上帝说要有ANR，于是Bugly就有了ANR上报，那么ANR到底是什么？

> 最近很多童鞋问起精神哥ANR的问题，那么这次就来聊一下，鸡爪怎么泡才好吃，噢不，是如何快速定位ANR。

ANR是什么

简单说，通常就是App运行的时候，duang~卡住了，怎么搞都动不了。当卡住超过一定时间，Android系统认为这就是一次“ANR（Application Not Responding）”。

具体说，在以下情况发生时，会发生ANR（可能在不同ROM 中时间有所更改）：

- 用户的输入在5s内没被App响应；
- BroadcastReceiver的onReceiver()超过10s；
- Service中各生命周期函数执行超过20s。

ANR必须死

用户在App的绝大部分操作，都需要有App的主动回应，比如按下按钮之后按钮样式的改变、下拉滚动条内容的移动、加载资源时的菊花转转转，它们都是“操作-反馈”配对的模式。对于我们手机上最常见的触摸操作，0.1s的响应延迟已经有很明显的卡顿感了。而对于常见的ANR，用户至少要等5s以上！

发生了ANR，往往会弹出对话框，问用户是继续等待还是直接关掉：

相信几乎所有Android手机用户都见过这个煞并卵的ANR对话框，但大部分普通用户根本不知道这个对话框在讲什么，并且往往也只有关闭App。漫长的等待就给我看这个？从用户的体验看，就是心中一万只草泥马奔腾起来撞火车的感受。可见**ANR对于应用的影响并不亚于Crash**。

一般来说，界面相对越不“流畅”的App（说明UI线程耗时操作多）越容易发生ANR（一个输入事件在某个设备A上4秒有了反馈，并不意味着它在其他设备B上是安全的）。ANR其实就是界面卡顿的极端情况。反过来，只要通过合理的方案消灭了App出现的ANR，往往也同时会使App展示界面表现会更加顺滑流畅。

一些典型的ANR 问题场景

这里举几个容易发生ANR的场景：

1）最常见的错误，UI线程等待其它线程释放某个锁，导致UI线程无法处理用户输入；

2）游戏中每帧动画都进行了比较耗时的大量计算，导致CPU忙不过来；

3）Web应用中，网络状态不稳定，而界面在等待网络数据；

4）UI线程中进行了一些磁盘IO（包括数据库、SD卡等等）的操作，在个别设备上因为硬件损坏等原因阻塞住了；

5）手机被其他App占用着CPU，自己获取不到足够的CPU 时间片，纯属误伤。

通过ANR 日志定位问题

当ANR发生时，我们往往通过Logcat和traces文件（目录/data/anr/）的相关信息输出去定位问题。主要包含以下几方面：

1）基本信息，包括进程名、进程号、包名、系统build号、ANR 类型等等；

2）CPU使用信息，包括活跃进程的CPU 平均占用率、IO情况等等；

3）线程堆栈信息，所属进程包括发生ANR的进程、其父进程、最近有活动的3个进程等等。

这里举个简单的例子（实际上因为各App所处环境各异，可能出现各种各样复杂的ANR情况）当App运行卡住，弹出ANR对话框，查看Logcat输出：

```
ActivityManager: ANR in com.tencent.bugly.demo (com.tencent.bugly.demo/.MainActivity)
ActivityManager: PID: 18617
ActivityManager: Reason: Input dispatching timed out (Waiting because the touched window has not finished processing the input events that were previously delivered to it.)
ActivityManager: Load: 18.42 / 18.09 / 18.29
ActivityManager: CPU usage from 5924ms to 475ms ago:
ActivityManager: 93% 18617/com.tencent.bugly.demo: 93% user + 0% kernel / faults: 75 minor
"""
ActivityManager: CPU usage from 2906ms to 3429ms later:
ActivityManager: 96% 18617/com.tencent.bugly.demo: 96% user + 0% kernel
"""
ActivityManager: 55% TOTAL: 51% user + 3.8% kernel
```

分析一下，从Logcat可以得到以下信息：

com.tencent.bugly.demo这个App的MainActivity发生了ANR，进程号18617；
ANR原因：用户输入超时；
ANR发生前、后一段时间分别附在情况：在ANR发生前后，CPU有90+%耗费在这个demo上，说明很可能是这个demo自身性能引起的。

接下来再看traces文件确认：

```
----- pid 18617 at xxxx -----
Cmd line: com.tencent.bugly.demo
JNI: CheckJNI is off; workarounds are off; pins=0; globals=272 (plus 2 weak)
DALVIK THREADS:
"main" prio=5 tid=1 SUSPENDED
| group="main" sCount=1 dsCount=0 obj=0x415e4e58 self=0x415d3028
| sysTid=18617 nice=0 sched=0/0 cgrp=apps handle=1074372948
| state=S schedstat=( 38588000572 591063492 5767 ) utm=3846 stm=12 core=0
at com.tencent.bugly.demo.MainActivity$3.doCalc(MainActivity.java:-38)
at com.tencent.bugly.demo.MainActivity$3.onClick(MainActivity.java:33)
"""
```

分析一下，traces文件中包含以下信息：

1、进程号：18617；包名：com.tencent.bugly.demo；

2、发生ANR时，main线程被挂起（也可能是其他等待状态，比如TIMED_WAIT）；

3、线程的几个重要参数：

- group：线程组名称“main”；
- sCount：Suspended个数“1”；
- obj：线程的Java对象地址；
- self：线程的Native对象地址；
- sysTid：线程号（这里主线程的线程号=进程号）“18617”；

4、具体堆栈：从堆栈可以很清晰看出是doCalc()方法出的问题，由onClick触发。

综合以上分析，问题还原为：com.tencent.bugly.demo这个App的MainActivity中有个耗时的doCalc方法在跑，无法响应用户的触摸或按键输入。OK，接下来在代码里找问题就好了。

如何解决ANR

当然是尽可能减少UI线程的耗时操作，以及BroadcastReceiver、Service生命周期中的标准回调方法啦。

Android官方文档建议：

1）使用AsyncTask类，可以很方便地实现子线程耗时操作与UI更新；

2）对于BroadcastReceiver的耗时操作，建议放到Service中执行；

3）对于自建的Thread，可以通过Handler使之与UI 线程通信（这里需要注意的是，Thread默认优先级和UI线程是一样的，建议设置一般线程优先级为Process.THREAD_PRIORITY_BACKGROUND）。

这些方案大家应该都知道，不过仍难免有大量的ANR是写代码时忽略了，在测试时没发生，最终在用户的手机上出现的。回想一下是不是都经历过用户会反馈“App卡死没反应了”，但开发GG客服MM们却又因为缺少日志或无法复现而束手无策？因此**要修复ANR，首先是要能发现用户ANR了，并且能知道是哪段代码导致ANR了，这样才能谈修复。**

为了帮助广大开发者解决这一难题，**腾讯Bugly针对iOS的卡顿及Android的ANR提供监测服务即将上线**，协助开发者轻松定位问题。