爱创课堂前端培训

# ReactJS

第3天课堂笔记（本课程共4天）

班级：北京前端训练营13期

讲师：张容铭

日期：2018年6月28日

张容铭老师

微博：@张容铭\_YYQH

QQ : 286031482

E-mail : yuye\_qinghe@qq.com

爱创课堂官网 ：[www.icketang.com](http://www.icketang.com)

目录

[ReactJS 1](#_Toc8307)

[一、组件通信 3](#_Toc10696)

[1.1父组件到子组件通信 3](#_Toc31423)

[1.2子组件到父组件通信 3](#_Toc361)

[1.3兄弟组件间通信 3](#_Toc32620)

[二、项目实现 4](#_Toc4946)

[2.1目录部署 4](#_Toc8707)

[2.2需要安装模块 5](#_Toc31289)

[2.3约束属性 5](#_Toc31338)

[三、Flux 7](#_Toc24140)

[3.1 flux架构 7](#_Toc18545)

[3.2 reflux 8](#_Toc16145)

[3.3 redux 8](#_Toc6324)

[3.4三大原则 9](#_Toc32054)

[3.5体验redux 9](#_Toc15775)

[3.6 store对象 10](#_Toc11269)

[3.7 action 10](#_Toc9633)

[3.8 react-reudx 10](#_Toc12119)

# 一、组件通信

组件是完整独立的，因此彼此之间数据是不会共享的，想共享数据要实现组件通信

## 1.1父组件到子组件通信

在父组件中，为子组件添加属性数据：变量，属性数据，状态数据，方法

当执行方法的时候，作用域默认是undefined，子组件绑定作用域就是子组件，父组件绑定作用域就是父组件（常用）

## 1.2子组件到父组件通信

当作用域是父组件的时候，在子组件中使用这个方法，并传递数据，在方法中根据数据内容修改状态，修改是父组件的，这样就实现了子组件到父组件的通信

## 1.3兄弟组件间通信

父组件可以向子组件通信，子组件也可以向父组件通信

如果这两子组件是兄弟关系，我们通过综合使用父组件到子组件通信，以及子组件到父组件通信，这样就可以实现兄弟组件间通信了

父组件向一个子组件传递方法，并绑定父组件作用域，

子组件执行这个方法，传递数据，并修改父组件状态（子组件到父组件通信）

父组件更新状态，进入存在期，并将数据传递给另一个子组件（父组件到子组件通信）

这个子组件，接收数据，更新视图

这样就实现了兄弟组件间通信

如果不相干的两个组件，没有公共的父组件，因此就无法通信，为了解决这个问题，react提出了flux思想（观察者模式的封装）

|  |
| --- |
| 1. // 引入库 2. import React, { Component } from 'react'; 3. import { render } from 'react-dom'; 4. // 一个子组件输入，一个子组件显示 5. // 父组件 6. class Parent extends Component { 7. // 定义状态 8. constructor(props) { 9. super(props); 10. this.state = { 11. msg: '' 12. } 13. } 14. // 定义方法 15. parentMethod(msg) { 16. // 更新状态 17. this.setState({msg}) 18. } 19. // 渲染 20. render() { 21. return ( 22. <div> 23. {/\*传递的方法要绑定状态\*/} 24. {/\*<Child method={this.parentMethod.bind(this)}></Child>\*/} 25. <Child method={msg => this.setState({msg})}></Child> 26. {/\*将状态传递给另一个组件\*/} 27. <Other msg={this.state.msg}></Other> 28. </div> 29. ) 30. } 31. } 32. // 子组件 33. class Child extends Component { 34. // 渲染 35. render() { 36. return ( 37. <div> 38. {/\*当用户输入的时候，更新状态\*/} 39. <input type="text" onChange={e => this.props.method(e.target.value)} /> 40. </div> 41. ) 42. } 43. } 44. // 另一个子组件 45. class Other extends Component { 46. // 渲染 47. render() { 48. return ( 49. <div> 50. <h1>{this.props.msg}</h1> 51. </div> 52. ) 53. } 54. } 55. // 渲染父组件 56. render(<Parent></Parent>, app) |

# 二、项目实现

技术架构：react，react-dom，bootstrap，webpack，less， prop-types，axios

## 2.1目录部署

css css样式库

data 模拟数据

img 相关图片

module 所有脚本模块

bootstrap.jsx 入口文件

app 项目组件

component 页面中所有组件

page 所有页面组件

index.html 项目入口文件

webpack.config.js 配置文件

## 2.2需要安装模块

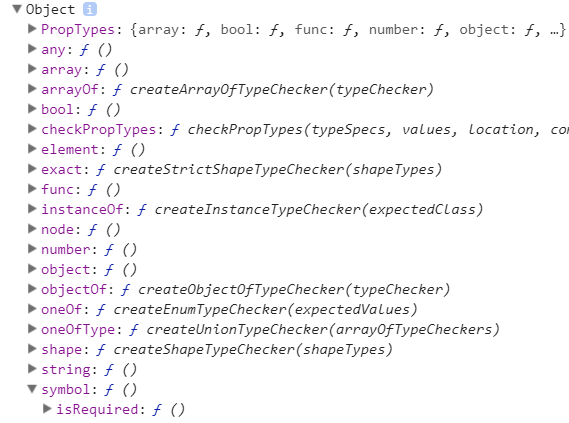
webpack, webpack-dev-sever, react, react-dom, prop-types, aixos, babel-loader, babel, babel-preset-react, babel-preset-es2015, style-loader, css-loader, less-loader, less

## 2.3约束属性

我们可以通过propTypes静态属性，来约束组件创建是传递的属性数据

该模块内置了很多类型，我们可以约束属性，

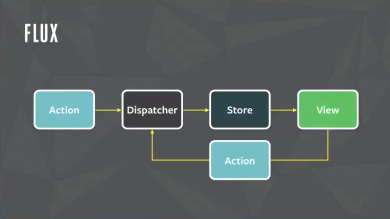
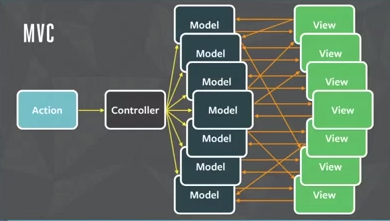
每个属性的下面，都有isRequired的方法，可以约束组件的属性是必填的



|  |
| --- |
| 1. // 引入库 2. import React, { Component } from 'react'; 3. import { render } from 'react-dom'; 4. // 引入组件属性类型 5. import PropTypes from 'prop-types'; 6. // 引入header组件 7. import Header from "../components/header/header.jsx" 8. // 引入页面组件 9. import Home from '../pages/home/home.jsx'; 10. // import Start from '../pages/start/start.jsx'; 11. // import Css from '../pages/css/css.jsx'; 12. // 抽象页面组件 13. import Page from '../pages/page/page.jsx'; 14. import './app.less'; 15. // console.log(PropTypes) 16. // 定义页面组件并暴露出来 17. export default class App extends Component { 18. // 初始化状态 19. constructor(props) { 20. super(props) 21. // 定义页面状态 22. let page = { 23. home: 'none', 24. start: 'none', 25. css: 'none', 26. component: 'none', 27. js: 'none', 28. marker: 'none', 29. } 30. // 根据属性修改状态 31. page[this.props.page] = 'block' 32. // 状态 33. this.state = { page } 34. } 35. // 获取选中的页面 36. getChoosePage() { 37. // 解构数据 38. let { page } = this.state; 39. // 遍历状态，找到选中的页面 40. for (var i in page) { 41. // 如果属性值为block，就是选中的 42. // i表示属性名称,就是页面名称，page[i]表示属性值 43. if (page[i] === 'block') { 44. // 找到页面了，返回页面名称 45. return i; 46. } 47. } 48. } 49. // 切换页面方法 50. togglePage(id) { 51. // console.log('parent', id) 52. // 父组件根据id，更新状态，排他法设置 53. // 所有页面隐藏 54. let { page } = this.state; 55. for (let key in page) { 56. // 隐藏页面 57. page[key] = 'none'; 58. } 59. // 设置选中的页面 60. page[id] = 'block'; 61. // 更新状态 62. this.setState({ page }) 63. } 64. render() { 65. // 解构 66. let { home, start, css, component, js, marker } = this.state.page; 67. return ( 68. <div> 69. {/\*子组件向父组件通信，父组件向子组件通信\*/} 70. <Header page={this.getChoosePage()} method={this.togglePage.bind(this)}></Header> 71. <Home style={home}></Home> 72. <Page style={start} url="data/start.json" title="起步" intro="简要介绍 Bootstrap，以及如何下载、使用，还有基本模版和案例，等等。"></Page> 73. <Page style={css} url="data/css.json" title="全局 CSS 样式" intro="设置全局 CSS 样式；基本的 HTML 元素均可以通过 class 设置样式并得到增强效果；还有先进的栅格系统。"></Page> 74. <Page style={component} url="data/component.json" title="组件" intro="无数可复用的组件，包括字体图标、下拉菜单、导航、警告框、弹出框等更多功能。"></Page> 75. <Page style={js} url="data/js.json" title="JavaScript 插件" intro="jQuery 插件为 Bootstrap 的组件赋予了“生命”。可以简单地一次性引入所有插件，或者逐个引入到你的页面中。"></Page> 76. <Page style={marker} url="data/marker.json" title="定制并下载 Bootstrap" intro="通过自定义 Bootstrap 组件、Less 变量和 jQuery 插件，定制一份属于你自己的 Bootstrap 版本吧。">定制</Page> 77. </div> 78. ) 79. } 80. } 81. // 默认属性数据 82. // App.defaultProps = { 83. // page: 'home' 84. // } 85. // 约束类型 86. App.propTypes = { 87. // 约束属性类型，是必填的 88. page: PropTypes.string.isRequired 89. } |

# 三、Flux

React是基于MVVM模式实现的一个视图层面的框架，自身只负责如何高效的渲染视图，对于数据的处理，本身很薄弱，例如要借助其他框架发送异步请求，组件之间的通信成分非常的高昂等等，所以为了解决数据通信的问题，react提出了flux思想，基于观察者模式实现的



最初React只是提出了思想，并没有实现，因此很多开发者基于该思想，实现了这套架构模式

例如 Redux，Reflux，flux，vuex等等

## 3.1 flux架构

在flux中，有四个模块

action 用户或者视图发布的消息

dispatcher 捕获action消息，并根据消息的类型处理数据，传递给store

store 存储应用程序的数据

view 我们看到的视图

交互流程是

用户或者组件发出action消息，被dispatcher捕获到了，根据消息处理数据，并将数据处理的结果存储在store中，store将新的数据，映射给视图去更新，用户和视图还可以发布新的消息，进入下一次通信的循环

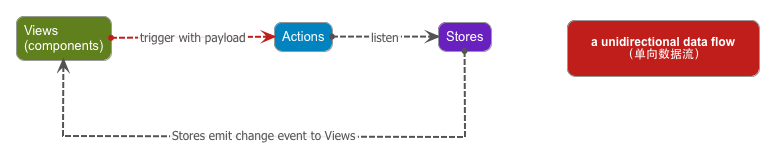
redux和reflux就是基于该模式实现的，

reflux是在ES5中使用的，通过组件的状态传递数据

redux是在ES6中使用的，通过组件的属性传递数据

## 3.2 reflux

reflux架构图



reflux是对flux架构模式的简化，删除了dispatcher模块，将dispatcher模块的业务逻辑放在了store中，因此包含三个模块

view 组件创建的视图

action 视图或者用户发布的消息

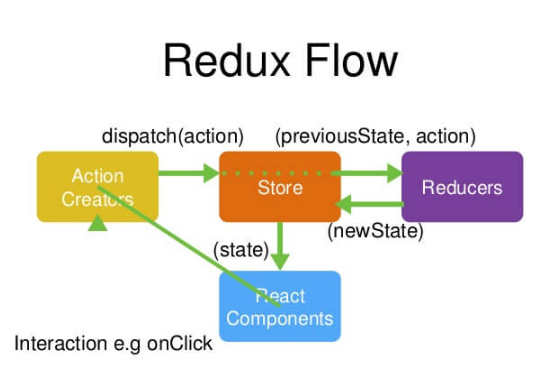
store 存储应用程序中的数据

通信流程是

用户或者视图发布action消息，消息被store捕获到了，store根据消息处理数据，并将处理的结果映射给视图，去更新视图，用户和视图还可以再次发布消息，进入下一次通信循环

## 3.3 redux

redux架构图



redux是严格按照flux思想设计的，包含四个模块

action 用户或者组件发布的消息

reducers 接收消息并根据消息处理数据的方法

store 存储应用程序的所有数据（store中，存储的数据叫state）

view 组件视图

通信流程是

组件或者用户发布消息，通过store被reducers捕获到，reducers根据action消息类型，处理数据，并将数据处理的结果提交给store，store数据更新了，将结果传递给组件去更新视图，用户和组件视图还可以再次发布消息，进入下一次通信循环

## 3.4三大原则

单一数据源

在整个系统中，只能有且只有一个store存储应用程序的数据（vuex也是只有一个store）

state是只读的

在store中，我们只能通过getState方法读取state数据，绝对不能直接修改store中的state数据，只能通过发布action来修改state数据

函数式编程

reducers中修改store中的数据，是通过定义函数实现的

早期redux的设计，并不仅仅为react设计的，它是抽象出来的一个数据层面的框架，我们可以单独使用。

## 3.5体验redux

我们要安装redux， 通过npm安装

npm install redux

redux是基于ES6编程的

引入redux后，提供了一些方法

applyMiddlewarre 定义中间件的

bindActionCreators 绑定创建消息者

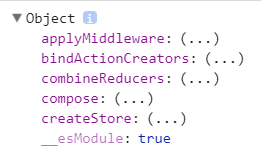
combineReducers 绑定调度者

compose 处理store

createStore 创建store

参数是调度者，

返回值就是store对象



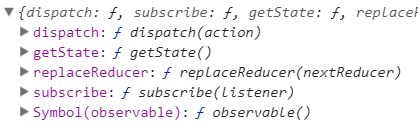
我们通过函数定义调度者

第一个参数表示store中的state对象

第二个参数表示接收到的action消息的

返回值是更新后的state对象

## 3.6 store对象



dispatch 用来发布消息的方法（相当于观察者模式中的发布消息的方法：trigger，emite等等）

getState 用来获取store中的状态数据

replaceReducer 替换调度者的方法

subscribe 监听state变化的方法（相当于观察者模式中的订阅消息的方法: on）

Symbol symbol对象

## 3.7 action

在redux中，action时候一个js对象，包含消息的类型，和传递的数据

## 3.8 react-reudx

在react中，使用redux，要安装react-redux

npm install react-redux

在模块中引入该库

Provider 为组件提供store对象的组件

通过store属性绑定store

connect 用来为组件添加state以及dispatch方法的函数

第一个参数是一个函数，用来说明state是如何提供给组件的

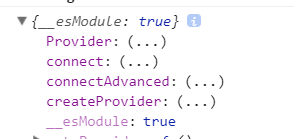
第二个参数是一个函数，用来说明dispatch方法是如何提供给组件

返回值是一个新的函数，接收组件作为参数

作用：将state和dispatch传递给组件

返回值是新组件，是要渲染到页面中的

注意：在组件外部只能通过属性传递数据，所以组件用属性接收其他数据



|  |
| --- |
| 1. // 移入react创建三个组件 2. import React, { Component } from 'react'; 3. import { render } from 'react-dom'; 4. // 引入redux 5. import { createStore } from 'redux'; 6. import { Provider, connect } from 'react-redux'; 7. // console.log(ReactRedux) 8. // 定义消息名称 9. const ADD = 'ADD'; 10. const REDUCE = 'REDUCE'; 11. // 定义调度者 12. function redcer(state = 0, action) { 13. // 根据消息类型，修改数据 14. switch (action.type) { 15. case ADD: 16. return state + action.num; 17. case REDUCE: 18. return state - action.num; 19. } 20. // 返回state 21. return state; 22. } 23. // 创建store 24. let store = createStore(redcer) 25. // 显示数据的组件 26. class ShowChild extends Component { 27. // constructor(props) { 28. // super(props) 29. // this.state = { 30. // num: 0 31. // } 32. // } 33. render() { 34. return ( 35. <div> 36. <h1>{'show： ' + this.props.state}</h1> 37. </div> 38. ) 39. } 40. // 组件构建完成 41. // componentDidMount() { 42. // // 监听消息 43. // store.subscribe(() => this.setState({num: store.getState()})) 44. // } 45. } 46. // 修改数据的组件 47. class ChangeChild extends Component { 48. render() { 49. return ( 50. <div> 51. {/\*<button onClick={e => store.dispatch({type: ADD, num: 10})}>增加10</button>\*/} 52. <button onClick={e => this.props.dispatch({ type: ADD, num: 10 })}>增加10</button> 53. <button onClick={e => this.props.reduceNum(2)}>减少2</button> 54. </div> 55. ) 56. } 57. } 58. // 定义组件 59. class App extends Component { 60. render() { 61. // console.log(this.props) 62. // 将属性解构，传递给组件 63. let { state, dispatch, reduceNum } = this.props; 64. return ( 65. <div> 66. <ShowChild state={state}></ShowChild> 67. <ChangeChild dispatch={dispatch} reduceNum={reduceNum}></ChangeChild> 68. </div> 69. ) 70. } 71. } 72. // 定义传递state的方式 73. function mapStateToProps(state) { 74. // 将state直接传递给组件的属性对象 75. return { 76. state 77. } 78. } 79. // 定义传递dispatch方法的方式 80. function mapDispatchToProps(dispatch) { 81. // 将dispatch方法直接传递给组件 82. return { 83. dispatch, 84. // 定义方法 85. reduceNum(num) { 86. // 构建消息对象 87. dispatch({ type: REDUCE, num }) 88. } 89. } 90. } 91. // 对app组件拓展 92. const DealApp = connect(mapStateToProps, mapDispatchToProps)(App) 93. // 渲染处理后的DealApp组件 94. // 通过Provider为组件添加store对象 95. render( 96. <Provider store={store}> 97. <DealApp></DealApp> 98. </Provider> 99. , app) |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |