Weex 初探

Haoero 2017/01/16

一. Weex是什么？

A framework for building Mobile cross-platform UI.

Weex 是一款轻量级的移动端跨平台动态性技术解决方案，前身是WeApp, 它借鉴了React Native的思想，并结合Web Component和Vue.js而诞生的。

- 目标：Write Once, Run Everywhere. 官方支持iOS、Android、HTML5.
- JS引擎：Weex使用V8， ReactNative使用JSCore
- 特点：轻量，可扩展，高性能
- 与RN对比：参见

二. Weex能做什么事？

2.1 做到了

- 致力于移动端
- 能够充分调度 native 的能力
- 能够充分解决或回避性能瓶颈
- 能够灵活扩展
- 能够多端统一
- 能够优雅“降级”到 HTML5
- 能够保持较低的开发成本
- 能够快速迭代
- 能够轻量实时发布
- 能够融入现有的 native 技术体系
- 能够工程化管理和监控

2.2 谁在使用Weex

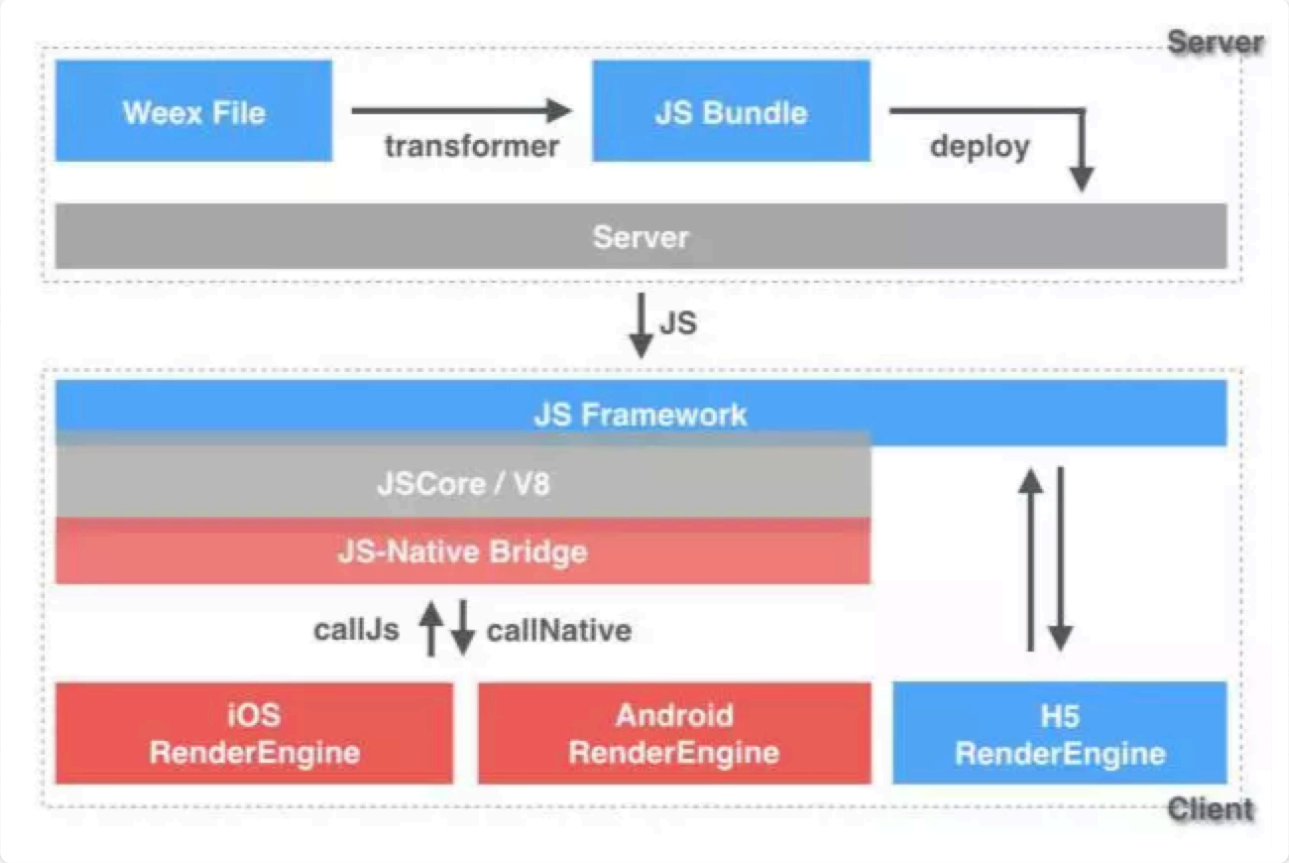
阿里系的大部分APP都接入Weex了，但是应该还没有全面替换原生页面。

三. Weex不能做什么？缺点呢？

- 开源社区不够活跃(开源组件少、遇坑自己填等);
- Native界面层级很多，影响一定的性能;
- 应该还有很多，但是需要实践深入些才能发现更多;

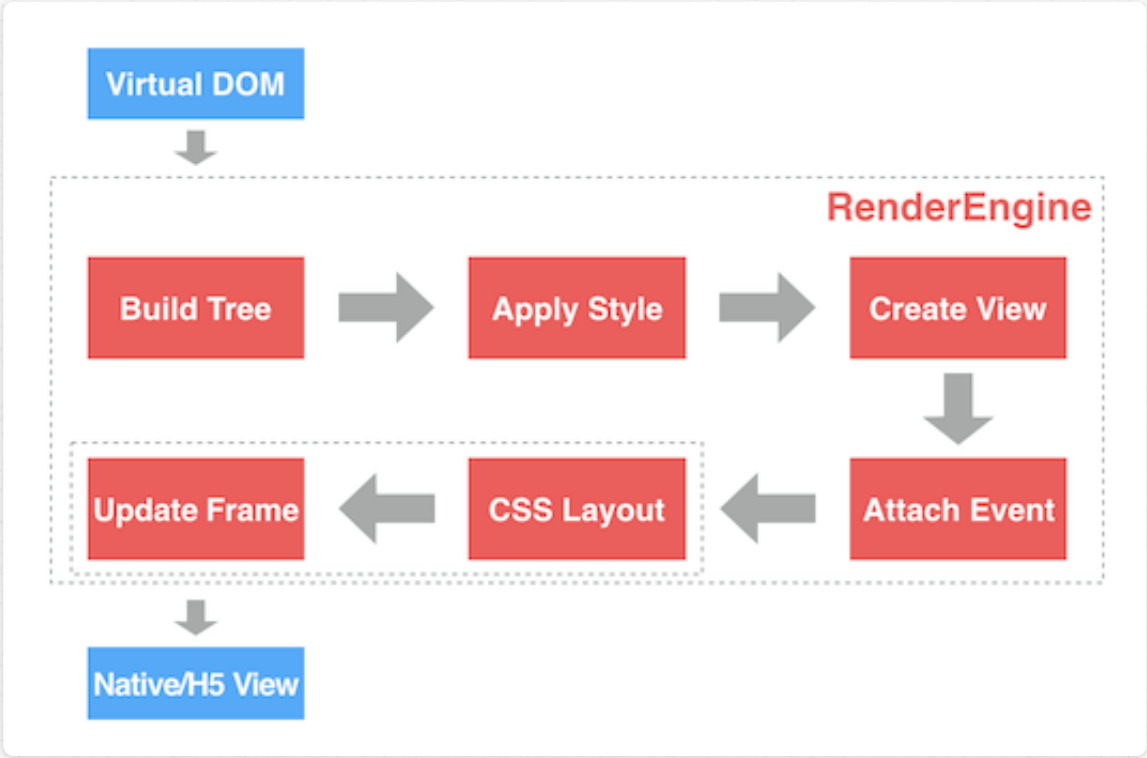
四. Weex工作原理

4.1 基本原理：



- 通过Transformer将DSL文件(.we文件)转换成JS Bundle;
- 将这些JS Bundle部署到分发服务器，并下发到各客户端;
- 客户端运行JS引擎解析JS Bundle，并通过JS-Native Bridge让JS引擎和Native代码可以相互通信;
- Native的渲染引擎将解析出来的View渲染成Native的控件

4.2 视图渲染流程

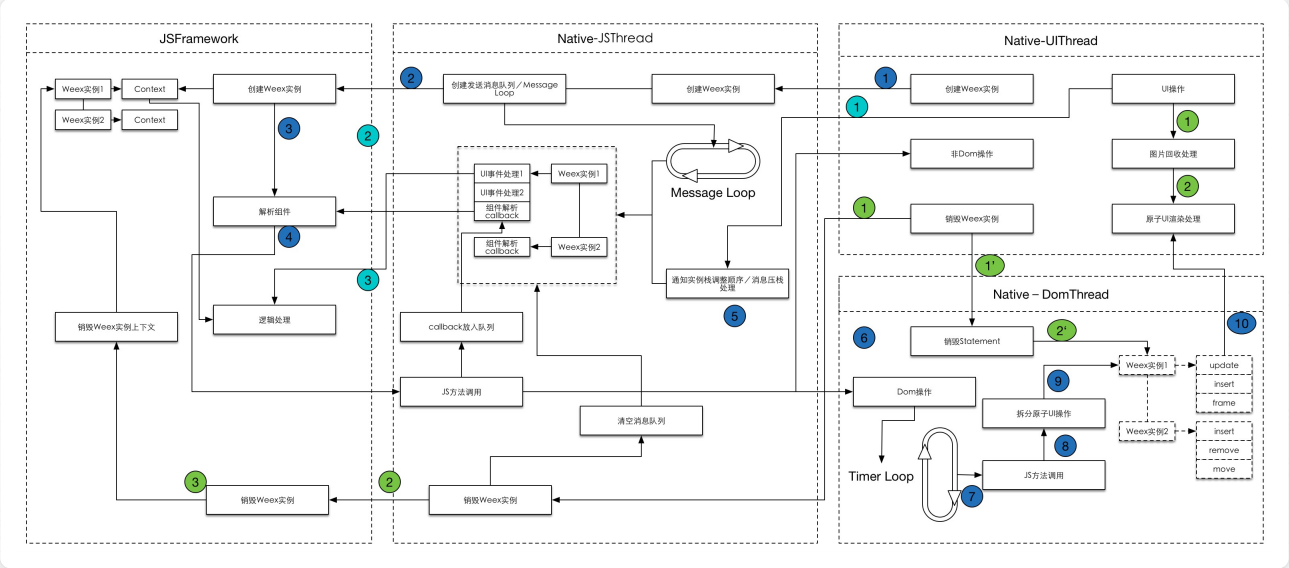


- 输入JS Engine生成的虚拟DOM.
- 构造树结构: 分析虚拟DOM JSON数据以构造渲染树(RT).
- 添加样式: 为渲染树的各个节点添加样式.
- 创建视图: 为渲染树各个节点创建Native视图.
- 绑定事件: 为Native视图绑定事件.
- CSS布局: 使用 css-layout 来计算各个视图的布局.
- 更新视窗(Frame): 采用上一步的计算结果来更新视窗中各个视图的最终布局位置.
- 最终页面呈现.

4.3 异步渲染机制

基本原理：采用流式渲染和离屏渲染

- 流式渲染: JS Framework每创建一个节点，立即发送给native，同时等待native的next tick命令的触发才能继续执行，这样就有效的解决了页面回退仍然在渲染，事件不能响应的问题。
- 离屏渲染: 用户在浏览页面的过程中，后台Dom线程继续渲染更新页面，将所有addDom、updateSytle等操作都以最小颗粒度拆分，保证所有的操作都在16ms内完成，大大提升首屏的加载性能以及滑动的流畅度。



[摘自Weex]

五. Weex开发

5.1 基本知识

5.1.1 环境配置

- Node.js: 通过Homebrew安装

```
brew install node
#验证是否安装成功
$ node -v
v6.3.1
$ npm -v
3.10.3
```

- weex-toolkit: 这个是weex的集成环境，包含了各种weex工具包和集成命令

```
$ npm install -g weex-toolkit
```

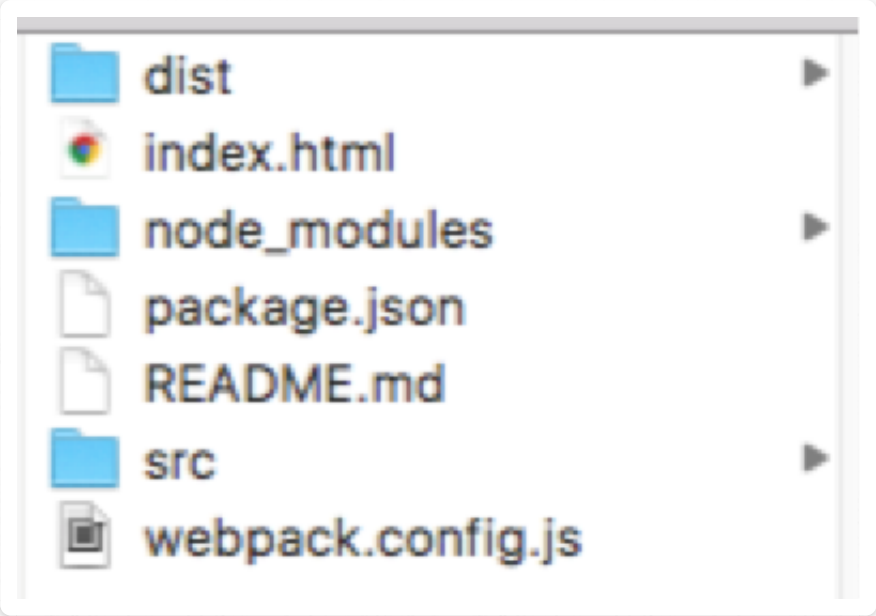
- 编辑器: 推荐WebStorm, 可以搜索下载安装Weex插件，就支持.we文件高亮和语法提示等功能了

5.1.2 初始化Weex项目

```
#命令行创建初始项目
```

```
$ weex init niuniu_weex
#集成npm环境
$ npm install
```

5.1.3 目录结构说明



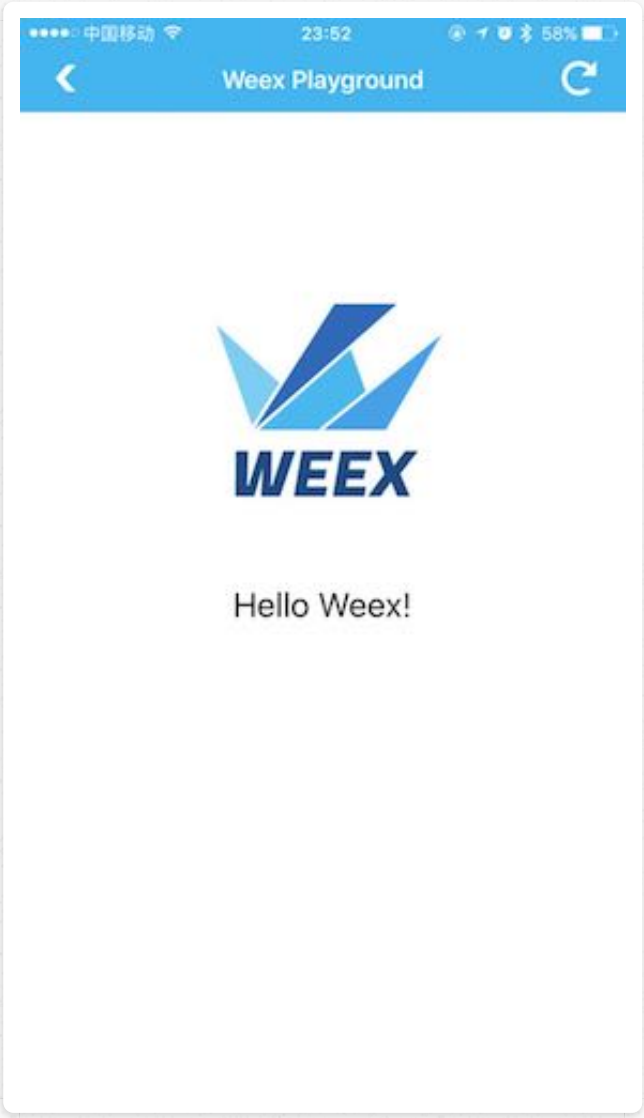
- dist: 生成的js文件存放目录
- src: 代码目录，存放所有.we的源文件
- src/weex-bootstrap.we 就是当前工程的初始页面
- webpack.config.js: webpack的配置文件
- package.json: 配置webpack的依赖库
- 启动: npm run serve，在浏览器里打开localhost:8080/index.html即可看到weex h5页面，即src/weex-bootstrap.we描述的页面

5.1.4 在Playground APP里看效果

- 下载阿里提供的Weex Playground, 里面列出了目前官方支持的所有基础控件，也可以扫描调试页面二维码，直接看效果，支持hot loading
- 一般来说，真正需要开发的就是src目录下的.we文件，如果需要看某个文件在手机端的效果，可以使用如下命令行来生成二维码:

```
weex weex-bootstrap.we --qr
```

- 手机与Mac连接同一个局域网后，用Weex Playground扫描二维码即可看到实时的效果
- 如果此时修改该文件后，比如调小字号之类的，保存后就可以立即看到变化;



5.2 iOS Demo实践

实现富途牛牛APP里，我的tab界面里功能入口列表，并可以动态切换成一行四个格子的样式, 如下图

5.2.1 集成Weex SDK到现有iOS工程

5.2.1.1 使用cocoapods管理,导入最新的Weex SDK，并重新pod install

```
pod 'WeexSDK', '0.9.4'
#也可以将官方DEMO工程下载回来后，在本地导入SDK
#pod 'WeexSDK', :path => '../ios_weex/sdk/'
```

5.2.1.2 初始化SDK环境，一般都是在Appdelegate里初始化

```
//初始化
[WXSDKEngine initSDKEnviroment];
//设置当前APP属性(可选)
[WXAppConfiguration setAppGroup:@"FUTU"];
[WXAppConfiguration setAppName:@"FTPhoneNiuNiu"];
[WXAppConfiguration setExternalUserAgent:@"ExternalUA"];
//设置日志级别
[WXLog setLogLevel:WXLogLevelDebug];
```

5.2.1.3 渲染weex Instance: 支持整体页面渲染和部分渲染两种模式，这一步本质是新建一个Weex的view, 这样Native就可以将其加到任意的容器上了，可以是整个VC也可以是一个subview

```
    @interface FTWeexMineFunctionListController : FTBaseViewController

    @property (nonatomic, strong) WXSDKInstance *weexInstance;

    @property (nonatomic, strong) NSURL *weexURL;

    @property (nonatomic, weak) UIView *weexView;

    @end
    @implementation FTWeexMineFunctionListController

    - (void)viewDidLoad
    {
        [super viewDidLoad];
        self.title = @"FUTU Weex";
        [self renderWeexView];
    }

    - (void)renderWeexView
    {
        [_weexInstance destroyInstance];
        //初始化weex instance
        _weexInstance = [[WXSDKInstance alloc] init];
        _weexInstance.viewController = self;
        _weexInstance.frame = self.view.frame;

        //加载本地js文件
        [_weexInstance renderWithURL:self.weexURL options:@{@"bundleUrl
":[self.weexURL absoluteString]} data:nil];
        //为了避免循环引用声明一个弱指针的`self`
        __weak typeof(self) weakSelf = self;
        //设置weexInstance创建完毕回调
        _weexInstance.onCreate = ^(UIView *view) {
            weakSelf.weexView = view;
            weakSelf.weexView.backgroundColor = [UIColor clearColor];
            [weakSelf.weexView removeFromSuperview];
            [weakSelf.view addSubview:weakSelf.weexView];
        };
        // 设置weexInstance出错的回调
        _weexInstance.onFailed = ^(NSError *error) {
        //process failure
        NSLog(@"处理失败:%@",error);
        };
        //设置渲染完成的回调
        _weexInstance.renderFinish = ^ (UIView *view) {
            //process renderFinish
```

```
            NSLog(@"渲染完成");
        };
    }

    //当前的JS文件的本地URL，该文件是test.we文件经过命令行生成的 $ weex mylist.
we -o
    - (NSURL *)weexURL
    {
        if (!_weexURL) {
            NSString *filePath = [NSString stringWithFormat:@"%@/bundle
_js/test.js",[NSBundle mainBundle].bundlePath];
            _weexURL = [NSURL fileURLWithPath:filePath];
        }
        return _weexURL;
    }
    @end
```

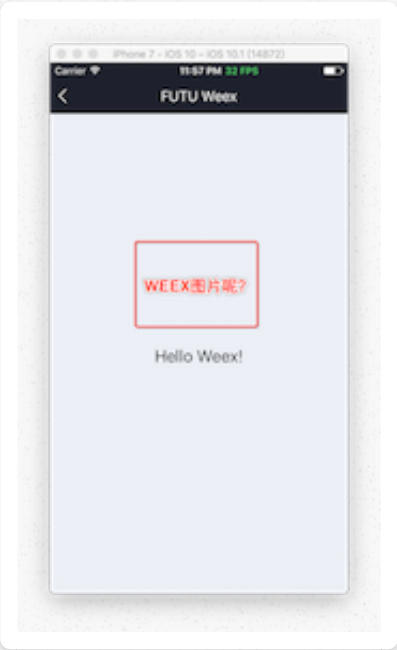
Weex支持两种Native加载页面的方式：

- 直接加载本地js bundle文件(跟上方的demo一样)：适合发布，下载server的js bundle后，直接加载本地js文件即可，速度更快
- 动态加载.we文件：适合开发时hot loading，可以看到页面改动效果，而不需要重新run

```
//进入.we文件目录后，执行以下代码启动本地serve
//这样当.we文件有变化时，本地serve就会动态生成新的js bundle
$ weex -s .

//然后render weex，  其中localhost可以用当前ip代替
NSString *urlString = [NSString stringWithFormat:@"http://%@:8081/%@"
, @"localhost", @"test.we"];
[_weexInstance renderWithURL:[NSURL URLWithString:urlString]];
```

至此，通过显示当前的vc，iOS客户端就可以展示weex对应的js bundle页面了，即可看到test.we(代码跟weex-bootstrap.we一样)对应的原生界面了；但是现在有个问题，在浏览器和playground里打开时，都有weex的图标，但是现在`WEEX图标没显示出来`,这是为什么呢？



5.2.1.4 Weex SDK只提供渲染，如果你需要类似请求网络，拉取图片，URL跳转这些特性，或者需要自定义控件等额外的功能，那就需要自己实现对应的协议和接口，这种扩展能力是非常灵活的，这一部分也是Weex开发中最重要最常用的一节。[具体参见]

```
    //WXImgLoaderDefaultImpl 实现了下载网络图片的接口，供Weex调用即可，这个就是一个Handler
    [WXSDKEngine registerHandler:[WXImgLoaderDefaultImpl new] withProtocol:@protocol(WXImgLoaderProtocol)];

    //WXSelectComponent 实现了自定义选择图片的控件，这样JS就可以直接使用这个控件了，而这个不是官方控件
    [WXSDKEngine registerComponent:@"select" withClass:NSClassFromString(@"WXSelectComponent")];

    //WXEventModule 实现了URL/Scheme跳转，这样JS就能很方便的吊起Native的原生界面了，这就是一个Module
    [WXSDKEngine registerModule:@"event" withClass:[WXEventModule class]];
```

5.2.1.5 注册WXImgLoaderDefaultImpl这个handler，我使用了SDWebImage库实现图片拉取

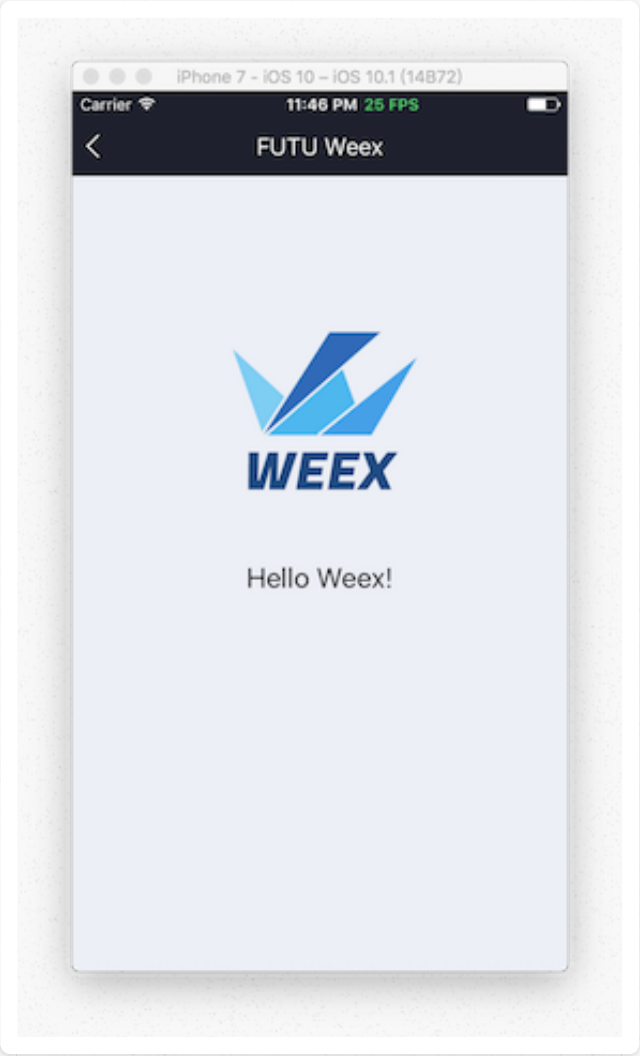
```
    #pragma mark WXImgLoaderProtocol

    - (id<WXImageOperationProtocol>)downloadImageWithURL:(NSString *)url imageFrame:(CGRect)imageFrame userInfo:(NSDictionary *)userInfo completed:(void(^)(UIImage *image,  NSError *error, BOOL finished))completedBlock
    {
        if ([url hasPrefix:@"//"]) {
            url = [@"http:" stringByAppendingString:url];
```

```
        }
        return (id<WXImageOperationProtocol>)[[SDWebImageManager shared
Manager] downloadImageWithURL:[NSURL URLWithString:url] options:0 progr
ess:^(NSInteger receivedSize, NSInteger expectedSize) {

        } completed:^(UIImage *image, NSError *error, SDImageCacheType
cacheType, BOOL finished, NSURL *imageURL) {
            if (completedBlock) {
                completedBlock(image, error, finished);
            }
        }];
    }
```

这样再重新运行APP，即可以看到如下与Playground APP里一模一样的界面效果了



5.2.2 Weex界面编程（Vue.js代码）

上一节中，我们实现了在Native工程里展示简单的Weex的界面，但是实际开发中经常会遇到JS与Native交互的过程，上一节使用到的图片下载是Weex主动调取的过程，而我们的Demo还需要需要其他稍微复杂一点的交互,例如，JS里每个Cell点击需要跳转到对应的页面，也就是处理Scheme跳转，接下来将从编写Weex界面开始，一点点实现我

们的Demo.

1. Weex使用的Vue框架，所以基本语法的熟悉可以先到本篇文章的 <六.不得不说的Vue.js>快速过一下;
2. Weex页面结构
3. Weex页面的生命周期

```
<script>
  module.exports = {
    data: {},
    methods: {},

    init: function () {
        //此时没有执行数据绑定，也没有创建Virtual-DOM，所以不能通过`this`获取`data`数据
        console.log('在初始化内部变量，并且添加了事件功能后被触发');
    },
    created: function () {
        //created 的名称有点令人迷惑，会让人以为节点全部都创建完成了，其实只是刚完成了数据绑定，还没开始编译模板。
        //此时可以通过 this 操作 data 中的数据，也可以调用 methods 中的方法，但是获取不到 Virtual-DOM 的节点。
        console.log('完成数据绑定之后，模板编译之前被触发');
    },
    ready: function () {
        //ready 开始执行时，表示当前组件已经渲染完成。
        //这个过程是自底向上触发的，会首先先执行子组件的 ready方法
        //此时可以通过`this.$el(id)`获取到节点的 Virtual-DOM，也可以通过`this.$vm(id)`获取到子组件的Virtual-DOM实例。
        //但是这里不要频繁操作数据，每次数据变动都会触发局部页面的重新渲染
        console.log('模板已经编译并且生成了 Virtual DOM 之后被触发');
    },
    destroyed: function () {
        //和 ready 类似，它也是自底向上执行，先触发子组件的 destroyed 方法，再触发自身。
        console.log('在页面被销毁时调用');
    }
  }
</script>
```

4. Demo页面实现: DEMO中列表是一行2个或4个格子，在Native实现是也比较简单，就是一个典型的UICollectionView, 但Weex只提供了一个类似UITableView的Component，那就是`list`，但是我们需要实现每个Cell带两个格子的布局

- template布局: 使用list, 另外实现两个横向的格子，Weex提供了很方便的属性设置

`repeated` 它不仅仅用在list的每一行，只要是重复的view就可以用这个标签，只需要数据源是两个，那Weex就会横向生成了两个格子，而且默认是均分的

```
<template>
<list class="list">
    <div class="cell" style="height:{{cell_height}}; border-bottom-wi
dth: {{border_width}}" repeat="{{dataList}}" row_index="{{$index}}">
        <div class="col" style="border-left-width: {{border_width}}"
repeat="{{rowList}}" col_index="{{$index}}">
            <image style="width: 64; height: 64;margin-left: 38;" src
="{{iconUrl}}"></image>
            <div style="flex: 1;flex-direction: column;">
                <text class="mainTitle" style="margin-left: 12;">{{na
meCn}}</text>
                <text class="subTitle" style="margin-left: 12; margin
-top: 10">{{summary_cn}}</text>
            </div>
        </div>
        #这个blankCell主要是为了画分割线，当某一行只有一个数据时，右侧为空，但
是没有分割线，会很别扭
        <div class="blankCell" style="border-left-width: {{border_wid
th}}" if="{{($index === (dataList.length-1)) }}">
        </div>
    </div>
</div>
</template>
```

- style设置: 类似于CSS里的style设置，给可以抽出重复的style

```
<style>
.list {
    flex-direction: column;
    background-color: transparent;
}
.cell {
    background-color: #ffffff;
    flex-direction: row;
    width: 750;
    border-bottom-color: #DDE2EB;
}

.col {
    flex:1;
    flex-direction: row;
    align-items: center;
    border-left-color: #DDE2EB;
```

```
}

.blankCell {
    flex: 1;
    flex-direction: row;
    border-left-color: #DDE2EB;
}

.mainTitle {
    font-size: 32;
    color: #284058;
}

.subTitle {
    font-size: 24;
    color: #808F9E;
}

</style>
```

- script: 等同于js逻辑代码，里面包含了各种事件，方法，以及数据源等

```
<script>
module.exports = {
    data: {
    rawDataList:[
    {
        nameCn: "业务办理",
        summary_cn: "入金/转仓/IPO",
        iconUrl: "http://cdn.futunn.com/images/homepage/entrance/mine
_icon_service.png?20161219",
        linkUrl: "https://my.futu5.com/account/stock-service#/"
    },
    {
        nameCn: "模拟炒股",
        summary_cn: "参赛拿千元奖金",
        iconUrl: "http://cdn.futunn.com/images/homepage/entrance/mine
_icon_sim_stock.png",
        linkUrl: "https://mobile.futunn.com/match"
    },
    {
        nameCn: "积分任务",
        summary_cn: "小积分兑好礼",
        iconUrl: "http://cdn.futunn.com/images/homepage/entrance/mine
_icon_mission.png",
        linkUrl: "https://mobile.futunn.com/credits/index#/"
    },
```

```
    {
        nameCn: "富途种子",
        summary_cn: "免佣新玩法",
        iconUrl: "http://cdn.futunn.com/images/homepage/entrance/mine
_icon_seed.png",
        linkUrl: "https://my.futu5.com/commseed/list"
    },
    {
        nameCn: "邀请好友",
        summary_cn: "推荐开户拿免佣",
        iconUrl: "http://cdn.futunn.com/images/homepage/entrance/mine
_icon_invite.png",
        linkUrl: "https://www.futunn.com/invite"
    },
    {
        nameCn: "我的收藏",
        summary_cn: "牛牛圈/资讯文章",
        iconUrl: "http://cdn.futunn.com/images/homepage/entrance/mine
_icon_collect.png",
        linkUrl: "futunn://collection"
    },
    {
        nameCn: "奖励领取",
        summary_cn: "你的红包都在这",
        iconUrl: "http://cdn.futunn.com/images/homepage/entrance/mine
_icon_gift.png",
        linkUrl: "https://www.futunn.com/mprize/index#/"
    },
    {
        nameCn: "精彩活动",
        summary_cn: "更多好礼等你拿",
        iconUrl: " http://cdn.futunn.com/images/homepage/entrance/min
e_icon_active.png?1",
        linkUrl: "https://www.futunn.com/activity/four-years"
    },
    {
        nameCn: "帮助中心",
        summary_cn: "炒股常见问题解答",
        iconUrl: "http://cdn.futunn.com/images/homepage/entrance/mine
_icon_help.png",
        linkUrl: "https://www.futu5.com/faq/h5-hc-index#/"
    },
    {
        nameCn: "产品反馈",
        summary_cn: "对牛牛吐吐槽",
        iconUrl: "http://cdn.futunn.com/images/homepage/entrance/mine
_icon_feedback.png?1",
        linkUrl: "futunn://feedback"
    },
    {
```

```
            nameCn: "在线客服",
            summary_cn: "在线客服实时解答",
            iconUrl: "http://cdn.futunn.com/images/homepage/entrance/mine
_icon_ask.png",
            linkUrl: "futunn://chat/10002"
        }
        ],
        //界面真正使用的数据源
        dataList:[],
        //手机上显示的cell高度
        device_cell_height: 70,
        //we文件里显示的cell高度
        cell_height: 140,
        //分割线宽度
        border_width: 0.5
        },
    created: function () {
        //数据源，目前是本地写死的，这里也可以请求http接口获取动态数据
        this.dataList = this.handleDataTransform();

    },

    methods: {
        //转换数据结构，将数组转成一行两个的形式
        handleDataTransform: function() {
            var res = [];
            var raw = this.rawDataList;
            for (var i = 0; i < raw.length; i ++)
            {
                //一行2个数据，也可以设置成一行4个
                if (i%2 == 0)
                {
                    var tempList = [];
                    tempList.push(raw[i]);
                    if (i + 1 < raw.length)
                    {
                        tempList.push(raw[i+1]);
                    }
                    res.push({rowList: tempList});
                }
            }
            return res;
        },

        }
    }
    </script>
```

- 现在重新生成js，再次进入运行工程，进入界面即可看到如下新的界面了

- 接下来处理JS的点击事件，即Scheme的解析: 这一步主要是使用扩展功能里的Module，之前也提到过，Native注册自定义的Module，js就能调用了

```
#import <Foundation/Foundation.h>
#import <WeexSDK/WXModuleProtocol.h>

@interface WXEventModule : NSObject <WXModuleProtocol>

@end

#import "WXEventModule.h"
#import "FLUrlHelper.h"
#import "FTWebViewController.h"

@implementation WXEventModule

@synthesize weexInstance;

//一定要在这里声明，这样才能在js里调用该方法，打开URL的event
```

```
WX_EXPORT_METHOD(@selector(openURL:))
//js打日志的接口
WX_EXPORT_METHOD(@selector(log:))

- (void)openURL:(NSString *)url
{
    //解析scheme
    if ([url hasPrefix:FUTUNN_URL_SCHEME]) {
        [FLUrlHelper handleOpenUrl:[NSURL URLWithString:url]];
    } else {
        //解析不了时，当成普通的链接打开
        FTWebViewController *web = [[FTWebViewController alloc] init]
;
        web.hidesBottomBarWhenPushed = YES;
        [web setUrlString:url andTitle:nil];

        [[weexInstance.viewController navigationController] pushViewC
ontroller:web animated:YES];
    }
}

- (void)log:(NSString *)text
{
    NNLogFile(@"MODULE_WEEX", @"%@,", text);
}

@end
```

- js调用Native里的Module方法

先在script 的 methdos里定义js里点击每个格子的方法，参数是当前点击的格子对应数据dic

```
    handleCellClick: function (cellDic) {
        //打日志接口
        this.$call("event", "log", 'cellDic = ' + cellDic);
        //调用本地的scheme处理dic里的linkUrl
        this.$call("event", "openURL", cellDic['linkUrl']);
    }
```

然后在template里添加点击`onclick`方法`handleCellClick`

```
    <div class="col" repeat="{{rowList}}" onclick="handleCellClick(rowL
ist[$index])">
```

重新运行后就可以点击每个格子执行对应的跳转动作了

- 调整为一行4个格子的布局

调整格子高度

```
  //we文件里显示的cell高度
  cell_height: 180,
```

调整格子`.col`的style为纵向布局

```
.col {
    flex:1;
    //纵向布局
    flex-direction: column;
    align-items: center;
    border-left-color: #DDE2EB;
  }
```

将template里的单个格子的布局改成，icon在上，文字在下的布局

```
    <div class="col" style="border-left-width: {{border_width}}" repeat
="{{rowList}}" col_index="{{$index}}" onclick="handleCellClick(rowList[
$index])">
        <image style="width: 48; height: 48;margin-top: 40;" src="{{ico
nUrl}}"></image>
        <text class="mainTitle" style="margin-top: 25;">{{nameCn}}</tex
t>
    </div>
```

把script里数据源转换成每一列包含4个元素

```
    handleDataTransform: function() {
        var res = [];
        var raw = this.rawDataList;
        for (var i = 0; i < raw.length; i ++)
        {
            if (i%4 == 0)
            {
                var tempList = [];
                tempList.push(raw[i]);
                if (i + 1 < raw.length)
                {
                    tempList.push(raw[i+1]);
```

```
                }
                if (i + 2 < raw.length)
                {
                    tempList.push(raw[i+2]);
                }
                if (i + 3 < raw.length)
                {
                    tempList.push(raw[i+3]);
                }
                res.push({rowList: tempList});
            }
        }
        return res;
    },
```

重新进入页面就可以看到如下效果了：

5.3 Demo总结

整体来说Weex开发这个Demo是比较简单的，尤其对有Web开发经验的童鞋就更容易上手了，列一下刚开始需要熟悉的文档和教程：

* 官方教程: 熟悉Weex的全局概念
* 官方手册: 其中`Advanced`一节很重要，客户端开发必须要熟悉`Module`,`Component`,`Handler`这三个重要特性
* awesome-weex: 基本包含了现有比较完整的Weex教程和Demo

六. 不得不说的Vue.js

6.1 Vue.js 是什么?

Vue.js（读音 /vjuː/, 类似于 view） 是一套构建用户界面的 渐进式框架。与其他重量级框架不同的是，Vue 采用自底向上增量开发的设计。Vue 的核心库只关注视图层，并且非常容易学习，非常容易与其它库或已有项目整合。另一方面，Vue 完全有能力驱动采用单文件组件和 Vue 生态系统支持的库开发的复杂单页应用。

Vue.js 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。

6.2 响应式数据绑定

将数据和视图绑定，也就是Data-Binding，这样当数据变化时，界面就能同步变化，实现了数据驱动的响应式编程，表现形式类似于OC里的KVO，这种模式就是经典的MVVM模式。

```
<!-- this is our View -->
<div id="example-1">
    Hello {{ name }}!
</div>

<!-- this is our Model -->
var exampleData = {
    name: 'Vue.js'
}

<!-- this is our ViewModel -->
var exampleVM = new Vue({
    el: '#example-1',
    data: exampleData
})
```

6.3 样式、逻辑和界面的分离

```
<!-- css -->
<style>
.message {
```

```
    color: red;
}
</style>

<!-- html -->
<template>
    <div class="message">{{ message }}</div>
</template>

<!-- js -->
<script>
export default {
    props: ['message'],
    created() {
        console.log('MyComponent created!')
    }
}
</script>
```

6.4 组件化

Web页面布局本质上就是构建DOM树，其由div，span等元素构成，而这些元素就是一个个组件，Vue.js可以很方便的自定义组件；

- 小巧精致
- 可重用
- 自包含，高内聚

```
// example组件定义
var Example = Vue.extend({
    template: '<div>{{ message }}</div>',
    data: function () {
        return {
            message: 'Hello Vue.js!'
        }
    }
})

// register it with the tag <example>
Vue.component('example', Example)

// example组件使用
<example></example>
```

七. 总结

Weex的方案很好的解决了移动端动态化这个问题，但开源还不久，开源社区不够活跃，这点对开发者而言是机遇也是考验，希望Weex能越来越强大。

参考资料

- 官方教程
- 官方手册
- 关于Weex你需要知道的一切
- 深度揭秘阿里移动端高性能动态化方案Weex
- weex vs react-native
- VUE.JS: A (RE)INTRODUCTION
- Weex学习与实践(一):Weex,你需要知道的事
- 对无线电商动态化方案的思考