# Android基础框架

## Activity

### 操作系统如何管理内存的：

1. 使用寄存器进行将进程地址和物理内存进行映射
2. 虚拟内存进行内存映射到硬盘上增大内存
3. 虚拟内存是进行内存分页管理
4. 页表实现分页，就是 页+地址偏移。
5. 如果程序的内存在硬盘上，那么就需要用页置换算法来将其调入内存中：先进先出、最近未使用最少等等
6. 博客推荐：现代操作系统部分章节笔记

### 对android系统的理解（概率60%）

5大优势：

1. 开放性
2. 挣脱运营商的束缚
3. 丰富的硬件选择
4. 不受任何限制的开发商
5. 无缝结合的Google应用

Android的5大不足：

1. 安全和隐私
2. 首先开卖Android手机的不是最大运营商
3. 运营商仍然能够影响到Android手机
4. 同类机型用户减少
5. 过分依赖开发商缺少标准配置

### 安卓的系统架构（概率40%）

1. http://wenku.baidu.com/view/3ef50eddd15abe23482f4d7e.html
2. 应用层：使用Java语言进行开发的一些应用程序
3. 应用框架层：主要是谷歌发布的一些操作支持的类库（API框架），开发人员可以使用这些类库方便的进行程序开发，但是在开发时必须遵守框架的开发原则
4. 系统运行库层：当使用安卓框架层进行开发时，安卓操作系统会自动使用一些c/c++的库文件来支持所使用的各个组件，使其可以更好的为程序服务；
5. Linux内核层：安卓操作系统主要是基于Linux内核，程序的安全性，驱动程序，进程管理等都由Linux内核提供。

### 6android虚拟机的进程和linux的关系（概率20%）

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

### 160. Dvm是什么（概率30%）

1. Android 运行环境主要指的虚拟机技术——Dalvik。Android中的所有Java程序都是运行在Dalvik VM上的。Android上的每个程序都有自己的线程，DVM只执行.dex的Dalvik executable 文件。每个Android应用在底层都对应有一个独立的DVM实例并在其解释下执行。
2. 36.android的IPC通信方式，线程（进程间）通信机制有哪些
3. 1.ipc通信方式：binder、contentprovider、socket
4. 2.操作系统进程通讯方式：共享内存、socket、管道

### ipc机制（及原理）（概率70%）

1. ）IPC（Inter-Process Communication，进程间通信）,
2. aidl是 Android Interface definition language的缩写，它是一种android内部进程通信接口的描述语言，通过它我们可以定义进程间的通信接口.编译器可以通过扩展名为aidl的文件生成一段代码，通过预先定义的接口达到两个进程内部通信进程的目的.
3. BroadcastReceiver也可以实现进程间通信
4. ContentProvider 提供进程间数据共享

### 2).Android的 Service工作流程

1. 1.Android虚拟机启动之前系统会先启动service Manager进程；
2. 2.service Manager打开binder驱动，并通知binder kernel驱动程序这个进程将作为System Service Manager；
3. 3.然后该进程将进入一个循环，等待处理来自其他进程的数据。
4. 4.用户创建一个System service后，通过defaultServiceManager得到一个远程ServiceManager的接口，通过这个接口我们可以调用addService函数将System service添加到Service Manager进程中；
5. 5.然后client可以通过getService获取到需要连接的目的Service的IBinder对象，这个IBinder是Service的BBinder在binder kernel的一个参考，
6. 6.所以service IBinder 在binder kernel中不会存在相同的两个IBinder对象，每一个Client进程同样需要打开Binder驱动程序。对用户程序而言，我们获得这个对象就可以通过binder kernel访问service对象中的方法。
7. 7.Client与Service在不同的进程中，通过这种方式实现了类似线程间的迁移的通信方式，对用户程序而言当调用Service返回的IBinder接口后，访问Service中的方法就如同调用自己的函数。

### 16.Android IPC:Binder原理

1. 1.在Activity和Service进行通讯的时候，用到了Binder。
2. 1.当属于同个进程我们可以继承Binder然后在Activity中对Service进行操作
3. 2.当不属于同个进程，那么要用到AIDL让系统给我们创建一个Binder，然后在Activity中对远端的Service进行操作。
4. 2.系统给我们生成的Binder：
5. 1.Stub类中有:接口方法的id，有该Binder的标识，有asInterface(IBinder)(让我们在Activity中获取实现了Binder的接口，接口的实现在Service里，同进程时候返回Stub否则返回Proxy)，有onTransact()这个方法是在不同进程的时候让Proxy在Activity进行远端调用实现Activity操作Service
6. 2.Proxy类是代理，在Activity端，其中有:IBinder mRemote(这就是远端的Binder)，两个接口的实现方法不过是代理最终还是要在远端的onTransact()中进行实际操作。
7. 3.哪一端的Binder是副本，该端就可以被另一端进行操作，因为Binder本体在定义的时候可以操作本端的东西。所以可以在Activity端传入本端的Binder，让Service端对其进行操作称为Listener，可以用RemoteCallbackList这个容器来装Listener，防止Listener因为经历过序列化而产生的问题。
8. 4.当Activity端向远端进行调用的时候，当前线程会挂起，当方法处理完毕才会唤醒。
9. 5.如果一个AIDL就用一个Service太奢侈，所以可以使用Binder池的方式，建立一个AIDL其中的方法是返回IBinder，然后根据方法中传入的参数返回具体的AIDL。
10. 6.IPC的方式有：Bundle（在Intent启动的时候传入，不过是一次性的），文件共享(对于SharedPreference是特例，因为其在内存中会有缓存)，使用Messenger(其底层用的也是AIDL，同理要操作哪端，就在哪端定义Messenger)，AIDL，ContentProvider(在本进程中继承实现一个ContentProvider，在增删改查方法中调用本进程的SQLite，在其他进程中查询)，Socket

### 17.描述一次跨进程通讯

1. 1.client、proxy、serviceManager、BinderDriver、impl、service
2. 2.client发起一个请求service信息的Binder请求到BinderDriver中，serviceManager发现BinderDiriver中有自己的请求 然后将clinet请求的service的数据返回给client这样完成了一次Binder通讯
3. 3.clinet获取的service信息就是该service的proxy，此时调用proxy的方法，proxy将请求发送到BinderDriver中，此时service的 Binder线程池循环发现有自己的请求，然后用impl就处理这个请求最后返回，这样完成了第二次Binder通讯
4. 4.中间client可挂起，也可以不挂起，有一个关键字oneway可以解决这个

### 209.IPC是什么（概率70%）

1. IPC（Inter-Process Communication，进程间通信）,aidl是 Android Interface definition language的缩写，它是一种android内部进程通信接口的描述语言，通过它我们可以定义进程间的通信接口.编译器可以通过扩展名为aidl的文件生成一段代码，通过预先定义的接口达到两个进程内部通信进程的目的.
2. BroadcastReceiver也可以实现进程间通信
3. ContentProvider 提供进程间数据共享
4. 62.Launcher进程启动另外一个进程的过程：启动一个app

### 18.android重要术语解释

1. 1.ActivityManagerServices，简称AMS，服务端对象，负责系统中所有Activity的生命周期
2. 2.ActivityThread，App的真正入口。当开启App之后，会调用main()开始运行，开启消息循环队列，这就是传说中的UI线程或者叫主线程。与ActivityManagerServices配合，一起完成Activity的管理工作
3. 3.ApplicationThread，用来实现ActivityManagerService与ActivityThread之间的交互。在ActivityManagerService需要管理相关Application中的Activity的生命周期时，通过ApplicationThread的代理对象与ActivityThread通讯。
4. 4.ApplicationThreadProxy，是ApplicationThread在服务器端的代理，负责和客户端的ApplicationThread通讯。AMS就是通过该代理与ActivityThread进行通信的。
5. 5.Instrumentation，每一个应用程序只有一个Instrumentation对象，每个Activity内都有一个对该对象的引用。Instrumentation可以理解为应用进程的管家，ActivityThread要创建或暂停某个Activity时，都需要通过Instrumentation来进行具体的操作。
6. 6.ActivityStack，Activity在AMS的栈管理，用来记录已经启动的Activity的先后关系，状态信息等。通过ActivityStack决定是否需要启动新的进程。
7. 7.ActivityRecord，ActivityStack的管理对象，每个Activity在AMS对应一个ActivityRecord，来记录Activity的状态以及其他的管理信息。其实就是服务器端的Activity对象的映像。
8. 8.TaskRecord，AMS抽象出来的一个“任务”的概念，是记录ActivityRecord的栈，一个“Task”包含若干个ActivityRecord。AMS用TaskRecord确保Activity启动和退出的顺序。如果你清楚Activity的4种launchMode，那么对这个概念应该不陌生。

### 151. Android项目的目录结构（概率30%）

1. 1)src目录：src（即SourceCode)，src目录用来存放应用程序中所有的源代码，代码的源文件一般存放在该目录下的相应的包下。
2. 2)gen目录：gen目录，该目录下一般只有一个文件，即R文件。该文件夹下面有个R.java文件，R.java是在建立项目时自动生成的，定义该项目所有资源的索引。
3. 3)Android4.1目录：该文件夹下包含android.jar文件，这是一个Java归档文件，其中包含构建应用程序所需的所有的AndroidSDK库（如Views、Controls）和APIs。
4. 通过android.jar将自己的应用程序绑定到AndroidSDK和AndroidEmulator，这允许你使用所有Android的库和包，且使你的应用程序在适当的环境中调试。
5. 4)assets目录：该目录存放应用程序中使用的外部资源文件，如声音视频等，在Android程序中可以通过输入/输出流对该目录中的文件进行读写。
6. 5)res目录：该目录用来存放程序这用到的图片、界面布局文件及XML格式的描述文件。该目录下有多个目录。新建一Android项目，在res目录下一般会有drawable,layout,valuse三个目录。drawable用来存放图片资源,目录下一般会有三个目录分别存放高中低三种不同分辨率的图片，layout目录用来存放应用程序的界面布局文件，values目录下存放字符串资源，颜色资源，数组资源等XML文件。
7. 6)AndroidManifest.xml

### 64.程序运行的权限和文件系统权限的区别（概率10%）

1. 运行时权限 Dalvik( android 授权) 文件系统 linux 内核授权

### 配文件中存放什么？（概率60%）

1. 版本号、权限，四大组件，意图，第三方key等
2. 182. Manifest中放什么东西（概率60%）
3. AndroidManifest.xml是每个android程序中必须的文件。它位于整个项目的根目录，描述了package中暴露的组件（activities,services,等等），他们各自的实现类，各种能被处理的数据和启动位置。
4. 除了能声明程序中的Activities,ContentProviders,Services,和IntentReceivers,还能指定
5. permissions和instrumentation（安全控制和测试）

### 198.Android程序的入口（概率30%）

1. action节点中的android.intent.action.MAIN表明它所在的Activity是整个应用程序的入口点

## 四大组件

### Application类的作用（概率60%）

1. Application和Activity、Service一样是android框架的一个系统组件，当android程序启动时系统会创建一个 application对象，用来存储系统的一些信息。通常我们是不需要指定一个Application的，这时系统会自动帮我们创建，如果需要创建自己的Application，也很简单创建一个类继承 Application并在manifest的application标签中进行注册(只需要给Application标签增加个name属性把自己的 Application的名字定入即可)。
2. android系统会为每个程序运行时创建一个Application类的对象且仅创建一个，所以Application可以说是单例 (singleton)模式的一个类。且application对象的生命周期是整个程序中最长的，它的生命周期就等于这个程序的生命周期。因为它是全局的单例的，所以在不同的Activity、Service中获得的对象都是同一个对象。所以通过Application来进行一些，数据传递、数据共享、数据缓存等操作。

### 理解Window和WindowManager

1. Window用于显示View和接收各种事件，Window有三种类型：应用Window(每个Activity对应一个Window)、子Window(不能单独存在，附属于特定Window)、系统window(Toast和状态栏)
2. Window分层级，应用Window在1-99、子Window在1000-1999、系统Window在2000-2999，WindowManager提供了增删改View三个功能。
3. Window是个抽象概念：每一个Window对应着一个View和ViewRootImpl，Window通过ViewRootImpl来和View建立联系，View是Window存在的实体，只能通过WindowManager来访问Window。
4. WindowManager的实现是WindowManagerImpl其再委托给WindowManagerGlobal来对Window进行操作，其中有四个List分别储存对应的View、ViewRootImpl、WindowManger.LayoutParams和正在被删除的View
5. Window的实体是存在于远端的WindowMangerService中，所以增删改Window在本端是修改上面的几个List然后通过ViewRootImpl重绘View，通过WindowSession(每个应用一个)在远端修改Window。
6. Activity创建Window：Activity会在attach()中创建Window并设置其回调(onAttachedToWindow()、dispatchTouchEvent()),Activity的Window是由Policy类创建PhoneWindow实现的。然后通过Activity#setContentView()调用PhoneWindow的setContentView。

### Android四大组件的加载过程，请详细介绍下

1. ??????????????????????????????????

