# android手机性能测试

## 测试工具DDMS（Dalvik Debug Monitor Service）

### 安装与配置

1. 首先安装JDK，1.5以上的版本（目前java vuser不支持JDK1.7）
2. 在安装完JDK 后，就需要下载及安装Android SDK，即： android-sdk-windows，压缩包大约有551M左右
3. 解压缩android-sdk-windows，放在C盘的根目录下，配置系统变量path 的值为：C: \android-sdk-windows\tools

### 启动DDMS

1. 可以在开始--运行中进入DDMS
2. 也可以在C: \android-sdk-windows\tools目录下启动ddms.bat

### 连接DDMS

1. 使用数据线连接安卓系统的手机，确认手机是处于“USB调试”模式。
	1. 在手机上按下“Menu”键，在弹出的菜单中选择“Setting（设置）”；
	2. 选择“应用程序”；
	3. 在此界面勾选“未知来源”，然后选择“开发”；
	4. 勾选“USB调试”，“保持唤醒状态”；
2. 在ddms的左边框中会显示手机已经打开的应用程序(APP)进程，如果不显示，可以多连接几次，或者换个手机试



### 操作DDMS

1. 点击选中想要监测的进程，比如system\_process进程；
2. 点击选中Devices视图界面中最上方一排图标中的“Update Heap”图标；
3. 点击Heap视图中的“Cause GC”按钮；
4. 此时在Heap视图中就会看到当前选中的进程的内存使用量的详细情况。

### 分析DDMS

如何才能知道我们的程序是否有内存泄漏的可能性呢。这里需要注意一个值：Heap视图中部有一个Type叫做data object，即数据对象，也就是我们的程序中大量存在的类类型的对象。在data object一行中有一列是“Total Size”，其值就是当前进程中所有Java数据对象的内存总量，一般情况下，这个值的大小决定了是否会有内存泄漏。可以这样判断：

1. 不断的操作当前应用，同时注意观察data object的Total Size值；
2. 正常情况下Total Size值都会稳定在一个有限的范围内，也就是说由于程序中的的代码良好，没有造成对象不被垃圾回收的情况，所以说虽然我们不断的操作会不断的生成很多对象，而在虚拟机不断的进行GC的过程中，这些对象都被回收了，内存占用量会会落到一个稳定的水平；
3. 反之如果代码中存在没有释放对象引用的情况，则data object的Total Size值在每次GC后不会有明显的回落，随着操作次数的增多Total Size的值会越来越大，直到到达一个上限后导致进程被kill掉。
4. 此处已system\_process进程为例，在我的测试环境中system\_process进程所占用的内存的data object的Total Size正常情况下会稳定在2.2~2.8之间，而当其值超过3.55后进程就会被kill掉

## Android程序的内存泄漏与规避方法

### 造成Android应用程序内存泄漏的原因

#### 1、引用没释放造成的内存泄露

* 1. 注册没有取消造成的内存泄漏

这种Android的内存泄露比纯Java的内存泄漏还要严重，因为其他一些Android程序可能引用系统的Android程序的对象(比如注册机制)。即使Android程序已经结束了，但是别的应用程序仍然还有对Android程序的某个对象的引用，泄漏的内存依然不能被垃圾回收。

* 1. 集合中对象没有关闭造成的内存泄漏

通常把一些对象的引用加入到了集合中，当我们不需要该对象时，并没有把它的引用从集合中清理掉，慢慢地这个集合就会越来越大。如果这个集合是静态的话，那情况就会更严重。

#### 2、资源对象没有关闭造成的内存泄漏

资源对象比如Cursor、File文件等往往都用了一些缓冲，在不使用的时候应该及时关闭它们，以便它们的缓冲及时回收内存。这些缓冲不仅存在于Java虚拟机内，还存在于Java虚拟机外，如果仅仅是把它的引用设置为空,而不关闭它们，那么往往会造成内存泄漏。

#### 3、一些不良代码造成的内存压力原因如下：

* 1. Bitmap没有调用recycle( )；
	2. 构造Adapter时，没有使用缓存的convertView；
	3. ThreadLocal使用不当；

### 内存泄漏的检测及定位

#### 1、内存泄漏的检测

Android应用程序是基于虚拟机的，其内存管理都是由Dalvik[2]代为管理，GC的回收不是及时的。一个正常的应用程序在其运行稳定后其内存的占用量是基本稳定的，不应该是无限制的增长。同样，对任何一个类的对象的使用个数也有一个相对稳定的上限，不应该是持续增长的。当我们持续地观察某个应用程序运行过程中使用内存的大小和各实例的个数时，如果内存的大小持续增长，则说明系统存在内存泄漏的问题;如果特定类的实例对象个数随时间而增长，则说明这个类的实例可能存在泄漏情况。

在重复打开关闭某个应用程序的时候，内存一直在向上爬升，也就是说每次关闭这个Activity的时候，有些应该释放的内存并没有被释放掉。

#### 2、内存泄漏的位置定位

查找内存泄漏一种比较彻底的方法就是代码走查，我们可以一行一行地分析对象的创建去留等等[4]，但会很耗时间也比较迷茫。这里可以通过Eclipse Memory Analyzer Tool(MAT)工具来定位内存泄漏的位置，该方法只适用于Java层的查找，对C/C++没用，也就是说只针对于被虚拟机来管理的进程和内存。MAT的解析文件是.hprof文件，这个文件里面存放了某进程的内存快照，MAT通过解析.hprof文件就会自动生成一个内存泄漏推测报告，通过分析这个报告就可以准确定位到有可能存在内存泄漏的具体位置。

