Android APP调试技术

|  |  |  |
| --- | --- | --- |
| 版本号 | 描述 | 日期(更新日期)/更新人 |
| 1.0 | Android APP调试手段 | 2018/07/10 刘志保 |
|  |  |  |

写这份文档的目的:

<1> :程序普通的断点调试,减少没必要的日志调试方式,效率不高(每打印一个日志就要重新安装,调试完成后还需要删除没必要的日志信息);

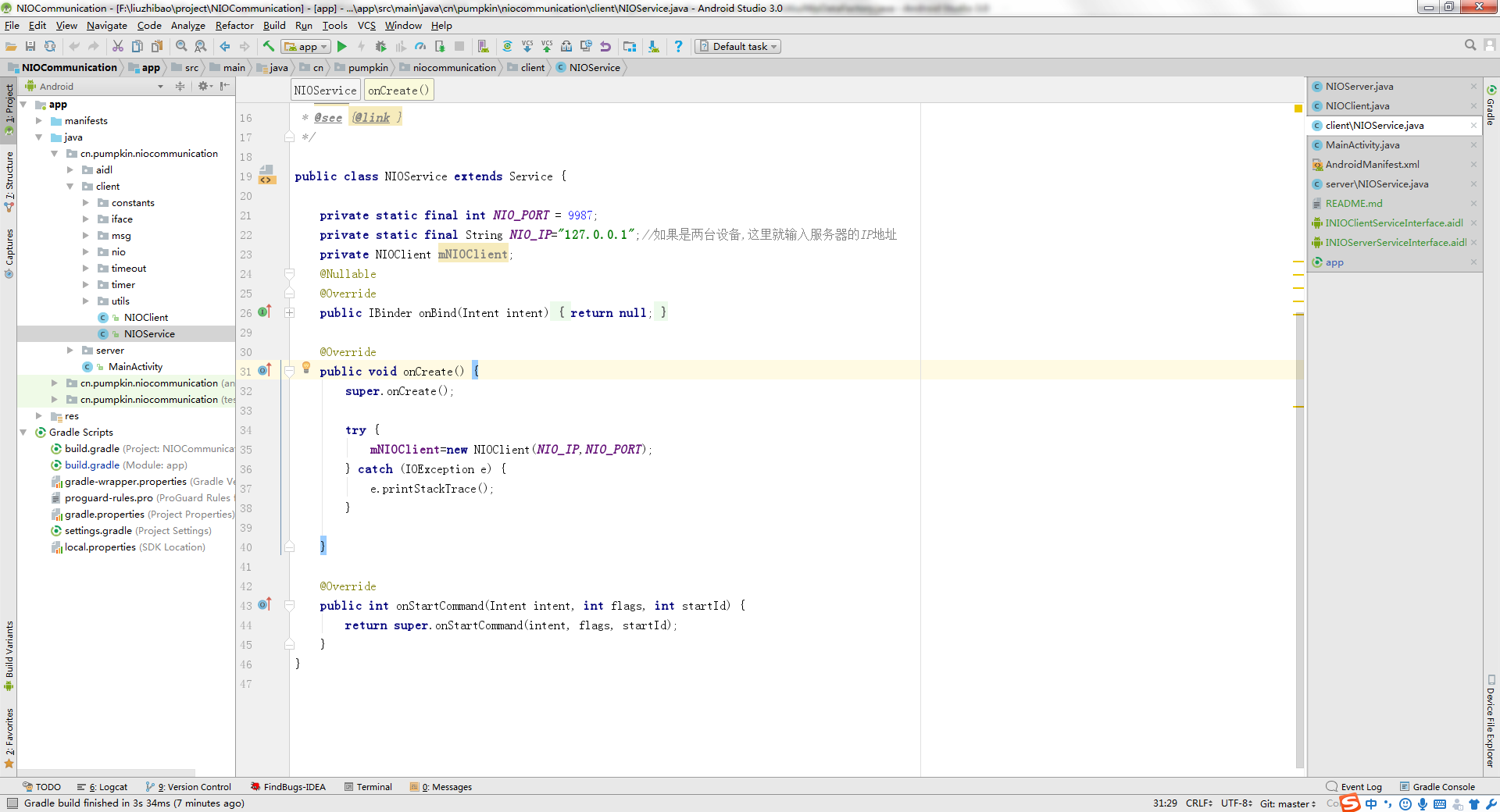
<2> :程序跨线程调试,查看线程运行状态,以及异常死锁状态;

<3> :程序跨进程调试,当一个程序有两个或者多个进程时,并且它们之间有交互过程,需要随时查看它们运行状态,可以进行进程调试;

注意:

下面的快捷键是用的是eclipse keymap,默认的android studio的调试快捷键不一样.

<1> 调试相关菜单介绍:



1.

前面的都是打开文件,保存所有文件,刷新等等,无需再解释了.

2.锤子 : make project



3.三角 : run(全速运行)



4.小三角 : debug app(调试运行)



