爱创课堂前端培训

# ReactJS

第2天课堂笔记（本课程共4天）

讲师：张容铭

日期：2018年6月27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

[名称 1](#_Toc30542)

[复习 3](#_Toc10211)

[一、DOM事件 4](#_Toc30408)

[二、事件委托 4](#_Toc12060)

[三、状态 5](#_Toc588)

[3.1无状态组件 5](#_Toc4772)

[3.2有状态组件 5](#_Toc2193)

[四、组件生命周期 6](#_Toc9181)

[4.1创建期 7](#_Toc17644)

[4.2子组件 8](#_Toc2650)

[4.3组件存在期 9](#_Toc31728)

[4.4销毁期 12](#_Toc28291)

[五、侵入式库 12](#_Toc23036)

[六、axios 13](#_Toc17001)

[七、组件的约束性 14](#_Toc27462)

[7.1非约束性组件 14](#_Toc25136)

[7.2约束性组件 14](#_Toc8482)

[7.3下拉框的约束性 16](#_Toc20653)

[7.3.1单选下拉框的约束性 17](#_Toc13972)

[7.3.2多选下拉框的约束性 17](#_Toc14904)

# 复习

1 node， 2 webpack， webpacl-dev-server 2 获取react

webpack配置： entry， output， module， rules: [{test, loader}]

虚拟DOM， 渲染， 组件

jsx：

组件的属性 this.props, 类.defaultProps

虚拟DOM四类属性：元素属性， 特殊元素属性， 非元素属性， 自定义数据属性

ES：类：安全类， 继承：特性寄生组合式继承

原子继承

# 一、DOM事件

DOM元素可以绑定事件，

<div onclick=”callback”></div>

虚拟DOM绑定事件

<div onClick={this.callback}></div>

1 事件名称大写

2 事件回调函数定义在组件类中，通过this访问

3 绝对不能执行（不能有参数集合）

ES6语法中：事件回调函数this

默认值是undefined，为了能够在组件中使用其他的方法，我们要想方设法将this指向组件

有两种方式，

1 通过ES5提供方法bind改变作用域，也可以传递自定义参数

ES6编程中，事件回调函数的this的指向，可以是任意的

2 通过ES6提供的箭头函数改变作用域，也可以传递自定义参数

事件回调函数中，默认有个参数，就是事件对象（是react封装后的事件对象）

因为react是通过事件委托模式实现的（1减少事件数量，2避免内存外泄，3预言未来元素）

|  |
| --- |
| 1. // 定义组件 2. class Demo extends Component { 3. // 事件回调函数 4. clickBtn1(e) { 5. console.log(this, arguments) 6. console.log(e.target, e.currentTarget) 7. } 8. // 渲染的方法 9. render() { 10. // 组件有且只能有一个根元素 11. return ( 12. <div> 13. {/\*事件\*/} 14. <button onClick={this.clickBtn1}>按钮1</button> 15. {/\*改变作用域\*/} 16. <button onClick={this.clickBtn1.bind(this, 100, true)}>按钮2</button> 17. <button onClick={this.clickBtn1.bind(obj, 100, true)}>按钮2</button> 18. {/\*ES6改变作用域\*/} 19. <button onClick={(e) => {this.clickBtn1(e, 200, true)}}>按钮3</button> 20. {/\*两点省略\*/} 21. <button onClick={e => this.clickBtn1(e, 300, true)}>按钮3</button> 22. </div> 23. ) 24. } 25. } |

# 二、事件委托

将子元素的事件委托给父元素，在事件的触发过程中，会经过捕获，触发，冒泡的过程，我们可以在冒泡阶段捕获每一个元素，根据某个特征，找到该子元素，执行事件回调函数

|  |
| --- |
| 1. // 事件委托模式 1 减少事件数量 2. var ul = document.getElementsByTagName('ul')[0]; 3. // 绑定事件 4. ul.onclick = function(e) { 5. // 根据元素名称找到li 6. if (e.target.tagName.toUpperCase() === 'LI') { 7. // 改变背景 8. e.target.style.background = 'green' 9. } 10. // console.log(e.target.tagName) 11. } |

# 三、状态

状态跟属性一样，也是用来存储组件数据的

属性存储的数据是组件外部提供的数据

状态存储的数据是组件内部维护的数据

根据组件是否具有状态，将组件分成两类，一类是无状态组件，一类是有状态组件

## 3.1无状态组件

## 3.2有状态组件

状态跟属性类似

属性可以通过this.props获取

状态也可以通过this.state获取

区别是

组件的属性只能在外部改变

组件的状态只能在内部改变

this.setState改变状态数据

参数是对象

key表示状态数据名称

value表示新的值

注意：绝对不能通过this.state更新数据

属性可以定义默认属性数据

通过defaultProps静态属性数据定义

状态也可以定义初始化的状态

在构造函数中，通过this.state定义

注意：当我们重写构造函数的时候，一定要执行super继承父类构造函数

不论是属性数据和状态数据的改变都会导致render方法的执行

|  |
| --- |
| 1. class Demo extends Component { 2. // 初始化状态数据 3. constructor(props) { 4. // 构造函数继承 5. super(props); 6. // 初始化状态 7. this.state = { 8. // 显示的新闻索引 9. num: 0 10. } 11. } 12. // 创建字列表 13. createChildItem(arr) { 14. // 根据数组创建列表 15. return arr.map((item, index) => { 16. return <span key={index}>{item}</span> 17. }) 18. } 19. // 创建每一行列表 20. createList() { 21. let len = this.props.data.length; 22. // 遍历属性数据 23. return this.props.data.map((item, index) => { 24. // 列表要设置key 25. // 根据index和num值决定到底应该显示哪一个 26. return <li key={index} style={{ 27. display: index === this.state.num % len ? 'block' : 'none' 28. }}>{this.createChildItem(item)}</li> 29. }) 30. } 31. // 切换li 32. toggleLi() { 33. // 更新num数据 34. this.setState({ 35. num: ++this.state.num 36. }) 37. } 38. render() { 39. // console.log(this) 40. // 渲染 41. return ( 42. <div> 43. <span onClick={this.toggleLi.bind(this)}>换一换</span> 44. <ul>{this.createList()}</ul> 45. </div> 46. ); 47. } 48. } |

# 四、组件生命周期

react为了说明组件的创建，存在，销毁过程，提供了组件生命周期

共分三大周期

创建期：组件创建过程

ES5中五个阶段： getDefaultProps, getInitailState, componentWillMount, render, componentDidMount,

ES6中，只有三个阶段componentWillMount, render, componentDidMount

存在期：组件存在过程

五个阶段：componentWillReceiveProps, shouldComponentUpdate, componentWillUpdate, render, componentDidUpdate

销毁期: 组件销毁过程

一个阶段：componentWillUnmount

为了详细说明每个周期，提供了不同的阶段

## 4.1创建期

包含五个阶段

getDefaultProps 定义默认状态数据

this指向构造函数，而不是组件实例化对象

ES6不存在，用defaultProps代替

getInitailState 初始化状态数据

es6不存在，在constructor中初始化

在这个阶段已经有了属性数据，所以工作中，我们用属性数据初始化状态数据，实现数据有外部流入内部

componentWillMount 组件即将更新

属性数据已经存在，状态数据已经存在，

注意：千万修改他们，一旦修改就进入存在期（我们可以请求数据或者初始化插件）

我们无法访问虚拟DOM

render

属性数据，状态数据已经存在，返回值是虚拟DOM

我们渲染虚拟DOM的时候，可以使用他们了

所以方法中，访问不了虚拟DOM

componentDidMount

组件已经创建完成，我们可以访问属性数据，状态数据，虚拟DOM

我们可以发送请求，更新DOM，绑定事件等等

一旦该方法执行完毕，标志着组件创建器的结束，存在期的开始

ES6中，创建期的三个方法，this指向组件实例化对象，没有参数

|  |
| --- |
| 1. // 定义组件 2. class Demo extends Component { 3. // 2 初始化状态数据 4. constructor(props) { 5. // props代表属性数据 6. // 构造函数继承 7. super(props); 8. console.log('constructor', this, arguments) 9. // 初始化状态 10. this.state = { 11. num: 100, 12. // 将外部的数据，转化成内部的数据 13. color: this.props.color 14. } 15. } 16. // 3 组件即将构建 17. componentWillMount() { 18. console.log('componentWillMount', this.state, this.props, arguments, findDOMNode(this)) 19. } 20. // 4 渲染组件 21. render() { 22. console.log('render', this.state, this.props, arguments) 23. // 解构数据 24. let { color, num } = this.state; 25. // 返回值就是虚拟DOM 26. return ( 27. <h1>{color + '--' + num}</h1> 28. ) 29. } 30. // 组件构建完成 31. componentDidMount() { 32. console.log('componentDidMount', this.state, this.props, arguments, findDOMNode(this)) 33. } 34. } 35. // 1 默认属性 36. Demo.defaultProps = { 37. color: 'red' 38. } |

