

# Table of Contents

介绍	1.1
任务列表	1.2
添加的小功能列表	1.3
模块	1.4
加解密模块	1.4.1
获取手机系统相关信息	1.4.2
跳转到native静态页面	1.4.3
远程资源本地加载	1.4.4
开放支付SDK调用	1.4.5
转场跳转【非push-pop】	1.4.6
statusBar字体颜色设置-iOS	1.4.7
获取页面meta信息	1.4.8
刷新渲染页面	1.4.9
组件	1.5
native组件	1.5.1
安全键盘input	1.5.1.1
web组件	1.5.2
处理器	1.6
导航控制器适配苹果右滑返回	1.6.1
深入理解	1.7
近期问题清单	1.8
web问题	1.8.1
android问题	1.8.2
iOS问题	1.8.3
flex布局	1.9
web syntax	1.10
动态下载资源相关	1.11

---

动态引擎	1.12
weex参考资料	1.13
附录I	1.14
语义化版本	1.15
层次结构效果图	1.16
工具	1.17
DeepDream接入文档	1.18
web接入文档	1.18.1
android接入文档	1.18.2
iOS接入文档	1.18.3

---

# weex

## 任务列表

项目故事版

## 已经实现的插件列表

加解密 网络请求

## 仓库

模块	仓库地址
web	git@192.168.1.77:jfwebx/jfwebx.we.git
andoird	git@192.168.1.77:jfwebx/jfwebx.android.git
ios	git@192.168.1.77:jfwebx/jfwebx.ios.git

## 参考资料

1. 无线电商的思考
2. weex教程

# 任务列表

环境搭建

SDK相关的学习

weex前端框架搭建，包括打包，配置es6，抽离公用部分

实现一个动态web首页，包括轮播，tabbar，面板。

实现一个加解密的module

实现一个Component 安全键盘

登陆页面

更新文档

集成一个简易的demo

csslayout 布局技术的学习

集成开放支付SDK 封装web版的支付控件

点击进入一个微店，可以选购买商品，连接支付SDK

网络请求pipe封装，加入LRU缓存等，4个端，包括web, ios, android, 网关

web动态化引擎，web资源版本化离线动态更新，web页面加载

动态化配置管理平台,提供web端和api两个端口进行实现，api可封装成插件，其他平台进行集成

web推送技术的实现

为我们自己以及圈子写一个应用， <https://www.xiaomiquan.com/> ,类似于微博，分享快乐

从demo中抽离出动态化组件，并单独进行研发

动态化接口的对接以及后端接口参考实现

## 图像例子

```
sequenceDiagram
    A->> B: QueryKJ
    B->> C: Forward query
    Note right of C: Thinking...
```

```
graph TD;
    A-->B;
    A-->C;
    B-->D;
    C-->D;
```

## 任务列表

---

## 临时添加的小功能

### 1. `fontUrl`

在渲染文件开始的时候传入该参数，用来传递放在本地的字体文件的路径

## 一. 现有模块列表

1. `encryptPwdmoudle`
2. `systemInfoModule`
3. `ddjumpmoudle`
4. `loadResourceModule`
5. `openPayModule`
6. `transitionModule`
7. `statusBarModule`
8. `loadMetaModule`
9. `refresh`

## 二. 模块定义规范

### native端与web端之间数据传输格式以及参数规范

1. 返回 JSON 的格式
2. 必须包括 code 节点，通过 data 传递数据（可选）
3. 0000 表示成功，其余定义各种异常

```
{  
  "code" : "0000",  
  "data" : "xjkdjskjisdanfakdjvkcavnkdv"  
}
```

ps: code的规范，前2位代表模块，后2位表示错误码

状态码	模块	错误描述
0101	01模块	01 格式错误

## 二.moudle以及组件定义相关

1. 类文件必须以DD (DeepDream) 开头
2. 插件name、moudle的name均以dd开头,全部小写
3. 返回参数值必须全部为小写字母
4. ...

## encryptPwdModule

### 概述

简单封装一个用于密码加密和解密的小模块。将秘钥等等信息直接打包，防止被抓包攻击。

### API

#### 1. *encryptString(encryptstr, callback)*

加密

##### 参数定义

  |  encryptstr

需要加密的字符串明文，必填

#### 2. *decryptString(decryptstr, callback)*

解密

##### 参数定义

  |  decryptstr

需要解密的字符串密文，必填

### 返回参数结构定义

```
{  
  "code": "0000",  
  "data": "加密后的密文或者解密后的明文"  
}
```

### Usage

```
//引入
var DDEncryptModule = require('@weex-module/encryptPwdModule');

//调用
//加密
DDEncryptModule.encryptString("1234567890", function(e) {
    nativeLog("1234567890 encrypt is" + e.encodeStr);
});

//解密
DDEncryptModule.decryptString("0987654321", function(e) {
    nativeLog("0987654321 decrypt is" + e.decodeStr);
});
```

## systemInfoModule

### 概述

众所周知，`ios` 和 `android` 系统之间有很大的区别，`web` 端并不能准确的对其做出更好的适配。为了方便 `web` 做更灵活的相关适配，这里简单封装一个获取系统信息的模块 `DDSystemInfoModule`。

### API

#### `getSystemInfo(options)`

获取系统相关信息

### 参数

- `options` 一些方便扩充的参数，可不传

### 返回参数结构定义

```
{  
    "code": "0000",  
    "data": {  
        "platform": "ios", //ios || android  
        "system_version": "10.1.1",  
        "screen_width": "375",  
        "screen_height": "667",  
        "nfc_support": false,  
        ... //其它待扩充参数  
    }  
}
```

*PS:* 参数值均小写处理

### 具体参数

- `platform`  
系统平台，`ios`或者`android`

- `system_version`

手机系统版本号

- `screen_width`

手机屏幕宽度

- `screen_height`

手机屏幕高度

- `nfc_support`

手机是否支持NFC

## Usage

```
//引入
var systemInfo = require('@weex-module/systemInfoModule');

//调用
systemInfo.getSystemInfo('', function(res) {
    nativeLog(res.data.platform);
    nativeLog(res.data.system_version);
    nativeLog(res.data.screen_width);
    nativeLog(res.data.screen_height);
});
```

## jumpNativeModule

### 概述

有时候我们需要native做一些静态的不会变的页面，当我们需要跳转到该native页面时，需要一个桥来处理这个跳转，这里是一系列的跳转（跳转到native静态页面）。

还有一种就是一个用自定义的webView打开一个链接url，这个时候也需要做一些桥的处理。

还有另外一种是直接用系统浏览器打开一个url。

调用系统-拨打电话、发送邮件、打开通讯录。

### API

#### 1. *openUrlInWebView(url)*

用自定义的webView打开url

##### 参数定义

url

需要打开的链接

#### 2. *gotoSettingViewController(option)*

跳转到设置页面

##### 参数定义

option

需要传递的数据值，可为空，具体传递参数需要具体讨论

#### 3. *gotoAboutViewController(option)*

跳转到关于页面

##### 参数定义

option

需要传递的数据值，可为空，具体传递参数需要具体讨论

## Usage

```
//引入
var jumpmoudle = require('@weex-module/jumpNativeModule');

//1.openUrlInWebView
jumpmoudle.openUrlInWebView("http://www.baidu.com",function(e) {
    nativeLog(e);
});

//2.gotoSettingViewController
jumpmoudle. gotoSettingViewController("",function(e) {
    nativeLog(e);
});

//3.gotoAboutViewController
jumpmoudle. gotoAboutViewController("",function(e) {
    nativeLog(e);
});
```

## 更多需要持续集成开发的跳转

- 拨打电话
- 发送邮件
- 打开通讯录
- 跳转到系统的设置页面
-

## loadResourceModule

### 概述

在经过一段时间的开发以及测试发现，完全运用远程链接的方式去加载并渲染一个页面，相当耗费用户流量以及时间，因为每次渲染都会去重新下载该js文件并下载页面中所用到的所有资源（图片、字体等）。所以我们通过讨论想出一种解决方式，通过本地存储一份web的js文件以及所用到的其他资源文件，然后在js跳转新页面以及加载资源的时候调用本地的文件，这样可以保证所有的资源都是通过本地化的方式来加载；如果文件在本地已存在，则直接加载本地文件，否则native这边去下载资源到本地，并返回本地路径给到web那边，这样做好处就是既节省了时间成本，也相应的提高了用户体验（本地加载比远程快的不是一星半点）。所以我们这个模块 `loadResourceModule` 也就应运而生了。

### API

#### 1. `getUrlPath(string urlStr, callback)`

根据远程连接获取该文件的本地绝对路径

### 参数

- `urlStr`

文件远程连接地址，必传

### 返回参数结构定义

```
{  
    "code": "0000",  
    "data": "file:///xxxxx/JFFolder/homeIcon.png"  
}
```

- `data`

文件本地连接地址

#### 2. `getRoutePath(string key, callback)`

根据模块的Key值,获取路由绝对路径

## 参数

- key

js文件对应的Key值，必传

## 返回参数结构定义

```
{  
    "code": "0000",  
    "data": "file:///xxxxx/JFFolder/home.js"  
}
```

- data

本地js文件的绝对路径地址

## Usage

```
//引入  
var loadPath = require('@weex-module/loadResourceModule');  
  
//调用资源下载  
var fileUrl = "http://www.jfpal.com/index.js"  
loadPath.getUrlPath(fileUrl, function(res) {  
    nativeLog(res.data)  
    //file:///User/dadasdasd/IMAGEFolder/homeIcon.png  
});  
  
//获取路由文件全路径  
loadPath.getRoutePath('keyStr', function(res) {  
    nativeLog(res.data)  
    //file:///User/dadasdasd/JSFolder/index.js  
});
```

## openPayModule

### 概述

由于现有的开放支付SDK是由native开发出来的，所以我们需要搭建这个桥 `openPayModule` 来调用即富开放支付SDK来完成刷卡支付功能。

### API

#### *pay(payInfo, callback)*

调用支付功能

### 参数

- `payInfo`  
支付所用具体参数对象
- 详细参数如下

```
{
    "organizationId" : "000000000000", //组织机构号
    "orderId" : "123456789987654", //订单号
    "cash" : "1000", //交易金额（单位为分-> 传入100即表示1元）
    "goodsDesc" : "该产品是xxx", //产品描述
    "merchantName" : "iPad", //产品名称
    "merchantId" : "0020000002", //产品商户号
    "paymentWay" : "有卡支付", //支付方式
    "longitude" : "131.111123", //经度
    "latitude" : "12.111111", //维度
    "publicKey" : "kfasjkhfgkasfjkasfvkjashdfbajkshdfvadhjksf", //校验后端信息
}
```

### 返回参数结构定义

```
{
    "code": "0000",
    "data": "交易成功", //支付状态
}
```

## Usage

```
//引入
var DDPay = require('@weex-module/openPayModule');

var payInfo = 具体参数;

DDPay.pay(payInfo, function(res) {
    nativeLog(res.data)
    //file:///User/dadasdasd/IMAGEFolder/homeIcon.png
});
```

## transitionModule

### 概述

简单实现presentViewController的方式来实现控制器之间的跳转

### API

#### 1. *present(options, callback)*

弹出一个新的页面

### 参数

- `options` (object): some options.
  - `url` (string)  
新页面的链接（全路径），不能为空
  - `animated` (string)  
是否需要动画， 默认有动画，可以为空
- `callback` (object): the callback function after executing this action.

### 返回参数结构定义

```
{  
  "code": "0000",  
  "data": options,  
}
```

- `code` 状态码
- `data` 调用的时候所传过来的参数对象

#### 2. *dismiss(options, callback)*

关闭弹出的那个页面

## 参数

- `options` (object): some options.  
想要传递的一些参数
- `callback` (object): the callback function after executing this action.

## 返回参数结构定义

```
{  
  "code": "0000",  
  "data": options,  
}
```

- `code` 状态码
  - `data` 调用的时候所传过来的参数对象
- 

## Usage

```
//引入
var transition = require('@weex-module/transitionModule');

//打开新页面
//该对象不可为空，url必传(文件全路径)
var params = {
    'url': 'navigator-demo.js',
    'animated' : 'true',
}

transition.present(params, function(res) {
    nativeLog(res.data)
});

//关闭打开的页面
//该对象完全自定义传入，可不传
var params1 = {
    'status': '完成',
    'response': '2134567890',
}

transition.dismiss(params1, function(res) {
    nativeLog(res.data)
    //file:///User/dadasdasd/JSFolder/index.js
});
```

## statusBarModule

### 概述

苹果在iOS7以上系统，设备状态栏与导航栏的一体化处理，我们需要对状态栏字体颜色做控制与处理，目前仅支持两种颜色（`dark` 与 `light`），方便更灵活动态的控制状态栏字体颜色。

### API

#### `setStatubarStyle(styleStr, callback)`

### 参数

- `styleStr`

状态栏风格，该参数为必填参数，可取的值目前有：

- `dark`

暗黑风格

- `light`

亮白风格

### Usage

```
//引入
var statusbar = require('@weex-module/statusBarModule');

//调用 只能设置两种值 'dark' 'light'
statusbar.setStatubarStyle('light', function(res){

});
```

## loadMetaModule

### 概述

动态化配置一些页面属性，该属性的集合从服务端获取，由 `native` 存储并提供给 `web` 使用

### API

#### `getSourceMeta(key, callback)`

获取某个单页的meta信息

### 参数

- `key`

页面对应的key值，必传，(可能是页面名)

### 返回参数结构定义

```
{  
  "code": "0000",  
  "data": {},  
}
```

PS: `data`里面的数据依据实际情况而定

### Usage

```
//引入  
var metamodule = require('@weex-module/loadMetaModule');  
  
//调用api  
metamodule.getSourceMeta('home', function(res) {  
  nativeLog(res.data)  
});
```

获取页面meta信息

---

## refresh

### 概述

在某些特定的情况下（比如说更换皮肤），需要重新刷新(渲染)整个页面，所以需要我们提供接口来刷新整个渲染页。

### API

#### *refreshPage()*

刷新渲染页

### 参数

无

### 返回参数

无

### Usage

```
//引入
var ddrefresh = require('@weex-module/refresh');

//刷新
ddrefresh.refreshPage()
```

## DeepDream自定义的组件

封装一些比较通用的页面或者页面模块。

### 已有组件列表

1. `<ddsecurityinput>`

### 简单的规范

## native 端自定义的组件

封装一些比较通用的View提供给web做动态化处理。

### 简单的规范

- 组件注册时候的name(在html中用到的组件名)全部统一小写字母
- 组件的新增属性名、event、属性值等也全部统一小写字母
- 组件命名开头以 `dd` 开头, `native` 代码可以大写 `DD`

## 安全键盘组件的实现

---

### <ddsecretyinput>

思路:

在input组件的基础下做些修改，将自有的 securitykeyboard 绑定到 ddsecretyinput 组件上面。

方法:

在原有的基础上扩充一种type类型（ddpassword），其他的处理方式与原有的保持一致。仍然支持input组件所拥有的所有属性的使用。

## 定义

- 组件名(name): ddsecretyinput
- 新增type(new type): ddpassword

## 新增扩展

- clearmode
  - iOS端 UITextField 所拥有的 clearButtonMode 属性的简单设置，在官方提供的 input里面未包含该属性的设置方法。
  - 可设置的value（设置小按钮的显示时间）如下：
    - always 一直显示
    - whileedit 开始输入的时候显示
    - endedit 结束输入的时候显示
  - 支持所有type的ddsecretyinput
- random
  - 安全键盘上面数字、字母符号显示顺序的随机性
  - true 表示打乱顺序
  - false 表示不打乱顺序
  - 需要注意一点，random 属性只对type为 ddpassword 的 ddsecretyinput 起作用，其他type均无任何效果

- `maxlength` 属性仅作用于数字键盘，用于做银行卡密码的校验,设置 `maxlength` 后无法切换到字母键盘以及符号键盘
- 其他属性与官方一致,全部支持

一个简单例子：

```
<ddsecurityinput
  class="securityinput"
  type="ddpassword"
  clearmode="whileedit"
  autofocus="false"
  random="true"
  placeholder="请输入用户名以及密码"
  onChange="onchange"
  onInput="onInput">
</ddsecurityinput>
```

### PS:需要注意一点

使用该组件的时候，必须要 `<ddsecurityinput type="ddpassword"> </ddsecurityinput>` 写. 而不能 `<ddsecurityinput type="ddpassword" />`

## DDNavigationNewImpl

### 概述

一个简单的适配器，用于导航控制器适配苹果右滑返回，手指慢慢拖动返回功能

## 深入理解

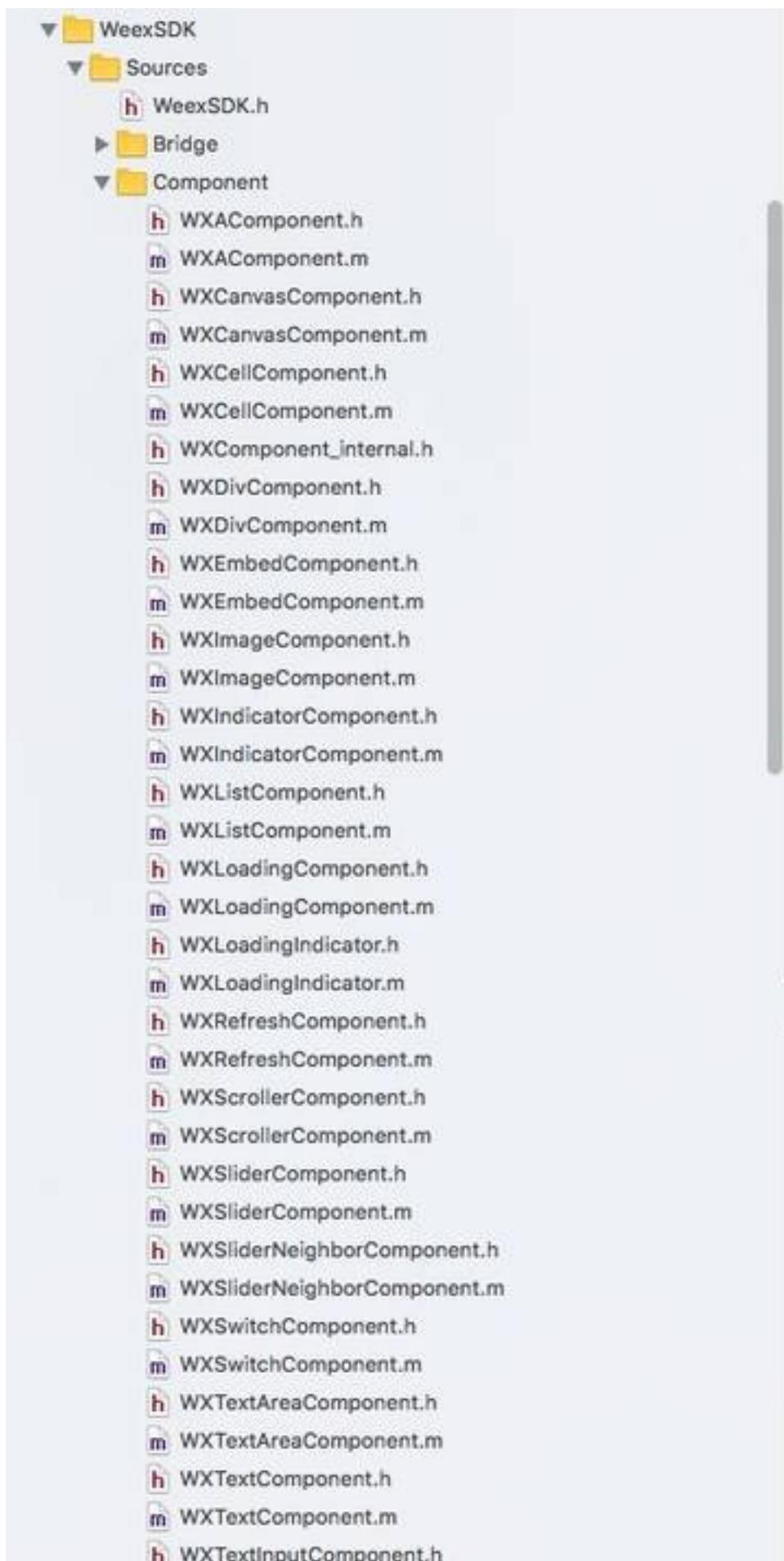
### 1. 核心概念

- Component(组件) 在屏幕内可见，有特定行为。能被配置不同的属性和样式,能响应用户交互. 常见的组件有: `<div>` , `<text>` , `<image>`
- Module(模块) 是被JS Framework调用的API. 可以采用异步的方式进行调用, 例如: 发送AJAX
- Handler( 处理器) 是为了解耦各种框架实现的一种机制。比如 SDImage

### 1. 已经实现的扩展,高级别的组件

weex的一个线程运行 js引擎 `main.js`

```
// WXSDKEngine#initSDKEnviroment
[self _registerDefaultComponents];
[self _registerDefaultModules];
[self _registerDefaultHandlers];
```





# 待解决问题

## 1. we页面与native之间跳转通知的问题

weex本身的生命周期 参考 : <https://yq.aliyun.com/articles/59936>

WXBaseViewController 绑定了view的生命周期， 通过module进行回调给weex 实例， 实例中绑定 viewdisappear 等事件。参考：

<https://github.com/alibaba/weex/issues/201>

## 2. 通用业务逻辑复用AOP的实现

是否可以在js 引擎中 执行， 这样所有weex实例共享一份代码。当前通过require common 模块来实现

## 3. 路由的思考

- <https://github.com/alibaba/weex/issues/959>

## 4. 整合 数据流框架

- 参考 <https://github.com/Jinjiang/weex-x>

## 5. Component 的生命周期

## web 问题

1. 图片容器默认需要size,否则ui渲染会重叠;
2. 引用远程资源url需要是完整路径。 错误写法://at.alicdn.com/t/font\_u2wvrspbk18xgvi.ttf ;
3. 盒子容器样式会重写;
4. 内建模块不支持嵌套调用; 如:

```
storage.getItem('x', function(e){  
    navigator.push()  
})
```

- i. ios、 android 宽高缩放比例不一致;
5. 不能发送事件;
6. 构建新组建外层必须包含实体节点(div,container...),否则onviewappear等函数, 将不被执行;

# Android 问题

1. 引用com.taobao.android:weex\_sdk:0.5.1@aar 做为weexSdk 在开发中会发现 wxinput没有光标, onchange事件不能获取,自定义键盘弹出会有系统的混乱;

方案: 使用playground项目里面的weex\_sdk 引入

2. 场景:当自定义键盘弹出后, 点击android物理返回按钮, 键盘不能关闭

方案: 在com.dd.weex.widget.KeyboardUtil类中监听EditText里面的 KeyListener。在key方法里处理按下物理按钮的事件即可。原因是android所有的View同样全部实现了KeyEvent.Callback接口并重写了该方法, onKeyUp方法用来捕捉手机键盘按键抬起的事件。

3. 去掉ActionBar后, web页面展示整体上移了一定单位;

方案: 在包com.dd.weex.utils.ScreenUtil, 屏蔽height-=actionBar这句代码

4. 自定义component的生命周期;

状态: 待研究

5. 场景: 使用远程URL渲染,遇到子页面打不开;

方案: 检查自己的代码里是否有用网络请求下载下来对应的js文件。远程渲染的过程是先下载远程js, 再渲染。具体可以参考playground 的WXpageactivity和WXSDKInstance

6. 在登录页面切换editText的时候, 在弹出自定义键盘后, 系统键盘不会自动关闭

方案: 在week\_sdk中, 找到  
com.taobao.weex.ui.componet.AbstractEditComponent类, 在该类中因控制键盘的方法是私有的, 我们自定义三个方法提供给子类。开启键盘, 关闭键盘, 键盘的状态; 然后再子类WXInput中实现什么时候控制键盘, 关键代码如下:

```
@Override
public boolean onTouch(View v, MotionEvent motionEvent) {
    if (motionEvent.getAction() == MotionEvent.ACTION_UP) {
        v.setFocusableInTouchMode(true);
        v.setFocusable(true);
        v.requestFocus();
        if (!isShowingSoftKeyboard()) {
            Log.d("onTouch", ".....显示键盘");
            v.postDelayed(new Runnable() {
                @Override
                public void run() {
                    openSoftKeyboard();
                }
            }, 16);
            return true;
        } else {
            Log.d("onTouch", ".....关闭键盘");
            v.postDelayed(new Runnable() {
                @Override
                public void run() {
                    closeSoftKeyboard();
                }
            }, 16);
            return true;
        }
    }
    return false;
}

@Override
public void onFocusChange(View v, boolean hasFocus) {
    if(!hasFocus){
        v.postDelayed(new Runnable() {
            @Override
            public void run() {
                closeSoftKeyboard();
            }
        }, 16);
        Log.d("onFocusChange", ".....关闭键盘");
    }
}
```

## 7. 加载本地sd下的js文件，不能正常渲染

方案: 在com.taobao.weex.WXSDKInstance 下添加如下代码

```
Uri uri=Uri.parse(url);
if(uri!=null && TextUtils.equals(uri.getScheme(),"file")){
    render(pageName, WXFileUtils.loadAsset(assembleFilePath(uri), mContext),options,jsonInitData,width,height,flag);
    return;
}
/*
 * 如果Scheme storage
 */
if(uri!=null && TextUtils.equals(uri.getScheme(),"storage")){
    render(pageName, read(url),options,jsonInitData,width,height,flag);
    return;
}
```

在 AndroidManifest 下对应的weexPageActiviry 添加一个scheme

```
<activity
    android:name=".activity.WeexPageActivity"
    android:label="@string/app_name"
    android:screenOrientation="portrait"
    android:theme="@style/AppNoTitle">
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="com.taobao.android.intent.category.DDWEEX"/>
        <data android:scheme="http"/>
        <data android:scheme="https"/>
        <data android:scheme="file"/>
        <data android:scheme="storage"/>
    </intent-filter>
</activity>
```

## 1. 自定义字体不能显示问题

方案: 研究com.taobao.weex.utils.TypefaceUtil 中的方法loadTypeface中的逻辑。 sdk的代码主要逻辑是TYPE\_LOCAL 是加载asset下的字体文件, TYPE\_FILE 是加载SD下任何目录的字体文件, TYPE\_NETWORK是加载网络的资源。 而之前我的写法一是: options.put("fontUrl", "file://assets/fonts/font2.ttf"); 这个路径对于sdk来说, 其实是TYPE\_FILE类型, url会被过滤成fonts/font2.ttf, 所以路径为fonts/font2.ttf的文件根本是不存在的; 写法二是: options.put("fontUrl", "/storage/emulated/0/Android/font3.ttf") 意思是想去加载sd上的字体文件, 但是在loadTypeface的逻辑表示该路径没有找到对应的scheme, 默认是TYPE\_LOCAL类型, 所以在assets路径下 /storage/emulated/0/Android/font3.ttf文件是不存在的, 所以不会显示出来;

加载字体文件正确写法:

```
1> options.put("fontUrl", "file:/storage/emulated/0/Android/font.ttf"); 2>
options.put("fontUrl", "fonts/font.ttf");
```

## iOS历史问题以及待处理问题列表

### 1. 安全键盘input事件不响应问题

- 状态: 已解决
- 解决方式: 通过代理的方式传递按键的每一次点击后输入框里面的值,然后去主动的调用onInput方法
- 遗留: 无

### 1. 渲染页面跳转后返回状态栏位置多出20像素的白条

- 状态:已解决
- 解决方式: 通过使用weex官方提供的 `WXRootViewController` 导航控制器, 并让自己的渲染控制器继承 `WXBaseViewController` 发现该问题不再出现。
- 遗留: 具体实现原理尚不清楚, 后续再研究
  - 已解决
  - 加上代码

```
if ([self respondsToSelector:@selector(setEdgesForExtendedLayout:)]) {
    [self setEdgesForExtendedLayout:UIRectEdgeNone];
}
```

即可, 意思就是取消全部布局 `EdgesForExtendedLayout` 的默认值为 `UIRectEdgeAll`,会让布局随着iOS6走, 所以头上会多20的状态栏高度空白, 该属性是iOS7新增的, 所以需要判断是否可以响应, 不然会在iOS6上面 crash (不过现在iOS6已基本退出市场, 可以不加判断)

### 1. weexSDK -v0.9.4 不支持bundle下面字体文件加载的方式

将下面的代码加入到 `wxUtility` 文件中的 `+ (void)getIconfont:(NSURL *)url completion:(void(^)(NSURL *url, NSError *))completionBlock` 方法中(line number : 404~410), 用于支持bundle文件夹下面的文件

```
//1. bundle File url
if ([url.absoluteString hasPrefix:[NSString stringWithFormat:@"file://%@",[[NSBundle mainBundle] bundlePath]]]) {
    completionBlock(url, error);
    return;
}

//2. Documents File url
```

## flex 弹性盒子布局参考资料

Flexbox 从本质上就是一个 Box-model 的延伸，我们都知道 Box-model 定义了一个元素的盒模型，然而 Flexbox 更进一步的去规范了这些盒模型之间彼此的相对关系。

[flex 布局介绍](#)

[css flexbox 语法](#)

[flex布局语法](#)

[android flexbox 布局](#)

[ios flexbox 布局](#)

# validator

## 概述

组件props数据指定验证.

## 使用方法

`validator.call(weex, { data: Type }) .type` 可以是下面原生构造器:

- String
- Number
- Boolean
- Function
- Object
- Array

## 范例

```
import validator from 'helpers/validator';

export default {
  data: {
    checked: [],
  },
  init: function () {
    validator.call(this, {
      sliders: Array,
    });
  },
}
```

---

# lodash

## 概述

验证器.

## 支持

- `fill`. 创建一个单位长度的空数组

```
_ .fill(length)
```

- `key`. 获取JSON所有keys, 返回 [key1, key2, ...]
  - `keyBy`. JSON按照key重组.
  - `isEmpty`. 判断是否为空.
  - `getType`. 获取类型.
  - `checkType`. 判断类型.
  - `isArray`. 是否为数组.
  - `isNumber`. 是否为数字.
  - `isFunction`. 是否为函数.
  - `isObject`. 是否为JSON.
- 

## platform

### 概述

平台功能.

### 支持

- `$env` . 获取系统环境参数.
  - `$h` . 计算高度.
- 

## dd.pipe

### 概述

公用类.

## prototype

- updateWeex

## 支持

- pop.
- push.
- toast.
- tabRoute. 获取tab资源路径.
- addFont. 添加字体.
- require.
- present.
- dismiss.

## 使用

```
var DD = require('dd.pipe');
export default new function () {
  let pipe;
  return {
    name: 'page',
    created: function () {
      pipe = new DD(this);
    },
    method: {
      update: function () {
        pipe.updateWeex(this);
      }
    }
  }
}
```

### 1.1 pop(params)

- 参数

- params {Object}
- Object { animated: Boolean }

- 范例

```
pipe.pop();
```

### 1.2 push(params)

- 参数

- params {Object} / {String}
  - Object { src: String, animated: Boolean }
  - String

- 范例

```
pipe.push({ src: 'login.js' });

pipe.push('home.js');

pipe.push('home');
```

### 1.3 toast(params)

- 参数

- params {Object} / {String}
  - message
  - duration

- 范例

```
pipe.toast('提示');

pipe.toast({
  message: '提示',
  duration: 2
});
```

### 1.4 tabRoute(params, callback)

- 参数

- params {String}
- callback

- 范例

```
pipe.tabRoute('pageName', function (url) {
  // url
});
```

### 1.5 present(params)

- 参数

- params {String}

- 范例

```
pipe.present('pageName');
```

## 1.6 dismiss()

- 参数

- 范例

```
pipe.dismiss();
```

---

# Store

## 概述

数据存储类.

## 支持

- set
- get
- getAll
- getAllKeys
- remove
- clear

## 使用

```
var store = require('store');
```

## 1.1 set(key, value)

- 参数

- key {String}
- value {Any}

- 范例

```
store.set('key', 'value').then(function (success) {
  console.log('success=====》', success)
}, function (error) {
  console.log('error=====》', error)
})
```

## 1.2 get(key)

- 参数

- key {String}

- 范例

```
store.get('key').then(function (success) {
  console.log('success=====》', success)
}, function (error) {
  console.log('error=====》', error)
})
```

## 1.3 getAll()

- 参数

- 范例

## 1.4 getAllKeys()

- 参数

- 范例

## 1.5 remove(key)

- 参数

- key {String}

- 范例

## 1.6 clear(dayDuration)

- 参数

- key {String}
- 范例

```
store.clear(dayDuration).then(function (success) {
  console.log('success=====》', success)
}, function (error) {
  console.log('error=====》', error)
})
```

## 动态下载资源文件的思考

### 1. 文件路径相关的依赖

- 渲染js文件中引入的路径是否需要提前告知web
    - iOS的Document路径每编译一次都会发生变化
  - 跳转js文件是否需要下载到本地
  - js中的文件路径是否需要配置化管理-- 或者通过后端处理
  - 通过插件的方式告诉web下载下来的文件的路径
1. 必须统一打入app包中的文件夹路径与下载到document下面的文件夹路径，要保证远程js调用的时候能找到
- 

## 动态更新（2016-12-7）

### 1. 方案：三端（serve、native、web）检查资源状态。

### 2. 解释：

- native获取serve版本信息，并和本地进行比较。吻合，则加载本地JS入口文件；不吻合，则下载服务器相应JS入口文件，再加载。
- web渲染加载资源通过传参类型、文件名等给native；
- native收到web资源请求后，对比本地资源。如果存在，返回路径；如果不存在，通过请求serve获取资源下载到本地，再返回路径；

### 1. 注意

文件名唯一。暂定 资源名.时间戳.类型 格式命名。

### 2. 流程图

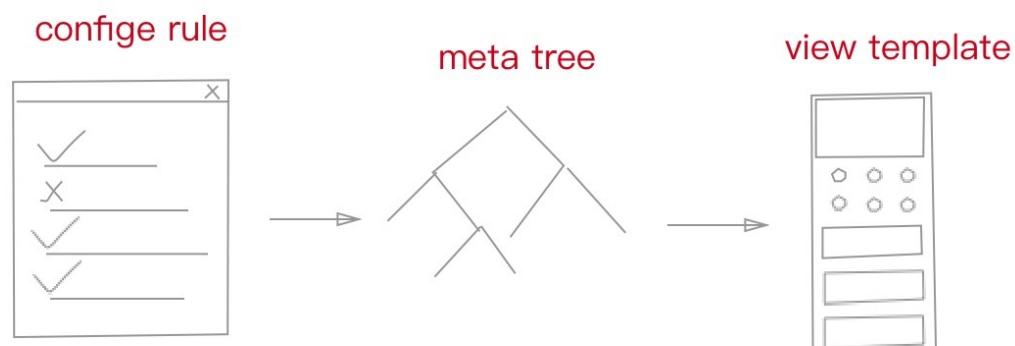
```
sequenceDiagram
    participant native
    participant serve
    participant web
    native ->> serve: 请求版本更新 GET /version/latest
    serve -->> native: 返回版本更新路由信息
    native->> web: 渲染入口
    web ->> native: 获取路由资源 getRoutePath
    native -->> web: 获取本地路由资源路径
```

```
graph LR;
A((native客户端)) --1.发送本地版本信息--> B((serve服务端));
B --2.返回最新WEB入口文件以及相关信息--> A;
A --3.渲染最新入口资源--> D((web前端));
D --4.发送资源唯一标识符--> A;
A --5.返回相应的本地资源路径--> D;
```

```
graph LR;
A((web前端)) --上传有更新资源--> B((serve服务端,维护资源,保证唯一性));
```

# 动态引擎

## 技术架构



## 接口文档

- **请求报文**

参数	格式	是否必须	备注	取值
version	string	N	请求版本号, 空表示最新	20161212175203/ lastest

- **回报报文**

参数	格式	是否必须	备注	取值
base	string	Y	基础路径	url
entry	string	Y	入口资源模块	resource中的key
version	string	Y	动态版本	20161212175203
resource	map	Y	资源模块列表.key为模块名,value为文件名	一维hash

## 1. 检查更新

- 请求方式:

```
GET http://localhost:4567/api/latest
```

- 返回范例

```
{  
  "base": "http://192.168.2.42:12580/dist/source/",  
  "entry": "index",  
  "version": "20161212175203"  
  "resource": {  
    "index": "index1.0.js",  
    "fd_home": "home_1.0.js",  
    "my": "my_2.0.js",  
  }  
}
```

## Weex相关文档以及资料

- [weex官网](#)
- [weex官方文档-英文](#)
- [weex官方文档-中文](#)
- [weex-Github](#)
- [weex官方demoApp](#)

## 其它有用的资料以及工具

- [码云-代码仓库管理](#)
- [vue技术实现页面的示例](#)
- [对无线电商动态化方案的思考](#)
-

## 附录I

### Native现有模块列表

模块	模块码	模块作用	模块链接
encryptPwdmoudle	10	用于密码加解密	调用native加解密
systemInfoModule	11	用于获取手机硬件信息	获取手机设备信息
ddjumpmoudle	12	web跳转到native页面	跳转Native页面
loadResourceModule	13	web通过native本地去获取资源路径	远程资源通过本地获取
openPayModule	14	调用开放支付SDK	开放支付SDK
transitionModule	15	页面转场动画	页面转场
statusBarModule	16	iOS状态栏设置颜色	状态栏颜色
loadMetaModule	17	获取页面meta信息	获取meta信息
refresh	18	刷新渲染页面	刷新渲染页面

### Native现有组件列表

组件	组件码	组件作用	组件链接
<ddsecretyinput>	30	即付宝安全密码键盘输入框	安全键盘

### 返回码简单定义

- 系统级

返回码	含义
0000	所有均正常
0001	网络错误
0002	下载失败
0003	文件删除失败
0004	文件存储失败

- 代码级错误码（模块、组件调用相关）

错误码	含义
01	缺少必填参数
02	参数格式不正确
03	找不到对应的值
04	未知错误

ps：代码级返回码的规范，前2位代表模块，后2位表示错误码；不一定所有的模块均会对应有相应的错误码

## 简单的示例

返回码	含义
1001	加解密模块缺少必填参数
1002	加解密模块参数格式不正确
1401	开放支付SDK缺少必填参数

## 语义化版本

语义化版本用三组数字表示，按照major.minor.patch（主要版本.次要版本.补丁）的顺序排列，比如2.3.1。

patch版本将通常每周发布，通常只是修复问题而不加入新的功能； minor版本将通常每月发布，加入一些新的功能但是相对旧版本来说并没有大的更改； major版本将通常每半年发布一次，加入一些新功能并且可能带来一些重大更改。

## 简单页面以及其对应的层次结构展示

### 首页截图



### 首页详细层次图

## 层次结构效果图



# 工具

## 接口mock工具

```
pip install flask-restful
```

```
python tools/mock_python.py
```

```
curl
```

## 动态引擎接入文档

这里我们团队对weex简单的封装了一下，包括新增一些常用的native组件、模块、处理器以及web端的组件、模块等。最主要的一块还是[动态引擎](#)（动态加载文件并渲染成native）[]

在接入我们这个项目之前，你首先要仔细了解一下weex的工作原理以及weex原有的组件、模块等内容，如果你还没有做这些的话，请先查看去了解并学习下列内容点

1. [weex官网](#)
2. [weex官方文档](#)
3. [weex-github](#)

若果你已经了解的以上内容，那么你可以往下了。

我们团队自定义的常用组件:[点这里](#)

我们团队自定义的常用模块:[点这里](#)

了解以上内容后，你就可以往下看了.....

# web接入文档

## 概述

## Other

# android接入文档

## 概述

## Other

# iOS接入文档

简单封装了weex，更方便动态化的使用weex

## 概述

## Usage

iOS使用有四种方式：

1. 从native跳转到我们的DDWeex控制器
2. 直接渲染本地文件作为App的root控制器（多数用于测试）
3. 直接渲染远程文件作为App的root控制器（网络要求比较高，开始渲染时，可能会有些卡顿和闪屏）
4. 通过我们动态引擎的方式去请求入口文件下载到本地并作为App的root控制器

## Other