

课堂目标

资源

起步

上手

文件结构

React&ReactDOM

JSX

元素渲染

**更新已渲染的元素**

循环绑定元素

map绑定

过滤元素

ref和refs

字符串写法

函数写法

组件&props

函数组件

类声明组件

两种方式的区别

组合组件

提取组件

父子组件通信

Props的只读性

state

组件状态

修改状态

关于setState

生命周期

受控组件

受控组件实现

非受控组件的实现

# 课堂目标

---

- 安装 `create-react-app` 脚手架
- 熟练React基础语法
- 掌握JSX语法
- 掌握setState
- 掌握React生命周期
- 掌握props传递参数
- 掌握React组件通信

## 资源

---

- [react](#)
- [create-react-app](#)

## 起步

---

### 上手

- `npm i -g create-react-app` 安装官方脚手架
- `react-react-app 01_react` 初始化
- react的api比较少,基本学习一次,就再也不用再看文档,它的核心是JS

## 文件结构

1

## React&ReactDOM

React只做逻辑层,ReactDOM去渲染真实的DOM

删除src下面所有代码,新建index.js

```
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import App from './App'
4 ReactDOM.render(<App />, document.querySelector('#root'))
```

新建 App.js

```
1 import React from 'react'
2 class App extends React.Component {
3   render() {
4     return (
5       <div>
6         hello,小马哥
7       </div>
8     )
9   }
10 }
11 export default App;
```

以上代码感觉都没有问题,但是发现了一个很有趣的标签语法,它既不是字符串也不是HTML

它被称为JSX,是一个JavaScript的语法扩展.我们建议在 React 中配合使用 JSX, JSX 可以很好地描述 UI 应该呈现出它应有交互的本质形式。JSX 可能会使人联想到模版语言,但它具有 JavaScript 的全部功能。

## JSX

在JavaScript中直接写的标签,是一个JSX(JS+XML,由于HTML也是XML的一种,可以认为JS+HTML)元素,也是一个react元素,这个元素实际是一个对象,就是**虚拟DOM元素**

React 认为渲染逻辑本质上与其他 UI 逻辑内在耦合.比如, 在 UI 中需要绑定处理事件、在某些时刻状态发生变化时需要通知到 UI, 以及需要在 UI 中展示准备好的数据。

```
1 import React, { Component } from 'react'
2
3 import logo from './favicon.ico'
4 const ele = <h2>hello,world</h2>
5 function formatName(user) {
6   // 也可以是个表达式
7   return user.firstName + ' ' + user.lastName;
8 }
9 const user = {
10   firstName: '张',
11   lastName: '三丰'
12 }
13
14 //jsx也可以是表达式
15 function getGreeting(user) {
16   if (user) {
17     return <h1>hello {formatName(user)}</h1>
18   }
19   return <h1>hello,小马哥</h1>
20 }
21 export default class App extends Component {
22
23   render() {
24     return (
25       <div>
26         {ele}
```

```
27         {/* jsx运算 */}  
28         {2 + 1}  
29         <br />  
30         {/*jsx嵌入表达式*/}  
31         {formatName(user)}  
32         {/* 添加属性 */}  
33         <img src={logo} alt="" />  
34     </div>  
35     // getGreeting()  
36 )  
37 }  
38 }
```

强烈建议大家阅读官网,针对 `React.render()` [内部的实现操作](#)

## 元素渲染

元素是构成React应用的最小砖块,比如:

```
const ele = <h1>hello,world</h1>
```

与浏览器的 DOM 元素不同, React 元素是创建开销极小的普通对象。React DOM 会负责更新 DOM 来与 React 元素保持一致

上节课的 `ReactDOM.render()` 其实就是在渲染DOM节点

## 更新已渲染的元素

React元素是**不可变对象**,一旦被创建,无妨更改它的子元素或者属性

计时器的例子

```
1 function tick() {
2   const element = (
3     <div>
4       <h1>Hello, world!</h1>
5       <h2>{new Date().toLocaleTimeString()}</h2>
6     </div>
7   );
8   ReactDOM.render(element,
9     document.querySelector('#root'));
10 }
11 setInterval(tick, 1000);
```

大多数情况下,React应用只会调用一次 `ReactDOM.render()`

## React只需要更新它需要更新的部分

React DOM会将元素和它的子元素与它们之前的状态进行比较,并只会进行必要的更新来使DOM达到预期的状态

## 循环绑定元素

当数据从后端请求回来之后,在React中,一般都需要循环绑定元素

### map绑定

在React中,循环绑定元素都是使用 `map` 方法,不能使用 `forEach` 是因为 `forEach` 没有返回值

```
1 let ul = (<ul>
2   { arr.map((item, index)=>{
3     return <li key={index}>{item}</li>
4   }) }
5 </ul>);
```

结果会是一个 JSX 元素组成的数组，放入页面中，不会使用逗号分隔开。

循环绑定的 JSX 元素，必须要有 **key** 属性，来区分不同的元素，否则会报错。

## 过滤元素

同样通过 `map` 方法,只要把不符合条件的元素,返回为 `null` 即可,原因在于,`null` 会被表示为空.如果使用 `filter`,那么就没有办法对元素进行处理,只能过滤,还是需要使用 `map` 进行处理

```
1 let ul = (<ul>
2   { arr.map((item, index)=>{
3     return (
4       item.price < 1000 ? null : <li key={index}>{item}
5     </li>;
6   )
7   }) }
  </ul>)
```

## ref和refs

在React中,类似于Vue,可以通过 `ref` 标记元素,然后通过 `this.refs` 获取元素,只是Vue中使用的是 `this.$refs` 获取元素

## 字符串写法

通过设置一个字符串值来标记元素,然后通过这个字符串作为属性获取元素

```

1 class Input extends Component {
2   handleChange = (e) => {
3     console.log(this.refs.a.value)
4   }
5   render() {
6     return (
7       <div>
8         <input type="text" ref="a" onChange=
9         {this.handleChange} />
10      </div>
11    )
12  }

```

## 函数写法

函数作为一个ref的属性值,这个函数接受一个参数,就是真实的DOM元素

可以把这个元素挂载到实例上,方便后面的操作

```

1 class Input extends React.Component {
2
3   componentDidMount() {
4     console.log(this.a); //获取真实的DOM元素
5   }
6   render() {
7     return (
8       <div>
9         <input type="text" ref={x=>this.a = x}/>
10      </div>
11    );
12  }
13 }

```



# 组件&props

React创建组件有来两种方式

- 函数声明
- 类声明

React组件特点:

- 组件名称应该首字母大写,否则会报错
- 组件定义之后,可以像JSX元素一样使用
- 必须使用 `render` 函数才能将虚拟DOM渲染成真实的DOM
- 使用组件时,可以使用单标签,也可以使用双标签

## 函数组件

组件,从概念上类似于 JavaScript 函数。它接受任意的入参 (即“props”),并返回用于描述页面展示内容的 React 元素。

- 函数声明的组件,必须返回一个JSX元素
- 可以通过属性给组件传递值,函数通过props参数接收

定义组件最简单的方式就是编写JavaScript函数

```
1 function Welcome(props){  
2     return <h2>hello,{props.name}</h2>  
3 }
```

我们称为该组件为"函数组件",因为它本质上就是一个函数

## 类声明组件

使用[ES6的class](#)的方式定义组件

## 类声明组件需要注意

- 在React中有一个属性Component,是一个基类,使用类声明组件时,**必须继承这个基类**
- 在类中,**必须有render函数**, constructor 不是必须的
- 在 render 函数中,**需要return一个JSX元素**

```
1 class Welcome extends Component {  
2     render() {  
3         return (  
4             // 它会将 JSX 所接收的属性 (attributes) 转换为  
             单个对象传递给组件，这个对象被称之为 “props”。  
5             <h2>Welcome,{this.props.name}</h2>  
6         );  
7     }  
8 }
```

注意： 组件名称必须以大写字母开头。 React会将小写字母开头的组件称之为标签

例如,<div /> 代表HTML的div标签,而 <Welcome /> 则代表一个组件,并且需在作用域内使用Welcome

## 两种方式的区别

真实项目中,都只使用class定义组件

- class定义的组件中有this,状态,生命周期
- function声明都没有

## 组合组件

组件可以输出的时候嵌入其他的组件。这就可以让我们用同一组件中来抽离出任意层次的细节(复用组件)。按钮，表单，对话框，甚至整个屏幕的内容：在 React 应用程序中，这些通常都会以组件的形式表示。

什么是复合组件：

- 将多个组件进行组合,例如调用两次相同的组件
- 结构非常复杂时需要将组件拆分成小组件
- 会存在父子关系的数据传递

```
1 import React, { Component } from 'react';
2 import ReactDOM from 'react-dom';
3
4
5 class Welcome extends Component {
6   render() {
7     return (
8       <div>
9         <h3>Welcome,{this.props.name}</h3>
10      </div>
11    );
12  }
13 }
14 class App extends Component {
15   render() {
16     return (
17       <div>
18         <Welcome name='张三'></Welcome>
19         <Welcome name='李四'></Welcome>
20         <Welcome name='王五'></Welcome>
21       </div>
22     )
23   }
```

```
24 }
25 // 组合组件
26 ReactDOM.render(
27   <App></App>
28   ,
29   document.querySelector('#root'));
```

可想而知,React开发其实就是组件化开发,因为React真的是组件化开发的鼻祖

## 提取组件

将组件拆分成更小的组件

```
1
2 import React, { Component } from 'react';
3 import ReactDOM from 'react-dom';
4
5
6 // 所有props是只读的,不能直接修改
7 class Avatar extends Component {
8   render() {
9     return (
10       <img src={this.props.user.avatarUrl} alt=
11         {this.props.user.name} />
12     );
13   }
14 }
15
16 class UserInfo extends Component {
17   constructor(props) {
18     super(props);
19     console.log(this.props);
20   }
```

```
21
22     render() {
23         return (
24             <div className="userinfo">
25                 <Avatar user={this.props.user}>
26                 </Avatar>
27                 <div className='username'>
28                     <h3>{this.props.user.name}</h3>
29                 </div>
30             </div>
31         );
32     }
33 }
34
35
36 class Comment extends Component {
37     constructor(props) {
38         super(props);
39     }
40     render() {
41         return (
42             <div className='comment'>
43                 <UserInfo user={this.props.user}>
44                 </UserInfo>
45                 <div className="Comment-text">
46                     {this.props.user.text}
47                 </div>
48                 <div className="Comment-date">
49                     {this.props.user.date}
50                 </div>
51             </div>
52         );
53     }
54 }
```

```

54 class App extends Component {
55     constructor() {
56         super();
57         this.user = {
58             avatarUrl:
59             'https://hcdn1.apeland.cn/media/course/icon2.png',
60             name: '张三',
61             text: "hello,React component",
62             date: new Date().toLocaleString()
63         }
64     }
65     render() {
66         return (
67             <div>
68                 <Comment user={this.user}></Comment>
69             </div>
70         )
71     }
72 }
73 // 组合组件
74 ReactDOM.render(
75     <App></App>
76     ,
77     document.querySelector('#root'));
78

```

最初看上去，提取组件可能是一件繁重的工作，但是，在大型应用中，构建可复用组件库是完全值得的。根据经验来看，如果 UI 中有一部分被多次使用（Button，Panel，Avatar），或者组件本身就足够复杂（App，Comment），那么它就是一个可复用组件的候选项。你说对吧！

## 父子组件通信

## 父传子

父组件通过行间属性传递数据到子组件,子组件通过实例上的props属性接收新的数据

React 的数据是单向数据流，只能一层一层往下传递。当组件的属性发生改变，那么当前的视图就会更新

## 子传父

通过给子组件传递一个函数，子组件调用父亲的函数将值作为参数传递给父组件，父组件更新值，刷新视图。

父组件中定义一个函数，通过属性传递给子组件。

这个传递的函数必须是一个箭头函数

```
1 import React, { Component } from 'react';
2 import ReactDOM from 'react-dom';
3
4 class ChildCom extends Component {
5     constructor(props) {
6         super(props);
7         console.log(props);
8         this.state = {
9             val: ''
10        }
11        this.handleClick =
12        this.handleClick.bind(this);
13        this.handleChange =
14        this.handleChange.bind(this);
15    }
16    handleChange(event){
17        this.setState({
18            val: event.target.value
19        })
20    }
21 }
```

```

18     }
19     handleClick(){
20         if(this.state.val){
21             this.props.addHandler(this.state.val);
22             // 清空输入框
23             this.setState({
24                 val: ''
25             })
26         }
27     }
28     render() {
29         return (
30             <div>
31                 <input type="text" value =
32 {this.state.val} onChange = {this.handlerChange}/>
33                 <button onClick={this.handleClick}>添
34 加</button>
35                 {
36                     this.props.menus.map((item,index)
37 => {
38                         return <p key = {index}>{item}
39 </p>
40                     })
41                 }
42             </div>
43         );
44     }
45 }
46
47 class App extends Component {
48     constructor(props) {
49         super(props);
50         this.state = {

```



```

49         menus: ['烤腰子', '辣炒鸡丁', '炸黄花鱼']
50     }
51 }
52 // 一定要使用箭头函数
53 addHandler = (val) => {
54     this.state.menus.push(val);
55     this.setState({
56         menus: this.state.menus
57     })
58 }
59
60 render() {
61     // 修改状态之后, 会重新调用render
62     return (
63         <div>
64             <ChildCom menus={this.state.menus}
65             addHandler={this.addHandler}></ChildCom>
66         </div>
67     );
68 }
69
70
71
72 ReactDOM.render(<App />,
    document.querySelector('#root'));

```

## Props的只读性

组件无论是使用[函数声明](#)还是通过 [class 声明](#)，都决不能修改自身的 props

```
1 //该函数不会尝试更改入参，且多次调用下相同的入参始终返回相同
  的结果。
2 function sum(a, b) {
3     return a + b;
4 }
5 //它更改了自己的入参
6 function withdraw(account, amount) {
7     account.total -= amount;
8 }
```

所有 React 组件都必须像纯函数一样保护它们的 props 不被更改。

## state

### 组件状态

组件中数据的来源

- 属性:是由外接传递过来的
- 状态:是自己的,只能通过 `setState` 来改变状态

只有类声明的组件中,才有状态

### 修改状态

除了 `constructor` 之外的其它地方,如果需要修改状态,都只能通过 `this.setState` 方法

这个方法传入的第一个参数,可以是一个对象,也可以是一个函数

- 是一个对象，这个对象中包含需要改变的属性，它会与原有的状态进行合并
- 是一个函数，接收第一个参数是 `prevState`，上一个状态对象，第二个参数是 `props`

这个方法的第二个参数，是一个回调函数，在状态改变之后执行。

如果下一个状态依赖于上一个状态，需要写成函数的方式

## 关于setState

- 在 react 组件的生命周期或事件的绑定中，setState 是异步的
- 在定时器或原生的事件中，setState 不一定是异步的

```
1 // state.count 当前为 0
2 componentDidMount(){
3     this.setState({count: this.state.count + 1});
4     console.log(this.state.count)
5 }
6 // 输出 0
```

在元素渲染章节中，我们只了解了一种更新 UI 界面的方法。通过调用 `ReactDOM.render()` 来修改我们想要渲染的元素

```
1 function tick() {
2     const element = (
3         <div>
4             <h1>Hello, world!</h1>
5             <h2>{new Date().toLocaleTimeString()}</h2>
6         </div>
7     );
8     ReactDOM.render(element,
9         document.querySelector('#root'));
10 }
11 setInterval(tick, 1000);
```

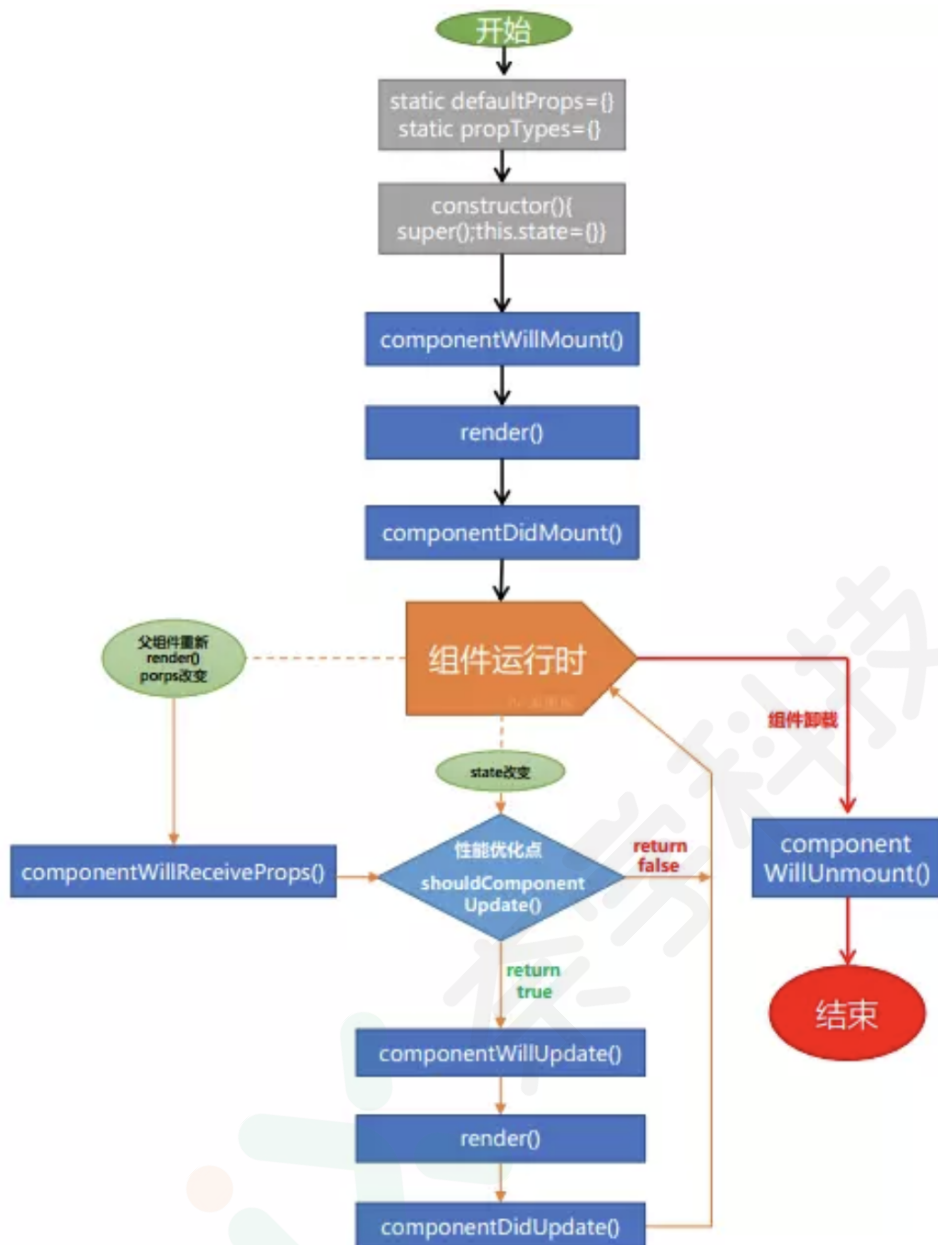
## 本节学习如何封装真正可复用的Clock组件

```
1 import React, { Component } from 'react';
2 import ReactDOM from 'react-dom';
3
4 // 学习如何封装真正可复用的Clock组件。
5 class Clock extends Component {
6   constructor(props) {
7     super(props);
8     this.state = {
9       date: new Date().toLocaleString()
10    }
11  }
12  componentDidMount() {
13    this.timer = setInterval(() => {
14      // 注意1 不能直接修改state
15      // this.state.date = new Date(); //错误
16
17      // 注意2:  setState()是异步的
18      this.setState({
19        date: new Date().toLocaleString()
20      });
21    }, 1000);
22  }
23  componentWillUnmount() {
24    clearInterval(this.timer);
25  }
26  render() {
27    // 修改状态之后,会重新调用render
28    return (
29      <div>
30        <h3>当前时间为:{this.state.date}</h3>
31      </div>
32    );
```

```
33     }  
34 }  
35  
36  
37  
38 ReactDOM.render(<Clock />,  
    document.querySelector('#root'));
```

## 生命周期





如图，React生命周期主要包括三个阶段：初始化阶段、运行中阶段和销毁阶段，在React不同的生命周期里，会依次触发不同的钩子函数，下面我们就来详细介绍一下React的生命周期函数

```

1 import React, { Component } from 'react';
2 import ReactDOM from 'react-dom';
3

```

```
4 class SubCounter extends Component {
5     // 组件将要接收属性
6     componentWillReceiveProps(newProps){
7         console.log('9.子组件将要接收到新属
性',newProps);
8     }
9     shouldComponentUpdate(newProps,newState){
10         console.log('10.子组件是否需要更新')
11         if(newProps.num % 3 === 0){
12             return true;
13         }else{
14             return false;
15         }
16     }
17 }
18 componentWillUpdate() {
19     console.log('11、子组件将要更新');
20 }
21
22 componentDidUpdate() {
23     console.log('13、子组件更新完成');
24 }
25
26 componentWillUnmount() {
27     console.log('14、子组件将卸载');
28 }
29 render() {
30     console.log('12.子组件挂载中')
31     return (
32         <div>
33             <p>{this.props.num}</p>
34         </div>
35     );
36 }
37 }
```

```
38
39
40 class Counter extends Component {
41     static defaultProps = {
42         //1.加载默认属性
43         name: '小马哥',
44         age: 18
45     }
46     constructor(props) {
47         super(props);
48         //2.记载默认状态
49         this.state = {
50             num: 0
51         }
52     }
53     componentWillMount() {
54         // 此时可以访问属性和状态，可以进行api调用，但没办法做DOM相关操作
55         console.log('3.父组件将要被挂载');
56
57     }
58     componentDidMount() {
59         // 组件已挂载，可进行状态更新操作。通常 都在此方法中发送请求
60         console.log('5.组件挂载完成');
61     }
62
63     shouldComponentUpdate(newProps, newState) {
64         // 组件是否需要更新，返回布尔值，优化点
65         console.log('6.父组件是否被更新');
66         // console.log(newProps, newState);
67         if (newState.num % 2 === 0) {
68             return true;
69         } else {
```



```
70          // 此函数 会返回一个boolean值,返回true更新页
           面,返回false不更新页面
71          return false;
72      }
73  }
74
75  componentWillUpdate(){
76      console.log('7.父组件将要更新');
77
78  }
79  componentDidUpdate(){
80      console.log('8.父组件更新完成');
81
82  }
83
84  handlerClick = () => {
85      // 可能,只是说可能,官网上都是这样说的.....会导
           致计数可能不准确,
86      // this.setState({
87      //     num: parseInt(this.props.increment) +
           this.state.num
88      // })
89      // 发现点击之后,得到的结果为0,这是因为setState()
           是异步的
90      // console.log(this.state.num);
91
92      // 要解决这个问题,可以让 setState() 接收一个函数
           而不是一个对象。
93      // 这个函数用上一个 state 作为第一个参数,将此次
           更新被应用时的 props 做为第二个参数
94      this.setState((state, props) => {
95          return {
96              num: state.num +
           parseInt(props.increment)
97          }
```

```

98         }, () => {
99             console.log(this.state.num);
100         })
101     }
102     render() {
103         // 修改状态之后,会重新调用render
104         console.log('4.render(父组件)渲染了');
105
106         return (
107             <div>
108                 <h3>当前数值:{this.state.num}</h3>
109                 <button onClick=
110                 {this.handleClick}>+1</button>
111                 <h3>我是子组件</h3>
112                 <SubCounter num = {this.state.num}>
113             </SubCounter>
114             </div>
115         );
116     }
117 }
118
119 ReactDOM.render(<Counter increment='1' />,
    document.querySelector('#root'));

```

## 受控组件

受控组件，就是受状态控制的组件，需要与状态进行相应的绑定

- 受控组件必须要有一个 **onChange 事件**，否则不能使用
- 受控组件可以**赋予默认值**（实际上就是设置初始状态）

- 官方推荐使用受控组件的写法

可以使用受控组件实现双向绑定。

非受控组件，则不是通过与状态进行绑定来实现的，而是通过操作 **DOM** 来实现。除非操作 DOM，否则没有办法设置默认值。

## 受控组件实现

- 设置初始状态，也就是设置默认值
- 将输入框的 value 值与相应状态进行绑定
- 使用 onChange 事件，对状态进行修改，从而反映到 value 上

```
1 class Input extends Component {
2   constructor() {
3     super();
4     this.state = {
5       val: '' // 这个位置用来设置默认值
6     }
7   }
8   handleChange = (e) => {
9     let val = e.target.value
10    this.setState({val});
11  }
12  render() {
13    return (
14      <div>
15        // 让 value 与状态进行绑定，通过事件处理修改状态来
        达到修改值的效果
16        <input type="text" value={this.state.val}
17        onChange={this.handleChange} />
18      {this.state.val}
19    )
20  }
21 }
```

```
18     </div>
19   )
20 }
21 }
```

## 非受控组件的实现

- 通过 `ref` 标记一个元素，然后通过 `this.refs.xx` 来获取这个元素
- 通过 `onChange` 事件监听到 `value` 的变化，获取到这个数据
- 然后通过操作 DOM 将数据放到需要的地方

```
1 class Input extends Component {
2   constructor() {
3     super();
4     this.state = {
5       val: ''
6     }
7   }
8   handleChange = (e) => {
9     //=> 这里可以通过 e.target.value 获取
10    let val = this.refs.a.value;
11    this.setState({val});
12  }
13  render() {
14    return (
15      <div>
16        <input type="text" onChange={this.handleChange}
17        ref="a" />
18        {this.state.val}
19      </div>
20    )
21  }
```

实际上，上面实现的只是单向绑定，如果要实现双向绑定，需要使用两个 onChange 事件，可以不需要用到状态

下面来实现双向绑定：

```
1 class Input extends Component {
2   handleChange = (e) => {
3     //=> 这里可以通过 e.target.value 获取
4     if (e.target === this.refs.a) {
5       this.refs.b.value = e.target.value;
6     } else {
7       this.refs.a.value = e.target.value;
8     }
9   }
10  render() {
11    return (
12      <div onChange={this.handleChange}>
13        <input type="text" ref="a" />
14        <input type="text" ref="b" />
15      </div>
16    )
17  }
18 }
```