## 4.2子组件

虚拟DOM与真实DOM行为是一致的，

真实DOM可以添加子元素，虚拟DOM也可以

组件与虚拟DOM行为是一致的，

虚拟DOM可以添加子元素，组件也可以添加子组件

子组件：在该组件中创建的组件，称之为该组件的子组件，该组件称之为父组件

例如

Slider是父组件

GoBack是子组件

|  |
| --- |
| 1. <Slider> 2. <GoBack /> 3. </Slider> |

组件是完整独立的，因此组件的数据，彼此之间是不会共享的，想让组件之间共享数据，我们可以使用组件通信技术

父组件到子组件通信

将父组件的数据传递给子组件

子组件是定义在父组件中的，所以我们可以给子组件添加属性数据，子组件就接收了该数据，从而实现了父组件中的数据传递给子组件

传递给子组件的数据可以是父组件中的

变量中存储的数据

父组件的属性数据

父组件的状态数据

父组件的方法

默认方法this指向undefind

事件回调函数通常要绑定组件作用域

此时父组件方法作用域就是子组件

为了纠正这个问题，我们在传递子组件方法的时候，一定要绑定父组件作用域

不论是子组件中绑定作用域，还是父组件中绑定作用域，我们都可以传递参数

父组件的方法，一旦绑定父组件实例化对象，此时就无法访问子组件实例化对象了

为了解决这个问题，我们可以将父组件传递的方法，在子组件方法中实行

|  |
| --- |
| 1. // 父组件 2. class Slider extends Component { 3. // 状态数据 4. constructor(props) { 5. super(props); 6. // 初始化 7. this.state = { 8. text: '网址导航' 9. } 10. } 11. // 定义方法 12. clickMethods() { 13. console.log(this, arguments) 14. } 15. // 渲染 16. render() { 17. // let title = '返回顶部'; 18. return ( 19. <div> 20. {/\*传递变量\*/} 21. <GoBack title={title}></GoBack> 22. {/\*传递属性数据\*/} 23. <GoBack title={this.props.text}></GoBack> 24. {/\*状态数据\*/} 25. <GoBack title={this.state.text}></GoBack> 26. {/\*传递方法\*/} 27. <GoBack parentMethod={this.clickMethods.bind(this, 'red', 'green')} title={this.state.text}></GoBack> 28. </div> 29. ) 30. } 31. } |

## 4.3组件存在期

父组件可以向子组件传递属性数据，当传递的这些数据发生改变，组件就进入了存在期，执行存在期的方法

（本质上，不论是属性数据的改变，还是状态数的改变，组件都会进入了存在期，执行存在期的方法）

componentWillReceiveProps 组件即将接收新的属性数据

只有属性改变的时候，会执行这个方法，状态数据的改变不会执行这个方法

所以工作中，我们常常在这个方法中，用属性数据对状态数据赋值，实现数据由外部流入内部

参数是新的属性数据

组件上的仍然旧的属性数据

shouldComponentUpdate 组件是否应该更新（优化的方法，避免死循环）

第一个参数是新的属性数据

第二个参数是新的状态数据

组件上的仍然是旧的属性数据和旧的状态数据

必须有返回值，表示是否应该更新

true表示可以更新

false表示不能更新

注意：不要在这个方法中，更新数据，否则死循环，我们在高频操作（事件）中使用

componentWillUpdate 组件即将更新（为了更新属性数据和状态数据）

第一个参数表示新的属性数据

第二个参数表示新的状态数据

组件上仍然是旧的属性数据和旧的状态数据

虚拟DOM也没有更新

方法执行完毕，属性数据和状态数据才会更新

render 组件渲染虚拟DOM，跟创建期的render方法是同一个

组件上已经是新的属性数据和新的状态数据

方法执行完毕，虚拟DOM就更新了

注意：绝对不能访问虚拟DOM

componentDidUpdate 组件（所有一切）更新完毕

第一个参数是旧的属性数据

第二个按时是旧的状态数据

组件上已经是新的属性数据和新的状态数据

该方法执行完毕，并不意味着存在期的结束，存在期仍然继续，是存在期的一次更新的结束

以上方法作用域上下文对象都指向组件实例化对象

|  |
| --- |
| 1. class GoBack extends Component { 2. // 初始化状态数据 3. constructor(props) { 4. super(props); 5. // 初始化 6. this.state = { 7. // 创建时候，将外部的数据，存储在内部 8. title: this.props.title 9. } 10. } 11. // 监听窗口的滚动 12. componentDidMount() { 13. window.onscroll = () => { 14. // 如果滚动条距离顶部200,显示返回顶部 15. if (window.scrollY > 200) { 16. this.setState({ title: '返回顶部' }) 17. // 否则显示网址导航 18. } else { 19. this.setState({ title: '网址导航' }) 20. } 21. } 22. } 23. // 定义事件回调函数 24. childCallback(...arg) { 25. // 返回顶部 26. window.scrollTo(0, 0) 27. } 28. // 存在期的五个方法 29. // 1 组件即将接收新的属性数据 30. componentWillReceiveProps(props, state) { 31. console.log(111, 'componentWillReceiveProps', props, state, this.props, this.state, props.title, this.props.title) 32. // 将外部的数据，存储在内部 33. this.setState({ 34. title: props.title 35. }) 36. } 37. // 2 组件是否应该更新 38. shouldComponentUpdate(props, state) { 39. // this.setState({title: '123'}) 40. console.log(222, 'shouldComponentUpdate', props, state, this.props, this.state, state.title, this.state.title) 41. // 再去更新状态，将死循环 42. // this.setState({color: 'red'}) 43. // 是否应该更新 44. // return true; 45. // 如果属性或者状态数据改变了，往下进行才有必要 46. return props.title !== this.props.title || state.title != this.state.title 47. return true 48. } 49. // 3 组件即将更新 50. componentWillUpdate(...arg) { 51. // this.setState({title: '123'}) 52. console.log(333, 'componentWillUpdate', arg, this.props, this.state) 53. } 54. // 4 组件渲染新的虚拟DOM 55. render(...arg) { 56. console.log(444, 'render', arg, this.props, this.state) 57. // <span className="go-back" onClick={this.childCallback.bind(this)}>{this.props.title}</span> 58. return ( 59. <span className="go-back" onClick={this.childCallback.bind(this)}>{this.state.title}</span> 60. ) 61. } 62. // 5 组件更新完毕 63. componentDidUpdate(...arg) { 64. // this.setState({title: '123'}) 65. console.log(555, 'componetnDidUpdate', arg, this.props, this.state) 66. } 67. } |

## 4.4销毁期

销毁期只有一个阶段：componentWillUnmount

组件从页面移除，将进入销毁期，该方法的执行，标志着组件一生的结束，是最后一次能够访问组件的方法了

作用域上下文对象this指向组件实例化对象

没有参数

|  |
| --- |
| 1. // 定义组件 2. class Demo extends Component { 3. // 渲染的方法 4. render() { 5. // 组件有且只能有一个根元素 6. return ( 7. <div> 8. <h1>安徽电信</h1> 9. </div> 10. ) 11. } 12. // 组件渲染完成 13. componentWillUnmount() { 14. console.log(this, arguments) 15. } 16. } 17. // 渲染组件 18. render(<Demo></Demo>, app) 19. setTimeout(function() { 20. // 2秒钟，我们在app中渲染新的虚拟DOM 21. render(<h1>爱创课堂</h1>, app) 22. }, 2000) |

