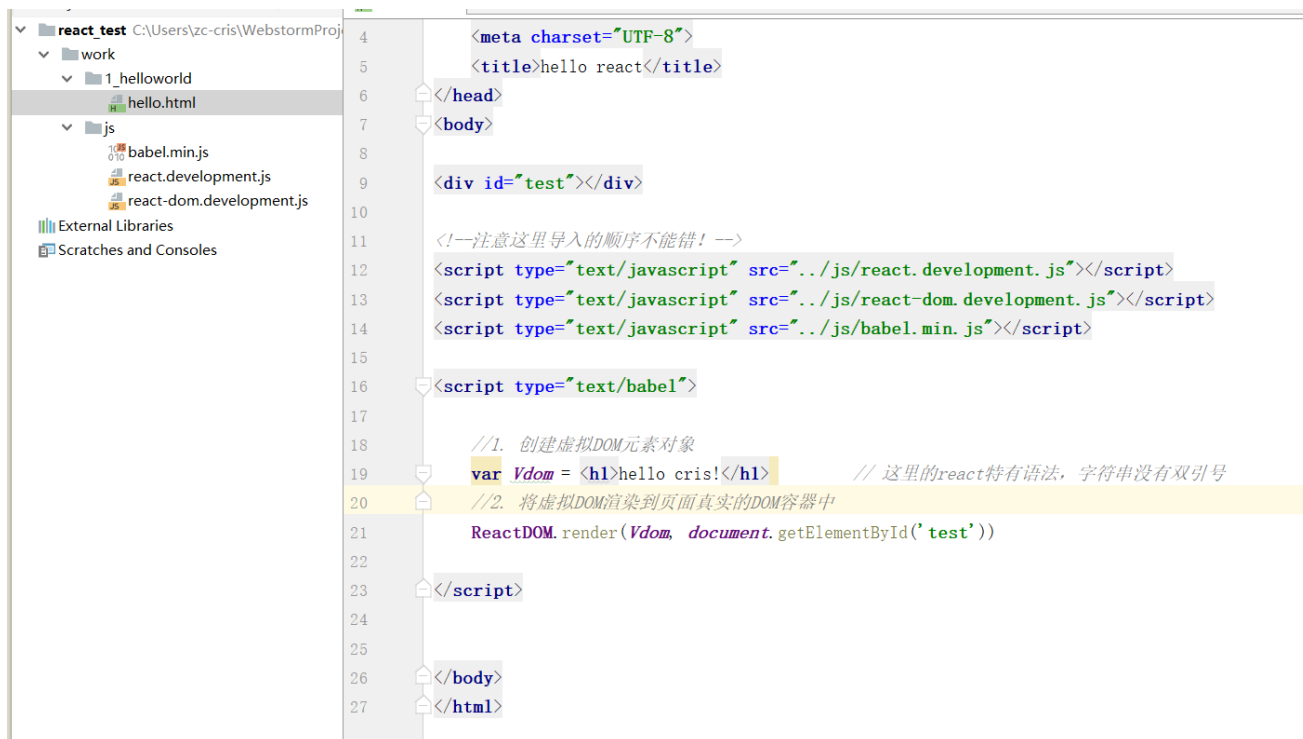


React 学习日记

1. hello world

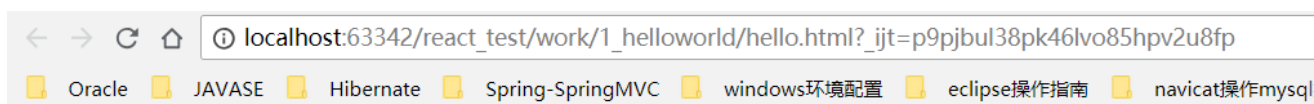
首先下载webstorm，然后激活（百度），最后导入React的三个基本依赖，书写一个hello world



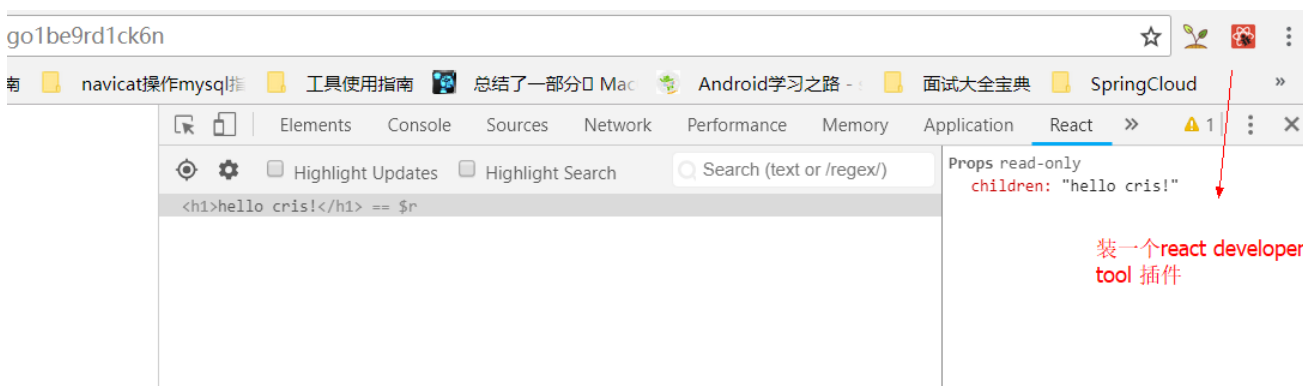
The screenshot shows the WebStorm IDE interface. On the left, the file explorer displays the project structure: `react_test` (root) contains `work` (folder), which contains `1_helloworld` (folder), which contains `hello.html` (file) and `js` (folder). The `js` folder contains `babel.min.js`, `react.development.js`, and `react-dom.development.js`. The `hello.html` file is open in the editor, showing the following code:

```
4 <meta charset="UTF-8">
5 <title>hello react</title>
6 </head>
7 <body>
8
9 <div id="test"></div>
10
11 <!--注意这里导入的顺序不能错! -->
12 <script type="text/javascript" src="../js/react.development.js"></script>
13 <script type="text/javascript" src="../js/react-dom.development.js"></script>
14 <script type="text/javascript" src="../js/babel.min.js"></script>
15
16 <script type="text/babel">
17
18 //1. 创建虚拟DOM元素对象
19 var Vdom = <h1>hello cris!</h1> // 这里的react特有语法，字符串没有双引号
20 //2. 将虚拟DOM渲染到页面真实的DOM容器中
21 ReactDOM.render(Vdom, document.getElementById('test'))
22
23 </script>
24
25 </body>
26 </html>
```

页面显示如下：



hello cris!



2. jsx语法

```
7 <body>
8 <div id="test1"></div>
9 <script type="text/javascript" src="../js/react.development.js"></script>
10 <script type="text/javascript" src="../js/react-dom.development.js"></script>
11 <script type="text/javascript" src="../js/babel.min.js"></script>
12
13 <script type="text/babel">
14   const a = "cris"
15   const b = "haha"
16   //1. 创建虚拟dom, 变量需要使用{}引起来
17   const Vdom = <h2 id={a.toUpperCase()}>{b.toUpperCase()}</h2>
18   //2. 渲染虚拟dom: 将虚拟dom对象添加到哪个真实dom对象中
19   ReactDOM.render(Vdom, document.getElementById("test1"))
20
21
22 </script>
```

测试页面

localhost:63342/react_test/work/2_jsx/test.html?_ijt=j05a64v3jvknf7kgcbadfrt15f

HAHA

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>test jsx</title>
  </head>
  <body>
    <div id="test1">
      <h2 id="CRIS">HAHA</h2>
    </div>
    <script type="text/javascript" src="../js/react.development.js"></script>
    <script type="text/javascript" src="../js/react-dom.development.js"></script>
    <script type="text/javascript" src="../js/babel.min.js"></script>
    <script type="text/babel">...</script>
    <div class="yformater-layer">...</div>
  </body>
</html>
```

2.1 jsx语法实战：将一个数组里的内容通过jsx语法转换为列表格式

```
0  </neau>
7  <body>
8  <h2>i like react! </h2>
9  <div id="test">
10 </div>
11
12 <script type="text/javascript" src="../js/react.development.js"></script>
13 <script type="text/javascript" src="../js/react-dom.development.js"></script>
14 <script type="text/javascript" src="../js/babel.min.js"></script>
15
16 <script type="text/babel">
17   const array = ["pig", "rabbit", "tomato", "beaf"]
18   // 创建虚拟dom
19   const Vdom = (
20     <ul>
21       {array.map((name, index) => <li>{name.toUpperCase()}</li>)}
22     </ul>
23   )
24
25   // 渲染虚拟dom
26   ReactDOM.render(Vdom, document.getElementById("test"))
27 </script>
28
29
```

引入react 标签；注意顺序

一定要用babel解析

这里的格式比较怪异：
1. 首先使用jsx语法，引用变量需要{}
2. 数组的map函数（这里特别像java8的stream API）
3. 箭头函数（其实就是java8的lambda表达式）

步骤都是固定的：
1. 创建虚拟dom
2. 渲染虚拟dom

- 这里可能比较晦涩的一点是：react的jsx语法相当于融合和h5的标签以及js代码于一块，需要适应一下

i like react!

- PIG
- RABBIT
- TOMATO
- BEAF

3. React 面向组件编程

面向对象编程--》面向模块编程--》面向组件编程

3.1 关于React中面向组件编程的两种方式（工厂函数和继承类）

```

7 <body>
8 <div id="test1"></div>
9 <div id="test2"></div>
10
11 <script type="text/javascript" src="../js/react.development.js"></script>
12 <script type="text/javascript" src="../js/react-dom.development.js"></script>
13 <script type="text/javascript" src="../js/babel.min.js"></script>
14
15 <script type="text/babel">
16   //1. 自定义创建组件标签
17   /*1.1 简单函数创建组件标签*/
18   function MyComponent1() {
19     return <h2>hello,cris,i am component1</h2>
20   }
21
22   /*1.2 复杂方式创建组件标签（继承）*/
23   class MyComponent2 extends React.Component{
24
25     render(){
26       return <h3>hello,cris, i am component2</h3>
27     }
28   }
29
30   //2. 渲染组件标签
31   ReactDOM.render(<MyComponent1/,>, document.getElementById("test1"))
32   ReactDOM.render(<MyComponent2/,>, document.getElementById("test2"))
33
34 </script>

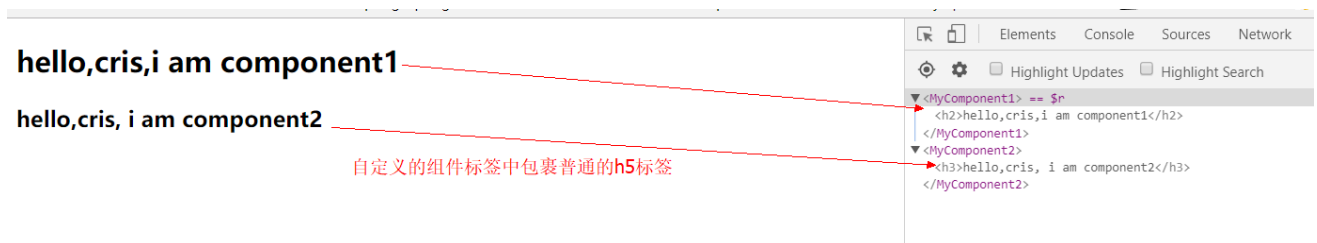
```

第一种方式：通过我们自定义的函数来构建React中的组件标签

第二种方式：通过继承React的内置对象Component来构建React的组件标签

渲染方式比较特别

测试效果：



3.2 组件Component的三大属性：state

- 实战案例：根据用户点击动态切换文本

```

1 <body>
2 <div id="test"></div>
3
4 <script type="text/javascript" src="../js/react.development.js"></script>
5 <script type="text/javascript" src="../js/react-dom.development.js"></script>
6 <script type="text/javascript" src="../js/babel.min.js"></script>
7
8 <script type="text/babel">
9   /*
10    自定义组件：显示h2 类型的文本：设置一个初始文本a；当我们点击这个文本的时候，显示另外的文件内容b，
    根据点击动态变更文本内容
11    */
12
13    //1. 创建自定义的组件标签（推荐使用继承的方式，如果没有状态state，可以使用函数形式创建组件标签）
14    class MyComponent extends React.Component {
15
16      //一、初始化状态state（注意：state是一个对象）

```

```

17     constructor(props) {
18         super(props)
19         this.state = {
20             isOk: false
21         }
22
23         // 将新增的方法的this强制指向为组件对象(类似java中的装饰者模式, 将handleClick函数重新
装饰, 我们点击事件绑定的函数
24         // 其实是装饰后的新handleClick函数)
25         this.handleClick = this.handleClick.bind(this)
26     }
27
28     // 我们自定义点击函数 (默认this不是组件对象, 而是undefined)
29     handleClick() {
30         console.log('handleClick()', this)
31         // 得到状态并取反
32         const isOk = !this.state.isOk
33         // 三、更新状态(es6的新语法, 根据对象的属性名直接赋值)
34         this.setState({isOk})
35     }
36
37     // 重写组件类的方法
38     render() {
39         //二、读取状态值
40         // const isOk = this.state.isOk
41         const {isOk} = this.state //结构赋值: es6的语法 (对于后端开发人员确实有点
怪异)
42         return <h2 onClick={this.handleClick}>{isOk ? "你觉得ok吗?" : "我觉得还ok啊!"}
</h2>
43     }
44 }
45
46 ReactDOM.render(<MyComponent/>, document.getElementById("test"))
47
48 </script>
49 </body>

```

- 实战案例顺序:

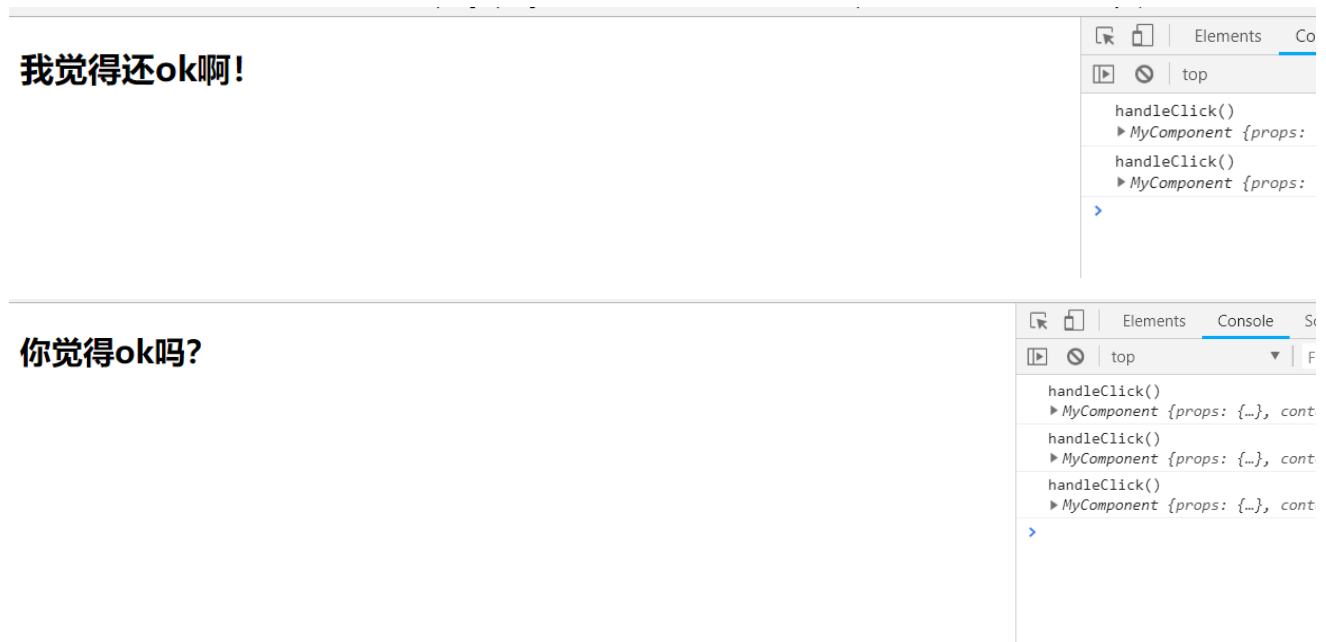
1. 通过继承自定义我们的component标签
 - 在render方法里面定义h5标签, 同时绑定监听事件 (onClick), 每次点击都会进行重绘 (调用render方法) 获取状态state对象的属性值, 根据属性值的变换动态改变显示文本 (三元运算符)
 - 注意在构造函数中将自定义的onClick函数的this改为当前的component对象
2. 渲染我们的component标签
3. 关于state的解析顺序: 初始化state对象; 读取state对象的状态值, 根据这个状态值的不同进行组件渲染; 写一个函数 (例如点击函数) 动态修改state对象的状态值;

- 总结:

1. 点击事件以及绑定的点击函数

2. 点击函数的binder装饰
3. jsx语法中{}还可以写三目运算符等进行判断

- 效果图：



3.3 组件Component的三大属性： props

- 实战案例：显示外界传来的数据并做限制

```
1 <body>
2 <div id="test"></div>
3 <div id="test2"></div>
4
5 <script type="text/javascript" src="../../js/react.development.js"></script>
6 <script type="text/javascript" src="../../js/react-dom.development.js"></script>
7 <script type="text/javascript" src="../../js/prop-types.js"></script>
8 <script type="text/javascript" src="../../js/babel.min.js"></script>
9
10 <script type="text/babel">
11
12   /*需求：
13    * 1. 动态显示传入的参数的数据
14    * 2. 对参数的数据进行默认值设置
15    * 3. 对参数的数据类型做出限制
16    * 3. 对参数的数据是否必要做出限制
17    * */
18   //1. 定义组件（函数的形式）
19   /* function Person(props) {
20     return (
21       <ul>
22         <li>{props.name}</li>
23         <li>{props.age}</li>
```

```

24         <li>{props.sex}</li>
25     </ul>
26 )
27 }*/
28 //1. 定义组件 (对象继承的方式)
29 class Person extends React.Component{
30
31     render(){
32         return (
33             <ul>
34                 <li>{this.props.name}</li>
35                 <li>{this.props.age}</li>
36                 <li>{this.props.sex}</li>
37             </ul>
38         )
39     }
40 }
41
42 // 为组件标签设置props属性对象的键的数据格式和是否必须做出限制
43 Person.propTypes = {
44     name: PropTypes.string.isRequired,
45     age: PropTypes.number.isRequired,
46     sex: PropTypes.string
47 }
48 // 为组件标签的props对象设置默认的键值
49 Person.defaultProps = {
50     name: '老张',
51     age: 19
52 }
53
54 const p1 = {
55     name: 'cris',
56     age: 12,
57     sex: '男'
58 }
59 const p2 = {
60     sex: '女'
61 }
62
63 /*...符号的作用
64 * 1.打包: function fn(...as) {} fn(1,2,3)    将1,2,3打包成数组as
65 * 2.解包: const arr1 = [1,2,3] const arr2 = [4, ...arr1, 6] 将arr1解包成一个个参数放入到
arr2中
66 * */
67 //2. 渲染组件
68 //ReactDOM.render(<Person name={p1.name} age={p1.age} sex={p1.sex}/>,
document.getElementById("test"))
69 // es6 的语法糖 (解包和装包符号 ...) 通过{...p1}的格式将p1这个对象的键值对封装到Person这个自
定义标签的props属性对象中
70 // 关于语法糖: 让程序员吃起来很甜的语法 (简洁); 但是理解起来确实有点抽象; 为了代码整洁, 极简而定
义的语法仁者见仁, 智者见智 (个人柑橘很厉害)
71 ReactDOM.render(<Person {...p1}/>, document.getElementById("test"))
72 ReactDOM.render(<Person {...p2}/>, document.getElementById("test2"))

```

