

Android 程序调试

- [Android 下如何调试程序?](#) 
- [Android DDMS 如何使用?](#) 

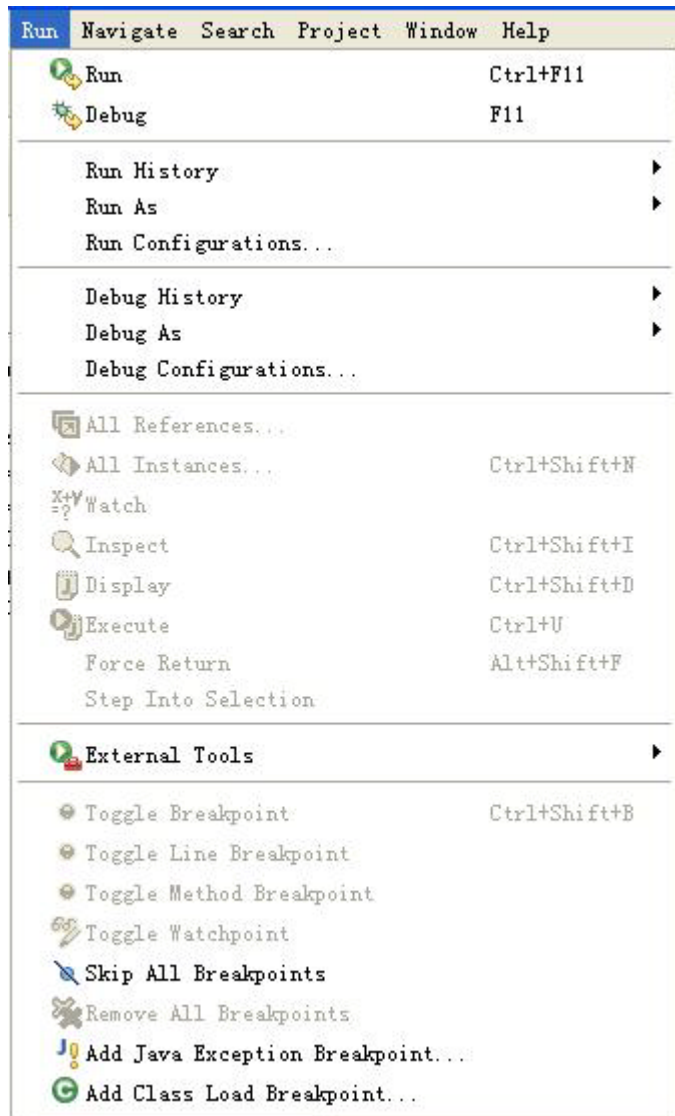
Android 下如何调试程序?

写代码是每个程序员最乐意做的事,然而在开发中也会遇到很多令程序员很头疼的事情。如果说让程序员最头疼的事情是看到无数 bug、软件的发布遥遥无期,那么让程序员最最头疼的事情是程序在调试状态下没有问题而在实际运行中确有问题。调试程序是每个程序员工作中必不可少的部分,而且可以毫不夸张地说 调试程序暂用了程序员 50%的工作时间。由此可见,调试程序是每个程序员必不可少的技术,调试水平的高低决定了程序员水平的高低。在开发 Android 程序前,有必要总结下如何调试 Android 程序。目前就开发过程中,常用调试程序的方法总结如下:

1. 使用 Eclipse 开发平台调试;
2. 结合 Android SDK 调试;
3. 使用 JUnit 调试;

使用 Eclipse 开发平台调试

这是使用 Eclipse 工具开发 Android 必须熟练掌握的调试技术,主要包括:设置断点、查看变量值、查看当前堆栈等。打开 Eclipse 工具,单击“Run”



以及在调试的过程中，打开其他调试面板，相信只要使用一次就完全明白了。不要小瞧这些调试工具，只要你细心，说不定其他同事好几天没有解决的 bug，你通过这些工具就发现了。所以熟练使用这些工具，是开发人员必须的，在有些时候甚至可以事半功倍的效果。

