爱创课堂前端培训

# React 16

第2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7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 1](#_Toc22456)

[复习 3](#_Toc4477)

[一、React 4](#_Toc5710)

[1.1换肤 4](#_Toc17648)

[1.1.1异步请求 4](#_Toc18296)

[1.2组件生命周期 5](#_Toc900)

[1.2.1创建期 5](#_Toc30630)

[1.2.2获取虚拟DOM 6](#_Toc2268)

[1.2.3子组件 6](#_Toc6425)

[1.2.4父组件到子组件通信 7](#_Toc14008)

[复习 9](#_Toc25862)

[1.2.5存在期 10](#_Toc11786)

[1.2.6销毁期 12](#_Toc23271)

[1.3组件的约束性 12](#_Toc29060)

[1.3.1非约束性组件 12](#_Toc30522)

[1.3.2约束性组件 13](#_Toc26084)

[1.3.3数据双向绑定 14](#_Toc17718)

[1.3.4下拉框的约束性 15](#_Toc25922)

[1.3.5单选下拉框 16](#_Toc19271)

[1.3.6多选下拉框 16](#_Toc1579)

[1.4使用非react类库 18](#_Toc4177)

[1.5使用侵入式类库 18](#_Toc31835)

# 复习

浏览器原理 浏览器内核，

渲染流程 DOM tree， render tree，

repaint， reflow

react 虚拟DOM， 组件开发， 多端适配（混合开发）

虚拟DOM createElement(name|comp, props, ...children)

key, ref, props, type

render(vdom, dom, fn)

组件 class Demo extends Component { render() {} }

jsx语法 <div></div> <input type=”text” />

插值语法 {} type={}

注释

属性

元素属性 style={{}}

特殊元素属性 className, htmlFor

自定义数据属性

非元素属性

ref, this.refs

dangerouslySetInnerHTML={\_\_html: ‘’}

key

组件属性 this.props, defaultProps,

属性约束 propTypes isRequired

事件 onClick={this.fn}

参数：事件对象 事件委托

this： this.fn.bind(this, 100) e => this.fn(e, 100)

状态：this.state, constructor(props) { super(props); this.state={} } this.setState({key: value])

# 一、React

## 1.1换肤

列表布局

容器宽度W = 成员宽度iw \* 成员数量n + 成员边距m \* ( 成员数量n - 1 )

W = 178 \* 6 + 10 \* (6 - 1)

W = 1118

### 1.1.1异步请求

react中发送异步请求，使用axios库

|  |
| --- |
| 1. // 定义组件 2. class Skin extends Component { 3. // 构造函数 4. constructor(props) { 5. // 构造函数继承 6. super(props); 7. // 初始化状态 8. this.state = { 9. data: [] 10. } 11. } 12. // 事件回调函数 13. chooseItem(id, e) { 14. // console.log(e.target, e.currentTarget) 15. // console.log(id, e.currentTarget.getAttribute('data-id')) 16. // 设置背景 17. document.body.style.backgroundImage = `url(img/skin/big\_${id}.jpg)` 18. } 19. // 创建列表的方法 20. createList() { 21. return this.state.data.map((item, index) => ( 22. <li key={index} data-id={item.id} onClick={e => this.chooseItem(item.id, e)}> 23. <img src={'img/skin/' + item.src} alt=""/> 24. <p>{item.title}</p> 25. </li> 26. )) 27. } 28. // 渲染 29. render() { 30. // console.log(this.state.data) 31. // 返回虚拟DOM 32. return ( 33. <div className="skin"> 34. <div className="container"> 35. <ul>{this.createList()}</ul> 36. </div> 37. </div> 38. ) 39. } 40. // 组件创建完成的时候 41. componentDidMount() { 42. // 发送请求 43. axios.get('data/skin.json') 44. // 监听数据返回 45. // .then(({ data }) => console.log(data)) 46. .then(({ data }) => this.setState({ data })) 47. } 48. } |

## 1.2组件生命周期

为了说明组件的创建，存在，销毁的过程，提供了组件生命周期，共分三大周期

创建期：说明组件创建的过程，类似人的少年。

存在期：说明组件存在的过程。类似人的中年

销毁期：说明组件销毁的过程，类似人的老年

### 1.2.1创建期

创建期为了说明组件的创建全过程，分成了五个阶段

在ES5开发中，由五个方法表示：getDefaultPorps, getInistalState, componentWillMount, render, componentDidMount.