# 五、侵入式库

我们可以在react中，使用一些第三方库和框架，例如jquery，axios等等

使用的这些库或者框架可以分成两类

一类是非侵入式类库（框架）

一类是侵入式类库（框架）

如果库或者框架操作了虚拟DOM，并且没有被react获知，那么就是侵入式的，

如果库或者框架没有操作了虚拟DOM，那么就是非侵入式的

例如使用jquery

使用jquery操作虚拟DOM创建的真实DOM元素，（设置样式，设置属性，绑定事件等等）

此时就是侵入式的

使用jquery发送异步请求，绑定窗口事件等等，

此时就是非侵入式的

如果使用非侵入式类库，可以正常使用

如果使用侵入式的类库，要注意组件生命周期。

在react中，尽量不要使用侵入式的插件

尽量用react的方式更新DOM

我们可以通过npm安装jquery

npm install jQuery 1.7

npm install jquery 3.+

发送异步请求，需要服务，可以通过执行 webpack-dev-server 指令启动服务器

默认端口号是8080.一旦8080被占用，会自动加一

# 六、axios

react中发送异步请求，建议我们使用axios库

严格遵守ES6中Promise规范实现的框架

Promise规范是将异步写成同步的规范

我们通过get，post等简便方法发送请求

通过then方法，监听回调

回调函数参数中，data属性表示返回的数据

|  |
| --- |
| 1. // 定义组件 2. class News extends Component { 3. // 定义默认状态数据 4. constructor(props) { 5. super(props); 6. this.state = { 7. // 将属性中的数据转换成状态 8. list: this.props.data 9. } 10. } 11. // 创建新闻列表的方法 12. createList() { 13. return this.state.list.map((item, index) => { 14. // 列表要添加key 15. return <li style={{ 16. background: 'pink' 17. }} key={index}>{item}</li> 18. }) 19. } 20. // 点击推荐按钮，显示新的信息 21. showMessage() { 22. // 可以通过axios更新数据 23. axios.get('data/data.json') 24. // 在外部写回调, 参数解构，参数集合不能省略 25. .then(({ data }) => this.setState({ list: data })) 26. // 缓存this 27. // let me = this; 28. // 发送异步请求 29. // $.get('data/data.json') 30. // 监听返回 31. // .then(list => this.setState({ list })) 32. // .then(list => { 33. // // console.log(res) 34. // // 将返回的数据，放在状态中，即可渲染 35. // this.setState({ list }) 36. // }) 37. // ES5 38. // .then(function() {}.bind(this)) 39. } 40. // 渲染新闻 41. render() { 42. return ( 43. <div> 44. <span onClick={this.showMessage.bind(this)}>推荐</span> 45. <ul>{this.createList()}</ul> 46. </div> 47. ) 48. } 49. // 组件渲染完成，我们将li的背景色改成绿色 50. componentDidMount() { 51. // $('li').css('background', 'green') 52. } 53. // 存在期再次更新即可 54. componentDidUpdate() { 55. // $('li').css('background', 'green') 56. } 57. } |

# 七、组件的约束性

表单元素与用户产生交互，会产生数据，产生的数据存储位置决定组件的约束性

如果数据存储在元素自身，就是非约束性组件

如果数据存储在组件状态中，就是约束性组件

## 7.1非约束性组件

如果数据存储在元素自身，就是非约束性组件

定义元素的默认值

defaultValue，defaultChecked

获取元素的数据

1 为元素设置ref属性

2 通过this.refs获取元素

3 通过该元素获取数据

修改元素的数据

1 为元素设置ref属性

2 通过this.refs获取元素

3 通过该元素的value或者checked属性修改元素的值

## 7.2约束性组件

如果数据存储在组件状态中，就是约束性组件

定义元素的默认值

value或者checked绑定状态数据

此时一定要定义onChnage事件，并且在事件回调函数中，将元素输入的内容，同步到组件的状态中

获取数据

获取状态中的数据

修改数据

修改状态中的数据

react也是一个基于MVVM模式实现的框架，也实现了数据的双向绑定

数据由模型流入视图，通过数据绑定或者插值实现的

数据由视图流入模型，通过事件监听事件的

在angular中，数据的双向绑定过程中，可以自动化的做表单校验，但是在react中我们要手动的校验

|  |
| --- |
| 1. // 非约束性组件 2. class Demo extends Component { 3. // 查看数据 4. showResult() { 5. // 2 通过this.refs获取数据 6. // 3 通过元素获取值 7. console.log(this.refs.username.value, this.refs.intrest.checked) 8. } 9. // 修改数据 10. changeValue() { 11. // 1 设置ref 12. // 2 获取元素 13. // 3 通过元素，修改数据 14. this.refs.username.value = '安徽电信'; 15. this.refs.intrest.checked = false; 16. } 17. // 渲染方法 18. render() { 19. return ( 20. <div> 21. {/\*定义默认值\*/} 22. {/\*1 设置ref\*/} 23. <input ref="username" type="text" defaultValue="用户名"/> 24. <input ref="intrest" type="checkbox" defaultChecked/> 25. <button onClick={this.showResult.bind(this)}>获取数据</button> 26. <button onClick={e => this.changeValue(e)}>修改数据</button> 27. </div> 28. ) 29. } 30. } 31. // 约束性组件 32. class Demo extends Component { 33. // 定义状态 34. constructor(props) { 35. super(props); 36. // 初始化状态 37. this.state = { 38. username: '爱创课堂', 39. intrest: true 40. } 41. } 42. // 查看数据 43. showResult() { 44. // 获取状态中数据 45. console.log(this.state) 46. } 47. // 修改数据 48. changeValue() { 49. // 修改状态中数据 50. this.setState({ 51. username: '安徽电信', 52. intrest: false 53. }) 54. } 55. // 定义事件回调函数 56. changeUername(e) { 57. // console.log(e.target.value) 58. // 根据输入的内容，存储数据 59. // this.setState({ 60. // username: e.target.value 61. // }) 62. // 我们认为用户名是4-8位任意的字符 63. let username = e.target.value; 64. // 检测输入的是否正确 65. if (/^.{4,8}$/.test(username)) { 66. // 更新数据 67. this.setState({ 68. username 69. }) 70. } 71. } 72. changeIntrest(intrest) { 73. this.setState({ 74. intrest 75. }) 76. } 77. // 渲染方法 78. render() { 79. // 解构 80. let { username, intrest } = this.state; 81. return ( 82. <div> 83. {/\*默认值要绑定状态\*/} 84. <input type="text" value={username} onChange={e => this.changeUername(e)}/> 85. {/\*<input type="checkbox" checked={intrest} onChange={e => this.changeIntrest(e.target.checked)}/>\*/} 86. <input type="checkbox" checked={intrest} onChange={e => this.setState({intrest: e.target.checked})}/> 87. <button onClick={this.showResult.bind(this)}>获取数据</button> 88. <button onClick={e => this.changeValue(e)}>修改数据</button> 89. </div> 90. ) 91. } 92. } |

## 7.3下拉框的约束性

我们通过select元素定义下拉框，通过option定义选项

默认option的值是内容值，当定义了value就是value

我们为select元素添加multiple属性就是多选或者复选下拉框了

select的值也是通过value来定义

### 7.3.1单选下拉框的约束性

非约束性单选下拉框

默认值：用defaultValue定义

获取值：

1为select元素设置ref属性

2 通过this.refs属性获取元素

3 通过元素获取值

修改值：

1为select元素设置ref属性

2 通过this.refs属性获取元素

3 通过元素修改值

约束性单选下拉框

默认值：为value绑定状态，要在onChange事件中，修改状态

获取值：

获取状态值

修改值：

修改状态值

所以单选下拉框跟input是一样的

### 7.3.2多选下拉框的约束性

一旦为select元素添加multiple属性，此时就是多选下拉框，

多选下拉框的值不再是下拉框了，而是数组

所以我们添加的默认值是数组

非约束性多选下拉框

默认值：通过defaultValue定义默认值

获取值

通过select获取所有选项，遍历选项，获取选中的选项的值

修改值

通过select获取所有选项，遍历选项，如果选项的值在选中的数组中，设置为true，否则为false

约束性多选下拉框

默认值：为value绑定状态（绑定数组），设置onChange事件

获取值

获取状态值

修改值

修改状态

|  |
| --- |
| 1. // 约束性多选下拉框 2. class Demo extends Component { 3. // 定义状态数据 4. constructor(props) { 5. super(props); 6. this.state = { 7. color: ['isRed', 'isGreen'] 8. } 9. } 10. // 查看数据 11. showResult() { 12. console.log(this.state.color) 13. } 14. // 修改数据 15. changeValue(e) { 16. // console.log(e.target.value) 17. // 遍历所有的选项，找出选中的选项，将其值存储在状态中 18. let options = e.target.options; 19. // 选项的值 20. let color = []; 21. // 4中判断数组方式 1 instanceof Array 2 constructor === Array 3 Object.prototype.toString.call [objct Array] 4 isArray 22. // console.log(Array.isArray(options)) 23. // 遍历 24. for (let i = 0; i < options.length; i++) { 25. // 选项是否被选中 26. if (options[i].selected) { 27. // 存储选项的值 28. color.push(options[i].value) 29. } 30. } 31. // 更新状态 32. this.setState({ color }) 33. } 34. changeColorValue() { 35. // 修改状态值 36. this.setState({ 37. color: ['isRed', 'isBlue'] 38. }) 39. } 40. // 渲染方法 41. render() { 42. return ( 43. <div> 44. {/\*选项一旦定义value值，value就不再是默认值了\*/} 45. <select multiple value={this.state.color} onChange={this.changeValue.bind(this)}> 46. <option value="isRed">red</option> 47. <option value="isGreen">green</option> 48. <option value="isBlue">blue</option> 49. </select> 50. <button onClick={this.showResult.bind(this)}>获取数据</button> 51. <button onClick={e => this.changeColorValue(e)}>修改数据</button> 52. </div> 53. ) 54. } 55. } |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |