## 第三章 View工作原理

View直接继承Object类 TextView直接继承View

Left=getLeft（）；

从android3.0开始 View增加了额外的参数：x y translationx和Tracslationy，其中x和y是View的左上角的坐标，而tansaltionY是View左上角相对于父容器的偏移量，这几个参数也是相对于父容器的坐标，并且tanslationy和tansaltionx默认值都是0

x=left + tanslattionX…

MotionEcent

ACTION\_DWON ACTION\_MOVE

ACTION\_UP

getX/getY 返回的事相对于当前的view左上角的X和y坐标，而getRawX／getRawY返回的事相对于手机屏幕左上角的xy坐标

TouchSlop 是系统识别最小滑动的距离

VelocityTracker 用于追踪手指在滑动过程中的速度，包括水平和竖直方向的速度，他的过程很简单。

GestureDelector 手势识别，用于辅助监测用户的单机华东长按双击等行为。

Scroller 弹性滑动对象，用于实现View的弹性滑动，我们知道，当我们使用scrollTo／scrollBy方法来进行滑动，其过程是瞬间完成的，这个没有过渡效果，用户体验不好，这个时候就可以使用Scroller来实现过渡效果的滑动，其过程不是瞬间完成的。而是在一定的时间间隔内完成，他需要和View的ComputeScroll方法配合使用才能共同完成这个功能。

通过三种方式可以完成View的滑动：第一种是通过View本省提供的scrollTo/ScrollBy方法来实现滑动，第二种就是通过动画View施加平移效果实现滑动，第三种就是通过改变View的LayoutParams使得View重新布局从而实现滑动。ScrollBY内部也是调用了scrollTO方法

注意：使用scrollTO和scrollBy来实现View的滑动，只能将View的内容进行滑动，并不能将View本身进行滑动。也就是说不管怎么滑动，也不可能将当前View滑动到附近View所在的区域。

使用动画主要是操作view的tanslationx和translationY属性。

View动画是对view的影像做操作，他不能真正改变view的位置参数，包括宽高。如果希望动画后的状态得以保存还必须将fillAfter属性设置为trure，否则动画完成会自动消失。

通过改变布局参数来实现滑动

MarginlayoutParams params=(MarginLayoutParams)mButton1.getLayoutParams();

Params.width+=100;

Params.leftMargin+=100;

mButton1.requestLayout();

三种方法对比：

scrollBy scrollTo他是View提供的原生方法，其作用是专门用于View的滑动，它可以比较方便实现滑动效果并且不影响内部元素的单击事件，但是他只能滑动View内容，并不能滑动View本身

View动画不能改变View本身的属性，如果动画元素不需要影响用户的交互可以使用动画。一些复杂效果必须要通过动画才能实现

改变布局参数：操作稍微复杂，适用于交互的View

事件分发机制：

public Boolean dispatchTouchEvent（MotionEvent ev）

用来进行事件分发。如果事件能够传递给当前的view，那么此方法一定会被调用，返回结果受当前View的onTouvhEvent和下级View的dispathcTouchEvent方法的影响（看下面的伪代码）

public Boolean onInterceptTouchEvent(MotionEvent ev)

在上述方法中内部调用，用来判断是否拦截某个事件，如果当前view拦截了某个事件那么在同一个事件序列中，此方法不会被再次调用，返回结果表示是否拦截当前事件

public boolean onTouchEvent（MotionEvent ev）

在dispatchEvevent方法中调用，用来处理点击事件，返回结果表示是否消耗当前事件，如果不消耗，则在同一事件序列中，当前view无法再次接收到事件

public Boolean dispatchTouchEvent(MotionEvent ev){

Boolean consume=false;

If(onInterceptTouchEvent(ev)){

Consume=onTouchEvent(ev)

}else{

consume=child.dispatchTouchEvent(ev)

}

return consume;

}

这段为代码表示对于一个viewGroup，点击时间产生后·，首先会传递给它，这时它的dispathTouchEvent就会被调用，如果这个ViewGroup的onInterceptTouchEvent方法放回true就表示他要拦截当前事件，接着事件就会交给这个ViewGroup处理，即它的onTOuchEvent方法就会被调用，如果这个ViewGroup的onInterceptTouchEvent方法返回false就表示当前·事件，这是当前事件就会继续传递给他的子元素，接着子元素的dispatchTouchEvent方法就会被调用，如果此反复直到事件被最终处理

当一个View需要处理事件时，如果他设置了OnTOuchListener，那么OntouchListencer中的ontouch方法就会被调用，这世间如何处理还要看他的onzTouh返回值，如果返回false，则当前view的onTOuchevent方法就会调用，如果返回true，那么onTOuchEvent方法将不会被调用，由此可见，给View设置onTOuchListeneer，其优先级比onTouchEvent要搞，在onTouchEvent方法中过，如果当前设置有onClickListener，那么它的onClikc方法会被调用，可以看到 onClickListener优先级最低

当一个点击事件产生后，他传递过程如下顺序：Activity-》Window-》View，即。顶级View接收到消息·，就会按照事件分发机制去分发事件，如果一个View的onTOuchEvent返回false，那么他的父容器的ontouchEvent将会被调用，以此类推。如果所有的元素都不处理这个事件，那么这个最终传递给Activity处理，即Activity的onTouchEvent方法会被调用。

1. 同一个事件序列是指从手指接触屏幕的那一刻起，到手指离开屏幕的那一刻结束，在这个过程中一系列事件，这个事件序列以down事件开始，中间含有数量不定的move事件，最终以up事件结束
2. 正常情况下，一个事件序列只能被一个view拦截且消耗。可以参考3，因为一旦一个元素拦截此事件，那么同一个事件序列内的所有事件都会直接交给他处理，因此同一个事件序列中的事件不能分别由两个view同时处理，但是特殊外，比如一个view将本该自己的处理的事件通过onTouchEvent强行传递给其他的view处理
3. 某个view一旦决定拦截，那么这一个事件序列都只能由他处理（如果事件序列能够传递给他的话），并且他的onInterceptTouchEvent不会被在调用。就是说当一个view决定拦截一个事件后，那么系统会把同一个事件序列内的其他方法都直接交给她处理，因此不用在调用这个view的onInterceptTouchEvent去询问她是否要拦截
4. 某个view一旦开始处理事件，如果他不消耗ACTION-DOWN事件（onTOuchEvent返回了false），那么同一事件序列中其他事件都不会再交给他来处理，并且事件将重新交由它的父元素去处理，即父元素的onTouchEvent会被调用。事件一旦交给一个View处理，那么他就必须消耗掉，否着同一事件序列中剩下的事件就不再交给他处理。
5. 如果view不消耗ACTIOINDOWN以外的其他事件，那么这个点击事件就会消失，此时父元素onTuochEvent并不会调用，并且当前view可以持续收到后续的事件，最终这些消失的点击事件会传递给Activity处理
6. ViewGroup默认不拦截任何事件。Android源码中viewGroup的onInterceptTouchEvent方法默认返回false
7. View的onTouchEvent默认都会消耗事件（返回true），除非他是不可点击的（clickable和longclickable同时是false）。View的longClickable属性默认都是为flaes，click属性要分情况，比如button的lcickable属性默认为trure，而TextView的clickable属性默认为false
8. View没有onInterceptTouchEvent方法，一旦有点击事件传递给他，那么它的onTouchEvent方法就会被调用
9. View的enable属性不影响onTouchEvent的默认返回值，哪怕一个view是disable状态的，只要他是的clickable或者longclick有一个为true，那么它的onTouchEvent就返回ture
10. Onclick会发生的前提是当前view是可点击的，并且它收到了down和up的事件
11. 时间传递过程是由外向内的，即事件总是先传递给父元素，然后再由父元素分发子view，通过requestDisallowInterceptTouchEvent方法可以在子元素中干预父元素的事件分发过程，但是Actiondown事件除外

事件分发机制源码解析：

1Activity对点击事件分发

点击事件用MotionEvent来表示，当一个点击事件操作发生后，事件最先传递给当前的Activity 由Activiyt的dispatchTouchEnvent来进行事件分发，具体的工作由Windows来完成的，windows会将事件传递给décorview、décorview一般就是当前界面的底层容器，（即setContentView所设置的View容器）通过Activity.getWindow.getDecorView()可获得。我们分析activtiy的dispatvhTouchEvent开始分析

public Boolean dispatchTouchEvent（Motion ev）{  
if(ev.getAction()==MotionEvent.Action\_dow){

onUserInteraction()

}

if(getWindow().superDispatchTouchEvent(ev)){//这个是抽象方法  
return ture;

}

return onTouchEvent(ev)  
}

首先时间开始交给Activity，所附属的windwos进行分发，如果返回true，整个事件循环就结束了，返回false表示事件没人处理，所有的View的OntouchEvetn都会返回false，那么Activyt的ontouchEvent就会被调用

看到注释处，windwos是一个抽象类，而windows是super dispatchtouvhevent方法也是一个抽象方法，因此我们必须找到windows实现类才行

public abstract Boolean superDispatchTouchEvent(MotionEvent ev)

其实windows（唯一）实现类就是phonewindwos，这一点从windows的园中也可以看出来，在windows说明中，有这么一段话：  
 windows类可以控制顶级View外观和行为策略，他的唯一实现位于android.policy.phonewindows中，当你要实例化这个windows类的时候，你并不知道他的细节，应为这个类会被重构，只有一个工厂方法可以调用，尽管这个看起来有点模糊，不过我们可以看一下android.policy.PhoneWindows这个类：

publc Boolean superdispatchTouchEvent(motion ev){  
return mDecor.superDispatvhzTouvhEvent(evnt)  
}

到这里逻辑就很清楚，PhoneWindow将事件直接传递给了DecorView，这个DecorVIew是什么呢？

Private final class DecorView extends FrameLayout implements RootViewSurfaceTaster{

Private DecorView mDecor;

Public final View getDecorView(){  
if(mDecor==null){

installDecor();  
}

return mDecor;  
}

我们知道通过((VIewGroup)getWindow().getDecorView().findViewById(Androdi.r.id.content)).getChildAt(0)这种方式就可以获取Activity所设置的mDecor显然就是getWindow().getDecorView()返回的View，而我们通过setContentVIew设置的View是它的一个子View，目前时间传递到了DecorVIew这里，由于DecorView继承自FrameLayout且是父view，所以最终事件会传递给View，事件肯定会传递给View，不然应用如何响应点击事件，事件已经传递到顶级View，即在Activity中通过setContentView所设置的View，另外顶级View，一般来说就是VIewGroup

3 顶级View对点击事件分发过程

点击时间产生后到达顶层View（一般是一个VIewgroup）以后，会调用VIewGroup的dispatchTouchEvent方法，如果顶级ViewGroup拦击事件即onInterceptTouchEvent 回true，则事件由ViewGroup处理，这时如果ViewGroup的mOntouchListencer被设置，则onTouch会被调用，否者onTouchEvent会被调用，也就是说，如果都提供的话，onTouch会屏蔽掉onTouchEvent，在onToucheEvent中，如果设置了mOnclickListener，则onzclick会被调用。如果顶级ViewGroup不拦劫事件，则事件会传递给他所在的点击事件链上的子View，这时子view的dispatchTouchEvent会被调用，到此，事件已经从顶级view传递给下一层View，接下来的传递过程和顶级view是一致的

首先看ViewGroup对点击事件的分发过程，其主要实现在ViewGroup的dispatchTouchEvent方法中，这个方法比较长，分段说明：

final Boolean intercepted;

if(actionMasked==MotionEvent.ACTION\_DOWN||mFristTouchTarget!=null){

final Boolean disallowIntercept=(mGroupFlags&FLAG\_DISALLOW\_INTERCEPT)!=0;

if(!disallowIntercept){  
intercepted=onInterceptTouchEvent(ev);

ev.setAction(action);  
}else{

inrercepted=false

}

}else{

inrecepted=true;

}  
mFirstTouchTarget!=null表示当事件由ViewGroup的子元素成功处理时mFirstTouchTarget指向子元素，ViewGroup不拦截事件并将事件交给子元素处理时。一旦事件由当前ViewGroup拦截时，mFirstTouchTarget为null。那么当Action\_move和action\_up事件来到时，将导致ViewGroup的onInterceptTouchEvent不会被调用，并且同一序列中的其他时间都会默认交给他处理。  
第四章：View的工作原理

除了View的三大流程以外，View常见的回调方法也是需要熟练掌握，比如构造方法，onAttach、onVisibilityChanged、onDetach等，另外对于一些具有滑动效果的自定义view，我们还需要处理view的滑动，如果遇到滑动冲突就还需解决性对应的滑动冲突，

初识ViewRoot和DecorView

ViewRoot对应于ViewRootImpl类，它是连接WindowMangaer和DecorVIew的纽带，View的三大流程均是通过ViewRoot来完成的，在ActivityTHread中，党Activity对象创建完毕后，会将DecorView添加到Windows中，同时会创建ViewRootImpl对象，并将ViewRootImpl对象和DecorVIew建立关联·，这个过程可以参观如下源码：

Root=new ViewRootImpl(view.getCOntext(),display);

Root.setView(view,wparams,panelParentVIew);

View的绘制流程是从ViewRoot和performTravesals方法开始的，他经过measure、layout和draw三个过程才能最终将一个View绘制出来，其中measure用来测量View的宽高，layout用来测量View在父容器中的放置位置而draw则负责将View会知道屏幕上，针对performTraversals大致流程：  
 performTraversals会一次调用performmeasure、performlayout和performdrwas，三个方法，这三个方法分别完成顶级Vie的measure、layout和jdraw着三个流程，其中在performTravesals会调用measure方法，在measure方法中会有屌用onmeasure方法，早起onmeasure方法中则会对所有子元素进行measure过程，这个时候measure流程就从父容器传递到子元素，这样就完成了一次mesure过程，接着子元素会重复夫容器的·1measue过程，如此反复就完成了整个viw树的遍历，同理，performlayout和performdraw的传递过程和performmeasure类似，唯一不同的，performdraw的传递过程实在draw方法通过dispatcherdraw来实现，不过这并没有·本质区别·

Measure过车决定了View的宽高，measure完成以后，可以通过getMEasuredWidth和getMeasuredHeight方法来获取到View测量后的宽高，在几乎所有的情况下他都等同于View最终的宽高，但是特殊情况除外，1这点在本章后面会进行说明，Layout过程决定了View的四个顶点的坐标和实际的View宽高，完成以后，可以通过getTop、getBottom、getLeft和getRight来拿到View的四个顶点位置，并可以通过getWIdth和getHeight方法来拿到View的最终宽高。Fraw过程则决定了View的显示，只是Draw方法完成以后View的内容才能显示在屏幕上

DecorView作为顶级View，一般情况下它内部会包含一个竖直方向的LinearLayout，在这个LinearLayout里面有上下两个部分，上边是标题栏，下边是内容栏。在Activity中我们通过setContentView所设置的布局文件其实就是被加到内容栏。在内容栏id是Content，因此可以理解为Activity指定布局的方法不叫setView而叫setcontentView，因为我们的布局确实加到了id为content的framelayout中，如何得到content呢？ViewGroup content=findViewById（r.android.id.content）如何得到我们设置的View？content.getChildAt(0).同时，通过源码我们发现，decorView其实就是一个FrameLayout，View层的都先经过DecorView，然后才传递·给我们的View

