Android 统计SDK开发者使用指南

腾讯移动分析出品

目录

[Android 统计SDK开发者使用指南 1](#_Toc423682788)

[1 开始嵌入SDK 4](#_Toc423682789)

[1.1 安装和部署 4](#_Toc423682790)

[1.2 升级SDK 7](#_Toc423682791)

[1.3 代码混淆 7](#_Toc423682792)

[2 初始化并启动MTA 8](#_Toc423682793)

[3 基础指标统计 10](#_Toc423682794)

[3.1 页面统计 10](#_Toc423682795)

[3.2 会话统计 11](#_Toc423682796)

[3.3 错误统计 12](#_Toc423682797)

[3.3.1 主动上报app捕获的错误或异常 12](#_Toc423682798)

[3.3.2 自动捕获并上报app未捕获的异常 12](#_Toc423682799)

[3.3.3 Native crash捕获与上报 13](#_Toc423682800)

[4 自定义事件 15](#_Toc423682801)

[4.1 注册自定义事件 15](#_Toc423682802)

[4.2 【次数统计】Key-Value参数的事件 15](#_Toc423682803)

[4.3 【次数统计】带任意参数的事件 16](#_Toc423682804)

[4.4 【时长统计】Key-Value参数事件 16](#_Toc423682805)

[4.5 【时长统计】带有统计时长的自定义参数事件 17](#_Toc423682806)

[5 接口监控 19](#_Toc423682807)

[6 网速监控 21](#_Toc423682808)

[6.1 本地指定域名端口测速 21](#_Toc423682809)

[6.2 前台指定域名测速 21](#_Toc423682810)

[7 用户身份唯一标识（MID） 21](#_Toc423682811)

[8 云标签 22](#_Toc423682812)

[8.1 活动页面曝光事件 22](#_Toc423682813)

[8.2 活动页面按钮点击事件 22](#_Toc423682814)

[8.3 用户付费事件 23](#_Toc423682815)

[9 高级功能 23](#_Toc423682816)

[9.1 用户画像 23](#_Toc423682817)

[9.2 游戏统计 23](#_Toc423682818)

[9.3 在线配置更新 24](#_Toc423682819)

[10 数据上报 26](#_Toc423682820)

[10.1 数据上报策略 26](#_Toc423682821)

[10.2 数据上报相关的设置接口 27](#_Toc423682822)

[11 APP设置接口 28](#_Toc423682823)

[12 注意事项 30](#_Toc423682824)

[12.1 何时调用StatConfig配置接口 30](#_Toc423682825)

[12.2 调试app 30](#_Toc423682826)

[12.3 发布app 30](#_Toc423682827)

[12.4 SDK冲突问题 31](#_Toc423682828)

[12.5 提示兼容性错误 33](#_Toc423682829)

[12.6 多进程支持 33](#_Toc423682830)

[13 特殊需求 35](#_Toc423682831)

[13.1 只做少数页面统计 35](#_Toc423682832)

[13.2 只统计用户打开app的次数 35](#_Toc423682833)

[13.3 只统计app的设备、网络等信息 35](#_Toc423682834)

[13.4 只上报app未捕获的异常 35](#_Toc423682835)

[13.5 只用于接口监控 35](#_Toc423682836)

[13.6 只做用户画像分析 35](#_Toc423682837)

## 开始嵌入SDK

### 安装和部署

欢迎使用腾讯移动分析（简称MTA）Android统计SDK，您可以按照下面6步开始SDK的统计。

#### Step 1 获取AppKey

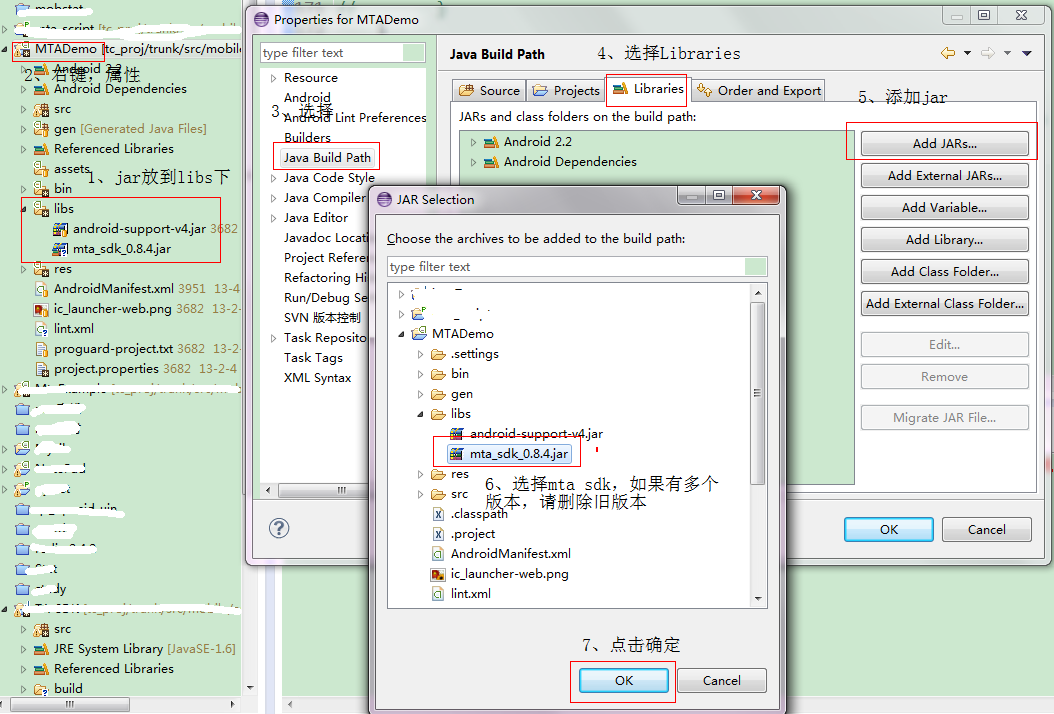
登陆腾讯移动分析移动统计前台，按照步骤提示注册应用，可获得AppKey。

腾讯内部用户：<http://mta.oa.com>

腾讯外部用户：<http://mta.qq.com>

#### Step 2 向工程中导入SDK

下载SDK压缩包，解压至本地目录，将其中lib目录下的**mta-sdk-x.x.x.jar**[[1]](#footnote-1)**复制到您的应用工程libs目录**（若不存在请新建一个）中。以Eclipse为例：右键点击工程根目录→选择Properties → Java Build Path →Libraries →点击Add JARs…选中当前工程libs目彔下的**mta-sdk-x.x.x.jar**文件，点击“OK”按钮即导入成功。



#### Step 3 配置AndroidManifest.xml文件

需要添加下面两类配置：

|  |  |  |  |
| --- | --- | --- | --- |
| **Meta-Data** | **类型** | **用途** | **必选** |
| *TA\_APPKEY* | String，不能为纯数字字符串 | MTA提供给每个app的appkey，用来定位该应用程序的唯一性 | **√** |
| *InstallChannel* | String，若为纯数字字符串不能超过int表示的范围 | 用来标注应用推广渠道，区分新用户的来源来查看统计 | **√** |

（注意：appkey和installChannel也能够在代码中设置，见[APP设置接口](#_APP设置接口)）

|  |  |  |
| --- | --- | --- |
| **需要的权限** | **用途** | **必选** |
| *INTERNET* | 允许应用程序联网，以便向我们的服务器端发送数据 | **√** |
| *READ\_PHONE\_STATE* | 获取用户手机的IMEI，用来唯一的标识用户。(运行在平板上的应用会读取mac地址作为用户的唯一标识 | **√** |
| *ACCESS\_NETWORK\_STATE* | 获取设备的网络状态 | **√** |
| *ACCESS\_WIFI\_STATE* | 获取设备的WIFI网络状态 | **√** |
| *WRITE\_EXTERNAL\_STORAGE* | 获取SD卡信息 | **√** |

**示例文件**：

#### Step 4 在代码中添加SDK的引用

**<?xml version="1.0" encoding="utf-8"?>**

**<manifest ......>**

***<uses-sdk android:minSdkVersion="7" android:targetSdkVersion="7"/>***

**<!—为MTA授权。< -->**

**<!—如果是第三方lib项目，请在手册中提示app开发者授予以下权限！ < -->**

***<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.READ\_PHONE\_STATE"/>***

***<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"/>***

***<uses-permission android:name="android.permission.WRITE\_SETTINGS"/>***

**<!-- 请在application处配置appkey和渠道或在代码处调用StatConfig类接口 < -->**

***<application ......>***

***<activity ......>***

***......***

***</activity>***

**<!-- 请将value改为MTA分配的appkey < -->**

***<meta-data android:name="TA\_APPKEY" android:value="ABCDEFG12233456"/>***

**<!-- 请将value改为app发布对应的渠道，不同的发布渠道使用不同的名字 < -->**

***<meta-data android:name="InstallChannel" android:value="play"/>***

**<!-- 注意：若填写的渠道为纯数字字符串类型，请不要超过int表示的范围！ < -->**

**<!-- 注意：若填写的渠道为纯数字字符串类型，请不要超过int表示的范围！ < -->**

**<!-- mid-sdk-3.5以上版本增加 < -->**

**<provider**

**android:name="com.tencent.mid.api.MidProvider"**

**android:authorities="你的包名.TENCENT.MID.V3"**

**android:exported="true" >**

**</provider>**

***</application>***

**</manifest>**

import com.tencent.stat.StatConfig

import com.tencent.stat.StatService

**StatConfig类**：MTA配置类，可以设置上报策略、Debug开关、session超时时间等，**需要在初始化MTA之前被调用才能及时生效**，通常使用SDK默认配置即可。

**StatService类**：MTA统计类，需要开发者按下面的步骤主动调用接口。

#### Step5 添加SDK的统计

在代码处调用类StatService提供的接口（见章节2、3、4、5、6），开始嵌入MTA的统计功能。

#### Step 6 验证数据上报是否正常

当您完成以下的MTA嵌入工作后，启动app，触发MTA统计接口，经过5秒左右，正常情况下，在您的app首页就能看到实时指标在更新，说明您已成功嵌入MTA，可继续深入的统计开发。

如果经过几分钟后，尚未看到实时指标更新，请检查以下事项：

1. 设备的wifi是否打开，是否正常联网；
2. APPKEY、权限等设置是否正确；
3. 确保已触发MTA统计接口；
4. 打开MTA的debug开关，查看标签为“MtaSDK“的logcat提示，是否有错误日志。
5. 如果logcat提示“Compatibility problem was found in this device!“，请先删除apk重新安装，可参考[兼容性错误](#_提示兼容性错误)。

### 升级SDK

新版本SDK兼容老版本接口，升级时只需要替换旧的jar包即可：先在工程所在libs目录下删除旧的jar包，复制新jar包到libs路径，同时，在Java Build Path里面删除旧的jar包，并添加新jar包引用。

### 代码混淆

请保留以下选项：

-keep class com.tencent.stat.\*\* {\* ;}

-keep class com.tencent.mid.\*\* {\* ;}

## 初始化并启动MTA

在所有其它StatService方法被调用之前调用以下接口初始化MTA。**第三方合作SDK lib必须初始化MTA，其它非lib类的项目可自行决定是否初始化，不影响正常使用。初始化MTA并不会上报任何数据，仅仅是激活MTA，并预加载数据库的配置信息。**

***boolean StatService.startStatService(Context ctx, String appkey,***

***String mtaSdkVersion)***

**参数**：Ctx Application页面的设备上下文

Appkey MTA提供的appkey，若为null，则按读取StatConfig.setAppKey()或manifest.xml配置的appkey

requiredMtaVer 当前app依赖的MTA SDK版本号，只能为**com.tencent.stat.common.StatConstants.VERSION**，用于SDK版本冲突检测

