1. **异常 错误**

Throwable

Error Exception

不应该试图捕 RuntimeException CheckedException

OutOfMemoryError 无需声明 必须捕获

ArithemticException(整数除0) ClassNotFoundException

ClassCastException（类型强转) NoSuchMethodException

IndexOutOfBoundsException IoException

NullPointerException

1. **抽象类 接口**

1.抽象类有构造，但不能实例化，其抽象类由子类调用。

2.继承抽象类可实现部分父类方法，非抽象方法可不实现，接口必须全部实现

3.单继承，可实现多接口

4.接口声明类型，抽象类实现接口，具体类型可选择实现java接口，或继承实现接口的抽象类 java接口——抽象类——具体类 缺省适配模式

1. **List set map 用法区别**

List：继承Collection，按对象进入的顺序保存元素，不做排序，元素不可重复

LinkedList，ArrayList，Vector继承于List

LinkList：基于双向链表实现，索引需要前向或后向遍历，插入只需记录本项

的前后项，所以插入快，索引慢。

ArrayList、Vevtor：

相同点：以数组形式存贮数据，允许按序号索引元素，插入元素涉及到元素

移动等内存操作，所以，索引快，插入慢

不同点： ArrayList：是非线程安全的，效率高，当需要增长时，默认增长原

来的一半。

Vector：是基于线程安全的，效率低，当需要增长时，默认增长

原来的一倍。

Set：继承Collection，每次只接受一个对象，按自己内部的排序方法排序，元素位置由该元素的HashCode决定，对象不可重复。

Map:没有继承Collection，元素按键值对应存储，无放入顺序，不允许有相同的key。

HashMap，HashTable，LinkeHashMap继承于Map

HashMap:非线程安全，高效，支持null

HashTable:线程安全，低效，不支持null

1. **String StringBuffer StringBuilder**

String：内部实现 final char[] value，故不能修改，较另外另个效率较低，适合少

量数据使用。

StringBuffer、StringBuilder：都继承于AbstractStringBuilder，内部实现 char[] value，可修改，适合操作大量数据使用，StringBuilder非线程安全，效率高，适合单线程；StringBuffer线程安全，较StringBuilder效率低，适合多线程。

1. **内存优化、性能优化工具**
2. **GC什么情况下会出现内存泄漏，怎么调试**

### 非静态内部类的静态实例

非静态内部类默认会持有外部类的引用，而又使用了该非静态内部类创建了一个静态的实例，该实例的生命周期和应用的一样长，这就导致了该静态实例一直会持有该Activity的引用，导致Activity的内存资源不能正常回收。正确的做法为：将该内部类设为静态内部类或将该内部类抽取出来封装成一个单例，如果需要使用Context，请使用ApplicationContext

### Activity的静态成员变量

静态成员变量的生命周期和应用的一样长，当Activity finish()之后，静态成员变量一直持有改Activity的引用，导致内存释放不了。

### Handler造成的内存泄漏

在Activity中直接new一个Handler的匿名内部类，这样造成匿名内部类持有一个外部类(通常是Activity)的引用(不然怎么更新ui)，但是Handler常常伴随着一个执行耗时操作的异步线程(如下载多张图片)，如果在完成耗时操作之前，Activity退出，异步线程持有handler的引用，handler持有Activity的引用，从而导致内存泄漏。

#### 防止Handler内存泄漏的措施

**通过程序逻辑来进行保护**

1.在关闭Activity的时候，把线程也关了；

2.如果Handler是被delay的Message持有的引用，在Activity的onDestroy()方法中，调用Handler响应书的removeCallbacks()方法，把消息对象从消息队列移除就行了。

**将Handler声明为静态类**

静态类不持有外部类的引用，所以Activity可以被随意回收，代码如下：

...private static class MyHandler extends Handler {

@Override

public void handleMessage(Message msg) {

}

}

...

**使用弱引用**

当Activity在内存中的对象没有任何引用，使用弱引用会让Activity对象很容易被Gc回首。代码如下：

...private static class MyHandler extends Handler {

WeakReference<CrmContactActivity> mReference;

public MyHandler(CrmContactActivity activity) {

mReference = new WeakReference<CrmContactActivity>(activity);

}

@Override

public void handleMessage(Message msg) {

final CrmContactActivity activity = mReference.get();

if (activity == null || activity.isFinishing()) {

return;

}

if (msg.what == LOAD\_MORE) {

activity.getCustomersByCompanyName(activity.companyName);

} else if (msg.what == THE\_END) {

activity.showViewIfHasCompanyName((ArrayList<CustomerModel>) activity.matchedCustomerList);

}

}

}

...

### 注册某个对象后没有反注册

### 广播接收器注册后在Activity退出时忘了反注册

### 集合对象没清理造成的内存泄漏

把大量对象的引用放入集合中，但我们不需要该对象时，记得从集合中将不需要的引用清理掉，同理，当对象不需要时，记得将对象的引用设置为null。

### 资源文件未关闭

最常见的是文件流执行完读写操作后，忘记关闭了输入流，输出流；数据库、Content Provider操作完后Cursor忘记了close等等。  
安全一点的方式是在写代码的时候，首先写完头尾，避免尾部关闭操作忘记了写。

### 代码不严谨

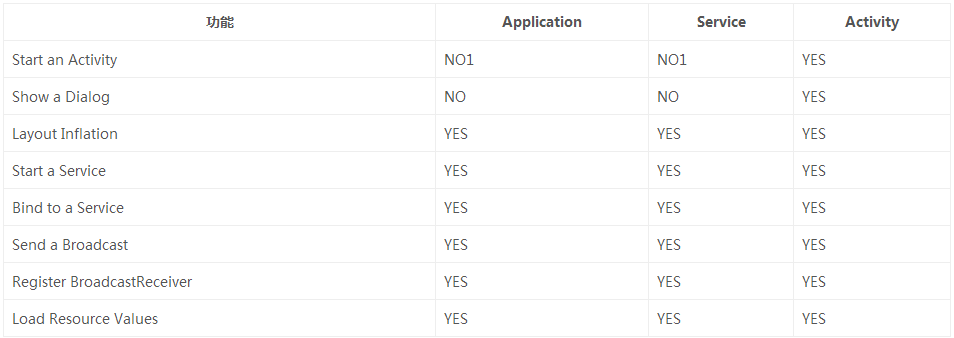
### Bitmap对象使用完后，忘记了调用recycle()方法销毁；

### 解析图片的时候忘记了设置采样率

### 自定义View时TypedArray使用完后忘记调用recycle()方法释放内存

### ListView的适配器类中没有复用convertView

# 一些建议

1. 对于生命周期比Activity长的对象如果需要应该使用ApplicationContext
2. 对于需要在静态内部类中使用非静态外部成员变量（如：Context、View )，可以在静态内部类中使用弱引用来引用外部类的变量来避免内存泄漏
3. 对于不再需要使用的对象，显示的将其赋值为null，比如使用完Bitmap后先调用recycle()，再赋为null
4. 保持对对象生命周期的敏感，特别注意单例、静态对象、全局性集合等的生命周期
5. 对于生命周期比Activity长的内部类对象，并且内部类中使用了外部类的成员变量，可以这样做避免内存泄漏：
6. 将内部类改为静态内部类
7. 静态内部类中使用弱引用来引用外部类的成员变量
8. 在涉及到Context时先考虑ApplicationContext，当然它并不是万能的，对于有些地方则必须使用Activity的Context，对于Application，Service，Activity三者的Context的应用场景如下：  
     
   其中：NO1表示Application和Service可以启动一个Activity，不过需要创建一个新的task任务队列。而对于Dialog而言，只有在Activity中才能创建
9. **==与equals**

**==** : 它的作用是判断两个对象的地址是不是相等。即，判断两个对象是不是同一个对象(基本数据类型==比较的是值，引用数据类型==比较的是内存地址)。

**equals()** : 它的作用也是判断两个对象是否相等。但它一般有两种使用情况：

* 情况1：类没有覆盖 equals() 方法。则通过 equals() 比较该类的两个对象时，等价于通过“==”比较这两个对象。
* 情况2：类覆盖了 equals() 方法。一般，我们都覆盖 equals() 方法来比较两个对象的内容是否相等；若它们的内容相等，则返回 true (即，认为这两个对象相等)。

1. **java**面向对象编程三大特性: 封装 继承 多态

在Java中有两种形式可以实现多态：继承（多个子类对同一方法的重写）和接口

1. okhttp

支持连接同一地址的链接共享同一个socket，

通过连接池来减小响应延迟

支持GZIP压缩  
响应缓存减少重复请求

1. **jvm dvm art**
2. Art减少编译，减少了CPU使用频率，使用明显改善电池续航；
3. Dalvik每次都要编译再运行，Art只会首次启动编译；
4. Art应用启动更快、运行更快、体验更流畅、触感反馈更及时；
5. Art占用空间比Dalvik大（原生代码占用的存储空间更大），就是用“空间换时间”。
6. **四大组件 五大存储 六大布局**

Sp 文件 sqlite 内容提供器（Conent Provider）方式 网络存储方式

LinearLayout 线性布局 TableLayout 表格布局 FrameLayout 帧布局

RelativeLayout 相对布局 GridLayout 网格布局AbsoluteLayout 绝对布局

1. **架构的目的是什么？**

解耦 复用 可读性 健壮性 提高并行开发效率

1. **Android 5层**

**App、framework、**系统运行库层、硬件抽象层、linux核心层

1. **Vector、ArrayList、LinkedList有何区别**

Vector 、ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。

对于随机访问get和set，Vector 、ArrayList优于LinkedList，因为LinkedList要移动指针。

对于新增和删除操作add和remove，LinedList比较占优势，因为ArrayList要移动数据

Vector线程安全 ArrayList、LinkedList非线程安全

1. **事实上**