# Activity的面试基础详解

### 1，activity的四种状态

running、paused、stopped、killed

### 2，Android的进程优先级

前台 / 可见 / 服务 / 后台 / 空

### 3，Android的任务栈，启动模式

栈：先进后出

* standard （sidadende） 默认模式，每次启动会创建新实例
* singleTop 栈顶复用模式 ，新activity在栈顶，就复用，并回调onNewIntent方法；如果存在且不再栈顶，那么Activity会被重新创建； 使用场景：浏览器的书签，通讯消息聊天界面
* singleTask 栈内复用模式，检查整个栈内如果有那就复用，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； 使用场景：某个Activity当做主界面的时候
* singleInstance 单实例模式，加强版singleTask，该Activity只能单独位于一个任务栈且该栈只有它一个 使用场景：比如浏览器BrowserActivity很耗内存，很多app都会要调用它，这样就可以把该Activity设置成单例模式。比如：闹钟闹铃。

### 4，Scheme跳转协议

通过自定义scheme协议，方便跳转app中的各个页面 通过scheme协议，服务器可以定制化告诉App跳转到那个页面 通过通知栏消息定制化跳转页面，通过H5页面跳转页面等。

### 5，生命周期

启动： onCreate （用户不可见） -> onStart （用户可见但不在前台在后台，无法与用户交互） -> onResume (用户可见，在前台并获得焦点) 点击Home回主界面（Activity不可见） -> onPause -> onStop 再次回到原Activity -> onRestart -> onStart -> onResume 退出Activity -> onPause -> onStop -> onDestory

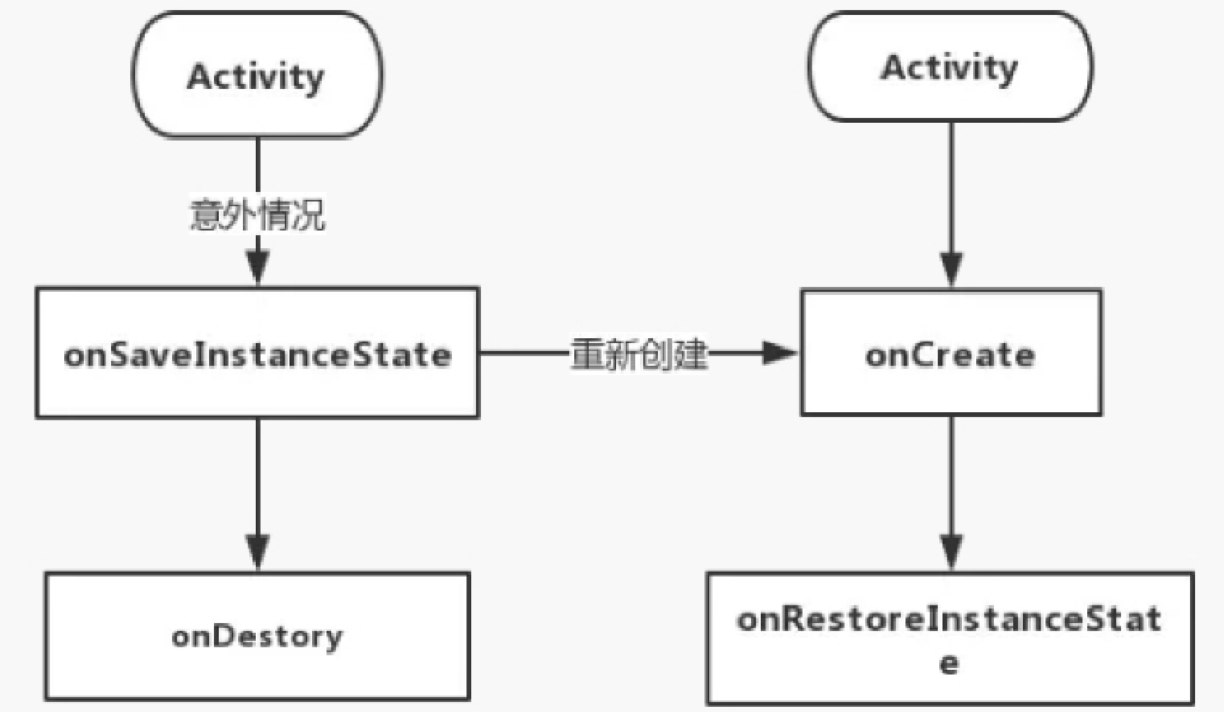
* onCreate：创建，常用来初始化工作，比如调用 setContentView 加载界面布局资源，初始化 Activity 所需数据等；
* onDestroy：表即将被销毁，这是 Activity 生命周期中的最后一个回调，常做回收工作、资源释放；
* onStart：启动，此时 Activity 可见但不在前台，还处于后台，无法与用户交互；
* onStop：即将停止，可以做一些稍微重量级的回收工作，比如注销广播接收器、关闭网络连接等，同样不能太耗时；
* onResume：获得焦点，此时 Activity 可见且在前台并开始活动，这是与 onStart 的区别所在；
* onPause：正在停止，可做存储数据、停止动画等工作，但是不能太耗时，因为会影响到新 Activity的显示，onPause 必须先执行完，新 Activity 的 onResume 才会执行；
* onRestart： 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；

### 6，Activity A 启动另一个 Activity B 会调用哪些方法？如果 B 是透明主题的又或则是个 DialogActivity 呢 ？

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；

### 7，说下 onSaveInstanceState()方法的作用 ? 何时会被调用？

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 onSaveInstanceState方法所保存的 Bundle 对象同时传参给onRestoreInstanceState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； 

### 8，了解哪些 Activity 常用的标记位 Flags？

* 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；

### 9，说下 Activity 跟 window，view 之间的关系？

Activity 创建时通过 attach()初始化了一个 Window 也就是PhoneWindow，一个 PhoneWindow 持有一个 DecorView 的实例，

DecorView 本身是一个 FrameLayout，继承于 View，Activty 通过setContentView 将 xml 布局控件不断 addView()添加到 View 中，

最终显示到 Window 于我们交互；

Activity剪窗花的人（控制的）；Window窗户（承载的一个模型）；View窗花（要显示的视图View）；LayoutInflater剪刀---将布局（图纸）剪成窗花。

这个问题真的很不好回答。所以这里先来个算是比较恰当的比喻来形容下它们的关系吧。Activity像一个工匠（控制单元），Window像窗户（承载模型），View像窗花（显示视图）LayoutInflater像剪刀，Xml配置像窗花图纸。

1：Activity构造的时候会初始化一个Window，准确的说是PhoneWindow。 2：这个PhoneWindow有一个“ViewRoot”，这个“ViewRoot”是一个View或者说ViewGroup，是最初始的根视图。  3：“ViewRoot”通过addView方法来一个个的添加View。比如TextView，Button等 4：这些View的事件监听，是由WindowManagerService来接受消息，并且回调Activity函数。比如onClickListener，onKeyDown等。

### 10，横竖屏切换的 Activity 生命周期变化？

* 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 Android 9 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

官方纠正后，原话如下

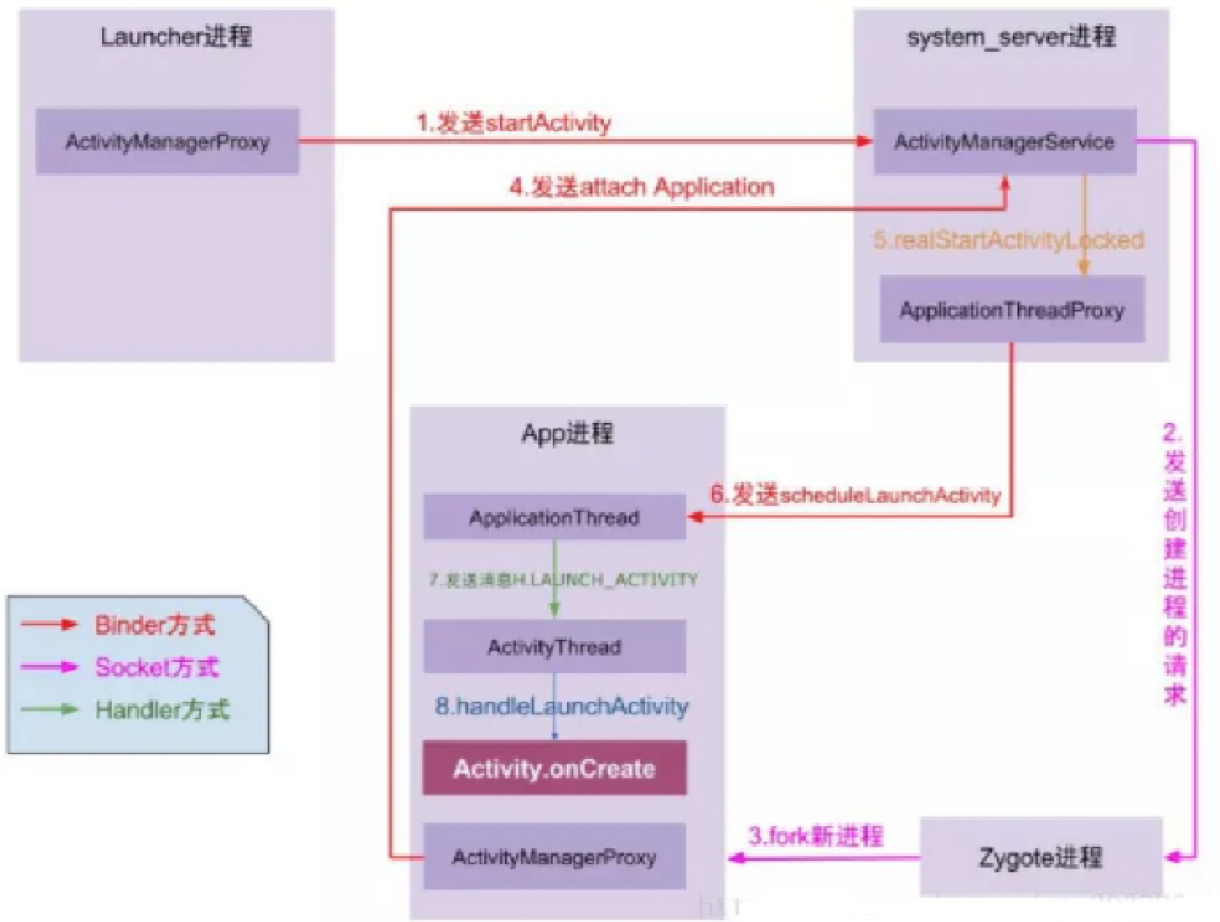
如果您的应用面向 Android 3.2 即 API 级别 13 或更高级别（按照 minSdkVersion 和 targetSdkVersion属性所声明的级别），则还应声明 "screenSize" 配置，因为当设备在横向与纵向之间切换时，该配置也会发生变化。即便是在 Android 3.2 或更高版本的设备上运行，此配置变更也不会重新启动Activity

* 设置 Activity 的android:configChanges="orientation|keyboardHidden|screenSize"时，机型测试通过，切屏不会重新调用各个生命周期，只会执行 onConfigurationChanged 方法；

### 11，如何启动其他应用的 Activity？

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 intentfilter 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；

### 12，Activity 的启动过程？

1. 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
2. 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
3. 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 

### 13，Activity之间传值有哪几种方法？

1，通过Intent直接传参数 2，通过Bundle封装数据传指 3，通过startActivityforResult回调，接口回调 4，通过发送广播的形式，数据需要序列化 5，通过存储介质，比如数据库，sharedpreference，文件等 6，通过事件总线EventBus形式

### 14，设备横竖屏切换的时候，接下来会发生什么？

1、不设置Activity的android:configChanges时，切屏会重新调用各个生命周期，切横屏时会执行一次，切竖屏时会执行两次

2、设置Activity的android:configChanges=”orientation”时，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

3、设置Activity的android:configChanges=”orientation|keyboardHidden”时，切屏不会重新调用各个生命周期，只会执行onConfigurationChanged方法

# Fragment的面试详解

### 1，Fragment为什么被称为第五大组件

Fragment一开始是用于平板的扩展页面，后面全部应用于Activity内部切换 Fragment有生命周期，切依附于Activity

### 2，Fragment的加载到Activity的2种方式

* 添加Fragment到Activity的布局文件xml中
* 动态在Activity中添加Fragment

### 3，FragmentPagerAdapter与FragmentStatePagerAdapter的区别？

二者都继承 PagerAdapter

* 前者适用于页面较少，数据相对静态的页面，Fragment数量较少的情况，其destroyItem方法里是detach，并不会回收内存
* 后者适用于页面较多，数据动态性较大，占用内存较多，多Fragment的情况，其destoryItem里面是remove，会释放内存；页面不可见会移除Fragment释放资源

viewpager中，fragment嵌套fragment的时候必须使用FragmentStatePageAdapter才起作用

### 4，Fragment的生命周期

Fragment生命周期比Activity多5个方法，Fragment里没有onRestart onAttach，onDetach，onCreateView，onDestoryView onViewCreated和onActivityCreated

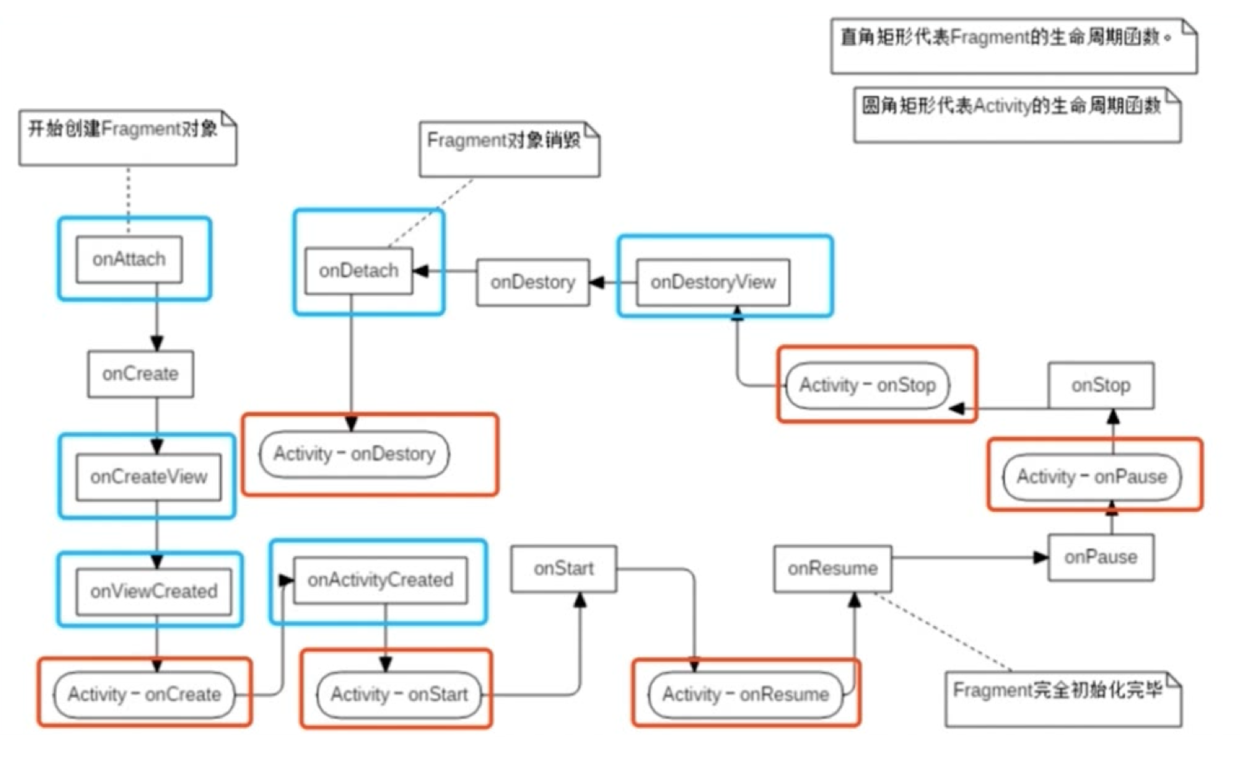


image.png

### 5，Fragment的通信

* 在frgment中调用Activity中的方法，getActivity
* 在Activity中调用Fragment中的方法，接口回调
* 在Fragment中调用Fragment中的方法，findFragmentById

### 6，getFragmentManager、getSupportFragmentManager 、getChildFragmentManager 之间的区别？

* 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；

### 7，Fragment中的add与replace的区别（Fragment重叠）

* 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；

# Service面试详解

参考：<https://blog.csdn.net/javazejian/article/details/52709857>

### 1，Service是什么？

Service是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，无法做耗时的操作

### 2，Service和Thread的区别

service运行在主线程中，无法做耗时的操作

thread是线程的最小单元，一般指耗时线程

### 3，Service启动方式

* startService

1. 定义一个类继承Service
2. 在Manifest.xml文件中配置该service
3. 使用context的startService方法启动
4. 不再使用时，调用stopService方法停止该服务

* bindService

1. 创建bindservice服务端，继承自service并在类中，创建一个实现IBinder接口的实例对象并提供公共方法给客户端调用
2. 从onBind回调方法返回此Binder实例
3. 在客户端中，从onServiceConnected回调方法接受Binder，并私用提供的方法调用绑定服务。

4，IntentService面试详解

如果有一个任务，可以分成很多个子任务，需要按照顺序来完成，如果需要放到一个服务中完成，那么使用IntentService是最好的选择。

Service是运行在主线程当中的，所以在service里面编写耗时的操作代码，则会卡主线程会ANR。为了解决这样的问题，谷歌引入了IntentService

* 一种特殊的service，继承自service并且本身就是一个抽象类
* 内部通过HandlerThread和Handler实现异步操作，所以它可以做耗时的操作，内部适中也是通过handler异步
* 本质就是一个封装了HandlerThred和handler的异步框架
* 它创建了一个独立的工作线程来处理所有一个一个的intent，创建了一个工作队列来逐个发送intent给onHandlerIntent()
* 不需要主动调用stopSelf()来结束服务，因为源码里面自己实现了自动关闭
* 默认实现了onBind()返回的null，默认实现的onStartCommand()的目的是将intent插入到工作队列

总结：使用IntentService的好处有哪些。首先，省去了手动开线程的麻烦；第二，不用手动停止service；第三，由于设计了工作队列，可以启动多次---startService(),但是只有一个service实例和一个工作线程。一个一个熟悉怒执行。

### 5，Service的生命周期

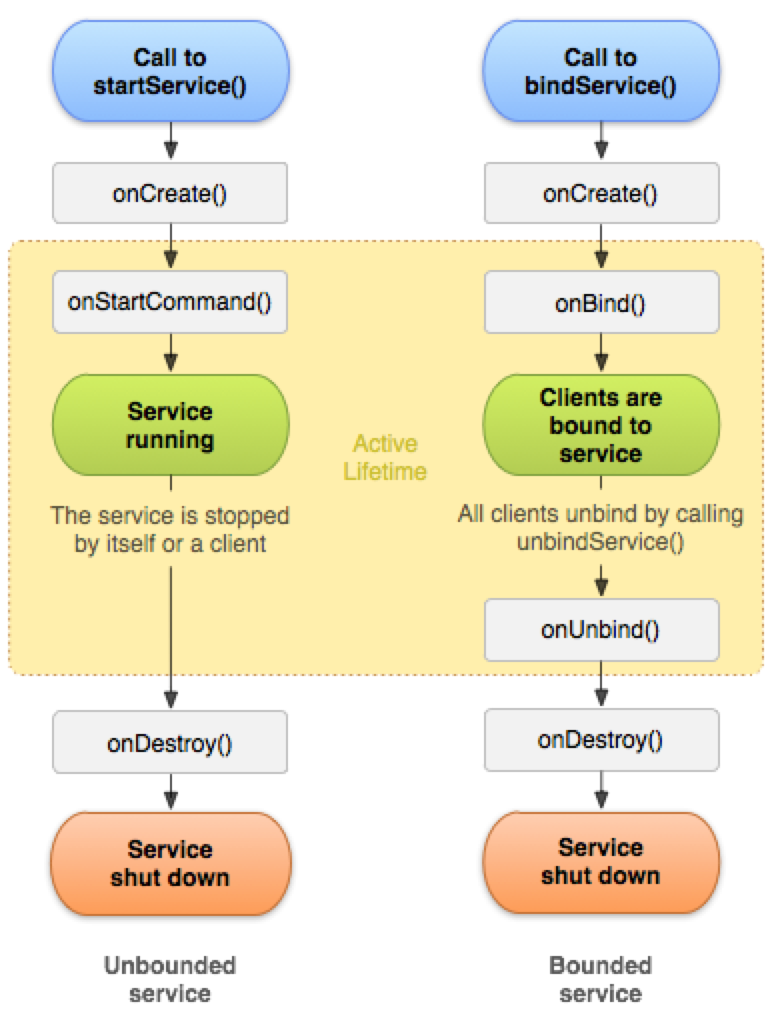


image.png

# BroadcastReceiver面试详解

### 1，广播定义（类似观察者模式）

BroadcastReceiver是一种全局监听器，用来实现系统中不同组件之间的通信。有时候也会用来作为传输少量而且发送频率低的数据，但是如果数据的发送频率比较高或者数量比较大就不建议用广播接收者来接收了，因为这样的效率很不好，因为BroadcastReceiver接收数据的开销还是比较大的。

### 2，使用场景

* 同一个app具有多个进程的不同组件之间的消息通讯
* 不同app之间的组件之间消息通讯

eg：

1）App全局监听：在AndroidManifest中静态注册的广播接收器，一般我们在收到该消息后，需要做一些相应的动作，而这些动作与当前App的组件，比如Activity或者Service的是否运行无关，比如我们在集成第三方Push SDK时，一般都会添加一个静态注册的BroadcastReceiver来监听Push消息，当有Push消息过来时，会在后台做一些网络请求或者发送通知等等。

2）组件局部监听：这种主要是在Activity或者Service中使用registerReceiver()动态注册的广播接收器，因为当我们收到一些特定的消息，比如网络连接发生变化时，我们可能需要在当前Activity页面给用户一些UI上的提示，或者将Service中的网络请求任务暂停。所以这种动态注册的广播接收器适合特定组件的特定消息处理。

### 3，广播的种类

* Normal Broadcast ：Context.sendBroadcast 普通的广播，完全异步，可以在同一时刻被所有接收者接收到，消息传递效率高且无法中断广播的传播
* System Broadcast : Context.sendOrderedBroadcast 有序广播； 发送有序广播后，广播接收者将按预先声明的优先级依次接收Broadcast。优先级高的优先接收到广播，而在其onReceiver()执行过程中，广播不会传播到下一个接收者，此时当前的广播接收者可以abortBroadcast()来终止广播继续向下传播，也可以将intent中的数据进行修改设置，然后将其传播到下一个广播接收者。 sendOrderedBroadcast(intent, null);//发送有序广播
* 粘性广播：sendStickyBroadcast()来发送该类型的广播信息，这种的广播的最大特点是，当粘性广播发送后，最后的一个粘性广播会滞留在操作系统中。如果在粘性广播发送后的一段时间里，如果有新的符合广播的动态注册的广播接收者注册，将会收到这个广播消息，虽然这个广播是在广播接收者注册之前发送的，另外一点，对于静态注册的广播接收者来说，这个等同于普通广播。
* Local Broadcast : 只在自身App内传播 本地广播

### 4，实现广播Receiver

静态注册：注册完成就一直运行，Menifest文件中 动态注册：跟随Activity的生命周期，必须要在onDestory中销毁，否则内存泄露，因为可以一直接收到

### 5，内部实现机制

自定义广播接受者，并复写onRecvice方法 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，将广播相应的消息循环队列中 消息循环拿到广播后，回调onReceive方法

### 6，本地广播LocalBroadcastManager详解

关于优势：

* 使用它发送的广播只能在自身的App内传播，不用担心泄露隐私数据
* 其他App无法对你的App发送该广播，不用担心安全漏洞
* 比系统全局广播高效，为甚？系统广播需要加入更多的广播池，加载流程多

为何它高效

* 主要因为它内部是通过Handler实现的，它的sendBroadcast方法就是通过handler发送一个message实现的。
* 系统广播是通过Binder实现比handler要复杂低效，用handler实现保障了其他应用就无法向本应用发广播，而本应用内发广播不会离开应用本身
* 内部协作靠2个Map集合，mReceivers和mActions，当然还有一个List集合mPendingBroadcast，主要就是存储待接收的广播对象

本地广播可以用来做什么？ 比如，做一个杀不死的服务-监听火的App，比如微信，友盟和极光的广播来启动自己

本地广播是不能用静态注册的；静态注册的目的--程序停止后也能监听

# Android中的动画

### 1，几种动画？

帧动画：指通过指定每一帧的图片和播放时间，有序的进行播放而形成动画效果，比如想听的律动条。

补间动画：指通过指定View的初始状态、变化时间、方式，通过一系列的算法去进行图形变换，从而形成动画效果，主要有Alpha、Scale、Translate、Rotate四种效果。注意：只是在视图层实现了动画效果，并没有真正改变View的属性，比如滑动列表，改变标题栏的透明度。

属性动画：在Android3.0的时候才支持，通过不断的改变View的属性，不断的重绘而形成动画效果。相比于视图动画，View的属性是真正改变了。比如view的旋转，放大，缩小。

### 2，Android 动画框架实现原理

传统的动画框架：View.startAnimation();

弊端：移动后不能点击。原因？跟实现机制有关系。

所有的透明度、旋转、平移、缩放动画，都是在view不断刷新调用draw的情况下实现的。

调用的canvas.translate(xxx),canvas.scaleX(xxx)…. Xxx:matrix像素矩阵来控制动画的数据。记得看源码，结合多只缩放的demo看源码。

### 3，属性动画实现原理

工作原理：在一定时间间隔内，通过不断对值进行改变，并不断将该值赋给对象的属性，从而实现该对象在该属性上的动画效果。

* ValueAnimator：通过不断控制值的变化（初始值->结束值），将值手动赋值给对象的属性，再不断调用View的invalidate()方法，去不断onDraw重绘view，达到动画的效果。

主要的三种方法：

1. ValueAnimator.ofInt(int values)：估值器是整型估值器IntEaluator
2. ValueAnimator.ofFloat(float values):估值器是浮点型估值器FloatEaluator

c) ValueAnimator.ofObject(ObjectEvaluator, start, end):将初始值以对象的形式过渡到结束值，通过操作对象实现动画效果，需要实现Interpolator接口，自定义估值器

估值器TypeEvalutor，设置动画如何从初始值过渡到结束值的逻辑。插值器(Interpolator）决定值的变化模式（匀速、加速等）;估值器（TypeEvalutor）决定值的具体变化数值。

// 自定义估值器，需要实现TypeEvaluator接口

public class ObjectEvaluator implements TypeEvaluator{     
// 复写evaluate（），在evaluate（）里写入对象动画过渡的逻辑       
@Override         
public Object evaluate(float fraction, Object startValue, Object endValue) {           // 参数说明           
 // fraction：表示动画完成度（根据它来计算当前动画的值）          
  // startValue、endValue：动画的初始值和结束值          
  ...   
 // 写入对象动画过渡的逻辑               
 return value;            
  // 返回对象动画过渡的逻辑计算后的值     
  } }

* ObjectAnimator:直接对对象的属性值进行改变操作，从而实现动画效果

ObjectAnimator继承自ValueAnimator类，底层的动画实现机制还是基本值的改变。它是不断控制值的变化，再不断自动赋给对象的属性，从而实现动画效果。这里的自动赋值，是通过调用对象属性的set/get方法进行自动赋值，属性动画初始值如果有就直接取，没有则调用属性的get()方法获取，当值更新变化时，通过属性的set()方法进行赋值。每次赋值都是调用view的postInvalidate()/invalidate()方法不断刷新视图（实际调用了onDraw()方法进行了重绘视图）。

//Object 需要操作的对象； propertyName 需要操作的对象的属性； values动画初始值&结束值， //如果是两个值，则从a->b值过渡，如果是三值，则从a->b->c

ObjectAnimator animator = ObjectAnimator.ofFloat(Object object, String propertyName, float ...values);

如果采用ObjectAnimator类实现动画，操作的对象的属性必须有get()和set()方法。

其他用法：

1）AnimatorSet组合动画

AnimatorSet.play(Animator anim)   ：播放当前动画   
AnimatorSet.after(long delay)   ：将现有动画延迟x毫秒后执行   
AnimatorSet.with(Animator anim)   ：将现有动画和传入的动画同时执行   
AnimatorSet.after(Animator anim)   ：将现有动画插入到传入的动画之后执行   
AnimatorSet.before(Animator anim) ：  将现有动画插入到传入的动画之前执行

1. ViewPropertyAnimator直接对属性操作，View.animate()返回的是一个ViewPropertyAnimator对象，之后的调用方法都是基于该对象的操作，调用每个方法返回值都是它自身的实例

 View.animate().alpha(0f).x(500).y(500).setDuration(500).setInterpolator()

3)设置动画监听

Animation.addListener(new AnimatorListener() {             
 @Override             
 public void onAnimationStart(Animation animation) {//动画开始时执行           }              
 @Override             
 public void onAnimationRepeat(Animation animation) {//动画重复时执行           }            
 @Override             
 public void onAnimationCancel()(Animation animation) {//动画取消时执行           }             
 @Override             
 public void onAnimationEnd(Animation animation) {//动画结束时执行           }        
 });

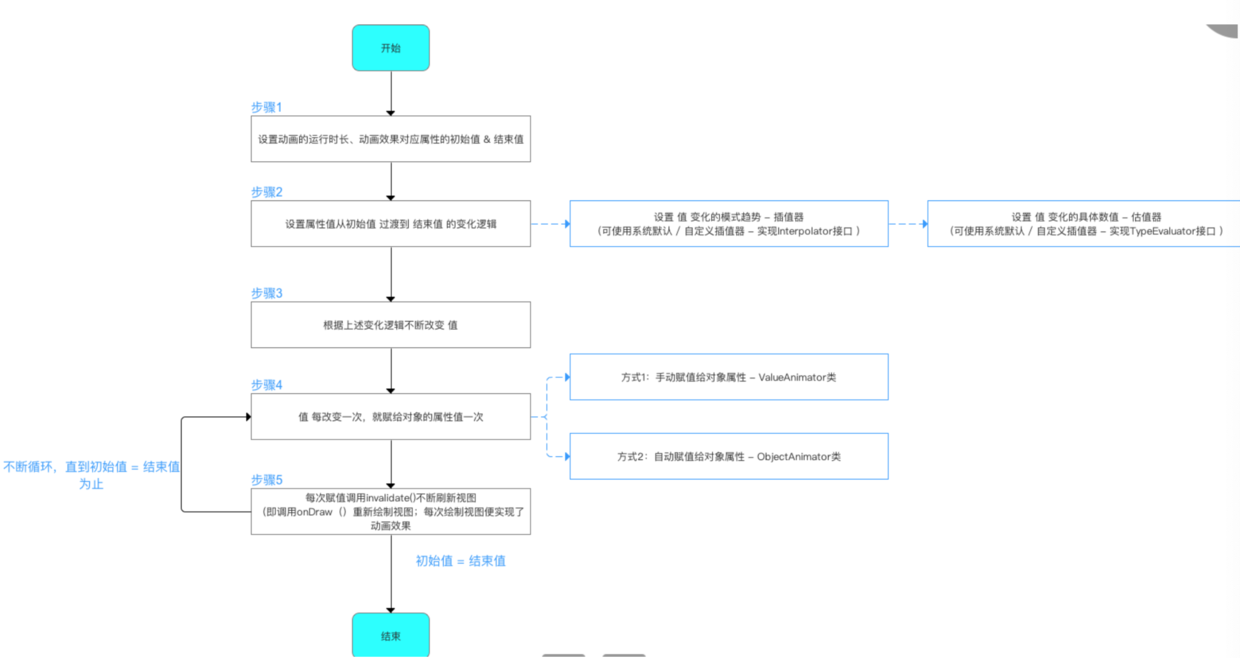
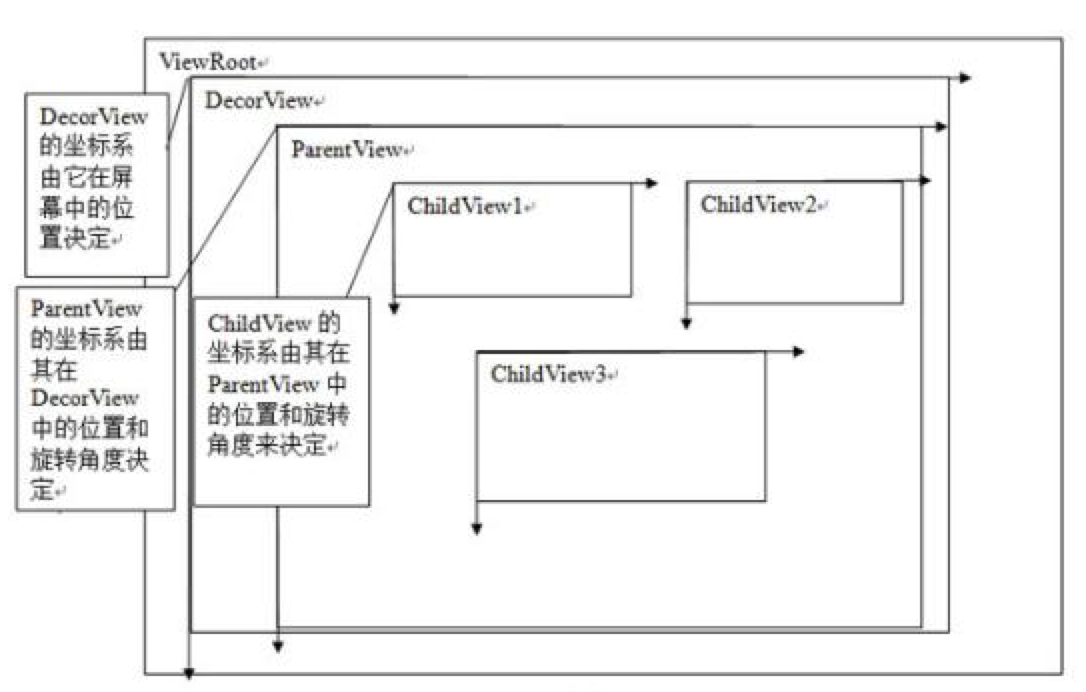


image.png

4，补间动画实现原理

主要有四种AlpahAnimation ScaleAnimation RotateAnimation TranslateAnimation四种，对透明度、缩放、旋转、位移四种动画。 在调用View.startAnimation时，先调用View.setAnimation(Animation)方法给自己设置一个Animation对象，再调用invalidate来重绘自己。在View.draw(Canvas, ViewGroup, long)方法中进行了getAnimation(), 并调用了drawAnimation(ViewGroup, long, Animation, boolean)方法，此方法调用Animation.getTranformation()方法，再调用applyTranformation()方法，该方法中主要是对Transformation.getMatrix().setTranslate/setRotate/setAlpha/setScale来设置相应的值，这个方法系统会以60FPS的频率进行调用。 具体是在调Animation.start()方法中会调用animationHandler.start()方法，从而调用了scheduleAnimation()方法，这里会调用mChoreographer.postCallback(Choregrapher.CALLBACK\_ANIMATION, this, null)放入事件队列中，等待doFrame()来消耗事件。

当一个 ChildView 要重画时，它会调用其成员函数 invalidate() 函数将通知其 ParentView 这个 ChildView 要重画，这个过程一直向上遍历到 ViewRoot，当 ViewRoot 收到这个通知后就会调用ViewRoot 中的 draw 函数从而完成绘制。View::onDraw() 有一个画布参数 Canvas, 画布顾名思义就是画东西的地方，Android 会为每一个 View 设置好画布，View 就可以调用 Canvas 的方法，比如：drawText, drawBitmap, drawPath 等等去画内容。每一个 ChildView 的画布是由其 ParentView 设置的，ParentView 根据 ChildView 在其内部的布局来调整 Canvas，其中画布的属性之一就是定义和 ChildView 相关的坐标系，默认是横轴为 X 轴，从左至右，值逐渐增大，竖轴为 Y 轴，从上至下，值逐渐增大。 

Android 补间动画就是通过 ParentView 来不断调整 ChildView 的画布坐标系来实现的，在ParentView的dispatchDraw方法会被调用。

dispatchDraw() {   
 ....   
 Animation a = ChildView.getAnimation()   
 Transformation tm = a.getTransformation();   
 Use tm to set ChildView's Canvas; Invalidate();   
 ....   
}

这里有两个类：Animation 和 Transformation，这两个类是实现动画的主要的类，Animation 中主要定义了动画的一些属性比如开始时间、持续时间、是否重复播放等，这个类主要有两个重要的函数：getTransformation 和 applyTransformation，在 getTransformation 中 Animation 会根据动画的属性来产生一系列的差值点，然后将这些差值点传给 applyTransformation，这个函数将根据这些点来生成不同的 Transformation，Transformation 中包含一个矩阵和 alpha 值，矩阵是用来做平移、旋转和缩放动画的，而 alpha 值是用来做 alpha 动画的（简单理解的话，alpha 动画相当于不断变换透明度或颜色来实现动画），调用 dispatchDraw 时会调用 getTransformation 来得到当前的 Transformation。某一个 View 的动画的绘制并不是由他自己完成的而是由它的父 view 完成。

1）补间动画TranslateAnimation,View位置移动了，可是点击区域还在原来的位置，为什么？

View在做动画是，根据动画时间的插值，计算出一个Matrix，不停的invalidate，在onDraw中的Canvas上使用这个计算出来的Matrix去draw view的内容。某个view的动画绘制并不是由它自己完成，而是由它的父view完成，使它的父view画布进行了移动，而点击时还是点击原来的画布。使得它看起来变化了。

# 数据库面试相关

### 1，数据库的操作类有那些，如何导入外部数据库？

读懂题目。如果碰到问题比较模糊的时候可以适当问问面试官。

配合面试官来面试：面试是一个相互了解的过程，要充分利用面试的题目和时间把自己的能力和技术展现出来，面试官能够看到你的真实技术。

1） 使用数据库的方式有哪些？

（1） openOrCreateDatabase(String path);

（2） 继承SqliteOpenHelper类对数据库及其版本进行管理(onCreate,onUpgrade)

当在程序当中调用这个类的方法getWritableDatabase()或者getReadableDatabase();的时候才会打开数据库。如果当时没有数据库文件的时候，系统就会自动生成一个数据库。

2） 操作的类型：增删改查CRUD

直接操作SQL语句：SQliteDatabase.execSQL(sql);

面向对象的操作方式：SQLiteDatabase.insert(table, nullColumnHack, ContentValues);

如何导入外部数据库？

一般外部数据库文件可能放在SD卡或者res/raw或者assets目录下面。

写一个DBManager的类来管理，数据库文件搬家，先把数据库文件复制到”/data/data/包名/databases/”目录下面，然后通过db.openOrCreateDatabase(db文件),打开数据库使用。

我上一个项目就是这么做的，由于app上架之前就有一些初始数据需要内置，也会碰到数据的升级等问题，我是这么做的…… 同时我碰到最有意思的问题就是关于数据库并发操作的问题，比如：多线程操作数据库的时候，我采取的是封装使用互斥锁来解决……

# Webview面试详解

### 1，常见的坑

* android api 16以及以前的版本存在远程代码执行安全漏洞，主要是因为程序没有正确的限制使用Webview.addJavascriptInterface方法，

远程攻击者可以通过使用Java Reflection Api利用该漏洞执行任意Java对象的方法，16后面的版本改了之后申明@JavascriptInterace注解就可以了

* Webview在布局文件中的使用，webvew卸载其他容器中时
* JsBridge ，调用原生相互通讯
* webviewClient.onPageFinished（会跳转其他url的时候不断调用） -> WebChromeClient.onProgressChanged(优先调用)
* 后台耗电（需要在onDestory里把webview 销毁掉）
* Webview硬件加速导致页面渲染问题；只能通过暂时关闭加速

### 2，内存泄露问题

* 独立进程，涉及到了进程间的通讯（推荐）
* 动态添加webview，对传入的webview中使用的Context使用弱引用，动态添加webview意思在布局创建个ViewGroup用来放置Webview，Activity创建时add进来，在Activity停止时remove掉

# Binder面试详解

### 1，Binder机制的简单理解

在Android系统的Binder机制中，是有Client,Service,ServiceManager,Binder驱动程序组成的，其中Client，service，Service Manager运行在用户空间，Binder驱动程序是运行在内核空间的。而Binder就是把这4种组件粘合在一块的粘合剂，其中核心的组件就是Binder驱动程序，Service Manager提供辅助管理的功能，而Client和Service正是在Binder驱动程序和Service Manager提供的基础设施上实现C/S 之间的通信。其中Binder驱动程序提供设备文件/dev/binder与用户控件进行交互，

  Client、Service，Service Manager通过open和ioctl文件操作相应的方法与Binder驱动程序进行通信。而Client和Service之间的进程间通信是通过Binder驱动程序间接实现的。而Binder Manager是一个守护进程，用来管理Service，并向Client提供查询Service接口的能力。

### 2，为什么使用Binder（已有的跨进程都不适合Android）

* Android使用的Linux内核拥有非常多的跨进程通讯机制（管道，消息队列，信号，信号量，共享内存，socket ）

扩展：<https://blog.csdn.net/qq_26626709/article/details/52206067>）

* 性能 （相对Socket更高效）
* 安全（跟socket的只一个url就链接的不安全相比，binder支持双方做调用时身份校验）

### 3，binder通信模型

通讯录，serviceManager，binder驱动（类似电话基站）

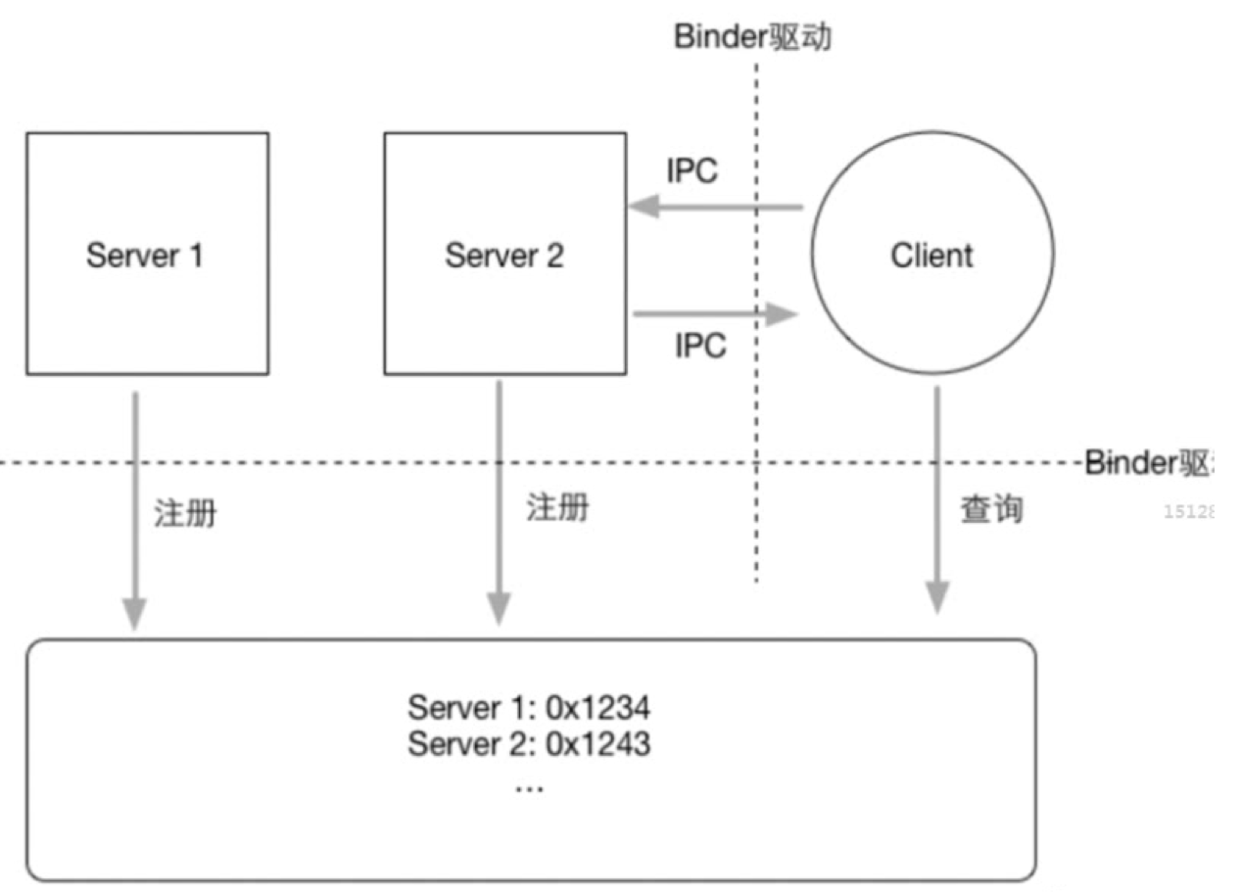


image.png

怎么调用？（SM = ServiceManager）

先去serviceManager注册一个方法，然后通过binder驱动来转换成代理对象返回给客户端完成注册。

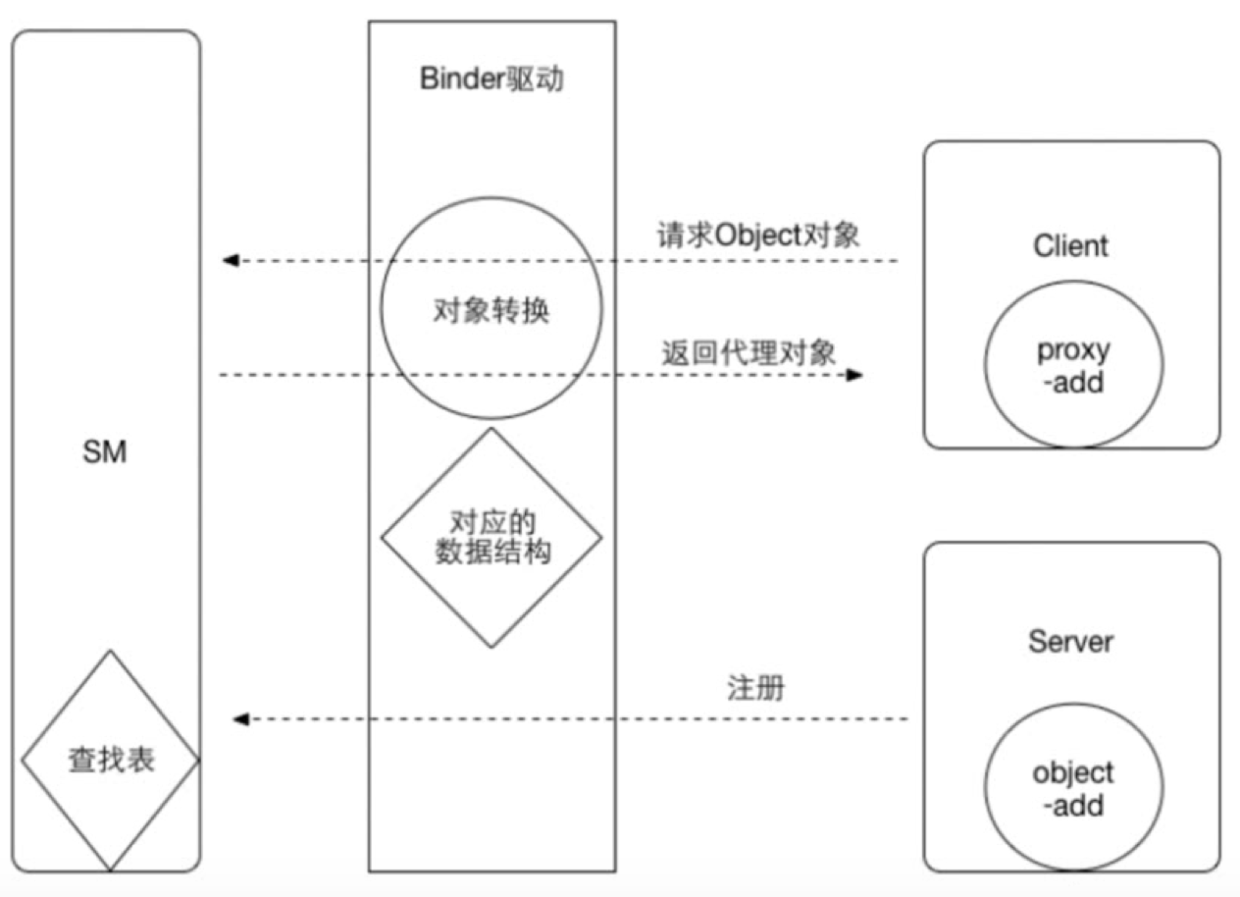


image.png

### 4，到底什么是Binder？

通常意义下，Binder指的是一种通信机制

对于Server进程来说，Binder指的是Binder本地对象

对于Client来说，Binder指的是Binder代理对象

对于传输过程而言，Binder是可以进行垮进程传递的对象

Android中实现IBinder这个接口就可以垮进程使用

### 5，AIDL解决了什么问题？

AIDL的全称：Android Interface Definition Language，安卓接口定义语言。

由于Android系统中的进程之间不能共享内存，所以需要提供一些机制在不同的进程之间进行数据通信。

远程过程调用：RPC—Remote Procedure Call。 安卓就是提供了一种IDL的解决方案来公开自己的服务接口。

AIDL:可以理解为双方的一个协议合同。双方都要持有这份协议---文本协议 xxx.aidl文件

（安卓内部编译的时候会将aidl协议翻译生成一个xxx.java文件---代理模式：Binder驱动有关的，Linux底层通讯有关的。）

在系统源码里面有大量用到aidl，比如系统服务。电视机顶盒系统开发。你的服务要暴露给别的开发者来使用。

### 6，Android开发中何时使用多进程？使用多进程的好处是什么？

即时通讯或者社交应用，webview可以单独给进程

在启动一个不可见的轻量级私有进程，在后台收发消息，或者做一些耗时的事情，或者开机启动这个进程，然后做监听等。还有就是防止主进程被杀守护进程，守护进程和主进程之间相互监视，有一方被杀就重新启动它。

### 7，Android中进程间通信有那些实现方式？

Intent，Binder（AIDL），Messenger，BroadcastReceiver

### 8，binder的内存拷贝过程

相比其他的IPC通信，比如消息机制、共享内存、管道、信号量等，Binder仅需一次内存拷贝，即可让目标进程读取到更新数据，同共享内存一样相当高效，其他的IPC通信机制大多需要2次内存拷贝。Binder内存拷贝的原理为：进程A为Binder客户端，在IPC调用前，需将其用户空间的数据拷贝到Binder驱动的内核空间，由于进程B在打开Binder设备(/dev/binder)时，已将Binder驱动的内核空间映射(mmap)到自己的进程空间，所以进程B可以直接看到Binder驱动内核空间的内容改动

