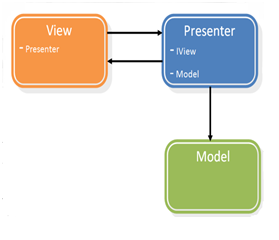
# MVP in Android



Model：

封装的各种数据的实体类，以及提供对数据操作的接口，比如本地查询数据或者网上加载数据等操作。完全独立，只涉及数据层，不依赖。

View：

xml布局文件以及Activity或Fragment

Xml布局文件编写静态ui界面，Activity或Fragment提供可交互的ui操作，比如弹窗或点击事件等。

Activity或Fragment里通常拥有Presenter的引用，各种业务操作可先定义为接口，由Presenter来实现，然后通过Presenter的引用来调用。

Presenter：业务层操作

通常拥有Activity或Fragment的引用，也拥有Model层的引用，先通过Model层的引用获取数据后再通过Activity的引用来通知Activity该做出的反应。

# RxJava

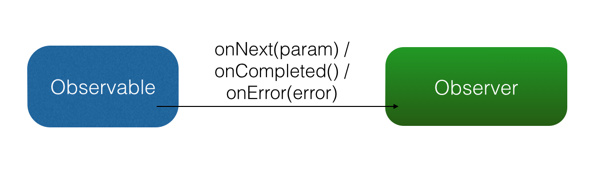
抛物线： <http://gank.io/post/560e15be2dca930e00da1083>

Github： <https://github.com/ReactiveX/RxJava>    
<https://github.com/ReactiveX/RxAndroid>

RxJava是什么？------异步，一个在JAVA VM上使用可观测的序列来组成异步、基于事件的程序的库

RxJava 有四个基本概念：Observable (可观察者，即被观察者)、 Observer (观察者)、 subscribe (订阅)、事件。Observable 和 Observer 通过 subscribe() 方法实现订阅关系，从而 Observable 可以在需要的时候发出事件来通知 Observer。

与传统观察者模式不同， RxJava 的事件回调方法除了普通事件 onNext() （相当于 onClick() / onEvent()）之外，还定义了两个特殊的事件：onCompleted() 和 onError()。



使用方式：

（被观察者.订阅（观察者），虽然与现实相反，但这样支持链式调用）

必须包括三步：

1. 创建被观察者
2. 创建观察者
3. 关联观察者和被观察者的联系

代码示例：



# 观察者模式（发布/订阅模式）

概念

现实概念上的观察者模式：A对象（观察者）需要时时关注B对象（被观察者）的某种状态是否发生改变，在发生变化的一瞬间做出反应。

程序里的观察者模式：B对象（被观察者）的某种状态发生变化时，需要主动通知A对象（观察者），让A对象（观察者）做出反应。所以也称作发布/订阅模式，A对象（观察者）订阅B对象（被观察者）的某一状态，当B对象的被订阅的状态发生变化时，就发布通知，告知所有订阅这一状态的观察者需要做出相应的反应了。

两者的区别有点类似于《计算机组成原理》里的程序查询方式和中断方式，一种是主动查询，一种是被动等待通知。

Android源码里的例子

OnClickListener()------按钮的点击事件

Button-----被观察者 onClickListener-----观察者（观察Button的onClick()状态）

两者通过Button.setOnClickListener(onClickListener)来建立订阅关系



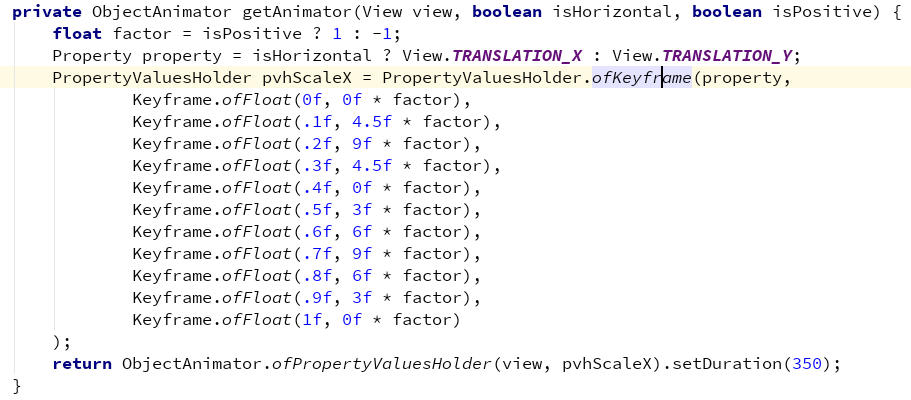
# PropertyValuesHolder

用途：结合动画来实现view的抖动

使用方法：

第一步：生成Keyframe对象；  
第二步：利用PropertyValuesHolder.ofKeyframe()生成PropertyValuesHolder对象  
第三步：ObjectAnimator.ofPropertyValuesHolder()生成对应的Animator

示例代码：



Keyframe.ofFloat(0, 0)表示动画进度为0时，动画所在的数值位置为0；Keyframe.ofFloat(0.25f, -20f)表示动画进度为25%时，动画所在的数值位置为-20；Keyframe.ofFloat(1f,0)表示动画结束时，动画所在的数值位置为0；

# ViewStub

在开发应用程序的时候，经常会遇到这样的情况，会在运行时动态根据条件来决定显示哪个View或某个布局。那么最通常的想法就是把可能用到的View都写在上面，先把它们的可见性都设为View.GONE，然后在代码中动态的更改它的可见性。这样的做法的优点是逻辑简单而且控制起来比较灵活。但是它的缺点就是，耗费资源。虽然把View的初始可见View.GONE但是在Inflate布局的时候View仍然会被Inflate，也就是说仍然会创建对象，会被实例化，会被设置属性。也就是说，会耗费内存等资源。

推荐的做法是使用[android](http://lib.csdn.net/base/15).view.ViewStub，ViewStub 是一个轻量级的View，它一个看不见的，不占布局位置，占用资源非常小的控件。可以为ViewStub指定一个布局，在Inflate布局的时候，只有 ViewStub会被初始化，然后当ViewStub被设置为可见的时候，或是调用了ViewStub.inflate()的时候，ViewStub所向 的布局就会被Inflate和实例化，然后ViewStub的布局属性都会传给它所指向的布局。这样，就可以使用ViewStub来方便的在运行时，要还 是不要显示某个布局。

 但ViewStub也不是万能的，下面总结下ViewStub能做的事儿和什么时候该用ViewStub，什么时候该用可见性的控制。

 首先来说说ViewStub的一些特点：

         1. ViewStub只能Inflate一次，之后ViewStub对象会被置为空。按句话说，某个被ViewStub指定的布局被Inflate后，就不会够再通过ViewStub来控制它了。

         2. ViewStub只能用来Inflate一个布局文件，而不是某个具体的View，当然也可以把View写在某个布局文件中。

 基于以上的特点，那么可以考虑使用ViewStub的情况有：

         1. 在程序的运行期间，某个布局在Inflate后，就不会有变化，除非重新启动。

  因为ViewStub只能Inflate一次，之后会被置空，所以无法指望后面接着使用ViewStub来控制布局。所以当需要在运行时不止一次的显示和 隐藏某个布局，那么ViewStub是做不到的。这时就只能使用View的可见性来控制了。

         2. 想要控制显示与隐藏的是一个布局文件，而非某个View。

          因为设置给ViewStub的只能是某个布局文件的Id，所以无法让它来控制某个View。

# android:drawableRight

TextView的drawable属性

TextView有一些属性可以在Text的四周设置一个drawable对象，图片,shape等合法的drawable都可以用。

* drawableStart API 14才有
* drawableLeft
* drawableTop
* drawableBottom
* drawableRight
* drawableEnd API 14才有
* drawablePadding 用以设置drawable与text之间的空间

TextView的drawable的一些注意事项

1. . TextView的padding作用在drawable之外
2. . TextView的高度或宽度为wrap\_content时将是文字和drawable中较大的那一个，再加上padding和margin
3. . gravity只对文字起作用，对drawable不起作用
4. . drawable会在其所在的维度居中显示，比如drawableLeft是上下垂直居中的，以此类推

局限性

有些时候它也有一些局限性而没有办法用它：

1. 当drawable本身没有高度时（比如shape），这个drawable高度就会依赖于文字，因为padding是加在drawable之外，所以只会依赖于文字的高度。有些时候这不是想要的结果。

2. 当Icon需要与文字分开单独控制时，很显然这要分成二个View。

3. 当需要对Icon进行特殊的个性化时，比如添加背景，特效等。

4. 其他一些造成无法使用的。

5. 除上述情况外，就要考虑使用drawable了。

# Xml焦点控制

android:descendantFocusability----父容器和子控件的焦点获取问题

通过android:descendantFocusability这个属性可以解决上面的问题。这个属性定义了当一个焦点要传递给父容器或者子控件时，父容器和子控件之间获得焦点的关系。具体值如下：

beforeDescendants：父容器会比其子控件率先获得焦点。

afterDescendants：如果没有任何子控件要获得焦点的话，那么父容器才会获得焦点。

blocksDescendants：父容器会阻止其子控件获得焦点（也就是说焦点会由父容器获得）。

android:duplicateParentState

指的是当前控件是否跟随父控件的(点击、焦点等)状态

android:nextFocusRight

控制下一个焦点

# adb命令--删除系统应用

1、 mount -o rw,remount /system 卸载系统应用时先运行这句

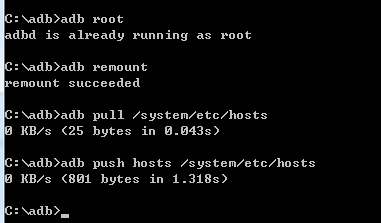
2、 后把 /system/app 和/data/data 下的相关文件删掉

3、 reboot重启盒子

4、 安装debug应用

添加一下、system目录的权限，就能删了

# adb修改ect/host



# adb 启动任意activity

<http://www.jianshu.com/p/54fd9627860a>

adb shell am start –n com.qiyi.wstv/con.qiyi.wstv.MainActivity



隐式启动

格式:adb shell am start -a <ACTION> -c <CATEGORY> -d <DATA\_URI> -t <MIME\_TYPE>

启动邓超明星页

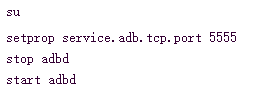
adb shell am start -a android.intent.action.VIEW -d wsiqiyi://wstv.qiyi.com/star?starId=500000&ElementID=500002

# Adb wifi连接调试

<http://blog.csdn.net/dabaoonline/article/details/50802952>

方法一：需要root权限

手机端下载超级终端，输入下面命令：



Android 终端 <https://github.com/jackpal/Android-Terminal-Emulator>

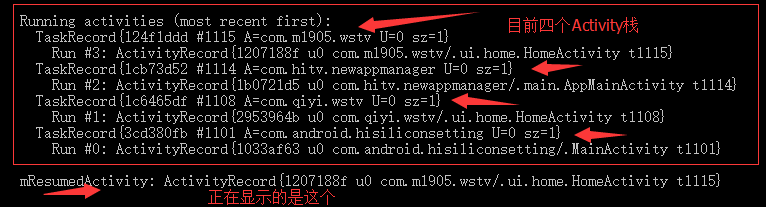
方法二：需要usb线，不需要root

Usb连接手机调试后在pc端的cmd窗口输入下列命令：



# adb查看目前的Activity栈

命令：adb shell dumpsys activity



# TextView各种padding解析、长度测量

TextView测量文字宽度

TextPaint textPaint = textView.getPaint();

float textPaintWidth = textPaint.measureText(text);

<http://www.jianshu.com/p/fd9cce7a333f>

----------------------------------------------------------------------------------------

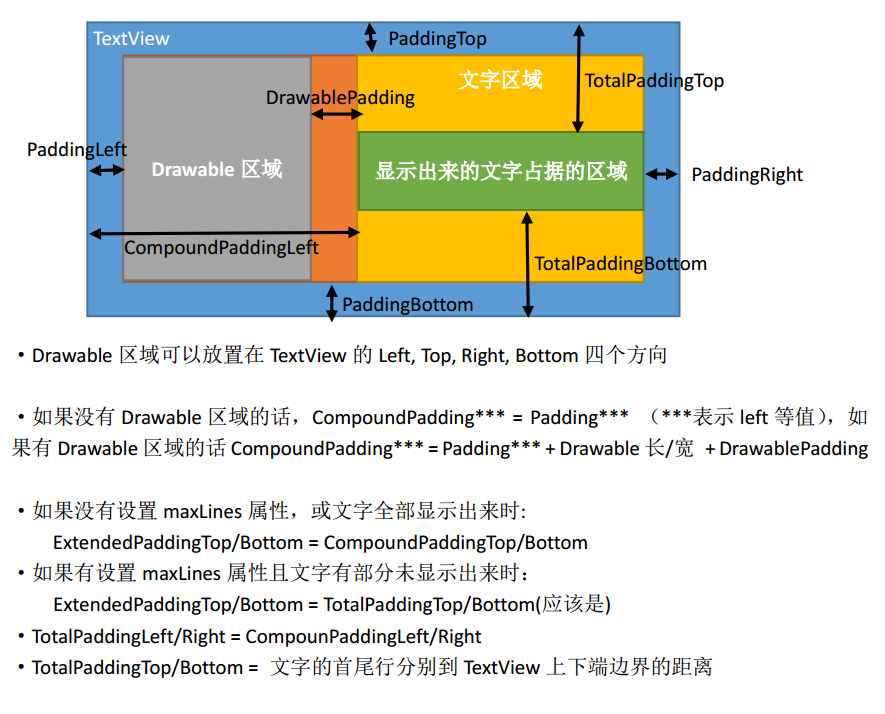
getCompoundPadding\*\*\*\*

getCompoundDrawablePadding()

getExtendedPaddingBottom/Top

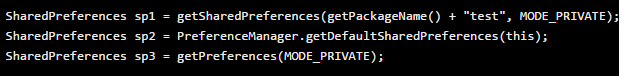
getMinWidth/Height

getTotalPadding\*\*\*



# SharePreference监听

registerOnSharedPreferenceChangeListener(listener)  
这个方法才是我要说的重点，因为之前有些需求就是更改了SharedPreferences之后，要通知相应的组件做出改变，我以前的处理方式是通过事件订阅实现的，发一个event出去，然后目标收到event再做出反应，当时觉得特别蛋疼，两边都要做些操作，显的特别啰嗦，当时就在想可不可以在SharedPreferences上设置一个观察者，一旦有什么风吹草动，就自动通知目标，不曾想，人家早已经实现了，只是我愚昧无知，今天去看了下源码发现了这个方法，相见恨晚。



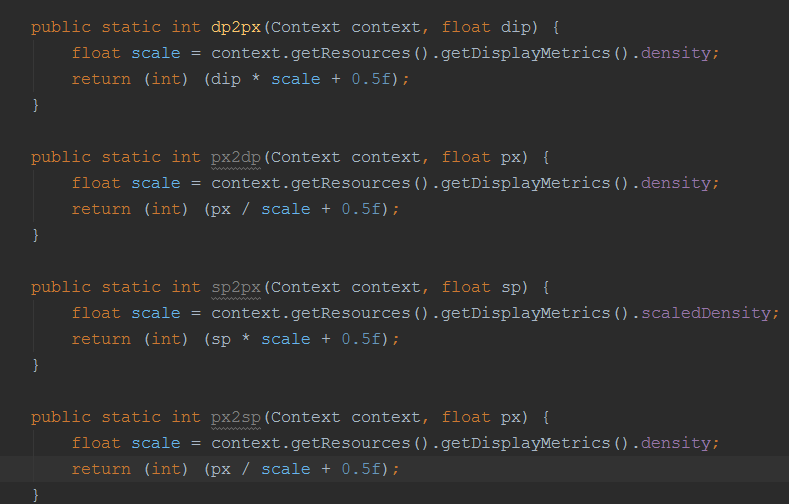
# Dp和px

* dip        ： Density independent pixels ，设备无关像素。
* dp        ：就是dip
* px        ： 像素
* dpi       ：dots per inch ， 直接来说就是一英寸多少个像素点。常见取值 120，160，240。我一般称作像素密度，简称密度
* density ： 直接翻译的话貌似叫 密度。常见取值 1.5 ， 1.0 。和标准dpi的比例（160px/inc）
* 分辨率   ： 横纵2个方向的像素点的数量，常见取值 480X800 ，320X480
* 屏幕尺寸： 屏幕对角线的长度。电脑电视同理。
* 屏幕比例的问题。因为只确定了对角线长，2边长度还不一定。所以有了4：3、16：9这种，这样就可以算出屏幕边长了。

换算公式如下：

PX = density \* DP

代码



# Gradle

概念：

projects 和 tasks是 Gradle 中最重要的两个概念。

任何一个 Gradle 构建都是由一个或多个 projects 组成。

每个 project 都由多个 tasks 组成。每个 task 都代表了构建执行过程中的一个原子性操作。如编译，打包，生成 javadoc，发布到某个仓库等操作。

在两个任务之间指明依赖关系 dependsOn

task hello << {

println 'Hello world!'

