## 1，软引用与强、弱引用

软引用：

在Android应用的开发中，为了防止内存溢出，在处理一些占用内存大而且声明周期较长的对象时候，可以尽量应用软引用和弱引用技术。

使用：假设我们的应用会用到大量的默认图片，比如应用中有默认的头像，默认游戏图标等等，这些图片很多地方会用到。如果每次都去读取图片，由于读取文件需要硬件操作，速度较慢，会导致性能较低。所以我们考虑将图片缓存起来，需要的时候直接从内存中读取。但是，由于图片占用内存空间比较大，缓存很多图片需要很多的内存，就可能比较容易发生OutOfMemory异常。这时，我们可以考虑使用软引用技术来避免这个问题发生。

使用软引用以后，在OutOfMemory异常发生之前，这些缓存的图片资源的内存空间可以被释放掉的，从而避免内存达到上限，避免Crash发生。

需要注意的是，在垃圾回收器对这个Java对象回收前，SoftReference类所提供的get方法会返回Java对象的强引用，一旦垃圾线程回收该Java对象之后，get方法将返回null。所以在获取软引用对象的代码中，一定要判断是否为null，以免出现NullPointerException异常导致应用崩溃。

首先定义一个HashMap，保存软引用对象。

|  |
| --- |
| private Map<String, SoftReference<Bitmap>> imageCache = new HashMap<String, SoftReference<Bitmap>>(); |

再来定义一个方法，保存Bitmap的软引用到HashMap。

|  |
| --- |
| public void addBitmapToCache(String path) {  // 强引用的Bitmap对象  Bitmap bitmap = BitmapFactory.decodeFile(path);  // 软引用的Bitmap对象  SoftReference<Bitmap> softBitmap = new SoftReference<Bitmap>(bitmap);  // 添加该对象到Map中使其缓存  imageCache.put(path, softBitmap);  } |

弱引用：

弱引用与软引用的根本区别在于：只具有弱引用的对象拥有更短暂的生命周期，可能随时被回收。而只具有软引用的对象只有当内存不够的时候才被回收，在内存足够的时候，通常不被回收。

强引用：

不会被GC回收，即使当内存不够抛出OutOfMemory。一般情况不可或缺的对象会声明成强引用。

## 2，Activity的启动模式

在android里，有4种activity的启动模式，分别为：

“standard” (默认)

“singleTop”

“singleTask”

“singleInstance”

首先要知道什么是 Task ？  
Task 翻译成中文叫做任务，那么什么是任务呢？  
Task 就是一个栈 (A task is a stack of activities.) ，这个栈里面存放了很多 Activity ，它遵循着后进先出的原则。  
栈有两个动作：压栈（把对象压入到栈当中）和弹栈（把栈中的第一个对象从栈里面拿出来）。

standard 模式启动模式，每次激活Activity时都会创建Activity，并放入任务栈中。

singleTop 如果在任务的栈顶正好存在该Activity的实例， 就重用该实例，否者就会创建新的实例并放入栈顶(即使栈中已经存在该Activity实例，只要不在栈顶，都会创建实例)。

singleTask 如果在栈中已经有该Activity的实例，就重用该实例(会调用实例的onNewIntent())。重用时，会让该实例回到栈顶，因此在它上面的实例将会被移除栈。如果栈中不存在该实例，将会创建新的实例放入栈中。

singleInstance 在一个新栈中创建该Activity实例，并让多个应用共享改栈中的该Activity实例。一旦该模式的Activity的实例存在于某个栈中，任何应用再激活该Activity时都会重用该栈中的实例，其效果相当于多个应用程序共享一个应用，不管谁激活该Activity都会进入同一个应用中。

## 3、Cursor标准用法

Cursor是Android查询数据后得到的一个管理数据集合的类，正常情况下，如果查询得到的数据量较小时不会有内存问题，而且虚拟机能够保证Cusor最终会被释放掉。

然而如果Cursor的数据量特表大，特别是如果里面有Blob信息时，应该保证Cursor占用的内存被及时的释放掉，而不是等待GC来处理。并且Android明显是倾向于编程者手动的将Cursor close掉，因为在源代码中我们发现，如果等到垃圾回收器来回收时，也就是**如果不手动关闭，系统会报错，会给用户以错误提示**。$r- C+ Z

