小程序运行机制调研

文章中部分内容引用自：http://blog.csdn.net/xiangzhihong8/article/details/66521459#comments

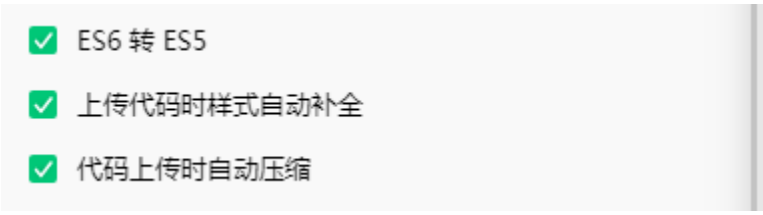
概述：小程序开发类似于react、vue等前端框架，虽然使用了wxml，wxss等文件命名，但本质上还是运行在web平台上的，而且最终打包上传的代码也是js，html，css。最终生成的代码在微信中的运行的原理如官网所说：视图层目前使用 WebView 作为渲染载体，而逻辑层是由独立的 JavascriptCore 作为运行环境，数据的沟通是通过两边提供的evaluateJavascript所实现。

1. 打包

小程序通过开发者工具打包上传代码，在打包的过程中需要解析wxml，wxss。打包的过程是通过nwjs + react实现的。

1.1 nwjs

nwjs是一款轻量级桌面应用开发工具，通过nodejs + webkit实现，这样就可以使用前端语言来开发桌面应用，nodejs提供了基础的API，webkit实现视图层面的逻辑，可以理解为一款加强型的“浏览器“，在这个“浏览器”上可以运行nodejs代码。正式因为有nodejs参与，开发工具的打包功能原理就很好理解：



- ES6语法转换通过babel实现
- 样式自动补全通过autoprefixer实现
- 自动压缩通过uglifyjs实现

而在开发者工具的安装包里，也可以找到 node_module文件夹，里面包含了nodejs用到的工具包。

nwjs的文档：https://github.com/nwjs/nw.js

1.2 react

小程序具体在运行、打包时怎么跟react思想统一不可知，但是单向数据绑定、手动触发视图更新、组件化等思想都证明了小程序采用了react的设计思想，最有力的证据是，在小程序开发者工具的安装包中的node_module中有react，redux等相关的包。

在细节上，小程序根据自身的特点进行单独处理，如：

```
this.setData({
 'object.text': 'changed data'
});
```

react中类似的方法 setState 不允许更新一个对象参数的一部分属性，setState({ 'object.text' : 'test' }) 只会设置一个新的属性‘object.text’，而小程序之所以这么处理，是由于架构限制

1.3 打包后的结构

所有的小程序基本都最后都被打成上面的结构
1、WAService.js 框架JS库，提供逻辑层基础的API能力
2、WAWebview.js 框架JS库，提供视图层基础的API能力
3、WAConsole.js 框架JS库，控制台
4、app-config.js 小程序完整的配置，包含我们通过app.json里的所有配置，综合了默认配置型
5、app-service.js 我们自己的JS代码，全部打包到这个文件
6、page-frame.html 小程序视图的模板文件，所有的页面都使用此加载渲染，且所有的WXML都拆解为JS实现打包到这里
7、pages 所有的页面，这个不是我们之前的wxml文件了，主要是处理WXSS转换，使用js插入到header区域。

小程序启动时会从CDN下载小程序的完整包

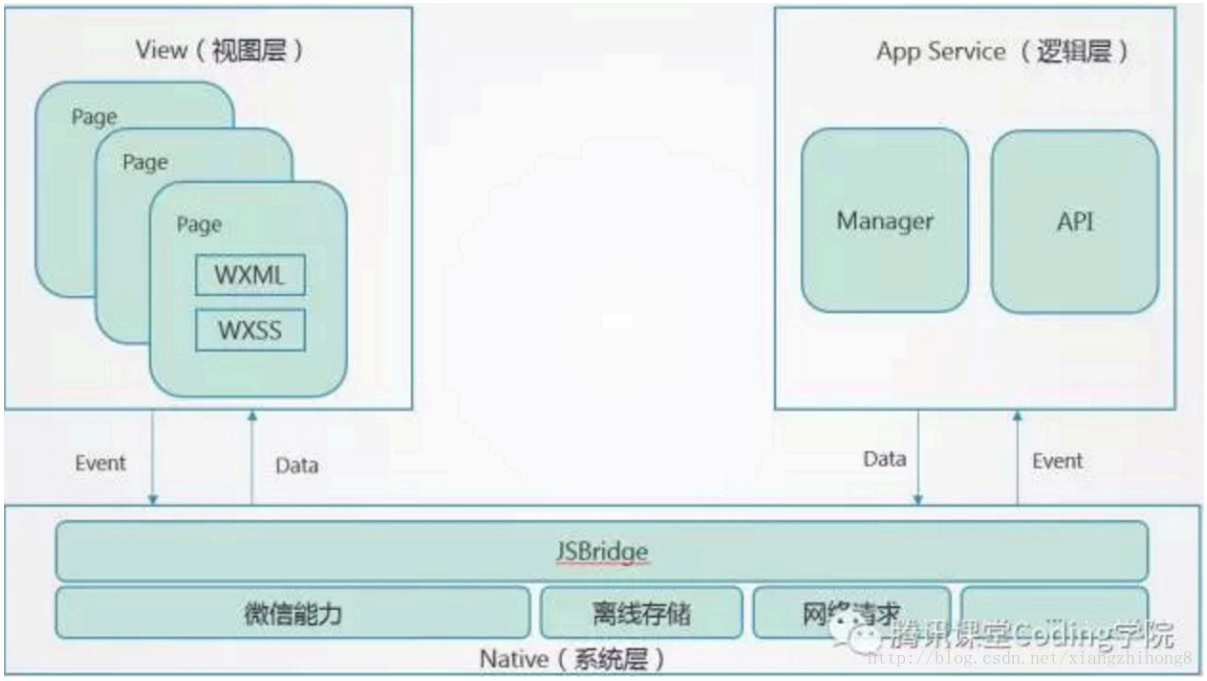
2. 架构

根据官网：

小程序的视图层目前使用 WebView 作为渲染载体，而逻辑层是由独立的 JavascriptCore 作为运行环境。在架构上，WebView 和 JavascriptCore 都是独立的模块，并不具备数据直接共享的通道。当前，视图层和逻辑层的数据传输，实际上通过两边提供的evaluateJavascript所实现。即用户传输的数据，需要将其转换为字符串形式传递，同时把转换后的数据内容拼接成一份 JS 脚本，再通过执行 JS 脚本的形式传递到两边独立环境。

而evaluateJavascript的执行会受很多方面的影响，数据到达视图层并不是实时的。

根据描述可以得到下图：



总结一下：逻辑模块与视图模块互不影响，通过jsbridge连接沟通

上文说到的react，从这里可以判断是运行在视图层

而在通信的过程中，“传输的数据，需要将其转换为字符串形式传递“，就会引起诸多问题：

- setData
  回调不可用，上面说到小程序采用的是react思想，手动触发视图更新，而在react中，setState也是异步的，但react层面的setState只经历的更新数据，没有经过字符串转换（字符串经过处理后eval执行），而小程序多了一层字符串转换后，如果还是直接采用react的异步模式就是出现回调不准的问题。 ps：这点只是个人猜测，并没有验证
- setData的性能问题

3. 实现技术

小程序在启动时，会将处理逻辑的js代码加载到一个webview中，称为appService，在另一个webview中处理视图，称为appView。appService常驻内存，所以小程序会至少打开 2 个webview，所有的通信动作都是在不同的webview之间进行。

appService

可以理解AppService即一个简单的页面，主要功能是负责逻辑处理部分的执行，底层提供一个WAService.js的文件来提供各种api接口，主要是以下几个部分：
消息通信封装为WeixinJSBridge（开发环境为window.postMessage,
IOS下为WKWebview的window.webkit.messageHandlers.invokeHandler.postMessage，android下用WeixinJSCore.invokeHandler）

1、日志组件Reporter封装
2、wx对象下面的api方法
3、全局的App,Page,getApp,getCurrentPages等全局方法
4、还有就是对AMD模块规范的实现

然后整个页面就是加载一堆JS文件，包括小程序配置config，上面的WAService.js（调试模式下有asdebug.js），剩下就是我们自己写的全部的js文件，一次性都加载。

开发环境：

1、页面模板：app.nw/app/dist/weapp/tpl/appserviceTpl.js
2、配置信息，是直接写入一个js变量，__wxConfig。
3，其他配置

线上环境：

在上线后是应用部分会打包为2个文件，名称app-config.json和app-service.js，然后微信会打开webview去加载。
1、WAService.js（底层支持）
2、app-config.json（应用配置）
3、app-service.js（应用逻辑）

开发环境与线上环境都运行在JavaScriptCore引擎里面。

appView

这里可以理解为h5的页面，提供UI渲染，底层提供一个WAWebview.js来提供底层的功能,具体如下：
1、消息通信封装为WeixinJSBridge（开发环境为window.postMessage,
IOS下为WKWebview的window.webkit.messageHandlers.invokeHandler.postMessage，android下用WeixinJSCore.invokeHandler）
2、日志组件Reporter封装
3、wx对象下的api，这里的api跟WAService里的还不太一样，有几个跟那边功能差不多，但是大部分都是处理UI显示相关的方法
4、小程序组件实现和注册
5、VirtualDOM，Diff和Render UI实现
6、页面事件触发

在此基础上，AppView有一个html模板文件，通过这个模板文件加载具体的页面，这个模板主要就一个方法，$gwx，主要是返回指定page的VirtualDOM，而在打包的时候，会事先把所有页面的WXML转换为ViirtualDOM放到模板文件里，而微信自己写了2个工具wcc（把WXML转换为VirtualDOM）和wcsc（把WXSS转换为一个JS字符串的形式通过style标签append到header里）。

appView中的小程序原生组件：

在WAWebview.js里有个对象叫exparser，它完整的实现小程序里的组件，我们使用的所有组件，都会被提前注册好，在Webview里渲染的时候进行替换组装。
exparser有个核心方法：
regiisterBehavior：注册组件的一些基础行为，供组件继承
registerElement：注册组件，跟我们交互接口主要是属性和事件

Service和View通信

使用消息publish和subscribe机制实现两个Webview之间的通信，实现方式就是统一封装一个WeixinJSBridge对象，而不同的环境封装的接口不一样，具体实现的技术如下：

windows环境

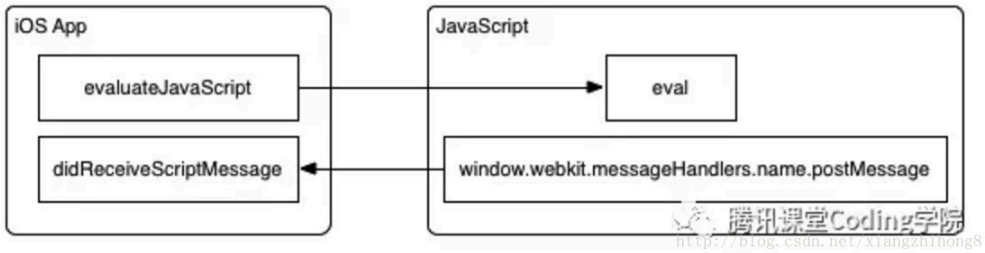
通过window.postMessage实现（使用chrome扩展的接口注入一个contentScript.js，它封装了postMessage方法，实现webview之间的通信，并且也它通过chrome.runtime.connect方式，也提供了直接操作chrome native原生方法的接口）
发送消息：window.postMessage(data, ‘*’);, // data里指定 webviewID
接收消息：window.addEventListener(‘message’, messageHandler); // 消息处理并分发，同样支持调用nwjs的原生能力。
在contentScript里面看到一句话，证实了appservice也是通过一个webview实现的，实现原理上跟view一样，只是处理的业务逻辑不一样。

```
'webframe' === b ? postMessageToWebPage(a) : 'appservice' === b && postMessageToWebPage(a)
```

- 1

IOS

通过 WKWebview的window.webkit.messageHandlers.NAME.postMessage实现微信navite代码里实现了两个handler消息处理器：
invokeHandler: 调用原生能力
publishHandler: 消息分发



Android

通过WeixinJSCore.invokeHanlder实现，这个WeixinJSCore是微信提供给JS调用的接口（native实现）
invokeHandler: 调用原生能力
publishHandler: 消息分发

4. 总结

- 小程序本质上与我们平时混合开发一样，相当于微信放开了自己的webview，同时提供了一套接和规范来统一开发人员。
- 逻辑层与视图层独立，各自以webview为依托执行
- 在逻辑层采用类react的模式，视图层用react渲染
- 现阶段小程序的性能限制也是混合开发的限制