# 一、Android基础基础

1. ****四大组件的意义及使用，生命周期回调及意义****

Activity： 是一个应用组件，用户可与其提供的屏幕进行交互，以执行拨打电话、拍摄照片、发送电子邮件或查看地图等操作，

Activity的生命周期:

[Activity锁屏状态下的生命周期](http://blog.csdn.net/xgangzai/article/details/53453121" \t "http://blog.csdn.net/qq_32811865/article/details/_blank)

 一、 onCreate ：当活动第一次启动的时候，触发该方法，可以在此时完成活动的初始化工作。

onCreate 方法有一个参数，该参数可以为空（ null ），也可以是之前调用 onSaveInstanceState （）方法保存的状态信息。onRestoreInstanceState(Bundle)恢复过来。 默认的实现负责了大部分 UI 实例状态的保存，采用的方式是调用 UI 层上每个拥有 id 的 view 的 onSaveInstanceState()方法 ，并且保存当前获得焦点的 view 的 id （所有保存的状态信息都会在默认的 onRestoreInstanceState(Bundle) 实现中恢复)。如果你覆写这个方法来保存额外的没有被各个view保存的信息，你可能想要在默认实现过程中调用或者自己保存每个视图的所有状态。如果被调用，这个方法会在 onStop() 前被触发，但系统并不保证是否在 onPause() 之前或者之后触发。应用中按返回键，onSaveInstanceState() 方法就不会被调用。因为在这种情况下，用户的行为决定了不需要保存Activity的状态。通常onSaveInstanceState() 方法只适合用于保存一些临时性的状态，而onPause() 方法适合用于数据的持久化保存。如果你没有实现自己的 onSaveInstanceState()，但是 Activity 上控件的样子可能依然能被保存并恢复。原来 Activity 类已实现了onSaveInstanceState()，在 onSaveInstanceState() 的默认实现中，会调用所有控件的相关方法，把控件们的状态都保存下来，比如 EditText 中输入的文字、CheckBox 是否被选中等等。然而不是所有的控件都能被保存，这取决于你是否在 layout 文件中为控件赋了一个名字(android:id)。有名的就存，无名的不管。

二、  onStart ：该方法的触发表示所属活动将被展现给用户。

三、  onResume ：当一个活动和用户发生交互的时候，触发该方法。

四、  onPause ：当一个正在前台运行的活动因为其他的活动需要前台运行而转入后台运行的时候，触发该方法。这时候需要将活动的状态持久化，比如正在编辑的数据库记录等。

五、  onStop ：当一个活动不再需要展示给用户的时候，触发该方法。如果内存紧张，系统会直接结束这个活动，而不会触发 onStop 方法。 所以保存状态信息是应该在onPause时做，而不是onStop时做。活动如果没有在前台运行，都将被停止或者Linux管理进程为了给新的活动预留足够的存储空间而随时结束这些活动。因此对于开发者来说，在设计应用程序的时候，必须时刻牢记这一原则。在一些情况下，onPause方法或许是活动触发的最后的方法，因此开发者需要在这个时候保存需要保存的信息。

六、onRestart：当处于停止状态的活动需要再次展现给用户的时候，触发该方法。

七、 onDestroy ：当活动销毁的时候，触发该方法。和 onStop 方法一样，如果内存紧张，系统会直接结束这个活动而不会触发该方法。

Activity栈。

·standard: 标准模式，一调用startActivity()方法就会产生一个新的实例。

·singleTop: 如果已经有一个实例位于Activity栈的顶部时，就不产生新的实例，而只是调用Activity中的newInstance()方法。如果不位于栈顶，会产生一个新的实例。

            使用场景如新闻类或者阅读类App的内容页面。

|  |
| --- |
| singletop 这个我曾经用来避免多次创建，比如点击一个按钮启动一个activity，如果快速点击多次会导致反复启动，一种办法是在点击事件里过滤，另一个办法是设置目标activity是singletop |

·singleTask: 会在一个新的task中产生这个实例，以后每次调用都会使用这个，不会去产生新的实例了。

              使用场景如浏览器的主界面。不管从多少个应用启动浏览器，只会启动主界面一次，其余情况都会走onNewIntent，并且会清空主界面上面的其他页面。

|  |
| --- |
| 联系人activity |

·singleInstance: 这个跟singleTask基本上是一样，只有一个区别：在这个模式下的Activity实例所处的task中，只能有这个activity实例，不能有其他的实例。

|  |
| --- |
| 来电显示界面 |

Service:

    Service是在一段不定的时间运行在后台，不和用户交互应用组件。每个Service必须在manifest中通过<service>来声明。可以通过contect.startservice和contect.bindserverice来启动。Service和其他的应用组件一样，运行在进程的主线程中。这就是说如果service需要很多耗时或者阻塞的操作，需要在其子线程中实现。service的两种模式（startService()/bindService()不是完全分离的）。

      生命周期：

    使用context.startService() 启动Service是会经历:

　　context.startService() ->onCreate()- >onStart()->Service running

　　context.stopService() | ->onDestroy() ->Service stop

    使用使用context.bindService()启动Service会经历：

　　context.bindService()->onCreate()->onBind()->Service running

    onUnbind() -> onDestroy() ->Service stop

Brocast receiver

BroadcastReceiver 用于异步接收广播Intent。

　·正常广播 Normal broadcasts（用 Context.sendBroadcast()发送）是完全异步的。它们都运行在一个未定义的顺序，通常是在同一时间。这样会更有效，但意味着receiver不能包含所要使用的结果或中止的API。

　·有序广播 Ordered broadcasts（用 Context.sendOrderedBroadcast()发送）每次被发送到一个receiver。所谓有序，就是每个receiver执行后可以传播到下一个receiver，也可以完全中止传播--不传播给其他receiver。 而receiver运行的顺序可以通过matched intent-filter 里面的android:priority来控制，当priority优先级相同的时候，Receiver以任意的顺序运行。

1. 对于sendBroadCast的intent对象，需要设置其action name；

2. 推荐使用显式指明receiver，在配置文件AndroidManifest.xml指明；

3. 一个receiver可以接收多个action;

4. 每次接收广播都会重新生成一个接收广播的对象，再次调用onReceive；

5. 在BroadCast 中尽量不要处理太多逻辑问题，建议复杂的逻辑交给Activity 或者 Service 去处理。

ContentProvider

    主要用于对外共享数据，也就是通过ContentProvider把应用中的数据共享给其他应用访问，其他应用可以通过ContentProvider对指定应用中的数据进行操作。　　当外部应用需要对ContentProvider中的数据进行添加、删除、修改和查询操作时，可以使用ContentResolver类来完成，要获取ContentResolver对象，可以使用Context提供的getContentResolver()方法。

1. ****AsyncTask、Handler的使用****

1、先实例化一个AsyncTask的继承子类，此时将会创建一个task。接下来变执行execute(params)方法启动异步任务。（同一个AsyncTask的实例只能执行execute一次，多次执行会抛出错误）。

