

FRAMEWORKS

陈磊

2018 年 9 月 17 日

目录

1	介绍	1
2	Vue.js	1
2.1	周边配套	1
2.2	Tips	1
2.3	Compatible	6
2.4	Vue.js TypeScript	6
2.5	Vuex 调用	10
3	React	11
3.1	常用依赖	11
3.2	文件组织	12
3.3	React 与 Redux	12
3.4	create-react-app 工程构建	12
3.5	扩展	16
4	React Native	16
4.1	环境配置	16
4.2	基本命令	17
4.3	打包	17
4.4	入口文件更改	20
4.5	工具/依赖 (dependencies)	20
4.6	调试	21
4.7	工程结构	22

1 介绍	2
4.8 Tips	22
4.9 问题及解决	22
4.10 原理	23
5 Weex	23
5.1 搭建开发环境	23
5.2 Demo	23
5.3 问题及解决	24
6 React Native vs Weex	24
6.1 对比表格	24
6.2 评论摘抄	25
7 Element	26
7.1 组件使用	26
7.2 兼容性	26
8 Electron	26
8.1 技术栈	27
8.2 扩展	28

1 介绍

1. jQuery: Dom 工具.
2. lodash: A modern JavaScript utility library delivering modularity, performance, & extras.

2 Vue.js

Vue.js 开发简单直观, 简单实用的东西通常寿命会比较长.

2.1 周边配套

1. 开发小程序: Meituan-Dianping/mpvue
2. 开发原生 APP: weex

2.2 Tips

2.2.1 本地服务通过 IP 无法访问

1. 方案 1, 更改 package.json 中的命令: `webpack-dev-server --port 3000`
→ `--hot --host 0.0.0.0`。
2. 方案 2, 更改 config/index.js 中 `host: 'localhost'` 为 `host: '0.0.0.0'`。

2.2.2 动态组件加载

场景: 根据不同的条件加载不同的组件, 效果类似 React 中, 根据条件 Return 不同的视图。

```
1 <component :is='ComponentName'></component>
2
3 <!-- 组件需要传参的场景 -->
4 <component :is='ComponentName' :yourPropName="binddingIt"></
   → component>
```

如果需要异步加载组件, 则采用

```
1 data () {
2   return {
3     // 无法异步加载
4     // ComponentName: MyComponent,
5     ComponentName: () => import('@components/dynamic/MyComponent')
6   }
7 }
```

参考 [vuejs-dynamic-async-components-demo](#)

2.2.3 组件内事件添加额外的参数

封装的组件提供的事件已经有返回的数据, 需要添加额外的参数作预处理。

```
1 // myComponent 组件内定义的事件
2 this.$emit('on-change', val);

1 <!-- 使用组件, 关键在于添加 $event -->
2 <myComponent @on-change="myChangeEvent($event, myParams)" />
```

```
1 // 第一个参数为 myComponent 组件内的返回数据，第二个参数为自定义参数
2 myChangeEvent(val, myParams) {
3
4 }
```

2.2.4 watch 对象变化

```
1 watch: {
2   form: {
3     handler(val) {
4       this.$emit('data-change', val);
5     },
6     // here
7     deep: true,
8   },
9 },
```

2.2.5 extend 实现 JS 调用的组件封装

```
1 // MyComponent/index.js
2 import Vue from 'vue';
3 // MyComponent/main.vue 用一般的组件写法编写
4 import mainVue from './main';
5 const ConfirmBoxConstructor = Vue.extend(mainVue);
6 const MyComponent = (options) => {
7   const instance = new ConfirmBoxConstructor({
8     el: document.createElement('div'),
9     // 参数将赋值到 main.vue 中的 data 中，实现配置
10    data: options,
11  });
12
13  document.body.appendChild(instance.$el);
14 };
15
16 MyComponent.myMethod = () => {
17   // define here
18 }
19
20 export default MyComponent;
```

调用组件

```
1 import MyComponent from 'MyComponent'
2 export default {
3   mounted() {
4     const options = {
5       // custom here
6     };
7     MyComponent(options);
8     // 方法调用
9     // MyComponent.myMethod();
10  }
11 }
```

2.2.6 ES6

以下几个 ES6 功能应用于 Vue.js 将获得不错的收益¹, 特别是对于无需构建工具的情况.

1. 箭头函数: 让 this 始终指向到 Vue 实例上.
2. 模板字符串: 应用于 Vue 行内模板, 可以方便换行, 无需用加号链接. 也可以应用于变量套入到字符串中.

```
1 Vue.component({
2   template: `<div>
3     <h1></h1>
4     <p></p>
5   </div>`
6   data: {
7     time: `time: ${Date.now()}`
8   }
9 });
```

3. 模块 (Modules): 应用于声明式的组件 `Vue.component`, 甚至不需要 `webpack` 的支持.

```
1 import component1 from './component1.js';
2 Vue.component('component1', component1);
```

4. 解构赋值: 可应用于只获取需要的值, 减少不必要的赋值, 比如只获取 `Vuex` 中的 `commit` 而不需要 `store`.

¹ANTHONY GORE, 4 Essential ES2015 Features For Vue.js Development, 2018-01-22

```
1 actions: {  
2   increment ({ commit }) {  
3     commit(...);  
4   }  
5 }
```

5. 扩展运算符: 数组和对象等批量导出, 而不需要用循环语句. 比如, 将路由根据功能划分为多个文件, 再用扩展运算符在 index 中合在一起.

2.2.7 组件重新渲染

通过设置 v-if 实现, 从 Dom 中剔除再加入.

```
1 <demo-component v-if="ifShow"></demo-component>
```

2.2.8 绑定数据后添加属性视图未重新渲染

如果存在异步请求, 在数据上添加属性的情况, 需要先预处理好获取的数据, 然后在将其赋值到 data 中变量. 数据绑定后, 再添加属性, 不会触发界面渲染.

```
1 API.getSomething().then(res => {  
2   // 1. 先添加属性  
3   // handle 表示对数据的处理, 包括对象中属性的添加  
4   const handledRes = handle(res);  
5   // 2. 然后绑定到 data 中的变量  
6   this.varInDate = handledRes;  
7 });
```

2.2.9 全局引入 SCSS 变量文件²

场景: 将常用的变量存储到 vars.scss, 应用变量时需要在每个需要的地方 import.

1. npm install sass-resources-loader --save-dev
2. 更改 build/webpack.base.conf.js, 适用于 vue-cli.

²https://www.reddit.com/r/vuejs/comments/7o663j/sassscss_in_vue_where_to_store_variables/?st=JC9T45PB&sh=4f87ec9d

```
1 {
2   test: /\.vue$/,
3   loader: 'vue-loader',
4   options: {
5     loaders: {
6       sass: ['vue-style-loader', 'css-loader', {
7         loader: 'sass-loader',
8         options: {
9           indentedSyntax: true
10        }
11      }, {
12        loader: 'sass-resources-loader',
13        options: {
14          resources: path.resolve(__dirname, "./styles/vars.
15                                ↪ scss")
16        }
17      }],
18      scss: ['vue-style-loader', 'css-loader', 'sass-loader', {
19        loader: 'sass-resources-loader',
20        options: {
21          resources: path.resolve(__dirname, "./styles/vars.
22                                ↪ scss")
23        }
24      }],
25    },
26    // other vue-loader options go here
27  }
28 }
```

2.3 Compatible

2.3.1 IE vuex requires a promise polyfill in this browser

```
1 npm install --save-dev babel-polyfill
```

```
1 // build/webpack.base.conf.js
2 entry: {
3   app: [
4     'babel-polyfill',
5     './src/main.js'
```

```
6 ]  
7 }
```

vuex requires a promise polyfill in this browser

2.4 Vue.js TypeScript

本节内容基于 @vue/cli v3.0.0-rc.4 完成。

2.4.1 项目创建

```
1 npm i -g @vue/cli  
2 vue create typescript-demo  
3 # 自定义配置  
4 # babel, Router, Vuex, TypeScript, TSLint  
5  
6 # 本地启动  
7 npm run serve
```

TypeScript-Vue-Starter 也提供了一个模版工程，比较基础，实际应用过程都会有遇到很多其他问题。

2.4.2 实例对应写法

```
1 <script lang="ts">  
2 import { Vue, Component Prop, Watch } from 'vue-property-decorator  
  ↪';  
3 import Hello from '@components/Hello.vue';  
4  
5 @Component({  
6   components: {  
7     'com-hello': Hello,  
8   }  
9 })  
10 export default class Demo extends Vue {  
11   // props  
12   @Prop() inVar1!: string;  
13  
14   // data  
15   private var1!: string;  
16   private var2: string = 'var2';
```



```
17 private var3!: boolean;
18 private var4!: number[];
19
20 // created
21 private created(): void {
22     // excute something
23 }
24
25 // mounted
26 private mounted(): void {
27     // excute something
28 }
29
30 // watch
31 @Watch('var1')
32 private onVar1Change(): void {
33     this.var2 = this.var1;
34 }
35
36 // computed
37 get somthing(): string {
38     return this.var2;
39 }
40
41 // methods
42 private func1(): void {
43     // excute something
44 }
45 }
46 </script>
```

2.4.3 Vuex 定义

Vuex 的使用思路：

1. 不同功能放到不同的模块，并开启命名空间。
2. 所有的操作和命名空间名称作为变量存储在一个 types 的文件中。
3. 每一个模块都包含一个 clear 的清除操作，并在根模块下加一个清除操作，可以清楚所有的模块状态。

模块定义：

```
1 // store/modules/moduleA.ts
2 import { Commit, Dispatch } from 'vuex';
3 import * as TYPES from '@store/types';
4
5 // state 类型定义
6 interface State {
7   var1: string;
8 }
9
10 // state
11 const initState: State = {
12   var1: '',
13 }
14
15 // getters
16 const getters = {};
17
18 // mutations
19 const mutations = {
20   [TYPES.SET_VAR1](state: State, payload: string) {
21     state.var1 = payload;
22   },
23
24   [TYPES.MODULEA_CLEAR](state: State) {
25     state.var1 = '';
26   }
27 }
28
29 const actions = {
30   [TYPES.SET_VAR1](context: {commit: Commit, dispatch: Dispatch},
31     ↪ var1: string) {
32     context.commit(TYPES.SET_VAR1, var1);
33   },
34
35   [TYPES.MODULEA_CLEAR](context: {commit: Commit, dispatch: Dispatch}
36     ↪ ) {
37     context.commit(TYPES.MODULEA_CLEAR);
38   },
39 }
```

```
39 export {  
40   State,  
41 }  
42  
43 export default {  
44   namespace: true,  
45   state: initState,  
46   getters,  
47   actions,  
48   mutations,  
49 }
```

根模块定义:

```
1 // store/index.ts  
2 import Vue from 'vue';  
3 import Vuex, { Dispatch, Store } from 'vuex';  
4  
5 import moduleA, { State as moduleAState } from '@store/modules/  
  ↳ moduleA';  
6  
7 Vue.use(Vuex);  
8  
9 // 根模块定义所有状态类型  
10 interface RootState {  
11   [TYPES.NAMESPACE_MODULEA]: moduleAState,  
12 }  
13  
14 const actions = {  
15   clear(context: {dispatch: Dispatch}) {  
16     context.dispatch(`${TYPES.NAMESPACE_MODULEA}/${TYPES.  
      ↳ MODULEA_CLEAR}`, {}, { root: true });  
17   }  
18 }  
19  
20 const store: Store<RootState> = new Vuex.Store({  
21   moudles: {  
22     [TYPES.NAMESPACE_MODULEA]: moduleA,  
23   }  
24   actions,  
25 });  
26
```

```
27 export default store;
```

2.5 Vuex 调用

```
1 <script lang="ts">
2 import { Vue, Component Prop } from 'vue-property-decorator';
3 import { Action, State } from 'vuex-class'
4
5 @Component
6 export default class Demo extends Vue {
7   // state 映射
8   @State('这里是state名', {namespace: '这里是命名空间名称'}) private
      ↪ stateIsLogin!: boolean;
9   // action 映射
10  @Action('这里是action名称', {namespace: '这里是命名空间名称'})
      ↪ private actionLogin!: () => Promise<any>;
11 }
12 </script>
```

2.5.1 什么时候应用 class

有一些工具，需要单独封装，如果这个工具是一个工具集合，并且内部存在一些逻辑依赖，可采用 class 书写。比如，同一个项目的请求配置通常是一样的，这个时候可以将 axios 封装成 class，内部设置默认的配置，并提供可外部配置的方式。axios 的拦截、请求都依赖同一套配置，通过 class 可在构造器输入配置，屏蔽内部复杂性，并实现可配置。反之，类似接口层，各个 api 之间并无关联，则无需采用 class。

3 React

```
1 npm install -g create-react-app
2 create-react-app my-app
```

React 与 Vue 类似，主要专注于视图层，结合其他的库实现扩展，比如路由、数据状态维护等。

3.1 常用依赖

1. 路由: React Router
2. 类型检查: prop-types
3. 数据管理: React Redux

3.1.1 UI

- element-react
- ant-design

3.1.2 优化

- reselector
- immutable.js
- seamless-immutable.js

3.2 文件组织

3.3 React 与 Redux

引入 Redux 用于应用状态维护，为了方便与 React 结合，因此会同时引入 React Redux。用户在视图层（React）操作，触发事件（action），事件触发 reducer，在 reducer 中决定如何返回新的状态，状态的更新触发视图的更新。从而实现了单向的数据流。

reducer 和 action 分文件夹放置，reducer 通过 combineReducers 合并，然后再通过 createStore 创建 store。store 通过 Provider 放置在 App 的顶层元素，并分发给所有子组件。对每个具体的功能，分别按需引入需要的 state（reducer 中定义）和 action，并用 connect 将此两项和视图层（react 实现）连接起来，实现完整的组件。由此，实现了数据、视图、事件之间的分离，通过 connect 实现连接。

reducer 是什么呢？reducer 按照 Redux 的输入书写。每次输入确定的数据，通过 reducer 处理，能够得到固定的输出。也就是说 reducer 中不包含随机因素或者环境因素，因此称之为纯函数。

3.4 create-react-app 工程构建

需求：

1. 不暴露 create-react-app 配置项（并支持 Sass、支持热加载）
2. 区分环境变量

3.4.1 不暴露 create-react-app 配置项

Sass 支持需安装 node-sass 和 sass-loader。

```
1 // config-overrides.js
2 const { injectBabelPlugin, getLoader } = require('react-app-rewired
  ↪ ');
3 const rewireReactHotLoader = require('react-app-rewire-hot-loader')
  ↪ ;
4
5
6 // load sass file
7 const autoprefixer = require('autoprefixer');
8 const fileLoaderMatcher = function (rule) {
9   return rule.loader && rule.loader.indexOf('file-loader') !== -1;
10 };
11 const rewireSass = (config, env) => {
12
13   // customize theme
14   config.module.rules[1].oneOf.unshift(
15     {
16       test: /\.scss$/,
17       use: [
18         require.resolve('style-loader'),
19         require.resolve('css-loader'),
20         {
21           loader: require.resolve('postcss-loader'),
22           options: {
23             // Necessary for external CSS imports to work
24             // https://github.com/facebookincubator/create-react-app/
25             ↪ issues/2677
26             ident: 'postcss',
27             plugins: () => [
28               require('postcss-flexbugs-fixes'),
29               autoprefixer({
30                 browsers: [
31                   '>1%',
32                   'last 4 versions',
```

```
32         'Firefox ESR',
33         'not ie < 9', // React doesn't support IE8 anyway
34     ],
35     flexbox: 'no-2009',
36 },
37 ],
38 },
39 },
40 {
41     loader: require.resolve('sass-loader'),
42     options: {
43         // theme vars, also can use theme.js instead of this.
44         modifyVars: { "@brand-primary": "#1DA57A" },
45     },
46 },
47 ]
48 }
49 );
50
51 // css-modules
52 config.module.rules[1].oneOf.unshift(
53     {
54         test: /\.css$/,
55         exclude: /node_modules|antd-mobile\.css/,
56         use: [
57             require.resolve('style-loader'),
58             {
59                 loader: require.resolve('css-loader'),
60                 options: {
61                     modules: true,
62                     importLoaders: 1,
63                     localIdentName: '[local]__[hash:base64:5]'
64                 },
65             },
66             {
67                 loader: require.resolve('postcss-loader'),
68                 options: {
69                     // Necessary for external CSS imports to work
70                     // https://github.com/facebookincubator/create-react-app/
71                     // → issues/2677
```

```
71     ident: 'postcss',
72     plugins: () => [
73       require('postcss-flexbugs-fixes'),
74       autoprefixer({
75         browsers: [
76           '>1%',
77           'last 4 versions',
78           'Firefox ESR',
79           'not ie < 9', // React doesn't support IE8 anyway
80         ],
81         flexbox: 'no-2009',
82       }),
83     ],
84   },
85 ],
86 ]
87 }
88 );
89
90 // file-loader exclude
91 let l = getLoader(config.module.rules, fileLoaderMatcher);
92 l.exclude.push(/\.scss$/);
93
94 return config;
95 };
96
97 module.exports = function override(config, env) {
98   // do stuff with the webpack config...
99   config = injectBabelPlugin(['import', { libraryName: 'antd-mobile
100     ↪ ', style: 'css' }], config);
101   config = rewireSass(config, env);
102   config = rewireReactHotLoader(config, env);
103   return config;
104 };
```

1. 覆盖配置: <https://mobile.ant.design/docs/react/use-with-create-react-app-cn>
2. 热加载: react-app-rewire-hot-loader
3. Sass: antd-mobile-samples

3.4.2 环境变量

配置变量以 `REACT_APP_` 开头，例如 `REACT_APP_ENVIRONMENT=PRODUCTION`。

```
1 # 默认配置
2 .env
3
4 # 开发配置
5 .env.development
6
7 # 生产配置
8 .env.production
```

1. 环境变量配置: Adding Development Environment Variables In .env

3.5 扩展

1. 关于 actionTypes, actions, reducer 文件分割的提议:GitHub, erikras/ducks-modular-redux
2. React 生命周期及方法图:wojtekmaj/react-lifecycle-methods-diagram

4 React Native

1. 主页: <https://facebook.github.io/react-native>
2. GitHub: <https://github.com/facebook/react-native>
3. 示例项目: amazing-react-projects
4. Demo Project: react-native

4.1 环境配置

4.1.1 系统环境

1. 安装 nodejs.
2. `npm install -g react-native-cli`.

Android

1. JDK (并配置环境变量)
2. 安装 Android Studio <http://www.android-studio.org>
3. 通过 SDK Manager 下载 SDK, 并配置环境变量.

```
1 REM set var
2 set ANDROID_HOME=C:\Users\chen1\AppData\Local\Android\Sdk
3
4 REM set Android home path
5 setx /m ANDROID_HOME "%ANDROID_HOME%"
6
7 REM set path
8 setx /m path "%path%;%ANDROID_HOME%\tools;%ANDROID_HOME%\platform-
  ↪ tools;"
```

iOS

1. App Store 安装 XCode.
2. 其他工具安装

```
1 brew install node
2 brew install watchman
3 npm install -g react-native-cli
```

4.1.2 编辑器

1. Visual Studio Code. 安装扩展 React Native Tools 用于调试.
2. Atom. 安装nuclide.

4.1.3 参考

1. <https://facebook.github.io/react-native/docs/getting-started.html>

4.2 基本命令

1. 新建工程: `react-native init demo-project`.
2. Android 运行: `react-native run-android`.
3. iOS 运行: `react-native run-ios`.

新建工程后首先 `npm install` 安装依赖. 示例项目 python 和 node-gyp-bin 相关错误可以尝试先执行 `yarn add node-sass` 或者 `npm install -f node-
↪ sass` (<https://github.com/sass/node-sass/issues/1980>).

4.3 打包

4.3.1 Android 打包

生成签名密钥

```
1 $ keytool -genkey -v -keystore my-release-key.keystore -alias my-  
    ↪ key-alias -keyalg RSA -keysize 2048 -validity 10000  
2 Enter keystore password:  
3 Keystore password is too short - must be at least 6 characters  
4 Enter keystore password: chenlei  
5 Re-enter new password: chenlei  
6 What is your first and last name?  
7 [Unknown]: HereChen  
8 What is the name of your organizational unit?  
9 [Unknown]: HereChen  
10 What is the name of your organization?  
11 [Unknown]: HereChen  
12 What is the name of your City or Locality?  
13 [Unknown]: Chengdu  
14 What is the name of your State or Province?  
15 [Unknown]: Sichuan  
16 What is the two-letter country code for this unit?  
17 [Unknown]: 51  
18 Is CN=HereChen, OU=HereChen, O=HereChen, L=Chengdu, ST=Sichuan, C  
    ↪ =51 correct?  
19 [no]: yes  
20  
21 Generating 2,048 bit RSA key pair and self-signed certificate (  
    ↪ SHA256withRSA) with a validity of 10,000 days  
22     for: CN=HereChen, OU=HereChen, O=HereChen, L=Chengdu, ST=  
    ↪ Sichuan, C=51  
23 Enter key password for <my-key-alias>  
24     (RETURN if same as keystore password):  
25 [Storing my-release-key.keystore]
```

gradle 设置

1. my-release-key.keystore 文件放到工程 android/app 文件夹下.
2. 编辑 android/app/gradle.properties, 添加如下信息.

```
1 MYAPP_RELEASE_STORE_FILE=my-release-key.keystore  
2 MYAPP_RELEASE_KEY_ALIAS=my-key-alias
```

```
3 MYAPP_RELEASE_STORE_PASSWORD=chenlei
4 MYAPP_RELEASE_KEY_PASSWORD=chenlei
```

3. 编辑 `android/app/build.gradle`, 添加如下信息.

```
1 ...
2 android {
3     ...
4     defaultConfig { ... }
5     signingConfigs {
6         release {
7             storeFile file(MYAPP_RELEASE_STORE_FILE)
8             storePassword MYAPP_RELEASE_STORE_PASSWORD
9             keyAlias MYAPP_RELEASE_KEY_ALIAS
10            keyPassword MYAPP_RELEASE_KEY_PASSWORD
11        }
12    }
13    buildTypes {
14        release {
15            ...
16            signingConfig signingConfigs.release
17        }
18    }
19 }
20 ...
```

生成 apk

```
1 cd android && ./gradlew assembleRelease
```

打包后在 `android/app/build/outputs/apk/app-release.apk`.

安装 apk 方式

1. Genymotion 可以拖拽 apk 进行安装.
2. `adb install app-release.apk` 安装.

如果报签名错误, 可先卸载之前的 debug 版本.

4.3.2 iOS 打包

iOS 版本编译需要在 Mac 上进行.

签名 没有证书....

生成 ipa 以下流程以 Xcode 9 为例.

1. 打开工程: Xcode 打开 ios 文件夹下 *.xcodproj 文件 (工程).
 2. 选择编译机型: Xcode 虚拟机选择栏中选择 Generic iOS Device.
 3. 编译设置: Xcode -> Product -> Scheme -> Edit Scheme -> Run -> Info -> Build Configuration 选择 Release
 4. JS 改为离线 (打包进 APP)???
- TODO: 命令行打包

4.3.3 参考

1. Generating Signed APK, Facebook Open Source
2. 打包 APK, React Native 中文网
3. ReactNative 之 Android 打包 APK 方法(趟坑过程), ZPengs, 2017.02.09, 简书

4.4 入口文件更改

从 0.49 开始, 只有一个入口, 不区分 ios 和 android. <https://github.com/facebook/react-native/releases/tag/v0.49.0>

React Native CLI 新建的工程, 默认入口是 index.js. 在 android\app\
↪ build.gradle 中更改入口.

```
1 project.ext.react = [  
2     entryFile: "index.android.js"  
3 ]
```

对应更改 android\app\src\main\java\com**\MainApplication.java.

```
1 protected String getJSMainModuleName() {  
2     return "index.android";  
3 }
```

4.5 工具/依赖 (dependencies)

4.5.1 导航

<https://facebook.github.io/react-native/docs/navigation.html>

1. react-navigation 提供了常用的导航方式 (Stack, Tab, Drawer), 推荐.
2. NavigatorIOS 为内建的导航, 仅在 IOS 上可用.

4.5.2 UI

尚未找到两端 (Web, Native) 完整好用的 UI, 若后端采用 ant-design 可用 ant-design-mobile.

1. ant-design-mobile 每个组件是否支持 Native 有说明.
2. react-native-elements
3. NativeBase

4.5.3 HTTP 请求

<https://facebook.github.io/react-native/docs/network.html>

1. fetch 为内建接口.
2. **axios** 为使用较广泛的第三方请求库, 推荐使用.

4.6 调试

<https://facebook.github.io/react-native/docs/debugging.html>

根据提示, 可以菜单按钮选择重新加载或热加载. Android 可摇晃手机显示菜单.

4.6.1 虚拟机

1. Genymotion, 需要先注册, 然后选择 for personal 使用. 如果系统开启了 Hyper-V, 需要先关闭.
2. Android Studio 内建虚拟机, 同样需要关闭 Hyper-V.
3. Visual Studio Emulator for Android 需要开启 Hyper-V.

4.6.2 调试工具: Chrome

1. Remote JS Debugging 开启 JS 调试.
2. 浏览器端进去 <http://localhost:8081/debugger-ui/>, 并开启开发工具.
3. 可在 Sources 中设置断点或者代码中写入 debugger.

4.6.3 调试工具: VSCode

1. 安装扩展: React Native Tools.
2. F5 生成 launch.json 文件.
3. 进入调试菜单 (Ctrl + Shift + D), 选择 Debug Android.
4. 设置断点或者写入 `debugger` 开始调试, 在 output 栏输出.

4.6.4 HTTP 调试问题备注

应用 Fiddler 调试 HTTP, 模拟器设置了代理后, APP 无法热加载 JS bundle. 目前只有用 Chrome 或者断点的方式来调试.

4.7 工程结构

4.7.1 结构

```
1 android/      # Android 工程
2 ios/          # IOS 工程
3 src/          # 开发前端资源
4 -- assets/    # 静态资源
5 -- components/ # 组件
6 -- api/       # 接口
7 -- route/     # 导航(路由)
8 -- config/    # 常量配置
9 -- pages/     # 页面/功能
10 -- utils/    # 常用工具
11 -- reducers  相关
12 -- index.js  # APP 入口
13 index.js     # 入口文件
```

4.7.2 参考


1. Organizing a React Native Project
2. React native project setup—a better folder structure

4.8 Tips

1. Android 查看当前的 Android 设备 `adb devices`.

2. Android 虚拟机: Ctrl + M 打开菜单 (Android Studio 自带虚拟机没有菜单和摇晃手机, 可以这种方式打开菜单).
3. iPhone 虚拟机啊重新加载资源: command + R.

4.9 问题及解决

1. VSCode Debug 无法加载的情况, 首先重启 VSCode 再启动项目.
2. 添加antd-mobile后报错, 无法解析 react-dom, 依赖中加入react-dom并安装即可.
3. 集成react-native-navigation需要注意 Android SDK 版本, 版本过低可能出现编译错误 (Error:Error retrieving parent for item: No resource  found).

4.10 原理

1. React Native 将代码由 JSX 转化为 JS 组件, 启动过程中利用 instantiateReactComponent 将 ReactElement 转化为复合组件 ReactCompositeComponent 与元组件 ReactNativeBaseComponent, 利用 ReactReconciler 对他们进行渲染³。
2. UIManager.js 利用 C++ 层的 Instance.cpp 将 UI 信息传递给 UIManagerModule.java, 并利用 UIManagerModule.java 构建 UI⁴。
3. UIManagerModule.java 接收到 UI 信息后, 将 UI 的操作封装成对应的 Action, 放在队列中等待执行。各种 UI 的操作, 例如创建、销毁、更新等便在队列里完成, UI 最终得以渲染在屏幕上⁵。

5 Weex

1. 主页: <http://weex.apache.org>
2. GitHub: <https://github.com/apache/incubator-weex/>
问题: 入口在哪儿?

案例

1. 网易严选
2. 点我达骑手 Weex 最佳实践

³ReactNative 源码篇: 渲染原理

⁴ReactNative 源码篇: 渲染原理

⁵ReactNative 源码篇: 渲染原理

3. weexteam/weex-hackernews

5.1 搭建开发环境

```
1 npm install -g weex-toolkit
```

5.2 Demo

web

```
1 weex create weex
2 cd weex
3 npm install
4 npm run dev & npm run serve
```

命令

<https://github.com/weexteam/weex-pack>

```
1 # debug
2 weex debug
3
4 # add platform
5 weex platform add android
6 weex platform add ios
7
8 # run
9 weex run web
10 weex run android
11 weex run ios
12
13 # build
14 weex build web
```

5.3 问题及解决

1. <https://maven.google.com/> 链接不上, 更改\platforms\android\build.gradle
→ 文件, 换成 <https://dl.google.com/dl/android/maven2/>。

2. adb: failed to stat app/build/outputs/apk/playground.apk: No such file
 ↪ or directory, 替换 platforms/android/app/build.gradle 文件中的
 weex-app.apk 为 playground.apk.
3. weex debug 报错可先安装 `npm install -g weex-devtool`.

6 React Native vs Weex

6.1 对比表格

属性	React Native	Weex
开源时间	2015/03	2016/06
开源企业	Facebook	Alibaba
协议	BSD 3-clause	Apache License 2.0
主页标语	Build native mobile apps using JavaScript and React	A framework for building Mobile cross-paltform UIs
核心理念	Learn Once, Write Anywhere	Write Once, Run Everywhere
前端框架	React	Vue.js
JS Engine	JavaScriptCore(iOS/Android)	JavaScriptCore(iOS) / v8(Android)
三端开发	部分组件需要区分平台开发	强调三端统一
代码写法	JSX(JavaScript + XML)	Web 写法
调试	虚拟机	可用 Chrome 查看效果
社区支持	社区活跃, 有多个流行产品的实践	目前, 开发者主要在国内, 没有太多的实践案例
优势	生态好, 第三方依赖多, 有可借鉴的经验	基于 Vue.js, 上手快, 能更好的保证三端一致

以下参考都是 2016 年文章.

1. compare weex to react native
2. Weex 简介
3. Weex & React Native

6.2 评论摘抄

After a few days of experimentation, I realized Weex and its documentation were not yet developed enough to for us to use to deliver top-quality apps. This was my experience with Weex. Sam Landfried, 2017.10.20, Is VueJS' Weex a Suitable Alternative to React Native?

7 Element

<https://github.com/ElemeFE/element>

7.1 组件使用

7.1.1 自定义表单校验

```
1 export default {
2   data: function () {
3     var checkVars = function (rule, value, callback) {
4       if (!value) {
5         callback(new Error('不能为空'));
6       } else {
7         callback();
8       }
9     };
10    return {
11      rules: {
12        vars: [{
13          required: true,
14          trigger: 'change',
15          validator: checkVars
16        }]
17      }
18    };
19  }
20 }
```

```
18   }  
19   }  
20 }
```

7.2 兼容性

7.2.1 IE 图标不显示

可用文字替代伪元素中的内容。

8 Electron

Electron 构建应用,主要包括两部分:一是构建 Web App;二是 Electron 提供扩展能力(内置了 Node,可用 node 的模块),Web App 实现本地交互能力。构建 Electron 应用工程,可首先构建 Web App 工程,稍作修改适应客户端最后打包的需求。

8.1 技术栈

应用 (TypeScript + React) + 打包 (electron-builder)。

8.1.1 准备

1. Electron 支持 TypeScript, Announcing TypeScript support in Electron, 模板工程参考 electron-quick-start-typescript。
2. React TypeScript 模板工程参考 TypeScript-React-Starter。
3. 打包工具 electron-builder。

思路,首先建立一个 Web 应用工程(通常为单页应用,无需管理多窗口),然后建一个桌面应用入口文件,入口文件加载 Web App 应用,最后用工具将 electron 和 Web 应用打包在一起。结合以上的两个模板工程,优先构建 Web 应用,然后加入 Electron 相关依赖的脚本。

8.1.2 工程搭建

创建工程流程,可先用 create-react-app 创建 React 应用工程,然后作调整,并加入 Electron 相关依赖的脚本。

1. 创建 React 工程

```
1 create-react-app my-app --scripts-version=react-scripts-ts
```

2. 更改 tsconfig: 解决 main.ts 编译问题

```
1 {  
2   "compilerOptions": {  
3     "module": "commonjs",  
4     /* 关闭窗口时设置为 null 禁用 null 检查 */  
5     "strictNullChecks": false  
6   }  
7 }
```

3. 静态资源路径需要改为相对路径。
4. 将 electron-quick-start-typescript 中的 main.ts 移到当前工程 src 下。
注意 main.ts 编译后 index.html 以及静态资源路径问题。

8.2 扩展

1. Technical Differences Between Electron and NW.js(formerly node-webkit)