# Handler和AsyncTask面试详解

### 1，什么是Handler？

Handler是线程的消息通讯的桥梁，主要用来发送消息及处理消息。

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

* 可以让对应的Message和Runnable在未来的某个时间点进行相应处理
* 让自己想要处理的耗时操作放在子线程，让更新UI的操作放在主线程

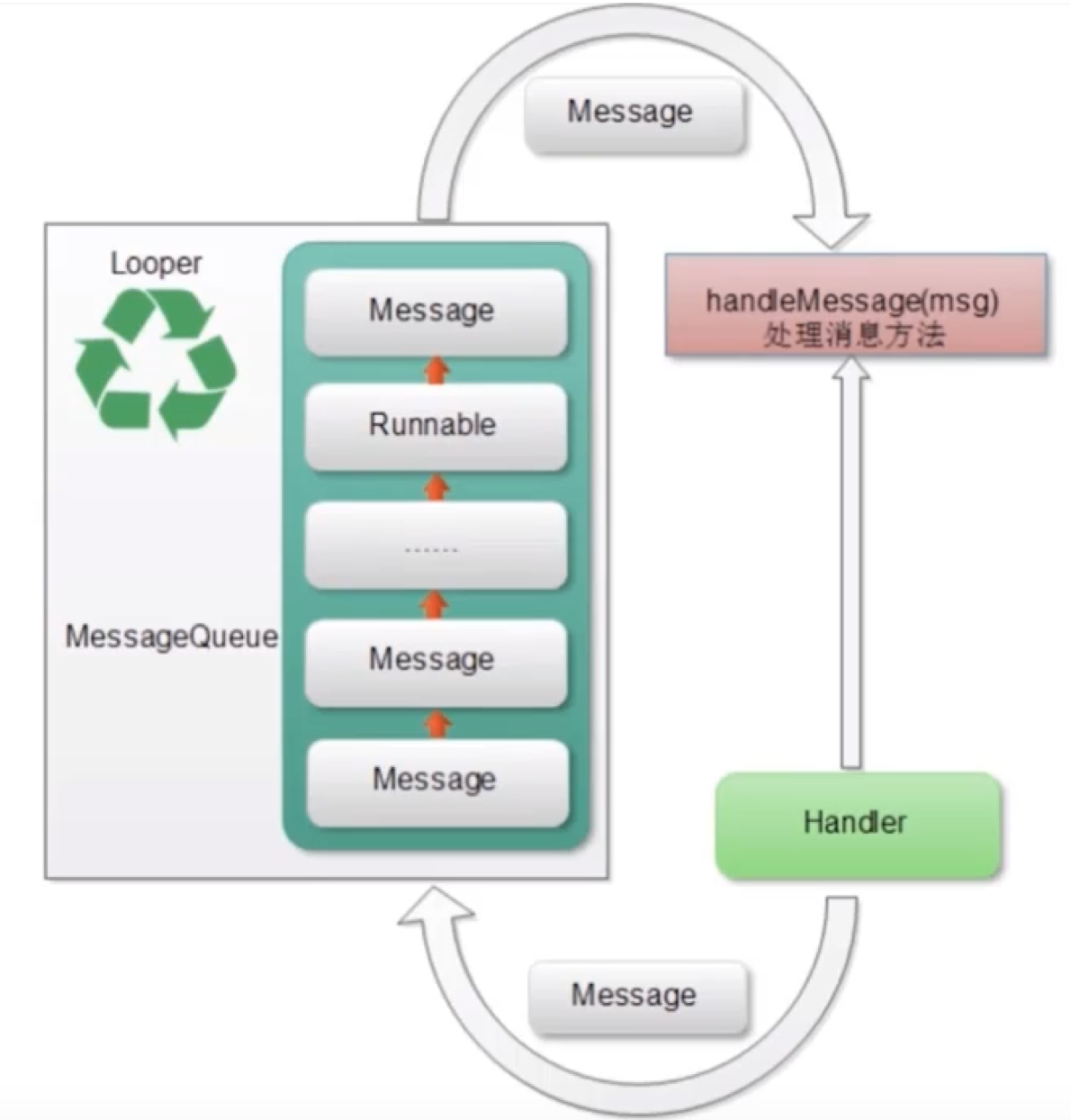


image.png

### 2，Handler的使用方法

* post（Runnable）其实底层调用的还是sendMessage，主要是系统自己封装了
* sendMessage（message）

### 3，Handler机制原理

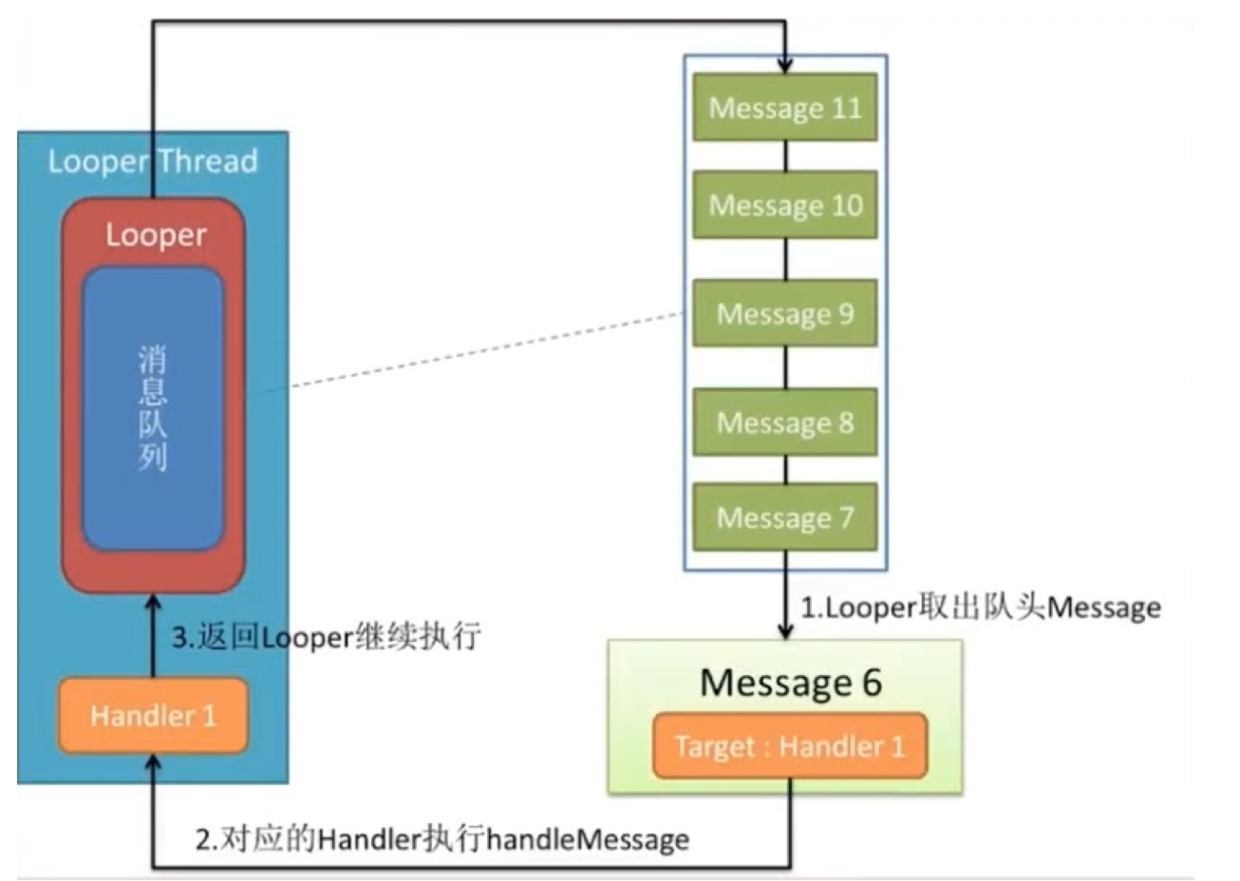


image.png

handler，它的作用就是实现线程之间的通信。

handler整个流程中，主要有四个对象，handler，Message,MessageQueue,Looper。当应用创建的时候，就会在主线程中创建handler对象，

我们通过要传送的消息保存到Message中，handler通过调用sendMessage方法将Message发送到MessageQueue中，Looper对象就会不断的调用loop()方法

不断的从MessageQueue中取出Message交给handler进行处理。从而实现线程之间的通信。

### 4，解决Handler的内存泄露

原因：静态内部类持有外部类的匿名引用，导致外部activity无法释放

解决：handler内部持有外部activity的弱引用，并把handler改为静态内部类，mHandler.removeCallback

### 5，什么是AsyncTask

本质上就是一个封装了线程池和handler的异步框架，无法做耗时操作

怎么使用：三个参数（），5个方法（onPreExecute前期操作，doInBackground，onPostExecute）

### 6，AsyncTask的机制原理

它本质上是一个静态的线程池，它派生出来的子类可以实现不同的异步任务，这些任务都是提交到静态的线程池中执行。

线程池中的工作线程执行doInBackground（mParams）方法执行异步任务

当任务状态改变之后，工作线程会向UI线程发送消息，AsyncTask内部的InternalHandler想要这些消息，并调用相关的回调函数

### 7，AsyncTask的注意事项

内存泄露：跟handler一样

生命周期：不调用cancel，无法销毁，可能导致崩溃

### 8，HandlerThread面试详解，背景，它是什么，有什么优缺点？

开启Thread子线程进行耗时操作，多次创建和销毁线程是很耗系统资源的，为了解决这个问题，google搞了一个HandlerThread；

它就是一个线程类，继承了Thread，有内部Looper对象，可以进行looper循环，通过获取HandlerThread的looper对象传递给Handler对象，

可以在handlerMessage方法中执行异步任务。

* 优点：不会有堵塞，减少了性能的消耗
* 缺陷：不能同时进行多任务的处理，需要等待，处理效率低

与线程池注重并发不同，HandlerThread是一个串行队列，HandlerThread背后只有一个线程。

### 9，HandlerThread源码解析

### 10，Handler、 Thread 和 HandlerThread 的差别

# View 的绘制与分发，自定义等

### 1，view树的绘制流程

measure（ 测量大小）-> layout（安置位置）-> draw （绘制）自上而下进行遍历

### 2，measure的重要方法

View的measure过程由ViewGroup传递而来，在调用View.measure方法之前，会首先根据View自身的LayoutParams和父布局的MeasureSpec确定子view的MeasureSpec，然后将view宽高对应的measureSpec传递到measure方法中，那么子view的MeasureSpec获取规则是怎样的？分几种情况进行说明

* 父布局是EXACTLY模式：

    a.子view宽或高是个确定值，那么子view的size就是这个确定值，mode是EXACTLY（是不是说子view宽高可以超过父view？见下一个）

    b.子view宽或高设置为match\_parent,那么子view的size就是占满父容器剩余空间，模式就是EXACTLY

    c.子view宽或高设置为wrap\_content,那么子view的size就是占满父容器剩余空间，不能超过父容器大小，模式就是AT\_MOST

* 父布局是AT\_MOST模式：

a.子view宽或高是个确定值，那么子view的size就是这个确定值，mode是EXACTLY

   b.子view宽或高设置为match\_parent,那么子view的size就是占满父容器剩余空间,不能超过父容器大小，模式就是AT\_MOST

   c.子view宽或高设置为wrap\_content,那么子view的size就是占满父容器剩余空间，不能超过父容器大小，模式就是AT\_MOST

* 父布局是UNSPECIFIED模式：

a.子view宽或高是个确定值，那么子view的size就是这个确定值，mode是EXACTLY

b.子view宽或高设置为match\_parent,那么子view的size就是0，模式就是UNSPECIFIED

c.子view宽或高设置为wrap\_content,那么子view的size就是0，模式就是UNSPECIFIED

获取到宽高的MeasureSpec后，传入view的measure方法中来确定view的宽高，这个时候还要分情况

1.当MeasureSpec的mode是UNSPECIFIED,此时view的宽或者高要看view有没有设置背景，如果没有设置背景，就返回设置的minWidth或minHeight,这两个值如果没有设置默认就是0，如果view设置了背景，就取minWidth或minHeight和背景这个drawable固有宽或者高中的最大值返回

2.当MeasureSpec的mode是AT\_MOST和EXACTLY，此时view的宽高都返回从MeasureSpec中获取到的size值，这个值的确定见上边的分析。因此如果要通过继承view实现自定义view，一定要重写onMeasure方法对wrap\_conten属性做处理，否则，他的match\_parent和wrap\_content属性效果就是一样的

measure（ViewGroup.LayoutParams，MeasureSpec测量规格，如何测量，三种模式（父容器，固定大小，子容器））

* UNSPECIFIED：不对View大小做限制，如：ListView，ScrollView
* EXACTLY：确切的大小，如：100dp或者march\_parent
* AT\_MOST：大小不可超过某数值，如：wrap\_content

onMeasure（2个参数，宽和高）

setMeasuredDimension（设置测量的大小，一定会调用，宽和高）

### 3，layout方法

view中的onLayout方法是抽象方法，viewgroup继承view必须实现onLayout方法

layout方法的作用是用来确定view本身的位置，onLayout方法用来确定所有子元素的位置，当ViewGroup的位置确定之后，它在onLayout中会遍历所有的子元素并调用其layout方法，在子元素的layout方法中onLayout方法又会被调用。layout方法的流程是，首先通过setFrame方法确定view四个顶点的位置，然后view在父容器中的位置也就确定了，接着会调用onLayout方法，确定子元素的位置，onLayout是个空方法，需要继承者去实现。

getMeasuredHeight和getHeight方法有什么区别？getMeasuredHeight（测量高度）形成于view的measure过程，getHeight（最终高度）形成于layout过程，在有些情况下，view需要measure多次才能确定测量宽高，在前几次的测量过程中，得出的测量宽高有可能和最终宽高不一致，但是最终来说，还是会相同，有一种情况会导致两者值不一样，如下，此代码会导致view的最终宽高比测量宽高大100px

public void layout(int l,int t,int r, int b){  
 super.layout(l,t,r+100,b+100);  
}

### 4，draw

View的绘制过程遵循如下几步：

a.绘制背景 background.draw(canvas) b.绘制自己（onDraw） c.绘制children（dispatchDraw） d.绘制装饰（onDrawScrollBars）

View绘制过程的传递是通过dispatchDraw来实现的，它会遍历所有的子元素的draw方法，如此draw事件就一层一层的传递下去了

ps：view有一个特殊的方法setWillNotDraw，如果一个view不需要绘制内容，即不需要重写onDraw方法绘制，可以开启这个标记，系统会进行相应的优化。默认情况下，View没有开启这个标记，默认认为需要实现onDraw方法绘制，当我们继承ViewGroup实现自定义控件，并且明确知道不需要具备绘制功能时，可以开启这个标记，如果我们重写了onDraw,那么要显示的关闭这个标记

子view宽高可以超过父view？能

1.android:clipChildren = "false" 这个属性要设置在父 view 上。代表其中的子View 可以超出屏幕。

2.子view 要有具体的大小，一定要比父view 大 才能超出。比如 父view 高度 100px 子view 设置高度150px。子view 比父view大，这样超出的属性才有意义。（高度可以在代码中动态赋值，但不能用wrap\_content / match\_partent）。

3.对父布局还有要求，要求使用linearLayout(反正我用RelativeLayout 是不行)。你如果必须用其他布局可以在需要超出的view上面套一个linearLayout 外面再套其他的布局。

4.最外面的布局如果设置的padding 不能超出

### 5，自定义view需要注意的几点

1.让view支持wrap\_content属性，在onMeasure方法中针对AT\_MOST模式做专门处理，否则wrap\_content会和match\_parent效果一样（继承ViewGroup也同样要在onMeasure中做这个判断处理）

if(widthMeasureSpec == MeasureSpec.AT\_MOST && heightMeasureSpec == MeasureSpec.AT\_MOST){     
 setMeasuredDimension(200,200); // wrap\_content情况下要设置一个默认值，最终的值需要计算得到刚好包裹内容的宽高值   
}else if(widthMeasureSpec == MeasureSpec.AT\_MOST){       
 setMeasuredDimension(200,heightMeasureSpec );   
}else if(heightMeasureSpec == MeasureSpec.AT\_MOST){       
 setMeasuredDimension(heightMeasureSpec ,200);   
}

2.让view支持padding（onDraw的时候，宽高减去padding值，margin由父布局控制，不需要view考虑），自定义ViewGroup需要考虑自身的padding和子view的margin造成的影响

3.在view中尽量不要使用handler，使用view本身的post方法 4.在onDetachedFromWindow中及时停止线程或动画 5.view带有滑动嵌套情形时，处理好滑动冲突

**扩展：优化自定义view** 减少在onDraw里面大量计算和对象创建，大量内存分配 尽量少用invalidate()次数 view里面耗时的操作layout，减少requestLayout()避免让UI系统重新遍历整棵树 如果有一个很复杂的布局，不如将这个复杂的布局直接使用自己写的viewgroup来实现，减少了一个树的层次关系，全部都是自己测量和layout，打到优化的目的，facebook经常这么搞

### 6，介绍下实现一个自定义view的基本流程

1、自定义View的属性 编写attr.xml文件 2、在layout布局文件中引用，同时引用命名空间 3、在View的构造方法中获得我们自定义的属性 ，在自定义控件中进行读取（构造方法拿到attr.xml文件值） 4、重写onMesure 5、重写onDraw

### 7，View的事件分发机制

1，为什么有事件分发机制？ view是树形结构的，view会重叠，点击会无法判断，所以需要事件分发来统一调度

2，三个重要的事件分发方法？

dispatchTouchEvent onInterceptTouchEvent onTouchEvent

### 3，事件分发流程

Activity->PhoneWindow->DecorView->ViewGroup->View

如果事件没有被拦截，就抛给Activity

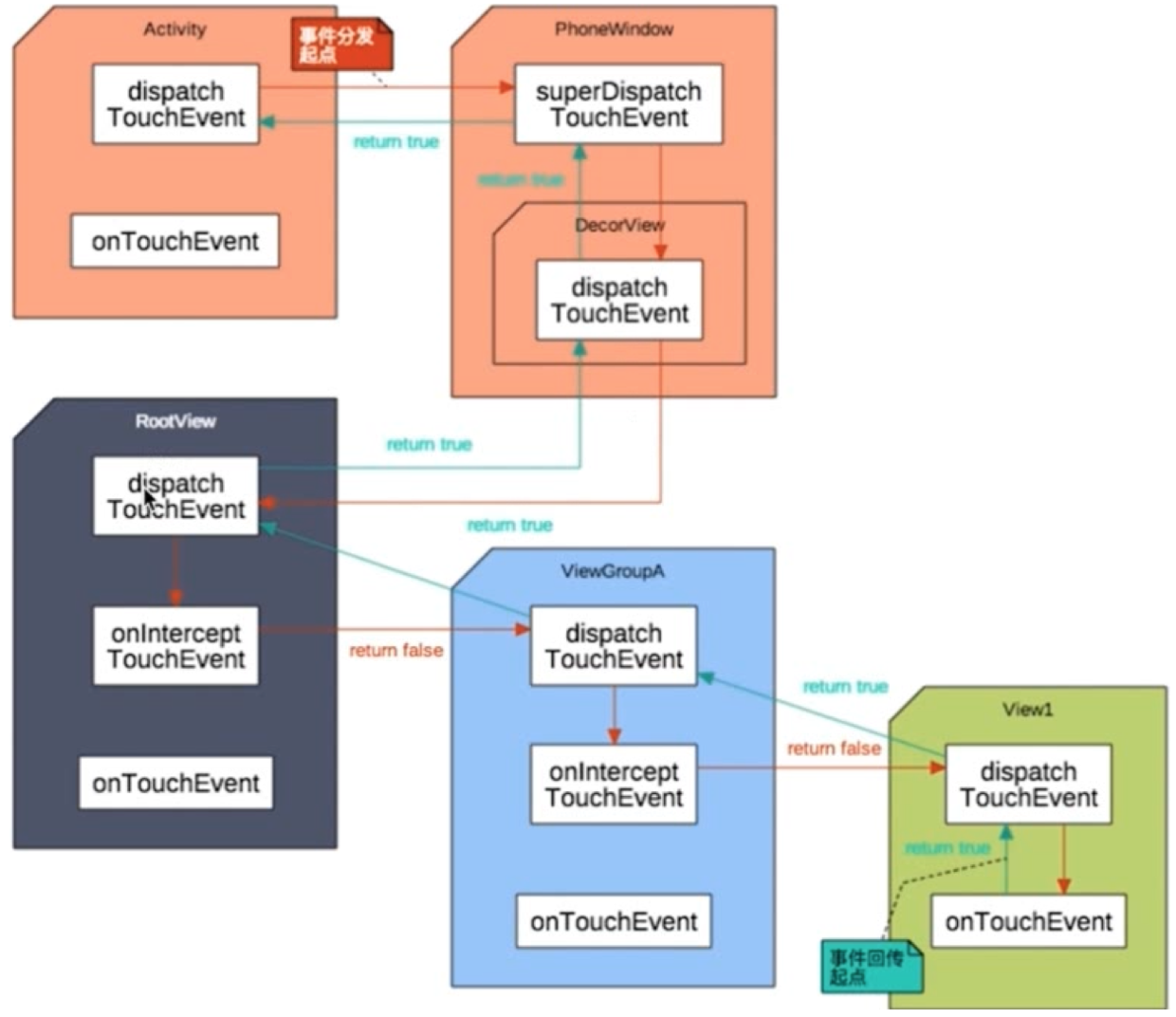


image.png

1、Touch事件传递的相关API有dispatchTouchEvent、onTouchEvent、onInterceptTouchEvent  
2、Touch事件相关的类有View、ViewGroup、Activity  
3、Touch事件会被封装成MotionEvent对象，该对象封装了手势按下、移动、松开等动作  
4、Touch事件通常从Activity#dispatchTouchEvent发出，只要没有被消费，会一直往下传递，到最底层的View。  
5、如果Touch事件传递到的每个View都不消费事件，那么Touch事件会反向向上传递,最终交由Activity#onTouchEvent处理.  
6、onInterceptTouchEvent为ViewGroup特有，可以拦截事件.  
7、Down事件到来时，如果一个View没有消费该事件，那么后续的MOVE/UP事件都不会再给它

