Android篇

Activity

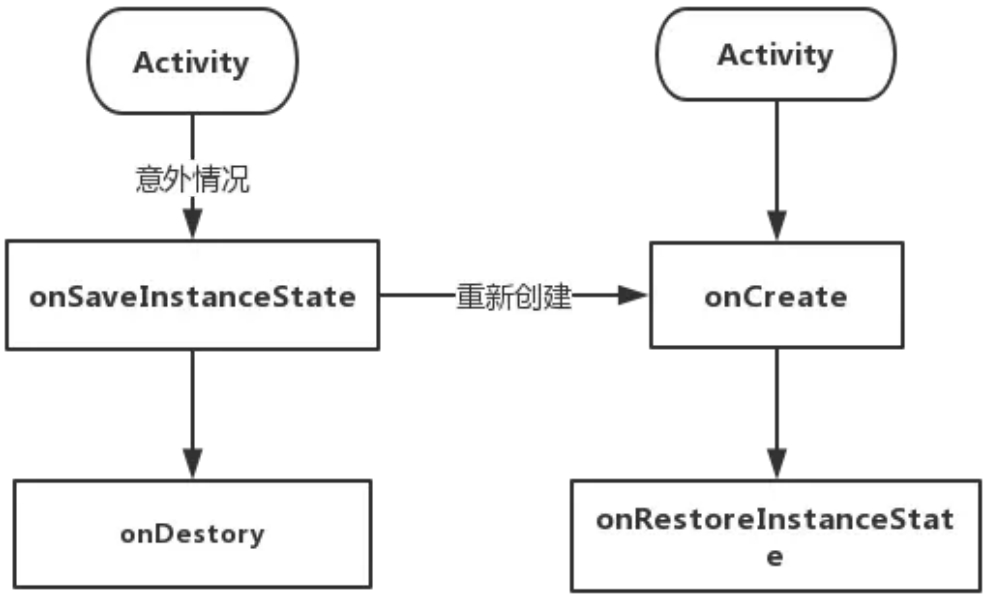
1、说下Activity生命周期 ？

- 参考解答：在正常情况下，Activity的常用生命周期就只有如下7个
  - **onCreate()**：表示Activity**正在被创建**，常用来**初始化工作**，比如调用setContentView加载界面布局资源，初始化Activity所需数据等；
  - **onRestart()**：表示Activity**正在重新启动**，一般情况下，当前Acitivty从不可见重新变为可见时，OnRestart就会被调用；
  - **onStart()**：表示Activity**正在被启动**，此时Activity**可见但不在前台**，还处于后台，无法与用户交互；
  - **onResume()**：表示Activity**获得焦点**，此时Activity**可见且在前台**并开始活动，这是与onStart的区别所在；
  - **onPause()**：表示Activity**正在停止**，此时可做一些**存储数据、停止动画**等工作，但是不能太耗时，因为这会影响到新Activity的显示，onPause必须先执行完，新Activity的onResume才会执行；
  - **onStop()**：表示Activity**即将停止**，可以做一些稍微重量级的回收工作，比如注销广播接收器、关闭网络连接等，同样不能太耗时；
  - **onDestroy()**：表示Activity**即将被销毁**，这是Activity生命周期中的最后一个回调，常做**回收工作、资源释放**；
- 延伸：从**整个生命周期**来看，onCreate和onDestroy是配对的，分别标识着Activity的创建和销毁，并且只可能有**一次调用**； 从Activity**是否可见**来说，onStart和onStop是配对的，这两个方法可能被**调用多次**； 从Activity**是否在前台**来说，onResume和onPause是配对的，这两个方法可能被**调用多次**； 除了这种区别，在实际使用中没有其他明显区别；

- 参考解答：Activity A 启动另一个Activity B，回调如下
  - Activity A 的onPause() → Activity B的onCreate() → onStart() → onResume() → Activity A的onStop()；
  - 如果B是透明主题又或则是个DialogActivity，则不会回调A的onStop；

