操作手册

V2.0.1

目录

1. 欢迎使用

欢迎使用性能测试保障平台，本操作手册编写的主要目的是为了指导用户正确的操作和使用本产品。内容主要描述产品各项功能的使用方法。

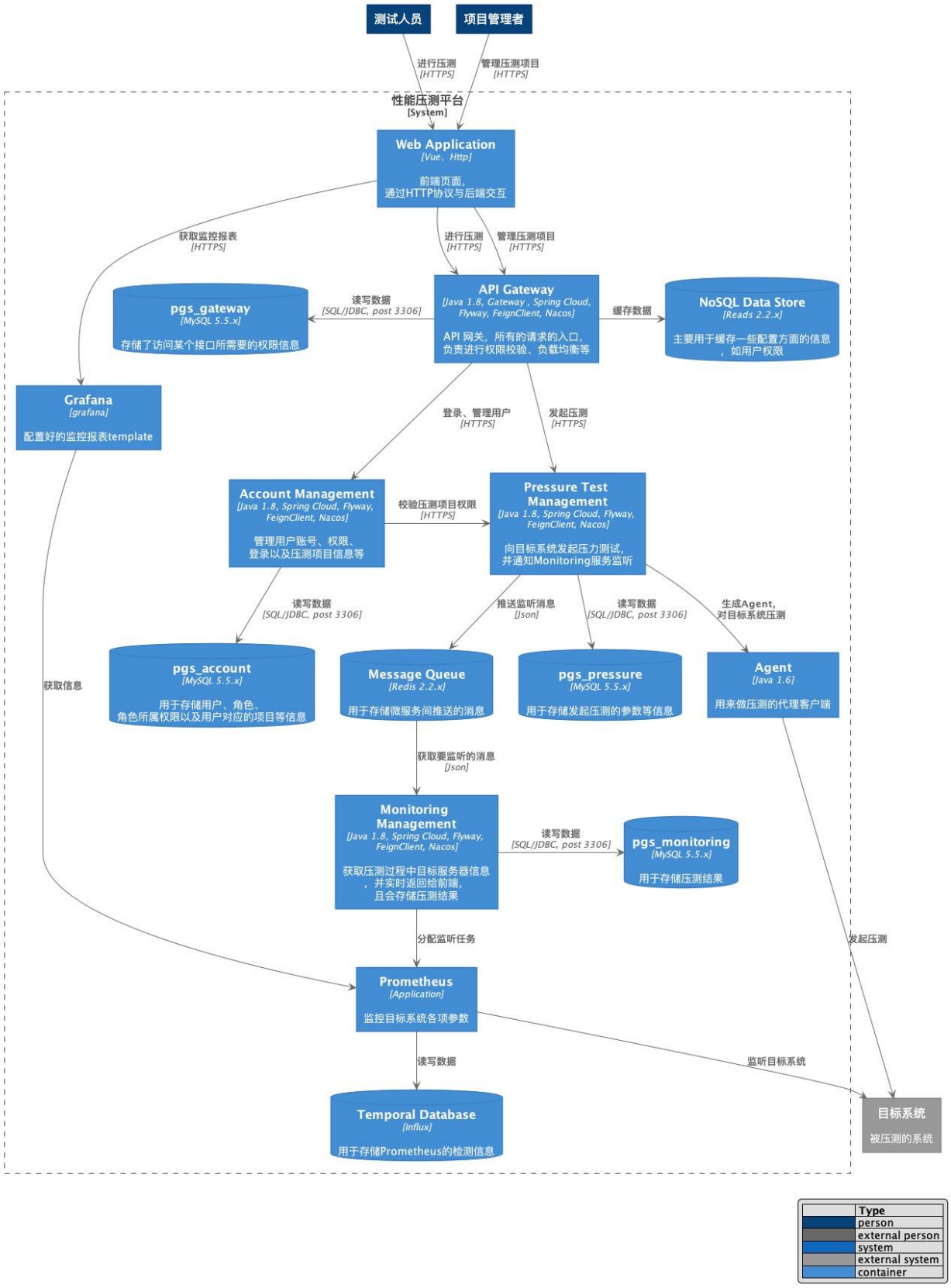
1.1. 更新记录

日期	内容
2020-11-19	创建文档
2022-08-16	升级版本 2.0.1

2. 结构流程说明

Jpower 平台是结合项目管理、性能测试、服务器监控、性能优化于一体的综合性能保障平台。

平台功能主要分为五部分组成：项目管理、测试脚本、测试场景、执行队列、场景结果。

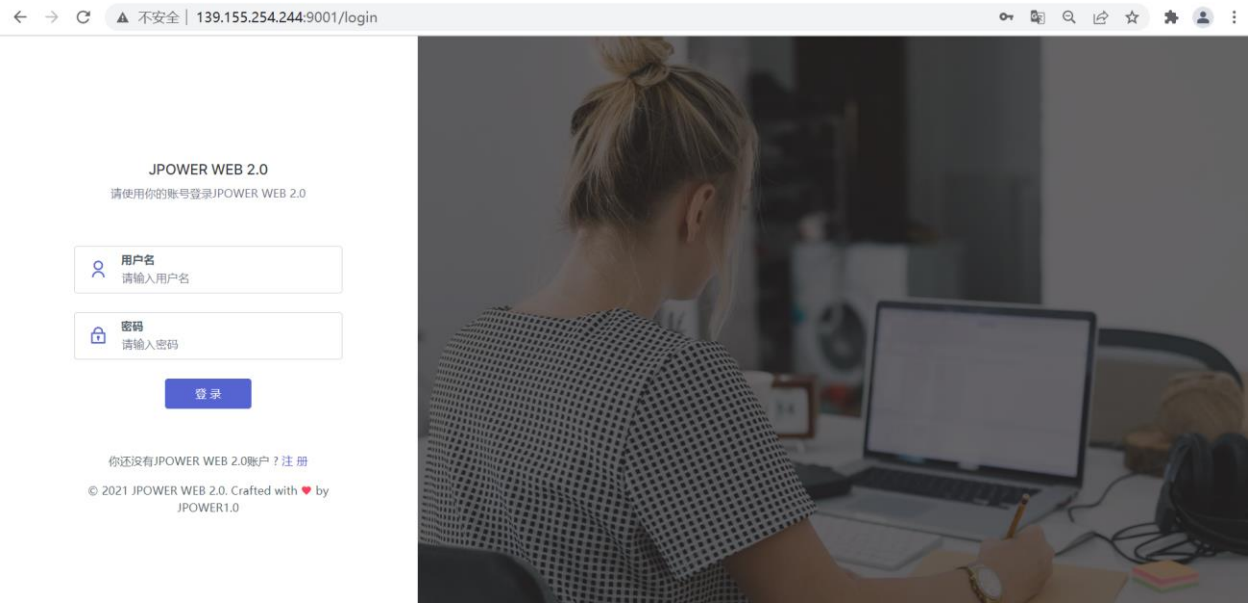


3. 平台使用说明

3.1. 启动

1、平台部署详见《平台安装部署手册》；

2、通过浏览器输入地址 http://139.155.254.244:9001/login，输入用户名密码登陆。



3.2. 项目管理

3.3.1. 创建项目

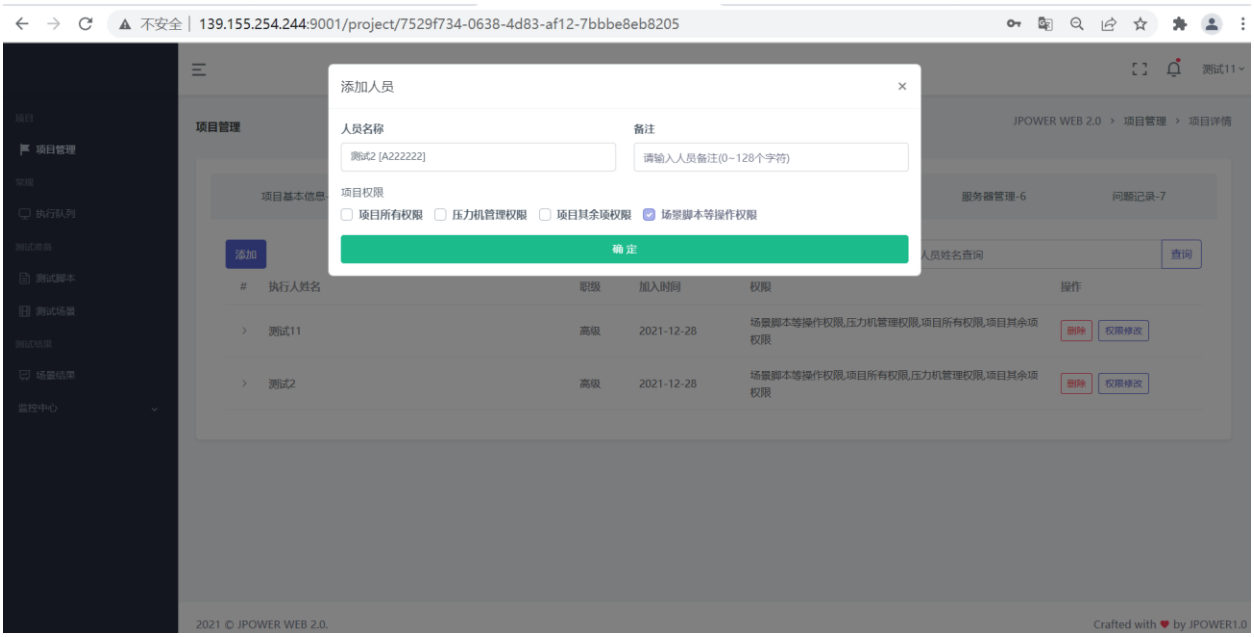
测试人员可根据分配的测试需求，直接创建项目

3.3.2. 项目基本信息

创建项目后，可对项目基本信息进行维护，包含项目名称、项目开始时间、项目结束时间等

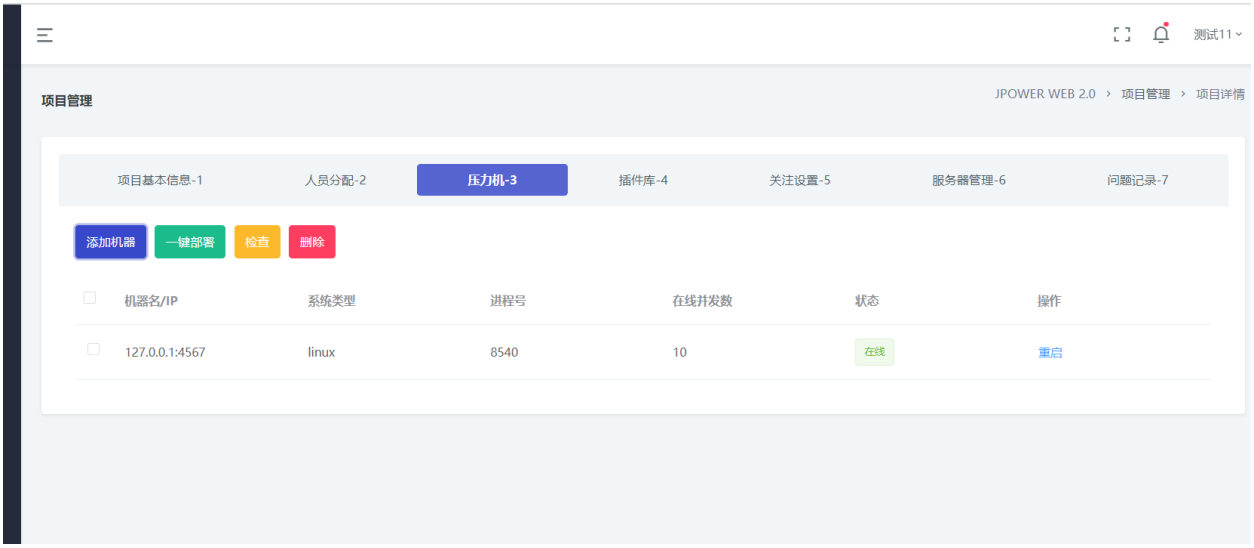
3.3.3. 人员分配

创建项目人员，可对该项目涉及的人员及进行添加，并分配权限。



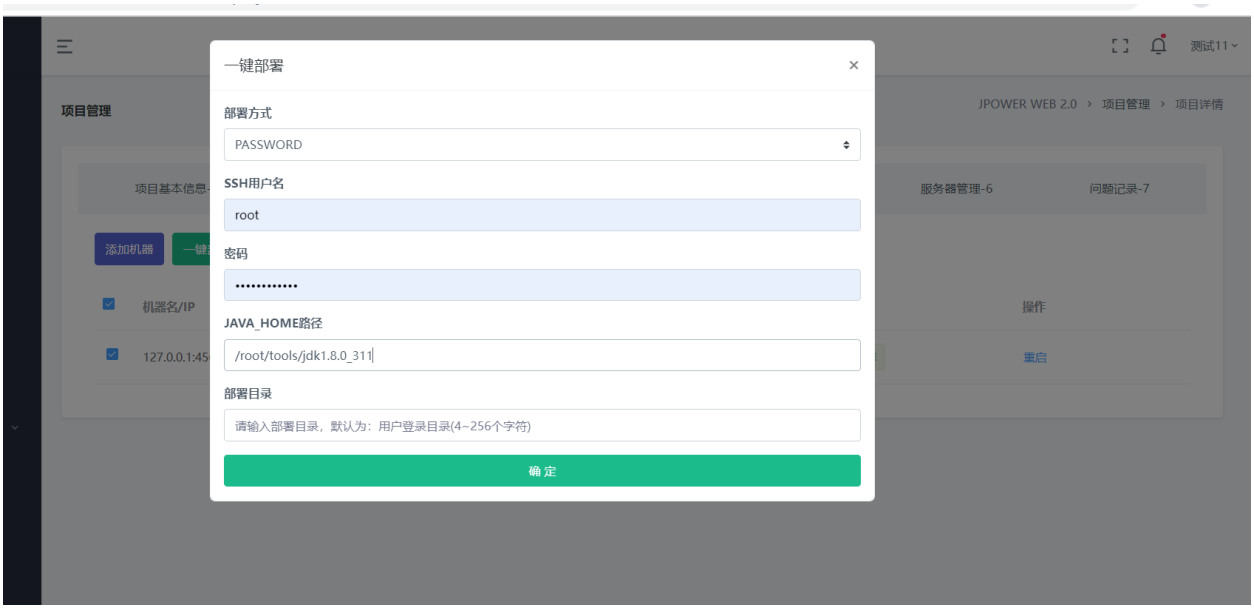
3.3.4. 压力机部署

添加需要部署的压力机 ip 地址和端口

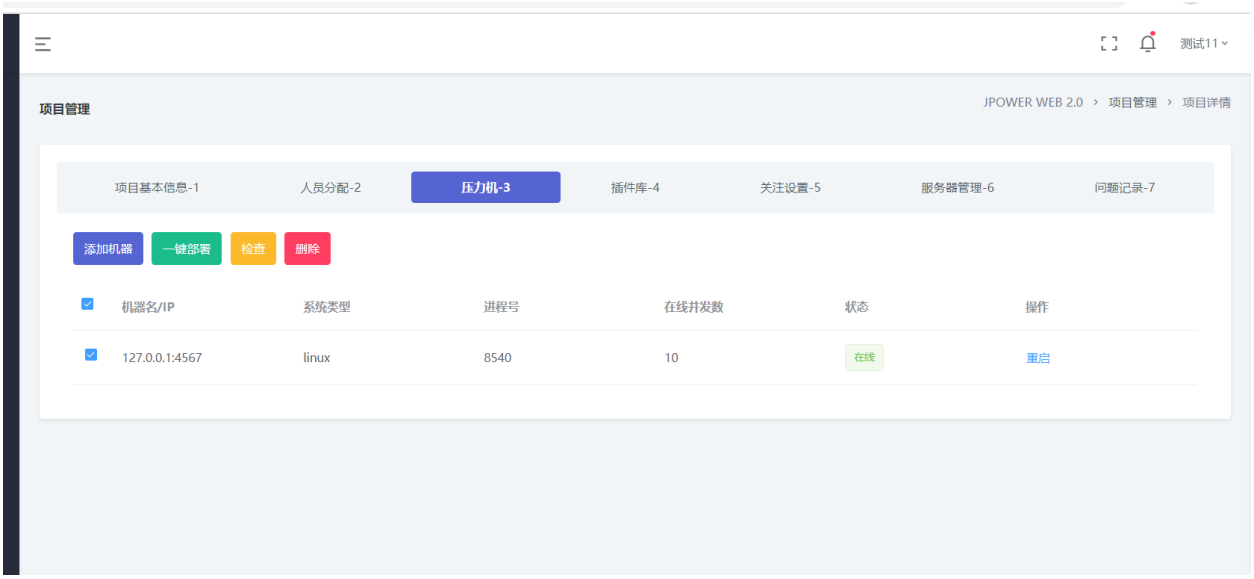


可将添加的压力机地址进行一键部署，输入服务器账号/密码，java_home 路径(通过压力机服务器账号名密码登陆后，执行 echo $JAVA_HOME 命令，查看 java_home 路径)，每个 ip 仅能添加一个项目。

【部署方式】:分为 SSH 用户密码方式和 SSH-RS 密钥两种方式，我们常用的方式是“SSH 用户密码方式”

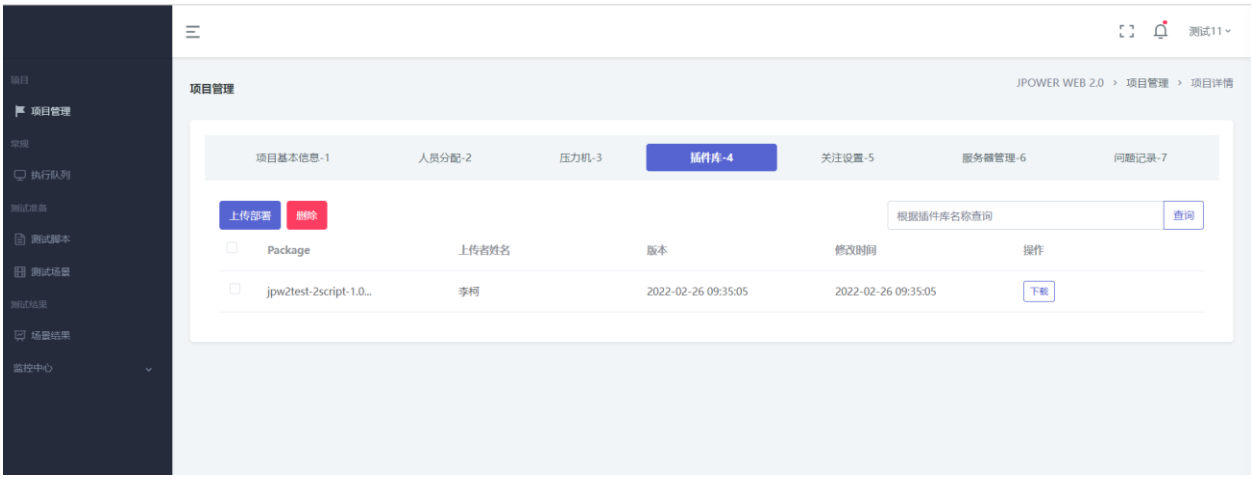


部署后状态情况如下图，可删除、重启、状态检查等



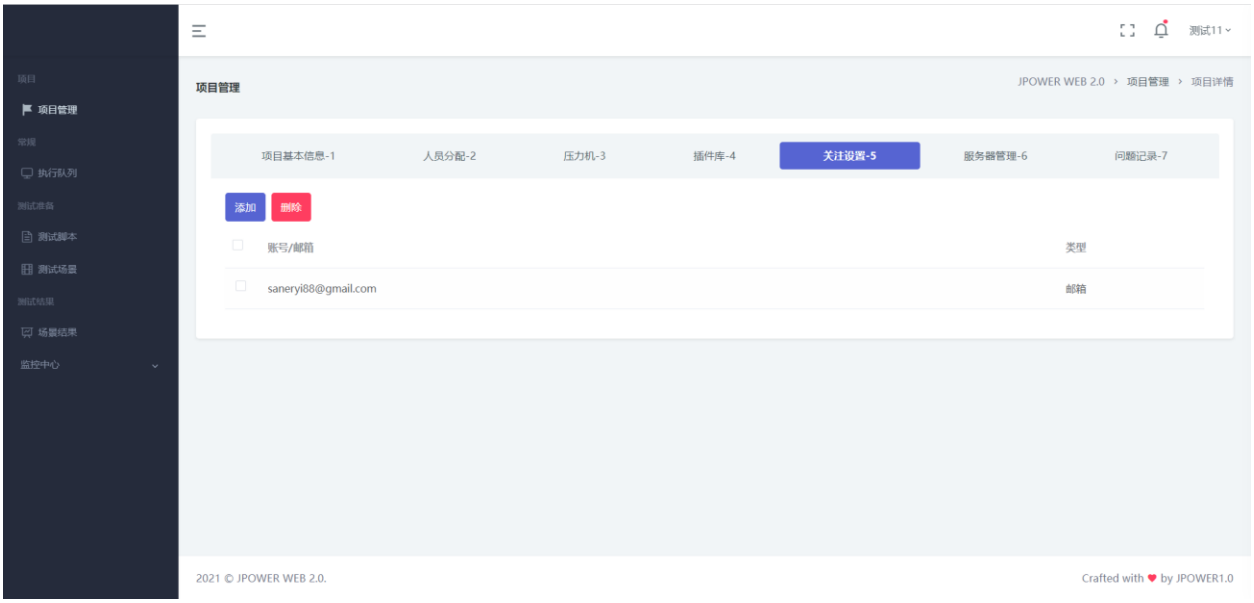
3.3.5. 插件库

项目中涉及调用的插件如证数、MD5 加密 jar 等可上传插件库



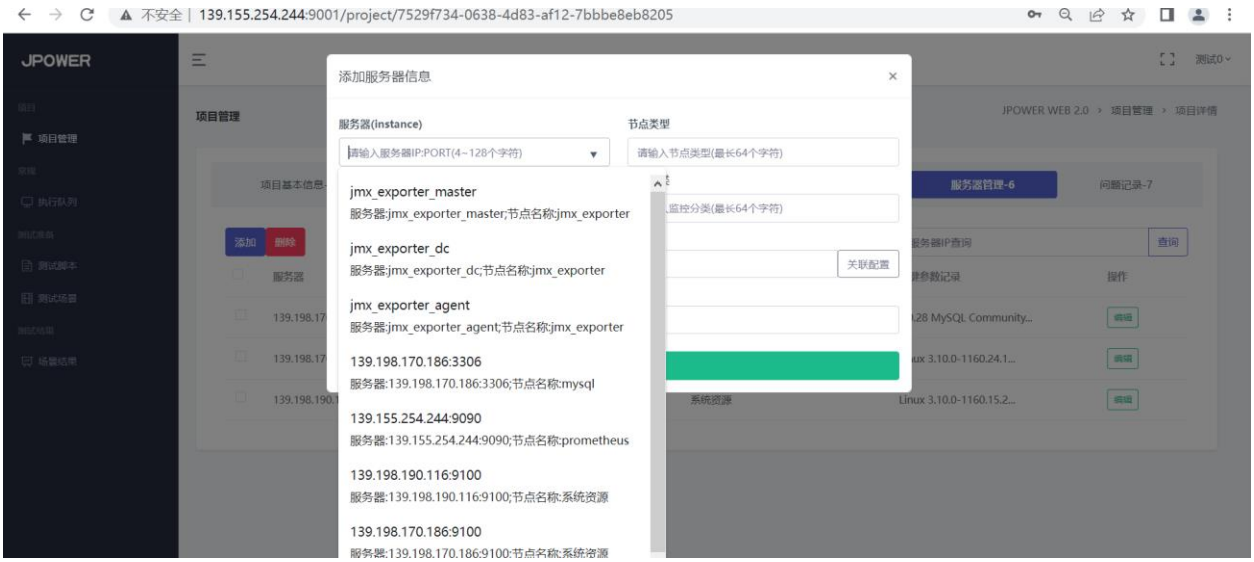
3.3.6. 关注设置

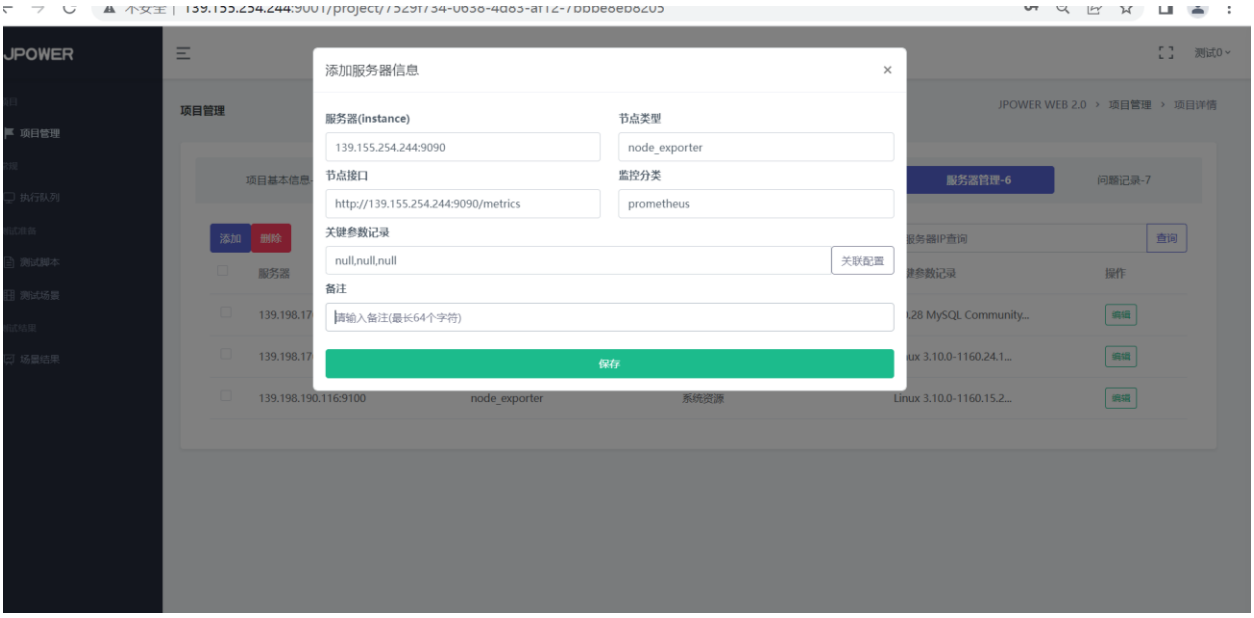
关注设置，可根据需求添加需要推送的邮箱或者注册得 Jpower 账号，重要测试结果可选是否推送给相关人员。

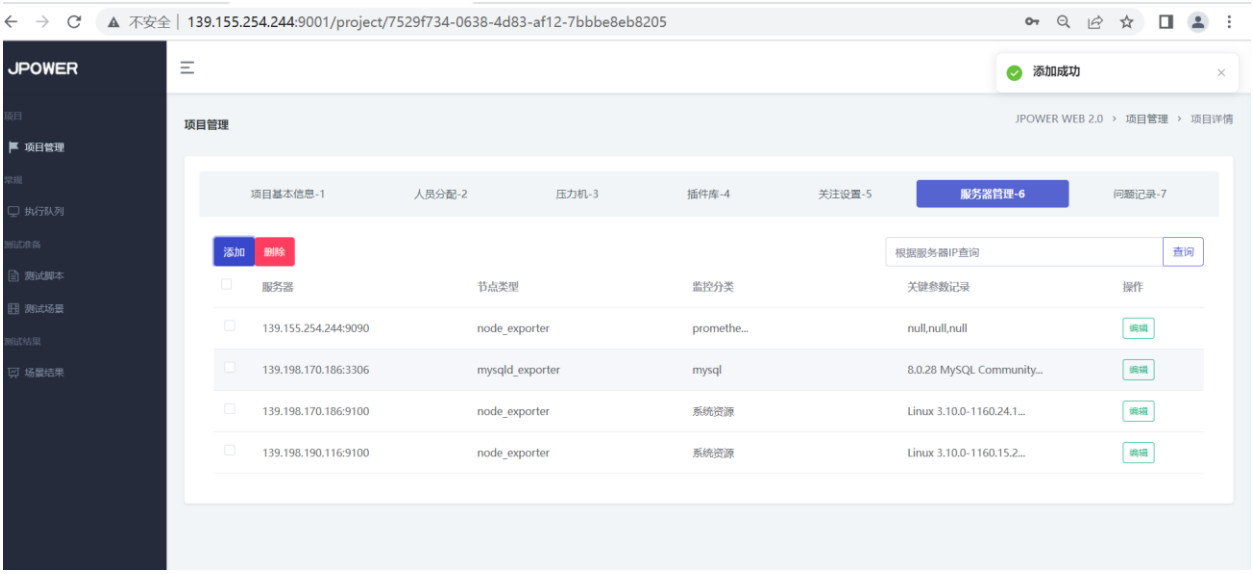


3.3.7. 服务器管理

测试人员可对该项目涉及服务器，进行添加维护，以便于测试执行中实时查看服务器资源情况，可从列表中查询，进行节点选择，其它参数可同步显示。



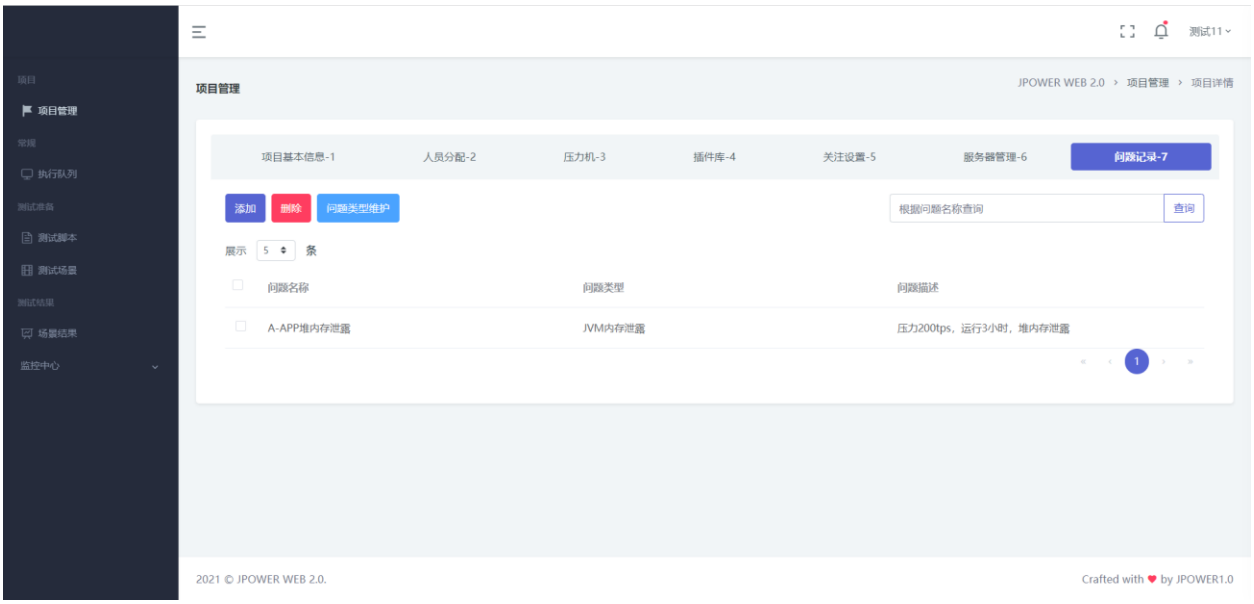




添加后可对服务器进行维护。

3.3.8. 问题记录

测试人员，可对该项目测试过程中，遇到的重大性能问题进行记录维护，以便于后续遇到同样情况，进行分析排查。



3.3. 测试脚本

主要是测试前期的准备工作，包括测试脚本的准备，测试数据的准备。

3.5.1. 脚本开发

脚本开发工作,主要市线下开发，将开发脚本打包上传至平台。

3.5.2. 参数设置

上传脚本后，系统自动识别脚本中的事务及参数，在此界面中可查看到。点击左侧脚本名称后，右侧对应显示脚本事务及脚本参数信息，如下图，脚本事务不可编辑，脚本参数可编辑。

