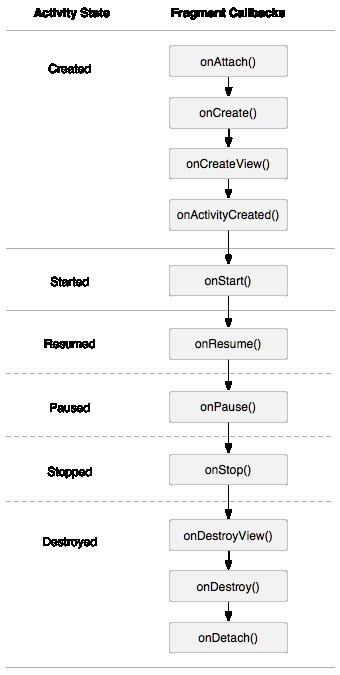
**1、Fragment 跟 Activity 之间是如何传值的**

当 Fragment 跟 Activity 绑 定 之 后 ， 在 Fragment 中 可 以 直 接 通 过 getActivity（）方法获取到其绑定的 Activity 对象，这样就可以调用 Activity 的方法了。在 Activity 中可以通过如下方法获取到 Fragment 实例

FragmentManager fragmentManager = getFragmentManager();  
Fragment fragment = fragmentManager.findFragmentByTag(tag);  
Fragment fragment = fragmentManager.findFragmentById(id);

获取到 Fragment 之后就可以调用 Fragment 的方法。也就实现了通信功能。

**2、描述一下 Fragment 的生命周期**



**3、Fragment 的 replace 和 add 方法的区别**

Fragment 本 身 并 没 有 replace 和 add 方 法 ， 这 里 的 理 解 应 该 为 使 用 FragmentManager 的 replace 和 add 两种方法切换 Fragment 时有什么不同。 我 们 经 常 使 用 的 一 个 架 构 就 是 通 过 RadioGroup 切 换 Fragment ， 每 个 Fragment 就是一个功能模块。  
**case** R.id.*rb\_1*:  
rb\_1.setBackgroundColor(Color.*RED*);  
transaction.show(fragment1);  
// transaction.replace(R.id.fl, fragment1, "Fragment1");  
**break**;  
**case** R.id.*rb\_2*:  
rb\_2.setBackgroundColor(Color.*YELLOW*);  
// transaction.replace(R.id.fl, fragment2, "Fragment2");  
transaction.show(fragment2);  
**break**;  
**case** R.id.*rb\_3*:  
rb\_3.setBackgroundColor(Color.*BLUE*);  
// transaction.replace(R.id.fl, fragment3, "Fragment3");  
transaction.show(fragment3);  
**break**;

实现这个功能可以通过 replace 和 add 两种方法。 Fragment 的容器一个 FrameLayout，add 的时候是把所有的 Fragment 一层 一层的叠加到了 FrameLayout 上了，而 replace 的话首先将该容器中的其他 Fragment 去除掉然后将当前 Fragment 添加到容器中。 一个 Fragment 容器中只能添加一个 Fragment 种类，如果多次添加则会报异 常，导致程序终止，而 replace 则无所谓，随便切换。 因为通过 add 的方法添加的 Fragment，每个 Fragment 只能添加一次，因此 如果要想达到切换效果需要通过 Fragment 的的 hide 和 show 方法结合者使用。将要显示的 show 出来，将其他 hide 起来。这个过程 Fragment 的生命周期没有变化。 通 过 replace 切 换 Fragment ， 每 次 都 会 执 行 上 一 个 Fragment 的 onDestroyView，新 Fragment 的 onCreateView、onStart、onResume 方法。 基于以上不同的特点我们在使用的使用一定要结合着生命周期操作我们的视图 和数据。

**4、Fragment 如何实现类似 Activity 栈的压栈和出栈效果  
的？**

Fragment 的事物管理器内部维持了一个双向链表结构，该结构可以记录我 们每次 add 的 Fragment 和 replace 的 Fragment， 然后当我们点击 back 按钮 的时候会自动帮我们实现退栈操作。

**5、Fragment 在你们项目中的使用**

Fragment 是 android3.0 以后引入的的概念，做局部内容更新更方便，原 来 为 了 到达 这 一 点 要 把 多个 布 局 放 到 一 个 activity 里 面 ， 现在 可 以 用多 Fragment 来代替，只有在需要的时候才加载 Fragment，提高性能。 Fragment 的好处：

（1）Fragment 可以使你能够将 activity 分离成多个可重用的组件，每个都有 它自己的生命周期和 UI。

（2）Fragment 可以轻松得创建动态灵活的 UI 设计，可以适应于不同的屏幕尺 寸。从手机到平板电脑。

（3）Fragment 是一个独立的模块,紧紧地与 activity 绑定在一起。可以运行中 动态地移除、加入、交换等。

（4）Fragment 提供一个新的方式让你在不同的安卓设备上统一你的 UI。

（5）Fragment 解决 Activity 间的切换不流畅，轻量切换。

（6）Fragment 替代 TabActivity 做导航，性能更好。

（7）Fragment 在 4.2.版本中新增嵌套 fragment 使用方法，能够生成更好的 界面效果。

**6、如何切换 fragement,不重新实例化**

翻看了 Android 官方 Doc，和一些组件的源代码，发现 replace()这个方法只是 在上一个 Fragment 不再需要时采用的简便方法. 正确的切换方式是 add()， 切换时 hide()，add()另一个 Fragment；再次切换时， 只需 hide()当前，show()另一个。 这样就能做到多个 Fragment 切换不重新实例化：  
**public void switchContent(Fragment from, Fragment to) {**  
**if (mContent != to) {**  
**mContent = to;**  
**FragmentTransaction transaction**  
**=mFragmentMan.beginTransaction().setCustomAnimations(android.R.anim.fade\_**  
**in, R.anim.slide\_out);**  
**if (!to.isAdded()) { //** 先判断是否被 **add** 过  
**transaction.hide(from).add(R.id.content\_frame, to).commit(); //**  
隐藏当前的 **fragment**，**add** 下一个到 **Activity** 中  
**} else {**  
**transaction.hide(from).show(to).commit(); //** 隐藏当前的

**fragment**，显示下一个  
**}**  
**}**  
**}**

**7、如果在 Android3.0 之前的版本中使用 Fragment 需要添中哪个包？**

android-support-v4.jar 包，通过 getSupportFragmentManager（）返回 Fragment 实例

**8、请解释下在单线程模型中 Message、Handler、Message Queue、Looper 之间的关系，并分别说明它们的作用。**  
**Handler 简介：**

一个 Handler 允许你发送和处理 Message 和 Runable 对象，这些对象和一个线程的MessageQueue 相关联。每一个线程实例和一个单独的线程以及该线程的 MessageQueue 相关联。当你创建一个新的 Handler 时，它就和创建它的线程绑定在一起了。这里，线程我们也可以理解为线程的 MessageQueue。从这一点上来看，Handler 把 Message 和 Runable 对象传递给 MessageQueue，而且在这些对象离开 MessageQueue 时，Handler 负责执行他们。

Handler 有两个主要的用途：（1）确定在将来的某个时间点执行一个或者一些 Message和 Runnable 对象。（2）在其他线程（不是 Handler 绑定线程）中排入一些要执行的动作。

**Message 简介：**

Message 类就是定义了一个信息，这个信息中包含一个描述符和任意的数据对象，这个信息被用来传递给 Handler.Message 对象提供额外的两个 int 域和一个 Object 域，这可以让 你 在 大 多 数 情 况 下 不 用 作 分 配 的 动 作 。尽管 Message 的构造 函数是 public 的，但 是获取 Message 实例的 最好方法是调用Message.obtain(),或者 Handler.obtainMessage()方法，这些方法会从回收对象池中获取一个。

**MessageQueue 简介：**

这是一个包含 message 列表的底层类。Looper 负责分发这些 message。Messages 并不是直接加到一个 MessageQueue 中，而是通过 MessageQueue.IdleHandler 关联到 Looper。你 可 以 通 过 Looper.myQueue() 从 当 前 线 程 中 获 取 MessageQueue 。

**Looper 简介：**

是 MessageQueue 的管理者 ，用来执行一个线程中的 message 循环。默认情况，没有一个消息循环关联到线程。在线程中调用 prepare()创建一个 Looper，然后用 loop()来处理 messages，直到循环终止。大多数和 message loop 的交互是通过 Handler。

**9、子线程中能不能 new handler？为什么？**

不 能 , 如 果 在 子 线 程 中 直 接 new Handler() 会 抛 出 异 常

java.lang.RuntimeException: Can't create handler inside thread that hasnot called

在没有调用 Looper.prepare()的时候不能创建 Handler,因为在创建Handler 的源码中做了如下操作Handler 的构造方法中

public static Looper myLooper() {  
return sThreadLocal.get();  
}

mLooper = Looper.myLooper();  
if (mLooper == null) {  
throw new RuntimeException(  
"Can't create handler inside thread that has not called  
Looper.prepare()");  
}

**10、Handler 机制**  
Android 中主线程也叫 UI 线程，那么从名字上我们也知道主线程主要是用来创建、更新 UI 的，而其他耗时操作，比如网络访问，或者文件处理，多媒体处理等都需要在子线程中操作，之所以在子线程中操作是为了保证 UI 的流畅程度，手机显示的刷新频率是 60Hz，也就是一秒钟刷新 60 次，每 16.67 毫秒刷新一次，为了不丢帧，那么主线程处理代码最好不要超过 16 毫秒。当子线程处理完数据后，为了防止 UI 处理逻辑的混乱，Android 只允许主线程修改 UI，那么这时候就需要 Handler 来充当子线程和主线程之间的桥梁了。

我 们 通 常 将 Handler 声 明 在 Activity 中 ， 然 后 覆 写 Handler 中 的handleMessage 方 法 , 当 子 线 程 调 用 handler.sendMessage() 方 法 后handleMessage 方法就会在主线程中执行。这里面除了 Handler、Message 外还有隐藏的 Looper 和 MessageQueue对象。

在主线程中 Android 默认已经调用了 Looper.preper()方法，调用该方法的目的是在 Looper 中创建 MessageQueue 成员变量并把 Looper 对象绑定到当前线程中。当调用 Handler 的 sendMessage（对象）方法的时候就将 Message对象添加到了 Looper 创建的 MessageQueue 队列中，同时给 Message 指定了 target 对象，其实这个 target 对象就是 Handler 对象。主线程默认执行了Looper.looper（）方法，该方法从 Looper 的成员变量 MessageQueue 中取出 Message，然后调用 Message 的 target 对象的 handleMessage()方法。这样就完成了整个消息机制。

**11、事件分发中的 onTouch 和 onTouchEvent 有什么区别，又该如何使用？**

这两个方法都是在 View 的 dispatchTouchEvent 中调用的，onTouch 优先于 onTouchEvent 执行。如果在 onTouch 方法中通过返回 true 将事件消费掉，onTouchEvent 将不会再执行。

另外需要注意的是，onTouch 能够得到执行需要两个前提条件，第一mOnTouchListener 的值不能为空，第二当前点击的控件必须是 enable 的。因此如果你有一个控件是非 enable 的，那么给它注册 onTouch 事件将永远得不到执行。对于这一类控件，如果我们想要监听它的 touch 事件，就必须通过在该控件中重写 onTouchEvent 方法来实现。

**12、请描述一下 Android 的事件分发机制**

Android 的事件分发机制主要是 Touch 事件分发，有两个主角:ViewGroup和 View。Activity 的 Touch 事件事实上是调用它内部的 ViewGroup 的 Touch事件，可以直接当成 ViewGroup 处理。

View 在 ViewGroup 内，ViewGroup 也可以在其他 ViewGroup 内，这时候把内部的 ViewGroup 当成 View 来分析。

先分析 ViewGroup 的处理流程：首先得有个结构模型概念：ViewGroup和 View 组成了一棵树形结构，最顶层为 Activity 的 ViewGroup，下面有若干的 ViewGroup 节点，每个节点之下又有若干的 ViewGroup 节点或者 View 节点，依次类推。

当一个 Touch 事件(触摸事件为例)到达根节点，即 Acitivty 的 ViewGroup时 ， 它 会 依 次 下 发 ， 下 发 的 过 程 是 调 用 子 View(ViewGroup) 的dispatchTouchEvent 方法实现的。简单来说，就是 ViewGroup 遍历它包含着的子 View，调用每个 View 的 dispatchTouchEvent 方法，而当子 View 为ViewGroup 时，又会通过调用 ViwGroup 的 dispatchTouchEvent 方法继续调用其内部的 View 的 dispatchTouchEvent 方法。上述例子中的消息下发顺序是这样的：①-②-⑤-⑥-⑦-③-④。dispatchTouchEvent 方法只负责事件的分发，它拥有 boolean 类型的返回值，当返回为 true 时，顺序下发会中断。在上述例子中如果⑤的 dispatchTouchEvent 返回结果为 true，那么⑥-⑦-③-④将都接收不到本次 Touch 事件。

1.Touch 事件分发中只有两个主角:ViewGroup 和 View。ViewGroup 包含onInterceptTouchEvent、dispatchTouchEvent、onTouchEvent 三个相关事件。View 包含 dispatchTouchEvent、onTouchEvent 两个相关事件。其中ViewGroup 又继承于 View。

2.ViewGroup 和 View 组成了一个树状结构， 根节点为 Activity 内部包含的一个 ViwGroup。

3.触摸事件由 Action\_Down、Action\_Move、Aciton\_UP 组成，其中一次完整的触摸事件中，Down 和 Up 都只有一个，Move 有若干个，可以为 0 个。

4.当 Acitivty 接收到 Touch 事件时， 将遍历子 View 进行 Down 事件的分发。ViewGroup 的遍历可以看成是递归的。分发的目的是为了找到真正要处理本次完整触摸事件的 View，这个 View 会在 onTouchuEvent 结果返回 true。

5.当某个子 View 返回 true 时，会中止 Down 事件的分发，同时在ViewGroup 中记录该子 View。 接下去的 Move 和 Up 事件将由该子 View 直接进行处理。由于子 View 是保存在 ViewGroup 中的，多层 ViewGroup 的节点结构时， 上级 ViewGroup 保存的会是真实处理事件的 View 所在的 ViewGroup对象:如 ViewGroup0-ViewGroup1-TextView 的结构中，TextView 返回了true，它将被保存在 ViewGroup1 中，而 ViewGroup1 也会返回 true，被保存在 ViewGroup0 中。当 Move 和 UP 事件来时，会先从 ViewGroup0 传递至ViewGroup1，再由 ViewGroup1 传递至 TextView。

6. 当 ViewGroup 中 所 有 子 View 都 不 捕 获 Down 事 件 时 ， 将 触 发ViewGroup 自 身 的 onTouch 事 件 。 触 发 的 方 式 是 调 用super.dispatchTouchEvent 函数，即父类 View 的 dispatchTouchEvent 方法。在所有子 View 都不处理的情况下，触发 Acitivity 的 onTouchEvent 方法。

7.onInterceptTouchEvent 有两个作用：1.拦截 Down 事件的分发。2.中止Up 和 Move 事件向目标 View 传递，使得目标 View 所在的 ViewGroup 捕获Up 和 Move 事件。

**13、什么是 AIDL 以及如何使用**

①aidl 是 Android interface definition Language 的英文缩写，意思Android 接口定义语言。②使用 aidl 可以帮助我们发布以及调用远程服务，实现跨进程通信。

③将服务的 aidl 放到对应的 src 目录，工程的 gen 目录会生成相应的接口类我们通过 bindService（Intent，ServiceConnect，int）方法绑定远程服务，在 bindService 中 有 一 个 ServiceConnec 接 口 ， 我 们 需 要 覆 写 该 类 的

onServiceConnected(ComponentName,IBinder)方法，这个方法的第二个参数 IBinder 对象其实就是已经在 aidl 中定义的接口，因此我们可以将 IBinder对象强制转换为 aidl 中的接口类。我们通过 IBinder 获取到的对象（也就是 aidl 文件生成的接口）其实是系统产生的代理对象， 该代理对象既可以跟我们的进程通信， 又可以跟远程进程通信，作为一个中间的角色实现了进程间通信。

**14、dp 和 px 之间的关系**

dp：是 dip 的简写，指密度无关的像素。

指一个抽象意义上的像素，程序用它来定义界面元素。一个与密度无关的，在逻辑尺寸上，与一个位于像素密度为 160dpi 的屏幕上的像素是一致的。要 把 密 度 无 关 像 素 转 换 为 屏 幕 像 素 ， 可 以 用 这 样 一 个 简 单 的 公 式 ：pixels=dips\*(density/160)。举个例子，在 DPI 为 240 的屏幕上，1 个 DIP 等于 1.5 个物理像素。

布局时最好使用 dp 来定义我们程序的界面，因为这样可以保证我们的 UI在各种分辨率的屏幕上都可以正常显示。