3、说下onSaveInstanceState()方法的作用？何时会被调用？

- 参考解答：发生条件：异常情况下（**系统配置发生改变时导致Activity被杀死并重新创建、资源内存不足导致低优先级的Activity被杀死**）
  - 系统会调用onSaveInstanceState来保存当前Activity的状态，此方法调用在onStop之前，与onPause没有既定的时序关系；
  - 当Activity被重建后，系统会调用onRestoreInstanceState，并且把onSave(简称)方法所保存的Bundle对象**同时传参**给onRestore(简称)和onCreate()，因此可以通过这两个方法判断Activity**是否被重建**，调用在onStart之后；



- 推荐文章：
  - 官方文档

4、说下 Activity的四种启动模式、应用场景 ？

- 参考回答：

- **singleTop栈顶复用模式**：如果新Activity已经位于任务栈的栈顶，那么此Activity不会被重新创建，同时会回调**onNewIntent**方法，如果新Activity实例已经存在但不在栈顶，那么Activity依然会被重新创建；
- **singleTask栈内复用模式**：只要Activity在一个任务栈中存在，那么多次启动此Activity都不会重新创建实例，并回调**onNewIntent**方法，此模式启动Activity A，系统首先会寻找是否存在A想要的任务栈，如果不存在，就会重新创建一个任务栈，然后把创建好A的实例放到栈中；
- **singleInstance单实例模式**：这是一种加强的singleTask模式，具有此种模式的Activity只能单独地位于一个任务栈中，且此任务栈中只有唯一一个实例；

- 推荐文章：
  - 官方文档

5、了解哪些Activity常用的标记位Flags？

- 参考回答：
  - **FLAG_ACTIVITY_NEW_TASK :** 对应singleTask启动模式，其效果和在XML中指定该启动模式相同；
  - **FLAG_ACTIVITY_SINGLE_TOP :** 对应singleTop启动模式，其效果和在XML中指定该启动模式相同；
  - **FLAG_ACTIVITY_CLEAR_TOP :** 具有此标记位的Activity，当它启动时，在同一个任务栈中所有位于它上面的Activity都要出栈。这个标记位一般会和singleTask模式一起出现，在这种情况下，被启动Activity的实例如果已经存在，那么系统就会回调onNewIntent。如果被启动的Activity采用standard模式启动，那么它以及连同它之上的Activity都要出栈，系统会创建新的Activity实例并放入栈中；
  - **FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS :** 具有这个标记的 Activity 不会出现在历史 Activity 列表中；
- 推荐文章：
  - 官方文档

6、说下 Activity跟window，view之间的关系？

- 参考回答：
  - Activity在创建时会调用 **attach()** 方法初始化一个**PhoneWindow(继承于Window)**，**每一个Activity都包含了唯一一个PhoneWindow**
  - Activity通过**setContentView**实际上是调用的 **getWindow().setContentView**将View设置到PhoneWindow上，而PhoneWindow内部是通过 **WindowManager**

- 延伸
  - **WindowManager**为每个**Window**创建**Surface**对象，然后应用就可以通过这个**Surface**来绘制任何它想要绘制的东西。而对于**WindowManager**来说，这只不过是一块矩形区域而已
    - **Surface**其实就是一个持有像素点矩阵的对象，这个像素点矩阵是组成显示在屏幕的图像的一部分。我们看到显示的每个**Window**（包括对话框、全屏的**Activity**、状态栏等）都有他自己绘制的**Surface**。而最终的显示可能存在**Window**之间遮挡的问题，此时就是通过**SurfaceFlinger**对象渲染最终的显示，使他们以正确的**Z-order**显示出来。一般**Surface**拥有一个或多个缓存（一般2个），通过双缓存来刷新，这样就可以一边绘制一边加新缓存。
  - **View**是**Window**里面用于交互的**UI**元素。**Window**只**attach**一个**View Tree（组合模式）**，当**Window**需要重绘（如，当**View**调用**invalidate**）时，最终转为**Window**的**Surface**，**Surface**被锁住（**locked**）并返回**Canvas**对象，此时**View**拿到**Canvas**对象来绘制自己。当所有**View**绘制完成后，**Surface**解锁（**unlock**），并且**post**到绘制缓存用于绘制，通过**Surface Flinger**来组织各个**Window**，显示最终的整个屏幕
- 推荐文章：
  - Activity、View、Window的理解一篇文章就够了

