## 插件化、热修复 、热更新的理解

* 插件化 – apk 分为宿主和插件部分，插件在需要的时候才加载进来
* 热修复 – 更新的类或者插件粒度较小的时候，我们会称之为热修复，一般用于修复bug
* 热更新 – 2016 Google 的 Android Studio 推出了Instant Run 功能 同时提出了3个名词
  + “ 热部署” – 方法内的简单修改，无需重启app和Activity。
  + “暖部署” – app无需重启，但是activity需要重启，比如资源的修改。
  + “冷部署” – app需要重启，比如继承关系的改变或方法的签名变化等。
* 站在app开发者角度的“热”是指在不发版的情况来实现更新
* 而Google提出的“热”是指值是否需要重新启动。 - 同时在开发插件化的时候也有两种情景
* 一种是插件与宿主apk没有交互，只是在用户使用到的时候进行一次吊起
* 还有一种是与宿主有很多的交互

## 你认为android热更新框架哪个好:

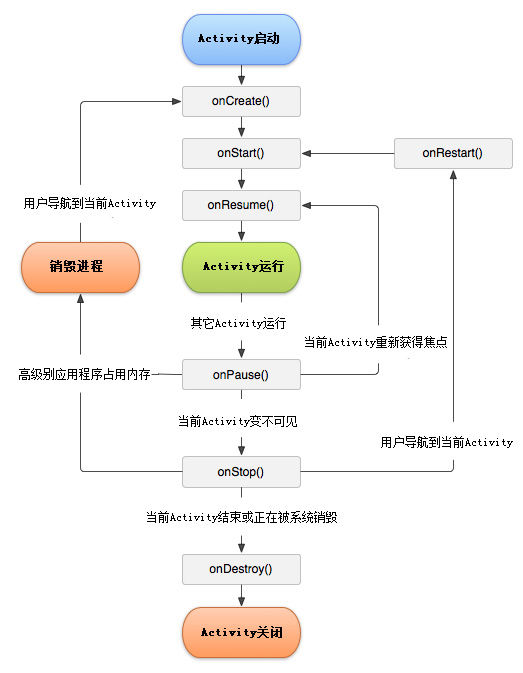
* 1.阿里的热更新框架已经开源 了。但已经很久没有更新过新版本了。当前的版本只支持到了 Android 4.4。由于 5.0 起新的 ART 虚拟机、更严格的 SELinux 策略以及对 64 位的支持之类的事，使得 Xposed 都在开发上做了很多调整。我不知道 Dexposed 现在是否支持，但至少阿里没有开源。
* 2.在本地动态执行远端下发的代码是极度危险的行为。利用此方法执行非法代码等或用于绕过 Google Play 等市场的审查是违反相关协议的，也是对用户极度不负责任的行为。
* 3.在一些访问非常密集的地方使用热更新可能会对效率产生相对比较大的影响，应该避免使用.
* 4.我们可以对 Java 的 ScriptEngine 进行一些封装成为一个 HotPatch 类使得它更适合做热更新的工作。
* 5.首先,检查热更新补丁的管道一定要建立在 https 上，因为下发代码是极其危险的，如果被劫持，后果是无法想象的。其次,请求时最好自动带上 Android 版本、手机型号、地区、版本号等信息，以方便更精确地下发，千万不能下发错。
* 6.Java在运行时加载对应的类是通过ClassLoader来实现的，ClassLoader本身是一个抽象来，Android中使用PathClassLoader类作为Android的默认的类加载器
* 7.我们的如果想做hotpatch，一定要保证我们的hotpacth dex文件出现在dexElements列表的前面。

## 二.常用的热更新技术框架:

* 基于QQ空间的HotFix →→ 要使用到android dex分包方案→拆分dex的项目的话，可以参考一下谷歌的multidex方案实现.
* 大众点评的NuWa←项目补丁自动化做的很完整
* alibaba/AndFix
* 阿里巴巴的DexPosed
* dalvik\_patch实现multidex
* 使用React-Native实现app热部署的一次实践
* alibaba/AndFix

## 基础

## activity生命周期图解



**注意**：

锁定屏与解锁屏幕 只会调用onPause()，而不会调用onStop方法，开屏后则调用onResume()。在实际操作中会有所出入，比如在三星手机测试的时候锁定手机调用了onPause()和onStop()方法，解锁时候调用的是：onRestart()，onStart()和 onResume()方法。

