# 预览概要

1. 触摸事件的传递
2. 画一个可拖动的按钮
3. youku Menu
4. SlideMenu

# 1 触摸事件传递

## 1.1 触摸事件的传递的流程

触摸事件是一个自顶向下的过程，最早发送在activity，接着activity向下传递到第一个viewGroup,viewGroup将事件接着往下传递，一直传到除了需要处理这个事件的类。如果一直没有任何控件能够出来这个事件，事件就会往上传递，一直到activity。

- Activity中只有两个关于触摸事件的方法

  public class MainActivity extends Activity {

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}


/**
 *  分发事件
 *
 * super
 *  往下一级传递 如果是ViewGroup 则会在中间执行一个onInterceptTouchEvent方法
 * return true
 *  则该方法消费了当前事件 该事件到此终止
 * return false
 *  则告诉父控件自己不分发 由父控件的onTouchEvent处理
 * */
@Override
public boolean dispatchTouchEvent(MotionEvent ev) {
    Log.v("520it","MainActivity ---> dispatchTouchEvent"+ev.getAction());
    return super.dispatchTouchEvent(ev);
```

  // return false; }

```
  /**
   * 处理用户行为的方法可以是 按住 滑动 弹起
   * @param event 当前的动作事件
   * @return boolean
   *  false == super 不处理当前的事件
   *  true 处理当前的事件 一般该事件最早是ACTION_DOWN 如果为true则能获取下一个ACTION事
件
   *
   * */
  @Override
  public boolean onTouchEvent(MotionEvent event) {
      Log.v("520it","MainActivity ---> onTouchEvent");
      return super.onTouchEvent(event);
  }
```

}

- ViewGroup中多了一个拦截事件的方法

```
public class MySimpleViewGroup extends LinearLayout {

    public MySimpleViewGroup(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    public boolean dispatchTouchEvent(MotionEvent ev) {
        Log.v("520it","ViewGroup --->> dispatchTouchEvent");
        return super.dispatchTouchEvent(ev);
//      return false;
    }

    /**
     * 拦截事件的回调 该事件只有ViewGroup才有
     *  return true
     *      说明事件拦截成功 会直接走本类里面的onTouchEvent方法
     *  return false
     *  super
     *      说明不想拦截 会走下一层级的dispatchTouchEvent()
     * */
    @Override
    public boolean onInterceptTouchEvent(MotionEvent ev) {
        Log.v("520it","ViewGroup --->> onInterceptTouchEvent");
        return super.onInterceptTouchEvent(ev);
//      return false;
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        Log.v("520it","ViewGroup --->> onTouchEvent");
        return super.onTouchEvent(event);
    }

}
```

- View跟MainActivity一样

```
public class MySimpleView extends View {

    public MySimpleView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    public boolean dispatchTouchEvent(MotionEvent ev) {
        Log.v("520it","View --->>> dispatchTouchEvent");
        return super.dispatchTouchEvent(ev);
//      return false;
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        Log.v("520it","View --->>> onTouchEvent");
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                Log.v("520it", "ACTION_DOWN");
                break;
            case MotionEvent.ACTION_MOVE:
                Log.v("520it", "ACTION_MOVE");
                break;
            case MotionEvent.ACTION_UP:
                Log.v("520it", "ACTION_UP");
                break;
        }
//      return super.onTouchEvent(event);
        return true;
    }

}
```

# 1.2 触摸事件涉及到的方法：

dispatchTouchEvent:

- return true 当前的类消费完触摸事件，不像下级传递
- return fasle 当前的类不消费触摸事件，返回上级类处理
- super. dispatchTouchEvent 交由onInterceptTouchEvent或onTouchEvent处理

onInterceptTouchEvent:

- return true 当前的类拦截这个触摸事件，交由当前类的onTouchEvent处理
- return fasle 不拦截当前的触摸事件，交由子类的dispatchTouchEvent处理
- super 与 return false 同义

onTouchEvent:

- return true 当前类消费这个事件了，不像上传递了。
- return fasle 将当前事件返回上级onTouch，不接受以后的事件了
- super与 return false 同义

---

# 2 滑动按钮


1. 创建一个类SlideButton继承View
2. 实现带有两个参数的构造器（获取两个位图）

```
public SlideButton(Context context, AttributeSet attrs) {
    super(context, attrs);
    initPaint();
    initBmp();
}

private void initPaint() {
    mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
}

private void initBmp() {
    mBackgroundBmp = BitmapFactory.decodeResource(getResources(),
            R.drawable.switch_background);
    mSlideBmp = BitmapFactory.decodeResource(getResources(),
            R.drawable.slide_button_background);
}
```

3. 实现onMeasure()设置控件大小为底部位图的大小

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    setMeasuredDimension(mBackgroundBmp.getWidth(),
            mBackgroundBmp.getHeight());
}
```

4. 实现onDraw() 绘制默认的图片

```
@Override
protected void onDraw(Canvas canvas) {
    canvas.drawBitmap(mBackgroundBmp, 0, 0, mPaint);

    canvas.drawBitmap(mSlideBmp, 0, 0, mPaint);


}
```

5. 实现onTouchEvent() 滑动的时候控制滑块的位置 注意设置滑块的范围

```
private float mSlideLeft=0;

@Override
protected void onDraw(Canvas canvas) {
    canvas.drawBitmap(mBackgroundBmp, 0, 0, mPaint);

    if (mSlideLeft<0) {
        canvas.drawBitmap(mSlideBmp, 0, 0, mPaint);
    }else if (mSlideLeft>mBackgroundBmp.getWidth()-mSlideBmp.getWidth()) {
        canvas.drawBitmap(mSlideBmp, mBackgroundBmp.getWidth()-
mSlideBmp.getWidth(), 0, mPaint);
    }else {
        canvas.drawBitmap(mSlideBmp, mSlideLeft, 0, mPaint);
    }

}

@Override
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
    case MotionEvent.ACTION_DOWN:
        break;
    case MotionEvent.ACTION_MOVE:
        float touchX=event.getX();
        mSlideLeft=touchX-mSlideBmp.getWidth()/2;
        break;
    case MotionEvent.ACTION_UP:
        float slideHalf=mSlideLeft+mSlideBmp.getWidth()/2;
        float backgroundHalf=mBackgroundBmp.getWidth()/2;
        if (slideHalf>backgroundHalf) {
            mSlideLeft=mBackgroundBmp.getWidth()-mSlideBmp.getWidth();
        }else {
            mSlideLeft=0;
        }
        break;
    }
    //刷新界面 让系统调用onDraw()
    invalidate();
    return true;
}
```

6. 在onDraw()中根据最后的界面决定当前控件的状态

```
 @Override
 protected void onDraw(Canvas canvas) {

     ...

     //根据当前的状态决定开关
     if (mSlideLeft==0&&mIsOpen) {
         mIsOpen=false;
         ..
     }else if(mSlideLeft==mBackgroundBmp.getWidth()-
mSlideBmp.getWidth()&&!mIsOpen){
         mIsOpen=true;
         ..
     }
 }
```

7. 创建开关的回调接口

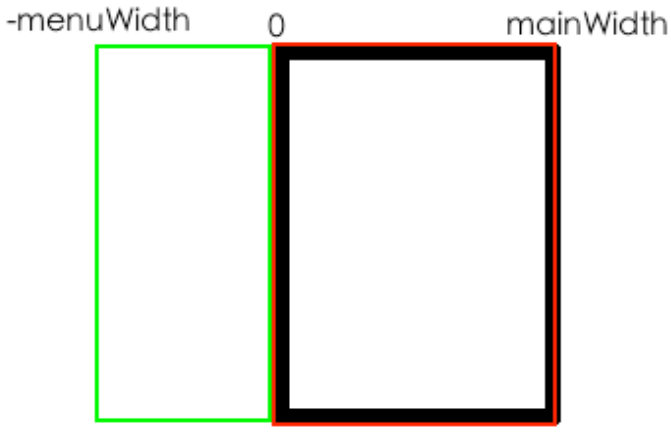
```
public interface ISlideButtonChangeListener{

    public void onButtonStateChanged(boolean flag);

}


public class SlideButton extends View {

    private ISlideButtonChangeListener mListener;

    public void setmListener(ISlideButtonChangeListener mListener) {
        this.mListener = mListener;
    }


    @Override
    protected void onDraw(Canvas canvas) {

        ...

        //根据当前的状态决定开关
        if (mSlideLeft==0&&mIsOpen) {
            mIsOpen=false;
            if (mListener!=null) {
                mListener.onButtonStateChanged(mIsOpen);
            }
        }else if(mSlideLeft==mBackgroundBmp.getWidth()-
mSlideBmp.getWidth()&&!mIsOpen){
            mIsOpen=true;
            if (mListener!=null) {
                mListener.onButtonStateChanged(mIsOpen);
            }
        }
    }
}
```

---

# 3 侧滑菜单

- 创建一个类继承ViewGroup,并实现构造器
- 实现onMeasure onLayout 方法

黑色 屏幕
绿色menu 红色main

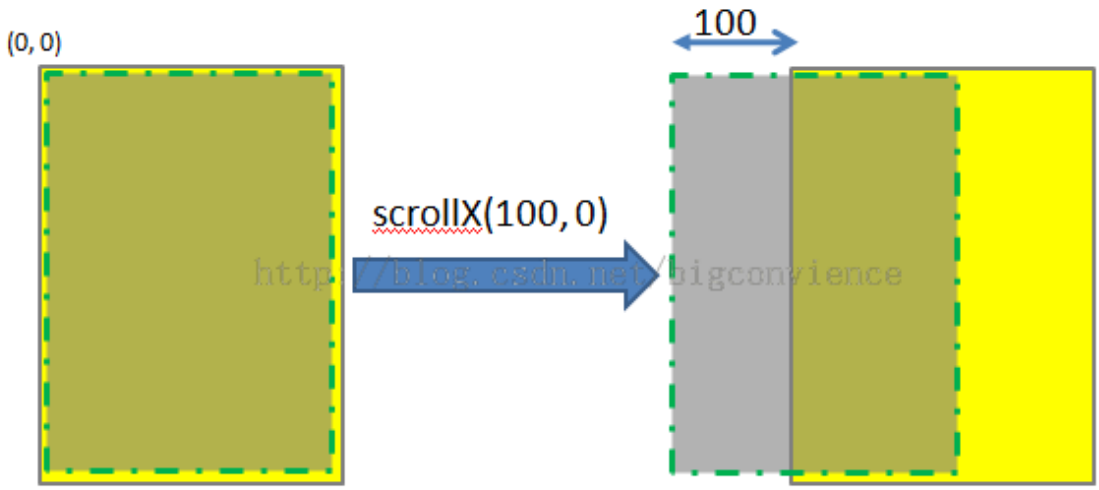



```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    measureChildren(widthMeasureSpec, heightMeasureSpec);
    mMenuView = getChildAt(0);
    mMainView = getChildAt(1);

    int widthSize = MeasureSpec.getSize(widthMeasureSpec);
    int heightSize = MeasureSpec.getSize(heightMeasureSpec);
    setMeasuredDimension(widthSize, heightSize);
}

@Override
protected void onLayout(boolean changed, int l, int t, int r, int b) {
    mMenuView.layout(-mMenuView.getMeasuredWidth(), 0, 0,
            mMenuView.getMeasuredHeight());
    mMainView.layout(0, 0, mMainView.getMeasuredWidth(),
            mMainView.getMeasuredHeight());
}
```

- 实现onTouchEvent() 并试验按下的scrollTo()/scrollBy()

我们在自定义控件的时候常常需要控件滚动，android提供了三个api来进行滚动控件的内容

1. ScrollTo(x,y) 快速滚动到(x,y) 注意方向平常的坐标系是相反的
2. ScrollBy(x,y) 移动x,y的偏移量
3. Scroller Scroller本身不能使View移动，要配合computeScroll才能实现滑动，原理也是不停让控件重绘




```
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
//              下面两个方法 偏移量都是坐标系的反向

//              跳到某个位置 多次点击还在该位置
//              scrollTo(-mMenuView.getWidth(), 0);
//              每次在原来的位置进行偏移
                scrollBy(100, 0);
                break;
            case MotionEvent.ACTION_MOVE:

                break;
            case MotionEvent.ACTION_UP:

                break;
        }
        return true;
    }
```

- 实现手指拖动时 界面跟着移动

```
//当前按下的那个点 可以认为是起始点
private float touchX;

@Override
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            touchX=event.getX();
            break;
        case MotionEvent.ACTION_MOVE:
            //移动某个瞬间的点 可以认为是瞬间的结束点
            float currenX=event.getX();
            //计算瞬时两个点的距离
            float deltaX=currenX-touchX;
            //因为scrollBy的坐标系的反的 所有这里应该为负
            scrollBy((int) -deltaX, 0);
            //下一瞬间的起点就是此刻的结束点
            touchX=currenX;
            break;
        case MotionEvent.ACTION_UP:

            break;
    }
    return true;
}
```

- 处理拖动越界的问题

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            touchX=event.getX();
            break;
        case MotionEvent.ACTION_MOVE:
            float currenX=event.getX();
            float deltaX=currenX-touchX;
            //这里可能出现越界的问题 可以先计算下 如果下一瞬间移动到位置越界就需要处理
            //1.计算将要移动的位置
            int destinationX=(int) (getScrollX()+(-deltaX));
            if (destinationX<-mMenuView.getWidth()) {//往左拖
                scrollTo(-mMenuView.getWidth(), 0);
            }else if (destinationX>0) {//往右拖
                scrollTo(0, 0);
            }else {
                scrollBy((int) -deltaX, 0);
            }
            touchX=currenX;
            break;
        case MotionEvent.ACTION_UP:
            break;
    }
    return true;
}
```

- 手指抬起的时候 如果没有滑动到位 需要计算到底要隐藏菜单还是显示菜单

```
case MotionEvent.ACTION_UP:
    //1.计算菜单的中心点位置
    float menuHalfe=-mMenuView.getWidth()/2;
    //2.如果菜单左边超过菜单的一半则显示 如果没超过则隐藏
    if (getScrollX()>menuHalfe) {
        scrollTo(0, 0);
    }else {
        scrollTo(-mMenuView.getWidth(), 0);
    }
    break;
```

- 上面的做法中 我们发现显示/隐藏动画没有动画效果

```
case MotionEvent.ACTION_UP:
        //1.计算菜单的中心点位置
        float menuHalfe=-mMenuView.getWidth()/2;
        //2.如果菜单左边超过菜单的一半则显示 如果没超过则隐藏
        if (getScrollX()>menuHalfe) {
        //  scrollTo(0, 0);
            mScroller.startScroll(getScrollX(), 0, 0-getScrollX(), 0, 400);
            invalidate();
        }else {
        //  scrollTo(-mMenuView.getWidth(), 0);
            mScroller.startScroll(getScrollX(), 0,
                    -mMenuView.getWidth()-getScrollX(), 0, 400);
            invalidate();
        }
        break;


@Override
public void computeScroll() {
    //computeScrollOffset()返回true说明还需要滚动
    if (mScroller.computeScrollOffset()) {
        //mScroller.getCurrX() 当前计算滚动到哪里
        scrollTo(mScroller.getCurrX(), 0);
        invalidate();
    }
}
```

- 如果是点击左边的列表左右滑动发现没有效果 先判断 如果发现是左右滑动 则拦截事件 不要列表再去处理了

```
float tabX;
@Override
public boolean onInterceptTouchEvent(MotionEvent ev) {
    switch (ev.getAction()) {
    case MotionEvent.ACTION_DOWN:
        tabX=ev.getX();
        break;
    case MotionEvent.ACTION_MOVE:
        //上下滑动说明需要拦截事件
        Log.v("520it", Math.abs(ev.getX()-tabX)+" velp");
        if (Math.abs(ev.getX()-tabX)>8) {
            return true;
        }
        tabX=ev.getX();
        break;
    case MotionEvent.ACTION_UP:
        break;
    }
    return super.onInterceptTouchEvent(ev);
}
```