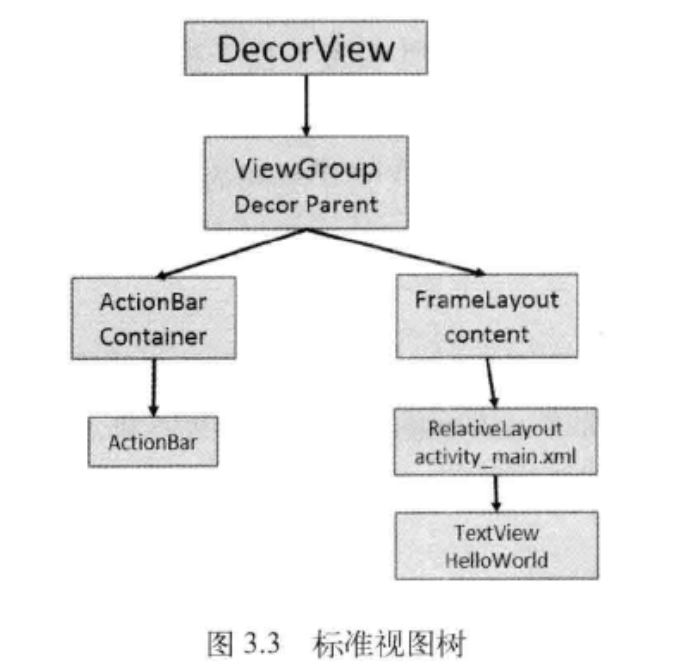
android 群英传

控制树 上层控件负责下层子控件的测量与绘制，并传递及交互事件。Activity使用的findVidwById()方法，就是在控制树中以树册深度优先遍历来查找对应的元素

Phonewindows将一个DecorView设置为整个应用窗口的根view。Decorview封装了一些方法，将要显示的内容呈现在PhoneWidows上，这里所有的view监听事件都是通过windowsMangerserver来进行接受，并通过Activiyt对象来回调对应的onClickListener。 在显示他将屏幕分成两部分，一个是titleview，另一个是contentView。 ContentView是一个ID为content的frameLayout，activity\_main.xml就是设置在这样一个FrameLayout里

图中所示的视图树第二层封装了一个linearLayout，作为viewGroup，这一层的布局结构会根据对应的参数设置不同的布局，如最常用的布局—上面显示的Titlebar，下面是content这样一个布局，如果用户设置requestwindowsfeature（windows.feature\_no\_title）来显示全屏，视图树中的布局就只有content，这就是解析了为什么reqeustwindowseature方法一定要在调用setcontentview方法之前才能生效的原因，

在代码中，当程序onCreate方法中调用了setcontentview方法后，activitymanagerservice会回调onResume方法，此时系统才会把整个decore添加到phonewinds中，并让其显示出来

自定义view

view类默认的onmeasure方法只支持exactly模式，如果在自定义歌控件时候不重写onmeasure方法的话，就只能使用exactyly模式。空间可以响应你指定的具体宽高值或者match\_parent属性，而如果要让自定义view支持wrap content属性，那么就必须重写on measure方法来指定wrap content的大小

通过measurespec这一个类，我们就获取了view的测量模式，和view想要绘制的大小。

实例：

protecte void onmeasure（int widthMeasurespec,int heightmeasureSpec）{  
super.onMeasure(widthMeasurespec,heightmeasurespec);  
}

在ida中按住ctrl 查看super onmeasure方法，可以发现，系统最终会调用setmeasureDimension（int measuredwindth，int measuredheight）方法将测量后的宽高值设置进去 从而完成测量工作，所以再重写onmeasure方法后，最后在哪个要做的工作就是把测量后的宽高值作为参数设置给setmeasuredimensiion方法

通过分析我们重写onmeasure方法：

protected void onmeausre(int widthmeasurespece,int heightmeasurespec){  
setmeasuredimentsion(measurewidth(widthmeasurespec,heightmeasurespec),  
)