脚本参数编辑：

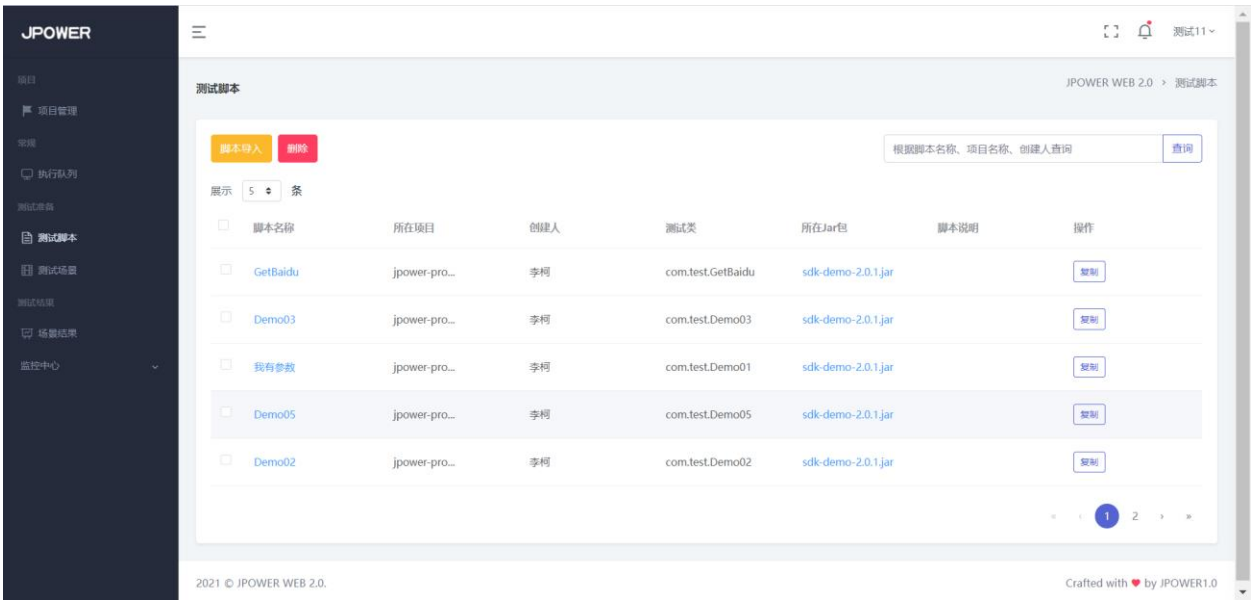
【参数名称】不可编辑

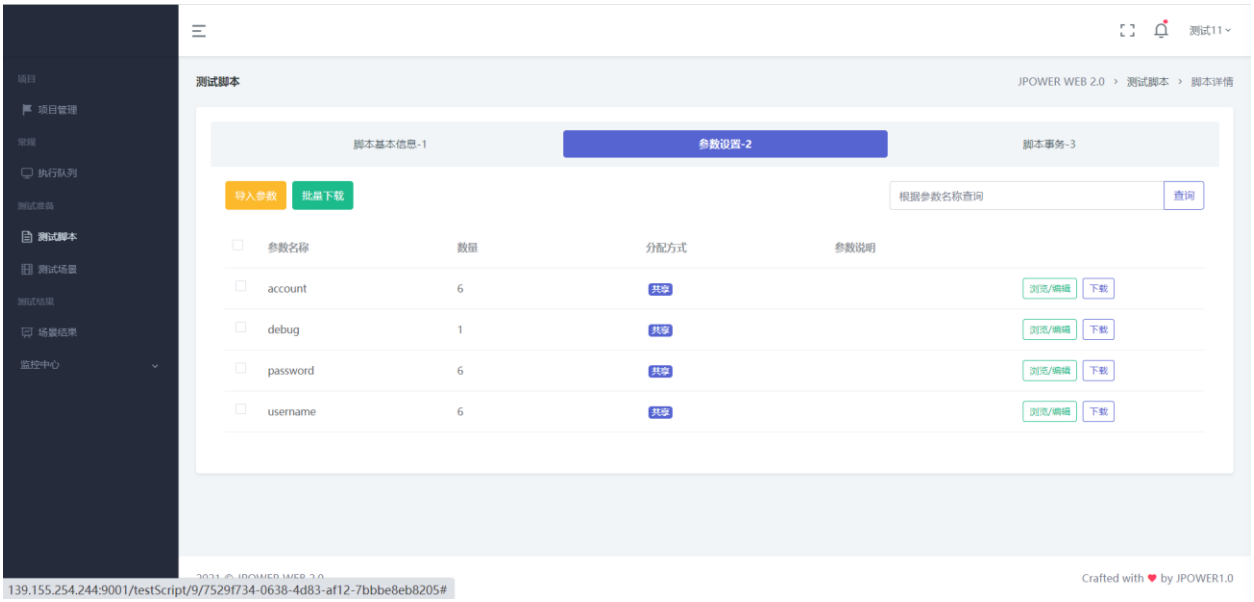
【分配方式】分为三种，共享、平分、预分块，其中共享是指此参数可与本项目中其他脚本共享参数，平分是指此参数可以与之取相同参数的脚本平分数据，预分块是指此参数可以与之取相同参数的脚本人为划分数据数量。

注意：常规意义的按照并发数的分配数据在脚本中代码设置。

【参数值起始行】使用参数文本中的第几行开始

【相同参数】可选择本项目下共享的参数

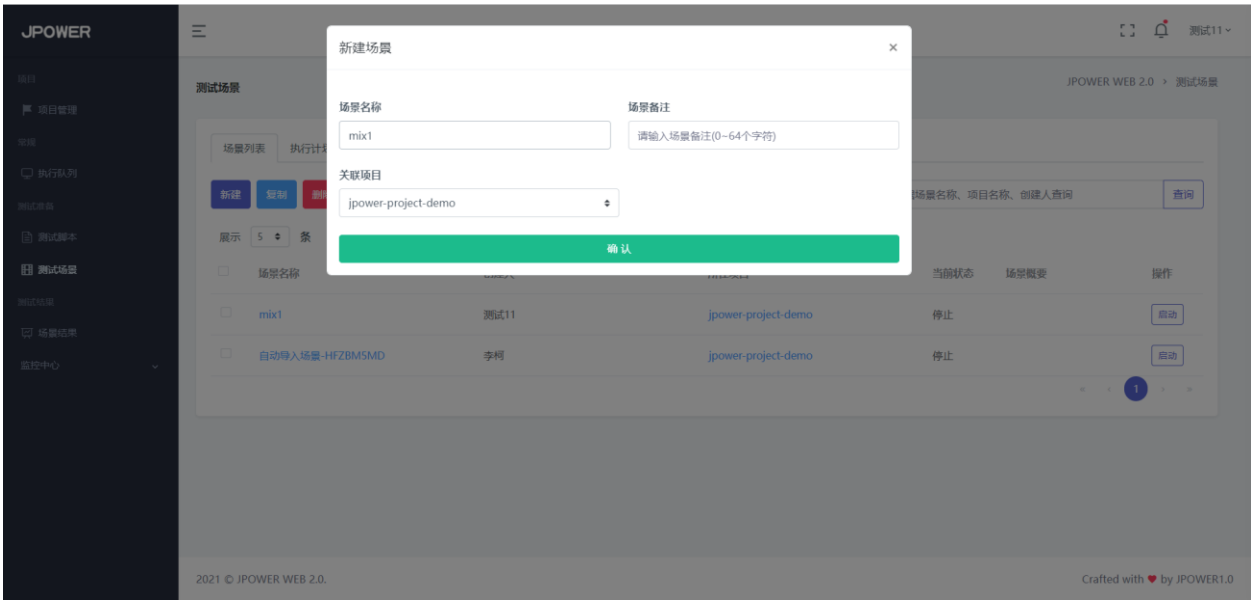




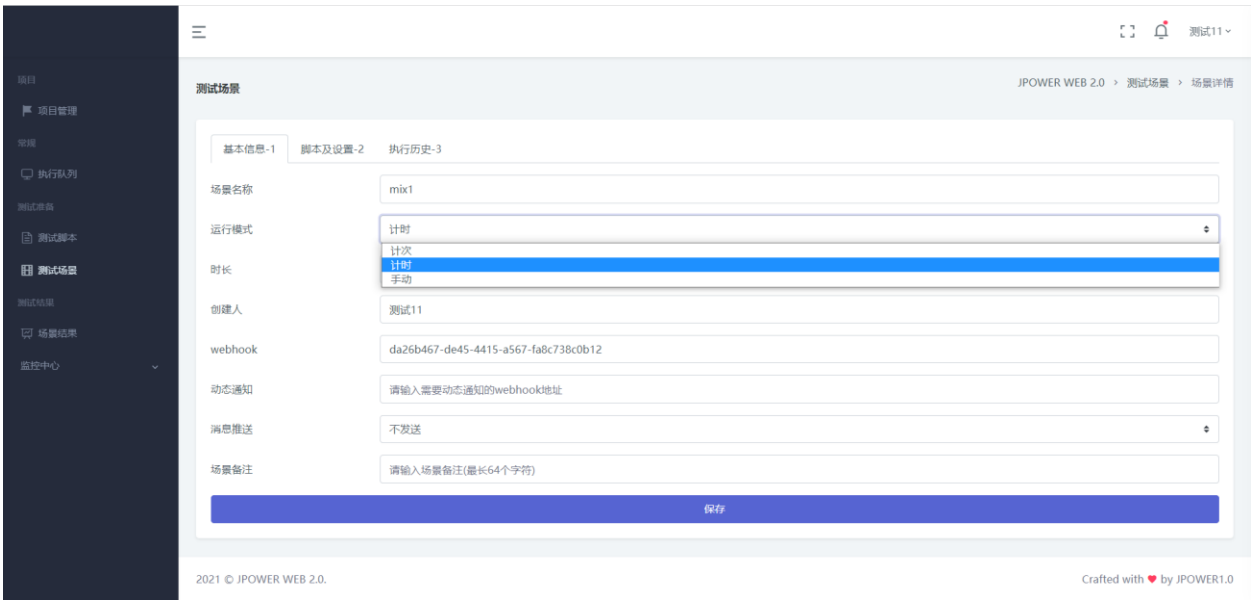
3.4. 场景执行

3.6.1. 测试场景

【新建】弹出新建场景弹框，进行场景的基本设置；



【运行类型】可以按照时间或按照次数或手动，按照时间—场景的运行时间，手动—不限制时间，手动停止场景

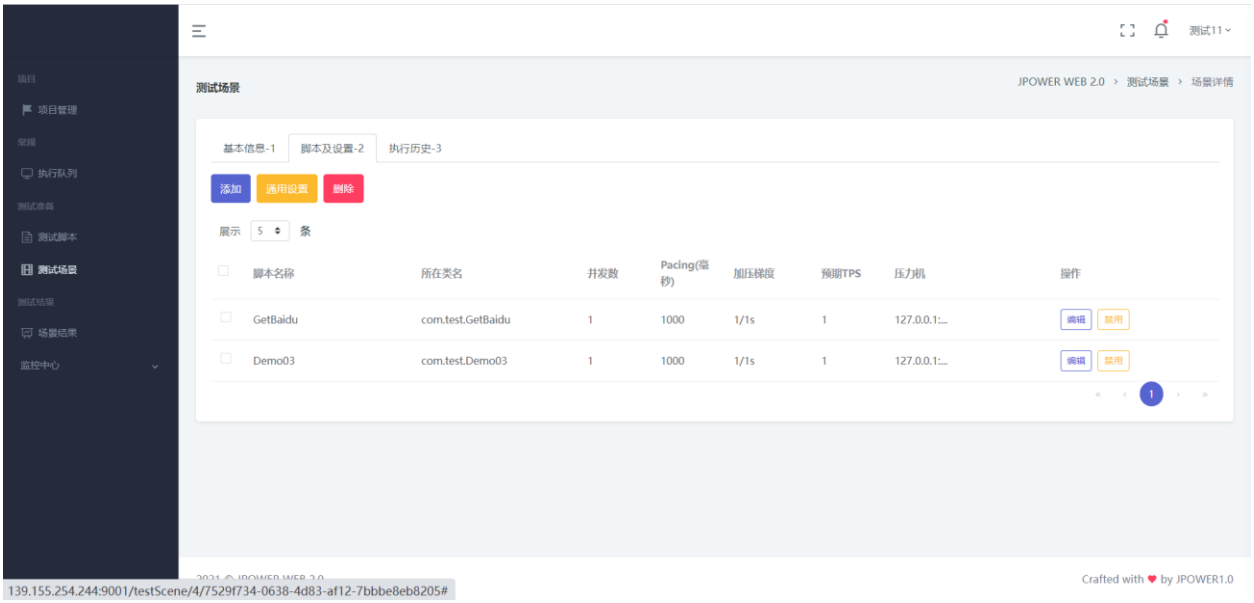


【线程数】并发用户数

【Pacing】时间间隔

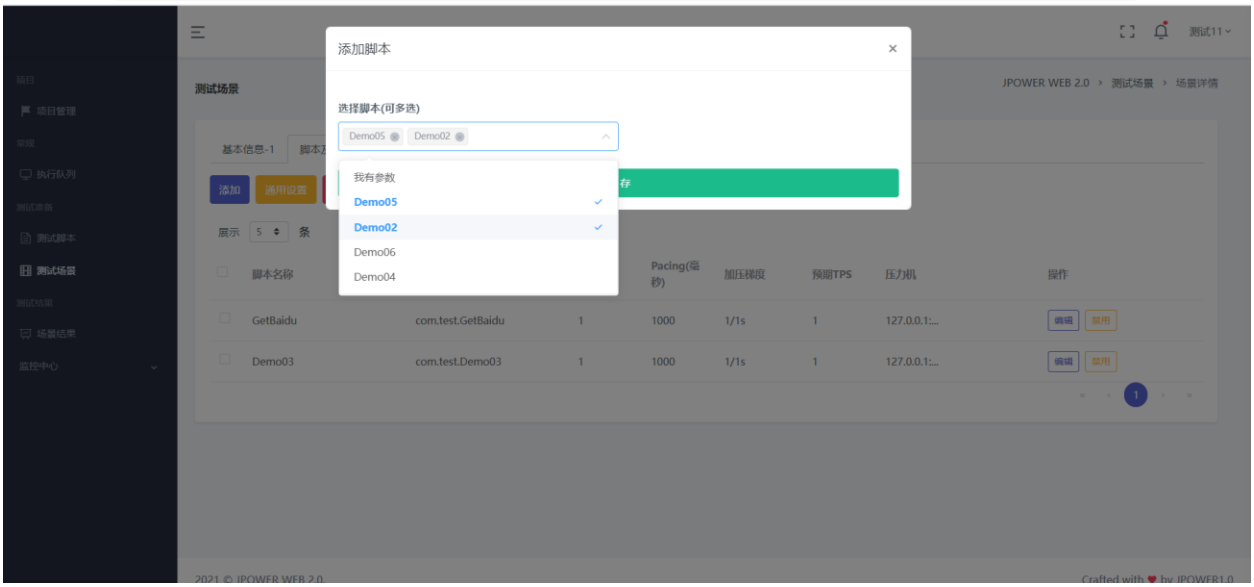
【压力梯度】增加压力的时间步长及并发用户数的增幅

【可用脚本】根据需求选择脚本，<>进行脚本的选择，出现在右侧文本框中的此场景中运行的脚本

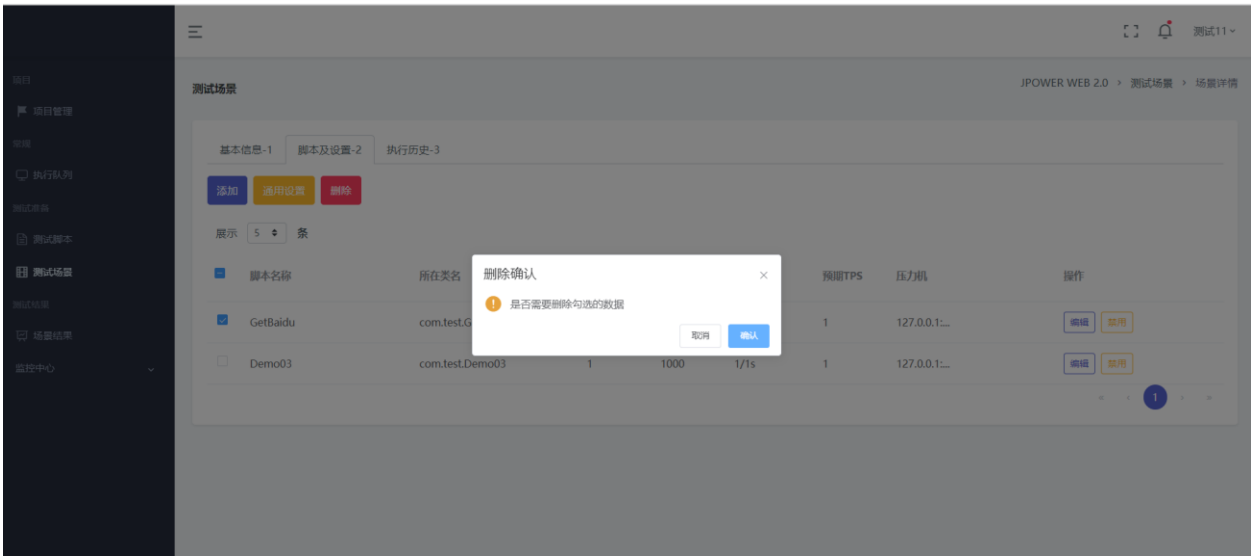


新建后在场景设计界面即可看到此场景，点击场景名称后，即可显示场景中脚本及具体设置

【添加脚本】可根据需求快速添加脚本

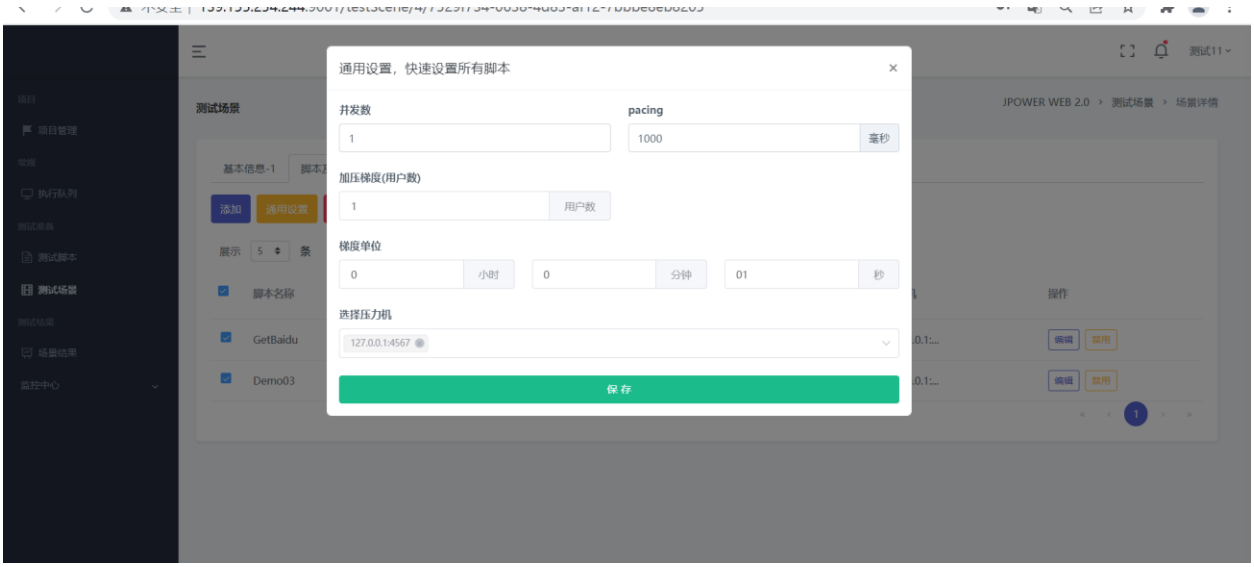


【删除脚本】可根据需求快速删除脚本

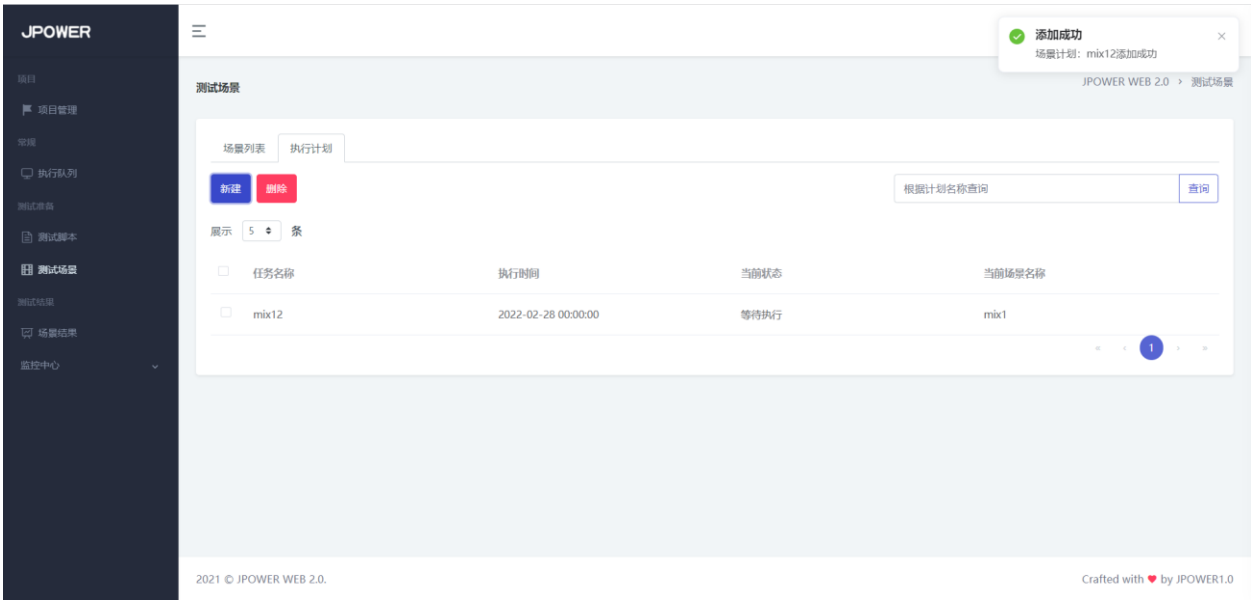


【场景设置】计秒、计次两种方式进行场景的设置，计秒为场景运行时间，计次是每个脚本均按照这些次数运行，为全局设置。

【一键设置脚本】

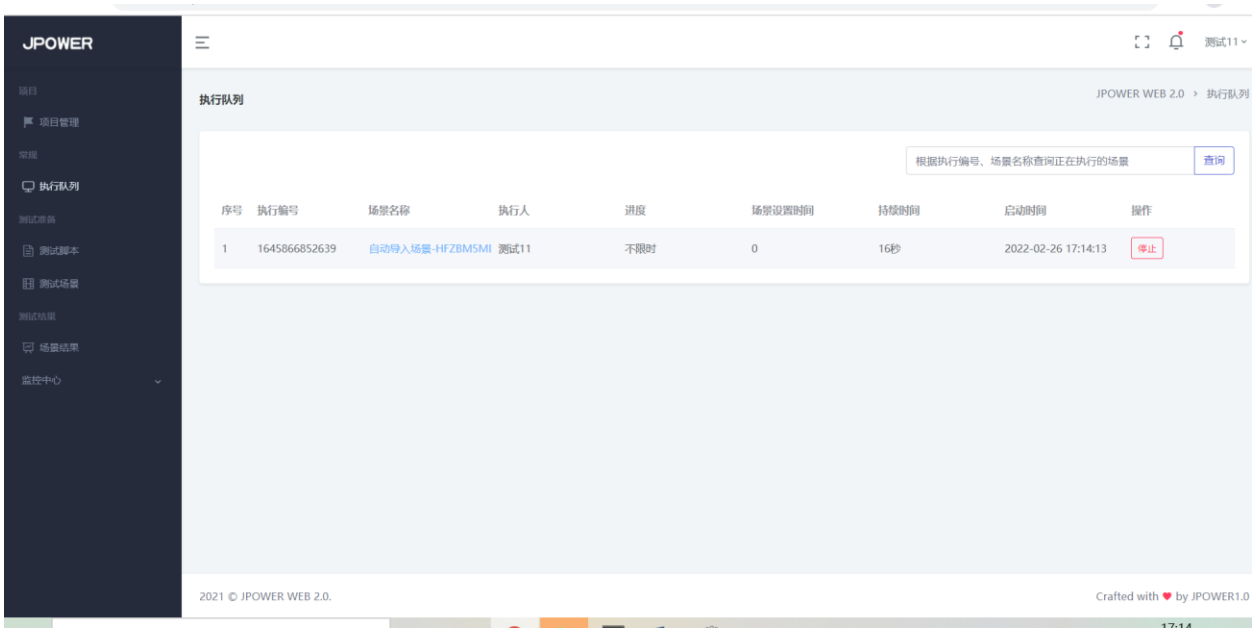


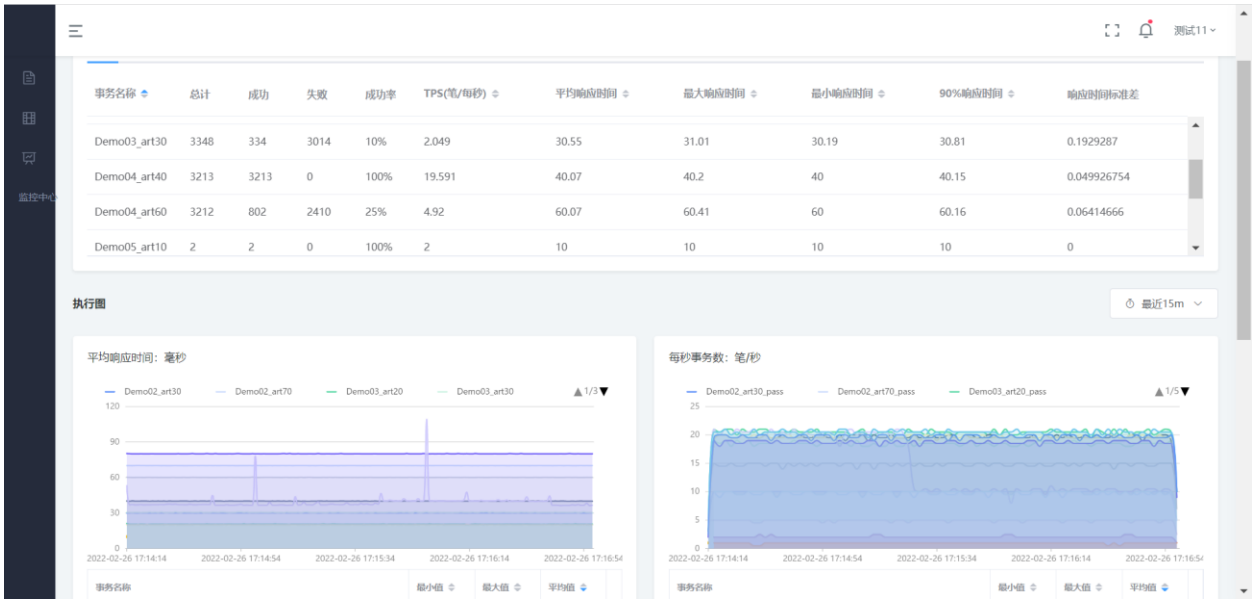
定时【执行计划】根据需要延时场景的执行，新增计划后，后台会自动调起场景运行。具体结果可在场景结果里查询。

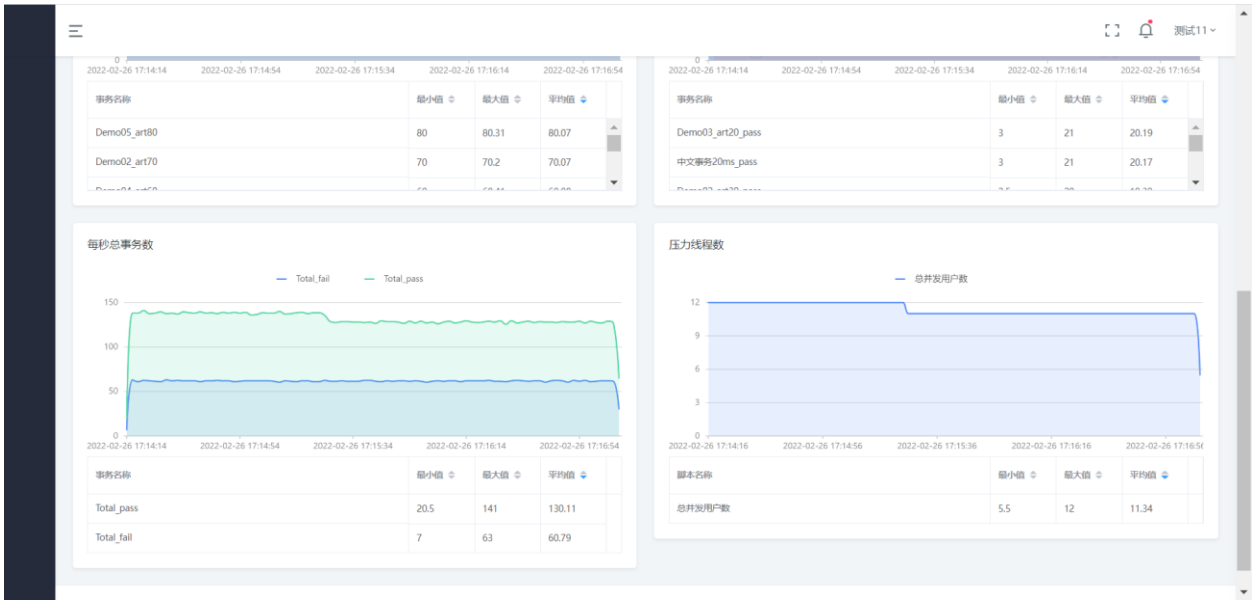


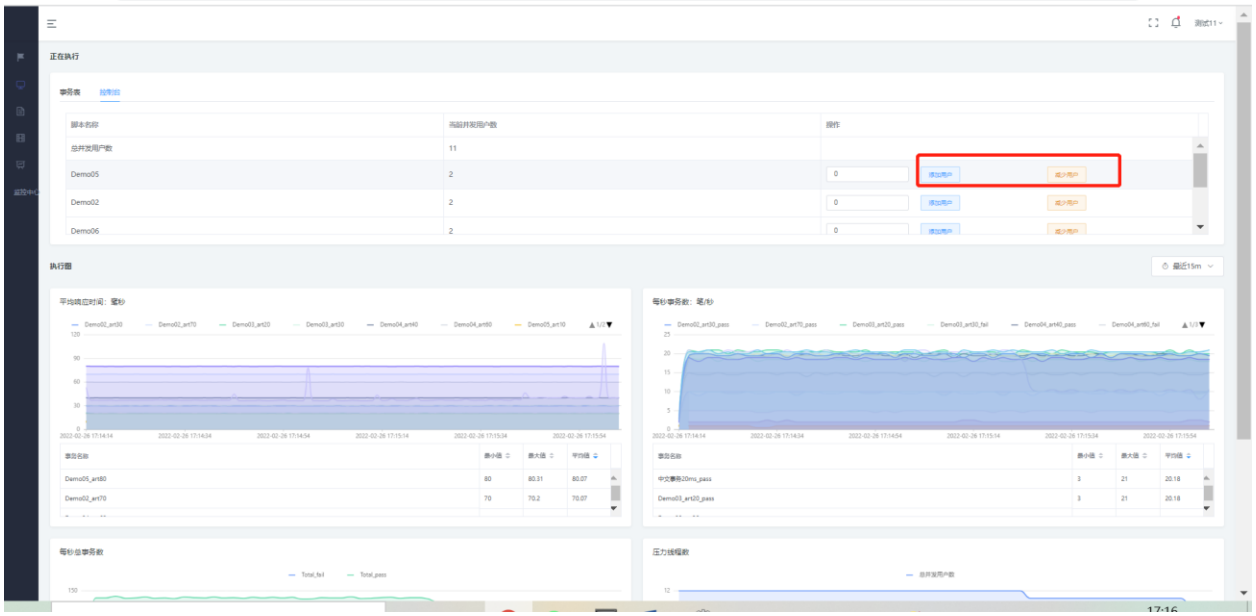
3.6.2. 执行队列

在案例执行期间，可以对 TPS、响应时间、失败率、等进行实时的图像及数据的显示。亦可在运行期间增加减小并发线程数。





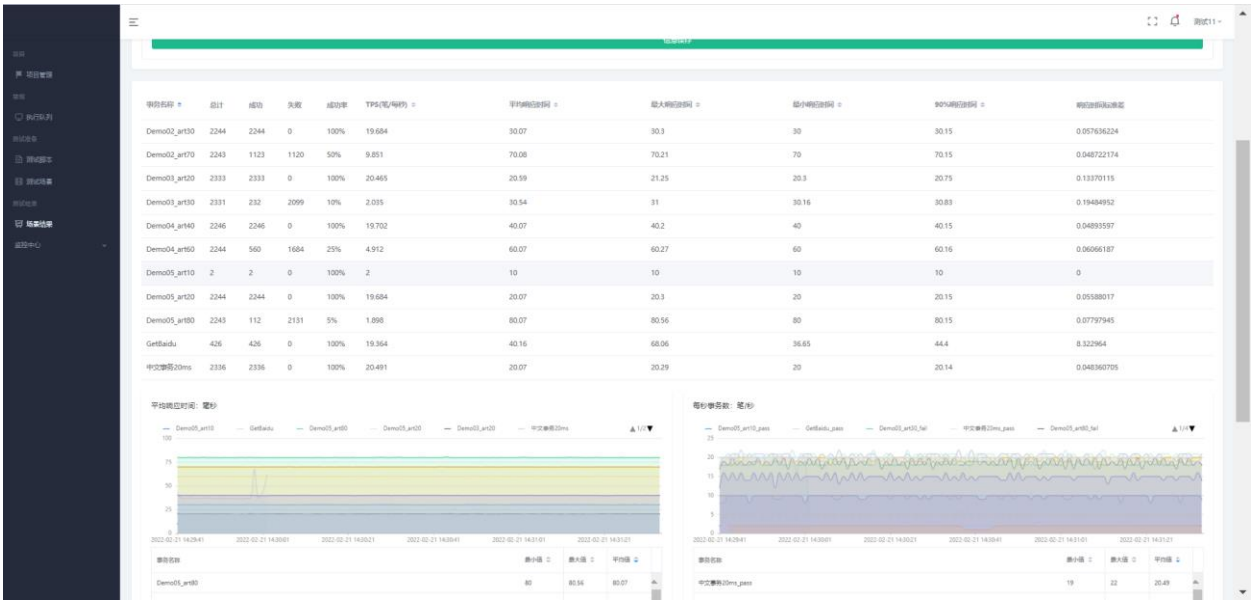


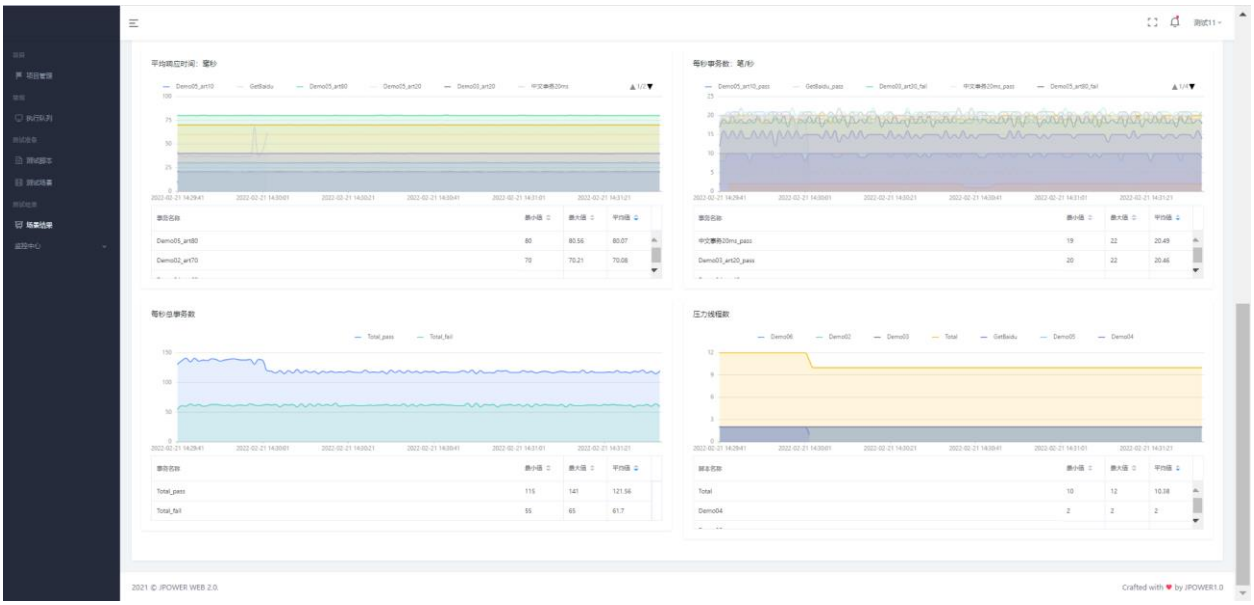


3.5. 场景结果

测试结果主要为测试报告，主要是执行案例时生成的结果数据，包括 TPS、响应时间、成功率等。

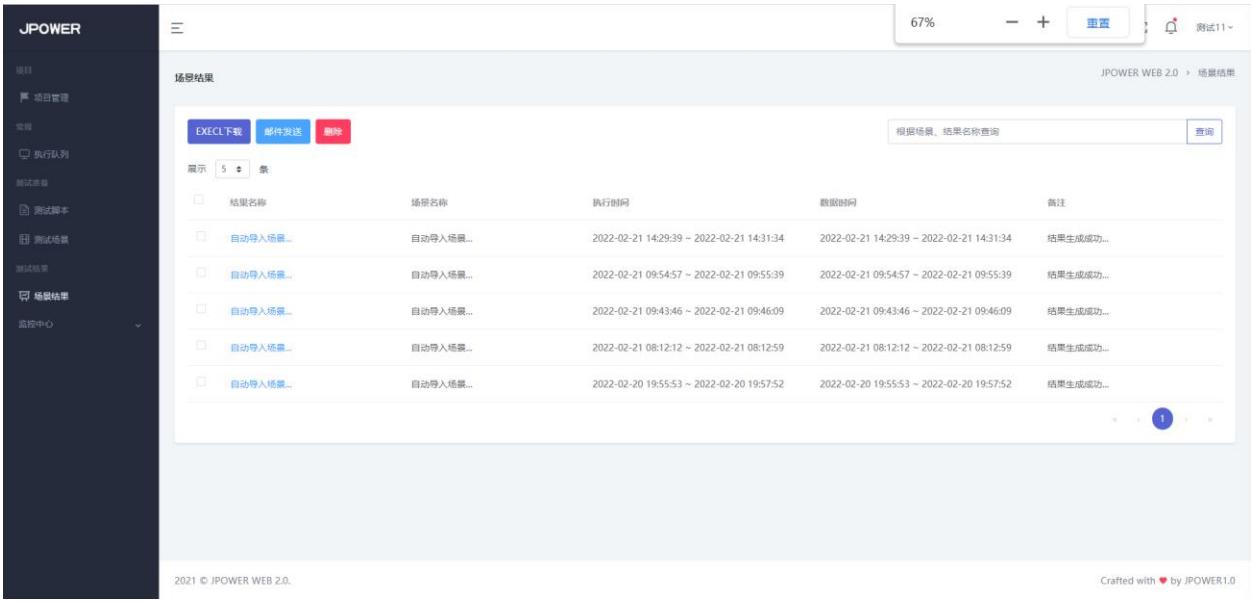
3.7.1. 场景结果





3.7.2. 下载报告

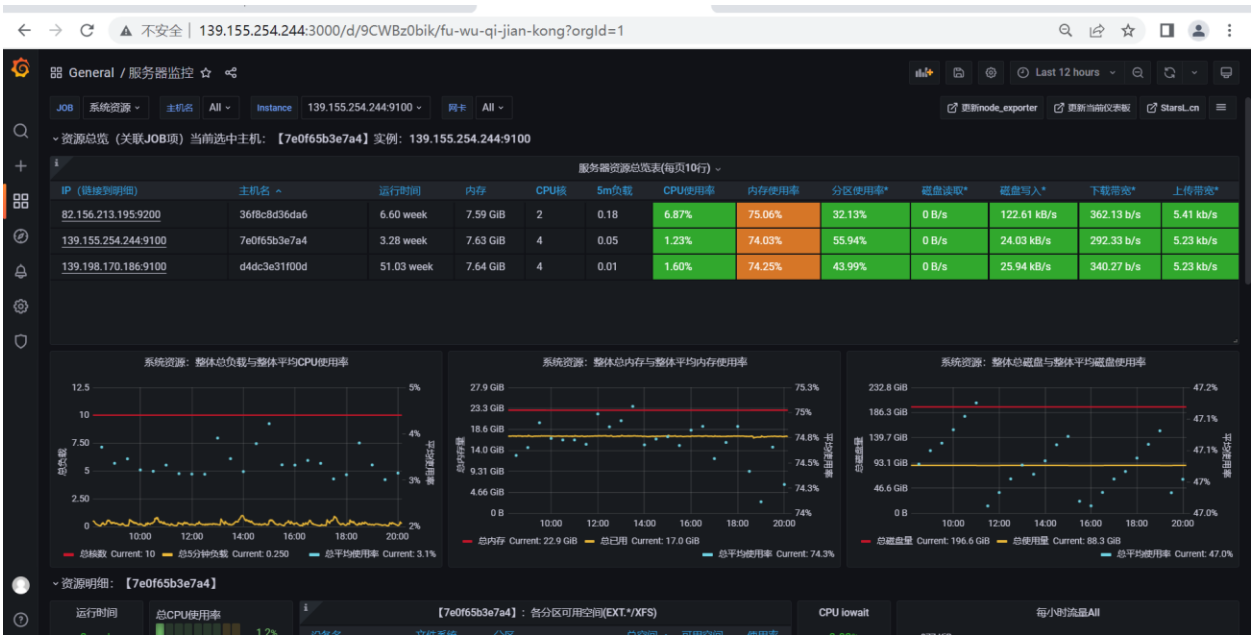
选取下载文件，点击【下载】。



3.6. 监控中心

3.7.3. 监控列表

资源监控，可根据 Grafna 查看，根据需求导入不通模板，进行查看；



4. 录制脚本客户端使用说明

4.1 安装

4.1.1mac 操作系统

打开 SCREEN_WRITER.dmg 安装包，将软件拖拽到应用程序中，即可完成安装。

4.1.2Windows 操作系统

打开 SCREEN_WRITER.exe 安装包，选择安装地址，点击下一步即可完成安装。

4.2 启动

双击点开 SCREEN_WRITER，弹出图 3.2-01 即可完成启动。

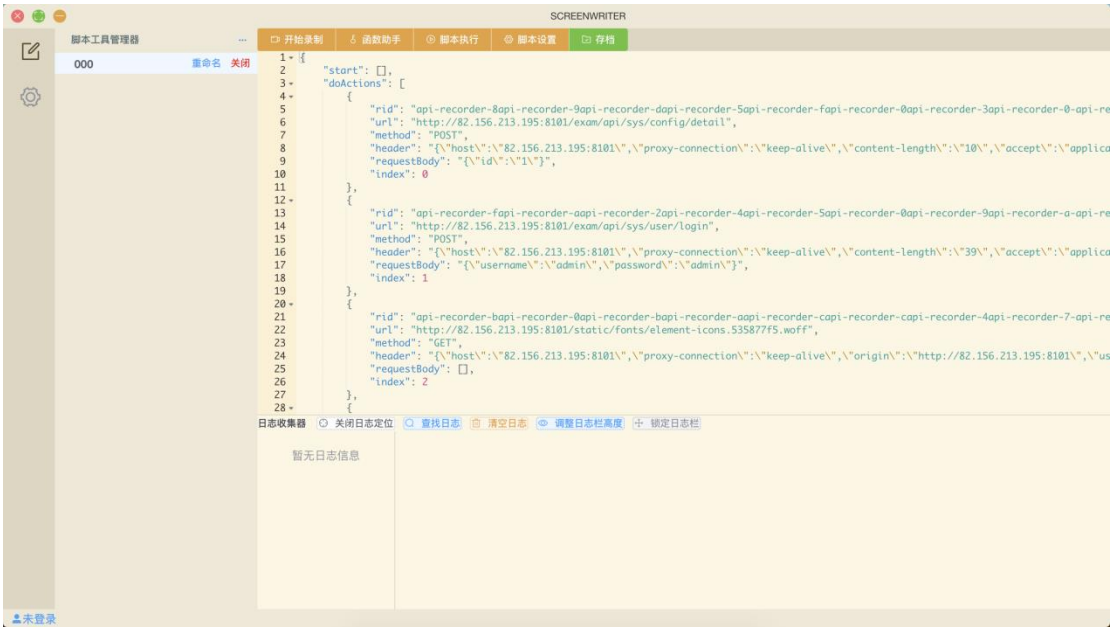
4.3 打开脚本工具管理器

测试人员打开脚本工具管理器可以对脚本进行增加、导入、重命名、删除等操作。

点击左侧脚本开发图标即可唤起脚本工具管理器。

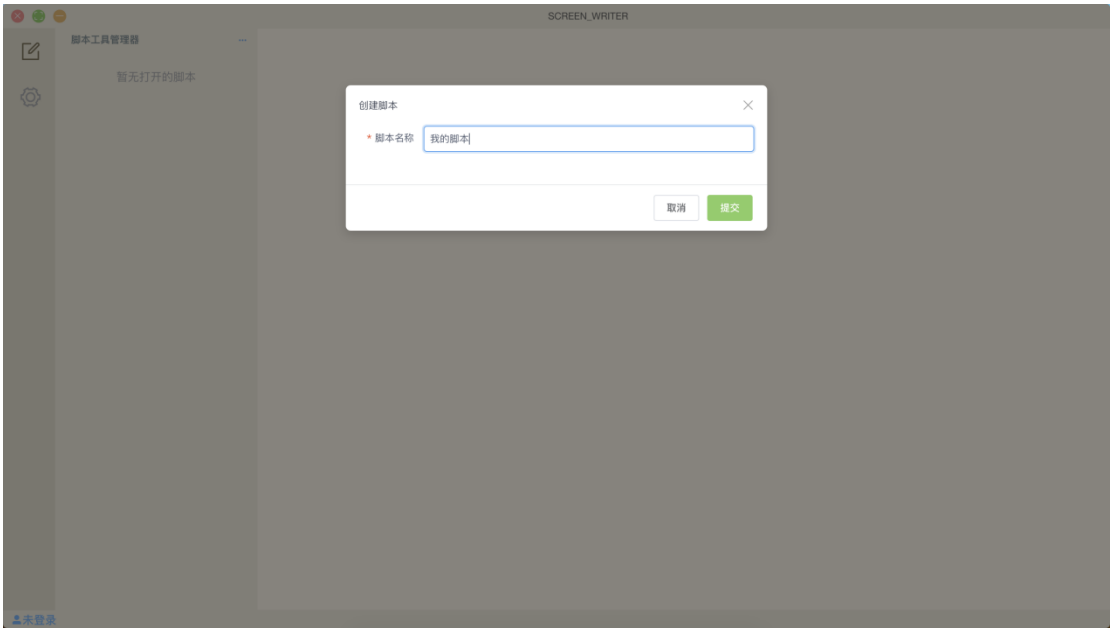
4.4 脚本导入

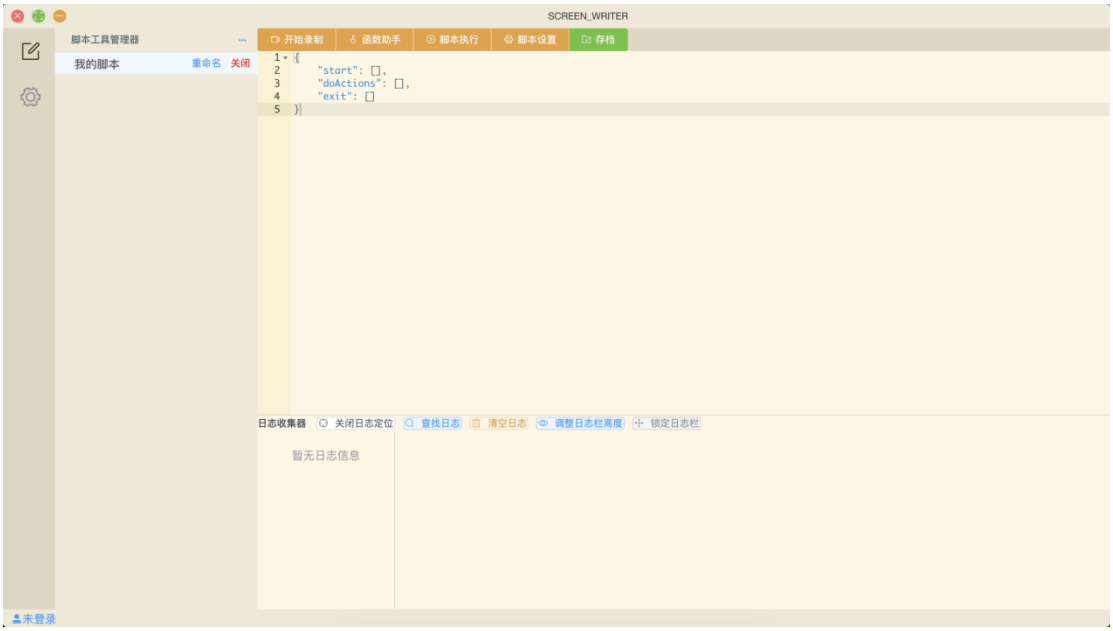
测试人员如有已开发完成的代码，工具支持导入并进行维护。进入脚本工具管理器后，点击脚本工具管理器右上角的【...】按钮，可以唤出创建脚本和导入脚本的弹窗。如图 3.3.1-01 图。点击【导入脚本】按钮即可唤出系统的文件选择器，选择需要导入的脚本，即可在脚本编辑区看到导入的脚本和脚本工具管理器的脚本名称如图。



4.5 创建脚本

测试人员对 PC 端系统或 WEB 端系统进行测试前，通常进行脚本的创建。为了工具的简洁使用，创建脚本仅需要输入脚本名称见图 3.3.2-01。创建成功后的脚本，将在右侧的脚本编辑区域展示出默认自带的脚本默认内容。默认内容不建议修改，但如需对接其他测试工具，可以根据不同的测试工具提供的接口进行脚本初始化内容进行修改。脚本编辑区域以及脚本默认内容见图

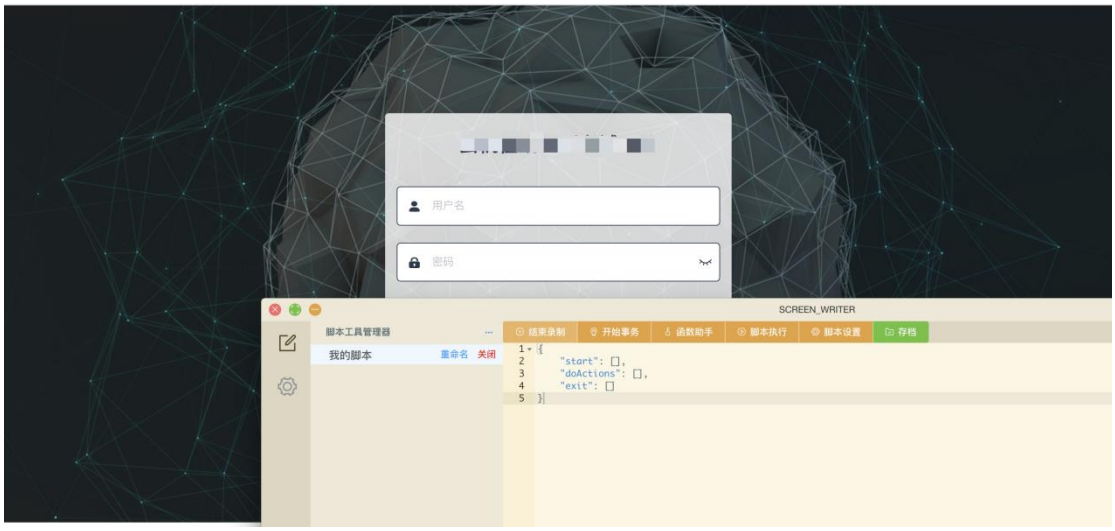




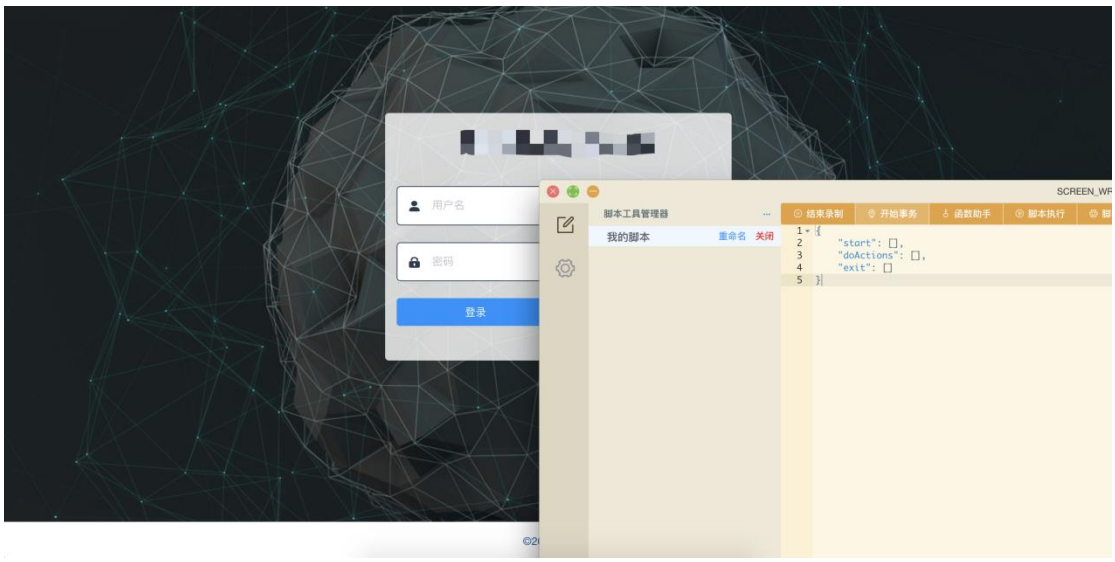
4. 4 脚本录制

为了便捷测试人员进行性能测试以及接口测试，我们主要推荐使用脚本录制功能。由于我们对网络底层进行细致的监听，所有的网络层都加以分类。以 WEB 端为例，我们仅对浏览器类型的网络进行监听，做到完全无干扰录制并生成 JSON 代码，正因为如此，我们不区分浏览器，不限浏览器版本号并且不限操作系统，只要可以完成安装即可进行录制，录制后的信息将完全展示在脚本编辑区域内，做到完全模仿用户使用的操作。

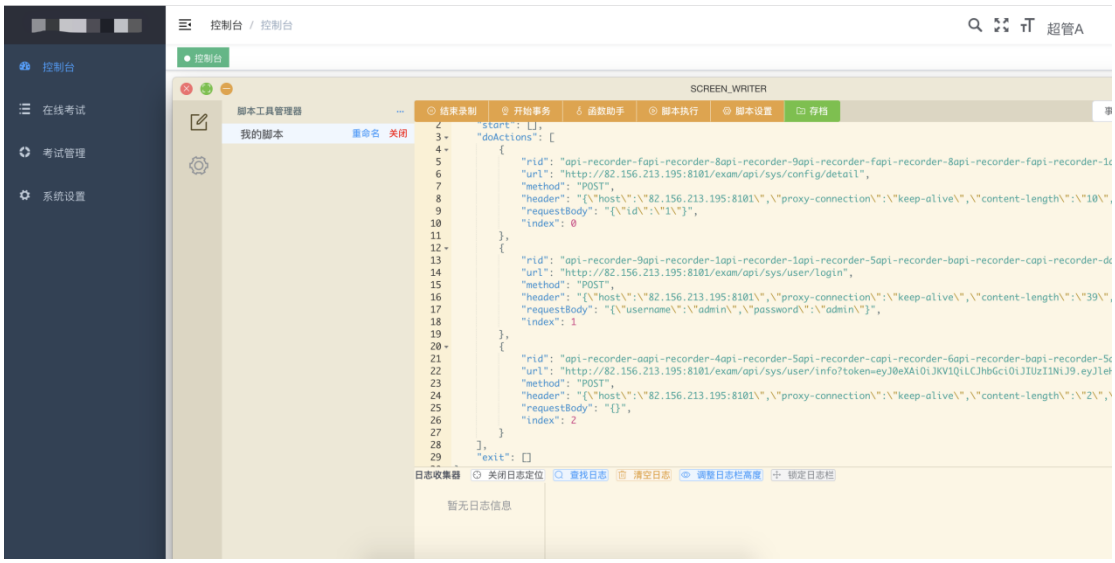
点击【开始录制】即唤出【录制设置】弹窗，见图。其中【浏览器地址】可以输入浏览器的快捷简称也可以输入浏览器的地址，【测试地址】输入需要录制的地址，【录制模块】包含开始模块、事务模块，结束模块，这三个模块分别对应脚本编辑区域的 start、doAction、exit 三段脚本代码。



点击【开始录制】即可根据【测试地址】调用起来浏览器，见图



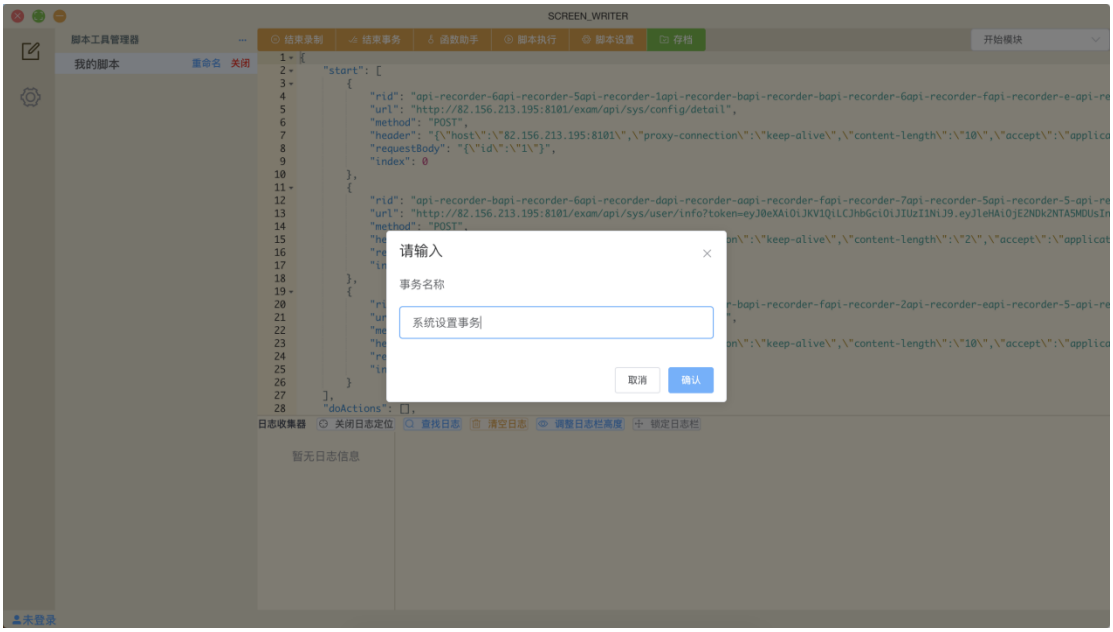
在录制过程中，可以使用【开始事务】与【结束事务】功能，【事务】功能见本文模块、见图

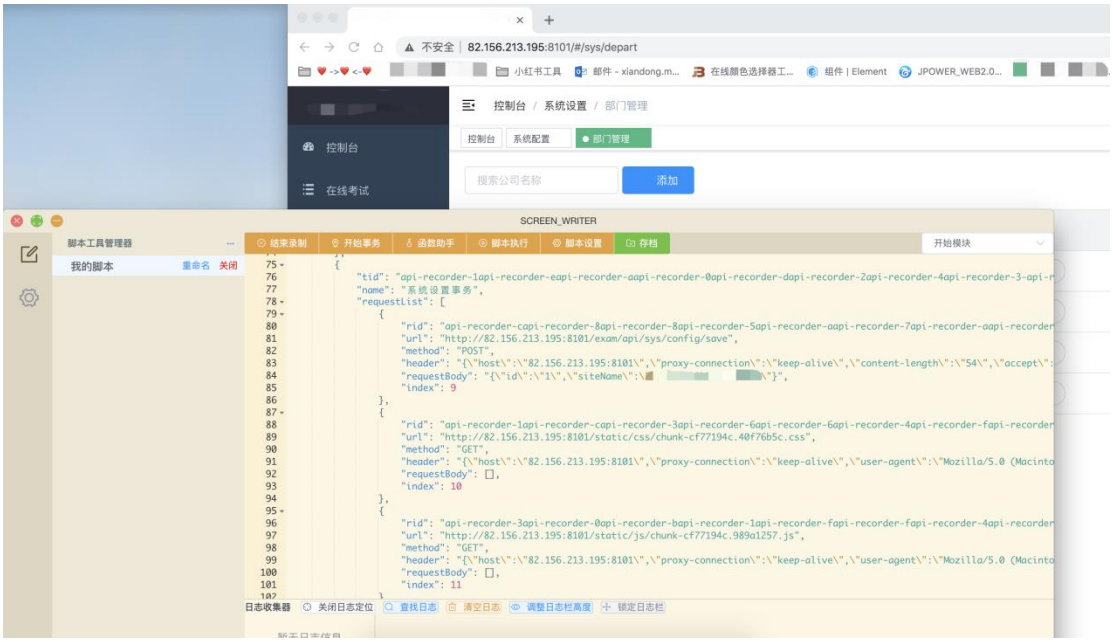


录制功能使用完成，即可点击【结束录制】即可结束录制功能，同时也结束了对网络层的监听，见图

4.5 脚本录制-事务模块

性能测试通常以创建事务进行统计事务的 TPS、接口测试也可以以事务进行统计一系列动作是否完成。事务不仅是一笔交易，一系列操作或一个请求都可以是一个事务。添加事务与结束事务也使用非常简洁的操作，仅需要点击【开始事务】输入事务名称，见图。结束事务无需任何输入，直接点击结束事务即可





4.6 日志收集器

测试人员进行脚本执行后，日志收集器进行针对脚本进行统计收集。收集的内容主要以：请求的路径、请求信息(请求网址、请求方法、请求头、请求体)、响应信息(响应代码、响应头、响应体)。针对收集的内容进行查找，脚本对应定位等功能。请求信息见图 4.6-01、4.6-02，响应信息见图 4.6-03、4.6-04，查找日志见图 4.6-05，日志定位见图 4.6-06，清空日志见图 4.6-07。

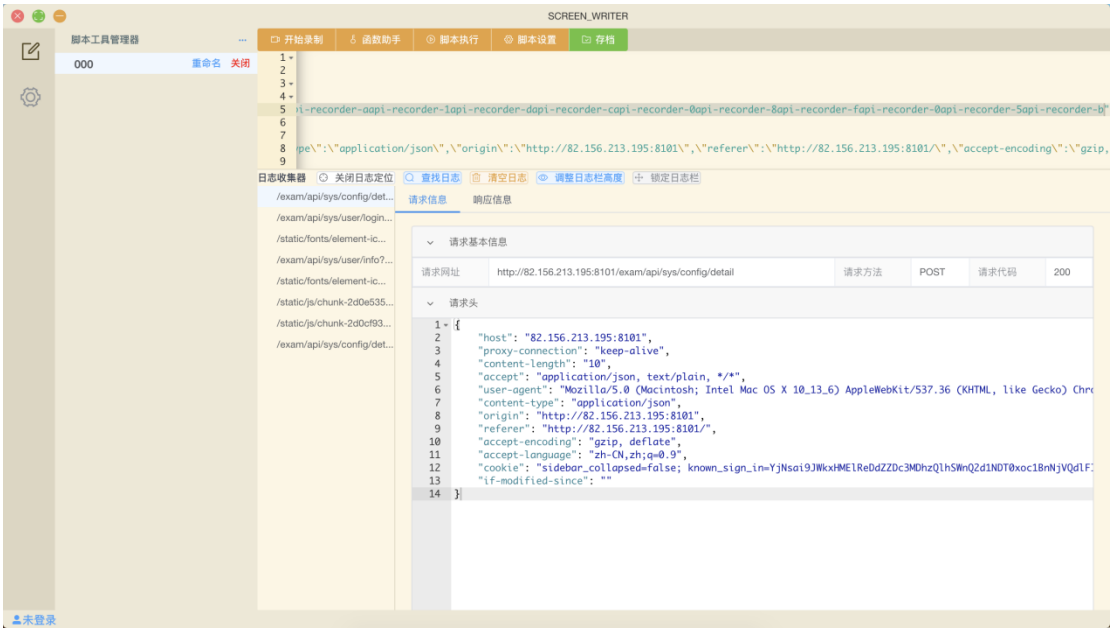
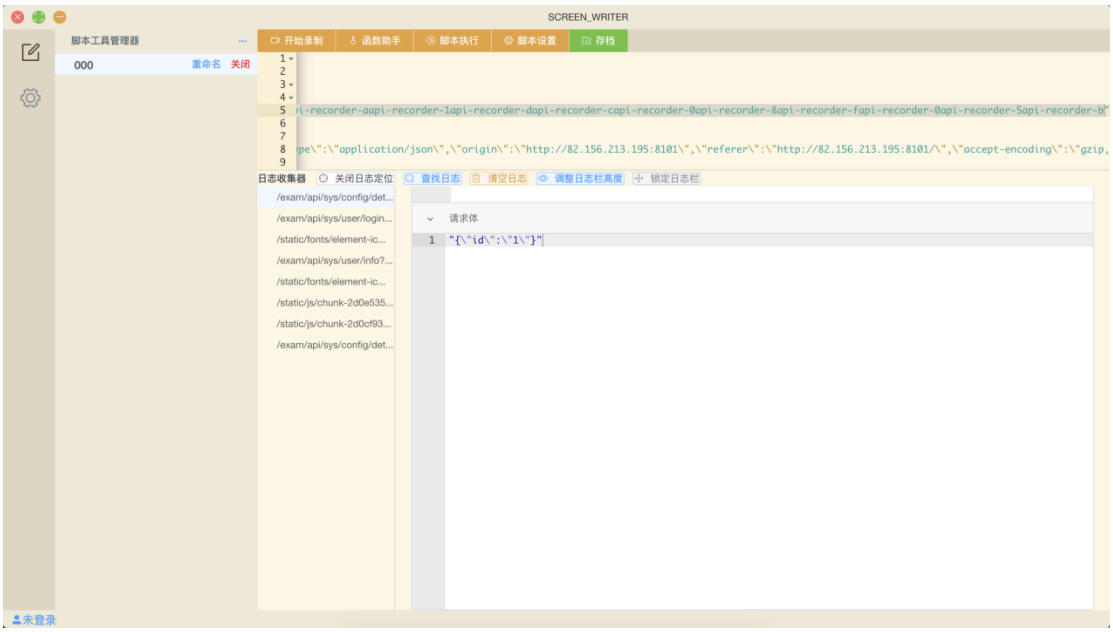


图 4.6-01



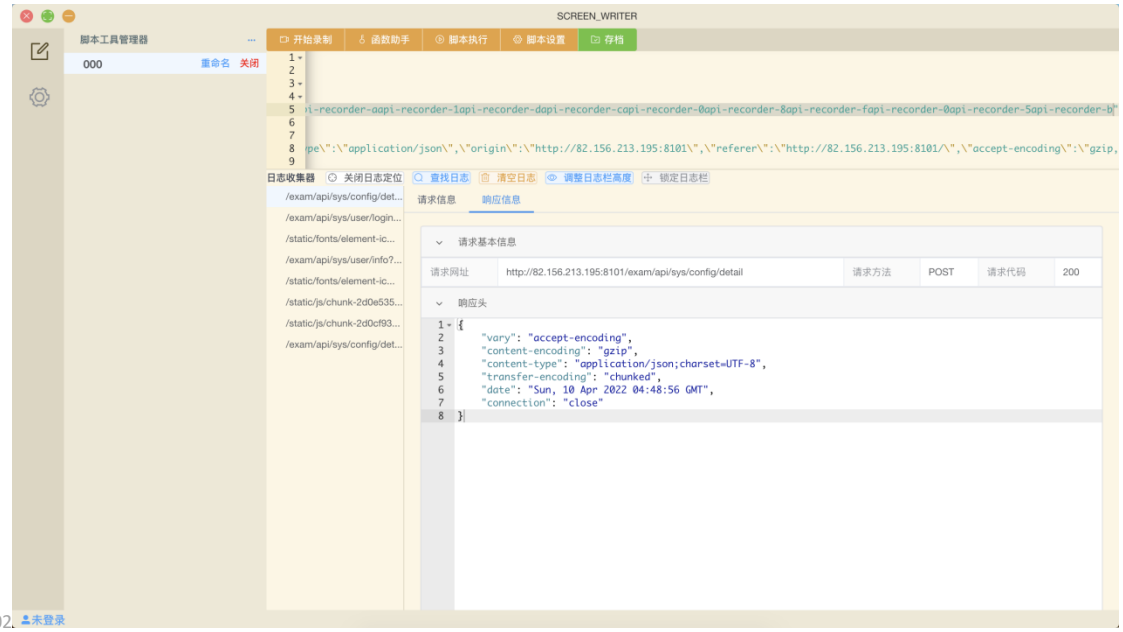


图 3.7-02

图 4.6-03

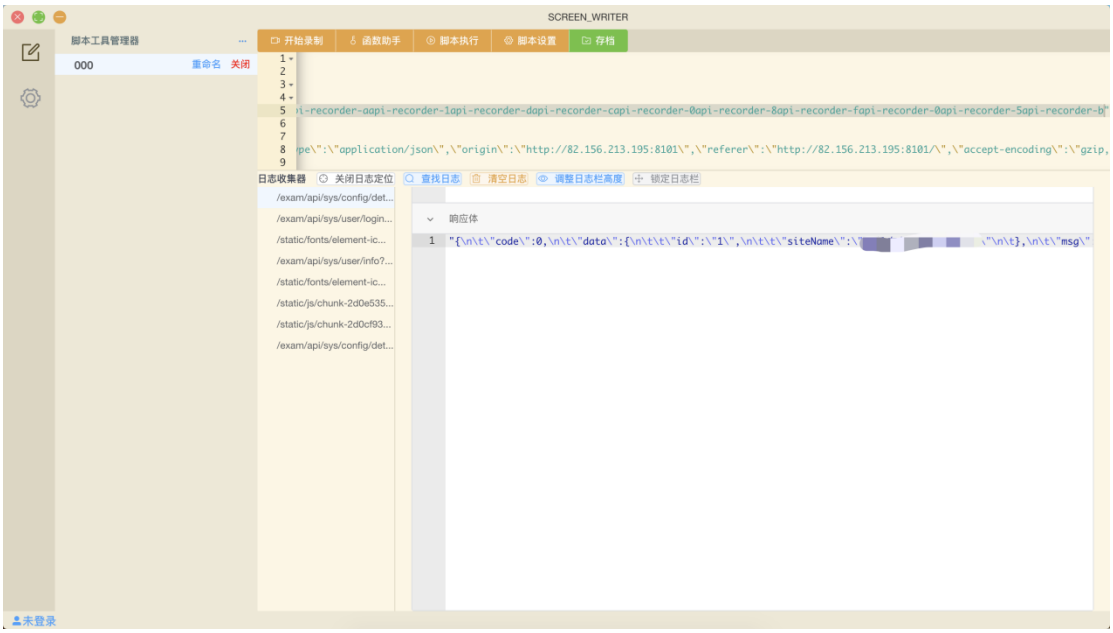


图 4.6-04

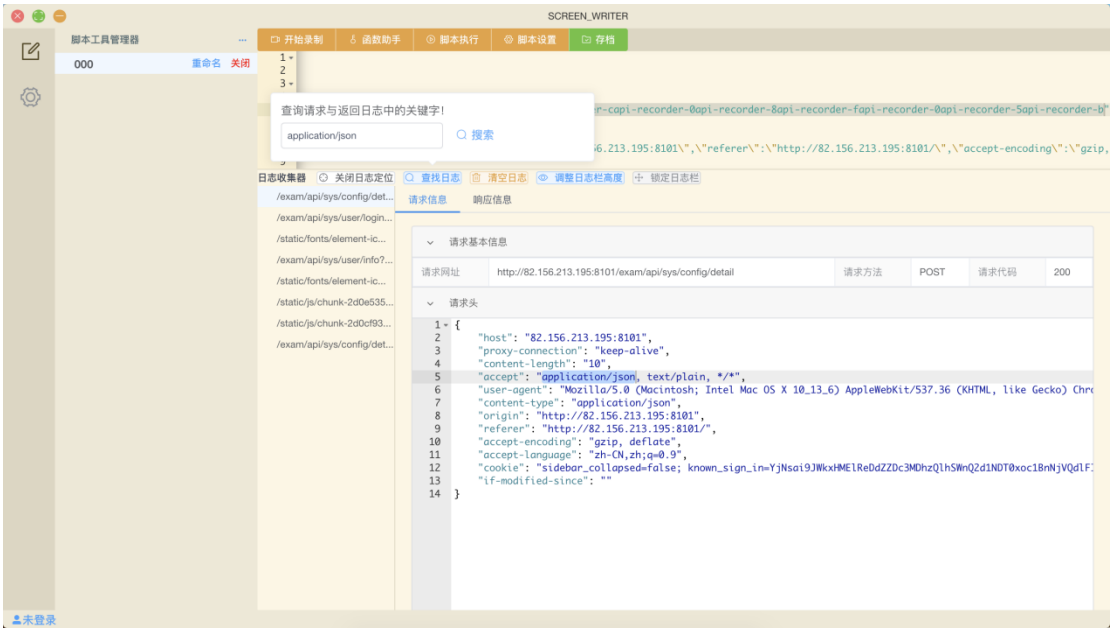


图 4.6-05

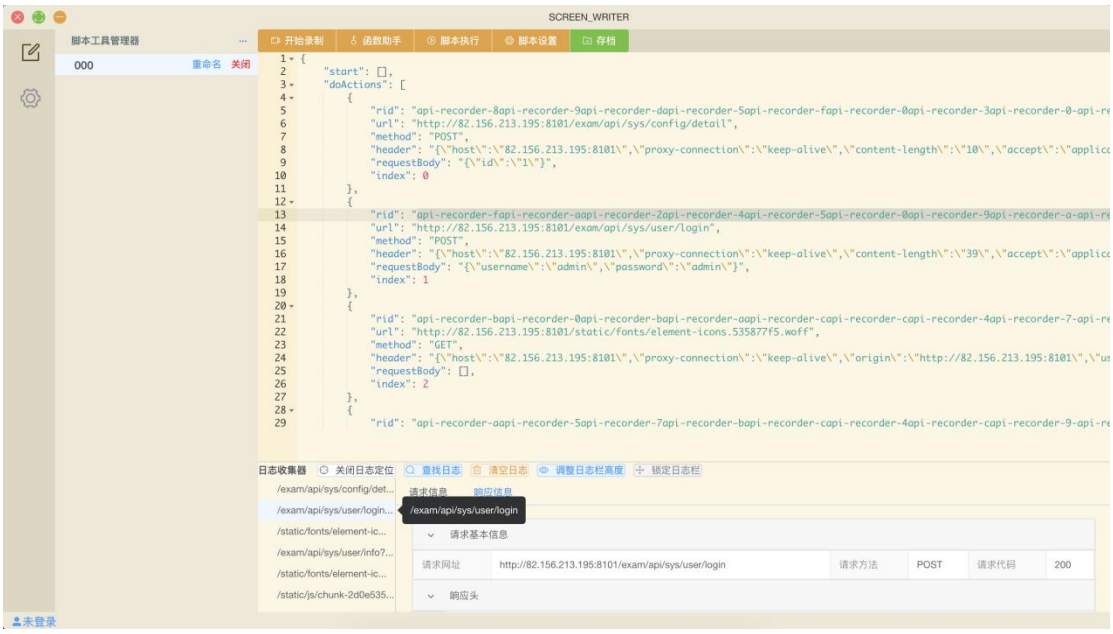


图 4.6-06

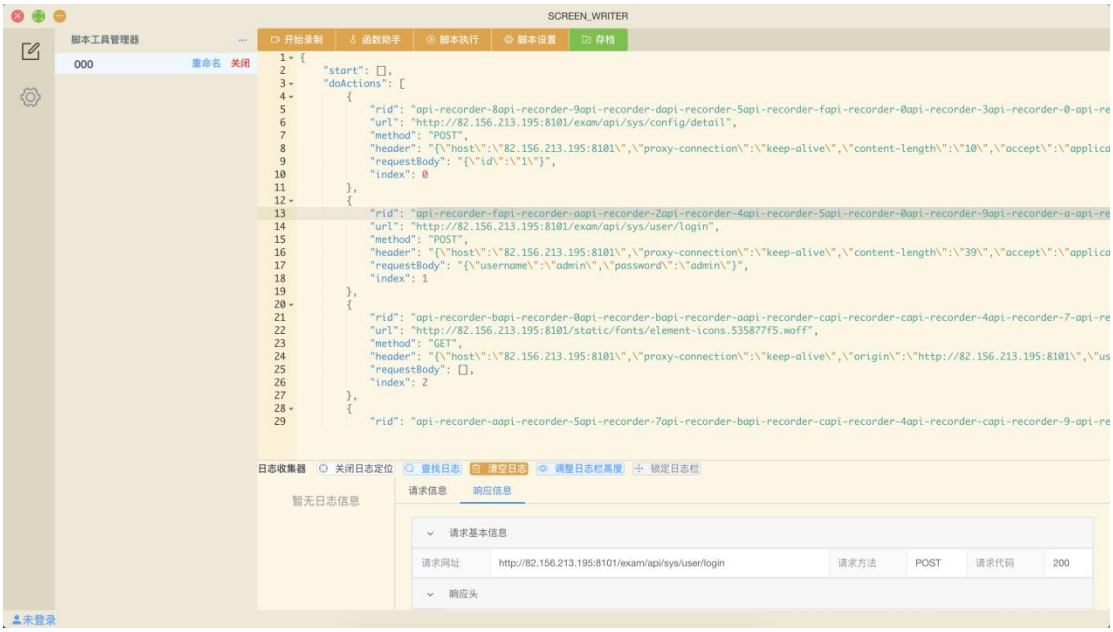


图 4.6-07

4.7 存档

测试人员进行调试后，确定可以保存脚本，点击【存档】按钮后即可唤出系统的文件选择路径，文件名称默认为脚本工具管理器的脚本命名，见图 4.7-01。

保存成功后吗，可以使用任何文本工具打开，脚本为 JSON 文件，其中包含脚本信息，见图 4.7-02。非常不建议使用文本工具编辑脚本。

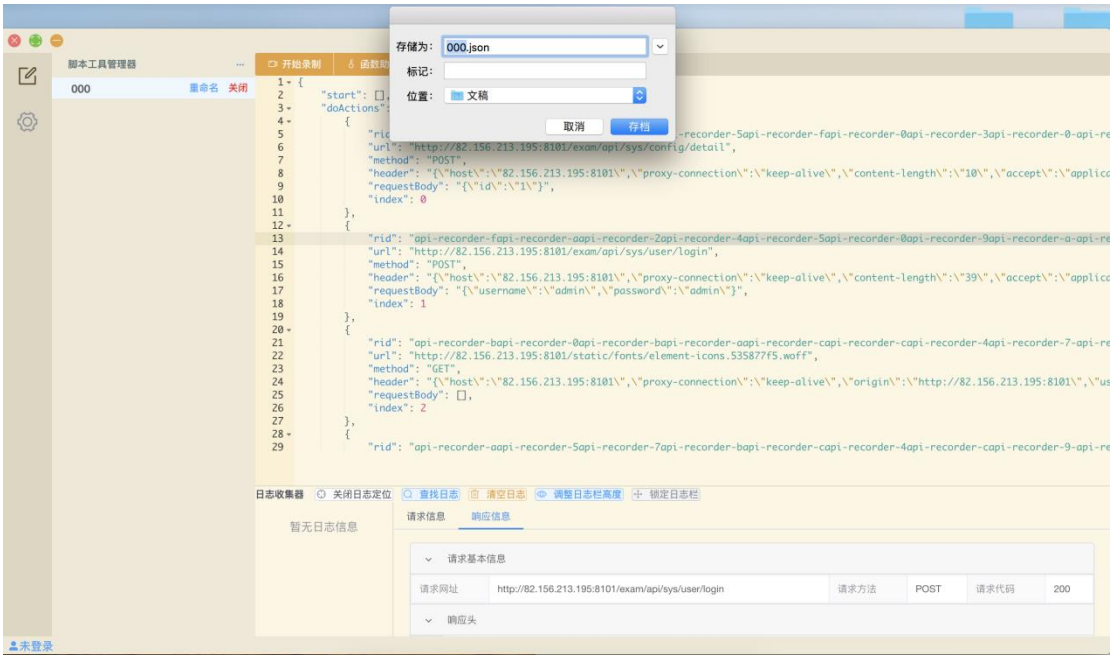


图 4.7-01

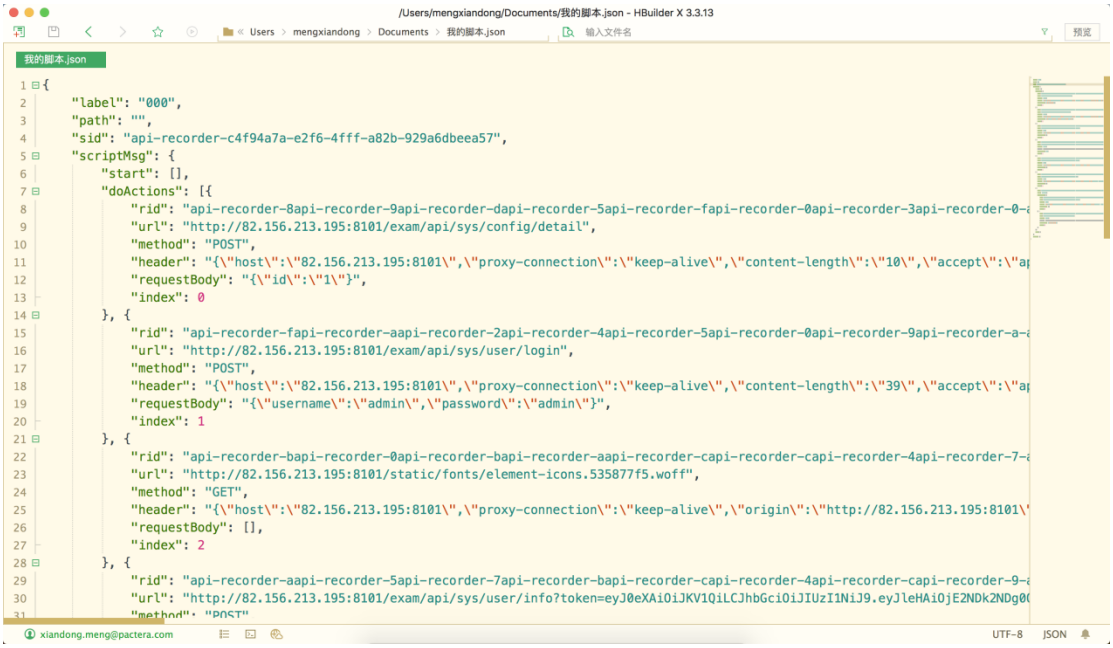


图 4.7-02

5. 接口脚本开发说明

5.1java 脚本开发 sdk 准备

本平台使用 java 开发，框架支持的脚本为 java 类，即理论上能使用 java 实现的操作，都可以使用本平台运行，故本平台脚本没有明确的"协议"之分。

本平台所有的 java 脚本均须继承 com.jpower.pgs.sdk.template.VUSample 类（我们叫它为虚拟用户类），并覆写以下三个方法：init()、doActions()、exit()。

脚本整体结构如下：

```
// 参数注解告诉平台需要参数化的数据

@Parameters(values = {"param1","param2"})

// 事务注解，非必需

@Actions(actions = {"事务名称 1", "事务名称 2")})
```

```
public class YourTest extends VUSample {
    private String action1 = "事务名称 1";
    private String action1 = "事务名称 2";

    @Override
    public void init() throws Exception {
        //初始化
    }

    @Override
    public void doActions() throws Exception {
        try{
            // 顺序获取参数: 在该用户分得的参数范围内顺序取
            String param1 = getOrdinalValFrom("param1");
            // 随机获取参数: 在该用户分得的参数范围内随机取
            String param2 = getRandomValFrom("param2");

            //业务迭代
            startAction(action1);
            //do something
            endAction(action1,true);

            startAction(action2);
            //do something
            endAction(action2,true);
```

```
            }catch(Exception e){
                //异常计数
                addNaNFailAction(e.getClass().getSimpleName());
                logError("异常信息",e);
            }
        }

        @Override
        public void exit() throws Exception {
            //资源释放
        }
    }
```

客户端 sdk 环境封装在 client-sdk-x.x.x.jar 中，如果需要人工开发脚本，请导入此 jar 包到你的 IDE 中：

若使用 maven 项目，可以进行本地安装（其中 x.x.x 表示 sdk 版本号）：

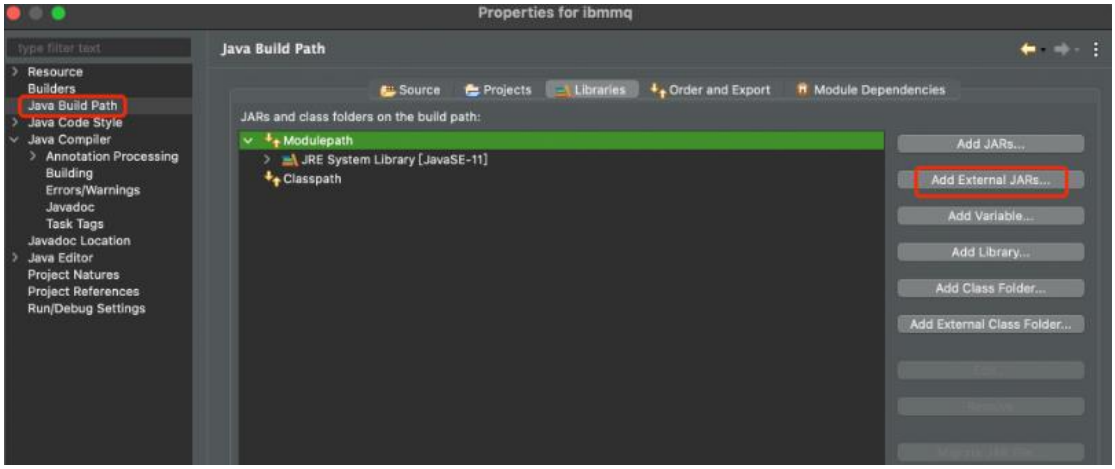
```
mvn install:install-file -DgroupId=com.jpower.pgs -DartifactId=client-sdk -Dversion=x.x.x -Dpackaging=jar -Dfile=client-sdk-x.x.x.jar
```

然后在项目 pom.xml 中加入：

```
<dependency>
    <groupId>com.jpower.pgs</groupId>
```

```
    <artifactId>client-sdk</artifactId>
    <version>x.x.x</version>
    <scope>provided</scope>
</dependency>
```

若使用 eclipse 开发工具（非 maven 方式），则可以直接在项目 lib 中添加该包：



5.2 命名规范

脚本规范请参考 java 开发规范，脚本命名不能使用数字或以数字开头，对一个项目的若干脚本和事务的命名应做到纵横有序。

如项目 ProjectDemo 有 3 个脚本，每个脚本有若干事务，第一步均为登录，其命名推荐方式如下（本文叫纵横有序脚本命名法）：

T01_Script: T01_01_login T01_02_actionName T01_03_actionName

T02_Script: T02_01_login T02_02_actionName T02_03_actionName

T03_Script: T03_01_login T03_02_actionName T03_03_actionName

【login\actionName 可使用中文，Script 取有业务意义的英文名或按开发方接口名称】

📌 提示：若不使用上述方式命名，多个不同脚本中有相同的事务名称，则该事务的统计结果的是这多个脚本中该同名事务的所有结果。

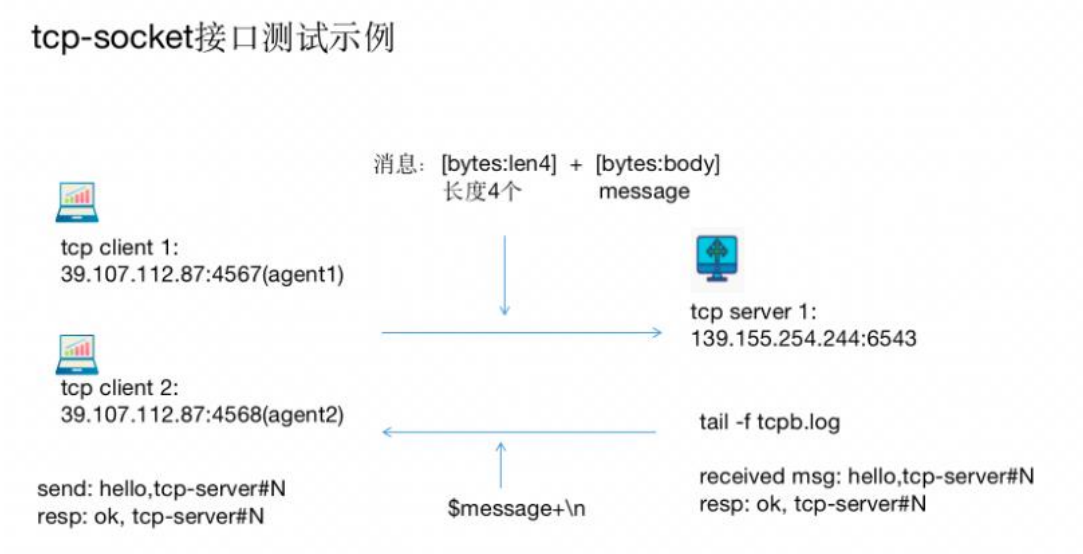
📌 提示：平台统计结果中不显示脚本属性，只显示脚本中的事务指标，故采用上述命名方式可以区分出不同脚本的事务。

5.3java tcp-socket 客户端样例

场景业务简述如下：

服务端：tcp-socket 服务端接收二进制消息，消息前 4 个字节是头信息：表示消息体的长度，服务端处理过程是接收 4 个字节，获得消息体的长度，再读取该长度的字节数数据，随后将接收到的 hello 替换成 ok，发送到客户端，响应消息简单的以字符"\n"表示响应结束。

客户端：按服务端接口规定，先计算要发送信息的长度，将长度数字转为 4 个字节二进制数据发送，再发送具体的消息内容的二进制，客户端可以收或不收服务端返回的信息；



客户端 java 代码实现如下

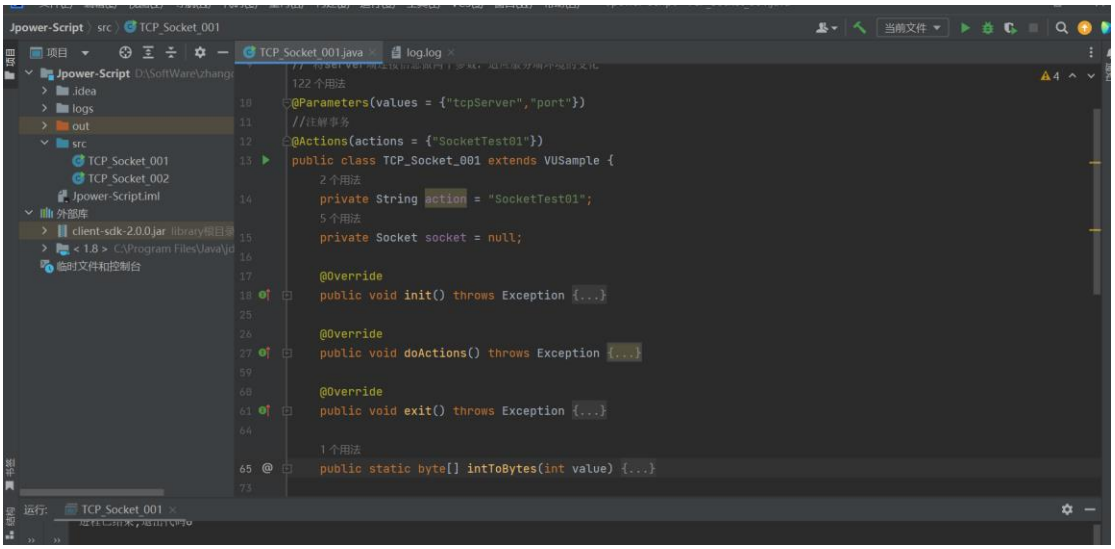
```
import com.jpower.pgs.sdk.annotation.Parameters;
import com.jpower.pgs.sdk.template.VUSample;

import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

// 将 server 端连接信息做两个参数，适应服务端环境的变化
@Parameters(values = {"tcpServer","port"})
public class SocketTest01 extends VUSample {
    private String action = "SocketTest01";
    private Socket socket = null;
```

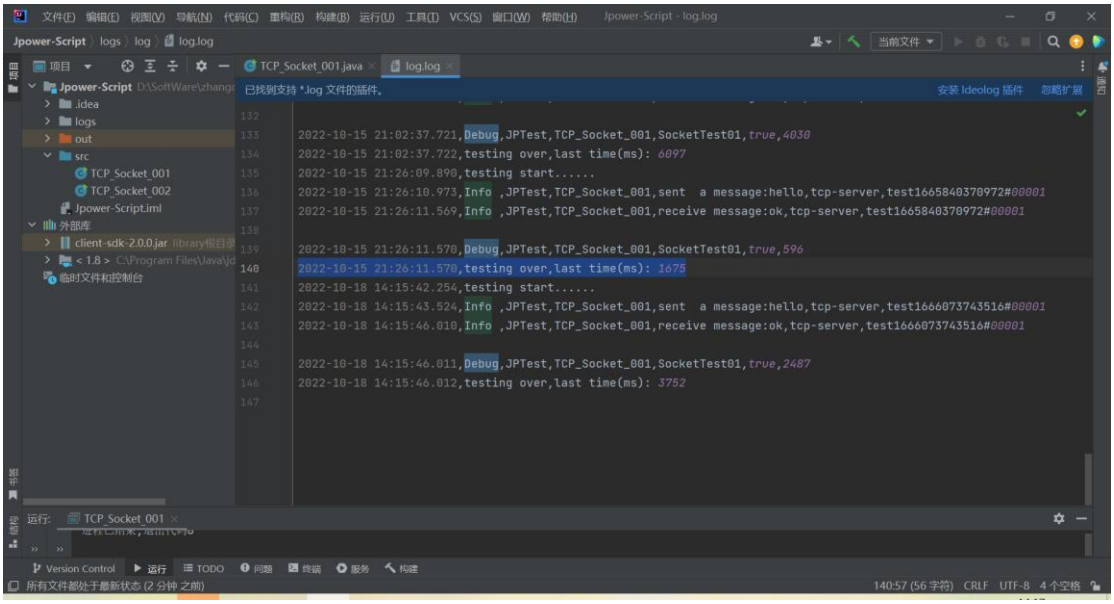
```
    @Override
    public void init() throws Exception {
        socket = new Socket(getOrdinalValFrom("tcpServer"),
                Integer.parseInt(getOrdinalValFrom("port")));
        socket.setSoTimeout(5000);
    }

    @Override
    public void doActions() throws Exception {
        OutputStream os = socket.getOutputStream();
        InputStream is = socket.getInputStream();

        String data = "hello,tcp-
server,test"+System.currentTimeMillis()+"#"+getLoopCount(5);
        int len = data.getBytes("utf-8").length;
        byte[] len2Bts = intToBytes(len);

        startAction(action);
        os.write(len2Bts);
        os.write(data.getBytes("utf-8"));
        os.flush();
        logInfo("sent  a message:"+ data);

        byte[] receive = new byte[128];
        while(true){
```

```
            int k = is.read(receive);
            if(-1 ==k){
                break;
            }
            // 收到\n 为一次交互的结束
            if(receive[k-1] == '\n'){
                String resp = new String(receive,0,k);
                logInfo("receive message:"+ resp);
                break;
            }
        }

        endAction(action,true);
    }

    @Override
    public void exit() throws Exception {
        socket.close();
    }

    public static byte[] intToBytes( int value ) {
        byte[] bts = new byte[4];
        bts[3] =  (byte) ((value>>24) & 0xFF);
        bts[2] =  (byte) ((value>>16) & 0xFF);
        bts[1] =  (byte) ((value>>8) & 0xFF);
```

```
        bts[0] =  (byte) (value & 0xFF);
        return bts;
    }

    // main 方法用来在开发环境业务调试，在平台上场景执行是非必需
    public static void main(String[] args) throws Exception {
        SocketTest01 test = new SocketTest01();
        test.setArgument("tcpServer","139.155.254.244");
        test.setArgument("port","6543");
        test.executeTest(5);
    }
```

样例 iead 展示：





、

5.4java http 客户端样例

使用 sdk 包提供的 httpclient:

使用 OKHttp 库

5.5 第三方类库的使用

java 测试类的开发中经常会使用第三方提供的类库 jar 包。本平台实现第三方类库的加载使用可以通过至少两种方式实现：

脚本中调试通过后，在 web 页面找到对应的项目，进入项目管理页面的“插件库”上传到平台；

与测试类一起打包（使用于导入包体积比较小的情况）；

📌 当导入包比较多时，可以通过 jar 命令将多个包合并成一个包后上传到项目插件库

```
操作参考：
step1.  mkdir jartmp
step2.  将要合并到a.jar、b.jar拷贝到jartmp
step3.  进入jartmp,解压jar包：
        jar -xvf a.jar
        jar -xvf b.jar
step4.  删除jartmp中的a.jar、b.jar
step5.  压缩所有class文件到新包：
        jar -cvfM all-dependency.jar .   #最后有个点，表示当前路径下所有文件
```

📌 注：脚本的 jar 包在哪个项目，依赖的第三方 jar 包必须上传到对应的项目。插件库是以项目分隔的。

6. 常见问题

#1. 自动部署失败

可进行的操作和原因：

手动 ssh 连通测试：确认 ssh 登录的用户名和密码输入没有错误；

权限检查：确认启动用户有磁盘读写权限；

确认 java 版和 java_home 正确；

如果环境网络较慢或返回信息是 timeout，可以通过以下 api 设置更大的超时时间：

get 请求：

curl --location --request GET 'http://$master-ip:port/api/master/deploy-help/set-agent-login-timeout/${timeout}' \

--header 'Authorization:eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJiNzliYzUxNi1lOWIwLTRjMzktOTM4NiO5M2ZlYTFhMDYxNWQiLCJyZWFsTmFtZSI6Iuadjuafrz IiLCJleHAiOjE2MzU3NDkONDgsInVzZXJuYW1lIjoibGlrZTEifQ.p7Yk6k8z4vaLPUhSUgXHTQMtvZa4dDbQZC-XnmjGWoU'

#2. 场景启动成功但是执行队列没有场景执行

可进行的操作和原因：

查看场景执行历史中的错误信息，确认 agent 是否在线；

确认场景的参数文件是否可以正常读取：到脚本中看参数是否可以正常浏览；

#3. 场景启动进入执行队列看不到实时信息

可进行的操作和原因：

agent 无法写入数据到 influxdb：请检查 influxdb 是否正常工作，存储是否可以写入，influxdb 网络是否超时，连接用户名、密码、token 等信息是否正确；

确认 data-center 是否正常工作，并请检查 data-center 配置的 influxdb 连接用户名、密码、token 等信息是否正确；

可以使用以下接口确认 influxdb 是否可以写入：

浏览器执行： http://$masterIp/api/master/deploy-help/influx-test

或(linux): curl -X GET 'http://$masterIp/api/master/deploy-help/influx-test'

[$masterIp 为实际环境中 master 的域名或 ip 地址]

{"code":200,"message":"操作成功","data":"bucket write success."}

#4. 场景执行中无法增加并发

可进行的操作和原因：

参数检查：参数使用平分方式，后添加的用户无可分配参数；

参数检查：参数使用预分块方式，后添加的用户无可分配参数；

agent 网络检查：可能是执行操作时，master 到 agent 的网络超时；

#5. 没有生成场景结果

可进行的操作和原因：

同问题 3；

重跑场景；

#6. 场景执行时，控制台没有数据

可进行的操作和原因：

master 和 influx 所在系统时间不同步：确保 master 和 influx 系统时间误差不超过 2s；建议整个应用各系统设置统一的时间同步服务器；