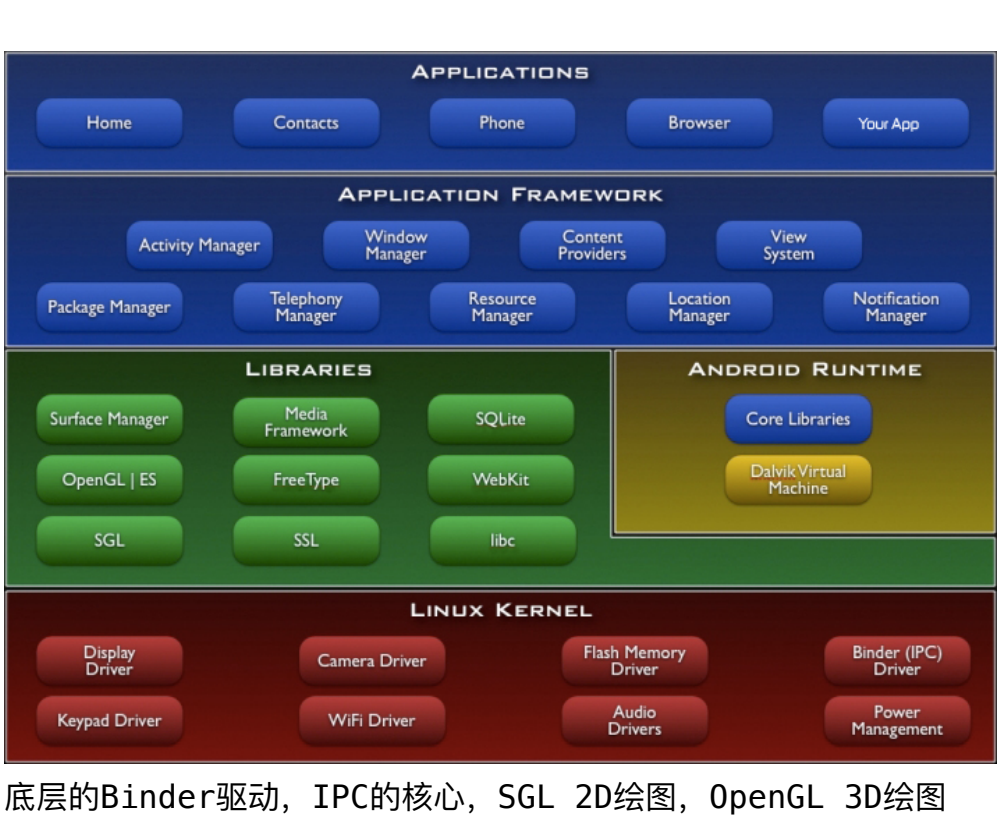
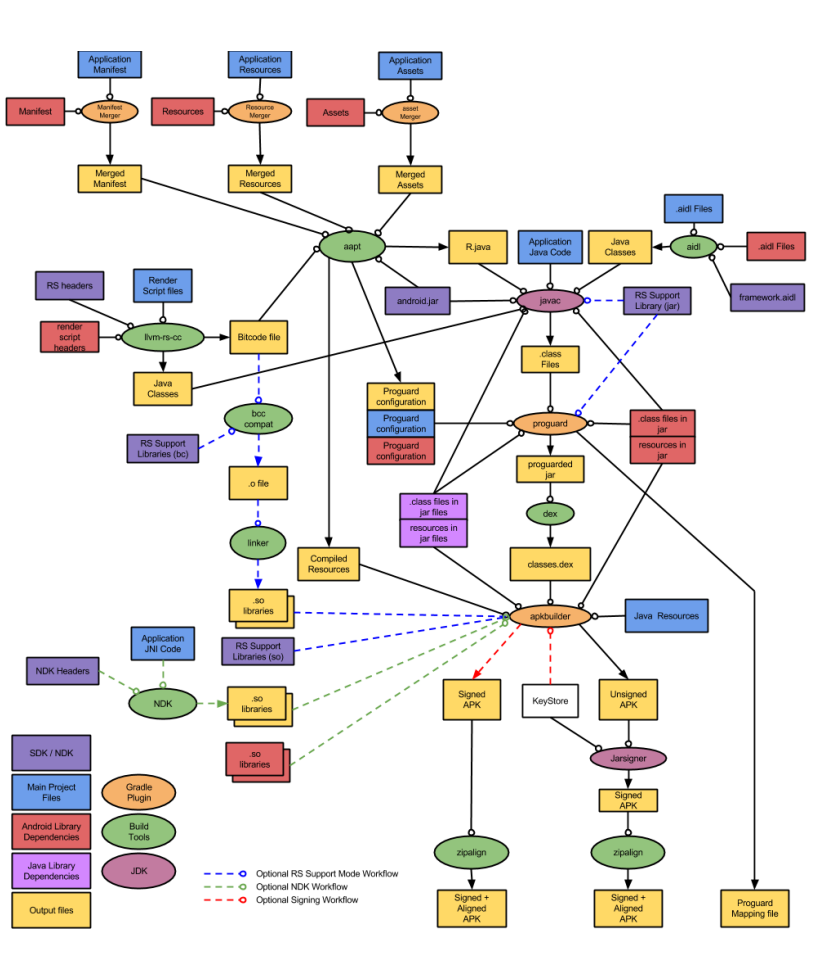
android公共技术
公共技术点之 Android 动画基础 公共技术点之 Java 动态代理_ 公共技术点之依赖注入
公共技术点之 View 事件传递 公共技术点之 View 绘制流程

1.Android的Framework 和 android apk的打包过程





底层的Binder驱动，IPC的核心，SGL 2D绘图，OpenGL 3D绘图

2.多线程
AsyncTask: asynctask配合线程池使用 AsyncTask的缺陷和问题 - 彩虹天堂 - 博客频道 - CSDN.NET
关于线程池：asynctask对应的线程池ThreadPoolExecutor都是进程范围内共享的，都是static的，所以是asynctask控制着进程范围内所有的子类实例。由于这个限制的存在，当使用默认线程池时，如果线程数超过线程池的最大容量，线程池就会爆掉(3.0后默认串行执行，不会出现这个问题)。针对这种情况，可以尝试自定义线程池，配合asyntask使用。
关于默认线程池：核心线程池中最多有CPU_COUNT+1个，最多有CPU_COUNT*2+1个，线程等待队列的最大等待数为128，但是可以自定义线程池。线程池是由AsyncTask来管理的，线程池允许tasks并行运行，需要注意的是并发情况下数据的一致性问题，新数据可能会被老数据覆盖掉，类似volatile变量。所以如果希望tasks能够串行运行的话，使用SERIAL_EXECUTOR。
自定义线程池：executeOnExecutor(Executor exec, Params... params) 自定义Executor
execute(Params... params) {return executeOnExecutor(sDefaultExecutor, params);}

AsyncTask在不同SDK版本中的区别 调用AsyncTask的excute方法不能立即执行程序的原因分析及改善方案 | 张明云的博客
通过查阅官方文档发现，AsyncTask首次引入时，异步任务是在一个独立的线程中顺序地执行，也就是说一次只能执行一个任务，不能并行地执行，从1.6开始，AsyncTask中引入了线程池，支持同时执行5个异步任务，也就是说同时只能有5个线程运行，超过的线程只能等待，等待前面的线程某个执行完了才被调度和运行。换句话说，如果一个进程中的AsyncTask实例个数超过5个，那么假如前5个都运行很长时间的话，那么第6个只能等待机会了。这是AsyncTask的一个限制，而且对于2.3以前的版本无法解决。如果你的应用需要大量的后台线程去执行任务，那么你只能放弃使用AsyncTask，自己创建线程池来管理Thread，或者干脆不用线程池直接使用Thread也无妨。不得不说，虽然AsyncTask较Thread使用起来比较方便，但是它最多只能同时运行5个线程，这也大大局限了它的实力，你必须要小心的设计你的应用，错开使用AsyncTask的时间，尽力做到分时，或者保证数量不会大于5个，否则就可能遇到上面提到的问题。可能是Google意识到了AsyncTask的局限性了，从Android 3.0开始对AsyncTask的API做出了一些调整：每次只启动一个线程执行一个任务，完成之后再执行第二个任务，也就是相当于只有一个后台线程在执行所提交的任务。

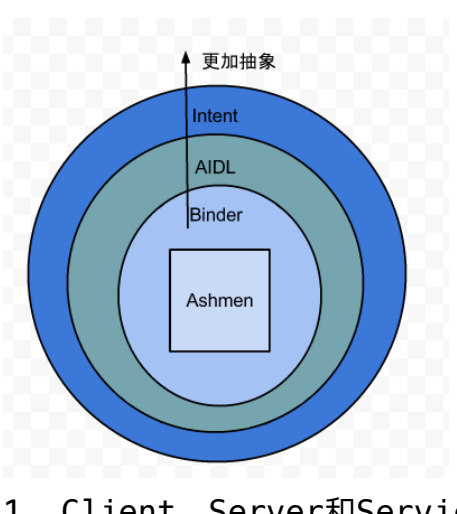
1、生命周期
很多开发者会认为一个在Activity中创建的AsyncTask会随着Activity的销毁而销毁。然而事实并非如此。AsyncTask会一直执行，直到doInBackground()方法执行完毕。然后，如果cancel(boolean)被调用，那么onCancelled(Result result)方法会被执行；否则，执行onPostExecute(Result result)方法。如果我们的Activity销毁之前，没有取消 AsyncTask，这有可能让我们的AsyncTask崩溃(crash)。因为它想要处理的view已经不存在了。所以，我们总是必须确保在销毁活动之前取消任务。总之，我们使用AsyncTask需要确保AsyncTask正确地取消。
另外，即使我们正确地调用了cancle() 也未必能真正地取消任务。因为如果在doInBackgroud里有一个不可中断的操作，比如BitmapFactory.decodeStream()，那么这个操作会继续下去。
2、内存泄漏
如果AsyncTask被声明为Activity的非静态的内部类，那么AsyncTask会保留一个对Activity的引用。如果Activity已经被销毁，AsyncTask的后台线程还在执行，它将继续在内存里保留这个引用，导致Activity无法被回收，引起内存泄露。
3、结果丢失
屏幕旋转或Activity在后台被系统杀掉等情况会导致Activity的重新创建，之前运行的AsyncTask会持有一个之前Activity的引用，这个引用已经无效，这时调用onPostExecute()再去更新界面将不再生效。
4、并行还是串行
在Android 1.6之前的版本，AsyncTask是串行的，在1.6至2.3的版本，改成了并行的。在2.3之后的版本又做了修改，可以支持并行和串行，当想要串行执行时，直接执行execute()方法，如果需要并行执行，则要执行executeOnExecutor(Executor)。

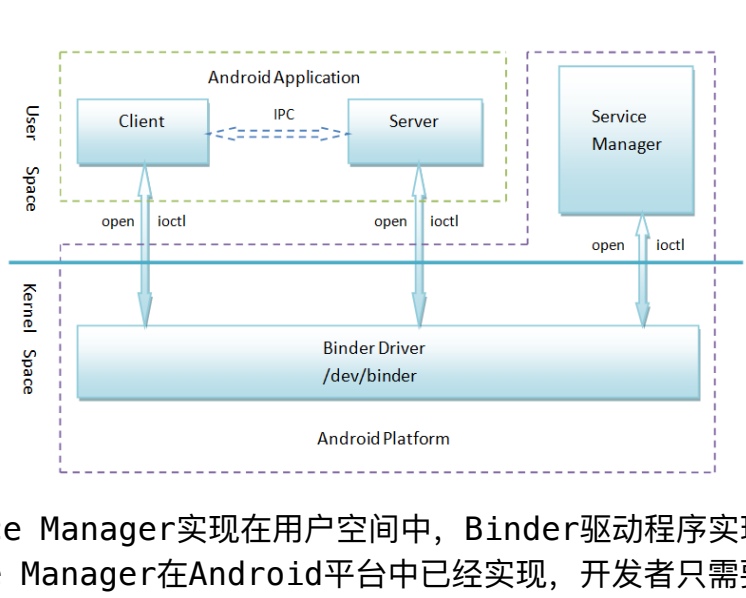
3.android安全机制
（1）Linux Sandbox 沙箱机制：android将数据分为system和data两个区。其中system是只读的，data用来存放应用自己的数据，这保证了系统数据不会被随意改写。
应用之间的数据相互独立，每个应用都会有一个user id和group id，只有相同的user id并且来自同一个作者，才能访问它们的数据。作者通过对apk签名来标识自己，签名和uid构成了双重的保证。
（2）用户权限机制：文件权限，UID，GID
（3）应用权限机制：android permission机制限制应用访问特定的资源，例如照相机、网络、外部存储等api
如何让两个app运行在同一个进程里？1.两个app要用相同的private key来签名；2.两个app的Manifest文件中要添加一样的属性 android:sharedUserId （设置成相同的UID）

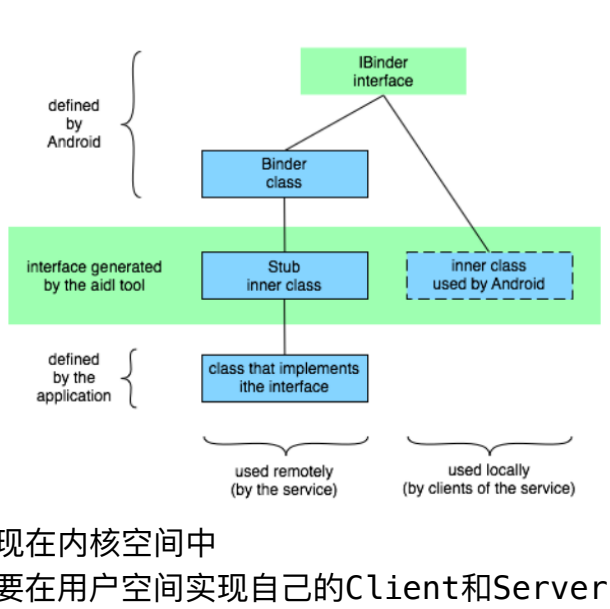
4.Binder机制
跨进程通信（IPC）：四大组件之间通过Intent相互跳转，android实现IPC的方式是binder机制！
android中的跨进程通信的实现（一）——远程调用过程和aidl
Android中的Binder机制的简要理解_Linux编程_Linux公社-Linux系统门户网站
Android中的Binder机制的简要理解二 - sunxingzhesunjinbiao的专栏 - 博客频道 - CSDN.NET
In the Android platform, the binder is used for nearly everything that happens across processes in the core platform.

最底层的是android的ashmen（Anonymous shared memoryy）机制，它负责辅助实现内存的分配，以及跨进程所需要的内存共享。AIDL（android interface definition language）对Binder的使用进行了封装，可以让开发者方便的进行方法的远程调用，后面会详细介绍。Intent是最高一层的抽象，方便开发者进行常用的跨进程调用。

从英文字面上意思看，Binder具有粘结剂的意思，那么它把什么东西粘结在一起呢？在Android系统的Binder机制中，由一系统组件组成，分别是Client、Server、Service Manager和Binder驱动程序，其中Client、Server和Service Manager运行在用户空间，Binder驱动程序运行内核空间。Binder就是一种把这四个组件粘合在一起的粘结剂了，其中，核心组件便是Binder驱动程序了，Service Manager提供了辅助管理的功能，Client和Server正是在Binder驱动和Service Manager提供的基础设施上，进行Client-Server之间的通信。







1. Client、Server和Service Manager实现在用户空间中，Binder驱动程序实现在内核空间中
2. Binder驱动程序和Service Manager在Android平台中已经实现，开发者只需要在用户空间实现自己的Client和Server
3. Binder驱动程序提供设备文件/dev/binder与用户空间交互，Client、Server和Service Manager通过open和ioctl文件操作函数与Binder驱动程序进行通信
4. Client和Server之间的进程间通信通过Binder驱动程序间接实现
5. Service Manager是一个守护进程，用来管理Server，并向Client提供查询Server接口的能力

服务器端。一个Binder服务器端就是一个Binder类的对象。当创建一个Binder对象后，内部就会开启一个线程，这个线程用于接收binder驱动发送的信息，收到消息后，会执行相关的服务代码。
Binder驱动。当服务端成功创建一个Binder对象后，Binder驱动也会相应创建一个mRemote对象，该对象的类型也是Binder类。客户就可以借助这个mRemote对象来访问远程服务。
客户端。客户端要想访问Binder的远程服务，就必须获取远程服务的Binder对象在binder驱动层对应的mRemote引用。当获取到mRemote对象的引用后，就可以调用相应Binder对象的服务了。
在这里，我们可以看到，客户端是通过Binder驱动来调用服务端的相关服务。首先，在服务端创建一个Binder对象，然后相应在Binder驱动中创建一个Binder对象，接着客户端通过获取Binder驱动中Binder对象的引用来调用服务端的服务。在Binder机制中正是借着Binder驱动将不同进程间的组件bind（粘连）在一起，实现通信。

mmap将一个文件或者其它对象映射进内存。文件被映射到多个页上，如果文件的大小不是所有页的大小之和，最后一个页不被使用的空间将会清零。munmap执行相反的操作，删除特定地址区域的对象映射。
当使用mmap映射文件到进程后，就可以直接操作这段虚拟地址进行文件的读写等操作，不必再调用read，write等系统调用。但需注意，直接对该段内存写时不会写入超过当前文件大小的内容。
采用共享内存通信的一个显而易见的好处是效率高，因为进程可以直接读写内存，而不需要任何数据的拷贝。对于像管道和消息队列等通信方式，则需要在内核和用户空间进行四次的数据拷贝，而共享内存则只拷贝两次数据：一次从输入文件到共享内存区，另一次从共享内存区到输出文件。实际上，进程之间在共享内存时，并不总是读写少量数据后就解除映射，有新的通信时，再重新建立共享内存区域。而是保持共享区域，直到通信完毕为止，这样，数据内容一直保存在共享内存中，并没有写回文件。共享内存中的内容往往是在解除映射时才写回文件的。因此，采用共享内存的通信方式效率是非常高的。

aidl主要就帮助我们完成了包装数据和解包的过程，并调用了transact过程，而用来传递的数据包我们就称为parcel
AIDL：xxx.aidl -> xxx.java，注册service
1.用aidl定义需要被调用方法接口；2.实现这些方法；3.调用这些方法。

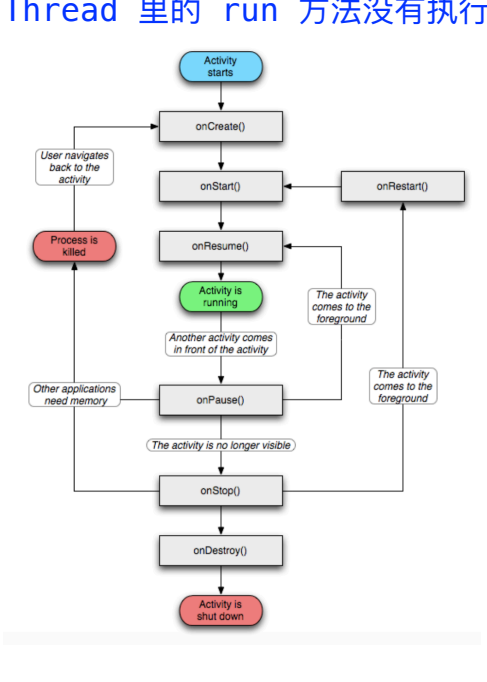
5.NDK
Dalvik虚拟机在调用一个成员函数的时候，如果发现该成员函数是一个JNI方法，那么就会直接跳到它的地址去执行。也就是说，JNI方法是直接在本地操作系统上执行的，而不是由Dalvik虚拟机解释器执行。由此也可看出，JNI方法是Android应用程序与本地操作系统直接进行通信的一个手段。
**JNI原理：[Dalvik虚拟机JNI方法的注册过程分析 - 老罗的Android之旅 - 博客频道 - CSDN.NET]
例子：当libnanosleep.so文件被加载的时候，函数JNI_OnLoad就会被调用。在函数JNI_OnLoad中，参数vm描述的是当前进程中的Dalvik虚拟机，通过调用它的成员函数GetEnv就可以获得一个JNIEnv对象。有了这个JNIEnv对象之后，我们就可以调用另外一个函数jniRegisterNativeMethods来向当前进程中的Dalvik虚拟机注册一个JNI方法。

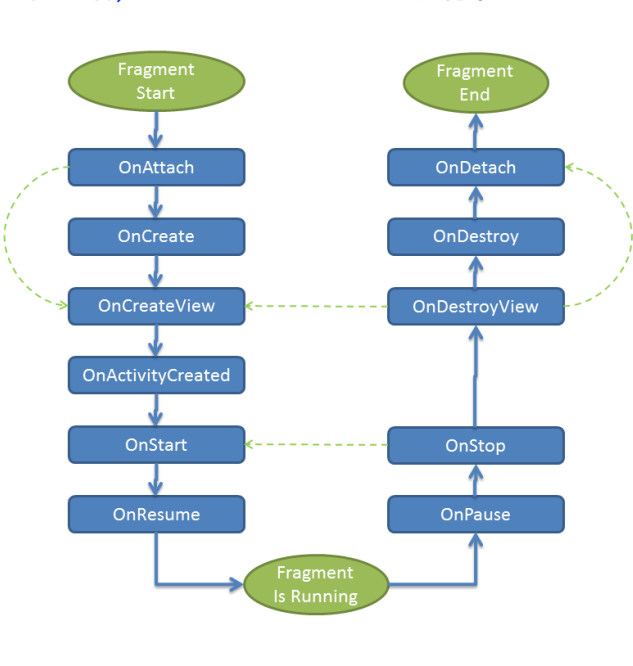
6.android系统启动过程，app启动过程
app启动过程：android activity启动过程 从桌面点击到activity启动的过程
1、Launcher线程捕获onclick的点击事件，调用Launcher.startActivitySafely，进一步调用Launcher.startActivity，最后调用父类Activity的startActivity。
2、Activity和ActivityManagerService交互，引入Instrumentation，将启动请求交给Instrumentation，调用Instrumentation.execStartActivity。
3、调用ActivityManagerService的startActivity方法，这里做了进程切换（具体过程请查看源码）。
4、开启Activity，调用onCreate方法。

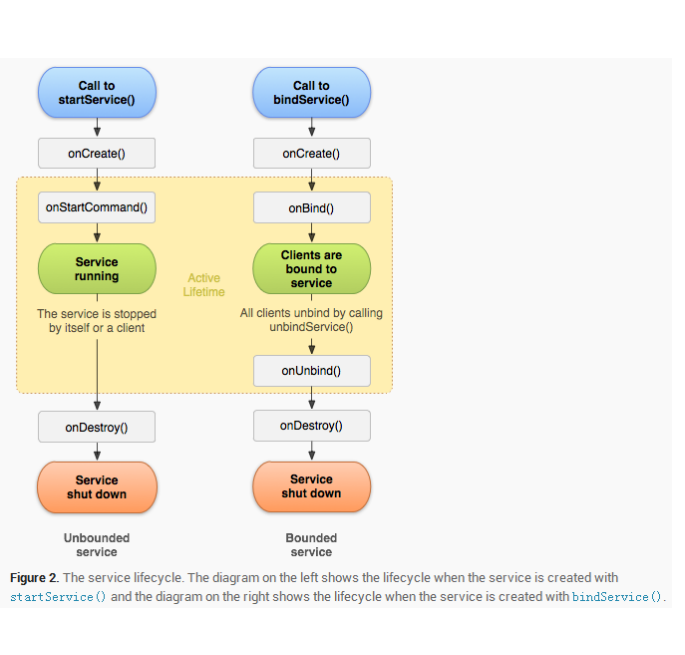
7.Activity，Fragment，Service生命周期
常见的例子：程序正运行着来电话了，这个程序咋办?中止了呗，如果中止的时候新出的一个Activity是全屏的onPause->onStop，恢复的时候 onStart->onResume，如果打断这个应用程序的是一个Theme为Translucent或者Dialog的Activity那么只是onPause，恢复的时候onResume。
onCreate：在这里创建界面，做一些数据的初始化工作
onStart：到这一步变成用户可见不可交互的
onResume：变成和用户可交互的
onPause：到这一步是可见但不可交互的，系统会停止动画等消耗CPU的事情，应该在这里保存你的一些数据，因为这个时候你的程序的优先级降低，有可能被系统收回。在这里保存的数据，应该在 onResume 里读出来。
注意:这个方法里做的事情时间要短，因为下一个 activity不会等到这个方法完成才启动
onstop：变得不可见 ，被下一个activity覆盖了 （onPause和onStop的区别是是否可见）
onDestroy：这是 activity 被干掉前最后一个被调用方法了，可能是外面类调用 finish 方法或者是系统为了节省空间将它暂时性的干掉，可以用 isFinishing()来判断它，如果你有一个 Progress Dialog 在线程中转动，请在onDestroy里把它cancel掉，不然等线程结束的时候，调用 Dialog 的 cancel 方法会抛异常的。
onPause，onstop， onDestroy，三种状态下 activity 都有可能被系统干掉
**启动另一个Activity然后finish，先调用旧Activity的onPause方法，然后调用新的Activity的onCreate->onStart->onResume方法，然后调用旧Activity的onStop->onDestory方法。
如果没有调用finish那么onDestory方法不会被调用，而且在onStop之前还会调用onSaveInstanceState方法
**onRestart方法执行完了之后还会调用onStart方法

fragment：【supportFragmentManager，childFragment】

service：Android Service的生命周期 - 圣骑士wind - 博客园 android-Service和Thread的区别 - 路人浅笑 - 博客园
Service和Intent Service：没啥区别，只是IntentService在onCreate方法中开启新的HandlerThread去执行
service运行的进程和线程：当它运行的时候如果是Local Service，那么对应的Service 是运行在主进程的 main 线程上的。如：onCreate，onStart 这些函数在被系统调用的时候都是在主进程的 main 线程上运行的。如果是Remote Service，那么对应的 Service 则是运行在独立进程的 main 线程上。
1.服务不是单一的进程。服务没有自己的进程，应用程序可以不同，服务运行在相同的进程中。
2.服务不是线程，可以在线程中工作。
3.在应用中，如果是长时间的在后台运行，而且不需要交互的情况下，使用服务。
4.同样是在后台运行，不需要交互的情况下，如果只是完成某个任务，之后就不需要运行，而且可能是多个任务，需要长时间运行的情况下使用线程。
5.如果任务占用CPU时间多，资源大的情况下，要使用线程。
Thread 的运行是独立于 Activity 的，也就是说当一个 Activity 被 finish 之后，如果你没有主动停止 Thread 或者 Thread 里的 run 方法没有执行完毕的话，Thread 也会一直执行。







8.View绘画机制
View的绘制主要涉及三个方法：onMeasure()、onLayout()和onDraw()。
（1）onMeasure主要用于计算view的大小，onLayout主要用于确定view在ContentView中的位置，onDraw主要是绘制view；
（2）在执行onMeasure()、onLayout()方法时都会先通过相应的标志位或者对应的坐标点来判断是否需要执行对应的函数，如我们经常调用的invalidate方法就只会执行onDraw方法，因为此时的视图大小和位置均未发生改变，除非调用requestLayout方法完整强制进行view的绘制，从而执行上面三个方法。
进度条组件：https://github.com/hujiaweibujidao/ProgressView
文字标注组件：https://github.com/hujiaweibujidao/Annotation

9.事件传递机制
android 事件处理机制总结，ScrollView ViewPager ListView GridView嵌套小结
当手指触摸到屏幕时，系统就会调用相应View的onTouchEvent，并传入一系列的action。

dispatchTouchEvent的执行顺序为：
首先触发ACTIVITY的dispatchTouchEvent,然后触发ACTIVITY的onUserInteraction
然后触发LAYOUT的dispatchTouchEvent,然后触发LAYOUT的onInterceptTouchEvent
这就解释了重写ViewGroup时必须调用super.dispatchTouchEvent();

（1）dispatchTouchEvent:
此方法一般用于初步处理事件，因为动作是由此分发，所以通常会调用super.dispatchTouchEvent。
这样就会继续调用onInterceptTouchEvent，再由onInterceptTouchEvent决定事件流向。
（2）onInterceptTouchEvent:
若返回值为True事件会传递到自己的onTouchEvent();
若返回值为False传递到下一个view的dispatchTouchEvent();
（3）onTouchEvent():
若返回值为True，事件由自己处理消耗，后续动作序列让其处理；
若返回值为False，自己不消耗事件了，向上返回让其他的父view的onTouchEvent接受处理；

三大方法关系的伪代码：如果当前View拦截事件，就交给自己的onTouchEvent去处理，否则就丢给子View继续走相同的流程。

```
public boolean dispatchTouchEvent(MotionEvent ev) {
    boolean consume = false;
    if (onInterceptTouchEvent(ev)) {
        consume = onTouchEvent(ev);
    } else {
        consume = child.dispatchTouchEvent(ev);
    }
    return consume;
}
```

onTouchEvent的传递
当有多个层级的View时，在父层级允许的情况下，这个action会一直向下传递直到遇到最深层的View。所以touch事件最先调用的是最底层View的onTouchEvent，如果View的onTouchEvent接收到某个touch action并作了相应处理，最后有两种返回方式return true和return false；return true会告诉系统当前的View需要处理这次的touch事件，以后的系统发出的ACTION_MOVE，ACTION_UP还是需要继续监听并接收的，而且这次的action已经被处理掉了，父层的View是不可能触发onTouchEvent了。所以每一个action最多只能有一个onTouchEvent接口返回true。如果return false，便会通知系统，当前View不关心这一次的touch事件，此时这个action会传向父级，调用父级View的onTouchEvent。但是这一次的touch事件之后发出的任何action，该View都不会再接受，onTouchEvent在这一次的touch事件中再也不会触发，也就是说一旦View返回false，那么之后的ACTION_MOVE，ACTION_UP等ACTION就不会在传入这个View，但是下一次touch事件的action还是会传进来的。

父层的onInterceptTouchEvent截获
前面说了底层的View能够接收到这次的事件有一个前提条件：在父层级允许的情况下。假设不改变父层级的dispatch方法，在系统调用底层onTouchEvent之前会先调用父View的onInterceptTouchEvent方法判断，父层View是不是要截获本次touch事件之后的action。如果onInterceptTouchEvent返回了true，那么本次touch事件之后的所有action都不会再向深层的View传递，统统都会传给父层View的onTouchEvent，就是说父层已经截获了这次touch事件，之后的action也不必询问onInterceptTouchEvent，在这次的touch事件之后发出的action时onInterceptTouchEvent不会再次调用，直到下一次touch事件的来临。如果onInterceptTouchEvent返回false，那么本次action将发送给更深层的View，并且之后的每一次action都会询问父层的onInterceptTouchEvent需不需要截获本次touch事件。只有ViewGroup才有onInterceptTouchEvent方法，因为一个普通的View肯定是位于最深层的View，touch事件能够传到这里已经是最后一站了，肯定会调用View的onTouchEvent。

底层View的getParent().requestDisallowInterceptTouchEvent(true)
对于底层的View来说，有一种方法可以阻止父层的View截获touch事件，就是调用getParent().requestDisallowInterceptTouchEvent(true);方法。一旦底层View收到touch的action后调用这个方法那么父层View就不会再调用onInterceptTouchEvent了，也无法截获以后的action（如果父层ViewGroup和最底层View需要截获不同焦点，或不同手势的touch，不能使用这个写死）。

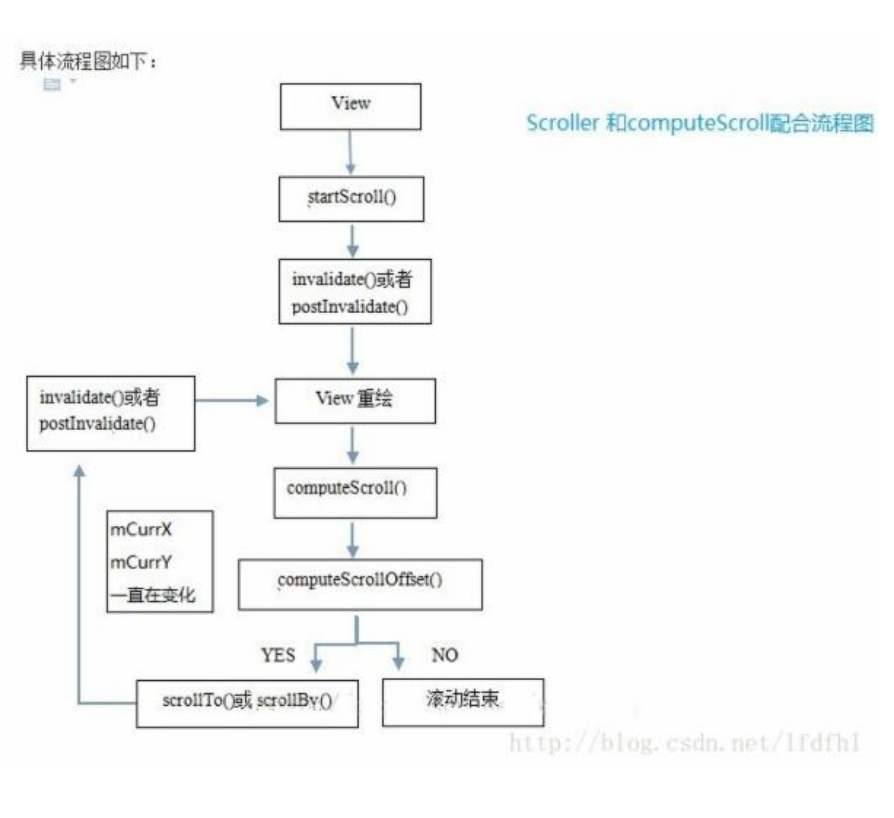
曾经开发过程中遇到的两个示例：左边是处理ViewPager和Listview的冲突：记录水平和垂直方向的偏移量，如果水平方向的偏移量更多的话就让viewpager处理page滑动
右边是处理viewpager和imagebanner的滚动冲突：同样是记录偏移量，如果发生在ImageBanner上的水平偏移量大于垂直偏移量的话就让banner滚动
**想想为什么右边是重写dispatchTouchEvent方法而不是onInterceptTouchEvent方法?

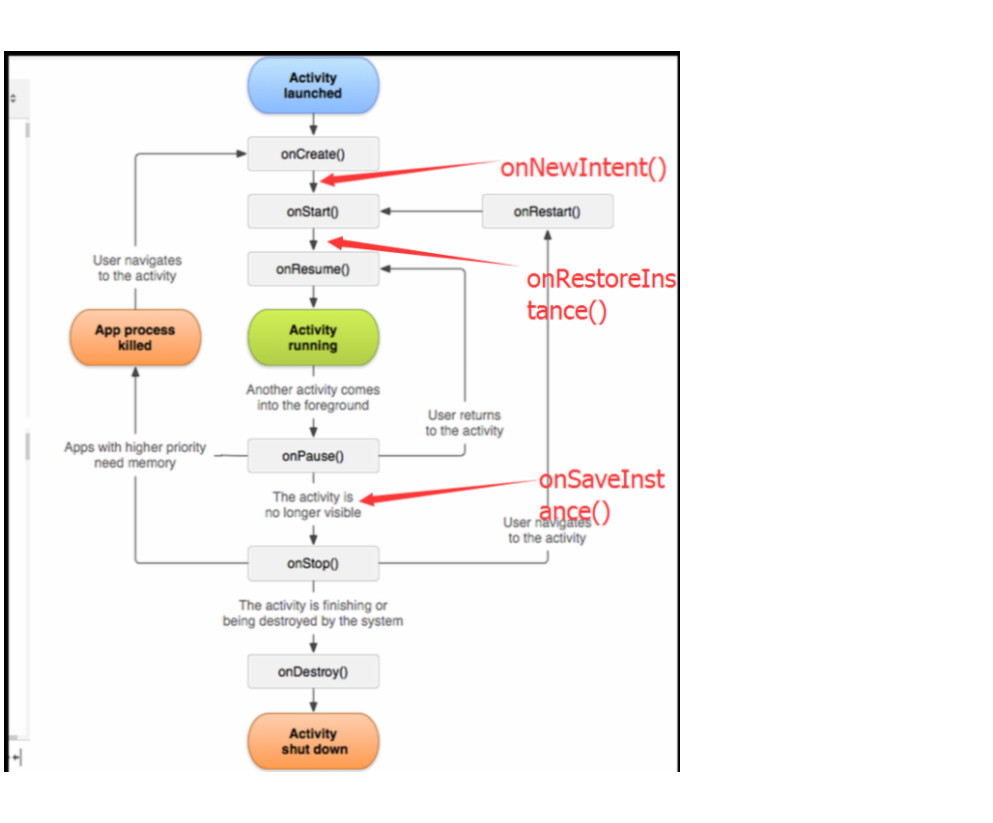
```
FixedViewPager
@Override
public boolean
onInterceptTouchEvent(MotionEvent ev) {

    switch(ev.getAction() &
MotionEvent.ACTION_MASK) {
        case MotionEvent.ACTION_DOWN:
            mX = ev.getX();
            mY = ev.getY();
            break;
        case MotionEvent.ACTION_MOVE:
            float x = ev.getX();
            float y = ev.getY();

            float dX = x - mX;
            float dY = y - mY;
            float tmp = Math.abs(dX) /
Math.abs(dY);

            mX = x;
            mY = y;
            if (tmp > 1) {
                return true;
            } else {
                return
super.onInterceptTouchEvent(ev);
            }
    }
    return super.onInterceptTouchEvent(ev);
}
```

```
FixedImageLoadBanner
@Override
public boolean dispatchTouchEvent(MotionEvent
ev) {
    if (0 != mX || 0 != mY) {
        float dY = ev.getRawY() - mY;
        float dX = ev.getRawX() - mX;
        if (Math.abs(dY) > Math.abs(dX)) {

requestDisallowInterceptTouchEvent(false);
        } else {

requestDisallowInterceptTouchEvent(true);
        }
    }
    mX = ev.getRawX();
    mY = ev.getRawY();

    return super.dispatchTouchEvent(ev);
}
```

10.ART和Dalvik区别
art上应用启动快，运行快，但是耗费更多存储空间，安装时间长，总的来说ART的功效就是"空间换时间"。
ART: Ahead of Time Dalvik: Just in Time
什么是Dalvik:Dalvik是Google公司自己设计用于Android平台的Java虚拟机。Dalvik虚拟机是Google等厂商合作开发的Android移动设备平台的核心组成部分之一。它可以支持已转换为 .dex（即Dalvik Executable）格式的Java应用程序的运行，.dex格式是专为Dalvik设计的一种压缩格式，适合内存和处理器速度有限的系统。Dalvik 经过优化，允许在有限的内存中同时运行多个虚拟机的实例，并且每一个Dalvik 应用作为一个独立的Linux 进程执行。独立的进程可以防止在虚拟机崩溃的时候所有程序都被关闭。
什么是ART:Android操作系统已经成熟，Google的Android团队开始将注意力转向一些底层组件，其中之一是负责应用程序运行的Dalvik运行时。Google开发者已经花了两年时间开发更快执行效率更高更省电的替代ART运行时。 ART代表Android Runtime，其处理应用程序执行的方式完全不同于Dalvik，Dalvik是依靠一个Just-In-Time (JIT)编译器去解释字节码。开发者编译后的应用代码需要通过一个解释器在用户的设备上运行，这一机制并不高效，但让应用能更容易在不同硬件和架构上运行。ART则完全改变了这套做法，在应用安装时就预编译字节码到机器语言，这一机制叫Ahead-Of-Time (AOT) 编译。在移除解释代码这一过程后，应用程序执行将更有效率，启动更快。
ART优点：1、系统性能的显著提升。2、应用启动更快、运行更快、体验更流畅、触感反馈更及时。3、更长的电池续航能力。4、支持更低的硬件。
ART缺点：1、更大的存储空间占用，可能会增加10%-20%。2、更长的应用安装时间。

11.Scroller原理
Scroller执行流程里面的三个核心方法
（1）mScroller.startScroll() （2）mScroller.computeScrollOffset() （3）view.computeScroll()
1，在mScroller.startScroll()中为滑动做了一些初始化准备。
比如：起始坐标，滑动的距离和方向以及持续时间(有默认值)，动画开始时间等
2，mScroller.computeScrollOffset()方法主要是根据当前已经消逝的时间来计算当前的坐标点。
因为在mScroller.startScroll()中设置了动画时间，那么在computeScrollOffset()方法中依据已经消逝的时间就很容易得到当前时刻应该所处的位置并将其保存在变量mCurrX和mCurrY中。除此之外该方法还可判断动画是否已经结束。





12.Activity Manager Service， ActivityThread

13.Android几种进程
（1）前台进程：即与用户正在交互的Activity或者Activity用到的Service等，如果系统内存不足时前台进程是最后被杀死的；
（2）可见进程：可以是处于暂停状态（onPause）的Activity或者绑定在其上的Service，即被用户可见，但由于失了焦点而不能与用户交互；
（3）服务进程：其中运行着使用startService方法启动的Service，虽然不被用户可见，但是却是用户关系的，例如用户正在非音乐界面听的音乐或者正在非下载页面自己下载的文件等；当系统要用空间运行前两者进程时才会被终止；
（4）后台进程：其中运行着执行onStop方法而停止的程序，但是却不是用户当前关心的，例如后台挂着的QQ，这样的进程系统一旦没有内存就首先被杀死；
（5）空进程：不包含任何应用程序的程序组件的进程，这样的进程系统是一般不会让他存在的；
如何避免后台进程被杀死？
1. 调用startForegound，让你的Service所在的进程成为前台进程
2. Service的onStartCommond返回START_STICKY或START_REDELIVER_INTENT
3. Service的onDestory里面重新启动自己

14.Activity启动模式
standard：Activity的默认加载方法，该方法会通过跳转到一个新的activity，同时将该实例压入到栈中（不管该activity是否已经存在在Task栈中，都是采用new操作，生命周期从onCreate()开始）。例如： 栈中顺序是A B C D ，此时D通过Intent跳转到A，那么栈中结构就变成 A B C D A ，点击返回按钮的 显示顺序是 D C B A，依次摧毁。
singleTop：singleTop模式下，当前Activity D位于栈顶的时候，如果通过Intent跳转到它本身的Activity （即D），那么不会重新创建一个新的D实例（走onNewIntent()），所以栈中的结构依旧为A B C D，如果跳转到B，那么由于B不处于栈顶，所以会新建一个B实例并压入到栈中，结构就变成了A B C D B。应用实例：三条推送，点进去都是一个activity，这肯定用singletop啊
singleTask：singleTask模式下，Task栈中只能有一个对应Activity的实例。例如：现在栈的结构为：A B C D。此时D通过Intent跳转到B（走onNewIntent()），则栈的结构变成了：A B。其中的C和D被栈弹出销毁了，也就是说位于B之上的实例都被销毁了。通常应用于首页，首页肯定得在栈底部，也只能在栈底部。
singleInstance：singleInstance模式下，会将打开的Activity压入一个新建的任务栈中。例如：Task栈1中结构为：A B C ，C通过Intent跳转到了D（D的模式为singleInstance），那么则会新建一个Task 栈2，栈1中结构依旧为A B C，栈2中结构为D，此时屏幕中显示D，之后D通过Intent跳转到D，栈2中不会压入新的D，所以2个栈中的情况没发生改变。如果D跳转到了C，那么就会根据C对应的launchMode的在栈1中进行对应的操作，C如果为standard，那么D跳转到C，栈1的结构为A B C C ，此时点击返回按钮，还是在C，栈1的结构变为A B C，而不会回到D。

launchMode为singleTask的时候，通过Intent启动到一个Activity，如果系统已经存在一个实例，系统就会将请求发送到这个实例上，但这个时候，系统就不会再调用通常情况下我们处理请求数据的onCreate方法，而是调用onNewIntent方法。

onSaveInstanceState的调用遵循一个重要原则，即当系统"未经你许可"时销毁了你的activity，则onSaveInstanceState会被系统调用，这是系统的责任，因为它必须要提供一个机会让你保存你的数据。至于onRestoreInstanceState方法，需要注意的是，onSaveInstanceState方法和onRestoreInstanceState方法"不一定"是成对的被调用的。
onRestoreInstanceState被调用的前提是，activity A"确实"被系统销毁了，而如果仅仅是停留在有这种可能性的情况下，则该方法不会被调用，例如，当正在显示activity A的时候，用户按下HOME键回到主界面，然后用户紧接着又返回到activity A，这种情况下activity A一般不会因为内存的原因被系统销毁，故activity A的onRestoreInstanceState方法不会被执行。
另外，onRestoreInstanceState的bundle参数也会传递到onCreate方法中，你也可以选择在onCreate方法中做数据还原。

onSaveInstanceState(Bunble bundle)通常和onRestoreInstanceState(Bunble bundle)不会成对出现，onRestoreInstanceState这玩意儿不太好触发，给大家提个好办法，横竖屏切换的时候100%会触发。然后保存在onRestoreInstanceState bundle里面的数据，就是onCreate的那个参数bundle啦，要怎么恢复就看开发者了。

15.TMImageView图片库的设计
一种Andriod安卓图片库中间层设计（快速切换引擎、杜绝OOM、视觉统一、极简接入）
Feature机制

16.listview优化
1. 首先，虽然大家都知道，还是提一下，利用好 convertView 来重用 View，切忌每次 getView() 都新建。ListView 的核心原理就是重用 View。ListView 中有一个回收器，Item 滑出界面的时候 View 会回收到这里，需要显示新的 Item 的时候，就尽量重用回收器里面的 View。
2. 利用好 View Type，例如你的 ListView 中有几个类型的 Item，需要给每个类型创建不同的 View，这样有利于 ListView 的回收，当然类型不能太多；
3. 尽量让 ItemView 的 Layout 层次结构简单，这是所有 Layout 都必须遵循的；
4. 善用自定义 View，自定义 View 可以有效的减小 Layout 的层级，而且对绘制过程可以很好的控制；
5. 尽量能保证 Adapter 的 hasStableIds() 返回 true，这样在 notifyDataSetChanged() 的时候，如果 id 不变，ListView 将不会重新绘制这个 View，达到优化的目的；
6. 每个 Item 不能太高，特别是不要超过屏幕的高度，可以参考 Facebook 的优化方法，把特别复杂的 Item 分解成若干小的 Item，特别推荐看一下这个文章：code.facebook.com/posts...
7. 为了保证 ListView 滑动的流畅性，getView() 中要做尽量少的事情，不要有耗时的操作。特别是滑动的时候不要加载图片，停下来再加载，这个库可以帮助你 Glide：github.com/bumptech/gli...
8. 使用 RecycleView 代替。 ListView 每次更新数据都要 notifyDataSetChanged()，有些太暴力了。RecycleView 在性能和可定制性上都有很大的改善，推荐使用。
9. 有时候，需要从根本上考虑，是否真的要使用 ListView 来实现你的需求，或者是否有其他选择?

17. webview
如何使用webview在js中调用java方法?
webView.addJavascriptInterface(new Object(){xxx},"xxx");
答案：可以使用 WebView 控件执行 JavaScript 脚本，并且可以在 JavaScript 中执行 Java 代码。要想让 WebView 控件执行 JavaScript，需要调用 WebSettings.setJavaScriptEnabled 方法，代码如下：

```
WebView webView = (WebView) findViewById(R.id.webview);
WebSettings webSettings = webView.getSettings();
// 设置 WebView 支持 JavaScript
webSettings.setJavaScriptEnabled(true);
webView.setWebChromeClient(new WebChromeClient());
```

JavaScript 调用 Java 方法需要使用 WebView.addJavascriptInterface 方法设置 JavaScript 调用的 Java 方法，代码如下：

```
webView.addJavascriptInterface(new Object()
{
    // JavaScript 调用的方法
    public String process(String value)
    {
        // 处理代码
        return result;
    }
}, "demo");          // demo 是 Java 对象映射到 JavaScript 中的对象名
```

可以使用下面的 JavaScript 代码调用 process 方法，代码如下：

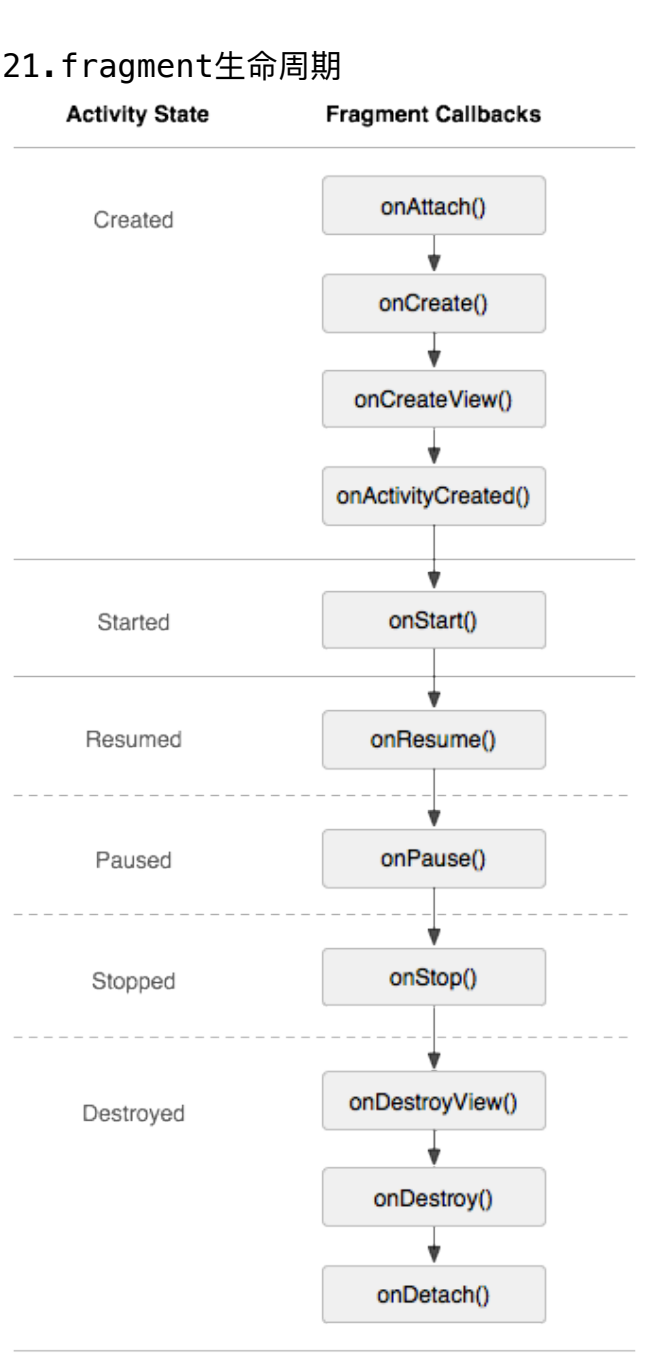
```
<script language="javascript">
    function search()
    {
        // 调用 searchWord 方法
        result.innerHTML = "<font color='red'>" + window.demo.process('data') + "</font>";
```

18.surfaceView和View最本质的区别在于：
surfaceView是在一个新起的单独线程中可以重新绘制画面，而View必须在UI的主线程中更新画面。
在UI的主线程中更新画面可能会引发问题，比如你更新画面的时间过长，那么你的主UI线程会被你正在画的函数阻塞。那么将无法响应按键，触屏等消息。当使用surfaceView 由于是在新的线程中更新画面所以不会阻塞你的UI主线程。但这也带来了另外一个问题，就是事件同步。比如你触屏了一下，你需要surfaceView中 thread处理，一般就需要有一个event queue的设计来保存touch event，这会稍稍复杂一点，因为涉及到线程同步。

19.<include> <merge> <viewstub>标签
<merge/> 和 <include/> - Chrisfang6 - 博客园 Android ViewStub的基本使用 - OPEN 开发经验库
简言之：<include> <merge>都是用来解决重复布局的问题，但是merge标签能够在布局重用的时候减少UI层级结构。
viewstub标签是用来给其他的view事先占据好位置，当需要的时候调用inflater()或者是setVisible () 方法显示这些View。

20.ANR排错
（1）ANR一般有三种类型：
1. KeyDispatchTimeout(5 seconds) --主要类型按键或触摸事件在特定时间内无响应
2. BroadcastTimeout(10 seconds) --BroadcastReceiver在特定时间内无法处理完成
3. ServiceTimeout(20 seconds) --小概率类型 Service在特定的时间内无法处理完成
（2）如何避免
1. UI线程尽量只做跟UI相关的工作
2. 耗时的工作（比如数据库操作，I/O，连接网络或者别的有可能阻碍UI线程的操作）把它放入单独的线程处理
3. 尽量用Handler来处理UIthread和别的thread之间的交互
（3）如何排查
1. 首先分析log
2. 从trace.txt文件查看调用stack，adb pull data/anr/traces.txt ./mytraces.txt
3. 看代码
4. 仔细查看ANR的成因（iowait?block?memoryleak?）
（4）监测ANR的Watchdog Android监听应用ANR

21.fragment生命周期



============================== 常见面试问题 ==============================

** 横竖屏切换时候 activity 的生命周期?
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

** （图片相关的）OOM怎么处理
Android 内存溢出解决方案（OOM） 整理总结 - 酷_莫名简单、KNothing - 51CTO技术博客
一：在内存引用上做些处理，常用的有软引用、弱引用
二：在内存中加载图片时直接在内存中做处理，如：边界压缩
三：动态回收内存
四：优化Dalvik虚拟机的堆内存分配
五：自定义堆内存大小

** 界面优化
Android开发—性能优化之如何防止过度绘制 | 阳和移动开发
太多重叠的背景 （overdraw）
这个问题其实最容易解决，建议就是检查你在布局和代码中设置的背景，有些背景是被隐藏在底下的，它永远不可能显示出来，这种没必要的背景一定要移除，因为它很可能会严重影响到app的性能。如果采用的是selector的背景，将normal状态的color设置为"@android:color/transparent"，也同样可以解决问题。
太多重叠的view
第一个建议是：使用ViewStub来加载一些不常用的布局，它是一个轻量级且默认不可见的视图，可以动态的加载一个布局，只有你用到这个重叠着的view的时候才加载，推迟加载的时间。
第二个建议是：如果使用了类似viewpager+Fragment这样的组合或者有多个Fragment在一个界面上，需要控制Fragment的显示和隐藏，尽量使用动态地Inflation view，它的性能要比SetVisiblity好。
复杂的Layout层级
这里的建议比较多一些，首先推荐用Android提供的布局工具Hierarchy Viewer来检查和优化布局。第一个建议是：如果嵌套的线性布局加深了布局层次，可以使用相对布局来取代。第二个建议是：用标签来合并布局，这可以减少布局层次。第三个建议是：用标签来重用布局，抽取通用的布局可以让布局的逻辑更清晰明了。记住，这些建议的最终目的都是使得你的Layout在Hierarchy Viewer里变得宽而浅，而不是窄而深。

** 移动端获取网络数据优化的几个点
0. 连接复用：节省连接建立时间，如开启 keep-alive。
对于 Android 来说默认情况下 HttpURLConnection 和 HttpClient 都开启了keep-alive。只是 2.2 之前 HttpURLConnection 存在影响连接池的 Bug，具体可见：Android HttpURLConnection 及 HttpClient 选择
1. 请求合并：即将多个请求合并为一个进行请求，比较常见的就是网页中的 CSS Image Sprites。如果某个页面内请求过多，也可以考虑做一定的请求合并。
2. 减少请求数据的大小：对于post请求，body可以做gzip压缩的，header也可以作数据压缩(不过只支持http 2.0)。
3. 返回的数据的body也可以作gzip压缩，body数据体积可以缩小到原来的30%左右。(也可以考虑压缩返回的json数据的key数据的体积，尤其是针对返回数据格式变化不大的情况，支付宝聊天返回的数据用到了)
4. 根据用户的当前的网络质量来判断下载什么质量的图片(电商用的比较多)