Duang~ Android堆栈惨遭毁容？精神哥揭露毁容真相！

2015.3.5 腾讯Bugly 微信分享

最近精神哥接到多个小伙伴的投诉，说无法看懂Bugly崩溃克星页面上显示的堆栈信息！精神哥赶紧把正研究的Top Crash崩溃和心爱的鸡爪放下，开始着手跟进。经分析发现，大家误会Bugly崩溃克星了，这一切都是Proguard搞的鬼！

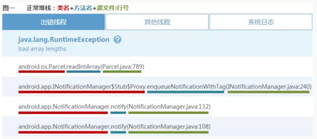
下面请容精神哥一一道来！

> **问题描述**

先看看图一中显示的正常堆栈内容，正常堆栈中每一个调用帧（Frame），都会有3个元素组成：

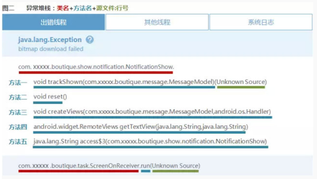
- 类名（Full Class Name，红线标注）；
- 方法名（Method，蓝线标注）；
- 源文件及行号（SourceFile:LineNum，绿线标注）。

图一 正常堆栈：类名+方法名+源文件:行号



我们再看看图二显示的异常堆栈内容，可以发现栈中的某一调用帧（Frame），一个类名下会存在多个方法名，而且第一个方法尾部的源文件及行号是(Unknown Source)，根本看不出源文件和行号。

图二 异常堆栈：类名+方法名+源文件:行号



但，有经验的同学童鞋应该能看出来，这个堆栈是被Proguard还原过的！

怎么看？很简单，出错时JVM生成的堆栈中每一个"方法描述"，只有"方法名"而没有"方法返回类型"及"方法参数"。但被Proguard还原过的堆栈，应该有"方法返回类型"及"方法参数"。

那么这里有两个问题：

为什么堆栈会是Unknown Source？是Bugly崩溃克星没有上报吗？
为什么com.xx.a.a会被还原成多个方法（图2示例中就被还原了5个方法）？

精神哥继续给大家分析分析！

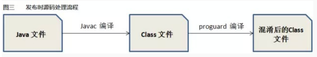
> **为什么堆栈会是Unknown Source？**

是Bugly崩溃克星忘记上报了吗？不是！那是因为，你代码编译的姿势不对！

有经验的童鞋应该会发现，开发阶段上报Bugly的Crash崩溃堆栈都是有源码及行号的，但发布后就变成了Unknown Source，为什么？

如图3所示，我们发布时源码信息会先经过javac编译，再经过proguard混淆，才被打包进发布的apk中，最终Crash崩溃后Bugly获取到的堆栈中有木有源码及行号就要看这两步了。

图三 发布时源码处理流程



所以想让Crash崩溃堆栈不再Unknown Source，需要两个保证：

保证一：javac编译保留源文件名及行号

【TODO】javac编译保留源文件名及行号
源文件名、行号、变量名称，都存在class文件的debug信息中，javac编译时可以选择是否保留debug信息，那么我们肯定是要保留的！下面是使用不同编译方法时保存源文件名及行号的解决方法示例。

命令行javac编译的解决方法

```
javac -g:{lines,source} XXX.java
```

使用Ant编译的解决方法

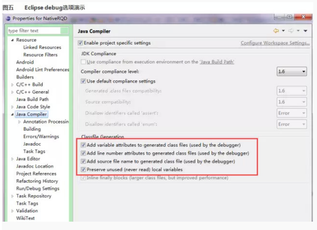
图四 ant脚本debug选项演示

```
<!-- 编译项目的.java文件为.class文件 -->
<target name="compile" depends="init">
    <echo>开始编译...</echo>
    <javac encoding="${encoding}"
        debug="true" debuglevel="source,lines"
        extdirs=""
        source="1.6"
        target="1.6"
        destdir="${classes}"
        includeAntRuntime ="false"
        >
        <src path="${src}"/>
    </javac>
    <echo>结束编译...</echo>

</target>
```

使用Eclipse编译的解决方法

图五 Eclipse debug选项演示



保证二：Proguard混淆中保留原文件名及行号

【TODO】Proguard中keep住源文件及行号

```
-keepattributes SourceFile,LineNumberTable
```

> **为什么com.xx.a.a被还原成多个方法了？**

如图六，大家看到Class1_Promoted类下面的两个方法都被混淆成了a方法。

图六 混淆还原Mapping示例1

```
mapping.txt
1  proguard.retrace.Class1_Promoted -> proguard.retrace.a:
2      void doClass1(int) -> a
3      void doClass1() -> a
4  proguard.retrace.Class2_UnPromoted -> proguard.retrace.Class2_UnPromoted:
5      void doCrash() -> doCrash
6  proguard.retrace.Main_UnPromoted -> proguard.retrace.Main_UnPromoted:
7      void doMain() -> doMain
8      void main(java.lang.String[]) -> main
```

混淆前同名doClass1，混淆后同名a

那么问题来了：假如堆栈中有 proguard.retrace.a.a(Unknown Source)，应该被还原成神马呢？

因为它可以是proguard.retrace.Class_Promoted类下的方法void doClass1(int)，也可以是void doClass1()。所以Proguard还原工具直接把这两个都给你列出来都作为你的还原结果，所以：

```
proguard.retrace.a.a(Unknown Source)
```

被还原成

```
proguard.retrace.Class_Promoted.
void doClass1(int)
void doClass1()
```

图二中的5个方法也就是这么来的。

真不能怪Proguard还原工具，谁让Java堆栈中不给出方法的完整描述或签名（返回值，方法名，方法参数），而只有方法名，但Java语法又允许方法名相同。但这是有解决办法的！

Proguard还原工具其实还能根据行号进行区分，如果你的堆栈已经解决了UnknownSource问题，那么你的还原Mapping文件就不一样了。如图7，大家看到mapping文件中多了"数字:数字"这类内容，这个就是行号，也就是说proguard.retrace.a类中24行到26行属于void doClass(int)方法，29到31行属于void doClass1()方法。

假如堆栈中有proguard.retrace.a.a(Demo.java:25)，就知道要被还原成proguard.retrace.Class_Promoted.void doClass1(int)了。

图七 混淆还原Mapping示例2

```
mapping.txt
1  proguard.retrace.Class1_Promoted -> proguard.retrace.a:
2      18:22:void doClass1(int) -> b
3      24:26:void doClass1(int) -> a
4      29:31:void doClass1() -> a
5  proguard.retrace.Class2_UnPromoted -> proguard.retrace.Class2_UnPromoted:
6      20:21:void doCrash() -> doCrash
7  proguard.retrace.Main_UnPromoted -> proguard.retrace.Main_UnPromoted:
8      18:19:void doMain() -> doMain
9      23:24:void main(java.lang.String[]) -> main
```

> **精神哥总结**

Proguard作为非常优秀的工具，让我们安装包体积变小了，代码混淆更安全了，代码裁剪优化速度更快了，但确实也埋下了一些坑，增加了我们定位崩溃时的成本。跟着精神哥的这篇文章把原文件名和行号补上后，绝大部分的堆栈问题都可以被解决了！

那么堆栈问题就没了吗？有人曾问我：堆栈里显示A方法调用了D方法，跟着D方法崩溃了，但实际查看代码A并没有调用D方法啊？

精神哥汗了，不知道Proguard会做代码优化的么？认真看看是不是有A->B->C->D？如果有！那是因为Proguard大神很喜欢把你没用的->B->C给干掉，直接A->D了！