崩溃监控组件介绍

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 修订者 | 修订内容 | 时间 |
| 1.0 | 陈航 | 文档初始化 | 2016.06.19 |
| 1.0 | 陈航 | 增加平台对比、监控原理、使用介绍等内容 | 2016.06.19 |

目录

[1 崩溃监控组件介绍 3](#_Toc454178801)

[2 国内质量跟踪平台对比 3](#_Toc454178802)

[1) Umeng崩溃监控 4](#_Toc454178803)

[2) Bugly质量跟踪平台 4](#_Toc454178804)

[3) 网易云捕 7](#_Toc454178805)

[4) 功能总结 9](#_Toc454178806)

[3 Android端崩溃监控原理 9](#_Toc454178807)

[1. 捕获Java异常 9](#_Toc454178808)

[2. 捕获native异常 10](#_Toc454178809)

[3. CrashHandler使用介绍 12](#_Toc454178810)

## 崩溃监控组件介绍

当应用上线之后，如果出现问题，如程序崩溃了，怎么样定位问题变得尤为重要。如果可以把问题产生时的环境记录下来，开发者可以迅速地定位问题并修复。在国内，已经有很多平台提供了这样的服务，开发者只要在应用中集成服务，即可：实时上报 Crash 信息，全面监控相关异常情况、根据 Crash 次数，形成相关趋势图，及时了解 Crash 动态、提供全面完整的机器运行信息，全方位定位 Crash 信息、详细显示崩溃堆栈信息，直接定位出错代码行号。

## 国内质量跟踪平台对比

如果开发者信赖第三方的服务，也可以在App中集成第三方崩溃监控分析服务，借助平台的能力对App的崩溃问题进行监控分析。



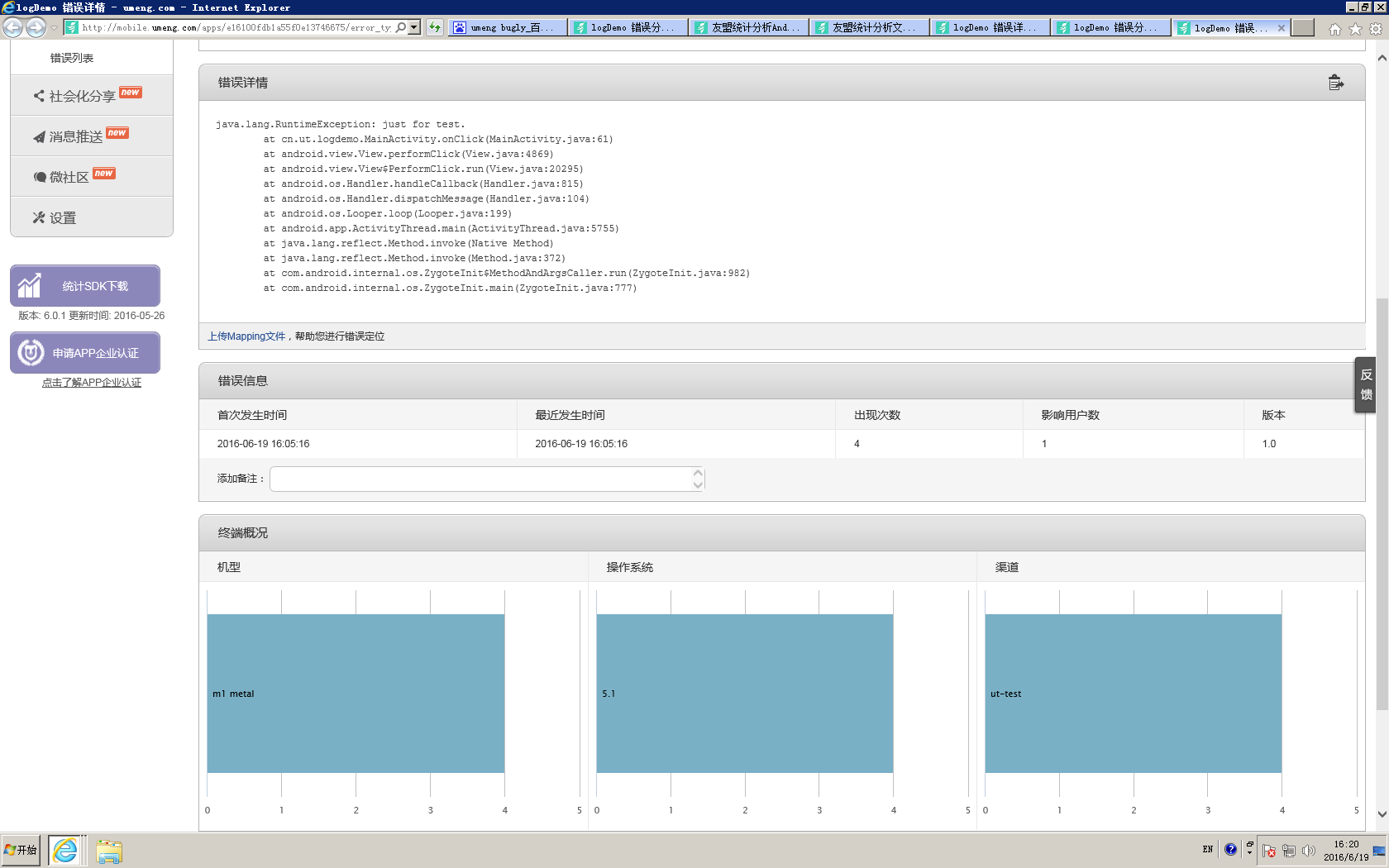
根据CSDN发布的2015年移动开发工具“元素周期表”，我们可以知道，目前较为常用的国内外第三方的崩溃监控分析服务有：

