Weex源码分析系列（二）之Component组件源码解析

1、前言

在上一篇文章《Weex源码分析系列（一）之Module组件源码解析》中我们结合源码学习了Module的注册、调用、回调等流程，并且分析一个Weex自带Module的实现。

那么本篇文章我们开始分析Weex的另一个重要组件Component，关于Component的注册、调用等分析本文为你娓娓道来。

2、初始Component

2.1 Component的定位

在《Android 扩展》中我们可以看到Component的定位：

> Component 扩展 实现特别功能的Native控件。例如：RichTextview，RefreshListview 等。

大家知道Android的四大组件中用户唯一有感知的就是Activity，而在Weex的这些组件中用户唯一有感知的也就是Component。实际上它就是Weex里的Widget，比如我们在WeexList中开发的Js代码中写的那些控件，最终在Native都是一个个的Component。

2.2 Component的使用

2.2.1 内置Component

对于普通的界面开发，我们一般不会见到Component的踪迹，因为Weex已经提供了一套基础的Component组件与基础Html标签的对应，例如基础Component组件的注册：

```
    registerComponent(
        new SimpleComponentHolder(
```

```
            WXImage.class,
            new WXImage.Ceator()
        ),false,WXBasicComponentType.IMAGE,WXBasicComponentType.IMG
    );
    registerComponent(WXBasicComponentType.CELL, WXCell.class, true);
    registerComponent(WXBasicComponentType.INDICATOR, WXIndicator.class,
true);
    registerComponent(WXBasicComponentType.VIDEO, WXVideo.class, false);
    registerComponent(WXBasicComponentType.INPUT, WXInput.class, false);
    registerComponent(WXBasicComponentType.TEXTAREA,
Textarea.class,false);
    registerComponent(WXBasicComponentType.SWITCH, WXSwitch.class, false);
```

2.2.2 自定义Component

项目中总有些效果是内置的Component无法实现的，**此时我们就要像自定义控件一样自定义Component，具体实例可以参考WeexList中的CircleImageView与RefreshView两个自定义Component。**下面主要说下注意事项：

- **自定义Component需要提供的方法上加上注解WXComponentProp，并加上name，作为Js端调用的方法名；**

```
    @WXComponentProp(name = "setSrc")
    public void setImage(String url) {

    }
```

- **Weex初始化的时候注册这个Component；**

```
    WXSDKEngine.registerComponent("circleImageView",
CircleImageView.class);
```

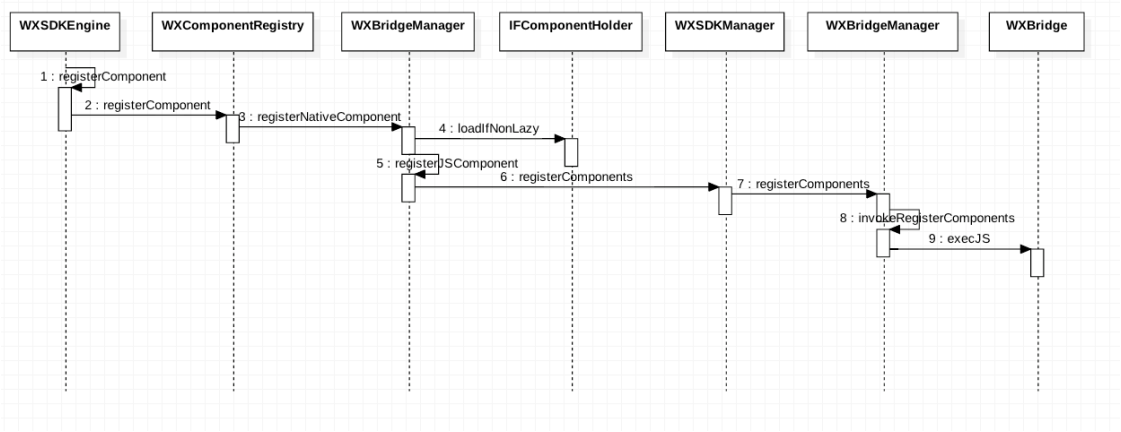
- **Js端的使用；**

```
    <circleImageView :setSrc="item.bphoto" style="width:100;height:100">
</circleImageView>
```

注意这个 :setSrc 就是上面注解上的name；

3、Component源码分析

3.1 Component注册

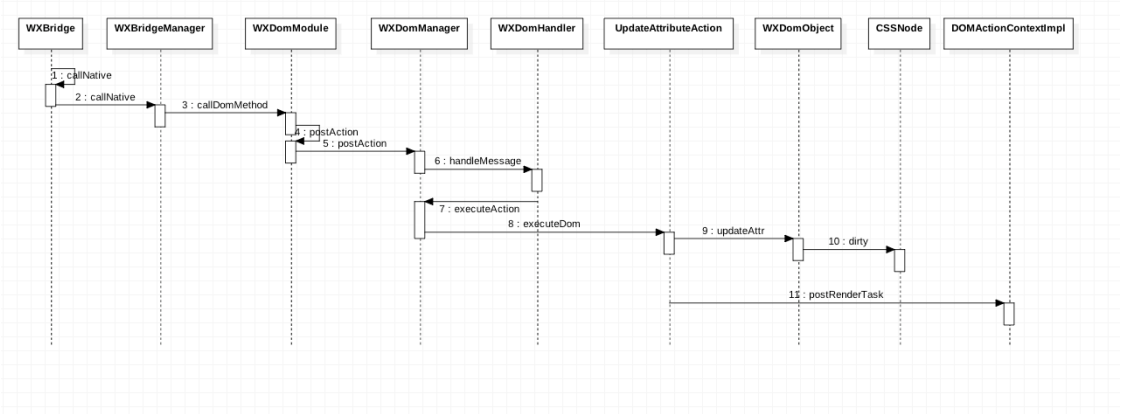


Component的注册和Module很像，大家从时序图上可以看出有几个很熟悉的类；Component同样分为本地注册与Module注册；

- 本地注册
    - **同样由WXSDKEngine发起，需要注意IFComponentHolder中的loadIfNonLazy，会判断Component是不是lazy（为提升初始化效率），不是lazy的话则直接解析这个Component，获取其Method信息保存；不是Lazy的话则在第一次使用的时候解析、保存；可以类比Module是不是global；**
    - **实际上还是保存了本地Component与Js端的一个对应；**
- Js注册
    - **Js注册更和Module的Js注册没有区别，同样最终也是通过WXBridge执行的与Js交互；**

3.2 Component调用

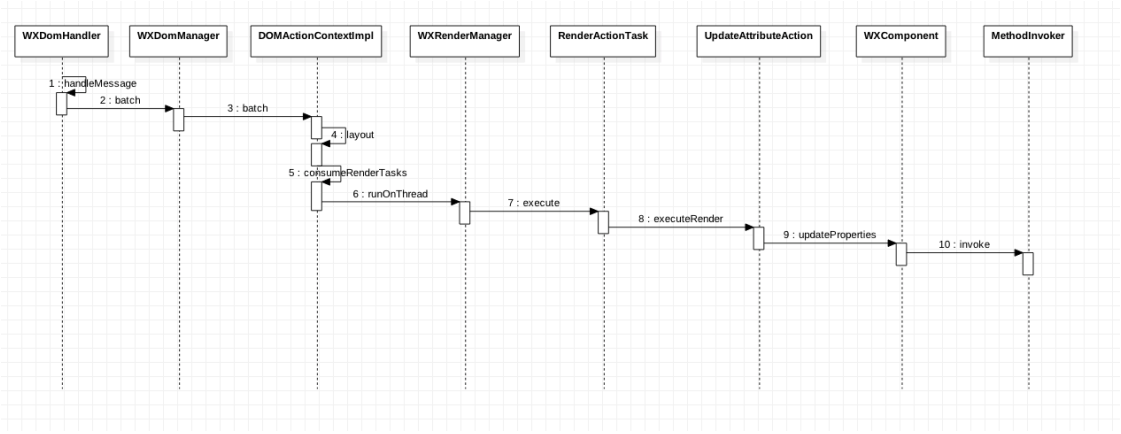
以自定义Component的方法调用为例，Component的调用相对比较复杂，我们拆分成两步来看，调用准备和调用执行；



调用准备说明：

- **照例Js的调用由WXBridge开始；**
- **将Js调用的任务封装成一个DOMAction；**
- **然后由WXDomHandler将任务切换到Dom线程也就是主线程执行；**
- **接下来的步骤就是在DOMActionContextImpl的任务注册了；**

```
    @Override
    public void postRenderTask(RenderAction action) {
        mNormalTasks.add(new RenderActionTask(action,
mWXRenderManager.getRenderContext(mInstanceId)));
        mDirty = true;
    }
```



调用执行说明：

- **执行由WXDomHandler发起，触发绘制；**
- **然后到了DOMActionContextImpl中的layout方法————》其中有一个**

consumeRenderTasks方法，从名字我们就可以看出是消费RenderTask的（备注：此处对应了保存任务的mNormalTasks）；

- 然后RenderActionTask调用executeRender方法，开始触发执行；
- 最终是在WXComponent中包装成MethodInvoker，反射调用执行；

3.3 Component源码盘点

通过上述对Component注册、调用等的源码分析，我们可以看到Component相比较Module还是比较复杂的。如果大家仔细跟Component源码的话会发现一个问题：第一步调用准备和第二步调用执行是如何串起来的？

在第一步调用准备结束之后，只是将任务加到了mNormalTasks保存，并没有任何执行任务的代码，那第二步调用执行是如何被调用的？ 这个问题也困扰了我若干分钟。

下面说说我对这块的探索：Weex的绘制逻辑和Android原生很类似，Android会每隔16毫秒发出一次VSYNC信号触发对UI进行渲染，而Weex也会每隔16毫秒发出一个消息触发绘制，具体的逻辑在WXDomHandler中的WX_DOM_BATCH类型消息中；

```
public class WXDomHandler implements Handler.Callback {

  @Override
  public boolean handleMessage(Message msg) {
        if (!mHasBatch) {
            mHasBatch = true;

mWXDomManager.sendEmptyMessageDelayed(WXDomHandler.MsgType.WX_DOM_BATCH,
DELAY_TIME);
        }
        case MsgType.WX_DOM_BATCH:
            mWXDomManager.batch();
            mHasBatch = false;
            break;
  }
}
```

从mWXDomManager.batch()开始后续的逻辑就串起来了，DOMActionContextImpl中的consumeRenderTasks方法对RenderTask进行消费也就是第一步调用准备Task的执行；

4、内置组件

Weex对常用控件都进行了封装具体在com.taobao.weex.ui.component下可以找到，**一些文档上没写的属性之类的可以在源码中查找，毕竟源码面前，了无秘密。**

下面我们简单看一个常用的控件：列表控件，Weex里内置了WXListComponent来支持列表控件；

```
@Component(lazyload = false)
public class WXListComponent extends
BasicListComponent<BounceRecyclerView> {
    ......

    @WXComponentProp(name = Constants.Name.COLUMN_GAP)
    public void setColumnGap(float columnGap) throws InterruptedException
{
        if(mRecyclerDom != null && mRecyclerDom.getColumnGap() !=
mColumnGap) {
            markComponentUsable();
            updateRecyclerAttr();
            WXRecyclerView wxRecyclerView = getHostView().getInnerView();
            wxRecyclerView.initView(getContext(), mLayoutType,
mColumnCount, mColumnGap, getOrientation());
        }
    }

    ......
}
```

可以看出：

- **WXListComponent基于RecycleView，这样View的复用是有保障的；**
- **lazyload为false，也就是注册即解析（重量级组件优先级就是高）；**
- **提供了类如setColumnGap来设置列间隔等方法；**

对于我们自定义的Component都可以参照这个思路，具体实例可以参考WeexList。

5、Component总结

- **Component的定位是实现特别功能的Native控件；**
- **Component的源码实现相较于Module更加复杂，尤其是从调用准备到调用**

执行这个断点的链接；

- **调用准备是包装成RenderTask，然后加到集合保存；**
- **调用执行是RenderTask的执行过程，最终是方法的反射调用；**
- **两步之间的断点链接是由Weex的渲染机制保证的；**

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！