7、横竖屏切换的Activity生命周期变化？

- 参考回答：
  - **不设置**Activity的android:configChanges时，切屏会销毁当前Activity，然后重新加载调用各个生命周期，切横屏时会执行一次，切竖屏时会执行两次； onPause()→onStop()→onDestory()→onCreate()→onStart()→onResume()
  - 设置Activity的android:configChanges="**orientation**"，经过机型测试
    - **在Android5.1 即API 23级别下**，切屏还是会重新调用各个生命周期，切横、竖屏时只会执行一次
    - **在Android9 即API 28级别下**，切屏不会重新调用各个生命周期，只会执行onConfigurationChanged方法
    - **后经官方查正**，原话如下
      - 如果您的应用面向**Android 3.2即API 级别 13**或更高级别（按照 minSdkVersion 和 targetSdkVersion 属性所声明的级别），则还应声明 "screenSize" 配置，因为当设备在横向与纵向之间切换时，该配置也会发生变化。即便是在 Android 3.2 或更高版本的设备上运行，此配置变更也不会重新启动 Activity

测试通过，切屏不会重新调用各个生命周期，只会执行onConfigurationChanged方法；

- 推荐文章：
  - Android 横竖屏切换加载不同的布局

8、如何启动其他应用的Activity？

- 参考回答：
  - 在保证有权限访问的情况下，通过隐式Intent进行目标Activity的IntentFilter匹配，原则是：
    - 一个intent只有同时匹配某个Activity的intent-filter中的action、category、data才算完全匹配，才能启动该Activity；
    - 一个Activity可以有多个 intent-filter，一个 intent只要成功匹配任意一组 intent-filter，就可以启动该Activity；
- 推荐文章：
  - action、category、data的具体匹配规则

9、Activity的启动过程？(重点)

- 参考回答：
  - **点击App图标后通过startActivity远程调用到AMS中，AMS中将新启动的activity以activityrecord的结构压入activity栈中，并通过远程binder回调到原进程，使得原进程进入pause状态，原进程pause后通知AMS我pause了**
  - **此时AMS再根据栈中Activity的启动intent中的flag是否含有new_task的标签判断是否需要启动新进程，启动新进程通过startProcessXXX的函数**
  - **启动新进程后通过反射调用ActivityThread的main函数，main函数中调用looper.prepar和lopper.loop启动消息队列循环机制。最后远程告知AMS我启动了。AMS回调handleLauncherAcitivyt加载activity。在handlerLauncherActivity中会通过反射调用Application的onCreate和activity的onCreate以及通过handleResumeActivity中反射调用Activity的onResume**

- 推荐文章：
  - Android四大组件启动机制之Activity启动过程

Fragment

1、谈一谈Fragment的生命周期？

- 参考回答：
  - Fragment从创建到销毁整个生命周期中涉及到的方法依次为：onAttach()→onCreate()→onCreateView()→onActivityCreated()→onStart()→onResume()→onPause()→onStop()→onDestroyView()→onDestroy()→onDetach()，其中和Activity有不少名称相同作用相似的方法，而不同的方法有:
    - **onAttach()**：当Fragment和Activity建立关联时调用；
    - **onCreateView()**：当fragment创建视图调用，在onCreate之后；
    - **onActivityCreated()**：当与Fragment相关联的Activity完成onCreate()之后调用；
    - **onDestroyView()**：在Fragment中的布局被移除时调用；
    - **onDetach()**：当Fragment和Activity解除关联时调用；
- 推荐文章：
  - Android之Fragment优点