注：还有一些内存泄漏通过MAT是查不出来的，如native的代码对C/C++无效

### 规避内存泄漏的方法

1. 在编写应用程序的过程中，对于BraodcastReceiver、ContentObserver、FileObserver在Activity onDestory或者某类声明周期结束之后一定要注销掉，否则这个Activity类会被系统强引用，不会被内存回收。
2. 在定义成员变量时，不要直接对Activity进行引用而作为成员变量。如果不得不这么做，那么可以用private Weak Reference mActivity来声明。同样，对于Service等其他有自己声明周期的对象来说，直接引用都需要谨慎考虑是否会存在内存泄漏的可能。
3. 在应用程序中，很多内存泄漏是由于循环引用而造成的，比如a中包含了b，b中包含了c，c中又包含a，这样只要一个对象存在，那么其他对象肯定会一直常驻内存。因此，在编写应用程序时要从逻辑上来分析是否需要这样的设计。
4. Bitmap对象不再使用时，调用recycle()方法释放内存。如果一个Bitmap对象比较占内存，当它不再被使用的时候，可以调用Bitmap.recycle()方法回收此对象的像素所占用的内存，这个不是必须的，可视情况而定。
5. 还要注意释放对象的引用。当一个生命周期较短的对象A，被一个生命周期较长的对象B保有其引用的情况下，在A的生命周期结束时，要在B中清除掉对A的引用。

## Android常见容易引起内存泄漏的一些代码

#### ****查询数据库没有关闭游标****

**程序中经常会进行查询数据库的操作，但是经常会有使用完毕Cursor后没有关闭的情况。如果我们的查询结果集比较小，对内存的消耗不容易被发现，只有在常时间大量操作的情况下才会复现内存问题，这样就会给以后的测试和问题排查带来困难和风险示例代如下码：**



#### 构造Adapter时，没有使用缓存的 convertView

#### Bitmap对象不在使用时调用recycle()没有及时释放

如果一个Bitmap对象比较占内存，当它不在被使用的时候，可以调用Bitmap.recycle()方法回收此对象的像素所占用的内存

#### 没有及时释放对象的引用

## OOM调试

### 方式一：使用内存监测工具 DDMS –> Heap：（真机、模拟器均可使用）

#### 工具介绍

1. 启动eclipse后，切换到DDMS透视图，并确认Devices视图、Heap视图都是打开的，没打开的直接Window>ShowView>自己选；
2. 将手机通过USB链接至电脑，链接时需要确认手机是处于“USB调试”模式
3. 链接成功后，在DDMS的Devices视图中将会显示手机设备的序列号，以及设备中正在运行的部分进程信息；
4. 点击选中想要监测的进程，如果在进程列表中未出现你的进程的话随便选中一条让Device一排的工具处于可用状态，再点击下Update Heap让其自动找到我们跑的应用的进程，比如小马临时跑的两个应用进程如图：



1. **点击Heap视图中的“Cause GC”按钮；**



1. **点击Cause GC之后就可以看到我们应用的内存情况如下图：**



**说明：**

* 1. **点击“Cause GC”按钮相当于向虚拟机请求了一次gc操作；**
	2. **当内存使用信息第一次显示以后，无须再不断的点击“Cause GC”，Heap视图界面会定时刷新，在对应用的不断的操作过程中就可以看到内存使用的变化；**
	3. **内存使用信息的各项参数根据名称即可知道其意思**

#### 定位分析

**这里需要注意一个值：Heap视图中部有一个Type叫做data object，即数据对象，也就是我们的程序中大量存在的类类型的对象。在data object一行中有一列是“Total Size”，其值就是当前进程中所有Java数据对象的内存总量，如果大家想要看“Total Size”是分配的具体信息可以点击“data object这一行来查看详细信息，如下图”：**

****

**一般情况下，在data object行的“Total Size”这个值的大小决定了是否会有内存泄漏。可以这样判断：**
**a) 不断的操作当前应用，同时注意观察data object的Total Size值；**
**b) 正常情况下Total Size值都会稳定在一个有限的范围内，也就是说由于程序中的的代码良好，没有造成对象不被垃圾回收的情况，所以说虽然我们不断的操作会不断的生成很多对 象，而在虚拟机不断的进行GC的过程中，这些对象都被回收了，内存占用量会会落到一个稳定的水平；**
**c) 反之如果代码中存在没有释放对象引用的情况，则data object的Total Size值在每次GC后不会有明显的回落，随着操作次数的增多Total Size的值会越来越大，**
**直到到达一个上限后导致进程被杀掉。**

**d) 此处以com.xiaoma.www进程为例，在我的测试环境中com.xiaoma.www进程所占用的内存的data object的Total Size正常情况下稳定在0.8~1.0M之间，而当其值超过3~5M每次启动应用该值不稳定的时候进程就会被系统杀掉啦！**

**方式二:**

### 方式二：DDMS –> Heap：使用内存分析工具 MAT(Memory Analyzer Tool)

1. 生成.hprof文件 （生成很简单，直接点击Device 工具栏中的 Dump HPROF file即可生成
2. 使用MAT导入.hprof文件
3. 使用MAT的视图工具分析内存

总之，使用DDMS的Heap视图工具可以很方便的确认我们的程序是否存在内存泄漏的可能性。这个地方顺带着也简单的说一下，如果大家想要跟踪更详细的内存是怎样分配的话，可以学着使用下Windows>ShowView>Allocation Tracker工具来定位你的内存什么时候被什么东西占用了，是bundlea或HashMap或ArrayList焦点或是其它的什么占用了这些都可以在这个插件中查看到的，简单的使用方法就是：选中Device进程列表中的某一个进程，让Allocation Tracker里面的跟踪工具处于可用状态，点击Stop Tracking来跟踪程序，过一定时间后，点击Get Allocations来获取内存分配的消息就可以查看更为详细的内存分配情况了，如下图：