}

task intro(dependsOn: hello) << {

println "I'm Gradle"

}

//或者

task0.dependsOn task2, task3

为任务增加自定义属性

task myTask {

ext.myProperty = "myValue"

}

task printTaskProperties << {

println myTask.myProperty

}

依赖任务的不同输出

task distribution << {

println "We build the zip with version=$version"

}

task release(dependsOn: 'distribution') << {

println 'We release now'

}

gradle.taskGraph.whenReady {taskGraph ->

if (taskGraph.hasTask(release)) {

version = '1.0'

} else {

version = '1.0-SNAPSHOT'

}

}

gradle -q distribution 的输出结果

Output of gradle -q distribution

\> gradle -q distribution

We build the zip with version=1.0-SNAPSHOT

gradle -q release 的输出结果

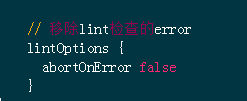
Output of gradle -q release

\> gradle -q release

We build the zip with version=1.0

We release now

whenReady 会在已发布的任务之前影响到已发布任务的执行。即使已发布的任务不是主要任务(也就是说，即使这个任务不是通过命令行直接调用)



<http://stormzhang.com/android/2016/03/13/gradle-config/>

# Groovy

Groovy是一种基于JVM的动态语言，说简单点，就是可以再Java虚拟机上运行的脚步语音。它和Java类似（算是Java的升级版，但是又具备脚本语言的特点），都在Java虚拟机中运行。当运行Groovy脚本时它会先被编译成Java类字节码，然后通过JVM虚拟机执行这个Java字节码类。

Task是Groovy的核心所在，Task是完成Groovy任务的最小执行单元。添加一个Task很简单，指定 task [task name]即可。

task hello {

println ‘hello world’

}

Hello.enable = false 关闭或开启一个task

复制文件task

Task testCopy(type: Copy){

From ‘src/man/res/layout’

Into ‘src/man/new

}

一个task编译时打出UP-TO-DATE表示该Task之前已经编译过了，且没有任何修改，所以跳过该task的执行，也就是Gradle的增量编译。

字符串的定义：单引号，双引号，斜杠，三引号

> apply plugin: 'com.android.application'

常见的还有另一个就是 apply plugin: 'library'，作用也就是设置当前model是主程序还是库。

作用清楚了，下面来理解一下，为什么要这样设置。gradle是基于groovy语言的，

参考文献：

在AS中自定义Gradle插件 [http://www.jianshu.com/p/d53399cd507b#](http://www.jianshu.com/p/d53399cd507b)

构建神器 <http://jiajixin.cn/2015/08/07/gradle-android/>

Gradle tip <http://blog.csdn.net/lzyzsd/article/category/2795779>

Groovy探索GString <http://blog.csdn.net/hivon/article/details/2271000>

Diff with Java <http://www.groovy-lang.org/differences.html>

Gradle for Android <https://segmentfault.com/a/1190000004229002>

Gradle Plugin User Guide <http://tools.android.com/tech-docs/new-build-system/user-guide>

Groovy语言规范-语法 <http://ifeve.com/groovy-syntax/>

学习Gradle：

1.了解groovy基本语法

2.粗度 Gradle User Guide 和 Gradle Plugin User Guide

<https://docs.gradle.org/current/userguide/userguide.html>

<http://tools.android.com/tech-docs/new-build-system/user-guide>

3.实战

Groovy学习：

官方教程 <http://www.groovy-lang.org/differences.html>

IMB 精通Groovy <http://www.ibm.com/developerworks/cn/education/java/j-groovy/j-groovy.html>

//获取所有的task, 过滤掉某些task的执行

tasks.whenTaskAdded{task ->

if (task.name.contains('AndroidTest')) {

task.enabled = false;

}

}

gradle执行分三个阶段，初始化阶段（setting.gradle)，配置阶段(build.gradle生成Project和Gradle对象）

执行阶段(执行各个task)。

//afterEvaluate 会在gradle的配置阶段结束时执行

aftertEvaluate {

android.applicationVariants.each { variant ->

def dx = tasks.findByName("dex$variant.name.capitalize()}")

def hello = "hello$variant.name.capitalize()}"

task(hello) << {

println "hello"

}

tasks.findByName(hello).dependsOn dx.taskDependencies.getDependencies(dx)

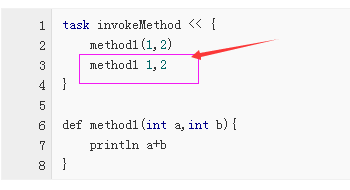
dx.dependsOn tasks.findByName(hello)

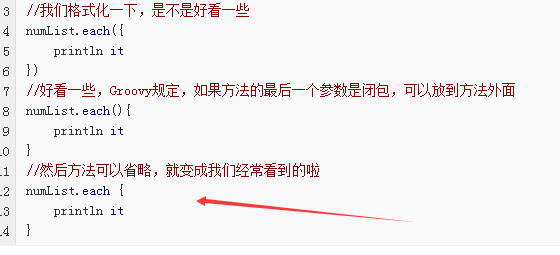
}

}

Groovy里的方法

Groovy方法的括号可以省略

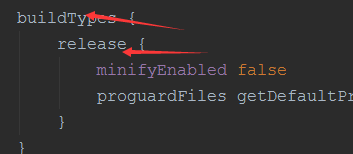




Groovy支持闭包作为方法参数，如上图第三行代码

Groovy规定，如果方法的最后一个参数是闭包，那么闭包可以放在括号外面即方法后面，如上图第7行

Groovy支持省略掉方法的括号，因此最终如上图第.2行代码



这在build.gradle里很常见，其实都是一个个的方法，参数为闭包，都省略掉方法的括号，所以最终写出来都如是 方法名{} 格式。

属性和get,set方法





# build.gradle

查看android插件的各个标签作用

<http://google.github.io/android-gradle-dsl/current/index.html>

读取local.properties里的变量时

Properties properties = new Properties();  
properties.load(project.rootProject.file('local.properties').newDataInputStream())  
println properties.getProperty('KEYPASS')

也可以 properties.load(new FileInputStream(project.rootProject.file(‘local.properties’)

应用于一些敏感数据时，可以用上述方法，比如签名配置，各种app key

多渠道打包配置

1.配置AndroidManifest.xml  
以友盟渠道为例，渠道信息一般都是写在 AndroidManifest.xml文件中：

<meta-data android:name="UMENG\_CHANNEL" android:value="xiaomi" />

如果不使用多渠道打包方法，那就需要我们手动一个一个去修改value中的值，xiaomi，360，qq，wandoujia等等。使用多渠道打包的方式，就需要把上面的value配置成下面的方式：

<meta-data android:name="UMENG\_CHANNEL" android:value="${UMENG\_CHANNEL\_VALUE}" />

其中${UMENG\_CHANNEL\_VALUE}中的值就是你在gradle中自定义配置的值。

2.在build.gradle设置productFlavors  
写法如下：

productFlavors {

wandoujia {

manifestPlaceholders = [UMENG\_CHANNEL\_VALUE: "wandoujia"]

}

xiaomi {

manifestPlaceholders = [UMENG\_CHANNEL\_VALUE: "xiaomi"]

}

qq {

manifestPlaceholders = [UMENG\_CHANNEL\_VALUE: "qq"]

}

360 {

manifestPlaceholders = [UMENG\_CHANNEL\_VALUE: "360"]

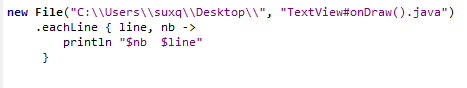
}

}

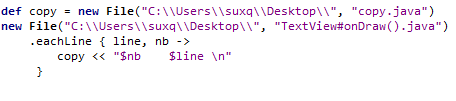
其中[UMENG\_CHANNEL\_VALUE: "wandoujia"]就是对应  
${UMENG\_CHANNEL\_VALUE}的值。

# Groovy 文件操作

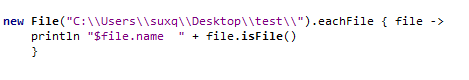
读文件：



写文件



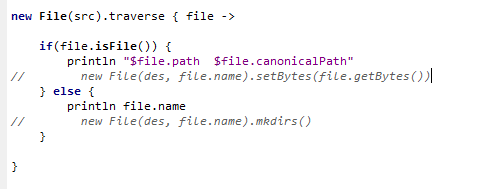
遍历文件夹，包括文件和子文件夹



创建文件夹



遍历文件夹



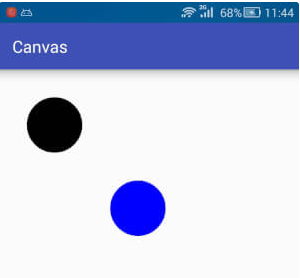
# Canvas画布坐标点

canvas.translation(float dx, float dy)

画布坐标系的平移，记住是基于当前坐标系中心点的位置，最开始默认坐标系原点在左上角。

// 在坐标原点绘制一个黑色圆形mPaint.setColor(Color.BLACK);canvas.translate(200,200);canvas.drawCircle(0,0,100,mPaint);

// 在坐标原点绘制一个蓝色圆形mPaint.setColor(Color.BLUE);canvas.translate(200,200);canvas.drawCircle(0,0,100,mPaint);



# 自定义View

View的坐标系

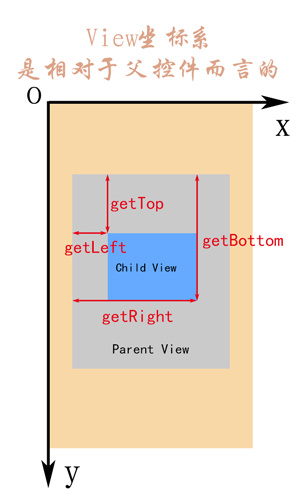
注意：View的坐标系统是相对于父控件而言的

getTop(); //获取子View左上角距父View顶部的距离

getLeft(); //获取子View左上角距父View左侧的距离

getBottom(); //获取子View右下角距父View顶部的距离

getRight(); //获取子View右下角距父View左侧的距离



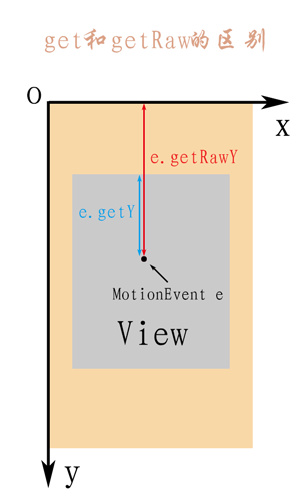
MotionEvent中 get 和 getRaw 的区别

event.getX(); //触摸点相对于其所在组件坐标系的坐标

event.getY();

event.getRawX(); //触摸点相对于屏幕默认坐标系的坐标

event.getRawY();



安卓支持的颜色模式



PS：其中字母表示通道类型，数值表示该类型用多少位二进制来描述。如ARGB8888则表示有四个通道(ARGB),每个对应的通道均用8位来描述。

注意：我们常用的是ARGB8888和ARGB4444，而在所有的安卓设备屏幕上默认的模式都是RGB565,请留意这一点。

A 从ox00到oxff表示从透明到不透明。

RGB 从0x00到0xff表示颜色从浅到深。

当RGB全取最小值(0或0x000000)时颜色为黑色，全取最大值(255或0xffffff)时颜色为白色

代码里使用颜色的方法

1.java中定义颜色

int color = Color.GRAY; //灰色

int color = Color.argb(127, 255, 0, 0); //半透明红色

int color = 0xaaff0000; //带有透明度的红色

2.在xml文件中定义颜色





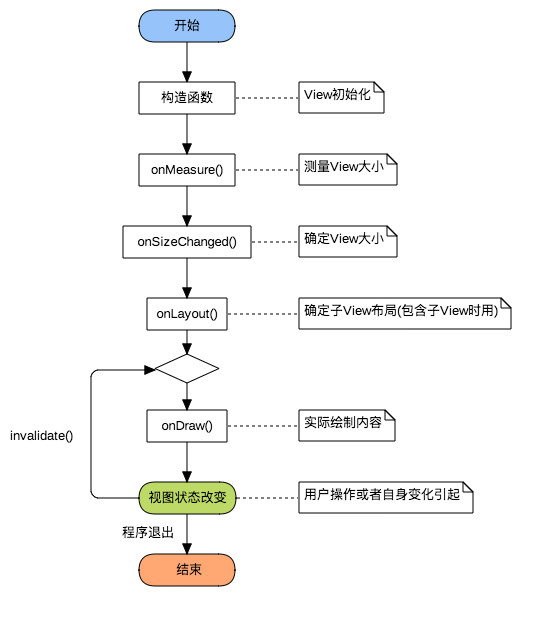
3.在java文件中引用xml中定义的颜色：



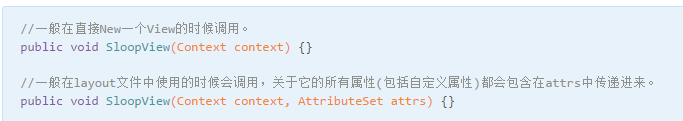
4.在xml文件(layout或style)中引用或者创建颜色



自定义View绘制流程



1.构造方法



2.测量View大小



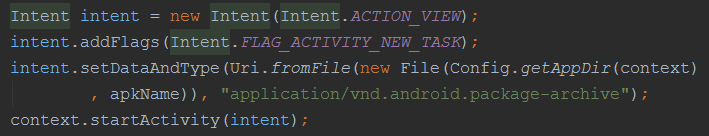


# 打开系统安装界面

设置 intent 的 dataAndType ，用隐式调用法来启动系统的安装界面

传入apk的uri路径：[file://mnt/sdcard/aa.apk](file://mnt\sdcard\aa.apk)

和指定的type：application/vnd.android.package-archive



# CopyOnWriteArrayList

一个线程安全的，可随机读取的 list。

CopyOnWrite容器即写时复制的容器。通俗的理解是当我们往一个容器添加元素的时候，不直接往当前容器添加，而是先将当前容器进行Copy，复制出一个新的容器，然后新的容器里添加元素，添加完元素之后，再将原容器的引用指向新的容器。这样做的好处是我们可以对CopyOnWrite容器进行并发的读，而不需要加锁，因为当前容器不会添加任何元素。所以CopyOnWrite容器也是一种读写分离的思想，读和写不同的容器。

读的时候不需要加锁，如果读的时候有多个线程正在向CopyOnWriteArrayList添加数据，读还是会读到旧的数据，因为写的时候不会锁住旧的CopyOnWriteArrayList。

CopyOnWrite并发容器用于读多写少的并发场景。比如白名单，黑名单，商品类目的访问和更新场景，假如我们有一个搜索网站，用户在这个网站的搜索框中，输入关键字搜索内容，但是某些关键字不允许被搜索。这些不能被搜索的关键字会被放在一个黑名单当中，黑名单每天晚上更新一次。当用户搜索时，会检查当前关键字在不在黑名单当中，如果在，则提示不能搜索。

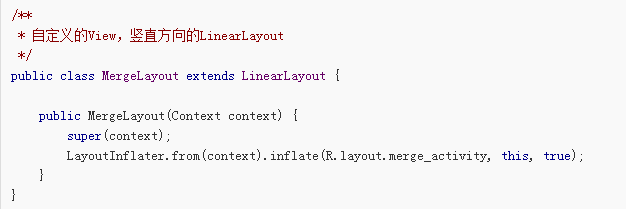
CopyOnWrite容器有很多优点，但是同时也存在两个问题，即内存占用问题和数据一致性问题。

**内存占用问题**。因为CopyOnWrite的写时复制机制，所以在进行写操作的时候，内存里会同时驻扎两个对象的内存，旧的对象和新写入的对象

**数据一致性问题**。CopyOnWrite容器只能保证数据的最终一致性，不能保证数据的实时一致性。所以如果你希望写入的的数据，马上能读到，请不要使用CopyOnWrite容器。

# Merge标签

1. merge必须放在布局文件的根节点上。
2. merge并不是一个ViewGroup，也不是一个View，它相当于声明了一些视图，等待被添加。
3. merge标签被添加到A容器下，那么merge下的所有视图将被添加到A容器下。
4. 因为merge标签并不是View，所以在通过LayoutInflate.inflate方法渲染的时候， 第二个参数必须指定一个父容器，且第三个参数必须为true，也就是必须为merge下的视图指定一个父亲节点。
5. 如果Activity的布局文件根节点是FrameLayout，可以替换为merge标签，这样，执行setContentView之后，会减少一层FrameLayout节点。
6. 自定义View如果继承LinearLayout，建议让自定义View的布局文件根节点设置成merge，这样能少一层结点。
7. 因为merge不是View，所以对merge标签设置的所有属性都是无效的。



# TextView设置下划线

如果在xml中



如果在代码中



或者



# TextView设置不同字体样式

效果图



实现

SpannableString

实现了CharSequence接口，可以当作字符串使用，直接传入Tv中

使用方法：

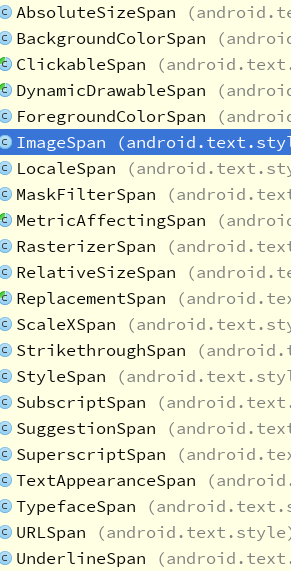
1. SpannableString text = **new** SpannableString(“输入课程名称，如管理”);

2. text.setSpan(**new** ForegroundColorSpan(ContextCompat.*getColor*(**this**, R.color.***text\_color\_gray***))

, 0, 10, Spanned.SPAN\_EXCLUSIVE\_EXCLUSIVE);

3.text.setSpan(**new** AbsoluteSizeSpan(textSize1), 10, 14, Spanned.SPAN\_EXCLUSIVE\_EXCLUSIVE);

4.TextView.setText(text);

关键方法：setSpan

支持设置的字体格式：

继承CharacterSty均可

字体颜色： ForegroundColorSpan

字体大小： AbsoluteSizeSpan

背景颜色： BackgroundColorSpan

超链接： URLSpan

粗体、斜体： StyleSpan

删除线： StrikethroughSpan

下划线： UnderlineSpan

图片： ImageSpan

# android:clipToPadding android:clipToChildren

clipToPadding就是说控件的绘制区域是否在padding里面的，true的情况下如果你设置了padding那么绘制的区域就往里 缩，clipChildren是指子控件是否超过padding区域，这两个属性默认是true的，所以在设置了padding情况下，默认滚动是在 padding内部的，要达到上面的效果主要把这两个属性设置了false那么这样子控件就能画到padding的区域了。

# OnGlobalLayoutListener

用于获取view宽高

我们知道在oncreate中View.getWidth和View.getHeight无法获得一个view的高度和宽度，这是因为View组件布局要在onResume回调后完成。所以现在需要使用getViewTreeObserver().addOnGlobalLayoutListener()来获得宽度或者高度。这是获得一个view的宽度和高度的方法之一。