理解measureSpec：确切来说，MeasureSpec在很大程度上决定了一个View的尺寸规格，之所以说是很大程度上是应为这个过程还搜到父容器影响，因为父容器影响View的MeasureSpec的创建过程，在测量过程中，系统会将View的LayoutParams根据夫容器所施加的规则转换成对应的MeasureSpec，然后在根据这个measureSpec来测量出View的宽高，这里的宽高是测量宽高，不一定等于View的最终宽高，MeasureSpec看起来有点复杂，其实他的实现很简单：

MeasureSpec代表一个32位int值，高2位代表SpecMode，低30位代表SpecSize，SpecMode是测量模式，而SepcSize是指在某种测量模式下的规格大小，下面先看看MeasureSpec内部的一些常量的定义：  
MeasureSpec通过将SpecMode和SpecSIze打包成一个int值来避免过多的对象内存分配，为了方便操作，其提供了打包和解包方法，SpecMode和SpecSize也是一个int值，一组SpecMde和SpecSize可以打包为一个MEasureSpec，而一个MeasureSpec可以通过解包的形式来得出其原始的SpecMode和SpecSize需要注意的是这里提到的MeasureSpec是指MeasureSpec所代表的int值，而并非MeasureSpec本身。

SpecMode有三类，每一类代表特殊的含义  
UNSPECIFiED

父容器不对View有任何的限制，需要多大给多大，这种情况一般用于系统内部，表示一种测量的状态

Exactly

父容器已检查出View所需要的精确大小，这个时候View最终打小就是SpecSize所指的值，他对应于LayoutParams中的match\_parent和具体的数值这两种模式

AT\_Most

父容器指定了一个可用的大小即SpecSize，View的大小不能大于这个值，具体是什么值要看不同的View的具体实现，他对应于LayoutParams中的wrap\_content

MeasureSpec和LayoutParams的对应关系

系统内部都是通过measureSpec来进行view的测量，在View的测量的时候，系统会将LayoutParams在父容器的约束下转化成对应的MeausreSPec，然后再根据这个MeasureSpec来确定view测量后的宽高。MeasureSpec不是唯一由LayoutParams决定的，LayoutParmas需要和父容器一起才能决定View的MeasureSpec

对于DecroView，其MeasureSpec由窗口的尺寸和其自身的Layoutparams来共同决定，对于普通View，其MeasureSpec由父容器的MeasureSpec和自身的LayoutParmas来共同决定，MeasureSPec一旦确定后，onMEasure中就可以确定View的测量高

对于ViewDecor的MEasurespec 产生如下规格根据LayoutParams中的宽高参数来划分  
LayoutParams.MATCH\_PARENT:精确模式，大小就是窗口大小：  
LayoutParams.WRAP\_CONTENT最大模式，大小不定，但是不能超过窗口大小，

固定大小：精确模式

对于普通View来说，这里是指我我们布局中的View,View的measure过程由ViewGroup传递而来，先看一下ViewGroup的measureChildWithMargins方法

这个方法会对子元素进行measure，在调用子元素的measure方法前会先通过getChildMeasureSpec方法来得到子元素MeasureSpec。从代码来看，显然，子元素的meaaurespec的创建与父容器measurespec和子元素本身的LayoutParams有关，此外还和View的margin及pading有关，具体情况可以看看ViewGroup的getChildMeasureSpec方法

这个方法不难理解，它的主要作用是根据父容器的MeasureSPce同时结合view本身的LayoutParams来确定子元素的MeasureSpec，参数中padding是指父容器中已占用的空间大小，因此子元素可用的大小为父容器的尺寸减去padding，具体代码如下：  
int specSize=MeasureSpec.getSize(spec)

int size=Math.max(0,specsize-padding)

getChildMeasureSpec清除展示了普通的View的MeasureSpec的创建规则，为了更清楚地理解getChildMEasure的逻辑，这里提供一个表，表中对getChildMeasureSpec的工作原理进行了梳理：  
 从这个表我们能确认：当View采用固定宽高的时候，不管父容器的MeasureSpec是什么，View的measureSpec都是精准模式，并且大小遵循LayoutParams中的大小。当view宽高时match\_parent时，如果父容器是最大模式，那么View也是最大模式并且其大小超过父容器的剩余空间，当View宽高时wrap content时，不管父容器模式是什么，view的模式总是最大化并且不能超过父容器的剩余空间。

View的工作流程：

Measure 确定view的测量宽高，layout确定view的最终宽高和四个顶点的位置，draw则将view绘制到屏幕上。  
 measure过程：

Measure过程要分情况，如果是一个原始的view，那么通过mesure方法就能完成了测量过程，如果是一个viewgroup，除了完成自身的测量过程，还会遍历调用所有的子元素的measure方法，

View的measure过程由measure方法来完成，measure方法是一个final类型的方法，不能重写此方法，在view的measure方法中去调用view的onMeasure方法，因此只需要看onMeasure实现即可，view的onMeasure方法·：

Protected void onmeasure（int widthMeasureSpec,int heightMeasureSpec）{  
setMeasuredDimension(getDefaultSize(getSuggetedMinimumWidth),getDefaultSize(getSuggestedMinimumHeight),heightMeasureSpec)};  
}

setMeasuredDimension方法设置view宽高的测量值。

getDefaulSize这个方法放回的是measureSpec中的specSize，而这个specSize就是view测量后的大小，view最终的大小是在layout确定的，但是几乎所有情况下view的测量大小和最终大小是相等的

。。。

我们可以直接的出结论：直接继承View的自定义控件需要重写onMeausre方法并设置wrap content时的自身大小，否则在布局中使用wrap content就相当于使用match parent。

## Android 的drawable

Drawable有很多种，他们都表示一种图像钙奶呢，但是他们又不全是图片，剔骨工颜色可以构造出各式各样的图像效果，在实际开发中，Drawable常被用来作为View的背景使用，Drawabl一般都是通过xml来定义的，当让我们也可以通过代码创建具体的Drawable对象，只是用代码创建稍显复杂，在Andrid的设计中，Drawable是一个抽象类，它是所有的Drawable对象的基类，每个具体爹Drawable都是他的子类，不如ShapeDrawable、BitmapDrawable等，

Drawable的内部宽高这个参数比较重要，通过getIntrinsicWidth和getintrinsicHeight这两个方法可以获取到他们。但是并不是所有的Drawable都有内部宽高，比如一张图片所形成的Drawbale，它的内部宽高，但是一个颜色所形成的Drawable她就没有内部宽高，另外需要注意的是，Drawable的内部宽高不等同于他的大小，一般来说，Drawable是没有大小概念的，当作为View的背景时，Drawable会被拉伸至View的同等大小

Drawable的分类

BitmapDrawable 他表示就是一张图片，我们可以直接引用原始图片，但是也可以通过xml的方式来描述它，通过xml来描述的BitmapDrawble可以设置更多的效果，

## Android 动画深入分析

View动画的作用对象是View，他只吃四种动画：分别是平移动画、缩放动画、旋转动画和透明度动画。出了这四种动画外，帧动画也属于View动画，但是帧动画表现形式和上面四种变换效果不一样，为了更好区分四种变换和帧动画。

View动画四种变换效果对应Animation四个子类：TranslateAnimation、Scaleanimation、RotateAnimation和AplhaAnimation，这四种动画可以通过xml来定义，也可以通过代码来动态创建，对于View动画来说，建议采用XML来定义动画可读性好