在ES6开发中，前两个方法移除了，但是也有五个过程（移除了前两个方法）

第一个过程：定义默认属性数据

通过defaultProps静态属性定义，此时组件就有了默认属性数据

使用组件的时候，传递的同名属性数据将被覆盖

第二个过程：初始化状态数据

在构造函数中初始化（为this.state赋值）

构造函数：第一个参数是属性对象，我们要通过super实现构造函数继承，并且还要传递属性参数对象。

此时参数中的属性对象(props)与组件实例化对象中的属性对象(this.props)就相同。

在构造函数中，我们可以访问属性数据，因此我们可以用属性数据为this.state初始化

这样就实现了外部数据（属性数据）流入组件内部（状态数据）,这是工作中常见的

第三个过程：组件即将构建

通过componentWillMount定义

在该方法中，我们可以访问属性数据，可以访问状态数据，但是不能访问虚拟DOM，

在方法中，我们不要修改状态数据（否则直接进入存在期）

工作中，我们常常在这个方法中，发送请求，初始化一些插件等等

第四个过程：渲染组件

通过render方法实现

返回值就是渲染输出的虚拟DOM

在函数体内不能访问虚拟DOM，但是可以访问属性数据和状态数据，所以可以用这些数据渲染虚拟DOM

注意：在该阶段不要修改状态或者属性数据

第五个阶段：组件创建完成

通过componentDidMount定义

我们可以使用属性数据，可以使用状态数据，还可以访问虚拟DOM

我们可以发送请求，实现交互，绑定事件等等

该方法执行完毕，标志着组件的创建期的结束，存在期的开始

创建期的方法（除了render），在组件的一生中，只执行一次

### 1.2.2获取虚拟DOM

react-dom渲染库，提供了一个findDOMNode的方法。可以获取组件中的虚拟DOM。

|  |
| --- |
| 1. // 定义组件 2. class Gotop extends Component { 3. // 2 初始化状态数据 4. constructor(props) { 5. // 构造函数继承 6. super(props); 7. console.log(2, 'constructor', props, this.props, this.state) 8. // 初始化 9. this.state = { 10. text: props.title 11. } 12. } 13. // 3 组件即将创建 14. componentWillMount() { 15. console.log(3, 'componentWillMount', this.props, this.state, findDOMNode(this)) 16. } 17. // 4 渲染虚拟DOM 18. render() { 19. console.log(4, 'render', this.props, this.state) 20. // 虚拟DOM 21. return <span>{this.props.title}</span> 22. } 23. // 5 组件渲染完成 24. componentDidMount() { 25. console.log(5, 'componentDidMount', this.props, this.state, findDOMNode(this)) 26. } 27. } 28. // 1 默认属性数据 29. Gotop.defaultProps = { 30. title: '返回顶部' 31. } |

### 1.2.3子组件

虚拟DOM可以有子虚拟DOM，组件跟虚拟DOM的行为是一致的，因此组件也可以有子组件

定义在该组件内部的组件，称之为该组件的子组件，该组件就是父组件。

|  |
| --- |
| 1. <Ickt> 2. <Demo /> 3. <Ickt> |

Ickt组件就是父组件，Demo组件就是子组件

组件是一个完整独立的，因此组件之间数据不会共享，想让组件之间共享数据，我们要实现组件间通信

### 1.2.4父组件到子组件通信

子组件在父组件内部使用，因此我们可以为子组件添加父组件中的数据，作为子组件属性数据，通常有两类数据

一类是数据：变量与对象，属性数据，状态数据等

这些数据改变，子组件接收的属性数据就会改变

一类是方法：方法可以在子组件中使用，有两种用法

注意：传递的方法不能执行，执行了相当于传递方法的返回值数据了

第一种用法 作为子组件的事件回调函数

默认参数是事件对象，  
 this默认是undefined

改变this指向有两个位置：

1 在子组件中绑定

this指向子组件

此时我们可以添加参数，参数在事件对象之前

2 在父组件中绑定 （工作中常用）

此时子组件绑定将失效，并且this指向父组件

此时子组件与父组件都可以传递参数，

参数的顺序是：父组件传递的，子组件传递的，事件对象

第二种用法 在子组件方法中执行

默认没有参数，但是我们可以传递

this默认指向子组件属性对象，

我们将父组件传递的方法绑定父组件实例化对象，this就指向父组件了

父组件方法中this指向父组件

子组件方法中this指向子组件

所以整个过程中，我们既可以访问父组件，也可以访问子组件

因此访问的对象是最全的，也是工作中常用的方式

这种方式有个简便的写法

直接用箭头函数代替子组件定义的方法，

并且这种简便的写法，在工作中更加的常用

|  |
| --- |
| 1. // 定义父组件 2. class Slider extends Component { 3. // 定义状态 4. constructor(props) { 5. super(props); 6. // 初始化状态 7. this.state = { 8. info: '专业前端培训学校' 9. } 10. } 11. render() { 12. let title = 'hello' 13. // 使用子组件 14. return ( 15. <div> 16. {/\*传递变量固定值\*/} 17. {/\*<Gotop title={title}></Gotop>\*/} 18. {/\*属性数据\*/} 19. {/\*<Gotop title={this.props.title}></Gotop>\*/} 20. {/\*状态数据\*/} 21. {/\*<Gotop title={this.state.info}></Gotop>\*/} 22. {/\*传递方法\*/} 23. {/\*父组件中绑定this\*/} 24. {/\*<Gotop title={title} method={this.parentMethod}></Gotop>\*/} 25. <Gotop title={title} method={this.parentMethod.bind(this, 200)}></Gotop> 26. </div> 27. ) 28. } 29. // 父组件方法 30. parentMethod() { 31. console.log('parent', this, 111, arguments) 32. } 33. } 34. // 定义子组件 35. class Gotop extends Component { 36. render() { 37. // console.log(this) 38. // 虚拟DOM 39. /\*return <span onClick={this.props.method.bind(this, 100)}>{this.props.title}</span>\*/ 40. // return <span onClick={this.childMethod.bind(this)}>{this.props.title}</span> 41. // 简便写法 42. return <span onClick={e => this.props.method(300, e)}>{this.props.title}</span> 43. } 44. // 子组件方法 45. childMethod(e) { 46. console.log(this, arguments) 47. // console.log(this.props) 48. // 子组件方法中，执行父组件方法 49. this.props.method(true, e, 100); 50. } 51. } |

### 复习

发送异步请求 axios

组件生命周期：

创建期，存在期，销毁期

创建期：defaultProps， constructor， componentWillMount， render， componentDidMount

子组件

组件通信

父组件到子组件通信

1 数据：变量，状态，属性

2 方法

作为子组件事件回调函数

在子组件方法中执行

### 1.2.5存在期

一旦组件的属性数据或者状态数据发生改变，组件将进入存在期

存在期共分五个阶段：componentWillReceiveProps， shouldComponentUpdate，componentWillUpdate， render， componentDidUpdate

第一个阶段 组件即将接收新的属性数据：componentWillReceiveProps

只有当属性改变的时候，才会执行该方法，状态改变不会执行，直接进入第二个方法

第一个参数表示新的属性数据

组件实例化对象中，仍然是旧的属性数据

作用：我们通常用新的属性数据，为状态数据赋值，实现数据由外部流入内部

第二个阶段 组件是否应该更新：shouldComponentUpdate

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

组件实例化对象中，仍然是旧的属性数据，和旧的状态数据

必须有返回值

true 表示可以进行更新

false 不能进行更新

作用：高频更新中，我们可以订阅该方法，优化更新

工作中，我们通常比较新的属性数据与旧的属性数据，或者新的状态数据与旧的状态数据是否发生改变，改变才更新，没有改变，不要更新。

第三个阶段 组件即将更新：componentWillUpdate

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

组件实例化对象上的，仍然是旧的属性数据，和旧的状态数据

虚拟DOM仍然是旧的

当该方法执行完毕，数据才会更新

作用：初始化插件，修改插件配置，请求数据等等

第四个阶段 渲染虚拟DOM：render

组件实例化对象上的，变成了新的属性数据和新的状态数据了

返回值就是渲染的虚拟DOM

作用：渲染虚拟DOM

注意：不能在该方法中，访问虚拟DOM

第五个阶段 组件更新完毕：componentDidUpdate

第一个参数表示旧的属性数据

第二个参数表示旧的状态数据

组件实例化对象上，是新的属性数据，和新的状态数据

访问的虚拟DOM已经是新的了

作用：请求数据，交互，插件等等

注意：该方法执行完毕，并不是说存在期的结束，只是存在期的一次更新的结束

注意：所有生命周期方法，this都指向组件实例化对象

|  |
| --- |
| 1. // 定义子组件 2. class Gotop extends Component { 3. // 构造函数 4. constructor(props) { 5. super(props); 6. this.state = { 7. // 外部的数据，存储在内部 8. text: props.title 9. } 10. } 11. // 1 组件即将接收新的属性数据 12. componentWillReceiveProps(newProps) { 13. console.log(111, 'componentWillReceiveProps', 'props', this.props, newProps) 14. // 转换数据 15. this.setState({ 16. // 新属性数据，赋值状态数据 17. text: newProps.title 18. }) 19. } 20. // 2 组件是否应该更新 21. shouldComponentUpdate(newProps, newState) { 22. console.log(222, 'shouldComponentUpdate', 'props', this.props, newProps, 'state', this.state, newState) 23. // 必须有返回值 24. // return true; 25. // return false 26. // 判断属性数据或者状态数据是否发生改变 27. return newProps.title !== this.props.title || newState.text !== this.state.text; 28. } 29. // 3 组件即将更新 30. componentWillUpdate(newProps, newState) { 31. console.log(333, 'componentWillUpdate', 'props', this.props, newProps, 'state', this.state, newState) 32. } 33. // 4 渲染虚拟DOM 34. render() { 35. console.log(444, 'render', 'props', this.props, 'state', this.state) 36. // 简便写法 37. // return <span>{this.props.title}</span> 38. return <span>{this.state.text}</span> 39. } 40. // 5 组件更新完毕 41. componentDidUpdate(oldProps, oldState) { 42. console.log(555, 'componentDidUpdate', 'props', this.props, oldProps, 'state', this.state, oldState) 43. } 44. // 组件创建完成，订阅事件 45. componentDidMount() { 46. window.onscroll = () => { 47. // 如果距离顶部超过200像素，显示返回顶部 48. if (window.scrollY > 200) { 49. this.setState({ 50. text: '返回顶部' 51. }) 52. } else { 53. // 否则显示网址导航 54. this.setState({ 55. text: '网址导航' 56. }) 57. } 58. } 59. } 60. } |

### 1.2.6销毁期

当组件从页面中移除，组件将进入销毁期，销毁期只有一个阶段方法

componentWillUnmount 组件即将销毁

该方法执行完毕，组件就开始销毁了，我们就再也访问不到该组件了

|  |
| --- |
| 1. // 定义组件 2. class Skin extends Component { 3. // 渲染 4. render() { 5. // console.log(this.state.data) 6. // 返回虚拟DOM 7. return ( 8. <div className="skin"> 9. <h1>hello</h1> 10. </div> 11. ) 12. } 13. // 销毁期方法 14. componentWillUnmount() { 15. console.log(this, arguments) 16. } 17. } 18. // 渲染 19. render(<Skin></Skin>, app) 20. // 2秒之后渲染其他虚拟DOM 21. setTimeout(() => { 22. render(<h1>爱创课堂</h1>, app) 23. }, 2000) |