结合 Android SDK 调试

在复杂的程序运行过程中，如何调试程序了？把程序运行过程的信息保存为文件或者输出到 IDE 中，这样就可以知道程序是否是正常运行了。

在 Android 中可以使用 Log 类，Log 类在 android.util 包中，可以使用它将运行过程的信息输出到 IDE 中，直接查看程序运行的过程。Log 类提供了若干静态方法：

```
Log.v(String tag, String msg);  
Log.d(String tag, String msg);  
Log.i(String tag, String msg);  
Log.w(String tag, String msg);  
Log.e(String tag, String msg);
```

分别对应 Verbose, Debug, Info, Warning, Error。tag 是一个标识, 可以是任意字符串, 通常可以使用类名+方法名, 主要是用来在查看日志时提供一个筛选条件。程序运行后, 在 show view 中选择 Logcat 就可以直接看到输出了。也可以在程序运行后, 可以通过 DDMS 查看程序的运行过程记录, 并可以通过 String tag 来过滤输出的信息, 关于 Android DDMS 如何使用, 请阅读 [Android DDMS 使用详细说明](#)。

除了以上方法外, 我们也可以把程序运行过程信息的输出当作程序运行的一部分, 比如使用 Toast Notification 将输出信息显示在界面中, 当然这些只是些调试代码, 在发布程序时需要去掉。

最后一种方法，也是最有效的一种方法，直接将运行过程的信息以文件的方式存储，在程序运行后打开文件，查看输出的信息。在一些复杂的工具中，都是用这种日志文件的方法来记录文件运行的过程。如何在 Android 中读写文件，请阅读 [Android 数据存储（总结篇）](#)。

看了以上 2 种方法是否觉得：以上只是在发现问题后找到问题的原因，解决问题，是不是有些被动的、消极的，有没有其他有效的方法来避免 bug？看到这里，有些“牛”人就说了：我写的代码几乎没有 bug，我的代码好几年都没有发生过崩溃现象了。从我个人的观点说：的确牛。至少我自己，感觉自己的代码似乎很脆弱，要想写一个完全正确的代码真的不容易。自己考虑了很多，为什么会这样，难道是自己写的代码的确很差？至少我自己在写代码的过程中都是很仔细的，尽量把问题考虑清楚了在写的，每次修改都是小心翼翼的！后来发现，每段代码在写的时候都是有一些“运行环境”的，在后来使用的过程中，这个环境逐渐被破坏，以致最后修改的乱七八糟。如果你也有同受，建议你仔细阅读以下说明！

使用 JUnit 调试

Android 增加了对 JUnit 的支持，这对程序员来说，是个很好消息。

首先说明下 JUnit 是用来解决什么问题的？JUnit 是采用测试驱动开发的方式，也就是说在开发前先写好测试代码，主要用来说明被测试的代码会被如何使用，错误处理等；然后开始写代码，并在测试代码中逐步测试这些代码，直到最后在测试代码中完全通过。

看了是否感觉有些不符合程序员的思维习惯（先写代码然后在调试），的确这也是 JUnit 是对程序员思维的“颠覆”。在这里我自己也感觉，好像很难做到，为什么？在一匹“马”没有完全设计好前，怎么规定这匹“马”将来会如何跑？而且即使把“马”将来会如何“跑”定义好了，在实现的时候“马”被改变了怎么办？说到底还是：一个人不能同时具有 2 个角色，否则自己有时候就不知道当前是哪个角色！

说到这里，我就说明下，我自己对 JUnit “错误”的使用方法，这也许与 JUnit 测试驱动开发的目的相矛盾，但是的确可以有效地减少 bug。JUnit 从核心来说就是将源代码与测试代码完全分开，将测试代码作为一个单独的程序。前面介绍的方法，都将源代码与测试代码合为一体，由于源代码的重要性大于测试代码的重要性，所以测试代码经常有不完整、结构不清晰等问题，这样程序员的单元

测试也就不完整。JUnit 就是被我用来做完整的单元测试，对当前的部分代码，测试其在每种“环境”下的运行结果。现简要说下 JUnit 的几个主要功能：

1. JUnit 首先有管理测试用例的功能。修改了哪些代码，这些代码的修改会对哪些部分有影响，通过 JUnit 将这次的修改做个完整测试。这也就 JUnit 中所谓的 TestSuite。
2. 如何定义需要测试的代码？这也就是 JUnit 中所谓的 TestCase，根据源代码的测试需要定义每个 TestCase，并将 TestCase 添加到相应的 TestSuite 方便管理。
3. 如何定义测试的“环境”？在 TestCase 测试前会先调用“环境”配置，在测试中使用，当然也可以在直接测试用例中定义测试“环境”。
4. 最为重要的部分，测试结果的检测。对于每种正常、异常情况下的测试，运行结果是什么、结果是否是我们预期的等都需要有个明确的定义，JUnit 在这方面提供了强大的功能。

以上部分与我们平常使用 IDE 调试的过程是完全一样的，只不过是增加了测试用例管理、测试结果检测等功能，提高了单元的效率，保证了单元测试的完整性，明确了单元测试的目标。带着以上 4 个问题，简要举例并分析如下：

源代码如下：

```
public class SampleCalculator
{
public int add(int augend , int addend)
{return augend + addend ;}
public int subtration(int minuend , int subtrahend)
{return minuend - subtrahend ;}
}
```

测试代码 (TestCase) 如下：

```
import junit.framework.TestCase;
public class TestSample extends TestCase
{
public void testAdd()
```

```
{
SampleCalculator calculator = new SampleCalculator();
int result = calculator.add(50 , 20);
assertEquals(70 , result);
}
public void testSubtration()
{
SampleCalculator calculator = new SampleCalculator();
int result = calculator.subtration(50 , 20);
assertEquals(30 , result);
}
}
```

以上 TestSample 测试用例中就对 SampleCalculator 进行了完整的单元测试，并对测试结果做了预期说明。当然还需要将 TestSample 增加到 TestCase 中方便管理。

```
import junit.framework.Test;
import junit.framework.TestSuite;
public class TestAll{
public static Test suite() {
TestSuite suite = new TestSuite(" TestSuite Test" );
suite.addTestSuite( TestSample.class);
return suite;
}
}
```

以上就将 TestSample 增加到” TestSuite Test” 中，将来在选择测试用例的过程中只要选择了 TestSuite Test， TestSample 就将加入当前测试中。如果将来 SampleCalculator 增加了其他功能，只需要在 TestSample 增加相应的 测试，就可以对 TestSample 进行完整单元测试。

看到这里对上面 4 个问题，应该都有了大致的了解。最后需要说明的：对 TestCase 的管理，是完全界面化的，只需要按照 JUnit 的要求实现会自动产生 UI 界面，这个不必担心，还需要下载 [JUnit packeage](#)，根据需求选择自己需要的。大胆尝试下，你会发现编程真的可以如此“美好”。

总结说明

以上是在工作中总结的代码调试的方法，并结合 Android 应用程序开发，为将来深入开发 Android 应用程序打好坚实的基础。

相关文章

- [android.test.InstrumentationTestRunner 解析](#)
- [android.app.instrumentation 解析](#)
- [Android、JUnit 深入浅出（三）——JUnit 深入解析（上）](#)
- [Android、JUnit 深入浅出（二）——JUnit 例子分析](#)
- [Android、JUnit 深入浅出（一）——JUnit 初步解析](#)

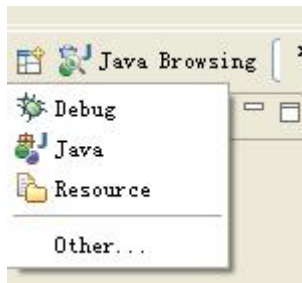
Android DDMS 如何使用？

DDMS 的全称是 Dalvik Debug Monitor Service，它为我们提供例如：为测试设备截屏，针对特定的进程查看正在运行的线程以及堆信息、Logcat、广播状态信息、模拟电话呼叫、接收 SMS、虚拟地理坐标等等。

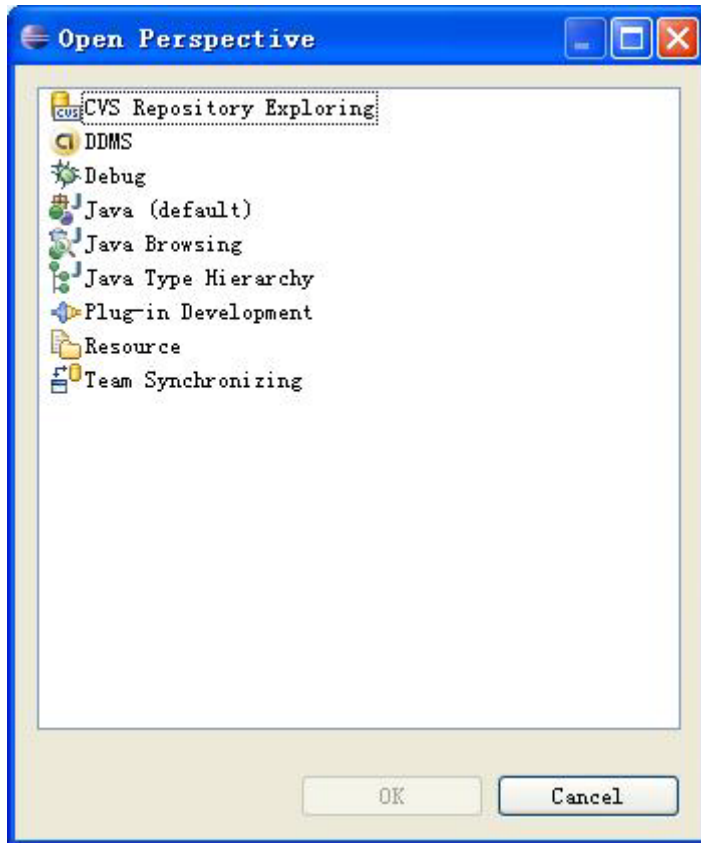
如何启动 DDMS

DDMS 工具存放在 SDK - tools/路径下，启动 DDMS 方法如下：

1. 直接双击 `ddms.bat` 运行；
2. 在 Eclipse 调试程序的过程中启动 DDMS，在 Eclipse 中的界面如下：



选择“Other”，界面如下：

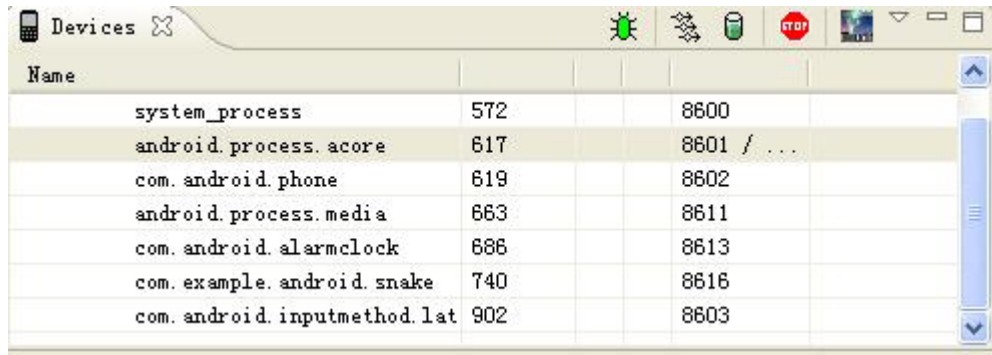


双击 DDMS 就可以启动了。

DDMS 对 Emulator 和外接测试机有同等效用。如果系统检测到它们(VM)同时运行，那么 DDMS 将会默认指向 Emulator。以上 2 种启动后的操作有些不一样，建议分别尝试下。

DDMS 的工作原理

DDMS 将搭建起 IDE 与测试终端 (Emulator 或者 connected device) 的链接, 它们应用各自独立的端口监听调试器的信息, DDMS 可以实时监测到测试终端的连接情况。当有新的测试终端连接后, DDMS 将捕捉到 终端的 ID, 并通过 adb 建立调试器, 从而实现发送指令到测试终端的目的。



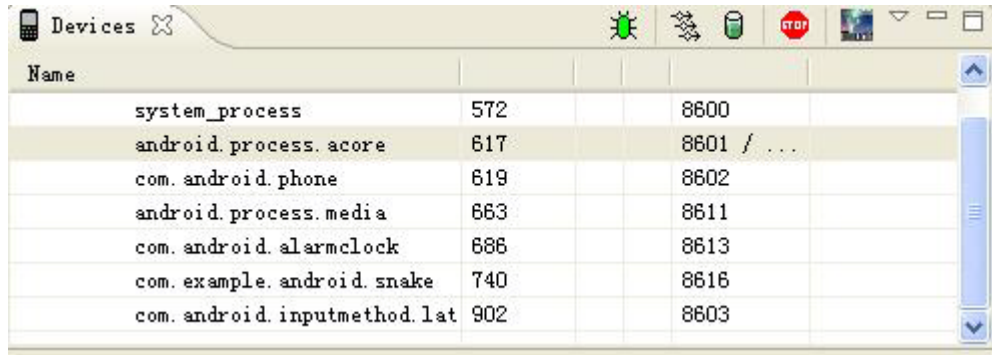
Name			
system_process	572		8600
android.process.acore	617		8601 / ...
com.android.phone	619		8602
android.process.media	663		8611
com.android.alarmclock	686		8613
com.example.android.snake	740		8616
com.android.inputmethod.la	902		8603

DDMS 监听第一个终端 App 进程的端口为 8600, APP 进程将分配 8601, 如果有更多终端或者更多 APP 进程将按照这个顺序依次类推。DDMS 通过 8700 端口 (" base port") 接收所有终端的指令。

下边通过 GUI 详细了解 DDMS 的一些功能

Devices

在 GUI 的左上角可以看到标签为 " Devices" 的面板, 这里可以查看到所有与 DDMS 连 接的终端的详细信息, 以及每个终端正在运行的 APP 进程, 每个进程最右边相对应的是与调试器链接的端口。因为 Android 是基于 Linux 内核开发的操 作平台, 同时也保留了 Linux 中特有的进程 ID, 它介于进程名和端口号之间。

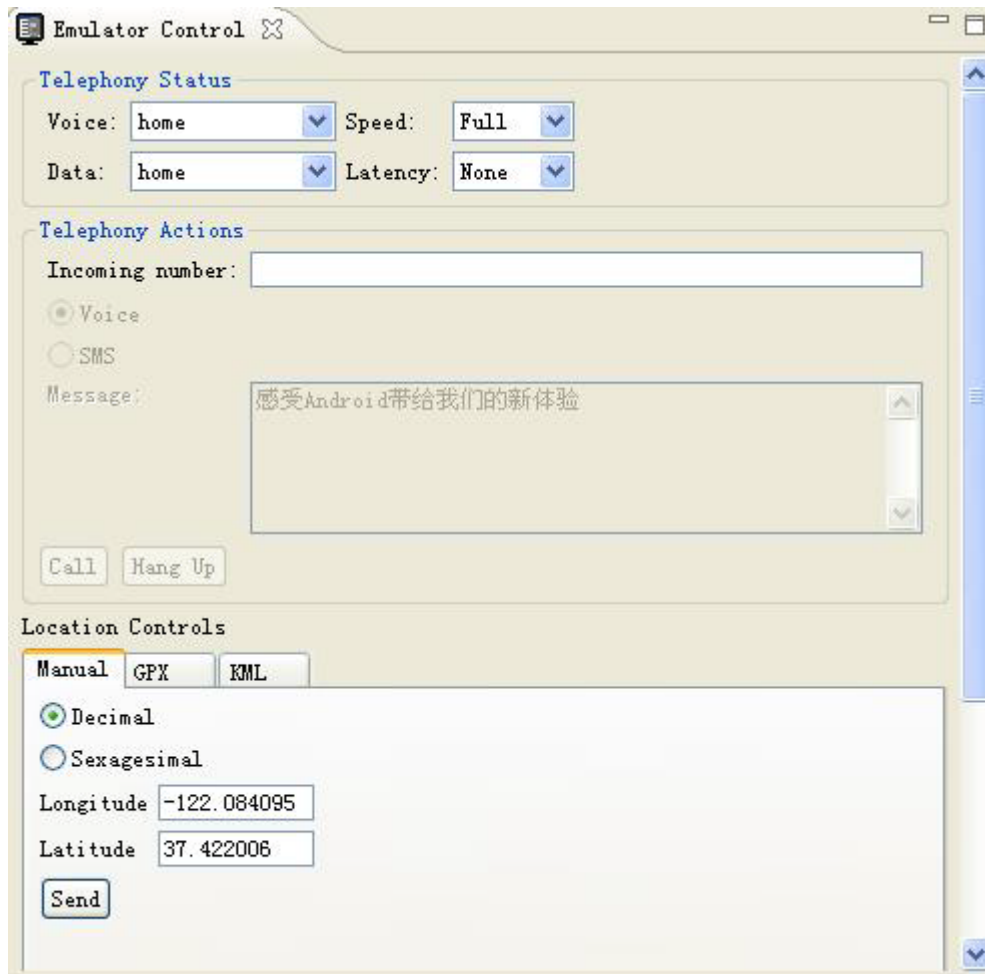


Name	PID	PPID
system_process	572	8600
android.process.acore	617	8601 / ...
com.android.phone	619	8602
android.process.media	663	8611
com.android.alarmclock	686	8613
com.example.android.snake	740	8616
com.android.inputmethod.la	902	8603

在面板的右上角有一排很重要的按钮他们分别是 Debug the selected process、Update Threads、Update Heap、Stop Process 和 ScreenShot。

Emulator Control

通过这个面板的一些功能可以非常容易的使测试终端模拟真实手机所具备的一些交互功能，比如：接听电话，根据选项模拟各种不同网络情况，模拟接受 SMS 消息和发送虚拟地址坐标用于测试 GPS 功能等。