### Activity的生命周期（概率80%）

（6+1）onCreate，onStart，onResume，onPause，onStop，onDestroy；onRestart。

### 横竖屏切换Activity周期的变化（概率30%）

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"时，切屏不会重新调用各个生命周期，只会执行onConfigurationChanged方法

### 44.Activity的启动模式（四种）（概率100%）

1. Standard(每次创建一个新的activity)、singleTop()、singleTask、singleInstance
2. standard
3. 默认模式，可以不用写配置。在这个模式下，都会默认创建一个新的实例。因此，在这种模式下，可以有多个相同的实例，也允许多个相同Activity叠加。
4. singleTop
5. 可以有多个实例，但是不允许多个相同Activity叠加。即，如果Activity在栈顶+的时候，启动相同的Activity，不会创建新的实例，而会调用其onNewIntent方法。
6. singleTask
7. 只有一个实例。在同一个应用程序中启动他的时候，若Activity不存在，则会在当前task创建一个新的实例，若存在，则会把task中在其之上的其它Activity destory掉并调用它的onNewIntent方法。
8. 如果是在别的应用程序中启动它，则会新建一个task，并在该task中启动这个Activity，singleTask允许别的Activity与其在一个task中共存，也就是说，如果我在这个singleTask的实例中再打开新的Activity，这个新的Activity还是会在singleTask的实例的task中。
9. singleInstance
10. 只有一个实例，并且这个实例独立运行在一个task中，这个task只有这个实例，不允许有别的Activity存在。

### Activity的启动模式有哪些？（概率90%）

1. standard，每次激活activity时，都创建activity实例，并放入任务栈。
2. singleTop，如果activity自己激活自己，即任务栈栈顶就是该activity，则不需要创建，其余情况都要创建activity实例。
3. singleTask，如果要激活的那个activity在任务栈中存在该实例，则不需要创建，只需要把此activity放入栈顶，并把该activity以上的activity实例都pop；
4. singleInstance，如果应用1的任务栈中创建了MainActivity实例，如果应用2也要激活MainActivity，则不需要创建，两应用共享该Activity实例；

### 如何让Activity变成一个窗口？（概率20%）

1. 答：在清单文件AndroidManifest.xml中相应的<activity>标签内设置属性
2. android :theme="@android:style/Theme.Dialog"

### 后台Activity被回收如何恢复（概率60%）

1. 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
2. 6.Activity缓存方法
3. 1.配置改变导致Activity被杀死，横屏变竖屏：在onStop之前会调用onSaveInstanceState()保存数据在重建Activity之后，会在onStart()之后调用onRestoreInstanceState(),并把保存下来的Bundle传给onCreate()和它会默认重建Activity当前的视图，我们可以在onCreate()中，回复自己的数据。
4. 2.内存不足杀掉Activity，优先级分别是：前台可见，可见非前台，后台。
5. 178. 简单介绍服务（概率30%）
6. 服务是没有界面的长生命周期的代码。一个很好的例子是媒体播放器从列表中播放歌曲。在一个媒体播放器程序中，大概要有一个或多个活动（activity）来供用户选择歌曲并播放它。然而，音乐的回放就不能使用活动（activity）了，因为用户希望他导航到其他界面时音乐继续播放。这种情况下，媒体播放器活动（activity）要用Context.startService()启动一个服务来在后台运行保持音乐的播放。系统将保持这个音乐回放服务的运行直到它结束。注意一下，你要用Context.bindService()方法连接服务（如果它没有运行，要先启动它）。当连接到服务后，你可以通过服务暴露的一个接口和它通信。对于音乐服务，它允许你暂停、倒带，等等。
7. 173. Service用在哪个线程（概率70%）
8. 默认情况下Service是运行在启动该Service的应用主线程的，如果Service中的操作占用大量的CPU资源或有阻断操作（比如播放MP3或者访问网络）会影响应用主线程的响应性能，甚至会造成“应用程序无响应(ANR)”问题。
9. 62.Service是如何启动（概率40%）
10. 1通过startService
11. Service会经历onCreate->onStart
12. stopService的时候直接onDestroy
13. 如果是调用者(TestServiceHolder)自己直接退出而没有调用stopService的
14. 话，Service会一直在后台运行。
15. 下次TestServiceHolder再起来可以stopService。
16. 2通过bindService
17. Service只会运行onCreate，这个时候TestServiceHolder和TestService绑定在一起
18. TestServiceHolder退出了，Srevice就会调用onUnbind->onDestroyed
19. 所谓绑定在一起就共存亡了。
20. 154. 如何启用service，如何停用service？（概率60%）
21. 答案：①：通过调用Context.startservice()启动，调用Context.stopservice()结束，startservice()可以传递参数给service.
22. ②：通过调用context.bindservice()启动，调用Context.unbindservice()结束，还可以通过serviceConnection访问service。
23. 119.Service实现的两种启动方式（概率60%）
24. 1.Context.startService()方式启动
25. ①Context.startService()方式的生命周期： 启动时，startService –> onCreate() –> onStart()停止时，stopService –> onDestroy()如果调用者直接退出而没有停止Service，则Service 会一直在后台运行 Context.startService()方法启动服务，在服务未被创建时，系统会先调用服务的onCreate()方法，接着调用onStart()方法。如果调用startService()方法前服务已经被创建，多次调用startService()方法并不会导致多次创建服务，但会导致多次调用onStart()方法。采用startService()方法启动的服务，只能调用Context.stopService()方法结束服务，服务结束时会调用onDestroy()方法附代码
26. 2.Context.bindService()方式启动：①Context.bindService()方式的生命周期： 绑定时,bindService -> onCreate() –> onBind()调用者退出了，即解绑定时,Srevice就会unbindService –>onUnbind() –> onDestory()Context.bindService()方式启动 Service的方法：绑定Service需要三个参数：bindService(intent, conn, Service.BIND\_AUTO\_CREATE);第一个：Intent对象第二个：ServiceConnection对象，创建该对象要实现它的onServiceConnected()和 onServiceDisconnected()来判断连接成功或者是断开连接第三个：如何创建Service，一般指定绑定的时候自动创建附代码
27. 153. IntentService有何优点? （概率80%）
28. 答案：IntentService使用队列的方式将请求的Intent加入队列，然后开启一个worker thread(线程)来处理队列中的Intent，对于异步的startService请求，IntentService会处理完成一个之后再处理第二个，每一个请求都会在一个单独的worker thread中处理，不会阻塞应用程序的主线程，这里就给我们提供了一个思路，如果有耗时的操作与其在Service里面开启新线程还不如使用IntentService来处理耗时操作。
29. 49.IntentService是什么
30. 1.含有HandlerThread的Service，可以多次startService()来多次在子线程中进行 onHandlerIntent()的调用。
31. 152.Intentservice的用法（概率80%）
32. ？？？？
33. 8.怎么保证service不被杀死
34. 1.提升service优先级
35. 2.提升service进程优先级
36. 3.onDestroy方法里重启service
37. 156. AIDL全称是什么？能处理哪些数据？（概率40%）
38. 答案：全称是：Android Interface Definition Landuage是一种街口描述语言；编译器可以通过aidl文件生成一段代码，通过预先定义的接口达到两个进程内部通信进程跨界对象访问的目的，aidl的IPC的机制和COM或CORBA类似，是基于接口的，是轻量级的。使用代理类在客户端和实现层间接传递值。

### 37·AIDL的全称是什么，如何工作（概率20%）

1. AIDL的英文全称是Android Interface Define Language
2. 当A进程要去调用B进程中的service时，并实现通信，我们通常都是通过AIDL来操作的
3. A工程：
4. 首先我们在net.blogjava.mobile.aidlservice包中创建一个RemoteService.aidl文件，在里面我们自定义一个接口，含有方法get。ADT插件会在gen目录下自动生成一个RemoteService.java文件，该类中含有一个名为RemoteService.stub的内部类，该内部类中含有aidl文件接口的get方法。
5. 说明一：aidl文件的位置不固定，可以任意
6. 然后定义自己的MyService类，在MyService类中自定义一个内部类去继承RemoteService.stub这个内部类，实现get方法。在onBind方法中返回这个内部类的对象，系统会自动将这个对象封装成IBinder对象，传递给他的调用者。
7. 其次需要在AndroidManifest.xml文件中配置MyService类，代码如下：
8. <!-- 注册服务 -->
9. <service android:name=".MyService">
10. <intent-filter>
11. <!-- 指定调用AIDL服务的ID -->
12. <action android:name="net.blogjava.mobile.aidlservice.RemoteService" />
13. </intent-filter>
14. </service>
15. 为什么要指定调用AIDL服务的ID,就是要告诉外界MyService这个类能够被别的进程访问，只要别的进程知道这个ID，正是有了这个ID,B工程才能找到A工程实现通信。
16. 说明：AIDL并不需要权限
17. B工程：
18. 首先我们要将A工程中生成的RemoteService.java文件拷贝到B工程中，在bindService方法中绑定aidl服务
19. 绑定AIDL服务就是将RemoteService的ID作为intent的action参数。
20. 说明：如果我们单独将RemoteService.aidl文件放在一个包里，那个在我们将gen目录下的该包拷贝到B工程中。如果我们将RemoteService.aidl文件和我们的其他类存放在一起，那么我们在B工程中就要建立相应的包，以保证RmoteService.java文件的报名正确，我们不能修改RemoteService.java文件
21. bindService(new Inten("net.blogjava.mobile.aidlservice.RemoteService"), serviceConnection, Context.BIND\_AUTO\_CREATE);
22. ServiceConnection的onServiceConnected(ComponentName name, IBinder service)方法中的service参数就是A工程中MyService类中继承了RemoteService.stub类的内部类的对象。
23. Android采用RPC的方式来实现（remote procedure call）远程通信，并且Android通过接口定义语言AIDL来生成两个进程之间互相访问的代码。例如，你在Activity里的代码需要访问Service中的一个方法，那么就可以通过这种方法来实现了。
24. ?创建一个AIDL文件（参见代码）
25. ?实现AIDL文件生成的Java接口（参见代码）
26. ?将你的接口暴露给客户端
27. ?客户端调用
28. 221.AIDL（AIDL文件由谁来写 服务器的service满足什么条件 声明的时候加什么权限）（概率50%）
29. AIDL Android Interface definition language，在服务端定义aidl文件 自动生成java文件 然后在service中实现这个aidl 在onbind中返回这个对象 在客户端把服务端的aidl文件完全复制过来 包名必须完全一致 在onServiceConnected方法 中 把 Ibinder对象 用asInterface方法转化成 aidl对象 然后调用方法即可
30. 服务器的service需要添加intent-filter 客户端类要通过 bindService 方式来启动另外一个进程的服务，这样才能实现和服务进行交互。如果通过startService方式来启动服务，则不能与服务进行交互
31. 有时候我们不想所有的程序都可以访问我们的远程服务 所以可以给服务设置权限和过滤：
32. 1.我们在onbind中进行校验 用某种方式 如果验证不通过那么就直接返回null
33. 2.我们可以在服务端的AndroidMiniFest.xml中 设置所需的权限 <permission android:name="aaaaaa" android:protectionLevel="normal"/>
34. 然后在onbind中 检查是否有这个权限了 如果没有那么直接返回null即可
35. (权限非必须加)
36. 45.广播机制的作用（概率40%）
37. 广播机制，本质上它就是一种组件间的通信方式，如果是两个组件位于不同的进程当中，那么可以用Binder机制来实现，如果两个组件是在同一个进程中，那么它们之间可以用来通信的方式就更多了，这样看来，广播机制似乎是多余的。然而，广播机制却是不可替代的，它和Binder机制不一样的地方在于，广播的发送者和接收者事先是不需要知道对方的存在的，这样带来的好处便是，系统的各个组件可以松耦合地组织在一起，这样系统就具有高度的可扩展性，容易与其它系统进行集成。
38. 实现思路：
39. 在第1步中，广播的接收者把广播接收器注册到ActivityManagerService中；在第2步中，广播的发送者同样是把广播发送到ActivityManagerService中，由ActivityManagerService去查找注册了这个广播的接收者，然后把广播分发给它们。
40. 在第2步的分发的过程，其实就是把这个广播转换成一个消息，然后放入到接收器所在的线程消息队列中去，最后就可以在消息循环中调用接收器的onReceive函数了。这里有一个要非常注意的地方是，由于ActivityManagerService把这个广播放进接收器所在的线程消息队列后，就返回了，它不关心这个消息什么时候会被处理，因此，对广播的处理是异步的，即调用sendBroadcast时，这个函数不会等待这个广播被处理完后才返回
41. 实例连接：http://wenku.baidu.com/view/b73dd968af1ffc4ffe47ac0c.html

### 33·注册广播的方式（概率30%）

1. android中，不同进程之间传递信息要用到广播，可以有两种方式来实现。
2. 第一种方式：在Manifest.xml中注册广播，是一种比较推荐的方法，因为它不需要手动注销广播（如果广播未注销，程序退出时可能会出错）。
3. 具体实现在Manifest的application中添加：
4. <receiver android:name=".mEvtReceiver">
5. <intent-filter>
6. <action android:name="android.intent.action.BOOT\_COMPLETED" />
7. </intent-filter>
8. </receiver>
9. 上面两个android:name分别是广播名和广播的动作（这里的动作是表示系统启动完成），如果要自己发送一个广播，在代码中为：
10. Intent i = new Intent("android.intent.action.BOOT\_COMPLETED");
11. sendBroadcast(i);
12. 这样，广播就发出去了，然后是接收。
13. 接收可以新建一个类，继承至BroadcastReceiver，也可以建一个BroadcastReceiver的实例，然后得写onReceive方法，实现如下：
14. protected BroadcastReceiver mEvtReceiver = new BroadcastReceiver() {
15. @Override
16. public void onReceive(Context context, Intent intent) {
17. String action = intent.getAction();
18. if (action.equals("android.intent.action.BOOT\_COMPLETED")) {
19. //Do something
20. }
21. }
22. };
23. 第二种方式，直接在代码中实现，但需要手动注册注销，实现如下：
24. IntentFilter filter = new IntentFilter();
25. filter.addAction("android.intent.action.BOOT\_COMPLETED");
26. registerReceiver(mEvtReceiver, filter); //这时注册了一个recevier ,名为mEvtReceiver，然后同样用上面的方法以重写onReceiver，
27. 最后在程序的onDestroy中要注销广播，实现如下：
28. @Override
29. public void onDestroy() {
30. super.onDestroy();
31. unregisterReceiver(mPlayerEvtReceiver);
32. }
33. Android系统中的广播是广泛用于应用程序之间通信的一种手段，它类似于事件处理机制，不同的地方就是广播的处理是系统级别的事件处理过程（一般事件处理是控件级别的）。在此过程中仍然是离不开Intent对象，理解广播事件的处理过程，灵活运用广播处理机制，在关键之处往往能实现特别的效果，
34. 在Android 中如果要发送一个广播必须使用sendBroadCast 向系统发送对其感兴趣的广播接收器中。
35. 使用广播必须要有一个intent 对象必设置其action动作对象
36. 使用广播必须在配置文件中显式的指明该广播对象
37. 每次接收广播都会重新生成一个接收广播的对象
38. 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
39. 11广播的种类

