

Weex源码分析系列（三）之Weex渲染流程分析

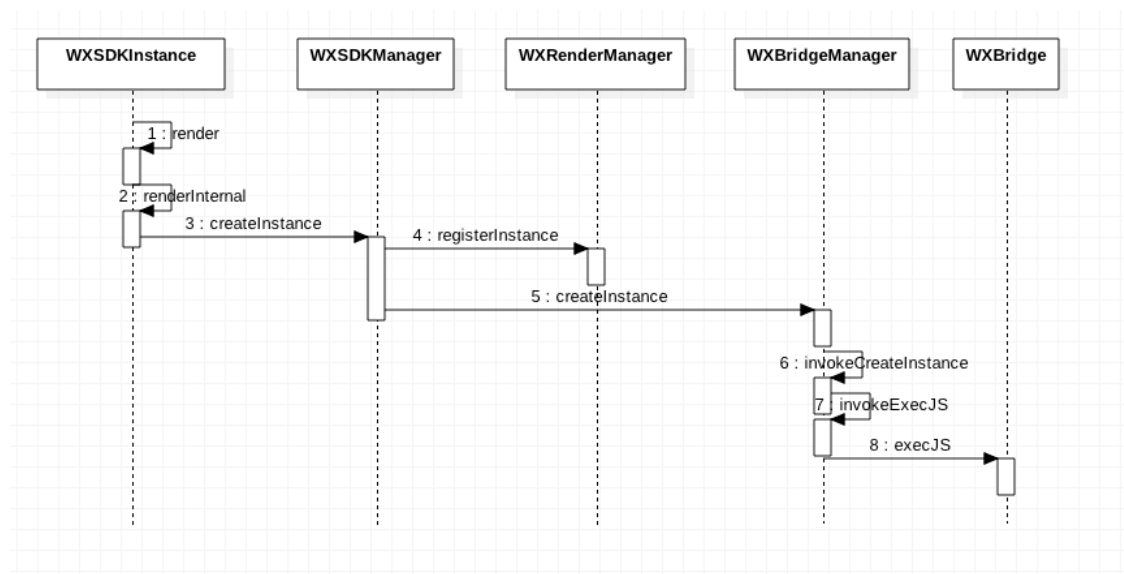
1、前言

在前两篇文章中我们结合源码学习了Module、Component的注册、调用、回调等流程，相信大家一定收获颇多，对Weex的理解也一定愈加深入。

那么本篇文章我们分析Weex的渲染流程，来看一看我们写的Js文件是如何在Native端变成Android里View的。

2、Weex渲染过程

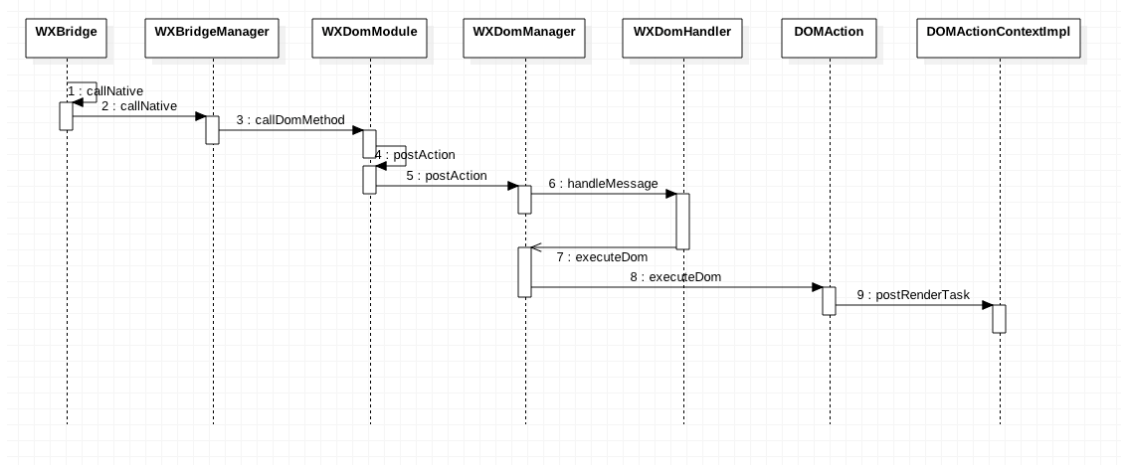
2.1 渲染触发点



在Activity中我们开个某个Weex页面使用的是WXSDKInstance中的render方法，最终也是按照常规套路通过WXBridge调用Js继续处理。

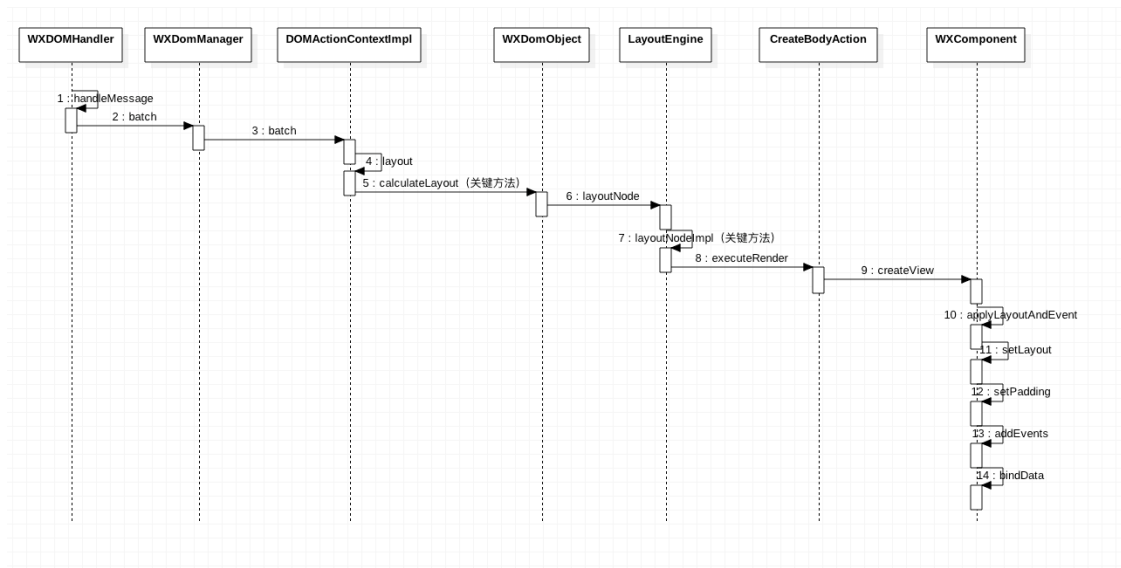
备注：Js引擎处理后回调Native这一部分要复杂的多，我们拆分成几步来看。

2.2 渲染准备



备注：这是渲染准备阶段，实际上和上一篇分析Component的调用准备是一样的；都是加一个任务保存到mNormalTasks。

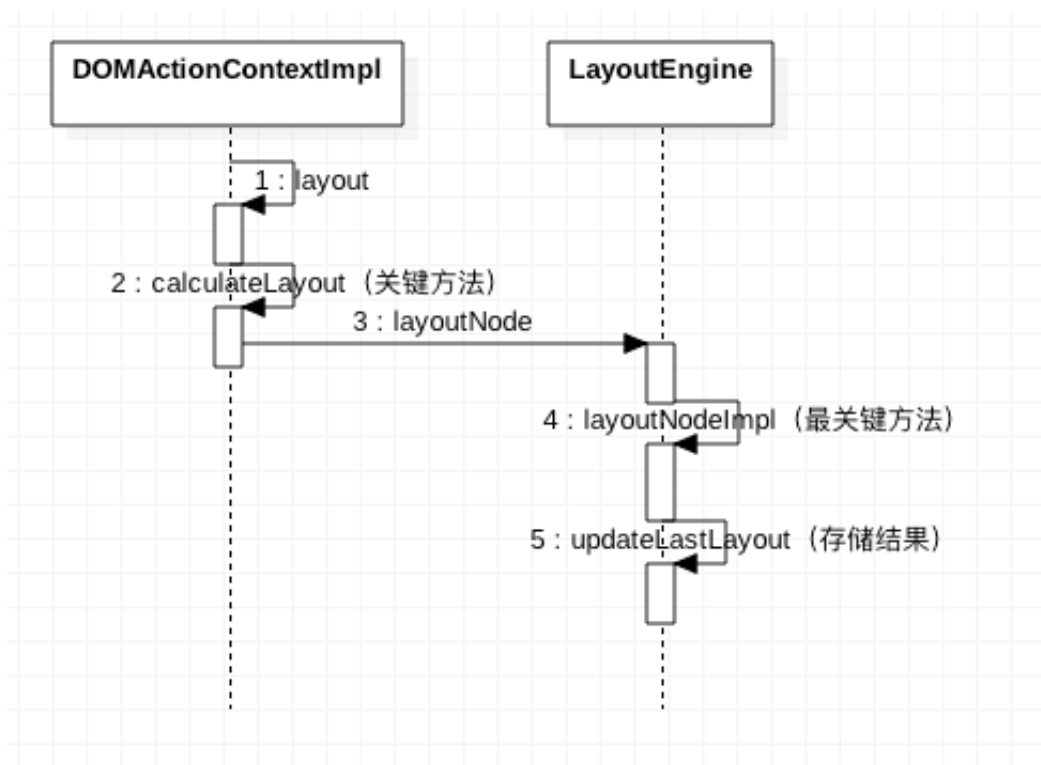
2.2 渲染流程分析



从上我们可以看出真实的渲染过程由WXDomHandler发起，关键方法在DOMActionContextImpl和WXComponent中（标出关键方法的地方）；也经历了测量、布局、绘制、处理事件、设置数据等流程。

渲染流程非常重要，里面有很多关键步骤，下面我们——分析；

2.3 calculateLayout



calculateLayout分析：

- DOMActionContextImpl.calculateLayout()开始，执行到 LayoutEngine.layoutNodeImpl()，这步是真实的解析、保存Js中储存的布局；layoutNodeImpl()方法长达720行，解析FlexBox布局，并且保存结果到CSSLayout中。

2.4 createView

```

/**
 * create view
 */
public final void createView() {
    mHost = initComponentsHostView(mContext);
    if (mHost == null && !isVirtualComponent()) {
        //compatible
        initView();
    }
    if(mHost != null){
        mHost.setId(WXViewUtils.generateViewId());
        ComponentObserver observer;
        if ((observer = getInstance().getComponentObserver()) != null)
    {

```

```

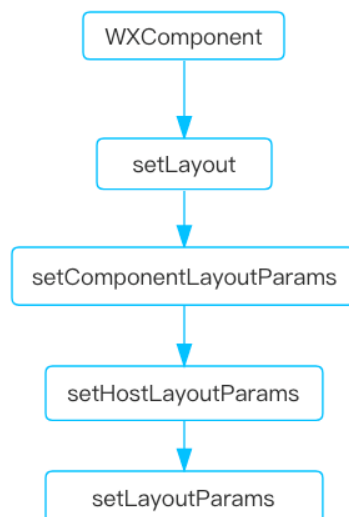
        observer.onViewCreated(this, mHost);
    }
}
onHostViewInitialized(mHost);
}

```

createView分析：

- **创建目标Component对象，跟进去可以看到initComponentHostView方法，就是我们自定义Component必须重载的方法；**

2.5 setLayout



```

/**
 * Layout view
 */
public final void setLayout(ImmutableDomObject domObject) {

    .....
    Spacing parentPadding = (nullParent?new
Spacing():mParent.getDomObject().getPadding());
    Spacing parentBorder = (nullParent?new
Spacing():mParent.getDomObject().getBorder());
    Spacing margin = mDomObj.getMargin();
    int realWidth = (int) mDomObj.getLayoutWidth();

```

```

        int realHeight = (int) mDomObj.getLayoutHeight();
        int realLeft = (int) (mDomObj.getLayoutX() -
parentPadding.get(Spacing.LEFT) -
                                parentBorder.get(Spacing.LEFT));
        int realTop = (int) (mDomObj.getLayoutY() -
parentPadding.get(Spacing.TOP) -
                                parentBorder.get(Spacing.TOP)) +
siblingOffset;
        int realRight = (int) margin.get(Spacing.RIGHT);
        int realBottom = (int) margin.get(Spacing.BOTTOM);
        .....
        mAbsoluteY = (int) (nullParent?0:mParent.getAbsoluteY() +
mDomObj.getLayoutY());
        mAbsoluteX = (int) (nullParent?0:mParent.getAbsoluteX() +
mDomObj.getLayoutX());
        .....
        setComponentLayoutParams(realWidth, realHeight, realLeft, realTop,
realRight, realBottom, rawOffset);
        .....
    }

```

setLayout分析：

- **setLayout()获取真实的宽高及四个顶点的位置，类比原生Android中的Measure与Layout过程；**
- **setComponentLayoutParams()中转换原生识别的LayoutParams，并且会调用setLayoutParams()，我们知道这个方法会调用走原生View的Measure、Layout、Draw等流程；**

2.5 addEvents

```

public void addEvent(String type) {

    .....
    View view = getRealView();
    if (type.equals(Constants.Event.CLICK) && view != null) {
        addClickListener(mClickListener);
    } else if ((type.equals(Constants.Event.FOCUS) ||
type.equals(Constants.Event.BLUR))) {
        addFocusChangeListener(new WXComponent.OnFocusChangeListener()
        {

            public void onFocusChange(boolean hasFocus) {
                Map<String, Object> params = new HashMap<>();

```

```

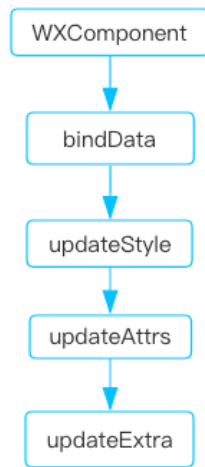
        params.put("timeStamp", System.currentTimeMillis());
        fireEvent(hasFocus ? Constants.Event.FOCUS :
Constants.Event.BLUR, params);
    }
});
} else if (view != null &&
    needGestureDetector(type)) {
    if (view instanceof WXGestureObservable) {
        if (mGesture == null) {
            mGesture = new WXGesture(this, mContext);
            boolean isPreventMove =
WXUtils.getBoolean(getDomObject().getAttrs().get(Constants.Name.PREVENT_MO
VE_EVENT), false);
            mGesture.setPreventMoveEvent(isPreventMove);
        }
        mGestureType.add(type);
        ((WXGestureObservable)
view).registerGestureListener(mGesture);
    } else {
        WXLogUtils.e(view.getClass().getSimpleName() + " don't
implement " +
            "WXGestureObservable, so no gesture is
supported.");
    }
} else {
    Scrollable scroller = getParentScroller();
    if (type.equals(Constants.Event.APPEAR) && scroller != null) {
        scroller.bindAppearEvent(this);
    }
    if (type.equals(Constants.Event.DISAPPEAR) && scroller !=
null) {
        scroller.bindDisappearEvent(this);
    }
}
}
}

```

addEvents分析：

- addEvents()添加View的事件处理；

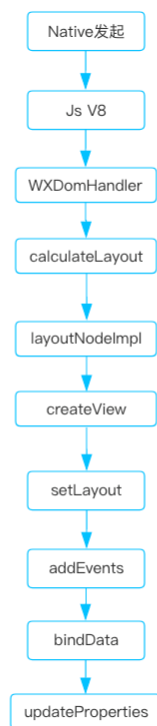
2.6 bindData



bindData :

- 更新Style、绑定数据等；
- 具体都会执行到updateProperties()方法中，其中实现是MethodInvoker反射调用方法；

3、渲染流程图



总结：

- Weex渲染流程由Native发起，通过JsBridge传给V8引擎，处理后回传指令到Native；
- Dom相关的操作使用WXDomHandler切换到Dom线程操作；
- layoutNodeImpl是核心测量过程解析FlexBox布局，计算Dom的位置信息并存储；
- 接下来WXRenderHandler将后续工作线程切换到RenderThread也就是UI线程；
- 由Component创建具体的View；
- setLayout实际上是将位置信息转换为原生View识别的params；
- addEvents添加事件；
- bindData设置style及赋值；

4、对比

下面我们对Weex的渲染和Android的渲染流程进行一下对比：

- 对于Android原生的渲染需要经过Measure、Layout、Draw等步骤；

- 对于Weex来说，Android原生的渲染流程是全有的而且只是一部分，因为我们虽然写的是Js代码但是实际显示的确是Native控件；
- 那么Weex比原生多的流程就是：与V8的交互、关于Dom的解析与生成、设置属性与赋值（扩展）等；

5、总结

- Weex渲染流程的分析难度比Module、Component等组件难度要大的多，毕竟Module等只是一个组件而这时一个完整的流程；
- Weex渲染流程的分析依赖于Module、Component等组件的实现，这也是我首先分析这两个组件的原因；
- 在Weex渲染流程的分析中我们第一次接触到了Weex中的线程切换，之后会细说；
- 建议大家都实际跟踪下Weex的源码，里面有很多可以学习的细节；

欢迎持续关注Weex源码分析项目：[Weex-Analysis-Project](#)

欢迎关注微信公众号：定期分享Java、Android干货！

