Monkey注入的时间间隔对系统测试的影响分析

版本： V1.0

日期：2018-08-14

前 言

文档说明

针对monkey的注入时间间隔分析对于系统测试的影响

阅读对象

处理monkey稳定性问题的工程师。

内容介绍

本文档采用下面醒目标志来表示在操作过程中应该特别注意的地方。

注意：

提醒操作中应注意的事项。

说明：

说明比较重要的事项。

相关文档

声明

本文件所含数据和信息都属于紫光展锐机密及紫光展锐财产，紫光展锐保留所有相关权利。当您接受这份文件时，即表示您同意此份文件内含机密信息，且同意在未获得紫光展锐同意前，不使用或复制、整个或部分文件。紫光展锐有权在未经事先通知的情况下，对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负责任何与文件相关的直接或间接的、任何伤害或损失。

**目 录**

[第1章 Monkey测试简介 5](#_Toc523487465)

[1.1 Monkey测试介绍 5](#_Toc523487466)

[1.2 Monkey参数说明 5](#_Toc523487467)

[1.3 Monkey代码中参数的处理逻辑 9](#_Toc523487468)

[1.4 测试部门使用的monekey 参数 19](#_Toc523487469)

[第2章 monkey中时间间隔对于系统测试的影响 20](#_Toc523487470)

[版本历史 23](#_Toc523487471)

# Monkey测试简介

## Monkey测试介绍

Monkey是Android中的一个命令行工具，可以运行在模拟器里或实际设备中。它向系统发送伪随机的用户事件流(如按键输入、触摸屏输入、手势输入等)，实现对APP进行压力测试。Monkey测试是一种为了测试软件的稳定性、健壮性的快速有效的方法。

## Monkey参数说明

Monkey命令中的参数可以通过adb shell monkey –help来查看：

bash arg: -help

args: [-help]

arg: "-help"

usage: monkey

**[-p ALLOWED\_PACKAGE [-p ALLOWED\_PACKAGE] ...]**

如果用此参数指定了一个或几个包，Monkey将只允许系统启动这些包里的Activity。如果你的应用程序还需要访问其它包里的Activity(如选择取一个联系人)，那些包也需要在此同时指定。如果不指定任何包，Monkey将允许系统启动全部包里的Activity。要指定多个包，需要使用多个 -p选项，每个-p选项只能用于一个包。

**[-c MAIN\_CATEGORY [-c MAIN\_CATEGORY] ...]**

如果用此参数指定了一个或几个类别，Monkey将只允许系统启动被这些类别中的某个类别列出的Activity。如果不指定任何类别，Monkey将选 择下列类别中列出的Activity： Intent.CATEGORY\_LAUNCHER或Intent.CATEGORY\_MONKEY。要指定多个类别，需要使用多个-c选项，每个-c选 项只能用于一个类别。

**[--ignore-crashes]**

当应用程序崩溃或发生任何异常时，Monkey将停止运行。如果设置此选项，Monkey将继续向系统发送事件，直到计数完成。

**[--ignore-timeouts]**

当应用程序发生任何超时错误(如“Application Not Responding”对话框)时，Monkey将停止运行。如果设置此选项，Monkey将继续向系统发送事件，直到计数完成。

**[--ignore-security-exceptions]**

当应用程序发生许可错误(如启动一个需要某些许可的Activity)时，Monkey将停止运行。如果设置了此选项，Monkey将继续向系统发送事件，直到计数完成。

**[--monitor-native-crashes]**

监视并报告Android系统中native层的崩溃事件。如果设置--kill-process-after-error，系统将停止运行。

**[--ignore-native-crashes]**

当Android系统中native层发生异常时，Monkey将停止运行。如果设置此选项，Monkey将继续向系统发送事件，直到计数完成。

**[--kill-process-after-error]**

一般情况下，Monkey 发生了某个错误将停止，而出问题的应用会留在系统上继续执行，设置该选项它会通知系统停止发生错误的进程**。**

**[--hprof]**

使用此参数时会在 Monkey 启动前后生成内存的快照，在设备的 /data/misc 目录下生成一个 hprof 文件。(需要验证下，TBD)

**[--match-description TEXT]**

仅报告与描述匹配的信息

**[--pct-touch PERCENT]**

点击事件百分比(down up)

**[--pct-motion PERCENT]**

直线滑动事件百分比(down 滑动 up)

**[--pct-trackball PERCENT]**

轨迹球滑动事件百分比(手机端基本上没有 TBD)

**[--pct-syskeys PERCENT]**

导航栏、Home、Back、音量键等

**[--pct-nav PERCENT]**

导航事件：上下左右(一般没有该事件)

**[--pct-majornav PERCENT]**

导航事件：返回，确认，菜单(一般没有该事件)

**[--pct-appswitch PERCENT]**

各Activity的启动比率，启动得越多，越容易覆盖更多的Activity

**[--pct-flip PERCENT]**

键盘翻转事件(键盘弹出及取消操作TBD)

**[--pct-anyevent PERCENT]**

其他任意事件(例如：不常用事件，各种按键之类 TBD)

**[--pct-pinchzoom PERCENT]**

多点手势的缩放事件

**[--pct-permission PERCENT]**

权限事件的百分比(包括：生成权限和取消权限)

**[--pkg-blacklist-file PACKAGE\_BLACKLIST\_FILE]**

黑名单列表，不允许列表中的应用被测试，不能和白名单一起使用

**[--pkg-whitelist-file PACKAGE\_WHITELIST\_FILE]**

白名单列表，只允许列表中应用被测试，相当于多个-p

**[--wait-dbg]**

Monkey 停止执行，等待 Debugger 连接

**[--dbg-no-events]**

用于手动操作，Monkey启动app后，不会发送事件，让测试人员自己手动操作

**[--setup scriptfile] [-f scriptfile [-f scriptfile] ...]**

操作文件，事件不由Monkey产生， 而是按照传入的操作文件来操作要搭配-f 一起使用，保证该文件是在第一个执行的。

**[--port port]**

连接服务器进行Monkey操作，服务器上通过tcp或者adb生成事件, 可以参见monkey源码README.NETWORK.txt

**[-s SEED] [-v [-v] ...]**

后面紧跟种子数,种子数相同的情况下多次运行的结果都相同。种子数只是随机的初始值，和随机数的取值空间无关。-v 设置输出日志级别，默认日志级别为0，每增加一个-v，日志级别加1。超出三个的，按三个来计算。

**[--throttle MILLISEC]**

后面紧跟两个事件之间间隔的毫秒数，如果不设置，Monkey会以尽可能快的速度产生事件。

**[--randomize-throttle]**

随机生成事件之间的间隔时间(毫秒)，要和--throttle 搭配一起使用。生成的时间区间在[0, throttle]之间。

**[--profile-wait MILLISEC]**

后面紧跟事件之间的毫秒数，适用于用脚本跑Monkey的情况，默认值为5000毫秒

**[--device-sleep-time MILLISEC]**

设备空闲时间，适用于脚本跑的Monkey,默认是30000毫秒

**[--randomize-script]**

随机执行多个脚本文件，要与-f 参数一起使用

**[--script-log]**

TBD

**[--bugreport]**

当出现anr，crash时，生成bugreport(TBD)

**[--periodic-bugreport]**

紧跟次数，按照配置的几次次数，生成几次bugreport(TBD)

**[--permission-target-system]**

要配置--pct-permission参数一起使用

**COUNT**

事件的次数

## Monkey代码中参数的处理逻辑

本小节中将从代码方面对1.2中所提到的参数进行说明。Monkey代码路径：development/cmds/monkey。

解析参数的函数如下：

Monkey.java

private int run(String[] args) {

……

//参数：--wait-dbg，等待debugger的连接

for (String s : args) {

if ("--wait-dbg".equals(s)) {

Debug.waitForDebugger();

}

}

……

mArgs = args;

for (String a: args) {

//在monkey.log中打印了命令行的参数

Logger.err.println(" arg: \"" + a + "\"");

}

……

//解析其余参数

if (!processOptions()) {

return -1;

}

}

分析主要的参数设置

1. **[--ignore-crashes] [--ignore-timeouts]**

**[--ignore-security-exceptions]**

**[--monitor-native-crashes] [--ignore-native-crashes]**

} else if (opt.equals("--**ignore-crashes**")) {

mIgnoreCrashes = true;

} else if (opt.equals("--**ignore-timeouts**")) {

mIgnoreTimeouts = true;

} else if (opt.equals("--**ignore-security-exceptions**")) {

mIgnoreSecurityExceptions = true;

} else if (opt.equals("--**monitor-native-crashes**")) {

mMonitorNativeCrashes = true;

} else if (opt.equals("--**ignore-native-crashes**")) {

mIgnoreNativeCrashes = true;