## 1.3组件的约束性

当用户与组件中的表单元素产生交互的时候，此时就会产生数据，我们要存储这些数据，根据数据存储的位置，我们将组件分成两类

非约束性组件： 如果产生数据存储在元素自身，那么这类组件就是非约束性组件

约束性组件： 如果产生的数据存储在组件的状态中，那么这类组件就是约束性组件

### 1.3.1非约束性组件

非约束性组件中的数据要存储在元素自身，

定义默认值

通过defaultvalue,或者是defaultChecked非元素属性设置

获取数据

1 为元素设置ref属性

2 在组件中通过this.refs获取该元素

3 通过该元素获取value或者checked属性值

修改数据

1 为元素设置ref属性

2 在组件中通过this.refs获取该元素

3 通过修改该元素的value或者checked属性值，来修改数据

|  |
| --- |
| 1. // 定义组件 2. class Demo extends Component { 3. // 事件回调函数 4. showResult() { 5. // 获取元素，再获取值 6. console.log(this.refs.username.value) 7. console.log(this.refs.password.checked) 8. } 9. changeResult() { 10. // 获取元素再修改value或者checked 11. this.refs.username.value = '专业前端培训学校'; 12. this.refs.password.checked = false; 13. } 14. render() { 15. // 虚拟DOM 16. return ( 17. <div> 18. <p> 19. <label>用户名</label> 20. {/\*非约束性组件中，默认值通过defaultValue或者defaultChecked\*/} 21. {/\*设置ref属性\*/} 22. <input type="text" ref="username" defaultValue="爱创课堂"/> 23. </p> 24. <p> 25. <label>是否设置密码</label> 26. <input type="checkbox" ref="password" defaultChecked/> 27. </p> 28. <p> 29. <button onClick={e => this.showResult(e)}>获取数据</button> 30. <button onClick={e => this.changeResult(e)}>修改数据</button> 31. </p> 32. </div> 33. ) 34. } 35. } |