### 36·ContentProvider的URI的配置？（概率20%）

1. 清单文件之指定URI或者代码里面指定URI，contentProvider通过URI访问数据
2. 73、contentprovider怎么实现数据共享（概率40%）
3. 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。
4. 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。
5. 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？
6. 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。
7. 175.Android如何访问自定义ContentProvider（概率30%）
8. 1.第一：得到ContentResolver类对象：ContentResolver cr = getContentResolver（）；
9. 2. 第二：定义要查询的字段String数组。
10. 3. 第三：使用cr.query();返回一个Cursor对象。
11. 4. 第四：使用while循环得到Cursor里面的内容。
12. 174. ContentProvider和SQL的区别（概率30%）
13. Sql只能在该工程的内部共享数据，ContentProvider能在工程之间实现数据共享。
14. 10.framgment生命周期？（概率30%）
15. MyFragment onAttach() 粘贴到activity上
16. MyFragment onCreate() fragment创建
17. MyFragment onCreateView() fragment创建自己的视图
18. MainActivity onCreate()
19. MyFragment onActivityCreated() 可以处理fragment数据的初始化
20. MainActivity onStart()
21. MyFragment onStart()
22. MainActivity onResume()
23. MyFragment onResume()
24. 按后退键
25. MyFragment onPause()
26. MainActivity onPause()
27. MyFragment onStop()
28. MainActivity onStop()
29. MyFragment onDestoryView() 销毁掉自己的视图
30. MyFragment onDestory()
31. MyFragment onDetach() 解除和activity的关系
32. MainActivity onDestory()
33. 59.Intent可以被传递的数据类型（概率70%）
34. intent是连接Activity, Service, BroadcastReceiver, ContentProvider四大组件的信使,，可以传递八种基本数据类型以及string, Bundle类型，以及实现了Serializable(java)或者Parcelable(安卓)的类型。
35. Intent可以划分成显式意图和隐式意图。
36. 显式意图：调用Intent.setComponent()或Intent.setClass()方法明确指定了组件名的Intent为显式意图，显式意图明确指定了Intent应该传递给哪个组件。
37. 隐式意图：没有明确指定组件名的Intent为隐式意图。 Android系统会根据隐式意图中设置的动作(action)、类别(category)、数据（URI和数据类型）找到最合适的组件来处理这个意图。
38. 85、Intent的原理（概率80%）
39. intent是连接Activity, Service, BroadcastReceiver, ContentProvider四大组件的信使,，可以传递八种基本数据类型以及string, Bundle类型，以及实现了Serializable或者Parcelable的类型。
40. Intent可以划分成显式意图和隐式意图。
41. 显式意图：调用Intent.setComponent()或Intent.setClass()方法明确指定了组件名的Intent为显式意图，显式意图明确指定了Intent应该传递给哪个组件。
42. 隐式意图：没有明确指定组件名的Intent为隐式意图。 Android系统会根据隐式意图中设置的动作(action)、类别(category)、数据（URI和数据类型）找到最合适的组件来处理这个意图。
43. 87、intent和activity的区别（概率40%）
44. Activity活动，可以理解为窗口，是人机交互的核心，所以又被人们说成是所有程序的根本。Intent意图，负责所有Activity窗口通讯。Service服务是相对Activity来说的，不需要人机交互，但可以为Activity提供交互必需要的一些东西。
45. Android中，Activity是所有程序的根本，所有程序的流程都运行在Activity之中，Activity具有自己的生命周期. 由系统控制生命周期，程序无法改变，但可以用onSaveInstanceState保存其状态对于Activity，关键是其生命周期的把握（如下图），其次就是状态的保存和恢复（onSaveInstanceState onRestoreInstanceState），以及Activity之间的跳转和数据传输（intent）。
46. Intent
47. Android中提供了Intent机制来协助 应用间的交互与通讯，Intent负责对应用中一次操作的动作、动作涉及数据、附加数据进行描述，Android则根据此Intent的描述，负责找到对 应的组件，将 Intent传递给调用的组件，并完成组件的调用。Intent不仅可用于应用程序之间，也可用于应用程序内部的Activity/Service之间的交互。
48. 213. Android 如何指定浏览器访问 指定页面? （概率30%）
49. 指定action 及 uri 就可以访问制定好的 浏览器了
50. 176.Uri有什么组成（概率30%）
51. 答：URI由是部分组成：
52. A，标准的前缀: ContentProvider（内容提供者）的scheme已经由Android所规定， scheme为：content://
53. B，唯一标识整个Content Provider: 主机名（或叫Authority）用于唯一标识这个ContentProvider，外部调用者可以根据这个标识来找到它。
54. C，描述了数据的路径，确定返回哪类数据: 路径（path）可以用来表示我们要操作的数据，路径的构建应根据业务而定
55. 1.ID唯一标注请求的数据: 我们要请求数据的ID
56. 1.常用的存储方式有哪些？（概率50%）
57. 1: Shared Preferences(SP存储)(以键值对的形式存储私有的,简单数据类型)
58. 2: Internal Storage(内存存储)(私有的数据存储在设备的内存里面)
59. 3: External Storage(外部存储)(将公共的数据存储在SD卡里面)
60. 4: SQLite Databases(数据库存储)(将数据存储在私有的数据库里面)
61. (数据库存储包括SQLite和ContentProvider,其实ContentProvider的底层就是在操作数据库)
62. 5: Network Connection(网络存储)(存储数据在web端,在你自己的数据服务器上)
63. 2.安卓中动画有几类，他们的特点和区别？（概率80%）
64. 1: Property Animation(属性动画别加入在3.0)
65. 2: View Animation (补间动画分为:透明,旋转,位移,缩放)
66. 注意:补间动画是假动画,点击区域不变.
67. 3: Drawable Animation(帧动画帧动画类似于放电影，通过播放已经排列放好的图片来实现。)
68. 3.handler机制原理（概率100%）
69. 为什么引进？UI控件不是线程安全的,子线程不能更新UI,所以安卓引入了Handler机制,
70. 为什么系统不对UI控件的访问加锁机制,缺点有两个
71. 1、加上锁机制会让UI访问的逻辑变的复杂,
72. 2、其次锁机制会降低UI访问的效率,因为锁机制会阻塞某些线程的执行,
73. Handler主要用于线程间的通信。
74. 一个Handler允许发送和处理Message和Runable对象，
75. UI主线程会自动分配一个Looper（消息轮询器），每个Looper中封装着MessageQueue（消息队列），遵循先进先出原则。Looper负责不断的从自己的消息队列里取出队头的任务或消息执行。
76. 一般是在子线程执行完耗时操作之后，通过Handler的sendMessage或post方法将Message和Runable对象传递给MessageQueue，而且在这些对象离开MessageQueue时，Handler负责执行他们（用到handleMessage方法，主要执行刷新UI的代码）。
77. 其中Message类就是定义了一个信息，这个信息中包含一个描述符和任意的数据对象，这个信息被用来传递给Handler.Message对象提供额外的两个int域和一个Object域。
78. 155. 解释下单线程中message、handler、Message Queue、Looper之间的关系。（概率95%）
79. 答案：简单来说，Handler获取当前线程中的looper对象，looper用来从存放Message说的MessageQueue中获取Message，再有Handler进行Message的分发和处理。
80. 14.Handler、Looper消息队列模型，各部分的作用
81. 1.MessageQueue：读取会自动删除消息，单链表维护，在插入和删除上有优势。在其next()中会无限循环，不断判断是否有消息，有就返回这条消息并移除。
82. 2.Looper：Looper创建的时候会创建一个MessageQueue，调用loop()方法的时候消息循环开始，loop()也是一个死循环，会不断调用messageQueue的next()，当有消息就处理，否则阻塞在messageQueue的next()中。当Looper的quit()被调用的时候会调用messageQueue的quit(),此时next()会返回null，然后loop()方法也跟着退出。
83. 3.Handler：在主线程构造一个Handler，然后在其他线程调用sendMessage(),此时主线程的MessageQueue中会插入一条message，然后被Looper使用。
84. 4.系统的主线程在ActivityThread的main()为入口开启主线程，其中定义了内部类Activity.H定义了一系列消息类型，包含四大组件的启动停止。
85. 5.MessageQueue和Looper是一对一关系，Handler和Looper是多对一
86. 48.HandlerThread是什么
87. 1.MessageQueue + Looper + Handler
88. 191. Asynctask的原理（概率60%）
89. AsyncTask的本质是一个线程池，所有提交的异步任务都会在这个线程池中的工作线程内执行，当工作线程需要跟UI线程交互时，工作线程会通过向在UI线程创建的Handler传递消息的方式，调用相关的回调函数，从而实现UI界面的更新。
90. 4除了handler和子线程，还有一个处理线程的是什么，主要方法是什么？（概率60%）
91. AsynTask
92. doInBackground() 这个方法运行在后台线程中，主要负责执行那些很耗时的操作，如访问网络。该方法必须重写。
93. onPostExecute(Result) 这个方法运行于UI主线程，在doInBackground(Params…)方法执行后调用，该方法用于接收后台任务执行后返回的结果，刷新UI显示
94. 6线程之间的通信方式（如何避免ANR）（概率70%）
95. （1）. AsyncTask
96. （2.）子线程 + handler
97. 223.线程间通信 （概率40%）
98. 1.同步（本质上就是“共享内存”式的通信）
99. 2.while轮询的方式
100. 3.wait/notify机制
101. 4.管道
102. 5.接口回调
103. 6.共享变量
104. 7.android 特有的handler AsyncTask

