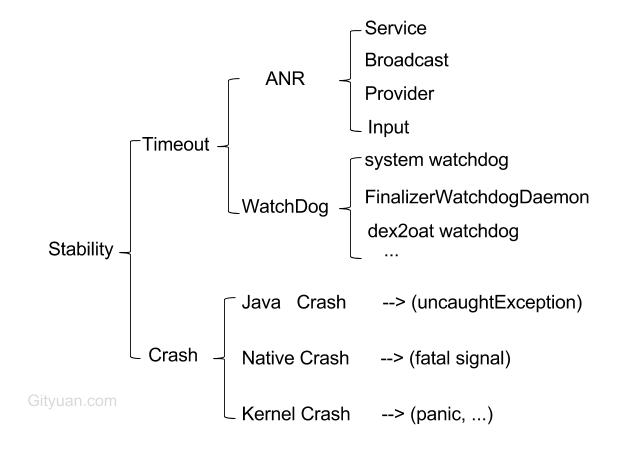
**一. 稳定性简述**

Android系统稳定性对于用户体验至关重要. 对于稳定性问题从表现来看有: 死机重启, 自动关机, 无法开机,冻屏,黑屏以及闪退, 无响应等情况; 从技术层面来划分无外乎两大类: 长时间无法执行完成(Timeout) 以及异常崩溃(crash). 主要分类如下:



**1.1 Timeout**

长时间无法执行完成, 这只是描述性命题, 对于系统来说必须指定每一项操作超过指定阈值美誉执行完成, 则判定为超时(Timeout).

对于Android系统来说,比较常见的便是Service, Broadcast, provider以及input, 当普通app进程超过一定时间没有执行完, 则会弹出应用无响应(Application Not Responding, ANR)的对话框. 如果该app运行在system进程, 更准确的来说,应该是(System Not Responding, SNR). 虽然有ANR和SNR之分, 但习惯上大家都统称为ANR问题.

[理解Android ANR的触发原理](http://gityuan.com/2016/07/02/android-anr/), 对于组件ANR问题, 有些是需要的执行时间比较长, 即便触发ANR, 只要再多给些时间还是可以正常运行的; 有些则是由于发生了死锁,即便给再长的时间都无法恢复的问题.

* Service Timeout:比如前台服务在20s内未执行完成；
* broadcast Timeout：比如前台广播在10s内未执行完成
* ContentProvider Timeout：内容提供者执行超时
* InputDispatching Timeout: 输入事件分发超时5s，包括按键和触摸事件。

除了ANR, 还有另一个类型那就是WatchDog.[WatchDog工作原理](http://gityuan.com/2016/06/21/watchdog/), 最为常见的便是运行在system进程中的”watchdog”线程; 还有运行在各个app进程(包括system进程)的”FinalizerWatchdogDaemon”,该线程用于监控执行GC的过程中, 守护线程”FinalizerDaemon”回收某个对象时间过长的监视器; 当然不至于这些, 还有dex2oat, wifi等watchdog.

当发生ANR或许WatchDog后, 便需要收集系统相关信息,用于分析和修复异常, 见[理解Android ANR的信息收集过程](http://gityuan.com/2016/12/02/app-not-response/). 整个过程中进程Trace的输出是最为核心的环节，另外该过程会清空/data/anr/traces.txt的老文件, 那么原来之前的traces信息一般地会先输出到dropbox, 有些情况就会丢失traces. Java和Native进程采用不同的策略，如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 进程类型 | trace命令 | 文章 | 描述 |
| Java | kill -3 [pid] | [解读Java进程的Trace文件](http://gityuan.com/2016/11/26/art-trace/) | 不适用于Native进程 |
| Native | debuggerd -b [pid] | [Native进程之Trace原理](http://gityuan.com/2016/11/27/native-traces/) | 也适用于Java进程 |

**1.2 Crash**

异常崩溃(Crash)的问题, 毫无疑问这不是时间上能解决的问题, 而是出现了未知的异常. 一旦触发崩溃会出现相应的调用栈, 但不糊输出各个进程的traces.

对于Java层Crash, [理解Java Crash处理流程](http://gityuan.com/2016/06/24/app-crash/), 往往是抛出了一个未捕获的异常uncaughtException而导致的崩溃. 那是不是把所有的异常都catch住系统就没有问题呢, 这个是要分情况的,有时候的异常强制崩溃可能会留下更大的问题, 有些异常的抛出是需要深入分析Root Cause,从根源来解决问题, 而非简单粗糙的捕获住所有的异常.

