目　录

Android 工程师之 Android 面试大纲

1. Activity 面试题

（1）Activity 是什么

Activity 是四大组件之一，它提供一个界面让用户点击和各种滑动操作，这就是 Activity

（2）Activity 四种状态

- runing
- paused
- stopped
- killed

（3）Activity 生命周期

- onCreate()
- onStart()
- onResume()
- onPause()
- onStop()
- onDestroy()
- onRestart()

（4）进程的优先级

- 空进程
- 后台进程
- 服务进程
- 可见进程
- 前台进程

（5）Activity 任务栈

- 先进后出

（6）Activity 启动模式

- standard

- singletop
- singletask
- singleinstance

（7）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

（8）Context、Activity、Application 之间有什么区别

Activity 和 Application 都是 Context 的子类。Context 从字面上理解就是上下文的意思，在实际应用中它也确实是起到了管理上下文环境中各个参数和变量的总用，方便我们可以简单的访问到各种资源。虽然 Activity 和 Application 都是 Context 的子类，但是他们维护的生命周期不一样。前者维护一个 Activity 的生命周期，所以其对应的 Context 也只能访问该 activity 内的各种资源。后者则是维护一个 Application 的生命周期

2. Fragment 面试题

（1）Fragment 为什么被称为第五大组件

Fragment 比 Activity 更节省内存，其切换模式也更加舒适，使用频率不低于四大组件，且有自己的生命周期，而且必须依附于 Activity

（2）Activity 创建 Fragment 的方式

- 静态创建
- 动态创建

（3）FragmentPageAdapter 和 FragmentPageStateAdapter 的区别

- FragmentPageAdapter 在每次切换页面的的时候，是将 Fragment 进行分离，适合页面较少的 Fragment 使用以保存一些内存，对系统内存不会多大影响
- FragmentPageStateAdapter 在每次切换页面的时候，是将 Fragment 进行回收，适合页面较多的 Fragment 使用，这样就不会消耗更多的内存

（4）Fragment 生命周期

- onAttach()
- onCreate()
- onCreateView()
- onActivityCreated()
- onStart()
- onResume()
- onPause()
- onStop()
- onDestroyView()
- onDestroy()
- onDetach()

（5）Fragment 的通信

- Fragment 调用 Activity 中的方法：getActivity
- Activity 调用 Fragment 中的方法：接口回调
- Fragment 调用 Fragment 中的方法：FragmentManager.findFragmentById

（6）Fragment 的 replace、add、remove 方法

- replace：替代 Fragment 的栈顶页面
- add：添加 Fragment 到栈顶页面
- remove：移除 Fragment 栈顶页面

3. Service 面试题

（1）Service 是什么

Service 是四大组件之一，它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。

（2）Service 和 Thread 的区别

- Service 是安卓中系统的组件，它运行在独立进程的主线程中，不可以执行耗时操作。Thread 是程序执行的最小单元，分配 CPU 的基本单位，可以开启子线程执行耗时操作

- Service 在不同 Activity 中可以获取自身实例，可以方便的对 Service 进行操作。Thread 在不同的 Activity 中难以获取自身实例，如果 Activity 被销毁，Thread 实例就很难再获取得到

（3）Service 启动方式

- startService
- bindService

（4）Service 生命周期

startService:

- onCreate()
- onStartCommand()
- onDestroy()

bindService:

- onCreate()
- onBind()
- onUnbind()
- onDestroy()

4. Broadcast Receiver 面试题

（1）Broadcast Receiver 是什么

Broadcast 是四大组件之一，是一种广泛运用在应用程序之间传输信息的机制，通过发送 Intent 来传送我们的数据。

（2）Broadcast Receiver 的使用场景

- 同一 App 具有多个进程的不同组件之间的消息通信
- 不同 App 之间的组件之间的消息通信

（3）Broadcast Receiver 的种类

- 普通广播
- 有序广播
- 本地广播
- Sticky 广播

（4）Broadcast Receiver 的实现

- 静态注册：注册后一直运行，尽管 Activity、进程、App 被杀死还是可以接收到广播
- 动态注册：跟随 Activity 的生命周期

（5）Broadcast Receiver 实现机制

- 自定义广播类继承 BroadcastReceiver，复写 onReceiver()
- 通过 Binder 机制向 AMS 进行注册广播
- 广播发送者通过 Binder 机制向 AMS 发送广播
- AMS 查找符合相应条件的广播发送到 BroadcastReceiver 相应的循环队列中
- 消息队列执行拿到广播，回调 BroadcastReceiver 的 onReceiver()

（6）LocalBroadcastManager 特点

- 本地广播只能在自身 App 内传播，不必担心泄漏隐私数据
- 本地广播不允许其他 App 对你的 App 发送该广播，不必担心安全漏洞被利用
- 本地广播比全局广播更高效
- 以上三点都是源于其内部是用 Handler 实现的

5. WebView 面试题

（1）WebView 安全漏洞

API16 之前存在远程代码执行安全漏洞，该漏洞源于程序没有正确限制使用 WebView.addJavascriptInterface 方法，远程攻击者可通过使用 Java 反射机制利用该漏洞执行任意 Java 对象的方法

（2）WebView 销毁步骤

WebView 在其他容器上时（如：LinearLayout），当销毁 Activity 时，需要在 onDestroy()中先移除容器上的 WebView，然后再将 WebView.destroy()，这样就不会导致内存泄漏

（3）WebView 的 jsbridge

客户端和服务端之间可以通过 Javascript 来互相调用各自的方法

（4）WebViewClient 的 onPageFinished

WebViewClient 的 onPageFinished 在每次完成页面的时候调用，但是遇到未加载完成的页面跳转其他页面时，就会一直调用，使用 WebChromeClient.onProgressChanged 可以替代。

（5）WebView 后台耗电

在 WebView 加载页面的时候，会自动开启线程去加载，如果不很好的关闭这些线程，就会导致电量消耗加大，可以采用暴力的方法，直接在 onDestroy 方法中 System.exit(0)结束当前正在运行中的 java 虚拟机

（6）WebView 硬件加速

Android3.0 引入硬件加速，默认会开启，WebView 在硬件加速的情况下滑动更加平滑，性能更加好，但是会出现白块或者页面闪烁的副作用，建议 WebView 暂时关闭硬件加速

（7）WebView 内存泄漏

由于 WebView 是依附于 Activity 的，Activity 的生命周期和 WebView 启动的线程的生命周期是不一致的，这会导致 WebView 一直持有对这个 Activity 的引用而无法释放，解决方案如下：

- 独立进程，简单暴力，不过可能涉及到进程间通信（推荐）
- 动态添加 WebView，对传入 WebView 中使用的 Context 使用弱引用

6. Binder 面试题

（1）Linux 内核的基本知识

- 进程隔离/虚拟地址空间：进程间是不可以共享数据的，相当于被隔离，每个进程被分配到不同的虚拟地址中
- 系统调用：Linux 内核对应用有访问权限，用户只能在应用层通过系统调用，调用内核的某些程序
- binder 驱动：它负责各个用户的进程，通过 binder 通信内核来进行交互的模块

（2）为什么使用 Binder

- 性能上，相比传统的 Socket 更加高效
- 安全性高，支持协议双方互相校验

（3）Binder 通信模型

- ervice 服务端通过 Binder 驱动在 ServiceManager 的查找表中注册 Object 对象的 add 方法
- Client 客户端通过 Binder 驱动在 ServiceManager 的查找表中找到 Object 对象的 add 方法，并返回 proxy 的 add 方法，add 方法是个空实现，proxy 也不是真正的 Object 对象，是通过 Binder 驱动封装好的代理类的 add 方法
- 当 Client 客户端调用 add 方法时，Client 客户端通过 Binder 驱动将 proxy 的 add 方法，请求 ServiceManager 来找到 Service 服务端真正对象的 add 方法，进行调用

（4）AIDL

- 客户端通过 aidl 文件的 Stub.asInterface()方法，拿到 Proxy 代理类
- 通过调用 Proxy 代理类的方法，将参数进行封包后，调用底层的 transact()方法
- transact()方法会回调 onTransact()方法，进行参数的解封
- 在 onTransact()方法中调用服务端对应的方法，并将结果返回

7. Handler 面试题

（1）Handler 是什么

Handler 通过发送和处理 Message 和 Runnable 对象来关联相对应线程的 MessageQueue

（2）Handler 使用方法

post(runnable)

sendMessage(message)

（3）Handler 工作原理

（4）Handler 引起的内存泄漏

原因：非静态内部类持有外部类的匿名引用，导致 Activity 无法释放

解决： Handler 内部持有外部 Activity 的弱引用

Handler 改为静态内部类

Handler.removeCallback()

8. AsyncTask 面试题

（1）AsyncTask 是什么

它本质上就是一个封装了线程池和 Handler 的异步框架

（2）AsyncTask 使用方法

三个参数

- Params：表示后台任务执行时的参数类型，该参数会传给 AysncTask 的 doInBackground()方法
- Progress：表示后台任务的执行进度的参数类型，该参数会作为 onProgressUpdate()方法的参数
- Result：表示后台任务的返回结果的参数类型，该参数会作为 onPostExecute()方法的参数

五个方法

- onPreExecute()：异步任务开启之前回调，在主线程中执行
- doInBackground()：执行异步任务，在线程池中执行
- onProgressUpdate()：当 doInBackground 中调用 publishProgress 时回调，在主线程中执行
- onPostExecute()：在异步任务执行之后回调，在主线程中执行
- onCancelled()：在异步任务被取消时回调

（3）AsyncTask 工作原理

（4）AsyncTask 引起的内存泄漏

- 原因：非静态内部类持有外部类的匿名引用，导致 Activity 无法释放
- 解决：AsyncTask 内部持有外部 Activity 的弱引用
- AsyncTask 改为静态内部类
- AsyncTask.cancel()

（5）AsyncTask 生命周期

在 Activity 销毁之前，取消 AsyncTask 的运行，以此来保证程序的稳定

（6）AsyncTask 结果丢失

由于屏幕旋转、Activity 在内存紧张时被回收等情况下，Activity 会被重新创建，此时，旧的 AsyncTask 持有旧的 Activity 引用，这个时候会导致 AsyncTask 的 onPostExecute()对 UI 更新无效

（7）AsyncTask 并行 or 串行

- AsyncTask 在 Android 2.3 之前默认采用并行执行任务，AsyncTask 在 Android 2.3 之后默认采用串行执行任务
- 如果需要在 Android 2.3 之后采用并行执行任务，可以调用 AsyncTask 的 executeOnExecutor()

9. HandlerThread 面试题

（1）HandlerThread 产生背景

当系统有多个耗时任务需要执行时，每个任务都会开启一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 提供了 HandlerThread，HandlerThread 是在线程中创建一个 Looper 循环器，让 Looper 轮询消息队列，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

（2）HanlderThread 的特点、

- HandlerThread 本质上是一个线程，继承自 Thread
- HandlerThread 有自己的 Looper 对象，可以进行 Looper 循环，可以创建 Handler
- HandlerThread 可以在 Handler 的 handlerMessage 中执行异步方法
- 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，需要等待进行处理，处理效率较低
- HandlerThread 与线程池不同，HandlerThread 是一个串行队列，背后只有一个线程

10. IntentService 面试题

（1）IntentService 是什么

IntentService 是继承自 Service 并处理异步请求的一个类，其内部采用 HandlerThread 和 Handler 实现的，在 IntentService 内有一个工作线程来处理耗时操作，其优先级比普通 Service 高。当任务完成后，IntentService 会自动停止，而不需要手动调用 stopSelf()。另外，可以多次启动 IntentService，每个耗时操作都会以工作队列的方式在 IntentService 中 onHandlerIntent()回调方法中执行，并且每次只会执行一个工作线程

（2）IntentService 使用方法

- 创建 Service 继承自 IntentService
- 覆写构造方法和 onHandlerIntent()方法
- 在 onHandlerIntent()中执行耗时操作

11. Android 项目构建面试题

（1）git 常用命令

- git init：仓库的初始化
- git status：查看当前仓库的状态
- git diff：查看仓库与上次修改的内容
- git add：将文件放进暂存区
- git commit：提交代码
- git clone：克隆代码
- git bransh：查看当前分支
- git checkout：切换当前分支

（2）git 工作流

- fork/clone（主流）
  - fork：将别人的仓库代码 fork 到自己的仓库上
  - clone：克隆下自己仓库的代码
  - update、commit：修改代码并提交到自己的仓库
  - push：提交到自己的仓库
  - pull request：请求添加到别人的仓库
- clone

（3）proguard 是什么

ProGuard 工具是用于压缩、优化和混淆我们的代码，其主作用是移除或混淆代码中无用类、字段、方法和属性

（4）proguard 技术功能

- 压缩
- 优化
- 混淆
- 预检测

（5）proguard 工作原理

将无用的字段或方法存入到 EntryPoint 中，将非 EntryPoint 的字段和方法进行替换。

（6）为什么要混淆

由于 Java 是一门跨平台的解释性语言，其源代码被编译成 class 字节码来适应其他平台，而 class 文件包含了 Java 源代码信息，很容易被反编译。

12. ANR 面试题

（1）什么是 ANR

Application Not Responding，页面无响应的对话框

（2）发生 ANR 的条件

应用程序的响应性是由 ActivityManager 和 WindowManager 系统服务监视的，当 ANR 发生条件满足时，就会弹出 ANR 的对话框

- Activity 超过 5 秒无响应
- BroadcastReceiver 超过 10 秒无响应
- Service 超过 20 秒无响应

（3）造成 ANR 的主要原因

主线程被 IO 操作阻塞

- Activity 的所有生命周期回调都是执行在主线程的
- Service 默认执行在主线程中
- BoardcastReceiver 的回调 onReceive()执行在主线程中
- AsyncTask 的回调除了 doInBackground，其他都是在主线程中

- 没有使用子线程 Looper 的 Handler 的 handlerMessage，

post(Runnable)都是执行在主线程中

（4）如何解决 ANR

- 使用 AsyncTask 处理耗时 IO 操作
- 使用 Thread 或 HandlerThread 提高优先级
- 使用 Handler 处理工作线程的耗时操作
- Activity 的 onCreate 和 onResume 回调尽量避免耗时操作

13. OOM 面试题

（1）什么是 OOM

OOM 指 Out of memory（内存溢出），当前占用内存加上我们申请的内存资源超过了 Dalvik 虚拟机的最大内存限制就会抛出 Out of memory 异常

（2）OOM 相关概念

- 内存溢出：指程序在申请内存时，没有足够的空间供其使用
- 内存泄漏：指程序分配出去的内存不再使用，无法进行回收
- 内存抖动：指程序短时间内大量创建对象，然后回收的现象

（3）解决 OOM

Bitmap 相关

- 图片压缩
- 加载缩略图
- 在滚动时不加载图片
- 回收 Bitmap
- 使用 inBitmap 属性
- 捕获异常

其他相关

- listview 重用 convertView、使用 lru
- 避免 onDraw 方法执行对象的创建
- 谨慎使用多进程

14. Bitmap 面试题

（1）recycle

- 在安卓 3.0 以前 Bitmap 是存放在堆中的，我们只要回收堆内存即可
- 在安卓 3.0 以后 Bitmap 是存放在内存中的，我们需要回收 native 层和 Java 层的内存
- 官方建议我们 3.0 以后使用 recycle 方法进行回收，该方法也可以不主动调用，因为垃圾回收器会自动收集不可用的 Bitmap 对象进行回收
- recycle 方法会判断 Bitmap 在不可用的情况下，将发送指令到垃圾回收器，让其回收 native 层和 Java 层的内存，则 Bitmap 进入 dead 状态
- recycle 方法是不可逆的，如果再次调用 getPixels()等方法，则获取不到想要的结果

（2）LruCache 原理

LruCache 是个泛型类，内部采用 LinkedHashMap 来实现缓存机制，它提供 get 方法和 put 方法来获取缓存和添加缓存，其最重要的方法 trimToSize 是用来移除最少使用的缓存和使用最久的缓存，并添加最新的缓存到队列中

15. 内存泄漏面试题

（1）Java 内存泄漏引起的主要原因

长生命周期的对象持有短生命周期对象的引用就很可能发生内存泄漏

（2）Java 内存分配策略

- 静态存储区：又称方法区，主要存储全局变量和静态变量，在整个程序运行期间都存在
- 栈区：方法体的局部变量会在栈区创建空间，并在方法执行结束后会自动释放变量的空间和内存
- 堆区：保存动态产生的数据，如：new 出来的对象和数组，在不使用的时候由 Java 回收器自动回收

（3）Android 解决内存泄漏的例子

- 单例造成的内存泄漏：在单例中，使用 context.getApplicationContext()作为单例的 context

- 匿名内部类造成的内存泄漏：由于非静态内部类持有匿名外部类的引用，必须将内部类设置为 static
- Handler 造成的内存泄漏：使用 static 的 Handler 内部类，同时在实现内部类中持有 Context 的弱引用
- 避免使用 static 变量：由于 static 变量会跟 Activity 生命周期一致，当 Activity 退出后台被后台回收时，static 变量是不安全，所以也要管理好 static 变量的生命周期
- 资源未关闭造成的内存泄漏：比如 Socket、Broadcast、Cursor、Bitmap、ListView 等，使用完后要关闭
- AsyncTask 造成的内存泄漏：由于非静态内部类持有匿名内部类的引用而造成内存泄漏，可以通过 AsyncTask 内部持有外部 Activity 的弱引用同时改为静态内部类或在 onDestroy()中执行 AsyncTask.cancel()进行修复

16. 内存管理面试题

（1）Android 内存管理机制

- 分配机制
- 管理机制

（2）内存管理机制的特点

- 更少的占用内存
- 在合适的时候，合理的释放系统资源
- 在系统内存紧张的时候，能释放掉大部分不重要的资源
- 能合理的在特殊生命周期中，保存或还原重要数据

（3）内存优化方法

- Service 完成任务后应停止它，或用 IntentService（因为可以自动停止服务）代替 Service
- 在 UI 不可见的时候，释放其 UI 资源
- 在系统内存紧张的时候，尽可能多的释放非重要资源
- 避免滥用 Bitmap 导致内存浪费
- 避免使用依赖注入框架
- 使用针对内存优化过的数据容器

- 使用 ZIP 对齐的 APK
- 使用多进程

17. 冷启动和热启动面试题

（1）什么是冷启动和热启动

- 冷启动：在启动应用前，系统中没有该应用的任何进程信息
- 热启动：在启动应用时，在已有的进程上启动应用（用户使用返回键退出应用，然后马上又重新启动应用）

（2）冷启动和热启动的区别

- 冷启动：创建 Application 后再创建和初始化 MainActivity
- 热启动：创建和初始化 MainActivity 即可

（3）冷启动时间的计算

这个时间值从应用启动（创建进程）开始计算，到完成视图的第一次绘制为止

（4）冷启动流程

- Zygote 进程中 fork 创建出一个新的进程
- 创建和初始化 Application 类、创建 MainActivity
- inflate 布局、当 onCreate/onStart/onResume 方法都走完
- contentView 的 measure/layout/draw 显示在界面上

总结:Application 构造方法->attachBaseContext()->onCreate()->Activity 构造方法->onCreate()->配置主题中背景等属性->onStart()->onResume()->测量布局绘制显示在界面上

（5）冷启动优化

- 减少第一个界面 onCreate()方法的工作量
- 不要让 Application 参与业务的操作
- 不要在 Application 进行耗时操作
- 不要以静态变量的方式在 Application 中保存数据
- 减少布局的复杂性和深度
- 不要在 mainThread 中加载资源
- 通过懒加载方式初始化第三方 SDK

18. 其他优化面试题

（1）Android 不用静态变量存储数据

- 静态变量等数据由于进程已经被杀死而被初始化
- 使用其他数据传输方式：文件/sp/contentProvider

（2）SharePreference 安全问题

- 不能跨进程同步
- 文件不宜过大

（3）内存对象序列化

- Serializeble：是 java 的序列化方式，Serializeble 在序列化的时候会产生大量的临时对象，从而引起频繁的 GC
- Parcelable：是 Android 的序列化方式，且性能比 Serializeble 高，Parcelable 不能使用在要将数据存储在硬盘上的情况

（4）避免在 UI 线程中做繁重的操作