### 1.3.2约束性组件

如果交互产生的数据，存储在组件的状态中。此时该类组件就是约束性组件。

定义默认值

我们为元素设置value或者checked属性，来设置默认值

注意：

一定要通过插值语法绑定状态

设置onChange事件，在事件回调函数中，修改状态

1 获取事件对象

2 获取该元素，以及获取该元素的值

3 用该元素的值，修改状态

对于输入框来说，有时我们需要对其进行检测

如果检测结果为true，我们更新状态

如果检测结果为false，我们不去更新状态

这样就实现了对表单内容的验证

获取数据

获取状态数据

修改数据

修改状态数据

### 1.3.3数据双向绑定

react也是基于MVVM模式实现的框架，因此也支持数据双向绑定

vue中数据双向绑定通过v-model实现

react中数据双向绑定是手动实现的

数据由模型进入视图：通过插值语法实现的

数据由视图进入模型：通过事件监听实现的

并且在事件监听过程中，可以对数据做校验

|  |
| --- |
| 1. // 定义组件 2. class Demo extends Component { 3. // 构造函数 4. constructor(props) { 5. super(props); 6. // 初始化状态 7. this.state = { 8. username: '爱创', 9. password: true 10. } 11. } 12. // 事件回调函数 13. showResult() { 14. // 获取状态值 15. console.log(this.state) 16. } 17. changeResult() { 18. // 修改状态值 19. this.setState({ 20. username: '爱创课堂', 21. password: false 22. }) 23. } 24. // 修改用户名 25. changeUserName(e) { 26. // 获取元素的值 27. let username = e.target.value; 28. // 长度不能查过8位 29. if (username.length > 8) { 30. // 不要更新了 31. return; 32. } 33. // 可以更新状态 34. this.setState({ username }) 35. } 36. // 修改密码 37. changePassowrd(e) { 38. // 获取数据 39. let password = e.target.checked; 40. // 更新状态 41. this.setState({ 42. password 43. }) 44. } 45. render() { 46. // 解构状态 47. let { username, password } = this.state; 48. // 虚拟DOM 49. return ( 50. <div> 51. <p> 52. <label>用户名</label> 53. {/\*默认值要绑定状态\*/} 54. {/\*<input type="text" value={this.state.username}/>\*/} 55. <input type="text" onChange={e => this.changeUserName(e)} value={username}/> 56. </p> 57. <p> 58. <label>是否设置密码</label> 59. {/\*<input type="checkbox" checked={this.state.password}/>\*/} 60. {/\*<input type="checkbox" onChange={e => this.changePassowrd(e)} checked={password}/>\*/} 61. {/\*对于简单的数据更新，可以直接在箭头函数内完成\*/} 62. <input type="checkbox" onChange={e => this.setState({ password: e.target.checked })} checked={password}/> 63. </p> 64. <p> 65. <button onClick={e => this.showResult(e)}>获取数据</button> 66. <button onClick={e => this.changeResult(e)}>修改数据</button> 67. </p> 68. </div> 69. ) 70. } 71. } |

### 1.3.4下拉框的约束性

通过select元素定义下拉框，

通过option定义每一个选项，默认是内容值，定义了value属性就是value值

下拉框分成两类，一类是单选下拉框，一类是多（复）选下拉框

通过multiple属性切换

单选下拉框值是字符串

多选下拉框值是数组

### 1.3.5单选下拉框

非约束性的

默认值

通过为select设置defaultValue属性，来设置默认值值

获取值

设置ref属性，通过this.refs获取元素，再获取值

修改值

设置ref属性，通过this.refs获取元素，再修改值

约束性的

默认值

为select元素的value属性绑定状态

同时还要设置onChange事件，在事件回调函数中，修改状态

获取值

获取状态值

修改值

修改状态

### 1.3.6多选下拉框

多选下拉框要将value绑定数组

|  |
| --- |
| 1. // 非约束性组件 2. class Demo extends Component { 3. // 事件回调函数 4. showResult() { 5. // 获取元素 6. let select = this.refs.color; 7. // 获取选项并定义结果 8. let options = select.options; 9. let result = [] 10. // 遍历选项 11. for (let i = 0; i < options.length; i++) { 12. // 找出选中的选项 13. if (options[i].selected) { 14. // 将内容存储在result中 15. result.push(options[i].value) 16. } 17. } 18. // 找出结果 19. console.log(result) 20. } 21. changeResult() { 22. // 选中前两个 23. let choose = ['isRed', 'isGreen']; 24. // 获取options 25. let options = this.refs.color.options; 26. // 遍历options，找出value在choose数组中的选项，将其selected设置为true，不在choose数组中的选项，将selected设置为false 27. // ickt: for (let i = 0; i < options.length; i++) { 28. // // 每个选项就是options[i] 29. // let item = options[i]; 30. // // item.selected = false; 31. // // 遍历choose数组 32. // for (let j = 0; j < choose.length; j++) { 33. // // 每个成员是choose[j],判断是否与item.value相等 34. // if (choose[j] === item.value) { 35. // // 将selected设置为true 36. // item.selected = true; 37. // // 我们要查看先一个选项option，而不是choose下一个成员 38. // continue ickt; 39. // // break; 40. // } 41. // } 42. // // 不在choose数组中，selected要设置成false 43. // item.selected = false; 44. // } 45. // es5 46. // for (let i = 0; i < options.length; i++) { 47. // // 查看是否在choose数组中 48. // if (choose.indexOf(options[i].value) >= 0) { 49. // // 在choose数组中，设置为选中态 50. // options[i].selected = true; 51. // } else { 52. // // 不在choose数组中，设置为非选中态 53. // options[i].selected = false; 54. // } 55. // } 56. // es6 57. // 将类数组对象，转化成数组对象并遍历 58. Array.from(options) 59. // 遍历数组 60. .forEach(item => { 61. // 查看是否在choose数组中 62. if (choose.indexOf(item.value) >= 0) { 63. return item.selected = true; 64. } 65. item.selected = false; 66. }) 67. } 68. render() { 69. // 虚拟DOM 70. return ( 71. <div> 72. <p> 73. <select multiple defaultValue={['isRed', 'isBlue']} ref="color"> 74. <option value="isRed">red</option> 75. <option value="isGreen">green</option> 76. <option value="isBlue">blue</option> 77. </select> 78. </p> 79. <p> 80. <button onClick={e => this.showResult(e)}>获取数据</button> 81. <button onClick={e => this.changeResult(e)}>修改数据</button> 82. </p> 83. </div> 84. ) 85. } 86. } |

## 1.4使用非react类库

当我们在react中，使用非react家族类库的时候，我们操作元素一定要注意组件生命周期

|  |
| --- |
| 1. class Demo extends Component { 2. // 构造函数 3. constructor(props) { 4. super(props); 5. // 状态 6. this.state = { 7. url: './demo.jpg', 8. width: 200 9. } 10. } 11. render() { 12. return ( 13. <div> 14. <div className="container" ref="container"></div> 15. <div> 16. <img src={this.state.url} width="50" height="50" alt=""/> 17. </div> 18. </div> 19. ) 20. } 21. // 组件创建完成 22. componentDidMount() { 23. // 使用插件 24. new ImageZoom(this.refs.container, this.state.url, this.state.width) 25. } 26. } |

## 1.5使用侵入式类库

在react中使用其他插件，类库的时候，一定要注意该类库是否具有侵入性

如果该类库操作了虚拟DOM，该类库就是侵入式的，

如果该类库没有操作虚拟DOM，该类库就是非侵入式的

例如，我们使用的image-zoom就是一个侵入式的类库

使用侵入式的类库，一定要注意组件生命周期

对于一个类库来说，是侵入式的还是非侵入式，要看我们在react中如何来使用

例如，使用jQuery

如果我们用jQuery只是发送异步请求，没有操作虚拟DOM，因此就是非侵入式类库

如果我们用jQuery修改元素的样式，就操作了虚拟DOM，因此就是侵入式的类库

在ES6中，使用jQuery，我们要安装该模块

npm install jquery

在react中，不建议使用侵入式类库操作虚拟DOM。

|  |
| --- |
| 1. // 定义组件 2. class Demo extends Component { 3. // 构造函数 4. constructor(props) { 5. super(props); 6. // 状态 7. this.state = { 8. news: ["网瘾真的可以戒吗?", "lol解说当场被开除", "快刀切水果"] 9. } 10. } 11. // 创建新闻列表 12. createList() { 13. // return this.state.news.map((item, index) => <li key={index}>{item}</li>) 14. return this.state.news.map((item, index) => <li style={{ 15. background: 'orange' 16. }} key={index}>{item}</li>) 17. } 18. render() { 19. return ( 20. <div> 21. <ul>{this.createList()}</ul> 22. </div> 23. ) 24. } 25. // 组件创建完成 26. componentDidMount() { 27. // 2秒之后请求了新的数据 28. setTimeout(() => { 29. this.setState({ 30. news: ["承认骗局！这理论创始人被判罚1.05亿", "台湾\"电竞\"队赢了日本 结局却尴尬了", "马化腾曝出新招：要准备开发VR版微信", "中国离婚率连涨15年 女性越来越\"敢离\"", "姑娘发现\"家里有矿\" 但都是给弟弟的"] 31. }) 32. }, 2000) 33. // $('li').css({ 34. // background: 'green' 35. // }) 36. } 37. // 存在期更新结束，再次修改样式 38. componentDidUpdate() { 39. // $('li').css({ 40. // background: 'green' 41. // }) 42. } 43. } |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |