MTA Android Crash接入指南

本指南用于MTA Android Crash的接入，用于引导crash模块的配置升级、API调用。

Crash模块已全新重构升级，支持Java和所有架构的Native so丰富的异常数据采集，支持完整的堆栈还原，支持实时堆栈展示报表，支持实时监控告警等一系列特性。

# 一、配置升级

## 1.1. 库文件

MTA支持Java和Native异常捕获，其中Java Crash模块默认集成在MTA主体jar包中，Native Crash（即c/c++或so的异常捕获）需要额外添加so文件，并调用API启用，若您的工程涉及到Native编码，建议打开上Native Crash模块，否则，不需要额外添加这部分的文件。

**Java Crash**：使用新的mta 3.x.x.jar替换老版。

**Native Crash**：libMtaNativeCrash\_v2.so，删除老的ibMtaNativeCrash.so，需要注意不同ABI的文件存放，具体见下。

MTA Native Crash支持当前Android系统所支持的所有架构：

armeabi

armeabi-v7a

arm64–v8a

x86

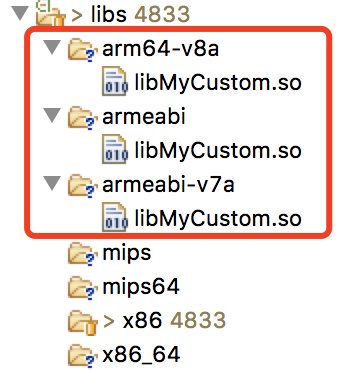
x86\_64

mips

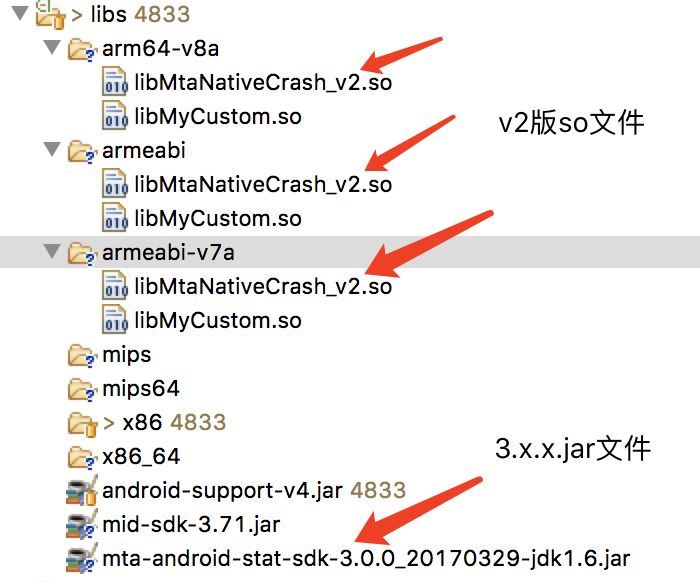
mips64

在集成过程中，一定要注意不同架构SO库文件的存放，否则可能会引起问题，总的原则是：只保留工程所支持的架构SO，不支持的SO千万不要额外导入。下面举例说明。

**集成前**：假设libMyCustom.so为实际工程的SO文件，所支持架构列表只有armeabi、armeabi-v7a、arm64-v8a三种。



**集成后**：根据只保留工程所支持SO架构的原则，那么只需要把Mta Native Crash对应的架构下的so文件复制到工程对应的目录中即可，即MTA armeabi的libMtaNativeCrash\_v2.so复制到工程对应的armeabi目录下，注意一定要匹配，保持支持的架构目录的so文件一致，而不支持的架构不需要添加，一个集成带native crash模块的MTA后的工程示例见下图。



## 1.2. 工程配置

主要的配置参考MTA的接入配置功能，下面列举AndroidMenifest.xml文件的配置部分。

<application

……工程application指标的的其它配置

<!-- MTA配置硕 < -->

<!-- 请将value改为MTA分配的appkey，若已通过API调用可跳过本配置 < -->

<meta-data

android:name="TA\_APPKEY"

android:value="A91LM44JGFLV" />

<!-- 请将value改为APP的发布渠道（市场），若已通过API调用可跳过本配置 < -->

<meta-data

android:name="InstallChannel"

android:value="应用宝" />

<!-- MID 3.x版的配置部分，需要将"您的APP包名"替换成工程实际的包名 < -->

<provider

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

android:authorities="您的APP包名.TENCENT.MID.V3"

android:exported="true" >

</provider>

</application>

<!-- MTA权限配置项 -->

<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" />

# 二、API接口

为了方便使用，最新的Crash模块统一封装成类StatCrashReporter对外提供服务，同时兼容旧的接口。为了方便管理，建议以新的API调用替换旧的接口，具体的API升级见下面。

#### 2.1. Java Crash异常捕获

***void setJavaCrashHandlerStatus (boolean enable)***

**说明**：开启或禁用java异常捕获，初始化不会带来任何的流量和性能消耗。生效后，会注册DefaultUncaughtExceptionHandler，crash时捕获相关信息，存储在本地并上报。可通过添加StatCrashCallback监听Crash发生。

**参数**：enable 是否开启java异常捕获，默认开启；

true：开启；

false：禁用

**对应的get接口**：getJavaCrashHandlerStatus()

**调用位置**：App初始化时调用，比如Application.onCreate或MainActivity.onCreate

**示例**：

StatCrashReporter.getStatCrashReporter(getApplicationContext()).setJavaCrashHandlerStatus(true);

#### 2.2. Native Crash异常捕获

***void setJniNativeCrashStatus (boolean enable)***

**说明**：开启或禁用Native异步捕获，初始化不会带来任何的流量和性能消耗。生效后，会注册signal到native层，crash时捕获相关信息，存储在本地并上报。可通过添加StatCrashCallback监听Crash发生。

**参数**：enable 是否开启Native异常捕获，默认为false；

true：开启；

false：禁用

**对应的get接口**：getJniNativeCrashStatus ()

**调用位置**：App初始化时调用，比如Application.onCreate或MainActivity.onCreate

**示例**：

StatCrashReporter.getStatCrashReporter(getApplicationContext()).setJniNativeCrashStatus(true);

#### 2.3. 监听Crash发生

***void addCrashCallback(StatCrashCallback cb)***

**说明**： 添加StatCrashCallback监听Java或Native的Crash发生。

**参数**：StatCrashCallback crash发生时的StatCrashCallback回调

public interface StatCrashCallback {

// thread：crash的线程信息

// throwable：crash的堆栈信息

public abstract void onJavaCrash(Thread thread, Throwable throwable);

// nativeCrashStacks：native crash的tombstone格式文件

public abstract void onJniNativeCrash(String nativeCrashStacks);

}

**对应的remove接口**：removeCrashCallback(StatCrashCallback cb)

**调用位置**：App初始化时调用，比如Application.onCreate或MainActivity.onCreate

**示例**：

StatCrashReporter.getStatCrashReporter(getApplicationContext()).addCrashCallback(

new StatCrashCallback() {

@Override

public void **onJniNativeCrash**(String nativeCrashStacks) { // native crash happened

// do something

}

@Override

public void **onJavaCrash**(Thread thread, Throwable ex) {// java crash happened

// do something

}

});

#### 2.4. 上报策略

Crash产生时，默认下次App启动时初始化MTA后的3秒开始上报，可通过setReportDelaySecOnStart(int reportDelaySecOnStart) ，reportDelaySecOnStart的单位为秒，范围[0, 10\*60]。

也可以通过设置setEnableInstantReporting设置实时上报，即crash时有网络的条件下尽量立即上报，若上报失败，则下次启动时上报。

为方便开发者调试，在Debug模式，即StatConfig.setDebugEnable(true)，采用实时上报策略。

#### 2.5. 多进程环境

在多进程环境中，若需要在多个进程捕获异常，需要在每个进程都初始化MTA或Native Crash接口，建议在Application.onCreate进行。

2.6 一个完整的示例

示例包括以上主要API调用，请根据需要自行决定调用哪些API。

// 设置appkey，应用的唯一标识，在MTA官网申请得到，也可通过Manifest配置

// 记得把以下appkey替换成自己的

StatConfig.setAppKey(app, "A91LM44JGFLV");

// 设置投放渠道，即应用市场，也可通过Manifest配置

StatConfig.setInstallChannel(app, "应用宝");

StatService.setContext(app);

// 这个是开启Mta的统计功能

StatService.registerActivityLifecycleCallbacks(app);

StatCrashReporter crashReporter = StatCrashReporter.getStatCrashReporter(app);

// 开启异常时的实时上报

crashReporter.setEnableInstantReporting(true);

// 开启java异常捕获

crashReporter.setJavaCrashHandlerStatus(true);

// 开启Native c/c++，即so的异常捕获

// 请根据需要添加，记得so文件

crashReporter.setJniNativeCrashStatus(true);

// crash时的回调，业务可根据需要自选决定是否添加

crashReporter.addCrashCallback(new StatCrashCallback() {

@Override

public void onJniNativeCrash(String tombstoneString) {

// native dump内容，包含异常信号、进程、线程、寄存器、堆栈等信息

// 具体请参考：Android原生的tombstone文件格式

log("MTA StatCrashCallback onJniNativeCrash:\n" + tombstoneString);

}

@Override

public void onJavaCrash(Thread thread, Throwable ex) {

//thread:crash线程信息

// ex:crash堆栈

log("MTA StatCrashCallback onJavaCrash:\n", ex);

}

});