流程中会对这些ignore前缀的全局变量做判断，如果没有设置为true，则会在monkey的run循环中停止，代码如下：

private int runMonkeyCycles() {

……

if (shouldAbort) {

shouldAbort = false;

Logger.out.println("\*\* Monkey aborted due to error.");

Logger.out.println("Events injected: " + eventCounter);

return eventCounter;

}

……

}

2. **[--pct-touch PERCENT] [--pct-motion PERCENT]**

**[--pct-trackball PERCENT] [--pct-syskeys PERCENT]**

**[--pct-nav PERCENT] [--pct-majornav PERCENT]**

**[--pct-appswitch PERCENT] [--pct-flip PERCENT]**

**[--pct-anyevent PERCENT] [--pct-pinchzoom PERCENT]**

**[--pct-permission PERCENT]**

这一类参数是设置事件的百分比，也是非常重要的参数，其影响了monkey测试的测试点覆盖。在测试命令中如果没有设置的话，monkey设置了默认值。

} else if (opt.equals("--pct-touch")) {

int i = MonkeySourceRandom.FACTOR\_TOUCH;

mFactors[i] = -nextOptionLong("touch events percentage");

} else if (opt.equals("--pct-motion")) {

int i = MonkeySourceRandom.FACTOR\_MOTION;

mFactors[i] = -nextOptionLong("motion events percentage");

} else if (opt.equals("--pct-trackball")) {

int i = MonkeySourceRandom.FACTOR\_TRACKBALL;

mFactors[i] = -nextOptionLong("trackball events percentage");

} else if (opt.equals("--pct-rotation")) {

int i = MonkeySourceRandom.FACTOR\_ROTATION;

mFactors[i] = -nextOptionLong("screen rotation events percentage");

} else if (opt.equals("--pct-syskeys")) {

int i = MonkeySourceRandom.FACTOR\_SYSOPS;

mFactors[i] = -nextOptionLong("system (key) operations percentage");

} else if (opt.equals("--pct-nav")) {

int i = MonkeySourceRandom.FACTOR\_NAV;

mFactors[i] = -nextOptionLong("nav events percentage");

} else if (opt.equals("--pct-majornav")) {

int i = MonkeySourceRandom.FACTOR\_MAJORNAV;

mFactors[i] = -nextOptionLong("major nav events percentage");

} else if (opt.equals("--pct-appswitch")) {

int i = MonkeySourceRandom.FACTOR\_APPSWITCH;

mFactors[i] = -nextOptionLong("app switch events percentage");

} else if (opt.equals("--pct-flip")) {

int i = MonkeySourceRandom.FACTOR\_FLIP;

mFactors[i] = -nextOptionLong("keyboard flip percentage");

} else if (opt.equals("--pct-anyevent")) {

int i = MonkeySourceRandom.FACTOR\_ANYTHING;

mFactors[i] = -nextOptionLong("any events percentage");

} else if (opt.equals("--pct-pinchzoom")) {

int i = MonkeySourceRandom.FACTOR\_PINCHZOOM;

mFactors[i] = -nextOptionLong("pinch zoom events percentage");

} else if (opt.equals("--pct-permission")) {

int i = MonkeySourceRandom.FACTOR\_PERMISSION;

mFactors[i] = -nextOptionLong("runtime permission toggle events percentage");

从上述的代码可以看到，事件的百分比值都保存在数组mFactors中，且以值为负数保存。同时在MonkeySourceRandom的构造函数中对于事件百分比也有默认赋值：

mFactors[FACTOR\_TOUCH] = 15.0f;

mFactors[FACTOR\_MOTION] = 10.0f;

mFactors[FACTOR\_TRACKBALL] = 15.0f;

// Adjust the values if we want to enable rotation by default.

mFactors[FACTOR\_ROTATION] = 0.0f;

mFactors[FACTOR\_NAV] = 25.0f;

mFactors[FACTOR\_MAJORNAV] = 15.0f;

mFactors[FACTOR\_SYSOPS] = 2.0f;

mFactors[FACTOR\_APPSWITCH] = 2.0f;

mFactors[FACTOR\_FLIP] = 1.0f;

// disbale permission by default

mFactors[FACTOR\_PERMISSION] = 0.0f;

mFactors[FACTOR\_ANYTHING] = 13.0f;

mFactors[FACTOR\_PINCHZOOM] = 2.0f;

如果在monkey执行命令行中设置了百分比，则会重新设置。

for (int i = 0; i < MonkeySourceRandom.FACTORZ\_COUNT; i++) {

if (mFactors[i] <= 0.0f) {

((MonkeySourceRandom) mEventSource).setFactors(i, mFactors[i]);

