**react**

**一、VS Code插件**

1. 代码提示类插件

1.1 Reactjs code snippets

1.2 React Redux ES6 Snippets

1.3 React-Native/React/Redux snippets for es6/es7

1.4 JavaScript (ES6) code snippets(es6代码片段)

1.5 Typescript React code snippets(这是tsx的react组件片段)

2. 美化类插件

2.1 One Dark Pro(atom风格主题)

2.2 vscode-icons(文件图标)

3. 其他实用类插件

3.1 Beautify css/sass/scss/less(样式代码格式化)

3.2 npm Intellisense(对package.json内中的依赖包的名称提示)

3.3 Path Intellisense(文件路径补全)

3.4 cssrem(px转换为rem)

3.5 CSS Modules(对使用了css modules的jsx标签的类名补全和跳转到定义位置)

4. React/Redux/react-router Snippets

**二、VS Code快捷键**

扩展搜索 ES7 React/Redux/GraphQL/React-Native snippetsd或者React-Native/React/Redux snippets for es6/es7 并安装

**rfc**回车-->生成无状态组件

**rcc**回车-->生成有状态组件

**rcep**→快速创建一个组件（使用extends方式，包含PropTypes）

**rconst** -->快速创建一个 constuctor

**三、React中的生态圈**

ReactJS+ReactNative+ReactVR(360)+Redux

第三方:reactNavigation reactRouter

https://github.com/enaqx/awesome-react

**四、ReactJS（3w+1h）**

what?

reactjs是由Facebook所推出的一个js的库

when?

用在数据操作比较频繁的场景，实现高性能的ui层

why?

①性能瓶颈

vdom(virtual dom)

②代码的开发和维护成本

单向数据流

how?

引入js文件

**五、搭建项目——基于cli**

1、全局安装create-react-app脚手架

npm install -g create-react-app

2、创建项目，安装依赖

create-react-app my-react-app

3、进入项目

cd my-react-app

4、启动项目

npm start

**安装时报错：**

reate-react-app my-app 报错 “npm ERR! Unexpected end of JSON input while parsing near“

查了资料都说清npm缓存（试了之后并没有用）：

npm cache clean --force

解决：把淘宝镜像切回npm就好了

npm config set registry [http://registry.cnpmjs.org](http://registry.cnpmjs.org/)

**六、react打包错误**

index.html打开时空白页：

也就是说默认为你的是一个web服务器,所以生成的index.html文件中的路径如下:  
<script type="text/javascript" src="/static/js/main.a0b7d8d3.js">  
所以当你打开网页的时候,打开开发者模式(F12),你会发现提示这个文件找不到,因为路径有问题!  
正确的路径应该是:  
<script type="text/javascript" src="./static/js/main.a0b7d8d3.js">

解决：修改package.json文件，添加homepage属性：

{

"homepage": "./",

}

**1、babel**

babel是一个js的编译器，可以将es6等高级语法 、jsx转换为浏览器能识别的版本

<script type='text/babel'></script>

**2、方法**

//React和ReactDOM导入的名称必须这么写

import React from 'react';

import ReactDOM from 'react-dom';

ReactDOM.render(

<h1>123</h1>,

document.getElementById('example'))

**3、五大核心概念**

**3.1 jsx （react基石）**

jsx:JavaScriptXML 不是一种新的语言，只是js语法的一种扩展，允许在js中直接写标签

jsx的优缺点：

使用jsx封装、数据处理很方便

增加了数据和视图的耦合度

是否必须使用jsx?

推荐使用

**语法:**

①遇到**<，**如果首字母是小写，会按照html来解析；如果首字母是大写，会按照组件来进行渲染

②遇到**{**，会按照js的表达式来处理

注意事项：在一个位置要渲染多个元素，要记得放在一个顶层容器中

**3.2 component (视图)**

组件:可被反复使用的，带有特定功能的视图

vue：

\*\*\*.vue

angular：

ng g component demo01

<app-demo01></app-demo01>

react的组件的创建和使用:

方法一：React.createClass

var MyComponent = React.createClass({

render:function(){

return <h1>123</h1>

}

});

<MyComponent></MyComponent>

注意事项:

①组件类的命名要遵循全驼峰式(第一个字母大写)

②组件类的render方法 在渲染元素时候，第一个标签不要直接换行

③如果渲染多个标签，放在一个容器中

④每一个标签都得有开始和结束标记<input/>

方法二：使用ES6语法的class

import React from 'react';

import PropTypes from 'prop-types'; //react中判断类型

export default class InputControlES6 extends React.Component {

constructor(props) {

super(props);

this.state = {

text: props.initialValue || 'placeholder'

};

// 使用箭头函数直接替换函数在类中的声明，不然需要绑定this，像这样

this.handleChange = this.handleChange.bind(this);

}

//方法使用箭头函数不用绑定this

handleChange(event)=> {

this.setState({

text: event.target.value //双向绑定

});

}

render() {

return (

<div>

Type something:

<input onChange={ ()=>this.handleChange } value={this.state.text}/>

</div>

);

}

}

//设置该组件传入参数的类型

InputControlES6.propTypes = {

initialValue: React.PropTypes.string

};

//设置该组件传入参数的默认值

InputControlES6.defaultProps = {

initialValue: ''

};

**3.3重点 props (父-->子 /子-->父)**

回顾:

vue: props down

<any myTitle='123'></any>

Vue.component('',{

props:['myTitle']

})

Angular:props down

<any myTitle='123'></any>

import {Input} from '@angular/core'

@Input() myTitle = "" //this.myTitle

react:

(1)基础用法(this.props 父-->子)

<any myTitle='123'></any>

this.props.myTitle

如果想要指定子标记，而且还能显示

this.props.children

解决方案:

React.Children.map(

this.props.children,

(value)=>{

return value

}

)

(2)通信（子-->父）

共识:

父组件在调用子组件时，是可以通过自定义属性来完成方法的传递

子-->父的通信工作：

①父组件准备一个有参数的方法

②将此方法传递给子组件

③子组件在调用父组件通过自定义属性传来的方法时，可以通过方法的参数将值传递给父组件

事件处理技巧:

①btn处理单击事件时，如果需要自定义传值:

|  |
| --- |
| var MyBtn = React.createClass({  handleClick:function(msg){  console.log('btn is clicked',msg) //==>打印出10  },  render:function(){  return <div>  <button onClick={()=>this.handleClick(10)}>clickMe</button>  </div>  }  }) |

②动态样式处理

vue>>>:style :class

angular>>>ngStyle ngClass

react:

<h1 style={{ opacity : \*\* }}></h1>

|  |
| --- |
| <script type="text/babel">  var MyButton = React.createClass({  handleClick:function(){  console.log("btn is clicked")  this.props.func('zhangsan') //父组件中属性func所传来的方法  },  render:function(){  return <button onClick={this.handleClick} > //绑定事件  { this.props.btnName } //父组件中属性btnName所传来的字符串  </button>  }  })  var MyCart = React.createClass({  login:function(uname){ //设置形参  console.log(uname+"登录成功")  },  register:function(){  console.log("注册成功")  },  render:function(){  return <div>  <h2>购物车</h2>  <MyButton func={ this.login } btnName="登录"></MyButton>  <MyButton func={ this.register } btnName="注册"></MyButton>  </div>  }  })  ReactDOM.render(  <MyCart></MyCart>,  document.getElementById('example')  )  </script> |

练习: demo14\_lianxi.html

①实现一个组件类 Test

②在这个组件类中 渲染一个h1,button

③点击button，启动定时器，将h1的透明度

0~0.1~0.2~。。。1

|  |
| --- |
| <script type='text/babel'>  //封装组件  var Test = React.createClass({  //初始化状态  getInitialState:function(){  return {opacityValue:0}  },  handleClick:function(){  //修改一个数值，这个值能够影响h1的样式的透明度  setInterval(()=>{  var nowValue = this.state.opacityValue;  nowValue+=0.1  //边界值的判断  if(nowValue>1){  nowValue = 0  }  //写操作  this.setState({opacityValue:nowValue})  },100)  },  render:function(){  return <div>  <h1 style={{opacity:this.state.opacityValue}}>测试</h1>  <button onClick={this.handleClick}>  clickMe  </button>  </div>  }  })  //调用组件  ReactDOM.render(  <Test></Test>,  document.getElementById('example')  )  </script> |

**3.4 ref(父主动的到子组件取值)**

使用场景:

①直接找到一个组件中的某一个元素

②找到子组件的实例对象（父需要主动的到子组件中获取数据）

基础用法:

<any ref="myInput"></any>

this.ref**s**.myInput.value/...

|  |
| --- |
| <script type="text/babel">  var Test = React.createClass({  count:10,  handleClick:function(){  //ref的形式获取输入框的值  console.log(this.refs.myInput.value)  },  render:function(){  return <div>  <input type='text' ref="myInput"/>  {/\*在点击按钮时，通过ref的形式来得到输入框的值 \*/}  <button onClick={this.handleClick} >clickMe</button>  </div>  }  })  var MyComponent = React.createClass({  getInfo:function(){  console.log(this.refs.myTest.count) //父组件主动获取子组件的值  },  render:function(){  return <div>  <h2>这是MyComponent</h2>  <Test ref='myTest'></Test>  <button onClick={this.getInfo}>获取子组件信息</button>  </div>  }  })  ReactDOM.render(  <MyComponent></MyComponent>,  document.getElementById('example')  )  </script> |

**3.5 state(绑定/3个操作数据的方式)**

react中是靠state来完成数据的定义、读和写操作，

①state用在视图中才会有绑定的效果

②3个基本操作:

初始化：

getInitialState:function(){

return {count:0,page:1,list:[]}

}

读：

this.state.count

写：

var newCount=this.state.count

newCount++

this.setState({

count : newCount,

page : 2

})

this.setState({ count:newCount }, ()=>{

//状态写成功之后的回调函数(先读后写)

})

练习:demo12-lianxi.html (:- 17:15)

①实现一个组件类，组件只渲染一个button

②点击button，button的文本变成'开'/'关'

|  |
| --- |
| var MyButton = React.createClass({  //初始化状态  getInitialState:function(){  return {isOn:true}  },  handleClick:function(){  var nowStatus = this.state.isOn  this.setState({isOn:!nowStatus})  },  render:function(){  return <button onClick={this.handleClick}>  {this.state.isOn?'开':'关'}  </button>  }  }) |

**4、路由**

安装react-router-dom：npm install react-router-dom --save-dev

在views中添加home和detail页：

|  |
| --- |
| import React from 'react'  import { Link } from 'react-router-dom'  export default class Home extends React.Component {  render() {  return (  <div>  <h3>我是home</h3>  <a href="#/detail"> 去detail</a>  {/\* ①传递参数 \*/}  {/\* <a href="#/detail/123"> 去detail</a> \*/}  {/\* 通过Link跳转 \*/}  <Link to="/about"> 去about</Link>  </div>  )  }  } |
| import React from 'react'  import { Link } from 'react-router-dom'  export default class Home extends React.Component {  constructor(props) {  super(props)  this.state = {}  }  render() {  return (  <div>  <h3>我是detail</h3>  {/\* <a href="#/"> 去home </a> \*/}  <Link to="/"> 去home </Link>  </div>  )  }  componentDidMount() {  // ①获取通过路由:id传过来的参数  // console.log(this.props.match.params)  // ②接收隐式传参  console.log(this.props.history.location.state)  }  } |
| import React, { Component } from 'react'  import PropTypes from 'prop-types'  export class about extends Component {  constructor(props) {  super(props)  this.state = {}  }  static propTypes = {}  render() {  return (  <div>  <h3>我是about</h3>  <div>  <span onClick={() => { this.backHome() }}>返回home</span>  </div>  <div>  <span onClick={() => { this.jumpToDetail() }}>跳转到detail</span>  </div>  </div>  )  }  backHome() {  //通过js代码跳转  this.props.history.goBack()  }  jumpToDetail() {  // 可能出现死循环使用replace  // this.props.history.replace('/detail');  // this.props.history.push('detail');  // ②隐式传参  this.props.history.push({  pathname: '/detail',  state: {  id: '321'  }  })  }  }  export default about |

在router/router.js中：

|  |
| --- |
| import React from 'react'  import { HashRouter, Route, Switch } from 'react-router-dom'  import Home from '../views/home/home'  import About from '../views/about/about'  import Detail from '../views/detail/detail'  const BasicRoute = () => (  <HashRouter> //跟容器，路由相关的都要包裹在此容器中，内部只能有唯一根元素  <Switch>  // exact表示精确匹配，和跳转的地址一致才能显示，参数使用 **:**  <Route exact path="/" component={Home} />  <Route exact path="/about" component={About} />  <Route exact path="/detail" component={Detail} />  {/\*①通过:id携带参数 \*/}  {/\* <Route exact path="/detail/:id" component={Detail} /> \*/}  </Switch>  </HashRouter>  )  export default BasicRoute |

在app.js中：

|  |
| --- |
| import React, { Component } from 'react'  import Router from './router/router'  export default class App extends Component {  constructor(props) {  super(props)  this.state = {}  }  render() {  return (  <div>  <h2>App跟组件</h2>  <Router />  </div>  )  }  } |

**5.使用Ant Design组件库**

安装：npm install antd -S

**5.1全部导入使用：**

// 在index.js中，全局导入andt样式

import 'antd/dist/antd.css'

// 在需要使用组件的页面局部引入

import { DatePicker } from 'antd'

直接当作标签使用：<DatePicker></DatePicker>

**5.2按需导入使用：**

①安装babel-plugin-import：npm install -D babel-plugin-import

②修改配置文件 .babelrc  
 {  
 "plugins": [  
 ["import", { "libraryName": "antd", "libraryDirectory": "es", "style": "css" }] // `style: true` 会加载 less 文件  
 ]  
 }

③居然没有找到 .babelrc文件，打开 node\_modules/react-scrpts/config/webpack.config.js，搜索 babelrc，配置babel处理 js 文件时，直接禁用了使用 babelrc

④解决：

方法1：

Ⅰ、根目录新建：.babelrc文件，添加：

|  |
| --- |
| {  "plugins": [  ["import", { "libraryName": "antd", "libraryDirectory": "lib"}, "ant"],  ["import", { "libraryName": "antd-mobile", "libraryDirectory": "lib"}, "antd-mobile"]  ]  } |

Ⅱ、在module模块 ,loader: require.resolve('babel-loader')对象下，options中的plugins数组中添加：

[

"import",

{libraryName: "antd", style: 'css'} // 移动端添加 "libraryName": "antd-mobile"

] //antd按需加载

完整如下：

|  |
| --- |
| {  test: /\.(js|mjs|jsx|ts|tsx)$/,  include: paths.appSrc,  loader: require.resolve('babel-loader'),  options: {  plugins: [  [  "import",  {libraryName: "antd", style: 'css'} // 移动端添加 "libraryName": "antd-mobile"  ]  ],  },  }, |

方法2：

Ⅰ、安装：npm i -D react-app-rewired customize-cra

Ⅱ、修改package.json

|  |
| --- |
| "scripts": { - //"start": "react-scripts start", + "start": "react-app-rewired start", - //"build": "react-scripts build", + "build": "react-app-rewired build", - //"test": "react-scripts test", + "test": "react-app-rewired test", } |

Ⅲ、项目根目录创建一个 config-overrides.js 用于修改默认配置。

|  |
| --- |
| const { override, fixBabelImports } = require('customize-cra'); module.exports = override(  fixBabelImports('import', {  libraryName: 'antd',  libraryDirectory: 'es',  style: 'css',  }), ); |

**6、进阶用法**

**6.1 生命周期**

react给组件的生命周期设定了钩子函数，意义就是为了让组件更可控，可以在某一个阶段，做我们指定的事情

回顾:

vue：create/mount/update/destroy

angular：ngOnInit ngOnChanges ngOnDestroy

ionic：ionViewCanEnter/Leave

ionViewDidLoad/WillUnload

ionViewWillEnter/Leave

ionViewDidEnter

React component lifecycle hooks:

①mount:挂载 （负责初始化）

componentWillMount/DidMount

componentDidMount钩子中使用Ajax请求数据，而不是componentWillMount原因：请求是异步的，所以无论你放在两个中的任何一个里面，几乎绝对都会在组件渲染之后，再进行数据渲染，也就是说避免不了二次渲染(第一次渲染为默认值，第二次为请求后的数据渲染)，所以效果上放到哪里都一样，但是在DidMount中可以使用refs了。然后重要的是componentWillMount可能在某些情况下触发多次。

②update:更新 (负责数据变化)

componentWillUpdate/DidUpdate

componentWillReceiveProps(只有在props传来的值变化时才会执行，放心的设置状态)

③unmout:卸载 (负责清理工作)

componentWillUnmount

注意:

①通过ref找元素或者组件实例，必须得等到挂载之后

②初始化工作可以统一在componentDidMount里

③和update相关的钩子函数中要谨慎使用setState

④子组件不可以修改父组件通过属性所传来的值(是只读的)；但是父组件是可以将一个变化的值通过属性传来，当父组件的数据变化，子组件是可以通过componentWillReceiveProps来接收最新变化的数据做处理

官方的编程规范:

react中方法可以分为两类，一个是自定义(handleClick..)，一个自带的(render/componentDidMount/getInitialState)

将自带的写在前边，自定义的写在中间，将render方法写在最后边

**6.2 表单的特殊处理（借助于状态的3个基本操作来实现）**

如果表单元素中 出现了value/checked/selected类似的属性，表单元素操作的结果是有问题，解决方案就是借助于state的绑定效果来实现

①初始化状态

②将状态的值绑定到value/checked/selected属性

③在触发change事件时，修改状态即可

**6.3 实现循环或者判断的功能（重点**）

react没有像vue/angular一样的指令来增强的html模板

(1)条件判断:

方案1：借助于短路逻辑来实现

{expression && 渲染的视图}

适用场景:用在条件相对简单的场合

方案2:借助于方法的封装和调用

showSth(){}

{

this.showSth()

}

(2)循环:

vue：<any v-for="(tmp,index) in myList"></any>

ng：<any \*ngFor="let tmp of myList;let myIndex=index"></any>

react:

{

this.state.\*\*.map((value,index)=>{

return <any key={index}></any>

})

}

综合练习: demo19\_lianxi.html (:-14:40)

①准备两个组件类 MyList MyItem

②MyItem模板

li 包含span/button(delete)

③MyList模板

input/button(add)/ul(调用MyItem)

④功能

点击add，将input的值获取到，保存在状态list，

根据list，在ul中循环的创建多个MyItem

（提醒：要记得将值传递给MyItem，span接收来显示）

点击delete按钮，将下标传递给MyList，将当前的从ul中移除

点击add之后，清空下输入框

|  |
| --- |
| <script type="text/babel">  var MyItem = React.createClass({  handleClick:function(){  //子与父通信的第三步：调用通过自定义属性delFunc所传来的方法  this.props.delFunc()//this.props.myIndex)  },  render:function(){  return <li>  <span>{this.props.myValue}</span>  <button onClick={this.handleClick}>delete</button>  </li>  }  })  var MyList = React.createClass({  getInitialState:function(){  return {list:[]}  },  //子--》父通信第一步：准备一个有参数的方法  handleDel:function(index){  var nowList = this.state.list;  nowList.splice(index,1)  this.setState({list:nowList})  },  handleAdd:function(){  //获取输入框的值  var myConent = this.refs.myInput.value;  //存在状态中  var nowList = this.state.list;  nowList.push(myConent);    this.setState({list:nowList},()=>{  //在设置list状态成功之后，打印数组  console.log(this.state.list);  });  //清空输入框  this.refs.myInput.value = ""  },  render:function(){  return <div>  <input type='text' ref="myInput" placeholder="请输入待做事项"/>  <button onClick={this.handleAdd}>  add  </button>  <ul>  {  this.state.list  .map((value,index)=>{  //子--》父通信第二步：将这个有参数的方法传递给子组件  return <MyItem  delFunc={()=>this.handleDel(index)}  myValue={value} key={index}></MyItem>  })  }  </ul>  </div>  }  })  ReactDOM.render(  <MyList></MyList>,  document.getElementById('example')  )  </script> |

**7、ReactNative**

扩展:

C:\xampp\htdocs\codes\rn\rn\_myapp\node\_modules\react-native\local-cli\server\server.js

1、概述

ReactNative：

Build native mobile apps using JavaScript and React

采用js和react来完成移动端原生app开发(NativeApp)

竞品:

weex(阿里,采用vue的语法来实现nativeAPP)

flutter(Google，采用dart语言来编程)

环境:

①pc端的模板项目（负责业务代码）

②移动端的模拟器（预览效果）

C:\xampp\htdocs\codes\rn\rn\_myapp\android\app\build\outputs\apk

步骤1：确认下是否安装myapp的安装包

步骤2：启动myapp

步骤3：确认电脑的ip、8081

查看ip地址:windows+R->cmd->ipconfig

172.163.100.145:8081

步骤4: 在模拟器中 找到myapp的应用，在此应用内部，按下menu按键/启动摇一摇，启动一个开发菜单

点击dev settings（开发设置）

点击debug server host & port for device

在输入框中输入ip地址和端口: 172.163.100.145:8081

步骤5：检查配置是否生效

摇一摇--》reload

第一次配置ip和端口，reload之类 默认效果看不着，将myapp关闭(近期任务，下滑关闭)，再重新打开

情况1：闪退--》 删除重新安装

情况2：设置--》软件，清除缓存再重装

2、组件的基本处理

组件是可被反复使用的，带有特定功能的视图

创建:

import React,{Component} from 'react'

export default class Demo01 extends Component{

render(){

return \*\*\*

}

}

调用:

import Demo01 from '\*\*\*\*'

<Demo01></Demo01>

注意事项:

①es6的模块化规范中，如果导出一个模块指定default关键字，在引入模块时是无需指定花括号；否则是需要通过花括号指定模块名字

②所有的类要先引入再调用

③不要写html元素，这里不是web编程

练习: (:- 10:45)

①src/components/demo02\_lianxi.js

②在demo02\_lianxi.js的封装一个组件类Demo02Component;在此类中指定要渲染两个Text；一个Text 颜色是红色；另外一个是绿色

//<h1 style={{opacity:0.2}}></h1>

3、数据基本处理

3.1 state

初始化:

constructor(){

super()

this.state = {count:0}

}

读：this.state.count

写: this.setState({count:1})

3.2 fetch

fetch(url)

.then((response)=>{

return response.json()

})

.then((result)=>{

//result就是服务器端所返回的一个经过json反序列化之后的对象或者数组

})

练习: demo05\_lianxi.js (:- 14:25)

①封装一个组件 Demo05Component

②组件挂载完成之后 请求服务器端

http://\*\*\*\*\*\*:8080/product/list

将返回的数据中的data数组保存在状态中

③组件的模板内容 在View中根据数组中元素的个数动态的创建多个Text(商品的title)

3.3 组件间通信

props来完成父--》子的通信工作，也可以通过传递一个有参数的方法给子组件来实现子--》父的通信工作

4、样式和布局

<Text style={{color:'red'}}></Text>

(rn是支持绝大多数css中的样式设定)

布局也是支持flexbox的flexDirection/justifyContent/alignItems

但是有一个注意事项:默认主轴是column

样式的封装和复用:

①引入StyleSheet

import {StyleSheet} from 'react-native'

②创建要封装的样式

var MyStyles = StyleSheet.create({

myText:{color:'red'}

})

③调用

<Text style={MyStyles.myText}></Text>

5、RN中的各种封装的组件的用法(重点)

5.1 View

View 容器

5.2 Text

Text 显示文本

5.3 Button

Button 按钮

<Button onPress={this.handlePress} title="clickMe" color='red'></Button>

5.4 Image

本地图片:

<Image source={require('\*\*\*\*\*')}></Image>

网络图片:

<Image style={{width:100,height:100}} source={{uri:""}}></Image>

注意事项:

①网络资源图片必须设置宽高才能显示

②加载本地图片时，路径不允许做任何的计算（可以将图片存在服务器，网络资源图片时允许做计算的）

5.5 FlatList

FlatList是rn所提供的一种高性能列表

showItem(info){

return 列表项要渲染的内容

}

<FlatList data={} renderItem={this.showItem}></FlatList>

练习:

①demo10\_lianxi.js Demo10Component

②准备一个状态，初始化一个数组

③组件挂载完毕之后，请求http://\*\*\*:8080/product/list，将data保存在状态中

④将状态对应的数组渲染在FlatList，列表项显示商品title、删除的按钮(点击按钮，将列表项从FlatList中移除)