## 介绍不同场景下Activity生命周期的变化过程

* 启动Activity：

onCreate()—>onStart()—>onResume()，Activity进入运行状态。

* Activity退居后台：

当前Activity转到新的Activity界面或按Home键回到主屏： onPause()—>onStop()，进入停滞状态。

* Activity返回前台： onRestart()—>onStart()—>onResume()，再次回到运行状态。
* Activity退居后台，且系统内存不足，

系统会杀死这个后台状态的Activity，若再次回到这个Activity,则会走onCreate()–>onStart()—>onResume()

* 锁定屏与解锁屏幕

只会调用onPause()，而不会调用onStop方法，开屏后则调用onResume()

## Activity销毁但Task如果没有销毁掉，当Activity重启时这个AsyncTask该如何解决？

比如屏幕旋转这个例子，在重建Activity的时候，会回调

Activity.onRetainNonConfigurationInstance()

重新传递一个新的对象给AsyncTask，完成引用的更新

## 若Activity已经销毁,此时AsynTask执行完并返回结果,会报异常么?

* 当一个App旋转时，整个Activity会被销毁和重建。
* 当Activity重启时，AsyncTask中对该Activity的引用是无效的，因此onPostExecute()就不会起作用
* 若AsynTask正在执行，折会报 view not attached to window manager 异常
* 同样也是生命周期的问题，在 Activity 的onDestory()方法中调用Asyntask.cancal方法，让二者的生命周期同步

## 内存不足时,系统会杀死后台的Activity,如果需要进行一些临时状态的保存,在哪个方法进行

* Activity的 onSaveInstanceState() 和 onRestoreInstanceState()并不是生命周期方法,不同于 onCreate()、onPause()等生命周期方法，它们并不一定会被触发。
* 当应用遇到意外情况（如：内存不足、用户直接按Home键）由系统销毁一个Activity，onSaveInstanceState() 会被调用。
* 但是当用户主动去销毁一个Activity时，例如在应用中按返回键，onSaveInstanceState()就不会被调用。除非该activity是被用户主动销毁的
* 通常onSaveInstanceState()只适合用于保存一些临时性的状态，而onPause()适合用于数据的持久化保存。

## 介绍Activity 四中launchMode:

我们可以在AndroidManifest.xml配置的android:launchMode属性为以下四种之一。

* 1、**standard**

standard模式是默认的启动模式，不用为配置android:launchMode属性即可，当然也可以指定值为standard。standard启动模式，不管有没有已存在的实例，都生成新的实例。

* 2、 **singleTop**

我们在上面的基础上为指定属性android:launchMode=”singleTop”，系统就会按照singleTop启动模式处理跳转行为。跳转时系统会先在栈结构中寻找是否有一个Activity实例正位于栈顶，如果有则不再生成新的，而是直接使用。如果系统发现存在有Activity实例,但不是位于栈顶，重新生成一个实例。 这就是singleTop启动模式，如果发现有对应的Activity实例正位于栈顶，则重复利用，不再生成新的实例。

* 3、 **singleTask**

如果发现有对应的Activity实例，则使此Activity实例之上的其他Activity实例统统出栈，使此Activity实例成为栈顶对象，显示到幕前。

* 4、**singleInstance**

这种启动模式比较特殊，因为它会启用一个新的栈结构，将Acitvity放置于这个新的栈结构中，并保证不再有其他Activity实例进入。

## LaunchMode使用场景

* singleTop适合接收通知启动的内容显示页面。

例如，某个新闻客户端的新闻内容页面，如果收到10个新闻推送，每次都打开一个新闻内容页面是很烦人的。

* singleTask适合作为程序入口点。

例如浏览器的主界面。不管从多少个应用启动浏览器，只会启动主界面一次，其余情况都会走onNewIntent，并且会清空主界面上面的其他页面。

* singleInstance应用场景：

闹铃的响铃界面。 你以前设置了一个闹铃：上午6点。在上午5点58分，你启动了闹铃设置界面，并按 Home 键回桌面；在上午5点59分时，你在微信和朋友聊天；在6点时，闹铃响了，并且弹出了一个对话框形式的 Activity(名为 AlarmAlertActivity) 提示你到6点了(这个 Activity 就是以

* SingleInstance

加载模式打开的)，你按返回键，回到的是微信的聊天界面，这是因为 AlarmAlertActivity 所在的 Task 的栈只有他一个元素， 因此退出之后这个 Task 的栈空了。如果是以 SingleTask 打开 AlarmAlertActivity，那么当闹铃响了的时候，按返回键应该进入闹铃设置界面。

## 如何把一个应用设置为系统应用

* Android设置是Debug版本,且root,直接将该apk用adb工具push到system/app或system/priv-app
* 如果是非root设备,需要编译后烧写镜像
* 有些权限(如WRITE\_SECURE\_SETTINGS)不开放给第三方应用,只能在对应设备源码总编译然后作为系统app使用

## Activity,Window,View三者的联系和区别?

* 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。

## Activity启动Service的两种方式

* startService:生命周期和调用者不同.启动后若调用者未调用stopService而直接退出,Service仍会运行
* bindService:生命周期与调用者绑定,调用者一旦退出,Service就会调用unBind->onDestory

## Android两个应用能在同一个任务栈吗？

栈一般以包名命名，两个应用的签名和udid要相同

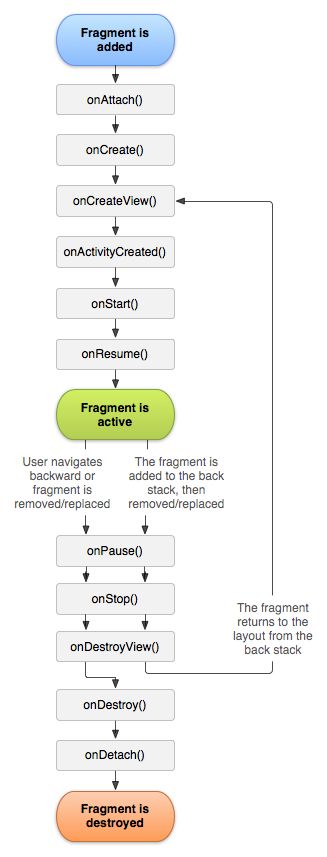
## Fragment是什么?你曾经遇到哪些有关Fragment的问题?

* Fragment可以作为Activity界面的一部分组成出现
  + 其作用是：

碎片整理，局部刷新。

* 一个Activity中可以同时出现多个Fragment,并一个Fragment也可以在多个Activity中使用.
* 在Activity中可以添加,删除,替换Fragment.Fragment可以响应自己的输入时间,并且有自己的生命周期,但其生命周期收Activity影响.

## Fragment生命周期



## 如何实现Activity窗口快速变暗

利用只读属性动画+WindowManager

/\*

\*@param from\>=0&&from\<=1.0f

\* @param to\>=0&&to\<=1.0f

\*

\* \*/

private void dimBackground(final float from, final float to) {

final Window window = getWindow();

ValueAnimator valueAnimator = ValueAnimator.ofFloat(from, to);

valueAnimator.setDuration(500);

valueAnimator.addUpdateListener(new AnimatorUpdateListener() {

@Override

public void onAnimationUpdate(ValueAnimator animation) {

WindowManager.LayoutParams params = window.getAttributes();

params.alpha = (Float) animation.getAnimatedValue();

window.setAttributes(params);

}

});

valueAnimator.start();

}

## 是否使用过本地广播,和全局广播有什么区别?

本地广播在本应用范围内传播,不用担心隐私数据泄露,不用担心别的应用伪造广播.相比全局广播,本地广播更高效.

## 注册广播的几种方法?

* 1.静态注册:在清单文件中注册， 常见的有监听设备启动，常驻注册不会随程序生命周期改变
* 2.动态注册:在代码中注册，随着程序的结束，也就停止接受广播了

补充一点：有些广播只能通过动态方式注册，比如时间变化事件、屏幕亮灭事件、电量变更事件，因为这些事件触发频率通常很高，如果允许后台监听，会导致进程频繁创建和销毁，从而影响系统整体性能

## 为什么Android引入广播机制?

* ａ:从MVC的角度考虑(应用程序内) 其实回答这个问题的时候还可以这样问，android为什么要有那4大组件，现在的移动开发模型基本上也是照搬的web那一套MVC架构，只不过是改了点嫁妆而已。
  + android的四大组件本质上就是为了实现移动或者说嵌入式设备上的MVC架构
  + 它们之间有时候是一种相互依存的关系，有时候又是一种补充关系，引入广播机制可以方便几大组件的信息和数据交互。
* b：程序间互通消息(例如在自己的应用程序内监听系统来电)
* c：效率上(参考UDP的广播协议在局域网的方便性)
* d：设计模式上(反转控制的一种应用，类似监听者模式)

## 了解IntentServices吗?

* IntentService是Service的子类，是一个异步的，会自动停止的服务，很好解决了传统的Service中处理完耗时操作忘记停止并销毁Service的问题
* 生成一个默认的且与线程相互独立的工作线程执行所有发送到onStartCommand()方法的Intent,可以在onHandleIntent()中处理.
* 串行队列,每次只运行一个任务,不存在线程安全问题,所有任务执行完后自动停止服务,不需要自己手动调用stopSelf()来停止.

## 如何提升Service进程优先级

在AndroidManifest.xml文件中对于intent-filter可以通过android:priority = “1000”这个属性设置最高优先级，1000是最高值，如果数字越小则优先级越低，同时适用于广播。

ContentProvider和sql的区别

ContentProvider的主要还是用于数据共享，其可以对Sqlite，SharePreferences，File等进行数据操作用来共享数据。而sql的可以理解为数据库的一门语言，可以使用它完成CRUD等一系列的操作

## 数据存储相关

* 文件存储：

通过Java.io.FileInputStream和java.io.FileOutputStream这两个类来实现对文件的读写，java.io.File类则用来构造一个具体指向某个文件或者文件夹的对象。

* SharedPreferences：

SharedPreferences是一种轻量级的数据存储机制，他将一些简单的数据类型的数据，包括boolean类型，int类型，float类型，long类型以及String类型的数据，以键值对的形式存储在应用程序的私有Preferences目录（/data/data/<包名>/shared\_prefs/）中，这种Preferences机制广泛应用于存储应用程序中的配置信息。

* 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，我们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。

* 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

## 如何将打开res aw目录中的数据库文件?

* 在Android中不能直接打开res aw目录中的数据库文件，而需要在程序第一次启动时将该文件复制到手机内存或SD卡的某个目录中，然后再打开该数据库文件。
* 复制的基本方法是使用getResources().openRawResource方法获得res aw目录中资源的 InputStream对象，然后将该InputStream对象中的数据写入其他的目录中相应文件中。
* 在Android SDK中可以使用SQLiteDatabase.openOrCreateDatabase方法来打开任意目录中的SQLite数据库文件。

## 什么是aar?aar是jar有什么区别?

“aar”包是 Android 的类库项目的二进制发行包。 文件扩展名是.aar，maven 项目类型应该也是aar，但文件本身是带有以下各项的 zip 文件：

* /AndroidManifest.xml (mandatory)
* /classes.jar (mandatory)
* /res/ (mandatory)
* /R.txt (mandatory)
* /assets/ (optional)
* /libs/\*.jar (optional)
* /jni//\*.so (optional)
* /proguard.txt (optional)
* /lint.jar (optional)

这些条目是直接位于 zip 文件根目录的。 其中R.txt 文件是aapt带参数–output-text-symbols的输出结果。

jar打包不能包含资源文件，比如一些drawable文件、xml资源文件之类的,aar可以。

## SQLite支持事务吗?添加删除如何提高性能?

SQLite作为轻量级的数据库，比MySQL还小，但支持SQL语句查询，提高性能可以考虑通过原始经过优化的SQL查询语句方式处理

## 如何将SQLite数据库(dictionary.db文件)与apk文件一起发布?

可以将dictionary.db文件复制到Eclipse Android工程中的res aw目录中。所有在res aw目录中的文件不会被压缩，这样可以直接提取该目录中的文件。可以将dictionary.db文件复制到res aw目录中

## 如何保证Service在后台不被kill

* 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
* 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
* 双进程Service： 让2个进程互相保护\*\*，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
* QQ黑科技: 在应用退到后台后，另起一个只有 1 像素的页面停留在桌面上，让自己保持前台状态，保护自己不被后台清理工具杀死
* 在已经root的设备下，修改相应的权限文件,将App伪装成系统级的应用 Android4.0系列的一个漏洞，已经确认可行
* 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。***鉴于目前提到的在Android->-*** ***Service层做双守护都会失败***，我们可以fork出c进程，多进程守护。死循环在那检查是否还存在，具体的思路如下（Android5.0以上的版本不可行）
* 用C编写守护进程(即子进程)，守护进程做的事情就是循环检查目标进程是否存在，不存在则启动它。 在NDK环境中将1中编写的C代码编译打包成可执行文件(BUILD\_EXECUTABLE)。主进程启动时将守护进程放入私有目录下，赋予可执行权限，启动它即可。
* 联系厂商，加入白名单

## Android中如何获得手机的唯一标示.

* 1 首先尝试读取IMEI、Mac地址、CPU号等物理信息（有不少工具可以修改IMEI）；
* 2 如果均失败，可以自己生成UUID然后保存到文件（文件也可能被篡改或删除）

参考:[[http://blog.csdn.net/xushuaic/article/details/25077179](https://link.zhihu.com/?target=http%3A//blog.csdn.net/xushuaic/article/details/25077179" \t "_blank)]

## mipmap文件夹和drawable文件夹的区别

它只是用来放启动图标的,好处就是，你只用放一个mipmap图标，它就会给你各种版本（比如平板，手机）的apk自动生成相应分辨率的图标，以节约空间。

## ListView卡顿的原因以及优化策略

* 重用converView： 通过复用converview来减少不必要的view的创建，另外Infalte操作会把xml文件实例化成相应的View实例，属于IO操作，是耗时操作。
* 减少findViewById()操作： 将xml文件中的元素封装成viewholder静态类，通过converview的setTag和getTag方法将view与相应的holder对象绑定在一起，避免不必要的findviewbyid操作
* 避免在 getView 方法中做耗时的操作: 例如加载本地 Image 需要载入内存以及解析 Bitmap ，都是比较耗时的操作，如果用户快速滑动listview，会因为getview逻辑过于复杂耗时而造成滑动卡顿现象。用户滑动时候不要加载图片，待滑动完成再加载，可以使用这个第三方库glide
* Item的布局层次结构尽量简单，避免布局太深或者不必要的重绘
* 尽量能保证 Adapter 的 hasStableIds() 返回 true 这样在 notifyDataSetChanged() 的时候，如果item内容并没有变化，ListView 将不会重新绘制这个 View，达到优化的目的
* 在一些场景中，ScollView内会包含多个ListView，可以把listview的高度写死固定下来。 由于ScollView在快速滑动过程中需要大量计算每一个listview的高度，阻塞了UI线程导致卡顿现象出现，如果我们每一个item的高度都是均匀的，可以通过计算把listview的高度确定下来，避免卡顿现象出现
* 使用 RecycleView 代替listview： 每个item内容的变动，listview都需要去调用notifyDataSetChanged来更新全部的item，太浪费性能了。RecycleView可以实现当个item的局部刷新，并且引入了增加和删除的动态效果，在性能上和定制上都有很大的改善
* ListView 中元素避免半透明： 半透明绘制需要大量乘法计算，在滑动时不停重绘会造成大量的计算，在比较差的机子上会比较卡。 在设计上能不半透明就不不半透明。实在要弄就把在滑动的时候把半透明设置成不透明，滑动完再重新设置成半透明。
* 尽量开启硬件加速： 硬件加速提升巨大，避免使用一些不支持的函数导致含泪关闭某个地方的硬件加速。当然这一条不只是对 ListView。

## ViewHolder为什么要被声明成静态内部类

* 这个是考静态内部类和非静态内部类的主要区别之一。非静态内部类会隐式持有外部类的引用，就像大家经常将自定义的adapter在Activity类里，然后在adapter类里面是可以随意调用外部activity的方法的。
* 当你将内部类定义为static时，你就调用不了外部类的实例方法了，因为这时候静态内部类是不持有外部类的引用的。声明ViewHolder静态内部类，可以将ViewHolder和外部类解引用。
* 大家会说一般ViewHolder都很简单，不定义为static也没事吧。确实如此，但是如果你将它定义为static的，说明你懂这些含义。万一有一天你在这个ViewHolder加入一些复杂逻辑，做了一些耗时工作，那么如果ViewHolder是非静态内部类的话，就很容易出现内存泄露。
* 如果是静态的话，你就不能直接引用外部类，迫使你关注如何避免相互引用。 所以将 ViewHolder内部类 定义为静态的，是一种好习惯

## 动画相关

## Android中的动画有哪些?

* 逐帧动画(Drawable Animation)：

加载一系列Drawable资源来创建动画，简单来说就是播放一系列的图片来实现动画效果，可以自定义每张图片的持续时间

* 补间动画(Tween Animation)：

Tween可以对View对象实现一系列简单的动画效果，比如位移，缩放，旋转，透明度等等。但是它并不会改变View属性的值，只是改变了View的绘制的位置，比如，一个按钮在动画过后，不在原来的位置，但是触发点击事件的仍然是原来的坐标。

* 属性动画(Property Animation)：

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

## Android动画原理

* Animation框架定义了透明度，旋转，缩放和位移几种常见的动画，而且控制的是整个View
* 实现原理是每次绘制视图时View所在的ViewGroup中的drawChild函数获取该View的Animation的Transformation值
* 然后调用canvas.concat(transformToApply.getMatrix())，通过矩阵运算完成动画帧，如果动画没有完成，继续调用invalidate()函数，启动下次绘制来驱动动画
* 动画过程中的帧之间间隙时间是绘制函数所消耗的时间，可能会导致动画消耗比较多的CPU资源，最重要的是，动画改变的只是显示，并不能相应事件

## View绘制相关

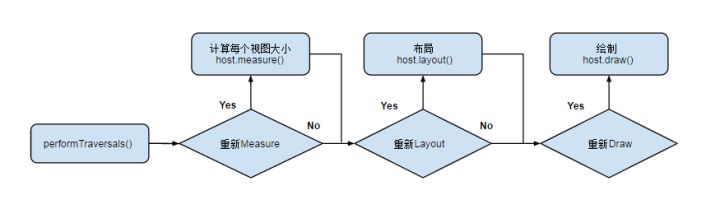
## SurfaceView和View的区别

* SurfaceView中采用了双缓存技术，在单独的线程中更新界面
* View在UI线程中更新界面

## 介绍下自定义view的基本流程

* 1、 明确需求，确定你想实现的效果
* 2、确定是使用组合控件的形式还是全新自定义的形式，组合控件即使用多个系统控件来合成一个新控件，你比如titilebar，这种形式相对简单，参考
* 3、如果是完全自定义一个view的话，你首先需要考虑继承哪个类，是View呢，还是ImageView等子类。
* 4、根据需要去复写View#onDraw、View#onMeasure、View#onLayout方法
* 5.根据需要去复写dispatchTouchEvent、onTouchEvent方法
* 6、根据需要为你的自定义view提供自定义属性，即编写attr.xml,然后在代码中通过TypedArray等类获取到自定义属性值
* 7、需要处理滑动冲突、像素转换等问题 谈谈View的绘制流程

## 谈谈View的绘制流程



measure()方法，layout()，draw()三个方法主要存放了一些标识符，来判断每个View是否需要再重新测量，布局或者绘制，主要的绘制过程还是在onMeasure，onLayout，onDraw这个三个方法中

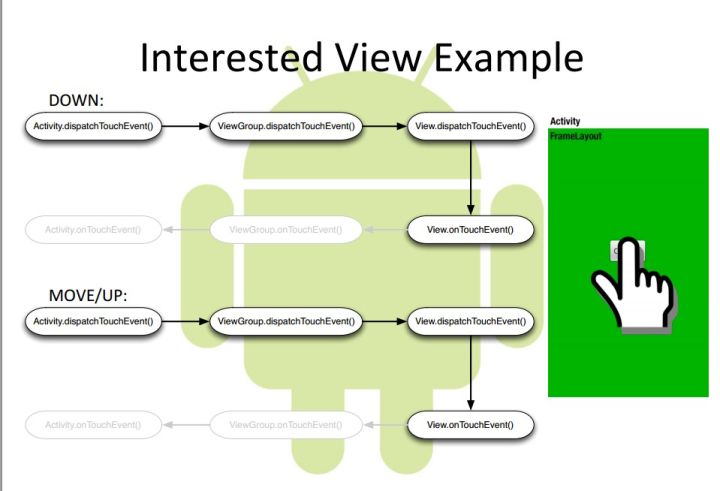
* 1.onMesarue() 为整个View树计算实际的大小，即设置实际的高(对应属性:mMeasuredHeight)和宽(对应属性: mMeasureWidth)，每个View的控件的实际宽高都是由父视图和本身视图决定的。
* 2.onLayout() 为将整个根据子视图的大小以及布局参数将View树放到合适的位置上。
* 3.onDraw() 开始绘制图像，绘制的流程如下
  + 首先绘制该View的背景
  + 调用onDraw()方法绘制视图本身 (每个View都需要重载该方法，ViewGroup不需要实现该方法)
  + 如果该View是ViewGroup，调用dispatchDraw ()方法绘制子视图 绘制滚动条

## 自定义View执行invalidate()方法,为什么有时候不会回调onDraw()

* 自定义一个view时，重写onDraw。调用view.invalidate(),会触发onDraw和computeScroll()。前提是该view被附加在当前窗口. view.postInvalidate(); //是在非UI线程上调用的
* 自定义一个ViewGroup，重写onDraw。onDraw可能不会被调用，原因是需要先设置一个背景(颜色或图)。表示这个group有东西需要绘制了，才会触发draw，之后是onDraw。因此，一般直接重写dispatchDraw来绘制viewGroup.自定义一个ViewGroup,dispatchDraw会调用drawChild.

## 事件传递机制

## 谈谈touch事件的传递流程



* 所有Touch事件都被封装成了MotionEvent对象，包括Touch的位置、时间、历史记录以及第几个手指(多指触摸)等。
* 事件类型分为ACTION\_DOWN, ACTION\_UP, ACTION\_MOVE, ACTION\_POINTER\_DOWN, ACTION\_POINTER\_UP, ACTION\_CANCEL，每个事件都是以ACTION\_DOWN开始ACTION\_UP结束。
* 对事件的处理包括三类，分别为传递——dispatchTouchEvent()函数、拦截——onInterceptTouchEvent()函数、消费——onTouchEvent()函数和OnTouchListener()

**简单来说**:

* 事件从Activity.dispatchTouchEvent()开始传递，只要没有被停止或拦截，从最上层的View(ViewGroup)开始一直往下(子View)传递。子View可以通过onTouchEvent()对事件进行处理。
* 事件由父View(ViewGroup)传递给子View，ViewGroup可以通过onInterceptTouchEvent()对事件做拦截，停止其往下传递。
* 如果事件从上往下传递过程中一直没有被停止，且最底层子View没有消费事件，事件会反向往上传递，这时父View(ViewGroup)可以进行消费，如果还是没有被消费的话，最后会到Activity的onTouchEvent()函数。
* 如果View没有对ACTION\_DOWN进行消费，之后的其他事件不会传递过来。

= OnTouchListener优先于onTouchEvent()对事件进行消费。 上面的消费即表示相应函数返回值为true。

## View中setOnTouchListener中的onTouch,onTouchEvent,onClick的执行顺序

onTouch优于onTouchEvent,onTouchEvent优于onClick

触摸事件的分发机制详见：

[Android触摸事件分发机制完全解析《一》](https://link.zhihu.com/?target=http%3A//blog.csdn.net/androidstarjack/article/details/71910168" \t "_blank)

## 什么是Dalvik虚拟机

* Dalvik虚拟机是Android平台的核心。
* 它可以支持.dex格式的程序的运行
* .dex格式是专为Dalvik设计的一种压缩格式
* 可以减少整体文件尺寸
* 提高I/O操作的速度
* 适合内存和处理器速度有限的系统

## Dalvik虚拟机和JVM有什么区别

* Dalvik 基于寄存器，而 JVM 基于栈。基于寄存器的虚拟机对于更大的程序来说，在它们编译的时候，花费的时间更短。
* Dalvik执行.dex格式的字节码，而JVM执行.class格式的字节码

## Android为每个应用程序分配的内存大小是多少

* 一般是16m或者24m,但是可以通过android:largeHeap申请更多内存

## 如何解决方法数65k问题?

使用Android Studio 的gradle 可以构建MutilDex