爱创课堂前端培训

# React 16

第4天课堂笔记（本课程共5天）

班级：北京前端训练营16期

讲师：张容铭

日期：2018年11月10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

[React 16](#_Toc26654)

[复习](#_Toc4458)

[一、redux](#_Toc24858)

[1.1安装redux](#_Toc11008)

[1.2使用redux](#_Toc3933)

[1.3 action](#_Toc5456)

[1.4 reducer](#_Toc16072)

[1.5 strore](#_Toc21276)

[二、观察者模式解决组件通信](#_Toc23511)

[三、react-redux](#_Toc18607)

[3.1 redux实现组件间通信](#_Toc9346)

[3.2 connect方法](#_Toc17319)

[3.3 Provider组件](#_Toc16713)

[四、路由](#_Toc7778)

[4.1使用路由](#_Toc3976)

[下午复习](#_Toc17268)

[4.2默认路由](#_Toc28549)

[4.3路由重定向](#_Toc14853)

[4.4路由参数](#_Toc15320)

[4.5路由导航](#_Toc27162)

[五、react-router-redux](#_Toc32022)

[5.1路由中使用redux](#_Toc8)

[5.2路由reducer](#_Toc17393)

[5.3 state拓展](#_Toc20647)

[六、redux拓展](#_Toc9507)

[6.1静态action](#_Toc27651)

[6.2动态action](#_Toc28124)

[6.3异步action](#_Toc14995)

[七、ant-design](#_Toc7318)

[7.1官网](#_Toc8521)

[7.2移动端官网](#_Toc18797)

[7.3安装](#_Toc19187)

[7.4体验antd-mobile](#_Toc17020)

[7.5 babel-plugins-import](#_Toc27072)

# 复习

组件间通信

父组件向子组件通信 1数据 2 方法

子组件向父组件通信 1 传递方法， 2 执行方法， 3 修改状态

兄弟组件间通信 1 子组件向父组件通信 2 父组件向子组件通信

引入模块

混合 1 混合类继承Component， 2 组件类继承混合类

flux store， view， action， dispatcher

reflux 基于状态传递数据的， actions， store， view

redux 基于属性传递数据的， actions， reducers， store， view

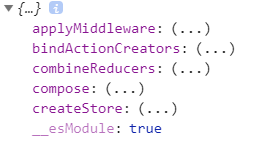
# 一、redux

## 1.1安装redux

在es6开发中，redux也是一个npm模块。我们需要安装

npm install redux

## 1.2使用redux



createStrore方法，用来创建store对象

参数是reducer函数

返回值就是store对象，

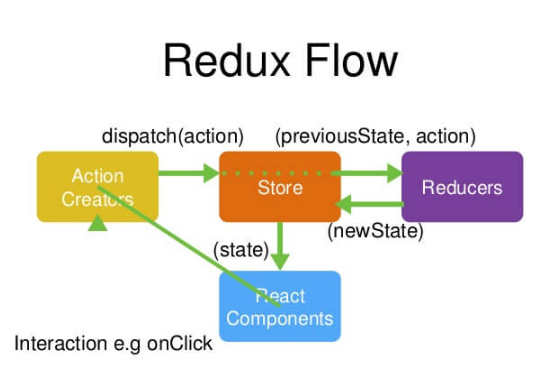
整个应用程序，store对象只有一个

combineReducers方法，用来合并多个reducer函数的

例如：后面学习路由，我们可以通过该方法，合并应用程序的reducer以及路由的reducer

applyMiddleware添加中间件

例如，安装异步actions中间件



## 1.3 action

actions表示消息对象，就是一个js对象，但是该对象要能够说明消息类型

所以通常我们用type属性表示消息类型

工作中，很多地方会判断或使用消息类型，因此我们通常将消息类型定义成常量

## 1.4 reducer

reducer是一个函数，在函数中，我们可以根据action来处理store中的state数据（类似vuex中的mutations，也观察者模式中的on方法）

第一个参数表示store中的state数据

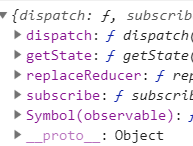
第二个参数表示action对象

必须有返回值，就是修改后的state数据，将存储在store中

注意：当我们创建store的时候，reducer默认执行一次，此时我们可以为state定义默认参数，这样store中的state数据就有了默认值

## 1.5 strore

通过createStore方法，即可创建一个store对象



dispatch 用来发布消息的方法（类似vuex中的dispatch, commit, 类似观察者模式中的trigger, emit）

getState 获取store中的state的方法

replaceReducer 替换store中的reducer

subscribe 用来监听state变化的方法（类似观察者模式中的on）

注意：reducer监听actions，subscribe监听state

symbol 表示symbol属性

拓展

当我们使用解构语法的时候，我们可以为解构的变量定义别名

例如 import { key as a } from ‘’

|  |
| --- |
| 1. import { createStore } from 'redux'; 2. // 定义消息类型常量 3. const ADD = 'ADD'; 4. const COUNT = 'COUNT'; 5. // 首先定义action对象 6. let addNum = { type: ADD, num: 5 }; 7. let countNum = { type: COUNT, num: 2 }; 8. // 定义reducer就是定义函数 9. function reducer(state = 0, action) { 10. // 根据action修改state 11. // console.log(state, action); 12. // 判断action的类型 13. switch (action.type) { 14. case ADD: 15. state += +action.num; 16. break; 17. // 由于state是基本类型，因此可以直接返回，是对象不行 18. // return state += action.num; 19. // return state 20. case COUNT: 21. state -= action.num; 22. break; 23. } 24. // 返回值就是新的state 25. return state; 26. } 27. // 定义store,参数就是reducer函数 28. let store = createStore(reducer); 29. console.log(111, store, store.getState()) 30. // 监听store的变化, 打印state对象 31. store.subscribe(res => console.log(123, store.getState())) 32. // 发布action 33. // store.dispatch() 34. let { dispatch } = store; 35. dispatch(addNum) 36. dispatch(addNum) 37. dispatch(countNum) |

# 二、观察者模式解决组件通信

组件之间通信成本很高，尤其两个不相干的组件是没有办法通信的，

我们要解决组件间通信，我们可以使用观察者模式

一个组件订阅消息

一个组件发布消息

发布消息之后，我们订阅这则消息，并在组件内部修改状态，这样即可实现组件间的通信，也是reflux的实现。

|  |
| --- |
| 1. // 定义组件 2. class App extends Component { 3. render() { 4. // 渲染虚拟DOM 5. return ( 6. <div> 7. <button onClick={e => Observer.trigger('countNum', 2)}>减少2</button> 8. <AddNum></AddNum> 9. <ShowNum></ShowNum> 10. </div> 11. ) 12. } 13. } 14. // 增加5的按钮 15. class AddNum extends Component { 16. render() { 17. // 点击按钮，发布消息 18. return <button onClick={e => Observer.trigger('addnum', 5, 100, 'hello')}>增加5</button> 19. } 20. } 21. // 展示数字 22. class ShowNum extends Component { 23. // 构造函数 24. constructor(props) { 25. super(props); 26. this.state = { 27. num: 0 28. } 29. } 30. render() { 31. return <h1>num: {this.state.num}</h1> 32. } 33. // 组件创建完成订阅事件 34. componentDidMount() { 35. // 订阅事件 36. Observer.on('addnum', (num, ...args) => { 37. // console.log(args) 38. // 更新状态 39. this.setState({ 40. num: this.state.num + num 41. }) 42. }) 43. // 订阅消息 44. Observer.on('countNum', num => this.setState({ 45. num: this.state.num - num 46. })) 47. } 48. } |

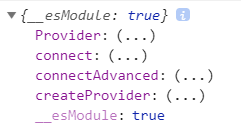
# 三、react-redux

## 3.1 redux实现组件间通信

redux希望可以应用所有框架中，因此在不同的框架中使用redux，就要安装不同的插件。

在react中使用，就要安装react-redux

我们将其引入，提供了一个方法和一组件



## 3.2 connect方法

redux是通过拓展组件的属性，实现组件间通信的，

所以我们通过connet方法，来拓展组件的属性，

两个参数

第一个参数是函数：如何为组件的属性拓展（如何接收）store中的state数据

参数是state数据

返回值表示拓展组件属性的对象（返回对象中有什么数据，此时组件属性就有什么数据）

第二个参数是函数：如何为组件的属性拓展（如何接收）store中的dispatch方法

参数是dispatch方法

返回值表示拓展组件属性的对象（返回对象中有什么方法，此时组件属性就有什么方法）

connect方法执行完毕，得到一个方法，该方法可以拓展组件

参数就是组件

返回值是一个新组件，包含拓展的state数据以及dispatch方法

## 3.3 Provider组件

为组件提供store对象的

必须设置store属性，属性值就是全局store对象

此时子组件将通过connect方法，来接收poriver提供的store对象中的数据

注意：这里只能是子组件接收store数据，其后代组件不能接收store数据，想接收有两种方式

1 在子组件中，通过属性依次传递（常用）

2 通过connect方法拓展组件，去接收store数据

我们渲染组件的时候，渲染Provider组件

|  |
| --- |
| 1. // 定义组件 2. class App extends Component { 3. render() { 4. console.log(123, this) 5. // 解构属性数据 6. let { dispatch, num, addNum } = this.props; 7. // 渲染虚拟DOM 8. return ( 9. <div> 10. <button onClick={e => dispatch(countNum2)}>减少2</button> 11. <AddNum dispatch={dispatch} addNum={addNum}></AddNum> 12. {/\*<ShowNum num={num}></ShowNum>\*/} 13. {/\*拓展后的组件\*/} 14. <DealShowNum></DealShowNum> 15. </div> 16. ) 17. } 18. } 19. // 增加5的按钮 20. class AddNum extends Component { 21. // 发布消息 22. sendMsg() { 23. this.props.dispatch(addNum5) 24. } 25. render() { 26. console.log('addNum', this) 27. // 点击按钮，发布消息 28. return ( 29. <div> 30. <button onClick={this.sendMsg.bind(this)}>增加5</button> 31. <button onClick={e => this.props.dispatch({ type: ADD, num: 10 })}>增加10</button> 32. {/\*简化我们的消息发布\*/} 33. <button onClick={e => this.props.addNum(100)}>增加100</button> 34. <button onClick={e => this.props.addNum(1000)}>增加1000</button> 35. </div> 36. ) 37. } 38. } 39. // 展示数字 40. class ShowNum extends Component { 41. render() { 42. console.log('showNum', this) 43. return <h1>num: {this.props.num}</h1> 44. } 45. } 46. // 渲染组件 47. // render(<App></App>, app) 48. // 定义组件如何接收store数据 49. // 如何接收state数据 50. function mapStateToProps(state) { 51. // 返回值表示拓展给属性的对象 52. return { 53. num: state, 54. color: 'red' 55. } 56. } 57. // 如果接收dispatch方法 58. function mapDispatchToProps(dispatch) { 59. // 返回值表示高拓展给属性的对象 60. return { 61. dispatch, 62. demo() {}, 63. // 简化发送addNum的消息 64. addNum(num) { 65. dispatch({ type: ADD, num }) 66. } 67. } 68. } 69. // 创建拓展组件的函数 70. const dealFn = connect(mapStateToProps, mapDispatchToProps); 71. // 拓展App组件 72. const DealApp = dealFn(App); 73. // 拓展ShowNum组件 74. const DealShowNum = dealFn(ShowNum); 75. // 为app提供store对象 76. // 渲染该组件 77. render(( 78. <Provider store={store}> 79. <DealApp></DealApp> 80. </Provider> 81. ), app) |

# 四、路由

为了实现单页面应用程序，我们要使用路由，从14版本开始，路由为了实现多端适配，将路由拆分成各个方向，例如，web方向要使用web路由，native方向要使用native路由等等

我们在web开发中，我们要使用web路由：react-router-dom

npm install react-router-dom

## 4.1使用路由

使用路由分成三步

第一步 确定路由渲染位置

在应用程序组件（App组件）中，使用Switch组件定义路由的渲染位置

通过Route组件定义每一条规则

name定义路由的名称

path定义路由的规则

component定义渲染的组件

第二步 确定路由渲染策略

内置很多中渲染策略

BrowserRouter：基于path策略，实现的路由，需要后端的配合

HashRouter： 基于Hash策略，实现的路由，不发送新的请求，不需要后端配合

将应用程序组件写在该组件内

第三步 渲染路由

通过render方法，渲染第二步得到的结果



|  |
| --- |
| 1. class App extends Component { 2. render() { 3. // 渲染虚拟DOM 4. return ( 5. <div> 6. <header>header part</header> 7. {/\*第一步 确定路由渲染位置\*/} 8. <Switch> 9. {/\*定义每一条规则\*/} 10. <Route path="/home" component={Home}></Route> 11. <Route path="/list/:page" component={List}></Route> 12. <Route path="/detail/:id" component={Detail}></Route> 13. </Switch> 14. </div> 15. ) 16. } 17. } 18. // 第二步 确定渲染路由方式 19. let routes = ( 20. <HashRouter> 21. <App></App> 22. </HashRouter> 23. ) 24. // 第三步 渲染路由 25. render(routes, app) |

## 下午复习

redux

createStore

combineReducres

applyMiddlewave

四个部分 actions， reducers, store, view

actions 对象

reducers(state, action) 函数

store dispatch, getState, subscribe

观察者模式解决组件通信

redux解决组件通信

connect(mapStateToProps, mapDispatchToProps)(App)

Provider store

1 属性传递， 2 connet继续拓展

路由

1 Switch， Route name, path, component

2 渲染策略：HashRouter，BrowserRouter

3 render()

## 4.2默认路由

path的属性值定义成\*，我们既可定义默认路由

其他路由没有匹配到，会进入该路由

由于该路由匹配的非常广，因此我们将其定义在最后面

## 4.3路由重定向

我们通过Redirect组件，实现路由重定向

from 表示匹配的路径

to 表示重定向的路径

## 4.4路由参数

作为路由渲染的组件，属性对象都继承了三个属性

location：模拟浏览器的location对象，存储路由的地址信息数据（pathname, hash, serach等）

history：模拟浏览器的history对象，用来管理路由的历史记录

match：类似location，但是解析的是路由规则中的数据

注意：只有路由渲染的组件具有这些数据，其后代组件中不具备这些数据，使用这些数据，我们可以通过属性传递的方式实现（类似redux）

## 4.5路由导航

react中，我们通过Link组件定义路由导航

必须设置to属性，表示地址（注意：即使是hash策略，也不要写#）

默认渲染成a标签

|  |
| --- |
| 1. // 定义组件 2. class App extends Component { 3. render() { 4. // 渲染虚拟DOM 5. return ( 6. <div> 7. <header>header part</header> 8. {/\*路由导航\*/} 9. <Link to="/">首页</Link> 10. <Link to="/list/1">列表页</Link> 11. <Link to="/detail/2">详情页</Link> 12. {/\*第一步 确定路由渲染位置\*/} 13. <Switch> 14. {/\*定义每一条规则\*/} 15. {/\*<Route path="/home" component={Home}></Route>\*/} 16. <Route path="/list/:page" component={List}></Route> 17. <Route path="/detail/:id" component={Detail}></Route> 18. {/\*输入ickt进入detail/ickt\*/} 19. <Redirect from="/ickt" to="detail/ickt"></Redirect> 20. {/\*首页是默认路由\*/} 21. <Route path="\*" component={Home}></Route> 22. </Switch> 23. </div> 24. ) 25. } 26. } 27. // 详情页 28. class Detail extends Component { 29. render() { 30. console.log(this) 31. // 点击按钮，发布消息 32. return ( 33. <div> 34. <h1>detail page</h1> 35. <Demo params={this.props.match.params} location={this.props.location}></Demo> 36. </div> 37. ) 38. } 39. } |

