Android基础面试题 （☆☆☆）

1、什么是ANR 如何避免它?

答：在Android上，如果你的应用程序有一段时间响应不够灵敏，系统会向用户显示一个对话框，这个对话框称作应 用程序无响应（ANR：Application NotResponding）对话框。 用户可以选择让程序继续运行，但是，他们在使用你的 应用程序时，并不希望每次都要处理这个对话框。因此 ，在程序里对响应性能的设计很重要这样，这样系统就不会显 示ANR给用户。

不同的组件发生ANR的时间不一样，Activity是5秒，BroadCastReceiver是10秒，Service是20秒（均为前台）。

如果开发机器上出现问题，我们可以通过查看/data/anr/traces.txt即可，最新的ANR信息在最开始部分。

- 主线程被IO操作（从4.0之后网络IO不允许在主线程中）阻塞。
- 主线程中存在耗时的计算
- 主线程中错误的操作，比如Thread.wait或者Thread.sleep等 Android系统会监控程序的响应状况，一旦出现下面两种情况，则弹出ANR对话框
- 应用在5秒内未响应用户的输入事件（如按键或者触摸）
- BroadcastReceiver未在10秒内完成相关的处理
- Service在特定的时间内无法处理完成 20秒

修正：

1、使用AsyncTask处理耗时IO操作。

2、使用Thread或者HandlerThread时，调用Process.setThreadPriority(Process.THREAD_PRIORITY_BACKGROUND)设置优先级，否则仍然会降低程序响应，因为默认Thread的优先级和主线程相同。

3、使用Handler处理工作线程结果，而不是使用Thread.wait()或者Thread.sleep()来阻塞主线程。

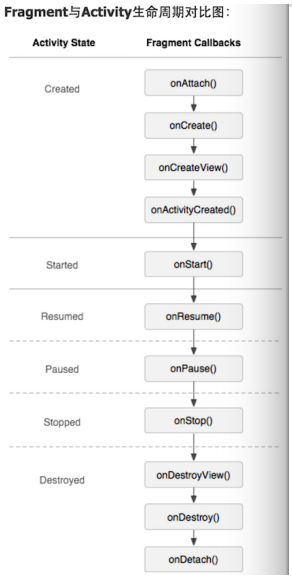
4、Activity的onCreate和onResume回调中尽量避免耗时的代码。 BroadcastReceiver中onReceive代码也要尽量减少耗时，建议使用IntentService处理。

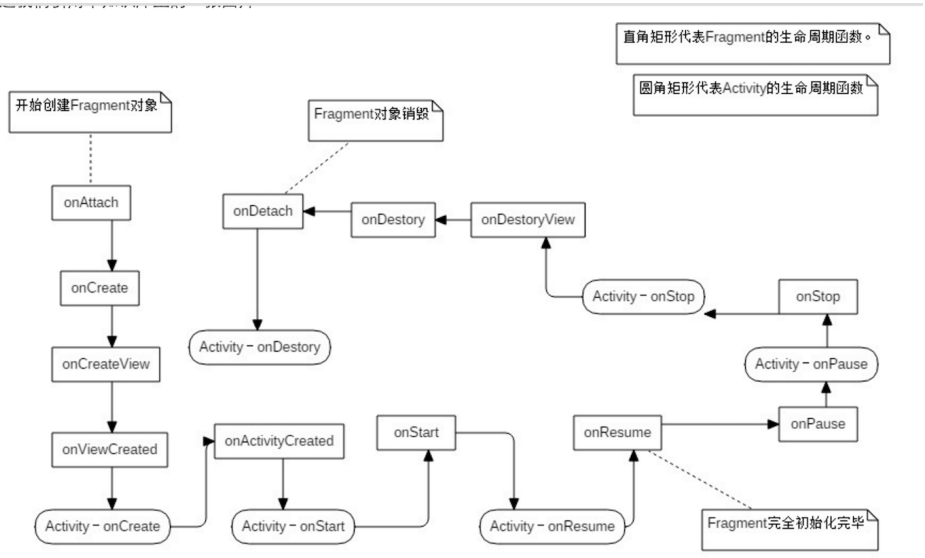
解决方案：

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

深入回答

2、Activity和Fragment生命周期有哪些?





3、横竖屏切换时候Activity的生命周期

不设置Activity的android:configChanges时，切屏会重新回调各个生命周期，切横屏时会执行一次，切竖屏时会执行两次。 设置Activity的android:configChanges=”orientation”时，切屏还是会调用各个生命周期，切换横竖屏只会执行一次 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

4、AsyncTask的缺陷和问题，说说他的原理。

AsyncTask是什么?

AsyncTask是一种轻量级的异步任务类，它可以在线程池中执行后台任务，然后把执行的进度和最终结果传递给主线程并在主线程中更新UI。

AsyncTask是一个抽象的泛型类，它提供了Params、Progress和Result这三个泛型参数，其中Params表示参数的类型，Progress表示后台任务的执行进度和类型，而Result则表示后台任务的返回结果的类型，如果AsyncTask不需要传递具体的参数，那么这三个泛型参数可以用Void来代替。

关于线程池：

AsyncTask对应的线程池ThreadPoolExecutor都是进程范围内共享的，且都是static的，所以是Asynctask控制着进程范围内所有的子类实例。由于这个限制的存在，当使用默认线程池时，如果线程数超过线程池的最大容量，线程池就会爆掉(3.0后默认串行执行，不会出现个问题)。针对这种情况，可以尝试自定义线程池，配合Asynctask使用。

关于默认线程池：

AsyncTask里面线程池是一个核心线程数为CPU + 1，最大线程数为CPU * 2 + 1，工作队列长度为128的线程池，线程等待队列的最大等待数为28，但是可以自定义线程池。线程池是由AsyncTask来处理的，线程池允许tasks并行运行，需要注意的是并发情况下数据的一致性问题，新数据可能会被老数据覆盖掉。所以希望tasks能够串行运行的话，使用SERIAL_EXECUTOR。

AsyncTask在不同的SDK版本中的区别：

调用AsyncTask的execute方法不能立即执行程序的原因及改善方案通过查阅官方文档发现，AsyncTask首次引入时，异步任务是在一个独立的线程中顺序的执行，也就是说一次只执行一个任务，不能并行的执行，从1.6开始，AsyncTask引入了线程池，支持同时执行5个异步任务，也就是说只能有5个线程运行，超过的线程只能等待，等待前的线程直到某个执行完了才被调度和运行。换句话说，如果进程中的AsyncTask实例个数超过5个，那么假如前5都运行很长时间的话，那么第6个只能等待机会了。这是AsyncTask的一个限制，而且对于2.3以前的版本无法解决。如果你的应用需要大量的后台线程去执行任务，那么只能放弃使用AsyncTask，自己创建线程池来管理Thread。不得不说，虽然AsyncTask较Thread使用起来方便，但是它最多只能同时运行5个线程，这也大大局限了它的作用，你必须要小心设计你的应用，错开使用AsyncTask时间，尽力做到分时，或者保证数量不会大于5个，否就会遇到上面提到的问题。可能是Google意识到了AsynTask的局限性了，从Android 3.0开始对AsyncTask的API做出了一些调整：每次只启动一个线程执行一个任务，完了之后再执行第二个任务，也就是相当于只有一个后台线程在执行所提交的任务。

一些问题：

1.生命周期

很多开发者会认为一个在Activity中创建的AsyncTask会随着Activity的销毁而销毁。然而事实并非如此。AsynTask会一直执行，直到doInBackground()方法执行完毕，然后，如果cancel(boolean)被调用,那么onCancelled(Result result)方法会被执行；否则，执行onPostExecute(Result result)方法。如果我们的Activity销毁之前，没有取消AsyncTask，这有可能让我们的应用崩溃(crash)。因为它想要处理的view已经不存在了。所以，我们是必须确保在销毁活动之前取消任务。总之，我们使用AsyncTask需要确保AsyncTask正确的取消。

2.内存泄漏

如果AsyncTask被声明为Activity的非静态内部类，那么AsyncTask会保留一个对Activity的引用。如果Activity已经被销毁，AsyncTask的后台线程还在执行，它将继续在内存里保留这个引用，导致Activity无法被回收，引起内存泄漏。

3.结果丢失

屏幕旋转或Activity在后台被系统杀掉等情况会导致Activity的重新创建，之前运行的AsyncTask会持有一个之前Activity的引用，这个引用已经无效，这时调用onPostExecute()再去更新界面将不再生效。

4.并行还是串行

在Android1.6之前的版本，AsyncTask是串行的，在1.6之后的版本，采用线程池处理并行任务，但是从Android 3.0开始，为了避免AsyncTask所带来的并发错误，又采用一个线程来串行执行任务。可以使用executeOnExecutor()方法来并行地执行任务。

AsyncTask原理

- AsyncTask中有两个线程池（SerialExecutor和THREAD_POOL_EXECUTOR）和一个Handler（InternalHandler），其中线程池SerialExecutor用于任务的排队，而线程池THREAD_POOL_EXECUTOR用于真正地执行任务，InternalHandler用于将执行环境从线程池切换到主线程。
- sHandler是一个静态的Handler对象，为了能够将执行环境切换到主线程，这就要求sHandler这个对象必须在主线程创建。由于静态成员会在加载类的时候进行初始化，因此这就变相要求AsyncTask的类必须在主线程中加载，否则同一个进程中的AsyncTask都将无法正常工作。

5、onSaveInstanceState() 与 onRestoreIntanceState()

Activity的 onSaveInstanceState() 和 onRestoreInstanceState()并不是生命周期方法，它们不同于 onCreate()、onPause()等生命周期方法，它们并不一定会被触发。当应用遇到意外情况（如：内存不足、用户直接按Home键）由系统销毁一个Activity时，onSaveInstanceState() 会被调用。但是当用户主动去销毁一个Activity时，例如在应用中按返回键，onSaveInstanceState()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()只适合用于保存一些临时性的状态，而onPause()适合用于数据的持久化保存。 在activity被杀掉之前调用保存每个实例的状态,以保证该状态可以在onCreate(Bundle)或者onRestoreInstanceState(Bundle) (传入的Bundle参数是由onSaveInstanceState封装好的)中恢复。这个方法在一个activity被杀死前调用，当该activity在将来某个时刻回来时可以恢复其先前状态。 例如，如果activity B启用后位于activity A的前端，在某个时刻activity A因为系统回收资源的问题要被杀掉，A通过onSaveInstanceState将有机会保存其用户界面状态，使得将来用户返回到activity A时能通过onCreate(Bundle)或者onRestoreInstanceState(Bundle)恢复界面的状态

深入理解

6、android中进程的优先级?

1. **前台进程：**

即与用户正在交互的Activity或者Activity用到的Service等，如果系统内存不足时前台进程是最晚被杀死的

2. **可见进程：**

可以是处于暂停状态(onPause)的Activity或者绑定在其上的Service，即被用户可见，但由于失了焦点而不能与用户交互

3. **服务进程：**

其中运行着使用startService方法启动的Service，虽然不被用户可见，但是却是用户关心的，例如用户正在非音乐界面听的音乐或者正在非下载页面下载的文件等；当系统要空间运行，前两者进程才会被终止

4. **后台进程：**

其中运行着执行onStop方法而停止的程序，但是却不是用户当前关心的，例如后台挂着的QQ，这时的进程系统一旦没了有内存就首先被杀死

5. **空进程：**

不包含任何应用程序的进程，这样的进程系统是一般不会让他存在的

7、Bunder传递对象为什么需要序列化？Serialzable和Parcelable的区别?

因为bundle传递数据时只支持基本数据类型，所以在传递对象时需要序列化转换成可存储或可传输的本质状态（字节流）。序列化后的对象可以在网络、IPC（比如启动另一个进程的Activity、Service和Reciver）之间进行传输，也可以存储到本地。

Serializable（Java自带）：

Serializable 是序列化的意思，表示将一个对象转换成存储或可传输的状态。序列化后的对象可以在网络上进传输，也可以存储到本地。

Parcelable（android专用）：

除了Serializable之外，使用Parcelable也可以实现相同的效果，不过不同于将对象进行序列化，Parcelable方式的实现原理是将一个完整的对象进行分解，而分解后的每一部分都是Intent所支持的数据类型，这也就实现传递对象的功能了。

区别总结如下图所示：

	Serializable 接口	Parcelable 接口
平台	Java 的序列化接口	Android 的序列化接口
序列化原理	将一个对象转换成可存储或则可存储的状态	将一个对象进行分解，且分解后的每一部分都是传递可支持的数据类型
优缺点	简单但效率较低，开销大序列化（ObjectOutputStream 类）和反序列化（ObjectInputStream 类）过程都需要大量的 I/O 操作	高效但使用较麻烦
使用场景	适合将对象序列化到存储设备或则将对象序列化后通过网络设备传输	主要用在内存的序列化

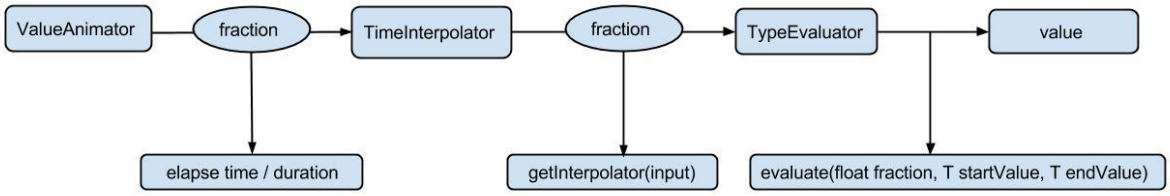
8、动画

- tween 补间动画。通过指定View的初末状态和变化方式，对View的内容完成一系列的图形变换来实现动画效果。 Alpha, Scale ,Translate, Rotate。
- frame 帧动画。AnimationDrawable控制animation-list.xml布局
- PropertyAnimation 属性动画3.0引入，属性动画核心思想是对值的变化。

Property Animation 动画有两个步聚：

1.计算属性值

2.为目标对象的属性设置属性值，即应用和刷新动画



计算属性分为3个过程：

过程一：

计算已完成动画分数 elapsed fraction。为了执行一个动画，你需要创建一个 ValueAnimator，并且指定目标对象属性的开始、结束和持续时间。在调用 start 后的整个动画过程中，ValueAnimator 会根据已经完成的动画时间计算得到一个0 到 1 之间的分数，代表该动画的已完成动画百分比。0表示 0%，1 表示 100%。

过程二：

计算插值（动画变化率）interpolated fraction 。当 ValueAnimator计算完已完成的动画分数后，它会调用当前设置的TimeInterpolator，去计算得到一个interpolated（插值）分数，在计算过程中，已完成动画百分比会被加入到新的插值计算中。

过程三：

计算属性值当插值分数计算完成后，ValueAnimator会根据插值分数调用合适的TypeEvaluator去计算运动中的属性值。 以上分析引入了两个概念：已完成动画分数（elapsed fraction）、插值分数(interpolated fraction)。

原理及特点：

1.属性动画：

插值器：作用是根据时间流逝的百分比来计算属性变化的百分比

估值器：在1的基础上由这个东西来计算出属性到底变化了多少数值的类

其实就是利用插值器和估值器，来计出各个时刻View的属性，然后通过改变View的属性来实现View的动画效果。

2.View动画:

只是影像变化，view的实际位置还在原来地方。

3.帧动画：

是在xml中定义好一系列图片之后，使用AnimatonDrawable来播放的动画。

它们的区别：

属性动画才是真正的实现了 view 的移动，补间动画对view 的移动更像是在不同地方绘制了一个影子，实际对象还是处于原来的地方。 当动画的 repeatCount 设置为无限循环时，如果在Activity退出时没有及时将动画停止，属性动画会导致Activity无法释放而导致内存泄漏，而补间动画却没问题。 xml 文件实现的补间动画，复用率极高。在 Activity切换，窗口弹出时等情景中有着很好的效果。 使用帧动画时需要注意，不要使用过多特别大的图，容导致内存不足。

为什么属性动画移动后仍可点击?

播放补间动画的时候，我们所看到的变化，都只是临时的。而属性动画呢，它所改变的东西，却会更新到这个View所对应的矩阵中，所以当ViewGroup分派事件的时候，会正确的将当前触摸坐标，转换成矩阵变化后的坐标，这就是为什么播放补间动画不会改变触摸区域的原因了。

9、Context相关

- 1、Activity和Service以及Application的Context是不一样的,Activity继承自ContextThemeWraper.其他的继承自ContextWrapper。
- 2、每一个Activity和Service以及Application的Context是一个新的ContextImpl对象。
- 3、getApplication()用来获取Application实例的，但是这个方法只有在Activity和Service中才能调用的到。那也许在绝大多数情况下我们都是在Activity或者Servic中使用Application的，但是如果在一些其它的场景，比如BroadcastReceiver中也想获得Application的实例，这时就可以借助getApplicationContext()方法，getApplicationContext()比getApplication()方法的作用域会更广一些，任何一个Context的实例，只要调用getApplicationContext()方法都可以拿到我们的Application对象。
- 4、创建对话框时不可以用Application的context，只能用Activity的context。
- 5、Context的数量等于Activity的个数 + Service的个数 +1，这个1为Application。

10、Android各版本新特性

Android5.0新特性

- **MaterialDesign设计风格**
- **支持64位ART虚拟机**（5.0推出的ART虚拟机，在5.0之前都是Dalvik。他们的区别是： Dalvik,每次运行,字节码都需要通过即时编译器转换成机器码(JIT)。 ART,第一次安装应用的时候,字节码就会预先编译成机器码(AOT)）

- 通知详情可以用户自己设计

Android6.0新特性

- **动态权限管理**
- 支持快速充电的切换
- 支持文件夹拖拽应用
- 相机新增专业模式

Android7.0新特性

- **多窗口支持**
- **V2签名**
- 增强的Java8语言模式
- 夜间模式

Android8.0（O）新特性

- **优化通知**

  通知渠道 (Notification Channel) 通知标志 休眠 通知超时 通知设置 通知清除

- **画中画模式**：清单中Activity设置android:supportsPictureInPicture
- **后台限制**
- 自动填充框架
- 系统优化
- 等等优化很多

Android9.0（P）新特性

- **室内WIFI定位**
- **“刘海”屏幕支持**
- 安全增强
- 等等优化很多

Android10.0（Q）目前曝光的新特性

- **夜间模式**：包括手机上的所有应用都可以为其设置暗黑模式。

- **桌面模式**：提供类似于PC的体验，但是远远不能代替PC。
- **屏幕录制**：通过长按”电源”菜单中的"屏幕快照"来开启。

11、Json

JSON的全称是JavaScript Object Notation，也就是JavaScript 对象表示法 JSON是存储和交换文本信息的语法，类似XML，但是比XML更小、更快，更易解析 JSON是轻量级的文本数据交换格式，独立于语言，具有可描述性，更易理解，对象可以包含多个名称/值对，比如：

```
{"name":"zhangsan" , "age":25}
```

使用谷歌的GSON包进行解析，在 Android Studio 里引入依赖：

```
compile 'com.google.code.gson:gson:2.7'
```

值得注意的是实体类中变量名称必须和json中的值名字相同。

使用示例：

1、解析成实体类：

```
Gson gson = new Gson();
Student student = gson.fromJson(json1, Student.class);
```

2、解析成int数组：

```
Gson gson = new Gson();
int[] ages = gson.fromJson(json2, int[].class);
```

3、直接解析成List.

```
Gson gson = new Gson();
List<Integer> ages = gson.fromJson(json2,  newTypeToken<List<Integer>>()
{}.getType);

Gson gson = new Gson();
List<Student> students = gson.fromJson(json3, newTypeToke<List<Student>>()
{}.getType);
```

优点：

- 轻量级的数据交换格式
- 读写更加容易

- 易于机器的解析和生成

缺点:

- 语义性较差，不如 xml 直观

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

DOM解析

优点:

1.XML树在内存中完整存储,因此可以直接修改其数据结构.

2.可以通过该解析器随时访问XML树中的任何一个节点.

3.DOM解析器的API在使用上也相对比较简单.

缺点:

如果XML文档体积比较大时,将文档读入内存是非消耗系统资源的.

使用场景:

- DOM 是与平台和语言无关的方式表示 XML文档的官方 W3C 标准.
- DOM 是以层次结构组织的节点的集合.这个层次结构允许开人员在树中寻找特定信息.分析该结构通常需要加载整个文档和构造层次结构,然后才能进行任何工作.
- DOM 是基于对象层次结构的.

SAX解析

优点:

SAX 对内存的要求比较低,因为它让开发人员自己来决定所要处理的标签.特别是当开发人员只需要处理文档中包含的部分数据时,SAX 这种扩展能力得到了更好的体现.

缺点:

用SAX方式进行XML解析时,需要顺序执行,所以很难访问同一文档中的不同数据.此外,在基于该方式的解析编码程序也相对复杂.

使用场景:

对于含有数据量十分巨大,而又不用对文档的所有数据行遍历或者分析的时候,使用该方法十分有效.该方法不将整个文档读入内存,而只需读取到程序所需的文档标记处即可.

Xmlpull解析

android SDK提供了xmlpullapi,xmlpull和sax类似,是基于流（stream）操作文件,后者根据节点事件回调开发者编写的处理程序.因为是基于流的处理,因此xmlpull和sax都比较节约内存资源,不会像dom那样要把所有节点以对象树的形式展现在内存中.xmpull比sax更简明,而且不需要扫描完整个流.

13、Jar和Aar的区别

Jar包里面只有代码，aar里面不光有代码还包括资源文件，比如 drawable 文件，xml资源文件。对于一些不常变动的 Android Library，我们可以直接引用 aar，加快编译速度。

14、Android为每个应用程序分配的内存大小是多少

android程序内存一般限制在16M，也有的是24M。近几年手机发展较快，一般都会分配两百兆左右，和具体机型有关。

15、更新UI方式

- Activity.runOnUiThread(Runnable)
- View.post(Runnable)，View.postDelay(Runnable, long)（可以理解为在当前操作视图UI线程添加队列）
- Handler
- AsyncTask
- Rxjava
- LiveData

16、ContentProvider使用方法。

进行跨进程通信，实现进程间的数据交互和共享。通过Context 中 getContentResolver() 获得实例，通过 Uri匹配进行数据的增删改查。ContentProvider使用表的形式来组织数据，无论数据的来源是什么，ConentProvider 都会认为是一种表，然后把数据组织成表格。

17、Thread、AsyncTask、IntentService的使用场景与特点。

1. Thread线程，独立运行与于 Activity 的，当Activity 被 finish 后，如果没有主动停止Thread或者 run 方法没有执行完，其会一直执行下去。

2. AsyncTask 封装了两个线程池和一个Handler（SerialExecutor用于排队，THREAD_POOL_EXECUTOR为真正的执行任务，Handler将工作线程切换到主线程），其必须在 UI线程中创建，execute 方法必须在 UI线程中执行，一个任务实例只允许执行一次，执行多次抛出异常，用于网络请求或者简单数据处理。

3. IntentService：处理异步请求，实现多线程，在onHandleIntent中处理耗时操作，多个耗时任务会依次执行，执行完毕自动结束。

18、Merge、ViewStub 的作用。

Merge: 减少视图层级，可以删除多余的层级。

ViewStub: 按需加载，减少内存使用量、加快渲染速度、不支持 merge 标签。

19、activity的startActivity和context的startActivity区别?

(1)、从Activity中启动新的Activity时可以直接mContext.startActivity(intent)就好

(2)、如果从其他Context中启动Activity则必须给intent设置Flag:

```
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK) ;
mContext.startActivity(intent);
```

20、怎么在Service中创建Dialog对话框?

1.在我们取得Dialog对象后，需给它设置类型，即:

```
dialog.getWindow().setType(WindowManager.LayoutParams.TYPE_SYSTEM_ALERT)
```

2.在Manifest中加上权限:

```
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINOW" />
```

21、Asset目录与res目录的区别?

assets：不会在 R 文件中生成相应标记，存放到这里的资源在打包时会打包到程序安装包中。（通过 AssetManager 类访问这些文件）

res：会在 R 文件中生成 id 标记，资源在打包时如果使用到则打包到安装包中，未用到不会打入安装包中。

res/anim：存放动画资源。

res/raw：和 asset 下文件一样，打包时直接打入程序安装包中（会映射到 R 文件中）。

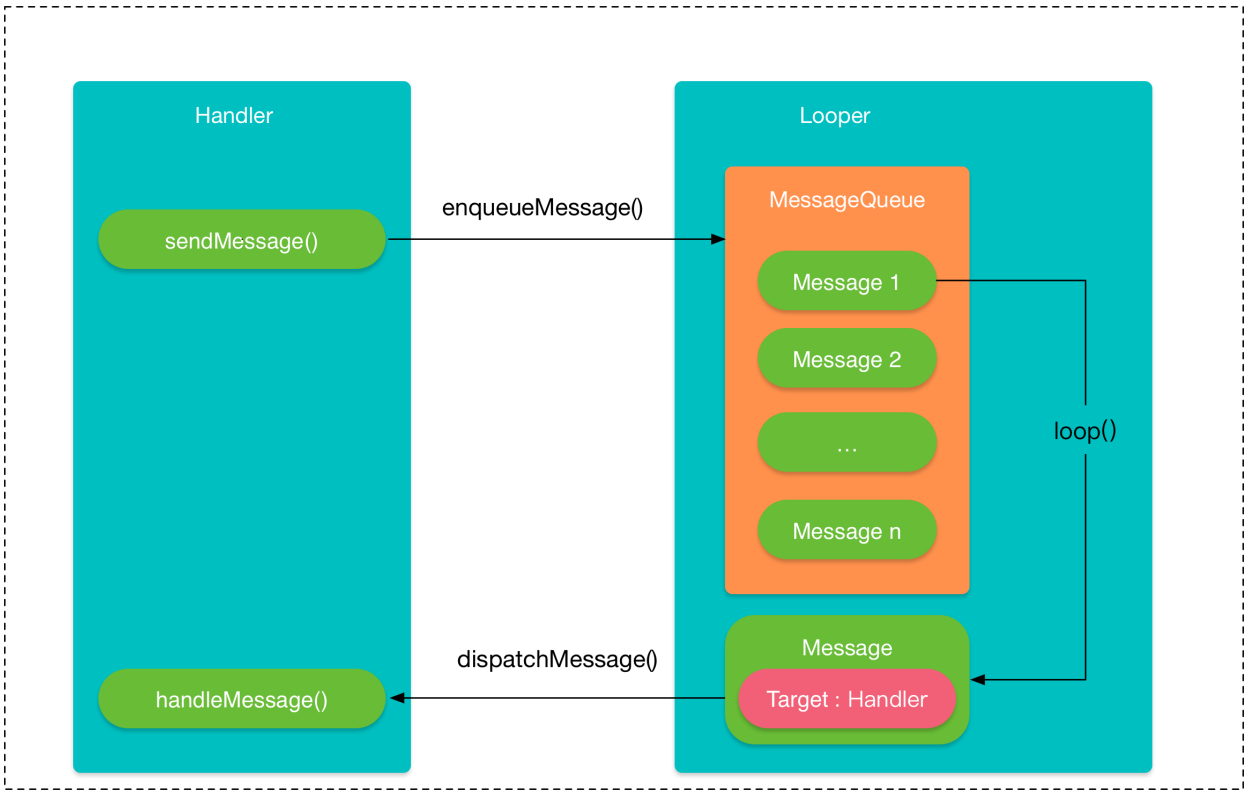
22、Android怎么加速启动Activity?

- onCreate() 中不执行耗时操作 把页面显示的 View 细分一下，放在 AsyncTask 里逐步显示，用 Handler 更好。这样用户的看到的就是有层次有步骤的一个个的 View 的展示，不会是先看到一个黑屏，然后一下显示所有 View。最好做成动画，效果更自然。
- 利用多线程的目的就是尽可能的减少 onCreate() 和 onReume() 的时间，使得用户能尽快看到页面，操作页面。

- 减少主线程阻塞时间。
- 提高 Adapter 和 AdapterView 的效率。
- 优化布局文件。

23、Handler机制

Android消息循环流程图如下所示：



主要涉及的角色如下所示：

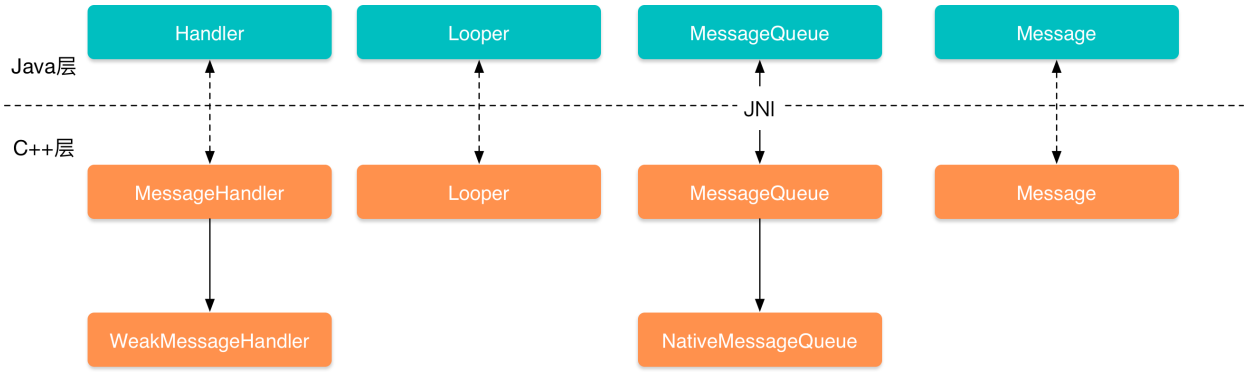
- message：消息。
- MessageQueue：消息队列，负责消息的存储与管理，负责管理由 Handler 发送过来的 Message。读取会自动删除消息，单链表维护，插入和删除上有优势。在其next()方法中会无限循环，不断判断是否有消息，有就返回这条消息并移除。
- Looper：消息循环器，负责关联线程以及消息的分发，在该线程下从MessageQueue获取 Message，分发给Handler，Looper创建的时候会创建一个MessageQueue，调用loop()方法的时候消息循环开始，其中会不断调用messageQueue的next()方法，当有消息就处理，否则阻塞在messageQueue的next()方法中。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()方法也就跟着退出。
- Handler：消息处理器，负责发送并处理消息，面向开发者，提供 API，并隐藏背后实现的细节。

整个消息的循环流程还是比较清晰的，具体说来：

- 1、Handler通过sendMessage()发送消息Message到消息队列MessageQueue。

- 2、Looper通过loop()不断提取触发条件的Message，并将Message交给对应的target handler来处理。
- 3、target handler调用自身的handleMessage()方法来处理Message。

事实上，在整个消息循环的流程中，并不只有Java层参与，很多重要的工作都是在C++层来完成的。我们来看下这些类的调用关系。



注：虚线表示关联关系，实线表示调用关系。

在这些类中MessageQueue是Java层与C++层维系的桥梁，MessageQueue与Looper相关功能都通过MessageQueue的Native方法来完成，而其他虚线连接的类只有关联关系，并没有直接调用的关系，它们发生关联的桥梁是MessageQueue。

总结

- Handler 发送的消息由 MessageQueue 存储管理，并由 Looper 负责回调消息到 handleMessage()。
- 线程的转换由 Looper 完成，handleMessage() 所在线程由 Looper.loop() 调用者所在线程决定。

Handler 引起的内存泄露原因以及最佳解决方案

Handler 允许我们发送延时消息，如果在延时期间用户关闭了 Activity，那么该 Activity 会泄露。 这个泄露是因为 Message 会持有 Handler，而又因为 Java 的特性，内部类会持有外部类，使得 Activity 会被 Handler 持有，这样最终就导致 Activity 泄露。

解决：将 Handler 定义成静态的内部类，在内部持有 Activity 的弱引用，并在Acitivity的onDestroy()中调用handler.removeCallbacksAndMessages(null)及时移除所有消息。

为什么我们能在主线程直接使用 Handler，而不需要创建 Looper ?

通常我们认为 ActivityThread 就是主线程。事实上它并不是一个线程，而是主线程操作的管理者。在 ActivityThread.main() 方法中调用了 Looper.prepareMainLooper() 方法创建了 主线程的 Looper ,并且调用了 loop() 方法，所以我们就可以直接使用 Handler 了。

因此我们可以利用 Callback 这个拦截机制来拦截 Handler 的消息。如大部分插件化框架中Hook ActivityThread.mH 的处理。

主线程的 Looper 不允许退出

主线程不允许退出，退出就意味 APP 要挂。

Handler 里藏着的 Callback 能干什么?

Handler.Callback 有优先处理消息的权利 ，当一条消息被 Callback 处理并拦截（返回 true），那么 Handler 的 handleMessage(msg) 方法就不会被调用了；如果 Callback 处理了消息，但是并没有拦截，那么就意味着一个消息可以同时被 Callback 以及 Handler 处理。

创建 Message 实例的最佳方式

为了节省开销，Android 给 Message 设计了回收机制，所以我们在使用的时候尽量复用 Message ，减少内存消耗：

- 通过 Message 的静态方法 Message.obtain()；
- 通过 Handler 的公有方法 handler.obtainMessage()。

子线程里弹 Toast 的正确姿势

本质上是因为 Toast 的实现依赖于 Handler，按子线程使用 Handler 的要求修改即可，同理的还有 Dialog。

妙用 Looper 机制

- 将 Runnable post 到主线程执行；
- 利用 Looper 判断当前线程是否是主线程。

主线程的死循环一直运行是不是特别消耗CPU资源呢?

并不是，这里就涉及到Linux pipe/epoll机制，简单说就是在主线程的MessageQueue没有消息时，便阻塞在loop的queue.next()中的nativePollOnce()方法里，此时主线程会释放CPU资源进入休眠状态，直到下个消息到达或者有事务发生，通过往pipe管道写端写入数据来唤醒主线程工作。这里采用的epoll机制，是一种IO多路复用机制，可以同时监控多个描述符，当某个描述符就绪(读或写就绪)，则立刻通知相应程序进行读或写操作，本质是同步I/O，即读写是阻塞的。所以说，主线程大多数时候都是处于休眠状态，并不会消耗大量CPU资源。

handler postDelay这个延迟是怎么实现的?

handler.postDelay并不是先等待一定的时间再放入到MessageQueue中，而是直接进入MessageQueue，以MessageQueue的时间顺序排列和唤醒的方式结合实现的。

Handler 都没搞懂，拿什么去跳槽啊?

24、程序A能否接收到程序B的广播?

能，使用全局的BroadCastRecevier能进行跨进程通信，但是注意它只能被动接收广播。此外，LocalBroadCastRecevier只限于本进程的广播间通信。

25、数据加载更多涉及到分页，你是怎么实现的?

分页加载就是一页一页加载数据，当滑动到底部、没有更多数据加载的时候，我们可以手动调用接口，重新刷新RecyclerView。

26、通过google提供的Gson解析json时，定义JavaBean的规则是什么?

1). 实现序列化 Serializable

2). 属性私有化，并提供get，set方法

3). 提供无参构造

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）

27、json解析方式的两种区别?

1，SDK提供JSONArray，JSONObject

2，google提供的 Gson 通过fromJson()实现对象的反序列化（即将json串转换为对象类型） 通过toJson()实现对象的序列化 （即将对象类型转换为json串）

28、线程池的相关知识。

Android中的线程池都是直接或间接通过配置ThreadPoolExecutor来实现不同特性的线程池.Android中最常见的类具有不同特性的线程池分别为FixThreadPool、CachedhreadPool、SingleThreadPool、ScheduleThreadExecutr.

1).FixThreadPool

只有核心线程,并且数量固定的,也不会被回收,所有线程都活动时,因为队列没有限制大小,新任务会等待执行.

优点:更快的响应外界请求.

2).SingleThreadPool

只有一个核心线程,确保所有的任务都在同一线程中按序完成.因此不需要处理线程同步的问题.

3).CachedThreadPool

只有非核心线程,最大线程数非常大,所有线程都活动时会为新任务创建新线程,否则会利用空闲线程(60s空闲时间,过了就会被回收,所以线程池中有0个线程的可能)处理任务.

优点:任何任务都会被立即执行(任务队列SynchronousQuue相当于一个空集合);比较适合执行大量的耗时较少的任务.

4).ScheduledThreadPool

核心线程数固定,非核心线程（闲着没活干会被立即回收数）没有限制.

优点:执行定时任务以及有固定周期的重复任务

29、内存泄露，怎样查找，怎么产生的内存泄露?

1.资源对象没关闭造成的内存泄漏

描述： 资源性对象比如(Cursor，File文件等)往往都用了一些缓冲，我们在不使用的时候，应该及时关闭它们，以便它们的缓冲及时回收内存。它们的缓冲不仅存在于 java虚拟机内，还存在于java虚拟机外。如果我们仅仅是把它的引用设置为null,而不关闭它们，往往会造成内存泄漏。因为有些资源性对象，比如SQLiteCursor(在析构函数finalize(),如果我们没有关闭它，它自己会调close()关闭)，如果我们没有关闭它，系统在回收它时也会关闭它，但是这样的效率太低了。因此对于资源性对象在不使用的时候，应该调用它的close()函数，将其关闭掉，然后才置为null.在我们的程序退出时一定要确保我们的资源性对象已经关闭。

程序中经常会进行查询数据库的操作，但是经常会有使用完毕Cursor后没有关闭的情况。如果我们的查询结果集比较小，对内存的消耗不容易被发现，只有在常时间大量操作的情况下才会复现内存问题，这样就会给以后的测试和问题排查带来困难和风险。

2.构造Adapter时，没有使用缓存的convertView

描述： 以构造ListView的BaseAdapter为例，在BaseAdapter中提供了方法： public View getView(int position, ViewconvertView, ViewGroup parent) 来向ListView提供每一个item所需要的view对象。初始时ListView会从BaseAdapter中根据当前的屏幕布局实例化一定数量的 view对象，同时ListView会将这些view对象缓存起来。当向上滚动ListView时，原先位于最上面的list item的view对象会被回收，然后被用来构造新出现的最下面的list item。这个构造过程就是由getView()方法完成的，getView()的第二个形参View convertView就是被缓存起来的list item的view对象(初始化时缓存中没有view对象则convertView是null)。由此可以看出，如果我们不去使用 convertView，而是每次都在getView()中重新实例化一个View对象的话，即浪费资源也浪费时间，也会使得内存占用越来越大。 ListView回收list item的view对象的过程可以查看: android.widget.AbsListView.java --> voidaddScrapView(View scrap) 方法。 示例代码：

```
public View getView(int position, ViewconvertView, ViewGroup parent) {
View view = new Xxx(...);
... ...
return view;
}
```

修正示例代码：

```
public View getView(int position, ViewconvertView, ViewGroup parent) {
View view = null;
if (convertView != null) {
view = convertView;
populate(view, getItem(position));
...
} else {
view = new Xxx(...);
...
}


return view;
}
```

3.Bitmap对象不在使用时调用recycle()释放内存

描述： 有时我们会手工的操作Bitmap对象，如果一个Bitmap对象比较占内存，当它不在被使用的时候，可以调用Bitmap.recycle()方法回收此对象的像素所占用的内存，但这不是必须的，视情况而定。可以看一下代码中的注释：

/* •Free up the memory associated with thisbitmap's pixels, and mark the •bitmap as "dead", meaning itwill throw an exception if getPixels() or •setPixels() is called, and will drawnothing. This operation cannot be •reversed, so it should only be called ifyou are sure there are no •further uses for the bitmap. This is anadvanced call, and normally need •not be called, since the normal GCprocess will free up this memory when •there are no more references to thisbitmap. /

4.试着使用关于application的context来替代和activity相关的context

这是一个很隐晦的内存泄漏的情况。有一种简单的方法来避免context相关的内存泄漏。最显著地一个是避免context逃出他自己的范围之外。使用Application context。这个context的生存周期和你的应用的生存周期一样长，而不是取决于activity的生存周期。如果你想保持一个长期生存的对象，并且这个对象需要一个context,记得使用application对象。你可以通过调用 Context.getApplicationContext() or Activity.getApplication()来获得。更多的请看这篇文章如何避免 Android内存泄漏。

5.注册没取消造成的内存泄漏

一些Android程序可能引用我们的Anroid程序的对象(比如注册机制)。即使我们的Android程序已经结束了，但是别的引用程序仍然还有对我们的Android程序的某个对象的引用，泄漏的内存依然不能被垃圾回收。调用registerReceiver后未调用unregisterReceiver。 比如:假设我们希望在锁屏界面(LockScreen)中，监听系统中的电话服务以获取一些信息(如信号强度等)，则可以在LockScreen中定义一个PhoneStateListener的对象，同时将它注册到TelephonyManager服务中。对于LockScreen对象，当需要显示锁屏界面的时候就会创建一个LockScreen对象，而当锁屏界面消失的时候LockScreen对象就会被释放掉。 但是如果在释放 LockScreen对象的时候忘记取消我们之前注册的PhoneStateListener对象，则会导致LockScreen无法被垃圾回收。如果不断的使锁屏界面显示和消失，则最终会由于大量的LockScreen对象没有办法被回收而引起OutOfMemory,使得system_process 进程挂掉。 虽然有些系统程序，它本身好像是可以自动取消注册的(当然不及时)，但是我们还是应该在我们的程序中明确的取消注册，程序结束时应该把所有的注册都取消掉。

6.集合中对象没清理造成的内存泄漏

我们通常把一些对象的引用加入到了集合中，当我们不需要该对象时，并没有把它的引用从集合中清理掉，这样这个集合就会越来越大。如果这个集合是static的话，那情况就更严重了。

查找内存泄漏可以使用Android Studio 自带的AndroidProfiler工具或MAT，也可以使用Square产品的LeakCanary.

1、使用AndroidProfiler的MEMORY工具：

运行程序，对每一个页面进行内存分析检查。首先，反复打开关闭页面5次，然后收到GC（点击Profile MEMORY左上角的垃圾桶图标），如果此时total内存还没有恢复到之前的数值，则可能发生了内存泄露。此时，再点击Profile MEMORY左上角的垃圾桶图标旁的heap dump按钮查看当前的内存堆栈情况，选择按包名查找，找到当前测试的Activity，如果引用了多个实例，则表明发生了内存泄露。

2、使用MAT：

1、运行程序，所有功能跑一遍，确保没有改出问题，完全退出程序，手动触发GC，然后使用adb shell dumpsys meminfo packagename -d命令查看退出界面后Objects下的Views和Activities数目是否为0，如果不是则通过Leakcanary检查可能存在内存泄露的地方，最后通过MAT分析，如此反复，改善满意为止。

1、在使用MAT之前，先使用as的Profile中的Memory去获取要分析的堆内存快照文件.hprof，如果要测试某个页面是否产生内存泄漏，可以先dump出没进入该页面的内存快照文件.hprof，然后，通常执行5次进入/退出该页面，然后再dump出此刻的内存快照文件.hprof，最后，将两者比较，如果内存相除明显，则可能发生内存泄露。（注意:MAT需要标准的.hprof文件，因此在as的Profiler中GC后dump出的内存快照文件.hprof必须手动使用android sdk platform-tools下的hprof-conv程序进行转换才能被MAT打开）

2、然后，使用MAT打开前面保存的2份.hprof文件，打开Overview界面，在Overview界面下面有4中action，其中最常用的就是Histogram和Dominator Tree。

Dominator Tree：支配树，按对象大小降序列出对象和其所引用的对象，注重引用关系分析。选择Group by package，找到当前要检测的类（或者使用顶部的Regex直接搜索），查看它的Object数目是否正确，如果多了，则判断发生了内存泄露。然后，右击该类，选择Merge Shortest Paths to GC Root中的exclude all phantom/weak/soft etc.references选项来查看该类的GC强引用链。最后，通过引用链即可看到最终强引用该类的对象。

Histogram：直方图注重量的分析。使用方式与Dominator Tree类似。

3、对比hprof文件，检测出复杂情况下的内存泄露：

通用对比方式：在Navigation History下面选择想要对比的dominator_tree/histogram，右击选择Add to Compare Basket，然后在Compare Basket一栏中点击红色感叹号（Compare the results）生成对比表格（Compared Tables），在顶部Regex输入要检测的类，查看引用关系或对象数量去进行分析即可。

针对于Historam的快速对比方式：直接选择Histogram上方的Compare to another Heap Dump选择要比较的hprof文件的Historam即可。

30、类的初始化顺序依次是?

（静态变量、静态代码块）>（变量、代码块）>构造方法

31、JSON的结构?

json是一种轻量级的数据交换格式， json简单说就是对象和数组，所以这两种结构就是对象和数组两种结构，通过这两种结构可以表示各种复杂的结构

1、对象：对象表示为“{}”扩起来的内容，数据结构为 {key：value,key：value,...}的键值对的结构，在面向对象的语言中，key为对象的属性，value为对应的属性值，所以很容易理解，取值方法为 对象.key 获取属性值，这个属性值的类型可以是 数字、字符串、数组、对象几种。

2、数组：数组在json中是中括号“[]”扩起来的内容，数据结构为 ["java","javascript","vb",...]，取值方式和所有语言中一样，使用索引获取，字段值的类型可以是 数字、字符串、数组、对象几种。 经过对象、数组2种结构就可以组合成复杂的数据结构了。

32、ViewPager使用细节，如何设置成每次只初始化当前的Fragment，其他的不初始化（提示：Fragment懒加载）？

自定义一个 LazyLoadFragment 基类，利用 setUserVisibleHint 和 生命周期方法，通过对 Fragment 状态判断，进行数据加载，并将数据加载的接口提供开放出去，供子类使用。然后在子类 Fragment 中实现 requestData 方法即可。这里添加了一个 isDataLoaded 变量，目的是避免重复加载数据。考虑到有时候需要刷新数据的问题，便提供了一个用于强制刷新的参数判断。

```
public abstract class LazyLoadFragment extends BaseFragment {
    protected boolean isViewInitiated;
    protected boolean isDataLoaded;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        isViewInitiated = true;
        prepareRequestData();
    }
    @Override
    public void setUserVisibleHint(boolean isVisibleToUser) {
        super.setUserVisibleHint(isVisibleToUser);
        prepareRequestData();
    }
    public abstract void requestData();
    public boolean prepareRequestData() {
        return prepareRequestData(false);
    }
    public boolean prepareRequestData(boolean forceUpdate) {
        if (getUserVisibleHint() && isViewInitiated && (!isDataLoaded ||
forceUpdate)) {
            requestData();
            isDataLoaded = true;
            return true;
        }
        return false;
    }
}
```

35、Android为什么引入Parcelable?

可以肯定的是，两者都是支持序列化和反序列化的操作。

两者最大的区别在于 存储媒介的不同，Serializable 使用 I/O 读写存储在硬盘上，而 Parcelable 是直接 在内存中读写。很明显，内存的读写速度通常大于 IO 读写，所以在 Android 中传递数据优先选择 Parcelable。

Serializable 会使用反射，序列化和反序列化过程需要大量 I/O 操作， Parcelable 自己实现封送和解封（marshalled &unmarshalled）操作不需要用反射，数据也存放在 Native 内存中，效率要快很多。

36、有没有尝试简化Parcelable的使用?

使用Parcelable插件（Android Parcelable code generator）进行实体类的序列化的实现。

37、Bitmap 使用时候注意什么?

1、要选择合适的图片规格（bitmap类型）：

```
ALPHA_8    每个像素占用1byte内存
ARGB_4444  每个像素占用2byte内存
ARGB_8888  每个像素占用4byte内存（默认）
RGB_565  每个像素占用2byte内存
```

2、降低采样率。BitmapFactory.Options 参数inSampleSize的使用，先把options.inJustDecodeBounds设为true，只是去读取图片的大小，在拿到图片的大小之后和要显示的大小做比较通过calculateInSampleSize()函数计算inSampleSize的具体值，得到值之后。options.inJustDecodeBounds设为false读图片资源。

3、复用内存。即，通过软引用(内存不够的时候才会回收掉)，复用内存块，不需要再重新给这个bitmap申请一块新的内存，避免了一次内存的分配和回收，从而改善了运行效率。

4、使用recycle()方法及时回收内存。

5、压缩图片。

38、Oom 是否可以try catch ?

只有在一种情况下，这样做是可行的：

在try语句中声明了很大的对象，导致OOM，并且可以确认OOM是由try语句中的对象声明导致的，那么在catch语句中，可以释放掉这些对象，解决OOM的问题，继续执行剩余语句。

但是这通常不是合适的做法。

Java中管理内存除了显式地catch OOM之外还有更多有效的方法：比如SoftReference, WeakReference, 硬盘缓存等。 在JVM用光内存之前，会多次触发GC，这些GC会降低程序运行的效率。 如果OOM的原因不是try语句中的对象（比如内存泄漏），那么在catch语句中会继续抛出OOM。

39、多进程场景遇见过么?

1、在新的进程中，启动前台Service，播放音乐。 2、一个成熟的应用一定是多模块化的。首先多进程开发能为应用解决了OOM问题，因为Android对内存的限制是针对于进程的，所以，当我们需要加载大图之类的操作，可以在新的进程中去执行，避免主进程OOM。而且假如图片浏览进程打开了一个过大的图片，java heap 申请内存失败，该进程崩溃并不影响我主进程的使用。

40、Canvas.save()跟Canvas.restore()的调用时机

save：用来保存Canvas的状态。save之后，可以调用Canvas的平移、放缩、旋转、错切、裁剪等操作。

restore：用来恢复Canvas之前保存的状态。防止save后对Canvas执行的操作对后续的绘制有影响。

save和restore要配对使用（restore可以比save少，但不能多），如果restore调用次数比save多，会引发Error。save和restore操作执行的时机不同，就能造成绘制的图形不同。

41、数据库升级增加表和删除表都不涉及数据迁移，但是修改表涉及到对原有数据进行迁移。升级的方法如下所示：

将现有表命名为临时表。 创建新表。 将临时表的数据导入新表。 删除临时表。

如果是跨版本数据库升级，可以有两种方式，如下所示：

逐级升级，确定相邻版本与现在版本的差别，V1升级到V2,V2升级到V3，依次类推。 跨级升级，确定每个版本与现在数据库的差别，为每个case编写专门升级大代码。

```
public class DBservice extends SQLiteOpenHelper{
    private String CREATE_BOOK = "create table book(bookId integer
primarykey,bookName text);";
    private String CREATE_TEMP_BOOK = "alter table book rename to _temp_book";
    private String INSERT_DATA = "insert into book select *,'' from
_temp_book";
    private String DROP_BOOK = "drop table _temp_book";
    public DBservice(Context context, String name, CursorFactory factory,int
version) {

    super(context, name, factory, version);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_BOOK);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    switch (newVersion) {
        case 2:
            db.beginTransaction();

            db.execSQL(CREATE_TEMP_BOOK);
            db.execSQL(CREATE BOOK);
```

```
                db.execSQL(INSERT_DATA);
                db.execSQL(DROP_BOOK);

                db.setTransactionSuccessful();
                db.endTransaction();

                break;
        }
    }
```

42、编译期注解跟运行时注解

运行期注解(RunTime)利用反射去获取信息还是比较损耗性能的，对应@Retention（RetentionPolicy.RUNTIME）。

编译期(Compile time)注解，以及处理编译期注解的手段APT和Javapoet，对应@Retention(RetentionPolicy.CLASS)。 其中apt+javaPoet目前也是应用比较广泛，在一些大的开源库，如EventBus3.0+,页面路由 ARout、Dagger、Retrofit等均有使用的身影，注解不仅仅是通过反射一种方式来使用，也可以使用APT在编译期处理

43、bitmap recycler 相关

在Android中，Bitmap的存储分为两部分，一部分是Bitmap的数据，一部分是Bitmap的引用。 在Android2.3时代，Bitmap的引用是放在堆中的，而Bitmap的数据部分是放在栈中的，需要用户调用recycle方法手动进行内存回收，而在Android2.3之后，整个Bitmap，包括数据和引用，都放在了堆中，这样，整个Bitmap的回收就全部交给GC了，这个recycle方法就再也不需要使用了。

bitmap recycler引发的问题：当图像的旋转角度小余两个像素点之间的夹角时，图像即使旋转也无法显示，因此，系统完全可以认为图像没有发生变化。这时系统就直接引用同一个对象来进行操作，避免内存浪费。

44、强引用置为null，会不会被回收?

不会立即释放对象占用的内存。 如果对象的引用被置为null，只是断开了当前线程栈帧中对该对象的引用关系，而 垃圾收集器是运行在后台的线程，只有当用户线程运行到安全点(safe point)或者安全区域才会扫描对象引用关系，扫描到对象没有被引用则会标记对象，这时候仍然不会立即释放该对象内存，因为有些对象是可恢复的（在 finalize方法中恢复引用 ）。只有确定了对象无法恢复引用的时候才会清除对象内存。

45、Bundle传递数据为什么需要序列化?

序列化，表示将一个对象转换成可存储或可传输的状态。序列化的原因基本三种情况：

1.永久性保存对象，保存对象的字节序列到本地文件中；

2.对象在网络中传递；

3.对象在IPC间传递。

46、广播传输的数据是否有限制，是多少，为什么要限制?

Intent在传递数据时是有大小限制的，大约限制在1MB之内，你用Intent传递数据，实际上走的是跨进程通信（IPC），跨进程通信需要把数据从内核copy到进程中，每一个进程有一个接收内核数据的缓冲区，默认是1M；如果一次传递的数据超过限制，就会出现异常。

不同厂商表现不一样有可能是厂商修改了此限制的大小，也可能同样的对象在不同的机器上大小不一样。

传递大数据，不应该用Intent；考虑使用ContentProvider或者直接匿名共享内存。简单情况下可以考虑分段传输。

47、是否了解硬件加速?

硬件加速就是运用GPU优秀的运算能力来加快渲染的速度，而通常的基于软件的绘制渲染模式是完全利用CPU来完成渲染。

1.硬件加速是从API 11引入，API 14之后才默认开启。对于标准的绘制操作和控件都是支持的，但是对于自定义View的时候或者一些特殊的绘制函数就需要考虑是否需要关闭硬件加速。

2.我们面对不支持硬件加速的情况，就需要限制硬件加速，这个兼容性的问题是因为硬件加速是把View的绘制函数转化为使用OpenGL的函数来进完成实际的绘制的，那么必然会存在OpenGL中不支持原始回执函数的情况，对于这些绘制函数，就会失效。

3.硬件加速的消耗问题，因为是使用OpenGL，需要把系统中OpenGL加载到内存中，OpenGL API调用就会占用8MB，而实际上会占用更多内存，并且使用了硬件必然增加耗电量了。

4.硬件加速的优势还有display list的设计，使用这个我们不需要每次重绘都执行大量的代码，基于软件的绘制模式会重绘脏区域内的所有控件，而display只会更新列表，然后绘制列表内的控件。

5. CPU更擅长复杂逻辑控制，而GPU得益于大量ALU和并行结构设计，更擅长数学运算。

48、ContentProvider的权限管理(读写分离，权限控制-精确到表级，URL控制)。

对于ContentProvider暴露出来的数据，应该是存储在自己应用内存中的数据，对于一些存储在外部存储器上的数据，并不能限制访问权限，使用ContentProvider就没有意义了。对于ContentProvider而言，有很多权限控制，可以在AndroidManifest.xml文件中对节点的属性进行配置，一般使用如下一些属性设置：

- android:grantUriPermssions:临时许可标志。

- android:permission:Provider读写权限。
- android:readPermission:Provider的读权限。
- android:writePermission:Provider的写权限。
- android:enabled:标记允许系统启动Provider。
- android:exported:标记允许其他应用程序使用这个Provider。
- android:multiProcess:标记允许系统启动Provider相同的进程中调用客户端。

49、Fragment状态保存

Fragment状态保存入口:

1、Activity的状态保存, 在Activity的onSaveInstanceState()里, 调用了FragmentManger的saveAllState()方法, 其中会对mActive中各个Fragment的实例状态和View状态分别进行保存.

2、FragmentManager还提供了public方法: saveFragmentInstanceState(), 可以对单个Fragment进行状态保存, 这是提供给我们用的。

3、FragmentManager的moveToState()方法中, 当状态回退到ACTIVITY_CREATED, 会调用saveFragmentViewState()方法, 保存View的状态.

50、直接在Activity中创建一个thread跟在service中创建一个thread之间的区别?

在Activity中被创建：该Thread的就是为这个Activity服务的，完成这个特定的Activity交代的任务，主动通知该Activity一些消息和事件，Activity销毁后，该Thread也没有存活的意义了。

在Service中被创建：这是保证最长生命周期的Thread的唯一方式，只要整个Service不退出，Thread就可以一直在后台执行，一般在Service的onCreate()中创建，在onDestroy()中销毁。所以，在Service中创建的Thread，适合长期执行一些独立于APP的后台任务，比较常见的就是：在Service中保持与服务器端的长连接。

51、如何计算一个Bitmap占用内存的大小，怎么保证加载Bitmap不产生内存溢出?

```
Bitamp 占用内存大小 = 宽度像素 x （inTargetDensity / inDensity） x 高度像素 x （inTargetDensity / inDensity）x 一个像素所占的内存
```

注：这里inDensity表示目标图片的dpi（放在哪个资源文件夹下），inTargetDensity表示目标屏幕的dpi，所以你可以发现inDensity和inTargetDensity会对Bitmap的宽高进行拉伸，进而改变Bitmap占用内存的大小。

在Bitmap里有两个获取内存占用大小的方法。

getByteCount()：API12 加入，代表存储 Bitmap 的像素需要的最少内存。 getAllocationByteCount()：API19 加入，代表在内存中为 Bitmap 分配的内存大小，代替了 getByteCount() 方法。 在不复用 Bitmap 时，getByteCount() 和 getAllocationByteCount 返回的结果是一样的。在通过复用 Bitmap 来解码图片时，那么 getByteCount() 表示新解码图片占用内存的大 小，getAllocationByteCount() 表示被复用 Bitmap 真实占用的内存大小（即 mBuffer 的长度）。

为了保证在加载Bitmap的时候不产生内存溢出，可以使用BitmapFactory进行图片压缩，主要有以下几个参数：

BitmapFactory.Options.inPreferredConfig：将ARGB_8888改为RGB_565，改变编码方式，节约内存。 BitmapFactory.Options.inSampleSize：缩放比例，可以参考Luban那个库，根据图片宽高计算出合适的缩放比例。 BitmapFactory.Options.inPurgeable：让系统可以内存不足时回收内存。

52、对于应用更新这块是如何做的？(灰度，强制更新，分区域更新)

1、通过接口获取线上版本号，versionCode 2、比较线上的versionCode 和本地的versionCode，弹出更新窗口 3、下载APK文件（文件下载） 4、安装APK

灰度： (1)找单一渠道投放特别版本。 (2)做升级平台的改造，允许针对部分用户推送升级通知甚至版本强制升级。 (3)开放单独的下载入口。 (4)是两个版本的代码都打到app包里，然后在app端植入测试框架，用来控制显示哪个版本。测试框架负责与服务器端api通信，由服务器端控制app上A/B版本的分布，可以实现指定的一组用户看到A版本，其它用户看到B版本。服务端会有相应的报表来显示A/B版本的数量和效果对比。最后可以由服务端的后台来控制，全部用户在线切换到A或者B版本~

无论哪种方法都需要做好版本管理工作，分配特别的版本号以示区别。 当然，既然是做灰度，数据监控（常规数据、新特性数据、主要业务数据）还是要做到位，该打的数据桩要打。 还有，灰度版最好有收回的能力，一般就是强制升级下一个正式版。

强制更新:一般的处理就是进入应用就弹窗通知用户有版本更新，弹窗可以没有取消按钮并不能取消。这样用户就只能选择更新或者关闭应用了，当然也可以添加取消按钮，但是如果用户选择取消则直接退出应用。

增量更新：bsdiff：二进制差分工具bsdiff是相应的补丁合成工具,根据两个不同版本的二进制文件，生成补丁文件.patch文件。通过bspatch使旧的apk文件与不定文件合成新的apk。 注意通过apk文件的md5值进行区分版本。

53、请解释安卓为啥要加签名机制。

1、发送者的身份认证 由于开发商可能通过使用相同的 Package Name 来混淆替换已经安装的程序，以此保证签名不同的包不被替换。

2、保证信息传输的完整性 签名对于包中的每个文件进行处理，以此确保包中内容不被替换。

3、防止交易中的抵赖发生， Market 对软件的要求。

54、为什么bindService可以跟Activity生命周期联动?

1、bindService 方法执行时，LoadedApk 会记录 ServiceConnection 信息。

2、Activity 执行 finish 方法时，会通过 LoadedApk 检查 Activity 是否存在未注销/解绑的 BroadcastReceiver 和 ServiceConnection，如果有，那么会通知 AMS 注销/解绑对应的 BroadcastReceiver 和 Service，并打印异常信息，告诉用户应该主动执行注销/解绑的操作。

55、如何通过Gradle配置多渠道包?

用于生成不同渠道的包

```
android {
    productFlavors {
        xiaomi {}
        baidu {}
        wandoujia {}
        _360 {}        // 或“"360"{}”，数字需下划线开头或加上双引号
    }
}
```

执行./gradlew assembleRelease ，将会打出所有渠道的release包;

执行./gradlew assembleWandoujia，将会打出豌豆荚渠道的release和debug版的包;

执行./gradlew assembleWandoujiaRelease将生成豌豆荚的release包。

因此，可以结合buildType和productFlavor生成不同的Build Variants，即类型与渠道不同的组合。

56、activty和Fragmengt之间怎么通信，Fragmengt和Fragmengt怎么通信?

（一）Handler

（二）广播

（三）事件总线：EventBus、RxBus、Otto

（四）接口回调

（五）Bundle和setArguments(bundle)

57、自定义view效率高于xml定义吗？说明理由。

自定义view效率高于xml定义：

1、少了解析xml。

2.、自定义View 减少了ViewGroup与View之间的测量,包括父量子,子量自身,子在父中位置摆放,当子view变化时,父的某些属性都会跟着变化。

58、广播注册一般有几种，各有什么优缺点?

第一种是常驻型(静态注册)：当应用程序关闭后如果有信息广播来，程序也会被系统调用，自己运行。

第二种不常驻(动态注册)：广播会跟随程序的生命周期。

动态注册

优点： 在android的广播机制中，动态注册优先级高于静态注册优先级，因此在必要情况下，是需要动态注册广播接收者的。

缺点： 当用来注册的 Activity 关掉后，广播也就失效了。

静态注册

优点： 无需担忧广播接收器是否被关闭，只要设备是开启状态，广播接收器就是打开着的。

59、服务启动一般有几种，服务和activty之间怎么通信，服务和服务之间怎么通信

方式：

1、startService：

onCreate()--->onStartCommand() ---> onDestory()

如果服务已经开启，不会重复的执行onCreate()， 而是会调用onStartCommand()。一旦服务开启跟调用者(开启者)就没有任何关系了。 开启者退出了，开启者挂了，服务还在后台长期的运行。 开启者不能调用服务里面的方法。

2、bindService：

onCreate() --->onBind()--->onunbind()--->onDestory()

bind的方式开启服务，绑定服务，调用者挂了，服务也会跟着挂掉。 绑定者可以调用服务里面的方法。

通信：

1、通过Binder对象。

2、通过broadcast(广播)。

60、ddms 和 traceView 的区别?

ddms 原意是：davik debug monitor service。简单的说 ddms 是一个程序执行查看器，在里面可以看见线程和堆栈等信息，traceView 是程序性能分析器。traceview 是 ddms 中的一部分内容。

Traceview 是 Android 平台特有的数据采集和分析工具，它主要用于分析 Android 中应用程序的 hotspot（瓶颈）。Traceview 本身只是一个数据分析工具，而数据的采集则需要使用 Android SDK 中的 Debug 类或者利用DDMS 工具。二者的用法如下：开发者在一些关键代码段开始前调用 Android SDK 中 Debug 类的 startMethodTracing 函数，并在关键代码段结束前调用 stopMethodTracing 函数。这两个函数运行过程中将采集运行时间内该应用所有线程（注意，只能是 Java线程） 的函数执行情况， 并将采集数据保存到/mnt/sdcard/下的一个文件中。 开发者然后需要利用 SDK 中的 Traceview工具来分析这些数据。

61、ListView卡顿原因

Adapter的getView方法里面convertView没有使用setTag和getTag方式；

在getView方法里面ViewHolder初始化后的赋值或者是多个控件的显示状态和背景的显示没有优化好，抑或是里面含有复杂的计算和耗时操作；

在getView方法里面 inflate的row 嵌套太深（布局过于复杂）或者是布局里面有大图片或者背景所致；

Adapter多余或者不合理的notifySetDataChanged；

listview 被多层嵌套，多次的onMessure导致卡顿，如果多层嵌套无法避免，建议把listview的高和宽设置为match_parent. 如果是代码继承的listview，那么也请你别忘记为你的继承类添加上LayoutPrams，注意高和宽都mactch_parent的；

62、AndroidManifest的作用与理解

AndroidManifest.xml文件，也叫清单文件，来获知应用中是否包含该组件，如果有会直接启动该组件。可以理解是一个应用的配置文件。

作用：

- 为应用的 Java 软件包命名。软件包名称充当应用的唯一标识符。
- 描述应用的各个组件，包括构成应用的 Activity、服务、广播接收器和内容提供程序。它还为实现每个组件的类命名并发布其功能，例如它们可以处理的 Intent - 消息。这些声明向 Android 系统告知有关组件以及可以启动这些组件的条件的信息。
- 确定托管应用组件的进程。
- 声明应用必须具备哪些权限才能访问 API 中受保护的部分并与其他应用交互。还声明其他应用与该应用组件交互所需具备的权限
- 列出 Instrumentation类，这些类可在应用运行时提供分析和其他信息。这些声明只会在应用处于开发阶段时出现在清单中，在应用发布之前将移除。
- 声明应用所需的最低 Android API 级别

- 列出应用必须链接到的库

63、LaunchMode应用场景

standard，创建一个新的Activity。

singleTop，栈顶不是该类型的Activity，创建一个新的Activity。否则，onNewIntent。

singleTask，回退栈中没有该类型的Activity，创建Activity，否则，onNewIntent+ClearTop。

注意:

设置了"singleTask"启动模式的Activity，它在启动的时候，会先在系统中查找属性值affinity等于它的属性值taskAffinity的Task存在；如果存在这样的Task，它就会在这个Task中启动，否则就会在新的任务栈中启动。因此， 如果我们想要设置了"singleTask"启动模式的Activity在新的任务中启动，就要为它设置一个独立的taskAffinity属性值。

如果设置了"singleTask"启动模式的Activity不是在新的任务中启动时，它会在已有的任务中查看是否已经存在相应的Activity实例， 如果存在，就会把位于这个Activity实例上面的Activity全部结束掉，即最终这个Activity 实例会位于任务的Stack顶端中。

在一个任务栈中只有一个”singleTask”启动模式的Activity存在。他的上面可以有其他的Activity。这点与singleInstance是有区别的。

singleInstance，回退栈中，只有这一个Activity，没有其他Activity。

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。

singleInstance应用场景：

闹铃的响铃界面。 你以前设置了一个闹铃：上午6点。在上午5点58分，你启动了闹铃设置界面，并按 Home 键回桌面；在上午5点59分时，你在微信和朋友聊天；在6点时，闹铃响了，并且弹出了一个对话框形式的 Activity(名为 AlarmAlertActivity) 提示你到6点了(这个 Activity 就是以 SingleInstance 加载模式打开的)，你按返回键，回到的是微信的聊天界面，这是因为 AlarmAlertActivity 所在的 Task 的栈只有他一个元素， 因此退出之后这个 Task 的栈空了。如果是以 SingleTask 打开 AlarmAlertActivity，那么当闹铃响了的时候，按返回键应该进入闹铃设置界面。

64、说说Activity、Intent、Service 是什么关系

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

65、ApplicationContext和ActivityContext的区别

这是两种不同的context，也是最常见的两种.第一种中context的生命周期与Application的生命周期相关的，context随着Application的销毁而销毁，伴随application的一生，与activity的生命周期无关.第二种中的context跟Activity的生命周期是相关的，但是对一个Application来说，Activity可以销毁几次，那么属于Activity的context就会销毁多次.至于用哪种context，得看应用场景。还有就是，在使用context的时候，小心内存泄露，防止内存泄露，注意一下几个方面：

- 不要让生命周期长的对象引用activity context，即保证引用activity的对象要与activity本身生命周期是一样的。
- 对于生命周期长的对象，可以使用application context。
- 避免非静态的内部类，尽量使用静态类，避免生命周期问题，注意内部类对外部对象引用导致的生命周期变化。

66、Handler、Thread和HandlerThread的差别

1、Handler：在android中负责发送和处理消息，通过它可以实现其他支线线程与主线程之间的消息通讯。

2、Thread：Java进程中执行运算的最小单位，亦即执行处理机调度的基本单位。某一进程中一路单独运行的程序。

3、HandlerThread：一个继承自Thread的类HandlerThread，Android中没有对Java中的Thread进行任何封装，而是提供了一个继承自Thread的类HandlerThread类，这个类对Java的Thread做了很多便利的封装。HandlerThread继承于Thread，所以它本质就是个Thread。与普通Thread的差别就在于，它在内部直接实现了Looper的实现，这是Handler消息机制必不可少的。有了自己的looper，可以让我们在自己的线程中分发和处理消息。如果不用HandlerThread的话，需要手动去调用Looper.prepare()和Looper.loop()这些方法。

67、ThreadLocal的原理

ThreadLocal是一个关于创建线程局部变量的类。使用场景如下所示：

- 实现单个线程单例以及单个线程上下文信息存储，比如交易id等。
- 实现线程安全，非线程安全的对象使用ThreadLocal之后就会变得线程安全，因为每个线程都会有一个对应的实例。 承载一些线程相关的数据，避免在方法中来回传递参数。

当需要使用多线程时，有个变量恰巧不需要共享，此时就不必使用synchronized这么麻烦的关键字来锁住，每个线程都相当于在堆内存中开辟一个空间，线程中带有对共享变量的缓冲区，通过缓冲区将堆内存中的共享变量进行读取和操作，ThreadLocal相当于线程内的内存，一个局部变量。每次可以对线程自身的数据读取和操作，并不需要通过缓冲区与 主内存中的变量进行交互。并不会像synchronized那样修改主内存的数据，再将主内存的数据复制到线程内的工作内存。ThreadLocal可以让线程独占资源，存储于线程内部，避免线程堵塞造成CPU吞吐下降。

在每个Thread中包含一个ThreadLocalMap，ThreadLocalMap的key是ThreadLocal的对象，value是独享数据。

68、计算一个view的嵌套层级

```
private int getParents(ViewParents view){
    if(view.getParents() == null)
        return 0;
    } else {
    return (1 + getParents(view.getParents));
   }
}
```

69、MVP，MVVM，MVC解释和实践

MVC:

- 视图层(View) 对应于xml布局文件和java代码动态view部分
- 控制层(Controller) MVC中Android的控制层是由Activity来承担的，Activity本来主要是作为初始化页面，展示数据的操作，但是因为XML视图功能太弱，所以Activity既要负责视图的显示又要加入控制逻辑，承担的功能过多。
- 模型层(Model) 针对业务模型，建立数据结构和相关的类，它主要负责网络请求，数据库处理，I/O的操作。

总结

具有一定的分层，model彻底解耦，controller和view并没有解耦 层与层之间的交互尽量使用回调或者去使用消息机制去完成，尽量避免直接持有 controller和view在android中无法做到彻底分离，但在代码逻辑层面一定要分清 业务逻辑被放置在model层，能够更好的复用和修改增加业务。

MVP

通过引入接口BaseView，让相应的视图组件如Activity，Fragment去实现BaseView，实现了视图层的独立，通过中间层Preseter实现了Model和View的完全解耦。MVP彻底解决了MVC中View和Controller傻傻分不清楚的问题，但是随着业务逻辑的增加，一个页面可能会非常复杂，UI的改变是非常多，会有非常多的case，这样就会造成View的接口会很庞大。

MVVM

MVP中我们说过随着业务逻辑的增加，UI的改变多的情况下，会有非常多的跟UI相关的case，这样就会造成View的接口会很庞大。而MVVM就解决了这个问题，通过双向绑定的机制，实现数据和UI内容，只要想改其中一方，另一方都能够及时更新的一种设计理念，这样就省去了很多在View层中写很多case的情况，只需要改变数据就行。

MVVM与DataBinding的关系?

MVVM是一种思想，DataBinding是谷歌推出的方便实现MVVM的工具。

看起来MVVM很好的解决了MVC和MVP的不足，但是由于数据和视图的双向绑定，导致出现问题时不太好定位来源，有可能数据问题导致，也有可能业务逻辑中对视图属性的修改导致。如果项目中打算用MVVM的话可以考虑使用官方的架构组件ViewModel、LiveData、DataBinding去实现MVVM。

三者如何选择?

- 如果项目简单，没什么复杂性，未来改动也不大的话，那就不要用设计模式或者架构方法，只需要将每个模块封装好，方便调用即可，不要为了使用设计模式或架构方法而使用。
- 对于偏向展示型的app，绝大多数业务逻辑都在后端，app主要功能就是展示数据，交互等，建议使用mvvm。
- 对于工具类或者需要写很多业务逻辑app，使用mvp或者mvvm都可。

70、SharedPrefrences的apply和commit有什么区别?

这两个方法的区别在于：

1. apply没有返回值而commit返回boolean表明修改是否提交成功。
2. apply是将修改数据原子提交到内存, 而后异步真正提交到硬件磁盘, 而commit是同步的提交到硬件磁盘，因此，在多个并发的提交commit的时候，他们会等待正在处理的commit保存到磁盘后在操作，从而降低了效率。而apply只是原子的提交到内容，后面有调用apply的函数的将会直接覆盖前面的内存数据，这样从一定程度上提高了很多效率。
3. apply方法不会提示任何失败的提示。 由于在一个进程中，sharedPreference是单实例，一般不会出现并发冲突，如果对提交的结果不关心的话，建议使用apply，当然需要确保提交成功且有后续操作的话，还是需要用commit的。

71、Base64、MD5是加密方法么?

Base64是什么?

Base64是用文本表示二进制的编码方式，它使用4个字节的文本来表示3个字节的原始二进制数据。 它将二进制数据转换成一个由64个可打印的字符组成的序列：A-Za-z0-9+/

MD5是什么?

MD5是哈希算法的一种，可以将任意数据产生出一个128位（16字节）的散列值，用于确保信息传输完整一致。我们常在注册登录模块使用MD5，用户密码可以使用MD5加密的方式进行存储。如：md5(hello world,32) = 5eb63bbbe01eeed093cb22bb8f5acdc3

加密，指的是对数据进行转换以后，数据变成了另一种格式，并且除了拿到解密方法的人，没人能把数据转换回来。 MD5是一种信息摘要算法，它是不可逆的，不可以解密。所以它只能算的上是一种单向加密算法。 Base64也不是加密算法，它是一种数据编码方式，虽然是可逆的，但是它的编码方式是公开的，无所谓加密。

72、HttpClient和HttpConnection的区别?

Http Client适用于web浏览器，拥有大量灵活的API，实现起来比较稳定，且其功能比较丰富，提供了很多工具，封装了http的请求头，参数，内容体，响应，还有一些高级功能，代理、COOKIE、鉴权、压缩、连接池的处理。 但是，正因此，在不破坏兼容性的前提下，其庞大的API也使人难以改进，因此Android团队对于修改优化Apache Http Client并不积极。(并在Android 6.0中抛弃了Http Client，替换成OkHttp)

HttpURLConnection对于大部分功能都进行了包装，Http Client的高级功能代码会较复杂，另外，HttpURLConnection在Android 2.3中增加了一些Https方面的改进(包括Http Client，两者都支持https)。且在Android 4.0中增加了response cache。当缓存被安装后(调用HttpResponseCache的install()方法)，所有的HTTP请求都会满足以下三种情况：

- 所有的缓存响应都由本地存储来提供。因为没有必要去发起任务的网络连接请求，所有的响应都可以立刻获取到。
- 视情况而定的缓存响应必须要有服务器来进行更新检查。比如说客户端发起了一条类似于 “如果/foo.png这张图片发生了改变，就将它发送给我” 这样的请求，服务器需要将更新后的数据进行返回，或者返回一个304 Not Modified状态。如果请求的内容没有发生，客户端就不会下载任何数据。
- 没有缓存的响应都是由服务器直接提供的。这部分响应会在稍后存储到响应缓存中。

在Android 2.2版本之前，HttpClient拥有较少的bug，因此使用它是最好的选择。 而在Android 2.3版本及以后，HttpURLConnection则是最佳的选择。它的API简单，体积较小，因而非常适用于Android项目。压缩和缓存机制可以有效地减少网络访问的流量，在提升速度和省电方面也起到了较大的作用。对于新的应用程序应该更加偏向于使用HttpURLConnection，因为在以后的工作当中Android官方也会将更多的时间放在优化HttpURLConnection上面。

73、ActivityA跳转ActivityB然后B按back返回A，各自的生命周期顺序，A与B均不透明。

ActivityA跳转到ActivityB：

```
Activity A: onPause
Activity B: onCreate
Activity B: onStart
Activity B: onResume
Activity A: onStop
```

ActivityB返回ActivityA：

```
Activity B: onPause
Activity A: onRestart
Activity A: onStart
Activity A: onResume
Activity B: onStop
Activity B: onDestroy
```

74、如何通过广播拦截和abort一条短信?

可以监听这条信号，在传递给真正的接收程序时，我们将自定义的广播接收程序的优先级大于它，并且取消广播的传播，这样就可以实现拦截短信的功能了。

75、BroadcastReceiver，LocalBroadcastReceiver 区别?

1、应用场景

1、BroadcastReceiver用于应用之间的传递消息；

2、而LocalBroadcastManager用于应用内部传递消息，比broadcastReceiver更加高效。

2、安全

1、BroadcastReceiver使用的Content API，所以本质上它是跨应用的，所以在使用它时必须要考虑到不要被别的应用滥用；

2、LocalBroadcastManager不需要考虑安全问题，因为它只在应用内部有效。

3、原理方面

(1) 与BroadcastReceiver是以 Binder 通讯方式为底层实现的机制不同，LocalBroadcastManager 的核心实现实际还是 Handler，只是利用到了 IntentFilter 的 match 功能，至于 BroadcastReceiver 换成其他接口也无所谓，顺便利用了现成的类和概念而已。

(2) LocalBroadcastManager因为是 Handler 实现的应用内的通信，自然安全性更好，效率更高。

76、如何选择第三方，从那些方面考虑?

大方向：从软件环境做判断

性能是开源软件第一解决的问题。

一个好的生态，是一个优秀的开源库必备的，取决标准就是观察它是否一直在持续更新迭代，是否能及时处理github上用户提出来的问题。大家在社区针对这个开源库是否有比较活跃的探讨。

背景，该开源库由谁推出，由哪个公司推出来的。

用户数和有哪些知名的企业落地使用

小方向：从软件开发者的角度做判断

是否解决了我们现有问题或长期来看带来的维护成本。

公司有多少人会。

学习成本。

77、简单说下接入支付的流程，是否自己接入过支付功能?

Alipay支付功能：

1.首先登录支付宝开放平台创建应用，并给应用添加App支付功能， 由于App支付功能需要签约，因此需要上传公司信息和证件等资料进行签约。

2.签约成功后，需要配置秘钥。使用支付宝提供的工具生成RSA公钥和私钥，公钥需要设置到管理后台。

3.android studio集成

```
（1）copy jar包；
（2）发起支付请求，处理支付请求。
```

78、单例实现线程的同步的要求：

1.单例类确保自己只有一个实例(构造函数私有:不被外部实例化,也不被继承)。

2.单例类必须自己创建自己的实例。

3.单例类必须为其他对象提供唯一的实例。

79、如何保证Service不被杀死?

Android 进程不死从3个层面入手：

A.提供进程优先级，降低进程被杀死的概率

方法一：监控手机锁屏解锁事件，在屏幕锁屏时启动1个像素的 Activity，在用户解锁时将 Activity 销毁掉。

方法二：启动前台service。

方法三：提升service优先级：

在AndroidManifest.xml文件中对于intent-filter可以通过android:priority = "1000"这个属性设置最高优先级，1000是最高值，如果数字越小则优先级越低，同时适用于广播。

B. 在进程被杀死后，进行拉活

方法一：注册高频率广播接收器，唤起进程。如网络变化，解锁屏幕，开机等

方法二：双进程相互唤起。

方法三：依靠系统唤起。

方法四：onDestroy方法里重启service：service + broadcast 方式，就是当service走ondestory的时候，发送一个自定义的广播，当收到广播的时候，重新启动service；

C. 依靠第三方

根据终端不同，在小米手机（包括 MIUI）接入小米推送、华为手机接入华为推送；其他手机可以考虑接入腾讯信鸽或极光推送与小米推送做 A/B Test。

80、说说ContentProvider、ContentResolver、ContentObserver 之间的关系?

ContentProvider：管理数据，提供数据的增删改查操作，数据源可以是数据库、文件、XML、网络等，ContentProvider为这些数据的访问提供了统一的接口，可以用来做进程间数据共享。

ContentResolver：ContentResolver可以为不同URI操作不同的ContentProvider中的数据，外部进程可以通过ContentResolver与ContentProvider进行交互。

ContentObserver：观察ContentProvider中的数据变化，并将变化通知给外界。

81、如何导入外部数据库?

把原数据库包括在项目源码的 res/raw。

android系统下数据库应该存放在 /data/data/com.（package name）/ 目录下，所以我们需要做的是把已有的数据库传入那个目录下。操作方法是用FileInputStream读取原数据库，再用FileOutputStream把读取到的东西写入到那个目录。

82、LinearLayout、FrameLayout、RelativeLayout性能对比，为什么?

RelativeLayout会让子View调用2次onMeasure，LinearLayout 在有weight时，也会调用子 View 2次onMeasure

RelativeLayout的子View如果高度和RelativeLayout不同，则会引发效率问题，当子View很复杂时，这个问题会更加严重。如果可以，尽量使用padding代替margin。

在不影响层级深度的情况下,使用LinearLayout和FrameLayout而不是RelativeLayout。

为什么Google给开发者默认新建了个RelativeLayout，而自己却在DecorView中用了个LinearLayout?

因为DecorView的层级深度是已知而且固定的，上面一个标题栏，下面一个内容栏。采用RelativeLayout并不会降低层级深度，所以此时在根节点上用LinearLayout是效率最高的。而之所以给开发者默认新建了个RelativeLayout是希望开发者能采用尽量少的View层级来表达布局以实现性能最优，因为复杂的View嵌套对性能的影响会更大一些。

83、scheme跳转协议

Android中的scheme是一种页面内跳转协议，通过定义自己的scheme协议，可以跳转到app中的各个页面

服务器可以定制化告诉app跳转哪个页面

App可以通过跳转到另一个App页面

可以通过H5页面跳转页面

84、HandlerThread

1、HandlerThread原理

当系统有多个耗时任务需要执行时，每个任务都会开启个新线程去执行耗时任务，这样会导致系统多次创建和销毁线程，从而影响性能。为了解决这一问题，Google提出了HandlerThread，HandlerThread本质上是一个线程类，它继承了Thread。HandlerThread有自己的内部Looper对象，可以进行loopr循环。通过获取HandlerThread的looper对象传递给Handler对象，可以在handleMessage()方法中执行异步任务。创建HandlerThread后必须先调用HandlerThread.start()方法，Thread会先调用run方法，创建Looper对象。当有耗时任务进入队列时，则不需要开启新线程，在原有的线程中执行耗时任务即可，否则线程阻塞。它在Android中的一个具体的使用场景是IntentService。由于HanlderThread的run()方法是一个无限循环，因此当明确不需要再使用HandlerThread时，可以通过它的quit或者quitSafely方法来终止线程的执行。

2、HanlderThread的优缺点

- HandlerThread优点是异步不会堵塞，减少对性能的消耗。
- HandlerThread缺点是不能同时继续进行多任务处理，要等待进行处理，处理效率较低。
- HandlerThread与线程池不同，HandlerThread是一个串队列，背后只有一个线程。

85、IntentService

IntentService是一种特殊的Service，它继承了Service并且它是一个抽象类，因此必须创建它的子类才能使用IntentService。

原理

在实现上，IntentService封装了HandlerThread和Handler。当IntentService被第一次启动时，它的onCreate()方法会被调用，onCreat()方法会创建一个HandlerThread，然后使用它的Looper来构造一个Handler对象mServiceHandler，这样通过mServiceHandler发送的消息最终都会在HandlerThread中执行。

生成一个默认的且与主线程互相独立的工作者线程来执行所有传送至onStartCommand()方法的Intetnt。

生成一个工作队列来传送Intent对象给onHandleIntent()方法，同一时刻只传送一个Intent对象，这样一来，你就不必担心多线程的问题。在所有的请求(Intent)都被执行完以后会自动停止服务，所以，你不需要自己去调用stopSelf()方法来停止。

该服务提供了一个onBind()方法的默认实现，它返回null。

提供了一个onStartCommand()方法的默认实现，它将Intent先传送至工作队列，然后从工作队列中每次取出一个传送至onHandleIntent()方法，在该方法中对Intent做相应的处理。

为什么在mServiceHandler的handleMessage()回调方法中执行完onHandlerIntent()方法后要使用带参数的stopSelf()方法?

因为stopSel()方法会立即停止服务，而stopSelf（int startId）会等待所有的消息都处理完毕后才终止服务，一般来说，stopSelf(int startId)在尝试停止服务之前会判断最近启动服务的次数是否和startId相等，如果相等就立刻停止服务，不相等则不停止服务。

86、如何将一个Activity设置成窗口的样式。

中配置：

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

另外

```
android:theme="@android:style/Theme.Translucnt"
```

是设置透明。

87、Android中跨进程通讯的几种方式

1：访问其他应用程序的Activity 如调用系统通话应用

```
Intent callIntent = new Intent(Intent.ACTION_CALL,Uri.parse("tel:12345678");
startActivity(callIntent);
```

2：Content Provider 如访问系统相册

3：广播（Broadcast） 如显示系统时间

4：AIDL服务

88、显示Intent与隐式Intent的区别

对明确指出了目标组件名称的Intent，我们称之为“显式Intent”。

对于没有明确指出目标组件名称的Intent，则称之为“隐式 Intent”。

对于隐式意图，在定义Activity时，指定一个intent-filter，当一个隐式意图对象被一个意图过滤器进行匹配时，将有三个方面会被参考到：

动作(Action)

类别(Category ['kætɪg(ə)rɪ])

数据(Data)

```
<activity android:name=".MainActivity"  android:label="@string/app_name">
            <intent-filter>
                <action android:name="com.wpc.test" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="image/gif"/>
            </intent-filter>
</activity>
```

89、Android Holo主题与MD主题的理念，以及你的看法

Holo Theme

Holo Theme 是 Android Design 的最基础的呈现方式。因为是最为基础的 Android Design 呈现形式，每一台 Android 4.X 的手机系统内部都有集成 Holo Theme 需要的控件，即开发者不需要自己设计控件，而是直接从系统里调用相应的控件。在 UI 方面没有任何的亮点，和 Android4.X 的设置/电话的视觉效果极度统一。由此带来的好处显而易见，这个应用作为 Android 应用的辨识度极高，且完全不可能与系统风格产生冲突。

Material Design

Material design其实是单纯的一种设计语言，它包含了系统界面风格、交互、UI,更加专注拟真,更加大胆丰富的用色,更加丰富的交互形式,更加灵活的布局形式

1.鲜明、形象的界面风格，

2.色彩搭配使得应用看起来非常的大胆、充满色彩感，凸显内容

3.Material design对于界面的排版非常的重视

4.Material design的交互设计上采用的是响应式交互，这样的交互设计能把一个应用从简单展现用户所请求的信息，提升至能与用户产生更强烈、更具体化交互的工具。

90、如何让程序自动启动?

定义一个Braodcastreceiver，action为BOOT——COMPLETE，接受到广播后启动程序。

91、Fragment 在 ViewPager 里面的生命周期，滑动 ViewPager 的页面时 Fragment 的生命周期的变化。

92、如何查看模拟器中的SP与SQList文件。如何可视化查看布局嵌套层数与加载时间。

93、各大平台打包上线的流程与审核时间，常见问题(主流的应用市场说出3-4个)

94、屏幕适配的处理技巧都有哪些?

一、为什么要适配

为了保证用户获得一致的用户体验效果,使得某一元素在Android不同尺寸、不同分辨率的、不同系统的手机上具备相同的显示效果，能够保持界面上的效果一致,我们需要对各种手机屏幕进行适配!

- Android系统碎片化：基于Google原生系统，小米定制的MIUI、魅族定制的flyme、华为定制的EMUI等等;
- Android机型屏幕尺寸碎片化：5寸、5.5寸、6寸等等;
- Android屏幕分辨率碎片化：320x480、480x800、720x1280、1080x1920等。

二、基本概念

- 像素（px）：像素就是手机屏幕的最小构成单元，px = 1像素点 一般情况下UI设计师的设计图会以px作为统一的计量单位。
- 分辨率：手机在横向、纵向上的像素点数总和 一般描述成 宽*高 ，即横向像素点个数 * 纵向像素点个数（如1080 x 1920），单位：px。
- 屏幕尺寸：手机对角线的物理尺寸。单位 英寸（inch），一英寸大约2.54cm 常见的尺寸有4.7寸、5寸、5.5寸、6寸。
- 屏幕像素密度（dpi）：每英寸的像素点数，例如每英寸内有160个像素点，则其像素密度为160dpi，单位：dpi（dots per inch）。
- 标准屏幕像素密度（mdpi）： 每英寸长度上还有160个像素点（160dpi），即称为标准屏幕像素密度（mdpi）。
- 密度无关像素（dp）：与终端上的实际物理像素点无关，可以保证在不同屏幕像素密度的设备上显示相同的效果，是安卓特有的长度单位，dp与px的转换：1dp = （dpi / 160 ） * 1px。

- 独立比例像素（sp）：字体大小专用单位 Android开发时用此单位设置文字大小，推荐使用12sp、14sp、18sp、22sp作为字体大小。

三、适配方案

适配的最多的3个分辨率：1280*720,1920*1080,800*480。

解决方案：

对于Android的屏幕适配，我认为可以从以下4个方面来做：

1、布局组件适配

- 请务必使用密度无关像素 dp 或独立比例像素 sp 单位指定尺寸。
- 使用相对布局或线性布局，不要使用绝对布局
- 使用wrap_content、match_parent、权重
- 使用minWidth、minHeight、lines等属性

dimens使用：

不同的屏幕尺寸可以定义不同的数值，或者是不同的语言显示我们也可以定义不同的数值，因为翻译后的长度一般都不会跟中文的一致。此外，也可以使用百分比布局或者AndroidStudio2.2的新特性约束布局。

2、布局适配

使用限定符（屏幕密度限定符、尺寸限定符、最小宽度限定符、布局别名、屏幕方向限定符)根据屏幕的配置来加载相应的UI布局。

3、图片资源适配

使用自动拉伸图.9png图片格式使图片资源自适应屏幕尺寸。

普通图片和图标：

建议按照官方的密度类型进行切图即可，但一般我们只需xxhdpi或xxxhdpi的切图即可满足我们的需求；

4、代码适配：

在代码中使用Google提供的API对设备的屏幕宽度进行测量，然后按照需求进行设置。

5、接口配合：

本地加载图片前判断手机分辨率或像素密度，向服务器请求对应级别图片。

95、动态布局的理解

96、怎么去除重复代码?

97、Recycleview和ListView的区别

98、动态权限适配方案，权限组的概念

99、Android系统为什么会设计ContentProvider?

100、下拉状态栏是不是影响activity的生命周期

101、如果在onStop的时候做了网络请求，onResume的时候怎么恢复?

102、Debug和Release状态的不同

103、dp是什么，sp呢，有什么区别

103、自定义View，ViewGroup注意那些回调?

104、android中的存储类型

105、Activity的生命周期，finish调用后其他生命周期还会走么?

106、有遇到过哪些屏幕和资源适配问题?

107、项目中遇到哪些难题，最终你是如何解决的?

108、listview图片加载错乱的原理和解决方案。

109、invalidate和requestLayout的区别及使用。

110、如何反编译，对代码逆向分析。

111、RemoteViews实现和使用场景

112、对服务器众多错误码的处理（错误码有好几万个）

113、adb常用命令行

114、Android中如何查看一个对象的回收情况?

115、Activity正常和异常情况下的生命周期

116、关于< include >< merge >< stub >三者的使用场景

117、Android对HashMap做了优化后推出的新的容器类是什么?

118、说下你对服务的理解，如何杀死一个服务。

119、断点续传实现?

在本地下载过程中要使用数据库实时存储到底存储到文件的哪个位置了，这样点击开始继续传递时，才能通过HTTP的GET请求中的setRequestProperty("Range","bytes=startIndex-endIndex");方法可以告诉服务器，数据从哪里开始，到哪里结束。同时在本地的文件写入时，RandomAccessFile的seek()方法也支持在文件中的任意位置进行写入操作。最后通过广播或事件总线机制将子线程的进度告诉Activity的进度条。关于断线续传的HTTP状态码是206，即HttpStatus.SC_PARTIAL_CONTENT。