2、在execute()被执行后，将会触发onPreExecute()回调方法，设置进度条的初始属性。在onPreExecute()执行完毕后，将会在后台线程开始执行doInBackground(params)，该方法接收execute传入的参数，进行耗时操作，这里是模拟网络文件下载任务。

3、doInBackground()在后台线程运行中，如果需要与UI主线程交互更新进度，可以调用publishProgress(values)方法，将会触发位于UI主线程运行的onProgressUpdate(values)的回调方法，代码中在这里更新进度条的进度。

4、 当后台任务执行完成后，调用onPostExecute(Result)，传入的参数是doInBackground()中返回的对象。

注意事项：

1、不要在同一个AsyncTask实例中多次执行execute()，正确的方法是new一个AsyncTask执行一次execute()。

2、耗时任务一定要在doInBackground()中处理，不要在其他回调方法中处理耗时任务以免引起UI主线程的阻塞。

3、不要再doInBackground()中更新UI界面，应该通过publishProgress()调用回调方法更新UI。

4、onCancelled()只能触发AsyncTask的cancel()方法，并无法取消正在线程池运行的线程任务，但可以通过标志位来停止线程任务。

5、在不同的android版本中，AsyncTask多任务运行，有些是可以并行有些则是顺序执行，不过在高版本Android中，可以通过指定参数设置线程池执行规则。

6、AsyncTask适合处理短时间的操作,长时间的操作,比如下载一个很大的视频,这就需要你使用自己的线程来下载,不管是断点下载还是其它的。

Handler可以做两件事，第一可以在子线程中更新UI，第二可以用来处理消息。

1. ****Android系统层次框架结构****
2. 应用程序  
   所有的应用程序都是使用JAVA语言编写的，每一个应用程序由一个或者多个活动组成，活动必须以Activity类为超类，活动类似于操作系统上的进程，但是活动比操作系统的进程要更为灵活，与进程类似的是，活动在多种状态之间进行切换。
3. 应用程序框架  
   应用程序的****架构****设计简化了组件的重用；任何一个应用程序都可以发布它的功能块并且任何其它的应用程序都可以使用其所发布的功能块
4. 系统运行库

 a)程序库

    Android包含一些C/C++库，这些库能被Android系统中不同的组件使用。

b)Android 运行库

  Android 包括了一个核心库，该核心库提供了JAVA编程语言核心库的大多数功能。

4) Linux 内核

Android 的核心系统服务依赖于 Linux 2.6 内核 ，如安全性，内存管理，进程管理， 网络协议栈和驱动模型。

6、Android常见的存储方式

第一种： 使用SharedPreferences存储数据

适用范围：保存少量的数据，且这些数据的格式非常简单：字符串型、基本类型的值。比如应用程序的各种配置信息（如是否打开音效、是否使用震动效果、小游戏的玩家积分等），解锁口 令密码等

第二种： 文件存储数据

 核心原理: Context提供了两个方法来打开数据文件里的文件IO流 FileInputStream openFileInput(String name); FileOutputStream(String name , int mode),这两个方法第一个参数 用于指定文件名，第二个参数指定打开文件的模式。

 第三种：SQLite存储数据

SQLite是轻量级嵌入式数据库引擎，它支持 SQL 语言，并且只利用很少的内存就有很好的性能。

 4 使用ContentProvider存储数据

 5 网络存储数据

****7、Looper、Handler和MessageQueue的关系****

1) Looper类别用来为一个线程开启一个消息循环。默认情况下Android中新诞生的线程是没有开启消息循环的。（主线程除外，主线程系统会自动为其创建Looper对象，开启消息循环）

Looper对象通过MessageQueue来存放消息和事件。一个线程只能有一个Looper，对应一个MessageQueue。

(2) 通常是通过Handler对象来与Looper交互的。Handler可看做是Looper的一个接口，用来向指定的Looper发送消息及定义处理方法。

默认情况下Handler会与其被定义时所在线程的Looper绑定，比如，在主线程中定义，其是与主线程的Looper绑定。

mainHandler = new Handler() 等价于new Handler（Looper.myLooper()）.

Looper.myLooper()：Return the Looper object associated with the current thread 获取当前进程的looper对象。

还有一个类似的 Looper.getMainLooper() 用于获取主线程的Looper对象。

(3) 在非主线程中直接new Handler() 会报如下的错误:

E/AndroidRuntime( 6173): Uncaught handler: thread Thread-8 exiting due to uncaught exception

E/AndroidRuntime( 6173):Java.lang.RuntimeException: Can't create handler inside thread that has not called Looper.prepare()

原因是非主线程中默认没有创建Looper对象，需要先调用Looper.prepare()启用Looper。

(4) Looper.loop(); 让Looper开始工作，从消息队列里取消息，处理消息。

注意：写在Looper.loop()之后的代码不会被执行，这个函数内部应该是一个循环，当调用mHandler.getLooper().quit()后，loop才会中止，其后的代码才能得以运行。

****8、Activity的启动流程（考察对Framwork的熟悉程度）****

一:开始请求执行启动Activity

二：ActivityManagerService接收启动Activity的请求

三：执行栈顶Activity的onPause方法

四：启动Activity所属的应用进程

五：执行启动Acitivity

六：栈顶Activity执行onStop方法

总结一下一个完成的冷启动app过程应该是经过：

Zygote Fork Proccess

-> Application:attachBaseContext()

-> Application:onCreate()

-> MainActiviity:onCreate()

针对于解决第三方插件初始化耗时方案一般是：

1.SDK分优先级加载，非必要SDK由懒加载实现。

2.可以多线程初始化的sdk由多线程方式来进行初始化。

避免冷启动：

****优化方案：****

****懒加载****

****减少oncreate的工作量，不要让application参与业务操作，不要再application进行耗时操作，不要在application保存静态变量，减少布局的深度、复杂性****

既然冷启动那么慢，我们就在非用户主动kill进程或系统通知kill进程的其他情况下不再主动退出进程。那答案很简单了，就是在位于Activity栈底activity中Hook其返回键行为，保证用户点击返回键后不再退出app。在我们App里位于我们栈底的一定是我们的MainActivity，因为一系统行为都是由其向下衍生的。所以只需加入以下几句话：

@Overide:

Public void onBackPressed(){

moveTaskToBack(true);}

moveTaskToBack：作用是不再Finish到此Activity，仅仅是把它放到后台隐藏。类似于用户主动触发系统Home键的效果。在同是点击返回键优化前后的对比如下：

总结：

Activity的启动流程一般是通过调用startActivity或者是startActivityForResult来开始的

startActivity内部也是通过调用startActivityForResult来启动Activity，只不过传递的requestCode小于0

Activity的启动流程涉及到多个进程之间的通讯这里主要是ActivityThread与ActivityManagerService之间的通讯

ActivityThread向ActivityManagerService传递进程间消息通过ActivityManagerNative，ActivityManagerService向ActivityThread进程间传递消息通过IApplicationThread。

ActivityManagerService接收到应用进程创建Activity的请求之后会执行初始化操作，解析启动模式，保存请求信息等一系列操作。

ActivityManagerService保存完请求信息之后会将当前系统栈顶的Activity执行onPause操作，并且IApplication进程间通讯告诉应用程序继承执行当前栈顶的Activity的onPause方法；

ActivityThread接收到SystemServer的消息之后会统一交个自身定义的Handler对象处理分发；

ActivityThread执行完栈顶的Activity的onPause方法之后会通过ActivityManagerNative执行进程间通讯告诉ActivityManagerService，栈顶Actiity已经执行完成onPause方法，继续执行后续操作；

ActivityManagerService会继续执行启动Activity的逻辑，这时候会判断需要启动的Activity所属的应用进程是否已经启动，若没有启动则首先会启动这个Activity的应用程序进程；

ActivityManagerService会通过socket与Zygote继承通讯，并告知Zygote进程fork出一个新的应用程序进程，然后执行ActivityThread的mani方法；

在ActivityThead.main方法中执行初始化操作，初始化主线程异步消息，然后通知ActivityManagerService执行进程初始化操作；

ActivityManagerService会在执行初始化操作的同时检测当前进程是否有需要创建的Activity对象，若有的话，则执行创建操作；

ActivityManagerService将执行创建Activity的通知告知ActivityThread，然后通过反射机制创建出Activity对象，并执行Activity的onCreate方法，onStart方法，onResume方法；

ActivityThread执行完成onResume方法之后告知ActivityManagerService onResume执行完成，开始执行栈顶Activity的onStop方法；

ActivityManagerService开始执行栈顶的onStop方法并告知ActivityThread；

ActivityThread执行真正的onStop方法；

****9、多进程开发的注意事项(Application类区分进程，进程间内存不可见、进程间通讯方式)****

Application类区分进程：

可以通过Process,myPid 和 RunningAppProcessInfo 区分

int pid = android.os.Process.myPid();

ActivityManager mActivityManager = (ActivityManager) context.getSystemService(Context.ACTIVITY\_SERVICE);

for (ActivityManager.RunningAppProcessInfo appProcess : mActivityManager.getRunningAppProcesses()) {

        if(appProcess.pid == pid) {

                if(appProcess.processName.equalsIgnoreCase("com.myProcess")) {return true;}}}

****10、activity四种启动模式区别和应用场景。****

singleTop适合接收通知启动的内容显示页面。例如，某个新闻客户端的新闻内容页面，如果收到10个新闻推送，每次都打开一个新闻内容页面是很烦人的。

singleTask适合作为程序入口点。例如浏览器的主界面。不管从多少个应用启动浏览器，只会启动主界面一次，其余情况都会走onNewIntent，并且会清空主界面上面的其他页面。之前打开过的页面，打开之前的页面就ok，不再新建。

singleInstance适合需要与程序分离开的页面。例如闹铃提醒，将闹铃提醒与闹铃设置分离。singleInstance不要用于中间页面，如果用于中间页面，跳转会有问题，比如：A -> B (singleInstance) -> C，完全退出后，在此启动，首先打开的是B。

****11、service生命周期，两种启动方式的区别。****

    　service的生命周期，从它被创建开始，到它被销毁为止，可以有两条不同的路径：

A started service

　　被开启的service通过其他组件调用 [startService()](http://developer.android.com/reference/android/content/Context.html /l startService(android.content.Intent) /t _blank" \t "http://blog.csdn.net/qq_32811865/article/details/_blank)被创建。

　　这种service可以无限地运行下去，必须调用[stopSelf()](http://developer.android.com/reference/android/app/Service.html /l stopSelf() /t _blank" \t "http://blog.csdn.net/qq_32811865/article/details/_blank)方法或者其他组件调用[stopService()](http://developer.android.com/reference/android/content/Context.html /l stopService(android.content.Intent) /t _blank" \t "http://blog.csdn.net/qq_32811865/article/details/_blank)方法来停止它。

　　当service被停止时，系统会销毁它。

A bound service

　　被绑定的service是当其他组件（一个客户）调用[bindService()](http://developer.android.com/reference/android/content/Context.html /l bindService(android.content.Intent, android.content.ServiceConnection, int) /t _blank" \t "http://blog.csdn.net/qq_32811865/article/details/_blank)来创建的。

　　客户可以通过一个[IBinder](http://developer.android.com/reference/android/os/IBinder.html /t _blank" \t "http://blog.csdn.net/qq_32811865/article/details/_blank)接口和service进行通信。

　　客户可以通过 [unbindService()](http://developer.android.com/reference/android/content/Context.html /l unbindService(android.content.ServiceConnection) /t _blank" \t "http://blog.csdn.net/qq_32811865/article/details/_blank)方法来关闭这种连接。

　　一个service可以同时和多个客户绑定，当多个客户都解除绑定之后，系统会销毁service。

　　这两条路径并不是完全分开的。

　　即是说，你可以和一个已经调用了 startService()而被开启的service进行绑定。

比如，一个后台音乐service可能因调用 startService()方法而被开启了，稍后，可能用户想要控制播放器或者得到一些当前歌曲的信息，可以通过bindService()将一个activity和service绑定。这种情况下，[stopService()](http://developer.android.com/reference/android/content/Context.html /l stopService(android.content.Intent) /t _blank" \t "http://blog.csdn.net/qq_32811865/article/details/_blank)或 stopSelf()实际上并不能停止这个service，除非所有的客户都解除绑定

****12、 实现ipc的方法有哪些？****

IPC是Inter-Process Communication的缩写，意为进程间通信或者跨进程通信，是指两个进程进行数据交换的过程。下面就介绍一下IPC 的几种方式：

1.Bundle

这种方式使用的是比较多的，也是很常见的。四大组件中的三大组件（Activity，BroadcaseReceiver，Service）都是支持在Intent中传递Bundle数据的，由于Bundle实现了Parcelable接口，所以可以很方便的在不同进程间传输。这里就不再赘述了。

2.使用文件共享

共享文件也是一种不错的进程间通信方式，两个进程通过读/写同一个文件来交换数据。这里也不再赘述。但有一点要注意：android系统是基于Linux的，使得其并发读/写文件可以没限制地进行，甚至两线程同时对同一文件进行写操作都是允许的，尽管这可能出问题。So，重点：文件共享方式适合在对数据同步要求不高的进程间进行通信，并且要妥善处理并发读/写问题。

3.Messenger

Messenger译为信使，顾名思义，主要作用就是传递消息。通过它可在不同进程中传递Message对象，在Message中放入要传递的数据，即可轻松地实现数据的进程间传递了。Messenger是一种轻量级的IPC方案，底层实现是AIDL。Messenger的使用方法很简单，他对AIDL做了封装，使得我们更简单的进行进程间通信。