但是需要注意的是OnGlobalLayoutListener可能会被多次触发，因此在得到了高度之后，要将OnGlobalLayoutListener注销掉。另外mHeaderViewHeight和mHeaderView都需要写在当前java文件类（比如Activity）的成员变量中。不能直接在onCreate中定义否则会编译不通过：

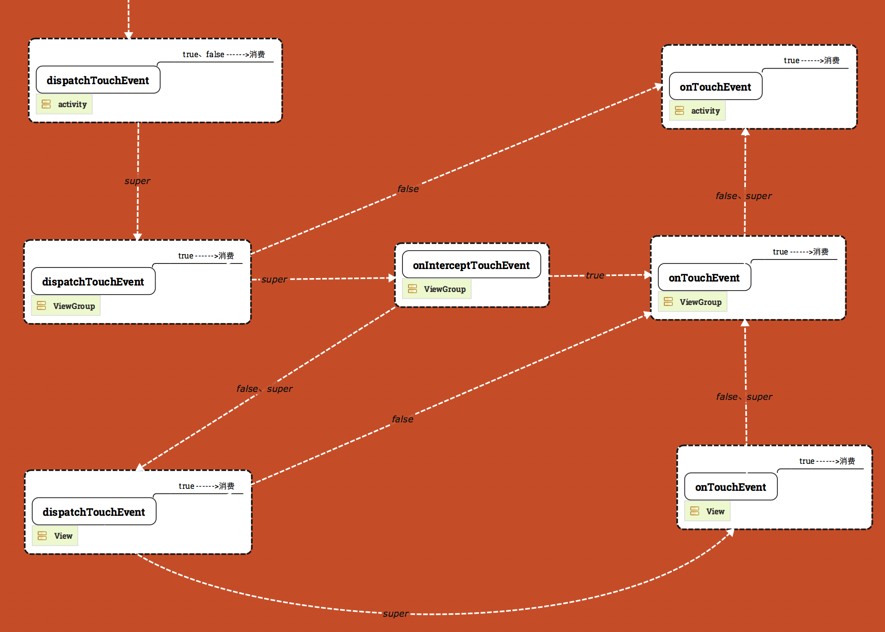
# 点击事件分发机制

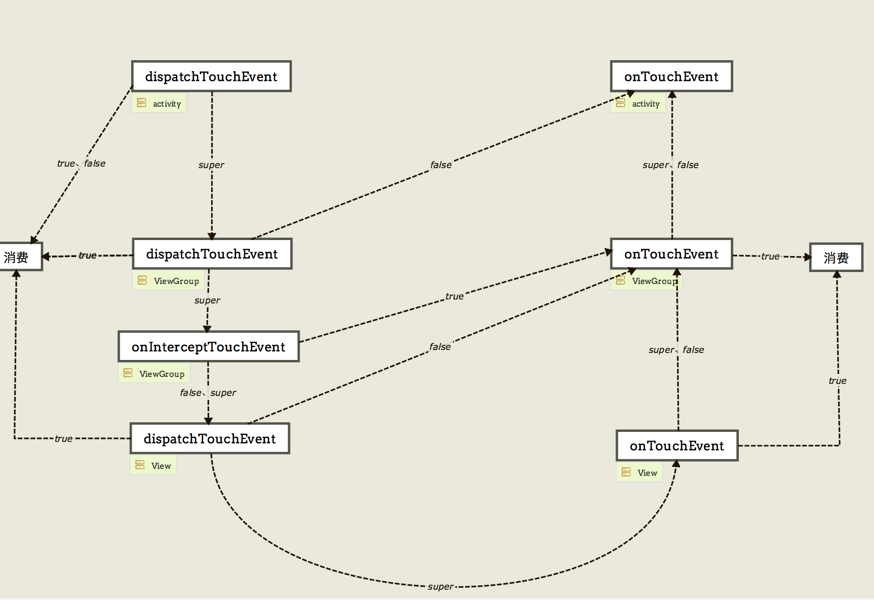
分三个对象，最顶层的Activity、中间层的ViewGroup、底层的View

事件从上往下传递，由下往上消费，中间某一过程如果被谁消费掉之后，事件就停止传递了。

所以回调的方法主要有三个，分发dispatchKeyEvent、拦截InteruptEvent、消费OnTouchEvent。

基本概念，return true表示消费





# AndroidTv的点击事件分发

<http://www.imooc.com/article/12664>

在Tv开发中没有以前我们phone端的dispatchTouchEvent、onInterceptTouchEvent、onTouchEvent 事件来分发，而需要使用dispatchKeyEvent、onKeyDown、onKeyLisenter 等事件来分发处理焦点事件传递

应用层事件分发从ViewRootImpl$ViewPostImeInputStage -> processKeyEvent()开始

主要的分发处理逻辑：

（PhoneWindow$）DecorView -> dispatchKeyEvent()

View -> dispatchKeyEvent()

ViewGroup -> dispatchKeyEvent()

Activity -> dispatchKeyEvent()

KeyEvent -> dispatch()

由于都是各种递归嵌套调用，流程图太复杂，已放弃。

最开始从DecorView的dispatchKeyEvent中分发事件，这里面先是判断事件是否是由activity触发还是由dialog触发，activity触发的交由activity的dispatchKeyEvent处理，dialog触发的交由viewGroup的dispatchKeyEvent处理。如果这两大分支返回true，则停止分发，否则交由PhoneWindow的onKeyDown/Up处理。

第一大分支，Activity的dispatchKeyEvent中：由多态的性质，如果有重写该方法的就交由重写的处理，处理的如果返回true，则停止分发，返回supe的交由父类处理，否则就是都不处理，那就仍给PhoneWindow处理。

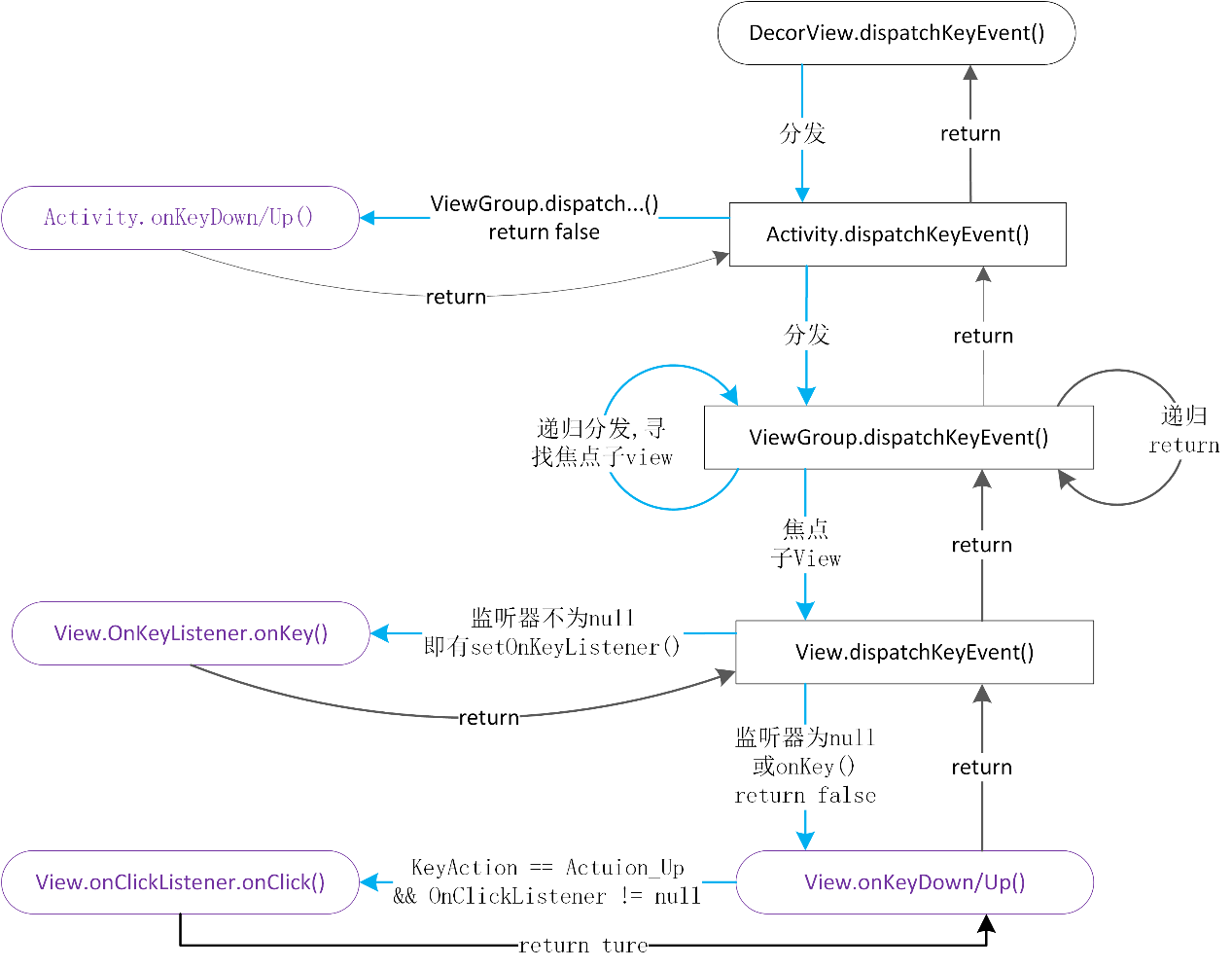
基类Activity的处理：如果是返回事件，交由PhoneWindow处理，停止分发。否则交由ViewGroup的dispatchKeyEvent继续分发。ViewGroup里会递归调用子View，可以是ViewGroup，Linearlayout，TextView等控件。总之，ViewGroup就是递归分发到View的dispatchKeyEvent里处理。

View的dispatchKeyEvent处理：如果有设置监听器OnKeyListener的话，交由listener的onKey处理，返回true，停止分发，否则交由KeyEvent的dispatch分发。

KeyEvent的dispatch处理：调用activity或dialog的onKeyDown/Up处理，如果返回true，则停止分发。

·dispatchKeyEvent只做事件的分发，返回true，代表事件已被消耗，停止分发。返回false代表事件没被处理。通常我们会重写activity的dispatchKeyEvent方法来拦截事件，达到控制焦点的作用。很多人都是直接return false，但这其实是错误的。Return false表示activity包括它所有的子View都不处理，也就是activity里的onKeyDown/Up和布局里的各种view设置的OnKeyListenre不会触发，但焦点仍然会移动，这是因为焦点的移动是由ViewRootImpl$ViewPostImeInputState.processKeyEvent里处理的，这个方法是先将KeyEvent扔给Activity分发处理，如果activity返回true，则表示处理完毕，如果activity返回false，则继续往下处理焦点的移动。

所以，在activity的dispatchKeyEvent里返回true，才可以达到停止焦点的移动以及事件的分发；返回false时，只是停止事件的分发，焦点仍旧会移动；至于返回super.dispatchKeyEvent()则是将分发处理交由父类实现，一般是交由基类Activity处理，基类Activity默认会先将事件分发给布局文件ViewGroup，然后ViewGroup递归分发给子View，如果布局文件都没有消耗该事件，那么就交由Activity的OnKeyDown处理。如果View或者Activity都没有重写onKeyDown处理，那么所有的分发处理默认都是返回false。



# BUG解决汇总

· SQLiteCantOpenDatabaseException: unable to open database file (code 14)

Sqlite问题，<http://www.jianshu.com/p/6ad0491404da>

·为什么有的页面挑战Action\_Up会在下个页面处理，有的不会？

# VerticalGridView

VerticalGridView是谷歌专门为Android TV适配的RecyclerView，VerticalGridView很好的解决了列表的焦点问题，但是如果使用不当，很容易出现item子项焦点错乱问题。

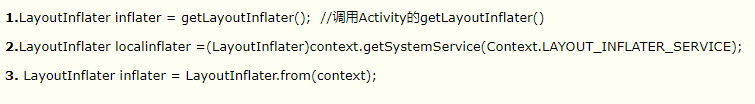
Item聚焦后的放大缩小得自己实现，VerticalGridView只是为了方便在Tv上Item的滑动效果而已。

Android Tv中的焦点问题：

<http://www.laoyuyu.me/2017/02/17/%E9%82%A3%E4%BA%9B%E5%B9%B4%EF%BC%8C%E6%88%91%E4%BB%AC%E7%88%AC%E8%BF%87%E7%9A%84%E5%9D%91-Andriud-TV%E7%AF%87%EF%BC%88%E4%B8%80%EF%BC%89/>

# LayoutInflater原理

获取对象方法：



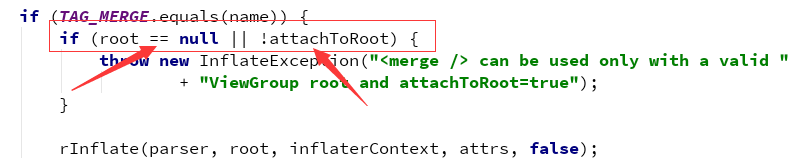
Inflate()方法



inflate方法的作用也就是将xml的布局文件转换成View对象，这样才可以在代码中使用该view。

这个view是否需要父布局，这些就由参数2、参数3来控制。

Inflate过程分两类，一是加载merge标签的布局，二是加载非merge标签的布局。



所以，如果是merge标签的布局的话，必须传入参数2，且参数3为true。



如果是非merge标签的话，且第二个参数root = null时，第三个参数将失去作用。Inflate返回xml的根标签的View对象，且在根标签设置的Layout属性将失去作用，最明显的就是宽高将不起作用。

如果第二个参数root 不为Null时，第三个参数默认为true，此时inflate会自动调用root.addView()，将创建的布局文件的View对象加入到指定的父控件中，所以此时如果在手动调用addview时，将会报错。

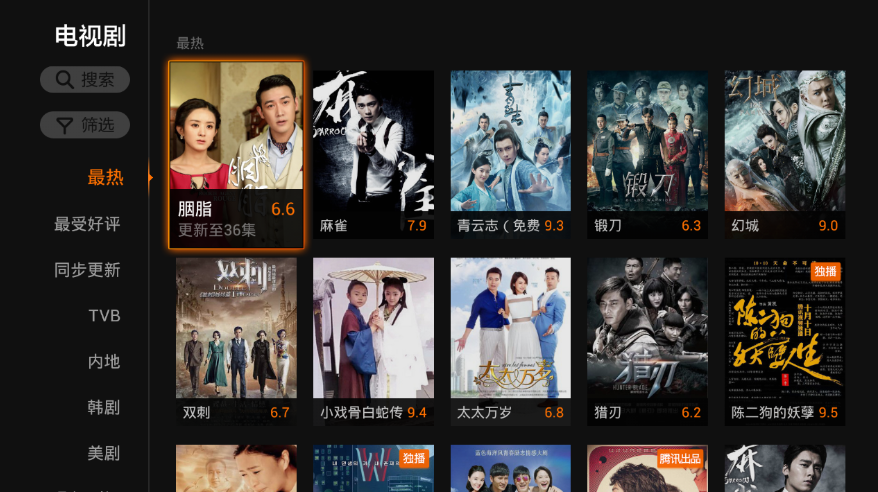
# Tv Home界面

ViewPager + TabLayout + Fragment(RecyclerView?)

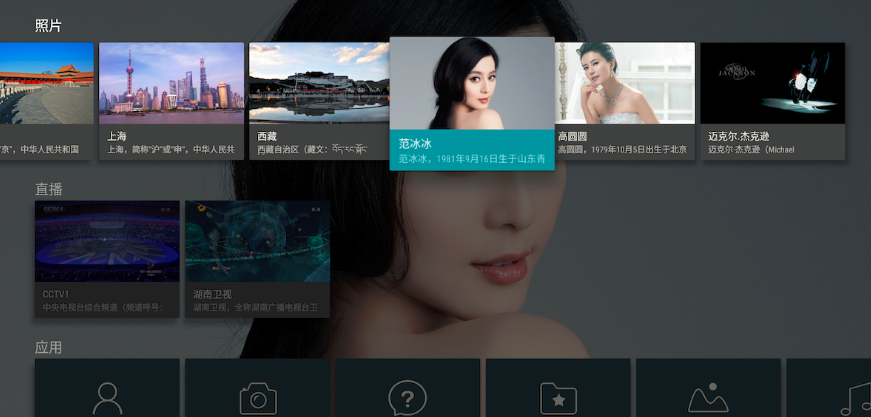


RecyclerView + LinearLayoutManager 左菜单

RecyclerView + GridLayoutManager(5列) 右列表



VerticalGridView + HorizontalGridView



主页json

<https://raw.githubusercontent.com/AiAndroid/stream/master/tv/game/home.json>

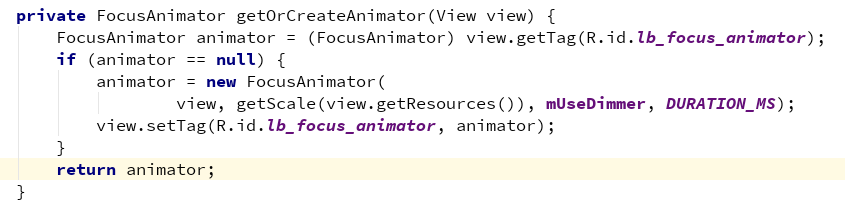
# Google Home界面分析

Item 放大缩小动画

FocusHighlightHelper. BrowseItemFocusHighlight. onItemFocused()



getOrCreateAnimator()：缓存FocusAnimator动画，将动画缓存在View的Tag里面



FocusAnimator是实现TimeAnimator.TimeListener接口的动画，放大缩小的动画就是在这里实现。



动画的实现已经找到，接下去找找如何跟view绑定在一起的。

RowPresenter.onCreateViewHoler()

-> RowPresenter.initializeRowViewHolder()

-> ListRowPresenter.initializeRowViewHolder()

-> FocusHighlightHelper.setupBrowseItemFocusHighlight()

-> ItemBridgeAdapter.setFocusHighlight()

说到底，其实也是在 Adapter 的 onCreateViewHolder 中对 Item View 设置焦点监听，值得学习的一点就是，它还保存了原焦点监听器。这样放大缩小效果就由内部实现，我们仍旧可以使用View 的焦点监听器，不会有任何影响。



布局实现

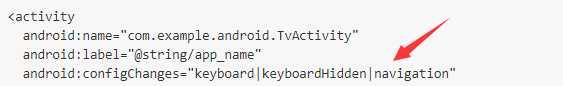
BaseRowFragment创建VerticalGridView，垂直方向的RecyclerView

# Google Tv官方教程

处理手柄掉线

TV的手柄通常是蓝牙设备，它为了省电而定期的休眠并且与TV设备断开连接。这意味着如果不处理这些重连事件，应用可能被中断或者重新开始。

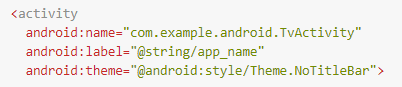
任何TV应用activity相关于断开和重连事件。这些事件必须在应用的manifest配置去处理。

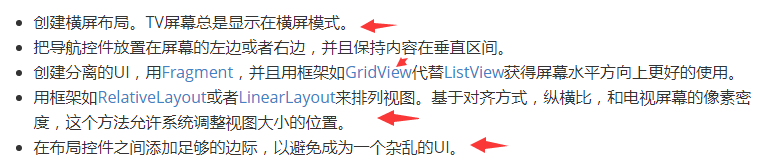


布局注意

无标题栏

如果没有用v17 leanback类，我们应该在TV activity使用这个主题来隐去标题栏的显示。



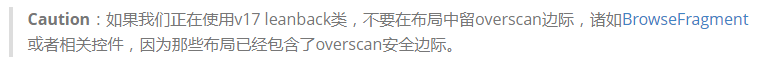


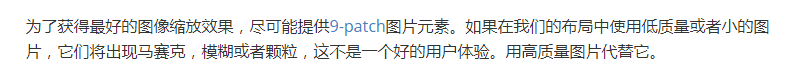
Overscan

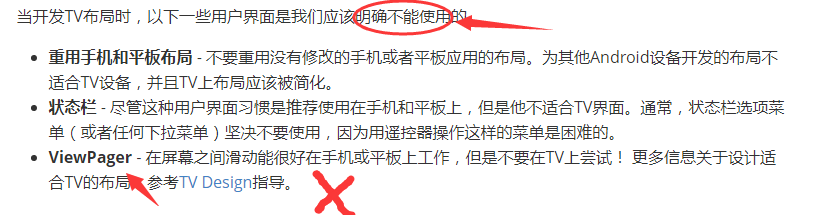
由于TV标准的演进，TV的布局有一个独特的需求是总是希望给观众显示全屏图像。因为这个原因，TV设备可能剪掉应用布局的外边缘去确保整个显示器被填满。这种行为通常简称为overscan。

避免屏幕元素由于overscan被剪掉，可以在布局所有的边缘增加总共10%的边际。这换算为在activity的基础布局上左右边缘留48dp的边际和在上下留27dp的边际。



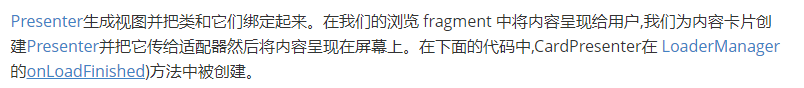


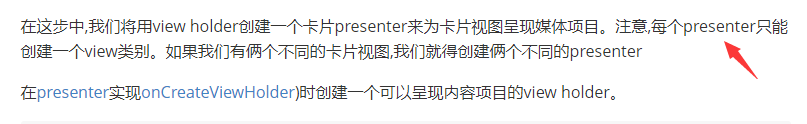












# Android 焦点

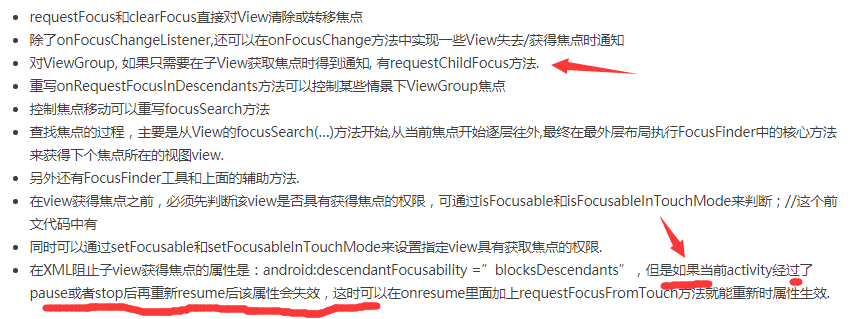


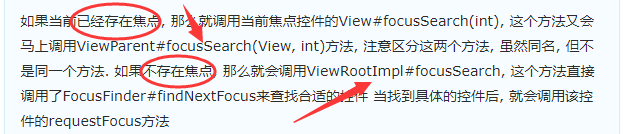




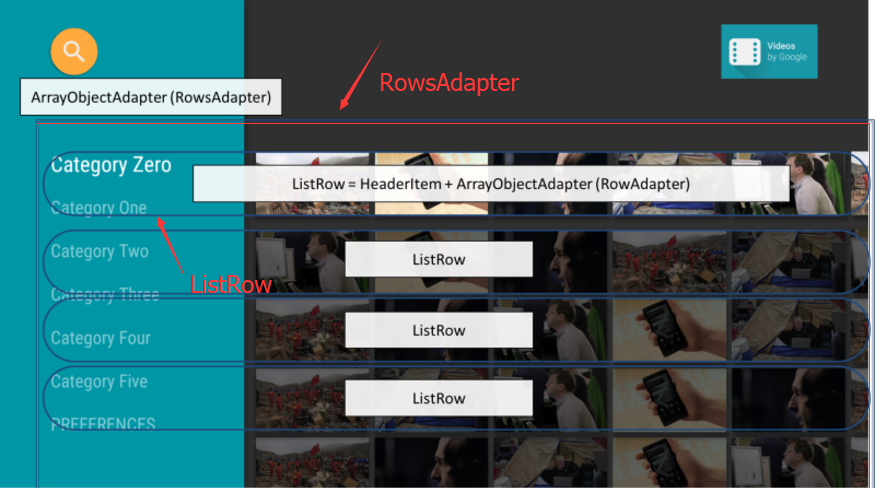


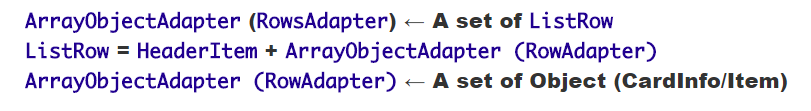






# BrowseFragment解析





不管是每一行的整行item，还是每行里面的每一个card item，都是装在ArrayObjectAdapter里面。

整个页面有唯一一个RowsAdapter，里面有每一行的item，即ListRow对象。一个ListRow对象表示一行的布局，它包括一个HeadItem和一个RowAdapter对象。这个RowAdapter里装着每个card item。

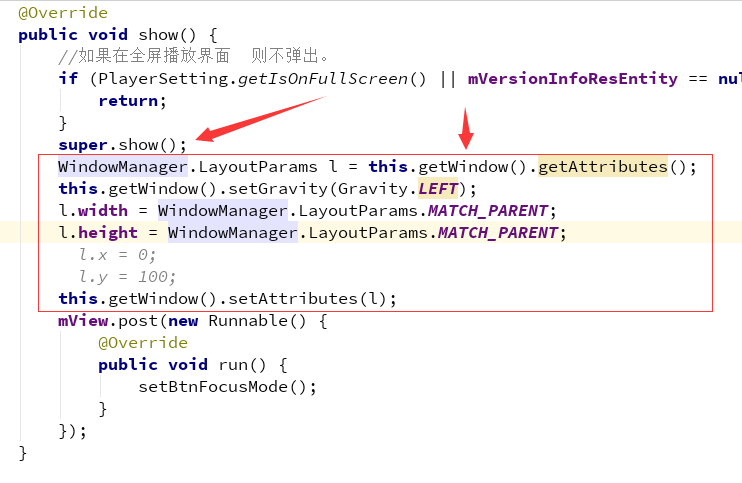
每个ArrayObjectAdapter在创建对象时需要传入一个Presenter对象，Presenter里有内部类ViewHolder，所以该类是用于将Adapter里的数据和Ui绑定起来，有些类似于RecyclerView的Adapter作用，虽然它的命名是Presenter。

示例代码：





# Dialog设置显示位置及大小



必须在super.show()代码下面设置才能生效，因为super.show()又会去改动参数值，导致参数值被覆盖而失效。

另外，Gravity.CENTER|LEFT|TOP等是设置原点坐标的，如果想要让该原点坐标生效，随后必须跟着设置 l.width 和 l.height 属性。

# cms操作指南

* 1. commom\_app\_pref\_config.xml查找layout字段
  2. 查到layout后到cms --》终端编排--》爱奇艺--》找到对应的布局，进去修改。

# 单元测试中通过反射测试私有方法

Context context = **new** MockContext();



# Activity开发艺术探索笔记

Activity启动缓慢

原因持续收集中。。。

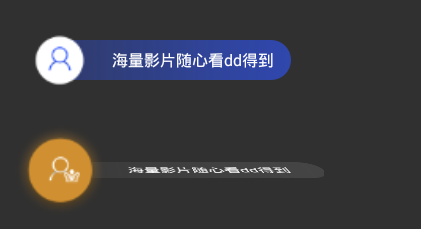
1.可能是由于上一个Activity的onPause里干太多事，新的Activity的onResume必须等旧的Activity的onPause执行完才执行。

开启多进程方式

只有一种办法：android:process属性

# 3D翻页效果

绕 X Y 轴旋转可以实现3D旋转效果，绕 Z 轴旋转实现 2D 旋转效果



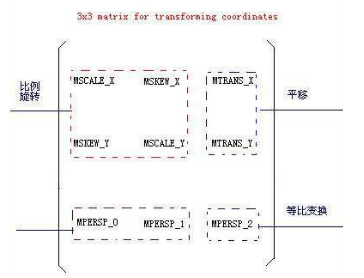
*/\*\*  
 \* 翻页动画的实现  
 \*/***class** Rotate3dAnimation **extends** Animation {  
 **private final float mCenterX**;  
 **private final float mCenterY**;  
 **private** Camera **mCamera**;  
 **private** OnInterpolatedTimeListener **mTimeListener**;  
  
 **public** Rotate3dAnimation(**float** centerX, **float** centerY) {  
 **mCenterX** = centerX;  
 **mCenterY** = centerY;  
 }  
  
 **public void** setOnInterpolatedTimeListener(OnInterpolatedTimeListener timeListener) {  
 **mTimeListener** = timeListener;  
 }  
  
 @Override  
 **public void** initialize(**int** width, **int** height, **int** parentWidth, **int** parentHeight) {  
 **super**.initialize(width, height, parentWidth, parentHeight);  
 **mCamera** = **new** Camera();  
 }  
  
 @Override  
 **protected void** applyTransformation(**float** interpolatedTime, Transformation t) {  
 **if** (**mTimeListener** != **null**) {  
 **mTimeListener**.onInterpolatedTime(interpolatedTime);  
 }  
 **final float** fromDegrees = 0;  
 **float** degrees = fromDegrees + ((180 - fromDegrees) \* interpolatedTime);  
  
 **final float** centerX = **mCenterX**;  
 **final float** centerY = **mCenterY**;  
 **final** Camera camera = **mCamera**;  
  
 **final** Matrix matrix = t.getMatrix();  
 **boolean** overHalf = (interpolatedTime > 0.5f);  
 **if** (overHalf) {  
 degrees = degrees - 180;  
 }  
 camera.save();  
 camera.translate(0.0f, 0.0f, interpolatedTime);  
 camera.rotateX(degrees);  
 camera.getMatrix(matrix);  
 camera.restore();  
  
 matrix.preTranslate(-centerX, -centerY);  
 matrix.postTranslate(centerX, centerY);  
 }  
}  
  
**interface** OnInterpolatedTimeListener {  
 **void** onInterpolatedTime(**float** interpolatedTime);  
}

*//默认旋转中心在（0，0），所以旋转前先把view往上平移 height/2，此时旋转中心就位于view的中间轴  
//因此旋转效果就是view的绕着中间轴翻转*matrix.preTranslate(-centerX, -centerY);  
*//旋转前移动了view，旋转结束后自然要将view移回原位*matrix.postTranslate(centerX, centerY);  
  
*//以上动画操作均在一帧内，所以虽然view是先往上平移，然后翻转一帧的角度后，最后在往下平移回原位  
//但一帧内我们肉眼分辨不清，所以我们看到的效果就是View在原位延中间轴翻转的效果*

# Matrix图像变化矩阵

<http://blog.csdn.net/cquwentao/article/details/51445269>

<http://blog.csdn.net/zhangke3016/article/details/52093776>



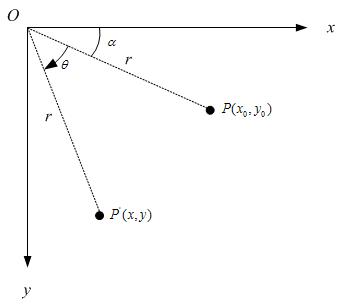
scale是缩放，skew是错切，trans是平移，persp代表透视

这里写图片描述

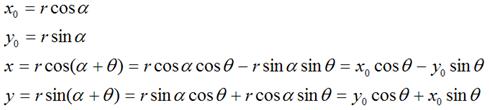
x，y分别代表x，y轴上的坐标，而1代表屏幕在z轴上的坐标为默认的。如果将1变大，那么屏幕会拉远， 图形会变小。（z轴垂直屏幕是正方向）

举例：旋转图像

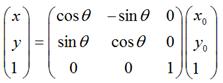
假定有一个点 ，相对坐标原点顺时针旋转后的情形，同时假定P点离坐标原点的距离为r，如下图：



那么就有：



换做矩阵运算就如下图：

 （旋转运算矩阵）

小结：

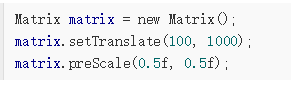
左上角的四个值，可以和x，y值做乘法运算，所以可以影响到旋转等操作，而右上角的模块，只能做加法，所以只能影响到平移。右下角的模块主要管z轴，自然就可以进行等比的缩放了，左下角的模块一般不去动他，否则会把x，y值加入到z轴中来，会不可控。

复合运算操作

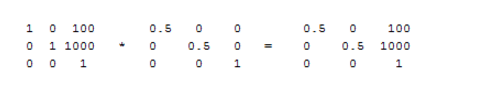
有关于变换的set方法都可以带来不同的效果，但是每个set都会把上个效果清除掉，例如依次调用了setSkew,setTranslate，那么最终只有setTranslate会起作用，那么如何才和将两种效果复合呢。Matrix给我们提供了很多方法。但是主要都是2类：

preXXXX:以pre开头，例如preTranslate   
postXXXX:以post开头，例如postScale

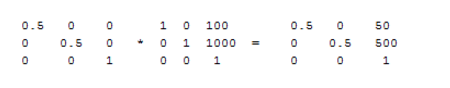
他们分别代表了前乘，和后乘。看一段代码：



这里matrix前乘了一个scale矩阵，换算成数学式如下：

pre

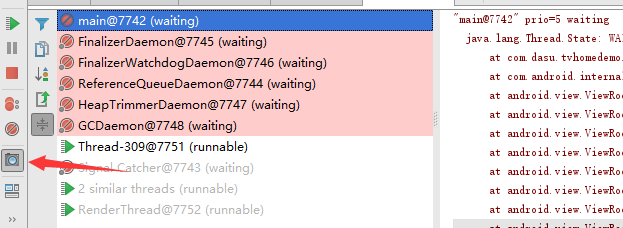
从上面可以看出，最终得出的matrix既包含了缩放信息也有平移信息。   
后乘自然就是matrix在后面，而缩放矩阵在前面，由于矩阵前后乘并不等价，也就导致了他们的效果不同。我们来看看后乘的结果：

post

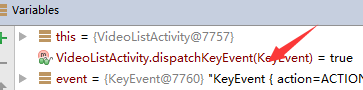
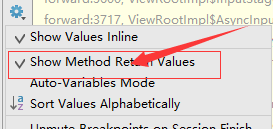
可以看到，结果跟上面不同，并且这也不是我们想要的结果，这里缩放没有更改，但是平移被减半了，换句话说，平移的距离也被缩放了。所以需要注意前后乘法的关系。

# AS 调试技巧

查看当前线程信息

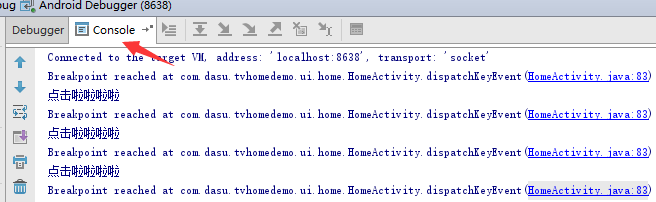


开启显示方法的返回值

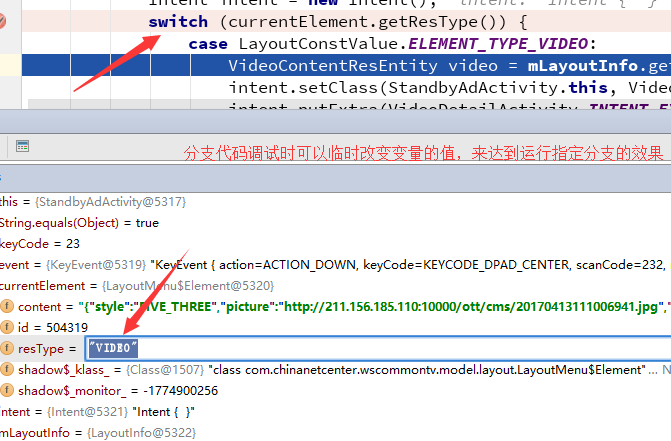


不添加代码临时添加日志输出





临时改变变量的值



# Chrome插件

<http://stormzhang.com/devtools/2016/01/15/google-chrome-extension/>