**MtaSDkException异常**：启动失败时会抛出MtaSDkException异常，可能是参数出错，也可能是SDK版本冲突，具体的冲突解决办法见注意事项中的“[SDK冲突问题](#_多版本sdk冲突问题)”。同时，MTA会自动禁止所有功能。

**调用位置**：

1、对于普通app：AndroidManifest.xml指定首先启动的activity的onCreate()处，StatConfig类的方法之后。

2、对于lib工程，在其它所有StatService方法被调用之前，StatConfig类的方法之后。

（注意：StatConfig配置类需要在此方法前才能及时生效）

@Override

**protected void** onCreate(**Bundle** savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_mtamain);

// androidManifest.xml指定本activity最先启动

// 因此，MTA的初始化工作需要在本onCreate中进行

// 在startStatService之前调用StatConfig配置类接口，使得MTA配置及时生效

// 2.3.0版本新增，参数为Application

StatService.setContext(this.getApplication());

initMTAConfig(true);

String appkey = "amtaandroid0";

// 初始化并启动MTA

**// 第三方SDK必须按以下代码初始化MTA，**其中appkey为规定的格式或MTA分配的代码。

// 其它普通的app可自行选择是否调用

try {

**// 第三个参数必须为：com.tencent.stat.common.StatConstants.VERSION**

StatService.**startStatService**(this, appkey,

**com.tencent.stat.common.StatConstants.VERSION**);

} catch (MtaSDkException e) {

// MTA初始化失败

logger.error("MTA start failed.");

logger.error("e");

}

}

## 基础指标统计

基础指标包括页面统计，会话统计，错误统计3个部分。

### 页面统计

使用下面的函数统计某个页面的访问情况：

* 标记一次页面访问的开始

***void StatService.onResume(Context ctx)或***

***void StatService.trackBeginPage(Context ctx, String pageName)***

**参数**：Ctx 页面的设备上下文

pageName自定义页面名称，需跟trackEndPage一一匹配使用

**调用位置**：每个activity的onResume()

@Override

**protected void** onResume() {

**super**.onResume();

StatService.**onResume**(this);

}

（注意：每次调用，MTA会检查是否产生新会话（session超时），即生成启动次数。）

* 标记一次页面访问的结束

***void StatService.onPause (Context ctx)或***

***void StatService.trackEndPage(Context ctx, String pageName)***

**参数**：Ctx 页面的设备上下文

pageName自定义页面名称，需跟trackBeginPage一一匹配使用

**调用位置**：每个activity的onPause()

@Override

**protected void** onPause() {

**super**.onPause();

StatService.**onPause**(this);

}

（注意：onResume和onPause或trackBeginPage和trackEndPage需要成对使用才能正常统计activity，为了统计准确性，建议在每个activity中都调用以上接口，否则可能会导致MTA上报过多的启动次数，解决办法参考“[特殊需求](#_特殊需求)”）

* 通过继承的方式统计页面访问

开发者可以通过app本身的acivity基类，调用MTA的onResume和onPause，并在所有子类中，重载这2个方法，实现页面统计功能。可参考MtaDemo中的BaseActivity和DrivedActivity代码。

另外，MTA SDK中的下面两个类实现了StatService.onResume()和StatService.onPause()的调用，可直接继承以下类并在子类中重载页面统计接口。

***com.tencent.stat.EasyActivity*** 继承自android.app.Activity

***com.tencent.stat.EasyListActivity*** 继承自android.app.ListActivity

### 会话统计

会话统计用于统计启动次数，由SDK本身维护，通常开发者无需额外设置或调用接口。

以下3种情况下，会视为用户打开一次新的会话：

1. 应用第一次启动，或者应用进程在后台被杀掉之后启动
2. 应用退到后台或锁屏超过X之后再次回到前台

X秒通过StatConfig.setSessionTimoutMillis(int)函数设置，默认为30000ms，即30秒

举例说明：用户打开手机QQ连续操作10分钟，之后按home键（或锁屏）退到后台，超过30秒后再次回到QQ，此时，SDK会上报一次会话。

注意：请根据您的app业务情况决定是否调整超时时间。

1. 调用SDK提供的startNewSession()函数

***void StatService.startNewSession(Context ctx)***

**参数**：Ctx 页面的设备上下文

### 错误统计

收集应用程序的异常信息可以帮助您完善自己的程序。此外，SDK可以帮助开发者检查app未捕获的异常。

### 主动上报app捕获的错误或异常

***void StatService.reportError(Context ctx, String errormsg)***

**参数**：Ctx 页面的设备上下文

errmsg 出错信息字符串

**调用位置**：需要上报错误信息的地方

***void StatService.reportException(Context ctx, Throwable err)***

**参数**：Ctx 页面的设备上下文

err 抛出的异常

**调用位置**：需要上报异常信息的地方

### 自动捕获并上报app未捕获的异常

**try**{

**int** retval = totalNumber / personInRoom;

}

**catch** (ArithmeticException e){

StatService.**reportException**(**this**, e);

}

SDK将异常处理类设置到线程上，作为所有线程出现未捕获异常时的缺省行为。因此，只要集成了SDK，app中未捕获到的异常也能上报。

（注意：如果app已设置Thread.setDefaultUncaughtExceptionHandler()，那么MTA会在处理完异常后，再调用原来的异常处理函数继续处理。）

