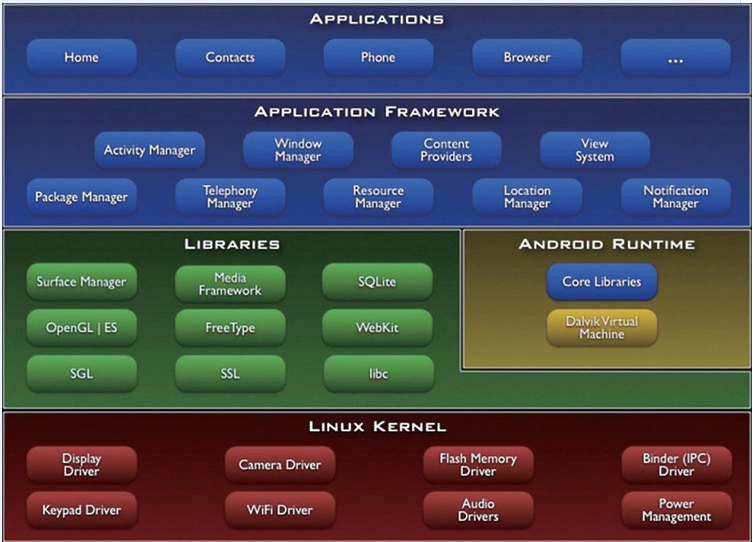


# 一. Android入门

## 1、描述一下android的系统架构



android系统架构分从下往上为linux内核层、运行库、应用程序框架层、和应用程序层。

|  |  |
| --- | --- |
| 应用程序层 | android应用程序使用框架的api并在框架下运行，这就带来了程序开发的高度一致性，另一方面也告诉我们，要想写出优质高效的程序就必须对整个 applicationframework进行非常深入的理解。精通applicationframework，你就可以真正的理解android的设计和运行机制，也就更能够驾驭整个应用层的开发 |
| Application framework | java应用程序开发人员主要是使用该层封装好的api进行快速开发。applications:该层是java的应用程序层，android内置googlemaps、e-mail、即时通信工具、浏览器、mp3播放器等处于该层，java开发人员开发的程序也处于该层，而且和内置的应用程序具有平等的位置，可以调用内置的应用程序，也可以替换内置的应用程序。 |
| Libraries  android Runtime | 即c/c++函数库部分，大多数都是开放源代码的函数库，例如webkit（引擎），该函数库负责 android网页浏览器的运行，例如标准的c函数库libc、openssl、sqlite等，当然也包括支持游戏开发2dsgl和 3dopengles，在多媒体方面有mediaframework框架来支持各种影音和图形文件的播放与显示。 |
| 负责解释和执行生成的dalvik格式的字节码 |
| Linux kernel | 负责硬件的驱动程序、网络、电源、系统安全以及内存管理等功能 |

总结：

下层为上层服务，上层需要下层的支持，调用下层的服务，这种严格分层的方式带来的极大的稳定性、灵活性和可扩展性，使得不同层的开发人员可以按照规范专心特定层的开发。

## 2、Dalvik和标准Java虚拟机之间的主要差别？

|  |  |  |
| --- | --- | --- |
|  | Dalvik | JVM |
| 数据存储处理 | 基于寄存器 | 基于栈 |
| 运行环境 | 同时运行多个Dalvik实例 | 只能运行一个JVM实例 |
| 运行文件格式 | dex文件 | class文件 |
| 效率 | 较高 | 较低 |

## 3、Android平台手机 5大优势：

**1、Android价格占优 价廉性能并不低**

消费者选择产品，价格是必然要考虑的一大因素，iphone虽好，但是价格让一般人望而却步。苹果就像是宝马、奔驰，虽然大家都认为它很好，但是一般人消费不起，只有看的份。而Android，如同大众，满大街跑的都是，不仅如此，还有一些型号是可以与宝马、奔驰相媲美的。

虽然Android平台的手机价廉，但是其性能却一点也不低廉，触摸效果并不比苹果差到哪里去。Android平台简单实用，无论是功能还是外观设计，都可以与苹果一决高下。当消费者考虑价格因素之后，在数量众多的android手机中，消费者总是会找到一款满意的Android手机取代价格高昂的iPhone。

**2、应用程序发展迅速**

智能机玩的就是个应用，Android平台上的应用有上百万个, 来自Android应用商店最大的优势是，不对应用程序进行严格的审查。在这一点上优于苹果。

**3、智能手机厂家助力**

苹果的自我中心是它成功的一大法宝，从硬件到软件，到其独特的推广方式。苹果形成了一个很好的很完整的产业链。在一个封闭的圈子中创造一个又一个的奇想，让消费者的体验得到了很大的满足。但是这样并不能使其长期占据有利位置。

现在，世界很多智能手机厂家几乎都加入了Android阵营，并推出了一系列的Android智能机。摩托罗拉、三星、HTC、LG、Lumigon等厂家都与谷歌建立了Android平台技术联盟。

**4、运营商的鼎力支持**

在国内，三大运营商是卯足了劲的推出Android智能机。联通的“0元购机”，电信的千元3G，移动的索爱A8i定制机，都显示了运营商对Android智能机的期望。

在美国，T-Mobile、Sprint、AT&T和Verizon全部推出了Android手机。此外，日本的KDDI，NTT DoCoMo，Telecom Italia(意大利电信)、T-Mobile (德国)、Telefónica( 西班牙)等众多运营商都是Android的支持者。

**5、机型多 硬件配置优**

自从谷歌推出Android系统以来，各大厂家纷纷推出自己的Android平台手机，HTC、索尼爱立信、魅族、摩托罗拉、夏普、LG、三星、联想,中兴,华为等等.

**6、系统开源 利于创新**

苹果的自我封闭性，使其创新必须源自内部。而Android是开源的，允许第三方修改，这在很大程度上容许厂家根据自己的硬件更改版本，从而能够更好的适应硬件，与之形成良好的结合。

相比于苹果的封闭，开源能够提供更好的安全性能，也给开发人员提供了一个更大的创新空间，从而使Android版本升级更快。

## 4、Android的不足

　　一、安全和隐私

　　由于手机 与互联网的紧密联系，个人隐私很难得到保守。除了上网过程中经意或不经意留下的个人足迹，Google这个巨人也时时站在你的身后，洞穿一切，因此，互联网的深入将会带来新一轮的隐私危机。

　二、机型太多, 缺少标准配置

在 Android平台中，由于其开放性，有很多的第三方厂商生产了各种型号的手机, 而对于应用来说要适配所有的机型还是挺困难的.

# 二. Android的四大组件

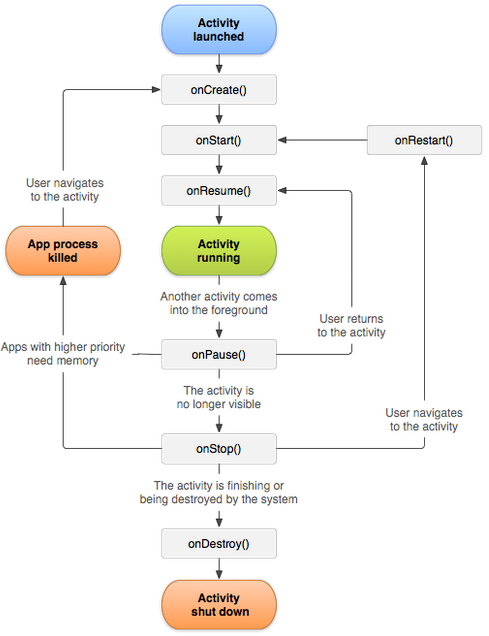
## 2.1. Activity相关

### 5、什么是Activity(Servlet)

Activity是一个负责与**用户交互**的组件，Activity中所有操作都与用户密切相关，可以通过setContentView(View)来**显示指定控件**。

在一个android应用中，一个Activity通常就是一个单独的屏幕，它上面可以显示一些控件也可以监听并处理用户的事件做出响应。

### 6、请描述一下Activity生命周期



onCreate(Bundle savedInstanceState)：

创建activity时调用。设置在该方法中，还以Bundle的形式提供对以前储存的任何状态的访问！

onStart()：

activity变为在屏幕上对用户可见时调用。

onResume()：

activity开始与用户交互时调用（无论是启动还是重新启动一个活动，该方法

总是被调用的）。

onPause()：

activity被暂停或收回cpu和其他资源时调用，该方法用于保存活动状态的，也

是保护现场，压栈吧！

onStop()：

activity被停止并转为不可见阶段及后续的生命周期事件时调用。

onRestart()：

重新启动activity时调用。该活动仍在栈中，而不是启动新的活动。

onDestroy()：

activity被完全从系统内存中移除时调用，该方法被调用

### 7、activity的启动模式有哪些？是什么含义？

答：在android里，有4种activity的启动模式，分别为：

“standard” (默认)

“singleTop”

“singleTask”

“singleInstance”

当应用运行起来后就会开启一条线程，线程中会运行一个任务栈，当Activity实例创建后就会放入任务栈中。Activity启动模式的设置在AndroidManifest.xml文件中，通过配置Activity的属性android:launchMode="…"。

**1. Standared模式（默认）**

每次启动当前的Activity时都会创建其对象, 并将其放到当前Task栈中

**2. SingleTop模式**

如果当前Activity已显示在前台, 启动当前Activity不会创建, 否则会创建

**3. SingleTask模式**

如果当前Acctivity对象已经存在于当前Task栈中(可以不在栈顶), 启动当前Activity不会创建, 否则会创建。

**4. SingleInstance模式**

只要当前Acctivity的对象存在(可以在其它Task栈中), 启动当前Activity不会创建, 否则会创建.