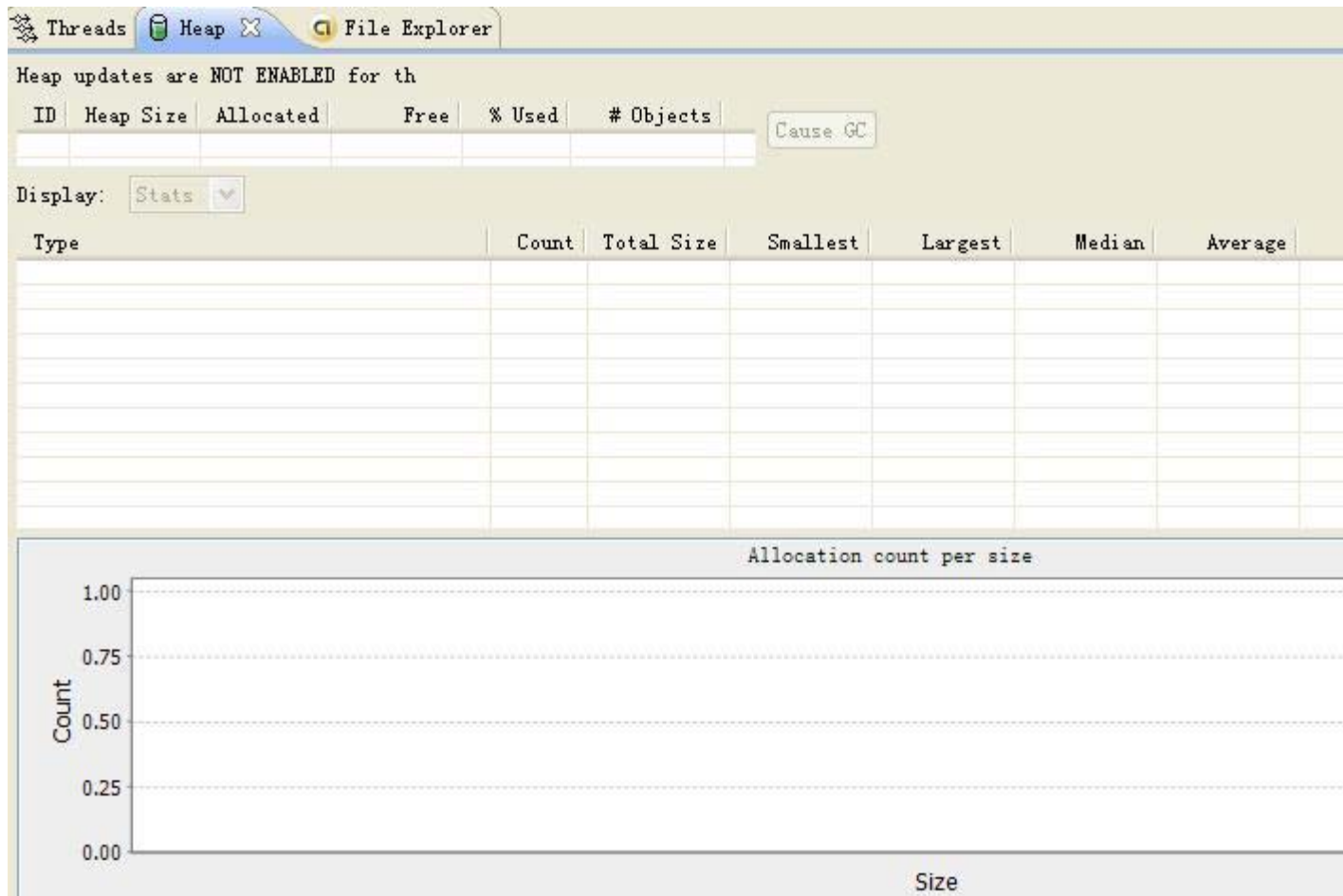
Telephony Status: 通过选项模拟语音质量以及信号连接模式。

Telephony Actions: 模拟电话接听和发送 SMS 到测试终端。

Location Control: 模拟地理坐标或者模拟动态的路线坐标变化并显示预设的地理标识, 可以通过以下 3 种方式:

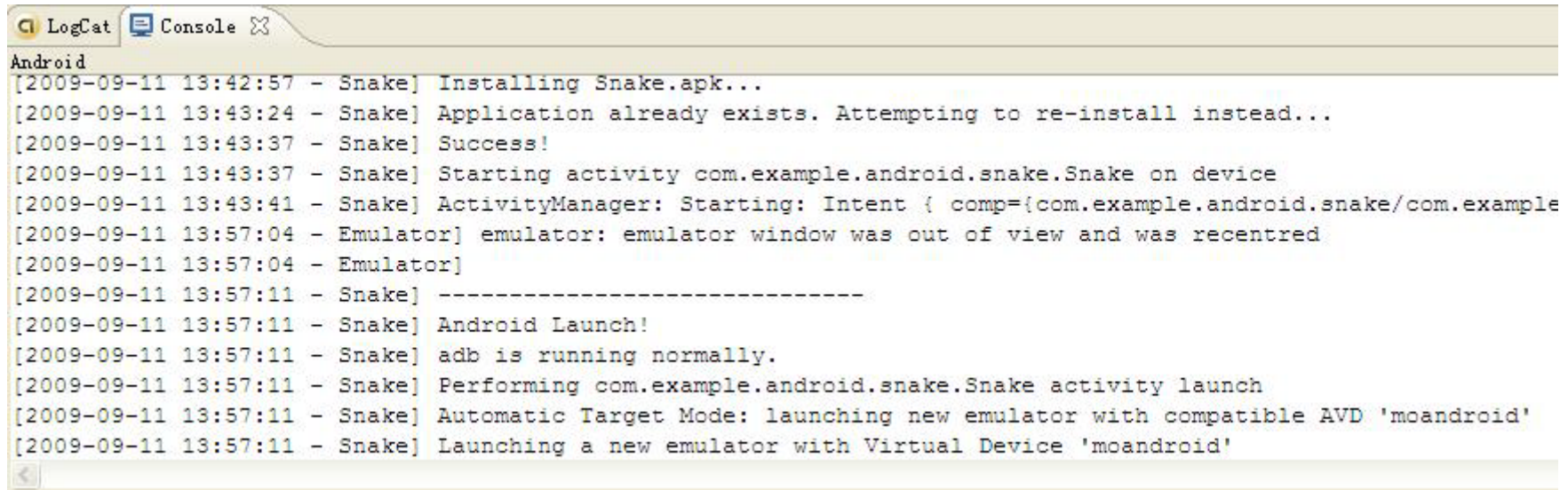
- **Manual:** 手动为终端发送二维经纬坐标。
- **GPX:** 通过 GPX 文件导入序列动态变化地理坐标, 从而模拟行进中 GPS 变化的数值。
- **KML:** 通过 KML 文件导入独特的地理标识, 并以动态形式根据变化的地理坐标显示在测试终端。

Threads、Heap、File Explorer



这几项，我们在其他开发工具中也经常使用，就不在此详细说明了。通过 File Explorer 可以查看 Android 模拟器中的文件，可以很方便的导入/出文件。

Locate、Console



```
LogCat Console
Android
[2009-09-11 13:42:57 - Snake] Installing Snake.apk...
[2009-09-11 13:43:24 - Snake] Application already exists. Attempting to re-install instead...
[2009-09-11 13:43:37 - Snake] Success!
[2009-09-11 13:43:37 - Snake] Starting activity com.example.android.snake.Snake on device
[2009-09-11 13:43:41 - Snake] ActivityManager: Starting: Intent { comp={com.example.android.snake/com.example
[2009-09-11 13:57:04 - Emulator] emulator: emulator window was out of view and was recentred
[2009-09-11 13:57:04 - Emulator]
[2009-09-11 13:57:11 - Snake] -----
[2009-09-11 13:57:11 - Snake] Android Launch!
[2009-09-11 13:57:11 - Snake] adb is running normally.
[2009-09-11 13:57:11 - Snake] Performing com.example.android.snake.Snake activity launch
[2009-09-11 13:57:11 - Snake] Automatic Target Mode: launching new emulator with compatible AVD 'moandroid'
[2009-09-11 13:57:11 - Snake] Launching a new emulator with Virtual Device 'moandroid'
```

Locate: 显示输出的调试信息，详见 [Android 下如何调试程序?](#)；

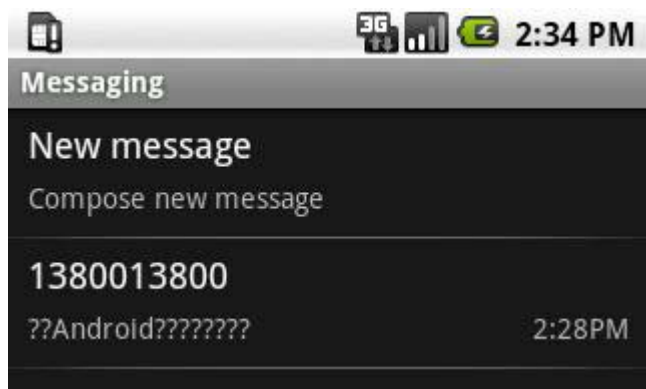
Console: 是 Android 模拟器输出的信息，加载程序等信息；

使用 DDMS 模拟发送短信，操作过程如下：

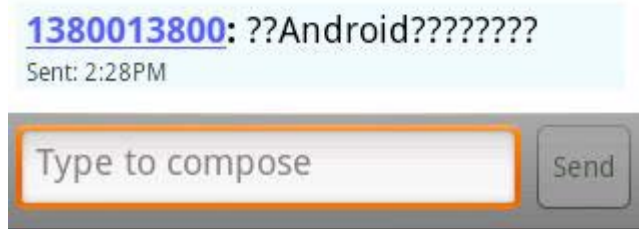
在 Emulator Control\Telephony Actions 中输入以下内容



单击发送后，在 Android 模拟器中打开 Messaging，看到下面的短信：



单击新短信，详细查看短信内容：



中文显示为乱码，在未来的开发中，我们必须要注意中文字符的问题。

总结说明

DDMS 是我们开发人员最好的调试工具，它将是每个从事 Android 开发的人员都不可缺少的。

随机日志

- [Android 2.0 SDK 发布了](#)
- [另眼看中国移动的 Mobile Market](#)
- [据统计 Android Market 应用程序数量已超 10000](#)
- [作为一个 Android 开发者，你会不会因 Android 1.6 而感到困扰？](#)
- [Android 画图学习总结（四）——Animation（上）](#)