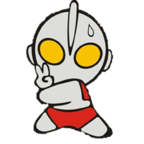
**慕课网【Android面试】听课笔记一**

[](http://www.jianshu.com/u/65a6ca5b9b1d)

作者 [Oterman](http://www.jianshu.com/u/65a6ca5b9b1d) 关注

2017.06.22 16:16 字数 5050 阅读 84评论 0喜欢 3

以下内容为慕课网【Android面试】课程的视频整理笔记，同时加上了一些自己的理解，整理较为粗糙，可以根据这个框架再去细致的了解。

**内容为一到四章。**

--

**一、课程简介**

* 基础知识
  + Fragment
  + Activity
  + Service
  + 广播
  + Binder
  + Webview安全漏洞
* 异步消息处理
  + Handler
  + IntentService
  + AsyncTask
  + HandlerThread
* View相关
  + View渲染和绘制
  + 事件分发
  + ListView优化
* Android构建
  + 编译打包
  + Gradle
  + Proguard混淆
  + 渠道包
  + Git
* 开源框架
  + 网络框架:Retrofit,Okhttp,Volley
  + 图片:Glide,Fresco,UIL
  + IOC:Dagger3 ButterKnife
* 热门前沿
  + 插件化
  + RxJava
  + 热更新
  + 进程保活

--

**二、 技术要求及面试前准备**

良好的开发习惯；独立思考的能力；主动并善于沟通；

**招聘要求例子**：  
1.常用设计模式及数据机构；(手写算法，加分点)  
2.Android开发经验；  
3.Java语言基础;（java的高级特性）  
4.网络编程，性能优化，内存优化；

**高级工程师要求**：  
1.基本知识点  
2.深入知识点  
3.基本知识点的细节  
4.系统核心机制

开口去说，事先预演，说的时候才知道问题所在，有条理。

**面试中要求:**  
1.礼貌,态度谦虚；  
2.听懂问题直接回答；

**感悟**  
基础知识，看简历面试，根据公司需求，看你牛不牛；

1.选一个擅长的领域，如网络框架；  
2.基础要背，灵活的背诵，背诵常用API；  
3.了解领域前沿技术；  
4.研究一个熟悉库的源码；

**三、基础相关面试题**

**3-1 Activity面试详解一**

**1.Activity的四种状态**

running,  
paused, 弹出对话框  
stopped,完全被覆盖  
killed：被回收掉了

**2. Activity生命周期**

启动：onCreate->onStart->onResume  
onStart：可见 ，但是不能交互；  
onResume:可见，可交互；

点击Home，返回主页面：onPause->onStop

onPause：可见，暂停  
onStop:被完全覆盖

点击重新回到app:onRestart->onStart->onResume  
退出：onPause->onStop->onDestroy

**3. 进程优先级**

参考文章：

[android 进程优先级](http://blog.csdn.net/fhy_2008/article/details/7328967)

**前台进程**：正在交互的进程，或者处于前台的service进程；  
包含以下组件的进程属于前台进程：

* 处于前台正与用户交互的Activity
* 与前台Acitity绑定的Service
* 调用了startForeground()的Service
* 正在执行onCreate(),onStart()或者onDestroy()方法的service
* 正在执行onReceive()的BroadcastReceiver

前台进程优先级很高，基本不会被杀死。

**可见进程**：处于可见，但是不是交互；  
以下进程属于可见进程：

* 进程中包含了出于paused状态的activity（弹出了对话框）
* 进程中包含了与可见进程绑定的Service

**服务进程**：包含了已启动的服务的进程

**后台进程**：前台进程按下home键切换到后台 内存不足时，容易被回收

服务进程的优先级高于后台进程，所以可以在服务中开启线程做一些耗时的操作。

**空进程**： 优先级最低，容易被回收，保存它的唯一目的是为了缓存一些启动数据，以便下次更快的启动应用

**3-2 Activity面试二**

**1. 任务栈**

任务栈和启动模式息息相关  
flag:

* FLAG\_ACTIVITY\_NEW\_TASK 对应android:launchMode="singleTask"模式，通常在service中启动一个acitivity时，需要加该flag，否则会报错；
* FLAG\_ACTIIVTY\_SINGLE\_TOP 对应android:launchMode="singleTop"
* FLAG\_ACTIVITY\_CLEAR\_TOP 与在清单文件指定android:launchMode="singleTask"效果相同

**2. 启动模式**

activity的复用

1**.standard**: 每次启动都会重新创建，不管是否存在；  
2.**singleTop**： 栈顶复用模式，在栈顶时，则不会创建，但是仅仅只是在栈顶；  
3.**singleTask**：栈内复用模式，单例，检查整个栈；onNewIntent；  
4.**singleInstance**:整个系统只有一个实例，打电话

**3.scheme跳转协议**

通过定义scheme语法实现跳转：

1. app内页面跳转;
2. 通过H5页面跳转到app;
3. 从一个app到另一个app;

**3-3 Fragment面试一**

参考文章：

[Android Fragment 真正的完全解析（上）](http://blog.csdn.net/lmj623565791/article/details/37970961)  
[Android Fragment 真正的完全解析（下）](http://blog.csdn.net/lmj623565791/article/details/37992017)

**1.为什么Fragment被称为第五大组件?**

大屏幕 展现ui,ui切换效果，动态灵活的加入到activity

**2.Fragment加载到Activity中的两种方式**

**静态加载**：布局文件中通过fragment标签引入

**动态加载**：  
FragmentManager 管理  
FragmentTransation 添加，替换

transaction=fm.benginTransaction();

transaction.add(); //添加到上面去，会重叠显示

transaction.replace();//替换 remove()和add的合体

transactioin.hide();//隐藏起来，不会重新绘制

transaction.remove(); //移除掉

transaction.commit();

**3.Fragment回退栈**

多个frament显示时，为了能点击back键时，能有像activity一样的效果，我们需要使用FragmentTransaction.addToBackStack（）方法；

**4.FragmentPagerAdapter与FragmentStatePagerAdapter区别**

ViewPager与Fragment结合使用；

前者适用页面较少: detach 并没有回收内存，ui分离  
后者使用页面较多: remove 回收内存

**3-4 Fragment面试二**

**1. Fragment生命周期**

不是独立的，需要结合Activity的生命周期；  
以下是启动到销毁的全过程：  
onAttach  
onCreate  
onCreateView  
onViewCreated  
Activity:onCreate  
onActivityCrearted  
Activity:onStart  
onStart  
Activity:onResume  
onResume  
初始化完成

onPause  
Activity:onPause  
onStop  
Activity:onStop  
onDestroyView  
onDestroy  
onDetach  
Activity:onDestroy

**2.Fragment通信**

Fragment中getActivity可以获取到依附的activity；

Activity中可以保存fragment的引用，也可以getFragmentManager().findFragmentById()获取到fragment;还可以使用接口回掉；

Fragment之间通信：以activity作为桥梁，先获取fragment，然后可以进行通信；

通信推荐使用接口回掉：Activity应该向总线一样，处理各种事情，fragment和fragment之间松耦合；

**3.重要方法**

FragmentManager  
replace 替换  
add 添加到Fragment队列，fragment视图，添加到容器中显示；

remove 从Fragment队列中删除,fragment视图会从容器中移除；

**3-5 Service面试一**

**1.Service是什么**

一个一种可以在后台执行长时间运行操作而没有用户界面的应用组件。  
四大组件之一 后台处理耗时东西 执行长时间的东西  
不用看到界面 应用组件 程序退出后仍能继续运行 保活  
运行主线程 和广播一样。Service中不能执行耗时操作

**2.与Thread的区别**

没有关系 翻译误解 后台混淆  
**定义**：  
Thread:程序执行的最小单元，分配CPU的基本单位  
Service:Android中的一种特殊机制，运行依托于所在的线程，他运行在主线程中，后台代表不依赖于UI,可以在其创建子线程执行耗时，在Activity中创建线程无法对现场进行控制；轻量级IPC，

**实际开发中**：  
Service不可以做耗时，ANR，没有UI，后台，

**应用场景**：  
线程：网络 新开现场  
服务：后台播放音乐，开启数据统计；

**3-6 Service面试二**

**1.两种启动方式**

使用步骤：

* 自己写个类继承Service
* 清单文件注册Service

两种启动方式：

**startService**（intent）：开启后，无限期运行，Activity销毁后也不会停止，手动停止stopService;

**bindService**(intent,conn,flag):交互，绑定到Activity，unBind解除  
创建服务端，实现Ibinder类,提供方法供外部调用；  
onBind返回Binder实例  
客户端中，onServiceConnecte中接收Binder实例  
onServiceDisconnected:意外中断时调用，意外！！并非主动取消；

**2.前台启动**

创建前台服务可以提高进程的优先级，从服务进程提升到前台进程；创建前台进程要与一个通知进行关联；

典型代码：

Notification.Builder builder = new Notification.Builder(this);

PendingIntent contentIntent = PendingIntent.getActivity(this, 0,

new Intent(this, MainActivity.class), 0);

builder.setContentIntent(contentIntent);

builder.setSmallIcon(R.drawable.icon);

builder.setTicker("Foreground Service Start");

builder.setContentTitle("Foreground Service");

builder.setContentText("Make this service run in the foreground.");

Notification notification = builder.build();

startForegroundCompat(NOTIFICATION\_ID, notification);

以上代码位于Service **的onCreate方法中**；

**3.生命周期**

onBind 绑定时调用  
onCreate 创建时调用  
onStartCommand 返回int:START\_STICKY，内存不足或其他意外情况被销毁时，系统会重建并保证重新调用onStartCommand这个方法

onDestroy 资源清理

**3-7 BroadCast面试相关**

**1.广播**

定义 观察者模式思维，是一种广泛运用在应用程序之间的传输信息的机制，发送的广播是一个Intent，这个Intent可以携带我们想要的数据。不同程序之间应用程序之间数据共享。

场景  
同一个app内多进程不同组件进行通信；  
不同app之间进行通信；

种类

[说说Android的广播(1) - 普通广播,有序广播和粘性广播](https://yq.aliyun.com/articles/53919)

**普通广播**：传给各个符合接受条件的receiver，各个receiver的接收顺序不被保证，广播不能被截获；

**有序广播**：按照优先级高低依次传给receiver,优先级高的可以截获广播，使得优先级低的无法收到广播；  
**粘性广播**：api 23标记已过时，它不安全，没有保护；发送粘性广播需要额外的权限，发送粘性广播后，广播会保留在系统中，不管对应的广播接收者有没有创建好，都会收到广播；即粘性广播会等待它的接收者准备好后，再发给它的接收者；

**2.注册广播接收者**

自己实现广播接收者：

public class xxReceiver extends BroadcastReceiver{

public void onReceiver(Context context,Intent intent){

//do something

}

}

1.**静态注册**：在配置文件中进行注册，activity销毁后，仍然能够销毁；不够灵活；

<receiver android:name="xxReceiver">

<intent-filter>

<action android:name="xxx.aa.bb"/>

</inten-filter>

</receiver>

2.**动态注册**：代码中调用，registReceiver,跟随Activity的生命周期,activity销毁后，广播会销毁；动态注册后，需要调用unregistReceiver来销毁广播；

IntentFilter filter=new IntentFilter();

filter.addAction("xxx.aa.bb");

registReceiver(new xxReceiver(),filter);

3.**发送自定义广播**

Intent intent=new Intent();

intent.setAction("xxx.aa.bb");

sendBroadcast(intent);

**3.内部实现机制**

[BroadcastReceiver的工作过程](http://blog.csdn.net/jly0612/article/details/51258621)

**注册广播**：通过Binder机制，以跨进程方式向AMS进行注册（ActivityManagerService）,AMS负责广播注册的真正实现，它会保存远程的Receiver对象IntentFilter对象;

**发送广播**：广播的发送也是交给AMS来处理的，5.0之后，广播默认不会发给未启动的应用,AMS接收到广播之后，它会根据过滤规则把符合条件的广播接收者保存到BroadcastQueue队列中，然后通过Handler来发送消息，消息循环拿到广播，进行处理，回掉广播接收者的onReceiver方法；

**4.LocalBroadcastManager**

[Android 中LocalBroadcastManager的使用方式](https://my.oschina.net/ososchina/blog/340339)

看源码  
1.发送的广播只在app内传播，数据不用被泄露；  
2.其他app无法对你app发送广播，接受不到别的app的广播，不会有安全漏洞可以利用；  
3.比系统的全局广播更加有效；

内部通过主线程的Hanlder来实现，sendBroadcast是通过handler发送message,与通常意义上的发送广播不一样；  
通过handler发送，高效，同时也保证了在应用内使用，其他app无法利用漏洞；

**3-8 webview安全漏洞面试相关**

[android面试热点：webview使用及与js数据交互](http://blog.csdn.net/jerrywu145/article/details/57512366)

**1.常见的坑**

1.level16 之前，webview.addJavascriptInterface,远程攻击者利用发射机制，执行任意java对象的方法；

2.布局文件中的使用：写在其他的容器中；销毁时，先remove，然后销毁；

3.jsbridge:通过Js来构建桥，远程调用js

4.webviewClient.onPageFinished ->webchromeclient.onProgresschanged。 跳转url时，不断调用，用后一个替换；

5.后台耗电

6.硬件加速，页面渲染问题； 设置关闭硬件加速；

**2. 内存泄漏问题**

原因：会新开线程，与匿名内部内持有外部内引用导致无法回收类似；

1.独立进程，简单暴力，设计进程通信；  
2.动态添加webview，对传入的webview的所需的context传入弱引用，动态将其放在某个viewgroup中，销毁时，先从容器移除，然后调用自身方法来销毁；

**3-9 Binder面试相关**

难点！！

**1.Linux内核基础知识**

1.进程隔离 /虚拟地址空间 避免进程a操作进程b的数据  
2.系统调用  
3.binder驱动

**2.通信机制**

1.为什么要使用binder  
传统有管道，socket等跨进程的通信机制；  
性能  
安全

2.通信模型

通信录：binder驱动（电话基站） serviceManager

3.什么是Binder

通常意义，跨进程通信机制  
对Server来说，只Binder本地对象，对client说，代理对象；  
传输过程，跨进程传递对象

**3.AIDL**

自动生成

asInterface方法 统一进程 、不会走跨进程；不是同一进程，代理；

--

**四、异步消息处理机制相关面试问题**

**4-1 Handler问题讲解一**

**自己理解**：  
Handler是Android提供的消息处理机制，主要用来处理子线程中更新UI问题，Handler背后有  
Looper,MessageQueue,Message这几个类，他们是一个整体，通过相互协作，共同完成子线程访问UI这一目标；

Handler：主要负责发送消息以及处理消息，发送消息时，会将消息加入到MessageQueue中；处理消息，则是在handleMessage方法中进行；  
MessageQueue:主要包含对消息对象的插入和读取；  
Looper:消息循环器，它不停的从MessageQueue中获取消息，获取不到消息就会等待，获取到消息后就会分发给发送消息的那个handler去处理消息；

在创建handler对象时，需要关联一个looper对象，该looper对象是从当前线程进行获取，当前线程是通过threadlocal保存looper对象，threadlocal会为每一个线程都保存一份单独的变量，调用Looper.prepare()方法即可在当前现场创建looper对象并将其保存在当前线程中，looper对象创建时同时也会创建一个MessageQueue队列与之相关联；在任何线程都可以构建一个Handler，Looper,MessageQueue消息处理循环系统；

通常我们在UI线程即Activity中创建handler对象，looper对象已经由ui线程在初始化时初始完毕了，在主线程中，我们不需要关心looper的创建及looper.loop()消息循环的开启，因为主线程已经帮我们做好了这些事情。

**1.什么是handler**

主线程更新ui，子线程发送消息更新ui.消息机制的上层接口。  
handler通过发送和处理Message和Runnable对象来关联对应线程的MessageQueue。  
可以让对应的Message及Runnable在未来的某个时间点进行处理；  
让耗时操作在子线程，更新ui操作放在主线程。

**2.使用方法**

post(runnable)  
主线程创建handler成员变量；hanlder会关联创建他的线程；

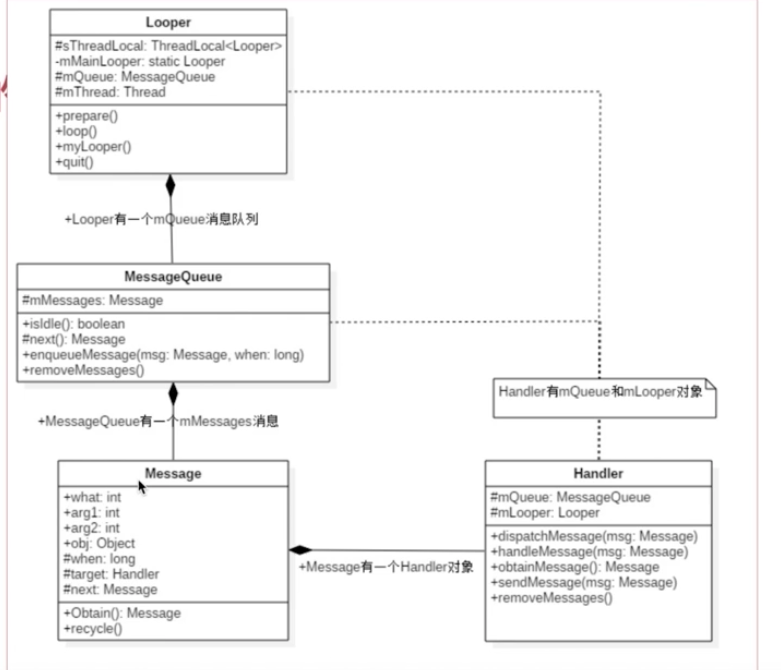
sendMessage(message)  
复写handleMessage方法；  
message=Message.obtain();  
msg.what=xxx;  
msg.arg1=  
msg.arg2=

handler.sendMessage(msg);

**4-2 Handler问题讲解二**

**1.机制原理**

看源码



handler.PNG

Looper  
MessageQueue  
ThreadLocal  
Handler  
Message

**2.内存泄漏及解决方法**

[Android使用Handler造成内存泄露的分析及解决方法](http://www.cnblogs.com/xujian2014/p/5025650.html)

原因： 内部类会持有外部类的匿名引用，当外部的activity需要回收时，此时handler可能在做一些耗时操作，由于持有了外部类引用，就会导致外部的activity无法回收，从而导致内存泄漏；

解决方法：handler内部持有外部Activity的弱引用；把handler改为静态内部类；mhandler.removeCallback();

**4-3 AsyncTask相关问题**

**1.什么是AsyncTask**

AsyncTask是一个轻量级的异步任务类，它可以在线程池中执行后台耗时任务，然后把执行的结果以及执行的进度传递给主线程并在主线程中更新UI；内部封装了线程池和Handler,它可以让我们避免直接使用Thread及Handler；

**2.使用方法**

三个参数，五个方法  
第一个：输入  
第二个：进度  
第三个：结果

onPreExecute();  
doInBackground();  
onProgressUpdated();  
onPostExecuted();  
publishProgress();

典型用法：

private class DownloadTask extends AsyncTask<URL,Integer,Long>{

/\*\*

\* 主线程中执行，在执行异步任务之前调用

\*/

@Override

protected void onPreExecute() {

super.onPreExecute();

}

/\*\*

\* 线程池中执行

\*/

@Override

protected Long doInBackground(URL... params) {

int count=params.length;

long totalSize=0;

for(int i=0;i<count;i++){

totalSize+=Downloader.download(params[i]);

publishProgress(i/count\*100);//发布进度更新

if(isCancelled()){

break;

}

}

return totalSize;

}

@Override

protected void onPostExecute(Long result) {

showDialog("下载了："+result.intValue()+"字节");

}

@Override

protected void onProgressUpdate(Integer... values) {

super.onProgressUpdate(values);

//更新显示进度

}

}

创建异步任务并执行：

new DownloadTask().execute(url1,url2);

**3.机制原理**

源码  
封装了线程池及interalHandler，利用线程池来执行后台任务，线程池的工作线程执行doInBackground方法执行异步任务,在执行任务的过程中及执行完毕后，通过内部的handler发送消息，handler处理消息时回掉相应的函数。

**4.注意事项**

内存泄漏  
非静态内部类持有外部内的引用，致使无法回收，引起泄露，与handler泄露一样；

生命周期  
不会随着activity的销毁而销毁，必须主动调用cancle来取消异步任务,而且在activity结束时，必须在onDestroy中取消任务；

结果丢失  
旋转重建，asynctask持有之前activity的引用，然而当异步任务处理完毕想要更新ui时，却无法更新；

并行or串行  
2.3之后串行，默认是串行，可以执行并行，executeOnExecutors();无法做高并发操作；

**4-4 HandlerThread**

**1.是什么**

背景 开启子线程进行耗时操作，耗费资源，多次创建和销毁线程耗费资源；  
hanlder thread looper  
thread内部有一个looper

特点

本质线程类，集成thread  
内部有looper对象，run方法中开启了loop循环  
通过获取looper对象传递给Handler对象，可以在handleMessage中执行异步任务；

优点是不会有堵塞，减少了性能的消耗，缺点是不能同时执行多任务的处理，需要进行等待处理，处理的效率比较低；

和线程池并发不同，它是串行队列，它背后只有一个线程；它的run方法是一个无限循环，明确不需要使用时，需要调用quit或quitSafely退出；

典型应用是IntentService;

**2.源码分析**

**4-5 IntentService**

**1.是什么**

它是执行并处理异步请求的一个特殊的service,由于它是一个Service，它的优先级比普通的后台线程要高，不易被系统杀死，它适合执行一些优先级较高的后台任务。

内部封装了handlerthread和handler，利用handlerThread处理耗时操作，启动方式和传统的service启动方式一样，同时，任务执行完毕后，它会自动的停止，不需要我们手动调用stopserivice方法。

它可以启动多次，每一个耗时的操作都会以工作队列的方式在IntentService得onHandleIntent回掉方法中执行，并且，每次只会执行一个工作线程，执行完毕后才会执行第二个。它是串行的。

**2.使用方法**

自定义一个类继承IntentService,实现onHandlerIntent()方法，该方法是在子线程中执行，所以可以进行一些耗时的操作；

class MyIntentService extends IntentService{

public MyIntentService(String name) {

super(name);

}

/\*\*

\* 工作线程中执行

\*/

@Override

protected void onHandleIntent(@Nullable Intent intent) {

String task\_action = intent.getStringExtra("task\_action");

SystemClock.sleep(3000);//模拟耗时任务

Log.d("MyIntentService", "handle task:" + task\_action);

}

}

启动方式和传统一样：

Intent service=new Intent(this,MyIntentService.class);

service.putExtra("task\_action","task1");

startService(service);

service.putExtra("task\_action","task2");

startActivity(service);

**3.源码分析**

封装了HandlerThread和Hanler的异步框架；

handler发送消息，处理消息时回掉onHandleIntent（）方法；创建hanlder需要提供looper,looper来源于HandlerThread,在HandlerThread这个子线程构建了一个消息循环系统，handler处理消息的线程就是创建handler的线程，即handlerThread这个线程；

参见《Android开发艺术探索》 403页。