```
73
74
75 </script>
76 </body>
```

- 测试

- cris
- 12
- 男

- 老张
- 19
- 女

即使后台不传递数据，也可以通过默认props设置键值对的值

- 总结

1. 两种方式（函数和继承）将外界传来的参数封装为自定义标签的props属性对象
2. ... 符号的快速使用：解包和装包
3. 为自定义标签的props属性设置默认的键值对和对props属性的键值对的值做出类型和是否必要的限制

3.4 组件Component的三大属性：refs（以及事件对象event）

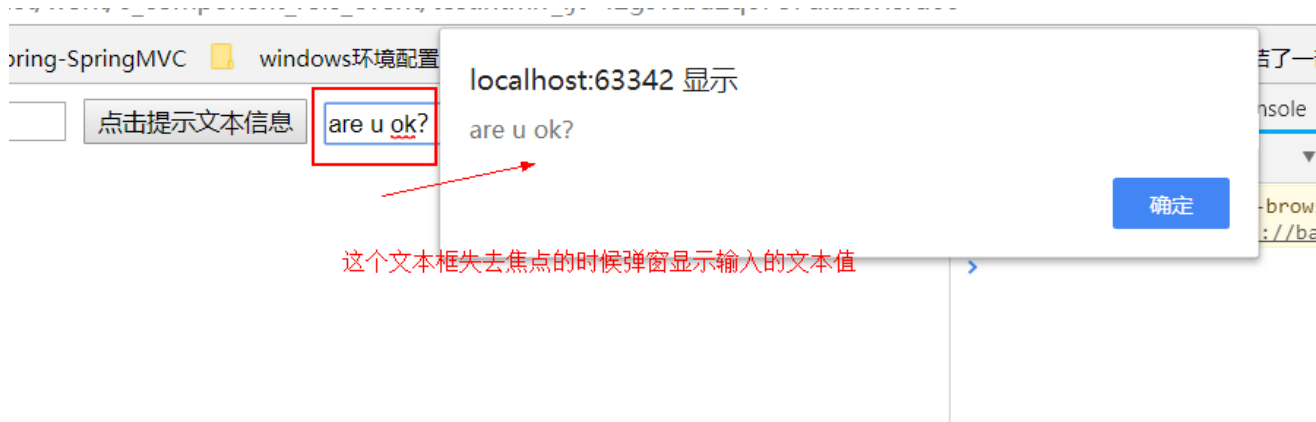
- 实战案例：点击按钮显示文本内容和文本框失去焦点事件

```
1 <body>
2
3 <div id="test"></div>
4 <script type="text/javascript" src="../js/react.development.js"></script>
5 <script type="text/javascript" src="../js/react-dom.development.js"></script>
6 <script type="text/javascript" src="../js/prop-types.js"></script>
7 <script type="text/javascript" src="../js/babel.min.js"></script>
8
9 <!-- 实战案例:
10     1. 自定义组件，点击按钮，输入第一个文本框中的已输入文本信息
11     2. 当第二个文本框失去焦点的时候，提示这个文本框已输入的文本信息
12 -->
13 <script type="text/babel">
14     class MyComponent extends React.Component{
15
16         constructor(props){
17             super(props)
18             this.showInput = this.showInput.bind(this)
19             this.handleBlur = this.handleBlur.bind(this)
20         }
21     }
```


[illegible]

- 测试效果





- 总结

1. ref (refs) 主要是用于自定义组件中的html标签的事件函数（例如点击事件绑定函数）对组件中的其他html标签做出修改操作
2. event 主要是用于自定义组件中的html标签的事件绑定函数对该html标签做出的改变操作

4. 综合案例实战（动态显示用户添加的列表数据）

4.1 实战要求

- 显示所有todo列表
- 输入文本, 点击按钮显示到列表的首位, 并清除输入的文本

4.2 开发步骤

1. 拆分组件: 拆分界面, 抽取组件
2. 实现静态组件: 使用组件实现静态页面效果
3. 实现动态组件

4.3 分析显示数据的存放位置

我们需要思考, 动态显示的页面数据到底应该放在哪个自定义的组件中?

如果只有一个组件需要这个数据, 我们可以放在这一个组建中

如果有多个组件需要这个数据, 建议方法他们公共的父组件中

4.4 开发流程

1. 先将页面的效果静态显示出来

```
1 <body>
2 <div id="test"></div>
3
4 <script type="text/javascript" src="../js/react.development.js"></script>
```

```
5 <script type="text/javascript" src="../js/react-dom.development.js"></script>
6 <script type="text/javascript" src="../js/prop-types.js"></script>
7 <script type="text/javascript" src="../js/babel.min.js"></script>
8
9 <script type="text/babel">
10
11 class App extends React.Component {
12
13     render() {
14         return (
15             <div>
16                 <h1>动态显示列表内容</h1>
17                 <Content/>
18                 <List/>
19             </div>
20         )
21     }
22 }
23
24 class Content extends React.Component {
25
26     render(){
27         return (
28             <div>
29                 <input type="text" />
30                 <button>请点击以显示您添加的列表, add #3</button>
31             </div>
32         )
33     }
34
35 }
36
37 class List extends React.Component {
38
39     render(){
40         return (
41             <div>
42                 <ul>
43                     <li>打球</li>
44                     <li>看NBA</li>
45                     <li>学习</li>
46                 </ul>
47             </div>
48         )
49     }
50 }
51
52 // 渲染组件标签
53 ReactDOM.render(<App/>, document.getElementById("test"))
54
55 </script>
56 </body>
```

效果图:

动态显示列表内容

请点击以显示您添加的列表, add #3

- 打球
- 看NBA
- 学习

2. 完成显示数据的初始化

```
1  class App extends React.Component {
2
3      constructor(props){
4          super(props)
5          this.state = {
6              /*js中的数组创建方式*/
7              toDoList: ['吃饭', '睡觉', '逛街']
8          }
9      }
10
11     render() {
12         /*将重复的代码抽取出来*/
13         const {toDoList} = this.state
14         return (
15             <div>
16                 <h1>动态显示列表内容</h1>
17                 <Content count={toDoList.length}/>
18                 /*将初始化数据传递给子组件的props属性对象*/
19                 <List toDoList={toDoList}/>
20             </div>
21         )
22     }
23 }
24
25 class Content extends React.Component {
26
27     render(){
28         return (
29             <div>
30                 <input type="text" />
31                 <button>请点击以显示您添加的列表, add #{this.props.count +1}</button>

```

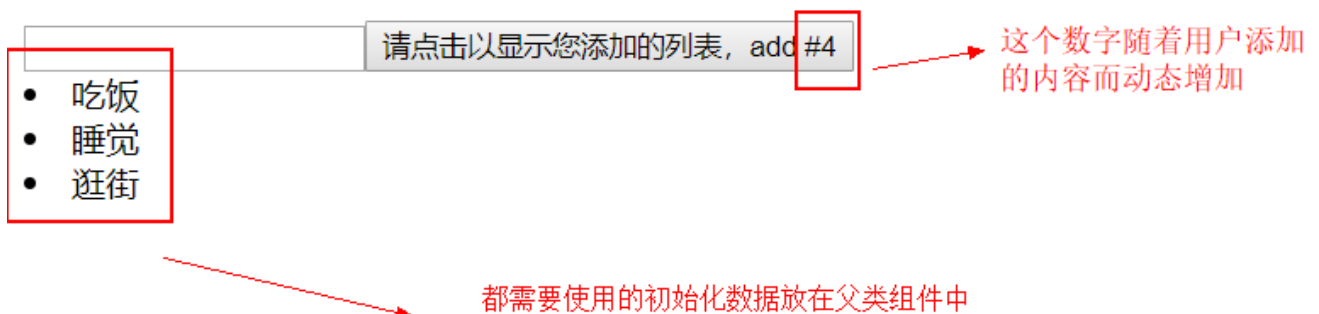
```

32         </div>
33     )
34 }
35
36 }
37 Content.propTypes = {
38     count: PropTypes.number.isRequired
39 }
40
41 class List extends React.Component {
42
43     render(){
44         /*根据props属性对象的key进行结构赋值*/
45         const {todoList} = this.props
46         return (
47             <div>
48                 /*js代码在React中一定要用{}包裹，操作数组中的数据一般都需要使用箭头函数*/
49                 {
50                     todoList.map((todo,index) => <li key={index}>{todo}</li>)
51                 }
52                 <ul>
53                 </ul>
54             </div>
55         )
56     }
57 }
58
59 /*子组件需要对从外界传入的数据做出限制*/
60 List.propTypes = {
61     todoList: PropTypes.array.isRequired
62 }

```

显示效果

动态显示列表内容



3. 完成动态效果

3.1 注意:

哪个组件的状态state需要更新，那么这个更新函数就需要放在哪个组件中，本案例的父组件的state属性对象有一个数组属性，子组件需要动态更新这个数组，就需要调用父组件的更新函数

3.2 代码

```
1      class App extends React.Component {
2
3          constructor(props){
4              super(props)
5              this.state = {
6                  /*js中的数组创建方式*/
7                  toDoList: ['吃饭', '睡觉', '逛街']
8              }
9              // 自定义的方法需要修改this的指向
10             this.addToDoList = this.addToDoList.bind(this)
11         }
12
13         /*父组件定义的state属性对象更新方法*/
14         addToDoList(todo){
15             const {toDoList} = this.state
16             /*将新添加的数据放入数组的第一个位置，记住：js中的数组的长度是可变的*/
17             toDoList.unshift(todo)
18             /*一定要使用setSate方法更新*/
19             this.setState({toDoList})
20         }
21
22         render() {
23             /*将重复的代码抽取出来*/
24             const {toDoList} = this.state
25             return (
26                 <div>
27                     <h1>动态显示列表内容</h1>
28                     /*不仅可以将父组件的state属性对象的属性传递给子组件，函数同理*/
29                     <Content count={toDoList.length} addToDoList={this.addToDoList}/>
30                     /*将初始化数据传递给子组件的props属性对象*/
31                     <List toDoList={toDoList}/>
32                 </div>
33             )
34         }
35     }
36
37     class Content extends React.Component {
38
39         constructor(props){
40             super(props)
41             this.addToDoList = this.addToDoList.bind(this)
42         }
43
44         addToDoList(){
45             //1. 读取输入的数据
46             const value = this.inputToDo.value.trim()
47             //2. 检查数据的合法性
48             if (!value){
```

```

49         alert("数据不能为空")
50         return
51     }
52     //3. 调用父组件的函数进行数据的添加
53     this.props.addToDoList(value)
54     // 4. 清除用户输入数据
55     this.inputToDo.value = ''
56 }
57
58 render(){
59     return (
60         <div>
61             { /*使用箭头函数将当前input标签赋值给组件的属性*/ }
62             <input type="text" ref={(input) => (this.inputToDo = input)} />
63             { /*自定义的函数绑定监听事件*/ }
64             <button onClick={this.addToDoList}>请点击以显示您添加的列表, add #
65 {this.props.count + 1}</button>
66         </div>
67     )
68 }
69
70 Content.propTypes = {
71     count: PropTypes.number.isRequired,
72     addToDoList: PropTypes.func.isRequired
73 }

```

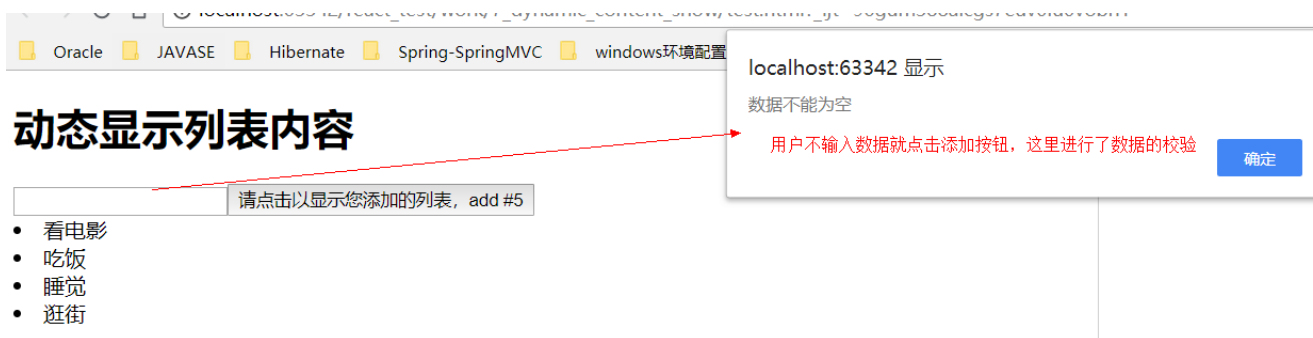
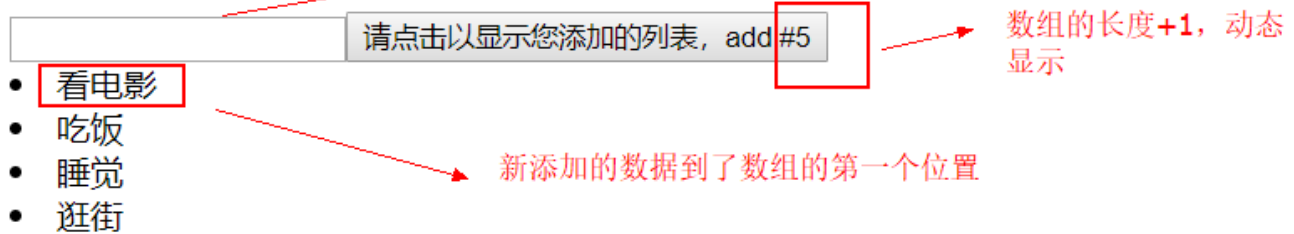
4.5 最终显示效果

动态显示列表内容



动态显示列表内容

添加数据成功则清除文本框的内容



4.6 组件化编程总结

1. 拆分组件
2. 实现静态组件 (只有静态界面, 没有动态数据和交互)
3. 实现动态组件
 1. 实现初始化数据动态显示 (初始化数据到底放在哪个组件? 这个问题很重要, 涉及到初始化数据的更新函数的开发位置)
 2. 实现交互功能 (事件监听函数的实现)

5. 收集表单数据 (受控组件和非受控组件两种方式)

5.1 代码

```
1 <div id="test"></div>
2
3 <script type="text/javascript" src="../../js/react.development.js"></script>
4 <script type="text/javascript" src="../../js/react-dom.development.js"></script>
5 <script type="text/javascript" src="../../js/prop-types.js"></script>
6 <script type="text/javascript" src="../../js/babel.min.js"></script>
7
8 <script type="text/babel">
9
10 /*案例要求: 用户输入用户名和密码后, 点击提交按钮, 界面阻止提交动作并弹窗用户输入的内容*/
```

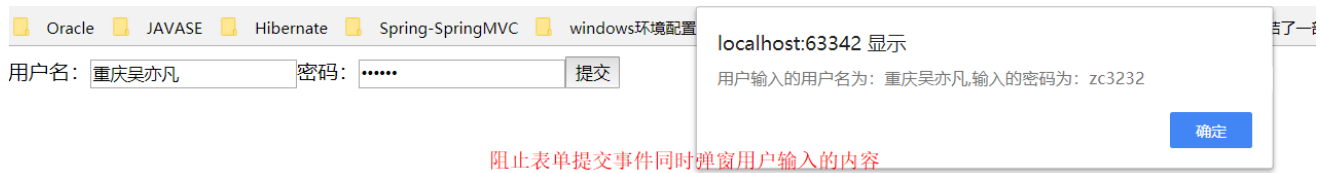


```

11 class LoginForm extends React.Component {
12
13     constructor(props){
14         super(props)
15         /*第二种方法: 受控组件*/
16         this.state = {
17             pwd: ''
18         }
19         // 自定义的函数一定首先要改变this的指向
20         this.handleSubmit = this.handleSubmit.bind(this)
21         this.handleChange = this.handleChange.bind(this)
22     }
23
24     /*第一种方法: 非受控组件*/
25     handleSubmit(event){
26         const nameInput = this.nameInput.value
27         const {pwd} = this.state
28         /*注意: 这里使用了模板输入语句, 可以直接输出变量内容 (${变量名}), 类似于java中lombok
29         的@S1f4j的作用*/
30         alert(`用户输入的用户名为: ${nameInput}, 输入的密码为: ${pwd}`)
31         // alert(nameInput)
32         // 阻止表单的默认提交事件
33         event.preventDefault()
34     }
35     /*第二种方法: 受控组件*/
36     handleChange(event){
37         // 拿到当前发生事件的组件(target)的值
38         const pwd = event.target.value
39         // 更新到自定义标签类的初始化state中去
40         this.setState({pwd})
41     }
42
43     render (){
44         return (
45             <form action="login" onSubmit={this.handleSubmit}>
46                 用户名: <input type="text" ref={input => this.nameInput = input}/>
47                 密码: <input type="password" value={this.state.pwd} onChange=
48                 {this.handleChange}/>
49                 <input type="submit"/>
50             </form>
51         )
52     }
53
54     ReactDOM.render(<LoginForm/>, document.getElementById("test"))
55
56 </script>
57 </body>

```

5.2 实现效果



5.3 关于受控组件和非受控组件

都是包含表单数据的组件，但是

受控组件：表单项输入数据能够自动收集成状态state（上面的密码输入框为例）

非受控组件：需要时才手动读取表单输入框的数据（上面的用户名输入框为例）

6. 组件生命周期

6.1 实战案例：完成渐变动画效果（定时器）以及自定义组件的销毁（组件的生命周期）

1. 让指定的文本做显示/隐藏的渐变动画
2. 切换持续时间为2S
3. 点击按钮从界面中移除组件界面

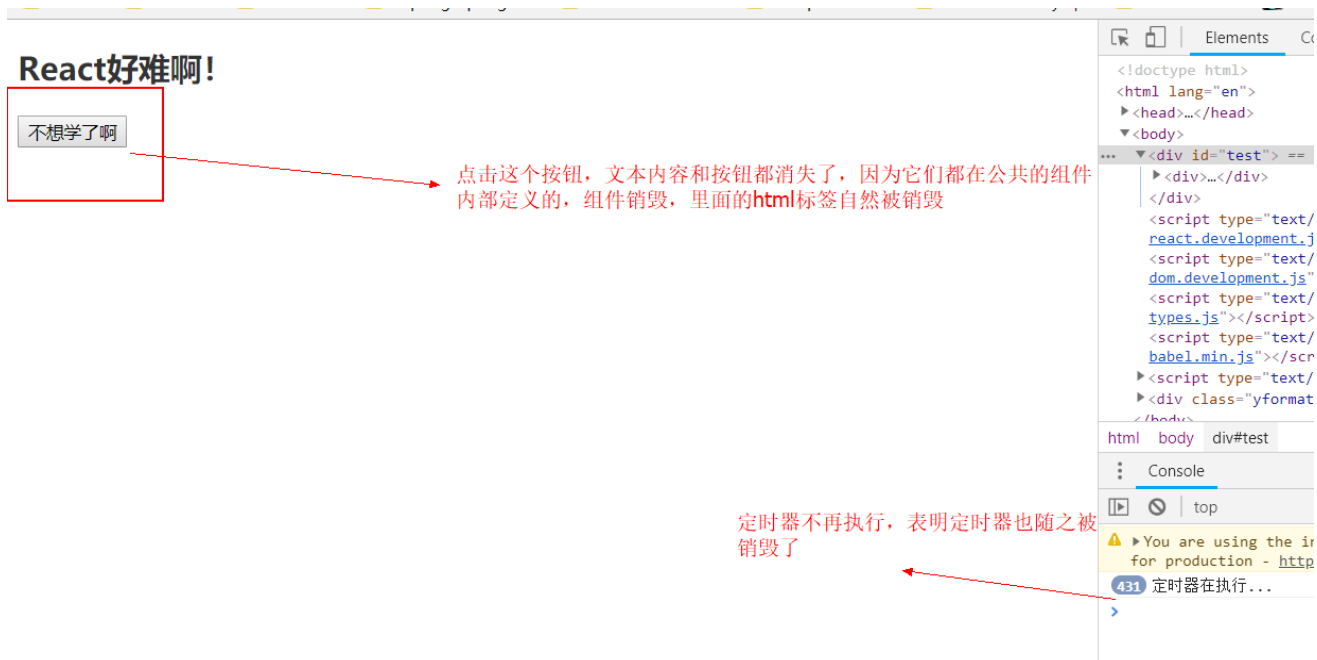
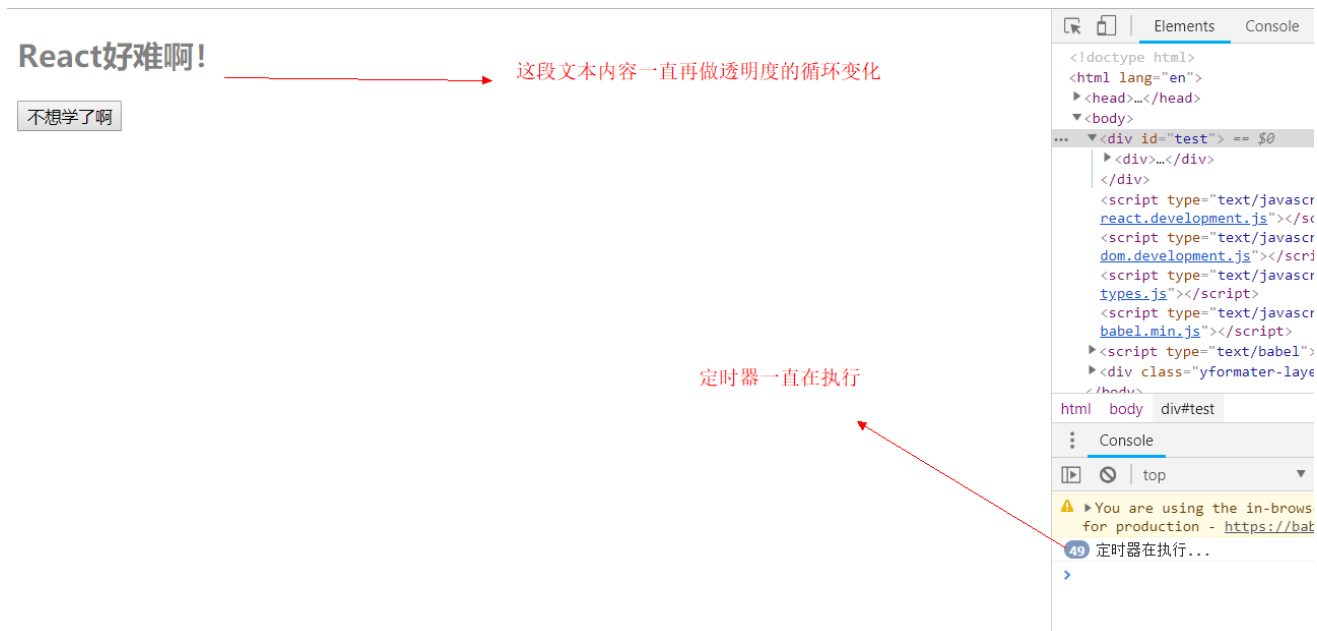
```
1 <body>
2 <div id="test"></div>
3 <script type="text/javascript" src="../js/react.development.js"></script>
4 <script type="text/javascript" src="../js/react-dom.development.js"></script>
5 <script type="text/javascript" src="../js/prop-types.js"></script>
6 <script type="text/javascript" src="../js/babel.min.js"></script>
7 <script type="text/babel">
8   class MyComponent extends React.Component{
9     constructor(props){
10       super(props)
11       this.state = {
12         opacity:1
13       }
14       this.destroyComponenet = this.destroyComponenet.bind(this)
15     }
16     /*
17     自定义组件的生命周期方法，用于定义该组件已经挂载完毕之后执行的方法，即render方法渲染后
18     */
19     componentDidMount(){
20       /*这里是启动一个循环定时器，用于不停的重复变换文本内容的透明度*/
21       this.intervalId = setInterval(function () {
22         console.log("定时器在执行...")
```

```

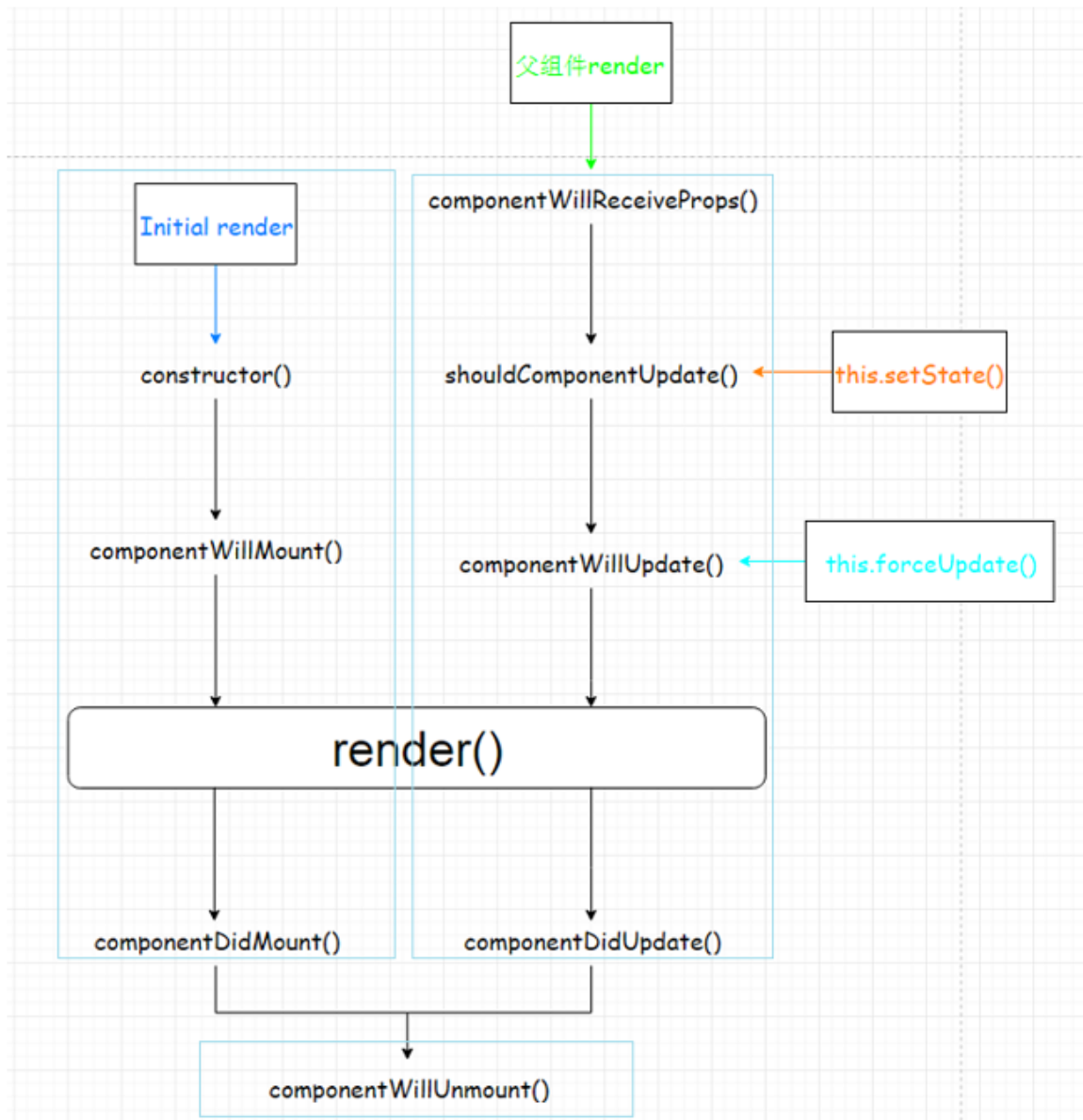
23      /*注意这里不能使用const，因为要每隔200毫秒减少组件的透明度，所以使用let关键字*/
24      let {opacity} = this.state
25      opacity -= 0.1
26      if (opacity <= 0){
27          opacity = 1
28      }
29      this.setState({opacity})
30      }.bind(this),200) //setInterval回调函数默认this是window对
象，所以需要更换this指向；React的组件生命周期函数中的this默认指向当前组件对象
31    }
32
33    /*点击按钮执行自定义组件的销毁流程*/
34    destroyComponenet(){
35        // 调用React的方法来销毁组件
36        ReactDOM.unmountComponentAtNode(document.getElementById("test"))
37    }
38
39    /*销毁组件之前需要进行的操作（销毁定时器，防止内存泄漏（很重要））*/
40    componentWillUnmount(){
41        // 清理定时器
42        clearInterval(this.intervalId)
43    }
44
45    render(){
46        const {opacity} = this.state
47        return (
48            <div>
49                <h2 style={{opacity: opacity}}>{this.props.msg}</h2>
50                <button onClick={this.destroyComponenet}>不想学了阿</button>
51            </div>
52        )
53    }
54 }
55 /*渲染自定义组件时传入的属性自动封装到props属性中*/
56 ReactDOM.render(<MyConponent msg="React好难啊！"/>, document.getElementById("test"))
57
58 </script>
59 </body>

```

6.2 实战效果



6.3 生命周期流程图



6.4 生命周期流程图详解

1) 组件的三个生命周期状态:↵

* Mount: 插入真实 DOM↵

* Update: 被重新渲染↵

* Unmount: 被移出真实 DOM↵

2) React 为每个状态都提供了钩子(hook)函数↵

* componentWillMount()↵

* componentDidMount()↵

* componentWillUpdate()↵

* componentDidUpdate()↵

* componentWillUnmount()↵

3) 生命周期流程:↵

a. 第一次初始化渲染显示: ReactDOM.render()↵

* constructor(): 创建对象初始化 state↵

* componentWillMount(): 将要插入回调↵

* render(): 用于插入虚拟 DOM 回调↵

* componentDidMount(): 已经插入回调↵

b. 每次更新 state: this.setState()↵

* componentWillUpdate(): 将要更新回调↵

* render(): 更新(重新渲染)↵

* componentDidUpdate(): 已经更新回调↵

c. 移除组件: ReactDOM.unmountComponentAtNode(containerDom)↵

* componentWillUnmount(): 组件将要被移除回调↵

6.5 重要的钩子函数

▪ 2.7.5. 重要的勾子

- 1) `render()`: 初始化渲染或更新渲染调用
- 2) `componentDidMount()`: 开启监听, 发送 ajax 请求
- 3) `componentWillUnmount()`: 做一些收尾工作, 如: 清理定时器
- 4) `componentWillReceiveProps()`: 后面需要时讲

7. 虚拟DOM和DOM Diff算法

7.1 实战案例: 实时显示当前时间 (定时器)

```
1  <body>
2  <div id="test"></div>
3  <script type="text/javascript" src="../js/react.development.js"></script>
4  <script type="text/javascript" src="../js/react-dom.development.js"></script>
5  <script type="text/javascript" src="../js/prop-types.js"></script>
6  <script type="text/javascript" src="../js/babel.min.js"></script>
7  <script type="text/babel">
8      class MyComponent extends React.Component{
9
10         constructor(props){
11             super(props)
12             this.state = {
13                 date: new Date()
14             }
15         }
16
17         componentDidMount(){
18             // 箭头函数替代普通的回调函数, 非常方便的将函数内的this指向当前对象
19             setInterval(() =>{
20                 /*let {date} = this.state
21                 date = new Date()
22                 this.setState({date})*/
23                 this.setState({
24                     date: new Date()
25                 })
26             },1000)
27         }
28
29         render (){
```

```

30         return (
31             <div>
32                 <h3>请输入你的名字</h3>
33                 <input type="test" placeholder="输入内容不会影响其他部分"/>
34                 <span>
35                     现在当地时间是: {this.state.date.toLocaleTimeString()}
36                 </span>
37             </div>
38         )
39     }
40 }
41
42 ReactDOM.render(<MyComponent/>, document.getElementById("test"))
43 </script>
44 </body>

```

7.2 效果展示

请输入你的名字

现在当地时间是: 下午10:27:34

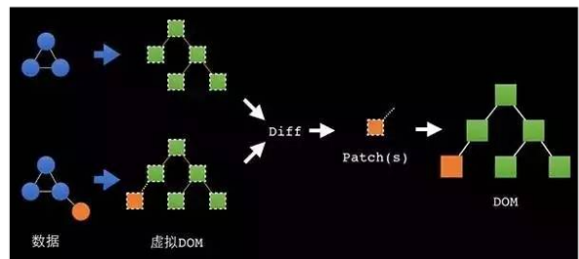
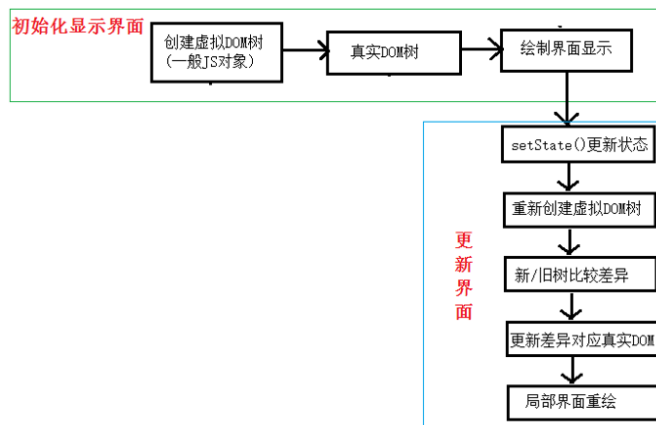


这里输入文本不会影响到其他部分的变换，例如时间显示部分

7.3 关于React 虚拟DOM为什么可以提高效率？

- 首先，React 的jsx语法上手比较有成本，但是一旦掌握，对于开发者的开发效率将会大大提升（因为我们不在操纵原生DOM对象，转而操作React 的虚拟DOM对象，相当于架起来一座桥梁）
- React 虚拟DOM的DOM Diff算法可以根据我们修改的虚拟DOM部分，去修改原生DOM的相关部分，不需要像以前那样全部重新渲染，开发人员只需要关注开发虚拟DOM变化的部分，渲染的部分交给DOM Diff算法，大大提高了渲染效率和开发效率

7.4 虚拟DOM Diff算法原理图



8. react 脚手架开发react 应用

8.1 下载react 脚手架

```

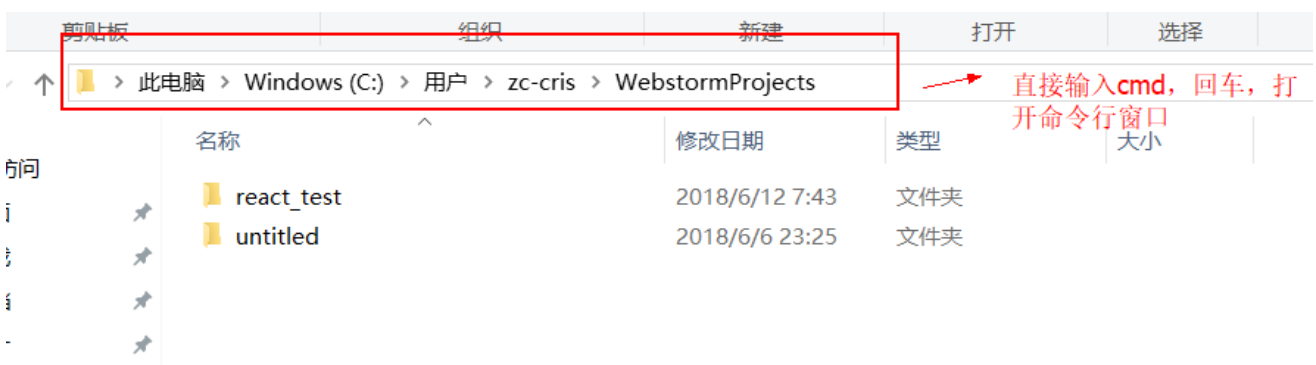
C:\Users\zc-cris\WebstormProjects\react_test>npm root -g
C:\Users\zc-cris\AppData\Roaming\npm\node_modules

C:\Users\zc-cris\WebstormProjects\react_test>npm install -g create-react-app
C:\Users\zc-cris\AppData\Roaming\npm\create-react-app -> C:\Users\zc-cris\AppData\Roaming\npm\node_modules\create-react-app\index.js
+ create-react-app@1.5.2
  
```

查看npm 的根目录

下载react 的脚手架

8.2 创建一个react应用



```

C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.17134.112]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\zc-cris\WebstormProjects>create-react-app react_app

Creating a new React app in C:\Users\zc-cris\WebstormProjects\react_app.

Installing packages. This might take a couple of minutes.
  
```

通过脚手架创建React 应用

11' 不是内部或外部命令，也不是可运行的程序或批处理文件。

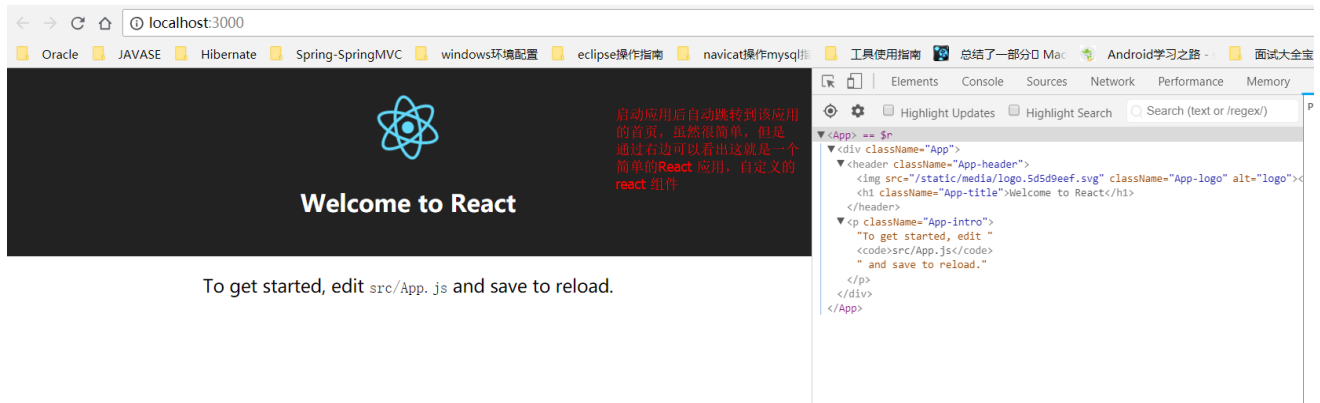
```
C:\Users\zc-cris\WebstormProjects>cd react_app
```

进入创建好的react 应用

```
C:\Users\zc-cris\WebstormProjects\react_app>npm start
```

通过命令直接启动

```
> react_app@0.1.0 start C:\Users\zc-cris\WebstormProjects\react_app
> react-scripts start
Starting the development server...
Compiled successfully!
```



Note that the development build is not optimized.
To create a production build, use `npm run build`.

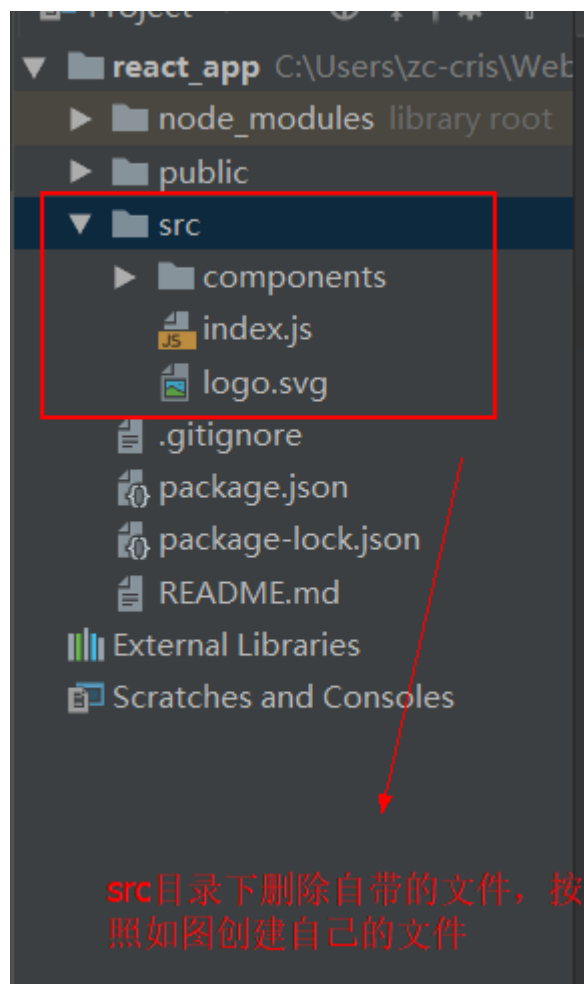
```
终止批处理操作吗 (Y/N)? y
```

通过命令行停止我们刚启动的React 应用只需要ctrl+c，然后输入y 即可

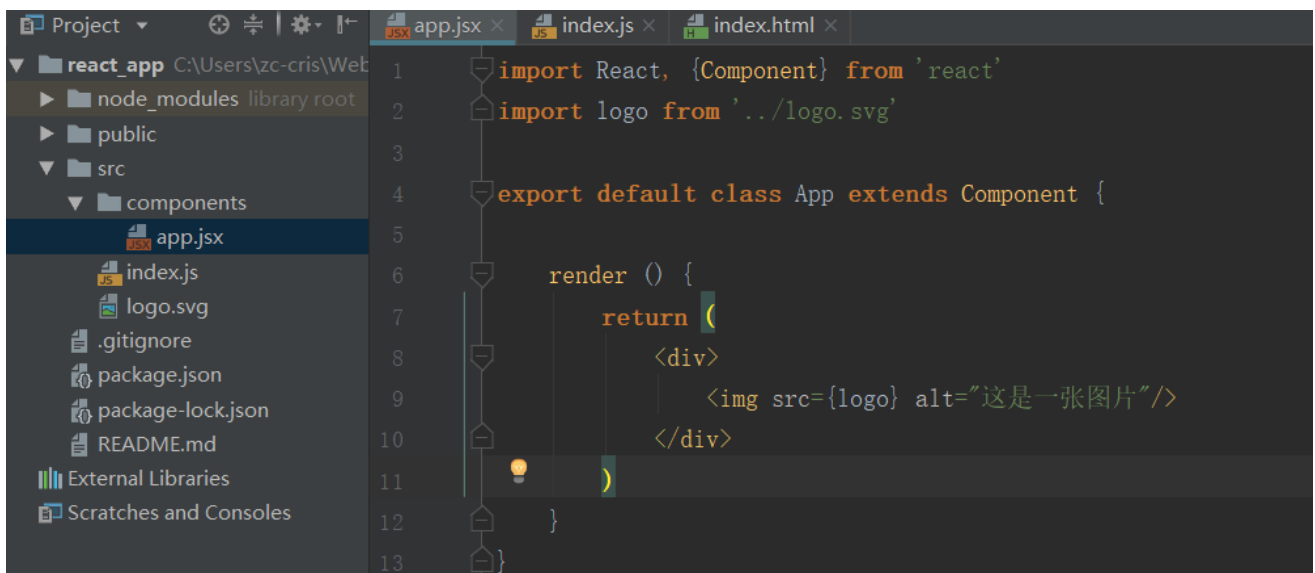
```
C:\Users\zc-cris\WebstormProjects\react_app>
```

8.3 初步开发我们自己的react 应用（极度重点）

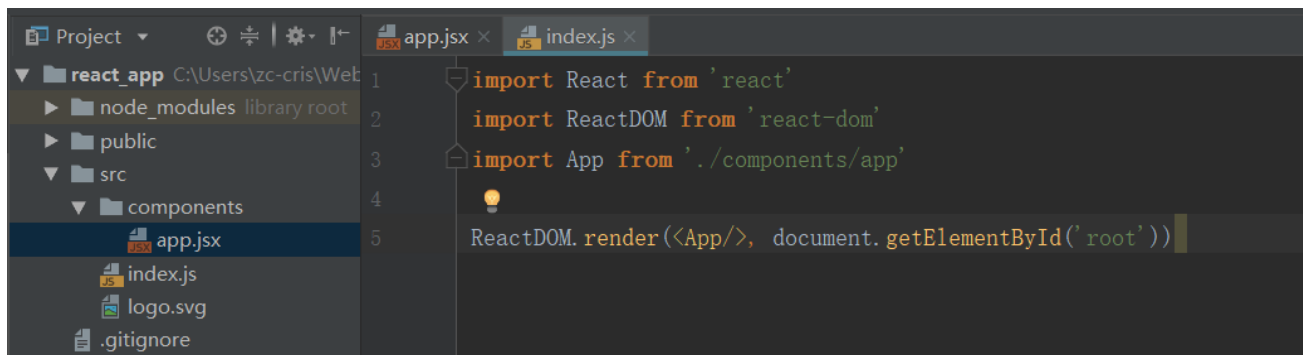
1. 首先使用webstorm 打开我们刚创建好的react应用



2. 开发我们自己的组件




3. index.js 引入我们的组件



The screenshot shows a code editor with a file explorer on the left and two open files, `app.jsx` and `index.js`, on the right. The file explorer shows the project structure: `react_app` (C:\Users\zc-cris\WebstormProjects) containing `node_modules` (library root), `public`, `src` (containing `components`), `app.jsx`, `index.js`, `logo.svg`, and `.gitignore`. The `app.jsx` file contains the following code:

```
1 import React from 'react'
2 import ReactDOM from 'react-dom'
3 import App from './components/app'
4
5 ReactDOM.render(<App/>, document.getElementById('root'))
```

4. 控制台启动



The screenshot shows a terminal window with the following output:

```
Microsoft Windows [版本 10.0.17134.112]
(c) 2018 Microsoft Corporation。保留所有权利。

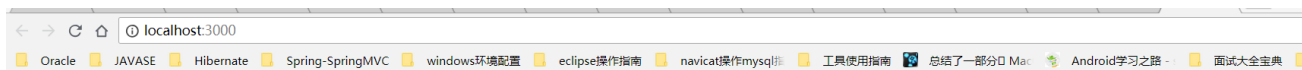
C:\Users\zc-cris\WebstormProjects\react_app>npm start

> react_app@0.1.0 start C:\Users\zc-cris\WebstormProjects\react_app
> react-scripts start

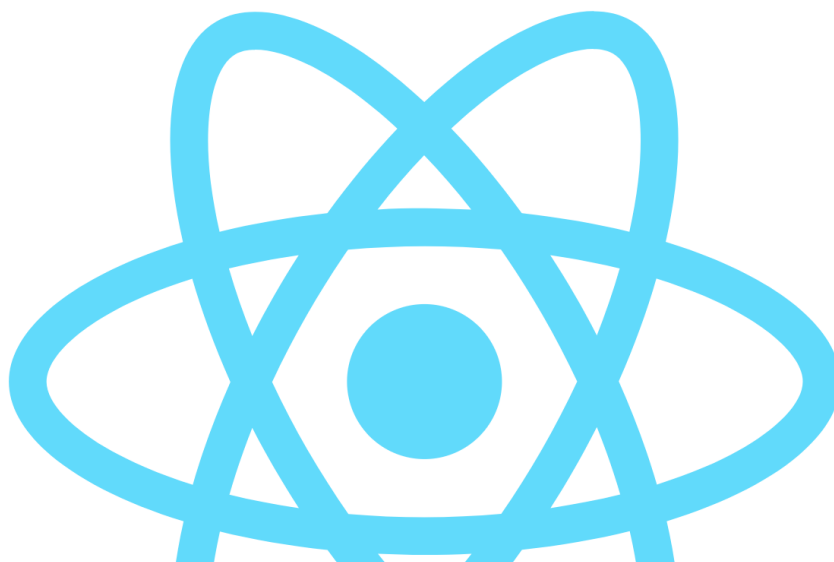
Starting the development server...

Compiled successfully!
```

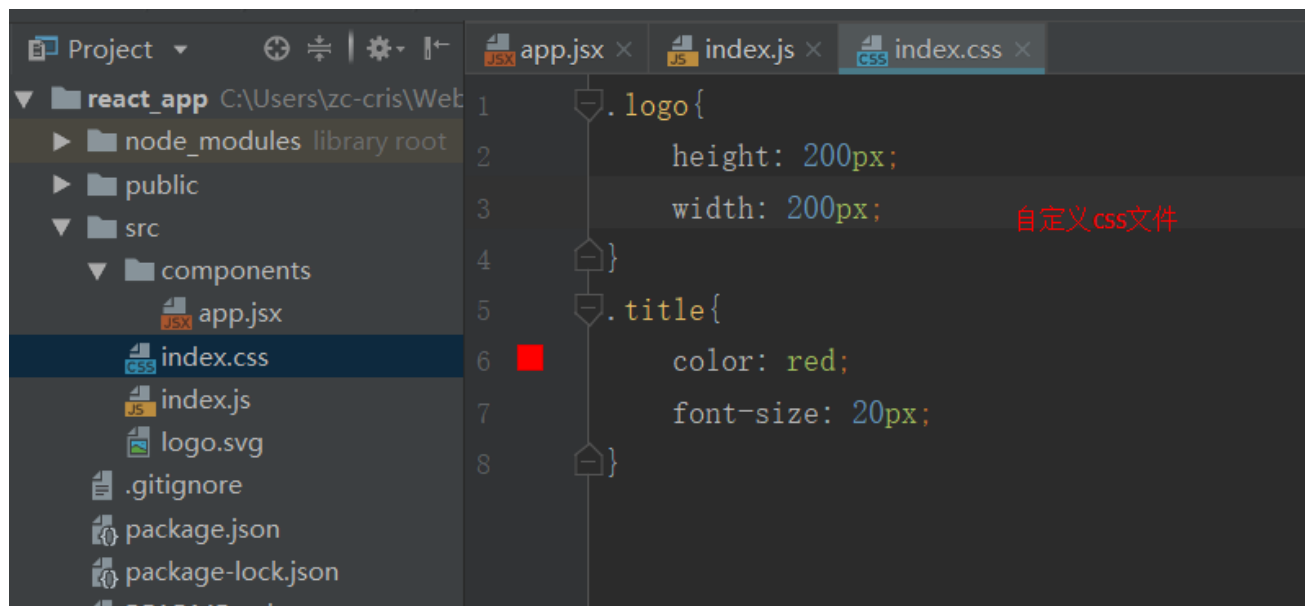
5. 效果展示



图片已经展示了出来!



6. 自定义css



```
app.jsx × index.js × index.css ×
1 import React, {Component} from 'react'
2 import logo from '../logo.svg'
3
4 export default class App extends Component {
5
6   render () {
7     return (
8       <div>
9         <img className='logo' src={logo} alt="这是一张图片"/>
10        <p className='title'>hello react</p>
11      </div>
12    )
13  }
14 }
```

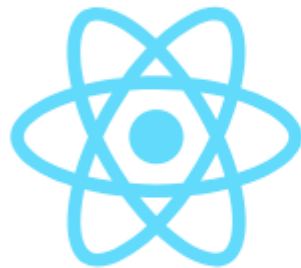
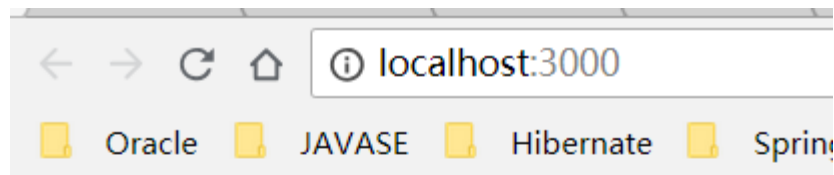
通过className 在jsx文件中使用样式

```
app.jsx × index.js × index.css ×
1 import React from 'react'
2 import ReactDOM from 'react-dom'
3 import App from './components/app'
4
5 import './index.css'
6
7 ReactDOM.render(<App/>, document.getElementById('root'))
```

一定还要在index.js 文件中引入样式文件

注意：
如果是引入第三方组件，直接输入引入的组件名以及组件文件名即可（文件后缀名脚手架自动识别）；
如果是引入我们自己的自定义组件或者样式文件，必须以./ 或者../ 开头，切记

效果图：

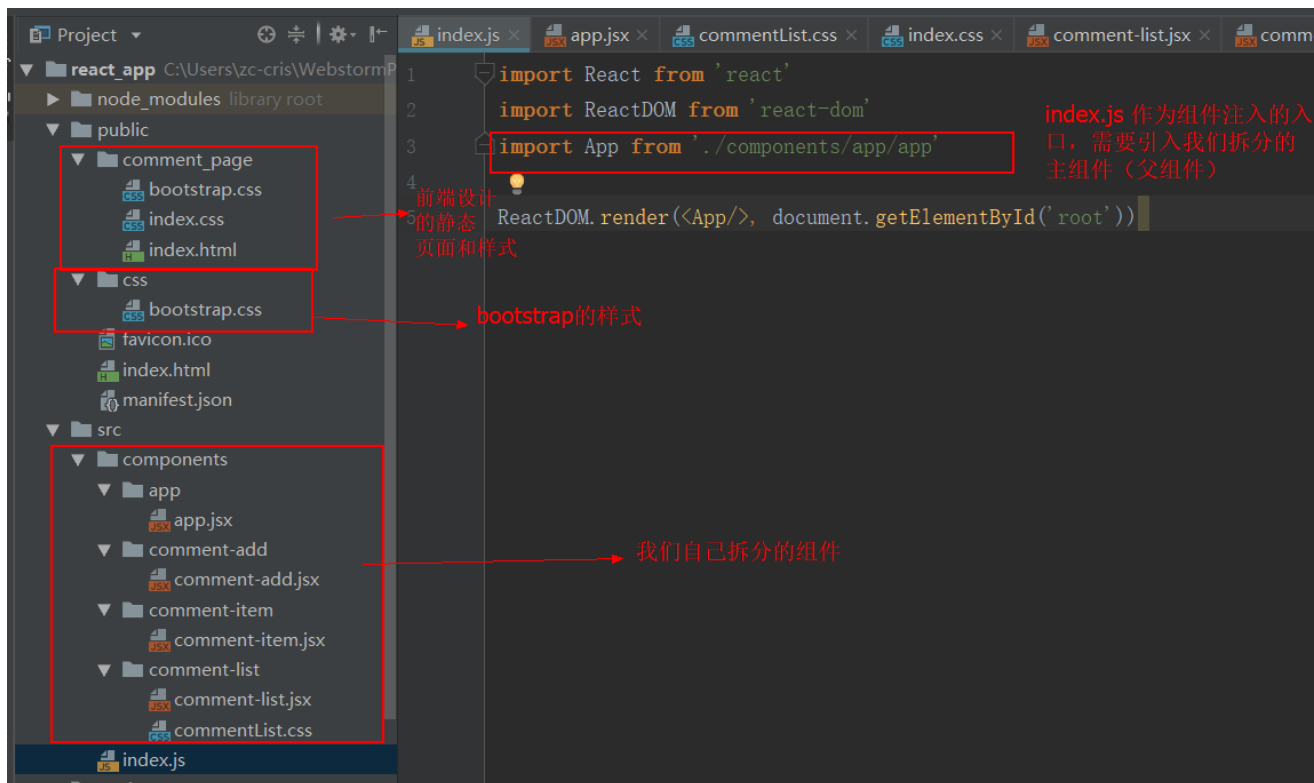


hello react

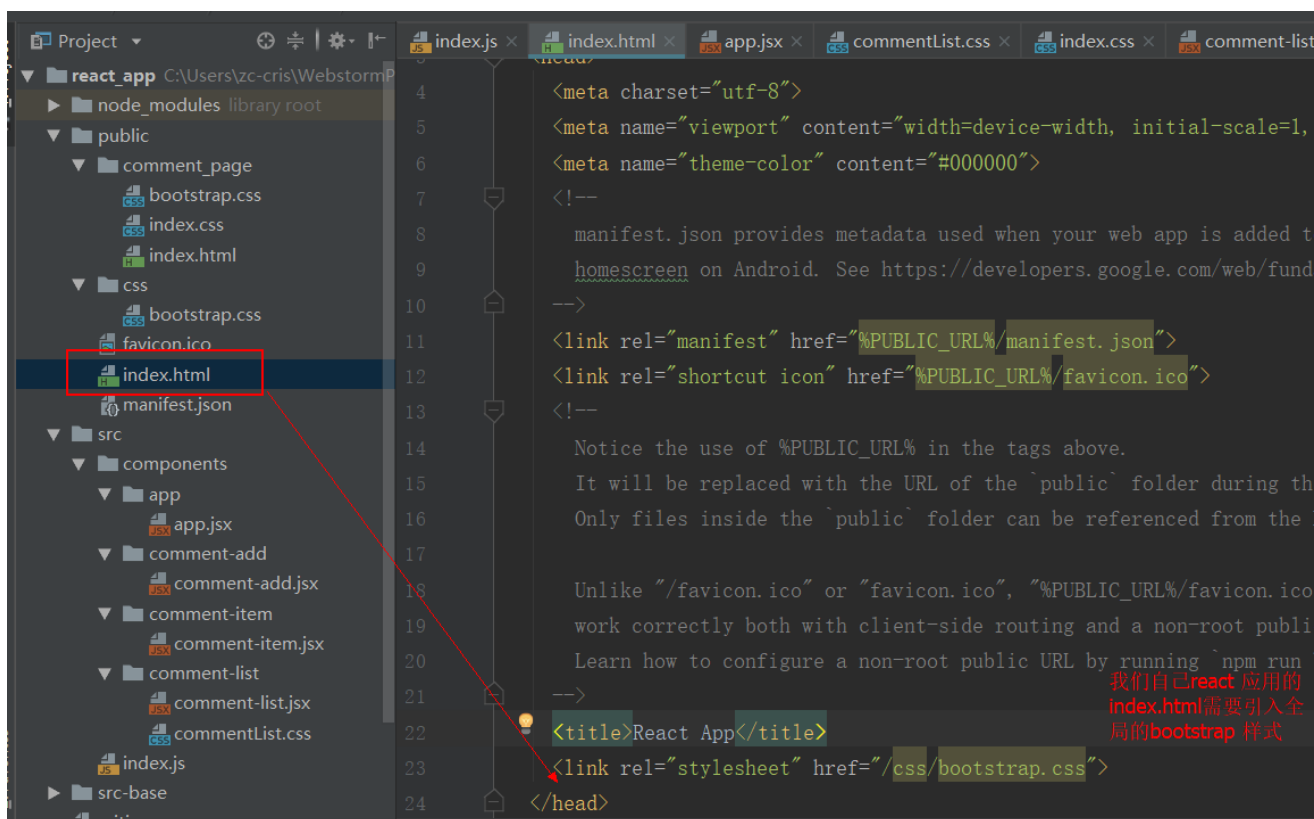
9. 评论demo实战练习

9.1 组件拆分和静态效果展示

1. 结构图



2. 引入全局bootstrap 样式



3. 主组件app.jsx


```

1 import React, {Component} from 'react'
2
3 /*导入我们拆分出来的自定义标签*/
4 import CommentAdd from '../comment-add/comment-add'
5 import CommentList from '../comment-list/comment-list'
6
7 export default class App extends Component {
8
9     render () {
10         return (
11             <div>
12                 <header className="site-header jumbotron">
13                     <div className="container">
14                         <div className="row">
15                             <div className="col-xs-12">
16                                 <h1>请发表对React的评论</h1>
17                             </div>
18                         </div>
19                     </div>
20                 </header>
21                 <div className="container">
22                     <CommentAdd/>
23                     <CommentList/>
24                 </div>
25             </div>
26         )
27     }
28 }

```

4. 拆分的子组件

- 评论添加组件 comment-add.jsx

```

1 import React, {Component} from 'react'
2
3 export default class CommentAdd extends Component {
4
5     render () {
6         return (
7             <div className="col-md-4">
8                 <form className="form-horizontal">
9                     <div className="form-group">
10                         <label>用户名</label>
11                         <input type="text" className="form-control" placeholder="用户名" />
12                     </div>
13                     <div className="form-group">
14                         <label>评论内容</label>
15                         <textarea className="form-control" rows="6" placeholder="评论内容">
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

18         <div className="col-sm-offset-2 col-sm-10">
19             <button type="button" className="btn btn-default pull-right">提交
20         </button>
21     </div>
22 </div>
23 </form>
24 </div>
25 )
26 }

```

- 评论显示组件 comment-list.jsx

```

1  import React, {Component} from 'react'
2  import './commentList.css'      /*导入css样式*/
3
4  export default class CommentList extends Component {
5
6      render () {
7          return (
8              <div className="col-md-8">
9                  <h3 className="reply">评论回复: </h3>
10                 <h2 style={{display: 'none'}}>暂无评论, 点击左侧添加评论!!! </h2>
11                 <ul className="list-group">
12                     <li className="list-group-item">
13                         <div className="handle">
14                             <a href="javascript:;">删除</a>
15                         </div>
16                         <p className="user"><span>xxx</span><span>说:</span></p>
17                         <p className="centence">React不错!</p>
18                     </li>
19                     <li className="list-group-item">
20                         <div className="handle">
21                             <a href="javascript:;">删除</a>
22                         </div>
23                         <p className="user"><span>yyy</span><span>说:</span></p>
24                         <p className="centence">React有点难!</p>
25                     </li>
26                 </ul>
27             </div>
28         )
29     }
30 }

```

- 评论显示组件的css样式表 commentList.css

```

1  .reply {
2      margin-top: 0px;
3  }
4
5  li {

```

```

6     transition: .5s;
7     overflow: hidden;
8 }
9
10 .handle {
11     width: 40px;
12     border: 1px solid #ccc;
13     background: #fff;
14     position: absolute;
15     right: 10px;
16     top: 1px;
17     text-align: center;
18 }
19
20 .handle a {
21     display: block;
22     text-decoration: none;
23 }
24
25 .list-group-item .centence {
26     padding: 0px 50px;
27 }
28
29 .user {
30     font-size: 22px;
31 }

```

5. 效果图

控制台启动: npm start

请发表对React的评论

用户名

评论内容

提交

评论回复:

xxx说:
 React不错!
 删除

yyy说:
 React有点难!
 删除

9.2 评论初始化及其动态显示

- 评论列表需要设计成数组，那么这个数组是放在哪个组件里面呢？是放在comment-add.jsx还是comment-list.jsx 中呢？我们这里选择放在父组件 app.jsx 中，因为这个数组将会在两个子组件中都会使用到，公共的部分我们都放在父组件中，而且需要初始化（放入到state属性中）

1. 父组件改造

```
export default class App extends Component {  
  // 我们之前的写法，用于初始化组件对象的state 属性，实际开发中都是使用下面的写法，更简洁  
  /*  
    constructor(props) {  
      super(props)  
      this.state = {  
        comments : [  
          {username: 'cris', content: 'react 很有意思!'},  
          {username: '桥本有菜', content: 'react 有点难啊 - * - !'},  
        ]  
      }  
    }  
  */  
  // 给组件对象指定state属性  
  state = {  
    comments : [  
      {username: 'cris', content: 'react 很有意思!'},  
      {username: '桥本有菜', content: 'react 有点难啊 - * - !'},  
    ]  
  }  
}  
  
render () {
```

两种方法给父组件的state属性对象中增加数组
实际开发中都是越简单越好，所以选择下面这种写法

```
  const {comments} = this.state  
  
  return (  
    <div>  
      <header className="site-header jumbotron">  
        <div className="container">  
          <div className="row">  
            <div className="col-xs-12">  
              <h1>请发表对React的评论</h1>  
            </div>  
          </div>  
        </div>  
      </header>  
      <div className="container">  
        <CommentAdd/>  
        <CommentList comments = {comments}/>  
      </div>  
    </div>  
  )  
}
```

将父组件的state属性对象中的数组传入到显示组件中

2. 显示列表组件改造

- 引入PropTypes 规则包

```
C:\Users\zc-cris\WebstormProjects\react_app>npm install --save prop-types
npm WARN ajv-keywords@3.2.0 requires a peer of ajv@^6.0.0 but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32",
+ prop-types@15.6.1
updated 1 package in 13.217s
```

```
commentItem.css x commentList.css x package.json x app.jsx x
1 {
2   "name": "react_app",
3   "version": "0.1.0",
4   "private": true,
5   "dependencies": {
6     "prop-types": "^15.6.1",
7     "react": "^16.4.1",
8     "react-dom": "^16.4.1",
9     "react-scripts": "1.1.4"
10  },
11  "scripts": {
12    "start": "react-scripts start",
13    "build": "react-scripts build",
14    "test": "react-scripts test --env=jsdom",
15    "eject": "react-scripts eject"
16  }
```

package.json 必须显示这个依赖说明是导入成功了

- comment-list.jsx

```
1 import React, {Component} from 'react'
2 import PropTypes from 'prop-types' // 如果要对外界传来的数据做出显示, 需要导入这个规则包
3
4 // 引入显示列表每个item的组件
5 import CommentItem from '../comment-item/comment-item'
6 import './commentList.css' /*导入css样式*/
7
8 export default class CommentList extends Component {
```

```

9
10  /*给组件类添加props属性验证规则,现在的做法更加简洁*/
11  static propTypes = {
12      comments: PropTypes.array.isRequired
13  }
14
15  render () {
16
17      const {comments} = this.props
18
19      return (
20          <div className="col-md-8">
21              <h3 className="reply">评论回复: </h3>
22              <h2 style={{display: 'none'}}>暂无评论, 点击左侧添加评论!!! </h2>
23              <ul className="list-group">
24                  {
25                      comments.map((comment, index) => <CommentItem comment={comment} key=
{index}/>)
26                  }
27              </ul>
28          </div>
29      )
30  }
31  }
32  /*对外界传入的参数需要做出验证,这是我们以前的做法*/
33  /*
34  CommentList.propTypes = {
35      comments: PropTypes.array.isRequired
36  }
37  */

```

3. 评论项改造

- commentItem.css

```

1  li {
2      transition: .5s;
3      overflow: hidden;
4  }
5
6  .handle {
7      width: 40px;
8      border: 1px solid #ccc;
9      background: #fff;
10     position: absolute;
11     right: 10px;
12     top: 1px;
13     text-align: center;
14 }
15
16 .handle a {

```

```

17     display: block;
18     text-decoration: none;
19 }
20
21 .list-group-item .centence {
22     padding: 0px 50px;
23 }
24
25 .user {
26     font-size: 22px;
27 }

```

- comment-item.jsx

```

1  import React, {Component} from 'react'
2  import PropTypes from 'prop-types'
3
4  import './commentItem.css'
5
6  export default class CommentItem extends Component {
7
8      /*传入的必须是一个js对象，用于显示每条评论*/
9      static propTypes = {
10         comment: PropTypes.object.isRequired
11     }
12
13     render () {
14
15         const {comment} = this.props
16
17         return (
18             <li className="list-group-item">
19                 <div className="handle">
20                     <a href="javascript:;">删除</a>
21                 </div>
22                 <p className="user"><span>{comment.username}</span><span>说:</span></p>
23                 <p className="centence">{comment.content}</p>
24             </li>
25         )
26     }
27 }

```

4. 动态显示效果图

请发表对React的评论

用户名

评论内容

提交

评论回复:

cris说:
react 很有意思!

桥本有菜说:
react 有点难啊 - - !

这个数据是父组件初始化动态生成的

9.3 评论添加

1. 改造App.jsx

/*父组件的添加评论的方法*/

```
addComment = (comment) => {  
  const {comments} = this.state  
  comments.unshift(comment)  
  // 一定要更新父组件的state属性, 添加comment到comments 数组的行为才可以生效  
  this.setState({comments})  
}
```

```
render () {  
  const {comments} = this.state  
  return (  
    <div>  
      <header className="site-header jumbotron">  
        <div className="container">  
          <div className="row">  
            <div className="col-xs-12">  
              <h1>请发表对React的评论</h1>  
            </div>  
          </div>  
        </div>  
      </header>  
      <div className="container">  
        <CommentAdd addComment={this.addComment}/>  
        <CommentList comments = {comments}/>  
      </div>  
    </div>  
  )  
}
```

父组件的添加评论到评论数组的方法
并且传入到子组件中去
方法由父组件创建
调用由子组件完成

2. 改造comment-add.jsx

```

1 export default class CommentAdd extends Component {
2
3   static propTypes = {
4     addComment: PropTypes.func.isRequired
5   }
6
7   /*这样初始化state属性而不是使用constructor构造函数，更加简洁*/
8   state = {
9     username: '',
10    content: ''
11  }
12
13  /*使用箭头函数的形式生成提交用户评论的函数，因为箭头函数默认this指向的是当前对象*/
14  handleSubmit = () => {
15
16    // 收集用户名和用户评论（推荐使用受控组件的方式：即使用对象的state属性来初始化，封装数据，而且很方便的就可以清除用户输入的数据）
17    // 封装成comment 对象(必须要注意comment对象的属性和state的属性名必须一致)
18    const comment = this.state
19
20    // 提交用户名和评论（实际上是调用父组件的添加方法，因为数组是在父组件中）
21    this.props.addComment(comment)
22
23    //清除输入的数据
24    this.setState({
25      username: '',
26      content: ''
27    })
28  }
29
30
31  handleUsernameChange = (event) => {
32    /*拿到输入框的用户名，然后设置到state属性中去*/
33    const username = event.target.value
34    this.setState({username})
35  }
36
37  handleContentChange = (event) => {
38    const content = event.target.value
39    this.setState({content})
40  }
41
42  render () {
43    /*先获取到state属性中的key对应的value*/
44    const {username, content} = this.state
45
46    return (
47      <div className="col-md-4">
48        <form className="form-horizontal">
49          <div className="form-group">
50            <label>用户名</label>

```

```

51         <input type="text" className="form-control" placeholder="用户名"
value={username}
52             onChange={this.handleUsernameChange}/>
53     </div>
54     <div className="form-group">
55         <label>评论内容</label>
56         <textarea className="form-control" rows="6" placeholder="评论内容"
value={content}
57             onChange={this.handleContentChange}></textarea>
58     </div>
59     <div className="form-group">
60         <div className="col-sm-offset-2 col-sm-10">
61             <button type="button" className="btn btn-default pull-right"
onClick={this.handleSubmit}>提交</button>
62         </div>
63     </div>
64 </form>
65 </div>
66 )
67 }
68 }

```

3. 完成效果



请发表对React的评论

用户数据被添加到评论数组中

用户名

评论内容

评论内容

用户提交评论后清除输入的内容

评论回复:

Rio说:

删除

我觉得React还好啦! ^ - ^

明日花绮罗说:

删除

我很喜欢React啊!

cris说:

删除

react 很有意思!

桥本有菜说:

删除

react 有点难啊 - - - !

9.4 评论删除

1. APP 定义删除评论的方法并传递给 CommentList

```
1  /*父组件删除评论的方法*/
2  deleteComment = (index) => {
3      const {comments} = this.state
4      // slice 方法可以用于数组的新增, 删除和修改
5      // comments.splice(index, 1, {})    指定index 位置替换一个数据
6      // comments.splice(index, 0, {})    指定index 位置添加一个数据
7      comments.splice(index, 1)          //指定index 位置删除一个数据
8      // 一定要更新父组件的state属性, 添加comment到comments 数组的行为才可以生效
9      this.setState({comments})
10 }
11
12 render () {
13
14     const {comments} = this.state
15
16     return (
17         <div>
18             <header className="site-header jumbotron">
19                 <div className="container">
20                     <div className="row">
21                         <div className="col-xs-12">
22                             <h1>请发表对React的评论</h1>
23                         </div>
24                     </div>
25                 </div>
26             </header>
27         </div>
28     )
29 }
```

```

25         </div>
26     </header>
27     <div className="container">
28         <CommentAdd addComment={this.addComment}/>
29         { /*这里选择将父组件的deleteComment方法通过CommentList 传递给CommentItem */}
30         <CommentList comments = {comments} deleteComment = {this.deleteComment}/>
31     </div>
32 </div>
33 )
34 }

```

2. 完善 CommentList (传递deleteComment 函数和做文本显示判断)

```

1  import React, {Component} from 'react'
2  import PropTypes from 'prop-types'      // 如果要对外界传来的数据做出显示，需要导入这个规则包
3
4  // 引入显示列表每个item的组件
5  import CommentItem from '../comment-item/comment-item'
6  import './commentList.css'              /*导入css样式*/
7
8  export default class CommentList extends Component {
9
10     /*给组件类添加props属性验证规则*/
11     static propTypes = {
12         comments: PropTypes.array.isRequired,
13         deleteComment: PropTypes.func.isRequired
14     }
15
16     render () {
17
18         const {comments, deleteComment} = this.props
19         // 定义一个常量，计算评论列表是否需要显示暂无评论的文本 ()
20         /* == 和 != 比较若类型不同，先尝试转换类型，再作值比较，最后返回值比较结果 。而 === 和 !==
21         只有在相同类型下,才会比较其值，建议使用 === */
22         const display = comments.length === 0 ? 'block' : 'none'
23
24         return (
25             <div className="col-md-8">
26                 <h3 className="reply">评论回复: </h3>
27                 { /*这里style的属性使用结构赋值的es6 语法*/}
28                 <h2 style={{display}}>暂无评论，点击左侧添加评论!!! </h2>
29                 <ul className="list-group">
30                     {
31                         /*这里将父组件的deleteComment 方法传递给 CommentItem 子组件，
32                         CommentList 充当了中间桥梁的作用，还需要传递index (数组索引给 CommentItem ) */
33                         comments.map((comment, index) => <CommentItem comment={comment} key=
34                             {index} index={index} deleteComment={deleteComment}/>)
35                     }
36                 </ul>
37             </div>
38         )
39     }
40 }

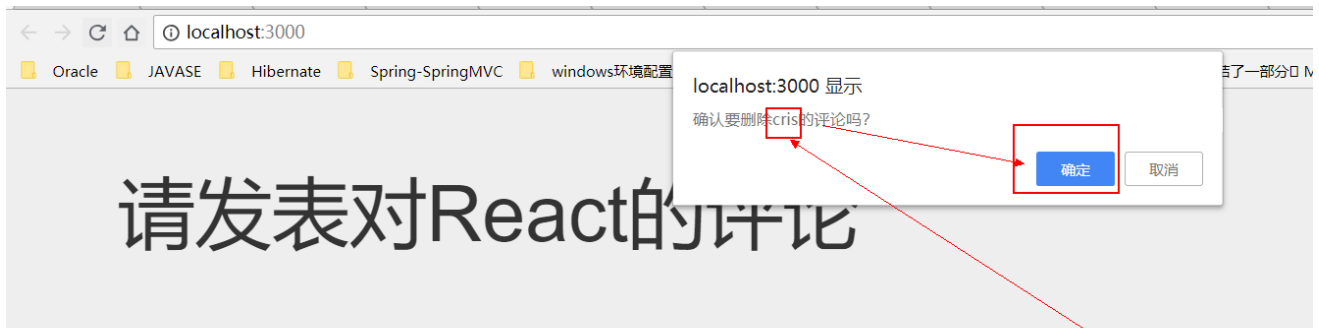
```

```
35     )
36   }
37 }
```

3. CommentItem 的删除事件

```
1  import React, {Component} from 'react'
2  import PropTypes from 'prop-types'
3
4  import './commentItem.css'
5
6  export default class CommentItem extends Component {
7
8    /*传入的必须是一个js对象，用于显示每条评论*/
9    static propTypes = {
10      comment: PropTypes.object.isRequired,
11      index: PropTypes.number.isRequired,
12      deleteComment: PropTypes.func.isRequired
13    }
14
15    /*用户点击删除按钮事件*/
16    handleClick = () => {
17      const {index, deleteComment, comment} = this.props
18      /*这里使用变量输出语法，记得使用 `xxx` 符号而不是 'xxx'*/
19      if (window.confirm(`确认要删除${comment.username}的评论吗? `)) {
20        // 用户确认后删除用户评论数据
21        deleteComment(index)
22      }
23    }
24
25    render () {
26
27      const {comment} = this.props
28
29      return (
30        <li className="list-group-item">
31          <div className="handle">
32            <a href="javascript:;" onClick={this.handleClick}>删除</a>
33          </div>
34          <p className="user"><span>{comment.username}</span><span>说:</span></p>
35          <p className="centence">{comment.content}</p>
36        </li>
37      )
38    }
39  }
```

4. 最终显示效果



请发表对React的评论

用户名

用户名

评论内容

评论内容

提交

评论回复：

暂无评论，点击左侧添加评论！！

用户数据被删除完后显示该文本内容

用户名

詹姆斯

评论内容

我还能打10个!

提交

评论回复：

暂无评论，点击左侧添加评论！！

用户名

用户名

评论内容

评论内容

提交

评论回复：

詹姆斯说：

我还能打10个!

删除

数据添加成功!

10. react通过 axios 完成ajax交互

10.1 实战案例：通过axios 调用github 官方接口得到数据并显示

```
1  <body>
2  <div id="test"></div>
3  <script type="text/javascript" src="../js/react.development.js"></script>
4  <script type="text/javascript" src="../js/react-dom.development.js"></script>
5  <script type="text/javascript" src="../js/prop-types.js"></script>
6  <!--方便测试引入 cdn -->
7  <script src="https://cdn.bootcss.com/axios/0.17.1/axios.js"></script>
8  <script type="text/javascript" src="../js/babel.min.js"></script>
9  <script type="text/babel">
10   /*
11    * 需求：根据指定的关键字在 github 上搜索匹配的最受关注的库，显示库名，以超链接的形式展现，点击即可跳转到这个库的 github 首页去
12    *      测试接口：http://api.github.com/search/repositories?q=rea&sort=stars
13    */
14    class TestAxiosComponent extends React.Component {
15
16      /*将ajax请求返回的数据放入到组件的state属性对象中*/
17      state = {
18        repoName: '',
19        repoUrl: ''
20      }
21
22      /*异步发送ajax 请求一般都放在这个生命周期方法里面*/
23      componentDidMount() {
24        const url = 'http://api.github.com/search/repositories?q=rea&sort=stars'
25        // 使用 axios 协助发送异步的 ajax 请求
26        axios.get(url)
27          // .post(url, {})      // 这是post 请求
28          .then(response => {
29            const result = response.data
30            // console.log(response)
31            // 获取到远程返回数据的指定值(第一个仓库的name 和 html地址)
32            const {html_url, name} = result.items[0]
33            //然后更新到我们组件的state 属性对象中去
34            this.setState({
35              repoName: name,
36              repoUrl: html_url
37            })
38          })
39      }
40
41      render() {
42        const {repoName, repoUrl} = this.state
43        //如果没有通过ajax 请求异步获取到值
44        if (!repoName) {
45          return <h2>Loading...</h2>
46        } else {
47          // 如果获取到值了
```

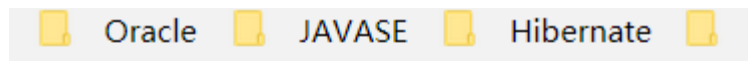


```

48         return <h2>The most start repository is <a href={repoUrl}>{repoName}</a>
49     }
50 }
51 }
52
53 ReactDOM.render(<TestAxiosComponent/>, document.getElementById("test"))
54 </script>
55 </body>

```

10.2 显示效果

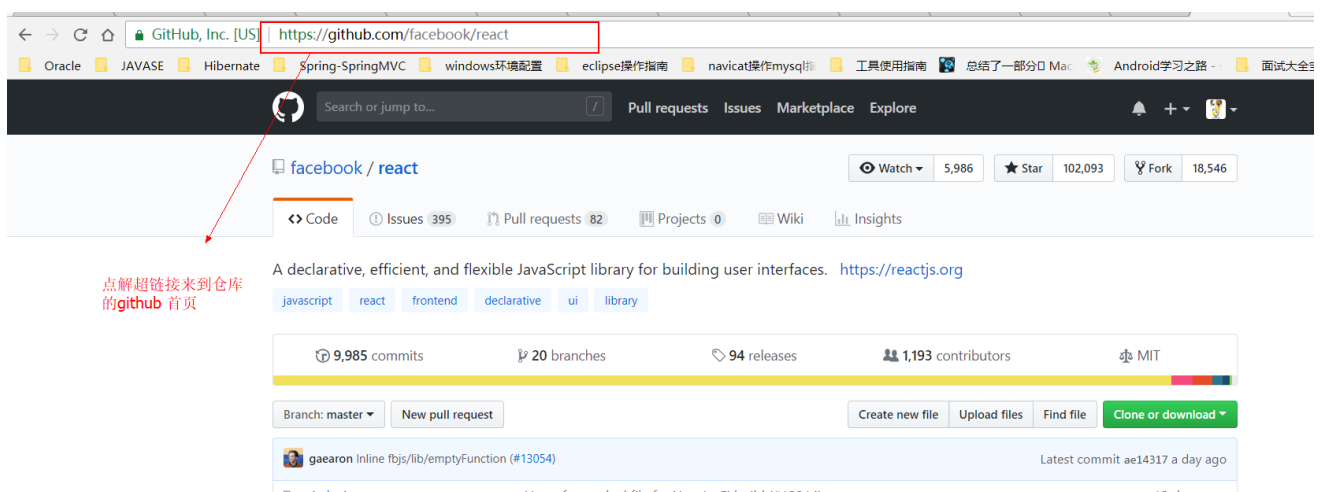


Loading...

访问test.html 首先是初始化组件，因为还没有获取到异步数据，显示loading...

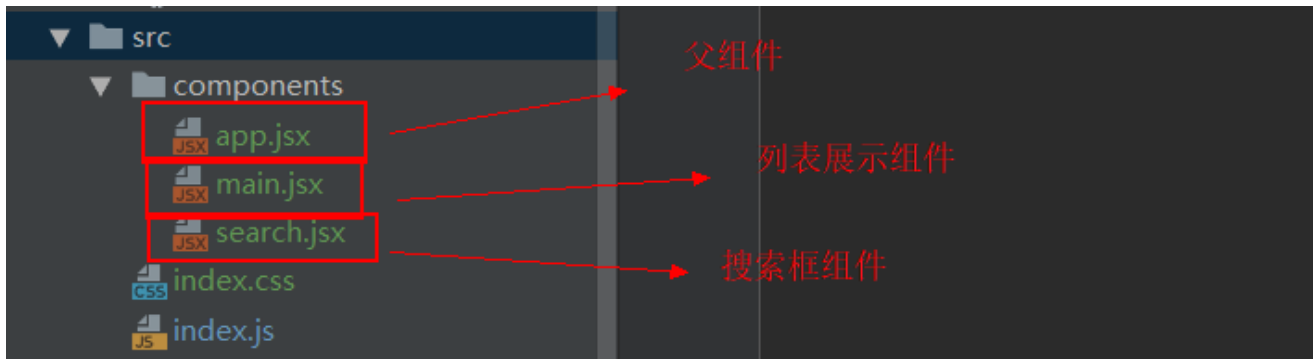
The most start repository is react

异步数据拿到以后自动显示仓库名字（以超链接的形式）



11. 利用ajax 请求完成giuhub 用户搜索案例实战

11.1 搭建环境



- index.js

```
1 import React from 'react'
2 import ReactDOM from 'react-dom'
3
4 import App from './components/app'
5 import './index.css'
6
7 ReactDOM.render(<App/>, document.getElementById('root'))
```

- app.jsx

```
1 import React, {Component} from 'react'
2
3 import Search from './search'
4 import Main from './main'
5
6 export default class App extends Component {
7
8   render () {
9     return (
10       <div className="container">
11         <Search/>
12         <Main/>
13       </div>
14     )
15   }
16 }
```

- main.jsx

```
1 import React, {Component} from 'react'
2
3 export default class Main extends Component {
4
5   render () {
6     return (
7       <div className="row">
8         <div className="card">
9           <a href="https://github.com/reactjs" target="_blank">
```

```

10         
11     </a>
12     <p className="card-text">reactjs</p>
13 </div>
14 </div>
15 )
16 }
17 }

```

- search.jsx

```

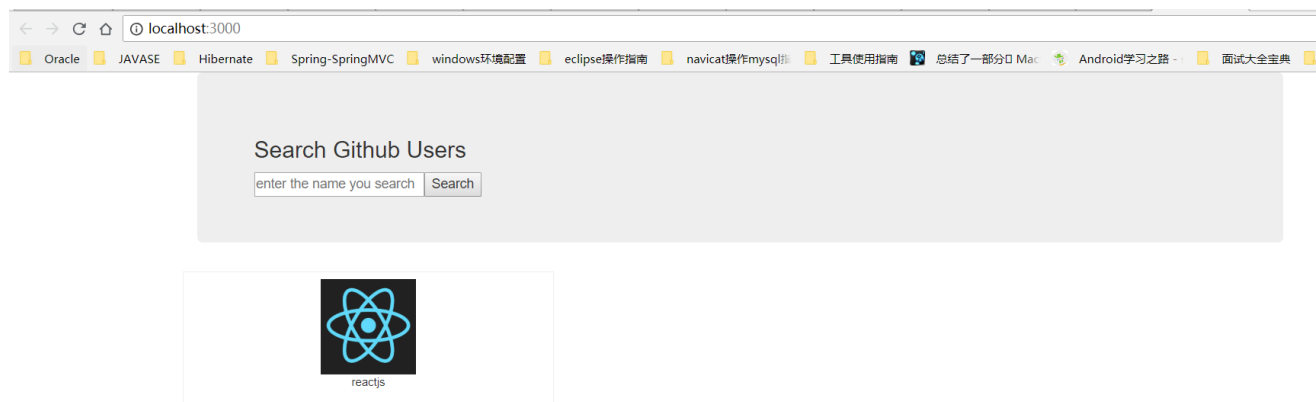
1  import React, {Component} from 'react'
2
3  export default class Search extends Component {
4
5      render () {
6          return (
7              <section className="jumbotron">
8                  <h3 className="jumbotron-heading">Search Github Users</h3>
9                  <div>
10                     <input type="text" placeholder="enter the name you search"/>
11                     <button>Search</button>
12                 </div>
13             </section>
14         )
15     }
16 }

```

- index.css 略

11.2 静态效果图

- npm start 启动react 应用



11.3 main.js (用户数据展示组件的初始化)

```
1 import React, {Component} from 'react'
2
3 export default class Main extends Component {
4
5     state = {
6         initView: true,           // 初始化状态
7         loading: false,          // 发送请求的loading状态
8         users: null,             // 获取到用户数据的状态, 即成功状态
9         errorMsg: null          // 获取用户数据失败的状态
10    }
11
12    /*需要根据状态来进行组件的渲染*/
13    render () {
14
15        const {initView, loading, users, errorMsg} = this.state
16        if (initView){
17            return <h2>请输入用户名进行检索</h2>
18        } else if (loading){
19            return <h2>正在检索中...请稍后</h2>
20        } else if (errorMsg){
21            return <h2>{errorMsg}</h2>
22        } else {
23            return (
24                users.map((user, index) => ( // 箭头函数不仅可以绑定this 到当前对
象, 还可以省略 return 关键字
25                    <div className="row">
26                        <div className="card">
27                            <a href={user.url} target="_blank">
28                                <img src={user.avatarUrl} style={{width: 100}}/>
29                            </a>
30                            <p className="card-text">{user.name}</p>
31                        </div>
32                    </div>
33                ))
34            )
35        }
36    }
37 }
```

- 效果图



11.4 用户交互（极度重点）

1. 用户交互分析



2. App.jsx

```
1 import React, {Component} from 'react'
2
3 import Search from './search'
4 import Main from './main'
5
6 export default class App extends Component {
7
8   state = {
9     searchName: ''
10  }
```

```

11
12  /*父组件的state 属性对象作为中间参数, 获取Search 组件用户输入的数据然后传递到Main 组件中去*/
13  setSearchName = (searchName) => {
14      this.setState({searchName})
15  }
16
17  render () {
18      return (
19          <div className="container">
20              <Search setSearchName={this.setSearchName}/>  /*将父组件的函数设置给子组件用
于更新父组件的state 属性的searchName 值*/
21              <Main searchName = {this.state.searchName}/>
22          </div>
23      )
24  }
25  }

```

3. Search.jsx

```

1  import React, {Component} from 'react'
2  import PropTypes from 'prop-types'
3
4  export default class Search extends Component {
5
6      static propTypes = {
7          setSearchName: PropTypes.func.isRequired
8      }
9
10     state = {
11         input: ''
12     }
13
14     handleClick = () => {
15         // 获取用户输入的搜索数据 (非受控组件的形式)
16         const searchName = this.input.value.trim()
17         // const searchName = this.state.input           // 受控组件的形式
18
19         if (searchName){
20             // 调用父组件的搜索数据设置方法
21             this.props.setSearchName(searchName)
22         } else {
23             alert("请输入搜索的用户名")
24         }
25     }
26
27     // 如果使用受控组件就稍微麻烦一些
28     handleChange = (event) =>{
29         const input = event.target.value.trim()
30         this.setState({input})
31     }
32
33     render () {

```

```

34     return (
35         <section className="jumbotron">
36             <h3 className="jumbotron-heading">Search Github Users</h3>
37             <div>
38                 { /*这里使用非受控组件进行数据的获取*/ }
39                 <input type="text" placeholder="enter the name you search" ref={input =>
this.input = input}/>
40
41                 { /*如果使用非受控组件就比较好清除输入框的内容，但是要麻烦一些（虽然官方推荐）
*/ }
42                 { /*<input type="text" placeholder="enter the name you search" value=
{this.state.input} onChange={this.handleChange}/>*/ }
43
44                 <button onClick={this.handleClick}>Search</button>
45             </div>
46         </section>
47     )
48 }
49 }

```

4. Main.jsx

```

1  import React, {Component} from 'react'
2  import PropTypes from 'prop-types'
3  import axios from 'axios'          // 导入axios 包
4
5  export default class Main extends Component {
6
7      static propTypes = {
8          searchName: PropTypes.string
9      }
10
11      state = {
12          initView: true,           // 初始化状态
13          loading: false,           // 发送请求的loading状态
14          users: null,              // 获取到用户数据的状态，即成功状态
15          errorMsg: null           // 获取用户数据失败的状态
16      }
17
18      /*每次外界传入到组件的props 属性对象的值变化的时候就回调*/
19      componentWillReceiveProps(newProps) {
20          const {searchName} = newProps
21
22          // 更新状态为请求中
23          this.setState({
24              initView: false,       // 必须先设置初始化状态为false
25              loading: true
26          })
27
28          //发送ajax 请求
29          const url = `https://api.github.com/search/users?q=${searchName}` // 使用命名参
数语法需要`xxx${yyy}`, 不要使用'xxx'x或者 "xxx"

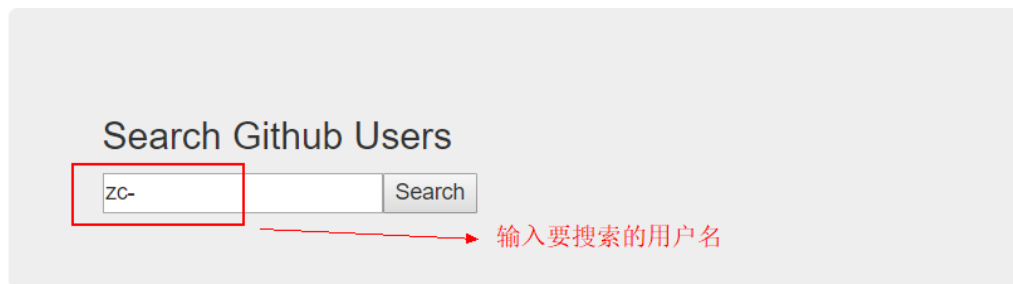
```

组

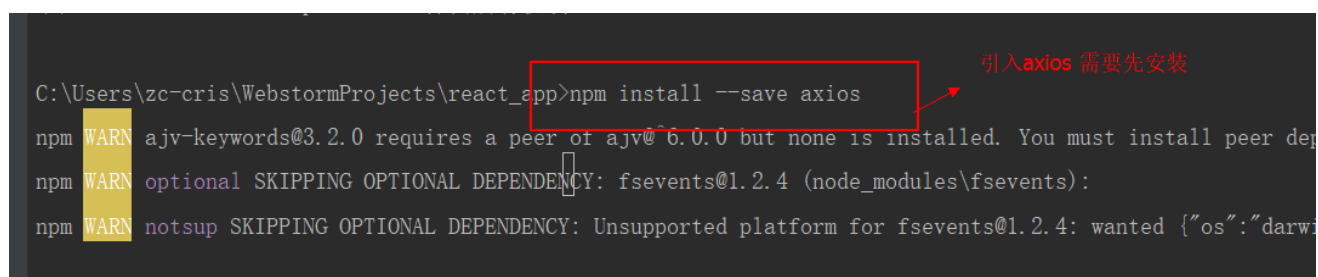
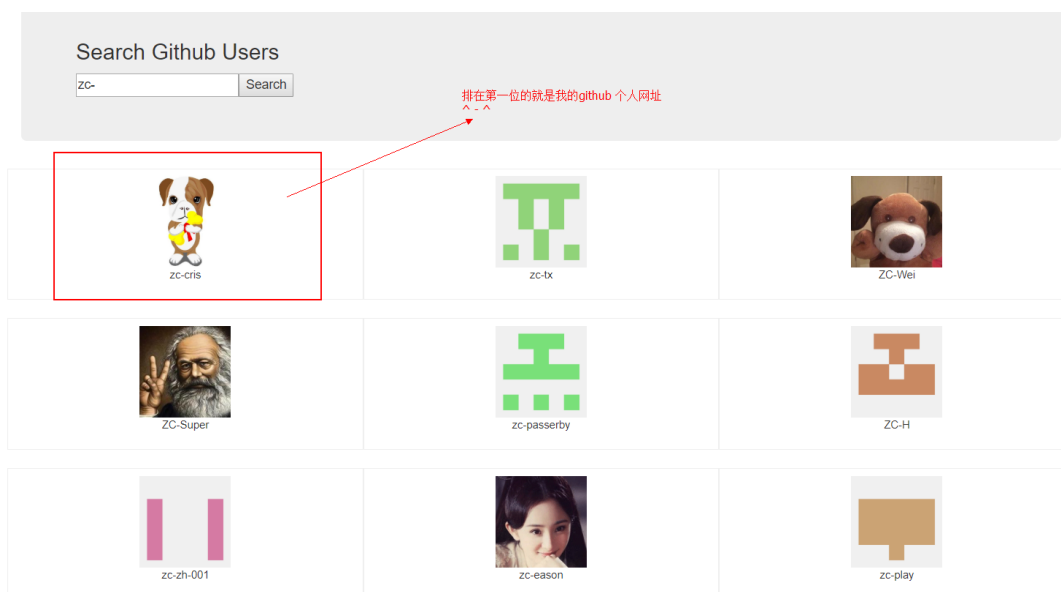
函数体在 () 中

```
30     axios.get(url)
31       .then(response => {
32         // 得到响应数据
33         const result = response.data
34         // 这里非常类似java 8 中的Stream API, 都是将一个数组的数据转换为另一种格式数据的数
35
36         const users = result.items.map(item => ({
37           name: item.login,
38           url: item.html_url,
39           avatarUrl: item.avatar_url
40         }))
41         // 将状态改为成功 (即获取到远程服务器的数据)
42         this.setState({loading: false, users})
43       })
44       .catch(error => {
45         // 将状态改为失败
46         this.setState({loading: false, errorMsg: error.message})
47       })
48
49     /*需要根据状态来进行组件的渲染*/
50     render() {
51       console.log(this.props.searchName)
52       const {initView, loading, users, errorMsg} = this.state
53       if (initView) {
54         return <h2>请输入用户名进行检索</h2>
55       } else if (loading) {
56         return <h2>正在检索中...请稍后</h2>
57       } else if (errorMsg) {
58         return <h2>{errorMsg}</h2>
59       } else {
60         return (
61           <div className="row">
62             {
63               users.map((user, index) => (
64                 // 箭头函数不仅可以绑定this 到当前对象, 还可以省略 return 关键字: 需要
65
66                 <div className="card" key={index}>
67                   <a href={user.url} target="_blank">
68                     <img src={user.avatarUrl} style={{width: 100}}/>
69                     </a>
70                     <p className="card-text">{user.name}</p>
71                   </div>
72                 ))
73             }
74           </div>
75         )
76       }
77     }
78   }
```


11.5 最终效果图



请输入用户名进行检索



12. 组件通信总结（绝对重点）

12.1 通过props 通信的机制（今后开发不建议使用这种方式）

- 1) 共同的数据放在父组件上, 特有的数据放在自己组件内部(state)
- 2) 通过props可以传递一般数据和函数数据, 只能一层一层传递
- 3) 一般数据-->父组件传递数据给子组件-->子组件读取数据
- 4) 函数数据-->子组件传递数据给父组件-->子组件调用函数

12.2 通过发布-订阅通信的机制

- 1) 工具库: PubSubJS
- 2) 下载: `npm install pubsub-js --save`
- 3) 使用:

`import PubSub from 'pubsub-js' //引入`

`PubSub.subscribe('delete', function(data){ }); //订阅`

`PubSub.publish('delete', data) //发布消息`

12.3 使用pub-sub 机制改造上面的github 用户搜索案例

- 下载pubsub-js

```
C:\Users\zc-cris\WebstormProjects\react_app>npm install --save pubsub-js
npm WARN ajv-keywords@3.2.0 requires a peer of ajv@ 6.0.0 but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"darwin","arch":"a
```

- 代码改进 (App.jsx)

```
1  import React, {Component} from 'react'
2
3  import Search from './search'
4  import Main from './main'
5
6  export default class App extends Component {
7
8      render() {
9          return (
10             <div className="container">
11                 <Search/>
12                 <Main/>
13             </div>
14          )
15      }
16  }
```

- 代码改进 (search.jsx)

```
1 import React, {Component} from 'react'
2 import PubSub from 'pubsub-js' // 引入pubsub-js 这个发布订阅包
3
4 export default class Search extends Component {
5
6
7     state = {
8         input: ''
9     }
10
11     handleClick = () => {
12         // 获取用户输入的搜索数据 (非受控组件的形式)
13         const searchName = this.input.value.trim()
14         // const searchName = this.state.input // 受控组件的形式
15
16         if (searchName) {
17             // 需要发布消息 (消息名和消息参数)
18             PubSub.publish('search_user_name', searchName)
19         } else {
20             alert("请输入搜索的用户名")
21         }
22     }
23
24     // 如果使用受控组件就稍微麻烦一些
25     handleChange = (event) => {
26         const input = event.target.value.trim()
27         this.setState({input})
28     }
29
30     render() {
31         return (
32             <section className="jumbotron">
33                 <h3 className="jumbotron-heading">Search Github Users</h3>
34                 <div>
35                     /*这里使用非受控组件进行数据的获取*/
36                     <input type="text" placeholder="enter the name you search" ref={input =>
37 this.input = input}/>
38
39                     /*如果使用非受控组件就比较好清除输入框的内容，但是要麻烦一些（虽然官方推荐）
40                     */
41
42                     /*<input type="text" placeholder="enter the name you search" value=
43 {this.state.input} onChange={this.handleChange}/>*/
44
45                     <button onClick={this.handleClick}>Search</button>
46                 </div>
47             </section>
48         )
49     }
50 }
```

- 代码改进 (main.jsx)

```
1 import React, {Component} from 'react'
2 import axios from 'axios' // 导入axios 包
3 import PubSub from 'pubsub-js'
4
5 export default class Main extends Component {
6
7
8   state = {
9     initView: true,           // 初始化状态
10    loading: false,           // 发送请求的loading状态
11    users: null,              // 获取到用户数据的状态, 即成功状态
12    errorMsg: null            // 获取用户数据失败的状态
13  }
14
15  /*注意: 订阅事件需要在组件初始化完成后即绑定*/
16  componentDidMount() {
17
18    /*强烈建议回调函数都用箭头的形式*/
19    PubSub.subscribe('search_user_name', (msg, searchName) => {
20      // 更新状态为请求中
21      this.setState({
22        initView: false,       // 必须先设置初始化状态为false
23        loading: true
24      })
25
26      //发送ajax 请求
27      const url = `https://api.github.com/search/users?q=${searchName}` // 使用命名参数语法需要`xxx${yyy}`, 不要使用'xxx'x或者 "xxx"
28      axios.get(url)
29        .then(response => {
30          // 得到响应数据
31          const result = response.data
32          // 这里非常类似java 8 中的Stream API, 都是将一个数组的数据转换为另一种格式数据的数组
33
34          const users = result.items.map(item => ({
35            name: item.login,
36            url: item.html_url,
37            avatarUrl: item.avatar_url
38          }))
39          // 将状态改为成功 (即获取到远程服务器的数据)
40          this.setState({loading: false, users})
41        })
42        .catch(error => {
43          // 将状态改为失败
44          this.setState({loading: false, errorMsg: error.message})
45        })
46      })
47
48    /*需要根据状态来进行组件的渲染*/
49    render() {
```

```

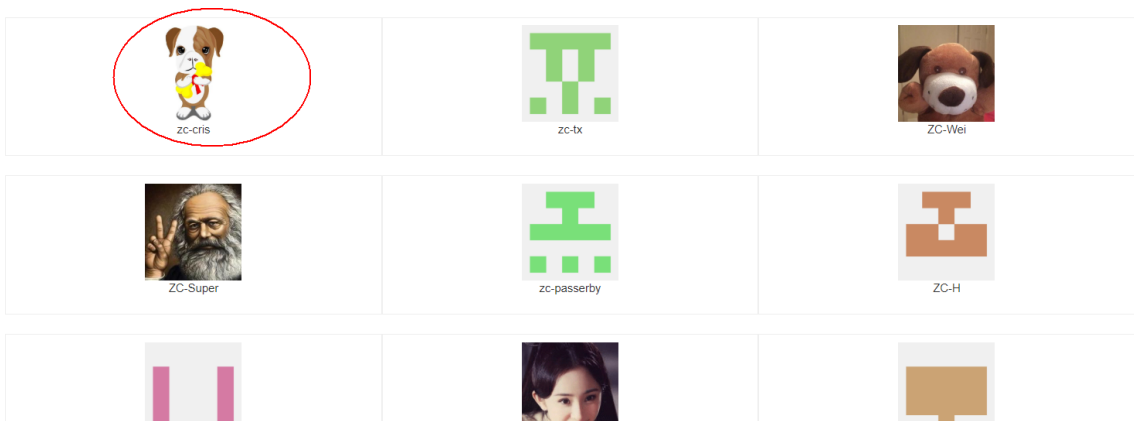
50 console.log(this.props.searchName)
51 const {initView, loading, users, errorMsg} = this.state
52 if (initView) {
53     return <h2>请输入用户名进行检索</h2>
54 } else if (loading) {
55     return <h2>正在检索中...请稍后</h2>
56 } else if (errorMsg) {
57     return <h2>{errorMsg}</h2>
58 } else {
59     return (
60         <div className="row">
61             {
62                 users.map((user, index) => ( // 箭头函数不仅可以绑定this 到当前对象, 还
可以省略 return 关键字: 需要函数体在 () 中
63
64                 <div className="card" key={index}>
65                     <a href={user.url} target="_blank">
66                         <img src={user.avatarUrl} style={{width: 100}}/>
67                     </a>
68                     <p className="card-text">{user.name}</p>
69                 </div>
70
71             ))
72         }
73     </div>
74 )
75 }
76 }
77 }

```

- 改进效果

Search Github Users

发布订阅模式大大简化了属性传递的步骤
比通过props 这种方式方便了许多



12.4 什么时候使用pub-sub 模式?

当我们的组件和组件之间传递的参数层级大于或者等于3的时候，就需要考虑使用pub-sub模式

以用户评论管理案例为例：

1. 评论添加不需要使用pub-sub模式，因为comment-add组件添加完成后之间调用父组件的添加方法（这里层级是2）
2. 但是评论删除功能，需要父组件的删除方法-->comment-list组件-->comment-item组件（这里层级是3），所以建议使用发布-订阅模式，comment-item组件不再调用传递来的父组件的删除方法，而是发布删除消息，父组件接收到后直接调用自己的删除函数即可

12.5 使用pub-sub 模式改进用户评论管理案例

1. comment-item.jsx

```
1      /*用户点击删除按钮事件*/
2      handleClick = () => {
3          const {index, comment} = this.props
4          /*这里使用变量输出语法，记得使用 `xxx` 符号而不是 'xxx'*/
5          if (window.confirm(`确认要删除${comment.username}的评论吗?`)) {
6              /*用户确定删除那么就需要发布删除事件给App父组件*/
7              // 这里的两个参数：发布事件名，父组件删除item函数需要的index索引
8              PubSub.publish('delete_comment', index)
9          }
10     }
```

2. App.jsx

```
1      /*订阅事件一般都放在组件初始化完成后*/
2      componentDidMount(){
3          PubSub.subscribe('delete_comment', (msg, index) => {
4              this.deleteComment(index)
5          })
6      }
```

3. 效果图

请发表对React的评论

用户名

评论内容

提交

评论回复：

cris说:
react 很有意思!

删除

桥本有菜说:
react 有点难啊 - - !

删除

请发表对React的评论

用户名

评论内容

评论回复:

cris说:

react 很有意思!

删除

提交

删除成功!

13. React-router

13.1 react-router的理解

- 1) react的一个插件库
- 2) 专门用来实现一个SPA应用
- 3) 基于react的项目基本都会用到此库

13.2 SPA的理解

- 1) 单页Web应用 (single page web application, SPA)
- 2) 整个应用只有一个完整的页面
- 3) 点击页面中的链接不会刷新页面, 本身也不会向服务器发请求
- 4) 当点击路由链接时, 只会做页面的局部更新
- 5) 数据都需要通过ajax请求获取, 并在前端异步展现

13.3 路由的理解

- 1) 什么是路由? a. 一个路由就是一个映射关系(key:value) b. key为路由路径, value可能是function/component
- 2) 路由分类 a. 后台路由: node服务器端路由, value是function, 用来处理客户端提交的请求并返回一个响应数据 b. 前台路由: 浏览器端路由, value是component, 当请求的是路由path时, 浏览器端前没有发送http请求, 但界面会更新显示对应的组件
- 3) 后台路由 a. 注册路由: router.get(path, function(req, res)) b. 当node接收到一个请求时, 根据请求路径找到匹配的路由, 调用路由中的函数来处理请求, 返回响应数据
- 4) 前端路由 a. 注册路由: b. 当浏览器的hash变为#about时, 当前路由组件就会变为About组件

13.4 实战案例：根据用户点击显示不同的路由组件

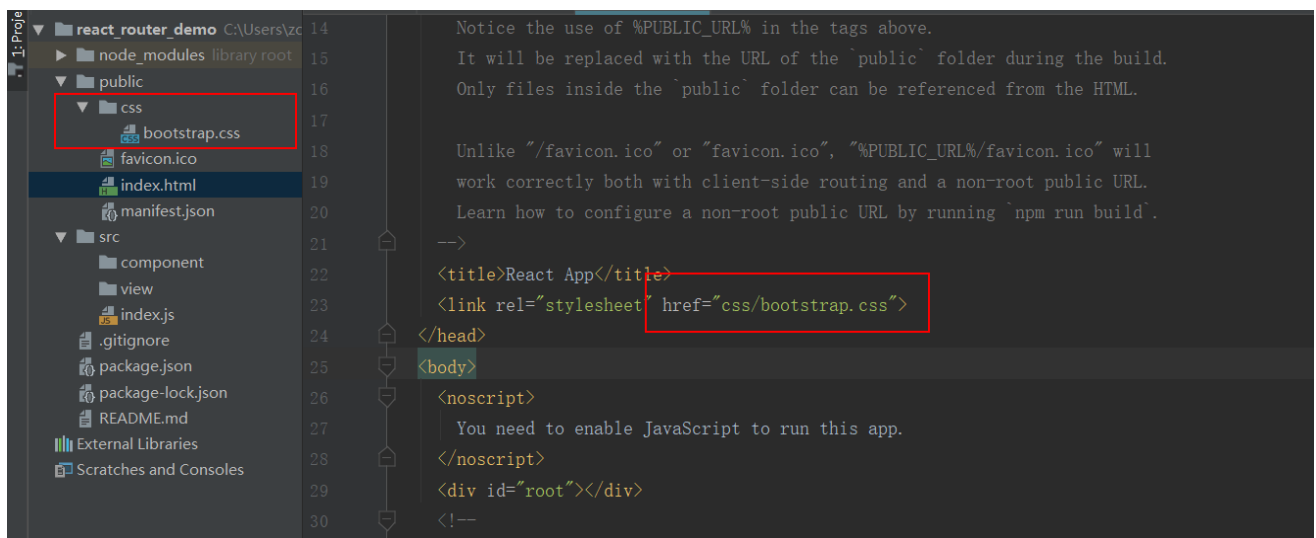
1. 快速搭建一个react 应用

```
\\zc-cris\\WebstormProjects>create-react-app react_router_demo
```

2. 安装react-router 插件

```
C:\Users\zc-cris\WebstormProjects\react_router_demo>npm install --save react-router-dom
```

3. 引入bootstrap.css



```
<link rel="stylesheet" href="css/bootstrap.css">
```

4. 完成各个组件

- index.jsx

```
1 import React from 'react'
2 import {render} from 'react-dom'
3 import {BrowserRouter} from 'react-router-dom'
4
5 import App from './component/App'
6
7 render(
8   (
```



```

9      /*使用BrowserRouter 来管理我们的应用*/
10     <BrowserRouter>
11       <App/>
12     </BrowserRouter>
13   ),
14   document.getElementById("root")
15 )

```

- home.jsx

```

1  import React, {Component} from 'react'
2
3  export default class Home extends Component {
4
5      render () {
6          return (
7              <div>
8                  home route component
9              </div>
10          )
11      }
12  }

```

- about.jsx

```

1  import React, {Component} from 'react'
2
3  export default class About extends Component {
4
5      render () {
6          return (
7              <div>
8                  about route component
9              </div>
10          )
11      }
12  }

```

- app.jsx

```

1  import React, {Component} from 'react'
2  import {NavLink, Switch, Route, Redirect} from 'react-router-dom'
3
4  import About from '../view/about'
5  import Home from '../view/home'
6
7  export default class App extends Component {
8      render() {
9          return (
10              <div>