* Fabric/Crashlytics：国外知名的崩溃监控分析服务，被Twitter收购后并入Fabric服务，目前Fabric提供Answers(统计分析)、Beta(内测发布)、Crashlytics(崩溃监控)服务
* Bugsnag：国外轻量级崩溃监控分析服务，支持平台，类别较多，支持WebHook种类较多
* Umeng：国内移动统计分析服务平台，提供统计分析、更新，分享，推送等服务，其中，错误分析也是在统计分析的基础上添加
* Bugly：腾讯Bugly项目组推出移动应用崩溃监控分析平台，提供崩溃、脚本错误、ANR(Android)/卡顿(iOS)问题等监控分析服务
* 网易云捕：网易推出的为开发者轻松排查BUG的较为全面的SDK，能收集以下崩溃异常日志

对比各大平台后发现，国外的平台由于服务器在国外，网络访问慢，导致某些crash信息不能顺利收集，这对国内用户是非常不利的。国内平台主要对比了友盟、腾讯Buglyd和网易云捕，友盟的崩溃监控功能较弱，只有最基本的java异常捕获，Bugly和网易云捕不相上下，功能基本相当。

### Umeng崩溃监控

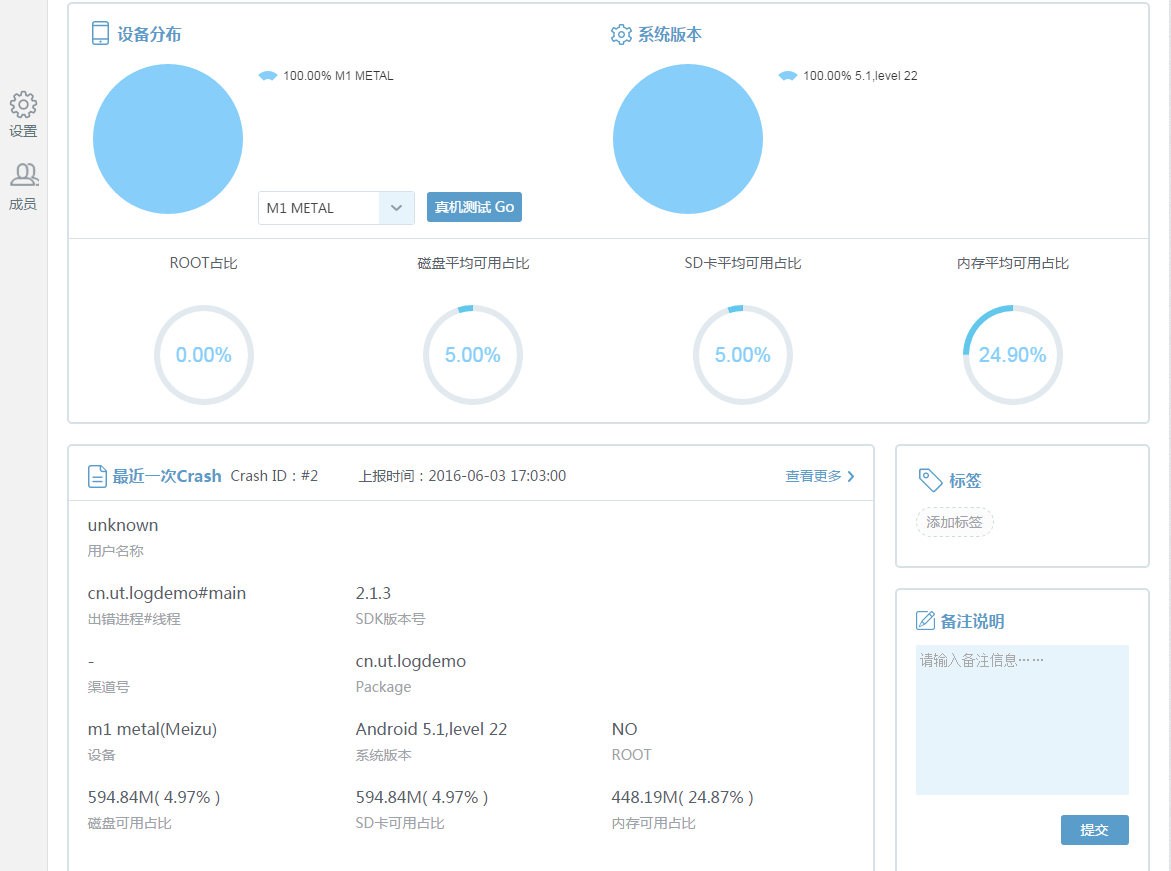
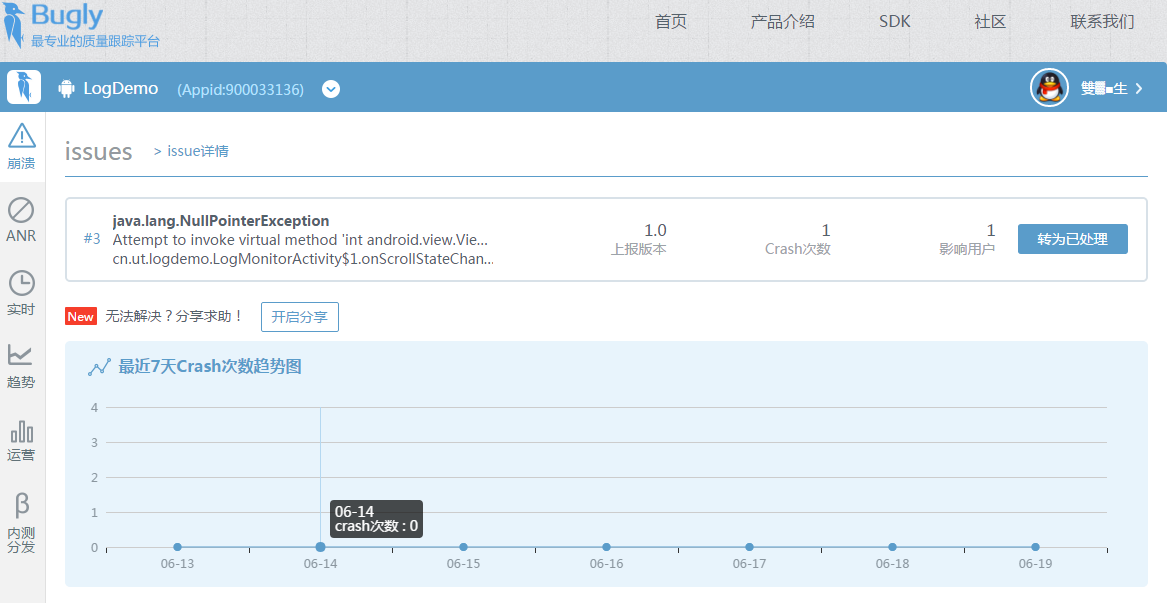
接入了友盟的统计功能之后，一旦App崩溃，则友盟会在应用下次启动的时候，把崩溃信息上报给服务器。上报内容有：出错栈、出错时间、设备机型、设备sdk。界面如下：



对比其他平台，友盟的崩溃监控功能不够强大，网页上是数据展示也没那么友好且不够全面，而且只能捕捉Android java的UncaughtException，对于native异常则无能为力。

### Bugly质量跟踪平台

Bugly是腾讯推出的崩溃监控分析平台，除了捕捉java的UncaughtException外，还可以提供native出错捕获、javascript异常捕获、ANR异常、lua异常捕获。并在网页端提供crash趋势分析、全面的信息展示。另外，Bugly还提供了运营数据统计的功能，可以进行用户分析和用户留存统计。下面是使用Bugly平台的异常信息展示。



除了提供丰富的内容以及强大的统计分析能力外，Bugly SDK的接口使用也很方便，开发者可以根据自己的需求进行配置。如：



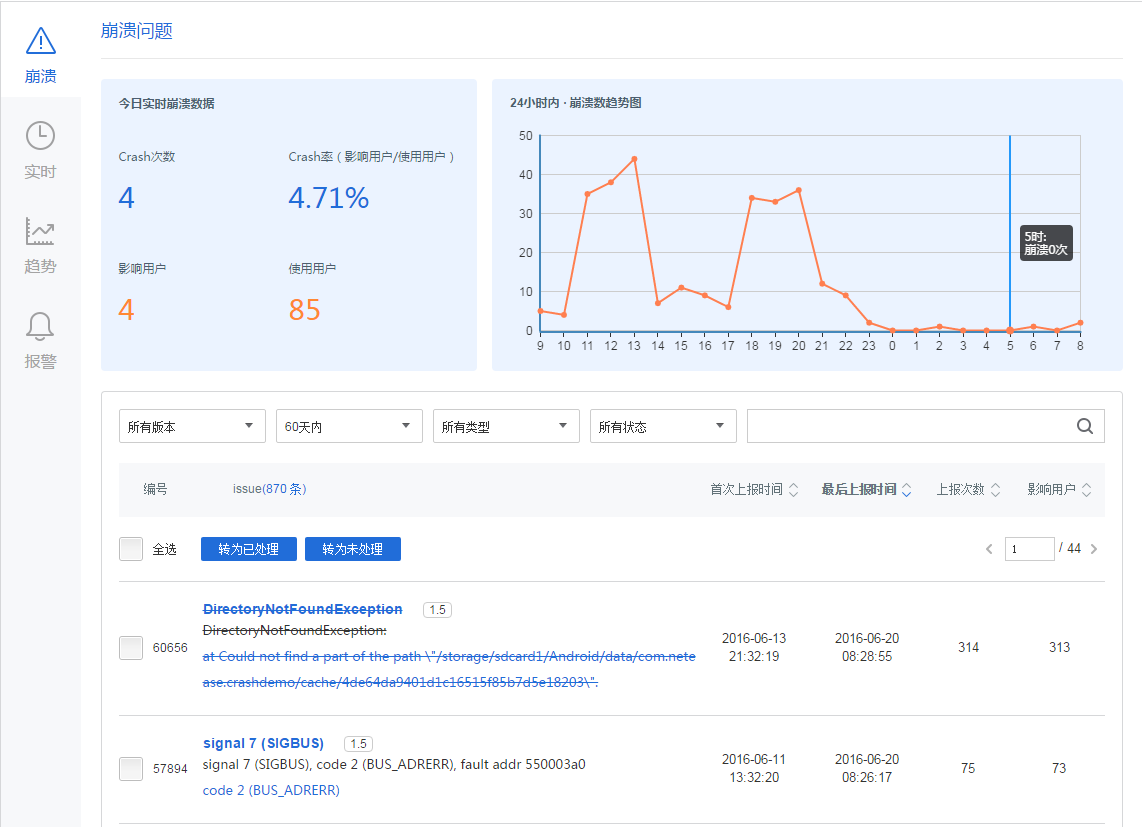


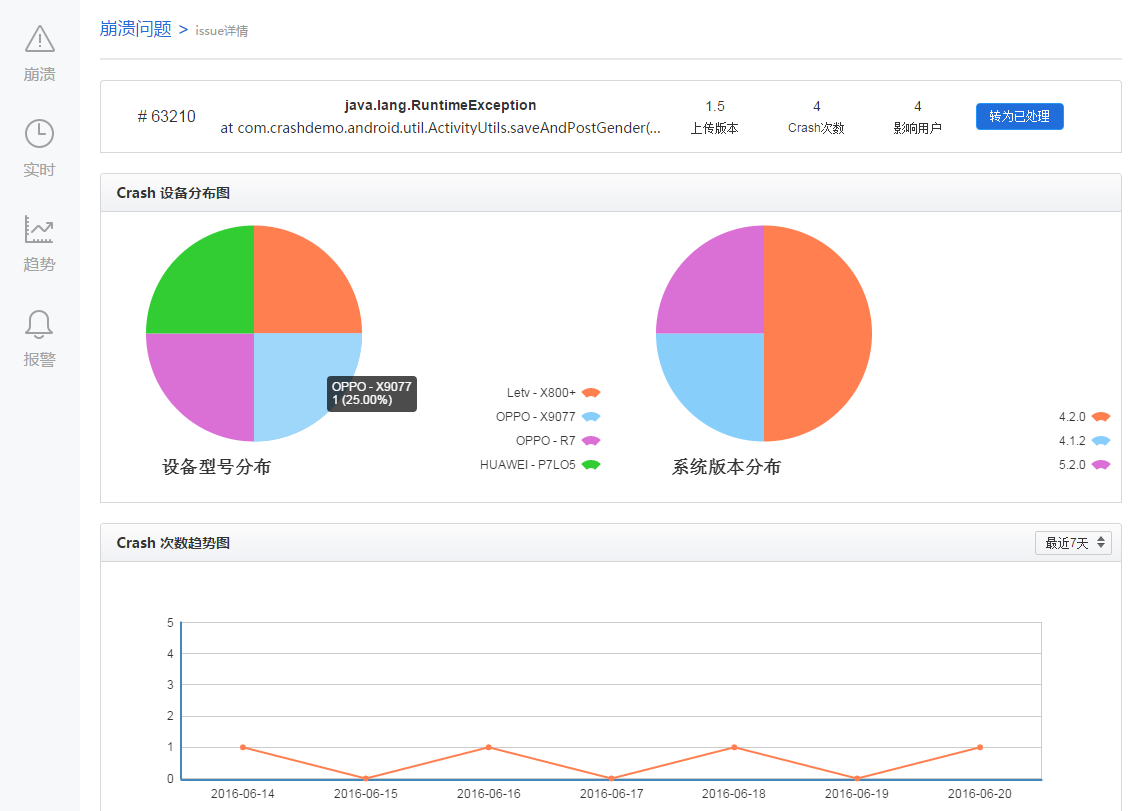
### 网易云捕

网易云捕和腾讯Bugly提供的功能很相似，网易云捕没有运营统计，全都是崩溃监控的功能。在Crash监控方面，不仅可以捕捉java和native异常，还可以实现javascript、lua异常捕捉。网易云捕可以分为四个功能：

* 实时监控
* 趋势分析
* Crash分布查看
* 崩溃堆栈

展示效果如下：





### 功能总结

对比了各个平台后，大致可以把功能分为以下几点：

* 崩溃信息归类展示，如可按照时间、异常类型、异常是否处理、不同渠道、不同版本把问题列表分类
* 崩溃详细信息，包括：设备型号、系统版本、crash次数、崩溃时间、CPU类型、内存、存储空间状态、出错线程、是否root、出错栈、包名、版本号、渠道号
* 趋势分析，对崩溃列表进行统计，比如按时间统计、按crash次数统计、按启动后多少秒内crash统计。
* 报警设置，可以关联开发者手机或者微信公众号，发生crash后提示开发者

## Android端崩溃监控原理

### 捕获Java异常

应用发生崩溃，一部分是由于代码逻辑出现异常，但开发者并没有主动处理(try-catch)，系统抛出(throw)未处理异常(UncaughtException)后，关闭应用进程。

对于这个场景的崩溃，系统有提供API方法让开发者可以统一处理UncaughtException，因此，通过系统API方法可以快速实现未处理异常的崩溃的监听，即：**Thread.setDefaultUncaughtExceptionHandler()** - 监听Java未处理异常。核心代码如下：

**private void** setUncaughtException() {  
 Thread.UncaughtExceptionHandler uncaughtExceptionHandler = **new** Thread.UncaughtExceptionHandler() {  
 @Override  
 **public void** uncaughtException(Thread thread, Throwable ex) {  
 // *TODO: 2016/6/20 收集出错信息，上传服务器、进行后续处理* }  
 };  
 Thread.*setDefaultUncaughtExceptionHandler*(uncaughtExceptionHandler);  
}

一般建议在处理时都按示例代码中方式进行注册监听，即在注册监听之前，先获取默认设置的监听函数，在处理完成后，传递给默认的监听函数。实现未处理异常的监听，可以捕获到应用大部分的崩溃问题。需要注意的是：捕获的未处理的异常，因为线程嵌套问题而被封装为一个RuntimeException，其真正的错误原因是在Throwable对象的Cause对象中，所以，有必要对Cause的对象进行遍历获取

### 捕获native异常

因为应用发生崩溃，还有一部分是因为native代码问题而引发，所以开发者需要对Android Native库(c/c++开发)进行此类崩溃问题的监听。而这个场景的崩溃，则可以依赖Linux/Unix信号处理机制实现监听: sigaction(int signo, void handler, void pre\_handler) - 注册错误信号监听。核心代码如下：

#include <stdbool.h>  
#include <signal.h>  
  
#include "native\_crash\_handler.h"  
#include "utils.h"  
  
static struct sigaction old\_actions[NSIG];  
  
#define THREAD\_NAME "native\_crash\_handler"  
extern JNIEnv \*jni\_get\_env(const char \*name);  
  
// Monitored signals.  
static const int monitored\_signals[] = {  
 SIGILL,  
 SIGABRT,  
 SIGBUS,  
 SIGFPE,  
 SIGSEGV,  
#ifndef \_MIPS\_ARCH  
 SIGSTKFLT,  
#else  
 SIGEMT,  
#endif  
 SIGPIPE  
};  
  
  
*/\*\*  
 \* Callback called when a monitored signal is triggered.  
 \*/*void sigaction\_callback(int signal, siginfo\_t \*info, void \*reserved)  
{  
 JNIEnv \*env;  
 if (!(env = jni\_get\_env(THREAD\_NAME)))  
 return;  
  
 // Call the Java LibVLC method that handle the crash.  
 (\*env)->CallStaticVoidMethod(env, fields.LibVLC.clazz,  
 fields.LibVLC.onNativeCrashID);  
  
 // Call the old signal handler.  
 old\_actions[signal].sa\_handler(signal);  
}  
  
  
void init\_native\_crash\_handler()  
{  
 struct sigaction handler;  
 memset(&handler, 0, sizeof(struct sigaction));  
  
 handler.sa\_sigaction = sigaction\_callback;  
 handler.sa\_flags = SA\_RESETHAND;  
  
 // Install the signal handlers and save their old actions.  
 for (unsigned i = 0; i < sizeof(monitored\_signals) / sizeof(int); ++i)  
 {  
 const int s = monitored\_signals[i];  
 sigaction(s, &handler, &old\_actions[s]);  
 }  
}  
  
  
void destroy\_native\_crash\_handler()  
{  
 // Uninstall the signal handlers and restore their old actions.  
 for (unsigned i = 0; i < sizeof(monitored\_signals) / sizeof(int); ++i)  
 {  
 const int s = monitored\_signals[i];  
 sigaction(s, &old\_actions[s], NULL);  
 }  
}

另外，除了在native代码设置信号监听，还需要在java中设置回调，如上述例子中就是设置LibVLC的onNativeCrash进行监听。

**private static void** onNativeCrash() {  
 **if** (sOnNativeCrashListener != **null**)  
 sOnNativeCrashListener.onNativeCrash();  
}

这样，当native代码发生错误的时候，就能监听到信号，从而触发java里面设置的回调。但是这样还不够，现在还不能收集出错信息，一般发生native错误的时候，我们需要用adb命令把logcat抓取出来，比如抓取最近1000条调试信息，这样服务器才有分析的依据。

## CrashHandler使用介绍

目前，我们已经做了自己的CrashHandler组件，使用方法如下：

**private void** setupCrashHandler() {  
  
 CrashListener crashListener = **new** CrashListener() {  
 @Override  
 **public boolean** onCrash(Context context, CrashInfo crashInfo) {  
 ReportActivity.*startup*(context, crashInfo.toString()); //启动新的页面，显示出错信息  
 android.os.Process.*killProcess*(android.os.Process.*myPid*());  
 System.*exit*(1);  
 **return true**;  
 }  
 };  
 CrashHandler.*register*(mContext, crashListener);  
}

在CrashInfo中，已经收集好了出错线程、调用栈、内存情况、存储卡情况、网络设置等信息。开发者可以根据需要进行使用，如上传到服务器。另外，crash信息还会以文件形式保存在本地，保存路径默认为：/Android/data/[app\_package\_name]/cache/log/

## 后续扩展

现阶段只能捕获java异常和native异常，并且异常信息是保存在本地文件中，在应用上线后远远不够，以下是需要后续扩展的需求：

* lua异常捕获
* 与服务器协商接口。目前想法是把异常信息以json格式上报给服务端
* 定义详细的json字段，如：异常类型、异常名字、出错线程、出错栈、出错时间、软件版本、内存状态、存储状态、网络状态、系统版本等等