腾讯云可观测平台

终端性能监控

产品文档

腾讯云

文档目录

终端性能监控
终端性能监控概述

最近更新时间：2024-05-14 12:36:06

简介

终端性能监控是全方位定位检测 App 应用性能和用户体验的工具，多维度自动分析出各维度存在可疑的性能缺陷。帮助您精确衡量 App 应用的性能，以低成本、高效率发现 App 应用各类问题。

APP 监控功能说明

功能名称	说明
崩溃（Crash）	通过对崩溃问题个例提取关键特征进行聚合，便于您定位分析崩溃根因。
启动	支持启动耗时、慢启动占比等进行启动指标分析，您可通过慢启动问题列表定位分析慢启动根因。
卡慢	通过流畅度等指标分析，便于您分析 App 页面性能情况。
ANR	多维度还原线上用户真实体验，通过收集用户真实使用 App 过程中碰到的 ANR 问题及问题发生时各线程堆栈信息，提取关键特征聚类。
网络	通过请求耗时、慢请求占比、网络错误率等指标进行网络问题分析。
WebView	通过页面加载耗时、慢加载占比以及 JS 错误率等进行 WebView 指标分析，并通过问题列表对 WebView 以及 JSError 问题进行下钻。

数据存储说明

问题个例数据（慢启动、慢请求等问题列表）：存储 60天。
15分钟的指标数据（指标分析界面）：存储30天。
1小时的指标数据（指标分析界面）：存储30天。

接入指南
安卓应用场景
集成和初始化

最近更新时间：2024-05-14 12:36:06

操作场景

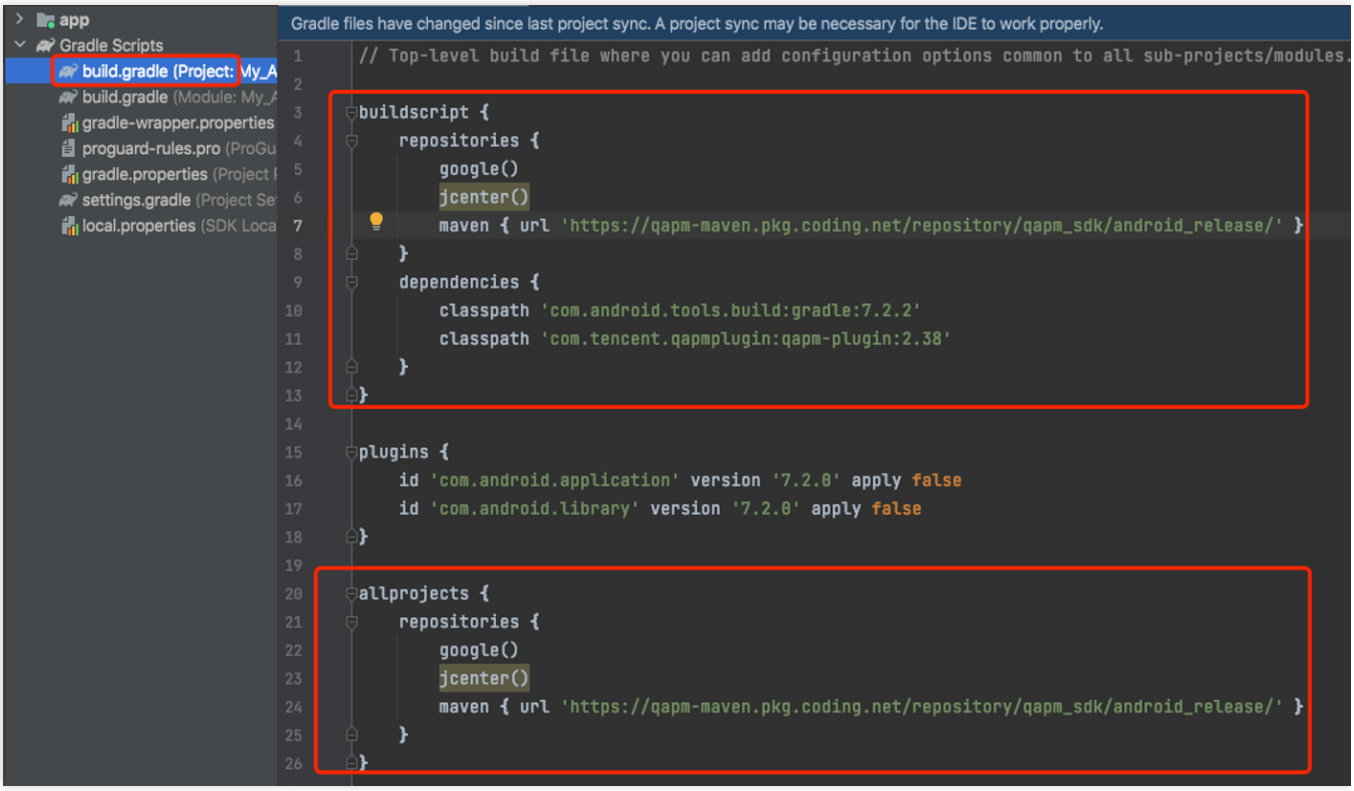
本文指导您使用 Android SDK 的集成与初始化。

操作步骤

步骤一：Gradle 集成

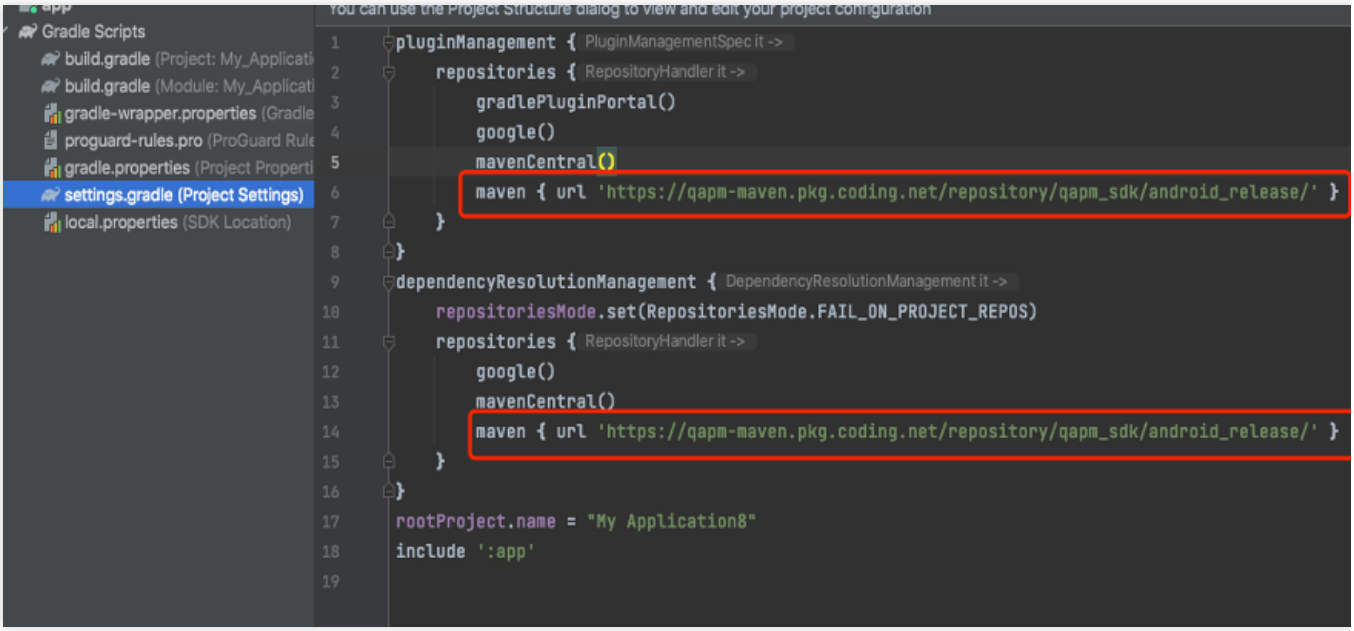
1. **在工程级的 build.gradle 中加入 maven 依赖**。
   i. 添加 buildscript 和 allprojects（gradle 7.0 以上不需要添加 allprojects）

gradle 7.0 以下版本采用如下配置：

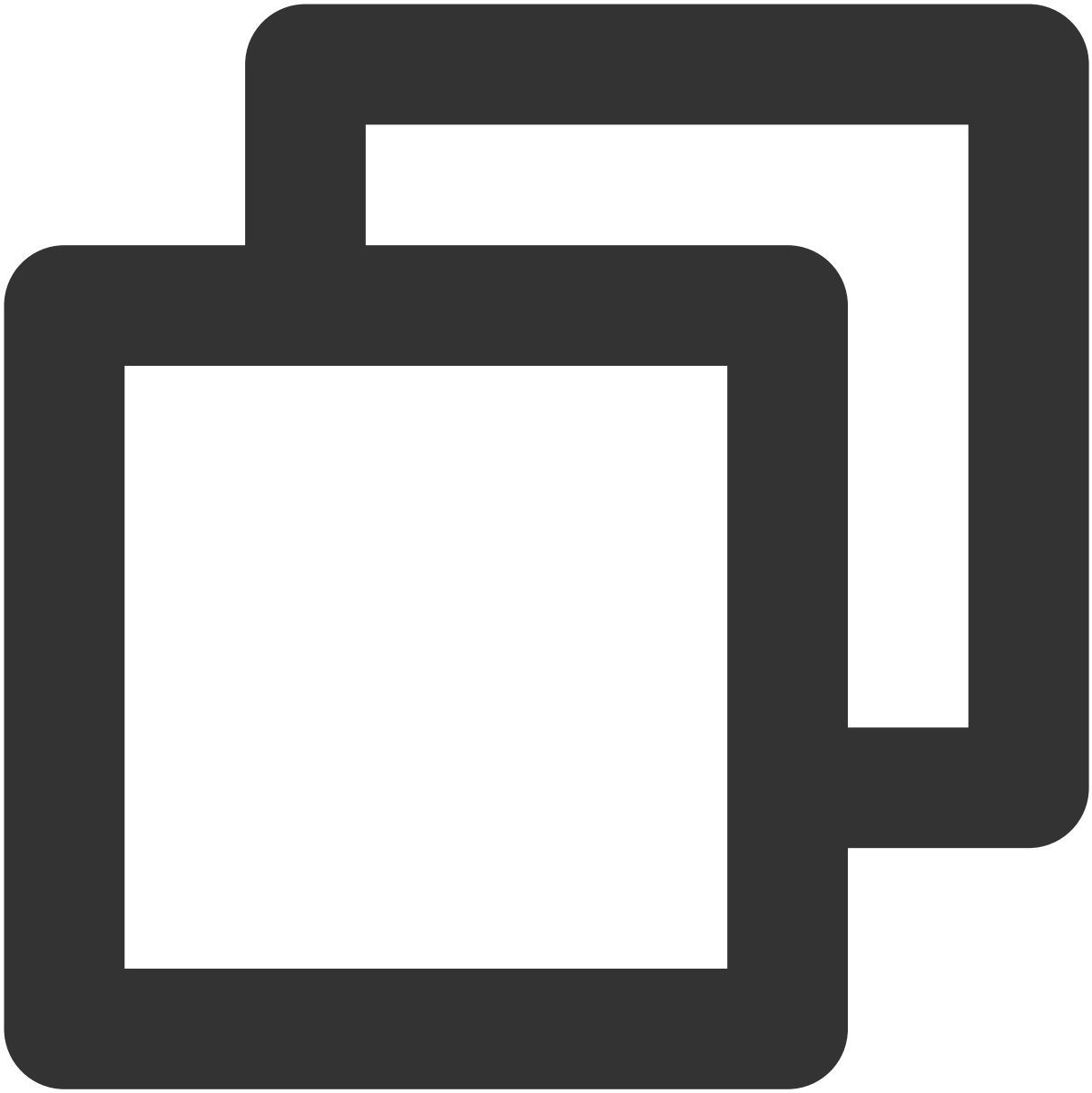


gradle 7.0 以上版本采用如下配置：

Gradle 7.0 以上版本不支持 allprojects。allproject 的 maven 依赖需要在 setting.gradle 中配置。如下图：



参见代码：



```
maven {url'https://qapm-maven.pkg.coding.net/repository/qapm_sdk/android_release/'}
```

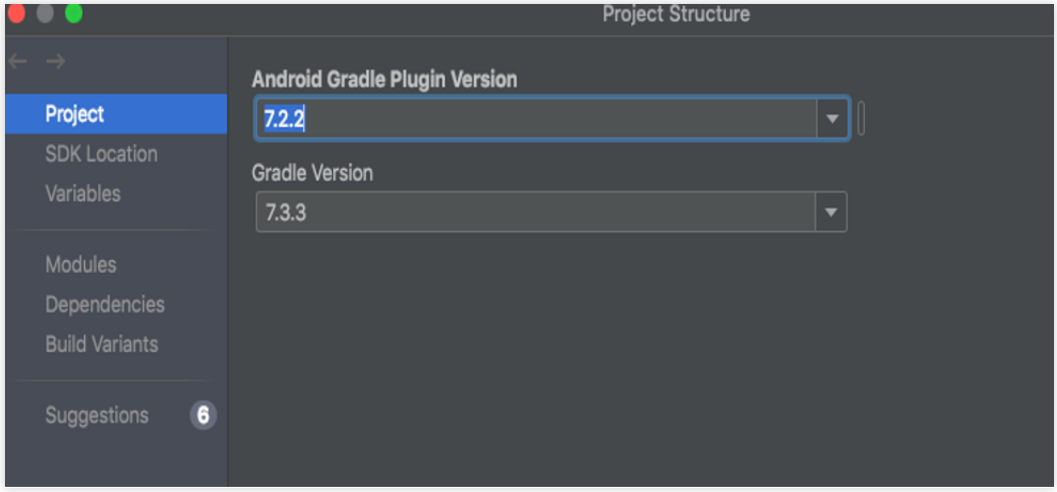
ii. 在 buildscript 中的 `com.android.tools.build:gradle:*.*.*` 要填写成您的 gradle plugin 版本。如下图：

```
buildscript {
    repositories {
        google()
        jcenter()
        maven { url 'https://qapm-maven.pkg.coding.net/repository/qapm_sdk/android_rel
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:7.2.2'
        classpath 'com.tencent.qapmplugin:qapm-plugin:2.38'
    }
}
```

gradle plugin version

说明：

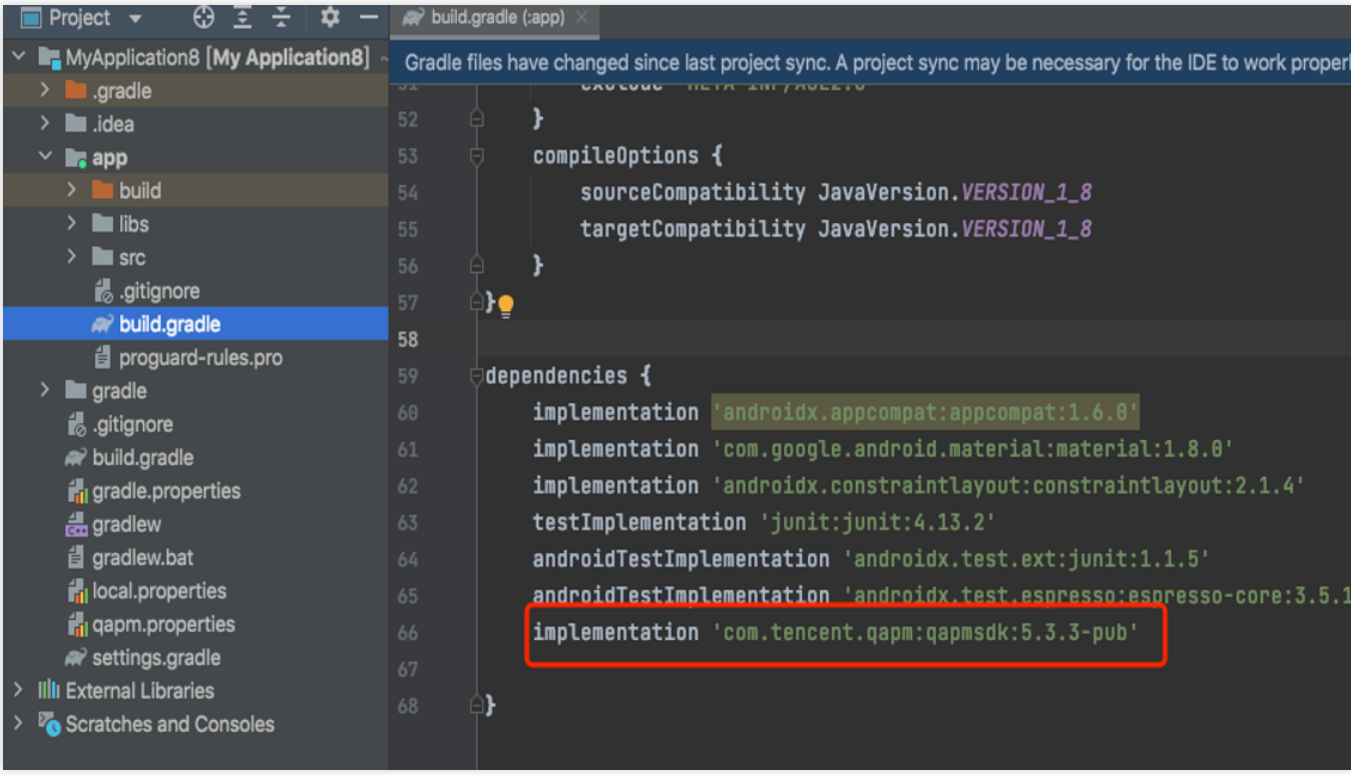
1. Gradle 版本和 gradle plugin 版本可以在菜单 file > project structure 中查看。如下图：



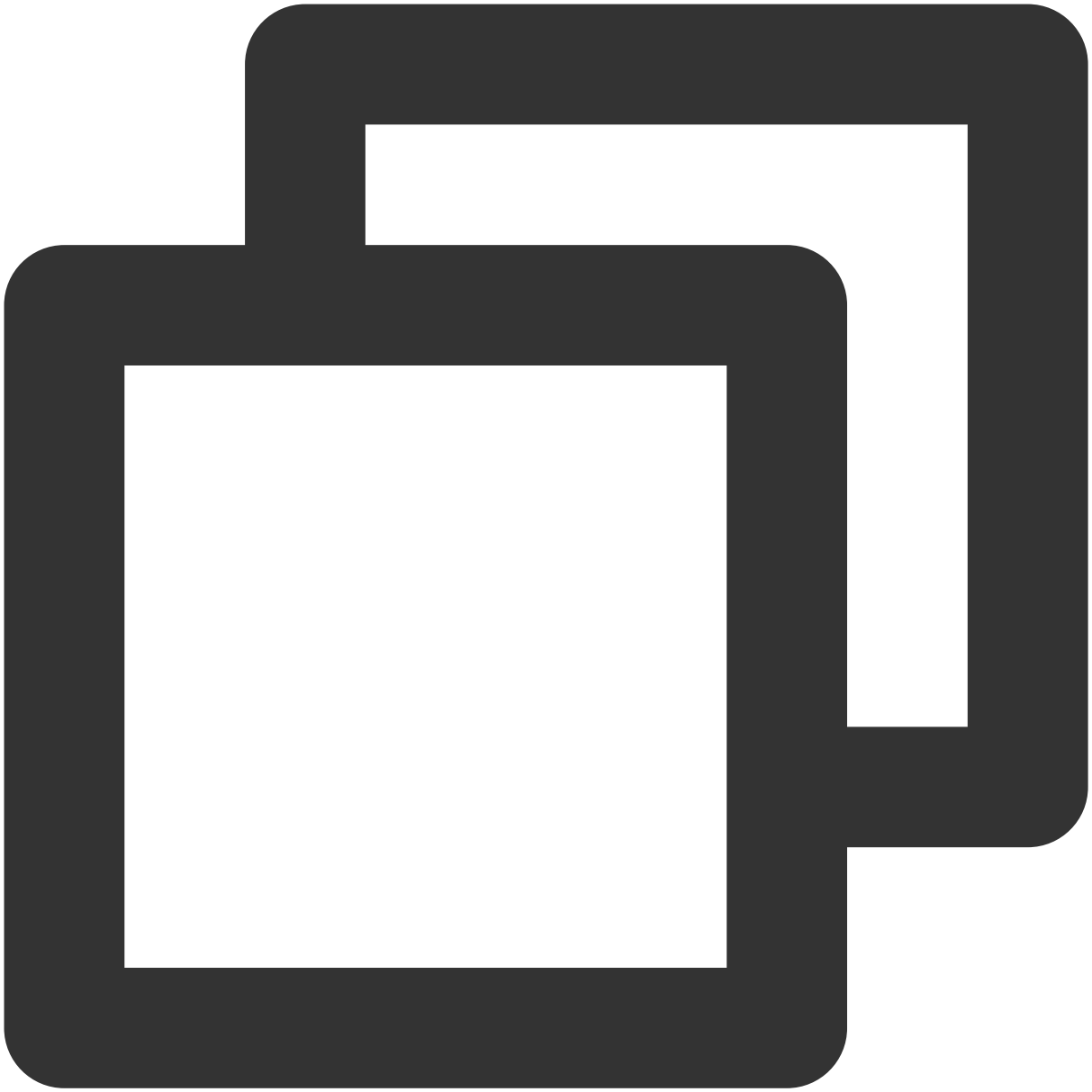
2. Gradle 和 gradle plugin 相对应关系，如下图。

插件版本	所需的最低 Gradle 版本
8.1	8.0
8.0	8.0
7.4	7.5
7.3	7.4
7.2	7.3.3
7.1	7.2
7.0	7.0
4.2.0+	6.7.1

2. 在 App 的 build.gradle 中引入模块（studio 版本在 3.0 以下的请使用 compile 的引用头）。


Project
MyApplication8 [My Application8]
.gradle
.idea
app
build
libs
src
.gitignore
build.gradle
proguard-rules.pro
gradle
.gitignore
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
qapm.properties
settings.gradle
External Libraries
Scratches and Consoles
build.gradle (:app)
Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properl
52 }
53 compileOptions {
54 sourceCompatibility JavaVersion.VERSION_1_8
55 targetCompatibility JavaVersion.VERSION_1_8
56 }
57 }
58
59 dependencies {
60 implementation 'androidx.appcompat:appcompat:1.6.0'
61 implementation 'com.google.android.material:material:1.8.0'
62 implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
63 testImplementation 'junit:junit:4.13.2'
64 androidTestImplementation 'androidx.test.ext:junit:1.1.5'
65 androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1
66 implementation 'com.tencent.qapm:qapmsdk:5.3.3-pub'
67
68 }

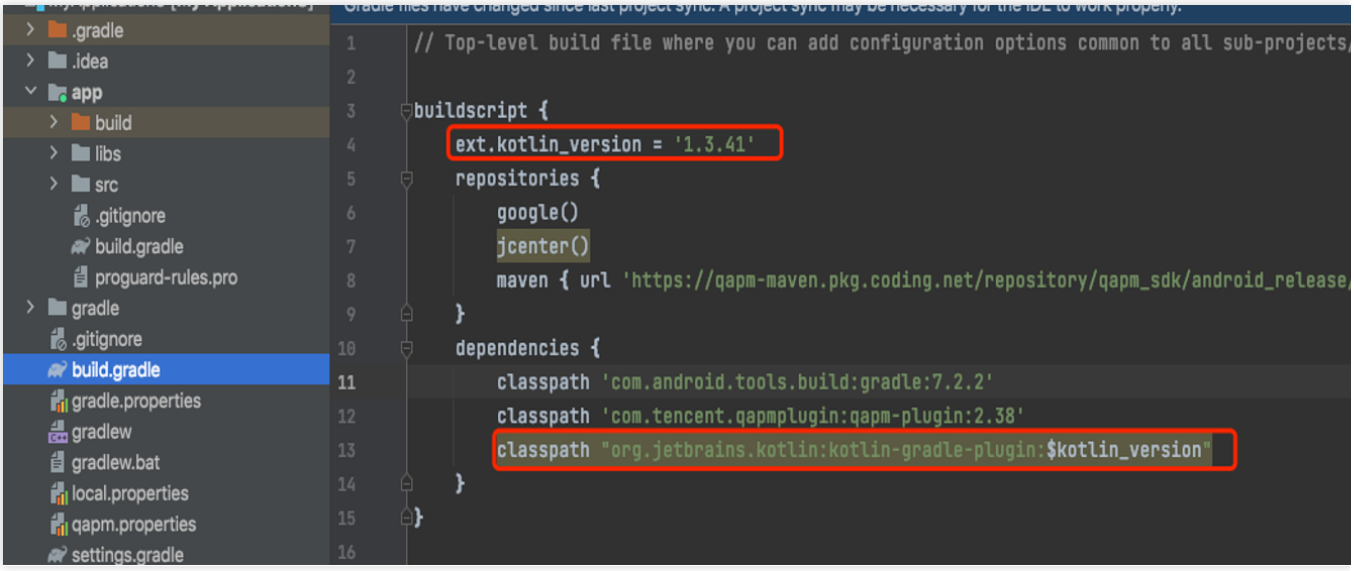
参见代码：



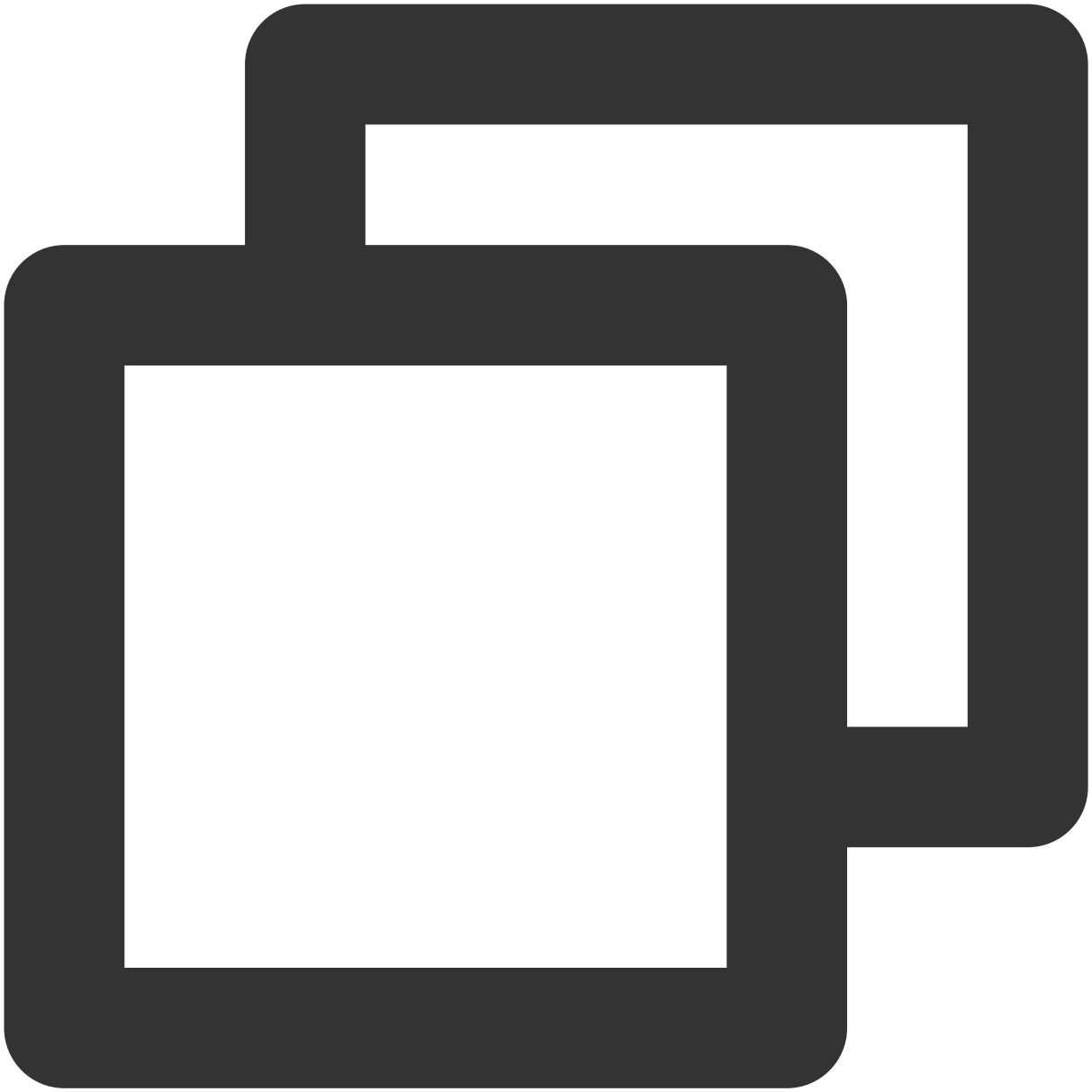
```
implementation 'com.tencent.qapm:qapmsdk:5.3.9-pub'
```

3. **引入 kotlin 依赖**。

i. 在工程级的 build.gradle 下加入如下代码：

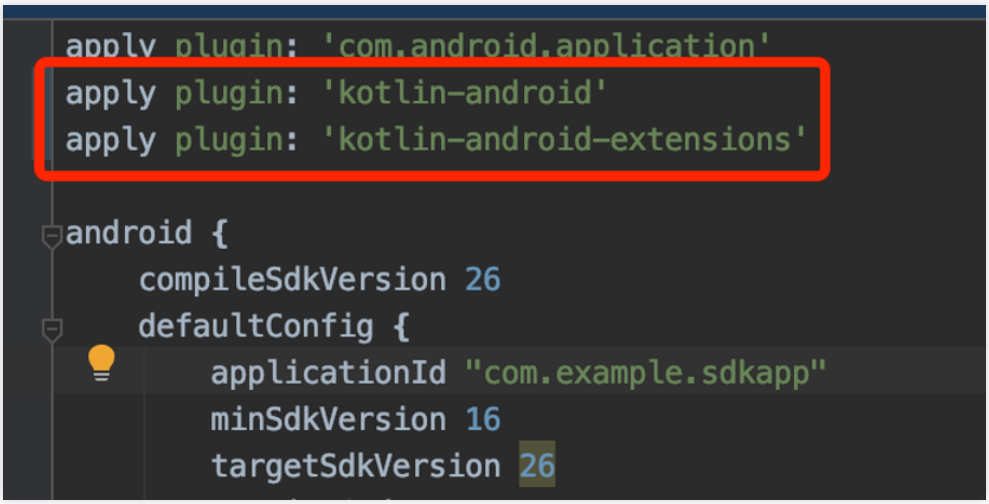

.gradle
.idea
app
build
libs
src
.gitignore
build.gradle
proguard-rules.pro
gradle
.gitignore
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
qapm.properties
settings.gradle
// Top-level build file where you can add configuration options common to all sub-projects
buildscript {
ext.kotlin_version = '1.3.41'
repositories {
google()
jcenter()
maven { url 'https://qapm-maven.pkg.coding.net/repository/qapm_sdk/android_release
}
dependencies {
classpath 'com.android.tools.build:gradle:7.2.2'
classpath 'com.tencent.qapmplugin:qapm-plugin:2.38'
classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
}
}

参见代码：

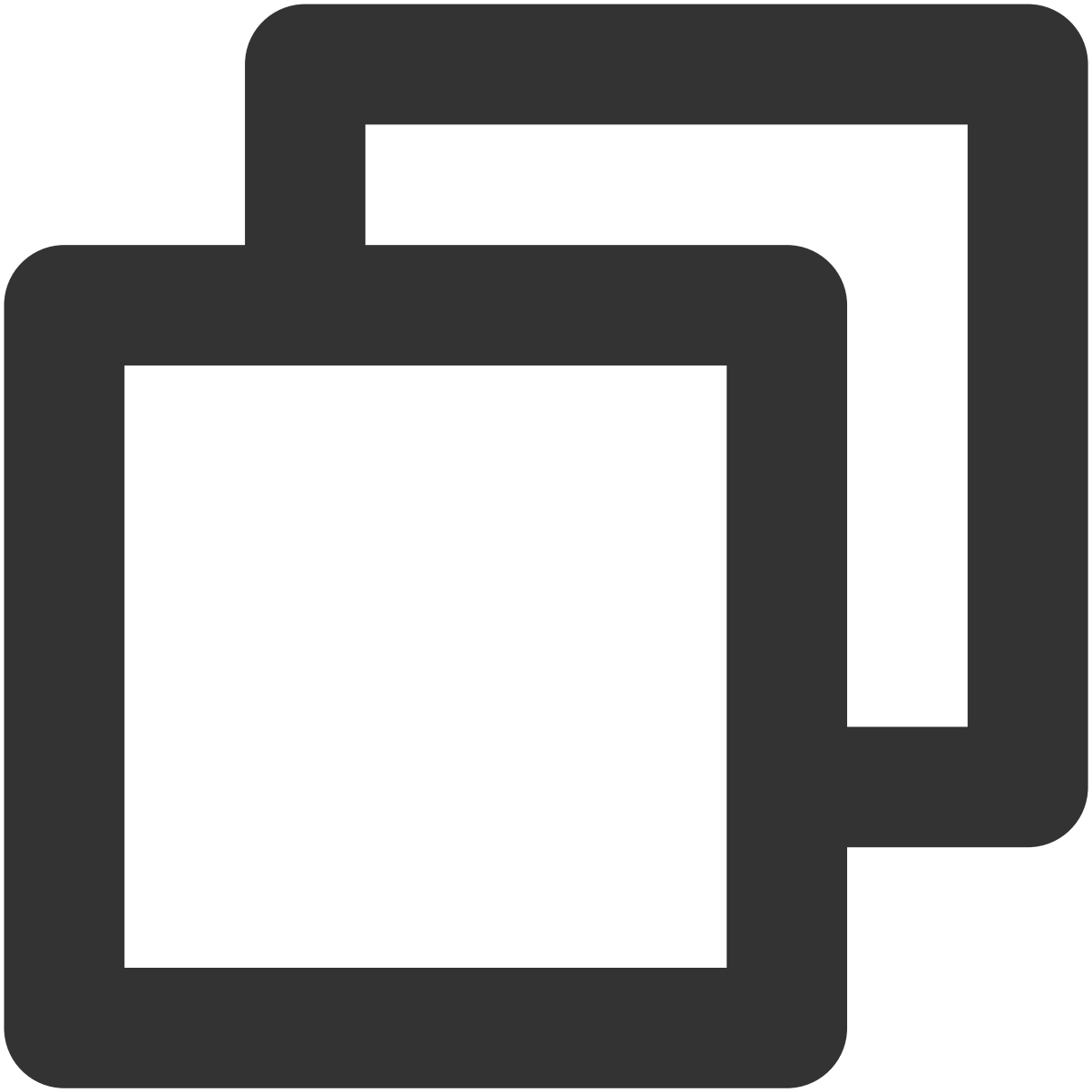


```
ext.kotlin_version = '1.3.41'
classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
```

ii. 在 App 的 build.gradle 下加入如下代码：

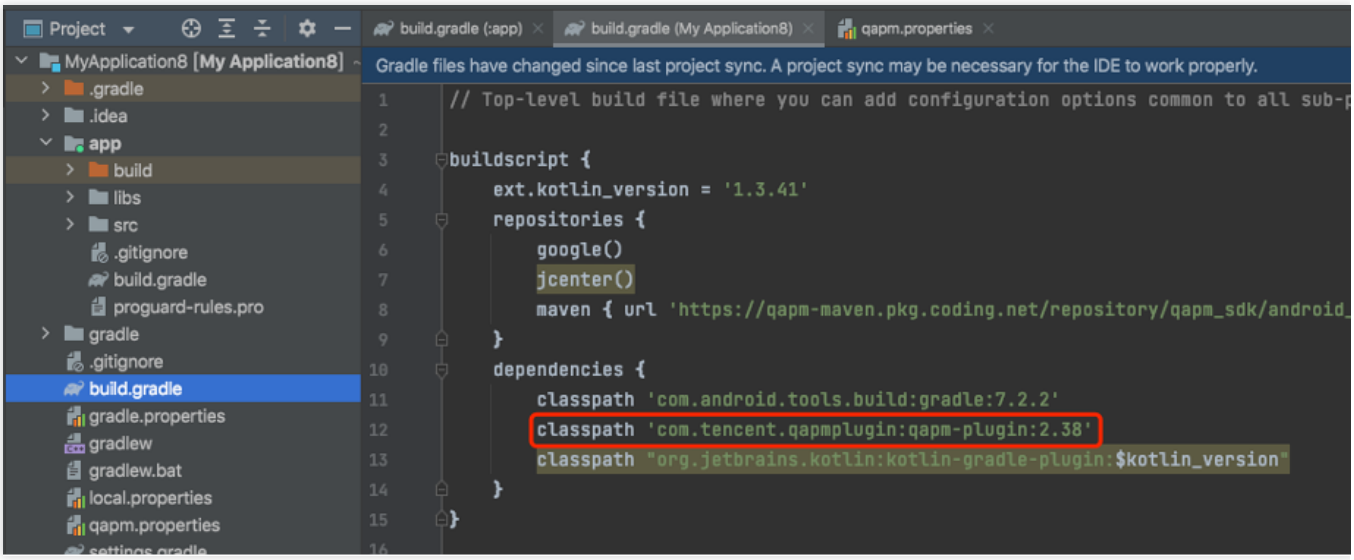

apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
android {
compileSdkVersion 26
defaultConfig {
applicationId "com.example.sdkapp"
minSdkVersion 16
targetSdkVersion 26

参见代码：

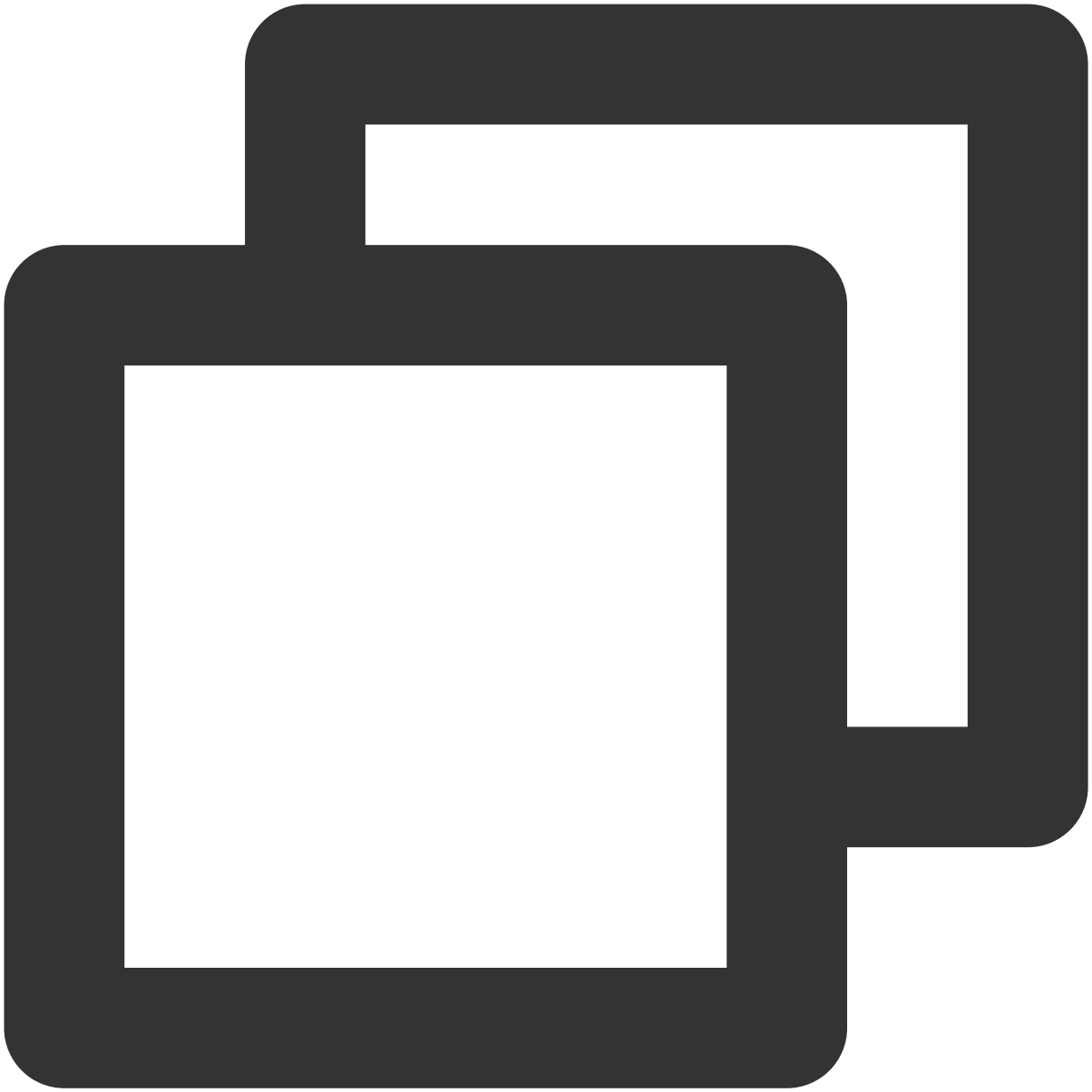


```
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
```

4. **在工程级的 build.gradle 文件中加入以下配置**。


Project
MyApplication8 [My Application8]
.gradle
.idea
app
build
libs
src
.gitignore
build.gradle
proguard-rules.pro
gradle
.gitignore
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
qapm.properties
build.gradle (:app)
build.gradle (My Application8)
qapm.properties
Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly.
// Top-level build file where you can add configuration options common to all sub-p
buildscript {
ext.kotlin_version = '1.3.41'
repositories {
google()
jcenter()
maven { url 'https://qapm-maven.pkg.coding.net/repository/qapm_sdk/android_
}
dependencies {
classpath 'com.android.tools.build:gradle:7.2.2'
classpath 'com.tencent.qapmplugin:qapm-plugin:2.38'
classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
}
}

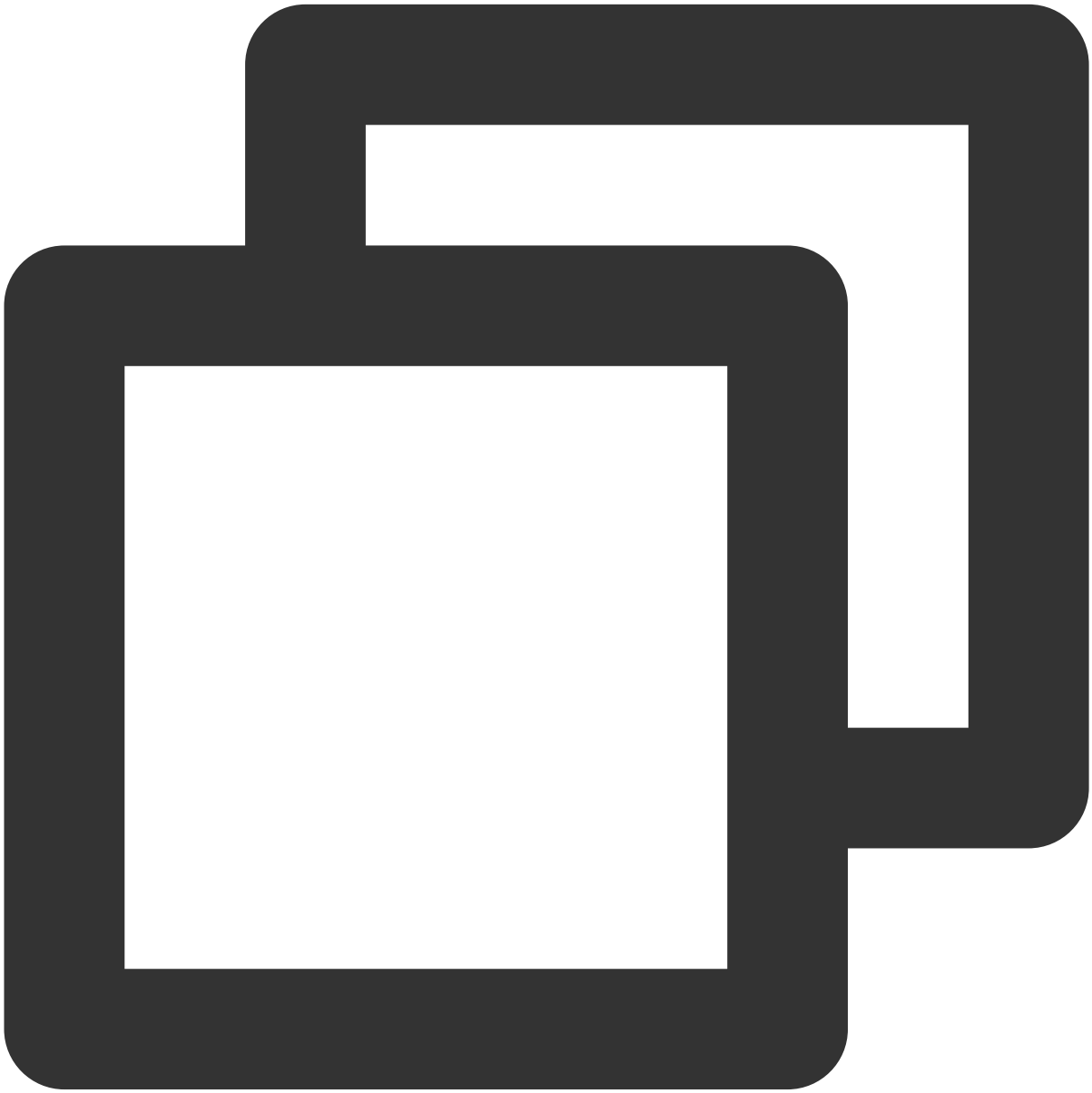
参见代码：



```
classpath 'com.tencent.qapmplugin:qapm-plugin:2.39'
```

5. 在 App 的 build.gradle 文件中加入以下配置。

参见代码：



```
apply plugin: 'qapm-plugin'
QAPMPluginConfig{
// 可选,默认为空,请在Application所在的类中输入attachBaseContext,看有没有这个的重写方法，如
```

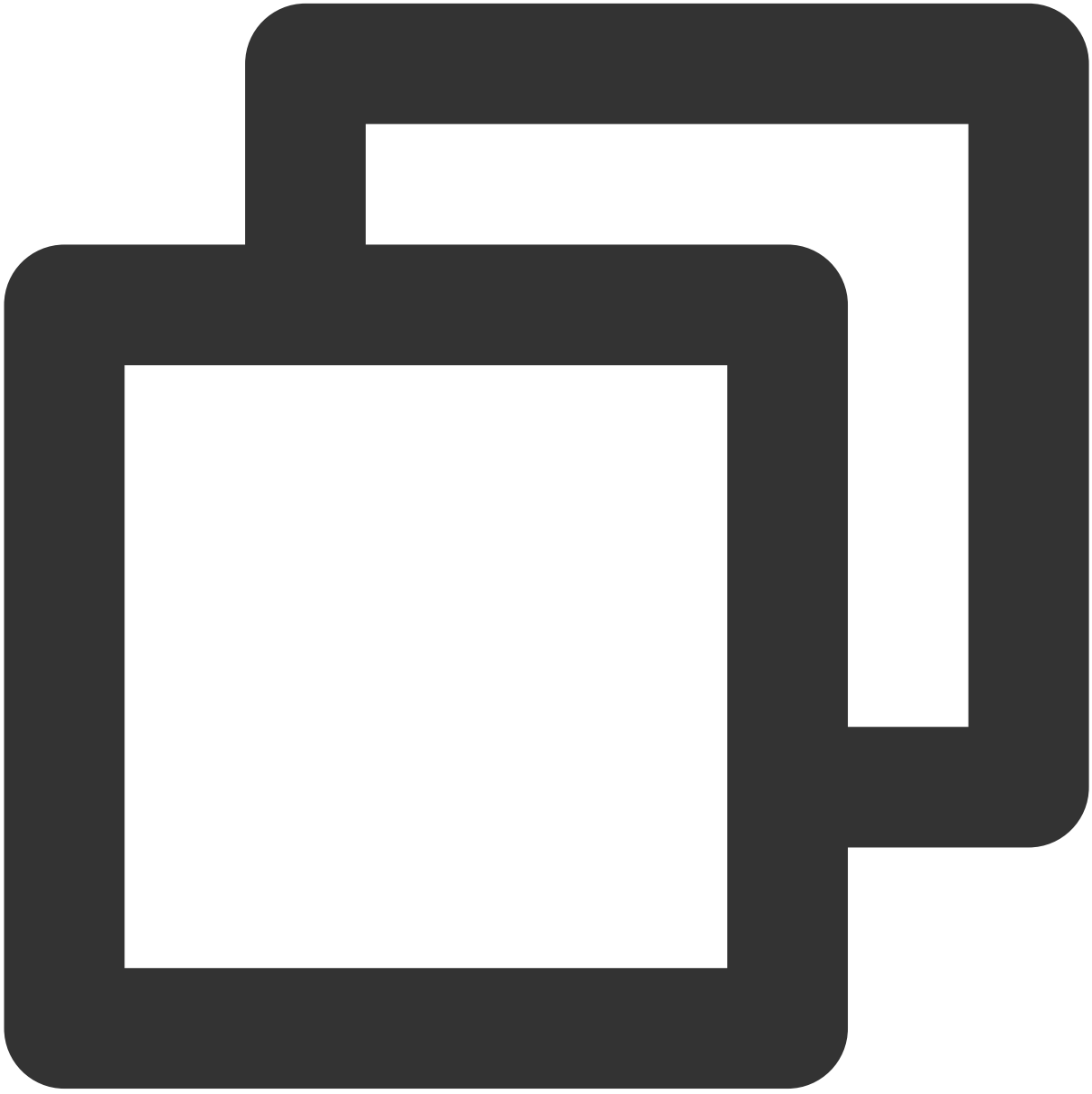
```
}
```

说明：

若未配置4-5项，则会影响“App 启动”的数据上报。

步骤二：参数配置

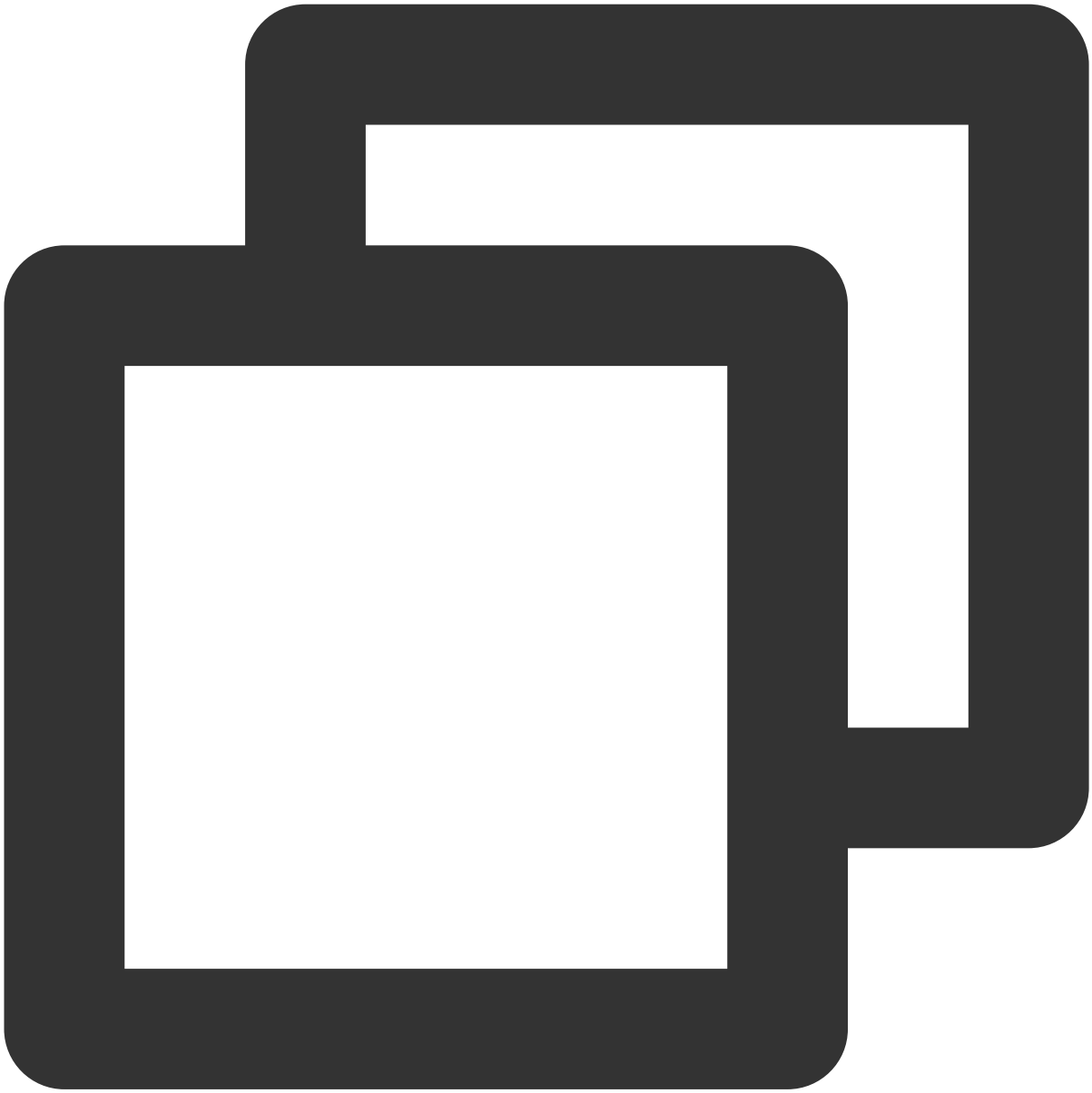
1. 在 AndroidManifest.xml 中添加以下权限。



```
<!--上报信息所需-->
<uses-permission android:name="android.permission.INTERNET" />
<!--采集信息所需-->
```

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

2. 为了避免混淆 SDK，在 App 的 proguard-rules.pro 文件中增加以下配置：

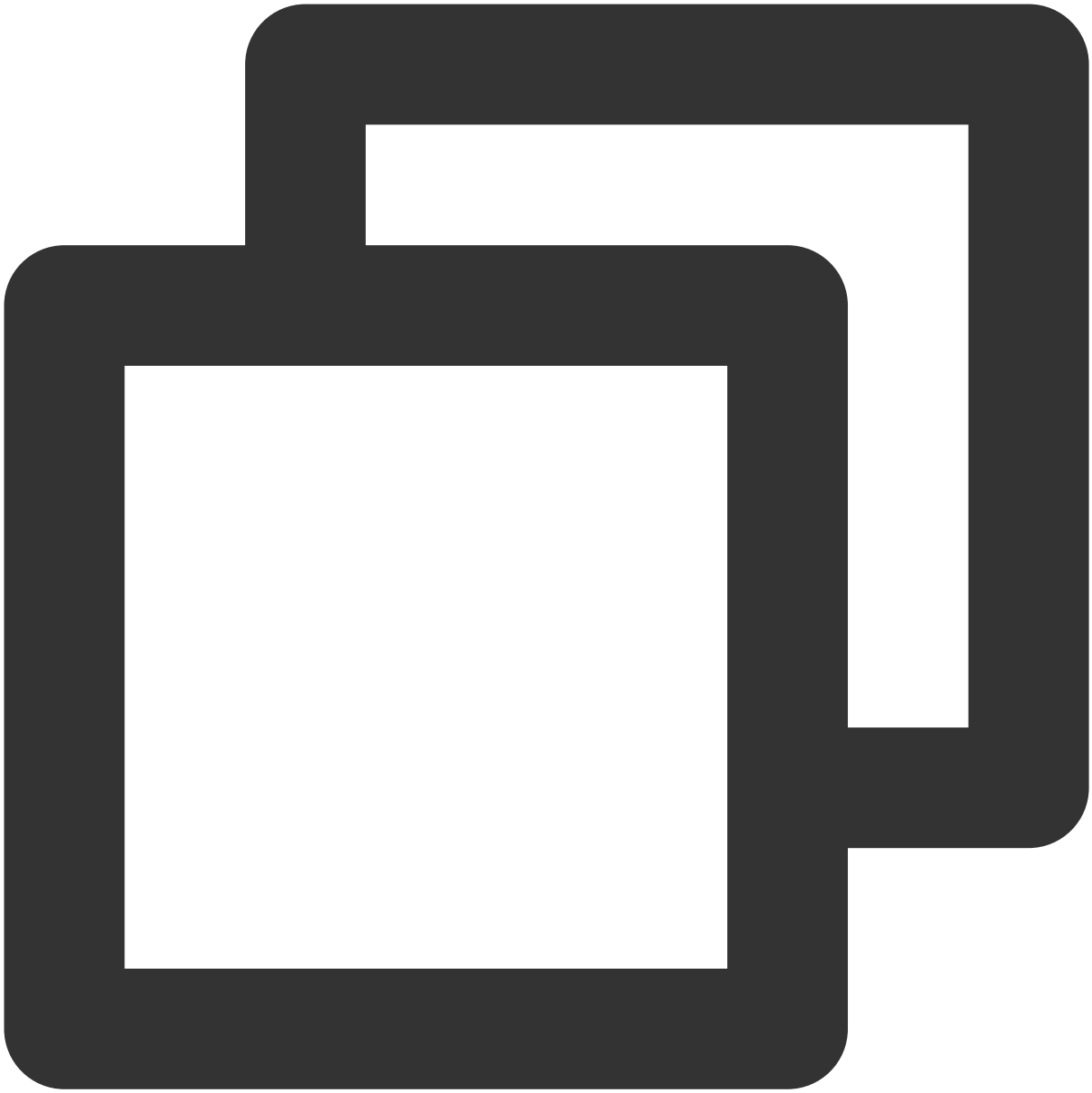


```
-keep class com.tencent.qapmsdk.**{*;}
# 如需要网络监控，请确保okhttp3不被混淆
-keep class okhttp3.**{*;}
```

步骤三：初始化 SDK

1. 登录 腾讯云可观测平台 控制台，在**终端性能监控**页面，选择 应用管理 > 应用设置 后，获取 Appkey（上报 ID）。

2. 拷贝下面代码，并修改其中部分字段。下列项均是必需的接口设置，其余接口配置请参考初始化的接口分析（建议在 Application 中初始化 QAPM）。



```
// 设置手机型号和设备ID。
// 需要传入设备的标识，任意字符串。deviceId（必需！！）
// deviceId可以用来开启白名单，避免数据被抽样上报(崩溃和启动以外的数据抽样率为0.1%)
QAPM.setProperty(QAPM.PropertyKeyDeviceId, "设备的标识");
// 需要传入手机型号（必需！！）
QAPM.setProperty(QAPM.PropertyKeyModel, "填写手机型号");

// 设置Application（必需）
QAPM.setProperty(QAPM.PropertyKeyAppInstance, getApplication());
// 设置AppKey（必需，用于区分上报的产品,该值由终端性能监控的产品配置页面获取，可参考上一步骤 ）
QAPM.setProperty(QAPM.PropertyKeyAppId, "YourAppKey");
```

```
// 设置产品版本，用于后台检索字段（必需）
QAPM.setProperty(QAPM.PropertyKeyAppVersion, "YourApp Version");
// 设置UUID，用于拉取被混淆堆栈的mapping （必需，若使用了QAPM符号表上传插件，可以直接使用该变量）
// 该变量会在build时生成，爆红不用理
QAPM.setProperty(QAPM.PropertyKeySymbolId, BuildConfig.QAPM_UUID);
// 设置用户ID，任意字符串，用于后台检索字段（必需）
// userId可以用来开启白名单，避免数据被抽样上报(崩溃和启动以外的数据抽样率为0.1%)
QAPM.setProperty(QAPM.PropertyKeyUserId, "123456");
// 设置Log等级，（可选），线上版本请设置成QAPM.LevelOff或者 QAPM.LevelWarn
QAPM.setProperty(QAPM.PropertyKeyLogLevel, QAPM.LevelInfo);
// 设置QAPM的外网上报域名（必需）。国内站：https://app.rumt-zh.com  国际站：https://app.rur
QAPM.setProperty(QAPM.PropertyKeyHost,"https://app.rumt-zh.com");
QAPM.setProperty(QAPM.PropertyKeyHost,"https://app.rumt-sg.com");
// 启动QAPM
QAPM.beginScene(QAPM.SCENE_ALL, QAPM.ModeStable);
```

说明：

AppKey 可参考步骤1在终端性能监控-应用配置页面获取。

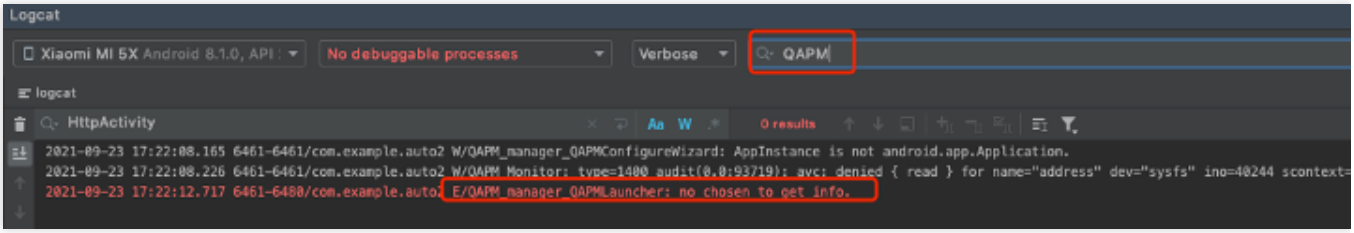
崩溃和启动数据是全量上报，其他数据因为数据数目过多，采取抽样上报，抽样率为0.1%（千分之一）。如果需要全量上报，可以开启白名单，App 将会在下次启动时改变抽样率。

可以将设置好 userId 或者 deviceId 通过 应用管理 页面添加白名单里，开启白名单。

多个进程需要各自初始化 QAPM。

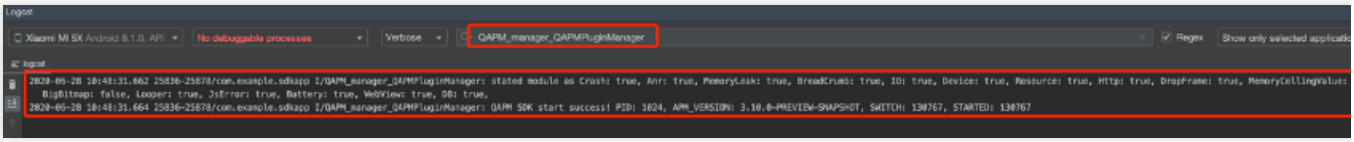
步骤四：接入验证

1. 若打印以下日志，代表该用户未被抽样命中，需重新设置下抽样率：



参见 TAG : QAPM_manager_QAPMLauncher

2. 若打印以下日志，则代表初步接入成功，可以验证数据上报/尝试开启高级功能：



参见 TAG : QAPM_manager_QAPMPluginManager

初始化的接口分析

接口名称	参数	参数说明	注意事项
public static QAPM setProperty(int key, Stringvalue) **作用**：设置 QAPM 的相关参数	key	**必填**。需要设置的 Key。	-
	QAPM.PropertyKeyLogLevel	**选填**。开启日志等级（建议 Debug 版本开启 QAPM.LevelDebug, release 版本开启 QAPM.LevelWarn）。	
	QAPM.PropertyNeedTranslate	**选填**。堆栈是否需要翻译，这里默认是需要翻译的。如果 apk 是没有混淆的需要传入 false，否则前端可能会全部展示为 unTranslated。	
public static boolean beginScene(String sceneName, int mode) **作用**：开启监控	sceneName	**必填**。场景名。	正式版本建议开启 QAPM.Mo 研发版本建议开启 QAPM.Mo 默认会开启 ModeStable 功能 ModeStable 使用或运算的方 的功能，如开启 Stable 和内存 beginScene(“Stable&触顶”, QAPM.ModeStable| QAPM.ModeCeiling)。可使用 排除不需要的功能，如关闭网 QAPM.ModeStable^QAPM.M
	mode	**必填**。开启的功能。	
	QAPM.ModeStable	**选填**。开启全部功能（建议外发版本开启。包含区间性能、crash、anr、webview 页面加载、JsError、网络）。	
	QAPM.ModeWebView	**选填**。开启 WebView 页面加载监控。	
	QAPM.ModeJsError	**选填**。开启 WebView 的 JS 异常监控。	
	QAPM.ModeHTTPInWeb	**选填**。开启 WebView 的网络监控。	
	QAPM.ModeHTTP	**选填**。开启网络监	

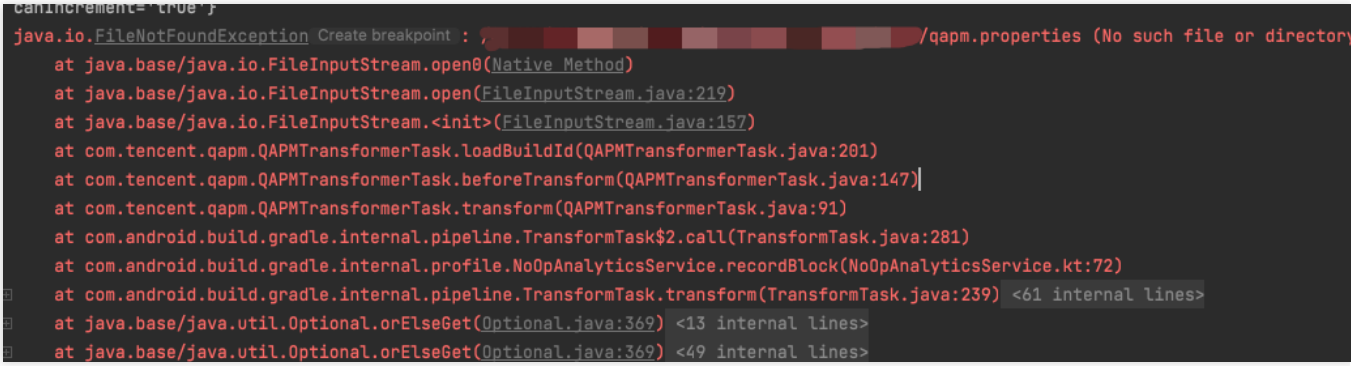
		控。	
public static boolean endScene(String sceneName, long mode) **作用**：结束监控（只针对掉帧和区间性能采集有效）	sceneName	**必填**。需要关掉的场景名（与 beginScene 的要相对应）。	-
	QAPM.ModeDropFrame	**选填**。关闭掉帧监控。	
	QAPM.ModeResource	选填。关闭区间性能监控。	

其他问题

说明：

通过 qapm 插件编译打包 App 时，App 需要一个 uuid 作为构建 id，如果项目目录下存在 qapm.properties 文件，并且文件里 qapm_uuid 属性的值存在，该值将被作为构建 id，否则插件会随机生成一个构建 id。

qapm-plugin 2.39及之前版本在编译 App 的过程中会报 IO 错误：`java.io.FileNotFoundException, qapm.properties (No such file or directory)` 。



该报错仅在编译期间产生，不会影响 App 运行。

功能配置
网络监控

最近更新时间：2024-05-14 12:36:06

开启功能

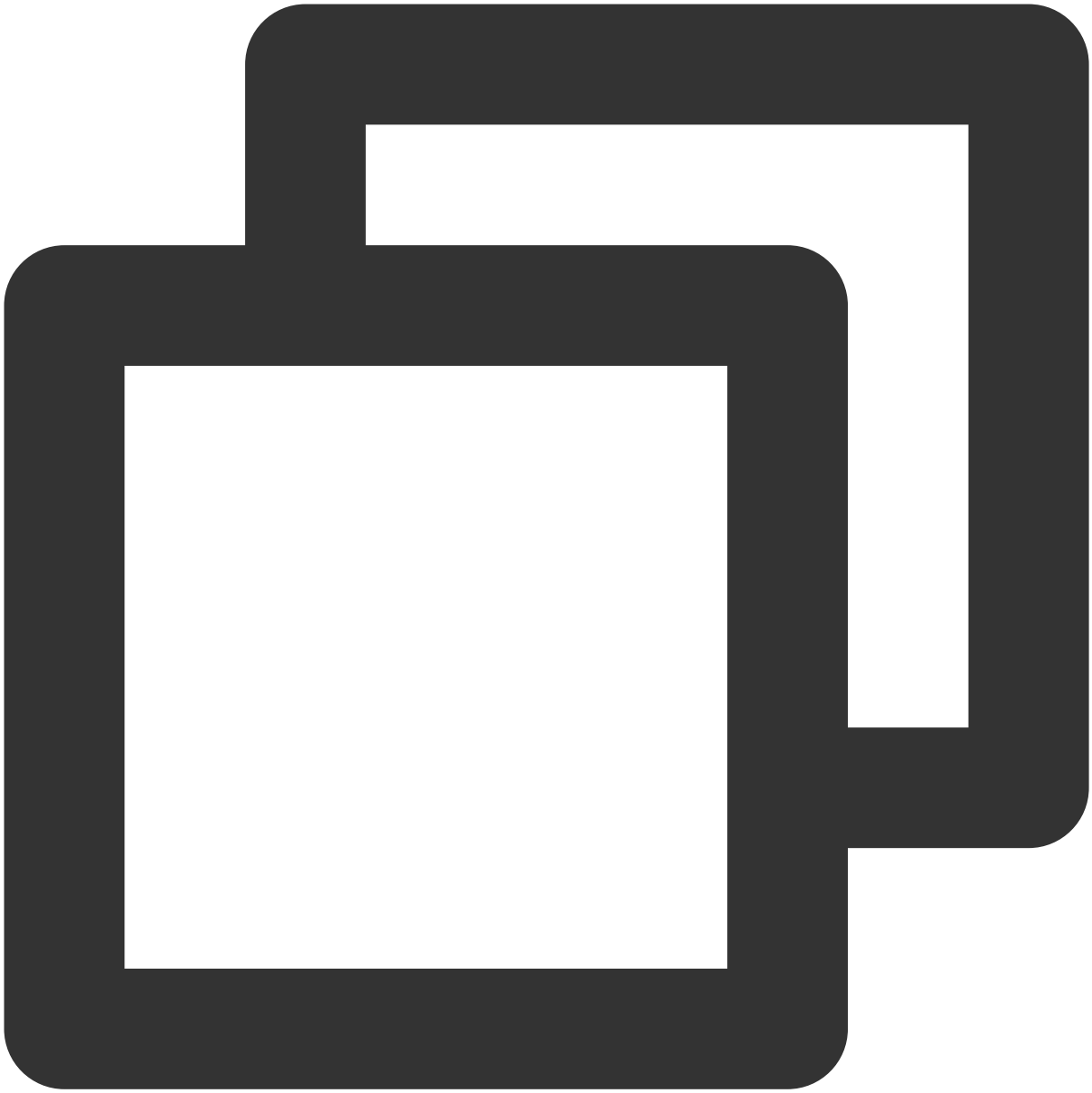
网络监控需要使用 qapm-plugin 插件进行插桩才可使用，默认会插在网络层的各个出入口。

前提条件

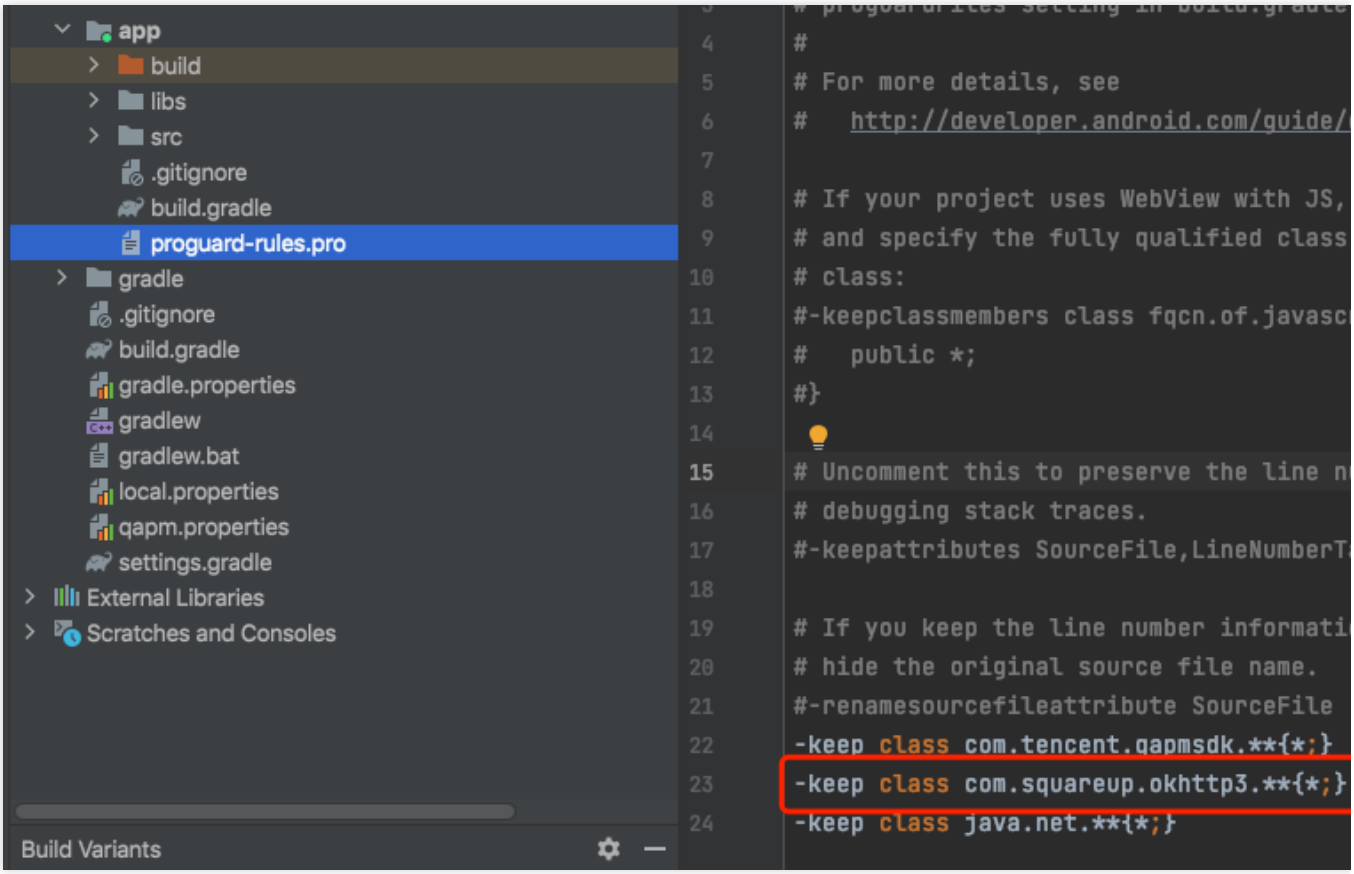
需要在 终端性能监控 > 应用管理 > 白名单 中添加用户白名单或设备白名单，白名单用于初始化 SDK。
在 app 级别的 build.gradle 中配置了 qapm-plugin 插件，参见 集成和初始化。
目前只支持 okhttp3 监控。okhttp3 还依赖 okio 1.14.0 以上版本的库。

配置步骤

在 app 目录的 proguard-rules.pro 文件里增加混淆规则，防止 okttp3 的代码被混淆。



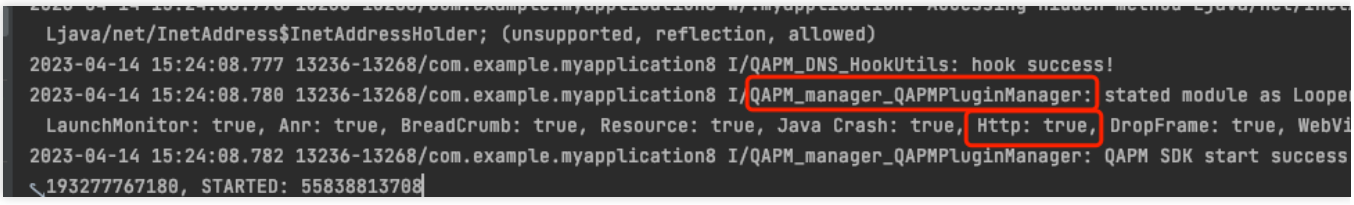
```
-keep class com.squareup.okhttp3.**{*;}
```



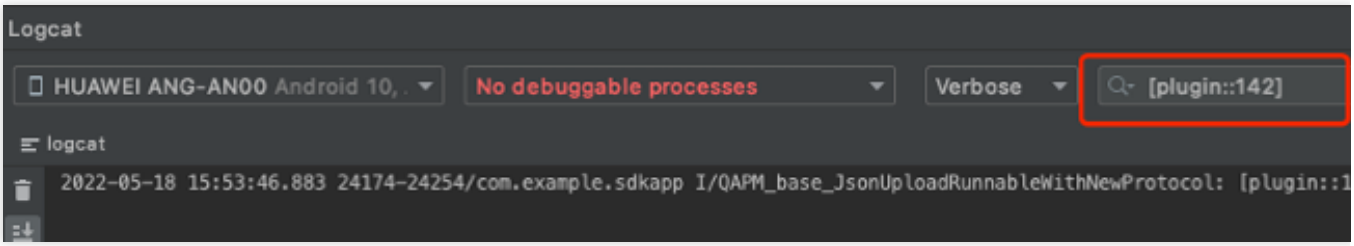
校验功能是否正常

检索 TAG:QAPM_manager_QAPMPluginManager

每次网络请求后1分钟，如打印以下日志，则代表网络数据上报成功：



检索 TAG: [plugin::142]



注意：

需要使用 qapm-plugin 插件进行插桩才可用，否则无效。

SDK 只负责抓取网络请求的相关信息，问题数据由后台分析，如慢请求（请求时间大于xxs），网络错误（请求响应码 > 400）。

数据在 终端性能监控 >网络 > 慢请求和错误请求列表 中查看。

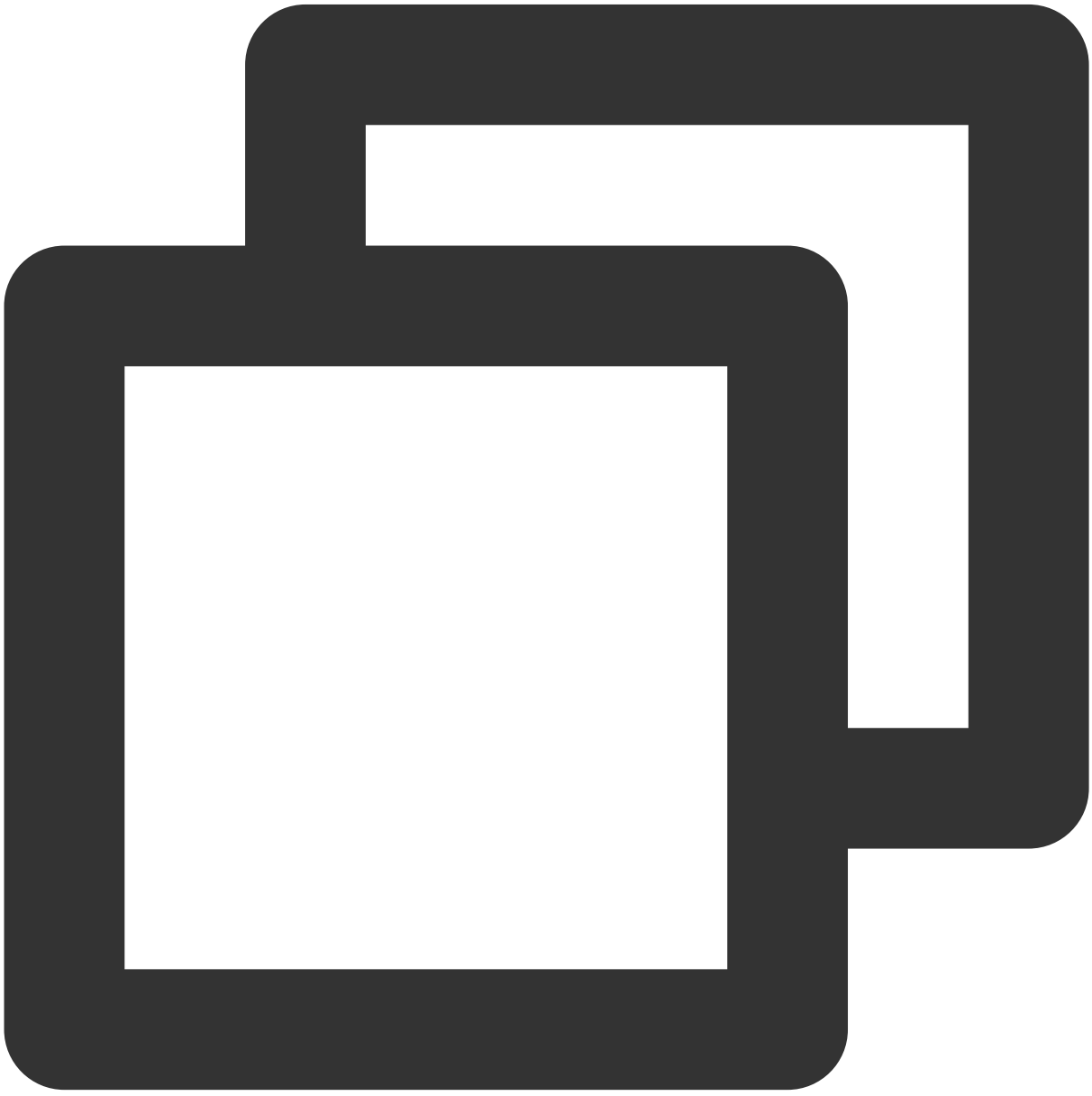
若没有配置正确的白名单，SDK 不会开启网络监控。

WebView、JsError、Web 网络监控

最近更新时间：2024-05-14 12:36:06

开启功能

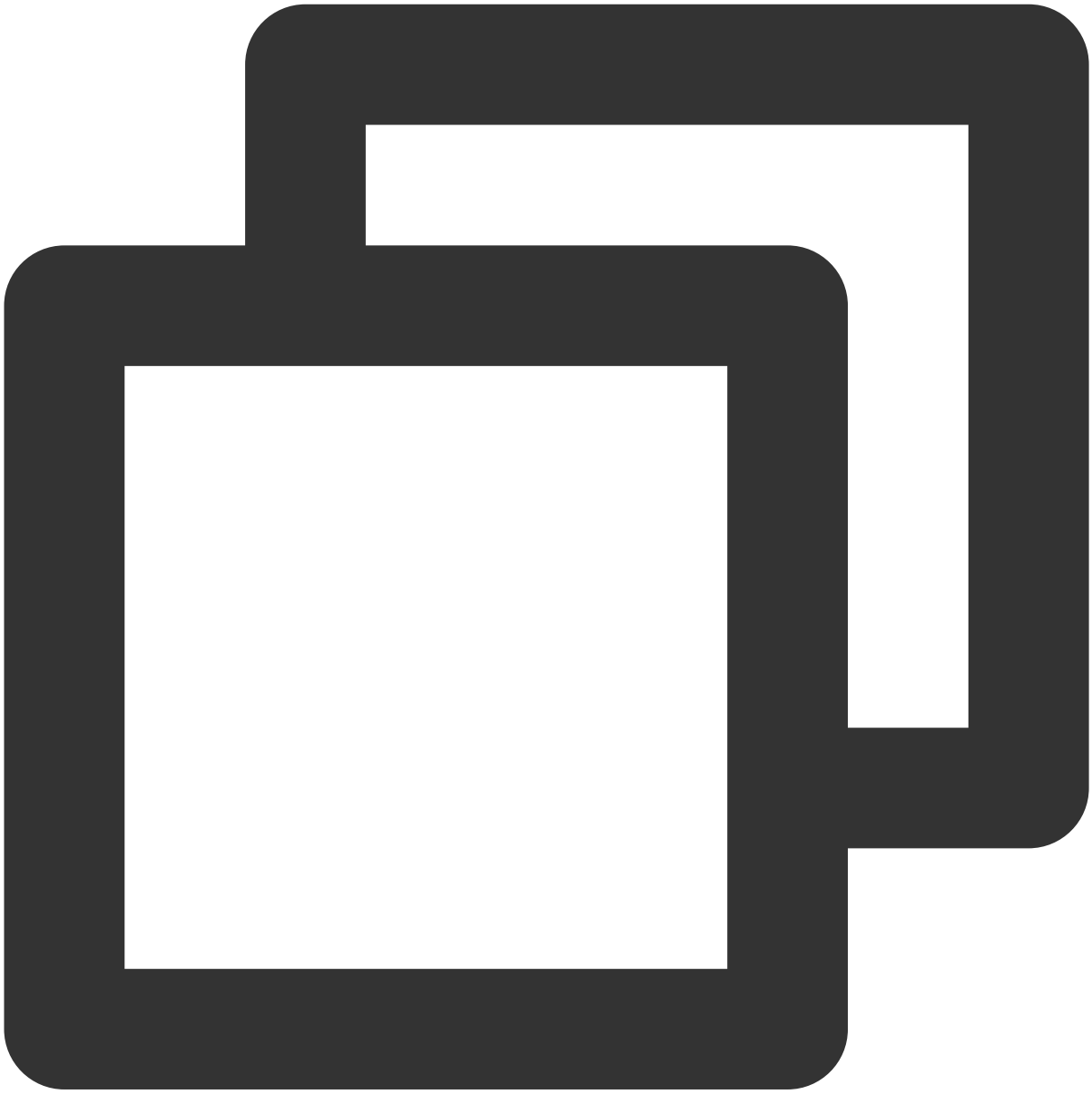
初始化需要开启 WebView、JsError、Web 网络监控，如下是在 Stable 的基础上开启当前栏目的三个功能。代码如下：



```
QAPM.beginScene(QAPM.SCENE_ALL, QAPM.ModeStable | QAPM.ModeWebView | QAPM.ModeJsErr
```

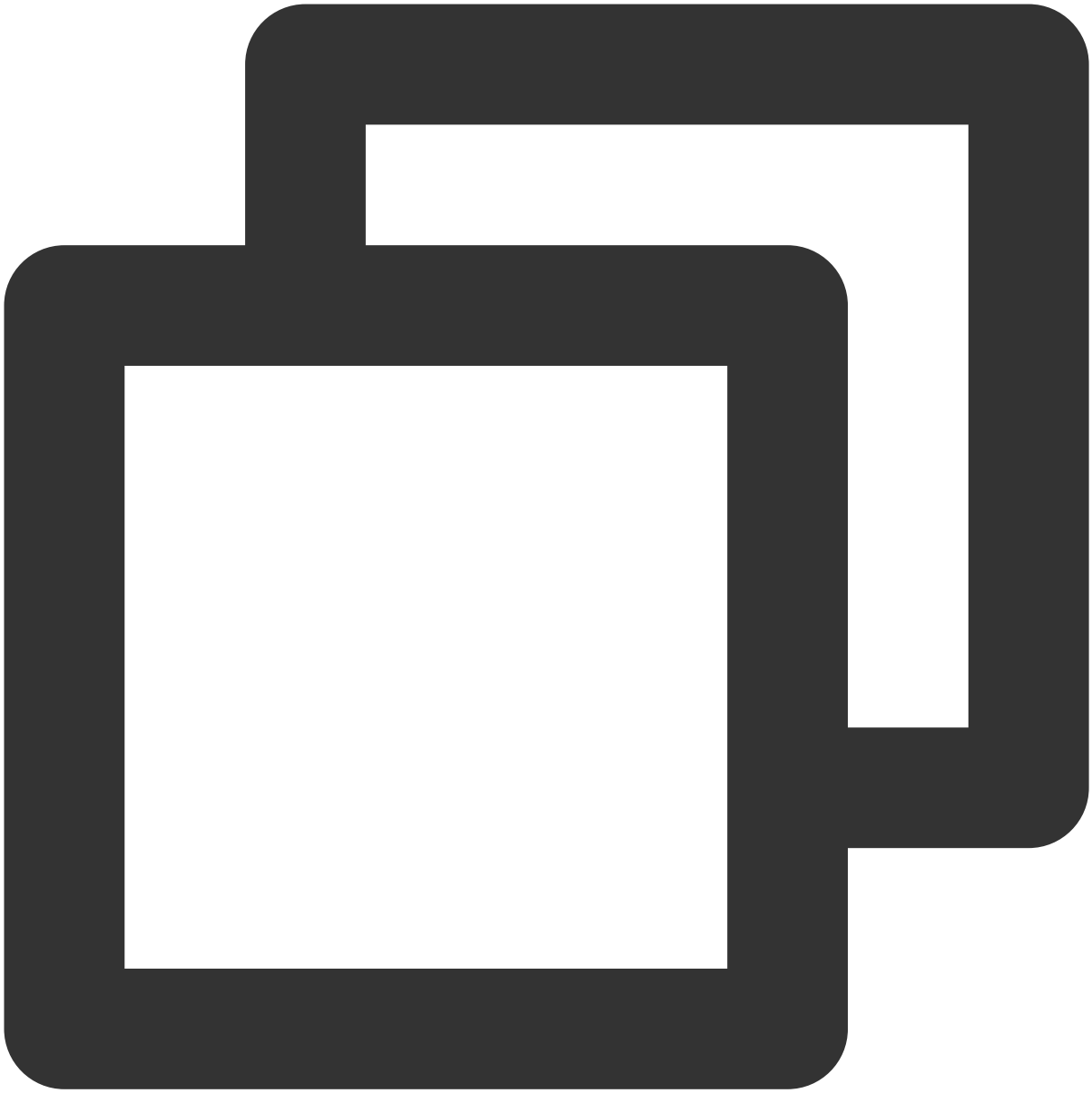
除此之外，还需要配置以下代码：

WebView 监控需要开启与 JavaScript 交互，在 WebView 初始化时调用如下代码开启：



```
WebSettings webSetting = webView.getSettings();
webSetting.setJavaScriptEnabled(true);
```

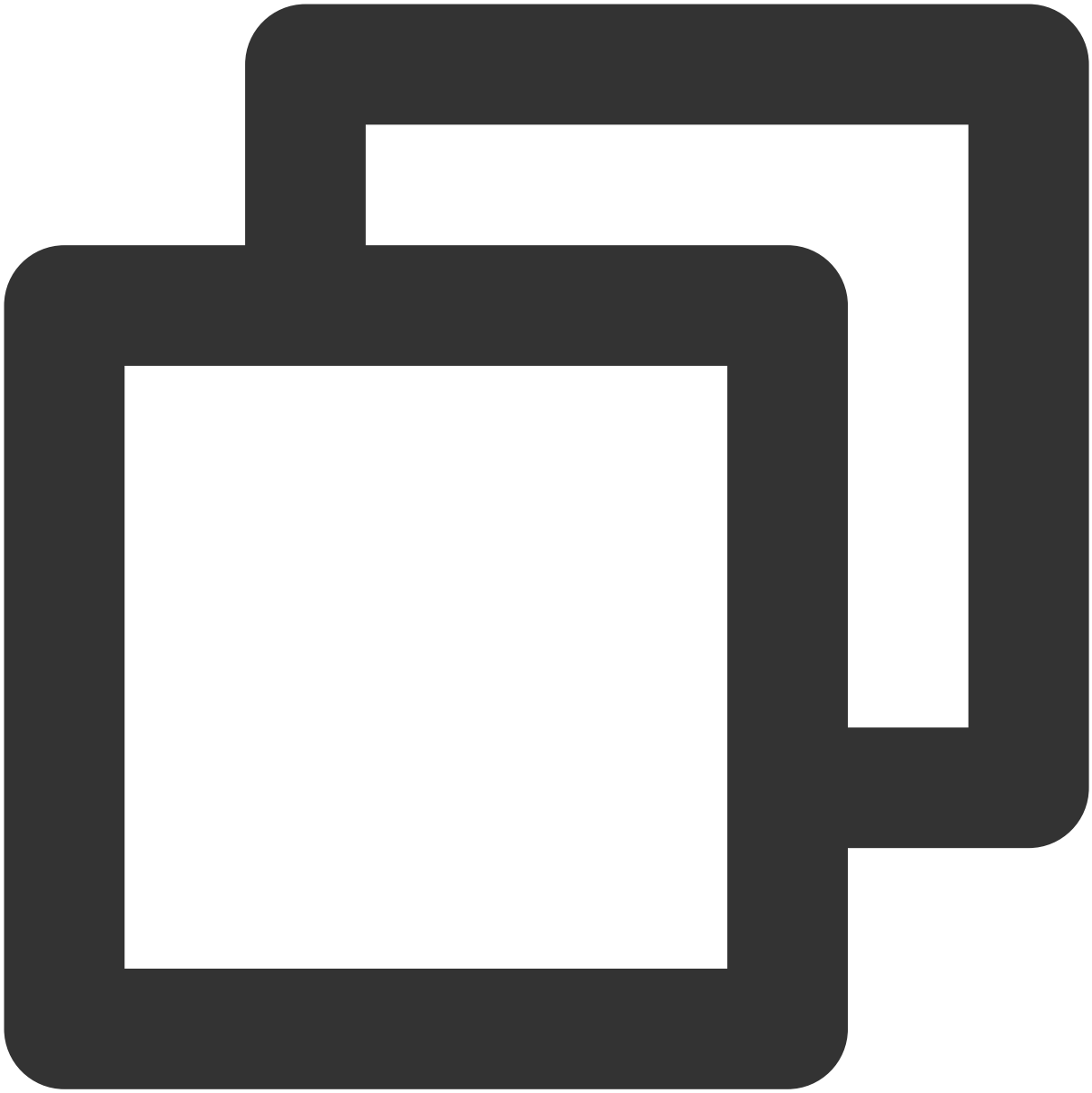
在 WebView 初始化完成之后加入 Java 与 JS 之间的调用接口通道，目的是让 JS 层获取到 Java 层的一些配置信息：



```
webView.addJavascriptInterface(QAPMJavaScriptBridge.getInstance(),"QAPMAndroidJsBri
```

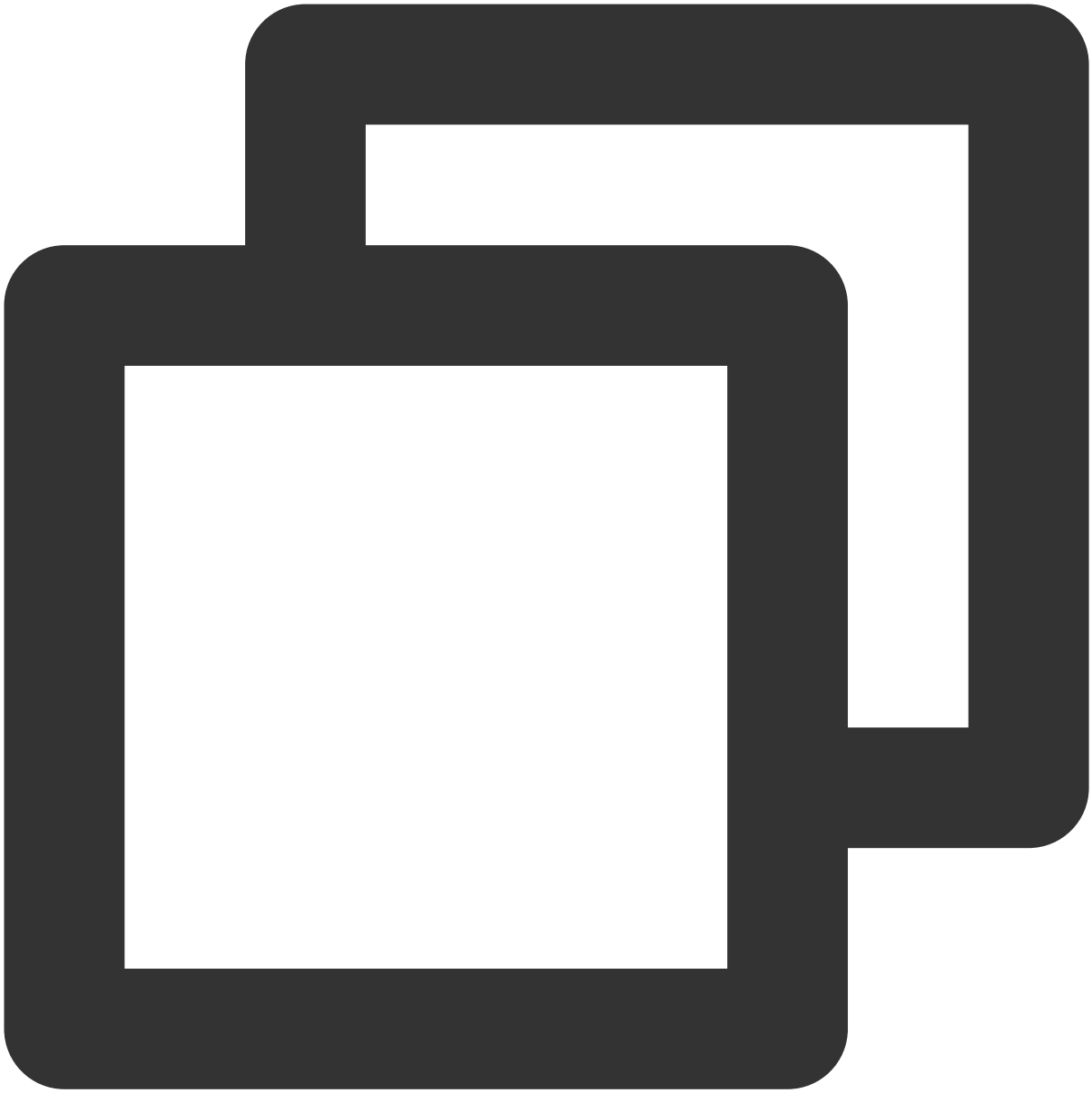
在 WebView 的 shouldInterceptRequest 代码里加入以下方法，用于拦截 web-sdk 并改用本地 SDK 资源，**请确保在该回调中的最早地方调用以下代码**。

如果是 x5 请使用以下代码：



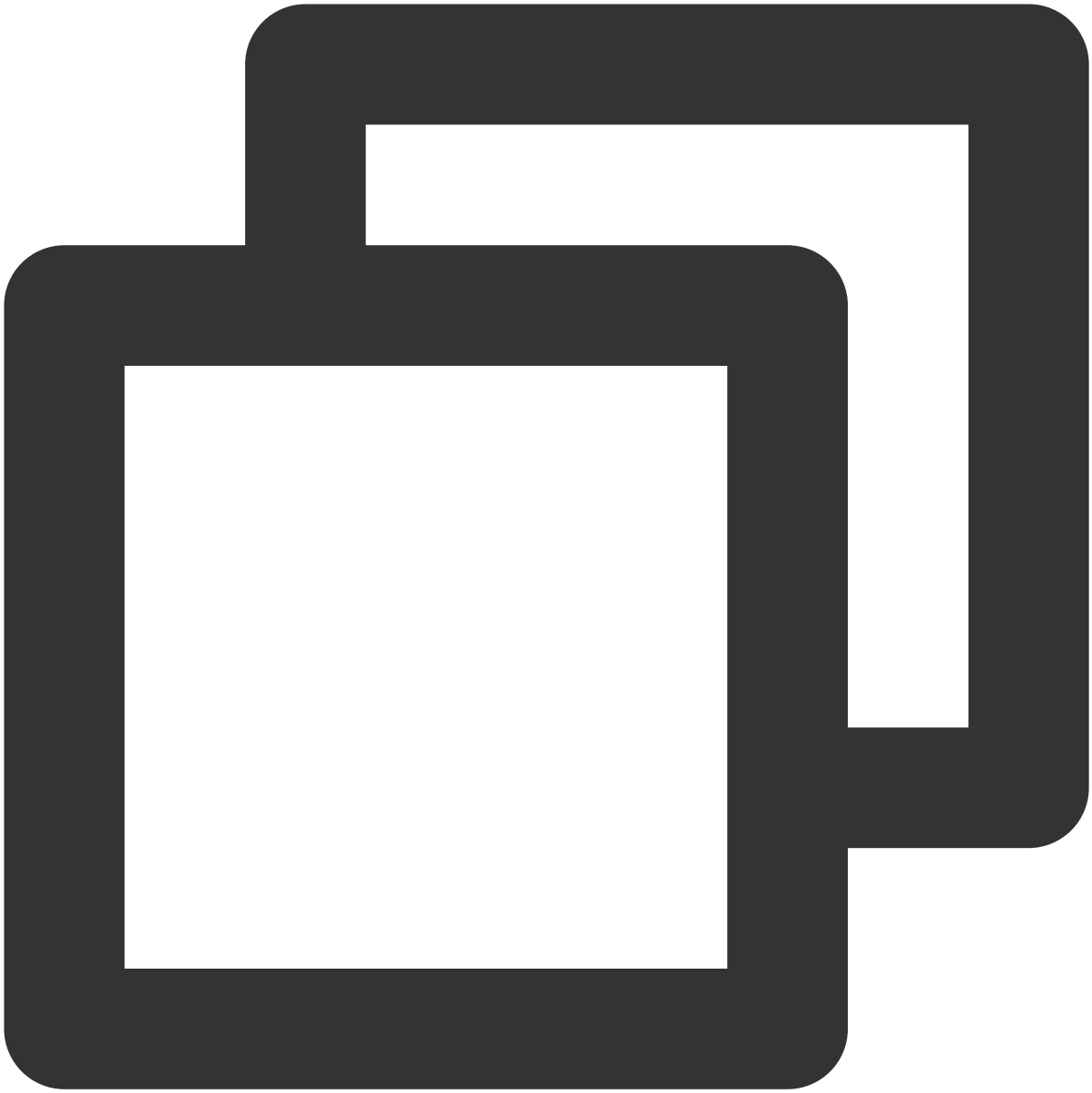
```
@Overridepublic
public WebResourceResponse shouldInterceptRequest(WebView webView, String s) {
    Object response =QAPMJavaScriptBridge.getInstance().shouldInterceptRequestWithX
    if (response != null) {
        return (WebResourceResponse)response;
     }
     return super.shouldInterceptRequest(webView,s);
     }
```

如果是原生 WebView 请使用以下代码:



```
@Overridepublic
public WebResourceResponse shouldInterceptRequest(WebViewwebView, String s) {
    WebResourceResponse response =QAPMJavaScriptBridge.getInstance().shouldIntercep
    if (response != null) {
        return response;
    }
    return super.shouldInterceptRequest(webView, s);
  }
```

在 WebView 的 onPageFinished 代码里加入以下方法，用于注入 JS 脚本：

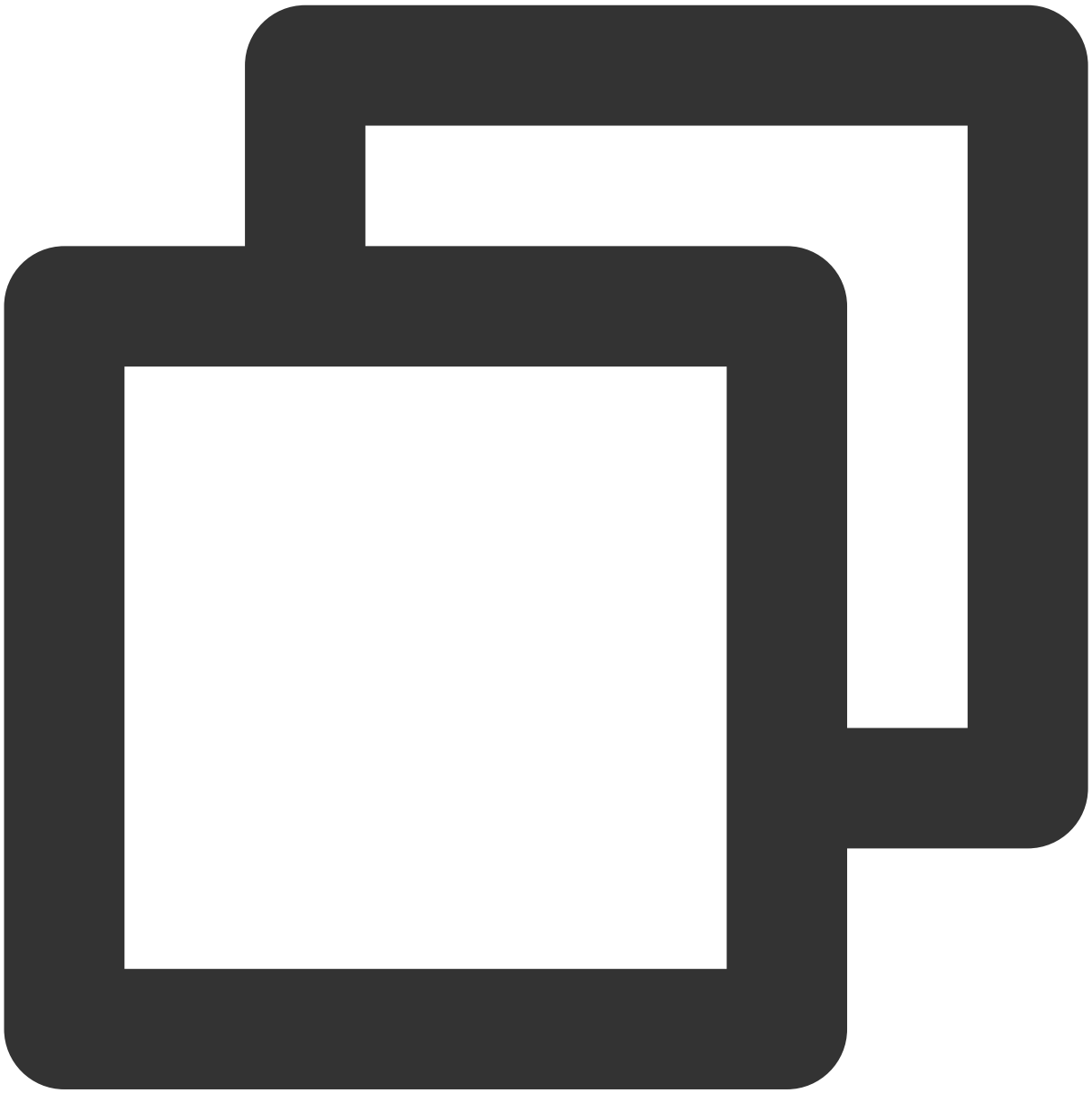


```
webView.setWebViewClient(new WebViewClient(){
    @Override
    public void onPageFinished (WebView view, String url) {
        super.onPageFinished(view, url);
        QAPMJavaScriptBridge.getInstance().initFileJS(view);
    }
});
```

校验功能是否正常

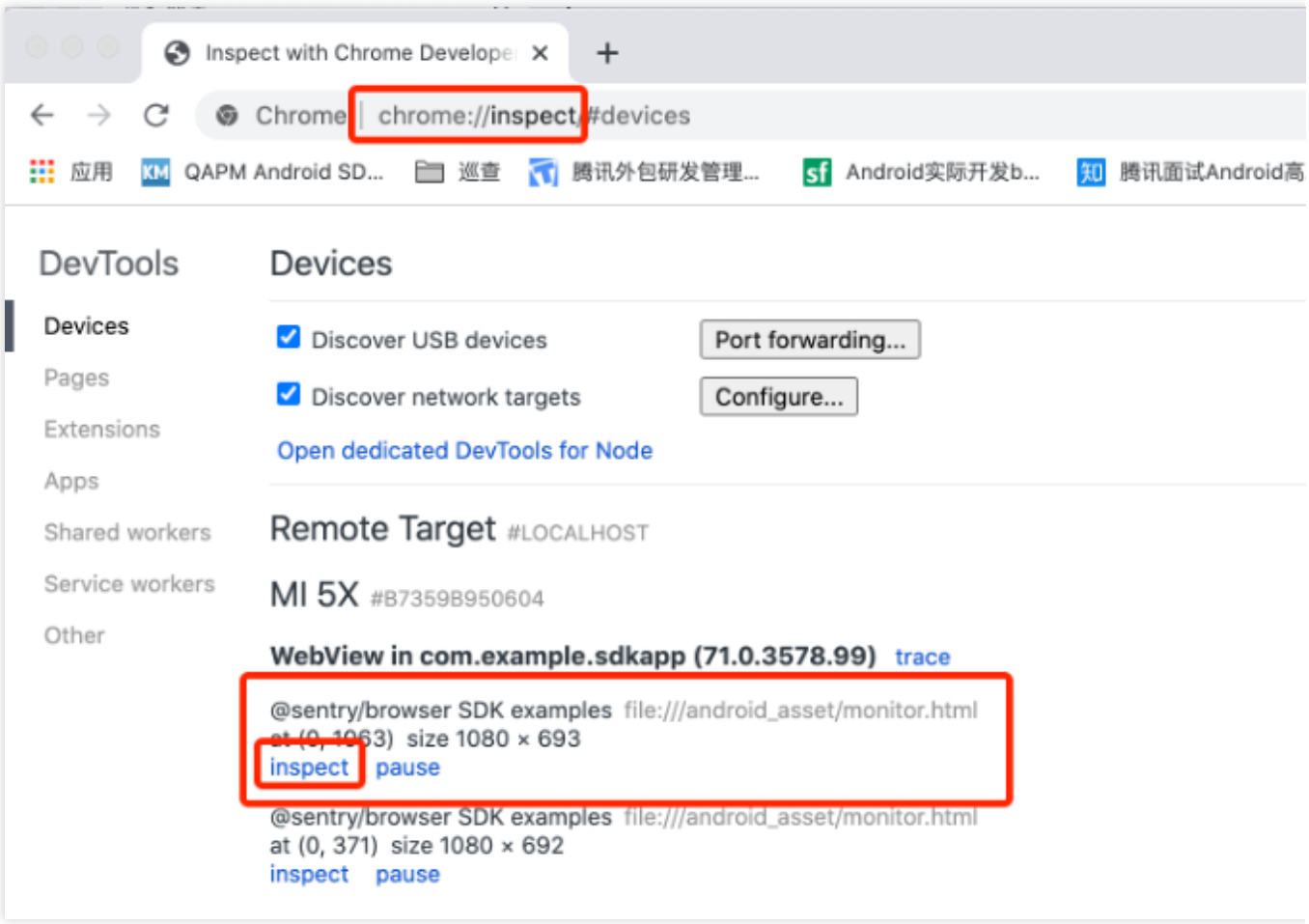
原生 WebView、JsError 监控：

1. 代码中加入以下代码（用于远程调试）。

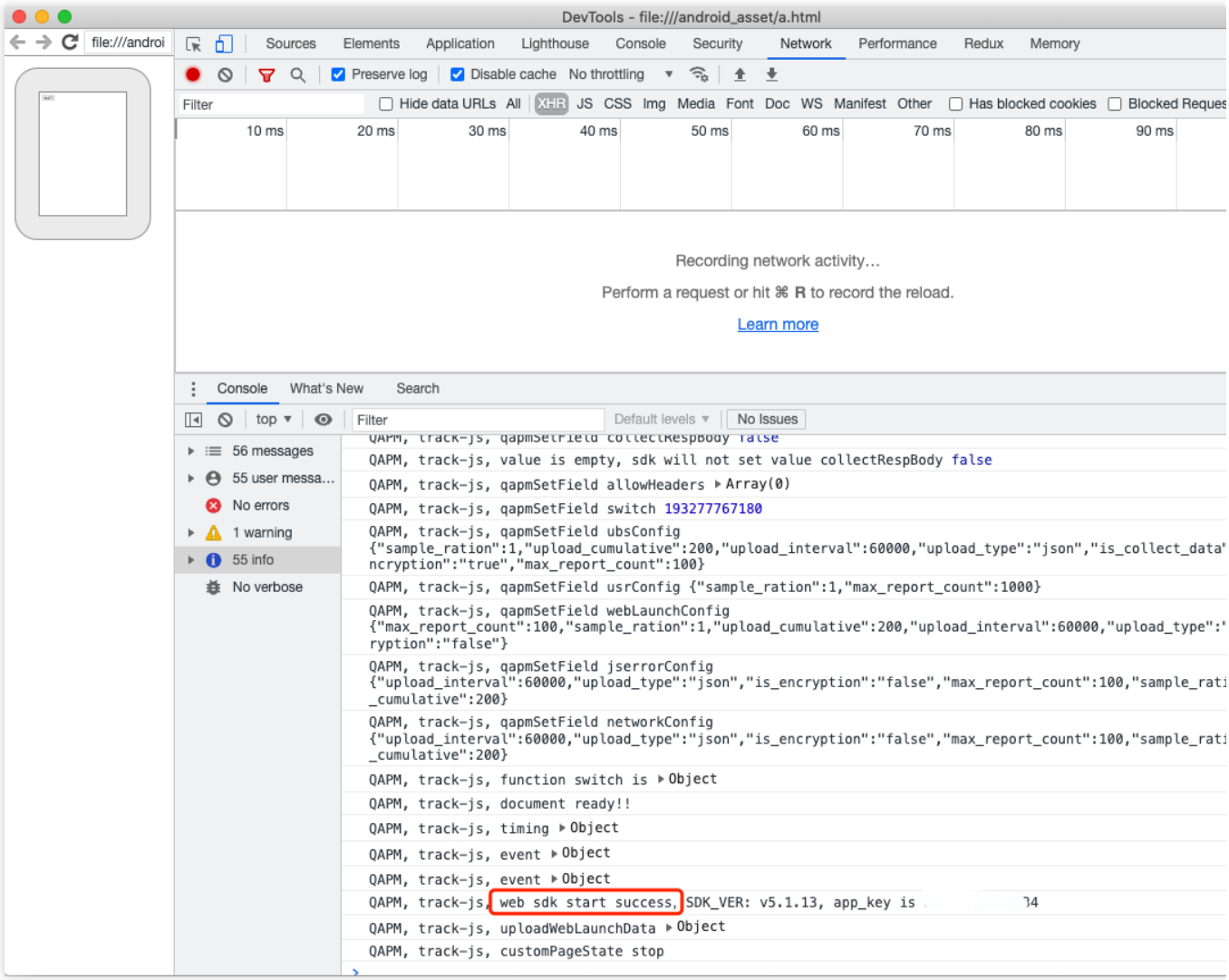


```
WebView.setWebContentsDebuggingEnabled(true);
```

2. 打开谷歌浏览器，地址栏输入 `chrome://inspect` ，在出现的设备中单击 **inspect**。



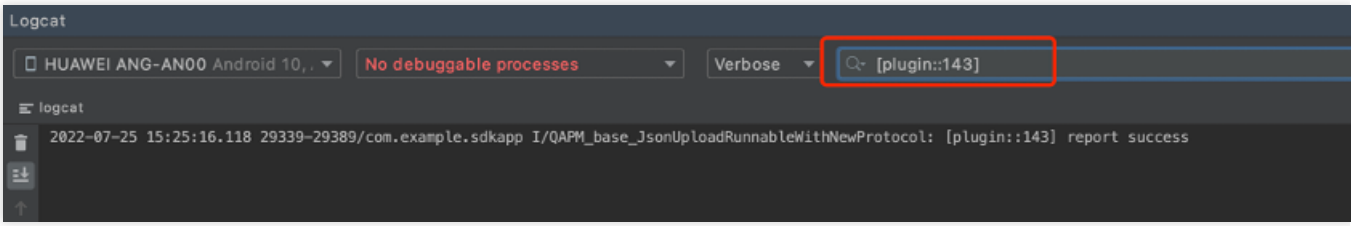
3. 进入后找到 Console 模块查询日志，如出现 `web start success ,vxxx` ，则代表 WebSDK 注入成功。



4. 检测各个功能是否上报正常，以 JsError 上报为例，如下：

检索 TAG: [plugin::143]

每次触发 JsError 错误，如打印以下日志，则代表 JsError 数据上报成功。



其余检索 TAG 分别如下：

页面加载：plugin::141（每次 Web 页面加载完成后即会上报）。

网络请求：plugin::154（出现错误网络和慢请求时会上报）。

注意：

1. 如需查看 WebView 监控是否正常需要通过 chrome 等浏览器调试查看。

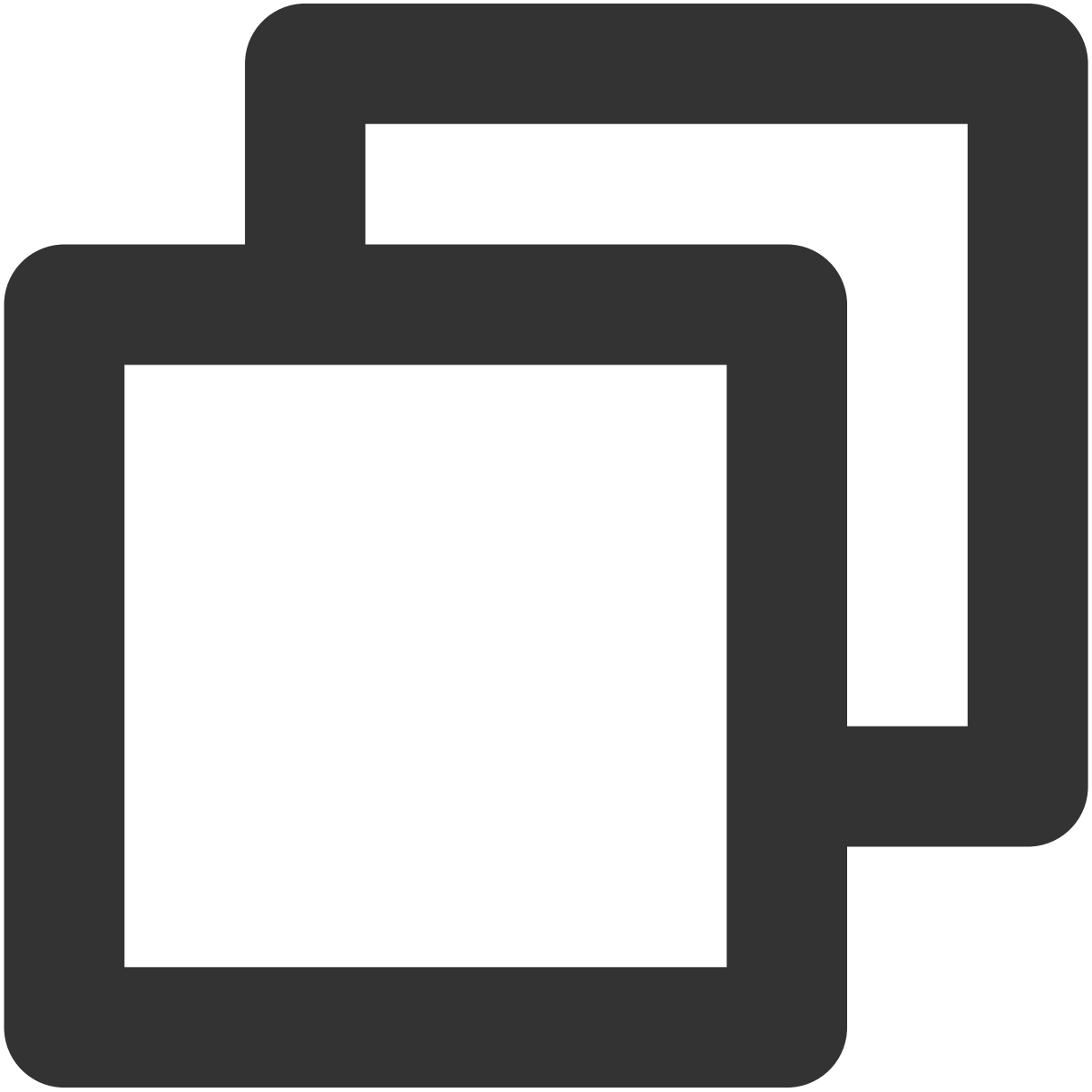
2. 页面加载只有页面加载时长大于3.5s才可在问题个例详情里查看。
3. 网络请求只有在网络错误和网络慢时才会上报。

Crash、ANR 监控

最近更新时间：2024-05-14 12:36:06

开启功能

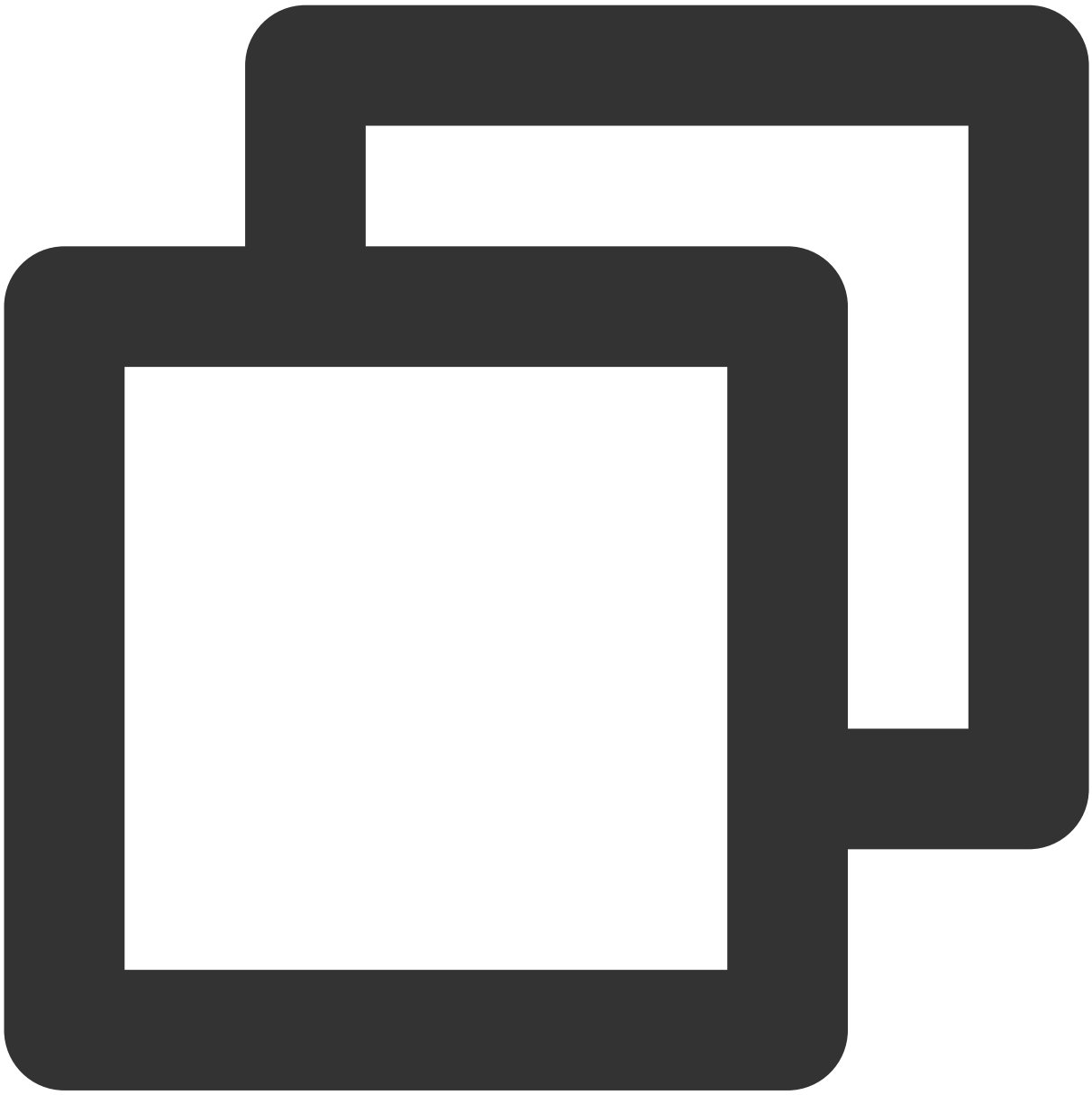
初始化需要开启 Crash、ANR 监控，该监控会默认监控 Crash 和 ANR 信息。



```
// ModeStable模式默认包含了Crash、ANR监控
```

```
QAPM.beginScene(QAPM.SCENE_ALL, QAPM.ModeStable);
```

QAPM 提供了相关接口，如有额外的需要，可以发生了 Crash 或者 ANR 时，上传用户自定义的日志文件，示例如下：

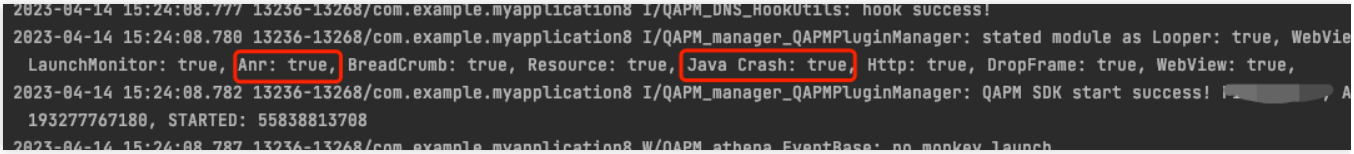


```
QAPM.setProperty(QAPM.PropertyExtraDataListener, new IExtraDataListener() {
    // 发生ANR时会走这个回调
    @Override
    public List<String> onAnrExtraFileHandler() {
        List<String> files = new ArrayList<>();
        File[] fileArray = new File("xxxx").listFiles(); //xxx处请填写文件夹名称
```

```
            for (File file : fileArray) {
                files.add(file.getAbsolutePath());
            }
            return files;
        }
        // 发生Crash时会走这个回调
        @Override
        public List<String> onCrashExtraFileHandler() {
            List<String> files = new ArrayList<>();
            File[] fileArray = new File("xxxx").listFiles();//xxx处请填写文件夹名称
            for (File file : fileArray) {
                files.add(file.getAbsolutePath());
            }
            return files;
        }
    });
```

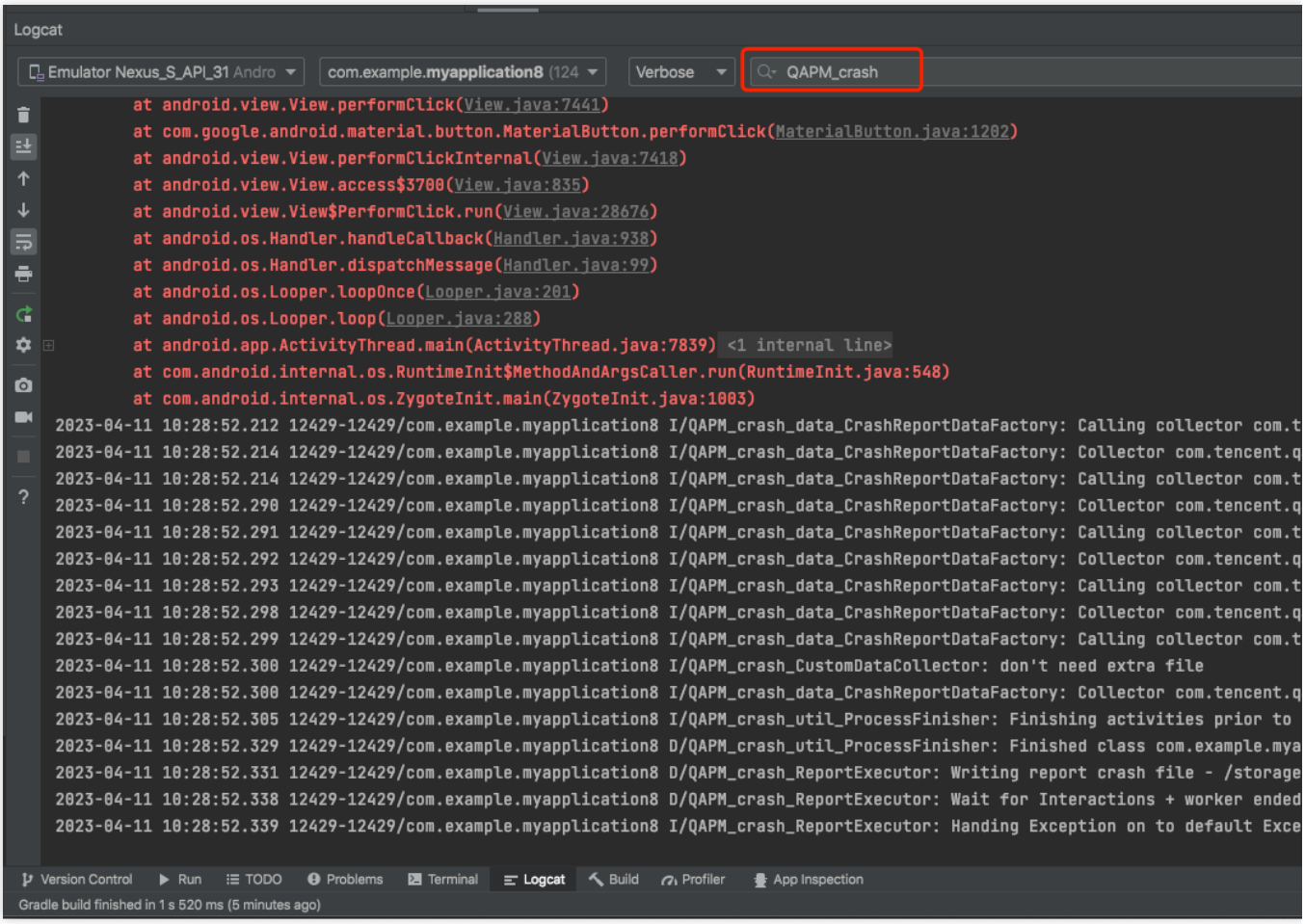
校验功能是否正常

检索 TAG: QAPM_manager_QAPMPluginManager



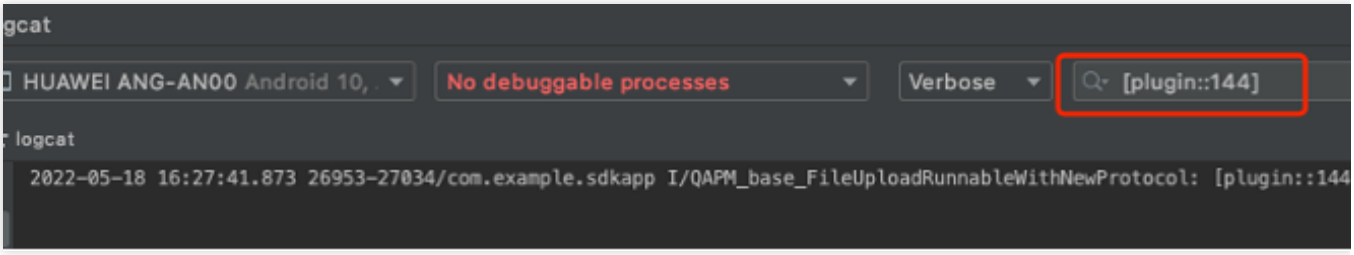
检索 TAG: QAPM_crash

当发生 Crash、Anr 时，打印如下日志，则代表 QAPM 正常收集了此次异常：



检索 TAG: plugin::144

当打印如下日志，则代表 QAPM 将此次异常上报成功，此处举例 JavaCrash 的上报情况：



其他 crash 检索 TAG 分别如下：

ANR： [plugin::140]。

NativeCrash： [plugin::146]。

说明：

为避免出现卡死的情况，接口回调里的逻辑请尽量简单明了。

上传的文件大小限制为20MB，大于限制则不上传，请选择认为有帮助的日志文件。

Crash 可以在移动监控的崩溃页面查看，ANR 可在总览页面中查看 ANR 率。

卡顿、帧率监控

最近更新时间：2024-05-14 12:36:06

前提条件

已完成 集成与初始化。

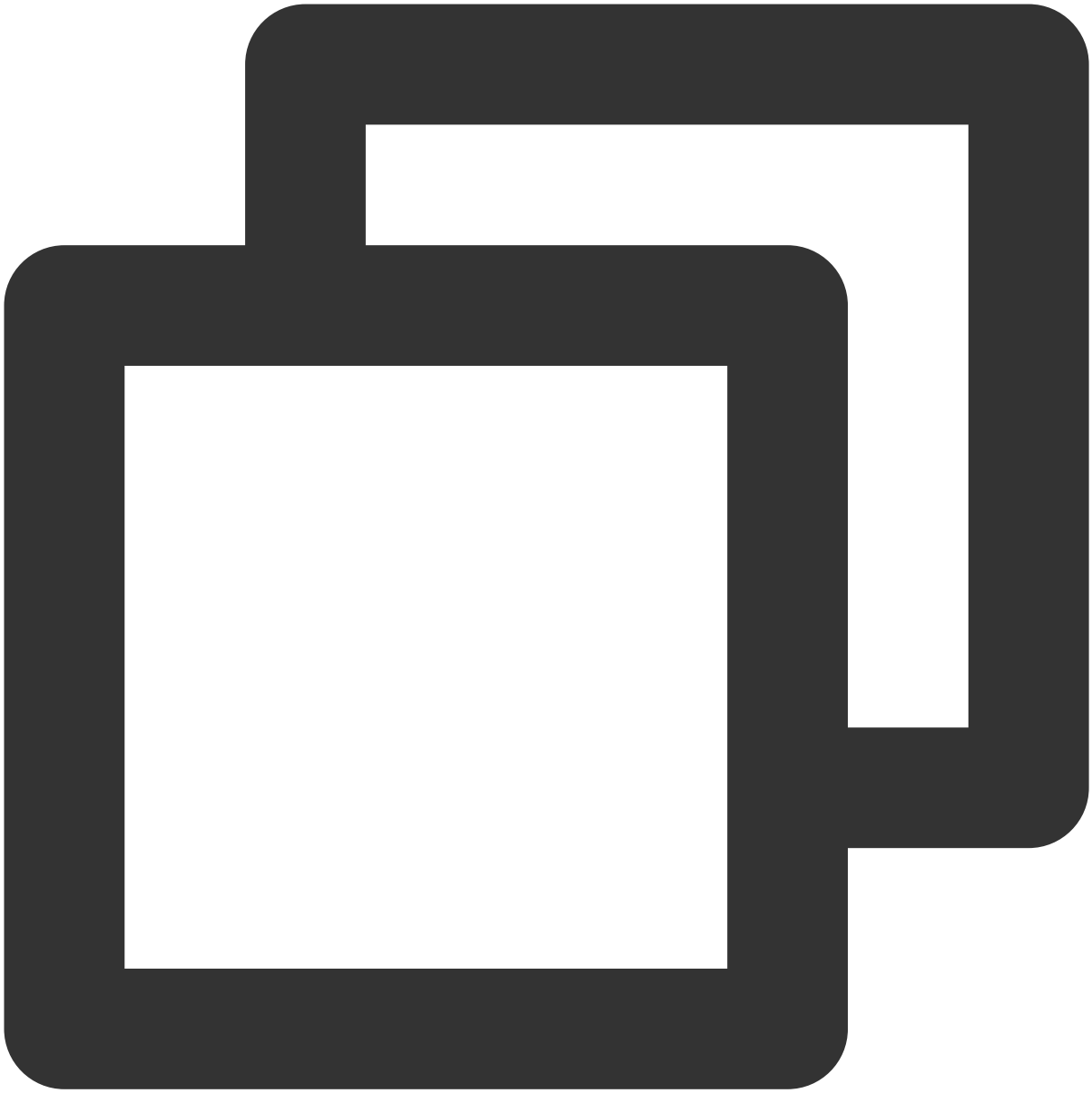
功能配置

开启监控

初始化需要开启卡顿监控。卡顿无需埋点，而掉帧率需要额外埋点，建议打点在滑动列表上，如（ListView、GridView、RecyclerView 等）。

掉帧率埋点

在每次滑动前调用 QAPM.beginScene("xxx滑动", QAPM.ModeDropFrame);
在滑动结束后调用 QAPM.endScene("xxx滑动", QAPM.ModeDropFrame);
一般可以通过重写滑动组件 onScrollStateChanged 方法来实现，示例如下：

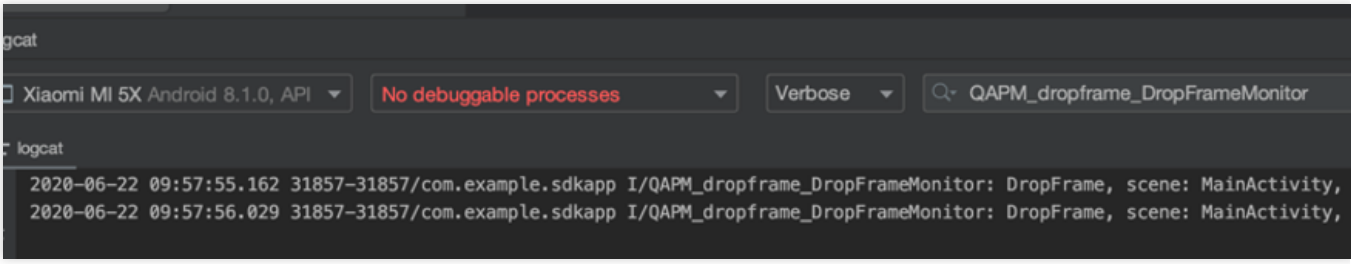


```
@Override
public void onScrollStateChanged(AbsListView view, int scrollState) {
    if (scrollState == AbsListView.OnScrollListener.SCROLL_STATE_IDLE) {
        QAPM.endScene("xxx滑动", QAPM.ModeDropFrame); //xxx滑动名称由您自定义，您可以填写
    } else {
        QAPM.beginScene("xxx滑动", QAPM.ModeDropFrame);//xxx滑动名称由您自定义，您可以填
    }
```

校验功能是否正常

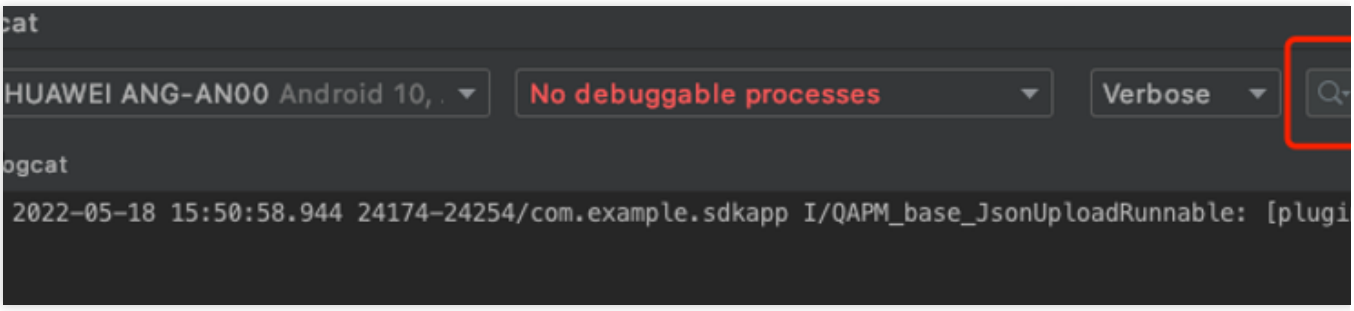
检索 TAG：QAPM_dropframe_DropFrameMonitor

滑动结束（调用 endScene）后，打印出以下日志则代表掉帧率数据已经存入了本地数据库：



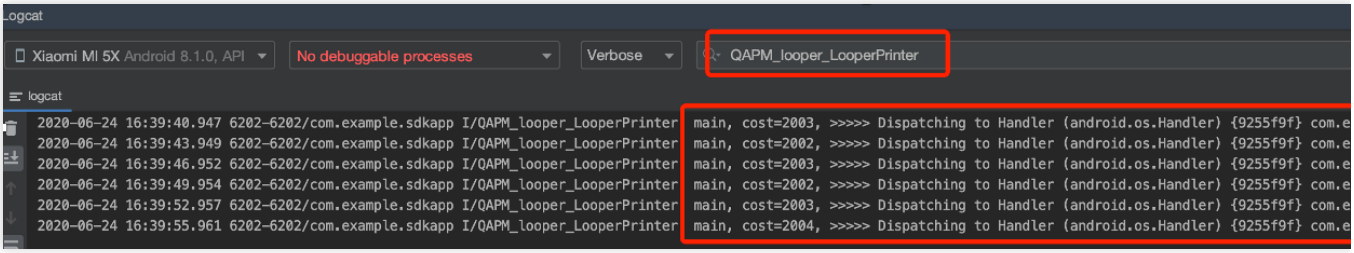
检索 TAG: [plugin::101]

存在 App 本地数据库的掉帧数据将会上报，打印以下数据，代表上报成功：

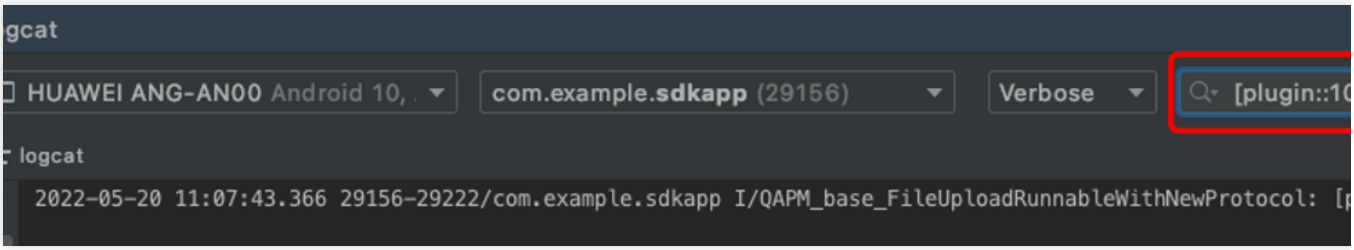


检索 TAG: QAPM_looper_LooperPrinter

如打印以下日志，则代表卡顿监控正常：



如打印以下日志，则代表卡顿上报正常：


gcat
HUAWEI ANG-AN00 Android 10, .
com.example.sdkapp (29156)
Verbose
[plugin::1
logcat
2022-05-20 11:07:43.366 29156-29222/com.example.sdkapp I/QAPM_base_FileUploadRunnableWithNewProtocol: [

启动监控

最近更新时间：2024-05-14 12:36:06

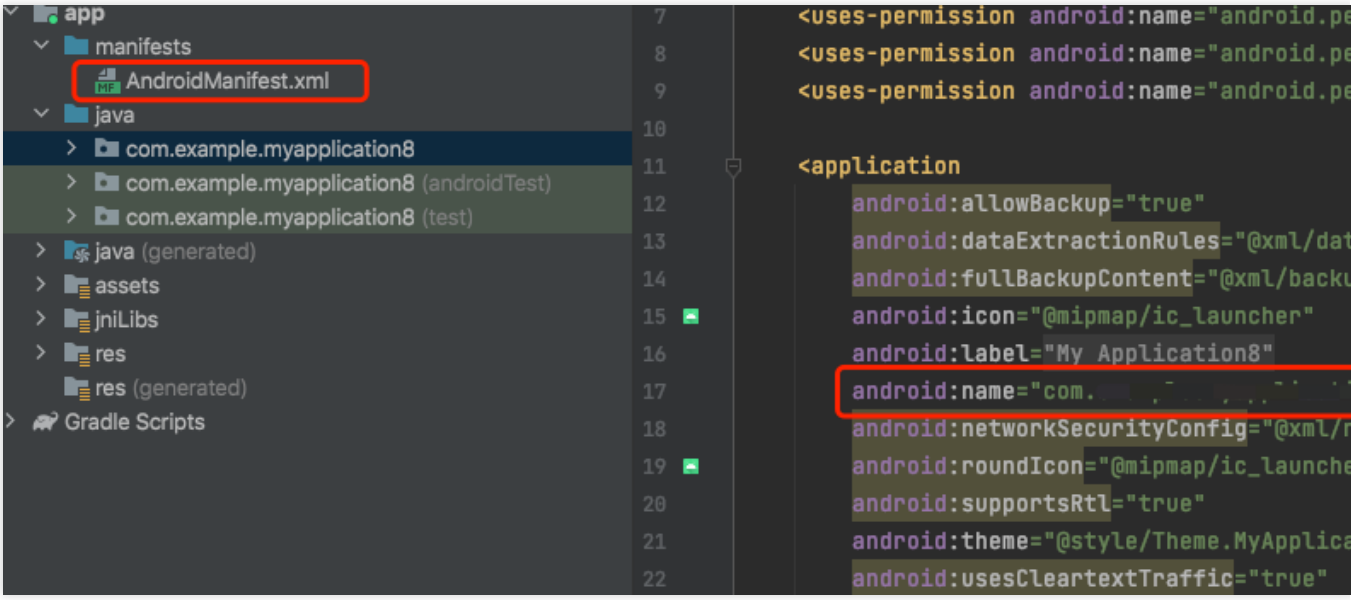
启动监控需要使用 qapm-plugin 插件在编译期间进行插桩操作，默认插桩点为 Application 与 Activity 的各个生命周期。在 App SDK 统计默认的启动耗时为 Application 的 attachBaseContext 到第一个 Activity 的 onResume 结束。

前提条件

在 App 级别的 build.gradle 中配置了 qapm-plugin 插件。

配置步骤

1. 需要手动添加一个 Application 的子类，例如 BaseApplication（名称不作要求，子类可以不用实现任何方法，可以不用添加任何属性）。
2. 在 AndroidManifest.xml 文件的 application 的节点添加 android:name 属性，属性的 value 为“包名+Application 子类的类名”。



额外打点

如果想统计启动区间内的某些方法的耗时，则需要额外的打点，示例如下：

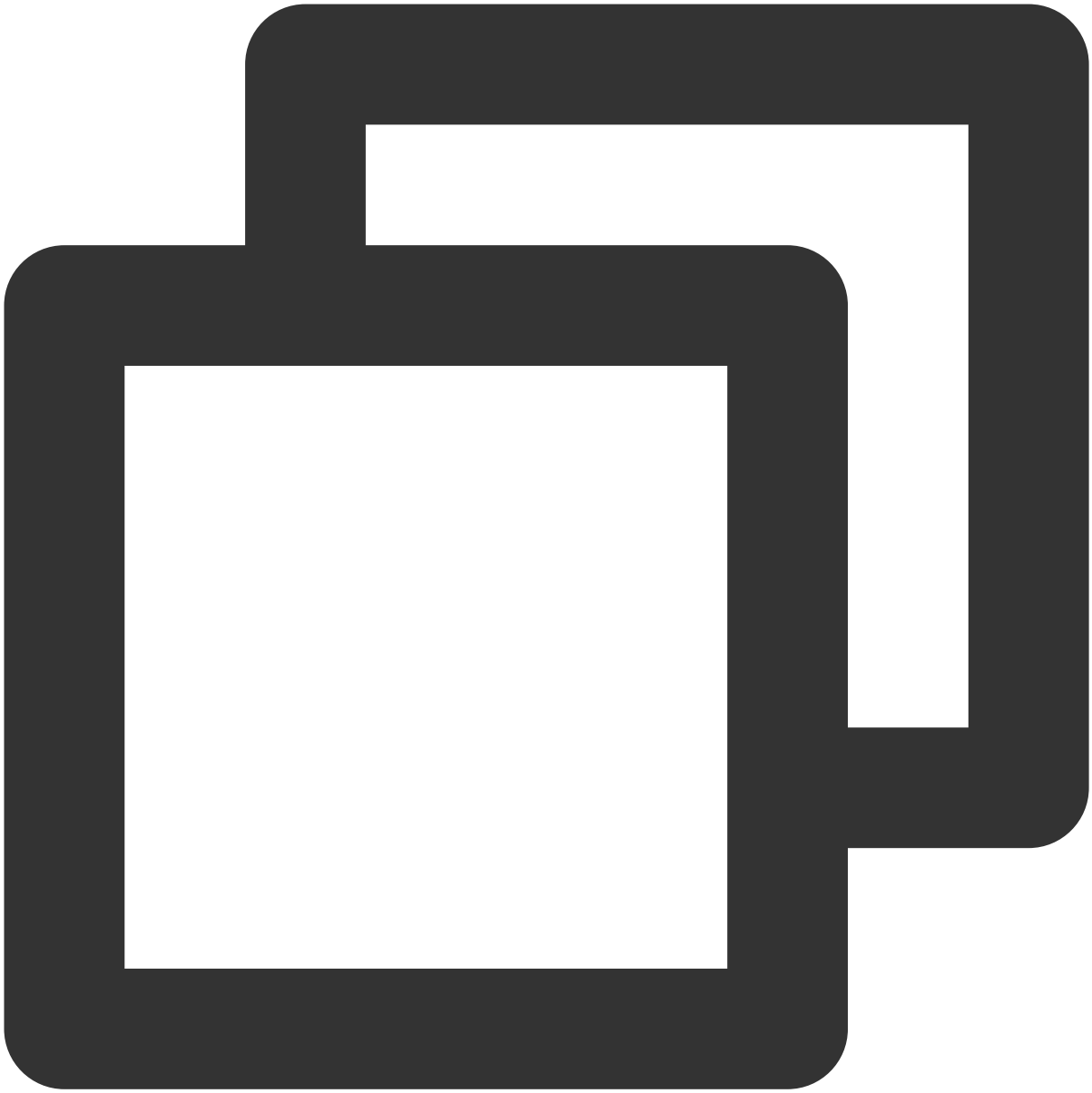
```
public class BaseApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        b();
        try {
            Thread.sleep( millis: 2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        c();
    }

    private void b() {
        QAPM.beginScene(StageConstant.QAPM_APPLAUNCH,  extraInfo: "b方法名", QAPM.ModeResource);
        /**
         * 业务逻辑
         */
        QAPM.endScene(StageConstant.QAPM_APPLAUNCH,  extraInfo: "b方法名", QAPM.ModeResource);
    }

    private void c() {
        QAPM.beginScene(StageConstant.QAPM_APPLAUNCH,  extraInfo: "c方法名", QAPM.ModeResource);
        /**
         * 业务逻辑
         */
        QAPM.endScene(StageConstant.QAPM_APPLAUNCH,  extraInfo: "c方法名", QAPM.ModeResource);
    }
}
```

参见代码：

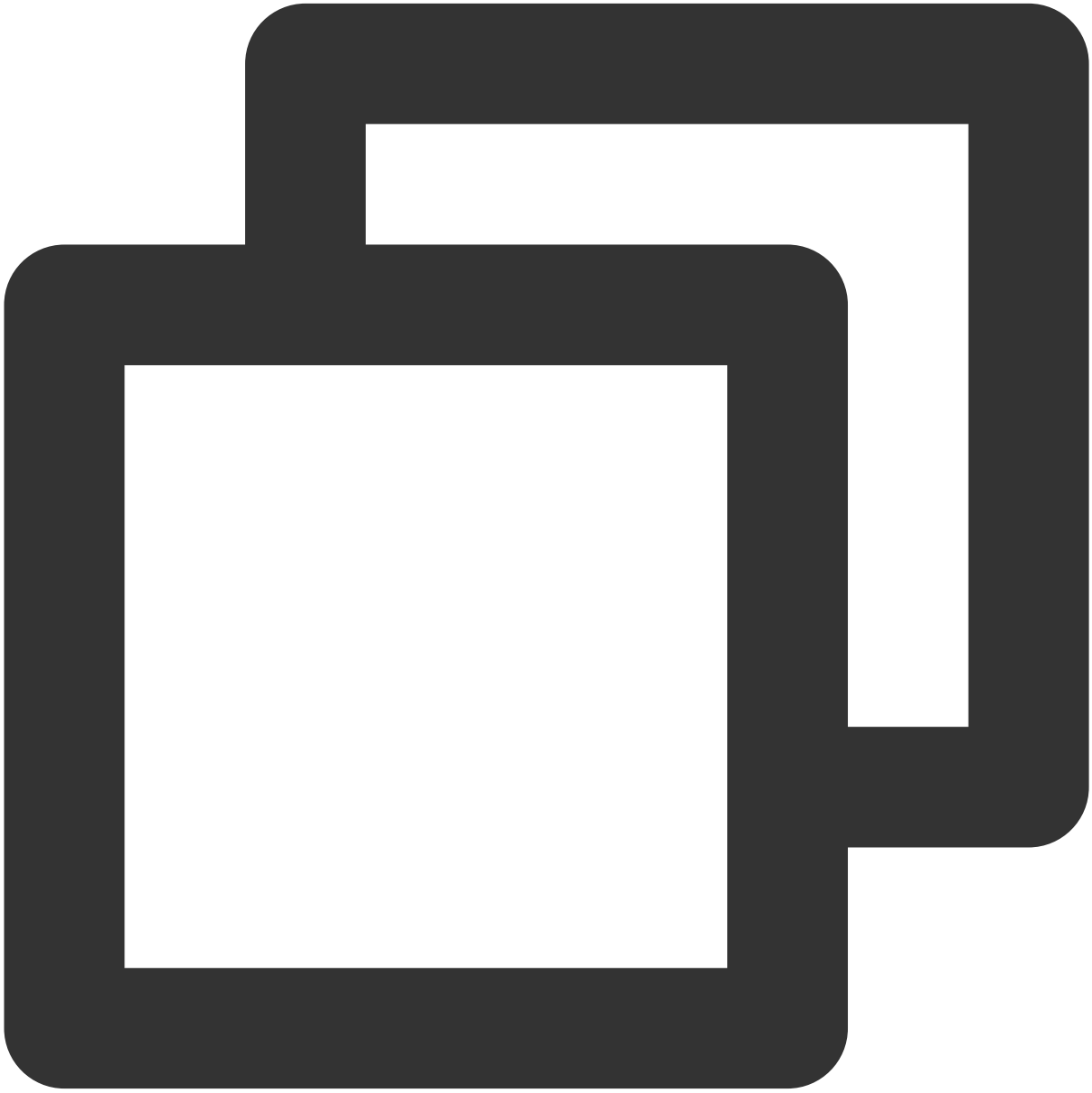


```
QAPM.beginScene(StageConstant.QAPM_APPLAUNCH, "xxx方法名", QAPM.ModeResource);
/**业务逻辑*/
QAPM.endScene(StageConstant.QAPM_APPLAUNCH, "xxx方法名", QAPM.ModeResource);
```

如果想自定义启动的结束点，则需要在第一个 Activity 调用 onResume 的20s内额外打点，示例如下：

```
/**
 * 需要自定义结束点的用户需要在onResume之后的20s内，否则以APM的结束点为准
 */
QAPM.endScene(StageConstant.QAPM_APPLAUNCH, QAPM.ModeResource);
```

参见代码：

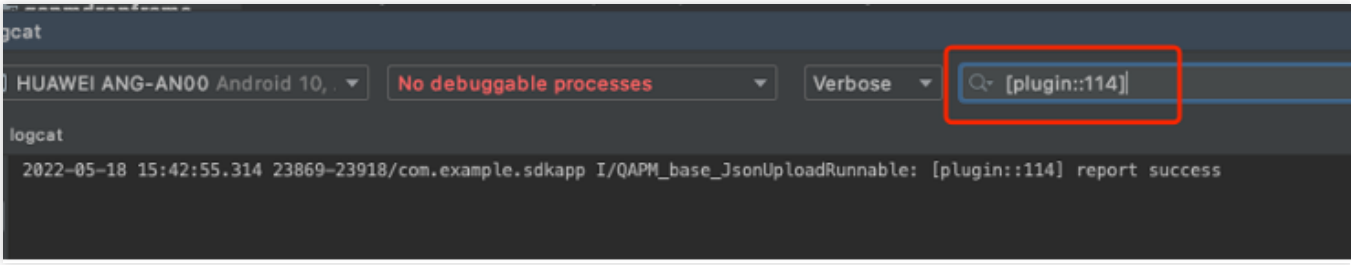


```
/**
 * 需要自定义结束点的用户需要在onResume之后的20s内，否则以APM的结束点为准
```

```
 */
QAPM.endScene(StageConstant.QAPM_APPLAUNCH, QAPM.ModeResource);
```

校验功能是否正常

每次启动后20s或者切换到后台，如打印以下日志，则代表启动指标数据上报成功。
检索 TAG: [plugin::114]



注意：
需要使用 qapm-plugin 插件进行插桩才可用，并且需要手动增加一个 Application 的子类。否则无效。
启动总耗时大于2.5s才会上报个例数据。
启动的问题数据可以在 终端性能监控 > 启动 > 问题列表 查看。

计算方式

冷启动：

应用自设备启动后或系统终止应用后的首次启动。
Android 计算方式：mainActivityOnResume_end - attachBaseContext_start。
iOS 计算方式：第一个页面首帧 UI 上屏时间 - 应用进程创建时间。

首次启动：

应用安装后第一次启动，是特殊的冷启动。
Android 计算方式：mainActivityOnResume_end - attachBaseContext_start
iOS 计算方式：第一个页面首帧 UI 上屏时间 - 应用进程创建时间。

热启动：

在进程及Activity实例还存在(iOS 是在应用在后台且存活)的前提下，应用切后台三分钟后，再次切回前台，定义为热启动。
Android 计算方式：activityOnResume_end - activityOnRestart_start。

iOS 计算方式：ApplicationDidBecomeActive - ApplicationWillEnterForeground。

启动耗时：

总启动耗时/总启动次数

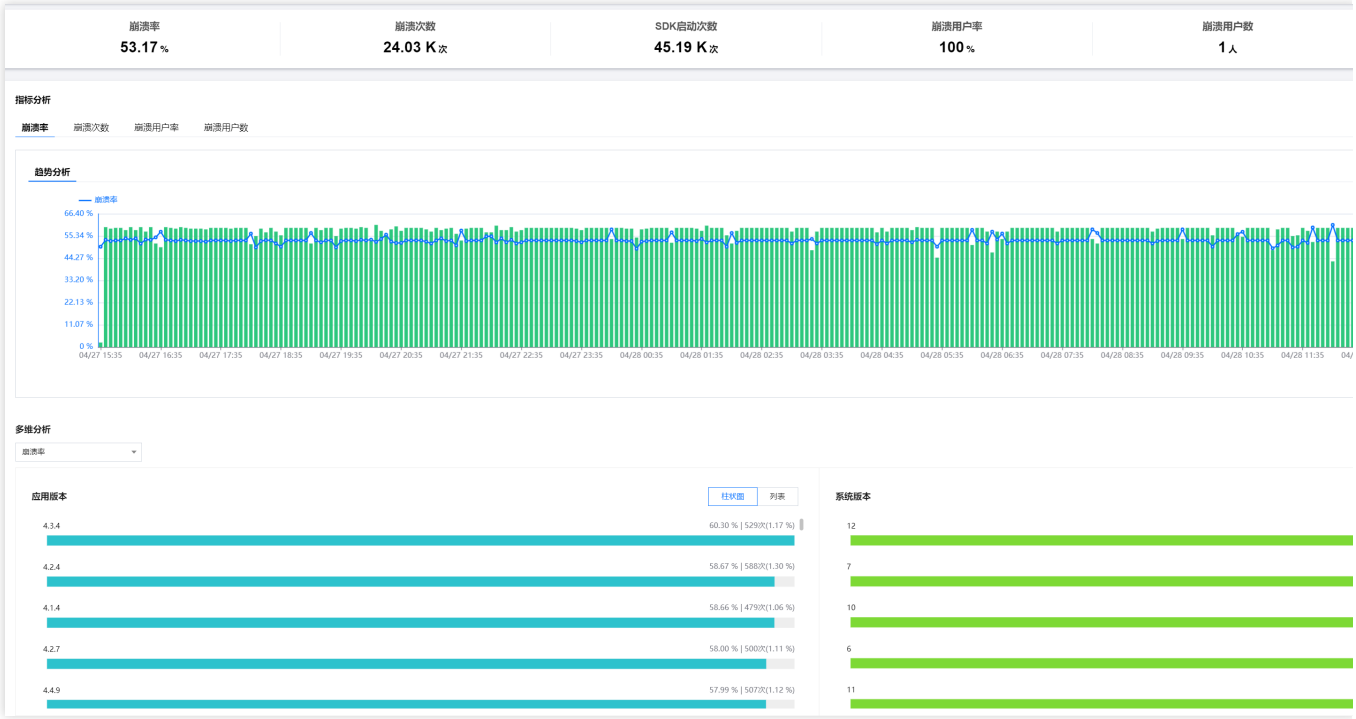
控制台操作指南
崩溃

最近更新时间：2024-05-13 18:03:25

终端性能监控通过对崩溃问题个例提取关键特征进行聚合，便于您针对 App 崩溃的根因分析。

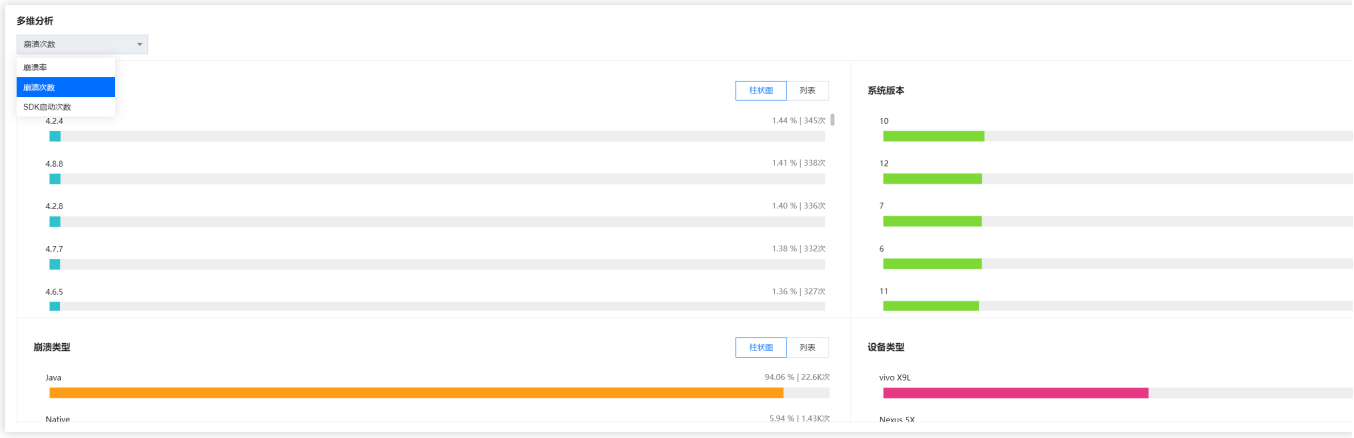
功能入口

1. 登录 腾讯云可观测平台。
2. 在左侧导航栏中选择**终端性能监控 > 崩溃**，选择需要查看的业务系统、应用、时间范围分析崩溃问题。



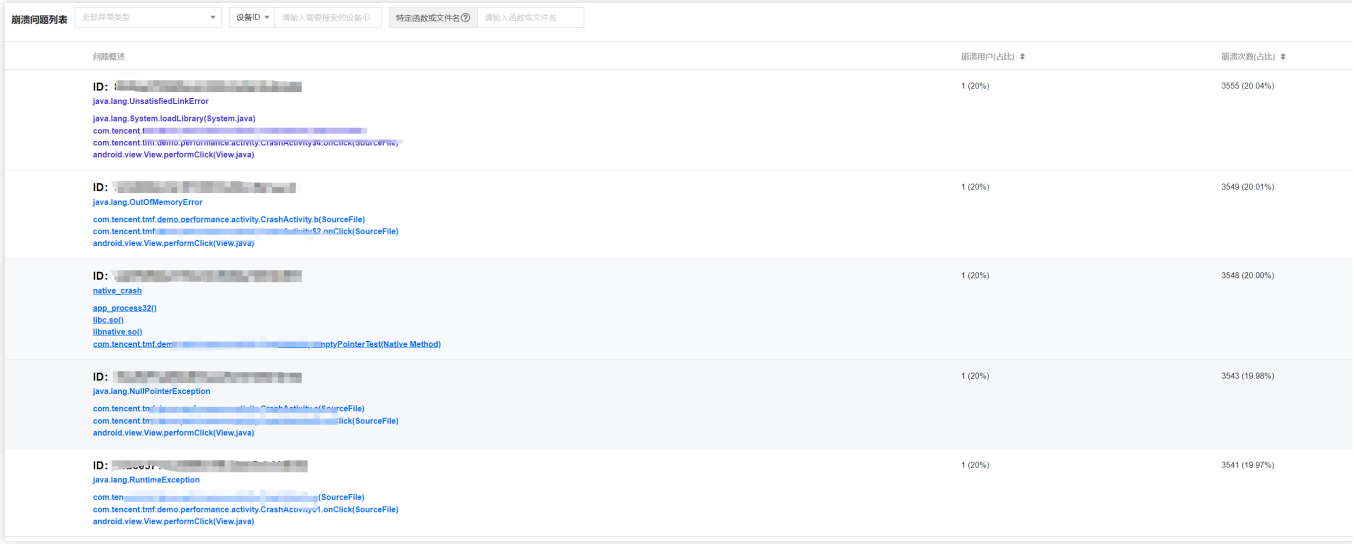
多维分析

多维分析基于应用版本、系统版本、崩溃类型、设备类型、应用状态等多个维度分析关键指标，便于您聚焦的现象进行针对性的崩溃根因分析。



崩溃问题列表

崩溃问题列表展示了所有设备的崩溃问题。您可以根据问题异常类型、问题设备 ID、特定函数或文件名快速筛选相关崩溃问题。您还可以单击**问题概述**查看崩溃问题详情，定位分析崩溃根因。



指标说明

相关指标说明如下表所示：

指标名称	指标说明
崩溃率	指定时间范围内崩溃发生次数/App 启动次数

崩溃用户率	指定时间范围内受到崩溃影响的用户数/启动 App 的用户数
崩溃次数	指定时间范围内崩溃发生次数
崩溃用户数	指定时间范围内受到崩溃影响的用户数
崩溃类型	按照崩溃问题发生位置将崩溃类型分类为 Java 崩溃与 Native 崩溃
SDK 启动次数	应用启动次数

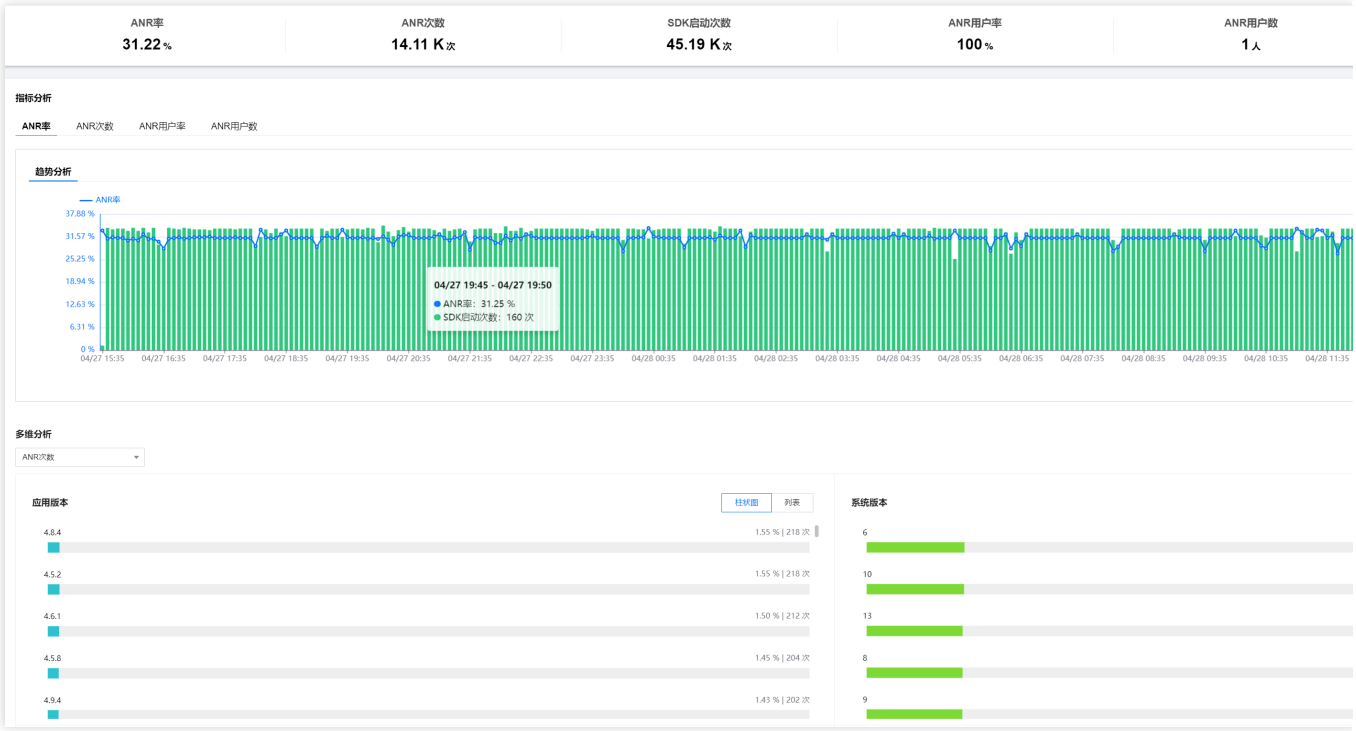
ANR

最近更新时间：2024-05-13 18:03:25

终端性能监控通过对 ANR 问题个例提取关键特征进行聚合，便于您针对 App 的 ANR 问题进行根因分析。

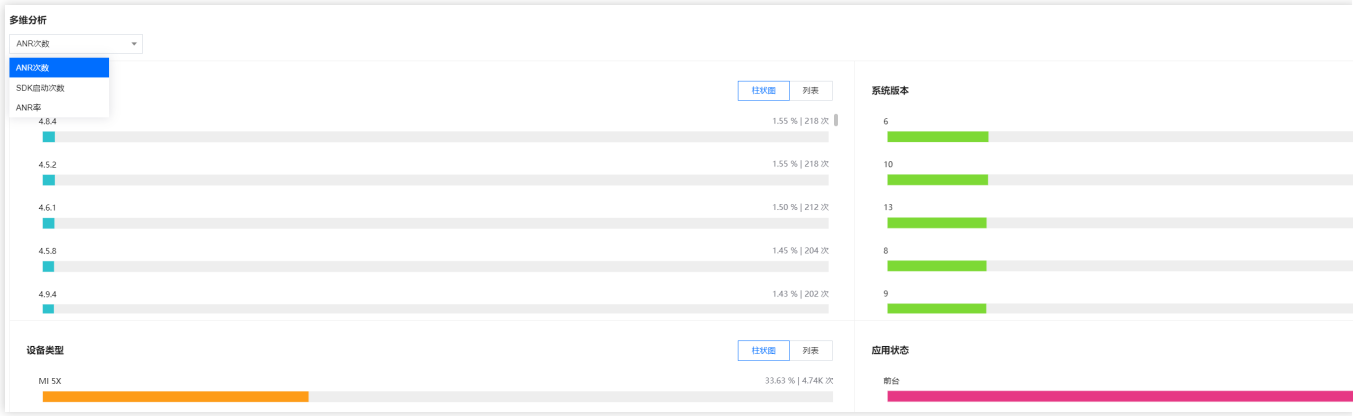
功能入口

1. 登录 腾讯云可观测平台。
2. 在左侧菜单栏中单击**终端性能监控** > **ANR**。
3. 选择需要查看的业务系统、应用、时间范围等分析 ANR 问题。



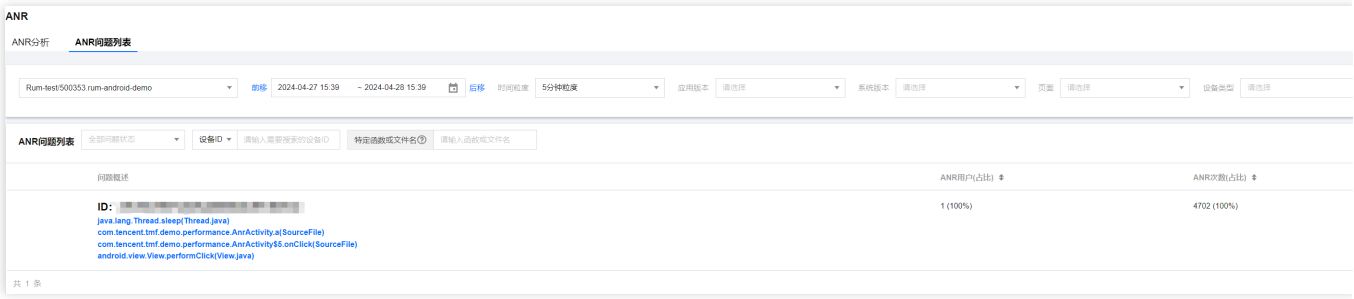
多维分析

多维分析基于 ANR 次数、启动次数、 ANR 率多个维度分析关键指标，便于您聚焦现象进行针对性的 ANR 情况根因分析。



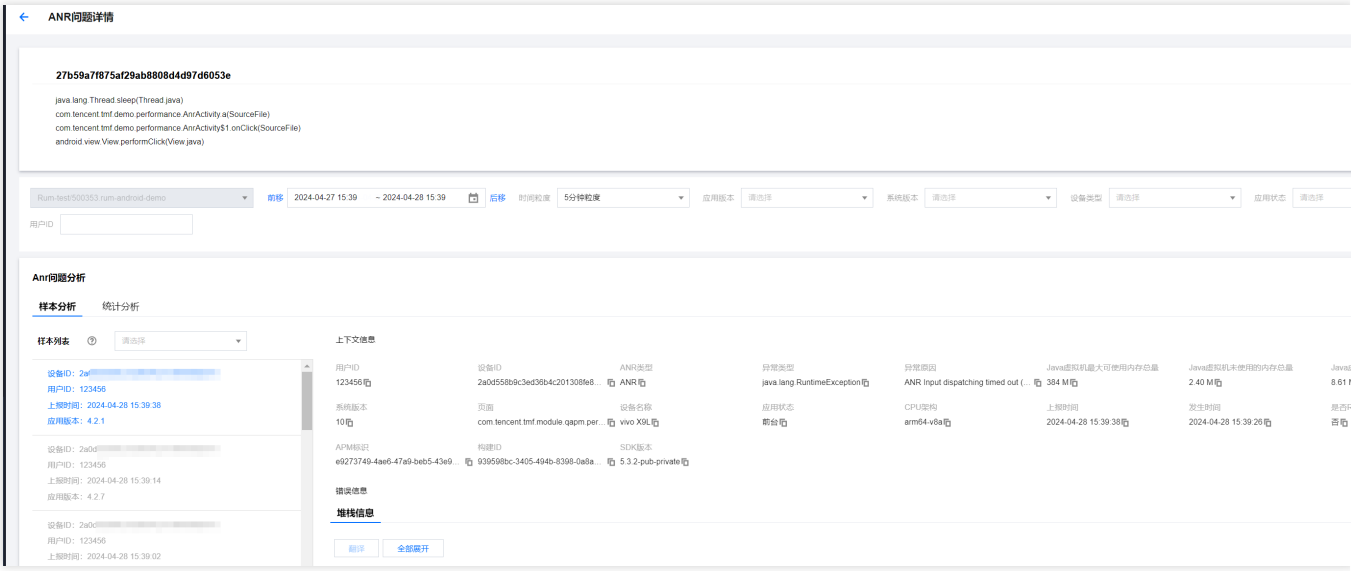
ANR 问题列表

ANR 问题列表页提供针对聚类问题的展示、搜索、排序、管理功能，并可通过单击查看详情按钮下钻到问题详情页。



ANR 问题详情

问题详情页提供针对某一类问题的多维统计分析及针对各问题个例的分析功能，您可以单击对应的问题描述进入问题详情。



指标说明

ANR 相关指标说明如下表所示：

指标名称	指标说明
ANR 率	指定时间范围内应用发生 ANR 设备数 / 总设备数
ANR 次数	指定时间范围内应用发生 ANR 的次数
SDK 启动次数	应用启动次数
ANR 用户率	指定时间范围内受到 ANR 影响的用户数 / 启动应用总用户数
ANR 用户数	指定时间范围内受到 ANR 影响的用户数
用户数	启动应用总用户数

网络

最近更新时间：2024-05-13 18:03:25

通过吞吐量、请求次数、网络响应时间、慢请求占比、HTTP 错误率、网络错误率、TCP 建连时间等指标进行网络问题分析。

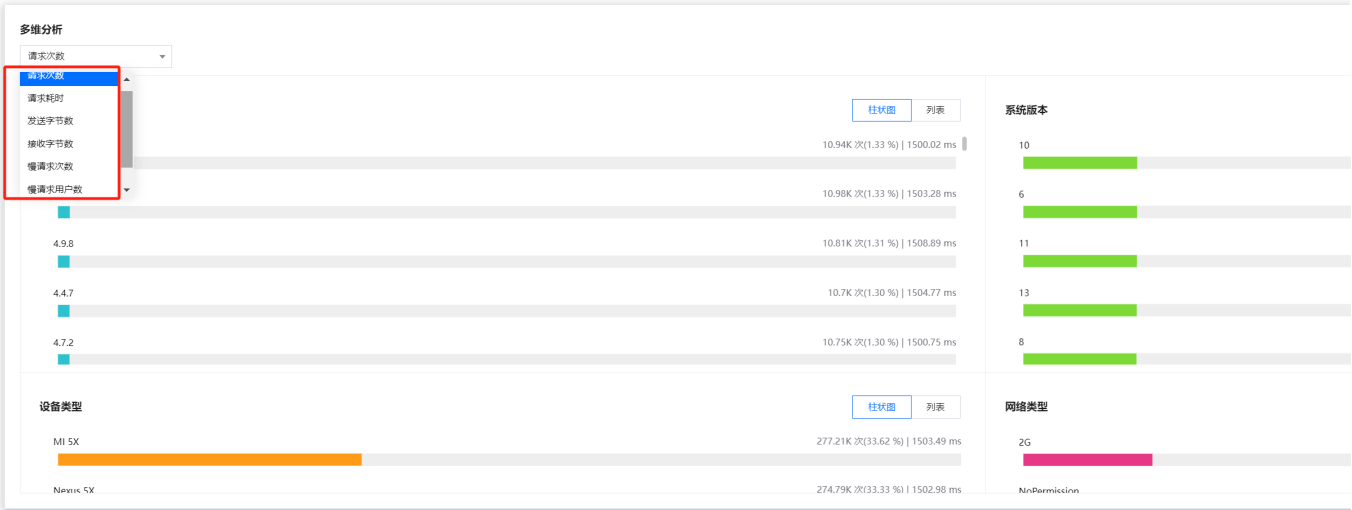
功能入口

1. 登录 腾讯云可观测平台。
2. 在左侧导航栏中选择**终端性能监控 > 网络**，支持从业务系统、应用、时间范围等多个维度分析网络问题。

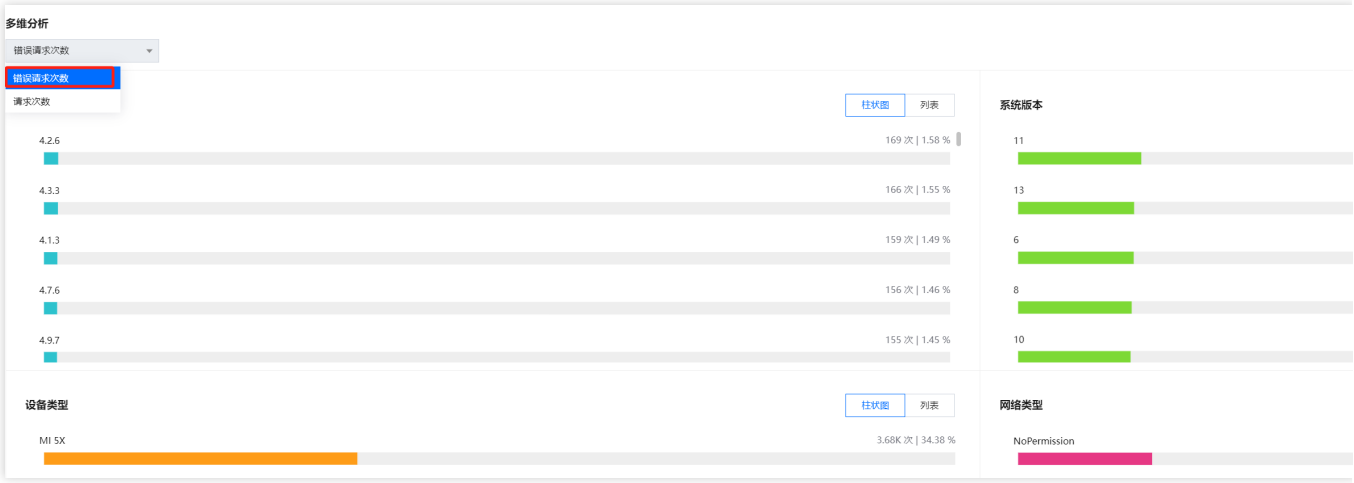
多维分析

多维分析基于应用版本、系统版本、域名、URL、设备类型、网络类型、地区和运营商等多个维度分析关键指标，便于您聚焦的现象进行针对性的慢请求/错误请求根因分析。

慢请求

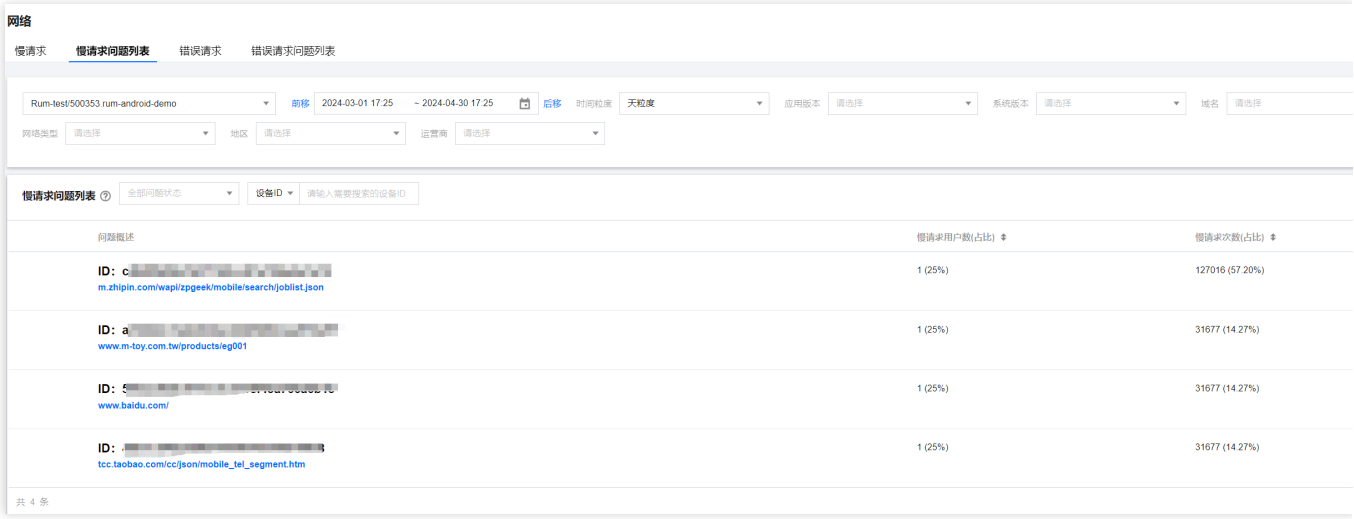


错误请求

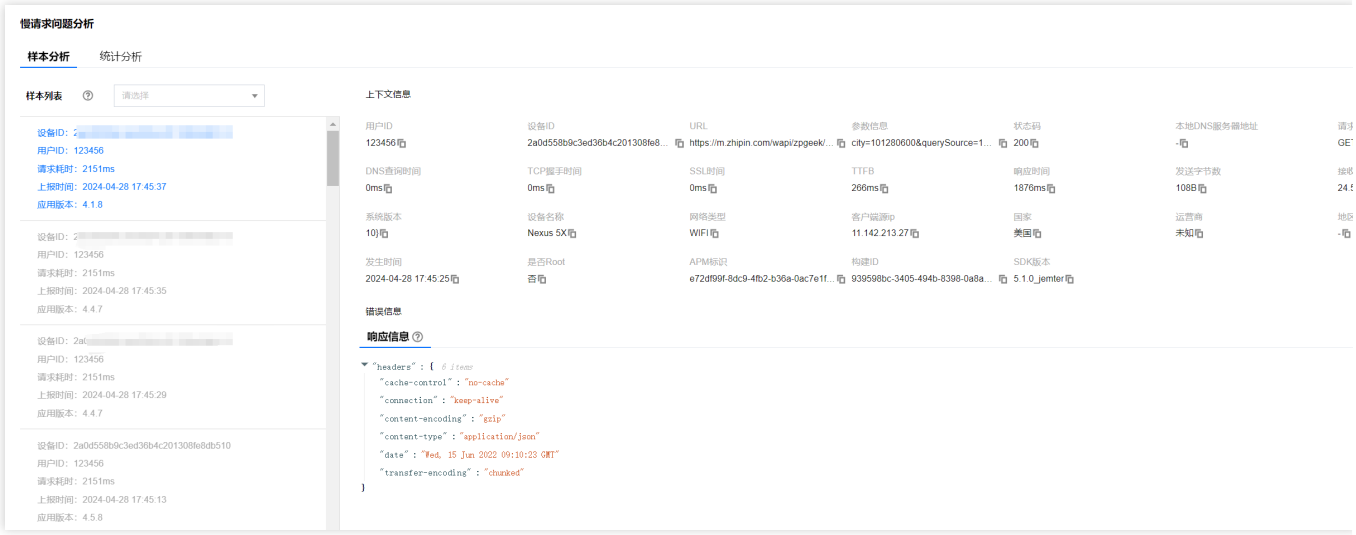


慢请求问题列表

慢请求问题列表展示了所有设备的慢请求问题。您可以根据问题异常类型、问题设备 ID、特定函数或文件名快速筛选相关慢加载设备。您还可以单击**问题概述**下的相关链接查看慢请求问题详情，定位分析 App 慢请求根因。

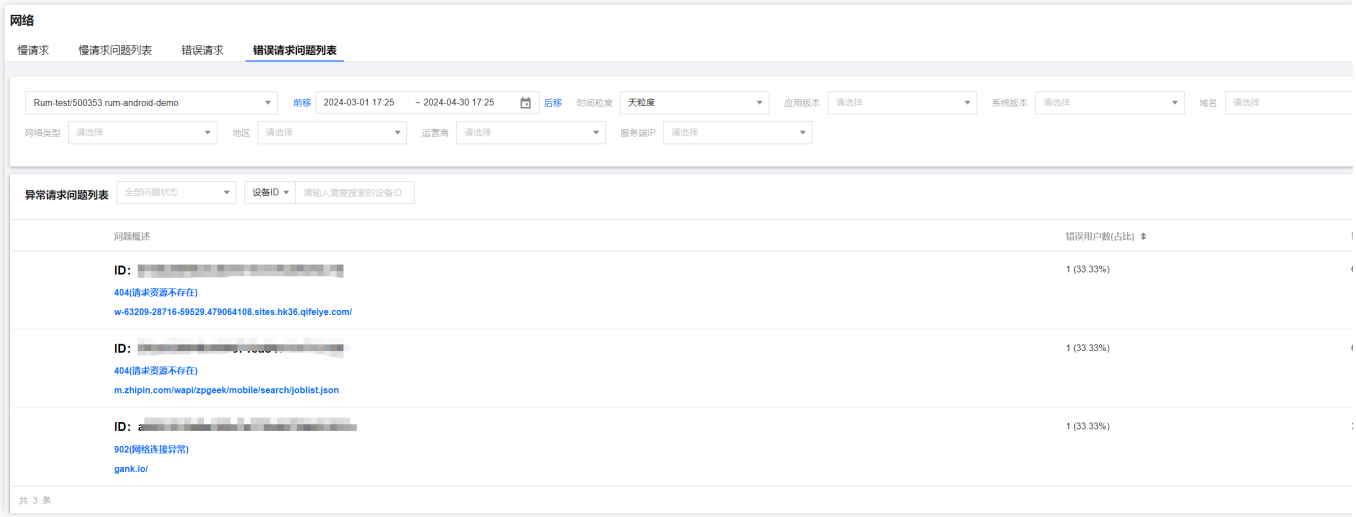


对于每一个 HTTP 请求样本，传输数据大于 50KB 时，传输速度小于 10KB/s 为慢请求；若传输数据小于等于 50KB，响应时间大于 2s 为慢请求，这些慢请求样本会显示在问题列表中。



错误请求问题列表

发生 HTTP 请求错误、 DNS 解析错误、无法建连、连接超时等网络方面的错误。将会展示在错误请求列表中，您可以单击**问题概述**下的相关链接，查看错误请求问题详情，定位分析错误请求根因。



您还可以单击**问题概述**查看错误请求问题详情，定位分析错误原因。

指标说明

相关指标说明如下表所示：

指标名称	指标说明
请求耗时	应用请求耗时
慢请求比例	慢请求占比为选定时间段内，慢请求数量与访问量的比值 传输数据大于50KB时，传输速度小于10KB/s为慢请求 若传输数据小于等于50KB，响应时间大于2s为慢请求
慢请求次数	慢请求次数为选定时间段内的慢请求数量 传输数据大于50KB时，传输速度小于10KB/s为慢请求 若传输数据小于等于50KB，响应时间大于2s为慢请求
请求次数	应用请求总次数
慢请求用户比例	指定时间范围内受到慢请求影响的用户数与总用户的比值
慢请求用户数	指定时间范围内受到慢请求影响的用户数
请求错误率	请求错误次数/总请求次数
错误请求次数	在选定时间段内，出现网络错误的数量 错误请求指发生 HTTP 请求错误、DNS 解析错误、无法建连、连接超时等网络方面的错误
错误用户比例	指定时间范围内受到错误请求影响的用户数与总用户的比值
错误用户数	指定时间范围内受到错误请求影响的用户数

Webview

最近更新时间：2024-05-13 18:03:25

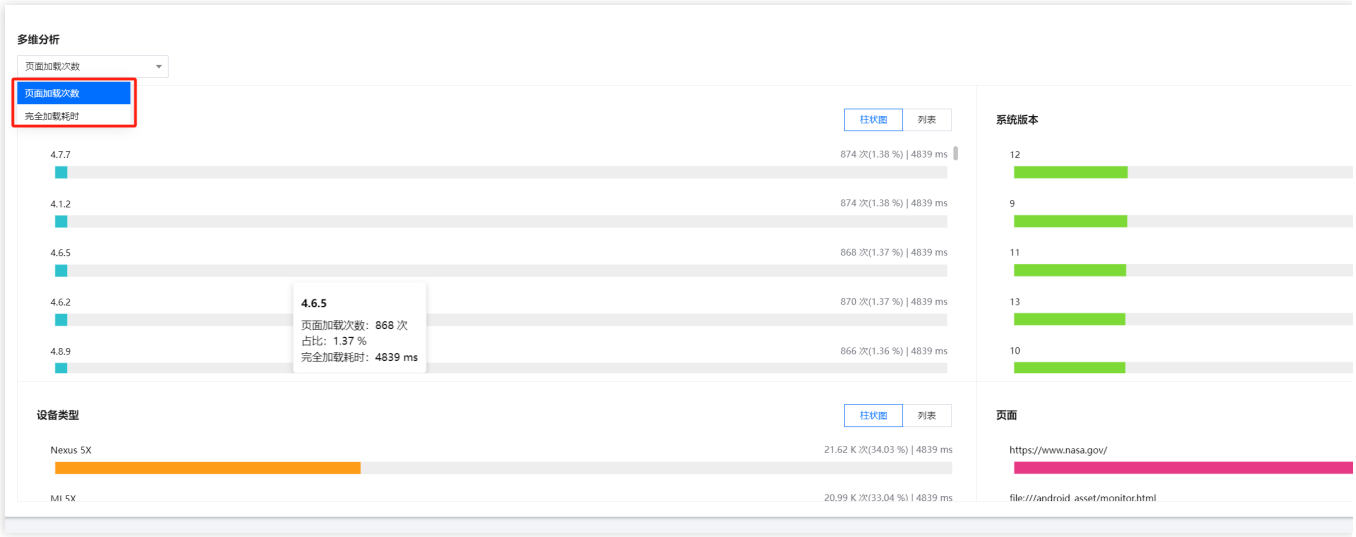
通过页面加载耗时、慢加载占比以及 JS 错误率等进行 WebView 指标分析，并通过问题列表对 WebView 以及 JSError 问题进行下钻。

功能入口

1. 登录 终端性能监控控制台。
2. 在左侧导航栏中选择 **webview**，选择需要查看的业务系统、应用、时间范围分析 Webview 问题。

慢加载、JS 错误多维分析

多维分析基于应用版本、系统版本、设备类型、页面、网络类型、运营商和地区等多个维度分析关键指标，便于您聚焦的现象进行针对性的慢加载/ js 错误根因分析。



慢加载问题列表

慢加载问题列表展示了所有设备的慢加载问题。您可以根据问题异常类型、问题设备 ID 快速筛选相关慢加载设备。您还可以单击**问题概述**查看慢加载问题详情，定位分析 App 慢加载根因。

说明：

慢加载样本上报默认采样率为0.1%，因此问题列表中问题样本数量与指标统计不吻合为正常现象。

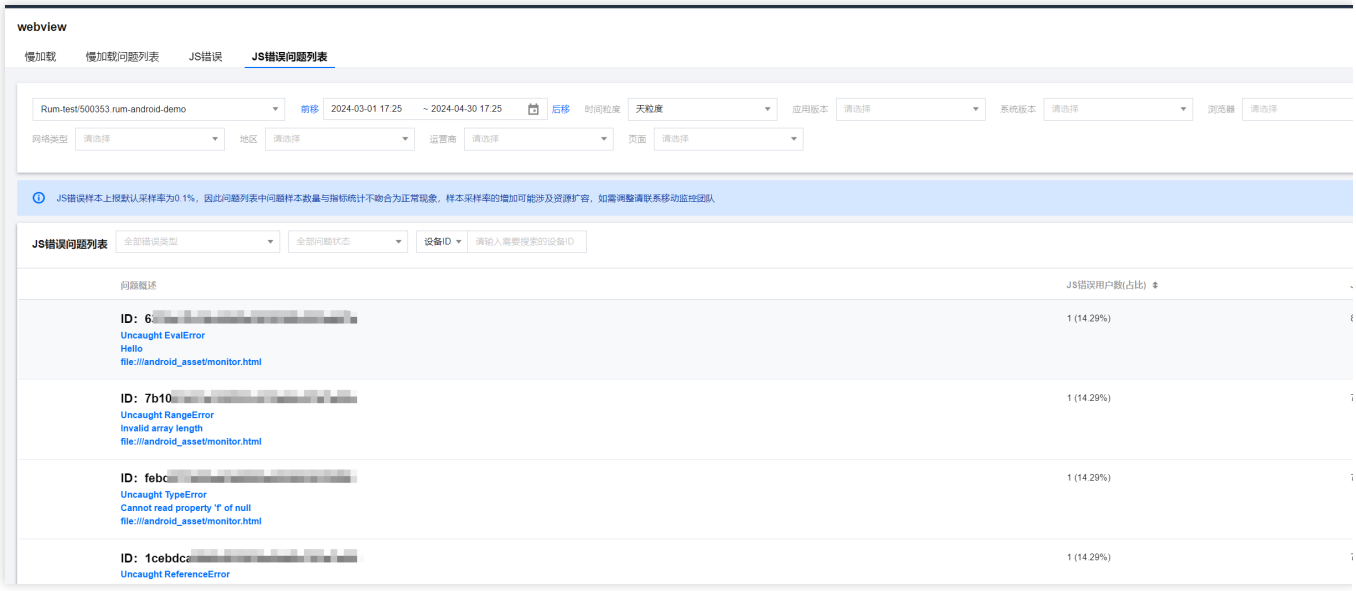
对于每一个页面加载样本，页面完全加载时间大于3500ms 为慢加载，这些慢加载样本会显示在问题列表中。

JS 错误问题列表

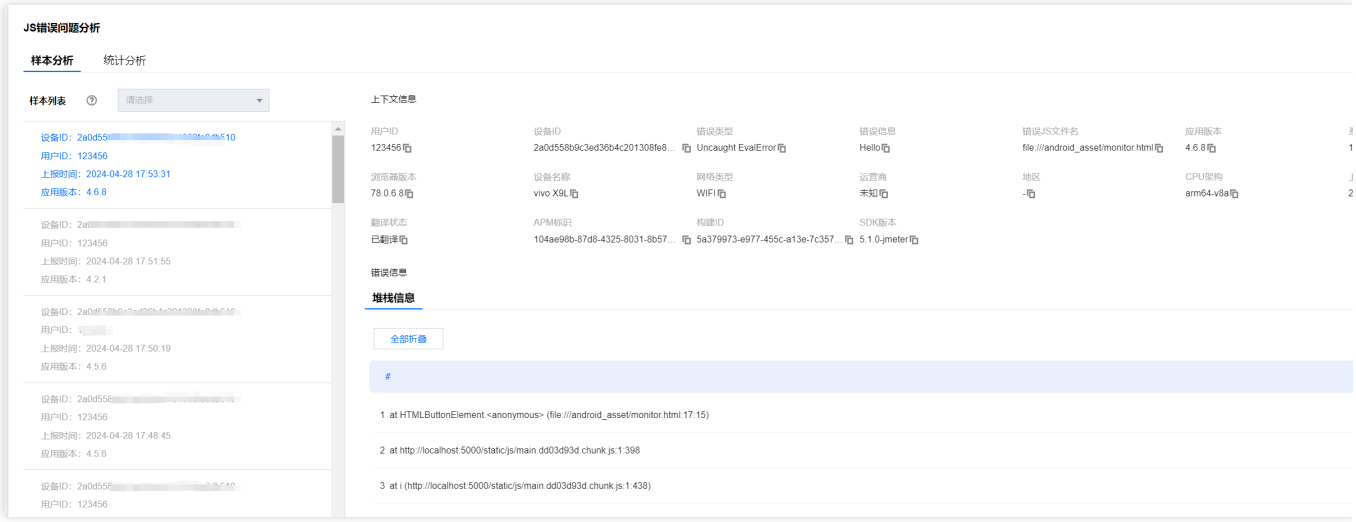
您可以在 JS 错误问题列表查看所有 JS 报错。

说明：

JS 错误问题上报默认采样率为0.1%，因此问题列表中问题样本数量与指标统计不吻合为正常现象。



您还可以单击**问题概述**查看 JS 错误问题详情，定位分析 JS 错误原因。



指标说明

相关指标说明如下表所示：

指标名称	指标说明
页面加载次数	打开或刷新页面的次数
完全加载耗时	指整个网页完全加载的时间
JS 错误次数	指定时间内的 JS 错误总数

JS 错误率	发生 JS 错误的用户数/访问 webview 页面的总用户数，由于计算资源限制，该指标分子及分母为未去重数据
JS 错误用户比例	指定时间范围内受到 JS 错误影响的用户数与总用户的比值
JS 错误用户数	指定时间范围内受到 JS 错误影响的用户数

应用管理

最近更新时间：2024-05-13 18:03:25

操作场景

在应用管理页您可以查看您已接入的终端应用信息，以及新的接入应用、应用 ID 、设置白名单信息。

操作步骤

接入应用

1. 登录 腾讯云可观测平台。
2. 在左侧导航栏中选择**终端性能监控 > 总览**。
3. 单击**应用接入**，填写应用名称、选择 Android 或 IOS 应用类型和所属业务系统，然后单击**下一步**。

获取应用 ID

您可以在**应用管理 >应用设置**页面，您可以切换成菜单为应用设置，在应用设置列表中获取应用 ID 。

白名单配置

您可以在**应用管理 > 白名单管理**页面，点击白名单配置下方的**添加**，通过配置用户 ID/设备 ID 白名单可以避免数据上报受采样影响。即加入白名单的用户/设备不受采样率影响，将会全部采样。您可以选择用户或者设备类型，并填写相关 ID 即可。