数据存储

1、描述一下 Android 数据持久存储方式？

- 参考回答：Android 平台实现数据持久存储的常见几种方式：
  - **SharedPreferences 存储**：一种轻型的数据存储方式，本质是基于 XML 文件存储的 key-value 键值对数据，通常用来存储一些简单的配置信息（如应用程序的各种配置信息）；
  - **SQLite 数据库存储**：一种轻量级嵌入式数据库引擎，它的运算速度非常快，占用资源很少，常用来存储大量复杂的关系数据；
  - **ContentProvider**：四大组件之一，用于数据的存储和共享，不仅可以让不同应用程序之间进行数据共享，还可以选择只对哪一部分数据进行共享，可保证程序中的隐私数据不会有泄漏风险；
  - **File 文件存储**：写入和读取文件的方法和Java 中实现 I/O 的程序一样；
  - **网络存储**：主要在远程的服务器中存储相关数据，用户操作的相关数据可以同步到服务器上；

2、SharedPreferences 的应用场景？注意事项？

- 参考回答：
  - SharedPreferences 是一种轻型的数据存储方式，本质是基于 XML 文件存储的 key-value 键值对数据，通常用来存储一些简单的配置信息，如 int，String，boolean、float 和 long；
  - 注意事项：
    - **勿存储大型复杂数据，这会引起内存 GC、阻塞主线程使页面卡顿产生 ANR**
    - **勿在多进程模式下，操作 Sp**
    - **不要多次 edit 和 apply，尽量批量修改一次提交**
    - **建议 apply，少用 commit**

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

- 参考回答：
  - **apply 没有返回值而 commit 返回 boolean 表明修改是否提交成功。**
  - **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。

- **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。

4、了解 SQLite 中的事务操作吗？是如何做的

- 参考回答：
  - SQLite 在做 CRDU 操作时都默认开启了事务，然后把 SQL 语句翻译成对应的 SQLiteStatement 并调用其相应的 CRUD 方法，此时整个操作还是在 rollback journal 这个临时文件上进行，只有操作顺利完成才会更新 db 数据库，否则会被回滚；

5、使用 SQLite 做批量操作有什么好的方法吗？

- 参考回答：
  - 使用 SQLiteDatabase 的 beginTransaction 方法开启一个事务，将批量操作 SQL 语句转化为 SQLiteStatement 并进行批量操作，结束后 endTransaction()

6、如何删除 SQLite 中表的个别字段

- 参考回答：
  - SQLite 数据库只允许增加字段而不允许修改和删除表字段，只能创建新表保留原有字段，删除原表

7、使用 SQLite 时会有哪些优化操作？

- 参考回答：
  - 使用事务做批量操作
  - 及时关闭 Cursor，避免内存泄露
  - 耗时操作异步化：数据库的操作属于本地 IO 耗时操作，建议放入异步线程中处理
  - ContentValues 的容量调整：ContentValues 内部采用 HashMap 来存储 Key-Value 数据，ContentValues 初始容量为 8，扩容时翻倍。因此建议对 ContentValues 填入的内容进行估量，设置合理的初始化容量，减少不必要的内部扩容操作
  - 使用索引加快检索速度：对于查询操作量级较大、业务对查询要求较高的推荐使用索引

IPC

1、Android 中进程和线程的关系？ 区别？

- 参考回答：
  - 线程是 CPU 调度的**最小单元**，同时线程是一种**有限**的系统资源
  - 进程一般指一个执行单元，在 PC 和移动设备上一个程序或则一个应用
  - 一般来说，一个 App 程序**至少有一个**进程，一个进程**至少有一个**线程（包含与被包含的关系）， 通俗来讲就是，在 App 这个工厂里面有一个进程，线程就是里面的生产线，但主线程（主生产线）只有一条，而子线程（副生产线）可以有多个
  - 进程有自己独立的地址空间，而进程中的线程共享此地址空间，都可以**并发**执行

2、如何开启多进程 ？ 应用是否可以开启 N 个进程 ？

- 参考回答：
  - 在 AndroidMenifest 中给四大组件指定属性 android:process 开启多进程模式
  - 在内存允许的条件下可以开启 N 个进程

3、为何需要 IPC？多进程通信可能会出现的问题？

- 参考回答：
  - 所有运行在不同进程的四大组件（Activity、Service、Receiver、ContentProvider）共享数据都会失败，这是由于 Android 为每个应用分配了独立的虚拟机，不同的虚拟机在内存分配上有不同的地址空间，这会导致在不同的虚拟机中访问同一个类的对象会产生多份副本。比如常用例子（**通过开启多进程获取更大内存空间、两个或则多个应用之间共享数据、微信全家桶**）
  - 一般来说，使用多进程通信会造成如下几方面的问题
    - **静态成员和单例模式完全失效**：独立的虚拟机造成
    - **线程同步机制完全实效**：独立的虚拟机造成
    - **SharedPreferences 的可靠性下降**：这是因为 Sp 不支持两个进程并发进行读写，有一定几率导致数据丢失
    - **Application 会多次创建**：Android 系统在创建新的进程会分配独立的虚拟机，所以这个过程其实就是启动一个应用的过程，自然也会创建新的 Application

4、Android 中 IPC 方式、各种方式优缺点，为什么选择 Binder？

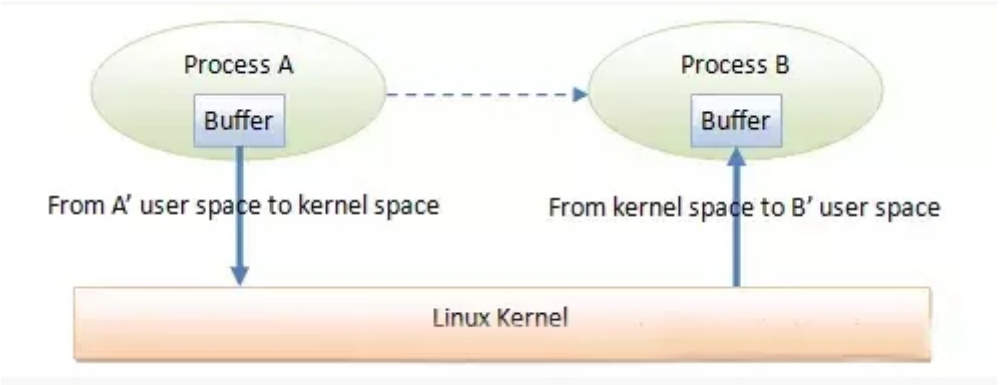
- 参考回答：

进程间通信的方式-对比

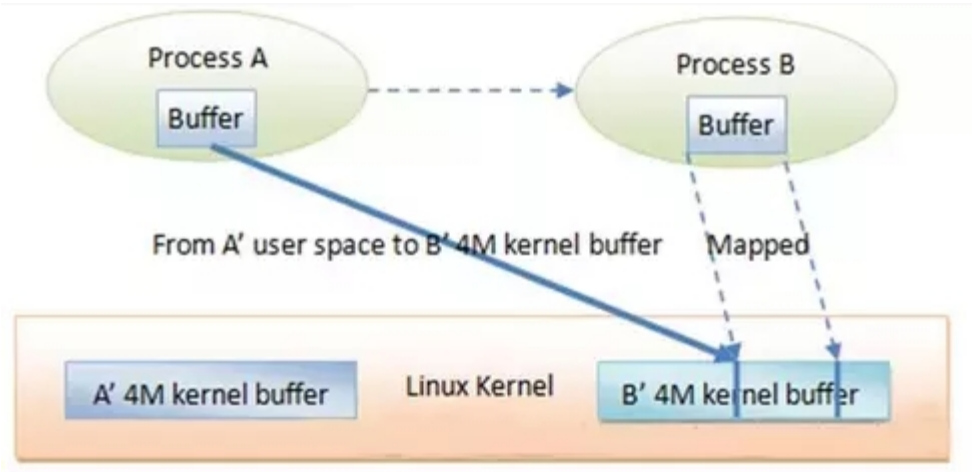
名称	优点	缺点	适用场景
Intent	简单易用	只能传输Bundle所支持的数据类型	四大组件间的进程间通信
文件共享	简单易用	不适合高并发	简单的数据共享，无高并发场景
AIDL	功能强大，支持一对多并发实时通信	使用稍微复杂，需要注意线程同步	复杂的进程间调用，Android中最常用
Messenger	比AIDL稍微简单易用些	比AIDL功能弱，只支持一对多串行实时通信	简单的进程间通信
ContentProvider	强大的数据共享能力，可通过call方法扩展	受约束的AIDL，主要对外提供数据线的CRUD操作	进程间的大量数据共享
RemoteViews	在跨进程访问UI方面有奇效	比较小众的通信方式	某些特殊的场景
Socket	跨主机，通信范围广	只能传输原始的字节流	常用于网络通信中

与 Linux 上传统的 IPC 机制，比如 System V，Socket 相比，Binder 好在哪呢？

- **传输效率高、可操作性强**：传输效率主要影响因素是内存拷贝的次数，拷贝次数越少，传输速率越高。从 Android 进程架构角度分析：对于消息队列、Socket 和管道来说，数据先从发送方的缓存区拷贝到内核开辟的缓存区中，再从内核缓存区拷贝到接收方的缓存区，一共两次拷贝，如图：



而对于 Binder 来说，数据从发送方的缓存区拷贝到内核的缓存区，而接收方的缓存区与内核的缓存区是映射到同一块物理地址的，节省了一次数据拷贝的过程，如图：



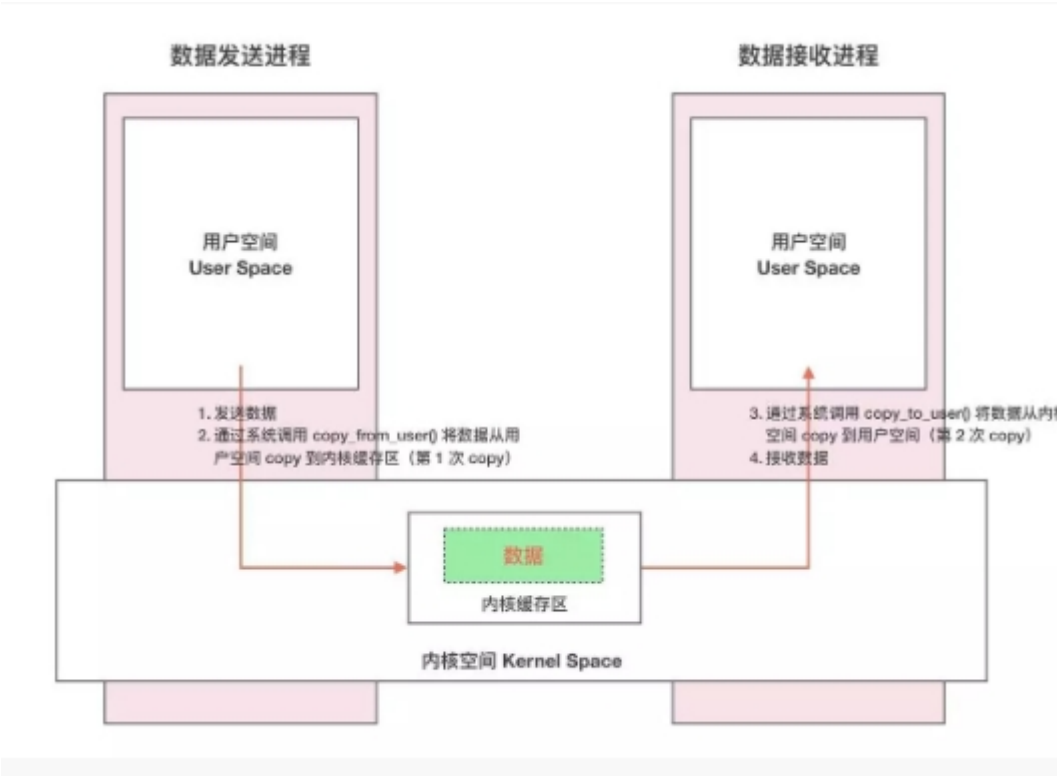
由于共享内存操作复杂，综合来看，Binder 的传输效率是最好的。

- **实现 C/S 架构方便**：Linux 的众 IPC 方式除了 Socket 以外都不是基于 C/S 架构，而 Socket 主要用于网络间的通信且传输效率较低。Binder 基于 C/S 架构 ，Server 端与 Client 端相对独立，稳定性较好。
- **安全性高**：传统 Linux IPC 的接收方无法获得对方进程可靠的 UID/PID，从而无法鉴别对方身份；而 Binder 机制为每个进程分配了 UID/PID 且在 Binder 通信时会根据 UID/PID 进行有效性检测。

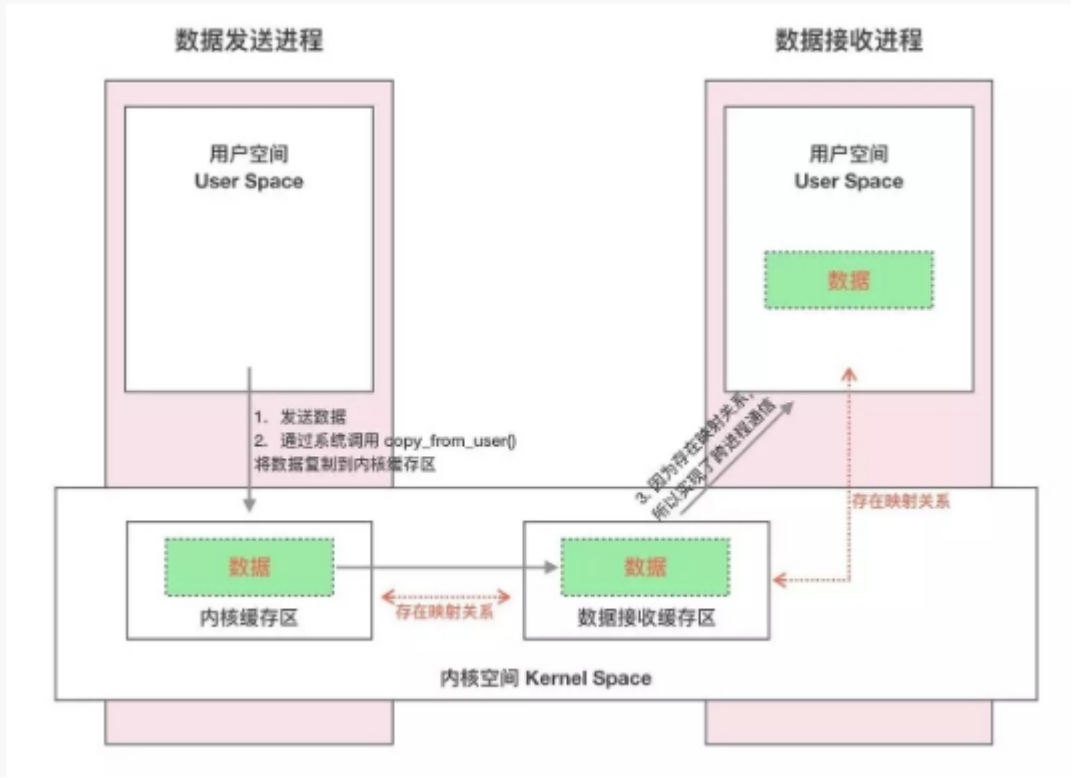
5、Binder 机制的作用和原理？

- 参考回答：
  - Linux 系统将一个进程分为**用户空间**和**内核空间**。对于进程之间来说，用户空间的数据不可共享，内核空间的数据可共享，为了保证安全性和独立性，一个进程不能直接操作或者访问另一个进程，即 Android 的进程是相互独立、

隔离的，这就需要跨进程之间的数据通信方式

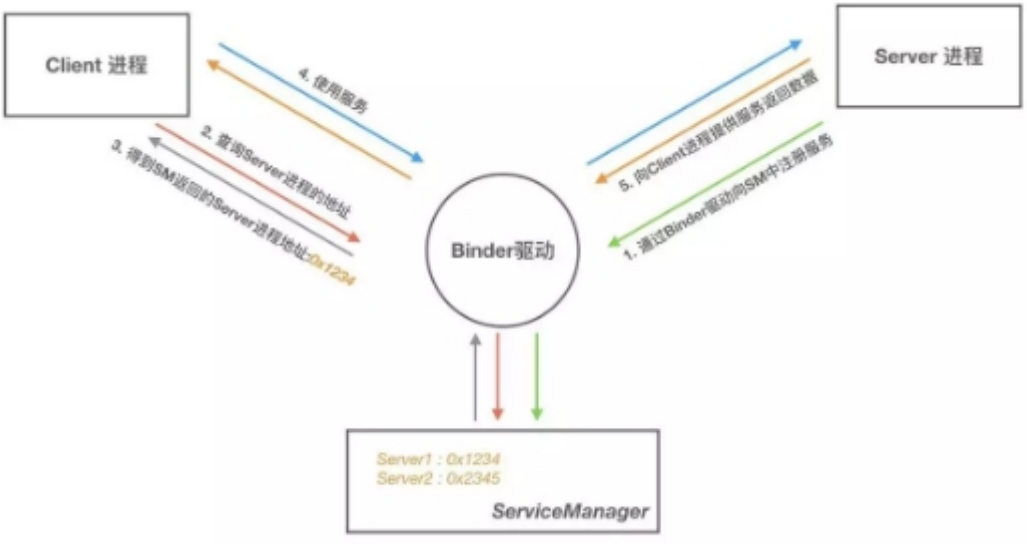


- 一次完整的 Binder IPC 通信过程通常是这样：
  - 首先 Binder 驱动在内核空间创建一个数据接收缓存区；
  - 接着在内核空间开辟一块内核缓存区，建立内核缓存区和内核中数据接收缓存区之间的映射关系，以及内核中数据接收缓存区和接收进程用户空间地址的映射关系；
  - 发送方进程通过系统调用 copyfromuser() 将数据 copy 到内核中的内核缓存区，由于内核缓存区和接收进程的用户空间存在内存映射，因此也就相当于把数据发送到了接收进程的用户空间，这样便完成了一次进程间的通信。

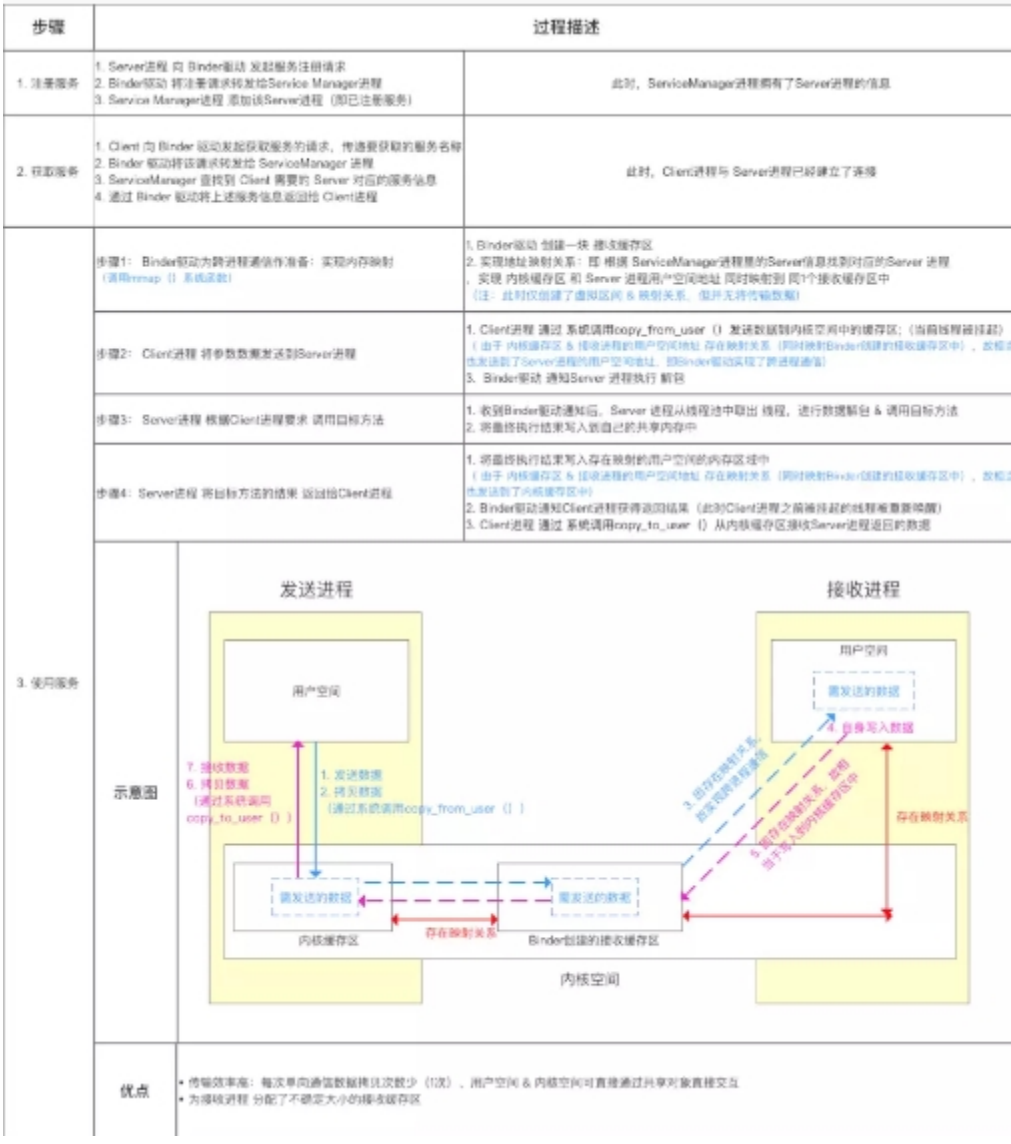


6、Binder 框架中 ServiceManager 的作用？

- 参考回答：
  - **Binder 框架** 是基于 C/S 架构的。由一系列的组件组成，包括 Client、Server、ServiceManager、Binder 驱动，其中 Client、Server、Service Manager 运行在用户空间，Binder 驱动运行在内核空间



- **Server&Client**：服务器&客户端。在 Binder 驱动和 Service Manager 提供的基础设施上，进行 Client-Server 之间的通信。
- **ServiceManager**（如同 DNS 域名服务器）服务的管理者，将 Binder 名字转换为 Client 中对该 Binder 的引用，使得 Client 可以通过 Binder 名字获得 Server 中 Binder 实体的引用。
- **Binder 驱动**（如同路由器）：负责进程之间 binder 通信的建立，传递，计数管理以及数据的传递交互等底层支持。



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

- 参考回答：
  - 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。

- 序列化实现的两种方式：实现 Serializable/Parcelable 接口。不同点如图：

	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、讲讲 AIDL？原理是什么？如何优化多模块都使用 AIDL 的情况？

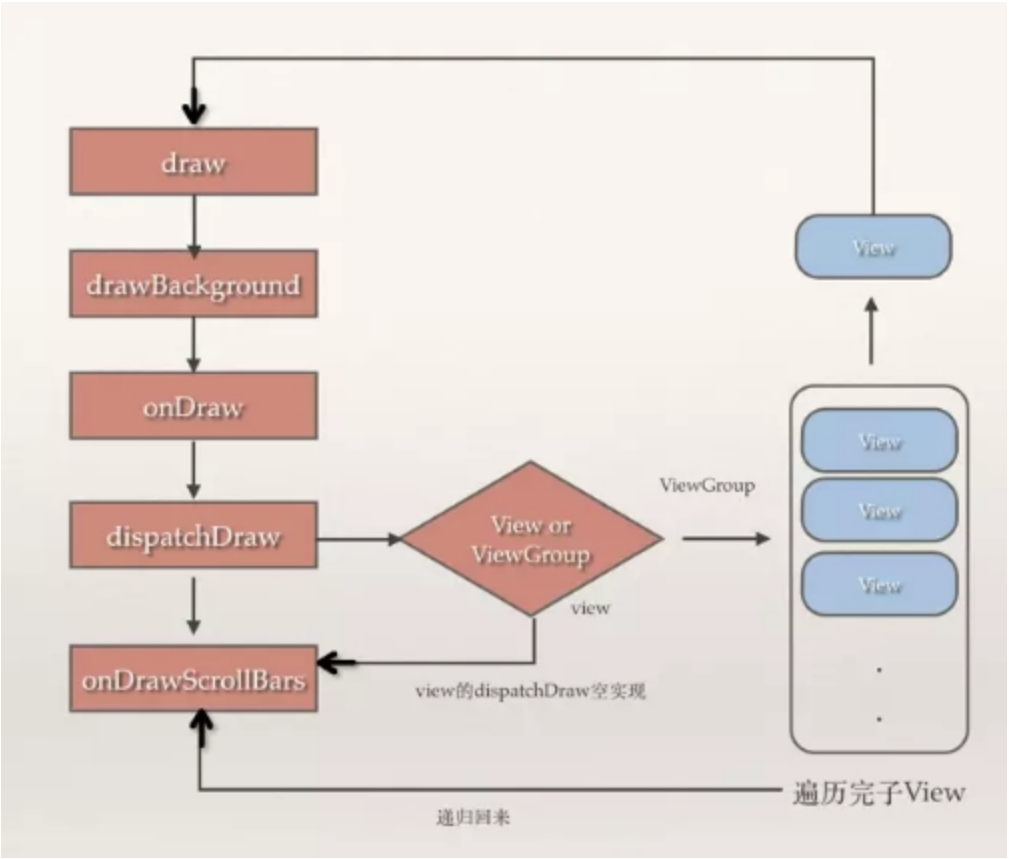
- 参考回答：
  - AIDL(Android Interface Definition Language，Android 接口定义语言)：如果在一个进程中要调用另一个进程中对象的方法，可使用 AIDL 生成可序列化的参数，AIDL 会生成一个服务端对象的代理类，通过它客户端实现间接调用服务端对象的方法。
  - AIDL 的本质是系统提供了一套可快速实现 Binder 的工具。关键类和方法：
    - **AIDL 接口**：继承 IInterface。
    - **Stub 类**：Binder 的实现类，服务端通过这个类来提供服务。
    - **Proxy 类**：服务器的本地代理，客户端通过这个类调用服务器的方法。
    - **asInterface()**：客户端调用，将服务端的返回的 Binder 对象，转换成客户端所需要的 AIDL 接口类

型对象。如果客户端和服务端位于统一进程，则直接返回 Stub 对象本身，否则返回系统封装后的 Stub.proxy 对象
        - **asBinder()**：根据当前调用情况返回代理 Proxy 的 Binder 对象。
        - **onTransact()**：运行服务端的 Binder 线程池中，当客户端发起跨进程请求时，远程请求会通过系统底层封装后交由此方法来处理。
        - **transact()**：运行在客户端，当客户端发起远程请求的同时将当前线程挂起。之后调用服务端的 onTransact()直到远程请求返回，当前线程才继续执行。
    - 当有多个业务模块都需要 AIDL 来进行 IPC，此时需要为每个模块创建特定的 aidl 文件，那么相应的 Service 就会很多。必然会出现系统资源耗费严重、应用过度重量级的问题。解决办法是建立 Binder 连接池，即将每个业务模块的 Binder 请求统一转发到一个远程 Service 中去执行，从而避免重复创建 Service。
        - **工作原理**：每个业务模块创建自己的 AIDL 接口并实现此接口，然后向服务端提供自己的唯一标识和其对应的 Binder 对象。服务端只需要一个 Service，服务器提供一个 queryBinder 接口，它会根据业务模块的特征来返回相应的 Binder 对象，不同的业务模块拿到所需的 Binder 对象后就可进行远程方法的调用了

View

1、讲下 View 的绘制流程？

- 参考回答：
    - View 的工作流程主要是指 measure、layout、draw 这三大流程，即测量、布局和绘制，其中 measure 确定 View 的**测量宽/高**，layout 确定 View 的**最终宽/高**和**四个顶点的位置**，而 draw 则将 View **绘制到屏幕**上
    - View 的绘制过程遵循如下几步：
        - **绘制背景** background.draw(canvas)
        - **绘制自己**（onDraw）
        - **绘制 children**（dispatchDraw）
        - **绘制装饰**（onDrawScollBars）



2、MotionEvent 是什么？包含几种事件？什么条件下会产生？

- 参考回答：
  - MotionEvent 是手指接触屏幕后所产生的一系列事件。典型的事件类型有如下：
    - ACTION_DOWN：手指刚接触屏幕
    - ACTION_MOVE：手指在屏幕上移动
    - ACTION_UP：手指从屏幕上松开的一瞬间
    - ACTION_CANCELL：手指保持按下操作，并从当前控件转移到外层控件时触发
  - 正常情况下，一次手指触摸屏幕的行为会触发一系列点击事件，考虑如下几种情况：
    - 点击屏幕后松开，事件序列：DOWN→UP
    - 点击屏幕滑动一会再松开，事件序列为 DOWN→MOVE→.....→MOVE→UP

3、描述一下 View 事件传递分发机制？

- 参考回答：
  - View 事件分发本质就是对 MotionEvent 事件分发的过程。即当一个 MotionEvent 发生后，系统将这个点击事件传递到一个具体的 View 上

- 点击事件的传递顺序：Activity（Window）→ViewGroup→ View
- 事件分发过程由三个方法共同完成：
  - **dispatchTouchEvent**：用来进行事件的分发。如果事件能够传递给当前 View，那么此方法一定会被调用，返回结果受当前 View 的 onTouchEvent 和下级 View 的 dispatchTouchEvent 方法的影响，表示是否消耗当前事件
  - **onInterceptTouchEvent**：在上述方法内部调用，对事件进行拦截。该方法只在 ViewGroup 中有，View（不包含 ViewGroup）是没有的。一旦拦截，则执行 ViewGroup 的 onTouchEvent，在 ViewGroup 中处理事件，而不接着分发给 View。且只调用一次，返回结果表示是否拦截当前事件
  - **onTouchEvent**： 在 dispatchTouchEvent 方法中调用，用来处理点击事件，返回结果表示是否消耗当前事件

4、如何解决 View 的事件冲突 ？ 举个开发中遇到的例子 ？

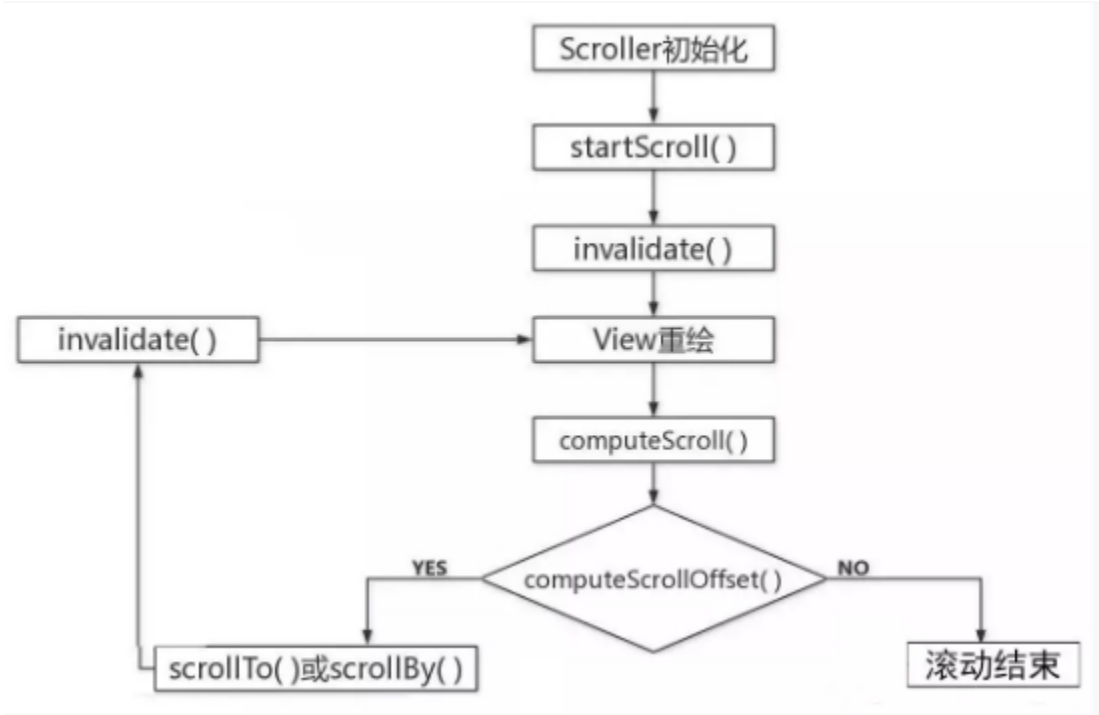
- 参考回答：
  - 常见开发中事件冲突的有 ScrollView 与 RecyclerView 的滑动冲突、RecyclerView 内嵌同时滑动同一方向
  - 滑动冲突的处理规则：
    - 对于由于外部滑动和内部滑动方向不一致导致的滑动冲突，可以根据滑动的方向判断谁来拦截事件。
    - 对于由于外部滑动方向和内部滑动方向一致导致的滑动冲突，可以根据业务需求，规定何时让外部 View 拦截事件，何时由内部 View 拦截事件。
    - 对于上面两种情况的嵌套，相对复杂，可同样根据需求在业务上找到突破点。
  - 滑动冲突的实现方法：
    - **外部拦截法**：指点击事件都先经过父容器的拦截处理，如果父容器需要此事件就拦截，否则就不拦截。具体方法：需要重写父容器的 onInterceptTouchEvent 方法，在内部做出相应的拦截。
    - **内部拦截法**：指父容器不拦截任何事件，而将所有的事件都传递给子容器，如果子容器需要此事件就直接消耗，否则就交由父容器进行处理。具体方法：需要配合 requestDisallowInterceptTouchEvent 方法。

5、scrollTo()和 scollBy()的区别？

- 参考回答：
  - scollBy 内部调用了 scrollTo，它是基于当前位置的相对滑动；而 scrollTo 是绝对滑动，因此如果使用相同输入参数多次调用 scrollTo 方法，由于 View 的初始位置是不变的，所以只会出现一次 View 滚动的效果
  - 两者都只能对 View 内容的滑动，而非使 View 本身滑动。可以使用 Scroller 有过度滑动的效果

6、Scroller 是怎么实现 View 的弹性滑动？

- 参考回答：
  - 在 MotionEvent.ACTION_UP 事件触发时调用 startScroll()方法，该方法并没有进行实际的滑动操作，而是记录滑动相关量（滑动距离、滑动时间）
  - 接着调用 invalidate/postInvalidate()方法，请求 View 重绘，导致 View.draw 方法被执行
  - 当 View 重绘后会在 draw 方法中调用 computeScroll 方法，而 computeScroll 又会去向 Scroller 获取当前的 scrollX 和 scrollY；然后通过 scrollTo 方法实现滑动；接着又调用 postInvalidate 方法来进行第二次重绘，和之前流程一样，如此反复导致 View 不断进行小幅度的滑动，而多次的小幅度滑动就组成了弹性滑动，直到整个滑动过成结束



7、 invalidate()和 postInvalidate()的区别 ？

- 参考回答：
  - invalidate()与 postInvalidate()都用于刷新 View，主要区别是 invalidate()在主线程中调用，若在子线程中使用需要配合 handler；而 postInvalidate()可在子线程中直接调用。

8、SurfaceView 和 View 的区别？

- 参考回答：
  - View 需要在 UI 线程对画面进行刷新，而 SurfaceView 可在子线程进行页面的刷新
  - View 适用于主动更新的情况，而 SurfaceView 适用于被动更新，如频繁刷新，这是因为如果使用 View 频繁刷新会阻塞主线程，导致界面卡顿
  - SurfaceView 在底层已实现双缓冲机制，而 View 没有，因此 SurfaceView 更适用于需要频繁刷新、刷新时数据处理量很大的页面（如视频播放界面）

9、自定义 View 如何考虑机型适配 ？

- 参考回答：
  - 合理使用 warp_content，match_parent
  - 尽可能的是使用 RelativeLayout
  - 针对不同的机型，使用不同的布局文件放在对应的目录下，android 会自动匹配。
  - 尽量使用点 9 图片。
  - 使用与密度无关的像素单位 dp，sp
  - 引入 android 的百分比布局。
  - 切图的时候切大分辨率的图，应用到布局当中。在小分辨率的手机上也会有很好的显示效果。