Weex源码分析系列（一）之Module组件源码解析

1、前言

经过前面两篇文章的实践，我们学习了Weex的使用。本篇开始我们深入Weex的源码，一起探索Weex在安卓平台上是如何构建一套JS的运行框架，那从Module开始说起吧。

本文从源码入手分析Module的注册、调用、回调等流程，并且分析一个Weex自带Module的实现。

2、初识Module

2.1 Module的定位

在《Android 扩展》中我们可以看到Module的定位：

> Module 扩展非UI的特定功能，例如 sendHttp、openURL 等。

也就是说是**非UI类型的功能提供，在本地注册供Js运行时调用，有了Module，我们就可以自己扩展一些Weex没有提供的能力给Js，让Js更加强大！**

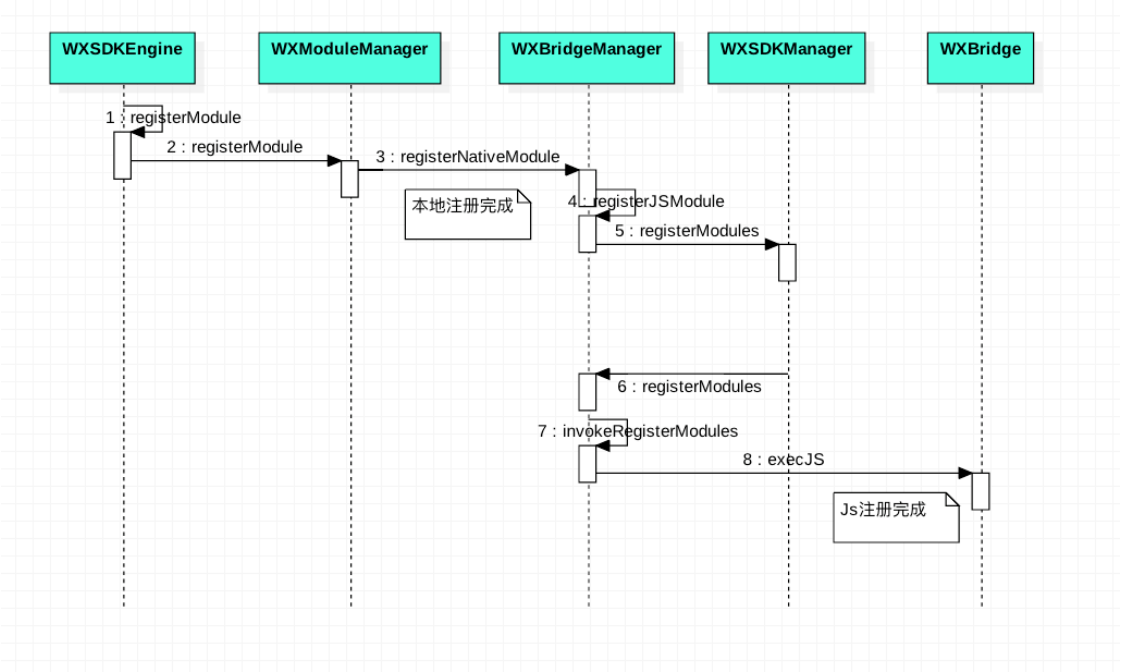
2.2 Module的使用

还记不记得在上两篇文章中我们就用到了Module来实现网络请求的能力，声明+调用即完成了网络请求的调用；

```
var stream = weex.requireModule('stream') // 声明
stream.fetch // 请求
```

3、Module源码分析

3.1 Module注册

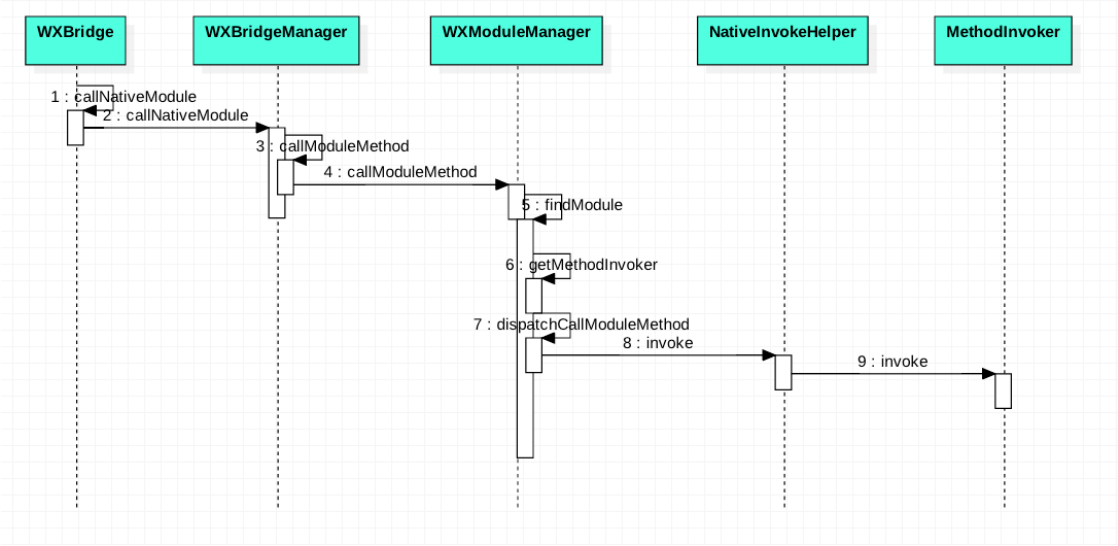


对于注册，也分为**本地注册和Js注册**：

- 本地注册：
  - **如果这个Module是全局的，则直接反射创建Module对象，然后存储在sGlobalModuleMap中；**
  - **Native注册Module的过程很简单，在sModuleFactoryMap中保存了Module的Name及对应的ModuleFactory；**
- Js注册：
  - **WXBridge是Native与Js交互的桥梁，执行的是JNI层的Java_com_taobao_weex_bridge_WXBridge_execJS方法；**
  - **Js的注册最终调用到了WXBridge的execJS方法，类型为METHOD_REGISTER_MODULES；**

备注：注册的意义在于Js与Native定义了一个协议，一个对应关系；调用的时候能找到Module。

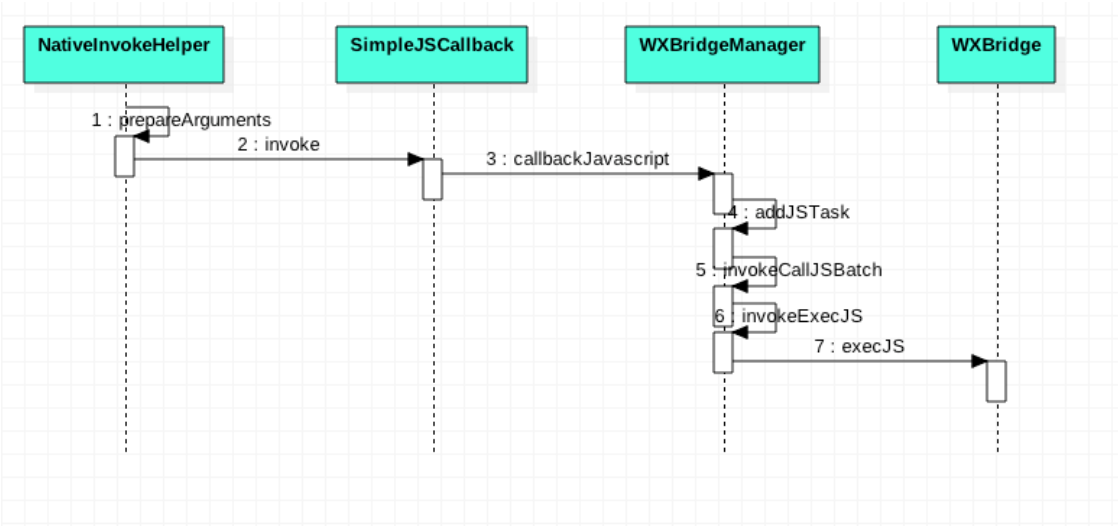
3.2 Module调用



调用分析：

- **Module的调用是由Native开始，通过WXBridge这个Js与Native交互的桥梁；**
- **在WXModuleManager会调用findModule，上面本地注册的时候有提到（是全局的话则直接反射创建），此处未被创建的Module对象会被创建；**
- **调用getMethodInvoker方法获取调用方法包装成的对象Invoker；**
- **通过NativeInvokerHelper来调用，实际是是MethodInvoker反射调用的；**

3.3 Native回调Js



调用分析：

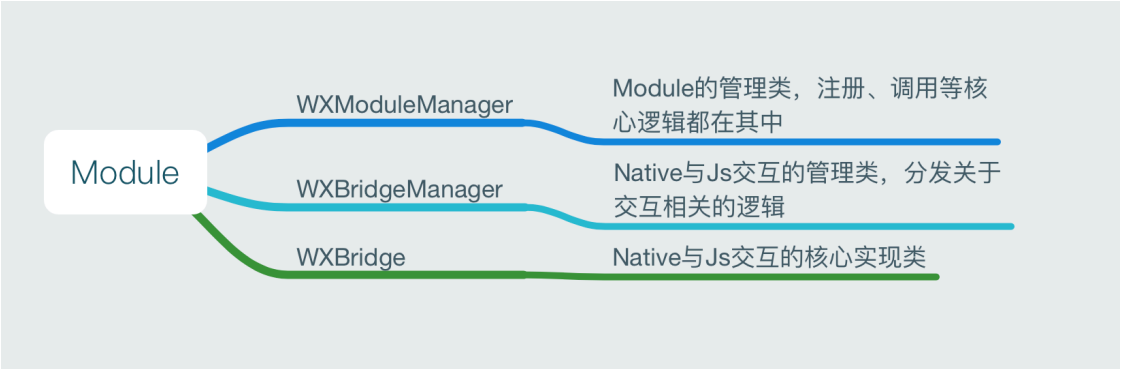
- **NativeInvokerHelper反射调用方法的时候回将JSCallback包装成SimpleJSCallback对象；**
- **Native调用完成需要回调的时候调用到WXBridgeManager的callbackJavascript方法；**
- **随后发送消息，任务执行就被切换到了WeexJSBridgeThread线程；**
- **最后也是调用的WXBridge回调Js端；**

3.4 Module源码总结

Module的源码并不复杂：

- **在客户端定义了Module来实现特定能力例如网络请求（就是一个提供方法的类）；**
- **然后注册到Native和Js（注册过程就是让双方能找到对方）；**
- **调用是从Js发起，通过WXBridge（交互桥梁，以后还会经常遇到）与Native交互的；**
- **Native找到本地注册的Class，通过反射调用Module方法；**
- **回调也是通过WXBridge，传递信息给Js；**

3.5 Module相关的关键类



4、特定Module实现分析

分析了Module的注册、调用、回调等步骤，我们就来实例分析一个Weex中自带的Module：WXStreamModule，我们网络请求的功能就是它实现的，在Weex代码中我们调用网络请求的是fetch方法，**那么我们在WXStreamModule方法中寻找fetch方法：**

```
  @JSMethod(uiThread = false)
  public void fetch(String optionsStr, final JSCallback callback,
JSCallback progressCallback){
            ......

            String method = optionsObj.getString("method");
            String url = optionsObj.getString("url");
            JSONObject headers = optionsObj.getJSONObject("headers");
            String body = optionsObj.getString("body");
            String type = optionsObj.getString("type");
            int timeout = optionsObj.getIntValue("timeout");

            sendRequest()

            ......
  }
```

- **通过上述optionsObj中获取各种数据的方式，我们可以推知在Weex代码中怎么去声明想要的参数，例如：Weex的Demo中没有写传递header的例子，我们查看获取Header的方式也知道需要将其封装成一个JsonObject，从根源来发现解决问题的途径是看源码的好处之一，源码面前了无秘密！**

- **sendRequest是通过IWXHttpAdapter实现的，没有设置的话会使用DefaultWXHttpAdapter；然后在一个FixedThreadPool里通过HttpURLConnection执行的网络请求。**

```
public class DefaultWXHttpAdapter implements IWXHttpAdapter {
    private ExecutorService mExecutorService =
Executors.newFixedThreadPool(3);

    @Override
    public void sendRequest(final WXRequest request, final OnHttpListener
listener) {
        HttpURLConnection connection = openConnection(request, listener);
    }

}
```

备注：通过细看WXStreamModule的源码我们不难发现最终网络请求的默认执行类DefaultWXHttpAdapter有一定缺陷，固定线程数的线程池实际上并不适合网络请求的场景，尤其在网络请求密集的场景下（整个模块、甚至应用都使用

Weex来做，这种场景不会罕见）。因此如果我们要大面积使用Weex最好自己实现IWXHttpAdapter或者调用Native的方法。这是看源码的另一好处：从源码角度分析框架实现的利弊，定制更适合自己业务的实现。

自定义Module比较简单，就不在本文细说了，可以参照官方文档或者自带的Module的实现，也可以参考我写的WeexList，里面有自定义Module的使用。

5、Module总结

- **Module的定位是扩展非UI的特定功能；**
- **Module源码的分析，包括：注册、调用、回调等；**
- **分析一个自带的Module：WXStreamModule；**
- **通过本文也可以发现源码阅读的好处：**
  - 从根源来发现解决问题的途径（Header的传递方式）；
  - 源码角度分析框架实现的利弊，定制适合自己的实现（DefaultWXHttpAdapter里的问题，自己定制）；

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！