### 8，View的加载流程

View随着Activity的创建而加载，startActivity启动一个Activity时，在ActivityThread的handleLaunchActivity方法中会执行Activity的onCreate方法，这个时候会调用setContentView加载布局创建出DecorView并将我们的layout加载到DecorView中，当执行到handleResumeActivity时，Activity的onResume方法被调用，然后WindowManager会将DecorView设置给ViewRootImpl,这样，DecorView就被加载到Window中了，此时界面还没有显示出来，还需要经过View的measure，layout和draw方法，才能完成View的工作流程。我们需要知道View的绘制是由ViewRoot来负责的，每一个DecorView都有一个与之关联的ViewRoot,这种关联关系是由WindowManager维护的，将DecorView和ViewRoot关联之后，ViewRootImpl的requestLayout会被调用以完成初步布局，通过scheduleTraversals方法向主线程发送消息请求遍历，最终调用ViewRootImpl的performTraversals方法，这个方法会执行View的measure layout 和draw流程

### 9，View的滑动方式

a.layout(left,top,right,bottom):通过修改View四个方向的属性值来修改View的坐标，从而滑动View

b.offsetLeftAndRight() offsetTopAndBottom():指定偏移量滑动view

c.LayoutParams,改变布局参数：layoutParams中保存了view的布局参数，可以通过修改布局参数的方式滑动view

d.通过动画来移动view：注意安卓的平移动画不能改变view的位置参数，属性动画可以

e.scrollTo/scrollBy:注意移动的是view的内容，scrollBy(50,50)你会看到屏幕上的内容向屏幕的左上角移动了，这是参考对象不同导致的，你可以看作是它移动的是手机屏幕，手机屏幕向右下角移动，那么屏幕上的内容就像左上角移动了

f.scroller:scroller需要配置computeScroll方法实现view的滑动，scroller本身并不会滑动view，它的作用可以看作一个插值器，它会计算当前时间点view应该滑动到的距离，然后view不断的重绘，不断的调用computeScroll方法，这个方法是个空方法，所以我们重写这个方法，在这个方法中不断的从scroller中获取当前view的位置，调用scrollTo方法实现滑动的效果

### 10. Requestlayout， onlayout， onDraw， DrawChild 区别与联系

RequestLayout()方法：会导致调用Measure()方法和layout()。将会根据标志位判断是否需要onDraw();

说明：只是对View树重新布局layout过程包括measure()和layout()过程，如果view的l,t,r,b没有必变，那就不会触发onDraw；但是如果这次刷新是在动画 里，mDirty非空，就会导致onDraw。

onLayout()：摆放viewGroup里面的子控件 onDraw()：绘制视图本身；（每个View都需要重载该方法，ViewGroup不需要实现该方法） drawChild(): 重新回调每一个子视图的draw方法。child.draw(canvas, this, drawingTime);

### 11. invalidate()和 postInvalidate() 的区别及使用

View.invalidate():层层上传到父级，直到传递到ViewRootImpl后出发了scheduleTraversals(),然后整个View树开始重新按照View绘制流程进行重绘任务。 invalidate()：在UI主线程当中刷新View； postInvalidate()：在子线程当中刷新View；其实最终调用的就是invalidate，原理依然是通过工作线程向主线程发送消息这一机制。

View.postInvalidate最终会调用ViewRootImpl.dispatchInvalidateDelayed()

public void postInvalidate() {           
 postInvalidateDelayed(0);       
}       
public void postInvalidateDelayed(long delayMilliseconds) {         
  // We try only with the AttachInfo because there's no point in invalidating         // if we are not attached to our window           
 final AttachInfo attachInfo = mAttachInfo;          
  if (attachInfo != null) {               
 attachInfo.mViewRootImpl.dispatchInvalidateDelayed(this,delayMilliseconds);          
  }       
}       
public void dispatchInvalidateDelayed(View view, long delayMilliseconds) {           
 Message msg = mHandler.obtainMessage(MSG\_INVALIDATE, view);          
 mHandler.sendMessageDelayed(msg, delayMilliseconds);   
//这里的mHandler是ViewRootHandler实例，在该Handler的handleMessage方法中调用了view.invalidate()方法。       
}  
 public void handleMessage(Message msg) {               
 switch (msg.what) {               
 case MSG\_INVALIDATE:                 
 ((View) msg.obj).invalidate();           
          break;   
 }  
 }

### 12，LinearLayout对比RelativeLayout

性能对比：LinearLayout的性能要比RelativeLayout好。

因为RelativeLayout会测量两次。而默认情况下（没有设置weight）LinearLayout只会测量一次。

为什么RelativeLayout会测量两次？首先RelativeLayout中的子view排列方式是基于彼此依赖的关系，而这个依赖可能和布局中view的顺序无关，在确定每一个子view的位置的时候，就需要先给每一个子view排一下序。又因为RelativeLayout允许横向和纵向相互依赖，所以需要横向纵向分别进行一次排序测量。

### 13，RecyclerView在很多方面能取代ListView，Google为什么没把ListView划上一条过时的横线？

ListView采用的是RecyclerBin的回收机制在一些轻量级的List显示时效率更高。

### 14， RecyclerView 的缓存机制

四级缓存，主要三个类（Recycler，RecycledViewPool和ViewCacheExtension） Recycler：管理已经废弃或者与RecyclerView分离的ViewHolder，里面有2个重要的成员 mAttachedScrap mChangedScrap

RecyclerView的四级缓存包括：

一级缓存：mAttachedScrap、mChangedScrap 二级缓存：mCacheViews 三级缓存：mViewCacheExtension 四级缓存：mRecyclerPool

一级缓存为屏幕内缓存，二级缓存为屏幕外缓存，三级缓存为自定义缓存，四级缓存为缓存池缓存。一二三即缓存直接不需要重新绑定View，四级缓存需要绑定Holder设置数据。禁用缓存可以使用ViewHolder的setIsRecyclable方法。

参考：https://blog.csdn.net/yoonerloop/article/details/84727902

### 15，SurfaceView，它是什么？他的继承方式是什么？他与View的区别(从源码角度，如加载，绘制等)。

SurfaceView中采用了双缓冲机制，保证了UI界面的流畅性，同时SurfaceView不在主线程中绘制，而是另开辟一个线程去绘制，所以它不妨碍UI线程；

SurfaceView继承于View，他和View主要有以下三点区别：

（1）View底层没有双缓冲机制，SurfaceView有； （2）view主要适用于主动更新，而SurfaceView适用与被动的更新，如频繁的刷新 （3）view会在主线程中去更新UI，而SurfaceView则在子线程中刷新；

SurfaceView的内容不在应用窗口上，所以不能使用变换（平移、缩放、旋转等）。也难以放在ListView或者ScrollView中，不能使用UI控件的一些特性比如View.setAlpha()

View：显示视图，内置画布，提供图形绘制函数、触屏事件、按键事件函数等；必须在UI主线程内更新画面，速度较慢。

SurfaceView：基于view视图进行拓展的视图类，更适合2D游戏的开发；是view的子类，类似使用双缓机制，在新的线程中更新画面所以刷新界面速度比view快，Camera预览界面使用SurfaceView。

GLSurfaceView：基于SurfaceView视图再次进行拓展的视图类，专用于3D游戏开发的视图；是SurfaceView的子类，openGL专用。

# Android的构建

### 1，android的构建流程

Java的文件编译成class 字节码文件，然后把字节码加依赖的第三方jar文件一起打包成classes.dex 安卓devilk虚拟机可执行的文件

再打包资源文件，再把dex文件和save文件合并成未签名的包，然后签名打包成一个完整的包

### 2，jenkins持续集成构建

……待续 #MVC/MVP/MVVM

1,MVC定义，acitivity又充当了View和Controller

# Android 插件化

### 1，由来

65536/64k 方法超过限制

### 2，解决的问题

动态加载APK，资源加载，代码加载

# Android热更新

### 1，热更新流程

1. 线上检查到严重的crash
2. 拉出bugfix分支并分支上修复问题
3. jenkins构建和补丁生成
4. app通过推送或者主动拉取补丁文件
5. 将bugfix代码合到主分支上

### 2，更新框架介绍：

Dexposed AndFix Nuwa

### 3，原理

java的加载器：PathClassLoader，DexClassLoader 机制：ClassLoader遍历dexElements这个数组， 我们知道Java虚拟机 —— JVM 是加载类的class文件的，而Android虚拟机——Dalvik/ART VM 是加载类的dex文件，  而他们加载类的时候都需要ClassLoader,ClassLoader有一个子类BaseDexClassLoader，而BaseDexClassLoader下有一个 数组——DexPathList，是用来存放dex文件，当BaseDexClassLoader通过调用findClass方法时，实际上就是遍历数组， 找到相应的dex文件，找到，则直接将它return。

而热修复的解决方法就是将新的dex添加到该集合中，并且是在旧的dex的前面，所以就会优先被取出来并且return返回。

# 进程保活

### 1，Android进程的优先级

Foreground Process 前台进程 Visible process 可见进程 Service process 服务进程 Background process 后台进程 Empty process 空进程

### 2，Android进程的回收策略

Low memory killer：通过一些比较复杂的评分机制，对进程进行打分，然后将分数高的判定为bad进程，杀死并释放内存。

OOM \_ODJ : 判别进程的优先级。

### 3，进程保活方案

利用系统广播 利用service机制拉活 利用native进程拉活，不过后面被限制了 利用Jobscheduler机制拉活，5.0后 利用账号同步机制拉活

a: Service设置成START\_STICKY kill 后会被重启(等待5秒左右)，重传Intent，保持与重启前一样 b: 通过 startForeground将进程设置为前台进程， 做前台服务，优先级和前台应用一个级别，除非在系统内存非常缺，否则此进程不会被 kill c: 双进程Service： 让2个进程互相保护对方，其中一个Service被清理后，另外没被清理的进程可以立即重启进程 d: 用C编写守护进程(即子进程) : Android系统中当前进程(Process)fork出来的子进程，被系统认为是两个不同的进程。当父进程被杀死的时候，子进程仍然可以存活，并不受影响(Android5.0以上的版本不可行）联系厂商，加入白名单 e.锁屏状态下，开启一个一像素Activity

# Git

1，工作区

2，git常用命令

git init git status git diff git clone git add git commit

3，git工作流

fork操作

# Proguard

1，Proguard技术点

压缩（没用到的资源和文件可以不被压缩到包里）,优化（代码字节码不用的可以移除）,混淆,预检测

2，Proguard工作原理

EntryPoint的类

# Android其他相关面试题