}

}

同时在循环中，会判断设置的百分比是否有效，判断函数如下：

MonkeySourceRandom.java

private boolean adjustEventFactors() {

//百分比分为两类：命令行设置及默认设置

for (int i = 0; i < FACTORZ\_COUNT; ++i) {

if (mFactors[i] <= 0.0f) { // user values are zero or negative

userSum -= mFactors[i];

} else {

defaultSum += mFactors[i];

++defaultCount;

}

}

……

// compute the adjustment necessary

//从下面的代码逻辑可以看到事件的百分比是先优先user，也即通过命令行设置的

//百分比，然后剩下的百分比再按照默认百分比的总值进行计算

float defaultsTarget = (100.0f - userSum);

float defaultsAdjustment = defaultsTarget / defaultSum;

// fix all values, by adjusting defaults, or flipping user values back to >0

for (int i = 0; i < FACTORZ\_COUNT; ++i) {

if (mFactors[i] <= 0.0f) { // user values are zero or negative

mFactors[i] = -mFactors[i];

} else {

mFactors[i] \*= defaultsAdjustment;

}

}

// if verbose, show factors

if (mVerbose > 0) {

Logger.out.println("// Event percentages:");

for (int i = 0; i < FACTORZ\_COUNT; ++i) {

Logger.out.println("// " + i + ": " + mFactors[i] + "%");

}

}

……

}

不同事件百分比生成事件的函数如下：

private void generateEvents() {

//从(0.0,1.0)之间生成随机数

float cls = mRandom.nextFloat();

……

if (cls < mFactors[FACTOR\_TOUCH]) {

generatePointerEvent(mRandom, GESTURE\_TAP);

return;

} else if (cls < mFactors[FACTOR\_MOTION]) {

generatePointerEvent(mRandom, GESTURE\_DRAG);

return;

} else if (cls < mFactors[FACTOR\_PINCHZOOM]) {

generatePointerEvent(mRandom, GESTURE\_PINCH\_OR\_ZOOM);

return;

} else if (cls < mFactors[FACTOR\_TRACKBALL]) {

generateTrackballEvent(mRandom);

return;

} else if (cls < mFactors[FACTOR\_ROTATION]) {

generateRotationEvent(mRandom);

return;

} else if (cls < mFactors[FACTOR\_PERMISSION]) {

mQ.add(mPermissionUtil.generateRandomPermissionEvent(mRandom));

return;

}

}

**TOUCH,MOTION,PINCHZOOM**三种触屏事件的生成函数如下：

generatePointerEvent(……){

……

//初始时key down

mQ.addLast(new MonkeyTouchEvent(MotionEvent.ACTION\_DOWN)

.setDownTime(downAt)

.addPointer(0, p1.x, p1.y)

.setIntermediateNote(false));

//如果是monitor事件，随机生成10以内的MOVE事件

int count = random.nextInt(10);

for (int i = 0; i < count; i++) {

randomWalk(random, display, p1, v1);

mQ.addLast(new MonkeyTouchEvent(MotionEvent.ACTION\_MOVE)

.setDownTime(downAt)

.addPointer(0, p1.x, p1.y)

.setIntermediateNote(true));

//如果是屏幕放大缩小事件，同样是随机生成10以内的放大缩小事件，包括

// ACTION\_POINTER\_DOWN，ACTION\_MOVE和ACTION\_POINTER\_UP

//最后是key up

randomWalk(random, display, p1, v1);

mQ.addLast(new MonkeyTouchEvent(MotionEvent.ACTION\_UP)

.setDownTime(downAt)

.addPointer(0, p1.x, p1.y)

.setIntermediateNote(false));

}

**TRACKBALL**的事件生成函数如下：

generateTrackballEvent(Random random){

//10次轨迹球的move事件

for (int i = 0; i < 10; ++i) {

// generate a small random step

int dX = random.nextInt(10) - 5;

int dY = random.nextInt(10) - 5;

mQ.addLast(new MonkeyTrackballEvent(MotionEvent.ACTION\_MOVE)

.addPointer(0, dX, dY)

.setIntermediateNote(i > 0));

}

//10%的概率，move之后执行click操作，也即ACTION\_DOWN和ACTION\_UP

}

**ROTATION** 事件的生成函数如下：

generateRotationEvent(……) {

//从SCREEN\_ROTATION\_DEGREES数组中随机抽取值进行翻转，包括

//0，90，180，270，插入事件则是通过WindowManagerservice的thawRotation

//来自动旋转

}

**PERMISSION**事件的生成函数如下：

generateRandomPermissionEvent(…..){

//从目标package数组中随机找到pkg，然后从pkg对应的权限列表随机抽取

//一项权限，插入事件则是通过packagemanagerservic的//grantRuntimePermission和revokeRuntimePermission来注册和取消权限

}

上述的事件基本上都是触屏事件，接下来分析key 事件的生成和处理函数。

NAV NAV\_KEYS 上下左右方向键中随机抽取

MAJOR\_NAV MAJOR\_NAV\_KEYS 菜单键和确认键随机抽取

SYSOPS SYS\_KEYS home，back，拨号键，挂断键，音量上下键，话筒静音键和扬声器静音键

处理函数包括按下及放开两步

MonkeyKeyEvent e = new MonkeyKeyEvent(KeyEvent.ACTION\_DOWN, lastKey);

mQ.addLast(e);

e = new MonkeyKeyEvent(KeyEvent.ACTION\_UP, lastKey);

mQ.addLast(e);

另外，FLIP是通过直接往dev/input/event0中写入原始的键盘弹出数据来模拟键盘的弹出和收起。

还有第三类主要的事件处理，那就是APPSWITCH。从app列表中抽取app，通过ams的startActivity来开启对应应用的activity。

MonkeyActivityEvent e = new MonkeyActivityEvent(mMainApps.get(

mRandom.nextInt(mMainApps.size())));

3. **[--pkg-blacklist-file PACKAGE\_BLACKLIST\_FILE]**

**[--pkg-whitelist-file PACKAGE\_WHITELIST\_FILE]**

} else if (opt.equals("--pkg-blacklist-file")) {

mPkgBlacklistFile = nextOptionData();

} else if (opt.equals("--pkg-whitelist-file")) {

mPkgWhitelistFile = nextOptionData();

从代码里可以看到黑白名单的文件路径通过cmd解析后保存在mPkgBlacklistFile和mPkgWhitelistFile中。

黑名单的作用是不测试文件中包含的app包，实现方法则是解析出来包名后，不会添加进入mMainApps，而startactivity则是从mMainApps中随机抽取。

Set<String> invalidPackages = new HashSet<>();

if ((mPkgBlacklistFile != null)

&& (!loadPackageListFromFile(mPkgBlacklistFile, invalidPackages))) {

return false;

}

MonkeyUtils.getPackageFilter().addInvalidPackages(invalidPackages);

白名单的作用则是只测试文件中包含的app包，实现则是通过判断函数checkEnteringPackage来实现，如果是白名单中的包名才会加入mMainApps，其他的包则不会加入mMainApps，当然就不会被随机抽取。

注意：黑名单不能和白名单或者-p pkg同时存在。

private boolean loadPackageLists() {

if (((mPkgWhitelistFile != null) || (MonkeyUtils.getPackageFilter().hasValidPackages()))

&& (mPkgBlacklistFile != null)) {

Logger.err.println("\*\* Error: you can not specify a package blacklist "

+ "together with a whitelist or individual packages (via -p).");

return false;

}

本小节从代码层面分析了monkey测试中主要参数的代码逻辑，其他剩下的参数在monkey测试中较少使用，此处不再做详细的说明。

## 1.4 测试部门使用的monekey 参数

Monkey测试：事件百分比因为没有设置，所以用的是默认百分比

monkey -s 48 --throttle 200 --ignore-crashes --ignore-timeouts --ignore-security-exceptions --pkg-blacklist-file $external\_log\_dir/blacklist --kill-process-after-error -v -v -v 20000000 1>>$external\_log\_dir/monkey.log 2>>$external\_log\_dir/monkey.log &

monkey+sleep测试：事件百分比已设置，且总和为100%，所以只测试这三类事件

monkey --pct-syskeys 20 --pct-appswitch 55 --pct-permission 25 --throttle 1000 --pkg-blacklist-file $external\_log\_dir/blacklist --ignore-crashes --ignore-timeouts --ignore-security-exceptions --kill-process-after-error 600 >> $monkeyLog

monkey+reboot测试：事件百分比设置总和为100%，所以也只测试这两类事件。

monkey --pct-syskeys 20 --pct-appswitch 80 --throttle 1500 1500 --ignore-crashes --ignore-timeout &