可以通过下面的接口关闭自动捕获异常功能（默认为true）

***void StatConfig.setAutoExceptionCaught(boolean isAutoExceptionCaught)***

注意：必须在所有StatService方法调用之前设置此接口。

@Override

**protected void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

// app的最初始代码处

StatConfig.**setAutoExceptionCaught(false);**

}

通过LogCat检查有以下警告，表明已经成功**禁用**了本功能。



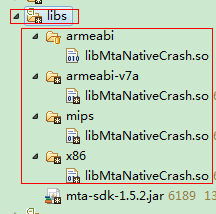
### Native crash捕获与上报

1. **配置libMtaNativeCrash.so**

若要捕获Native crash，需要把libMtaNativeCrash.so加到工程中，分2种情况。

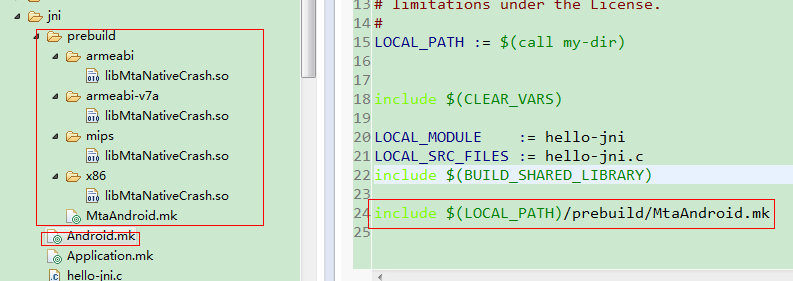
a） 纯Java的Android项目

工程不涉及到C/C++开发，可直接复制解压缩后的MTA包，把lib目录下的文件全部复制到工程目录下的libs目录下，见下图。



1. 带C/C++的Android项目

eclipse在构建C/C++代码时可能会自动删除libMtaNativeCrash.so，因此需要确保libMtaNativeCrash.so在发布时已存在。可网上搜索相关的解决方案或参考MtaJNIDemo工程： a）在jni目录下新建prebuild目录，将armeabi、armeabi-v7a、mips、x86目录和MtaAndroid.mk文件复制到prebuild目录下；b）在jni目录下的Android.mk末尾添加“include $(LOCAL\_PATH)/prebuild/MtaAndroid.mk”即可。



1. **调用初始化NativeCrash接口**

在初始化时调用以下接口，SDK将会捕获Android Native层的异常，采集异常堆栈并在下次启动时上报。

***void StatConfig.initNativeCrashReport(Context ctx, String tombstones\_dir)***

**参数**：Ctx 页面的设备上下文

tombstones\_dir 通常为null。tombstones文件生成目录，请确保该路径对当前app是可读写的；若为null，则使用默认位置：/data/data/your\_package/app\_tombstones。当NDK异常发生时，MTA首先会保存异常堆栈信息到这个目录下，下次启动时读取本目录文件并上报，最后删除文件。

**调用位置**：初始化时

@Override

**protected void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

// app的最初始代码处

StatConfig.**initNativeCrashReport (this, null);**

}

## 自定义事件

可以统计某些用户自定义事件的发生次数，时间，变化趋势，例如广告点击，短信数量等等，通常event\_id用于表示某种行为或功能的统计（如统计“点击”按钮被触发多少次），而参数则用于标识统计的具体对象（如名称为“OK”的按钮），由“event\_id”和“参数”唯一标识一个事件。

自定义事件分为2大类：

1. 统计次数：统计指定行为被触发的次数
2. 统计时长：统计指定行为消耗的时间，单位为秒。需要begin接口与end接口成对使用才生效。

其中每类事件都有Key-Value参数类型和不定长字符串参数类型，由于Key-Value参数类型的接口能表达更丰富的内容，我们**推荐优先使用Key-Value类参数接口**。另外，如果代码同时使用了这2种参数类型，event\_id最好不一样。

**注意：**event\_id需要先在腾讯移动分析网站上面注册，才能参与正常的数据统计。event\_id不能包含空格或转义字符。

### 注册自定义事件

自定义事件的注册（配置）包括事件id的注册和事件id下参数信息的注册。

1. 登陆mta前台，选择左边选择“自定义事件”。
2. 选择“新增事件”，按照需求填写事件id、key、value等信息。
3. 可以在查看详情下的“参数”查看事件id下所有参数上报的明细。

### 【次数统计】Key-Value参数的事件

***void StatService.trackCustomKVEvent(Context ctx, String event\_id,***

***Properties properties)***

**参数**： Ctx 页面的设备上下文

event\_id 事件标识

properties Key-Value参数对，key和value都是String类型

**调用位置**：代码任意处

### 【次数统计】带任意参数的事件

**public void** onOKBtnClick(View v) {

// 统计按钮被点击次数，统计对象：OK按钮

Properties prop = new Properties();

prop.setProperty("name", " OK ");

StatService.**trackCustomKVEvent**(**this**, " button\_click", prop);

}

**public void** onBackBtnClick(View v) {

// 统计按钮被点击次数，统计对象：back按钮

Properties prop = new Properties();

prop.setProperty("name", " back ");

StatService.**trackCustomKVEvent**(**this**, " button\_click", prop);

}

***void StatService.******trackCustomEvent(Context ctx, String event\_id, String... args)***

**参数**： Ctx 页面的设备上下文

event\_id 事件标识

args 事件参数

**调用位置**：代码任意处

**public void** onClick(View v) {

// 统计按钮被点击次数，统计对象：OK按钮

StatService.**trackCustomEvent**(**this**, "button\_click", "OK ");

}

### 【时长统计】Key-Value参数事件

可以指定事件的开始和结束时间，来上报一个带有统计时长的事件。

***void StatService.trackCustomBeginKVEvent(***

***Context ctx, String event\_id, Properties properties)***

***void StatService.trackCustomEndKVEvent(***

***Context ctx, String event\_id, Properties properties)***

**参数**： Ctx 页面的设备上下文

event\_id 事件标识

properties Key-Value参数对，key和value都是String类型

**public void** onClick(View v) {

Properties prop = new Properties();

prop.setProperty("level", "5");

// 统计用户通关所花时长，关卡等级： 5

// 用户通关前

StatService.**trackCustomBeginKVEvent**(**this**, " playTime", prop);

// 用户正在游戏中….

// …….

// 用户通关完成时

StatService.**trackCustomEndKVEvent**(**this**, " playTime", prop);

}

**调用位置**：代码任意处

注意：trackCustomBeginKVEvent和trackCustomEndKVEvent必须成对出现，且参数列表完全相同，才能正常上报事件。

### 【时长统计】带有统计时长的自定义参数事件

可以指定事件的开始和结束时间，来上报一个带有统计时长的事件。

***void StatService.trackCustomBeginEvent(***

***Context ctx, String event\_id, String... args)***

***void StatService.trackCustomEndEvent(***

***Context ctx, String event\_id, String... args)***

**参数**： Ctx 页面的设备上下文

event\_id 事件标识

args 事件参数

**public void** onClick(View v) {

// 统计用户通关所花时长

// 用户通关前

StatService.**trackCustomBeginEvent**(**this**, " playTime", "level5");

// 用户正在游戏中….

// …….

// 用户通关完成时

StatService.**trackCustomEndEvent**(**this**, " playTime", " level5");

}

**调用位置**：代码任意处

注意：trackCustomBeginEvent和trackCustomEndKvent必须成对出现，且参数列表完全相同，才能正常上报事件。

## 接口监控

统计应用对某个外部接口（特别是网络类的接口，如连接、登陆、下载等）的调用情况。当开发者用到某个外部接口，可调用该函数将一些指标进行上报，MTA将统计出每个接口的调用情况，并在接口可用性发生变化时进行告警通知; 对于调用量很大的接口，也可以采样上报，云监控统计将根据sampling参数在展现页面进行数量的还原。

***void StatService.reportAppMonitorStat (***

***Context ctx, StatAppMonitor monitor)***

**参数**： ctx 页面的设备上下文

monitor 监控对象，需要根据接口情况设置接口名称、耗时、返回值类型、返回码、请求包大小、响应包大小和采样率等信息，详见doc/api目录下的文档

**调用位置**：被监控的接口

StatAppMonitor方法名列表

|  |  |
| --- | --- |
| **接口名** | **说明** |
| setInterfaceName(String interfaceName) | 设置监控的接口名称 |
| setReqSize(long reqSize) | 请求包大小，单位：byte |
| setRespSize(long respSize) | 响应包大小，单位：byte |
| setResultType(int resultType) | SUCCESS\_RESULT\_TYPE；  FAILURE\_RESULT\_TYPE；  LOGIC\_FAILURE\_RESULT\_TYPE |
| setMillisecondsConsume(long millisecondsConsume) | 调用耗时，单位：毫秒（ms） |
| setReturnCode(int returnCode) | 监控接口业务返回码 |
| setSampling(int sampling) | 采样率： 默认为1，表示100%。如果是1/2，则填2，如果是1/4，则填4，若是1/n，则填n |

// 新建监控接口对象

**StatAppMonitor monitor = new StatAppMonitor("ping:www.qq.com");**

String ip = "www.qq.com";

Runtime run = Runtime.getRuntime();

java.lang.Process proc = null;

try {

String str = "ping -c 3 -i 0.2 -W 1 " + ip;

long starttime = System.currentTimeMillis();

// 被监控的接口

proc = run.exec(str);

proc.waitFor();

long difftime = System.currentTimeMillis() - starttime;

// 设置接口耗时

**monitor.setMillisecondsConsume(difftime);**

int retCode = proc.waitFor();

// 设置接口返回码

**monitor.setReturnCode(retCode);**

// 设置请求包大小，若有的话

**monitor.setReqSize(1000);**

// 设置响应包大小，若有的话

**monitor.setRespSize(2000);**

// 设置抽样率

// 默认为1，表示100%。

// 如果是50%，则填2(100/50)，如果是25%，则填4(100/25)，以此类推。

**monitor.setSampling(2);**

if (retCode == 0) {

logger.debug("ping连接成功");

// 标记为成功

**monitor.setResultType(StatAppMonitor.SUCCESS\_RESULT\_TYPE);**

} else {

logger.debug("ping测试失败");

// 标记为逻辑失败，可能由网络未连接等原因引起的

// 但对于业务来说不是致命的，是可容忍的

**monitor.setResultType(StatAppMonitor.LOGIC\_FAILURE\_RESULT\_TYPE);**

}

} catch (Exception e) {

logger.e(e);

// 接口调用出现异常，致命的，标识为失败

**monitor.setResultType(StatAppMonitor.FAILURE\_RESULT\_TYPE);**

} finally {

proc.destroy();

}

// 上报接口监控

StatService.**reportAppMonitorStat**(ctx, monitor);

## 网速监控

网速监控分为2种类型，一种是在APP本地提供域名和端口列表进行测速；另一种是在前台配置域名列表，由SDK或APP在适当的时候进行测速。所有的网速监控都是异步操作。

### 本地指定域名端口测速

在APP本地指定测速的域名和列表，由开发者决定何时进行测速。

***void StatService.testSpeed(Context ctx,***

***Map<String, Integer> domainMap)***

**参数**： ctx 页面的设备上下文

domainMap 待测速的域名和端口列表

Map<String, Integer> map = **new** HashMap<String, Integer>();

map.put("www.qq.com", 80);

map.put("pingma.qq.com", 80);

StatService.*testSpeed*(ctx, map);

### 前台指定域名测速

开发者在前台配置待监控的域名和端口列表，由服务器下发到SDK，然后在app需要测速的地方调用以下接口，便会对配置的所有域名进行测速监控。

通常SDK在app启动时会主动测速，如果要在特定的地方测速，需要开发者主动调用本接口。

***void StatService.testSpeed(Context ctx)***

**参数**： ctx 页面的设备上下文

（注意：主动调用本接口产生网络I/O，可能会影响用户体验，请慎重使用）

## 用户身份唯一标识（MID）

MTA对于每个上报过数据的设备都会生成一个唯一的身份ID：MTA Identity，简称MID，开发者可以通过接口获取由MTA采集的设备信息。

***public static String StatConfig.getMid(Context ctx)***

**参数**： ctx 页面的设备上下文

**返回值**：String 唯一身份标识MID

**调用位置**：任意

**注意事项**：MID是由SDK向服务器请求后得到的，因此，APP首次安装激活时，如果没有上报过数据，获取到的MID可能为默认值"0"。

## 云标签

云标签事件用于mta做数据挖掘并对挖掘后的用户贴标签，若需要使用mta云标签功能的应用需要上报此类事件，自定义事件ID和参数在配置时有特殊要求。

|  |  |  |
| --- | --- | --- |
| **事件ID** | **必填参数** | **说明** |
| mta\_tag\_activity\_open | aty、gid | 活动页面曝光事件 |
| mta\_tag\_activity\_click | aty、gid、btn | 活动页面曝光事件 |
| mta\_tag\_user\_pay | target、amount | 用户付费事件 |

### 活动页面曝光事件

活动页面曝光事件，事件ID为**“mta\_tag\_activity\_open”。**

Properties prop = **new** Properties();

prop.setProperty("aty", "抢票活动"); // 活动页面

prop.setProperty("gid", "1"); // 用户组名称

StatService.*trackCustomKVEvent*(**this**, "mta\_tag\_activity\_open", prop);

### 活动页面按钮点击事件

活动页面曝光事件，事件ID为“mta\_tag\_activity\_click“。

Properties prop = **new** Properties();

prop.setProperty("aty", "抢票活动"); // 活动页面

prop.setProperty("gid", "1"); // 用户组名称

prop.setProperty("btn", "报名"); // 按钮

StatService.*trackCustomKVEvent*(**this**, "mta\_tag\_activity\_click", prop);

### 用户付费事件

用户付费事件，事件ID为“mta\_tag\_user\_pay”。

Properties prop = **new** Properties();

prop.setProperty("target", "Note2"); // 付费对象

prop.setProperty("amount", "350"); // 付费金额

StatService.*trackCustomKVEvent*(**this**, "mta\_tag\_user\_pay", prop);

## 高级功能

### 用户画像

用户画像统计需要app开发者主动上报QQ号码，若没有上报QQ号码，则无法使用用户画像及QQ登录数等特色功能。

***void StatService.reportQQ(Context ctx, String qq)***

**参数**：Ctx 页面的设备上下文

qq qq号码

**调用位置**：用户登陆成功或其它能获取qq号码的地方

@Override

**protected void** userLogin() {

// 获取QQ号码

String qq = getUserQQ();

StatService.**reportQQ**(this, qq);

}

### 游戏统计

统计游戏用户需要调用下面接口方法上报游戏用户ID，分区，等级相关信息。若在App用户使用过程中，相关信息发生改变，需要重新调用接口上报。

***void StatService.reportGameUser(Context ctx, StatGameUser gameUser)***

**参数**：ctx 页面的设备上下文

gameUser 游戏玩家对象，需要设置分区分服、玩家账号和等级，即需要调用StatGameUser类的set方法。

**调用位置**：游戏用户登陆

StatGameUser方法列表

|  |  |
| --- | --- |
| **接口名** | **说明** |
| setWorldName(String worldName) | 游戏区服名称或标志 |
| setAccount(String account) | 玩家账号 |
| setLevel(String level) | 等级 |

@Override

**protected void** userLogin() {

// 游戏用户上报

StatGameUser gameUser = new StatGameUser();

gameUser.setWorldName("world1");

gameUser.setAccount("account1");

gameUser.setLevel("100");

StatService.reportGameUser(ctx, gameUser);

}

### 在线配置更新

开发者在腾讯移动分析网站上设置Key-Value值之后，可以调用下面的接口动态获取线上最新的参数值。

何时更新本地参数：用户在前台配置在线参数，并不是实时下发的，而是当SDK上报会话统计日志时才会更新。调试时，可在配置参数10分钟后，让app退到后台超过30秒发生超时或把app进程杀死重启，产生一个会话，便会更新。

***String StatConfig.getCustomProperty(String key)***

**参数**：key 用户在前台配置的key

**返回值**：对应key的value值，若不存在则返回null

***String StatConfig.getCustomProperty(String key, String defaultValue)***

**参数**：key 用户在前台配置的key

**返回值**：对应key的value值，若不存在则返回defaultValue

**protected void** someAction() {

// 获取在线参数onlineKey

String onlineValue = StatConfig.getCustomProperty("onlineKey", "off" );

if(onlineValue.equalsIgnoreCase("on")){

// do something

}else{

// do something else

}

}

## 数据上报

### 数据上报策略

设置数据上报策略，可以有效节省流量。使用以下3种方式调整app的数据上报策略：

1. app启动时指定上报策略（默认为APP\_LAUNCH）

***void StatConfig.setStatSendStrategy(StatReportStrategy statSendStrategy)***

腾讯移动分析目前支持的上报策略包括6种。

|  |  |  |
| --- | --- | --- |
| **编号** | **策略名称** | **说明** |
| 1 | INSTANT | 实时发送，app每产生一条消息都会发送到服务器。 |
| 2 | ONLY\_WIFI | 只在wifi状态下发送，非wifi情况缓存到本地。 |
| 3 | BATCH | 批量发送，默认当消息数量达到30条时发送一次。 |
| 4 | *APP\_LAUNCH* | 只在启动时发送，本次产生的所有数据在下次启动时发送。 |
| 5 | DEVELOPER | 开发者模式，只在app调用void commitEvents(Context)时发送，否则缓存消息到本地。 |
| 6 | PERIOD | 间隔一段时间发送，每隔一段时间一次性发送到服务器。 |

SDK默认为APP\_LAUNCH+wifi下实时上报，对于响应要求比较高的应用，比如竞技类游戏，可关闭wifi实时上报，并选择APP\_LAUNCH或PERIOD上报策略。

1. 考虑到wifi上报数据的代价比较小，为了更及时获得用户数据，SDK默认在WIFI网络下实时发送数据。可以调用下面的接口禁用此功能（在wifi条件下仍使用原定策略）。

***void StatConfig.setEnableSmartReporting (boolean isEnable)***

1. 通过在Web界面配置，开发者可以在线更新上报策略，替换app内原有的策略，替换后的策略立即生效并存储在本地，app后续启动时会自动加载该策略。

上面3种方式的优先级顺序：wifi条件下智能实时发送>web在线配置>本地默认。

### 数据上报相关的设置接口

1. 设置最大缓存未发送消息个数（默认1024）

***void StatConfig.setMaxStoreEventCount(int maxStoreEventCount)***

缓存消息的数量超过阈值时，最早的消息会被丢弃。

1. （仅在发送策略为BATCH时有效）设置最大批量发送消息个数（默认30）

***void StatConfig.setMaxBatchReportCount(int maxBatchReportCount)***

1. （仅在发送策略为PERIOD时有效）设置间隔时间（默认为24\*60，即1天）

***void StatConfig.setSendPeriodMinutes(int minutes)***

1. 开启SDK LogCat开关（默认false）

***void StatConfig.setDebugEnable(boolean debugEnable)***

（注意：在发布产品时，请将此开关设为false）

## APP设置接口

使用这些函数可以动态调整APP和SDK的相关设置。

1. 会话时长（默认30000ms，30000ms回到应用的用户视为同一次会话）

***void StatConfig.setSessionTimoutMillis(int sessionTimoutMillis)***

1. 消息失败重发次数（默认3）

***void StatConfig.setMaxSendRetryCount(int maxSendRetryCount)***

1. 用户自定义时间类型事件的最大并行数量（默认1024）

***void StatConfig.setMaxParallelTimmingEvents(int max)***

1. 设置安装渠道

***void StatConfig.setInstallChannel(String installChannel)***

1. 设置app key

***void StatConfig.setAppKey(Context ctx, String appkey)***

1. 设置统计功能开关（默认为true）

***void StatConfig.setEnableStatService(boolean enableStatService)***

如果为false，则关闭统计功能，不会缓存或上报任何信息。

1. 设置session内产生的消息数量（默认为0，即无限制）

***void StatConfig.setMaxSessionStatReportCount(int maxSessionStatReportCount)***

如果为0，则不限制；若大于0，每个session内产生的消息数量不会超过此值，若超过了，新产生的消息将会被丢弃。

1. 设置每天/每个进程时间产生的会话数量（默认为20）

为防止开发者调用MTA不合理导致上报大量的会话数量（sesion），SDK默认每天/每个进程时间内最多产生的会话数量，当达到此值时，SDK不再产生并上报新的会话。当进程重启或跨天时，会被清0。

***void StatConfig.setMaxDaySessionNumbers (int maxDaySessionNumbers)***

1. 设置单个事件最大长度（默认为4k，单位：bytes）

为防止上报事件长度过大导致用户流量增加，SDK默认不上报超过4k的单个事件；对于错误异常堆栈事件，异常堆栈长度不超过100（可以超过4k）。

***void StatConfig.setMaxReportEventLength (int maxReportEventLength)***

1. 支持多进程（默认为false）

同一个app多个进程同时使用MTA，请参考注意事项中的“多进程支持”。

***void StatConfig.setEnableConcurrentProcess(boolean enableConcurrentProcess)***

1. 设置自定义用户ID

上报开发者自定义用户ID。

***void StatConfig.setCustomUserId(Context ctx, String customUserId)***

## 注意事项

开发者使用SDK过程中，需要注意的事项。

### 何时调用StatConfig配置接口

为了使StatConfig配置及时生效，请在app初始化时（例如mainActivity.onCreate函数）调用StatConfig提供的接口，保证StatConfig接口在StatService接口前被调用。

### 调试app

开发者在开发app，调试的过程中，需要注意的事项。

1. SDK Debug开关

如果需要查看SDK日志及上报数据内容，请设置StatConfig.setDebugEnable(true)为开启状态，但发布时，请保持关闭状态。

1. App异常处理

SDK默认开启捕获app未处理的异常并在LogCat打印异常信息，如果开发者在调试的时候，不希望SDK对app未处理的异常进行捕获，请将StatConfig.setAutoExceptionCaught(false)设置为禁用状态，发布版本时，再根据情况决定是否开启。

### 发布app

开发者在完成app开发，准备发布的过程中，需要注意的事项。

1. SDK Debug开关

请保持SDK的debug开关为关闭状态，即调用StatConfig.setDebugEnable(false)。

1. App异常处理

如果app上线时，需要收集app未处理的异常，请保持StatConfig.setAutoExceptionCaught(true)为开启状态。

1. 渠道设置

请根据不同的发布渠道，在AndroidManifest.xml配置或代码中调用StatConfig.setInstallChannel()接口设置对应的渠道名称。

1. Appkey及权限

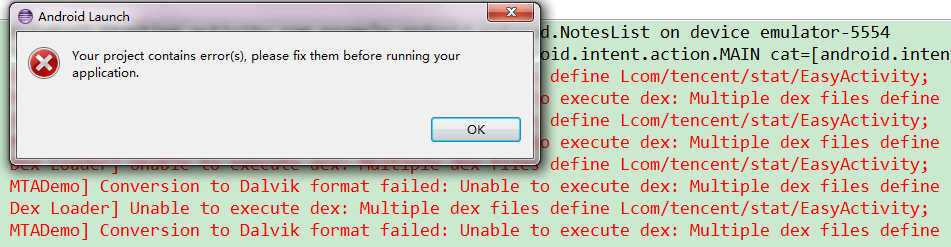
请检查appkey设置是否正确，且AndroidManifest.xml添加SDK所需权限。

### SDK冲突问题

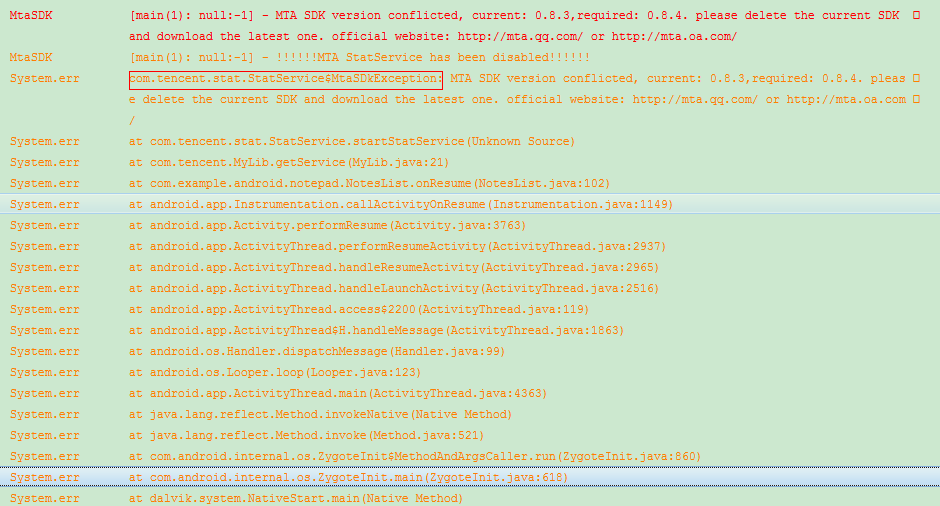
合作方lib或其它app嵌MTA时，调用MTA初始化接口StatService. startStatService(Context ctx, String appkey, String requiredMtaVer)，其中requiredMtaVer为当前app所依赖的版本号常量，当存在多个app或lib同时调用MTA时，MTA能够根据此常量判断当前SDK版本是否满足条件，**只有当前使用的SDK版本不小于requiredMtaVer时，才不会有冲突**。当发生冲突时，可以根据以下方案排除冲突。

#### 1 多个SDK文件

当存在多个MTA SDK文件，在启动app时，Console提示：

 **解决方法**：把版本号比较低的SDK jar包删除即可。

#### 2 MtaSDkException



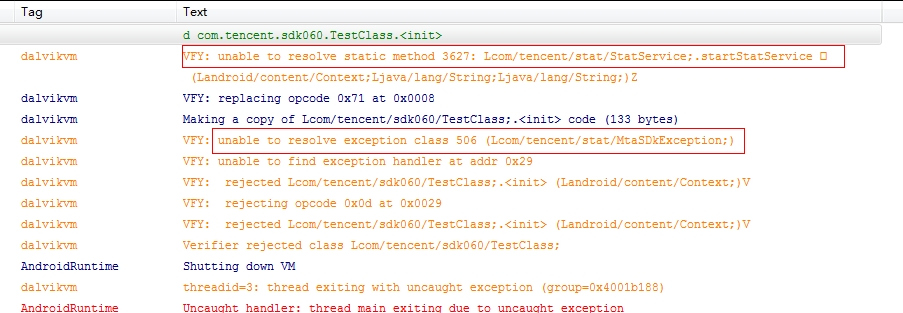
原因：当前所使用的MTA SDK版本低于合作方要求的MTA版本

解决办法：

1. 确保startStatService传递的参数都是正确的，且requiredMtaVer为常量：**com.tencent.stat.common.StatConstants.VERSION**
2. 使用最新SDK：先删除当前版本，然后在[MTA官方网站](#_Step_1_获取AppKey)下载最新的SDK，更新当前开发环境即可。

#### 0.8.4版本之前的SDK

对于早期的开发者，若使用0.8.4之前的旧SDK，调用第三方lib时出现以下异常，是由于老SDK不兼容新版本导致的，请更新至最新版本MTA SDK即可。



### 提示兼容性错误

MTA自带兼容性检测和保护，当出现兼容性问题时，logcat会出现以下类似错误，此时，为了不影响APP正常使用，MTA已自动禁止，不会采集或上报任何数据。



解决办法：

1. 确保APP已授权且没有第三方软件禁止权限；
2. 确保SDK版本号是最新版本；
3. 确保APP本身没有异常，并删除apk，重装安装。

若通过以上2点还没排除问题，很可能是由于设备本身系统兼容性导致，请联系我们，并告知设备型号、Android系统版本和异常信息。

### 多进程支持

支持一个app下多个进程同时使用MTA，假设某一款app有主进程“com.tencent.mta.MainActivity”和子进程“com.tencent.mta.LogService”。

1. 主进程

主进程按照正常的流程接入MTA。

1. 子进程

子进程初始化代码处须调用“StatConfig.setEnableConcurrentProcess(true)”，开启多进程的支持。

主进程和子进程是2个独立的内存空间，但共享同一个appkey，会导致上报多份会话统计数据，因此，建议针对主进程和子进程分别使用不同的appkey，步骤如下：

1. 删除manifest.xml中的“TA\_APPKEY”。
2. 主进程和子进程初始化代码处分别调用StatConfig.setAppKey(this, appkey)，设置各自的appkey。

## 特殊需求

### 只做少数页面统计

MTA使用android系统的onPause和onResume做页面统计和监听用户是否退到后台，如果仅仅只是在某些页面调用了这2个方法，可能会导致上报大量的会话信息，可通过以下办法解决：

1、将session超时时间（默认为30秒）设置到一个合理值，如1小时

2、设置每天/每个进程时间产生的会话数量（默认为20）

### 只统计用户打开app的次数

在app开始的地方，比如在第一个启动的activity的onCreate函数，调用StatService.trackCustomEvent(this, "onLaunch");其中“onLaunch”为自定义事件id，请根据实际情况修改。

### 只统计app的设备、网络等信息

在app开始的地方，可通过触发任意一个自定义事件激活MTA即可，见10.2。

### 只上报app未捕获的异常

在app开始的地方调用StatConfig.setAutoExceptionCaught(true)开启MTA捕获异常的功能，然后再通过任意的自定义事件触发mta即可。

### 只用于接口监控

在被监控的地区调用监控接口reportAppMonitorStat(Context, StatAppMonitor)即可。

### 只做用户画像分析

调用reportQQ(Context, String)。

1. x.x.x为SDK版本号，以实际SDK的名字为准 [↑](#footnote-ref-1)