- 参考回答：
  - 相似点：都可包含布局、可有自己的生命周期
  - 不同点：
    - Fragment相比较于Activity多出4个回调周期，在控制操作上更灵活；
    - Fragment可以在XML文件中直接进行写入，也可以在Activity中动态添加；
    - Fragment可以使用show()/hide()或者replace()随时对Fragment进行切换，并且切换的时候不会出现明显的效果，用户体验会好；Activity虽然也可以进行切换，但是Activity之间切换会有明显的翻页或者其他的效果，在小部分内容的切换上给用户的感觉不是很好；

3、Fragment中add与replace的区别（Fragment重叠）

- 参考回答：
  - add不会重新初始化fragment，replace每次都会。所以如果在fragment生命周期内获取获取数据,使用replace会重复获取；
  - 添加相同的fragment时，replace不会有任何变化，add会报IllegalStateException异常；
  - replace先remove掉相同id的所有fragment，然后在add当前的这个fragment，而add是覆盖前一个fragment。所以如果使用add一般会伴随hide()和show()，避免布局重叠；
  - 使用add，如果应用放在后台，或以其他方式被系统销毁，再打开时，hide()中引用的fragment会销毁，所以依然会出现布局重叠bug，可以使用replace或使用add时，添加一个tag参数；

- 参考回答：
  - getFragmentManager()所得到的是所在fragment 的**父容器**的管理器，getChildFragmentManager()所得到的是在fragment 里面**子容器**的管理器， 如果是fragment嵌套fragment，那么就需要利用getChildFragmentManager()；
  - 因为Fragment是3.0 Android系统API版本才出现的组件，所以3.0以上系统可以直接调用getFragmentManager()来获取FragmentManager()对象，而3.0以下则需要调用getSupportFragmentManager() 来间接获取；

5、FragmentPagerAdapter与FragmentStatePagerAdapter的区别与使用场景

- 参考回答：
  - 相同点 ：二者都继承PagerAdapter
  - 不同点 ：**FragmentPagerAdapter**的每个Fragment会持久的保存在FragmentManager中，只要用户可以返回到页面中，它都不会被销毁。因此适用于那些数据**相对静态**的页，Fragment**数量也比较少**的那种；
    FragmentStatePagerAdapter只保留当前页面，当页面不可见时，该Fragment就会被消除，释放其资源。因此适用于那些**数据动态性**较大、**占用内存**较多，多Fragment的情况；

Service

1、谈一谈Service的生命周期？

- 参考回答：Service的生命周期涉及到六大方法
  - **onCreate()**：如果service没被创建过，调用startService()后会执行onCreate()回调；如果service已处于运行中，调用startService()不会执行onCreate()方法。也就是说，onCreate()只会在第一次创建service时候调用，多次执行startService()不会重复调用onCreate()，此方法适合完成一些初始化工作；
  - **onStartComand()**：服务启动时调用，此方法适合完成一些数据加载工作，比如会在此处创建一个线程用于下载数据或播放音乐；
  - **onBind()**：服务被绑定时调用；
  - **onUnBind()**：服务被解绑时调用；
  - **onDestroy()**：服务停止时调用；
- 推荐文章：
  - Android组件系列----Android Service组件深入解析

- 参考回答：Service的两种启动模式
  - **startService()**：通过这种方式调用startService，onCreate()只会被调用一次，多次调用startSercie会多次执行onStartCommand()和onStart()方法。如果外部没有调用stopService()或stopSelf()方法，service会一直运行。
  - **bindService()**：如果该服务之前**还没创建**，系统回调顺序为onCreate()→onBind()。如果调用bindService()方法前服务**已经被绑定**，多次调用bindService()方法不会多次创建服务及绑定。如果调用者希望与正在绑定的服务**解除绑定**，可以调用unbindService()方法，回调顺序为onUnbind()→onDestroy()；

- 推荐文章：
  - Android Service两种启动方式详解

3、如何保证Service不被杀死 ？

- 参考回答：

        - **START_STICKY**：如果返回START_STICKY，表示Service运行的进程被Android系统强制杀掉之后，Android系统会将该Service依然设置为started状态（即运行状态），但是不再保存onStartCommand方法传入的intent对象
        - **START_NOT_STICKY**：如果返回START_NOT_STICKY，表示当Service运行的进程被Android系统强制杀掉之后，不会重新创建该Service
        - **START_REDELIVER_INTENT**：如果返回START_REDELIVER_INTENT，其返回情况与START_STICKY类似，但不同的是系统会保留最后一次传入onStartCommand方法中的Intent再次保留下来并再次传入到重新创建后的Service的onStartCommand方法中
    - **提高Service的优先级** 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；
    - **在onDestroy方法里重启Service** 当service走到onDestroy()时，发送一个自定义广播，当收到广播时，重新启动service；
    - **提升Service进程的优先级** 进程优先级由高到低：前台进程 — 可视进程 — 服务进程 — 后台进程 — 空进程 可以使用startForeground将service放到前台状态，这样低内存时，被杀死的概率会低一些；
    - **系统广播监听Service状态**
    - **将APK安装到/system/app，变身为系统级应用**
- **注意**：以上机制都不能百分百保证Service不被杀死，除非做到系统白名单，与系统同生共死

4、能否在Service开启耗时操作 ？ 怎么做 ？

- 参考回答：
    - Service默认并不会运行在子线程中，也不运行在一个独立的进程中，它同样执行在主线程中（UI线程）。换句话说，不要在Service里执行耗时操作，除非手动打开一个子线程，否则有可能出现主线程被阻塞（ANR）的情况；

5、用过哪些系统Service ？

- 参考回答：

6、了解ActivityManagerService吗？发挥什么作用

- 参考回答： ActivityManagerService是Android中最核心的服务 ， 主要负责系统中四大组件的启动、切换、调度及应用进程的管理和调度等工作，其职责与操作系统中的进程管理和调度模块类似；
- 推荐文章：
  - ActivityManagerService分析——AMS启动流程

Broadcast Receiver

1、广播有几种形式？都有什么特点 ？

- 参考回答：
  - 普通广播：开发者自身定义 intent的广播（最常用），所有的广播接收器几乎会在同一时刻接受到此广播信息，**接受的先后顺序随机**；
  - 有序广播：发送出去的广播被广播接收者**按照先后顺序接收**，同一时刻只会有一个广播接收器能够收到这条广播消息，当这个广播接收器中的逻辑执行完毕后，广播才会继续传递，且优先级（priority）高的广播接收器会先收到广播消息。有序广播可以被接收器截断使得后面的接收器无法收到它；
  - 本地广播：仅在自己的应用内发送接收广播，也就是只有自己的应用能收到，数据更加安全，效率更高，但只能采用**动态注册**的方式；
  - 粘性广播：这种广播会**一直滞留**，当有匹配该广播的接收器被注册后，该接收器就会收到此条广播；
- 推荐文章：
  - Android四大组件：BroadcastReceiver史上最全面解析

- 参考回答：

3、广播发送和接收的原理了解吗 ？（ Binder机制、AMS ）

- 参考回答：

- 推荐文章：
  - 广播的底层实现原理

ContentProvider

1、ContentProvider了解多少？

- 参考回答：

存下的数据只能被该应用程序使用，而前者可以让不同应用程序之间进行数据共享，它还可以选择只对哪一部分数据进行共享，从而保证程序中的隐私数据不会有泄漏风险。

- 推荐文章：
  - Android：关于ContentProvider的知识都在这里了！

2、ContentProvider的权限管理？

- 参考回答：
  - 读写分离
  - 权限控制-精确到表级
  - URL控制

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

- 参考回答：
  - **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。

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；

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**

- 推荐文章：
  - 史上最全面，清晰的SharedPreferences解析
  - SharedPreferences在多进程中的使用及注意事项

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 journ
    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；

- 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
- 推荐文章：
  - Android developer官方文档--进程和线程

- 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
- 推荐讲解：
  - 如何开启多进程?应用是否可以开启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
- 推荐文章：
  - Android developer官方文档--进程和线程

4、Android中IPC方式、各种方式优缺点，为什么选择Binder？

- 参考回答：

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。

- 推荐文章：
  - 为什么 Android 要采用 Binder 作为 IPC 机制？

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。

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。

图片出自Carson_Ho文章 —— Android跨进程通信：图文详解 Binder机制 原理

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：

8、讲讲AIDL？原理是什么？如何优化多模块都使用AIDL的情况？

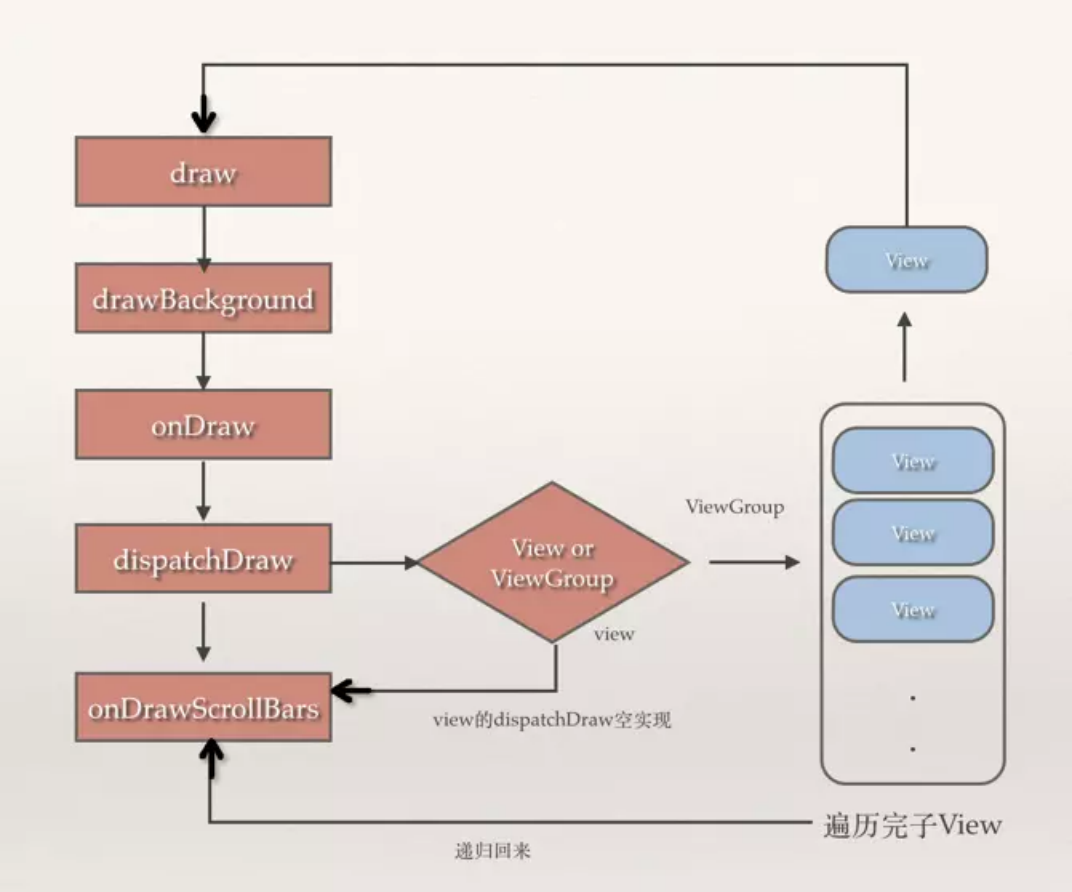
- 参考回答：
  - AIDL(Android Interface Definition Language，Android接口定义语言)：如果在一个进程中要调用另一个进程中对象的方法，可使用AIDL生成可序列化的参数，AIDL会生成一个服务端对象的代理类，通过它客户端实现间接调用服务端对象的方法。
  - AIDL的本质是系统提供了一套可快速实现Binder的工具。关键类和方法：
    - **AIDL接口**：继承IInterface。
    - **Stub类**：Binder的实现类，服务端通过这个类来提供服务。
    - **Proxy类**：服务器的本地代理，客户端通过这个类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