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

在Android中退出程序比较麻烦，尤其是在多个Activity的程序中，

2.2之前:

|  |
| --- |
| ActivityManager am = (ActivityManager)getSystemService (Context.ACTIVITY\_SERVICE);  am.restartPackage(getPackageName()); |

2.2之后上面的方式不可用

|  |
| --- |
| 可以自定义一个Activity的栈，在程序退出时将栈中的所有的Activity进行finish，这种方法，我以前的文章中有详述 |

或

|  |
| --- |
| 先让程序到Home界面，然后再将process杀死  Intent intent = new Intent(Intent.ACTION\_MAIN);  intent.addCategory(Intent.CATEGORY\_HOME);  intent.setFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP);  startActivity(intent);  android.os.Process.killProcess(Process.myPid()); |

### 9、如果后台的Activity由于某原因被系统回收了，如何在被系统回收之前保存当前状态？

答：重写onSaveInstanceState()方法，在此方法中保存需要保存的数据，该方法将会在activity被回收之前调用。通过重写onRestoreInstanceState()方法可以从中提取保存好的数据.

### 10、activity在屏幕旋转时的生命周期

答：1. 不设置Activity的android:configChanges时，切屏会重新调用各个生命周期，切横屏时会执行一次，切竖屏时会执行两次；

2. 设置Activity的android:configChanges="orientation"时，切屏还是会重新调用各个生命周期，切横、竖屏时只会执行一次；

3. 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。

## 2.2. Service相关

### 11、如何开发一个Service组件？

服务的开发比较简单，如下：

第一步：继承Service类 public class SMSService extends Service {}

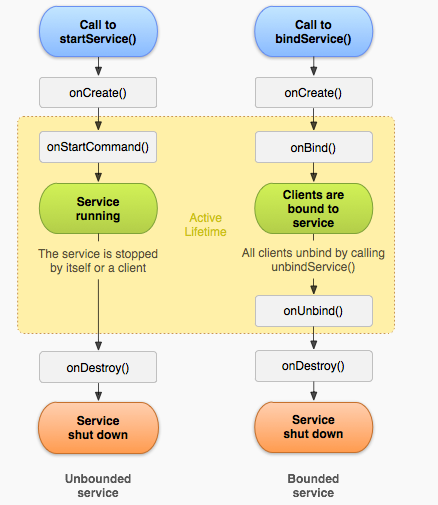
第二步：在AndroidManifest.xml文件中的<application>节点里对服务进行配置:<service android:name=".SMSService" />

第三步：启动服务

方法一：context.startService()：调用者与服务之间没有关连，即使调用者退出了，服务仍然运行

方法二：context.bindService()：调用者与服务绑定在了一起，调用者一旦退出，服务也就终止，大有“不求同时生，必须同时死”的特点

### 12、Service的生命周期？



onCreate()：

该方法在服务被创建时调用，该方法只会被调用一次，无论调用多少次startService()或bindService()方法，服务也只被创建一次。

onDestroy()：

该方法在服务被终止时调用。与采用Context.startService()方法启动服务有关的生命周期方法

onStart()：

只有采用Context.startService()方法启动服务时才会回调该方法。该方法在服务开始运行时被调用。多次调用startService()方法尽管不会多次创建服务，但onStart() 方法会被多次调用。

onBind()：

只有采用Context.bindService()方法启动服务时才会回调该方法。该方法在调用者与服务绑定时被调用，当调用者与服务已经绑定，多次调用Context.bindService()方法并不会导致该方法被多次调用。

onUnbind()：

只有采用Context.bindService()方法启动服务时才会回调该方法。该方法在调用者与服务解除绑定时被调用

### 13、 Service和Thread的区别？

答：servie是系统的组件，它由系统进程托管（service manager）；它们之间的通信类似于client和server，是一种轻量级的ipc通信，这种通信的载体是binder，它是在linux层交换信息的一种ipc。而thread是由本应用程序托管。

1). Thread：Thread 是程序执行的最小单元，它是分配CPU的基本单位。可以用 Thread 来执行一些异步的操作。

2). Service：Service 是android的一种机制，当它运行的时候如果是Local Service，那么对应的 Service 是运行在主进程的 main 线程上的。如：onCreate，onStart 这些函数在被系统调用的时候都是在主进程的 main 线程上运行的。如果是Remote Service，那么对应的 Service 则是运行在独立进程的 main 线程上。

既然这样，那么我们为什么要用 Service 呢？其实这跟 android 的系统机制有关，我们先拿 Thread 来说。Thread 的运行是独立于 Activity 的，也就是说当一个 Activity 被 finish 之后，如果你没有主动停止 Thread 或者 Thread 里的 run 方法没有执行完毕的话，Thread 也会一直执行。因此这里会出现一个问题：当 Activity 被 finish 之后，你不再持有该 Thread 的引用。另一方面，你没有办法在不同的 Activity 中对同一 Thread 进行控制。

举个例子：如果你的 Thread 需要不停地隔一段时间就要连接服务器做某种同步的话，该 Thread 需要在 Activity 没有start的时候也在运行。这个时候当你 start 一个 Activity 就没有办法在该 Activity 里面控制之前创建的 Thread。因此你便需要创建并启动一个 Service ，在 Service 里面创建、运行并控制该 Thread，这样便解决了该问题（因为任何 Activity 都可以控制同一 Service，而系统也只会创建一个对应 Service 的实例）。

因此你可以把 Service 想象成一种消息服务，而你可以在任何有 Context 的地方调用 Context.startService、Context.stopService、Context.bindService，Context.unbindService，来控制它，你也可以在 Service 里注册 BroadcastReceiver，在其他地方通过发送 broadcast 来控制它，当然这些都是 Thread 做不到的。

### 14、简单描述AIDL

答：由于每个应用程序都运行在自己的进程空间，并且可以从应用程序UI运行另一个服务进程，而且经常会在不同的进程间传递对象。在Android平台，一个进程通常不能访问另一个进程的内存空间，所以要想对话，需要将对象分解成操作系统可以理解的基本单元，并且有序的通过进程边界。

通过代码来实现这个数据传输过程是冗长乏味的，Android提供了AIDL工具来处理这项工作。

AIDL (Android Interface Definition Language)是一种IDL 语言，用于生成可以在Android设备上两个进程之间进行进程间通信(IPC)的代码。如果在一个进程中（例如Activity）要调用另一个进程中（例如Service）对象的操作，就可以使用AIDL生成可序列化的参数。

序列化 (将对象转换为0101二进制数据)

B应用 Student对象

A应用(学员管理系统)

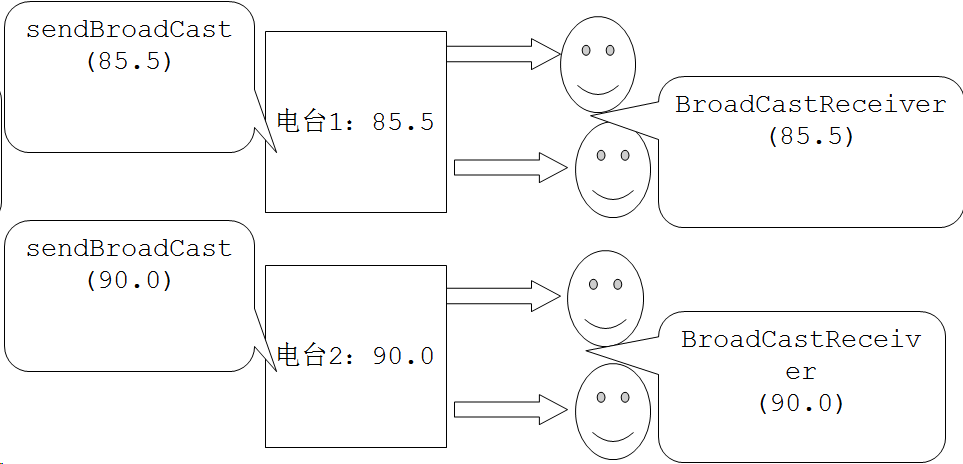
Id=12

Student getStudentById(id)

Student(tom, 12)

## 2.3. BroadCastReceiver相关

### 15、请描述一下Broadcast Receiver。



Broadcast Receiver用于接收并处理广播通知（broadcast announcements）。多数的广播是系统发起的，如来电来信、地域变换、电量不足等。程序也可以播放一个广播。程序可以有任意数量的 broadcast receivers来响应它觉得重要的通知。broadcast receiver可以通过多种方式通知用户：启动activity、使用NotificationManager、开启背景灯、振动设备、播放声音等，最典型的是在状态栏显示一个图标，这样用户就可以点它打开看通知内容。通常我们的某个应用或系统本身在某些事件（电池电量不足、来电来短信）来临时会广播一个Intent出去，我们可以利用注册一个Broadcast Receiver来监听到这些Intent并获取Intent中的数据。

### 16、注册广播有几种方式，这些方式有何优缺点？

答：首先写一个类要继承BroadcastReceiver

第一种:在清单文件中声明,添加

<receive android:name=".IncomingSMSReceiver " >

<intent-filter>

   <action android:name="android.provider.Telephony.SMS\_RECEIVED")

<intent-filter>

<receiver>

第二种在Activity中使用代码进行注册如:

IntentFilter filter =  new IntentFilter("android.provider.Telephony.SMS\_RECEIVED");

IncomingSMSReceiver receiver = new IncomgSMSReceiver();

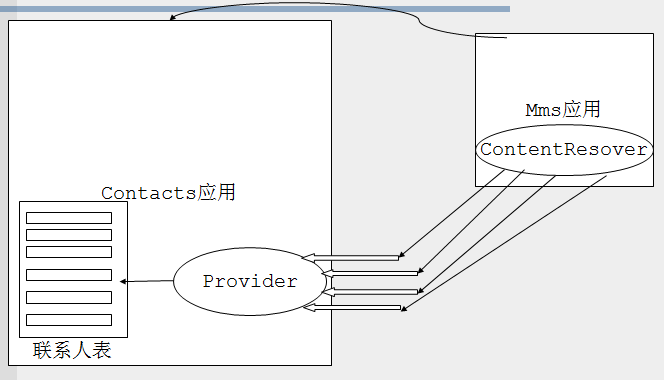
registerReceiver(receiver.filter);

两种注册类型的区别是：

1. 第一种是常驻型，也就是说当应用程序关闭后，如果有信息广播来，程序也会被系统调用自动运行。
2. 第二种不是常驻型广播，也就是说广播跟随程序的生命周期。

## 2.4. ContentProvider相关

### 17、请介绍下ContentProvider是如何实现数据共享的。



一个程序可以通过实现一个Content provider的抽象接口将自己的数据完全暴露出去，而且Content providers是以类似数据库中表的方式将数据暴露。Content providers存储和检索数据，通过它可以让所有的应用程序访问到，这也是应用程序之间唯一共享数据的方法。

要想使应用程序的数据公开化，可通过2种方法：创建一个属于你自己的Content provider或者将你的数据添加到一个已经存在的Content provider中，前提是有相同数据类型并且有写入Content provider的权限。

如何通过一套标准及统一的接口获取其他应用程序暴露的数据？Android提供了ContentResolver，外界的程序可以通过ContentResolver接口访问ContentProvider提供的数据。

## 2.5. Intent和Intent Filter。

### 18、请描述一下Intent和 Intent Filter

Intent在Android中被翻译为"意图"，它是activity，service和broadcast receiver之间互相激活时传递信息的”信使”。 它类似于我们WEB中的HttpServletRequest.

而Intent Filter相当于我们给Servlet配置的url-patten, 用于匹配某个具体的Intent对象.

## 2.6.综合

### 19、Android的四大组件是哪些，它们的作用？

Activity：

Activity是Android程序与用户交互的窗口，是Android构造块中最基本的一种，它需要为保持各界面的状态，做很多持久化的事情，妥善管理生命周期以及一些跳转逻辑

service：

后台服务于Activity，封装有一个完整的功能逻辑实现，接受上层指令，完成相关的事物，定义好需要接受的Intent提供同步和异步的接口

Content Provider：

是Android提供的第三方应用数据的访问方案，可以派生Content Provider类，对外提供数据，可以像数据库一样进行选择排序，屏蔽内部数据的存储细节，向外提供统一的接口模型，大大简化上层应用，对数据的整合提供了更方便的途径

BroadCast Receiver：

接受一种或者多种Intent作触发事件，接受相关消息，做一些简单处理，转换成一条Notification，统一了Android的事件广播模型

# 三. Android的UI

## 20、简单介绍一下Android中的View和ViewGroup

1、View

在Andorid应用程序中，UI元素称为View，它们都继承了android.view.View类。View有众多的子类，包括ViewGroup、基础控件、高级控件和布局。 基础控件主要包括：Button、ImageButton、ToggleButton、TextView、RadioButton、CheckBox、ImageView、ProgressBar、SeekBar等。

2、ViewGroup

    android.view.ViewGroup类是android.view.View重要的子类，ViewGroup类通常叫做“容器”，它就是由个控件组成的复杂控件，因为它也是View类的子类，所以本身也是控件。 ViewGroup是高级控件的和布局的父类，高级控件是和布局与基础控件一样都是不指具体那个类，而是一类容器的总称。

    高级控件都直接或者间接的继承了android.view.ViewGroup类，常用的高级控件主要包括：AutoCompleteTextView、Spinner、ListView、GridView、Gallery等。

## 21、请介绍下Android中常用的五种布局。

常用五种布局方式，分别是： LinearLayout （线性布局），RelativeLayout（相对布局），TableLayout（表格布局），FrameLayout（框架布局），AbsoluteLayout（绝对布局）。

|  |  |
| --- | --- |
| 布局 | 特点 |
| LinearLayout | 线性布局，每一个LinearLayout里面又可分为垂直布局和水平布局。当垂直布局时，每一行就只有一个元素，多个元素依次垂直往下；水平布局时，只有一行，每一个元素依次向右排列。 |
| RelativeLayout | 相对布局可以理解为某一个元素为参照物，来定位的布局方式。主要属性有：相对于某一个元素android:layout\_below、android:layout\_toLeftOf相对于父元素的地方android:layout\_alignParentLeft、android:layout\_alignParentRigh |
| FrameLayout | 所有东西依次都放在左上角，会重叠，这个布局比较简单，也只能放一点比较简单的东西。 |
| TableLayout | 表格布局，每一个TableLayout里面有表格行TableRow，TableRow里面可以具体定义每一个元素。每一个布局都有自己适合的方式，这五个布局元素可以相互嵌套应用，做出美观的界面。 |
| AbsoluteLayout | 绝对布局用X,Y坐标来指定元素的位置，这种布局方式也比较简单，但是在屏幕旋转时，往往会出问题，而且多个元素的时候，计算比较麻烦。 |

## 22、android中的动画有哪几类，它们的特点和区别是什么

答：两种，一种是View动画。这种实现方式可以使视图组件移动、放大、缩小以及产生透明度的变化; 另一种Drawable动画，传统的动画方法，通过顺序的播放排列好的图片来实现，类似于电影

## 23、ListView的优化方案

1、如果自定义适配器，那么在getView方法中要考虑方法传进来的参数contentView是否为null，如果为null就创建contentView并返回，如果不为null则直接使用。在这个方法中尽可能少创建view。

2、给contentView设置tag（setTag（）），传入一个viewHolder对象，用于缓存要显示的数据，可以达到图像数据异步加载的效果。

3、如果listview需要显示的item很多，就要考虑分页加载。比如一共要显示100条或者更多的时候，我们可以考虑先加载20条，等用户拉到列表底部的时候再去加载接下来的20条。

## 24、View的刷新:

view.postinvaliate()可以在分线程刷新  
 view.invaliate()只能在主线程中执行

## 25、NotifactionManager使用原理

1.  通过getSystemService方法获得一个NotificationManager对象。

2.  创建一个Notification对象。每一个Notification对应一个Notification对象。在这一步需要设置显示在屏幕上方状态栏的通知消息、通知消息前方的图像资源ID和发出通知的时间。一般为当前时间。

3.  由于Notification可以与应用程序脱离。也就是说，即使应用程序被关闭，

Notification仍然会显示在状态栏 中。当应用程序再次启动后，又可以重新控制这些

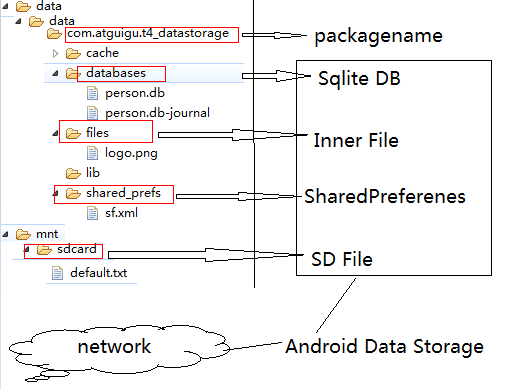
Notification。如清除或替换它们。因此，需要创建一个PendingIntent对象。该对象由Android系统负责维护，因此，在应用程序关闭后，该对象仍然不会被释放。

4.  使用Notification类的setLatestEventInfo方法设置Notification的详细信息。

5.  使用NotificationManager类的notify方法显示Notification消息。在这一步需要指定标识Notification的唯一ID。这个ID必须相对于同一个NotificationManager对象是唯一的，否则就会覆盖相同ID的Notificaiton。

# 四. Android数据存储与解析

## 26、请介绍下Android的数据存储方式。



Android提供了5种方式存储数据：

1). ***SharedPreferences存储***: 它是Android提供的用来存储一些简单配置信息的一种机制，采用了XML格式将数据存储到设备中。只能在同一个包内使用，不能在不同的包之间使用。

2). ***内部文件存储***: Android提供了openFileInput()和openFileOutput()方法来读取设备上的文件。

3). ***外部文件存储***: 可以将文件保存到SD卡中,后面可以进行读取工作

4). ***SQLite数据库存储***: SQLite是Android所带的一个标准的数据库，它支持SQL语句，它是一个轻量级的数据库。

5). ***网络存储:*** 通过网络上提供给我们的存储空间来上传(存储)和下载(获取)我们存储在网络空间中的数据信息。

## 27、android中有哪几种解析xml的类，官方推荐哪种？以及它们的原理和区别。

***方式一：DOM解析***

优点： 一次性将文档加载到内存中生成一个对象树, 解析更面向对象, 可以进行CRUD

缺点：   占用内存大, 对于大xml文件不合适

***方式二：SAX解析***

优点： 基于事件方法回调机制, 边读边回调方法并将数据传入,占用内存小.

缺点： 解析工作一开始就不能停止, 只能读, 不能CUD

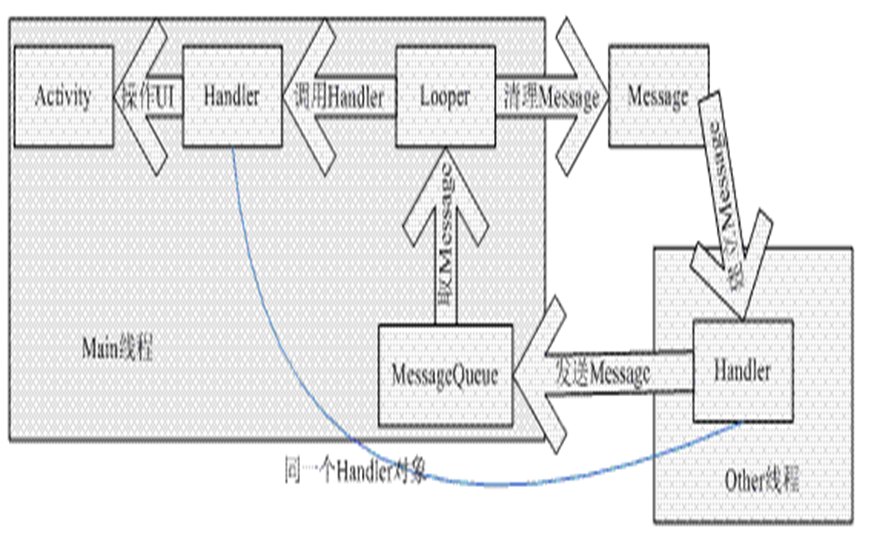
***方式三：PULL解析***

android SDK提供了xmlpull api，xmlpull和sax类似,但它可以在找到需要的数据后停止解析

# 五. Android核心机制

## 5.1. 消息与异步通信机制

### 28、请解释下在单线程模型中Message、Handler、Message Queue、Looper之间的关系。



简单的说，Handler获取当前线程中的looper对象，looper用来从存放Message的MessageQueue中取出Message，再有Handler进行Message的分发和处理。

Message Queue(消息队列)：用来存放通过Handler发布的消息，通常附属于某一个创建它的线程，可以通过Looper.myQueue()得到当前线程的消息队列

Handler：可以发布或者处理一个消息或者操作一个Runnable，通过Handler发布消息，消息将只会发送到与它关联的消息队列，然也只能处理该消息队列中的消息。

Looper：是Handler和消息队列之间通讯桥梁，程序组件首先通过Handler把消息传递给Looper，Looper把消息放入队列。Looper也把消息队列里的消息广播给所有的

Handler：Handler接受到消息后调用handleMessage进行处理

Message：消息的类型，在Handler类中的handleMessage方法中得到单个的消息进行处理

在单线程模型下，为了线程通信问题，Android设计了一个Message Queue(消息队列)， 线程间可以通过该Message Queue并结合Handler和Looper组件进行信息交换。下面将对它们进行分别介绍：

1. Message

    Message消息，理解为线程间交流的信息，处理数据后台线程需要更新UI，则发送Message内含一些数据给UI线程。

2. Handler

    Handler处理者，是Message的主要处理者，负责Message的发送，Message内容的执行处理。后台线程就是通过传进来的 Handler对象引用来sendMessage(Message)。而使用Handler，需要implement 该类的 handleMessage(Message)方法，它是处理这些Message的操作内容，例如Update UI。通常需要子类化Handler来实现handleMessage方法。

3. Message Queue

    Message Queue消息队列，用来存放通过Handler发布的消息，按照先进先出执行。

每个message queue都会有一个对应的Handler。Handler会向message queue通过两种方法发送消息：sendMessage或post。这两种消息都会插在message queue队尾并按先进先出执行。但通过这两种方法发送的消息执行的方式略有不同：通过sendMessage发送的是一个message对象,会被 Handler的handleMessage()函数处理；而通过post方法发送的是一个runnable对象，则会自己执行。

4. Looper

    Looper是每条线程里的Message Queue的管家。Android没有Global的Message Queue，而Android会自动替主线程(UI线程)建立Message Queue，但在子线程里并没有建立Message Queue。所以调用Looper.getMainLooper()得到的主线程的Looper不为NULL，但调用Looper.myLooper() 得到当前线程的Looper就有可能为NULL。对于子线程使用Looper，API Doc提供了正确的使用方法：这个Message机制的大概流程：

    1. 在Looper.loop()方法运行开始后，循环地按照接收顺序取出Message Queue里面的非NULL的Message。

    2. 一开始Message Queue里面的Message都是NULL的。当Handler.sendMessage(Message)到Message Queue，该函数里面设置了那个Message对象的target属性是当前的Handler对象。随后Looper取出了那个Message，则调用 该Message的target指向的Hander的dispatchMessage函数对Message进行处理。在dispatchMessage方法里，如何处理Message则由用户指定，三个判断，优先级从高到低：

    1) Message里面的Callback，一个实现了Runnable接口的对象，其中run函数做处理工作；

    2) Handler里面的mCallback指向的一个实现了Callback接口的对象，由其handleMessage进行处理；

    3) 处理消息Handler对象对应的类继承并实现了其中handleMessage函数，通过这个实现的handleMessage函数处理消息。

3. Handler处理完该Message (update UI) 后，Looper则设置该Message为NULL，以便回收！

### 29、说说你对AsyncTask的理解

在开发Android移动客户端的时候往往要使用多线程来进行操作，我们通常会将耗时的操作放在单独的线程执行，避免其占用主线程而给用户带来不好的用户体验。但是在子线程中无法去操作主线程（UI线程），在子线程中操作UI线程会出现错误。因此android提供了一个类Handler来在子线程中来更新UI线程，用发消息的机制更新UI界面，呈现给用户。这样就解决了子线程更新UI的问题。但是费时的任务操作总会启动一些匿名的子线程，太多的子线程给系统带来巨大的负担，随之带来一些性能问题。因此android提供了一个工具类AsyncTask，顾名思义异步执行任务。这个AsyncTask生来就是处理一些后台的比较耗时

的任务，给用户带来良好用户体验的，从编程的语法上显得优雅了许多，不再需要子线程和Handler就可以完成异步操作并且刷新用户界面。

## 5.2. 触摸机制

### 30、说说你对Android中的触控事件机制的理解

Android的触摸事件：onClick, onScroll, onFling等等，都是由许多个Touch组成的。其中Touch的第一个状态肯定是ACTION\_DOWN, 表示按下了屏幕。之后是ACTION\_MOVE(/表示为移动手势) 和ACTION\_UP(表示为离开屏幕) . 一个Action\_DOWN, n个ACTION\_MOVE, 1个ACTION\_UP，就构成了Android中众多的事件, 它是基于一种消费机制, 那什么是消费机制?

## 5.3. 其它

### 31、说说mvc模式的原理，它在android中的运用

mvc是model,view,controller的缩写，mvc包含三个部分：

　　模型（model）对象：是应用程序的主体部分，所有的业务逻辑都应该写在该层。

　　视图（view）对象：是应用程序中负责生成用户界面的部分。也是在整个mvc架构中用户唯一可以看到的一层，接收用户的输入，显示处理结果。

　　控制器（control）对象：是根据用户的输入，控制用户界面数据显示及更新model对象状态的部分，控制器更重要的一种导航功能，响应用户出发的相关事件，交给m层处理。

　　android鼓励弱耦合和组件的重用，在android中mvc的具体体现如下：

　　1)视图层（view）：一般采用xml文件进行界面的描述，使用的时候可以非常方便的引入。

　　2)控制层（controller）：android的控制层的重任通常落在了众多的acitvity的肩上.

　　3)模型层（model）：对数据库的操作、对网络等的操作都应该在model里面处理，当然对业务计算等操作也是必须放在的该层的。

# 六. Android应用开发相关

## 32、什么情况会导致Force Close ？如何避免？能否捕获导致其的异常？

答：程序出现异常，比如nullpointer。

避免：编写程序时逻辑连贯，思维缜密。能捕获异常，在logcat中能看到异常信息

## 33、Android中内存泄露出现情况有哪些？

1. 数据库的cursor没有关闭， 可以使用startManagerCursor(cursor)
2. 构造adapter时,没有使用缓存contentview，衍生listview的优化问题-----减少创建view的对象,充分使用contentview,可以使用一静态类来优化处理getview的过程
3. Bitmap对象不使用时没有释放，可以通过调用bitmap.recycle()释放内存

## 34、什么是ANR，如何避免它？

ANR：Application Not Responding。在Android中，活动管理器和窗口管理器这两个系统服务负责监视应用程序的响应，当用户操作的在5s内应用程序没能做出反应，BroadcastReceiver在10秒内没有执行完毕，就会出现应用程序无响应对话框，这既是ANR。

避免方法：Activity应该在它的关键生命周期方法（如onCreate()和onResume()）里尽可能少的去做创建操作。潜在的耗时操作，例如网络或数据库操作，或者高耗时的计算如改变位图尺寸，应该在子线程里（或者异步方式）来完成。主线程应该为子线程提供一个Handler，以便完成时能够提交给主线程。

# 七. 第三方组件