对于Native层Crash, [理解Native Crash处理流程](http://gityuan.com/2016/06/25/android-native-crash/), 则是由于进程收到signal信号而引发的崩溃.当进程收到信号,并会触发信号处理函数,通过socket发送信息到debuggerd进程, debuggerd进程收到事件后通过 ptrace attach到目标进程, 获取cpu/memory/traces等关键信息后dettach. Native crash情况比较多, 其中最为场景便是SIGSEGV段错误异常, 往往是内存出现异常,比如访问了权限不足的内存地址等.

对于Kernel层Crash, 这是比较难分析的一大类, 不少情况是跟硬件导致的,比较cpu问题, 硬件驱动问题等.

# Throwable

## 一.概述

Android有一套异常处理机制, 分析Crash时最常见的便是先查看其调用栈stackTrace. 对于调用栈, 是从下往上调用的. 其中经常会遇到”Caused by”, 以及”… 8 more”等信息, 具体是什么含义呢, 本文就是解答这个问题.

### 1.1 初始化Throwable对象

[-> Throwable.java]

public Throwable(String detailMessage, Throwable cause) {

this.detailMessage = detailMessage;

this.cause = cause;

this.stackTrace = EmptyArray.STACK\_TRACE\_ELEMENT;

fillInStackTrace(); // [见小节1.2]

}

### 1.2 fillInStackTrace

[-> Throwable.java]

public Throwable fillInStackTrace() {

if (stackTrace == null) {

return this;

}

stackState = nativeFillInStackTrace();

stackTrace = EmptyArray.STACK\_TRACE\_ELEMENT;

return this;

}

## 二. 打印调用栈

### 2.1 printStackTrace

[-> Throwable.java]

public void printStackTrace() {

printStackTrace(System.err);

}

public void printStackTrace(PrintStream err) {

try {

printStackTrace(err, "", null); //[见小节2.2]

} catch (IOException e) {

throw new AssertionError();

}

}

### 2.2 printStackTrace

[-> Throwable.java]

private void printStackTrace(Appendable err, String indent, StackTraceElement[] parentStack) throws IOException {

err.append(toString()); //[见小节2.2.1]

err.append("\n");

StackTraceElement[] stack = getInternalStackTrace(); //[见小节2.2.2]

if (stack != null) {

//parentStack为null, 则duplicates=0;

int duplicates = parentStack != null ? countDuplicates(stack, parentStack) : 0;

for (int i = 0; i < stack.length - duplicates; i++) {

err.append(indent);

err.append("\tat ");

err.append(stack[i].toString());

err.append("\n");

}

if (duplicates > 0) {

...

}

}

//打印suppressed异常

if (suppressedExceptions != null) {

for (Throwable throwable : suppressedExceptions) {

err.append(indent);

err.append("\tSuppressed: ");

throwable.printStackTrace(err, indent + "\t", stack);

}

}

Throwable cause = getCause(); //[见小节2.2.3]

if (cause != null) {

err.append(indent);

err.append("Caused by: ");

cause.printStackTrace(err, indent, stack); //[见小节2.2.4]

}

}

其中indent为空.

#### 2.2.1 toString

[-> Throwable.java]

public String toString() {

//获取detailMessage

String msg = getLocalizedMessage();

//获取exception类名

String name = getClass().getName();

if (msg == null) {

return name;

}

return name + ": " + msg;

}

这是输出的第一行.例如:

1. detailMessage为”Error receiving broadcast Intent”;
2. name 为java.lang.RuntimeException;

则组合为: java.lang.RuntimeException: Error receiving broadcast Intent

#### 2.2.2 getInternalStackTrace

private StackTraceElement[] getInternalStackTrace() {

if (stackTrace == EmptyArray.STACK\_TRACE\_ELEMENT) {

// 当stackTrace为空, 则获取native调用栈

stackTrace = nativeGetStackTrace(stackState);

stackState = null;

return stackTrace;

} else if (stackTrace == null) {

return EmptyArray.STACK\_TRACE\_ELEMENT;

} else {

return stackTrace;

}

}

#### 2.2.3 getCause

public Throwable getCause() {

if (cause == this) {

return null;

}

return cause;

}

默认cause等于this, 当创建的Throwable带有cause的话, 则返回的便是新的Throwable对象.

### 2.3 打印Caused by

[-> Throwable.java]

private void printStackTrace(Appendable err, String indent, StackTraceElement[] parentStack) throws IOException {

err.append(toString());

err.append("\n");

StackTraceElement[] stack = getInternalStackTrace();

if (stack != null) {

//获取栈帧的重复个数 [见小节2.3.1]

int duplicates = parentStack != null ? countDuplicates(stack, parentStack) : 0;

// 从栈顶往下依次输出 调用栈 直到跟parentStack重复的栈为止.

for (int i = 0; i < stack.length - duplicates; i++) {

err.append(indent);

err.append("\tat ");

err.append(stack[i].toString());

err.append("\n");

}

if (duplicates > 0) {

//最终以省略号结尾, 并带上栈帧重复的个数

err.append(indent);

err.append("\t... ");

err.append(Integer.toString(duplicates));

err.append(" more\n");

}

}

if (suppressedExceptions != null) {

...

}

Throwable cause = getCause();

if (cause != null) {

... //第二次cause为空,一般为空. 也存在级联情况

}

}

#### 2.3.1 countDuplicates

private static int countDuplicates(StackTraceElement[] currentStack, StackTraceElement[] parentStack) {

int duplicates = 0;

int parentIndex = parentStack.length;

for (int i = currentStack.length; --i >= 0 && --parentIndex >= 0;) {

StackTraceElement parentFrame = parentStack[parentIndex];

//比如栈帧相同的个数

if (parentFrame.equals(currentStack[i])) {

duplicates++;

} else {

break;

}

}

return duplicates;

}

比较parentStack和当前currentStack, 分别从栈底开始比较, 最终得到栈帧重复个数duplicates

## 三. 实例

举例说明:

java.lang.RuntimeException: Error receiving broadcast Intent{}

in com.android.server.notification.NotificationManagerService$3@ef5346c

at android.app.LoadedApk$ReceiverDispatcher$Args.run(LoadedApk.java:1134)

at android.os.Handler.handleCallback(Handler.java:751)

at android.os.Handler.dispatchMessage(Handler.java:95)

at android.os.Looper.loop(Looper.java:154)

at com.android.server.SystemServer.run(SystemServer.java:352)

at com.android.server.SystemServer.main(SystemServer.java:220)

at java.lang.reflect.Method.invoke(Native Method)

at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:874)

at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:764)

Caused by: java.lang.IndexOutOfBoundsException: Index: 0, Size: 0

at java.util.ArrayList.get(ArrayList.java:411)

at com.android.server.notification.NotificationManagerService$NotificationRankers.onUserSwitched(NotificationManagerService.java:3912)

at com.android.server.notification.NotificationManagerService$3.onReceive(NotificationManagerService.java:804)

at android.app.LoadedApk$ReceiverDispatcher$Args.run(LoadedApk.java:1124)

... 8 more

可以看出, Args.run()的1134行抛出异常RuntimeException; 造成这个异常的原因是该方法的1124行代码出现IndexOutOfBoundsException(数组越界). 这两个stack栈底重复的栈帧个数为8个. 既然是重复了8个, 那么展开就等价于:

java.lang.RuntimeException: Error receiving broadcast Intent{}

in com.android.server.notification.NotificationManagerService$3@ef5346c

at android.app.LoadedApk$ReceiverDispatcher$Args.run(LoadedApk.java:1134)

at android.os.Handler.handleCallback(Handler.java:751)

at android.os.Handler.dispatchMessage(Handler.java:95)

at android.os.Looper.loop(Looper.java:154)

at com.android.server.SystemServer.run(SystemServer.java:352)

at com.android.server.SystemServer.main(SystemServer.java:220)

at java.lang.reflect.Method.invoke(Native Method)

at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:874)

at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:764)

Caused by: java.lang.IndexOutOfBoundsException: Index: 0, Size: 0

at java.util.ArrayList.get(ArrayList.java:411)

at com.android.server.notification.NotificationManagerService$NotificationRankers.onUserSwitched(NotificationManagerService.java:3912)

at com.android.server.notification.NotificationManagerService$3.onReceive(NotificationManagerService.java:804)

at android.app.LoadedApk$ReceiverDispatcher$Args.run(LoadedApk.java:1124)

//以下部分是duplicates部分

at android.os.Handler.handleCallback(Handler.java:751)

at android.os.Handler.dispatchMessage(Handler.java:95)

at android.os.Looper.loop(Looper.java:154)

at com.android.server.SystemServer.run(SystemServer.java:352)

at com.android.server.SystemServer.main(SystemServer.java:220)

at java.lang.reflect.Method.invoke(Native Method)

at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:874)

at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:764)