在onmeasur方法中，我们调用滋生的measurewidht measureheight方法，分别对宽高重现定义。

我们以measurewidth方法为例，讲解如何自定义测量值

第一步 从meausrpe对象中提出具体的测量模式和大小

int specmode=measurespec.getmode(measurespec)

int spectsize=measurespec.getsize(measurespec)

接下来，通过判定测量模式，给出不同的值，当specmode为exactly时，直接使用specsize即可，党specmode为其他两种模式，需要给他一个默认的大小，特别地，如果指定wrap content属性，即at most 模式，则需要取出我们指定的大小与specsieze中的最小的来作为最后的测量值，measurewidth方法：

private int measurewidth（int measure）  
{  
int result=0;

int specmode= measurespec.getmode(measurespec)

int specsize=measurespec.getsize(measurespec);

if (specmode=measurespec.exactyly){  
result=specsize;  
}else{  
result =200;

if(specmode=measurespecc.at\_most){  
result=math.min(result,spectsize);

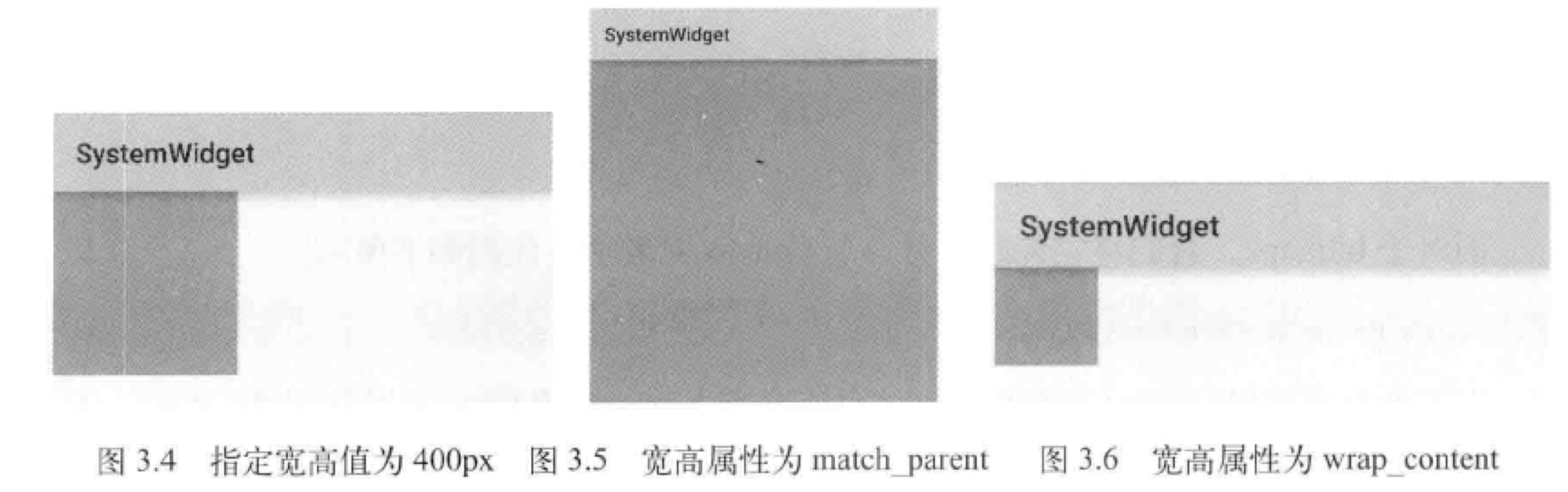
}

}

return result;

measureheight方法与measurewidht基本一致

当我们指定宽高属性为wrap content属性时，如果不重写onmeasure方法，那么系统就不知道该使用默认多大的尺寸，因此，他就会默认填充整个父布局，所重写onmeasure方法目的，就为了能够给view一个wrap content属性下的默认大小



view 绘制

当测量好一个view之后，我们就可以简单重写ondraw方法，并在canvas对象上来绘制所需要的图形，首先我们来了解一下利用系统2d绘制api所需要使用到的canvas对象。

