爱创课堂前端培训

# ReactJS第二期

课程：爱创课堂公开课

讲师：张容铭

日期：2016年12月10日

张容铭老师

微博：@张容铭\_YYQH

QQ : 286031482

E-mail : yuye\_qinghe@qq.com

爱创课堂官网 ：[www.icketang.com](http://www.icketang.com)

[ReactJS第二期 1](#_Toc14322)

[复习 3](#_Toc2493)

[样式的设置 5](#_Toc4810)

[注释 5](#_Toc4041)

[事件 6](#_Toc22025)

[状态 6](#_Toc31358)

[无状态（stateless）组件 7](#_Toc23221)

[有状态组件 7](#_Toc30652)

[创建初始化状态 7](#_Toc13396)

[更改状态 7](#_Toc7539)

[组件构建完成 7](#_Toc14227)

# 复习

学习React

两个半API

React.createElement 用来创建虚拟DOM。

第一个参数表示元素名称（也可以是组件的名称）

第二个参数表示属性对象

从第三个参数开始表示该虚拟dom的子虚拟dom

ReactDOM.render 将虚拟dom渲染到页面中

第一个参数表示虚拟dom（如果是组件要转化成虚拟dom）

第二个参数表示真实dom容器元素

第三个参数表示回调函数

React.createClass 用来创建组件的

参数是一个对象，用来描述组件的

Render方法输出组件中虚拟DOM树的

作用域是组件实例化对象

必须有返回值是虚拟DOM树

JSX语法，用来简化虚拟dom创建

核心内容：像创建XML元素一样创建虚拟DOM

例如创建div虚拟DOM元素 <div></div>

所以为了让浏览器支持，我们需要编译代码

一种是在浏览器端引入bowser.js解析，注意，将script标签type类型设置成text/babel

一种在工程化中使用，以fis为例

通过babel2插件解析react

第一步 获取所有jsx文件

第二步 引入插件

第三步 将jsx拓展名转化成js

特殊元素属性

Class（js中的一个保留字）要改成className

For（js中的关键）要改成htmlFor

插值

语法 {js表达式}

Js表达式可以是变量，可以是运算，可以是方法等等

Map方法

用来遍历数组并映射输出

回调函数有三个参数

第一个参数表示数组成员值

第二个参数表示数组索引值

第三个参数表示原数组

返回值即使方法执行后数组成员

非元素属性key

用来标示列表元素

组件的属性

用来存储数组让组件展示不同的样式

定义默认属性 getDefaultProps方法来创建默认属性

# 样式的设置

React支持为虚拟dom添加css样式，通过style属性

Style属性值不能是字符串，只能是一个对象

如果css属性的名称是有多个单词构成的，要转化成驼峰式命名

Eg: font-size => fontSize

如果css属性名称有css3前缀，注意第一个字母也要大写

Eg:-webkit-box-shadow => WebkitBoxShadow

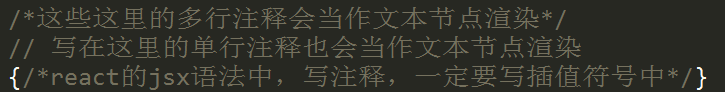


|  |
| --- |
| 1. // 定义mystyle变量 2. var myStyle = { 3. // 字体颜色是绿色 4. color: 'green', 5. // 字体大小是50px 6. fontSize: '50px', 7. // 设置投影 8. WebkitBoxShadow: '10px 10px red' 9. }; 10. // 在jsx语法中使用js对象，要使用插值，插值的语法就是{变量} 11. var h1 = (<h1 style={myStyle}>爱创课堂</h1>); |

# 注释

React的jsx语法中，书写注释只能书写在插值符号中，并且是多行注释

如果写在插值符号外面，无论单行注释还是多行注释，都会当作文本节点渲染



|  |
| --- |
| 1. render: function() { 2. // 必须通过return返回虚拟dom树 3. console.log(this) 4. return ( 5. <div> 6. /\*这些这里的多行注释会当作文本节点渲染\*/ 7. // 写在这里的单行注释也会当作文本节点渲染 8. {/\*react的jsx语法中，写注释，一定要写插值符号中\*/} 9. <button onClick={this.clickBtn}>按钮</button> 10. <p>点击了{this.state.num}次</p> 11. </div> 12. ) 13. } |

# 事件

为虚拟dom定义事件跟在html中为元素定义事件的方式是一样的

语法 on+事件名称（事件名称首字母大写）

我们可以通过this为事件回调函数绑定组件实例化对象中方法  
 事件回调函数绑定的方法不要加上()执行

事件回调函数

作用域是组件实例化对象，我们可以在事件回调函数中，使用组件实例化对象

参数

第一个参数表示react封装的事件对象（react建议使用）

第二个参数表示reactID

第三个参数表示原生事件对象



|  |
| --- |
| 1. var Demo = React.createClass({ 2. // 定义默认状态 3. getInitialState: function() { 4. // 必须通过return 返回初始化状态 5. return { 6. // 表示点击的次数 7. num: 0 8. } 9. }, 10. // 定义一个事件回调函数方法 11. clickBtn: function () { 12. // 触发交互，我们要改变状态中存储的数据 13. this.setState({ 14. // 改变的状态数据 15. num: ++this.state.num 16. }) 17. }, 18. // 虚拟DOM数输出方法 19. render: function() { 20. // 必须通过return返回虚拟dom树 21. return ( 22. <div> 23. <button onClick={this.clickBtn}>按钮</button> 24. <p>点击了{this.state.num}次</p> 25. </div> 26. ) 27. } 28. }) |

# 状态

根据组件的状态特征分成两类，

一类是有状态组件，

一类是无状态组件

## 无状态（stateless）组件

如果组件创建以后不在发生改变，这一类组件称之为无状态

这类组件的形态是由组件的属性控制的，一旦创建不会发生改变

## 有状态组件

如果组件创建后，会根据内部存储的状态的改变而改变，这类组件称之为有状态组件

状态跟属性一样，都是为组件存储数据

属性的数据是由外部提供的

状态的数据是内部维护的（组件内部发生事件交互，组件内部发生异步请求）

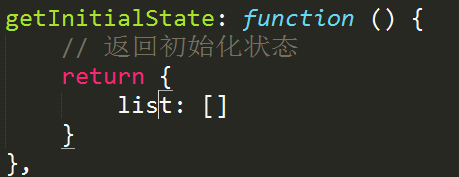
状态还是属性的改变都会触发组件重新渲染（触发render执行）

## 创建初始化状态

getInitialState

返回值就是初始化状态数据

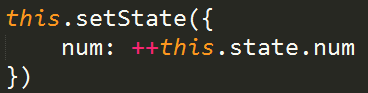
必须通过return返回



## 更改状态

在组件内部更改状态用setState方法

参数就是改变的状态数据



|  |
| --- |
| 1. getInitialState: function() { 2. // 必须通过return 返回初始化状态 3. return { 4. // 表示点击的次数 5. num: 0 6. } 7. }, |

# 组件构建完成

componentDidMount表示组件构建完成时候的回调函数

通常在这个阶段使用侵入式插件绑定事件，请求数据等等

因为此时组件已经构建完成



|  |
| --- |
| 1. // 获取数据通常要等到组件构建完成 2. componentDidMount: function () { 3. // 缓存this 4. var me = this; 5. // 请求数据，并将结果保存在内部变量list中 6. ajax('data/skin.json', function (res) { 7. if (res && res.errno === 0) { 8. // 将结果保存在组件中 9. // console.log(me) 10. me.setState({ 11. list: res.data 12. }) 13. } 14. // console.log(res) 15. }) 16. } |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |