阐述一下 Activity 的生命周期。参考链接

1、Activity 生命周期的顺序为：

onCreate() -> onContentChanged() -> onStart() -> onPostCreate()-> onResume() -> onPostResume -> onPause() -> onStop() -> onDestroy()

- onCreate() 方法：首次创建 Activity 时调用，应该在此方法中执行所有正常的静态设置 — 创建视图、将数据绑定到列表等等。
- onRestart() 方法：在 Activity 已停止（调用 onStop()）并即将再次启动前调用，始终后接 onStart()
- onStart() 方法：在 Activity 即将对用户可见之前调用，可后接 onStop()进入隐藏状态
- onResume() 方法：在 Activity 即将开始与用户进行交互之前调用，始终后接 onPause()
- onPause() 方法：当系统即将开始继续另一个 Activity 时调用。用于确认对持久性数据的未保存更改、停止动画以及其他可能消耗 CPU 的内容。应该非常迅速地执行所需操作，因为它返回后，下一个 Activity 才能继续执行。
- onStop() 方法：在 Activity 对用户不再可见时调用。可后接 onRestart()。
- onDestroy() 方法：在 Activity 被销毁前调用。

2、方法调用后进程是否可终止问题：

- onCreate(),onRestart(),onStart(),onResume() 调用后，系统不能随时终止 Activity 进程。
- 而 onPause(),onStop(),onDestroy() 调用后，可以终止。
- Activity 创建后，必须要走到 onPause() 方法，然后才能终止进程 — 如果系统在紧急情况下必须恢复内存，则可能不会调用 onStop() 和 onDestroy()。
- 所以，在 onPause() 调用时，向存储设备写入至关重要的持久性数据（例如用户编辑）。

3、其他几个关键点

- onPause 必须先执行完，新 Activity 的 onResume 才会执行
- 新 Activity 采用透明主题或为 dialog 样式，当前 Activity 不会回调 onStop
- 异常终止问题
    - Activity 异常终止时，会调用 onSaveInstanceState 来保存当前 Activity 状态，调用时机在 onStop 之前，和 onPause 没有固定顺序
    - onRestoreInstanceState 方法调用时机在 onStart 之后
    - onRestoreInstanceState 一旦被调用，其参数 Bundle savedInstanceState 一定是有值的，不用判断非空；而 onCreate 的 Bundle savedInstanceState 在正常启动时为 null

谈谈 Android 的四大组件。参考链接1、参考链接2

四大组件：Activity、Service、BroadcastReceiver、ContentProvider

- Activity：
    - 一个 Activity 通常就是一个单独的屏幕，它上面可以显示一些控件也可以监听并处理用户的事件做出响应
    - Activity 之间通过 Intent 进行通信，应用内使用显式 Intent，跨应用使用隐式 Intent
    - 在隐式 Intent 中，有三个最重要的部分：动作（Action）、类别（category）以及动作对应的数据（data）
    - Action、category 和 Uri 分别对应着 intent-filter 中的 action、category 和 data

- Service：
  - Service 是实现程序后台运行的解决方案，非常适合去执行不需要和用户交互（即没有界面）并且还要求长期执行的任务
  - 通过 Context 的 startService 和 stopService 可以启动、停止服务，启动服务后启动者和服务就没有关系了，启动者结束生命周期，服务依然在
  - 通过 Context 的 bindService 和 unBindService 可以绑定、解绑服务，绑定服务后服务会返回给启动者一个 IBind 接口实例，启动者可以通过这个接口实例回调服务的方法。此时 Context 退出，服务也会跟着解绑退出
- BroadcastReceiver：
  - 应用可以使用广播接收器对感兴趣的外部事件或内部事件进行注册监听，可以静态注册、动态注册
  - 广播接收器没有用户界面。但它们可以启动一个 Activity 或 Service 来响应它们收到的信息，或者用 NotificationManager 来通知用户
  - 广播分为普通广播、有序广播、粘性广播。注册有序广播接收器可以设定接收优先级按序接收或拦截；发送粘性广播后注册广播接收器也可以接收到该粘性广播
- ContentProvider：可参考《第二行代码》Page 270
  - 内容提供器用于提供在不同程序之间实现数据共享的功能，允许一个程序访问另一个程序中的数据，同时还能保证被访数据的安全性
  - 每个 ContentProvider 都用一个 Uri 作为独立的标识
  - 访问其他程序中的数据
    - 可以通过 Context 的 getContentResolver 方法获取到 ContentResolver 的实例，通过该实例的一系列方法对目标数据进行 CRUD 操作
    - 通过内容 Uri 表明想要访问的目标程序
  - 创建自己的内容提供器
    - 继承 ContentProvider，实现抽象方法
    - 通过 UriMacther 类的 match 方法判断其他程序想要访问的数据地址

Service 与 IntentService 的区别。Link

- 相同点：
  - 都是在后台运行，不可见
  - 都可在任何线程、系统组件进行开启
- 不同点：
  - Service 默认运行在主线程，不可直接做耗时操作；
  - IntentService 默认运行在子线程，可直接执行耗时操作
  - Service 不会自动退出，需要手动 stopSelf、stopService、unbindService 或者内存不足被动回收；
  - IntentService 的 onHandleIntent 方法执行完毕后会自动退出。当有多次请求（即多次调用 startService）时，因只有一个工作线程，所以会一个一个进行处理，处理完毕后退出

Android 应用中如何保存数据。

分为保存键值集（SharedPreferences）、保存文件（分为内部存储和外部存储）、在 SQL 数据库中保存数据以及在网络中使用网络存储和 ContentProvider。

其中：

- 键值集适合保存私有原始轻量化数据
- 内部存储适合保存私有数据

- 外部存储适合保存公共数据
- 私有数据库中适合保存结构化数据
- 网络存储需使用自己的网络服务器存储数据

如何在 Android 应用中执行耗时操作。

将耗时操作放到非 UI 线程执行，常用的包括：AsyncTask、Handler 配合 Thread 等。

两个 Fragment 之间如何通信。参考链接

1、 拿到另一个 Fragment 的实例

- 通过宿主 Activity 拿到 FragmentManager，进而再 findFragmentById 获取到另一个 Fragment，调用其公开方法
- 使用 Fragment 的 setArguments 获取到另一个 Fragment 的实例

2、通过 Activity 来操作

- 使用接口：在一个 Fragment 中定义接口，并在 onAttach 方法中拿到 Activity 初始化接口对象。Activity 来实现该接口，并在实现中调用另一个 Fragment 的相应方法
- 使用 Activity 的公共方法：直接 getActivity 并调用 Activity 的公共方法

3、使用广播

- 使用广播
- 使用 EventBus

Activity 之间如何通信，Activity 与 Service 之间如何通信

Activity 之间通信方式

1、基于消息的通讯机制：Intent。缺点：只能传递基本数据类型或序列化数据，不可序列化的数据无法传递

2、广播

3、EventBus

4、数据存储（File、SharedPreferences、SQLite、ContentProvider）

5、利用 static 静态数据

Activity 与 Service 之间

1、Binder 对象

2、广播

3、EventBus

4、Intent

两个不同的 app 之间如何交互。

- Activity（Intent）
- Broadcast
- ContentProvider
- Service（AIDL）跨进程通信
- 利用 ShareUserId 共享数据：两个应用的 ShareUserId 相同，则共享对方 data 目录下的文件

什么是 Fragment?

- 碎片是一种可以嵌入在活动中的 UI 片段
- 可以将多个碎片组合在一个 Activity 中，也可以在多个 Activity 中重复使用一个碎片
- 可以在 Activity 运行时动态添加、删除碎片
- 可以把碎片当作 Activity 的模块化组成部分，具有自己的生命周期，能接收自己的输入事件

为什么建议只使用默认的构造方法来创建 Fragment? Link

- 应当使用默认构造方法创建 Fragment，然后使用 setArguments 添加 Bundle 来传递参数
- 因为系统在储存 Fragment 的状态时（比如说屏幕旋转导致的重新加载），系统会为我们自动存储 Bundle

为什么 Bundle 被用来传递数据，为什么不能使用简单的 Map 数据结构? 链接1，链接2，链接3

原因一：速度和内存占用问题

- 使用 Bundle 传递数据一般用于启动 Activity，启动 Fragment 时，数据量一般都不大
- Bundle 内部由 ArrayMap 实现。ArrayMap 的内部实现是两个数组，一个数组存放 key 的 hashCode 值，一个数组存放 key 与 value 值。在查找、添加、删除数据时，会根据二分查找法查找 key 的 hashCode 值所对应的 index 值，再根据 index 去查找 key 所对应的 value 值
- HashMap 的实现则是采用数组加链表的结构，默认使用一个容量为 16 的数组来存储，数组中每一个元素又是一个链表的头节点
- 所以在小数据量的情况下，ArrayMap（Bundle） 比 HashMap 在速度和内存上都更占优势

原因二：序列化问题

- Android 中使用 Intent 来传递数据时，需要保证数据是基本数据类型或可序列化类型
- Bundle 使用 Parcelable 序列化，HashMap 使用 Serialable 序列化
- 在 Android 中，更推荐使用 Parcelable 序列化，因为 Serializable 接口使用了反射机制，这个过程相对缓慢，而且往往会产生出很多临时对象，可能会触发垃圾回收器频繁地进行垃圾回收。相比而言，Parcelable 接口比 Serializable 接口效率更高，性能方面要高出 10x 多倍。

阐述一下 Fragment 的生命周期。参考链接

- Fragment 的生命周期与 Activity 的生命周期协调一致
- Activity 的每次生命周期回调都会引发每个片段的类似回调
- 片段还有几个额外的生命周期回调，用于处理与 Activity 的唯一交互，以执行构建和销毁片段 UI 等操作，包括：
    - onAttach：在片段已与 Activity 关联时调用
    - onCreateView：在这可创建片段的视图层次结构
    - onActivityCreated：在 Activity 的 onCreate() 方法已返回时调用
    - onDestroyView：在移除与片段关联的视图层次结构时调用
    - onDetach：在取消片段与 Activity 的关联时调用

下图为 Activity 的每个连续状态如何决定片段可以收到的回调方法

解释下 Android 的 View 。

- View 是 UI 组件最基本的构建块
- 一个 View 占据屏幕的一块区域，负责绘制和事件处理

Activity、Window、View 关系参考

- 每个 Activity 包含了唯一的一个 PhoneWindow（Window 的子类），在 attach 方法里进行初始化
- Activity 实现了 Window 的接口，Window 接收到外界的改变时就会回调 Activity 相应的方法
- View 是 Android 中的视图单元，必须依附于 Window 而存在
- View 和 Window 之间通过 ViewRootImpl 进行关联到一起，ViewRootImpl 可以看作 ViewTree 的管理者（Activity 被创建完成后，会将 DecorView 添加到 Window 中，同时创建 ViewRootImpl 并与 DecorView 关联）

你能创建自定义 View 吗？具体是如何创建的？工作原理？参考链接

- 自定义 View 包括自定义 View 和自定义 ViewGroup
- 自定义 View 一般继承自 View、SurfaceView 或其他 View，不包含子 View
- 自定义 ViewGroup 是利用现有的组件，根据特定的布局方式，来组成新的组件。一般继承自 ViewGroup 或各种 Layout，包含有子 View
- 自定义 View 的流程：
    - 构造函数：初始化一些内容，获取自定义属性
    - 测量 View 的大小（onMeasure）：View 大小不仅由自身决定，也会受父控件的影响，我们自己进行测量可以适应各种情况
    - 确定 View 的大小（onSizeChanged）：在视图大小发生改变时调用
    - 确定子 View 布局（onLayout）：在自定义 ViewGroup 时用到，用于确定子 View 的位置。循环取出子 View，计算出各个子 View 的位置，调用子 View 的 layout 方法设置其位置
    - 绘制内容（onDraw）
    - 对外提供操作方法和监听回调：控制 View 的状态，监听 View 的变化等

注：onMeasure 中 View 的测量模式与 View 绘制流程中的测量时使用的测量模式一样，可参见View 三大流程源码分析

什么是 ViewGroup，它与 View 的区别在哪里?

- ViewGroup 是一个放置 View 的容器，继承自 View，是一种特殊的 View
- ViewGroup 的职能为：给 childView 计算出建议的宽、高和测量模式；决定 childView 的位置；
- View 的职能为：根据测量模式和 ViewGroup 建议的宽高计算出自己的宽和高；在 ViewGroup 为其指定的区域内绘制自己的形态

Fragment 和 Activity 有什么区别？它们之间又有什么关系?

- Fragment 在 Android 3.0 后引入，可以翻译为碎片、片段
- Activity 为活动，代表着当前屏幕显示的一个窗口
- Fragment 可以重用，必须嵌套在 Activity 中使用，由 Activity 的 FragmentManager 管理，生命周期受 Activity 影响

谈谈 Serializable 接口和 Parcelable 接口的区别。在 Android 中最好使用哪种接口?

Activity 的启动模式有哪些？参考链接

- standard 模式：创建新实例，放入当前任务中
- singleTop （栈顶复用）模式：
  - 当前任务中已存在，且在栈顶，不创建
  - 当前任务中已存在，不在栈顶，创建新实例
  - 当前任务中不存在，创建新实例
- singleTask （栈内复用）模式：
  - 首先检测要启动 Activity 所需任务栈是否存在（根据 affinity 判断）
  - 不存在：创建新任务栈，创建新实例
  - 存在：如果为后台任务栈就切换为前台任务栈 -> 有实例则 clearTop，没有实例就创建新实例
- SingleInstance 模式：
  - 启动一个新的任务，创建新实例
  - 并且新的任务中不会再添加其他 Activity

Activity 启动模式的应用场景?

解释一下 Android 中的 Intent 。Link

1、Intent 是一个消息传递对象，可以启动 Activity、Service，发送广播，同时携带必要的数据。

2、分为显式 Intent 和隐式 Intent，显式 Intent 按名称指定要启动的组件，隐式 Intent 则是声明要执行的操作，允许其他的应用组件来处理。

3、Intent 的实现是基于 Binder 机制；Intent 可以通过 Bundle 携带可序列化的对象，Bundle 使用 Parcelable 进行序列化数据。

什么是隐式 Intent ?

- 不指定特定的组件，而是声明要执行的常规操作（Action），从而允许其他应用中的组件来处理它
- 如声明发送功能，那么拥有发送 Action 的组件都可进行响应

什么是显式 Intent ?

- 按名称指定要启动的组件，完全限定类名
- 通常会在自己的应用中使用显式 Intent 来启动组件，因为知道想要启动的 Activity 或服务的类名

解释一下 AsyncTask。参考链接

- AsyncTask 是 Android 提供的一个助手类，对 Thread 和 Handler 进行了封装，方便我们在后台线程执行耗时操作，在主线程更新 UI 等
- AsyncTask 有四个回调方法：onPreExecute、doInBackground、onProgressUpdate、onPostExecute。
  - onPreExecute 在执行后台操作之前调用，运行在主线程
  - doInBackground 是必须实现的一个核心方法，可以执行后台耗时操作，运行在自子线程
  - onProgressUpdate 在 doInBackground 方法中调用publishProgress 方法时会进行回调，可以进行后台操作状态的展示更新，运行在主线程
  - onPostExecute 在后台操作完成后调用，运行在主线程

AsyncTask 的原理?

1、首先调用 AsyncTask 的构造方法，构造时对 Handler、WorkerRunnable（Callable） 和 FutureTask 进行初始化

2、然后调用 AsyncTask 的 execute 方法（可以手动设置 Executor，不设置则使用系统默认的 SerialExecutor）

3、首先判断当前 AsyncTask 状态，正在运行或者已经运行过就退出

4、调用 onPreExecute 执行准备工作

5、由 Executor 调用 FutureTask 的 run 方法，在 WorkerRunnable 中执行了 doInBackground

6、依旧是在 WorkerRunnable 中，调用 postResult，将执行结果通过 Handler 发送给主线程；调用 publishProgress 时，也是通过 Handler 将消息发送到主线程的消息队列中

如何理解 Android 中的广播。Link

- 广播接收器是一种用于响应系统范围广播通知的组件
- 广播接收器不会显示用户界面，但可以创建状态栏通知，在发生广播事件时提醒用户
- 广播接收器更常见的用途是作为通向其他组件的"通道"，比如开启一个服务，发起一个通知等，设计用于执行极少量的工作

如何理解 Android 的 LocalBroadcastManager 。参考链接

- LocalBroadcastManager 为本地广播
- 使用这个机制发出的广播仅在应用内部传递
- 广播接收器也只能接收本应用发出的广播
- 本地广播比系统全局广播更加安全、有效

广播和 EventBus 的区别参考1、参考2

1、全局广播是重量级别的，消耗资源比较多，优点是可以跨进程；

2、本地广播会比全局广播轻量一些，不能跨进程

3、EventBus 不能跨进程，比广播更轻量

4、广播基于 Android 提供的 Binder 机制，而 EventBus 基于反射，但二者都符合观察者模式，观察者和观察目标低耦合

补充，广播的大概工作机制如下：

- 广播接收者 BroadcastReceiver 通过 Binder 机制向 AMS(Activity Manager Service)进行注册；
- 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（IntentFilter/Permission 等）的 BroadcastReceiver，将广播发送到 BroadcastReceiver（一般情况下是Activity）相应的消息循环队列中；
- 消息循环执行拿到此广播，回调 BroadcastReceiver 中的 onReceive()方法。

什么是 JobScheduler？参考链接

- 当一定的条件（网络、时间、电量等）被满足时，JobScheduler 会为应用执行某项任务
- 与 AlarmManager 不同，JobScheduler 的执行时间是不确定的
- JobScheduler 允许同时执行多个任务

什么是 DDMS？你可以用它来做什么？参考链接

- DDMS 是 Android 应用程序调试和分析工具，可以截屏、查看进程、线程、堆栈信息、Logcat、广播状态信息、模拟呼叫、短信、位置等
- 但是现在已经不推荐使用
- Android Studio 3.0 以后推荐使用 Android Profile 来分析网络、内存、cpu 使用状态
- 推荐使用 ADB 来发送命令、传文件、截图等

解释一下什么是 support libary，以及为什么要引入 support library？ Link

Android 支持库提供了许多没有内置到框架中的功能。

1、向下兼容。保证使用了新系统功能的 App 可以兼容旧的系统，保证了高版本 SDK 开发的向下兼容。

2、提供了布局模式、界面元素，可以避免再做一些重复性的工作

3、支持不同形态的设备，可以通过支持库为手机、电视、手表等提供功能

如何理解 Android 中的 ContentProvider 。它通常用来干什么? 参考链接

- 内容提供器管理一组共享的应用数据
- 其他应用可以通过我们应用程序的内容提供程序查询、甚至修改我们应用的数据
- 我们也可以通过其他应用程序的内容提供器查询、修改该应用的数据
- 也适用于读取和写入应用不共享的私有数据
- 定义内容提供器需要继承自 ContentProvider，并实现相应的方法

什么是 Data Binding？ Link、参考链接2

数据绑定指：将数据源和 UI 进行绑定，并使其同步

在 Android 中：

- 使用 Android 官方提供的 Data binding 库，将 xml 布局和数据源进行绑定
- UI 布局接收用户事件并同步更新数据源，修改数据源也可同步展示到 UI 上

数据绑定和 MVVM 的区别：

- MVVM 是一种架构模式
- Data Binding 是一种实现数据和 UI 绑定的框架
- Data Binding 是构建 MVVM 模式的一个工具

Android 的核心组件具体都有什么?

- Activity
- Service
- BroadcastReceiver
- ContentProvider

什么是 ADB？ 参考链接

ADB 是 Android Debug Bridge 的简称，即 Android 调试桥。是一个通用命令行工具，允许与模拟器或连接的 Android 设备进行通信。包括三个组件：

- 客户端：运行在开发计算机上。用来发送命令，可以通过 adb 命令在命令行终端调用到客户端
- 服务器：以后台进程形式运行在开发计算机上。用来管理客户端（开发计算机）和后台程序（Android 终端）之间的通信
- 后台程序（adbd）：以后台进程形式运行在模拟器或 Android 设备上。在设备上运行命令

什么是 ANR？ 如何避免发生 ANR？ 参考链接1、参考链接2、参考链接3

- ANR 是 Application Not Responding，即应用程序未响应
- 以下情况会引起 ANR：
  - 主线程被 IO 操作堵塞
  - 主线程存在耗时计算
  - 主线程存在错误操作，如 Thread.sleep 或 Thread.wait 等
- 以下情况会弹出 ANR 对话框
  - 应用在 5 秒内未响应用户的输入事件（触摸、按键）
  - Activity、Application 回调方法超时时间：5 秒
  - BroadcastReceiver 回调方法超时时间：10 秒（前台 App 10 秒，后台 App 60 秒）
  - Service 回调方法超时时间：20 秒
- 如何避免发生 ANR：

  归为一句话：将 IO 操作放在工作线程来处理，并减少其他耗时操作和错误操作
  - 使用 AsyncTask 处理耗时 IO 操作
  - 使用 Thread 或 HandlerThread 时，要调用 Process.setThreadPriority(PROCESS_THREAD_PRIORITY_BACKGROUND)设置线程优先级，降低与主线程竞争的能力，因为默认 Thread 优先级和主线程相同
  - 使用 Handler 处理工作线程结果，不要使用 Thread.sleep，Thread.wait
  - Activity 的 onCreate 和 onResume 回调中，避免耗时代码
  - BroadcastReceiver 的 onReceive 方法也应避免耗时代码，可以开启 IntentService 代替
  - 可以通过查看 `/data/anr/traces.txt` 文件定位 ANR 情况

AndroidManifest.xml 是什么？[参考链接]

- AndroidManifest.xml 是应用清单，用来向 Android 系统提供必要的应用信息，系统必须得有这些信息才可以运行应用的代码
- 具体功能如下：
  - 为应用的 Java 软件包命名，软件包名称充当应用的唯一标志符
  - 描述四大组件，以及它们可以处理的 Intent 信息，启动这些组件的条件信息等
  - 确定托管应用组件的进程
  - 声明权限，权限是用于限制对部分代码或设备上数据的访问，为了保护可能被误用以致破坏或损害用户体验的关键数据和代码
  - （声明应用所需的最低 Android API 级别）
  - （列出应用必须链接到的库）

解释一下 broadcast 和 intent 在 app 内传递消息的工作流程。

Bitmap 如何优化，三级缓存的思想与逻辑

优化策略：

- 对图片质量进行压缩，通过 Bitmap.compress()
- 对图片尺寸进行压缩，通过设置 BitmapFactory.Options.inSampleSize 值
- 对图片进行复用：内存缓存、磁盘缓存、网络获取
- 及时释放 Bitmap 内存
- 捕获 OutOfMemory 异常
- 将图片放置于合适的 drawable 文件夹下

三级缓存：

- 最近最少使用原则
- 缓存策略包括缓存的添加、获取和删除操作

- LruCache 内部采用一个 LinkedHashMap 以强引用的方式来存储缓存对象，提供了 get、put、remove 方法来操作缓存对象
- DiskLruCache 使用 open 方法来创建，创建时指定缓存存储路径。缓存添加通过 Editor 来完成，通过图片 url 的 key 可以获取到相应的 Editor 对象，从而获取到文件输出流写入到文件系统。DiskLruCache 提供了 get 方法可以得到一个 Snapshot 对象，通过 Snapshot 对象可以得到缓存的文件输入流从而拿到缓存对象

补充：

知名的图片加载框架有 Glide、Picasso、Fresco 等

一个优秀的图片加载框架应该有如下功能：

- 图片的同步加载、异步加载（向调用者提供所加载的图片）
- 图片压缩
- 内存缓存、磁盘缓存、网络拉取

Android 应用有哪些不同的存储数据的方式?

- 键值集（SharedPreferences）
- SQL 数据库
- 文件
- 网络存储

什么是 Dalvik 虚拟机?

- Dalvik 虚拟机是 Google 设计用于 Android 平台的虚拟机
- 支持已转换为 .dex 格式的 Java 程序的运行，.dex 格式是专门为 Dalvik 虚拟机设计的一种压缩格式
- 允许在有限内存中运行多个虚拟机实例，并且每一个都作为独立的 Linux 进程运行

Dalvik 与 JVM 的关系：

- Dalvik 基于寄存器，JVM 基于堆栈
- Dalvik 有自己的字节码，并不使用 Java 字节码
- 从 Android 2.2 开始，Dalvik 开始支持即时编译
- Dalvik 会通过 Zygote 进行类的预加载和资源的预加载，完成虚拟机的初始化

Dalvik 虚拟机模式和 ART（Android Runtime）虚拟机模式的区别。即时编译、ART

- Dalvik 采用 JIT 即时编译技术，ART 采用 Ahead-of-time AOT 预编译技术
- Dalvik 虚拟机在应用程序启动时，JIT 通过进行连续的性能分析来优化程序代码的执行，在程序运行的过程中，Dalvik 虚拟机在不断的将字节码编译成机器码
- ART 虚拟机在应用程序安装的过程中，ART 就已经将所有的字节码重新编译成了机器码。应用程序运行过程中无需进行实时的编译工作，只需要进行直接调用
- 所以 ART 虚拟机提高了程序运行效率，减少手机电量消耗
- 机器码占用空间更大，所以 ART 下应用占用内部空间更大，首次安装因需要预编译所以时间相比 Dalvik 会略长

AsyncTask 的生命周期和(它所属的) Activity 的生命周期有什么关系？这种关系可能会导致什么样的问题？ 如何避免这些问题发生？

参考链接

- AsyncTask 并不会随着 Activity 的销毁而销毁，而是会一直执行 doInBackground 方法直到方法结束
- doInBackground 方法执行结束后，如果 cancel(boolean) 方法调用了，执行 onCancelled 方法；cancel(boolean) 没调用时，执行 onPostExecute 方法

导致的问题：

- 当使用非静态内部 AsyncTask 类时，AsyncTask 类会持有外部类 Activity 的引用，当 Activity 销毁时，如果 AsyncTask 还在执行，会造成 Activity 对象无法回收，内存泄漏

解决方案：

- 如果 doInBackground 方法内有循环操作时，应使用 isCancelled 来进行判断，避免后续无用的循环操作
- 销毁 Activity 实例时，调用 AsyncTask 的 cancel 方法
- 使用静态内部类，持有外部类的弱引用

Intent filter 是用来做什么的?

- Intent filter 即 Intent 过滤器，是应用清单文件中的表达式，用来指定该组件要接收的 Intent 类型
- 比如给 Activity 声明了 Intent filter 后，可以使其他应用使用某一特定类型的 Intent 启动该 Activity
- Intent filter 可以包括 action（必须有一个）、category 和 data

什么是 Sticky Intent? Link

- Sticky Intent 是发送粘性广播时的消息传递对象

什么是 AIDL？列举一下通过 AIDL 创建被绑定的服务（bounded service）的步骤。参考链接

Android 的权限有多少个不同的保护等级? 参考链接

- 正常权限：应用需要访问沙盒外部数据或资源，但对用户隐私或其他应用风险很小的区域。系统会自动授予应用正常权限
- 危险权限：应用需要涉及用户的隐私信息或资源，或对用户数据、其他应用产生影响。用户需手动向应用授予此类权限
- 特殊权限：SYSTEM_ALERT_WINDOW、WRITE_SETTINGS 等，大多数应用不该使用它们，需要的话需要发送请求用户授予的 Intent

在转屏时你如何保存 Activity 的状态？参考《Android 开发艺术探索》Page 8

- 资源相关的系统配置发生改变时会导致 Activity 被杀死并重新创建

保存 Activity 状态的方法：

- 系统会在 Activity 即将被销毁且有机会重新显示的情况下，会在 onStop 方法之前调用 onSaveInstanceState 方法
- 系统会默认帮我们保存当前 Activity 的视图结构，通过查看每个 View 的 onSaveInstanceState 和 onRestoreInstanceState 方法可以查看系统会自动为 View 保存哪些状态
- 我们可以手动在 onSaveInstanceState 方法中保存数据

恢复 Activity 状态：

- Activity 启动时，会回调 onCreate 和 onRestoreInstanceState 方法（onStart 方法之后调用），我们可以在这两个方法里拿到 Activity 销毁时保存的数据
- 在 onCreate 方法中取 Bundle 对象时，需要先判断是否为空，因为正常启动的 Activity 时系统无保存状态

常用布局有几种，区别

LinearLayout、RelativeLayout、FrameLayout、ConstraintLayout

区别

1、RelativeLayout 子 View 的排列方式是基于彼此的依赖关系，所以在测量时会在横向和纵向上分别测量一次

2、LinearLayout 为线性排列，在没有设置 weight 属性时只测量一次，设置了 weight 属性也是测量两次。

3、当布局较简单时使用 LinearLayout，布局复杂时使用 RelativeLayout 以降低布局层级深度

4、帧布局会默认把所有控件摆放在布局的左上角，并覆盖在前一控件的上层，当然可以通过 layout_gravity 属性来指定对齐方式

5、ConstraintLayout 使用约束的方式来指定各个控件的位置和关系，View 的位置受到三类约束：其他 View、父容器、基准线，并且支持设置比例。可以使布局完全扁平化，性能更高

如何实现 XML 命名空间？参考链接1、参考链接2

- 命名空间里存放的是特定属性的集合，可以避免元素命名冲突
- 为布局文件的根元素增加 xmlns 属性，即可通过不同的命名空间调用相应的属性

View.GONE 和 View.INVISIBLE 之间的区别。

- View.GONE：控件不可见，在布局中不占据空间
- View.INVISIBLE：控件不可见，但在布局中还占据空间

Bitmap 和 .9（nine-patch）图片之间有什么区别?

- 点 9 图是一种可伸缩的位图，可以指定图片哪些部分可以拉伸，哪些不可以拉伸。避免图片拉伸变形，在不同分辨率下可以达到较好的适配效果。
- 而 Bitmap 展示出来后在不同的屏幕分辨率下可能会被异常拉伸

谈谈位图池。Link

内存泄漏是什么，如何发现，为什么引起，如何解决（参考链接）

是什么

存在还被引用着，但已经无用的对象，造成该对象占用的空间无法被垃圾回收器回收

如何发现

1、静态代码分析工具：Lint

2、严格模式：StrictMode

3、Android Profiler（Android Memory Monitor）

4、LeakCanary

5、Memory Analyzer（MAT）

6、adb shell dumpsys meminfo [PackageName]

为什么引起

一般都是由于长生命周期对象持有短生命周期对象引用造成：

1、非静态内部类

2、单例模式

3、静态集合类

4、资源未及时释放

5、HashSet 集合中对象的属性被修改

如何解决

1、改用静态内部类

2、在单例模式中使用生命周期更长的 Context

3、通过程序逻辑切段非静态内部类所持有的外部类引用

4、及时关闭各种连接（数据库、网络、cursor 等），释放资源

Android 桌面的小部件是什么?

- AppWidgetProvider 的本质是一个广播接收器，继承自 BroadcastReceiver
- AppWidgetProvider 能接收 Widget 的相关广播，如 Widget 的更新、删除、开启、禁用等
- 桌面小部件的加载以及更新采用的都是 RemoteViews

什么是 AAPT？参考链接

- AAPT 是 Android Asset Packaging Tool 的缩写，为 SDK 自带的工具
- 可以查看、创建、更新 zip 格式的文档附件（zip、jar、apk），也可以将资源文件编译成二进制文件

你如何排查应用崩溃的原因?

- Java Crash：
  - 通过 Logcat 查看堆栈信息
  - 通过 UncaughtExceptionHandler 来记录异常日志并查看
  - ANR 可通过查看系统 /data/anr/ 文件夹下的 traces.txt 查看分析
  - 第三方 SDK 如腾讯的 Bugly，友盟等
- Native Crash：
  - 通过 Logcat 分析
  - 通过第三方 SDK

为什么你应该避免在主线程上运行非用户界面相关的代码?

- 如果主线程执行耗时操作的话，当 UI 事件发生时，让用户等待时间超过 5 秒而未处理就会出现 ANR
- 主线程主要负责把 UI 事件分发给合适的 View 或 Widget

你是如何做应用适配的？参考链接

1、屏幕适配

- 使用 wrap_content 和 match_parent 而非硬编码尺寸，以适应各种屏幕尺寸和方向
- 使用相对布局，禁用绝对布局
- 使用限定符（尺寸限定符、最小宽度限定符、屏幕方向限定符）
- 使用点九图
- 使用密度无关像素单位
- 图片文件放置到合适的 drawable 目录下

2、系统适配

- 同一 API 在不同版本下拥有不同的接口方法时，动态判断系统版本号
- 使用支持库提供的 API

px、dp、sp、ppi、dpi 有什么区别，如何换算，给出公式

1、px 为像素点数

2、ppi 与 dpi 均代表像素密度，即每英寸上的像素点数

3、dp 为像素无关密度，以 160 ppi 为基准，1dp = 1px

4、sp 与 dp 类似，用于描述字体大小，以 160 ppi 为基准，当字体大小为 100% 时，1sp = 1px

换算公式（待定？）：

```
ppi = √(屏幕高度像素数² + 屏幕宽度像素数²) / 屏幕对角线英寸数

px = dp * ppi / 160
px = sp * ppi / 160
```

如何理解 Doze 模式。如何理解应用程序待机模式（App Standby）。

在 Android 中，你可以使用什么来进行后台操作?

- 采用 Service
- 采用子线程搭配 Handler

什么是 ORM？它是如何工作的?

- ORM 即 Object Relational Mapping，对象关系映射
- 目前的数据库是关系型数据库，ORM 的工作就是把数据库中的关系数据映射为程序中的对象

什么是 Loader？参考链接

- Loader 即加载器，可以在 Activity 和 Fragment 中轻松实现异步加载数据
- 对数据源变化进行监听，实时更新数据

什么是 NDK，为什么它是有用的? 参考链接

- NDK 即原生开发工具包，是一组允许在 Android 应用使用原生代码语言（C、C++）的工具

优势：

- 可以从设备获取卓越的性能用于计算密集型应用，如游戏或物理模拟
- 可以复用 C 或 C++ 库

- 可以在平台之间移植应用

如何理解严格模式（StrictMode）。 参考链接

StrictMode 是用来检测程序中违例情况的开发者工具。最常用的场景是检查主线程中文件读写和网络读写等耗时操作。

StrictMode 主要检测两大问题：线程策略（ThreadPolicy）和 VM 策略（VMPolicy）

线程策略检查的内容有：

- 自定义的耗时调用
- 磁盘读取
- 磁盘写入
- 网络操作

VM 策略检查内容有：

- Activity 泄漏
- 未关闭的 Closable 对象泄漏
- Sqlite 对象泄漏
- 检测实例数量

什么是 Lint？它的用途是什么?

- Lint 是 Android Studio 提供的一个静态代码检查工具
- 帮助我们发现代码中的潜在 bug、安全、性能、国际化、辅助性、错误拼写等问题

什么是 SurfaceView？参考链接

- SurfaceView 拥有独立的绘图表面，即不与宿主窗口共享一个绘图表面
- 由于拥有独立的绘图表面，SurfaceView 的 UI 就可以在一个独立的线程中进行绘制
- 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，另一方面又会不会导致用户输入不能及时响应

ListView 和 RecyclerView 有什么区别？参考链接1，参考链接2

效率上：

- RecyclerView 效率较于 ListView 更高
- RecyclerView 的 ViewHolder 模式更加规范

功能上：

- RecyclerView 新增了 LayoutManager，布局效果更丰富，包括线性布局、网格布局、瀑布流布局等
- RecyclerView 高度解耦，LayoutManager 负责布局显示，ItemDecoration 负责 item 分割线，ItemAnimator 负责 item 增删动画。

使用上：

- ListView 自带空数据处理：setEmptyView 方法
- ListView 自带 HeaderView、FooterView
- RecyclerView 提供局部刷新方法：notifyItemChanged
- RecylerView 默认封装了一些动画

什么是 ViewHolder 模式？为什么我们应该使用它?

- 使用 ListView 或 RecyclerView 时，每个 item 在进入用户视野时会从 item 的 xml 文件中创建 view 对象，这一步可以利用 ConvertView 对创建好的布局进行缓存，直接从 ConvertView 中取即可。
- 拿到 item 布局后，再 findViewById 找到 item 的子 view 的控件对象，这一步是树查找操作极其耗时。
- 可以创建一个 ViewHolder 对象，将 item 子控件的实例都存放在 ViewHolder 对象中，这样就不用每次 findViewById 了，提高效率

ListView 如何优化

1、对布局进行缓存，可以避免在滚动时每次都将布局加载一遍

2、使用 ViewHolder 对布局控件的实例进行缓存，并把 ViewHolder 对象存储在刚缓存的 View 中。可以避免每次显示时都执行 finViewById 操作

3、分批加载与分页加载相结合

4、Adapter 的 getView 方法中减少耗时逻辑或加载图片等

5、减少 item 布局的层次

6、快速滑动时不要加载图片

什么是 PendingIntent ?

PendingIntent 可以理解为延迟执行的 Intent，在某个特定条件下才会执行该 Intent。

Flags 的类型：

- FLAG_ONE_SHOT：得到的 PendingIntent 只能使用一次，使用一次后自动调用 cancel 方法解除 PendingIntent 和 Intent 的关联
- FLAG_NO_CREATE：当 PendingIntent 不存在时，不进行创建，直接返回 null
- FLAG_CANCEL_CURRENT：当 PendingIntent 已存在时，执行 cancel 进行解除，再创建一个新的
- FLAG_UPDATE_CURRENT：不存在时就进行创建，创建后每次使用会对数据进行更新
- FLAG_IMMUTABLE：创建好 PendingIntent 后就保持一成不变

send 方法：

调用 send 方法时会启动包装的 Intent

cancel 方法：

调用 cancel 方法时会解除 PendingIntent 和被包装 Intent 之间的关联，只有创建该 PengdingIntent 的程序才有权解除

你能手动调用垃圾回收吗?

周期地更新页面的最好方式是什么?

- 使用 Alarm 机制
- 使用定时器 Timer 类

Alarm 机制简介：参考《第一行代码》Page469

- AlarmManager 设置定时任务的常用方法包括：

    - set(@AlarmType int type, long triggerAtMillis, PendingIntent operation)
    - setRepeating(@AlarmType int type, long triggerAtMillis,
      long intervalMillis, PendingIntent operation)
    - setExact(@AlarmType int type, long triggerAtMillis, PendingIntent operation)
- 顾名思义，set 方法设置的是执行一次的定时任务，setRepeating 方法设置的是循环执行的定时任务
- 从 Android 4.4 开始，系统会对 Alarm 唤醒对齐，所以 set 方法设置的 Alarm 任务执行可能会有一段时间的误差；而 setExact 方法可以保证唤醒时间的精准。
- 第一个参数 AlarmType，有以下值可选：
    - ELAPSED_REALTIME：定时任务的触发时间从系统开机算起，不会唤醒 CPU
    - ELAPSED_REALTIME_WAKEUP：会唤醒 CPU
    - RTC：定时任务的出发时间从 1970 年 1 月 1 日 0 点算起，不会唤醒 CPU
    - RTC_WAKEUP：会唤醒 CPU
    - 系统开机时间可通过 SystemClock.elapsedRealtime 获得
    - 从 1970 年 1 月 1 日 0 点起的时间可通过 System.currentTimeMillis 获得 intervalMillis 参数代表
- 第二个参数 triggerAtMillis 代表定时任务触发的时间
- 第三个参数是一个 PendingIntent 代表任务触发时的执行动作
- setRepeating 方法中的 intervalMillis 代表定时任务的执行时间间隔
- 另外有 setInexactRepeating 方法用于设定不精确时间的循环定时任务，比 setRepeating 方法更加节能

有哪些类型的广播?

按执行顺序分为：

- 标准广播：完全异步执行，所有广播接收器一起接到，无法被某个接收器截断
- 有序广播：同步执行，优先级高的广播接收器先接收，并可以截断该条广播

按广播定义类型分为：

- 系统广播：Android 系统发出的广播，比如电池电量改变、时区改变、网络状态改变等
- 自定义广播：自己定义的广播

按广播范围分为：

- 全局广播：可以被任何应用程序接收到的广播
- 本地广播：只能在本应用程序内部传递的广播

按注册后是否可接收分为：

- 粘性广播：粘性广播发送后，再进行注册的广播接收器也可接收到最后发出的一条该广播（注：粘性广播在 API 21 中已被 deprecated）
- 非粘性广播：广播发送后注册的广播接收器无法接收到该广播

如何理解上下文（Context）。怎么使用它? 参考链接1、参考链接2

- Android 程序要有一个完整的 Android 工程环境，这个工程环境下有四大组件，四大组件需要有各自的上下文
- 因此 Context 是维持 Android 程序中各个组件正常工作的一个重要功能类

使用中注意点：

- getApplication 和 getApplicationContext 都可以获取到 Application 实例，并且获取的是同一个对象
- getApplication 只可以在 Activity 和 Service 中调用，而任何一个 Context 实例都可以调用 getApplicationContext
- 和 UI 相关的不建议使用 Application Context，单例模式下需要考虑内存泄漏问题
- 上图数字1：启动 Activity 在这些类中是可以的，但是需要创建一个新的 task。一般情况不推荐。
- 上图数字2：在这些类中去 layout inflate 是合法的，但是会使用系统默认的主题样式，如果你自定义了某些样式可能不会被使用。
- 注：ContentProvider、BroadcastReceiver 之所以在上述表格中，是因为在其内部方法中都有一个 context 用于使用

Context 的继承关系：

由图可见：

- ContextWrapper 是 Context 的封装类，ContextImpl 是 Context 的实现类
- 我们平时需要 Context 调用的方法都在 ContextWrapper 中可以看到声明，均是调用了成员变量 mBase 的相应方法
- mBase 为 ContextImpl 对象，在合适的时机由系统调用 attachBaseContext 方法传递进来

你知道什么是视图树(View Tree)吗？怎样优化它的深度？

onTrimMemory() 方法是什么？参考链接1、参考链接2

1、onTrimMemory() 方法定义

- Android 4.0 之后加入的一个回调，任何实现了 CompomentCallbacks2 接口的类都可以重写实现该方法
- 主要作用是指导应用程序在不同的情况下进行自身的内存释放，以避免被系统直接杀掉
- 系统会根据不同等级的内存使用情况，调用这个回调方法，并传入相应的等级

等级分类如下：

- TRIM_MEMORY_UI_HIDDEN（20）：常用的一个等级，在 UI 界面被隐藏时回调。此时应释放一些 UI 占用的大块内存

程序正常运行时的回调：

- TRIM_MEMORY_RUNNING_MODERATE（5）：应用程序正常运行，进程不会被杀掉。但内存已经有点低了，系统可能会开始通过 LRU 缓存规则去杀死缓存进程了
- TRIM_MEMORY_RUNNING_LOW（10）：应用程序正常运行，进程不会被杀掉。但手机内存已经非常低了，此时应该释放不需要的资源
- TRIM_MEMORY_RUNNING_CRITICAL（15）：应用程序正常运行，但系统已经根据 LRU 缓存规则杀掉了大部分缓存进程。此时应尽可能释放不必要的资源，否则系统会继续杀死缓存进程，并可能开始杀死后台运行服务了

程序是缓存时的回调：

- TRIM_MEMORY_BACKGROUND（40）：手机内存已经很低了，系统准备开始根据 LRU 缓存规则来杀死进程了。此时我们的进程已经被加入 LRU 列表中了，此时释放一些资源可以使手机内存保持充足，从而使我们程序更长时间保存在缓存中
- TRIM_MEMORY_MODERATE（60）：手机内存已经很低了，此时我们的程序进程处于 LRU 缓存列表的中间位置，如果手机内存资源不能得到释放，我们的缓存进程就有可能被杀死
- TRIM_MEMORY_COMPLETE（80）：手机内存已经很低了，此时我们的程序进程处于 LRU 缓存列表的最边缘位置，系统将会优先考虑杀死我们的程序进程，此时应该释放所有能释放的资源

2、可以实现 onTrimMemory() 方法的组件包括：

- Application
- Activity
- Fragment
- Service
- ContentProvider

Android 应用可以使用多进程吗？怎样使用？参考链接

如何使用：

- 依赖于 android:process 属性
- 如果该属性值以 `:` 开头，代表这个进程是应用私有的，无法跨应用共用
- 如果该属性值以小写字母开头，代表这个进程是全局进程，可以被多个应用共用
- 适用于：Application、Activity、Service、Broadcast、ContentProvider

多进程的好处：

- 增加 App 可用内存
- 独立于主进程，确保某些任务的执行与完成

多进程的缺点：

- 数据共享问题，跨进程共享数据可以通过 Intent、Messenger、AIDL 实现
- 由于每个进程可能会使用自己的 SQLOpenHelper 实例，当两个进程同时对数据库操作时，会造成 SQLite 被锁
- Application.onCreate() 不必要的初始化，因为每个进程都会执行自己的 Application.onCreate() 方法

内存溢出（OutOfMemory）是怎么发生的？参考链接

内存溢出指程序在申请内存空间时，系统没法提供足够的内存空间供其使用

- 内存泄漏导致
- 保存了多个占用内存过大的对象（如 bitmap）或加载单个超大图片

加载 bitmap 导致的内存溢出解决方案：

- 加载多图使用软引用、弱引用
- 图不再使用时，使用 Bitmap.recycle 加速回收
- 使用文件缓存

文本样式接口（Spannable）是什么？

什么是过度绘制（overdraw）？参考链接、参考链接2

- GPU 过度绘制指：屏幕上的某个像素，在同一帧时间内，被绘制了多次

产生过度绘制原因：

- 多层 View 叠加绘制导致

解决过度绘制方法：

- 移除默认的 Window 背景
- 移除不必要的背景
- 写合理且高效的布局，减少层级嵌套
  - 使用 ViewStub 来加载一些不常用的布局
  - 使用 merge 标签减少布局嵌套层次
  - 可复用的组件抽取出来使用 include 引入
- 自定义控件进行优化（clipRect、quickReject）

FlatBuffers 和 JSON 的区别。Link

阐述一下 Android 中的 HashMap , ArrayMap 和 SparseArray 。Link

阐述一下 Looper, Handler 和 HandlerThread 。参考链接

一句话：Looper 不断从 MessageQueue 中取出 Message 交给相应的 Handler 处理。

以上称为消息处理机制（消息循环）。

- Android 中消息循环和消息队列都是针对具体线程的，除了 UI 线程之外，默认创建的工作线程是没有消息循环的
- Handler 用来将消息压入消息队列以及处理消息
- 普通工作线程想具有消息循环机制的话，先调用 Looper.prepare 创建消息队列、构造 Looper，再调用 Looper.loop 开启消息循环。此时该线程为 LooperThread
- 在 LooperThread 中创建 Handler 对象，此时 Handler 对象会自动关联到当前线程的 Looper 对象
- （构造 Handler 时如果不传 Looper，则会自动调用 mLooper = Looper.myLooper()，Handler 对象会自动关联到当前线程的 Looper 对象）
- 使用 HandlerThread 可以很方便的开启一个包含 Looper 的子线程，也就是 HandlerThread 自动帮我们 Looper.prepare，Looper.loop。我们只要调用 HandlerThread.start 开启线程后，通过该线程的 Looper 对象去构建相应的 Handler 对象即可。
- HandlerThread 提供了 quit 和 quitSafely 方法，可以很方便的终止线程消息队列

关于如何将 Message 压入 MessageQueue?

- 调用 Handler 的 send(Message message) 方法发送一个 Message，最终会调用到 MessageQueue 的 enqueueuMessage 方法，将消息放入消息队列
- 调用 Handler 的 post(Runnable r) 方法发送一个 Runnable，Runnable 先被封装为 Message 的 callback，再发送该 Message

- 调用 View 的 post(Runnable r) 方法发送一个 Runnable，和上一条类似。不过调用的是 UI 线程的 Handler 发送的 Message
- 调用 Activity 的 runOnUiThread(Runnable r) 方法，在 UI 线程调用时，直接执行 Runnable，在非 UI 线程调用时，调用 UI 线程的 Handler 将该 Runnable 发送出去

关于 Looper 从 MessageQueue 中取出消息后的分发?

- Looper 从消息队列取出消息后
- 首先调用 Handler 的 dispatchMessage 进行分发，我们可以重写此方法更改逻辑。
- dispatchMessage 的默认策略如下：
    - Message 的 callback 不为空时，优先调用 Message 的 callback
    - Handler 的 mCallback 不为空时，调用 Handler 的 mCallback
    - 上面俩都为空时，才调用 handleMessage，也就是我们经常重写的那个方法

Handler 的作用

1、负责消息的发送：通过调用 post 方法或 sendMessage 方法最终向消息队列中插入一条消息

2、负责消息的处理：首先判断 Message 的 callback 是否为 null，不为 null 直接执行其 run 方法；否则再判断 mCallback 是否为 null，不为 null 直接执行其 handleMessage 方法；否则调用 Handler 对象的 handleMessage 方法

Handler 是什么，原理，使用方法

是什么

Handler 用于线程间通信，主要负责 Android 消息机制中消息的发送和接收。

发送

通过向消息队列插入一条消息实现发送，

使用方法：

发送消息可以通过 Handler 对象的 sendMessage 的一系列方法和 post 一系列方法来实现。

接收

1、首先检查 Message 的 callback 是否为空，不为空则调用其 run 方法

2、如果为空，则检查 mCallback 是否为空，不为空调用其 handleMessage 方法。通过 mCallback 来创建 Handler 对象，可以实现不用派生 Handler 的子类就可以使用 Handler

3、如果为空，则调用 handleMessage 方法

接收的使用方法：

1、派生 Handler 的子类重写 handleMessage 方法

2、构造 Handler 对象时传入 Handler.Callback 参数

Handler 和 AsyncTask 的关系

AsyncTask 对 Handler 进行了封装，我们在使用时只需要派生 AsyncTask 的子类，并重写 onPreExecute、doInBackground、onProgressUpdate、onPostExecute 这几个方法即可。

谈谈对 RxJava 的理解

- Rxjava 是一个实现异步操作的库
- RxJava 基于事件，事件在整个过程中进行流动，流动的同时可以进行线程切换、事件转换等等
- 最常见的使用情景是：在子线程进行数据计算、网络请求，然后回到主线程展示结果

LayoutInflater 的 inflate 方法的几个参数分别代表什么？参考链接

- 获取到 LayoutInflater 对象可以通过 `LayoutInflater.from(Context context)` 方法或者 `activity.getLayoutInflater()` 方法
- resouse（int）代表：欲加载的 xml 布局文件的 ID
- root（ViewGroup）代表：当 attachToRoot 为 true 时，root 为 inflate 方法加载出来的 view 的根布局；attach 为 false 时，root 仅仅为将要加载出来的 View 提供一组 LayoutParams 参数
- attachToRoot（boolean）代表：是否要将加载出来的 View 附加到上面的 root 中
- return（View）代表：inflate 方法的返回值是一个 View，为即将加载出来的视图结构的根布局。如果 attachToRoot 为 true，则返回的 View 为刚才的 root；否则返回加载出来 View 的根布局

Maven 和 Gradle 的区别

Gradle 的优势在哪

Gradle 的功能：依赖管理、多模块构建、

- Maven 基于 XML 配置繁琐，阅读性差，Gradle 基于 Groovy，简化了构建代码的行数，易于阅读

1、依赖管理方面：Gradle 支持依赖动态版本管理，解决依赖冲突机制更明确

2、多模块构建方面：Gradle 使用 allprojects 和 subprojects 来定义里面的配置是应用于所有项目还是子项目，更加灵活

3、构建周期方面：Gradle 本身与项目构建周期是解耦的，可以灵活的增删 task

OkHttp、Retrofit 的区别

如何做性能优化？参考1

内存优化

1、避免内存泄漏

- 使用静态内部类加弱饮用的方式
- 单例模式使用生命周期更长的 Context
- 通过程序逻辑切段引用（关闭子线程、清除消息队列的所有消息）
- 静态集合中的无用对象及时移除
- 及时关闭无用的连接

2、图片加载进行优化，防止瞬间申请过大内存

- 按需加载（质量压缩、尺寸压缩）
- 图片的复用（三级缓存：内存缓存、磁盘缓存、网络拉取）
- 使用合适的颜色模式
- ListView、RecyclerView 滑动时不进行加载图片

UI 优化（布局优化、绘制优化）

1、布局优化

- 减少过度绘制

- 简单布局使用 LinearLayout，复杂布局使用 RelativeLayout，以减少布局嵌套，或者使用谷歌最新推出的 ConstraintLayout

2、绘制优化

在 View 的 onDraw 方法中：

- 避免创建新的局部对象，onDraw 方法是实时执行的，频繁创建临时对象会造成系统不断 gc，降低效率
- 避免执行耗时操作
- 避免使用循环操作

速度优化（线程优化、网络优化）

1、线程优化

- 解决 ANR 问题
- 避免在 UI 线程执行耗时操作，在子线程执行并使用 AsyncTask 或 Handler 来协助

2、网络优化

图片加载时

- 采用谷歌的 WebP 格式的图片，可大幅节省流量
- 使用缩略图

其他方面

- 对服务端返回的数据进行缓存
- 尽可能使用断点下载
- 刷新数据时进行局部刷新而不是全部刷新

电量优化

- 进行网络请求时，先判断网络状态
- 同时有 Wi-Fi 和移动网络的情况下，优先使用 Wi-Fi 网络请求
- 后台任务尽可能少的唤醒 CPU

启动优化

- Application 的创建过程减少耗时操作
- 减少布局层次，提高首次加载速度
- Activity 生命周期的回调方法中尽量减少耗时操作

MVC、MVP、MVVM 区别（参考）

1、在 MVC 中，View 为 XML 布局文件，Controller 对应于 Activity，处理数据、业务和 UI，Model 为实体模型（数据的获取、存储、状态变化等）。在 Android 中作为 View 的 XML 功能太弱，大量 View 的逻辑写在 Activity 中，所以 Activity 同时充当了 View 和 Controller 的角色，相当臃肿

2、MVP 模式中，View 对应于 Activity 和 XML，Model 依然是实体模型，Presenter 负责 View 与 Model 间的交互和业务逻辑。通过一个抽象的 View 接口将 Presenter 与 View 层进行解耦，Presenter 持有该 View 接口，对该接口进行操作，而不是直接操作 View 层，视图操作和业务逻辑进行了解耦。

缺点：

a、接口粒度不好控制

b、MVP 以 UI 为驱动，更新 UI 时需要考虑线程问题和 Activity 生命周期问题

c、View 和 Presenter 还是有一定的耦合，View 层某个元素发生变化，对应接口还是得改

3、MVVM 中 View 对应于 Activity 和 XML，Model 依然是实体模型，ViewModel 负责 View 和 Model 间的交互及业务逻辑。MVVM 与 MVP 的区别是

a、MVVM 是数据驱动的，数据变化后会自动更新 UI，UI 变化也能反馈到数据层

b、低耦合，ViewModel 只关注数据和业务逻辑，不持有 UI 控件引用，完全不和 UI 打交道，UI 控件变化时，ViewModel 也无需改变，和 MVP 相比，View 和 ViewModel 高度解耦

c、更新 UI 时无需考虑线程问题

d、可复用性强，一个 ViewModel 可以复用到多个 View 中

缺点：

调试比较麻烦：数据绑定使得一个位置的 Bug 被快速传递到别的位置，要定位原始出问题的地方变得不那么容易

介绍 Android 的内存管理机制

创建线程的几种方式，优缺点?

1、继承自 Thread

2、Runnable

3、Callable

4、FutureTask

Layout_gravity 和 gravity 区别，paddingLeft 和 Layout_marginLeft 区别

Layout_gravity 和 gravity 区别

- layout_gravity 用于指定控件在父布局中的对齐方式
- gravity 用于指定文字在控件中的对齐方式

paddingLeft 和 Layout_marginLeft 区别

- padding 指内边距，View 内部的内容到 View 的边的距离值
- margin 指外边距，View 的边到父视图的距离值

动画的种类有什么，有啥区别

动画包括 View 动画（补间动画）、帧动画（Drawable 动画）、属性动画。

1、View 动画仅能作用于 View，只改变了 View 的绘制效果，实际属性值未变

2、帧动画通过加载一系列的 Drawable 资源来创建动画，就像放电影

3、属性动画作用于 View 的属性，通过改变 View 对象的实际属性来实现动画

介绍下 Android 中的 IPC 机制（参考）

Android 中的 IPC 方式有：

- AIDL（Binder 机制）
- Intent / Bundle
- 文件共享
- Messenger（底层实现为 AIDL）

- ContentProvider
- Socket

Binder 机制：

1、一个进程向驱动申请成为 ServiceManager，来管理 Service

2、各个 Service 依次向 ServiceManager 进行注册

3、Client 想同 Service 进行通信，向 ServiceManager 进行请求对象

4、ServiceManager 给 Client 返回代理对象

5、Client 调用代理对象的方法，传给 ServiceManager

6、ServiceManager 再传递给 Service，拿到 Service 的返回值给 Client

介绍下 AIDL 的原理（参考）

1、IBinder 是一个接口，代表一种跨进程传输的能力，实现了这个接口的对象就拥有了跨进程传输的能力

2、IInterface 接口代表了远程 Server 对象所具有的能力

3、静态抽象类 Stub 类为 Binder 本地对象，Stub 类实现了 IBinder、IInterface 接口，说明 Stub 类为 Binder 本地对象，且拥有 Client 需要的能力。

4、Stub 类的 onTransact 方法会处理 Client 传递来的方法：先通过 code 确定是哪个方法，然后从 data 中取出目标方法所需的参数，执行方法，最后向 reply 中写入值。

5、Stub 类有一个静态内部类 Proxy，为 Binder 代理对象。代理对象对于接口方法的处理如下：首先用 Parcel 把数据序列化，然后发起远程调用，调用 Server 端本地对象的 onTransact 方法，远程调用结束后，读取执行结果。