要想在android界面中绘制相应的图像，就必须canvas上进行绘制，canvas就像是一个画板，使用paint就可以在上面作画了，通常需要通过继承view并重写它的ondraw方法来绘制图。

Ondraw 有一个参数，就是canvas avas对象，使用这个canvas对象就可以进行绘制，而在其他地方放，通常需要使用代码创建一canvas对象，

Canvas canvas=new canvas（bitmap）

当创建一个canvas对象时，为什么要穿进一个btimap对象，如果不穿入一个bitmap对象， 他通过这个bitmap创建的canvas画布紧密联系在一起的，这个过程称为装载画布。这个bitmap用来存储所有绘制的canvas上的像素信息。所以当你通过这种方式创建了一个canvas对象，后面调用所有的canvas.drawxxx方法都发生在这个bitmap上，如果在view类的ondraw方法中，通过

canvas.drawbtimap(bitmap1,0,0,null)

canvas.drawbitmap(bitmap2,0,0,null);

了解cavas与bitmap关系

而对于 bitmap2 我们将它装载到另一个canvas对象中

canvas mcanvas=new vanvas（bitmap2） mcavas.drwaxxx

通过mcavas 将绘制效果作用在bitmap2 上，在刷行view的时候，就会发现通过ondraw方法画出来的bitmap2已经发生改变，这就是因为bitmap2承载了在mcanvas上所进行绘制的图操作，虽然我们也使用canvas绘制的API，但其实并没有将图形直接绘制在ondraw方法指定的画布上，而是通过改变bitmap，然后让view重绘，从而显示改变之后的bitmap。

如果该view还需要使用wrap content属性，那么还必须重写onmeasure方法

在view中通常以下比较重要的回调方法

onfinshinflate 从xml加载组件厚回调。

Onsizechanged 组件大小改变时回调。

onmeasure onlayout

ontouchevent 监听到触摸事件时回调。

对现有Textview进行扩张，原生的Textview使用onDraw方法绘制要显示的文字，当继承了系统的Textview之后，如果重写onDraw方法，则不会修改TextView任何效果，

protected void onDraw(Canvas canvas){  
super.onDraw(canvas);  
}

程序调用super.ondraw(canvas)方法实现原生空间的功能，但是在调用super.ondraw()方法前喝之后，我们都可以实现自己的逻辑，分别在系统绘制文字前后，完成自己的操作。

Protected void on Draw(Canvas canvas){  
//对于TextVIew来说既是在绘制文本内容前

Super.ondRzw(canvas)

//对于TExtView来来说既是在绘制文本内容后

}

而代码最重要的部分则是在onDraw（）方法中，为了改变原声行为，在系统调用suepr ondraw（canvas）方法前，也就是在绘制文字之下，绘制两个不同大小的矩形，形成一个重叠效果，再让系统调用super ondraw canvas 方法，执行会指文字的工作。

创建符合空控件，创建符合控件可以很好创建出具有重用功能的控件集，这种方式通常要挤成一个合适的ViewGroup，再让它添加指定的可配置的属性。让它具有更强的扩张性。  
 通常情况下，这些界面都会被抽象出来形成一个公用的UI组件·，所有需要添加标题栏的结脉你都会应用这样的一个TopBar，而不是每个界面都在布局文件中写一个这样一个Torbar，这样不仅可以提高界面的服用率，更能再修改Ui时，做到快速修改，而不需要对每个页面的标题栏都进行修改

1 定义属性

只需要在res资源目录中values目录下创建一个attrs xml比属性定义文件，并在该文件中修改相应的属性即可

resoures

declare stulybale name

attr name

我们在declare styleable 标签申明了使用自定义属性，病痛哟name属性来确认引用的名称，最后，通过attr标签来生命具体的自定义属性，比如在这里定义标题文字的字体大小颜色，等。并通过format属性来指定属性类型。有些属性可以是属性属性，也可以是引用属性。比如按钮的背景，可以指定为具体的颜色，也可以把它指定为一张图片。所以使用 | 来分隔不同的属性

TypedAtrry 这样数据结构或自定义属性集，后面引用styleable的Topbar就是我们在xml中通过decalre styleable name=“tupbar”所指定的name名

通过TypedArray 对象的getString getColor等方法，就可以获取这些定义属性值

TypedArray对象的getString getColor 等方法，

TypedArrya ta=context.obtainStyledAttributes(attrs,R.stylaeable.Topbar)

引用UI模版

自然是需要使用的地方引用UI模版，在引用前，需要指定引用第三方控件的名字空间，在布局文件中：  
xmlns：android=“http：//scemas.android.com/apk/res/android”

这行行代码就是在指定引用的名字空间，xmlns 即 xml namespace 这里指定了名字空间为android ，因此在接下来使用系统属性的时候，才可以使用android，来引用android系统的属性，同样的，如果要使用自定义属性，那么就需要创建自己的名字空间，第三方都需要使用如下代码：  
xmln:custom=”http://schemaas.androoi.com/spk/res-auto

这里我们将引入第三方控件的名字空间取名为custorm，之后在xml文件中使用使用自定义属性时，就可以通过这个名字空间来引用

shiyongzidingyideview与系统原生view最大的区别就是在声明控件时，需要指定完整的包名，在引用自定义属性时，需要使用自定义xmlns名字

一个新的自定义view 通常需要继承view 并重写它的ondraw onmeasure 等方法来实现绘制，同时通过onTOuchevent等触控事件来实现交互逻辑，。

自定义viewgroup

实现一个类似原生android scrollview的自定义viewgroup，自定义viewgroup可以实现scrillview所具有的上下滑动功能，但是在滑动过程中，增加一个粘性效果，即当一个子view向上滑动大于一定的距离后，松开手指，他将自动向上滑动，显示下一个view

当然，在viewgoup能够滚动前，需要先放置好他的view，使用便利的方式来通知子view对自身进行测量，

proteted void onmeasure(int widthmesureSpec,int heightmeasurespec)

{

super.onmeasure(widthmesurespec,heightmeasurespec);

int count=getchildcount()

for(int i=0;i<count:i++){  
view childview=getCildAt(i)

measureChild(childView,int widthMeasureSpec,heightMesureSPec);

}  
}

接下来就要对子view进行放置位置设定，让每个子view都显示完整的一屏，这样在滑动时候，可以比较好的实现后面的效果，在放置view前，需要确定整个viewgroup的高度。在本例中，由于让每个子view占一屏高度，viwegroup的高度即子view的个数乘以屏幕的高度

marginLayoutparams mlp=(marginLayoutparams)getLayoutParasm();

mlp.height=sScreenHeight\*childCount;

setLayoutParams(mlp)

在获取整个viewgroup 高度之后，就可以通过遍历来设置定每个子view需要放置的位置了，直接通过调用子view的layout方法，并将具体的位置作为参数传递进去即可

protected void onLayout(Boolean changed,int l,int t,int b){  
int childcount=getCildCount();

MarginLayoutParams mlp=(MarginLayoutParams)getLayoutParams();

Mlp.hieght=mScreenHeight\*childCount;

SetLayoutParams(mlp)

For(int i=0;i<childCoubt;i++)  
{

View child=getChildAt(i)

If(child.getVisibility()!=View.Gone){  
child.layout(l,i\*mScreenHieght,r,(i+1)\*mScreenHieght);

}  
}

}

case MotionEvent.Action\_down;

mLayut=u;

break;

case MotionEvent.Action\_move:

if(!mScroller.isFinishshed()){  
mScroller.isFinishied()}{  
mScroller.abortAnimation();  
}

}

int dy=mLayoutY-y;

if(getScrollY()<0){  
dy=0;

}

if(getScroY()>getHeight()-mScreenHeight){  
dy=0;

}

scrollBy(0,dy);

mLastY=y;

break;

事件拦截机制：

MotionEvent 里面封装了不少好东西，比如触摸点，可以通过event getX方法event egetRawX 方法取出坐标，可通过不同的Action来区分，