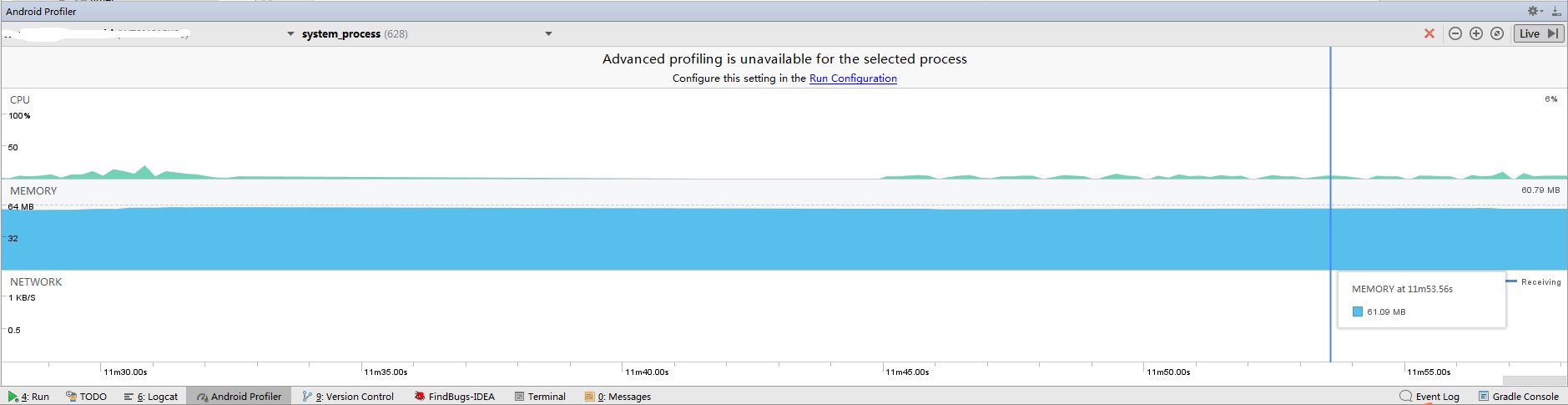
注意 : 这个什么时候使用呢?

使用这个首先要先将断点设置好,而且这个断点一般设置在APP初始化过程中的那些程序里面,比如一些程序启动要读取配置文件,要读取数据库等待,不清楚读取的数据或者状态是否正确,即设置断点,然后使用debug APP按钮进行程序.debug app这个调试功能只能够断点一次(每次运行起来以后)

5.仪表 : profile(查看制定进程的CPU,MEMORY,NETWORK使用情况)



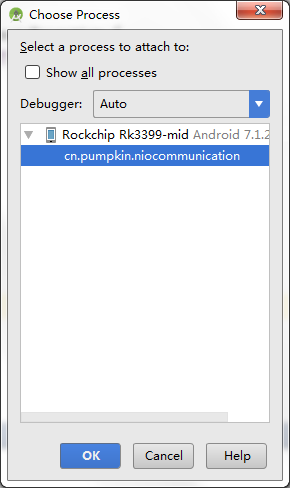
点击,并且选择usb设备,然后稍等片刻:



这个是性能优化参考的,和调试作用不大.

6.attached debugger to process : 对制定的进程进行debug调试,这个是调试用的最多的工具.



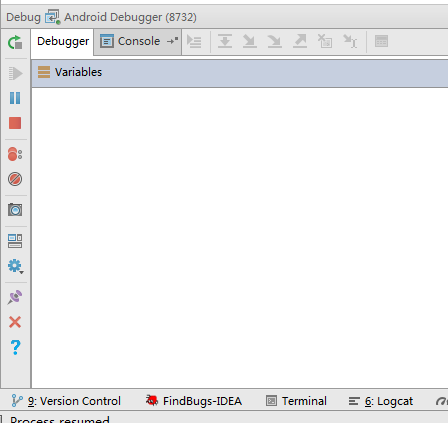


APP在运行起来后,如果突然/随时想看程序运行的位置和状态,可以随时把断点加上,然后让程序触发运行到断点处,就可以看到程序的值和状态信息.这个是调试最常用的,非常重要.

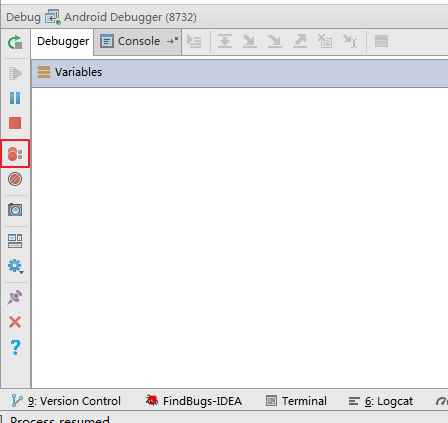
7.后面的基本上和调试没什么关系



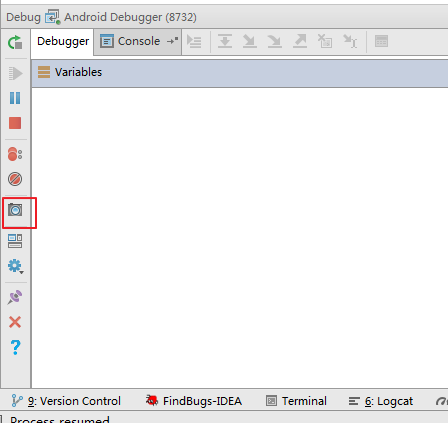
8.调试过程中的菜单:



其中下图红色框中(View breakpoint),这个平时很有用,比如,平时断点打多了,自己不想到处去找断点在哪里,即点击这个菜单,即可以知道哪些地方添加了断点,也可以取消已增加的断点的地方,但是要看到这些菜单,必须是正在调试过程中.



另外还有一个非常重要的调试:



如上图,如果一个APP有很多线程在运作,如果出现线程死锁,而且是概率性的,那么这样的问题就非常麻烦了,而且很容易找到查找的方向,添加断点,打印日志都很难找出原因,比如发现APP没有死(查看进程还在),但是又没有动作的迹象,好像程序”被绑架”了.这种由于线程间导致互掐的问题是不可能通过断点和打印日志出来的,这个时候就需要这个menu,如下:



9. 当程序处于断点调试状态,如何step into/step out/run to next breakpoint,下方图红框内那几个就是断点运行的步伐种类.这个地方不需要太多,个人感觉只需要记住快捷键(下面是在android studio里面设置成eclipse keymap了)

F5 = step into(进入方法体,或者继续跳到下一步)

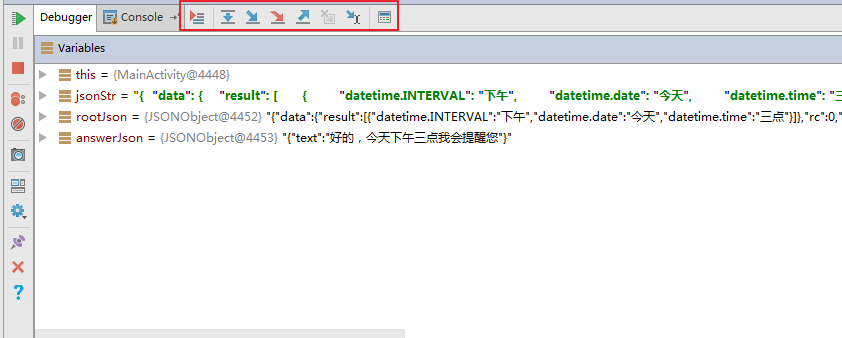
F6 = step over(忽略方法体和行句,均按照一步运行)

F7 = step out(跳出当前方法体)

F8 = run to next breakpoint()(跳转到下一个方法体)

如果是android studio keymap,依次对应F7,F8,F9.

下图左上角那个箭头是继续运行(这个继续运行会忽略断点的,在断点处不会停下来了,类似全速运行)



<2> : 上面后面继续补充,这里开发一个实例来补充上面的调试过程.新建一个AngryPandaDebugger工程:

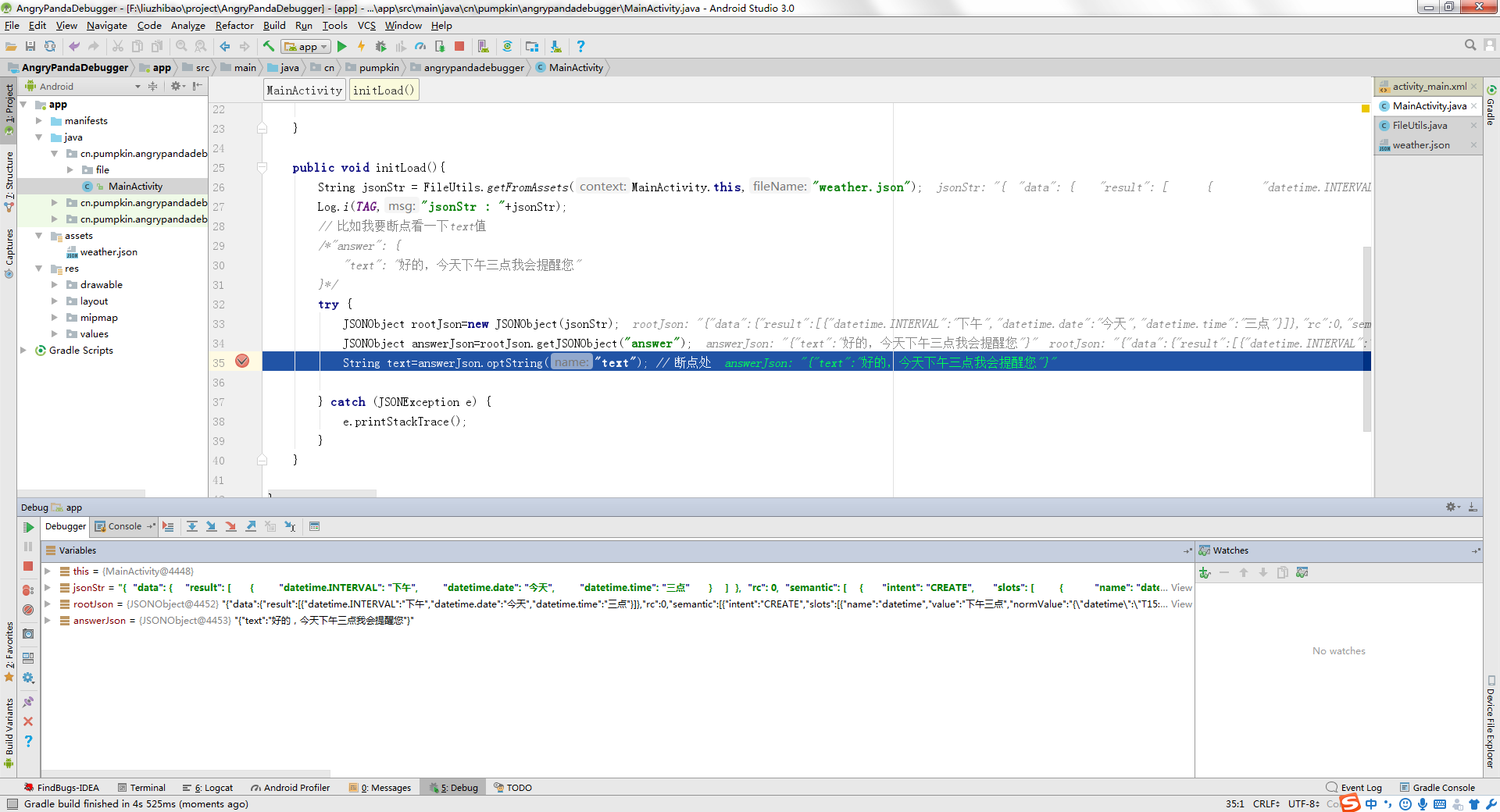
Debug ‘App’ 断点调试

下面测试<1>中的第4点,前面三点和调试关系不大,忽略.

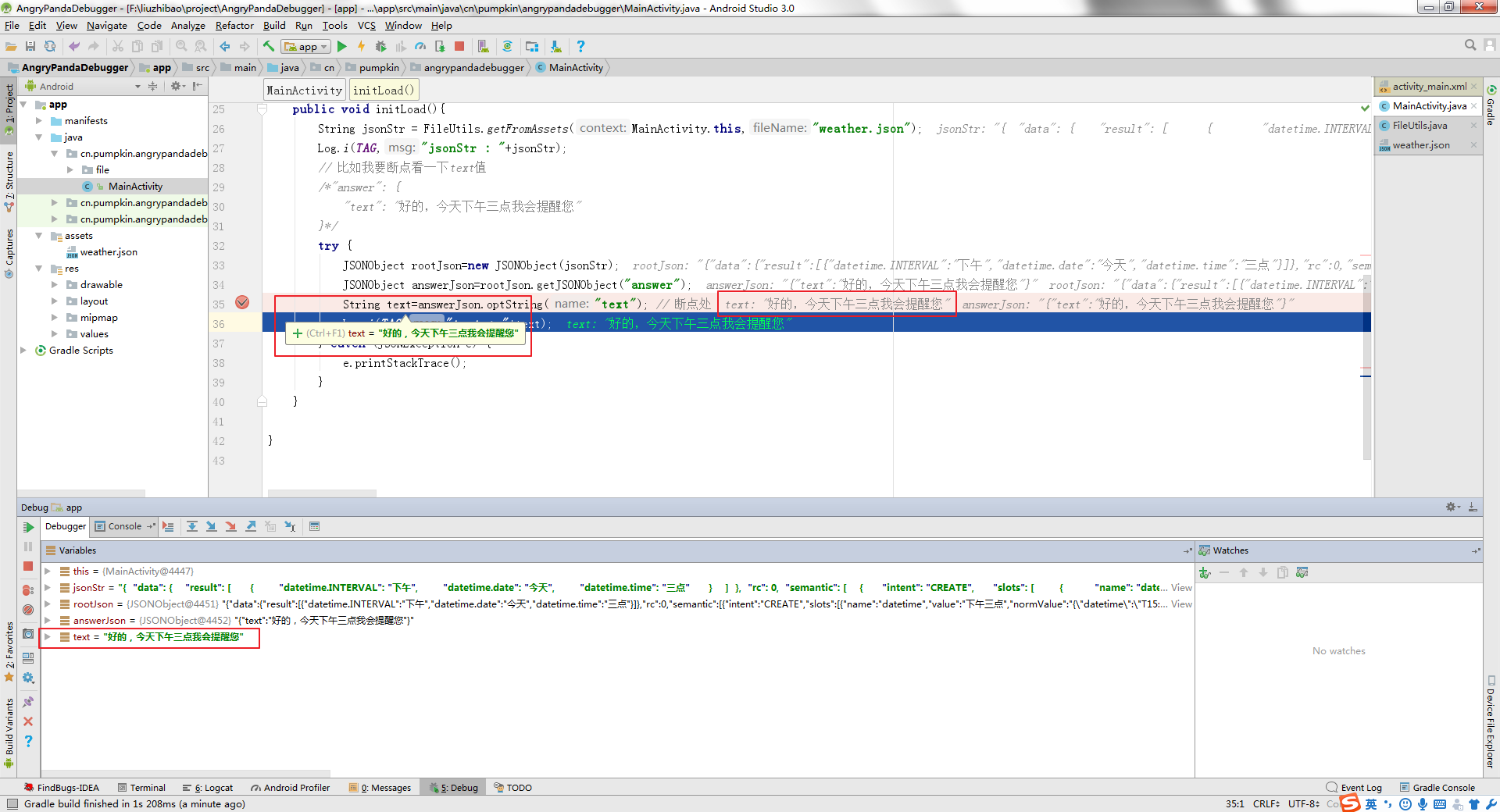
程序如下 :



首先如图先设置断点,这段程序在onCreate里面就开始运行了,所以不能使用第6点的断点方式查看.断点运行起来:



蓝色背景的那一行就是程序运行停留的地方,这个时候就可以看到编辑界面或者下方都有变量的值,但是会发现text值没有显示,这是因为程序还没有运行断点那一行,这个时候只需要按F6快捷键,即可.



三个地方可以看到,可以将鼠标光标放到text上面,就会弹出一个黄色背景的弹出框显示text的值,即如下:



左边”+”号可以点开,这个在看数组,集合,复杂结构体的时候非常有用.

绑定已运行的进程进行debug调试

下面看看<1>中的第6如何操作:

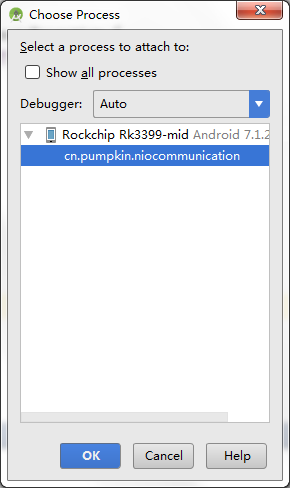
修改一下程序:增加一个按钮.



先把程序安装到设备里面(先要安装哦),然后按照第6的步骤进行操作:

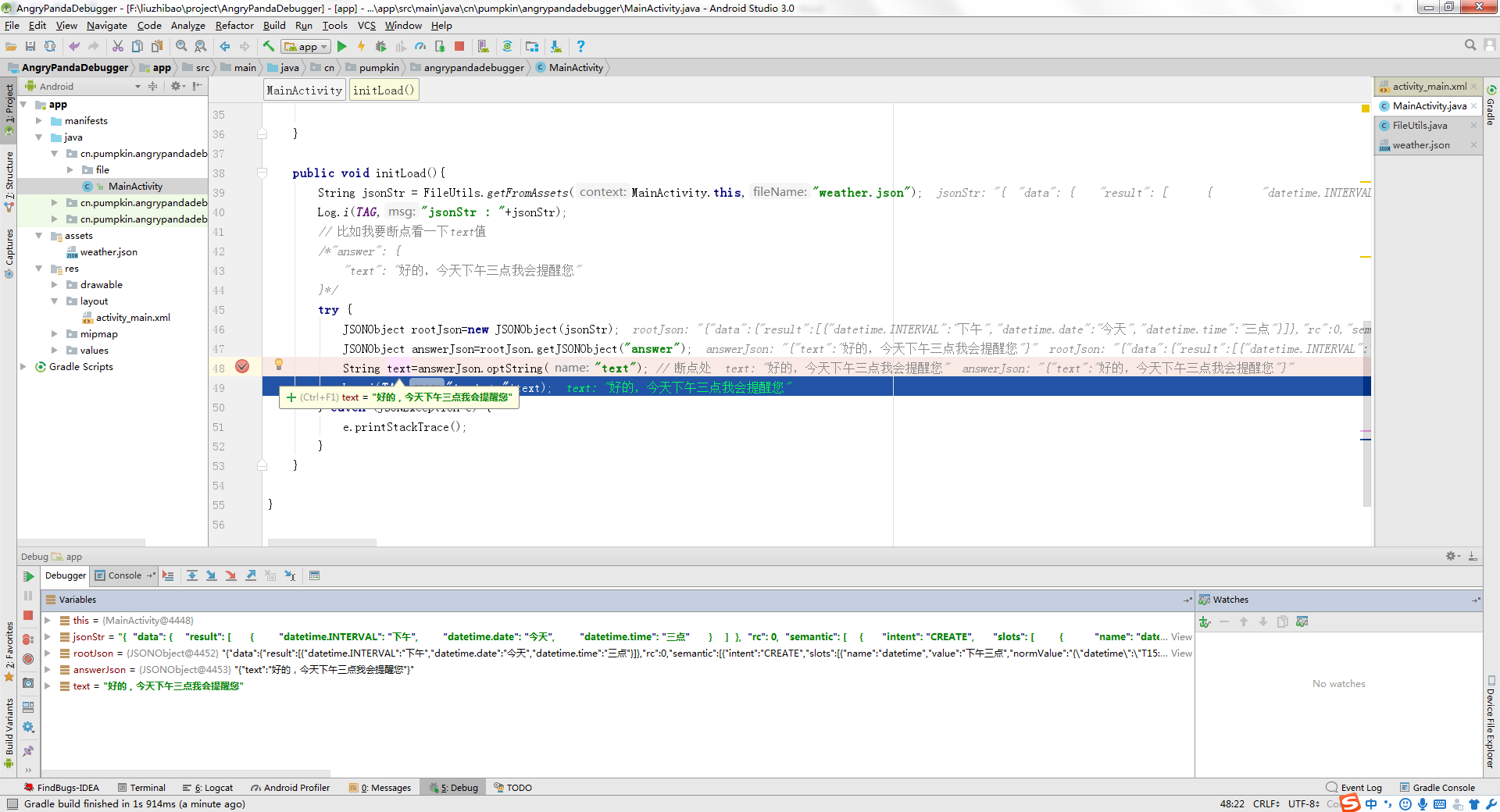
安装后,然后设置断点,然后再点击按钮触发.

先点击第6中比较的菜单,然后弹出:



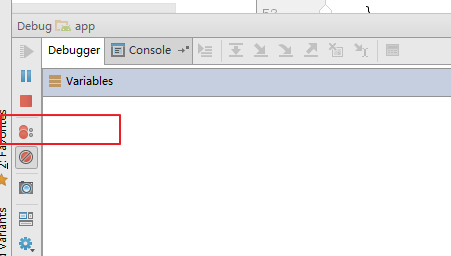
点击”OK”按钮.

然后设置断点,在点击APP界面按钮即可触发.

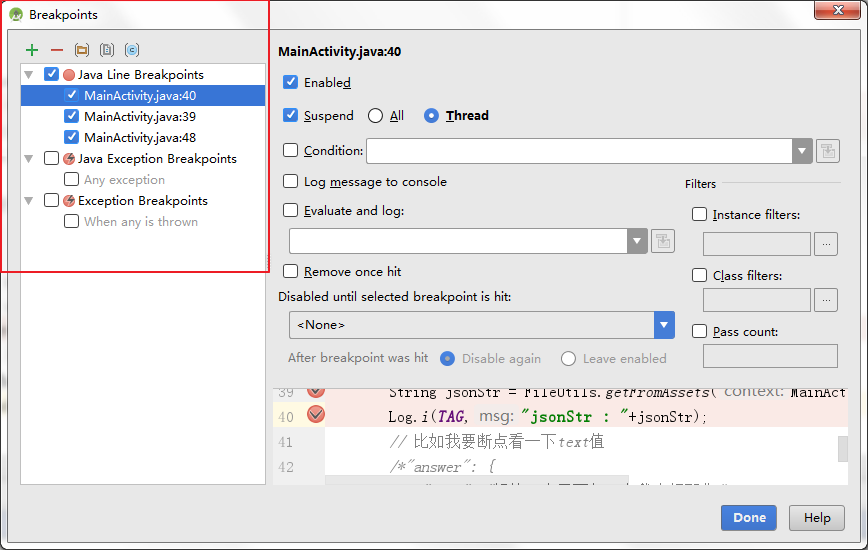


查看debug断点位置信息

然后我们再来看<1>第8中第一个介绍的菜单,在上面的基础上再增加几个断点,然后点击



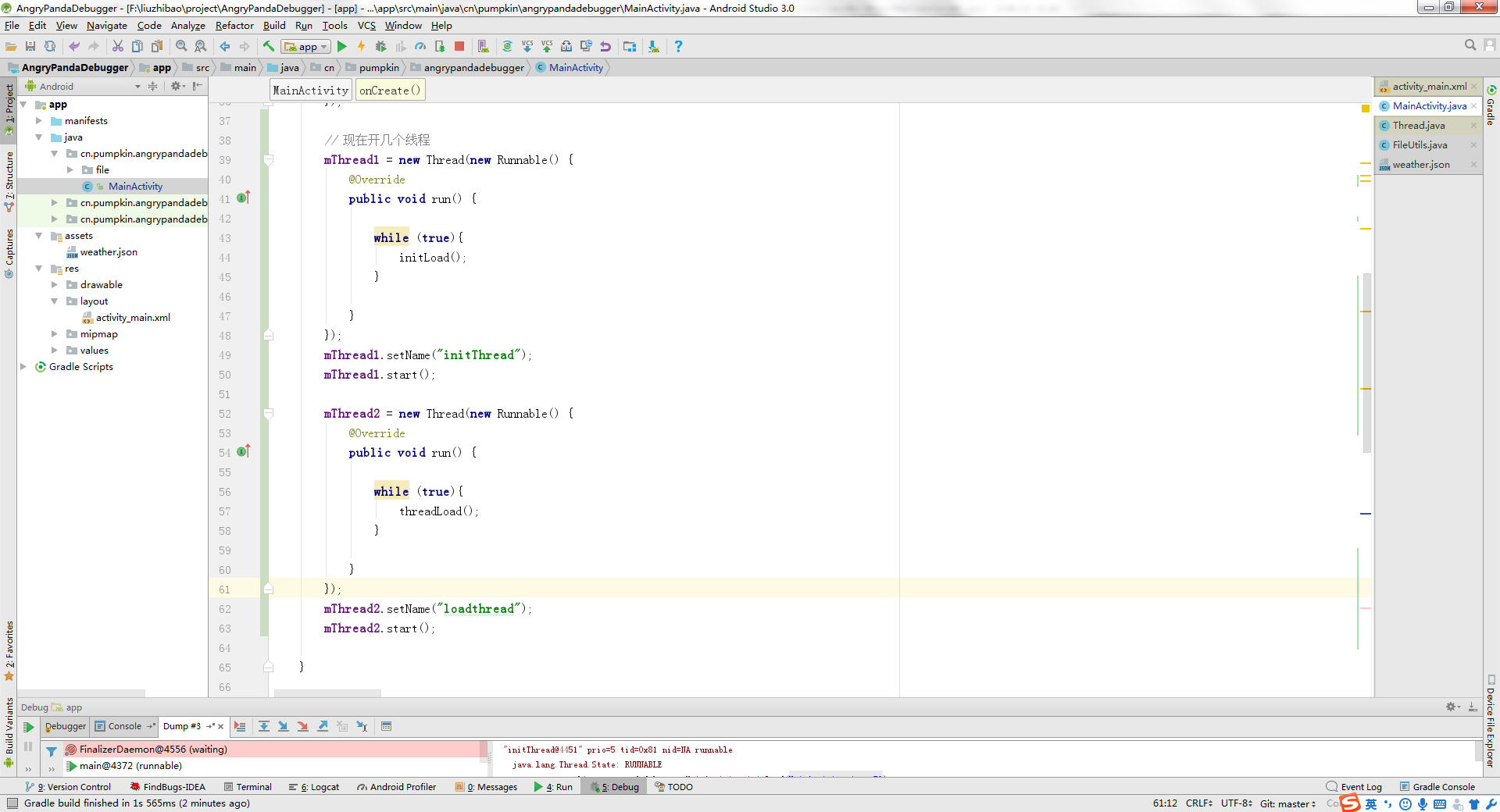
点击后:弹出对话框,一幕了然.



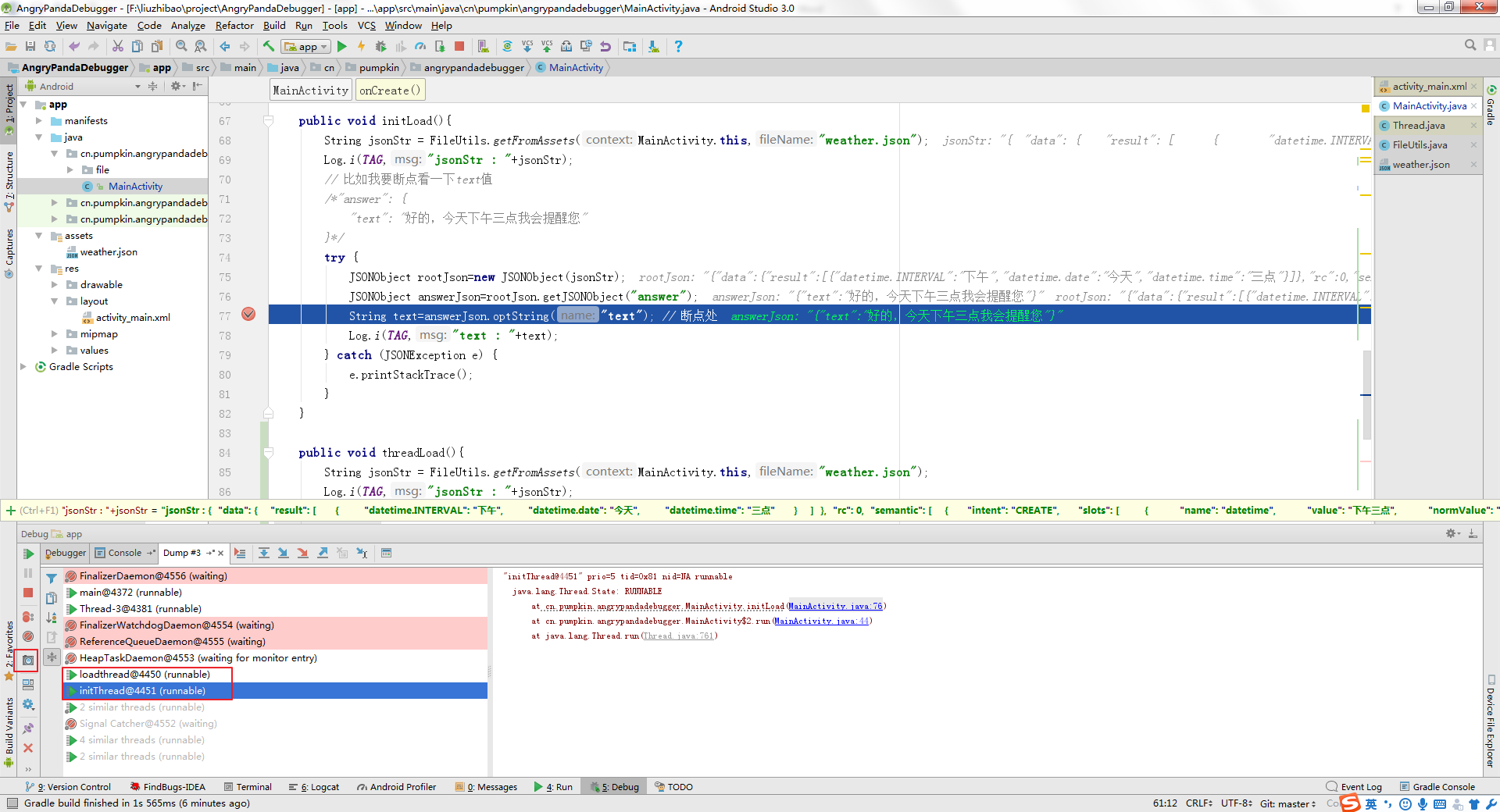
线程间debug调试

下面来看看<1>第8点”相机” 菜单的使用方法,即线程查看调试,这个相对复杂一些.

增加两个线程到程序里面:一个线程叫做”initthread”,一个叫做”loadthread”



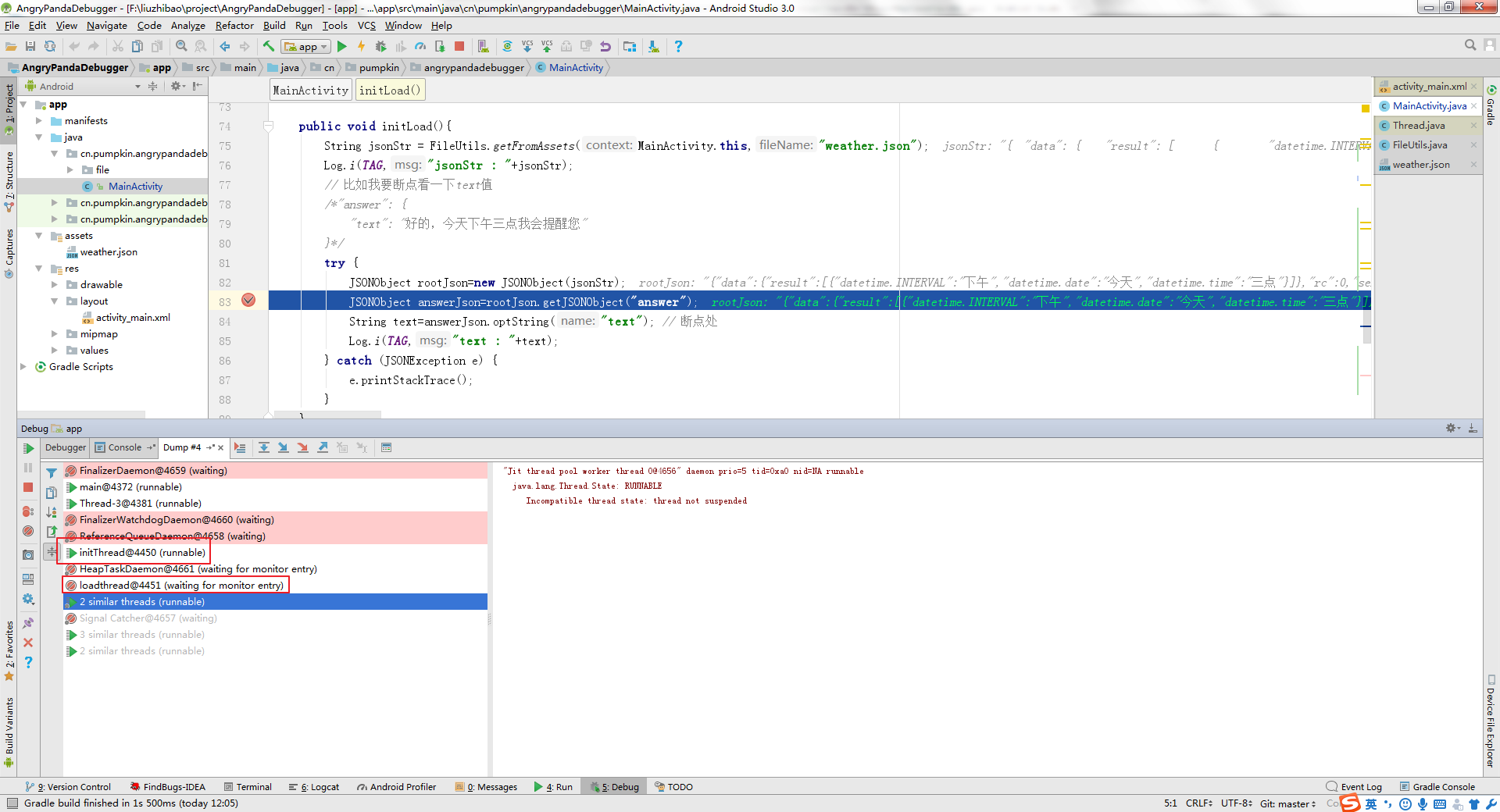
点击菜单运行:



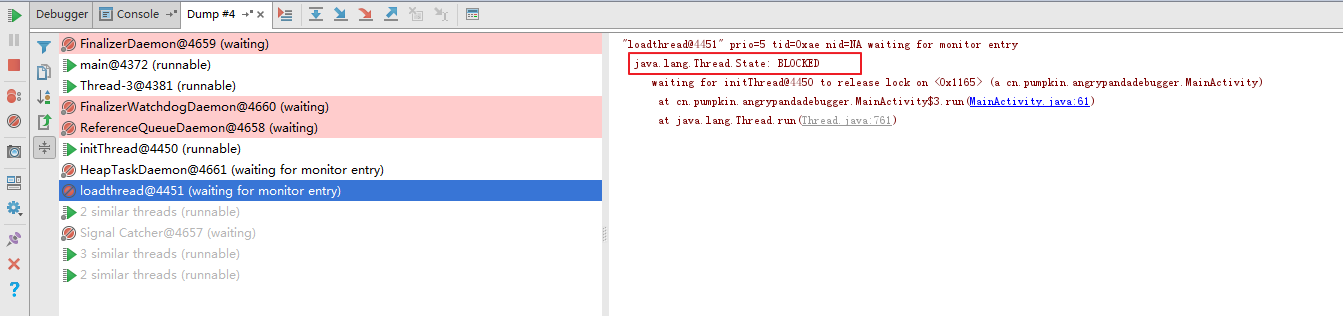
按快捷键F8,会发现程序基本上会在initLoad和threadLoad两个方法中来回切换,正常!现在我再加把锁,如下:



当然实际中的程序肯定互相锁住不是很容易发现,这里只研究调试技术,忽略程序!,看看线程情况:



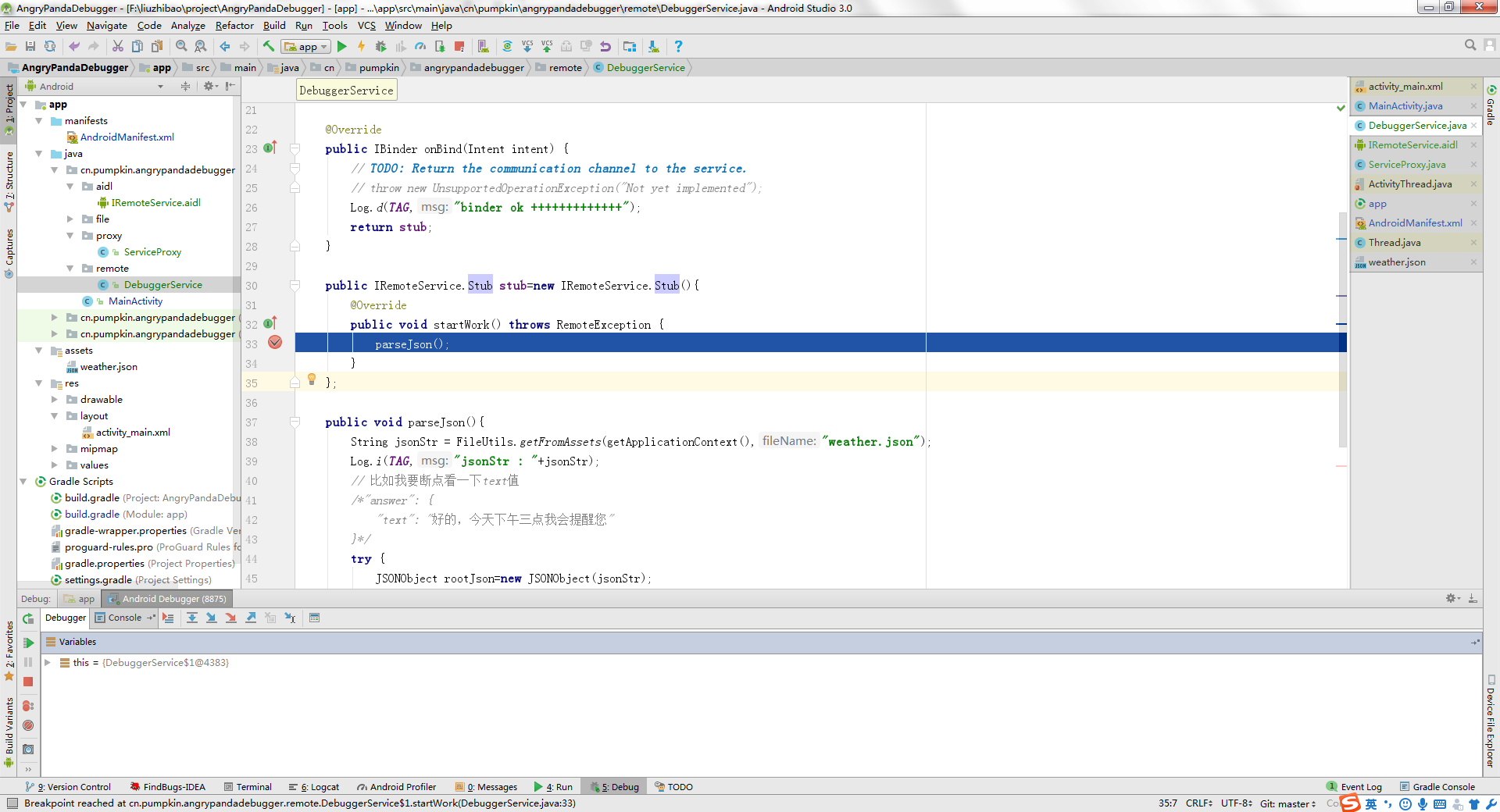
InitThread线程处于(runnable)过程中,即获得锁,可以正常运行的线程,但是loadThread线程处于(waiting for monitor entry),一直处于等待,程序一直得不到运行.点击loadThread线程那一行,右边就有这个线程的说明,显示”BLOCKED”.



当你的程序有可能处于”假死”状态时,而且明知自己的程序线程多的时候,一定要看看线程是否死锁.

进程间debug断点调试

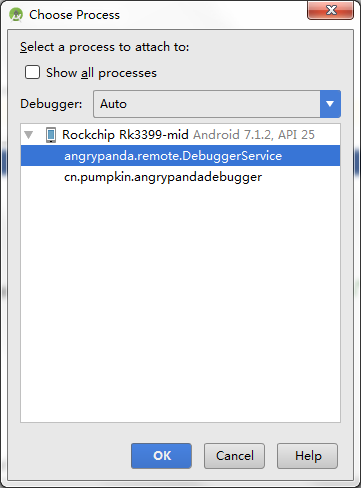
然后继续<1>第8点,看看进程间是如何设置断点调试的, 增加一个Service,并且设置为远程服务,使用aidl进行调用.添加的程序有点多,就不一一罗列.大致如下:



这个地方事先说一下,网上推荐的那些方法基本上都是不可靠的.

正确的操作步骤如下:

1. : 点击debug运行;
2. : 运行起来后, 点击,弹出框中选择:



点击OK

C> : 设置断点后,等待触发,其他查看方法与前面无异.