### 26·工作中如何利用线程池的？（概率60%）

1. 优势:线程池会缓存一定数量的线程,通过线程池就可以避免因为频繁创建和销毁线程所带来的系统开销(一般会根据cpu支持的线程最大数获取线程数,取的4分之1,一般情况下只有一个APP在前端运行,没什么问题)
2. 使用场景:比如说同时要下载多张图片,这个时候需要开线程,但是如果图片数量太多,每张图片图片都需要一个线程,那么就会造成cpu的销毁,所以这个时候就需要使用线程池来指定最大线程数量,其余需要下载的图片在线程池中等等下载.
3. Java通过Executors提供四种线程池，分别为：
4. newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。
5. newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。
6. newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。
7. newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。
8. 26.Asynctask和线程池，GC相关（怎么判断哪些内存该GC，GC算法）
9. 1.Asynctask：异步任务类，单线程线程池+Handler
10. 2.线程池：
11. 1.ThreadPoolExecutor：通过Executors可以构造单线程池、固定数目线程池、不固定数目线程池。
12. 2.ScheduledThreadPoolExecutor：可以延时调用线程或者延时重复调度线程。
13. 3.GC相关：重要
14. 1.搜索算法：
15. 1.引用计数
16. 2.图搜索，可达性分析
17. 2.回收算法：
18. 1.标记清除复制：用于青年代
19. 2.标记整理：用于老年代
20. 3.堆分区：
21. 1.青年区eden 80%、survivor1 10%、survivor2 10%
22. 2.老年区
23. 4.虚拟机栈分区：
24. 1.局部变量表
25. 2.操作数栈
26. 3.动态链接
27. 4.方法返回地址
28. 5.GC Roots:
29. 1.虚拟机栈(栈桢中的本地变量表)中的引用的对象
30. 2.方法区中的类静态属性引用的对象
31. 3.方法区中的常量引用的对象
32. 4.本地方法栈中JNI的引用的对象
33. 56.Sharedpreferences源码简述
34. 1.储存于硬盘上的xml键值对，数据多了会有性能问题
35. 2.ContextImpl记录着SharedPreferences的重要数据，文件路径和实例的键值对
36. 3.在xml文件全部内加载到内存中之前，读取操作是阻塞的，在xml文件全部内加载到内存中之后，是直接读取内存中的数据
37. 4.apply因为是异步的没有返回值, commit是同步的有返回值能知道修改是否提交成功
38. 5.多并发的提交commit时，需等待正在处理的commit数据更新到磁盘文件后才会继续往下执行，从而降低效率; 而apply只是原子更新到内存，后调用apply函数会直接覆盖前面内存数据，从一定程度上提高很多效率。 3.edit()每次都是创建新的EditorImpl对象.
39. 6.博客推荐：全面剖析SharedPreferences
40. 189. 数据库如何写sql语句（概率40%）
41. 公司框架中中有对数据库的增删改查，有时候自己也写一些复杂的sql语句】
42. 107.Sqlite中的事务如何实现（概率60%）
43. 开启事务： start transaction
44. sql:…update account set money=money-500 where id=1;--检验隔离性
45. sql…update account set money=money+500 where id=2;
46. commit提交-验证原子性、一致性
47. 28.数据库性能优化：索引和事务，需要找本专门的书大概了解一下
48. 58.浏览器输入地址到返回结果发生了什么
49. 1.DNS解析
50. 2.TCP连接
51. 3.发送HTTP请求
52. 4.服务器处理请求并返回HTTP报文
53. 5.浏览器解析渲染页面
54. 6.连接结束
55. 5.tcp和udp区别.（概率80%）
56. TCP---(长链接)(管发管收,数据安全)传输控制协议,提供的是面向连接、可靠的字节流服务,传输数据前经过“三次握手”建立连接，保证数据传输的可靠性，但效率比较低。一般用于对于数据传输安全性较高的场合。
57. UDP---(短连接,只管发,不管收)用户数据报协议，是一个简单的面向数据报的运输层协议，面向无连接。UDP不提供可靠性，数据传输可能发生错序，丢包，但效率较高。一般用于对于实时性（聊天）要求较高的场合。
58. 42.如何设计在 UDP 上层保证 UDP 的可靠性传输；
59. 1.简单来讲，要使用UDP来构建可靠的面向连接的数据传输，就要实现类似于TCP协议的超时重传，有序接受，应答确认，滑动窗口流量控制等机制,等于说要在传输层的上一层（或者直接在应用层）实现TCP协议的可靠数据传输机制。
60. 2.比如使用UDP数据包+序列号，UDP数据包+时间戳等方法，在服务器端进行应答确认机制，这样就会保证不可靠的UDP协议进行可靠的数据传输。
61. 3.基于udp的可靠传输协议有：RUDP、RTP、UDT
62. 222.Http head里放哪些信息（概率50%）
63. 常见的head头信息
64. Accept：浏览器可接受的MIME类型。accept="image/\*"
65. Accept-Charset：浏览器可接受的字符集。utf-8
66. Accept-Encoding：浏览器能够进行解码的数据编码方式，比如gzip。Servlet能够向支持gzip的浏览器返回经gzip编码的HTML页面。许多情形下这可以减少5到10倍的下载时间。
67. Accept-Language：浏览器所希望的语言种类，当服务器能够提供一种以上的语言版本时要用到。zh-cn
68. Connection：表示是否需要持久连接。如果Servlet看到这里的值为“Keep-Alive”HTTP 1.1默认进行持久连接
69. Content-Type：application/json;charset=utf-8
70. Content-Length：表示请求消息正文的长度。
71. Cookie：这是最重要的请求头信息之一
72. Host：初始URL中的主机和端口。
73. 27.网络
74. 1.ARP协议:在IP以太网中，当一个上层协议要发包时，有了该节点的IP地址，ARP就能提供该节点的MAC地址。
75. 2.HTTP HTTPS的区别:
76. 1.HTTPS使用TLS(SSL)进行加密
77. 2.HTTPS缺省工作在TCP协议443端口
78. 3.它的工作流程一般如以下方式:
79. 1.完成TCP三次同步握手
80. 2.客户端验证服务器数字证书，通过，进入步骤3
81. 3.DH算法协商对称加密算法的密钥、hash算法的密钥
82. 4.SSL安全加密隧道协商完成
83. 5.网页以加密的方式传输，用协商的对称加密算法和密钥加密，保证数据机密性；用协商的hash算法进行数据完整性保护，保证数据不被篡改
84. 3.http请求包结构，http返回码的分类，400和500的区别
85. 1.请求：请求行、头部、数据
86. 2.返回：状态行、头部、数据
87. 1.包结构：
88. 2.http返回码分类：1到5分别是，消息、成功、重定向、客户端错误、服务端错误
89. 4.Tcp
90. 1.fin-c = x , 表示现在需要关闭c到s了。ack-c = y,表示上一条s的消息已经接收完毕
91. 2.ack-s = x + 1，表示需要关闭的fin-c消息已经接收到了，同意关闭
92. 3.fin-s = y + 1，表示s已经准备好关闭了，就等c的最后一条命令
93. 4.ack-c = y + 1，表示c已经关闭，让s也关闭
94. 1.三次握手：防止了服务器端的一直等待而浪费资源，例如只是两次握手，如果s确认之后c就掉线了，那么s就会浪费资源
95. 1.syn-c = x，表示这消息是x序号
96. 2.ack-s = x + 1，表示syn-c这个消息接收成功。syn-s = y，表示这消息是y序号。
97. 3.ack-c = y + 1，表示syn-s这条消息接收成功
98. 1.可靠连接，三次握手，四次挥手
99. 2.四次挥手：TCP是全双工模式
100. 3.滑动窗口，停止等待、后退N、选择重传
101. 4.拥塞控制，慢启动、拥塞避免、加速递减、快重传快恢复
102. 195. Http和Https有什么区别（概率80%）
103. http和https使用的是完全不同的连接方式,用的端口也不一样,前者是80,后者是443。http的连接很简单,是无状态的,... HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议 要比http协议安全
104. 66.访问网络都有哪两种方式（概率40%）
105. get与post是两种不同的提交方式
106. get是把参数数据队列加到提交表单的ACTION属性所指的URL中，值和表单内各个字段一一对应，在URL中可以看到。post是通过HTTP post机制，将表单内各个字段与其内容放置在HTML HEADER内一起传送到ACTION属性所指的URL地址。用户看不到这个过程。
107. 对于get方式，服务器端用Request.QueryString获取变量的值，对于post方式，服务器端用Request.Form获取提交的数据。
108. get传送的数据量较小，不能大于2KB。post传送的数据量较大，一般被默认为不受限制。但理论上，IIS4中最大量为80KB，IIS5中为100KB。
109. get安全性非常低，post安全性较高。
110. HTTP 定义了与服务器交互的不同方法，最基本的方法是 GET 和 POST。事实上 GET 适用于多数请求，而保留 POST 仅用于更新站点。
111. 128.Doget和doPost的区别（概率60%）
112. 通过get方式提交的数据有大小的限制，通常在1024字节左右。也就是说如果提交的数据很大，用get方法就可需要小心；而post方式没有数据大小的限制，理论上传送多少数据都可以。
113. 通过get传递数据，实际上是将传递的数据按照”key，value”的方式跟在URL的后面来达到传送的目的的；而post传递数据是通过http请求的附件进行的，在URL中并没有明文显示
114. 95、HttpClient与HttpUrlConnection的区别(okhttp) （概率40%）
115. 后者更为常用。HttpClient是个开源框架，封装了访问http的请求头，参数，内容体，响应等等， HttpURLConnection是java的标准类，什么都没封装，用起来太原始，不方便HttpClient实际上是对Java提供方法的一些封装，在HttpURLConnection中的输入输出流操作，在这个接口中被统一封装成了HttpPost(HttpGet)和HttpResponse，这样，就减少了操作的繁琐性。
116. 159. 断点续传（概率50%）
117. 断点续传下载，其实就是记录下原来下载的位置，然后从那个位置开始下载就是了。
118. 218.批量下载怎么实现（概率50%）
119. 批量下载多任务并行下载、自动断点续传、失败自动重试机制、支持大文件（超过2G）下载、异常处理和异常恢复机制、管理下载文件的生命周期（下载文件的增删改查）等，下载过程中需要多页面多方位同步下载进度和下载状态（比如商店APP，A页面开启点击下载，B页面和C页面也可以看到下载进度和暂停这个下载，而D页面 可以删除这个下载，无论哪个页面有操作，其它页面都会收到响应的回调进行更新，并且还可能还需要常驻一个service来把下载情况同步发送到通知栏的），而且需要断点续传节约流量（比如做批量应用更新APP、单个应用自己更新下载新版本、做影视需要批量缓存视频的APP，做MP3歌曲下载APP），除此之外还需要考虑网络状态变化时自动暂停等。
120. 推荐一个开源框架框架file-downloader
121. 21.如何实现一个网络框架(参考Volley)
122. 1.缓存队列,以url为key缓存内容可以参考Bitmap的处理方式，这里单独开启一个线程。
123. 2.网络请求队列，使用线程池进行请求。
124. 3.提供各种不同类型的返回值的解析如String，Json，图片等等。
125. 30.网络劫持的类型原理：可以百度一下了解一下具体概念
126. 1.DNS劫持、欺骗、污染
127. 2.http劫持：重定向、注入js，http注入、报文扩展
128. 65.访问网络如何加密（概率90%）
129. 1：对称加密（ＤＥＳ，ＡＥＳ）和非对称（ＲＳＡ公钥与私钥）。
130. （支付宝里的商户的公钥和私钥）
131. 2：MD5（算法）
132. 3：Base64
133. 216.如何保证接口数据安全（概率90%）
134. 1.对数据使用加密 常见的加密算法DES AES？对称加密非对称加密？
135. 2.可以使用信息标题+时间+key生成一个签名(sign)
136. 3.token 可以设计token=5位随机数+时间戳 并用AES加密，验证时间和服务器时间不能超过3分，同一个时间戳，随机数只能使用一次。
137. 4.使用https
138. 32.retrofit的了解
139. 1.动态代理创建一个接口的代理类
140. 2.通过反射解析每个接口的注解、入参构造http请求
141. 3.获取到返回的http请求，使用Adapter解析成需要的返回值。
142. 55.Serializable和Parcelable 的区别
143. 42.移动互联数据的交换格式 （概率20%）
144. 移动互联数据交互格式有XML和JSON
145. 1.JSON和XML的数据可读性基本相同
146. 2.JSON和XML同样拥有丰富的解析手段
147. 3.JSON相对于XML来讲，数据的体积小
148. 4.JSON与JavaScript的交互更加方便
149. 5.JSON对数据的描述性比XML较差
150. 6.JSON的速度要远远快于XML
151. json解析方式:手动解析(JSONOBJ和JSONArray)自动解析方式有:gson
152. XML:解析方式:dom,sax,pull:
153. Dom:一次性全部加载进内容里面进行解析,不能中途停止
154. Sax:基于事件驱动性,一行一行解析,不能中途停止,
155. Pull:基于事件驱动下,一行一行解析,可以中途停止
156. 96，json解析方式的两种区别（概率40%）
157. 1，SDK提供JSONArray，JSONObject
158. 2，google提供的 Gson
159. 通过fromJson()实现对象的反序列化（即将json串转换为对象类型）
160. 通过toJson()实现对象的序列化 （即将对象类型转换为json串）
161. 通过google提供的Gson解析json时，定义JavaBean的规则是什么？
162. 1). 实现序列化 Serializable
163. 2). 属性私有化，并提供get，set方法
164. 3). 提供无参构造
165. 4). 属性名必须与json串中属性名保持一致 （因为Gson解析json串底层用到了Java的反射原理）
166. 83，如何序列化（概率40%）
167. 1). 实现序列化 Serializable
168. 2). 属性私有化，并提供get，set方法
169. 3). 提供无参构造
170. 4). 属性名必须与json串中属性名保持一致 （因为Gson解析json串底层用到了Java的反射原理）
171. 56.xml有几种解析方式、区别（概率30%）
172. 基本的解析方式有三种: DOM,SAX,Pull
173. dom解析：解析器读入整个文档，然后构建一个驻留内存的树结构，然后就可以使用 DOM 接口来操作这个树结构。优点是对文档增删改查比较方便，缺点占用内存比较大。
174. sax解析：基于事件驱动型,优点占用内存少，解析速度快，缺点是只适合做文档的读取，不适合做文档的增删改，不能中途停止。
175. pull解析：同样基于事件驱动型,android 官方API提供,可随时终止,调用next() 方法提取它们（主动提取事件）
176. 80、android中常用的布局（概率60%）
177. 第一种：帧布局（框架布局）FrameLayout，在这个布局中，所有的子元素统统放于这块区域的左上角，并且后面的子元素直接覆盖在前面的子元素之上，将前面的子元素部分和全部遮挡。
178. 第二种：线性布局LinearLayout，最常用的一种布局方式，所有子控件的对齐方式，取决于如何定义 orientation的属性：vertical 垂直方向 ，如果按照这种方向所有的子控件将按照垂直的方式分布在布局上，每行只允许有一个子元素，horizontal水平方向 ，这时子控件将会以水平的方向分布在布局中。
179. 第三种：绝对布局AbsoluteLayout，又可以叫做坐标布局，可以直接指定子元素的绝对位置，这种布局简单直接，直观性强，但是由于手机屏幕尺寸差别比较大，使用绝对定位的适应性会比较差。
180. 第四种：相对布局RelativeLayout，允许子元素指定它们相对于其父元素或兄弟元素的位置，这是实际布局中最常用的布局方式之一。它灵活性大很多，当然属性也多，操作难度也大，属性之间产生冲突的的可能性也大，使用相对布局时要多做些测试。
181. 第五种：表格布局TableLayout，表格布局TableLayout以行列的形式管理子元素，每一行是一个TableRow布局对象，当然也可以是普通的View对象，TableRow里每放一个元素就是一列，总列数由列数最多的那一行决定。
182. 第六种：网格布局 GridLayout，在Android 4.0中，新引入的GridLayout网格布局，GridLayout布局使用虚细线将布局划分为行，列和单元格，也支持一个控件在行，列上都有交错排列。而GridLayout使用的其实是跟LinearLayout类似的API，只不过是修改了一下相关的标签而已，所以对于开发者来说，掌握GridLayout还是很容易的事情。
183. 第七种:百分比布局
184. 204. 如何根据权重实现布局（概率80%）
185. //权重和父容器orientation有关
186. horizontal 指水平方向权重 android:layout\_width
187. vertical 指垂直方向权重 android:layout\_height
188. Layout\_weight是线性布局，也就是LinearLayout里面用到的
189. 在layout\_width设置为fill\_parent的时候，layout\_weight代表的是你的控件要优先尽可能的大,但尽可能大是有限度的，即fill\_parent.
190. 在layout\_width设置为wrap\_content的时候，layout\_weight代表的是你的控件要优先尽可能的小,但这个小是有限度的，即wrap\_content.
191. 106.如何实现ViewPager（概率30%）
192. 先在布局文件中调用v4包下的ViewPager，在java代码中初始化ViewPager，并设置adapter（继承PagerAdapter），再设置监听onPageChangeListener
193. 210.Viewpager是如何使用（概率30%）
194. 我们可以做很多事情，从最简单的导航，到页面菜单等等。那如何使用它呢，与LisstView类似，我们也需要一个适配器，他就是PagerAdapter，也可以与fragment结合使用
195. 125.ViewFlipper和ViewPager的区别（概率30%）
196. 1.ViewFilpper类继承于ViewAnimator类。而ViewAnimator类继承于FrameLayout。
197. 查看ViewAnimator类的源码可以看出此类的作用主要是为其中的View切换提供动画效果。该类有如下几个和动画相关的方法。
198. 2.ViewPager用于实现多页面的切换效果，该类存在于Google的兼容包里面，所以在引用时记得在BuilldPath中加入“android-support-v4.jar”

### 40·安卓自定义组件？（概率100%）

Android自定义组件有三种实现思路:

1. 1) 继承某个现有组件，在其基础上添加额外功能,如继承Gallery实现CoverFlow效果
2. 2) 复合型组件定义: 继承某个Layout，实现复合组件自定义，如TextView和EditText组合实现登录注册组件
3. 3) 继承View，实现onDraw()方法，实现自己绘制组件，如翻页效果组件

### 197.View更新页面的五种方式（概率70%）

1. Activity.runOnUiThread(Runnable)
2. View.post(Runnable)
3. View.postDelayed(Runnable,long);
4. View.postInvalidate();
5. View.Invalidate();
6. 2.android view绘制机制和加载过程，请详细说下整个流程
7. 1.ViewRootImpl会调用performTraversals(),其内部会调用performMeasure()、performLayout、performDraw()。
8. 2.performMeasure()会调用最外层的ViewGroup的measure()→onMeasure(),ViewGroup的onMeasure()是抽象方法，但其提供了measureChildren()，这之中会遍历子View然后循环调用measureChild()这之中会用getChildMeasureSpec()+父View的MeasureSpec+子View的LayoutParam一起获取本View的MeasureSpec，然后调用子View的measure()到View的onMeasure()→setMeasureDimension(getDefaultSize(),getDefaultSize()),getDefaultSize()默认返回measureSpec的测量数值，所以继承View进行自定义的wrap\_content需要重写。
9. 3.performLayout()会调用最外层的ViewGroup的layout(l,t,r,b),本View在其中使用setFrame()设置本View的四个顶点位置。在onLayout(抽象方法)中确定子View的位置，如LinearLayout会遍历子View，循环调用setChildFrame()→子View.layout()。
10. 4.performDraw()会调用最外层ViewGroup的draw():其中会先后调用background.draw()(绘制背景)、onDraw()(绘制自己)、dispatchDraw()(绘制子View)、onDrawScrollBars()(绘制装饰)。
11. 5.MeasureSpec由2位SpecMode(UNSPECIFIED、EXACTLY(对应精确值和match\_parent)、AT\_MOST(对应warp\_content))和30位SpecSize组成一个int,DecorView的MeasureSpec由窗口大小和其LayoutParams决定，其他View由父View的MeasureSpec和本View的LayoutParams决定。ViewGroup中有getChildMeasureSpec()来获取子View的MeasureSpec。
12. 6.三种方式获取measure()后的宽高：
13. 1.Activity#onWindowFocusChange()中调用获取
14. 2.view.post(Runnable)将获取的代码投递到消息队列的尾部。
15. 3.ViewTreeObservable.
16. 1.android事件分发机制，请详细说下整个流程
17. 99、android事件分发机制（概率90%）
18. 当你点击了某个控件，首先会去调用该控件所在布局的dispatchTouchEvent方法，然后在布局的dispatchTouchEvent方法中找到被点击的相应控件，再去调用该控件的dispatchTouchEvent方法
19. 138.Android事件分发的理解（概率100%）
20. 事件分发：public boolean dispatchTouchEvent(MotionEvent ev)
21. Touch 事件发生时 Activity 的 dispatchTouchEvent(MotionEvent ev) 方法会以隧道方式（从根元素依次往下传递直到最内层子元素或在中间某一元素中由于某一条件停止传递）将事件传递给最外层 View 的 dispatchTouchEvent(MotionEvent ev) 方法，并由该 View 的 dispatchTouchEvent(MotionEvent ev) 方法对事件进行分发。dispatchTouchEvent 的事件分发逻辑如下：
22. 如果 return true，事件会分发给当前 View 并由 dispatchTouchEvent 方法进行消费，同时事件会停止向下传递；
23. 如果 return false，事件分发分为两种情况：
24. 如果当前 View 获取的事件直接来自 Activity，则会将事件返回给 Activity 的 onTouchEvent 进行消费；
25. 如果当前 View 获取的事件来自外层父控件，则会将事件返回给父 View 的 onTouchEvent 进行消费。
26. 如果返回系统默认的 super.dispatchTouchEvent(ev)，事件会自动的分发给当前 View 的 onInterceptTouchEvent 方法。
27. ▐ 事件拦截：public boolean onInterceptTouchEvent(MotionEvent ev)
28. 在外层 View 的 dispatchTouchEvent(MotionEvent ev) 方法返回系统默认的 super.dispatchTouchEvent(ev) 情况下，事件会自动的分发给当前 View 的 onInterceptTouchEvent 方法。onInterceptTouchEvent 的事件拦截逻辑如下：
29. 如果 onInterceptTouchEvent 返回 true，则表示将事件进行拦截，并将拦截到的事件交由当前 View 的 onTouchEvent 进行处理；
30. 如果 onInterceptTouchEvent 返回 false，则表示将事件放行，当前 View 上的事件会被传递到子 View 上，再由子 View 的 dispatchTouchEvent 来开始这个事件的分发；
31. 如果 onInterceptTouchEvent 返回 super.onInterceptTouchEvent(ev)，事件默认会被拦截，并将拦截到的事件交由当前 View 的 onTouchEvent 进行处理。
32. ▐ 事件响应：public boolean onTouchEvent(MotionEvent ev)
33. 在 dispatchTouchEvent 返回 super.dispatchTouchEvent(ev) 并且 onInterceptTouchEvent 返回 true 或返回 super.onInterceptTouchEvent(ev) 的情况下 onTouchEvent 会被调用。onTouchEvent 的事件响应逻辑如下：
34. 如果事件传递到当前 View 的 onTouchEvent 方法，而该方法返回了 false，那么这个事件会从当前 View 向上传递，并且都是由上层 View 的 onTouchEvent 来接收，如果传递到上面的 onTouchEvent 也返回 false，这个事件就会“消失”，而且接收不到下一次事件。
35. 如果返回了 true 则会接收并消费该事件。
36. 如果返回 super.onTouchEvent(ev) 默认处理事件的逻辑和返回 false 时相同。
37. 到这里，与 Touch 事件相关的三个方法就分析完毕了。
38. 186.如何进行手势判断（概率30%）
39. 用onTouchEvent(ev)判断手势
40. 108.Android中常用的菜单以及区别（概率30%）
41. 1、左右推出的Menu
42. 使用方法：
43. 一.首先导入SlidingMenu这个库文件
44. 二.在你的项目中引入SlidingMenu这个库
45. eclipse中的流程：右击你的项目 -- Properties -- Android -- Add 选择SlidingMenu库
46. 三.在你的项目中使用这个SlidingMenu
47. 配置设定方法中 setMode() 是常用属性有：
48. 1）SlidingMenu.LEFT 左测菜单
49. 2）SlidingMenu.RIGHT 右侧菜单
50. 3）SlidingMenu.LEFT\_RIGHT 左右2测菜单
51. 2、Path菜单
52. 3、GridView菜单
53. 4、popuwindow菜单
54. 5、系统菜单
55. 121.SurfaceView实现的原理（概率80%）
56. SurfaceView，它拥有独立的绘图表面，即它不与其宿主窗口共享同一个绘图表面。由于拥有独立的绘图表面，因此SurfaceView的UI就可以在一个独立的线程中进行行绘制。又由于不占用主线程资源，SurfaceView一方面可以实现复杂而高效的UI，另一方面又不会导致用户输入得不到及时响应。
57. 57.webview的辅助类（概率20%）
58. WebSettings、WebViewClient、WebChromeClient
59. WebSettings
60. 设置WebView的一些属性、状态等,例如允许使用javascript，允许使用缓存，允许使用内置的缩放组件
61. setJavaScriptEnabled(true); 设置支持js
62. WebViewClient
63. 主要帮助WebView处理各种通知、请求事件（例如，点击链接时候如何显示界面，页面开始加载，加载完毕之后有何动作等）
64. shouldOverrideUrlLoading() onPageStarted() onPageFinished()
65. WebChromeClient
66. 辅助WebView处理Javascript的对话框、网站图标、网站Title、加载进度等
67. 核心方法有onJsAlert() onJsConfirm() onJsPrompt() onProgressChanged() onReceivedIcon()
68. onReceivedTitle()
69. 58.webview加载网页的两种方式（概率20%）
70. loadUrl(url) 通过url地址加载互联网或本地网页
71. 如：loadUrl("http://www.baidu.com"); // 互联网地址
72. loadUrl("file://android\_asset/html/index.html"); /／本地网页
73. loadData(data,type,encoding) 直接加载网页内容，容易乱码，一般用 loadDataWithBaseURL代替
74. 84、如何在android中调用js（概率70%）
75. loadUrl("javascript:test()");
76. 224.android中Webview与javascript的交互（互相调用）（概率70%）
77. 1.mWebView.getSettings().setJavaScriptEnabled(true);
78. 2.android对象方法定义
79. 3.设置本地调用对象及其接口 mWebView.addJavascriptInterface
80. (new JavaScriptObject(mContext), "obj");
81. 4.android调用js方法 mWebView.loadUrl("javascript:funFromjs()");
82. 5.js调用android方法obj.fun1FromAndroid（JavaScriptObject中定义好的方法）
83. 52.android中asset和raw的区别；（概率40%）
84. \*res/raw和assets的相同点：
85. 1.两者目录下的文件在打包后会原封不动的保存在apk包中，不会被编译成二进制。
86. \*res/raw和assets的不同点：
87. 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.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。
88. 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
89. \*读取文件资源：
90. 1.读取res/raw下的文件资源，通过以下方式获取输入流来进行写操作
91. InputStream is = getResources().openRawResource(R.id.filename);
92. //通过 资源 id 直接打开 raw 下的 流文件
93. 2.读取assets下的文件资源，通过以下方式获取输入流来进行写操作
94. AssetManager am = null;
95. am = getAssets();
96. InputStream is = am.open("filename");
97. // 通过 AssectManager 管理器 来打开 assect 下的流文件
98. 150. 9-PatchImage是什么（概率30%）
99. .9.PNG是安卓开发里面的一种特殊的图片，这种格式的图片通过ADT自带的编辑工具生成，使用九宫格切分的方法，使图片支持在android 环境下的自适应展示。

### 25·简述音乐播放的方式与区别？（概率20%）

1. 1）：MediaPlayer：主要用于播放音频，可以播放视频，但是一般不用其进行视频播放。
2. 2）：SoundPool：主要用于播放一些短促的声音片段，主要优势是cpu资源占有量低，反应延迟小。

### 208.播放视频的实现方式（概率60%）

1. 1）使用系统自带的播放器来播放，指定Action为ACTION\_VIEW,Data为Uri，Type为其MIME类型。
2. //调用系统自带的播放器
3. Intent intent = new Intent(Intent.ACTION\_VIEW);
4. intent.setDataAndType(uri, "video/mp4");
5. startActivity(intent);
6. 2)使用VideoView组件来播放, 可以结合MediaController来实现播控, 只是不能随意更改视频的大小及位置。
7. 3)使用MediaPlayer和SurfaceView来实现，这种方式很灵活,可以自定义视频播放的大小和位置。

### 14·MVC的使用？（概率90%）

1. mvc是model,view,controller的缩写，mvc包含三个部分：(逻辑,控制,展示分开)
2. l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。
3. l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。
4. l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
5. android鼓励弱耦合和组件的重用，在android中mvc的具体体现如下：
6. 1)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。
7. 2)视图层（view）：一般采用xml文件进行界面的描述
8. 3)控制层（controller）：android的控制层的acitvity
9. 39.mvc、mvp、mvvm：
10. 1.mvc:数据、View、Activity，View将操作反馈给Activity，Activitiy去获取数据，数据通过观察者模式刷新给View。循环依赖
11. 1.Activity重，很难单元测试
12. 2.View和Model耦合严重
13. 2.mvp:数据、View、Presenter，View将操作给Presenter，Presenter去获取数据，数据获取好了返回给Presenter，Presenter去刷新View。PV，PM双向依赖
14. 1.接口爆炸
15. 2.Presenter很重
16. 3.mvvm:数据、View、ViewModel，View将操作给ViewModel，ViewModel去获取数据，数据和界面绑定了，数据更新界面更新。
17. 1.viewModel的业务逻辑可以单独拿来测试
18. 2.一个view 对应一个 viewModel 业务逻辑可以分离，不会出现全能类
19. 3.数据和界面绑定了，不用写垃圾代码，但是复用起来不舒服
20. 20·NDK是什么？（概率50%）
21. NDK 是 Native Development Kit 的简称。它是一个工具集，集成了 Android 的交叉编译环境，并提供了一套比较方便的 Makefile ，可以帮助开发者快速开发 C 或是 C++ 的动态库，并自动的将 so 和 java 程序打包成 apk ，在 Android 上运行。Android 上，应用程序的开发，大部分基于 Java 语言来实现。要使用 c 或是 c++ 的程序或库，就需要使用 NDK 来实现。
22. 29·SIM卡的EF文件有何作用？（概率10%）
23. SIM卡的文件系统有自己的规范，主要是为了和手机通讯，SIM本身可以有自己的操作系统，EF就是作 存储并和手机通讯用的。
24. 30.最长的短信多长，多少个字节？（概率10%）
25. 70个汉字，140个字节
26. 179. 一条最长短信约占多少byte（概率10%）
27. 160
28. 60.拦截短信（概率40%）
29. 短信其实是一个有序广播
30. 1：创建一个继承BroadcastReceiver类的广播接收器。
31. 2：在清单文件中注册广播并在<intent-filter>中设置优先级 priority（只要比你想拦截程序的短信接收器优先级高就可以了）
32. 3：在onReceiver（）中调用abortBroadcast()就可以了
33. 117.Gps定位和网络定位的区别（概率50%）
34. GPS定位需设置权限android.permission.ACCESS\_LOCATION\_FINE
35. 网络定位需设置权限android.permission.ACCESS\_LOCATION\_COARSE
36. 即GPS定位比网络定位在精确度方面更精准，但在速度方面稍差一些，而且GPS定位适合在室外环境下使用，网络定位在室内，室外都可以。
37. 124.嵌入式操作系统（概率20%）
38. 嵌入式实时操作系统是指当外界事件或数据产生时，能够接受并以足够快的速度予以处理，其处理的结果又能在规定的时间之内来控制生产过程或对处理系统作出快速响应，并控制所有实时任务协调一致运行的嵌入式操作系统。主要用于工业控制、军事设备、航空航天等领域对系统的响应时间有苛刻的要求，这就需要使用实时系统。又可分为软实时和硬实时两种，而android是基于linux内核的，因此属于软实时。
39. 67.一键退出的实现（概率40%）
40. 定义一个类继承Application，定义一个集合存放所有的activity，定义一个添
41. 加的方法，再写一个退出的方法，使用for循环全部调用finish方法，然后在每个
42. Activity的onCreate方法中调用自定义类里的添加方法，然后在需要使用一键退出
43. 的地方调用类中的退出方法即可。
44. 211.如何自定义一个Adapter（概率60%）
45. 继承自BaseAdapter实现里面的方法，listView在开始绘制的时候，系统首先调用getCount（）函数，根据他的返回值得到listView的长度，然后根据这个长度，调用getView（）逐一绘制每一行。如果你的getCount（）返回值是0的话，列表将不显示同样return 1，就只显示一行。系统显示列表时，首先实例化一个适配器（这里将实例化自定义的适配器）。当手动完成适配时，必 须手动映射数据，这需要重写getView（）方法。系统在绘制列表的每一行的时候将调用此方法。getView()有三个参数，position表示将显示的是第几行，covertView是从布局文件中inflate来的 布局。我们用LayoutInflater的方法将定义好的main.xml文件提取成View实例用来显示。然后 将xml文件中的各个组件实例化（简单的findViewById()方法）。这样便可以将数据对应到各个组件上了。但是按钮为了响应点击事件，需要为它添加点击监听器，这样就能捕获点击事件。至此一个自定 义的listView就完成了，现在让我们回过头从新审视这个过程。系统要绘制ListView了，他首先获得 要绘制的这个列表的长度，然后开始绘制第一行，怎么绘制呢？调用getView()函数。在这个函数里面 首先获得一个View（实际上是一个ViewGroup），然后再实例并设置各个组件，显示之。好了，绘制完这一行了。那 再绘制下一行，直到绘完为止。在实际的运行过程中会发现listView的每一行没有焦点了，这是因为Button抢夺了listView的焦点，只要布局文件中将Button设置为没有焦点就OK了。
46. 23.listView优化（概率60%）
47. 1)、convertView复用，对convetView进行判空，当convertView不为空时重复使用，为空则初始化，从而减少了很多不必要的View的创建
48. 2)定义一个ViewHolder，封装Listview Item条目中所有的组件，将convetView的tag设置为ViewHolder,不为空时通过ViewHolder的属性获取对应组件即可
49. 3)、当ListView加载数据量较大时可以采用分页加载和图片异步加载
50. 69.listview的分页加载（概率20%）
51. ListView分页加载通常有两种实现方式：一种是ListView底部设置一个按钮，用户点击即加载；另一种是当用户滑动到底部时自动加载。
52. 在此说一下第一种方式点击底部按钮即加载的实现思路：
53. 通过addFooterView()方法为listview底部加入一个“加载更多”按钮和加载进度的视图，默认按钮显示，加载进度为gone。当用户点击“加载更多”按钮时，按钮设置为gone，加载进度显示，并开启子线程从网络上加载下一页数据，加载完毕，发消息给handler，再handler的handleMessage（）方法中通过adapter的notifyDataSetChanged方法通知listview刷新，显示刚加入的数据，并恢复按钮显示，加载进度隐藏。
54. 41.listview分页加载的步骤（概率20%）
55. 通常实现分页加载有两种方式，一种是在ListView底部设置一个按钮，用户点
56. 击即加载。另一种是当用户滑动到底部时自动加载。
57. 在ListView底部设置一个按钮，用户点击即加载实现思路:
58. // 加上底部View，注意要放在setAdapter方法前
59. ListView.addFooterView(moreView);
60. bt.setOnClickListener(new OnClickListener() {
61. @Override
62. public void onClick(View v) {
63. pg.setVisibility(View.VISIBLE);// 将进度条可见
64. bt.setVisibility(View.GONE);// 按钮不可见
65. handler.postDelayed(new Runnable() {
66. @Override
67. public void run() {
68. loadMoreDate();// 加载更多数据
69. bt.setVisibility(View.VISIBLE);
70. pg.setVisibility(View.GONE);
71. mSimpleAdapter.notifyDataSetChanged();// 通知listView刷新数据
72. }
73. }, 2000);
74. }
75. });
76. 当用户滑动到底部时自动加载实现思路: 实现OnScrollListener 接口重写
77. onScrollStateChanged 和onScroll方法，使用onscroll方法实现”滑动“后处理检查是否还有新的记录，如果有,添加记录到adapter, adapter调用 notifyDataSetChanged 更新数据;如果没有记录了，则不再加载数据。使用onScrollStateChanged可以检测是否滚到最后一行且停止滚动然后执行加载.
78. 50.listview失去焦点怎么处理（概率20%）
79. 在listview子布局里面写，可以解决焦点失去的问题
80. android:descendantFocusability="blocksDescendants"
81. 180. 上拉加载下拉刷新如何实现（概率50%）
82. 答：实现OnScrollListener 接口重写onScrollStateChanged 和onScroll方法，使用onscroll方法实现”滑动“后处理检查是否还有新的记录，如果有，调用 addFooterView，添加记录到adapter, adapter调用 notifyDataSetChanged 更新数据;如果没有记录了，把自定义的mFooterView去掉。使用onScrollStateChanged可以检测是否滚到最后一行且停止滚动然后执行加载
83. 111.gradView上拉加载和下拉刷新怎么实现（概率40%）
84. 实现OnScrollListener 接口重写onScrollStateChanged 和onScroll方法，使用onscroll方法实现”滑动“后处理检查是否还有新的记录，如果有，调用 addFooterView，添加记录到adapter, adapter调用 notifyDataSetChanged 更新数据;如果没有记录了，把自定义的mFooterView去掉。使用onScrollStateChanged可以检测是否滚到最后一行且停止滚动然后执行加载
85. 193. 瀑布流如何实现（概率30%）
86. 就是异步下载图片的时候把图片添加到scrollview中，
87. 你需要几列就在scrollview中放置几个linearlayout，在scrollview滑动的的时候动态的测量linearlayout随机生成linearlayout的高度
88. 46.图片轮播的实现思路（概率30%）
89. 图片轮播是由viewpage和handler两者结合所实现的，
90. 170. 图片三级缓存的机制（概率70%）
91. 实现图片缓存也不难，需要有相应的cache策略。这里我采用 内存-文件-网络 三层cache机制，其中内存缓存包括强引用缓存和软引用缓存（SoftReference），其实网络不算cache，这里姑且也把它划到缓存的层次结构中。当根据url向网络拉取图片的时候，先从内存中找，如果内存中没有，再从缓存文件中查找，如果缓存文件中也没有，再从网络上通过http请求拉取图片。在键值对（key-value）中，这个图片缓存的key是图片url的hash值，value就是bitmap。所以，按照这个逻辑，只要一个url被下载过，其图片就被缓存起来了。
92. 202. Android的几种缓存方式 （概率40%）
93. 内存缓存，文件缓存，网络缓存
94. 11·图片异步加载怎么做？（概率80%）
95. 三级缓存:
96. 1:首先从强引用中获取(Lrucache),如果强引用中没有,再去软引用中获取(SoftReference),如果软引用再去SD卡获取(SD卡被Dis Lrucache替代掉),如果这三级缓存中都没有,就开启网络去请求图片,请求回来的图片,在加入到强引用中(Lrucache),如果强引用已经满了,这时候,强引用通过Lrucache算法,删除掉一些最近不常用的图片,这时候咱们把这些图片放到软引用里面,(当内存不足到时候软引用就会被被垃圾回收机制回收),如果垃圾回收机制没有,软引用也满了,将不常用的图片移除掉,同时加入到SD卡,(现在被Dis Lrucache替代)
97. (key 一般使用的是网络请求图片的url,来保证唯一性,一般情况下我会做一下MD5加密,这样做的目的是为了保证不会出现非法字符)
98. 现在SoftReference已经被替代掉,现在我们只使用两级缓存,在加网络, Lrucache, Dis Lrucache
99. LRC算法:最近最少使用的(内部如何实现的)
100. 强引用: Lrucache
101. 软引用: SoftReference
102. 弱引用:weak(随时被垃圾回收机制回收)
103. 虚引用:phantomreference
104. 可定义图片异步加载工具类，核心方式实现思路如下：
105. 1.先从内存缓存（Map<String,SoftReference<Bitmap>>中获取图片显示
106. 2.获取不到的话从本地SD卡里获取并显示
107. 3.都获取不到的话通过子线程从网络加载图片并保存到内存及SD卡中并通过handler显示
108. 20.Bitmap的处理
109. 1.当使用ImageView的时候，可能图片的像素大于ImageView，此时就可以通过BitmapFactory.Option来对图片进行压缩，inSampleSize表示缩小2^(inSampleSize-1)倍。
110. 2.BitMap的缓存：
111. 1.使用LruCache进行内存缓存。
112. 2.使用DiskLruCache进行硬盘缓存。
113. 3.实现一个ImageLoader的流程：同步异步加载、图片压缩、内存硬盘缓存、网络拉取
114. 1.同步加载只创建一个线程然后按照顺序进行图片加载
115. 2.异步加载使用线程池，让存在的加载任务都处于不同线程
116. 3.为了不开启过多的异步任务，只在列表静止的时候开启图片加载
117. 9.安卓内存的优化？（概率90%）
118. http://blog.csdn.net/awangyunke/article/details/20380719
119. handler内存泄露（thread， AsyncTask)
120. http://www.linuxidc.com/Linux/2013-12/94065.htm
121. context内存泄露
122. http://blog.sina.com.cn/s/blog\_5da93c8f0102w86x.html
123. Bitmap内存泄露
124. 1）静态变量引起内存泄露(内存泄露多了就变成溢出)
125. 在代码优化的过程中，我们需要对代码中的静态变量特别留意。静态变量是类相关的变量，它的生命周期是从这个类被声明，到这个类彻底被垃圾回收器回收才会被销毁。所以，一般情况下，静态变量从所在的类被使用开始就要一直占用着内存空间，直到程序退出。如果不注意，静态变量引用了占用大量内存的资源，造成垃圾回收器无法对内存进行回收，就可能造成内存的浪费
126. 2）使用Application(单例模式)的Context
127. 在Android中，Application Context的生命周期和应用的生命周期一样长，而不是取决于某个Activity的生命周期。如果想保持一个长期生命的对象，并且这个对象需要一个 Context，就可以使用Application对象。可以通过调用Context.getApplicationContext()方法或者 Activity.getApplication()方法来获得Application对象。
128. 3）及时关闭资源
129. Cursor是Android查询数据后得到的一个管理数据集合的类。正常情况下，如 果我们没有关闭它，系统会在回收它时进行关闭，但是这样的效率特别低。如果查询得到的数据量较小时还好，如果Cursor的数据量非常大，特别是如果里面 有Blob信息时，就可能出现内存问题。所以一定要及时关闭Cursor。
130. 4）使用Bitmap及时调用recycle() // 把 重复循环的方法设置 为 null，释放内存
131. 前面的章节讲过，在不使用Bitmap对象时，需要调用recycle()释放内存，然后将它设置为null。虽然调用recycle()并不能保证立即释放占用的内存，但是可以加速Bitmap的内存的释放。
132. 在代码优化的过程中，如果发现某个Activity用到了Bitmap对象，却没有显式的调用recycle()释放内存，则需要分析代码逻辑，增加相关代码，在不再使用Bitmap以后调用recycle()释放内存。
133. 5）对Adapter进行优化
134. 下面以构造ListView的BaseAdapter为例说明如何对Adapter进行优化。
135. @软引用和弱引用。
136. 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。
137. 如果一个对象只具有弱引用，那么在垃圾回收器线程扫描的过程中，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。弱引用也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。
138. 弱引用与软引用的根本区别在于：只具有弱引用的对象拥有更短暂的生命周期，可能随时被回收。而只具有软引用的对象只有当内存不够的时候才被回收，在内存足够的时候，通常不被回收。
139. 6）UI优化
140. 在Android应用开发过程中，屏幕上控件的布局代码和程序的逻辑代码通常是分开 的。界面的布局代码是放在一个独立的xml文件中的，这个文件里面是树型组织的，控制着页面的布局。通常，在这个页面中会用到很多控件，控件会用到很多的 资源。Android系统本身有很多的资源，包括各种各样的字符串、图片、动画、样式和布局等等，这些都可以在应用程序中直接使用。这样做的好处很多，既 可以减少内存的使用，又可以减少部分工作量，也可以缩减程序安装包的大小。
141. 13·内存泄露如何解决？（概率90%）
142. http://wenku.baidu.com/view/03dc4b46b307e87100f69604.html
143. 1）、 数据库的cursor没有关闭
144. 2）、 构造adapter没有使用缓存contentview
145. Listview:性能优化:
146. 1: Listview是需要和适配器配合使用的,常用的适配器有arrayAdapter,SimplerAdapter, SimpleCursorAdapter,BaseAdapter,一般情况下我使用的是继承BaseAdapter,实现它的四个方法,其中最重要的两个方法,getCount,和getView,在不做任何处理的情况下getview每次都重新生成一个view,这样的结果就是很消耗内存,谷歌官网提供了优化listview内存的方法,通过ConvertView的复用和使用ViewHolder来减少findbyid的次数,从而达到控件的复用和减少cpu的消耗,但是因为控件的复用,在加载图片的时候,复用的控件里面有残留的图片,从而造成图片的错位,所以我们必须在初始化的时候设置默认图片来清除缓存的图片,同时在加载的时候图片的时候,如果不进行listview的性能优化,就会造成OOM,所以一般情况下我们会加载二次采样过后的url,同时我们会进行三级缓存的处理,(同上)
147. 二次采样:
148. http://m.blog.csdn.net/article/details?id=51245133
149. 既然是二次采样，那当然要分为两步了，下面我们来说说每次采样的主要工作：
150. 1.第一次采样
151. 第一次采样我主要是想要获得图片的压缩比例，假如说我有一张图片是200\*200，那么我想把这张图片的缩略图显示在一个50\*50的ImageView上，那我的压缩比例应该为4，那么这个4应该怎么样来获得呢？这就是我们第一步的操作了，我先加载图片的边界到内存中，这个加载操作并不会耗费多少内存，加载到内存之后，我就可以获得这张图片的宽高参数，然后根据图片的宽高，再结合控件的宽高计算出缩放比例。
152. 2.第二次采样
153. 在第一次采样的基础上，我来进行二次采样。二次采样的时候，我把第一次bab采样后算出来的结果作为一个参数传递给第BitmapFactory，这样在加载图片的时候系统就不会将整张图片加载进来了，而是只会加载该图片的一张缩略图进来，这样不仅提高了加载速率，而且也极大的节省了内存，而且对于用户来说，他也不会有视觉上的差异
154. 3.代码实现
155. 第一次采样:首先获取new BitmapFactory.Options()获取option对象, inJustDecodeBounds设置为true只会加载图片的边框进来，并不会加载图片具体的像素点,通过BitmapFactory.decodeFile(filePath, options);来进行第一次加载图片,通过outWidth原图的宽度,通过outHeight获取原图的高,接下来通过图片的跨度和控件的高度来获取一个采样率sampleSize,接下来设置inJustDecodeBounds为false,同时设置缩放比,设置inPreferredConfig的图片的图片格式有(分别为ALPHA\_8，RGB\_565，ARGB\_4444，ARGB\_8888),最后进行图片的加载.
156. 衍生的listview优化问题：减少创建View的对象，充分使用contentview，可以使用静态类来处理优化getView的过程 3、Bitmap对象不使用时采用recycle()释放内存 4、Activity中的对象生命周期大于Activity
157. ）Bitmap对象不使用recycle（）释放内存
158. ）Activity中的对象生命周期大于Activity；
159. 内存泄露大多数都是由于相同的错误导致的，即：对Context拥有较长时间的引用。
160. 避免Context相关的内存泄露，记住以下事情：
161. 不要保留对Context-Activity长时间的引用（对Activity的引用的时候，必须确保拥有和Activity一样的生命周期）
162. 尝试使用Context-Application来替代Context-Activity
163. 如果你不想控制内部类的生命周期，应避免在Activity中使用非静态的内部类，而应该使用静态的内部类，并在其中创建一个对Activity的弱引用。这种情况的解决办法是使用一个静态的内部类，其中拥有对外部类的WeakReference，如同ViewRoot和它的Winner类那样
164. GC（垃圾回收）不能解决内存泄露问题
165. 194. 图片的内存不足（概率40%）
166. 可能是你加载的当前的图片占用内存大于当前剩余内存，解决方法就是及时释放内存，不显示的图片要释放掉。也可能是你加载的图片是空的，导致内存不足。android系统给图片分配的内存只有8M,当加载大量图片时往往会出现OOM。
167. Android加载大量图片内存溢出解决方案：
168. 1）尽量不要使用setImageBitmap或setImageResource或BitmapFactory.decodeResource来设置一张大图，因为这些函数在完成decode后，最终都是通过java层的createBitmap来完成的，需要消耗更多内存，可以通过BitmapFactory.decodeStream方法，创建出一个bitmap，再将其设为ImageView的 source
169. 2）使用BitmapFactory.Options对图片进行压缩
170. InputStream is = this.getResources().openRawResource(R.drawable.pic1);
171. BitmapFactory.Options options=new BitmapFactory.Options();
172. options.inJustDecodeBounds = false;
173. options.inSampleSize = 10; //width，hight设为原来的十分一
174. Bitmap btp =BitmapFactory.decodeStream(is,null,options);
175. 3）运用Java软引用，进行图片缓存，将需要经常加载的图片放进缓存里，避免反复加载
176. 及时销毁不再使用的Bitmap对象
177. if(!bmp.isRecycle() ){
178. bmp.recycle() //回收图片所占的内存
179. system.gc() //提醒系统及时回收
180. }
181. 51.内存泄漏
182. 1.其他线程持有一个Listener，Listener操作activity。那么在线程么有完毕的时候，activity关闭了，原本是要被回收的但是，不能被回收。
183. 2.例如Handler导致的内存泄漏，Handler就相当于Listener。
184. 3.在activity关闭的时候注意停止线程，或者将Listener的注册取消
185. 3.使用弱引用，这样即使Listener持有了activity，在GC的时候还是会被回收
186. 4.工具:LeakCanary
187. 52.过度绘制、卡顿优化:
188. 1.过度绘制：
189. 1.移除Window默认的Background：getWidow.setBackgroundDrawable(null);
190. 2.移除XML布局文件中非必需的Background
191. 3.减少布局嵌套(扁平化的一个体现，减少View数的深度，也就减少了View树的遍历时间，渲染的时候，前后期的工作，总是按View树结点来)
192. 4.在引入布局文件里面，最外层可以用merge替代LinearLayout,RelativeLayout，这样把子UI元素直接衔接在include位置
193. 5.工具：HierarchyViewer 查看视图层级
194. 2.卡顿优化：16ms数据更新
195. 214.性能优化（概率90%）
196. UI方面：
197. 1.减少多层嵌套（绘制流程）
198. 2.ListView GridView 复用组件
199. 3.ViewPager缓存page最好为3 并且每个pager显示的时候再去加载数据，减少预加 载，避免浪费。
200. 4.图片显示慢 提高下载速度（InputStream再包一层缓冲流会更快(如BufferedInputStream)）三级缓存 二次采样 用开源库
201. 缓存方面：
202. 1.合理利用lrucache，disklrucache通过算法来平衡内存与硬盘缓存
203. 2.网络＋数据库 做增量加载
204. 网络方面：
205. 1.使用合适的数据格式比如json
206. 2.多个请求合并成一个，尽量不做重定向
207. 3.合理的缓存，控制最大请求并发量，及时取消已失效的请求，过滤重复请求，timeout时间设置，请求优先级设置等。
208. 代码规范：
209. 1.for loop中不要声明临时变量，不到万不得已不要在里面写try catch。
210. 2.避免频繁GC，内存泄漏，OOM(有机会专门说)
211. 3.合理使用数据类型，比如StringBuilder代替String，(笔试题最常见的是str+="str"中有几个对象) ，少用枚举enum，少用父类声明(List,Map)
212. 4.合理使用线程池，减少线程创建开销。
213. 5.使用单例
214. 6.多用常量，少用显式的"action\_key"，并维护一个常量类，别重复声明这些常量。
215. 省电方面：
216. 1.合理使用service
217. 2.后台定时任务用Alarm代替Timer（例如心跳包）注意：小米手机五分钟以内Alarm无效
218. 3.减少网络使用 对数据进行压缩（上传文件）wifi与非wifi区别对待（下载大文件，看视频等）
219. 4.应用前台后台，屏幕熄灭点亮 区别对待
220. 215.Android6.0动态权限申请（概率90%）
221. 6.0权限分为普通权限和单独申请权限，普通权限只需要在AndroidManifest.xml里申请即可，单独申请的权限分为9组 每组包含几个权限，只要其中一个申请成功就默认整租权限都可以使用
222. 申请步骤
223. 1.将targetSdkVersion设置为23
224. 2.在AndroidManifest.xml中申请你需要的权限，包括普通权限和需要申请的特殊权限
225. 3.申请权限
226. 1.检查权限ContextCompat.checkSelfPermission()
227. 2.向用户解释为何申请权限ActivityCompat.shouldShowRequestPermissionRationale()判断用户是否勾选了不再提醒
228. 3.申请权限ActivityCompat.requestPermissions() 可以申请一个权限也可以同时申请多个权限
229. 4.权限申请结果回调onRequestPermissionsResult
230. 常见问题
231. 1.不能使用Application的Contxt只能使用Activity， fragment的上下文
232. 2.权限申请时使用的请求码必须小于16
233. 3.小米4的6.0的shouldShowRequestPermissionRationale 就一直返回false，而且在申请权限时，如果用户选择了拒绝，则不会再弹出对话框了 所以说这个地方有坑，解决方法在回调里面处理，如果用户拒绝了这个权限，则打开本应用信息界面，由用户自己手动开启这个权限
234. 48.全局异常处理（概率50%）
235. Android使用UncaughtExceptionHandler捕获全局异常（未捕获的异常处理程序）
236. Android系统的“程序异常退出”，给应用的用户体验造成不良影响。为了捕获应用运行时异常并给出友好提示，便可继承UncaughtExceptionHandler类来处理。通过Thread.setDefaultUncaughtExceptionHandler()方法将异常处理类设置到线程上即可。
237. Demo下载地址：http://code.google.com/p/android-custom-view/downloads/list
238. 54.ANR的形成，各个组件上出现ARN的时间限制是多少
239. 212. Android 中ANR异常的处理方法 ：( 意思是: 应用程序没有响应 ) （概率40%）
240. 有两种情况会报ANR: 自己添加的
241. 1.主线程在5秒内没有响应输入事件（事件处理/UI线程）
242. 2.广播在10秒内没有完成返回( BroadcastReceiver )
243. 3.service 20秒没有响应时间
244. 下面是导致android中出现ANR异常的的情况:
245. 1.在主线程内进行网络操作
246. 2.在主线程内进行一些缓慢的磁盘操作(例如执行没有优化的SQL查询)
247. 处理的逻辑是:
248. 1.New一个新的线程，进行数据请求
249. 2.获取数据后，调用handler.sendmessage方法
250. 3.在handler的handle()方法中更新UI
251. 38·屏幕适配？（概率100%）
252. ه最好可以通过权重(layout\_weight)的方式来分配每个组件的大小，也可以通过具体的像素(dip)来确定大小。
253. 尽量使用Relativelayout 。
254. 已知应用支持平台设备的分辨率,可以提供多个layout\_320\*480 ...
255. drawable-hdpi,drawable-mdpi,drawable-ldpi分别代表分辨率为480\*800,360\*480,240\*360, 放置图片大小相差1.5倍
256. 最后还需要在AndroidManifest.xml里添加下面一段，没有这一段自适应就不能实现：
257. <supports-screens
258. android:largeScreens="true"
259. android:normalScreens="true"
260. android:anyDensity = "true"/>
261. 在</application>标签和</manifest> 标签之间添加上面那段代码。即可。
262. 备注：三者的解析度不一样，就像你把电脑的分辨率调低，图片会变大一样，反之分辨率高，图片缩小
263. 还可以通过.9.png实现图片的自适应
264. 181.手机适配如何做，自适应（概率80%）
265. 答:
266. 在布局中设置控件尺寸是尽量使用fill-parent,warp-content,dp等值，文字使用sp
267. 尽量使用Relativelayout 。
268. 对不同的屏幕提供不同大小的图片，
269. 已知应用支持平台设备的分辨率,可以提供多个layout\_320\*480 ...
270. drawable-hdpi,drawable-mdpi,drawable-ldpi分别代表分辨率为480\*800,360\*480,240\*360, 放置图片大小相差1.5倍
271. 最后还需要在AndroidManifest.xml里添加下面一段，没有这一段自适应就不能实现：
272. <supports-screens
273. android:largeScreens="true"
274. android:normalScreens="true"
275. android:anyDensity = "true"/>
276. 在</application>标签和</manifest> 标签之间添加上面那段代码。即可。
277. 219.机型适配遇到的问题（概率60%）
278. 1.无法创建快捷方式
279. 2.重复创建快捷方式
280. 3.照相机拍照指定存储路径，有的时候图片无法存储，大部分手机onActivityResult方法中的data返回为空但个别手机不为空
281. 4.照相机摄像头个数返回不准确，有的手机摄像头方向不正确
282. 5.双卡问题，短信，通话记录，联系人的数据库差异
283. 6.UI适配问题
284. 29.APK打包流程和其内容
285. 1.流程
286. 1.aapt生成R文件
287. 2.aidl生成java文件
288. 3.将全部java文件编译成class文件
289. 4.将全部class文件和第三方包合并成dex文件
290. 5.将资源、so文件、dex文件整合成apk
291. 6.apk签名
292. 7.apk字节对齐
293. 2.内容：so、dex、asset、资源文件
294. 53.apk瘦身:
295. 1.classes.dex：通过代码混淆，删掉不必要的jar包和代码实现该文件的优化
296. 2.资源文件：通过Lint工具扫描代码中没有使用到的静态资源