使用Messenger，分为服务端和客户端：

（1）服务端进程

首先，需要在服务端创建一个Service  来处理客户端的连接请求，同时创建一个Handler（重写handleMessage方法处理接受的数据）并通过此Handler来创建一个Messenger对象，然后在Service的onBind中返回这个Messenger对象底层的 Binder即可。

（2）客户端进程

客户端进程中，首先需要绑定服务端的Service，绑定成功后，用服务端返回的IBinder对象来创建一个Messenger，然后通过Messenger就可以向服务端发送消息了，发送的消息类型为Message对象。【注：如果需要服务端能够 回应客户端，就和服务端一样，我们需要在客户端也创建一个Handler（重写handleMessage方法处理回传的数据）并创建一个新的Messenger，并把这个Messenger对象通过Message的replyTo参数传递给服务端，服务端通过这个replyTo参数就可以回应客户端】。

4.AIDL

上面说到Messenger，其是以串行的方式处理客户端发来的消息，如果有大量的并发请求，那么使用Messenger就不太合适了。同时Messenger主要作用就是传递消息，很多时候我们可能需要跨进程调用服务端的方法，这种情形Messenger就无法做到了，但是我们可以使用AIDL来实现跨进程的方法调用。下面就介绍一下AIDL的使用。

使用使用AIDL，也分为服务端和客户端：

（1）服务端

服务端首先要创建一个Service用来监听客户端的连接请求，然后创建一个AIDL文件，将暴露给客户端的接口在这个AIDL文件中声明，最后在这个Service中实现这个AIDL接口即可。

（2）客户端

客户端要做的事情稍微简单一些，首先需要绑定服务端的Service，绑定成功后，将服务端返回的Binder对象转成AIDL所属的类型，接着就可以调用AIDL中的方法了。

5.ContentProvider

ContentProvider是Android中提供的专门用于不同应用间进行数据共享的方式，从这一点，它天生就适合进程间通信。和Messenger一样，contentProvider的底层实现同样是Binder，由此可见，Binder在Android中是何等的重要。虽然ContentProvider底层是用Binder，但它的使用过程要比AIDL 简单许多，因为系统已经做了封装。

系统预置了许多ContentProvider，比如通讯录信息，日程变信息等，要跨进程访问这些信息，只需要通过ContentResolver的query、update、insert和delete方法即可。

使用：

创建一个自定义的ContentProvider很简单，只需要继承ContentProvider类并实现六个抽象方法即可：onCreate、query、update、insert、delete和getType。除了onCreate由系统回调并运行在主线程中，其他五个方法由外界回调并运行在Binder线程池中。

6.Socket

Socket 也称为“套接字”。是网络通信中的概念，它分为流式套接字和用户数据报套接字两种，分别对应于网络传输控制层中的TCP和UDP协议。TCP是面向连接的协议，提供稳定的双向通信功能，TCP连接的建立需要经过“三次握手”才能完成，为了提供稳定的数据传输功能，其本身提供了超时重传机制，因此具有很高的稳定性。而UDP是无连接的，提供不稳定的单向通信功能，当然UDP也能实现双向通信功能。在性能上，UDP 具有更高的效率，缺点是不保证数据一定能够正确传输，尤其是在网络阻塞的情况下。

****13、handler的内在原理。消息队列为空会怎样？****

Handler 创建时 会用当前线程的looper来构建内部的消息循环系统 handler创建完毕后looper和messageQueue就可以和handler一起协同工作了

handler 通过post方法将一个Runnable对象发送到handler内部的looper中去处理 也可以通过handler的send方法发送消息 这个消息会在looper中去处理

其实post方法最终也是通过send方法来完成的

当send方法被调用时会调用messageQueue的enqueueMessage方法将消息入列 looper发现了这个消息就会处理最终消息中的Runnable或者handler的handleMessage方法就会被调用

Looper是运行在创建handler所在的线程中的

若消息队列为空，线程则会阻塞等待。

# 二、Resource相关

****1、.9图片的意义****

9patch图片的作用就是在图片拉伸的时候保证其不会失真。所以我们使用.9图片，让图片在指定的位置拉伸和在指定的位置显示内容，这样图片的边边角角就不会出现失真了。

****2、style和theme的作用及用法****

   (1) Theme是针对窗体级别的，改变窗体样式的，如窗口标题、边框等。不能作用于单个View组件，是对整个应用的所有Activity或单个Activity起作用。

   (2) Style是针对窗体元素级别的,改变指定控件或者Layout的样式

1. ****dpi、sp、px的区别以及转换关系 、****

dpi(dots per inch)：单位英寸上的像素数量，是单位。公式为： dpi = 屏幕对角线像素数（px） / 屏幕对角线长度（in）。

px   即像素，1px代表屏幕上的一个物理像素点。

sp:scale-independent pixels（缩放独立像素），与缩放无关的抽象像素，它和dp很相似，但唯一的区别在于，Android系统允许用户自定义文字尺寸大小（小，正常，大，超大等），当文字尺寸是“正常”时，1sp=1dp=0.00625inch（英寸），当文字尺寸是“大”或“超大”时，1sp>1dp=0.00625inch

****dp与px****

px=dp\*(dpi/160)

dp=px/(dpi/160)

****sp与px****

px=sp\*(dpi/160)

sp=px/(dpi/160)

1. ****raw和assets文件夹的作用，二者有何区别****

\*res/raw和assets的不同点：

1.res/raw中的文件会被映射到R.Java文件中，访问的时候直接使用资源ID即[R.id](http://r.id/" \t "http://blog.csdn.net/qq_32811865/article/details/_blank).filename；assets文件夹下的文件不会被映射到R.java中，访问的时候需要AssetManager类。

2.res/raw不可以有目录结构，而assets则可以有目录结构，也就是assets目录下可以再建立文件夹

****5、Android系统如何在多个资源文件夹下查找匹配最合适的资源****

获取手机当前的基本配置信息(语言，横竖屏，屏幕密度，屏幕尺寸等等)

根据这些配置信息，排除apk包中与这些配置信息相矛盾的资源目录，假设系统语言是cn，那么所有的其他语言的目录都会被排除掉，注意系统并不会根据一个dpi的冲突而排除掉含有其他dpi的目录，dpi这个qualifier非常特殊

按照qualifirer的优先级，依次拿出配置信息中优先级最高的qualifier，去资源目录中寻找包含这个qualifer的目录，如果有，排除掉其他目录，根据配置中下一个优先级的qualifier继续匹配，如果没有，接着去拿下一个优先级的qualifier，继续寻找和排除，直到找到最合适的目录。

# 三、View相关

****1、常用组件的使用：ListView、RecyclerView及Adapter的使用****

****listview的显示需要三个数据：listview，适配器，数据。数据量少的时候我们可以用SImpleAdapter，数据量多的时候可以用BaseAdapter。这样拓展性比较好，也容易优化。****

****2、View之间的继承关系****

****3、Invalidate与postInvalidate的区别****

Android的invalidate与postInvalidate都是用来刷新界面的。

在UI主线程中，用invalidate()；本质是调用View的onDraw（）绘制。

主线程之外，用postInvalidate()。

****4、自定义View的实现方式(根据项目经验询问相关组件)。****

1、自定义View的属性： 自定义View的属性，首先在res/values/  下建立一个attrs.xml ， 在里面定义我们的属性和声明我们的整个样式。

2、在View的构造方法中获得我们自定义的属性

public CustomTitleView(Context context, AttributeSet attrs)

    {

        this(context, attrs, 0);

    }

3、重写onMesure ]

4、重写onDraw

****5、onMeasure/onLayout/onDraw的作用****

****6、Paint、Matrix、Shader等绘制相关类的方法作用****

****7、详细描述事件分发机制****

1. ****图片处理****
2. ****一般项目中如何加载大图****

通过设置BitmapFactory.Options中inSampleSize的值就可以实现图片内存的压缩

1. ****图片压缩的方式****

Android中图片是以bitmap形式存在的，那么bitmap所占内存，直接影响到了应用所占内存大小，首先要知道bitmap所占内存大小计算方式：图片长度 x 图片宽度 x 一个像素点占用的字节数

ALPHA\_8

表示8位Alpha位图,即A=8,一个像素点占用1个字节,它没有颜色,只有透明度

ARGB\_4444

表示16位ARGB位图，即A=4,R=4,G=4,B=4,一个像素点占4+4+4+4=16位，2个字节

ARGB\_8888

表示32位ARGB位图，即A=8,R=8,G=8,B=8,一个像素点占8+8+8+8=32位，4个字节

RGB\_565

表示16位RGB位图,即R=5,G=6,B=5,它没有透明度,一个像素点占5+6+5=16位，2个字节

1. 质量压缩

ByteArrayOutputStream baos = new ByteArrayOutputStream();

            int quality = Integer.valueOf(editText.getText().toString());

            bit.compress(CompressFormat.JPEG,quality, baos);

            byte[] bytes = baos.toByteArray();

            bm = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);

            Log.i("wechat", "压缩后图片的大小" + (bm.getByteCount() / 1024 / 1024)

                    + "M宽度为" + bm.getWidth() + "高度为" + bm.getHeight()

                    + "bytes.length=  " + (bytes.length / 1024) + "KB"

                    + "quality=" + quality);

1. 采样率压缩

BitmapFactory.Options options = new BitmapFactory.Options();

            options.inSampleSize= 2;

            bm = BitmapFactory.decodeFile(Environment

                    .getExternalStorageDirectory().getAbsolutePath()

                    + "/DCIM/Camera/test.jpg", options);

            Log.i("wechat", "压缩后图片的大小" + (bm.getByteCount() / 1024 / 1024)

                    + "M宽度为" + bm.getWidth() + "高度为" + bm.getHeight());

1. 缩放法压缩（martix）

Matrix matrix = new Matrix();

           matrix.setScale(0.5f, 0.5f);

            bm = Bitmap.createBitmap(bit, 0, 0, bit.getWidth(),

                    bit.getHeight(), matrix, true);

            Log.i("wechat", "压缩后图片的大小" + (bm.getByteCount() / 1024 / 1024)

                    + "M宽度为" + bm.getWidth() + "高度为" + bm.getHeight());

可以看出来，bitmap的长度和宽度分别缩小了一半，图片大小缩小了四分之一。

1. RGB\_565法

bm = Bitmap.createScaledBitmap(bit,150, 150, true);Log.i("wechat", "压缩后图片的大小" + (bm.getByteCount() / 1024) + "KB宽度为"+ bm.getWidth() + "高度为" + bm.getHeight());这里是将图片压缩成用户所期望的长度和宽度，但是这里要说，如果用户期望的长度和宽度和原图长度宽度相差太多的话，图片会很不清晰。

1. ****如何不压缩图片加载高清图****

首先不压缩，按照原图尺寸加载，那么屏幕肯定是不够大的，并且考虑到内存的情况，不可能一次性整图加载到内存中，所以肯定是局部加载，那么就需要用到一个类：BitmapRegionDecoder 其次，既然屏幕显示不完，那么最起码要添加一个上下左右拖动的手势，让用户可以拖动查看。BitmapRegionDecoder主要用于显示图片的某一块矩形区域，如果你需要显示某个图片的指定区域，那么这个类非常合适。

1. ****图片加载过程中，一般会使用缓存，这个缓存的主要作用是什么****

另外一种图片缓存的方式就是内存缓存技术。在Android中，有一个叫做LruCache类专门用来做图片缓存处理的。

它有一个特点，当缓存的图片达到了预先设定的值的时候，那么近期使用次数最少的图片就会被回收掉。

步骤：

（1）要先设置缓存图片的内存大小，我这里设置为手机内存的1/8,

           手机内存的获取方式：int MAXMEMONRY = (int) (Runtime.getRuntime() .maxMemory() / 1024);

（2）LruCache里面的键值对分别是URL和对应的图片

（3）重写了一个叫做sizeOf的方法，返回的是图片数量。

（4）下面的方法分别是清空缓存、添加图片到缓存、从缓存中取得图片、从缓存中移除。移除和清除缓存是必须要做的事，因为图片缓存处理不当就会报内存溢出，所以一定要引起注意。

****5、谈谈自己熟悉的图片加载框架****

ImageLaoder：

1. 多线程下载图片，图片可以来源于网络，文件系统，项目文件夹assets中以及drawable中等
2. 支持随意的配置ImageLoader，例如线程池，图片下载器，内存缓存策略，硬盘缓存策略，图片显示选项以及其他的一些配置
3. 支持图片的内存缓存，文件系统缓存或者SD卡缓存
4. 支持图片下载过程的监听
5. 根据控件(ImageView)的大小对Bitmap进行裁剪，减少Bitmap占用过多的内存
6. 较好的控制图片的加载过程，例如暂停图片加载，重新开始加载图片，一般使用在ListView,GridView中，滑动过程中暂停加载图片，停止滑动的时候去加载图片
7. 提供在较慢的网络下对图片进行加载
8. ****默认实现多种内存缓存算法**** 这几个图片缓存都可以配置缓存算法，不过ImageLoader 默认实现了较多缓存算法，如Size 最大先删除、使用最少先删除、最近最少使用、先进先删除、时间最长先删除等。

Picasso介绍：

1.在adapter中需要取消已经不在视野范围的ImageView图片资源的加载，否则会导致图片错位，Picasso已经解决了这个问题。

   2.使用复杂的图片压缩转换来尽可能的减少内存消耗

   3.自带内存和硬盘二级缓存功能

 三、Glide简介:

* GIF动画的解码：通过调用Glide.with(context).load(“图片路径“)方法，GIF动画图片可以自动显示为动画效果。如果想有更多的控制，还可以使用Glide.with(context).load(“图片路径“).asBitmap()方法加载静态图片，使用Glide.with(context).load(“图片路径“).asGif()方法加载动画图片
* 本地视频剧照的解码：通过调用Glide.with(context).load(“图片路径“)方法，Glide能够支持Android设备中的所有视频剧照的加载和展示
* 缩略图的支持：为了减少在同一个view组件里同时加载多张图片的时间，可以调用Glide.with(context).load(“图片路径“).thumbnail(“缩略比例“).into(“view组件“)方法加载一个缩略图，还可以控制thumbnail()中的参数的大小，以控制显示不同比例大小的缩略图
* Activity生命周期的集成：当Activity暂停和重启时，Glide能够做到智能的暂停和重新开始请求，并且当Android设备的连接状态变化时，所有失败的请求能够自动重新请求
* 转码的支持：Glide的toBytes()和transcode()两个方法可以用来获取、解码和变换背景图片，并且transcode()方法还能够改变图片的样式
* 动画的支持：新增支持图片的淡入淡出动画效果（调用crossFade()方法）和查看动画的属性的功能
* OkHttp和Volley的支持：默认选择HttpUrlConnection作为网络协议栈，还可以选择OkHttp和Volley作为网络协议栈
* 其他功能：如在图片加载过程中，使用Drawables对象作为占位符、图片请求的优化、图片的宽度和高度可重新设定、缩略图和原图的缓存等功能

1. ****动画类****

****1、Android有哪些动画的实现方式****

a)、View Animation：视图动画，也叫Tween（补间）动画可以在一个视图容器内执行一系列简单变换（位置、大小、旋转、透明度）。譬如，如果你有一个TextView对象，您可以移动、旋转、缩放、透明度设置其文本，当然，如果它有一个背景图像，背景图像会随着文本变化。补间动画通过XML或Android代码定义，建议使用XML文件定义，因为它更具可读性、可重用性。

***特别特别注意：补间动画执行之后并未改变View的真实布局属性值。切记这一点，譬如我们在Activity中有一个Button在屏幕上方，我们设置了平移动画移动到屏幕下方然后保持动画最后执行状态呆在屏幕下方，这时如果点击屏幕下方动画执行之后的Button是没有任何反应的，而点击原来屏幕上方没有Button的地方却响应的是点击Button的事件。***

****b）、Drawable Animation（Drawable动画）：****Drawable动画其实就是Frame动画（帧动画），它允许你实现像播放幻灯片一样的效果，这种动画的实质其实是Drawable，所以这种动画的XML定义方式文件一般放在res/drawable/目录下

c)、****Property Animation（属性动画）：****属性动画实现原理就是修改控件的属性值实现的动画。

总结：

****View动画：****

View动画只能够为View添加动画，如果想为非View对象添加动画须自己实现；且View动画支持的种类很少；尤其是他改变的是View的绘制效果，View的属性没有改变，其位置与大小都不变； View动画代码量少，使用简单方便。

****Property动画：****

弥补了View动画的缺陷，你可以为一个对象的任意属性添加动画，对象自己的属性会被真的改变；当对象的属性变化的时候，属性动画会自动刷新屏幕；属性动画改变的是对象的真实属性，而且属性动画不止用于View，还可以用于任何对象。

****2、Interpolator类的意义和常用的Interpolator****

直译过来的话是补间器（插值器）的意思，它的主要作用是可以控制动画的变化速率，系统默认的Interpolator其实就是一个先加速后减速的Interpolator，

****3、ViewAnimation与属性动画有什么区别****

View动画执行之后并未改变View的真实布局属性值。切记这一点，譬如我们在Activity中有一个 Button在屏幕上方，我们设置了平移动画移动到屏幕下方然后保持动画最后执行状态呆在屏幕下方，这时如果点击屏幕下方动画执行之后的Button是没 有任何反应的，而点击原来屏幕上方没有Button的地方却响应的是点击Button的事件。

属性动画弥补了View动画的缺陷，你可以为一个对象的任意属性添加动画，对象自己的属性会被真的改变；当对象的属性变化的时候，属性动画会自动刷新屏幕；属性动画改变的是对象的真实属性，而且属性动画不止用于View，还可以用于任何对象。

1. ****如何自定义ViewAnimation****
2. ****属性动画的实现原理****

属性动画要求动画作用的对象提供该属性的set方法，属性动画根据你传递的该熟悉的初始值和最终值，以动画的效果多次去调用set方法，每次传递给set方法的值都不一样，确切来说是随着时间的推移，所传递的值越来越接近最终值。如果动画的时候没有传递初始值，那么还要提供get方法，因为系统要去拿属性的初始值。

# 六、开放平台应用

****1、是否使用过第三方平台****

****2、常用开放平台的熟悉度(微信、QQ、微博、支付宝等常用的支付和分享)****

****3、是否进行过对第三方平台的统一封装？****

****4、是否自己开发过SDK？****

# 七、设计相关

****1、有哪些常用的设计模式、设计原则****

常见设计模式介绍:

单例模式(singleton)：有些时候，允许自由创建某个类的实例没有意义，还可能造成系统性能下降。如果一个类始终只能创建一个实例，则这个类被称为单例类，这种模式就被称为单例模式。

观察者模式：当主题对象的状态发生变化时，系统能通知所有的依赖于此对象的观察者对象，从而使得观察者对象能够自动更新。

简单工厂：简单工厂模式是由一个工厂对象决定创建出哪一种产品类的实例。

工厂方法：如果我们不想在工厂类中进行逻辑判断，程序可以为不同产品类提供不同的工厂，不同的工厂类和产不同的产品。

代理模式：当客户端代码需要调用某个对象时，客户端实际上不关心是否准确得到该对象，它只要一个能提供该功能的对象即可，此时我们就可返回该对象的代理（Proxy）。

命令模式：某个方法需要完成某一个功能，完成这个功能的大部分步骤已经确定了，但可能有少量具体步骤无法确定，必须等到执行该方法时才可以确定。

策略模式：客户端程序可以自由选择其中一种算法，或让Context为客户端选择一种最佳算法——使用策略模式的优势是为了支持算法的自由切换。

门面模式：门面模式可为这些类提供一个简化的接口，从而简化访问这些类的复杂性。

桥接模式：是****把变化部分抽象出来****，使变化部分与主类分离开来，从而将多个的变化彻底分离。最后****提供一个管理类来组合不同维度上的变化****，通过这种组合来满足业务的需要。

Builder模式：将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示

（1）：单一职责原则：定义：不要存在多于一个导致类变更的原因。通俗的说，即一个类只负责一项职责。

（2）：里氏替换原则：子类可以扩展父类的功能，但不能改变父类原有的功能

（3）：依赖倒置原则：高层模块不应该依赖低层模块，二者都应该依赖其抽象；抽象不应该依赖细节；细节应该依赖抽象。

（4）：接口隔离原则：客户端不应该依赖它不需要的接口；一个类对另一个类的依赖应该建立在最小的接口上。

（5）：迪米特法则：一个对象应该对其他对象保持最少的了解。尽量降低类与类之间的耦合。

（6）：开闭原则：一个软件实体如类、模块和函数应该对扩展开放，对修改关闭。当软件需要变化时，尽量通过扩展软件实体的行为来实现变化，而不是通过修改已有的代码来实现变化。

****2、设计模式在Android源码中的应用****

****3、设计模式在项目中的应用****

****4、项目中的分包方式****

src

├ cn.eoe.app --存放程序全局性类的包

├ cn.eoe.app.adapter --存放适配器的实现类的包

├ cn.eoe.app.adapter.base --存放适配器基类的包

├[cn.eoe.app.biz](http://cn.eoe.app.biz/" \t "http://blog.csdn.net/qq_32811865/article/details/_blank) --存放ＤＡＯ类的包

├ cn.eoe.app.config －－存放常量，配置和api接口等类的包

├ cn.eoe.app.db --关于sqlite操作相关的类的包

├[cn.eoe.app.db.biz](http://cn.eoe.app.db.biz/" \t "http://blog.csdn.net/qq_32811865/article/details/_blank) --详细的增删改查类的包，暂时仅有一个类

├ cn.eoe.app.entity --实体类包

├ cn.eoe.app.entity.base --实体类基类包

├ cn.eoe.app.https --网络访问相关类的包

├ cn.eoe.app.indicator --导航相关的类包

├ cn.eoe.app.slidingmenu --滑动菜单相关类包

├ cn.eoe.app.ui --界面相关的包，activity的类

├ cn.eoe.app.ui.base --activity相关的基类包

├ cn.eoe.app.utils --工具类包

├ cn.eoe.app.view --Fragment相关类的包

├ cn.eoe.app.widget --自定义view组件包

├ com.google.zxing.camera --第三方定义，控制摄像头包

├ com.google.zxing.decoding -- 二维码图像解码包

├ com.google.zxing.view -- 自定义View，控制拍摄取景框和动画等

1. ****MVC在android中的应用，利弊****

(1)耦合性低。所谓耦合性就是模块代码之间的关联程度。利用MVC框架使得View（视图）层和Model（模型）层可以很好的分离，这样就达到了解耦的目的，所以耦合性低，减少模块代码之间的相互影响。

(2)可扩展性好。由于耦合性低，添加需求，扩展代码就可以减少修改之前的代码，降低bug的出现率。

(3)模块职责划分明确。主要划分层M,V,C三个模块，利于代码的维护。

当然一个小的项目且无需频繁修改需求就不用MVC框架来设计了，那样反而觉得代码过度设计，代码臃肿。

****6、android项目中的常见层次结构,包划分****

****7、有没有设计过项目中的层级结构、包划分****

****8、MVP与MVC的区别，MVP的优缺点****

MVC的耦合性还是较高的，View可以直接访问Model，导致3者之间构成了回路。所以两者的主要区别是，MVP中View不能直接访问Model，需要通过Presenter发出请求，View与Model不能直接通信。

优点：（1）MVP模式会解除View与Model的耦合，有效的降低View的复杂性。同时又带来了良好的可扩展性、可测试性，保证系统的整洁性和灵活性。

（2）MVP模式可以分离显示层与逻辑层，它们之间通过接口进行通信，降低耦合。理想化的MVP模式可以实现同一份逻辑代码搭配不同的显示界面，因为它们之间并不依赖与具体，而是依赖于抽象。这使得Presenter可以运用于任何实现了View逻辑接口的UI，使之具有更广泛的适用性，保证了灵活度。

缺点：1.业务复杂时，可能使得Activity变成更加复杂，比如要实现N个IView，然后写更多个模版方法。

2.业务复杂时，各个角色之间通信会变得很冗长和复杂，回调链过长。

3.Presenter处理业务，让业务变得很分散，不能全局掌握业务，很难去回答某个业务究竟是在哪里处理的。

4.用Presenter替代Controller是一个危险的做法，可能出现内存泄漏，生命周期不同步，上下文丢失等问题。

# 八、其它

****1、算法： 常用排序算法，复杂度，比较器用的哪种？快排怎么写？完全二叉树高度为n结点最多有多少，汉诺塔问题怎么解决，链表和数组比较？****

****2、操作系统： 进程冲突，生产者消费者问题，设逻辑分页和物理分页好处是什么，什么是脏内存。****

脏内存：栈内存由于反复使用，每次使用后程序不会去清理，因此分配到时保留原来的值。

****3、网络：http1.1相比以前版本有什么改变，七层/五层模型，tcpip分别对应哪层。https的对称加密。****

改变：1 可扩展性

Ø  在消息中增加版本号，用于兼容性判断。

Ø  HTTP/1.1增加了OPTIONS方法，它允许客户端获取一个服务器支持的方法列表。

Ø  为了与未来的协议规范兼容，HTTP/1.1在请求消息中包含了Upgrade头域，通过该头域，客户端可以让服务器知道它能够支持的其它备用通信协议，服务器可以据此进行协议切换，使用备用协议与客户端进行通信。

2 缓存：当缓存对象的Age超过Expire时变为stale对象，cache不需要直接抛弃stale对象，而是与源服务器进行重新激活（revalidation）。

3 带宽优化：HTTP/1.1中在请求消息中引入了range头域，它允许只请求资源的某个部分。

4 长连接：HTTP 1.1支持长连接（PersistentConnection）和请求的流水线（Pipelining）处理，在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟。

5 消息传递：发送方将消息分割成若干个任意大小的数据块，每个数据块在发送时都会附上块的长度，最后用一个零长度的块作为消息结束的标志。这种方法允许发送方只缓冲消息的一个片段，避免缓冲整个消息带来的过载。

6 Host头域：HTTP1.1的请求消息和响应消息都应支持Host头域，且请求消息中如果没有Host头域会报告一个错误（400 Bad Request）。此外，服务器应该接受以绝对路径标记的资源请求。

7 错误提示：HTTP/1.1引入了一个Warning头域，增加对错误或警告信息的描述。新增了24个状态响应码。

8 内容协商：为了满足互联网使用不同母语和字符集的用户，一些网络资源有不同的语言版本（如中文版、英文版），服务器会优先选取品质因子高的值对应的资源版本作为响应。

****4、Java: public等四个权限关键字的区别，synchronized的用法区别，可否嵌套。 hashmap底层实现，扩容策略，初始化。 arraylist和linkedlist的实现和区别。 classloader的作用，双亲委托。 gc算法(优缺点)，为什么叫新生代老年代(晋升机制)，强软弱虚四种引用的区别。****

    public,protected,friendly,private的访问权限如下：

   关键字       当前类      同一package       子孙类      其他package

    public          √                         √                          √              √

    protected       √               √                         √                          ×

    friendly        √                        √                          ×                     ×

    private         √                        ×                           ×                  ×

不写时默认为friendly

public声明的变量及方法，表明在整个包内包外都可使用。

    private声明的变量及方法，只在声明的类内可以使用。

    protected包外不可使用。包内可以使用。

   不使用关键字默认为包内使用。

1.arraylist与linklist区别

   1.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。

             2.对于随机访问get和set，ArrayList觉得优于LinkedList，因为LinkedList要移动指针。

           3.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。

****synchronized是：****Java语言的关键字，当它用来修饰一个方法或者一个代码块的时候，能够保证在同一时刻最多只有一个线程执行该段代码。

     一、当两个并发线程访问同一个对象object中的这个synchronized(this)同步代码块时，一个时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。

     二、然而，当一个线程访问object的一个synchronized(this)同步代码块时，另一个线程仍然可以访问该object中的非synchronized(this)同步代码块。

     三、尤其关键的是，当一个线程访问object的一个synchronized(this)同步代码块时，其他线程对object中所有其它synchronized(this)同步代码块的访问将被阻塞。

     四、第三个例子同样适用其它同步代码块。也就是说，当一个线程访问object的一个synchronized(this)同步代码块时，它就获得了这个object的对象锁。结果，其它线程对该object对象所有同步代码部分的访问都被暂时阻塞。

     五、以上规则对其它对象锁同样适用.

A. 无论synchronized关键字加在方法上还是对象上，如果它作用的对象是非静态的，则它取得的锁是对象；如果synchronized作用的对象是一个静态方法或一个类，则它取得的锁是对类，该类所有的对象同一把锁。

B. 每个对象只有一个锁（lock）与之相关联，谁拿到这个锁谁就可以运行它所控制的那段代码。

C. 实现同步是要很大的系统开销作为代价的，甚至可能造成死锁，所以尽量避免无谓的同步控制。

****5、android: activity退出怎么保存数据。 怎么把数据写入文件。 picasso的缓存策略，lrucache底层实现，linkedhashmap底层实现，缓存文件怎么命名。 RxJava优缺点，实习项目相关。 自定义view有几个构造方法，第三个参数作用。 listview的convert view作用，用viewholder为什么可以优化他****。

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中的所有信息和状态数据都不会消失，当activity重新回到前台后，所有的改变都会保留。

但是当内存系统内存不足时，调用onPause()和onStop()方法的activity可能被摧毁。此时内存中就不会存在有该activity实例对象了。

为了避免这种情况，我们可以覆盖onSaveInstanceState()方法来接受一个Bundle类型的参数，我们可以将该activity的信息和状态数据存入此Bundle中。这样，即使该activity在后台被系统摧毁，然后重新回到前台的时候，在调用onCreate()方法时，该Bundle对象会传入onCreate方法的参数中。开发者可以在onCreate中取出自己保存的数据。

在Adapter中，先判断convertview是否为空，若为空，则直接加载布局，若不为空，则直接使用布局，这能够很有效的使用Android为listview提供的缓存机制：只加载一屏的布局，之后滑动出来的item使用的是之前已经加载的布局的缓存；使用静态的ViewHoulder的目的则是节省了findViewById的时间。

****6、app被杀死怎么启动****

****7、耗电太多怎么破****

****8、怎么统计crash****

****9、怎么减少用户流量消耗****

****10、事件分发机制，ontouchlistener返回false才会调用onclicklistener****

****11、方法数超过65535怎么办****

谷歌官方推荐使用MultiDexApplication,

****12、binder机制****

****13、如何避免anr****

    使用AsyncTask处理耗时IO操作，

    使用Thread或者handlerthread提高优先级

    使用handler来处理工作线程的耗时任务

1：UI线程尽量只做跟UI相关的工作

2：耗时的工作（比如数据库操作，I/O，连接网络或者别的有可能阻碍UI线程的操作）把它放入单独的线程处理

3：尽量用Handler来处理UIthread和别的thread之间的交互

****14、listview优化****

1.在adapter中的getView方法中尽量少使用逻辑

2.尽最大可能避免GC

3.滑动的时候不加载图片

4.将ListView的scrollingCache和animateCache设置为false

5.item的布局层级越烧越好

6.使用ViewHolder

****15、bitmap怎么避免oom****

第一种方法：按比例缩小bitmap

第二种方法：按指定大小来缩小bitmap

****16、Java静态内部类和内部类的区别****

1）静态内部类可以有静态成员（方法和属性），而非静态内部类则不能有静态成员（成员或属性）

2）静态内部类只能够访问外部类的静态成员和静态方法，而非静态内部类则可以访问外部类的所有成员（方法和属性）

3）实例化一个非静态的内部类的方法：

     OutClass.InnerClass  innerClass = new OutClass().new InnerClass();

4）实例化一个静态内部类的方法：

     不依赖于外部类的实例，直接实例化静态内部类对象：

      OutClass,InnerClass innerClass =  new OutClass.InnerClass();

****17、 retrofit原理****

****18、recyclerview和listview异同****

1 缓存机制对比：RecyclerView比ListView多两级缓存，支持多个离ItemView缓存，支持开发者自定义缓存处理逻辑，支持所有RecyclerView共用同一个RecyclerViewPool(缓存池)。

2 局部刷新：通过局部刷新，就能避免调用许多无用的bindView.

****列表页展示界面，需要支持动画，或者频繁更新，局部刷新，建议使用RecyclerView，更加强大完善，易扩展；其它情况(如微信卡包列表页)两者都OK，但ListView在使用上会更加方便，快捷。****

****19、singletask启动standard的activity在哪个栈****

****20、android多进程和多线程的实现，进程和线程区别。****

****21、java泛型类型擦除发生在什么时候，通配符有什么需要注意的。****

****22、 RxJava优点，map，flatmap的原理。 可不可以多次subscribeOn，ObserveOn，会有什么后果。****

****23、 lambda表达式？****

****24、http协议和https，ssl和tls握手。****

****25、tcp三次握手的过程？如果确认信号没传到服务器会发生什么？为什么不是两次握手？****

****26、一个无序数组怎么找出两个和为特定值的数？快排后首尾两游标。****

****27、开发过程中有没有实际遇到内存泄露情况，怎么解决的。****

****28、socket相关****

****29、什么叫高内聚低耦合？如何编写重构的代码？****

****30、有几种排序法、请写出冒泡排序的伪代码****

|  |  |
| --- | --- |
|  | bubblesort(A)  {     for i = 1 to length[A]     {         for j = length[A] to i+1         {             if A[j] < A[j-1]             {                exchane A[j] and A[j-1];             }         }     }  } |

****31、怎么使键盘弹出的时候edittext的光标始终处于最后一行。****

****32、listview上面有个浮动窗口下拉刷新的时候会挡住item，如何解决。****

****33、安卓在网络请求的时候,服务器已经返回数据了,但是手机突然断网,没接收到数据,如何处理?****

****34、当用户进入一个页面时，该页面正在loading网络数据时，用户快速点击了返回按钮，如何处理？****

****35、进程间如何通信、AIDL的实现原理。****

****36、如何把SharedPreferences的数据共享给其它进程****