> 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）



- 推荐文章：
  - 官方文档

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。

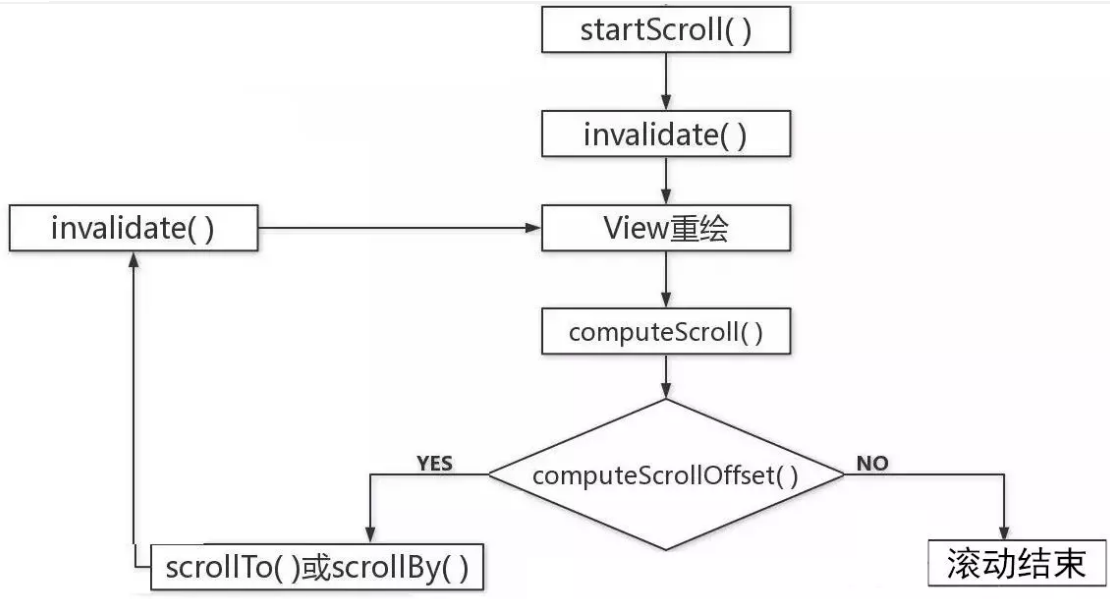
- 滑动冲突的实现方法：
  - **外部拦截法**：指点击事件都先经过父容器的拦截处理，如果父容器需要此事件就拦截，否则就不拦截。具体方法：需要重写父容器的onInterceptTouchEvent方法，在内部做出相应的拦截。
  - **内部拦截法**：指父容器不拦截任何事件，而将所有的事件都传递给子容器，如果子容器需要此事件就直接消耗，否则就交由父容器进行处理。具体方法：需要配合requestDisallowInterceptTouchEvent方法。

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
- 推荐文章：
  - View 的滑动原理和实现方式

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。

- 引入android的自分比布局。
- 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。

你当前所处：Android篇：2019初中级Android开发社招面试解答（上）

Android篇：2019初中级Android开发社招面试解答（中）

Android篇：2019初中级Android开发社招面试解答（下）

关注下面的标签，发现更多相似文章

Android

评论

输入评论...

小如童童 IT @ 路启
横竖屏切换的Activity生命周期变化？
这个是不是有点问题，在配置了 android:configChanges="orientation" 之后 生命周期不会走了啊，只会调onConfigurationChanged

1月前 回复

爵小友 Android开发工程...
回复 小如童童: 我也试了横竖屏幕切换，有点不对
1月前

Android的后花园 Lv3 (作者)
回复 小如童童: 已更正 😃