自定义控件以及事件分发

自定义控件包含：1.继承测量重绘控件， 2.事件分发 3.重绘控件

现实中的使用：

1. 继承控件：相同的容器控件，只是文字不一样
2. 事件分发：父控件写点击事件，switch控件优先接收点击事件（事件拦截，被消费），父控件点击事件无效，解决：重写switch的dispatchTouch（）方法为false，则事件传递到父控件TouchEvent（）方法中

点击百度地图，在修改位置同时，显示自定义按钮：创建同级View，监听dispatchTouchEvent()方法的执行。

## 1.自定义控件的分类（显示中的使用）

## 1.1**组合控件**

组合控件，顾名思义就是将一些小的控件组合起来形成一个新的控件，这些小的控件多是系统自带的控件。比如很多应用中普遍使用的标题栏控件，其实用的就是组合控件，那么下面将通过实现一个简单的标题栏自定义控件来说说组合控件的用法。

public class TitleView extends RelativeLayout {

// 返回按钮控件

private Button mLeftBtn;

// 标题Tv

private TextView mTitleTv;

……….

}

## **1.2自绘控件**

自绘控件的内容都是自己绘制出来的，在View的onDraw方法中完成绘制。

public class CounterView extends View implements OnClickListener {

@Override

protected void onDraw(Canvas canvas) {

………..

}

}

## **1.3继承控件（自定义属性）**

就是继承已有的控件，创建新控件，保留继承的父控件的特性，并且还可以引入新特性。

自定义属性：

1. 通常将自定义属性定义在/values/attr.xml文件中（attr.xml文件需要自己创建）。或者直接在style目录下写

转载至：https://www.cnblogs.com/itgungnir/p/6217447.html

<?xml version="1.0" encoding="utf-8"?>

<resources>

<attr name="rightPadding" format="dimension" />

<declare-styleable name="CustomMenu">

<attr name="rightPadding" />

</declare-styleable>

</resources>

1.Xml使用自定义属性

先定义一个namespace。Android中默认的namespace是android，因此我们通常可以使用“android:xxx”的格式去设置一个控件的某个属性，android这个namespace的定义是在XML文件的头标签中定义的。

android这个namespace的定义是在XML文件的头标签中定义的

xmlns:android="http://schemas.android.com/apk/res/android"

自定义的属性不在这个命名空间

xmlns:app="http://schemas.android.com/apk/res-auto"

1. 在java代码中获取并设置自定义控件

在自定义View / ViewGroup中，我们可以通过TypedArray获取到自定义的属性。示例代码如下：

public CustomMenu(Context context, AttributeSet attrs, int defStyleAttr) {

super(context, attrs, defStyleAttr);

TypedArray a = context.getTheme().obtainStyledAttributes(attrs, R.styleable.CustomMenu, defStyleAttr, 0);

int indexCount = a.getIndexCount();

for (int i = 0; i < indexCount; i++) {

int attr = a.getIndex(i);

switch (attr) {

case R.styleable.CustomMenu\_rightPadding:

mMenuRightPadding = a.getDimensionPixelSize(attr, 0);

break;

}

}

a.recycle();

}

或者

TypedArray typedArray = getContext().getTheme().obtainStyledAttributes(attrs, R.styleable.IvTvVertiaclView,defStyleAttr,0);  
String tvStr = typedArray.getString(R.styleable.IvTvVertiaclView\_tv);  
Drawable topDrawable = typedArray.getDrawable(R.styleable.IvTvVertiaclView\_iv);  
typedArray.recycle();

1. 构造参数的重写

ViewGroup中又四个构造方法可以重写，它们分别有一、二、三、四个参数。四个参数的构造方法我们通常用不到

public CustomMenu(Context context) {

this(context, null);

}

public CustomMenu(Context context, AttributeSet attrs) {

this(context, attrs, 0);

}

public CustomMenu(Context context, AttributeSet attrs, int defStyleAttr) {

super(context, attrs, defStyleAttr);

// 业务代码

}

最终都会到三个参数的构造方法中处理，减少了重复代码。

实际使用：项目中经常会把重复的控件，继承重写，设置布局，在style中添加一些属性，以后可以直接使用，大大节省工作时间。

# 自定义View详解

转载至：https://www.cnblogs.com/itgungnir/p/6217447.html

## 1.重要的参数

1.构造方法（如上）

2. onMeasure()方法

onMeasure()方法中主要负责测量，决定控件本身或其子控件所占的宽高。我们可以通过onMeasure()方法提供的参数widthMeasureSpec和heightMeasureSpec来分别获取控件宽度和高度的**测量模式**和**测量值**（测量 = 测量模式 + 测量值）。

**测量模式分为以下三种情况：**

1) EXACTLY：当宽高值设置为具体值时使用，如100DIP、match\_parent等，此时取出的size是精确的尺寸；

2) AT\_MOST：当宽高值设置为wrap\_content时使用，此时取出的size是控件最大可获得的空间；

3) UNSPECIFIED：当没有指定宽高值时使用（很少见）。

**onMeasure()方法中常用的方法：**

1) getChildCount()：获取子View的数量；

2) getChildAt(i)：获取第i个子控件；

3) subView.getLayoutParams().width/height：设置或获取子控件的宽或高；

4) measureChild(child, widthMeasureSpec, heightMeasureSpec)：测量子View的宽高；

5) child.getMeasuredHeight/width()：执行完measureChild()方法后就可以通过这种方式获取子View的宽高值；

6) getPaddingLeft/Right/Top/Bottom()：获取控件的四周内边距；

7) setMeasuredDimension(width, height)：重新设置控件的宽高。如果写了这句代码，就需要删除“super. onMeasure(widthMeasureSpec, heightMeasureSpec);”这行代码。

5、onDraw()

onDraw()方法负责绘制，即如果我们希望得到的效果在Android原生控件中没有现成的支持，那么我们就需要自己绘制我们的自定义控件的显示效果。

onDraw()方法中使用最多的两个类：Paint和Canvas。**常用的方法看博客**

6. onLayout()

onLayout()方法负责布局，大多数情况是在自定义ViewGroup中才会重写，主要用来确定子View在这个布局空间中的摆放位置。（子控件摆放的位置，一般viewGrop中才会腹泻）

onLayout(boolean changed, int l, int t, int r, int b)方法有5个参数，其中changed表示这个控件是否有了新的尺寸或位置；l、t、r、b分别表示这个View相对于父布局的左/上/右/下方的位置。

1) getChildCount()：获取子View的数量；

2) getChildAt(i)：获取第i个子View

3) getWidth/Height()：获取onMeasure()中返回的宽度和高度的测量值；

4) child.getLayoutParams()：获取到子View的LayoutParams对象；

5) child.getMeasuredWidth/Height()：获取onMeasure()方法中测量的子View的宽度和高度值；

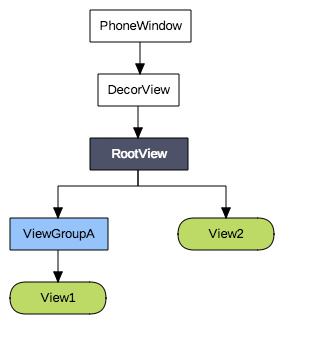
6) getPaddingLeft/Right/Top/Bottom()：获取控件的四周内边距；

7) child.layout(l, t, r, b)：设置子View布局的上下左右边的坐标。

# 事件分发

## 1.事件分发概念

为什么有事件分发：安卓上面的View是树形结构的，View可能会重叠在一起，当我们点击的地方有多个View都可以响应的时候，这个点击事件应该给谁呢？为了解决这一个问题，就有了事件分发机制。



window: 一个抽象类, 而 PhoneWindow 作为 Window 的唯一亲儿子(唯一实现类)，是所有视图的最顶层容器，视图的外观和行为都归他管，不论是背景显示，标题栏还是事件处理都是他管理的范畴

DecorView : 主题颜色和标题栏等内容就是显示在DecorView中的。为 PhoneWindow 服务的，除了自己要干活之外，也负责消息的传递，PhoneWindow 的指示通过 DecorView 传递给下面的 View，而下面 View 的信息也通过 DecorView 回传给 PhoneWindow。

## 2.事件分发、拦截与消费

下表省略了 PhoneWidow 和 DecorView。

√ 表示有该方法。

X 表示没有该方法。

| **类型** | **相关方法** | **Activity** | **ViewGroup** | **View** |
| --- | --- | --- | --- | --- |
| 事件分发 | dispatchTouchEvent | √ | √ | √ |
| 事件拦截 | onInterceptTouchEvent | X | √ | X |
| 事件消费 | onTouchEvent | √ | √ | √ |

## 3.事件分发流程

前面我们了解到了我们的View是树形结构的，基于这样的结构，我们的事件可以进行有序的分发。

事件收集之后最先传递给 Activity， 然后依次向下传递，大致如下：

Activity －> PhoneWindow －> DecorView －> ViewGroup －> ... －> View

如果没有任何View消费掉事件，那么这个事件会按照反方向回传，最终传回给Activity，如果最后 Activity 也没有处理，本次事件才会被抛弃:

Activity <－ PhoneWindow <－ DecorView <－ ViewGroup <－ ... <－ View

**非常经典的**[**责任链模式**](https://zh.wikipedia.org/wiki/%E8%B4%A3%E4%BB%BB%E9%93%BE%E6%A8%A1%E5%BC%8F)**吗，** 如果我能处理就拦截下来自己干，如果自己不能处理或者不确定就交给责任链中下一个对象。

**这种设计是非常精巧的，上层View既可以直接拦截该事件，自己处理，也可以先询问(分发给)子View，如果子View需要就交给子View处理，如果子View不需要还能继续交给上层View处理。既保证了事件的有序性，又非常的灵活。**

## 4.事件分发机制详解

件分发机制的原理：**责任链模式，事件层层传递，直到被消费。**

### 常见事件

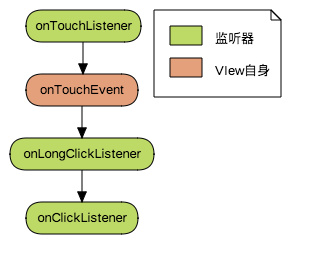
根据面向对象思想，事件被封装成 MotionEvent 对象。手指触摸相关的常见事件:

| **事件** | **简介** |
| --- | --- |
| ACTION\_DOWN | 手指 **初次接触到屏幕** 时触发。 |
| ACTION\_MOVE | 手指 **在屏幕上滑动** 时触发，会会多次触发。 |
| ACTION\_UP | 手指 **离开屏幕** 时触发。 |
| ACTION\_CANCEL | 事件 **被上层拦截** 时触发。（当手指一动到控件之外后，触发父控件拦截，所以出发了cancle） |

### 2.各种监听器的触发时间

* 单击事件(onClickListener) 需要两个两个事件(ACTION\_DOWN 和 ACTION\_UP )才能触发，如果先分配给onClick判断，等它判断完，用户手指已经离开屏幕，黄花菜都凉了，定然造成 View 无法响应其他事件，应该最后调用。(最后)
* 长按事件(onLongClickListener) 同理，也是需要长时间等待才能出结果，肯定不能排到前面，但因为不需要ACTION\_UP，应该排在 onClick 前面。(onLongClickListener > onClickListener)
* 触摸事件(onTouchListener) 如果用户注册了触摸事件，说明用户要自己处理触摸事件了，这个应该排在最前面。(最前)
* View自身处理(onTouchEvent) 提供了一种默认的处理方式，如果用户已经处理好了，也就不需要了，所以应该排在 onTouchListener 后面。(onTouchListener > onTouchEvent)

**所以事件的调度顺序应该是 onTouchListener > onTouchEvent > onLongClickListener > onClickListener**。



1. ViewGroup 中可能有多个 ChildView，如何判断应该分配给哪一个？

这个很容易，就是把所有的 ChildView 遍历一遍，如果手指触摸的点在 ChildView 区域内就分发给这个View。

2. 当该点的 ChildView 有重叠时应该如何分配？

当 ChildView 重叠时，**一般会分配给显示在最上面的 ChildView**。  
如何判断哪个是显示在最上面的呢？后面加载的一般会覆盖掉之前的，所以**显示在最上面的是最后加载的**。

-------子View放在链表中，最上层（最后加载的）会被放在链表的第一个，遍历的时候第一个遍历

**3. ViewGroup 和 ChildView 同时注册了事件监听器(onClick等)，哪个会执行?**

事件优先给 ChildView，会被 ChildView消费掉，ViewGroup 不会响应。

**4.View 中 onClick 事件需要同时接收到 ACTION\_DOWN 和 ACTION\_UP 才能触发，如果分配给了不同的 View，那么 onClick 将无法被正确触发**。

安卓为了保证所有的事件都是被一个 View 消费的，对第一次的事件( ACTION\_DOWN )进行了特殊判断，View 只有消费了 ACTION\_DOWN 事件，才能接收到后续的事件(可点击控件会默认消费所有事件)，并且会将后续所有事件传递过来，不会再传递给其他 View，除非上层 View 进行了拦截。如果上层 View 拦截了当前正在处理的事件，会收到一个 ACTION\_CANCEL，表示当前事件已经结束，后续事件不会再传递过来。