## Android基本常识

### 1. 写10个简单的linux命令

|  |  |
| --- | --- |
| 命令 | 作用 |
| mkdir | 创建文件夹 |
| rmdir | 删除文件夹 |
| mv | 移动文件 |
| rm | 删除文件 |
| cp | 拷贝文件 |
| cat | 查看文件 |
| tail | 查看文件尾部 |
| more | 分页查看文件 |
| cd | 切换当前目录 |
| ls | 列出文件清单 |
| reboot | 重启 |
| date | 显示日期 |
| cal | 显示日历 |
| ps | 查看系统进程相当于windows的任务管理器 |
| ifconfig | 配置网络 |

### 2. 书写出android工程的目录结构

│ build.gradle 项目Gradle构建脚本  
│ gradle.properties 项目Gradle属性文件  
│ gradlew 在没有安装gradle的pc上使用,没用  
│ gradlew.bat 在没有安装gradle的pc上使用,没用  
│ local.properties 指定sdk所在目录  
│ settings.gradle 项目Gradle设置文件  
│   
├─.gradle  
├─.idea  
├─app  
│ │ .gitignore git忽略文件列表  
│ │ app.iml 临时文件,不需要关心  
│ │ build.gradle Module Gradle构建脚本  
│ │ proguard-rules.pro proguard混淆规则  
│ │   
│ ├─build 构建目录，相当于Eclipse中默认Java工程的bin目录。编译生成的apk在此目录  
│ ├─libs 依赖包  
│ └─src  
│ ├─androidTest 测试相关代码文件夹  
│ │ └─java  
│ │ └─com  
│ │ └─itheima  
│ │ └─myapplication  
│ │ ApplicationTest.java  
│ │   
│ └─main  
│ │ AndroidManifest.xml 清单文件  
│ │   
│ ├─assets  
│ ├─aidl  
│ ├─java 项目源码  
│ │ └─com  
│ │ └─itheima  
│ │ └─myapplication  
│ │ MainActivity.java  
│ │   
│ ├─jni 放置c代码  
│ ├─jniLibs 放置so库  
│ ├─assets  
│ └─res 资源文件  
│ ├─drawable .9图片只能放到drawable目录下  
│ ├─layout  
│ │ activity\_main.xml  
│ │   
│ ├─menu  
│ │ menu\_main.xml  
│ │   
│ ├─mipmap-hdpi 类似drawable-hdpi  
│ │ ic\_launcher.png  
│ │   
│ ├─mipmap-mdpi 类似drawable-mdpi  
│ │ ic\_launcher.png  
│ │   
│ ├─mipmap-xhdpi 类似drawable-xdpi  
│ │ ic\_launcher.png  
│ │   
│ ├─mipmap-xxhdpi 类似drawable-xxdpi  
│ │ ic\_launcher.png  
│ │   
│ ├─values  
│ │ dimens.xml  
│ │ strings.xml  
│ │ styles.xml  
│ │   
│ └─values-w820dp  
│ dimens.xml  
│   
├─build  
└─gradle  
 └─wrapper gradle wrapper可以看作是对gradle的封装，它可以使得在没有安装gradle的电脑上也可以使用Gradle进行构建  
 gradle-wrapper.jar  
 gradle-wrapper.properties

### 3. 什么是ANR 如何避免它？

在Android上，如果你的应用程序有一段时间响应不够灵敏，系统会向用户显示一个对话框，这个对话框称作应用程序无响应（ANR：Application Not Responding）对话框。用户可以选择让程序继续运行，但是，他们在使用你的应用程序时，并不希望每次都要处理这个对话框。因此，在程序里对响应性能的设计很重要，这样，系统不会显示ANR给用户。不同的组件发生ANR的时间不一样，主线程（Activity、Service）是5秒，BroadCastReceiver是10秒。

解决方案：

将所有耗时操作，比如访问网络，Socket通信，查询大量SQL语句，复杂逻辑计算等都放在子线程中去，然后通过handler.sendMessage()、runonUITread()、AsyncTask等方式更新UI。无论如何都要确保用户界面操作的流畅度。如果耗时操作需要让用户等待，那么可以在界面上显示进度条。

### 4. 谈谈Android的优点和不足之处

优点：

* 开放性，开源，免费，可定制
* 挣脱运营商束缚
* 丰富的硬件选择
* 不受任何限制的开发商
* 无缝结合的Google应用

缺点：

* 安全问题、隐私问题
* 同质化严重
* 运营商对Android手机仍然有影响
* 山寨化严重
* 过分依赖开发商，缺乏标准配置

### 5.一条最长的短信息约占多少byte?

在国内的三大运营商通常情况下中文70(包括标点)，英文160个。对于国外的其他运行商具体多长需要看运营商类型了。

android内部是通过如下代码进行判断具体一个短信多少byte的。

ArrayList<String> android.telephony.SmsManager.divideMessage(String text)

### 6. sim卡的EF文件有何作用？

基本文件EF(Elementary File)是SIM卡文件系统的一部分。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 文件 | 文件标识符 | 文件缩写 | 中文名称 | 文件作用 |
| MF | 3F00 | 根目录 | 备注：所有非ETSI GSM协议中规定的应用文件由各厂家自行定义在根目录下（如：PIN1，PIN2…） |  |
| EFICCID | 2FE2 | ICCID | SIM卡唯一的识别号 | 包含运营商、卡商、发卡时间、省市代码等信息 |
| DFGSM | 7F20 | GSM目录 | 备注：根据ETSIGSM09.91的规定Phase2(或以上)的SIM卡中应该有7F21并指向7F20,用以兼容Phase1的手机 |  |
| EFLP语言选择 | 6F05 | LP | 语言选择文件 | 包含一种或多种语言编码 |
| EFIMSI | 6F07 | IMSI | 国际移动用户识别符 | 包含SIM卡所对应的号段，比如46000代表135－139号段、46002代表1340－1348 |
| EFKC语音加密密钥 | 6F20 | Kc | 计算密钥 | 用于SIM卡的加密、解密 |
| EFPLMNsel网络选择表 | 6F30 | PLMNsel | 公共陆地网选择 | 决定SIM卡选择哪种网络，在这里应该选择中国移动的网络 |
| EFHPLMN归属地网络选择表 | 6F31 | HPLMN | 两次搜索PLMN的时间间隔 | 两次搜索中国移动的网络的时间间隔 |
| EFACMmax最大计费额 | 6F37 | ACMmax | 包含累积呼叫表的最大值 | 全部的ACM数据存在SIM卡中，此处取最大值 |
| EFSST SIM卡服务表 | 6F38 | SST | SIM卡服务列表 | 指出SIM卡可以提供服务的种类，哪些业务被激活哪些业务没有开通 |
| EFACM累加计费计数器 | 6F39 | ACM | 累计呼叫列表 | 当前的呼叫和以前的呼叫的单位总和 |
| EFGID1分组识别1 | 6F3E | GID1 | 1级分组识别文件 | 包含特定的SIM-ME组合的标识符，可以识别一组特定的SIM卡 |
| EFGID2分组识别2 | 6F3F | GID2 | 2级分组识别文件 | 包含特定的SIM-ME组合的标识符，可以识别一组特定的SIM卡 |
| EFPUCT单位价格/货币表 | 6F41 | PUCT | 呼叫单位的价格和货币表 | PUCT是与计费通知有关的信息，ME用这个信息结合EFACM，以用户选择的货币来计算呼叫费用 |
| EFCBMI小区广播识别号 | 6F45 | CBMI | 小区广播信息标识符 | 规定了用户希望MS采纳的小区广播消息内容的类型 |
| EFSPN服务提供商 | 6F46 | SPN | 服务提供商名称 | 包含服务提供商的名称和ME显示的相应要求 |
| EFCBMID | 6F48 | CBMID | 数据下载的小区广播消息识别符 | 移动台将收到的CBMID传送给SIM卡 |
| EFSUME | 6F54 | SUME | 建立菜单单元 | 建立SIM卡中的菜单 |
| EFBCCH广播信道 | 6F74 | BCCH | 广播控制信道 | 由于BCCH的存储，在选择小区时，MS可以缩小对BCCH载波的搜索范围 |
| EFACC访问控制级别 | 6F78 | ACC | 访问控制级别 | SIM卡有15个级别，10个普通级别，5个高级级别 |
| EFFPLMN禁止网络号 | 6F7B | FPLMN | 禁用的PLMN | 禁止选择除中国移动以外的其他运营商，比如中国联通、中国卫通等 |
| EFLOCI位置信息 | 6F7E | LOCI | 位置信息 | 存储临时移动用户识别符、位置区信息等内容 |
| EFAD管理数据 | 6FAD | AD | 管理数据 | 包括关于不同类型SIM卡操作模式的信息。例如：常规模式（PLMN用户用于GSM网络操作），型号认证模式（允许ME在无线设备的认证期间的特殊应用）；小区测试模式（在小区商用之前，进行小区测试），制造商特定模式（允许ME制造商在维护阶段进行特定的性能自动测试） |
| EFPHASE阶段 | 6FAE | PHASE | 阶段标识 | 标识SIM卡所处的阶段信息，比如是普通SIM卡还是STK卡等 |
| DFTELECOM | 7F10 | 电信目录 |  |  |
| EFADN缩位拨号 | 6F3A | AND | 电话簿 | 用于将电话记录存放在SIM卡中 |
| EFFDN固定拨号 | 6F3B | FDN | 固定拨号 | 包括固定拨号（FDN）和/或补充业务控制字串（SSC），还包括相关网络/承载能力的识别符和扩展记录的识别符，以及有关的α识别符 |
| EFSMS短消息 | 6F3C | SMS | 短消息 | 用于将短消息记录存放在SIM卡中 |
| EFCCP能力配置参数 | 6F3D | CCP | 能力配置参数 | 包括所需要的网络和承载能力的参数，以及当采用一个缩位拨号号码，固定拨号号码，MSISDN、最后拨号号码、服务拨号号码或禁止拨号方式等，建立呼叫时相关的ME配置 |
| EFMSISDN电话号码 | 6F40 | MSISDN | 移动基站国际综合业务网号 | 存放用户的手机号 |
| EFSMSP短信息参数 | 6F42 | SMSP | 短消息业务参数 | 包括短信中心号码等信息 |
| EFSMSS短信息状态 | 6F43 | SMSS | 短消息状态 | 这个标识是用来控制流量的 |
| EFLND最后拨号 | 6F44 | LND | 最后拨叫号码 | 存储最后拨叫号码 |
| EFExt1扩展文件1 | 6F4A | EXT1 | 扩展文件1 | 包括AND，MSISDN或LND的扩展数据 |
| EFExt2扩展文件2 | 6F4B | EXT2 | 扩展文件2 | 包含FDN的扩展数据 |

### 7. 如何判断是否有SD卡？

通过如下方法：Environment.getExternalStorageState().equals(Environment.MEDIA\_MOUNTED)，如果返回true就是有sdcard，如果返回false则没有。

### 8. dvm的进程和Linux的进程, 应用程序的进程是否为同一个概念？

dvm指dalvik的虚拟机。每一个Android应用程序都拥有一个独立的Dalvik虚拟机实例，应用程序都在它自己的进程中运行。而每一个dvm都是在Linux 中的一个进程，所以说可以近似认为是同一个概念。

什么是android DVM:Dalvik是Google公司自己设计用于Android平台的Java虚拟机，每一个Dalvik 应用作为一个独立的Linux 进程执行。独立的进程可以防止在虚拟机崩溃的时候所有程序都被关闭。

Dalvik和Java虚拟机的区别

1.Dalvik主要是完成对象生命周期管理，堆栈管理，线程管理，安全和异常管理，以及垃圾回收等等重要功能。

2.Dalvik负责进程隔离和线程管理，每一个Android应用在底层都会对应一个独立的Dalvik虚拟机实例，其代码在虚拟机的解释下得以执行。

3.不同于Java虚拟机运行java字节码，Dalvik虚拟机运行的是其专有的文件格式Dex

4.dex文件格式可以减少整体文件尺寸，提高I/O操作的类查找速度。

5.odex是为了在运行过程中进一步提高性能，对dex文件的进一步优化。

6.所有的Android应用的线程都对应一个Linux线程，虚拟机因而可以更多的依赖操作系统的线程调度和管理机制

7.有一个特殊的虚拟机进程Zygote，他是虚拟机实例的孵化器。它在系统启动的时候就会产生，它会完成虚拟机的初始化，库的加载，预制类库和初始化的操作。如果系统需要一个新的虚拟机实例，它会迅速复制自身，以最快的数据提供给系统。对于一些只读的系统库，所有虚拟机实例都和Zygote共享一块内存区域。

### 9. Android程序与Java程序的区别？

Android程序用android sdk开发，java程序用javasdk开发。

Android SDK引用了大部分的Java SDK，少数部分被AndroidSDK抛弃，比如说界面部分，java.awt swing package除了java.awt.font被引用外，其他都被抛弃，在Android平台开发中不能使用。android sdk 添加工具jar httpclient , pull opengl

### 10.启动应用后，改变系统语言，应用的语言会改变么？

这个一般是不会的，一般需要重启应用才能改变应用语言。但是对应应用来说如果做了国际化处理则支持如果没有处理那系统语言再更改也是无用的。

### 11.请介绍下adb、ddms、aapt的作用

adb是Android Debug Bridge ，Android调试桥的意思，ddms是Dalvik Debug Monitor Service，dalvik调试监视服务。aapt即AndroidAsset Packaging Tool，在SDK的build-tools目录下。该工具可以查看，创建，更新ZIP格式的文档附件(zip, jar, apk)。也可将资源文件编译成二进制文件，尽管我们没有直接使用过该工具，但是开发工具会使用这个工具打包apk文件构成一个Android 应用程序。

Android 的主要调试工具是adb(Android debuging bridge)，ddms是一个在adb基础上的一个图形化工具。adb，它是一个命令行工具。而ddms功能与adb相同，只是它有一个图形化界面。对不喜欢命今操作方式的人来说是一个不错的选择。

### 12.ddms 和traceview的区别

简单的说ddms是一个程序执行查看器，在里面可以看见线程和堆栈等信息，traceView是程序性能分析器。

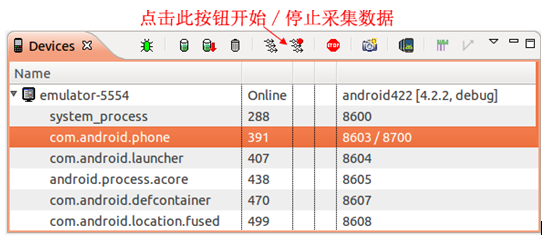
### 13.补充知识：TraceView的使用

#### 13.1 TraceView简介

Traceview是Android平台特有的数据采集和分析工具，它主要用于分析Android中应用程序的hotspot（瓶颈）。Traceview本身只是一个数据分析工具，而数据的采集则需要使用Android SDK中的Debug类或者利用DDMS工具。二者的用法如下：

开发者在一些关键代码段开始前调用Android SDK中Debug类的startMethodTracing函数，并在关键代码段结束前调用stopMethodTracing函数。这两个函数运行过程中将采集运行时间内该应用所有线程（注意，只能是Java线程）的函数执行情况，并将采集数据保存到/mnt/sdcard/下的一个文件中。开发者然后需要利用SDK中的Traceview工具来分析这些数据。

借助Android SDK中的DDMS工具。DDMS可采集系统中某个正在运行的进程的函数调用信息。对开发者而言，此方法适用于没有目标应用源代码的情况。DDMS工具中Traceview的使用如下图所示。

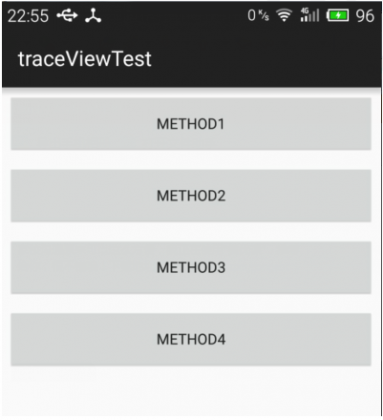


img

点击上图中所示按钮即可以采集目标进程的数据。当停止采集时，DDMS会自动触发Traceview工具来浏览采集数据。

下面，我们通过一个示例程序向读者介绍Debug类以及Traceview的使用。

实例程序如下图所示：界面有4个按钮，对应四个方法。

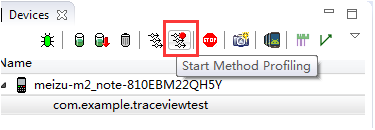


img

点击不同的方法会进行不同的耗时操作。

public class MainActivity extends ActionBarActivity {  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
 }  
  
 public void method1(View view) {  
 int result = jisuan();  
 System.out.println(result);  
 }  
 private int jisuan() {  
 for (int i = 0; i < 10000; i++) {  
 System.out.println(i);  
 }  
 return 1;  
 }  
  
 public void method2(View view) {  
 SystemClock.sleep(2000);  
 }  
  
 public void method3(View view) {  
 int sum = 0;  
 for (int i = 0; i < 1000; i++) {  
 sum += i;  
 }  
 System.out.println("sum=" + sum);  
 }  
  
 public void method4(View view) {  
 Toast.makeText(this, "" + new Date(), 0).show();  
 }  
 }

我们分别点击按钮一次，要求找出最耗时的方法。点击前通过DDMS 启动 Start Method Profiling按钮。

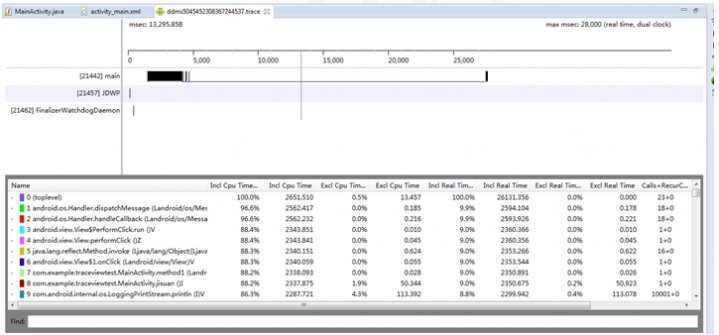


img

然后依次点击4个按钮，都执行后再次点击上图中红框中按钮，停止收集数据。

接下来我们开始对数据进行分析。

当我们停止收集数据的时候会出现如下分析图表。该图表分为2大部分，上面分不同的行，每一行代表一个线程的执行耗时情况。main线程对应行的的内容非常丰富，而其他线程在这段时间内干得工作则要少得多。图表的下半部分是具体的每个方法执行的时间情况。显示方法执行情况的前提是先选中某个线程。



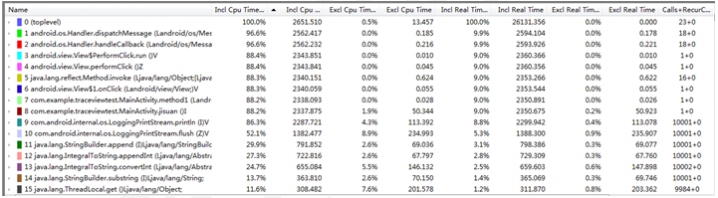
img

我们主要是分析main线程。

上面方法指标参数所代表的意思如下：

|  |  |
| --- | --- |
| 列名 | 描述 |
| Name | 该线程运行过程中所调用的函数名 |
| Incl Cpu Time | 某函数占用的CPU时间，包含内部调用其它函数的CPU时间 |
| Excl Cpu Time | 某函数占用的CPU时间，但不含内部调用其它函数所占用的CPU时间 |
| Incl Real Time | 某函数运行的真实时间（以毫秒为单位），内含调用其它函数所占用的真实时间 |
| Excl Real Time | 某函数运行的真实时间（以毫秒为单位），不含调用其它函数所占用的真实时间 |
| Call+Recur Calls/Total | 某函数被调用次数以及递归调用占总调用次数的百分比 |
| Cpu Time/Call | 某函数调用CPU时间与调用次数的比。相当于该函数平均执行时间 |
| Real Time/Call | 同CPU Time/Call类似，只不过统计单位换成了真实时间 |

我们为了找到最耗时的操作，那么可以通过点击Incl Cpu Time，让其按照时间的倒序排列。我点击后效果如下图：



img

通过分析发现：method1最耗时，耗时2338毫秒。

img

img

那么有了上面的信息我们可以进入我们的method1方法查看分析我们的代码了。

## Activity

### 1.什么是Activity?

四大组件之一，通常一个用户交互界面对应一个activity。activity 是Context的子类，同时实现了window.callback和keyevent.callback， 可以处理与窗体用户交互的事件。

常见的Activity类型有FragmentActivitiy，ListActivity，TabAcitivty等。

如果界面有共同的特点或者功能的时候，还会自己定义一个BaseActivity。

### 2.请描述一下Activity 生命周期

Activity从创建到销毁有多种状态，从一种状态到另一种状态时会激发相应的回调方法，这些回调方法包括：onCreate、onStart、onResume、onPause、onStop、onDestroy

其实这些方法都是两两对应的，

onCreate创建与onDestroy销毁；

onStart可见与onStop不可见；

onResume可编辑（即焦点）与onPause；

这6个方法是相对应的，那么就只剩下一个onRestart方法了，这个方法在什么时候调用呢？

答案就是：在Activity被onStop后，但是没有被onDestroy，在再次启动此Activity时就调用onRestart（而不再调用onCreate）方法；如果被onDestroy了，则是调用onCreate方法。

### 3.如何保存Activity的状态？

Activity的状态通常情况下系统会自动保存的，只有当我们需要保存额外的数据时才需要使用到这样的功能。

一般来说， 调用onPause()和onStop()方法后的activity实例仍然存在于内存中， activity的所有信息和状态数据不会消失， 当activity重新回到前台之后， 所有的改变都会得到保留。

但是当系统内存不足时， 调用onPause()和onStop()方法后的activity可能会被系统摧毁， 此时内存中就不会存有该activity的实例对象了。如果之后这个activity重新回到前台， 之前所作的改变就会消失。为了避免此种情况的发生， 我们可以覆写onSaveInstanceState()方法。onSaveInstanceState()方法接受一个Bundle类型的参数， 开发者可以将状态数据存储到这个Bundle对象中， 这样即使activity被系统摧毁，当用户重新启动这个activity而调用它的onCreate()方法时， 上述的Bundle对象会作为实参传递给onCreate()方法， 开发者可以从Bundle对象中取出保存的数据， 然后利用这些数据将activity恢复到被摧毁之前的状态。

需要注意的是， onSaveInstanceState()方法并不是一定会被调用的， 因为有些场景是不需要保存状态数据的. 比如用户按下BACK键退出activity时， 用户显然想要关闭这个activity， 此时是没有必要保存数据以供下次恢复的， 也就是onSaveInstanceState()方法不会被调用. 如果调用onSaveInstanceState()方法， 调用将发生在onPause()或onStop()方法之前。

@Override  
protected void onSaveInstanceState(Bundle outState) {  
 super.onSaveInstanceState(outState);  
}

### 4.两个Activity之间跳转时必然会执行的是哪几个方法？

一般情况下比如说有两个activity，分别叫A，B，当在A里面激活B组件的时候， A会调用 onPause()方法，然后B调用onCreate() ，onStart()， onResume()。这个时候B覆盖了窗体， A会调用onStop()方法. 如果B是个透明的，或者是对话框的样式， 就不会调用A的onStop()方法。

### 5.横竖屏切换时Activity的生命周期

此时的生命周期跟清单文件里的配置有关系。

1.不设置Activity的android:configChanges时，切屏会重新调用各个生命周期，默认首先销毁当前activity，然后重新加载。

2.设置Activity的android:configChanges="orientation|keyboardHidden|screenSize"时，切屏不会重新调用各个生命周期，只会执行onConfigurationChanged方法。

通常在游戏开发， 屏幕的朝向都是写死的。

### 6.如何将一个Activity设置成窗口的样式

只需要给我们的Activity配置如下属性即可。

android:theme="@android:style/Theme.Dialog"

### 7.如何退出Activity？如何安全退出已调用多个Activity的Application？

1.通常情况用户退出一个Activity只需按返回键，我们写代码想退出activity直接调用finish()方法就行。

2.记录打开的Activity：每打开一个Activity，就记录下来。在需要退出时，关闭每一个Activity即可。

//伪代码  
List<Activity> lists;// 在application 全局的变量里面  
lists = new ArrayList<Activity>();  
lists.add(this);  
for (Activity activity : lists) {  
 activity.finish();  
}  
lists.remove(this);

3.发送特定广播：

在需要结束应用时，发送一个特定的广播，每个Activity收到广播后，关闭即可。

//给某个activity 注册接受接受广播的意图   
registerReceiver(receiver, filter)  
  
//如果过接受到的是 关闭activity的广播 就调用finish()方法把当前的activity finish()掉

4.递归退出

在打开新的Activity时使用startActivityForResult，然后自己加标志，在onActivityResult中处理，递归关闭。

5.其实 也可以通过 intent的flag 来实现intent.setFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP)激活一个新的activity。此时如果该任务栈中已经有该Activity，那么系统会把这个Activity上面的所有Activity干掉。其实相当于给Activity配置的启动模式为SingleTop。

### 8.请描述一下Activity的启动模式都有哪些以及各自的特点

启动模式（launchMode）在多个Activity跳转的过程中扮演着重要的角色，它可以决定是否生成新的Activity实例，是否重用已存在的Activity实例，是否和其他Activity实例公用一个task里。这里简单介绍一下task的概念，task是一个具有栈结构的对象，一个task可以管理多个Activity，启动一个应用，也就创建一个与之对应的task。

Activity一共有以下四种launchMode：

* standard 默认启动模式
* singleTop 栈顶复用模式
* singleTask 栈内复用模式
* singleInstance 单例模式

我们可以在AndroidManifest.xml配置的android:launchMode属性为以上四种之一即可。

下面我们结合实例一一介绍这四种lanchMode：

#### 8.1 standard

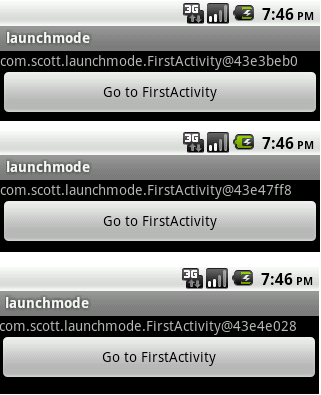
standard模式是默认的启动模式，不用为<activity>配置android:launchMode属性即可，当然也可以指定值为standard。

我们将创建一个Activity，命名为FirstActivity，来演示一下标准的启动模式。FirstActivity代码如下：

public class FirstActivity extends Activity {  
 @Override  
 public void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.first);  
 TextView textView = (TextView) findViewById(R.id.tv);  
 textView.setText(this.toString());  
 Button button = (Button) findViewById(R.id.bt);  
 button.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent intent = new Intent(FirstActivity.this, FirstActivity.class);  
 startActivity(intent);  
 }  
 });  
 }  
 }

