顶象设备指纹介绍

一、产品简介

通过用户上网设备软硬件指纹信息，为用户生成网络空间的身份标识。配合顶象风控系统使用，可有效对抗设备伪造、自动注册、羊毛党等恶意行为。

二、名词解释

名称	释义
AppId	验证码公钥，长度为32位字符串，验证码唯一标识。开通服务后可在设备指纹的二级菜单"应用管理"中获取
AppSecret	验证码私钥，长度为32位字符串，与验证码公钥对应，开通服务后可在设备指纹的二级菜单"应用管理"中获取，请妥善保管，勿泄漏给他人
token	设备的相关信息，用来获得设备指纹ConstID使用
ConstID	设备指纹ID，通过采集设备的硬件信息生成的设备唯一标识
用户前端	Web端或集成SDK的Android端、iOS端
用户后端	指企业的后台服务器

三、设备指纹前端接入

- **支持Web接入**，支持IE8+、Chrome、Firefox、360浏览器、QQ浏览器等主流浏览器及Android、iOS上的内嵌Webview。如何引用JS请见Web接入章节
- **支持Android接入**，如何获取SDK并接入，请见Android接入章节
- **支持iOS接入**，如何获取SDK并接入，请见iOS接入章节
- **支持微信小程序接入**

3.1 Web接入

第一步：引入

在页面 HTML 中引入 `const-id.js`，代码形如：

```
<script src="https://cdn.dingxiang-inc.com/ctu-group/constid-js/index.js">
</script>
```

第二步：生成并使用

页面加载后，初始化设备指纹，需要在 JavaScript 中调用 `_dx.ConstID(options, callback)` 方法获取设备指纹token，代码形如：

```
var options = {
  appId: '【这里填写 AppID】', // 唯一标识，必填
  server: 'https://constid.dingxiang-inc.com/udid/c1', // constId 服务接口，可选
  userId: '【这里填写 userID】' // 用户标识，可选
};

_dx.ConstID(options, function (err, token) {
  if (err) {
    // console.log('error: ' + err);
    return;
  }
  // console.log('const-id token is ' + token);
});
```

同时也支持Promise的用法

```
_dx.ConstID(options).then(function(token) {
  console.log(token)
}).catch(function(err) {
  console.log(err)
})
```

options 字段说明

字段	类型	是否必填	说明
appId	String	是	当前应用的标识
server	String	否	constId 服务接口，可选，如不填，则默认会用云服务接口
scene	String	否	场景标识，例如 `login`、`survey` 等
userId	String	否	业务方的用户唯一标识，例如用户名、用户ID、手机号、Email等
timeout	number	否	超时失败时间，单位为毫秒

PC浏览器兼容

浏览器	最低版本
IE	8
Edge	20
Chrome	60
Safari	11
Firefox	60

浏览器	最低版本
360	10
Sougou	8
QQ	4

移动端浏览器兼容

浏览器	最低版本
Chrome	60
UC	12
QQ	8
Safari	11
原生	安卓4.0及以上

3.2 Android接入

一、 环境要求

条目	说明
开发目标	Android 4.0+
开发环境	Android Studio 3.0.1 或者 Eclipse + ADT
CPU架构	ARM 或者 x86
SDK三方依赖	无

二、 法规要求

根据《工业和信息化部 337号令》的规定，重点对以下四个方面开展规范整治工作。
（一）违规收集用户个人信息方面
（二）违规使用用户个人信息方面
（三）不合理索取用户权限方面
（四）为用户账号注销设置障碍方面
法规地址：http://miit.gov.cn/n1146295/n1652858/n1652930/n3757020/c7506353/content.html

其中SDK涉及到第一条：收集个人信息（包括唯一设备识别码、网络设备硬件地址等信息）。
法规规定，所采集的数据项目需在隐私政策中明确声明，在客户不同意隐私政策的情况下，不允许进行采集。
且申请授权需与场景相关，请根据实际情况做出合理调整。

解决方式

需要客户在集成了SDK的app中增加《隐私政策声明》，来规避风险。并在客户同意隐私政策后，进行sdk的调用。

隐私政策（拟，可根据实际情况进行修改）

为了识别设备/账号异常状态，我们将会接受并记录您所使用的设备相关信息（包括设备型号、操作系统、设备设置、唯一设备标识符（IMEI码）、网络设备硬件地址（MAC）、设备环境等软硬件特征信息，设备所在位置相关信息（包括您授权的GPS位置以及WLAN接入点、蓝牙和基站等信息）

三、 集成SDK

3.1 下载SDK

点击下载SDK

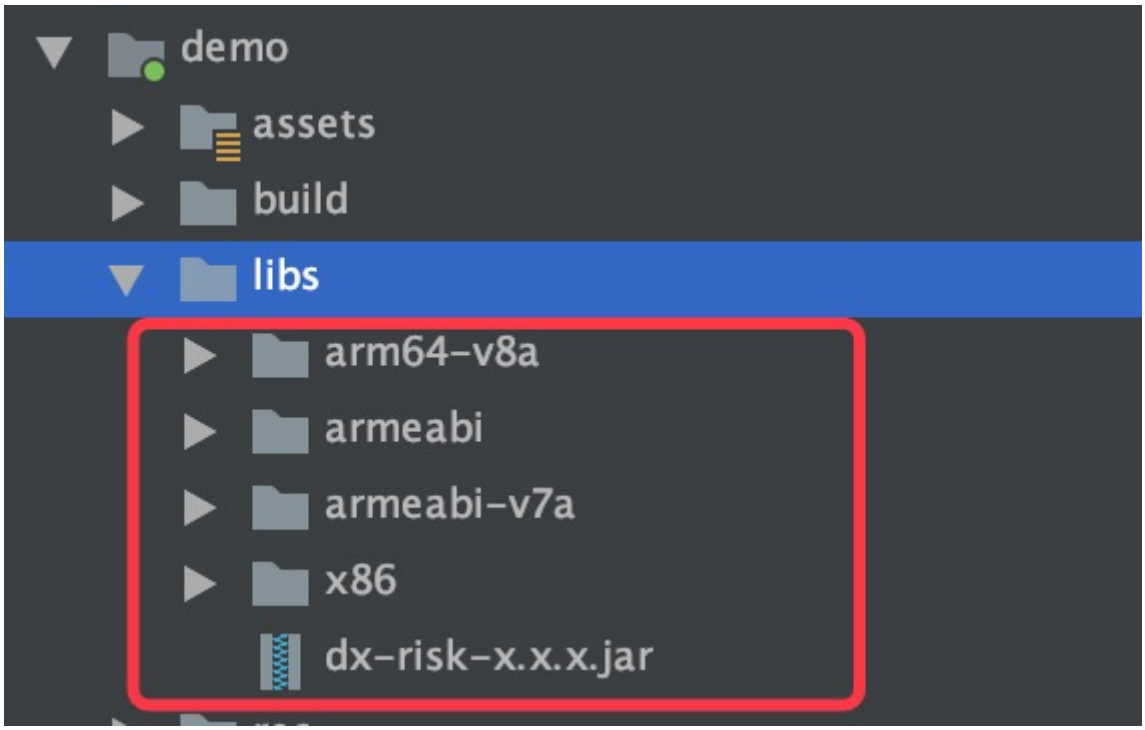
点击下载demo(仅做代码配置演示使用，其中appId请在顶象后台申请，SDK需要替换为链接中下载的SDK)

3.2 Android Studio 集成

SDK包集成内容：

- `libs`文件夹下的jar和so
- `assets`文件夹下的配置文件

3.2.1 把`libs`下`jar`和`so`库放到相应模块的libs目录下，`assets`下的文件放置到项目`assets`下



3.2.2 在该Module的build.gradle中如下配置:

```
android{
    sourceSets {
```

```
        main {
            jniLibs.srcDirs = ['libs']
            assets.srcDirs = ['assets']
        }
    }

    packagingOptions {
        doNotStrip "**/libDX*.so"
    }
}

repositories{
    flatDir{
        dirs 'libs'
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation files('libs/dx-risk-x.x.x.jar')
}
```

3.3 添加SDK所需权限

```
<!-- 必选-默认申请 -->
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>

<!-- 可选-6.0或以上需动态申请 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

3.4 Proguard混淆配置

```
-dontwarn com.dx.mobile.**
-dontwarn *.com.dx.mobile.**
-dontwarn *.com.mobile.strenc.**
-keep class com.dx.mobile.risk.**{*;}
-keep class com.security.inner.**{*;}
-keep class *.com.dx.mobile.**{*;}
-keep class *.com.mobile.strenc.**{*;}
```

3.5 API 6.0或以上动态权限申请说明

需要动态申请权限如下:

```
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.READ_EXTERNAL_STORAGE
android.permission.READ_PHONE_STATE
android.permission.ACCESS_COARSE_LOCATION
android.permission.ACCESS_FINE_LOCATION
```

动态申请代码实例(Activity下):

```
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_demo);

   // API 23或以上的动态申请权限
   if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
       String[] permissionArray = {
               "android.permission.ACCESS_COARSE_LOCATION",
               "android.permission.ACCESS_FINE_LOCATION",
               "android.permission.WRITE_EXTERNAL_STORAGE",
               "android.permission.READ_EXTERNAL_STORAGE",
               "android.permission.READ_PHONE_STATE",
       };
       this.requestPermissions(permissionArray, 1);
   }

}


@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions,
int[] grantResults) {
   super.onRequestPermissionsResult(requestCode, permissions, grantResults);
   // start getToken
   new Thread(new Runnable() {
       @Override
       public void run() {
           HashMap<String, String> params = new HashMap<String, String>();
           String token = DXRisk.getToken("appid", params);
       }
   }).start();

}
```

四. 接口使用说明

4.1 方法和参数说明

功能描述

采集端的设备指纹信息，上传至风控后台，再由风控后台返回token。 该API为耗时操作，因此必须在非主线程上调用，否则会抛异常。

方法说明

`DXRisk.java` 该类是DxRisk SDK的风控组件接口，负责采集本地信息并返回用户前端token。

初始化setup

`SDK`使用前必须调用先`setup`，`setup`主要用于数据/环境初始化，一般在`Application`的`onCreate`下调用：

```
/**
* 初始化参数，环境
* @param context
* @return
*/
public static boolean setup(Context context)
```

PS：下列两种方式获取token在网络通畅的情况下没有任何的不同。

常规Token

```
/**
 * @return token 通常返回长度为40的字符串。在网络卡顿或不通的情况下，返回4-5k的字符串。
 * @throws DXRiskErrorException 如在主线程调用本API，或者appId为空等等，则会抛出该异常
 */
public static String getToken(String appId, HashMap<String, String> paramsMap)
throws DXRiskErrorException
```

精简Token

获取轻量级Token可获取的设备信息信息远少于getToken()，可能会造成在判断设备是否有风险时出现较大误差，请谨慎使用。

```
/**
 `* @return token 通常返回长度为40的字符串。在网络卡顿或不通的情况下，返回1k的字符串。
 `* @throws DXRiskErrorException 如在主线程调用本API，或者appId为空等等，则会抛出该异
常
 */
public static String getLightToken(String appId, HashMap<String, String>
paramsMap) throws DXRiskErrorException
```

4.2 使用示例

4.2.1 初始化setup

建议`Application.onCreate`下调用

```
@Override
public void onCreate() {
  super.onCreate();
  // 环境初始化
  DXRisk.setup(this);
}
```

4.2.2 获取token

整个过程由于是耗时操作，必须要在非主线程上执行，否则会crash

```
new Thread(){

    @Override
    public void run() {
      /* 私有化配置 */
      HashMap<String, String> params = new HashMap<String, String>();
      // 私有化部署服务端url
      params.put(DXRisk.KEY_URL, "https://constid.dingxiang-inc.com");
      // 开启线上数据备份
      params.put(DXRisk.KEY_BACKUP, DXRisk.VALUE_ENABLE_BACKUP);
      // 设置请求token超时时长ms，不设置默认为500ms
      parsms.put(DXRisk.KEY_DELAY_MS_TIME, "2000");

      // 开通服务后可在实时风险决策的菜单获取
      String appId = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
       // 获取设备指纹token
      final String token = DXRisk.getToken(appId, paramsMap);

      // TODO 把token通过Post请求，传到用户后端
  }
}.start();
```

4.3 异常说明

在获取token过程中，如果因为网络超时或者加解密失败，该接口有可能会返回为null，同时会输出tag为DXRISK的错误信息，具体描述如下：

```
DXRISK_REQUEST_NETWORK_ERR            -1001
DXRISK_REQUEST_DECRYPT_ERR            -1002
DXRISK_REQUEST_UNCOMPRESS_ERR         -1003
```

```
DXRISK_REQUEST_RESPONSE_EMPTY_ERR     -1004
DXRISK_REQUEST_DATA_PARSE_ERR         -1005
DXRISK_REQUEST_DIRTY_DATA_ERR         -1006
DXRISK_CONST_ID_EMPTY                 -1007
```

如果出现上述错误信息，请联系顶象技术人员。

3.3 iOS接入

一、环境需求

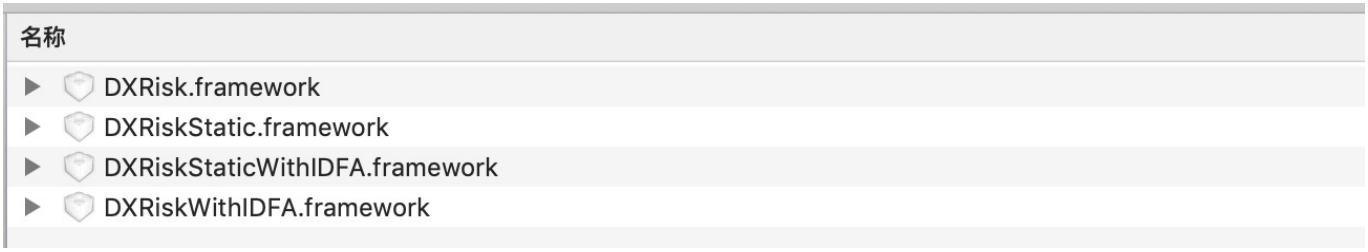
条目	说明
兼容平台	iOS 8.0+
开发环境	XCode 4.0 +
CPU架构	armv7, arm64, i386, x86_64
SDK依赖	libz, libresolv, libc++ , SystemConfiguration.framework , CoreLocation.framework , CoreTelephony.framework

二、集成SDK

2.1 下载SDK

点击下载SDK，SDK的目录结构如下：

点击下载集成demo



- dx-risk-iOS-x.x.x-xxxxxxx目录 DXRisk sdk
  - DXRisk.framework 不带idfa获取逻辑的Dynamic Library Framework
  - DXRiskWithIDFA.framework 带idfa获取逻辑的Dynamic Library Framework
  - DXRiskStatic.framework 不带idfa获取逻辑的Static Library Framework
  - DXRiskStaticWithIDFA.framework 带idfa获取逻辑的Static Library Framework

2.2 将SDK接入XCode

2.2.1 导入Framework

DXRisk.framework，DXRiskWithIDFA.framework，DXRiskStatic.framework，DXRiskStaticWithIDFA.framework`其中之一直接拖入工程目录中，或者右击总文件夹添加文件。

- 如果App中包含广告相关的功能，则选择DXRiskWithIDFA.framework 或者 DXRiskStaticWithIDFA.framework，该版本可以提供更精准的token
- 如果没有广告，获取idfa可能导致拒绝上架，此时请选择DXRisk.framework 或者 DXRiskStatic.framework

2.2.2 添加FrameWork到工程

若在项目中添加DXRisk.framework或者DXRiskWithIDFA.framework其中之一，选择Target -> General，在Frameworks,Libraries,and Embedded Content中，将DXRisk.framework或者 DXRiskWithIDFA.framework 对应的 Embed 切换到Embed & Sign。如下图：

若在项目中添加DXRiskStatic.framework或者DXRiskStaticWithIDFA.framework其中之一，需要在Build Settings -> Other Linker Flags 设置 -ObjC 如下图：

![4A23453213E696EC12C2AE2A1070EA0C.jpg](https://cdn.dingxiang-inc.com/images/293/29386b91-2918-4b02-8f30-da3ac1547197.jpg)

2.2.3 配置打包脚本

以下的操作仅限导入DXRisk.framework ， DXRiskWithIDFA.framework动态库

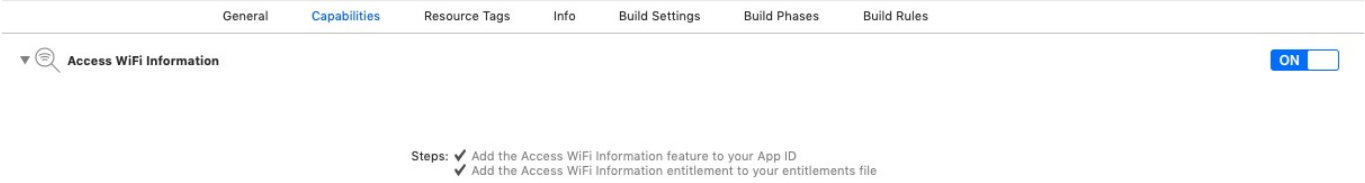
此步骤主要是解决上传Store架构不符合的问题，如项目中已配置过Carthage或有其他相关的打包Framework调整脚本，可略过此步自行调整 选择Target -> Build Phases，点击+按钮，添加如下脚本：

```
APP_PATH="${TARGET_BUILD_DIR}/${WRAPPER_NAME}"

# This script loops through the frameworks embedded in the application and
# removes unused architectures.
find "$APP_PATH" -name '*.framework' -type d | while read -r FRAMEWORK
do
FRAMEWORK_EXECUTABLE_NAME=$(defaults read "$FRAMEWORK/Info.plist"
CFBundleExecutable)
FRAMEWORK_EXECUTABLE_PATH="$FRAMEWORK/$FRAMEWORK_EXECUTABLE_NAME"
echo "Executable is $FRAMEWORK_EXECUTABLE_PATH"

EXTRACTED_ARCHS=()

for ARCH in $ARCHS
```

```
do
echo "Extracting $ARCH from $FRAMEWORK_EXECUTABLE_NAME"
lipo -extract "$ARCH" "$FRAMEWORK_EXECUTABLE_PATH" -o
"$FRAMEWORK_EXECUTABLE_PATH-$ARCH"
EXTRACTED_ARCHS+=("$FRAMEWORK_EXECUTABLE_PATH-$ARCH")
done

echo "Merging extracted architectures: ${ARCHS}"
lipo -o "$FRAMEWORK_EXECUTABLE_PATH-merged" -create "${EXTRACTED_ARCHS[@]}"
rm "${EXTRACTED_ARCHS[@]}"

echo "Replacing original executable with thinned version"
rm "$FRAMEWORK_EXECUTABLE_PATH"
mv "$FRAMEWORK_EXECUTABLE_PATH-merged" "$FRAMEWORK_EXECUTABLE_PATH"

done
```

2.2.4 权限注意

在iOS 12的环境最好在Capabilities->Access WiFi Information 进行开启操作，这样方便设备指纹数据采集。



三、接口使用说明

3.1 方法和参数说明

```
// 风控组件：DXRiskManager类
@interface DXRiskManager : NSObject

// 字符串常量
extern NSString* const DXRiskManagerKeyUserId;
extern NSString* const DXRiskManagerKeyEmail;
extern NSString* const DXRiskManagerKeyPhone;
extern NSString* const DXRiskManagerKeyUserExtend1;
extern NSString* const DXRiskManagerKeyUserExtend2;
extern NSString* const DXRiskManagerKeyURL;//私有化服务器地址
extern NSString* const DXRiskManagerKeyBackup;//私有化下使用，将数据备份到顶象服务器
（开启为DXRiskManagerKeyBackupEnable 值）
extern NSString* const DXRiskManagerKeyBackupAppId; // 私有化下使用，指定数据备份到
顶象服务器的AppId

extern NSString* const DXRiskManagerKeyDegradeNotify;//数据降级通知，若打开，服务端
会有响应的降级统计
extern NSString* const DXRiskManagerKeyCountry;//国家地区设置，默认中国
```

```
extern NSString* const DXRiskManagerKeyDelayMsTime; //可填设置请求超时毫秒时间（默认值为 500 ，范围是：【100 ： 3000】）
// NoticeDegrade参数
/* The NoticeDegrade Value. This value only be used pair with key:DXRiskManagerKeyDegradeNotify to notify token degrade. */
extern NSString* const DXRiskManagerKeyDegradeNotifyEnable;
// Backup参数
/* The Backup Value. This value only be used pair with key:DXRiskManagerKeyBackup to set data backup. */
extern NSString* const DXRiskManagerKeyBackupEnable;
// Country参数
/* The Country Value. This value only be used pair with key:DXRiskManagerKeyCountry to set country. */
extern NSString* const DXRiskManagerCountryChina;
/* The Country Value. This value only be used pair with key:DXRiskManagerKeyCountry to set country. */
extern NSString* const DXRiskManagerCountryIndonesia;

/**
 采集端的设备指纹信息，上传至风控后台，再由风控后台返回token。
 该API为耗时操作，因此必须在非主线程上调用。

 @param appId appId   开通服务后可在实时风险决策的二级菜单“应用管理”中获取
 @param extendsParams 业务方用户唯一标识，用户名，用户ID，Email等等，常用Key值可参考DXRiskManagerKeyUserId，DXRiskManagerKeyEmail等等
 @return token
 */
+ (NSString *)getToken:(NSString *)appId extendParams:(NSDictionary *)extendsParams;

/**
 DXRiskManager -- 初始化方法
 */
+ (BOOL)setup;
@end
```

3.2 使用示例

```
// 整个过程由于是耗时操作，必须要在非主线程上执行，否则会阻塞UI。如果本身已经在非UI线程上执行，则不需要另开线程
    dispatch_queue_t dxrisk_queue = dispatch_queue_create("com.dingxiang.dxrisk", DISPATCH_QUEUE_CONCURRENT);
    dispatch_async(dxrisk_queue, ^{
        // 根据业务逻辑，填充自定义字段
        NSDictionary *dic = @{DXRiskManagerKeyUserId: @"123456"};
        // 如需设置海外服务器，可选值请参考头文件字段
        // NSDictionary *dic = @{DXRiskManagerKeyUserId: @"123456",DXRiskManagerKeyServiceArea:DXRiskManagerServiceAreaSoutheastAsia};
        // 如需自定义服务端URL，填充DXRiskManagerKeyURL字段，如有需要则添加DXRiskManagerKeyBackup 参数，Value固定为DXRiskManagerKeyBackupEnable，DXRiskManagerKeyBackupAppId参数可不填，参数为顶象备份数据库提供的备份Appid，则如下注释
```

```
        // NSDictionary *dic = @{DXRiskManagerKeyUserId:
@"123456",DXRiskManagerKeyURL:@"http://xxxxxxx",DXRiskManagerKeyBackup:DXRiskManag
erKeyBackupEnable,DXRiskManagerKeyBackupAppId:@"xxxxxxxxxxxxxxxxxxx"};
        // 获取token
        // 注意：token最好不要保存在某个局部变量或者字段，每次使用时，都通过API获取。
        BOOL isSuccess = [DXRiskManager setup];
        NSLog(@"setup success: %@" , isSuccess ? @"YES":@"NO");
        NSString *constID = [DXRiskManager getToken:@"xxxxxxxxxxxxxxxxxxx"
extendsParams:dic];
        NSLog(@"constID: %@", constID);
        // TODO 把constid通过Post请求，传到业务后台。
        // 下面是模拟频繁调用的过程
        while(TRUE) {
            NSLog(@"constID: %@", [DXRiskManager getToken:@"xxxxxxxxxxxxxxxxxxx"
extendsParams:dic]);
            [NSThread sleepForTimeInterval:.5];
        }
    });
```

3.3 异常说明

在获取token过程中，如果因为网络超时或者加解密失败，该接口有可能会返回为null，同时会输出tag为DXRISK的错误信息，具体描述如下：

```
DXRISK_REQUEST_NETWORK_ERR            -1001
DXRISK_REQUEST_DECRYPT_ERR            -1002
DXRISK_REQUEST_UNCOMPRESS_ERR         -1003
DXRISK_REQUEST_RESPONSE_EMPTY_ERR     -1004
DXRISK_REQUEST_DATA_PARSE_ERR         -1005
DXRISK_REQUEST_DIRTY_DATA_ERR         -1006
DXRISK_CONST_ID_EMPTY                 -1007
```

如果出现上述错误信息，请联系顶象技术人员。

3.4 微信小程序接入

js接入

一、下载设备指纹js

SaaS版本下载

私有化版本下载

请注意选择对应的版本，下载完成后放到本地项目中。

二、获取密钥

未注册用户可在顶象官网进行账号注册，创建应用获取应用密钥AppID和AppSecret。

三、使用

1. 代码接入

```
const ConstId = require('本地设备指纹js存放路径')
Page({
    onLoad: function () {
      new ConstId({
        appId: '【这里填写在顶象官网申请到的 AppID】', // 唯一标识，必填
        server: 'https://host/udid/w1' // constId 私有化服务接口，选填
      }, (e, id) => {
        if (e) {
          console.log(e)
          return
        }
        console.log('constId:', id)
      })
    }
})
```

2. SaaS用户在小程序后台配置业务域名https://constid.dingxiang-inc.com，私有化用户配置部署域名，测试阶段可以开启微信开发者工具右上角详情->本地设置->“不校验域名”。

四、用户端对于设备指纹token的获取

用户前端接入设备指纹的SDK后，获取设备指纹token，并传入用户后端，具体业务实现由用户自行开发。

五、设备指纹后端接入

接口详细描述： 根据token,appId,sign三个参数获得设备信息

1.SDK接入方法说明

点击下载SDK

1.1 Java SDK接入

1. 包的引入

```
<!-- 将constid-client-sdk.jar deploy到自己的maven私有仓库或insall本地，然后添加依赖: -->
<dependency>
    <groupId>com.dingxianginc</groupId>
    <artifactId>constid-client-sdk</artifactId>
    <version>3.0</version>
</dependency>
```

2. Java使用示例

```
public class Demo {
    private static String appId = "你的AppID";
    private static String appSecret = "你的AppSecret";
    private static String token = "SDK里面获取到的token";

    public static void main(String[] args) {
        // 填写设备指纹域名或者url如：http://127.0.0.1:8080
        String result =
DeviceFingerprintHandle.getDeviceInfo("https://constid.dingxiang-inc.com", appId,
            appSecret, token);
    }
}
```

1.2 PHP SDK接入

点击下载SDK

使用示例

```
class Demo {

    // 根据实际情况填写
    const appKey = "你的AppID";

    // 根据实际情况填写
    const appSecret = "你的AppSecret";

    // 根据实际情况填写
    const token = "SDK里面获取到的token";
}

// 根据token获取设备详细信息工具类
$requestHandle = new DeviceFingerprintHandle();
// 设置请求超时时间。因为存在设备指纹降级和网络抖动的情况，默认2秒。可以根据实际情况调整
// $requestHandle->setTimeout(2);

// 填写设备指纹域名或者url入：http://127.0.0.1:9090
$responseData = $requestHandle->getDeviceInfo("https://constid.dingxiang-
inc.com/udid/api/getDeviceInfo",
    Demo::appKey, Demo::appSecret, Demo::token);
$result = json_decode($responseData, true);

// 请求状态码。非 200 表示没有获取到设备明细信息
if ($result['stateCode'] == 200)
    echo "设备明细信息如下：  : " . json_encode($result['data'], true);
else
    echo $result['message'];
```

2.HTTP接口接入方法

2.1 请求参数

字段	类型	描述
appId	String	当前应用的标识
sign	String	sign = MD5(appSecret + token + appSecret),AppSecret为appId对应的密钥
token	String	用户前端获取的token

2.2 成功响应

字段	类型	描述
stateCode	int	状态码
message	String	状态描述
data	Json	返回设备信息及设备风险检测数据

```
{
    "stateCode": 200,
    "message": "请求响应成功",
    "data": {
        "constId": "22e38229a7eda501c58bf3ddee1a340a",
        ......
    }
}
```

2.3 错误响应

字段	类型	描述
stateCode	int	状态码
message	String	状态描述
data	Json	异常时数据为空

```
{
    "stateCode": -10002,
    "message": "签名为空或验证失败",
    "data": null
}
```

错误码	描述
-10001	appId不存在或已经过期

错误码	描述
-10002	签名为空或验证失败
-10003	token为空或没有对应的设备信息
-10004	token已经过期
-10005	服务器内部异常
-10006	证书已经失效
-10007	服务器限流

3. HTTP接口返回参数明细说明

3.1 免费用户

字段名	字段描述	说明
token	设备token	默认返回
constId	设备id	默认返回

3.1.1 web端返回结果

字段名	字段描述	说明
token	设备token	默认返回
constId	设备id	默认返回

3.2 收费用户

3.2.1 移动端返回结果

字段名	字段描述	说明
token	设备token	默认返回
deviceType	设备类型	默认返回
constId	设备id	默认返回
producter	生产厂商	配置返回
macAddress	mac地址	配置返回
isEmulator	模拟器运行	配置返回
isRoot	是否root	配置返回
isMultirun	是否多开	配置返回
isInject	是否存在注入风险	配置返回

字段名	字段描述	说明
isMemdump	是否存在内存dump风险	配置返回
isDebug	是否存在调试风险	配置返回
isFlawJanus	是否有Janus漏洞	配置返回
isHook	是否存在hook风险	配置返回
isJailBreak	是否越狱	配置返回
isVpn	是否使用vpn	配置返回
isProxy	是否使用代理	配置返回

3.2.2 web端返回结果

字段名	字段描述	说明
token	设备token	默认返回
deviceType	设备类型	默认返回
constId	设备id	默认返回
can	canvas指纹	配置返回
web	WebGl指纹	配置返回
res	设备分辨率	配置返回
hc	硬件的并发特性	配置返回
isLiedBrowser	是否伪造浏览器	配置返回
isCookEnable	是否禁用cookie	配置返回
isTamperUa	是否篡改浏览器ua	配置返回
isTamperRes	是否篡改分辨率	配置返回
isTamperCd	是否篡改浏览器颜色深度	配置返回

六、FAQ

详情点击：https://www.dingxiang-inc.com/docs/preview/detail/const-id-faq