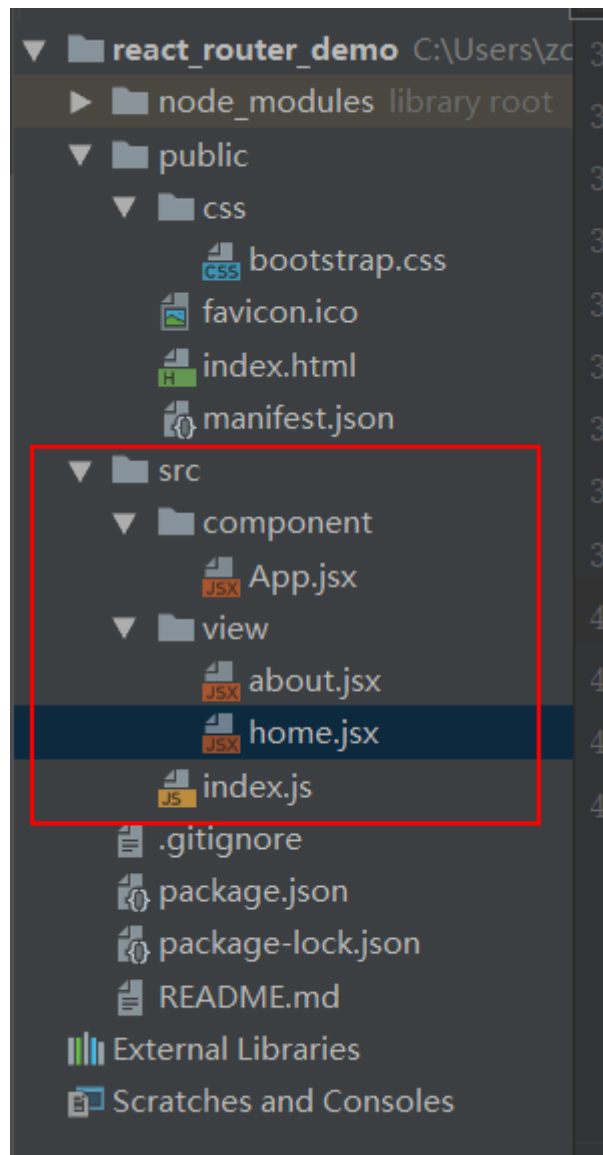
```

```

11     <div className="row">
12         <div className="col-xs-offset-2 col-xs-8">
13             <div className="page-header"><h2>React Router Demo</h2></div>
14         </div>
15     </div>
16
17     <div className="row">
18         <div className="col-xs-2 col-xs-offset-2">
19             <div className="list-group">
20                 { /*路由链接, 用于切换路由组件的链接*/ }
21                 <NavLink className='list-group-item' to='/about'>About</NavLink>
22                 <NavLink className='list-group-item' to='/home'>Home</NavLink>
23             </div>
24         </div>
25
26         <div className='col-xs-6'>
27             <div className='panel'>
28                 <div className='panel-body'>
29                     { /*Switch 标签用于切换路由组件*/ }
30                     <Switch>
31                         <Route path='/about' component={About}/>
32                         <Route path='/home' component={Home}/>
33                         { /*首次进入页面默认显示 about 路由组件*/ }
34                         <Redirect to='/about' />
35                     </Switch>
36                 </div>
37             </div>
38         </div>
39     </div>
40 </div>
41 )
42 }
43 }

```

- 结构图



5. 简单测试效果



React Router Demo

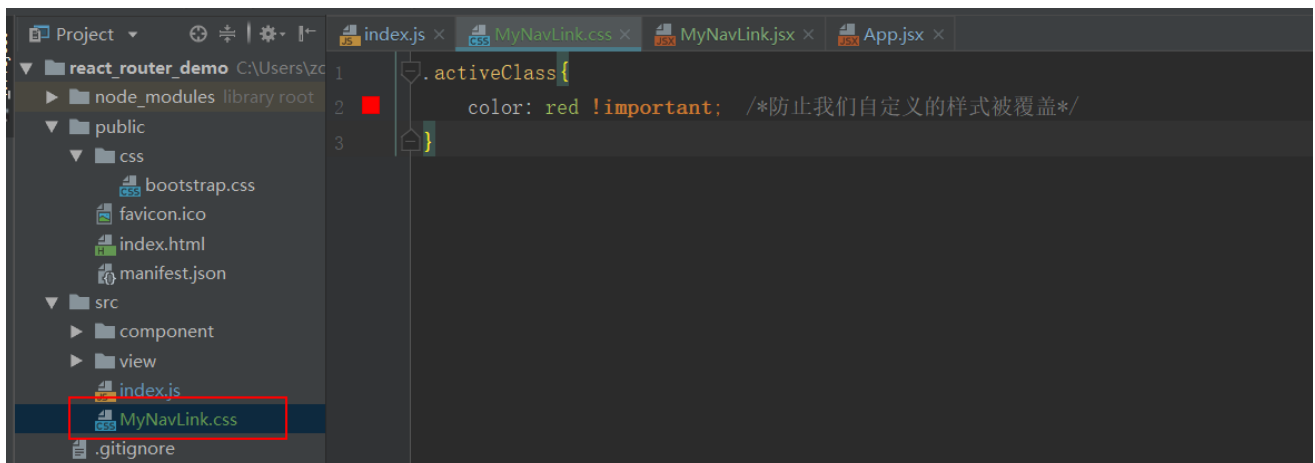


home route component

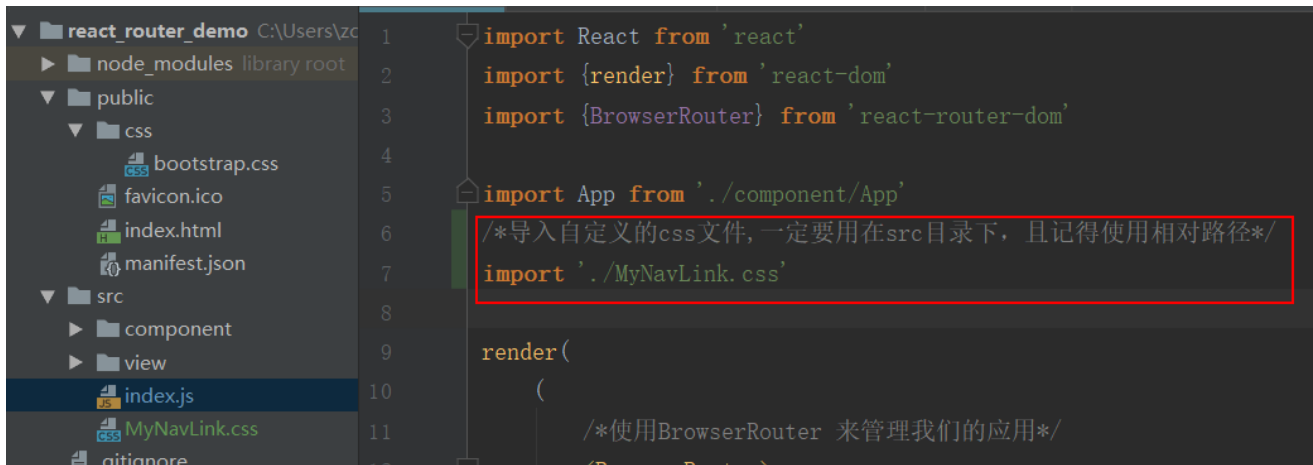
- 常用的几个路由标签
 - : 路由键链接
 - : 路由组件切换容器
 - : 路由组件
 - : 指定默认显示的路由组件

6. 包装NavLink 标签，自定义样式

- 创建自定义样式文件



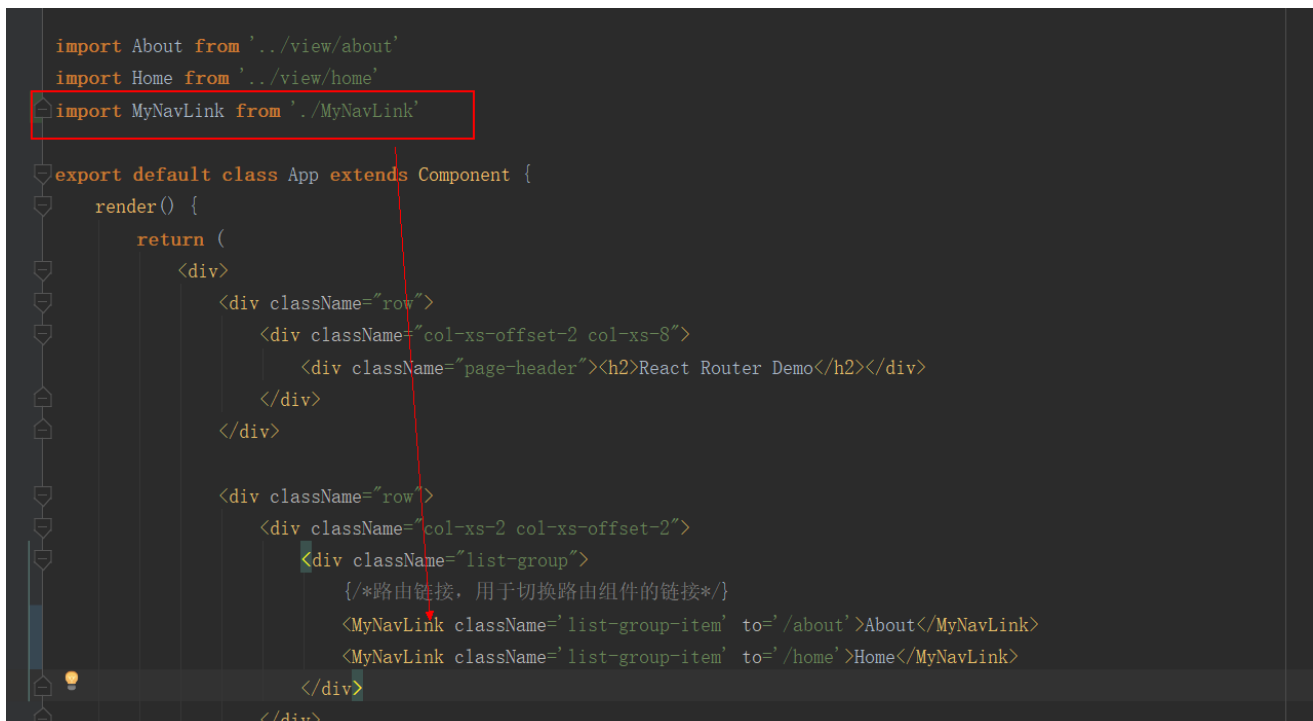
- index.js 记得导入自定义样式表文件



- 自定义MyNavLink 标签，代理React-Router 的NavLink 标签



- App.jsx 导入自定义的组件



- 效果图



7. 嵌套路由

- 嵌套路由组件代码(news.jsx 和 message.jsx)

```
1  import React, {Component} from 'react'
2
3  export default class News extends Component {
4
5      state = {
6          news: [
7              'news01',
8              'news02',
9              'news03'
10         ]
11     }
12
13     render () {
14         return (
15             <div>
16                 <ul>
17                     {
18                         this.state.news.map((n, index) => <li key={index}>{n}</li>)
19                     }
20                 </ul>
21             </div>
22         )
23     }
24 }
25
26
27
28 import React, {Component} from 'react'
29
30 export default class Message extends Component {
```

```

31
32     state = {
33         messages: [
34
35         ]
36     }
37
38     componentDidMount(){
39         setTimeout(() => {
40             const messages = [
41                 {id: '1', title: 'message01'},
42                 {id: '2', title: 'message02'},
43                 {id: '3', title: 'message03'}
44             ]
45             this.setState({messages})
46         }, 1000)
47     }
48
49     render () {
50         return (
51             <div>
52                 <ul>
53                     {
54                         this.state.messages.map((m, index) => <a href='xxx'><li key={index}>
{m.title}</li></a>)
55                     }
56                 </ul>
57             </div>
58         )
59     }
60 }

```

- 重写父路由组件 (home.jsx)

```

1  import React, {Component} from 'react'
2  import {Route, Switch, Redirect} from 'react-router-dom'
3
4  import MyNavLink from '../component/MyNavLink'
5  import News from './second/news'
6  import Message from './second/message'
7
8  export default class Home extends Component {
9
10     render () {
11         return (
12             <div>
13                 <h2>home route component</h2>
14                 <ul className='nav nav-tabs'>
15                     <li>
16                         <MyNavLink to='/home/news'>News</MyNavLink>
17                     </li>
18                     <li>

```

```

19         <MyNavLink to='/home/messages'>Messages</MyNavLink>
20     </li>
21 </ul>
22 <div>
23     <Switch>
24         <Route path='/home/news' component={News}/>
25         <Route path='/home/messages' component={Message}/>
26         <Redirect to='/home/news' />
27     </Switch>
28 </div>
29 </div>
30 )
31 }
32 }

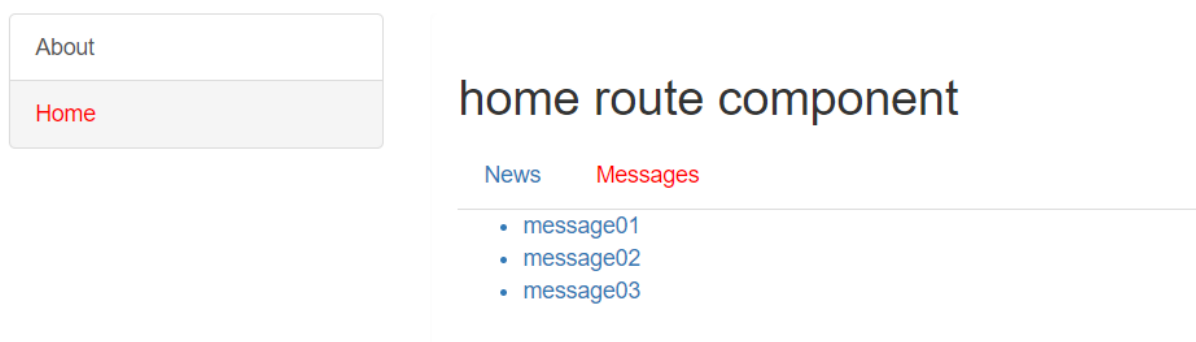
```

- 效果图

React Router Demo

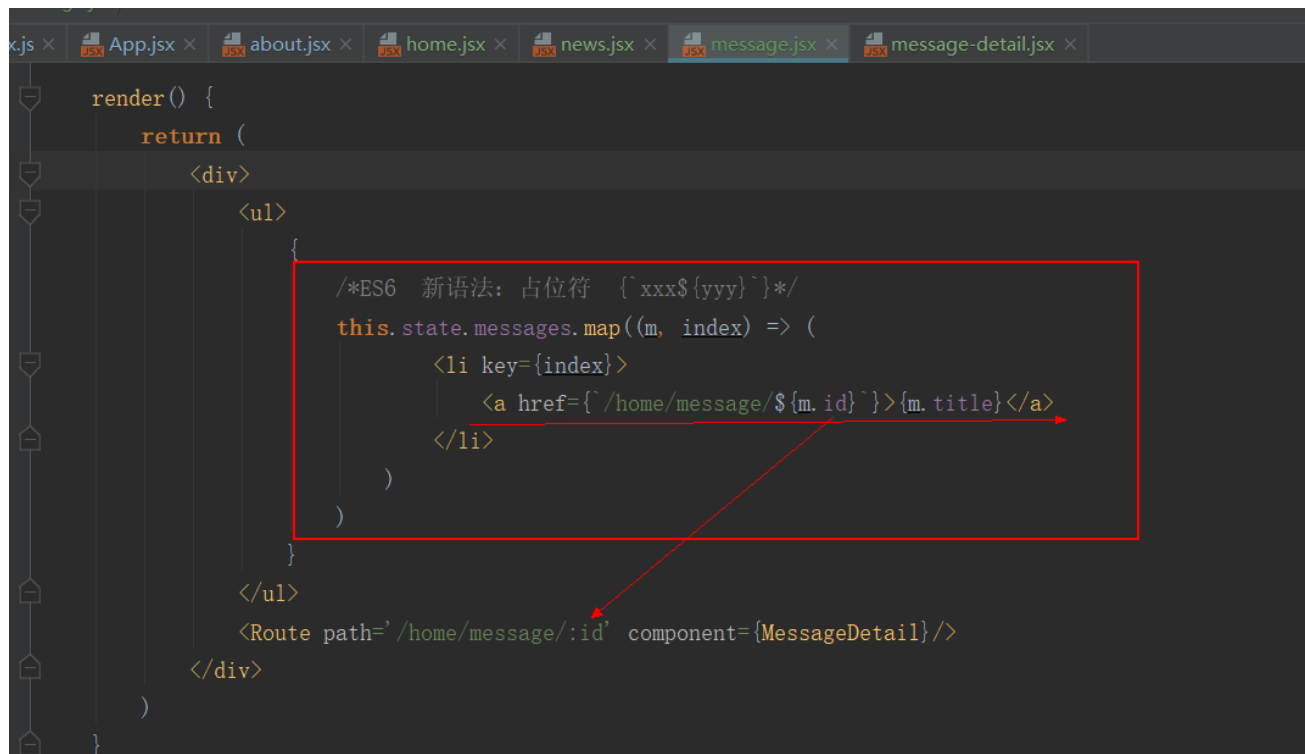


React Router Demo



8. 向路由组件传递数据（通过路由路径后的参数）

- message.jsx 改进



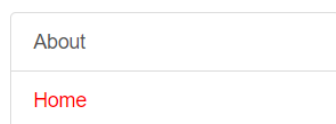
```
render() {
  return (
    <div>
      <ul>
        {
          /*ES6 新语法: 占位符  `{xxx}${yyy}`*/
          this.state.messages.map((m, index) => (
            <li key={index}>
              <a href={` /home/message/${m.id}`} >{m.title}</a>
            </li>
          ))
        }
      </ul>
      <Route path='/home/message/:id' component={MessageDetail}/>
    </div>
  )
}
```

- 新建三级路由组件message-detail.jsx

```
1  import React from 'react'
2
3  const details = [
4    {id: '1', title: 'message01', content: 'hello cris'},
5    {id: '2', title: 'message02', content: 'hello 大帅'},
6    {id: '3', title: 'message03', content: 'hello 克里斯'}
7  ]
8  export default function MessageDetail(props) {
9
10     // 得到请求路径对应的参数id
11     const {id} = props.match.params
12     // debugger
13     // 若要使用单行箭头函数直接返回一个对象字面量, 请使用一个括号包裹改对象字面量, 而不是直接使用大括号, 否则ES6解析引擎会将其解析为一个多行箭头函数
14     const message = details.find((m) => (m.id===id) )      /*返回数组中第一个匹配的元素*/
15
16     return (
17       <ul>
18         <li>{message.id}</li>
19         <li>{message.title}</li>
20         <li>{message.content}</li>
21       </ul>
22     )
23 }
```

- 效果图

React Router Demo



home route component

News Messages

- message01
- message02
- message03
- 3
- message03
- hello 克里斯

点击路由链接，即可显示对应的路由组件内容

和之前的通过路由链接直接切换路由组件不同；
这里是通过点击 超链接切换到路由组件（并将id数据传递给路由组件去后台查询数据后才渲染）

9. 路由链接和非路由链接

以上面的message.jsx 为例

```
1 render() {
2   return (
3     <div>
4       <ul>
5         {
6           /*ES6 新语法: 占位符  `${xxx}${yyy}` */
7           this.state.messages.map((m, index) => (
8             <li key={index}>
9               <a href={` /home/message/${m.id}`}>{m.title}</a>
10             </li>
11           )
12         )
13       }
14     </ul>
15     <Route path='/home/message/:id' component={MessageDetail}/>
16   </div>
17 )
18 }
```

其中 `{m.title}` 就是非路由链接

React Router Demo

AboutHome

home route component

NewsMessages

- message01
- message02
- message03
- 3
- message03
- hello 克里斯

点击非路由链接将会发送请求

Name	Stat...	Type	Initiator	Size	Time	Waterfall
websocket	101	we...	websock...	0 B	Pen...	
3	304	doc...	Other	178 B	7 ms	
bootstrap.css	304	styl...	2	244 B	9 ms	
bundle.js	304	script	2	181 B	6 ms	
info?t=15298081...	200	xhr	abstract...	368 B	1 ms	

5 requests | 971 B transferred | Finish: 193 ms | DOMContentLoaded: 142 ms | Load: 189 m

Console

改进非路由链接为路由链接（将a标签改为NavLink 标签即可）

```
1 { /*<a href={` /home/message/${m.id}`}>{m.title}</a>*/}  
2 <NavLink to={` /home/message/${m.id}`}>{m.title}</NavLink>
```

效果图

React Router Demo

AboutHome

home route component

NewsMessages

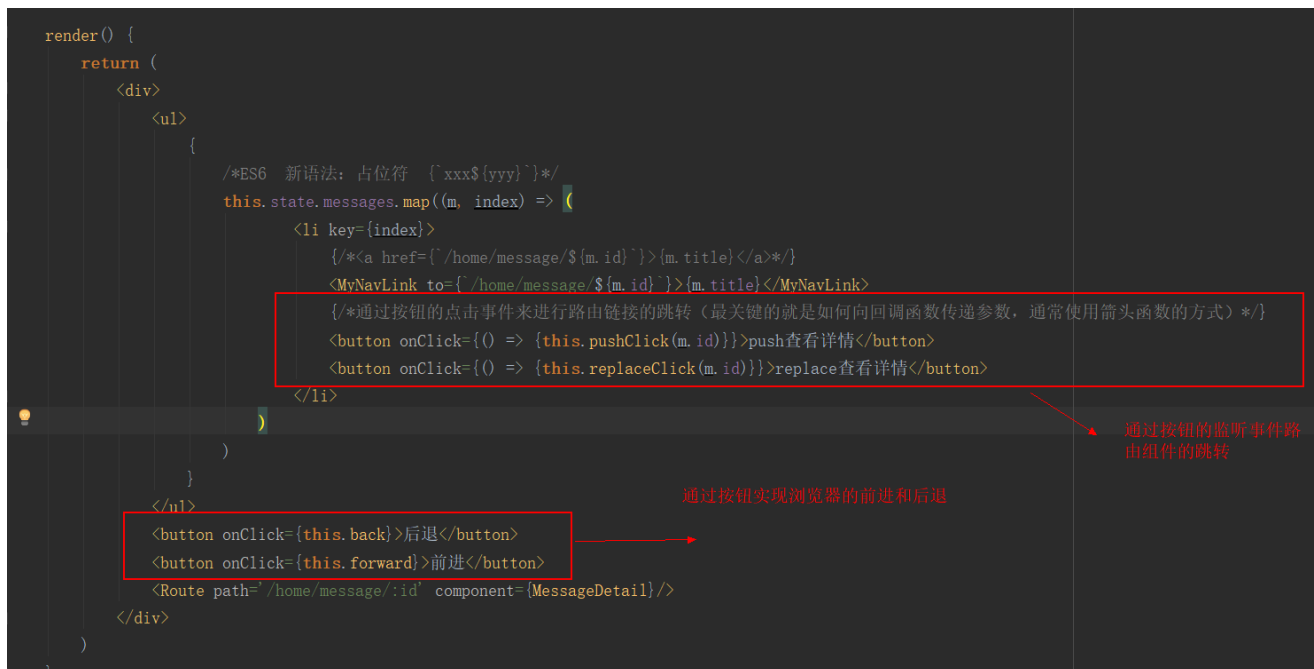
- message01
- message02
- message03
- 3
- message03
- hello 克里斯

不会再发请求了，这就是路由链接

Recording network activity...
Perform a request or hit F5 to record the reload.

10. 路由链接的两种跳转方式（超链接和按钮事件）

除了上面的链接形式，我们还可以通过按钮的点击事件进行路由组件的跳转，message.jsx 改进如下



对应的方法实现

```
1  handleClick = (id) => {
2    this.props.history.push(`/home/message/${id}`)
3  }
4  handleClick = (id) => {
5    this.props.history.replace(`/home/message/${id}`)
6  }
7  back = () => {
8    this.props.history.goBack()
9  }
10 forward = () => {
11   this.props.history.goForward()
12 }
```

效果图

React Router Demo

The demo interface shows a sidebar with 'About' and 'Home' (selected). The main content area is titled 'home route component' and contains a 'News' section and a 'Messages' section. The 'Messages' section has a table with three rows, each containing 'push查看详情' and 'replace查看详情' buttons. Below the table are '后退' (Back) and '前进' (Forward) buttons. Red arrows and text explain the functions: 'push' adds a new link to the history stack, 'replace' replaces the current link, and the navigation buttons implement browser forward and back functionality.

About

Home

home route component

News Messages

• message01	push查看详情	replace查看详情
• message02	push查看详情	replace查看详情
• message03	push查看详情	replace查看详情

后退 前进

• 1

- message01
- hello cris

点击push, 相当于向history 这个栈结构添加了一个新的链接

点击replace, 相当于向history的当前链接替换为按钮指向的链接

通过按钮实现浏览器的前进和后退功能

14. React-ui (antd-mobile) 略

15. React-Redux (极度重点)