297. 3.图片资源：使用tinypng和webP，下面详细介绍图片资源优化的方案,矢量图
298. 4.SO文件将不用的去掉，目前主流app一般只放一个arm的so包
299. 
300. 50.class和dex
301. 1.dvm执行的是dex格式文件，jvm执行的是class文件，android程序编译完之后生产class文件。然后dex工具会把class文件处理成dex文件，然后把资源文件和.dex文件等打包成apk文件。
302. 2.dvm是基于寄存器的虚拟机，而jvm执行是基于虚拟栈的虚拟机。寄存器存取速度比栈快的多，dvm可以根据硬件实现最大的优化，比较适合移动设备。
303. 3.class文件存在很多的冗余信息，dex工具会去除冗余信息，并把所有的class文件整合到dex文件中。减少了I/O操作，提高了类的查找速度
304. 184. 项目中的混淆打包（概率70%）
305. a在项目的project.properties文件中加入如下代码：
306. proguard.config=proguard.cfg即可
307. 好处：1.隐藏代码，避免被反编译2.减少代码提交，提高效率4.android中四大组件不能被混淆，所用在清单文件中声明的都是被混淆5.native不能被混淆
308. 39·版本更新?数据库的更新？跨版本？（概率30%）
309. // 应用升级( 版本号升级 ) 或软件升级，数据库就会跟着升级
310. 应用升级==版本号升级 == 同时数据库也会 跟着升级
311. 软件升级 == 同时数据库也会跟着升级
312. 版本：
313. 首先在启动页面,首先请求网络,获取服务器端版本号,同时获取本地APP的版本号,如果服务器的版本号高于本地的版本号,就开启线程去现在APP,同时安装APP,如果服务器的版本号和本地版本号相同,就不下载app,直接进去app.
314. 在清单文件更改版本号，上传更新的应用到平台，利用平台的更新提醒用户是否更新新版本。
315. 数据库更新：
316. 如果数据库版本号大于当前版本号,这个时候就会执行onupgress方法,为指定的表增加字段或
317. 先更改版本号，在onupgress方法内判断版本号是否一致，不一致则更新数据库。
318. 206.Android中项目升级注意的问题（概率50%）
319. 在清单文件里面的android:versionCode和android:versionName有版本文件, 客户端同时存储该应用当前版本号 (SharedPreferences/Sqlite), 每次打开应用,去检测服务器版本号与本地版本号是否一致,如果不一 致,则自定义对话框提示是否下载更新
320. 22.ClassLoader的基础知识
321. 1.双亲委托：一个ClassLoader类负责加载这个类所涉及的所有类，在加载的时候会判断该类是否已经被加载过，然后会递归去他父ClassLoader中找。
322. 2.可以动态加载Jar通过URLClassLoader
323. 3.ClassLoader 隔离问题 JVM识别一个类是由：ClassLoader id+PackageName+ClassName。
324. 4.加载不同Jar包中的公共类：
325. 1.让父ClassLoader加载公共的Jar，子ClassLoader加载包含公共Jar的Jar，此时子ClassLoader在加载公共Jar的时候会先去父ClassLoader中找。(只适用Java)
326. 2.重写加载包含公共Jar的Jar的ClassLoader，在loadClass中找到已经加载过公共Jar的ClassLoader，也就是把父ClassLoader替换掉。(只适用Java)
327. 3.在生成包含公共Jar的Jar时候把公共Jar去掉。
328. 23.插件化框架描述：dynamicLoadApk为例子
329. 1.可以通过DexClassLoader来对apk中的dex包进行加载访问
330. 2.如何加载资源是个很大的问题，因为宿主程序中并没有apk中的资源，所以调用R资源会报错，所以这里使用了Activity中的实现ContextImpl的getAssets()和getResources()再加上反射来实现。
331. 3.由于系统启动Activity有很多初始化动作要做，而我们手动反射很难完成，所以可以采用接口机制，将Activity的大部分生命周期提取成接口，然后通过代理Activity去调用插件Activity的生命周期。同时如果像增加一个新生命周期方法的时候，只需要在接口中和代理中声明一下就行。
332. 4.缺点：
333. 1.慎用this，因为在apk中使用this并不代表宿主中的activity，当然如果this只是表示自己的接口还是可以的。除此之外可以使用that代替this。
334. 2.不支持Service和静态注册的Broadcast
335. 3.不支持LaunchMode和Apk中Activity的隐式调用。
336. 24.热修复：Andfix为例子
337. 1.大致原理：apkpatch将两个apk做一次对比，然后找出不同的部分。可以看到生成的apatch了文件，后缀改成zip再解压开，里面有一个dex文件。通过jadx查看一下源码，里面就是被修复的代码所在的类文件,这些更改过的类都加上了一个\_CF的后缀，并且变动的方法都被加上了一个叫@MethodReplace的annotation，通过clazz和method指定了需要替换的方法。然后客户端sdk得到补丁文件后就会根据annotation来寻找需要替换的方法。最后由JNI层完成方法的替换。
338. 2.无法添加新类和新的字段、补丁文件很容易被反编译、加固平台可能会使热补丁功能失效
339. 172. 如何控制耗电量 （概率60%）
340. 在以下情况下耗电比较多：
341. 1、 大数据量的传输。
342. 2、 不停的在网络间切换。
343. 3、 解析大量的文本数据。
344. 1、 在需要网络连接的程序中，首先检查网络连接是否正常，如果没有网络连接，那么就不需要执行相应的程序。
345. 2、 使用效率高的数据格式和解析方法。(fastJson,LoganSquare-master)
346. 3、使用GZIP压缩方式下载数据，能减少网络流量
347. 4、 其它一些优化方法：
348. 回收java对象，特别是较大的java对像
349. 对定位要求不是太高的话尽量不要使用GPS定位，可能使用wifi和移动网络cell定位即可。GPS定位消耗的电量远远高于移动网络定位。
350. 尽量不要使用浮点运算。
351. 获取屏幕尺寸等信息可以使用缓存技术，不需要进行多次请求。
352. 217.接口测试工具用过哪些（概率60%）
353. 1.有在线网页测试http://coolaf.com/
354. 2.postman（Chrome的插件）
355. 3.Androidstudio插件 jsonviewer
356. 112.用过哪些框架（概率70%）
357. XUtils AFinal,okhttp,volley 第三方开源框架
358. 63.开源框架源码
359. 1.Fresco
360. 2.oKhttp：
361. 1.同步和异步：
362. 2.连接池：
363. 3.选择路线与建立连接
364. 3.okio
365. 1.简介；
366. 136. 分享平台有哪些（概率30%）
367. ShardSdk、友盟等。
368. 131.常用的推送，并简单介绍（概率80%）
369. 并行的都包括：即时信息和空间协议(IMPP)、空间和即时信息协议(PRIM)；
370. 推送的实现原理：
371. XMPP是一种基于XML的协议，它继承了在XML环境中灵活的发展性。因此，基于XMPP的应用具有超强的可扩展性。XMPP的前身是Jabber，一个开源形式组织产生的网络即时通信协议。所有从一个client到另一个client的jabber消息和数据都要xmpp server，client连接到server，server利用本地目录系统的证书对其认证，client制定目标地址，让server告知目标状态，server查找，连接并进行相互认证，client间进行交互。
372. XMPP系统特点：
373. 1)客户机/服务器通信模式；(2)分布式网络；(3)简单的客户端；(4)XML的数据格式。
374. 通俗解释：
375. 其实XMPP 是一种很类似于http协议的一种数据传输协议，它的过程就如同“解包装--〉包
376. 装”的过程，用户只需要明白它接受的类型，并理解它返回的类型，就可以很好的利用xmpp来进行数据通讯。
377. 43.xmpp的底层是怎样实现的(即时聊天)？（概率20%）
378. socket，双广播，连接广播和通知广播
379. 140. Ddms和Traceview的区别（概率30%）
380. 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。
381. 21百度地图和谷歌地图的优缺点？（概率60%）
382. 优势：
383. 1一打开百度地图页面，显示的就是目前所在城市地图，谷歌地图不然。
384. 2百度地图的界面，百度地图很多都是在显眼位置用文字做提示，一目了然，谷歌界面有点炫，很多用图标来表示，但是用起来相对复杂，不熟悉的用户需要确定这个图标是否是所需要选择的。
385. 而百度地图搜索框下就直接有“公交”“驾车”的选项，界面简洁直接很多，谷歌地图需要点击获取路线，再点击小图标来选择“公交”这些选项，而且图标不够引人注目，使用起来相对复杂些，试想一个很少用网络地图的人分别使用二者地图，那个更方便呢？应该是百度，大多数用户通过地图是查询位置-这个地方在哪，周围有什么地物，查询路线-从一个方到另一个地方如何到达，用百度地图，这些问题很容易解决，而用谷歌地图得摆弄一会儿。
386. 3百度地图的查询路线更好,首先还是界面更简洁直接，然后是数据的问题，先不管为他们提供数据的公司如何，刚刚特意同时查了下，同样地起点和终点， 百度地图给出了5条路线，谷歌地图给出的是4条路线，而且百度地图路线的选择时可以选择“较快捷”“少换乘”“少步行”三种方式，更方便。打车费用也直接 显示出来，而且个人体会10元内的打车费用基本正确。
387. 4百度地图有个视野内搜索功能，可以很方便地查找到目标位置，谷歌地图没有。这个很重要。
388. 5一个重要界面布局问题，百度地图是地图视野在左，而谷歌地图是地图视野在右，我们现在时喜欢从左到右看东西啊，这点百度地图更符合用户习惯。（当然也可能是我常用百度地图，所以这么觉得）
389. PS:刚在知乎上有朋友说到——因“左图右列表”强调的是“地图”，反之强调的是“搜索”；我们发现用户在经过这么些年的教育后，“不在乎”地图在哪边，相对其它体验，这一点影响用户选择服务的权重并不高。
390. 6百度地图已经和都市圈合作推出了部分城市如北京、上海、深圳、广州的三维地图，谷歌地图没有。
391. 7百度地图地图视野上方有个截图功能，可以直接拉框截图，谷歌地图没有。
392. 8以北京实时路况信息为例，直接输入北京查询，百度地图显示的是主城区的实时路况，而谷歌显示的区域很大，包括了密4云、廊坊市了，但是实时路况信息还是显示的主城区，显示路况的彩色条就全挤在一起了，设计得不如百度好。
393. 8使用百度的占大多数，经常可以听到“百度一下啊”的话语。随之带来百度地图的使用率也相对较高。
394. 9百度地图的地图百宝箱、房产、天气等信息。
395. 劣势：
396. 1如果需要从网络地图上获取数据的话，谷歌地图完胜，而且还有谷歌地球在一旁辅助，如虎添翼，从网上各种各样的谷歌地图相关工具软件就可以看出。
397. 2路线查询时输入地点名称却有多个符合条件时，会给出一个下拉菜单给予选择，同一地点谷歌地图提供的数据精确到街道，而百度地图精确到区。
398. 3谷歌地图可以将侧边栏隐藏起来，看到更大的地图视野，而百度地图是点击全屏来显示更大的视野，两种切换方式比起来谷歌的更舒服。
399. 4谷歌地图有个分享链接功能，而且还提供“粘贴 HTML 以便嵌入到网站”这一功能，百度没有。
400. 5界面布局百度地图更简洁，但是色彩搭配上和画面感上谷歌地图更好。
401. 6谷歌的品牌形象占优。
402. 199. 高德地图了解（概率30%）
403. 地图渲染技术：性能提升10倍，所占空间降低80%，比传统地图软件节省流量超过90%
404. 专业在线导航功能：覆盖全国364个城市、全国道路里程352万公里
405. 在线导航功能：最新高德在线导航引擎，全程语音指引提示，完善偏航判定和偏航重导功能。
406. AR虚拟实景：AR功能结合手机摄像头和用户位置、方向等信息，将信息点以更直观的方式展现给用户，为发现和目标地点指引
407. 丰富的出行查询功能：地名信息查询、分类信息查询、公交换乘、驾车路线规划、公交线路查询、位置收藏夹等丰富的基础地理信息查询工具。
408. 锁屏语音提示，即使你的iPhone在锁屏状态也能听到高德导航的语音提示，不用担心一直开着iPhone屏幕费电了。
409. 夜间导航HUD抬头提示，打开高德导航并开启HUD，把iPhone放到汽车挡风玻璃下，高德导航会把路线提示倒映到汽车挡风玻璃上，看起来特别方便，不用担心低头看手机影响驾驶了。
410. 动态导航
411. 交通路况实时播报
412. 智能计算到达目的地所需的时间
413. 避堵路线方案规划
414. 摄像头提醒
415. 离线下载
416. 2D、3D离线地图
417. 分地区下载地图包
418. 全国地图包、全国概要图
419. 地图搜索
420. 热门地点、线路搜索
421. 公交、自驾出行线路规划
422. 公交、火车、天气查询服务
423. 全新引擎
424. 最新3D版本，360度旋转视角
425. 矢量数据传送
426. 观看更流畅、更清晰
427. 兴趣点
428. 餐饮、住宿、优惠、
429. 演出、团购全覆盖
430. 海量兴趣点随意搜
431. 叫车服务
432. 2013年7月,高德地图更新，新增打车功能，综合快的打车、嘀嘀打车两家资源，覆盖80%市场份额。与阿里合作后，高德地图凭借高市场占有率，逐渐从单一地图向LBS工具箱转变。新版高德地图不仅增加了可预约的出租车数量，提高了打车效率，覆盖城市也扩展到26个
433. 88、解释SVN（概率40%）
434. Svn是一种集中式文件版本管理系统。
435. 集中式管理的工作流程如下图：集中式代码管理的核心是服务器，所有开发者在开始新一天的工作之前必须从服务器获取代码，然后开发，最后解决冲突，提交。所有的版本信息都放在服务器上。如果脱离了服务器，开发者基本上是不可以工作。下面举例说明：开始新一天的工作：1，从服务器下载项目组最新代码。2：进入自己的分支，进行工作，每隔一个小时向服务器自己的分支提交一次代码（很多人都有这个习惯。因为有时候自己对代码改来改去，最后又想还原到前一个小时的版本，或者看看前一个小时自己修改了那些代码，就需要这样做了）。3：下班时间快到了，把自己的分支合并到服务器主分支上，一天的工作完成，并反映给服务器。这就是经典的svn工作流程，从流程上看，有不少缺点，但也有优点。缺点：1、 服务器压力太大，数据库容量暴增。2、 如果不能连接到服务器上，基本上不可以工作，看上面第二步，如果服务器不能连接上，就不能提交，还原，对比等等。3、不适合开源开发（开发人数非常非常多，但是Google app engine就是用svn的）。但是一般集中式管理的有非常明确的权限管理机制（例如分支访问限制），可以实现分层管理，从而很好的解决开发人数众多的问题。优点：1、 管理方便，逻辑明确，符合一般人思维习惯。2、 易于管理，集中式服务器更能保证安全性。3、 代码一致性非常高。4、 适合开发人数不多的项目开发。5、大部分软件配置管理的大学教材都是使用svn和vss。
436. 133．如何判断那种网络（概率30%）
437. 1.网络是否连接（包括Wifi和移动网络）
438. // 是否有可用网络
439. private boolean isNetworkConnected() {
440. ConnectivityManager cm =
441. (ConnectivityManager) mContext.getSystemService(Context.CONNECTIVITY\_SERVICE);
442. NetworkInfo network = cm.getActiveNetworkInfo();
443. if (network != null) {
444. return network.isAvailable();
445. }
446. return false;
447. }
448. 2.wifi是否可用
449. // Wifi是否可用
450. private boolean isWifiEnable() {
451. WifiManager wifiManager = (WifiManager) mContext
452. .getSystemService(Context.WIFI\_SERVICE);
453. return wifiManager.isWifiEnabled();
454. }
455. GPS是否可用
456. [javascript] view plaincopyprint?
457. // Gps是否可用
458. private boolean isGpsEnable() {
459. LocationManager locationManager =
460. ((LocationManager) mContext.getSystemService(Context.LOCATION\_SERVICE));
461. return locationManager.isProviderEnabled(LocationManager.GPS\_PROVIDER);
462. }
463. 220.直播项目 难点在哪. （概率70%）
464. 业务逻辑，维护房间，主播上线通知，异常情况的考虑
465. 直播的运营 盈利模式
466. 190. 电商领域的核心（概率80%）
467. 订单，支付，购物，和信息安全
468. 171. 你看好的行业领域（概率30%）
469. 穿戴
470. 185. 什么是020，b2b，b2c（概率30%）
471. http://blog.sina.com.cn/s/blog\_64e090b001016843.html