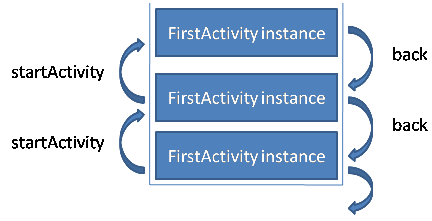
FirstActivity界面中的TextView用于显示当前Activity实例的序列号，Button用于跳转到下一个FirstActivity界面。

然后我们连续点击几次按钮，将会出现下面的现象：



img

我们注意到都是FirstActivity的实例，但序列号不同，并且我们需要连续按后退键两次，才能回到第一个FirstActivity。standard模式的原理如下图所示：



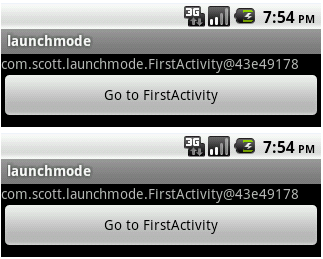
img

如图所示，每次跳转系统都会在task中生成一个新的FirstActivity实例，并且放于栈结构的顶部，当我们按下后退键时，才能看到原来的FirstActivity实例。

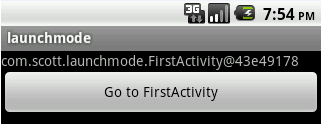
这就是standard启动模式，不管有没有已存在的实例，都生成新的实例。

#### 8.2 singleTop

我们在上面的基础上为<activity>指定属性android:launchMode="singleTop"，系统就会按照singleTop启动模式处理跳转行为。我们重复上面几个动作，将会出现下面的现象：

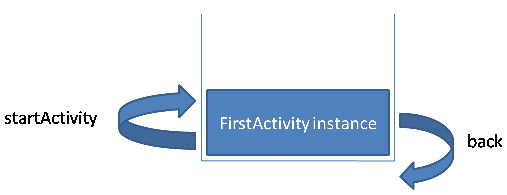


img



img

我们看到这个结果跟standard有所不同，三个序列号是相同的，也就是说使用的都是同一个FirstActivity实例；如果按一下后退键，程序立即退出，说明当前栈结构中只有一个Activity实例。singleTop模式的原理如下图所示：



img

正如上图所示，跳转时系统会先在栈结构中寻找是否有一个FirstActivity实例正位于栈顶，如果有则不再生成新的，而是直接使用。也许朋友们会有疑问，我只看到栈内只有一个Activity，如果是多个Activity怎么办，如果不是在栈顶会如何？我们接下来再通过一个示例来证实一下大家的疑问。

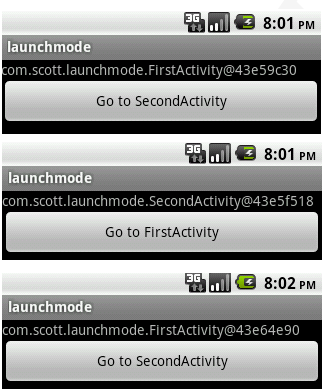
我们再新建一个Activity命名为SecondActivity，如下：

public class SecondActivity extends Activity {  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.second);  
 TextView textView = (TextView) findViewById(R.id.tv);  
 textView.setText(this.toString());  
 Button button = (Button) findViewById(R.id.button);  
 button.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent intent = new Intent(SecondActivity.this, FirstActivity.class);  
 startActivity(intent);   
 }  
 });  
 }  
 }

然后将之前的FirstActivity跳转代码改为：

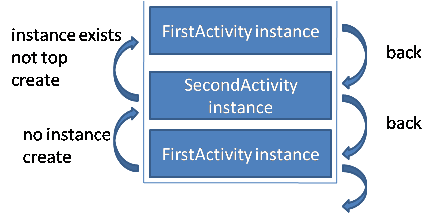
Intent intent = new Intent(FirstActivity.this, SecondActivity.class);   
startActivity(intent);

这时候，FirstActivity会跳转到SecondActivity，SecondActivity又会跳转到FirstActivity。演示结果如下：



img

我们看到，两个FirstActivity的序列号是不同的，证明从SecondActivity跳转到FirstActivity时生成了新的FirstActivity实例。原理图如下：



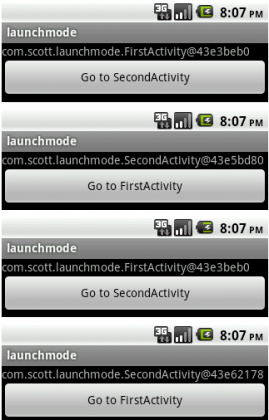
img

我们看到，当从SecondActivity跳转到FirstActivity时，系统发现存在有FirstActivity实例,但不是位于栈顶，于是重新生成一个实例。

这就是singleTop启动模式，如果发现有对应的Activity实例正位于栈顶，则重复利用，不再生成新的实例。

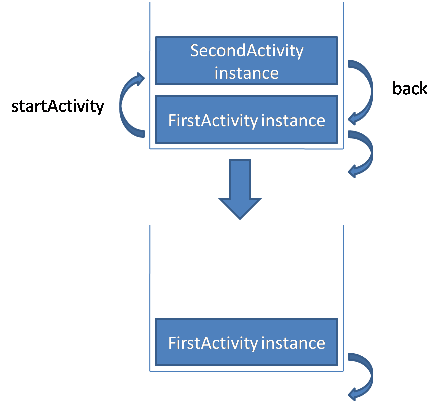
#### 8.3 singleTask

在上面的基础上我们修改FirstActivity的属性android:launchMode="singleTask"。演示的结果如下：



img

我们注意到，在上面的过程中，FirstActivity的序列号是不变的，SecondActivity的序列号却不是唯一的，说明从SecondActivity跳转到FirstActivity时，没有生成新的实例，但是从FirstActivity跳转到SecondActivity时生成了新的实例。singleTask模式的原理图如下图所示：



img

在图中的下半部分是SecondActivity跳转到FirstActivity后的栈结构变化的结果，我们注意到，SecondActivity消失了，没错，在这个跳转过程中系统发现有存在的FirstActivity实例，于是不再生成新的实例，而是将FirstActivity之上的Activity实例统统出栈，将FirstActivity变为栈顶对象，显示到幕前。也许朋友们有疑问，如果将SecondActivity也设置为singleTask模式，那么SecondActivity实例是不是可以唯一呢？在我们这个示例中是不可能的，因为每次从SecondActivity跳转到FirstActivity时，SecondActivity实例都被迫出栈，下次等FirstActivity跳转到SecondActivity时，找不到存在的SecondActivity实例，于是必须生成新的实例。但是如果我们有ThirdActivity，让SecondActivity和ThirdActivity互相跳转，那么SecondActivity实例就可以保证唯一。

这就是singleTask模式，如果发现有对应的Activity实例，则使此Activity实例之上的其他Activity实例统统出栈，使此Activity实例成为栈顶对象，显示到幕前。

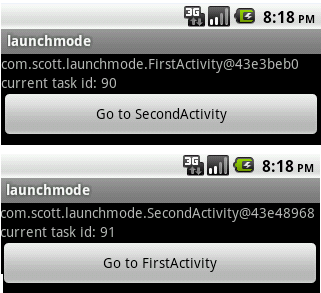
#### 8.4 singleInstance

这种启动模式比较特殊，因为它会启用一个新的栈结构，将Activity放置于这个新的栈结构中，并保证不再有其他Activity实例进入。

我们修改FirstActivity的launchMode="standard"，SecondActivity的launchMode="singleInstance"，由于涉及到了多个栈结构，我们需要在每个Activity中显示当前栈结构的id，所以，我们为每个Activity添加如下代码：

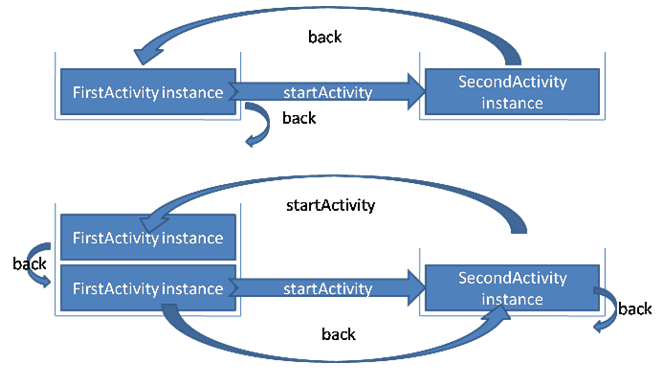
TextView taskIdView = (TextView) findViewById(R.id.taskIdView);   
taskIdView.setText("current task id: " + this.getTaskId());

然后我们再演示一下这个流程：



img

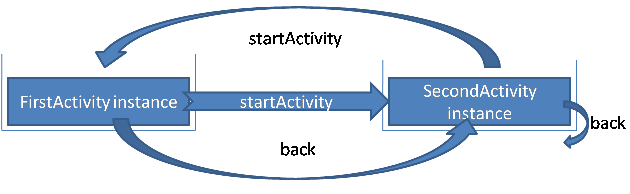
我们发现这两个Activity实例分别被放置在不同的栈结构中，关于singleInstance的原理图如下：



img

我们看到从FirstActivity跳转到SecondActivity时，重新启用了一个新的栈结构，来放置SecondActivity实例，然后按下后退键，再次回到原始栈结构；图中下半部分显示的在SecondActivity中再次跳转到FirstActivity，这个时候系统会在原始栈结构中生成一个FirstActivity实例，然后回退两次，注意，并没有退出，而是回到了SecondActivity，为什么呢？是因为从SecondActivity跳转到FirstActivity的时候，我们的起点变成了SecondActivity实例所在的栈结构，这样一来，我们需要“回归”到这个栈结构。

如果我们修改FirstActivity的launchMode值为singleTop、singleTask、singleInstance中的任意一个，流程将会如图所示：



img

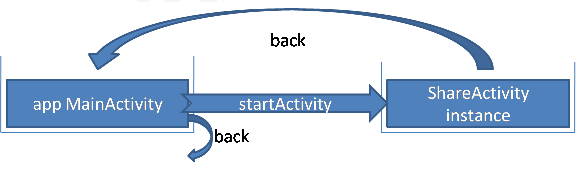
singleInstance启动模式可能是最复杂的一种模式，为了帮助大家理解，我举一个例子，假如我们有一个share应用，其中的ShareActivity是入口Activity，也是可供其他应用调用的Activity，我们把这个Activity的启动模式设置为singleInstance，然后在其他应用中调用。我们编辑ShareActivity的配置：

<activity  
 android:name=".ShareActivity"  
 android:launchMode="singleInstance" >  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 <intent-filter>  
 <action android:name="android.intent.action.SINGLE\_INSTANCE\_SHARE" />  
 <category android:name="android.intent.category.DEFAULT" />  
 </intent-filter>  
</activity>

然后我们在其他应用中这样启动该Activity：

Intent intent = new Intent("android.intent.action.SINGLE\_INSTANCE\_SHARE");   
startActivity(intent);

当我们打开ShareActivity后再按后退键回到原来界面时，ShareActivity做为一个独立的个体存在，如果这时我们打开share应用，无需创建新的ShareActivity实例即可看到结果，因为系统会自动查找，存在则直接利用。大家可以在ShareActivity中打印一下taskId，看看效果。关于这个过程，原理图如下：



img

## Service

### 1.Service是否在main thread中执行, service里面是否能执行耗时的操作?

默认情况,如果没有显示的指servic所运行的进程, Service和activity是运行在当前app所在进程的main thread(UI主线程)里面。

service里面不能执行耗时的操作(网络请求，拷贝数据库，大文件 )。

特殊情况 ,可以在清单文件配置 service 执行所在的进程，让service在另外的进程中执行。

<service  
 android:name="com.baidu.location.f"  
 android:enabled="true"  
 android:process=":remote" >  
</service>

### 2.Activity怎么和Service绑定，怎么在Activity中启动自己对应的Service？

Activity通过bindService(Intent service, ServiceConnection conn,int flags)跟Service进行绑定，当绑定成功的时候Service会将代理对象通过回调的形式传给conn，这样我们就拿到了Service提供的服务代理对象。

在Activity中可以通过startService和bindService方法启动Service。一般情况下如果想获取Service的服务对象那么肯定需要通过bindService()方法，比如音乐播放器，第三方支付等。如果仅仅只是为了开启一个后台任务那么可以使用startService()方法。

### 3.请描述一下Service的生命周期

service有绑定模式和非绑定模式，以及这两种模式的混合使用方式。不同的使用方法生命周期方法也不同。

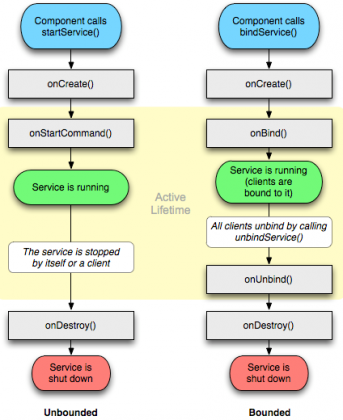
非绑定模式：当第一次调用startService的时候执行的方法依次为onCreate()、onStartCommand()，当Service关闭的时候调用onDestory方法。

绑定模式：第一次bindService（）的时候，执行的方法为onCreate()、onBind(）解除绑定的时候会执行onUnbind()、onDestory()。

上面的两种生命周期是在相对单纯的模式下的情形。我们在开发的过程中还必须注意Service实例只会有一个，也就是说如果当前要启动的Service已经存在了那么就不会再次创建该Service当然也不会调用onCreate（）方法。

一个Service可以被多个客户进行绑定，只有所有的绑定对象都执行了onBind（）方法后该Service才会销毁，不过如果有一个客户执行了onStart()方法，那么这个时候如果所有的bind客户都执行了unBind()该Service也不会销毁。

Service的生命周期图如下所示，帮助大家记忆。



img

### 4.什么是IntentService？有何优点？

我们通常只会使用Service，可能IntentService对大部分同学来说都是第一次听说。那么看了下面的介绍相信你就不再陌生了。如果你还是不了解那么在面试的时候你就坦诚说没用过或者不了解等。并不是所有的问题都需要回答上来的。

#### IntentService简介

IntentService是Service的子类，比普通的Service增加了额外的功能。先看Service本身存在两个问题：

service不会专门启动一条单独的进程，Service与它所在应用位于同一个进程中；

service也不是专门一条新线程，因此不应该在Service中直接处理耗时的任务；

#### IntentService特征

会创建独立的worker线程来处理所有的Intent请求；

会创建独立的worker线程来处理onHandleIntent()方法实现的代码，无需处理多线程问题；

所有请求处理完成后，IntentService会自动停止，无需调用stopSelf()方法停止Service；

为Service的onBind()提供默认实现，返回null；

为Service的onStartCommand提供默认实现，将请求Intent添加到队列中；

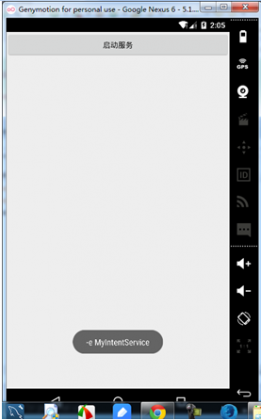
#### 使用IntentService

本人写了一个IntentService的使用例子供参考。该例子中一个MainActivity一个MyIntentService，这两个类都是四大组件当然需要在清单文件中注册。这里只给出核心代码：

MainActivity.java

public void click(View view){  
 Intent intent = new Intent(this, MyIntentService.class);  
 intent.putExtra("start", "MyIntentService");  
 startService(intent);  
}  
  
 MyIntentService.javapublic class MyIntentService extends IntentService {  
 private String ex = "";  
 private Handler mHandler = new Handler() {  
 public void handleMessage(android.os.Message msg) {  
 Toast.makeText(MyIntentService.this, "-e " + ex, Toast.LENGTH\_LONG).show();  
 }  
 };  
 public MyIntentService(){  
 super("MyIntentService");  
 }  
 @Override  
 public int onStartCommand(Intent intent, int flags, int startId) {  
 ex = intent.getStringExtra("start");  
 return super.onStartCommand(intent, flags, startId);  
 }  
 @Override  
 protected void onHandleIntent(Intent intent) {  
 /  
 \* 模拟执行耗时任务  
 \* 该方法是在子线程中执行的，因此需要用到handler跟主线程进行通信  
 \*/  
 try {  
 Thread.sleep(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 mHandler.sendEmptyMessage(0);  
 try {  
 Thread.sleep(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

运行后效果如下：



img

### 5.说说Activity、Intent、Service是什么关系

他们都是Android开发中使用频率最高的类。其中Activity和Service都是Android四大组件之一。他俩都是Context类的子类ContextWrapper的子类，因此他俩可以算是兄弟关系吧。不过兄弟俩各有各自的本领，Activity负责用户界面的显示和交互，Service负责后台任务的处理。Activity和Service之间可以通过Intent传递数据，因此可以把Intent看作是通信使者。

### 6.Service和Activity在同一个线程吗

对于同一app来说默认情况下是在同一个线程中的，mainThread （UI Thread）。

### 7.Service里面可以弹吐司么

可以的。弹吐司有个条件就是得有一个Context上下文，而Service本身就是Context的子类，因此在Service里面弹吐司是完全可以的。比如我们在Service中完成下载任务后可以弹一个吐司通知用户。

## BroadCastReceiver

### 1.请描述一下BroadcastReceiver

BroadCastReceiver是Android四大组件之一，主要用于接收系统或者app发送的广播事件。广播分两种：有序广播和无序广播。

内部通信实现机制：通过Android系统的Binder机制实现通信。

无序广播：完全异步，逻辑上可以被任何广播接收者接收到。优点是效率较高。缺点是一个接收者不能将处理结果传递给下一个接收者，并无法终止广播intent的传播。

有序广播：按照被接收者的优先级顺序，在被接收者中依次传播。比如有三个广播接收者A，B，C，优先级是A > B> C。那这个消息先传给A，再传给B，最后传给C。每个接收者有权终止广播，比如B终止广播，C就无法接收到。此外A接收到广播后可以对结果对象进行操作，当广播传给B时，B可以从结果对象中取得A存入的数据。在通过Context.sendOrderedBroadcast(intent, receiverPermission,resultReceiver, scheduler, initialCode, initialData, initialExtras)时我们可以指定resultReceiver广播接收者，这个接收者我们可以认为是最终接收者，通常情况下如果比他优先级更高的接收者如果没有终止广播，那么他的onReceive会被执行两次，第一次是正常的按照优先级顺序执行，第二次是作为最终接收者接收。如果比他优先级高的接收者终止了广播，那么他依然能接收到广播。

在我们的项目中经常使用广播接收者接收系统通知，比如开机启动、sd挂载、低电量、外播电话、锁屏等。

如果我们做的是播放器，那么监听到用户锁屏后我们应该将我们的播放之暂停等。

### 2.在manifest和代码中如何注册和使用BroadcastReceiver

在清单文件中注册广播接收者称为静态注册，在代码中注册称为动态注册。静态注册的广播接收者只要app在系统中运行则一直可以接收到广播消息，动态注册的广播接收者当注册的Activity或者Service销毁了那么就接收不到广播了。

静态注册：在清单文件中进行如下配置

<receiver android:name=".BroadcastReceiver1" >  
 <intent-filter>  
 <action android:name="android.intent.action.CALL" >  
 </action>  
 </intent-filter>  
</receiver>

动态注册：在代码中进行如下注册

receiver = new BroadcastReceiver();   
IntentFilter intentFilter = new IntentFilter();   
intentFilter.addAction(CALL\_ACTION);   
context.registerReceiver(receiver, intentFilter);

### 3.BroadCastReceiver的生命周期

1. 广播接收者的生命周期非常短暂的，在接收到广播的时候创建，onReceive()方法结束之后销毁；
2. 广播接收者中不要做一些耗时的工作，否则会弹出Application No Response错误对话框；
3. 最好也不要在广播接收者中创建子线程做耗时的工作，因为广播接收者被销毁后进程就成为了空进程，很容易被系统杀掉；
4. 耗时的较长的工作最好放在服务中完成；

## ContentProvider

### 1.请介绍下ContentProvider是如何实现数据共享的

在Android中如果想将自己应用的数据（一般多为数据库中的数据）提供给第三发应用，那么我们只能通过ContentProvider来实现了。

ContentProvider是应用程序之间共享数据的接口。使用的时候首先自定义一个类继承ContentProvider，然后覆写query、insert、update、delete等方法。因为其是四大组件之一因此必须在AndroidManifest文件中进行注册。

<provider  
 android:name="com.jackchan.contenProvider.provider.PersonContentProvider"  
 android:authorities="com.itheima.person"  
 android:exported="true"/>

第三方可以通过ContentResolver来访问该Provider。

### 2.请介绍下Android的数据存储方式

* File存储
* SharedPreference存储
* ContentProvider存储
* SQLiteDataBase存储
* 网络存储

### 3.为什么要用ContentProvider？它和sql的实现上有什么差别？

ContentProvider屏蔽了数据存储的细节,内部实现对用户完全透明,用户只需要关心操作数据的uri就可以了，ContentProvider可以实现不同app之间共享。

Sql也有增删改查的方法，但是sql只能查询本应用下的数据库。而ContentProvider 还可以去增删改查本地文件. xml文件的读取等。

### 4、说说ContentProvider、ContentResolver、ContentObserver之间的关系

* ContentProvider 内容提供者，用于对外提供数据
* ContentResolver.notifyChange(uri)发出消息
* ContentResolver 内容解析者，用于获取内容提供者提供的数据
* ContentObserver 内容监听器，可以监听数据的改变状态
* ContentResolver.registerContentObserver()监听消息

## Android中的布局

### 1.Android中常用的布局都有哪些

* FrameLayout
* RelativeLayout
* LinearLayout
* AbsoluteLayout
* TableLayout
* GridLayout

### 2.谈谈UI中， Padding和Margin有什么区别？

android:padding和android:layout\_margin的区别，其实概念很简单，padding是站在父view的角度描述问题，它规定它里面的内容必须与这个父view边界的距离。margin则是站在自己的角度描述问题，规定自己和其他（上下左右）的view之间的距离，如果同一级只有一个view，那么它的效果基本上就和padding一样了。

## ListView

### 1.ListView如何提高其效率？

* 复用ConvertView
* 自定义静态类ViewHolder
* 使用分页加载
* 使用WeakRefrence引用ImageView对象

### 2.当ListView数据集改变后，如何更新ListView

使用该ListView的adapter的notifyDataSetChanged()方法。该方法会使ListView重新绘制。

### 3.ListView如何实现分页加载

设置ListView的滚动监听器：setOnScrollListener(new OnScrollListener{….})。在监听器中有两个方法： 滚动状态发生变化的方法(onScrollStateChanged)和listView被滚动时调用的方法(onScroll)

在滚动状态发生改变的方法中，有三种状态：

* 手指按下移动的状态：SCROLL\_STATE\_TOUCH\_SCROLL: // 触摸滑动
* 惯性滚动（滑翔（flgin）状态）：SCROLL\_STATE\_FLING: // 滑翔
* 静止状态：SCROLL\_STATE\_IDLE: // 静止

对不同的状态进行处理：

分批加载数据，只关心静止状态：关心最后一个可见的条目，如果最后一个可见条目就是数据适配器（集合）里的最后一个，此时可加载更多的数据。在每次加载的时候，计算出滚动的数量，当滚动的数量大于等于总数量的时候，可以提示用户无更多数据了。

### 4.ListView可以显示多种类型的条目吗

这个当然可以的，ListView显示的每个条目都是通过baseAdapter的getView(int position, View convertView, ViewGroup parent)来展示的，理论上我们完全可以让每个条目都是不同类型的view，除此之外adapter还提供了getViewTypeCount()和getItemViewType(int position)两个方法。在getView方法中我们可以根据不同的viewtype加载不同的布局文件。

### 5.ListView如何定位到指定位置

可以通过ListView提供的lv.setSelection(48);方法。

### 6.当在ScrollView中如何嵌入ListView

通常情况下我们不会在ScrollView中嵌套ListView，但是如果面试官非让我嵌套的话也是可以的。

在ScrollView添加一个ListView会导致listview控件显示不全，通常只会显示一条，这是因为两个控件的滚动事件冲突导致。所以需要通过listview中的item数量去计算listview的显示高度，从而使其完整展示，如下提供一个方法供大家参考。

lv = (ListView) findViewById(R.id.lv);  
 adapter = new MyAdapter();  
 lv.setAdapter(adapter);  
 setListViewHeightBasedOnChildren(lv);  
public void setListViewHeightBasedOnChildren(ListView listView) {  
 ListAdapter listAdapter = listView.getAdapter();  
 if (listAdapter == null) {  
 return;  
 }  
 int totalHeight = 0;  
 for (int i = 0; i < listAdapter.getCount(); i++) {  
 View listItem = listAdapter.getView(i, null, listView);  
 listItem.measure(0, 0);  
 totalHeight += listItem.getMeasuredHeight();  
 }  
 ViewGroup.LayoutParams params = listView.getLayoutParams();  
 params.height = totalHeight + (listView.getDividerHeight() \* (listAdapter.getCount() - 1));  
 params.height += 5;// if without this statement,the listview will be a little short  
 listView.setLayoutParams(params);  
}

### 7.ListView中如何优化图片

图片的优化策略比较多。

#### 处理图片的方式：

如果ListView中自定义的Item中有涉及到大量图片的，一定要对图片进行细心的处理，因为图片占的内存是ListView项中最头疼的，处理图片的方法大致有以下几种：

* 不要直接拿路径就去循环BitmapFactory.decodeFile;使用Options保存图片大小、不要加载图片到内存去。
* 对图片一定要经过边界压缩尤其是比较大的图片，如果你的图片是后台服务器处理好的那就不需要了
* 在ListView中取图片时也不要直接拿个路径去取图片，而是以WeakReference（使用WeakReference代替强引用。比如可以使用WeakReference mContextRef）、SoftReference、WeakHashMap等的来存储图片信息。
* 在getView中做图片转换时，产生的中间变量一定及时释放

#### 异步加载图片基本思想：

* 先从内存缓存中获取图片显示（内存缓冲）
* 获取不到的话从SD卡里获取（SD卡缓冲）
* 都获取不到的话从网络下载图片并保存到SD卡同时加入内存并显示（视情况看是否要显示）

原理：

优化一：先从内存中加载，没有则开启线程从SD卡或网络中获取，这里注意从SD卡获取图片是放在子线程里执行的，否则快速滑屏的话会不够流畅。

优化二：于此同时，在adapter里有个busy变量，表示listview是否处于滑动状态，如果是滑动状态则仅从内存中获取图片，没有的话无需再开启线程去外存或网络获取图片。  
优化三：ImageLoader里的线程使用了线程池，从而避免了过多线程频繁创建和销毁，如果每次总是new一个线程去执行这是非常不可取的，好一点的用的AsyncTask类，其实内部也是用到了线程池。在从网络获取图片时，先是将其保存到sd卡，然后再加载到内存，这么做的好处是在加载到内存时可以做个压缩处理，以减少图片所占内存。

### 8.ListView中图片错位的问题是如何产生的

图片错位问题的本质源于我们的listview使用了缓存convertView，假设一种场景，一个listview一屏显示九个item，那么在拉出第十个item的时候，事实上该item是重复使用了第一个item，也就是说在第一个item从网络中下载图片并最终要显示的时候，其实该item已经不在当前显示区域内了，此时显示的后果将可能在第十个item上输出图像，这就导致了图片错位的问题。所以解决之道在于可见则显示，不可见则不显示。

### 9.Java中引用类型都有哪些

Java中对象的引用分为四种级别，这四种级别由高到低依次为：强引用、软引用、弱引用和虚引用。

**强引用（StrongReference）**

这个就不多说，我们写代码天天在用的就是强引用。如果一个对象被被人拥有强引用，那么垃圾回收器绝不会回收它。当内存空间不足，Java虚拟机宁愿抛出OutOfMemoryError错误，使程序异常终止，也不会靠随意回收具有强引用的对象来解决内存不足问题。

Java的对象是位于heap中的，heap中对象有强可及对象、软可及对象、弱可及对象、虚可及对象和不可到达对象。应用的强弱顺序是强、软、弱、和虚。对于对象是属于哪种可及的对象，由他的最强的引用决定。如下代码：

String abc=new String("abc"); //1   
SoftReference<String> softRef=new SoftReference<String>(abc); //2   
WeakReference<String> weakRef = new WeakReference<String>(abc); //3   
abc=null; //4   
softRef.clear();//5

第一行在heap堆中创建内容为“abc”的对象，并建立abc到该对象的强引用,该对象是强可及的。  
第二行和第三行分别建立对heap中对象的软引用和弱引用，此时heap中的abc对象已经有3个引用，显然此时abc对象仍是强可及的。 第四行之后heap中对象不再是强可及的，变成软可及的。 第五行执行之后变成弱可及的。

**软引用（SoftReference）**

如果一个对象只具有软引用，那么如果内存空间足够，垃圾回收器就不会回收它，如果内存空间不足了，就会回收这些对象的内存。只要垃圾回收器没有回收它，该对象就可以被程序使用。软引用可用来实现内存敏感的高速缓存。

软引用可以和一个引用队列（ReferenceQueue）联合使用，如果软引用所引用的对象被垃圾回收，Java虚拟机就会把这个软引用加入到与之关联的引用队列中。软引用是主要用于内存敏感的高速缓存。在jvm报告内存不足之前会清除所有的软引用，这样以来gc就有可能收集软可及的对象，可能解决内存吃紧问题，避免内存溢出。什么时候会被收集

取决于gc的算法和gc运行时可用内存的大小。当gc决定要收集软引用时执行以下过程,以上面的softRef为例：

* 首先将softRef的referent（abc）设置为null，不再引用heap中的new String("abc")对象。
* 将heap中的newString("abc")对象设置为可结束的(finalizable)。
* 当heap中的new String("abc")对象的finalize()方法被运行而且该对象占用的内存被释放， softRef被添加到它的ReferenceQueue(如果有的话)中。

注意：对ReferenceQueue软引用和弱引用可以有可无，但是虚引用必须有。被 SoftReference 指到的对象，即使没有任何 Direct Reference，也不会被清除。一直要到 JVM 内存不足且没有Direct Reference 时才会清除，SoftReference 是用来设计 object-cache 之用的。如此一来 SoftReference 不但可以把对象 cache 起来，也不会造成内存不足的错误（OutOfMemoryError）。file:///C:/Users/ADMINI~1/AppData/Local/Temp/msohtmlclip1/01/clip\_image002.jpg

**弱引用（WeakReference）**

如果一个对象只具有弱引用，那该类就是可有可无的对象，因为只要该对象被gc扫描到了随时都会把它干掉。弱引用与软引用的区别在于：只具有弱引用的对象拥有更短暂的生命周期。在垃圾回收器线程扫描它所管辖的内存区域的过程中，一旦发现了只具有弱引用的对象，不管当前内存空间足够与否，都会回收它的内存。不过，由于垃圾回收器是一个优先级很低的线程，因此不一定会很快发现那些只具有弱引用的对象。弱引用可以和一个引用队列（ReferenceQueue）联合使用，如果弱引用所引用的对象被垃圾回收，Java虚拟机就会把这个弱引用加入到与之关联的引用队列中。

**虚引用（PhantomReference）**

"虚引用"顾名思义，就是形同虚设，与其他几种引用都不同，虚引用并不会决定对象的生命周期。如果一个对象仅持有虚引用，那么它就和没有任何引用一样，在任何时候都可能被垃圾回收。虚引用主要用来跟踪对象被垃圾回收的活动。

虚引用与软引用和弱引用的一个区别在于：虚引用必须和引用队列（ReferenceQueue）联合使用。当垃圾回收器准备回收一个对象时，如果发现它还有虚引用，就会在回收对象的内存之前，把这个虚引用加入到与之关联的引用队列中。程序可以通过判断引用队列中是否已经加入了虚引用，来了解被引用的对象是否将要被垃圾回收。程序如果发现某个虚引用已经被加入到引用队列，那么就可以在所引用的对象的内存被回收之前采取必要的行动。

建立虚引用之后通过get方法返回结果始终为null,通过源代码你会发现,虚引用通向会把引用的对象写进referent,只是get方法返回结果为null。先看一下和gc交互的过程再说一下他的作用。

1.不把referent设置为null, 直接把heap中的newString("abc")对象设置为可结束的(finalizable)。

2.与软引用和弱引用不同, 先把PhantomRefrence对象添加到它的ReferenceQueue中.然后在释放虚可及的对象。

## JNI&NDK

1.在Android中如何调用C语言

当我们的Java需要调用C语言的时候可以通过JNI的方式，Java Native Interface。Android提供了对JNI的支持，因此我们在Android中可以使用JNI调用C语言。在Android开发目录的libs目录下添加xxx.so文件，不过xxx.so文件需要放在对应的CPU架构名目录下，比如armeabi，x86等。

在Java代码需要通过System.*loadLibrary*(libName);加载so文件。同时C语言中的方法在java中必须以native关键字来声明。普通Java方法调用这个native方法接口，虚拟机内部自动调用so文件中对应的方法。

2.请介绍一下NDK

1.NDK 是一系列工具的集合

NDK 提供了一系列的工具，帮助开发者快速开发C（或C++）的动态库，并能自动将so 和java 应用一起打包成apk。NDK 集成了交叉编译器，并提供了相应的mk 文件隔离CPU、平台、ABI 等差异，开发人员只需要简单修改mk 文件（指出“哪些文件需要编译”、“编译特性要求”等），就可以创建出so。

2.NDK 提供了一份稳定、功能有限的API 头文件声明  
Google 明确声明该API 是稳定的，在后续所有版本中都稳定支持当前发布的API。从该版本的NDK 中看出，这些API支持的功能非常有限，包含有：C 标准库（libc）、标准数学库（libm）、压缩库（libz）、Log 库（liblog）。

3、JNI调用常用的两个参数 JNIEnv\*env, jobject obj

第一个是指向虚拟机对象的指针，是一个二级指针。里面封装了很多方法和中间变量供我们使用。

第二个代表着调用该方法的Java对象的C语言表示。

## Android中的网络访问

### 1、Android中如何访问网络

Android提供了org.apache.http.HttpClientConnection和java.net.HttpURLConnection两个连接网络对象。使用哪个都行，具体要看企业领导的要求了。 除此之外一般我比较喜欢使用xUtils中的HttpUtils功能，该模块底层使用的就是org.apache.http.client.HttpClient，使用起来非常方便。

### 2、如何解析服务器传来的JSON文件

在Android中内置了JSON的解析API，在org.json包中包含了如下几个类：JSONArray、JSONObject、JSONStringer、JSONTokener和一个异常类JSONException。

1、JSON解析步骤

1）读取网络文件数据并转为一个json字符串

InputStreamin = conn.getInputStream();   
StringjsonStr = DataUtil.Stream2String(in);//将流转换成字符串的工具类

2）将字符串传入相应的JSON构造函数中

* 通过构造函数将json字符串转换成json对象
* JSONObject jsonObject = new JSONObject(jsonStr)；
* 通过构造函数将json字符串转换成json数组：

JSONArray array = new JSONArray(jsonStr);

3）解析出JSON中的数据信息：

* 从json对象中获取你所需要的键所对应的值
* JSONObject json=jsonObject.getJSONObject("weatherinfo");   
  Stringcity = json.getString("city");   
  Stringtemp = json.getString("temp")
* 遍历JSON数组，获取数组中每一个json对象，同时可以获取json对象中键对应的值
* for(int i = 0; i < array.length(); i++) {   
  JSONObject obj = array.getJSONObject(i);   
  Stringtitle=obj.getString("title");   
  Stringdescription=obj.getString("description");   
  }
* 2、生成JSON对象和数组

1）生成JSON：

方法1、创建一个map，通过构造方法将map转换成json对象

Map<String,Object> map = new HashMap<String, Object>();   
map.put("name","zhangsan");   
map.put("age",24);   
JSONObjectjson = new JSONObject(map);

方法2、创建一个json对象，通过put方法添加数据

JSONObjectjson=new JSONObject();   
json.put("name","zhangsan");   
json.put("age",24);

2）生成JSON数组：

创建一个list，通过构造方法将list转换成json对象

Map<String,Object> map1 = new HashMap<String, Object>();   
map1.put("name","zhangsan");   
map1.put("age",24);   
Map<String,Object> map2 = new HashMap<String, Object>();   
map2.put("name","lisi");   
map2.put("age",25);   
List<Map<String,Object>> list=new ArrayList<Map<String,Object>>();   
list.add(map1);   
list.add(map2);   
JSONArrayarray=new JSONArray(list);   
System.out.println(array.toString());

### 3、如何解析服务器传来的XML格式数据

Android为我们提供了原生的XML解析和生成支持。

1、XML解析

* 获取解析器: Xml.newPullParser()
* 设置输入流: setInput()
* 获取当前事件类型: getEventType()
* 解析下一个事件, 获取类型: next()
* 获取标签名: getName()
* 获取属性值: getAttributeValue()
* 获取下一个文本: nextText()
* 获取当前文本: getText()

5种事件类型: START\_DOCUMENT, END\_DOCUMENT, START\_TAG,END\_TAG, TEXT

示例代码：

public List<Person>getPersons(InuptStream in){   
 XmlPullParserparser=Xml.newPullParser();//获取解析器  
 parser.setInput(in,"utf-8");  
 for(inttype=){ //循环解析  
 }   
}

2、XML生成

* 获取生成工具: Xml.newSerializer()
* 设置输出流: setOutput()
* 开始文档: startDocument()
* 结束文档: endDocument()
* 开始标签: startTag()
* 结束标签: endTag()
* 属性: attribute()
* 文本: text()

示例代码：

XmlSerializer serial=Xml.newSerializer();//获取xml序列化工具  
serial.setOuput(put,"utf-8");  
serial.startDocument("utf-8",true);  
serial.startTag(null,"persons");  
for(Person p:persons){  
 serial.startTag(null,"persons");   
 serial.attribute(null,"id",p.getId().toString());  
 serial.startTag(null,"name");   
 serial.attribute(null,"name",p.getName().toString());  
 serial.endTag(null,"name");  
 serial.startTag(null,"age");   
 serial.attribute(null,"age",p.getAge().toString());  
 serial.endTag(null,"age");  
 serial.endTag(null,"persons");   
}

### 4、如何从网络上加载一个图片显示到界面

可以通过BitmapFactory.decodeStream(inputStream);方法将图片转换为bitmap，然后通过imageView.setImageBitmap(bitmap);将该图片设置到ImageView中。这是原生的方法，还可以使用第三方开源的工具来实现，比如使用SmartImageView作为ImageView控件，然后直接设置一个url地址即可。也可以使用xUtils中的BitmapUtils工具。

### 5、如何播放网络视频

除了使用Android提供的MediaPlayer和VideoView外通常还可以使用第三方开源万能播放器，VitamioPlayer。该播放器兼容性好，支持几乎所有主流视频格式。

## Intent

### 1、Intent传递数据时，可以传递哪些类型数据？

Intent可以传递的数据类型非常的丰富，java的基本数据类型和String以及他们的数组形式都可以，除此之外还可以传递实现了Serializable和Parcelable接口的对象。

### 2、Serializable和Parcelable的区别

* 在使用内存的时候，Parcelable 类比Serializable性能高，所以推荐使用Parcelable类。
* Serializable在序列化的时候会产生大量的临时变量，从而引起频繁的GC。
* Parcelable不能使用在要将数据存储在磁盘上的情况。尽管Serializable效率低点，但在这种情况下，还是建议你用Serializable 。
* 实现：

1、Serializable 的实现，只需要继承Serializable 即可。这只是给对象打了一个标记，系统会自动将其序列化。

2、Parcelabel 的实现，需要在类中添加一个静态成员变量 CREATOR，这个变量需要继承 Parcelable.Creator 接口。

public class MyParcelable implements Parcelable {  
 private int mData;  
 public int describeContents() {  
 return 0;  
 }  
 public void writeToParcel(Parcel out, int flags) {  
 out.writeInt(mData);  
 }  
 public static final Parcelable.Creator<MyParcelable> CREATOR  
 = new Parcelable.Creator<MyParcelable>() {  
 public MyParcelable createFromParcel(Parcel in) {  
 return new MyParcelable(in);  
 }  
 public MyParcelable[] newArray(int size) {  
 return new MyParcelable[size];  
 }  
 };  
 private MyParcelable(Parcel in) {  
 mData = in.readInt();  
 }  
}

### 3、请描述一下Intent 和 IntentFilter

Android 中通过 Intent 对象来表示一条消息，一个 Intent对象不仅包含有这个消息的目的地，还可以包含消息的内容，这好比一封 Email，其中不仅应该包含收件地址，还可以包含具体的内容。对于一个 Intent 对象，消息“目的地”是必须的，而内容则是可选项。  
通过Intent 可以实现各种系统组件的调用与激活。  
IntentFilter: 可以理解为邮局或者是一个信笺的分拣系统… 这个分拣系统通过3个参数来识别

* Action: 动作view
* Data: 数据uri uri
* Category : 而外的附加信息

Action 匹配

Action 是一个用户定义的字符串，用于描述一个 Android 应用程序组件，一个 IntentFilter 可以包含多个 Action。在 AndroidManifest.xml 的 Activity 定义时可以在其 <intent-filter >节点指定一个 Action 列表用于标示 Activity 所能接受的“动作”， 例如：

<intent-filter>   
 <action android:name="android.intent.action.MAIN"/>   
 <actionandroid:name="cn.itheima.action" />   
 ……   
</intent-filter>

如果我们在启动一个 Activity 时使用这样的Intent 对象

Intentintent =new Intent();   
intent.setAction("cn.itheima.action");

那么所有的 Action 列表中包含了“cn.itheima”的 Activity 都将会匹配成功。 Android 预定义了一系列的 Action 分别表示特定的系统动作。这些Action 通过常量的方式定义在 android.content. Intent中，以“ACTION\_”开头。我们可以在 Android 提供的文档中找到它们的详细说明。

URI 数据匹配

一个 Intent 可以通过 URI 携带外部数据给目标组件。在 <intent-filter >节点中，通过 <data/>节点匹配外部数据。

mimeType 属性指定携带外部数据的数据类型，scheme 指定协议，host、port、path 指定数据的位置、端口、和路径。如下：

<data android:mimeType="mimeType"  
 android:scheme="scheme"  
 android:host="host"  
 android:port="port"   
 android:path="path"/>

电话的uri tel:12345

自己定义的uri itcast://cn.itcast/person/10  
如果在 Intent Filter 中指定了这些属性，那么只有所有的属性都匹配成功时 URI 数据匹配才会成功。

Category 类别匹配

<intent-filter >节点中可以为组件定义一个 Category 类别列表，当 Intent 中包含这个列表的所有项目时 Category 类别匹配才会成功。

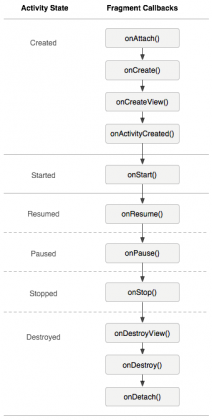
## Fragment1、Fragment跟Activity之间是如何传值的

当Fragment跟Activity绑定之后，在Fragment中可以直接通过getActivity（）方法获取到其绑定的Activity对象，这样就可以调用Activity的方法了。在Activity中可以通过如下方法获取到Fragment实例：

FragmentManager fragmentManager = getFragmentManager();  
Fragment fragment = fragmentManager.findFragmentByTag(tag);  
Fragment fragment = fragmentManager.findFragmentById(id);

获取到Fragment之后就可以调用Fragment的方法。也就实现了通信功能。

2、描述一下Fragment的生命周期



img

## 能否在子线程中更新UI

可以的，在onCreate()的setContentView()后new一个Thread去更新UI是不会报错的，但是延迟1s后再更新UI就会报错，这是因为在onCreate()的时候ViewRoot的requestLayout()方法没有执行，layout布局文件还没有创建完成。而ViewRoot的requestLayout()方法中才会调用checkThread()方法检查当前是否是主线程，不是的话就抛CalledFromWrongThreadException。Android中的定义是：不建议在子线程中更新UI，否则会产生不可预知的错误