所以我们使用Cursor的方式一般如下：& % \) p

Cursor cursor = null;

try{

cursor = mContext.getContentResolver().query(uri,null,null,null,null);

if(cursor != null){

cursor.moveToFirst();

//do something

}

}catch(Exception e){

e.printStatckTrace();

}finally{

if(cursor != null){

cursor.close();

}

}

有一种情况下，我们不能直接将Cursor关闭掉，这就是在CursorAdapter中应用的情况，但是注意，CursorAdapter在Acivity结束时并没有自动的将Cursor关闭掉，因此，你需要在onDestroy函数中，手动关闭。

* @Override 3 o, S0 X( j( Y/ B$ ]. [1 {2 s  
  protected void onDestroy() { ' [\* T. [. \_1 Q0 @7 O; \  
  if (mAdapter != null && mAdapter.getCurosr() != null) { 1 ~7 }2 l& E4 Y. ?) J/ V! n( K+ L: c  
  mAdapter.getCursor().close();   
  } ( `9 d- E" U% i" S/ V2 X\* N  
  super.onDestroy();   
  }

CursorAdapter中的changeCursor函数，会将原来的Cursor释放掉，并替换为新的Cursor，所以你不用担心原来的Cursor没有被关闭。

你可能会想到使用Activity的managedQuery来生成Cursor，这样Cursor就会与Acitivity的生命周期一致了，多么完美的解决方法！然而事实上managedQuery也有很大的局限性。

managedQuery生成的Cursor必须确保不会被替换，因为可能很多程序事实上查询条件都是不确定的，因此我们经常会用新查询的Cursor来替换掉原先的Cursor。因此这种方法适用范围也是很小。

## 4、扑克牌中的哲理

1.小王一出，基本上都会被大王拍死。说明老大在，老二最好不要发话！

2.没有一张大牌开路，再顺的小牌都出不去。说明领导很重要！

3.无论你多会记牌、打牌，都抵不过人家手中的一把好牌。说明实力比能力更重要！

4.如果一堆小牌连不起来，即使拿个双王也未必会赢。说明再牛的领导也需要一个好团队！

5.必要的时候，即使拆散自己的牌也要送走搭档，说明大局很重要 ！

## 6、横竖平切换的生命周期

(1)设置 android:configChanges="orientation" 和不设置这个属性，生命周期表现为重新创建activity

(2)设置 android:configChanges="orientation|keyboardHidden"，在2.3上表现为不重新创建activity，4.0如下

　　a)android:targetSdkVersion<="12"，生命周期表现为不重新创建activity

　　b)android:targetSdkVersion>"12"，表现为重新创建activity

(3)设置 android:configChanges="orientation|keyboardHidden|screenSize"，在2.3和4.0上都表现为不重新创建

## 7、程序员出路（远远不止这三个）

三条出路可选择   
出路一:向IT培训师转型

分析:IT企业在招聘新人后，一般都须经过培训才能上岗，这个培训有时是在岗位上进行的，由富有实践经验的“老人”负责。年轻人想要进入IT行业，必须接受这样的专业培训。如今IT培训市场蛋糕相当大，利润空间可观，行业发展前景看好。根据国际数据公司(IDC)统计数据显示，2004年中国IT培训市场共实现销售额24.4亿元，相比2003年增长了22.1%，2005年市场前景发展更为可观。我无论是在IT公司内部转型担任培训工作，还是到IT培训机构担任教师，都应该有发展空间。这样的工作不会有太大压力，且比较稳定，对年龄也没什么要求。更何况，这还没有脱离我所喜欢的IT行业。

优势:在这个行业工作了8年，拥有丰富编程、项目经验，曾参与过许多成功案例，这都是我的优势。而8年积攒的行业人脉关系，还可以帮助我发展企业客户，顺便担任市场拓展工作。

挑战:我在过去8年中一直埋头苦干，只是在公司培训新人时担任过培训工作，缺少培训工作经验。如果要向培训师转型，我需要接受一些专业训练，提高担任培训师的能力。

出路二:向IT营销转型

分析:IT与其它行业不同，由于技术性较强，营销人员一般也须有较强的技术背景，且最好有着丰富经验。互联网公司、

电子商务公司、软件公司目前都需要有技术背景的营销人员;另外，电子商务的再度火爆，使得传统企业开始重视电子商务营销，也需要有技术背景的营销人员。这类工作薪水不低，加上提成应该可以拿到高于程序员的薪水，同时也比较稳定。

优势:深厚的技术背景仍然是我的核心竞争力，在8年的从业经历中，我广泛涉猎了各种编程软件和工具，对IT行业也有相对深入的了解，这些都为我转向营销提供了平台。

挑战:我过去的工作主要是跟机器打交道，而营销工作恰恰需要跟人打交道，这对我来说是最大的挑战，要求我彻底改变工作方式，加强沟通能力与人际交往能力;此外，营销工作会面临销售指标的压力，这些问题都是我转型前必须深入考虑的。

出路三:向项目管理努力

分析:如果说前面两条出路转型的成分多，那么项目管理可以说是我更为便捷的发展之路。目前，软件项目经理是人才市场上炙手可热的人才，有丰富经验、外语好的软件项目经理是抢手的香饽饽，供不应求，薪水自然也是水涨船高。最重要的是，一个优秀的项目经理可以在这个岗位上长久工作下去，并且有向高层进一步提升的可能。

优势:对我这样有经验、有技术、有人脉、有能力、参与过多个软件开发、有一定经验的人，项目管理无疑是我发展的一个很好的方向。

挑战:我以往参与过不少项目的开发，但一直是项目团队里的一分子，没担任过项目带头人。而很多公司在招聘项目经理时，都需要有带领团队进行项目开发的经验，这是我的一个劣势。建议我参加一些项目管理方面的培训课程，自学相关的知识，增长自己在流程管理、沟通协调等方面的能力，以便积累资本，逐步向项目管理岗位迈进。

## 8、String作为函数参数传递时，传值还是传址的疑惑（一般人看不明白，不用纠结）

public class Test {

 /\*\*  
  \* @param args  
  \*/  
 public static void change(String str){  
  System.out.println("before = str\*\*\*\*\*" + str.hashCode());  
  str = "welcome";  
  System.out.println("after = str\*\*\*\*\*" + str.hashCode());  
 }  
 public static void main(String[] args) {  
  // TODO Auto-generated method stub  
  //MyTestStr.demoStr();  
  String str = "hello";  
  System.out.println("str---" + str);  
  System.out.println("str---" + str.hashCode());  
  change(str);  
  System.out.println("str---" + str);  
  System.out.println("str---" + str.hashCode());  
    
 }

输出结果为

before change,str-----hello  
before change,hashcode of str-----99162322  
before =,str\*\*\*\*\*hello  
before =, hashcode of str\*\*\*\*\*99162322  
after =,str\*\*\*\*\*welcome  
after =, hashcode of str\*\*\*\*\*1233099618  
after change,str-----hello  
after change,hashcode of str-----99162322

如果你在没有执行=之前输出hashcode，那么这个编码应该是和main方法里面的a的hashcode相同的。  
但是当你执行了=之后，这个hashcode应该已经不同了，它们不是同一个对象。

看一下String类的源代码，因为String用了一个private final char value[]来保存字符串，也就是这个字符串一旦创建，就不可以修改，直接导致了String类没有set方法。我们也就没法修改String中内容，当你用=给String赋值时，实际上并没有修改原先那个引用的值，而是新给了一个值。

 一个变量被当作参数传入了方法，Java是把这个变量的值传入这个方法里面——String是一个应用类型，它在堆里面的值就是一个引用，因此传String的变量到方法，事实上传的是这个引用。当你对这个变量进行赋值的时候，修改的是这个变量的值，而不是这个变量指向（引用）的对象，即可以更改变量引用的地址，但不能改变引用的内容。  
    如果你要一个String传入一个方法后，方法里面对该String的修改对方法外面有效的话，你可以用StringBuffer，并且在方法里面不是直接对这个变量进行赋值，而是通过StringBuffer的方法对这个字符串的对象进行操作，如append，不过这严格来说也不是String对象了。

再如String a1 = "a1";a1=a1+"bb"时，相当与String tmp=new  String(a1) + new String("bb");  a1=tmp;  
这样容易看出tmp指向重新在堆中分配内存，a1作为引用被重新赋值不再指向a所指向的堆内存，说明此时a1和a没有任何关系，不难理解了！

## 加油吧 童鞋们！

