

    bmsql_warehouse:
      actualDataNodes: ds_${0..3}.bmsql_warehouse
      databaseStrategy:
        standard:
          shardingColumn: w_id
          shardingAlgorithmName: mod_4

    bmsql_district:
      actualDataNodes: ds_${0..3}.bmsql_district
      databaseStrategy:
        standard:
          shardingColumn: d_w_id
          shardingAlgorithmName: mod_4

    bmsql_customer:
      actualDataNodes: ds_${0..3}.bmsql_customer
      databaseStrategy:
        standard:
```



```
shardingColumn: c_w_id
shardingAlgorithmName: mod_4
```

```
bmsql_item:
actualDataNodes: ds_${0..3}.bmsql_item
databaseStrategy:
standard:
shardingColumn: i_id
shardingAlgorithmName: mod_4
```

```
bmsql_history:
actualDataNodes: ds_${0..3}.bmsql_history
databaseStrategy:
standard:
shardingColumn: h_w_id
shardingAlgorithmName: mod_4
```

```
bmsql_oorder:
actualDataNodes: ds_${0..3}.bmsql_oorder
databaseStrategy:
standard:
shardingColumn: o_w_id
shardingAlgorithmName: mod_4
```

```
bmsql_stock:
actualDataNodes: ds_${0..3}.bmsql_stock
databaseStrategy:
standard:
shardingColumn: s_w_id
shardingAlgorithmName: mod_4
```

```
bmsql_new_order:
actualDataNodes: ds_${0..3}.bmsql_new_order
databaseStrategy:
standard:
shardingColumn: no_w_id
shardingAlgorithmName: mod_4
```

```
bmsql_order_line:
actualDataNodes: ds_${0..3}.bmsql_order_line
databaseStrategy:
standard:
shardingColumn: ol_w_id
shardingAlgorithmName: mod_4
```

```
shardingAlgorithms:
mod_4:
type: MOD
props:
sharding-count: 4
```

4. 数据准备。

数据库初始化，通过 Proxy 建表并插入数据。

```
cd /usr/local/benchmarksql/run
./runDatabaseDestroy.sh props.proxy
./runDatabaseBuild.sh props.proxy
```

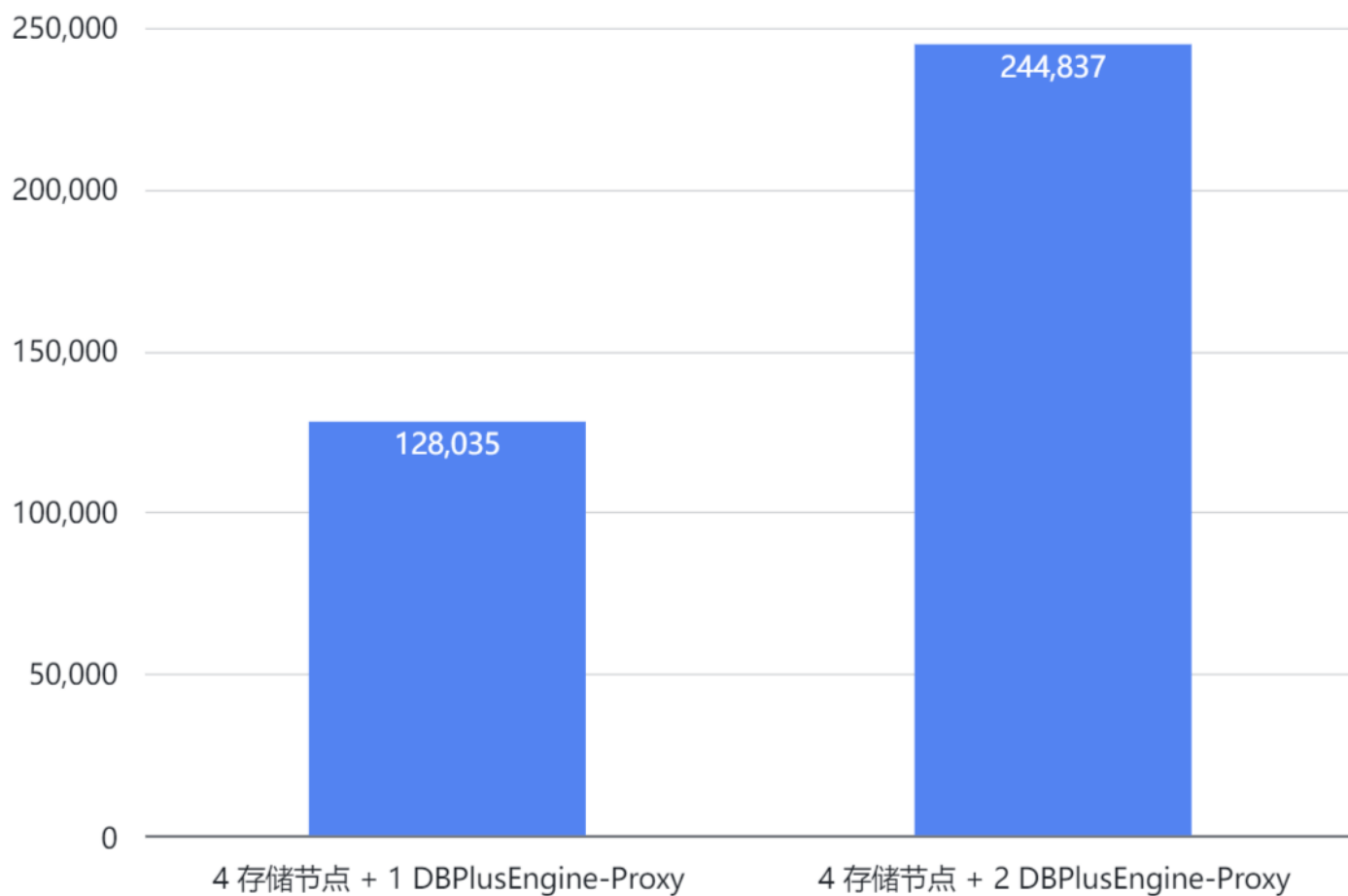
5. 压测阶段。

```
./runBenchmark.sh props.proxy
```

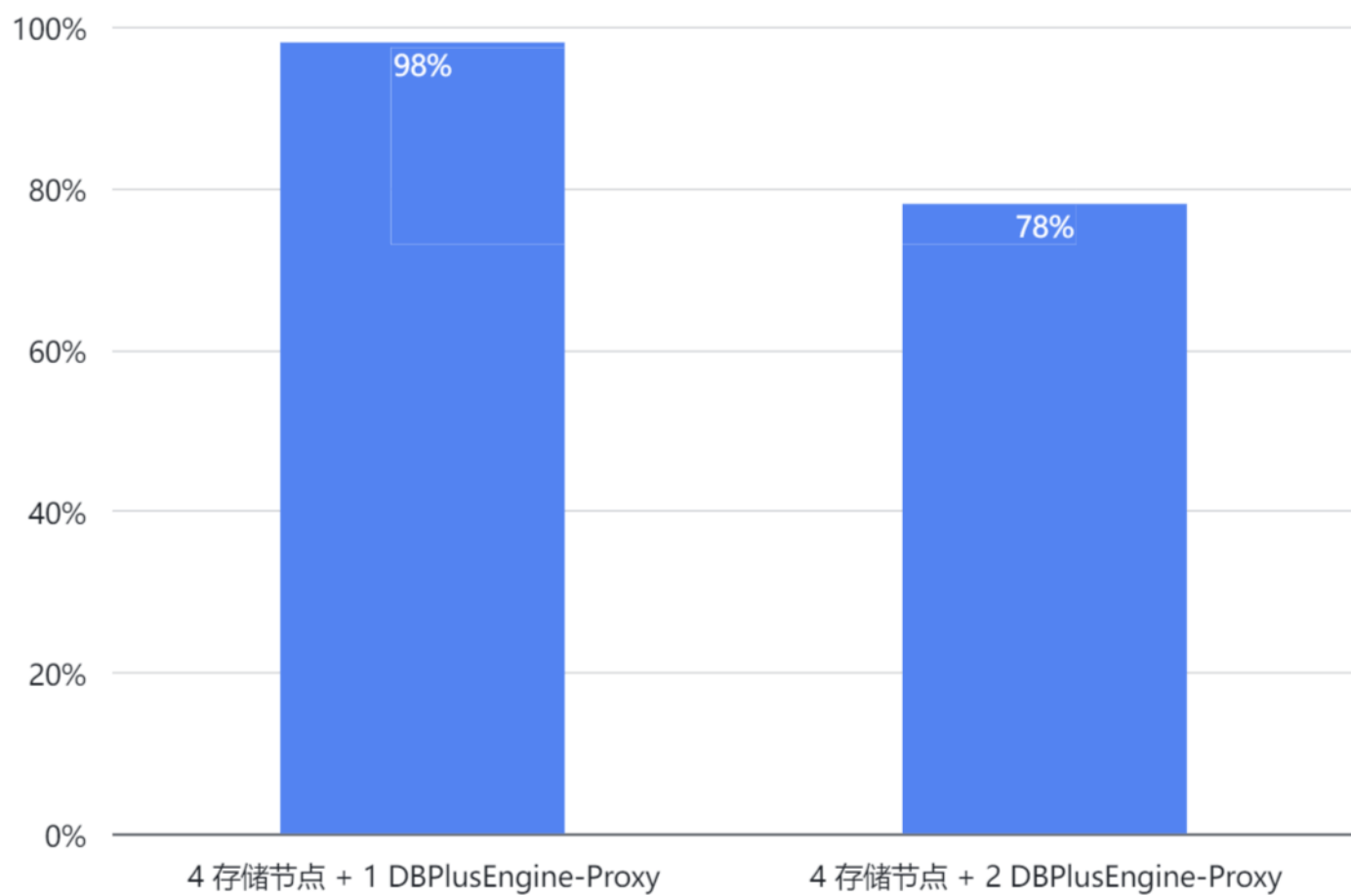
测试结果

测试对象	tpmC	存储节点 AVG(CPU Load)	计算节点 AVG(CPU Load)
4 存储节点 + 1 DB PlusEngine-Proxy	128,035	12.5%	98%
4 存储节点 + 2 DB PlusEngine-Proxy	244,837	28%	78%

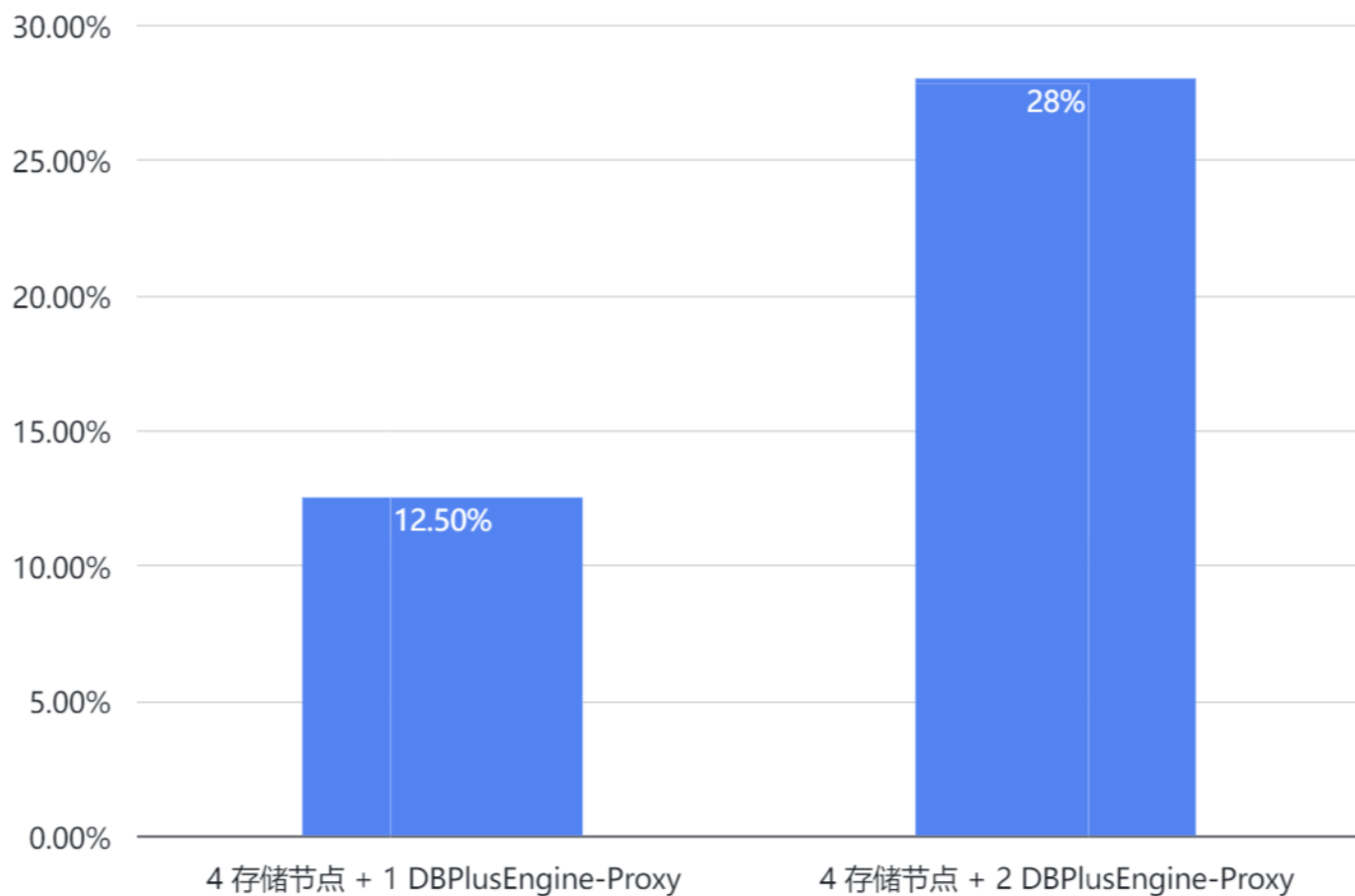
tpmC



计算节点 AVG(CPU Load)



存储节点 AVG(CPU Load)

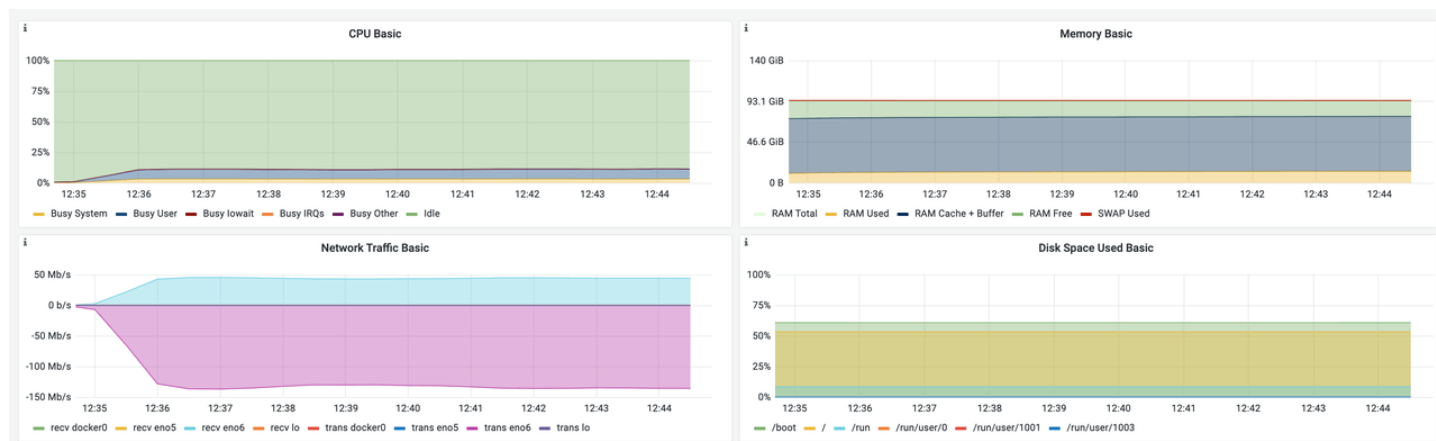


结论：在此场景下扩容计算节点可以近线性提升整体吞吐量。

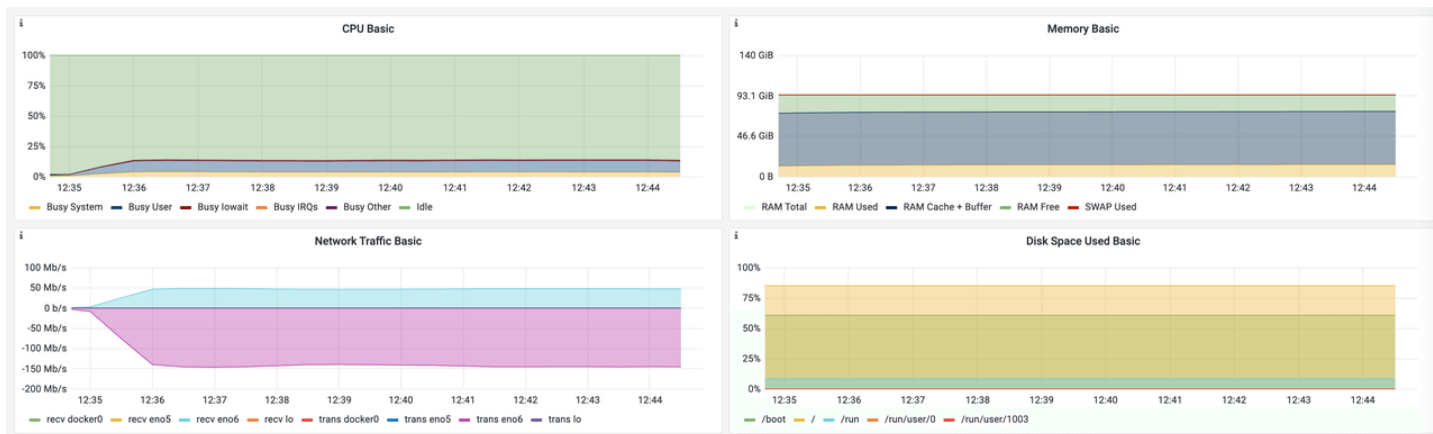
监控信息

BenchmarkSQL ——> Proxy ——> MySQL(4)

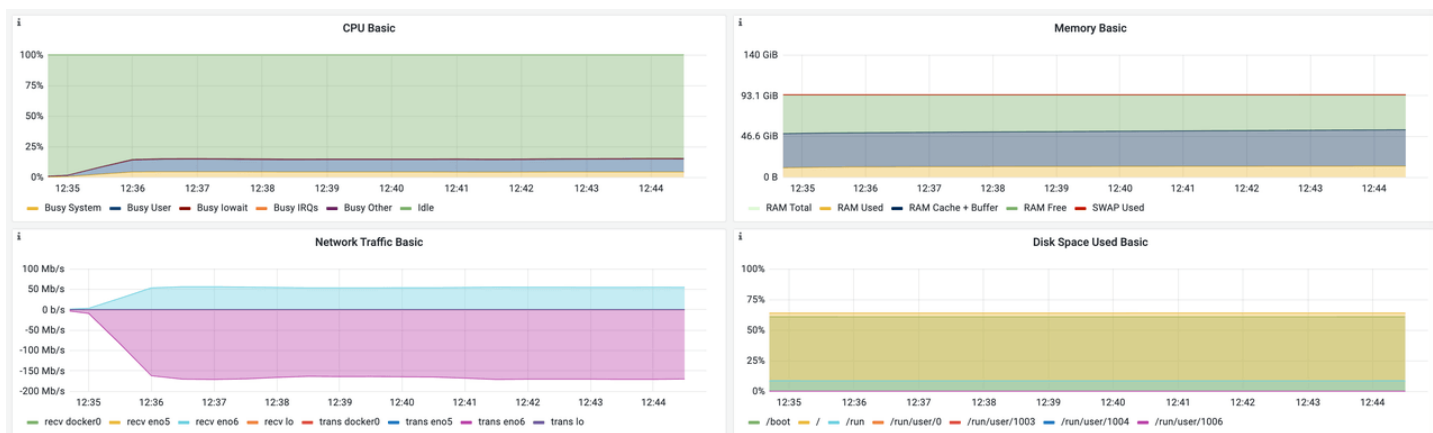
■ MySQL(1)



■ MySQL(2)



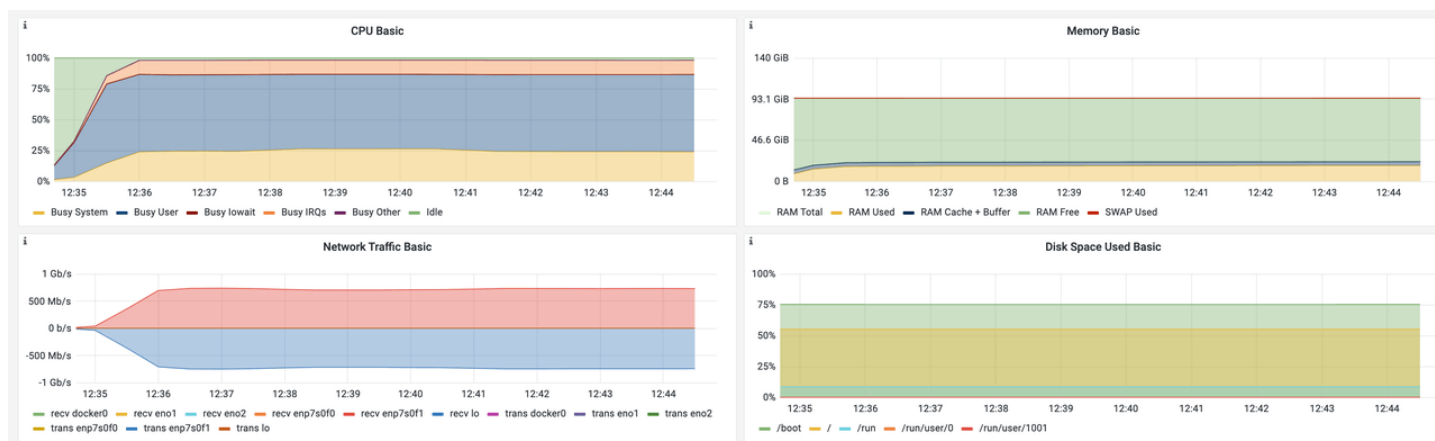
■ MySQL(3)



■ MySQL(4)



■ Proxy

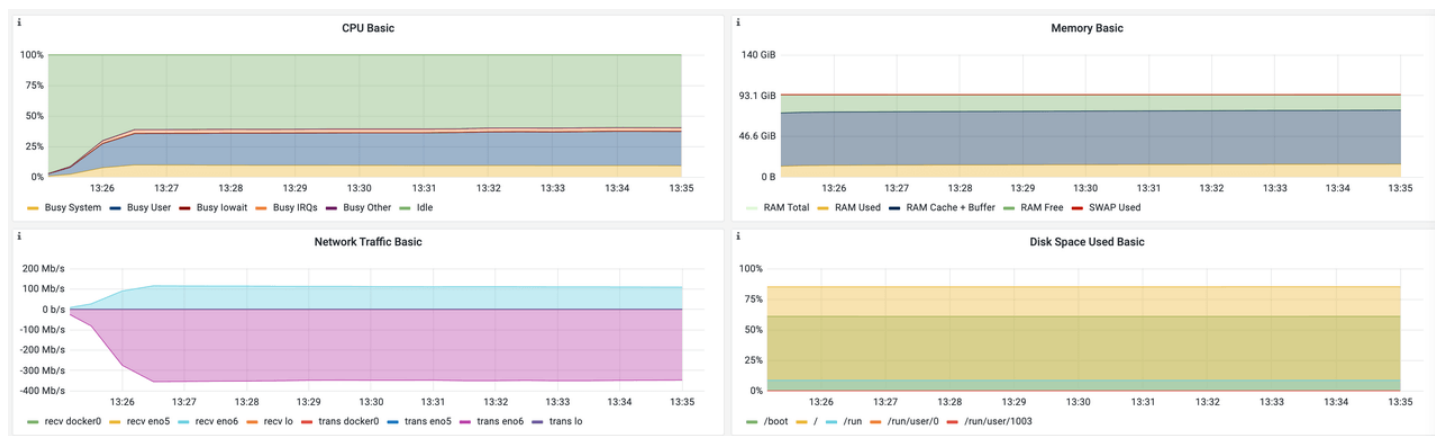


BenchmarkSQL —> Proxy(2) —> MySQL(4)

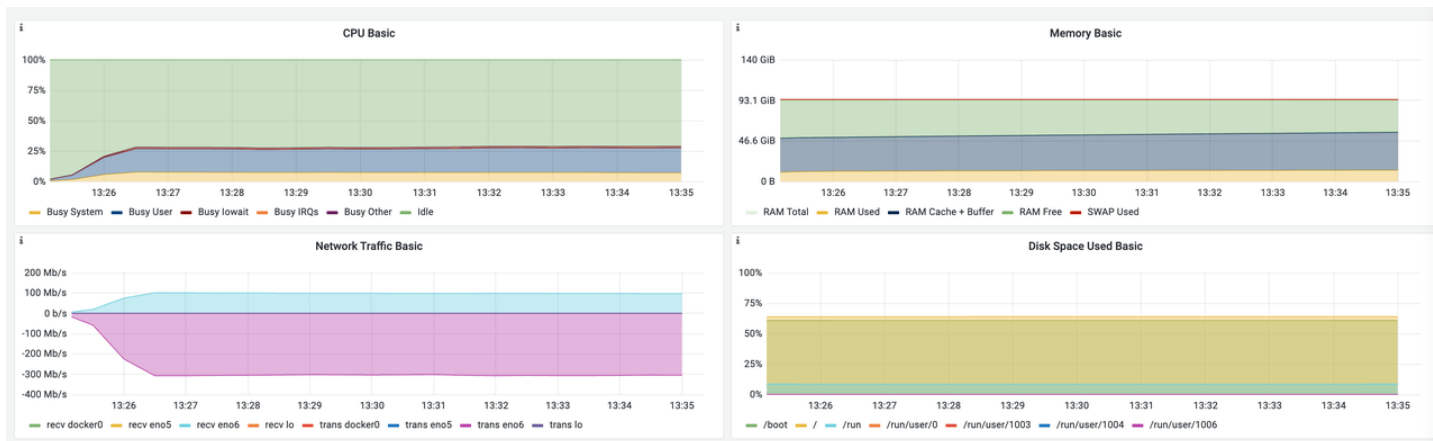
MySQL(1)



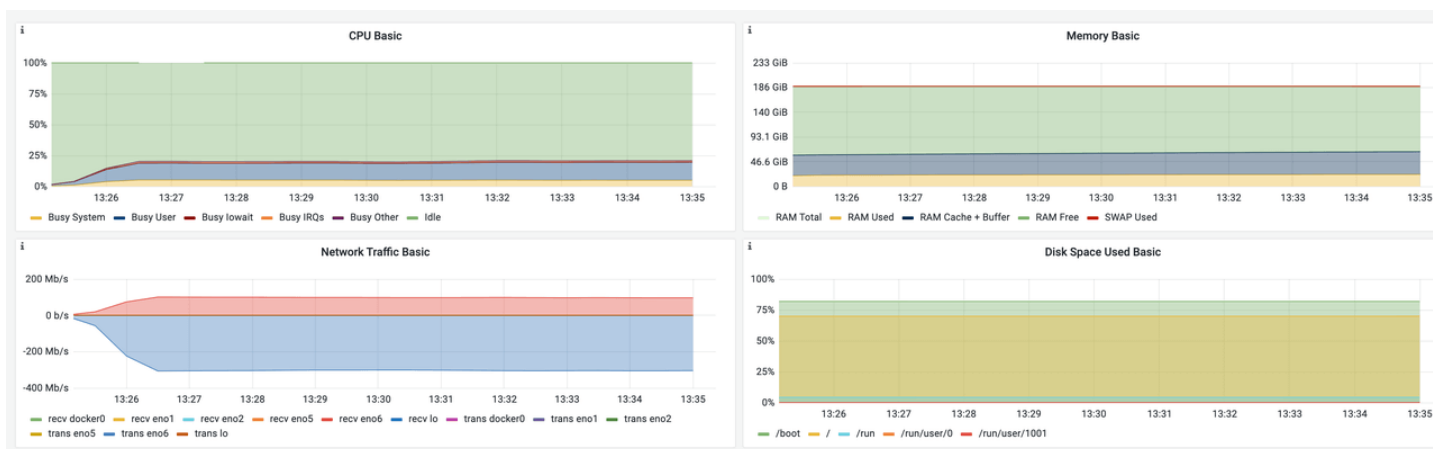
MySQL(2)



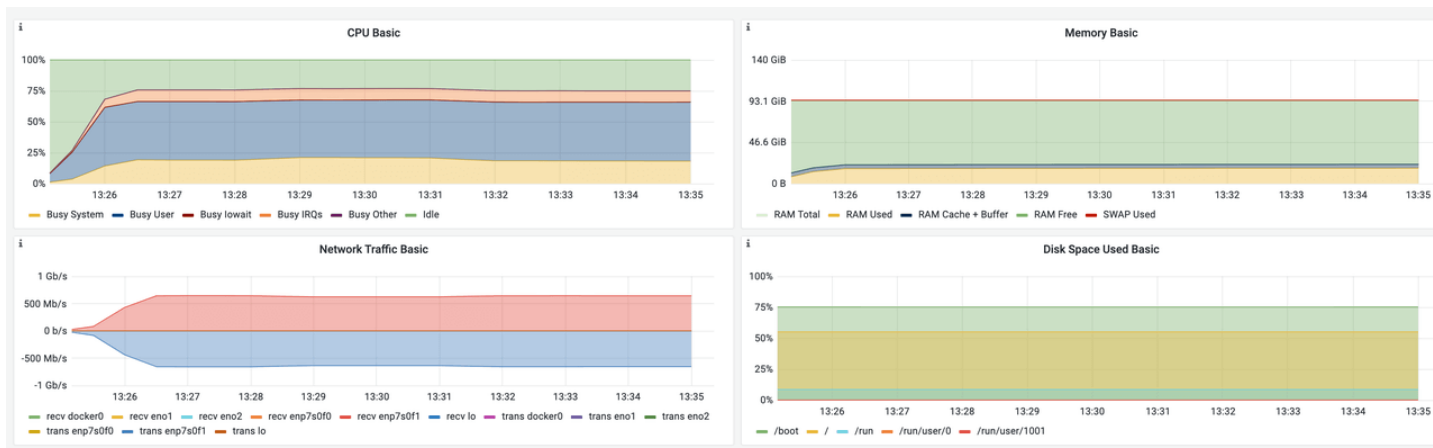
MySQL(3)



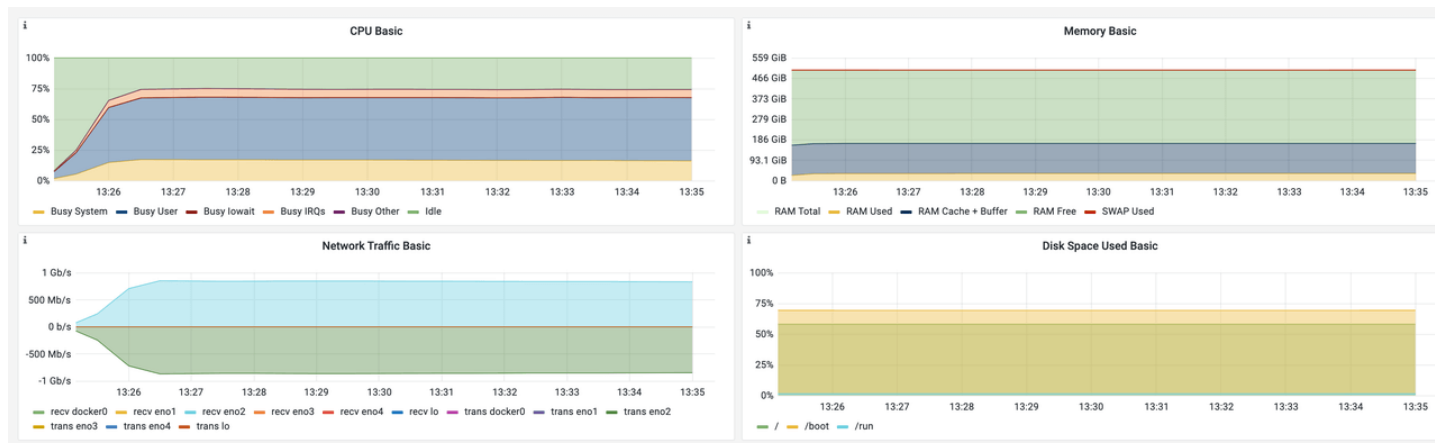
■ MySQL(4)



■ Proxy(1)



■ Proxy(2)



名词解释

warehouses: 仓库数，决定数据量。

terminals: 线程数。

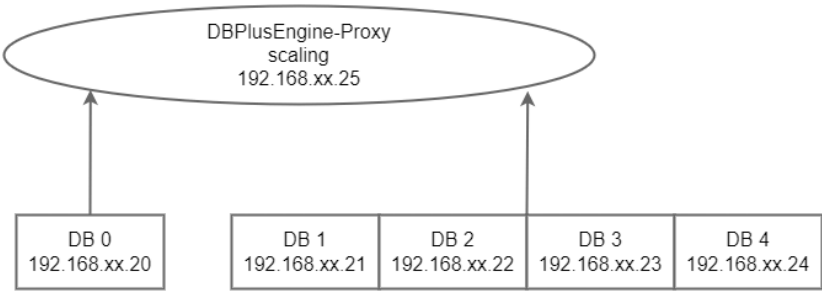
runMins: 测试执行时长。

tpmC: 每分钟处理事务数。

5.2 数据集成性能测试

5.2.1 测试方案一：数据导出

数据迁移，将单库单表 36G 的数据量 (1 亿 4 千万行) 迁移到目标源上。



测试目的

当数据数据达到一定量级后，数据库的性能会显著下降，数据库的读写操作后会随之受到影响。在不影响业务场景操作的情况下，我们通过使用 DBPlusEngine-Proxy 的 scaling 功能将原数据迁移到多个目标库中。

测试工具

Sysbench

测试环境

应用	IP 地址	端口
DBPlusEngine-Proxy	192.168.xx.25	3307
MySQL	192.168.xx.20	13306
MySQL	192.168.xx.21	13306
MySQL	192.168.xx.22	13306
MySQL	192.168.xx.23	13306
MySQL	192.168.xx.24	13306

服务器配置

条目	配置
CPU	48 C
内存	96 G
硬盘	SSD 820 G
JDK	17.0.2

测试步骤

1. 创建测试表。

```
mysql -utest -h192.168.xx.20 -P13306 -p
```

```
DROP DATABASE IF EXISTS migration_ds_0;
```

```
CREATE DATABASE IF NOT EXISTS migration_ds_0;
```

```
mysql -utest -h192.168.xx.21 -P13306 -p
```

```
DROP DATABASE IF EXISTS migration_ds_1;
```

```
CREATE DATABASE IF NOT EXISTS migration_ds_1;
```

```
mysql -utest -h192.168.xx.22 -P13306 -p
```

```
DROP DATABASE IF EXISTS migration_ds_2;
```

```
CREATE DATABASE IF NOT EXISTS migration_ds_2;
```

```
mysql -utest -h192.168.xx.23 -P13306 -p
```

```
DROP DATABASE IF EXISTS migration_ds_3;
```

```
CREATE DATABASE IF NOT EXISTS migration_ds_3;
```

```
mysql -utest -h192.168.xx.24 -P13306 -p
```

```
DROP DATABASE IF EXISTS migration_ds_4;
```

```
CREATE DATABASE IF NOT EXISTS migration_ds_4;
```

```
mysql -uroot -h192.168.xx.25 -P3307 -proot
```

```
DROP DATABASE IF EXISTS sphereex_demo;
```

```
CREATE DATABASE IF NOT EXISTS sphereex_demo;
```

2. 安装测试工具。

```
tar -xvzf sysbench-1.0.20.tar.gz -C /usr/local  
cd /usr/local/sysbench-1.0.20  
./autogen.sh  
./configure  
make && make install
```

3. 数据准备。

使用 Sysbench 工具向源端数据库插入 140,000,000 的数据。

```
sysbench oltp_read_only --mysql-host='192.168.xx.20' --mysql-port=13306 --mysql-user=test --mysql-password='test' --mysql-db=migration_ds_0 --tables=1 --table-size=140000000 --report-interval=10 --time=3600 --threads=256 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off --auto_inc=off cleanup
```

```
sysbench oltp_read_only --mysql-host='192.168.xx.20' --mysql-port=13306 --mysql-user=test --mysql-password='test' --mysql-db=migration_ds_0 --tables=1 --table-size=140000000 --report-interval=10 --time=3600 --threads=256 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off --auto_inc=off prepare
```

4. 迁移过程。

通过 Proxy，新建逻辑数据库并配置好资源和规则。

```
mysql -uroot -h 192.168.xx.25 -P3307 -proot
```

```
CREATE DATABASE sphereex_demo;
```

```
USE sphereex_demo;
```

```
REGISTER STORAGE UNIT ds_0 (
  URL="jdbc:mysql://192.168.xx.20:13306/migration_ds_0?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="test",
  PROPERTIES("maximumPoolSize"=500,"idleTimeout"=60000)
);
```

添加目标源。

```
REGISTER STORAGE UNIT ds_1 (
  URL="jdbc:mysql://192.168.xx.21:13306/migration_ds_1?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="test",
  PROPERTIES("maximumPoolSize"=50,"idleTimeout"=60000)
), ds_2 (
  URL="jdbc:mysql://192.168.xx.22:13306/migration_ds_2?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="test",
  PROPERTIES("maximumPoolSize"=50,"idleTimeout"=60000)
), ds_3 (
  URL="jdbc:mysql://192.168.xx.23:13306/migration_ds_3?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="test",
  PROPERTIES("maximumPoolSize"=50,"idleTimeout"=60000)
), ds_4 (
  URL="jdbc:mysql://192.168.xx.24:13306/migration_ds_4?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="test",
  PROPERTIES("maximumPoolSize"=50,"idleTimeout"=60000)
);
```

创建规则。

```
CREATE SHARDING TABLE RULE sbtest1(
  STORAGE_UNITS(ds_1,ds_2,ds_3,ds_4),
  SHARDING_COLUMN=id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"=4)),
  KEY_GENERATE_STRATEGY(COLUMN=id,TYPE(NAME="snowflake"))
);
```

添加迁移配置。

```
ALTER MIGRATION RULE (
  READ(
    WORKER_THREAD=40,
    BATCH_SIZE=1000,
```

```
SHARDING_SIZE=7000000
),
WRITE(
  WORKER_THREAD=40,
  BATCH_SIZE=1000
)
);
```

配置源端资源。

```
REGISTER MIGRATION SOURCE STORAGE UNIT ds_0 (
  URL="jdbc:mysql://192.168.xx.20:13306/migration_ds_0?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="test",
  PROPERTIES("maximumPoolSize"=50,"idleTimeout"="60000")
);
```

启动执行任务。

```
MIGRATE TABLE ds_0.sbtest1 INTO sbtest1;
```

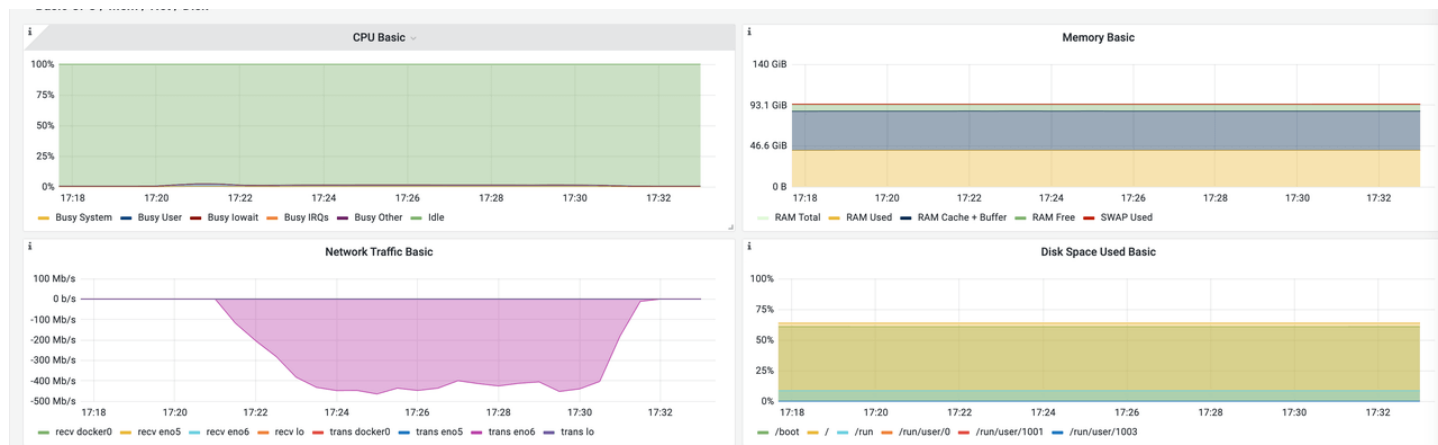
测试结果

整体迁移时间 10 min 49 s。

测试结论：在此配置下计算节点先达到瓶颈。

监控信息

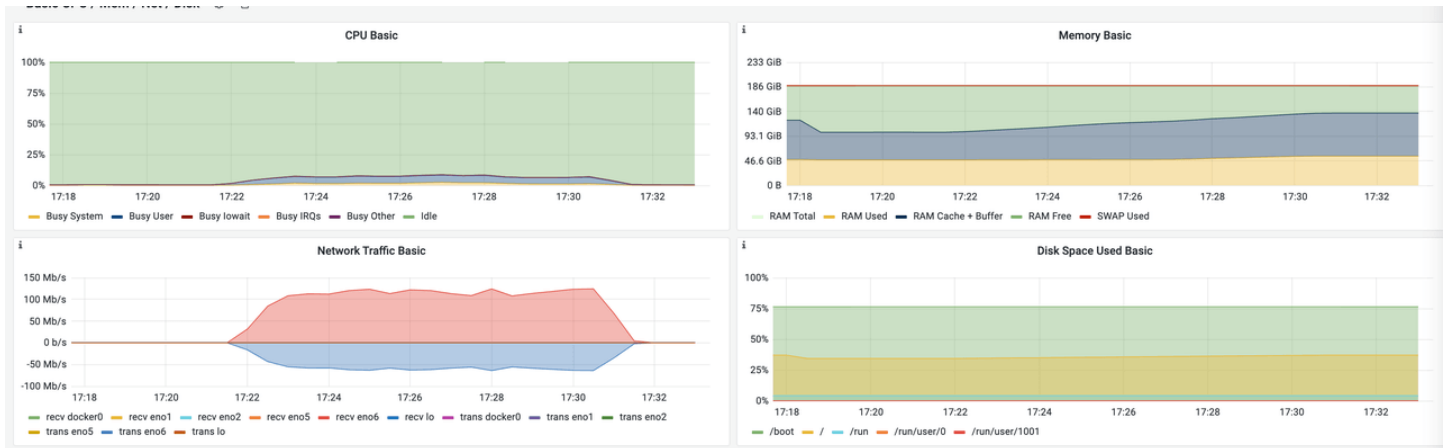
Source DB



DBPlusEngine-Proxy



One of target DB



名词解释

id: 迁移任务标识。

tables: 所迁移的表名称。

job_item_count: 迁移作业分片数。

active: 迁移任务是否进行, false 代表禁用不再运行。

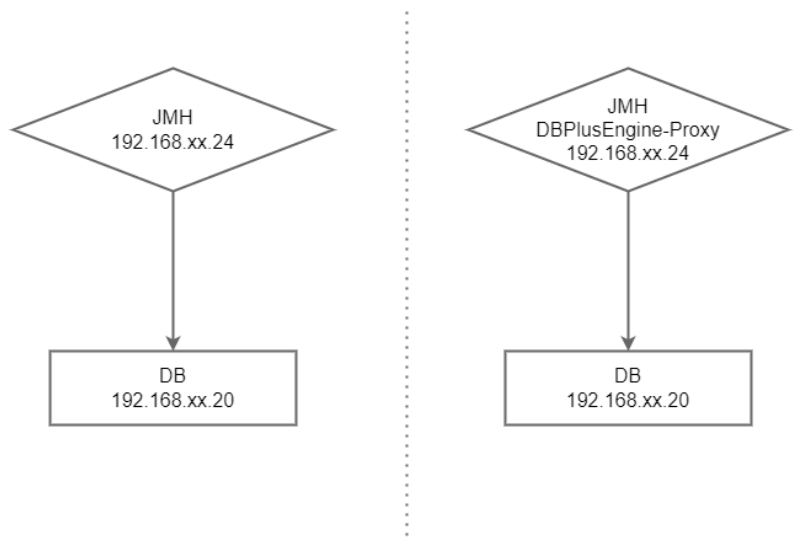
create_time: 迁移任务开始时间。

stop_time: 迁移任务结束时间。

5.2.2 测试方案二：数据导入

使用 JMH 结合 DB Plus Engine-Driver, 向数据库表中插入数据, 插入时长 5 分钟, 并统计插入数据量。

使用 JMH 直接向 MySQL 插入数据, 时长 5 分钟, 并统计插入数据量。



测试目的

- 对比两种导数方式，得到：
- 计算出 DBPlusEngine-Driver 的性能指标
 - 对比 MySQL 原生插入性能损耗

测试工具

JMH

测试环境

应用	IP 地址	端口	版本
DBPlusEngine-Driver	192.168.xx.24	3307	1.2
MySQL	192.168.xx.20	13306	8.0.29

服务器配置

条目	配置
CPU	48 C
内存	96 G
硬盘	SSD 820 G
JDK	17.0.2

主要参数

- --Dtables=1，表数量
- -f 1，执行次数
- -r 300，执行时间
- -t 128，使用线程数
- -w 0，预热次数

测试步骤

1. 创建测试库。

```
mysql -utest -h192.168.xx.20 -P13306 -p
```

```
DROP SCHEMA IF EXISTS insert_db;
```

```
CREATE TABLE `sbtest1` (  
  `id` int unsigned NOT NULL AUTO_INCREMENT,  
  `k` int NOT NULL DEFAULT '0',  
  `c` char(255) NOT NULL DEFAULT '',  
  `pad` char(60) NOT NULL DEFAULT '',  
  PRIMARY KEY (`id`),  
  KEY `k_1` (`k`)  
) ENGINE=InnoDB AUTO_INCREMENT=1;
```

2. 安装测试工具。

提示：用户需要提前配置 JAVA 及 MAVEN 环境，本示例中使用 openjdk 17.0.2、Apache Maven 3.6.3。

DBPlusEngine-Driver 构建。

用户可自行编写基于 JMH 框架的测试用例，也可用以下代码编译出 JMH 的测试用例集：

```
git clone https://github.com/SphereEx-QE/database-jmh.git  
cd database-jmh/jmh-shardingsphere5  
mvn clean package && cd target
```

testssj.yaml 配置：

```
dataSources:  
  write_ds:  
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource  
    driverClassName: com.mysql.jdbc.Driver  
    jdbcUrl: jdbc:mysql://192.168.xx.20:13306/insert_db?serverTimezone=UTC&useSSL=false&useUnicode=true&  
characterEncoding=UTF-8  
    username: root  
    password: root  
    maximumPoolSize: 300  
    minimumIdle: 1  
  
rules: []  
props:
```

mysql.properties 配置：

```
jdbc-url=jdbc:mysql://192.168.xx.21:13306/pure_mysql?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
username=root
password=root
```

3. 执行测试。

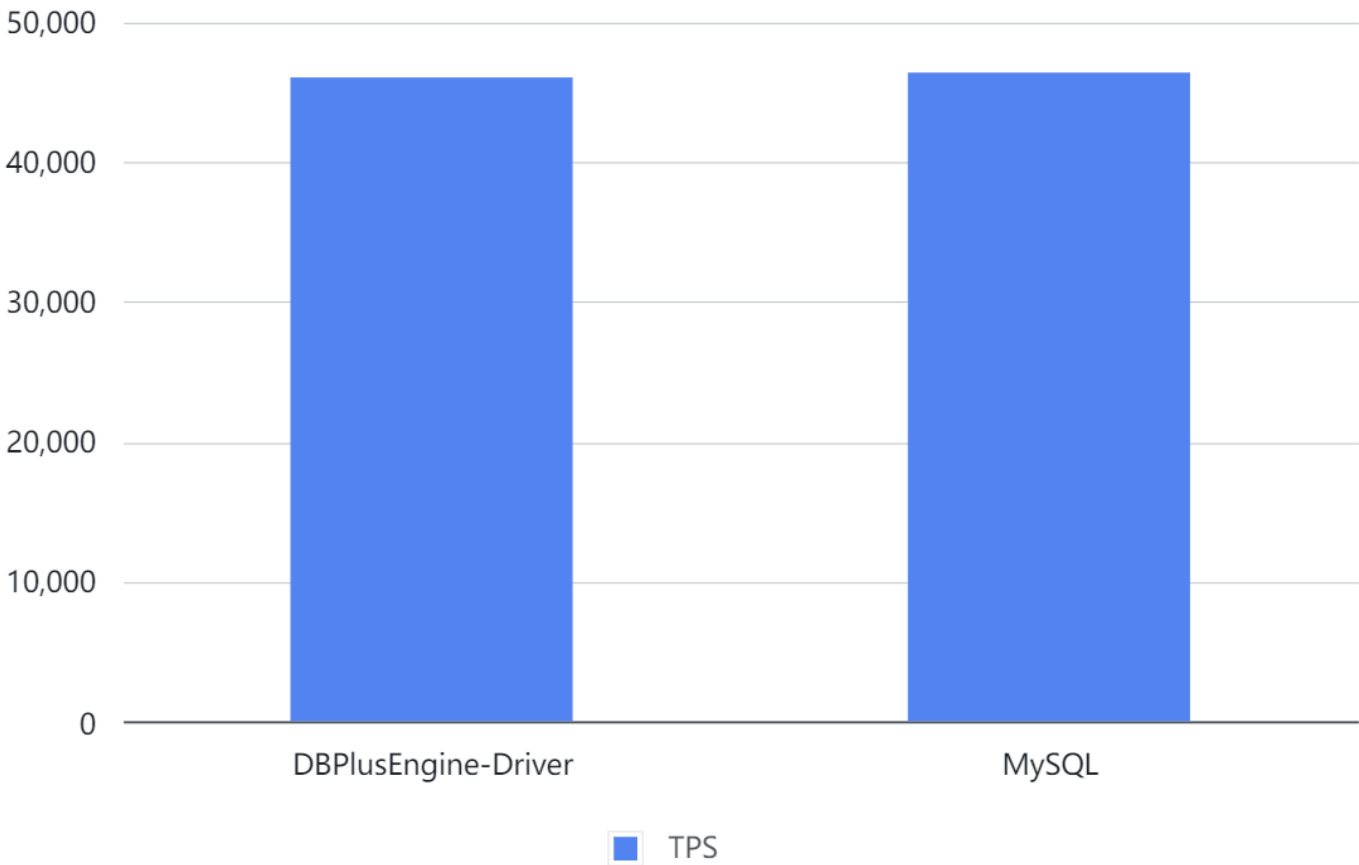
在如下命令中传递 test.yaml 的路径参数以及其他 JMH 的参数 (附录)

```
# 如果想了解 JMH 的参数，可以在下面的命令后加 -help 查看
java -classpath 'dependency/*:jmh-shardingsphere5-1.0-SNAPSHOT.jar' -Dshardingsphere.configurationFile=/root/testssj.yaml -Dtables=1 org.openjdk.jmh.Main "com.sphereex.jmh.shardingsphere5.ShardingSphereInsertOnlyBenchmark" -f 1 -i1 -r200 -t 100 -wi 0
java -classpath 'dependency/*:jmh-jdbc-1.0-SNAPSHOT.jar' -Dconf=/root/mysql.properties -Dtables=1 org.openjdk.jmh.Main "com.sphereex.jmh.jdbc.UnpooledReadOnlyBenchmarkJDBC" -f 1 -i1 -r200 -t 100 -wi 0
```

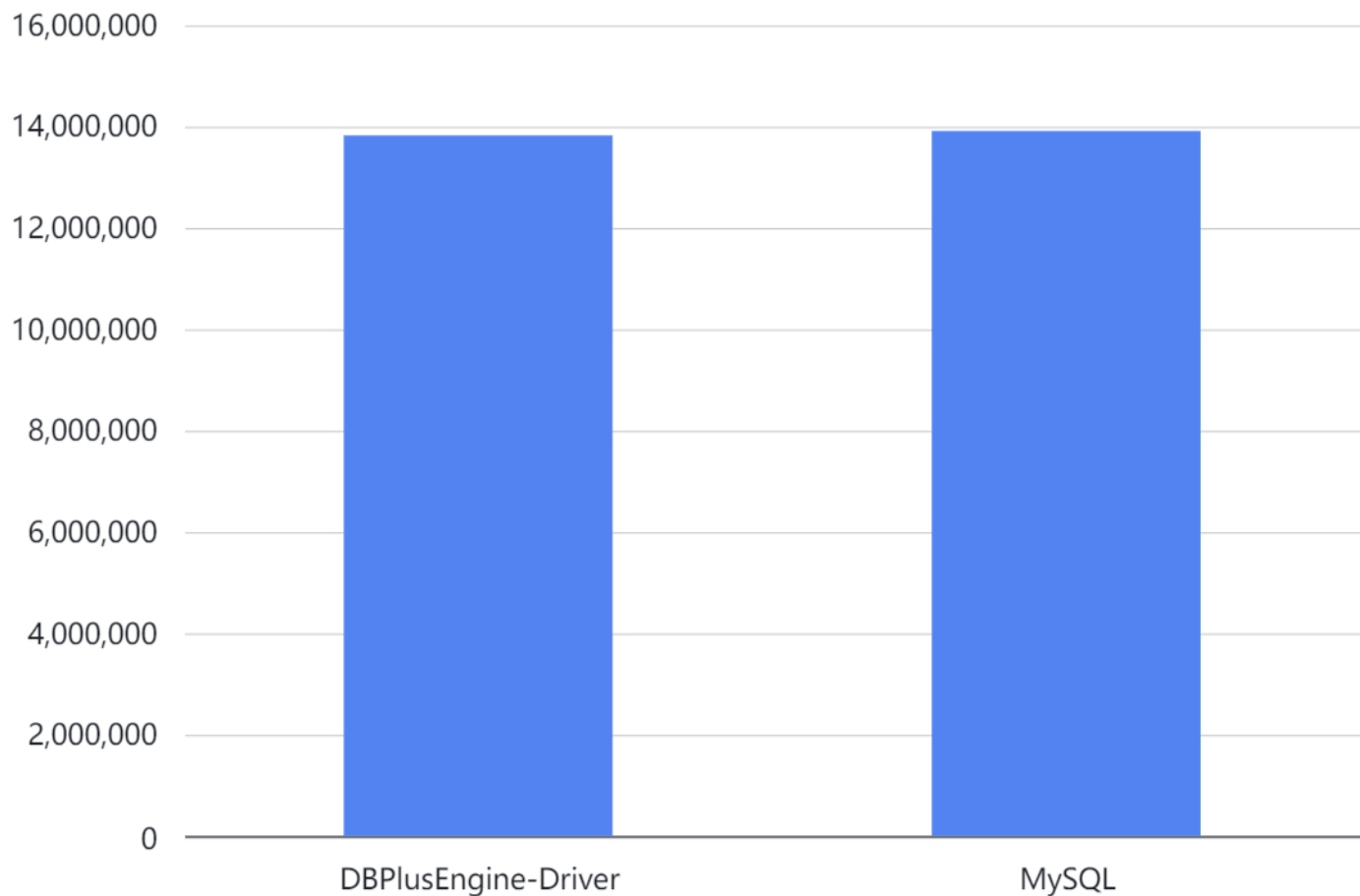
测试结果

测试对象	TPS	Rows Count
DBPlusEngine-Driver	46,055	13,821,323
MySQL	46,402	13,909,701

TPS 对比



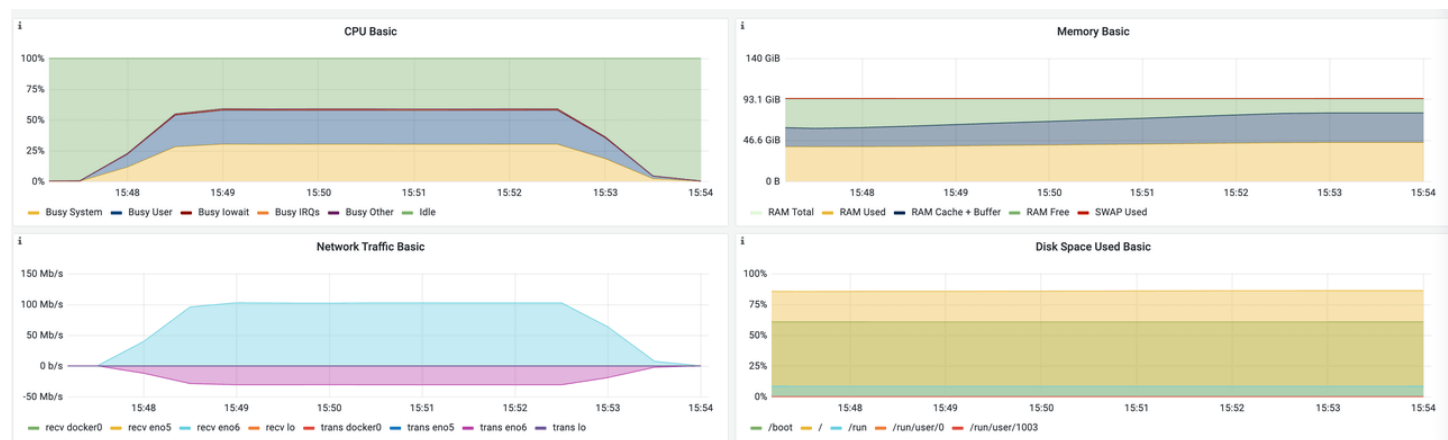
5 min 插入 Rows Count 对比



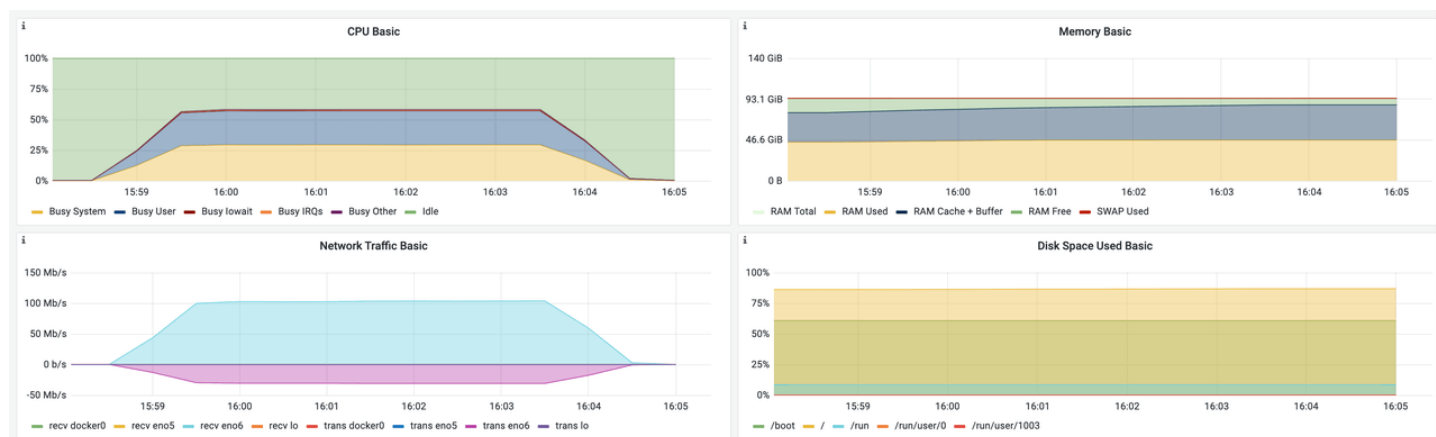
结论：通过对比 DBPlusEngine-Driver & MySQL 插入情况，说明 DBPlusEngine-Driver 几乎无性能损耗。
该结论仅供参考，用户可以根据实际业务情况，实际场景进行测试。

监控信息

DBPlusEngine-Driver



MySQL



名词解释

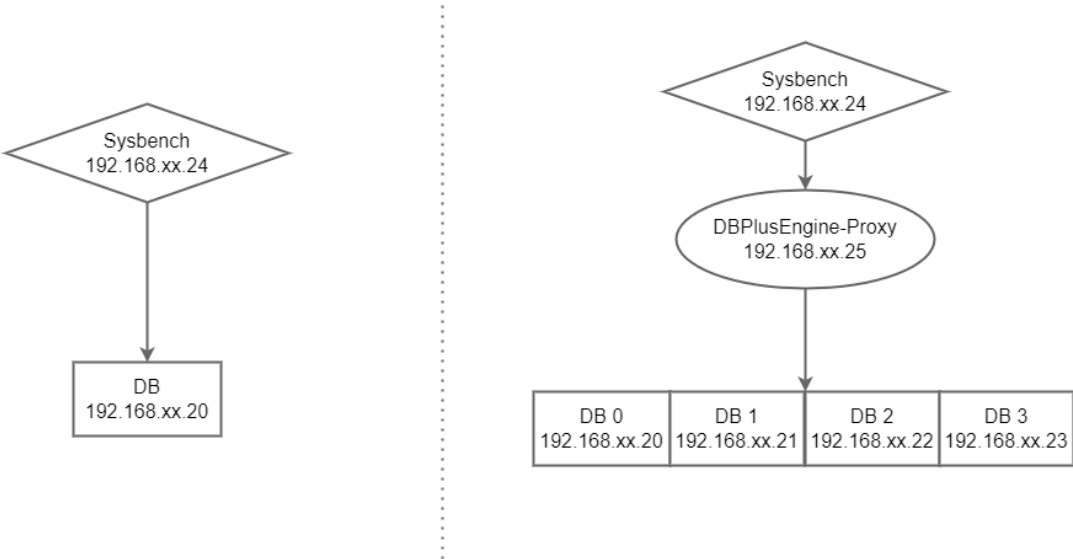
TPS：每秒的事务处理数

5.3 插件功能性能测试

提供通用方法，由用户侧测试。

5.3.1 数据分片

数据分片测试场景。



测试目的

通过 DBPlusEngine-Proxy 的分片规则将数据路由分发到对应的数据库和表中。使用 sysbench 工具执行读写场景压测，从而获取分片功能的性能指标。

测试工具

Sysbench

测试环境

应用	IP 地址	端口	版本
DBPlusEngine-Proxy	192.168.xx.25	3307	1.2
MySQL	192.168.xx.20	13306	8.0.29
MySQL	192.168.xx.21	13306	8.0.29
MySQL	192.168.xx.22	13306	8.0.29
MySQL	192.168.xx.23	13306	8.0.29

服务器配置

条目	配置
CPU	48 C
内存	96 G
硬盘	SSD 820 G
JDK	17.0.2

测试步骤

1. 创建测试库。

```
mysql -utest -h192.168.xx.20 -P13306 -p
```

```
DROP DATABASE IF EXISTS sbtest;
```

```
CREATE DATABASE IF NOT EXISTS sbtest;
```

```
mysql -utest -h192.168.xx.21 -P13306 -p
```

```
DROP DATABASE IF EXISTS sbtest;
```

```
CREATE DATABASE IF NOT EXISTS sbtest;
```

```
mysql -utest -h192.168.xx.22 -P13306 -p
```

```
DROP DATABASE IF EXISTS sbtest;
```

```
CREATE DATABASE IF NOT EXISTS sbtest;
```

```
mysql -utest -h192.168.xx.23 -P13306 -p
```

```
DROP DATABASE IF EXISTS sbtest;
```

```
CREATE DATABASE IF NOT EXISTS sbtest;
```

创建逻辑库。

```
mysql -uroot -h192.168.xx.25 -P3307 -proot
```

```
DROP DATABASE IF EXISTS sharding_db;
```

```
CREATE DATABASE IF NOT EXISTS sharding_db;
```

通过 DistSQL 关联数据源。

```
REGISTER STORAGE UNIT ds_0 (  
  HOST="192.168.xx.20",  
  PORT=13306,  
  DB="sbtest",  
  USER="test",  
  PASSWORD="test"  
) , ds_1 (  
  HOST="192.168.xx.21",  
  PORT=13306,  
  DB="sbtest",  
  USER="test",  
  PASSWORD="test"  
) , ds_2 (  
  HOST="192.168.xx.22",
```

```
PORT=13306,  
DB="sbtest",  
USER="test",  
PASSWORD="test"  
), ds_3 (  
HOST="192.168.xx.23",  
PORT=13306,  
DB="sbtest",  
USER="test",  
PASSWORD="test"  
);
```

通过 DistSQL 创建分片规则。

```
CREATE SHARDING TABLE RULE t_user (  
STORAGE_UNITS(ds_0, ds_1, ds_2, ds_3),  
SHARDING_COLUMN=id, TYPE(NAME="MOD", PROPERTIES("sharding-count"=10))  
);
```

2. 安装测试工具。

```
tar -xzf sysbench-1.0.20.tar.gz -C /usr/local  
cd /usr/local/sysbench-1.0.20  
./autogen.sh  
./configure  
make && make install
```

3. 数据准备。

```
sysbench oltp_read_write --mysql-host='192.168.xx.25' --mysql-port=3307 --mysql-user=root --mysql-password='root' --mysql-db=sharding_db --tables=1 --table-size=140000000 --report-interval=5 --time=600 --threads=128 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform --auto_inc=off cleanup
```

```
sysbench oltp_read_write --mysql-host='192.168.xx.25' --mysql-port=3307 --mysql-user=root --mysql-password='root' --mysql-db=sharding_db --tables=1 --table-size=140000000 --report-interval=5 --time=600 --threads=128 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform --auto_inc=off prepare
```

4. 压测阶段。

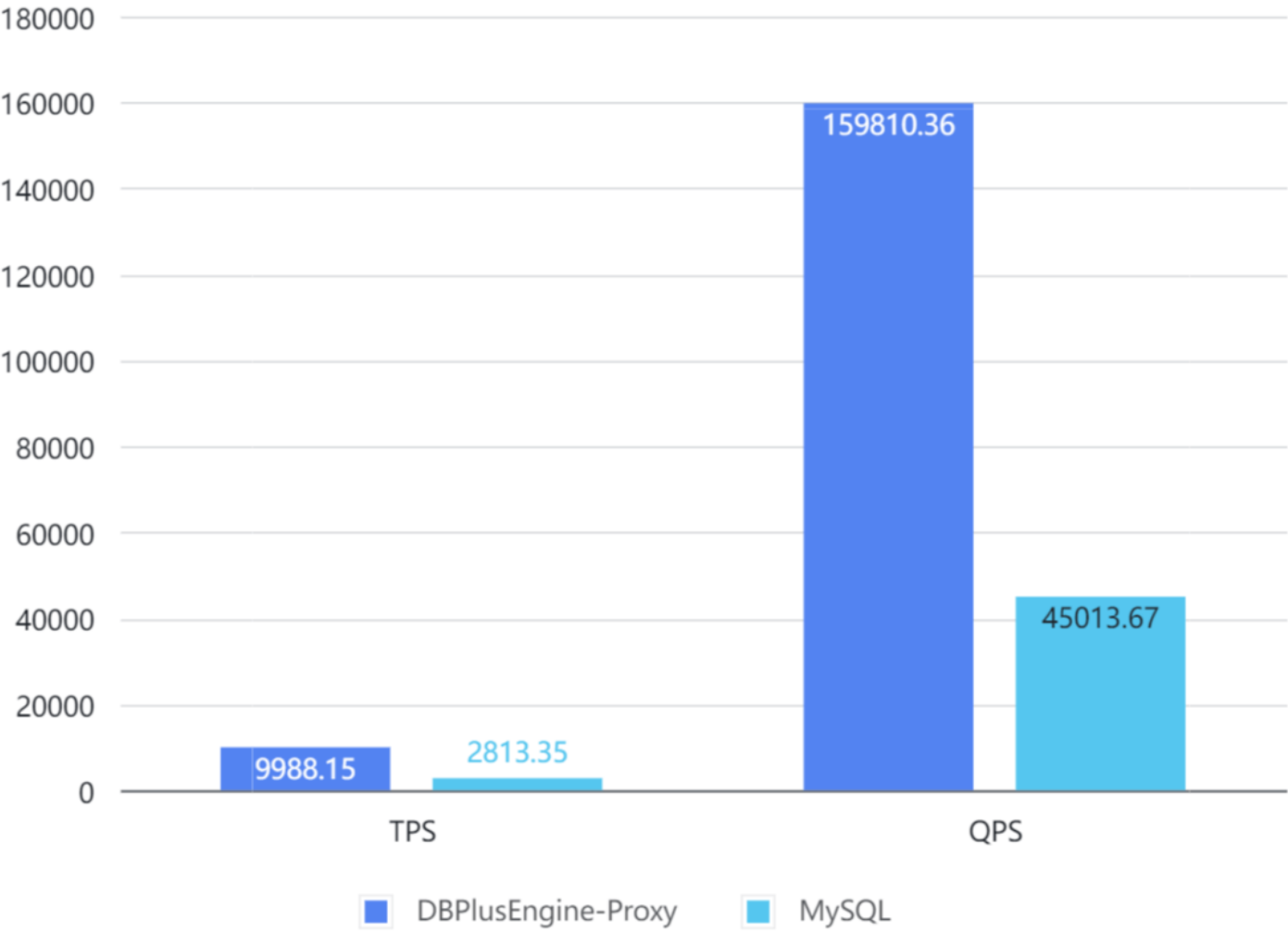
■ 读写测试

```
sysbench oltp_read_write --mysql-host='192.168.xx.25' --mysql-port=3307 --mysql-user=root --mysql-password='root' --mysql-db=sharding_db --tables=1 --table-size=140000000 --report-interval=5 --time=180 --threads=200 --max-requests=0 --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform --auto_inc=off run
```

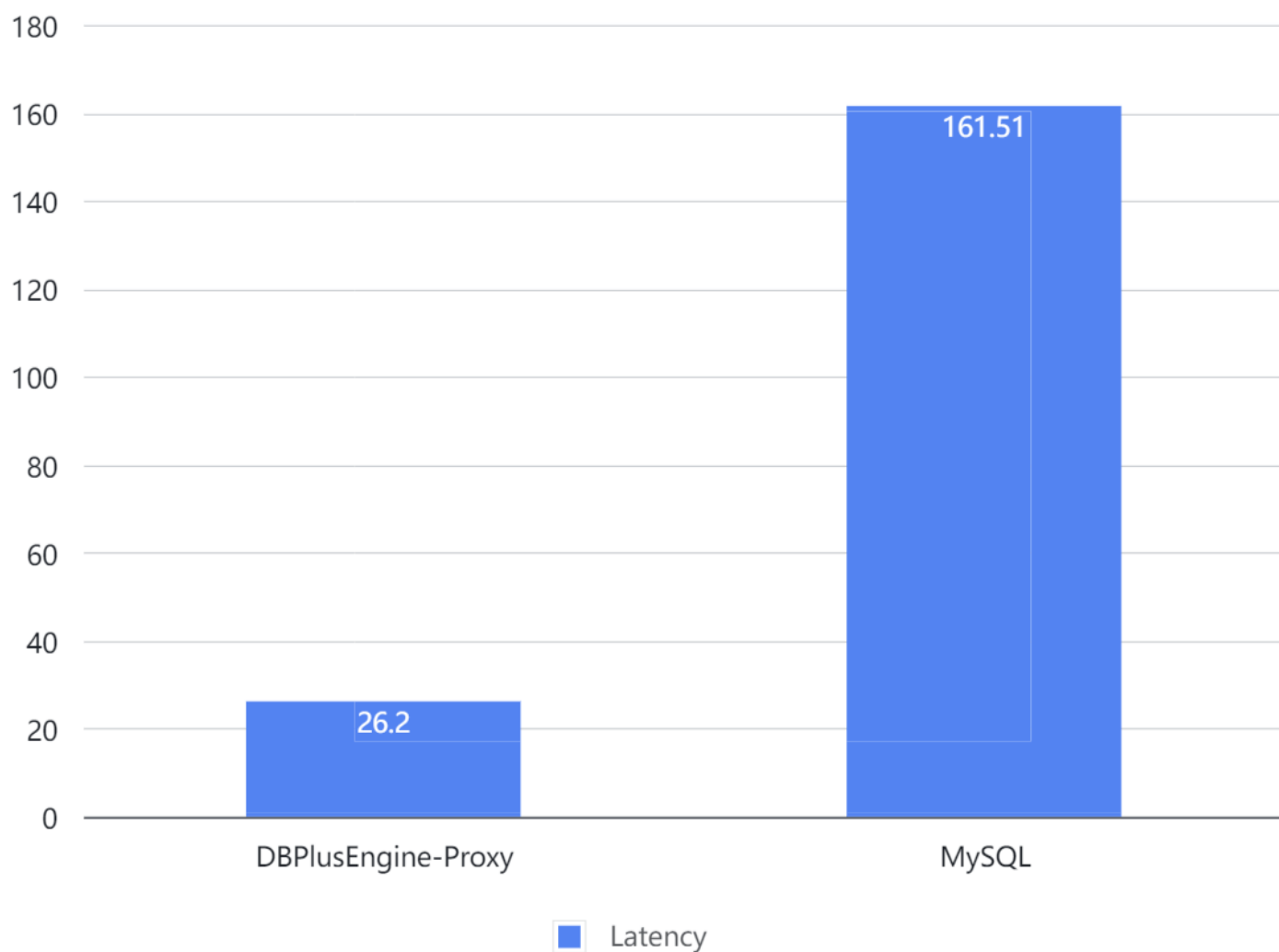
测试结果

应用	Threads	TPS	Latency99%(ms)
DBPlusEngine-Proxy	256	10,710	43.39
MySQL	256	1,827	475.79

1亿4千万行数据下性能对比



Latency (ms) [95th percentile]

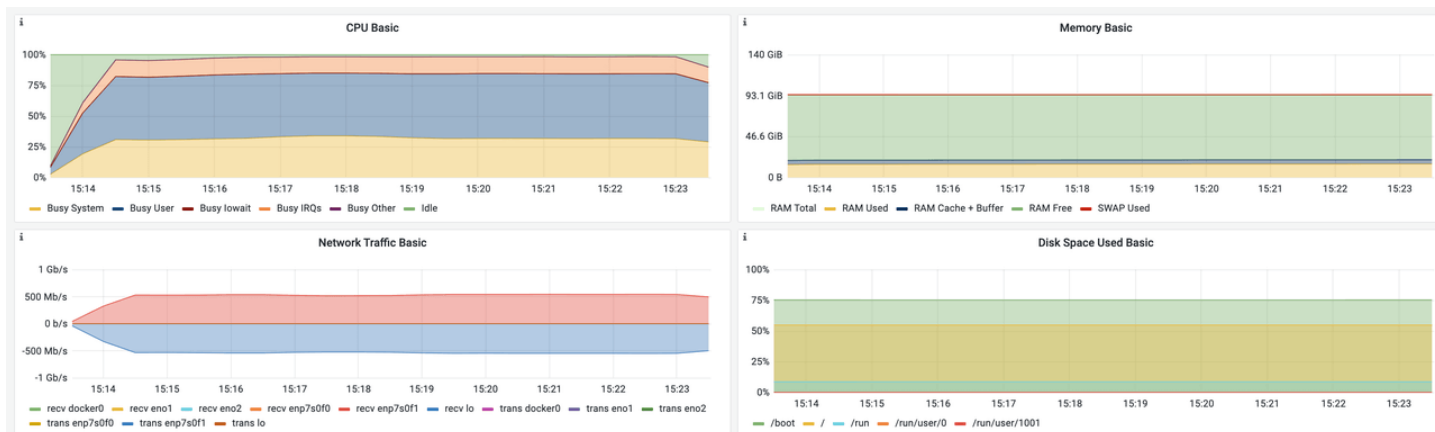


结论：在 sharding 场景下，计算节点扩容，吞吐量可近线性提升。
该结论仅供参考，用户可以根据实际业务情况，实际场景进行测试。

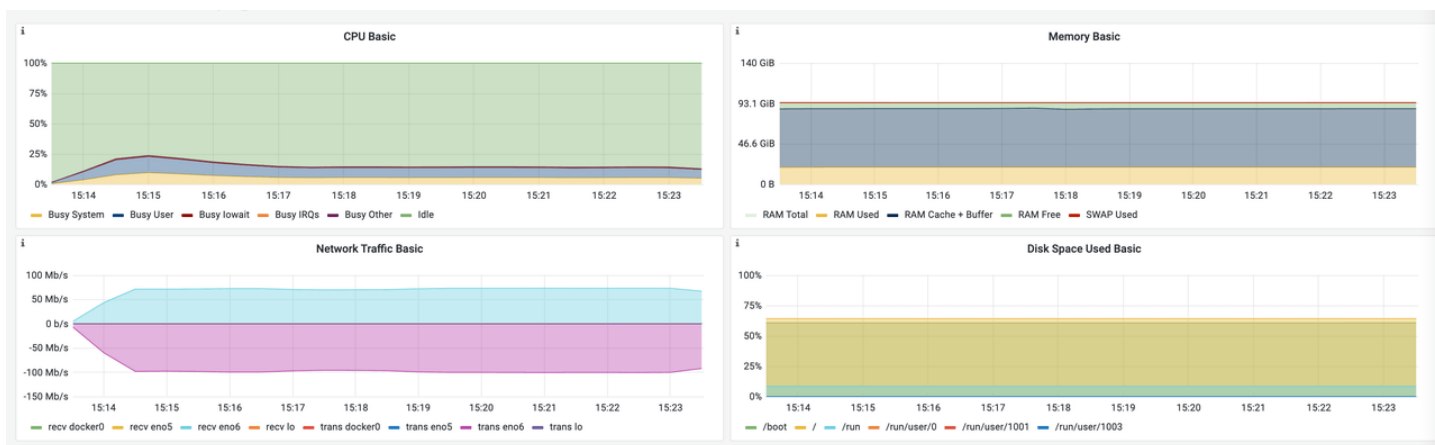
监控信息

Sysbench —> Proxy —> MySQL(4)

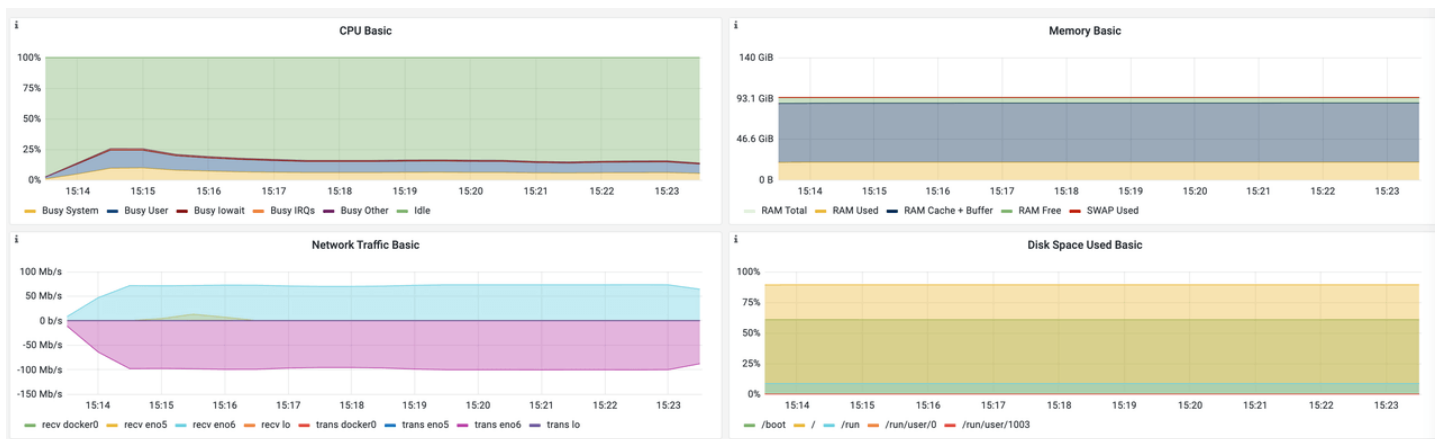
■ Proxy



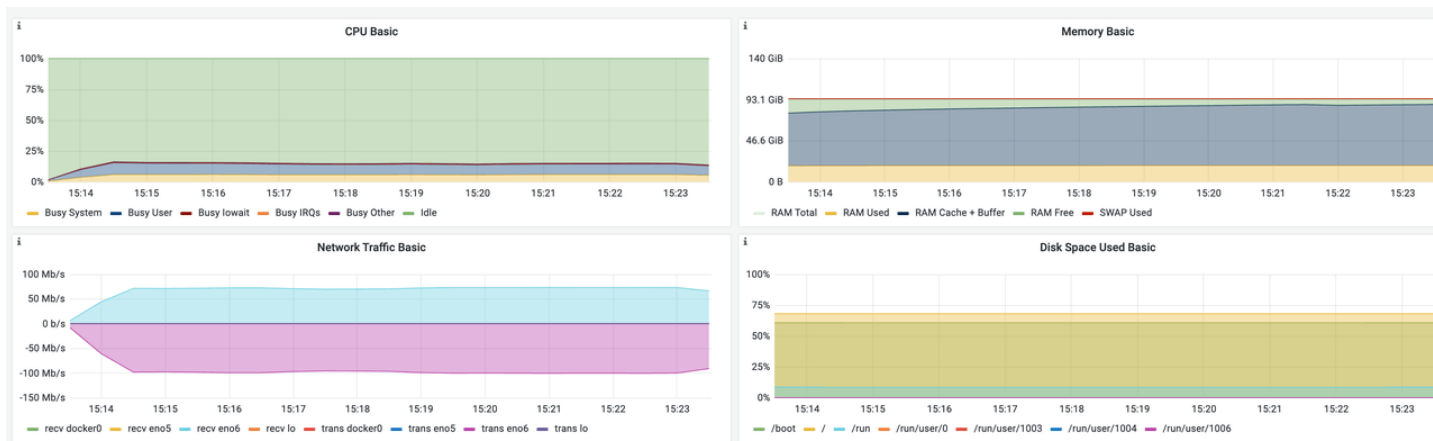
MySQL(1)



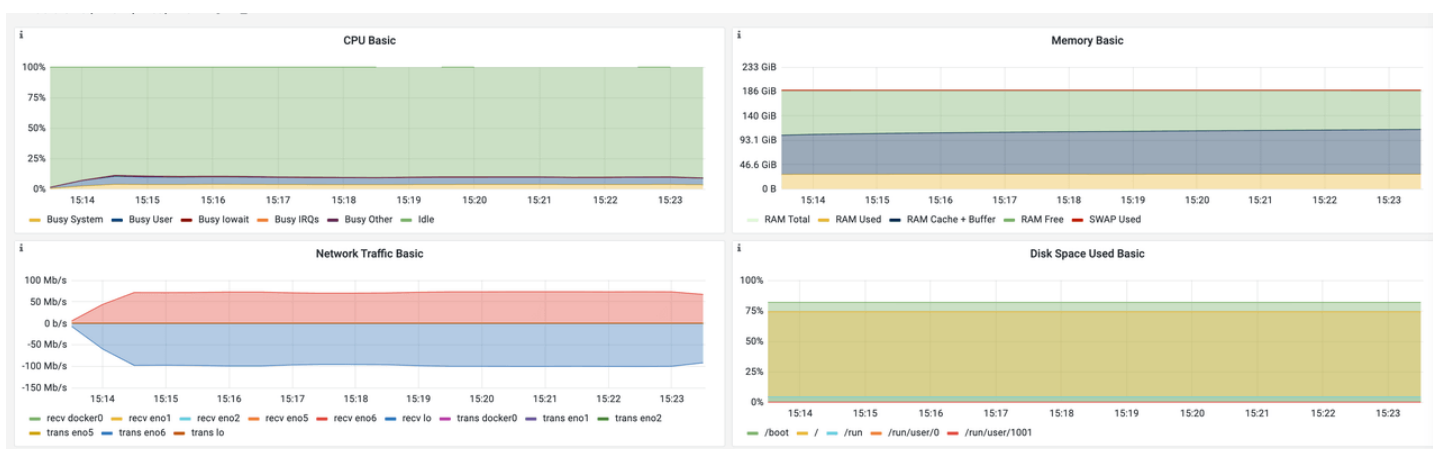
MySQL(2)



MySQL(3)

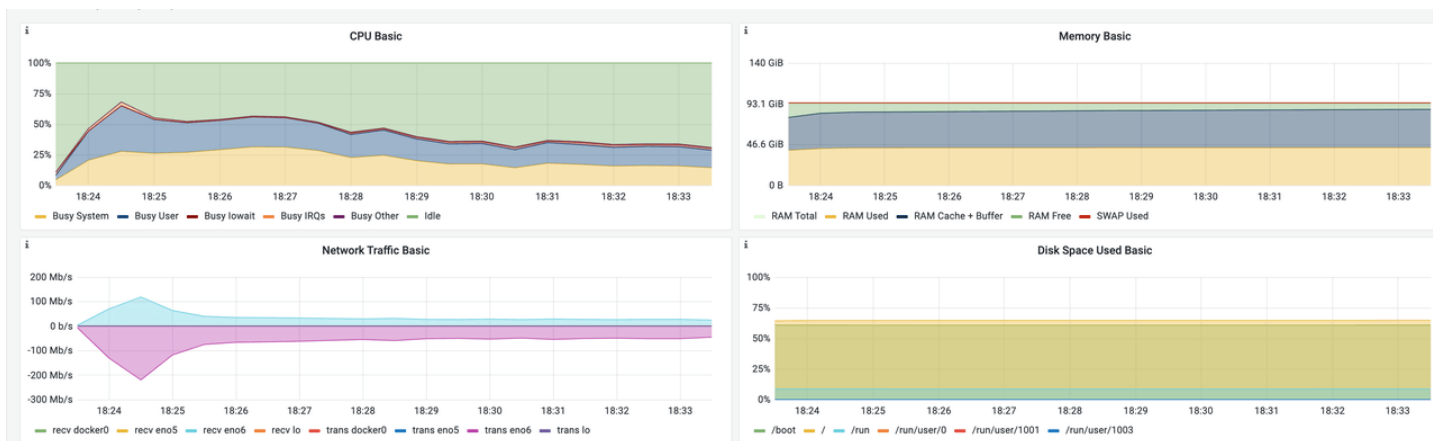


■ MySQL(4)



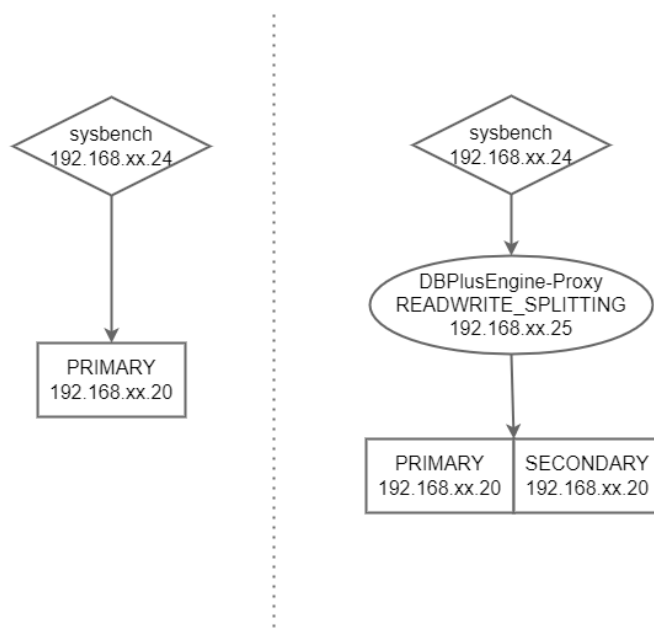
Sysbench —> MySQL

■ MySQL



5.3.2 读写分离

读写分离测试场景。



测试目的

DBPlusEngine-Proxy 根据读写分离规则，将读请求和写请求分别路由至主库与从库。使用 sysbench 工具执行读写场景压测，获取读写分离功能的性能指标。

测试工具

Sysbench

测试环境

应用	IP 地址	端口	版本
DBPlusEngine-Proxy	192.168.xx.25	3307	1.2
MySQL	192.168.xx.20	13308	8.0.29
MySQL	192.168.xx.20	13309	8.0.29

服务器配置

条目	配置
CPU	48 C
内存	96 G
硬盘	SSD 820 G
JDK	17.0.2

测试步骤

1. 创建测试库。

```
# 主库
mysql -utest -h192.168.xx.20 -P13308 -p
```

```
CREATE DATABASE IF NOT EXISTS test;
```

创建逻辑库。

```
mysql -uroot -h192.168.xx.25 -P3307 -proot
```

```
DROP DATABASE IF EXISTS sphereex_demo;
CREATE DATABASE IF NOT EXISTS sphereex_demo;
```

通过 DistSQL 关联数据源。

```
use sphereex_demo;

REGISTER STORAGE UNIT write_ds (
  HOST="192.168.xx.20",
  PORT=13308,
  DB="test",
  USER="test",
  PASSWORD="test"
), read_ds (
  HOST="192.168.xx.20",
  PORT=13309,
  DB="test",
  USER="test",
  PASSWORD="test"
);
```

通过 DistSQL 创建读写分离规则。

```
CREATE READWRITE_SPLITTING RULE readwrite_ds (
WRITE_STORAGE_UNIT=write_ds,
READ_STORAGE_UNITS(read_ds),
TYPE(NAME="FIXED_REPLICA_RANDOM")
);
```

2. 安装测试工具。

```
tar -xvzf sysbench-1.0.20.tar.gz -C /usr/local
cd /usr/local/sysbench-1.0.20
./autogen.sh
./configure
make && make install
```

3. 数据准备。

```
sysbench oltp_point_select --mysql-host='192.168.xx.25' --mysql-port=3307 --mysql-user=test --mysql-password='test' --mysql-db=sphereex_demo --tables=10 --table-size=1000000 --report-interval=10 --time=3600 --threads=256 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off --auto_inc=off cleanup
```

```
sysbench oltp_point_select --mysql-host='192.168.xx.25' --mysql-port=3307 --mysql-user=test --mysql-password='test' --mysql-db=sphereex_demo --tables=1 --table-size=1000000 --report-interval=10 --time=3600 --threads=256 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off --auto_inc=off prepare
```

4. 压测阶段。

■ 读写测试

```
sysbench oltp_read_write --mysql-host='192.168.xx.25' --mysql-port=3307 --mysql-user=root --mysql-password='root' --mysql-db=sphereex_demo --tables=1 --table-size=1000000 --report-interval=5 --time=120 --threads=200 --max-requests=0 --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform --auto_inc=off run
```

```
sysbench oltp_read_write --mysql-host='192.168.xx.20' --mysql-port=13308 --mysql-user=root --mysql-password='root' --mysql-db=test --tables=1 --table-size=1000000 --report-interval=5 --time=120 --threads=200 --max-requests=0 --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform --auto_inc=off run
```

测试结果

sysbench ——> Proxy ——> MySQL

SQL statistics:

queries performed:

read:	4185530
write:	1674212
other:	837106
total:	6696848
transactions:	418553 (3485.96 per sec.)
queries:	6696848 (55775.38 per sec.)
ignored errors:	0 (0.00 per sec.)
reconnects:	0 (0.00 per sec.)

General statistics:

total time:	120.0640s
total number of events:	418553

Latency (ms):

min:	7.61
avg:	57.35
max:	1035.85
95th percentile:	164.45
sum:	24005109.36

Threads fairness:

```
events (avg/stddev):      2092.7650/22.11
execution time (avg/stddev): 120.0255/0.01
```

```
# sysbench —> MySQL
```

```
SQL statistics:
```

```
queries performed:
```

```
read:      2398650
write:     959460
other:     479730
total:     3837840
transactions: 239865 (1995.35 per sec.)
queries:    3837840 (31925.58 per sec.)
ignored errors: 0 (0.00 per sec.)
reconnects: 0 (0.00 per sec.)
```

```
General statistics:
```

```
total time:      120.2079s
total number of events: 239865
```

```
Latency (ms):
```

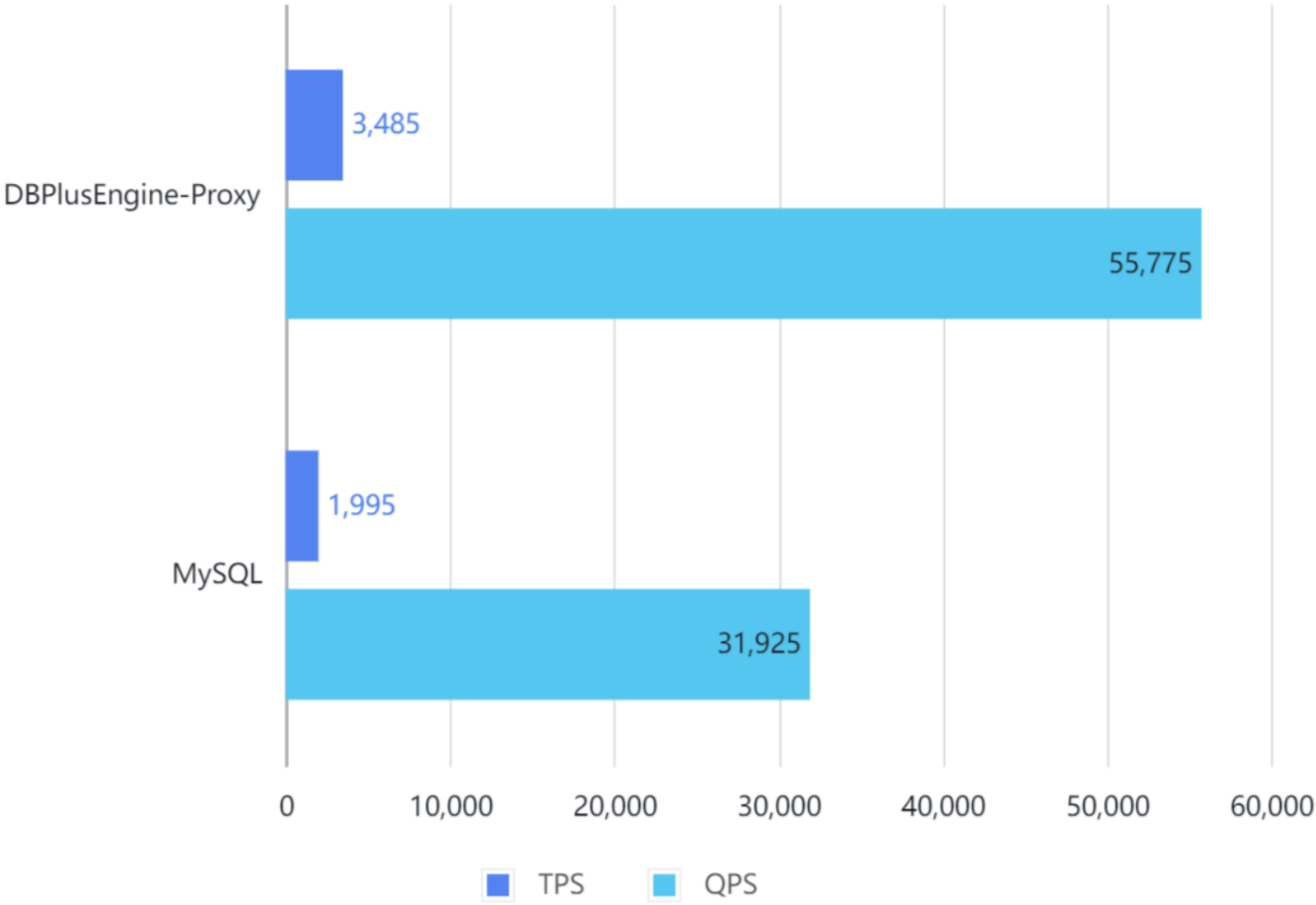
```
min:      2.93
avg:     100.16
max:    1735.34
95th percentile: 282.25
sum:    24023941.63
```

```
Threads fairness:
```

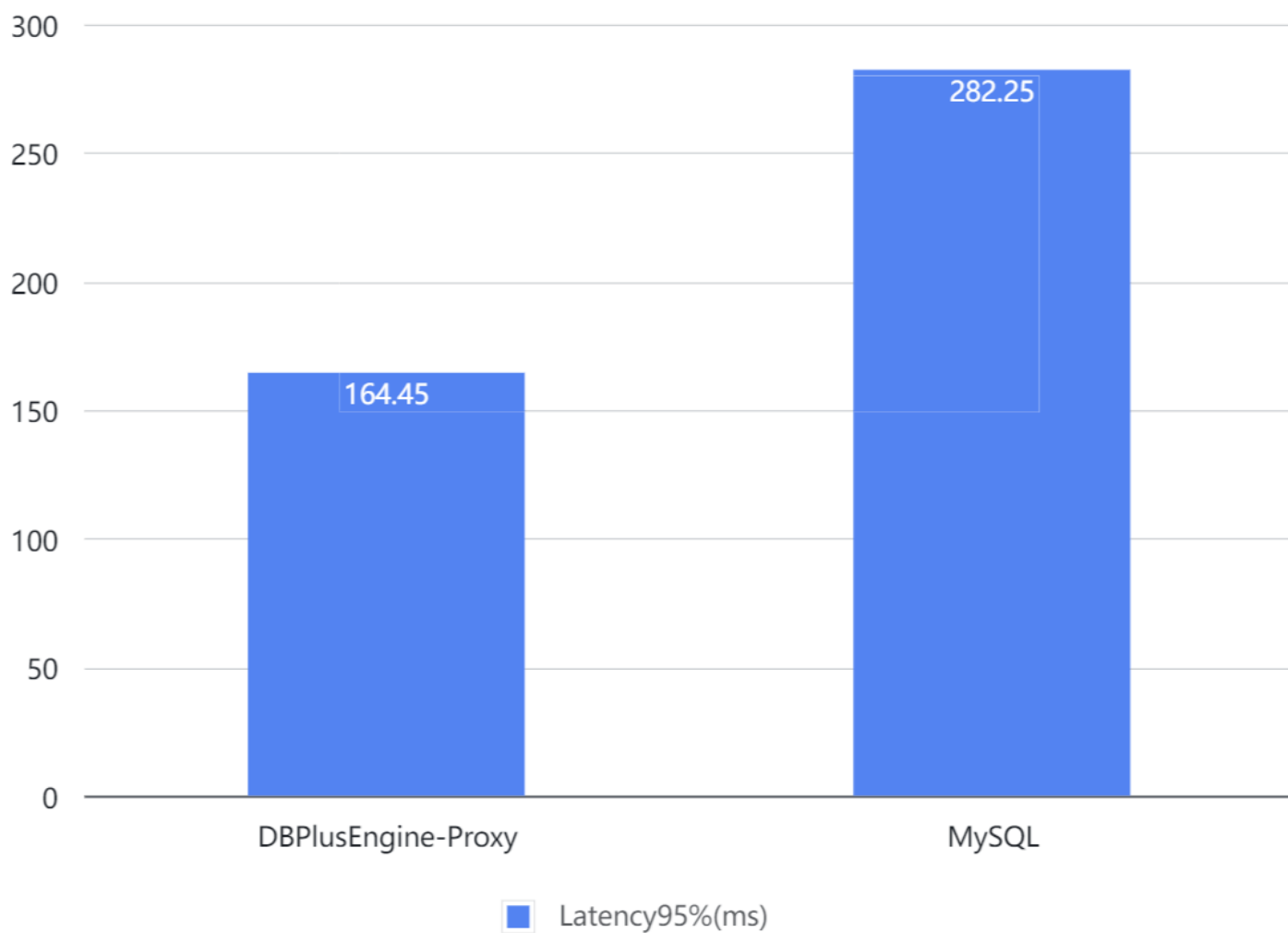
```
events (avg/stddev):      1199.3250/13.23
execution time (avg/stddev): 120.1197/0.08
```

应用	TPS	QPS	Latency95%(ms)
DBPlusEngine-Proxy	3,485	55,775	164.45
MySQL	1,995	31,925	282.25

读写分离功能性能对比



Latency (ms) [95th percentile]

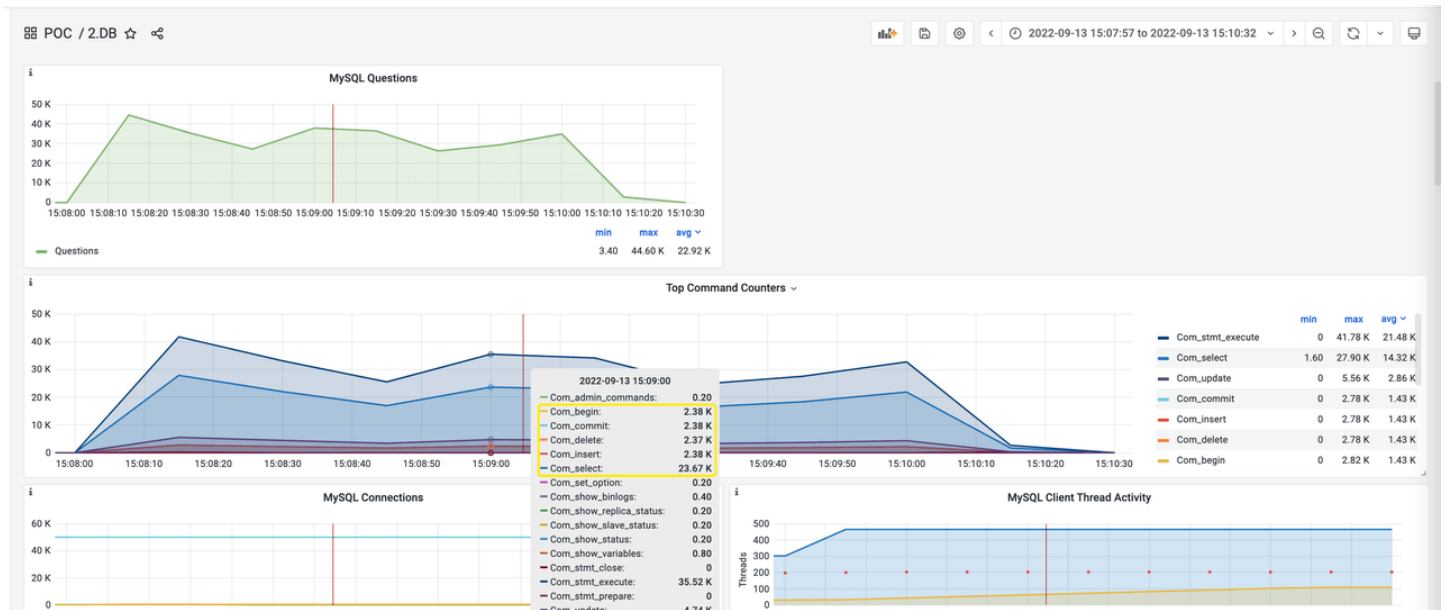


结论：在读写分离场景下性能及吞吐量得到显著提升。
该结论仅供参考，用户可以根据实际业务情况，实际场景进行测试。

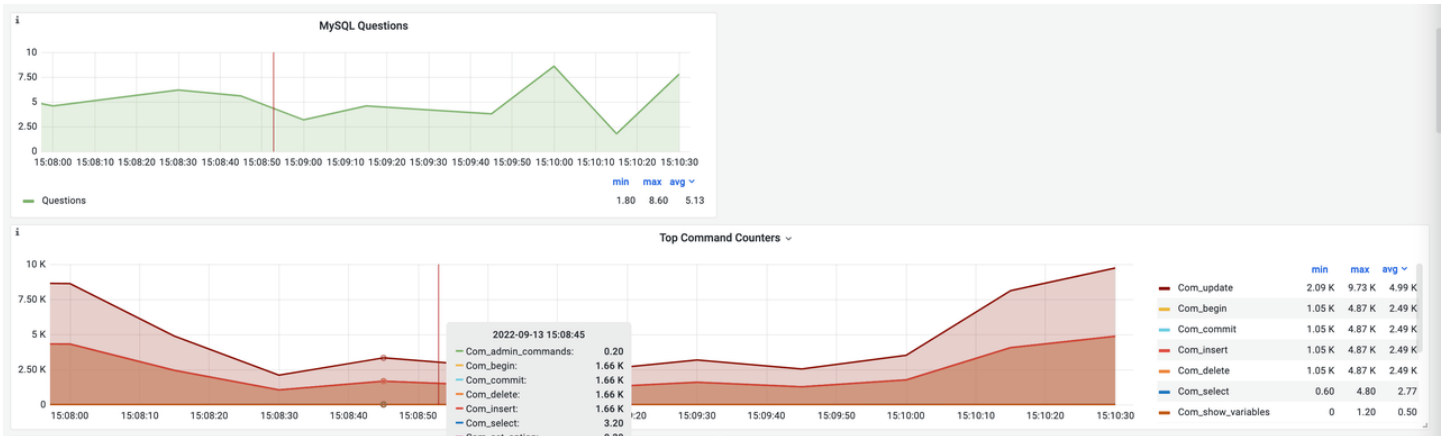
监控信息

sysbench —> MySQL

PRIMARY

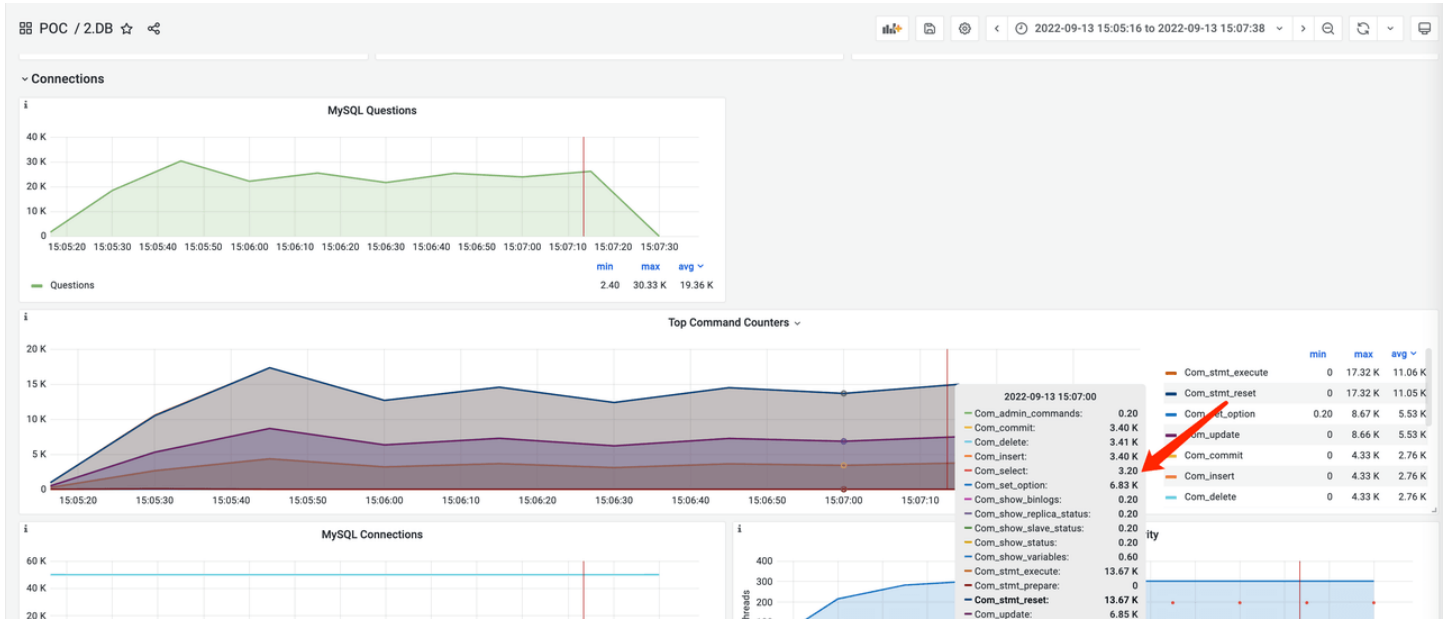


SECONDARY

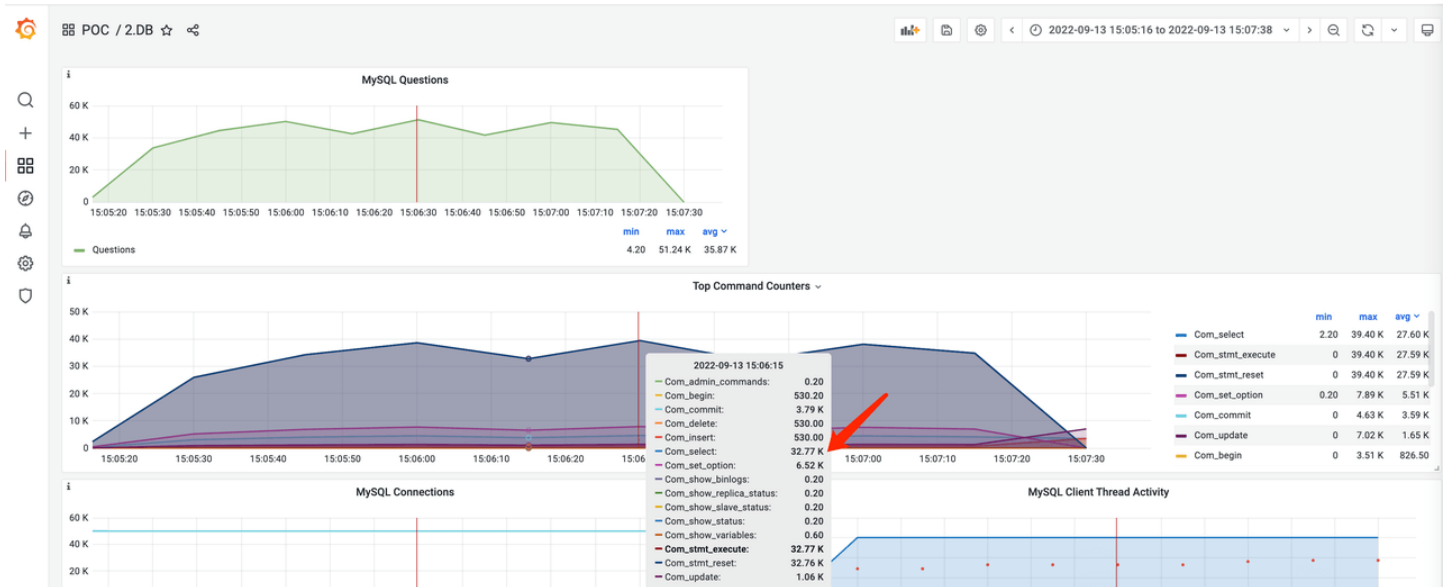


sysbench —> Proxy —> MySQL

PRIMARY

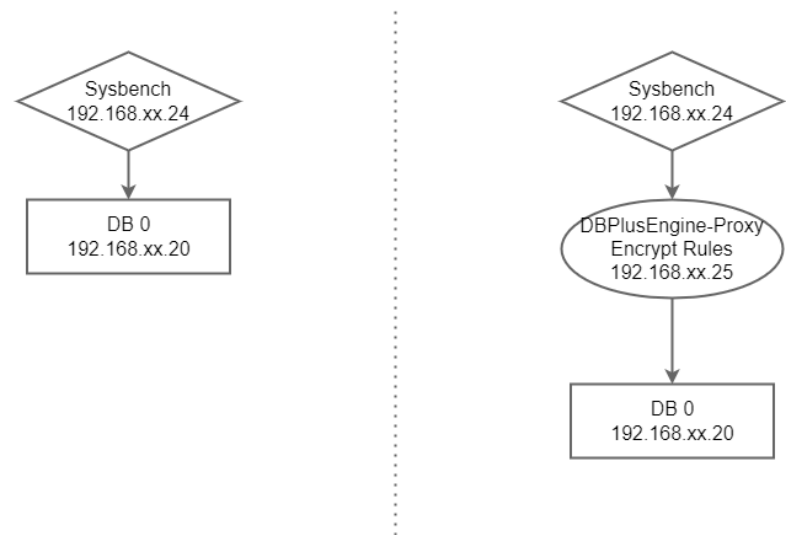


SECONDARY



5.3.3 数据加密

数据加密测试场景。



测试目的

通过 DBPlusEngine-Proxy 根据加密规则向数据库插入数据。使用 sysbench 工具执行读写场景压测，从而获得加密功能的性能指标。

测试工具

Sysbench

测试环境

应用	IP 地址	端口	版本
DBPlusEngine-Proxy	192.168.xx.25	3307	1.2
MySQL	192.168.xx.20	13306	8.0.29

服务器配置

条目	配置
CPU	48 C
内存	96 G
硬盘	SSD 820 G
JDK	17.0.2

测试步骤

1. 创建测试库

```
mysql -utest -h192.168.xx.20 -P13306 -p
```

```
DROP DATABASE IF EXISTS encrypt_db;
```

```
CREATE DATABASE IF NOT EXISTS encrypt_db;
```

创建逻辑库。

```
mysql -uroot -h192.168.xx.25 -P3307 -proot
```

```
DROP DATABASE IF EXISTS sphereex_demo;
```

```
CREATE DATABASE IF NOT EXISTS sphereex_demo;
```

通过 DistSQL 关联数据源。

```
USE sphereex_demo;
```

```
REGISTER STORAGE UNIT sphereex_demo(  
  HOST="192.168.xx.20",  
  PORT=13306,  
  DB="encrypt_db",  
  USER="test",  
  PASSWORD="test"  
);
```

通过 DistSQL 创建加密规则。

```
CREATE ENCRYPT RULE t_user (  
  COLUMNS(  
    (NAME=idCard,CIPHER=idcard_cipher,TYPE(NAME="AES",PROPERTIES("aes-key-value"="123456abc"))),  
    (NAME=mobile, CIPHER =mobile_cipher,TYPE(NAME="AES",PROPERTIES("aes-key-value"="123456abc"))));
```

2. 安装测试工具

```
tar -xvzf sysbench-1.0.20.tar.gz -C /usr/local  
cd /usr/local/sysbench-1.0.20  
./autogen.sh  
./configure  
make && make install
```

3. 数据准备

```
sysbench oltp_read_only --mysql-host='192.168.xx.25' --mysql-port=3307 --mysql-user=root --mysql-password='root' --mysql-db=sphereex_demo --tables=10 --table-size=1000000 --report-interval=10 --time=3600 --threads=128 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off --auto_inc=off cleanup
```

```
sysbench oltp_read_only --mysql-host='192.168.xx.25' --mysql-port=3307 --mysql-user=root --mysql-password='root' --mysql-db=sphereex_demo --tables=10 --table-size=1000000 --report-interval=10 --time=3600 --threads=128 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off --auto_inc=off prepare
```

4. 压测阶段

读写测试

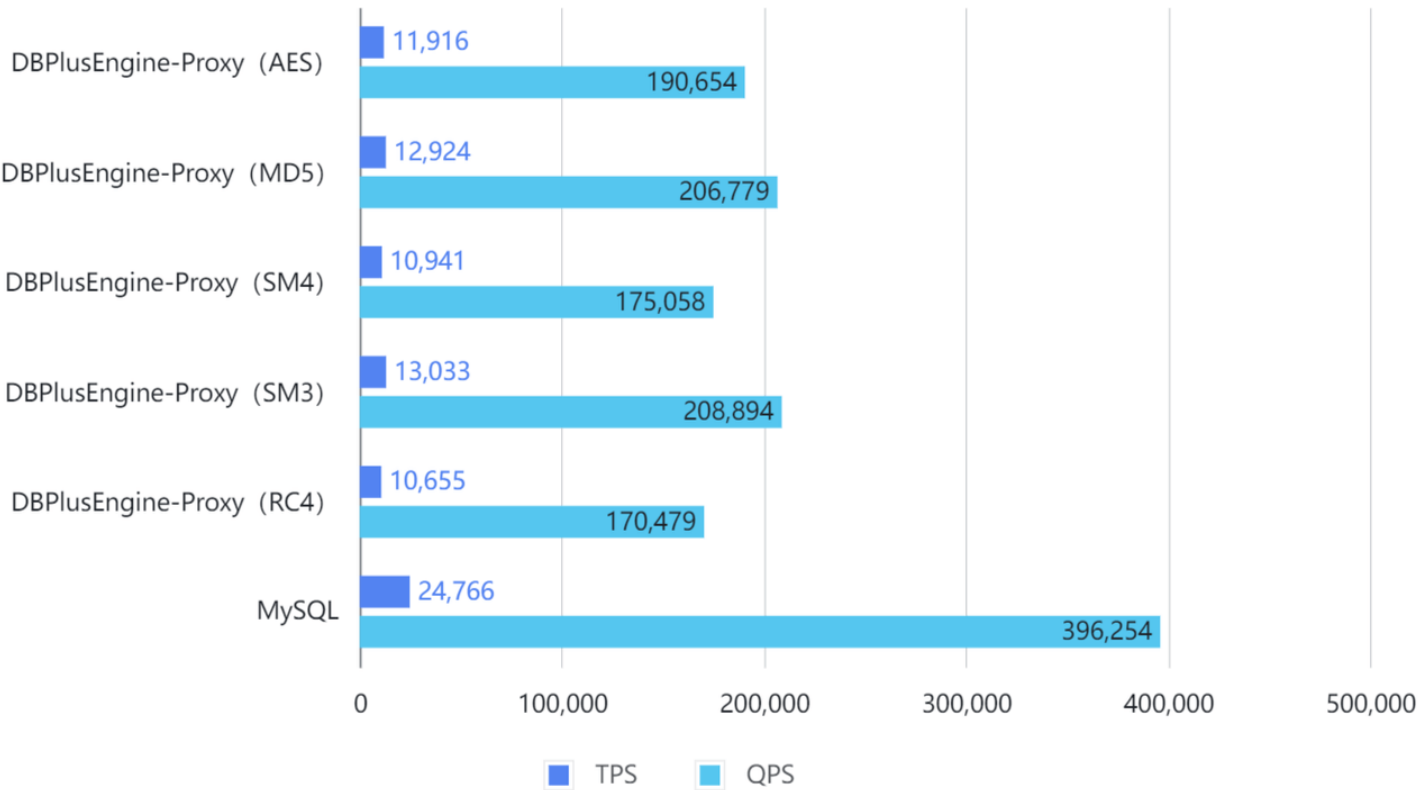
```
sysbench oltp_read_write --mysql-host='192.168.xx.25' --mysql-port=3307 --mysql-user=root --mysql-password='root' --mysql-db=sphereex_demo --tables=1 --table-size=1000000 --report-interval=5 --time=180 --threads=200 --max-requests=0 --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform --auto_inc=off run
```

测试结果

应用	TPS	QPS
DBPlusEngine-Proxy (AES)	11,916	190,654
DBPlusEngine-Proxy (MD5)	12,924	206,779
DBPlusEngine-Proxy (SM4)	10,941	175,058
DBPlusEngine-Proxy (SM3)	13,033	208,894
DBPlusEngine-Proxy (RC4)	10,655	170,479
MySQL	24,766	396,254

结论：在数据加密场景下，吞吐量均有较大衰减，不同加密算法间略有差异。
该结论仅供参考，用户可以根据实际业务情况，实际场景进行测试。

Proxy 各类加密算法 & MySQL 性能对比



监控信息

DBPlusEngine-Proxy (AES)

■ Proxy

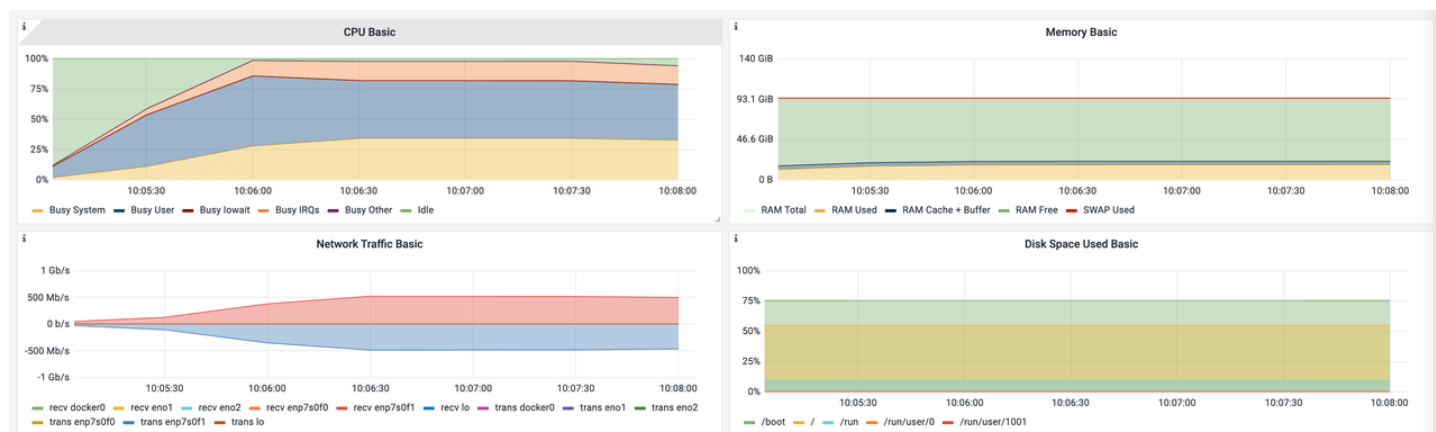


■ MySQL

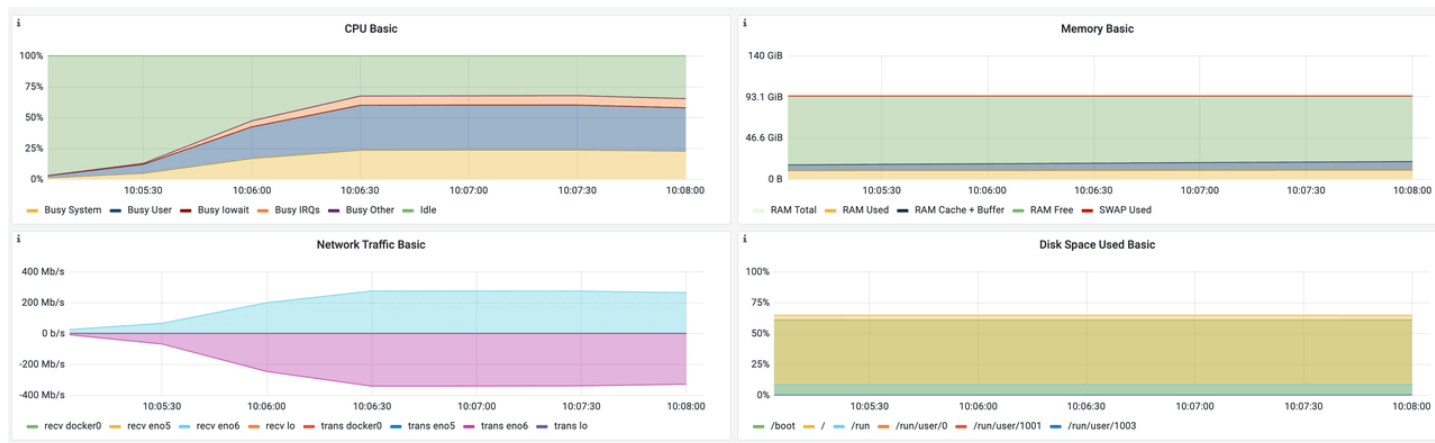


DB Plus Engine-Proxy (MD5)

■ Proxy

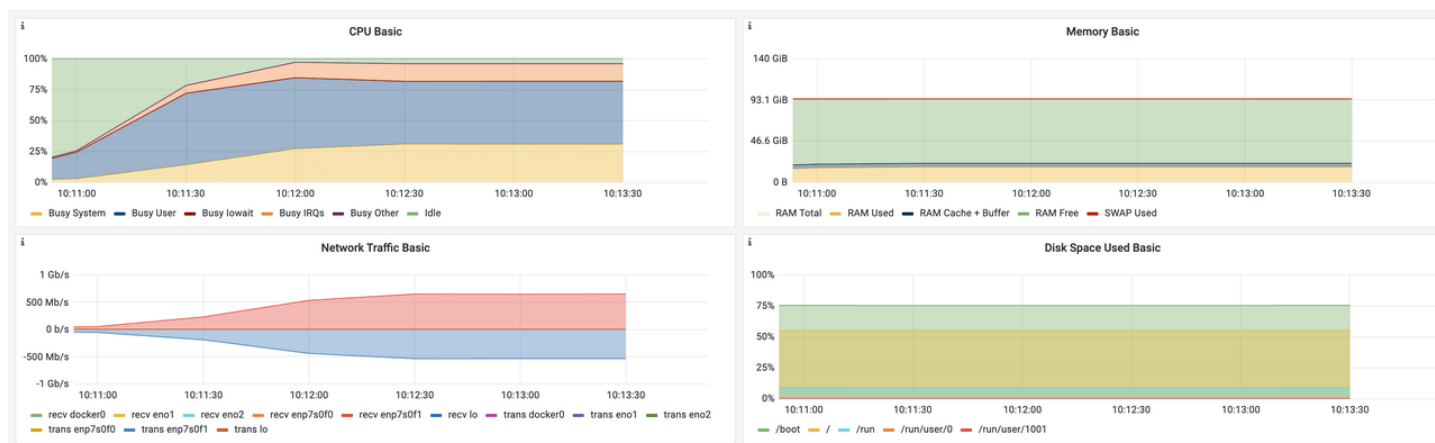


■ MySQL

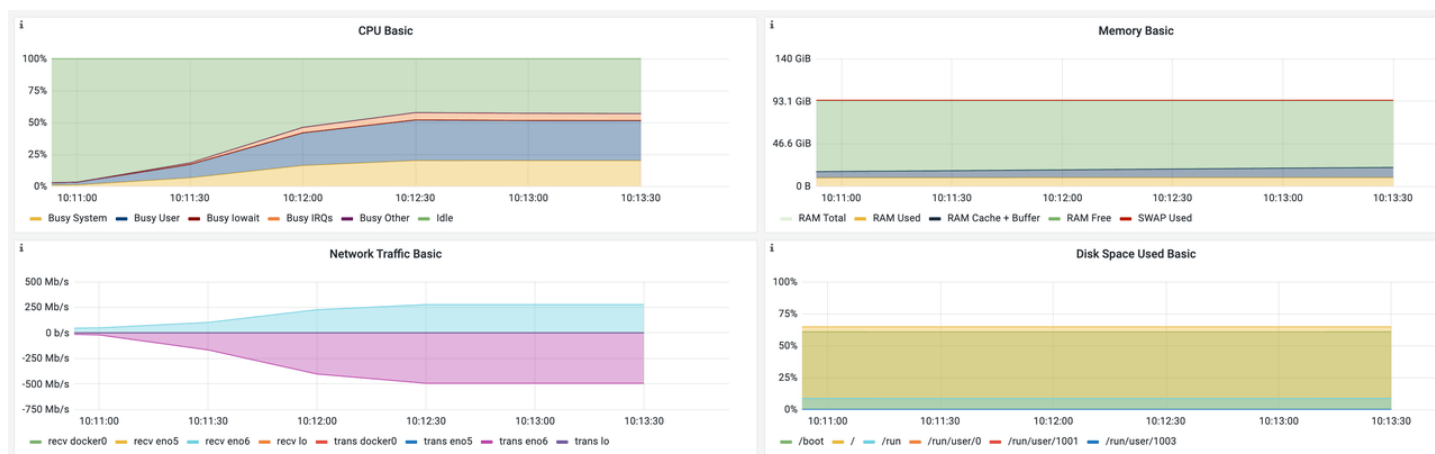


DB Plus Engine-Proxy (SM4)

■ Proxy



■ MySQL

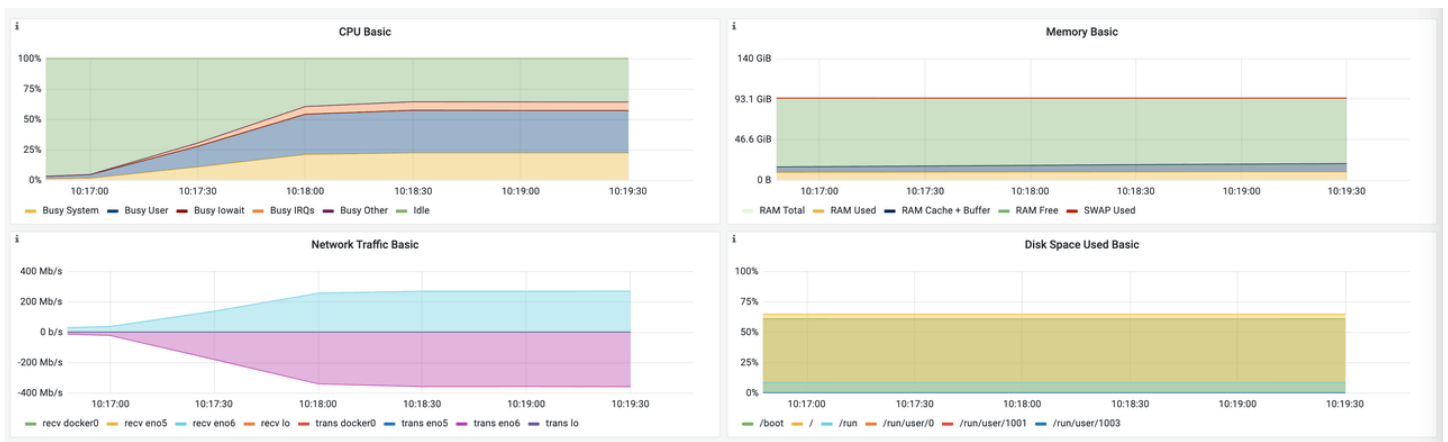


DB Plus Engine-Proxy (SM3)

■ Proxy

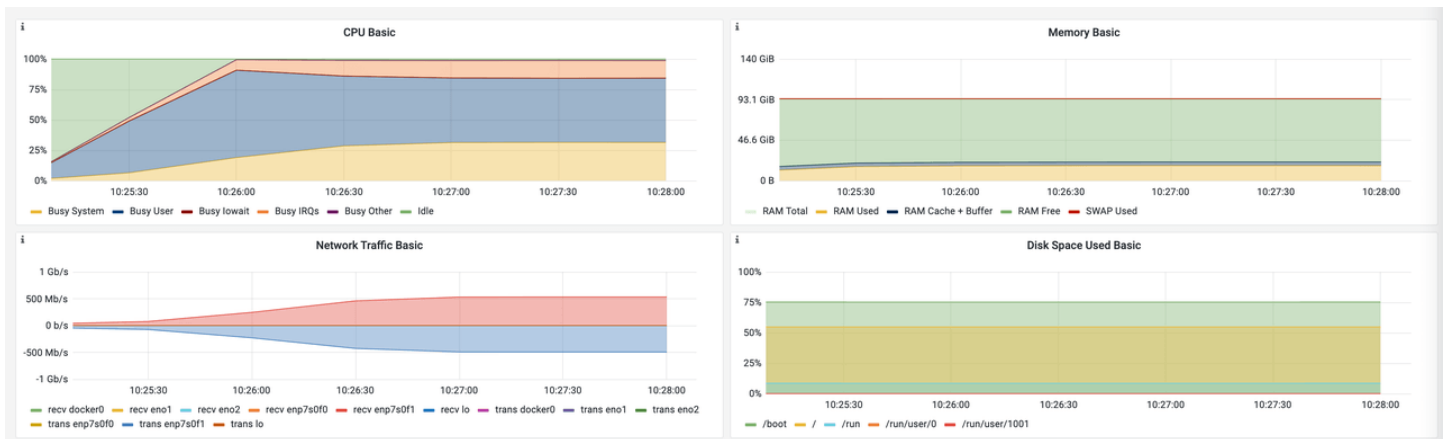


MySQL

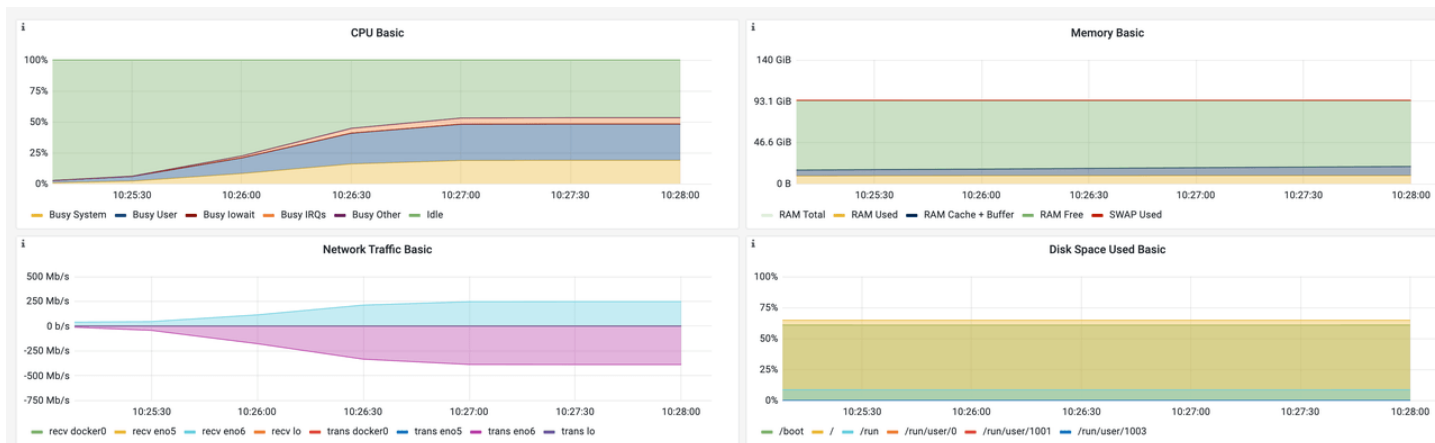


DB Plus Engine-Proxy (RC4)

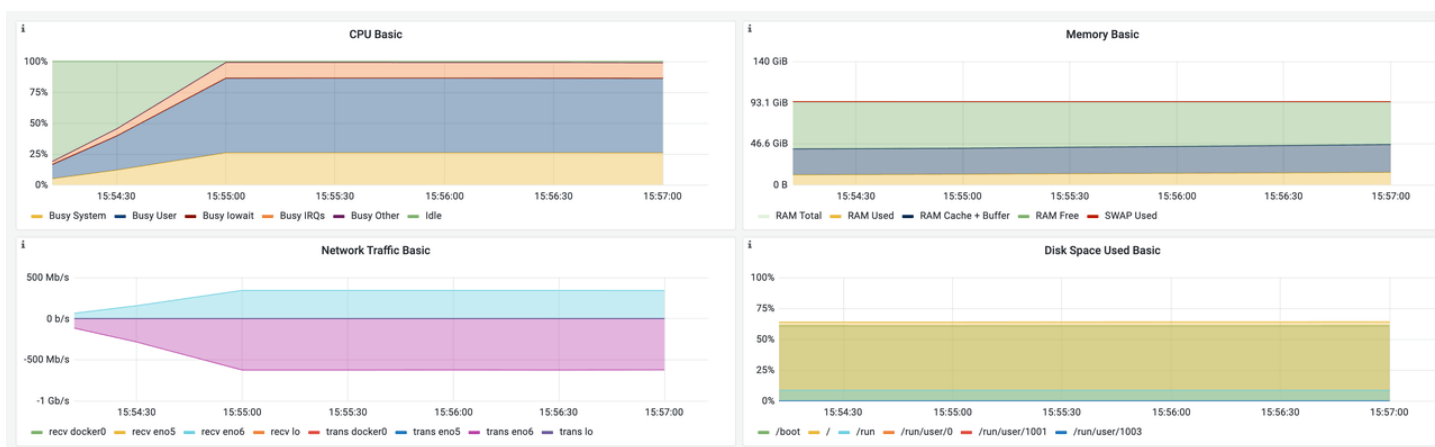
Proxy



MySQL

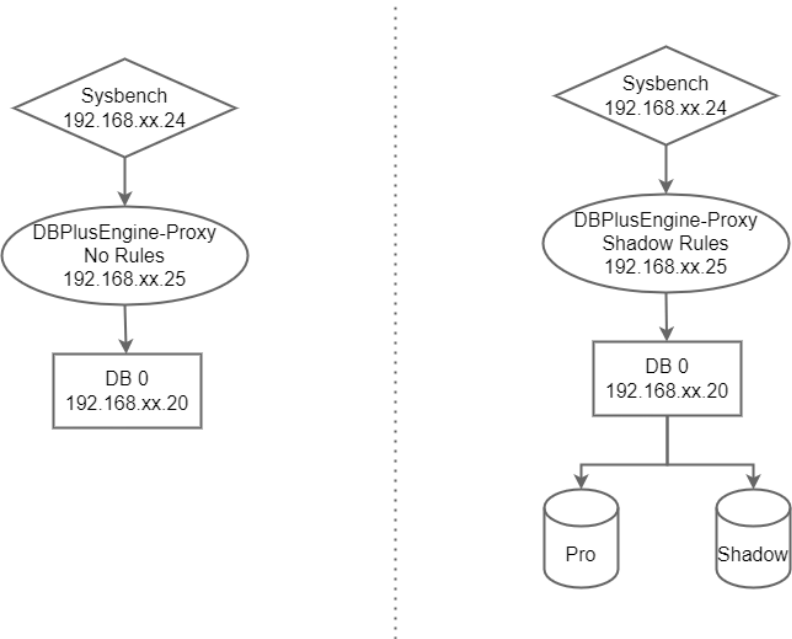


MySQL



5.3.4 影子库

影子库测试场景。



测试目的

分别对生产库和影子库创建数据，然后通过值匹配算法，根据路由规则分别对生产库和影子库进行请求，使用 sysbench 压测工具进行点查操作，获得影子库的具体性能指标。

测试工具

Sysbench

测试环境

应用	IP 地址	端口	版本
DBPlusEngine-Proxy	192.168.xx.25	3307	1.2
MySQL	192.168.xx.20	13306	8.0.29
MySQL	192.168.xx.21	13306	8.0.29

服务器配置

条目	配置
CPU	48 C
内存	96 G
硬盘	SSD 820 G
JDK	17.0.2

测试步骤

1. 创建测试库

创建生产库和影子库。

```
mysql -utest -h192.168.xx.20 -P13306 -p
```

```
DROP DATABASE IF EXISTS prod_ds;
```

```
CREATE DATABASE IF NOT EXISTS prod_ds;
```

```
mysql -utest -h192.168.xx.21 -P13306 -p
```

```
DROP DATABASE IF EXISTS shadow_ds;
```

```
CREATE DATABASE IF NOT EXISTS shadow_ds;
```

创建逻辑库。

```
mysql -uroot -h192.168.xx.25 -P3307 -proot
```

```
DROP DATABASE IF EXISTS sphereex_demo;
```

```
CREATE DATABASE IF NOT EXISTS sphereex_demo;
```

通过 DistSQL 关联数据源。

```
USE sphereex_demo;
```

```
REGISTER STORAGE UNIT ds (
```

```
  HOST="192.168.xx.20",
```

```
  PORT=13306,
```

```
  DB="prod_ds",
```

```
  USER="test",
```

```
  PASSWORD="test"
```

```
),ds_shadow (
```

```
  HOST="192.168.xx.21",
```

```
  PORT=13306,
```

```
  DB="shadow_ds",
```

```
  USER="test",
```

```
  PASSWORD="test"
```

```
);
```

通过 DistSQL 创建影子库规则。

```
CREATE SHADOW RULE shadowDataSource(
```

```
  SOURCE=ds,
```

```
  SHADOW=ds_shadow,
```

```
  t_user(
```

```
    TYPE(NAME="VALUE_MATCH", PROPERTIES("operation"="insert", "column"="status", "value"=1)),
```

```
    TYPE(NAME="SIMPLE_HINT", PROPERTIES("shadow"=true)))
```

```
);
```

2. 安装测试工具

```
tar -xvzf sysbench-1.0.20.tar.gz -C /usr/local
cd /usr/local/sysbench-1.0.20
./autogen.sh
./configure
make && make install
```

对 Sysbench 的 oltp_common.lua 脚本改造，支持 Shadow 的 hint 算法。

```
local stmt_defs = {
  point_selects = {
    "SELECT c FROM sbtest%u WHERE id=? /*shadow:true*/",
    t.INT},
  simple_ranges = {
    "SELECT c FROM sbtest%u WHERE id BETWEEN ? AND ?",
    t.INT, t.INT},
  sum_ranges = {
    "SELECT SUM(k) FROM sbtest%u WHERE id BETWEEN ? AND ?",
    t.INT, t.INT},
  order_ranges = {
    "SELECT c FROM sbtest%u WHERE id BETWEEN ? AND ? ORDER BY c",
    t.INT, t.INT},
  distinct_ranges = {
    "SELECT DISTINCT c FROM sbtest%u WHERE id BETWEEN ? AND ? ORDER BY c",
    t.INT, t.INT},
  index_updates = {
    "UPDATE sbtest%u SET k=k+1 WHERE id=? /*shadow:true*/",
    t.INT},
  non_index_updates = {
    "UPDATE sbtest%u SET c=? WHERE id=? /*shadow:true*/",
    {t.CHAR, 120}, t.INT},
  deletes = {
    "DELETE FROM sbtest%u WHERE id=? /*shadow:true*/",
    t.INT},
  inserts = {
    "INSERT INTO sbtest%u (id, k, c, pad) VALUES (?, ?, ?, ?)/*shadow:true*/",
    t.INT, t.INT, {t.CHAR, 120}, {t.CHAR, 60}},
}
```

3. 数据准备

```
sysbench oltp_read_only --mysql-host='192.168.xx.20' --mysql-port=13306 --mysql-user=test --mysql-password='test' --mysql-db=prod_ds --tables=10 --table-size=1000000 --report-interval=10 --time=3600 --threads=128 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off --auto_inc=off cleanup
```

```
sysbench oltp_read_only --mysql-host='192.168.xx.20' --mysql-port=13306 --mysql-user=test --mysql-password='test' --mysql-db=prod_ds --tables=10 --table-size=1000000 --report-interval=10 --time=3600 --threads=128 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off --auto_inc=off prepare
```

```
sysbench oltp_read_only --mysql-host='192.168.xx.21' --mysql-port=13306 --mysql-user=test --mysql-password='test' --mysql-db=shadow_ds --tables=10 --table-size=1000000 --report-interval=10 --time=3600 --threads=128 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off --auto_inc=off cleanup
```

```
sysbench oltp_read_only --mysql-host='192.168.xx.21' --mysql-port=13306 --mysql-user=test --mysql-password='test' --mysql-db=shadow_ds --tables=10 --table-size=1000000 --report-interval=10 --time=3600 --threads=128 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off --auto_inc=off prepare
```

4. 压测阶段

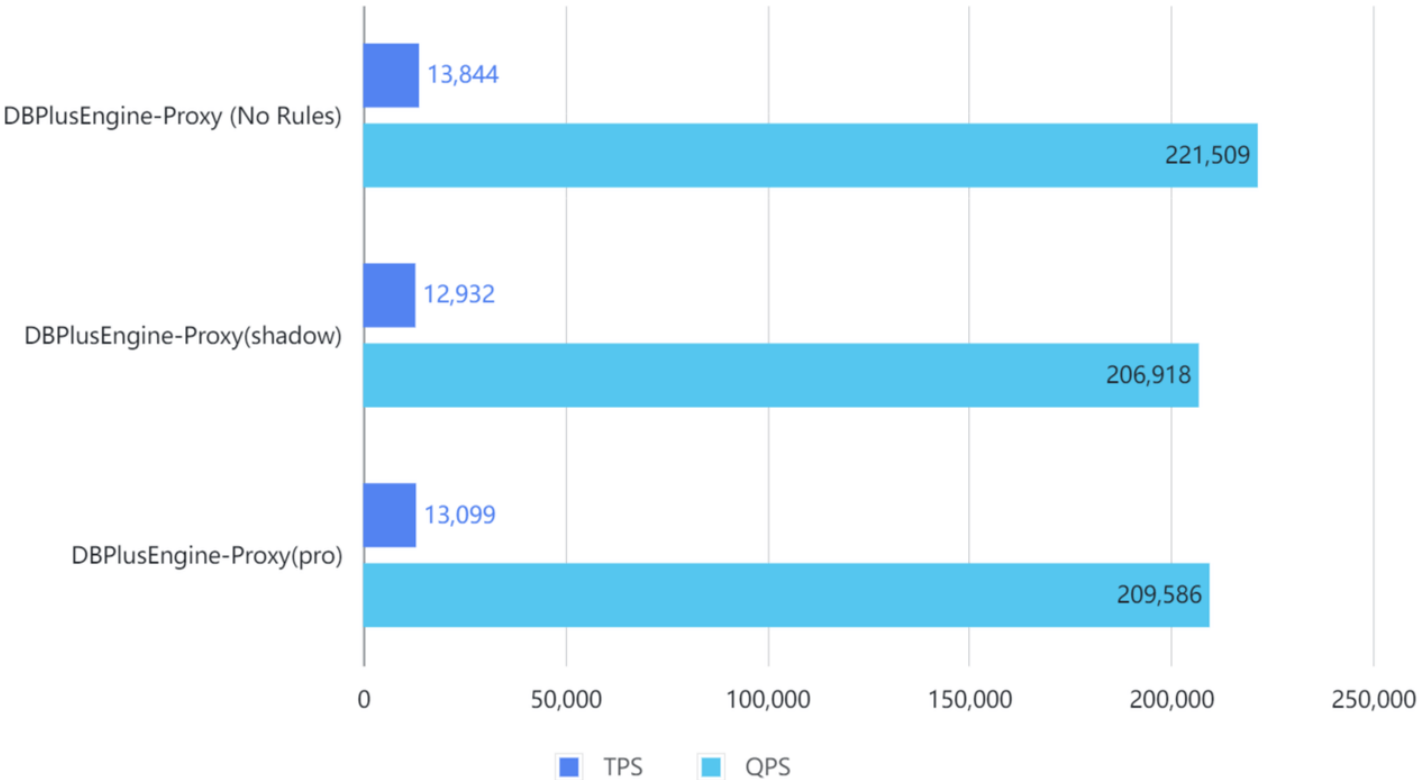
■ 只读测试。

```
sysbench oltp_point_select --mysql-host='192.168.xx.25' --mysql-port=3307 --mysql-user=root --mysql-password='root' --mysql-db=sphereex_demo --tables=10 --table-size=1000000 --report-interval=5 --time=600 --threads=128 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform --auto_inc=off run
```

测试结果

应用	TPS	QPS
DB Plus Engine-Proxy (No Rules)	13,844	221,509
DB Plus Engine-Proxy(shadow)	12,932	206,918
DB Plus Engine-Proxy(pro)	13,099	209,586

影子库相关性能对比



结论：在影子库场景下，流量路由至生产库 & 影子库性能差异不大，吞吐量无明显下降。
该结论仅供参考，用户可以根据实际业务情况，实际场景进行测试。

监控信息

DBPlusEngine-Proxy(No Rules)

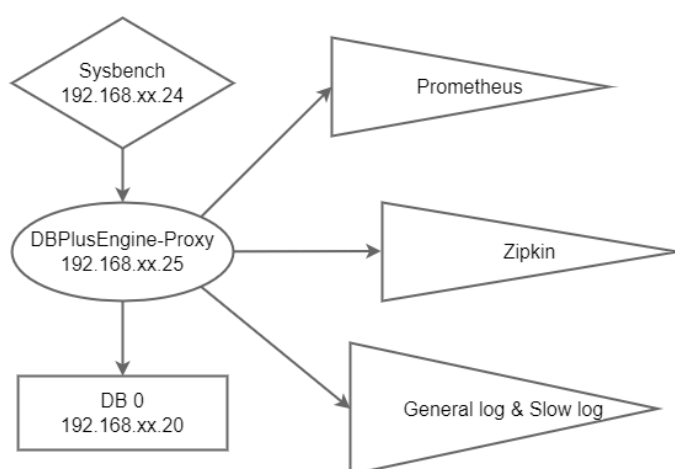


DB Plus Engine-Proxy(shadow / pro)



5.3.5 探针

探针测试场景。



测试目的

DBPlusEngine-Proxy 通过 plugin 结合 Prometheus/Zipkin/General log & Slow log 开启后性能损耗或其他组件获取可观测性，使用 sysbench 压测工具进行读写操作，获取 plugin 在开启各项功能后，DBPlusEngine-Proxy 的性能情况。

测试工具

Sysbench

测试环境

应用	IP 地址	端口	版本
DBPlusEngine-Proxy	192.168.xx.25	3307	1.2
DBPlusEngine-Plugin	192.168.xx.25	■	1.2
MySQL	192.168.xx.20	13306	8.0.29
Prometheus	■	■	■
Zipkin	■	■	■
General log & Slow log	■	■	■

服务器配置

条目	配置
CPU	48 C
内存	96 G
硬盘	SSD_SATA3 820 G
JDK	17.0.2

配置 agent.yaml，开启相关监控

```
# Copyright © 2022, Beijing Sifei Software Technology Co., LTD.
# All Rights Reserved.
# Unauthorized copying of this file, via any medium is strictly prohibited.
# Proprietary and confidential

plugins:
  logging:
    BaseLogging:
      props:
        slow-query-log: true
        long-query-time: 5000
        general-query-log: true
# metrics:
# Prometheus:
#   host: "0.0.0.0"
#   port: 9090
#   props:
#     jvm-information-collector-enabled: "true"
# tracing:
# Jaeger:
#   host: "localhost"
#   port: 5775
```

```
# props:
#   service-name: "shardingsphere"
#   jaeger-sampler-type: "const"
#   jaeger-sampler-param: "1"
# Zipkin:
#   host: "localhost"
#   port: 9411
# props:
#   service-name: "shardingsphere"
#   url-version: "/api/v2/spans"
#   sampler-type: "const"
#   sampler-param: "1"
# SkyWalking:
#   props:
#     opentracing-tracer-class-name: "org.apache.skywalking.apm.toolkit.opentracing.SkywalkingTracer"
# OpenTelemetry:
#   props:
#     otel-resource-attributes: "service.name=shardingsphere"
#     otel-traces-exporter: "zipkin"
```

使用 start-with-agent.sh 脚本启动 DBPlusEngine-Proxy。

测试步骤

1. 创建测试库

```
mysql -utest -h192.168.xx.20 -P13306 -p
```

```
DROP DATABASE IF EXISTS agent_ds;
```

```
CREATE DATABASE IF NOT EXISTS agent_ds;
```

创建逻辑库。

```
mysql -uroot -h192.168.xx.25 -P3307 -proot
```

```
DROP DATABASE IF EXISTS sphereex_demo;
```

```
CREATE DATABASE IF NOT EXISTS sphereex_demo;
```

通过 DistSQL 关联数据源。

```
USE sphereex_demo;

REGISTER STORAGE UNIT ds_0 (
  HOST="192.168.xx.20",
  PORT=13306,
  DB="agent_ds",
  USER="test",
  PASSWORD="test"
);
```

2. 安装测试工具

```
tar -xzf sysbench-1.0.20.tar.gz -C /usr/local
cd /usr/local/sysbench-1.0.20
./autogen.sh
./configure
make && make install
```

3. 数据准备

```
sysbench oltp_read_only --mysql-host='192.168.xx.25' --mysql-port=3307 --mysql-user=root --mysql-password='root' --mysql-db=sphereex_demo --tables=10 --table-size=1000000 --report-interval=10 --time=3600 --threads=128 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off --auto_inc=off cleanup

sysbench oltp_read_only --mysql-host='192.168.xx.25' --mysql-port=3307 --mysql-user=root --mysql-password='root' --mysql-db=sphereex_demo --tables=10 --table-size=1000000 --report-interval=10 --time=3600 --threads=128 --max-requests=0 --percentile=99 --mysql-ignore-errors="all" --rand-type=uniform --range_selects=off --auto_inc=off prepare
```

4. 压测阶段

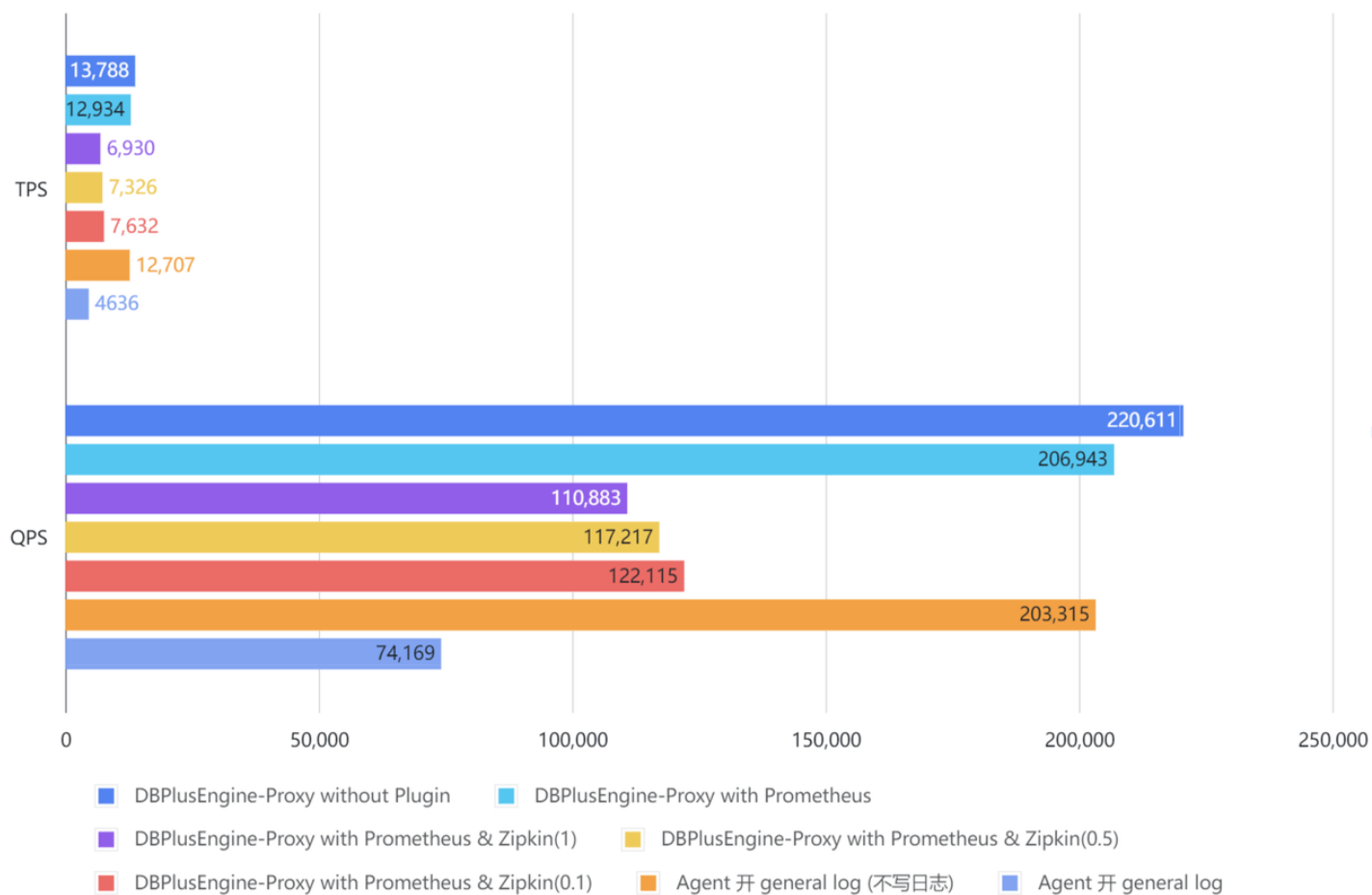
- 只读测试。

```
sysbench oltp_read_write --mysql-host='192.168.xx.25' --mysql-port=3307 --mysql-user=root --mysql-password='root' --mysql-db=sphereex_demo --tables=1 --table-size=1000000 --report-interval=5 --time=180 --threads=200 --max-requests=0 --mysql-ignore-errors="all" --range_selects=off --rand-type=uniform --auto_inc=off run
```

测试结果

应用	TPS	QPS
DBPlusEngine-Proxy without Plugin	13,788	220,611
DBPlusEngine-Proxy with Prometheus	12,934	206,943
DBPlusEngine-Proxy with Prometheus & Zipkin(1)	6,930	110,883
DBPlusEngine-Proxy with Prometheus & Zipkin(0.5)	7,326	117,217
DBPlusEngine-Proxy with Prometheus & Zipkin(0.1)	7,632	122,115
Agent 开 general log (不写日志)	12,707	203,315
Agent 开 general log	4,636	74,169

探针相关性能对比

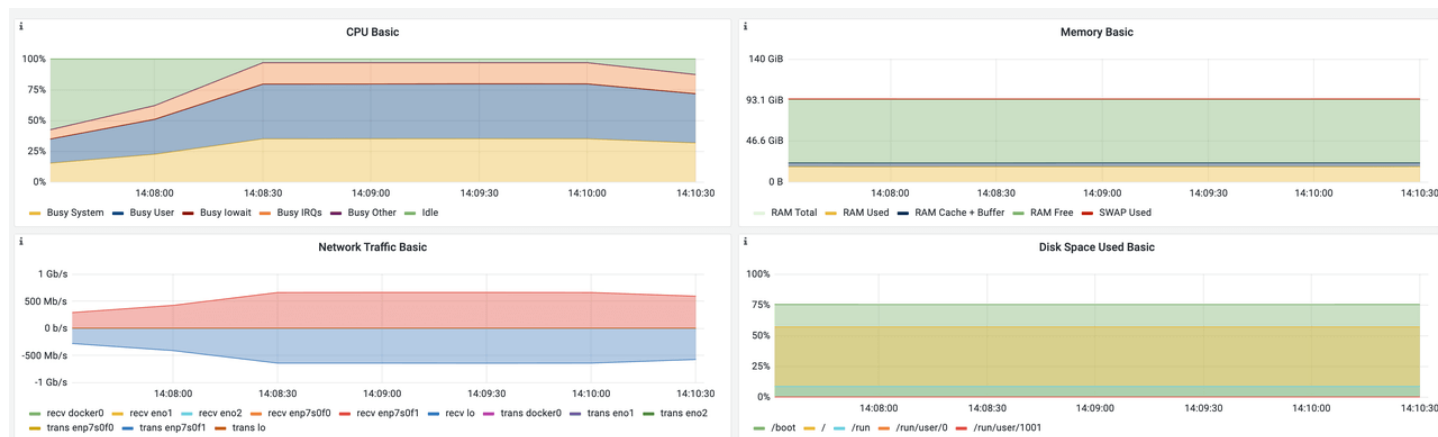


结论：在通过探针增强可观测性场景下，开启监控指标收集，性能略有下降。开启性能跟踪，性能衰减明显，随采样率不同稍有差异。用户可根据自身情况，选取合适的采样率及获取目标性能。

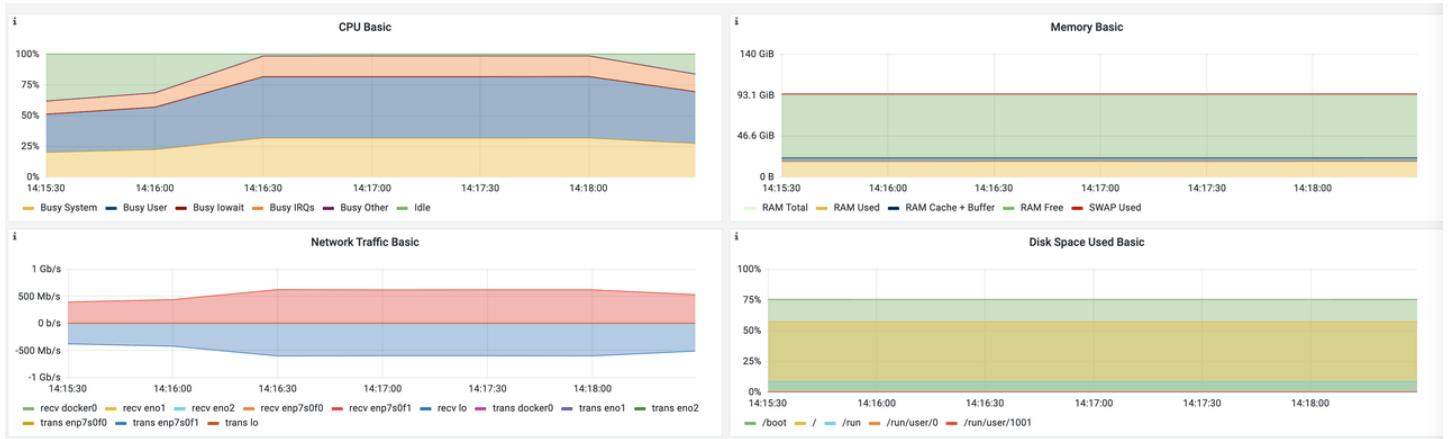
该结论仅供参考，用户可以根据实际业务情况，实际场景进行测试。

监控信息

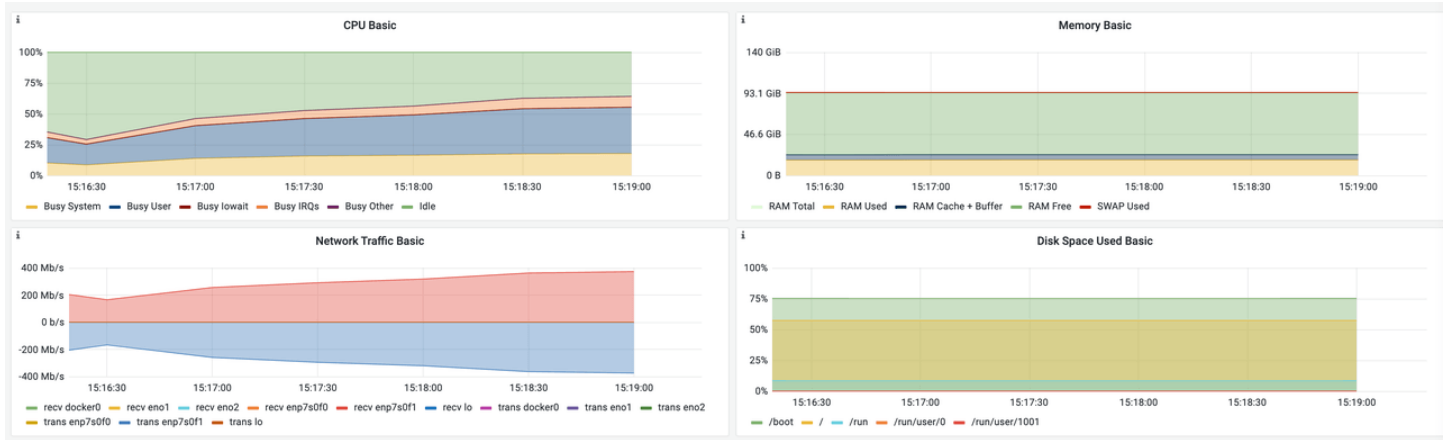
DBPlusEngine-Proxy without Plugin



DBPlusEngine-Proxy with Prometheus



DBPlusEngine-Proxy with Prometheus & Zipkin(1)



DBPlusEngine-Proxy with Prometheus & Zipkin(0.5)



DBPlusEngine-Proxy with Prometheus & Zipkin(0.1)



Agent 开 general log (不写日志)



Agent 开 general log



5.4 附录：JMH 使用说明

5.4.1 附录：JMH 使用说明

Usage: java -jar ... [regex*] [options]

[opt] means optional argument.

<opt> means required argument.

"+" means comma-separated list of values.

"time" arguments accept time suffixes, like "100ms".

Command line options usually take precedence over annotations.

- [arguments] Benchmarks to run (regex+). (default: .*)
- bm <mode> Benchmark mode. Available modes are: [Throughput/thrpt, AverageTime/avgt, SampleTime/sample, SingleShotTime/ss, All/all]. (default: Throughput)
- bs <int> Batch size: number of benchmark method calls per operation. Some benchmark modes may ignore this setting, please check this separately. (default: 1)
- e <regex+> Benchmarks to exclude from the run.
- f <int> How many times to fork a single benchmark. Use 0 to disable forking altogether. Warning: disabling forking may have detrimental impact on benchmark and infrastructure reliability, you might want to use different warmup mode instead. (default: 5)
- foe <bool> Should JMH fail immediately if any benchmark had experienced an unrecoverable error? This helps to make quick sanity tests for benchmark suites, as well as make the automated runs with checking error codes. (default: false)
- gc <bool> Should JMH force GC between iterations? Forcing the GC may help to lower the noise in GC-heavy benchmarks, at the expense of jeopardizing GC ergonomics decisions. Use with care. (default: false)
- h Display help, and exit.
- i <int> Number of measurement iterations to do. Measurement iterations are counted towards the benchmark score. (default: 1 for SingleShotTime, and 5 for all other modes)
- jvm <string> Use given JVM for runs. This option only affects forked runs.
- jvmArgs <string> Use given JVM arguments. Most options are inherited from the host VM options, but in some cases you want to pass the options only to a forked VM. Either single space-separated option line, or multiple options are accepted. This option only affects forked runs.
- jvmArgsAppend <string> Same as jvmArgs, but append these options after the already given JVM args.

- jvmArgsPrepend <string> Same as `jvmArgs`, but prepend these options before the already given JVM arg.
- l List the benchmarks that match a filter, and exit.
- lp List the benchmarks that match a filter, along with parameters, and exit.
- lprof List profilers, and exit.
- lrf List machine-readable result formats, and exit.
- o <filename> Redirect human-readable output to a given file.
- opi <int> Override operations per invocation, see `@OperationsPerInvocation` Javadoc for details. (default: 1)
- p <param={v,}*> Benchmark parameters. This option is expected to be used once per parameter. Parameter name and parameter values should be separated with equals sign. Parameter values should be separated with commas.
- prof <profiler> Use profilers to collect additional benchmark data. Some profilers are not available on all JVMs and/or all OSes. Please see the list of available profilers with `-lprof`.
- r <time> Minimum time to spend at each measurement iteration. Benchmarks may generally run longer than iteration duration. (default: 10 s)
- rf <type> Format type for machine-readable results. These results are written to a separate file (see `-rff`). See the list of available result formats with `-lrf`. (default: CSV)
- rff <filename> Write machine-readable results to a given file. The file format is controlled by `-rf` option. Please see the list of result formats for available formats. (default: `jmh-result.<result-format>`)
- si <bool> Should JMH synchronize iterations? This would significantly lower the noise in multithreaded tests, by making sure the measured part happens only when all workers are running. (default: true)
- t <int> Number of worker threads to run with. 'max' means the maximum number of hardware threads available on the machine, figured out by JMH itself. (default: 1)
- tg <int+> Override thread group distribution for asymmetric benchmarks. This option expects a comma-separated list of thread counts within the group. See `@Group/@GroupThreads` Javadoc for more information.
- to <time> Timeout for benchmark iteration. After reaching this timeout, JMH will try to interrupt the running tasks. Non-cooperating benchmarks may ignore this timeout. (default: 10 min)
- tu <TU> Override time unit in benchmark results. Available time units are: [m, s, ms, us, ns]. (default: SECONDS)

- v <mode> Verbosity mode. Available modes are: [SILENT, NORMAL, EXTRA]. (default: NORMAL)
- w <time> Minimum time to spend at each warmup iteration. Benchmarks may generally run longer than iteration duration. (default: 10 s)
- wbs <int> Warmup batch size: number of benchmark method calls per operation. Some benchmark modes may ignore this setting. (default: 1)
- wf <int> How many warmup forks to make for a single benchmark. All iterations within the warmup fork are not counted towards the benchmark score. Use 0 to disable warmup forks. (default: 0)
- wi <int> Number of warmup iterations to do. Warmup iterations are not counted towards the benchmark score. (default: 0 for SingleShotTime, and 5 for all other modes)
- wm <mode> Warmup mode for warming up selected benchmarks. Warmup modes are: INDI = Warmup each benchmark individually, then measure it. BULK = Warmup all benchmarks first, then do all the measurements. BULK_INDI = Warmup all benchmarks first, then re-warmup each benchmark individually, then measure it. (default: INDI)
- wmb <regex+> Warmup benchmarks to include in the run in addition to already selected by the primary filters. Harness will not measure these benchmarks, but only use them for the warmup.

快速入门中使用 SphereEx-DBPlusSuite 的产品：SphereEx-Boot 和 SphereEx-Console，通过这两个工具平台分别实现“数据分片”和“数据加密”两个场景。

6.1 场景一：数据分片

此场景利用 SphereEx-Boot（管理 SphereEx-DBPlusEngine 集群的命令行工具）安装 SphereEx-DBPlusEngine 集群，通过 DistSQL 命令行方式快速实现数据分片。

数据分片场景仅限于本地快速构建一个 SphereEx-DBPlusEngine 集群，并连接 MySQL 数据库，实现数据分片的场景，不包含其他组件，以及数据库的安装。

6.1.1 操作步骤

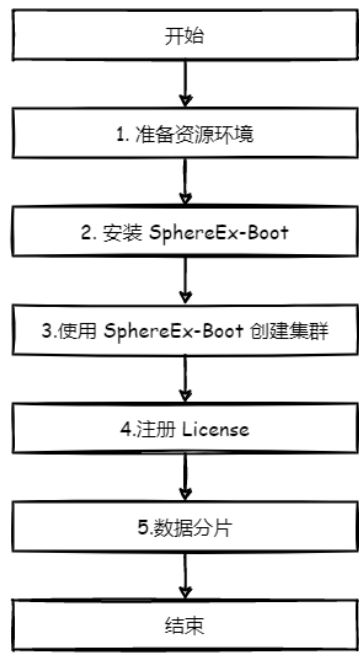


Fig. 1: 操作步骤

6.1.2 准备资源环境

	实例	IP 地址	服务端口	备注
1	SphereEx-Boot	127.0.0.1	/	SphereEx- Boot 工具
2	SphereEx-DBPlusEngine	127.0.0.1	3308	SphereEx-DBPlusEngine 集群
3	ZooKeeper	127.0.0.1	2181	ZooKeeper
4	MySQL 8.0	127.0.0.1	13306	用于分片的数据库（需提前准备）

说明：操作时需要替换为实际环境中的 IP 地址、端口。

6.1.3 安装 SphereEx-Boot

安装准备

安装 SphereEx-Boot 前，确保操作系统已经安装了如下软件：

- sshpass 1.0.0+
- Python 2.7 或者 Python 3.5+
- pip 20.0.0+

安装 SphereEx-Boot

1. 执行如下命令，安装 SphereEx-Boot。

```
# pip install sphereex-boot-1.1.0.tar.gz
Processing ./sphereex-boot-1.1.0.tar.gz
Requirement already satisfied: ansible<=2.10.7,>=2.8.0 in /usr/lib/python2.7/site-packages (from sphereex-boot==1.1.0) (2.10.7)
.....
Building wheel for sphereex-boot (setup.py) ... done
Created wheel for spex: filename=sphereex_boot-1.1.0-py2-none-any.whl size=212249
sha256=5de4a3b4f347da57697ad942f74284be38ecf12e5e0d1fefda06e5e4282c22eb
Stored in directory: /root/.cache/pip/wheels/e5/c4/d7/64a96c1f48ffabc68df6aa164beb7a93a1ad4d81aba4b751d7

Successfully built sphereex-boot

Installing collected packages: sphereex-boot
Successfully installed sphereex-boot-1.1.0
```

查看 SphereEx-Boot 安装结果

执行命令查看 SphereEx-Boot 版本，如果输出的版本为当前安装的版本，说明安装成功。

```
# spex --version
Version 1.1.0
```

6.1.4 使用 SphereEx-Boot 创建集群

本章节介绍使用 SphereEx-Boot 如何快速创建 SphereEx- DBPlusEngine 集群。

操作步骤

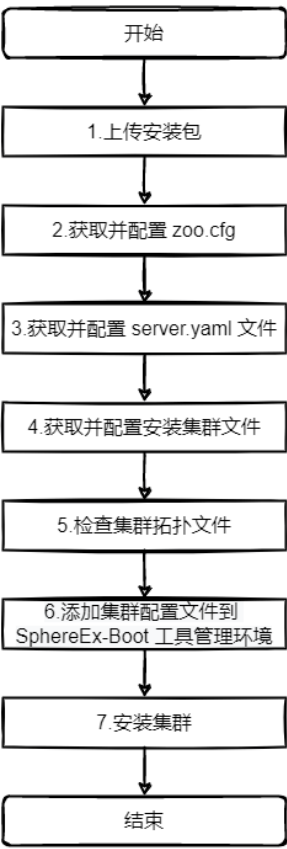


Fig. 2: 安装集群

安装集群

1. 集群规划

集群名称	IP 地址	账号	密码
demo	127.0.0.1	root	root

说明：操作时需要替换为自己的 IP 地址、登录账号和密码。

2. 上传安装包

- 使用 MySQL

将 mysql-connector-java-8.jar、sphereex-dbplusengine-proxy-1.2.0.tar.gz、apache-zookeeper-3.7.0-bin.tar.gz 安装包上传至 /root/demo 路径下。

说明：此处实际的安装包名称，会按照版本变化有所改变，按照实际情况上传相应的安装包即可。

```
# mkdir -p /root/demo
# cd /root/demo
# ll
mysql-connector-java-8.jar
```

```
sphereex-dbplusengine-proxy-1.2.0.tar.gz
apache-zookeeper-3.7.0-bin.tar.gz
```

■ 使用 PostgreSQL

使用任何 PostgreSQL 的客户端连接。如: `psql -U root -h 127.0.0.1 -p 3308`。

■ 使用 openGauss

将以 `org.opengauss` 包名为前缀的 openGauss 的 JDBC 驱动程序复制至目录 `/root/demo` 路径下。

使用任何 openGauss 的客户端连接。如: `gsql -U root -h 127.0.0.1 -p 3308`。

3. 获取并配置 zoo.cfg 文件

zoo.cfg 是 ZooKeeper 的配置文件，系统根据此配置文件中的信息，安装 Zookeeper 软件。

```
# tar -xvf apache-zookeeper-3.7.0-bin.tar.gz
# cd apache-zookeeper-3.7.0-bin/conf
# cp zoo_sample.cfg /root/demo/zoo.cfg
# cd /root/demo
# ls zoo.cfg
zoo.cfg
```

配置 zoo.cfg 文件, 将数据目录和端口参数值替换，其他参数保持默认。

```
# vi zoo.cfg
dataDir=/root/demo/zk
clientPort=2181
```

4. 获取并配置 server.yaml 文件

需要获取 conf 文件，其中此步骤需要配置的是其中的 server.yaml 文件。

server.yaml 文件是 DBPlusEngine 集群的配置文件，其中包括 Zookeeper 的相关信息、集群用户的信息，以及事物相关设置等。

```
# tar -xvf sphereex-dbplusengine-proxy-1.2.0.tar.gz
# cd sphereex-dbplusengine-proxy-1.2.0
# cp -r conf /root/demo/conf
# cd /root/demo
# ls conf/server.yaml
conf/server.yaml
```

配置 server.yaml 文件, 将以下参数的注释去掉。

```
# vi conf/server.yaml
mode:
  type: Cluster
  repository:
    type: ZooKeeper
  props:
    namespace: governance_ds
    server-lists: localhost:2181
    retryIntervalMilliseconds: 500
    timeToLiveSeconds: 60
    maxRetries: 3
    operationTimeoutMilliseconds: 500

rules:
  - !AUTHORITY
    users:
      - user: root@%
        password: root
      - user: sharding
        password: sharding
    privilege:
```

```
type: ALL_PERMITTED
- !TRANSACTION
defaultType: XA
providerType: Atomikos
```

5. 获取并配置安装集群文件

生成配置文件。

```
# spex config template --type full --output ./
# ls cluster-template.yaml
cluster-template.yaml
```

集群拓扑文件初始化, 按照实际环境修改集群拓扑文件, 修改信息如下:

```
# vi cluster-template.yaml
cluster_name: demo
install_user: root
install_password: 'root'

proxy:
  version: '1.2.0'
  install_dir: /root/demo/engine
  conf_dir: /root/demo/conf
  file: /root/demo/sphereex-dbplusengine-proxy-1.2.0.tar.gz
# agent_conf_file:
  depend_files:
    - /root/demo/mysql-connector-java-8.jar
  port: 3308
  servers:
    - host: 127.0.0.1

zookeeper:
  version: '3.7.0'
  install_dir: /root/demo/zk
  data_dir: /root/demo/zk
  conf_file: /root/demo/zoo.cfg
  file: /root/demo/apache-zookeeper-3.7.0-bin.tar.gz
  port: 2181
  servers:
    - host: 127.0.0.1
  myid: 1
```

6. 检查集群拓扑文件

```
# spex config check -f cluster-template.yaml
Proxy: The host:127.0.0.1 port:3308 missing 'agent_conf_file' value //忽略此报错即可, 此处不需要 agent。
No error with ZooKeeper
```

7. 添加集群配置文件到 SphereEx-Boot 工具管理环境

```
# spex config add -f cluster-template.yaml
Add cluster successfully
```

8. 安装集群

```
# spex cluster install --name demo
check proxy install dir exist!
Completed.....
create install directory
127.0.0.1:3308 => success
install proxy
127.0.0.1:3308 => success
copying shell file
```

```

127.0.0.1:3308 => success
copying config file
127.0.0.1:3308 => success
copying agent config file
skipped host:127.0.0.1 item:None
copying depend file
127.0.0.1:3308 => success
Completed.....
Operation ZooKeeper
check ZooKeeper install dir exist!
Completed.....
Operation ZooKeeper
create ZooKeeper install directory
127.0.0.1:2181 => success
create ZooKeeper data directory
127.0.0.1:2181 => success
install ZooKeeper
127.0.0.1:2181 => success
copy ZooKeeper config file
127.0.0.1:2181 => success
create myid
127.0.0.1:2181 => success
Completed.....

```

启动集群

```

# spex cluster start --name demo
Operation ZooKeeper
start ZooKeeper
127.0.0.1:2181 => success
/usr/bin/java
Starting zookeeper ... STARTED
ZooKeeper JMX enabled by default
Using config: /root/demo/zk/apache-zookeeper-3.7.0-bin/bin/./conf/zoo.cfg
Completed.....
start proxy
/usr/bin/java
we find java version: java8, full_version=1.8.0_322, full_path=/usr/bin/java
The port is 3308
The classpath is /root/demo/engine/sphereex-dbplusengine-proxy-1.2.0/conf::/root/demo/engine/sphereex-dbplusengine-proxy-1.2.0/lib/*:/root/demo/engine/sphereex-dbplusengine-proxy-1.2.0/ext-lib/*
main class org.apache.shardingsphere.proxy.Bootstrap 3308 /root/demo/engine/sphereex-dbplusengine-proxy-1.2.0/conf 0.0.0.0
false
Starting the SphereEx-DB-Plus-Engine ... PID: 10335
Please check the STDOUT file: /root/demo/engine/sphereex-dbplusengine-proxy-1.2.0/logs/stdout.log
Completed.....

```

查看集群运行状态

```

# spex cluster status --name demo
proxy status
127.0.0.1:3308 => success
PID:10335 PORT:3308 %CPU:16.9 %MEM:0.2 START:14:09 TIME:0:01
Results summary
+-----+-----+-----+-----+-----+-----+
| HOST | PORT | PID | %CPU | %MEM | START | TIME |
+-----+-----+-----+-----+-----+-----+
| 127.0.0.1 | 3308 | 10335 | 16.9 | 0.2 | 14:09 | 0:01 |
+-----+-----+-----+-----+-----+-----+
Completed.....
Operation ZooKeeper
ZooKeeper status

```

```
127.0.0.1:2181 => success
/usr/bin/java
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: standalone
ZooKeeper JMX enabled by default
Using config: /root/demo/zk/apache-zookeeper-3.7.0-bin/bin/./conf/zoo.cfg
Results summary
+-----+-----+-----+
| HOST | PORT | STATUS |
+-----+-----+-----+
| 127.0.0.1 | 2181 | Mode: standalone |
+-----+-----+-----+
```

登录集群

本章节介绍如何连接已经创建好的集群

```
# mysql -uroot -p -P3308 -h127.0.0.1
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.22 sphereex-dbplusengine-proxy-1.2.0

Copyright (c) 2000, 2021, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

6.1.5 注册 License

1. 上传 License

将获得的 License 文件上传至默认的存储目录：/demo/conf。

```
# scp -r sphere.license root@10.10.10.1:/root/demo/conf
```

2. 查看 License-key 内容。

```
# cat /root/demo/conf/sphere.license
Ic5OXgAAACMAAAACAAAACwAAAxjcmVEaWdlc3RTSEEtNTEy
```

3. 注册 License。

登录 SphereEx- DBPlusEngine 集群，注册 License。

```
# mysql -uroot -p -P3308 -h127.0.0.1
sql> REGISTER LICENSE '[Ic5OXgAAACMAAAACAAAACwAAAxjcmVEaWdlc3RTSEEtNTEy]';
Query OK, 0 rows affected (1.29 sec)
```

4. 查看 License 是否生效。

```
sql> SHOW LICENSE INFO;
+-----+-----+-----+-----+-----+-----+-----+
| company_name | instance_count | resource_count | expiry_date | generation_data | status | license |
+-----+-----+-----+-----+-----+-----+-----+
| sphereextest | 8 | 8 | 2023-01-01 00:00:00 | 2022-09-08 18:00:00 | valid | Ic5OXgAAACMAAAACAAAACwAAAxjcmVEaWdlc3RTSEEtNTEy |
+-----+-----+-----+-----+-----+-----+-----+
```

6.1.6 数据分片

本章节将介绍使用 DistSQL（命令工具）如何创建数据分片。也可以使用 SphereEx-Console（SphereEx 企业数据服务可视化平台）实现数据分片功能。

操作步骤

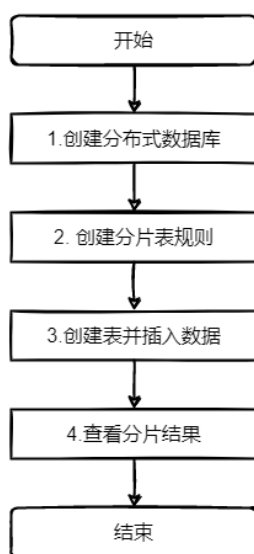


Fig. 3: 数据分片操作步骤

创建分片

1. 创建分布式数据库

a. 创建数据源

登陆 MySQL 数据库: 127.0.0.1/13306，创建数据库 test01 和 test02，如：

```
# mysql -uroot -proot -P13306 -h127.0.0.1
sql> CREATE DATABASE test01;
Query OK, 1 row affected (0.00 sec)
```

```
sql> CREATE DATABASE test02;
Query OK, 1 row affected (0.00 sec)
```

```
sql> SHOW DATABASES;
```

```
+-----+
| Database |
+-----+
| information_schema |
| mysql           |
| performance_schema |
| sys             |
| test01          |
| test02          |
+-----+
5 rows in set (0.01 sec)
```

b. 创建逻辑库

登陆 Proxy

```
# mysql -uroot -proot -P3308 -h127.0.0.1
```

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 37

Server version: 5.7.22 SphereEx-DBPlusEngine-Proxy 1.2.0 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
sql>
```

创建逻辑库

```
sql> CREATE DATABASE testdb;
```

Query OK, 0 rows affected (0.65 sec)

查看逻辑库

```
sql> SHOW DATABASES;
```

```
+-----+
| schema_name |
+-----+
| mysql       |
| information_schema |
| performance_schema |
| sys        |
| testdb     |
+-----+
5 rows in set (0.00 sec)
```

c. 添加数据源

切换至逻辑库 testdb

```
sql> USE testdb;
```

Database changed

添加数据源

```
sql> REGISTER STORAGE UNIT test01 (
```

```
-> HOST="127.0.0.1",
-> PORT=13306,
-> DB="test01",
-> USER="root",
```



```
-> PASSWORD="root"
-> ),test02 (
->   HOST="127.0.0.1",
->   PORT=13306,
->   DB="test02",
->   USER="root",
->   PASSWORD="root"
-> );
Query OK, 0 rows affected (0.09 sec)
```

d. 查看数据源信息

```
sql> SHOW STORAGE UNITS\G
***** 1. row *****
      name: test02
      type: MySQL
      host: 127.0.0.1
      port: 13306
      db: test02
connection_timeout_milliseconds: 30000
idle_timeout_milliseconds: 60000
max_lifetime_milliseconds: 2100000
max_pool_size: 50
min_pool_size: 1
read_only: false
other_attributes: {"dataSourceProperties":{"cacheServerConfiguration":"true","elideSetAutoCommits":"true",
"useServerPrepStmts":"true","cachePrepStmts":"true","useSSL":"false","rewriteBatchedStatements":"true",
"cacheResultSetMetadata":"false","useLocalSessionState":"true","maintainTimeStats":"false","prepStmtCacheSize":"8192",
"tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048","serverTimezone":"UTC","netTimeoutForStreamingResults":"0",
"zeroDateTimeBehavior":"round"},"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,
"leakDetectionThreshold":0,"registerMbeans":false,"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false}
***** 2. row *****
      name: test01
      type: MySQL
      host: 127.0.0.1
      port: 13306
      db: test01
connection_timeout_milliseconds: 30000
idle_timeout_milliseconds: 60000
max_lifetime_milliseconds: 2100000
max_pool_size: 50
min_pool_size: 1
read_only: false
other_attributes: {"dataSourceProperties":{"cacheServerConfiguration":"true","elideSetAutoCommits":"true",
"useServerPrepStmts":"true","cachePrepStmts":"true","useSSL":"false","rewriteBatchedStatements":"true",
"cacheResultSetMetadata":"false","useLocalSessionState":"true","maintainTimeStats":"false","prepStmtCacheSize":"8192",
"tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048","serverTimezone":"UTC","netTimeoutForStreamingResults":"0",
"zeroDateTimeBehavior":"round"},"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,
"leakDetectionThreshold":0,"registerMbeans":false,"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false}
2 rows in set (0.00 sec)
```

至此，分布式数据库创建完成。

2. 创建分片表规则

选择逻辑库

```
sql> USE testdb;
Database changed
```

创建数据分片规则

```
sql> CREATE SHARDING TABLE RULE t_user(
-> STORAGE_UNITS(test01,test02),
-> SHARDING_COLUMN=user_id,
-> TYPE(NAME="hash_mod",PROPERTIES("sharding-count"=4))
-> );
Query OK, 0 rows affected (0.07 sec)
```

查看数据分片规则

```
sql> SHOW SHARDING TABLE RULE t_user\G
***** 1. row *****
      table: t_user
      actual_data_nodes:
      actual_data_sources: test01,test02
      database_strategy_type:
      database_sharding_column:
      database_sharding_algorithm_type:
      database_sharding_algorithm_props:
      table_strategy_type: STANDARD
      table_sharding_column: user_id
      table_sharding_algorithm_type: hash_mod
      table_sharding_algorithm_props: sharding-count=4
      key_generate_column:
      key_generator_type:
      key_generator_props:
      auditor_types:
      allow_hint_disable:
1 row in set (0.01 sec)
```

3. 创建表并插入数据

a. 创建表

```
sql> CREATE TABLE `t_user` (
-> `user_id` int NOT NULL,
-> `order_id` int NOT NULL,
-> `status` varchar(45) DEFAULT NULL,
-> PRIMARY KEY (`user_id`)
-> );
Query OK, 0 rows affected (0.28 sec)
```

b. 插入数据

```
sql> INSERT INTO t_user VALUES
-> (1,1,'active'),
-> (2,2,'active'),
-> (3,3,'active'),
-> (4,4,'active');
Query OK, 4 rows affected (0.10 sec)
```

c. 预览分片信息

```
sql> PREVIEW SELECT * FROM t_user;
+-----+-----+
| data_source_name | actual_sql |
+-----+-----+
| test01          | SELECT * FROM t_user_0 UNION ALL SELECT * FROM t_user_2 |
| test02          | SELECT * FROM t_user_1 UNION ALL SELECT * FROM t_user_3 |
+-----+-----+
2 rows in set (0.04 sec)
```

4. 查看分片结果

登陆数据库 test01，查看实际分片数据

```
# mysql -uroot -proot -P13306 -h127.0.0.1
```

```
sql> use test01;
```

```
sql> select * from t_user_0;
```

```
+-----+-----+-----+
| user_id | order_id | status |
+-----+-----+-----+
|    4    |    4    | active |
+-----+-----+-----+
```

```
1 row in set (0.00 sec)
```

```
sql> select * from t_user_2;
```

```
+-----+-----+-----+
| user_id | order_id | status |
+-----+-----+-----+
|    2    |    2    | active |
+-----+-----+-----+
```

```
1 row in set (0.00 sec)
```

登陆数据库 test02，查看实际分片数据

```
sql> use test02;
```

```
sql> select * from t_user_1;
```

```
+-----+-----+-----+
| user_id | order_id | status |
+-----+-----+-----+
|    1    |    1    | active |
+-----+-----+-----+
```

```
1 row in set (0.00 sec)
```

```
sql> select * from t_user_3;
```

```
+-----+-----+-----+
| user_id | order_id | status |
+-----+-----+-----+
|    3    |    3    | active |
+-----+-----+-----+
```

```
1 row in set (0.00 sec)
```

6.2 场景二：数据加密

此场景利用 SphereEx-Console（SphereEx 企业数据服务可视化平台）安装 SphereEx-DBPlusEngine 集群，通过 SphereEx-Console 方式快速实现数据加密。

数据加密场景仅限于利用 SphereEx-Console 快速构建一个本地 SphereEx-DBPlusEngine 集群，并链接一个 MySQL 数据库，实现数据加密的场景（不包含其他组件，以及数据库的安装）。

6.2.1 操作步骤

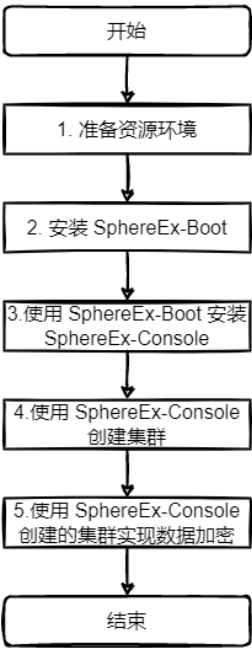


Fig. 4: 创建数据加密流程

6.2.2 准备资源环境

	实例	IP 地址	服务端口	备注
1	SphereEx-Boot	127.0.0.1	/	SphereEx-Boot 工具
2	SphereEx-Console	127.0.0.1	8088	SphereEx-Console 工具
3	SphereEx-DBPlusEngine	127.0.0.1	3212	SphereEx-DBPlusEngine 集群
4	MySQL 8.0	127.0.0.1	3306	通过 SphereEx-Boot 安装 Sphere-Console 所需数据库 (主库)
5	MySQL 8.0	127.0.0.1	3307	通过 SphereEx-Boot 安装 Sphere-Console 所需数据库 (从库)
6	ZooKeeper	127.0.0.1	3666/3667	ZooKeeper
7	MySQL 8.0	/	13306	用于加密的数据库 (需提前准备)

说明：操作时需要替换为自己的 IP 地址、登录账号和密码。

6.2.3 安装 SphereEx-Boot

请参见场景一：安装 SphereEx-Boot。

如果已经安装 SphereEx-Boot，此步骤可忽略。

6.2.4 安装 SphereEx-Console

安装 SphereEx-Console 前，确保操作系统已经安装 JDK 1.8。

1. 查看 JDK 版本

```
# java -version
openjdk version "1.8.0_312"
OpenJDK Runtime Environment (build 1.8.0_312-b07)
OpenJDK 64-Bit Server VM (build 25.312-b07, mixed mode)
```

2. 导出配置文件

```
# mkdir console
# cd console
# spex console export
# ls console_install.yaml
console_install.yaml
```

3. 修改配置文件

按照如下参数值进行修改，或者按照实际的环境修改，未提及参数保持默认即可。

```
# vi console_install.yaml
##### SphereEx-Console Deploy Configuration Example #####

# This file is an example configuration file.
# Here you can configure for the SphereEx-Console deploy.
# Deploy to remote must use ssh

#===== Mysql Deploy Configuration =====

# Required, Whether to install a new Mysql server
# If true Mysql master-slave will be installed

install_mysql: true

# If install Mysql is true the param is required

mysql:
  master:
    # install on local or remote
    file: /root/console/mysql.tar.xz
    host: 127.0.0.1
    listen_port: 3306
    ssh_user: root
    ssh_password: root
    ssh_port: 22
    install_dir: /root/console
```

```

slave:
# install on local or remote
host: 127.0.0.1
file: /root/console/mysql.tar.xz
# If two Mysql servers are on the same machine, listen port must be different
listen_port: 3307
ssh_user: root
ssh_password: root
ssh_port: 22
install_dir: /root/console

#===== Console Deploy Configuration =====
console:
# install on local or remote
ssh_user: root
ssh_password: root
ssh_port: 22
host: 127.0.0.1
# Required, SphereEx-Console file. for example:/root/console/SphereEx-Console.tar.gz
file: /root/console/sphereex-console.1.1.0.tar.gz
# Required, deployment directory, remote or local
install_dir: /root/console
# Required, deployment directory, remote or local
console_config:
server:
port: 8088
# software home path
software:
home: /opt/software

```

4. 执行安装 console 命令

```

# spex console install -n console_demo -c console_install.yaml
create install directory
host:127.0.0.1 status:success
install mysql8
host:127.0.0.1 status:success
push mysql conf
host:127.0.0.1 status:success
create install directory
host:127.0.0.1 status:success
install mysql8
host:127.0.0.1 status:success
push mysql conf
host:127.0.0.1 status:success
start mysql
host:127.0.0.1 status:success
start mysql
host:127.0.0.1 status:success
push mysql shell
host:127.0.0.1 status:success
opt shell
host:127.0.0.1 status:success
push mysql shell
host:127.0.0.1 status:success
opt shell
host:127.0.0.1 status:success
create install directory
host:127.0.0.1 status:success
push console file
host:127.0.0.1 status:success
install console

```

```
host:127.0.0.1 status:success
push console yml
host:127.0.0.1 status:success
remove install file
host:127.0.0.1 status:success
pull console sql
host:127.0.0.1 status:success
pull console sql data
host:127.0.0.1 status:success
push console sql
host:127.0.0.1 status:success
init console mysql task
host:127.0.0.1 status:success
```

5. 启动 console。

进入 console 的安装目录，执行启动命令

```
# cd sphereex-console-1.1.0
# bin/start.sh
Starting the SphereEx-Console ...
Please check the STDOUT file: /root/console/sphereex-console-1.1.0/logs/stdout.log
```

6. 访问 console

使用服务器的外网 IP 地址加端口通过浏览器访问 SphereEx-Console 平台，使用配置文件中账号和密码登录。

例如：127.0.0.1:8088

默认用户名：admin

默认密码：admin



Fig. 5: Console 登录页

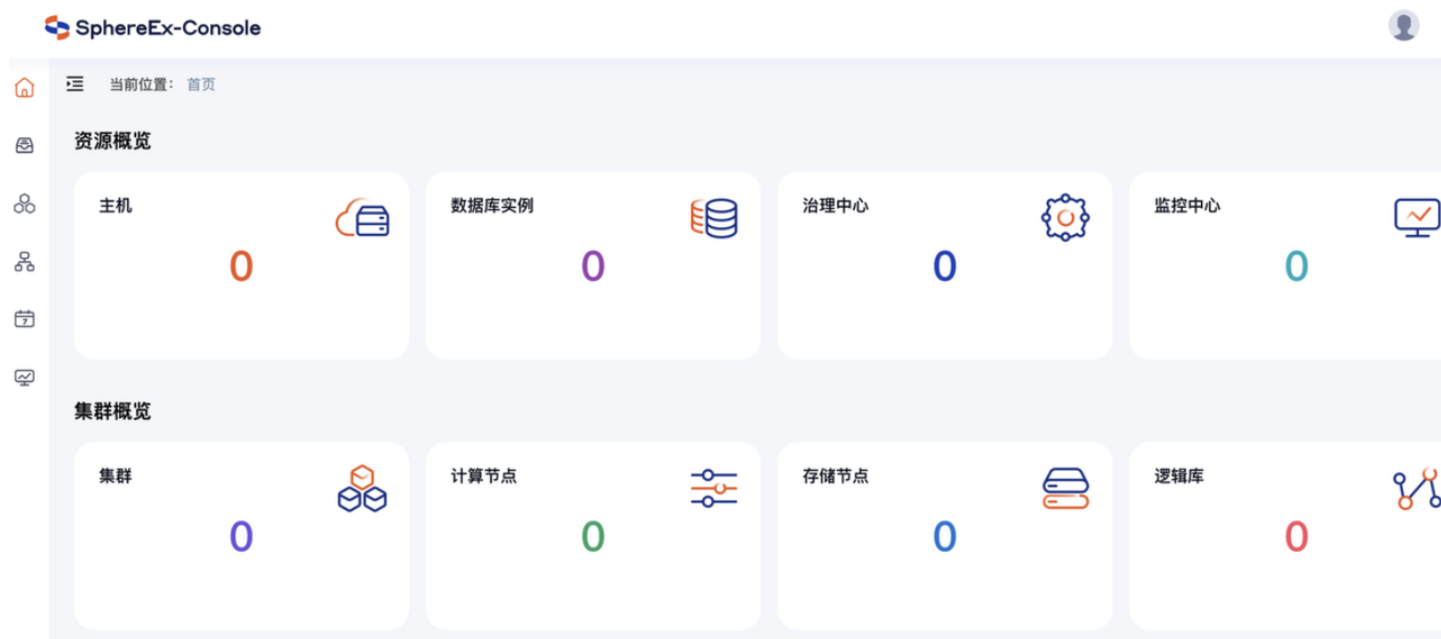


Fig. 6: Console 概览页

6.2.5 使用 SphereEx-Console 创建集群

此部分内容主要介绍使用 SphereEx-Console 如何创建 SphereEx-DBPlusEngine 集群，以及实现数据加密。

操作步骤

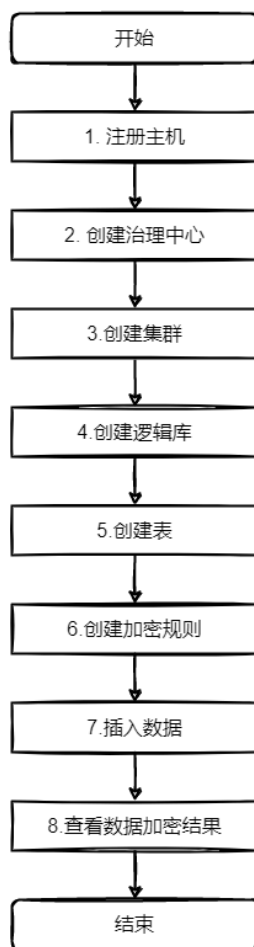


Fig. 7: Console 创建集群

1. 注册主机

左侧导航栏选择“资源-主机”，点击“注册主机”，填写所需信息，点击“确定”，完成注册主机。

单台注册

批量注册

* 主机 IP

* 主机用户

test

* 用户口令

.....

* SSH 端口

22

主机名称

主机标签

test

描述

0/500

探活

取消

确定

Fig. 8: 注册主机

2. 创建治理中心

将 ZooKeeper 的安装包上传至 /opt/software 路径下

左侧导航栏选择“资源-治理中心”，点击“新建”，填写所需信息，点击“确定”，完成治理中心的创建。

创建治理中心集群

* 治理中心类型

ZooKeeper

* 治理中心名称

test_zk

* 治理中心软件

apache-zookeeper-3.7.0-bin.tar.gz

描述

0/500

Fig. 9: 创建治理中心

配置监控

监控中心

请选择

监控插件软件

请选择

添加节点

节点 IP	节点端口	监控插件端口	内部通讯端口	内部选举端口	安装目录	操作
	3666		2888	3888		
	3667		2889	3889		删除

取消

确定

Fig. 10: 配置监控

3. 创建集群

将 SphereEx-DBPlusEngine 的安装包上传至 /opt/software 路径下。
填写必要的信息，治理中心选择之前创建的治理中心。

* 集群名称

集群test001

描述

0/500

治理中心地址

:3666,:3667

* 命名空间

namespace001

监控中心

请选择

日志中心

请选择

Fig. 11: 创建集群

添加计算节点

最少需要有一个节点安装成功

* 计算节点软件

SphereEx-DBPlusEngine-Proxy-.tar.gz

filebeat 软件

请选择

节点 IP	节点端口	安装目录	监控插件端口	Logstash	节点名称	操作	操作结果
	3212	/data/c	9196	请选择		安装 卸载	

Fig. 12: 增加计算节点

所需信息填写完成后，点击“安装”按钮。
“安装”按钮变为灰色，操作结果显示“安装成功”，说明集群创建成功，点击确定，返回集群列表。

节点 IP	节点端口	安装目录	监控插件端口	Logstash	节点名称	操作	操作结果
	3212	/data/c	9196	请选择		安装 卸载	安装成功



Fig. 13: 安装成功

在“注册存储节点”界面，选择需要做分片的数据库，并填写所需的信息。

* 所属实例

MySQL | :13306

* 数据库地址

* 数据库端口

13306

* 数据库类型

MySQL

* 用户名

root

* 口令

* 数据库

ss03

* 节点名称

dbtest03

扩展参数 (&分割)

URL 连接串

jdbc:mysql://:13306/ss03?

探活

确定

取消

Fig. 14: 注册存储节点

点击探活，探活成功后，点击确定。
使用相同方法注册第二个数据库。

节点管理

返回

集群test001

计算节点

存储节点

注册存储节点

刷新

节点名称	所属逻辑库	所属集群	数据库地址	数据库端口	数据库类型	数据库	用户名	操作
dbtest04		集群test001		13306	MySQL	ss04	root	探活 修改 删除
dbtest03		集群test001		13306	MySQL	ss03	root	探活 修改 删除

Fig. 15: 节点管理

4. 创建逻辑库

在左侧导航栏点击“逻辑库”，进入逻辑库列表，选择集群名称“集群 test001”，点击“创建逻辑库”。
输入“逻辑库名称”，选择存储节点，点击符号“>”，点击确定，将数据库和集群关联。

创建逻辑库

×

* 逻辑库名称

logical001

✓

* 所属集群

集群test001

▼

存储节点

☐ 可选节点

0/0

无数据

<

>

☐ 已选节点

0/2

☐ dbtest03

13306/ss03

 +

☐ dbtest04

13306/ss04

 +

确定

取消

Fig. 16: 节点管理

5. 创建表

选择“对象”，选择集群和逻辑库，点击“新建表”。
根据参考示例，输入建表语句，点击“确定”。

```
sql> CREATE TABLE t_encrypt(  
  id INT(8),  
  mobile VARCHAR(50),  
  id_card_no VARCHAR(50)  
);
```

169

新建表

[修改建表语句](#)

集群	集群test001
逻辑库	logical001
* 建表语句	<pre>CREATE TABLE t_encrypte(id INT(8), mobile VARCHAR(50), id_card_no VARCHAR(50));</pre>
参考示例	

Fig. 17: 新建表

6. 创建加密规则

选择创建的数据表：t_encrypt，点击“添加加密”→选择“常规”加密→点击“添加加密插件”

插件/添加加密

表名 t_encrypt

表类型 ☒ 常规 ☐ 自定义

☒ 是否使用加密列表查询



Fig. 18: 添加加密插件

选择“加密算法”，选择“需要加密的列”，勾选“存储明文”，点击“添加”→点击“确定”。

添加插件



* 加密算法

MD5



* 需要加密的列

mobile



存储明文



添加

Fig. 19: 添加插件

7. 插入数据

登录 SphereEx-DBPlusEngine 集群，执行插入数据命令。

```
# mysql -uroot -proot -P3212 -h127.0.0.1
sql> use logical001;
sql> INSERT INTO t_encrypt (id, mobile_plain, id_card_no)
-> VALUES (1, 18236483857, 220605194709308170),
-> (2, 15686689114, 360222198806088804),
-> (3, 13983621809, 430204195612042092);
Query OK, 3 rows affected (0.02 sec)
```

8. 查看数据加密结果

登录 SphereEx-DBPlusEngine 集群，查看数据是密文显示。

说明：如果需要 SphereEx-DBPlusEngine 集群中显示明文，不勾选“是否使用加密列查询”即可。

```
# mysql -uroot -proot -P3212 -h127.0.0.1
sql> use logical001;
sql> SELECT * FROM t_encrypt;
```

id	mobile	id_card_no
1	6f655ea3b73e32d54ae2d34d672c0c55	220605194709308170
2	a2193d0e8de4c2c4f166448e22ba97ae	360222198806088804
3	3cf1fd442630071ca664bbd5e683f7f6	430204195612042092

```
3 rows in set (0.00 sec)
```

登录 MySQL 中查看数据是密文显示，见 mobile_cipher。

因为勾选了“存储明文”，手机号一列在数据库中同时保留了明文存储，见 mobile_plain。

```
# mysql -uroot -proot -P13306 -h127.0.0.1
sql> use ss03;
sql> select * from t_encrypt;
```

id	mobile_cipher	mobile_plain	id_card_no
1	6f655ea3b73e32d54ae2d34d672c0c55	18236483857	220605194709308170
2	a2193d0e8de4c2c4f166448e22ba97ae	15686689114	360222198806088804
3	3cf1fd442630071ca664bbd5e683f7f6	13983621809	430204195612042092

7.1 数据分片

传统的将数据集中存储至单一节点的解决方案，在性能、可用性和运维成本这三方面已经难于满足海量数据的场景。

从性能方面来说，由于关系型数据库大多采用 B+ 树类型的索引，在数据量超过阈值的情况下，索引深度的增加也将使得磁盘访问的 IO 次数增加，进而导致查询性能的下降；同时，高并发访问请求也使得集中式数据库成为系统的最大瓶颈。

从可用性的方面来讲，服务化的无状态性，能够达到较小成本的随意扩容，这必然导致系统的最终压力都落在数据库之上。而单一的数据节点，或者简单的主从架构，已经越来越难以承担。数据库的可用性，已成为整个系统的关键。

从运维成本方面考虑，当一个数据库实例中的数据达到阈值以上，对于 DBA 的运维压力就会增大。数据备份和恢复的时间成本都将随着数据量的大小而愈发不可控。

在传统的关系型数据库无法满足互联网场景需要的情况下，将数据存储至原生支持分布式的 NoSQL 的尝试越来越多。但 NoSQL 对 SQL 的不兼容性以及生态圈的不完善，使得它们在与关系型数据库的博弈中始终无法完成致命一击，而关系型数据库的地位却依然不可撼动。

7.1.1 概述

SphereEx-DBPlusEngine 分片插件可最小化透明化分库分表所带来的影响，可让用户像使用一个数据库一样使用水平分片之后的数据库集群，目前可对多种数据库实现分片方案。

基本概念

表

表是透明化数据分片的关键概念。SphereEx-DBPlusEngine 通过提供多样化的表类型，适配不同场景下的数据分片需求。

■ 逻辑表

相同结构的水平拆分数据库（表）的逻辑名称，是 SQL 中表的逻辑标识。例：订单数据根据主键尾数拆分为 10 张表，分别是 t_order_0 到 t_order_9，他们的逻辑表名为 t_order。

■ 真实表

在水平拆分的数据库中真实存在的物理表。即上个示例中的 t_order_0 到 t_order_9。

■ 表关系

指分片规则一致的一组分片表。使用绑定表进行多表关联查询时，必须使用分片键进行关联，否则会出现笛卡尔积关联或跨库关联，从而影响查询效率。例如：t_order 表和 t_order_item 表，均按照 order_id 分片，并且使用 order_id 进行关联，则此两张表互为绑定表关系。绑定表之间的多表关联查询不会出现笛卡尔积关联，关联查询效率将大大提升。举例说明，如果 SQL 为：

```
SELECT i.* FROM t_order o JOIN t_order_item i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
```

在不配置绑定表关系时，假设分片键 order_id 将数值 10 路由至第 0 片，将数值 11 路由至第 1 片，那么路由后的 SQL 应该为 4 条，它们呈现为笛卡尔积：

```
SELECT i.* FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
```

```
SELECT i.* FROM t_order_0 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
```

```
SELECT i.* FROM t_order_1 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
```

```
SELECT i.* FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
```

在配置绑定表关系，并且使用 order_id 进行关联后，路由的 SQL 应该为 2 条：

```
SELECT i.* FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
```

```
SELECT i.* FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
```

其中 t_order 表由于指定了分片条件，SphereEx-DBPlusEngine 将会以它作为整个绑定表的主表。所有路由计算将会只使用主表的策略，那么 t_order_item 表的分片计算将会使用 t_order 的条件。

■ 广播表

指所有的分片数据源中都存在的表，表结构及其数据在每个数据库中均完全一致。适用于数据量不大且需要与海量数据的表进行关联查询的场景，例如：字典表。

■ 单表

指所有的分片数据源中仅唯一存在的表。适用于数据量不大且无需分片的表。

存储资源

数据分片的最小单元，由数据源名称和真实表组成。例：ds_0.t_order_0。逻辑表与真实表的映射关系，可分为均匀分布和自定义分布两种形式。

■ 均匀分布

指数据表在每个数据源内呈现均匀分布的态势，例如：

```
db0
├── t_order0
└── t_order1
db1
├── t_order0
└── t_order1
```

存储资源的配置如下：

```
db0.t_order0, db0.t_order1, db1.t_order0, db1.t_order1
```

■ 自定义分布

指数据表呈现有特定规则的分布，例如：

```
db0
├── t_order0
└── t_order1
db1
├── t_order2
├── t_order3
└── t_order4
```

存储资源的配置如下：

```
db0.t_order0, db0.t_order1, db1.t_order2, db1.t_order3, db1.t_order4
```

分片

■ 分片键

用于将数据库（表）水平拆分的数据库字段。例：将订单表中的订单主键的尾数取模分片，则订单主键为分片字段。SQL 中如果无分片字段，将执行全路由，性能较差。除了对单分片字段的支持，SphereEx-DBPlusEngine 也支持根据多个字段进行分片。

■ 分片算法

用于将数据分片的算法，支持 =、>=、<=、>、<、BETWEEN 和 IN 进行分片。分片算法可由开发者自行实现，也可使用 SphereEx-DBPlusEngine 内置的分片算法语法糖，灵活度非常高。分片算法具体说明请参考[分片算法](#)。

■ 自动化分片算法

分片算法语法糖，用于便捷的托管所有数据节点，使用者无需关注真实表的物理分布。包括取模、哈希、范围、时间等常用分片算法的实现。

- 自定义分片算法提供接口让应用开发者自行实现与业务实现紧密相关的分片算法，并允许使用者自行管理真实表的物理分布。自定义分片算法又分为：

标准分片算法：

用于处理使用单一键作为分片键的 =、IN、BETWEEN AND、>、<、>=、<= 进行分片的场景。

复合分片算法：

用于处理使用多键作为分片键进行分片的场景，包含多个分片键的逻辑较复杂，需要应用开发者自行处理其中的复杂度。

Hint 分片算法：

用于处理使用 Hint 行分片的场景。

■ 分片策略

包含分片键和分片算法，由于分片算法的独立性，将其独立抽离。真正可用于分片操作的是分片键 + 分片算法，也就是分片策略。

■ 强制分片路由

对于分片字段并非由 SQL 而是其他外置条件决定的场景，可使用 SQL Hint 注入分片值。例：按照员工登录主键分库，而数据库中并无此字段。SQL Hint 支持通过 Java API 和 SQL 注释两种方式使用。详情请参见[强制分片路由](#)。

■ 行表达式

行表达式是为了解决配置的简化与一体化这两个主要问题。在繁琐的数据分片规则配置中，随着数据节点的增多，大量的重复配置使得配置本身不易被维护。通过行表达式可以有效地简化数据节点配置工作量。

对于常见的分片算法，使用 Java 代码实现并不有助于配置的统一管理。通过行表达式书写分片算法，可以有效地将规则配置一同存放，更加易于浏览与存储。

行表达式的使用非常直观，只需要在配置中使用 `${ expression }` 或 `$->{ expression }` 标识行表达式即可。目前支持数据节点和分片算法这两个部分的配置。行表达式的内容使用的是 Groovy 的语法，Groovy 能够支持的所有操作，行表达式均能够支持。例如：

`${begin..end}` 表示范围区间 `${[unit1, unit2, unit_x]}` 表示枚举值，行表达式中如果出现连续多个 `${ expression }` 或 `$->{ expression }` 表达式，整个表达式最终的结果将会根据每个子表达式的结果进行笛卡尔组合。

例如，以下行表达式：

```
${['online', 'offline']}_table${1..3}
```

最终会解析为：

```
online_table1, online_table2, online_table3, offline_table1, offline_table2, offline_table3
```

■ 分布式主键

传统数据库软件开发中，主键自动生成技术是基本需求。而各个数据库对于该需求也提供了相应的支持，比如 MySQL 的自增键，Oracle 的自增序列等。数据分片后，不同数据节点生成全局唯一主键是非常棘手的问题。同一个逻辑表内的不同实际表之间的自增键由于无法互相感知而产生重复主键。虽然可通过约束自增主键初始值和步长的方式避免碰撞，但需引入额外的运维规则，使解决方案缺乏完整性和可扩展性。

目前有许多第三方解决方案可以完美解决这个问题，如 UUID 等依靠特定算法自生成不重复键，或者通过引入主键生成服务等。为了方便用户使用、满足不同用户不同使用场景的需求，SphereEx-DBPlusEngine 不仅提供了内置的分布式主键生成器，例如 UUID、SNOWFLAKE，还抽离出分布式主键生成器的接口，方便用户自行实现自定义的自增主键生成器。

- 分布式事务
 - XA
 - BASE

其他概念

■ SQL 解析

分为词法解析和语法解析。先通过词法解析器将 SQL 拆分为一个个不可再分的单词。再使用语法解析器对 SQL 进行理解，并最终提炼出解析上下文。解析上下文包括表、选择项、排序项、分组项、聚合函数、分页信息、查询条件以及可能需要修改的占位符的标记。

■ SQL 路由

根据解析上下文匹配用户配置的分片策略，并生成路由路径。目前支持分片路由和广播路由。

■ SQL 改写

将 SQL 改写为在真实数据库中可以正确执行的语句。SQL 改写分为正确性改写和优化改写。

■ SQL 执行

通过多线程执行器异步执行。

■ 结果归并

将多个执行结果集归并以便于通过统一的 JDBC 接口输出。结果归并包括流式归并、内存归并和使用装饰者模式的追加归并这几种方式。

■ 查询优化

由 Federation 执行引擎提供支持，对关联查询、子查询等复杂查询进行优化，同时支持跨多个数据库实例的分布式查询，内部使用关系代数优化查询计划，通过最优计划查询出结果。

适用场景

海量数据高并发的 OLTP 场景

由于关系型数据库大多采用 B+ 树类型的索引，在数据量超过阈值的情况下，索引深度的增加也将使得磁盘访问的 IO 次数增加，进而导致查询性能的下降。通过 SphereEx-DBPlusEngine 数据分片，按照某个业务维度，将存放在单一数据库中的数据分散地存放至多个数据库或表中，可以达到提升性能的效果，满足高并发的 OLTP 场景下的性能要求。

海量数据实时分析 OLAP 场景

在传统的数据库架构中，如果用户想要进行数据分析，需要先使用 ETL 工具，将数据同步至数据平台中，然后再进行数据分析，使用 ETL 工具会导致数据分析的实效性大打折扣。SphereEx-DBPlusEngine 提供静态入口以及异构语言的支持，独立于应用程序部署，适用于实时分析的 OLAP 场景。

使用前提

- 服务器中已安装 SphereEx-DBPlusEngine 和数据库集群，且服务运行正常；
- 相关使用限制需提前了解，详见下方“使用限制”内容。

使用限制

兼容常用的路由至单数据节点的 SQL；由于路由至多数据节点的 SQL 由于场景相对复杂，分为稳定支持、实验性（不推荐使用）支持和不支持这三种情况。

分类	描述
稳定支持	常规查询，子查询，分页查询，运算表达式中包含分片键
实验性支持	子查询（子查询和外层未指定分片键/键值不一致），跨库关联查询
不支持	CASE WHEN，Oracle 和 SQL Server 的部分分页查询

已稳定支持

全面支持 DML、DDL、DCL、TCL 和常用 DAL。支持分页、去重、排序、分组、聚合、表关联等复杂查询。支持 PostgreSQL 和 openGauss 数据库 SCHEMA DDL 和 DML 语句。详情参考[兼容性列表](#)。

常规查询

- SELECT 主语句

```
SELECT select_expr [, select_expr ...] FROM table_reference [, table_reference ...]
[WHERE predicates]
[GROUP BY {col_name | position} [ASC | DESC], ...]
[ORDER BY {col_name | position} [ASC | DESC], ...]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

- select_expr

```
* |
[DISTINCT] COLUMN_NAME [AS] [alias] |
(MAX | MIN | SUM | AVG)(COLUMN_NAME [alias] [AS] [alias] |
COUNT(* | COLUMN_NAME [alias] [AS] [alias])
```

- table_reference

```
tbl_name [AS] alias [index_hint_list]
| table_reference ([INNER] | [LEFT|RIGHT] [OUTER]) JOIN table_factor [JOIN ON conditional_expr | USING (column_list)]
```

子查询

子查询和外层查询同时指定分片键，且分片键的值保持一致时，由内核提供稳定支持。

例如：

```
SELECT * FROM (SELECT * FROM t_order WHERE order_id = 1) o WHERE o.order_id = 1;
```

用于分页的子查询，由内核提供稳定支持。

例如：


```
SELECT * FROM (SELECT row_.*, rownum rownum_ FROM (SELECT * FROM t_order) row_ WHERE rownum <= ?) WHERE rownum > ?;
```

分页查询

完全支持 MySQL、PostgreSQL、openGauss 子查询。

MySQL、PostgreSQL 和 openGauss 都支持 LIMIT 分页，无需子查询：

```
SELECT * FROM t_order o ORDER BY id LIMIT ? OFFSET ?
```

运算表达式中包含分片键

当分片键处于运算表达式中时，无法通过 SQL 字面提取用于分片的值，将导致全路由。例如，假设 create_time 为分片键：

```
SELECT * FROM t_order WHERE to_date(create_time, 'yyyy-mm-dd') = '2019-01-01';
```

实验性支持

实验性支持特指使用 Federation 执行引擎提供支持。该引擎处于快速开发中，用户虽基本可用，但仍需大量优化，是实验性产品。

子查询

子查询和外层查询未同时指定分片键，或分片键的值不一致时，由 Federation 执行引擎提供支持。

例如：

```
SELECT * FROM (SELECT * FROM t_order) o;
SELECT * FROM (SELECT * FROM t_order) o WHERE o.order_id = 1;
SELECT * FROM (SELECT * FROM t_order WHERE order_id = 1) o;
SELECT * FROM (SELECT * FROM t_order WHERE order_id = 1) o WHERE o.order_id = 2;
```

跨库关联查询

当关联查询中的多个表分布在不同的数据库实例上时，由 Federation 执行引擎提供支持。假设 't_order' 和 't_order_item' 是多数据节点的分片表，并且未配置绑定表规则，'t_user' 和 't_user_role' 是分布在不同的数据库实例上的单表，那么 Federation 执行引擎能够支持如下常用的关联查询：

```
SELECT * FROM t_order o INNER JOIN t_order_item i ON o.order_id = i.order_id WHERE o.order_id = 1;
SELECT * FROM t_order o INNER JOIN t_user u ON o.user_id = u.user_id WHERE o.user_id = 1;
SELECT * FROM t_order o LEFT JOIN t_user_role r ON o.user_id = r.user_id WHERE o.user_id = 1;
SELECT * FROM t_order_item i LEFT JOIN t_user u ON i.user_id = u.user_id WHERE i.user_id = 1;
SELECT * FROM t_order_item i RIGHT JOIN t_user_role r ON i.user_id = r.user_id WHERE i.user_id = 1;
SELECT * FROM t_user u RIGHT JOIN t_user_role r ON u.user_id = r.user_id WHERE u.user_id = 1;
```

不支持

不支持的 SQL：

- CASE WHEN 中包含子查询
- CASE WHEN 中使用逻辑表名（请使用表别名）
- INSERT INTO tbl_name (col1, col2, ...) SELECT * FROM tbl_name WHERE col3 = ?（SELECT 子句不支持 * 和内置分布式主键生成器）

- REPLACE INTO tbl_name (col1, col2, ...) SELECT * FROM tbl_name WHERE col3 = ? (SELECT 子句不支持 * 和内置分布式主键生成器)
- SELECT MAX(tbl_name.col1) FROM tbl_name (查询列是函数表达式时, 查询列前不能使用表名, 可以使用表别名)

CASE WHEN

以下 CASE WHEN 语句不支持:

- CASE WHEN 中包含子查询
- CASE WHEN 中使用逻辑表名 (请使用表别名)

注意事项

1. 分片表 Join 问题, 多表关联中涉及的小表可以配置为广播表, 可提升关联效率;
2. 分布式主键问题, 在 Sharding 场景下, 自增字段无法保证全局唯一性, 因此需搭建发号器来生成全局唯一的 ID, 如 Snowflake、UUID 或 NanoID 等;
3. 分布式事务问题: SphereEx-DBPlusEngine 提供了 LOCAL, XA, BASE 三种模式的分布式事务的, GTM 暂时没有实现。

7.1.2 原理介绍

SphereEx-DBPlusEngine 内核处理流程包括标准的 SQL Parser 和 SQL Binder 模块, 这两个模块用于 SQL 具体特征的识别, 并根据其结果, 将 SQL 执行流程分为 Simple Push Down Engine 和 SQL Federation Engine。

关于原理内容的详细介绍, 请参考[内核原理-数据分片](#)。

7.1.3 使用指南

1. 分片设计: 分片键

在开始分片之前, 首先需要确定分片键 (Sharding Key)。很多例子会使用自增 ID 或时间戳字段进行拆分, 这些不是绝对的; 更好的建议是结合业务, 进行选择。可考虑按如下策略进行选择:

- 对系统中执行的 SQL 进行统计分析, 选择出需要分片那个表中最频繁被使用到或最为重要的字段类分片。这其中可能包含一些来自 OLAP 类的查询, 可将此部分 SQL 排除在外。
- 如业务非常复杂, 存在多种维度划分可能, 可以考虑适当多维度字段拆表。必要时可考虑通过业务逻辑进行拆分 (而非简单字段划分)。
- 如涉及到多维度拆分, 需维护一定数据冗余。一般主维度可以由程序写入, 次维度则异步写入, 保证两者最终一致。
- 如涉及到关联查询的问题, 可配合分片策略来实施。如考虑 ER 表、广播表等配合策略。如非常复杂, 也可考虑在 OLAP 系统中提供支持。数据同步策略上, 可考虑推拉结合。
- 最终选择拆分字段, 应是稳定的, 不变更的, 避免跨片移动问题。

2. 分片设计: 分片策略

针对分片算法, 常见的算法有 LIST、RANGE、HASH 等。根据各拆分算法特点, 可进行选择。若范围均匀可采用 HASH, 冷热数据明显可采用 RANGE 等。同时可配合一些特性化设计, 如采用二级映射方式解决扩缩容问题、特征编码字段满足多特征拆分等。

	Hash	List	Range
优点	分布及访问均匀	人工可以识别	人工可以识别有序, 范围查询友好收缩/收缩相对友好
缺点	无序, 范围查询需跨片人工无法识别分布收缩/扩容不友好	容易分布不均匀	数据分布/数据访问不均匀

这其中最为常见的就是 HASH、RANGE, 说明如下。

■ HASH

虽然分库分表的方案众多，但是 Hash 分库分表是最大众最普遍的方案。随机分片其实并不是随机，也遵循一定规则。通常采用 HASH 取模的方式进行分片拆分，所以有时候也称为离散分片。随机分片的数据相对均匀，不容易出现热点和并发访问的瓶颈。但涉及后面数据迁移的话，不太方便。可使用一致性 HASH 算法在很大程度上避免此问题。此外，离散分片也容易面临跨分片查询的复杂问题。

■ RANGE

通过数据的范围进行分库分表，是最朴实的一种分库方案，它也可以和其他分库分表方案灵活结合使用。当需要使用分片字段进行范围查找时，RANGE 分片策略可快速定位数据进行高效查询。大多数情况下有效避免跨分片查询的问题。在后期扩容时，也比较方便，只需要添加节点即可，无需对其他分片的数据进行迁移。但这种分布方式容易存在数据热点问题。

3. 设计：分片数量规划

由于业务特点不同，对于单分片数量无明确限制，需结合测试结果来判定，满足效率和预留空间需求即可。

4. 设计：分布式 ID

在分库分表环境下，数据分布在不同分片上，不能再借助数据库自增长特性直接生成，否则会造成不同分片上的数据表主键重复，可使用全局发号器来生成全局唯一的 ID，如 Snowflake、UUID 或 NanoID。

5. 针对已上线业务，需确认是否存在不支持的语句，并适配改造

数据经过拆分后，相较于传统集中式必然面临一些 SQL 受限的问题，这里需要业务对 SQL 特点进行梳理，结合分片使用限制说明，进行适配改造。

6. 针对已上线业务的迁移与扩容

对于新项目而言，配置好分片规则后即可直接使用。对于已上线的业务，需要对数据进行迁移，即从集中式到分布式的迁移，可参考数据迁移插件。对于二次扩容的情况，可参考弹性伸缩插件。

7.1.4 行表达式

在繁琐的数据分片规则配置中，随着数据节点的增多，大量的重复配置使得配置本身不易被维护。通过行表达式可以有效地简化数据节点配置工作量。

对于常见的分片算法，使用 Java 代码实现并不有助于配置的统一管理。通过行表达式书写分片算法，可以有效地将规则配置一同存放，更加易于浏览与存储。

语法说明

行表达式的使用非常直观，只需要在配置中使用 `${ expression }` 或 `$->{ expression }` 标识行表达式即可。目前支持数据节点和分片算法这两个部分的配置。行表达式的内容使用的是 Groovy 的语法，Groovy 能够支持的所有操作，行表达式均能够支持。例如：

`${begin..end}` 表示范围区间

`${[unit1, unit2, unit_x]}` 表示枚举值

行表达式中如果出现连续多个 `${ expression }` 或 `$->{ expression }` 表达式，整个表达式最终的结果将会根据每个子表达式的结果进行笛卡尔组合。

例如，以下行表达式：

```
${['online', 'offline']}_table${1..3}
```

最终会解析为：

```
online_table1, online_table2, online_table3, offline_table1, offline_table2, offline_table3
```

配置说明

对于均匀分布的数据节点，如果数据结构如下：

```
db0
├── t_order0
└── t_order1
db1
├── t_order0
└── t_order1
```

用行表达式可以简化为：

```
db${0..1}.t_order${0..1}
```

或者

```
db$->{0..1}.t_order$->{0..1}
```

对于自定义的数据节点，如果数据结构如下：

```
db0
├── t_order0
└── t_order1
db1
├── t_order2
├── t_order3
└── t_order4
```

用行表达式可以简化为：

```
db0.t_order${0..1},db1.t_order${2..4}
```

或者

```
db0.t_order$->{0..1},db1.t_order$->{2..4}
```

对于有前缀的数据节点，也可以通过行表达式灵活配置，如果数据结构如下：

```
db0
├── t_order_00
├── t_order_01
├── t_order_02
├── t_order_03
├── t_order_04
├── t_order_05
├── t_order_06
├── t_order_07
├── t_order_08
├── t_order_09
├── t_order_10
├── t_order_11
├── t_order_12
├── t_order_13
├── t_order_14
├── t_order_15
├── t_order_16
├── t_order_17
├── t_order_18
├── t_order_19
└── t_order_20
db1
├── t_order_00
└── t_order_01
```

```
|—— t_order_02  
|—— t_order_03  
|—— t_order_04  
|—— t_order_05  
|—— t_order_06  
|—— t_order_07  
|—— t_order_08  
|—— t_order_09  
|—— t_order_10  
|—— t_order_11  
|—— t_order_12  
|—— t_order_13  
|—— t_order_14  
|—— t_order_15  
|—— t_order_16  
|—— t_order_17  
|—— t_order_18  
|—— t_order_19  
|—— t_order_20
```

可以使用分开配置的方式，先配置包含前缀的数据节点，再配置不含前缀的数据节点，再利用行表达式笛卡尔积的特性，自动组合即可。上面的示例，用行表达式可以简化为：

```
db${0..1}.t_order_0${0..9}, db${0..1}.t_order_${10..20}
```

或者

```
db$->{0..1}.t_order_0$->{0..9}, db$->{0..1}.t_order_${10..20}
```

对于只有一个分片键的使用 = 和 IN 进行分片的 SQL，可以使用行表达式代替编码方式配置。

行表达式内部的表达式本质上是一段 Groovy 代码，可以根据分片键进行计算的方式，返回相应的真实数据源或真实表名称。

例如：分为 10 个库，尾数为 0 的路由到后缀为 0 的数据源，尾数为 1 的路由到后缀为 1 的数据源，以此类推。用于表示分片算法的行表达式为：

```
ds${id % 10}
```

或者

```
ds$->{id % 10}
```

7.1.5 分片策略

包含分片键和分片算法，由于分片算法的独立性，将其独立抽离。真正可用于分片操作的是分片键 + 分片算法，也就是分片策略。

7.1.6 强制分片路由

对于分片字段并非由 SQL 而是其他外置条件决定的场景，可使用 SQL Hint 注入分片值。例：按照员工登录主键分库，而数据库中并无此字段。SQL Hint 支持通过 Java API 和 SQL 注释（待实现）两种方式使用。- 实现动机通过解析 SQL 语句提取分片键列与值并进行分片是 SphereEx-DBPlusEngine 对 SQL 零侵入的实现方式。若 SQL 语句中没有分片条件，则无法进行分片，需要全路由。在一些应用场景中，分片条件并不存在于 SQL，而存在于外部业务逻辑。因此需要提供一种通过外部指定分片结果的方式，在 SphereEx-DBPlusEngine 中叫做 Hint。- 实现机制 DBPlusEngine 使用 ThreadLocal 管理分片键值。可以通过编程的方式向 HintManager 中添加分片条件，该分片条件仅在当前线程内生效。除了通过编程的方式使用强制分片路由，SphereEx-DBPlusEngine 还可以通过 SQL 中的特殊注释的方式引用 Hint，使开发者可以采用更加透明的方式使用该功能。指定了强制分片路由的 SQL 将会无视原有的分片逻辑，直接路由至指定的真实数据节点。

7.1.7 操作指南

DistSQL

1. 准备环境，完成 SphereEx-DBPlusEngine 及数据库部署；
2. 创建逻辑库，注册存储（数据库）资源；
3. 创建分片规则，配置数据源、分片算法及分片数量；
4. 创建与分片规则同名的分片表；
5. 插入测试数据，确认执行路径，完成配置。

Console

1. 准备环境，完成 SphereEx-Console、SphereEx-DBPlusEngine 及数据库部署；
2. 登陆 SphereEx-Console，在“集群” - “对象”页中，点击“新建表”按钮；
3. 在新建表的页面中，填写建表语句，然后点击下方“添加分片插件”按钮；
4. 在分片插件页面中，对分片信息进行配置；
5. 确认表分布情况，完成配置。

7.1.8 配置示例

以下配置示例不涉及存量数据，对于有存量数据拆分的情况，需参考[数据迁移插件](#)操作。

使用 DistSQL 配置分片

环境说明

\	实例	IP 地址	服务端口	主机名	备注
1	DBPlusEngine 1.2.0	192.168.xx.102	3307	dbplusengine	
2	MySQL 8.0.28	192.168.xx.105	3306	mysql_0	
3	MySQL 8.0.28	192.168.xx.106	3306	mysql_1	

拓扑图

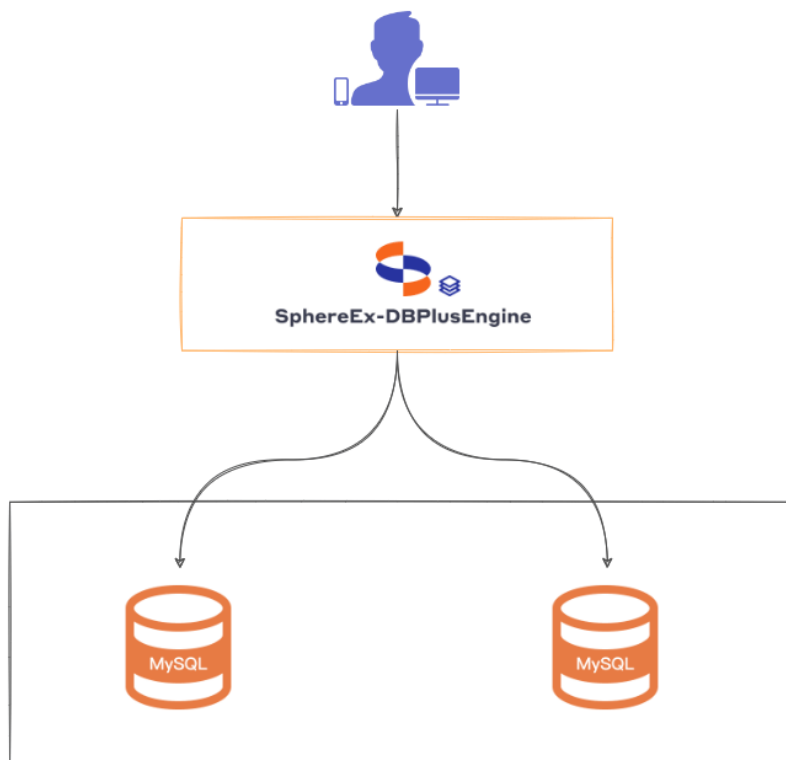


Fig. 1: 拓扑图

配置过程

请提前在 MySQL 两个数据源中分别创建 mysql_0 和 mysql_1 数据库。

1. 在 SphereEx-DBPlusEngine 创建名为 testdb 的逻辑库。

```
mysql> CREATE DATABASE testdb;  
Query OK, 0 rows affected (0.15 sec)
```

```
mysql> SHOW DATABASES;
```

```
+-----+  
| schema_name |  
+-----+  
| mysql       |  
| information_schema |  
| performance_schema |  
| sys        |  
| testdb     |  
+-----+  
5 rows in set (0.00 sec)
```

2. 将准备好的数据源注册到 SphereEx-DBPlusEngine 中。


```
mysql> USE testdb;
Database changed

mysql> REGISTER STORAGE UNIT ds_0 (
  URL="jdbc:mysql://192.168.xx.105:3306/mysql_0?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
), ds_1 (
  URL="jdbc:mysql://192.168.xx.106:3306/mysql_1?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
);
Query OK, 0 rows affected (0.60 sec)

mysql> SHOW STORAGE UNITS\G
***** 1. row *****
      name: ds_1
      type: MySQL
      host: 192.168.xx.106
      port: 3306
      db: mysql_1
connection_timeout_milliseconds: 30000
idle_timeout_milliseconds: 60000
max_lifetime_milliseconds: 2100000
max_pool_size: 50
min_pool_size: 1
read_only: false
other_attributes: {"dataSourceProperties":{"cacheServerConfiguration":"true","elideSetAutoCommits":"true",
"useServerPrepStmts":"true","cachePrepStmts":"true","rewriteBatchedStatements":"true","cacheResultSetMetadata":"false",
"useLocalSessionState":"true","maintainTimeStats":"false","prepStmtCacheSize":"200000","tinyInt1isBit":"false",
"prepStmtCacheSqlLimit":"2048","netTimeoutForStreamingResults":"0","zeroDateTimeBehavior":"round"},
"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,"leakDetectionThreshold":0,"poolName":
"HikariPool-83","registerMbeans":false,"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false}
***** 2. row *****
      name: ds_0
      type: MySQL
      host: 192.168.xx.105
      port: 3306
      db: mysql_0
connection_timeout_milliseconds: 30000
idle_timeout_milliseconds: 60000
max_lifetime_milliseconds: 2100000
max_pool_size: 50
min_pool_size: 1
read_only: false
other_attributes: {"dataSourceProperties":{"cacheServerConfiguration":"true","elideSetAutoCommits":"true",
"useServerPrepStmts":"true","cachePrepStmts":"true","rewriteBatchedStatements":"true","cacheResultSetMetadata":"false",
"useLocalSessionState":"true","maintainTimeStats":"false","prepStmtCacheSize":"200000","tinyInt1isBit":"false",
"prepStmtCacheSqlLimit":"2048","netTimeoutForStreamingResults":"0","zeroDateTimeBehavior":"round"},
"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,"leakDetectionThreshold":0,"poolName":
"HikariPool-84","registerMbeans":false,"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false}
2 rows in set (0.00 sec)
```

3. 创建分片规则，并确认创建结果，此处创建分片规则的名称需要与业务表名保持一致。

```
mysql> CREATE SHARDING TABLE RULE t_user(
  STORAGE_UNITS(ds_0,ds_1),
  SHARDING_COLUMN=user_id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="4"))
);
Query OK, 0 rows affected (0.40 sec)

mysql> SHOW SHARDING TABLE RULE t_user\G
***** 1. row *****
```



```

table: t_user
actual_data_nodes:
actual_data_sources: ds_0,ds_1
database_strategy_type:
database_sharding_column:
database_sharding_algorithm_type:
database_sharding_algorithm_props:
  table_strategy_type: STANDARD
  table_sharding_column: user_id
  table_sharding_algorithm_type: hash_mod
table_sharding_algorithm_props: sharding-count=4
key_generate_column:
key_generator_type:
key_generator_props:
1 row in set (0.00 sec)

```

通过以上配置，t_user 表会被分为 4 个分片，存储在 2 个数据源中。其中 user_id 是分片字段，所采用的分片算法是 hash_mod。

4. 创建 t_user 表，并插入若干条记录。

```

mysql> CREATE TABLE `t_user` (
  `user_id` int NOT NULL,
  `order_id` int NOT NULL,
  `status` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`user_id`)
);
Query OK, 0 rows affected (0.45 sec)

mysql> INSERT INTO t_user VALUES
(1,1,'active'),
(2,2,'active'),
(3,3,'active'),
(4,4,'active');
Query OK, 4 rows affected (0.20 sec)

mysql> SELECT * FROM t_user ORDER BY user_id;
+-----+-----+
| user_id | order_id | status |
+-----+-----+
| 1 | 1 | active |
| 2 | 2 | active |
| 3 | 3 | active |
| 4 | 4 | active |
+-----+-----+
4 rows in set (0.06 sec)

```

5. 使用 PREVIEW 命令对 SQL 的执行路径进行查看。

```

mysql> PREVIEW SELECT * FROM t_user;
+-----+-----+
| data_source_name | actual_sql |
+-----+-----+
| ds_0 | SELECT * FROM t_user_0 UNION ALL SELECT * FROM t_user_2 |
| ds_1 | SELECT * FROM t_user_1 UNION ALL SELECT * FROM t_user_3 |
+-----+-----+
2 rows in set (0.05 sec)

```

6. 最后，在物理库中确认分片的分布情况。

```

--ds_0
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+

```

```

+-----+
| information_schema |
| mysql              |
| mysql_0            |
| performance_schema |
| sys                |
+-----+
5 rows in set (0.00 sec)

mysql> USE mysql_0;

mysql> SHOW TABLES;
+-----+
| Tables_in_mysql_0 |
+-----+
| t_user_0          |
| t_user_2          |
+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM t_user_0;
+-----+-----+-----+
| user_id | order_id | status |
+-----+-----+-----+
| 4       | 4        | active |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM t_user_2;
+-----+-----+-----+
| user_id | order_id | status |
+-----+-----+-----+
| 2       | 2        | active |
+-----+-----+-----+
1 row in set (0.00 sec)

--ds_1
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql          |
| mysql_0        |
| performance_schema |
| sys            |
+-----+
5 rows in set (0.00 sec)

mysql> USE mysql_1;

mysql> SHOW TABLES;
+-----+
| Tables_in_mysql_1 |
+-----+
| t_user_1          |
| t_user_3          |
+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM t_user_1;
+-----+-----+-----+
| user_id | order_id | status |
+-----+-----+-----+

```

```
| 1 | 1 | active |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM t_user_3;
+-----+-----+
| user_id | order_id | status |
+-----+-----+
| 3 | 3 | active |
+-----+-----+
1 row in set (0.00 sec)
```

如输出结果所示，t_user 的记录存储在 2 个数据库的 4 张分片表中。

使用 Console 配置分片

环境说明

	实例	IP 地址	服务端口	主机名	备注
1	DBPlusEngine	10.9.122.80	3307	dbplusengine	
2	Console	10.9.122.81	8089	console	
3	MySQL	10.9.122.82	13306	test_2	
4	MySQL	10.9.122.82	13306	test_3	

拓扑图

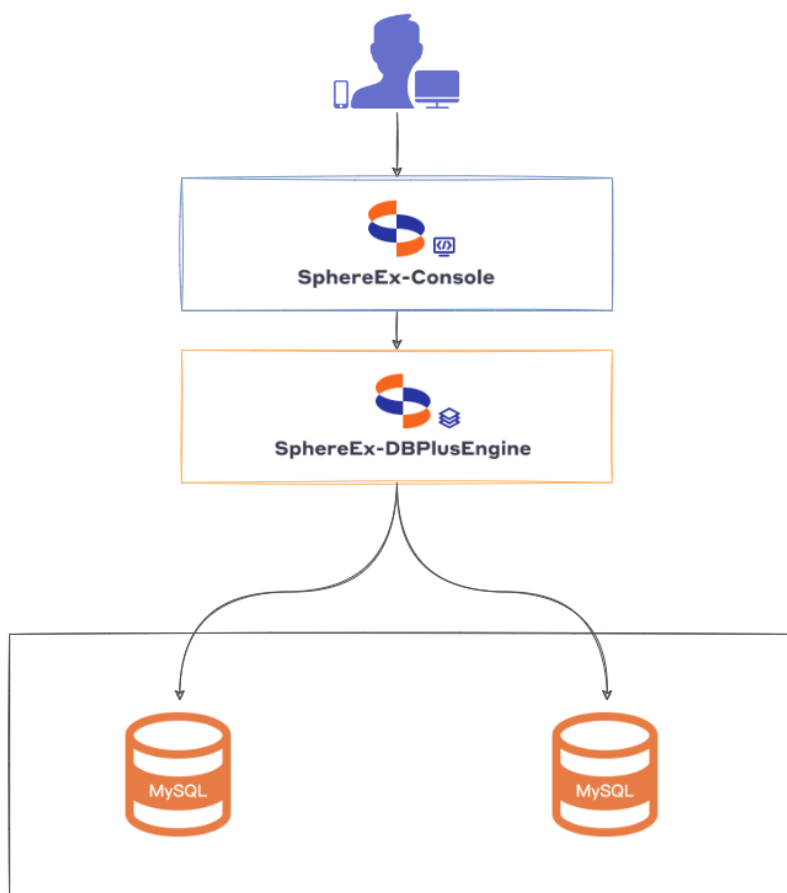


Fig. 2: 拓扑图

配置过程

1. 登陆 SphereEx-Console，在“集群” - “对象”页中，点击“新建表”按钮。
2. 在新建表的页面中，在表单框中填写建表语句，然后点击下方“添加分片插件”按钮。
3. 在分片插件页面中，对分片信息进行配置。

在此页面中，可选择指定的表类型、存储节点、分片键、分片算法和分片数量等信息，然后点击“确定”按钮。

4. 确认表分布情况。

此时页面会弹出“新建成功”的提示信息。我们可通过点击对象列表中的“查看表分布”属性链接，确认分片表的分片信息。通过 SphereEx-Console 创建分片表的操作结束。

使用自定义分片

通过配置分片策略类型和算法类名，实现自定义扩展。CLASS_BASED 允许向算法类内传入额外的自定义属性，传入的属性可以通过属性名为 props 的 java.util.Properties 类实例取出。参考 Git 的 org.apache.shardingsphere.example.extension.sharding.algorithm.classbased.fixture.ClassBasedStandardShardingAlgorithmFixture 。

类型：CLASS_BASED

可配置属性：

	属性名称	数据类型	说明
1	strategy	String	分片策略类型，支持 STANDARD、COMPLEX 或 HINT（不区分大小写）
2	algorithmClassName	String	分片算法全限定名

配置示例

使用数据分片时，在 shardingAlgorithms 属性下配置对应的数据分片算法即可。

```
rules:
- !SHARDING
  tables:
    t_order:
      actualDataNodes: ds_${0..1}.t_order_${0..1}
      tableStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: t-order-inline
      keyGenerateStrategy:
        column: order_id
        keyGeneratorName: snowflake
    t_order_item:
      actualDataNodes: ds_${0..1}.t_order_item_${0..1}
      tableStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: t_order-item-inline
      keyGenerateStrategy:
        column: order_item_id
        keyGeneratorName: snowflake
    t_account:
      actualDataNodes: ds_${0..1}.t_account_${0..1}
      tableStrategy:
        standard:
          shardingAlgorithmName: t-account-inline
      keyGenerateStrategy:
        column: account_id
        keyGeneratorName: snowflake
  defaultShardingColumn: account_id
  bindingTables:
  - t_order,t_order_item
  broadcastTables:
  - t_address
  defaultDatabaseStrategy:
    standard:
      shardingColumn: user_id
      shardingAlgorithmName: database-inline
  defaultTableStrategy:
    none:

  shardingAlgorithms:
    database-inline:
      type: INLINE
```

```
props:
  algorithm-expression: ds_${user_id % 2}
t-order-inline:
  type: INLINE
  props:
    algorithm-expression: t_order_${order_id % 2}
t_order-item-inline:
  type: INLINE
  props:
    algorithm-expression: t_order_item_${order_id % 2}
t-account-inline:
  type: INLINE
  props:
    algorithm-expression: t_account_${account_id % 2}
keyGenerators:
  snowflake:
    type: SNOWFLAKE
```

7.1.9 FAQ

1. 使用分片插件后，DDL 可以通过 SphereEx-DBPlusEngine 来完成吗？

可以。通过 SphereEx-DBPlusEngine 可以像使用一个数据库一样来维护整个分布式架构，无需在数据库中逐一执行。

2. 如果把目前在用的 MySQL 数据库迁移到分布式集群当中，是否有对应的方案？

有的。可以关注 SphereEx-DBPlusEngine 数据迁移和弹性扩容插件的能力，提供一站式的从集中式到分布式的数据迁移，同时还可以根据资源的使用情况合理的给存储节点做扩所容。

7.2 数据库发现

高可用是现代系统的最基本诉求，作为系统基石的数据库，对于高可用的要求也是必不可少的。本章节所讨论的高可用，是指 SphereEx-DBPlusEngine 对存储节点可用状态的发现的方案。

在存算分离的分布式数据库体系中，存储节点和计算节点的高可用方案是不同的。对于有状态的存储节点来说，需要其自身具备数据一致性同步、探活以及主节点选举等能力；对于无状态的计算节点来说，需要感知存储节点的变化时，还需要独立架设负载均衡器，并具备服务发现和请求分发的能力。

概述

SphereEx-DBPlusEngine 自身提供计算节点，并通过数据库作为存储节点构建集群。本身并不提供数据库的高可用能力，但可以借助数据库高可用能力，并通过自身数据库发现及动态感知的能力，帮助用户整合主从切换、故障发现、流量切换治理等一体化的数据库高可用方案。因此，它采用的高可用方案是利用数据库自身的高可用方案做存储节点高可用，并自动识别其变化。特别是在分布式场景下搭配主从流量控制的特性，可以提供更为完善的高可用读写分离方案。

SphereEx-DBPlusEngine 需要自动感知多样化的存储节点高可用方案的同时，也能够动态集成对接读写分离方案。

适用场景

■ 高可用场景

SphereEx-DBPlusEngine 不提供数据库高可用的能力，它通过感知数据库主从关系的切换来保证系统的可用性。确切来说，SphereEx-DBPlusEngine 提供数据库发现的能力，自动感知数据库主从关系，并修正计算节点对数据库的连接。

■ 动态读写分离场景

高可用和读写分离一起使用时，读写分离无需配置具体的主库和从库。同时，支持在半同步复制，异步复制时探测从库的延迟时间，并动态路由延迟较低的从库提供读流量。数据库发现能力会动态的识别读写分离的主从关系，并正确地疏导读写流量。高可用还提供从库全部宕机，读流量自动路由至主库，从而保证业务系统的可用性。

7.2.1 数据库支持

目前 SphereEx-DBPlusEngine 支持多种数据库架构高可用发现，具体支持类型参考[数据库发现算法](#)。

7.2.2 使用前提

- 已安装部署 SphereEx-DBPlusEngine 和 Zookeeper，服务运行正常；
- 已安装部署数据库单主集群，服务运行正常。

7.2.3 不支持项

- 不支持多主集群模式；
- 不支持级联集群模式；
- 不支持主主集群模式。

7.2.4 注意事项

- 重复的 ruleName 将无法被创建；
- 正在被使用的 discoveryType 和 discoveryHeartbeat 无法被删除；
- 带有 - 的命名在改动时需要使用 ""；
- 移除 discoveryRule 时不会移除被该 discoveryRule 使用的 discoveryType 和 discoveryHeartbeat。

7.2.5 原理介绍

数据库发现能力的实现包含了前置检查，动态发现主库，动态发现从库，以及心跳检测机制这四个方面。

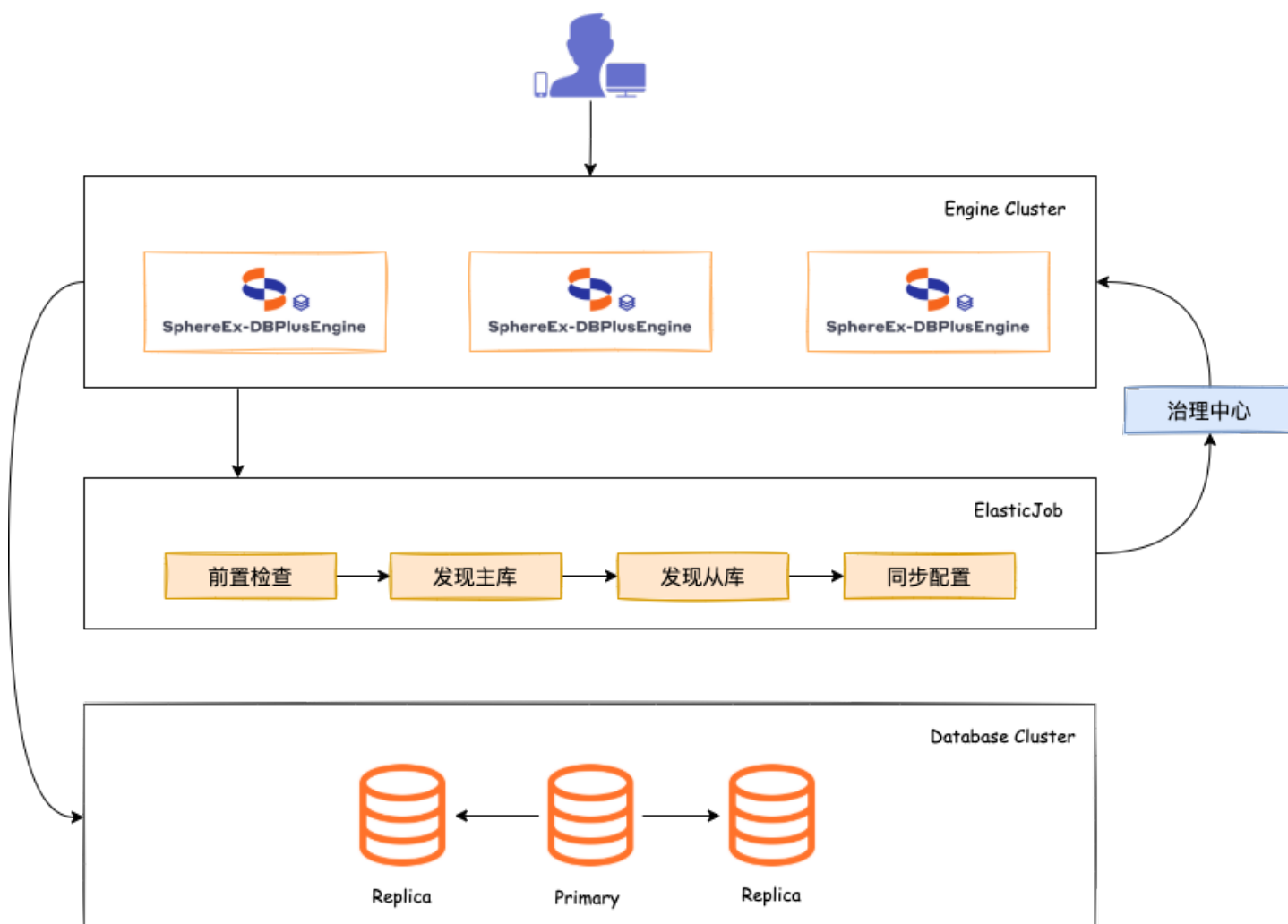


Fig. 3: 数据库发现原理

7.2.6 前置检查

在数据库发现配置过程中，SphereEx-DBPlusEngine 会对环境进行检查，当所有检查项均通过校验后，才可正常完成配置。下面以 MGR 集群为例来介绍所检查的内容。

- 检查是否安装了 MGR 插件，以下命令应输出插件名为 group_replication 的插件信息。

```
SELECT * FROM information_schema.PLUGINS WHERE PLUGIN_NAME='group_replication';
```

- 查看 MGR 组成员数量，MGR 要求最少有 3 个节点，以下命令输出的数字应大于等于 3。

```
SELECT count(*) FROM performance_schema.replication_group_members;
```

- 校验 MGR 集群的 group_replication_group_name 信息是否与配置中的组名一致。

-- group_name 是 MGR 组的标识，一组 MGR 集群对应同一个 group name

```
SELECT * FROM performance_schema.global_variables WHERE VARIABLE_NAME='group_replication_group_name';
```

- 查看当前 MGR 是否设置为单主模式，需满足单主模式，group_replication_single_primary_mode 参数值应为 ON。


```
SELECT * FROM performance_schema.global_variables WHERE VARIABLE_NAME='group_replication_single_primary_mode';
```

- 查询 MGR 组集群中所有的节点地址，端口及成员状态信息，用于校验我们配置的数据源是否正确。

```
SELECT MEMBER_HOST, MEMBER_PORT, MEMBER_STATE FROM performance_schema.replication_group_members;
```

7.2.7 动态发现主库

SphereEx-DBPlusEngine 会通过如下 SQL 命令获取主库 URL：

```
SELECT MEMBER_HOST, MEMBER_PORT FROM performance_schema.replication_group_members WHERE MEMBER_ID = "  
SELECT VARIABLE_VALUE FROM performance_schema.global_status WHERE VARIABLE_NAME = 'group_replication_primary_  
member'
```

然后将输出的结果与我们所配置的数据源的 URL 逐一去对比，命中的数据源则为主库，将主库更新至注册中心，通过注册中心分发至集群内其它计算节点中。

7.2.8 动态发现从库

SphereEx-DBPlusEngine 会将从库状态分为启动与禁用，并且从库的状态实时地同步至 SphereEx-DBPlusEngine 中，以保证读流量可以被正确的路由。

- 获取 MGR 组中所有的节点

```
SELECT MEMBER_HOST, MEMBER_PORT, MEMBER_STATE FROM performance_schema.replication_group_members;
```

- 禁用从库

从库的禁用，依据的是我们配置的数据源以及 MGR 组中所有的节点。SphereEx-DBPlusEngine 会逐一检查我们配置的数据源是否可以正常的获取 Connection，并校验数据源 URL 是否包含 MGR 组中的节点。无法正常获取 Connection 或校验失败，SphereEx-DBPlusEngine 会事件驱动禁用数据源，以及同步注册中心。

- 启用从库

恢复宕机从库，重新加入 MGR 组后，检查我们配置中是否使用了被恢复的数据源。如果使用，则会告诉 SphereEx-DBPlusEngine 需要该数据源恢复成启用状态。

7.2.9 心跳检测机制

为了保证主从状态同步的实时性，高可用模块引入心跳检测机制，在高可用模块初始化时将上述的流程以 Job 的方式交由调度框架执行，实现了功能开发和作业调度的分离。我们可以根据业务特点，灵活配置心跳周期。

操作指南

1. 准备环境，完成 SphereEx-DBPlusEngine 集群及数据库集群部署；
2. 创建逻辑库，注册存储（数据库）资源；
3. 创建数据库发现规则，包括存储节点信息，数据库发现类型及心跳机制；
4. 创建动态读写分离规则；
5. 确认数据库发现规则配置信息，完成配置。

配置示例

以 3 节点 MGR 场景为例，基于动态读写分离的场景，数据库发现规则配置如下所示。

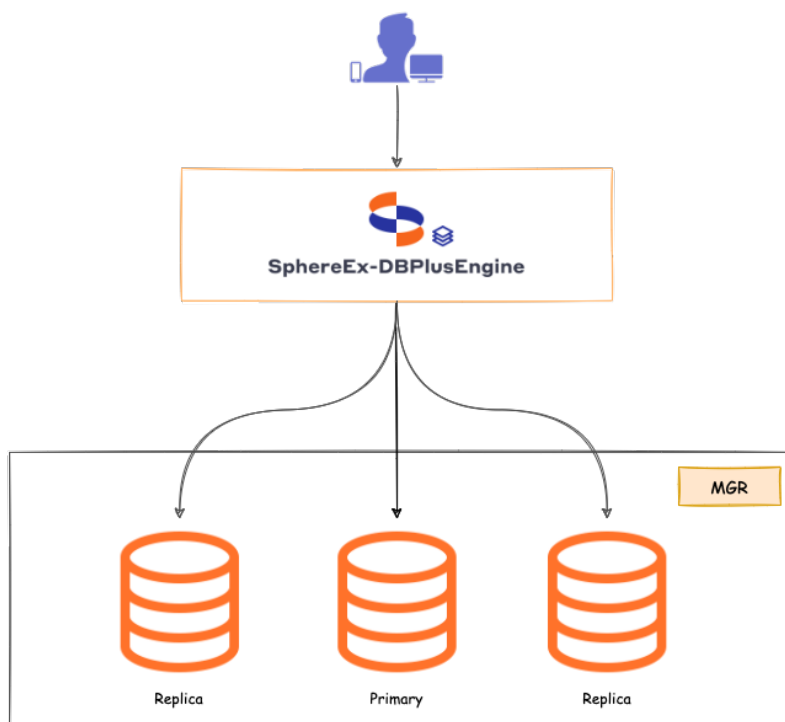


Fig. 4: 拓扑图

1. 在 MGR 主节点中创建名为 ha 的数据库

```
mysql -utest -h192.168.56.104 -P3306 -p  
SELECT * FROM performance_schema.replication_group_members;
```

-- 连接主库

```
DROP SCHEMA IF EXISTS ha;  
CREATE SCHEMA IF NOT EXISTS ha;
```

2. 在 Proxy 中创建名为 testdb 的逻辑库

```
mysql -uroot -p -P3307 -h127.0.0.1
```

```
mysql> CREATE DATABASE testdb;  
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> USE testdb;  
Database changed
```

3. 将 MGR 各个节点注册到 Proxy 中

```
mysql> REGISTER STORAGE UNIT ds_0 (  
  URL="jdbc:mysql://192.168.xx.103:3306/ha?serverTimezone=UTC&useSSL=false",  
  USER="test",  
  PASSWORD="Test@123"
```

```
), ds_1 (
  URL="jdbc:mysql://192.168.xx.104:3306/ha?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
), ds_2 (
  URL="jdbc:mysql://192.168.xx.105:3306/ha?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
);
```

查看当前已注册的数据源。

```
mysql> SHOW STORAGE UNITS;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| name | type | host | port | db | connection_timeout_milliseconds | idle_timeout_milliseconds | max_lifetime_milliseconds | max_pool_size | min_pool_size | read_only | other_attributes |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ds_2 | MySQL | 192.168.xx.105 | 3306 | ha | 30000 | 60000 | 2100000 | 50 | 1 | false | {
"dataSourceProperties":{"cacheServerConfiguration":"true","elideSetAutoCommits":"true","useServerPrepStmts":"true",
"cachePrepStmts":"true","rewriteBatchedStatements":"true","cacheResultSetMetadata":"false","useLocalSessionState":"true",
"maintainTimeStats":"false","prepStmtCacheSize":"8192","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"netTimeoutForStreamingResults":"0","zeroDateTimeBehavior":"round"},"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,"keepaliveTime":0,"leakDetectionThreshold":0,"registerMbeans":false,"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false} |
| ds_1 | MySQL | 192.168.xx.104 | 3306 | ha | 30000 | 60000 | 2100000 | 50 | 1 | false | {
"dataSourceProperties":{"cacheServerConfiguration":"true","elideSetAutoCommits":"true","useServerPrepStmts":"true",
"cachePrepStmts":"true","rewriteBatchedStatements":"true","cacheResultSetMetadata":"false","useLocalSessionState":"true",
"maintainTimeStats":"false","prepStmtCacheSize":"8192","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"netTimeoutForStreamingResults":"0","zeroDateTimeBehavior":"round"},"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,"keepaliveTime":0,"leakDetectionThreshold":0,"registerMbeans":false,"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false} |
| ds_0 | MySQL | 192.168.xx.103 | 3306 | ha | 30000 | 60000 | 2100000 | 50 | 1 | false | {
"dataSourceProperties":{"cacheServerConfiguration":"true","elideSetAutoCommits":"true","useServerPrepStmts":"true",
"cachePrepStmts":"true","rewriteBatchedStatements":"true","cacheResultSetMetadata":"false","useLocalSessionState":"true",
"maintainTimeStats":"false","prepStmtCacheSize":"8192","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"netTimeoutForStreamingResults":"0","zeroDateTimeBehavior":"round"},"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,"keepaliveTime":0,"leakDetectionThreshold":0,"registerMbeans":false,"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false} |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

4. 创建数据库发现规则

在创建数据库发现规则时，我们需要配置数据源，数据库发现类型，及心跳。对于 MGR 的类型，还需要填入 group-name 信息，具体值可通过 my.cnf 来确认。结合业务需要，设置数据库发现的心跳周期，此处配置方式同 crontab 格式相同，最小单位为秒。以下示例中为每 5 秒探测一次数据库节点情况。

```
mysql> CREATE DB_DISCOVERY RULE primary_replica_ds (
  STORAGE_UNITS(ds_0, ds_1, ds_2),
  TYPE(NAME='MySQL.MGR',PROPERTIES('group-name'='b309fcc3-93e8-11ec-b5bd-080027c850b1')),
```

```
HEARTBEAT(PROPERTIES('keep-alive-cron'='0/5 * * * * ?'))
);
Query OK, 0 rows affected (0.28 sec)
```

数据库发现规则创建完成后，通过 SHOW 命令进行确认。

```
mysql> SHOW DB_DISCOVERY RULES;
+-----+-----+-----+-----+
| group_name | data_source_names | primary_data_source_name | discovery_type |
| discovery_heartbeat |
+-----+-----+-----+-----+
| primary_replica_ds | ds_0,ds_1,ds_2 | ds_0 | {name=primary_replica_ds_mysql_mgr, type=MySQL.MGR, props={group-name=b309fcc3-93e8-11ec-b5bd-080027c850b1}} | {name=primary_replica_ds_heartbeat} |
+-----+-----+-----+-----+
1 row in set (0.06 sec)

mysql> SHOW DB_DISCOVERY HEARTBEATS;
+-----+-----+
| name | props |
+-----+-----+
| primary_replica_ds_heartbeat | {keep-alive-cron=0/5 * * * * ?} |
+-----+-----+
1 row in set (0.01 sec)
```

如上输出展示了具体数据库发现规则的配置，规则已配置完成。

5. 创建动态读写分离规则

对于有动态读写分离需求的场景，可在该步骤进行读写分离规则的创建。在动态读写分离配置中写入自定义的数据库发现规则的名称即可，根据上一步骤内容所示，此处应填入 primary_replica_ds。

```
mysql> CREATE READWRITE_SPLITTING RULE dynamic_readwrite_ds (
  AUTO_AWARE_RESOURCE=primary_replica_ds
);
Query OK, 0 rows affected (0.08 sec)
```

接下来，确认读写分离规则。

```
mysql> SHOW READWRITE_SPLITTING RULES;
+-----+-----+-----+-----+-----+-----+
| name | auto_aware_data_source_name | write_data_source_query_enabled | write_storage_unit_name | read_storage_unit_names | load_balancer_type | load_balancer_props |
+-----+-----+-----+-----+-----+-----+
| dynamic_readwrite_ds | primary_replica_ds | NULL | ds_0 | ds_1,ds_2 | | |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.04 sec)
```

通过输出内容可确认，当前 ds_0 为写节点，ds_1 和 ds_2 为读节点。数据库发现规则和动态读写分离规则配置完毕。

7.3 读写分离

面对日益增加的系统访问量，数据库的吞吐能力面临着极大的挑战。对于瞬时有大量并发读操作和较少写操作类型的应用系统来说，可将数据库拆分为主库和从库，主库负责处理事务性的增删改操作，从库负责处理查询操作，能够有效的避免由数据更新导致的行锁，使得整个系统的查询性能得到极大的改善。

通过一主多从的配置方式，可以将查询请求均匀的或者按照权重分散到多个数据副本，能够进一步的提升系统的处理能力，同时也提升了从库资源的利用率。

7.3.1 概述

读写分离是一种数据库性能优化技术，在读请求明显多于写请求的业务场景中，使用读写分离技术可大幅度提升系统吞吐能力，即主库负责处理事务性的增删改请求，从库只负责处理查询请求。

读写分离是 SphereEx-DBPlusEngine 插件之一，可以对进入 SphereEx-DBPlusEngine 的业务流量做数据库层的读写分离处理，可支持配置动态读写分离和静态读写分离。

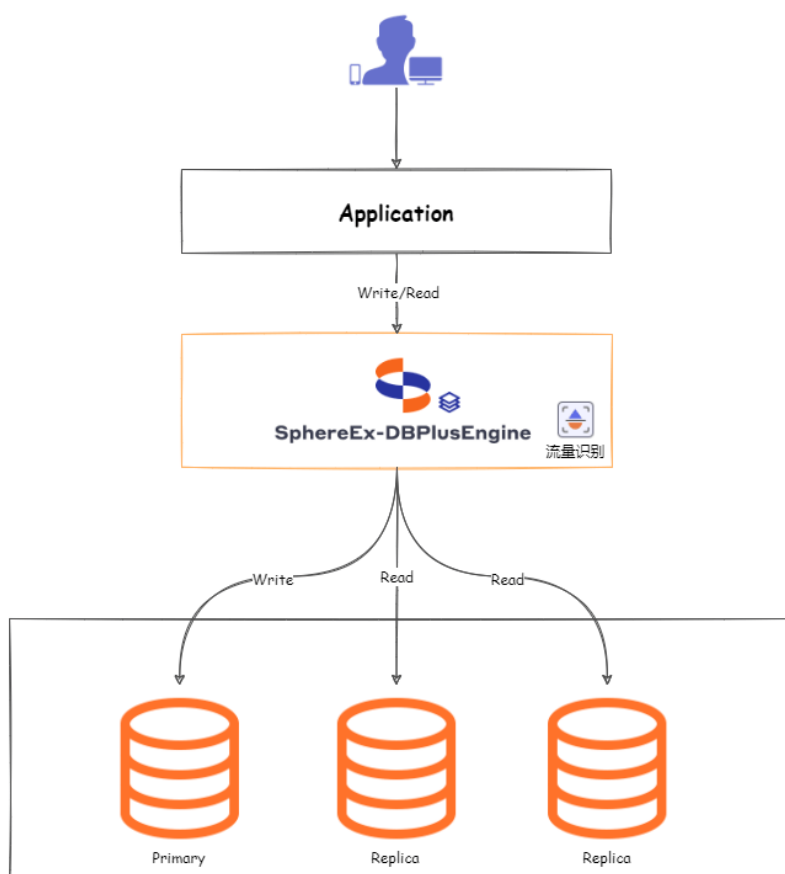


Fig. 5: 读写分离架构图

同时，SphereEx-DBPlusEngine 内置了多种读库负载均衡算法，包括轮询算法、随机访问算法和权重访问算法，能够满足用户绝大多数业务场景的需要，所支持的读库负载均衡算法参考[读写分离算法](#)。

SphereEx-DBPlusEngine 读写分离所支持的数据库架构如下表。

\	MySQL	PostgreSQL	openGauss	GuassDB	AuroraDB
静态规则	支持	支持	支持	支持	支持
动态规则	支持 MGR, MHA	不支持	支持	支持	支持

SphereEx-DBPlusEngine 读写分离动态和静态读写分离规则所支持的功能如下表。

\	动态读写分离规则	静态读写分离规则
数据库拓扑探测	支持	不支持
主从延迟判断	支持 (MHA)	不支持
从库负载均衡	支持	支持

基本概念

■ 主库

在读写分离场景中负责处理 INSERT、UPDATE 以及 DELETE 请求的数据库。

■ 从库

在读写分离场景中负责处理 SELECT 请求的数据库。

■ 从库负载均衡

指在具有多个从库的集群环境中，可通过指定负载均衡策略对 SELECT 请求进行路由。

适用场景

- 读请求明显多于写请求的场景；
- 系统并发量大，连接数不足的场景；
- 对于数据查询实时性不敏感的场景。

使用前提

- 服务器中已安装 SphereEx-DBPlusEngine 和数据库集群，且服务运行正常；
- 如需使用动态读写分离规则，需要与数据库的发现规则搭配使用。

使用规范

支持项

- 提供一主多从的读写分离配置，可独立使用，也可配合数据分片使用；
- 事务中的数据读写均用主库；
- 基于 Hint 的强制主库路由；
- 从库全部宕机主库提供读流量。

不支持项

- 主库和从库的数据同步；
- 主库和从库的数据同步延迟导致的数据不一致；
- 主库多写；
- 主从库间的事务一致性。主从模型中，事务中的数据读写均用主库。

注意事项

由于主库和从库间的数据通常是以异步模式进行传输的，主库和从库数据会存在短暂不一致的情况，具体延时取决于部署环境。

在架构规划阶段，需考虑业务查询请求对数据时延的接受范围，通过在 SphereEx-DBPlusEngine 中配置主从延迟阈值（仅基于 MHA 的动态规则支持），来满足业务对延迟的要求。

7.3.2 原理介绍

SphereEx-DBPlusEngine 的读写分离主要依赖内核的相关功能，包括解析引擎和路由引擎。

SphereEx-DBPlusEngine 对 SQL 语义进行分析，读写分离模块将读操作和写操作分别路由至主库与从库，即解析引擎将用户的 SQL 转化为 SphereEx-DBPlusEngine 可以识别的 Statement 信息，路由引擎根据 SQL 的读写类型以及事务的状态来做 SQL 的路由。

在多从库的环境中，SphereEx-DBPlusEngine 支持配置多种负载均衡算法，包括轮询算法、随机访问算法和权重访问算法等，可参考概述中对负载均衡算法的描述。

SphereEx-DBPlusEngine 针对动态读写分离规则，提供了主从节点延迟校验机制，底层调用了 MySQL SHOW SLAVE STATUS 命令。当主从延迟超过设定阈值时，该从库会被禁用，不再提供查询服务，如果所有从节点的延迟都超过了设定阈值，可由主库承担读请求。当延迟恢复到低于阈值以内时，从库将重新提供读服务。该校验机制满足了不同业务场景对主从延迟要求的需求。配置方法参考数据库发现中关于 delay-milliseconds-threshold 参数的说明。

同时，通过 SphereEx-DBPlusEngine 的熔断能力可手动禁用和启用从读节点，用来应对如硬件、软件升级、网络维护等情况。

在读写分离的类别方面，SphereEx-DBPlusEngine 支持基于数据源的读写分离和基于 SQL Hint 的读写分离。

■ 基于数据源

基于数据源的读写分离场景，SphereEx-DBPlusEngine 识别到 INSERT、DELETE、UPDATE 请求和 SELECT 请求，将 SQL 分别路由至不同的数据源中，如下图所示。

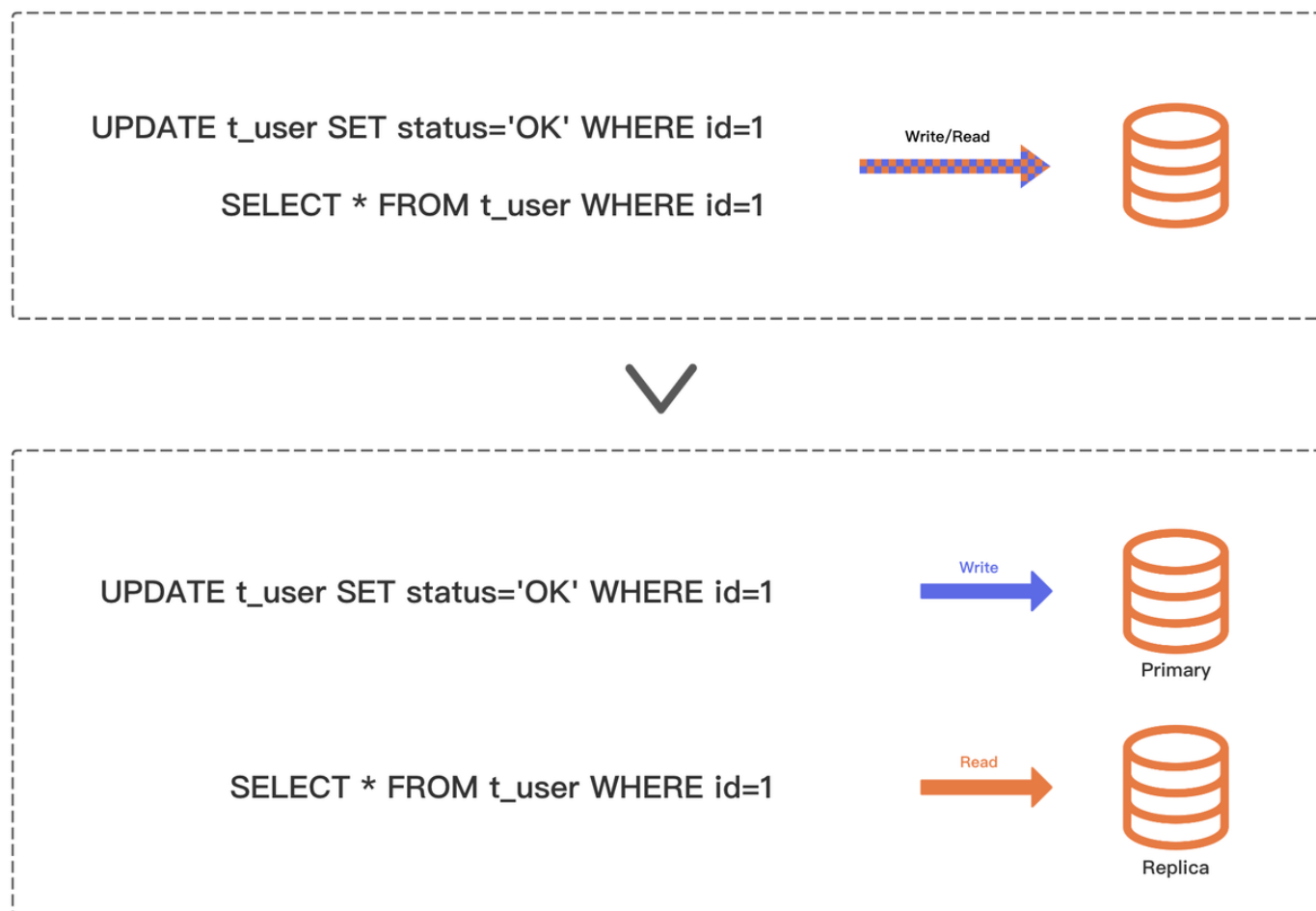


Fig. 6: 读写分离原理图

■ 基于 SQL Hint

基于 SQL Hint 的读写分离场景，SphereEx-DBPlusEngine 可根据 SQL Hint 信息指定 SQL 配置读写规则，如指定某查询 SQL 必须检索主库数据，可满足对数据一致性有要求的业务场景。

7.3.3 使用指南

关于动态读写分离和静态读写分离的说明如下：-动态读写分离：SphereEx-DBPlusEngine 自动探测（依赖于数据库发现规则）数据库集群的主从角色，当数据库节点的角色发生变化时，读写分离的配置会自动调整，可支持 MGR、MHA 的高可用方案；-静态读写分离：在 SphereEx-DBPlusEngine 中固定主库和从库的配置，它不会动态感知主从角色变化。当出现角色变更时，需人工介入对读写分离做相应调整，可支持多种高可用方案。

1. 环境确认

在使用 SphereEx-DBPlusEngine 读写分离插件前，需要对当前环境做较全面的确认，如数据库的主从架构信息、主从所有节点的硬件配置信息以及主从节点间网络的延迟等信息。

2. 规则确认

■ 动静规则

SphereEx-DBPlusEngine 提供了动态和静态读写分离配置规则。

对于 MGR 和 MHA 架构，建议使用动态读写分离规则，SphereEx-DBPlusEngine 会自动识别节点的角色，即可自适应主从角色的变化，无需人工介入调整读写分离策略。

对于使用了域名、VIP 的数据库集群，或 1 主 1 从最简架构，建议使用静态读写分离规则。

■ 负载均衡

对于存在多个从节点的环境，一般场景里可考虑使用轮询负载。如从库的主机资源存在差异，可以考虑配置权重比例。

3. 数据源确认及注册

依次确认所有数据源的连通性，可在 SphereEx-DBPlusEngine 所在节点进行访问确认。

在 DBPlusEngine 中，通过 REGISTER STORAGE UNIT 命令对数据源进行注册。

4. 创建数据库发现规则（可选）

如果选择使用动态读写分离规则，则需提前配置好数据库发现规则，如业务明确的主从延迟要求，需在该步骤进行配置。

5. 创建读写分离规则

创建读写分离规则时，根据第二步所规划的规则进行配置，包括动静规则、负载均衡规则以及对应比例。

6. 配置验证

创建测试表，使用 PREVIEW SQL 命令验证增删改查的路由情况，SELECT 语句的路由应指向从节点。

7.3.4 操作指南

动态读写分离的配置需要依赖于数据库发现规则，SphereEx-DBPlusEngine 可在数据库集群角色发生变化时动态、实时调整读写分离策略。由于本章节为读写分离插件介绍，因此关于数据库发现能力更详细说明，请参考[数据库发现](#)，操作步骤如下。

- 1. 在 MGR 中创建数据库，准备好数据源
- 2. 在 SphereEx-DBPlusEngine 中完成逻辑库的创建
- 3. 添加数据源，完成集群构建
- 4. 创建数据库发现规则
- 5. 创建动态读写分离规则
- 6. 创建测试表并插入测试数据
- 7. 依次验证读/写请求的执行路径

对于静态读写分离，配置读写数据源信息即可，无需关注数据库发现规则。

7.3.5 操作示例

SphereEx-DBPlusEngine 的读写分离配置相对简单，下面通过两个常见的场景来快速了解 SphereEx-DBPlusEngine 动态和静态读写分离能力。

动态读写分离配置

环境说明

本场景基于 MGR 架构配置动态读写分离，使用 4 台测试机，其中 1 台用来部署 SphereEx-DBPlusEngine，另外 3 台为 MGR 一主两从。因该场为功能演示，因此测试机采用相对基本的配置（1c/2g）即可。

\	实例	IP 地址	服务端口	主机名	备注
1	DBPlusEngine 1.2.0	192.168.xx.102	3307	dbplusengine	
2	MySQL 8.0.28	192.168.xx.103	3306	db_1	MGR Primary
3	MySQL 8.0.28	192.168.xx.104	3306	db_2	MGR Replica
4	MySQL 8.0.28	192.168.xx.105	3306	db_3	MGR Replica

拓扑图

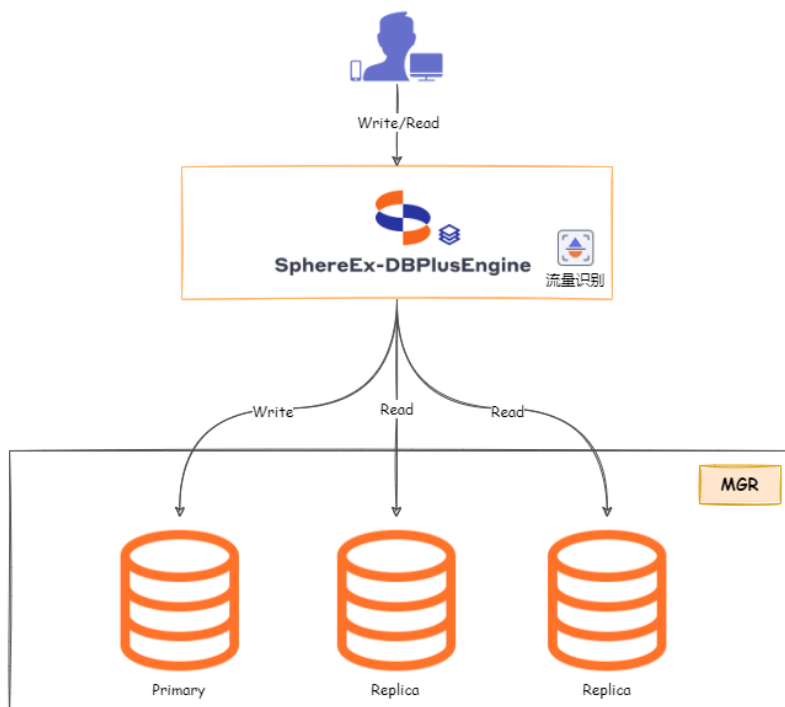


Fig. 7: 动态读写分离

配置过程

1. 在 MGR 中创建数据库，为后续构建集准备好数据源。

```
--登陆 MySQL 实例，创建名为 test 的数据库
# mysql -uroot -p -h192.168.xx.103 -P3306
mysql> CREATE DATABASE test;
Query OK, 1 row affected (0.05 sec)
```

```
mysql> SHOW DATABASES;
```

```
+-----+
| Database |
+-----+
| information_schema |
| mysql            |
| performance_schema |
| sys              |
| test             |
+-----+
```

```
5 rows in set (0.00 sec)
```

2. 使用 MySQL 客户端登陆 SphereEx-DBPlusEngine，创建逻辑库。

```
--访问 DBPlusEngine 实例
# mysql -uroot -p -P3307 -h192.168.xx.102
mysql> SHOW DATABASES;
+-----+
| schema_name |
+-----+
| mysql        |
| information_schema |
| performance_schema |
| sys          |
+-----+
4 rows in set (0.06 sec)

mysql> CREATE DATABASE testdb;
Query OK, 0 rows affected (0.05 sec)

mysql> SHOW DATABASES;
+-----+
| schema_name |
+-----+
| mysql        |
| information_schema |
| performance_schema |
| sys          |
| testdb       |
+-----+
5 rows in set (0.06 sec)
```

3. 添加数据源，完成集群构建。

在 SphereEx-DBPlusEngine 注册 MGR 的各个节点。以下所添加资源为 Schema 级别，相比实例级别粒度更细。

```
mysql> USE testdb;
Database changed

mysql> REGISTER STORAGE UNIT ds_0 (
  URL="jdbc:mysql://192.168.xx.103:3306/test?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
), ds_1 (
  URL="jdbc:mysql://192.168.xx.104:3306/test?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
), ds_2 (
  URL="jdbc:mysql://192.168.xx.105:3306/test?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
);
Query OK, 0 rows affected (0.67 sec)

mysql> SHOW STORAGE UNITS\G
```

4. 创建数据库发现规则。

动态读写分离的配置需要依赖于数据库发现规则，SphereEx-DBPlusEngine 即可在数据库集群角色发生变化时动态、实时调整读写分离策略。本章节为读写分离插件介绍，因此关于数据库发现能力更详细说明，请参考[数据库发现](#)。

```
--创建数据库拓扑发现规则
mysql> CREATE DB_DISCOVERY RULE primary_replica_rule (
  STORAGE_UNITS(ds_0,ds_1,ds_2),
  TYPE(NAME='MySQL.MGR',PROPERTIES('group-name'='b309fcc3-93e8-11ec-b5bd-080027c850b1')),
  HEARTBEAT(PROPERTIES('keep-alive-cron'='0/5 * * * * ?'))
);
Query OK, 0 rows affected (0.85 sec)
```

```
mysql> SHOW DB_DISCOVERY RULES primary_replica_rule\G
***** 1. row *****
      group_name: primary_replica_rule
      data_source_names: ds_0,ds_1,ds_2
primary_data_source_name: ds_0
      discovery_type: {name=primary_replica_ds1_MySQL.MGR, type=MySQL.MGR, props={group-name=b309fcc3-93e8-11ec-
b5bd-080027c850b1}}
      discovery_heartbeat: {name=primary_replica_rule_heartbeat}
1 rows in set (0.01 sec)
```

该步骤创建了一个数据库的发现规则，发现的数据源范围是 ds_0、ds_1 和 ds_2，并标注数据库集群类型为 MGR，SphereEx-DBPlusEngine 每 5 秒探测一次拓扑结构及各个数据源的角色。

5. 创建动态读写分离规则。

基于上一步配置的数据库拓扑发现规则来创建动态的读写分离规则。AUTO_AWARE_RESOURCE 对应着数据库拓扑发现规则的名称。

```
mysql> CREATE READWRITE_SPLITTING RULE dynamic_readwrite_rule (
      AUTO_AWARE_RESOURCE=primary_replica_rule
);
Query OK, 0 rows affected (0.06 sec)

mysql> SHOW READWRITE_SPLITTING RULES;
+-----+-----+-----+-----+-----+
| name          | auto_aware_data_source_name | write_data_source_name | read_data_source_names | load_balancer_type |
| load_balancer_props |
+-----+-----+-----+-----+-----+
| dynamic_readwrite_rule | primary_replica_rule      | ds_0                  | ds_1,ds_2              |                    |
+-----+-----+-----+-----+-----+
1 row in set (0.05 sec)
```

6. 创建测试表，并插入若干条记录。

```
mysql> DROP TABLE IF EXISTS t_user;
Query OK, 0 rows affected (1.16 sec)

mysql> CREATE TABLE `t_user` (
  `id` int(8) NOT NULL,
  `mobile` char(20) NOT NULL,
  `id_card_no` varchar(18) NOT NULL,
  PRIMARY KEY (`id`)
);
Query OK, 0 rows affected (1.20 sec)

mysql> INSERT INTO t_user (id, mobile, id_card_no) VALUES
(1,18236483857, 220605194709308170),
(2,15686689114, 360222198806088804),
(3,14523360225, 411601198601098107),
(4,18143924353, 540228199804231247),
(5,15523349333, 360924195311103360);
Query OK, 5 rows affected (0.40 sec)
```

7. 验证写请求的执行路径。

PREVIEW 是 DBPlusEngine 用来确认 SQL 执行路由的命令，当命令执行后，只输出执行路径、不会真实被执行。

```
mysql> PREVIEW INSERT INTO t_user (id, mobile, id_card_no) VALUES (6,13261527931, 513229195302236086);
+-----+-----+
| data_source_name | actual_sql |
+-----+-----+
| ds_0            | INSERT INTO t_user (id, mobile, id_card_no) VALUES (6,13261527931, 513229195302236086) |
+-----+-----+
1 row in set (0.01 sec)
```

```
mysql> PREVIEW INSERT INTO t_user (id, mobile, id_card_no) VALUES (7,13921892133, 500108194806107214);
+-----+
| data_source_name | actual_sql |
+-----+
| ds_0             | INSERT INTO t_user (id, mobile, id_card_no) VALUES (7,13921892133, 500108194806107214) |
+-----+
1 row in set (0.01 sec)
```

通过以上输出信息确认，不同的 INSERT 请求均被 SphereEx-DBPlusEngine 路由到了 ds_0 当中，即 MGR 的主库，结果符合预期。

8. 验证读请求的执行路径。

接下来通过 PREVIEW 命令验证查询操作的路由，根据读写分离的配置规则，返回结果应为 ds_1 或 ds_2。

```
mysql> PREVIEW SELECT * FROM t_user WHERE id=1;
+-----+
| data_source_name | actual_sql |
+-----+
| ds_1             | SELECT * FROM t_user WHERE id=1 |
+-----+
1 row in set (0.05 sec)
```

```
mysql> PREVIEW SELECT * FROM t_user WHERE id=1;
+-----+
| data_source_name | actual_sql |
+-----+
| ds_2             | SELECT * FROM t_user WHERE id=1 |
+-----+
1 row in set (0.06 sec)
```

```
mysql> PREVIEW SELECT * FROM t_user WHERE id=2;
+-----+
| data_source_name | actual_sql |
+-----+
| ds_1             | SELECT * FROM t_user WHERE id=2 |
+-----+
1 row in set (0.01 sec)
```

```
mysql> PREVIEW SELECT * FROM t_user WHERE id=2;
+-----+
| data_source_name | actual_sql |
+-----+
| ds_2             | SELECT * FROM t_user WHERE id=2 |
+-----+
1 row in set (0.05 sec)
```

通过以上输出信息确认，SELECT 请求被 SphereEx-DBPlusEngine 路由到了 ds_1 和 ds_2 中，同时以轮询方式进行查询，结果符合预期。

到这里，通过 SphereEx-DBPlusEngine 完成了基于 MGR 一主两从的动态读写分离场景的配置。

静态读写分离配置

相比动态读写分离，由于没有数据库拓扑发现的配置，静态的配置会更简单一些，准备好数据源之后，直接配置规则即可。同时，该示例中加入了权重的配置参考。

1. 在主从集群中创建数据库，准备好数据源
2. 在 SphereEx-DBPlusEngine 中完成逻辑库的创建
3. 添加数据源，完成集群构建
4. 创建静态读写分离规则
5. 创建测试表并插入测试数据
6. 依次验证读/写请求的执行路径

环境说明

\	实例	IP 地址	服务端口	主机名	备注
1	DBPlusEngine 1.2.0	192.168.xx.102	3307	dbplusengine	
2	MySQL 8.0.28	192.168.xx.103	3306	db_1	MGR Primary
3	MySQL 8.0.28	192.168.xx.104	3306	db_2	MGR Replica
4	MySQL 8.0.28	192.168.xx.105	3306	db_3	MGR Replica

拓扑图

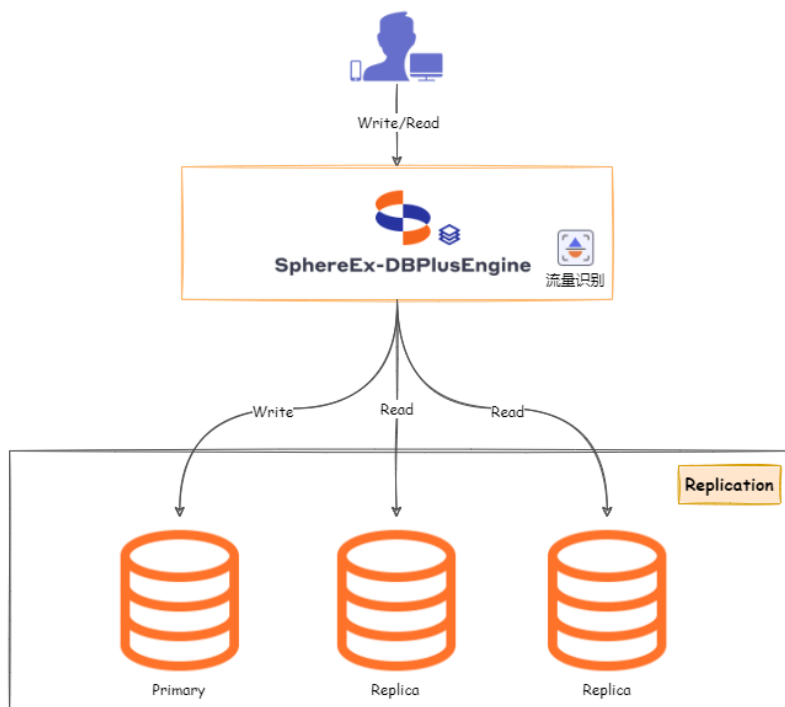


Fig. 8: 静态读写分离

配置过程

1. 在主从集群中创建数据库，为后续构建集准备好数据源。

```
# mysql -uroot -p -h192.168.xx.103 -P3306
mysql> CREATE DATABASE test;
Query OK, 1 row affected (0.05 sec)
```

```
mysql> SHOW DATABASES;
```

```
+-----+
| Database |
+-----+
| information_schema |
| mysql            |
| performance_schema |
| sys              |
| test             |
+-----+
5 rows in set (0.00 sec)
```

2. 使用 MySQL 客户端登陆 SphereEx-DBPlusEngine，创建逻辑库。

```
# mysql -uroot -p -P3307 -h192.168.xx.102
mysql> SHOW DATABASES;
+-----+
| schema_name |
+-----+
| mysql       |
| information_schema |
| performance_schema |
| sys        |
+-----+
4 rows in set (0.00 sec)
```

```
mysql> CREATE DATABASE testdb;
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> SHOW DATABASES;
+-----+
| schema_name |
+-----+
| mysql       |
| information_schema |
| performance_schema |
| sys        |
| testdb     |
+-----+
5 rows in set (0.05 sec)
```

3. 添加数据源，完成集群构建。

```
mysql> USE testdb;
Database changed

mysql> REGISTER STORAGE UNIT ds_0 (
  URL="jdbc:mysql://192.168.xx.103:3306/test?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
), ds_1 (
  URL="jdbc:mysql://192.168.xx.104:3306/test?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
), ds_2 (
  URL="jdbc:mysql://192.168.xx.105:3306/test?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
);
Query OK, 0 rows affected (0.62 sec)

mysql> SHOW STORAGE UNITS\G
```

4. 创建静态读写分离规则。

在创建静态读写分离的过程中，需要在指定资源的位置配置所有的数据源名称。

```
mysql> CREATE READWRITE_SPLITTING RULE ms_group_0 (
  WRITE_STORAGE_UNIT=ds_0,
  READ_STORAGE_UNITS(ds_1,ds_2),
  TYPE(NAME="random")
);
Query OK, 0 rows affected (0.26 sec)
```

```
mmysql> SHOW READWRITE_SPLITTING RULES;
```

```
+-----+-----+-----+-----+-----+
| name | auto_aware_data_source_name | write_data_source_query_enabled | write_storage_unit_name | read_storage_unit_names | load_balancer_type | load_balancer_props |
+-----+-----+-----+-----+-----+
|      |                               |                                |                          |                          |                    |                      |
```



```

+-----+-----+-----+-----+-----+-----+
| ms_group_0 |          | ds_0 | ds_1,ds_2 | random |          |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

如上，即创建好了基于 1 主 2 从架构的静态读写分离规则。

5. 创建测试表，并插入若干条记录。

```

--在 SphereEx-DBPlusEngine 中执行以下命令
mysql> DROP TABLE IF EXISTS t_user;
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE `t_user` (
  `id` int(8) NOT NULL,
  `mobile` char(20) NOT NULL,
  `id_card_no` varchar(18) NOT NULL,
  PRIMARY KEY (`id`)
);
Query OK, 0 rows affected (1.15 sec)

mysql> INSERT INTO t_user (id, mobile, id_card_no) VALUES
(1,18236483857, 220605194709308170),
(2,15686689114, 360222198806088804),
(3,14523360225, 411601198601098107),
(4,18143924353, 540228199804231247),
(5,15523349333, 360924195311103360);
Query OK, 5 rows affected (0.16 sec)

```

6. 验证写请求的执行路径。

通过 PREVIEW 命令确认 INSERT 语句的路由。

```

mysql> PREVIEW INSERT INTO t_user (id, mobile, id_card_no) VALUES (6,13261527931, 513229195302236086);
+-----+-----+
| data_source_name | actual_sql |
+-----+-----+
| ds_0 | INSERT INTO t_user (id, mobile, id_card_no) VALUES (6,13261527931, 513229195302236086) |
+-----+-----+
1 row in set (0.15 sec)

mysql> PREVIEW INSERT INTO t_user (id, mobile, id_card_no) VALUES (7,13921892133, 500108194806107214);
+-----+-----+
| data_source_name | actual_sql |
+-----+-----+
| ds_0 | INSERT INTO t_user (id, mobile, id_card_no) VALUES (7,13921892133, 500108194806107214) |
+-----+-----+
1 row in set (0.00 sec)

```

通过以上输出信息确认，不同的 INSERT 请求均被 SphereEx-DBPlusEngine 路由到了 ds_0 当中，即写入请求路到了主库，结果符合预期。

7. 验证读请求的执行路径。

在规则中配置了读库权重信息，下面验证随机读是否生效。

```

mysql> PREVIEW SELECT * FROM t_user WHERE id=1;
+-----+-----+
| data_source_name | actual_sql |
+-----+-----+
| ds_1 | SELECT * FROM t_user WHERE id=1 |
+-----+-----+
1 row in set (0.10 sec)

```

```
mysql> PREVIEW SELECT * FROM t_user WHERE id=1;
```

```
+-----+-----+
| data_source_name | actual_sql |
+-----+-----+
| ds_1             | SELECT * FROM t_user WHERE id=1 |
+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> PREVIEW SELECT * FROM t_user WHERE id=1;
```

```
+-----+-----+
| data_source_name | actual_sql |
+-----+-----+
| ds_2             | SELECT * FROM t_user WHERE id=1 |
+-----+-----+
1 row in set (0.04 sec)
```

通过以上输出信息确认，SELECT 请求被 SphereEx-DBPlusEngine 路由到了 ds_1 和 ds_2 中，从输出比例来看为 2:1 权重，结果符合预期。

到这里，通过 SphereEx-DBPlusEngine 完成了基于 1 主 2 从架构的静态读写分离场景的配置。

基于 SQL Hint

SQL Hint 可将指定 SQL 路由到任意数据源，因此在读写分离的场景中，可通过 SQL Hint 对指定 SQL 做更灵活的配置。如某 SQL 必须检索主库数据的场景，可以使用 SQL Hint 的方式满足对数据一致性有要求的业务场景。

基于上一个场景的环境，下面验证下使用 SQL Hint 进行查询，通过 Hint 的方式将 SELECT * FROM t_user 请求路由到主库。

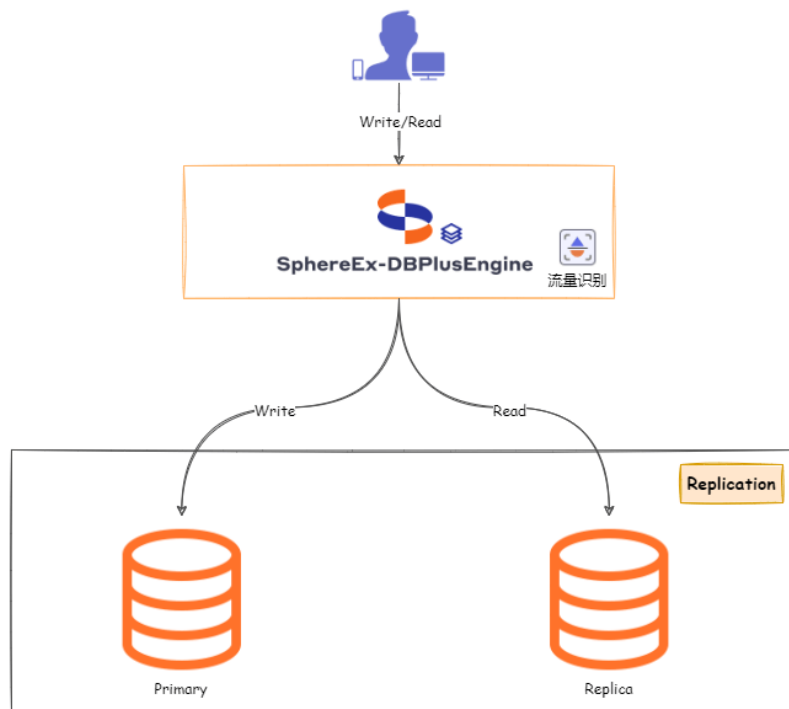


Fig. 9: 使用 Hint

具体的使用方式是在 SQL 头部加上 `/* SHARDINGSPHERE_HINT: WRITE_ROUTE_ONLY=true */` 这样的注释信息即可。为了验证 Hint 的干预结果，再次使用 PREVIEW 命令来验证 SQL 的执行路径，操作如下。

```
# mysql -uroot -p -h192.168.xx.102 -P3307 -c
-- 如果使用 MySQL 客户端进行验证，请务必加上 '-c' 参数，否则 hint 信息将无法生效

mysql> USE testdb;

mysql> PREVIEW SELECT * FROM t_user;
+-----+-----+
| data_source_name | actual_sql |
+-----+-----+
| ds_1             | SELECT * FROM t_user |
+-----+-----+
1 row in set (0.04 sec)

mysql> /* SHARDINGSPHERE_HINT: WRITE_ROUTE_ONLY=true */ PREVIEW SELECT * FROM t_user;
+-----+-----+
| data_source_name | actual_sql |
+-----+-----+
| ds_0             | /* SHARDINGSPHERE_HINT: WRITE_ROUTE_ONLY=true */ SELECT * FROM t_user |
+-----+-----+
1 row in set (0.00 sec)

-- 在执行时，请确保 /* SHARDINGSPHERE_HINT: WRITE_ROUTE_ONLY=true */ 前没有空格
```

通过以上输出可见，当使用 Hint 进行干预时，指定的读请求语句已经被路由到了主库，符合预期。

7.3.6 参数

■ delay-milliseconds-threshold

使用方式：delay-milliseconds-threshold: 时长 (ms)

参数说明：该参数用来配置动态读写分离规则对主从延迟的限制，单位为 ms，取值范围为正整数。当超过设定阈值时，该从库将被禁用掉，当所有从库都超过设定阈值时，主库负责读和写。该参数需要在数据库发现规则中进行配置，请参考[数据库发现说明](#)。

7.3.7 FAQ

1. 当从库出现宕机故障后，读写分离的规则是否需要做调整？

如果配置的是动态读写分离规则，从库宕机后不需要调整规则。

如果配置的是静态读写分离规则，则需运维人员介入，结合实际情况修改规则中的资源配置。

2. 数据库集群添加新的从库后，如 1 主 2 从变为 1 主 3 从，读写分离的规则需要如何调整？

首先在数据库集群中添加从节点，确认数据已经完成同步。然后使用 DistSQL 将新的从节点信息配置到读写分离规则当中，配置完成后立即生效。

3. 当数据库集群主从角色发生切换后，对动、静态读写分离的影响有哪些？

对动态读写分离规则无影响，动态读写分离基于数据库发现规则，读写流量会根据最新角色关系做数据路由。

对静态读写分离规则有影响，发生角色切换后，需运维人员及时变更读写分离策略。

4. 当从库出现网络闪断时，业务的读请求会被如何处理？

基于动态规则：业务无影响，读请求只会路由到网正常的节点；

基于静态规则：业务受影响，读请求会根据静态规则继续路由读请求，在网络闪断期间，业务会出现超时情况。需运维人员介入，及时禁用该节点，待网络恢复正常后再启用。

7.4 数据加密

安全控制一直是治理的重要环节，数据安全一直是极为重视和敏感的话题。数据加密是指对某些敏感信息通过加密规则进行数据的变形，实现敏感隐私数据的可靠保护。涉及客户安全数据或者一些商业性敏感数据，如身份证号、手机号、卡号、客户号等个人信息按照相关部门规定，都需要进行数据加密。

对于数据加密的需求，在现实的业务场景中一般分为两种情况：

1. 新业务上线，安全部门规定需将涉及用户敏感信息，例如银行、手机号码等进行加密后存储到数据库，在使用的时候再进行解密处理。因为是全新系统，因而没有存量数据清洗问题，所以实现相对简单。
2. 已上线业务，之前一直将明文存储在数据库中。相关部门突然需要对已上线业务进行加密整改。这种场景一般需要处理 3 个问题：
 - 历史数据需要进行加密处理，即洗数。
 - 如何能在不改动业务 SQL 和逻辑情况下，将新增数据进行加密处理，并存储到数据库；在使用时，再进行解密取出。
 - 如何较为安全、无缝、透明化地实现业务系统在明文与密文数据间的迁移。

7.4.1 概述

SphereEx-DBPlusEngine 根据业界对加密的需求及业务改造痛点，提供了一套完整、安全、透明化、低改造成本的数据加密整合解决方案，目前 SphereEx-DBPlusEngine 内置了 MD5、AES、RC4、SM4 和 SM3 算法。

在真实业务场景中，相关业务开发团队则往往需要针对公司安全部门需求，自行实行并维护一套加解密系统。而当加密场景发生改变时，自行维护的加密系统往往又面临着重构或修改风险。此外，对于已经上线的业务，在不修改业务逻辑和 SQL 的情况下，透明化、安全低风险地实现无缝进行加密改造也相对复杂。对于一些安全级别更高的非幂等加密算法，提供不可逆的幂等列用于查询，即查询辅助列。

为了提升功能的易用性，SphereEx-DBPlusEngine 数据加密插件提供了云端密钥管理、洗数以及反洗数能力。

基本概念

- 逻辑列

用于计算加解密列的逻辑名称，是 SQL 中列的逻辑标识。逻辑列包含密文列（必须）、查询辅助列（可选）和明文列（可选）。

- 逻辑列类型（dataType）

用于定义逻辑列的类型，例如 INT NOT NULL，VARCHAR(200) DEFAULT NULL 等，具体可以参见官方文档中各种方言字段的定义。

- 密文列（cipherColumn）

加密后的数据列。

- 密文列类型（cipherDataType）

用于定义密文列的类型，同逻辑列类型。

- 查询辅助列（assistedQueryColumn）

用于查询的辅助列。对于一些安全级别更高的非幂等加密算法，提供不可逆的幂等列用于查询。

- 查询辅助列类型（assistedQueryDataType）

用于定义查询辅助列类型，可结合实际需要进行人工定义。

- 模糊查询列

对于需要模糊查询的场景，该字段用于实现模糊查询。

- 模糊查询列类型

用于定义模糊查询列类型，可结合实际需要进行人工定义。

- 明文列（plainColumn）

存储明文的列，用于在加密数据迁移过程中仍旧提供服务。在清洗数据结束后可以删除。

- 明文列类型（plainDataType）

用于定义明文列类型，同逻辑列类型。

- 加密算法

指对数据加密过程中，所使用的具体加密方式，如 AES 及 MD5 等。

- 加密洗数（encrypting）

对数据库中未加密数据进行批量加密。

- 解密洗数（decrypting）

对数据库中已加密数据进行批量解密。

- 密钥管理

密钥的存放及管理方式。

特色能力

■ 云端密钥管理

由于加密 key 以明文形式配置在 props 文件中存在安全隐患，SphereEx-DBPlusEngine 可通过增加 AWS 云端密钥管理功能来管理加密的 key。例如利用 AWS 的 secretKey 功能保存密钥，从而提升整个加密的安全性和便捷性。程序在初始化加密算法时，会同 AWS 建立连接，从而获取存储在 AWS 中的相关密钥，然后将密钥存储在算法中。在整个数据加密的过程中不涉及与云端的网络交互。

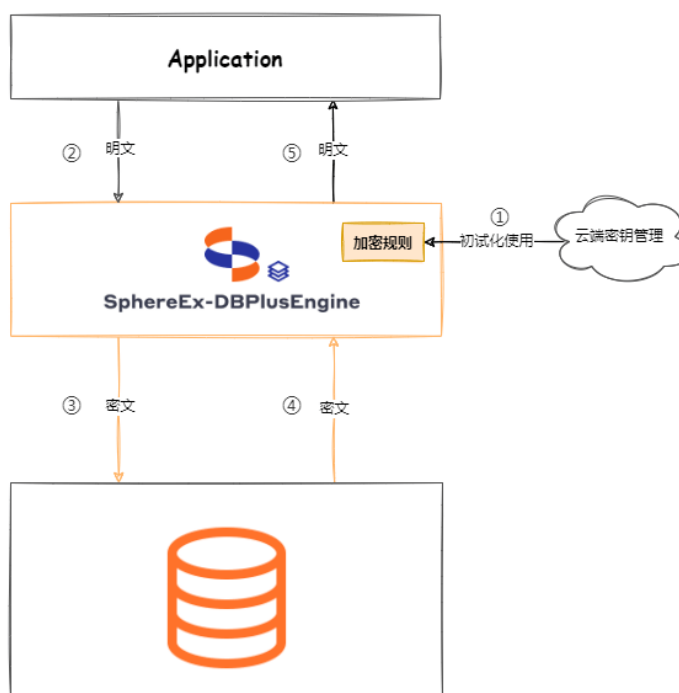


Fig. 10: 云端密钥管理

■ 加密洗数

目前 SphereEx-DBPlusEngine 已经提供了加密的解决方案，对于新表新业务而言，直接使用加密规则配置即可，但是对于已有数据表，就需要将这些表中的明文字段进行洗数，转化为加密内容，SphereEx-DBPlusEngine 提供了完整的洗数方案。加密洗数通过 DistSQL 来触发洗数任务，程序收到洗数任务的请求之后，会根据当前的洗数规则以及加密规则创建洗数任务。洗数任务主要由两部分构成，一部分是查询任务，一部分是更新任务，查询任务负责查询用户的表数据并获取需要加密的明文字段，然后推送到通道中；更新任务则从通道中获取数据，并加密更新。整个任务的创建以及执行过程都会与治理中心进行交互，因此用户可以通过相关 DistSQL 来查询任务进度以及清理任务。

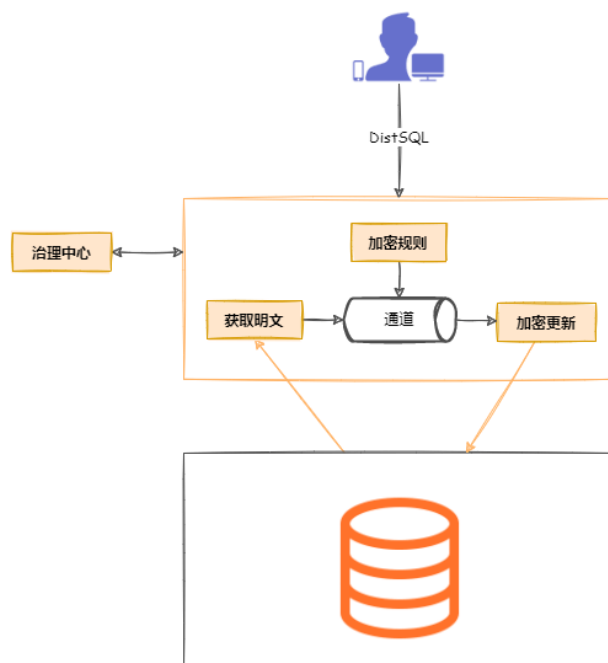


Fig. 11: 加密洗数

■ 反洗数

对于已经是密文存储的系统，此时如果需要还原到明文存储的状态，可通过 SphereEx-DBPlusEngine 反向洗数的能力，构建出对应的明文列。在操作上与洗数类似，反洗数前提是需要提前创建好对应的明文列。

■ 重洗数

对于更换加密算法或更换密钥等情况，可通过 SphereEx-DBPlusEngine 来重新洗数。重洗数方案会调用反洗数和洗数流程，需要用户手动操作，并且过程中会有明文数据存储。

在以上洗数过程中，SphereEx-DBPlusEngine 支持断点续传的能力，为用户提供更好地洗数体验。

适用场景

■ 新上线业务的的数据加密场景

对于想要快速上线新业务，同时又需要完成安全部门的加密规定的场景，通过 SphereEx-DBPlusEngine 加密插件，用户可以快速完成数据的合规化加密，无需自行开发复杂的加密系统。同时 SphereEx-DBPlusEngine 加密锁具备的灵活性，也能够帮助客户避免加密场景变更带来的复杂重构和修改风险。

■ 已上线业务的数据加密改造场景

对于已经上线的成熟业务，用户不仅需要考虑历史数据的清洗，还需要考虑新旧功能的切换。通过 SphereEx-DBPlusEngine 加密插件，用户可以方便地完成系统的加密改造，它还能够帮助用户安全快速地切换新旧功能，对存量数据进行加密清洗。用户无需改动任何业务逻辑和 SQL 就能够透明化地使用加解密功能。

使用前提

SphereEx-DBPlusEngine 尽可能让业务无感完成加密改造，现阶段有部分功能存在受限情况，因此在加密设计阶段，需调研业务系统是否使用了如大于、小于等操作，SphereEx-DBPlusEngine 具体使用限制见下文。

使用限制

- 需自行处理数据库中原始的存量数据；
- 模糊查询支持%、_、contact 函数，暂不支持 escape；
- 加密字段无法支持查询不区分大小写功能；
- 加密字段无法支持比较操作，如：大于、小于、ORDER BY、BETWEEN 等；
- 加密字段无法支持计算操作，如：AVG、SUM 以及计算表达式；

注意事项

删除明文列是高危且不可逆操作，因此当完成数据加密改造后，建议先稳定运行数月后再执行该操作。

优势

- 自动化 & 透明化数据加密过程，用户无需关注加密中间实现细节。
- 内置多种加密算法，用户仅需简单配置即可使用。
- 提供加密算法 API 接口，用户可实现接口，从而使用自定义加密算法进行数据加密。
- 针对已上线业务，可实现明文数据与密文数据同步存储，并通过配置决定使用明文列还是密文列进行查询。可实现在不改变业务查询 SQL 前提下，已上线系统对加密前后数据进行安全、透明化迁移。

7.4.2 原理介绍

SphereEx-DBPlusEngine 通过对用户输入的 SQL 进行解析，并依据用户提供的加密规则对 SQL 进行改写，从而实现对原文数据进行加密，并将原文数据（可选）及密文数据同时存储到底层数据库。在用户查询数据时，它仅从数据库中取出密文数据，并对其解密，最终将解密后的原始数据返回给用户。

SphereEx-DBPlusEngine 自动化 & 透明化了数据加密过程，让用户无需关注数据加密的实现细节，像使用普通数据那样使用加密数据。此外，无论是已在线业务进行加密改造，还是新上线业务使用加密功能，SphereEx-DBPlusEngine 都可以提供一套相对完善的解决方案。

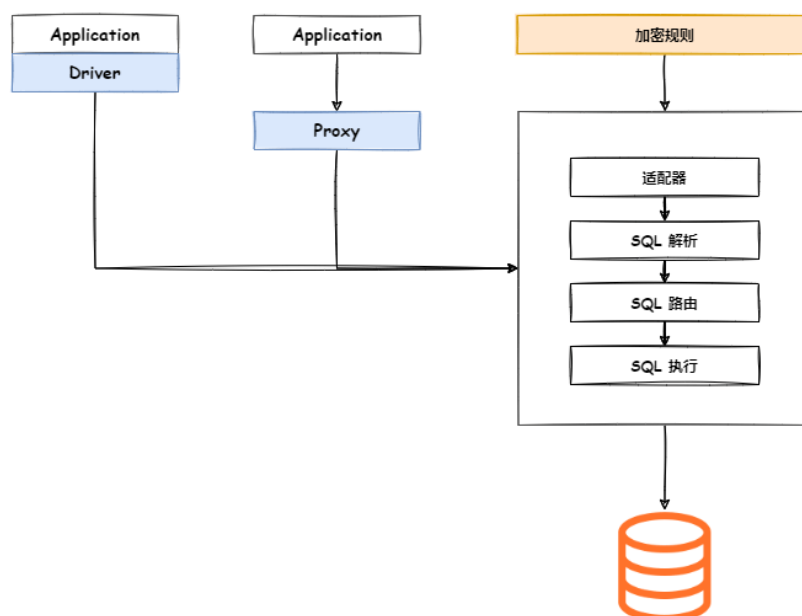


Fig. 12: 原理图

加密模块将用户发起的 SQL 进行拦截，并通过 SQL 语法解析器进行解析、理解 SQL 行为，再依据用户传入的加密规则，找出需要加密的字段和所使用的加解密算法对目标字段进行加解密处理后，再与底层数据库进行交互。SphereEx-DBPlusEngine 会将用户请求的明文进行加密后存储到底层数据库；并在用户查询时，将密文从数据库中取出进行解密后返回给终端用户。通过屏蔽对数据的加密处理，使用户无需感知解析 SQL、数据加密、数据解密的处理过程，就像在使用普通数据一样使用加密数据。

举例说明，假如数据库里有一张表叫做 t_user，这张表里实际有两个字段 pwd_plain，用于存放明文数据、pwd_cipher，用于存放密文数据、pwd_assisted_query，用于存放辅助查询数据，同时定义 logicColumn 为 pwd。那么，用户在编写 SQL 时应该面向 logicColumn 进行编写，即 INSERT INTO t_user SET pwd = '123'。SphereEx-DBPlusEngine 接收到该 SQL，通过用户提供的加密配置，发现 pwd 是 logicColumn，于是便对逻辑列及其对应的明文数据进行加密处理。SphereEx-DBPlusEngine 将面向用户的逻辑列与面向底层数据库的明文列和密文列进行了列名以及数据的加密映射转换。如下图所示：

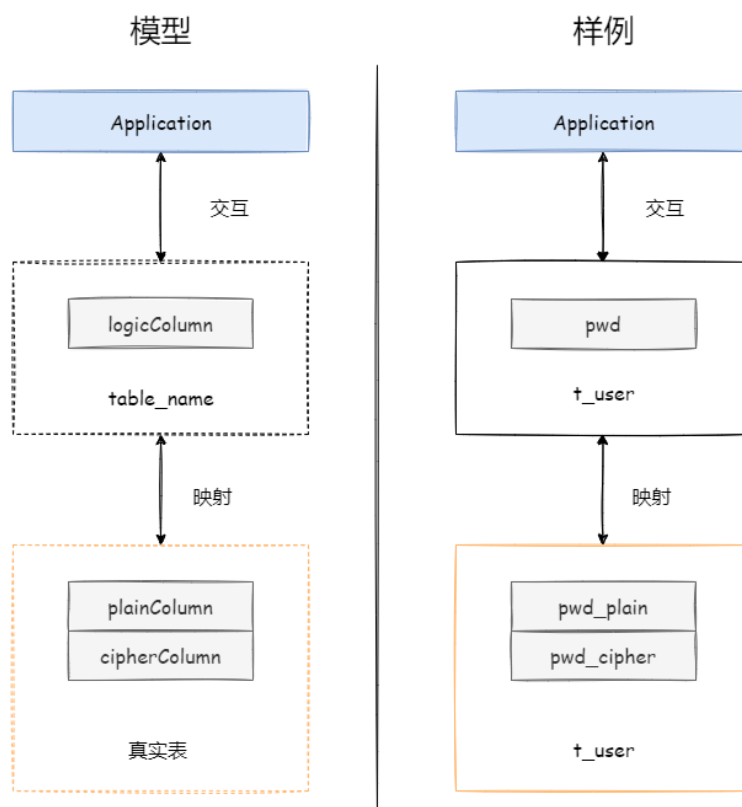


Fig. 13: 模型图

即依据用户提供的加密规则，将用户 SQL 与底层数据表结构割裂开来，使得用户的 SQL 编写不再依赖于真实的数据库表结构。而用户与底层数据库之间的衔接、映射、转换交由 SphereEx-DBPlusEngine 进行处理。

下方图片展示了使用加密模块进行增删改查时，其中的处理流程和转换逻辑，如下图所示。

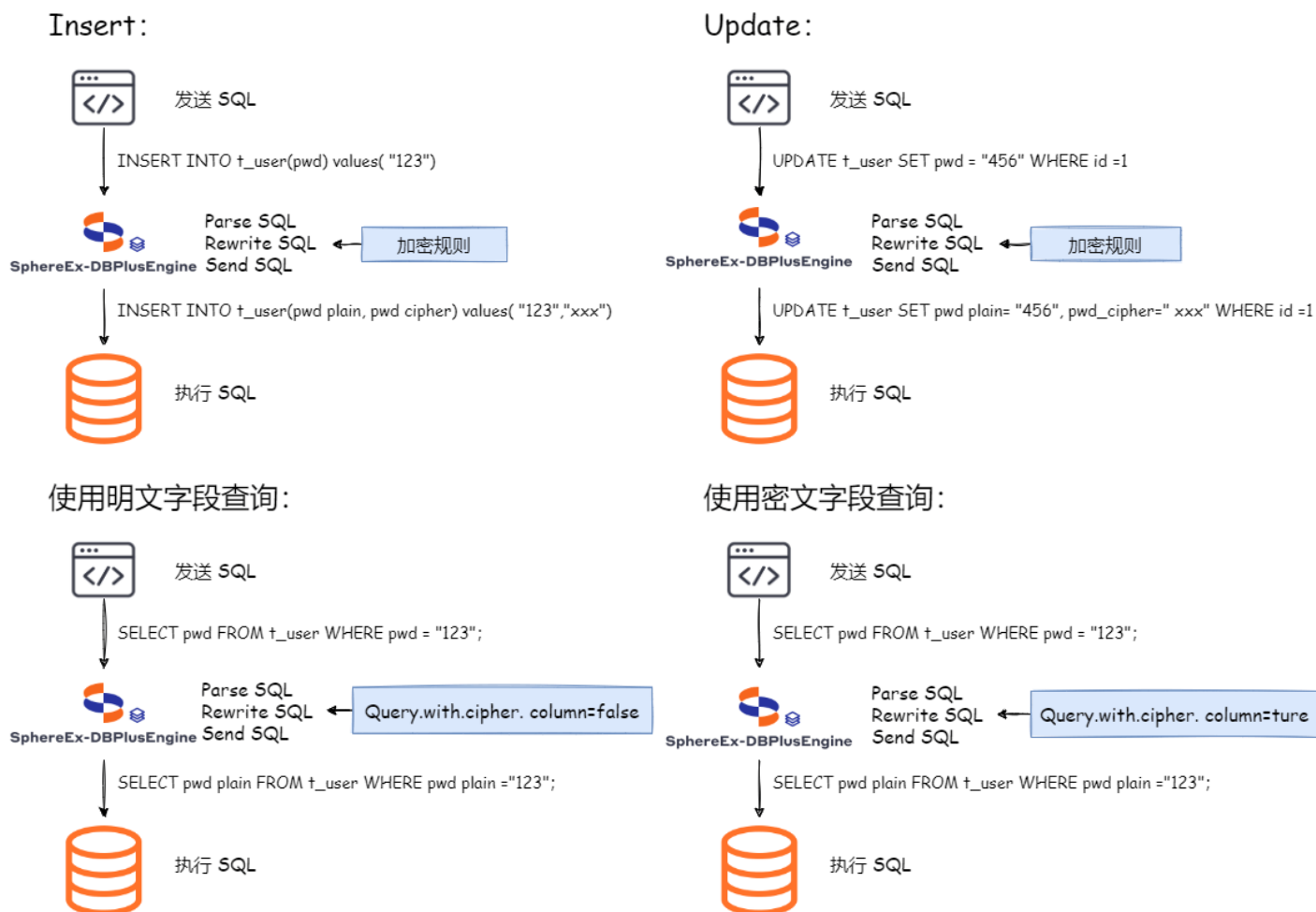


Fig. 14: 加密流程图

7.4.3 加密算法解析

SphereEx-DBPlusEngine 提供了两种加解密的接口，即 `EncryptAlgorithm` 和 `QueryAssistedEncryptAlgorithm`。一方面，SphereEx-DBPlusEngine 为用户提供了内置的加解密实现类，用户只需进行配置即可使用；另一方面，为了满足用户不同场景的需求，我们还开放了相关加解密接口，用户可依据这两种类型的接口提供具体实现类。再进行简单配置，即可让 SphereEx-DBPlusEngine 调用用户自定义的加解密方案进行数据加密。

- `EncryptAlgorithm` 该解决方案通过提供 `encrypt()`, `decrypt()` 两种方法对需要加密的数据进行加解密。在用户进行 INSERT, DELETE, UPDATE 时，SphereEx-DBPlusEngine 会按照用户配置，对 SQL 进行解析、改写、路由，并调用 `encrypt()` 将数据加密后存储到数据库，而在 SELECT 时，则调用 `decrypt()` 方法将从数据库中取出的加密数据进行逆向解密，最终将原始数据返回给用户。当前，SphereEx-DBPlusEngine 针对这种类型的加密解决方案提供了五种具体实现类，分别是 MD5(不可逆)，AES(可逆)，RC4(可逆)，SM3（不可逆）和 SM4（可逆），用户只需配置即可使用这五种内置的方案。
- `QueryAssistedEncryptAlgorithm`

相比较于第一种加密方案，该方案更为安全和复杂。它的理念是：即使是相同的数据，如两个用户的密码相同，它们在数据库里存储的加密数据也应当是不一样的。这种理念更有利于保护用户信息，防止撞库成功。

它提供三种函数进行实现，分别是 `encrypt()`, `decrypt()`, `queryAssistedEncrypt()`。在 `encrypt()` 阶段，用户通过设置某个变动种子，例如时间戳。针对原始数据 + 变动种子组合的内容进行加密，就能保证即使原始数据相同，也因为变动种子的存在，致使加密后的加密数据是不一样的。在 `decrypt()` 可依据之前规定的加密算法，利用种子数据进行解密。虽然这种方式确实可以增加数据的保密性，但是另一个问题却随之出现：相同的数据在数据库里存储的内容是不一样的，那么当用户按照这个加密列进行等值查询 (`SELECT FROM table WHERE encryptedColumnn = ?`) 时会发现无法将所有相同的原始数据查询出来。

为此，我们提出了辅助查询列的概念。该辅助查询列通过 queryAssistedEncrypt() 生成，与 decrypt() 不同的是，该方法通过对原始数据进行另一种方式的加密，但是针对原始数据相同的数据，这种加密方式产生的加密数据是一致的。将 queryAssistedEncrypt() 后的数据存储到数据中用于辅助查询真实数据。因此，数据库表中多出这一个辅助查询列。

由于 queryAssistedEncrypt() 和 encrypt() 产生不同加密数据进行存储，而 decrypt() 可逆，queryAssistedEncrypt() 不可逆。在查询原始数据的时候，我们会自动对 SQL 进行解析、改写、路由，利用辅助查询列进行 WHERE 条件的查询，却利用 decrypt() 对 encrypt() 加密后的数据进行解密，并将原始数据返回给用户。这一切都是对用户透明化的。当前，SphereEx-DBPlusEngine 针对这种类型的加密解决方案并没有提供具体实现类，却将该理念抽象成接口，提供给用户自行实现。SphereEx-DBPlusEngine 将调用用户提供的该方案的具体实现类进行数据加密。

7.4.4 加密配置

加密配置元素分为四部分：数据源配置，加密算法配置，加密表配置以及查询属性配置，其详情如下图所示：

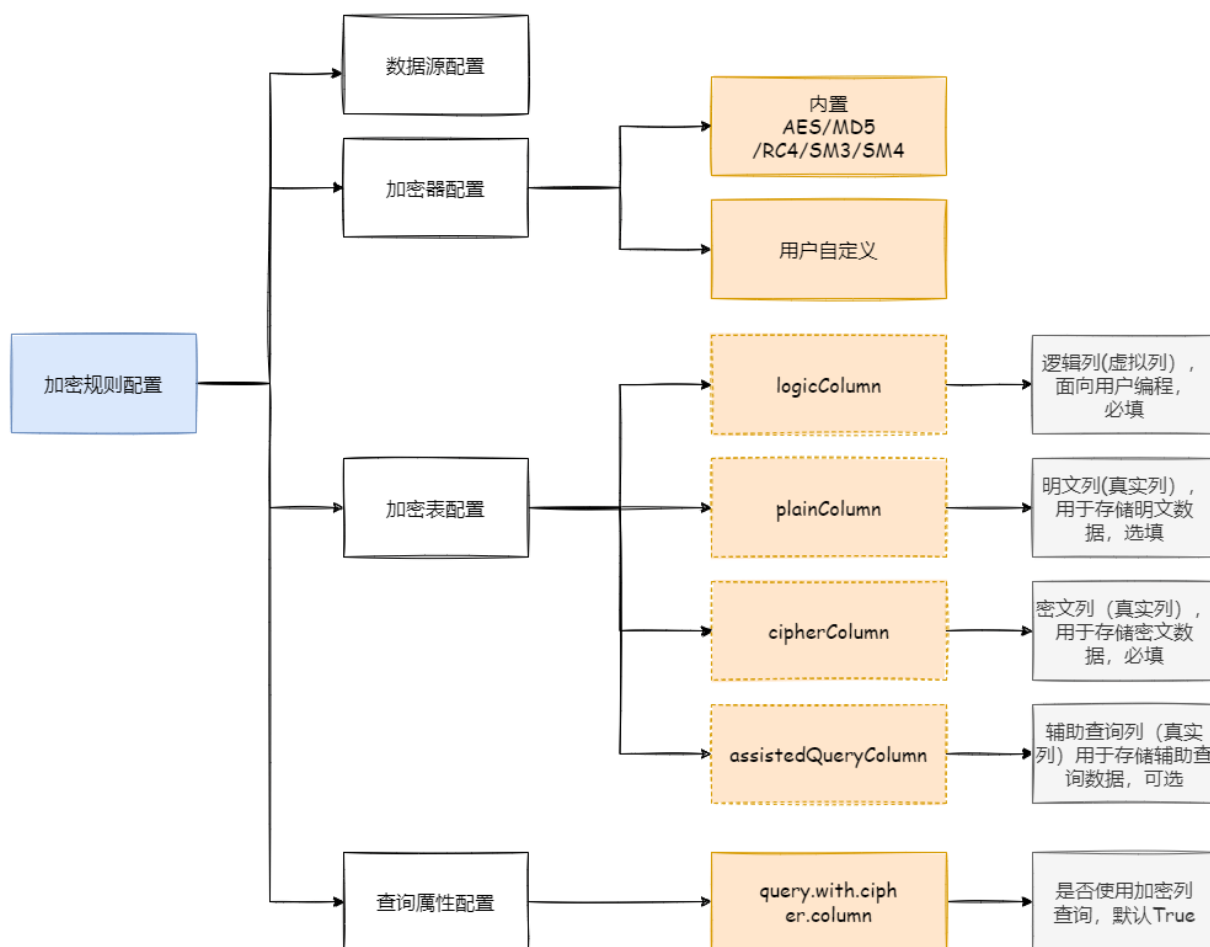


Fig. 15: 规则配置图

数据源配置：指数据源配置。

加密算法配置：指使用什么加密算法进行加解密。目前 SphereEx-DBPlusEngine 内置了五种加解密算法：AES，MD5，RC4，SM3 和 SM4。

加密表配置：用于告诉 SphereEx-DBPlusEngine 数据表里哪个列用于存储密文数据 (cipherColumn)、使用什么算法加解密 (encryptorName)、哪个列用于存储辅助查询数据 (assistedQueryColumn)、使用什么算法加解密 (assistedQueryEncryptorName)、哪个列用于存储明文数据 (plainColumn) 以及用户想使用哪个列进行 SQL 编写 (logicColumn)。

我们可以从加密模块存在的意义来理解。加密模块最终目的是希望屏蔽底层对数据的加密处理，也就是说我们不希望用户知道数据是如何被加解密的、如何将明文数据存储到 plainColumn，将密文数据存储到 cipherColumn。换句话说，我们不希望用户知道 plainColumn 和 cipherColumn 的存在和使用。所以，我们需要给用户提供一个概念意义上的列，这个列可以脱离底层数据库的真实列，它可以是数据库表里的一个真实列，也可以不是，从而使得用户可以随意改变底层数据库的 plainColumn 和 cipherColumn 的列名。或者删除 plainColumn，选择永远不再存储明文，只存储密文。只要用户的 SQL 面向这个逻辑列进行编写，并在加密规则里给出 logicColumn 和 plainColumn、cipherColumn 之间正确的映射关系即可。

查询属性的配置：当底层数据库表里同时存储了明文数据、密文数据后，该属性开关用于决定是直接查询数据库表里的明文数据进行返回，还是查询密文数据通过 SphereEx-DBPlusEngine 解密后返回。该属性开关支持字段级、表级别和全局规则级别配置，由 QUERY_WITH_CIPHER_COLUMN 进行配置，字段级别优先级最高。

7.4.5 使用指南

加密-新上线业务

新上线业务由于一切从零开始，不存在历史数据清洗问题，所以相对简单。选择合适的加密算法，如 AES 后，只需配置逻辑列（面向用户编写 SQL）和密文列（数据表存密文数据）即可，逻辑列和密文列可以相同也可以不同。建议配置如下（YAML 格式展示）：

```
-!ENCRYPT
encryptors:
  aes_encryptor:
    type: AES
    props:
      aes-key-value: 123456abc
tables:
  t_user:
    columns:
      pwd:
        cipherColumn: pwd
        encryptorName: aes_encryptor
```

使用这套配置，SphereEx-DBPlusEngine 只需将 logicColumn 和 cipherColumn 进行转换，底层数据表不存储明文，只存储了密文，这也是安全审计部分的要求所在。如果用户希望将明文、密文一同存储到数据库，只需添加 plainColumn 配置即可。操作流程如下：1. 设计加密表，确定加密字段，可结合表结构、数据特点及应用情况综合考虑；2. 在 SphereEx-DBPlusEngine 创建逻辑库，注册存储节点（空）；3. 创建加密规则，可使用 SphereEx-Console 或 DistSQL 完成；4. 创建加密表，完成加密表构建；5. 根据业务实际情况，再选择保留或删除明文字段。

洗数-已上线业务改造

对于已上线的业务，需评估改造后 SQL 的支持度，另外存量数据洗数是必不可少的一个环节。1. 确认加密范围根据合规需求，明确具体的加密数据范围。2. 确认加密算法选择并确认适合的加密算法。3. 梳理业务 SQL，确认支持情况确认当前业务中是否存在非等值的密态计算情况，如 like、>、< 等，若存在，则需考虑业务改造。4. 洗数完成以上内容后，对于已上线的项目，需要对存量数据进行洗数。该过程无需引用外部组件，SphereEx-DBPlusEngine 可完成洗数过程，该过程可在线进行。5. 启用密文列启用密文列之后，应用的请求将通过 SphereEx-DBPlusEngine 与密文字段产生交互，此时明文字段依然在写入数据。6. 下线明文列当系统经过足够长时间的使用和观察，可将明文字段进行下线。敏感字段只以密文形式存在系统中。

反洗数

1. 新增对应明文字段
2. 修改加密配置，在配置中增加明文字段，此时 queryWithCipher 为 true
3. 开始反洗数
4. 修改 queryWithCipher 为 false
5. 删除加密规则
6. 最后再情况加密字段

重洗数

1. 启动自动换密钥流程
2. 执行重洗数第二阶段
3. 提交重洗数任务

7.4.6 操作指南

数据加密-新业务

1. 设计加密表，确定加密字段，可结合表结构、数据特点及应用情况综合考虑；
2. 在 SphereEx-DBPlusEngine 创建逻辑库，注册存储节点（空）；
3. 创建加密规则，可使用 SphereEx-Console 或 DistSQL 完成；
4. 创建加密表，完成加密表构建；
5. 根据业务实际情况，再选择保留或删除明文字段。

加密洗数-成熟业务

1. 设计加密表，确定加密字段，可结合表结构、数据特点及应用情况综合考虑；
2. 在 SphereEx-DBPlusEngine 创建逻辑库，注册数据源（明文数据）；
3. 使用 DistSQL 创建加密规则；
4. 通过 ENCRYPT TABLE 命令开始洗数；
5. 通过 ALTER ENCRYPT RULE 命令启用加密字段；
6. 根据业务实际情况，再选择保留或删除明文字段。

7.4.7 配置示例

新上线业务加密配置-DistSQL

业务场景分析：新上线业务由于一切从零开始，不存在历史数据清洗问题，所以相对简单。

解决方案说明：选择合适的加密算法，如 AES 后，只需配置逻辑列（面向用户编写 SQL）和密文列（数据表存密文数据）即可，逻辑列和密文列可以相同也可以不同。

SphereEx-DBPlusEngine 只需将 logicColumn 和 cipherColumn，assistedQueryColumn 进行转换，底层数据表不存储明文，只存储了密文，这也是安全审计部分的要求所在。如果用户希望将明文、密文一同存储到数据库，只需添加 plainColumn 配置即可。整体处理流程如下图所示：

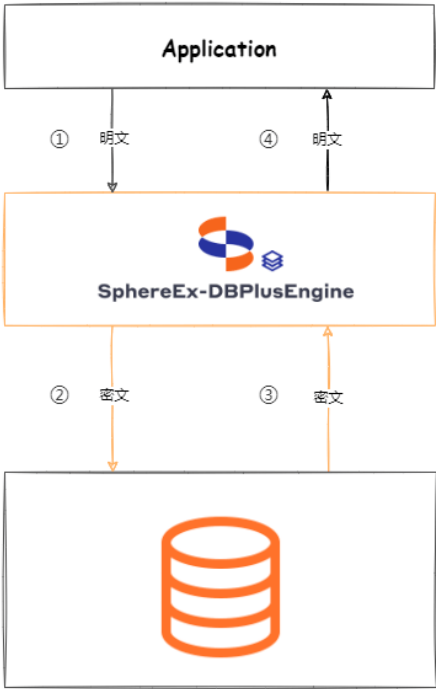


Fig. 16: 整体处理流程图

环境说明

	实例	IP 地址	服务端口	主机名	备注
1	DBPlusEngine 1.2.0	192.168.xx.102	3307	dbplusengine	
2	MySQL 5.7.26	192.168.xx.106	3306	ds_encryption	

拓扑图

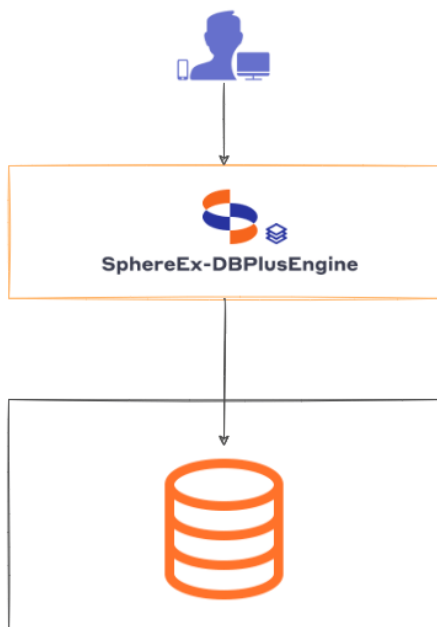


Fig. 17: 拓扑图

配置过程

1. 准备数据源

在 MySQL 中创建名为 ds_encryption 的数据库。

```
# mysql -utest -p -h192.168.xx.106

mysql> CREATE DATABASE ds_encryption;
Query OK, 1 row affected (0.05 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| ds_encryption |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.01 sec)
```


2. 在 SphereEx-DBPlusEngine 中创建名为 testdb 的逻辑库。

```
# mysql -uroot -p -P3307 -h127.0.0.1
```

```
mysql> CREATE DATABASE testdb;
Query OK, 0 rows affected (0.85 sec)
```

```
mysql> SHOW DATABASES;
```

```
+-----+
| schema_name |
+-----+
| mysql       |
| information_schema |
| performance_schema |
| sys        |
| testdb     |
+-----+
5 rows in set (0.00 sec)
```

4. 将数据源注册到 SphereEx-DBPlusEngine 中

```
mysql> USE testdb;
Database changed
```

```
mysql> REGISTER STORAGE UNIT ds_0 (
  URL="jdbc:mysql://192.168.xx.103:3306/mysql_0?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
);
Query OK, 0 rows affected (0.81 sec)
```

5. 在 SphereEx-DBPlusEngine 中创建分片规则

```
mysql> CREATE ENCRYPT RULE t_user (
  COLUMNS(
    (
      NAME = id_card_no,
      PLAIN = id_card_no_plain,
      CIPHER = id_card_no_cipher,
      ENCRYPT_ALGORITHM(
        TYPE(
          NAME = 'AES',
          PROPERTIES('aes-key-value' = '123456abc')
        )
      )
    ),
    (
      NAME = mobile,
      PLAIN = mobile_plain,
      CIPHER = mobile_cipher,
      ENCRYPT_ALGORITHM(
        TYPE(
          NAME = 'AES',
          PROPERTIES('aes-key-value' = '123456abc')
        )
      )
    )
  ),
  QUERY_WITH_CIPHER_COLUMN = true
);
Query OK, 0 rows affected (6.56 sec)
```

```
mysql> SHOW ENCRYPT TABLE RULE t_user\G
***** 1. row *****
table: t_user
```

```

    logic_column: id_card_no
    logic_data_type:
    cipher_column: id_card_no_cipher
    cipher_data_type:
    plain_column: id_card_no_plain
    plain_data_type:
    assisted_query_column:
    assisted_query_data_type:
    like_query_column:
    like_query_data_type:
    encryptor_type: AES
    encryptor_props: aes-key-value=123456abc
    assisted_query_type:
    assisted_query_props:
    like_query_type:
    like_query_props:
query_with_cipher_column: true
***** 2. row *****
    table: t_user
    logic_column: mobile
    logic_data_type:
    cipher_column: mobile_cipher
    cipher_data_type:
    plain_column: mobile_plain
    plain_data_type:
    assisted_query_column:
    assisted_query_data_type:
    like_query_column:
    like_query_data_type:
    encryptor_type: AES
    encryptor_props: aes-key-value=123456abc
    assisted_query_type:
    assisted_query_props:
    like_query_type:
    like_query_props:
query_with_cipher_column: true
2 rows in set (0.00 sec)

```

在以上配置中，对 t_user 表中 mobile 和 id_card_no 字段通过 AES 算法进行加密。

6. 创建表并插入若干条测试记录

```

mysql> CREATE TABLE t_user (
  id INT(8),
  mobile VARCHAR(50),
  id_card_no VARCHAR(50)
);
Query OK, 0 rows affected (1.90 sec)

mysql> INSERT INTO t_user (id, mobile, id_card_no)
VALUES (1, 18236483857, 220605194709308170),
(2, 15686689114, 360222198806088804),
(3, 14523360225, 411601198601098107),
(4, 18143924353, 540228199804231247),
(5, 15523349333, 360924195311103360),
(6, 13261527931, 513229195302236086),
(7, 13921892133, 500108194806107214),
(8, 15993370854, 451322194405305441),
(9, 18044280924, 411329199808285772),
(10, 13983621809, 430204195612042092);
Query OK, 10 rows affected (0.61 sec)

```

7. 确认测试数据插入结果，同时执行 join 的操作进行验证。

```
mysql> SELECT * FROM t_user;
+-----+-----+-----+
| id | mobile | id_card_no |
+-----+-----+-----+
| 1 | 18236483857 | 220605194709308170 |
| 2 | 15686689114 | 360222198806088804 |
| 3 | 14523360225 | 411601198601098107 |
| 4 | 18143924353 | 540228199804231247 |
| 5 | 15523349333 | 360924195311103360 |
| 6 | 13261527931 | 513229195302236086 |
| 7 | 13921892133 | 500108194806107214 |
| 8 | 15993370854 | 451322194405305441 |
| 9 | 18044280924 | 411329199808285772 |
| 10 | 13983621809 | 430204195612042092 |
+-----+-----+-----+
10 rows in set (0.25 sec)

mysql> SELECT * FROM t_user WHERE mobile = '18236483857';
+-----+-----+-----+
| id | mobile | id_card_no |
+-----+-----+-----+
| 1 | 18236483857 | 220605194709308170 |
+-----+-----+-----+
1 row in set (0.50 sec)

mysql>
mysql> SELECT * FROM t_user WHERE mobile = '18236483857'
-> AND id_card_no = '220605194709308170';
+-----+-----+-----+
| id | mobile | id_card_no |
+-----+-----+-----+
| 1 | 18236483857 | 220605194709308170 |
+-----+-----+-----+
1 row in set (0.26 sec)
```

8. 在 MySQL 中进行查看，可见数据库中 mobile 和 id_card_no 字段均存储的是密文数据。

```
mysql -utest -p -h192.168.xx.103

mysql> USE ds_encryption;
Database changed

mysql> SHOW TABLES;
+-----+
| Tables_in_ds_encryption |
+-----+
| t_user |
+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM t_user;
+-----+-----+-----+
| id | mobile_cipher | id_card_no_cipher |
+-----+-----+-----+
| 1 | p31PkI9nIunYdH+AngyNUA== | pQv0JEkM94QzktJdM8UMg/uLrU71G6n6DALdPp9w6L0= |
| 2 | CV8+uYRaWOzcTQnQX3RcwA== | dCF7k4haK0aIV/d7dtwgZlb4lIFLJ913hrPim1+J278= |
| 3 | jnfu7o44KgN/PV1zhiu7jw== | 8iulp3+XTSv2XHGUUHKV0UsLuFx7yEpQVT+47EFfg94= |
| 4 | ZJDrTv/XljdqdG1yp0t95w== | iqU6myMGfgl/XnxCtjhbMrwlauriWu8crxPS6BH2pMk= |
| 5 | FnQMYGnFJaiWmTHeNYzbFA== | KAPrCXoo1svMt5NWe0UaKYZlI1rSEVddHbBJO1jPlqw= |
| 6 | lv2ECfTCgQQksvdPp6k3Ug== | BBBPAuwU+iJluI9d9TA+H81BPnVXBaly1BE3EplN4e8= |
| 7 | z46vpnHCFtkIF2EtntxpHQ== | Bc39nPtyz1ji9Rc8k4f7G9CKfPew23mKFwp8guK7ybg= |
| 8 | p/IJdGcCikhpCu5gVZj4jg== | nnv/kS1i7uHXKncUOuLzE8OWM0nGlGkLokT2dltSaQ= |
| 9 | NvPcQv4w3EqD77+VAX0KCA== | +yeo5LWKNWcekFqYawCKjsctAZqe104DrI7AeZdR/Uk= |
| 10 | xOyg9E0X9lhy9mUx0QyL0A== | U7P1CMcxn6VPHYHPgTAtjHEbb6N6vhGOpdJtVjAdHIA= |
```

```
+-----+-----+-----+-----+
10 rows in set (0.00 sec)
```

9. 删除明文字段

(注：删除明文列是高危且不可逆操作，建议先稳定运行数月后再执行该操作。)

```
--首先在数据库中删除明文字段
--然后在 Proxy 中刷新元数据信息即可
mysql> REFRESH table METADATA;
```

以上为使用 DistSQL 的方式完成了加密表的配置，下面将演示通过 Console 来完成加密的配置。

新上线业务加密配置-Console

环境说明

	实例	IP 地址	服务端口	主机名	备注
1	DBPlusEngine 1.2.0	192.168.xx.102	3307	dbplusengine	
2	Console 1.1.0	192.168.xx.101	8089	console	
3	MySQL 5.7.26	192.168.xx.106	3306	ds_encryption	

拓扑图

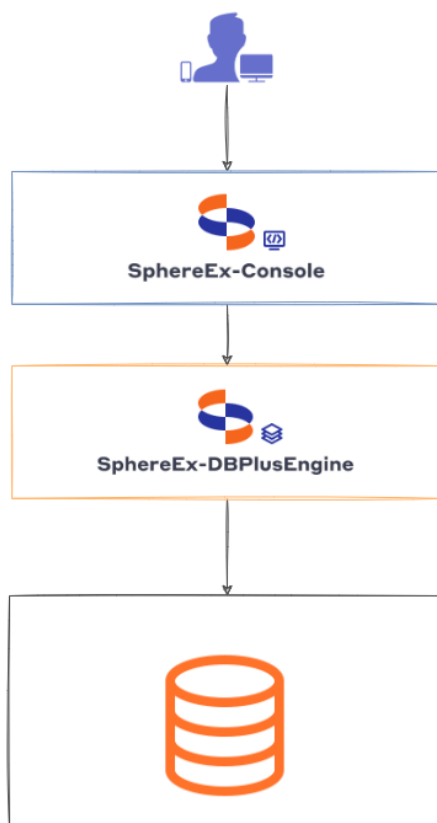


Fig. 18: 拓扑图

配置过程

1. 登陆 SphereEx-Console，在“集群” - “对象” 页中，点击“新建表”按钮。

根据参考示例，输入如下建表语句，点击“确定”。

```
CREATE TABLE t_encrypt(  
  id INT(8),  
  mobile VARCHAR(50),  
  id_card_no VARCHAR(50)  
);
```

2. 添加加密插件

选择创建的数据表：t_encrypt，点击“添加加密”->选择“常规”加密->点击“添加加密插件”，选择“加密算法”，选择“需要加密的列”，勾选“存储明文”，点击“添加”->点击“确定”。

3. 插入测试数据

```
# mysql -uroot -proot -P3307 -h192.168.xx.102
```

```
mysql> use logical001;  
mysql> INSERT INTO t_encrypt (id, mobile_plain, id_card_no)
```

```
-> VALUES (1, 18236483857, 220605194709308170),
-> (2, 15686689114, 360222198806088804),
-> (3, 13983621809, 430204195612042092);
Query OK, 3 rows affected (0.02 sec)
```

4. 确认数据加密结果

登录 SphereEx-DBPlusEngine 查看数据是密文显示。

(说明：如果需要 SphereEx-DBPlusEngine 中显示明文，不勾选“是否使用加密列查询”。)

```
mysql -uroot -proot -P3307 -h192.168.xx.102
```

```
mysql> use logical001;
mysql> SELECT * FROM t_encrypt;
+-----+
| id | mobile          | id_card_no  |
+-----+
| 1 | 6f655ea3b73e32d54ae2d34d672c0c55 | 220605194709308170 |
| 2 | a2193d0e8de4c2c4f166448e22ba97ae | 360222198806088804 |
| 3 | 3cf1fd442630071ca664bbd5e683f7f6 | 430204195612042092 |
+-----+
3 rows in set (0.00 sec)
```

登录 MySQL 中查看数据，确认是密文显示。

```
mysql -uroot -proot -P3306 -h192.168.xx.106
```

```
mysql> use t_encrypt_db;
mysql> select * from t_encrypt;
+-----+
| id | mobile_cipher      | mobile_plain | id_card_no  |
+-----+
| 1 | 6f655ea3b73e32d54ae2d34d672c0c55 | 18236483857 | 220605194709308170 |
| 2 | a2193d0e8de4c2c4f166448e22ba97ae | 15686689114 | 360222198806088804 |
| 3 | 3cf1fd442630071ca664bbd5e683f7f6 | 13983621809 | 430204195612042092 |
+-----+
```

以上是 Console 配置加密表的过程，操作相对更简化。

已上线业务加密配置-DistSQL

对于已经上线运行一段时间的业务系统，数据库里会存在大量明文历史数据，本示例将演示如何对历史项目中的数据进行洗数。

1. 准备 MySQL 环境（明文），模拟已上线数据库

```
mysql> CREATE DATABASE ds_mysql;
Query OK, 1 row affected (0.00 sec)

mysql> USE ds_mysql;
Database changed

mysql> CREATE TABLE t_user (
  id INT(8) PRIMARY KEY,
  mobile VARCHAR(50),
  id_card_no VARCHAR(50)
);
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> INSERT INTO t_user (id, mobile, id_card_no)
VALUES (1, 18236483857, 220605194709308170),
(2, 15686689114, 360222198806088804),
```

```
(3, 14523360225, 411601198601098107),
(4, 18143924353, 540228199804231247),
(5, 15523349333, 360924195311103360),
(6, 13261527931, 513229195302236086),
(7, 13921892133, 500108194806107214),
(8, 15993370854, 451322194405305441),
(9, 18044280924, 411329199808285772),
(10, 13983621809, 430204195612042092);
Query OK, 10 rows affected (0.00 sec)
Records: 10 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT * FROM t_user;
+----+-----+-----+
| id | mobile | id_card_no |
+----+-----+-----+
| 1 | 18236483857 | 220605194709308170 |
| 2 | 15686689114 | 360222198806088804 |
| 3 | 14523360225 | 411601198601098107 |
| 4 | 18143924353 | 540228199804231247 |
| 5 | 15523349333 | 360924195311103360 |
| 6 | 13261527931 | 513229195302236086 |
| 7 | 13921892133 | 500108194806107214 |
| 8 | 15993370854 | 451322194405305441 |
| 9 | 18044280924 | 411329199808285772 |
| 10 | 13983621809 | 430204195612042092 |
+----+-----+-----+
10 rows in set (0.00 sec)
```

此时，已经完成了 MySQL 环境的准备，在 ds_mysql 数据库中存在一张名为 t_user 的明文表。

2. 在 SphereEx-DBPlusEngine 中创建名为 testdb 的逻辑库，并将 MySQL 数据源注册到其中

```
mysql> CREATE DATABASE testdb;
Query OK, 0 rows affected (0.00 sec)

mysql> USE testdb;
Database changed

mysql> REGISTER STORAGE UNIT ds_0 (
  URL="jdbc:mysql://192.168.xx.103:3306/ds_mysql?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
);
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW STORAGE UNITS\G
***** 1. row *****
      name: ds_0
      type: MySQL
      host: 192.168.xx.103
      port: 3306
      db: ds_mysql
connection_timeout_milliseconds: 30000
idle_timeout_milliseconds: 60000
max_lifetime_milliseconds: 2100000
max_pool_size: 50
min_pool_size: 1
read_only: false
other_attributes: {"dataSourceProperties":{"cacheServerConfiguration":"true","elideSetAutoCommits":"true",
"useServerPrepStmts":"true","cachePrepStmts":"true","rewriteBatchedStatements":"true","cacheResultSetMetadata":"false",
"useLocalSessionState":"true","maintainTimeStats":"false","prepStmtCacheSize":"8192","tinyInt1isBit":"false",
"prepStmtCacheSqlLimit":"2048","netTimeoutForStreamingResults":"0","zeroDateTimeBehavior":"round"},
"healthCheckProperties":{},"initializationFailTimeout":1,"validationTimeout":5000,"keepaliveTime":0,"leakDetectionThreshold":0,"registerMbeans":false,"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false}
```

1 row in set (0.00 sec)

```
mysql> SHOW TABLES;
+-----+-----+
| Tables_in_testdb | Table_type |
+-----+-----+
| t_user           | BASE TABLE |
+-----+-----+
1 row in set (0.01 sec)
```

通过如上输出可确认，逻辑库中已经识别到明文表 t_user。

3. 洗数工作前的准备

在开始洗数前，需要在 MySQL 中增加对应的密文字段。这里将增加 2 个密文字段，分别对应 mobile 和 id_card_no。

```
mysql> ALTER TABLE t_user ADD COLUMN mobile_cipher varchar(50) AFTER mobile;
Query OK, 0 rows affected (0.04 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> ALTER TABLE t_user ADD COLUMN id_card_no_cipher varchar(50) AFTER id_card_no;
Query OK, 0 rows affected (0.01 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> DESC t_user;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id         | int       | NO   | PRI | NULL    |      |
| mobile     | varchar(50) | YES  |     | NULL    |      |
| mobile_cipher | varchar(50) | YES  |     | NULL    |      |
| id_card_no | varchar(50) | YES  |     | NULL    |      |
| id_card_no_cipher | varchar(50) | YES  |     | NULL    |      |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM t_user;
+-----+-----+-----+-----+-----+
| id | mobile | mobile_cipher | id_card_no | id_card_no_cipher |
+-----+-----+-----+-----+-----+
| 1 | 18236483857 | NULL | 220605194709308170 | NULL |
| 2 | 15686689114 | NULL | 360222198806088804 | NULL |
| 3 | 14523360225 | NULL | 411601198601098107 | NULL |
| 4 | 18143924353 | NULL | 540228199804231247 | NULL |
| 5 | 15523349333 | NULL | 360924195311103360 | NULL |
| 6 | 13261527931 | NULL | 513229195302236086 | NULL |
| 7 | 13921892133 | NULL | 500108194806107214 | NULL |
| 8 | 15993370854 | NULL | 451322194405305441 | NULL |
| 9 | 18044280924 | NULL | 411329199808285772 | NULL |
| 10 | 13983621809 | NULL | 430204195612042092 | NULL |
+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)
```

当前密文字段内容为空，稍后将会通过洗数操作在线转换密文数据。

4. 在 SphereEx-DBPlusEngine 中配置加密规则

```
CREATE ENCRYPT RULE t_user (
  COLUMNS(
    (
      NAME = id_card_no,
      PLAIN = id_card_no,
      CIPHER = id_card_no_cipher,
      ENCRYPT_ALGORITHM(
        TYPE(
```



```

    NAME = 'AES',
    PROPERTIES('aes-key-value' = '123456abc')
  )
),
(
  NAME = mobile,
  PLAIN = mobile,
  CIPHER = mobile_cipher,
  ENCRYPT_ALGORITHM(
    TYPE(
      NAME = 'AES',
      PROPERTIES('aes-key-value' = '123456abc')
    )
  )
),
QUERY_WITH_CIPHER_COLUMN = false
);
Query OK, 0 rows affected (0.05 sec)

```

-- NAME: 为业务中字段名称, 基于应用免改造的出发点, 可设置与明文字段名称一致
 -- PLAIN: 明文字段名称
 -- CIPHER: 密文字段名称
 -- ENCRYPT_ALGORITHM: 加密算法相关配置
 -- QUERY_WITH_CIPHER_COLUMN: 是否启用密文字段, 当前密文字段为空, 因此需要设置为 **false**

```

mysql> SHOW ENCRYPT TABLE RULE t_user\G
***** 1. row *****
    table: t_user
    logic_column: id_card_no
    logic_data_type:
    cipher_column: id_card_no_cipher
    cipher_data_type:
    plain_column: id_card_no
    plain_data_type:
    assisted_query_column:
    assisted_query_data_type:
    like_query_column:
    like_query_data_type:
    encryptor_type: AES
    encryptor_props: aes-key-value=123456abc
    assisted_query_type:
    assisted_query_props:
    like_query_type:
    like_query_props:
    query_with_cipher_column: false
***** 2. row *****
    table: t_user
    logic_column: mobile
    logic_data_type:
    cipher_column: mobile_cipher
    cipher_data_type:
    plain_column: mobile
    plain_data_type:
    assisted_query_column:
    assisted_query_data_type:
    like_query_column:
    like_query_data_type:
    encryptor_type: AES
    encryptor_props: aes-key-value=123456abc
    assisted_query_type:
    assisted_query_props:

```

```
like_query_type:
like_query_props:
query_with_cipher_column: false
2 rows in set (0.00 sec)
```

```
mysql> PREVIEW SELECT * FROM t_user;
```

```
+-----+-----+
| data_source_name | actual_sql |
+-----+-----+
| ds_0             | SELECT `t_user`.`id`, `t_user`.`mobile` AS `mobile`, `t_user`.`id_card_no` AS `id_card_no` FROM t_user |
+-----+-----+
1 row in set (0.00 sec)
```

通过 PREVIEW 结果来看，当前是通过明文字段来响应请求，与预期一致。

5. 开始洗数在配置好加密字段及加密规则后，可开始洗数。

```
mysql> ENCRYPT TABLE t_user;
Query OK, 0 rows affected (0.05 sec)
```

```
mysql> SHOW ENCRYPTING LIST;
```

```
+-----+-----+-----+-----+-----+-----+
| id          | tables | job_item_count | active | create_time   | stop_time |
+-----+-----+-----+-----+-----+-----+
| j5201e2af8bea809d29f6ab4b3ac4c99b5789 | t_user | 1              | true  | 2023-02-05 13:56:32 | NULL      |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SHOW ENCRYPTING STATUS j5201e2af8bea809d29f6ab4b3ac4c99b5789;
```

```
+-----+-----+-----+-----+-----+-----+
| item | data_source | status | active | encrypting_finished_percentage | error_message |
+-----+-----+-----+-----+-----+-----+
| 0    | testdb     | EXECUTE_INVENTORY_TASK | true  | 100                          |               |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.03 sec)
```

如上所示，加密完成进度为 100%，可登陆 MySQL 查看真实表的数据。此时 MySQL 中的数据如下：

```
mysql> SELECT * FROM t_user;
```

```
+-----+-----+-----+-----+-----+
| id | mobile | mobile_cipher | id_card_no | id_card_no_cipher |
+-----+-----+-----+-----+-----+
| 1 | 18236483857 | p31Pkl9nlunYdH+AngyNUA== | 220605194709308170 | pQv0JEkM94QzktJdM8UMg/
uLrU71G6n6DALdPp9w6LO= |
| 2 | 15686689114 | CV8+uYRaWOzcTQnQX3RcwA== | 360222198806088804 | dCF7k4haK0aIV/d7dtwzgzlb4lIFlJ913hrPim1+J278= |
| 3 | 14523360225 | jnfu7o44KgN/PV1zhiu7jw== | 411601198601098107 | 8iulp3+XTSv2XHGUUHKV0UsLuFx7yEpQVT+47EFfg94= |
| 4 | 18143924353 | ZJDrTv/XljdqdG1yp0t95w== | 540228199804231247 | iqU6myMGfgl/XnxCtjhbMrwlauriWu8crxPS6BH2pMk= |
| 5 | 15523349333 | FnQMYGnFjaiWmTHeNYzbFA== | 360924195311103360 | KAPrCXoo1svMt5NWe0UaKYZlI1rSEVddHbBJO1jPlqw= |
| 6 | 13261527931 | lv2ECfTCgQQksvdPp6k3Ug== | 513229195302236086 | BBBPAuwU+iJluI9d9TA+H81BPnVXBaly1BE3EplN4e8= |
| 7 | 13921892133 | z46vpnHCFTklF2EtntxpHQ== | 500108194806107214 | Bc39nPtzy1ji9Rc8k4f7G9CKfPew23mKFwp8guK7ybg= |
| 8 | 15993370854 | p/IJdGcCikhpCu5gVZj4jg== | 451322194405305441 | nnv/kS1i7uHXKncUOuLzE8OWM0nGlcGkLokT2dltSaQ= |
| 9 | 18044280924 | NvPcQv4w3EqD77+VAX0KCA== | 411329199808285772 | +yeo5LWKNWcekFqYawCKjsctAZqe104DrI7AeZdR/Uk= |
| 10 | 13983621809 | xOyg9E0X9lhy9mUx0QyLOA== | 430204195612042092 | U7P1CMcxn6VPHYHPgTAtjHEbb6N6vhGOpdJtVjAdHIA= |
+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)
```

表中明文密文同时存在，此时为业务提供响应的是明文字段。

6. 启用密文字段

当明文密文并行运行一段时间后，可结合实际情况启用密文字段。

```

ALTER ENCRYPT RULE t_user (
  COLUMNS(
    (
      NAME = id_card_no,
      PLAIN = id_card_no,
      CIPHER = id_card_no_cipher,
      ENCRYPT_ALGORITHM(
        TYPE(
          NAME = 'AES',
          PROPERTIES('aes-key-value' = '123456abc')
        )
      )
    ),
    (
      NAME = mobile,
      PLAIN = mobile,
      CIPHER = mobile_cipher,
      ENCRYPT_ALGORITHM(
        TYPE(
          NAME = 'AES',
          PROPERTIES('aes-key-value' = '123456abc')
        )
      )
    )
  ),
  QUERY_WITH_CIPHER_COLUMN = true
);
Query OK, 0 rows affected (0.02 sec)

```

```

mysql> SHOW ENCRYPT TABLE RULE t_user\G
***** 1. row *****
    table: t_user
    logic_column: id_card_no
    logic_data_type:
    cipher_column: id_card_no_cipher
    cipher_data_type:
    plain_column: id_card_no
    plain_data_type:
    assisted_query_column:
    assisted_query_data_type:
    like_query_column:
    like_query_data_type:
    encryptor_type: AES
    encryptor_props: aes-key-value=123456abc
    assisted_query_type:
    assisted_query_props:
    like_query_type:
    like_query_props:
    query_with_cipher_column: true
***** 2. row *****
    table: t_user
    logic_column: mobile
    logic_data_type:
    cipher_column: mobile_cipher
    cipher_data_type:
    plain_column: mobile
    plain_data_type:
    assisted_query_column:
    assisted_query_data_type:
    like_query_column:
    like_query_data_type:
    encryptor_type: AES
    encryptor_props: aes-key-value=123456abc

```

```

assisted_query_type:
assisted_query_props:
  like_query_type:
  like_query_props:
query_with_cipher_column: true
2 rows in set (0.01 sec)

mysql> PREVIEW SELECT * FROM t_user;
+-----+-----+-----+
| data_source_name | actual_sql |
+-----+-----+-----+
| ds_0 | SELECT `t_user`.`id`, `t_user`.`mobile_cipher` AS `mobile`, `t_user`.`id_card_no_cipher` AS `id_card_no`
FROM t_user |
+-----+-----+-----+
1 row in set (0.00 sec)

```

此时，通过 PREVIEW 输出所示，业务已开始使用了密文字段。洗数操作完成，明文字段可根据实际的使用需求进行变更调整。

模糊查询

SphereEx-DBPlusEngine 支持加密列 like 查询，以下为验证步骤。

1. 配置 MySQL 配置文件

为了避免 MySQL 存储中文乱码导致 LIKE 功能错乱，需要提前修改 my.cnf，确保进行了如下配置：

```

[client]
default-character-set=utf8mb4
[mysql]
default-character-set=utf8mb4
[mysqld]
port=3306
character-set-server=utf8mb4
character-set-filesystem=utf8mb4
collation-server=utf8mb4_general_ci
init-connect='SET NAMES utf8mb4'

```

2. 创建逻辑库，并添加资源

```

CREATE DATABASE encrypt_db;

USE encrypt_db;

REGISTER STORAGE UNIT ds_0 (
  HOST="127.0.0.1",
  PORT=3306,
  DB="demo_ds_0",
  USER="root",
  PASSWORD="123456"
), ds_1 (
  HOST="127.0.0.1",
  PORT=3306,
  DB="demo_ds_1",
  USER="root",
  PASSWORD="123456"
);

```

3. 创建加密规则

```

CREATE ENCRYPT KEY MANAGER aws_kms_key_manager (TYPE(NAME='SphereEx:AWS_KMS', PROPERTIES('access-key'=
'AKIAXRZPPZNGXS6VJUGY','secret-key'='RUZydkmkWvGViqkB+N0izEp5rDKavKkcqm/fhTq','aws-region'='ap-northeast-1',
'secret-name'='dbplus-engine/encrypt-key')));

CREATE ENCRYPT RULE t_user (
  COLUMNS(
    (
      NAME=user_name,PLAIN=user_name_plain,CIPHER=user_name_cipher,LIKE_QUERY_COLUMN=user_name_like,
      ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))),
      LIKE_QUERY_ALGORITHM(TYPE(NAME='SphereEx:CHAR_TRANSFORM_LIKE',PROPERTIES('key-manager'="aws_
kms_key_manager"))))
    ),
    (
      NAME=password,CIPHER=password_cipher,
      ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))))
    ),
    (
      NAME=email,CIPHER=email_cipher,
      ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))))
    ),
    (
      NAME=telephone,PLAIN=telephone_plain,CIPHER=telephone_cipher,LIKE_QUERY_COLUMN=telephone_like,
      ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))),
      LIKE_QUERY_ALGORITHM(TYPE(NAME='SphereEx:CHAR_TRANSFORM_LIKE',PROPERTIES('key-manager'="aws_
kms_key_manager"))))
    )
  ), QUERY_WITH_CIPHER_COLUMN = TRUE
), t_merchant (
  COLUMNS(
    (
      NAME=business_code,PLAIN=business_code_plain,CIPHER=business_code_cipher,LIKE_QUERY_COLUMN=business_
code_like,
      ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))),
      LIKE_QUERY_ALGORITHM(TYPE(NAME='SphereEx:CHAR_TRANSFORM_LIKE',PROPERTIES('key-manager'="aws_
kms_key_manager"))))
    ),
    (
      NAME=telephone,PLAIN=telephone_plain,CIPHER=telephone_cipher,LIKE_QUERY_COLUMN=telephone_like,
      ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))),
      LIKE_QUERY_ALGORITHM(TYPE(NAME='SphereEx:CHAR_TRANSFORM_LIKE',PROPERTIES('key-manager'="aws_
kms_key_manager"))))
    )
  ), QUERY_WITH_CIPHER_COLUMN = TRUE
);

```

4. 创建逻辑库，并创建测试表，插入若干条测试记录

```
USE encrypt_db;
```

```
DROP TABLE IF EXISTS t_merchant;
```

```

CREATE TABLE t_merchant (
  merchant_id INT PRIMARY KEY,
  country_id SMALLINT NOT NULL,
  merchant_name VARCHAR(50) NOT NULL,
  business_code VARCHAR(50) NOT NULL,
  telephone CHAR(50) NOT NULL,
  creation_date DATE NOT NULL);

```

```
INSERT INTO t_merchant .....
```

```

CREATE TABLE t_user (
  user_id INT PRIMARY KEY,

```

```
user_name VARCHAR(50) NOT NULL,
password VARCHAR(50) NOT NULL,
email VARCHAR(50) NOT NULL,
telephone CHAR(50) NOT NULL,
creation_date DATE NOT NULL);
```

```
INSERT INTO t_user .....
```

5. 测试数字 LIKE 查询

```
-- 测试 % 前缀匹配
```

```
SELECT * FROM t_merchant WHERE business_code LIKE '%1';
```

merchant_id	country_id	merchant_name	business_code	telephone	creation_date
1	86	test101	86000001	86100000001	2017-08-08
11	1	test111	01000011	01100000011	2017-08-08

```
2 rows in set (0.01 sec)
```

```
-- 测试 % 后缀匹配
```

```
SELECT * FROM t_merchant WHERE business_code LIKE '86%';
```

merchant_id	country_id	merchant_name	business_code	telephone	creation_date
1	86	test101	86000001	86100000001	2017-08-08
2	86	test102	86000002	86100000002	2017-08-08
3	86	test103	86000003	86100000003	2017-08-08
4	86	test104	86000004	86100000004	2017-08-08
5	86	test105	86000005	86100000005	2017-08-08
6	86	test106	86000006	86100000006	2017-08-08
7	86	test107	86000007	86100000007	2017-08-08
8	86	test108	86000008	86100000008	2017-08-08
9	86	test109	86000009	86100000009	2017-08-08
10	86	test110	86000010	86100000010	2017-08-08

```
10 rows in set (0.01 sec)
```

6. 测试字母 LIKE 查询

```
-- 测试 % 前缀匹配
```

```
SELECT * FROM t_user WHERE user_name LIKE '%san';
```

user_id	user_name	password	email	telephone	creation_date
10	zhangsan	111111	zhangsan@gmail.com	12345678900	2017-08-08
19	zhousan	123456	zhousan@gmail.com	12345678909	2017-08-08

```
2 rows in set (0.01 sec)
```

```
-- 测试 % 后缀匹配
```

```
SELECT * FROM t_user WHERE user_name LIKE 'j%';
```

user_id	user_name	password	email	telephone	creation_date
22	jerry	456789	jerry@gmail.com	12345678912	2017-08-08
23	james	567890	james@gmail.com	12345678913	2017-08-08
27	jack	222333	jack@gmail.com	12345678917	2017-08-08
28	jordan	333444	jordan@gmail.com	12345678918	2017-08-08
29	julie	444555	julie@gmail.com	12345678919	2017-08-08

```
5 rows in set (0.01 sec)
```

7. 测试汉字 LIKE 查询

```
- 测试 % 前缀匹配
SELECT * FROM t_user WHERE user_name LIKE '%靖';
+-----+-----+-----+-----+-----+
| user_id | user_name | password | email          | telephone | creation_date |
+-----+-----+-----+-----+-----+
| 34 | 郭靖 | 123123 | guojing@gmail.com | 18012512345 | 2017-08-08 |
+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)

-- 测试 % 后缀匹配
SELECT * FROM t_user WHERE user_name LIKE '王%';
+-----+-----+-----+-----+-----+
| user_id | user_name | password | email          | telephone | creation_date |
+-----+-----+-----+-----+-----+
| 35 | 王之涣 | 234234 | wangzhihuan@gmail.com | 18012512345 | 2017-08-08 |
| 36 | 王安石 | 345345 | jeff@gmail.com | 18012512345 | 2017-08-08 |
| 37 | 王羲之 | 345345 | wangxizhi@gmail.com | 18012512345 | 2017-08-08 |
| 38 | 王莽 | 456456 | wangmang@gmail.com | 18012512345 | 2017-08-08 |
| 39 | 王勃 | 567567 | wangbo@gmail.com | 18012512345 | 2017-08-08 |
+-----+-----+-----+-----+-----+
5 rows in set (0.01 sec)
```

反洗数

1. 环境准备

快速构建环境，创建物理库和逻辑库。

```
-- MySQL
mysql> create database demo_ds_0;

-- DBPlusEngine
mysql> create database encrypt_db;

mysql> ADD RESOURCE ds_0 (
  URL="jdbc:mysql://127.0.0.1:3306/demo_ds_0?serverTimezone=UTC&useSSL=false",
  USER="root",
  PASSWORD="123456",
  PROPERTIES("maximumPoolSize"="50","idleTimeout"="60000")
);

mysql> CREATE ENCRYPT RULE t_encrypt (
  COLUMNS(
    (NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))),
    (NAME=order_id,PLAIN=order_id,CIPHER=order_cipher,TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))),
    QUERY_WITH_CIPHER_COLUMN=true);

mysql> CREATE TABLE `t_encrypt` (
  `id` int NOT NULL,
  `user_id` varchar(200) DEFAULT NULL,
  `order_id` varchar(200) DEFAULT NULL,
  PRIMARY KEY (`id`)
);

mysql> insert into t_encrypt values (1,123,123),(2,234,234),(3,345,345);
```

将明文字段更新数据为空，为反洗数做环境准备。

```
mysql> update t_encrypt set user_plain = null, order_id = null;

mysql> select * from t_encrypt;
+-----+-----+-----+-----+-----+
| user_id | user_name | password | email          | telephone | creation_date |
+-----+-----+-----+-----+-----+
| 34 | 郭靖 | 123123 | guojing@gmail.com | 18012512345 | 2017-08-08 |
+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

```

| id | user_cipher | user_plain | order_cipher | order_id |
+---+-----+-----+-----+
| 1 | DZEH799l6UjthceKuCCKlw== | NULL | DZEH799l6UjthceKuCCKlw== | NULL |
| 2 | Q3YzQboxcl5L5DSx9bSp9Q== | NULL | Q3YzQboxcl5L5DSx9bSp9Q== | NULL |
| 3 | pJOAtTsHiqrpPRv/cv8MnA== | NULL | pJOAtTsHiqrpPRv/cv8MnA== | NULL |
+---+-----+-----+-----+
3 rows in set (0.01 sec)

```

2. 创建反洗钱配置

```

mysql> create DECRYPTING PROCESS CONFIGURATION(
READ(
  WORKER_THREAD=40,
  BATCH_SIZE=1000,
  SHARDING_SIZE=1000000
),
WRITE(
  WORKER_THREAD=40,
  BATCH_SIZE=1000
),
STREAM_CHANNEL(TYPE(NAME="MEMORY", PROPERTIES("block-queue-size"=10000)))
);

mysql> SHOW DECRYPTING PROCESS CONFIGURATION;

```

3. 开始反洗钱

```

mysql> DECRYPT table t_encrypt;
Query OK, 0 rows affected (0.19 sec)

mysql> SHOW DECRYPTING LIST;
+---+-----+-----+-----+-----+-----+
| id | tables | job_split_count | active | create_time | stop_time |
+---+-----+-----+-----+-----+-----+
| j53014fb26631db0f7149741422d86cc5dfe6 | t_encrypt | 1 | true | 2022-10-20 19:56:20 | NULL |
+---+-----+-----+-----+-----+-----+
1 row in set (0.06 sec)

mysql> SHOW DECRYPTING STATUS j53014fb26631db0f7149741422d86cc5dfe6;
+---+-----+-----+-----+-----+-----+
| item | data_source | status | active | decrypting_finished_percentage | error_message |
+---+-----+-----+-----+-----+-----+
| 0 | encrypt_db | EXECUTE_INVENTORY_TASK | true | 100 | |
+---+-----+-----+-----+-----+-----+
1 row in set (0.02 sec)

-- 提交反洗钱任务

mysql> COMMIT DECRYPTING j53014fb26631db0f7149741422d86cc5dfe6;
Query OK, 0 rows affected (0.21 sec)

```

4. 在 MySQL 数据库中验证数据

```

mysql> select * from t_encrypt;
+---+-----+-----+-----+
| id | user_cipher | user_plain | order_cipher | order_id |
+---+-----+-----+-----+
| 1 | DZEH799l6UjthceKuCCKlw== | 123 | DZEH799l6UjthceKuCCKlw== | 123 |
| 2 | Q3YzQboxcl5L5DSx9bSp9Q== | 234 | Q3YzQboxcl5L5DSx9bSp9Q== | 234 |
| 3 | pJOAtTsHiqrpPRv/cv8MnA== | 345 | pJOAtTsHiqrpPRv/cv8MnA== | 345 |
+---+-----+-----+-----+
3 rows in set (0.00 sec)

```

操作至此，反洗钱操作已完成。

重洗数

1. 准备工作，在 MySQL 中创建名为 demo_ds_0 的数据库

```
mysql> create database demo_ds_0;
```

2. 在 SphereEx-DBPlusEngine 中创建逻辑库，注册数据源

```
mysql> create database encrypt_db;
```

```
mysql> use encrypt_db
```

```
mysql> REGISTER STORAGE UNIT ds_0 (
  URL="jdbc:mysql://127.0.0.1:3306/demo_ds_0?serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=True",
  USER="root",
  PASSWORD="123456",
  PROPERTIES("maximumPoolSize"="100","idleTimeout"="30000")
)
```

3. 创建加密规则，建标并插入数据

```
CREATE ENCRYPT RULE t_encrypt (
  COLUMNS(
    (NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,ASSISTED_QUERY_COLUMN=user_assisted,LIKE_QUERY_COLUMN=user_
    like,ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'='123456abc'))),ASSISTED_QUERY_
    ALGORITHM(TYPE(NAME='MD5')), LIKE_QUERY_ALGORITHM(TYPE(NAME='CHAR_DIGEST_LIKE'))),
    (NAME=order_id,PLAIN=order_plain,CIPHER=order_cipher, ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-
    value'='654321abc'))))
  ), QUERY_WITH_CIPHER_COLUMN=true);
```

```
mysql> create table t_encrypt (
  id int primary key,
  user_id varchar(200),
  rder_id varchar(200),
  name varchar(200));
```

```
mysql> insert into t_encrypt values
(1,'a','aa','aaa'),
(2,'b','bb','bbb'),
(3,'c','cc','ccc');
```

4. 在 MySQL 中清除明文数据

```
mysql> update t_encrypt set user_plain = null,order_plain =null;
```

```
mysql> select * from t_encrypt;
```

id	user_cipher	user_assisted	user_like	user_plain	order_cipher	order_plain	name
1	FrpvlLvZyXewBRRnbeNnbA==	0cc175b9c0f1b6a831c399e269772661	`		NULL	7xINTg7CqT86N1xDAHxg==	NULL
2	XWj53fUjwGnpP/LDiJWvow==	92eb5ffee6ae2fec3ad71c777531578f	a		NULL	oTVrU2sLf+HWvYn9FJ04Zw==	NULL
3	MglbJfLRYx339IXZ3VHDVw==	4a8a08f09d37b73795649038408b5f33	d		NULL	zsz41oPgr06KG3WWrYixlg==	NULL

3 rows in set (0.00 sec)

5. 更换密钥

```
`` `sql
mysql> REENCRYPT TABLE WITH RULE t_encrypt (
  COLUMNS(
    (NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,ASSISTED_QUERY_COLUMN=user_assisted,LIKE_QUERY_COLUMN=user_
    like,ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'='665544aaa'))),ASSISTED_QUERY_
    ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'='523234abc'))), LIKE_QUERY_ALGORITHM(TYPE(NAME='CHAR_
    DIGEST_LIKE'))),
```

```
(NAME=order_id, PLAIN=order_plain, CIPHER=order_cipher, ENCRYPT_ALGORITHM(TYPE(NAME='AES', PROPERTIES('aes-key-value='445533ccc'))))
), QUERY_WITH_CIPHER_COLUMN=true);
```

■ 查看当前重洗数任务

```
mysql> SHOW REENCRYPTING LIST;
```

```
mysql> SHOW REENCRYPTING LIST;
```

id	tables	job_item_count	active	create_time	stop_time
j54014fb26631db0f7149741422d86cc5dfe6	t_encrypt	1	true	2022-12-23 18:08:19	NULL

1 row in set (0.21 sec)

■ 查看重洗数任务状态

```
-- 当发现任务状态到达 SUSPEND 证明第一阶段任务结束（完成了反洗数以及切换到明文查询）
```

```
mysql> SHOW REENCRYPTING STATUS j54014fb26631db0f7149741422d86cc5dfe6;
```

item	data_source	status	active	encrypting_finished_percentage	error_message
0	encrypt_db	SUSPEND	true	50	

1 row in set (0.01 sec)

■ 查看规则状态

```
SHOW ENCRYPT RULES;
```

```
-- 可以看到规则采用了明文查询
```

```
mysql> SHOW ENCRYPT RULES\G
```

```
***** 1. row *****
    table: t_encrypt
    logic_column: user_id
    logic_data_type:
    cipher_column: user_cipher
    cipher_data_type:
    plain_column: user_plain
    plain_data_type:
    assisted_query_column: user_assisted
    assisted_query_data_type:
    like_query_column: user_like
    like_query_data_type:
    encryptor_type: AES
    encryptor_props: aes-key-value=123456abc
    assisted_query_type: MD5
    assisted_query_props:
    like_query_type: CHAR_DIGEST_LIKE
    like_query_props:
    query_with_cipher_column: false
***** 2. row *****
    table: t_encrypt
    logic_column: order_id
    logic_data_type:
    cipher_column: order_cipher
    cipher_data_type:
    plain_column: order_plain
    plain_data_type:
    assisted_query_column:
    assisted_query_data_type:
    like_query_column:
    like_query_data_type:
    encryptor_type: AES
```

```

    encryptor_props: aes-key-value=654321abc
    assisted_query_type:
    assisted_query_props:
        like_query_type:
        like_query_props:
    query_with_cipher_column: false
2 rows in set (0.00 sec)

```

6. 确认明文数据

```

mysql> select * from t_encrypt;
+----+-----+-----+-----+-----+-----+-----+
| id | user_cipher | user_assisted | user_like | user_plain | order_cipher | order_plain | name |
+----+-----+-----+-----+-----+-----+-----+
| 1 | FrpvlLvZyXewBRRnbeNnbA== | 0cc175b9c0f1b6a831c399e269772661 | ` | a | 7xINTg7CqT86N1xDAHxgxg== | aa | aaa |
| 2 | XWj53fUjwGnpP/LDiJWvow== | 92eb5ffee6ae2fec3ad71c777531578f | a | b | oTVrU2sLf+HWvYn9FJ04Zw== | bb | bbb |
| 3 | MglbJfLRYx339IXZ3VHDVw== | 4a8a08f09d37b73795649038408b5f33 | d | c | zsz41oPgr06KG3WWrYixlg== | cc | ccc |
+----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)

```

7. 重洗数第二阶段

```
START REENCRYPTING j54014fb26631db0f7149741422d86cc5dfe6;
```

■ 查看重洗数任务状态

-- 当发现任务状态到达 **FINISHED** 证明第二阶段任务结束（完成了换密钥，清理加密数据，洗数，切换至密文查询）

```

mysql> SHOW REENCRYPTING STATUS j54014fb26631db0f7149741422d86cc5dfe6;
+----+-----+-----+-----+-----+-----+
| item | data_source | status | active | encrypting_finished_percentage | error_message |
+----+-----+-----+-----+-----+-----+
| 0 | encrypt_db | FINISHED | true | 100 | |
+----+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

■ 查看当前规则

-- 发现密钥已经替换，并且使用加密列查询

```

mysql> show encrypt rules\G
***** 1. row *****
    table: t_encrypt
    logic_column: user_id
    logic_data_type:
    cipher_column: user_cipher
    cipher_data_type:
    plain_column: user_plain
    plain_data_type:
    assisted_query_column: user_assisted
    assisted_query_data_type:
    like_query_column: user_like
    like_query_data_type:
    encryptor_type: AES
    encryptor_props: aes-key-value=665544aaa
    assisted_query_type: AES
    assisted_query_props: aes-key-value=523234abc
    like_query_type: CHAR_DIGEST_LIKE
    like_query_props:
    query_with_cipher_column: true
***** 2. row *****
    table: t_encrypt

```

```

logic_column: order_id
logic_data_type:
cipher_column: order_cipher
cipher_data_type:
plain_column: order_plain
plain_data_type:
assisted_query_column:
assisted_query_data_type:
like_query_column:
like_query_data_type:
  encryptor_type: AES
  encryptor_props: aes-key-value=445533ccc
assisted_query_type:
assisted_query_props:
  like_query_type:
  like_query_props:
query_with_cipher_column: true
2 rows in set (0.01 sec)

```

8. 在 MySQL 中确认数据替换情况

```

mysql> select * from t_encrypt;
+----+-----+-----+-----+-----+-----+-----+
| id | user_cipher | user_assisted | user_like | user_plain | order_cipher | order_plain | name |
+----+-----+-----+-----+-----+-----+-----+
| 1 | Ktmk9NUTHYXS+OApRCg+A== | FMAjB5mOAmqotmfX7lQwhQ== | ` | a | 5owposzhT/Zpk70GAQCBKw== | aa | aaa |
| 2 | 9MbqQaWlea72vZomWLoUMQ== | qiZJCKlQF7+/LzjoU9KYTQ== | a | b | aHp/Hywm/TkEMgnvjbOLOw== | bb | bbb |
| 3 | TLXTYnAbSCstVOz0Cv3iNg== | 3fGlHRInkQn+FclzMUWS8Q== | d | c | 4PhsQk1WvNg1gbD5KfwPpw== | cc | ccc |
+----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

9. 提交重洗数任务

```
mysql> COMMIT REENCRYPTING j54014fb26631db0f7149741422d86cc5dfe6;
```

7.4.8 FAQ

1. 洗数操作是否可以在线进行？对性能是否有影响？

可在线进行，SphereEx-DBPlusEngine 提供了线程配置方法，可尽量减少对业务的性能影响。同时，建议洗数据操作尽可能的在停机状态下操作，性能会更好。

2. 如果洗数操作意外中断，是否可以继续洗数？

可以。可重新执行洗数命令继续操作。

7.5 数据脱敏

随着《网络安全法》的颁布施行，对个人隐私数据的保护已经上升到法律层面。传统的应用系统普遍缺少对个人隐私数据的保护措施。数据脱敏，可实现在不需要对生产数据库中的数据进行任何改变的情况下，依据用户定义的脱敏规则，对生产数据库返回的数据进行专门的加密、遮盖和替换，确保生产环境的敏感数据能够得到保护。

7.5.1 挑战和目标

在真实业务场景中，相关业务开发团队则往往需要根据脱敏需求，自行实现并维护一套脱敏功能，而脱敏功能往往是耦合在各个业务逻辑中，不同的业务系统难以复用，而当脱敏场景发生改变时，自行维护的脱敏功能往往又面临着重构或修改的风险。

根据业界对脱敏的需求及业务改造痛点，提供了一套完整、安全、透明化、低改造成本的数据脱敏整合解决方案，是 SphereEx-DBPlusEngine 数据脱敏模块的主要设计目标。

7.5.2 应用场景

不论是快速上线新业务，还是已经上线的成熟业务，都可以接入 SphereEx-DBPlusEngine 脱敏功能，快速地完成脱敏规则的配置。客户无需开发耦合于业务系统的脱敏功能，不需要改动任何业务逻辑和 SQL 就能够透明化地使用脱敏功能。

7.5.3 使用限制

脱敏列只支持字符串类型，不支持其他非字符串类型。

7.5.4 处理流程

SphereEx-DBPlusEngine 通过对用户查询的 SQL 进行解析，并依据用户提供的脱敏规则对 SQL 执行结果进行装饰，从而实现原文数据进行脱敏。

7.5.5 整体架构

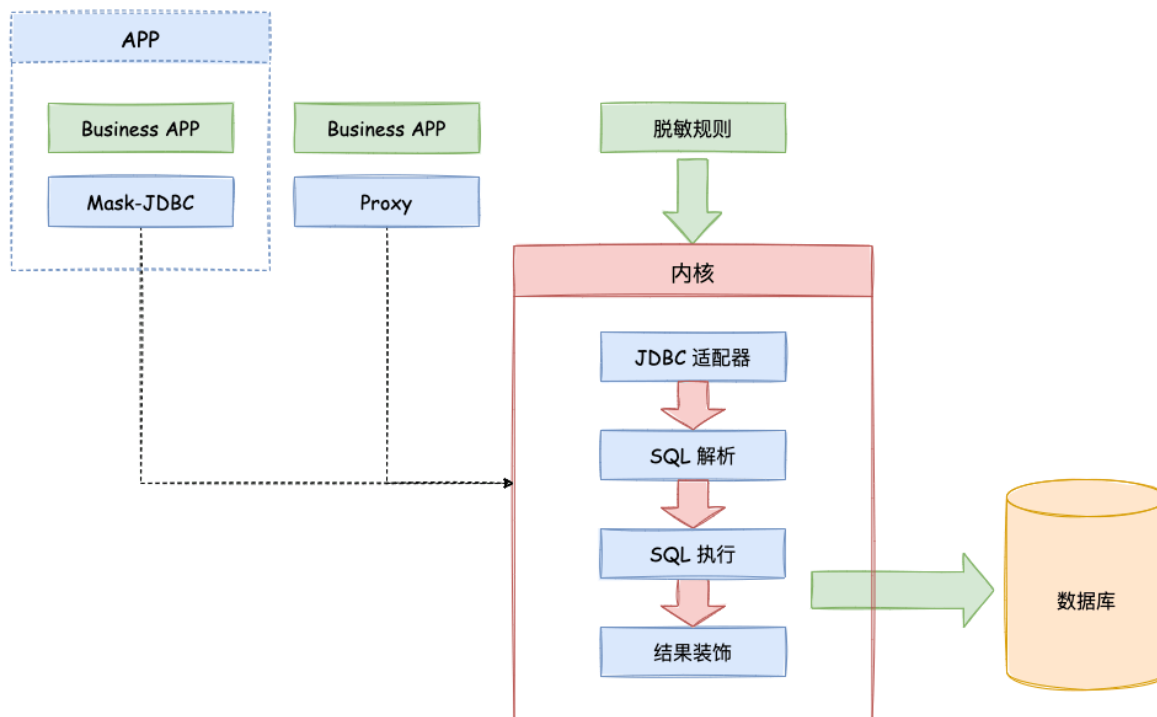


Fig. 19: 数据脱敏架构

脱敏模块将用户发起的 SQL 进行拦截，并通过 SQL 语法解析器进行解析、执行，再依据用户传入的脱敏规则，找出需要脱敏的字段和所使用的脱敏算法对查询结果进行装饰，返回给客户端。

7.5.6 脱敏规则

在详解整套流程之前，我们需要先了解下脱敏规则与配置，这是认识整套流程的基础。脱敏配置主要分为三部分：数据源配置，脱敏算法配置，脱敏表配置，脱敏规则创建即可生效。

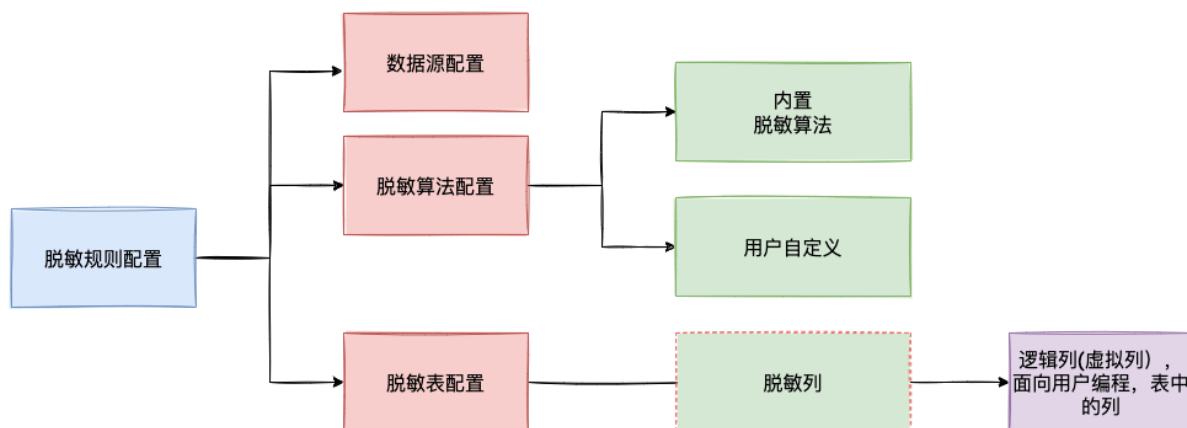


Fig. 20: 数据脱敏规则

数据源配置：指数据源配置。

脱敏算法配置：指使用什么脱敏算法。目前 SphereEx-DBPlusEngine 内置了多种脱敏算法：MD5、KEEP_FIRST_N_LAST_M、KEEP_FROM_X_TO_Y、MASK_FIRST_N_LAST_M、MASK_FROM_X_TO_Y、MASK_BEFORE_SPECIAL_CHARS、MASK_AFTER_SPECIAL_CHARS、PERSONAL_IDENTITY_NUMBER_RANDOM_REPLACE、MILITARY_IDENTITY_NUMBER_RANDOM_REPLACE 和 TELEPHONE_RANDOM_REPLACE。用户还可以通过实现 SphereEx-DBPlusEngine 提供的接口，自行实现一套脱敏算法。

关于算法类型的详情，请参见[内置脱敏算法列表](#)。

脱敏表配置：用于告诉 SphereEx-DBPlusEngine 数据表里哪个列用于数据脱敏、使用什么算法脱敏。

逻辑列：用于计算脱敏列的逻辑名称，它是 SQL 中列的逻辑标识。

7.5.7 脱敏处理过程

举例说明，假如数据库里有一张表叫做 t_user，这张表里有一个字段 phone_number 使用 MASK_FROM_X_TO_Y 进行算法处理，SphereEx-DBPlusEngine 不会改变数据存储，只会按脱敏算法对结果进行装饰，从而达到脱敏的效果

如下图所示：

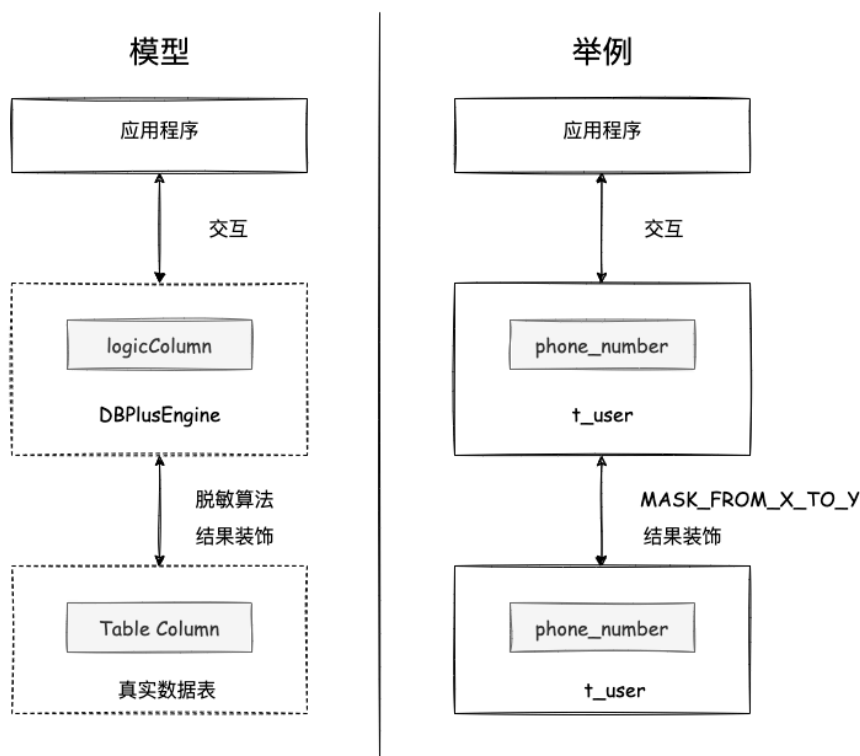


Fig. 21: 数据脱敏过程

7.5.8 配置示例

下面通过对电话号字段脱敏的简单的例子，来了解 SphereEx-DBPlusEngine 的配置和使用。

详细语法说明请参考[数据脱敏 DistSQL](#)。

1. 在 MySQL 中创建数据库 demo_ds_0

```
mysql> CREATE DATABASE demo_ds_0;
```

2. 在 SphereEx-DBPlusEngine 中创建名为 mask_db 的逻辑库

```
mysql> create database mask_db;
```

```
mysql> show databases;
```

```
mysql> use mask_db;
```



```
mysql> REGISTER STORAGE UNIT ds_0 (
  HOST="192.168.xx.103",
  PORT=3306,
  DB="demo_ds_0",
  USER="test",
  PASSWORD="Test@123"
);
```

```
mysql> SHOW STORAGE UNITS;
```

3. 在 mask_db 中创建测试表，并插入数据

```
mysql> CREATE TABLE t_mask (
  `id` int(11) NOT NULL,
  `telephone` varchar(64) DEFAULT NULL,
  `address` varchar(64) DEFAULT NULL,
  PRIMARY KEY (`id`)
);
Query OK, 0 rows affected (0.22 sec)

mysql> INSERT INTO t_mask (id, telephone, address) VALUES
(1,'13888888888','Beijing'),
(2,'18000100001','Nanjing'),
(3,'17337312345','Hangzhou'),
(4,'13654054321','Chengdu');
Query OK, 4 rows affected (0.06 sec)
```

4. 创建脱敏规则

将 telephone 字段中 4 到 7 位脱敏为 *。

```
mysql> CREATE MASK RULE t_mask (
  COLUMNS(
    (NAME=telephone,TYPE(NAME='KEEP_FIRST_N_LAST_M',PROPERTIES("first-n=3","last-m=4","replace-char="*")),
    (NAME=address,TYPE(NAME='MD5'))
  ));
Query OK, 0 rows affected (0.36 sec)

mysql> SHOW MASK RULES;
+-----+-----+-----+-----+
| table | column | algorithm_type | algorithm_props |
+-----+-----+-----+-----+
| t_mask | telephone | KEEP_FIRST_N_LAST_M | first-n=3,last-m=4,replace-char=* |
| t_mask | address | MD5 | |
+-----+-----+-----+-----+
2 rows in set (0.15 sec)
```

5. 查询脱敏后的数据

```
mysql> SELECT * FROM t_mask;
+----+-----+-----+
| id | telephone | address |
+----+-----+-----+
| 1 | 138****8888 | 78fb473f134eed43c959f9ebdeeb4050 |
| 2 | 180****0001 | 080fa4ae8318b4cc0be7f76fefe4838e |
| 3 | 173****2345 | 95e9c48630a252346b29cd52b365d3e2 |
| 4 | 136****4321 | f4aa575f70b3f78887deb96ce611b187 |
+----+-----+-----+
4 rows in set (0.00 sec)
```

可见，电话号码部分数字已经展示为星号，和预期一致。

6. 基于上述脱敏配置，变更后再进行校验

只展示 telephone 字段的首尾内容，中间用 ? 展示。

```
mysql> ALTER MASK RULE t_mask (
  COLUMNS(
    (NAME=telephone,TYPE(NAME='KEEP_FIRST_N_LAST_M',PROPERTIES("first-n=1","last-m=1","replace-char=?")),
    (NAME=address,TYPE(NAME='MD5'))
  ));
Query OK, 0 rows affected (0.27 sec)

mysql> SHOW MASK RULES;
+-----+-----+-----+-----+
| table | column | algorithm_type | algorithm_props |
+-----+-----+-----+-----+
| t_mask | telephone | KEEP_FIRST_N_LAST_M | first-n=1,last-m=1,replace-char=? |
| t_mask | address | MD5 | |
+-----+-----+-----+-----+
2 rows in set (0.04 sec)
```

7. 再次查询进行确认

```
mysql> SELECT * FROM t_mask;
+----+-----+-----+
| id | telephone | address |
+----+-----+-----+
| 1 | 1????????8 | 9bd8ad60a4e25ee4f07b545c65e268c6 |
| 2 | 1????????1 | 99116a7dc5f614c84f71834820505048 |
| 3 | 1????????5 | 7e2d6cf42a4f42358373f9363f341d3a |
| 4 | 1????????1 | f418a8bcbe75adb8721a9eacf16eeb0d |
+----+-----+-----+
4 rows in set (0.01 sec)
```

8. 删除过敏规则，查看数据

```
mysql> DROP MASK RULE t_mask;
Query OK, 0 rows affected (0.09 sec)

mysql> SHOW MASK RULES;
Empty set (0.00 sec)

mysql> SELECT * FROM t_mask;
+----+-----+-----+
| id | telephone | address |
+----+-----+-----+
| 1 | 13888888888 | Beijing |
| 2 | 18000100001 | Nanjing |
| 3 | 17337312345 | Hangzhou |
| 4 | 13654054321 | Chengdu |
+----+-----+-----+
4 rows in set (0.01 sec)
```

删除过敏规则后，可正常对数据进行检索，符合预期。

7.6 影子库压测

在基于微服务的分布式应用架构下，业务需要多个服务是通过一系列的服务、中间件的调用来完成，所以单个服务的压力测试已无法代表真实场景。在测试环境中，如果重新搭建一整套与生产环境类似的压测环境，成本过高，并且往往无法模拟线上环境的复杂度以及流量。因此，业内通常选择全链路压测的方式，即在生产环境进行压测，这样所获得的测试结果能够准确地反应系统真实容量和性能水平。

7.6.1 概述

基于内核的 SQL 解析能力以及可插拔平台架构，可实现压测数据与生产数据的隔离，帮助应用自动路由，支持全链路压测。帮助用户实现在生产环境进行压测，来获得较为准确的系统真实容量水平和性能测试结果。

SphereEx-DBPlusEngine 关注于全链路压测场景下数据库层面的解决方案，将压测数据自动路由至用户指定的数据库，是 SphereEx-DBPlusEngine 影子库插件的主要设计目标。

全链路压测是一项复杂而庞大的工作。需要各个微服务、中间件之间配合与调整，以应对不同流量以及压测标识的透传。通常会搭建一整套压测平台以适用不同测试计划。在数据库层面需要做好数据隔离，为了保证生产数据的可靠性与完整性，需要将压测产生的数据路由到压测环境数据库，防止压测数据对生产数据库中真实数据造成污染。这就要求业务应用在执行 SQL 前，能够根据透传的压测标识，做好数据分类，将相应的 SQL 路由到与之对应的数据源。

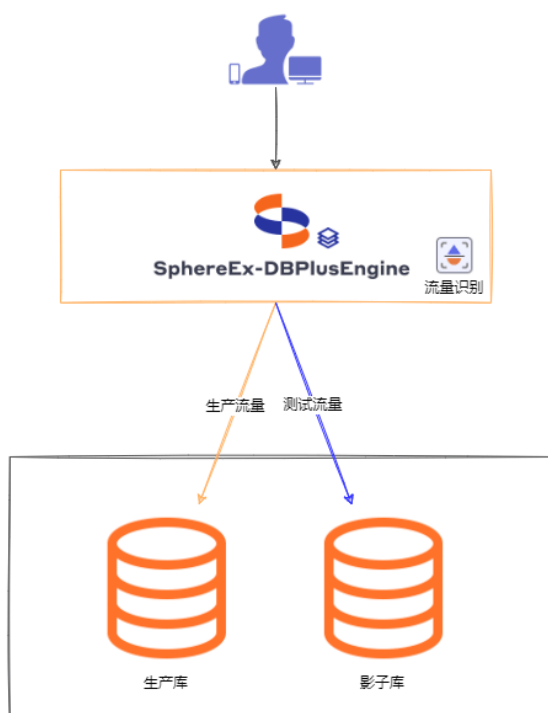


Fig. 22: 影子库

核心概念

■ 生产库

生产环境使用的数据库。

■ 影子库

压测中用来存储压测数据的数据库，应与生产数据库使用相同的配置。

适用场景

在基于微服务的分布式应用架构下，为了提升系统压力测试的准确性、降低测试成本，通常选择在生产环境进行压力测试，测试中风险也会大大提高。通过 SphereEx-DBPlusEngine 影子库插件能力，结合影子算法灵活的配置，可以获取系统真实的吞吐能力，同时可避免数据污染，满足复杂业务场景的在线压力测试需求。

使用前提

- 服务器中已安装 SphereEx-DBPlusEngine 和数据库集群，且服务运行正常；
- 已完成 SphereEx-Console 安装部署，在压测中实时掌握 SphereEx-DBPlusEngine 负载情况；（可选）
- 对于需要人为触发 Hint 的压测场景，MySQL 客户端需要使用 -c 参数，开启注释功能。

使用限制

基于 Hint 的影子算法

- 无。

基于列的影子算法

- 不支持情况：
 - 不支持 DDL。
 - 不支持范围、分组和子查询，如：BETWEEN 、GROUP BY ...HAVING 等。
- 支持情况：
 - INSERT

	SQL	是否支持
1	INSERT INTO table (column,...) VALUES (value,...)	支持
2	INSERT INTO table (column,...) VALUES (value,...),(value,...),...	支持
3	INSERT INTO table (column,...) SELECT column1 from table1 where column1 = value1	不支持

- SELECT/UPDATE/DELETE

	条件类型	SQL	是否支持
1	=	SELEC T/UPDATE/DELETE ...WHERE column = value	支持
2	LIKE/NOT LIKE	SELEC T/UPDATE/DELETE ...WHERE column LIKE/NOT LIKE value	支持
3	IN/NOT IN	SELEC T/UPDATE/DELETE ...WHERE column IN/NOT IN (v alue1,value2,...)	支持
4	BETWEEN	SELEC T/UPDATE/DELETE ...WHERE column BETWEEN value1 AND value2	不支持
5	GROUP BY ... HAVING...	SELEC T/UPDATE/DELETE ...WHERE ...GROUP BY column HAVING column > value	不支持
6	子查询	SELEC T/UPDATE/DELETE ...WHERE column = (SELECT column FROM table WHERE column = value)	不支持

注意事项

使用影子库压测方案时，需充分考虑业务系统的高可用问题，因此建议部署多个 SphereEx-DBPlusEngine 节点进行压测，避免发生单点故障。关于应用接入方案，可选择使用连接池配置多个 SphereEx-DBPlusEngine 地址，或采用其他成熟的负载均衡方案均可。

7.6.2 原理介绍

SphereEx-DBPlusEngine 通过解析 SQL，对传入的 SQL 进行影子判定，根据配置文件中用户设置的影子规则，将请求路由到生产库或者影子库。

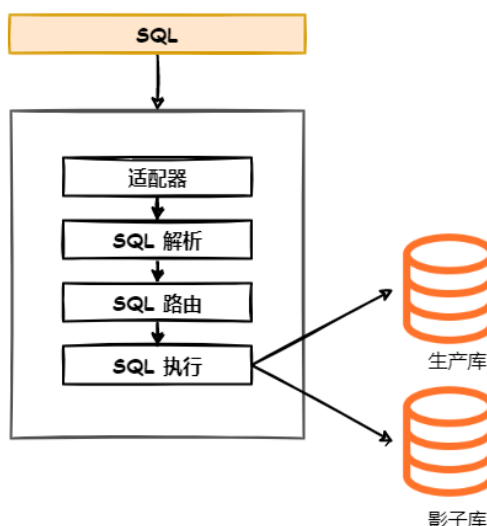


Fig. 23: 影子库原理

影子算法和业务实现紧密相关，目前提供基于列和基于 Hint 共两种类型影子算法。

- **基于列的影子算法**：通过识别 SQL 中的数据，匹配路由至影子库的场景。适用于由压测数据名单驱动的压测场景，支持精准匹配字段值和正则匹配字段值。
- **基于 Hint 的影子算法**：通过识别 SQL 中的「注释」，匹配路由至影子库的场景。适用于由上游系统透传标识驱动的压测场景。

以 INSERT 语句为例，在写入数据时，SphereEx-DBPlusEngine 会对 SQL 进行解析，再根据配置文件中的规则，构造一条路由链。在当前版本的功能中，影子功能处于路由链中的最后一个执行单元，即如果有其他需要路由的规则存在，如分片场景，SphereEx-DBPlusEngine 会首先根据分片规则，路由到某一个数据库，再执行影子路由判定流程，判定执行 SQL 满足影子规则的配置，数据路由到与之对应的影子库，生产数据则维持不变。

关于 DML 语句，SphereEx-DBPlusEngine 支持基于列和基于 Hint 两种影子库算法。

影子判定会首先判断执行 SQL 相关表与配置的影子表是否有交集。如果有交集，依次判定交集部分影子表关联的影子算法，有任何一个判定成功，SQL 语句路由到影子库。影子表没有交集或者影子算法判定不成功，SQL 语句路由到生产库。

关于 DDL 语句，SphereEx-DBPlusEngine 仅支持 Hint 影子算法。在压测场景下，DDL 语句一般不需要测试。主要在初始化或者修改影子库中影子表时使用。影子判定会首先判断执行 SQL 是否包含注解。如果包含注解，影子规则中配置的 HINT 影子算法依次判定。有任何一个判定成功，SQL 语句路由到影子库。执行 SQL 不包含注解或者 HINT 影子算法判定不成功，SQL 语句路由到生产库。

7.6.3 使用指南

基于生产环境，需要部署数个 SphereEx-DBPlusEngine 来承载业务流量，具体配置及数量视压测规模而定。还需要部署一套与生产库配置、架构一致的影子库来承载压测数据。

1. 环境确认

在使用 SphereEx-DBPlusEngine 读写分离插件前，需要对当前环境做较全面的确认，如生产库和影子库的配置信息、数据库软件版本信息，以及应用服务器到生产库和影子库的网络延迟等信息。

2. 规则确认

根据压测内容，参考影子库使用限制选择合适的基于列或者基于 Hint 算法，也可同时配置两种算法。

3. 启用 SQL 注释解析（可选）

如需要使用基于 Hint 的影子库算法，则需要将 `./conf/server.yaml` 文件中的 `sqlCommentParseEnabled` 参数值修改为 `true`，修改完成后需要重启 SphereEx-DBPlusEngine。

如果参数未正确完成配置，则 SphereEx-DBPlusEngine 将不能处理带有 Hint 的语句，压测数据会直接路由至生产库，因此请务必谨慎配置和验证，配置完成后可通过 `PREVIEW` 命令进行路由校验，参考影子配置验证说明。

4. 数据源确认及注册

依次确认所有生产、影子数据源的连通性，可在 SphereEx-DBPlusEngine 所在节点进行访问确认。在 SphereEx-DBPlusEngine 中，通过 `REGISTER STORAGE UNIT` 命令对数据源进行注册。

5. 创建影子库规则

创建影子库规则时，根据第二步所规划的规则进行配置，选择合适的影子库算法，或混合使用两种影子库算法。

6. 影子配置验证

使用 `PREVIEW SQL` 命令验证 SQL 路由情况，该命令不会真实执行 SQL，仅仅确认执行路径。如果影子库规则未生效，请再次确认影子库规则配置详情、压测 SQL 特征以及 `sqlCommentParseEnabled` 的配置情况。请务必确认影子规则准确无误后，再进入全链路压测流程。

7. 修改应用程序 url

将应用程序中数据库的访问信息修改为 SphereEx-DBPlusEngine 的访问地址，可通过连接池配置或其他负载均衡方案完成应用程序接入。

8. 进入压测流程

在合适的压测时间段，使用压测工具开始对系统进行全链路压测，压测过程需要通过 SphereEx-Console 实时观察 SphereEx-DBPlusEngine 负载情况。

9. 再次修改应用程序 url

全链路压测完成后，可将应用程序中数据库访问信息修改回数据库的地址，恢复系统拓扑。

10. 影子库处理

最后，根据使用需求，可对影子库中的测试数据保留或清理。如需清理，清理前请务必确保访问的数据库是影子库，避免对生产库误操作带来损失。

7.6.4 操作指南

1. 准备一个 SphereEx-DBPlusEngine 实例，部署操作将不再演示，准备影子库
2. 在 SphereEx-DBPlusEngine 中完成逻辑库的创建
3. 添加数据源，完成集群构建
4. 创建影子库规则及创建影子表
5. 验证影子规则生效情况

7.6.5 配置示例

环境说明

本示例中，需使用 3 台测试机，分别部署 1 个 SphereEx-DBPlusEngine 实例和 2 个 MySQL 实例。

\	实例	IP 地址	服务端口	主机名	备注
1	DBPlusEngine 1.2.0	192.168.xx.102	3307	dbplusengine	
2	MySQL 8.0.28	192.168.xx.103	3306	ds_0	生产库
3	MySQL 8.0.28	192.168.xx.104	3306	ds_1	影子库

拓扑图

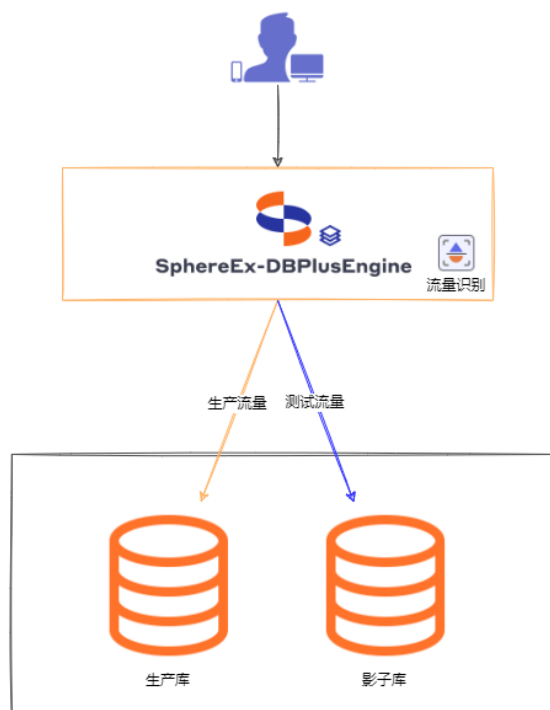


Fig. 24: 拓扑图

配置过程

1. 准备生产库、影子库数据源

创建 ds_product 数据源，模拟生产库，同时创建 t_user 表并插入若干条记录。

```
mysql -uroot -p -h192.168.xx.103 -P3306
```

```
CREATE DATABASE IF NOT EXISTS ds_product;  
Query OK, 1 row affected (0.01 sec)
```

```
USE ds_product;  
Database changed
```

```
CREATE TABLE t_user  
(  
  mobile char(11) PRIMARY KEY,  
  status varchar(18) NOT NULL,  
  type varchar(255) NOT NULL  
);  
Query OK, 0 rows affected (0.11 sec)
```

```
SHOW TABLES;
```



```

+-----+
| Tables_in_ds_product |
+-----+
| t_user                |
+-----+
1 row in set (0.00 sec)

INSERT INTO t_user (mobile, status, type)
VALUES (18099515621, 1, 'ds_product'),
       (15639784703, 1, 'ds_product'),
       (15716172114, 1, 'ds_product'),
       (18766747515, 1, 'ds_product'),
       (18099515602, 1, 'ds_product'),
       (15639784713, 1, 'ds_product'),
       (18099515622, 1, 'ds_product'),
       (15639784513, 1, 'ds_product'),
       (15716173114, 1, 'ds_product'),
       (18766746515, 1, 'ds_product');
Query OK, 10 rows affected (0.06 sec)
Records: 10 Duplicates: 0 Warnings: 0

SELECT * FROM t_user;
+-----+-----+-----+
| mobile | status | type |
+-----+-----+-----+
| 15639784513 | 1 | ds_product |
| 15639784703 | 1 | ds_product |
| 15639784713 | 1 | ds_product |
| 15716172114 | 1 | ds_product |
| 15716173114 | 1 | ds_product |
| 18099515602 | 1 | ds_product |
| 18099515621 | 1 | ds_product |
| 18099515622 | 1 | ds_product |
| 18766746515 | 1 | ds_product |
| 18766747515 | 1 | ds_product |
+-----+-----+-----+
10 rows in set (0.00 sec)

```

创建 ds_shadow 数据源，为影子库。

```
mysql -uroot -p -h192.168.xx.104 -P3306
```

```
CREATE DATABASE IF NOT EXISTS ds_shadow;
Query OK, 1 row affected (0.05 sec)
```

```
CREATE TABLE t_user
(
  mobile char(11) PRIMARY KEY,
  status varchar(18) NOT NULL,
  type varchar(255) NOT NULL
);
Query OK, 0 rows affected (0.11 sec)
```

2. 使用 MySQL 客户端登陆 SphereEx-DBPlusEngine，创建逻辑库

注意：请务必加 -c 参数进行访问，来确保后续带有注释的 SQL 可生效。

```
--访问 DBPlusEngine 实例
```

```
mysql -uroot -p -P3307 -h192.168.xx.102 -c
```

```
CREATE DATABASE testdb;
Query OK, 0 rows affected (0.21 sec)
```

```
SHOW DATABASES;
```

5 rows in set (0.00 sec)

3. 添加数据源，完成集群构建

在 SphereEx-DBPlusEngine 中注册两个数据库实例。

```
USE testdb;
Database changed

REGISTER STORAGE UNIT ds_0 (
  URL="jdbc:mysql://192.168.xx.103:3306/ds_product?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
), ds_1 (
  URL="jdbc:mysql://192.168.xx.104:3306/ds_shadow?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123"
);
Query OK, 0 rows affected (0.05 sec)

SHOW STORAGE UNITS\G
```

4. 创建影子库规则及创建影子表

注意：此处创建影子表使用了注释操作，建表之前请再次确认该会话是通过 -c 参数开启的。

```
--创建影子库规则
CREATE SHADOW RULE shadow_rule(
  SOURCE=ds_0,
  SHADOW=ds_1,
  t_user(TYPE(NAME=VALUE_MATCH, PROPERTIES("operation"="insert","column"="status","value"='2' )))
);
Query OK, 0 rows affected (0.20 sec)
```

该步骤创建了一个影子库规则，其中包括了基于值和基于 hint 的影子算法匹配：- 基于值匹配：当 insert 操作中，status 为 2 的记录会被写入到影子库中

```
SHOW SHADOW RULES;
+-----+-----+-----+-----+
| rule_name | source_name | shadow_name | shadow_table |
+-----+-----+-----+-----+
| shadow_rule | ds_0 | ds_1 | t_user |
+-----+-----+-----+-----+
1 row in set (0.10 sec)
```

5. 验证影子规则生效情况先通过 PREVIEW 命令验证影子规则已生效，复合影子库规则的数据应被写入到 ds_1 数据源中。

```
PREVIEW INSERT INTO t_user (mobile, status, type) VALUES (18236483857, 2, 'ds_shadow');
+-----+-----+-----+-----+
| data_source_name | actual_sql |
+-----+-----+-----+-----+
| ds_1 | INSERT INTO t_user (mobile, status, type) VALUES (18236483857, 2, 'ds_shadow') |
+-----+-----+-----+-----+
1 row in set (0.50 sec)
```

根据如上结果确认，复合影子规则的两条记录将会写入到影子库中，符合预期，下面将在 SphereEx-DBPlusEngine 中插入测试数据。

```
--在 Proxy 中查询 t_user 表的信息
SELECT * FROM t_user;
+-----+-----+-----+
| mobile | status | type |
+-----+-----+-----+
| 15639784513 | 1 | ds_product |
+-----+-----+-----+
```

```

15639784703 | 1 | ds_product |
15639784713 | 1 | ds_product |
15716172114 | 1 | ds_product |
15716173114 | 1 | ds_product |
18099515602 | 1 | ds_product |
18099515621 | 1 | ds_product |
18099515622 | 1 | ds_product |
18766746515 | 1 | ds_product |
18766747515 | 1 | ds_product |

```

```
+-----+-----+-----+
```

10 rows in set (0.01 sec)

--基于影子库列值算法的测试数据

INSERT INTO t_user (mobile, status, **type**)

VALUES (18236483857, 2, 'ds_shadow'),

(15686689114, 2, 'ds_shadow'),

(14523360225, 2, 'ds_shadow'),

(18143924353, 2, 'ds_shadow'),

(15523349333, 2, 'ds_shadow'),

(18143924153, 2, 'ds_shadow'),

(15523349313, 2, 'ds_shadow'),

(18143924253, 2, 'ds_shadow'),

(15523349323, 2, 'ds_shadow'),

(13261527931, 2, 'ds_shadow');

Query OK, 10 rows affected (0.06 sec)

mysql> **SELECT * FROM** t_user;

```
+-----+-----+-----+
```

```
| mobile | status | type |
```

```
+-----+-----+-----+
```

```

15639784513 | 1 | ds_product |
15639784703 | 1 | ds_product |
15639784713 | 1 | ds_product |
15716172114 | 1 | ds_product |
15716173114 | 1 | ds_product |
18099515602 | 1 | ds_product |
18099515621 | 1 | ds_product |
18099515622 | 1 | ds_product |
18766746515 | 1 | ds_product |
18766747515 | 1 | ds_product |

```

```
+-----+-----+-----+
```

10 rows in set (0.01 sec)

可见，在插入符合影子库的数据后，SphereEx-DBPlusEngine 中仍然只能查到生产库的数据，符合预期。

最后，在影子库中确认数据。

SELECT * FROM t_user;

```
+-----+-----+-----+
```

```
| mobile | status | type |
```

```
+-----+-----+-----+
```

```

13261527931 | 2 | ds_shadow |
14523360225 | 2 | ds_shadow |
15523349313 | 2 | ds_shadow |
15523349323 | 2 | ds_shadow |
15523349333 | 2 | ds_shadow |
15686689114 | 2 | ds_shadow |
18143924153 | 2 | ds_shadow |
18143924253 | 2 | ds_shadow |
18143924353 | 2 | ds_shadow |
18236483857 | 2 | ds_shadow |
18816175114 | 0 | ds_shadow |

```

```
+-----+-----+-----+
```

11 rows in set (0.00 sec)

从输出结果确认，能够看到 status 为 2 和 0 的数据，影子库场景的验证结果符合预期。

7.6.6 FAQ

1. 使用影子库压测方案是否会给生产库带来影响？

不会有影响。在正确配置了规则的前提下，影子库会对测试数据进行隔离，不会对生产库带来任何影响。

2. 能否创建两个完全一样的影子规则？

不支持。在 SphereEx-DBPlusEngine 可创建多个影子库规则，但是不能作用于同一个字段。

3. 使用影子库后，生产系统的性能衰减有多少？

根据测试结果，生产环境的性能损耗在 30% ~ 50% 之间，具体可参考[性能白皮书](#)。

7.7 流量治理

随着数据规模的不断膨胀，使用多节点集群的分布式方式逐渐成为趋势。对集群整体视角的统一管理能力，和针对单独组件细粒度的控制能力，是基于存算分离架构体系中不可或缺的功能。

7.7.1 概述

SphereEx-DBPlusEngine 为用户提供了流量治理的入口，为用户提供集群的集中化管理的统一管理能力以及在单点出现故障时精细化的操作能力。

集中化管理的挑战体现在包括数据库存储节点和中间件计算节点的状态统一管理，并且能够实时的探测到分布式环境下最新的变动情况，进一步为集群的控制和调度提供依据。

7.7.2 基本概念

■ 熔断

阻断 SphereEx-DBPlusEngine 的连接入口。当某个 SphereEx-DBPlusEngine 节点超过负载后，停止该节点对数据库的访问，使数据库能够保证足够的资源为其他节点提供服务。

适用场景

适用于计算节点过载保护的场景，当 SphereEx-DBPlusEngine 集群内某个计算节点超过负载后，通过熔断功能，阻断应用到该计算节点的流量，保证整个集群继续提供稳定服务。

使用前提

- 已安装部署 SphereEx-DBPlusEngine 和数据库集群，且服务运行正常；
- 已安装部署 SphereEx-Console，且服务运行正常。（可选）

使用限制

无

注意事项

熔断操作执行后，正在执行中的请求会被中断，业务有感知。

7.7.3 原理介绍

面对超负荷的流量下，SphereEx-DBPlusEngine 可针对某一节点进行熔断，以保证整个数据库集群得以继续运行。

7.7.4 使用指南

1. 确认当前集群流量

通过 SphereEx-Console 或数据库确认集群整体流量情况，判断是否有将要过载的计算节点。

2. 熔断计算节点

确认待熔断节点的会话情况，确认可熔断后，在其他 Proxy 以命令行方式对待熔断节点进行熔断，并确认熔断结果。被熔断节点的请求将被中断。

3. 确认熔断结果

再次通过 SphereEx-Console 观察流量情况。

7.7.5 操作指南

1. 通过 Console 确认当前集群及各节点流量

2. 确认集群环境

3. 确认当前 Proxy 中进行的请求

4. 熔断指定计算节点

5. 确认熔断结果

7.7.6 配置示例

1. 通过 Console 确认当前集群及各节点流量

2. 确认集群环境

```
mysql> SHOW COMPUTE NODES;
```

instance_id	host	port	status	mode_type	worker_id	labels	version
01faadb5-3b09-4449-ad1a-252ce0c50117	192.168.xx.102	3307	OK	Cluster	-1		1.3.0
9fca0beb-edc9-4b0f-8e3c-1a3187461565	192.168.xx.103	3307	OK	Cluster	-1		1.3.0

2 rows in set (0.01 sec)

3. 确认当前 Proxy 中进行的请求

```
mysql> SHOW PROCESSLIST;
```

4. 熔断指定计算节点

```
mysql> DISABLE INSTANCE '9fca0beb-edc9-4b0f-8e3c-1a3187461565';
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> SHOW COMPUTE NODES;
```

instance_id	host	port	status	mode_type	worker_id	labels	version
01faadb5-3b09-4449-ad1a-252ce0c50117	192.168.xx.102	3307	OK	Cluster	-1		1.3.0
9fca0beb-edc9-4b0f-8e3c-1a3187461565	192.168.xx.103	3307	CIRCUIT_BREAK	Cluster	-1		1.3.0

```
2 rows in set (0.02 sec)
```

5. 确认熔断结果

可再次通过 Console 观察流量情况。

若手动在被熔断节点进行操作，操作将无法被执行。

```
mysql> SELECT * FROM WHERE t_user WHERE ID=1;
ERROR 13010 (01000): Circuit break open, the request has been ignored.
```

7.7.7 FAQ

1. 是否可以熔断全部的计算节点？

不可以。熔断操作不能熔断当前节点，因此计算集群中会保留一个计算节点。

2. 对指定计算节点熔断后，是否会影响该节点上未执行完的请求？

有影响。正在执行中的请求会被中断，业务有感知。

3. 熔断操作对监控日志和治理中心是否有影响？

无影响。

4. 熔断 Proxy 后，是否会对 Traffic Rule 的策略带来影响？

无影响。被熔断的 Proxy 将不会接收 Traffic Rule 的流量。

7.8 双路由

SphereEx-DBPlusEngine 包括 Driver 和 Proxy 两种接入形态，在复杂的业务场景下，选择混合部署架构会更为合适，即混合部署 DBPlusEngine-Driver 和 DBPlusEngine-Proxy，并采用统一的注册中心来配置分片策略，灵活地搭建出适用于各种场景的应用系统。

为了提升应用程序的查询性能和稳定性，我们可以考虑将耗资源较多的 SQL 转发至独立服务形态的 DBPlusEngine-Proxy 中，由 DBPlusEngine-Proxy 负责计查询结果，再统一返回给应用程序。

7.8.1 概述

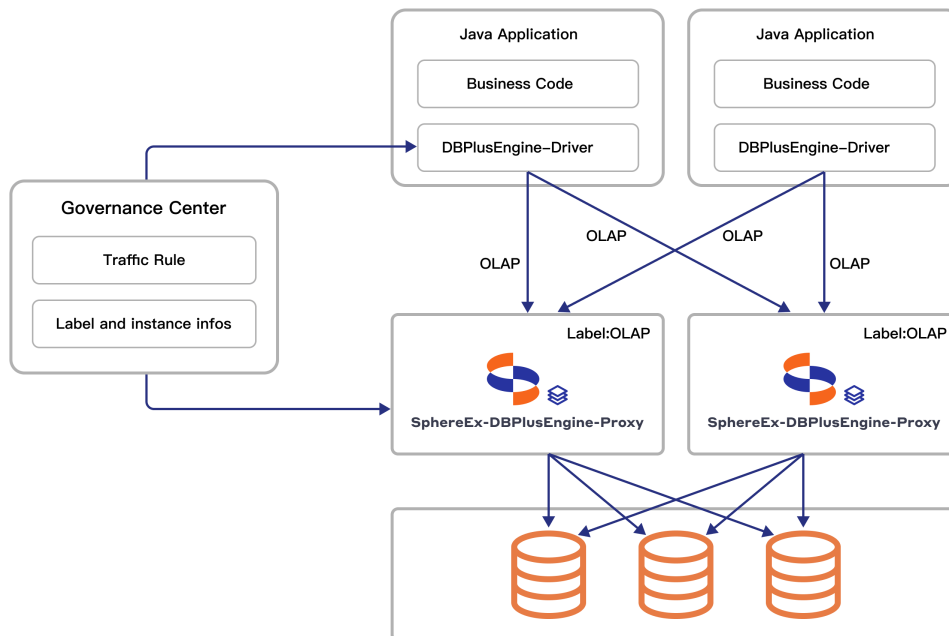
SphereEx-DBPlusEngine 的双路由插件允许用户根据 SQL 特点进行路由配置，耗资源的 SQL 可由 DBPlusEngine-Proxy 专门负责处理，充分发挥两种接入端所长。双路由插件会透明化 SQL 转发所带来的影响，让用户像使用一个数据库一样使用 DBPlusEngine-Driver 和 DBPlusEngine-Proxy 混合部署集群。

■ DBPlusEngine-Driver

DBPlusEngine-Driver 采用无中心化架构，与应用程序共享资源，适用于 Java 开发的高性能的轻量级 OLTP 应用，但是由于 DBPlusEngine-Driver 与应用程序共享资源，当执行消耗资源较多的 SQL 时，应用程序的稳定性和性能会受到影响。同时 DBPlusEngine-Driver 消耗的连接数较多，当应用程序和数据库部署在不同的网络分区时，网络延迟对于性能的影响也会更加明显。

■ DBPlusEngine-Proxy

DBPlusEngine-Proxy 提供统一的静态入口，并且独立于应用程序部署，适用于 OLAP 应用以及对分片数据库进行管理和运维的场景。用户使用 DBPlusEngine-Proxy 来执行资源消耗较多的 SQL，可以有效避免对应用程序产生影响。



基本概念

标签

DBPlusEngine-Proxy 实例配置的标签属性，用于对实例进行区分，DBPlusEngine-Driver 开启双路由功能时，转发的目标就是标签对应的 DBPlusEngine-Proxy 实例。

转发策略

对 DBPlusEngine-Driver 接入端的 SQL 进行转发的策略，包括了：目标 Proxy 实例标签、转发算法、负载均衡算法。根据算法的不同，可以将转发策略分为事务转发策略和普通转发策略，当策略中的算法配置为 TransactionTrafficAlgorithm 时，转发策略为事务转发策略，当策略中的算法配置为其他算法时，转发策略为普通转发策略。

转发算法

用于判断当前 SQL 是否需要转发的算法，包括了 HintTrafficAlgorithm、SegmentTrafficAlgorithm 和 TransactionTrafficAlgorithm。

- HintTrafficAlgorithm 基于 SQL Hint 功能进行 SQL 转发。
- SegmentTrafficAlgorithm 则基于 SQL 语句进行转发，内部提供了基于 SQL 字符串以及 SQL 正则匹配的转发算法。
- TransactionTrafficAlgorithm 是专门用于处理开启事务时，事务中的 SQL 语句如何进行转发的算法，目前支持 FIRST_SQL、JDBC 和 Proxy 三种策略。FIRST_SQL 表示基于事务语句中的第一条 SQL 转发结果，对事务中的 SQL 语句进行转发。JDBC 表示事务语句的 SQL 全部转发至 JDBC 接入端执行。Proxy 表示事务语句的 SQL 全部转发至 Proxy 接入端执行。

负载均衡算法

通过负载均衡算法将 SQL 语句转发到标签对应的不同 Proxy 实例上执行，目前支持 RANDOM 和 ROUND_ROBIN 两种负载均衡算法。

适用场景

混合部署 DBPlusEngine-Driver 和 DBPlusEngine-Proxy，需要将消耗资源较多的 SQL 转发到 DBPlusEngine-Proxy 执行的场景。

使用前提

- 双路由功能需要使用混合部署架构，同时部署 DBPlusEngine-Driver 和 DBPlusEngine-Proxy；
- 需部署 ZooKeeper，混合架构将通过注册中心（ZooKeeper）统一管理。

使用限制

- 用户配置的内核功能不支持的 SQL，转发之后 SQL 仍然不支持；
- 不支持开启事务后将 SQL 转发到不同的接入端或 Proxy 实例执行；
- 只允许在 DBPlusEngine-Proxy 接入端通过 YAML 配置或者 DistSQL 来添加路由规则。

注意事项

虽然通过将 DBPlusEngine-Driver 接入端消耗资源较多的 SQL 转发到 DBPlusEngine-Proxy 执行，可以有效提升应用程序的性能和稳定性，但是这也导致了部署架构更加复杂，用户需要判断哪些语句需要转发到 DBPlusEngine-Proxy，并在 DAO 层开发相关的业务逻辑来控制 SQL 的转发。

此外，在开启事务的场景下，将事务中的部分 SQL 转发到 DBPlusEngine-Proxy 执行，会对事务的一致性和可见性带来影响，从而影响业务系统对于事务的使用。

7.8.2 原理介绍

Traffic Rule 用于判断当前执行的 SQL 语句是否需要转发，内部会支持多种转发算法。下面展示了 Traffic Rule 配置示例，trafficStrategies 用于声明请求转发策略，当用户配置了多个转发策略时，会按照配置的顺序依次去匹配，第一个匹配成功的策略将用于请求转发处理。

转发策略包含三个属性：

- labels：请求转发 Proxy 对应的标签，支持配置多个标签，并且同一个标签也可能存在多个 Proxy 实例；
- algorithmName：请求转发匹配算法，例如匹配操作类型是 select，并且包含了 group by 的语句，匹配成功则根据 labels 查找转发的 Proxy 实例；
- loadBalancerName：用于指定转发的负载均衡策略，因为 labels 对应的 Proxy 实例可能包含多个，需要通过负载均衡策略决定转发到哪个实例上。

为了配合 Traffic Rule 中的 labels 属性，需要 Proxy 接入端支持 label 属性设置，并且能够将 Proxy 实例对应的 ip 和端口信息存储下来。

由于使用了双路由功能，原先在 JDBC 接入端执行的 SQL，会因为匹配转发策略，而被转发到 Proxy 接入端执行。在这种场景下，同一个事务单元中的语句，可能部分在 JDBC 接入端执行，部分在 Proxy 接入端执行，无法保证事务的一致性。

为了解决事务一致性的问题，我们在转发策略中增加了事务转发策略，事务转发策略的配置中包含 FIRST_SQL、JDBC、PROXY 三种类型事务匹配算法，FIRST_SQL 会根据第一条 SQL 转发的结果来决定事务单元的转发结果，JDBC 会将事务单

元不进行转发，在 JDBC 上执行执行，而 PROXY 会将事务单元转发到 Proxy 实例执行，为了保证数据的一致性，事务单元会在同一个实例上执行。

Traffic Rule 内部在初始化路由策略时，会根据是否针对事务场景，将策略进行分类维护，针对事务场景的策略，只有在开启事务时使用，否则使用其他非事务策略。

7.8.3 使用指南

- 1. 确认当前系统中消耗资源较高的 SQL，如聚合类语句，提取具体语句信息。可通过 SphereEx-Console 或其他监控工具来捕获。
- 2. 在 DBPlusEngine-Proxy 中，通过 DistSQL 完成指定 SQL 的配置。
- 3. 在 DBPlusEngine-Proxy 中开启 SQL 显示功能，业务发起新的请求，确认指定 SQL 的路由情况。

7.8.4 操作指南

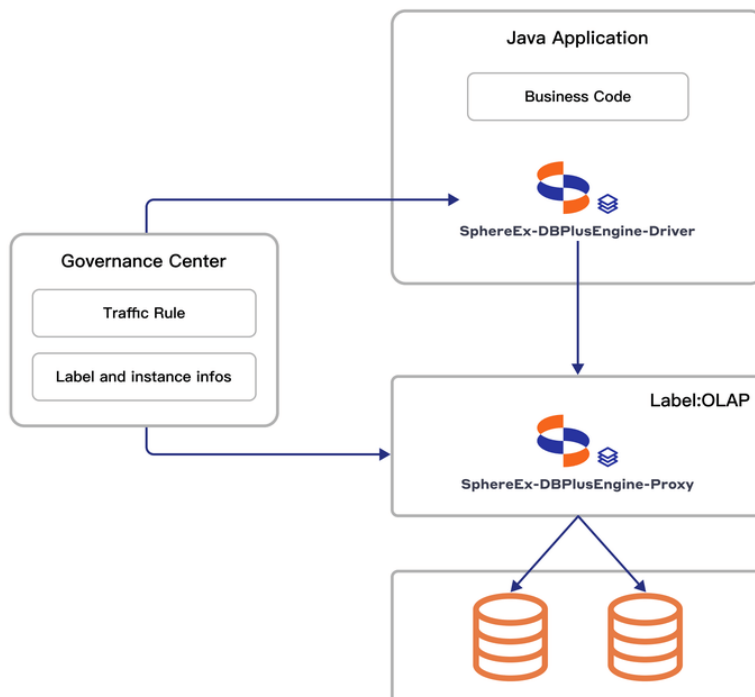
- 1. 确认需要转发到 Proxy 运行的 SQL 语句；
- 2. 在 Proxy 中，通过 DistSQL 完成双路由规则的配置；
- 3. 在 Proxy 中开启 SQL 展示；
- 4. 应用测发起请求，通过 Proxy 日志确认 SQL；
- 5. 确认结果无误后，在 Proxy 中关闭 SQL 展示；

7.8.5 配置示例

环境说明

	实例	IP 地址	服务端口	主机名	备注
1	DBPlusEngine 1.2.0	192.168.xx.102	3307	dbplusengine	
2	Zookeeper 3.6.3	192.168.xx.102	2181	dbplusengine	
3	MySQL 8.0.28	192.168.xx.103	3306	mysql_0	
4	MySQL 8.0.28	192.168.xx.104	3306	mysql_1	

拓扑图



配置过程

1. 确认需要转发到 Proxy 运行的 SQL 语句
2. 配置双路由规则

```

mysql> CREATE TRAFFIC RULE sql_match_traffic (
  LABELS(OLTP),
  TRAFFIC_ALGORITHM(TYPE(NAME=SQL_MATCH,PROPERTIES("sql" = "SELECT * FROM t_order WHERE order_id = ?; UPDATE t_
order SET order_id = ?;")),
  LOAD_BALANCER(TYPE(NAME=RANDOM, PROPERTIES("key"="value"))))
), sql_hint_traffic_1(
  TRAFFIC_ALGORITHM(TYPE(NAME= SQL_HINT)),
  LOAD_BALANCER(TYPE(NAME=ROUND_ROBIN, PROPERTIES("key"="value"))))
), sql_hint_traffic_2(
  LABELS(OLAP, order_by),
  TRAFFIC_ALGORITHM(TYPE(NAME= SQL_HINT))
), sql_hint_traffic_3(
  TRAFFIC_ALGORITHM(TYPE(NAME= SQL_HINT))
);

mysql> SHOW TRAFFIC RULES;
    
```

3. 在 Proxy 中开启 SQL 展示

```

mysql> SET VARIABLE sql_show = true;
Query OK, 0 rows affected (0.05 sec)

mysql> SHOW ALL VARIABLES;
+-----+-----+
| variable_name | variable_value |
+-----+-----+
    
```

```

sql_show          | true      |
sql_simple        | false     |
kernel_executor_size | 0         |
max_connections_size_per_query | 1         |
check_table_metadata_enabled | false     |
sql_federation_enabled | true      |
proxy_frontend_database_protocol_type |          |
proxy_frontend_flush_threshold | 128       |
proxy_hint_enabled | false     |
proxy_backend_query_fetch_size | -1        |
proxy_frontend_executor_size | 0         |
proxy_backend_executor_suitable | OLAP      |
proxy_frontend_max_connections | 0         |
proxy_backend_driver_type | JDBC      |
proxy_mysql_default_version | 5.7.22    |
proxy_default_port | 3307      |
proxy_netty_backlog | 1024      |
proxy_instance_type | Proxy     |
agent_plugins_enabled | true      |
cached_connections | 0         |
transaction_type   | LOCAL     |
+-----+-----+
21 rows in set (0.01 sec)

```

4. 应用测发起请求，通过 Proxy 日志确认 SQL

5. 在 Proxy 中关闭 SQL 展示

```

mysql> SET VARIABLE sql_show = false;
Query OK, 0 rows affected (0.01 sec)

```

```

mysql> SHOW ALL VARIABLES;

```

```

+-----+-----+
| variable_name | variable_value |
+-----+-----+
sql_show          | false         |
sql_simple        | false         |
kernel_executor_size | 0            |
max_connections_size_per_query | 1            |
check_table_metadata_enabled | false        |
sql_federation_enabled | true         |
proxy_frontend_database_protocol_type |            |
proxy_frontend_flush_threshold | 128          |
proxy_hint_enabled | false        |
proxy_backend_query_fetch_size | -1           |
proxy_frontend_executor_size | 0            |
proxy_backend_executor_suitable | OLAP         |
proxy_frontend_max_connections | 0            |
proxy_backend_driver_type | JDBC         |
proxy_mysql_default_version | 5.7.22       |
proxy_default_port | 3307         |
proxy_netty_backlog | 1024         |
proxy_instance_type | Proxy        |
agent_plugins_enabled | true         |
cached_connections | 0            |
transaction_type   | LOCAL        |
+-----+-----+
21 rows in set (0.00 sec)

```

7.8.6 语法和参数

创建 & 删除规则

创建双路由规则：

```
CREATE TRAFFIC RULE sql_match_traffic (
  LABELS(OLTP),
  TRAFFIC_ALGORITHM(TYPE(NAME=SQL_MATCH,PROPERTIES("sql" = "SELECT * FROM t_order WHERE order_id = ?; UPDATE t_order SET order_id = ?;")),
  LOAD_BALANCER(TYPE(NAME=RANDOM, PROPERTIES("key"="value")))
), sql_hint_traffic_1(
  TRAFFIC_ALGORITHM(TYPE(NAME= SQL_HINT)),
  LOAD_BALANCER(TYPE(NAME=ROUND_ROBIN, PROPERTIES("key"="value")))
), sql_hint_traffic_2(
  LABELS(OLAP, order_by),
  TRAFFIC_ALGORITHM(TYPE(NAME= SQL_HINT))
), sql_hint_traffic_3(
  TRAFFIC_ALGORITHM(TYPE(NAME= SQL_HINT))
);
```

LABELS 与 LOAD_BALANCER 为可选字段，示例中省略字段信息如下。

- sql_match_traffic 无省略字段
- sql_hint_traffic_1 省略 LABELS 字段
- sql_hint_traffic_2 省略 LOAD_BALANCER 字段
- sql_hint_traffic_3 省略 LABELS 与 LOAD_BALANCER 字段

删除双路由规则：

```
DROP TRAFFIC RULE sql_hint_traffic_2, sql_hint_traffic_3;
```

查看规则

```
-- 查询所有规则
SHOW TRAFFIC RULES;
-- 查询指定规则
SHOW TRAFFIC RULE sql_match_traffic;
```

示例：查看规则

```
SHOW TRAFFIC RULE sql_match_traffic;
+-----+-----+-----+-----+-----+-----+
| name      | labels | algorithm_type | algorithm_props | load_balancer_type | load_balancer_props |
+-----+-----+-----+-----+-----+-----+
| sql_match_traffic | OLTP   | SQL_MATCH   | sql=SELECT * FROM t_order WHERE order_id = ?; UPDATE t_order SET order_id = ?; | RANDOM | key=value |
```

修改规则

```
ALTER TRAFFIC RULE sql_match_traffic (  
  LABELS(OLTP,LIMIT),  
  TRAFFIC_ALGORITHM(TYPE(NAME=SQL_MATCH,PROPERTIES("sql" = "SELECT * FROM t_order WHERE order_id = ?;")),  
  LOAD_BALANCER(TYPE(NAME=RANDOM, PROPERTIES("key1"="value1")))  
, sql_hint_traffic_1(  
  TRAFFIC_ALGORITHM(TYPE(NAME= SQL_HINT))  
, sql_hint_traffic_2(  
  LABELS(OLAP, order_by),  
  TRAFFIC_ALGORITHM(TYPE(NAME= SQL_HINT))  
, sql_hint_traffic_3(  
  TRAFFIC_ALGORITHM(TYPE(NAME= SQL_HINT)),  
  LOAD_BALANCER(TYPE(NAME=ROUND_ROBIN, PROPERTIES("key"="value")))  
);
```

LABELS 与 LOAD_BALANCER 为可选字段，示例中省略字段信息如下。

- sql_match_traffic 无省略字段
- sql_hint_traffic_1 省略 LABELS 字段
- sql_hint_traffic_2 省略 LOAD_BALANCER 字段
- sql_hint_traffic_3 省略 LABELS 与 LOAD_BALANCER 字段

7.8.7 FAQ

1. 同一个事务单元中的语句，可能部分在 JDBC 接入端执行，部分在 Proxy 接入端执行，是否还能够保证事务的一致性？能够保证事务一致性。事务转发策略的配置中内置了三种事务匹配算法，包含 FIRST_SQL、JDBC、PROXY。

- FIRST_SQL 会根据第一条 SQL 转发的结果来决定事务单元的转发结果。
- JDBC 会将事务单元不进行转发，在 JDBC 上执行执行。
- PROXY 会将事务单元转发到 Proxy 实例执行，为了保证数据的一致性，事务单元会在同一个实例上执行。

7.9 数据迁移

当业务持续快速发展，数据量和并发量达到一定程度，传统单体数据库可能面临性能、可扩展性及可用性问题。业界曾提出 NoSQL 解决方案，通过数据分片和水平扩容解决以上问题，但是 NoSQL 数据库通常不支持事务和 SQL。

SphereEx-DBPlusEngine 支持对传统单机数据库进行数据分片，同时还支持分布式事务和 SQL，可以解决以上问题。SphereEx-DBPlusEngine 提供的数据库迁移方案可以助力传统单体数据库平滑切换到 SphereEx-DBPlusEngine 分布式集群中。

7.9.1 概述

SphereEx-DBPlusEngine 数据库迁移插件实现了同构数据库由单体到分布式的拆分迁移，该迁移操作可在线完成，同时支持源端目标段的数据比对。SphereEx-DBPlusEngine 数据库迁移可减少数据迁移时的业务影响，提供一站式的通用数据库迁移解决方案。

目前 SphereEx-DBPlusEngine 数据库迁移插件所支持的数据库产品为 MySQL、MySQL 分支版本、PostgreSQL 和 openGauss，具体版本、环境要求和权限要求如下。

	数据库	版本支持	环境要求	权限要求
1	MySQL	5.1.15 ~ 8.x	my.cnf 配置 <code>log-bin=mysql-binbinlog-format=rowbinlog-row-image=full</code> –确认命令如下 <code>show variables like '%log_bin%';show variables like '%binlog%';</code>	<code>GRANT REPLICATION SLAVE, REPLICATION CLIENT ON . TO :math:{u ser-name}@ {host} –确认命令如下 SHOW GRANTS FOR 'user';</code>
2	Aurora MySQL	3.02.2	my.cnf 配置 <code>log-bin=mysql-binbinlog-format=rowbinlog-row-image=full</code> –确认命令如下 <code>show variables like '%log_bin%';show variables like '%binlog%';</code>	<code>GRANT REPLICATION SLAVE, REPLICATION CLIENT ON . TO :math:{u ser-name}@ {host} –确认命令如下 SHOW GRANTS FOR 'user';</code>
3	PostgreSQL	9.4 或以上版本	postgresql.conf 配置: <code>wal_level = logical max_wal_senders = 10max_replication_slots = 10max_connections = 600</code>	pg_hba.conf 配置: <code>host replication repl_acct 0.0.0.0/0 md5</code>
4	openGauss	2.0.1 ~ 3.0.0	postgresql.conf 配置: <code>wal_level = logical max_wal_senders = 10max_replication_slots = 10max_connections = 600</code>	pg_hba.conf 配置: <code>host replication repl_acct 0.0.0.0/0 md5</code>

在作业执行过程中，SphereEx-DBPlusEngine 能够根据集群资源情况及进行合理地调度，尽可能提高整体资源利用率，避免资源争夺，从而保作业执行效率。同时也支持任务的自动故障转移，给长时间运行的任务提供必要保障。

基本概念

■ 节点

运行计算层或存储层组件进程的实例，可以是物理机、虚拟机和容器等。

■ 集群

为了提供特定服务而集合在一起的多个节点。

■ 源端

原始数据所在的存储集群。

■ 目标端

原始数据将要迁移的目标存储集群。

■ 数据迁移作业

把数据从某一个存储集群复制到另一个存储集群的完整流程。

■ 存量数据

在数据迁移作业开始前，数据节点中已有的数据。

■ 增量数据

在数据迁移作业执行过程中，业务系统所产生的新数据。

适用场景

单体数据库存储容量或连接数不足的情况，满足同构数据库由单体迁移拆分至分片环境的场景。

如一个应用系统中的传统单体数据库，单表记录数过亿且还在快速增长，单体数据库负载持续在高位，成为系统瓶颈，需要对数据库进行迁移拆分。

\	源端 -> 目标端	全量迁移	增量迁移	数据校验
1	MySQL -> MySQL	支持	支持	支持
2	PostgreSQL -> PostgreSQL	支持	支持	支持
3	openGauss -> openGauss	支持	支持	支持
4	Oracle -> MySQL	支持	不支持	支持
5	Oracle -> PostgreSQL	支持	不支持	支持
6	Aurora -> Aurora	支持	支持	支持

使用前提

- 已完成 SphereEx-DBPlusEngine 和目标端数据库环境部署，网络通畅且服务正常。
- 迁移作业流程中需要依赖 Zookeeper，需提前完成部署。
- 在增量追数据阶段，请务必确保数据库日志不能被清除。
- 扩所容后需要对数据进行一致性校验，该阶段需要窗口时间，需提前规划。

使用限制

支持项

- 支持迁移含整型唯一键或字符串唯一键的表、无主键唯一键的表、多字段主键的表（首字段字符型）、多字段唯一键的表（首字段整型）及 MySQL 二进制主键表（VARBINARY）。
- 支持迁移含整型主键或字符串主键的表。
- 迁移过程中，可支持 emoji 数据的迁移，MySQL 需设置 utf8mb4 字符集，PostgreSQL 需设置 utf-8 字符集。

不支持项

- 不支持在当前存储节点之上做迁移，需要准备一个全新的数据库集群作为迁移目标库。

注意事项

在存量和增量数据迁移的阶段，可在线进行传输，对业务无影响。而流量切换阶段，需要一定窗口时间，来确保源端和目标端数据已追平，该阶段需要短暂的停止业务。

7.9.2 原理介绍

SphereEx-DBPlusEngine 数据迁移通过 4 个阶段来完成单体库到分片的数据迁移：准备阶段、存量数据迁移阶段、增量数据同步阶段及流量切换阶段。

1. 准备阶段在准备阶段，数据迁移模块会进行数据源连通性及权限的校验，同时进行存量数据的统计、日志位点的记录，最后根据数据量和用户设置的并行度，对任务进行分片。
2. 存量数据迁移阶段

执行在准备阶段拆分好的存量数据迁移任务，存量迁移阶段采用 JDBC 查询的方式，直接从源端读取数据，基于配置的分片等规则写入到目标端。

3. 增量数据同步阶段

由于存量数据迁移耗费的时间受到数据量和并行度等因素影响，此时需要对这段时间内业务新增的数据进行同步。不同的数据库使用的技术细节不同，但总体上均为基于复制协议或 WAL 日志实现的变更数据捕获功能。

- MySQL：订阅并解析 binlog；

- PostgreSQL：采用官方逻辑复制 test_decoding。

这些捕获的增量数据，同样会由数据迁移模块写入到新数据节点中。当增量数据基本同步完成时（由于业务系统未停止，增量数据是不断的），则进入流量切换阶段。

4. 流量切换阶段

在此阶段，可能存在一定时间的业务只读窗口期，通过 SphereEx-DBPlusEngine 的停写功能或者设置数据库只读、控制源头写流量等方式，让源端数据节点中的数据短暂静态，确保增量同步完全完成。

这个只读窗口期时长取决于用户是否需要对数据进行一致性校验以及数据量。一致性校验是独立的任务，支持单独启停，支持断点续传。

确认完成后，数据迁移完成。然后用户可以把读流量或者写流量切换到 SphereEx-DBPlusEngine。

7.9.3 使用指南

1. 制定分片策略

在数据迁移前，需要结合业务制定合理的分片策略，相关内容请参考[分片插件](#)。

2. 优化数据迁移配置（可选）

SphereEx-DBPlusEngine 提供了数据迁移的优化配置，可对数据抽取及数据写入的线程进行控制。在资源充沛的情况下，可最大限度使用资源，高效完成迁移作业。同时，为了尽量减少对业务系统的影响，也可对迁移流量进行限制。

3. 数据校验（可选）

为了确保数据的一致性，可通过数据校验功能进行确认，该过程通常在窗口时间进行。同一时间只允许一个运行一个一致性校验作业，用户可以对一致性校验进行查询、中断、重启和删除。

4. 切换元数据

确认目标环境具备上线条件后，可触发集群流量切换，完成数据迁移。

7.9.4 操作指南

数据迁移拆分的关键操作步骤如下所示，为确保两端数据一致性，切换业务前需要一定窗口时间。具体操作命令请参考[数据迁移 DisSQL](#)。

1. 迁移规划，确认目标端数据库架构，以及分片数量
2. 在 Proxy 中构建集群，注册目标端数据库
3. 在 Proxy 中配置分片规则
4. 优化迁移配置（可选）
5. 注册源端数据源
6. 启动数据迁移
7. 确认迁移状态
8. 停止业务
9. 执行数据校验
10. 修改业务 url
11. 切换数据源后恢复业务

7.9.5 配置示例

本示例为通过 SphereEx-DBPlusEngine 将 MySQL 单体库迁移到 2 库 4 分片场景的操作演示。具体操作命令请参考[数据迁移 DisSQL](#)。

环境说明

\	实例	IP 地址	服务端口	主机名	备注
1 2	DBPl usEngine 1.3.0 Z	192.16 8.xx.102 192.16 8.xx.102	3307 2181	dbpl usengine dbpl	
3 4	ookeeper 3.6.3 MySQL 5.7.26	192.16 8.xx.103 192.16 8.xx.104	3306 3306	usengine mysql_0	
5	MySQL 5.7.26 MySQL 5.7.26	192.16 8.xx.105	3306	mysql_1 mysql_2	

拓扑图



配置过程

1. 准备源端环境

创建源端数据库，建表后插入若干条测试数据。

```
mysql -utest -p -h192.168.xx.106
mysql> CREATE DATABASE product;
mysql> SHOW DATABASES;
mysql> USE product;

mysql> CREATE TABLE `t_user` (
  `user_id` int NOT NULL,
  `order_id` int NOT NULL,
  `status` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`user_id`)
);

mysql> INSERT INTO t_user VALUES
(1,1,'active'),
(2,2,'active'),
(3,3,'active'),
(4,4,'active');

mysql> SELECT * FROM t_user ORDER BY user_id;
+-----+-----+-----+
| user_id | order_id | status |
+-----+-----+-----+
| 1 | 1 | active |
| 2 | 2 | active |
| 3 | 3 | active |
| 4 | 4 | active |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

2. 准备目标端环境

在目标端中分别创建 product_1 和 product_2 数据库。

```
mysql -utest -p -h192.168.xx.104
mysql> CREATE DATABASE product_1;
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| product_1 |
| sys |
+-----+
5 rows in set (0.00 sec)

mysql -utest -p -h192.168.xx.105
mysql> CREATE DATABASE product_2;
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| product_2 |
| sys |
+-----+
```

```
+-----+
5 rows in set (0.00 sec)
```

3. 创建逻辑库

创建逻辑库 sharding_db。

```
# mysql -uroot -p -P3307
mysql> CREATE DATABASE sharding_db;
mysql> SHOW DATABASES;
```

在完成注册目标端数据源后，创建分片规则。

```
mysql> USE sharding_db;

mysql> REGISTER STORAGE UNIT ds_1 (
  URL="jdbc:mysql://192.168.xx.104:3306/product_1?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123",
  PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
), ds_2 (
  URL="jdbc:mysql://192.168.xx.105:3306/product_2?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123",
  PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
);
Query OK, 0 rows affected (0.79 sec)

mysql> SHOW STORAGE UNITS\G

mysql> CREATE SHARDING TABLE RULE t_user(
  STORAGE_UNITS(ds_1,ds_2),
  SHARDING_COLUMN=user_id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="4")),
  KEY_GENERATE_STRATEGY(COLUMN=user_id,TYPE(NAME="snowflake"))
);
Query OK, 0 rows affected (18.00 sec)

mysql> SHOW SHARDING TABLE RULE t_user;
```

4. 注册源端数据源

通过 REGISTER MIGRATION SOURCE STORAGE UNIT 命令注册源端数据源。

```
mysql> REGISTER MIGRATION SOURCE STORAGE UNIT ds_0 (
  URL="jdbc:mysql://192.168.xx.103:3306/product?serverTimezone=UTC&useSSL=false",
  USER="test",
  PASSWORD="Test@123",
  PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
);
Query OK, 0 rows affected (0.15 sec)
```

5. 启动数据迁移

当源端和目标端环境准备好以后，即可开始启动数据迁移。

```
mysql> MIGRATE TABLE ds_0.t_user INTO t_user;
Query OK, 0 rows affected (1.01 sec)
```

如需同时迁移多张表，可使用逗号分隔多张表的名称，格式如下。

```
MIGRATE TABLE ds_0.t_user INTO sharding_db.t_user,ds_0.t_user1 INTO sharding_db.t_user1,ds_0.t_t_user2 INTO sharding_db.t_user2;
```

迁移开始后，可通过 SHOW 命令查看迁移列表。

```
mysql> SHOW MIGRATION LIST;
```

id	tables	sharding_total_count	active	create_time	stop_time
j010171e761478594bdab280c55014650e5f6	t_user	1	true	2022-09-16 15:11:34	NULL

1 row in set (0.75 sec)

根据列表中迁移 ID 的内容，进一步查看该迁移任务的状态。

```
mysql> SHOW MIGRATION STATUS 'j010171e761478594bdab280c55014650e5f6';
```

item	data_source	status	active	inventory_finished_percentage	incremental_idle_seconds
0		PREPARING	true	0	0

1 row in set (0.16 sec)

```
mysql> SHOW MIGRATION STATUS 'j010171e761478594bdab280c55014650e5f6';
```

item	data_source	status	active	inventory_finished_percentage	incremental_idle_seconds
0	ds_0	EXECUTE_INCREMENTAL_TASK	true	100	0

1 row in set (0.06 sec)

当状态为 EXECUTE_INCREMENTAL_TASK 时，可停业务对数据进行一致性校验。

6. 停止业务，避免新数据写入源端

7. 数据一致性校验

SphereEx-DBPlusEngine 可对源端和目标端的数据一致性进行校验，也包括两端记录数比对。比对过程需花费一定时间，具体量化指标可参考性能白皮书-数据比对的内容。

```
mysql> CHECK MIGRATION 'j010171e761478594bdab280c55014650e5f6' BY TYPE (NAME='CRC32_MATCH');
Query OK, 0 rows affected (0.08 sec)
```

目标端开启数据加密或异构迁移的情况需要使用 DATA_MATCH，查看数据校验算法的方法如下。

```
mysql> SHOW MIGRATION CHECK ALGORITHMS;
```

type	supported_database_types	description
CRC32_MATCH	MySQL	Match CRC32 of records.
DATA_MATCH	SQL92,MySQL,MariaDB,PostgreSQL,openGauss,Oracle,SQLServer,H2,DM,Aurora	Match raw data of records.
SphereEx:DATA_CONSISTENCY_CHECK_CRC32_MATCH	MySQL	Match CRC32 of records

3 rows in set (0.01 sec)

为进一步验证数据迁移的情况，可在物理库中对分片表依次确认。

■ product_1

```
mysql> USE product_1;
mysql> SHOW TABLES;
```

Tables_in_product_1
t_user_0
t_user_2

2 rows in set (0.00 sec)

```
mysql> SELECT * FROM t_user_0;
```

```
+-----+-----+-----+
| user_id | order_id | status |
+-----+-----+-----+
| 4 | 4 | active |
+-----+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM t_user_2;
```

```
+-----+-----+-----+
| user_id | order_id | status |
+-----+-----+-----+
| 2 | 2 | active |
+-----+-----+-----+
1 row in set (0.00 sec)
```

■ product_2

```
mysql> USE product_2;
```

```
mysql> SHOW TABLES;
```

```
+-----+
| Tables_in_product_2 |
+-----+
| t_user_1 |
| t_user_3 |
+-----+
```

```
2 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM t_user_1;
```

```
+-----+-----+-----+
| user_id | order_id | status |
+-----+-----+-----+
| 1 | 1 | active |
+-----+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM t_user_3;
```

```
+-----+-----+-----+
| user_id | order_id | status |
+-----+-----+-----+
| 3 | 3 | active |
+-----+-----+-----+
1 row in set (0.00 sec)
```

8. 修改业务 url

讲 url 地址变更为 Proxy 的地址，应用程序将访问目标端环境。

9. 切换数据源

完成迁移作业。

```
mysql> COMMIT MIGRATION 'j010171e761478594bdab280c55014650e5f6';
Query OK, 0 rows affected (0.57 sec)
```

刷新元数据，上线新数据源。

```
mysql> REFRESH TABLE METADATA;
Query OK, 0 rows affected (0.21 sec)
```

10. 恢复业务

7.9.6 FAQ

- 1. 单表到分片表，是否可能存在分片键重复的问题？
不会发生。
- 2. 索引是否一同迁移过来？如果有自增字段，是否会存在问题？
没有问题。索引会和表结构一同迁移过来。自增字段不影响数据，所有数据会一块迁移，包括自增字段。

7.10 弹性伸缩

随着业务快速发展，分布式数据库方案可能将再次面临数据存储容量的问题，因此，弹性扩缩容是分布式架构不可或缺的能力。

7.10.1 概述

SphereEx-DBPlusEngine 弹性伸缩插件可以实现同构存储节点在线地扩容或缩容，同时支持新旧环境的数据比对。
目前 SphereEx-DBPlusEngine 弹性伸缩插件所支持的数据库产品为 MySQL、PostgreSQL 和 openGauss，具体版本、环境要求和权限要求如下。

	数据库	版本支持	环境要求	权限要求
1	MySQL	5.1.15 ~ 8.x	my.cnf 配置 l og-bin=my sql-binbinlog-format=rowbin log-row-i mage=full –确认命令如下 show variables like ‘%log_b in%’ ;show variables like ‘% binlog%’ ;	GRANT RE PLICATION SLAVE, RE PLICA-TION CLIENT ON . TO :math: ‘{u sername}@{host}’ –确认命令如下 SHOW GRANTS FOR ‘user’ ;
2	PostgreSQL	9.4 或以上版本	postgres ql.conf 配置: wal_level = logical max_wal_senders = 10max _replicat ion_slots = 10max_co nnections = 600	pg_h ba.conf 配置: host re plication repl_acct 0.0.0.0/0 md5
3	openGauss	2.0.1 ~ 3.0.0	postgres ql.conf 配置: wal_level = logical max_wal_senders = 10max _replicat ion_slots = 10max_co nnections = 600	pg_h ba.conf 配置: host re plication repl_acct 0.0.0.0/0 md5

在作业执行过程中，SphereEx-DBPlusEngine 能够根据集群资源情况及进行合理地调度，尽可能提高整体资源利用率，避免资源争夺，从而保作业执行效率。同时也支持任务的自动故障转移，给长时间运行的任务提供必要保障。

基本概念

- 节点
指运行计算层或存储层进程的实例，可以是物理机、虚拟机、容器等。
- 集群
为了提供特定服务而集合在一起的多个节点。
- 数据迁移作业
把数据从某一个集群迁移到另一个集群的完整流程。
- 存量数据
在数据迁移作业开始前，数据节点中已存在的数据。
- 增量数据

在数据迁移作业执行过程中，业务系统所产生的新数据。

适用场景

- 存储节点扩容，应对存储节点容量或连接数不足的场景；
- 存储节点缩容，减少存储节点、提升资源利用率的场景。

使用前提

- 已完成 SphereEx-DBPlusEngine 和新增数据库节点部署，网络通畅且服务正常。
- 扩缩容作业流程中需要依赖 Zookeeper，需提前完成部署。
- 当前已使用 SphereEx-DBPlusEngine 数据分片架构，且扩缩容节点与当前存储节点为同构、同版本数据库。
- 因扩所容过程中存储节点会保留两版本分片数据，请务必确保物理存储空间为当前数据文件的两倍以上。
- 在追加增量数据阶段，请务必确保数据库日志不能被清除。
- 扩所容后需要对数据进行一致性校验，该阶段需要窗口时间，需提前规划。

使用限制

支持项

- 支持操作含整型唯一键或字符串唯一键的表；
- 支持操作含整型主键或字符串主键的表。

不支持项

- 不支持操作无主键且无唯一键的表；
- 不支持复合主键或复合唯一键的表。

注意事项

在存量和增量数据迁移的阶段，可在线进行数据传输，对业务无影响。而流量切换阶段，需要一定窗口时间，来确保源端和目标端数据已追平，该阶段需要短暂的停止业务。

7.10.2 原理介绍

分片算法和使用方式满足条件的情况下，可以做到高效扩缩容，不需要移动数据或者只移动少量数据。

与数据迁移插件原理类似，SphereEx-DBPlusEngine 弹性伸缩插件同样是通过 4 个阶段：准备阶段、存量数据迁移阶段、增量数据同步阶段及流量切换阶段。

不同的是，弹性伸缩是原地的扩容，即流量切换前的阶段，原节点会存储两份数据。

1. 准备阶段

在准备阶段，数据迁移模块会进行数据源连通性及权限的校验，同时进行存量数据的统计、日志位点的记录，最后根据数据量和用户设置的并行度，对任务进行分片。

2. 存量数据迁移阶段

执行在准备阶段拆分好的存量数据迁移任务，存量迁移阶段采用 JDBC 查询的方式，直接从源端读取数据，基于配置的分片等规则写入到目标端。

3. 增量数据同步阶段

由于存量数据迁移耗费的时间受到数据量和并行度等因素影响，此时需要对这段时间内业务新增的数据进行同步。不同的数据库使用的技术细节不同，但总体上均为基于复制协议或 WAL 日志实现的变更数据捕获功能。

- MySQL：订阅并解析 binlog；
- PostgreSQL：采用官方逻辑复制 test_decoding。

这些捕获的增量数据，同样会由数据迁移模块写入到新数据节点中。当增量数据基本同步完成时（由于业务系统未停止，增量数据是不断的），则进入流量切换阶段。

4. 流量切换阶段

在此阶段，可能存在一定时间的业务只读窗口期，通过 SphereEx-DBPlusEngine 的停写功能或者设置数据库只读、控制源头写流量等方式，让源端数据节点中的数据短暂静态，确保增量同步完全完成。

这个只读窗口期时长取决于用户是否需要对数据进行一致性校验以及数据量。一致性校验是独立的任务，支持单独启停，支持断点续传。

确认完成后，数据迁移完成。然后用户可以把读流量或者写流量切换到 SphereEx-DBPlusEngine。

7.10.3 使用指南

1. 制定分片策略

在数据迁移前，需要结合业务制定合理的分片策略，相关内容请参考[数据分片](#)。

2. 优化数据迁移配置（可选）

SphereEx-DBPlusEngine 提供了数据迁移的优化配置，可对数据抽取及数据写入的线程进行控制。在资源充沛的情况下，可最大限度使用资源，高效完成迁移作业。同时，为了尽量减少对业务系统的影响，也可对迁移流量进行限制。

3. 数据校验（可选）

为了确保数据的一致性，可通过数据校验功能进行确认，该过程通常在窗口时间进行。同一时间只允许一个运行一个一致性校验作业，用户可以对一致性校验进行查询、中断、重启和删除。

4. 切换元数据

确认目标环境具备上线条件后，可触发集群流量切换，完成数据迁移。

数据移动说明

\	算法	扩容是否会移动数据
1	Range	否
2	List	是
3	Mod	是
4	Hash_mod	是

7.10.4 操作指南

具体操作命令请参考[弹性伸缩 DistSQL](#)。

1. 存储空间确认

在扩缩容过程中，数据库中会存在两个版本的数据，因此请确保各个节点存储空间为当前数据文件的两倍以上，为扩缩容期间两版本数据做预留空间准备。

2. 当前分片确认

需确认操作前分片规则的信息，可使用 SHOW SHARDING TABLE NODES 或 PREVIEW 命令进行验证。

3. 优化数据迁移配置（可选）

SphereEx-DBPlusEngine 提供了数据迁移的优化配置，可对数据抽取及数据写入的线程进行控制。在资源充沛的情况下，可最大限度使用资源，高效完成作业。同时，为了尽量减少对业务系统的影响，也可对迁移流量进行限制。

4. 数据校验（可选）

为了确保数据的一致性，可通过数据校验功能进行确认，该过程通常在窗口时间进行。

5. 切换流量

确认目标环境具备上线条件后，可触发集群流量切换，完成扩缩容。

7.10.5 MySQL 使用手册

具体操作命令请参考弹性伸缩 DistSQL。

环境要求

支持的 MySQL 版本：5.1.15 ~ 8.0.x。

权限要求

1. 开启 binlog。

MySQL 5.7 my.cnf 示例配置：

```
[mysqld]
server-id=1
log-bin=mysql-bin
binlog-format=row
binlog-row-image=full
max_connections=600
```

执行以下命令，确认是否有开启 binlog：

```
show variables like '%log_bin%';
show variables like '%binlog%';
```

如以下显示，则说明 binlog 已开启

Variable_name	Value
log_bin	ON
binlog_format	ROW
binlog_row_image	FULL

2. 赋予 MySQL 账号 Replication 相关权限。

执行以下命令，查看该用户是否有迁移权限：

```
SHOW GRANTS FOR 'user';
```

示例结果：

Grants for \${username}@\${host}
GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO \${username}@\${host}
.....

完整流程示例

前提条件

1. 在 MySQL 中创建好数据库。

示例：

```
DROP DATABASE IF EXISTS resharding_ds_0;  
CREATE DATABASE resharding_ds_0 DEFAULT CHARSET utf8;
```

```
DROP DATABASE IF EXISTS resharding_ds_1;  
CREATE DATABASE resharding_ds_1 DEFAULT CHARSET utf8;
```

```
DROP DATABASE IF EXISTS resharding_ds_2;  
CREATE DATABASE resharding_ds_2 DEFAULT CHARSET utf8;
```

操作步骤

1. 在 proxy 新建逻辑数据库并配置好资源和规则，建表并初始化一些数据。

```
CREATE DATABASE sharding_db;  
  
USE sharding_db;  
  
REGISTER STORAGE UNIT ds_0 (  
  URL="jdbc:mysql://127.0.0.1:3306/resharding_ds_0?useServerPrepStmts=true&serverTimezone=UTC&useSSL=false&  
characterEncoding=utf-8",  
  USER="root",  
  PASSWORD="root"  
) , ds_1 (  
  URL="jdbc:mysql://127.0.0.1:3306/resharding_ds_1?useServerPrepStmts=true&serverTimezone=UTC&useSSL=false&  
characterEncoding=utf-8",  
  USER="root",  
  PASSWORD="root"  
);  
  
CREATE SHARDING TABLE RULE t_order(  
  STORAGE_UNITS(ds_0,ds_1),  
  SHARDING_COLUMN=order_id,  
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="4")),  
  KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))  
);  
  
CREATE TABLE t_order (order_id INT NOT NULL, user_id INT NOT NULL, status VARCHAR(45) NULL, PRIMARY KEY (order_id));  
  
INSERT INTO t_order (order_id, user_id, status) VALUES (1,2,'ok'),(2,4,'ok'),(3,6,'ok'),(4,1,'ok'),(5,3,'ok'),(6,5,'ok');
```

2. 在 Proxy 中添加新数据源。

示例：

```
REGISTER STORAGE UNIT ds_2 (  
  URL="jdbc:mysql://127.0.0.1:3306/resharding_ds_2?useServerPrepStmts=true&serverTimezone=UTC&useSSL=false&  
characterEncoding=utf-8",  
  USER="root",  
  PASSWORD="root"  
);
```

3. 启动扩缩容。

新增了 ds_2 数据源，示例：

```
RESHARD TABLE t_order BY(
  STORAGE_UNITS(ds_0, ds_1, ds_2),
  SHARDING_COLUMN=order_id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="6")),
  KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);
```

4. 查看扩缩容作业列表。

```
SHOW RESHARDING LIST;
```

示例结果：

id	tables	sharding_total_count	active	create_time	stop_time
j51017f973ac82cb1edea4f5238a258c25e89	t_order	2	true	2022-10-25 10:10:58	NULL

5. 查看扩缩容详情。

```
SHOW RESHARDING STATUS 'j51017f973ac82cb1edea4f5238a258c25e89';
```

示例结果：

item	data_source	status	active	processed_records_count	inventory_finished_percentage	remaining_seconds	incremental_idle_seconds	error_message
0	ds_0	EXECUTE_INCREMENTAL_TASK	true	3	100	0	92	
1	ds_1	EXECUTE_INCREMENTAL_TASK	true	3	100	0	92	

6. 执行停写（可选）。

该语句会拦截增删改和部分 DistSQL，是可选的。

```
STOP RESHARDING SOURCE WRITING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

7. 执行数据一致性校验。

```
CHECK RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89' BY TYPE (NAME='CRC32_MATCH');
```

数据一致性校验算法类型来自：

```
SHOW RESHARDING CHECK ALGORITHMS;
```

type	supported_database_types	description
CRC32_MATCH	MySQL	Match CRC32 of records.
DATA_MATCH	SQL92,MySQL,MariaDB,PostgreSQL,openGauss,Oracle,SQLServer,H2	Match raw data of records.

开启数据加密的情况需要使用 DATA_MATCH。

查询数据一致性校验进度：

```
SHOW RESHARDING CHECK STATUS 'j51017f973ac82cb1edea4f5238a258c25e89';
```

示例结果：

tables	result	finished_percentage	remaining_seconds	check_begin_time	check_end_time	duration_seconds	error_message
t_order	true	100	0	2022-10-25 10:13:33.220	2022-10-25 10:13:35.338	2	

8. 切换元数据。

```
APPLY RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

预览生效后的规则

```
PREVIEW SELECT * FROM t_order;
```

示例结果：

data_source_name	actual_sql
ds_0	select * from v1_t_order_0 UNION ALL select * from v1_t_order_3
ds_1	select * from v1_t_order_1 UNION ALL select * from v1_t_order_4
ds_2	select * from v1_t_order_2 UNION ALL select * from v1_t_order_5

9. 恢复停写（可选）。

如果在之前执行了停写，需要进行恢复。

```
RESTORE RESHARDING SOURCE WRITING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

10. 完成扩缩容作业。

示例：

```
COMMIT RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

单表扩容为分片表

1. MySQL 准备工作

```
DROP DATABASE IF EXISTS resharding_ds_0;
CREATE DATABASE resharding_ds_0 DEFAULT CHARSET utf8;
```

```
DROP DATABASE IF EXISTS resharding_ds_1;
CREATE DATABASE resharding_ds_1 DEFAULT CHARSET utf8;
```

```
DROP DATABASE IF EXISTS resharding_ds_2;
CREATE DATABASE resharding_ds_2 DEFAULT CHARSET utf8;
```

```
USE resharding_ds_0
```

```
CREATE TABLE t_order (
  order_id INT NOT NULL,
  user_id INT NOT NULL,
  status VARCHAR(45) NULL,
  PRIMARY KEY (order_id));
```

```
INSERT INTO t_order (order_id, user_id, status) VALUES
(1,2,'ok'),
(2,4,'ok'),
(3,6,'ok'),
(4,1,'ok'),
```

```
(5,3,'ok'),
(6,5,'ok');
```

2. 创建逻辑库，添加数据库源

```
DROP DATABASE IF EXISTS sharding_db;
```

```
CREATE DATABASE sharding_db;
```

```
USE sharding_db;
```

```
REGISTER STORAGE UNIT ds_0 (
  URL="jdbc:mysql://192.168.xx.103:13306/resharding_ds_0?useServerPrepStmts=true&serverTimezone=UTC&useSSL=false&
  characterEncoding=utf-8",
  USER="test",
  PASSWORD="Test@123"
), ds_1 (
  URL="jdbc:mysql://192.168.xx.104:13306/resharding_ds_1?useServerPrepStmts=true&serverTimezone=UTC&useSSL=false&
  characterEncoding=utf-8",
  USER="test",
  PASSWORD="Test@123"
);
```

3. 创建分片规则

```
CREATE SHARDING TABLE RULE no_use(
  STORAGE_UNITS(ds_0,ds_1),
  SHARDING_COLUMN=order_id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="4")),
  KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);
```

4. 添加新的数据源

```
REGISTER STORAGE UNIT ds_2 (
  URL="jdbc:mysql://192.168.xx.105:13306/resharding_ds_2?useServerPrepStmts=true&serverTimezone=UTC&useSSL=false&
  characterEncoding=utf-8",
  USER="test",
  PASSWORD="Test@123"
);
```

5. 检查单表 t_order

```
mysql> show sharding table nodes t_order;
ERROR 20000 (44000): Can not find table rule with logic tables `[t_order]`.
```

```
mysql> preview select * from t_order;
```

```
+-----+-----+
| data_source_name | actual_sql |
+-----+-----+
| ds_0            | select * from t_order |
+-----+-----+
1 row in set (0.21 sec)
```

6. 启动扩容任务

```
RESHARD TABLE t_order BY(
  STORAGE_UNITS(ds_0, ds_1, ds_2),
  SHARDING_COLUMN=order_id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="6")),
  KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);
```

7. 查看任务状态，任务完成后提交

```

SHOW RESHARDING LIST;

SHOW RESHARDING STATUS 'j51017f973ac82cb1edea4f5238a258c25e89';

CHECK RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89' BY TYPE (NAME='CRC32_MATCH');

SHOW RESHARDING CHECK STATUS 'j51017f973ac82cb1edea4f5238a258c25e89';

APPLY RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';

COMMIT RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';

```

8. 确认扩容结果

```

mysql> SHOW SHARDING TABLE NODES t_order;
+-----+-----+
| name | nodes |
+-----+-----+
| t_order | ds_0.t_order_0, ds_1.t_order_1, ds_2.t_order_2, ds_0.t_order_3, ds_1.t_order_4, ds_2.t_order_5 |
+-----+-----+
1 row in set (0.00 sec)

```

自动扩容

1. 在 MySQL 中准备相应的数据源
2. 在 SphereEx-DBPlusEngine 中创建好逻辑库，及规则

```

CREATE DATABASE sharding_db;

USE sharding_db;

REGISTER STORAGE UNIT ds_0 (
  URL="jdbc:mysql://127.0.0.1:3306/resharding_ds_0?useServerPrepStmts=true&serverTimezone=UTC&useSSL=false&characterEncoding=utf-8",
  USER="root",
  PASSWORD="root"
), ds_1 (
  URL="jdbc:mysql://127.0.0.1:3306/resharding_ds_1?useServerPrepStmts=true&serverTimezone=UTC&useSSL=false&characterEncoding=utf-8",
  USER="root",
  PASSWORD="root"
);

CREATE SHARDING TABLE RULE t_order(
  STORAGE_UNITS(ds_0,ds_1),
  SHARDING_COLUMN=order_id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="4")),
  KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);

CREATE TABLE t_order (order_id INT NOT NULL, user_id INT NOT NULL, status VARCHAR(45) NULL, PRIMARY KEY (order_id));

INSERT INTO t_order (order_id, user_id, status) VALUES (1,2,'ok'),(2,4,'ok'),(3,6,'ok'),(4,1,'ok'),(5,3,'ok'),(6,5,'ok');

```

3. 确认当前分片

```

SHOW SHARDING TABLE NODES t_order;
+-----+-----+
| name | nodes |
+-----+-----+

```

```
| t_order | ds_0.t_order_0, ds_1.t_order_1, ds_0.t_order_2, ds_1.t_order_3 |
+-----+-----+
```

4. 添加新的数据源

```
REGISTER STORAGE UNIT ds_2 (
  URL="jdbc:mysql://127.0.0.1:3306/resharding_ds_2?useServerPrepStmts=true&serverTimezone=UTC&useSSL=false&
characterEncoding=utf-8",
  USER="root",
  PASSWORD="root"
);
```

5. 启动扩容

```
AUTO RESHARD TABLE t_order BY(
  STORAGE_UNITS(ds_0, ds_1, ds_2),
  SHARDING_COLUMN=order_id,
  TYPE(NAME="hash_mod", PROPERTIES("sharding-count"="6")),
  KEY_GENERATE_STRATEGY(COLUMN=order_id, TYPE(NAME="snowflake"))
);
```

6. 查看任务详情

```
SHOW RESHARDING LIST;
```

```
SHOW RESHARDING STATUS 'j51017f973ac82cb1edea4f5238a258c25e89';
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
| item | data_source | status          | active | processed_records_count | inventory_finished_percentage | remaining_seconds |
incremental_idle_seconds | error_message |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
| 0 | ds_0 | EXECUTE_INCREMENTAL_TASK | true | 3 | 100 | 0 | 1 | | |
| 1 | ds_1 | EXECUTE_INCREMENTAL_TASK | true | 3 | 100 | 0 | 1 | | |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
2 rows in set (0.04 sec)
```

```
-- 如果查询不到该作业，则作业已经完成，执行下一步即可
```

7. 检查分片生效情况

```
SHOW SHARDING TABLE NODES t_order;
```

```
+-----+-----+-----+-----+-----+-----+-----+
| name | nodes |
+-----+-----+
| t_order | ds_0.v1_t_order_0, ds_1.v1_t_order_1, ds_2.v1_t_order_2, ds_0.v1_t_order_3, ds_1.v1_t_order_4, ds_2.v1_t_order_5 |
+-----+-----+
1 row in set (0.00 sec)
```

至此，完成扩容。

7.10.6 PostgreSQL 使用手册

具体操作命令请参考弹性伸缩 DistSQL。

环境要求

支持的 PostgreSQL 版本：9.4 或以上版本。

权限要求

1. 开启 `test_decoding`。
2. 调整 WAL 配置。

postgresql.conf 示例配置：

```
wal_level = logical
max_wal_senders = 10
max_replication_slots = 10
max_connections = 600
```

详情请参见 [Write Ahead Log](#) 和 [Replication](#)。

3. 配置 PostgreSQL 允许 Proxy 拥有 replication 权限。

pg_hba.conf 示例配置：

```
host replication repl_acct 0.0.0.0/0 md5
```

详情请参见 [The pg_hba.conf File](#)。

完整流程示例

前提条件

1. 在 PostgreSQL 中创建好数据库。

示例：

```
DROP DATABASE IF EXISTS resharding_ds_0;
CREATE DATABASE resharding_ds_0;

DROP DATABASE IF EXISTS resharding_ds_1;
CREATE DATABASE resharding_ds_1;

DROP DATABASE IF EXISTS resharding_ds_2;
CREATE DATABASE resharding_ds_2;
```


操作步骤

1. 在 proxy 新建逻辑数据库并配置好资源和规则，建表并初始化一些数据。

```
CREATE DATABASE sharding_db;

\c sharding_db;

REGISTER STORAGE UNIT ds_0 (
  URL="jdbc:postgresql://127.0.0.1:5432/resharding_ds_0",
  USER="root",
  PASSWORD="root"
), ds_1 (
  URL="jdbc:postgresql://127.0.0.1:5432/resharding_ds_1",
  USER="root",
  PASSWORD="root"
);

CREATE SHARDING TABLE RULE t_order(
  STORAGE_UNITS(ds_0,ds_1),
  SHARDING_COLUMN=order_id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="4")),
  KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);

CREATE TABLE t_order (order_id INT NOT NULL, user_id INT NOT NULL, status VARCHAR(45) NULL, PRIMARY KEY (order_id));

INSERT INTO t_order (order_id, user_id, status) VALUES (1,2,'ok'),(2,4,'ok'),(3,6,'ok'),(4,1,'ok'),(5,3,'ok'),(6,5,'ok');
```

2. 在 Proxy 中添加新数据源。

示例：

```
REGISTER STORAGE UNIT ds_2 (
  URL="jdbc:postgresql://127.0.0.1:5432/resharding_ds_2",
  USER="root",
  PASSWORD="root"
);
```

3. 启动扩缩容。

新增了 ds_2 数据源，示例：

```
RESHARD TABLE t_order BY(
  STORAGE_UNITS(ds_0, ds_1, ds_2),
  SHARDING_COLUMN=order_id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="6")),
  KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);
```

4. 查看扩缩容作业列表。

```
SHOW RESHARDING LIST;
```

示例结果：

id	tables	sharding_total_count	active	create_time	stop_time
j51017f973ac82cb1edea4f5238a258c25e89	t_order	2	true	2022-10-25 10:10:58	NULL

5. 查看扩缩容详情。

```
SHOW RESHARDING STATUS 'j51017f973ac82cb1edea4f5238a258c25e89';
```

示例结果：

item	data_source	status	active	processed_records_count	inventory_finished_percentage	remaining_seconds	incremental_idle_seconds	error_message
0	ds_0	EXECUTE_INCREMENTAL_TASK	true	3	100	0	92	
1	ds_1	EXECUTE_INCREMENTAL_TASK	true	3	100	0	92	

6. 执行停写（可选）。

该语句会拦截增删改和部分 DistSQL，是可选的。

```
STOP RESHARDING SOURCE WRITING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

7. 执行数据一致性校验。

```
CHECK RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

查询数据一致性校验进度：

```
SHOW RESHARDING CHECK STATUS 'j51017f973ac82cb1edea4f5238a258c25e89';
```

示例结果：

tables	result	finished_percentage	remaining_seconds	check_begin_time	check_end_time	duration_seconds	error_message
t_order	true	100	0	2022-10-25 10:13:33.220	2022-10-25 10:13:35.338	2	

8. 切换元数据。

```
APPLY RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

预览生效后的规则：

```
PREVIEW SELECT * FROM t_order;
```

示例结果：

data_source_name	actual_sql
ds_0	select * from v1_t_order_0 UNION ALL select * from v1_t_order_3
ds_1	select * from v1_t_order_1 UNION ALL select * from v1_t_order_4
ds_2	select * from v1_t_order_2 UNION ALL select * from v1_t_order_5

9. 恢复停写（可选）

如果在之前执行了停写，需要进行恢复。

```
RESTORE RESHARDING SOURCE WRITING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

10. 完成扩缩容作业。

示例：

```
COMMIT RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

7.10.7 openGauss 使用手册

具体操作命令请参考弹性伸缩 DistSQL。

环境要求

支持的 openGauss 版本：2.0.1 ~ 3.0.0。

权限要求

1. 调整 WAL 配置。

postgresql.conf 示例配置：

```
wal_level = logical  
max_wal_senders = 10  
max_replication_slots = 10  
wal_sender_timeout = 0  
max_connections = 600
```

详情请参见 [Write Ahead Log](#) 和 [Replication](#)。

2. 配置 PostgreSQL 允许 Proxy 拥有 replication 权限。

pg_hba.conf 示例配置：

```
host replication repl_acct 0.0.0.0/0 md5
```

详情请参见 [Configuring Client Access Authentication](#) 和 [Example: Logic Replication Code](#)。

完整流程示例

前提条件

1. 在 openGauss 中创建好数据库。

示例：

```
DROP DATABASE IF EXISTS resharding_ds_0;  
CREATE DATABASE resharding_ds_0;  
  
DROP DATABASE IF EXISTS resharding_ds_1;  
CREATE DATABASE resharding_ds_1;  
  
DROP DATABASE IF EXISTS resharding_ds_2;  
CREATE DATABASE resharding_ds_2;
```

操作步骤

1. 在 proxy 新建逻辑数据库并配置好资源和规则，建表并初始化一些数据。

```
CREATE DATABASE sharding_db;

USE sharding_db;

REGISTER STORAGE UNIT ds_0 (
  URL="jdbc:opengauss://127.0.0.1:5432/resharding_ds_0",
  USER="root",
  PASSWORD="root"
), ds_1 (
  URL="jdbc:opengauss://127.0.0.1:5432/resharding_ds_1",
  USER="root",
  PASSWORD="root"
);

CREATE SHARDING TABLE RULE t_order(
  STORAGE_UNITS(ds_0,ds_1),
  SHARDING_COLUMN=order_id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="4")),
  KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);

CREATE TABLE t_order (order_id INT NOT NULL, user_id INT NOT NULL, status VARCHAR(45) NULL, PRIMARY KEY (order_id));

INSERT INTO t_order (order_id, user_id, status) VALUES (1,2,'ok'),(2,4,'ok'),(3,6,'ok'),(4,1,'ok'),(5,3,'ok'),(6,5,'ok');
```

2. 在 Proxy 中添加新数据源。

示例：

```
REGISTER STORAGE UNIT ds_2 (
  URL="jdbc:opengauss://127.0.0.1:5432/resharding_ds_2",
  USER="root",
  PASSWORD="root"
);
```

3. 启动扩缩容。

新增了 ds_2 数据源，示例：

```
RESHARD TABLE t_order BY(
  STORAGE_UNITS(ds_0, ds_1, ds_2),
  SHARDING_COLUMN=order_id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="6")),
  KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);
```

4. 查看扩缩容作业列表。

```
SHOW RESHARDING LIST;
```

示例结果：

id	tables	sharding_total_count	active	create_time	stop_time
j51017f973ac82cb1edea4f5238a258c25e89	t_order	2	true	2022-10-25 10:10:58	NULL

5. 查看扩缩容详情。

```
SHOW RESHARDING STATUS 'j51017f973ac82cb1edea4f5238a258c25e89';
```

示例结果：

item	data_source	status	active	processed_records_count	inventory_finished_percentage	remaining_seconds	incremental_idle_seconds	error_message
0	ds_0	EXECUTE_INCREMENTAL_TASK	true	3	100	0	92	
1	ds_1	EXECUTE_INCREMENTAL_TASK	true	3	100	0	92	

6. 执行停写（可选）。

该语句会拦截增删改和部分 DistSQL，是可选的。

```
STOP RESHARDING SOURCE WRITING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

7. 执行数据一致性校验。

```
CHECK RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

查询数据一致性校验进度：

```
SHOW RESHARDING CHECK STATUS 'j51017f973ac82cb1edea4f5238a258c25e89';
```

示例结果：

tables	result	finished_percentage	remaining_seconds	check_begin_time	check_end_time	duration_seconds	error_message
t_order	true	100	0	2022-10-25 10:13:33.220	2022-10-25 10:13:35.338	2	

8. 切换元数据。

```
APPLY RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

预览生效后的规则：

```
PREVIEW SELECT * FROM t_order;
```

示例结果：

data_source_name	actual_sql
ds_0	select * from v1_t_order_0 UNION ALL select * from v1_t_order_3
ds_1	select * from v1_t_order_1 UNION ALL select * from v1_t_order_4
ds_2	select * from v1_t_order_2 UNION ALL select * from v1_t_order_5

9. 恢复停写（可选）。

如果在之前执行了停写，需要进行恢复。

```
RESTORE RESHARDING SOURCE WRITING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

10. 完成扩缩容作业。

示例：

```
COMMIT RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

7.10.8 FAQ

1. 在迁移中，分片键、分片算法及参数、分片数量及唯一值生成策略是否可以修改？
可以修改，修改后即移动对应的数据。
2. 如果有多张需要扩容的表，是否可以支持并行扩容？
支持。

8.1 SQL 概述

SphereEx-DBPlusEngine 是一款分布式数据库增强计算引擎，其下层可对接多种异构数据库。它天然高度兼容已完成对接的异构数据库语法，并在其基础上提供自有语法 SQL 支持，满足更多对集群开发、管理等方面需求。

8.1.1 SQL 种类

SphereEx-DBPlusEngine 支持几类 SQL：

- Database SQL

SphereEx-DBPlusEngine 支持多种异构数据库，并支持这些数据库上的原生 SQL。目前针对不同数据库支持的 SQL 程度不多，具体参见[SQL 兼容性](#)部分。

- Client SQL

SphereEx-DBPlusEngine 支持部分 SQL 客户端的语句。例如 MySQL 数据库的 MySQL Client 或 PostgreSQL 数据库的 PSQL Client 等。这些命令行工具，内置方便管理使用的 SQL，如 MySQL Client 中的 show tables、PSQL 中的等 `\d`。

- DistSQL

SphereEx-DBPlusEngine 支持部分扩展 SQL，命名为 DistSQL。主要目的是为了方便对 SphereEx-DBPlusEngine 自身进行管理、维护操作的。具体参见 [DistSQL](#) 部分。

8.2 优化器

SphereEx-DBPlusEngine 内置了两种优化器：分片下推引擎和联邦查询引擎。前者是在语句可准备命中分片的情况下进行工作，后者则可针对更为广泛的场景。联邦查询引擎对关联查询、子查询等复杂查询进行优化，同时支持跨多个数据库实例的分布式查询，内部使用关系代数优化查询计划，通过最优计划查询出结果。目前联邦查询优化引擎属于试用功能，虽基本功能可用，但仍需不断完善。

8.2.1 启用联邦查询引擎

联邦查询引擎，缺省是不启动的，需要修改参数才能启动。

```
sql-federation-enabled = true
```

8.2.2 可使用联邦查询场景

联邦查询引擎，主要是工作在不能准确命中分片的情况，下面通过示例进行说明。

■ 子查询

子查询和外层查询未同时指定分片键，或分片键的值不一致时，由联邦查询引擎提供支持。

```
SELECT * FROM (SELECT * FROM t_order) o;  
SELECT * FROM (SELECT * FROM t_order) o WHERE o.order_id = 1;  
SELECT * FROM (SELECT * FROM t_order WHERE order_id = 1) o;  
SELECT * FROM (SELECT * FROM t_order WHERE order_id = 1) o WHERE o.order_id = 2;
```

■ 跨库关联查询

当关联查询中的多个表分布在不同的数据库实例上时，由联邦查询引擎提供支持。假设 t_order 和 t_order_item 是多数据节点的分片表，并且未配置绑定表规则，t_user 和 t_user_role 是分布在不同的数据库实例上的单表，那么联邦查询引擎能够支持如下常用的关联查询：

```
SELECT * FROM t_order o INNER JOIN t_order_item i ON o.order_id = i.order_id WHERE o.order_id = 1;  
SELECT * FROM t_order o INNER JOIN t_user u ON o.user_id = u.user_id WHERE o.user_id = 1;  
SELECT * FROM t_order o LEFT JOIN t_user_role r ON o.user_id = r.user_id WHERE o.user_id = 1;  
SELECT * FROM t_order_item i LEFT JOIN t_user u ON i.user_id = u.user_id WHERE i.user_id = 1;  
SELECT * FROM t_order_item i RIGHT JOIN t_user_role r ON i.user_id = r.user_id WHERE i.user_id = 1;  
SELECT * FROM t_user u RIGHT JOIN t_user_role r ON u.user_id = r.user_id WHERE u.user_id = 1;
```

8.3 开发约束

SphereEx-DBPlusEngine 兼容对接数据库语法，但由于分布式架构和单机存在较大的架构差异，存在 SQL 使用限制。本文将介绍数据库开发过程中的使用限制。

8.3.1 标识符限制

针对各类对象名称的限制，主要取决于底层对接数据库的限制，在 SphereEx-DBPlusEngine 端未做任何限制，可以定义任何长度的标识符。当提交的标识符下发到底层数据库上执行失败时，会将错误反馈给用户。

8.3.2 资源限制

针对资源的创建，如 Database、Table 等，SphereEx-DBPlusEngine 自身未对这部分进行限制。上述创建的资源对应的元数据将保存在治理中心及 SphereEx-DBPlusEngine 本地的内存中。如内存空间不足且元数据过多的情况下，可能会出现 OOM 问题。

8.3.3 语法限制

SphereEx-DBPlusEngine 兼容大部分数据库语法，具体参见[SQL 兼容性](#)。

语法类别	语法子类	使用约束
DDL	TABLE	支持
	VIEW	支持（限分片内）
	TRIGGER	支持（限分片内）
	PROCEDURE	支持（限分片内）
	CURSOR	支持（限分片内）
DML	INSERT	支持（部分受限）
	DELETE	支持（部分受限）
	UPDATE	支持（部分受限）
	MERGE	支持（限分片内）
	REPLACE	支持
DQL	JOIN ON	支持
	LEFT/RIGHT/FULL JOIN	支持
	SEMI JOIN	支持
	ANTI JOIN	支持
	GROUP	支持
	GROUP... HAVING	支持
	SUBQUERY	支持
	UNION/UNION ALL	支持
	INTERSECT/MINUS	支持
	SUBQUERY	支持（限分片内）

8.4 连接管理

8.4.1 登录认证

Proxy 拥有严格的登录认证机制，只有经过身份认证的用户才可以成功建立连接。

概念

为保障用户数据和分布式配置信息的安全，DBPlusEngine-Proxy 提供了用户认证的功能，且该功能不可关闭，否则，未经认证的客户端连接将被拒绝。当前，DBPlusEngine-Proxy 已经支持了多种用户认证协议，包括：

- MySQL 客户端：old password、native、clear-text、sha256
- PostgreSQL 客户端：scram-sha-256、MD5、password
- openGauss 客户端：scram-sha-256、md5

DBPlusEngine-Proxy 为各种数据库协议设置了默认认证方式，且用户不能修改。其中：MySQL 使用 native，PG 使用 md5，OG 使用 scram-sha-256。为方便企业用户进行统一身份管理，DBPlusEngine 也提供了 LDAP (Lightweight Directory Access Protocol) 认证方式，LDAP 认证已支持 MySQL 和 PostgreSQL 客户端。下面是配置用户使用不同的认证方式示例。

```
- !AUTHORITY
users:
- user: root@%
  password: root
  authenticationMethodName: md5
- user: sharding
  password: sharding
authenticators:
md5:
  type: MD5
scram_sha256:
  type: SCRAM_SHA256
defaultAuthenticator: scram_sha256
```

特点

实际上，用户在使用 DBPlusEngine-Proxy 时不需要考虑选择何种协议，协议的协商过程是在 DBPlusEngine 与客户端之间自动完成的。

- 当使用 MySQL 客户端时，默认使用 mysql_native_password。仅当用户需要进行 LDAP 认证时，DBPlusEngine 会要求客户端切换为 mysql_clear_password 协议进行通信。
- 当使用 PostgreSQL 客户端时，默认使用 MD5。仅当用户需要进行 LDAP 认证时，DBPlusEngine 会要求客户端切换为 password 协议进行通信。

密码认证

默认情况下，Proxy 使用密码认证方式，登录用户需提供正确的用户名和密码。

特别的，由于 Proxy 支持多种数据库协议（如 MySQL、PostgreSQL 等），当用户应用不同的数据库客户端时，Proxy 能够自动适配密码通信协议，在复杂场景下为用户提供一致的安全体验。

主机限制

管理员可以为 Proxy 的用户限制登录主机地址，以提高安全等级。如：

```
- !AUTHORITY
users:
- user: root@127.0.0.1
password: root
```

以上配置指定了 root 用户只能从 127.0.0.1 这个地址访问 Proxy，从其他地址登录时，即使密码正确也将被拒绝。

LDAP 认证

为方便企业用户进行统一认证管理，Proxy 也提供了 LDAP (Lightweight Directory Access Protocol) 认证方式，LDAP 认证现已支持 MySQL 和 PostgreSQL 客户端。同时，Proxy 允许用户以非常灵活的方式来接入 LDAP，例如：

- 1. 可以配置为默认使用 LDAP，这样所有用户都通过 LDAP 认证；
- 2. 支持为用户配置 auth 属性，指定用户使用密码或 LDAP 认证，各个用户可以使用不同方式；
- 3. 各个用户可以使用不同的 LDAP 认证器，即对接不同的 LDAP 服务；
- 4. 支持为用户指定 DN 模板，满足复杂场景的需要；
- 5. 支持 LDAPS 协议，可进一步提升安全等级。

8.4.2 连接配置

SphereEx-DBPlusEngine 支持 MySQL 或 PostgreSQL 方式连接，用户可通过标准数据库的方式进行连接，可使用包括 JDBC、ODBC 等多种方式进行连接。

8.4.3 Proxy + LDAP & LDAPS 应用案例

背景

SphereEx-DBPlusEngine 增加了对 LDAP 登录认证的支持，以下应用案例展示了 LDAP 登录认证的使用流程。在案例展示过程中，使用 Wireshark 工具进行抓包，以更形象的展示 LDAP 与 LDAPS 协议的区别。LDAPS 是基于 SSL/TLS 的 LDAP 通信方式。

基础环境

名称	版本
MySQL	5.7+
SphereEx-DBPlusEngine	1.0+
Wireshark	3.6
ApacheDS	2.0.0

■ config-sharding-databases.yaml

```
schemaName: sharding_db

dataSources:
ds_0:
url: jdbc:mysql://127.0.0.1:3306/demo_ds_0?serverTimezone=UTC&useSSL=false
username: root
password:
```

```

connectionTimeoutMilliseconds: 30000
idleTimeoutMilliseconds: 60000
maxLifetimeMilliseconds: 1800000
maxPoolSize: 10
minPoolSize: 1
ds_1:
url: jdbc:mysql://127.0.0.1:3306/demo_ds_1?serverTimezone=UTC&useSSL=false
username: root
password:
connectionTimeoutMilliseconds: 30000
idleTimeoutMilliseconds: 60000
maxLifetimeMilliseconds: 1800000
maxPoolSize: 10
minPoolSize: 1
rules:
- !SHARDING
tables:
t_order:
actualDataNodes: ds_${0..1}.t_order
keyGenerateStrategy:
column: order_id
keyGeneratorName: snowflake
defaultDatabaseStrategy:
standard:
shardingColumn: user_id
shardingAlgorithmName: database_inline
defaultTableStrategy:
none:
shardingAlgorithms:
database_inline:
type: INLINE
props:
algorithm-expression: ds_${user_id % 2}
keyGenerators:
snowflake:
type: SNOWFLAKE

```

LDAP 服务器配置

采用 ApacheDS™ 的 Docker 镜像：tremolosecurity/apacheds。

a. 拉取镜像

```
docker pull tremolosecurity/apacheds:latest
```

b. 生成 SSL 证书

说明页：<https://directory.apache.org/apacheds/basic-ug/3.3-enabling-ssl.html>

按照页面中的方法，使用 keytool 生成证书。

这里以 Common name 为 localhost 为例，生成两个文件：

```

.
├── localhost.ks
└── localhost.cer

```

由于使用的容器对文件名有特殊要求，这里将 localhost.ks 重命名为 apacheds.jks。

```

.
├── apacheds.jks
└── localhost.cer

```

说明：证书文件可以保存在任意路径，如：

- /Users/\${yourname}/apacheds/apacheds.jks
- /Users/\${yourname}/apacheds/localhost.cer

c. 启动容器

```
docker run --detach --rm --name apacheds \
-p 10389:10389 \
-p 10636:10636 \
-v /Users/${yourname}/apacheds:/etc/apacheds \
-e APACHEDS_ROOT_PASSWORD=secret \
-e APACHEDS_TLS_KS_PWD=secret \
tremolosecurity/apacheds:latest
```

说明：

- 该容器映射出两个端口，其中 10389 用作 ldap 非加密连接，10636 用作 ldaps 加密连接。
- ApacheDS 服务中包含一个默认用户 uid=admin,ou=system，通过 APACHEDS_ROOT_PASSWORD 参数指定了其密码为 secret。

启动操作后，查看日志是否正常：

```
docker logs -f apacheds
```

d. ldapsearch 测试

ldapsearch 命令可以便捷的对 LDAP 服务发起访问，验证 LDAP 服务是否正常：

```
docker exec -it apacheds ldapsearch -x -H ldap://localhost:10389 -b ou=system -D "uid=admin,ou=system" -w secret
```

至此，LDAP 服务器配置完成。

JDK 导入证书

由于 LDAP 服务端使用的是自签名证书，在客户端访问前需要先导入 JRE 的 keystore 中。注意，导入过程需要输入该证书的密钥：secret。

```
keytool -import -alias localhost -keystore $JAVA_HOME/jre/lib/security/cacerts -file /Users/${yourname}/apacheds/localhost.cer
```

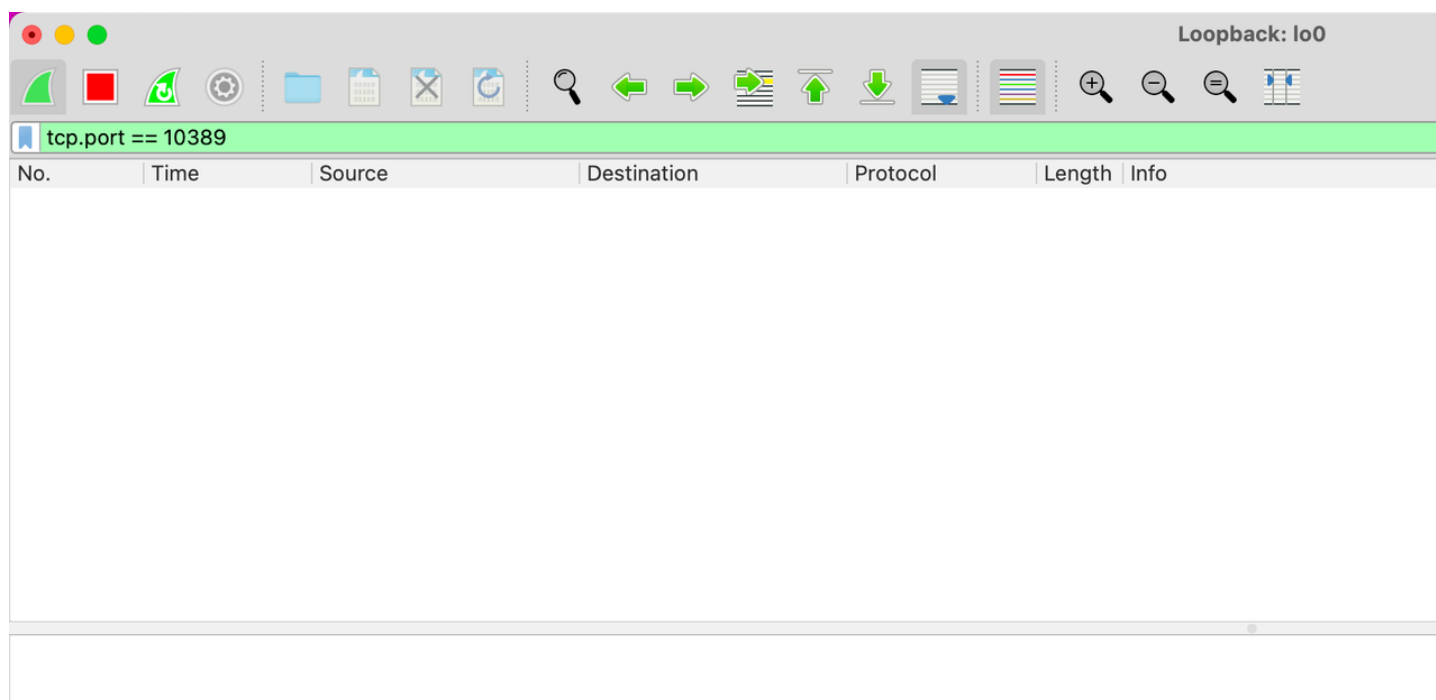
Proxy-LDAP 测试

a. server.yaml

```
authority:
users:
- user: root@%
- user: admin
- user: sharding
authenticators:
auth_ldap:
type: LDAP
props:
ldap_server_url: ldap://localhost:10389
ldap_dn_template: uid={0},ou=system
defaultAuthenticator: auth_ldap
```

b. 启动 Proxy。

c. 启动 Wireshark，开始抓取 10389 端口的数据包。



d. MySQL 客户端登录测试

注意指定`--enable-cleartext-plugin` 参数

由于 LDAP 服务器中只有一个 `admin` 用户，这里使用 `admin` 进行登录。

如果尝试其他用户，得到结果是登录失败。

```
mysql -h 127.0.0.1 -P 3307 -A -u admin -p --enable-cleartext-plugin
```

输入密码 `secret` 后登录成功：

```
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.7.36-ShardingSphere-Proxy 5.1.1-SNAPSHOT-dirty-ff0eb2a MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

e. 查看抓包数据

从抓取的 TCP 数据包中，我们可以轻松的找到一条包含用户 DN 和密码的报文：

tcp.port == 10389

No.	Time	Source	Destination	Protocol	Length	Info
18054	460.630615	127.0.0.1	127.0.0.1	TCP	68	50801 → 10389 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=
18055	460.630668	127.0.0.1	127.0.0.1	TCP	68	10389 → 50801 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MS
18056	460.630675	127.0.0.1	127.0.0.1	TCP	56	50801 → 10389 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=
18057	460.630680	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 10389 → 50801 [ACK] Seq=1 Ack=1 Win=
18058	460.632056	127.0.0.1	127.0.0.1	TCP	95	50801 → 10389 [PSH, ACK] Seq=1 Ack=1 Win=408256 Len=39
18059	460.632069	127.0.0.1	127.0.0.1	TCP	56	10389 → 50801 [ACK] Seq=1 Ack=40 Win=408256 Len=0 TSval=
18066	461.252802	127.0.0.1	127.0.0.1	TCP	70	10389 → 50801 [PSH, ACK] Seq=1 Ack=40 Win=408256 Len=14
18067	461.252848	127.0.0.1	127.0.0.1	TCP	56	50801 → 10389 [ACK] Seq=40 Ack=15 Win=408256 Len=0 TSva
18068	461.258041	127.0.0.1	127.0.0.1	TCP	63	50801 → 10389 [PSH, ACK] Seq=40 Ack=15 Win=408256 Len=7
18069	461.258052	127.0.0.1	127.0.0.1	TCP	56	10389 → 50801 [ACK] Seq=15 Ack=47 Win=408192 Len=0 TSva
18070	461.258447	127.0.0.1	127.0.0.1	TCP	56	50801 → 10389 [FIN, ACK] Seq=47 Ack=15 Win=408256 Len=0
18071	461.258459	127.0.0.1	127.0.0.1	TCP	56	10389 → 50801 [ACK] Seq=15 Ack=48 Win=408192 Len=0 TSva
18076	461.264930	127.0.0.1	127.0.0.1	TCP	56	10389 → 50801 [FIN, ACK] Seq=15 Ack=48 Win=408192 Len=0
18077	461.264957	127.0.0.1	127.0.0.1	TCP	56	50801 → 10389 [ACK] Seq=48 Ack=16 Win=408256 Len=0 TSva

> Frame 18058: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface lo0, id 0

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 50801, Dst Port: 10389, Seq: 1, Ack: 1, Len: 39

> Data (39 bytes)

```

0000 02 00 00 00 45 00 00 5b 00 00 40 00 40 06 00 00  ....E..[ .@.@...
0010 7f 00 00 01 7f 00 00 01 c6 71 28 95 ea ff 5f c7  ....q(.....
0020 8c f1 e9 b7 80 18 18 eb fe 4f 00 00 01 01 08 0a  ....0.....
0030 1d ae 0f 37 28 47 73 a7 30 25 02 01 01 60 20 02  ...7(Gs. 0%...`
0040 01 03 04 13 75 69 64 3d 61 64 6d 69 6e 2c 6f 75  ....uid= admin,ou
0050 3d 73 79 73 74 65 6d 80 06 73 65 63 72 65 74  =system·secret

```

f. 小结

从 server.yaml 配置可以看出，在已有 LDAP 服务器的前提下，使用 LDAP 进行登录认证并不复杂，只需要简单配置即可。另一方面，由于 LDAP 协议是未经加密的，在公开网络中使用 LDAP 认证有密码泄露的风险。

Proxy-LDAPS 测试

a. server.yaml

与 LDAP 案例的区别仅仅是更换了 LDAP 服务器的 URL。

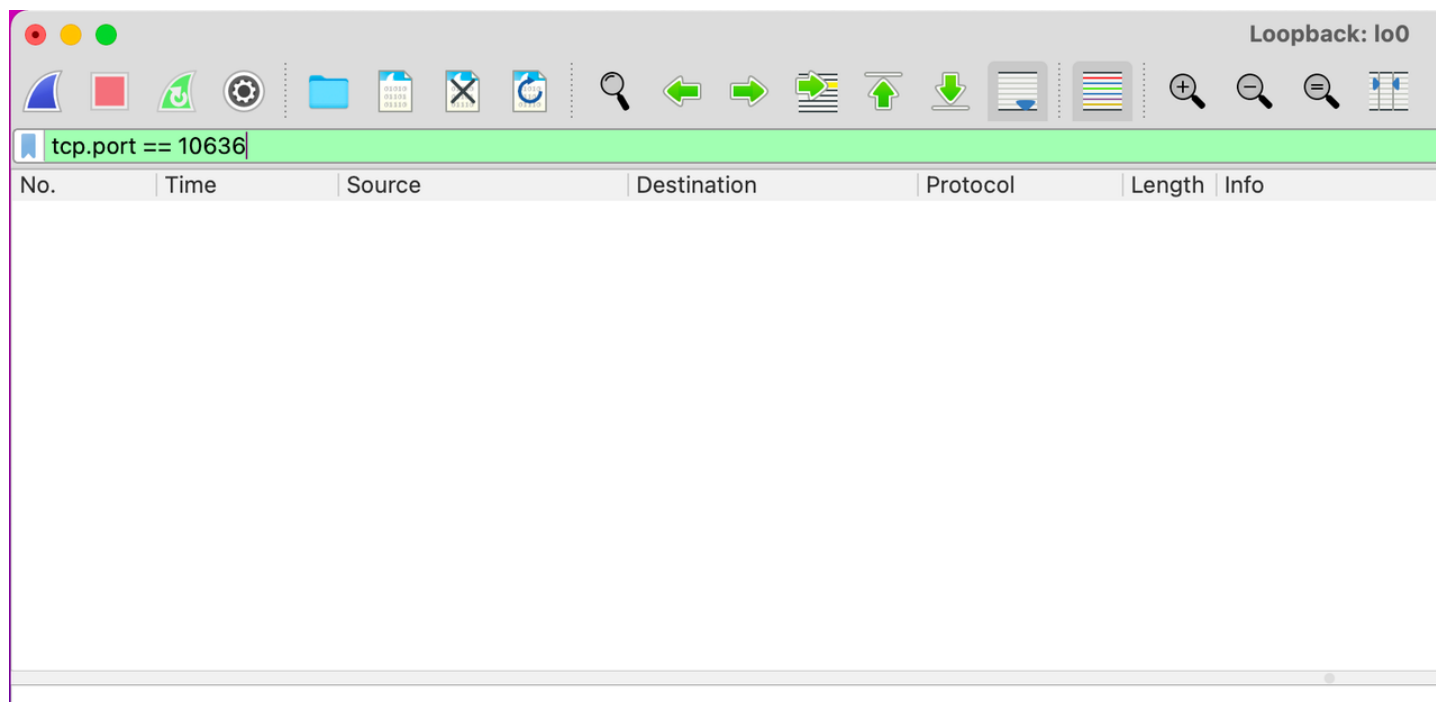
```

authority:
users:
- user: root@%
- user: admin
- user: sharding
authenticators:
auth_ldap:
type: LDAP
props:
ldap_server_url: ldaps://localhost:10636
ldap_dn_template: uid={0},ou=system
defaultAuthenticator: auth_ldap

```

b. 启动 Proxy

c. 启动 Wireshark，开始抓取 10636 端口的数据包



d. MySQL 客户端登录测试

注意指定`--enable-cleartext-plugin` 参数

由于 LDAP 服务器中只有一个 `admin` 用户，这里使用 `admin` 进行登录。

如果尝试其他用户，得到结果是登录失败。

```
mysql -h 127.0.0.1 -P 3307 -A -u admin -p --enable-cleartext-plugin
```

输入密码 `secret` 后登录成功：

```
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.7.36-ShardingSphere-Proxy 5.1.1-SNAPSHOT-dirty-ff0eb2a MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> █
```

e. 查看抓包数据从数据包可以看到，Proxy 与 LDAP 服务器之间建立了 TLS 通信，已无法通过抓包获取通信的内容：

tcp.port == 10636						
No.	Time	Source	Destination	Protocol	Length	Info
11283	236.951053	127.0.0.1	127.0.0.1	TCP	68	58899 → 10636 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=
11284	236.951129	127.0.0.1	127.0.0.1	TCP	68	10636 → 58899 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MS
11285	236.951140	127.0.0.1	127.0.0.1	TCP	56	58899 → 10636 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=
11286	236.951148	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 10636 → 58899 [ACK] Seq=1 Ack=1 Win
11287	236.983838	127.0.0.1	127.0.0.1	TLSv1.2	263	Client Hello
11288	236.983855	127.0.0.1	127.0.0.1	TCP	56	10636 → 58899 [ACK] Seq=1 Ack=208 Win=408064 Len=0 TSva
11291	237.511922	127.0.0.1	127.0.0.1	TLSv1.2	1300	Server Hello, Certificate, Server Key Exchange, Server
11292	237.511951	127.0.0.1	127.0.0.1	TCP	56	58899 → 10636 [ACK] Seq=208 Ack=1245 Win=407040 Len=0 T
11293	237.527550	127.0.0.1	127.0.0.1	TLSv1.2	131	Client Key Exchange
11294	237.527569	127.0.0.1	127.0.0.1	TCP	56	10636 → 58899 [ACK] Seq=1245 Ack=283 Win=408000 Len=0 T
11295	237.534346	127.0.0.1	127.0.0.1	TLSv1.2	62	Change Cipher Spec
11296	237.534361	127.0.0.1	127.0.0.1	TCP	56	10636 → 58899 [ACK] Seq=1245 Ack=289 Win=408000 Len=0 T
11297	237.551167	127.0.0.1	127.0.0.1	TLSv1.2	157	Encrypted Handshake Message
11298	237.551234	127.0.0.1	127.0.0.1	TCP	56	10636 → 58899 [ACK] Seq=1245 Ack=390 Win=407872 Len=0 T
11299	237.823653	127.0.0.1	127.0.0.1	TLSv1.2	163	Change Cipher Spec, Encrypted Handshake Message
11300	237.823669	127.0.0.1	127.0.0.1	TCP	56	58899 → 10636 [ACK] Seq=390 Ack=1352 Win=406912 Len=0 T
11301	237.825056	127.0.0.1	127.0.0.1	TLSv1.2	173	Application Data
11302	237.825067	127.0.0.1	127.0.0.1	TCP	56	10636 → 58899 [ACK] Seq=1352 Ack=507 Win=407744 Len=0 T
11303	237.944497	127.0.0.1	127.0.0.1	TLSv1.2	141	Application Data
11304	237.944529	127.0.0.1	127.0.0.1	TCP	56	58899 → 10636 [ACK] Seq=507 Ack=1437 Win=406848 Len=0 T
> Frame 11297: 157 bytes on wire (1256 bits), 157 bytes captured (1256 bits) on interface lo0, id 0 > Null/Loopback > Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 > Transmission Control Protocol, Src Port: 58899, Dst Port: 10636, Seq: 289, Ack: 1245, Len: 101 > Transport Layer Security						
0000	02 00 00 00 45 00 00 99	00 00 40 00 40 06 00 00	E... ..@..@..			
0010	7f 00 00 01 7f 00 00 01	e6 13 29 8c 5b 8f c2 93	).{...			
0020	aa b8 70 1a 80 18 18 d8	fe 8d 00 00 01 01 08 0a	..p.....			
0030	50 95 25 a2 51 f9 f7 b0	16 03 03 00 60 36 9c 1a	P-%Q.....`6..			
0040	dc e6 c1 ec 98 c7 a2 ce	4c 70 d0 19 81 dd e6 08	 Lp.....			
0050	61 10 98 39 82 e2 9e 1d	d9 b9 6c 57 a8 14 82 7f	a.9.....lW....			
0060	e3 64 c3 3d e9 ba 22 8d	87 16 84 c8 27 aa 6d 1c	.d=...". m.			
0070	56 93 16 bf fc 53 2b 7b	22 df 50 c0 1c 3d 47 fc	V.....S+{ ".P.=G.			
0080	74 0a 68 c8 37 4b 32 f5	ba b0 cf 4c 05 56 63 f5	t.h.7K2. L.Vc.			
0090	71 9f 95 b3 3b f1 22 fa	15 fd 1f 9f e1	q...;".			

f. 小结

通过 SSL/TLS 加密，可以有效的保护用户登录信息。而 SphereEx-DBPlusEngine 对 LDAPS 的支持非常友好，仅需导入证书和替换 URL 就可以完成 LDAP 到 LDAPS 的切换。

8.4.4 其他问题

■ 是否可使用 SSL 连接？

这分为两种情况，一是针对 SphereEx-DBPlusEngine 与底层数据库的连接；而是前端应用于 SphereEx-DBPlusEngine 的连接。针对前者，主要依赖于对应 SphereEx-DBPlusEngine 内置的 JDBC 驱动情况而定，针对 MySQL、PostgreSQL 都支持使用 SSL 方式进行连接。针对后者，SphereEx-DBPlusEngine 暂不支持 SSL 方式连接。

■ 是否支持连接池配置？

SphereEx-DBPlusEngine 支持主流的数据库连接池，如：DBCP, C3P0, BoneCP, HikariCP 等。在具体配置时，需要关注连接池上限，以满足可能的高并发场景需求。

■ 是否支持负载均衡配置？

SphereEx-DBPlusEngine 针对底层数据库提供读能力的负载均衡，具体参见读写分离部分。针对前端应用与 SphereEx-DBPlus Engine 连接，理论上支持任何四层负载均衡作为 SphereEx-DBPlusEngine 的负载均衡。

■ 是否支持通过 VIP 方式接入 SphereEx-DBPlusEngine？

SphereEx-DBPlusEngine 自身不提供 VIP 能力，可通过第三方服务完成。通过将 SphereEx-DBPlusEngine 作为标准服务来看待，通过 VIP 绑定来提升整体可用性。

8.5 流量管理

随着数据规模的不断膨胀，使用多节点集群的分布式方式逐渐成为趋势。对集群整体视角的统一管理能力，和针对单独组件细粒度的控制能力，是基于存算分离的现代数据库体系中不可或缺的功能。管控的挑战之一，在于对集群的集中化管理的统一管理能力以及在单点出现故障时精细化的操作能力。

集中化管理的挑战体现在将包括数据库存储节点和中间件计算节点的状态统一管理，并且能够实时的探测到分布式环境下最新的变动情况，进一步为集群的控制和调度提供依据。面对超负荷的流量下，针对某一节点进行熔断和限流，以保证整个数据库集群得以继续运行，是分布式系统下对单一节点控制能力的挑战。

8.5.1 熔断

阻断节点和数据库的连接。当某个 SphereEx-DBPlusEngine 节点超过负载后时或其他需要停止该节点对数据库的访问时，可通过命令停止对与数据库的链接，使数据库能够保证足够的资源为其他节点提供服务。当需要时，也可恢复状态，重新连接数据库对外提供服务。

相关操作

```
sql> DISABLE COMPUTE NODE IP=127.0.0.1,PORT=3306;  
sql> SHOW COMPUTE NODES;
```

具体参见[启用/禁用 Proxy](#)

8.5.2 限流

面对超负荷的请求开启限流，以保护部分请求可以得以高质量的响应。可通过影子库功能，将部分流量导向其他库，间接起到限流功能。

相关操作

具体参见[影子库](#)

8.5.3 读写分离

读写分离，将根据语句是否对数据修改将访问流量发送到数据库主库或从库。这一方式，可提升数据库的吞吐量，对读多写少的场景非常适合。

相关操作

具体参见[读写分离](#)

8.5.4 智能路由

针对复杂业务场景，可通过混合使用 SphereEx-DBPlusEngine 与 SphereEx-DBPlusEngine-Driver，后者是一种轻量化、编程态产品，支持与用户方 Java 应用部署在一起，可将其视为标准 JDBC Driver 的增强。两者的结合支持将业务访问流量，从 SphereEx-DBPlusEngine-Driver 一端转发到 SphereEx-DBPlusEngine 端处理，后者可充分利用其独享的资源，适合于重度的工作负载。这一功能被称为 Traffic Route。具体参见[智能路由](#)

产品特点

- SphereEx-DBPlus Driver 采用无中心化架构，与应用程序共享资源，适用于 Java 开发的高性能的轻量级 OLTP 应用，但是由于 SphereEx-DBPlus Driver 与应用程序共享资源，当执行消耗资源较多的 SQL 时，应用程序的稳定性和性能会受到影响。同时 SphereEx-DBPlus Driver 消耗的连接数较多，当应用程序和数据库部署在不同的网络分区时，网络延迟对于性能的影响也会更加明显。
- SphereEx-DBPlusEngine 提供统一的静态入口，并且独立于应用程序部署，适用于 OLAP 应用以及对分片数据库进行管理和运维的场景。用户使用 SphereEx-DBPlus Engine 来执行资源消耗较多的 SQL，可以有效避免对应用程序产生影响。为了提升应用程序的查询性能和稳定性，我们可以考虑将 SphereEx-DBPlus Driver 接入端消耗资源较多的 SQL，转发至和数据库位于同一网络区域的 SphereEx-DBPlusEngine 接入端，由 SphereEx-DBPlusEngine 接入端计算好查询结果，再统一返回给应用程序。

部署架构

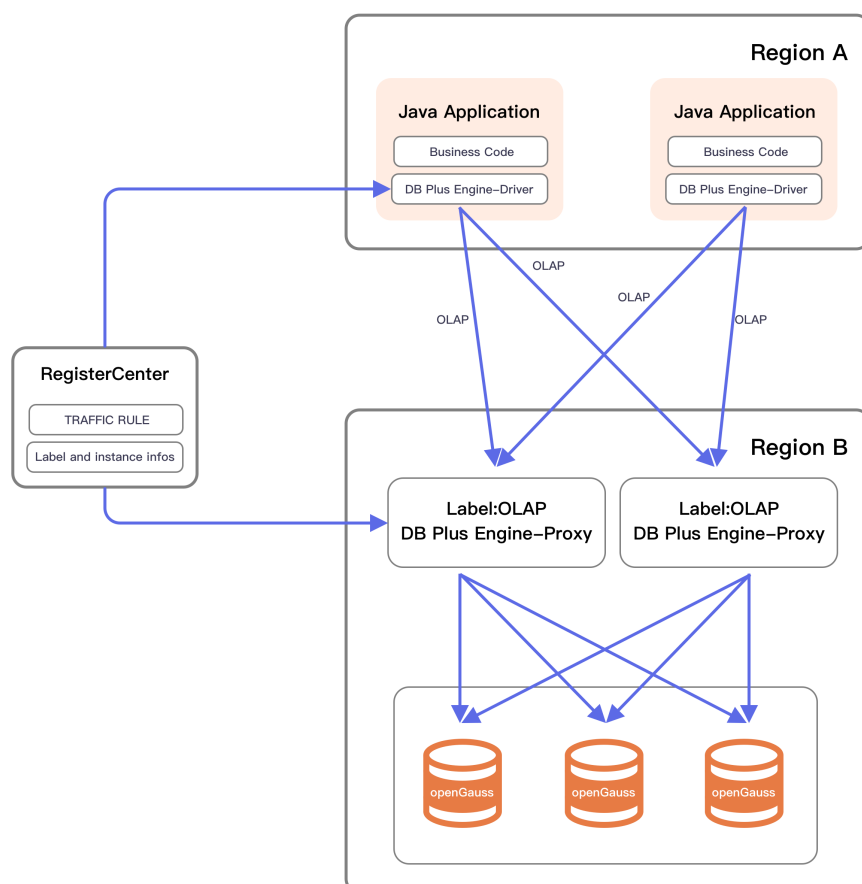


Fig. 1: 部署架构图

相关概念

■ 标签

SphereEx-DBPlusEngine 实例配置的标签属性，用于对实例进行区分。SphereEx-DBPlusEngine 开启 Traffic 功能时，转发的目标就是标签对应的 SphereEx-DBPlusEngine 实例。

■ 转发策略

对 SphereEx-DBPlus Driver 接入端的 SQL 进行转发的策略，包括：目标 Proxy 实例标签、转发算法、负载均衡算法。根据算法的不同，可以将转发策略分为事务转发策略和普通转发策略，当策略中的算法配置为 TransactionTrafficAlgorithm 时，转发策略为事务转发策略，当策略中的算法配置为其他算法时，转发策略为普通转发策略。

■ 转发算法

用于判断当前 SQL 是否需要转发的算法，包括了 HintTrafficAlgorithm、SegmentTrafficAlgorithm 和 TransactionTrafficAlgorithm。

■ 负载均衡算法

通过负载均衡算法将 SQL 语句转发到标签对应的不同 SphereEx-DBPlusEngineProxy 实例上执行，目前支持 RANDOM 和 ROUND_ROBIN 两种负载均衡算法。

使用规范

■ 支持项

- 支持所有常用的 SQL 进行转发。

■ 不支持项

- 用户配置的内核功能不支持的 SQL，转发之后 SQL 仍然不支持。
- 不支持开启事务后将 SQL 转发到不同的接入端或 SphereEx-DBPlusEngine-Proxy 实例执行。

使用方法

虽然通过将 SphereEx-DBPlusEngine-Driver 接入端消耗资源较多的 SQL 转发到 SphereEx-DBPlusEngine-Proxy 执行，可以有效提升应用程序的性能和稳定性，但是这也导致了部署架构更加复杂，用户需要判断哪些语句需要转发到 DBPlusEngine-Proxy，并在 DAO 层开发相关的业务逻辑来控制 SQL 的转发。此外，在开启事务的场景下，将事务中的部分 SQL 转发到 SphereEx-DBPlusEngine-Proxy 执行，会对事务的一致性和可见性带来影响，从而影响业务系统对于事务的使用。

8.5.5 强制路由

数据分片

可以通过 SQL Hint 的方式强制 SQL 在指定分片上运行。该功能的注释格式暂时只支持 /* */，内容需要以 SHARDING-SPHERE_HINT: 开始，可选的属性包括：

- {table}.SHARDING_DATABASE_VALUE：用于添加 {table} 表对应的数据源分片键值，多个属性使用逗号分隔；
- {table}.SHARDING_TABLE_VALUE：用于添加 {table} 表对应的表分片键值，多个属性使用逗号分隔。

分库不分表情况下，强制路由至某一个分库时，可使用 SHARDING_DATABASE_VALUE 方式设置分片，无需指定 {table}。

```
/* SHARDINGSPHERE_HINT: t_order.SHARDING_DATABASE_VALUE=1,t_order.SHARDING_TABLE_VALUE=1 */  
SELECT * FROM t_order;
```

读写分离

可通过 SQL Hint 的方式强制 SQL 运行在主库。SQL Hint 功能的注释格式暂时只支持 `/* */`，内容需要以 `SHARDINGSPHERE_HINT:` 开始，属性名为 `WRITE_ROUTE_ONLY`。

```
/* SHARDINGSPHERE_HINT: WRITE_ROUTE_ONLY=true */
SELECT * FROM t_order;
```

也可通过 SQL Hint 强制 SQL 运行在某个数据源。注释格式暂时只支持 `/* */`，内容需要以 `SHARDINGSPHERE_HINT:` 开始，属性名为 `DATA_SOURCE_NAME`。如果使用 MySQL 客户端连接需要添加 `-c` 选项保留注释，客户端默认是 `-skip-comments` 过滤注释。

```
/* SHARDINGSPHERE_HINT: DATA_SOURCE_NAME=ds_0 */
SELECT * FROM t_order;
```

8.5.6 会话管理

SphereEx-DBPlusEngine 支持会话管理，可通过 DistSQL 查看当前会话或杀掉会话。目前此功能仅限于存储节点为 MySQL 的情况。

相关操作

■ 查看会话

针对不同关联数据库支持不同的查看会话方法，关联 MySQL 数据库可使用 `show processlist` 命令查看会话。SphereEx-DBPlusEngine 会自动生成唯一的 UUID 标识作为 ID，并将 SQL 执行信息存储在各个实例中。当执行此命令时，SphereEx-DBPlusEngine 会通过治理中心收集并同步各个计算节点的 SQL 执行信息，然后汇总返回给用户。

```
mysql> show processlist;
```

Id	User	Host	db	Command	Time	State	Info
3de4ff8d-2b06-abd0-46d0-124857919943	root	127.0.0.1	sharding_db	Execute	6	Executing	0/1 select sleep(100)
f9e5c97431567415fe10badc5fa46378	root	127.0.0.1	sharding_db	Sleep	690		

■ 输出说明

输出模拟原生 MySQL 的输出，但 Id 字段较为特殊为随机字符串。

■ 前提条件

需要打开 `show-process-list-enabled` 参数，具体参见[参数列表](#)。

■ 杀掉会话

用户根据 `show processlist` 返回的结果，判断是否需要执行 `kill` 语句，SphereEx-DBPlusEngine 会根据 `kill` 语句中的 Id 取消正在执行中的 SQL。

```
mysql> kill 3de4ff8d-2b06-abd0-46d0-124857919943;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> show processlist;
Empty set (0.02 sec)
```

8.6 对象管理

SphereEx-DBPlusEngine 按照逻辑库、对象层次进行管理。

8.6.1 逻辑库

逻辑库的概念

SphereEx-DBPlusEngine 中的逻辑库，是管理一组对象的单元。其形式上模拟了数据库的 Database/Schema 的概念。

逻辑库的操作

■ 创建

```
mysql> create database mydb;
```

■ 删除

```
mysql> drop database mydb;
```

注意：删除逻辑库，将导致逻辑库下的相关对象被删除，且无法恢复。

■ 查询

```
mysql> show databases;
```

■ 使用

```
mysql> use mydb;
```

■ 其他约束

- 不支持引用其他 Database 的对象，例如 select * from db2.t1。
- 不支持修改逻辑库名称。
- 不支持设置逻辑库的字符集、排序规则、只读状态、加密状态等。

后置步骤：添加数据源

在完成逻辑库创建后，还不能管理表等对象。在这之前还有一个关键步骤—添加数据源。添加数据源可以理解为为后续对象增加容器，来承载这些对象。关于数据源的操作，可参考 RESOURCE 相关的 DistSQL。

■ 操作示例

```
REGISTER STORAGE UNIT ds_3307 (  
  HOST=127.0.0.1,  
  PORT=3307,  
  DB=test,  
  USER=testuser,  
  PASSWORD=testpwd  
)
```

■ 使用约束

SphereEx-DBPlusEngine 仅支持在一个逻辑库内添加同构的数据源，同构可理解为使用相同通信协议的数据库产品。

8.6.2 对象：表

表，是数据的基本承载主体。SphereEx-DBPlusEngine 通过封装底层数据源内的原始数据表，提供对外标准对象能力，并扩展出多种表类型，可满足不同场景需求。

表概念

名称	可见性	含义
逻辑表	可见	在 SphereEx-DBPlusEngine 对底层原始表的封装，是用户操作的主体，用户可视为标准数据库表使用。
物理表	不可见	由 SphereEx-DBPlusEngine 封装的底层数据库的真实物理表。这类表用户是直接不可访问的。
分片表	可见	由 SphereEx-DBPlusEngine 实现的一种逻辑表，是在数据分片场景中具有相同结构的一组水平拆分数据库（表）的逻辑对象。
广播表	可见	由 SphereEx-DBPlusEngine 实现的一种逻辑表，是在数据分片场景中具有相同结构的一组数据库（表）的逻辑对象。在所有分片内的表，结构和数据完全一致。
单表	可见	由 SphereEx-DBPlusEngine 实现的一种逻辑表，是在数据分片场景中一种数据库（表）的逻辑对象。其对应存储于某一数据源中。
加密表	可见	由 SphereEx-DBPlusEngine 实现的一种逻辑表，是在数据加密场景中一种数据库（表）的逻辑对象。
归档表	可见	由 SphereEx-DBPlusEngine 实现的一种逻辑表，其按照时间字段进行生命周期管理，可满足条件的分片实现删除或归档操作。现仅支持 MySQL

表操作

针对表的操作分为两种情况，一种是对逻辑表的操作（分片表、加密表等都是逻辑表的一种），另一种对物理表的操作。针对后者，SphereEx-DBPlusEngine 的处理与在数据源的操作是一致的，需要注意的当后者关联多个数据源时，是无法在指定数据源操作的。建议创建操作在数据源进行，并刷新元数据后，再在逻辑库进行其他操作。下文主要针对第一种情况进行说明。

- 增加 SphereEx-DBPlusEngine 针对表的增加操作，共分为三个步骤。
 - 第一步：添加数据源参见之前步骤
 - 第二步：创建规则根据需要需要的功能，创建对应的规则。下面以分片为例进行说明，其他参考[插件说明](#)。

```
sql> CREATE SHARDING TABLE RULE t_order(  
  RESOURCES(ds_3307,ds_3308),  
  SHARDING_COLUMN=order_id,  
  TYPE(NAME=hash_mod,PROPERTIES("sharding-count"=4)),  
  KEY_GENERATE_STRATEGY(  
    COLUMN=order_id,  
    TYPE(NAME=snowflake,PROPERTIES("worker-id"=123))  
  )  
);
```

- 第三步：创建表

```
sql> CREATE TABLE t_order (  
  order_id int NOT NULL,  
  user_id int NOT NULL,  
  status varchar(50) DEFAULT NULL,  
  PRIMARY KEY (order_id)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- 删除

SphereEx-DBPlusEngine 针对表的增加操作，共分为两个步骤。

- 删除表

```
sql> DROP TABLE IF EXISTS t_order;
```

- 删除规则

```
sql> DROP SHARDING TABLE RULE t_order;
```

需要注意的，目前 SphereEx-DBPlusEngine 尚未提供级联删除能力，如仅删除表会出现所谓“孤儿”现象，且可能造成后续创建同名对象的错误。建议可提供 Console 进行删除处理，或通过“show sharding table rules”命令定期查看是否存在未删除的无效对象。

- 修改

SphereEx-DBPlusEngine 针对表的修改操作，分为多种情况。

- 针对规则的修改针对规则的修改，将改变现有数据存储方式。在 SphereEx-DBPlusEngine 中通过与迁移能力相结合来完成的，这其中涉及到数据迁移问题。具体参见[数据迁移](#)
- 针对物理表的修改针对物理表的修改，建议通过命令行或图形方式，针对逻辑表进行修改。其对应操作将下发到物理表上。针对个别 SphereEx-DBPlusEngine 无法支持的表，可通过直接修改物理表后刷新元数据的方式完成。

- 查询

- 数据查询

在 SphereEx-DBPlusEngine 中创建的逻辑表，可通过正常的 DQL 方式查询。

- 结构查询

支持使用 DESC 或方言（如 MySQL 中的 show create table）查看元数据。但这种方式无法查看分片等细节，仅限于数据结构的查询。

其他问题

- 物理表分布

逻辑表的存储是通过若干张物理表组成。逻辑表与物理表间存在映射关系，可分为均匀分布和自定义分布两种形式。前者是指物理表在每个数据源内呈现均匀分布的态势，例如：

```
db0
├── t_order0
└── t_order1
db1
├── t_order0
└── t_order1
```

后者则可以通过表达式的配置方式，实现特定规则的分布，例如：

```
db0
├── t_order0
└── t_order1
db1
├── t_order2
├── t_order3
└── t_order4
```

- 物理表发现机制

对于挂载数据源中已存在的表，SphereEx-DBPlusEngine 是可以探测到并可直接使用。目前探测机制是采取手动方式，即在 SphereEx-DBPlusEngine 启动时或认为执行 refresh table metadata 语句均可刷新元数据。如果发生物理表的结构变更，建议使用 refresh 语句来刷新元数据。

- 同名表覆盖问题

当出现逻辑表与物理表同名的情况，SphereEx-DBPlusEngine 会优先使用逻辑表，即此时物理表无法在 SphereEx-DBPlusEngine 中识别并使用。

■ 广播表数据一致性

默认情况下，SphereEx-DBPlusEngine 不保证广播表数据一致性。如用户有刚需，可自行通过启动 XA 事务来保障。

■ 归档表

归档表，目前仅支持 MySQL，其底层实现是依赖于 MySQL 的分区特性。目前仅支持定期删除超期数据，尚不支持归档。

```
mysql> SHOW ARCHIVE RULES;
Empty set (0.36 sec)
```

```
mysql> CREATE ARCHIVE RULE t_order (
  ARCHIVE_STRATEGY(ARCHIVE_COLUMN=createtime, ARCHIVE_ALGORITHM(TYPE(NAME="SphereEx:TIME_PARTITION",
  PROPERTIES("interval-unit"="MONTHS", "interval-amount"=1, "retain-amount"=12, "pre-allocate-amount"=2))),
  SCHEDULER_ALGORITHM(TYPE(NAME="SphereEx:TIMED_TASK", PROPERTIES("schedule-cron-expr"="0 15 10 ? * *", "time-zone"="+08:00"))))
);
```

```
mysql> SHOW ARCHIVE TABLE RULE t_order;
```

table	archive_column	archive_algorithm_type	archive_algorithm_props	storage_algorithm_type	storage_algorithm_props	scheduler_algorithm_type	scheduler_algorithm_props
t_order	createtime	SphereEx:TIME_PARTITION	interval-amount=1,pre-allocate-amount=2,interval-unit=MONTHS,retain-amount=12			SphereEx:TIMED_TASK	schedule-cron-expr=0 15 10 ? * *,time-zone=+08:00

```
mysql> CREATE TABLE `t_order` (
  `order_id` bigint NOT NULL,
  `user_id` int NOT NULL,
  `status` varchar(50) NOT NULL,
  `merchant_id` int DEFAULT NULL,
  `remark` varchar(50) NOT NULL,
  `createtime` datetime NOT NULL,
  PRIMARY KEY (`order_id`, `creation_date`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
/*!50500 PARTITION BY RANGE COLUMNS(createtime)
(PARTITION p20230122 VALUES LESS THAN ('2023-01-25') ENGINE = InnoDB,
PARTITION p20230125 VALUES LESS THAN ('2023-01-28') ENGINE = InnoDB,
PARTITION p20230128 VALUES LESS THAN ('2023-01-31') ENGINE = InnoDB,
PARTITION p20230131 VALUES LESS THAN ('2023-02-03') ENGINE = InnoDB,
PARTITION p20230203 VALUES LESS THAN ('2023-02-06') ENGINE = InnoDB,
PARTITION p20230206 VALUES LESS THAN ('2023-02-09') ENGINE = InnoDB,
PARTITION p20230209 VALUES LESS THAN ('2023-02-12') ENGINE = InnoDB,
PARTITION p20230212 VALUES LESS THAN ('2023-02-15') ENGINE = InnoDB,
PARTITION p20230215 VALUES LESS THAN ('2023-02-18') ENGINE = InnoDB,
PARTITION p20230218 VALUES LESS THAN ('2023-02-21') ENGINE = InnoDB,
PARTITION p20230221 VALUES LESS THAN ('2023-02-24') ENGINE = InnoDB,
PARTITION p20230224 VALUES LESS THAN ('2023-02-27') ENGINE = InnoDB,
PARTITION p20230227 VALUES LESS THAN ('2023-03-02') ENGINE = InnoDB,
PARTITION p20230302 VALUES LESS THAN ('2023-03-05') ENGINE = InnoDB,
PARTITION p20230305 VALUES LESS THAN ('2023-03-08') ENGINE = InnoDB,
PARTITION p20230308 VALUES LESS THAN ('2023-03-11') ENGINE = InnoDB) */;
```

8.6.3 对象：字段

SphereEx-DBPlusEngine 通过对底层数据源原始数据的封装，提供对外标准对象能力。针对表内字段上，SphereEx-DBPlusEngine 也做了封装对外服务。从用户侧，几乎无感，可视为数据库字段同等处理。

字段类型

SphereEx-DBPlusEngine 支持底层数据源的所有字段类型，具体参考数据类型。

特殊场景：数据加密

在数据加密场景下，SphereEx-DBPlusEngine 针对字段有一些特殊处理。在实现上，SphereEx-DBPlusEngine 通过引入逻辑列来屏蔽底层加密字段的差异。可简单理解为是一层映射关系，可根据需要映射到明文字段或密文字段。可根据用户需求，SphereEx-DBPlusEngine 完成对应的转换过程。针对逻辑列的添加、删除是由加密规则操作来维护的，用户不能手工干预。

8.6.4 对象：约束

SphereEx-DBPlusEngine 本身未对底层数据源约束方面做过多处理，原生数据库支持的约束依然有效。但在数据分片场景下，数据分布在多个节点上，部分约束受到限制。

支持情况

约束名称	约束含义	支持情况（分片表）	支持情况（非分片表）	备注
PK	主键	部分支持	支持	支持包含分片键的主键约束
FK	外键	不支持	支持	
UK	UK	部分支持	支持	支持包含分片键的唯一性约束
Not NULL	非空	支持	支持	
CHECK	检查	部分支持	支持	支持不含跨分片校验的检查逻辑

限制情况

部分与约束相关的数据字典暂未支持，如 MySQL 下的 INFORMATION_SCHEMA.KEY_COLUMN_USAGE。

8.6.5 对象：索引

SphereEx-DBPlusEngine 本身未对底层数据源索引方面做过多处理，原生数据库支持的索引依然有效。但在数据分片场景下，数据分布在多个节点上，部分索引功能受到限制。

限制功能

- 分片条件下，不支持跨分片的唯一性索引。
- 分片条件下，不支持二级（或全局）索引。

8.6.6 对象：视图

SphereEx-DBPlusEngine 本身未对底层数据源视图方面做过多处理，原生数据库支持的视图依然有效。但在数据分片场景下，数据分布在多个节点上，部分视图功能受到限制。

8.6.7 对象：序列或自增长类型

序列或自增长类型，是数据库保证数据唯一性的有效手段。SphereEx-DBPlusEngine 本身未对底层数据源序列或自增长方面做处理，原生数据库支持依然有效。但在数据分片场景下，是有所不同的，受限与分布式环境的限制，很难如单机数据库的方式支持。一般推荐在分布式环境下，使用“分布式 ID”的功能替代，SphereEx-DBPlusEngine 通过一种名为“键生成器”的方式来支持多种分布式 ID。

键生成器用法

- 相关语法

具体参见 [DistSQL](#)。除了可用上述方式创建外，还可以在分片规则定义中直接使用。

- 生成示例

下面通过示例说明下键生成器的用法。

```
sql> CREATE SHARDING TABLE RULE t_order(
-> RESOURCES(ds_3307,ds_3308),
-> SHARDING_COLUMN=order_id,TYPER(NAME=hash_mod,PROPERTIES("sharding-count"=4)),
-> KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPER(NAME=uuid,PROPERTIES("worker-id"=123)))
->);
Query OK, 0 rows affected (0.03 sec)
```

```
sql> CREATE TABLE t_order (
-> order_id varchar(50) NOT NULL,
-> user_id int NOT NULL,
-> status varchar(50) DEFAULT NULL,
-> PRIMARY KEY (order_id)
->) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
Query OK, 0 rows affected (0.12 sec)
```

```
sql> INSERT INTO t_order(user_id, status) VALUES
-> (1, 'SUCCESS'),
-> (2, 'FAIL'),
-> (3, 'FAIL'),
-> (4, 'FAIL'),
-> (5, 'FAIL');
Query OK, 5 rows affected (0.01 sec)
```

```
sql> select * from t_order;
+-----+-----+-----+
| order_id          | user_id | status |
+-----+-----+-----+
| 48e10218c7414bdbaf640aea2c079d32 | 12 | FAIL |
| adcb6343fb43424baed8f297832d6897 | 9 | FAIL |
| d01fe0d8f99744949f6f734803a6fbad | 1 | SUCCESS |
| 0a52a963254f4c889924fcef27afa497 | 16 | FAIL |
```

```
| 0bee54f2ee8a4af8bb09250f01b2565b | 15 | FAIL |  
+-----+-----+-----+
```

定义并使用键生成器，会消耗 SphereEx-DBPlusEngine 的资源，经评估测试会有 60%~70% 左右的性能损耗，对于高并发、密集插入的场景需要关注。

内置分布式 ID

UUID

SphereEx-DBPlusEngine 内置 UUID 算法，可生成 36 位随机字符串。

- 与 MySQL 原生 UUID() 区别
 - 字段类型不同。如果原生 MySQL 是使用 BINARY 类型插入的，则使用 SphereEx-DBPlusEngine 无法插入，需定义为 VARCHAR 才可以。此处需关注未来用户迁移系统过程中的改造成本（数据类型需变化）。
 - 数据分配不均衡。在 HASH 分布策略下使用 UUID 数据分布是不均衡的。

NanoID

SphereEx-DBPlusEngine 内置 NanoID 算法，相较于 UUID 其具备更小、更安全、更快等优势。

Snowflake

SphereEx-DBPlusEngine 内置 Snowflake 算法，相较于 UUID 其具备更小和有序性等特点。

- 原理

在同一个进程中，它首先是通过时间位保证不重复，如果时间相同则是通过序列位保证。同时由于时间位是单调递增的，且各个服务器如果大体做了时间同步，那么生成的主键在分布式环境可以认为是总体有序的，这就保证了对索引字段的插入的高效性。例如 MySQL 的 InnoDB 存储引擎的主键。

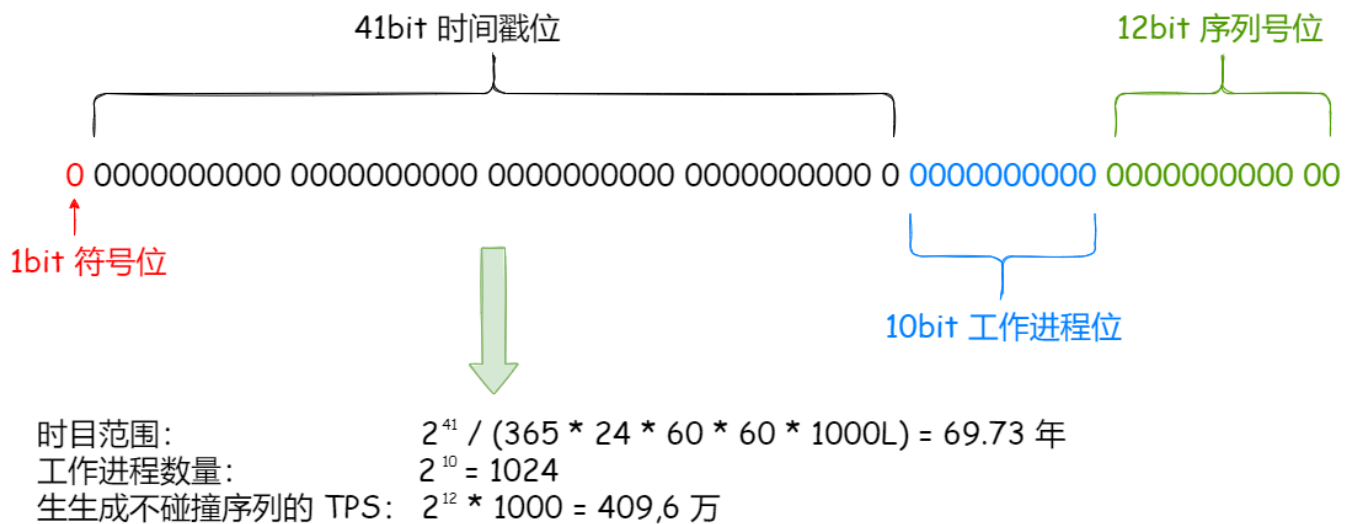


Fig. 2: Snowflake 原理图

■ 格式

使用雪花算法生成的主键，二进制表示形式包含 4 部分，从高位到低位分表为：1bit 符号位、41bit 时间戳位、10bit 工作进程位以及 12bit 序列号位。

- 符号位 (1bit)

预留的符号位，恒为零。

- 时间戳位 (41bit)

41 位的时间戳可以容纳的毫秒数是 2 的 41 次幂，一年所使用的毫秒数是：365 * 24 * 60 * 60 * 1000。通过计算可知：Math.pow(2, 41) / (365 * 24 * 60 * 60 * 1000L);

结果约等于 69.73 年。SphereEx-DBPlusEngine 的雪花算法的时间纪元从 2016 年 11 月 1 日零点开始，可以使用到 2086 年，相信能满足绝大部分系统的要求。

- 工作进程位 (10bit)

该标志在 Java 进程内是唯一的，如果是分布式应用部署应保证每个工作进程的 id 是不同的。该值默认为 0，可通过属性设置。

- 序列号位 (12bit)

该序列是用来在同一个毫秒内生成不同的 ID。如果在这个毫秒内生成的数量超过 4096 (2 的 12 次幂)，那么生成器会等待到下个毫秒继续生成。

■ 时钟回拨

服务器时钟回拨会导致产生重复序列，因此默认分布式主键生成器提供了一个最大容忍的时钟回拨毫秒数。如果时钟回拨的时间超过最大容忍的毫秒数阈值，则程序报错；如果在可容忍的范围内，默认分布式主键生成器会等待时钟同步到最后一次主键生成的时间后再继续工作。最大容忍的时钟回拨毫秒数的默认值为 0，可通过属性设置。

■ 限制条件

使用 Snowflake 算法，SphereEx-DBPlusEngine 集群规模不能超过 1024 个计算节点。

8.6.8 对象：库内计算

针对数据库场景的内置计算对象，如存储过程、触发器、函数等，SphereEx-DBPlusEngine 提供有限支持。对于原始数据库内部对象，SphereEx-DBPlusEngine 可通过直接透传下发方式支持。对于在数据分片、读写分离等场景下，不支持上述对象。

8.6.9 一致性

针对部分数据分片的场景，SphereEx-DBPlusEngine 提供 DDL 一致性保证。即执行过程中，要么全部分片完成此 DDL 操作，要么全部分片取消此 DDL 操作，不会出现数据结构不一致的情况。其实现原理是当执行过程异常时，提供反向 DDL SQL，完成回退操作。对于部分操作，SphereEx-DBPlusEngine 尚不提供回退能力或需要用户手工介入重试完成。

执行中操作影响

类别	语句	支持回退	对 DDL 影响	对 DQL 影响	对 DML 影响	对 TCL 影响	对 DistSQL 影响	可见性
DATABASE	CREATE	支持回退	无法执行	无法执行	无法执行	可以执行	无法执行	不可见
DATABASE	DROP	不支持回退	无法执行	无法执行	无法执行	可以执行	无法执行	不可见
SCHEMA	CREATE	支持回退	无法执行	无法执行	无法执行	可以执行	无法执行	不可见
SCHEMA	DROP	不支持回退	无法执行	无法执行	无法执行	可以执行	无法执行	不可见
SCHEMA	ALTER	支持回退	无法执行	可以执行	无法执行	可以执行	无法执行	不可见
TABLE	CREATE	支持回退	无法执行	可部分执行	无法执行	可以执行	无法执行	不可见
TABLE	DROP	不支持回退，需重试	无法执行	无法执行	无法执行	无法执行	无法执行	不可见
TABLE	ALTER (CREATE/ADD INDEX)	支持回退	无法执行	可以执行	无法执行	可以执行	无法执行	不可见
TABLE	ALTER (DROP INDEX)	不支持回退，需重试	无法执行	可以执行	无法执行	可以执行	无法执行	不可见
TABLE	ALTER (RENAME INDEX)	支持回退	无法执行	可以执行	无法执行	可以执行	无法执行	不可见
TABLE	ALTER (ADD COLUMN)	支持回退	无法执行	可部分执行	无法执行	可以执行	无法执行	不可见
TABLE	ALTER (DROP COLUMN)	不支持回退，需重试	无法执行	可部分执行	无法执行	可以执行	无法执行	不可见
TABLE	ALTER (MODIFY COLUMN)	不支持回退，需重试	无法执行	可以执行	无法执行	可以执行	无法执行	不可见
TABLE	ALTER (RENAME COLUMN)	支持回退	无法执行	可部分执行	无法执行	可以执行	无法执行	不可见
TABLE	ALTER (CHANGE COLUMN)	不支持回退，需重试	无法执行	可部分执行	无法执行	可以执行	无法执行	不可见

TABLE 包括分片表、广播表和单表。

操作示例

```
sql> create table t_order (order_id INT NOT NULL, user_id INT NOT NULL, status VARCHAR(45) NULL, `test_column` bigint(20) NOT NULL DEFAULT '0', `test_column2` bigint(20) NOT NULL DEFAULT '0');
```

-- 模拟故障

```
sql> alter table `t_order` modify column `test_column` varchar(10), modify column `test_column2` varchar(10);
-- ERROR 30000 (HY000): Unknown exception: Table 'demo_ds_1.t_order_0' doesn't exist,
-- please use SHOW DDL STATUS 'j5601839ceed1d09d73d9d1f7e7f77de75228' to check status.
-- 此时会有一个存储节点 ds_0 上执行成功，ds_1 上执行失败。由于不支持回滚，会触发 job 重试操作。
```

```
mysql> show ddl list;
```

id	tables	ddl_type	job_item_count	active	create_time	stop_time
j5601839ceed1d09d73d9d1f7e7f77de75228	t_order	ALTER_TABLE	1	false	2023-02-19 16:27:	

-- 修复故障

-- proxy 上重试 job

```
sql> start ddl 'j5601839ceed1d09d73d9d1f7e7f77de75228';
```

-- proxy 查看 job 执行完成并自动删除

```
mysql> show ddl list;
```

```
Empty set (0.02 sec)
```

异常说明

针对分布式对象执行 DDL 操作，在极端情况下会出现在不同计算节点看到元数据不一致的情况。针对这种情况可通过在对应计算节点执行 `REFRESH DATABASE METADATA db_name FROM GOVERNANCE CENTER;` 命令手工完成元数据刷新。具体参见 [DistSQL](#)。

8.7 对象设计

8.7.1 分片

核心概念

数据分片，是指按照某个维度将存放在单一数据库中的数据分散地存放至多个数据库或表中以达到提升性能瓶颈以及可用性的效果。数据分片的有效手段是对关系型数据库进行分库和分表。分库和分表均可以有效的避免由数据量超过可承受阈值而产生的查询瓶颈。除此之外，分库还能够用于有效的分散对数据库单点的访问量；分表虽然无法缓解数据库压力，但却能够提供尽量将分布式事务转化为本地事务的可能，一旦涉及到跨库的更新操作，分布式事务往往会使问题变得复杂。使用多主多从的分片方式，可以有效的避免数据单点，从而提升数据架构的可用性。通过分库和分表进行数据的拆分来使得各个表的数据量保持在阈值以下，以及对流量进行疏导应对高访问量，是应对高并发和海量数据系统的有效手段。数据分片的拆分方式又分为垂直分片和水平分片。

垂直分片

按照业务拆分的方式称为垂直分片，又称为纵向拆分，它的核心理念是专库专用。在拆分之前，一个数据库由多个数据表构成，每个表对应着不同的业务。而拆分之后，则是按照业务将表进行归类，分布到不同的数据库中，从而将压力分散至不同的数据库。下图展示了根据业务需要，将用户表和订单表垂直分片到不同的数据库的方案。

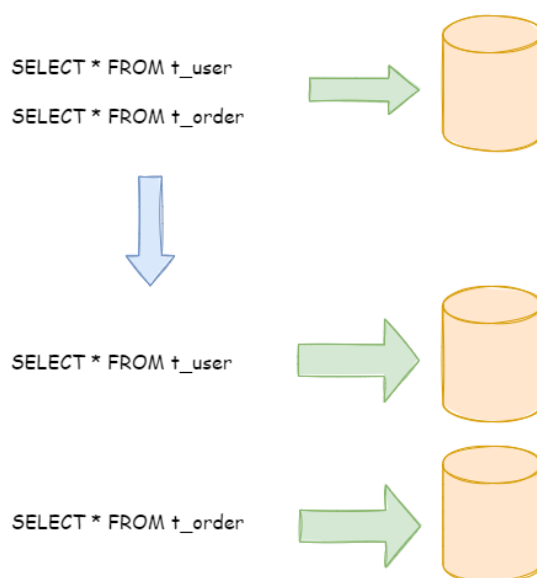


Fig. 3: 垂直分片原理图

垂直分片往往需要对架构和设计进行调整。通常来讲，是来不及应对互联网业务需求快速变化的；而且，它也并无法真正的解决单点瓶颈。垂直拆分可以缓解数据量和访问量带来的问题，但无法根治。如果垂直拆分之后，表中的数据量依然超过单节点所能承载的阈值，则需要水平分片来进一步处理。

水平分片

水平分片又称为横向拆分。相对于垂直分片，它不再将数据根据业务逻辑分类，而是通过某个字段（或某几个字段），根据某种规则将数据分散至多个库或表中，每个分片仅包含数据的一部分。例如：根据主键分片，偶数主键的记录放入 0 库（或表），奇数主键的记录放入 1 库（或表），如下图所示。

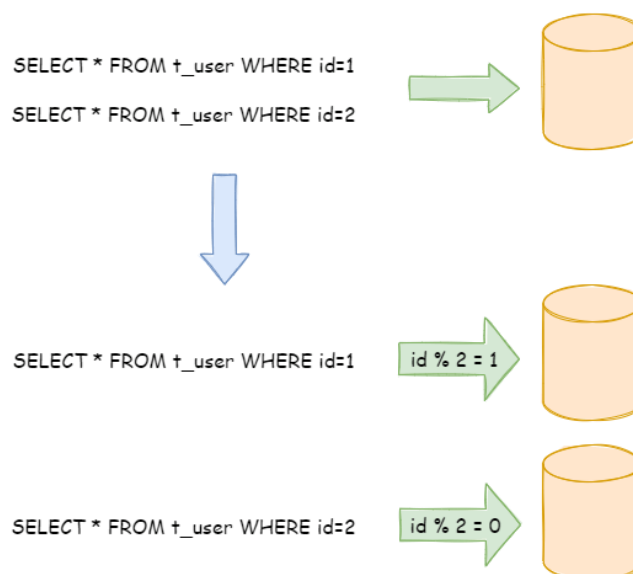


Fig. 4: 水平分片原理图

水平分片从理论上突破了单机数据量处理的瓶颈，并且扩展相对自由，是数据分片的标准解决方案。

名词解释

■ 表

表是透明化数据分片的关键概念。SphereEx-DBPlusEngine 通过提供多样化的表类型，适配不同场景下的数据分片需求。

■ 逻辑表

相同结构的水平拆分数据库（表）的逻辑名称，是 SQL 中表的逻辑标识。例：订单数据根据主键尾数拆分为 10 张表，分别是 t_order_0 到 t_order_9，他们的逻辑表名为 t_order。

■ 真实表

在水平拆分的数据库中真实存在的物理表。即上个示例中的 t_order_0 到 t_order_9。

■ 表引用

指分片规则一致的主表和子表。使用表引用进行多表关联查询时，必须使用分片键进行关联，否则会出现笛卡尔积关联或跨库关联，从而影响查询效率。例如：t_order 表和 t_order_item 表，均按照 order_id 分片，并且使用 order_id 进行关联，则此两张表互为绑定表关系。绑定表之间的多表关联查询不会出现笛卡尔积关联，关联查询效率将大大提升。

举例说明，如果 SQL 为：

```
SELECT i.* FROM t_order o JOIN t_order_item i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
```

在不配置绑定表关系时，假设分片键 order_id 将数值 10 路由至第 0 片，将数值 11 路由至第 1 片，那么路由后的 SQL 应该为 4 条，它们呈现为笛卡尔积：

```
SELECT i.* FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
SELECT i.* FROM t_order_0 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
SELECT i.* FROM t_order_1 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
SELECT i.* FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
```

在配置绑定表关系，并且使用 order_id 进行关联后，路由的 SQL 应该为 2 条：

```
SELECT i.* FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
SELECT i.* FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
```

其中 t_order 在 FROM 的最左侧，ShardingSphere 将会以它作为整个绑定表的主表。所有路由计算将会只使用主表的策略，那么 t_order_item 表的分片计算将会使用 t_order 的条件。因此，绑定表间的分区键需要完全相同。

■ 广播表

指所有的分片数据源中都存在的表，表结构及其数据在每个数据库中均完全一致。适用于数据量不大且需要与海量数据的表进行关联查询的场景，例如：字典表。

■ 单表

指所有的分片数据源中仅唯一存在的表。适用于数据量不大且无需分片的表。如单表增长到一定规模，SphereEx-DBPlusEngine 也支持将其转为分片表。

■ 数据节点

数据分片的最小单元，由数据源名称和真实表组成。例：ds_0.t_order_0。逻辑表与真实表的映射关系，可分为均匀分布和自定义分布两种形式。

■ 均匀分布

指数据表在每个数据源内呈现均匀分布的态势，例如：

```
db0
├── t_order0
└── t_order1
db1
├── t_order0
└── t_order1
```

数据节点的配置如下：

db0.t_order0, db0.t_order1, db1.t_order0, db1.t_order1

■ 自定义分布指数据表呈现有特定规则的分布，例如：

```
db0
├── t_order0
└── t_order1
db1
├── t_order2
├── t_order3
└── t_order4
```

数据节点的配置如下：

db0.t_order0, db0.t_order1, db1.t_order2, db1.t_order3, db1.t_order4

分片设计

■ 分片键

用于将数据库（表）水平拆分的数据库字段。例：将订单表中的订单主键的尾数取模分片，则订单主键为分片字段。SQL 中如果无分片字段，将执行全路由，性能较差。除了对单分片字段的支持，SphereEx-DBPlusEngine 也支持根据多个字段进行分片。分片字段的选择，涉及下列因素。

■ 数据结构：主键或唯一键

主键及唯一键，是数据库作为常见的约束，其是为了保证非空且唯一性。在分布式环境下，通常建议将主键或唯一键字段作为分片键或分片键的一部分，否则无法完成约束校验。这里有个引申问题，就是主键设计问题，在分布式数据库架构下，尽量不要用自增作为表的主键，自增性能很差、安全性不高、不适用于分布式架构。通常可使用如 UUID 或全局发号器（雪花算法）。总之，用有序的全局唯一替代自增，是分布式数据库主键的推荐做法。

■ 数据结构：索引

通过分片键可以把 SQL 查询路由到指定的分片，但是在现实的生产环境中，业务还要通过其他的索引访问表。针对原有系统的索引需要有单独策略。通常的策略是通过索引表的方式，即将索引转化为另一张分片表，对于查询来说通过二次查询解决，但显然这种方式不够优雅。因此，最优的设计不是创建一个索引表，而是将索引数据融入到分片键的信息中，这样通过查询的列就能直接知道所在的分片信息。效率更高，查询可以提前知道数据对应的分片信息，只需一次查询就能获取想要的结果。总结下，索引对分片字段的选择上，没有直接影响。对于高频索引查询，可以考虑通过分片键的设计上进行增强。也可以通过全局二级索引（有些分布式数据库支持）来实现或针对分片内做普通索引。

■ 数据结构：字段类型

作为分片键的字段，通常选择较为简单的数据类型字段，可以提高效率，如常见的数字、日期、文本等，对复杂字段如 LOB、JSON 等不推荐使用。此外，在设计时需考虑分片字段的类型稳定，尽量不要发生 DDL 变更。

■ 数据特征：表规模

表规模是否使用分片的关键因素之一。一旦表做了分片后，势必会造成一定的“功能退化”，如能采取其他方式缩小表的大小，尽量优先其他方式。可通过表的全生命周期规划，如常规的数据归档、压缩、转储、清理策略，减少数据量；或者利用数据库内置的如表分区、垂直分表等策略有效减小表的大小。

■ 数据特征：离散度

这里说的离散度是指按某个字段或字段组合后，应用分片算法后，数据是否足够分散。数据分片的初衷就是减少表的规模，尽量做到数据打散是其根本原则之一。这里需要统计数据拆分后离散程度，尽量选择能充分打散的字段作为分片键。这里需注意，如果选择字段是带有业务特征，还要关注未来业务变化对它的影响。

■ 访问特征：可变化性

选择固定、不再变化的字段作为分片键。虽然有些分布式数据库也支持分片键的修改，但毕竟修改后会涉及数据移动，成本代价很高；还是优选不变的字段为好。

■ 访问特征：事务隔离

尽量选择按字段拆分后的数据，对数据的变化处理可集中在分片内解决。这样大量的业务变化是可以通过本地事务完成，开销比全局的要小很多，效率也高。

■ 访问特征：数据过滤与关联

如此字段经常作为数据筛选字段被频繁使用，且选择率很好，可优先作为分片字段。另一种情况则是作为与其他关联表联合使用，优先选择那些参与到关联操作的字段为佳。尽量是数据在关联后，能在本地完成 join 动作，减少数据 shuffle 或上移汇聚类的操作。可通过对系统中执行的 SQL 进行统计分析，选择出需要分片那个表中最频繁被使用到或最为重要的字段类分片。这其中可能包含一些来自 OLAP 类的查询，可将此部分 SQL 排除在外。

■ 分片字段顺序

如涉及多个字段作为分片键的话，顺序因素一般没有什么影响。主要是针对分片算法，可利用字段做分片即可。但对于复合分片的情况，是要考虑分片字段的主次关系的。

■ 分片算法

分片算法，常规的有 LIST、RANGE、HASH 或自定义算法。根据各拆分算法特点，可进行选择。若范围均匀可采用 HASH，冷热数据明显可采用 RANGE 等。同时可配合一些特性化设计，如采用二级映射方式解决扩缩容问题、特征编码字段满足多特征拆分等。针对最为常见的两个算法描述如下：

● RANGE

通过数据的范围进行分库分表，是最朴实的一种分库方案，它也可以和其他分库分表方案灵活结合使用。当需要使用分片字段进行范围查找时，RANGE 分片策略可快速定位数据进行高效查询。大多数情况下有效避免跨分片查询的问题。在后期扩容时，也比较方便，只需要添加节点即可，无需对其他分片的数据进行迁移。但这种分布方式容易存在数据热点问题。

- HASH

虽然分库分表的方案众多，但是 Hash 分库分表是最大众最普遍的方案。随机分片其实并不是随机，也遵循一定规则。通常采用 HASH 取模的方式进行分片拆分，所以有时候也称为离散分片。随机分片的数据相对均匀，不容易出现热点和并发访问的瓶颈。但涉及后面数据迁移的话，不太方便。可使用一致性 HASH 算法在很大程度上避免此问题。此外，离散分片也容易面临跨分片查询的复杂问题。

- 自定义分片算法

提供接口让应用开发者自行实现与业务实现紧密相关的分片算法，并允许使用者自行管理真实表的物理分布。自定义分片算法又分为：

- 标准分片算法

用于处理使用单一键作为分片键的 =、IN、BETWEEN AND、>、<、>=、<= 进行分片的场景。

- 复合分片算法

用于处理使用多键作为分片键进行分片的场景，包含多个分片键的逻辑较复杂，需要应用开发者自行处理其中的复杂度。

- Hint 分片算法

用于处理使用 Hint 行分片的场景。

- 强制分片

对于分片字段并非由 SQL 而是其他外置条件决定的场景，可使用 SQL Hint 注入分片值。例：按照员工登录主键分库，而数据库中并无此字段。SQL Hint 支持通过 SQL 注释方式使用。详情请参见强制分片路由。

使用规范

虽然 SphereEx-DBPlusEngine 希望能够完全兼容所有的 SQL 以及单机数据库，但分布式为数据库带来了更加复杂的场景。

- SQL 支持程度

兼容全部常用的路由至单数据节点的 SQL；路由至多数据节点的 SQL 由于场景复杂，分为稳定支持、实验性支持和不支持这三种情况。

- 稳定支持

稳定支持的 SQL	必要条件
SELECT * FROM tbl_name	
SELECT * FROM tbl_name WHERE (col1 = ? or col2 = ?) and col3 = ?	
SELECT * FROM tbl_name WHERE col1 = ? ORDER BY col2 DESC LIMIT ?	
SELECT COUNT(*), SUM(col1), MIN(col1), MAX(col1), AVG(col1) FROM tbl_name WHERE col1 = ?	
SELECT COUNT(col1) FROM tbl_name WHERE col2 = ? GROUP BY col1 ORDER BY col3 DESC LIMIT ?, ?	
SELECT DISTINCT * FROM tbl_name WHERE col1 = ?	
SELECT COUNT(DISTINCT col1), SUM(DISTINCT col1) FROM tbl_name (SELECT * FROM tbl_name)	
SELECT * FROM (SELECT * FROM tbl_name WHERE col1 = ?) o WHERE o.col1 = ?	子查询和外层查询在同一片片后的数据节点
INSERT INTO tbl_name (col1, col2, ...) VALUES (?, ?, ...)	
INSERT INTO tbl_name VALUES (?, ?, ...)	
INSERT INTO tbl_name (col1, col2, ...) VALUES(1 + 2, ?, ...)	
INSERT INTO tbl_name (col1, col2, ...) VALUES (?, ?, ...), (?, ?, ...)	
INSERT INTO tbl_name (col1, col2, ...) SELECT col1, col2, ...FROM tbl_name WHERE col3 = ?	INSERT 表和 SELECT 表相同表或绑定表
REPLACE INTO tbl_name (col1, col2, ...) SELECT col1, col2, ...FROM tbl_name WHERE col3 = ?	REPLACE 表和 SELECT 表相同表或绑定表
UPDATE tbl_name SET col1 = ? WHERE col2 = ?	
DELETE FROM tbl_name WHERE col1 = ?	
CREATE TABLE tbl_name (col1 int, ...)	
ALTER TABLE tbl_name ADD col1 varchar(10)	
DROP TABLE tbl_name	
TRUNCATE TABLE tbl_name	
CREATE INDEX idx_name ON tbl_name	
DROP INDEX idx_name ON tbl_name	
DROP INDEX idx_name	

Fig. 5: 稳定支持的 SQL

全面支持 DML、DDL、DCL、TCL 和常用 DAL。支持分页、去重、排序、分组、聚合、表关联等复杂查询。

■ 实验性支持

实验性支持的 SQL	必要条件
SELECT * FROM (SELECT * FROM tbl_name) o	
SELECT * FROM (SELECT * FROM tbl_name) o WHERE o.col1 = ?	
SELECT * FROM (SELECT * FROM tbl_name WHERE col1 = ?) o	
SELECT * FROM (SELECT * FROM tbl_name WHERE col1 = ?) o WHERE o.col1 = ?	子查询和外层查询不在同一片片后的数据节点
SELECT (SELECT MAX(col1) FROM tbl_name) a, col2 from tbl_name	
SELECT SUM(DISTINCT col1), SUM(col1) FROM tbl_name	
SELECT col1, SUM(col2) FROM tbl_name GROUP BY col1 HAVING SUM(col2) > ?	
SELECT col1, col2 FROM tbl_name UNION SELECT col1, col2 FROM tbl_name	
SELECT col1, col2 FROM tbl_name UNION ALL SELECT col1, col2 FROM tbl_name	

Fig. 6: 实验性支持的 SQL

实验性支持特指使用 Federation 执行引擎提供支持。该引擎处于快速开发中，用户虽基本可用，但仍需大量优化，是实验性产品。

■ 不支持

不支持的 SQL	原因	解 决 方 案
INSERT INTO tbl_name (col1, col2, ...) SELECT * FROM tbl_name WHERE col3 = ?	SELECT 子句不支持 * 和内置分布式主键生成器	无
REPLACE INTO tbl_name (col1, col2, ...) SELECT * FROM tbl_name WHERE col3 = ?	SELECT 子句不支持 * 和内置分布式主键生成器	无
SELECT MAX(tbl_name.col1) FROM tbl_name	查询列是函数表达式时，查询列前不能使用表名	使 用 表 别名

Fig. 7: 不支持的 SQL

8.7.2 数据加密

背景说明

作为数据防护是否能够成功实施的关键，企业需要在关键数据的安全性、保持应用系统的功能可用性，和系统可维护性方面综合考虑，来确定适合企业需要的加密保护的技术方案。下表罗列常用加密技术及应对的安全风险。

安全风险	磁 盘 加 密	文 件 加 密	数 据 库 加 密 (TDE)	数据库加密 (三方加固)	应用层加密
防止磁盘丢失引起数据泄露	Y	Y	Y	Y	Y
防止系统 root 账户和管理员账户访问	N	Y	Y	Y	Y
控制数据库管理员访问数据	N	N	N	Y	Y
抵抗定向威胁供给 APT 造成数据泄露	N	Y	Y	Y	Y
提供细粒度访问记录、遵从法规	N	Y	N	Y	Y
确保备份数据和数据快照存储	N	Y	Y	Y	Y
非结构化数据和文件的保护	Y	Y	N	N	Y
防止硬件和数据库厂商“偷窥”	N	Y	N	Y	Y

■ 磁盘加密

磁盘采用的块级别加密技术，例如 AWS 的 EBS，阿里云的 ECS 等都支持磁盘加密。这种加密最大的好处在于，它对操作系统是透明的。性能在加密后较加密前有所降低，根据上层应用的不同性能下降幅度各异。

■ 文件加密

通过堆叠在其它文件系统之上（如 Ext2, Ext3, ReiserFS, JFS 等），为应用程序提供透明、动态、高效和安全的加密功能。典型的是用于加密指定的目录。需要关注的是这种加密方式可能会产生较大的性能损失。

■ 数据库加密-TDE

透明数据加密 TDE，是数据库提供了一种加密技术，即对数据文件执行实时 I/O 加密和解密。数据在写入磁盘之前进行加密，从磁盘读入内存时进行解密。TDE 不会增加数据文件的大小，开发人员无需更改任何应用程序。其对应密钥管理也是由数据库提供的 API 或组件实现，应用透明。在某些场景下磁盘或系统无法对用户开放（如云环境）的条件下，这种方式就比较适合。

■ 数据库加密-三方加固

数据库加密还有种方式是采用对数据库进行三方加固的方式，即将第三方专业数据库加密厂商的产品内置在数据库之中，提供透明数据加密能力。所谓透明是指，用户应用系统不需要做改造即可使用，且具有权限的用户看到的是明文数据，完全无感。此外，还可以增强原有数据库的安全能力，如提供三权分立、脱敏展示等。

■ 应用层加密

应用层加密，可以说是一种终极方案，其可保证在数据到达数据库之前，就已经做了数据加密，可实时保护用户敏感数据。这里关键需要提供应用透明性，保证应用无需改造或仅需少量改造。这种方式完全由用户自己控制，无需信任任何三方厂商提供的数据安全保障，得到充分的自由度和灵活性。例如可以跨多数据库提供统一安全加密策略等。

名词解释

■ 逻辑列

用于计算加解密列的逻辑名称，是 SQL 中列的逻辑标识。逻辑列包含密文列（必须）、查询辅助列（可选）和明文列（可选）。

■ 逻辑列类型（dataType）

用于定义逻辑列的类型，例如 INT NOT NULL，VARCHAR(200) DEFAULT NULL 等，具体可以参见官方文档中各种方言字段的定义。例如 MySQL create statement 中 column_definition 的定义 (<https://dev.mysql.com/doc/refman/8.0/en/create-table.html>)。

■ 密文列（cipherColumn）

加密后的数据列。

■ 密文列类型（cipherDataType）

用于定义密文列的类型，同逻辑列类型。

■ 查询辅助列（assistedQueryColumn）

用于查询的辅助列。对于一些安全级别更高的非幂等加密算法，提供不可逆的幂等列用于查询。

■ 查询辅助列类型（assistedQueryDataType）

用于定义查询辅助列类型，同逻辑列类型。

■ 明文列（plainColumn）

存储明文的列，用于在加密数据迁移过程中仍旧提供服务。

在洗数结束后可以删除。

■ 明文列类型（plainDataType）

用于定义明文列类型，同逻辑列类型。

■ 加密洗数（encrypting）

批量对数据库中未加密数据进行加密。

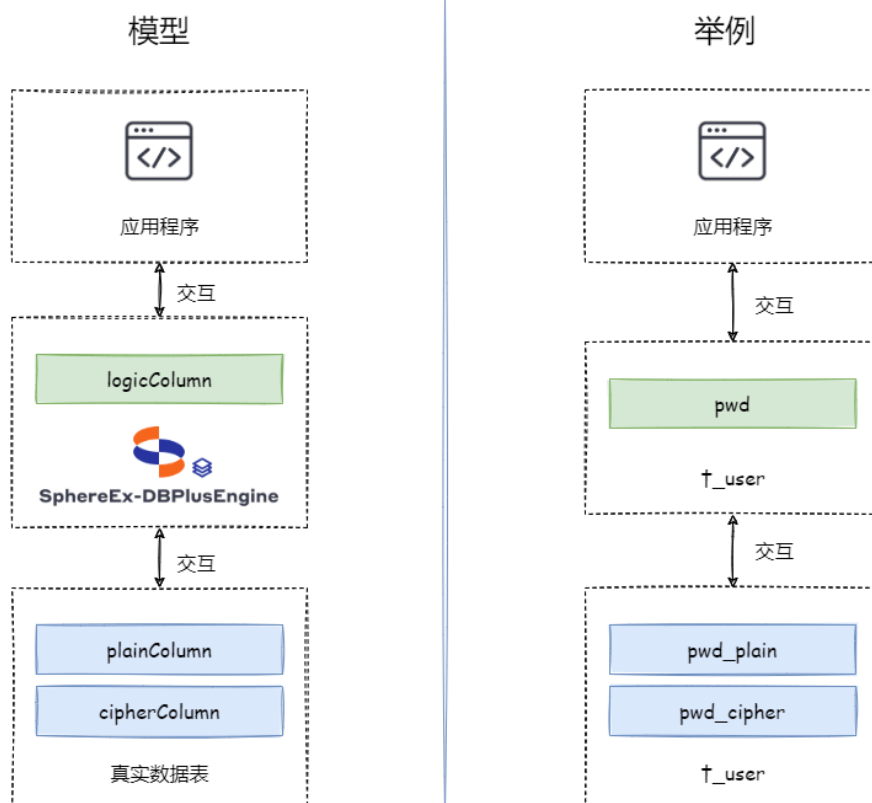
■ 解密洗数（decrypting）

批量对数据库中加密数据进行解密。

实现原理

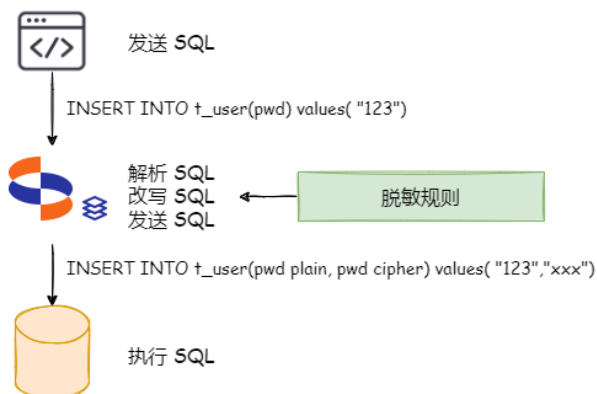
■ 数据加密原理

假如数据库里有一张表叫做 t_user，这张表里实际有两个字段 pwd_plain，用于存放明文数据、pwd_cipher，用于存放密文数据、pwd_assisted_query，用于存放辅助查询数据，同时定义 logicColumn 为 pwd。那么，用户在编写 SQL 时应该面向 logicColumn 进行编写，即 INSERT INTO t_user SET pwd = '123'。SphereEx-DBPlusEngine 接收到该 SQL，通过用户提供的加密配置，发现 pwd 是 logicColumn，于是便对逻辑列及其对应的明文数据进行加密处理。SphereEx-DBPlusEngine 将面向用户的逻辑列与面向底层数据库的明文列和密文列进行了列名以及数据的加密映射转换。如下图所示：

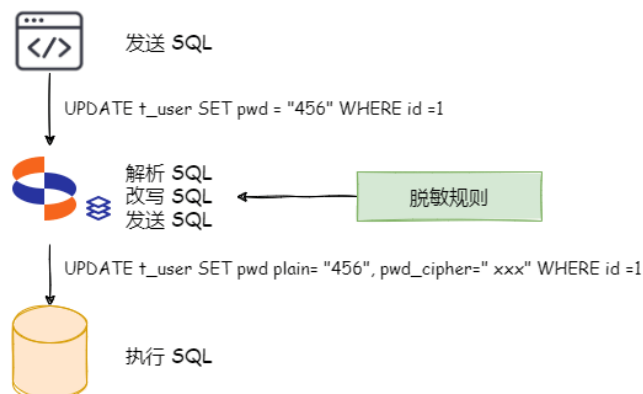


即依据用户提供的加密规则，将用户 SQL 与底层数据表结构割裂开来，使得用户的 SQL 编写不再依赖于真实的数据库表结构。而用户与底层数据库之间的衔接、映射、转换交由 SphereEx-DBPlusEngine 进行处理。下方展示了使用加密模块进行增删改查时，其中的处理流程和转换逻辑。

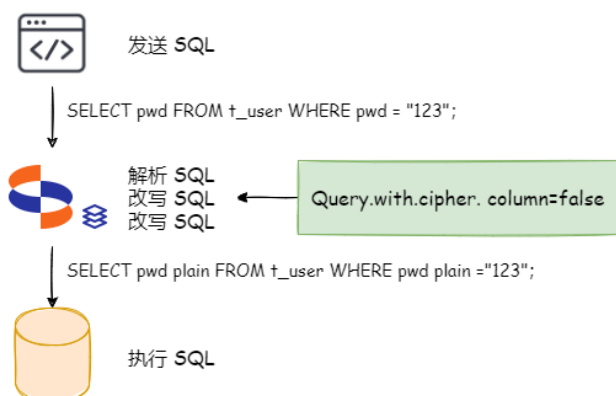
插入-使用逻辑列



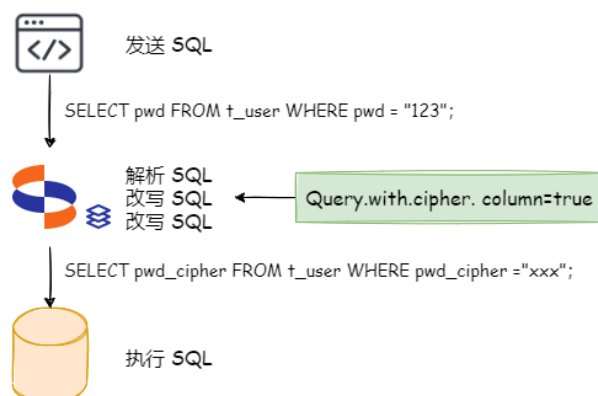
更新-使用逻辑列



查询-使用明文列

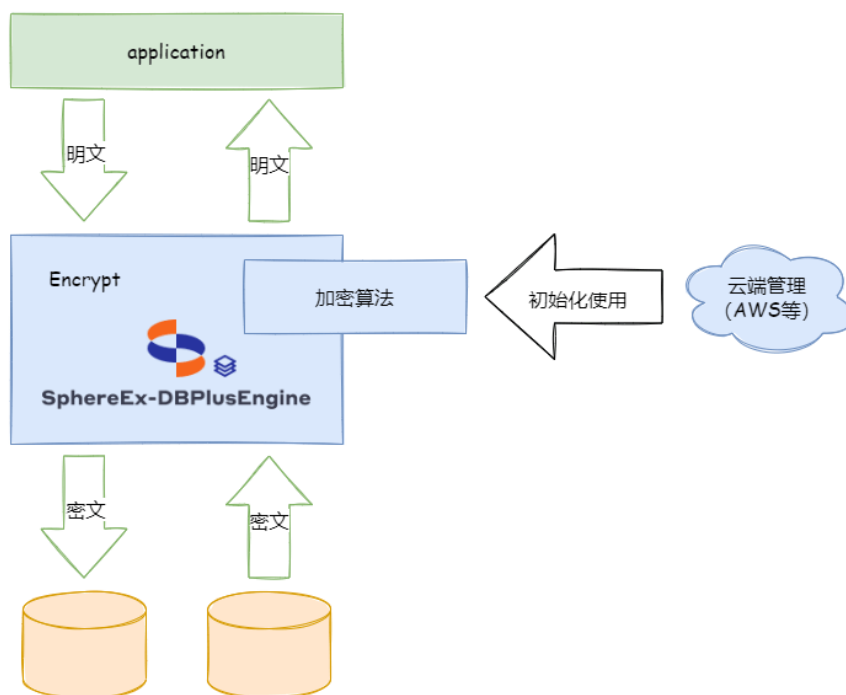


查询-使用密文列



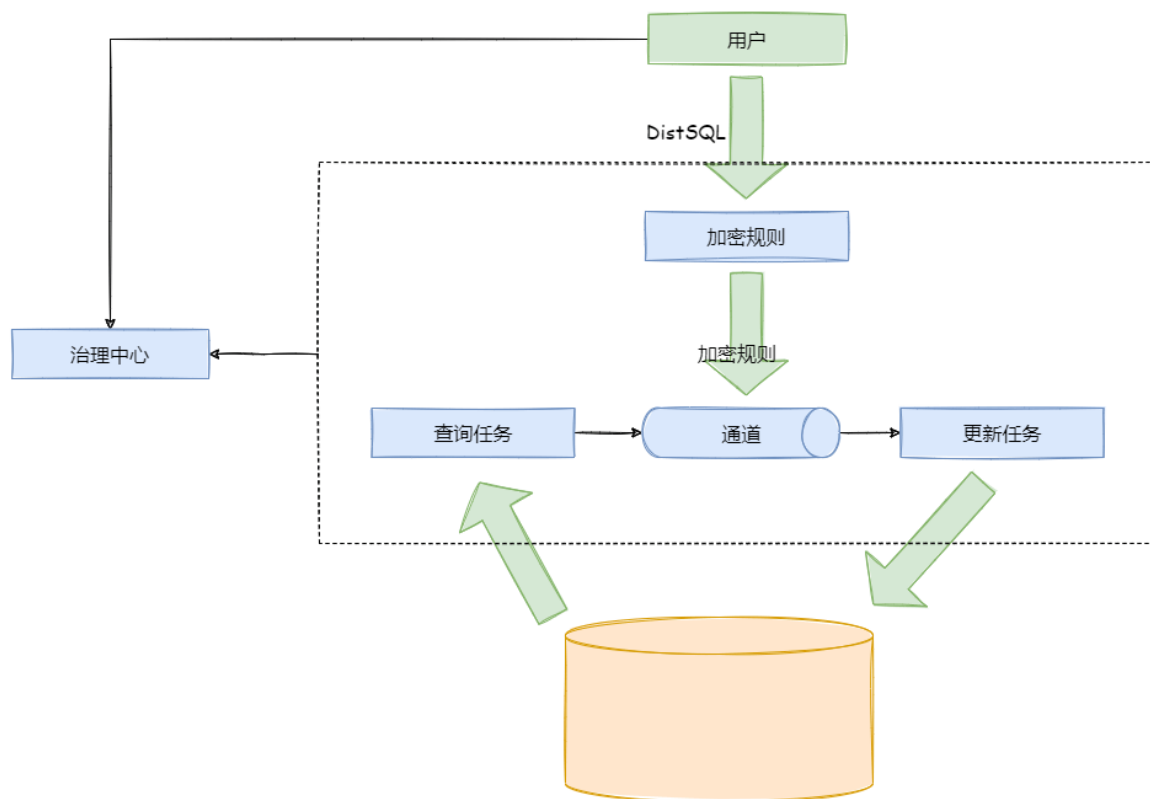
■ 云端密钥管理

将密钥管理在云端，例如利用 AWS 的 secretKey 功能保存密钥，从而提升整个加密的安全性和便捷性。程序在初始化加密算法时，会同 AWS 建立连接，从而获取存储在 AWS 中的相关密钥，然后将密钥存储在算法中。在整个数据加密的过程中不涉及与云端的网络交互。



■ 加密洗数

加密洗数通过 DistSQL 来触发洗数任务，程序收到洗数任务的请求之后，会根据当前的洗数规则以及加密规则创建洗数任务。洗数任务主要由两部分构成，一部分是查询任务，一部分是更新任务，查询任务负责查询用户的表数据并获取需要加密的明文字段，然后推送到通道中；更新任务则从通道中获取数据，并加密更新。整个任务的创建以及执行过程都会与治理中心进行交互，因此用户可以通过相关 DistSQL 来查询任务进度以及清理任务。



使用规范

■ 支持项

- 对数据库表中某个或多个列进行加解密；
- 兼容所有常用 SQL。

■ 不支持项

- 需自行处理数据库中原生的存量数据；
- 加密字段无法支持比较操作，如：大于、小于、ORDER BY、BETWEEN、LIKE 等；
- 加密字段无法支持计算操作，如：AVG、SUM 以及计算表达式。

8.8 事务管理

8.8.1 事务原理

虽然 SphereEx-DBPlusEngine 希望能够完全兼容所有的分布式事务场景，并在性能上达到最优，但在 CAP 定理所指导下，分布式事务必然有所取舍。SphereEx-DBPlusEngine 希望能够将分布式事务的选择权交给使用者，在不同的场景使用最适合的分布式事务解决方案。由于应用的场景不同，需要开发者能够合理的在性能与功能之间权衡各种分布式事务。强一致的事务与柔性事务的 API 和功能并不完全相同，在它们之间并不能做到自由的透明切换。在开发决策阶段，就不得不在强一致的事务和柔性事务之间抉择，使得设计和开发成本被大幅增加。基于 XA 的强一致事务使用相对简单，但是无法很好的应对互联网的高并发或复杂系统的长事务场景；柔性事务则需要开发者对应用进行改造，接入成本非常高，并且需要开发者自行实现资源锁定和反向补偿。

8.8.2 事务分类

SphereEx-DBPlusEngine 提供了三种事务模式：LOCAL，XA，BASE，以应对不同的场景。- LOCAL：适用于对数据一致性要求不高的场景。- XA：提供了原子性的保证，保证了数据不丢，不保证快照读。适用于对一致性要求相对高，没有快照读要求的场景。在对一致性要求较高的场景，较好的选择是使用 XA 的 Narayana 实现。如果底层存储节点的数据库隔离级是 Serializable，可以达到强一致性的语义。- BASE：在一致性和性能之间做了权衡，具体参考 Seata 官网。

8.8.3 使用方法

SphereEx-DBPlusEngine 中可通过配置文件 (server.yaml) 完成配置。主要是完成事务处理方式和分布式事务处理机制的配置。具体如下：

- defaultType

使用 LOCAL 事务

```
transaction:
  defaultType: LOCAL
```

使用 XA 事务

```
transaction:
  defaultType: XA
```

使用 BASE 事务

```
transaction:
  defaultType: BASE
```

- providerType

仅对 defaultType=XA 情况下适用。SphereEx-DBPlusEngine 支持多种分布式事务处理器：Atomikos、Narayana、Bitronix。建议用户选择使用 Narayana。

```
transaction:
  defaultType: XA
  providerType: Narayana
```

8.8.4 注意事项

本地事务

■ 支持项

- 完全支持非跨库事务，例如：仅分表，或分库但是路由的结果在单库中；
- 完全支持因逻辑异常导致的跨库事务。例如：同一事务中，跨两个库更新。更新完毕后，抛出空指针，则两个库的内容都能够回滚。

■ 不支持项

不支持因网络、硬件异常导致的跨库事务。例如：同一事务中，跨两个库更新，更新完毕后、未提交之前，第一个库宕机，则只有第二个库数据提交，且无法回滚。

XA 事务

■ 支持项

- 支持数据分片后的跨库事务；
- 两阶段提交保证操作的原子性和数据的强一致性；
- 服务宕机重启后，提交/回滚中的事务可自动恢复；
- 支持同时使用 XA 和非 XA 的连接池。

■ 不支持项

- 服务宕机后，在其它机器上恢复提交/回滚中的数据；
- MySQL 事务块内，SQL 执行出现异常，执行 Commit，数据保持一致；
- 配置 XA 事务后，存储单元名称最大长度不超过 45 个字符。

柔性事务

■ 支持项

- 支持数据分片后的跨库事务；
- 支持 RC 隔离级别；
- 通过 undo 快照进行事务回滚；
- 支持服务宕机后的，自动恢复提交中的事务。

■ 不支持项

不支持除 RC 之外的隔离级别。

■ 待优化项

SphereEx-DBPlusEngine 和 SEATA 重复 SQL 解析。

8.8.5 分布式事务实践

Narayana 使用方法

针对常见的分布式事务需求，SphereEx-DBPlusEngine 推荐使用 Narayana 实现的分布式事务，具体使用方法如下：

由于 Narayana 配置比较繁琐，DBPlusEngine-Proxy 提供了自动化配置功能，现在用户无需手动配置 jbossts-properties.xml 配置文件，DBPlusEngine 会自动根据用户在 server.yaml 里指定的 Narayana 事务配置，生成对应的 jbossts-properties.xml 配置文件。

- server.yaml

```
transaction:
  defaultType: XA
  providerType: Narayana
```

当使用 Narayana 作为 XA 事务管理器时，并且配置了使用 DB 方式存储 XA Recovery 信息，则 DBPlusEngine-Proxy 支持将故障的 Proxy 实例上尚未恢复的事务转移到其他 Proxy 上进行恢复。

配置如下：

```
transaction:
  defaultType: XA
  providerType: Narayana
  props:
    recoveryStoreUrl: jdbc:mysql://127.0.0.1:3306/jbossts?serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=true#
mysql8 使用 com.mysql.cj.jdbc.MysqlDataSourcerecoveryStoreDataSource: com.mysql.jdbc.jdbc2.optional.MysqlDataSource
    recoveryStoreUser: databaseUser
    recoveryStorePassword: databasePwd
```

通过 XA 语句控制的分布式事务

- 通过 XA START 可以手动开启 XA 事务，注意该事务完全由用户管理，DBPlusEngine 只负责将语句转发至后端数据库；
- 服务宕机后，需要通过 XA RECOVER 获取未提交或回滚的事务，也可以在 COMMIT 时使用 ONE PHASE 跳过 PERPARE。

MySQL [(none)]> use test1 Reading table information for completion of table and column names completion of table and column names You can turn off this feature to get a quicker startup with -A	MySQL [(none)]> use test2 Reading table information for completion of table and column names You can turn off this feature to get a quicker startup with -A
Database changed	Database changed
MySQL [test1]> XA START '61c052438d3eb'; Query OK, 0 rows affected (0.030 sec)	MySQL [test2]> XA START '61c0524390927'; Query OK, 0 rows affected (0.009 sec)
MySQL [test1]> update test set val = 'xatest1' where id = 1; where id = 1; Query OK, 1 row affected (0.077 sec)	MySQL [test2]> update test set val = 'xatest2'; Query OK, 1 row affected (0.010 sec)
MySQL [test1]> XA END '61c052438d3eb'; Query OK, 0 rows affected (0.006 sec)	MySQL [test2]> XA END '61c0524390927'; Query OK, 0 rows affected (0.008 sec)
MySQL [test1]> XA PREPARE '61c052438d3eb'; Query OK, 0 rows affected (0.018 sec)	MySQL [test2]> XA PREPARE '61c0524390927'; Query OK, 0 rows affected (0.011 sec)
MySQL [test1]> XA COMMIT '61c052438d3eb'; Query OK, 0 rows affected (0.011 sec)	MySQL [test2]> XA COMMIT '61c0524390927'; Query OK, 0 rows affected (0.018 sec)
MySQL [test1]> select * from test where id = 1; +----+-----+ id val	MySQL [test2]> select * from test where id = 1; +----+-----+ id val

```

+----+-----+      | +----+-----+
| 1 | xatest1 |      | 1 | xatest2 |
+----+-----+      | +----+-----+
1 row in set (0.016 sec)      | 1 row in set (0.129 sec)

MySQL [test1]> XA START '61c05243994c3';      | MySQL [test2]> XA START '61c052439bd7b';
Query OK, 0 rows affected (0.047 sec)      | Query OK, 0 rows affected (0.006 sec)

MySQL [test1]> update test set val = 'xarollback' where id = 1;      | MySQL [test2]> update test set val = 'xarollback'
' where id = 1;
Query OK, 1 row affected (0.175 sec)      | Query OK, 1 row affected (0.008 sec)

MySQL [test1]> XA END '61c05243994c3';      | MySQL [test2]> XA END '61c052439bd7b';
Query OK, 0 rows affected (0.007 sec)      | Query OK, 0 rows affected (0.014 sec)

MySQL [test1]> XA PREPARE '61c05243994c3';      | MySQL [test2]> XA PREPARE '61c052439bd7b';
Query OK, 0 rows affected (0.013 sec)      | Query OK, 0 rows affected (0.019 sec)

MySQL [test1]> XA ROLLBACK '61c05243994c3';      | MySQL [test2]> XA ROLLBACK '61c052439bd7b';
Query OK, 0 rows affected (0.010 sec)      | Query OK, 0 rows affected (0.010 sec)

MySQL [test1]> select * from test where id = 1;      | MySQL [test2]> select * from test where id = 1;
+----+-----+      | +----+-----+
| id | val |      | id | val |
+----+-----+      | +----+-----+
| 1 | xatest1 |      | 1 | xatest2 |
+----+-----+      | +----+-----+
1 row in set (0.009 sec)      | 1 row in set (0.083 sec)

MySQL [test1]> XA START '61c052438d3eb';
Query OK, 0 rows affected (0.030 sec)

MySQL [test1]> update test set val = 'recover' where id = 1;
Query OK, 1 row affected (0.072 sec)

MySQL [test1]> select * from test where id = 1;
+----+-----+
| id | val |
+----+-----+
| 1 | recover |
+----+-----+
1 row in set (0.039 sec)

MySQL [test1]> XA END '61c052438d3eb';
Query OK, 0 rows affected (0.005 sec)

MySQL [test1]> XA PREPARE '61c052438d3eb';
Query OK, 0 rows affected (0.020 sec)

MySQL [test1]> XA RECOVER;
+-----+-----+-----+-----+
| formatID | gtrid_length | bqual_length | data |
+-----+-----+-----+-----+
| 1 | 13 | 0 | 61c052438d3eb |
+-----+-----+-----+-----+
1 row in set (0.010 sec)

MySQL [test1]> XA RECOVER CONVERT XID;
+-----+-----+-----+-----+
| formatID | gtrid_length | bqual_length | data |
+-----+-----+-----+-----+
| 1 | 13 | 0 | 0x36316330353234333864336562 |
+-----+-----+-----+-----+

```

```
1 row in set (0.011 sec)
```

```
MySQL [test1]> XA COMMIT 0x36316330353234333864336562;  
Query OK, 0 rows affected (0.029 sec)
```

```
MySQL [test1]> XA RECOVER;  
Empty set (0.011 sec)
```

8.9 数据字典

SphereEx-DBPlusEngine 兼容底层数据库的部分数据字典，目前已适配兼容 MySQL 和 PG 的元数据库。查询元数据库可以直接在 JDBC 连接中使用 SQL 语句进行查询。但目前 SphereEx-DBPlusEngine 对系统库/表（如 information_schema、pg_catalog）支持有限，通过部分图形化数据库客户端连接 Proxy 时，可能客户端或 Proxy 会有错误提示。可以使用命令行客户端（mysql、psql、gsql 等）连接 Proxy 验证功能。

具体参见[数据字典](#)

8.10 数据类型

SphereEx-DBPlusEngine 支持 MySQL、PostgreSQL、openGauss 全部原生数据库数据类型，包括但不限于数值类型、日期类型、字符类型等。

8.11 运算符

SphereEx-DBPlusEngine 全部数据库运算符，包括但不限于逻辑运算符、算数运算符、比较运算符、位运算符、赋值运算符，并支持常规的运算符优先级。

8.12 SQL 兼容性

SQL 是使用者与数据库交流的标准语言。SQL 解析引擎负责将 SQL 字符串解析为抽象语法树，供 SphereEx-DBPlusEngine 理解并实现其增量功能。SphereEx-DBPlusEngine 目前支持 MySQL、PostgreSQL、SQLServer、Oracle、openGauss 以及符合 SQL92 规范的 SQL 方言。由于 SQL 语法的复杂性，目前仍然存在少量不支持的 SQL。此外，SphereEx-DBPlusEngine 为数据库提供了分布式协作的能力，同时将一部分数据库特性抽象到了上层，进行统一管理，以降低用户的使用难度。因此，对于统一提供的特性，原生的 SQL 将不再下发到数据库，并提示该操作不被支持，用户可使用 SphereEx-DBPlusEngine 提供的方式进行代替。

8.12.1 不兼容情况

MySQL

- 基础类语句

SQL
CLONE LOCAL DATA DIRECTORY = 'clone_dir'
INSTALL COMPONENT 'file://component1' , 'file://component2'
UNINSTALL COMPONENT 'file://component1' , 'file://component2'
REPAIR TABLE table_name
OPTIMIZE TABLE table_name
CHECKSUM TABLE table_name
CHECK TABLE table_name
SET RESOURCE GROUP group_name
DROP RESOURCE GROUP group_name
CREATE RESOURCE GROUP group_name TYPE=SYSTEM
ALTER RESOURCE GROUP rg1 VCPU=0-63

■ 用户和角色

SQL
CREATE USER 'username' @ 'localhost' IDENTIFIED BY 'password'
ALTER USER 'username' @ 'localhost' IDENTIFIED BY 'new_password'
DROP USER 'username' @ 'localhost'
CREATE ROLE 'rolename'
DROP ROLE 'rolename'
SHOW CREATE USER username
SET PASSWORD= 'password'
SET ROLE DEFAULT

■ 授权

SQL
GRANT ALL ON db1.* TO 'username' @ 'localhost'
GRANT SELECT ON db1.* TO 'username'
GRANT 'role1' , 'role2' TO 'user1' @ 'localhost'
REVOKE INSERT ON db1.* FROM 'username' @ 'localhost'
REVOKE 'role1' , 'role2' FROM 'user1' @ 'localhost'
REVOKE ALL PRIVILEGES,GRANT OPTION FROM user_or_role
SHOW GRANTS FOR 'username' @ 'localhost'
SHOW GRANTS FOR CURRENT_USER
FLUSH PRIVILEGES

PostgreSQL

SQL
CREATE TYPE avg_state AS (total bigint,count bigint)
CREATE AGGREGATE my_avg(int4)(stype=avg_state,sfunc=avg_transfn,finalfunc=avg_finalfn)
CREATE TABLE agg_data_2k AS SELECT g FROM generate_series(0,1999) g
CREATE SCHEMA schema_name
ALTER AGGREGATE alt_aggs(int) OWNER TO regress_alter_generic_user2
CREATE CONVERSION alt_conv1 FOR 'LATIN1' TO 'UTF8' FROM iso8859_1_to_utf8
CREATE FOREIGN DATA WRAPPER alt_fdw1
CREATE SERVER alt_ferv1 FOREIGN DATA WRAPPER alt_fdw1
CREATE LANGUAGE alt_lang1 HANDLER plpgsql_call_handler
CREATE STATISTICS alt_stat1 ON a,b FROM alt_regress_1
CREATE TEXT SEARCH DICTIONARY alt_ts_dict(template=simple)
CREATE RULE dev_view_test_ins AS ON INSERT TO dev_view_test DO INSTEAD INSERT INTO def_test SELECT new.*
ALTER TABLE alterlock SET(toast.autovacuum_enabled=off)
CREATE PUBLICATION pub1 FOR TABLE alter1.t1,ALL TABLES IN SCHEMA alter2

openGauss

SQL
CREATE TYPE avg_state AS (total bigint,count bigint)
CREATE AGGREGATE my_avg(int4)(stype=avg_state,sfunc=avg_transfn,finalfunc=avg_finalfn)
CREATE TABLE agg_data_2k AS SELECT g FROM generate_series(0,1999) g
CREATE SCHEMA schema_name
ALTER AGGREGATE alt_aggs(int) OWNER TO regress_alter_generic_user2
CREATE CONVERSION alt_conv1 FOR 'LATIN1' TO 'UTF8' FROM iso8859_1_to_utf8
CREATE FOREIGN DATA WRAPPER alt_fdw1
CREATE SERVER alt_ferv1 FOREIGN DATA WRAPPER alt_fdw1
CREATE LANGUAGE alt_lang1 HANDLER plpgsql_call_handler
CREATE STATISTICS alt_stat1 ON a,b FROM alt_regress_1
CREATE TEXT SEARCH DICTIONARY alt_ts_dict(template=simple)
CREATE RULE dev_view_test_ins AS ON INSERT TO dev_view_test DO INSTEAD INSERT INTO def_test SELECT new.*
ALTER TABLE alterlock SET(toast.autovacuum_enabled=off)
CREATE PUBLICATION pub1 FOR TABLE alter1.t1,ALL TABLES IN SCHEMA alter2

8.13 SQL 管理

8.13.1 SQL 慢查询

背景信息

慢查询日志功能，用于将执行耗时超过一定时间的 SQL 语句进行记录，便于 DBA 和开发人员识别可能存在问题的 SQL 语句，是数据库和 SQL 管理的重要参考来源。

配置及参数

慢查询日志功能基于 Agent 技术实现，仅适用于 SphereEx-DBPlusEngine 且启用 Agent 的场景。
 由于慢查询日志是基于 Agent 的功能，因此以下配置位于 agent.yaml 中：

```
plugins:
Logging:
props:
slow_query_log: true
long_query_time: 5000
```

- slow_query_log：是否开启慢查询日志功能，默认值为 true。
- long_query_time：慢查询时间阈值，执行耗时超过该阈值的 SQL 语句才会被记入慢查询日志中。该配置单位为毫秒 (ms)，默认值为 5000。

慢查询日志格式

timestamp: {time} db: {db database user: {user} host: {host} query_time: {query time} {sql} - timestamp: 日志记录产生的时间; - db: 数据库名称; - user: 当前连接中使用的用户名; - host: 客户端访问地址; - query_time: SQL 执行耗时, 单位为 ms; - sql: 出现慢查询时, 客户端发送的 SQL 语句。

慢查询日志示例

```
timestamp: 2022-07-01 00:00:00.000 db: sharding_db user: root host: 127.0.0.1 query_time: 159
CREATE TABLE `t_order` (
  `order_id` bigint(20) NOT NULL AUTO_INCREMENT,
  `user_id` int(11) NOT NULL,
  `status` varchar(50) COLLATE utf8mb4_bin DEFAULT NULL,
  PRIMARY KEY (`order_id`)
)
```

8.13.2 SQL 执行计划

SphereEx-DBPlusEngine 支持两种查看执行计划的方式，一种是传统的 EXPLAIN 方式，一种是 PREVIEW 方式。前者可查看下发到存储节点的 SQL 真实执行计划；后者可查看分布式情况下的数据路由情况。

■ EXPLAIN

EXPLAIN 使用很简单，直接在命令行中即可。对于存储节点使用 MySQL 数据库，这一命令可直接调用底层数据库的对应命令；对于存储节点使用 openGauss 数据库，可使用 EXPLAIN PERFORMANCE

```
mysql> explain select * from t_single0;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	t_single0	NULL	ALL	NULL	NULL	NULL	NULL	4	100.0	NULL


```
mysql> explain select * from t_shard1;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	t_shard1_0	NULL	ALL	NULL	NULL	NULL	NULL	2	100.0	NULL

说明：对于分片表来说，上述命令会显示某一分片上的执行计划，尚无法获得完整的分布式执行计划。

■ PREVIEW

通过 PREVIEW 命令进行查看，可显示具体执行的 SQL 及对应的数据分片。

```
mysql> preview select * from t_shard1;
```

data_source_name	actual_sql
test1_3307	select * from t_shard1_0 UNION ALL select * from t_shard1_2
test1_3308	select * from t_shard1_1 UNION ALL select * from t_shard1_3

8.13.3 SQL Hint

SQL HINT 就是通过 SQL 语句上增加注释，从而实现强制路由的一种 Hint 方式。它降低了用户改造代码的成本。通过注释的方式我们可以方便地将 SQL 直接送达指定数据库执行而忽视其它分片逻辑。以多租户场景为例，用户不用再配置复杂的分库逻辑，也无需改造业务逻辑，只需要将指定库添加到注释信息中即可。注释格式暂时只支持 `/* */`，内容需要以 `SHARDINGSPHERE_HINT:` 开始。

使用方法

■ 注释

SQL HINT 的使用非常简单。

第一步打开注释解析开关。将 `sqlCommentParseEnabled` 设置为 `true`。

第二步在 SQL 上增加注释即可。目前 SQL HINT 支持指定数据源路由和主库路由。

■ 数据源路由

目前只支持路由至一个数据源。内容需要以 `ShardingSphere hint:` 开始，属性名为 `dataSourceName`。

```
/* ShardingSphere hint: dataSourceName=ds_0 */  
SELECT * FROM t_order;
```

■ SQL 改写路由

支持开启或禁止 SQL 改写路由。内容需要以 `ShardingSphere hint:` 开始，属性名为 `SKIP_SQL_REWRITE`。

```
/* ShardingSphere hint: SKIP_SQL_REWRITE=true */  
SELECT * FROM t_order;
```

■ 主库路由

支持路由到主库，内容需要以 `ShardingSphere hint:` 开始，属性名为 `WRITE_ROUTE_ONLY`。

```
/* SHARDINGSPHERE_HINT: WRITE_ROUTE_ONLY=true */  
SELECT * FROM t_order;
```

■ 分片路由支持路由到某一片片，可选的属性包括：

- `{table}.SHARDING_DATABASE_VALUE`：用于添加 `{table}` 表对应的数据源分片键值，多个属性使用逗号分隔；
- `{table}.SHARDING_TABLE_VALUE`：用于添加 `{table}` 表对应的表分片键值，多个属性使用逗号分隔。注意，在分库不分表情况下，强制路由至某一个分库时，可使用 `SHARDING_DATABASE_VALUE` 方式设置分片，无需指定 `{table}`。

8.14 DistSQL

8.14.1 背景

DistSQL (Distributed SQL) 是 DBPlusEngine 特有的操作语言。它与标准 SQL 的使用方式完全一致，用于提供增量功能的 SQL 级别操作能力。详情参考[速查表:DistSQL](#)。

8.14.2 挑战

灵活的规则配置和资源管控能力是 DBPlusEngine 的特点之一。

DistSQL 让用户可以像操作数据库一样操作 DBPlusEngine，使其从面向开发人员的框架和中间件转变为面向运维人员的数据库产品。

DistSQL 细分为 RDL、RQL、RAL 和 RUL 四种类型。

- RDL (Resource & Rule Definition Language) 负责资源和规则的创建、修改和删除；
- RQL (Resource & Rule Query Language) 负责资源和规则的查询和展现；
- RAL (Resource & Rule Administration Language) 负责 Hint、事务类型切换、分片执行计划查询等管理功能。
- RUL (Resource Utility Language) 为 SphereEx-DBPlusEngine 的工具类语言，提供 SQL 解析、SQL 格式化、执行计划预览等功能。

8.14.3 目标

打破中间件和数据库之间的界限，让开发者像使用数据库一样使用 DBPlusEngine，是 DistSQL 的设计目标。

8.14.4 注意事项

DistSQL 只能用于 DBPlusEngine-Proxy，DBPlusEngine-Driver 暂不提供。

8.14.5 语法规则

在 DistSQL 语句中，除关键字外，其余元素的输入格式应符合以下规则。

8.14.6 标识符

标识符代表 SQL 语句中的一个对象，包括：

- 数据库名称
- 表名
- 列名
- 索引名称
- 资源名称
- 规则名称
- 算法名称标识符中允许使用的字符有：[a-z,A-Z,0-9,_]（字母、数字、下划线），且应以字母开头。

当标识符中出现关键字或特殊字符时，使用反引号（`）。

8.14.7 字面量

字面量包括字符串、整数值和布尔值：

字符串：是由单引号 (') 或双引号 (") 括起来的字符序列；

整数值：一般为正整数，如 0-9；

说明：部分 DistSQL 语法允许负值，此时可在数字前加负号 (-)，如 -1。

布尔值：TRUE 或 FALSE，大小写不敏感。

8.14.8 一致性

DistSQL 语句执行后，会更新存储在治理中心的元数据，并同步更新计算节点内元数据。当设置参数 meta-data-consistency-enabled=true 时，可实现一致性更新。

8.15 安全管理

8.15.1 用户管理

SphereEx-DBPlusEngine 提供多种用户，以供不同平台及场景使用。

用户	说明	使用方式
Engine 用户	SphereEx-DBPlusEngine 平台提供的用户，也是用户与平台交互的用户。可理解为“数据库用户”。	应用 APP 连接 Engine 平台使用-MySQL、PostgreSQL 的命令行工具或图形管理工具
Console 用户	仅供 Console 平台使用，可理解为管理控制台的自有用户	通过 Console 平台使用
数据库用户	SphereEx-DBPlusEngine 平台连接数据库使用的用户，用于读写数据库对象及参数、拓扑等	不能独立使用
第三方用户	SphereEx-DBPlusEngine 平台支持对接到第三方用户（如 LDAP）	视同 Engine 用户

8.15.2 角色管理

SphereEx-DBPlusEngine 提供多种角色，以供不同平台及场景使用。

角色	说明	使用方式
Engine 角色	SphereEx-DBPlusEngine 平台提供角色	
Console 角色	Console 平台提供角色	与 Console 用户配合使用

8.15.3 认证登录

下面说明特指 SphereEx-DBPlusEngine 用户。

基本概念

SphereEx-DBPlusEngine 拥有严格的登录认证机制，只有经过身份认证的用户才可以成功建立连接。为保障用户数据和分布式配置信息的安全，SphereEx-DBPlusEngine 提供了用户认证的功能，且该功能不可关闭，否则，未经认证的客户端连接将被拒绝。当前，SphereEx-DBPlusEngine 已经支持了多种用户认证协议，包括：

- MySQL 客户端：mysql_native_password、mysql_clear_password
- PostgreSQL 客户端：MD5、password
- openGauss 客户端：scram-sha-256、MD5

同时，为方便企业用户进行统一身份管理，SphereEx-DBPlusEngine 也提供了 LDAP (Lightweight Directory Access Protocol) 认证方式，LDAP 认证已支持 MySQL 和 PostgreSQL 客户端。实际上，用户在使用 SphereEx-DBPlusEngine 时不需要考虑选择何种协议，协议的协商过程是在 SphereEx-DBPlusEngine 与客户端之间自动完成的。当使用 MySQL 客户端时，默认使用 mysql_native_password。仅当用户需要进行 LDAP 认证时，SphereEx-DBPlusEngine 会要求客户端切换为 mysql_clear_password 协议进行通信。当使用 PostgreSQL 客户端时，默认使用 MD5。仅当用户需要进行 LDAP 认证时，SphereEx-DBPlusEngine 会要求客户端切换为 password 协议进行通信。

密码认证

默认情况下，SphereEx-DBPlusEngine 使用密码认证方式，登录用户需提供正确的用户名和密码。

特别的，由于 Proxy 支持多种数据库协议（如 MySQL、PostgreSQL 等），当用户应用不同的数据库客户端时，Proxy 能够自动适配密码通信协议，在复杂场景下为用户提供一致的安全体验。

主机限制

除了提供密码认证登录外，还可以为 SphereEx-DBPlusEngine 的用户限制登录主机地址，以提高安全等级。如：

```
- !AUTHORITY
users:
- user: root@127.0.0.1
  password: root
```

以上配置指定了 root 用户只能从 127.0.0.1 这个地址访问 Proxy，从其他地址登录时，即使密码正确也将被拒绝。

LDAP 认证

为方便企业用户进行统一认证管理，SphereEx-DBPlusEngine 也提供了 LDAP (Lightweight Directory Access Protocol) 认证方式，LDAP 认证现已支持 MySQL 和 PostgreSQL 客户端。同时，SphereEx-DBPlusEngine 允许用户以非常灵活的方式来接入 LDAP，例如：

- 可以配置为默认使用 LDAP，这样所有用户都通过 LDAP 认证；
- 支持为用户配置 auth 属性，指定用户使用密码或 LDAP 认证，各个用户可以使用不同方式；
- 各个用户可以使用不同的 LDAP 认证器，即对接不同的 LDAP 服务；
- 支持为用户指定 DN 模板，满足复杂场景的需要；
- 支持 LDAPS 协议，可进一步提升安全等级。

8.15.4 权限管理

定义

SphereEx-DBPlusEngine 为数据库提供了分布式协作的能力，同时将一部分数据库特性抽象到了上层，进行统一管理，以降低用户的使用难度，同时提高运行效率。权限控制，就是其中的一项能力。将权限控制交由 SphereEx-DBPlusEngine 统一管理，至少有以下几方面的好处：- 避免接入异构资源时为用户带来困惑，无需担心该使用哪种方言进行管理；- 使用逻辑库和逻辑表进行授权管理，与下层真实库表隔离，更便于用户理解；- 避免数据库资源变化导致的授权信息不一致，不会产生信息同步的消耗。于是，为了让权限控制更易用，SphereEx-DBPlusEngine 团队倾心打造了全新的权限控制体系。

概念

■ 用户

用户 (user) 是 SphereEx-DBPlusEngine 的使用者。

■ 初始用户

初始用户是指在 SphereEx-DBPlusEngine 启动前，通过配置文件设定的用户。

■ 普通用户

与初始用户相对应，普通用户是在 SphereEx-DBPlusEngine 运行过程中被动态创建的用户。

■ 角色

角色 (role)，是被命名的、一定数量的权限的集合。以角色为基础的权限控制，能够简化对用户的权限管理过程。

■ 权限

权限 (privilege) 是指用户对特定目标执行操作的权力。

■ DistSQL

DistSQL (Distributed SQL) 是 DBPlusEngine 特有的操作语言，在 DBPlusEngine 将权限控制能力进行抽象统一之后，提供了专有的 DistSQL 语法，方便管理员对用户和权限的管理维护。

■ DML

数据操纵语言，包含 INSERT、SELECT、UPDATE 和 DELETE 语句。

■ DDL

数据定义语言，包含 CREATE、ALTER、DROP 和 TRUNCATE 语句。

对系统影响

■ 细粒度权限管理

可精确控制授予每个用户的库级、表级、列级的操作权限。

■ 统一的交互语言

使用 SphereEx-DBPlusEngine 特有的 DistSQL 进行用户和权限管理，无论存储节点选择 MySQL、PostgreSQL、OpenGauss 还是 Oracle，都可以进行无差别权限控制。

■ 权限管控实时生效

对用户或授权的变更实时生效，无需重启 SphereEx-DBPlusEngine。

■ 授权信息在集群中自动同步

变更用户和授权信息，集群中其他计算节点也能实时收到该变化，完成用户的授权更新，管理员无需在多个节点进行重复操作，方便集群管理

原理

■ 权限存储

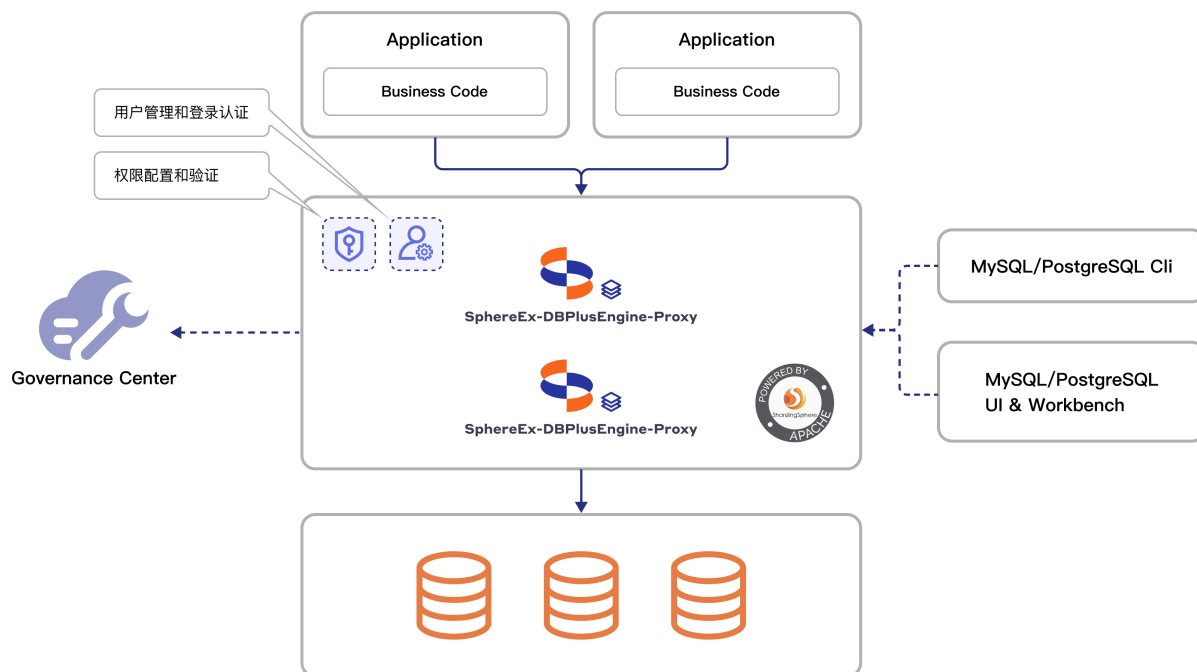


Fig. 8: 权限存储原理图

在 SphereEx-DBPlusEngine 的体系架构中，计算节点（DBPlusEngine-Proxy）是无状态的，并不提供数据存储能力，因此，用户账户及授权信息都将存储在治理中心。同时，借治理中心的能力，能够将信息实时分发给集群中的多个计算节点，这将大大减少用户在使用集群时的维护成本，提供管理效率。另一方面，由于提供了统一的权限管理机制，对于接收到的原生 DCL 语句，SphereEx-DBPlusEngine 不再转发到下层存储节点，而将会给出不支持的提示信息。用户须使用 SphereEx-DBPlusEngine 提供的 DistSQL 进行账户和授权的管理。

■ 权限提供者

SphereEx-DBPlusEngine 使用可插拔架构进行功能的组织和扩展。其中，权限引擎为用户提供了多种不同的权限提供者说明：权限提供者由管理员在 SphereEx-DBPlusEngine 启动前指定。

■ 鉴权流程

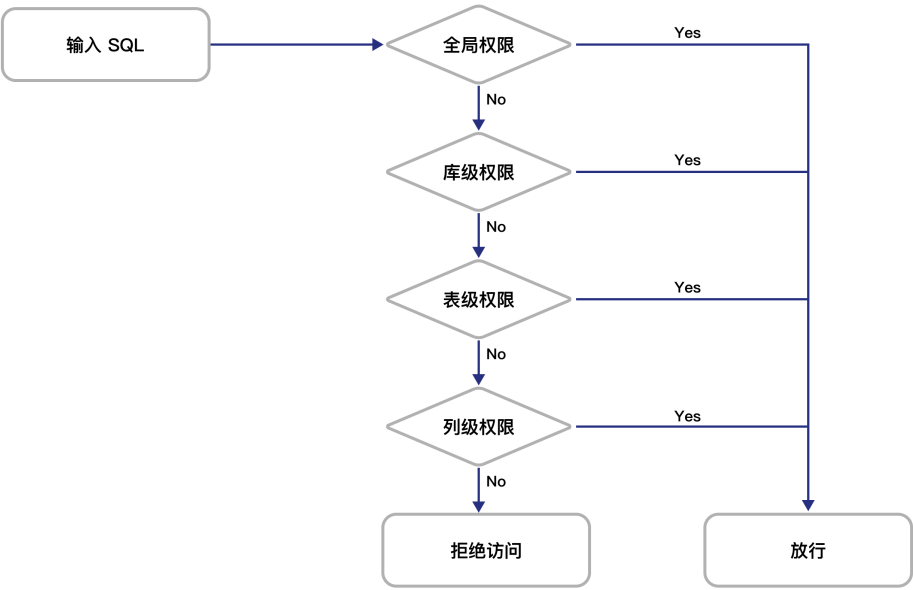


Fig. 9: 鉴权流程图

在 SphereEx-DBPlusEngine 中，权限按照自顶向下的顺序逐级校验，当用户拥有上层权限时，不再向下进行检查，以保证鉴权效率。例如：- 若用户拥有全局的 SELECT 权限，则在进行 SELECT 操作时，无需检查该用户是否拥有目标库表的 SELECT 授权；- 若用户拥有库级别的 INSERT 权限，则在进行 INSERT 操作时，无需检查该用户是否拥有目标表的 INSERT 授权。以此类推。

例如：- 若用户拥有全局的 SELECT 权限，则在进行 SELECT 操作时，无需检查该用户是否拥有目标库表的 SELECT 授权；- 若用户拥有库级别的 INSERT 权限，则在进行 INSERT 操作时，无需检查该用户是否拥有目标表的 INSERT 授权。以此类推。

权限分级

授权项	SEL ECT	INS ERT	UPD ATE	DEL ETE	CRE ATE	DRO P	ALT ER	IND EX	CRE ATE USER	SUP ER
全 局 权 限	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
对 象 权 限/库	Y	Y	Y	Y	Y	Y	Y	Y	■	■
对 象 权 限/库	Y	Y	Y	Y	Y	Y	Y	Y	■	■
对 象 权 限/表	Y	Y	Y	Y	Y	Y	Y	Y	■	■
对 象 权 限/列	Y	Y	Y	■	■	■	■	■	■	■

全局权限

全局权限，是指用户获得的授权不区分目标对象，用户可对任意的逻辑库、逻辑表执行对应操作。例如，以下指令将全局的 INSERT、SELECT、UPDATE 和 DELETE 权限赋予用户 'sharding' @ '%'，则该用户能够对任何逻辑库中的表执行 DML 操作。以下两个语句等效：

```
GRANT DIST INSERT,SELECT,UPDATE,DELETE TO 'sharding'@'%';
GRANT DIST INSERT,SELECT,UPDATE,DELETE TO sharding;
```

需要注意的是，全局权限包含两项特殊权限：CREATE_USER 和 SUPER。

其中，获得 CREATE_USER 授权的用户，可执行以下操作：

操作	说明
CREATE USER	创建用户
ALTER USER	修改用户
DROP USER	删除用户
CREATE ROLE	创建角色
DROP ROLE	删除角色
REVOKE ALL PRIVILEGES	撤销用户或角色的全部授权

SUPER 代表的是数据库系统的最高权限，默认情况下，SphereEx-DBPlusEngine 启动前配置的初始用户拥有 SUPER 授权。

对象权限

对象权限，指授予的权限受作用范围限制，用户不可在授权范围之外执行相应操作。

对象权限的作用范围可以是所有逻辑库，也可以指定单个或多个逻辑库及库中的表和列。

例如，以下指令将逻辑库 sharding_db 中 t_order 表的所有权限授予用户 'sharding' @ '%'，此后该用户可以对 sharding_db.t_order 表执行操作。但若无额外授权，该用户不能操作 sharding_db 中的其它表。

```
sql> GRANT DIST ALL ON sharding_db.t_order TO sharding;
```

DistSQL 清单（权限相关）

完整的权限管理 DistSQL 清单请参考[DistSQL 用户管理](#)

说明：- 为用户和角色「授权/撤销授权」的语法一致，将 'user' @ 'host' 替换为 role 即可；- REVOKE DIST ALL 会同时撤销赋予给用户的所有角色。

使用案例：权限配置初始化

■ 适用场景

权限引擎根据 server.yaml 中配置的权限规则来执行系统的初始化工作。

■ 数据规划

- users 用于指定初始用户，如这里将 root@% 设置为初始用户；
- privilege 中的 type 用于指定选用的服务提供者，如这里配置了企业级的权限提供者 SphereEx:PERMITTED。

■ 注意事项

- 初始用户默认具备 SUPER 权限。
- 若通过 DistSQL 对初始用户显示赋予非 SUPER 权限，则该初始用户将失去 SUPER 授权。
- 需再次赋予 SUPER 授权，需要使用 GRANT DIST SUPER TO user 语句。

■ 操作步骤

配置格式如下：

```
authority:
users:
- user: root@%
  password: root
- user: sharding
  password: sharding
privilege:
type: SphereEx:PERMITTED
```

使用案例：不使用角色管理

■ 适用场景

某应用系统为开发和运维人员提供了不同级别的 SphereEx-DBPlusEngine 账户，其中开发人员仅可以执行 DML 指令，运维人员则可以执行 DML + DDL 指令，另有一个 root 用户作为最高管理者。

■ 数据规划

全部账号需求如下：

用户名	使用人员	所需权限
root	最高管理员	SUPER
zhangsan	开发-张三	DML
wangwu	开发-王五	DML
develop_test	开发测试	DML
operator_1	运维人员-1	DML+DDL
operator_2	运维人员-2	DML+DDL

其中，root 用户为初始用户。

■ 操作步骤

1. 依次创建各个开发和运维用户，密码根据实际情况进行设置。

-- 不限制登录主机，省略 host 配置

```
sql> CREATE DIST USER zhangsan IDENTIFIED BY '123456';
sql> CREATE DIST USER wangwu IDENTIFIED BY '123456';
sql> CREATE DIST USER develop_test IDENTIFIED BY '123456';
sql> CREATE DIST USER operator_1 IDENTIFIED BY '123456';
sql> CREATE DIST USER operator_2 IDENTIFIED BY '123456';
```

2. 为开发用户授权。

```
sql> GRANT DIST INSERT,SELECT,UPDATE,DELETE TO zhangsan;
sql> GRANT DIST INSERT,SELECT,UPDATE,DELETE TO wangwu;
sql> GRANT DIST INSERT,SELECT,UPDATE,DELETE TO develop_test;
```

3. 为运维用户授权。

```
sql> GRANT DIST INSERT,SELECT,UPDATE,DELETE,CREATE,ALTER,DROP,TRUNCATE TO operator_1;
sql> GRANT DIST INSERT,SELECT,UPDATE,DELETE,CREATE,ALTER,DROP,TRUNCATE TO operator_2;
```

4. 若需新增开发用户，则重复以下两步。

■ 新建用户

```
sql> CREATE DIST USER new_developer IDENTIFIED BY '123456';
```

■ 授权

```
sql> GRANT DIST INSERT,SELECT,UPDATE,DELETE TO new_developer;
```

5. 若需新增运维用户，则重复以下两步。

- 新建用户

```
sql> CREATE DIST USER new_operator IDENTIFIED BY '123456';
```

- 授权

```
sql> GRANT DIST INSERT,SELECT,UPDATE,DELETE,CREATE,ALTER,DROP,TRUNCATE TO new_operator;
```

使用案例：使用角色管理

- 适用场景

某应用系统为开发和运维人员提供了不同级别的 SphereEx-DBPlusEngine 账户，其中开发人员仅可以执行 DML 指令，运维人员则可以执行 DML + DDL 指令，另有一个 root 用户作为最高管理者。

- 数据规划

全部账号需求如下：

用户名	使用人员	所需权限
root	最高管理员	SUPER
zhangsan	开发-张三	DML
wangwu	开发-王五	DML
develop_test	开发测试	DML
operator_1	运维人员-1	DML+DDL
operator_2	运维人员-2	DML+DDL

- 操作步骤

1. 依次创建各个开发和运维用户，密码根据实际情况进行设置。

不限制登录主机，省略 host 配置

```
sql> CREATE DIST USER zhangsan IDENTIFIED BY '123456';
sql> CREATE DIST USER wangwu IDENTIFIED BY '123456';
sql> CREATE DIST USER develop_test IDENTIFIED BY '123456';
sql> CREATE DIST USER operator_1 IDENTIFIED BY '123456';
sql> CREATE DIST USER operator_2 IDENTIFIED BY '123456';
```

2. 创建两个角色，develop_dml 和 operate_ddl。

```
sql> CREATE DIST ROLE develop_dml;
sql> CREATE DIST ROLE operate_ddl;
```

3. 为角色授权。

```
sql> GRANT DIST INSERT,SELECT,UPDATE,DELETE TO develop_dml;
sql> GRANT DIST INSERT,SELECT,UPDATE,DELETE,CREATE,ALTER,DROP,TRUNCATE TO operate_ddl;
```

4. 将开发角色赋予用户，这样用户就具有了角色所拥有的权限。

```
sql> GRANT DIST develop_dml TO zhangsan;
sql> GRANT DIST develop_dml TO wangwu;
sql> GRANT DIST develop_dml TO develop_test;
```

5. 为运维角色赋予用户。

```
sql> GRANT DIST operate_ddl TO operator_1;
sql> GRANT DIST operate_ddl TO operator_2;
```

6. 若需新增开发用户，则重复以下两步。

-- 新建用户

```
sql> CREATE DIST USER new_developer IDENTIFIED BY '123456';
```

-- 授权

```
sql> GRANT DIST develop_dml TO new_developer;
```

7. 若需新增运维用户，则重复以下两步。

-- 新建用户

```
sql> CREATE DIST USER new_operator IDENTIFIED BY '123456';
```

-- 授权

```
sql> GRANT DIST operate_ddl TO new_operator;
```

8.15.5 DistSQL 权限控制

DistSQL，作为 SphereEx-DBPlusEngine 支持的一种扩展 SQL。SphereEx-DBPlusEngine 也提供对应的权限控制。

权限分级

DistSQL 权限的分级与标准数据库权限类似，同样拥有全局权限、库级对象权限、表级对象权限。目前 DistSQL 不存在列级操作，因此权限分级中也没有列级权限。DistSQL 权限的授权项相较于标准的数据库权限种类更多。DistSQL 权限授权项由操作 + 操作对象组成，其组合方式如下。

\	CREATE	ALTER	DROP	SHOW
RESOURCE	Y	Y	Y	Y
SHARDING	Y	Y	Y	Y
READWRITE_SPLITTING	Y	Y	Y	Y
ENCRYPT	Y	Y	Y	Y
DB_DISCOVERY	Y	Y	Y	Y
SHADOW	Y	Y	Y	Y
SINGLE_TABLE	Y	Y	Y	Y

除了上述表格中展示的授权项，还提供 RDL、RQL、RAL 以及 RUL 语法类型作为授权项。

全局权限

全局权限，是指用户获得的授权不区分目标对象，用户可对任意的逻辑库、逻辑表执行对应操作。

例如，以下指令将全局的 SHOW SHARDING 权限赋予用户 'sharding' @ '%'，则该用户能够对任何逻辑库中的分片规则执行 SHOW 操作。

```
sql> GRANT DIST SHOW SHARDING TO 'sharding'@'%';
```

另外，因为 DistSQL 授权项较多，对于一次授予多个授权项时，可以使用语法类型作为授权项。例如，以下指令将所有操作对象全局的 SHOW 权限赋予用户 'sharding' @ '%'，则该用户能够对任何逻辑库中的任何操作对象执行 SHOW 操作。

```
sql> GRANT DIST RQL SHARDING TO 'sharding'@'%';
```

对象权限

对象权限，指授予的权限受作用范围限制，用户不可在授权范围之外执行相应操作。对象权限的作用范围可以是所有逻辑库，也可以指定单个或多个逻辑库及库中的表。例如，以下指令将逻辑库 sharding_db 中分片规则 t_order 的创建和修改权限授予用户 'sharding' @ '%'，此后该用户可以对 sharding_db.t_order 规则执行创建和修改操作。但若无额外授权，该用户不能操作 sharding_db 中的其它规则。

```
sql> GRANT DIST CREATE SHARDING, ALTER SHARDING ON sharding_db.t_order TO 'sharding'@'%';
```

8.15.6 管理安全

在 SphereEx-DBPlusEngine 中，用户能够通过 DistSQL 执行多种维度的管理操作，包括但不限于：

- Proxy 配置管理，如事务类型、日志开关等；
- 逻辑库管理；
- 存储资源管理；
- 数据分片规则管理；
- 读写分离规则管理；
- 加解密规则管理；
- 数据库发现规则管理；
- 影子库规则管理；
- 元数据查看等。

由于 DistSQL 功能强大，数据库管理员可为不同用户分配不同的 DistSQL 权限，以实现 SphereEx-DBPlusEngine 管理安全。例如：

```
sql> GRANT DIST SHOW SHARDING ON sharding_db.* TO 'sharding'@'%';
```

通过以上授权语句，将逻辑库 sharding_db 中的“查看分片规则”的权限授予 'sharding@%'，则该用户能够在 sharding_db 中执行 SHOW SHARDING TABLE RULES、SHOW SHARDING BINDING TABLE RULES 等 SHARDING 相关的 RQL，但无法执行其他未授权的 DistSQL。

例如 'sharding@%' 用户此时执行 CREATE SHARDING TABLE RULE 语句，将会得到异常提示：

```
Access denied for operation [CREATE] of subject sharding_db.table_name:SHARDING.]
```

若要向 'sharding@%' 用户授予所有 DistSQL 权限，可以通过如下方式：

```
sql> GRANT DIST RDL,RQL,RAL ON sharding_db.* TO 'sharding'@'%';
```

8.15.7 访问安全

数据访问安全，是企业级数据库必备的能力之一。SphereEx-DBPlusEngine 作为分布式数据库集群的入口，为用户提供了全面的访问控制能力。与传统集中式数据库或单协议数据库不同，SphereEx-DBPlusEngine 拥有管理多类型底层数据库和对接多协议客户端的能力，在进行访问控制时就会面临诸多挑战：

- 不同数据库类型拥有不同的逻辑概念；
- 不同数据库类型使用不同的方言；
- 不同数据库提供了不同的存储结构。

为了给用户提供一致的安全体验，SphereEx-DBPlusEngine 屏蔽了底层数据库的差异，提供统一易用的安全体系，具有如下特点：

- 细粒度权限管理：支持库级、表级、列级访问控制；

- 统一的交互语言：使用 DistSQL 进行用户和权限的管理，适用于不同的数据库协议；
- 存储独立：权限的信息保存在 DBPlusEngine 的治理中心，不依赖于底层数据库；
- 实时生效：对用户和权限的管控实时生效，无需重启或手动刷新。

例如，若管理员向 ‘sharding@%’ 授予 sharding_db 中 t_order 表的查询和写入权限，可以执行这样的 DistSQL：

```
sql> GRANT DIST SELECT, INSERT ON sharding_db.t_order TO 'sharding' '@%';
```

操作完成之后，‘sharding@%’ 用户立即获得相应授权。若用户执行未授权的操作，如 DELETE，将会收到拒绝提示：

```
sql> DELETE FROM sharding_db.t_order WHERE id = 1;
Access denied for operation [DELETE] of subject sharding_db.t_order]
```

■ SSL 说明

目前 SphereEx-DBPlusEngine 尚不支持 SSL 的访问方式。如果后端数据库启用 SSL 访问，可在 SphereEx-DBPlusEngine 中配置后端使用 SSL 方式连接。

8.15.8 存储安全

近年来，数据安全和隐私保护越来越受到重视。而数据加密是有效地解决上述问题的手段之一。

安全风险	磁盘加密	文件加密	数据库加密 (TDE)	数据库加密 (三方加固)	应用层加密
防止磁盘丢失引起数据泄露	Y	Y	Y	Y	Y
防止系统 root 账户和管理员账户访问	N	Y	Y	Y	Y
控制数据库管理员访问数据	N	N	N	Y	Y
抵抗定向威胁供给 APT 造成数据泄露	N	Y	Y	Y	Y
提供细粒度访问记录、遵从法规	N	Y	N	Y	Y
确保备份数据和数据快照存储	N	Y	Y	Y	Y
非结构化数据和文件的保护	Y	Y	N	N	Y
防止硬件和数据库厂商“偷窥”	N	Y	N	Y	Y

■ 磁盘加密

磁盘采用的块级别加密技术，例如 AWS 的 EBS，阿里云的 ECS 等都支持磁盘加密。这种加密最大的好处在于，它对操作系统是透明的。性能在加密后较加密前有所降低，根据上层应用的不同性能下降幅度各异。

■ 文件加密

通过堆叠在其它文件系统之上（如 Ext2, Ext3, ReiserFS, JFS 等），为应用程序提供透明、动态、高效和安全的加密功能。典型的是用于加密指定的目录。需要关注的是这种加密方式可能会产生较大的性能损失。

■ 数据库加密-TDE

透明数据加密 TDE，是数据库提供了一种加密技术，即对数据文件执行实时 I/O 加密和解密。数据在写入磁盘之前进行加密，从磁盘读入内存时进行解密。TDE 不会增加数据文件的大小，开发人员无需更改任何应用程序。其对应密钥管理也是由数据库提供的 API 或组件实现，应用透明。在某些场景下磁盘或系统无法对用户开放（如云环境）的条件下，这种方式就比较适合。

■ 数据库加密-三方加固

数据库加密还有种方式是采用对数据库进行三方加固的方式，即将第三方专业数据库加密厂商的产品内置在数据库之中，提供透明数据加密能力。所谓透明是指，用户应用系统不需要做改造即可使用，且具有权限的用户看到的是明文数据，完全无感。此外，还可以增强原有数据库的安全能力，如提供三权分立、脱敏展示等。

■ 应用层加密

应用层加密，可以说是一种终极方案，其可保证在数据到达数据库之前，就已经做了数据加密，可实时保护用户敏感数据。这里关键需要提供应用透明性，保证应用无需改造或仅需少量改造。这种方式完全由用户自己控制，无需信任任何三方厂商提供的数据安全保障，得到充分的自由度和灵活性。例如可以跨多数据库提供统一安全加密策略等。

SphereEx-DBPlusEngine 提供的数据库加密能力是属于应用层加密。但与传统的应用加密不同，SphereEx-DBPlusEngine 提供无感的加密能力。对于前端用户而言，只需考虑对数据库的访问，无需关注加密细节。具体参见[数据加密](#)

8.15.9 使用安全

脱敏使用

数据脱敏，尚不支持。

8.15.10 安全审计

全量审计日志，是指开启该功能后，系统将记录所有执行的 SQL 语句，并包含语句对应的数据库、用户、访问地址，访问耗时等信息，便于企业进行审计操作。

参数解释

general_query_log：全量审计日志仅有一个参数，值为 true 时启用全量日志，值为 false 时停用该功能。

前提条件

与“慢查询日志”功能相似，全量审计日志的实现也是基于 Agent 的，因此该功能仅适用于 SphereEx-DBPlusEngine 且启用 Agent 的场景。

配置示例

全量审计日志是基于 Agent 的功能，因此以下配置位于 agent.yaml 中：

```
plugins:
Logging:
props:
# Whether to enable general query log.
general_query_log: false
# Whether to enable slow query log.
#slow_query_log: true
# Long query threshold, in
```

其中，general_query_log 配置为 false，表示不启用全量日志。当需要启用全量日志时，将 general_query_log 配置 true，并重新启动 SphereEx-DBPlusEngine。

全量审计日志格式

```
db: {database} user: {user} host: {host} query_time: {query time} {sql}
```

- db：数据库名称；
- user：当前连接中使用的用户名称；
- host：客户端访问地址；
- query_time：SQL 执行耗时，单位为 ms；
- sql：客户端发送的 SQL 语句。

全量审计日志示例

```
[INFO ] 2022-07-01 00:00:00.000 [ShardingSphere-Command-0] GENERAL-QUERY - db: sharding_db user: root host: 127.0.0.1
query_time: 145
CREATE TABLE `t_order` (
  `order_id` bigint(20) NOT NULL AUTO_INCREMENT,
  `user_id` int(11) NOT NULL,
  `status` varchar(50) COLLATE utf8mb4_bin DEFAULT NULL,
  PRIMARY KEY (`order_id`)
)
```

8.16 数据管理

8.16.1 数据导出

SphereEx-DBPlusEngine 暂不支持

8.16.2 数据导入

SphereEx-DBPlusEngine 暂不支持

8.16.3 数据备份

SphereEx-DBPlusEngine 暂不支持，可在数据库端操作。

8.16.4 数据恢复

SphereEx-DBPlusEngine 暂不支持，可在数据库端操作。

8.16.5 数据集成（迁移）

数据迁移

SphereEx-DBPlusEngine 提供异构数据库间的迁移能力。

■ 支持范围

目标端是 proxy，目前支持的数据库类型还比较有限，包括：MySQL、PostgreSQL、openGauss。新的数据库类型正在支持中。

■ 源端支持情况

源端支持情况如下表所示：

	Oracle	MySQL	PostgreSQL	openGauss
全量迁移	Y	Y	Y	Y
增量迁移	N	Y	Y	Y
数据对比	Y	Y	Y	Y

源端支持的数据库类型更多一些。

第一类：除了全量迁移，还支持增量同步，包括：MySQL、PostgreSQL、openGauss。

第二类：只支持全量迁移，包括：Oracle 等商业数据库。目前经过实际验证的是 Oracle，对应目标端包括 PostgreSQL 和 MySQL。增量同步正在支持中。

■ 目标端支持情况

	MySQL	PostgreSQL	openGauss
全量迁移	Y	Y	Y
增量迁移	Y	Y	Y
数据对比	Y	Y	Y

数据一致性校验支持异构数据库，但是支持的算法类型有限制，目前推荐使用 DATA_MATCH 算法。

■ 第三方工具

也可以采用第三方工具的方式实现异构数据的导入导出，如 DataX 等。

运维指南面向的人群为运维人员，主要描述集群生态工具、集群组件、集群的运维管理工作。

- License 管理

9.1 部署环境

支持两种部署环境：1. 非容器化部署：在本地物理主机、虚拟化主机、云主机中部署。2. 容器化部署：K8S 部署中的 helm chart 方式部署。

如下为工具或组件在不同部署环境下的支持情况，“Y”代表支持，“N”代表不支持。

集群与工具	工具与组件	非容器化	容器化
集群生态工具	Boot	Y	N
集群生态工具	Console	Y	N
集群	计算节点	Y	Y
	治理中心	Y (ZooKeeper)	环境自有
	日志中心	Y	环境自有
	监控中心	Y	环境自有
	存储节点	用户自有	用户自有

9.2 非容器化部署与管理

在私有环境中部署 SphereEx-DBPlusSuite。

■ 部署架构图

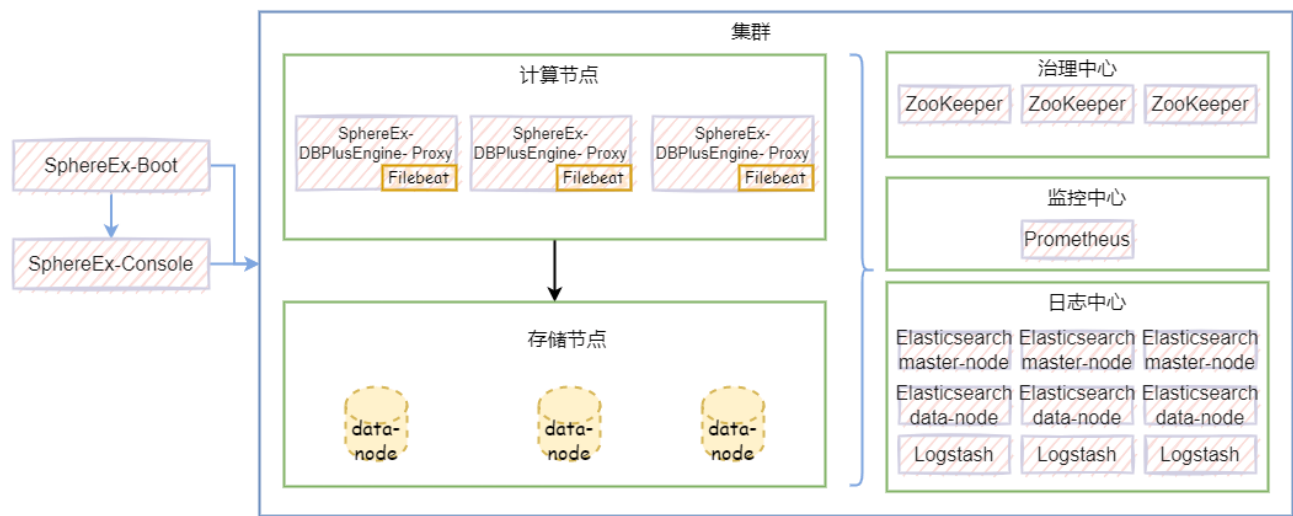


Fig. 1: 部署架构图

■ 最小化部署

主机	组件 / 插件	组件/插件	组件 / 插件	组件/插件	组件 / 插件	组件/插件
主机 1	治理中心 1	监控中心 -ZK-exporter	计算节点 1	日志中心 - filebeat	监控中心	日志中心- master1-data1
主机 2	治理中心 2	监控中心 -ZK-exporter	计算节点 2	日志中心 - filebeat		日志中心- master2-data2
主机 3	治理中心 3	监控中心 -ZK-exporter	计算节点 3	日志中心 - filebeat	日志中心 -logstash	日志中心- master3-data3

■ 推荐部署

主机	组件	组件插件
主机 1	治理中心 1	监控中心-ZK-exporter
主机 2	治理中心 2	监控中心-ZK-exporter
主机 3	治理中心 3	监控中心-ZK-exporter
主机 4	日志中心-master1-data1	
主机 5	日志中心-master2-data2	
主机 6	日志中心-master3-data3	
主机 7	日志中心-logstash	
主机 8	计算节点 1	日志中心-filebeat
主机 9	计算节点 2	日志中心-filebeat
主机 10	计算节点 3	日志中心-filebeat
主机 11	监控中心	

9.2.1 集群生态工具的安装卸载

安装 SphereEx-Boot

1. 新建操作系统用户：

```
# groupadd spex
# useradd -g spex spex
# passwd spex
```

2. 创建推荐的安装目录：/home/spex/boot

3. 依赖：

- sshpass 1.0.0+
- Python 2.7 或者 Python 3.5+
- pip 20.0.0+

4. 安装 SphereEx-Boot

- 执行如下命令，安装 SphereEx-Boot。

```
# pip install sphereex-boot-1.1.0.tar.gz
Processing ./sphereex-boot-1.1.0.tar.gz
Requirement already satisfied: ansible<=2.10.7,>=2.8.0 in /usr/lib/python2.7/site-packages (from sphereex-boot==1.1.0) (2.10.7)
.....
Building wheel for sphereex-boot (setup.py) ... done
Created wheel for spex: filename=sphereex_boot-1.1.0-py2-none-any.whl size=212249
sha256=5de4a3b4f347da57697ad942f74284be38ecf12e5e0d1fefda06e5e4282c22eb
Stored in directory: /root/.cache/pip/wheels/e5/c4/d7/64a96c1f48ffabc68df6aa164beb7a93a1ad4d81aba4b751d7
```

Successfully built sphereex-boot

Installing collected packages: sphereex-boot
Successfully installed sphereex-boot-1.1.0

- 查看 SphereEx-Boot 安装结果

执行命令查看 SphereEx-Boot 版本，如果输出的版本为当前安装的版本，说明安装成功。

```
# spex --version
Version 1.1.0
```

卸载 SphereEx-Boot

执行卸载命令

```
# pip uninstall sphereex-boot
```

安装 SphereEx-Console

部署说明

■ 最小化部署

最小化部署 Console，复用用户环境中已有的 5.7 版本及以上的 MySQL 数据库。

主机	生态工具
主机 1	Console

■ 推荐部署

推荐部署 Console，使用独立主机和 Console 中内置的 MySQL 软件安装 Console 后台 MySQL 数据库。

主机	生态工具	组件
主机 1	Console	Console 后台 MySQL 数据库（主）
主机 2		Console 后台 MySQL 数据库（从）

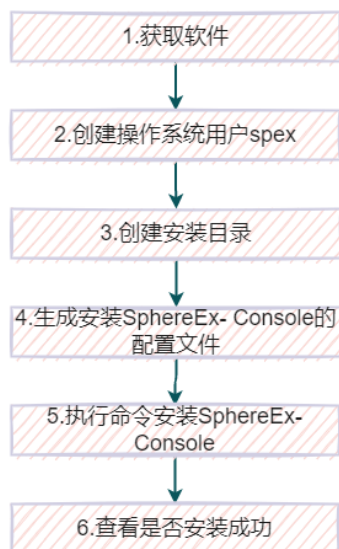


Fig. 2: 安装流程

安装准备

1. 新建操作系统用户

安装 Boot 时已创建此步可省略

```
# groupadd spex
# useradd -g spex spex
# passwd spex
```

2. 创建推荐的安装目录:

```
/home/spex/console
```

3. 依赖:

- JDK 1.8
- 开启 ssh 服务
- SphereEx-Boot

4. 环境检查 (校验依赖)

- 查看 JDK 版本


```
# java -version
```

- 检查是否开启 ssh 服务

```
# ps -aux | grep sshd
```

- 检查是否安装 SphereEx-Boot

```
# spex --version
```

5. 获取软件：请联系我们。
6. 使用 SphereEx-Boot 的 spex 命令导出配置文件 console_install.yaml

```
# spex console export
```

最小化安装 SphereEx-Console 初始化配置文件

最小化安装前提：

- 已有 MySQL 5.7 及以上数据库。
- 且已经创建 spex 数据库用户，并允许 console 所在主机访问。

1. 创建 spex 用户：

```
create database console
create user 'spex'@[SphereEx-Console IP] identified by 'XXXXXX';
grant all on console.* to spex@[SphereEx-Console IP];
```

2. 数据库初始化脚本

SphereEx-Console 解压后的目录中的 db 目录下的：

```
schema.sql
schema-data.sql
```

3. 按实际情况，参考如下方式修改配置文件，未提及参数保持默认即可。

```
vi console_install.yaml
##### SphereEx-Console Deploy Configuration Example #####

# This file is an example configuration file.
# Here you can configure for the SphereEx-Console deploy.
# Deploy to remote must use ssh

#===== Mysql Deploy Configuration =====

# Required, Whether to install a new Mysql server
# If true Mysql master-slave will be installed

install_mysql: false

#===== Console Deploy Configuration =====
console:
# install on local or remote
ssh_user: spex
ssh_password: spex
ssh_port: 22
host: xxx.xxx.xxx.xxx
# Required, SphereEx-Console file. for example:/root/console/SphereEx-Console.tar.gz
file: /home/spex/boot/console.tar.gz
# Required, deployment directory, remote or local
```

```
install_dir: /home/spex/console
# Required, deployment directory, remote or local
console_config:
  server:
    port: 8088
  # software home path
  software:
    home: /home/spex/console
    spring:
      # If install_mysql is true, the param will be overwritten
      # If you use your own Mysql, edit the configuration below
  datasource:
    url: jdbc:mysql://[IP]:[PORT]/console?serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=true # console mysql url
    username: spex # console mysql user
    password: sphereEx@2021 # console mysql password
    driverClassName: com.mysql.jdbc.Driver # console mysql driver
```

推荐安装 SphereEx-Console 初始化配置文件

参考推荐部署来规划主机，按实际情况，将 SphereEx-Console 和 MySQL 主从分别部署在两台主机上
参考如下方式修改配置文件，未提及参数保持默认即可。

```
vi console_install.yaml
##### SphereEx-Console Deploy Configuration Example #####

# This file is an example configuration file.
# Here you can configure for the SphereEx-Console deploy.
# Deploy to remote must use ssh

#===== Mysql Deploy Configuration =====

# Required, Whether to install a new Mysql server
# If true Mysql master-slave will be installed

install_mysql: true

# If install Mysql is true the param is required

mysql:
  master:
    # install on local or remote
    file: /home/spex/console/mysql.tar.xz
    host: xxx.xxx.xxx.xxx
    listen_port: 3306
    ssh_user: spex
    ssh_password: spex
    ssh_port: 22
    install_dir: /home/spex/console

  slave:
    # install on local or remote
    host: xxx.xxx.xxx.xxx
    file: /home/spex/boot/mysql.tar.xz
    # If two Mysql servers are on the same machine, listen port must be different
    listen_port: 3307
    ssh_user: root
    ssh_password: root
    ssh_port: 22
    install_dir: /home/spex/console

#===== Console Deploy Configuration =====
```

```
console:
# install on local or remote
ssh_user: spex
ssh_password: spex
ssh_port: 22
host: xxx.xxx.xxx.xxx
# Required, SphereEx-Console file. for example:/root/console/SphereEx-Console.tar.gz
file: /home/spex/boot/console.tar.gz
# Required, deployment directory, remote or local
install_dir: /home/spex/console
# Required, deployment directory, remote or local
console_config:
server:
port: 8088
# software home path
software:
home: /home/spex/console
```

执行安装 SphereEx-Console

1. 执行命令安装 SphereEx-Console

```
# spex console install -n {console_name} -c console_install.yaml
```

2. 查看 SphereEx-Console 列表

```
# spex console list
```

卸载 SphereEx-Console

```
# spex console uninstall -n {console_name}
```

SphereEx-Boot 的使用

- SphereEx-Boot 的启动与停止

无需启动和停止，安装完成直接使用 spex 命令即可。

SphereEx-Console 使用

- SphereEx-Console 的启动

登录 console 所在主机，进入 Console 的安装目录，执行启动命令。

```
bin/start.sh
```

使用服务器的 IP 地址加端口通过浏览器访问 SphereEx-Console 平台，使用配置文件中账号和密码登录。

例如：192.168.10.18:8080

内置超级管理员的用户名/密码：admin/admin



Fig. 3: Console 首页

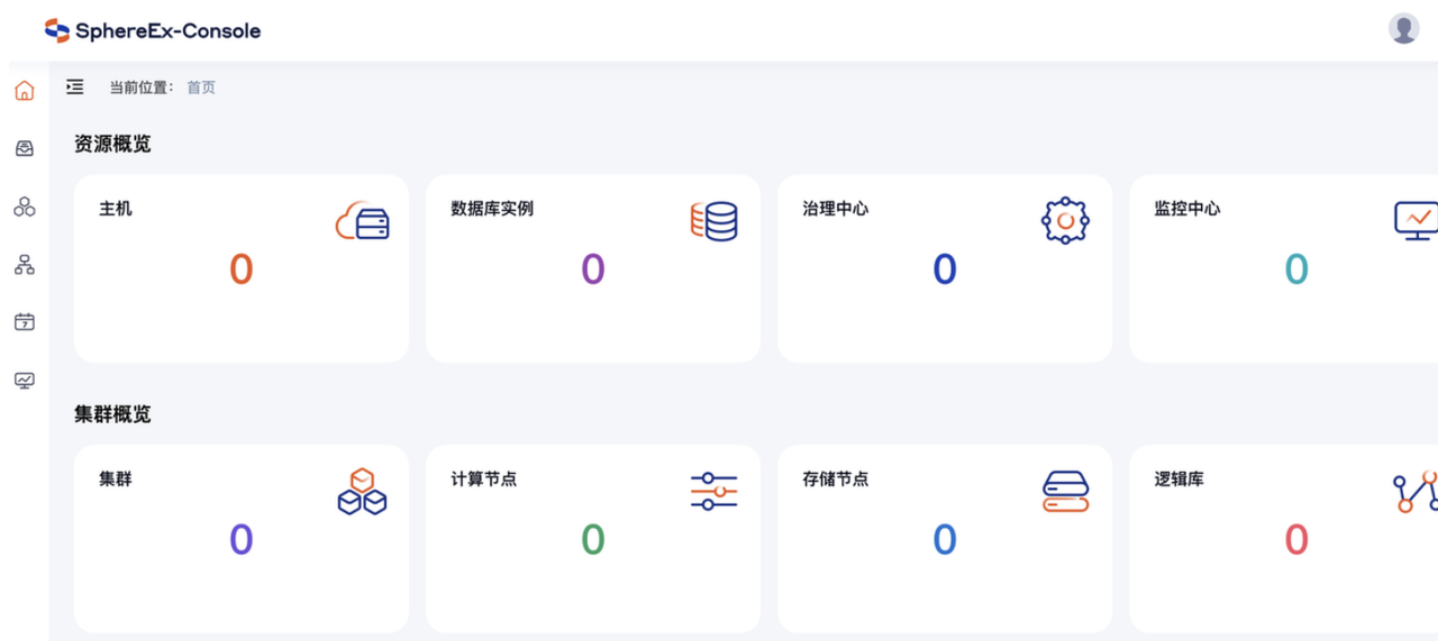


Fig. 4: 概览页

■ SphereEx-Console 的停止

登录 SphereEx-Console 所在主机，进入 SphereEx-Console 的安装目录，执行停止命令。

```
bin/stop.sh
```

■ SphereEx-Console 的备份

登录 SphereEx-Boot 所在主机，指定 SphereEx-Console 的名称进行卸载。

```
# spex console backup -n {name}
```

9.2.2 集群组件的安装卸载

计算节点和存储节点的安装卸载在集群的安装卸载部分做介绍获取软件：组件治理中心（ZooKeeper）、监控中心（Prometheus）、日志中心（Elasticsearch Logstash Filebeat）console 内置，无需额外获取。

■ 集群组件安装方式描述

自建资源或组件：指通过 console 或者 boot 新建的组件资源

注册资源或组件：指通过 console 注册的组件资源

如下列表中描述的是，生态工具可以完成的组件安装注册卸载的功能。其中安装是通过新建功能完成的，注册是注册功能完成的，注销和卸载是通过删除功能完成的。

功能	生态工具	计算节点	存储节点	治理中心	监控中心	日志中心	组件
安装	Boot	Y		Y			计算节点和治理中心的安装，是合并提供，暂不支持独立安装
	Console	Y		Y	Y	Y	实际执行调用 Boot 完成
注册	Boot						
	Console	Y*	Y	Y			此版本不提供对监控、日志中心的注册目前 Proxy 添加集群，刷新到列表
卸载	Boot	Y	Y	Y*			-计算节点：无法独立卸载，计算节点停止后将离开集群-存储节点：卸载不删除数据库，仅删除关联关系-治理中心：支持自建资源卸载 ZK，无法独立卸载，卸载集群时卸载-监控中心：卸载 prometheus，删除监控配置-日志中心：卸载删除 ELK，卸载软件，可独立卸载节点。
	Console	Y	Y	Y	Y	Y	-计算节点：即删除节点功能，卸载软件，卸载监控，卸载日志。-存储节点：卸载不删除数据库，仅删除关联关系。-治理中心：支持自建资源卸载 ZK，注册资源无法卸载。-监控中心：卸载 prometheus，删除监控配置和数据-日志中心：卸载删除 ES、Logstash，卸载软件，可独立卸载节点。
注销	Boot						
	Console		Y	Y			删除不卸载软件的情况下是执行注销操作-存储节点：删除数据库，执行注销操作，不卸载数据库本身-治理中心：删除注册的治理中心，执行注销操作

■ 组件与集群关系说明

如下是集群对组件的独享和共享说明：独享：集群的独享组件，即集群与组件是一对一的关系，一个组件只能被一个集群使用
共享：集群间的共享组件，即集群与组件是一对多的关系，一个组件可以被多个集群使用创建集群本质是创建 Engine-Proxy（计算节点）的集群。创建一个可用的集群，计算节点、治理节点、存储节点为必选的，监控中心日志中心为非必选的。

组件	独享/共享	是否必选	说明
计算节点	独享	是	无法单独安装，安装计算节点即安装集群
治理中心	共享	是	治理中心为共享，治理中心的一个 namespace 为一个独立的集群
存储节点	共享	是	存储节点与集群为多对多的关系，一个 db 为一个存储节点，存储节点只能注册
监控中心	共享	否	监控中心为共享
日志中心	共享	否	日志中心为共享组件，仅管理计算节点的日志。

主机的注册与注销

主机资源高于计算节点、存储节点、治理中心、日志中心、监控中心；需要在完成资源配置后，再完成组件的配置。主机的注册分为：单台注册、批量注册。

SphereEx-Console 注册单台主机

适用场景

单台注册主机，适用于注册少量主机，一台一台操作。

前提条件

主机已经安装操作系统，ssh 服务已经开启，可以独立访问

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 点击 资源-> 主机进入监控中心列表
- 3. 点击右上角 注册按钮，默认进入 单台注册 的页面
- 4. 填写信息，具体参数见下表。

字段	数据来源	选填/必填	数据存储	长度限制	说明
主机 IP	用户输入	必填	console mysql 存储	IP 格式	主机列表校验唯一性
主机用户	用户输入	必填	console mysql 存储	32	
用户口令	用户输入	必填	console mysql 存储		符合密码保护规则
SSH 端口	用户输入	必填	console mysql 存储		校验限制：1 024-5000
主机名称	有默认值，用户输入	选填	console mysql 存储	50	默认值为主机 IP，用户可以修改；主机列表校验唯一性
主机标签	用户选择；可以多选；可以输入	选填	console mysql 存储		用户输入的标签不存在时，后台可执行添加标签的动作
描述	用户输入	选填	console mysql 存储	500	用户自定义

SphereEx-Console 注册多台主机

适用场景

批量注册主机，适用于批量注册多台主机的场景

前提条件

主机已经安装操作系统，ssh 服务已经开启，可以独立访问

操作步骤

- 1. 登录 SphereEx-Console
- 2. 点击 资源-> 主机进入监控中心列表
- 3. 点击右上角 注册按钮，点击批量注册批量注册主机的页面
- 4. 点击 点击上传上传文件，文件格式 xlsx；文件最大支持 5M，文件内容最多包含 100 台主机
- 5. 点击 点击此处下载模版可以下载模版
- 6. 点击右下角 下一步进入校验页面

7. 模版中的参数见下表。

字段	数据来源	选填/必填	长度限制	说明
主机 IP	用户输入	必填	IP 格式	主机列表校验唯一性
主机用户	用户输入	必填	32	
用户口令	用户输入	必填		符合密码保护规则
SSH 端口	用户输入	必填		校验限制：1024-5000
主机名称	有默认值; 用户输入	选填		默认值为主机 IP，用户可以修改; 主机列表校验唯一性
监控中心	用户输入	选填		用户输入，通过【资源】【监控】查找需要的监控中心
监控插件	用户输入	选填		用户输入，通过【资源】【监控】【插件管理】查找需要的监控插件
监控插件端口	用户输入	选填		用户输入，通过【资源】【监控】【插件管理】查找需要的监控插件对应的默认端口
监控插件安装路径	用户输入	选填		用户输入，通过【资源】【监控】【插件管理】查找需要的监控插件对应的默认安装目录
主机标签	用户输入	选填		用户输入的标签不存在时，后台可执行添加标签的动作
描述	用户输入	选填	500	用户自定义

8. 校验结果为通过的点击下一步完成主机的批量注册

SphereEx-Console 注销主机资源

适用场景

不需要在 SphereEx-Console 中使用此主机资源

前提条件

主机未被使用，未配置监控，删除主机仅解除对主机的管理，不会删除物理主机

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 主机进入监控中心列表。
3. 点击列表操作列中的 删除按钮。
4. 点击提示框中的 确认完成主机的注销。

监控中心的安装卸载

SphereEx-Boot 安装监控中心

前提条件

SphereEx-Boot 主机和部署 Prometheus 的主机需要实现 SSH 互信认证。

说明：安装 Prometheus 有如下两种方式。

- 指定参数安装

适用场景

当需要对单台主机快速进行 Prometheus 安装时可以直接输入命令进行安装。操作步骤

1. 在 [Prometheus 官网](#) 下载好安装包 prometheus-xxx.xxx.tar.gz。

说明：也可以使用 Boot 工具包中的 Prometheus 安装包。

2. 直接输入命令进行安装。

```
$ spex component install --user "root" --password "123456" --install-dir /root/prometheus --package ./prometheus-2.33.3.linux-amd64.tar.gz --host 127.0.0.1
```

■ 指定配置文件安装

适用场景

当需要在多台主机部署 Prometheus 时，可以使用配置文件进行安装。

操作步骤

- 在 [Prometheus 官网](#) 下载好安装包 prometheus-xxx.xxx.tar.gz。
- 导出配置模板 prometheus-config.yaml。

```
$ spex component install -t
$ ls
prometheus-config.yaml
```

- prometheus-config.yaml 配置文件样例如下，用户根据实际情况进行配置。

```
install_user: root # 用户名
install_password: root # 用户密码
prometheus:
  file: /root/prometheus-x.x.x.tar.gz # prometheus 安装包路径
servers:
  - host: 10.0.1.1 # 部署主机 IP
    install_dir: /root/prometheus/ # 单独设置安装目录
```

- 配置好 prometheus-config.yaml 配置文件，使用如下命令进行安装。

```
$ spex component install -f ./prometheus-config.yaml
```

SphereEx-Console 安装监控中心

适用场景

需要使用 SphereEx-Console 中的监控功能。

注意事项

安装监控中心的前提条件：主机已经注册，注册使用的主机用户的账号有目录的读写权限。

操作步骤

- 登录 SphereEx-Console。
- 点击 **资源-> 监控中心** 进入监控中心列表。
- 点击右上角 **新建按钮**。
- 填写信息，具体参数见下表。

字段	数据来源	选填/必填	长度限制	说明
监控中心名称	用户输入	必填	50	在监控中心列表中校验唯一性
监控中心类型	下拉选择	必填		目前支持：Prometheus
添加节点				
IP	下拉选择，主机资源列表	必填		同一主机可以装不同的软件，所以已经装过其他软件的的主机也可以显示
端口	用户输入	必填	1024-5000	
安装目录	用户输入	必填		默认为 /opt 用户可以修改，没有时新建，没有权限时报错
描述	用户输入	选填	500	

5. 点击 **确定**，完成监控中心的安装。

SphereEx-Console 卸载监控中心

适用场景

不需要使用 SphereEx-Console 中的监控功能。

注意事项

- 删除监控中心需要保证监控中心未被使用，且被停止。
- 点击 **删除**会提示是否卸载软件，删除时选择卸载软件的监控中心将无法被 console 再次管理。

操作步骤

1. 登录 SphereEx-Console。
2. 在控制台左上角选择 **资源 > 监控中心**。
3. 单击列表中操作列中的 **删除**按钮。
4. 点击提示框中的 **确认**，完成监控中心的卸载

治理中心的注册、注销、安装、卸载

SphereEx-Console 注册治理中心

适用场景

将用户环境中已有的 ZooKeeper 注册到 SphereEx-Console。

注意事项

用户环境中已经 ZooKeeper 已经安装，可以正常提供服务。

操作步骤 1. 登录 SphereEx-Console。2. 在控制台左上角选择 **资源 > 治理中心**。3. 单击右上角 **注册**按钮。4. 填写信息，具体参数见下表。

字段	数据来源	选填/必填	长度限制	说明
治理中心类型	下拉选择	必填		目前支持：ZooKeeper
治理中心名称描述	用户输入	必填	50 500	校验唯一性

5. 点击 **注册主机**完成主机注册，信息填写参考主机的注册部分。
6. 点击提示框中的 **确认**，完成治理中心的注册。

SphereEx-Console 注销治理中心

适用场景

从 SphereEx-Console 中删除注册的 ZooKeeper。

注意事项

ZooKeeper 不会被停止或卸载，直接可以删除，监控配置将被删除，监控插件将被卸载。

操作步骤

1. 登录 SphereEx-Console。
2. 在控制台左上角选择 **资源 > 治理中心**。
3. 单击列表中操作列的 **删除按钮**。
4. 点击提示框中的 **确认**，完成治理中心的注销。
 - SphereEx-Console 安装治理中心

适用场景

通过 SphereEx-Console 安装 ZooKeeper。

注意事项

主机已经被注册，注册使用的主机用户的账号有目录的读写权限。

操作步骤

1. 登录 SphereEx-Console。
2. 在控制台左上角选择 **资源 > 治理中心**。
3. 单击右上角 **新建按钮**。
4. 填写信息，具体参数见下表。

治理中心基本信息：

字段	数据来源	选填/必填	长度限制	说明
治理中心类型	下拉选择	必填		目前支持：Zookeeper
治理中心名称	用户输入	必填	50	校验唯一性
治理中心软件	下拉选择	必填		默认填充 console 内置的 ZK 对应的软件版本
描述	用户输入	选填	500	
监控中心	下拉选择	选填		已经注册的监控中心
监控插件软件	下拉选择	依赖于【监控中心】		默认填充 console 内置的 ZK 对应的监控插件的软件如果监控中心已选，则此项为必填项如果监控中心未选，则此项不能填写

节点信息：

字段	数据来源	选填/必填	长度限制	说明
节点 IP	用户输入	必填		IP：端口集群列表校验唯一性，其子集或者交集都算违反唯一性约束
节点端口	用户输入	必填	0-65536	IP：端口集群列表校验唯一性，其子集或者交集都算违反唯一性约束
监控插件端口	自动填充, 用户可编辑	依赖于【监控中心】		默认值为监控插件的默认端口如果监控中心已选，则此项为必填项如果监控中心未选，则此项不能填写
内部通讯端口	自动填充, 用户可编辑	必填	0-65536	用于集群内部的通讯端口默认值为 2888
内部选举端口	自动填充, 用户可编辑	必填	0-65536	用于集群内部的选举端口默认值为 3888
安装目录	用户输入	必填		期望安装到的主机目录

5. 点击提示框中的确认，完成治理中心的安装。

SphereEx-Console 卸载治理中心

适用场景

从 SphereEx-Console 中删除通过 SphereEx-Console 安装的 ZooKeeper。

注意事项

ZooKeeper 需要先被停止，根据勾选判断是否卸载 ZooKeeper 软件，监控配置将被删除，监控插件将被卸载。

操作步骤

1. 登录 SphereEx-Console。
2. 在控制台左上角选择 资源 > 治理中心。
3. 单击列表中操作列的 删除按钮。
4. 点击提示框中的 确认，完成治理中心的卸载。

日志中心的安装与卸载

SphereEx-Console 安装日志中心

适用场景

需要使用 SphereEx-Console 中的日志管理功能。

注意事项

主机已经注册，注册使用的主机用户的账号有目录的读写权限。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 日志中心进入监控中心列表。
3. 点击右上角 新建按钮。
4. 填写信息，具体参数见下表。基本信息：

字段	数据来源	选填/必填	长度限制	说明
日志中心名称描述	用户输入	用户输入	50 500	校验唯一性

Elasticsearch（必填）信息：

字段	数据来源	选填/必填	长度限制	说明
软件	下拉选择	必填		默认填充 console 内置的 Elasticsearch 版本
节点 IP	下拉选择	主机列表	必选	节点 IP、角色、节点端口校验唯一性
角色	用户选择			必须安装一个 master-node 和 data-node 增加第三个节点是可以选择角色
内部通讯端口	用户输入	必填	校验限制：1024-5000	节点 IP、角色、节点端口校验唯一性
节点端口	用户输入	必填	校验限制：1024-5000	节点 IP、角色、节点端口校验唯一性
安装目录	用户输入	必填		当角色中无 data-node 时，数据目录不需要输入
数据目录	用户输入	必填		当角色中无 data-node 时，数据目录不需要输入
描述	用户输入	选填		

Logstash（必填）信息：

字段	数据来源	选填/必填	长度限制	说明
软件	下拉选择	必填		默认填充 console 内置的 logstash 版本
节点 IP	下拉选择主机列表	必选		节点 IP+ 端口校验唯一性
节点端口	用户输入	必填	校验限制：1024-5000	
安装目录	用户输入	必填		
描述	用户输入	选填		

5. 点击 **确定**，完成日志中心的安装。

SphereEx-Console 卸载日志中心

适用场景

不需要使用 SphereEx-Console 中的此日志中心了。

注意事项

- 删除日志中心需要保证日志中心未被使用，且被停止。
- 点击删除会提示是否卸载软件，删除时选择卸载软件的日志中心将无法被 console 再次管理。

操作步骤

1. 登录 SphereEx-Console。
2. 在控制台左上角选择 **资源 > 日志中心**。
3. 单击列表中操作列中的 **删除**按钮。
4. 点击提示框中的 **确认**，完成日志中心的卸载。

数据库资源的注册与注销

SphereEx-Console 注册数据库资源

适用场景

将用户环境中已有的数据库注册到 SphereEx-Console 中管理。

前提条件

数据库已经安装，可以正常提供服务；允许 SphereEx-Console 访问的数据库账号已经被创建。数据库的权限，需要根据业务用户自行规划。

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 点击 资源-> 数据库进入数据库列表。
- 3. 点击右上角 注册按钮。
- 4. 填写信息，具体参数见下表。

字段	数据来源	选填/必填	长度限制	说明
数据库地址	用户输入	必填		IP，数据库列表“实例地址”+“实例端口”唯一性校验
数据库端口	用户输入	必填	校验限制：1024-5000	
数据库类型	下拉选择	必填		目前支持：MySQL
数据库用户	用户输入	必填	32	连接数据库实例用户
用户口令	用户输入	必填		密码保护规则
描述	用户输入	选填	500	

- 5. 点击 注册主机完成主机注册，信息填写参考主机的注册部分。
- 6. 点击提示框中的 确认，完成数据库的注册。

SphereEx-Console 注销数据库资源

适用场景

不需要在 SphereEx-Console 中使用此数据库资源。

前提条件

从 SphereEx-Console 中删除主机仅解除对数据库资源的管理，不会删除物理数据库。

操作步骤 1. 登录 SphereEx-Console。2. 点击 资源-> 数据库进入监控中心列表。3. 点击列表操作列中的 删除按钮。4. 点击提示框中的 确认完成数据库的注销。

集群组件的管理

计算节点和存储节点的管理在该部分做介绍。

主机资源管理

主机为集群的基础资源，只能使用 Console 管理。
功能列表如下：

功能	方式	说明
探活	Console	探活
刷新	Console	刷新数据到数据库读取到页面
修改	Console	修改主机名密码
配置监控	Console	配置主机监控

探活

适用场景

探活主机

前提条件

主机已经被注册。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 主机进入主机列表。
3. 点击列表操作列中的 探活按钮完成主机探活。

指定主机刷新

适用场景

指定主机刷新

前提条件

主机已经被注册

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 主机进入主机列表。
3. 点击列表操作列中的 刷新按钮完成刷新。

所有主机刷新

适用场景

列表中所有主机刷新

前提条件

主机已经被注册

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 主机进入主机列表。
3. 点击右上角 刷新按钮完成刷新。

修改

适用场景

指定主机信息的修改

前提条件

主机已经被注册，未被使用时所有信息可以修改，已经被使用后主机 IP 不可以被修改。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 主机进入主机列表。
3. 点击列表操作列中的 修改按钮进入主机修改页面。
4. 填写如下信息完成主机信息的修改：

字段	数据来源	选填/必填	长度限制	说明
主机 IP	主机列表	必填	IP 格式	校验唯一性，未关联组件，未配置监控时可以修改
主机用户	主机列表	必填	32	可以修改
用户口令	主机列表	必填		符合密码保护规则，可以修改
SSH 端口	主机列表	必填		校验限制：1024-5000，可以修改
主机名称	主机列表	选填		默认值为主机 IP，用户可以修改主机列表校验唯一性
主机标签	主机列表	选填		用户输入的标签不存在时，后台可执行添加标签的动作，可以修改
主机标签	主机列表	选填		用户输入的标签不存在时，后台可执行添加标签的动作，可以修改

5. 点击提示框中的 确认完成主机的修改。

配置监控

请参考[主机监控告警](#)

监控中心管理

功能	方式	说明
查看监控插件	Console	查看内置的监控插件
探活	Console	ping 命令
刷新	Console	刷新数据到数据库读取到页面
修改	Console	修改主机名密码
启动	Console	启动监控中心资源
停止	Console	停止监控中心资源

查看监控插件

适用场景

查看内置的监控插件信息。

前提条件

无

操作步骤

1. 登录 SphereEx-Console
2. 点击 资源-> 监控中心进入列表

3. 点击右上角 **监控插件**按钮进入监控插件列表页面
4. 点击右上角 **返回**按钮返回监控中心列表页面

探活

适用场景

探活监控中心

前提条件

监控中心已经被安装

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **资源-> 监控中心**进入列表。
3. 点击列表操作列中的 **探活**按钮完成探活。

指定监控中心刷新

适用场景

指定监控中心刷新。

前提条件

监控中心已经被安装。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **资源-> 监控中心**进入监控中心列表。
3. 点击列表操作列中的 **刷新**按钮完成刷新。

修改

适用场景

指定监控中心信息的修改，可以修改监控中心名称和描述。

前提条件

监控中心已经被安装。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **资源-> 监控中心**进入列表。
3. 点击列表操作列中的 **修改**按钮进入修改页面。
4. 填写如下信息：

字段	数据来源	选填/必填	长度限制	说明
监控中心名称	用户输入	必填		可以修改
描述	用户输入	选填	500	可以修改

5. 点击 **确认**按钮完成完成监控中心信息的修改。

启动

适用场景

启动指定监控中心

前提条件

监控中心已经被安装，监控中心不可重复启动，启动已经启动的监控中心会提示已启动。

操作步骤 1. 登录 SphereEx-Console。2. 点击 资源-> 监控中心进入监控中心列表。3. 点击列表操作列中的 启动按钮完成启动。

停止

适用场景

停止指定监控中心

前提条件

监控中心已经被安装，监控中心不可重复停止，启动已经停止的监控中心会提示已停止，监控中心停止后，监控功能将无法使用。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 监控中心进入监控中心列表。
3. 点击列表操作列中的 停止按钮，完成停止。

治理中心管理

功能	方式	说明
探活	Console	Ping 命令
刷新	Console	刷新数据到数据库读取到页面
修改	Console	修改主机名密码
日志	Console	查看在 Console 平台中操作治理中心的日志
管理节点	Console	探活、刷新
配置监控	Console、Boot	注册主机、安装、卸载、停止、启动
启动	Console、Boot	启动治理中心资源
停止	Console、Boot	停止治理中心资源

探活

适用场景

探活治理中心

前提条件

治理中心已经被安装或注册

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 治理中心进入列表。
3. 点击列表操作列中的 探活按钮完成探活。

指定治理中心刷新

适用场景

指定治理中心刷新

前提条件

治理中心已经被注册或安装

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 点击 资源-> 治理中心进入主机列表。
- 3. 点击列表操作列中的 刷新按钮，完成刷新。

所有治理中心刷新

适用场景

列表中所有治理中心刷新

前提条件

治理中心已经被注册或安装。

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 点击 资源-> 治理中心进入主机列表。
- 3. 点击右上角 刷新按钮，完成刷新。

修改

适用场景

指定治理中心信息的修改，可以修改监控中心名称和描述。

前提条件

治理中心已经被安装或注册

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 点击 资源-> 治理中心进入列表。
- 3. 点击列表操作列中的 修改按钮，进入修改页面。
- 4. 填写如下信息：

字段	数据来源	选填/必填	长度限制	说明
治理中心类型	下拉选择	必填		不能修改
治理中心名称	用户输入	必填	50	校验唯一性可以修改
描述	用户输入	选填	500	可以修改

- 5. 点击确认按钮完成完成治理中心信息的修改

查看操作日志

适用场景

查看在 SphereEx-Console 平台内操作治理中心的日志。

前提条件

无

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 治理中心进入列表。
3. 点击操作列 日志按钮进入查看操作日志页面。可以从事件和日志两个维度查看操作的日志。
4. 点击右上角 刷新按钮刷新页面内容。
5. 点击右上角 返回按钮返回列表页面。

管理节点

节点探活

适用场景

探活治理中心节点

前提条件

治理中心中心已经被安装或注册

操作步骤

- a. 登录 SphereEx-Console。
- b. 点击 资源-> 治理中心进入列表。
- c. 点击列表操作列中的 管理节点按钮进入节点列表页面。
- d. 点击节点列表中操作列 探活按钮完成节点探活。
- e. 点击右上角 返回按钮返回治理中心列表。

节点刷新

适用场景

刷新治理中心节点

前提条件

治理中心已经被安装或注册。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 治理中心进入列表。
3. 点击列表操作列中的 管理节点按钮进入节点列表页面。
4. 点击节点列表中操作列 刷新按钮完成节点刷新。
5. 点击右上角 返回按钮返回治理中心列表。

配置监控

请参考治理中心监控

启动

适用场景

启动指定治理中心

前提条件

- 治理中心已经被安装，不可重复启动，启动已经启动的治理中心会提示已启动。
- 当治理中心为注册的组件时，无法通过 SphereEx-Console 启动。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 治理中心进入监控中心列表。
3. 点击列表操作列中的 启动按钮，完成启动。

停止

适用场景

停止指定治理中心

前提条件

- 治理中心已经被安装，不可重复停止，停止已经停止的治理中心会提示已停止。
- 当治理中心为注册的组件时，无法通过 SphereEx-Console 停止。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 治理中心进入监控中心列表。
3. 点击列表操作列中的 停止按钮完成停止。

日志中心管理

功能	方式	说明
查看 Filebeat	Console	查看内置的 filebeat 版本
管理节点	Console	添加节点、探活、刷新、启动、停止、删除
日志	Console	查看在 Console 中操作日志中心的日志
启动	Console	启动日志中心
停止	Console	停止日志中心

查看 Filebeat

适用场景

查看内置的 Filebeat 插件信息

前提条件

无

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 日志中心进入列表。
3. 点击右上角 **Filebeat** 按钮进入 Filebeat 插件列表页面。
4. 点击右上角 返回按钮返回日志中心列表页面。

管理节点

添加节点

适用场景

为一个日志中心添加节点

前提条件

主机已经注册，注册使用的主机用户的账号有目录的读写权限。

操作步骤

- a. 登录 SphereEx-Console。
- b. 点击 资源-> 日志中心进入列表。
- c. 点击操作列 **管理节点**按钮进入节点列表页面。
- d. 点击右上角 **添加节点**按钮进入添加节点页面。
- e. 填写信息，具体参数见下表。

Elasticsearch

字段	数据来源	选填/必填	长度限制	说明
软件	下拉选择	必填		默认填充 console 内置的 Elasticsearch 版本
节点 IP	下拉选择主机列表	必选		节点 IP+ 端口校验唯一性
角色	用户选择			角色不能为空
节点端口	用户输入	必填	校验限制：1024-5000	
安装目录	用户输入	必填		
软件	console 目录中的软件列表	必填		从 console 的软件目录中选择，对应的软件包，文档中说明软件位置
描述	用户输入	选填		

Logstash

字段	数据来源	选填/必填	长度限制	说明
软件	下拉选择	必填		默认填充 console 内置的 logstash 版本
节点 IP	下拉选择主机列表	必选		节点 IP+ 端口校验唯一性
节点端口	用户输入	必填	校验限制：1024-5000	
安装目录	用户输入	必填		
软件	console 目录中的软件列表	必填		从 console 的软件目录中选择，对应的软件包，文档中说明软件位置
描述	用户输入	选填	500	

- f. 点击 **确定** 日志中心添加节点的功能。
- g. 点击右上角 **返回** 按钮返回日志中心列表页面。

节点探活

适用场景

探活日志中心节点

前提条件

日志中心已经被安装

操作步骤

- a. 登录 SphereEx-Console。
- b. 点击 **资源-> 日志中心** 进入列表。
- c. 点击列表操作列中的 **管理节点** 按钮进入节点列表页面。
- d. 点击节点列表中操作列 **探活** 按钮完成节点探活。

节点刷新

适用场景

刷新日志中心节点

前提条件

日志中心已经被安装

操作步骤

- a. 登录 SphereEx-Console。
- b. 点击 **资源-> 日志中心** 进入列表。
- c. 点击列表操作列中的 **管理节点** 按钮进入节点列表页面。
- d. 点击节点列表中操作列 **刷新** 或页面右上角 **刷新** 按钮完成制定节点和全部节点的刷新。

查看节点日志

适用场景

查看在 SphereEx-Console 平台内操作日志中心节点的日志。

前提条件

无

操作步骤

- 登录 SphereEx-Console。
- 点击 **资源-> 日志中心** 进入列表。
- 点击列表操作列中的 **管理节点** 按钮进入节点列表页面。
- 点击操作列 **日志** 按钮进入查看操作日志页面。

可以从事件和日志两个维度查看操作的日志。

- 点击右上角 **刷新** 按钮刷新页面内容。
- 点击右上角 **返回** 按钮返回节点列表页面。

启动节点

适用场景

启动日志中心节点

前提条件

已经启动的节点，启动为灰色不可点

操作步骤

- 登录 SphereEx-Console。
- 点击 **资源-> 日志中心** 进入列表。
- 点击列表操作列中的 **管理节点** 按钮进入节点列表页面。
- 点击列表操作列中的 **启动** 按钮，完成启动。

停止节点

适用场景

停止日志中心节点

注意事项

- 停止节点可能导致日志中心不可用，请谨慎操作。已经停止的节点，停止为灰色不可点。
- 当停止最后一个节点后，日志中心将停止。

操作步骤

- 登录 SphereEx-Console。
- 点击 **资源-> 日志中心** 进入列表。
- 点击列表操作列中的 **管理节点** 按钮进入节点列表页面。
- 点击列表操作列中的 **停止** 按钮，完成停止。

卸载日志中心节点

适用场景

扩容日志中心

注意事项

- 只有停止的节点才可以删除；正在被使用的节点无法删除。
- 点击删除会提示是否卸载软件，删除时不勾选卸载软件的日志中心节点将无法被 console 再次管理，当只剩下最后一个节点时，将无法卸载，只能通过操作日志中心完成卸载。

操作步骤

- a. 登录 SphereEx-Console。
- b. 在控制台左上角选择 **资源 > 日志中心**。
- c. 单击列表中操作列中的 **删除** 按钮。
- d. 点击提示框中的 **确认**，完成日志中心节点的卸载。

查看日志中心日志

适用场景

查看在 SphereEx-Console 平台内操作日志中心的日志。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **资源-> 日志中心** 进入列表。
3. 点击操作列 **日志** 按钮进入查看操作日志页面。可以从事件和日志两个维度查看操作的日志。
4. 点击右上角 **刷新** 按钮刷新页面内容。
5. 点击右上角 **返回** 按钮返回列表页面。

启动

适用场景

启动日志中心

前提条件

如果节点已经启动，启动为灰色不可点。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **资源-> 日志中心** 进入列表。
3. 点击列表操作列中的 **启动** 按钮，完成启动。

停止

适用场景

停止日志中心

注意事项

停止日志中心，日志中心将不可用，请谨慎操作。已经停止的日志中心，停止为灰色不可点。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 日志中心进入列表。
3. 点击列表操作列中的 管理节点按钮进入节点列表页面。
4. 点击列表操作列中的 停止按钮，完成停止。

数据库资源管理功能说明

Console 目前仅支持 MySQL 数据库的管理。

功能	说明
探活	Ping 命令
刷新	刷新数据到数据库读取到页面
修改	修改主机名密码
配置监控	配置数据库监控，需要先配置主机

探活

适用场景

探活数据库实例

前提条件

数据库实例已经被注册。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 数据库进入主机列表。
3. 点击列表操作列中的 探活按钮，完成探活。

指定数据库实例刷新

适用场景

指定数据库实例刷新

前提条件

数据库实例已经被注册。

操作步骤 1. 登录 SphereEx-Console。2. 点击 资源-> 数据库进入主机列表。3. 点击列表操作列中的 刷新按钮，完成刷新。

所有数据库实例刷新

适用场景

列表中所有数据库实例刷新

前提条件

数据库实例已经被注册。

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 点击 资源-> 数据库进入主机列表。
- 3. 点击右上角 刷新按钮，完成刷新。

修改

适用场景

指定数据库实例信息的修改

前提条件

数据库实例已经被注册，修改数据库用户名和口令，不影响正在使用的存储节点，修改后注册的存储节点将使用新用户名和口令。

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 点击 资源-> 数据库进入主机列表。
- 3. 点击列表操作列中的 修改按钮进入主机修改页面。
- 4. 填写如下信息完成信息的修改：

字段	数据来源	选填/必填	长度限制	说明
数据库用户	用户输入	必填		可以修改
用户口令描述	用户输入用户输入	必填选填	500	可以修改可以修改

- 5. 点击提示框中的 确认完成修改。

配置监控

请参考数据库监控

9.2.3 集群安装卸载

前提条件

创建集群前，需先完成相关资源（主机、数据库）和组件（治理中心、监控中心、日志中心）的创建或注册。

驱动说明

SphereEx-Boot 安装集群时，用户需要自行下载数据库驱动，放入 proxy 的目录中。

SphereEx-Console 安装集群时，自带 MySQL 8.0 驱动。

SphereEx-Boot 安装集群

说明：安装卸载集群，实际上是安装卸载计算节点，使用 SphereEx-Boot 创建集群会同步创建治理中心。
链接到快速入门部分的【使用 SphereEx-Boot 创建集群】

SphereEx-Boot 卸载集群

使用 SphereEx-Boot 卸载集群会同步卸载治理中心。
使用命令 `spex cluster uninstall --name <cluster-name>` 卸载集群，会删除掉部署目录。

SphereEx-Console 安装集群

适用场景
实际上是安装计算节点和注册存储节点，治理中心已被创建或注册是创建集群的必要条件。
注意事项

- 主机已经注册，注册使用的主机用户的账号有目录的读写权限。
- 数据库实例已经被注册。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理-> 集群**进入集群列表。
3. 点击右上角 **新建**按钮。
4. 填写信息，具体参数见下表。集群信息

字段	数据来源	选填/必填	长度限制	说明
集群名称	用户输入	必填	50	校验唯一性
描述	用户输入	选填	500	
治理中心地址	治理中心名称列表	必填		
命名空间	用户输入	必填	50	字母数字下划线，不能添加一个已经存在的命名空间
监控中心	监控中心 IP: 端口	选填		
日志中心	日志中心名称	选填		

计算节点

字段	数据来源	选填/必填	长度限制	说明
计算节点软件	用户选择	必填		默认填充 console 支持的最新版本的 engine
filebeat 软件	用户选择	依赖于【日志中心】		默认填充 console 内置的 filebeat 日志中心已填，此项为必填日志中心未填，此项不能填
节点名称	用户输入	必填		默认为 IP: 端口，集群内唯一性校验
节点 IP	主机列表	必填		校验页面中计算节点的 IP 和端口的唯一性
节点端口	用户输入	必填	1024-5000	
监控插件端口	有默认值，用户可自定义	必填		默认填充 9090，可以指定
安装目录	用户输入	必填		默认为/opt，主机上无此目录时创建
logstash	下拉选择日志中心的 logstash	依赖于【日志中心】		日志中心已填，此项为必填日志中心未填，此项不能填
操作结果	返回安装、卸载结果或报错			

存储节点（可选）

字段	数据来源	选填/必填	长度限制	说明
节点名称	用户输入	必填	50	大小写字母，数字，下划线中的一种或组合
数据库实例	下拉选择	必填		
端口	自动填充	必填		依据数据库实例自动填充
用户名	自动填充，可以修改	必填	32	依据数据库实例自动填充
口令	自动填充，可以修改	必填		依据数据库实例自动填充
数据库	下拉选择	必填		
扩展参数（& 分割）	用户输入	选填	不限制	

5. 点击 **确定**，完成集群的安装。

SphereEx-Console 注册集群

适用场景

将已经安装好的集群，注册到 SphereEx-Console 中。

注意事项

集群已经被安装，且能正常提供服务。

操作步骤 1. 登录 SphereEx-Console。2. 点击 **集群管理** -> **集群** 进入集群列表。3. 点击右上角 **注册** 按钮，进入注册页面。4. 填写信息，具体参数见下表。基本信息

字段	数据来源	选填/必填	长度限制	说明
集群名称	用户输入	必填	50	唯一性校验 console 中集群的名称
描述	用户输入	选填	500	用户在 console 中自定义的集群描述信息

计算节点信息

字段	数据来源	选填/必填	长度限制	说明
计算节点地址	用户输入	必填		任意一个 proxy 的地址，目前只支持 IPv4 的地址
计算节点端口	用户输入	必填		对应 proxy 的端口
计算节点用户名	用户输入	必填	32	对应 proxy 的用户名
用户口令	用户输入	必填		对应 proxy 的密码

5. 点击注册主机完成主机注册，信息填写参考主机的注册部分。

6. 点击 **确定**，完成集群的注册。

SphereEx-Console 卸载集群

适用场景

从 SphereEx-Console 中删除集群、卸载计算节点软件。

注意事项

集群已停止，勾选同步卸载软件，同步卸载计算节点的软件，并且卸载监控和日志配置，删除 namespace 信息，SphereEx-Console 解除对集群的管理。

操作步骤

1. 登录 SphereEx-Console。
2. 在控制台左上角选择 **集群管理** > **集群** 进入集群列表。
3. 单击列表中操作列的 **删除** 按钮。
4. 点击提示框中的 **确认**，勾选同步卸载软件完成集群的卸载。

SphereEx-Console 注销集群

适用场景

从 SphereEx-Console 解除对集群的管理。

注意事项

无需停止集群，仅解除对集群的管理。

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 在控制台左上角选择 **集群管理 > 集群**。
- 3. 单击列表中操作列的 **删除按钮**。
- 4. 点击提示框中的 **确认**，**不勾选同步卸载软件完成集群的注销**。

集群运维管理

集群管理功能说明

功能	说明
刷新	刷新数据到数据库读取到页面
修改	修改集群信息
参数管理	管理集群参数
License 管理	管理集群 License
配置日志	配置集群监控，需要先配置主机
配置监控	配置集群监控，需要先配置主机
启动	启动集群
停止	停止集群

指定集群刷新

适用场景

指定集群刷新

前提条件

集群已经被注册或安装。

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 在控制台左上角选择 **集群管理 > 集群**进入集群列表。
- 3. 点击列表操作列中的 **刷新按钮**完成集群刷新。

所有集群刷新

适用场景

刷新列表中所有集群

前提条件

集群已经被注册或安装。

操作步骤

1. 登录 SphereEx-Console。
2. 在控制台左上角选择 **集群管理** > **集群** 进入集群列表。
3. 点击右上角 **刷新** 按钮完成刷新。

参数管理

适用场景

管理集群参数，在值域范围内修改参数，将参数重置成默认值。

注意事项

集群可以访问

操作步骤

1. 登录 SphereEx-Console。
2. 在控制台左上角选择 **集群管理** > **集群** 进入集群列表。
3. 点击列表操作列中的参数管理按钮进入参数管理页面。
4. 点击重置按钮弹出参数重置页面，左下角提示修改后是否立即生效，点击确认将参数重置为默认值。
5. 点击修改按钮弹出参数修改页面，左下角提示修改后是否立即生效，点击确认完成参数修改。

License

适用场景

管理集群 License，更新 license

注意事项

集群可以访问

操作步骤

1. 登录 SphereEx-Console。
2. 在控制台左上角选择 **集群管理** > **集群** 进入集群列表。
3. 点击列表操作列中的 **License** 按钮进入管理页面。

有 License 时，可以看到 License 基本信息。

无 License 时，可以看到无 License 提示。

4. 点击左上角返回按钮返回集群列表。
5. 点击右上角更新 License。添加 License key，点击预览可以预览 License 信息。
6. 点击确认完成 License 的更新

修改

适用场景

可以修改集群的描述和 SphereEx-Console 登录计算节点的信息。

前提条件

集群已经被注册或安装。

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 在控制台左上角选择 **集群管理** > **集群**进入集群列表。
- 3. 点击列表操作列中的 **修改**按钮，进入修改页面。
- 4. 填写如下信息：

字段	数据来源	选填/必填	长度限制	说明
集群名称	用户输入	必填	50	console 中集群的名称，不能修改
描述	用户输入	选填	500	用户在 console 中自定义的集群描述信息，可以修改
计算节点地址	用户输入	必填		任意一个 proxy 的地址，可以修改
计算节点端口	用户输入	必填		对应 proxy 的端口，可以修改
计算节点用户名	用户输入	必填	32	对应 proxy 的用户名，可以修改
用户口令	用户输入	必填		对应 proxy 的密码，可以修改

- 5. 点击提示框中的 **确认**完成修改。

启动

适用场景

启动指定集群

注意事项

不可重复启动，已经启动会提示已启动

集群启动顺序：

- a. 治理中心：为集群启动的必要条件，监测是否启动；如未启动，提示用户手动启动，集群将启动失败。
- b. 日志中心：为集群启动的可选条件，监测是否启动；如有异常，提示错误，未启动提示用户手动启动，不影响集群的启动。
- c. 监控中心：为集群启动的可选条件，监测是否启动；如有异常，提示错误，未启动提示用户手动启动，不影响集群的启动。
- d. 计算节点：为集群启动的必选条件，监测是否启动；如有异常，提示错误，未启动提示用户手动启动；部分节点可用，即可启动，至少有一个。
- e. 存储节点：如果存在存储节点则为必选项，如有异常，集群将启动失败，如果不存在存储节点，不影响集群启动。

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 在控制台左上角选择 **集群管理** > **集群**进入集群列表。
- 3. 点击列表操作列中的 **启动**按钮完成启动。

停止

适用场景

停止指定集群

注意事项

- 停止集群实际上只停止计算节点，集群停止后将无法使用请谨慎操作。
- 集群被停止后，操作列按钮可点，但是无数据。

操作步骤

1. 登录 SphereEx-Console。
2. 在控制台左上角选择 **集群管理 > 集群** 进入集群列表。
3. 点击列表操作列中的 **停止** 按钮完成停止。

配置日志

适用场景

将集群计算节点的日志在 SphereEx-Console 平台集中管理。

配置集群的日志，即配置计算节点的日志，将运行日志和慢日志采集汇总到日志中心。可以通过 Console 的 SQL 管理中慢日志和日志功能筛选、查看、导出日志。其中可以在慢日志中直接查看 sql 的执行计划。

注意事项

- 集群已经被安装或注册
- 集群计算节点所在的主机已经被注册，注册使用的主机用户的账号有目录的读写权限
- 日志中心已经安装

操作步骤

1. 登录 SphereEx-Console。
2. 在控制台左上角选择 **集群管理 > 集群** 进入集群列表。
3. 点击列表操作列中的 **配置日志** 按钮进入配置日志页面。
4. 填写如下信息：

字段	数据来源	选填/必填	长度限制	说明
计算节点 IP	前一页面输入	必填		不可编辑
Logstash	日志中心的 Logstash	必选		filebeat 与 proxy 是一对一关系 file beat 与 lo gsta sh 是多对一关系一个 lo gsta sh 只能归属于一个集群
Filebeat 安装目录	用户输入	必填		默认填充 File beat 默认安装目录

5. 点击 **注册主机** 完成主机注册，信息填写参考主机的注册部分。
6. 点击提示框中的 **确认** 完成配置。

配置监控

请参考[集群监控](#)

计算节点管理功能说明

功能	说明
添加节点	扩容集群
探活	探活节点
刷新	刷新节点信息
启用	启用计算节点
禁用	禁用计算节点，禁用后的计算节点，将无法接收流量
启动	启动计算节点
停止	停止计算节点
删除	卸载计算节点，缩容集群

添加节点

适用场景

为集群扩容计算资源

注意事项

主机已经注册，注册使用的主机用户的账号有目录的读写权限。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理-> 集群**进入集群列表。
3. 点击操作列 **管理节点**，默认进入计算点列表。
4. 点击右上角 **添加节点**，进入添加节点页面。
5. 填写信息，具体参数见下表：

字段	数据来源	选填/必填	长度限制	说明
节点 IP	用户选择	必填		已经注册的主机列表
端口	用户输入	必填	1024-5000	
安装目录	用户输入	必填		默认为/opt，主机上无此目录时创建
软件	用户选择	必填		默认填充 console 支持的最新版本的 engine
节点名称	用户输入	必填		默认为 IP: 端口，集群内唯一性校验
filebeat 软件	用户选择	依赖于【日志中心】		默认填充 console 内置的 filebeat 日志中心已填，此项为必填日志中心未填，此项不能填
监控插件端口	有默认值，用户可自定义	必填		默认填充 9090，可以指定
安装目录	用户输入	必填		默认为/opt，主机上无此目录时创建
Logstash	下拉选择日志中心的 logstash	依赖于【日志中心】		日志中心已填，此项为必填日志中心未填，此项不能填

6. 点击 **确定**，完成计算节点的添加。

探活

适用场景

探活计算节点

注意事项

集群已经被注册或安装

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **集群**进入集群列表。
3. 点击操作列 **管理节点**，默认进入计算点列表。
4. 点击列表操作列中的 **探活**按钮完成节点探活。

刷新

适用场景

刷新计算节点

注意事项

集群已经被注册或安装。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **集群**进入集群列表。
3. 点击操作列 **管理节点**，默认进入计算点列表。
4. 点击列表操作列中的 **刷新**按钮完成节点刷新。

启用

适用场景

启用已禁用的节点

注意事项

节点处于禁用状态

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **集群**进入集群列表。
3. 点击操作列 **管理节点**，默认进入计算点列表。
4. 点击列表操作列中的 **启用**按钮完成节点启用。

禁用

适用场景

禁用节点，禁用后的节点将不会有新的流量，正在执行的进程不会受影响。

注意事项

节点处于启用状态。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **集群**进入集群列表。
3. 点击操作列 **管理节点**，默认进入计算点列表。
4. 点击列表操作列中的 **禁用**按钮完成节点禁用。

使用 SphereEx-Console 启动计算节点

适用场景

启动处于停止状态的节点

注意事项

- 节点处于停止状态
- 集群处于停止状态时，可以点击管理节点逐个启动计算节点，只要有一个计算节点是可用时集群即可用。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **集群**进入集群列表。
3. 点击操作列 **管理节点**，默认进入计算点列表。
4. 点击列表操作列中的 **启动**按钮完成节点启动。

单独启动计算节点

1. 启动步骤

- a. 下载 DBPlusEngine-Proxy 的最新发行版。
- b. 解压缩后修改 conf/server.yaml 和以 config-前缀开头的文件，如：conf/config-xxx.yaml 文件，进行分片规则、读写分离规则配置。
- c. Linux 操作系统请运行 bin/start.sh，Windows 操作系统请运行 bin/start.bat 启动 ShardingSphere-Proxy。

2. 选择数据库协议

使用 PostgreSQL

- a. 使用任何 PostgreSQL 的客户端连接。如：psql -U root -h 127.0.0.1 -p 3307

使用 MySQL

- b. 将 MySQL 的 JDBC 驱动程序复制至目录 ext-lib/。
- c. 使用任何 MySQL 的客户端连接。如：mysql -u root -h 127.0.0.1 -P 3307

使用 openGauss

- a. 将以 org.opengauss 包名为前缀的 openGauss 的 JDBC 驱动程序复制至目录 ext-lib/。
- b. 使用任何 openGauss 的客户端连接。如：gsql -U root -h 127.0.0.1 -p 3307

3. 选择元数据持久化仓库

使用 ZooKeeper

默认集成 ZooKeeper Curator 客户端。

使用 Etcd

将 Etcd 的客户端驱动程序复制至目录 ext-lib/。

4. 使用分布式事务

具体可参考分布式事务。

5. 使用自定义算法

当用户需要使用自定义的算法类时，可通过以下方式配置使用自定义算法，以分片为例：

- a. 实现 ShardingAlgorithm 接口定义的算法实现类。
- b. 在项目 resources 目录下创建 META-INF/services 目录。
- c. 在 META-INF/services 目录下新建文件 org.apache.shardingsphere.sharding.spi.ShardingAlgorithm
- d. 将实现类的绝对路径写入至文件 org.apache.shardingsphere.sharding.spi.ShardingAlgorithm
- e. 将上述 Java 文件打包成 jar 包。
- f. 将上述 jar 包拷贝至 DBPlusEngine-Proxy 解压后的 ext-lib/ 目录。
- g. 将上述自定义算法实现类的 Java 文件引用配置在 YAML 文件中，具体可参考配置规则。

6. 注意事项

- a. DBPlusEngine-Proxy 默认使用 3307 端口，可以通过启动脚本追加参数作为启动端口号。如: bin/start.sh 3308
- b. DBPlusEngine-Proxy 使用 conf/server.yaml 配置注册中心、认证信息以及公用属性。
- c. DBPlusEngine-Proxy 支持多逻辑数据源，每个以 config- 前缀命名的 YAML 配置文件，即为一个逻辑数据源。
- d. DBPlusEngine-Proxy 默认监听地址 0.0.0.0，可以通过启动脚本指定监听地址，使用方式可通过 bin/start.sh -help 获取。

使用 SpehreEx-Console 停止计算节点

适用场景

停止处于启动状态的节点

注意事项

- 节点处于启动状态
- 最后一个计算节点被停止后集群会停止

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **集群**进入集群列表。
3. 点击操作列 **管理节点**，默认进入计算点列表。
4. 点击列表操作列中的 **停止**按钮完成节点停止。

单独停止计算节点

Linux 操作系统请运行 bin/stop.sh，Windows 操作系统请运行 bin/stop.bat 启动 ShardingSphere-Proxy。

删除

适用场景

从集群中计算节点

注意事项

勾选同步卸载软件会同步卸载计算节点的软件，并且卸载监控和日志配置

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 点击 **集群管理-> 集群**进入集群列表。
- 3. 点击操作列 **管理节点**，默认进入计算点列表。
- 4. 单击列表中操作列的 **删除**按钮。
- 5. 点击提示框中的 **确认**，节点的删除。

存储节点管理功能说明

功能	说明
注册	选择注册的数据库中添加存储节点到集群
刷新	刷新节点
探活	探活节点
修改	修改存储节点
删除	从 Console 列表中删除，正在被逻辑库引用的存储节点，不能被删除。删除不会影响其所属实例

注册

适用场景

集群扩容存储节点

注意事项

- 数据库实例已经被注册。
- 新添加的存储节点，属于集群，且未被使用。

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 点击 **集群管理-> 集群**进入集群列表。
- 3. 点击操作列 **管理节点**，点击 **存储节点**进入存储点列表。
- 4. 单击右上角 **添加节点**按钮，进入添加节点页面。
- 5. 填写信息，具体参数见下表。

字段	数据来源	选 填/必填	长 度 限 制	说明
所属实例	数据库实例列表	必填		实例 IP、实例端口、数据库唯一性校验，不可以重复添加。
实例 IP	自动填充	必填		不可编辑
实例端口	自动填充	必填		不可编辑
实例类型	自动填充	必填		不可编辑
用户名	自动填充	必填	32	可以更改
口令	自动填充	必填		可以更改
数据库	数据库实例内部	必填		可以更改
节点名称	用户输入	必填		
扩展参数	用户输入	选填		可以更改
URL 连接串	自动填充，不可编辑			灰色不可编辑

6. 点击 **确认**完成节点添加。

探活

适用场景

探活存储节点

注意事项

集群已经被注册或安装

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **集群**进入集群列表。
3. 点击操作列 **管理节点**，点击 **存储节点**进入存储点列表。
4. 点击列表操作列中的 **探活**按钮完成节点探活。

刷新

适用场景

刷新存储节点

注意事项

集群已经被注册或安装

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **集群**进入集群列表。
3. 点击操作列 **管理节点**，点击 **存储节点**进入存储点列表。
4. 点击列表操作列中的 **刷新**按钮完成节点刷新。

修改

适用场景

修改存储节点信息

注意事项

只能修改存储节点的用户名密码，适用于用户存储节点的用户名密码发生变更的场景，修改存储节点的用户名密码，也不会影响到其所属实例。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **集群**进入集群列表。
3. 点击操作列 **管理节点**，点击 **存储节点**进入存储点列表。
4. 点击列表操作列中的 **修改**按钮，进入修改存储节点页面。
5. 填写信息，具体参数见下表。

字段	数据来源	选填/必填	长度限制	说明
用户名	自动填充	必填	32	可以更改
口令	自动填充	必填		可以更改
扩展参数	用户输入	选填		可以更改

删除

适用场景

删除未被使用的存储节点

注意事项

存储节点为被逻辑库使用

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **集群**进入集群列表。
3. 点击操作列 **管理节点**，点击 **存储节点**进入存储点列表。
4. 点击列表操作列中的 **删除**按钮，完成存储节点的删除。

逻辑库管理

逻辑库管理功能列表

功能	说明
新建	新建逻辑库
刷新	刷新逻辑库
慢日志	查看逻辑库的慢日志
存储节点	为逻辑库增加可用的存储节点
删除	从集群中删除逻辑库，不会删除集群中的存储节点

新建

适用场景

新建逻辑库

注意事项

- 集群可以正常访问，集群中有可用的存储节点
- 未被任何逻辑库的使用的存储节点，为可用的存储节点

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **逻辑库**进入逻辑库列表。
3. 点击右上角 **新建**按钮，进入新建逻辑库的页面。
4. 填写信息，具体参数见下表。

字段	数据来源	选填/必填	长度限制	说明
逻辑库	用户输入	必填	30	校验唯一性，一套集群中的逻辑库名称保障唯一性
存储节点		选填		可选栏中为集群中可用的存储节点；已选栏中为逻辑库可用的存储节点；

5. 点击 **确认**完成逻辑库的新建。

刷新

适用场景

刷新逻辑库

注意事项

集群可以正常访问

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **逻辑库**进入逻辑库列表。
3. 点击右上角 **刷新**按钮，完成逻辑库刷新。

慢日志

适用场景

查看逻辑库产生的慢日志

注意事项

- 集群可以正常访问
- 慢日志已经打开
- 可以通过 SQL 语句、DistSQL 用户、计算节点、模糊查询；时间、SQL 类型，执行时长等值查询筛选慢日志

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **逻辑库**进入逻辑库列表。
3. 点击操作列 **慢日志**按钮，跳转至慢日志页面。

慢日志的内容包括：时间、用户、计算节点、逻辑库、SQL 类型、执行时长、SQL 语句。

悬停至 **查看执行计划**可以跳转至执行计划页面，查看慢日志的执行路径。

- 4. 点击 **导出**可以导出符合条件的所有日志。

存储节点

适用场景

管理逻辑库的存储节点

注意事项

- 集群可以正常访问
- 未逻辑表使用的存储节点可以移除

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 点击 **集群管理-> 逻辑库**进入逻辑库列表。
- 3. 点击操作列 **存储节点**按钮，跳转至存储节点管理页面。
- 4. 通过调整 **可选栏**和 **已选栏**中的存储节点，完成逻辑库存储节点的调整。

删除

适用场景

删除逻辑库

注意事项

- 集群可以正常访问。
- 从集群中删除逻辑库，不会删除集群的存储节点。
- 删除逻辑库会同步删除逻辑库中的插件配置，且不可恢复，请谨慎操作。

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 点击 **集群管理-> 逻辑库**进入逻辑库列表。
- 3. 点击操作列 **删除**按钮。
- 4. 点击提示中的 **确认**按钮完成逻辑库的删除。

对象管理

表

功能	说明
新建	新建逻辑表
表组管理	表组管理
查看表结构	查看建表（create table）语句
查看表分布	查看表的物理分布
修改表	修改表结构
删除表	删除表

插件

功能	说明
添加分片	为已经存在的表增加分片插件配置
修改分片	为已经存在的表修改分片插件配置
添加加密	为已经存在的表增加加密插件配置
修改加密	为已经存在的表修改加密插件配置

新建表

适用场景

在逻辑库中新建表

注意事项

集群可以正常访问，已有逻辑库

操作步骤

1. 登录 SphereEx-Console。
2. 点击 集群管理-> 对象进入表对象列表。
3. 点击右上角 新建表按钮，进入新建表的页面。新建表时，分片插件和加密插件可选添加；分片插件和加密插件分别可以使用常规和自定义的方式创建。
4. 填写信息，具体参数见下表。表配置

字段	数据来源	选填/必填	长度限制	说明
表类型	选择	必选		选择分片表时出现存储节点及以下信息选择广播表时不出现存储节点及以下信息

添加分片配置常规配置

分片插件配置：常规表示格式化的定义规则；自定义表示使用 DistSQL 定义规则，常规方式不用定义存储节点；常规和自定义二选一配置，选择哪种方式就出现对应的配置项。

字段	数据来源	选填/必填	长度限制	说明
存储节点	勾选逻辑库的存储节点	必填		可多选，存储节点的节点名称填充至存储节点的文本框中
存储节点分片键	自动填充表结构中解析出的字段	必填 必填		多个存储节点的名称逗号分隔，不可编辑 结合分片算法校验
分片算法	内置算法	必填		结合分片键的类型校验
算法参数	用户输入	必填	不限	根据分片算法校验参数, 根据不同算法提供参数示例
分布式序列	表结构中解析出的字段	选填		需要使用主键生成策略的列
分布式序列生成策略	内置算法			结合键值列的类型和长度校验【分布式序列】有数据时【分布式序列生成策略】为必填【分布式序列】无数据时【分布式序列生成策略】此行不显示
策略参数	用户输入		不限	根据键值生成策略法校验参数，根据不同策略提供参数示例【分布式序列生成策略】有数据时【策略参数】为必填【分布式序列生成策略】无数据时【策略参数，此行不显示

添加分片配置自定义配置

字段	数据来源	选填/必填	长度限制	说明
	用户输入	必填	不限	DistSQL，执行时校验

添加加密配置常规配置

加密插件配置：常规表示格式化的定义规则；自定义表示使用 DistSQL 定义规则，常规和自定义二选一配置，选择哪种方式就出现对应的配置项。

【使用加密列查询】勾选，查询时使用加密后数据列；不勾选，查询时使用加密前数据列。

默认为勾选。

当有明文存储未勾选时，【使用加密列查询】必须勾选，且不能更改。

当明文存储全部勾选时，【使用加密列查询】允许改为不勾选。

字段	数据来源	选填/必填	长度限制	说明
加密算法	内置算法	必填		结合逻辑列逻辑列类型长度，密文列数据类型长度校验
加密算法参数	用户输入	必填	不限	根据加密算法校验参数, 根据不同算法提供参数示例
需要加密的列	下拉选择	必选		需要加密的列为逻辑列，明文列和密文列自动创建，明文列的数据类型和长度取自用户表中需要加密的列的数据类型和长度；密文列的数据类型和长度根据算法定义，同时添加两个加密列的时候，添加第二个加密列时筛选时不能出现第一个被加密的字段
存储明文	用户勾选			勾选，在数据库中存储明文; 不勾选，不在数据库中存储明文

添加加密配置自定义配置

字段	数据来源	选填/必填	长度限制	说明
	用户输入	必填	不限	Distsql，执行时校验

5. 点击 确认完成表的新建。

表组管理

适用场景

将部署了相同算法的分片插件绑定成组，用于优化查询。

注意事项

集群可以正常访问，已有逻辑库，表的分片插件配置一致

操作步骤

1. 登录 SphereEx-Console。

2. 点击 **集群管理**-> **对象进入表对象列表**。
3. 点击右上角 **表组管理**按钮，进入表组管理的页面。
4. 填写信息，具体参数见下表。

字段	数据来源	选填/必填	长度限制	说明
首 个 成员	下拉列表中选择当前逻辑库中分片表	必填		用户预期绑定规则的表中任选一个，无主次之分
其 他 成员	下拉列表中选择当前逻辑库中分片表，符合与首个成员绑定表规则的其他表	必填		可多选

5. 点击 **确认**完成表组的管理。

查看表结构

适用场景

查看 create table 形式的表结构，无法看到插件配置。

注意事项

集群可以正常访问，已有逻辑库，已有表。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **对象进入表对象列表**。
3. 点击操作列 **查看表结构**按钮，进入查看表结构的页面。
4. 点击 **返回**，返回表对象列表。

查看表分布

适用场景

查看表所在的存储节点

注意事项

集群可以正常访问，已有逻辑库，已有表。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **对象进入表对象列表**。
3. 点击操作列 **查看表分布**按钮，进入查看表分布的页面。
4. 点击 **返回**，返回表对象列表。

修改表

适用场景

修改表结构

注意事项

集群可以正常访问，已有逻辑库，已有表。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **对象**进入表对象列表。
3. 点击操作列 **修改表**按钮，进入修改表的页面。
4. 填写修改表结构的 SQL，可以点击示例，查看语句示例。
5. 点击 **确认**完成表修改。

删除

适用场景

删除表，删除插件配置，删除表结构和数据。

注意事项

集群可以正常访问，已有逻辑库，已有表，删除是物理删除，无法恢复，请谨慎操作。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **对象**进入表对象列表。
3. 点击操作列 **删除**按钮，进入修改表的页面。
4. 点击 **确认**完成表删除。

插件-添加分片

适用场景

在已有表中增加分片插件配置。

注意事项

集群可以正常访问，已有逻辑库，已有表，表中无数据，表中无分片插件配置。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **对象**进入表对象列表。
3. 点击插件列 **添加分片**按钮，进入添加分片的页面。
4. 填写信息，具体参数见下表。

分片插件配置：常规表示格式化的定义规则；自定义表示使用 DistSQL 定义规则，常规方式不用定义存储节点；常规和自定义二选一配置，选择哪种方式就出现对应的配置项。

字段	数据来源	选填/必填	长度限制	说明
存储节点	勾选逻辑库的存储节点	必填		可多选，存储节点的节点名称填充至存储节点的文本框中
存储节点	自动填充	必填		多个存储节点的名称逗号分隔，不可编辑
分片键	表结构中解析出的字段	必填		结合分片算法校验
分片算法	内置算法	必填		结合分片键的类型校验
算法参数	用户输入	必填	不限	根据分片算法校验参数, 根据不同算法提供参数示例
分布式序列	表结构中解析出的字段	选填		需要使用主键生成策略的列
分布式序列生成策略	内置算法			结合键值列的类型和长度校验【分布式序列】有数据时【分布式序列生成策略】为必填；【分布式序列】无数据时【分布式序列生成策略】此行不显示
策略参数	用户输入	不限		根据键值生成策略法校验参数, 根据不同策略提供参数示例【分布式序列生成策略】有数据时【策略参数】为必填【分布式序列生成策略】无数据时【策略参数】此行不显示

自定义

字段	数据来源	选填/必填	长度限制	说明
	用户输入	必填	不限	DistSQL, 执行时校验

5. 点击 **确认完成分片插件配置** 的添加。

插件-修改分片

适用场景

在已有表中增加分片插件配置。

注意事项

集群可以正常访问，已有逻辑库，已有表，表中无数据，表中已有分片插件配置。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理-> 对象** 进入表对象列表。
3. 点击插件列 **修改分片** 按钮，进入修改分片的页面。进入修改分片插件的页面，可以看到分片插件的配置，如果是常规方式添加的分片插件，会以常规方式显示；如果是自定义方式添加的分片插件，会以自定义方式显示。
4. 填写信息，具体参数见下表。分片插件配置：常规表示格式化的定义规则；自定义表示使用 DistSQL 定义规则，常规方式不用定义存储节点；常规和自定义二选一配置，选择哪种方式就出现对应的配置项。

字段	数据来源	选填/必填	长度限制	说明
存储节点	勾选逻辑库的存储节点	必填		可多选，存储节点的节点名称填充至存储节点的文本框中
存储节点	自动填充	必填		多个存储节点的名称逗号分隔，不可编辑
分片键	表结构中解析出的字段	必填		结合分片算法校验
分片算法	内置算法	必填		结合分片键的类型校验
算法参数	用户输入	必填	不限	根据分片算法校验参数, 根据不同算法提供参数示例
分布式序列	表结构中解析出的字段	选填		需要使用主键生成策略的列
分布式序列生成策略	内置算法			结合键值的类型和长度校验【分布式序列】有数据时【分布式序列生成策略】为必填；【分布式序列】无数据时【分布式序列生成策略】此行不显示
策略参数	用户输入	不限		一 根据键值生成策略法校验参数, 根据不同策略提供参数示例【分布式序列生成策略】有数据时【策略参数】为必填【分布式序列生成策略】无数据时【策略参数】此行不显示

自定义

字段	数据来源	选填/必填	长度限制	说明
	用户输入	必填	不限	DistSQL，执行时校验

5. 点击 **确认** 完成分片插件配置的修改。

插件-添加加密

适用场景

在已有表中增加加密插件配置。

注意事项

集群可以正常访问，已有逻辑库，已有表，表中无加密插件配置，不涉及数据清洗。

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 点击集群管理-> 对象进入表对象列表。
- 3. 点击插件列添加加密按钮，进入添加加密的页面。
- 4. 填写信息，具体参数见下表。

加密插件配置：常规表示格式化的定义规则；自定义表示使用 DistSQL 定义规则，常规和自定义二选一配置，选择哪种方式就出现对应的配置项【使用加密列查询】勾选，查询时使用加密后数据列；不勾选，查询时使用加密前数据列；

默认为勾选；

当有明文存储未勾选时，【使用加密列查询】必须勾选，且不能更改；

当明文存储全部勾选时，【使用加密列查询】允许改为不勾选。

常规

字段	数据来源	选填/必填	长度限制	说明
加密算法	内置算法	必填		结合逻辑列逻辑列类型长度，密文列数据类型长度校验
加密算法参数	用户输入	必填	不限	根据加密算法校验参数, 根据不同算法提供参数示例
需要加密的列	下拉选择	必选		需要加密的列为逻辑列，明文列和密文列自动创建，明文列的数据类型和长度取自用户表中需要加密的列的数据类型和长度；密文列的数据类型和长度根据算法定义，同时添加两个加密列的时候，添加第二个加密列时筛选时不能出现第一个被加密的字段
存储明文	用户勾选			勾选，在数据库中存储明文; 不勾选，不在数据库中存储明文

自定义

字段	数据来源	选填/必填	长度限制	说明
	用户输入	必填	不限	DistSQL，执行时校验

- 5. 点击 确认完成加密插件配置的添加。

插件-修改加密

适用场景

在已有表中增加分片插件配置。

注意事项

集群可以正常访问，已有逻辑库，已有表，表中已有加密插件配置，不涉及数据清洗。

操作步骤

- 1. 登录 SphereEx-Console。
 - 2. 点击 集群管理-> 对象进入表对象列表。
 - 3. 点击插件列 修改加密按钮，进入修改加密的页面。进入修改加密插件的页面，可以看到加密插件的配置，如果是常规方式添加的加密插件，会以常规方式显示；如果是自定义方式添加的加密插件，会以自定义方式显示加密插件配置：常规表示格式化的定义规则；自定义表示使用 DistSQL 定义规则，常规和自定义二选一配置，选择哪种方式就出现对应的配置项。
 - 4. 填写信息，具体参数见下表。【使用加密列查询】勾选，查询时使用加密后数据列；不勾选，查询时使用加密前数据列；默认为勾选；
- 当有明文存储未勾选时，【使用加密列查询】必须勾选，且不能更改；
- 当明文存储全部勾选时，【使用加密列查询】允许改为不勾选。

常规配置

字段	数据来源	选填/必填	长度限制	说明
加密算法	内置算法	必填		结合逻辑列逻辑列类型长度，密文列数据类型长度校验
加密算法参数	用户输入	必填	不限	根据加密算法校验参数, 根据不同算法提供参数示例
需要加密的列	下拉选择	必选		需要加密的列为逻辑列，明文列和密文列自动创建，明文列的数据类型和长度取自用户表中需要加密的列的数据类型和长度；密文列的数据类型和长度根据算法定义，同时添加两个加密列的时候，添加第二个加密列时筛选时不能出现第一个被加密的字段
存储明文	用户勾选			勾选，在数据库中存储明文; 不勾选，不在数据库中存储明文

自定义配置

字段	数据来源	选填/必填	长度限制	说明
	用户输入	必填	不限	DistSQL，执行时校验

- 5. 点击 确认完成加密插件配置的修改。

SQL 管理

功能	说明
慢日志	新建逻辑表
执行计划	表组管理
SQL 审计	查看建表 (create table) 语句

慢日志

适用场景

查看集群产生的慢日志

注意事项

- 集群可以正常访问
- 慢日志已经打开
- 可以通过 SQL 语句、DistSQL 用户、计算节点、模糊查询；时间、逻辑库、SQL 类型，执行时长等值查询筛选慢日志。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 SQL 管理-> 慢日志进入慢日志页面。慢日志的内容包括：时间、用户、计算节点、逻辑库、SQL 类型、执行时长、SQL 语句。

悬停至 查看执行计划可以跳转至执行计划页面，查看慢 SQL。

3. 点击导出可以导出符合条件的所有日志

执行计划

适用场景

查看 SQL 的执行路径

注意事项

集群可以正常访问

操作步骤

1. 登录 SphereEx-Console。
2. 点击 SQL 管理-> 执行计划进入慢日志页面。
3. 选择集群选择逻辑库输入 SQL 语句，点击 查看执行计划按钮，查看执行计划。

SQL 审计

适用场景

通过查看集群执行过的所有 SQL，做事后审计。

注意事项

- 集群可以正常访问
- 审计日志已经打开
- 可以通过 SQL 语句、DistSQL 用户、计算节点、模糊查询；时间范围、SQL 类型，等值查询、筛选审计日志。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **SQL 管理->SQL 审计** 进入 SQL 审计日志页面。审计日志的内容包括：时间、DistSQL 用户、计算节点、逻辑库、SQL 类型、SQL 语句。
3. 点击 **导出** 可以导出符合条件的所有日志。

日志管理

通过 SphereEx-Console 查看日志

适用场景

通过查看集群计算节点的运行日志。

注意事项

- 集群可以正常访问
- 可以通过关键字模糊查询；时间范围等值查询筛选运行日志。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **日志管理** 进入日志管理页面。
3. 点击 **导出** 可以导出符合条件的所有日志。

日志管理相关参数

system-log-level 系统日志级别，非内置 logger 均使用该配置 **system-log-pattern** 系统日志格式，全局通用
DistSQL 参数修改示例

```
SET DIST VARIABLE system_log_level = DEBUG  
SET DIST VARIABLE system_log_pattern = "[%-8level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %logger{39} - %msg%n"
```

通过 SphereEx-Console 调整运行日志参数请参考[集群管理参数管理](#)

通过 DistSQL 调整运行日志请参考[日志管理](#)

参数详解请参考 [集群参数列表](#)

连接集群

用户可以使用多种方式登录到集群：

- SphereEx-Console：用户可使用 SphereEx-Console，登录后通过 GUI 界面完成管理动作（内置受约束的 DistSQL，SQL 终端此次尚不提供）。
- Database Command：例如 MySQL Client、PG PSQL，登录到集群中任意计算节点。通过 USE 定位到逻辑库，执行 SQL。使用示例：mysql -uroot -p -P3307 -h127.0.0.1
- Third Database Tools：例如 navicate 等工具，部分功能受限，待测试 Todo。

9.3 容器化部署与管理

9.3.1 Helm 部署方式部署 SphereEx-DBPlusEngine-Proxy 集群

背景信息

使用 Helm 在 Kubernetes 集群中引导 SphereEx-DBPlusEngine-Proxy 集群进行安装。

前提条件

kubernetes 1.20+

kubectl

helm 3.8.1+

可以动态申请 PV(Persistent Volumes) 的 StorageClass 用于持久化数据。(可选)

操作步骤

最简安装

将 SphereEx-DBPlusEngine-Proxy 添加到 Helm 本地仓库：

```
helm repo add sphereex https://xxx
```

以 sphereex-dbplusengine 命名安装 charts：

```
helm install sphereex-dbplusengine sphereex/sphereex-dbplusengine
```

配置安装

运行时配置修改

以 sphereex-dbplusengine 命名安装 charts：配置项包括治理节点 Zookeeper 的数据持久化，mysql 驱动自动下载，治理中心 Namespace 配置

```
helm install sphereex-dbplusengine sphereex/sphereex-dbplusengine --set governance.zookeeper.persistence.enabled=true --set governance.zookeeper.persistence.storageClass="< 你的 storageClass 名称>" --set compute.mysqlConnector.version="<mysql 驱动版本>" --set compute.serverConfig.mode.repository.props.namespace="governance_ds"
```

修改 values.yaml 文件, 详细配置项参见配置示例

```
tar -zxvf sphereex-dbplusengine.tar.gz
```

```
vim values.yaml
```

```
helm install sphereex-dbplusengine
```

卸载

```
helm uninstall sphereex-dbplusengine
```

默认删除所有发布记录，增加 --keep-history 参数保留发布记录。

配置示例

治理节点配置项

配置项	描述	值
governance.enabled	用来切换是否使用治理节点的 chart	true

治理节点-ZooKeeper 配置项

配置项	描述	值
governance.zookeeper.enabled	用来切换是否使用 ZooKeeper 的 chart	true
governance.zookeeper.replicaCount	ZooKeeper 节点数量	1
governance.zookeeper.persistence.enabled	标识 ZooKeeper 是否使用持久卷申领 (PersistentVolumeClaim) 用来申请持久卷 (PersistentVolume)	false
governance.zookeeper.persistence.storageClass	持久卷 (PersistentVolume) 的存储类 (StorageClass)	
governance.zookeeper.persistence.accessModes	持久卷 (PersistentVolume) 的访问模式	["ReadWriteOnce"]
governance.zookeeper.persistence.size	持久卷 (PersistentVolume) 大小	8Gi
governance.zookeeper.resources.limits	ZooKeeper 容器的资源限制	{}
governance.zookeeper.resources.requests.memory	ZooKeeper 容器申请的内存	256Mi
governance.zookeeper.resources.requests.cpu	ZooKeeper 容器申请的 cpu 核数	250m

计算节点-SphereEx-DBPlusEngine-Proxy 配置项

配置项	描述	值
compute.image.repository	SphereEx-DBPlusEngine-Proxy 的镜像名	`apache/shardin`gsphere-proxy`
compute.image.pullPolicy	SphereEx-DBPlusEngine-Proxy 镜像拉取策略	`IfNotPresent`
`compute.image.tag`	SphereEx-DBPlusEngine-Proxy 镜像标签	5.1.2
compute.imagePullSecrets.username	拉取私有仓库的用户名	""
compute.imagePullSecrets.password	拉取私有仓库的密码	""
compute.resources.limits	SphereEx-DBPlusEngine-Proxy 容器的资源限制	{}
compute.resources.requests.memory	SphereEx-DBPlusEngine-Proxy 容器申请的内存	2Gi
compute.resources.requests.cpu	SphereEx-DBPlusEngine-Proxy 容器申请的cpu核数	200m
`compute.replicas`	SphereEx-DBPlusEngine-Proxy 节点个数	3
compute.service.type	SphereEx-DBPlusEngine-Proxy 网络模式	ClusterIP
compute.service.port	SphereEx-DBPlusEngine-Proxy 暴露端口	3307
compute.mysqlConnector.version	MySQL 驱动版本	5.1.49
`compute.startPort`	SphereEx-DBPlusEngine-Proxy 启动端口	3307
compute.agent.enabled	SphereEx-DBPlusEngine-Proxy Engine 启动配置	false
compute.serverConfig	SphereEx-DBPlusEngine-Proxy 模式配置文件	""

计算节点 DBPlusEngine-Proxy Server 配置权限配置项

配置项	描述	值
serverConfig.authority.privilege.type	存储节点数据授权的权限提供者类型，缺省值为 ALL_PERMITTED	ALL_PERMITTED
serverConfig.authority.users[0].password	用于登录计算节点的密码	`root`
serverConfig.authority.users[0].user	用于登录计算节点的用户名，授权主机。格式: @hostname 为% 或空字符串表示不限制授权主机	root@%

计算节点 DBPlusEngine-Proxy Server 配置模式配置项

配置项	描述	值
serverConfig.mode.type	运行模式类型。现阶段仅支持 Cluster 模式	Cluster
serverConfig.mode.repository.props.namespace	注册中心命名空间	governance_ds
serverConfig.mode.repository.props.server-lists	注册中心连接地址	{{ printf "%s-zookeeper.%s:2181" .Release.Name .Release.Namespace }}
serverConfig.mode.repository.props.maxRetries	客户端连接最大重试次数	3
serverConfig.mode.repository.props.operationTimeoutMilliseconds	客户端操作超时的毫秒数	5000
serverConfig.mode.repository.props.retryIntervalMilliseconds	重试间隔毫秒数	500
serverConfig.mode.repository.props.timeToLiveSeconds	临时数据失效的秒数	60
serverConfig.mode.repository.type	持久化仓库类型。现阶段仅支持 ZooKeeper	ZooKeeper

计算节点 DBPlusEngine-Proxy Server 配置 props 配置项

配置项	描述	值
comp ute.serverConfig.props.proxy-front-end-database-protocol-type	后端协议类型 (支持 PostgreSQL、openGauss、MariaDB、MySQL), 如果需要进行其他非 MySQL 后端数据源链接, 请更换此项配置	MySQL

说明: 支持设置 CGROUP_MEM_OPTS 环境变量: 用于在容器环境中设置相关内存参数, 脚本中的默认值为: -XX:InitialRAMPercentage=80.0 -XX:MaxRAMPercentage=80.0 -XX:MinRAMPercentage=80.0

配置示例

values.yaml

```
#
# Copyright © 2022, Beijing Sifei Software Technology Co., LTD.
# All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
## @section Governance-Node parameters
```

```
## @param governance.enabled Switch to enable or disable the governance helm chart
##
governance:
  enabled: true
  ## @section Governance-Node ZooKeeper parameters
  zookeeper:
    ## @param governance.zookeeper.enabled Switch to enable or disable the ZooKeeper helm chart
    ##
    enabled: true
    ## @param governance.zookeeper.replicaCount Number of ZooKeeper nodes
    ##
    replicaCount: 3
    ## ZooKeeper Persistence parameters
    ## ref: https://kubernetes.io/docs/user-guide/persistent-volumes/
    ## @param governance.zookeeper.persistence.enabled Enable persistence on ZooKeeper using PVC(s)
    ## @param governance.zookeeper.persistence.storageClass Persistent Volume storage class
    ## @param governance.zookeeper.persistence.accessModes Persistent Volume access modes
    ## @param governance.zookeeper.persistence.size Persistent Volume size
    ##
    persistence:
      enabled: true
      storageClass: ""
      accessModes:
        - ReadWriteOnce
      size: 8Gi
    ## ZooKeeper's resource requests and limits
    ## ref: https://kubernetes.io/docs/user-guide/compute-resources/
    ## @param governance.zookeeper.resources.limits The resources limits for the ZooKeeper containers
    ## @param governance.zookeeper.resources.requests.memory The requested memory for the ZooKeeper containers
    ## @param governance.zookeeper.resources.requests.cpu The requested cpu for the ZooKeeper containers
    ##
    resources:
      limits: {}
      requests:
        memory: 4Gi
        cpu: 2

## @section Compute-Node parameters
##
compute:
  ## @section Compute-Node ShardingSphere-Proxy parameters
  ## ref: https://kubernetes.io/docs/concepts/containers/images/
  ## @param compute.image.repository Image name of ShardingSphere-Proxy.
  ## @param compute.image.pullPolicy The policy for pulling ShardingSphere-Proxy image
  ## @param compute.image.tag ShardingSphere-Proxy image tag
  ##
  image:
    repository: "uhub.service.ucloud.cn/sphere-ex/sphereex-dbplusengine-proxy"
    pullPolicy: IfNotPresent
    ## Overrides the image tag whose default is the chart appVersion.
    ##
    tag: "1.3.1"
  imagePullSecrets:
    username: ""
    password: ""
  ## ShardingSphere-Proxy resource requests and limits
  ## ref: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
  ## @param compute.resources.limits The resources limits for the ShardingSphere-Proxy containers
  ## @param compute.resources.requests.memory The requested memory for the ShardingSphere-Proxy containers
  ## @param compute.resources.requests.cpu The requested cpu for the ShardingSphere-Proxy containers
  ##
  resources:
    requests:
      cpu: 2
```



```

memory: 2Gi
limits:
  cpu: 4
  memory: 4Gi
## ShardingSphere-Proxy Deployment Configuration
## ref: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
## ref: https://kubernetes.io/docs/concepts/services-networking/service/
## @param compute.replicas Number of cluster replicas
##
replicas: 3
## @param compute.service.type ShardingSphere-Proxy network mode
## @param compute.service.port ShardingSphere-Proxy expose port
##
service:
  type: LoadBalancer
  port: 3307
## MySQL connector Configuration
## ref: https://shardingsphere.apache.org/document/current/en/quick-start/shardingsphere-proxy-quick-start/
## @param compute.mysqlConnector.version MySQL connector version
##
mysqlConnector:
  version: ""
## @param compute.startPort ShardingSphere-Proxy start port
## ShardingSphere-Proxy start port
## ref: https://shardingsphere.apache.org/document/current/en/user-manual/shardingsphere-proxy/startup/docker/
##
startPort: 3307
agent:
  enabled: true
terminationGracePeriodSeconds: 30
## @section Compute-Node ShardingSphere-Proxy ServerConfiguration parameters
## NOTE: If you use the sub-charts to deploy Zookeeper, the server-lists field must be "{{ printf \"%s-zookeeper.%s:2181\" .
Release.Name .Release.Namespace }}" ,
## otherwise please fill in the correct zookeeper address
## The server.yaml is auto-generated based on this parameter.
## If it is empty, the server.yaml is also empty.
## ref: https://shardingsphere.apache.org/document/current/en/user-manual/shardingsphere-jdbc/yaml-config/mode/
## ref: https://shardingsphere.apache.org/document/current/en/user-manual/shardingsphere-jdbc/builtin-algorithm/
metadata-repository/
##
serverConfig:
  ## @section Compute-Node ShardingSphere-Proxy ServerConfiguration authority parameters
  ## NOTE: It is used to set up initial user to login compute node, and authority data of storage node.
  ## ref: https://shardingsphere.apache.org/document/current/en/user-manual/shardingsphere-proxy/yaml-config/
authentication/
  ## @param compute.serverConfig.authority.privilege.type authority provider for storage node, the default value is ALL_
  PERMITTED
  ## @param compute.serverConfig.authority.users[0].password Password for compute node.
  ## @param compute.serverConfig.authority.users[0].user Username,authorized host for compute node. Format: <username>@
  <hostname> hostname is % or empty string means do not care about authorized host
  ##
  authority:
    privilege:
      type: ALL_PERMITTED
    users:
      - password: root
        user: root@%
  ## @section Compute-Node ShardingSphere-Proxy ServerConfiguration mode Configuration parameters
  ## @param compute.serverConfig.mode.type Type of mode configuration. Now only support Cluster mode
  ## @param compute.serverConfig.mode.repository.props.namespace Namespace of registry center
  ## @param compute.serverConfig.mode.repository.props.server-lists Server lists of registry center
  ## @param compute.serverConfig.mode.repository.props.maxRetries Max retries of client connection
  ## @param compute.serverConfig.mode.repository.props.operationTimeoutMilliseconds Milliseconds of operation timeout
  ## @param compute.serverConfig.mode.repository.props.retryIntervalMilliseconds Milliseconds of retry interval

```

```
## @param compute.serverConfig.mode.repository.props.timeToLiveSeconds Seconds of ephemeral data live
## @param compute.serverConfig.mode.repository.type Type of persist repository. Now only support ZooKeeper
## @param compute.serverConfig.mode.override Whether overwrite persistent configuration with local configuration
##
mode:
repository:
  props:
    maxRetries: 3
    namespace: ""
    operationTimeoutMilliseconds: 5000
    retryIntervalMilliseconds: 500
    server-lists: "{{ printf \"%s-zookeeper.%s:2181\" .Release.Name .Release.Namespace }}"
    timeToLiveSeconds: 600
  type: ZooKeeper
type: Cluster
props:
  proxy-frontend-database-protocol-type: MySQL
```

9.3.2 云上水平自动扩缩容 (HPA)

定义

在 Kubernetes 中，HorizontalPodAutoscaler (HPA) 自动更新工作负载资源，目的是自动扩缩工作负载以满足需求。

SphereEx-Operator 利用 Kubernetes 中 HPA 功能，结合 SphereEx-DBPlusEngine 的相关指标在 Kubernetes 集群中进行运行时自动扩缩容。

在开启自动扩缩容功能后，SphereEx-Operator 会在部署一个 SphereEx-DBPlusEngine 集群的同时在 Kubernetes 集群中应用一个 HPA 对象。

相关概念

■ HPA

HPA (HorizontalPodAutoscaler) 是指 SphereEx-Operator 利用 Kubernetes 中 HPA 功能对 SphereEx-DBPlusEngine 集群进行自动弹性扩缩容。

对系统的影响

在开启自动扩缩容之后，手动设置 SphereEx-DBPlusEngine 在 Kubernetes 中的副本将不会生效，SphereEx-DBPlusEngine 集群的副本数由 HPA 控制器的最大和最小值进行控制，进行弹性扩缩容将会在这两个值之间进行处理。

SphereEx-DBPlusEngine 的最小启动副本数也会由 HPA 的最小副本数进行控制，开启自动扩缩容功能后，SphereEx-DBPlusEngine 将以 HPA 的最小副本数进行启动。

使用 HPA 进行的扩缩容是对 SphereEx-DBPlusEngine 进行水平扩缩容。水平扩缩意味着对增加的负载的响应是部署更多的 Pod。

这与“垂直 (Vertical)”扩缩不同，对于 Kubernetes，垂直扩缩意味着将更多资源（例如：内存或 CPU）分配给已经为工作负载运行的 Pod。

使用限制

- 现阶段受限于 SphereEx-DBPlusEngine 指标不够丰富，只能通过运行时 CPU 进行压力负载测试。未来会增加其余指标，丰富对于 SphereEx-DBPlusEngine 的运行时压力计算方式。
- 如果想要使用 SphereEx-DBPlusEngine 在 Kubernetes 中的 HPA 功能，需要您的集群中安装 metrics-server 并能够正常使用 kubectl top 功能。
- SphereEx-Operator 在创建 SphereEx-DBPlusEngine 集群的同时，会在 SphereEx-DBPlusEngine 集群前方建立负载均衡。您的应用和 SphereEx-DBPlusEngine 是通过负载均衡建立链接。
- 因为 SphereEx-DBPlusEngine 和您的应用建立的是长链接，所以在扩容的过程中，对已有的长链接不会有明显的降低负载的效果。扩容的效果只会生效在扩容后新建立的链接上。
- 缩容也会有相对应的问题，在缩容的过程中，您的应用会有闪断的现象，是因为缩容的 SphereEx-DBPlusEngine 副本处于缩容的过程中，会从负载均衡上摘除，您的应用和 SphereEx-DBPlusEngine 建立的长链接也会随之销毁。

原理介绍

使用 HPA 进行的扩缩容是对 SphereEx-DBPlusEngine 进行水平扩缩容。水平扩缩意味着对增加的负载的响应是部署更多的 Pod。

这与“垂直 (Vertical)”扩缩不同，对于 Kubernetes，垂直扩缩意味着将更多资源（例如：内存或 CPU）分配给已经为工作负载运行的 Pod。

如果负载减少，并且 Pod 的数量高于配置的最小值，HorizontalPodAutoscaler 会指示工作负载资源（Deployment、StatefulSet 或其他类似资源）缩减。

在 Kubernetes 集群中，有一个控制器会按照一定的时间间隔对相关资源创建的 HPA 中的指标进行查询。在满足指标的阈值之后会按照计算公式对相对应的资源进行扩容或者缩容。

在 SphereEx-Operator 工作过程中，HPA 对象作用于 SphereEx-DBPlusEngine 的 Deployment 部署对象，并且持续的查询 SphereEx-DBPlusEngine 每一个副本的 CPU 利用率。

SphereEx-DBPlusEngine 的 CPU 利用率从该容器的 `/sys/fs/cgroup/cpu/cpuacct.usage` 中获取 CPU 的使用情况，并且以 shardingsphere.sphere-ex.com/v1alpha1.proxy 中 `automaticScaling.target` 字段设置的值为阈值的百分比，进行持续的计算。

当计算的数值达到阈值之后，HPA 控制器按照如下的公式进行副本数的计算：

$$\text{期望副本数} = \text{ceil}[\text{当前副本数} * (\text{当前指标} / \text{期望指标})]$$

值得注意的一点，CPU 利用率这一指标，是计算的每一个副本的 `resources.requests` 字段中 CPU 的数值。

在检查容差并决定最终值之前，控制平面还会考虑是否缺少任何指标，以及有多少 Pod 已就绪。

当使用 CPU 指标来扩缩时，任何还未就绪（还在初始化，或者可能是不健康的）状态的 Pod 或最近的指标度量值采集于就绪状态前的 Pod，该 Pod 也会被搁置。

原理图：

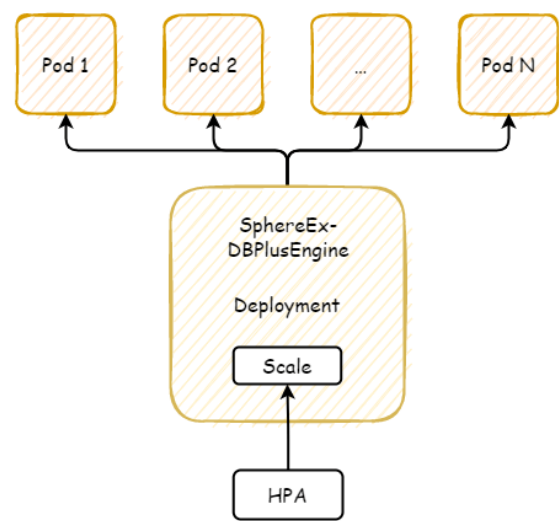


Fig. 5: 原理图

参数解释

Name	Description	Default Value
automaticScaling.enable	SphereEx-DB PlusEngine-Proxy 集群是否开启自动扩缩容	FALSE
automaticScaling.scaleUpWindows	SphereEx-DBPlusEngine-Proxy 自动扩容稳定窗口	30
automaticScaling.scaleDownWindows	SphereEx-DBPlusEngine-Proxy 自动缩容稳定窗口	30
automaticScaling.target	SphereEx-DBPlusEngine-Proxy 自动扩缩容阈值，数值为百分比，注意：现阶段暂时只支持 cpu 为指标进行扩缩容	70
automaticScaling.maxInstance	SphereEx-DBPlusEngine-Proxy 最大扩容副本数	4
automaticScaling.minInstance	SphereEx-DBPlusEngine-Proxy 最小启动副本数，缩容不会小于这个副本数	1

注意事项

在开启 SphereEx-DBPlusEngine 的 automaticScaling 功能时，将会由 HPA 接管 SphereEx-DBPlusEngine 的副本数，您的 SphereEx-DBPlusEngine 有可能会进行缩容操作，应用程序会出现闪断。

在开启 SphereEx-DBPlusEngine 的 automaticScaling 功能时，相对应的 HPA 会被删除。

操作步骤

- 在按照下述配置修改完毕 values.yaml 之后执行 helm install 新建一个 SphereEx-DBPlusEngine 集群。
- 或者使用 helm upgrade 更新现有的 SphereEx-DBPlusEngine 集群配置。

配置示例

如果想在 SphereEx-Operator 开启 SphereEx-DBPlusEngine 的自动扩缩容功能，需要在 SphereEx-DBPlusEngine-cluster charts 的 values.yaml 中开启如下配置：

```
automaticScaling:
  enable: true
  scaleUpWindows: 30
  scaleDownWindows: 30
  target: 20
  maxInstance: 4
  minInstance: 2
```

9.3.3 使用 Operator

什么是 DBPlusEngine-Operator

Kubernetes 的 Operator 模式允许你在不修改 Kubernetes 自身代码的情况下，通过一个或多个自定义资源关联控制器来扩展集群的能力。Operator 是 Kubernetes API 的客户端，充当自定义资源的控制器。

Operator 模式旨在捕获（正在管理一个或一组服务的）运维人员的关键目标。负责特定应用和 service 的运维人员，在系统应该如何运行、如何部署以及出现问题时如何处理等方面有深入的了解。

DBPlusEngine-Operator 帮助用户在 Kubernetes 环境中快速部署一套 DBPlusEngine-Proxy 集群，并且负责部署和维护这个集群周边的相关资源，监控集群状态。

DBPlusEngine-Mate 是 SphereEx 公司基于 Kubernetes 云原生自研的治理中心组件。

名词解释

CRD (CustomResourceDefinition) 用户自定义资源定义，代表 DBPlusEngine-Operator 将利用由 CRD 定义的 CR (Custom-Resource)，在 Kubernetes 集群中部署一套完整的 DBPlusEngine-Proxy 集群。

产品优势

- 配置简化

只需要编写简单的 yaml 即可在集群中部署一套完整的 DBPlusEngine-Proxy 集群。

- 易扩展

通过对 CR yaml 的修改，即可做到水平扩缩容等一系列功能。

- 易运维

使用 DBPlusEngine-Operator 将不用人为干预 DBPlusEngine-Proxy 在集群中的状态，DBPlusEngine-Operator 会自动检测集群状态并纠正。

逻辑架构

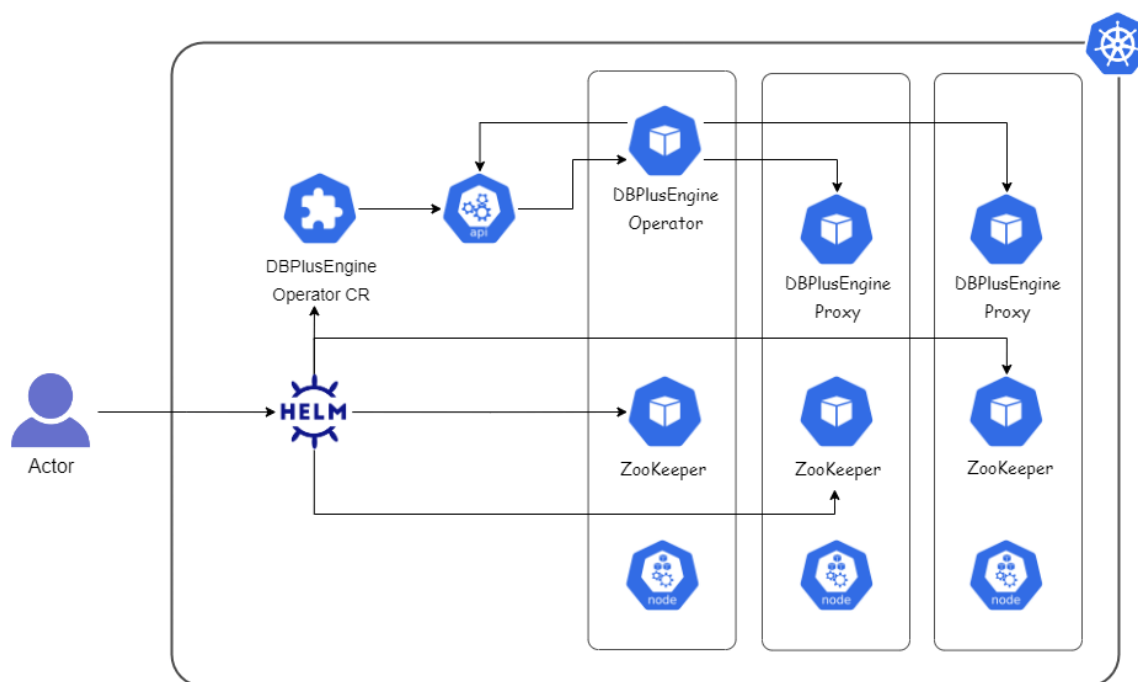


Fig. 6: 逻辑架构

安装 DBPlusEngine-Operator

配置 Operator Parameters, 配置文件位置 shardingsphere-operator/values.yaml。

执行

```
kubectl create ns dbplusengine-operator
helm install dbplusengine-operator dbplusengine-operator -n dbplusengine-operator
```

安装 DBPlusEngine-Proxy cluster

配置 Cluster Parameters, 配置文件位置 shardingsphere-cluster/values.yaml。

将 sphere-ex.license 移动至 shardingsphere-cluster/license，并保持命名 sphere-ex.license。

```
kubectl create ns dbplusengine
helm install dbplusengine-proxy dbplusengine-proxy -n dbplusengine
```

Operator Parameters

DBPlusEngine-Proxy operator parameters

Name	Description	Value
replicaCount	operator 副本数	2
image.repository	operator 镜像名	sphereex/dbplusengine-operator
image.pullPolicy	镜像拉取策略	IfNotPresent
image.tag	镜像 tag	0.0.1
imagePullSecrets	私有仓库镜像拉取密钥	[]
resources	operator 需要的资源	{}
webhook.port	operator webhook 启动端口	9443
health.healthProbePort	operator 健康检查端口	8081

Cluster Parameters

DBPlusEngine-Proxy cluster parameters

Name	Description	Value
replicaCount	DBPlusEngine-Operator-Cluster 集群启动副本数，注意：在开启 automaticScaling 后，这个参数将不再生效	"1"
automaticScaling.enable	DBPlusEngine-Operator-Cluster 集群是否开启自动扩缩容	false
automaticScaling.scaleUpWindows	DBPlusEngine-Operator-Cluster 自动扩容稳定窗口	30
automaticScaling.scaleDownWindows	DBPlusEngine-Operator-Cluster 自动缩容稳定窗口	30
automaticScaling.target	DBPlusEngine-Operator-Cluster 自动扩缩容阈值，数值为百分比，注意：现阶段暂时只支持 cpu 为指标进行扩缩容	70
automaticScaling.maxInstance	DBPlusEngine-Operator-Cluster 最大扩容副本数	4
automaticScaling.minInstance	DBPlusEngine-Operator-Cluster 最小启动副本数，缩容不会小于这个副本数	1
image.registry	DBPlusEngine-Operator-Cluster 镜像 host	docker.io
image.repository	DBPlusEngine-Operator-Cluster 镜像仓库名	sphereex/dbplusengine-proxy
image.tag	DBPlusEngine-Operator-Cluster 镜像 tag	5.1.2
resources	DBPlusEngine-Operator-Cluster 启动需求资源，在开启 automaticScaling 后，以 request 的资源乘以 target 的百分比为触发扩缩容动作的实际使用率	{}
service.type	DBPlusEngine-Operator-Cluster 对外暴露方式	ClusterIP
service.port	DBPlusEngine-Operator-Cluster 对外暴露端口	3307
startPort	DBPlusEngine-Operator-Cluster 启动端口	3307
imagePullSecrets	DBPlusEngine-Operator-Cluster 私有镜像仓库密钥	[]
mysqlDriver.version	DBPlusEngine-Operator-Cluster mysql 驱动版本，如果为空，将不下载驱动	""
GN.mode	DBPlusEngine-Operator-Cluster 治理中心模式，支持 sidecar / zookeeper	zookeeper
GN.SidecarRegistry	DBPlusEngine-Operator-Cluster sidecar 模式镜像 host	<镜像仓库 host>
GN.SidecarRepository	DBPlusEngine-Operator-Cluster sidecar 模式镜像仓库名	sphereex/dbplusengine-sidecar
GN.SidecarTag	DBPlusEngine-Operator-Cluster sidecar 模式镜像 tag	0.2.0
GN.sidecarServerAddr	DBPlusEngine-Operator-Cluster sidecar 模式镜像服务端地址	服务端地址
withAgent	DBPlusEngine-Operator-Cluster 是否启动 agent 参数	false

计算节点 DBPlusEngine-Proxy Server 配置权限配置项

Name	Description	Value
serverConfig.authority.privilege.type	存储节点数据授权的权限提供者类型，缺省值为 ALL_PERMITTED	ALL_PERMITTED
serverConfig.authority.users[0].password	用于登录计算节点的密码	root
serverConfig.authority.users[0].user	用于登录计算节点的用户名，授权主机。格式: @hostname 为% 或空字符串表示不限制授权主机	root@%

计算节点 DBPlusEngine-Proxy Server 配置模式配置项

Name	Description	Value
serverConfig.mode.type	运行模式类型。现阶段仅支持 Cluster 模式	Cluster
serverConfig.mode.repository.props.namespace	注册中心命名空间	governance_ds
serverConfig.mode.repository.props.server-lists	注册中心连接地址	{{ printf "%s-zookeeper.%s:2181" .Release.Name .Release.Namespace }}
serverConfig.mode.repository.props.maxRetries	客户端连接最大重试次数	3
serverConfig.mode.repository.props.operationTimeoutMilliseconds	客户端操作超时的毫秒数	5000
serverConfig.mode.repository.props.retryIntervalMilliseconds	重试间隔毫秒数	500
serverConfig.mode.repository.props.timeToLiveSeconds	临时数据失效的秒数	60
serverConfig.mode.repository.type	持久化仓库类型。现阶段仅支持 ZooKeeper	ZooKeeper

说明：支持设置 CGROUP_MEM_OPTS 环境变量：用于在容器环境中设置相关内存参数，脚本中的默认值为：

-XX:InitialRAMPercentage=80.0 -XX:MaxRAMPercentage=80.0 -XX:MinRAMPercentage=80.0

治理节点 ZooKeeper 配置项

配置项	描述	值
zookeeper.enabled	用来切换是否使用 ZooKeeper 的 chart	true
zookeeper.replicaCount	ZooKeeper 节点数量	1
zookeeper.persistence.enabled	标识 ZooKeeper 是否使用持久卷申领 (PersistentVolumeClaim) 用来申请持久卷 (PersistentVolume)	false
zookeeper.persistence.storageClass	持久卷 (PersistentVolume) 的存储类 (StorageClass)	""
zookeeper.persistence.accessModes	持久卷 (PersistentVolume) 的访问模式	["ReadWriteOnce"]
zookeeper.persistence.size	持久卷 (PersistentVolume) 大小	8Gi

配置示例

dbplusengine-operator/values.yaml

```
## @section DBPlusEngine-Operator-Cluster operator parameters
## @param replicaCount operator 副本数
##
replicaCount: 2
image:
  ## @param image.repository operator 镜像名
  ##
  repository: "sphere-ex/dbplusengine-operator"
  ## @param image.pullPolicy 镜像拉取策略
  ##
  pullPolicy: IfNotPresent
  # Overrides the image tag whose default is the chart appVersion.
  ## @param image.tag 镜像 tag
  ##
  tag: "0.1.0"
  ## @param imagePullSecrets 私有仓库镜像拉取密钥
  ## e.g:
  ## imagePullSecrets:
  ##   - name: mysecret
  ##
imagePullSecrets: []
  ## @param resources operator 需要的资源
  ## e.g:
  ## resources:
  ##   limits:
  ##     cpu: 2
  ##   limits:
  ##     cpu: 2
  ##
resources: {}
  ## @param webhook.port operator webhook 启动端口
  ##
webhook:
  port: 9443
  ## @param health.healthProbePort operator 健康检查端口
  ##
health:
  healthProbePort: 8081
```

dbplusengine-proxy/values.yaml

```
#
# Copyright © 2022, Beijing Sifei Software Technology Co., LTD.
# All Rights Reserved.
# Unauthorized copying of this file, via any medium is strictly prohibited.
# Proprietary and confidential
#

# @section DBPlusEngine-Proxy cluster parameters
## @param replicaCount DBPlusEngine-Proxy 集群启动副本数，注意：在开启 automaticScaling 后，这个参数将不再生效
##
replicaCount: "2"

#
##
##
GN:
  mode: zookeeper
  SidecarRegistry: uhub.service.ucloud.cn
  SidecarRepository: sphere-ex/dbplusengine-sidecar
```

```

SidecarTag: "0.2.0"
sidecarServerAddr: "so-dbplusengine-operator.ss"
## @param automaticScaling.enable DBPlusEngine-Proxy 集群是否开启自动扩缩容
## @param automaticScaling.scaleUpWindows DBPlusEngine-Proxy 自动扩容稳定窗口
## @param automaticScaling.scaleDownWindows DBPlusEngine-Proxy 自动缩容稳定窗口
## @param automaticScaling.target DBPlusEngine-Proxy 自动扩缩容阈值，数值为百分比，注意：现阶段暂时只支持 cpu 为指标进行扩缩容
## @param automaticScaling.maxInstance DBPlusEngine-Proxy 最大扩容副本数
## @param automaticScaling.minInstance DBPlusEngine-Proxy 最小启动副本数，缩容不会小于这个副本数
##
automaticScaling:
  enable: false
  scaleUpWindows: 30
  scaleDownWindows: 30
  target: 20
  maxInstance: 4
  minInstance: 1
## @param image.registry DBPlusEngine-Proxy 镜像 host
## @param image.repository DBPlusEngine-Proxy 镜像仓库名
## @param image.tag DBPlusEngine-Proxy 镜像 tag
##
image:
  registry: uhub.service.ucloud.cn
  repository: sphere-ex/dbplusengine-proxy
  tag: "1.2.0"
withAgent: false
## @param resources DBPlusEngine-Proxy 启动需求资源，在开启 automaticScaling 后，以 request 的资源乘以 target 的百分比为触发扩缩容动作的实际使用率
## e.g:
## resources:
##   limits:
##     cpu: 2
##   requests:
##     cpu: 2
##
resources:
  limits:
    cpu: '2'
  requests:
    cpu: '2'
## @param service.type DBPlusEngine-Proxy 对外暴露方式
## @param service.port DBPlusEngine-Proxy 对外暴露端口
##
service:
  type: ClusterIP
  port: 3307
## @param startPort DBPlusEngine-Proxy 启动端口
##
startPort: 3307
## @param imagePullSecrets DBPlusEngine-Proxy 私有镜像仓库密钥
## e.g:
## imagePullSecrets:
##   - name: mysecret
##
imagePullSecrets:
  username: ""
  password: ""
## @param mySQLDriver.version DBPlusEngine-Proxy mysql 驱动版本，如果为空，将不下载驱动
##
mySQLDriver:
  version: "5.1.43"
## @section DBPlusEngine-Proxy ServerConfiguration parameters
## NOTE: If you use the sub-charts to deploy Zookeeper, the server-lists field must be "{{ printf \"%s-zookeeper.%s:2181\" . Release.Name .Release.Namespace }}"

```

```

## otherwise please fill in the correct zookeeper address
## The server.yaml is auto-generated based on this parameter.
## If it is empty, the server.yaml is also empty.
## ref: https://shardingsphere.apache.org/document/current/en/user-manual/shardingsphere-jdbc/yaml-config/mode/
## ref: https://shardingsphere.apache.org/document/current/en/user-manual/shardingsphere-jdbc/builtin-algorithm/metadata-repository/
##
serverConfig:
  ## @section Compute-Node DBPlusEngine-Proxy ServerConfiguration authority parameters
  ## NOTE: It is used to set up initial user to login compute node, and authority data of storage node.
  ## @param serverConfig.authority.privilege.type authority provider for storage node, the default value is ALL_PERMITTED
  ## @param serverConfig.authority.users[0].password Password for compute node.
  ## @param serverConfig.authority.users[0].user Username,authorized host for compute node. Format: <username>@<hostname> hostname is % or empty string means do not care about authorized host
  ##
  authority:
    privilege:
      type: ALL_PERMITTED
    users:
      - password: root
        user: root@%
  ## @section Compute-Node DBPlusEngine-Proxy ServerConfiguration mode Configuration parameters
  ## @param serverConfig.mode.type Type of mode configuration. Now only support Cluster mode
  ## @param serverConfig.mode.repository.props.namespace Namespace of registry center
  ## @param serverConfig.mode.repository.props.server-lists Server lists of registry center
  ## @param serverConfig.mode.repository.props.maxRetries Max retries of client connection
  ## @param serverConfig.mode.repository.props.operationTimeoutMilliseconds Milliseconds of operation timeout
  ## @param serverConfig.mode.repository.props.retryIntervalMilliseconds Milliseconds of retry interval
  ## @param serverConfig.mode.repository.props.timeToLiveSeconds Seconds of ephemeral data live
  ## @param serverConfig.mode.repository.type Type of persist repository. Now only support ZooKeeper

  ##
  # mode:
  # repository:
  #   props:
  #     namespace: matenamespace
  #     server-lists: "127.0.0.1:21506"
  #   type: SphereEx:MATE
  # type: Cluster
mode:
  repository:
    props:
      maxRetries: 3
      namespace: governance_ds
      operationTimeoutMilliseconds: 5000
      retryIntervalMilliseconds: 500
      server-lists: "{{ printf \"%s-zookeeper.%%s:2181\" .Release.Name .Release.Namespace }}"
      timeToLiveSeconds: 600
      type: ZooKeeper
      type: Cluster
  ## @section ZooKeeper chart parameters

  ## ZooKeeper chart configuration
  ## https://github.com/bitnami/charts/blob/master/bitnami/zookeeper/values.yaml
  ##
  zookeeper:
    ## @param zookeeper.enabled Switch to enable or disable the ZooKeeper helm chart
    ##
    enabled: true
    ## @param zookeeper.replicaCount Number of ZooKeeper nodes
    ##
    replicaCount: 1
    ## ZooKeeper Persistence parameters
    ## ref: https://kubernetes.io/docs/user-guide/persistent-volumes/

```

```
## @param zookeeper.persistence.enabled Enable persistence on ZooKeeper using PVC(s)
## @param zookeeper.persistence.storageClass Persistent Volume storage class
## @param zookeeper.persistence.accessModes Persistent Volume access modes
## @param zookeeper.persistence.size Persistent Volume size
##
persistence:
  enabled: false
  storageClass: ""
  accessModes:
    - ReadWriteOnce
  size: 8Gi
```

清理

```
helm uninstall dbplusengine-proxy -n dbplusengine

helm uninstall dbplusengine-operator -n dbplusengine-operator

kubectl delete crd clusters.dbplusengine.sphere-ex.com \
proxyconfigs.dbplusengine.sphere-ex.com \
plocks.dbplusengine.sphere-ex.com \
pmetadata.dbplusengine.sphere-ex.com \
pnodes.dbplusengine.sphere-ex.com \
ppipelines.dbplusengine.sphere-ex.com \
psys.dbplusengine.sphere-ex.com \
pworkids.dbplusengine.sphere-ex.com
```

9.4 版本管理

如下为商业套件支持的所有版本。

分类	产品/组件	名称	版本
商业套件	商业套件	SphereEx-DBPlusSuite	1.1.0
集群生态工具	管控平台	SphereEx-Console	1.1.0
	后台管控平台	SphereEx-Boot	1.1.0
集群组件	计算节点	SphereEx-DBPlusEngine	1.1.0, 1.1.2, 1.2.0
	存储节点	MySQL	5.7+
	存储节点	Postgre	9.6+
	治理中心	ZooKeeper	3.7.0+
	监控中心	Prometheus	2.32.0
	监控中心	zookeeper_exporter	
	监控中心	node_exporter	1.3.1
	监控中心	mysql_export	0.14.0
	日志中心	Elasticsearch	7.5.2
	日志中心	Logstash	7.5.2
	日志中心	Filebeat	7.5.2

9.4.1 升降级

集群生态工具的升降级

SphereEx-DBPlusEngine 升级

Todo

SphereEx-Console 升降级

目前通过重新部署的方式升降级。

说明：目前 Console 可管理的 Engine 的版本是固定的，如需升降级需要参考版本说明部分。

SphereEx-Boot 升降级

目前通过重新部署的方式升降级。

集群组件的升降级

升降服务目前未集成到工具中，提供升降级人工服务[联系我们](#)。

集群升降级

Todo

当前在我们产品能力中没有此功能，未来需考虑集成到 Boot、Console 中。

现阶段可通过人工方案解决，包含以下内容：

1. Engine 升级步骤
2. Engine 回退步骤
3. Engine 升级影响
4. Engine 升级后兼容性评估

9.5 权限管理

9.5.1 集群生态工具的权限

SphereEx-Boot 的权限

SphereEx-Boot 无权限控制。

SphereEx-Console 的权限

角色权限说明

角色	权限	作用域	操作	说明
超级管理员	全部权限	系统管理员、用户、集群	Console 用户管理	用户名称: admin, console 内置, 登录后可修改密码
系统管理员	全部权限	用户、集群	Console 用户管理	由超级管理员创建
普通用户	指定集群的权限	集群	Console 用户管理	由超级管理员或系统管理员创建

SphereEx-Console 访问组件的权限

组件/资源	说明
主机	ssh 登录权限, 安装目录有
数据库	在 SphereEx-Console 中使用监控中心监控 db 需要在 db 所在的主机安装监控插件, 监控插件访问数据库需要额外分配用户。创建用户参考: create user '[mysql_monitor_user_name]' @ '[监控插件部署的 IP]' identified by '[password]' [with max_user_connections N];grant process,replication client,select on . to '[mysql_monitor_user_name]' @ '[监控插件部署的 IP]';flush privileges;
治理中心	无
监控中心	无
日志中心	无

功能列表

功能	说明
新建用户	在 SphereEx-Console 平台中添加用户
修改密码	修改集群信息
权限管理	管理集群参数
删除	删除 SphereEx-Console 平台中的用户

■ 新建用户

适用场景

创建 SphereEx-Console 管理用户。

注意事项

拥有超级管理员或系统管理员权限。

操作步骤

- 1. 登录 SphereEx-Console。
- 2. 点击首页右上角头像-> 用户管理进入用户列表。
- 3. 点击右上角 添加用户, 进入添加用户页面。

4. 填写信息，具体参数见下表。

字段	数据来源	选 填/必填	数据存储	长度限制	说明
用户	用户输入	必填	console m ysql 存储	32	用户列表校验唯一性
密码	用户输入	必填	console m ysql 存储	6-16 位	数字字母下划线，
确 认 密 码	用户输入	必填		console m ysql 存储	
角色	下拉选择	必填	console m ysql 存储		系统管理员
集 群 权 限	下拉选择	选填	console m ysql 存储		【角色】为【普通管理员】时出现
可 选 集 群	无权限的集群列表	选填	console m ysql 存储		
已 选 集 群	已有权限的集群列表	选填	console m ysql 存储		

5. 点击 **确定**，完成用户添加。

■ 修改密码

适用场景

修改 SphereEx-Console 的用户密码。

注意事项

如下为不同角色的用户可修改密码的范围。

角色	可修改的范围
超级管理员	自身、系统管理员、普通管理员
系统管理员	自身、普通管理员
普通管理员	自身

操作步骤

1. 登录 SphereEx-Console。
2. 点击首页右上角头像-> **用户管理**进入用户列表。
3. 点击操作列 **修改密码**按钮，进入修改密码页面。
4. 填写信息，具体参数见下表。

字段	数据来源	选填/必填	数据存储	长度限制	元件	说明
用户	自动填充	必填	console my sql 存储	32	文本框	不可编辑
新密码	用户输入	必填	console my sql 存储	6-16 位	文本框	数字字母下划线，
确认密码	用户输入	必填	console my sql 存储		文本框	与新密码一致

5. 点击 **确定**，完成用户添加。

■ 权限管理

适用场景

修改 SphereEx-Console 的用户密码

注意事项

管理普通用户的集群权限，角色不能修改，用户权限被修改后，需要重新登录才能刷新权限。

操作步骤

1. 登录 SphereEx-Console。

2. 点击首页右上角头像-> 用户管理进入用户列表。
3. 点击操作列 权限管理按钮，进入权限管理页面。
4. 填写信息，具体参数见下表。

字段	数据来源	选填/必填	长度限制	说明
用户	用户输入	必填	32	用户列表校验唯一性
集群权限	下拉选择	选填		【角色】为【普通管理员】时出现
可选集群	无权限的集群列表	选填		
已选集群	已有权限的集群列表	选填		

5. 点击 确定，完成权限调整。
- 删除

适用场景

删除系统管理员或普通管理员。

注意事项

删除用户，如下为不同角色的用户可删除用户的范围：

角色	可删除的用户范围
超级管理员	系统管理员、普通管理员
系统管理员	普通管理员

操作步骤

1. 登录 SphereEx-Console。
2. 点击首页右上角头像-> 用户管理进入用户列表。
3. 点击操作列 删除按钮。
4. 提示框中点击确定，完成用户的删除。

9.5.2 集群组件的权限

访问组件的权限：

组件名称	组件本身的权限	集群访问使用组件的权限	SphereEx-Console 访问组件的权限	监控插件访问组件
主机	无	无	ssh 登录权限, 安装目有读写权限	ssh 登录权限, 安装目有读写权限
数据库	无	依据用户使用底层 db 的权限	依据用户使用底层 db 的权限	创建用户参考: create user '[mysql_monitor_user_name]' @ '[监控插件部署的 IP]' identified by '[password]' [with max_user_connections N];
治理中心	无	无	无	无
监控中心	无	无	无	无
日志中心	无	无	无	无

9.5.3 集群权限

集群权限管理，即计算节点的权限管理。

角色	职能	作用域	操作	说明
超级用户 (root)	最高权限	集群	D ist-SQL	必须配置，不可删除，可修改口令。修改后写入治理中心，不回写本地文件-第一次使用本地文件启动，明文有风险
普通用户/角色	对象级权限	逻辑库	D ist-SQL	在逻辑库中限制对象的增删改查权限

权限分级

DBPlusEngine-Proxy 权限的分级与标准数据库权限类似，同样拥有全局权限、库级对象权限、表级对象权限、列级对象权限。

DBPlusEngine-Proxy 权限授权项由操作 + 操作对象组成，其组合方式如下。

授权项	SELECT	INSERT	UPDATE	DELETE	CREATE	D R O P	A LT ER	I ND EX	CREAT E_USER	S UP ER
全局权限	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
对象权限/库	✓	✓	✓	✓	✓	✓	✓	✓	/	/
对象权限/表	✓	✓	✓	✓	✓	✓	✓	✓	/	/
对象权限/列	✓	✓	✓	/	/	/	/	/	/	/

授权项	CREATE	ALTER	DROP	SHOW
STORAGE UNIT	✓	✓	✓	✓
SHARDING	✓	✓	✓	✓
READWRITE_SPLITTING	✓	✓	✓	✓
ENCRYPT	✓	✓	✓	✓
DB_DISCOVERY	✓	✓	✓	✓
SHADOW	✓	✓	✓	✓
SINGLE_TABLE	✓	✓	✓	✓

■ 全局权限

全局权限，是指用户获得的授权不区分目标对象，用户可对任意的逻辑库、逻辑表执行对应操作。

例如，以下指令将全局的 INSERT、SELECT、UPDATE 和 DELETE 权限赋予用户 'sharding' @ '%'，则该用户能够对任何逻辑库中的表执行 DML 操作。

-- 以下两个语句等效

```
GRANT DIST INSERT,SELECT,UPDATE,DELETE TO 'sharding'@'%';
GRANT DIST INSERT,SELECT,UPDATE,DELETE TO sharding;
```

例如，以下指令将全局的 SHOW SHARDING 权限赋予用户 'sharding' @ '%'，则该用户能够对任何逻辑库中的分片规则执行 SHOW 操作。

```
GRANT DIST SHOW SHARDING TO 'sharding'@'%';
```

另外，因为 DistSQL 授权项较多，对于一次授予多个授权项时，可以使用语法类型作为授权项。例如，以下指令将所有操作对象全局的 SHOW 权限赋予用户 'sharding' @ '%'，则该用户能够对任何逻辑库中的任何操作对象执行 SHOW 操作。

```
GRANT DIST RQL SHARDING TO 'sharding'@'%';
```

需要注意的是，全局权限包含两项特殊权限：CREATE_USER 和 SUPER。

其中，获得 CREATE_USER 授权的用户，可执行以下操作：

操作	说明
CREATE USER	创建用户
ALTER USER	修改用户
DROP USER	删除用户
CREATE ROLE	创建角色
DROP ROLE	删除角色
REVOKE ALL PRIVILEGES	撤销用户或角色的全部授权

SUPER 代表的是数据库系统的最高权限，默认情况下，ShardingSphere 启动前配置的初始用户拥有 SUPER 授权。

■ 对象权限

对象权限，指授予的权限受作用范围限制，用户不可在授权范围之外执行相应操作。

对象权限的作用范围可以是所有逻辑库，也可以指定单个或多个逻辑库及库中的表和列。

例如，以下指令将逻辑库 sharding_db 中 t_order 表的所有权限授予用户 'sharding' @ '%'，此后该用户可以对 sharding_db.t_order 表执行操作。但若无额外授权，该用户不能操作 sharding_db 中的其它表。

```
GRANT DIST ALL ON sharding_db.t_order TO sharding;
```

例如，以下指令将逻辑库 sharding_db 中分片规则 t_order 的创建和修改权限授予用户 'sharding' @ '%'，此后该用户可以对 sharding_db.t_order 规则执行创建和修改操作。但若无额外授权，该用户不能操作 sharding_db 中的其它规则。

```
GRANT DIST CREATE SHARDING, ALTER SHARDING ON sharding_db.t_order TO 'sharding'@'%';
```

DistSQL 清单

参考 [DistSQL 用户管理](#)

9.6 参数管理

9.6.1 集群生态工具参数管理

SphereEx-Boot 参数管理

- 参数查看安装完成 SphereEx-Boot 使用 help 查看参数和使用方法。

```
spex help
```

- 参数修改 SphereEx-Boot 不支持修改参数

SphereEx-Console 参数管理

- 参数查看

SphereEx-Console 通过配置查看

- 参数修改

SphereEx-Console 通过配置修改

9.6.2 集群组件参数管理

- 参数查看

无法通过 SphereEx-Console 各个组件的参数。

- 参数修改

用户登录组件的部属主机，通过各个组件的文件配置自行修改。

9.6.3 集群参数管理

通过 SphereEx-Console 修改集群参数

通过 SphereEx-Console 调整运行日志参数请参考[集群管理参数管理](#)

参数详解请参考 [集群参数列表](#)

通过客户端修改集群参数

- 查看参数：

```
SHOW VARIABLES LIKE 'general_%';  
SHOW VARIABLES;
```

- 修改参数：

```
set general_query_log = true;
```

9.7 日志管理

9.7.1 SphereEx-DBPlusSuite 中日志分为三部分：

- 集群生态工具：

SphereEx-Console 日志：SphereEx-Console 的运行日志；SphereEx-Boot 日志：无

- 集群组件：

集群组件产生的日志，由用户自行管理

- 集群日志：

即计算节点的日志

9.7.2 集群生态工具日志

- SphereEx-Boot 无日志文件，所以不展开介绍
- SphereEx-Console 日志文件位置：logs/
- SphereEx-Console 日志文件位置修改方式：conf/logback.xml
- SphereEx-Console 日志文件内容解读

9.7.3 集群组件日志

集群组件产生的日志，由用户自行管理，本文档不做说明。

9.7.4 集群日志

即计算节点的日志分为三种：运行日志、审计日志、慢日志。

日志名称	日志文件位置与名称	日志位置修改方式	参数	说明
运行日志	logs/stdout.log	conf/log-back.xml	无	默认开启，不能在线调整关闭；修改日志位置与名称需要重启生效
审计日志(全量日志)	logs/stdout.log	conf/log-back.xml	general_query_log: true 表示开启全量日志；false 表示关闭全量日志；默认关闭	参数可以在线调整修改日志位置与名称需要重启生效审计日志与运行日志的日志相同
慢日志	logs/slow_query.log 日志中心	conf/log-back.xml	“slow_query_log: true 表示开启慢日志；false 表示关闭慢日志；默认关闭 long_query_time: 慢日志阈值，单位为 ms 默认值为 10”	参数可以在线调整修改日志位置与名称需要重启生效

管理日志

参考集群管理、参数管理

集群日志清理

日志名称	存储位置	回卷策略
运行日志	计算节点的日志目录 logs/	通过 conf/logback.xml 配置，调整日志的回卷策略
审计日志（全量日志）	计算节点的日志目录 logs/	通过../ conf/logback.xml 配置，调整日志的回卷策略
慢日志	计算节点的日志目录 logs/	通过../ conf/logback.xml 配置，调整日志的回卷策略
慢日志	日志中心	通过配置 Elasticsearch 完成

集群日志格式解读

三类集群日志的格式解读，此部分内容后续整理补充

9.8 License 管理

范围：集群 license

9.8.1 License 在 DBPlusEngine-Proxy 中的使用

背景信息

License 功能	说明
限制时长	限制 DBPlusEngine 的使用时长
限制存储节点数量	限制 DBPlusEngine 集群创建存储节点的数量
限制计算节点 (Proxy)	限制单个集群中创建创建计算节点（DBPlusEngine-Proxy）的数量
限制计算节点 (Driver)	限制单个集群中创建创建计算节点（DBPlusEngine-Driver）的数量
限制版本	限制集群中 DBPlusEngine 的版本
硬件设备的 MAC 地址	限制集群中计算节点所在的硬件设备的 MAC 地址

注意事项

License 默认的存储目录为：config。

操作步骤

- 执行如下命令查看 license。

```
SHOW LICENSE INFO;
```

- 使用注册码注册 license。

```
REGISTER LICENSE 'license-key';
```

- 使用 license 文件注册 license。

```
REGISTER LICENSE '直接路径/license-file';
```

- 使用注册码文件预览 license。

```
SHOW LICENSE INFO 'license-key';
```

- 使用 license 文件预览 license。

```
SHOW LICENSE INFO '直接路径/license-file';
```

后续处理

如果出现如下错误请获取最新版 license 并更新 license 至最新状态。

- 无 license 提示

```
FATAL: License not registered
```

- license 不完整提示

```
FATAL: Incomplete license
```

- license 过期提示

```
FATAL: License is expired
```

- 非法 license 提示

```
ERROR: The xxx exceeds the limit of the license
```

9.8.2 无 License 或 License 超限后的表现

- 可以启动集群无法执行 RDL、RUL 和 RAL，可以执行少量 RQL
- 全局放行的 DistSQL
 1. SHOW STORAGE UNITS
 2. SHOW *
 3. COUNT * RULES
 4. EXPORT DATABASE CONFIGURATION
 5. CONVERT YAML CONFIGURATION
 6. REGISTER LICENSE
 7. 事务相关的标准语句
 - 实例数量超限可执行
 1. SET INSTANCE STATUS
 - DB 的 SQL 语句不能执行

9.8.3 License 过期提醒功能

当 License 距离过期天数 ≤ 30 天时在集群的计算节点运行日志中会输出 WARN 级别的提示。

提示内容为：

```
The license of DBPlusEngine will expire in xx day(s)
```

9.9 监控与告警

说明：1. 监控功能只能通过 SphereEx-Console 完成 2. 目前无告警功能

9.9.1 集群生态工具的监控

目前未对 SphereEx-Console 和 SphereEx-Boot 做监控。

9.9.2 集群组件的监控

主机监控

■ 监控指标

分类	子分类	指标
资源总览	资源总览	系统运行时长
		CPU 核数
		总内存
		总 CPU 使用率
		总内存使用率
		各空间使用
性能数据	性能数据	CPU 使用率
		内存统计
		网络带宽（每 秒）
		系统负载（1、5、15 分钟）
		磁盘读写吞吐量（每秒）
		磁盘读写速率 I OPS
		IO 操作占比（每秒）
		IO 读写耗时（每次）

■ 查看监控

适用场景

查看主机监控

注意事项

- 监控中心正常
- 主机已经配置监控

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **监控**-> **主机**进入监控列表。
3. 点击操作列中 **监控**按钮查看监控指标。

■ 配置监控

适用场景

配置主机监控

注意事项

- 监控中心正常
- 主机已经被注册，对监控插件安装目录有读写权限

操作步骤

1. 登录 SphereEx-Console。
2. 点击 资源-> 主机进入主机列表。
3. 点击操作列中的 配置监控进入监控配置页面。
4. 配置监控信息如下。

字段	数据来源	选填/必填	说明
主机 IP	上一页面		
监控中心	用户选择，监控中心列表	必填	已经安装的监控中心
监控插件端口	默认填充用户可以编辑	必填	默认填充监控插件对应的默认端口，可编辑
监控插件安装目录	默认填充，用户可以编辑		默认填充监控插件对应的默认安装目录，可编辑

5. 点击 安装完成监控插件的安装和监控中心配置。

数据库监控

■ 监控指标

分类	子分类	指标
概览	概览	实例存活
		文件打开数
		从库只读
		主从延迟
		从库 SQL 线程
		从库 IO 线程
		是否开启慢查询
		慢查询阈值
性能指标	性能指标	QPS (指定时间内的 Queries 值/ 时间)
		(com_insert + com_delete + com_update + com_select) 数量/时间
		入口流量
		出口流量
		慢查询数量
		当前连接数
		缓冲池利用率

■ 查看监控

适用场景

查看数据库实例监控

注意事项

- 监控中心正常
- 数据库已经配置监控

操作步骤

1. 登录 SphereEx-Console。

2. 点击 **监控->数据库** 进入监控列表。
3. 点击操作列中 **监控按钮** 查看监控指标。

■ 配置监控

适用场景

配置数据库监控

注意事项

- 监控中心正常
- 数据库主机已经被注册，对监控插件安装目录有读写权限

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **资源->数据库** 进入主机列表。
3. 点击操作列中的 **配置监控** 进入监控配置页面。
4. 配置监控信息如下。

字段	数据来源	选填/必填	说明
监控中心	用户选择，监控中心列表	必填	已经安装的监控中心
监控插件端口	默认填充	用户可以编辑	必填
监控插件安装目录	默认填充	用户可以编辑	默认填充监控插件对应的默认安装目录，可编辑
数据库监控用户	用户输入	必填	
数据库监控口令	用户输入	必填	密码保护

5. 点击 **安装完成** 监控插件的安装和监控中心配置。

治理中心监控

■ 监控指标

分类	子分类	指标
概览	概览	ZK 集群节点状态
		节点角色
		Follower 数
性能指标	性能指标	平均响应延迟
		最大响应延迟
		最小响应延迟
		收包数
		发包数
		活跃连接数
		堆积请求数
		主从状态
		znode 数
		watch 数
		临时节点数
		近似数据总大小
		打开文件描述符数
		最大文件描述符数
		阻塞中的 sync 操作

■ 查看监控

适用场景

查看治理中心实例监控

注意事项

- 监控中心正常
- 治理中心已经配置监控

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **监控**-> **治理中心** 进入治理中心列表。
3. 点击操作列中 **节点监控** 进入监控列表。
4. 点击操作列中 **监控按钮** 查看监控指标。

■ 配置监控

适用场景

配置治理中心监控

注意事项

- 监控中心正常
- 治理中心主机已经被注册，对监控插件安装目录有读写权限

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **资源**-> **治理中心** 进入主机列表。
3. 点击操作列中的 **配置监控** 进入监控配置页面。
4. 配置监控信息如下。

字段	数据来源	选填/必填	说明
监控中心	用户选择	必填	一个集群只允许有一个监控中心；
治理中心节点 IP	自动填充，前一页面信息	必填	不可编辑
治理中心节点端口	自动填充，前一页面信息	必填	不可编辑
监控插件端口	默认填充用户可以编辑	必填	默认填充监控插件对应的默认端口，可编辑
监控插件安装目录	默认填充用户可以编辑		默认填充监控插件对应的默认安装目录，可编辑

5. 点击 **安装完成** 监控插件的安装和监控中心配置。

监控中心监控

■ 监控指标

分类	子分类	指标
概览	概览	版本
		监控的实例数
		线程数
		上次成功重新加载配置的时间
		上次配置重新加载是否成功
		chunk 总数
		创建的 chunks 数量
		移除的 chunks 数量
		samples 总数

■ 查看监控

适用场景

查看监控中心自身监控

注意事项

监控中心正常

操作步骤

1. 登录 SphereEx-Console。
2. 点击 监控-> 监控中心进入监控列表。
3. 点击操作列中 监控按钮查看监控指标。

■ 配置监控

安装监控中心时默认打开监控中心的自身监控，无需单独配置。

日志中心监控

Todo

此功能在 Console 1.2 版本中实现。

集群监控

■ 监控指标

分类	子分类	指标
概览	集群概览	组件节点数
		存储节点数量
		计算节点数量
		治理中心节点数量
元数据	元数据	逻辑库数量
		用户数量
		表数量（逻辑表 + 单表）
		分片表数量
		广播表数量
		表组数量
		单表数量
		加密表数量
		所有插件的数量（不含单表插件）
性能数据	连接详情	路由数量（瞬时值、变化值、变化率）
		执行数量（瞬时值、变化值、变化率）
		解析数量（瞬时值、变化值、变化率）
		请求数量（瞬时值、变化值、变化率）
		连接数（瞬时值）
	性能详情	QPS
		TPS
		请求字节数
		响应字节数
		响应时间
		事务总数
		提交事务数
		回滚事务数
		事务回滚率
		连接持续时长（分析）
		请求持续时长（分析）
解析引擎	DML sql	Insert 总量
		Delete
		Update 总量
		Select 总量

continues on next page

Table 1 – continued from previous page

分类	子分类	指标
	DDL sql	sql 统计 (InsertDeleteUpdateSelect)
		DDL 总量
		DCL 总量
		DAL 总量
		TCL 总量
		sql 统计 (DDL DCL DAL TCL)
	DistSQL	RQL 总量
		RDL 总量
		RAL 总量
		DistSQL 统计 (RQL RDL RAL)
		解析时长
路由引擎	路由引擎	数据源路由 表路由分析 TOP10
线程情况	线程情况	当前线程数量 守护线程数量 线程峰值数量 线程启动总数 线程死锁数 JVM 线程状态数据
错误统计	错误统计	错误总数 错误数量

■ 查看监控

适用场景

查看集群监控

注意事项

- 监控中心正常
- 集群已经配置监控

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **监控**-> **集群**进入监控列表。
3. 点击操作列中 **监控**按钮查看监控指标。

■ 配置监控

适用场景

配置集群监控，实际上是配置计算节点的监控。

注意事项

- 监控中心正常。
- 集群计算节点主机已经被注册，对监控插件安装目录有读写权限。

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **集群管理**-> **集群**进入主机列表。
3. 点击操作列中的 **配置监控**进入监控配置页面。
4. 配置监控信息如下。

字段	数据来源	选 填/必填	说明
计算节点 IP	前一页面输入	必填	不可编辑
监控插件端口	用户输入	必填	自建集群，监控插件端口自动填充，不允许修改，注册集群，用户可自己填充

5. 点击 **添加监控配置**完成监控中心配置。

计算节点监控

■ 监控指标

分类	子分类	指标
概览	概要	JDK 版本信息
		启动时间
		计算节点版本
		运行状态
		运行时长
		已使用内存大小
		已使用缓存池大小
性能数据	性能数据	请求总量
		路由总量
		执行总量
		解析总量
		请求数量
		解析数量
		路由数量
		执行数量
		QPS
		TPS
元数据	元数据	事务回滚率
		逻辑库数量
		用户数量
		表数量（逻辑表 + 单表）
解析引擎	DML sql	分片表数量
		Insert 总量
		Delete 总量
		Update 总量
		Select 总量
	DDL sql	sql 统计（InsertDeleteUpdateSelect）
		DDL 总量
		DCL 总量
		DAL 总量
		TCL 总量
	DistSQL	sql 统计（DDL DCL DAL TCL）
		RQL 总量
		RDL 总量
		RAL 总量
		DistSQL 统计（RQL RDL RAL）
路由引擎	路由引擎	解析时长
		数据源路由
线程情况	线程情况	表路由分析 TOP10
		当前线程数量
		守护线程数量
		线程峰值数量

continues on next page

Table 2 – continued from previous page

分类	子分类	指标
错误统计	错误统计	线程启动总数
		线程死锁数
		JVM 线程状态数据
		错误总数
		错误数量

■ 查看监控

适用场景

查看集群监控

注意事项

- 监控中心正常
- 集群已经配置监控

操作步骤

1. 登录 SphereEx-Console。
2. 点击 **监控**-> **集群**进入集群列表。
3. 点击操作列中 **节点监控**进入计算节点列表。
4. 点击操作列中 **监控**查看监控指标。

■ 配置监控

通过配置集群监控完成。

存储节点监控

目前不支持单独监控存储节点。

9.9.3 Agent 管理

Agent 介绍

背景

如何观测集群的运行状态，使运维人员可以快速掌握当前系统现状，并进行进一步的维护工作，是分布式系统的全新挑战。登录到具体服务器的点对点运维方式，无法适用于面向大量分布式服务器的场景。通过对可系统观察性数据的遥测是分布式系统推荐的运维方式。Tracing（链路跟踪）、Metrics（指标监控）和 Logging（日志）是系统运行状况的可观察性数据重要的获取手段。APM（应用性能监控）是通过对系统可观察性数据进行采集、存储和分析，进行系统的性能监控与诊断，主要功能包括性能指标监控、调用链分析，应用拓扑图等。DBPlusEngine 并不负责如何采集、存储以及展示应用性能监控的相关数据，而是为应用监控系统提供必要的指标数据。换句话说，DBPlusEngine 仅负责产生具有价值的的数据，并通过标准协议或插件化的方式递交给相关系统。Tracing 用于获取 SQL 解析与 SQL 执行的链路跟踪信息。DBPlusEngine 默认提供了对 OpenTelemetry 的支持，可以导出 tracing 数据到 Jaeger、Zipkin。也支持用户通过插件化的方式开发自定义的 Tracing 组件。

■ 使用 OpenTelemetry

OpenTelemetry 在 2019 年由 OpenTracing 和 OpenCensus 合并而来。使用这种方式，只需要在 agent 配置文件中，根据 [OpenTelemetry SDK 自动配置说明](#)，填写合适的配置即可。

■ 使用 SkyWalking 的内置自动探针

DBPlusEngine 团队与 [Apache SkyWalking](#) 团队共同合作，在 SkyWalking 中实现了 DBPlusEngine 自动探针，可以将相关的应用性能数据自动发送到 SkyWalking 中。注意这种方式的自动探针不能与 DBPlusEngine 插件探针同时使用。

Metrics 则用于收集和展示整个集群的统计指标。DBPlusEngine 默认提供了对 Prometheus 的支持。

挑战

Tracing 和 Metrics 需要通过埋点来收集系统信息。大量的埋点使项目核心代码支离破碎，难于维护，且不易定制化统计指标。

目标

提供尽量多的性能和统计指标，并隔离核心代码和埋点代码，是 DBPlusEngine 可观察性模块的设计目标。

核心概念

■ 代理

基于字节码增强和插件化设计，以提供 tracing 和 metrics 埋点，以及日志输出功能。需要开启代理的插件功能后，才能将监控指标数据输出至第三方 APM 中展示。

■ APM

APM 是应用性能监控的缩写。着眼于分布式系统的性能诊断，其主要功能包括调用链展示，应用拓扑分析等。

■ Tracing

链路跟踪，通过探针收集调用链数据，并发送到第三方 APM 系统。

■ Metrics

系统统计指标，通过探针收集，并且写入到时序数据库，供第三方应用展示。

Agent 配置

Agent 配置 - 目录说明创建 Agent 目录，解压 Agent 二进制包到 agent 目录。

```
mkdir agent
tar -zxvf apache-shardingsphere-${latest.release.version}-shardingsphere-agent-bin.tar.gz -C agent
cd agent
tree
.
├── LICENSE
├── NOTICE
├── README.txt
├── conf
│   └── agent.yaml
├── plugins
│   ├── lib
│   │   ├── shardingsphere-agent-metrics-core-${latest.release.version}-SNAPSHOT.jar
│   │   └── shardingsphere-agent-plugin-core-${latest.release.version}-SNAPSHOT.jar
│   ├── logging
│   │   └── shardingsphere-agent-logging-file-${latest.release.version}-SNAPSHOT.jar
│   ├── metrics
│   │   └── shardingsphere-agent-metrics-prometheus-${latest.release.version}-SNAPSHOT.jar
│   └── tracing
│       ├── shardingsphere-agent-tracing-opentelemetry-${latest.release.version}-SNAPSHOT.jar
│       └── shardingsphere-agent-tracing-opentracing-${latest.release.version}-SNAPSHOT.jar
├── shardingsphere-agent-${latest.release.version}-SNAPSHOT.jar
├── template
│   ├── dbplusengine-driver-grafana-template.json
│   └── dbplusengine-proxy-grafana-template.json
```

■ 配置说明

agent.yaml 是配置文件，插件有 OpenTracing、OpenTelemetry、File、Prometheus。如果需要开启插件时，只需要开启应的插件即可。如下开启 logging 中的 File 插件配置。

```
plugins:
logging:
File:
props:
```

```
slow-query-log: true
long-query-time: 5000
general-query-log: true
# metrics:
# Prometheus:
#   host: "localhost"
#   port: 9090
#   props:
#     jvm-information-collector-enabled: "true"
# tracing:
# OpenTelemetry:
#   props:
#     otel.service.name: "shardingsphere"
#     otel.traces.exporter: "jaeger"
#     otel.exporter.otlp.traces.endpoint: "http://localhost:14250"
#     otel.traces.sampler: "always_on"
```

■ 参数说明

名称	说明	取值范围	默认值
slow-query-log	是否开启慢查询日志 true、false	TRUE	
long-query-time	慢查询阈值(毫秒) 正整数	5000	
general-query-log	全量查询日志 true、false	TRUE	
jvm-information-collector-enabled	是否开启 JVM 采集器 true、false	TRUE	
otel.service.name	链路跟踪的服务名称任意字符串	shardingsphere	
otel.traces.exporter	导出器 jaeger、zipkin	jaeger	
otel.exporter.otlp.traces.endpoint	数据发送地址实际接收数据地址	http://localhost:14250	
otel.traces.sampler	采样器 always_on	always_on	

OpenTelemetry 中的更多配置参数可以参考: [OpenTelemetry SDK 自动配置说明](#)

DBPlusEngine-Proxy 中使用 Agent

启动 Proxy 的同时启动 Agent

```
bin/start.sh -g
```

正常启动可以在对应的 DBPlusEngine-Proxy 日志查看到 plugin 的启动日志，访问 Proxy 后，可以通过配置的地址查看到 Metric 和 Tracing 的数据。

Agent Metrics

指标	描述	指标类型
build_info	构建版本信息	GAUGE
proxy_info	“运行信息，type 标签区分类型。boot-time 表示启动时间的时间戳，boot-duration 表示启动耗时（毫秒），uptime 表示截止已运行时间（毫秒）”	GAUGE
proxy_meta_data_info	元数据信息	GAUGE
proxy_state	状态信息，0 正常状态，1 熔断状态，2 锁定状态	GAUGE
proxy_current_connections	当前客户端连接数	GAUGE
parsed_sql_total	SQL 解析数，按 INSERT、DELETE、UPDATE、SELECT、DDL、DCL、DAL、TCL、RQL、RDL、RAL、RUL 类型区分	COUNTER
routed_sql_total	SQL 路由数，按 INSERT、DELETE、UPDATE、SELECT 类型区分	COUNTER
routed_result_total	路由结果数，统计路由到的存储节点和表的次数	COUNTER
proxy_transactions_total	事务总数，按 commit, rollback, autocommit 分类	COUNTER
proxy_execute_errors_total	执行异常数，按异常类型分类	COUNTER
proxy_requests_total	接收请求数	COUNTER
proxy_execute_total	执行总数（统计路由到存储节点执行的 SQL 数）	COUNTER
proxy_execute_error_total	执行异常数	COUNTER
proxy_request_bytes_total	请求字节数	COUNTER
proxy_response_bytes_total	响应字节数	COUNTER
parse_sql_latency_millis	解析 SQL 耗时	HISTOGRAM
route_sql_latency_millis	路由 SQL 耗时	HISTOGRAM
proxy_execute_latency_millis	执行耗时	HISTOGRAM
commit_sql_count_histogram	每次事务中提交 SQL 数量分布 HISTOGRAM	
rollback_sql_count_histogram	每次回滚事务中 SQL 数量分布	HISTOGRAM
proxy_connection_usage_seconds	连接到 Proxy 的连接持续时长分布	HISTOGRAM

DBPlusEngine-Driver 中使用 Agent

启动 Agent

- 准备集成了 DBPlusEngine-Driver 的项目，比如 SpringBoot 项目
- 启动时添加 javaagent 配置

```
java -javaagent:/xxx/shardingsphere-agent-${latest.release.version}.jar -jar spring-boot-dbplusengine-driver-test.jar
```

Agent Metrics

指标	描述	指标类型
build_info	构建版本信息	GAUGE
jdbc_state	状态，0 正常状态，1 熔断状态，2 锁定状态	GAUGE
jdb c_meta_data_info	元数据信息	GAUGE
parsed_sql_total	SQL 解析数，按 INSERT、DELETE、UPDATE、SELECT、DDL、DCL、DA L、TCL、类型区分	COUNTER
routed_sql_total	SQL 路由数，按 INSERT、DELETE、UPDATE、SELECT 类型区分	COUNTER
rou ted_result_total	路由结果数，统计路由到的存储节点和表的次数	COUNTER
jdbc_stateme nt_execute_total	执行 SQL 语句数	COUNTER
jdb c_statement_exec ute_errors_total	执行异常数	COUNTER
jdbc_tr ansactions_total	事务数，按 commit, rollback, autocommit 分类	COUNTER
jdb c_statement_execut e_latency_millis	执行 SQL 耗时	HIS-TOGRAM
parse_sq l_latency_millis	解析 SQL 耗时	HIS-TOGRAM
route_sq l_latency_millis	路由 SQL 耗时	HIS-TOGRAM

9.10 诊断优化

9.10.1 集群巡检内容

配置部分

- 操作系统配置
- SphereEx 产品配置
- SphereEx 授权配置
- 应用端连接配置

运行时部分

- 系统监控数据（CPU、MEM、NET、IO）
- SphereEx 监控数据
- 应用端监控数据

相关日志

- SphereEx 系统日志
- SphereEx 慢查询日志
- 应用端日志

9.11 安全管理

9.11.1 集群生态工具安全

SphereEx-Boot 安全

使用非 root 用户安装。

SphereEx-Console 安全

- 使用非 root 用户安装
- https 访问支持
- 链接到用户权限体系

9.11.2 集群组件安全

- 使用非 root 用户安装
- 访问组件无权限限制

9.11.3 集群安全

- 使用非 root 用户安装
- 集群访问安全：连接到计算节点运行日志
- 链接到集群的权限体系

9.11.4 数据安全

工具、组件、集群的配置数据存储数据的安全。

工具/组件	数据存储安全
SphereEx-Boot	无数据存储
SphereEx-Console	存储为底层数据库，推荐主从配置
治理中心	组件自身高可用保证
日志中心	组件自身高可用保证
监控中心	组件自身高可用保证
计算节点	无状态无需高可用
存储节点	组件自身高可用保证

9.11.5 访问安全

- 目前不支持 APP 到 SphereEx-DBPlusEngine-Proxy 的 SSL 访问
- 支持 SphereEx-DBPlusEngine-Proxy 到 DB 的 SSL 配置 url 配置示例: `jdbc:mysql://localhost:3306/testdb?useSSL=false`
- App 访问集群不支持 SSL
- 访问 SphereEx-Console 支持 https，证书需要用户自备
- SphereEx-Console 访问 SphereEx-DBPlusEngine-Proxy SSL 支持

9.12 可用性管理

9.12.1 集群自身高可用

多个计算节点高可用说明：因计算节点是无状态的，发生故障时无需切换，只要一个节点是存活的集群就是可用的。

9.12.2 集群业务高可用

集群和业务之间需添加接入层的高可用软件来保障高可用，SphereEx 可以配合用户通过解决方案来实现。

9.12.3 同城多中心架构图

■ 部署架构图

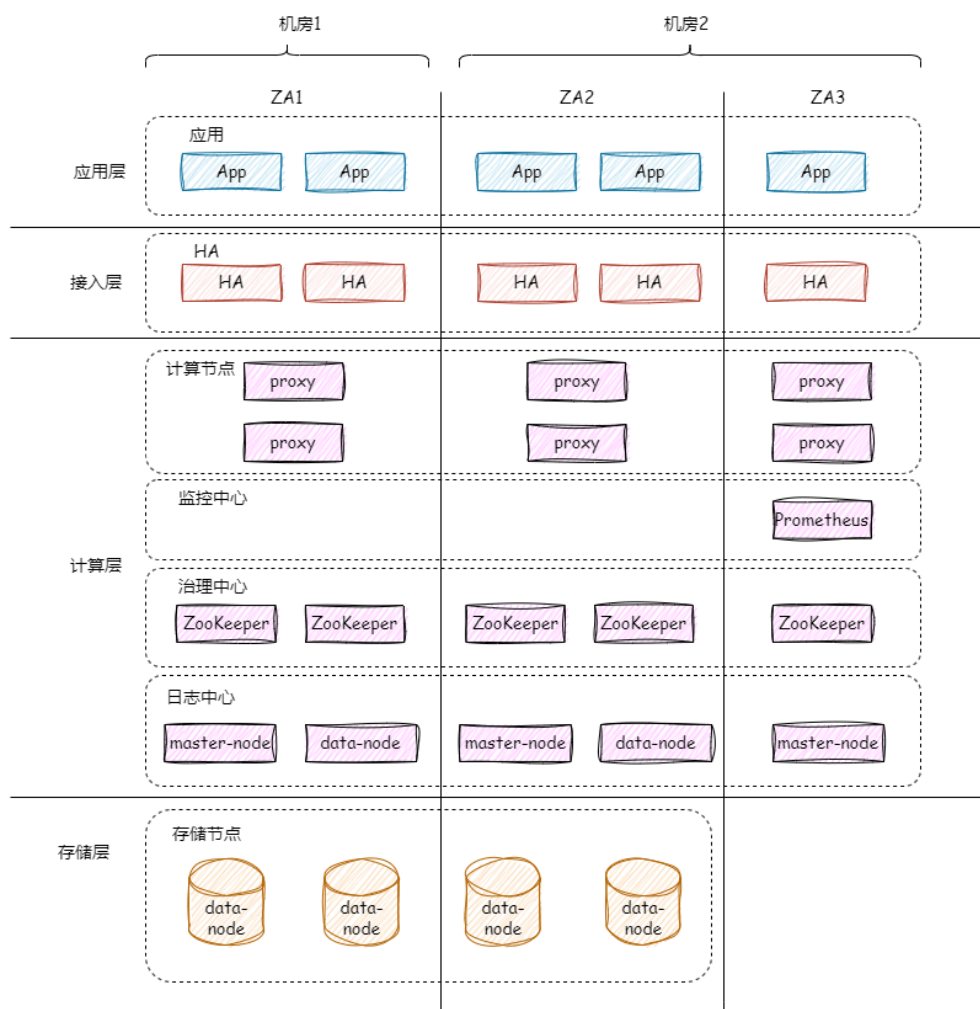


Fig. 7: 部署架构图

■ 切换说明

故障	RPO	PTO
单 AZ 故障	0	0
多 AZ 故障	0	取决于恢复任意 AZ 的时间

9.12.4 同机房架构图

部署架构图

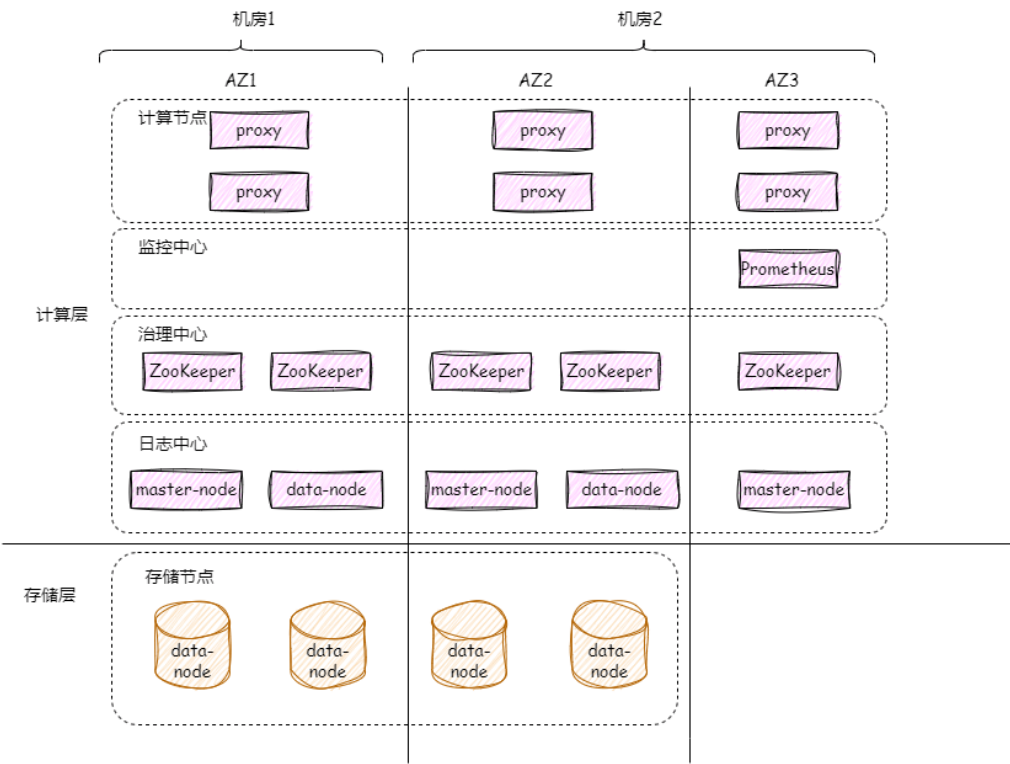


Fig. 8: 部署架构图

切换说明

故障	RPO	PTO
单 AZ 故障	0	0
多 AZ 故障	0	取决于恢复任意 AZ 的时间

9.12.5 集群组件可用性说明

组件名称	部署架构	后台技术	安全性说明
监控中心	单机	Prometheus	出现故障人工处理
监控插件	单机		出现故障人工处理
治理中心	集群	ZooKeeper	出现故障自动切换
日志中心	集群	Elasticsearch+logstash	出现故障自动切换
Filebeat	单机		出现故障人工处理
计算节点	单机	Engine-Proxy	出现故障自动下线，无需切换
存储节点	用户部署环境	用户部署环境	用户部署环境

ZooKeeper 集群不可用（大于半数节点不可用）

SQL 分类	SQL 语言分类	Engine 集群可用性描述
用户 SQL	DQL	可用
	DML	可用
	DDL	不可用
	DCL	不支持
	TCL	不支持
DistSQL	RQL	可用（从节点内存读取数据，重启后不可用）
	RDL	不可用
	RAL	不可用
	RUL	不可用

SQL 语言分类

SQL(Structure Query Language) 语言共分为四大类：数据查询语言 DQL，数据操作语言 DML，数据定义语言 DDL，数据控制语言 DCL。另外数据库中事务控制语言称为 TCL。

1. DQL 概述

DQL(Data Query Language) 数据查询语言，数据检索语句，用于从表中获取数据。关键字有：select。

Select 语句：查询表里的数据。

2. DML 概述

DML(Data Manipulation Language) 数据操作语言，用于操作数据库对象中包含的数据，操作的对象是记录。主要的关键字有：update、delete、insert。DML 操作可以手动控制事务的开启、提交和回滚。

Insert 语句：向数据表张插入一条记录。

Delete 语句：删除数据表中的一条或多条记录，也可以删除数据表中的所有记录，但是，它的操作对象仍是记录。

Update 语句：用于修改已存在表中的记录的内容。

3. DDL 概述

DDL(Data Definition Language) 数据定义语言，用于定义和管理数据库中的所有对象的语言，对数据库中的某些对象 (例如 database、table) 进行管理。操作对象包括数据库本身以及数据库对象，如表、视图等等。主要的关键字有：create、alter、drop、truncate。DDL 操作是隐性提交的，不能回滚。

Create 语句：可以创建数据库和数据库的一些对象。

Drop 语句：可以删除数据表、索引、触发程序、条件约束以及数据表的权限等。

Alter 语句：修改数据表定义及属性。

Truncate 语句：清空表里的数据。

4. DCL 概述

DCL(Data Control Language) 数据控制语句，操作对象是数据库的权限，并控制数据库操纵事务发生的时间及效果，对数据库实行监视等。这些操作使数据更加的安全。主要的关键字有：grant、revoke 。

Grant 语句：允许对象的创建者给某用户或某组或所有用户 (PUBLIC) 某些特定的权限。

Revoke 语句：可以废除某用户或某组或所有用户访问权限。

5. TCL 概述

TCL(Transaction Control Language) 事务控制语句，用于控制事务。主要的语句关键字有：commit、rollback、savepoint、set transaction。

Rollback 语句：用于事务的回退操作。

Savepoint 语句：为回退而存在，个数没有限制，与虚拟机中快照类似。savepoint 是事务中的一点。用于取消部分事务，当结束事务时，会自动的删除该事务中所定义的所有保存点。

Commit 语句：用于提交事务。

Set transaction 语句：设置事务的各种状态，比如只读、读/写、隔离级别。

DistSQL 语言分类

- RDL (Resource & Rule Definition Language) 负责资源和规则的创建、修改和删除；
- RQL (Resource & Rule Query Language) 负责资源和规则的查询和展现；
- RAL (Resource & Rule Administration Language) 负责 Hint、事务类型切换、分片执行计划查询等管理功能。
- RUL (Resource Utility Language) 为 SphereEx-DBPlusEngine 的工具类语言，提供 SQL 解析、SQL 格式化、执行计划预览等功能。

存储节点高可用感知

定义

高可用是现代系统的最基本诉求，作为系统基石的数据库，对于高可用的要求也是必不可少的。

在存算分离的分布式数据库体系中，存储节点和计算节点的高可用方案是不同的。

对于有状态的存储节点来说，需要其自身具备数据一致性同步、探活、主节点选举等能力；

对于无状态的计算节点来说，需要感知存储节点的变化，同时，还需要独立架设负载均衡器，并具备服务发现和请求分发的能力。

尽可能的保证 7X24 小时不间断的数据库服务，是 SphereEx-DBPlusEngine 高可用模块的主要设计目标。

SphereEx-DBPlusEngine 不提供数据库高可用的能力，它通过第三方提供的高可用方案感知数据库主从关系的切换。确切来说，SphereEx-DBPlusEngine 提供数据库发现的能力，自动感知数据库主从关系，并修正计算节点对数据库的连接。

动态读写分离

高可用和读写分离一起使用时，读写分离无需配置具体的主库和从库。高可用的数据源会动态的修正读写分离的主从关系，并正确的疏导读写流量。

- 支持项
 - MySQL MGR 单主模式。
 - MySQL 主从复制模式。
 - OpenGauss 主从复制模式。
- 不支持项
 - MySQL MGR 多主模式。#### 原理介绍

SphereEx-DBPlusEngine 提供的高可用方案，允许用户进行二次定制开发及实现扩展，主要分为四个步骤：前置检查、动态发现主库、动态发现从库、同步配置。

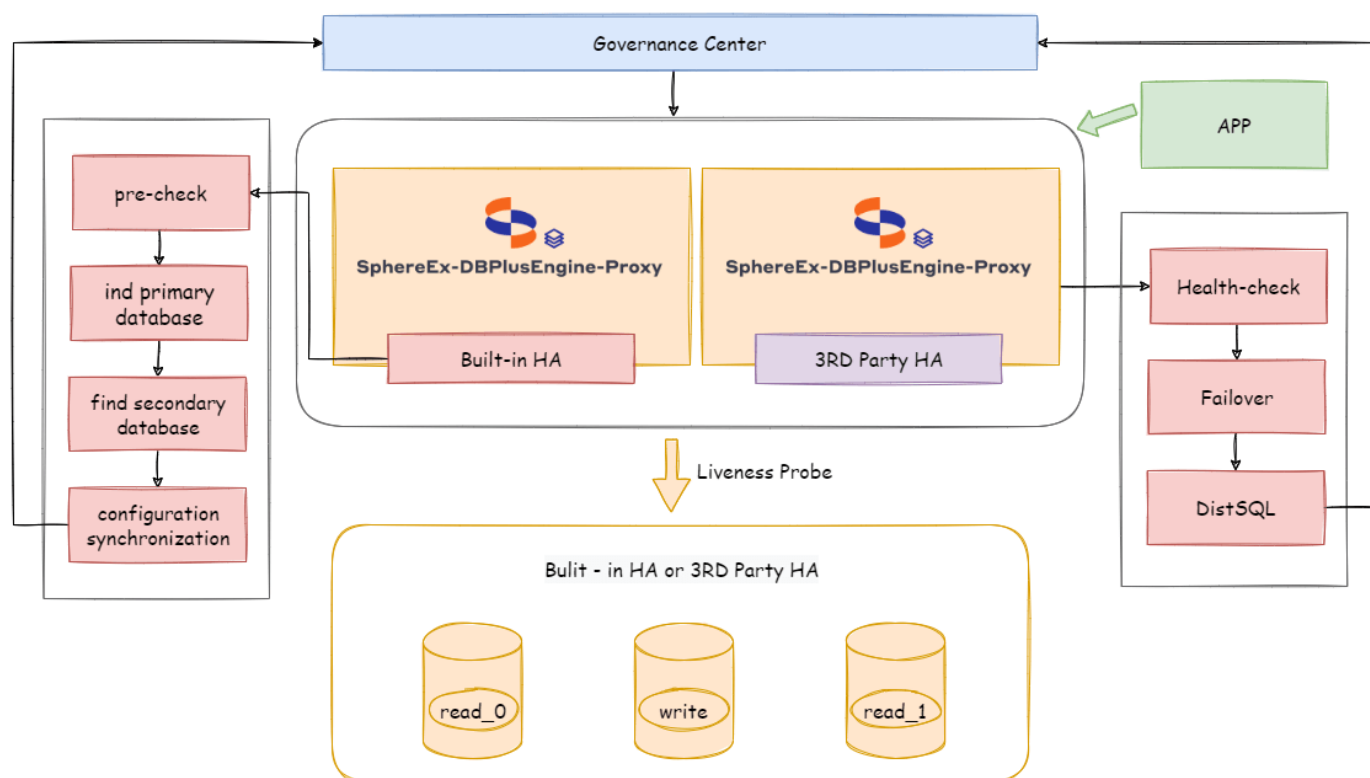


Fig. 9: 原理图

存储节点状态收集

用户可以通过配置 `storage_node_heartbeat_interval_milliseconds` 参数，配置存储节点状态收集间隔，单位毫秒，如果配置等于 0，则关闭存储节点状态收集功能。从计算节点的运行日志中可以查看到存储节点状态收集的日志

通过 DistSQL 配置

```
set dist variable storage_node_heartbeat_interval_milliseconds = 10000;
```

通过 yaml 文件配置

```
props:
  storage-node-heartbeat-interval-milliseconds: 10000
```

使用场景 1. 单播路由，过滤掉不可用的数据源。单播路由使用场景：比如广播表查询。2. 静态读写分离读路由，过滤掉不可用的数据源。

9.12.6 集群生态工具可用性说明

工具	部署架构	安全性说明
SphereEx-Boot	单机	出现故障人工处理
SphereEx-Console	单机	出现故障人工处理，后台数据库为主从架构，无法自动切换需人工介入

9.13 备份恢复

9.13.1 集群生态工具备份恢复

备份 SphereEx-Boot

无需备份

备份 SphereEx-Console

目前仅支持 SphereEx-Console 数据库的备份。

使用 SphereEx-Boot 备份 SphereEx-Console 后台的 MySQL 数据库。

```
spex console backup -n
```

恢复 SphereEx-Boot

无需恢复

恢复 SphereEx-Console

目前需要手动恢复备份的 SphereEx-Console 数据库 sql 文件

9.13.2 集群备份恢复

使用 SphereEx-DBPlusEngine 提供了 EXPORT/IMPORT 用于备份和恢复集群的元数据。目前支持对指定逻辑库的备份，备份内容包含数据源和 rule 两部分。

备份集群

登录任意计算节点，执行 DistSQL 备份集群指定逻辑库元数据。

方式一导出到 Proxy 所在机器的指定路径

```
EXPORT DATABASE CONFIGURATION TO FILE "/xxx/config-sharding.yaml";
```

方式二导出并将结果输出在客户端，由客户端自行保存为任意的 yaml 文件

```
EXPORT DATABASE CONFIGURATION;
```

恢复集群

登录任意计算节点，执行 DistSQL 恢复集群指定逻辑库元数据。
 创建的逻辑库名称要与导出时的名称一致，否则提示错误

```
CREATE DATABASE databaseName;

USE databaseName;

IMPORT DATABASE CONFIGURATION FROM FILE '/xxx/config-sharding.yaml'
```

9.14 故障排查

9.14.1 集群生态工具可用性判断

组件名称	可用性判断
SphereEx-Console	ping 命令探活，登录 web 端
SphereEx-Boot	spex -version 查看版本

9.14.2 集群组件可用性判断

组件名称	可用性判断
监控中心	SphereEx-Console 探活
监控插件	SphereEx-Console 查看监控
治理中心	SphereEx-Console 探活
日志中心	SphereEx-Console 探活、查看日志
Filebeat	SphereEx-Console 查看日志
计算节点	SphereEx-Console 探活
存储节点	SphereEx-Console 探活

9.14.3 集群可用性判断

- SphereEx-Console 探活
- SphereEx-Console 查看日志
- 查看计算节点日志

9.15 依赖管理

本章节主要介绍 DBPlusEngine 可选依赖的下载方式。

9.15.1 添加 Narayana 依赖包

添加 Narayana 依赖需要下载以下 jar 文件并将其添加至 ext-lib 下。

jar 文件下载地址

- arjuna-5.12.4.Final.jar
- common-5.12.4.Final.jar
- javax.activation-api-1.2.0.jar
- jaxb-api-2.3.0.jar
- jaxb-core-2.3.0.jar
- jaxb-impl-2.3.0.jar
- jboss-connector-api_1.7_spec-1.0.0.Final.jar
- jboss-logging-3.2.1.Final.jar
- jboss-transaction-api_1.2_spec-1.0.0.Alpha3.jar
- jboss-transaction-spi-7.6.0.Final.jar
- jta-5.12.4.Final.jar
- narayana-jts-integration-5.12.4.Final.jar
- shardingsphere-transaction-xa-narayana.jar

请根据 proxy 版本下载对应 shardingsphere-transaction-xa-narayana.jar 文件。

9.15.2 添加 Bitronix 依赖包

添加 Bitronix 依赖需要下载以下 jar 文件并将其添加 ext-lib 目录下。

jar 文件下载地址

- btm-2.1.3.jar
- shardingsphere-transaction-xa-bitronix.jar

请根据 proxy 版本下载对应 shardingsphere-transaction-xa-bitronix.jar 文件。

9.15.3 添加数据库驱动依赖

- DBPlusEngine 已经内置了 Postgre 数据库的驱动
- DBPlusEngine 未内置 MySQL 的驱动，需要用户自己加根据数据库的版本自行选择

10

最佳实践

10.1 资源参数模板

10.1.1 简单场景

企业使用 SphereEx-DBPlusEngine 承载某种使用场景，如分片、加密等。

分片场景

组件	配置
MySQL	8C 16G * 1
Proxy(低性能)	2C 4G * 2
Proxy(中性能)	4C 8G * 2
Proxy(高性能)	8C 16G * 2
ZooKeeper	2C 2G * 3

上述配置基于标准 TPCC 基准测试得出结果，与实际生产环境存在差异，仅供参考。

加密场景

组件	配置
MySQL	8C 16G * 1
Proxy(低性能)	2C 4G * 2
Proxy(高性能)	4C 8G * 2
ZooKeeper	2C 2G * 3

上述配置基于标准 TPCC 基准测试得出结果，与实际生产环境存在差异，仅供参考。

10.1.2 综合场景

企业使用 SphereEx-DBPlusEngine 承载多种使用场景，如分片、加密等。

小型业务

业务相对单一，但可以预见业务量会有大规模增长，为未来架构提前做预留。

组件	配置
MySQL	8C 32G * 1
Proxy	4C 4G * 2
ZooKeeper	2C 2G * 3

中型业务

业务较为复杂，数据量较大，且不断快速增加。需要在数据层架构上进行更新。

组件	配置
MySQL	32C 64G * 4
Proxy	8C 16G * 2
ZooKeeper	2C 2G * 3

大型业务

业务规模非常大，海量数据，数据需要尽快进行分片处理。同时对可用性要求很高。

组件	配置
MySQL	32C 64G * 16
Proxy	8C 16G * 4
ZooKeeper	2C 2G * 7

10.2 如何迁移数据库

10.2.1 数据表

有单字段主键，字段类型支持整型和字符串。推荐使用整型主键，并发度更高，性能更好。迁移过程中暂不支持对原表做 DDL 操作。

10.2.2 优化迁移配置

迁移配置可以根据情况自定义。有些配置项可以通过比较精确的计算来提升性能。

完整配置示例：

```
ALTER MIGRATION RULE (  
  READ(  
    WORKER_THREAD=40,  
    BATCH_SIZE=1000,  
    SHARDING_SIZE=10000000,  
    RATE_LIMITER (TYPE(NAME='QPS',PROPERTIES('qps'='500')))  
  ),
```

```
WRITE(  
  WORKER_THREAD=40,  
  BATCH_SIZE=1000,  
  RATE_LIMITER (TYPE(NAME='TPS',PROPERTIES('tps'='2000')))  
),  
STREAM_CHANNEL (TYPE(NAME='MEMORY',PROPERTIES('block-queue-size'='10000')))  
);
```

不配置也有默认值，可以通过 DistSQL 查看：

```
SHOW MIGRATION RULE;
```

并发数

全量迁移会根据表维度以及数据量维度进行拆分，提高并发度，从而提升性能。
默认按 1000 万数据一段进行拆分。假设有 2 张表，分别有 2 亿数据和 4 亿数据，可以拆分为 60 段。
如果机器配置高，并且对限流要求不高，那么可以考虑提高并发线程数 WORKER_THREAD 到 60。

限流

如果源端数据库已上线使用，并且对快速响应有较高要求，那最好开启限流，避免影响线上业务。
可以根据数据库服务器配置情况设置一个较小限流值，再根据负载动态修改限流值，重启 job 可以生效。
示例 DistSQL：

```
-- 修改限流值  
ALTER MIGRATION RULE (  
  READ(  
    RATE_LIMITER (TYPE(NAME='QPS',PROPERTIES('qps'='1000')))  
  )  
);  
  
-- 查询迁移 job 列表  
SHOW MIGRATION LIST;  
  
-- 关闭 job  
STOP MIGRATION {jobId};  
  
-- 启动 job  
START MIGRATION {jobId};
```

10.2.3 异构迁移

目标端是 proxy，目前支持的数据库类型还比较有限，包括：MySQL、PostgreSQL、openGauss。新的数据库类型正在支持中。

源端支持情况如下表所示：

	Oracle	MySQL	PostgreSQL	openGauss
全量迁移	Y	Y	Y	Y
增量迁移	N	Y	Y	Y
数据对比	Y	Y	Y	Y

源端支持的数据库类型更多一些。

第一类：除了全量迁移，还支持增量同步，包括：MySQL、PostgreSQL、openGauss。

第二类：只支持全量迁移，包括：Oracle 等商业数据库。目前经过实际验证的是 Oracle，对应目标端包括 PostgreSQL 和 MySQL。增量同步正在支持中。

目标端支持情况：

	MySQL	PostgreSQL	openGauss
全量迁移	Y	Y	Y
增量迁移	Y	Y	Y
数据对比	Y	Y	Y

数据一致性校验支持异构数据库，但是支持的算法类型有限制，目前推荐使用 DATA_MATCH 算法。

10.3 数据安全的最佳实践

10.3.1 适用场景

数据加密

数据加密是对信息进行保护的一种可靠方法，它利用密码技术对信息进行加密，实现信息隐蔽，从而起到保护信息安全的作用。数据加密适用于对数据资产具有安全要求的企业级应用，如以下两种常见的业务场景：

- 1. 新业务，需要对入库数据进行加密处理。

为了减少应用侧的复杂度，也能适应将来的加密需求变化，使用 SphereEx-DBPlusEngine 提供的数据层加密来实现加密功能。这样，加密规则对业务应用透明，加密需求变化时也不需要对应应用进行改造，利于实现一套灵活的系统架构。

- 2. 存量业务，已有明文数据。

对于已经在线上运行的系统，如何在对新增数据加密的同时，完成对存量数据的加密处理，是一项非常棘手的任务。基于 SphereEx-DBPlusEngine 提供的加密引擎和加密迁移方案，能够完整、安全、透明、低成本的完成数据改造，快速让存量业务满足企业和法律法规的安全要求。

全局权限

SphereEx-DBPlusEngine 提供了多种级别的用户权限管理能力，可分别为用户设置库级、表级、列级的操作权限，满足细粒度安全管理要求。全局权限功能适用于以下应用场景：

- 1. 需要限制登录主机的。

通过限制用户的登录主机，保证其只能通过可信的地址连接到 SphereEx-DBPlusEngine，拒绝非法地址的访问，极大程度的保障数据库系统的接入安全。

- 2. 需要按用户管理权限的。

通过为每一位数据库用户分别设置库、表、列的操作权限，进行细粒度的权限管理。

- 3. 需要按角色管理权限的。

当用户可以按照角色进行分类时，通过对角色进行授权，让角色下的每一个用户具备相同的操作权限。通过这种方式，既实现了细粒度的权限控制，又同时简化了管理流程，避免对多个用户进行重复的授权操作。

需要注意的是，全局权限管理能力适用于 SphereEx-DBPlusEngine Proxy，Driver 不具备该特性。

10.3.2 系统规划

部署一个全新的数据加密和权限管理场景，满足以下要求：- 对用户表 t_user 的身份证号字段 id_number 进行加密；- 新建一个用户 enc，只能对 t_user 进行 INSERT、SELECT 操作。

选择加密算法

SphereEx-DBPlusEngine 支持多种加密算法，包括 MD5、AES、RC4、SM3、SM4 等等，也可以根据业务需要接入自定义的算法。

本次业务选择常用的 AES 算法对身份证号进行加密。

选择权限控制器

SphereEx-DBPlusEngine 目前提供了三种安全级别的权限控制器，分别是：- ALL_PERMITTED：为所有用户授予所有权限，不进行安全拦截；- DATABASE_PERMITTED：限制用户访问逻辑库的权限，逻辑库级别数据隔离；- SphereEx:PERMITTED：由 SphereEx 提供的企业级权限控制，可以限制用户对库、表、列的操作权限。

由于本次应用需要仅对 enc 用户授予指定表的部分权限，因此选择使用 SphereEx:PERMITTED。

对开发的影响

SphereEx-DBPlusEngine 能够自动在数据插入时进行加密，也可以在数据查询时自动解密，对业务开发没有影响。

10.3.3 前提条件

1. 具备一套可用的 ZooKeeper，作为治理中心；
2. 至少一个可访问的数据存储节点，如一个 MySQL 数据库 demo_ds。

10.3.4 操作步骤

1. 安装 SphereEx-DBPlusEngine Proxy，并配置为“集群模式”；
2. 配置 server.yaml 中的 AUTHORITY 规则，将 privilege.type 设置为 SphereEx:PERMITTE，并设置 root 用户的初始密码；
3. 启动 SphereEx-DBPlusEngine Proxy；
4. 使用 root 用户登录到 SphereEx-DBPlusEngine Proxy；

```
mysql -h ${host} -P ${port} -u root -p
```

5. 创建逻辑库 encrypt_db；

```
mysql> CREATE DATABASE encrypt_db;  
Query OK, 0 rows affected (0.20 sec)
```

```
mysql> USE encrypt_db;  
Database changed  
mysql>
```

6. 添加 MySQL 存储节点 demo_ds 到 encrypt_db 中；


```
mysql> REGISTER STORAGE UNIT ds_0 (
-> HOST="{demo_ds_host}",
-> PORT="{demo_ds_port}",
-> DB="demo_ds",
-> USER="{demo_ds_user}",
-> PASSWORD="{demo_ds_password}"
-> );
Query OK, 0 rows affected (0.10 sec)
```

7. 为业务表 t_user 创建加密规则，对 id_number 字段进行加密；

```
mysql> CREATE ENCRYPT RULE t_user (
-> COLUMNS(
-> (NAME=id_number,CIPHER=id_number,TYPE(NAME="AES",PROPERTIES("aes-key-value"="123456abc")))
-> ), QUERY_WITH_CIPHER_COLUMN=false);
Query OK, 0 rows affected (0.46 sec)
```

8. 创建表 t_user ；

```
mysql> CREATE TABLE `t_user` (
-> `id` int(11) NOT NULL,
-> `name` varchar(32) NOT NULL,
-> `id_number` varchar(128) NOT NULL,
-> PRIMARY KEY (`id` )
-> );
Query OK, 0 rows affected (0.21 sec)
```

9. 创建用户 enc，并授予 t_user 表的 INSERT 和 SELECT 权限；

```
mysql> CREATE DIST USER 'enc'@'%' IDENTIFIED BY '123456';
Query OK, 0 rows affected (0.15 sec)

mysql> GRANT DIST SELECT,INSERT ON encrypt_db.t_user TO 'enc'@'%';
Query OK, 0 rows affected (0.02 sec)
```

10. 部署完成，登录 enc 用户进行验证：

```
-- Connect
mysql -h {host} -P {port} -u enc -p
-- USE db
mysql> USE encrypt_db;
Database changed
-- INSERT
mysql> INSERT INTO t_user values(1,'张三','110111*****');
Query OK, 1 row affected (0.08 sec)
-- SELECT
mysql> SELECT * FROM t_user;
+----+-----+-----+
| id | name | id_number |
+----+-----+-----+
| 1 | name1 | 2H385oL4Y16CB5ec41FaHQ== |
+----+-----+-----+
1 row in set (0.00 sec)
-- UPDATE
mysql> UPDATE t_user set name = '张三丰' WHERE id = 1;
ERROR 1997 (C1997): Runtime exception: [SQL checking failed. Error message: Access denied for operation [UPDATE] of subject encrypt_db.t_user:name.]
```

10.3.5 配置示例

server.yaml

```
mode:
  type: Cluster
repository:
  type: ZooKeeper
props:
  namespace: governance_ds
  server-lists: localhost:2181
  retryIntervalMilliseconds: 500
  timeToLiveSeconds: 60
  maxRetries: 3
  operationTimeoutMilliseconds: 500

authority:
  users:
    - user: root@%
      password: root
    - user: sharding
      password: sharding
  privilege:
    type: SphereEx:PERMITTED
```

10.3.6 FAQ

1. SphereEx-DBPlusEngine 提供的全局权限，与物理 DB 权限的关系是什么？

SphereEx-DBPlusEngine 的全局权限，是从逻辑层面对分布式数据库的接入用户进行权限控制，限制的是接入 DBPlusEngine 的用户对逻辑库、逻辑表以及逻辑列的访问权限。

DB 权限限制的是实际的物理连接中用户对 DB 的操作，其中一些对象如表名、列名是经过 DBPlusEngine 改写的，可能和业务 SQL 中的对象名称不一致（如分片表名、加密列名）。

另外，由于 DBPlusEngine 是分布式数据库，底层会接入多个物理 DB，各个物理 DB 的用户和权限可能不同，但 DBPlusEngine 对最终用户提供统一的认证和权限校验，以实现更好的体验。

2. 物理 DB 应提供怎样的授权？

用户需要通过 DBPlusEngine 来操作业务数据，因此在添加存储资源时，应该使用具备足够权限的连接用户。

3. DBPlusEngine 的权限配置如何备份？

DBPlusEngine 的用户和授权信息存储在治理中心，可以通过治理中心的持久化方式进行备份和恢复。未来 DBPlusEngine 也会提供 DistSQL 语句进行针对性的导出和导入，方便用户操作。

10.4 如何流量治理

10.4.1 流量不做变形

特性	描述
读写分离	分担主库的压力，将读流量分发给从库。可通过 DistSQL 手动禁用从库，从而达到下线从库的目的。读写分离也可搭配数据库发现功能，可动态的感知主库与从库的关系并动态的纠正，保证读流量和写流量的正确路由。
负载均衡	使用读写分离功能时，可配置从库负载均衡算法。比如：随机，权重、轮询等。
影子库	影子库功能是在线压测场景下，数据库层面的解决方案。通过识别执行 SQL 中携带的压测标识，结合影子算法灵活的配置，满足复杂业务场景的在线压测需求。可以将压测流量路由到影子库，线上正常流量路由到生产库。
限流	不支持
熔断	DBPlusEngine 提供实例熔断的功能，当计算节点存在元数据不一致、或终止实例提供服务时，可通过 DistSQL 进行熔断或恢复实例。
白名单	不支持
双路由	DBPlusEngine 提供了转发流量的功能，利用不同的 traffic 算法，可以在不同场景下，将 JDBC 的流量路由到 Proxy 端，目前算法有：SQL 字符串匹配、SQL 正则匹配、SQL Hint 匹配以及事务单元四种算法。

10.4.2 变形的流量

特性	描述
数据加密	DBPlusEngine 提供了数据加解密功能，具有完整、安全、透明化、低改造成本的特点。它可以让用户在不改造业务逻辑的基础上，完成数据的加解密，让用户数据完全合规，DBPlusEngine 还内置了多种加密算法，可以满足用户多种场景，另外它还提供了加密洗牌的功能，帮助用户完成既存数据的加密清洗。
数据分片	DBPlusEngine 提供了数据分片的功能，它可以将原有的数据按照一定的维度拆分至多个存储节点，帮助用户提升系统性能和可用性，同时降低运维成本。数据分片功能可以自动管理用户的 SQL，自动路由、改写，并且对结果集做归并，让用户像使用单表一样来使用分片表。

10.5 表关联

10.5.1 原理

DBPlusEngine 表关联目前支持多个分片表为绑定表场景的关联查询，以及广播表和分片表、广播表和单表的关联查询，不支持非绑定表的多个分片表关联查询，以及分片表和单表，单表和单表的跨库关联查询。

DBPlusEngine 在处理表关联查询时，尽可能地将关联查询下推至底层数据库执行，充分使用原生数据库的计算能力，DBPlusEngine 会对底层数据库返回的结果集进行归并，使得最终的结果满足逻辑 SQL 的关联查询语义。

10.5.2 建议

1. 使用绑定表进行多表关联查询时，通过使用分片键进行关联，可以避免笛卡尔积关联，从而提升查询效率；
2. 对于分片表和其他表关联的场景，可以将大表根据业务场景进行分片，需要关联的小表可以采用广播方式下发到各个数据源，当分片表和广播表需要关联查询时，选择当前数据源上的广播表关联即可；
3. 对于业务场景中单独使用，不需要和分片表关联的小表，可以将它们存储在指定的数据源上，以单表的形式进行更新和查询。

10.5.3 示例

以如下的 SQL 为例，我们来对比下使用绑定表和非绑定表关联的差异。

```
SELECT i.* FROM t_order o JOIN t_order_item i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
```

- 在不配置绑定表关系时，假设分片键 order_id 将数值 10 路由至第 0 片，将数值 11 路由至第 1 片，那么路由后的 SQL 应该为 4 条，它们呈现为笛卡尔积：

```
SELECT i.* FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
SELECT i.* FROM t_order_0 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
SELECT i.* FROM t_order_1 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
SELECT i.* FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
```

- 在配置绑定表关系，并且使用 order_id 进行关联后，路由的 SQL 应该为 2 条：

```
SELECT i.* FROM t_order_0 o JOIN t_order_item_0 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
SELECT i.* FROM t_order_1 o JOIN t_order_item_1 i ON o.order_id=i.order_id WHERE o.order_id in (10, 11);
```

可以看出配置绑定表关系之后，查询的效率大大提升。

10.6 排序

10.6.1 原理

DBPlusEngine 排序功能依赖于数据库底层的排序功能，当用户输入一条包含 ORDER BY 的 SQL 时，DBPlusEngine 会将逻辑 SQL 路由到对应的存储数据源上，并将逻辑 SQL 改写为真实 SQL。由于真实 SQL 中同样包含了 ORDER BY 语句，返回的结果集自身也是有序的，因此 DBPlusEngine 在对多个数据源的结果进行归并时，利用每个结果集的有序性，通过维护一个有序队列来实现全局的有序。

10.6.2 建议

1. 合理地设置 max-connections-size-per-query 参数，保证 DBPlusEngine 在执行时能够为每个分片结果集分配至少一个连接持有结果集，避免将流式结果集转换为内存结果集占用过多内存；具体可参考[连接模式](#)。

10.6.3 示例

以如下 t_score 表查询为例，逻辑 SQL 中按照 score 字段排序，返回的各个分片的结果集按照 score 倒序排序。将 3 个数据结果集的当前游标指向的数据值进行排序，并放入优先级队列，t_score_0 的第一个数据值最大，t_score_2 的第一个数据值次之，t_score_1 的第一个数据值最小，因此优先级队列根据 t_score_0，t_score_2 和 t_score_1 的方式排序队列。

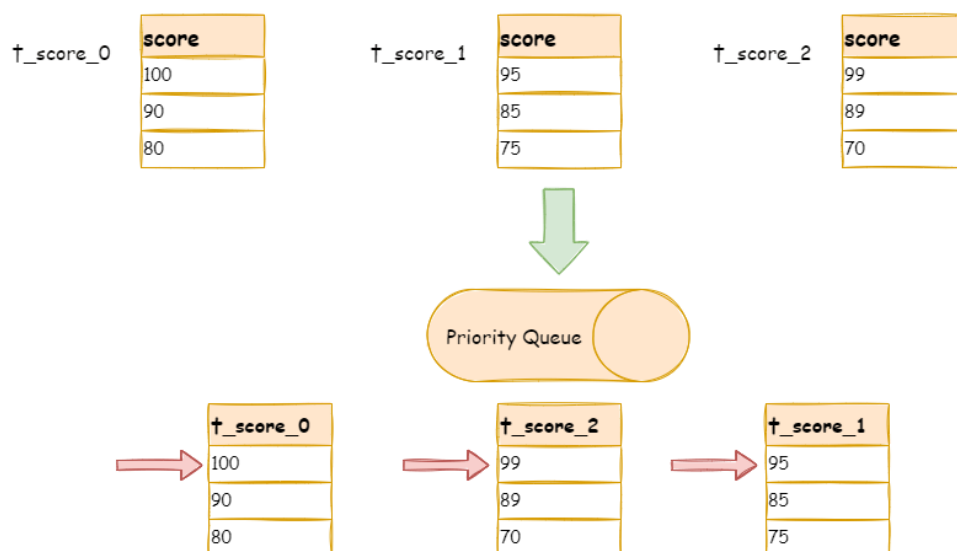


Fig. 1: 排序 1

下图则展现了进行 next 调用的时候，DBPlusEngine 排序是如何进行的。通过图中我们可以看到，当进行第一次 next 调用时，排在队列首位的 t_score_0 将会被弹出队列，并且将当前游标指向的数据值（也就是 100）返回至查询客户端，并且将游标下移一位之后，重新放入优先级队列。而优先级队列也会根据 t_score_0 的当前数据结果集指向游标的数据值（这里是 90）进行排序，根据当前数值，t_score_0 排列在队列的最后一位。之前队列中排名第二的 t_score_2 的数据结果集则自动排在了队列首位。

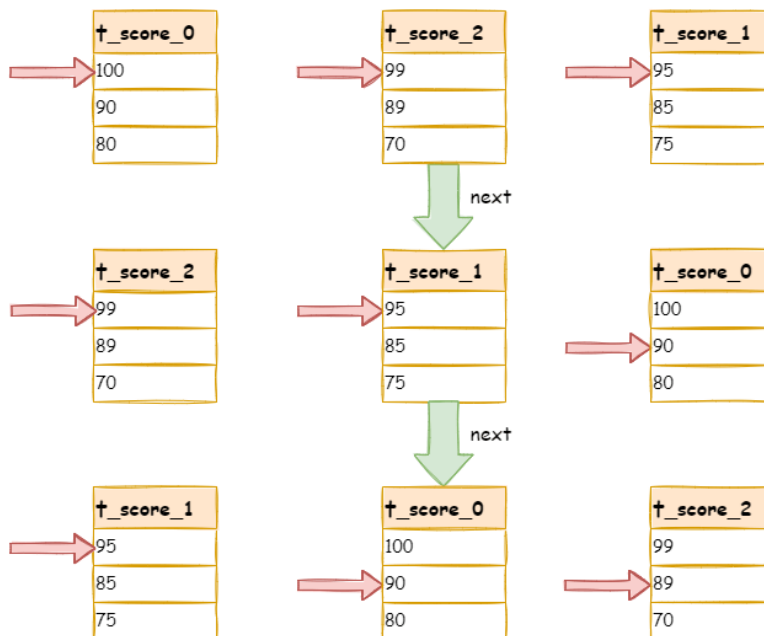


Fig. 2: 排序 2

可以看到，对于每个数据结果集中的数据有序，而多数据结果集整体无序的情况下，DBPlusEngine 无需将所有数据都加载至内存即可排序。它使用的是流式归并的方式，每次 next 仅获取唯一正确的一条数据，极大的节省了内存的消耗。

10.7 分组

10.7.1 原理

DBPlusEngine 分组功能可以分为流式分组和内存分组，流式分组是指通过流式查询的方式，逐条将数据加载至内存进行分组计算，流式分组要求 SQL 的排序项与分组项的字段以及排序类型（ASC 或 DESC）必须保持一致。内存分组是指将查询结果集一次加载至内存，然后在内存中进行分组计算，内存分组适用于 SQL 排序项与分组项的字段及排序类型（ASC 或 DESC）无法保持一致的场景，内存分组会占用更多内存。

10.7.2 建议

保证 SQL 的排序项与分组项的字段以及排序类型（ASC 或 DESC）一致，以使用流式分组，减少内存消耗；
说明：若按 AB 进行排序，按 BA 做分组或排序，不可使用到流式处理，因为字段不一致。

10.7.3 示例

以如下 SQL 为例：

```
SELECT name, SUM(score) FROM t_score GROUP BY name ORDER BY name;
```

假设根据科目分片，表结构中包含考生的姓名（为了简单起见，不考虑重名的情况）和分数，通过这个 SQL 我们可以获取每位考生的总分。

在分组项与排序项完全一致的情况下，取得的数据是连续的，分组所需的数据全数存在于各个数据结果集的当前游标所指向的数据值，因此可以采用流式归并。如下图所示。

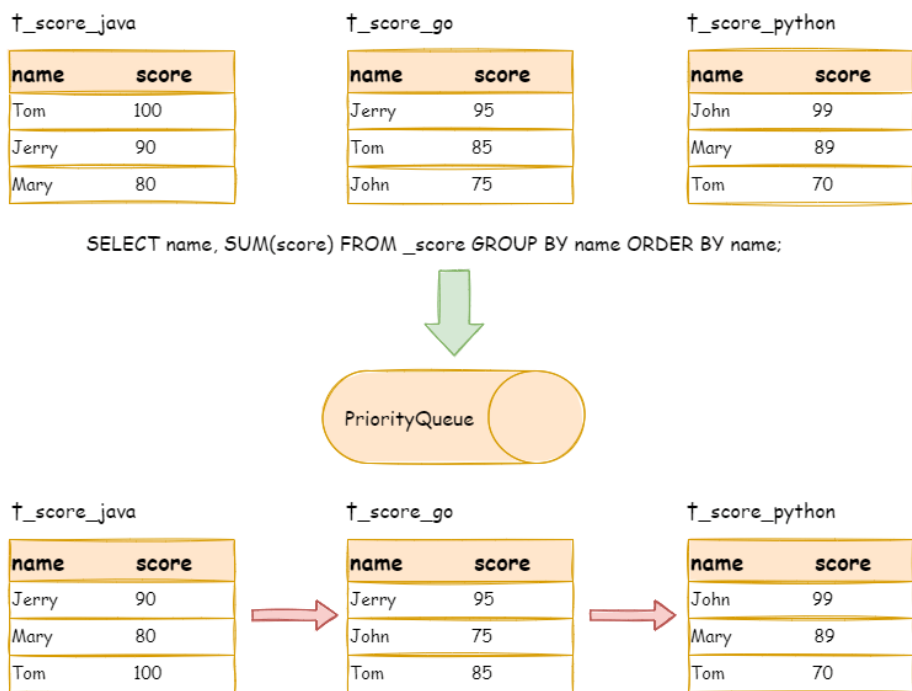


Fig. 3: 流式归并 1

DBPlusEngine 进行归并处理时，逻辑与排序归并类似。下图展现了进行 next 调用的时候，流式分组归并是如何进行的。

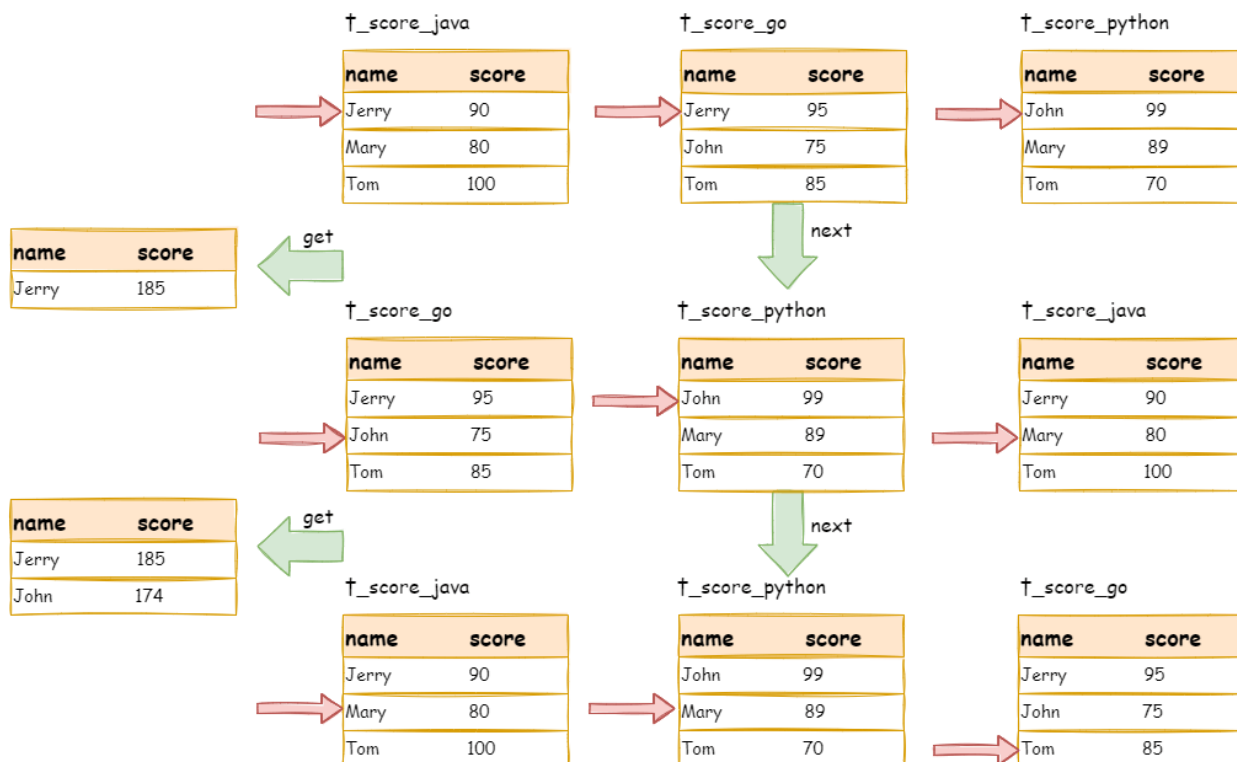


Fig. 4: 流式归并 2

通过图中我们可以看到，当进行第一次 `next` 调用时，排在队列首位的 `t_score_java` 将会被弹出队列，并且将分组值同为“Jerry”的其他结果集中的数据一同弹出队列。在获取了所有的姓名为“Jerry”的同学的分数之后，进行累加操作，那么，在第一次 `next` 调用结束后，取出的结果集是“Jerry”的分数总和。与此同时，所有的数据结果集中的游标都将下移至数据值“Jerry”的下一个不同的数据值，并且根据数据结果集当前游标指向的值进行重排序。因此，包含名字顺着第二位的“John”的相关数据结果集则排在队列的前列。

对于分组项与排序项不一致的情况，由于需要获取分组的相关的数据值并非连续的，因此无法使用流式归并，需要将所有的结果集数据加载至内存中进行分组和聚合。例如，若通过以下 SQL 获取每位考生的总分并按照分数从高至低排序：

```
SELECT name, SUM(score) FROM t_score GROUP BY name ORDER BY score DESC;
```

那么各个数据结果集中取出的数据与排序归并那张图的上半部分的表结构的原始数据一致，是无法进行流式归并的。

当 SQL 中只包含分组语句时，根据不同数据库的实现，其排序的顺序不一定与分组顺序一致。但由于排序语句的缺失，则表示此 SQL 并不在意排序顺序。因此，DBPlusEngine 通过 SQL 优化的改写，自动增加与分组项一致的排序项，使其能够从消耗内存的内存分组归并方式转化为流式分组归并方案。

10.8 分页查询

10.8.1 性能瓶颈

完全支持 MySQL、PostgreSQL 和 Oracle 的分页查询，SQLServer 由于分页查询较为复杂，仅部分支持。分页性能方面，查询偏移量过大的分页会导致数据库获取数据性能低下，以 MySQL 为例：

```
SELECT * FROM t_order ORDER BY id LIMIT 1000000, 10
```

这句 SQL 会使得 MySQL 在无法利用索引的情况下跳过 1,000,000 条记录后，再获取 10 条记录，其性能可想而知。而在分库分表的情况下（假设分为 2 个库），为了保证数据的正确性，SQL 会改写为：

```
SELECT * FROM t_order ORDER BY id LIMIT 0, 1000010
```

即将偏移量前的记录全部取出，并仅获取排序后的最后 10 条记录。这会在数据库本身就执行很慢的情况下，进一步加剧性能瓶颈。因为原 SQL 仅需要传输 10 条记录至客户端，而改写之后的 SQL 则会传输 $1,000,010 * 2$ 的记录至客户端。

10.8.2 DBPlusEngine 的优化

DBPlusEngine 进行了 2 个方面的优化。

首先，采用流式处理 + 归并排序的方式来避免内存的过量占用。由于 SQL 改写不可避免的占用了额外的带宽，但并不会导致内存暴涨。与直觉不同，大多数人认为 DBPlusEngine 会将 $1,000,010 * 2$ 记录全部加载至内存，进而占用大量内存而导致内存溢出。但由于每个结果集的记录是有序的，因此 DBPlusEngine 每次比较仅获取各个分片的当前结果集记录，驻留在内存中的记录仅为当前路由到的分片的结果集的当前游标指向而已。按归并思想合并 m 个长度为 n 的已排序数组，时间复杂度为 $O(mn(\log m))$ ，一般分片数量 m 都较小，可以认为时间复杂度为 $O(n)$ ，性能损耗很小。

其次，DBPlusEngine 对仅落至单分片的查询进行进一步优化。落至单分片查询的请求并不需要改写 SQL 也可以保证记录的正确性，因此在此种情况下，DBPlusEngine 并未进行 SQL 改写，从而达到节省带宽的目的。

10.8.3 分页方案优化

由于 LIMIT 并不能通过索引查询数据，因此如果可以保证 ID 的连续性，通过 ID 进行分页是比较好的解决方案：

```
SELECT * FROM t_order WHERE id > 100000 AND id <= 100010 ORDER BY id
```

或通过记录上次查询结果的最后一条记录的 ID 进行下一页的查询：

```
SELECT * FROM t_order WHERE id > 100000 LIMIT 10
```

10.8.4 分页子查询

Oracle 和 SQLServer 的分页都需要通过子查询来处理，DBPlusEngine 支持分页相关的子查询。

■ Oracle

支持使用 rownum 进行分页：

```
SELECT * FROM (SELECT row_.*, rownum rownum_ FROM (SELECT o.order_id as order_id FROM t_order o JOIN t_order_item i ON o.order_id = i.order_id) row_ WHERE rownum <= ?) WHERE rownum > ?
```

目前不支持 rownum + BETWEEN 的分页方式。

■ SQLServer

支持使用 TOP + ROW_NUMBER() OVER 配合进行分页：

```
SELECT * FROM (SELECT TOP (?) ROW_NUMBER() OVER (ORDER BY o.order_id DESC) AS rownum, * FROM t_order o) AS temp
WHERE temp.rownum > ? ORDER BY temp.order_id
```

支持 SQLServer 2012 之后的 OFFSET FETCH 的分页方式：

```
SELECT * FROM t_order o ORDER BY id OFFSET ? ROW FETCH NEXT ? ROWS ONLY
```

目前不支持使用 WITH xxx AS (SELECT ...) 的方式进行分页。由于 Hibernate 自动生成的 SQLServer 分页语句使用了 WITH 语句，因此目前并不支持基于 Hibernate 的 SQLServer 分页。目前也不支持使用两个 TOP + 子查询的方式实现分页。

■ MySQL, PostgreSQL

MySQL 和 PostgreSQL 都支持 LIMIT 分页，无需子查询：

```
SELECT * FROM t_order o ORDER BY id LIMIT ? OFFSET ?
```

10.9 Hint

DBPlusEngine 支持使用 Hint 的方式来影响中间层路由，以达到灵活使用 DBPlusEngine 的目的，DBPlusEngine 的 Hint 和原生数据库的 Hint 功能不同，可以看成是 DBPlusEngine 的一种特有使用方式。

10.9.1 建议

1. 分片字段不存在于 SQL 和数据库表结构中，而存在于外部业务逻辑，可以使用 Hint 方式来强制路由；
2. 业务场景存在某些场景，需要强制在主库进行操作，以保证数据强一致性的场景。

10.9.2 示例

- 读写分离 Hint 的使用示例：用户通过调用 HintManager.setWriteRouteOnly() 方法设置主库路由标记，或者使用 SQL Hint 来实现读写分离场景下的强制主库路由。SQL Hint 功能需要用户提前开启解析注释的配置，设置 sqlCommentParseEnabled 为 true。注释格式暂时只支持 /* */，内容需要以 ShardingSphere hint: 开始，属性名为 WRITE_ROUTE_ONLY。

```
/* SHARDINGSPHERE_HINT: WRITE_ROUTE_ONLY=true */
SELECT * FROM t_order;
```

然后执行 SQL 语句实现强制主库路由。

10.10 子查询

10.10.1 原理

DBPlusEngine 支持部分子查询，当子查询和外层查询同时指定分片键，且分片键的值能够路由到同一分片时，能够支持子查询。

```
SELECT * FROM (SELECT * FROM t_order WHERE order_id = 1) o WHERE o.order_id = 1;
```

此外，DBPlusEngine 可以支持用于分页的子查询。

```
SELECT * FROM (SELECT row_.*, rownum rownum_ FROM (SELECT * FROM t_order) row_ WHERE rownum <= ?) WHERE rownum > ?;
```

说明：对于不满足条件的情况，ShardingSphere 能执行但是结果错误。

10.11 左右连接

10.11.1 原理

DBPlusEngine 目前支持绑定表、广播表和单表的左右连接查询，在进行绑定表左右连接查询时，需要在连接条件中指定分片键。分片表和广播表，分片表和单表的连接查询暂不支持，需要依赖 SQL Federation 引擎提供查询能力。

关系表如下所示：

组合	关联条件	是否支持	处理引擎
分片表 + 分片表	分片键 + 绑定	是	内核
	分片键 + 非绑定	是	实验性 SQL Federation 引擎，需要手动开启
	非分片键	是	实验性 SQL Federation 引擎，需要手动开启
分片表 + 单表	分片键	是	实验性 SQL Federation 引擎，需要手动开启
	非分片键	是	实验性 SQL Federation 引擎，需要手动开启
分片 + 广播表	分片键	否	内核（存在 bug，左右连接数据重复，支持内连接）
	非分片键	否	内核（存在 bug，左右连接数据重复，支持内连接）
	不限制	是	内核
单表 + 单表	不限制	是	内核（同库场景）
	不限制	是	实验性 SQL Federation 引擎，需要手 （跨库场景）动开启

DBPlusEngine 在执行绑定表的连接查询时，会选择指定了分片条件的表作为主表进行路由改写，其他绑定表根据绑定关系进行改写，然后将改写后的真实 SQL 下推到数据库执行，由于绑定表具有相同的数据分布，并且在查询时指定了分片键作为关联条件，因此原生数据库的关联查询结果可以作为逻辑 SQL 的查询结果，最后通过归并引擎对多个分片的结果进行合并。

而在执行广播表和单表的连接查询时，DBPlusEngine 会以单表为主表进行路由，然后将 SQL 下推到数据库执行，得到关联查询的结果。

10.11.2 建议

- 1. 将多个具有相同分片规则的表配置为绑定表，以保证可以将关联查询下推至数据库执行。

10.11.3 示例

如下示例展示了 DBPlusEngine 支持的关联查询，t_order 和 t_order_item 为绑定表，t_config 为广播表，t_single 为单表。

```
-- 查询各个表的数据
mysql> select * from t_order;
+-----+-----+-----+
| order_id | user_id | content |
+-----+-----+-----+
| 2 | 2 | TEST2 |
| 4 | 1 | test11 |
| 1 | 1 | %TEST1 |
| 3 | 3 | test3 |
| 5 | 1 | test11 |
+-----+-----+-----+
5 rows in set (0.01 sec)

mysql> select * from t_order_item;
+-----+-----+-----+-----+
| item_id | order_id | content | user_id |
+-----+-----+-----+-----+
| 2 | 2 | TEST2 | 2 |
| 1 | 1 | TEST1 | 1 |
+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

```
mysql> SELECT * FROM t_config;
```

config_id	content
1	TEST1
2	TEST2
3	TEST3
4	TEST4

```
4 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM t_single;
```

single_id	content
1	TEST1
2	TEST2

```
2 rows in set (0.01 sec)
```

```
-- 绑定表左连接
```

```
mysql> select * from t_order o left join t_order_item i on o.order_id = i.order_id and o.user_id = i.user_id;
```

order_id	user_id	content	item_id	order_id	content	user_id
2	2	TEST2	2	2	TEST2	2
4	1	test11	NULL	NULL	NULL	NULL
1	1	%TEST1	1	1	TEST1	1
3	3	test3	NULL	NULL	NULL	NULL
5	1	test11	NULL	NULL	NULL	NULL

```
5 rows in set (0.01 sec)
```

```
-- 绑定表右连接
```

```
mysql> select * from t_order o right join t_order_item i on o.order_id = i.order_id and o.user_id = i.user_id;
```

order_id	user_id	content	item_id	order_id	content	user_id
2	2	TEST2	2	2	TEST2	2
1	1	%TEST1	1	1	TEST1	1

```
2 rows in set (0.01 sec)
```

```
-- 广播表、单表左连接
```

```
mysql> SELECT * FROM t_config o LEFT JOIN t_single s ON o.config_id = s.single_id;
```

config_id	content	single_id	content
1	TEST1	1	TEST1
2	TEST2	2	TEST2
3	TEST3	NULL	NULL
4	TEST4	NULL	NULL

```
4 rows in set (0.00 sec)
```

```
-- 广播表、单表右连接
```

```
mysql> SELECT * FROM t_config o RIGHT JOIN t_single s ON o.config_id = s.single_id;
```

config_id	content	single_id	content
1	TEST1	1	TEST1
2	TEST2	2	TEST2

2 rows in set (0.00 sec)

10.12 半连接和反连接

DBPlusEngine 目前支持如下形式的半连接：

```
SELECT * FROM t_order WHERE order_id IN (1, 2, 3);
```

不支持子查询中包含分片表的半连接：

说明：报错一般是因为没有改写表名，数据库找不到 table 报错；如果不需要改写的表名，能执行但是返回的结果不对。

```
SELECT * FROM t_order WHERE EXISTS (SELECT * FROM t_order_item);
```

DBPlusEngine 目前支持如下形式的反连接：

```
SELECT * FROM t_order WHERE order_id NOT IN (1, 2, 3);
```

不支持子查询中包含分片表的反连接：

说明：报错一般是因为没有改写表名，数据库找不到 table 报错；如果不需要改写的表名，能执行但是返回的结果不对。

```
SELECT * FROM t_order WHERE NOT EXISTS (SELECT * FROM t_order_item);
```

10.13 数据集成

DBPlusEngine 目前不支持导入导出相关的数据集成功能，支持 INSERT INTO VALUES 语句批量插入，以及 JDBC 接口的 BATCH API，允许用户批量插入数据。

10.14 性能优化常用策略

10.14.1 简介

DBPlusEngine 本身在代码、参数等方面已对通用场景的性能进行过大量优化，默认参数的情况下启动 DBPlusEngine 即可达到较优性能。对性能要求更高的用户，可以针对场景、环境进行进一步的优化。

10.14.2 Java 相关优化

Java 版本

目前 DBPlusEngine 最低支持 Java 8。截至本文编写，最新的 LTS（Long term support，长期支持版本）为 Java 17。新版本的 Java 相比旧版本 Java 在 JDK 类库、JIT、GC 等方面会有一定的性能提升。

JVM 参数

设置合适的 -Xmx

JVM 堆内存参数需要结合业务与环境设置，不能太小也不宜过大。堆内存太小可能会导致 GC 次数增加，堆内存过大可能会增加 GC 停顿时长，具体表现为某些 SQL 的执行延时增加。

对于 OLTP 场景，在所有 SQL 都能够下推到数据库执行的情况下，堆内存一般不超过 16 GB。

对于使用联邦查询的场景，由于部分计算需要在 DBPlusEngine 进程的内存中进行，堆内存可以根据实际数据量适当增加，避免发生 OOM。

-Xss 设置

-Xss 在一般场景下无须专门设置。以下情况可能需要考虑调整：

- 使用 jOOQ

在 jOOQ 类加载过程中，类加载相关调用栈会比较深，-Xss 过小（例如少于 512k）可能会发生 `java.lang.StackOverflowError`，一般 -Xss 不小于 1m 即可。

- SQL 特别长或参数特别多

当 SQL 较长或参数非常多时，SQL 解析可能会存在较深的调用栈。例如以下 SQL：

```
insert into some_table values (1,'foo'), (2,'bar'), ..., (100000,'baz')
```

在 values 数量非常多的情况下，SQL 解析所需的调用栈会比较深，此时如果 -Xss 过小就可能会发生 `java.lang.StackOverflowError`。

-XX:+AggressiveHeap

适用于：Java 10 及更高版本

启用该选项，JVM 会为长时间运行的内存密集型应用程序优化堆，在高并发场景对 DBPlusEngine 性能提升有帮助。

NUMA 环境相关参数

-XX:+UseNUMA

在 NUMA 环境下，使用该参数可以优化 JVM 的内存分配。由于 JVM 会启动时会检测当前是否 NUMA 环境，即使在非 NUMA 环境下启动了该参数，也不会产生任何影响。

JIT 相关参数

-XX:+SegmentedCodeCache

适用于：Java 9 及更高版本

该参数会启用分段代码缓存，对 DBPlusEngine 性能提升有帮助。

-XX:+UseJVMCICompiler

适用于：OpenJDK 11

使用 JVMCI 作为默认编译器。启用该选项对 DBPlusEngine 性能峰值有提升效果。

使用该参数要求先启用以下参数：

```
-XX:+UnlockExperimentalVMOptions
```

10.14.3 DBPlusEngine 优化

DBPlusEngine 参数优化

max-connections-size-per-query

默认值：1

该参数控制每次执行查询时，在每个数据源上允许获取的最大连接数量。当存在路由到多库/多表的查询请求时，适当增加该参数可以提高查询的并发执行度。

proxy-backend-query-fetch-size

适用于：Proxy

默认值：-1。

该参数控制 Proxy 向存储节点查询数据时，每次网络交互所传输的数据行数。

使用默认值 -1 的情况下，Proxy 向 MySQL 存储节点查询数据时会尽量使用流式查询，能够减少 Proxy 内存使用，避免在结果集非常大的情况下发生 OOM。

如果结果集行数可控，可以适当增加该参数，可能会减少 Proxy 与数据库的交互次数以减少 SQL 执行延时，但会增加 Proxy 的内存使用。

proxy-frontend-executor-size

适用于：Proxy

该参数控制 Proxy 中 Netty 的线程数量，默认值为 Proxy 所在 JVM 进程可用的逻辑 CPU 数量 * 2。

在 Proxy 承受一定负载的情况下，通过 top -H 可以查看 JVM 中线程的资源使用情况。其中，命名以 epollEventLoopGroup 为前缀的线程是 Netty 线程。

在 Proxy 吞吐量达到瓶颈的情况下：如果 Netty 线程的平均 CPU 使用率较低，可以适当减少线程数量，以减少线程切换等存在性能开销的情况；反之，则需要适当增加线程数量，以增加网络 I/O 处理能力。

DBPlusEngine 第三方依赖参数优化

关闭 Netty 泄漏检测

适用于：Proxy

Netty 具备泄漏检测能力，存在 4 种检测等级：- DISABLED：关闭。- SIMPLE：（默认）简单采样，对性能开销较小。- ADVANCED：在简单采样的基础上，同时记录泄漏资源的近期访问代码路径，开销会相对较大。- PARANOID：（仅适用于测试）检测所有资源是否存在泄漏，性能开销非常大。

Netty 泄漏检测默认是 SIMPLE 级别，性能开销较低。

但对于稳定版本的 Proxy，可以完全关闭泄漏检测以消除泄漏检测逻辑自身的性能开销。

```
-Dio.netty.leakDetection.level=DISABLED
```

10.14.4 DBPlusEngine Proxy 客户端优化

MySQL Connector/J 参数建议

启用 Server 端 Prepared Statement

参数名	MySQL Connector/J 默认值	连接 DBPlusEngine Proxy 建议值	描述
cachePrepStmts	true	true	在驱动内部缓存 Prepared Statement。
useServerPrepStmts	false	true	使用服务端 Prepared Statement。
prepStmtCacheSize	25	8192	驱动内部每个数据库连接的 Prepared Statement 缓存大小。
prepStmtCacheSqlLimit	256	2048	允许缓存的 SQL 语句长度限制。

在 DBPlusEngine Proxy 内部会对 Server 端 Prepared Statement 的语句进行 SQL 解析、SQL 上下文进行缓存，能够减少 SQL 执行的开销与延时。

TCP 缓冲区配置

默认情况下，MySQL Connector/J 的 TCP 缓冲区会使用操作系统的默认值。如果需要调整为特定的值，可以通过以下两个参数配置。- tcpRcvBuf - tcpSndBuf

合并批量操作

当使用 PreparedStatement 的 addBatch/executeBatch 等方法时，由于 MySQL 协议本身的局限性，批量操作无法直接通过 MySQL 协议实现。可以通过以下参数启用 MySQL Connector/J 的批量操作改写：- rewriteBatchedStatements=true - allowMultiQueries=false（MySQL Connector/J 默认值）

当开启批量操作改写后：- insert SQL 会自动被改写成多组 values 的形式；- update/delete 语句会自动改写成多语句的形式。

DBPlusEngine Proxy 内部会自动识别 MySQL COM_SET_OPTIONS 指令，因此使用批量操作的同时需要保持 allowMultiQueries=false 参数设置。

PostgreSQL/openGauss JDBC Driver 参数建议

TCP 缓冲区配置

默认情况下，PostgreSQL/openGauss JDBC Driver 的 TCP 缓冲区会使用操作系统的默认值。如果需要调整为特定的值，可以通过以下两个参数配置。- receiveBufferSize - sendBufferSize

10.14.5 环境优化

CPU 亲和性

在一些用户环境下，机器上某些 CPU 核心可能有专门用途。

例如在某 128 核的环境下，CPU 0-15 核被专门用于处理网络中断，此时如果 DBPlusEngine 在所有 CPU 上运行，网络处理能力、DBPlusEngine 最大吞吐量可能会受到影响。

此时可以为 DBPlusEngine 所在 JVM 执行可用的 CPU。例如，使用 numactl，指定 CPU 16-127 核可用于 DBPlusEngine，则将启动命令修改为：

```
numactl -C 16-127 java ...# 省略其他参数
```

排除网卡性能瓶颈

DBPlusEngine 属于网络 I/O 密集的应用，尤其是 Proxy，会同时与数据库客户端、存储节点交互，网络收发频繁。此时需要避免网络中断的处理成为 DBPlusEngine 的性能瓶颈，尤其在网卡队列较少的情况下，网卡中断处理达到瓶颈会限制 DBPlusEngine 的最大吞吐量。

如果网卡只有单队列，网络中断处理达到上限（通常表现是某个 CPU 核心的 hi/si 使用率接近 100%，其他 CPU 核心使用率相对较低），可以考虑通过启用 irqbalance 等方式在多个 CPU 核心分摊网络中断处理。

10.14.6 场景建议

为分片键配置索引

在数据分片场景下，分片键如果不是主键，建议增加索引，以减少通过分片键为条件的 SQL 执行延时。

分页修正与分页查询优化

对于能够路由到单库/单表的分页查询 SQL，DBPlusEngine 会直接将 SQL 下推到存储节点执行。

当分页查询的 SQL 路由到多个库或者多个表时，为确保查询的正确性，DBPlusEngine 会对语句的分页参数进行改写。尤其在大偏移量查询的情况下，经过分页修正的 SQL 查询效率会明显降低。对于可能涉及大偏移量查询的情况，建议不使用 LIMIT 方式进行分页。可以考虑构建行记录数量与行偏移量的二级索引、使用上次分页数据结尾 ID 作为下次查询条件的等分页方式，避免 SQL 执行延时过高。

避免在 SQL 中包含不需要的数据

用户在编写 SQL 时，建议精准控制所需的列、查询条件，可以有效减少 SQL 执行开销、网络 I/O 开销。与传统直接连接数据库情况相比，DBPlusEngine Proxy 的网络拓扑更为复杂，传输不必要的数据所带来的网络 I/O 开销会更明显。

配置是 DBPlusEngine-Driver 中唯一与应用开发者交互的模块，通过它可以快速清晰的理解 DBPlusEngine-Driver 所提供的功能。

本章节是 DBPlusEngine-Driver 的配置参考手册，需要时可当做字典查阅。

DBPlusEngine-Driver 提供了两种配置方式，用于不同的使用场景。通过配置，应用开发者可以灵活的使用数据分片、读写分离、数据加密、影子库等功能，并且能够叠加使用。

混合规则配置与单一规则配置一脉相承，只是从配置单一的规则项到配置多个规则项的异同。

需要注意的是，规则项之间的叠加使用是通过数据源名称和表名称关联的。如果前一个规则是面向数据源聚合的，下一个规则在配置数据源时，则需要使用前一个规则配置的聚合后的逻辑数据源名称；同理，如果前一个规则是面向表聚合的，下一个规则在配置表时，则需要使用前一个规则配置的聚合后的逻辑表名称。

11.1 License 在 DBPlusEngine-Driver 中的使用

11.1.1 背景信息

License 功能	说明
限制时长	限制 DBPlusEngine 的使用时长
限制存储节点数量	限制 DBPlusEngine 集群创建存储节点的数量
限制计算节点 (Proxy)	限制单个集群中创建创建计算节点（DBPlusEngine-Proxy）的数量
限制计算节点 (Driver)	限制单个集群中创建创建计算节点（DBPlusEngine-Driver）的数量
限制版本	限制集群中 DBPlusEngine 的版本
硬件设备的 MAC 地址	限制集群中计算节点所在的硬件设备的 MAC 地址

11.1.2 配置方式

JAVA API

引入 Maven 依赖

```
<dependency>
  <groupId>com.sphere-ex</groupId>
  <artifactId>shardingsphere-jdbc-core</artifactId>
  <version>${dbplusengine.version}</version>
</dependency>
```

配置示例

类名称：com.sphereex.dbplusengine.license.config.LicenseRuleConfiguration

可配置属性：

名称	数据类型	说明
license	String	license 注册码

YAML

引入 Maven 依赖

```
<dependency>
  <groupId>com.sphere-ex</groupId>
  <artifactId>shardingsphere-jdbc-core</artifactId>
  <version>${dbplusengine.version}</version>
</dependency>
```

配置示例

```
databaseName:
mode:
dataSources:
rules:
license: xxx
```

SpringBoot

引入 Maven 依赖

```
<dependency>
  <groupId>com.sphere-ex</groupId>
  <artifactId>shardingsphere-jdbc-core-spring-boot-starter</artifactId>
  <version>${dbplusengine.version}</version>
</dependency>
```

配置示例

```
spring.dbplusengine.rules.license: xxx
```

Spring 命名空间

引入 Maven 依赖

```
<dependency>
  <groupId>com.sphere-ex</groupId>
  <artifactId>shardingsphere-jdbc-core-spring-namespace</artifactId>
  <version>${dbplusengine.version}</version>
</dependency>
```

配置示例

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:license="https://www.sphere-ex.com/schema/dbplusengine/license"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    https://www.sphere-ex.com/schema/dbplusengine/license
    https://www.sphere-ex.com/schema/dbplusengine/license/license.xsd"
  >
  <license:license id="license" license="xxx"/>
</beans>
```

11.1.3 后续处理

如果出现如下错误请获取最新版 license 并更新 license 至最新状态。

- 无 license 提示

FATAL: License **not** registered

- license 不完整提示

FATAL: Incomplete license

- license 过期提示

FATAL: License **is** expired

- 非法 license 提示

ERROR: The xxx exceeds the limit of the license

11.1.4 无 License 或 License 超限后的表现

- 可以启动集群无法执行 RDL、RUL 和 RAL，可以执行少量 RQL
- 全局放行的 DistSQL
 1. SHOW STORAGE UNITS
 2. SHOW *
 3. COUNT * RULES
 4. EXPORT DATABASE CONFIGURATION
 5. CONVERT YAML CONFIGURATION

6. REGISTER LICENSE

7. 事务相关的标准语句

- 实例数量超限可执行

1. SET INSTANCE STATUS

- DB 的 SQL 语句不能执行

11.1.5 License 过期提醒功能

当 License 距离过期天数 ≤ 30 天时在集群的计算节点运行日志中会输出 WARN 级别的提示。

提示内容为：

```
The license of DBPlusEngine will expire in xx day(s)
```

11.2 Java API

11.2.1 简介

Java API 是 DBPlusEngine-Driver 中所有配置方式的基础，其他配置最终都将转化成为 Java API 的配置方式。

Java API 是最繁琐也是最灵活的配置方式，适合需要通过编程进行动态配置的场景下使用。

11.2.2 使用步骤

引入 Maven 依赖

```
<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-jdbc-core</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>
```

构建数据源

DBPlusEngine-Driver 的 Java API 通过 Schema 名称、运行模式、数据源集合、规则集合以及属性配置组成。

通过 ShardingSphereDataSourceFactory 工厂创建的 ShardingSphereDataSource 实现自 JDBC 的标准接口 DataSource。

```
String schemaName = "foo_schema"; // 指定逻辑 Schema 名称
ModeConfiguration modeConfig = ... // 构建运行模式
Map<String, DataSource> dataSourceMap = ... // 构建真实数据源
Collection<RuleConfiguration> ruleConfigs = ... // 构建具体规则
Properties props = ... // 构建属性配置
DataSource dataSource = ShardingSphereDataSourceFactory.createDataSource(schemaName, modeConfig, dataSourceMap,
ruleConfigs, props);
```

模式详情请参见[模式配置](#)。

数据源详情请参见[数据源配置](#)。

规则详情请参见[规则配置](#)。

使用数据源

可通过 DataSource 选择使用原生 JDBC，或 JPA、Hibernate、MyBatis 等 ORM 框架。
以原生 JDBC 使用方式为例：

```
// 创建 ShardingSphereDataSource
DataSource dataSource = ShardingSphereDataSourceFactory.createDataSource(schemaName, modeConfig, dataSourceMap,
ruleConfigs, props);

String sql = "SELECT i.* FROM t_order o JOIN t_order_item i ON o.order_id=i.order_id WHERE o.user_id=? AND o.order_id=?";
try (
    Connection conn = dataSource.getConnection();
    PreparedStatement ps = conn.prepareStatement(sql)) {
    ps.setInt(1, 10);
    ps.setInt(2, 1000);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while(rs.next()) {
            // ...
        }
    }
}
```

11.2.3 模式配置

配置入口

类名称：org.apache.shardingsphere.infra.config.mode.ModeConfiguration
可配置属性：

名称	数据类型	说明	默认值
type	String	运行模式类型可选配置：Memory、Standalone、Cluster	Memory
repository	PersistRepositoryConfiguration	持久化仓库配置 Memory 类型无需持久化，可以为 nullStandalone 类型使用 StandalonePersistRepositoryConfigurationCluster 类型使用 ClusterPersistRepositoryConfiguration	

Cluster 持久化配置

类名称：org.apache.shardingsphere.mode.repository.cluster.ClusterPersistRepositoryConfiguration
可配置属性：

名称	数据类型	说明
type	String	持久化仓库类型
namespace	String	注册中心命名空间
serverLists	String	注册中心连接地址
props	Properties	持久化仓库所需属性

持久化仓库类型的详情，请参见[内置持久化仓库类型列表](#)。

11.2.4 数据源配置

DBPlusEngine-Driver 支持所有的数据库 JDBC 驱动和连接池。

配置示例

示例的数据库驱动为 MySQL，连接池为 HikariCP，可以更换为其他数据库驱动和连接池。

```
Map<String, DataSource> dataSourceMap = new HashMap<>();
```

```
// 配置第 1 个数据源
```

```
HikariDataSource dataSource1 = new HikariDataSource();
dataSource1.setDriverClassName("com.mysql.jdbc.Driver");
dataSource1.setJdbcUrl("jdbc:mysql://localhost:3306/ds_1");
dataSource1.setUsername("root");
dataSource1.setPassword("");
dataSourceMap.put("ds_1", dataSource1);
```

```
// 配置第 2 个数据源
```

```
HikariDataSource dataSource2 = new HikariDataSource();
dataSource2.setDriverClassName("com.mysql.jdbc.Driver");
dataSource2.setJdbcUrl("jdbc:mysql://localhost:3306/ds_2");
dataSource2.setUsername("root");
dataSource2.setPassword("");
dataSourceMap.put("ds_2", dataSource2);
```

```
// 配置其他数据源
```

```
...
```

11.2.5 规则配置

规则是 DBPlusEngine 面向可插拔的一部分。本章节是 DBPlusEngine-Driver 的 Java 规则配置参考手册。

数据分片

配置入口

类名称：org.apache.shardingsphere.sharding.api.config.ShardingRuleConfiguration

可配置属性：

名称	数据类型	说明	默认值
tables (+)	Collection<ShardingTableRuleConfiguration>	分片表规则列表	■
autoTables (+)	Collection<ShardingAutoTableRuleConfiguration>	自动化分片表规则列表	■
bindingTableGroups (*)	Collection<String>	绑定表规则列表	无
broadcastTables (*)	Collection<String>	广播表规则列表	无
defaultDatabaseShardingStrategy (?)	ShardingStrategyConfiguration	默认分库策略	不分片
defaultTableShardingStrategy (?)	ShardingStrategyConfiguration	默认分表策略	不分片
defaultKeyGenerateStrategy (?)	KeyGeneratorConfiguration	默认自增列生成器配置	雪花算法
defaultShardingColumn (?)	String	默认分片列名称	无
shardingAlgorithms (+)	Map<String, ShardingSphereAlgorithmConfiguration>	分片算法名称和配置	无
keyGenerators (?)	Map<String, ShardingSphereAlgorithmConfiguration>	自增列生成算法名称和配置	无

分片表配置

类名称：org.apache.shardingsphere.sharding.api.config.ShardingTableRuleConfiguration

可配置属性：

名称	数据类型	说明	默认值
logicTable	String	分片逻辑表名称	■
actualDataNodes (?)	String	由数据源名 + 表名组成，以小数点分隔。多个表以逗号分隔，支持行表达式	使用已知数据源与逻辑表名称生成数据节点，用于广播表或只分库不分表且所有库的表结构完全一致的情况
databaseShardingStrategy (?)	ShardingStrategyConfiguration	分库策略	使用默认分库策略
tableShardingStrategy (?)	ShardingStrategyConfiguration	分表策略	使用默认分表策略
keyGenerateStrategy (?)	KeyGeneratorConfiguration	自增列生成器	使用默认自增主键生成器

自动分片表配置

类名称：org.apache.shardingsphere.sharding.api.config.ShardingAutoTableRuleConfiguration

可配置属性：

名称	数据类型	说明	默认值
logicTable	String	分片逻辑表名称	■
actualDataSources (?)	String	数据源名称，多个数据源以逗号分隔	使用全部配置的数据源
shardingStrategy (?)	ShardingStrategyConfiguration	分片策略	使用默认分片策略
keyGenerateStrategy (?)	KeyGeneratorConfiguration	自增列生成器	使用默认自增主键生成器

分片策略配置

标准分片策略配置

类名称：org.apache.shardingsphere.sharding.api.config.strategy.sharding.StandardShardingStrategyConfiguration
可配置属性：

名称	数据类型	说明
shardingColumn	String	分片列名称
shardingAlgorithmName	String	分片算法名称

复合分片策略配置

类名称：org.apache.shardingsphere.sharding.api.config.strategy.sharding.ComplexShardingStrategyConfiguration
可配置属性：

名称	数据类型	说明
shardingColumns	String	分片列名称，多个列以逗号分隔
shardingAlgorithmName	String	分片算法名称

Hint 分片策略配置

类名称：org.apache.shardingsphere.sharding.api.config.strategy.sharding.HintShardingStrategyConfiguration
可配置属性：

名称	数据类型	说明
shardingAlgorithmName	String	分片算法名称

不分片策略配置

类名称：org.apache.shardingsphere.sharding.api.config.strategy.sharding.NoneShardingStrategyConfiguration
可配置属性：无
算法类型的详情，请参见[内置分片算法列表](#)。

分布式序列策略配置

类名称：org.apache.shardingsphere.sharding.api.config.strategy.keygen.KeyGenerateStrategyConfiguration

可配置属性：

名称	数据类型	说明
column	String	分布式序列列名称
keyGeneratorName	String	分布式序列算法名称

算法类型的详情，请参见[内置分布式序列算法列表](#)。

读写分离

配置入口

类名称：org.apache.shardingsphere.readwritesplitting.api.ReadwriteSplittingRuleConfiguration

可配置属性：

名称	数据类型	说明
dataSources (+)	Collection< ReadwriteSplittingData SourceRuleConfiguration>	读写数据源配置
loadBalancers (*)	Map<String, ShardingSphere AlgorithmConfiguration>	从库负载均衡算法配置

主从数据源配置

类名称：org.apache.shardingsphere.readwritesplitting.api.rule.ReadwriteSplittingDataSourceRuleConfiguration

可配置属性：

名称	数据类型	说明	默认值
name	String	读写分离数据源名称	■
staticStrategy	Sta ticReadwriteSp littingStrateg yConfiguration	静态读写分离配置	■
d ynamicStrategy	Dyna micReadwriteSp littingStrateg yConfiguration	动态读写分离配置	■
lo adBalancerName (?)	String	读库负载均衡算法名称	轮询负载均衡算法

类名称：org.apache.shardingsphere.readwritesplitting.api.strategy.StaticReadwriteSplittingStrategyConfiguration

可配置属性：

名称	数据类型	说明
writeDataSourceName	String	写库数据源名称
readDataSourceNames	List<String>	读库数据源列表

类名称：org.apache.shardingsphere.readwritesplitting.api.strategy.DynamicReadwriteSplittingStrategyConfiguration

可配置属性：

名称	数据类型	说明	默认值
auto AwareDataSource-Name	String	数据库发现的逻辑数据源名称	■
writeDat aSource-QueryEnabled (?)	String	读库全部下线，主库是否承担读流量	true

算法类型的详情，请参见[内置负载均衡算法列表](#)。

数据库发现

配置入口

类名称：org.apache.shardingsphere.dbdiscovery.api.config.DatabaseDiscoveryRuleConfiguration

可配置属性：

名称	数据类型	说明
dataSources (+)	Collecti on<DatabaseDisco veryDataSourceRu leConfiguration>	数据源配置
discoveryHeartbeats (+)	Map<String, Database DiscoveryHeartBe atConfiguration>	监听心跳配置
discoveryTypes (+)	Map<String, Shard ingSphereAlgorit hmConfiguration>	数据库发现类型配置

数据源配置

类名称：org.apache.shardingsphere.dbdiscovery.api.config.rule.DatabaseDiscoveryDataSourceRuleConfiguration

可配置属性：

名称	数据类型	说明	默认值
groupName (+)	String	数据库发现组名称	■
dataSourceNames (+)	Co llection<String>	数据源名称，多个数据源用逗号分隔如：ds_0, ds_1	■
discov eryHeartbeatName (+)	String	监听心跳名称	■
d iscoveryTypeName (+)	String	数据库发现类型名称	■

监听心跳配置

类名称：org.apache.shardingsphere.dbdiscovery.api.config.rule.DatabaseDiscoveryHeartBeatConfiguration

可配置属性：

名称	数据类型	说明	默认值
props (+)	P roperties	监听心跳属性配置，keep-alive-cron 属性配置 cron 表达式，如：‘0/5 * * * * ?’	■

数据库发现类型配置

类名称：org.apache.shardingsphere.infra.config.algorithm.ShardingSphereAlgorithmConfiguration

名称	数据类型	说明	默认值
type (+)	String	数据库发现类型，如：MGR、openGauss	■
props (?)	Properties	数据库发现类型配置，如MGR 的 group-name 属性配置	■

数据加密

配置入口

类名称：org.apache.shardingsphere.encrypt.api.config.EncryptRuleConfiguration

可配置属性：

名称	数据类型	说明	默认值
tables (+)	Collection<EncryptTableRuleConfiguration>	加密表规则配置	
encryptors (+)	Map<String, ShardingSphereAlgorithmConfiguration>	加解密算法名称和配置	
queryWithCipherColumn (?)	boolean	是否使用加密列进行查询。在有原文列的情况下，可以使用原文列进行查询	true

加密表规则配置

类名称：org.apache.shardingsphere.encrypt.api.config.rule.EncryptTableRuleConfiguration

可配置属性：

名称	数据类型	说明
name	String	表名称
columns (+)	Collection<EncryptColumnRuleConfiguration>	加密列规则配置列表
queryWithCipherColumn (?)	boolean	该表是否使用加密列进行查询

加密列规则配置

类名称：org.apache.shardingsphere.encrypt.api.config.rule.EncryptColumnRuleConfiguration

可配置属性：

名称	数据类型	说明
logicColumn	String	逻辑列名称，必填
dataType	String	逻辑列类型，选填（如果配置了其它加密列、辅助查询列或者明文列类型时，必填）
cipherColumn	String	加密列名称，必填
cipherDataType	String	加密列类型，选填（如果配置了逻辑列类型，必填）
assistedQueryColumn	String	辅助查询列名称，选填
assistedQuery-DataType	String	辅助查询列类型，选填（如果配置了辅助查询列和逻辑列类型时，必填）
plainColumn	String	明文列名称，选填
plainDataType	String	明文列类型，选填（如果配置了明文列和逻辑列类型时，必填）
encryptorName	String	加密算法名称，必填

加解密算法配置

类名称：org.apache.shardingsphere.infra.config.algorithm.ShardingSphereAlgorithmConfiguration

可配置属性：

名称	数据类型	说明
name	String	加解密算法名称
type	String	加解密算法类型
properties	Properties	加解密算法属性配置

算法类型的详情，请参见[内置加密算法列表](#)。

影子库

配置入口

类名称：org.apache.shardingsphere.shadow.api.config.ShadowRuleConfiguration

可配置属性：

名称	数据类型	说明	默认值
dataSources	Map<String, ShadowDataSourceConfiguration>	影子数据源映射名称和配置	
tables	Map<String, ShadowTableConfiguration>	影子表名称和配置	
defaultShadowAlgorithm-Name	String	默认影子算法名称	
shadowAlgorithms	Map<String, ShardingSphereAlgorithmConfiguration>	影子算法名称和配置	无

影子数据源配置

类名称：org.apache.shardingsphere.shadow.api.config.datasource.ShadowDataSourceConfiguration

可配置属性：

名称	数据类型	说明
sourceDataSourceName	String	生产数据源名称
shadowDataSourceName	String	影子数据源名称

影子表配置

类名称：org.apache.shardingsphere.shadow.api.config.table.ShadowTableConfiguration

可配置属性：

名称	数据类型	说明
dataSourceNames	Collection<String>	影子表关联影子数据源映射名称列表
shadowAlgorithmNames	Collection<String>	影子表关联影子算法名称列表

影子算法配置

算法类型的详情，请参见[内置影子算法列表](#)。

SQL 解析

配置入口

类名称：org.apache.shardingsphere.parser.config.SQLParserRuleConfiguration

可配置属性：

名称	数据类型	说明
sqlCommentParseEnabled (?)	boolean	是否使用字符串截取的方式解析 SQL 注释
parseTreeCache (?)	CacheOption	解析语法树本地缓存配置
sqlStatementCache (?)	CacheOption	sql 语句本地缓存配置

本地缓存配置

类名称：org.apache.shardingsphere.sql.parser.api.CacheOption

可配置属性：

名称	数据类型	说明	默认值
initialCapacity	int	本地缓存初始容量	语法树本地缓存默认值 128，sql 语句缓存默认值 2000
maximumSize	long	本地缓存最大容量	语法树本地缓存默认值 1024，sql 语句缓存默认值 65535
concurrencyLevel	int	本地缓存并发级别，最多允许线程并发更新的个数	4

数据脱敏

背景信息

数据脱敏 Java API 规则配置允许用户直接通过编写 Java 代码的方式，完成对象的创建，Java API 的配置方式非常灵活，不需要依赖额外的 jar 包就能够集成各种类型的业务系统。

参数解释

配置入口

类名称：org.apache.shardingsphere.mask.api.config.MaskRuleConfiguration

可配置属性：

名称	数据类型	说明	默认值
tables (+)	Collection<MaskTableRuleConfiguration>	脱敏表规则配置	
maskAlgorithms (+)	Map<String, AlgorithmConfiguration>	脱敏算法名称和配置	

脱敏表规则配置

类名称：org.apache.shardingsphere.mask.api.config.rule.MaskTableRuleConfiguration

可配置属性：

名称	数据类型	说明
name	String	表名称
columns (+)	Collection<MaskColumnRuleConfiguration>	脱敏列规则配置列表

脱敏列规则配置

类名称：org.apache.shardingsphere.mask.api.config.rule.MaskColumnRuleConfiguration

可配置属性：

名称	数据类型	说明
logicColumn	String	逻辑列名称
maskAlgorithm	String	脱敏算法名称

脱敏算法配置

类名称：org.apache.shardingsphere.infra.config.algorithm.AlgorithmConfiguration

可配置属性：

名称	数据类型	说明
name	String	脱敏算法名称
type	String	脱敏算法类型
properties	Properties	脱敏算法属性配置

算法类型的详情，请参见[内置脱敏算法列表](#)。

操作步骤

1. 创建真实数据源映射关系，key 为数据源逻辑名称，value 为 DataSource 对象；
2. 创建脱敏规则对象 MaskRuleConfiguration，并初始化对象中的脱敏表对象 MaskTableRuleConfiguration、脱敏算法等参数；

配置示例

```
import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.Properties;

public final class MaskDatabasesConfiguration implements ExampleConfiguration {

    @Override
    public DataSource getDataSource() {
        MaskColumnRuleConfiguration passwordColumn = new MaskColumnRuleConfiguration("password", "md5_mask");
        MaskColumnRuleConfiguration emailColumn = new MaskColumnRuleConfiguration("email", "mask_before_special_chars_mask");
        MaskColumnRuleConfiguration telephoneColumn = new MaskColumnRuleConfiguration("telephone", "keep_first_n_last_m_mask");
        MaskTableRuleConfiguration maskTableRuleConfig = new MaskTableRuleConfiguration("t_user", Arrays.asList(passwordColumn, emailColumn, telephoneColumn));
        Map<String, AlgorithmConfiguration> maskAlgorithmConfigs = new LinkedHashMap<>(3, 1);
        maskAlgorithmConfigs.put("md5_mask", new AlgorithmConfiguration("MD5", new Properties()));
        Properties beforeSpecialCharsProps = new Properties();
        beforeSpecialCharsProps.put("special-chars", "@");
        beforeSpecialCharsProps.put("replace-char", "*");
        maskAlgorithmConfigs.put("mask_before_special_chars_mask", new AlgorithmConfiguration("MASK_BEFORE_SPECIAL_CHARS", beforeSpecialCharsProps));
        Properties keepFirstNLastMProps = new Properties();
        keepFirstNLastMProps.put("first-n", "3");
        keepFirstNLastMProps.put("last-m", "4");
        keepFirstNLastMProps.put("replace-char", "*");
        maskAlgorithmConfigs.put("keep_first_n_last_m_mask", new AlgorithmConfiguration("KEEP_FIRST_N_LAST_M", keepFirstNLastMProps));
        MaskRuleConfiguration maskRuleConfig = new MaskRuleConfiguration(Collections.singleton(maskTableRuleConfig), maskAlgorithmConfigs);
        try {
            return ShardingSphereDataSourceFactory.createDataSource(DataSourceUtil.createDataSource("demo_ds"), Collections.singleton(maskRuleConfig), new Properties());
        } catch (final SQLException ex) {
            ex.printStackTrace();
            return null;
        }
    }
}
```


混合规则

混合配置的规则项之间的叠加使用是通过数据源名称和表名称关联的。

如果前一个规则是面向数据源聚合的，下一个规则在配置数据源时，则需要使用前一个规则配置的聚合后的逻辑数据源名称；同理，如果前一个规则是面向表聚合的，下一个规则在配置表时，则需要使用前一个规则配置的聚合后的逻辑表名称。

配置项说明

```
/* 数据源配置 */
HikariDataSource writeDataSource0 = new HikariDataSource();
writeDataSource0.setDriverClassName("com.mysql.jdbc.Driver");
writeDataSource0.setJdbcUrl("jdbc:mysql://localhost:3306/db0?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8");
writeDataSource0.setUsername("root");
writeDataSource0.setPassword("");

HikariDataSource writeDataSource1 = new HikariDataSource();
// ... 忽略其他数据库配置项

HikariDataSource read0OfwriteDataSource0 = new HikariDataSource();
// ... 忽略其他数据库配置项

HikariDataSource read1OfwriteDataSource0 = new HikariDataSource();
// ... 忽略其他数据库配置项

HikariDataSource read0OfwriteDataSource1 = new HikariDataSource();
// ... 忽略其他数据库配置项

HikariDataSource read1OfwriteDataSource1 = new HikariDataSource();
// ... 忽略其他数据库配置项

Map<String, DataSource> datasourceMaps = new HashMap<>(6);

datasourceMaps.put("write_ds0", writeDataSource0);
datasourceMaps.put("write_ds0_read0", read0OfwriteDataSource0);
datasourceMaps.put("write_ds0_read1", read1OfwriteDataSource0);

datasourceMaps.put("write_ds1", writeDataSource1);
datasourceMaps.put("write_ds1_read0", read0OfwriteDataSource1);
datasourceMaps.put("write_ds1_read1", read1OfwriteDataSource1);

/* 分片规则配置 */
// 表达式 ds_${0..1} 枚举值表示的是主从配置的逻辑数据源名称列表
ShardingTableRuleConfiguration tOrderRuleConfiguration = new ShardingTableRuleConfiguration("t_order", "ds_${0..1}.t_order_${[0, 1]}");
tOrderRuleConfiguration.setKeyGenerateStrategy(new KeyGenerateStrategyConfiguration("order_id", "snowflake"));
tOrderRuleConfiguration.setTableShardingStrategy(new StandardShardingStrategyConfiguration("order_id", "tOrderInlineShardingAlgorithm"));
Properties tOrderShardingInlineProps = new Properties();
tOrderShardingInlineProps.setProperty("algorithm-expression", "t_order_${order_id % 2}");
tOrderRuleConfiguration.getShardingAlgorithms().putIfAbsent("tOrderInlineShardingAlgorithm", new ShardingSphereAlgorithmConfiguration("INLINE", tOrderShardingInlineProps));

ShardingTableRuleConfiguration tOrderItemRuleConfiguration = new ShardingTableRuleConfiguration("t_order_item", "ds_${0..1}.t_order_item_${[0, 1]}");
tOrderItemRuleConfiguration.setKeyGenerateStrategy(new KeyGenerateStrategyConfiguration("order_item_id", "snowflake"));
tOrderRuleConfiguration.setTableShardingStrategy(new StandardShardingStrategyConfiguration("order_item_id", "tOrderItemInlineShardingAlgorithm"));
Properties tOrderItemShardingInlineProps = new Properties();
tOrderItemShardingInlineProps.setProperty("algorithm-expression", "t_order_item_${order_item_id % 2}");
tOrderRuleConfiguration.getShardingAlgorithms().putIfAbsent("tOrderItemInlineShardingAlgorithm", new ShardingSphereAlgorithmConfiguration("INLINE", tOrderItemShardingInlineProps));
```

```
ShardingRuleConfiguration shardingRuleConfiguration = new ShardingRuleConfiguration();
shardingRuleConfiguration.getTables().add(tOrderRuleConfiguration);
shardingRuleConfiguration.getTables().add(tOrderItemRuleConfiguration);
shardingRuleConfiguration.getBindingTableGroups().add("t_order, t_order_item");
shardingRuleConfiguration.getBroadcastTables().add("t_bank");
// 默认分库策略
shardingRuleConfiguration.setDefaultDatabaseShardingStrategy(new StandardShardingStrategyConfiguration("user_id",
"default_db_strategy_inline"));
Properties defaultDatabaseStrategyInlineProps = new Properties();
defaultDatabaseStrategyInlineProps.setProperty("algorithm-expression", "ds_${user_id % 2}");
shardingRuleConfiguration.getShardingAlgorithms().put("default_db_strategy_inline", new
ShardingSphereAlgorithmConfiguration("INLINE", defaultDatabaseStrategyInlineProps));
// 分布式序列算法配置
Properties snowflakeProperties = new Properties();
shardingRuleConfiguration.getKeyGenerators().put("snowflake", new ShardingSphereAlgorithmConfiguration("SNOWFLAKE",
snowflakeProperties));

/* 数据加密规则配置 */
Properties encryptProperties = new Properties();
encryptProperties.setProperty("aes-key-value", "123456");
EncryptColumnRuleConfiguration columnConfigAes = new EncryptColumnRuleConfiguration("username", "username", "",
"username_plain", "name_encryptor");
EncryptColumnRuleConfiguration columnConfigTest = new EncryptColumnRuleConfiguration("pwd", "pwd", "assisted_query_
pwd", "", "pwd_encryptor");
EncryptTableRuleConfiguration encryptTableRuleConfig = new EncryptTableRuleConfiguration("t_user", Arrays.
asList(columnConfigAes, columnConfigTest));

Map<String, ShardingSphereAlgorithmConfiguration> encryptAlgorithmConfigs = new LinkedHashMap<>(2, 1);
encryptAlgorithmConfigs.put("name_encryptor", new ShardingSphereAlgorithmConfiguration("AES", encryptProperties));
encryptAlgorithmConfigs.put("pwd_encryptor", new ShardingSphereAlgorithmConfiguration("assistedTest",
encryptProperties));
EncryptRuleConfiguration encryptRuleConfiguration = new EncryptRuleConfiguration(Collections.
singleton(encryptTableRuleConfig), encryptAlgorithmConfigs);

/* 读写分离规则配置 */
Properties readwriteProps1 = new Properties();
readwriteProps1.setProperty("write-data-source-name", "write_ds0");
readwriteProps1.setProperty("read-data-source-names", "write_ds0_read0, write_ds0_read1");
ReadwriteSplittingDataSourceRuleConfiguration dataSourceConfiguration1 = new
ReadwriteSplittingDataSourceRuleConfiguration("ds_0", "Static", readwriteProps1, "roundRobin");
Properties readwriteProps2 = new Properties();
readwriteProps2.setProperty("write-data-source-name", "write_ds0");
readwriteProps2.setProperty("read-data-source-names", "write_ds1_read0, write_ds1_read1");
ReadwriteSplittingDataSourceRuleConfiguration dataSourceConfiguration2 = new
ReadwriteSplittingDataSourceRuleConfiguration("ds_1", "Static", readwriteProps2, "roundRobin");

// 负载均衡算法
Map<String, ShardingSphereAlgorithmConfiguration> loadBalanceMaps = new HashMap<>(1);
loadBalanceMaps.put("roundRobin", new ShardingSphereAlgorithmConfiguration("ROUND_ROBIN", new Properties()));

ReadwriteSplittingRuleConfiguration readWriteSplittingyRuleConfiguration = new ReadwriteSplittingRuleConfiguration(Arrays.
asList(dataSourceConfiguration1, dataSourceConfiguration2), loadBalanceMaps);

/* 其他配置 */
Properties otherProperties = new Properties();
otherProperties.setProperty("sql-show", "true");

/* shardingDataSource 就是最终被 ORM 框架或其他 jdbc 框架引用的数据源名称 */
DataSource shardingDataSource = ShardingSphereDataSourceFactory.createDataSource(datasourceMaps, Arrays.
asList(shardingRuleConfiguration, readWriteSplittingyRuleConfiguration, encryptRuleConfiguration), otherProperties);
```

11.3 JDBC 驱动

11.3.1 简介

DBPlusEngine-Driver 提供了 JDBC 驱动，可以仅通过配置变更即可使用，无需改写代码。

11.3.2 使用步骤

引入 Maven 依赖

```
<dependency>
  <groupId>com.sphere-ex</groupId>
  <artifactId>shardingsphere-jdbc-core</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>
```

使用驱动

使用原生驱动

```
Class.forName("org.apache.shardingsphere.driver.ShardingSphereDriver");
String jdbcUrl = "jdbc:shardingsphere:classpath:config.yaml";

String sql = "SELECT i.* FROM t_order o JOIN t_order_item i ON o.order_id=i.order_id WHERE o.user_id=? AND o.order_id=?";
try (
    Connection conn = DriverManager.getConnection(jdbcUrl);
    PreparedStatement ps = conn.prepareStatement(sql)) {
    ps.setInt(1, 10);
    ps.setInt(2, 1000);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while(rs.next()) {
            // ...
        }
    }
}
```

使用数据库连接池

```
String driverClassName = "org.apache.shardingsphere.driver.ShardingSphereDriver";
String jdbcUrl = "jdbc:shardingsphere:classpath:config.yaml";

// 以 HikariCP 为例
HikariDataSource dataSource = new HikariDataSource();
dataSource.setDriverClassName(driverClassName);
dataSource.setJdbcUrl(jdbcUrl);

String sql = "SELECT i.* FROM t_order o JOIN t_order_item i ON o.order_id=i.order_id WHERE o.user_id=? AND o.order_id=?";
try (
    Connection conn = dataSource.getConnection();
    PreparedStatement ps = conn.prepareStatement(sql)) {
    ps.setInt(1, 10);
    ps.setInt(2, 1000);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while(rs.next()) {
            // ...
        }
    }
}
```

```
}  
}  
}
```

配置说明

驱动类名称

org.apache.shardingsphere.driver.ShardingSphereDriver

URL 配置说明

- 以 jdbc:shardingsphere: 为前缀
- 配置文件: xxx.yaml, 配置文件格式与 [YAML 配置](#) 一致
- 配置文件加载规则:
 - 无前缀表示从绝对路径加载配置文件
 - classpath: 前缀表示从类路径中加载配置文件

+++ title = "Spring Boot" weight = 1 chapter = true +++

11.3.3 简介

DBplusEngine-Driver 提供 JDBC 驱动, 开发者可以在 Spring Boot 中配置 ShardingSphereDriver 来使用 DBplusEngine-Driver。

11.3.4 使用步骤

引入 Maven 依赖

```
<dependency>  
  <groupId>com.sphere-ex</groupId>  
  <artifactId>shardingsphere-jdbc-core</artifactId>  
  <version>${shardingsphere.version}</version>  
</dependency>
```

配置 Spring Boot

```
# 配置 DataSource Driver  
spring.datasource.driver-class-name=org.apache.shardingsphere.driver.ShardingSphereDriver  
# 指定 YAML 配置文件  
spring.datasource.url=jdbc:shardingsphere:classpath:xxx.yaml
```

使用数据源

直接使用该数据源；或者将 ShardingSphereDataSource 配置在 JPA、Hibernate、MyBatis 等 ORM 框架中配合使用。

+++ title = “Spring 命名空间” weight = 2 chapter = true +++

11.3.5 简介

DBPlusEngine-Driver 提供 JDBC 驱动，开发者可以在 Spring 中配置 ShardingSphereDriver 来使用 DBPlusEngine-Driver。

11.3.6 使用步骤

引入 Maven 依赖

```
<dependency>
  <groupId>com.sphere-ex</groupId>
  <artifactId>shardingsphere-jdbc-core</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>
```

配置 Spring Bean

配置项说明

名称	类型	说明
driverClass	属性	数据库 Driver，这里需要指定使用 ShardingSphereDriver
url	属性	YAML 配置文件路径

配置示例

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="shardingDataSource" class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
    <property name="driverClass" value="org.apache.shardingsphere.driver.ShardingSphereDriver" />
    <property name="url" value="jdbc:shardingsphere:classpath:xxx.yaml" />
  </bean>
</beans>
```

使用数据源

使用方式同 Spring Boot。

11.4 属性配置

DBPlusEngine 提供属性配置的方式配置系统级配置。

11.4.1 配置项说明

名称	数据类型	说明	默认值
sql-show (?)	boolean	是否在日志中打印 SQL 打印 SQL 可以帮助开发者快速定位系统问题。日志内容包含：逻辑 SQL，真实 SQL 和 SQL 解析结果。如果开启配置，日志将使用 Topic ShardingSphere-SQL，日志级别是 INFO	false
sql-simple (?)	boolean	是否在日志中打印简单风格的 SQL	false
kernel-executor-size (?)	int	用于设置任务处理线程池的大小每个 ShardingSphereDataSource 使用一个独立的线程池，同一个 JVM 的不同数据源不共享线程池	infinite
max-connections-size-per-query (?)	int	一次查询请求在每个数据库实例中所能使用的最大连接数	1
check-table-metadata-enabled (?)	boolean	在程序启动和更新时，是否检查分片元数据的结构一致性	false
sql-federation-enabled (?)	String	联邦查询执行器类型，包括：NONE，ORIGINAL，ADVANCED	NONE

11.4.2 操作步骤

属性配置直接配置在 Driver 所使用的配置文件中，格式如下：

```
props:
  sql-show: true
```

11.5 DBPlusEngine-Driver 使用 Agent 实现对日志采集功能

11.5.1 慢查询日志

背景信息

慢查询日志功能，用于将执行耗时超过一定时间的 SQL 语句进行记录，便于 DBA 和开发人员识别可能存在问题的 SQL 语句，是数据库和 SQL 管理的重要参考来源。

参数解释

- slow-query-log：是否开启慢查询日志功能，默认值为 true。
- long-query-time：慢查询时间阈值，执行耗时超过该阈值的 SQL 语句才会被记入慢查询日志中。该配置单位为毫秒 (ms)，默认值为 5000。

前提条件

- Agent 使用 SLF4J 进行日志桥接输出，所以要求应用程序存在 SLF4J 依赖，并有相关日志输出配置。
- 慢查询日志功能基于 Agent 技术实现，需要应用程序在启动时配置 javaagent 开启 Agent。

配置示例 - Agent 配置

conf/agent.yaml 为 Agent 配置文件，默认配置如下。

```
plugins:
  logging:
    BaseLogging:
      props:
        slow-query-log: true
        long-query-time: 5000
```

其中，slow-query-log 和 long-query-time 配置的是默认值，表示的意思是：

1. 开启慢查询日志；
2. 当 SQL 执行耗时超过 5000 毫秒时，记录慢查询日志。

■ 应用程序日志配置

以常用的 logback 作为日志输出为例，配置如下。

```
<configuration>
  <property name="log.context.name" value="project-using-DBPlusEngine-Driver" />
  <property name="log.charset" value="UTF-8" />
  <property name="log.pattern" value="[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %logger{36} - %msg%n" />
  <contextName>${log.context.name}</contextName>

  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder charset="${log.charset}">
      <pattern>${log.pattern}</pattern>
    </encoder>
  </appender>

  <logger name="SLOW-QUERY" level="info" additivity="false">
    <appender-ref ref="STDOUT" />
  </logger>

  <root>
    <level value="INFO" />
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

说明：慢日志输出的 logger name 为 SLOW-QUERY

慢查询日志格式

db: {database name} query_time: {query time} sql_type: {sql type}

{sql}

- db：数据库名称；
- query_time：SQL 执行耗时，单位为 ms；
- sql_type：SQL 类型，(SELECT,INSERT,UPDATE,DELETE,OTHER 其他类型)；
- sql：执行的具体 SQL 语句；

例如：

```
[WARN ] 2023-01-04 14:55:04.035 [http-nio-8888-exec-7] SLOW-QUERY - db: sharding_db query_time: 21 sql_type: SELECT
SELECT id,user_id,uuid,status,create_time,update_time,is_deleted AS deleted FROM t_order
```


11.5.2 全量审计日志

背景信息

全量审计日志，是指开启该功能后，系统将记录所有执行的 SQL 语句，并包含语句对应的数据库、执行耗时、SQL 类型等信息，便于企业进行审计操作。

参数解释

- general-query-log: 全量审计日志仅有一个参数，值为 true 时启用全量日志，值为 false 时停用该功能。

前提条件

- Agent 使用 SLF4J 进行日志桥接输出，所以要求应用程序存在 SLF4J 依赖，并有相关日志输出配置。
- 全量审计日志功能基于 Agent 技术实现，需要应用程序在启动时配置 javaagent 开启 Agent。配置示例
- Agent 配置

conf/agent.yaml 为 Agent 配置文件，默认配置如下。

```
plugins:
  logging:
    BaseLogging:
      props:
        general-query-log: true
```

说明: general-query-log 为 true 时代表开启全量审计日志，为 false 时代表停用全量审计日志。以启动时配置值为准。

- 应用程序日志配置

以常用的 logback 作为日志输出为例，配置如下。

```
<configuration>
  <property name="log.context.name" value="project-using-DBPlusEngine-Driver" />
  <property name="log.charset" value="UTF-8" />
  <property name="log.pattern" value="[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %logger{36} - %msg%n" />
  <contextName>${log.context.name}</contextName>

  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder charset="${log.charset}">
      <pattern>${log.pattern}</pattern>
    </encoder>
  </appender>

  <logger name="GENERAL-QUERY" level="info" additivity="false">
    <appender-ref ref="STDOUT" />
  </logger>

  <root>
    <level value="INFO" />
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

说明: 全量审计日志输出的 logger name 为 GENERAL-QUERY

全量审计日志格式

db: {database name} query_time: {query time} sql_type: {sql type}

{sql}

- db: 数据库名称;
- query_time: SQL 执行耗时，单位为 ms;
- sql_type: SQL 类型，(SELECT,INSERT,UPDATE,DELETE,OTHER 其他类型);
- sql: 执行的具体 SQL 语句;

例如:

```
[INFO ] 2023-01-04 14:55:04.035 [http-nio-8888-exec-7] GENERAL-QUERY - db: sharding_db query_time: 21 sql_type: SELECT
SELECT id,user_id,uuid,status,create_time,update_time,is_deleted AS deleted FROM t_order
```

11.6 特殊 API

本章节将介绍 DBPlusEngine-Driver 的特殊场景 API。

11.6.1 数据分片

本章节将介绍 DBPlusEngine-Driver 的分片场景 API。

强制路由

简介

DBPlusEngine 使用 ThreadLocal 管理分片键值进行强制路由。可以通过编程的方式向 HintManager 中添加分片值, 该分片值仅在当前线程内生效。

DBPlusEngine 还可以通过 SQL 中增加注释的方式进行强制路由。

Hint 的主要使用场景:

- 分片字段不存在 SQL 和数据库表结构中, 而存在于外部业务逻辑。
- 强制在主库进行某些数据操作。

操作步骤

1. 调用 HintManager.getInstance() 获取 HintManager 实例;
2. 调用 HintManager.addDatabaseShardingValue, HintManager.addTableShardingValue 方法设置分片键值;
3. 执行 SQL 语句完成路由和执行;
4. 调用 HintManager.close 清理 ThreadLocal 中的内容。

配置示例

使用 Hint 分片

规则配置

Hint 分片算法需要用户实现 org.apache.shardingsphere.sharding.api.sharding.hint.HintShardingAlgorithm 接口。DB-PlusEngine 在进行路由时, 将会从 HintManager 中获取分片值进行路由操作。

参考配置如下:

```
rules:
- !SHARDING
  tables:
    t_order:
      actualDataNodes: demo_ds_${0..1}.t_order_${0..1}
```

```
databaseStrategy:
  hint:
    algorithmClassName: xxx.xxx.xxx.HintXXXAlgorithm
tableStrategy:
  hint:
    algorithmClassName: xxx.xxx.xxx.HintXXXAlgorithm
defaultTableStrategy:
  none:
defaultKeyGenerateStrategy:
  type: SNOWFLAKE
  column: order_id

props:
  sql-show: true
```

获取 HintManager

```
HintManager hintManager = HintManager.getInstance();
```

添加分片键值

- 使用 `hintManager.addDatabaseShardingValue` 来添加数据源分片键值。
- 使用 `hintManager.addTableShardingValue` 来添加表分片键值。
分库不分表情况下，强制路由至某一个分库时，可使用 `hintManager.setDatabaseShardingValue` 方式设置分片值。

清除分片键值

分片键值保存在 `ThreadLocal` 中，所以需要在操作结束时调用 `hintManager.close()` 来清除 `ThreadLocal` 中的内容。

hintManager 实现了 **AutoCloseable** 接口，可推荐使用 **try with resource** 自动关闭。

完整代码示例

```
// Sharding database and table with using HintManager
String sql = "SELECT * FROM t_order";
try (HintManager hintManager = HintManager.getInstance();
    Connection conn = dataSource.getConnection();
    PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.addDatabaseShardingValue("t_order", 1);
    hintManager.addTableShardingValue("t_order", 2);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while (rs.next()) {
            // ...
        }
    }
}

// Sharding database without sharding table and routing to only one database with using HintManager
String sql = "SELECT * FROM t_order";
try (HintManager hintManager = HintManager.getInstance();
    Connection conn = dataSource.getConnection();
    PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.setDatabaseShardingValue(3);
}
```

```
try (ResultSet rs = preparedStatement.executeQuery()) {
    while (rs.next()) {
        // ...
    }
}
```

使用 SQL 注释的方式

使用规范

SQL Hint 功能需要用户提前开启解析注释的配置，设置 `sqlCommentParseEnabled` 为 `true`。注释格式暂时只支持 `/**/`，内容需要以 `ShardingSphere hint:` 开始，可选的属性包括：

- `{table}.SHARDING_DATABASE_VALUE`：用于添加 `{table}` 表对应的数据源分片键值，多个属性使用逗号分隔；
 - `{table}.SHARDING_TABLE_VALUE`：用于添加 `{table}` 表对应的表分片键值，多个属性使用逗号分隔。
- 分库不分表情况下，强制路由至某一个分库时，可使用 `SHARDING_DATABASE_VALUE` 方式设置分片，无需指定 `{table}`。

完整示例

```
/* SHARDINGSPHERE_HINT: t_order.SHARDING_DATABASE_VALUE=1,t_order.SHARDING_TABLE_VALUE=1 */
SELECT * FROM t_order;
```

11.6.2 读写分离

本章节将介绍 `DBPlusEngine-Driver` 的读写分离场景 API。

强制路由

简介

`DBPlusEngine` 使用 `ThreadLocal` 管理主库路由标记进行强制路由。可以通过编程的方式向 `HintManager` 中添加主库路由标记，该值仅在当前线程内生效。

`DBPlusEngine` 还可以通过 SQL 中增加注释的方式进行主库路由。

Hint 在读写分离场景下，主要用于强制在主库进行某些数据操作。

操作步骤

1. 调用 `HintManager.getInstance()` 获取 `HintManager` 实例；
2. 调用 `HintManager.setWriteRouteOnly()` 方法设置主库路由标记；
3. 执行 SQL 语句完成路由和执行；
4. 调用 `HintManager.close` 清理 `ThreadLocal` 中的内容。

配置示例

使用 Hint 强制主库路由

使用手动编程的方式

获取 HintManager

与基于 Hint 的数据分片相同。

设置主库路由

使用 `hintManager.setWriteRouteOnly` 设置主库路由。

清除分片键值

与基于 Hint 的数据分片相同。

完整代码示例

```
String sql = "SELECT * FROM t_order";
try (HintManager hintManager = HintManager.getInstance();
    Connection conn = dataSource.getConnection();
    PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.setWriteRouteOnly();
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while (rs.next()) {
            // ...
        }
    }
}
```

使用 SQL 注释的方式

使用规范

SQL Hint 功能需要用户提前开启解析注释的配置，设置 `sqlCommentParseEnabled` 为 `true`。注释格式暂时只支持 `/* */`，内容需要以 `SHARDINGSPHERE_HINT:` 开始，属性名为 `WRITE_ROUTE_ONLY`。

完整示例

```
/* SHARDINGSPHERE_HINT: WRITE_ROUTE_ONLY=true */
SELECT * FROM t_order;
```

11.6.3 分布式事务

通过 DBPlusEngine 使用分布式事务，与本地事务并无区别。除了透明化分布式事务的使用之外，DBPlusEngine 还能够在每次数据库访问时切换分布式事务类型。支持的事务类型包括本地事务、XA 事务和柔性事务。可在创建数据库连接之前设置，缺省为 DBPlusEngine 启动时的默认事务类型。

使用 Java API

引入 Maven 依赖

```
<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-jdbc-core</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>

<!-- 使用 XA 事务时，需要引入此模块 -->
<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-transaction-xa-core</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>

<!-- 使用 BASE 事务时，需要引入此模块 -->
<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-transaction-base-seata-at</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>
```

使用分布式事务

```
TransactionTypeHolder.set(TransactionType.XA); // 支持 TransactionType.LOCAL, TransactionType.XA, TransactionType.BASE
try (Connection conn = dataSource.getConnection()) { // 使用 ShardingSphereDataSource
    conn.setAutoCommit(false);
    PreparedStatement ps = conn.prepareStatement("INSERT INTO t_order (user_id, status) VALUES (?, ?)");
    ps.setObject(1, 1000);
    ps.setObject(2, "init");
    ps.executeUpdate();
    conn.commit();
}
```

使用 Spring Boot Starter

引入 Maven 依赖

```
<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-jdbc-core-spring-boot-starter</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>

<!-- 使用 XA 事务时，需要引入此模块 -->
<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-transaction-xa-core</artifactId>
```

```

<version>${shardingsphere.version}</version>
</dependency>

<!-- 使用 BASE 事务时，需要引入此模块 -->
<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-transaction-base-seata-at</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>

```

配置事务管理器

```

@Configuration
@EnableTransactionManagement
public class TransactionConfiguration {

    @Bean
    public PlatformTransactionManager txManager(final DataSource dataSource) {
        return new DataSourceTransactionManager(dataSource);
    }

    @Bean
    public JdbcTemplate jdbcTemplate(final DataSource dataSource) {
        return new JdbcTemplate(dataSource);
    }
}

```

使用分布式事务

```

@Transactional
@ShardingSphereTransactionType(TransactionType.XA) // 支持 TransactionType.LOCAL, TransactionType.XA, TransactionType.
BASE
public void insert() {
    jdbcTemplate.execute("INSERT INTO t_order (user_id, status) VALUES (?, ?)", (PreparedStatementCallback<Object>) ps -> {
        ps.setObject(1, i);
        ps.setObject(2, "init");
        ps.executeUpdate();
    });
}

```

使用 Spring 命名空间

引入 Maven 依赖

```

<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-jdbc-core-spring-namespaces</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>

<!-- 使用 XA 事务时，需要引入此模块 -->
<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-transaction-xa-core</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>

```

```
<!-- 使用 BASE 事务时，需要引入此模块 -->
<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-transaction-base-seata-at</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>
```

配置事务管理器

```
<!-- ShardingDataSource 的相关配置 -->
<!-- ... -->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="shardingDataSource" />
</bean>
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
  <property name="dataSource" ref="shardingDataSource" />
</bean>
<tx:annotation-driven />

<!-- 开启自动扫描 @ShardingSphereTransactionType 注解，使用 Spring 原生的 AOP 在类和方法上进行增强 -->
<sharding:tx-type-annotation-driven />
```

使用分布式事务

```
@Transactional
@ShardingSphereTransactionType(TransactionType.XA) // 支持 TransactionType.LOCAL, TransactionType.XA, TransactionType.
BASE
public void insert() {
  jdbcTemplate.execute("INSERT INTO t_order (user_id, status) VALUES (?, ?)", (PreparedStatementCallback<Object>) ps -> {
    ps.setObject(1, i);
    ps.setObject(2, "init");
    ps.executeUpdate();
  });
}
```

Atomikos 事务

DBPlusEngine 默认的 XA 事务管理器为 Atomikos。

数据恢复

在项目的 logs 目录中会生成 xa_tx.log, 这是 XA 崩溃恢复时所需的日志，请勿删除。

修改配置

可以通过在项目的 classpath 中添加 jta.properties 来定制化 Atomikos 配置项。

详情请参见 [Atomikos 官方文档](#)。

Narayana 事务

引入 Maven 依赖

```
<properties>
  <narayana.version>5.9.1.Final</narayana.version>
  <jboss-transaction-spi.version>7.6.0.Final</jboss-transaction-spi.version>
  <jboss-logging.version>3.2.1.Final</jboss-logging.version>
</properties>

<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-jdbc-core</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>

<!-- 使用 XA 事务时，需要引入此模块 -->
<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-transaction-xa-core</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>

<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-transaction-xa-narayana</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>
<dependency>
  <groupId>org.jboss.narayana.jta</groupId>
  <artifactId>jta</artifactId>
  <version>${narayana.version}</version>
</dependency>
<dependency>
  <groupId>org.jboss.narayana.jts</groupId>
  <artifactId>narayana-jts-integration</artifactId>
  <version>${narayana.version}</version>
</dependency>
<dependency>
  <groupId>org.jboss</groupId>
  <artifactId>jboss-transaction-spi</artifactId>
  <version>${jboss-transaction-spi.version}</version>
</dependency>
<dependency>
  <groupId>org.jboss.logging</groupId>
  <artifactId>jboss-logging</artifactId>
  <version>${jboss-logging.version}</version>
</dependency>
```


定制化配置项

可以通过在项目的 classpath 中添加 jbossts-properties.xml 来定制化 Narayana 配置项。

详情请参见 [Narayana 官方文档](#)。

设置 XA 事务管理类型

Yaml:

```
- !TRANSACTION
  defaultType: XA
  providerType: Narayana
```

SpringBoot:

```
spring:
  shardingSphere:
    props:
      xa-transaction-manager-type: Narayana
```

Spring Namespace:

```
<shardingSphere:data-source id="xxx" data-source-names="xxx" rule-refs="xxx">
  <props>
    <prop key="xa-transaction-manager-type">Narayana</prop>
  </props>
</shardingSphere:data-source>
```

Bitronix 事务

引入 Maven 依赖

```
<properties>
  <btm.version>2.1.3</btm.version>
</properties>

<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-jdbc-core</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>

<!-- 使用 XA 事务时，需要引入此模块 -->
<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-transaction-xa-core</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>

<dependency>
  <groupId>org.apache.shardingsphere</groupId>
  <artifactId>shardingsphere-transaction-xa-bitronix</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>

<dependency>
  <groupId>org.codehaus.btm</groupId>
  <artifactId>btm</artifactId>
```

```
<version>${btm.version}</version>
</dependency>
```

定制化配置项

详情请参见 [Bitronix 官方文档](#)。

设置 XA 事务管理类型

Yaml:

```
- !TRANSACTION
  defaultType: XA
  providerType: Bitronix
```

SpringBoot:

```
spring:
  shardingsphere:
    props:
      xa-transaction-manager-type: Bitronix
```

Spring Namespace:

```
<shardingsphere:data-source id="xxx" data-source-names="xxx" rule-refs="xxx">
  <props>
    <prop key="xa-transaction-manager-type">Bitronix</prop>
  </props>
</shardingsphere:data-source>
```

Seata 事务

启动 Seata 服务

按照 [seata-work-shop](#) 中的步骤，下载并启动 Seata 服务器。

创建日志表

在每一个分片数据库实例中执创建 undo_log 表（以 MySQL 为例）。

```
CREATE TABLE IF NOT EXISTS `undo_log`
(
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT COMMENT 'increment id',
  `branch_id` BIGINT(20) NOT NULL COMMENT 'branch transaction id',
  `xid` VARCHAR(100) NOT NULL COMMENT 'global transaction id',
  `context` VARCHAR(128) NOT NULL COMMENT 'undo_log context,such as serialization',
  `rollback_info` LONGBLOB NOT NULL COMMENT 'rollback info',
  `log_status` INT(11) NOT NULL COMMENT '0:normal status,1:defense status',
  `log_created` DATETIME NOT NULL COMMENT 'create datetime',
  `log_modified` DATETIME NOT NULL COMMENT 'modify datetime',
  PRIMARY KEY (`id`),
  UNIQUE KEY `ux_undo_log` (`xid`,`branch_id`)
) ENGINE = InnoDB
  AUTO_INCREMENT = 1
  DEFAULT CHARSET = utf8 COMMENT ='AT transaction mode undo table';
```

修改配置

在 classpath 中增加 seata.conf 文件。

```
client {
  application.id = example  ## 应用唯一主键
  transaction.service.group = my_test_tx_group  ## 所属事务组
}
```

根据实际场景修改 Seata 的 file.conf 和 registry.conf 文件。

11.7 YAML 配置

11.7.1 简介

YAML 提供通过配置文件的方式与 SphereEx-DBPlusEngine-JDBC 交互。配合治理模块一同使用时，持久化在配置中心的配置均为 YAML 格式。

说明：YAML 配置文件支持配置内容超过 3MB。

YAML 配置是最常见的配置方式，可以省略编程的复杂度，简化用户配置。

11.7.2 使用步骤

引入 Maven 依赖

```
<dependency>
  <groupId>com.sphere-ex</groupId>
  <artifactId>shardingsphere-jdbc-core</artifactId>
  <version>${shardingsphere.version}</version>
</dependency>
```

配置 YAML

DBPlusEngine-Driver 的 YAML 配置文件通过 Schema 名称、运行模式、数据源集合、规则集合以及属性配置组成。

JDBC 中的数据源的别名。在集群模式，使用该参数联通 DBPlusEngine-Driver 与 DBPlusEngine-Proxy 共同使用。

默认值：logic_db

databaseName (?):

mode:

dataSources:

rules:

- !**FOO_XXX**

...

- !**BAR_XXX**

...

props:

key_1: value_1

key_2: value_2

模式详情请参见[模式配置](#)。

数据源详情请参见[数据源配置](#)。

规则详情请参见[规则配置](#)。

构建数据源

通过 `YamlShardingSphereDataSourceFactory` 工厂创建的 `ShardingSphereDataSource` 实现自 JDBC 的标准接口 `DataSource`。

```
File yamlFile = // 指定 YAML 文件路径
DataSource dataSource = YamlShardingSphereDataSourceFactory.createDataSource(yamlFile);
```

使用数据源

使用方式同 Java API。

11.7.3 语法说明

!! 表示实例化该类

! 表示自定义别名

- 表示可以包含一个或多个

[] 表示数组，可以与减号相互替换使用

11.7.4 支持使用 Apollo 存储 yaml 配置

当您使用 Apollo 加载器时，需要添加对应的 apollo 的 pom 依赖，目前适配的版本为 1.9.0，如下：

```
<dependency>
  <groupId>com.ctrip.framework.apollo</groupId>
  <artifactId>apollo-client</artifactId>
  <version>${apollo.version}</version>
</dependency>
```

11.7.5 模式配置

配置项说明

```
mode (?): # 不配置则默认内存模式
type: # 运行模式类型。可选配置：Cluster
repository (?): # 持久化仓库配置。Memory 类型无需持久化
```

集群模式

```
mode:
type: Cluster
repository:
type: # 持久化仓库类型
props: # 持久化仓库所需属性
namespace: # 注册中心命名空间
server-lists: # 注册中心连接地址
foo_key: foo_value
bar_key: bar_value
```

持久化仓库类型的详情，请参见[内置持久化仓库类型列表](#)。

11.7.6 认证和授权

背景信息

在 SphereEx-DBPlusEngine-Proxy 中，通过 authority 来配置用户的认证和授权信息。

得益于 SphereEx-DBPlusEngine 的可插拔架构，Proxy 提供了两种级别的权限提供者，分别是：

- ALL_PERMITTED：每个用户都拥有所有权限，无需专门授权；
- DATABASE_PERMITTED：为用户授予指定逻辑库的权限，通过 user-database-mappings 进行定义。
- SphereEx:PERMITTED：企业级权限提供者，可进行库、表、列级细粒度授权管理。

在配置 authority 时，管理员可根据需要选择使用哪一种权限提供者。

参数解释

```
authority:
  users:
    - user: # 用于登录计算节点的用户名和授权主机的组合，格式：<username>@<hostname>，hostname 为 % 或空字符串表示不限制授权主机
      password: # 用户密码
      authenticationMethodName: # 可选项，用于为用户指定密码认证方式
    authenticators: # 可选项，默认不需要配置，Proxy 根据前端协议类型自动选择
    authenticatorName:
      type: # 密码认证类型
    defaultAuthenticator: # 可选项，指定一个 authenticatorName 作为默认的密码认证方式
  privilege:
    type: # 权限提供者类型，缺省值为 SphereEx:PERMITTED
```

配置示例

极简配置

```
authority:
  users:
    - user: root@%
      password: root
    - user: sharding
      password: sharding
```

说明：- 定义了两个用户：root@% 和 sharding；- 未定义 authenticators 和 authenticationMethodName，Proxy 将根据前端协议自动选择；- 未指定 privilege type，采用默认的 SphereEx:PERMITTED。

认证配置

自定义认证配置能够满足用户在一些特定场景下的需求。以 openGauss 作为前端协议类型为例，其默认的认证算法为 scram-sha-256。如果用户 sharding 需要用旧版本的 psql 客户端（不支持 scram-sha-256）连接 Proxy，则管理员可能允许 sharding 使用 md5 方式进行密码认证。

支持的协议情况如下：- MySQL 协议支持：mysql_native_password、mysql_clear_password - Postgre 协议支持：password、md5 - openGauss 协议支持：scram-sha-256、md5

配置方式如下：

```
authority:
  users:
    - user: root@127.0.0.1
      password: root
    - user: sharding
      password: sharding
      authenticationMethodName: md5
  authenticators:
    md5:
      type: MD5
  privilege:
    type: SphereEx:PERMITTED
```

说明：- 定义了两个用户：root@127.0.0.1 和 sharding；- 为用户 sharding 指定了 MD5 方式进行密码认证；- 没有为 root@127.0.0.1 指定认证方式，Proxy 将根据前端协议自动选择；- 指定权限提供者 SphereEx:PERMITTED。

授权配置

ALL_PERMITTED

```
authority:
  users:
    - user: root@127.0.0.1
      password: root
    - user: sharding
      password: sharding
  privilege:
    type: ALL_PERMITTED
```

说明：- 定义了两个用户：root@127.0.0.1 和 sharding；- 未定义 authenticators 和 authenticationMethodName，Proxy 将根据前端协议自动选择；- 指定权限提供者 ALL_PERMITTED。

DATABASE_PERMITTED

```
authority:
  users:
    - user: root@127.0.0.1
      password: root
    - user: sharding
      password: sharding
  privilege:
    type: DATABASE_PERMITTED
  props:
    user-database-mappings: root@127.0.0.1=*, sharding@%=test_db, sharding@%=sharding_db
```

说明：- 定义了两个用户：root@127.0.0.1 和 sharding；- 未定义 authenticators 和 authenticationMethodName，Proxy 将根据前端协议自动选择；- 指定权限提供者 SphereEx:PERMITTED，并授权 root@127.0.0.1 用户访问所有逻辑库 (*)，sharding 用户仅能访问 test_db 和 sharding_db。

SphereEx:PERMITTED

```
authority:
users:
- user: root@127.0.0.1
  password: root
- user: sharding
  password: sharding
privilege:
type: SphereEx:PERMITTED
props:
user-database-mappings: root@127.0.0.1=*, sharding@%=test_db, sharding@%=sharding_db
```

说明： - 定义了两个用户：root@127.0.0.1 和 sharding； - 未定义 authenticators 和 authenticationMethodName，Proxy 将根据前端协议自动选择； - 指定权限提供者者为 DATABASE_PERMITTED，并授权 root@127.0.0.1 用户访问所有逻辑库 (*) 的表和列。

相关参考

权限提供者具体实现可以参考 [权限提供者](#)。

11.7.7 存储单元配置

存储单元配置分为单存储单元配置和多存储单元配置。DBPlusEngine-Driver 支持所有的数据库 Driver 驱动和连接池。示例的数据库驱动为 MySQL，连接池为 HikariCP，可以更换为其他数据库驱动和连接池。

配置项说明

```
dataSources: # 存储单元配置，可配置多个 <storage-unit-name>
<data-source-name>: # 存储单元名称
  dataSourceClassName: # 存储单元完整类名
  driverClassName: # 数据库驱动类名，以数据库连接池自身配置为准
  jdbcUrl: # 数据库 URL 连接，以数据库连接池自身配置为准
  username: # 数据库用户名，以数据库连接池自身配置为准
  password: # 数据库密码，以数据库连接池自身配置为准
  # ... 数据库连接池的其它属性
```

配置示例

```
dataSources:
ds_1:
  dataSourceClassName: com.zaxxer.hikari.HikariDataSource
  driverClassName: com.mysql.jdbc.Driver
  jdbcUrl: jdbc:mysql://localhost:3306/ds_1
  username: root
  password:
ds_2:
  dataSourceClassName: com.zaxxer.hikari.HikariDataSource
  driverClassName: com.mysql.jdbc.Driver
  jdbcUrl: jdbc:mysql://localhost:3306/ds_2
  username: root
  password:
# 配置其他存储单元
```

11.7.8 属性配置

背景信息

SphereEx-DBPlusEngine-Proxy 提供了丰富的系统配置属性，用户可通过 server.yaml 进行配置。

参数解释

属性配置可以通过 DistSQL 在线修改。其中支持动态修改的属性立即生效，不支持动态修改的属性在重启后生效。

配置示例

完整配置示例请参考：

```
#props:
# max-connections-size-per-query: 1
# kernel-executor-size: 16 # Infinite by default.
# proxy-frontend-flush-threshold: 128 # The default value is 128.
# proxy-hint-enabled: false
# sql-show: false
# check-table-metadata-enabled: false
# show-process-list-enabled: false
# # Proxy backend query fetch size. A larger value may increase the memory usage of ShardingSphere Proxy.
# # The default value is -1, which means set the minimum value for different JDBC drivers.
# proxy-backend-query-fetch-size: -1
# check-duplicate-table-enabled: false
# proxy-frontend-executor-size: 0 # Proxy frontend executor size. The default value is 0, which means let Netty decide.
# # Available options of proxy backend executor suitable: OLAP(default), OLTP. The OLTP option may reduce time cost of writing
# packets to client, but it may increase the latency of SQL execution
# # and block other clients if client connections are more than `proxy-frontend-executor-size`, especially executing slow SQL.
# proxy-backend-executor-suitable: OLAP
# proxy-frontend-max-connections: 0 # Less than or equal to 0 means no limitation.
# sql-federation-enabled: false
# # Available proxy backend driver type: JDBC (default), ExperimentalVertex
# proxy-backend-driver-type: JDBC
# proxy-mysql-default-version: 5.7.22 # In the absence of schema name, the default version will be used.
# proxy-default-port: 3307 # Proxy default port.
# proxy-netty-backlog: 1024 # Proxy netty backlog.
```

11.7.9 日志配置

计算节点的日志分为三种：运行日志、审计日志、慢日志。

运行日志

```
props:
system-log-level: # 日志级别
system-log-pattern: # 日志格式
```


配置示例

```
props:
system-log-level: INFO
system-log-pattern: "[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %logger{36} - %msg%n"
```

11.7.10 规则配置

规则是 DBPlusEngine 面向可插拔的一部分。本章节是 DBPlusEngine-Driver 的 YAML 插件配置参考手册。

数据分片

配置项说明

```
rules:
- !SHARDING
tables: # 数据分片规则配置
  <logic-table-name> (+): # 逻辑表名称
    actualDataNodes (?): # 由数据源名 + 表名组成 (参考 Inline 语法规则)
    databaseStrategy (?): # 分库策略, 缺省表示使用默认分库策略, 以下的分片策略只能选其一
      standard: # 用于单分片键的标准分片场景
        shardingColumn: # 分片列名称
        shardingAlgorithmName: # 分片算法名称
      complex: # 用于多分片键的复合分片场景
        shardingColumns: # 分片列名称, 多个列以逗号分隔
        shardingAlgorithmName: # 分片算法名称
      hint: # Hint 分片策略
        shardingAlgorithmName: # 分片算法名称
      none: # 不分片
    tableStrategy: # 分表策略, 同分库策略
    keyGenerateStrategy: # 分布式序列策略
      column: # 自增列名称, 缺省表示不使用自增主键生成器
      keyGeneratorName: # 分布式序列算法名称
autoTables: # 自动分片表规则配置
  t_order_auto: # 逻辑表名称
    actualDataSources (?): # 数据源名称
    shardingStrategy: # 切分策略
      standard: # 用于单分片键的标准分片场景
        shardingColumn: # 分片列名称
        shardingAlgorithmName: # 自动分片算法名称
  bindingTables (+): # 绑定表规则列表
    - <logic_table_name_1, logic_table_name_2, ...>
    - <logic_table_name_1, logic_table_name_2, ...>
  broadcastTables (+): # 广播表规则列表
    - <table-name>
    - <table-name>
defaultDatabaseStrategy: # 默认数据库分片策略
defaultTableStrategy: # 默认表分片策略
defaultKeyGenerateStrategy: # 默认的分布式序列策略
defaultShardingColumn: # 默认分片列名称

# 分片算法配置
shardingAlgorithms:
  <sharding-algorithm-name> (+): # 分片算法名称
    type: # 分片算法类型
    props: # 分片算法属性配置
    # ...

# 分布式序列算法配置
```

keyGenerators:

<key-generate-algorithm-name> (+): # 分布式序列算法名称
type: # 分布式序列算法类型
props: # 分布式序列算法属性配置
...

读写分离**配置项说明****静态读写分离****rules:**

- **!READWRITE_SPLITTING**

dataSources:

<data-source-name> (+): # 读写分离逻辑数据源名称
static-strategy: # 读写分离类型
write-data-source-name: # 写库数据源名称
read-data-source-names: # 读库数据源名称，多个从数据源用逗号分隔
loadBalancerName: # 负载均衡算法名称

负载均衡算法配置

loadBalancers:

<load-balancer-name> (+): # 负载均衡算法名称
type: # 负载均衡算法类型
props: # 负载均衡算法属性配置
...

动态读写分离**rules:**

- **!READWRITE_SPLITTING**

dataSources:

<data-source-name> (+): # 读写分离逻辑数据源名称
dynamic-strategy: # 读写分离类型
auto-aware-data-source-name: # 数据库发现逻辑数据源名称
write-data-source-query-enabled: # 从库全部下线，主库是否承担读流量
loadBalancerName: # 负载均衡算法名称

负载均衡算法配置

loadBalancers:

<load-balancer-name> (+): # 负载均衡算法名称
type: # 负载均衡算法类型
props: # 负载均衡算法属性配置
...

算法类型的详情，请参见[内置负载均衡算法列表](#)。查询一致性路由的详情，请参见[使用规范](#)。

数据加密

配置项说明

```
rules:
- !ENCRYPT
tables:
  <table-name> (+): # 加密表名称
  columns:
    <column-name> (+): # 加密列名称
    dataType: # 逻辑列类型
    cipherColumn: # 密文列名称
    cipherDataType: # 加密列类型
    assistedQueryColumn (?): # 查询辅助列名称
    assistedQueryDataType: # 查询辅助列类型
    plainColumn (?): # 原文列名称
    plainDataType: # 原文列类型
    encryptorName: # 加密算法名称
    queryWithCipherColumn(?): # 该表是否使用加密列进行查询

# 加密算法配置
encryptors:
  <encrypt-algorithm-name> (+): # 加解密算法名称
  type: # 加解密算法类型
  props: # 加解密算法属性配置
  # ...

queryWithCipherColumn: # 是否使用加密列进行查询。在有原文列的情况下，可以使用原文列进行查询

# 密钥存储管理配置
keyManagers:
  <key-manager-name> (+): # 密钥存储管理器名称
  type: # 密钥存储管理器类型，支持 Local 存储，AWS 云端存储
  props: # 密钥存储管理器属性配置
  # ...
```

高可用

配置项说明

```
rules:
- !DB_DISCOVERY
dataSources:
  <data-source-name> (+): # 逻辑数据源名称
  dataSourceNames: # 数据源名称列表
  - <data-source>
  - <data-source>
  discoveryHeartbeatName: # 检测心跳名称
  discoveryTypeName: # 数据库发现类型名称

# 心跳检测配置
discoveryHeartbeats:
  <discovery-heartbeat-name> (+): # 心跳名称
  props:
    keep-alive-cron: # cron 表达式，如：'0/5 * * * * ?'

# 数据库发现类型配置
discoveryTypes:
  <discovery-type-name> (+): # 数据库发现类型名称
  type: # 数据库发现类型，如：MySQL.NORMAL_REPLICATION、MySQL.MGR、openGauss.NORMAL_REPLICATION、
  SphereEx.GaussDB_for_MySQL.NORMAL_REPLICATION
```

props (?):

group-name: 92504d5b-6dec-11e8-91ea-246e9612aaf1 # 数据库发现类型必要参数，如 MGR 的 group-name

影子库

配置项说明

```
rules:
- !SHADOW
  dataSources:
    shadowDataSource:
      sourceDataSourceName: # 生产数据源名称
      shadowDataSourceName: # 影子数据源名称
  tables:
    <table-name>:
      dataSourceNames: # 影子表关联影子数据源名称列表
      - <shadow-data-source>
      shadowAlgorithmNames: # 影子表关联影子算法名称列表
      - <shadow-algorithm-name>
  defaultShadowAlgorithmName: # 默认影子算法名称
  shadowAlgorithms:
    <shadow-algorithm-name> (+): # 影子算法名称
    type: # 影子算法类型
    props: # 影子算法属性配置
    # ...
```

SQL 解析

背景信息

SQL 解析 YAML 配置方式具有可读性高，使用简单的特点。通过 YAML 文件的方式，用户可以将代码与配置分离，并且根据需要方便地修改配置文件。

参数解释

```
rules:
- !SQL_PARSER
  sqlCommentParseEnabled: # 是否解析 SQL 注释
  sqlStatementCache: # SQL 语句本地缓存配置项
  initialCapacity: # 本地缓存初始容量
  maximumSize: # 本地缓存最大容量
  parseTreeCache: # 解析树本地缓存配置项
  initialCapacity: # 本地缓存初始容量
  maximumSize: # 本地缓存最大容量
```

操作步骤

1. 设置本地缓存配置
2. 设置解析配置
3. 使用解析引擎解析 SQL

配置示例

```
rules:
- !SQL_PARSER
  sqlCommentParseEnabled: true
  sqlStatementCache:
    initialCapacity: 2000
    maximumSize: 65535
  parseTreeCache:
    initialCapacity: 128
    maximumSize: 1024
```

数据脱敏

背景信息

数据脱敏 YAML 配置方式良好的可读性，通过 YAML 格式，能够快速地理解脱敏规则之间的依赖关系，DBPlusEngine 会根据 YAML 配置，自动完成对象的创建，减少用户不必要的编码工作。

参数解释

```
rules:
- !MASK
  tables:
    <table_name> (+): # 脱敏表名称
      columns:
        <column_name> (+): # 脱敏列名称
          maskAlgorithm: # 脱敏算法

# 脱敏算法配置
maskAlgorithms:
  <mask_algorithm_name> (+): # 脱敏算法名称
    type: # 脱敏算法类型
    props: # 脱敏算法属性配置
    # ...
```

算法类型的详情，请参见内置脱敏算法列表。

操作步骤

1. 在 YAML 文件中配置数据脱敏规则，包含数据源、脱敏规则、全局属性等配置项；
2. 根据 YAML 文件中的配置信息创建对象。

配置示例

数据脱敏 YAML 配置如下：

```
dataSources:
  unique_ds:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.jdbc.Driver
    jdbcUrl: jdbc:mysql://localhost:3306/demo_ds?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
    username: root
    password:

rules:
- !MASK
  tables:
    t_user:
      columns:
        password:
          maskAlgorithm: md5_mask
        email:
          maskAlgorithm: mask_before_special_chars_mask
        telephone:
          maskAlgorithm: keep_first_n_last_m_mask

  maskAlgorithms:
    md5_mask:
      type: MD5
    mask_before_special_chars_mask:
      type: MASK_BEFORE_SPECIAL_CHARS
      props:
        special-chars: '@'
        replace-char: '*'
    keep_first_n_last_m_mask:
      type: KEEP_FIRST_N_LAST_M
      props:
        first-n: 3
        last-m: 4
        replace-char: '*'
```

SQL 翻译

配置项说明

```
rules:
- !SQL_TRANSLATOR
  type: # SQL 翻译器类型
  useOriginalSQLWhenTranslatingFailed: # SQL 翻译失败是否使用原始 SQL 继续执行
```

请参考内置算法

分布式事务

背景信息

SphereEx-DBPlusEngine 提供了三种模式的分布式事务 LOCAL, XA, BASE 。

参数解释

```
rules:
- !TRANSACTION
  defaultType: # 事务模式，可选值 LOCAL/XA/BASE
  providerType: # 指定模式下的具体实现
```

操作步骤

使用 LOCAL 模式

server.yaml 配置文件内容如下：

```
rules:
- !TRANSACTION
  defaultType: LOCAL
```

使用 XA 模式

server.yaml 配置文件内容如下：

```
rules:
- !TRANSACTION
  defaultType: XA
  providerType: Narayana/Atomikos
```

说明：

SphereEx-DBPlusEngine 内置了 Narayana 的相关依赖，可以直接配置使用

使用 BASE 模式

server.yaml 配置文件内容如下：

```
rules:
- !TRANSACTION
  defaultType: BASE
  providerType: Seata
```

搭建 Seata Server，添加相关配置文件，和 Seata 依赖，具体步骤参考 [DBPlusEngine 集成 Seata 柔性事务](#)

数据分片路由缓存

背景信息

该项功能为**实验性功能**，需要与数据分片功能同时使用。数据分片路由缓存会将逻辑 SQL、分片键实际参数值、路由结果放入缓存中，以空间换时间，减少路由逻辑对 CPU 的使用。

建议仅在满足以下条件的情况下启用：- 纯 OLTP 场景 - DBPlusEngine 进程所在机器 CPU 已达到瓶颈 - CPU 开销主要在于 DBPlusEngine 路由逻辑 - 所有 SQL 已经最优且每次 SQL 执行都能命中单一片片

在不满足以上条件的情况下使用，可能对 SQL 的执行延时不会有明显改善，同时会增加内存的压力。

参数解释

```
rules:
- !SHARDING_CACHE
  allowedMaxSqlLength: 512 # 允许缓存的 SQL 长度限制
  routeCache:
    initialCapacity: 65536 # 缓存初始容量
    maximumSize: 262144 # 缓存最大容量
    softValues: true # 是否软引用缓存值
```

混合规则

混合配置的规则项之间的叠加使用是通过数据源名称和表名称关联的。

如果前一个规则是面向数据源聚合的，下一个规则在配置数据源时，则需要使用前一个规则配置的聚合后的逻辑数据源名称；同理，如果前一个规则是面向表聚合的，下一个规则在配置表时，则需要使用前一个规则配置的聚合后的逻辑表名称。

配置项说明

```
dataSources: # 配置真实存在的数据源作为名称
write_ds:
  # ... 省略具体配置
read_ds_0:
  # ... 省略具体配置
read_ds_1:
  # ... 省略具体配置

rules:
- !SHARDING # 配置数据分片规则
  tables:
    t_user:
      actualDataNodes: ds.t_user_${0..1} # 数据源名称 `ds` 使用读写分离配置的逻辑数据源名称
      tableStrategy:
        standard:
          shardingColumn: user_id
          shardingAlgorithmName: t_user_inline
      shardingAlgorithms:
        t_user_inline:
          type: INLINE
          props:
            algorithm-expression: t_user_${user_id % 2}

- !ENCRYPT # 配置数据加密规则
  tables:
    t_user: # 表名称 `t_user` 使用数据分片配置的逻辑表名称
    columns:
```



```
pwd:
  plainColumn: plain_pwd
  cipherColumn: cipher_pwd
  encryptorName: encryptor_aes
encryptors:
  encryptor_aes:
    type: aes
    props:
      aes-key-value: 123456abc

- !READWRITE_SPLITTING # 配置读写分离规则
dataSources:
  ds: # 读写分离的逻辑数据源名称 `ds` 用于在数据分片中使用
    type: Static
    props:
      write-data-source-name: write_ds # 使用真实存在的数据源名称 `write_ds`
      read-data-source-names: read_ds_0, read_ds_1 # 使用真实存在的数据源名称 `read_ds_0` `read_ds_1`
    loadBalancerName: roundRobin
  loadBalancers:
    roundRobin:
      type: ROUND_ROBIN

props:
  sql-show: true
```

12.1 配置文件

12.1.1 集群生态工具配置文件列表

SphereEx-Boot

配置文件名	位置	说明
cluster-template.yaml	当前操作目录	Boot 安装集群
console_install.yaml	当前操作目录	Boot 安装 Console 配置文件

SphereEx-Console

配置文件名	位置	说明
application.yml	conf 目录	软件运行内部文件，内的内容不建议修改
application-prod.yml	conf 目录	运行参数文件，可以修改

■ application.yml 配置说明

```
#profiles
spring:
  profiles:
    active: prod # 固定值不可以修改
servlet:
  multipart:
    enabled: true # 不可以修改
    max-file-size: 10MB # 单个文件最大大小
    max-request-size: 10MB # 单个请求文件总数大小

# mybatis-plus 配置不可以修改
mybatis-plus:
  mapper-locations: classpath:com/sphereex/console/mapper/*.xml
  configuration:
    log-impl: org.apache.ibatis.logging.slf4j.Slf4jImpl
  global-config:
    db-config:
      logic-delete-field: deleted
      logic-delete-value: 1
      logic-not-delete-value: 0
      id-type: auto
      where-strategy: not_empty

# url 白名单不可以修改
```

```
secure:
ignored:
  urls[0]: /
  urls[1]: /**/*.js
  urls[2]: /**/*.css
  urls[3]: /**/*.png
  urls[4]: /**/*.ico
  urls[5]: /**/*.html
  urls[6]: /error
  urls[7]: /static/**
  urls[8]: /api/login
  urls[9]: /api/logout
  urls[10]: /api/**/download
  urls[11]: /api/user/reset
  urls[12]: /api/monitor/config/reset
```

zookeeper 默认配置 不可修改

```
sphereex:
install:
  zookeeper:
    tick-time: 2000
    init-limit: 10
    sync-limit: 9
    communication-port: 2888
    election-port: 3888
```

线程池配置 不可修改

```
command:
thread:
pool:
  queue-capacity: 100 # 队列长度
  core-pool-size: 4 # 核心线程数
  max-pool-size: 8 # 最大线程数
  keep-alive-seconds: 600 # 空闲线程存活时间
```

■ application-prod.yml 配置说明

```
server:
port: 8088 # 启动端口 可修改

software:
home: /opt/software # 软件安装包目录 可修改

spring:
datasource:
url: jdbc:mysql://127.0.0.1:3306/console?serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=true # 数据库连接地址
可修改
username: root # 数据库用户 可修改
password: 123456 # 数据库密码 可修改
driverClassName: com.mysql.jdbc.Driver # 数据库驱动固定值 不可修改

jwt:
header: token # token 属性名称 不可修改
secret: your_secret # token 签名秘钥(字符串) 可修改
expiration: 30000000000 # token 过期时间毫秒 可修改

# console 连接 engine 线程池配置 不建议修改
sphereex:
proxy:
datasource:
hikari:
  maximum-pool-size: 4 # 连接池中可以保留连接的最大数量
  connection-timeout: 3000 # 从连接池获取连接时最大等待时间,单位毫秒
  minimum-idle: 2 # 最小空闲连接数量
```

idle-timeout: 500000 # 连接可以在池中的最大闲置时间,单位毫秒,
max-lifetime: 540000 # 连接最大存活时间,单位毫秒

■ cluster-template.yaml

通过 SphereEx-Boot 部署集群时, 需要提供一份 yaml 格式的集群拓扑配置文件, 配置数据如下。

cluster_name: 集群名字
install_user: 登录部署机用户名
install_password: 登录部署机用户密码
proxy: ShardingSphere-Proxy 配置
version: ShardingSphere-Proxy 版本标识
file: 主控机 ShardingSphere-Proxy 安装包文件路径
conf_dir: 主控机 ShardingSphere-Proxy 业务配置文件目录
depend_files: 主控机 ShardingSphere-Proxy 驱动 jar 包文件路径
install_dir: 部署机 ShardingSphere-Proxy 部署目录
port: 部署机 ShardingSphere-Proxy 启动端口
overwrite: 如果部署机安装目录已经存在, 是否重新安装。默认 true。
servers: 部署机信息列表
host: 部署机 IP 地址
port: 部署机 ShardingSphere-Proxy 启动端口。(非必要, 不配置时以 proxy 中配置为准)
install_dir: 部署机 ShardingSphere-Proxy 安装目录 (非必要, 不配置时以 proxy 中配置为准)
agent_conf_file: 主控机 ShardingSphere-Proxy 中 agent 配置文件路径 (非必要, 不配置时以 proxy 中配置为准)
overwrite: 如果部署机安装目录已经存在, 是否重新安装, 默认 true。(非必要, 不配置时以 proxy 中配置为准)
zookeeper: ZooKeeper 配置 (如果不需要 ZooKeeper 时可以不配置)
version: ZooKeeper 版本标识
file: 主控机 ZooKeeper 安装文件路径
conf_file: 主控机 ZooKeeper zoo.cfg 配置文件路径
install_dir: 部署机 Zookeeper 安装目录
data_dir: 部署机 ZooKeeper 配置文件 zoo.cfg 中 dataDir 配置值
port: 部署机 ZooKeeper 启动端口
overwrite: 如果部署机安装目录已经存在, 是否重新安装, 默认 true。
servers: ZooKeeper 部署机列表
host: 部署机 IP 地址
myid: ZooKeeper 集群 myid 值
port: 部署机 ZooKeeper 启动端口 (非必要, 不配置时以 ZooKeeper 中配置为准)
install_dir: 部署机 ZooKeeper 安装目录 (非必要, 不配置时以 ZooKeeper 中配置为准)
conf_file: 主控机 Zookeeper zoo.cfg 配置文件路径 (非必要, 不配置时以 ZooKeeper 中配置为准)
data_dir: 部署机 ZooKeeper 配置文件 zoo.cfg 中 dataDir 配置值 (非必要, 不配置时以 ZooKeeper 中配置为准)
overwrite: 如果部署机安装目录已经存在, 是否重新安装, 默认 true。(非必要, 不配置时以 ZooKeeper 中配置为准)

12.1.2 集群组件配置文件列表

组件类型	组件名称	配置文件名	位置	说明
存储节点		无	无	不做管理
治理中心	Zookeeper	zoo.cfg	zookeeper 安装目录/conf	SphereEx-Console 安装日志中心的配置文件
监控中心	Prometheus	prometheus.yml	安装目录	SphereEx-Console 安装监控中心的配置文件
监控插件	mysql_exporter	mysql_exporter_conf	安装目录	SphereEx-Console 安装 My SQL 监控插件配置文件
监控插件	Zookeeper_exporter	zoo.cfg	zookeeper 的安装目录/conf	SphereEx-Console 安装 Zookeeper 监控插件配置文件
日志中心	Elasticsearch	elasticsearch.yml	安装目录/config	SphereEx-Console 安装日志中心的配置文件
日志中心	Logstash	logstash.conf	安装目录/config	SphereEx-Console 安装日志中心的配置文件
日志插件	Filebeat	filebeat.yml	安装目录/config	SphereEx-Console 安装计算节点日志插件配置文件

■ zoo.cfg

```
tickTime={{zoo_tick_time}}
initLimit={{zoo_init_limit}}
syncLimit={{zoo_sync_limit}}
dataLogDir={{zoo_data_dir}} # 数据日志路径
dataDir={{zoo_data_dir}} # 数据路径
clientPort={{zoo_client_port}}
autopurge.snapRetainCount=500
autopurge.purgeInterval=24
4lw.commands.whitelist=* # 允许命令
admin.enableServer=false
{{zoo_server}} # server.1=192.168.1.148:2888:3888 集群其他服务
```

■ prometheus.yml

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s
scrape_configs:
- job_name: prometheus # 自身监控 后续添加修改监控 也只会修改这个字段下的配置
  static_configs:
  - targets:
    - 10.211.55.3:9090
```

■ mysql_exporter

```
"[client]
host={host} # 监控数据库 host
port={port} # 监控数据库 port
user={user} # 监控数据库 user
password={password}" # 监控数据库 password
```

Zookeeper_exporter zoo.cfg

```
4lw.commands.whitelist=* # 如果目标文件内没有开启允许监控会添加 已有不会添加
```

■ Elasticsearch.yml

- logstash.conf

536

```
hosts => {{output}} # elasticsearch host
index => "cluster-%[[@metadata][log_type]]@%{cluster_id}-%{+YYYY.MM.dd}"
}
}
```

■ filebeat.yml

```
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - {{general_log_path}} # 运行日志路径
  multiline.pattern: '^\[
  multiline.negate: true
  multiline.match: after
  fields_under_root: true
  fields:
    cluster_id: {{cluster_id}} # 集群 id
    cluster_name: {{cluster_name}} # 集群名
    node_id: {{node_id}} # 节点 id
    node_name: {{node_name}} # 节点名
    log_type: general # general slow
- type: log
  enabled: true
  paths:
    - {{slow_log_path}} # 慢日志路径
  multiline.pattern: '^timestamp:'
  multiline.negate: true
  multiline.match: after
  fields_under_root: true
  fields:
    cluster_id: {{cluster_id}} # 集群 id
    cluster_name: {{cluster_name}} # 集群名
    node_id: {{node_id}} # 节点 id
    node_name: {{node_name}} # 节点名
    log_type: slow # general slow

output.logstash:
  hosts: ["{{output_host}}"] # logstash 的 host port

processors:
- drop_fields:
  fields: ["log", "host", "input", "agent", "ecs"]
```

12.1.3 集群配置文件列表

集群配置文件即计算节点的配置文件。

yaml 支持大于 3M。

配置文件名	位置	说明
server.yaml	conf 目录	初始化参数文件
logback.xml	conf 目录	日志参数配置文件
agent.yaml	conf 目录	计算节点的 agent 参数文件
start.sh	bin 目录	启动参数运行时配置的配置文件

server.yaml 配置说明

■ 权限

用于配置登录计算节点的初始用户，和存储节点数据授权。

配置项说明

authority:

users:

- **user:** # 用于登录计算节点的用户名和授权主机，格式：<username>@<hostname>，hostname 为 % 或空字符串表示不限制授权主机
- password:** # 用户密码
- privilege:**
- type:** # 权限提供者类型，缺省值为 ALL_PERMITTED

配置示例

authority:

users:

- **user:** root@localhost
- password:** root
- **user:** my_user
- password:** pwd

privilege:

type: ALL_PERMITTED

以上配置表示：

- 用户 root，仅可从 localhost 连接 Proxy，密码为 root；
- 用户 my_user，可以从任意主机连接 Proxy，密码为 pwd；
- privilege 类型为 ALL_PERMITTED，表示对用户授予所有权限，不鉴权。

DATABASE_PERMITTED

authority:

users:

- **user:** root@localhost
- password:** root
- **user:** my_user
- password:** pwd

privilege:

type: DATABASE_PERMITTED

props:

user-database-mappings: root@localhost=sharding_db, root@localhost=test_db, my_user@=sharding_db

以上配置表示：

- privilege 类型为 DATABASE_PERMITTED，表示对用户授予库级别权限，需要配置；
 - 用户 root 仅可从 localhost 主机连接，可访问 sharding_db 和 test_db；
 - 用户 my_user 可从任意主机连接，可访问 sharding_db。
- 登录认证

密码认证

DBPlusEngine-Proxy 默认使用密码认证方式，配置格式如下：

authority:

users:

- **user:** root@root
- password:** root
- **user:** sharding
- password:** sharding

该配置中为 DBPlusEngine 指定了两个用户：

- root: @% 表示该用户可以通过任意 host 访问 DBPlusEngine，password 指定了密码为 root。
- sharding: 该用户没有指定 host 配置，默认值同样为 @%，password 指定了密码为 sharding。

当管理员需要限制特定用户的登录主机时，可以通过 `username@host` 这样的方式来指定，如：


```
- user: user1@192.168.1.111password: user1_password
```

表示 user1 用户只能通过 192.168.1.111 这个地址访问 DBPlusEngine，认证密码是 user1_password。

LDAP 认证

说明：- 在启用 LDAP 认证之前，用户应先行部署 LDAP 服务器，如 [OpenLDAP](#)。

- 当使用 MySQL 客户端时，需显示开启 cleartext-plugin，如：mysql -h 127.0.0.1 -P 3307 -u root -p --enable-cleartext-plugin

在 DBPlusEngine 中配置 LDAP 的方式如下：

示例 1

每个用户都需要经过 LDAP 认证，且使用相同的 DN 模板。

```
authority:
users:
- user: root@%
- user: sharding
authenticators:
auth_ldap:
type: LDAP
props:
ldap_server_url: ldap://localhost:389ldap_dn_template: cn={0},ou=users,dc=example,dc=org
defaultAuthenticator: auth_ldap
```

该配置指定了一个身份认证器 auth_ldap，它的类型为 LDAP，props 中给出了必要的配置：

- ldap_server_url: LDAP 服务器的访问地址
- ldap_dn_template: 用户 DN 模板

在使用以上配置时，用户 root 和 sharding 对应的用户 DN 分别是：

- root: cn=root,ou=users,dc=example,dc=org
- sharding: cn=sharding,ou=users,dc=example,dc=org

示例 2

每个用户都需要经过 LDAP 认证，但使用不同的 DN 模板。

```
authority:
users:
- user: root@%
props:
ldap_dn: cn=root,ou=admin,dc=example,dc=org
- user: sharding
authenticators:
auth_ldap:
type: LDAP
props:
ldap_server_url: ldap://localhost:389ldap_dn_template: cn={0},ou=users,dc=example,dc=org
defaultAuthenticator: auth_ldap
```

与 示例 1 的区别是：用户 root 与其他用户不在同一个 ou 中，因此为 root 单独指定了明确的用户 DN。在使用以上配置时，用户 root 和 sharding 对应的用户 DN 分别是：

- root: cn=root,ou=admin,dc=example,dc=org
- sharding: cn=sharding,ou=users,dc=example,dc=org

混合认证

混合认证意味着部分用户使用密码认证方式，另一部分用户使用 LDAP 认证方式，这是一种非常灵活的搭配，可满足特定安全场景的需要。

混合认证配置格式如下：

```
authority:
users:
- user: root@%
  auth: auth_ldap
- user: sharding
  password: sharding
- user: user1
  password: password_user1
authenticators:
auth_ldap:
type: LDAP
props:
  ldap_server_url: ldap://localhost:389ldap_dn_template: cn={0},ou=users,dc=example,dc=org
```

在以上配置中，没有指定 defaultAuthenticator，默认为使用密码认证方式。与此同时，通过显示配置 auth: auth_ldap，为用户 root 指定了身份认证器，要求该用户通过 LDAP 认证进行登录。在使用以上配置时，用户 root、sharding 和 user1 对应的认证方式分别是：

- root: LDAP
 - sharding: 密码
 - user1: 密码

说明：在混合认证场景中，管理员也可以通过默认启用 LDAP 认证，并使用 auth: password 将少部分用户设置为密码认证。

logback.xml 配置说明

参考[日志配置](#)

agent.yaml 配置说明

参考[Agent 管理部分](#)

12.2 集群元数据

12.2.1 元数据持久化仓库

背景信息

DBPlusEngine 提供了不同的元数据持久化方式，用户可以选择合适的方式来存储元数据。

参数解释

ZooKeeper 持久化

类型：ZooKeeper

可配置属性：

名称	数据类型	说明	默认值
retryIntervalMilliseconds	int	重试间隔毫秒数	500
maxRetries	int	客户端连接最大重试次数	3
timeToLiveSeconds	int	临时数据失效的秒数	60
operationTimeoutMilliseconds	int	客户端操作超时的毫秒数	500
digest	String	登录认证密码	

Etcd 持久化

类型：Etcd

可配置属性：

名称	数据类型	说明	默认值
timeToLiveSeconds	long	临时数据失效的秒数	30
connectionTimeout	long	连接超时秒数	30

SphereEx-DBPlusEngine-Mate 持久化

类型: SphereEx:MATE

可配置属性:

暂无

操作步骤

配置元数据持久化仓库类型。

配置示例

```
mode:
type: Cluster
repository:
type: SphereEx:MATE
props:
namespace: governance
server-lists: localhost:21506
```

12.2.2 集群元数据存储（ZooKeeper）

在定义的命名空间下，rules、props和metadata节点以YAML格式存储配置，可通过修改节点来实现对于配置的动态管理。nodes存储数据库访问对象运行节点，用于区分不同数据库访问实例。

```
namespace
├── rules # 全局规则配置
├── props # 属性配置
├── metadata # Metadata 配置
│   ├── ${dbName} # 逻辑数据库名称
│   │   ├── schemas # Schema 列表
│   │   │   ├── ${schemaName} # 逻辑 Schema 名称
│   │   │   │   ├── tables # 表结构配置
│   │   │   │   │   ├── ${tableName}
│   │   │   │   │   ├── ...
│   │   │   │   └── views # 视图结构配置
│   │   │   │       ├── ${viewName}
│   │   │   │       └── ...
│   │   │   └── ...
│   │   └── versions # 元数据版本列表
│   │       ├── ${versionNumber} # 元数据版本号
│   │       └── data_sources # 数据源配置
│   │           ├── rules # 规则配置
│   │           ├── ...
│   │           └── active_version # 激活的元数据版本号
│   └── ...
├── nodes
│   ├── compute_nodes
│   │   ├── online
│   │   └── proxy
```

```

├──├──├──├──├──UUID      # Proxy 实例唯一标识
├──├──├──├──├──....
├──├──├──├──├──jdbc
├──├──├──├──├──UUID      # JDBC 实例唯一标识
├──├──├──├──├──....
├──├──├──├──├──status
├──├──├──├──├──UUID
├──├──├──├──├──....
├──├──├──├──├──worker_id
├──├──├──├──├──UUID
├──├──├──├──├──....
├──├──├──├──├──process_trigger
├──├──├──├──├──process_list_id:UUID
├──├──├──├──├──....
├──├──├──├──├──labels
├──├──├──├──├──UUID
├──├──├──├──├──....
├──├──├──├──├──xa_recovery_id
├──├──├──├──├──UUID
├──├──├──├──├──├──├──UUID
├──├──├──├──├──storage_nodes
├──├──├──├──├──├──${databaseName.groupName.ds}
├──├──├──├──├──├──${databaseName.groupName.ds}
├──├──├──├──├──sys_data
├──├──├──├──├──├──shardingsphere
├──├──├──├──├──├──├──schemas
├──├──├──├──├──├──├──├──shardingsphere
├──├──├──├──├──├──├──├──├──tables      # 系统表
├──├──├──├──├──├──├──├──├──├──sharding_table_statistics # 分片统计表数据
├──├──├──├──├──├──├──├──├──├──├──8a2dcb0d97c3d86ef77b3d4651a1d7d0 # md5
├──├──├──├──├──├──├──├──├──├──├──cluster_information # 集群信息表

```

/rules

全局规则配置，可包括访问 DBPlusEngine-Proxy 用户名和密码的权限配置。

全局规则配置，可包含事务配置、SQL 解析配置等。

```

- !TRANSACTION
  defaultType: XA
  providerType: Atomikos
- !SQL_PARSER
  sqlCommentParseEnabled: true

```

/props

属性配置

```

kernel-executor-size: 20
sql-show: true

```

/metadata/\${schemaName}/dataSources

多个数据库连接池的集合，不同数据库连接池属性自适应（例如：DBCP，C3P0，Druid，HikariCP）。

```

ds_0:
  initializationFailTimeout: 1validationTimeout: 5000maxLifetime: 180000leakDetectionThreshold: 0minimumIdle:
  1password: root
  idleTimeout: 60000jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds_0?serverTimezone=UTC&useSSL=falsedataSourceClassName: com.
  zaxxer.hikari.HikariDataSource
  maximumPoolSize: 50connectionTimeout: 30000username: root
  poolName: HikariPool-1
ds_1:
  initializationFailTimeout: 1validationTimeout: 5000maxLifetime: 180000leakDetectionThreshold: 0minimumIdle:
  1password: root

```

```
idleTimeout: 60000jdbcUrl: jdbc:mysql://127.0.0.1:3306/ds_1?serverTimezone=UTC&useSSL=falsedataSourceClassName: com.
zaxxer.hikari.HikariDataSource
maximumPoolSize: 50connectionTimeout: 30000username: root
poolName: HikariPool-2
```

/metadata/\${schemaName}/rules

规则配置，可包括数据分片、读写分离、数据加密、影子库压测等配置。

```
- !SHARDING
XXX

- !READWRITE_SPLITTING
XXX

- !ENCRYPT
XXX
```

/metadata/\${schemaName}/tables

表结构配置，每个表使用单独节点存储，暂不支持动态修改。

```
name: t_order          # 表名
columns:               # 列 id:          # 列名 caseSensitive: falsedataType: 0generated: falsename: id
  primaryKey: true
  order_id:
    caseSensitive: falsedataType: 0generated: falsename: order_id
    primaryKey: false
indexes:               # 索引 t_user_order_id_index:      # 索引名 name: t_user_order_id_index
```

/nodes/compute_nodes

数据库访问对象运行实例信息，子节点是当前运行实例的标识。运行实例标识由运行服务器的 IP 地址和 PORT 构成。运行实例标识均为临时节点，当实例上线时注册，下线时自动清理。注册中心监控这些节点的变化来治理运行中实例对数据库的访问等。

/nodes/storage_nodes

可以治理读写分离从库，可动态添加删除以及禁用。

12.3 数据字典

12.3.1 shardingSphere 库

shardingSphere 库为 SphereEx-DBPlusEngine 的数据字典库，用于查看 SphereEx-DBPlusEngine 的字典信息。

12.3.2 sharding_table_statistics 表

shardingSphere 库中的 sharding_table_statistics 表是 DBPlusEngine 的数据字典表，用于查询分片表的统计信息，数据来源是底层 DB 的统计信息表，收集方式为，内置定时任务，频率为 30 秒一次，用户无法干预。

如下为查询示例

```
mysql> select * from shardingSphere.sharding_table_statistics;+----+-----+-----+-----+-----+-----+-----+
| id | logic_database_name | logic_table_name | actual_database_name | actual_table_name | row_count |
| size | +----+-----+-----+-----+-----+-----+-----+ | 3 | test |
| t_order | ds0 | t_order_2 | 1 | 16384 | | 4 | test | t_order | ds1 | t_order_3 | 1 | 16384 | | 2 |
test | t_order | ds1 | t_order_1 | 1 | 16384 | | 1 | test | t_order | ds0 | t_order_0 | 1 | 16384 |
+----+-----+-----+-----+-----+-----+-----+ 4 rows in set (0.05 sec)
```

12.3.3 对数据库数据字典的兼容性

SphereEx-DBPlusEngine 提供了对接数据库元数据的兼容能力，目前已实现对数据库的兼容情况。

说明：只支持空查询，用于兼容各个数据库客户端。

对 MySQL 元数据兼容情况如下

库	表
information_schema	tables
	views
	columns
mysql	db
performance_schema	accounts
	sys
	sys_config

对 PG 兼容情况如下

库	表
information_schema	tables
	views
	columns
pg_catalog	pg_tablespace
	pg_database

12.4 参数

12.4.1 集群生态工具参数

SphereEx-Boot 命令行参数列表

集群

命令	含义	参数
download	下载 ShardingSphere-Proxy, Zookeeper, Database driver 安装包	-type[proxy zk driver]Optionproxyorzkd-driver-proxy-urlTEXTProxydownloadurl-zk-urlTEXTZookeeperdownloadurl-driver-urlTEXTDatabase driverdownloadurl-output-dirTEXTOutputdir,Defaultcurrentdirectory-help
init	快速初始化集群操作的配置文件	-nameTEXTnameofcluster[required]-userTEXTinstalluser-passwordTEXTinstallpassword-proxy-filePATHpackagefileofproxy-proxy-conf-dirPATHconfigfileofproxy-proxy-agent-conf-filePATHconfigfileofagent-proxy-depend-filePATHdependfileofproxy, forexample, database driver. forexample: -proxy-depend-file/root/mysql-connector-java-5.1.47.jar-proxy-install-dirTEXTinstalldirectoryofproxy-proxy-portINTEGERstartportofproxy-proxy-overwriteTEXToverwriteifinstalldirectoryalreadyexists-proxy-hostTEXTproxyhostip, canbeenteredmultiple times, forexample: -proxy-host10.0.1.1-proxy-host10.0.1.2-zk-filePATHpackagefileofZooKeeper-zk-conf-filePATHzoo.cfgfileofzookeeper-zk-install-dirTEXTinstalldirectoryofzookeeper-zk-data-dirTEXTdata directoryofzookeeper-zk-portINTEGERclientPortofzookeeper-zk-overwriteTEXToverwriteifinstalldirectoryalreadyexists-zk-hostTEXTZooKeeperhostip, canbeenteredmultiple times, forexample: -zk-host10.0.1.1-zk-host10.0.1.2-download[all proxy zk driver]Downloadrelated installationpackages-proxy-urlTEXTProxydownloadurl-zk-urlTEXTZooKeeperdownloadurl-driver-urlTEXTDatabase driverdownloadurl-helpShowthismes sageandexit.
install	根据配置文件安装集群(proxy或者zk)	-nameTEXTClustername[required]-hostTEXTHostaddressandportvalue. Forexample: 127.0.0.1Specifythemachi

集群组件

命令	含义	参数
install	目前仅能安装 prometheus	-type[prometheus]Specify component type, default prometheus -f, -file PATH Specify profile -t, -template Export configuration template -user TEXT Install user -password TEXT Install password -install-dir TEXT Install directory -package PATH Installation package -host TEXT Host ip -help Show this message and exit.
uninstall	目前仅能卸载 prometheus	-type[prometheus]Specify component type, default prometheus -user TEXT Install user -password TEXT Install password -install-dir TEXT Install directory -host TEXT Host ip -help Show this message and exit.

集群配置文件

命令	含义	参数
add	添加 cluster 配置文件	-file PATH Cluster configuration file [required] -force If a cluster with the same name exists, do you want to overwrite the configuration -help Show this message and exit.
check	校验 cluster 配置文件可以选择 -f 或者 -n	-file PATH Cluster configuration file -n, -name TEXT Cluster name -help Show this message and exit.
delete	删除 cluster 配置文件	-force Forced delete configuration of cluster! -help Show this message and exit.
info	展示配置文件内容	-name TEXT Cluster name [required] -detail flag, detail info of the cluster configuration -help Show this message and exit.
template	导出全部配置模板或者扩容配置模板	-type[full

镜像

命令	含义	参数
clone	下载依赖	-r, -requirement Output spex dependent. -o, -output-dir PATH Output directory, Default current directory. -help Show this message and exit.

管理 SphereEx-Console

命令	含义	参数
install	安装 console	-c, -config PATH Console configuration file [required] -n, -name TEXT Console name [required] -help Show this message and exit.
uninstall	卸载 console	-n, -name TEXT Console name [required] -help Show this message and exit.
list	列出已经安装的 console 列表	
export	导出安装配置文件模板	-output TEXT Output to the path -help Show this message and exit.
backup	备份 console 使用的 mysql 库数据保存到 console 同级安装目录下在 console 安装的上一层不会因为卸载 console 被删除	-n, -name TEXT Console name [required] -help Show this message and exit.

12.4.2 集群组件参数列表

使用组件过程中的配置参数

■ 监控中心启动参数

```
--web.listen-address=:9090 # 监听端口
--web.enable-lifecycle # 开启热加载 可以通过命令热加载新建监控
```

■ 监控插件 mysql-exporter 启动参数

```
--web.listen-address=":{listen_port}" # 监听端口
--config.my-cnf="{conf}" # 上面内容的配置文件
```

■ 监控插件 zookeeper-exporter 启动参数

```
-bind-addr=":{listen_port}" # 监听端口
-zookeeper="{zk_host}" # 监控的 zk host:port
```

■ 监控插件 Node_exporter 启动参数

```
--web.listen-address=":{listen_port}" # 监听端口
```


12.4.3 集群参数列表

名称	数据类型 *	说明	默认值 *	动态生效 *
sql_federation_type	String	联邦查询执行器类型, 包括: NONE, ORIGINAL, ADVANCED。	NONE	是
sql_federation_enabled	boolean	是否启用 sql federation。	FALSE	TRUE/FALSE
show_process_list_enabled	boolean	是否启用显示进程列表。	FALSE	TRUE/FALSE
agent_plugins_enabled	boolean	TRUE	是否启用 agent plugin 插件。	TRUE/FALSE
transaction_type	String	LOCAL	事务类型。	LOCAL/XA/BASE/MANUALXA
kernel_executor_size	int	用于设置任务处理线程池的大小。每个 ShardingSphere-DataSource 使用一个独立的线程池, 同一个 JVM 的不同数据源不共享线程池。	infinite	否
max_connections_size_per_query	int	一次查询请求在每个数据库实例中所能使用的最大连接数。	1	是
check_table_metadata_enabled	boolean	在程序启动和更新时, 是否检查分片元数据的结构一致性。	false	是
proxy_frontend_flush_threshold	int	在 ShardingSphere-Proxy 中设置传输数据条数的 IO 刷新阈值。	128	是
proxy_hint_enabled	boolean	是否允许在 ShardingSphere-Proxy 中使用 Hint。使用 Hint 会将 Proxy 的线程处理模型由 IO 多路复用变更为每个请求一个独立的线程, 会降低 Proxy 的吞吐量。	false	是
proxy_backend_query_fetch_size	int	Proxy 后端与数据库交互的每次获取数据行数 (使用游标的情况下)。数值增大可能会增加 ShardingSphere Proxy 的内存使用。默认值为 -1, 代表设置为 JDBC 驱动的最小值。	1	是
proxy_frontend_executor_size	int	Proxy 前端 Netty 线程池线程数量, 默认值 0 代表使用 Netty 默认值。	0	否
proxy_backend_executor_suitable	String	可选选项: OLAP、OLTP。OLTP 选项可能会减少向客户端写入数据包的时间开销, 但如果客户端连接数超过 proxy-frontend-executor-size, 尤其是执行慢 SQL 时, 它可能会增加	OLAP	是

特殊说明支持动态修改的属性可以立即生效，不支持动态修改的属性需要重启后生效。参数配置可以通过修改计算节点的yaml文件配置重启生效，其中参数的下划线”_”需要换成中横线”-”；

12.5 Hint

12.5.1 数据分片强制路由

背景信息

SphereEx-DBPlusEngine 使用 ThreadLocal 管理分片键值进行强制路由。可以通过编程的方式向 HintManager 中添加分片值，该分片值仅在当前线程内生效。SphereEx-DBPlusEngine 还可以通过 SQL 中增加注释的方式进行强制路由。

Hint 的主要使用场景：- 分片字段不存在 SQL 和数据库表结构中，而存在于外部业务逻辑。- 强制在指定数据库进行某些数据操作。

操作步骤

1. 调用 HintManager.getInstance() 获取 HintManager 实例；
2. 调用 HintManager.addDatabaseShardingValue, HintManager.addTableShardingValue 方法设置分片键值；
3. 执行 SQL 语句完成路由和执行；
4. 调用 HintManager.close 清理 ThreadLocal 中的内容。

使用 Hint 分片配置示例

规则配置

Hint 分片算法需要用户实现 org.apache.shardingsphere.sharding.api.sharding.hint.HintShardingAlgorithm 接口。Apache ShardingSphere 在进行路由时，将会从 HintManager 中获取分片值进行路由操作。

参考配置如下：

```
rules:
- !SHARDING
  tables:
    t_order:
      actualDataNodes: demo_ds_${0..1}.t_order_${0..1}
      databaseStrategy:
        hint:
          algorithmClassName: xxx.xxx.xxx.HintXXXAlgorithm
      tableStrategy:
        hint:
          algorithmClassName: xxx.xxx.xxx.HintXXXAlgorithm
      defaultTableStrategy:
        none:
      defaultKeyGenerateStrategy:
        type: SNOWFLAKE
        column: order_id
  props:
    sql-show: true
```

获取 HintManager

```
HintManager hintManager = HintManager.getInstance();
```

添加分片键值

- 使用 `hintManager.addDatabaseShardingValue` 来添加数据源分片键值。
 - 使用 `hintManager.addTableShardingValue` 来添加表分片键值。
- 分库不分表情况下，强制路由至某一个分库时，可使用 `hintManager.setDatabaseShardingValue` 方式设置分片值。

清除分片键值

分片键值保存在 `ThreadLocal` 中，所以需要在操作结束时调用 `hintManager.close()` 来清除 `ThreadLocal` 中的内容。

hintManager 实现了 **AutoCloseable** 接口，可推荐使用 **try with resource** 自动关闭。

完整代码示例

```
// Sharding database and table with using HintManager
String sql = "SELECT * FROM t_order";
try (HintManager hintManager = HintManager.getInstance();
     Connection conn = dataSource.getConnection();
     PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.addDatabaseShardingValue("t_order", 1);
    hintManager.addTableShardingValue("t_order", 2);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while (rs.next()) {
            // ...
        }
    }
}

// Sharding database without sharding table and routing to only one database with using HintManager
String sql = "SELECT * FROM t_order";
try (HintManager hintManager = HintManager.getInstance();
     Connection conn = dataSource.getConnection();
     PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.setDatabaseShardingValue(3);
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while (rs.next()) {
            // ...
        }
    }
}
```

使用 SQL 注释的方式配置示例

使用规范

SQL Hint 功能的注释格式暂时只支持 `/* */`，内容需要以 `SHARDINGSPHERE_HINT:` 开始，可选的属性包括：

- `{table}.SHARDING_DATABASE_VALUE`：用于添加 `{table}` 表对应的数据源分片键值，多个属性使用逗号分隔；
- `{table}.SHARDING_TABLE_VALUE`：用于添加 `{table}` 表对应的表分片键值，多个属性使用逗号分隔。

分库不分表情况下，强制路由至某一个分库时，可使用 `SHARDING_DATABASE_VALUE` 方式设置分片，无需指定 `{table}`。

完整示例

```
/* SHARDINGSPHERE_HINT: t_order.SHARDING_DATABASE_VALUE=1,t_order.SHARDING_TABLE_VALUE=1 */  
SELECT * FROM t_order;
```

12.5.2 读写分离强制路由

背景信息

SphereEx-DBPlusEngine 使用 `ThreadLocal` 管理主库路由标记进行强制路由。可以通过编程的方式向 `HintManager` 中添加主库路由标记，该值仅在当前线程内生效。SphereEx-DBPlusEngine 还可以通过 SQL 中增加注释的方式进行主库路由。

Hint 在读写分离场景下，主要用于强制在主库进行某些数据操作。

操作步骤

1. 调用 `HintManager.getInstance()` 获取 `HintManager` 实例；
2. 调用 `HintManager.setWriteRouteOnly()` 方法设置主库路由标记；
3. 执行 SQL 语句完成路由和执行；
4. 调用 `HintManager.close()` 清理 `ThreadLocal` 中的内容。

使用 Hint 强制主库路由配置示例

获取 HintManager

与基于 Hint 的数据分片相同。

设置主库路由

使用 `hintManager.setWriteRouteOnly` 设置主库路由。

清除分片键值

与基于 Hint 的数据分片相同。

完整代码示例

```
String sql = "SELECT * FROM t_order";
try (HintManager hintManager = HintManager.getInstance();
    Connection conn = dataSource.getConnection();
    PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.setWriteRouteOnly();
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while (rs.next()) {
            // ...
        }
    }
}
```

使用 SQL 注释的方式强制主库路由配置示例

使用规范

SQL Hint 功能的注释格式暂时只支持 /* */，内容需要以 SHARDINGSPHERE_HINT: 开始，属性名为 WRITE_ROUTE_ONLY。

完整示例

```
/* SHARDINGSPHERE_HINT: WRITE_ROUTE_ONLY=true */
SELECT * FROM t_order;
```

使用 Hint 强制路由指定数据库配置示例

获取 HintManager

与基于 Hint 的数据分片相同。

设置路由至指定数据库

- 使用 hintManager.setDataSourceName 设置数据库名称。

完整代码示例

```
String sql = "SELECT * FROM t_order";
try (HintManager hintManager = HintManager.getInstance();
    Connection conn = dataSource.getConnection();
    PreparedStatement preparedStatement = conn.prepareStatement(sql)) {
    hintManager.setDataSourceName("ds_0");
    try (ResultSet rs = preparedStatement.executeQuery()) {
        while (rs.next()) {
            // ...
        }
    }
}
```

```
}
}
```

使用 SQL 注释的方式强制路由指定数据库配置示例

使用规范

SQL Hint 功能目前只支持路由至一个数据源。注释格式暂时只支持 `/* */`，内容需要以 `SHARDINGSPHERE_HINT:` 开始，属性名为 `DATA_SOURCE_NAME`。如果使用 MySQL 客户端连接需要添加 `-c` 选项保留注释，客户端默认是 `--skip-comments` 过滤注释。

完整示例

```
/* SHARDINGSPHERE_HINT: DATA_SOURCE_NAME=ds_0 */
SELECT * FROM t_order;
```

使用 SQL 注释的方式跳过解析层的改写逻辑强制透传

跳过解析层的改写逻辑将 SQL 直接透传到 `storage_unit`

使用规范

通过配置 SQL Hint `/* SHARDINGSPHERE_HINT: SKIP_SQL_REWRITE=false */`，跳过所有改写逻辑。跳过改写的 SQL Hint 可以和数据源透传 SQL Hint 相互配合，实现从路由到改写的灵活控制

完整示例

```
/* SHARDINGSPHERE_HINT: SKIP_SQL_REWRITE=false */
SELECT * FROM t_order;
```

12.6 DistSQL

12.6.1 用户管理

用户管理

语法	描述	类型
CREATE DIST USER 'user' @ 'host' IDENTIFIED BY 'password' ;	创建用户	RAL
ALTER DIST USER 'user' @ 'host' IDENTIFIED BY 'password' ;	修改用户密码	RAL
DROP DIST USER 'user' @ 'host' ;	删除用户	RAL
SHOW DIST USERS	查看用户	RAL

1. 创建用户

初始用户由管理员在 DBPlusEngine 启动前指定。在 server.yaml 文件中配置，用于配置登录计算节点的初始用户和存储节点数据授权。

```
authority:
  users:
    - user: root@%
      password: root
  privilege:
    type: SphereEx:PERMITTED
```

CREATE DIST USER 语句用于创建带有指定密码的新用户。

```
CREATE DIST USER 'user'@'host' IDENTIFIED BY 'password';
```

参数说明

在 DBPlusEngine 中，用户是由用户名和用户名所连接主机的组合。

示例

创建一个名为 'sharding' 的用户，密码为 '123456'，'%' 表示不限制登录主机。

```
CREATE DIST USER 'sharding'@'%' IDENTIFIED BY '123456';
```

创建一个只能在 127.0.0.1 登录的用户。

```
CREATE DIST USER 'sharding'@'127.0.0.1' IDENTIFIED BY '123456';
```

2. 修改用户密码

ALTER DIST USER 语句用于修改已有用户，目前只支持修改密码。

```
ALTER DIST USER 'user'@'host' IDENTIFIED BY 'password';
```

示例

将 'sharding' @ '%' 的用户的密码改为 'sharding'。

```
ALTER DIST USER 'sharding'@'%' IDENTIFIED BY 'sharding';
```

3. 删除用户密码

DROP DIST USER 语句用于删除用户。

```
DROP DIST USER 'user'@'host';
```

示例

将 'sharding' @ '%' 的用户从系统中移除。

4. 查看用户

SHOW DIST USERS 语句用于查看用户列表。

SHOW DIST USERS

参数说明

无

示例

查看全部用户，仅查询授权主机和用户名，不展示用户密码。

SHOW DIST USERS;

角色管理

语法	描述	类型
CREATE DIST ROLE roleName;	创建角色	RAL
DROP DIST ROLE roleName;	删除角色	RAL
GRANT DIST role TO 'user' @ 'host' ;	赋予用户角色	RAL
REVOKE DIST role FROM 'user' @ 'host' ;	取消用户角色	RAL
SHOW DIST ROLES;	查看角色	RAL

1. 创建角色示例

CREATE DIST ROLE 语句用于创建角色。

CREATE DIST ROLE roleName;

示例

创建一个名为 'dml_only' 的角色。

CREATE DIST ROLE dml_only;

2. 删除角色示例

DROP DIST ROLE 语句用于删除角色。

DROP DIST **ROLE** roleName;

示例

将 'dml_only' 的用户从系统中移除。

DROP DIST ROLE dml_only;

3. 赋予用户角色

```
GRANT DIST role TO 'user'@'host';
```

示例

将角色 dml_only 赋予给用户 'sharding' @ '%'。

```
GRANT DIST dml_only TO 'sharding'@'%';
```

将全局 SELECT、INSERT、UPDATE 和 DELETE 权限赋予给角色 dml_only。

```
GRANT DIST INSERT,SELECT,UPDATE,DELETE TO dml_only;
```

4. 取消用户角色

```
REVOKE DIST role FROM 'user'@'host';
```

示例

撤销赋予用户 'sharding' @ '%' 的 dml_only 角色。

```
REVOKE DIST dml_only FROM 'sharding'@'%';
```

5. 查看角色列表

SHOW DIST ROLES 语句用于查看角色列表。

```
SHOW DIST ROLES;
```

参数说明

无

示例

查看已创建的全部角色名称。

```
SHOW DIST ROLES;
```

赋予与撤销权限

语法	描述	类型
GRANT DIST SELECT TP 'user' @ 'host' ;	赋予全局权限	R A L
GRANT DIST privileges ON . to 'user' @ 'host' ;	赋予权限（所有对象）	R A L
GRANT DIST privileges ON schema.* to 'user' @ 'host' ;	赋予权限（库级）	R A L
GRANT DIST privileges ON schema.table to 'user' @ 'host' ;	赋予权限（表级）	R A L
GRANT DIST privileges(column name) ON schema.table to 'user' @ 'host' ;	赋予权限（列级）	R A L
REVOKE DIST ALL FROM 'user' @ 'host' ;	撤销所有权限	R A L
REVOKE DIST privileges FROM 'user' @ 'host' ;	撤销全局权限	R A L
REVOKE DIST privileges ON . FROM 'user' @ 'host' ;	撤销权限（所有对象）	R A L
REVOKE DIST privileges ON schema.* FROM 'user' @ 'host' ;	撤销权限（库级）	R A L
REVOKE DIST privileges ON schema.table FROM 'user' @ 'host' ;	撤销权限（表级）	R A L
REVOKE DIST privileges(column name) ON schema.table FROM 'user' @ 'host' ;	撤销权限（列级）	R A L
SHOW DIST GRANTS FOR 'user' @ 'host' ;	查看用户权限	FLUSH DIST PRIVILEGES;

1. 赋予全局权限

GRANT DIST privileges 语句用于向用户或角色授权。

```
GRANT DIST SELECT TP 'user'@'host';
```

示例

将全局 SELECT 权限授予用户 'sharding' @ '%'。

```
GRANT DIST SELECT TO 'sharding'@'%';
```

2. 赋予权限（所有对象）示例

```
GRANT DIST privileges ON *.* to 'user'@'host';
```

示例

将所有逻辑库的 SELECT 和 INSERT 权限授予用户 'sharding' @ '%'。

```
GRANT DIST SELECT, INSERT ON *.* TO 'sharding'@'%';
```

3. 赋予权限（库级）

```
GRANT DIST privileges ON schema.* to 'user'@'host';
```

示例

将逻辑库 sharding_db 的 SELECT 和 INSERT 权限授予用户 'sharding' @ '%'。

```
GRANT DIST SELECT, INSERT ON sharding_db.* TO 'sharding'@'%';
```

4. 赋予权限（表级）

```
GRANT DIST privileges ON schema.table to 'user'@'host';
```

示例

将逻辑表 sharding_db.t_order 的 SELECT 和 INSERT 权限授予用户 'sharding' @ '%'。

```
GRANT DIST SELECT, INSERT ON sharding_db.t_order TO 'sharding'@'%';
```

5. 赋予权限（列级）

```
GRANT DIST privileges(column name) ON schema.table to 'user'@'host';
```

示例

将逻辑表 sharding_db.t_order 中指定列的 SELECT 权限授予用户 'sharding' @ '%'。

```
GRANT DIST SELECT (order_id), SELECT (user_id, status) ON sharding_db.t_order TO 'sharding'@'%';
```

6. 撤销所有权限

REVOKE DIST privileges 语句用于撤销对用户或角色的授权。

```
REVOKE DIST ALL FROM 'user'@'host';
```

示例

撤销用户 'sharding' @ '%' 的全局 SELECT 授权。

```
REVOKE DIST ALL FROM 'sharding'@'%';
```

7. 撤销全局权限

```
REVOKE DIST privileges FROM 'user'@'host';
```

示例

撤销用户 'sharding' @ '%' 的全局 SELECT 授权。

```
REVOKE DIST SELECT FROM 'sharding'@'%';
```

撤销角色 dml_only 的全局 SELECT、INSERT、UPDATE 和 DELETE 授权。

```
REVOKE DIST INSERT,SELECT,UPDATE,DELETE FROM dml_only;
```

8. 撤销权限（所有对象）

```
REVOKE DIST privileges ON *.* FROM 'user'@'host';
```

示例

撤销用户 'sharding' @ '%' 在逻辑库 sharding_db 的 SELECT 和 INSERT 授权。

```
REVOKE DIST SELECT, INSERT ON *.* FROM 'sharding'@'%';
```

9. 撤销权限（库级）

```
REVOKE DIST privileges ON schema.* FROM 'user'@'host';
```

示例

撤销用户 'sharding' @ '%' 在逻辑库 sharding_db 的 SELECT 和 INSERT 授权。

```
REVOKE DIST SELECT, INSERT ON sharding_db.* FROM 'sharding'@'%';
```

10. 撤销权限（表级）

```
REVOKE DIST privileges ON schema.table FROM 'user'@'host';
```

示例

撤销用户 'sharding' @ '%' 在逻辑表 sharding_db.t_order 的 SELECT 和 INSERT 授权。

```
REVOKE DIST SELECT, INSERT ON sharding_db.t_order FROM 'sharding'@'%';
```

11. 撤销权限（列级）

```
REVOKE DIST privileges(column name) ON schema.table FROM 'user'@'host';
```

示例

撤销用户 'sharding' @ '%' 在逻辑表 sharding_db.t_order 中指定列的 SELECT 授权。

```
REVOKE DIST SELECT (order_id), SELECT (user_id, status) ON sharding_db.t_order FROM 'sharding'@'%';
```

12. 查看用户权限

```
SHOW DIST GRANTS FOR 'user'@'host';
```

示例

查看用户 'sharding' @ '%' 的权限

```
SHOW DIST GRANTS FOR 'sharding'@'%';
```

13. 刷新授权

```
FLUSH DIST PRIVILEGES;
```

参数说明

无

示例

```
FLUSH DIST PRIVILEGES;
```

12.6.2 计算节点

计算节点

语法	描述	类型
SHOW COMPUTE NODES	查询 proxy 实例信息	RQL
SHOW COMPUTE NODE INFO	查询当前 proxy 的实例信息	RQL
SHOW COMPUTE NODE MODE	查询当前 proxy 的 mode 配置	RQL
ENABLE COMPUTE NODE instanceId	启用 proxy 实例	RAL
DISABLE COMPUTE NODE instanceId	禁用 proxy 实例	RAL
LABEL COMPUTE NODE instanceId WITH labelName [,labelName]	为实例添加标签	RAL
UNLABEL COMPUTE NODE instanceId	移除实例的所有标签	RAL
UNLABEL COMPUTE NODE instanceId WITH labelName	为实例移除指定的标签	RAL
RELABEL COMPUTE NODE instanceId with labelName [, label-Name]	为实例重新添加标签（覆盖原有的标签）	RAL

查询 proxy 实例信息

```
SHOW COMPUTE NODES
```

参数说明

无

示例

```
SHOW COMPUTE NODES
```

输出说明

列	说明
in-stance_id	proxy 实例编号
host	主机地址
port	端口号
status	proxy 实例状态
mode_type	proxy 实例模式
worker_id	每一台计算节点有唯一的 worker_id，启动时自动分配，用于避免集群内计算节点雪花算法 ID 的重复，workerid 默认是 -1 配置分片时自动赋值，用户不能修改
labels	标签
version	版本

启用 proxy 实例

```
ENABLE COMPUTE NODE instanceId
```

示例

```
ENABLE COMPUTE NODE instance_1
```

禁用 proxy 实例

```
DISABLE COMPUTE NODE instanceId
```

示例

```
DISABLE COMPUTE NODE instance_1
```

为实例添加标签

```
LABEL COMPUTE NODE instanceId WITH labelName [ ,labelName ]
```

示例

```
LABEL COMPUTE NODE 10.7.7.136@3309 with olap ,group_by
```

移除实例的所有标签

```
UNLABEL COMPUTE NODE instanceId
```

示例

```
UNLABEL COMPUTE NODE 10.7.7.136@3309
```

为实例移除指定的标签

```
UNLABEL COMPUTE NODE instanceId WITH labelName
```

示例

```
UNLABEL COMPUTE NODE 10.7.7.136@3309 WITH group_by
```

为实例重新添加标签（覆盖原有的标签）

```
RELABEL COMPUTE NODE instanceId with labelName [ , label-Name ]
```

示例

```
RELABEL COMPUTE NODE 10.7.7.136@3309 with label_9, label_0
```


SHOW COMPUTE NODE INFO

描述

SHOW COMPUTE NODE INFO 语法用于查询当前 proxy 实例信息

语法

```
ShowComputeNodeInfo ::=
'SHOW' 'COMPUTE' 'NODE' 'INFO'
```

返回值说明

列	说明
instance_id	proxy 实例编号
host	主机地址
port	端口号
status	proxy 实例状态
mode_type	proxy 实例模式
worker_id	worker id
labels	标签

示例

- 查询当前 proxy 实例信息

```
SHOW COMPUTE NODE INFO;
```

```
mysql> SHOW COMPUTE NODE INFO;
+-----+-----+-----+-----+-----+-----+-----+
| instance_id | host | port | status | mode_type | worker_id | labels |
+-----+-----+-----+-----+-----+-----+-----+
| 734bb036-b15d-4af0-be87-2372d8b6a0cd | 192.168.5.163 | 3307 | OK | Cluster | -1 | |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

保留字

SHOW 、 COMPUTE 、 NODE 、 INFO

相关链接

- 保留字

SHOW COMPUTE NODE MODE

描述

SHOW COMPUTE NODE MODE 语法用于查询当前 proxy 的模式配置信息

语法

```
ShowComputeNodeMode ::=
'SHOW' 'COMPUTE' 'NODE' 'MODE'
```

返回值说明

列	说明
type	proxy 模式类型
repository	proxy 持久化仓库类型
props	proxy 持久化仓库属性参数

示例

- 查询当前 proxy 实例模式配置信息

```
SHOW COMPUTE NODE MODE;
```

```
mysql> SHOW COMPUTE NODE MODE;
+-----+-----+-----+-----+
| type | repository | props |
+-----+-----+-----+-----+
| Cluster | ZooKeeper | {"operationTimeoutMilliseconds":500,"timeToLiveSeconds":60,"maxRetries":3,"namespace":
"governance_ds","server-lists":"localhost:2181","retryIntervalMilliseconds":500} |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

保留字

SHOW 、 COMPUTE 、 NODE 、 MODE

相关链接

- 保留字

12.6.3 存储节点

存储节点

■ 语法列表

语法	描述	类型
REGISTER STORAGE UNIT storageUnitDefinition [, storageUnitDefinition] ...	注册 d b 级别的存储节点	R D L
ALTER STORAGE UNIT storageUnitDefinition [, storageUnitDefinition] ...	调整 d b 级别的存储节点	R D L
UNREGISTER STORAGE UNIT storageUnitName [, storageUnitName] ... [ignore single tables]	注销 d b 级别的存储节点	R D L
SHOW STORAGE UNITS [[FROM databaseName] [WHERE USAGE_COUNT = usageCount]]	查看当前逻辑库下的所有 d b 级别的存储节点	R Q L

1. 注册 db 级别的存储节点

```
REGISTER STORAGE UNIT storageUnitDefinition [, storageUnitDefinition] ...

storageUnitDefinition:
    simpleSource | urlSource

simpleSource:
    storageUnitName(HOST=hostname,PORT=port,DB=dbName,USER=user [,PASSWORD=password] [,PROPERTIES(property [,property]) ...])

urlSource:
    storageUnitName(URL=url,USER=user [,PASSWORD=password] [,PROPERTIES(property [,property]) ...])

property:
    key=value
```

参数说明

名称	数据类型	说明
storageUnitName	IDENTIFIER	存储单元名称
hostname	STRING	数据源地址
port	INT	数据源端口
dbName	STRING	物理库名称
url	STRING	URL 地址
user	STRING	用户名
password	STRING	密码

- 添加存储单元前请确认已经创建分布式数据库，并执行 use 命令成功选择一个数据库；
- 确认将要添加或修改的存储单元是可以正常连接的，否则将不能操作成功；
- 不允许重复的 storageUnitName ；
- PROPERTIES 用于自定义连接池参数，key 和 value 均为 STRING 类型；
- ALTER STORAGE UNIT 修改存储单元时不允许改变该存储单元关联的真实数据源；
- ALTER STORAGE UNIT 修改存储单元时会发生连接池的切换，此操作可能对进行中的业务造成影响，请谨慎使用；
- UNREGISTER STORAGE UNIT 只会删除逻辑存储单元，不会删除真实的数据源；
- 被规则引用的存储单元将无法被删除；

- 若存储单元只被 single table rule 引用，且用户确认可以忽略该限制，则可以添加可选参数 ignore single tables 进行强制删除。

示例

```
REGISTER STORAGE UNIT ds_0 (  
  HOST="127.0.0.1",  
  PORT=3306,  
  DB="db0",  
  USER="root",  
  PASSWORD="root"  
) , ds_1 (  
  HOST="127.0.0.1",  
  PORT=3306,  
  DB="db1",  
  USER="root"  
) , ds_2 (  
  HOST="127.0.0.1",  
  PORT=3306,  
  DB="db2",  
  USER="root",  
  PROPERTIES("maximumPoolSize"="10")  
) , ds_3 (  
  URL="jdbc:mysql://127.0.0.1:3306/db3?serverTimezone=UTC&useSSL=false",  
  USER="root",  
  PASSWORD="root",  
  PROPERTIES("maximumPoolSize"="10","idleTimeout"="30000")  
);
```

2. 调整 db 级别的存储节点

```
ALTER STORAGE UNIT storageUnitDefinition [, storageUnitDefinition] ...
```

storageUnitDefinition:

simpleSource | urlSource

simpleSource:

storageUnitName(**HOST**=hostname,PORT=port,DB=dbName,**USER**=user [,PASSWORD=password] [,PROPERTIES(property [,property]) ...])

urlSource:

storageUnitName(URL=url,**USER**=user [,PASSWORD=password] [,PROPERTIES(property [,property]) ...])

property:

key=value

参数说明

名称	数据类型	说明
storageUnitName	IDENTIFIER	存储单元名称
hostname	STRING	数据源地址
port	INT	数据源端口
dbName	STRING	物理库名称
url	STRING	URL 地址
user	STRING	用户名
password	STRING	密码

示例

```
ALTER STORAGE UNIT ds_0 (  
  HOST="127.0.0.1",
```

```

PORT=3309,
DB="db0",
USER="root",
PASSWORD="root"
),ds_1 (
URL="jdbc:mysql://127.0.0.1:3309/db1?serverTimezone=UTC&useSSL=false",
USER="root",
PASSWORD="root",
PROPERTIES("maximumPoolSize"="10","idleTimeout"="30000")
);

```

3. 注销 db 级别的存储节点

```
UNREGISTER STORAGE UNIT storageUnitName [, storageUnitName] ... [ignore single tables]
```

示例

```

UNREGISTER STORAGE UNIT ds_0, ds_1;
UNREGISTER STORAGE UNIT ds_2, ds_3 ignore single tables;

```

4. 查看当前逻辑库下的所有 db 级别的存储节点

```
SHOW STORAGE UNITS [ [FROM databaseName] ] [ WHERE USAGE_COUNT = usageCount ] ]
```

示例

查看当前逻辑库下的所有 db 级别的存储节点：

```

mysql> SHOW STORAGE UNITS;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| name | type | host | port | db | connection_timeout_milliseconds | idle_timeout_milliseconds | max_lifetime_milliseconds | max_pool_size | min_pool_size | read_only | other_attributes |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ds_0 | MySQL | 127.0.0.1 | 3306 | db_0 | 30000 | 60000 | 1800000 | 50 | 1 | false | {
"dataSourceProperties":{"cacheServerConfiguration":"true","elideSetAutoCommits":"true","useServerPrepStmts":"true",
"cachePrepStmts":"true","rewriteBatchedStatements":"true","cacheResultSetMetadata":"false","useLocalSessionState":"true",
"maintainTimeStats":"false","prepStmtCacheSize":"8192","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"netTimeoutForStreamingResults":"0","zeroDateTimeBehavior":"round"},"healthCheckProperties":{},"initializationFailTimeout":
1,"validationTimeout":5000,"leakDetectionThreshold":0,"poolName":"HikariPool-1","registerMbeans":false,
"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false} |
| ds_1 | MySQL | 127.0.0.1 | 3306 | db_1 | 30000 | 60000 | 1800000 | 50 | 1 | false | {
"dataSourceProperties":{"cacheServerConfiguration":"true","elideSetAutoCommits":"true","useServerPrepStmts":"true",
"cachePrepStmts":"true","rewriteBatchedStatements":"true","cacheResultSetMetadata":"false","useLocalSessionState":"true",
"maintainTimeStats":"false","prepStmtCacheSize":"8192","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"netTimeoutForStreamingResults":"0","zeroDateTimeBehavior":"round"},"healthCheckProperties":{},"initializationFailTimeout":
1,"validationTimeout":5000,"leakDetectionThreshold":0,"poolName":"HikariPool-2","registerMbeans":false,
"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false} |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

2 rows in set (0.84 sec)

查看当前逻辑库下的指定 ID 的 db 级别的存储节点

```
mysql> SHOW STORAGE UNITS FROM sharding_db;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| name | type | host | port | db | connection_timeout_milliseconds | idle_timeout_milliseconds | max_lifetime_milliseconds | max_pool_size | min_pool_size | read_only | other_attributes |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ds_0 | MySQL | 127.0.0.1 | 3306 | db_0 | 30000 | 60000 | 1800000 | 50 | 1 | false | {
"dataSourceProperties":{"cacheServerConfiguration":"true","elideSetAutoCommits":"true","useServerPrepStmts":"true",
"cachePrepStmts":"true","rewriteBatchedStatements":"true","cacheResultSetMetadata":"false","useLocalSessionState":"true",
"maintainTimeStats":"false","prepStmtCacheSize":"8192","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"netTimeoutForStreamingResults":"0","zeroDateTimeBehavior":"round"},"healthCheckProperties":{},"initializationFailTimeout
":1,"validationTimeout":5000,"leakDetectionThreshold":0,"poolName":"HikariPool-1","registerMbeans":false,
"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false} |
| ds_1 | MySQL | 127.0.0.1 | 3306 | db_1 | 30000 | 60000 | 1800000 | 50 | 1 | false | {
"dataSourceProperties":{"cacheServerConfiguration":"true","elideSetAutoCommits":"true","useServerPrepStmts":"true",
"cachePrepStmts":"true","rewriteBatchedStatements":"true","cacheResultSetMetadata":"false","useLocalSessionState":"true",
"maintainTimeStats":"false","prepStmtCacheSize":"8192","tinyInt1isBit":"false","prepStmtCacheSqlLimit":"2048",
"netTimeoutForStreamingResults":"0","zeroDateTimeBehavior":"round"},"healthCheckProperties":{},"initializationFailTimeout
":1,"validationTimeout":5000,"leakDetectionThreshold":0,"poolName":"HikariPool-2","registerMbeans":false,
"allowPoolSuspension":false,"autoCommit":true,"isolateInternalQueries":false} |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.84 sec)
```

输出说明

列	说明
name	数据源名称
type	数据源类型
host	数据源地址
port	数据源端口
db	数据库名称
attribute	数据源参数

5. 查看当前逻辑库下的未被使用的所有 db 级别的存储节点

```
SHOW UNUSED STORAGE UNITS
```

参数说明

无

示例

```
SHOW UNUSED STORAGE UNITS
```

12.6.4 事务

事务

语法	描述	类型
ALTER TRANSACTION RULE(DEFAULT=xx,TYPE(NAME=xxx, PROPERTIES("key1" = "value1" , "key2" = "value2" ...)))	更新事务规则配置	R A L
SHOW TRAFFIC[RULES / RULE ruleName]	查询事务规则配置	R Q L

1. 更新事务规则配置

```
ALTER TRANSACTION RULE(DEFAULT=xx,TYPE(NAME=xxx, PROPERTIES( "key1"="value1" , "key2"="value2" ...)))
```

参数说明

DEFAULT: 默认事务类型, 支持 LOCAL、XA、BASE; NAME: 事务管理器名称, 支持 Atomikos、Narayana 和 Bitronix

示例

```
ALTER TRANSACTION RULE(DEFAULT=XA,TYPE(NAME=Narayana, PROPERTIES( "databaseName"="jbossts" , "host"="127.0.0.1" )))
```

2. 查询事务规则配置

```
SHOW TRAFFIC[ RULES / RULE ruleName]
```

参数说明

无

示例

```
SHOW SQL_PARSER RULES;
```

12.6.5 SQL_Parser

SQL_Parser

语法	描述	类型
ALTER SQL_PARSER RULE SQL_COMMENT_PARSE_ENABLE=xx, PARSE_TREE_CACHE (INITIAL_CAPACITY=xx, MAXIMUM_SIZE=xx, CONCURRENCY_LEVEL=xx), SQL_STATEMENT_CACHE (INITIAL_CAPACITY=xxx, MAXIMUM_SIZE=xxx, CONCURRENCY_LEVEL=xxx)	更新解析引擎规则配置	RAL
SHOW SQL_PARSER RULE	查询解析引擎规则配置	RQL

1. 更新解析引擎规则配置

```
ALTER SQL_PARSER RULE SQL_COMMENT_PARSE_ENABLE=xx, PARSE_TREE_CACHE (INITIAL_CAPACITY=xx, MAXIMUM_SIZE=xx, CONCURRENCY_LEVEL=xx), SQL_STATEMENT_CACHE (INITIAL_CAPACITY=xxx, MAXIMUM_SIZE=xxx, CONCURRENCY_LEVEL=xxx)
```

参数说明

SQL_COMMENT_PARSE_ENABLE：是否使用字符串截取的方式解析 SQL 注释，PARSE_TREE_CACHE：语法树本地缓存配置，SQL_STATEMENT_CACHE：SQL 语句本地缓存配置项

示例

```
ALTER SQL_PARSER RULE SQL_COMMENT_PARSE_ENABLE=false, PARSE_TREE_CACHE (INITIAL_CAPACITY=10, MAXIMUM_SIZE=11, CONCURRENCY_LEVEL=1), SQL_STATEMENT_CACHE (INITIAL_CAPACITY=11, MAXIMUM_SIZE=11, CONCURRENCY_LEVEL=100)
```

2. 查询解析引擎规则配置

```
SHOW SQL_PARSER RULE
```

参数说明

无

示例

```
SHOW SQL_PARSER RULE;
```

12.6.6 License

License

语法	描述	类型
REGISTER LICENSE	注册 License	RDL
SHOW LICENSE INFO	查看 License	RQL

1. 注册 license

REGISTER LICENSE

示例

使用注册码注册 license

REGISTER LICENSE 'license-key';

使用 license 文件注册 license

REGISTER LICENSE '直接路径/license-file';

2. 查看 License

执行如下命令查看 license。

SHOW LICENSE INFO;

示例

查看 license。

SHOW LICENSE INFO;

使用注册码文件预览 license。

SHOW LICENSE INFO 'license-key';

使用 license 文件预览 license。

SHOW LICENSE INFO '直接路径/license-file';

12.6.7 Traffic

Traffic

语法	描述	类型
CREATE TRAFFIC RULE sql_match_traffic (LABELS(xxx),TRAFFI C_ALGORITHM(TYPE(NAME=xxx,PROPERTIES("key" = "value")), LOAD_BALANCER(TYPE(NAME=xxx,PROPERT IES("key" = "value"))))	创建双路由规则	RDL
ALTER TRAFFIC RULE sql_match_traffic (LABELS(xxx),TRAFFI C_ALGORITHM(TYPE(NAME=xxx,PROPERTIES("key" = "val ue"))),LOAD_BALANCER(TYPE(NAME=xxx,PROPERT IES("key" = "value"))))	修改双路由规则	RDL
DROP TRAFFIC RULE ruleName [, ruleName]	删除双路由规则	RDL
SHOW TRAFFIC RULES	查询双路由规则	RQL

1. 创建双路由规则

```
CREATE TRAFFIC RULE sql_match_traffic (LABELS(xxx),TRAFFIC_ALGORITHM(TYPE(NAME=xxx,PROPERTIES( "key" = "value" ))),  
LOAD_BALANCER(TYPE(NAME=xxx,PROPERTIES( "key"="value" ))))
```

参数说明

TRAFFIC_ALGORITHM 支持 SQL_MATCH 与 SQL_HINT 两种类型 LOAD_BALANCER 支持 RANDOM 与 ROUND_ROBIN 两种类型。

示例

```
CREATE TRAFFIC RULE sql_match_traffic (LABELS(OLTP),TRAFFIC_ALGORITHM(TYPE(NAME=SQL_MATCH,PROPERTIES( "sql" =  
"SELECT * FROM t_order WHERE order_id = ?; UPDATE t_order SET order_id = ?;"))),LOAD_BALANCER(TYPE(NAME=RANDOM)))
```

2. 修改双路由规则

```
ALTER TRAFFIC RULE sql_match_traffic (LABELS(xxx),TRAFFIC_ALGORITHM(TYPE(NAME=xxx,PROPERTIES( "key" = "value" ))),  
LOAD_BALANCER(TYPE(NAME=xxx,PROPERTIES( "key"="value" ))))
```

参数说明

TRAFFIC_ALGORITHM 支持 SQL_MATCH 与 SQL_HINT 两种类型 LOAD_BALANCER 支持 RANDOM 与 ROUND_ROBIN 两种类型。

示例

```
ALTER TRAFFIC RULE sql_match_traffic (LABELS(OLTP),TRAFFIC_ALGORITHM(TYPE(NAME=SQL_MATCH,PROPERTIES( "sql" =  
"SELECT * FROM t_order WHERE order_id = ?; UPDATE t_order SET order_id = ?;"))), LOAD_BALANCER(TYPE(NAME=RANDOM)))
```

3. 删除双路由规则

```
DROP TRAFFIC RULE ruleName [, ruleName]
```

示例

```
DROP TRAFFIC RULE sql_match_traffic
```

4. 查询双路由规则

```
SHOW TRAFFIC RULES;
```

示例

查询指定双路由规则或所有双路由规则

```
SHOW TRAFFIC RULES;
```

12.6.8 数据脱敏

本章节将对数据脱敏的语法进行详细说明。

CREATE MASK RULE

描述

The CREATE MASK RULE 语法用于创建数据脱敏规则。

语法定义

```
{{< tabs >}} {{< tab "语法" >}}
```

```
CreateEncryptRule ::=
'CREATE' 'MASK' 'RULE' ifNotExists? maskRuleDefinition (',' maskRuleDefinition)*
```

```
ifNotExists ::=
'IF' 'NOT' 'EXISTS'
```

```
maskRuleDefinition ::=
ruleName '(' 'COLUMNS' '(' columnDefinition (',' columnDefinition)* ')' ')'
```

```
columnDefinition ::=
'(' 'NAME' '=' columnName ',' maskAlgorithmDefinition ')'
```

```
maskAlgorithmDefinition ::=
'TYPE' '(' 'NAME' '=' maskAlgorithmType (',' propertiesDefinition)? ')'
```

```
propertiesDefinition ::=
'PROPERTIES' '(' key '=' value (',' key '=' value)* ')'
```

```
ruleName ::=
identifier
```

```
columnName ::=
identifier
```

```
maskAlgorithmType ::=
literal
```

```
key ::=
string
```

```
value ::=
literal
```

```
{{< /tab >}} {{< tab "铁路图" >}} {{< /tab >}} {{< /tabs >}}
```

补充说明

- maskAlgorithmType 指定数据脱敏算法类型，请参考 [数据脱敏算法](#)；
- 重复的 ruleName 将无法被创建；
- ifNotExists 子句用于避免出现 Duplicate mask rule 错误。

示例

创建数据脱敏规则

```
CREATE MASK RULE t_mask (  
  COLUMNS(  
    (NAME=phone_number,TYPE(NAME='MASK_FROM_X_TO_Y', PROPERTIES("from-x"=1, "to-y"=2, "replace-char"="*")),  
    (NAME=address,TYPE(NAME='MD5'))  
  ));
```

使用 ifNotExists 子句创建数据脱敏规则

```
CREATE MASK RULE IF NOT EXISTS t_mask (  
  COLUMNS(  
    (NAME=phone_number,TYPE(NAME='MASK_FROM_X_TO_Y', PROPERTIES("from-x"=1, "to-y"=2, "replace-char"="*")),  
    (NAME=address,TYPE(NAME='MD5'))  
  ));
```

保留字

CREATE 、 MASK 、 RULE 、 COLUMNS 、 NAME 、 TYPE

ALTER MASK RULE

描述

ALTER MASK RULE 语法用于修改数据脱敏规则。

语法定义

{{< tabs >}} {{<tab “语法” >}}

```
CreateEncryptRule ::=  
  'ALTER' 'MASK' 'RULE' maskRuleDefinition (',' maskRuleDefinition)*  
  
maskRuleDefinition ::=  
  ruleName '(' 'COLUMNS' '(' columnDefinition (',' columnDefinition)* ')' ')' '  
  
columnDefinition ::=  
  '(' 'NAME' '=' columnName ',' maskAlgorithmDefinition ')' '  
  
maskAlgorithmDefinition ::=  
  'TYPE' '(' 'NAME' '=' maskAlgorithmType (',' propertiesDefinition)? ')' '  
  
propertiesDefinition ::=
```

```
'PROPERTIES' '(' key '=' value (',' key '=' value)* ')'  
  
ruleName ::=  
  identifier  
  
columnName ::=  
  identifier  
  
maskAlgorithmType ::=  
  literal  
  
key ::=  
  string  
  
value ::=  
  literal
```

```
{{</tab>}} {{< tab "铁路图" >}} {{</tab>}} {{</tabs>}}
```

补充说明

- maskAlgorithmType 指定数据脱敏算法类型，请参考 [数据脱敏算法](#)。

示例

修改数据脱敏规则

```
ALTER MASK RULE t_mask (  
  COLUMNS(  
    (NAME=phone_number,TYPE(NAME='MASK_FROM_X_TO_Y', PROPERTIES("from-x"=1, "to-y"=2, "replace-char"="*")),  
    (NAME=address,TYPE(NAME='MD5'))  
  ));
```

保留字

ALTER、MASK、RULE、COLUMNS、NAME、TYPE

DROP MASK RULE

说明

DROP MASK RULE 语法用于删除数据脱敏规则

语法

```
{{< tabs >}} {{< tab  “语法” >}}

DropEncryptRule ::=
'DROP' 'MASK' 'RULE' ifExists? maskRuleName (',' maskRuleName)*

ifExists ::=
'IF' 'EXISTS'

maskRuleName ::=
identifier

{{< /tab >}} {{< tab  “铁路图” >}} {{< /tab >}} {{< /tabs >}}
```

补充说明

- ifExists 子句用于避免 Mask rule not exists 错误。

示例

- 删除数据脱敏规则

```
DROP MASK RULE t_mask, t_mask_1;
```

- 使用 ifExists 子句删除数据脱敏规则

```
DROP MASK RULE IF EXISTS t_mask, t_mask_1;
```

保留字

DROP, MASK, RULE

12.6.9 参数

参数 Variables

语法	描述	类型
SHOW DIST VARIABLES	查询 proxy 所有的属性	RQL
SHOW DIST VARIABLE WHERE name = variable_name	查询指定的 proxy 属性	RQL
SET DIST VARIABLE variable_name = xx	配置属性	RAL

1. 查询 proxy 所有的属性

```
SHOW DIST VARIABLES
```

参数说明

无

示例

SHOW DIST VARIABLES

2. 查询指定的 proxy 属性

SHOW DIST VARIABLE WHERE name = variable_name

示例

SHOW DIST VARIABLE WHERE name = sql_show;

3. 配置属性

SET DIST VARIABLE variable_name = xx

示例

SET DIST VARIABLE sql_show = **true**

集群参数列表请参考：[集群参数](#)；

12.6.10 元数据

元数据 Metadata

语法	描述	类型
SHOW TABLE METADATA tableName [, tableName] ...。	查询表的元数据	R Q L
REFRESH TABLE METADATA	刷新表的元数据	R U L
REFRESH DATABASE METADATA schemaName FROM GOVERNANCE CENTER	刷新 z 指定逻辑库的元数据	R U L
IMPORT DATABASE CONFIGURATION	将 Y A M L 中的配置导入到 database 中，仅支持对空库进行导入操作	R U L
EXPORT DATABASE CONFIGURATION	将 database 中的数据源和规则配置导出为 Y A M L 格式	R U L

1. 查询表的元数据

SHOW TABLE METADATA tableName [, tableName] ...

参数说明

无

示例

SHOW TABLE METADATA t_order;

2. 刷新表的元数据

```
REFRESH TABLE METADATA
```

示例

刷新所有表的元数据

```
REFRESH TABLE METADATA
```

刷新指定表的元数据

```
REFRESH TABLE METADATA t_order
```

刷新指定数据源中表的元数据

```
REFRESH TABLE METADATA t_order FROM STORAGE UNIT ds_1
```

刷新指定 schema 中表的元数据，如果 schema 中不存在表，则会删除该 schema

```
REFRESH TABLE METADATA FROM STORAGE UNIT ds_1 SCHEMA db_schema
```

3. 刷新指定逻辑库的元数据

```
REFRESH DATABASE METADATA schemaName FROM GOVERNANCE CENTER
```

示例

```
REFRESH DATABASE METADATA sharding_db FROM GOVERNANCE CENTER;
```

4. 将 YAML 中的配置导入到 database 中，仅支持对空库进行导入操作

```
IMPORT DATABASE CONFIGURATION
```

示例

```
IMPORT DATABASE CONFIGURATION FROM FILE "/xx x/config-sharding.yaml";
```

5. 将 database 中的数据源和规则配置导出为 YAML 格式

```
EXPORT DATABASE CONFIGURATION
```

示例

```
EXPORT DATABASE CONFIGURATION TO FILE "/xxx/config-sharding.yaml";
```


12.6.11 单表

单表 Single Table

语法	描述	类型
COUNT SINGLE_TABLE RULE	查询单表的数量，在后续版本中将废弃此语法	RQL
SHOW SINGLE TABLES	查看所有的单表	RQL
SET DEFAULT SINGLE TABLE STORAGE UNIT = xx	设置单表的存储节点	RAL
SHOW DEFAULT SINGLE TABLE STORAGE UNIT	查看单表默认存储节点配置	RQL

1. 查询单表的数量

```
COUNT SINGLE_TABLE RULE;
```

示例

```
COUNT SINGLE_TABLE RULE;
```

2. 查看所有的单表

```
SHOW SINGLE TABLES
```

示例

```
SHOW SINGLE TABLES
```

3. 设置单表的存储节点

```
SET DEFAULT SINGLE TABLE STORAGE UNIT = (storageUnitName | RANDOM)
```

参数说明

storageUnitName 需使用 RDL 管理的存储单元。RANDOM 代表随机存储。

示例

```
SET DEFAULT SINGLE TABLE STORAGE UNIT = ds_0
```

```
SET DEFAULT SINGLE TABLE STORAGE UNIT = RANDOM
```

4. 查看单表默认存储节点配置

```
SHOW DEFAULT SINGLE TABLE STORAGE UNIT;
```

示例

```
SHOW DEFAULT SINGLE TABLE STORAGE UNIT;
```

12.6.12 分片表

分片表

语法	描述	类型
CREATE SHARDING TABLE RULE shardingTableRuleDefinition [, shardingTableRuleDefinition] ...	创建分片规则	RDL
ALTER SHARDING TABLE RULE shardingTableRuleDefinition [, shardingTableRuleDefinition] ...	修改分片规则	RDL
DROP SHARDING TABLE RULE tableName [, tableName] ...	删除分片规则	RDL
SHOW SHARDING TABLE tableRule RULES [FROM databaseName]	查看分片规则	RQL
SHOW SHARDING TABLE NODES	查看逻辑库中分片表所在的分片规则	RQL
COUNT SHARDING RULE	统计分片规则的数量，在后续版本中将废弃此语法	RQL
DROP SHARDING ALGORITHM algorithmName [, algorithmName] ...	单独删除分片算法无法删除使用中的算法	RDL
SHOW SHARDING ALGORITHMS [FROM databaseName]	查看分片算法详情	RQL
SHOW UNUSED SHARDING ALGORITHMS [FROM databaseName]	查看未被使用的分片算法	RQL
SHOW SHARDING TABLE RULES USED ALGORITHM shardingAlgorithmName [FROM databaseName]	查看使用指定分片算法的表	RQL
DROP SHARDING KEY GENERATOR [IF EXISTS] keyGeneratorName [, keyGeneratorName] ...	删除分布式主键生成算法	RDL
SHOW SHARDING KEY GENERATORS [FROM databaseName]	SHOW UNUSED SHARDING KEY GENERATORS [FROM databaseName]	查看未被使用的分布式主键生成算法
SHOW SHARDING TABLE RULES USED KEY GENERATOR keyGeneratorName [FROM databaseName]	查看使用指定算法的表	RQL
CREATE DEFAULT SHARDING shardingScope STRATEGY (shardingStrategy)	创建缺省的分库分表策略	RDL
ALTER DEFAULT SHARDING shardingScope STRATEGY (shardingStrategy)	修改缺省的分库分表策略	RDL
DROP DEFAULT SHARDING shardingScope STRATEGY;	删除缺省的分库分表策略	RDL
SHOW DEFAULT SHARDING STRATEGY	查看缺省的分库分表策略的详情	RQL

1. 创建分片规则

CREATE SHARDING TABLE RULE shardingTableRuleDefinition [, shardingTableRuleDefinition] ...

shardingTableRuleDefinition:

shardingAutoTableRule | shardingTableRule

shardingAutoTableRule:

tableName(storageUnits, shardingColumn, algorithmDefinition [, keyGenerateDefinition] [, auditDeclaration])

shardingTableRule:

tableName(dataNodes [, databaseStrategy] [, tableStrategy] [, keyGenerateDefinition] [, auditDeclaration])

storageUnits:

STORAGE_UNITS(storageUnit [, storageUnit] ...)

dataNodes:

DATANODES(dataNode [, dataNode] ...)

```

storageUnit:
    storageUnitName | inlineExpression

dataNode:
    dataNodeName | inlineExpression

shardingColumn:
    SHARDING_COLUMN=columnName

algorithmDefinition:
    TYPE(NAME=shardingAlgorithmType [, PROPERTIES([algorithmProperties]))

keyGenerateDefinition:
    KEY_GENERATE_STRATEGY(COLUMN=columnName, strategyDefinition)

auditDeclaration:
    auditDefinition | auditStrategy

auditDefinition:
    AUDIT_STRATEGY([(singleAuditDefinition),(singleAuditDefinition)], ALLOW_HINT_DISABLE=true)

singleAuditDefinition:
    NAME=auditor1, algorithmDefinition

auditStrategy:
    AUDIT_STRATEGY(AUDITORS=[auditor1,auditor2], ALLOW_HINT_DISABLE=true)

```

参数说明

- STORAGE_UNITS 需使用 RDL 管理的数据源资源；
- shardingAlgorithmType 指定自动分片算法类型，请参考 [自动分片算法](#)；
- keyGenerateStrategyType 指定分布式主键生成策略，请参考 [分布式主键](#)；
- auditorAlgorithmType 指定分片审计策略，请参考 [分片审计](#)；
- 重复的 tableName 将无法被创建；
- shardingAlgorithm 能够被不同的 Sharding Table Rule 复用，因此在执行 DROP SHARDING TABLE RULE 时，对应的 shardingAlgorithm 不会被移除；
- 如需移除 shardingAlgorithm，请执行 DROP SHARDING ALGORITHM；
- 创建 sharding rule 后自动生成的算法命名规则为 tableName _ strategyType _ algorithmType；
- 创建 sharding rule 后自动生成的主键策略命名规则为 tableName _ strategyType；
- Sharding Table Rule 同时支持 Auto Table 和 Table 两种类型，两者在语法上有所差异；
- 执行 CREATE SHARDING TABLE RULE 时，将会自动创建新的分片算法，算法命名规则为 tableName_scope_shardingAlgorithmType，如 t_order_database_inline。
- DATABASE_STRATEGY 为 “NONE” 时不做分库策略

示例

创建分表 rule 使用默认的分库策略

```

CREATE SHARDING TABLE RULE t_order (
    STORAGE_UNITS(ds_0,ds_1),
    SHARDING_COLUMN=order_id,TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="4")),
    KEY_GENERATE_STRATEGY(COLUMN=another_id,TYPE(NAME="snowflake"))
);

```

创建分表 rule 不做分库策略

```
CREATE SHARDING TABLE RULE t_item (
  DATANODES("ds_0.t_item_${0..1}"),
  DATABASE_STRATEGY(TYPE="NONE"),
  TABLE_STRATEGY(TYPE="standard",SHARDING_COLUMN=order_id,SHARDING_ALGORITHM(TYPE(NAME="inline",PROPERTIES(
    "algorithm-expression"="t_item_${order_id % 2}"))))
);
```

2. 修改分片规则

```
ALTER SHARDING TABLE RULE shardingTableRuleDefinition [, shardingTableRuleDefinition] ...
```

shardingTableRuleDefinition:

shardingAutoTableRule | shardingTableRule

shardingAutoTableRule:

tableName(storageUnits, shardingColumn, algorithmDefinition [, keyGenerateDefinition] [, auditDeclaration])

shardingTableRule:

tableName(dataNodes [, databaseStrategy] [, tableStrategy] [, keyGenerateDefinition] [, auditDeclaration])

storageUnits:

STORAGE_UNITS(storageUnit [, storageUnit] ...)

dataNodes:

DATANODES(dataNode [, dataNode] ...)

storageUnit:

storageUnitName | inlineExpression

dataNode:

dataNodeName | inlineExpression

shardingColumn:

SHARDING_COLUMN=columnName

algorithmDefinition:

TYPE(NAME=shardingAlgorithmType [, PROPERTIES([algorithmProperties]))

keyGenerateDefinition:

KEY_GENERATE_STRATEGY(COLUMN=columnName, strategyDefinition)

auditDeclaration:

auditDefinition | auditStrategy

auditDefinition:

AUDIT_STRATEGY([(singleAuditDefinition),(singleAuditDefinition)], ALLOW_HINT_DISABLE=true)

singleAuditDefinition:

NAME=auditor1, algorithmDefinition

auditStrategy:

AUDIT_STRATEGY(AUDITORS=[auditor1,auditor2], ALLOW_HINT_DISABLE=true)

参数说明

- STORAGE_UNITS 需使用 RDL 管理的数据源资源；
- shardingAlgorithmType 指定自动分片算法类型，请参考 [自动分片算法](#)；
- keyGenerateStrategyType 指定分布式主键生成策略，请参考 [分布式主键](#)；
- auditorAlgorithmType 指定分片审计策略，请参考 [分片审计](#)；

- ### 示例

```
ALTER SHARDING TABLE RULE t_order (
  STORAGE_UNITS(ds_0,ds_1,ds_2,ds_3),
  SHARDING_COLUMN=order_id,TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="16")),
  KEY_GENERATE_STRATEGY(COLUMN=another_id,TYPE(NAME="snowflake"))
);
```

3. 删除分片规则

```
DROP SHARDING TABLE RULE t_order_item;
```

示例

删除规则名为 t_order_item 的分片规则

DROP SHARDING TABLE RULE t order item;

4. 查看分片规则

SHOW SHARDING TABLE tableRule | **RULES** [**FROM** databaseName]

示例

查看逻辑库中所有表的分片规则

[illegible]

3 rows in set (0.02 sec)

查看逻辑库中制定表的分片规则

```
mysql> SHOW SHARDING TABLE RULE t_order;
+-----+-----+-----+-----+-----+-----+-----+-----+
| table | actual_data_nodes | actual_data_sources | database_strategy_type | database_sharding_column | database_sharding_algorithm_type | database_sharding_algorithm_props | table_strategy_type | table_sharding_column | table_sharding_algorithm_type | table_sharding_algorithm_props | key_generate_column | key_generator_type | key_generator_props | auditor_types | allow_hint_disable |
+-----+-----+-----+-----+-----+-----+-----+-----+
| t_order | ds_{0..1}.t_order_{0..1} | | INLINE | user_id | INLINE | algorithm-expression:ds_{user_id % 2} | order_id | INLINE | algorithm-expression:t_order_{order_id % 2} | order_id | SNOWFLAKE | | DML_SHARDING_CONDITIONS | true | |
```

1 row in set (0.01 sec)

输出说明

列	说明
table	逻辑表名
actual_data_nodes	实际的数据节点
actual_data_sources	实际的数据源（通过 RDL 创建的规则时显示）
database_strategy_type	数据库分片策略类型
database_sharding_column	数据库分片键
database_sharding_algorithm_type	数据库分片算法类型
database_sharding_algorithm_props	数据库分片算法参数
table_strategy_type	表分片策略类型
table_sharding_column	表分片键
table_sharding_algorithm_type	表分片算法类型
table_sharding_algorithm_props	表分片算法参数
key_generate_column	分布式主键生成列
key_generator_type	分布式主键生成器类型
key_generator_props	分布式主键生成器参数
auditor_types	分片审计生成器参数
allow_hint_disable	是否禁用分片审计 hint

5. 查看逻辑库中分片表所在的分片规则

```
SHOW SHARDING TABLE NODES;
```

参数说明

无

示例

```
mysql> SHOW SHARDING TABLE NODES;
+-----+-----+
| name | nodes |
+-----+-----+
| t_order | ds_0.t_order_0, ds_1.t_order_1, ds_0.t_order_2, ds_1.t_order_3 |
+-----+-----+
```

1 row in set (0.02 sec)

输出说明:

列	说明
name	分片规则名称
nodes	分片节点

6. 单独删除分片算法

```
DROP SHARDING ALGORITHM algorithmName [, algorithmName] ...
```

示例

删除名称 t_order_hash_mod 的分片算法

```
DROP SHARDING ALGORITHM t_order_hash_mod;
```

7. 查看分片算法详情

```
SHOW SHARDING ALGORITHMS;
```

参数说明

无

示例

```
mysql> SHOW SHARDING ALGORITHMS;
+-----+-----+-----+-----+
| name      | type | props                                     |
| t_order_inline | INLINE | algorithm-expression=t_order_{$order_id % 2} || t_order_item_inline | INLINE | algorithm-
expression=t_order_item_{$order_id % 2} |
+-----+-----+-----+-----+
2 row in set (0.01 sec)
```

输出说明

列	说明
name	分片算法名称
type	分片算法类型
props	分片算法参数

8. 查看未被使用的分片算法

```
SHOW UNUSED SHARDING ALGORITHMS;
```

参数说明

无

示例

```
mysql> SHOW UNUSED SHARDING ALGORITHMS;
+-----+-----+-----+-----+
| name      | type | props                                     |
| t1_inline | INLINE | algorithm-expression=t_order_{$order_id % 2} |
+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

输出说明

列	说明
name	分片算法名称
type	分片算法类型
props	分片算法参数

9. 查看使用指定分片算法的表

```
SHOW SHARDING TABLE RULES USED ALGORITHM shardingAlgorithmName [FROM databaseName]
```

示例

```
mysql> SHOW SHARDING TABLE RULES USED ALGORITHM t_order_inline;
+-----+-----+
| type | name |
+-----+-----+
| table | t_order |
1 row in set (0.01 sec)
```

输出说明

列	说明
type	分片算法类型
name	分片算法名称（与表名相同）

10. 删分布式主键生成算法

```
DROP SHARDING KEY GENERATOR [IF EXISTS] keyGeneratorName [, keyGeneratorName] ...
```

示例

```
DROP SHARDING KEY GENERATOR snowflake_key_generator;
```

11. 查看分布式主键生成算法

```
SHOW SHARDING KEY GENERATORS;
```

参数说明

无

示例

```
mysql> SHOW SHARDING KEY GENERATORS;
+-----+-----+-----+
| name          | type      | props      |
+-----+-----+-----+
| t_order_snowflake | snowflake |          |
| t_order_item_snowflake | snowflake |          |
| uuid_key_generator | uuid      |          |
3 row in set (0.01 sec)
```


输出说明

列	说明
name	主键生成器名称
type	主键生成器类型
props	主键生成器参数

12. 查看未被使用的分布式主键生成算法

```
SHOW UNUSED SHARDING KEY GENERATORS;
```

参数说明

无

示例

```
mysql> SHOW UNUSED SHARDING KEY GENERATORS;
```

```
+-----+-----+-----+
| name      | type  | props  |
+-----+-----+-----+
| dml_audit | uuid  |        |
+-----+-----+-----+
1 row in set (0.01 sec)
```

输出说明

列	说明
name	主键生成器名称
type	主键生成器类型
props	主键生成器参数

13. 查看使用指定算法的表

```
SHOW SHARDING TABLE RULES USED KEY GENERATOR keyGeneratorName [FROM databaseName]
```

示例

```
mysql> SHOW SHARDING TABLE RULES USED KEY GENERATOR t_order_snowflake;
```

```
+-----+-----+
| type | name |
+-----+-----+
| table | t_order |
+-----+-----+
1 row in set (0.01 sec)
```

输出说明

列	说明
type	分片算法类型（分库算法或者分表算法）
name	分片规则名称

14. 创建缺省的分库分表策略

```
CREATE DEFAULT SHARDING shardingScope STRATEGY (shardingStrategy)
```

shardingScope:

DATABASE | **TABLE**

databaseStrategy:

DATABASE_STRATEGY(shardingStrategy)

tableStrategy:

TABLE_STRATEGY(shardingStrategy)

shardingStrategy:

TYPE=strategyType, shardingColumn, shardingAlgorithm

shardingColumn:

SHARDING_COLUMN=columnName

shardingAlgorithm:

SHARDING_ALGORITHM(algorithmDefinition)

strategyDefinition:

TYPE(NAME=keyGenerateStrategyType [, **PROPERTIES**([algorithmProperties])])

参数说明

- strategyType 指定分片策略
- 执行 **CREATE DEFAULT SHARDING STRATEGY** 时，同样会自动创建新的分片算法，算法命名规则为 default_scope_shardingAlgorithmType，如 default_database_inline。

示例

```
CREATE DEFAULT SHARDING DATABASE STRATEGY (  
TYPE="standard",SHARDING_COLUMN=order_id,SHARDING_ALGORITHM(TYPE(NAME="inline",PROPERTIES("algorithm-  
expression"="ds_${order_id % 2}")))  
);
```

15. 修改缺省的分库分表策略

```
ALTER DEFAULT SHARDING shardingScope STRATEGY (shardingStrategy)
```

shardingScope:

DATABASE | **TABLE**

databaseStrategy:

DATABASE_STRATEGY(shardingStrategy)

tableStrategy:

TABLE_STRATEGY(shardingStrategy)

shardingStrategy:

TYPE=strategyType, shardingColumn, shardingAlgorithm

shardingColumn:

SHARDING_COLUMN=columnName

shardingAlgorithm:

SHARDING_ALGORITHM(algorithmDefinition)

```
strategyDefinition:
  TYPE(NAME=keyGenerateStrategyType [, PROPERTIES([algorithmProperties]))]
```

参数说明

- strategyType 指定分片策略
- 执行 CREATE DEFAULT SHARDING STRATEGY 时，同样会自动创建新的分片算法，算法命名规则为 default_scope_shardingAlgorithmType，如 default_database_inline。

示例

```
ALTER DEFAULT SHARDING DATABASE STRATEGY (
  TYPE="standard",SHARDING_COLUMN=order_id,SHARDING_ALGORITHM(TYPE(NAME="inline",PROPERTIES("algorithm-expression"="ds_${order_id % 2}")))
);
```

16. 删除缺省的分库分表策略

```
DROP DEFAULT SHARDING shardingScope STRATEGY

shardingScope:
  DATABASE | TABLE
```

示例

```
DROP DEFAULT SHARDING DATABASE STRATEGY;
```

17. 查看缺省的分库分表策略的详情

```
SHOW DEFAULT SHARDING STRATEGY;
```

参数说明

无

示例

```
mysql> SHOW DEFAULT SHARDING STRATEGY;

+-----+-----+-----+-----+-----+-----+-----+
| name | type | sharding_column | sharding_algorithm_name | sharding_algorithm_type | sharding_algorithm_props |
+-----+-----+-----+-----+-----+-----+-----+
| TABLE | NONE | | | | |
| DATABASE | STANDARD | order_id | database_inline | INLINE | {algorithm-expression=ds_${user_id % 2}} |
+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.07 sec)
```

输出说明

列	说明
name	策略名称
type	分片策略类型
sharding_column	分片键
sharding_algorithm_name	分片算法名称
sharding_algorithm_type	分片算法类型
sharding_algorithm_props	分片算法参数

12.6.13 广播表

广播表

语法	描述	类型
CREATE BROADCAST TABLE RULE tableName [, tableName] ...	创建广播表	RDL
DROP BROADCAST TABLE RULES tableName [, tableName] ...	删除广播表	RDL
SHOW BROADCAST TABLE RULES [FROM databaseName]	查看广播表	RQL

1. 创建广播表

```
CREATE BROADCAST TABLE RULE tableName [, tableName] ...
```

参数说明

名称	数据类型	说明
tableName	IDENTIFIER	表名

示例

```
CREATE BROADCAST TABLE RULE t_a,t_b;
```

2. 删除分片表

```
DROP BROADCAST TABLE RULES tableName [, tableName] ...
```

参数说明

名称	数据类型	说明
tableName	IDENTIFIER	表名

示例

```
DROP BROADCAST TABLE RULE t_a;
```

3. 查看广播表

```
SHOW BROADCAST TABLE RULES [FROM databaseName]
```

参数说明

名称	数据类型	说明
databaseName	IDENTIFIER	逻辑库名称

示例

```
mysql> SHOW BROADCAST TABLE RULES;
+-----+
| broadcast_table |
+-----+
| t_1             |
| t_2             |
```

+-----+

2 rows in set (0.00 sec)

输出说明

列	说明
broadcast_table	广播表名称

12.6.14 表关联

表关联

语法	描述	类型
CREATE SHARDING TABLE REFERENCE RULE tableReferenceRuleDefinition [, tableReferenceRuleDefinition] ...	创建分片表关联	RDL
ALTER SHARDING TABLE REFERENCE RULE tableReferenceRuleDefinition [, tableReferenceRuleDefinition] ...	修改分片表关联	RDL
ALTER SHARDING BINDING TABLE RULE TO	仅 SphereEx-Console 使用，文档中未做说明	RDL
DROP SHARDING TABLE REFERENCE RULE tableReferenceRuleDefinition [, tableReferenceRuleDefinition] ...	删除表关联	RDL
SHOW SHARDING TABLE REFERENCE RULES [FROM databaseName]	查看表关联	RQL
SHOW SHARDING TABLE REFERENCE RULE ruleName	1.3.0 版本中实现	RQL

1. 创建分片表关联

```
CREATE SHARDING TABLE REFERENCE RULE tableReferenceRuleDefinition [, tableReferenceRuleDefinition] ...
tableReferenceRuleDefinition:
    (tableName [, tableName] ... )
```

示例

```
CREATE SHARDING TABLE REFERENCE RULE (t_order,t_order_item),(t_1,t_2);
```

2. 修改分片表关联

```
ALTER SHARDING TABLE REFERENCE RULE tableReferenceRuleDefinition [, tableReferenceRuleDefinition] ...
tableReferenceRuleDefinition:
    (tableName [, tableName] ... )
```

示例

```
ALTER SHARDING TABLE REFERENCE RULE (t_order,t_order_item);
```

3. 删除分片表关联

```

DROP SHARDING TABLE REFERENCE RULE tableReferenceRuleDefinition [, tableReferenceRuleDefinition] ...

tableReferenceRuleDefinition:
    (tableName [, tableName] ... )
    
```

示例
 删除所有表关联

```

DROP SHARDING TABLE REFERENCE RULE;
    
```

删除指定表关联

```

DROP SHARDING TABLE REFERENCE RULE (t_order,t_order_item);
    
```

4. 查看表关联

```

SHOW SHARDING TABLE REFERENCE RULES;
    
```

参数说明

无

示例

```

mysql> SHOW SHARDING TABLE REFERENCE RULES;
+-----+
| sharding_table_reference |
+-----+
| t_order,t_order_item    |
| t1,t2                   |
+-----+
2 rows in set (0.00 sec)
    
```

输出说明

列	说明
sharding_table_reference	表关联关系

12.6.15 加解密

加解密

语法	描述	类型
CREATE ENCRYPT RULE encryptRuleDefinition]	encryptRuleDefinition [, ...	RDL
ALTER ENCRYPT RULE encryptRuleDefinition]	encryptRuleDefinition [, ...	RDL
DROP ENCRYPT RULE	tableName [, tableName] ...	RDL
SHOW ENCRYPT RULES	[FROM databaseName]	RQL
SHOW ENCRYPT TABLE RULE	tableName [FROM databaseName]	RQL
COUNT ENCRYPT RULE	统计加密规则的数量，在后续版本中将废弃此语法	RQL
CREATE ENCRYPT KEY MANAGER keyManagerDefinition	MANAGER keyManagerName	RDL
ALTER ENCRYPT KEY MANAGER keyManagerDefinition	MANAGER keyManagerName	RDL
DROP ENCRYPT KEY	MANAGER ifExists? keyManagerName	RDL
select [,string] ... [,string] ...	[,] decrypt({tableName}, {encryptColumn}, {encrypt-Value})	RQL

1. 创建加密规则

```

CreateEncryptRule ::=
'CREATE' 'ENCRYPT' 'RULE' ifNotExists? encryptDefinition (',' encryptDefinition)*

ifNotExists ::=
'IF' 'NOT' 'EXISTS'

encryptDefinition ::=
ruleName '(' 'COLUMNS' '(' columnDefinition (',' columnDefinition)* ')' (',' 'QUERY_WITH_CIPHER_COLUMN' '=' ('TRUE' | 'FALSE'))? ')'

columnDefinition ::=
'(' 'NAME' '=' columnName (',' 'DATA_TYPE' '=' dataType)? (',' 'PLAIN' '=' plainColumnName (',' 'PLAIN_DATA_TYPE' '=' dataType)? )? (',' 'CIPHER' '=' cipherColumnName (',' 'CIPHER_DATA_TYPE' '=' dataType)? (',' 'ASSISTED_QUERY_COLUMN' '=' assistedQueryColumnName (',' 'ASSISTED_QUERY_DATA_TYPE' '=' dataType)? )? (',' 'LIKE_QUERY_COLUMN' '=' likeQueryColumnName (',' 'LIKE_QUERY_DATA_TYPE' '=' dataType)? )? ',' encryptAlgorithmDefinition (',' assistedQueryAlgorithmDefinition)? (',' likeQueryAlgorithmDefinition)? ')'

encryptAlgorithmDefinition ::=
'ENCRYPT_ALGORITHM' '(' 'TYPE' '(' 'NAME' '=' encryptAlgorithmType (',' propertiesDefinition)? ')'

assistedQueryAlgorithmDefinition ::=
'ASSISTED_QUERY_ALGORITHM' '(' 'TYPE' '(' 'NAME' '=' encryptAlgorithmType (',' propertiesDefinition)? ')'

likeQueryAlgorithmDefinition ::=
'LIKE_QUERY_ALGORITHM' '(' 'TYPE' '(' 'NAME' '=' encryptAlgorithmType (',' propertiesDefinition)? ')'

propertiesDefinition ::=
'PROPERTIES' '(' 'key' '=' value (',' 'key' '=' value)* ')'

tableName ::=
identifier

columnName ::=
identifier

dataType ::=
string

```

```
plainColumnName ::=
  identifier

cipherColumnName ::=
  identifier

assistedQueryColumnName ::=
  identifier

likeQueryColumnName ::=
  identifier

encryptAlgorithmType ::=
  string

key ::=
  string

value ::=
  literal
```

参数说明

参数解释

名称	数据类型	说明
tableName	IDENTIFIER	表名称
columnName	IDENTIFIER	逻辑列名称
DATA_TYPE	IDENTIFIER	逻辑列数据类型
PLAIN	IDENTIFIER	明文列名称
PLAIN_DATA_TYPE	IDENTIFIER	明文列数据类型
CIPHER	IDENTIFIER	加密列名称
CIPHER_DATA_TYPE	IDENTIFIER	加密列数据类型
ASSISTED_QUERY_COLUMN	IDENTIFIER	辅助查询列
ASSISTED_QUERY_DATA_TYPE	IDENTIFIER	辅助查询列数据类型
LIKE_QUERY_COLUMN	IDENTIFIER	模糊查询列
LIKE_QUERY_DATA_TYPE	IDENTIFIER	模糊查询列数据类型
ENCRYPT_ALGORITHM	STRING	加密算法名称
ASSISTED_QUERY_ALGORITHM	STRING	辅助查询算法
LIKE_QUERY_ALGORITHM	STRING	模糊查询算法

- PLAIN 指定明文数据列，CIPHER 指定密文数据列；
- encryptAlgorithmType 指定加密算法类型，请参考 [加密算法](#)；
- 重复的 tableName 将无法被创建；
- QUERY_WITH_CIPHER_COLUMN 支持大写或小写的 true 或 false, 支持表级别和列级别的设置；
- ASSISTED_QUERY_COLUMN 辅助查询列：在用户的 CIPHER (加密字段) 以及对应的加密算法无法支持查询时，可以配置该字段用来辅助查询。例如用户配置的加密算法对于同一个值进行多次加密的结果不同，那么就无法使用 CIPHER (加密字段) 进行查询，这时候就需要使用辅助查询列进行查询。辅助查询列对应的算法一般可以使用不可逆的算法，但是对于同一个值进行多次加密的结果需要相同。
- LIKE_QUERY_COLUMN 模糊查询列：用于帮助用户进行 like 查询时使用的列。一般 CIPHER (加密字段) 配置的算法无法支持 like 查询，所以当用户需要使用 like 查询时，需要配置模糊查询列以及对应的模糊查询算法。
- ASSISTED_QUERY_ALGORITHM 辅助查询算法：即辅助查询列对应的算法。一般是不可逆，并且多次加密结果一致的算法。
- LIKE_QUERY_ALGORITHM 模糊查询算法：即模糊查询列对应的算法，可以支持 like 查询的算法。

示例

创建数据加密规则

```
CREATE ENCRYPT RULE t_encrypt (
  COLUMNS(
    (NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'=
'123456abc')))),
    (NAME=order_id, CIPHER=order_cipher,ENCRYPT_ALGORITHM(TYPE(NAME='MD5'))))
  ),QUERY_WITH_CIPHER_COLUMN=true),
  t_encrypt_2 (
    COLUMNS(
      (NAME=user_id,DATA_TYPE='int(11)',PLAIN=user_plain,PLAIN_DATA_TYPE='int(11)',CIPHER=user_cipher,CIPHER_DATA_TYPE=
'varchar(255)',ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'= '123456abc')))),
      (NAME=order_id, CIPHER=order_cipher,ENCRYPT_ALGORITHM(TYPE(NAME='MD5'))))
    ), QUERY_WITH_CIPHER_COLUMN=FALSE);
```

使用 ifNotExists 子句创建数据加密规则

```
CREATE ENCRYPT RULE IF NOT EXISTS t_encrypt (
  COLUMNS(
    (NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'=
'123456abc')))),
    (NAME=order_id, CIPHER=order_cipher,ENCRYPT_ALGORITHM(TYPE(NAME='MD5'))))
  ),QUERY_WITH_CIPHER_COLUMN=true),
  t_encrypt_2 (
    COLUMNS(
      (NAME=user_id,DATA_TYPE='int(11)',PLAIN=user_plain,PLAIN_DATA_TYPE='int(11)',CIPHER=user_cipher,CIPHER_DATA_TYPE=
'varchar(255)',ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'= '123456abc')))),
      (NAME=order_id, CIPHER=order_cipher,ENCRYPT_ALGORITHM(TYPE(NAME='MD5'))))
    ), QUERY_WITH_CIPHER_COLUMN=FALSE);
```

2. 修改加密规则

```
ALTEREncryptRule ::=
  'ALTER' 'ENCRYPT' 'RULE' encryptDefinition (',' encryptDefinition)*

encryptDefinition ::=
  ruleName '(' 'COLUMNS' '(' columnDefinition (',' columnDefinition)* ')' (',' QUERY_WITH_CIPHER_COLUMN '=' ('TRUE' | 'FALSE'
'))? ')'

columnDefinition ::=
  '(' 'NAME' '=' columnName (',' 'DATA_TYPE' '=' dataType)? (',' 'PLAIN' '=' plainColumnName (',' 'PLAIN_DATA_TYPE' '=' dataType)?
)? (',' 'CIPHER' '=' cipherColumnName (',' 'CIPHER_DATA_TYPE' '=' dataType)? (',' 'ASSISTED_QUERY_COLUMN' '='
assistedQueryColumnName (',' 'ASSISTED_QUERY_DATA_TYPE' '=' dataType)?)? (',' 'LIKE_QUERY_COLUMN' '='
likeQueryColumnName (',' 'LIKE_QUERY_DATA_TYPE' '=' dataType)?)? ',' encryptAlgorithmDefinition (','
assistedQueryAlgorithmDefinition)? (',' likeQueryAlgorithmDefinition)? ')'

encryptAlgorithmDefinition ::=
  'ENCRYPT_ALGORITHM' '(' 'TYPE' '(' 'NAME' '=' encryptAlgorithmType (',' propertiesDefinition)? ')'

assistedQueryAlgorithmDefinition ::=
  'ASSISTED_QUERY_ALGORITHM' '(' 'TYPE' '(' 'NAME' '=' encryptAlgorithmType (',' propertiesDefinition)? ')'

likeQueryAlgorithmDefinition ::=
  'LIKE_QUERY_ALGORITHM' '(' 'TYPE' '(' 'NAME' '=' encryptAlgorithmType (',' propertiesDefinition)? ')'

propertiesDefinition ::=
  'PROPERTIES' '(' 'key' '=' value (',' 'key' '=' value)* ')'

tableName ::=
  identifier
```

```
columnName ::=
  identifier

dataType ::=
  string

plainColumnName ::=
  identifier

cipherColumnName ::=
  identifier

assistedQueryColumnName ::=
  identifier

likeQueryColumnName ::=
  identifier

encryptAlgorithmType ::=
  string

key ::=
  string

value ::=
  literal
```

参数说明

名称	数据类型	说明
tableName	IDENTIFIER	表名称
columnName	IDENTIFIER	逻辑列名称
DATA_TYPE	IDENTIFIER	逻辑列数据类型
PLAIN	IDENTIFIER	明文列名称
PLAIN_DATA_TYPE	IDENTIFIER	明文列数据类型
CIPHER	IDENTIFIER	加密列名称
CIPHER_DATA_TYPE	IDENTIFIER	加密列数据类型
ASSISTED_QUERY_COLUMN	IDENTIFIER	辅助查询列
ASSISTED_QUERY_DATA_TYPE	IDENTIFIER	辅助查询列数据类型
LIKE_QUERY_COLUMN	IDENTIFIER	模糊查询列
LIKE_QUERY_DATA_TYPE	IDENTIFIER	模糊查询列数据类型
ENCRYPT_ALGORITHM	STRING	加密算法名称
ASSISTED_QUERY_ALGORITHM	STRING	辅助查询算法
LIKE_QUERY_ALGORITHM	STRING	模糊查询算法

- PLAIN 指定明文数据列，CIPHER 指定密文数据列；
- encryptAlgorithmType 指定加密算法类型，请参考 [加密算法](#)；
- 重复的 tableName 将无法被创建；
- QUERY_WITH_CIPHER_COLUMN 支持大写或小写的 true 或 false, 支持表级别和列级别的设置；
- ASSISTED_QUERY_COLUMN 辅助查询列：在用户的 CIPHER (加密字段) 以及对应的加密算法无法支持查询时，可以配置该字段用来辅助查询。例如用户配置的加密算法对于同一个值进行多次加密的结果不同，那么就无法使用 CIPHER (加密字段) 进行查询，这时候就需要使用辅助查询列进行查询。辅助查询列对应的算法一般可以使用不可逆的算法，但是对于同一个值进行多次加密的结果需要相同。
- LIKE_QUERY_COLUMN 模糊查询列：用于帮助用户进行 like 查询时使用的列。一般 CIPHER (加密字段) 配置的算法无法支持 like 查询，所以当用户需要使用 like 查询时，需要配置模糊查询列以及对应的模糊查询算法。
- ASSISTED_QUERY_ALGORITHM 辅助查询算法：即辅助查询列对应的算法。一般是不可逆，并且多次加密结果一致的算法。