# 五、react-router-redux

## 5.1路由中使用redux

在路由中使用redux只需要两步

第一步 在路由策略组件中，通过Route组件，渲染redux处理后的DealApp组件

第二步 Provider组件中，渲染路由规则。

注意：路由组件Route只会向渲染的组件传递路由数据，不会传递store数据，想具有store数据，我们要通过connet方法拓展该路由渲染的组件

## 5.2路由reducer

想使用路由reducer，我们要安装react-router-redux模块

此时提过一个routerReducer方法，供我们使用。

此时一个应用程序就有多个reducer了，想同时使用他们就要使用redux提供的combineReducers方法了

参数是对象

key表示state名称

value是reducer函数

理解为：类似vuex中的module，对state切割

## 5.3 state拓展

state是只读的，因此在任何位置，都不能修改它，我们只能复制一份后，再使用，不能直接修改

如果state是一个值类型，我们可以直接修改，并返回新的结果

如果state是一个引用类型的，我们必须复制一份，再修改，最后将合并的结果返回

合并多个对象我们可以使用ES6提供了Object.assign()方法

注意:修改的对象，放在参数的最后面。

|  |
| --- |
| 1. function reducer(state = { num: 0, msg: '' }, action) { 2. // 定义处理的数据 3. let obj = {}; 4. // 判断action 5. switch (action.type) { 6. case ADD: 7. obj.num = state.num + action.num; 8. break; 9. case MSG: 10. obj.msg = action.msg; 11. break; 12. } 13. // 返回state 14. // 最终我们要将obj和state合并在一个新的对象中，再返回 15. return Object.assign({}, state, obj); 16. } 17. // 创建store 18. let store = createStore(combineReducers({ 19. routerReducer, 20. ickt: reducer 21. })); 22. class Header extends Component { 23. render() { 24. return ( 25. <div> 26. <header>header part</header> 27. {/\*此时使用多个reducer，已经对state切割，所以访问原来的数据，要加上命名空间\*/} 28. {/\*<h3>num: {this.props.state.ickt}</h3>\*/} 29. <h3>num: {this.props.state.ickt.num}</h3> 30. <h3>msg: {this.props.state.ickt.msg}</h3> 31. </div> 32. ) 33. } 34. } 35. // 1 通过Route引入DealApp 36. let routes = ( 37. <HashRouter> 38. {/\*<App></App>\*/} 39. {/\*通过Route组件，引入DealApp\*/} 40. <Route path="/" component={DealApp}></Route> 41. </HashRouter> 42. ) 43. // 2 在Provider中，渲染路由规则 44. render(( 45. <Provider store={store}> 46. {routes} 47. </Provider> 48. ), app) |

# 六、redux拓展

action是用户或者组件发布的消息，携带消息类型以及消息数据

## 6.1静态action

如果action对象的数据是固定不变的，就是静态action

不够灵活

## 6.2动态action

动态action是一个函数，接收数据参数，并且根据参数返回一个完整action对象

## 6.3异步action

当需要异步请求的时候，我们可以在请求完成的时候，发送action，更新数据

但是发送请求的这一切操作，都是在组件内部实现的。

react是一个视图层面的框架，它不希望我们在组件内部做数据的处理，如果能将异步请求这类操作，移入action中，就完美了，这样就需要发布异步的action消息。

redux中没有内置异步的消息，使用异步的消息我们要安装redux插件，

注意：redux插件都是作为中间件存在的。

我们可以通过redux-thunk中间件，发布异步action

npm install redux-thunk

当我们将其引入之后，提供了一个拓展createStore方法的中间件，

redux模块提供了applyMiddleware方法，用来安装中间件

参数是一个中间件

返回值是一个方法，可以拓展createStore方法

通过redux-thunk中间件拓展后的createStore方法，创建的store对象中，就可以使用异步action了。

异步action与动态action区别

动态action动态返回一个action对象

返回一个方法，

参数是dispatch

在方法中，我们可以执行异步操作，操作结束后，我们通过dispatch再次发布同步消息。（类似vuex中的actions模块）

|  |
| --- |
| 1. // 安装中间件，拓展createStore方法 2. const newCreateStore = applyMiddleware(thunkMiddleware)(createStore) 3. // 用newCreateStore创建的store对象，就可以使用异步action 4. let store = newCreateStore(combineReducers({ 5. routerReducer, 6. ickt: reducer 7. })); 8. // 动态action 9. function numAction(num) { 10. // 根据参数，拼接完成的action对象 11. return { type: ADD, num } 12. } 13. function msgAction(msg) { 14. return { type: MSG, msg } 15. } 16. // 异步action 17. function msgActionAsync(num) { 18. // console.log(arguments) 19. // 返回函数中的参数是dispatch 20. return dispatch => { 21. // console.log(dispatch) 22. // 发送请求, 请求时，我们还可以添加参数 23. axios.get('data/demo.json?num=' + num) 24. // 监听数据返回, 发布同步消息 25. .then(({ data }) => dispatch(msgAction(data.msg))) 26. } 27. } 28. // 首页 29. class Home extends Component { 30. // 获取数据的方法 31. getData() { 32. // 发送请求 33. axios 34. .get('data/demo.json') 35. // 监听数据返回, 向reducer发布消息，更新数据 36. .then(({ data }) => this.props.dispatch(msgAction(data.msg))) 37. } 38. render() { 39. console.log(this) 40. // 点击按钮，发布消息 41. return ( 42. <div> 43. <h1>home page</h1> 44. {/\*发布动态消息\*/} 45. <button onClick={e => this.props.dispatch(numAction(5))}>增加5</button> 46. <button onClick={e => this.props.dispatch(numAction(10))}>增加10</button> 47. <button onClick={e => this.props.dispatch(msgAction('爱创课堂'))}>显示信息</button> 48. <button onClick={e => this.getData()}>请求数据</button> 49. <button onClick={e => this.props.dispatch(msgActionAsync(10))}>异步请求数据</button> 50. {/\*<button onClick={e => this.props.dispatch(addNum)}>增加5</button> 51. <button onClick={e => this.props.dispatch(hello)}>显示信息</button>\*/} 52. </div> 53. ) 54. } 55. } |

# 七、ant-design

是一个React的UI框架

它包括移动端，pc端

## 7.1官网

https://ant.design/

## 7.2移动端官网

https://mobile.ant.design/index-cn

## 7.3安装

也是一个npm模块，所以通过npm来安装

npm install antd-mobile

## 7.4体验antd-mobile

提供了大量的组件，我们可以直接引入并使用

## 7.5 babel-plugins-import

antd-mobile提供了大量的组件，组件不论有没有使用，都打包在一起，会让文件很大，所以提供babel-plugins-import插件模块，我们在webpack配置babel-loader将其引入

在options属性中配置

在plugins中配置，类似babel-preset-es2015在presets中配置一样

属性值: [

[‘import’, {

style: ‘css’,

libraryName: ‘antd-mobile’

}]

]

options选项包括加载的所有配置，因此定义了options，就不要再定义query了，因此presets也写在options中。

我们引入css文件，要配置css加载机

|  |
| --- |
| 1. import React, { Component } from 'react'; 2. // 引入antd-mobile组件 3. import { WhiteSpace, WingBlank, Flex, Calendar } from 'antd-mobile'; 4. import './layout.less'; 5. // 无状态组件 6. const Block = props => <div className="block">Block</div> 7. // 定义组件并暴露接口 8. export default class Layout extends Component { 9. render() { 10. return ( 11. <div> 12. {/\*前后留白\*/} 13. <WhiteSpace size="lg"></WhiteSpace> 14. {/\*左右留白\*/} 15. <WingBlank> 16. <span>hello</span> 17. </WingBlank> 18. {/\*flex布局组件\*/} 19. <Flex justify="center"> 20. <Block></Block> 21. <Block></Block> 22. <Block></Block> 23. </Flex> 24. {/\*日历组件\*/} 25. <Calendar visible={true}></Calendar> 26. </div> 27. ) 28. } 29. } |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |