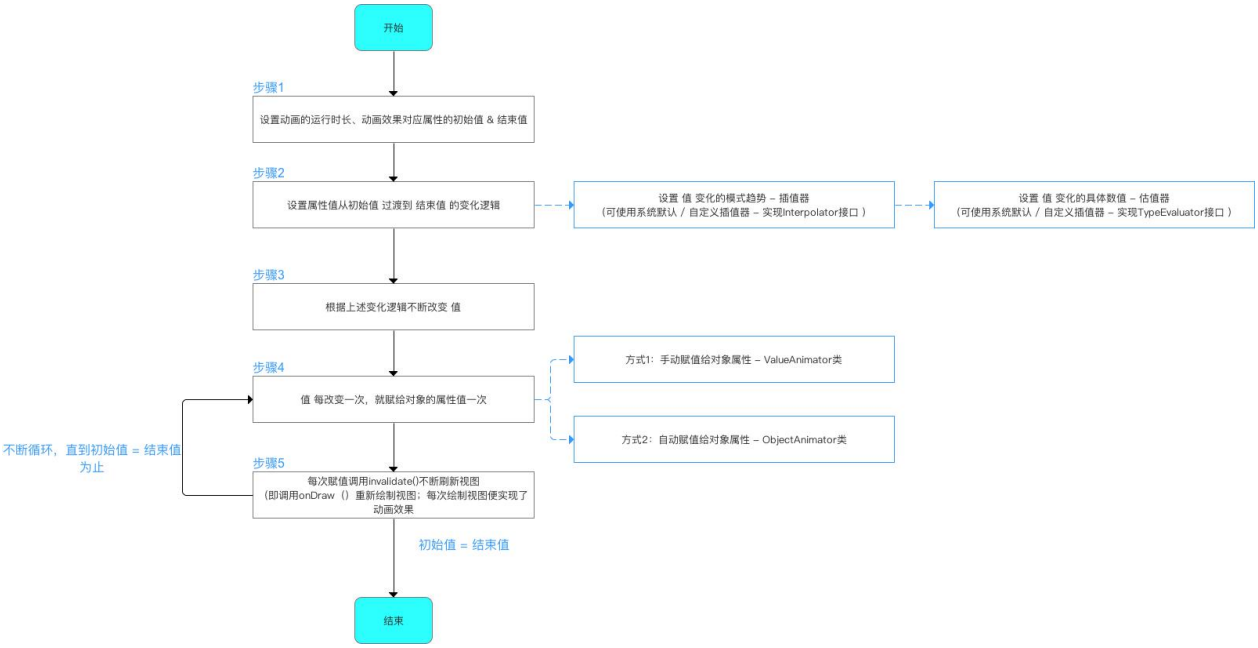
腾讯面试真题解析

Android 源码相关面试专题

1、Android 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。



1）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：

a) ValueAnimator.ofInt(int values)：估值器是整型估值器 IntEaluator

b) 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
    }
```

2) 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

2) 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() {

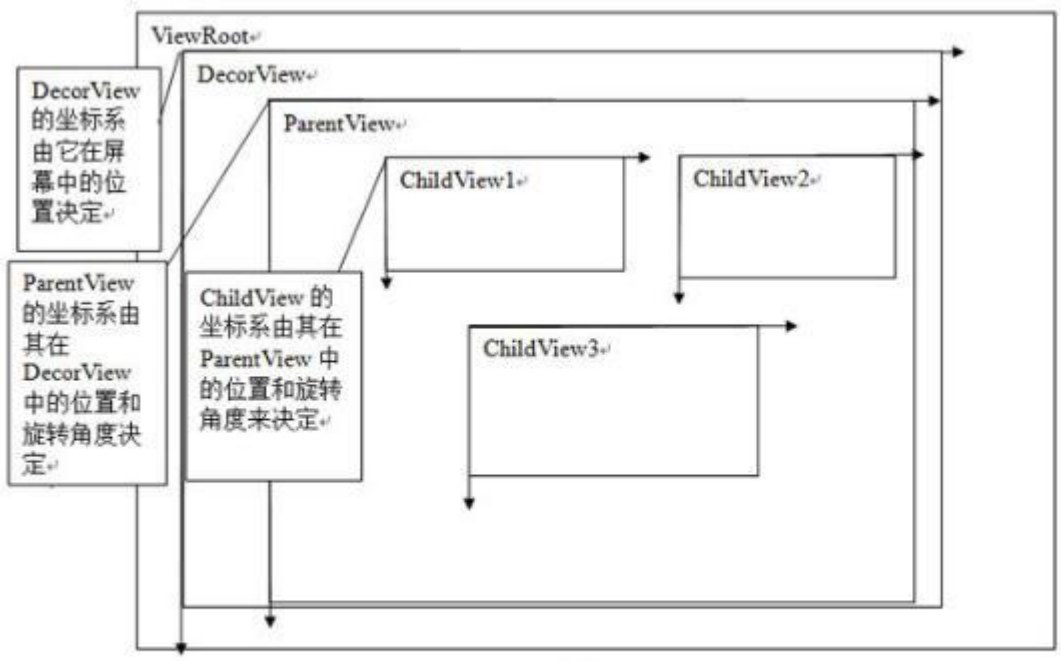
```
        @Override
        public void onAnimationStart(Animation animation) {
            //动画开始时执行
        }

        @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
        }
    });
```

2、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。

3、Android 各个版本 API 的区别

主要记住一些大版本变化:

android3.0 代号 Honeycomb，引入 Fragments, ActionBar,属性动画，硬件加速

android4.0 代号 Ice Cream，API14：截图功能，人脸识别，虚拟按键，3D 优化驱动

android5.0 代号 Lollipop API21：调整桌面图标及部件透明度等

android6.0 代号 M Marshmallow API23，软件权限管理，安卓支付，指纹支持，App 关联，

android7.0 代号 N Preview API24，多窗口支持(不影响 Activity 生命周期)，增加了 JIT 编译器，引入了新的应用签名方案 APK Signature Scheme v2（缩

短应用安装时间和更多未授权 APK 文件更改保护）,严格了权限访问

android8.0 代号 O API26，取消静态广播注册，限制后台进程调用手机资源，桌面图标自适应

android9.0 代号 P API27，加强电池管理，系统界面添加了 Home 虚拟键，提供人工智能 API，支持免打扰模式

4、Requestlayout，onlayout，onDraw，DrawChild 区别与联系

requestLayout()方法 :会导致调用 measure()过程 和 layout()过程 。 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()方法(如果该 View 是 ViewGroup 对象，需要实现该方法，对每个子视图进行布局)

onDraw()方法绘制视图本身 (每个 View 都需要重载该方法，ViewGroup 不需要实现该方法)

drawChild()去重新回调每个子视图的 draw()方法

5、invalidate 和 postInvalidate 的区别及使用

View.invalidate(): 层层上传到父级，直到传递到 ViewRootImpl 后触发了 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 线程刷新 view

postInvalidate：在工作线程刷新 view（底层还是 handler）其实它的原理就是 invalidate+handler

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

```
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。

```
case MSG_INVALIDATE:
```

((View) msg.obj).invalidate();

break;

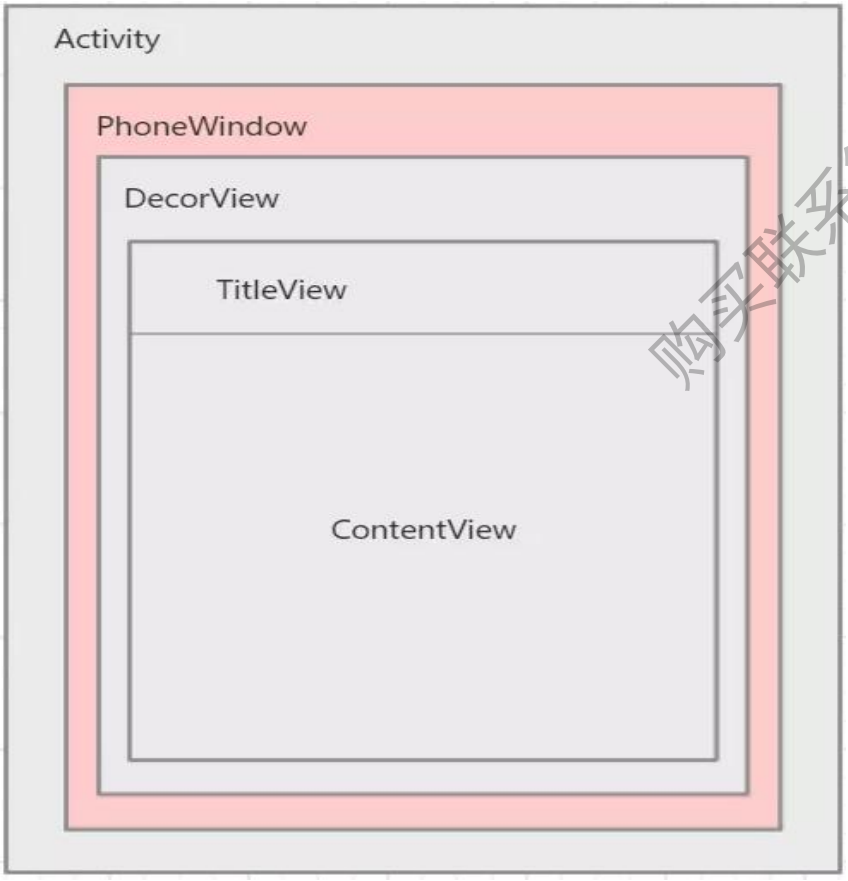
6、Activity-Window-View 三者的差别

Activity：是安卓四大组件之一，负责界面展示、用户交互与业务逻辑处理；

Window：就是负责界面展示以及交互的职能部门，就相当于 Activity 的下属，Activity 的生命周期方法负责业务的处理；

View：就是放在 Window 容器的元素，Window 是 View 的载体，View 是 Window 的具体展示。

三者的关系： Activity 通过 Window 来实现视图元素的展示，window 可以理解为一个容器，盛放着一个个的 view，用来执行具体的展示工作。



7、谈谈对 Volley 的理解

8、如何优化自定义 View

1）在要在 onDraw 或是 onLayout()中去创建对象，因为 onDraw()方法可能会被频繁调用，可以在 view 的构造函数中进行创建对象；

2）降低 view 的刷新频率，尽可能减少不必要的调用 invalidate()方法。或是调用带四种参数不同类型的 invalidate()，而不是调用无参的方法。无参变量需要刷新整个 view，而带参数的方法只需刷新指定部分的 view。在 onDraw()方法中减少冗余代码。

3）使用硬件加速，GPU 硬件加速可以带来性能增加。

4）状态保存与恢复，如果因内存不足，Activity 置于后台被杀重启时，View 应尽可能保存自己属性，可以重写 onSaveInstanceState 和 onRestoreInstanceState 方法，状态保存。

9、低版本 SDK 如何实现高版本 api？

使用@TargetApi 注解·

当代码中有比 AndroidManifest 中设置的 android:minSdkVersion 版本更高

的方法，此时编译器会提示警告，解决方法是在方法上加上@SuppressLint("NewApi"）或者@TargetApi()。但它们仅是屏蔽了 android lint 错误，在方法中还要判断版本做不同的操作。

@SuppressLint("NewApi"）屏蔽一切新 api 中才能使用的方法报的 android lint 错误

@TargetApi() 只屏蔽某一新 api 中才能使用的方法报的 android lint 错误，如@TargetApi(11)如果在方法中用了只有 API14 才开始有的方法，还是会报错。

10、描述一次网络请求的流程

1)域名解析

浏览器会先搜索自身 DNS 缓存且对应的 IP 地址没有过期 ;若未找到则搜索操作系统自身的 DNS 缓存；若还未找到则读本地的 hotsts 文件；还未找到会在 TCP/IP 设置的本地 DNS 服务器上找 ,如果要查询的域名在本地配置的区域资源中，则完成解析；否则根据本地 DNS 服务器会请求根 DNS 服务器；根 DNS 服务器是 13 台根 DNS，会一级一级往下找。

2）TCP 三次握手

客户端先发送 SYN=1，ACK=0，序列号 seq=x 报文；（SYN 在连接建立时用

来同步序号，SYN=1，ACK=0 代表这是一个连接请求报文，对方若同意建立连接，则应在响应报文中使 SYN=1，ACK=1）

服务器返回 SYN=1，ACK=1，seq=y, ack=x+1；

客户端再一次确认，但不用 SYN 了，回复服务端, ACK=1, seq=x+1, ack=y+1

3）建立 TCP 连接后发起 HTTP 请求

客户端按照指定的格式开始向服务端发送 HTTP 请求，HTTP 请求格式由四部分组成，分别是请求行、请求头、空行、消息体，服务端接收到请求后，解析 HTTP 请求，处理完成逻辑，最后返回一个具有标准格式的 HTTP 响应给客户端。

4）服务器响应 HTTP 请求

服务器接收处理完请求后返回一个 HTTP 响应消息给客户端，HTTP 响应信息格式包括：状态行、响应头、空行、消息体

5）浏览器解析 HTML 代码，请求 HTML 代码中的资源

浏览器拿到 html 文件后，就开始解析其中的 html 代码，遇到 js/css/image 等静态资源时，向服务器发起一个 http 请求，如果返回 304 状态码，浏览器会直

接读取本地的缓存文件。否则开启线程向服务器请求下载。

6）浏览器对页面进行渲染并呈现给用户

7）TCP 的四次挥手

当客户端没有东西要发送时就要释放连接（提出中断连接可以是 Client 也可以是 Server），客户端会发送一个 FIN=1 的没有数据的报文，进入 FIN_WAIT 状态，服务端收到后会给客户端一个确认，此时客户端不能发送数据，但可接收信息。

11、HttpUrlConnection 和 okhttp 关系

两者都可以用来实现网络请求，android4.4 之后的 HttpUrlConnection 的实现是基于 okhttp

-Bitmap 对象的理解

-looper 架构

-ActivityThread，AMS，WMS 的工作原理

-自定义 View 如何考虑机型适配

在 onMeasure()的 getDefaultSize()的默认实现中，当 view 的测量模式是 AT_MOST 或 EXACTLY 时，View 的大小都会被设置成子 View MeasureSpec 的 specSize.子 view 的 MeasureSpec 值是根据子 View 的布局参数和父容器的

MeasureSpec 值计算得来。当子 view 的布局参数是 wrap_content 时，对应的测量模式是 AT_MOST，大小是 parentSize,

-自定义 View 的事件

-AstncTask+HttpClient 与 AsyncHttpClient 有什么区别？

-LaunchMode 应用场景

-AsyncTask 如何使用?

-SpareArray 原理

-请介绍下 ContentProvider 是如何实现数据共享的？

-AndroidService 与 Activity 之间通信的几种方式

-IntentService 原理及作用是什么？

原理：IntentService 是继承 Service 的一个抽象类，它在 onCreate()方法中创建了一个 HandlerThread,并启动该线程。HandlerThread 是带有自己消息队列和 Looper 的线程，根据 HandlerThread 的 looper 创建一个 Handler，这样 IntentService 的 ServiceHandler 的 handleMessage()方法就运行在子线程中。handleMessage 中调用了 onHandleIntent()方法，它是一个抽象方法，继承 IntentService 类需要实现该方法，把耗时操作放在 onHandleIntent()方法中 ,等耗时操作运行完成后 ,会调用 stopSelf()方法 ,服务会调用 onDestory()方法消毁自己。如果 onHandleIntent()中的耗时操作未运行完前就调用了 stopSelf()方法，服务调用 onDestory()方法，但耗时操作会继续运行，直至运行完毕。如果同时多次启动 IntentService ,任务会放在一个队列中 ,onCreate()和 onDestory()方法都只会运行一次。

作用：用来处理后台耗时操作，如读取数据库或是本地文件等。

-说说 Activity、Intent、Service 是什么关系

-ApplicationContext 和 ActivityContext 的区别

-SP 是进程同步的吗?有什么方法做到同步？

-谈谈多线程在 Android 中的使用

-进程和 Application 的生命周期

-封装 View 的时候怎么知道 view 的大小

-RecycleView 原理

-AndroidManifest 的作用与理解

1、四大组件是什么
2、四个组件的生命周期

3、Activity 的四种启动模式对比

4、Activity 在有 Dialog 时按 Home 键的生命周期
5、两个 Activity 之间跳转时必然会执行的是哪几个方法
6、 前台切换到后台，然后再回到前台，Activity 生命周期回调方法。弹出 Dialog，生命值周期回调方法

7、 Sparse Array

8、 atomic 包

9、 Android 埋点

10、什么是 Service 以及描述下它的生命周期。Service 有哪些启动方法，有什么区别，怎样停用 Service

11、service 如何杀不死

12、 Service 与 Activity 怎么实现通信

13.什么是 Intent Service？有何优点？

14、什么时候使用 Service

15、请描述一下 Broadcast Receiver

16、如何注册广播

17、广播的生命周期

18、为什么要用 ContentProvider？它和 sql 的实现上有什么差别

19、请介绍下 Android 的数据存储方式

20、简单说说 Fragment

21、他们是怎么进行传递数据的

22、Fragment Manager , add 和 replace 有什么区别

23、Fragment 和 view 的区别

24、请介绍下 Android 中常用的五种布局

25、TextView 、ImageView ， Button,ImageButton 他们之间的联系和区别

26、RelativeLayout 和 FrameLayout 的区别

27、Padding 和 Margin 有什么区别

28、Listview 如何显示不同条目

29、ScrowView 使用的注意

30、Android UI 中的 View 如何刷新

31、什么是 ANR（异步加载） 如何避免它

32、Gradle 中 buildToolsVersion 和 TargetSdkVersion 的区别是什么

33、进程间怎么通信

34、假设手机本地需要缓存数据，如何保证和服务器的数据统一

35、分页怎么做的

36、什么 Context

37、Android 程序与 Java 程序的区别

38、Android 中的动画有哪几类，它们的特点和区别是什么

39、Android 工程的目录结构

40、android 的启动流程

1、四大组件是什么？
Activity【活动】：用于表现功能。
Service【服务】：后台运行服务，不提供界面呈现。
BroadcastReceiver【广播接收器】：用来接收广播。
Content Provider【内容提供商】：支持在多个应用中存储和读取数据，相当于数据库。
2、四个组件的生命周期？
Service 的生命周期：首先 Service 有两种启动方式，而在这两种启动方式下，它的生命周期不同。
通过 startService()方法启动的服务
初始化结束后系统会调用 void onStart(Intent intent) 方法，用于处理传递给 startService()的 Intent 对象。如音乐服务会打开 Intent 来探明将要播放哪首音乐，并开始播放。注意：多次调用 startService()方法会多次触发 onStart()方法。
通过 bindService ()方法启动的服务
初始化结束后系统会调用 IBinder onBind(Intent intent) 方法，用来绑定传递给 bindService 的 Intent 的对象。注意：多次调用 bindService()时，如果该服务已启动则不会再触发此方法。

3、Activity 的四种启动模式对比？

Standard:标准的启动模式，如果需要启动一个 activity 就会创建该 activity 的实例。也是 activity 的默认启动模式。
SingeTop:如果启动的 activity 已经位于栈顶，那么就不会重新创建一个新的 activity 实例。而是复用位于栈顶的 activity 实例对象。如果不位于栈顶仍旧会重新创建 activity 的实例对象。
SingleTask:设置了 singleTask 启动模式的 activity 在启动时，如果位于 activity 栈中，就会复用该 activity，这样的话，在该实例之上的所有 activity 都依次进行出栈操作，即执行对应的 onDestroy()方法，直到当前要启动的 activity 位于栈顶。一般应用在网页的图集，一键退出当前的应用程序。
singleInstance:如果使用 singleInstance 启动模式的 activity 在启动的时候会复用已经存在的 activity 实例。不管这个 activity 的实例是位于哪一个应用当中，都会共享已经启动的 activity 的实例对象。使用了 singlestance 的启动模式的 activity 会单独的开启一个共享栈，这个栈中只存在当前的 activity 实例对象。
4、Activity 在有 Dialog 时按 Home 键的生命周期？
当我们的 Activity 上弹出 Dialog 对话框时，程序的生命周期依然是 onCreate() —> onStart() —> onResume()，在弹出 Dialog 的时候并没有 onPause()和 onStop()方法。而在此时我们按下 Home 键，才会继续执行 onPause()和 onStop()方法。这说明对话框并没有使 Activity 进入后台，而是在点击了 Home 键后 Activity 才进入后台工作。

原因就是，其实 Dialog 是 Activity 的一个组件，此时 Activity 并不是不可见，而是被 Dialog 组件覆盖了其他的组件，此时我们无法对其他组件进行操作而已。

5、两个 Activity 之间跳转时必然会执行的是哪几个方法？
首先定义两个 Activity，分别为 A 和 B。

当我们在 A 中激活 B 时，A 调用 onPause()方法，此时 B 出现在屏幕时，B 调用 onCreate()、onStart()、onResume()。

这个时候 B【B 不是一个透明的窗体或对话框的形式】已经覆盖了 A 的窗体，A 会调用 onStop()方法。

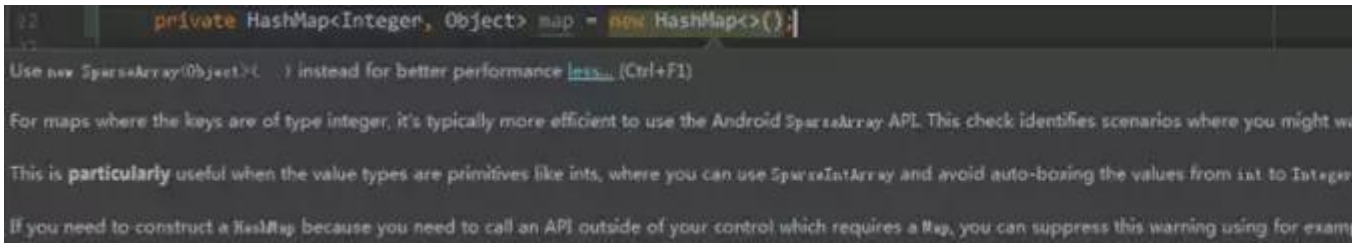
6、 前台切换到后台，然后再回到前台，Activity 生命周期回调方法。弹出 Dialog，生命值周期回调方法？
首先定义两个 Activity，分别为 A 和 B。

完整顺序为：A 调用 onCreate()方法 —> onStart()方法 —> onResume()方法。当 A 启动 B 时，A 调用 onPause()方法，然后调用新的 Activity B，此时调用 onCreate()方法 —> onStart()方法 —> onResume()方法将新 Activity 激活。之后 A 再调用 onStop()方法。当 A 再次回到前台时，B 调用 onPause()方法，A 调用 onRestart()方法 —> onStart()方法 —> onResume()方法，最后调用 B 的 onStop()方法 —> onDestory()方法。

弹出 Dialog 时，调用 onCreate()方法 —> onStart()方法 —> onResume()方法。

7、 SparseArray

当新建一个 key 为整型的 HashMap 时，会出现如下的提示信息，推荐使用 SparseArray 来替代 HashMap：



接下来就来介绍下 SparseArray：

a.数据结构：又称稀疏数组，内部通过两个数组分别存储 key 和 value，并用压缩的方式来存储数据

b.优点：可替代 key 为 int、value 为 Object 的 HashMap，相比于 HashMap

- 能更节省存储空间
- 由于 key 指定为 int，能节省 int 和 Integer 的装箱拆箱操作带来的性能消耗
- 扩容时只需要数组拷贝工作，而不需重建哈希表

c.适用场景：数据量不大（千以内）、空间比时间重要、需要使用 Map 且 key 为整型；不适合存储大容量数据，此时性能将退化至少 50%

d.使用

添加：public void put(int key, E value)

删除：

- public void delete(int key)
- public void remove(int key)实际上内部会调用 delete 方法

查找：

- public E get(int key)
- public E get(int key, E valueIfKeyNotFound)可设置假设 key 不存在时默认返回的 value
- public int keyAt(int index)获取相应的 key
- public E valueAt(int index)获取相应的 value

e.get/put 过程：元素会按照 key 从小到大进行存储，先使用二分法查询 key 对应在数组中的下标 index，然后通过该 index 进行增删查。源码分析见 SparseArray 解析

8、atomic 包

a.原子操作类：

与采取悲观锁策略的 synchronized 不同，atomic 包采用乐观锁策略去原子更新数据，并使用 CAS 技术具体实现

```
//保证自增线程安全的两种方式
public class Sample {
```

```
 private static Integer count = 0;
 synchronized public static void increment() {
 count++;
 }
}
public class Sample {
 private static AtomicInteger count = new AtomicInteger(0);
 public static void increment() {
 count.getAndIncrement();
 }
}
```

基础知识：Java 并发问题–乐观锁与悲观锁以及乐观锁的一种实现方式-CAS

b.类型

原子更新基本类型：

- AtomicInteger：原子更新 Integer
- AtomicLong：原子更新 Long
- AtomicBoolean：原子更新 boolean

以 AtomicInteger 为例，常用方法：

- getAndAdd(int delta)：取当前值，再和 delta 值相加
- addAndGet(int delta) ：先和 delta 值相加，再取相加后的最终值
- getAndIncrement()：取当前 值，再自增
- incrementAndGet() ：先自增，再取自增后的最终值
- getAndSet(int newValue)：取当前值，再设置为 newValue 值

原子更新数组：

- AtomicIntegerArray：原子更新整型数组中的元素
- AtomicLongArray：原子更新长整型数组中的元素
- AtomicReferenceArray：原子更新引用类型数组中的元素

以 AtomicIntegerArray 为例，常用方法：

- addAndGet(int i, int delta)：先将数组中索引为 i 的元素与 delta 值相加，再取相加后的最终值
- getAndIncrement(int i)：取数组中索引为 i 的元素的值，再自增

- compareAndSet(int i, int expect, int update)：如果数组中索引为 i 的元素的值和 expect 值相等，则更新为 update 值

原子更新引用类型：

- AtomicReference：原子更新引用类型
- AtomicReferenceFieldUpdater：原子更新引用类型里的字段
- AtomicMarkableReference：原子更新带有标记位的引用类型

```
//这几个类提供的方法基本一致，以 AtomicReference 为例
public class AtomicDemo {
private static AtomicReference<User> reference = new AtomicReference<>();
public static void main(String[] args) {
 User user1 = new User("a", 1);
 reference.set(user1);
 User user2 = new User("b",2);
 User user = reference.getAndSet(user2);
 System.out.println(user);
//输出
User{userName='a', age=1} System.out.println(reference.get());
//输出
User{userName='b', age=2}}static class User { private String userName;
 private int age; public User(String userName, int age) { this.userName = user
Name; this.age = age;
 } @Override public String toString() {
 return "User{" +"userName='" + userName + '\'' +", age=" + age + '}';
 }
}
}
```

原子更新字段：

- AtomicIntegeFieldUpdater：原子更新整型字段
- AtomicLongFieldUpdater：原子更新长整型字段
- AtomicStampedReference：原子更新带有版本号的引用类型

使用方法：由于原子更新字段类是抽象类，因此需要先通过其静态方法 newUpdater 创建一个更新器，并设置想更新的类和属性

注意：被更新的属性必须用 public volatile 修饰

```
//这几个类提供的方法基本一致，以 AtomicIntegeFieldUpdater 为例
public class AtomicDemo {
```

```
private static AtomicIntegerFieldUpdater updater = AtomicIntegerFieldUpdater.
newUpdater(User.class,"age");
public static void main(String[] args) { User user = new User("a", 1);
 int oldValue = updater.getAndAdd(user, 5);
 System.out.println(oldValue);
//输出 1
 System.out.println(updater.get(user));
//输出 6
}static class User { private String userName;
 public volatile int age;
 public User(String userName, int age) {
 this.userName = userName;
 this.age = age;
 } @Override public String toString() { return "User{" +"userName='" + userNam
e + '\'' +", age=" + age +'}';
 }
}
}
```

c.优点：

可以避免多线程的优先级倒置和死锁情况的发生，提升在高并发处理下的性能，相比于 synchronized ，在非激烈竞争的情况下，开销更小，速度更快

9、 Android 埋点

a.含义：预先在目标应用采集数据，对特定用户行为或事件进行捕获、处理，并以一定方式上报至服务器，便于后续进行数据分析

b.方式

代码埋点：在某个事件发生时通过预先写好的代码来发送数据

- 优点：控制精准，采集灵活性强，可自由选择何时发送自定义数据
- 缺点：开发、测试成本高，需要等发版才能修改线上埋点、更新代价大

无埋点/全埋点：在端上自动采集并上报尽可能多的数据，在计算时筛选出可用的数据

- 优点：很大程度上减少开发、测试的重复劳动，数据可回溯且覆盖全面

- 缺点：采集信息不够灵活，数据量大，后端筛选分析工作量大

可视化埋点：通过可视化工具选择需要收集的埋点数据，下发配置给客户端，终端点击时获取当前点击的控件根据配置文件进行选择上报

- 优点：很大程度上减少开发、测试的重复劳动，数据量可控且相对精确，可在线上动态埋点、而无需等待 App 发版
- 缺点：采集信息不够灵活，无法解决数据回溯的问题

几个实践：51 信用卡(https://mp.weixin.qq.com/s/svzwQkAy0favp0Ty3NtoLA)、网易(https://mp.weixin.qq.com/s/0dHKu5QIBL_4S7Tum-qW2Q)、58 同城(https://mp.weixin.qq.com/s/rP7PiN7lsnUxcY1BAhB_5Q)

10、什么是 Service 以及描述下它的生命周期。Service 有哪些启动方法，有什么区别，怎样停用 Service？

在 Service 的生命周期中，被回调的方法比 Activity 少一些，只有 onCreate, onStartCommand, onDestroy,onBind 和 onUnbind。

通常有两种方式启动一个 Service,他们对 Service 生命周期的影响是不一样的。

1 通过 startService

Service 会经历 onCreate 到 onStartCommand，然后处于运行状态，stopService 的时候调用 onDestroy 方法。

如果是调用者自己直接退出而没有调用 stopService 的话，Service 会一直在后台运行。

2 通过 bindService

Service 会运行 onCreate，然后是调用 onBind，这个时候调用者和 Service 绑定在一起。调用者退出了，Srevice 就会调用 onUnbind->onDestroyed 方法。所谓绑定在一起就共存亡了。调用者也可以通过调用 unbindService 方法来停止服务，这时候 Srevice 就会调用 onUnbind->onDestroyed 方法。

1.一旦在项目的任何位置调用了 Context 的 startService()方法，相应的服务就会启动起来，并回调 onStartCommand()方法。如果这个服务之前还没有创建过，onCreate()方法会先于 onStartCommand()方法执行。

2.服务启动了之后会一直保持运行状态，直到 stopService()或 stopSelf()方法被调用。注意虽然每调用一次 startService()方法，onStartCommand()就会执行一次，但实际上每个服务都只会存在一个实例。所以不管调用了多少次 startService()方法，只需调用一次 stopService()或 stopSelf()方法，服务就会停止下来了。

3.当调用了 startService()方法后，又去调用 stopService()方法，这时服务中的 onDestroy()方法就会执行，表示服务已经销毁了。类似地，当调用了 bindService()方法后，又去调用 unbindService()方法，onDestroy()方法也会执行，这两种情况都很好理解。但是需要注意，我们是完全有可能对一个服务既调用了 startService()方法，又调用了 bindService()方法的，这种情况下该如何才能让服务销毁掉呢？根据 Android 系统的机制，一个服务只要被启动或者被绑定了之后，就会一直处于运行状态，必须要让以上两种条件同时不满足，服务才能被销毁。所以，这种情况下要同时调用 stopService()和 unbindService()方法，onDestroy()方法才会执行这样就把服务的生命周期完整地走了一遍。

start –> bind -> unbind -> stop 经常使用服务长期后台运行，又可以调用服务中的方法 。

11、service 如何杀不死？

1.onStartCommand 方法，返回 START_STICKY（粘性）当 service 因内存不足被 kill，当内存又有的时候，service 又被重新创建

2.设置优先级，在服务里的 ondestory 里发送广播 在广播里再次开启这个服务，双进程守护

service 是否在 main thread 中执行, service 里面是否能执行耗时的操作?

默认情况,如果没有显示的指定 service 所运行的进程, Service 和 Activity 是运行在当前 app 所在进程的 main thread(UI 主线程)里面

service 里面不能执行耗时的操作(网络请求,拷贝数据库,大文件)，需要在子线程中执行 new Thread(){}.start();

service 里面可以弹土司吗

```
service 里面弹 toast 需要添加到主线程里执行

 @Override

   public void onCreate(){

       handler = new Handler(Looper.getMainLooper());

       System.out.println("service started");

      handler.post(new Runnable() {

           @Override
```

```
            public void run() {

                Toast.makeText(getApplicationContext(),
"Test",Toast.LENGTH_SHORT).show();

                });

    }
```

Activity 怎么和 service 绑定，怎么在 activity 中启动自己对应的 service？

startService() 一旦被创建 调用着无关，没法使用 service 里面的方法

bindService () 把 service 与调用者绑定 ,如果调用者被销毁, service 会销毁，可以使用 service 里面的方法

12、 Service 与 Activity 怎么实现通信

方法一：

1. 添加一个继承 Binder 的内部类，并添加相应的逻辑方法
2. 重写 Service 的 onBind 方法，返回我们刚刚定义的那个内部类实例
3. Activity 中创建一个 ServiceConnection 的匿名内部类，并且重写里面的 onServiceConnected 方法和 onServiceDisconnected 方法，这两个方法分别会在活动与服务成功绑定以及解除绑定的时候调用，在 onServiceConnected 方法中，我们可以得到一个刚才那个 service 的 binder 对象，通过对这个 binder 对象进行向下转型，得到我们那个自定义的 Binder 实例，有了这个实例，做可以调用这个实例里面的具体方法进行需要的操作了

方法二 通过 BroadCast(广播)的形式 当我们的进度发生变化的时候我们发送一条广播，然后在 Activity 的注册广播接收器，接收到广播之后更新视图

13.什么是 IntentService？有何优点？

普通的 service ,默认运行在 ui main 主线程，Sdk 给我们提供的方便的,带有异步处理的 service 类

OnHandleIntent() 处理耗时的操作，不需要开启子线程，这个方法已经在子线程中运行了

Intentservice 若未执行完成上一次的任务，将不会新开一个线程，是等待之前的任务完成后，再执行新的任务，等任务完成后再次调用 stopService()

startForeground(id, notification)

拥有 service 的进程具有较高的优先级。当内存不足时，拥有 service 的进程具有较高的优先级。

1． 如果 service 正在调用 onCreate，onStartCommand 或者 onDestory 方法，那么用于当前 service 的进程相当于前台进程以避免被 killed。

2． 如果当前 service 已经被启动(start)，拥有它的进程则比那些用户可见的进程优先级低一些，但是比那些不可见的进程更重要，这就意味着 service 一般不会被 killed.

3． 如果客户端已经连接到 service (bindService),那么拥有 Service 的进程则拥有最高的优先级，可以认为 service 是可见的。

4． 如果 service 可以使用 startForeground(int, Notification)方法来将 service 设置为前台状态，那么系统就认为是对用户可见的，并不会在内存不足时 killed。

14、什么时候使用 Service？

如果有其他的应用组件作为 Service,Activity 等运行在相同的进程中，那么将会增加该进程的重要性。

1.Service 的特点可以让他在后台一直运行,可以在 service 里面创建线程去完成耗时的操作.

2.Broadcast receiver 捕获到一个事件之后,可以起一个 service 来完成一个耗时的操作.

广播

15、请描述一下 Broadcast Receiver。

Android 中：系统在运行过程中，会产生会多事件，那么某些事件产生时，比如：电量改变、收发短信、拨打电话、屏幕解锁、开机，系统会发送广播，只要应用程序接收到这条广播，就知道系统发生了相应的事件，从而执行相应的代码。使用广播接收者，就可以收听广播

广播分两种：有序广播、无序广播

无序广播：无序广播不可中断，不能互相传递数据；

有序广播：一个接一个的传递，广播可中断，通过调用 abortBroadcast()方法；接收者之间可以传递数据（intent）；

无序广播（标准广播）

- 所有与广播中的 action 匹配的广播接收者都可以收到这条广播，并且是没有先后顺序，视为同时收到

有序广播

- 所有与广播中的 action 匹配的广播接收者都可以收到这条广播，但是是有先后顺序的，按照广播接收者的优先级排序

- 优先级的定义：-1000~1000
- <intent-filter android:priority="100" > <action android:name="com.example.broadcasttest.MY_BROADCAST"/> </intent-filter>
- 最终接收者：所有广播接收者都接收到广播之后，它才接收，并且一定会接收
- abortBroadCast：阻止其他接收者接收这条广播，类似拦截，只有有序广播可以被拦截

16、如何注册广播

广播的方式一般有两种，在代码中注册和在 AndroidManifest.xml 中注册，其中前者也被称为动态注册，后者也被称为静态注册。

动态注册：需要使用广播接收者时，执行注册的代码，不需要时，执行解除注册的代码

- 安卓中有一些广播接收者，必须使用代码注册，清单文件注册是无效的
1. 屏幕锁屏和解锁
2. 电量改变

```
public class MainActivity extends Activity {

    private IntentFilter intentFilter;

    private NetworkChangeReceiver networkChangeReceiver;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

          super.onCreate(savedInstanceState);

          setContentView(R.layout.activity_main);

          intentFilter = new IntentFilter();


intentFilter.addAction("android.net.conn.CONNECTIVITY_CHANGE");

          networkChangeReceiver = new NetworkChangeReceiver();

          registerReceiver(networkChangeReceiver, intentFilter);
```

```
    }

    @Override

    protected void onDestroy() {

           super.onDestroy();

           unregisterReceiver(networkChangeReceiver);

    }

    class NetworkChangeReceiver extends BroadcastReceiver {

            @Override

            public void onReceive(Context context, Intent intent) {

                Toast.makeText(context, "network
changes",Toast.LENGTH_SHORT).show();

            }

    }}
```

静态注册：

可以使用清单文件注册

- 广播一旦发出，系统就会去所有清单文件中寻找，哪个广播接收者的 action 和广播的 action 是匹配的，如果找到了，就把该广播接收者的进程启动起来

四大组件其中比较特殊的是广播接收者，可以不在清单文件中配置，可以通过代码进行注册。其他组件全部在清单文件中注册

避免使用隐式 Intent 广播（静态、动态注册）敏感信息，信息可能被其他注册了对应 BroadcastReceiver 的 App 接收

如果广播仅限于应用内，则可以使用 LocalBroadcastManager#sendBroadcast() 实现，避免敏感信息外泄和 Intent 拦截的风险。

```
Intent intent = new Intent("my-sensitive-event");
```

```
intent.putExtra("event", "this is a test event");

LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
```

17、广播的生命周期

a. 广播接收者的生命周期非常短暂的，在接收到广播的时候创建，onReceive()方法结束之后销毁；

b. 广播接收者中不要做一些耗时的工作，否则会弹出 Application No Response 错误对话框；

c. 最好也不要在广播接收者中创建子线程做耗时的工作，因为广播接收者被销毁后进程就成为了空进程，很容易被系统杀掉；

d. 耗时的较长的工作最好放在服务中完成；

内容提供者

请介绍下 ContentProvider 是如何实现数据共享的。

是四大组件之一，内容提供者的作用：把私有数据暴露给其他应用，通常，是把私有数据库的数据暴露给其他应用

应用的数据库是不允许其他应用访问的，内容提供者的作用就是让别的应用访问到你的数据库。

在清单文件中定义内容提供者的标签，注意必须要有 authorities 属性，这是内容提供者的主机名，功能类似地址

```
<provider android:name="com.itheima.contentprovider.PersonProvider"

    android:authorities="com.itheima.person"

    android:exported="true"

 ></provider>
```

把自己的数据通过 uri 的形式共享出去，android 系统下不同程序，数据默认是不能共享访问

需要去实现一个类去继承 ContentProvider

18、为什么要用 ContentProvider？它和 sql 的实现上有什么差别？

屏蔽数据存储的细节,对用户透明,用户只需要关心操作数据的 uri 就可以了

不同 app 之间共享,操作数据

Sql 也有增删改查的方法.

但是 contentprovider 还可以去增删改查本地文件. xml 文件的读取,更改,网络数据读取更改

19、请介绍下 Android 的数据存储方式

1.文件储存，在内部文件和 SD 卡

getCacheDir()，在 data/data/包名/cache

getFilesDir(),在 data/data/包名/files

SD 卡：首先通过 File file = new File(Environment.getExternalStorageDirectory(), "info.txt")，然后通过 io 存储

2.SharedPreference

3.SQLite 数据库： 当应用程序需要处理的数据量比较大时，为了更加合理地存储、管理、查询数据，往往使用关系数据库来存储数据。Android 系统的很多用户数据，如联系人信息，通话记录，短信息等，都是存储在 SQLite 数据库当中的，所以利用操作 SQLite 数据库的 API 可以同样方便的访问和修改这些数据。

4.ContentProvider: 主要用于在不同的应用程序之间实现数据共享的功能，不同于 sharepreference 和文件存储中的两种全局可读写操作模式，内容提供其可以选择只对哪一部分数据进行共享，从而保证我们程序中的隐私数据不会有泄漏的风险

Fragment

20、简单说说 Fragment?

用途：在一个 Activity 里切换界面，切换界面时只切换 Fragment 里面的内容

生命周期方法跟 Activity 一致，可以理解把其为就是一个 Activity，与 Activity 同存亡，Activity 的 XX 方法调用，Fragment 的 XX 方法就调用

21、他们是怎么进行传递数据的?

活动传递给 Fragment:为了方便碎片和活动之间进行通信， FragmentManager 提供了一个类似于 findViewById()的方法，专门用于从布局文件中获取碎片的实例，前提是自己在布局文件中定义 fragment 这个标签，代码如下所示：

```
RightFragment rightFragment = (RightFragment)
getFragmentManager().findFragmentById(R.id.right_fragment);
```

调用 FragmentManager 的 findFragmentById()方法，可以在活动中得到相应碎片的实例，然后就能轻松地调用碎片里的方法了。还有 findViewByTag，在 replace 的时候设置 tag

或者在 fragment 里声明接口，然后 activity 获得 fragment 对象调用接口里的方法

fragment 数据传递给活动，直接 getActivity 就可以调用活动里的方法了

activity 给 fragment 传递数据一般不通过 fragment 的构造方法来传递，会通过 setArguments 来传递，因为当横竖屏会调用 fragment 的空参构造函数，数据丢失。

fragment 和 fragment 数据传递

首先在一个 fragment 可以得到与它相关联的活动，然后再通过这个活动去获取另外一个 fragment 的实例，这样也就实现了不同 fragment 之间的通信功能

22、Fragment Manager , add 和 replace 有什么区别?

- 使用 FragmentTransaction 的时候，它提供了这样两个方法，一个 add ， 一个 replace ,add 和 replace 影响的只是界面，而控制回退的，是事务。
- add 是把一个 fragment 添加到一个容器 container 里。replace 是先 remove 掉相同 id 的所有 fragment，然后在 add 当前的这个 fragment。
- 在大部分情况下，这两个的表现基本相同。因为，一般，咱们会使用一个 FrameLayout 来当容器，而每个 Fragment 被 add 或者 replace 到这个 FrameLayout 的时候，都是显示在最上层的。所以你看到的界面都是一样的。但是， 使用 add 的情况下，这个 FrameLayout 其实有 2 层，多层肯定要比一层的来得浪费，所以还是推荐使用 replace。当然有时候还是需要使用 add 的。比如要实现轮播图的效果，每个轮播图都是一个独立的 Fragment，而他的容器 FrameLayout 需要 add 多个 Fragment，这样他就可以根据提供的逻辑进行轮播了。而至于返回键的时候，这个跟事务有关，跟使用 add 还是 replace 没有任何关系。
- replace()方法会将被替换掉的那个 Fragment 彻底地移除掉，因此最好的解决方案就是使用 hide()和 show()方法来隐藏和显示 Fragment，这就不会让 Fragment 的生命周期重走一遍了。

23、Fragment 和 view 的区别

Fragment 可以有效的对 view 进行管理（增删和替换）而且结构更加清晰，有模块化的实现思想。

用 view 很多逻辑代码可能都需要写在 Activity 里，如果 view 很多， 耦合度会很高。用 Fragment 则可以各自管理，起了解耦的作用。

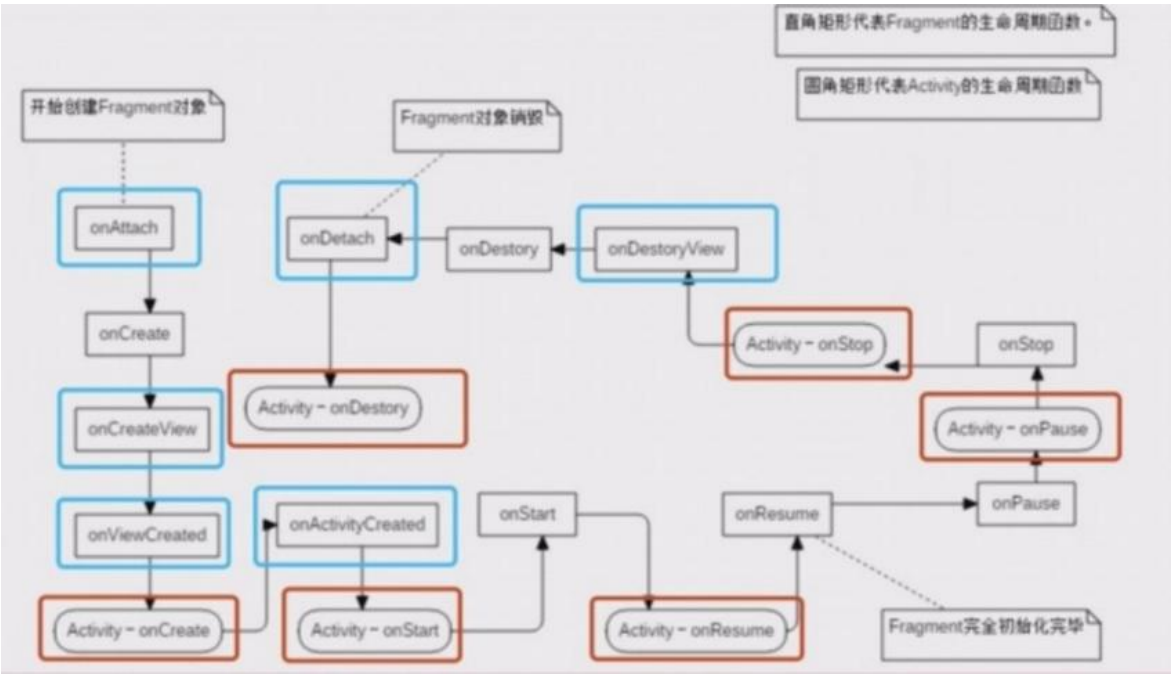
一般软件简单的话直接 view，复杂的用 Fragment。

viewpager 是一个滑动切换的控件，Fragment 是一个轻量级的 Activity，这个 Fragment 可以放到这个 Viewpager 里面去运行。

例如 QQ 或微信那样,可以来回切换不同的选项卡,即切换了不同的 Fragment。通常 Viewpager 会放 fargment 或者 view

Fragment 和 Activity 的区别

因为现在的手机屏幕都比较大了，Activity 会把内容拉的很长，用 Fragment 的话可以左侧是列表，右侧是内容。在一个 Activity 里切换界面，切换界面时只切换 Fragment 里面的内容。Fragment 通常用来作为一个 activity 界面的一部分。



view

24、请介绍下 Android 中常用的五种布局。

FrameLayout（帧布局），LinearLayout （线性布局），AbsoluteLayout（绝对布局），RelativeLayout（相对布局），TableLayout（表格布局）

- FrameLayout：从屏幕的左上角开始布局,叠加显示, 实际应用 播放器的暂停按钮.
- LinearLayout：线性布局，他首先是一个一个从上往下罗列在屏幕上。每一个 LinearLayout 里面又可分为垂直布局、水平布局。当垂直布局时，每一行就只有一个元素，多个元素依次垂直往下；水平布局时，只有一行，每一个元素依次向右排列。

- AbsoluteLayout ： 用 X,Y 坐标来指定元素的位置 android:layout_x="20px" ，android:layout_y="12px" ，指定平板机型的游戏开发、机顶盒开发中经常用到绝对布局
- RelativeLayout：在相对的布局中主要就进行避免覆盖的问题，就是组件 1 可能会覆盖在组件 2 上（屏幕适配），在相对的布局中主要就进行避免覆盖的问题，就是组件 1 可能会覆盖在组件 2 上
- TableLayout：有几行，就有几个<TableRow/>,有几列，那么在<TableRow>中就有几个<TestView>,TableRow 的子节点的宽和高是包裹内容,不需要指定宽高

25、TextView 、ImageView ， Button,ImageButton 他们之间的联系和区别

ImageView 控件负责显示图片,其图片来源既可以是资源文件的 id,也可以是 Drawable 对象或 Bitmap 对象,还可以是 内容提供者（Content Provider）的 Uri.

ImageButton 控件 继承自 ImageView。与 Button 相同之处：都用于响应按钮的点击事件

不同之处：ImageButton 只能显示图片;Button 用于显示文字

屏幕适配

- 开发时选取主流屏幕 1280*720，用相对布局和线性布局
- 用 dp sp 不用 px，dp 单位动态匹配
- 开发后期在不同的分辨率上测试，没有太大问题可以上线
- 权重适配:weight 只有线性布局有
- 代码适配:getWindowManager().getDefaultDisplay().getWidth();获得屏幕的宽高
- 如果屏幕放不下了，可以使用 ScrollView（可以上下拖动）
- 布局适配:layout-800x180 针对某一种屏幕 工作量大
- 尺寸适配:dp=px/设备密度=getResource().getDisplayMetrice().dsnsity;
- 根据不同分辨率的屏幕建立不同的 valuse，比如 valuse-1280x720,values 里的 dimens 里算出 dp，最后引用系统会自动匹配。

(约等于)320*240（0.5） 480*320（1） 480*800（1.5） 1280*720（2）就不用布局适配了

- 在进行开发的时候，我们需要把合适大小的图片放在合适的文件夹里面。大分辨率图片（单维度超过 1000）大分辨率图片建议统一放在 xxhdpi 目录下管理，否则将导致占用内存成倍数增加。

说明：*为了支持多种屏幕尺寸和密度，Android 为多种屏幕提供不同的资源目录进行适配。为不同屏幕密度提供不同的位图可绘制对象，可用于密度特定资源的配置限定符（在下面详述） 包括 ldpi（低）、 mdpi（中）、 hdpi（高）、 xhdpi（高）、 xxhdpi （超超高）和 xxxhdpi（超超超高）。例如，高密度屏幕的位图应使用 drawable-hdpi。根据当前的设备屏幕尺寸和密度，将会寻找最匹配的*

资源，如果将高分辨率图片放入低密度目录，将会造成低端机加载过大图片资源，又可能造成 OOM，同时也是资源浪费，没有必要在低端机使用大。

26、RelativeLayout 和 FrameLayout 的区别

FrameLayout 主要是在多层之间的布局，RelativeLayout 则是在同层之间不同位置之间的布局,效果上没有什么大的区别，都可以实现，只是看哪种实现更容易。比如我们收到信息了，我们可以在信息图标上增加 FrameLayout 用于显示气泡。

27、Padding 和 Margin 有什么区别？

Padding 是控件内容的距离 margin 是控件和控件间的距离

28、Listview 如何显示不同条目？

第一种：在 getview 方法里根据 position 进行判断，比如，position 等于 0 是，显示标题，否则显示数据，当然相应的 setOnItemClickListener 也要去判断

第二种：根据 listview 里的 getItemViewType，getViewTypeCount 方法显示不同条目

```
/** 根据位置 判断当前条目是什么类型 */

@Override

public int getItemViewType(int position) {  //20

    if (position == datas.size()) { // 当前是最后一个条目

        return MORE_ITEM;

    }

    return getInnerItemViewType(position); // 如果不是最后一个条目 返回默认类型

}

private int getInnerItemViewType(int position) {

    return DEFAULT_ITEM;

}
```

```
/** 当前 ListView 有几种不同的条目类型 */

@Override

public int getViewTypeCount() {

    return super.getViewTypeCount() + 1; // 2 有两种不同的类型

}
```

29、ScrowView 使用的注意

在不同的屏幕上显示内容不同的情况，其实这个问题我们往往是用滚动视图来解决的，也就是 ScrowView，

需要注意的是 ScrowView 中使用 layout_weight 是无效的，既然使用 ScrowView 了，就把它里面的控件的大小都设成固定的吧。

30、Android UI 中的 View 如何刷新

在主线程中 拿到 view 调用 Invalide()方法，在子线程里面可以通过 postInvalide()方法;

invalidate();//主线程，刷新当前视图 导致 执行 onDraw 执行

postInvalidate();//子线程

31、什么是 ANR（异步加载） 如何避免它？

android 在主线程是不能加载网络数据或图片、数据库查询、复杂业务逻辑处理以及费时任务操作，因为 Android 的 UI 操作并不是线程安全的，并且所有涉及 UI 的操作必须在 UI 线程中完成。Android 应用在 5s 内无响应的话会导致 ANR(Application Not Response)，这就要求开发者必须遵循两条法则：1、不能阻塞 UI 线程，2、确保只在 UI 线程中访问 Android UI 工具包。

Activity 5 秒 broadcast10 秒，服务 20 秒，耗时的操作在主 thread 里面完成

解决办法：Thread + Handler + Message ， Thread + Handler + post，AsyncTask，intentservice

runOnUiThread(Runnable)在子线程中直接使用该方法，可以更新 UI

实现侧边栏、和指示器效果、页面滑动有几种方式

侧边栏：自定义、slidingmenu、DrawerLayout 、SlidingDrawer

指示器效果：自定义、viewpager 里面的 PagerTabStrip、ActionBar Tab 标签、viewpagerindicate、FragmentTabHost、TabActivity、radiobutton

页面滑动：自定义、viewpager、手势识别器，其实就是 onTouchEvent 提供的简单工具类，onTouchEvent 将触摸事件委托给了手势识别器

其他

32、Gradle 中 buildToolsVersion 和 TargetSdkVersion 的区别是什么

compileSdkVersion, minSdkVersion 和 targetSdkVersion 的作用：他们分别控制可以使用哪些 API ，要求的 API 级别是什么，以及应用的兼容模式。TargetSdkVersion 设为 23 那么是按 6.0 设置的（运行时权限），小于 23 是按 6.0 以前的方式（安装时默认获得权限，且用户无法在安装 App 之后取消权限）

33、进程间怎么通信

binder 是安卓中的一个类，它实现了 IBinder 接口，是安卓中跨进程通信的方式。当绑定服务的时候会返回一个 binder 对象，然后通过他进行多进程间的通信。

其实进程间通信就是为了实现数据共享。一个程序不同组件在不同进程也叫多进程，和俩个应用没有本质区别。使用 process 属性可以实现多进程，但是会带来很多麻烦，主要原因是共享数据会失败，弊端有:静态和单利失效，同步失效，sharedprefer 可靠性减低等问题

Intent，Binder（AIDL），sharedpre、Messenger

- 使用 intent 的附加信息 extras 来传递，通过 bundle，传递的是 bundle 支持的类型，比如基本数据类型、实现 pracellable 或 serializeable 的对象
- 使用文件共享，序列化或是 sharedpre，不过不适用于读写并发的操作
- 通过 message 进行传递，在远程服务里创建 message 对像，在 onbind 里返回(message.getbinder)。在客户端绑定服务，拿着 message 对象发消息(可以用 bundle)。在远程服务的 handlermessage 方法就会收到。他是一个个处理的，如果大量并发请求用 aidl，mrssage 底层就是 aidl
- AIDL
- socket 可以实现俩个终端通信，也可以在一个设备的俩个进程通信。需要在服务里创建服务端
- ContentProvider（进程间数据共享）和 message 一样，底层也是 binder，除了 oncreate 方法其他方法(crud)都是运行在 bindler 线程里。所以在 oncerate 里不能做耗时操作。在其他应用访问通过 uri(主机名),例如 gercontentResolver.query(uri,null,null....)
- 有以上 6 种

使用多进程显而易见的好处就是分担主进程的内存压力。我们的应用越做越大，内存越来越多，将一些独立的组件放到不同的进程，它就不占用主进程的内存空间了。当然还有其他好处，有心人会发现 Android 后台进程里有很多应用是多个进程的，因为它们要常驻后台，特别是即时通讯或者社交应用，不过现在多进程已经被用烂了。典型用法是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。

34、假设手机本地需要缓存数据，如何保证和服务器的数据统一？

- 比如有个网络更新的功能,activity 可以每隔半小时开启 service 去访问服务器，获取最新的数据。
- 在缓存文件里面加入时间戳，根据实际情况在一定的时间差内再次访问网络数据、判断 URL

在缓存的第一行写一个上当前时间，读的时候判断是不是过期，根据需求看需要多久跟新

35、分页怎么做的？

分页根据服务器接口参数决定每次加载多少，getviewtype，getitemview

分批处理 解决的是时间等待的问题，不能解决内存占用的问题。

要想解决内存占用问题，可以采用分页方式

36、什么 Context

Android 工程环境中像 Activity、Service、BroadcastReceiver 等系统组件，而这些组件并不是像一个普通的 Java 对象 new 一下就能创建实例的了，而是要有它们各自的上下文环境，这就是 Context。可以这样讲，Context 是维持 Android 程序中各组件能够正常工作的一个核心功能类。

Context 功能很多，可以弹出 Toast、启动 Activity、启动 Service、发送广播、操作数据库等等等等都需要用到 Context。

Context 一共有 Application、Activity 和 Service 三种类型：Context 数量 = Activity 数量 + Service 数量 + 1 。1 代表着 Application 的数量，因为一个应用程序中可以有多个 Activity 和多个 Service，但是只能有一个 Application。

Application 通常作为工具类来使用的，Application 中在 onCreate()方法里去初始化各种全局的变量数据是一种比较推荐的做法，但是如果你想把初始化的时间点提前到极致，也可以去重写 attachBaseContext()方法。不能写在构造函数里。

37、Android 程序与 Java 程序的区别？

Android 程序用 android sdk 开发,java 程序用 javasdk 开发.

Android SDK 引用了大部分的 Java SDK，少数部分被 Android SDK 抛弃，比如说界面部分，java.awt swing package 除了 java.awt.font 被引用外，其他都被抛弃，在 Android 平台开发中不能使用。 Android sdk 添加工具 jar httpclient , pull opengl

38、Android 中的动画有哪几类，它们的特点和区别是什么？

三种：补间动画、帧动画、属性动画。

补间动画是放置到 res/anim/下面

帧动画是放置到 res/drawable/下面，子节点为 animation-list，在这里定义要显示的图片和每张图片的显示时长

补间动画

- 如果动画中的图像变换比较有规律时，例如图像的移动（TranslateAnimation）、旋转（RotateAnimation）、缩放（ScaleAnimation）、透明度渐变（AlphaAnimation），这些图像变化过程中的图像都可以根据一定的算法自动生成，我们只需要指定动画的第一帧和最后一帧图像即可，这种自动生成中间图像的动画就是补间动画。
- 补间动画，只是一个动画效果，组件其实还在原来的位置上，xy 没有改变

帧动画：传统的动画方法，通过顺序的播放排列好的图片来实现，类似电影，一张张图片不断的切换，形成动画效果，要自己指定每一帧

属性动画：动画的对象除了传统的 View 对象，还可以是 Object 对象，动画结束后，Object 对象的属性值被实实在在的改变了

39、Android 工程的目录结构

src：项目的 java 代码

assets：资源文件夹，存放视频或者音乐等较大的资源文件

bin：存放应用打包编译后的文件

res：资源文件夹，在这个文件夹中的所有资源，都会有资源 id，读取时通过资源 id 就可以读取

资源 id 不能出现中文

coverview 原理：

这个参数用于将之前加载好的布局进行缓存，以便之后可以进行重用，在 getView()方法中进行了判断，如果 convertView 为空，则使用

LayoutInflater 去加载布局，如果不为空则直接对 convertView 进行重用

40、android 的启动流程

当程序启动 Linux 内核后，会加载各种驱动和数据结构，当有了驱动以后，开始启动 Android 系统同时会加载用户级别的第一个进程 init（system\core\init.c），加载 init.rc 文件，会启动一个 Zygote 进程，此进程是 Android 系统的一个母进程，用来启动 Android 的其他服务进程，然后接着在里面启动各种硬件服务和 activity。Android 系统启动完成，打开了 Luncher 应用的 Home 界面。

Logcat

1. Log.v()

这个方法用于打印那些最为琐碎的，意义最小的日志信息。对应级别 verbose，是 Android 日志里面级别最低的一种。

2. Log.d()

这个方法用于打印一些调试信息， 这些信息对你调试程序和分析问题应该是有帮助的。对应级别 debug，比 verbose 高一级。

3. Log.i()

这个方法用于打印一些比较重要的数据，这些数据应该是你非常想看到的，可以帮你分析用户行为的那种。对应级别 info，比 debug 高一级。

4. Log.w()

这个方法用于打印一些警告信息，提示程序在这个地方可能会有潜在的风险，最好去修复一下这些出现警告的地方。对应级别 warn，比 info 高一级。

5. Log.e()

这个方法用于打印程序中的错误信息，比如程序进入到了 catch 语句当中。当有错误信息打印出来的时候，一般都代表你的程序出现严重问题了，必须尽快修复。对应级别 error，比 warn 高一级.

推送

所有需要客户端被动接收信息的功能模块,都可以用推送实现，比如 A 想给 B 发消息，A 调服务器接口，服务器只是存数据，它调推送的接口，推送去去找 B。推送用的是 xmpp 协议， 是一种基于 TCP/IP 的协议, 这种协议更适合消息发送

- socket 套接字, 发送和接收网络请求 - 长连接 keep-alive, 服务器基于长连接找到设备,发送消息 - 心跳包 , 客户端会定时(30 秒一次)向服务器发送一段极短的数据,作为心跳包, 服务器定时收到心跳,证明客户端或者,才会发消息.否则将消息保存起来,等客户端活了之后(重新连接),重新发送.