- LIKE_QUERY_ALGORITHM 模糊查询算法：即模糊查询列对应的算法，可以支持 like 查询的算法。

示例

```
ALTER ENCRYPT RULE t_encrypt (
  COLUMNS(
    (NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'=
'123456abc')))),
    (NAME=order_id,CIPHER=order_cipher,CIPHER_DATA_TYPE='varchar(255)',ENCRYPT_ALGORITHM(TYPE(NAME='MD5'))))
  ), QUERY_WITH_CIPHER_COLUMN=TRUE);
```

3. 删除加密规则

示例

删除加密规则

```
DROP ENCRYPT RULE t_encrypt, t_encrypt_2;
```

使用 ifExists 删除加密规则

```
DROP ENCRYPT RULE IF EXISTS t_encrypt, t_encrypt_2;
```

4. 查看加密规则

查询所有的数据加密规则

```
SHOW ENCRYPT RULES [FROM databaseName]
```

示例

```
mysql> SHOW ENCRYPT RULES FROM encrypt_db;
```

table	logic_column	logic_data_type	cipher_column	cipher_data_type	plain_column	plain_data_type	assisted_query_column	assisted_query_data_type	encryptor_type	encryptor_props	query_with_cipher_column
t_encrypt	user_id		user_cipher		user_plain				AES	aes-key-value=123456abc	true
t_encrypt	order_id		order_cipher						MD5		true
t_encrypt_2	user_id		user_cipher		user_plain				AES	aes-key-value=123456abc	false
t_encrypt_2	order_id		order_cipher						MD5		false

4 rows in set (0.78 sec)

输出说明

列	说明
table	逻辑表名
logic_column	逻辑列名
logic_data_type	逻辑列数据类型
cipher_column	密文列名
cipher_data_type	密文列数据类型
plain_column	明文列名
plain_data_type	明文列数据类型
assisted_query_column	辅助查询列名
assisted_query_data_type	辅助查询列数据类型
encryptor_type	加密算法类型
encryptor_props	加密算法参数
query_with_cipher_column	是否使用加密列进行查询

5. 查询所有的数据加密规则

```
SHOW ENCRYPT RULES [FROM databaseName]
```

示例

```
mysql> SHOW ENCRYPT RULES FROM encrypt_db;
+-----+-----+-----+-----+-----+-----+-----+-----+
| table | logic_column | logic_data_type | cipher_column | cipher_data_type | plain_column | plain_data_type | assisted_query_column | assisted_query_data_type | encryptor_type | encryptor_props | query_with_cipher_column |
+-----+-----+-----+-----+-----+-----+-----+-----+
| t_encrypt | user_id | | user_cipher | | user_plain | | AES | aes-key- |
value=123456abc | true |
| t_encrypt | order_id | | order_cipher | | | | MD5 | | true |
| t_encrypt_2 | user_id | | user_cipher | | user_plain | | AES | aes-key- |
value=123456abc | false |
| t_encrypt_2 | order_id | | order_cipher | | | | MD5 | | |
false |
4 rows in set (0.78 sec)
```

输出说明

列	说明
table	逻辑表名
logic_column	逻辑列名
logic_data_type	逻辑列数据类型
cipher_column	密文列名
cipher_data_type	密文列数据类型
plain_column	明文列名
plain_data_type	明文列数据类型
assisted_query_column	辅助查询列名
assisted_query_data_type	辅助查询列数据类型
encryptor_type	加密算法类型
encryptor_props	加密算法参数
query_with_cipher_column	是否使用加密列进行查询

6. 查询指定加密规则

示例

```
mysql> SHOW ENCRYPT TABLE RULE t_encrypt;
+-----+-----+-----+-----+-----+-----+-----+-----+
| table | logic_column | logic_data_type | cipher_column | cipher_data_type | plain_column | plain_data_type | assisted_query_column | assisted_query_data_type | encryptor_type | encryptor_props | query_with_cipher_column |
+-----+-----+-----+-----+-----+-----+-----+-----+
| t_encrypt | user_id | | user_cipher | | user_plain | | | AES | aes-key- |
value=123456abc | true |
| t_encrypt | order_id | | order_cipher | | | | MD5 | | true |
+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)

mysql> SHOW ENCRYPT TABLE RULE t_encrypt FROM encrypt_db;
+-----+-----+-----+-----+-----+-----+-----+-----+
| table | logic_column | logic_data_type | cipher_column | cipher_data_type | plain_column | plain_data_type | assisted_query_column | assisted_query_data_type | encryptor_type | encryptor_props | query_with_cipher_column |
+-----+-----+-----+-----+-----+-----+-----+-----+
| t_encrypt | user_id | | user_cipher | | user_plain | | | AES | aes-key- |
value=123456abc | true |
| t_encrypt | order_id | | order_cipher | | | | MD5 | | true |
+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

输出说明

列	说明
table	逻辑表名
logic_column	逻辑列名
logic_data_type	逻辑列数据类型
cipher_column	密文列名
cipher_data_type	密文列数据类型
plain_column	明文列名
plain_data_type	明文列数据类型
assisted_query_column	辅助查询列名
assisted_query_data_type	辅助查询列数据类型
encryptor_type	加密算法类型
encryptor_props	加密算法参数
query_with_cipher_column	是否使用加密列进行查询

7. 创建密钥管理规则

```
createEncryptKeyManager
: CREATE ENCRYPT KEY MANAGER keyManagerName keyManagerDefinition
;

keyManagerDefinition
: TYPE (NAME = keyManagerName [, PROPERTIES ([keyManagerProperties]) ])
;

keyManagerProperties:
keyManagerProperty [, keyManagerProperty] ...

keyManagerProperty:
key=value
```

参数说明

名称	数据类型	说明
keyManagerName	STRING	密钥管理器名称

示例

本地密钥管理

```
CREATE ENCRYPT KEY MANAGER local_key_manage (
TYPE(NAME='LOCAL',PROPERTIES("aes-key-value"="123456abc"))
);
```

云端密钥管理

```
CREATE ENCRYPT KEY MANAGER aws_key_manage (
TYPE(NAME='SphereEx:AWS_KMS',PROPERTIES(
"access-key"="aaaaa",
"secret-key"="bbbbb",
"aws-region"="us-east-1",
"secret-name"="testA"))
);
```

8. 修改密钥管理规则

```
alterEncryptKeyManager
: ALTER ENCRYPT KEY MANAGER keyManagerName keyManagerDefinition
;

keyManagerDefinition
: TYPE (NAME = keyManagerName [, PROPERTIES ([keyManagerProperties]) ])
;

keyManagerProperties:
keyManagerProperty [, keyManagerProperty] ...

keyManagerProperty:
key=value
```

参数说明

名称	数据类型	说明
keyManagerName	STRING	密钥管理器名称

示例

本地密钥管理

```
ALTER ENCRYPT KEY MANAGER local_key_manage (  
TYPE(NAME='LOCAL',PROPERTIES("aes-key-value"="123456abcd"))  
);
```

云端密钥管理

```
ALTER ENCRYPT KEY MANAGER aws_key_manage (  
TYPE(NAME='SphereEx:AWS_KMS',PROPERTIES(  
  "access-key"="aaaaa",  
  "secret-key"="bbbbb",  
  "aws-region"="us-east-1",  
  "secret-name"="testB"))  
);
```

9. 删除密钥管理规则

```
DROP ENCRYPT KEY MANAGER ifExists? keyManagerName;
```

参数说明

名称	数据类型	说明
keyManagerName	STRING	密钥管理器名称

示例

```
DROP ENCRYPT KEY MANAGER ifExists Aws_key_Manager;
```

10. 查询解密结果

select 用法一致，提供 decrypt() 函数，tableName 为需要解密的表，encryptColumn 为需要解密的列，encryptValue 为需要解密的真实值。使用时需要配置对应的加密规则。

```
select [,string] ... [,] decrypt({tableName},{encryptColumn},{encryptValue}), [,string] ...
```

示例

```
SELECT 'Lm04PRW+sp+mG/QuimAUew==' ,decrypt(t_user,pwd_cipher, 'Lm04PRW+sp+mG/QuimAUew==' ) UNION SELECT  
'XXX' ,decrypt(t_user,pwd_cipher, 'XXX' );
```

12.6.16 加解密洗数任务

加解密洗数任务

语法	描述	类型
ALTER ENCRYPTING RULE	修改数据加密洗数规则	RAL
SHOW ENCRYPTING RULE	查看数据加密洗数规则	RAL
ENCRYPT TABLE	启动加密洗数作业	RAL
SHOW ENCRYPTING LIST	查看加密洗数作业	RAL
SHOW ENCRYPTING STATUS	查看加密洗数状态	RAL
COMMIT ENCRYPTING	提交加密洗数作业	RAL
ROLLBACK ENCRYPTING	回滚加密洗数作业	RAL
ALTER DECRYPTING RULE	修改数据解密洗数规则	RAL
SHOW DECRYPTING RULE	查看数据解密洗数规则	RAL
DECRYPT TABLE	启动解密洗数作业	RAL
SHOW DECRYPTING LIST	查看解密洗数作业	RAL
SHOW DECRYPTING STATUS	查看解密洗数状态	RAL
COMMIT DECRYPTING	提交解密洗数作业	RAL
ROLLBACK DECRYPTING	回滚解密洗数作业	RAL
REENCRYPT TABLE WITH RULE tableName	换密钥第一阶段语句	RAL
START REENCRYPTING jobId	换密钥第二阶段洗数语句	RAL

1. 定义数据加密洗数规则

ALTER ENCRYPTING RULE

示例

```
ALTER ENCRYPTING RULE (READ(RATE_LIMITER ( TYPE(NAME= 'QPS' ,PROPERTIES('qps= '5000'))));
```

2. 查看数据加密洗数规则

SHOW ENCRYPTING RULE

参数说明

无

示例

SHOW ENCRYPTING RULE

3. 启动加密洗数作业

```
ENCRYPT TABLE tableName;
```

参数说明

参数名	说明
tableName	加密洗数的表

示例


```
ENCRYPT TABLE t_user;
```

4. 查看加密洗数作业

```
SHOW ENCRYPTING LIST;
```

参数说明
无
示例

```
SHOW ENCRYPTING LIST;
```

5. 查看加密洗数状态

```
SHOW ENCRYPTING STATUS jobId;
```

参数说明

参数名	说明
jobid	作业 id

示例

```
SHOW ENCRYPTING STATUS 'j51017f973ac82cb1edea4f5238a258c25e89';
```

6. 提交加密洗数作业

```
COMMIT ENCRYPTING jobId;
```

参数说明

参数名	说明
jobid	作业 id

示例

```
COMMIT ENCRYPTING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

7. 定义数据解密洗数规则

```
ALTER DECRYPTING RULE
```

示例

```
ALTER DECRYPTING RULE (READ(RATE_LIMITER ( TYPE(NAME= 'QPS' ,PROPERTIES('qps= '5000'))));
```

8. 查看数据解密洗数规则

```
SHOW DECRYPTING RULE
```

参数说明

无

示例

```
SHOW DECRYPTING RULE
```

9. 启动解密洗数作业

```
DECRYPT TABLE tableName;
```

参数说明

参数名	说明
tableName	加密洗数的表

示例

```
DECRYPT TABLE t_user;
```

10. 查看解密洗数作业

```
SHOW DECRYPTING LIST;
```

参数说明

无

示例

```
SHOW DECRYPTING LIST;
```

11. 查看解密洗数状态

```
SHOW DECRYPTING STATUS jobId;
```

参数说明

参数名	说明
jobid	作业 id

示例

```
SHOW DECRYPTING STATUS 'j51017f973ac82cb1edea4f5238a258c25e89';
```

12. 提交解密洗数作业

```
COMMIT DECRYPTING jobId;
```

参数说明

参数名	说明
jobid	作业 id

示例

```
COMMIT DECRYPTING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

13. 换密钥第一阶段语句

```
REENCRYPT TABLE WITH RULE tableName
```

参数说明

参数名	说明
tableName	换密钥的表

示例

```
REENCRYPT TABLE WITH RULE t_encrypt (  
  COLUMNS(  
    (NAME=user_id,PLAIN=user_plain,CIPHER=user_cipher,ASSISTED_QUERY_COLUMN=user_assisted,LIKE_QUERY_COLUMN=user_like,ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'='665544aaa'))),ASSISTED_QUERY_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'='523234abc'))), LIKE_QUERY_ALGORITHM(TYPE(NAME='CHAR_DIGEST_LIKE'))),  
    (NAME=order_id, PLAIN=order_plain,CIPHER =order_cipher, ENCRYPT_ALGORITHM(TYPE(NAME='AES',PROPERTIES('aes-key-value'='445533ccc'))))  
  ), QUERY_WITH_CIPHER_COLUMN=true);
```

14. 换密钥第二阶段洗数语句

```
START REENCRYPTING jobId;
```

参数说明

参数名	说明
jobid	换密钥作业的 id

示例

```
START REENCRYPTING j54014fb26631db0f7149741422d86cc5dfe6;
```

12.6.17 数据库发现

数据库发现

语法	描述	类型
CREATE DB_DISCOVERY RULE ruleDefinition [, ruleDefinition] ...	创建数据库发现规则	RDL
ALTER DB_DISCOVERY RULE ruleDefinition [, ruleDefinition] ...	修改数据库发现规则	RDL
DROP DB_DISCOVERY RULE ruleName [, ruleName] ...	删除数据库发现规则	RDL
DROP DB_DISCOVERY TYPE discoveryTypeName [, discoveryType-Name] ...	删除数据库发现类型	RDL
DROP DB_DISCOVERY HEARTBEAT d iscoveryHeartbeatName [, di scoveryHeartbeatName] ...	删除数据库发现监听心跳算法	RDL
SHOW DB_DISCOVERY RULES [FROM databaseName]	查看数据库发现规则	RQL
SHOW DB_DISCOVERY TYPES [FROM databaseName]	查看数据库发现类型	RQL
SHOW DB_DISCOVERY HEARTBEATS [FROM databaseName]	查看数据库发现监听心跳算法	RQL
COUNT DB_DISCOVERY RULE	统计数据库发现规则的数量，在后续版本中将废弃此语法	RQL

1. 创建数据库发现规则

CREATE DB_DISCOVERY RULE ruleDefinition [, ruleDefinition] ...

ruleDefinition:
ruleName (storageUnits, typeDefinition, heartbeatDefinition)

storageUnits:
STORAGE_UNITS(storageUnitName [, storageUnitName] ...)

typeDefinition:
TYPE(NAME=typeName [, PROPERTIES([properties]))

heartbeatDefinition
HEARTBEAT (PROPERTIES (properties))

properties:
property [, property] ...

property:
key=value

参数说明

名称	数据类型	说明
discoveryTypeName	IDENTIFIER	数据库发现类型名
ruleName	IDENTIFIER	规则名称
discoveryHeartbeatName	IDENTIFIER	监听心跳名称
typeName	STRING	数据库发现类型，如：MySQL.MGR
storageUnitName	IDENTIFIER	资源名称

- discoveryType 指定数据库发现服务类型，DBPlusEngine 内置支持 MySQL.NORMAL_REPLICATION、MySQL.MGR、openGauss.NORMAL_REPLICATION、SphereEx:GaussDB_for_MySQL.NORMAL_REPLICATION、SphereEx:PostgreSQL.NORMAL_REPLICATION 具体配置请参考[数据库发现内置算法](#)；
- 重复的 ruleName 将无法被创建；
- 正在被使用的 discoveryType 和 discoveryHeartbeat 无法被删除；
- 带有 - 的命名在改动时需要使用 " " ；

- 移除 discoveryRule 时不会移除被该 discoveryRule 使用的 discoveryType 和 discoveryHeartbeat 。

示例

创建 discoveryRule 时同时创建 discoveryType 和 discoveryHeartbeat

```
CREATE DB_DISCOVERY RULE db_discovery_group_0 (  
  STORAGE_UNITS(ds_0, ds_1, ds_2),  
  TYPE(NAME='MySQL.MGR', PROPERTIES('group-name'='92504d5b-6dec')),  
  HEARTBEAT(PROPERTIES('keep-alive-cron'='0/5 * * * * ?'))  
);
```

2. 修改数据库发现规则

```
ALTER DB_DISCOVERY RULE ruleDefinition [, ruleDefinition] ...
```

ruleDefinition:

ruleName (storageUnits, typeDefinition, heartbeatDefinition)

storageUnits:

STORAGE_UNITS(storageUnitName [, storageUnitName] ...)

typeDefinition:

TYPE(NAME=typeName [, PROPERTIES([properties])])

heartbeatDefinition

HEARTBEAT (PROPERTIES (properties))

properties:

property [, property] ...

property:

key=value

参数说明

名称	数据类型	说明
discoveryTypeName	IDENTIFIER	数据库发现类型名
ruleName	IDENTIFIER	规则名称
discoveryHeartbeatName	IDENTIFIER	监听心跳名称
typeName	STRING	数据库发现类型，如：MySQL.MGR
storageUnitName	IDENTIFIER	资源名称

- discoveryType 指定数据库发现服务类型，ShardingSphere 内置支持 MySQL.MGR ；
- 重复的 ruleName 将无法被创建；
- 带有 - 的命名在改动时需要使用 " " ；

示例

修改 discoveryRule 可以同时修改 discoveryType 和 discoveryHeartbeat

```
ALTER DB_DISCOVERY RULE db_discovery_group_0 (  
  STORAGE_UNITS(ds_0, ds_1, ds_2),  
  TYPE(NAME='MySQL.MGR', PROPERTIES('group-name'='246e9612-aaf1')),  
  HEARTBEAT(PROPERTIES('keep-alive-cron'='0/5 * * * * ?'))  
);
```

3. 删除数据库发现规则

DROP DB_DISCOVERY **RULE** ruleName [, ruleName] ...

参数说明

名称	数据类型	说明
ruleName	IDENTIFIER	规则名称

- 重复的 ruleName 将无法被创建；
- 移除 discoveryRule 时不会移除被该 discoveryRule 使用的 discoveryType 和 discoveryHeartbeat 。

示例

DROP DB_DISCOVERY **RULE** db_discovery_group_0;

4. 删除数据库发现类型

DROP DB_DISCOVERY **TYPE** discoveryTypeName [, discoveryTypeName] ...

参数说明

名称	数据类型	说明
discoveryTypeName	IDENTIFIER	数据库发现类型名
typeName	STRING	数据库发现类型，如：MySQL.MGR

- 移除 discoveryRule 时不会移除被该 discoveryRule 使用的 discoveryType 和 discoveryHeartbeat 。

示例

DROP DB_DISCOVERY **TYPE** db_discovery_group_0_mgr;

5. 删除数据库发现监听心跳

DROP DB_DISCOVERY HEARTBEAT discoveryHeartbeatName [, discoveryHeartbeatName] ...

参数说明

名称	数据类型	说明
discoveryHeartbeatName	IDENTIFIER	监听心跳名称

示例

DROP DB_DISCOVERY HEARTBEAT db_discovery_group_0_heartbeat;

6. 查看数据库发现规则

SHOW DB_DISCOVERY RULES [**FROM** databaseName]

参数说明

名称	数据类型	说明
databaseName	IDENTIFIER	逻辑库名称

示例

查看当前逻辑库下的数据库发现规则

```
mysql> SHOW DB_DISCOVERY RULES;
+-----+-----+-----+-----+
| group_name | data_source_names | primary_data_source_name | discovery_type |
| discovery_heartbeat | | | |
+-----+-----+-----+-----+
| db_discovery_group_0 | ds_0,ds_1,ds_2 | ds_0 | {name=db_discovery_group_0_mgr,type=MySQL.MGR,props={group-name=92504d5b-6dec}} | {name=db_discovery_group_0_heartbeat,props={keep-alive-cron=0/5 * * * * ?}} |
+-----+-----+-----+-----+
1 row in set (0.20 sec)
```

输出说明

列	说明
group_name	规则名称
data_source_names	数据源名称列表
primary_data_source_name	主数据源名称
discovery_type	数据库发现服务类型
discovery_heartbeat	数据库发现服务心跳

7. 查看数据库发现类型

SHOW DB_DISCOVERY TYPES [**FROM** databaseName]

参数说明

名称	数据类型	说明
databaseName	IDENTIFIER	逻辑库名称

示例

查看当前逻辑库下的数据库发现类型

```
mysql> SHOW DB_DISCOVERY TYPES;
+-----+-----+-----+
| name | type | props |
+-----+-----+-----+
| db_discovery_group_0_mgr | MySQL.MGR | {group-name=92504d5b-6dec} |
+-----+-----+-----+
1 row in set (0.01 sec)
```

输出说明

列	说明
name	类型名称
type	类型种类
props	类型参数

8. 查看数据库发现监听心跳算法

```
SHOW DB_DISCOVERY HEARTBEATS [FROM databaseName]
```

参数说明

名称	数据类型	说明
databaseName	IDENTIFIER	逻辑库名称

示例

查看当前逻辑库下的数据库心跳

```
mysql> SHOW DB_DISCOVERY HEARTBEATS;
+-----+-----+
| name          | props          |
+-----+-----+
| db_discovery_group_0_heartbeat | {keep-alive-cron=0/5 * * * * ?} |
+-----+-----+
1 row in set (0.01 sec)
```

输出说明

列	说明
name	心跳名称
props	心跳参数

12.6.18 读写分离

读写分离

语法	描述	类型
CREATE READWRITE_SPLITTING RULE readwriteSp littingRuleDefinition [, readwriteSpl ittingRuleDefinition] ...	创建读写分离规则	RDL
ALTER READWRITE_SPLITTING RULE readwriteSp littingRuleDefinition [, readwriteSpl ittingRuleDefinition] ...	修改读写分离规则	RDL
DROP READWRITE_SPLITTING RULE	删除读写分离规则	RDL
ALTER READWRITE_SPLITTING RULE [groupName] (ENABLE / DISABLE) storageUnitName [FROM databaseName]	启用 / 禁用读库	RDL
SHOW READWRITE_SPLITTING RULES [FROM databaseName]	查看读写分离规则	RQL
SHOW STATUS FROM READWRITE_SPLITTING (RULES / RULE groupName) [FROM databaseName]	查询读写分离数据源状态	RQL
COUNT READWRITE_SPLITTING RULE	统计读写分离规则的数量，在后续版本中将废弃此语法	RQL

1. 创建读写分离规则

```
CREATE READWRITE_SPLITTING RULE readwriteSplittingRuleDefinition [, readwriteSplittingRuleDefinition] ...

readwriteSplittingRuleDefinition:
    ruleName ([staticReadwriteSplittingRuleDefinition | dynamicReadwriteSplittingRuleDefinition]
        [, loadBalancerDefinition])

staticReadwriteSplittingRuleDefinition:
    WRITE_STORAGE_UNIT=storageUnitName, READ_STORAGE_UNITS(storageUnitName [, storageUnitName] ... )

dynamicReadwriteSplittingRuleDefinition:
    AUTO_AWARE_RESOURCE=autoAwareResourceName [, WRITE_DATA_SOURCE_QUERY_
ENABLED=writeDataSourceQueryEnabled]

loadBalancerDefinition:
    TYPE(NAME=loadBalancerType [, PROPERTIES([algorithmProperties] )) )

algorithmProperties:
    algorithmProperty [, algorithmProperty] ...

algorithmProperty:
    key=value

writeDataSourceQueryEnabled:
    TRUE | FALSE
```

参数说明

名称	数据类型	说明
ruleName	IDENTIFIER	规则名称
storageUnitName	IDENTIFIER	已注册的数据源名称
autoAwareResourceName	IDENTIFIER	数据库发现的逻辑数据源名称
writeDataSourceQueryEnabled	BOOLEAN	读库全部下线，主库是否承担读流量
loadBalancerType	STRING	负载均衡算法类型

- 支持创建静态读写分离规则和动态读写分离规则；
- 动态读写分离规则依赖于数据库发现规则；
- loadBalancerType 指定负载均衡算法类型，请参考 [读写分离算法](#)；
- 重复的 ruleName 将无法被创建。

示例

创建静态的读写分离 rule

```
// Static
CREATE READWRITE_SPLITTING RULE ms_group_0 (
WRITE_STORAGE_UNIT=write_ds,
READ_STORAGE_UNITS(read_ds_0,read_ds_1),
TYPE(NAME="random")
);
```

创建动态的读写分离 rule

```
// Dynamic
CREATE READWRITE_SPLITTING RULE ms_group_1 (
AUTO_AWARE_RESOURCE=group_0,
WRITE_DATA_SOURCE_QUERY_ENABLED=false,
TYPE(NAME="random")
);
```

2. 修改读写分离规则

```
ALTER READWRITE_SPLITTING RULE readwriteSplittingRuleDefinition [, readwriteSplittingRuleDefinition] ...
readwriteSplittingRuleDefinition:
    ruleName ([staticReadwriteSplittingRuleDefinition | dynamicReadwriteSplittingRuleDefinition]
        [, loadBalancerDefinition])

staticReadwriteSplittingRuleDefinition:
    WRITE_STORAGE_UNIT=storageUnitName, READ_STORAGE_UNITS(storageUnitName [, storageUnitName] ... )

dynamicReadwriteSplittingRuleDefinition:
    AUTO_AWARE_RESOURCE=autoAwareResourceName [, WRITE_DATA_SOURCE_QUERY_
ENABLED=writeDataSourceQueryEnabled]

loadBalancerDefinition:
    TYPE(NAME=loadBalancerType [, PROPERTIES([algorithmProperties] )) )

algorithmProperties:
    algorithmProperty [, algorithmProperty] ...

algorithmProperty:
    key=value

writeDataSourceQueryEnabled:
    TRUE | FALSE
```

参数说明

名称	数据类型	说明
ruleName	IDENTIFIER	规则名称
storageUnitName	IDENTIFIER	已注册的数据源名称
autoAwareResourceName	IDENTIFIER	数据库发现的逻辑数据源名称
writeDataSourceQueryEnabled	BOOLEAN	读库全部下线，主库是否承担读流量
loadBalancerType	STRING	负载均衡算法类型

- 支持创建静态读写分离规则和动态读写分离规则；
- 动态读写分离规则依赖于数据库发现规则；
- loadBalancerType 指定负载均衡算法类型，请参考 [读写分离算法](#)；
- 重复的 ruleName 将无法被创建

示例

```
ALTER READWRITE_SPLITTING RULE ms_group_1 (
WRITE_STORAGE_UNIT=write_ds,
READ_STORAGE_UNITS(read_ds_0,read_ds_1,read_ds_2),
TYPE(NAME="random",PROPERTIES(write_ds='2',read_ds_0='2',read_ds_1='2',read_ds_2='1'))
);
```

3. 删除读写分离规则

DROP READWRITE_SPLITTING **RULE** ruleName [, ruleName] ...

参数说明

名称	数据类型	说明
ruleName	IDENTIFIER	规则名称

示例

DROP READWRITE_SPLITTING **RULE** ms_group_1;

4. 启用/禁用读写分离规则

ALTER READWRITE_SPLITTING **RULE** [groupName] (ENABLE / DISABLE) storageUnitName [**FROM** databaseName]

示例

ALTER READWRITE_SPLITTING **RULE** group_1 ENABLE read_ds_1

5. 查看读写分离规则

SHOW READWRITE_SPLITTING RULES [**FROM** databaseName]

示例

静态读写分离规则

```
mysql> SHOW READWRITE_SPLITTING RULES;
+-----+-----+-----+-----+-----+-----+
| name      | auto_aware_data_source_name | write_data_source_name | read_data_source_names | load_balancer_type | load_balancer_props |
+-----+-----+-----+-----+-----+-----+
| ms_group_0 |          | ds_primary      | ds_slave_0, ds_slave_1 | random             |                      |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

动态读写分离规则

```
mysql> SHOW READWRITE_SPLITTING RULES FROM readwrite_splitting_db;
+-----+-----+-----+-----+-----+-----+
| name      | auto_aware_data_source_name | write_data_source_query_enabled | write_data_source_name | read_data_source_names | load_balancer_type | load_balancer_props |
+-----+-----+-----+-----+-----+-----+
| readwrite_ds | ms_group_0          |          |          |          | random             | read_weight:2:1 |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

静态读写分离规则和动态读写分离规则

```
mysql> SHOW READWRITE_SPLITTING RULES FROM readwrite_splitting_db;
+-----+-----+-----+-----+-----+-----+
| name      | auto_aware_data_source_name | write_data_source_query_enabled | write_data_source_name | read_data_source_names | load_balancer_type | load_balancer_props |
+-----+-----+-----+-----+-----+-----+
| readwrite_ds | ms_group_0          | write_ds          | read_ds_0, read_ds_1 | random             | read_weight=2:1 |
+-----+-----+-----+-----+-----+-----+
```

1 row in set (0.00 sec)

输出说明

列	说明
name	规则名称
auto_aware_data_source_name	自动发现数据源名称（配置动态读写分离规则显示）
write_data_source_query_enabled	读库全部下线，主库是否承担读流量
write_data_source_name	写数据源名称
read_data_source_names	读数据源名称列表
load_balancer_type	负载均衡算法类型
load_balancer_props	负载均衡算法参数

6. 查询读写分离数据源状态

SHOW STATUS **FROM** READWRITE_SPLITTING (RULES / **RULE** groupName) [**FROM** databaseName]

示例

SHOW STATUS **FROM** READWRITE_SPLITTING RULES;

12.6.19 影子库

影子库

语法	描述
CREATE SHADOW RULE shadowRuleDefinition [, shadowRuleDefinition] ...	创建影子库规则
ALTER SHADOW RULE shadowRuleDefinition [, shadowRuleDefinition] ...	修改影子库规则
DROP SHADOW RULE ruleName [, ruleName] ...	删除影子库规则
SHOW SHADOW shadowRule	RULES [FROM databaseName]
SHOW SHADOW TABLE RULES [FROM databaseName]	查看影子表
COUNT SHADOW RULE	统计影子库规则的数量，在后续版本中将废弃此语法
DROP SHADOW ALGORITHM algorithmName [, algorithmName] ...	删除影子库算法
SHOW SHADOW ALGORITHMS [FROM databaseName]	查看影子库算法
CREATE DEFAULT SHADOW ALGORITHM shadowAlgorithm	创建缺省影子库算法
ALTER DEFAULT SHADOW ALGORITHM shadowAlgorithm	修改缺省影子库算法
DROP DEFAULT SHADOW ALGORITHM [IF EXISTS]	删除缺省影子库算法
SHOW DEFAULT SHADOW ALGORITHM	查看缺省影子库算法

1. 创建影子库规则

CREATE SHADOW RULE shadowRuleDefinition [, shadowRuleDefinition] ...

shadowRuleDefinition: ruleName(storageUnitMapping, shadowTableRule [, shadowTableRule] ...)

storageUnitMapping: **SOURCE**=storageUnitName, SHADOW=storageUnitName

shadowTableRule: tableName(shadowAlgorithm [, shadowAlgorithm] ...)

shadowAlgorithm: **TYPE**(NAME=shadowAlgorithmType, PROPERTIES([algorithmProperties] ...))

algorithmProperties: algorithmProperty [, algorithmProperty] ...

algorithmProperty: **key**=value

参数说明

名称	数据类型	说明
ruleName	IDENTIFIER	规则名称
storageUnitName	IDENTIFIER	存储单元名称
tableName	IDENTIFIER	影子表名称
algorithmName	IDENTIFIER	影子算法名称
shadowAlgorithmType	STRING	影子算法类型

- 重复的 ruleName 无法被创建；
- storageUnitMapping 指定源数据库和影子库的映射关系，需使用 RDL 管理的 storage unit，请参考`存储单元 <>`__；
- shadowAlgorithm 可同时作用于多个 shadowTableRule；
- shadowAlgorithmType 目前支持的算法参考影子库内置算法；
- shadowTableRule 能够被不同的 shadowRuleDefinition 复用，因此在执行 DROP SHADOW RULE 时，对应的 shadowTableRule 不会被移除；
- shadowAlgorithm 能够被不同的 shadowTableRule 复用，因此在执行 ALTER SHADOW RULE 时，对应的 shadowAlgorithm 不会被移除。
- algorithmName 创建规则时会根据 ruleName、tableName 和 shadowAlgorithmType 和算法集合下标自动生成。默认算法名称为 default_shadow_algorithm。

示例

```
CREATE SHADOW RULE shadow_rule(
SOURCE=demo_ds,
SHADOW=demo_ds_shadow,
t_order(TYPE(NAME="SIMPLE_HINT", PROPERTIES("shadow"="true", "foo"="bar")),TYPE(NAME="REGEX_MATCH", PROPERTIES(
"operation"="insert", "column"="user_id", "regex"="[1]"))),
t_order_item(TYPE(NAME="VALUE_MATCH", PROPERTIES("operation"="insert", "column"="user_id", "value"="1"))));
```

2. 修改影子库规则

ALTER SHADOW RULE shadowRuleDefinition [, shadowRuleDefinition] ...

shadowRuleDefinition: ruleName(storageUnitMapping, shadowTableRule [, shadowTableRule] ...)

storageUnitMapping: **SOURCE**=storageUnitName, SHADOW=storageUnitName

shadowTableRule: tableName(shadowAlgorithm [, shadowAlgorithm] ...)

shadowAlgorithm: **TYPE**(NAME=shadowAlgorithmType, PROPERTIES([algorithmProperties] ...))

algorithmProperties: algorithmProperty [, algorithmProperty] ...

algorithmProperty: **key**=value

参数说明

名称	数据类型	说明
ruleName	IDENTIFIER	规则名称
storageUnitName	IDENTIFIER	存储单元名称
tableName	IDENTIFIER	影子表名称
algorithmName	IDENTIFIER	影子算法名称
shadowAlgorithmType	STRING	影子算法类型

- storageUnitMapping 指定源数据库和影子库的映射关系，需使用 RDL 管理的 storage unit，请参考 [存储单元](#)；
- shadowAlgorithm 可同时作用于多个 shadowTableRule；
- shadowAlgorithmType 目前支持 VALUE_MATCH、REGEX_MATCH 和 SIMPLE_HINT；
- shadowTableRule 能够被不同的 shadowRuleDefinition 复用，因此在执行 DROP SHADOW RULE 时，对应的 shadowTableRule 不会被移除；
- shadowAlgorithm 能够被不同的 shadowTableRule 复用，因此在执行 ALTER SHADOW RULE 时，对应的 shadowAlgorithm 不会被移除。
- algorithmName 创建规则时会根据 ruleName、tableName 和 shadowAlgorithmType 和算法集合下标自动生成。默认算法名称为 default_shadow_algorithm。

示例

```
ALTER SHADOW RULE shadow_rule(  
SOURCE=demo_ds,  
SHADOW=demo_ds_shadow,  
t_order(TYPE(NAME="SIMPLE_HINT", PROPERTIES("shadow"="true", "foo"="bar")),TYPE(NAME="REGEX_MATCH", PROPERTIES(  
"operation"="insert", "column"="user_id", "regex"="[1]"))),  
t_order_item(TYPE(NAME="VALUE_MATCH", PROPERTIES("operation"="insert", "column"="user_id", "value"='1'))));
```

3. 删除影子库规则

DROP SHADOW RULE ruleName [, ruleName] ...

参数说明

名称	数据类型	说明
ruleName	IDENTIFIER	规则名称

示例

```
DROP SHADOW RULE shadow_rule;
```

4. 查看影子库规则

```
SHOW SHADOW shadowRule | RULES [FROM databaseName]
shadowRule:
  RULE ruleName
```

参数说明

名称	数据类型	说明
ruleName	IDENTIFIER	规则名称

示例

查看所有影子库规则

```
mysql> SHOW SHADOW RULES;
+-----+-----+-----+-----+
| rule_name | source_name | shadow_name | shadow_table |
+-----+-----+-----+-----+
| shadow_rule_1 | ds_1 | ds_shadow_1 | t_order |
| shadow_rule_2 | ds_2 | ds_shadow_2 | t_order_item |
+-----+-----+-----+-----+
2 rows in set (0.02 sec)
```

查看指定影子库规则

```
mysql> SHOW SHADOW RULE shadow_rule_1;
+-----+-----+-----+-----+
| rule_name | source_name | shadow_name | shadow_table |
+-----+-----+-----+-----+
| shadow_rule_1 | ds_1 | ds_shadow_1 | t_order |
+-----+-----+-----+-----+
1 rows in set (0.01 sec)
```

输出说明

列	说明
rule_name	规则名称
source_name	源数据库
shadow_name	影子数据库
shadow_table	影子表

5. 查看影子表的影子算法

```
SHOW SHADOW TABLE RULES [FROM databaseName]
```

示例

```
mysql> SHOW SHADOW TABLE RULES;
+-----+-----+-----+-----+
| shadow_table | shadow_algorithm_name |
+-----+-----+-----+-----+
| t_order_1 | user_id_match_algorithm,simple_hint_algorithm_1 |
+-----+-----+-----+-----+
1 rows in set (0.01 sec)
```

输出说明

列	说明
shadow_table	影子表
shadow_algorithm_name	影子算法名称

6. 删除影子库算法

DROP SHADOW ALGORITHM algorithmName [, algorithmName] ...

参数说明

名称	数据类型	说明
algorithmName	IDENTIFIER	影子算法名称

- shadowAlgorithm 能够被不同的 shadowTableRule 复用，因此在执行 ALTER SHADOW RULE 时，对应的 shadowAlgorithm 不会被移除。
- 正在被使用的 shadowAlgorithm 不能被删除

7. 查看影子库算法

SHOW SHADOW ALGORITHMS [**FROM** databaseName]

示例

```
mysql> SHOW SHADOW ALGORITHMS;
+-----+-----+-----+-----+
| shadow_algorithm_name | type      | props                  | is_default |
+-----+-----+-----+-----+
| user_id_match_algorithm | REGEX_MATCH | operation=insert,column=user_id,regex=[1] | false      |
| simple_hint_algorithm_1 | SIMPLE_HINT | shadow=true,foo=bar      | false      |
+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

输出说明

列	说明
shadow_algorithm_name	影子算法名称
type	算法类型
props	算法参数
is_default	是否默认

8. 创建缺省影子库算法

CREATE DEFAULT SHADOW ALGORITHM shadowAlgorithm

shadowAlgorithm: **TYPE**(NAME=shadowAlgorithmType, PROPERTIES([algorithmProperties] ...))

algorithmProperties: algorithmProperty [, algorithmProperty] ...

algorithmProperty: **key**=value

参数说明

名称	数据类型	说明
algorithmName	IDENTIFIER	影子算法名称
shadowAlgorithmType	STRING	影子算法类型

- shadowAlgorithm 能够被不同的 shadowTableRule 复用，因此在执行 ALTER SHADOW RULE 时，对应的 shadowAlgorithm 不会被移除。
- algorithmName 创建规则时会根据 ruleName、tableName 和 shadowAlgorithmType 和算法集合下标自动生成。默认算法名称为 default_shadow_algorithm。

示例

```
CREATE DEFAULT SHADOW ALGORITHM TYPE(NAME="SIMPLE_HINT", PROPERTIES("shadow"="true", "foo"="bar");
```

9. 修改缺省影子库算法

```
ALTER DEFAULT SHADOW ALGORITHM shadowAlgorithm
```

```
shadowAlgorithm: TYPE(NAME=shadowAlgorithmType, PROPERTIES([algorithmProperties] ...))
```

```
algorithmProperties: algorithmProperty [, algorithmProperty] ...
```

```
algorithmProperty: key=value
```

参数说明

名称	数据类型	说明
algorithmName	IDENTIFIER	影子算法名称
shadowAlgorithmType	STRING	影子算法类型

- shadowAlgorithm 能够被不同的 shadowTableRule 复用，因此在执行 ALTER SHADOW RULE 时，对应的 shadowAlgorithm 不会被移除。
- algorithmName 创建规则时会根据 ruleName、tableName 和 shadowAlgorithmType 和算法集合下标自动生成。默认算法名称为 default_shadow_algorithm。

示例

```
ALTER DEFAULT SHADOW ALGORITHM TYPE(NAME="SIMPLE_HINT", PROPERTIES("shadow"="false", "foo"="bar");
```

10. 删除缺省影子库算法

```
DROP DEFAULT SHADOW ALGORITHM [IF EXISTS]
```

参数说明

无

示例

```
DROP DEFAULT SHADOW ALGORITHM;
```

11. 查看缺省影子库算法

```
SHOW DEFAULT SHADOW ALGORITHM;
```

参数说明

无

示例

```
SHOW DEFAULT SHADOW ALGORITHM;
```

12.6.20 数据迁移

数据迁移

语法	描述	类型
ALTER MIGRATION RULE	定义数据迁移规则	RAL
SHOW MIGRATION RULE	查看数据迁移规则	RAL
SHOW MIGRATION LIST	查看数据迁移规则列表	RAL
SHOW MIGRATION STATUS jobId	查看数据迁移作业状态	RAL
MIGRATE TABLE x INTO y	启动迁移作业	RAL
START MIGRATION jobId	开启停止的作业	RAL
STOP MIGRATION jobId	停止作业	RAL
CHECK MIGRATION jobId	检查作业数据一致性	RAL
START MIGRATION CHECK jobId	开启停止的数据一致性校验作业	RAL
STOP MIGRATION CHECK jobId	停止数据一致性校验作业	RAL
SHOW MIGRATION CHECK STATUS jobId	查看数据一致性校验进度	RAL
COMMIT MIGRATION jobId	提交作业	RAL
ROLLBACK MIGRATION jobId	撤销作业。注意：该语句会清理目标端表，请谨慎操作	RAL
REGISTER MIGRATION SOURCE STORAGE UNIT	注册数据迁移任务存储单元	RAL
SHOW MIGRATION CHECK ALGORITHMS	查看数据迁移算法	RAL

1. 定义数据迁移作业

```
ALTER MIGRATION RULE ();
```

配置说明

```
ALTER MIGRATION RULE (
  READ(-- 数据读取配置。如果不配置则部分参数默认生效。
    WORKER_THREAD=40,-- 从源端摄取全量数据的线程池大小。如果不配置则使用默认值。
    BATCH_SIZE=1000,-- 一次查询操作返回的最大记录数。如果不配置则使用默认值。
    SHARDING_SIZE=10000000,-- 全量数据分片大小。如果不配置则使用默认值。
    RATE_LIMITER (-- 限流算法。如果不配置则不限流。
      TYPE(-- 算法类型。可选项： QPS
        NAME='QPS',
        PROPERTIES(-- 算法属性
          'qps'='500'
        ))
    ),
  WRITE(-- 数据写入配置。如果不配置则部分参数默认生效。
    WORKER_THREAD=40,-- 数据写入到目标端的线程池大小。如果不配置则使用默认值。
    BATCH_SIZE=1000,-- 一次批量写入操作的最大记录数。如果不配置则使用默认值。
    RATE_LIMITER (-- 限流算法。如果不配置则不限流。
      TYPE(-- 算法类型。可选项： TPS
```

```

NAME='TPS',
PROPERTIES(-- 算法属性
'tps'='2000'
)))
),
STREAM_CHANNEL(-- 数据通道, 连接生产者和消费者, 用于 read 和 write 环节。如果不配置则默认使用 MEMORY 类型。
TYPE(-- 算法类型。可选项: MEMORY
NAME='MEMORY',
PROPERTIES(-- 算法属性
'block-queue-size'='10000' -- 属性: 阻塞队列大小
)))
);

```

默认配置:

```

+-----+-----+-----+
| read | write | stream_channel |
+-----+-----+-----+
| {"workerThread":40,"batchSize":1000,"shardingSize":10000000} | {"workerThread":40,"batchSize":1000} | {"type":"MEMORY",
"props":{"block-queue-size":10000}} |
+-----+-----+-----+

```

示例

```
ALTER MIGRATION RULE (READ(RATE_LIMITER (TYPE(NAME='QPS' ,PROPERTIES( 'qps'='5000' )))))
```

2. 查看数据迁移规则

SHOW MIGRATION RULE

参数说明

无

示例

```
SHOW MIGRATION RULE;
```

输出说明

列名	说明
read	数据读取配置。如果不配置则部分参数默认生效
write	数据写入配置。如果不配置则部分参数默认生效。
stream_channel	数据通道, 连接生产者和消费者, 用于 read 和 write 环节。如果不配置则默认使用 MEMORY 类型。

3. 查看数据迁移规则列表

SHOW MIGRATION LIST;

参数说明

无

示例

```

+-----+-----+-----+-----+-----+-----+
| id          | tables | sharding_total_count | active | create_time | stop_time |
+-----+-----+-----+-----+-----+-----+
| j01016e501b498ed1bdb2c373a2e85e2529a6 | t_order | 1 | true | 2022-08-22 16:37:01 | NULL |
+-----+-----+-----+-----+-----+-----+

```

输出说明

列名	说明
Id	迁移任务 ID
tables	被迁移的表名
active	任务是否活跃 true 或 false
create_time	任务创建时间
stop_time	任务停止时间，为 NULL 表示任务未停止

4. 查看数据迁移作业状态

```
SHOW MIGRATION STATUS jobId;
```

参数说明

参数名	说明
jobId	数据迁移作业 id

示例

```
SHOW MIGRATION STATUS 'j01016e501b498ed1bdb2c373a2e85e2529a6';
+-----+-----+-----+-----+-----+-----+-----+
| item | data_source | status | active | processed_records_count | inventory_finished_percentage | incremental_idle_seconds | error_message |
+-----+-----+-----+-----+-----+-----+-----+
| 0 | ds_0 | EXECUTE_INCREMENTAL_TASK | true | 6 | 100 | 81 | |
```

输出说明

列名	说明
item	迁移任务内序号
data_source	数据存储单元名称
status	任务状态，详细参考状态描述
active	任务是否活跃 true 或 false
processed_records_count	处理完成的记录数
inventory_finished_percentage	全量数据处理完成进度（百分比）
incremental_idle_seconds	增量同步任务不活跃时间（单位：秒）
error_message	错误信息

状态说明

取值	描述
PREPARING	准备中
RUNNING	运行中
EXECUTE_INVENTORY_TASK	全量迁移中
EXECUTE_INCREMENTAL_TASK	增量迁移中
FINISHED	已完成（整个流程完成了，新规则已生效）
PREPARING_FAILURE	准备阶段失败
EXECUTE_INVENTORY_TASK_FAILURE	全量迁移阶段失败
EXECUTE_INCREMENTAL_TASK_FAILURE	增量迁移阶段失败

5. 启动数据迁移

```
MIGRATE TABLE schema_name.table_name INTO schema_name.table_name;
```

示例
存储节点为 MySQL 启动数据迁移。

```
MIGRATE TABLE ds_0.t_order INTO t_order;
```

存储节点为 MySQL 或者指定目标端逻辑库：

```
MIGRATE TABLE ds_0.t_order INTO sharding_db.t_order;
```

同时迁移多张表，多张表使用逗号分隔

```
MIGRATE TABLE ds_0.t_order INTO sharding_db.t_order,ds_0.t_order1 INTO sharding_db.t_order1,ds_0.t_order2 INTO sharding_db.t_order2;
```

6. 开启已经停止的作业

```
START MIGRATION jobId
```

参数说明

参数名	说明
jobId	作业 id

示例
指定 jobId 启动已经停止的 job。

```
START MIGRATION 'j01016e501b498ed1bdb2c373a2e85e2529a6';
```

7. 停止的作业

```
STOP MIGRATION jobId
```

参数说明

参数名	说明
jobId	作业 id

示例
指定 jobId 启动已经停止的 job。

```
STOP MIGRATION 'j01016e501b498ed1bdb2c373a2e85e2529a6';
```

8. 执行数据一致性校验

```
CHECK MIGRATION jobId BY TYPE (NAME='CRC32_MATCH');
```

参数名	说明
jobId	作业 id

示例
指定 jobId 指定校验算法执行数据一致性校验

```
CHECK MIGRATION 'j01016e501b498ed1bdb2c373a2e85e2529a6' BY TYPE (NAME='CRC32_MATCH');  
Query OK, 0 rows affected (0.09 sec)
```

指定 jobId 使用默认校验算法执行数据一致性校验

```
CHECK MIGRATION 'j01016e501b498ed1bdb2c373a2e85e2529a6';  
Query OK, 0 rows affected (0.09 sec)
```

9. 开启停止的数据一致校验作业

```
START MIGRATION CHECK jobId
```

参数名	说明
jobId	作业 id

示例
指定 jobId 指定校验算法执行数据一致性校验

```
START MIGRATION CHECK 'j01016e501b498ed1bdb2c373a2e85e2529a6';  
Query OK, 0 rows affected (0.09 sec)
```

10. 停止数据一致校验作业

```
STOP MIGRATION CHECK jobId
```

参数名	说明
jobId	作业 id

示例
指定 jobId 指定校验算法执行数据一致性校验

```
STOP MIGRATION CHECK 'j01016e501b498ed1bdb2c373a2e85e2529a6';  
Query OK, 0 rows affected (0.09 sec)
```

11. 查看数据一致性校验进度

```
SHOW MIGRATION CHECK STATUS jobId
```

参数名	说明
jobId	作业 id

示例
指定 jobId 指查看数据一致性校验进度

```
SHOW MIGRATION CHECK STATUS 'j01016e501b498ed1bdb2c373a2e85e2529a6';
+-----+-----+-----+-----+-----+-----+-----+
| tables | result | finished_percentage | remaining_seconds | check_begin_time   | check_end_time     | duration_seconds |
| error_message |
+-----+-----+-----+-----+-----+-----+-----+
| t_order | true  | 100          | 0          | 2022-10-13 11:18:15.171 | 2022-10-13 11:18:15.878 | 0          |
+-----+-----+-----+-----+-----+-----+-----+
```

12. 提交作业

```
COMMIT MIGRATION jobId;
```

参数说明

参数名	说明
jobId	作业 id

示例
指定 jobId 提交作业。

```
COMMIT MIGRATION 'j01016e501b498ed1bdb2c373a2e85e2529a6';
```

13. 撤销作业

```
ROLLBACK MIGRATION jobId;
```

参数名	说明
jobId	作业 id

注意：该语句会清理目标端表，请谨慎操作。
示例
指定 jobId 回滚 job。

```
ROLLBACK MIGRATION 'j01016e501b498ed1bdb2c373a2e85e2529a6';
```

14. 注册数据迁移任务存储单元

REGISTER MIGRATION **SOURCE STORAGE** UNIT

参数说明

参考[存储节点部分](#)

示例

```
REGISTER MIGRATION SOURCE STORAGE UNIT ds_0 (
  URL="jdbc:mysql://127.0.0.1:3306/migration_ds_0?serverTimezone=UTC&useSSL=false",
  USER="root",
  PASSWORD="root",
  PROPERTIES("minPoolSize"="1","maxPoolSize"="20","idleTimeout"="60000")
);
```

15. 查看数据迁移算法

SHOW MIGRATION **CHECK** ALGORITHMS;

参数说明

无

示例

```
SHOW MIGRATION CHECK ALGORITHMS;
```

type	supported_database_types	description
CRC32_MATCH	MySQL	Match CRC32 of records.
DATA_MATCH	SQL92,MySQL,MariaDB,PostgreSQL,openGauss,Oracle,SQLServer,H2	Match raw data of records.

12.6.21 扩缩容

扩缩容

语法	描述	类型
ALTER RESHARDING RULE	定义扩缩容规则	RAL
RESHARD TABLE tableName by rule	启动扩缩容	RAL
SHOW RESHARDING RULE	查看扩缩容规则	RAL
SHOW RESHARDING LIST	查看扩缩容作业列表	RAL
SHOW RESHARDING STATUS jobId	查看扩缩容作业状态	RAL
START RESHARDING jobId	启动已经停止的扩缩容作业	RAL
STOP RESHARDING jobId	停止扩缩容作业	RAL
CHECK RESHARDING jobId	执行扩缩容数据一致性校验	RAL
START RESHARDING CHECK jobId	启动停止的扩缩容数据一致性校验作业	RAL
STOP RESHARDING CHECK jobId	停止的扩缩容数据一致性校验作业	RAL
DROP RESHARDING CHECK jobId	删除一致性校验作业	RAL
COMMIT RESHARDING jobId	提交扩缩容作业	RAL
FORCE COMMIT RESHARDING jobId	强制提交扩缩容作业	RAL
ROLLBACK RESHARDING jobId	撤销扩缩容作业	RAL
APPLY RESHARDING jobId	应用扩缩容新配置（切换元数据）	RAL
FORCE APPLY RESHARDING jobId	强制应用扩缩容新配置（切换元数据）	RAL
SHOW RESHARDING CHECK ALGORITHMS	查看扩缩容算法	RAL
STOP RESHARDING SOURCE WRITING jobId	停写	RAL
RESTORE RESHARDING SOURCE WRITING jobId	恢复停写	RAL
AUTO RESHARD TABLE tableName BY rule	扩缩容自动模式	RAL

1. 定义扩缩容规则

ALTER RESHARDING RULE

示例

ALTER RESHARDING RULE (READ(RATE_LIMITER (TYPE(NAME='QPS' ,PROPERTIES('qps'='5000')))))

2. 启动扩缩容

RESHARD TABLE tableName BY rule

参数说明

参数名	说明
tableName	执行扩缩容的表

示例

```

RESHARD TABLE t_order BY(
  STORAGE_UNITS(ds_0, ds_1, ds_2),
  SHARDING_COLUMN=order_id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="6")),
  KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);
    
```

3. 查看扩缩容规则

```
SHOW RESHARDING LIST;
```

参数说明

无

示例

```
SHOW RESHARDING LIST;
+-----+-----+-----+-----+-----+-----+
| id           | tables | sharding_total_count | active | create_time   | stop_time |
+-----+-----+-----+-----+-----+-----+
| j51017f973ac82cb1edea4f5238a258c25e89 | t_order | 2           | true  | 2022-10-25 10:10:58 | NULL      |
+-----+-----+-----+-----+-----+-----+
```

4. 查看扩缩容作业列表

```
SHOW RESHARDING LIST;
```

参数说明

无

示例

```
SHOW RESHARDING LIST;
+-----+-----+-----+-----+-----+-----+
| id           | tables | sharding_total_count | active | create_time   | stop_time |
+-----+-----+-----+-----+-----+-----+
| j51017f973ac82cb1edea4f5238a258c25e89 | t_order | 2           | true  | 2022-10-25 10:10:58 | NULL      |
+-----+-----+-----+-----+-----+-----+
```

5. 查看扩缩容作业状态

```
SHOW RESHARDING STATUS jobId;
```

参数说明

参数名	说明
jobid	作业 id

示例

```
SHOW RESHARDING STATUS 'j51017f973ac82cb1edea4f5238a258c25e89';
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| item | data_source | status           | active | processed_records_count | inventory_finished_percentage | remaining_seconds | incremental_idle_seconds | error_message |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 0    | ds_0        | EXECUTE_INCREMENTAL_TASK | true  | 3           | 100           | 0           | 92           |              |
| 1    | ds_1        | EXECUTE_INCREMENTAL_TASK | true  | 3           | 100           | 0           | 92           |              |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

6. 启动已经停止的扩缩容作业

START RESHARDING jobId

参数说明

参数名	说明
jobid	作业 id

示例

START RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';

7. 停止扩缩容作业

STOP RESHARDING jobId

参数说明

参数名	说明
jobid	作业 id

示例

STOP RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';

8. 执行扩缩容数据一致性

CHECK RESHARDING jobId

参数说明

参数名	说明
jobid	作业 id

示例

指定 jobId 指定校验算法执行数据一致性校验

CHECK RESHARDING 'j01016e501b498ed1bdb2c373a2e85e2529a6' **BY TYPE** (NAME='CRC32_MATCH');
Query OK, 0 rows affected (0.09 sec)

指定 jobId 使用默认校验算法执行数据一致性校验

CHECK RESHARDING 'j01016e501b498ed1bdb2c373a2e85e2529a6';
Query OK, 0 rows affected (0.09 sec)

9. 启动停止的扩缩容数据一致性检测作业

```
START RESHARDING CHECK jobId
```

参数说明

参数名	说明
jobid	作业 id

示例

```
START RESHARDING CHECK 'j51017f973ac82cb1edea4f5238a258c25e89';
```

10. 停止扩缩容数据一致性检测作业

```
STOP RESHARDING CHECK jobId
```

参数说明

参数名	说明
jobid	作业 id

示例

```
STOP RESHARDING CHECK 'j51017f973ac82cb1edea4f5238a258c25e89';
```

11. 删除扩缩容数据一致性检测作业

```
DROP RESHARDING CHECK jobId
```

参数说明

参数名	说明
jobid	作业 id

示例

```
DROP RESHARDING CHECK 'j51017f973ac82cb1edea4f5238a258c25e89';
```

12. 提交扩缩容作业

```
COMMIT RESHARDING jobId
```

参数说明

参数名	说明
jobid	作业 id

示例

```
COMMIT RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

13. 强制提交扩缩容作业

FORCE COMMIT RESHARDING jobId

参数说明

参数名	说明
jobid	作业 id

示例

FORCE COMMIT RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';

14. 撤销扩缩容作业

ROLLBACK RESHARDING jobId

参数说明

参数名	说明
jobid	作业 id

示例

ROLLBACK RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';

15. 应用扩缩容新配置（切换元数据）

APPLY RESHARDING jobId

参数说明

参数名	说明
jobid	作业 id

示例

APPLY RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';

16. 强制应用扩缩容新配置（切换元数据）

FORCE APPLY RESHARDING jobId

参数说明

参数名	说明
jobid	作业 id

示例

FORCE APPLY RESHARDING 'j51017f973ac82cb1edea4f5238a258c25e89';

17. 查看扩缩容算法

```
SHOW RESHARDING CHECK ALGORITHMS;
```

参数说明

无

示例

```
SHOW RESHARDING CHECK ALGORITHMS;
+-----+-----+-----+
| type | supported_database_types | description |
+-----+-----+-----+
| CRC32_MATCH | MySQL | Match CRC32 of records. |
| DATA_MATCH | SQL92,MySQL,MariaDB,PostgreSQL,openGauss,Oracle,SQLServer,H2 | Match raw data of records. |
+-----+-----+-----+
```

18. 停写

```
STOP RESHARDING SOURCE WRITING jobId
```

参数说明

参数名	说明
jobid	作业 id

示例

```
STOP RESHARDING SOURCE WRITING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

19. 恢复停写

```
RESTORE RESHARDING SOURCE WRITING jobId
```

参数说明

参数名	说明
jobid	作业 id

示例

```
RESTORE RESHARDING SOURCE WRITING 'j51017f973ac82cb1edea4f5238a258c25e89';
```

20. 扩所容自动模式

```
AUTO RESHARD TABLE tableName BY rule
```

参数说明

参数名	说明
tableName	执行扩缩容的表

示例

```
AUTO RESHARD TABLE t_order BY(
  STORAGE_UNITS(ds_0, ds_1, ds_2),
  SHARDING_COLUMN=order_id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="6")),
  KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);
```

21. 扩所容自动模式

```
AUTO RESHARD TABLE tableName BY rule
```

参数说明

参数名	说明
tableName	执行扩缩容的表

示例

```
AUTO RESHARD TABLE t_order BY(
  STORAGE_UNITS(ds_0, ds_1, ds_2),
  SHARDING_COLUMN=order_id,
  TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="6")),
  KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);
```

12.6.22 SQL 工具

SQL 工具

语句	说明	示例	类型
PARSE SQL	解析 SQL 并输出抽象语法树	PARSE SELECT * FROM t_order	RUL
FORMAT SQL	解析并输出格式化后的 SQL 语句	FORMAT SELECT * FROM t_order	RUL
PREVIEW SQL	预览 SQL 执行计划	PREVIEW SELECT * FROM t_order	RUL

12.6.23 版本序列号

版本序列号

语法	描述	类型
SHOW SERIES NUMBER;	查看计算节点（proxy）的版本序列号	RQL

1. 查看版本序列号

REGISTER LICENSE

示例

```
SHOW SERIES NUMBER;
+-----+
| commit_id |
+-----+
| m8e0d922df97bb9ce622633288a5131cc89b499e |
+-----+
```

12.6.24 归档

归档

语法	描述	类型
SHOW ARCHIVE RULES [FROM schemaName]	查看归档规则	RQL
CREATE ARCHIVE RULE ruleName (ARCHIV E_STRATEGY(ARCHIVE_COLUMN=columnName, ARCHIVE_ALGORI THM(TYPE(NAME= "archiveAlgorithmName", PROPERTIES("interval-unit" = "MONTHS", "interval-amount" =1, "retain-amount" =12, "pre-allocate-amount" =2))), SCHED- ULER_ALGORITH M(TYPE(NAME= "schedulerAlgorithmName", PROPERTIES("schedule-cron-expr" = "0 15 10 ? * * "))))	创建归档规则	RDL
ALTER ARCHIVE RULE ruleName (ARCHIV E_STRATEGY(ARCHIVE_COLUMN=columnName, ARCHIVE_ALGORI THM(TYPE(NAME= "archiveAlgorithmName", PROPERTIES("interval-unit" = "MONTHS", "interval-amount" =1, "retain-amount" =12, "pre-allocate-amount" =2))), SCHED- ULER_ALGORITH M(TYPE(NAME= "schedulerAlgorithmName", PROPERTIES("schedule-cron-expr" = "0 15 10 ? * * "))))	修改归档规则	RDL
DROP ARCHIVE RULE [IF EXISTS] ruleName	修改归档规则	RDL

1. 查看归档规则

SHOW ARCHIVE RULES [FROM schemaName]

示例

查看所有的归档规则

```
SHOW ARCHIVE RULES;
SHOW ARCHIVE RULES FROM sharding_db;
+-----+-----+-----+-----+-----+
| | | | | |
+-----+-----+-----+-----+-----+
```


table	archive_column	archive_algorithm_type	archive_algorithm_props	storage_algorithm_type
type	storage_algorithm_props	scheduler_algorithm_type	scheduler_algorithm_props	
t_order	createtime	SphereEx:TIME_PARTITION	interval-amount=1,pre-allocate-amount=2,interval-unit=MONTHS,retain-amount=12	
t_order_item	createtime	SphereEx:TIME_PARTITION	interval-amount=2,pre-allocate-amount=4,interval-unit=DAYS,retain-amount=13	
		SphereEx:TIMED_TASK	schedule-cron-expr=0 15 10 ? * *	
		SphereEx:TIMED_TASK	schedule-cron-expr=0 13 9 ? * *	

查看指定归档规则

```
SHOW ARCHIVE TABLE RULE t_order;
SHOW ARCHIVE TABLE RULE t_order FROM sharding_db;
```

table	archive_column	archive_algorithm_type	archive_algorithm_props	storage_algorithm_type
type	storage_algorithm_props	scheduler_algorithm_type	scheduler_algorithm_props	
t_order	createtime	SphereEx:TIME_PARTITION	interval-amount=1,pre-allocate-amount=2,interval-unit=MONTHS,retain-amount=12	
		SphereEx:TIMED_TASK	schedule-cron-expr=0 15 10 ? * *	

输出说明

列	说明
table	归档的表名
archive_column	归档字段
archive_algorithm_type	归档算法
archive_algorithm_props	归档算法参数
storage_algorithm_type	存储算法
storage_algorithm_props	存储算法参数
scheduler_algorithm_type	调度算法
scheduler_algorithm_props	调度算法参数

2. 创建归档规则

```
CREATE ARCHIVE RULE ruleName ( ARCHIVE_STRATEGY(ARCHIVE_COLUMN=columnName, ARCHIVE_ALGORITHM(TYPE(NAME=
"archiveAlgorithmName", PROPERTIES("interval-unit"="MONTHS", "interval-amount"=1, "retain-amount"=12, "pre-allocate-
amount"=2))), SCHEDULER_ALGORITHM(TYPE(NAME="schedulerAlgorithmName", PROPERTIES("schedule-cron-expr"="0 15 10
? * *"))))
```

示例

```
CREATE ARCHIVE RULE t_order (
  ARCHIVE_STRATEGY(ARCHIVE_COLUMN=createtime, ARCHIVE_ALGORITHM(TYPE(NAME="SphereEx:TIME_PARTITION",
  PROPERTIES("interval-unit"="MONTHS", "interval-amount"=1, "retain-amount"=12, "pre-allocate-amount"=2))),
  SCHEDULER_ALGORITHM(TYPE(NAME="SphereEx:TIMED_TASK", PROPERTIES("schedule-cron-expr"="0 15 10 ? * *"))
);
```

参数说明

相关算法请参考[内置算法](#)

3. 修改归档规则

```
ALTER ARCHIVE RULE ruleName ( ARCHIVE_STRATEGY(ARCHIVE_COLUMN=columnName, ARCHIVE_ALGORITHM(TYPE(NAME="archiveAlgorithmName", PROPERTIES("interval-unit"="MONTHS", "interval-amount"=1, "retain-amount"=12, "pre-allocate-amount"=2))), SCHEDULER_ALGORITHM(TYPE(NAME="schedulerAlgorithmName", PROPERTIES("schedule-cron-expr"="0 15 10 ? * *")))) )
```

示例

```
ALTER ARCHIVE RULE t_order (
  ARCHIVE_STRATEGY(ARCHIVE_COLUMN=createtime, ARCHIVE_ALGORITHM(TYPE(NAME="SphereEx:TIME_PARTITION",
  PROPERTIES("interval-unit"="MONTHS", "interval-amount"=1, "retain-amount"=12, "pre-allocate-amount"=2))),
  SCHEDULER_ALGORITHM(TYPE(NAME="SphereEx:TIMED_TASK", PROPERTIES("schedule-cron-expr"="0 15 10 ? * *"))))
);
```

参数说明

相关算法请参考[内置算法](#)

4. 删除归档规则

```
DROP ARCHIVE RULE ruleName ;
```

示例

```
DROP ARCHIVE RULE t_order ;
```

12.7 YAML 配置

12.7.1 简介

在 server.yaml 文件中，SphereEx-DBPlusEngine-Proxy 能够额外配置权限功能和更多的 Proxy 专有属性。

说明：YAML 配置文件支持配置内容超过 3MB。

本章节将介绍 SphereEx-DBPlusEngine-Proxy 的 YAML 额外配置。

12.7.2 模式配置

配置项说明

```
mode (?): # 不配置则默认内存模式
type: # 运行模式类型。可选配置：Cluster
repository (?): # 久化仓库配置。Memory 类型无需持久化
```

集群模式

```
mode:
type: Cluster
repository:
type: # 持久化仓库类型
props: # 持久化仓库所需属性
namespace: # 注册中心命名空间
server-lists: # 注册中心连接地址
foo_key: foo_value
bar_key: bar_value
```

持久化仓库类型的详情，请参见[内置持久化仓库类型列表](#)。

12.7.3 认证和授权

背景信息

在 SphereEx-DBPlusEngine-Proxy 中，通过 authority 来配置用户的认证和授权信息。

得益于 SphereEx-DBPlusEngine 的可插拔架构，Proxy 提供了两种级别的权限提供者，分别是：

- ALL_PERMITTED：每个用户都拥有所有权限，无需专门授权；
- DATABASE_PERMITTED：为用户授予指定逻辑库的权限，通过 user-database-mappings 进行定义。
- SphereEx:PERMITTED：企业级权限提供者，可进行库、表、列级细粒度授权管理。

在配置 authority 时，管理员可根据需要选择使用哪一种权限提供者。

参数解释

```
authority:
users:
- user: # 用于登录计算节点的用户名和授权主机的组合，格式：<username>@<hostname>，hostname 为 % 或空字符串表示不限制授权主机
password: # 用户密码
authenticationMethodName: # 可选项，用于为用户指定密码认证方式
authenticators: # 可选项，默认不需要配置，Proxy 根据前端协议类型自动选择
authenticatorName:
type: # 密码认证类型
defaultAuthenticator: # 可选项，指定一个 authenticatorName 作为默认的密码认证方式
privilege:
type: # 权限提供者类型，缺省值为 SphereEx:PERMITTED
```

配置示例

极简配置

```
authority:
users:
- user: root@%
password: root
- user: sharding
password: sharding
```

说明：- 定义了两个用户：root@% 和 sharding；- 未定义 authenticators 和 authenticationMethodName，Proxy 将根据前端协议自动选择；- 未指定 privilege type，采用默认的 SphereEx:PERMITTED。

认证配置

自定义认证配置能够满足用户在一些特定场景下的需求。以 openGauss 作为前端协议类型为例，其默认的认证算法为 scram-sha-256。如果用户 sharding 需要用旧版本的 psql 客户端（不支持 scram-sha-256）连接 Proxy，则管理员可能允许 sharding 使用 md5 方式进行密码认证。

支持的协议情况如下：- MySQL 协议支持：mysql_native_password、mysql_clear_password - Postgre 协议支持：password、md5 - openGauss 协议支持：scram-sha-256、md5

配置方式如下：

```
authority:
  users:
    - user: root@127.0.0.1
      password: root
    - user: sharding
      password: sharding
      authenticationMethodName: md5
  authenticators:
    md5:
      type: MD5
  privilege:
    type: SphereEx:PERMITTED
```

说明：- 定义了两个用户：root@127.0.0.1 和 sharding；- 为用户 sharding 指定了 MD5 方式进行密码认证；- 没有为 root@127.0.0.1 指定认证方式，Proxy 将根据前端协议自动选择；- 指定权限提供者 SphereEx:PERMITTED。

授权配置

ALL_PERMITTED

```
authority:
  users:
    - user: root@127.0.0.1
      password: root
    - user: sharding
      password: sharding
  privilege:
    type: ALL_PERMITTED
```

说明：- 定义了两个用户：root@127.0.0.1 和 sharding；- 未定义 authenticators 和 authenticationMethodName，Proxy 将根据前端协议自动选择；- 指定权限提供者 ALL_PERMITTED。

DATABASE_PERMITTED

```
authority:
  users:
    - user: root@127.0.0.1
      password: root
    - user: sharding
      password: sharding
  privilege:
    type: DATABASE_PERMITTED
  props:
    user-database-mappings: root@127.0.0.1=*, sharding@%=test_db, sharding@%=sharding_db
```

说明：- 定义了两个用户：root@127.0.0.1 和 sharding；- 未定义 authenticators 和 authenticationMethodName，Proxy 将根据前端协议自动选择；- 指定权限提供者 SphereEx:PERMITTED，并授权 root@127.0.0.1 用户访问所有逻辑库 (*)，sharding 用户仅能访问 test_db 和 sharding_db。

SphereEx:PERMITTED

```
authority:
users:
- user: root@127.0.0.1
  password: root
- user: sharding
  password: sharding
privilege:
type: SphereEx:PERMITTED
props:
user-database-mappings: root@127.0.0.1=*, sharding@%=test_db, sharding@%=sharding_db
```

说明： - 定义了两个用户：root@127.0.0.1 和 sharding； - 未定义 authenticators 和 authenticationMethodName，Proxy 将根据前端协议自动选择； - 指定权限提供者 DATABASE_PERMITTED，并授权 root@127.0.0.1 用户访问所有逻辑库 (*) 的表和列。

相关参考

权限提供者具体实现可以参考 [权限提供者](#)。

12.7.4 属性配置

背景信息

SphereEx-DBPlusEngine-Proxy 提供了丰富的系统配置属性，用户可通过 server.yaml 进行配置。

参数解释

名称	数据类型 *	说明	默认值 *	动 态 生 效 *
sql-federation-type	String	联邦查询执行器类型，包括：NONE，ORIGINAL，ADVANCED。	NONE	是
agent-plugin-enabled	boolean	是否启用 agent plugin 插件。TRUE/FALSE	TRUE	否
transaction-type	String	事务类型。LOCAL/XA/BASE/MANUALXA	LOCAL	否
kernel-executor-size	int	用于设置任务处理线程池的大小。每个 ShardingSphere-DataSource 使用一个独立的线程池，同一个 JVM 的不同数据源不共享线程池。	infinite	否
max-connections-size-per-query	int	一次查询请求在每个数据库实例中所能使用的最大连接数。	1	是
check-table-meta-data-enabled	boolean	在程序启动和更新时，是否检查分片元数据的结构一致性。	false	是
proxy-front-end-flush-threshold	int	在 ShardingSphere-Proxy 中设置传输数据条数的 IO 刷新阈值。	128	是
proxy-hint-enabled	boolean	是否允许在 ShardingSphere-Proxy 中使用 Hint。使用 Hint 会将 Proxy 的线程处理模型由 IO 多路复用变更为每个请求一个独立的线程，会降低 Proxy 的吞吐量。	false	是
proxy-backend-query-fetch-size	int	Proxy 后端与数据库交互的每次获取数据行数（使用游标的情况下）。数值增大可能会增加 ShardingSphere Proxy 的内存使用。默认值为-1，代表设置为 JDBC 驱动的最小值。	1	是
proxy-front-end-executor-size	int	Proxy 前端 Netty 线程池线程数量，默认值 0 代表使用 Netty 默认值。	0	否
proxy-backend-executor-suitable	String	可选选项：OLAP、OLTP。OLTP 选项可能会减少向客户端写入数据包的时间开销，但如果客户端连接数超过 proxy-front-end-executor-size，尤其是执行慢 SQL 时，它可能会增加 SQL 执行的延迟甚至阻塞其他客户端的连接。	OLAP	是
				640
proxy-front-end-	int	允许连接 Proxy 的最	0	是

属性配置可以通过 DistSQL 在线修改。其中支持动态修改的属性立即生效，不支持动态修改的属性在重启后生效。

配置示例

完整配置示例请参考：

```
#props:
# max-connections-size-per-query: 1
# kernel-executor-size: 16 # Infinite by default.
# proxy-frontend-flush-threshold: 128 # The default value is 128.
# proxy-hint-enabled: false
# sql-show: false
# check-table-metadata-enabled: false
# show-process-list-enabled: false
# # Proxy backend query fetch size. A larger value may increase the memory usage of ShardingSphere Proxy.
# # The default value is -1, which means set the minimum value for different JDBC drivers.
# proxy-backend-query-fetch-size: -1
# check-duplicate-table-enabled: false
# proxy-frontend-executor-size: 0 # Proxy frontend executor size. The default value is 0, which means let Netty decide.
# # Available options of proxy backend executor suitable: OLAP(default), OLTP. The OLTP option may reduce time cost of writing
# packets to client, but it may increase the latency of SQL execution
# # and block other clients if client connections are more than `proxy-frontend-executor-size`, especially executing slow SQL.
# proxy-backend-executor-suitable: OLAP
# proxy-frontend-max-connections: 0 # Less than or equal to 0 means no limitation.
# sql-federation-enabled: false
# # Available proxy backend driver type: JDBC (default), ExperimentalVertex
# proxy-backend-driver-type: JDBC
# proxy-mysql-default-version: 5.7.22 # In the absence of schema name, the default version will be used.
# proxy-default-port: 3307 # Proxy default port.
# proxy-netty-backlog: 1024 # Proxy netty backlog.
```

12.8 内置算法

12.8.1 简介

DBPlusEngine 通过 SPI 方式允许开发者扩展算法；与此同时，DBPlusEngine 也提供了大量的内置算法以便于开发者使用。

12.8.2 使用方式

内置算法均通过 type 和 props 进行配置，其中 type 由算法定义在 SPI 中，props 用于传递算法的个性化参数配置。

无论使用哪种配置方式，均是将配置完毕的算法命名，并传递至相应的规则配置中。本章节根据功能区分并罗列 DBPlusEngine 全部的内置算法，供开发者参考。

12.8.3 元数据持久化仓库

背景信息

DBPlusEngine 为不同的运行模式提供了不同的元数据持久化方式，用户在配置运行模式的同时可以选择合适的方式来存储元数据。

参数解释

ZooKeeper 持久化

类型: ZooKeeper
适用模式: Cluster
可配置属性:

名称	数据类型	说明	默认值
retryIntervalMilliseconds	int	重试间隔毫秒数	500
maxRetries	int	客户端连接最大重试次数	3
timeToLiveSeconds	int	临时数据失效的秒数	60
operationTimeoutMilliseconds	int	客户端操作超时的毫秒数	500
digest	String	登录认证密码	

Etcd 持久化

类型: Etcd
适用模式: Cluster
可配置属性:

名称	数据类型	说明	默认值
timeToLiveSeconds	long	临时数据失效的秒数	30
connectionTimeout	long	连接超时秒数	30

Mate 持久化

类型: SphereEx:MATE
适用模式: Cluster
可配置属性:
暂无

操作步骤

- 1. 在 server.yaml 中配置 Mode 运行模式
- 2. 配置元数据持久化仓库类型

配置示例

■ 集群模式

```
mode:  
type: Cluster  
repository:  
  type: SphereEx:MATE  
props:  
  namespace: governance  
  server-lists: localhost:21506
```


12.8.4 分片算法

自动分片算法

取模分片算法

类型：MOD

可配置属性：

属性名称	数据类型	说明
sharding-count	int	分片数量

哈希取模分片算法

类型：HASH_MOD

可配置属性：

属性名称	数据类型	说明
sharding-count	int	分片数量

基于分片容量的范围分片算法

类型：VOLUME_RANGE

可配置属性：

属性名称	数据类型	说明
range-lower	long	范围下界，超过边界的数据会报错
range-upper	long	范围上界，超过边界的数据会报错
sharding-volume	long	分片容量

基于分片边界的范围分片算法

类型：BOUNDARY_RANGE

可配置属性：

属性名称	数据类型	说明
sharding-ranges	String	分片的范围边界，多个范围边界以逗号分隔

自动时间段分片算法

类型：AUTO_INTERVAL

可配置属性：

属性名称	数据类型	说明
datetime-lower	String	分片的起始时间范围，时间戳格式：yyyy-MM-dd HH:mm:ss
datetime-upper	String	分片的结束时间范围，时间戳格式：yyyy-MM-dd HH:mm:ss
sharding-seconds	long	单一分片所能承载的最大时间，单位：秒，允许分片键的时间戳格式的秒带有时间精度，但秒后的时间精度会被自动抹去

一致性 Hash 分片算法

类型：SphereEx:CONSISTENT_HASH

可配置属性：

属性名称	数据类型	说明
sharding-count	int	必填项，用于指定分片数。
sharding-node-weight	String	非必填项，分片数据节点的权重，以逗号分隔，例如：“2,2,1,1”。权重越大，用于一致性 Hash 计算的虚拟节点越多，默认权重为 1。
virtual-node-count	int	必填项，分片数据节点对应的虚拟节点个数，虚拟节点数量越多，分片越均衡，但过多的虚拟节点也会影响性能。
consistent-hash-seed	int	非必填项，计算一致性 Hash 的种子参数，默认值为 0。

标准分片算法

DB Plus Engine 内置的标准分片算法实现类包括：

行表达式分片算法

使用 Groovy 的表达式，提供对 SQL 语句中的 = 和 IN 的分片操作支持，只支持单分片键。对于简单的分片算法，可以通过简单的配置使用，从而避免繁琐的 Java 代码开发，如: t_user_\${u_id % 8} 表示 t_user 表根据 u_id 模 8，而分成 8 张表，表名称为 t_user_0 到 t_user_7。详情请参见[行表达式](#)。

类型：INLINE

可配置属性：

属性名称	数据类型	说明	默认值
algorithm-expression	String	分片算法的行表达式	
allow-range-query-with-inline-sharding (?)	boolean	是否允许范围查询。注意：范围查询会无视分片策略，进行全路由	false

时间范围分片算法

类型：INTERVAL

可配置属性：

属性名称	数据类型	说明	默认值
datetime-pattern	String	分片键的时间戳格式，必须遵循 Java DateTimeFormatter 的格式。例如：yyyy-MM-dd HH:mm:ss	
datetime-lower	String	时间分片下界值，格式与 datetime-pattern 定义的时间戳格式一致	
datetime-upper(?)	String	时间分片上界值，格式与 datetime-pattern 定义的时间戳格式一致	当前时间
sharding-suffix-pattern	String	分片数据源或真实表的后缀格式，必须遵循 Java DateTimeFormatter 的格式，必须和 datetime-interval-unit 保持一致。例如：yyyyMM	
datetime-interval-amount(?)	int	分片键时间间隔，超过该时间间隔将进入下一分片	1
datetime-interval-unit(?)	String	分片键时间间隔单位，必须遵循 Java ChronoUnit 的枚举值。例如：MONTHS	DAYS

复合分片算法

复合行表达式分片算法

详情请参见行表达式。

类型：COMPLEX_INLINE

属性名称	数据类型	说明	默认值
sharding-columns(?)	String	分片列名称，多个列用逗号分隔。如不配置无法则不能校验	
algorithm-expression	String	分片算法的行表达式	
allow-range-query-with-inline-sharding(?)	boolean	是否允许范围查询。注意：范围查询会无视分片策略，进行全路由	false

Hint 分片算法

Hint 行表达式分片算法

详情请参见行表达式。

类型：HINT_INLINE

属性名称	数据类型	说明	默认值
algorithm-expression(?)	String	分片算法的行表达式	\${value}

自定义类分片算法

通过配置分片策略类型和算法类名，实现自定义扩展。

类型：CLASS_BASED

可配置属性：

属性名称	数据类型	说明
strategy	String	分片策略类型，支持 STANDARD、COMPLEX 或 HINT（不区分大小写）
algorithmClassName	String	分片算法全限定名

12.8.5 分布式序列算法

雪花算法

类型：SNOWFLAKE

可配置属性：

属性名称	数据类型	说明	默认值
max-vibration-offset (?)	int	最大抖动上限值，范围 [0, 4096)。注：若使用此算法生成值作分片值，建议配置此属性。此算法在不同毫秒内所生成的 key 取模 2^n (2^n 一般为分库或分表数) 之后结果总为 0 或 1。为防止上述分片问题，建议将此属性值配置为 (2^n)-1	1
max-tolerance-time-difference-milliseconds (?)	long	最大容忍时钟回退时间，单位：毫秒	10 毫秒

UUID

类型：UUID

可配置属性：无

12.8.6 读写分离算法

轮询算法

类型：ROUND_ROBIN

事务内，读请求根据 transaction-read-query-strategy 属性的配置进行路由。事务外，采用轮询策略路由到 replica。

可配置属性：

属性名称	数据类型	说明
transaction-read-query-strategy	String	事务内读请求路由策略，可选值：FIXED_PRIMARY(路由到 primary)、FIXED_REPLICA(根据轮询策略选择一个固定的 replica)、DYNAMIC_REPLICA(根据轮询策略路由到不同的 replica)，默认值：FIXED_PRIMARY。

随机访问算法

类型：RANDOM

事务内，读请求根据 transaction-read-query-strategy 属性的配置进行路由。事务外，采用随机策略路由到 replica。

可配置属性：

属性名称	数据类型	说明
transaction-read-query-strategy	String	事务内读请求路由策略，可选值：FIXED_PRIMARY(路由到 primary)、FIXED_REPLICA(根据轮询策略选择一个固定的 replica)、DYNAMIC_REPLICA(根据轮询策略路由到不同的 replica)，默认值：FIXED_PRIMARY。

延迟访问算法

类型：SPHERE_EX_DELAY_REPLICA

可配置属性：无

权重访问算法

类型：WEIGHT

可配置属性：

使用中的读库都必须配置权重

属性名称	数据类型	说明
{replica-name}	double	属性名使用读库名称，参数填写读库对应的权重值。权重参数范围最小值 > 0，合计 <= Double.MAX_VALUE。
transaction-read-query-strategy	String	事务内读请求路由策略，可选值：FIXED_PRIMARY(路由到 primary)、FIXED_REPLICA(根据权重策略选择一个固定的 replica)、DYNAMIC_REPLICA(根据权重策略路由到不同的 replica)，默认值：FIXED_PRIMARY。

操作步骤

1. 使用读写分离时，在 loadBalancers 属性下配置对应的负载均衡算法即可；

配置示例

```
rules:
- !READWRITE_SPLITTING
  dataSources:
    readwrite_ds:
      staticStrategy:
        writeDataSourceName: write_ds
        readDataSourceNames:
          - read_ds_0
          - read_ds_1
        loadBalancerName: random
  loadBalancers:
    random:
      type: RANDOM
      props:
        transaction-read-query-strategy: FIXED_PRIMARY
```

12.8.7 加密算法

MD5 加密算法

类型：MD5

可配置属性：

名称	数据类型	说明
salt	String	盐值（可选）

AES 加密算法

类型：AES

可配置属性：

名称	数据类型	说明
aes-key-value	String	AES 使用的 KEY

RC4 加密算法

类型：RC4

可配置属性：

名称	数据类型	说明
rc4-key-value	String	RC4 使用的 KEY

SM3 加密算法

类型：SM3

可配置属性：

名称	数据类型	说明
sm3-salt	String	SM3 使用的 SALT（空或 8 Bytes）

SM4 加密算法

类型：SM4

可配置属性：

名称	数据类型	说明
sm4-key	String	SM4 使用的 KEY（16 Bytes）
sm4-mode	String	SM4 使用的 MODE（CBC 或 ECB）
sm4-iv	String	SM4 使用的 IV（MODE 为 CBC 时需指定，16 Bytes）
sm4-padding	String	SM4 使用的 PADDING（PKCS5Padding 或 PKCS7Padding，暂不支持 NoPadding）

SphereEx:AES 加密算法

类型：SphereEx:AES

可配置属性：

名称	数据类型	说明
key-manager	String	密钥管理算法的名称

SphereEx:RC4 加密算法

类型：SphereEx:RC4

可配置属性：

名称	数据类型	说明
key-manager	String	密钥管理算法的名称

SphereEx:SM3 加密算法

类型：SphereEx:SM3

可配置属性：

名称	数据类型	说明
key-manager	String	密钥管理算法的名称

SphereEx:SM4 加密算法

类型：SphereEx:SM4
可配置属性：

名称	数据类型	说明
key-manager	String	密钥管理算法的名称

模糊加密算法

单字符按位变换加密算法

说明：该算法采用单字符按位变换加密方式，支持密钥云端管理。
类型：SphereEx:CHAR_TRANSFORM_LIKE
可配置属性：

名称	数据类型	说明
key-manager	String	密钥管理算法的名称

密钥管理

商业加密算法（SphereEx:AES、SphereEx:RC4、SphereEx:SM3、SphereEx:SM4）可以使用密钥管理功能统一管理密钥配置，支持本地密钥管理和云端密钥管理功能。

本地密钥管理

类型：LOCAL
可配置属性：

名称	数据类型	说明
aes-key-value	String	SphereEx:AES 使用的 KEY
rc4-key-value	String	SphereEx:RC4 使用的 KEY
sm3-salt	String	SphereEx:SM3 使用的 SALT（空或 8 Bytes）
sm4-key	String	SphereEx:SM4 使用的 KEY（16 Bytes）
sm4-mode	String	SphereEx:SM4 使用的 MODE（CBC 或 ECB）
sm4-iv	String	SphereEx:SM4 使用的 IV（MODE 为 CBC 时需指定，16 Bytes）
sm4-padding	String	SphereEx:SM4 使用的 PADDING（PKCS5Padding 或 PKCS7Padding, 暂不支持 NoPadding）

AWS 云端密钥管理

类型：SphereEx:AWS_KMS
可配置属性：

名称	数据类型	说明
access-key	String	aws access key 配置
secret-key	String	aws secret key 配置
aws-region	String	aws region 配置
secret-name	String	aws secret name 配置

12.8.8 影子算法

背景信息

影子库功能对执行的 SQL 语句进行影子判定。影子判定支持两种类型算法，用户可根据实际业务需求选择一种或者组合使用。

参数解释

列影子算法

列值匹配算法

类型：VALUE_MATCH

属性名称	数据类型	说明
column	String	影子列
operation	String	SQL 操作类型 (INSERT, UPDATE, DELETE, SELECT)
value	String	影子列匹配的值

列正则表达式匹配算法

类型：REGEX_MATCH

属性名称	数据类型	说明
column	String	匹配列
operation	String	SQL 操作类型 (INSERT, UPDATE, DELETE, SELECT)
regex	String	影子列匹配正则表达式

Hint 影子算法

SQL HINT 影子算法

类型：SQL_HINT

```
/* SHARDINGSPHERE_HINT: SHADOW=true */
```

配置示例

■ Java API

```
public final class ShadowConfiguration {
    // ...

    private AlgorithmConfiguration createShadowAlgorithmConfiguration() {
        Properties userIdInsertProps = new Properties();
        userIdInsertProps.setProperty("operation", "insert");
        userIdInsertProps.setProperty("column", "user_id");
        userIdInsertProps.setProperty("value", "1");
        return new AlgorithmConfiguration("VALUE_MATCH", userIdInsertProps);
    }
}
```

```
// ...  
}
```

■ YAML:

```
shadowAlgorithms:  
  user-id-insert-algorithm:  
    type: VALUE_MATCH  
    props:  
      column: user_id  
      operation: insert  
      value: 1
```

12.8.9 归档算法

归档算法

按时间归档算法

类型: SphereEx:TIME_PARTITION

可配置属性:

属性名称	数据类型	说明
interval-unit	String	分区时间间隔单位, 支持 YEARS, MONTHS, DAYS
interval-amount	int	分区时间间隔数量
retain-amount	int	暂时保留的分区数量
pre-allocate-amount	int	预分配的分区数量

按时间归档算法

类型: SphereEx:TIMED_TASK

可配置属性:

属性名称	数据类型	说明
schedule-cron-expr	String	定时任务表达式示例: "0 15 10 ? * *"

12.8.10 数据库发现算法

数据库发现算法

MySQL MGR 数据库发现算法

类型: MySQL.MGR

可配置属性:

属性名称	数据类型	说明
group-name	String	MySQL MGR 的组名称

说明:

1. 由于需要访问 MySQL information_schema、performance_schema 系统库, 查询 MGR 的环境以及实例的主从关系。

```
GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'username'@'localhost';
```

2. 由于需要动态发现主从拓扑关系，发现 MGR 中的主节点需要根据存储单元的 JDBC URL 种的 IP 和 PORT 进行匹配，所以建议 my.cnf 文件中的 report_host 和 report_port 参数与 JDBC URL 一致。

```
report_host = mysql_4
report_port = 3306
```

MySQL 主从数据库发现算法

类型: MySQL.NORMAL_REPLICATION

可配置属性:

属性名称	数据类型	说明
de lay-milliseconds-threshold	int	从库延迟阈值，从库大于等于该阈值时实现自动禁用
min-enabled-replicas	int	最小禁用数量，最少保留 n 个从库

说明:
使用此算法不需要特殊的权限，数据库用户具有 SELECT 权限即可

openGauss 主从数据库发现算法

类型: openGauss.NORMAL_REPLICATION

可配置属性: 无

说明:
使用此算法不需要特殊的权限，数据库用户具有 SELECT 权限即可

PostgreSQL 主从数据库发现算法

类型: SphereEx:PostgreSQL.NORMAL_REPLICATION

可配置属性: 无

说明:
使用此算法不需要特殊的权限，数据库用户具有 SELECT 权限即可

GaussDB_for_MySQL 主从数据库发现算法

类型: SphereEx:GaussDB_for_MySQL.NORMAL_REPLICATION

可配置属性: 无

说明:
使用此算法不需要特殊的权限，数据库用户具有 SELECT 权限即可

Aurora_for_MySQL 主从数据库发现算法

类型: SphereEx:Aurora_for_MySQL.NORMAL_REPLICATION

可配置属性: 无

说明:

使用此算法不需要特殊的权限，数据库用户具有 SELECT 权限即可

12.8.11 分片审计算法

分片审计算法

背景信息

分片审计功能是针对数据库分片场景下对执行的 SQL 语句进行审计操作。分片审计既可以进行拦截操作，拦截系统配置的非法 SQL 语句，也可以是对 SQL 语句进行统计操作。

参数解释

DML_SHARDING_CONDITIONS 算法

类型: DML_SHARDING_CONDITIONS

操作步骤

1. 配置数据分片规则时设置分配审计生成策略

配置示例

■ DML_SHARDING_CONDITIONS

```
auditors:
  sharding_key_required_auditor:
    type: DML_SHARDING_CONDITIONS
```

12.8.12 数据脱敏算法

背景信息

脱敏算法是脱敏功能使用的算法，DBPlusEngine 内置了多种算法，可以让用户方便使用。

参数解释

哈希脱敏算法

MD5 脱敏算法

类型：MD5

可配置属性：

名称	数据类型	说明
salt	String	盐值（可选）

遮盖脱敏算法

保留前 N 后 M 脱敏算法

类型：KEEP_FIRST_N_LAST_M

可配置属性：

名称	数据类型	说明
first-n	int	前 n 位
last-m	int	后 n 位
replace-char	String	替换字符

保留自 X 至 Y 脱敏算法

类型：KEEP_FROM_X_TO_Y

可配置属性：

名称	数据类型	说明
from-x	int	起始位置（从 0 开始）
to-y	int	结束位置（从 0 开始）
replace-char	String	替换字符

遮盖前 N 后 M 脱敏算法

类型：MASK_FIRST_N_LAST_M

可配置属性：

名称	数据类型	说明
first-n	int	前 n 位
last-m	int	后 n 位
replace-char	String	替换字符

遮盖自 X 至 Y 脱敏算法

类型：MASK_FROM_X_TO_Y

可配置属性：

名称	数据类型	说明
from-x	int	起始位置（从 0 开始）
to-y	int	结束位置（从 0 开始）
replace-char	String	替换字符

特殊字符前遮盖脱敏算法

类型：MASK_BEFORE_SPECIAL_CHARS

可配置属性：

名称	数据类型	说明
special-chars	String	特殊字符（首次出现）
replace-char	String	替换字符

特殊字符后遮盖脱敏算法

类型：MASK_AFTER_SPECIAL_CHARS

可配置属性：

名称	数据类型	说明
special-chars	String	特殊字符（首次出现）
replace-char	String	替换字符

替换脱敏算法

身份证随机替换脱敏算法

类型：PERSONAL_IDENTITY_NUMBER_RANDOM_REPLACE

可配置属性：

名称	数据类型	说明
alpha-two-country-area-code	String	两位字母国家/地区编码（可选，默认：CN）

军官证随机替换脱敏算法

类型：MILITARY_IDENTITY_NUMBER_RANDOM_REPLACE

可配置属性：

名称	数据类型	说明
type-codes	String	军官证种类编码（以英文逗号分隔，例如：军,人,士,文,职）

手机号随机替换脱敏算法

类型：TELEPHONE_RANDOM_REPLACE

可配置属性：

名称	数据类型	说明
network-numbers	String	网号（以英文逗号分隔）

座机号码随机替换

类型：LANDLINE_NUMBER_RANDOM_REPLACE

可配置属性：

名称	数据类型	说明
landline-numbers	String	座机号码（以英文逗号分隔）

操作步骤

- 1. 在脱敏规则中配置脱敏算法；
- 2. 为脱敏算法指定脱敏算法类型。

配置示例

```
rules:
- !MASK
tables:
  t_user:
    columns:
      password:
        maskAlgorithm: md5_mask
      email:
        maskAlgorithm: mask_before_special_chars_mask
      telephone:
        maskAlgorithm: keep_first_n_last_m_mask

maskAlgorithms:
md5_mask:
  type: MD5
mask_before_special_chars_mask:
  type: MASK_BEFORE_SPECIAL_CHARS
  props:
    special-chars: '@'
    replace-char: '*'
keep_first_n_last_m_mask:
  type: KEEP_FIRST_N_LAST_M
  props:
    first-n: 3
    last-m: 4
    replace-char: '*'
```

12.9 错误码

SQL 错误码以标准的 SQL State, Vendor Code 和详细错误信息提供, 在 SQL 执行错误时返回给客户端。

12.9.1 内核异常

元数据

SQL State	Vendor Code	错误信息
42000	10000	Resource does not exist
8000	10001	The URL %s is not recognized, please refer to the pattern %s
42000	10002	Can not support 3-tier structure for actual data node %s with JDBC %s
HY004	10003	Invalid format for actual data node %s
42000	10004	Unsupported SQL node conversion for SQL statement %s
42000	10010	Rule does not exist
42S02	10020	Schema %s does not exist
42S02	10021	Single table %s does not exist
HY000	10022	Can not load table with database name %s and data source name %s
0A000	11030	Can not drop schema %s because of contains tables

数据

SQL State	Vendor Code	错误信息
HY004	11000	Invalid value %s
HY004	11001	Unsupported conversion data type %s for value %s
HY004	11010	Unsupported conversion stream charset %s

语法

SQL State	Vendor Code	错误信息
42000	12000	You have an error in your SQL syntax: %s
42000	12001	SQL String can not be NULL or empty
42000	12002	Could not support variable %s
42S02	12003	Can not find column label %s
HV008	12004	Column index %d is out of range
0A000	12010	DROP TABLE ... CASCADE is not supported

连接

SQL State	Vendor Code	错误信息
08000	13000	Can not register driver, reason is: %s
01000	13010	Circuit break open, the request has been ignored
08000	13020	Can not get %d connections one time, partition succeed connection(%d) have released. Please consider increasing the maxPoolSize of the data sources or decreasing the max-connections-size-per-query in properties
08000	13030	Connection has been closed
08000	13031	Result set has been closed
HY000	13090	Load datetime from database failed, reason: %s

事务

SQL State	Vendor Code	错误信息
25000	14000	Switch transaction type failed, please terminate the current transaction
25000	14100	JDBC does not support operations across multiple logical databases in transaction
25000	14200	Can not start new XA transaction in a active transaction
25000	14201	Failed to create %s XA data source

锁

SQL State	Vendor Code	错误信息
HY000	15000	The table %s of schema %s is locked
HY000	15001	The table %s of schema %s lock wait timeout of %s milliseconds exceeded

审计

SQL State	Vendor Code	错误信息
44000	16000	SQL check failed, error message: %s

集群

SQL State	Vendor Code	错误信息
HY000	17000	Work ID assigned failed, which can not exceed 1024
HY000	17001	Can not find %s file for datetime initialize
HY000	17002	File access failed, reason is: %s

迁移

SQL State	Vendor Code	错误信息
44000	18001	Created process configuration already existed
44000	18002	Altered process configuration does not exist
HY000	18020	Failed to get DDL for table %s
42S01	18030	Duplicate resource names %s
42S02	18031	Resource names %s do not exist
HY000	18050	Before data record is %s, after data record is %s
08000	18051	Data check table %s failed
0A000	18052	Unsupported pipeline database type %s
0A000	18053	Unsupported CRC32 data consistency calculate algorithm with database type %s
HY000	18080	Can not find pipeline job %s
HY000	18081	Job has already started
HY000	18082	Sharding count of job %s is 0
HY000	18083	Can not split range for table %s, reason: %s
HY000	18084	Can not split by unique key %s for table %s, reason is: %s
HY000	18085	Target table %s is not empty
01007	18086	Source data source lacks %s privilege(s)
HY000	18087	Source data source required %s = %s, now is %s
HY000	18088	User %s does exist
08000	18089	Check privileges failed on source data source, reason is: %s
08000	18090	Data sources can not connect, reason is: %s
HY000	18091	Importer job write data failed
08000	18092	Get binlog position failed by job %s, reason is: %s
HY000	18093	Can not poll event because of binlog sync channel already closed
HY000	18094	Task %s execute failed

12.9.2 功能异常

数据分片

SQL State	Vendor Code	错误信息
44000	20000	Can not find table rule with logic tables %s
44000	20001	Can not get uniformed table structure for logic table %s, it has different meta data of actual ta
42S02	20002	Can not find data source in sharding rule, invalid actual data node %s
44000	20003	Data nodes must be configured for sharding table %s
44000	20004	Actual table %s.%s is not in table rule configuration
44000	20005	Can not find binding actual table, data source is %s, logic table is %s, other actual table is %s
44000	20006	Actual tables %s are in use
42S01	20007	Index %s already exists
42S02	20008	Index %s does not exist
42S01	20009	View name has to bind to %s tables
44000	20020	Sharding value can' t be null in insert statement
HY004	20021	Found different types for sharding value %s
HY004	20022	Invalid %s, datetime pattern should be %s, value is %s
0A000	20040	Can not support operation %s with sharding table %s
44000	20041	Can not update sharding value for table %s
0A000	20042	The CREATE VIEW statement contains unsupported query statement
44000	20043	PREPARE statement can not support sharding tables route to same data sources
44000	20044	The table inserted and the table selected must be the same or bind tables
0A000	20045	Can not support DML operation with multiple tables %s
42000	20046	%s ... LIMIT can not support route to multiple data nodes
44000	20047	Can not find actual data source intersection for logic tables %s
42000	20048	INSERT INTO ... SELECT can not support applying key generator with absent generate key colu
0A000	20049	Alter view rename .. to .. statement should have same config for %s and %s
HY000	20060	%s %s can not route correctly for %s %s
42S02	20061	Can not get route result, please check your sharding rule configuration
34000	20062	Can not get cursor name from fetch statement
HY000	20080	Sharding algorithm class %s should be implement %s
HY000	20081	Routed target %s does not exist, available targets are %s
44000	20082	Inline sharding algorithms expression %s and sharding column %s do not match
44000	20090	Can not find strategy for generate keys with table %s

读写分离

SQL State	Vendor Code	错误信息
HY004	20280	Invalid read database weight %s

数据库高可用

SQL State	Vendor Code	错误信息
HY000	20380	MGR plugin is not active in database %s
44000	20381	MGR is not in single primary mode in database %s
44000	20382	%s is not in MGR replication group member in database %s
44000	20383	Group name in MGR is not same with configured one %s in database %s

SQL 方言转换

SQL State	Vendor Code	错误信息
42000	20440	Can not support database %s in SQL translation
42000	20441	Translation error, SQL is: %s

流量治理

SQL State	Vendor Code	错误信息
42S02	20500	Can not get traffic execution unit

数据加密

SQL State	Vendor Code	错误信息
44000	20700	Can not find logic encrypt column by %s
44000	20701	Fail to find encrypt column %s from table %s
44000	20702	Altered column %s must use same encrypt algorithm with previous column %s in table %s
42000	20740	Insert value of index %s can not support for encrypt
0A000	20741	The SQL clause %s is unsupported in encrypt rule
HY004	20780	Encrypt algorithm %s initialization failed, reason is: %s

影子库

SQL State	Vendor Code	错误信息
HY004	20820	Shadow column %s of table %s does not support %s type
42000	20840	Insert value of index %s can not support for shadow

12.9.3 其他异常

SQL State	Vendor Code	错误信息
HY004	30000	Unknown exception: %s
0A000	30001	Unsupported operation: %s
0A000	30002	Unsupported command: %s

12.10 数据库协议支持

数据库协议	APP to SphereEx-DB Plus Engine-Proxy	App to SphereEx-DB Plus Engine-Driver	SphereEx-DB Plus Engine-Proxy to DB	SphereEx-DB Plus Engine-Driver to DB
MySQL	支持	支持	支持	支持
PostgreSQL	支持	支持	支持	支持
OpenGauss	支持	支持	支持	支持
Oracle	不支持	支持	不支持	支持
SQL Server	不支持	支持	不支持	支持
人大金仓	支持	支持	支持	支持
达梦	不支持（适配进行中）	支持	不支持（适配进行中）	支持
南大通用	不支持	支持	不支持	支持
神州通用	不支持	支持	不支持	支持

12.11 数据库支持列表

12.11.1 数据库支持列表

数据库分类	数据库	版本	SphereEx-DBPlusEngine-Proxy	SphereEx-DBPlusEngine-Driver
开源	MySQL	5.7 及以上	支持	支持
	PostgreSQL	9.6 及以上	支持	支持
开源-国产	OpenGauss	2.1.0 及以上	支持	支持
商业	Oracle	无版本限制	不支持	支持
	SQL Server	无版本限制	不支持	支持
商业-国产	人大金仓-KingbaseES	无版本限制	支持	支持
	云和恩墨-MogDB	无版本限制	支持	支持
	达梦-DM8	无版本限制	不支持（适配进行中）	支持
	南大通用-GBase 8c	无版本限制	不支持	支持
	神州通用-神通数据库 (openGauss 版) 无版本限制		不支持	支持

13.1 商业化服务条款

本服务条款是北京思斐软件技术有限公司（以下简称“SphereEx”）与您关于 SphereEx 商业版服务所涉及事项订立的有效合约。通过合同盖章方式使本条款生效，即表示您与 SphereEx 达成协议并同意接受本服务条款的全部内容。

13.1.1 1. 服务内容

1.1. 本条款中“服务”指：SphereEx 向您提供商业化产品服务以及相关的技术及网络支持服务（不包含免费试用功能）。

13.1.2 2. 服务费用

2.1. SphereEx 产品的价格体系（价格、计费模式等）将以双方签订合同的价格为准。

2.2. SphereEx 保留您在未按照约定支付全部费用之前不继续向您提供有偿服务和技术支持，或者终止服务和技术支持的权利，同时，SphereEx 有权要求您支付终止服务前您尚未支付的服务费用。

13.1.3 3. 权利义务

3.1. 您的权利、义务

3.1.1. 此条款的内容可能会存在变更，如本服务条款的内容更新，SphereEx 将提前 15 天在条款所发布渠道向您提示修改内容。如您不同意 SphereEx 对本服务条款相关条款所做的修改，您有权停止使用 SphereEx 的服务，如您继续使用 SphereEx 服务，则视为您接受此服务条款所做的修改。

3.1.2. 在接受本服务条款之前，请您仔细阅读本服务条款的全部内容。如果您对本服务条款有疑问，请通过 SphereEx 相关业务部门进行询问。

3.1.3. 您承诺：

3.1.3.1. 不得修改、翻译、或转让 SphereEx 提供的服务，也不得反编译或试图以其他方式发现、或泄漏 SphereEx 提供的软件的源代码（SphereEx 提供许可除外）；

3.1.3.2. 不得利用 SphereEx 提供的服务散发包含反动、色情等有害信息；

3.1.3.3. 不得利用 SphereEx 提供的服务发布违反国家规定和宗教政策的新闻信息；

3.1.3.4. 不得利用 SphereEx 提供的服务发布非法互联网活动或者妨碍互联网运行安全的信息；

3.1.3.5. 不得利用 SphereEx 提供的服务发布侵害他人合法权益的信息和有害于社会治安、公共道德的信息；

3.1.3.6. 不得利用 SphereEx 提供的服务进行任何破坏或试图破坏网络安全的行为（如病毒、木马、恶意代码等）；

3.1.3.7. 不得利用 SphereEx 提供的服务进行其他违反法律法规或国家政策的内容。

3.1.3.8. 如 SphereEx 发现您违反上述条款，有权根据实际情况采取相应的处理措施，包括但不限于立即终止服务、采取相应的法律措施等。

3.2. SphereEx 的权利、义务

3.2.1. SphereEx 应按照服务条款提供服务。

3.2.2. 服务期限内，SphereEx 将为您提供下述服务：

3.2.2.1. 为付费用户提供商业化产品、技术支持服务、故障处理服务、技术咨询服务、技术培训服务。

3.2.2.2. 为付费用户提供多种有效的联系方式，保证用户能够及时联系到故障联系人。

3.2.2.3. SphereEx 指定有效的故障反馈和处理流程，高效的处理用户故障。

3.2.3. SphereEx 应严格遵守保密义务。

13.1.4 4. 知识产权

4.1. SphereEx 向您提供的任何服务、资料、或技术支持等的知识产权均属于 SphereEx。

4.2 未经过 SphereEx 明确同意之外，您无权复制、转让、或提供他人使用上述资源，否则应承担相应的责任。

13.1.5 5. 保密条款

5.1. 本服务条款中的任何一方应对保密资料予以保密，(包括但不限于产品资料，产品价格，产品技术，特殊算法等)。

5.2. 除非国家机关依法强制要求领域外，签订保密条款的一方不得对外披露。

13.1.6 6. 服务的开通、使用和终止

6.1. 您自签订合同之日起即可使用 SphereEx 的服务。

6.2. 您在使用 SphereEx 的服务的过程中有任何疑问，均可联系 SphereEx。

6.3. 发生下列情形，服务将被终止：

6.3.1. 双方经过协商一致决定提前终止的。

6.3.2. 由于您严重违反本服务条款（包括但不限于 a. 您严重违反本服务条款中所做的承诺 b. 您严重违反法律规定等），SphereEx 有权按本服务条款的相关规定单方面终止服务。

6.3.3. 如您未按照合同约定履行付款义务，SphereEx 将暂停或终止服务。

13.1.7 7. 违约责任

7.1. 本服务条款任何一方违约均须依法承担违约责任。

7.2. 违约责任以双方实际签订的盖章合同为准。## 8. 不可抗力 8.1. 因不可抗力事件，使得无法履行本服务条款，遭受不可抗力的一方不承担责任，该方应立即通知另一方其不能履行或延迟履行合同义务的原因，并应在不可抗力事件发生的 15 个自然日内向另一方提供有关发生不可抗力的证明文件，按不可抗力事件对本合同的影响程度，双方协商是否终止本合同，或部分免除本合同的部分义务。

8.2. 不可抗力指的是发生战争、洪水、台风、地震或其他无法预见、无法避免、人力不能控制的不可抗力事件。## 9. 法律适用及争议解决 9.1. 本条款的解释、效力、履行及其他事宜，均受中华人民共和国法律（不包括香港、台湾、澳门法律）管辖。

9.2. 在执行本服务条款过程中如发生争议，双方应及时协商解决。

9.3 协商不成时，任何一方可直接向所在地区的人民法院提起诉讼。

13.1.8 10. 条款效力

10.1 如果本条款中所列的条款和规定与实际签订合同及合同的各个附件（包括一般条款和条件）所列的条款和约定有任何冲突，则应以实际签订合同的条款为准。

10.2 双方当事人协商一致后，可以以书面方式对本条款进行修改和补充，该书面修改和补充作为本条款的有效组成部分。

10.3 如本条款中有任何相关条款被司法机关视为无效、被撤销、不合法或不可执行，不影响本条款中其他条款的效力。

13.2 服务等级协议 SLA

本服务等级协议（Service Level Agreement，简称“SLA”）说明了 SphereEx 向客户提供商业版产品的服务等级方案。

13.2.1 服务标准（SLA）

故障分级

根据服务请求的类型、严重程度，我们将服务请求划分为三个级别，以下分别称作一级故障、二级故障和三级故障。具体划分如下：

1) 一级故障

产品发生处理能力、可用性完全或部分丧失，导致紧急问题甚至宕机瘫痪，或对最终用户的业务使用有严重影响，重要系统主要功能模块丧失功能达半小时以上，需要立即采取措施进行处理的问题。

2) 二级故障

产品发生处理能力、可用性降低或恶化，严重影响和限制了应用系统的使用与维护，或对部分最终用户的业务使用有较大影响的。

3) 三级故障

产品的可用性受损，但大部分业务操作仍可正常进行。

故障响应时限

根据故障等级，响应时限如下：

1) 一级故障：

7 * 24 小时值班电话，15 分钟内微信等远程方式响应，并立即开展远程技术支持，如无法解决，同城 4 小时内派工程师到现场。

2) 二级故障：

7 * 24 小时值班电话，30 分钟内微信等远程方式响应，1 小时内开展远程技术支持，如 4 个小时内无法解决，在下 1 个工作日内派工程师到现场。

3) 三级故障：

5 * 8 小时热线电话，30 分钟内电话或微信等远程方式响应，1 小时内开展远程技术支持，如果 1 个工作日内无法解决，在下 1 个工作日内派工程师到现场。

13.2.2 应急流程

紧急故障响应流程

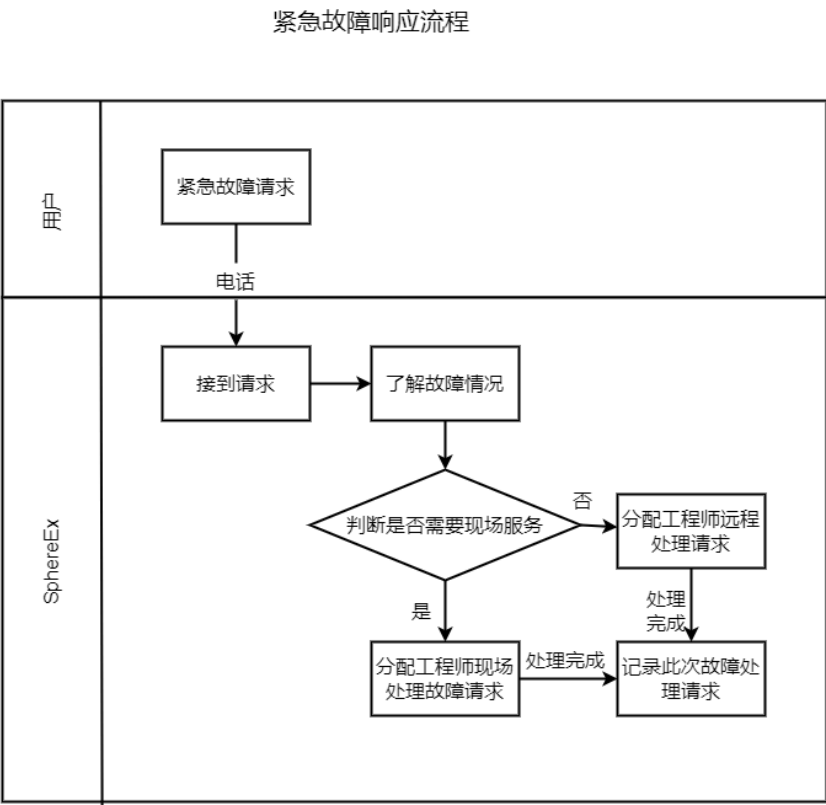


Fig. 1: 紧急故障响应流程图

13.2.3 服务类别

工时类（人天 / 人时）

- 1) 8 小时/人天（9:00 ~ 12:00 + 13:00 ~ 18:00），大于 4 小时且不足 8 小时的按 1 人天计算，不足 4 小时的按 0.5 人天计算；
- 2) 大于 0.5 小时且不足 1 小时的按 1 小时计算，不足 0.5 小时的按 0.5 小时计算；
- 3) 按劳动法规定，在 9:00 ~ 18:00 之外的时间以及周六、日按双倍计算，节假日、公休日按三倍计算；

包年类

- 1) 以合同签署的年限时间为准。

13.2.4 其他

1) 在接受本 SLA 条款之前, 请您仔细阅读本条款的全部内容。如果您对本服务条款有疑问, 请联系 SphereEx 相关业务部门进行询问。

2) SphereEx 有权对本 SLA 条款作出修改。如本 SLA 条款有任何修改, SphereEx 提前 15 天以网站公示或其他方式通知您。如您不同意修改后的 SLA, 您有权停止使用 SphereEx 提供的产品服务, 如您继续使用 SphereEx 提供的产品服务, 则视为您接受修改后的 SLA。

13.3 用户许可协议

本条款和条件(下称“条款”)适用于所有使用由“北京思斐软件技术有限公司”(下称“思斐”)所提供产品(参见如下定义)的用户。

请仔细阅读如下条款:

若您(下称“您”或“用户”)或您代表某公司或者其他机构或组织使用任何产品时, 您特此陈述您本人或作为该公司或该等其他机构的员工或代理, 您本人同意且有权代表该公司或该等其他机构同意接受条款项下所要求的全部条款和条件(以下统称为“本协议项”)。

若使用任何产品, 您知晓并同意:

- (A) 您已阅读本协议中所有的条款和条件;
- (B) 您已理解本协议中所有的条款和条件;
- (C) 您已同意本协议中所有条款和条件对您具有法律约束力。

如果您不同意本协议所述条款和条件中的任意内容, 则不能获得安装、使用产品的权利。

本协议的“生效日期”是指您第一次下载或通过其它各类形式得到任何产品的日期。

1. 产品, 指本协议项下任何思斐的产品和软件, 包括但不限于: SphereEx-DBPlusEngine、SphereEx-Console、SphereEx-Boot 等任何试用软件、任何软件或与其相关的升级、更新、故障修复或修改版本(统称“更新软件”)。无论本协议是否另有规定, (1) 用户不得自行复制或使用任何额外更新软件的副本, 除非用户在使用或获取该更新软件时, 已拥有对原始软件有效的许可并已支付更新软件或软件副本的相关费用; (2) 更新软件的使用系经许可的资源, 即作为该等资源提供方的用户为最初的终端购买者或持有有效许可使用该正在被更新产品的其他用户; (3) 制作和使用额外的副本仅限于必要的备份目的。
2. 全部协议, 本协议包括本条款以及任何思斐官方网站 (<http://www.sphere-ex.com>) 展示或者网页链接所附或引用的全部条款。本协议是双方就相关事项达成的完整协议, 取代思斐与用户之间就本条款相关事项所达成的其他任何协议, 无论是口头的还是书面的。
3. 使用许可, 思斐授予用户非排他性的、不可转让的、非可再授权的, 可撤回的和有限的许可进行访问和使用第 1 条所定义的产品, 该访问和使用许可仅限于用户内部使用之目的。通过电子下载或其他未经许可的来源获得产品的用户均应受限于本协议的内容。
4. 许可限制, 除非本协议另有明文规定, 否则用户将不被允许:
 - (a) 修改、翻译或制造产品的衍生作品;
 - (b) 反向编译、反向工程、破解产品的任何部分或试图发现有关产品的任何源代码、基本理念或运算方法;
 - (c) 销售、分派、再授权、出租、出借、出质、提供或另行翻译全部或部分产品;
 - (d) 制造、获取非法制造的、再版或复制产品;
 - (e) 删除或更改与产品相关联的任何商标、标志、版权或其他专有标志;
 - (f) 不得在没有思斐明确书面授权的情况下, 使用或许可他人使用产品为第三方提供服务, 无论是在产品服务过程中使用或采用分时的方式;
 - (g) 删除本产品中的任何版权声明或提示以及任何其他信息、内容;
 - (h) 利用本产品储存、发表、传播违反国家法律、法规以及国家政策规定的内容;
 - (i) 利用本产品储存、发表、传播侵害他人知识产权、商业秘密等合法权益的内容;

(j) 进行危害计算机网络安全的行为；

(k) 引起或许可其他任何其他方进行上述任何一种禁止行为。

5. 所有权，本产品由思斐开发或者由思斐获得软件权利人的许可。思斐和用户在本协议项下的许可需明确，思斐拥有以下各项的全部权利、所有权和相关利益：

(a) 产品（包括但不限于，任何更新软件、修订版本或其衍生作品）；

(b) 在思斐根据本协议提供任何服务的过程中或作为其提供服务的结果，由思斐发现、产生或发展出来的所有的概念、发明、发现、改进、信息、创意作品等；

(c) 前述各项所含的任何知识产权权利以及与公司产品相关的所有信息内容。在本协议项下，“知识产权”是指在任何管辖区域经申请和注册获得认可和保护的全部专利、版权、道德权利、商标、商业秘密和任何其他形式的权利。思斐与用户同意，在受限于本协议全部条款和条件的前提下，用户拥有使用产品而产生的数据的权利、所有权等相关利益。在此协议中无任何默示许可，思斐保留本协议项下未明确授权的全部权利。除非本协议明确约定，思斐在本协议下未授予用户任何许可权利，无论是通过暗示、默许或其他方式。

6. 保密，保密信息是指，无论是在本协议生效前或生效后，由思斐披露给用户的与本协议或与思斐相关的所有信息（无论是以口头的、书面、或其他有形、无形的形式），而鉴于该等信息披露给客户时相关事实和实际情况，客户知道或应当知道该等信息为思斐的保密信息。保密信息包括但不限于，商业计划的内容、产品、发明、设计图纸、财务计划、计算机程序、用户信息、战略和其他类似信息。在本协议期限内，除非获得明确许可，用户需保证保密信息的秘密性，并确保不会使用上述保密信息。用户将采用与保护其自身保密信息的同等谨慎程度（不论在何种情况下均不低于合理的谨慎程度）来保护思斐的保密信息，来避免使得保密信息被未经授权的使用和披露。保密信息只供用户根据本协议规定使用产品之目的而使用。此外，用户将：

(a) 除非用户为了根据本协议的规定而使用产品之目的外，不得以任何形式复制保密信息；

(b) 只向为确保用户可根据本协议使用产品而必需知道该保密信息的员工和顾问披露保密信息，前提是上述员工和顾问已签署了包含保密义务不低于本条所述内容的保密协议。保密信息不包括下列信息：

(i) 非因用户过错违反本协议导致已进入公共领域的；

(ii) 用户能合理证明其在通过思斐获得之前已知晓的；

(iii) 用户能证明没有使用或参考该保密信息而独立获得的；

(iv) 用户从其他无披露限制或无保密义务的第三方获得的。如无另行说明，由用户提供给思斐有关产品的任何建议，评论或者其他反馈（统称“反馈信息”）将构成同样成保密信息。此外，思斐有权使用、披露，复制，许可和利用上述反馈信息，而无需承担任何知识产权负担或其他任何形式的义务或限制的。根据相关法律，与本协议的履行和用户使用思斐产品相关的情况下：

(i) 思斐同意不会要求用户提供任何个人身份信息；

(ii) 用户同意不提供任何个人身份信息给思斐。

7. 免责声明，用户陈述、保证及承诺如下：

(a) 其所有员工和顾问都将遵守本协议的全部条款；

(b) 在履行本协议时将遵守全部可适用的政府部门颁发的法律、法规、规章、命令和其他要求（无论是现行有效还是之后生效的）。无论本协议是否另有规定，用户将持续对其雇员或顾问的全部作为或不作为承担责任，如同该等作为或不作为系其自身所为。产品系按照原状或现状提供给用户，不含任何形式的陈述、保证、承诺或条件。思斐及其供应商不保证任何产品将无任何故障、错误或漏洞。思斐和其供应商不为产品的如下内容提供任何陈述和保证（无论是明示或暗示，口头或书面），不论该内容是否依据法律之规定，行业惯例，交易习惯或其他原因而要求的：

(I) 保证适销性；

(II) 保证可适用于任何目的（不论思斐是否知晓、应当知晓、被建议或另行得知该目的）；

(III) 保证不侵权和拥有全部所有权。用户已明确知悉并同意产品上无任何陈述和保证。此外，鉴于进行入侵和网络攻击的新技术在不断发展，思斐并不保证产品或产品所使用的系统或网络将免于任何入侵或攻击。思斐仅保证其有权授予本合同约定的软件许可，不保证产品不侵犯第三方知识产权，也没有义务就侵权索赔或由此产生的费用向用户作出赔偿。

8. 损害赔偿，用户应赔偿、保护或使得思斐及其董事、高管、雇员、供应商、顾问、承包商和代理商（统称为“思斐受保障方”）免受所有现存或潜在的针对思斐受保障方因提起请求、诉讼或其他程序而引起的要求其赔偿损害损失、支付费用、罚款、调解、损失费用等支出（包括但不限于合理的律师费、费用、罚款、利息和垫付款），用户承担上述责任的前提是该请求、诉讼或其他程序，不论是否成功系在如下情况发生时导致、引起的，或以任何形式与下述情况相关的：

- (a) 任何对本协议的违反（包括但不限于，任何违反用户陈述和保证或约定的情况）；
- (b) 用户过失或故意产生的过错行为；或
- (c) 引起争议的数据和信息系在产品的使用过程中产生或收集的。

9. 责任限制，除了思斐存在欺诈或故意的过错行为，在任何情况下：

(A) 思斐都不会赔偿用户或任何第三方的因本协议或产品（包括用户使用或无法使用产品的情况）而遭受的任何利润损失、数据损失、使用损失、收入损失、商誉损失、任何经营活动的中断，任何其他商业损害或损失，或任何间接的、特殊的、附带的、惩戒性、惩罚性或伴随的损失，不论上述损失系因合同、侵权、严格责任或其他原因而确认的，即使思斐已被通知或因其他可能的渠道知晓上述损失发生的可能性；

(B) 思斐因本协议所需承担的全部赔偿责任不应超过已支付或将支付给思斐的全部款项总额（若有）。多项请求亦不得超过该金额限制。上述限制、排除情况及声明应在相关法律允许的最大范围内得以适用，即便任何补偿无法达到其实质目的。

10. 第三方供应商，产品可能包括由第三方供应商许可提供的软件或其他代码（下称“第三方软件”）。用户已知悉第三方供应商不对产品或其任何部分提供任何陈述和保证，不承担因产品或用户对第三方软件的使用或不能使用的情况而产生的任何责任。
11. 诊断和报告，用户了解并同意该产品包含诊断功能作为其默认配置。诊断功能用于收集有关使用环境和产品使用过程中的配置文件、节点数、软件版本、日志文档和其他信息，并将上述信息报告给思斐用于提前识别潜在的支持问题、了解用户的使用环境、并提高产品的使用性能。虽然用户可以选择更改诊断功能来禁用自动定时报告或仅用于报告服务记录，但用户需同意，每季度须至少运行一次诊断功能并将结果报告给思斐。
12. 终止，本协议期限从生效之日起，如用户违反本协议中条款则终止。无论本协议是否另有规定，在用户存在违反第 3、4 或 6 条时，或用户存在发布违法信息、严重违背社会公德、以及其他违反法律禁止性规定的行为，思斐有权立即终止本协议。本协议期满或提前终止时：

(a) 根据本协议所授予给用户的所有权利将立即终止，在此情况下用户应立即停止使用产品；

(b) 用户将及时将届时仍由其占有的所有保密信息（包括但不限于产品）提供给思斐，或根据思斐的自行审慎决定及指示，销毁该等保密信息全部副本。无论本协议是否另有规定，本条及以下条款在本协议到期或终止时仍有效：第 4、6、7、8、9、10、12、14、16 和 17 条。

13. 测试版软件，在用户使用产品用作下载和安装任何由思斐提供的公共测试版软件，该测试版软件应适用 Apache 2.0 许可或其他对公共测试版许可适用的条款和条件。

14. 第三方资源，思斐供应的产品可能包括对其他网站、内容或资源的超链接（下称“第三方资源”），且思斐此类产品的正常使用可能依赖于第三方资源的可用性。思斐无法控制任何第三方资源。用户承认并同意，思斐不就第三方资源的可用性承担任何责任，也不对该等第三方资源所涉及的或从其中获得的任何广告、产品或其他材料提供保证。用户承认并同意，思斐不应因第三方资源的可用性、或用户依赖于第三方资源所涉及的或从其中获得的任何广告、产品或其他材料的完整性、准确性及存续而可能遭受的损失或损害承担任何责任。

15. 版本升级，本产品将来是否提供升级版本将由思斐决定。如果本产品经思斐同意升级，除非升级版本有替代的用户许可协议，否则升级版本仍适用本协议。如果该软件标明为升级版本，用户必须遵守本协议。即使思斐在特殊情况下允许用户通过研究代码、修改代码进行版本升级，升级版本的所有权及相关知识产权亦属于思斐。

16. 网络安全与隐私保护，基于保护用户个人信息的基本原则，思斐将会采取合理的措施保护用户的个人信息。除法律法规规定的情形外，未经用户的许可思斐不会向第三方公开、透露用户个人信息。思斐对相关信息采用专业加密存储与传输方式，保障用户个人信息的安全。思斐将运用各种安全技术和程序建立完善的管理制度来保护你的个人信息，以免遭受未经授权的访问、使用或披露。未经用户的同意，思斐不会向思斐以外的任何公司、组织和个人披露用户的个人信息，但下列情况除外：

- (a) 用户或用户的监护人授权思斐披露的；
- (b) 有关法律要求思斐披露的；
- (c) 司法机关或行政机关基于法定程序要求思斐提供的；
- (d) 思斐为了维护自己合法权益而向乙方提起诉讼或者仲裁时；
- (e) 应用户的监护人的合法要求而提供用户的个人身份信息时。

17. 其他，本协议全部内容均在中华人民共和国境内履行，受中华人民共和国法律管辖并根据其解释（但不适用相关冲突法的法律条款）。用户与思斐同意与本协议有关的任何争议将向北京朝阳区法院提出，且不可撤销无条件的同意上述有法院对因本协议提起的全部诉讼、争议拥有排他的管辖权。一旦确定任何条款无效，非法或无法执行，思斐保留修改和解释该条款的权利。任何需要发送给用户的通知如公布在思斐的网站上则被视为已有效、合法地发送给用户。除了本合同项下应支付款项的义务外，任何一方将不对因不可抗力而导致的无法合理控制的全部或部分未能履行或延迟履

行本协议的行为负责，不可抗力包括但不限于火灾、暴风雨、洪水、地震、内乱、电信中断、电力中断或其他基础设施的中断、思斐使用的服务提供商存在问题导致服务中断或终止、罢工、故意毁坏事件、电缆被切断、病毒入侵或其他任意第三方故意或非法的行为引起的其他类似事件。在上述迟延履行情况出现时，可迟延履行协议的时间为因上述原因引起的延迟时间。本协议另有明确规定外，本协议所要求或认可的通知或通讯均需以书面形式经一方有权代表签署或授权并以直接呈递、隔夜快递，经确认的电子邮件发送，经确认的传真或邮寄挂号信、挂号邮件保留回单，预付快递费用的方式送达至以下地址或本条款所提供的其他地址。对本协议的任何修改、补充或删除或权利放弃，必须通过书面由双方适当授权的代表签署确认后方为有效。任何一方对任何权利或救济的不履行或迟延履行（部分或全部）不构成对该等权利或救济的放弃，也不影响任何其他权利或救济。本协议项下的所有权利主张和救济均可累积的且不排除本协议中包含的或法律所规定的其他任何权利或救济。对本协议中任何一项违约责任的豁免或延迟行使任何权利，并不构成对其他后续违约责任的豁免。

14
FAQ

14.1 使用 Aurora 数据库 + Aurora JDBC URL + Aurora 驱动的情况下, 未配置 proxy-frontend-database-protocol-type 参数 (即使用默认值 MySQL) 重启 Proxy 之后, 无法再次登录 Proxy, 报错: “ERROR 2013 (HY000): Lost connection to MySQL server at ‘reading initial communication packet’, system error: 2 “

回答:

1. 配置全局 props “proxy-frontend-database-protocol-type: Aurora” 参数重启生效
2. 不重启的情况下, 需要在./zkCli.sh 下执行下面的语句

```
create /governance_ds2/props  
set /governance_ds2/props 'proxy-frontend-database-protocol-type: Aurora'
```

14.2 SphereEx-DBPlusEngine-Proxy 在 AWS 上的收费标准 ?

1. AWS 的 BYOL
2. AWS 的 pay-as-you-go (仅限 global)

具体报价请联系我们

14.3 如何在 AWS 上使用 SphereEx-DBPlusEngine-Proxy AMI 镜像 ?

回答:

1. dbplusengine 应用安装在 /opt/sphereex-dbplusengine-proxy 目录下
2. dbplusengine 的 systemd 服务文件为 /usr/lib/systemd/system/dbplusengine-proxy.service

```
[Unit]  
Description=SphereEx dbplusengine Service  
Requires=network.target  
After=network.target  
[Service]  
Type=forking
```

```
LimitNOFILE=65536
ExecStart=/opt/sphereex-dbplusengine-proxy/bin/start.sh
ExecStop=/opt/sphereex-dbplusengine-proxy/bin/stop.sh
Restart=always
RestartSec=5
StartLimitInterval=0
[Install]
WantedBy=default.target
```

运维

查看

```
systemctl status dbplusengine-proxy
```

启动

```
systemctl start dbplusengine-proxy
```

重启

```
systemctl restart dbplusengine-proxy
```

停止

```
systemctl stop dbplusengine-proxy
```

配置 JVM 内存

默认的 jvm 内存参数为 “-Xmx2g -Xms2g -Xmn1g”，如果您想更改，可以按照如下部署：

1. 修改 dbplusengine-proxy.service 配置，在 Service 下添加如下配置

```
[Service]
...
Environment="JAVA_MEM_COMMON_OPTS=-Xmx512m -Xms512m -Xmn128m "
```

2. 重载服务

```
systemctl daemon-reload
```

3. 重启服务

```
systemctl restart dbplusengine-proxy
```

14.4 [Driver/Proxy] 为什么启动 Proxy 会报 Current instance version not allowed join the cluster ?

回答：

在 1.3.0 版本之后增加了集群版本校验功能，启动的计算节点时无法兼容集群版本时将无法启动。

14.5 [Proxy] MySQL 插入如下包含特殊字符的数据时，Engine-Proxy 会有报错 ERROR 1366 (HY000): Incorrect string value: '0:raw-latex: x`9F:raw-latex: x8E:raw-latex:x`81wo...' for column 'origin' at row 1

回答：

解决办法需要配置 Engine-Proxy 的连接串或 MySQL server 的字符集

1. 在 Engine-Proxy 的连接串上添加如下内容：

```
useUnicode=true&character_set_server=utf8mb4&connectionCollation=utf8mb4_unicode_ci
```

2. 在线调整 MySQL server 参数

```
set @@character_set_server='utf8mb4';  
set @@collation_server='utf8mb4_unicode_ci';
```

MySQL 的启动参数文件中添加如下两个配置项，重启后继续生效

```
character-set-server=utf8mb4  
collation-server=utf8mb4_unicode_ci
```

14.6 [Driver] 为什么配置了某个数据连接池的 spring-boot-starter (比如 druid) 和 shardingsphere-jdbc-spring-boot-starter 时，系统启动会报错？

回答：

1. 因为数据连接池的 starter (比如 druid) 可能会先加载并且其创建一个默认数据源，这将会使得 DBPlusEngine-Driver 创建数据源时发生冲突。
2. 解决办法为，去掉数据连接池的 starter 即可，DBPlusEngine-Driver 自己会创建数据连接池。

14.7 [Driver] 使用 Spring 命名空间时找不到 xsd?

回答：

Spring 命名空间使用规范并未强制要求将 xsd 文件部署至公网地址，但考虑到部分用户的需求，我们也将相关 xsd 文件部署至 DBPlusEngine 官网。

实际上 shardingsphere-jdbc-spring-namespace 的 jar 包中 META-INF:raw-latex:spring.schemas 配置了 xsd 文件的位置：META-INF\namespace\sharding.xsd 和 META-INF\namespace\replica-query.xsd，只需确保 jar 包中该文件存在即可。

14.8 [Driver] 引入 shardingsphere-transaction-xa-core 后，如何避免 spring-boot 自动加载默认的 JtaTransactionManager？

回答：

1. 需要在 spring-boot 的引导类中添加 @SpringBootApplication(exclude = JtaAutoConfiguration.class)。

14.9 [Proxy] Windows 环境下，运行 DBPlusEngine-Proxy，找不到或无法加载主类 org.apache.shardingsphere.proxy.Bootstrap，如何解决？

回答：

某些解压缩工具在解压 DBPlusEngine-Proxy 二进制包时可能将文件名截断，导致找不到某些类。

解决方案：

打开 cmd.exe 并执行下面的命令：

```
tar zxvf apache-shardingsphere-${RELEASE.VERSION}-shardingsphere-proxy-bin.tar.gz
```

14.10 [Proxy] 在使用 DBPlusEngine-Proxy 的时候，如何动态添加新的 logic schema？

回答：

使用 DBPlusEngine-Proxy 时，可以通过 DistSQL 动态的创建或移除 logic schema，语法如下：

```
CREATE (DATABASE | SCHEMA) [IF NOT EXISTS] schemaName;
```

```
DROP (DATABASE | SCHEMA) [IF EXISTS] schemaName;
```

例：

```
CREATE DATABASE sharding_db;
```

```
DROP SCHEMA sharding_db;
```

14.11 [Proxy] 在使用 DBPlusEngine-Proxy 时，怎么使用合适的工具连接到 DBPlusEngine-Proxy？

回答：

1. DBPlusEngine-Proxy 可以看做是一个 database server，所以首选支持 SQL 命令连接和操作。
2. 如果使用其他第三方数据库工具，可能由于不同工具的特定实现导致出现异常。
3. 目前已测试的第三方数据库工具如下：
 - Navicat：11.1.13、15.0.20。
 - DataGrip：2020.1、2021.1（使用 IDEA/DataGrip 时打开 introspect using JDBC metadata 选项）。
 - WorkBench：8.0.25。

14.12 [Proxy] 使用 Navicat 等第三方数据库工具连接 DBPlusEngine-Proxy 时，如果 DBPlusEngine-Proxy 没有创建 Schema 或者没有添加 Resource，连接失败？

回答：

1. 第三方数据库工具在连接 DBPlusEngine-Proxy 时会发送一些 SQL 查询元数据，当 DBPlusEngine-Proxy 没有创建 schema 或者没有添加 resource 时，DBPlusEngine-Proxy 无法执行 SQL。
2. 推荐先创建 schema 和 resource 之后再使用第三方数据库工具连接。
3. 有关 resource 的详情请参考。[相关介绍](#)

14.13 [分片] Cloud not resolve placeholder ...in string value ... 异常的解决方法？

回答：

行表达式标识符可以使用 \${...} 或 \${>{...}}，但前者与 Spring 本身的属性文件占位符冲突，因此在 Spring 环境中使用行表达式标识符建议使用 \${>{...}}。

14.14 [分片] inline 表达式返回结果为何出现浮点数？

回答：

Java 的整数相除结果是整数，但是对于 inline 表达式中的 Groovy 语法则不同，整数相除结果是浮点数。想获得除法整数结果需要将 A/B 改为 A.intdiv(B)。

14.15 [分片] 如果只有部分数据库分库分表，是否需要将不分库分表的表也配置在分片规则中？

回答：

不需要，DBPlusEngine 会自动识别。

14.16 [分片] 指定了泛型为 Long 的 SingleKeyTableShardingAlgorithm，遇到 ClassCastException: Integer can not cast to Long?

回答：

必须确保数据库表中该字段和分片算法该字段类型一致，如：数据库中该字段类型为 int(11)，泛型所对应的分片类型应为 Integer，如果需要配置为 Long 类型，请确保数据库中该字段类型为 bigint。

14.17 [分片、PROXY] 实现 StandardShardingAlgorithm 自定义算法时，指定了 Comparable 的具体类型为 Long，且数据库表中字段类型为 bigint，出现 ClassCastException: Integer can not cast to Long 异常。

回答：

实现 doSharding 方法时，不建议指定方法声明中 Comparable 具体的类型，而是在 doSharding 方法实现中对类型进行转换，可以参考 ModShardingAlgorithm#doSharding 方法

14.18 [分片] DBPlusEngine 提供的默认分布式自增主键策略为什么是不连续的，且尾数大多为偶数？

回答：

DBPlusEngine 采用 snowflake 算法作为默认的分布式自增主键策略，用于保证分布式的情况下可以无中心化的生成不重复的自增序列。因此自增主键可以保证递增，但无法保证连续。

而 snowflake 算法的最后 4 位是在同一毫秒内的访问递增值。因此，如果毫秒内并发度不高，最后 4 位为零的几率则很大。因此并发度不高的应用生成偶数主键的几率会更高。

在 3.1.0 版本中，尾数大多为偶数的问题已彻底解决，参见：<https://github.com/apache/shardingsphere/issues/1617>

14.19 [分片] 如何在 inline 分表策略时，允许执行范围查询操作（BETWEEN AND、>、<、>=、<=）？

回答：

1. 需要使用 4.1.0 或更高版本。
2. 调整以下配置项（需要注意的是，此时所有的范围查询将会使用广播的方式查询每一个分表）：
 - 4.x 版本：allow.range.query.with.inline.sharding 设置为 true 即可（默认为 false）。
 - 5.x 版本：在 InlineShardingStrategy 中将 allow-range-query-with-inline-sharding 设置为 true 即可（默认为 false）。

14.20 [分片] 为什么我实现了 KeyGenerateAlgorithm 接口，也配置了 Type，但是自定义的分布式主键依然不生效？

回答：

Service Provider Interface (SPI) 是一种为了被第三方实现或扩展的 API，除了实现接口外，还需要在 META-INF/services 中创建对应文件来指定 SPI 的实现类，JVM 才会加载这些服务。

具体的 SPI 使用方式，请大家自行搜索。

与分布式主键 KeyGenerateAlgorithm 接口相同，其他 DBPlusEngine 的扩展功能也需要用相同的方式注入才能生效。

14.21 [分片] DBPlusEngine 除了支持自带的分布式自增主键之外，还能否支持原生的自增主键？

回答：是的，可以支持。但原生自增主键有使用限制，即不能将原生自增主键同时作为分片键使用。

由于 DBPlusEngine 并不知晓数据库的表结构，而原生自增主键是不包含在原始 SQL 中内的，因此 DBPlusEngine 无法将该字段解析为分片字段。如自增主键非分片键，则无需关注，可正常返回；若自增主键同时作为分片键使用，DBPlusEngine 无法解析其分片值，导致 SQL 路由至多张表，从而影响应用的正确性。

而原生自增主键返回的前提条件是 INSERT SQL 必须最终路由至一张表，因此，面对返回多表的 INSERT SQL，自增主键则会返回零。

14.22 [数据加密] JPA 和数据加密无法一起使用，如何解决？

回答：

由于数据加密的 DDL 尚未开发完成，因此对于自动生成 DDL 语句的 JPA 与数据加密一起使用时，会导致 JPA 的实体类 (Entity) 无法同时满足 DDL 和 DML 的情况。

解决方案如下：

1. 以需要加密的逻辑列名编写 JPA 的实体类 (Entity)。
2. 关闭 JPA 的 auto-ddl，如 auto-ddl=none。
3. 手动建表，建表时应使用数据加密配置的 cipherColumn,plainColumn 和 assistedQueryColumn 代替逻辑列。

14.23 [DistSQL] 使用 DistSQL 添加数据源时，如何设置自定义的 Driver 连接参数或连接池属性？

回答：

1. 如需自定义 Driver 参数，请使用 urlSource 的方式定义 dataSource。
2. ShardingSphere 预置了必要的连接池参数，如 maxPoolSize、idleTimeout 等。如需增加或覆盖参数配置，请在 dataSource 中通过 PROPERTIES 指定。
3. 以上规则请参考 [相关介绍](#)

14.24 [DistSQL] 使用 DistSQL 删除资源时，出现 Resource [xxx] is still used by [SingleTableRule]。

回答：

1. 被规则引用的资源将无法被删除
2. 若资源只被 single table rule 引用，且用户确认可以忽略该限制，则可以添加可选参数 ignore single tables 进行强制删除

14.25 [DistSQL] 使用 DistSQL 添加资源时，出现 Failed to get driver instance for jdbcURL=xxx。

回答：

DBPlusEngine-Proxy 在部署过程中没有添加 jdbc 驱动，需要将 jdbc 驱动放入 DBPlusEngine-Proxy 解压后的 ext-lib 目录，例如：mysql-connector。

14.26 [其他] 如果 SQL 在 DBPlusEngine 中执行不正确，该如何调试？

回答：

在 DBPlusEngine-Proxy 以及 DBPlusEngine-Driver 1.5.0 版本之后提供了 sql.show 的配置，可以将解析上下文和改写后的 SQL 以及最终路由至的数据源的细节信息全部打印至 info 日志。sql.show 配置默认关闭，如果需要请通过配置开启。

注意：5.x 版本以后，sql.show 参数调整为 sql-show。

14.27 [其他] 阅读源码时为什么会出现编译错误？IDEA 不索引生成的代码？

回答：

DBPlusEngine 使用 lombok 实现极简代码。关于更多使用和安装细节，请参考 [lombok 官网](#)。

org.apache.shardingsphere.sql.parser.autogen 包下的代码由 ANTLR 生成，可以执行以下命令快速生成：

```
./mvnw -Dcheckstyle.skip=true -Drat.skip=true -Dmaven.javadoc.skip=true -Djacoco.skip=true -DskipITs -DskipTests install -T1C
```

生成的代码例如 org.apache.shardingsphere.sql.parser.autogen.PostgreSQLStatementParser 等 Java 文件由于较大，默认配置的 IDEA 可能不会索引该文件。可以调整 IDEA 的属性：idea.max.intellisense.filesize=10000

14.28 [其他] 使用 SQLSever 和 PostgreSQL 时，聚合列不加别名会抛异常？

回答：

SQLServer 和 PostgreSQL 获取不加别名的聚合列会改名。例如，如下 SQL：

```
SELECT SUM(num), SUM(num2) FROM tablexxx;
```

SQLServer 获取到的列为空字符串和 (2)，PostgreSQL 获取到的列为空 sum 和 sum(2)。这将导致 ShardingSphere 在结果归并时无法找到相应的列而出错。

正确的 SQL 写法应为：

```
SELECT SUM(num) AS sum_num, SUM(num2) AS sum_num2 FROM tablexxx;
```

14.29 [其他] Oracle 数据库使用 Timestamp 类型的 Order By 语句抛出异常提示 “Order by value must implements Comparable” ?

回答：

针对上面问题解决方式有两种：1. 配置启动 JVM 参数 “-oracle.jdbc.J2EE13Compliant=true” 2. 通过代码在项目初始化时设置 `System.getProperties().setProperty(“oracle.jdbc.J2EE13Compliant”, “true”)`;

原因如下：

`org.apache.shardingsphere.sharding.merge.dql.orderby.OrderByValue#getOrderValues()` 方法如下：

```
private List<Comparable<?>> getOrderValues() throws SQLException {
    List<Comparable<?>> result = new ArrayList<>(orderByItems.size());
    for (OrderItem each : orderByItems) {
        Object value = resultSet.getObject(each.getIndex());
        Preconditions.checkNotNull(value || value instanceof Comparable, "Order by value must implements Comparable");
        result.add((Comparable<?>) value);
    }
    return result;
}
```

使用了 `resultSet.getObject(int index)` 方法，针对 Timestamp oracle 会根据 `oracle.jdbc.J2EE13Compliant` 属性判断返回 `java.sql.Timestamp` 还是自定义 `oracle.sql.TIMESTAMP` 详见 `ojdbc` 源码 `oracle.jdbc.driver.TimestampAccessor#getObject(int var1)` 方法：

```
Object getObject(int var1) throws SQLException {
    Object var2 = null;
    if (this.rowSpaceIndicator == null) {
        DatabaseError.throwSQLException(21);
    }

    if (this.rowSpaceIndicator[this.indicatorIndex + var1] != -1) {
        if (this.externalType != 0) {
            switch (this.externalType) {
                case 93:
                    return this.getTimestamp(var1);
                default:
                    DatabaseError.throwSQLException(4);
                    return null;
            }
        }

        if (this.statement.connection.j2ee13Compliant) {
            var2 = this.getTimestamp(var1);
        } else {
            var2 = this.getTIMESTAMP(var1);
        }
    }

    return var2;
}
```

14.30 [其他] Windows 环境下，通过 Git 克隆 DBPlusEngine 源码时为什么提示文件名过长，如何解决？

回答：

为保证源码的可读性，DBPlusEngine 编码规范要求类、方法和变量的命名要做到顾名思义，避免使用缩写，因此可能导致部分源码文件命名较长。由于 Windows 版本的 Git 是使用 msys 编译的，它使用了旧版本的 Windows Api，限制文件名不能超过 260 个字符。

解决方案如下：

打开 cmd.exe（你需要将 git 添加到环境变量中）并执行下面的命令，可以让 git 支持长文件名：

```
git config --global core.longpaths true
```

如果是 Windows 10，还需要通过注册表或组策略，解除操作系统的文件名长度限制（需要重启）：> 在注册表编辑器中创建 HKLM\SYSTEM\CurrentControlSet\Control\FileSystem LongPathsEnabled，类型为 REG_DWORD，并设置为 1。> 或者从系统菜单点击设置图标，输入“编辑组策略”，然后在打开的窗口依次进入“计算机管理”>“管理模板”>“系统”>“文件系统”，在右侧双击“启用 win32 长路径”。

参考资料：<https://docs.microsoft.com/zh-cn/windows/desktop/FileIO/naming-a-file> <https://ourcodeworld.com/articles/read/109/how-to-solve-filename-too-long-error-in-git-powershell-and-github-application-for-windows>

14.31 [其他] Type is required 异常的解决方法？

回答：

DBPlusEngine 中很多功能实现类的加载方式是通过 SPI 注入的方式完成的，如分布式主键，注册中心等；这些功能通过配置中 type 类型来寻找对应的 SPI 实现，因此必须在配置文件中指定类型。

14.32 [其他] 服务启动时如何加快 metadata 加载速度？

回答：

1. 升级到 4.0.1 以上的版本，以提高 metadata 的加载速度。
2. 参照你采用的连接池，将：
 - 配置项 max.connections.size.per.query（默认值为 1）调高（版本 >= 3.0.0.M3 且低于 5.0.0）。
 - 配置项 max-connections-size-per-query（默认值为 1）调高（版本 >= 5.0.0）。

14.33 [其他] ANTLR 插件在 src 同级目录下生成代码，容易误提交，如何避免？

回答：

进入 Settings -> Languages & Frameworks -> ANTLR v4 default project settings 配置生成代码的输出目录为 target/gen，如图：

Fig. 1: Configure ANTLR plugin

14.34 [其他] 使用 Proxool 时分库结果不正确？

回答：

使用 Proxool 配置多个数据源时，应该为每个数据源设置 alias，因为 Proxool 在获取连接时会判断连接池中是否包含已存在的 alias，不配置 alias 会造成每次都只从一个数据源中获取连接。

以下是 Proxool 源码中 ProxoolDataSource 类 getConnection 方法的关键代码：

```
if(!ConnectionPoolManager.getInstance().isPoolExists(this.alias)) {  
    this.registerPool();  
}
```

更多关于 alias 使用方法请参考 [Proxool 官网](#)。

PS：sourceforge 网站需要翻墙访问。

14.35 [其他] 使用 Spring Boot 2.x 集成 DBPlusEngine 时，配置文件中的属性设置不生效？

回答：

需要特别注意，Spring Boot 2.x 环境下配置文件的属性名称约束为仅允许小写字母、数字和短横线，即 [a-z][0-9] 和 -。

原因如下：

Spring Boot 2.x 环境下，DBPlusEngine 通过 Binder 来绑定配置文件，属性名称不规范（如：驼峰或下划线等）会导致属性设置不生效从而校验属性值时抛出 NullPointerException 异常。参考以下错误示例：

下划线示例：database_inline

```
spring.shardingsphere.rules.sharding.sharding-algorithms.database_inline.type=INLINE  
spring.shardingsphere.rules.sharding.sharding-algorithms.database_inline.props.algorithm-expression=ds-${user_id % 2}
```

Caused by: org.springframework.beans.factory.BeanCreationException: Error creating bean **with** name 'database_inline': Initialization of bean failed; nested exception is java.lang.NullPointerException: Inline sharding algorithm expression cannot be null.

```
...  
Caused by: java.lang.NullPointerException: Inline sharding algorithm expression cannot be null.  
    at com.google.common.base.Preconditions.checkNotNull(Preconditions.java:897)  
    at org.apache.shardingsphere.sharding.algorithm.sharding.inline.InlineShardingAlgorithm.  
getAlgorithmExpression(InlineShardingAlgorithm.java:58)  
    at org.apache.shardingsphere.sharding.algorithm.sharding.inline.InlineShardingAlgorithm.init(InlineShardingAlgorithm.  
java:52)  
    at org.apache.shardingsphere.spring.boot.registry.AbstractAlgorithmProvidedBeanRegistry.  
postProcessAfterInitialization(AbstractAlgorithmProvidedBeanRegistry.java:98)  
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.  
applyBeanPostProcessorsAfterInitialization(AbstractAutowireCapableBeanFactory.java:431)  
    ...
```

驼峰示例：databaseInline

```
spring.shardingsphere.rules.sharding.sharding-algorithms.databaseInline.type=INLINE  
spring.shardingsphere.rules.sharding.sharding-algorithms.databaseInline.props.algorithm-expression=ds-${user_id % 2}
```

Caused by: org.springframework.beans.factory.BeanCreationException: Error creating bean **with** name 'databaseInline': Initialization of bean failed; nested exception is java.lang.NullPointerException: Inline sharding algorithm expression cannot be null.

```
...  
Caused by: java.lang.NullPointerException: Inline sharding algorithm expression cannot be null.  
    at com.google.common.base.Preconditions.checkNotNull(Preconditions.java:897)
```

```
at org.apache.shardingsphere.sharding.algorithm.sharding.inline.InlineShardingAlgorithm.  
getAlgorithmExpression(InlineShardingAlgorithm.java:58)  
at org.apache.shardingsphere.sharding.algorithm.sharding.inline.InlineShardingAlgorithm.init(InlineShardingAlgorithm.  
java:52)  
at org.apache.shardingsphere.spring.boot.registry.AbstractAlgorithmProvidedBeanRegistry.  
postProcessAfterInitialization(AbstractAlgorithmProvidedBeanRegistry.java:98)  
at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.  
applyBeanPostProcessorsAfterInitialization(AbstractAutowireCapableBeanFactory.java:431)  
...
```

从异常堆栈中分析可知: AbstractAlgorithmProvidedBeanRegistry.registerBean 方法调用 PropertyUtil.containsPropertyPrefix(environment, prefix) 方法判断指定前缀 prefix 的配置是否存在, 而 PropertyUtil.containsPropertyPrefix(environment, prefix) 方法, 在 Spring Boot 2.x 环境下使用了 Binder, 不规范的属性名称 (如: 驼峰或下划线等) 会导致属性设置不生效。