# monkey中时间间隔对于系统测试的影响

上一章节分析了monkey测试中的主要参数，其中参数throttle是每一个事件的间隔时间。而Monkey测试目的就是为了能有效的测试随机事件，那每个事件的时间间隔为多少才能使得monkey测试中的随机事件有效比例更高呢。下面以200ms为例作分析：

第一类：

Sleeping for 200 milliseconds

:Switch: #Intent;action=android.intent.action.MAIN;category=android.intent.category.LAUNCHER;launchFlags=0x10200000;component=com.android.fmradio/.FmMainActivity;end

// Allowing start of Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.android.fmradio/.FmMainActivity } in package com.android.fmradio

Sleeping for 200 milliseconds

:Sending Touch (ACTION\_DOWN): 0:(40.0,572.0)

// Injection Failed

:Sending Touch (ACTION\_UP): 0:(41.68047,562.9074)

// Injection Failed

Sleeping for 200 milliseconds

:Sending Touch (ACTION\_DOWN): 0:(121.0,718.0)

// Injection Failed

:Sending Touch (ACTION\_POINTER\_DOWN 1): 0:(130.08649,716.1339) 1:(296.0,550.0)

// Injection Failed

:Sending Touch (ACTION\_MOVE): 0:(133.41068,708.89746) 1:(304.7836,529.1349)

// Injection Failed

:Sending Touch (ACTION\_MOVE): 0:(139.8863,703.6457) 1:(324.0658,517.9852)

// Injection Failed

:Sending Touch (ACTION\_POINTER\_UP 1): 0:(153.60757,703.08044) 1:(324.58847,505.59927)

// Injection Failed

:Sending Touch (ACTION\_UP): 0:(161.51196,695.41595)

// Injection Failed

Sleeping for 200 milliseconds

从上面截取的monkey log可以看到，当随机事件为startactivity FmMainActivity后，紧随着200ms的key event inject失败，原因可以在android log中找到：

01-01 08:28:26.308 1287 1506 I InputDispatcher: Dropping event because there is no touchable window at (40, 572).

activity还在创建过程中，此时无法接收触屏事件，会被input直接drop，表现在monkey log中就是inject fail。所以startactivity的持续时间，会影响inject event的有效性。

第二类：

:Sending Key (ACTION\_DOWN): 23 // KEYCODE\_DPAD\_CENTER

// Injection Failed

:Sending Key (ACTION\_UP): 23 // KEYCODE\_DPAD\_CENTER

// Allowing start of Intent { cmp=com.android.music/.RequestPermissionsActivity } in package com.android.music

// activityResuming(com.android.music)

// Rejecting start of Intent { act=android.content.pm.action.REQUEST\_PERMISSIONS pkg=com.google.android.packageinstaller cmp=com.google.android.packageinstaller/com.android.packageinstaller.permission.ui.GrantPermissionsActivity } in package com.google.android.packageinstaller

Fail原因如下：

01-01 08:29:27.805 1287 1506 I InputDispatcher: Dropped event because of pending overdue app switch.

在monkey测试中，此类问题相比第一类较少。应该是在打开应用的权限界面时，需要使用google的packageinstaller 功能，但com.google.android.packageinstaller被加入黑名单，无法打开，导致按键dispatch超时，event drop。

结合上述的两类情况，统计了下monkey log中执行事件数与inject failed数的比值，GMS版本并将GMS应用添加了黑名单的情况下：

Monkey event ：19500次

Injection failed：3399次，其中第二类占47次。

排除gms加黑名单的影响，也即排除第二类failed的情况，失败的百分比大概为：

3352/19500 = 17%。

从上述的分析可以看到，monkey测试中startactivity的时间会影响事件的有效比例，系统负载越小，startactivity时间越少，影响的fail 事件也会越少。所以，在内存较小或者应用进程较多的情况下，200ms的时间间隔并不是最优的选择。

本地实验：使用native版本验证，时间间隔500ms的monkey脚本，失败百分比大概为11%左右，但使用500ms间隔，同样的测试周期事件总数相对较少。

**总结**：

在native版本的monkey测试中，同样的内存条件下，200ms和500ms的差异不明显。但从log分析来看，如果是GMS版本，且GMS应用未加黑名单的情况下，系统负载会较大，进而导致startactivity的时间增加。在此种情况下，500ms的时间间隔相比200ms会明显减少被drop的事件.

**版本历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 作者 | 备注 |
| Template V1.0 | 2018.08.14 | Zhihui.liao | draft |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |