**react框架学习15.5的版本**

学习网址：<https://discountry.github.io/react/>

菜鸟教程

React重点：

Props：在父组件中定义属性，通过属性向子组件传递数据（｛this.props.xxx｝）

子组件

react介绍

发展史：首先是使用jquery DOM的操作时代，简称面向过程开发🡪seajs和requirejs（时间段2012年左右至2014年，同时出现grunt和gulp前端构建工具）模块化的开发方式（把一个javascript功能封装成一个js文件，通过module.exports对外提供方法）🡪2014年至今发展成组件化，将DOM、js、css都封装到一个文件然后通过export暴露出去，组件化开发的框架把这个文件转成自定义的html标签（掌握react和vue）

1. react框架小版本之间没有什么区别，大版本之间有相应的区别
2. React 是一个用于构建用户界面的 JAVASCRIPT 库
3. React 起源于 Facebook 的内部项目，用来架设 Instagram 的网站，并于 2013 年 5 月开源
4. React 拥有较高的性能，代码逻辑非常简单
5. React 15.4版本和15.5版本的区别

1）15.4版本的数据验证包含在主文件（react.min.js文件）中

2）15.5版本把数据验证这一部分抽离出来单独形成一个文件（prop-types.js文件）

1. 16.0的版本即将发布

特点

1. **声明式设计** −React采用声明范式，可以轻松描述应用

**说明：**我们所有做的事情都是基于组件。一个项目都是先声明一个个的组件，最后将这些组件拼接起来，一个组件就是一个HTML标签。render()函数里面包含的内容就是组件。整个项目只有一个HTML文件，只需要往这个HTML文件中添加组件即可

1. **高效** −React通过对DOM的模拟，最大限度地减少与DOM的交互

**说明：**react采用的是虚拟DOM，而不是真正的DOM节点

1. **灵活** −React可以与已知的库或框架很好地配合

**说明：**react不限制使用任何框架和任何的库。只做一件事情就是封装组件

1. **JSX** − JSX 是 JavaScript 语法的扩展。React 开发不一定使用 JSX ，但我们建议使用它

**说明：**JSX语法使得可以在JavaScript中写DOM（虚拟），是通过create创建出来的

1. **组件** − 通过 React 构建组件，使得代码更加容易得到复用，能够很好的应用在大项目的开发中
2. **单向响应的数据流** – 数据只能从父组件流向子组件，不能从子组件传递向父组件

**说明：**数据只能从父组件流向子组件，不能从子组件传向父组件（子组件直接修改父组件的值）。是为了防止数据混乱（当子组件可以改变父组件的数据时，其它组件的数据同时也会发生改变，这时数据就比较混乱了）；耦合（当子组件直接改变父组件的数据时，就耦合了，组件将复用不了了）

**使用react，ReactJSX**

1. 引用react的核心库react.min.js（模拟DOM），在开发时一定要用开发者版本react.js，不然不会报错
2. 引用react的渲染库react-dom.min.js（把虚拟DOM渲染成真实DOM之后插入到DOM节点中），在开发时一定要用开发者版本react-dom.js。不然不会报错

说明（1）和（2）两条：版本13的时候核心库和渲染库在一起，版本14的时候（至今）分离出来

1. 引用babel.js帮我们转换ES6（将ES6代码转为ES5代码）和JSX语法

JSX语法创建虚拟DOM，代码如下：

<script type="text/babel">

ReactDOM.render( //把组件渲染成真实的DOM并插入到DOM节点中

<h1>Hello,world!</h1>, //虚拟DOM，也可称为组件

document.getElementById('example')//插入到真实DOM中

);

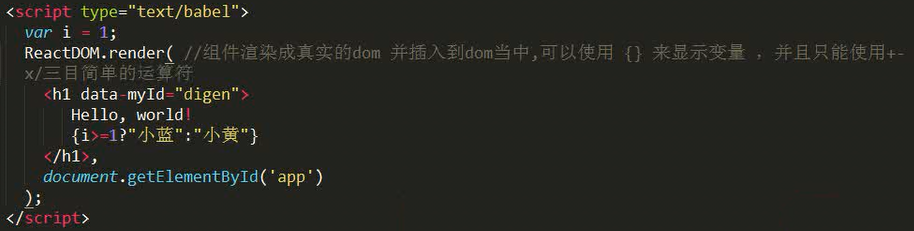
</script>

**注意：**使用react框架大多数是单页面应用。单页面应用时整个项目中只有一个ReactDOM.render()。在里面插入一个大组件，其余的组件都是插入到这个大组件中

1. ReactJSX语法规则：
   * 1. 向元素（虚拟DOM）添加自定义属性时需要使用 data- 前缀，不然会报错。如下：

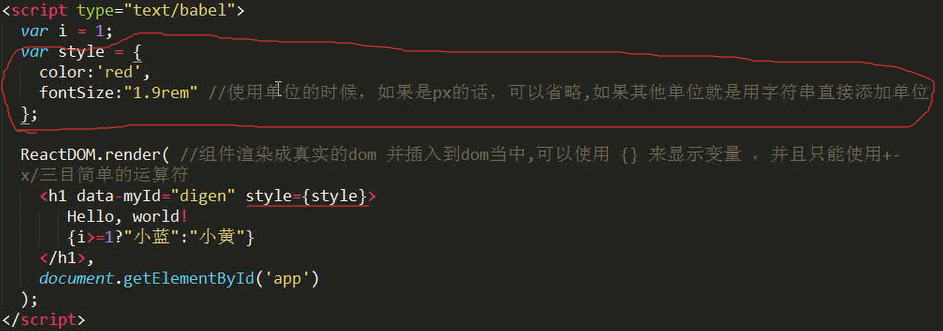


* + 1. 可以在 JSX 中使用 JavaScript 表达式。表达式写在花括号 {} 中，代码如下：

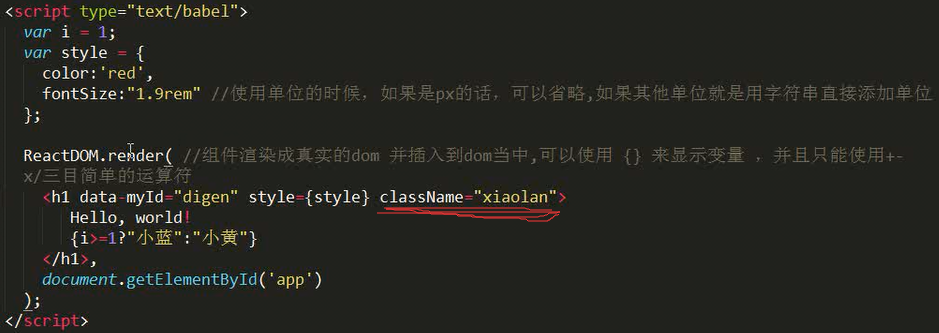


* + 1. 添加样式，react推荐使用内联样式

Ⅰ内联样式，代码如下：



Ⅱ添加class类名使用className=”xxx”，代码如下：



* + 1. 在render函数中的注释，使用{/\*注释\*/}，方法如下：



* + 1. 允许在模板中插入数组，数组会自动展开所有成员，语法如下：

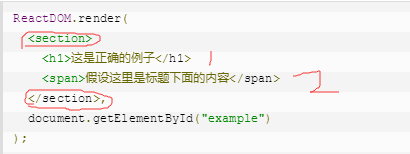


* + 1. HTML标签和React组件的区别

Ⅰ 、HTML标签，以小写开头

Ⅱ、React组件也是一个标签，但是所有的React组件必须以大写开头，不然会报错

* + 1. 代码中嵌套多个 HTML 标签，需要使用一个标签元素包裹它



**创建React组件和使用**

**注意：**原生 HTML 元素名以小写字母开头，而自定义的 React 组件必须以大写字母开头，比如 Hello 不能写成 hello。除此之外还需要注意组件类只能包含一个顶层标签，否则也会报错。

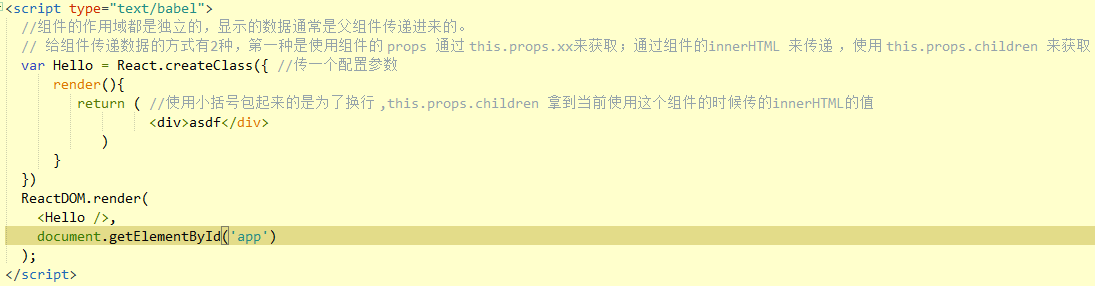
**说明：**一个组件封装了模版、逻辑和样式，组件的模版放在render（）函数里面

1. 语法如下：

**var Hello = React.createClass** 方法用于生成（创建）一个组件类

**<Hello />** 实例组件类并输出信息（输出组件信息）



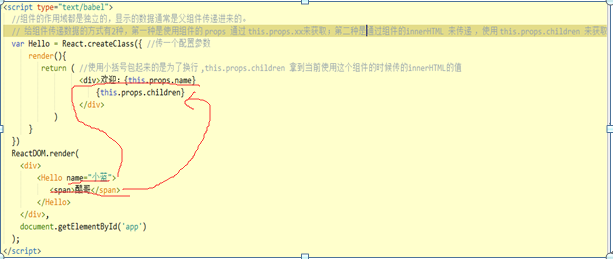


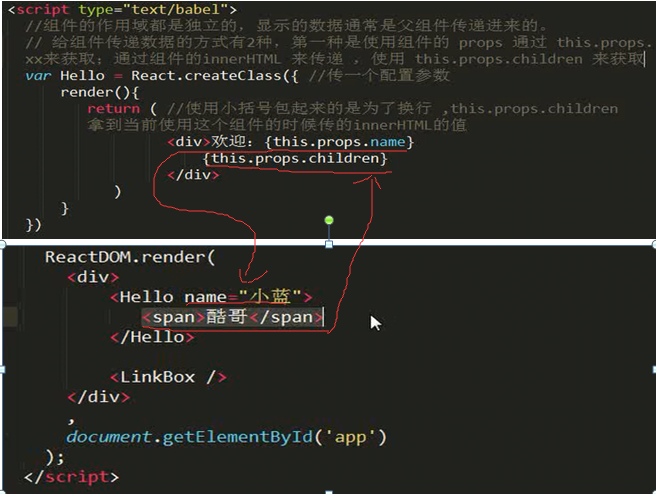
（2）如果需要向组件传递参数（只能从父组件向子组件传递数据），有两种 方式，第一种是使用组件的 props 通过 this.props.xx来获取；第二种是通过组件的innerHTML 来传递 ，使用 this.props.children 来获取,实例如下：

需要向组件传递参数，可以使用 this.props 对象

this表示当前的组件

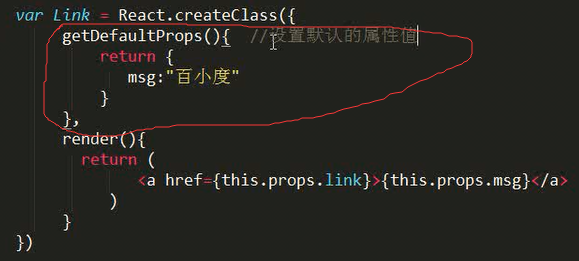
xx表示父组件传过来的属性名





1. 创建组件时给组件设置属性默认值（意义是当父组件没有传递属性时，就使用默认属性值，传递了则使用父组件属性）。使用getDefaultProps函数，如下：





**React复合组件**

顾名思义就是组件套组件，代码如下：



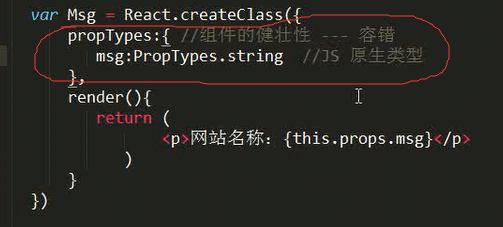


**React Props验证（属性的验证）**

介绍： props提供propTypes方法来验证传入数据是否有效， 传入无效数据时，JavaScript 控制台会抛出警告，以保证我们的应用组件被正确使用，保证组件的健壮性（容错率高），15.5版本将propTypes方法分离出来单独形成一个文件prop-types.js文件，使用时需要引入这个文件

**为什么要进行属性验证：**增加组件的健壮性，在别人使用的时候，我们在使用开发者版本时能够让别人有一个清晰的错误提示（发出警告）

在创建组件时可以进行props验证（传递类型不一致时会发出警告，注意引用的文件要是开发者版本，不然不会发出警告），如下：



更多验证器如下：

（网址：

<https://discountry.github.io/react/docs/typechecking-with-proptypes.html>）



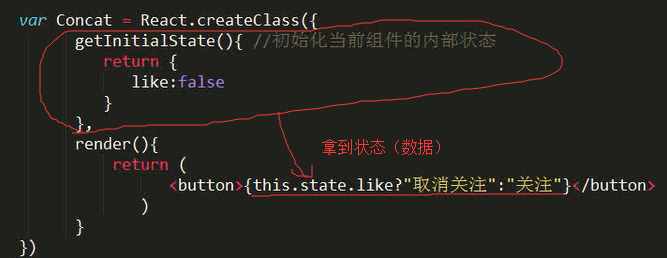


**React State(状态)**

**介绍说明：**默认情况下加载一条数据，将数据显示在DOM节点上，称为状态，数据变化之后称为状态改变。有时候会定义组件内部的数据，这时我们把组件内部定义的数据称为state（状态）。在react中，我们如果（调用某个函数或调用某个方法）跟新（数据）状态，会自动重新调用render函数来重新渲染（刷新）页面，我们称为数据驱动视图（简而言之就是数据改变，视图就改变，我们不需要做任何的DOM操作）。以后慢慢会把数据称为状态

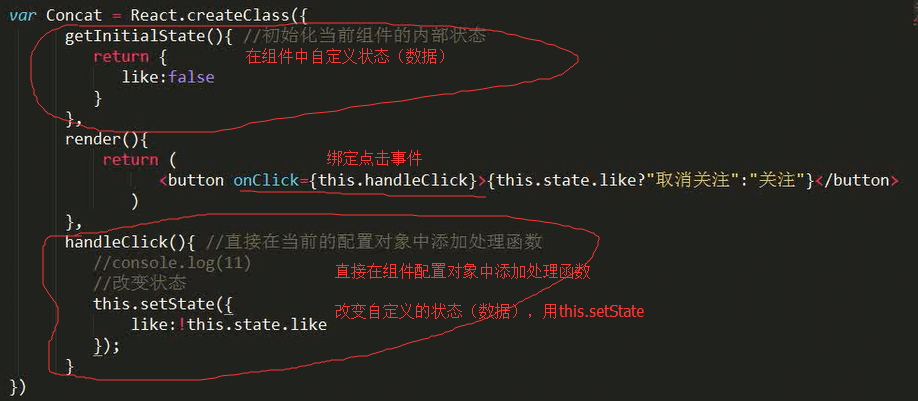
状态就是供组件内部使用的数据

1. 自定义初始化组件内部的状态（数据），通过getInitialState函数定义，通过this.state.xxx拿到（读取）自定义的状态（数据）来使用，如下：

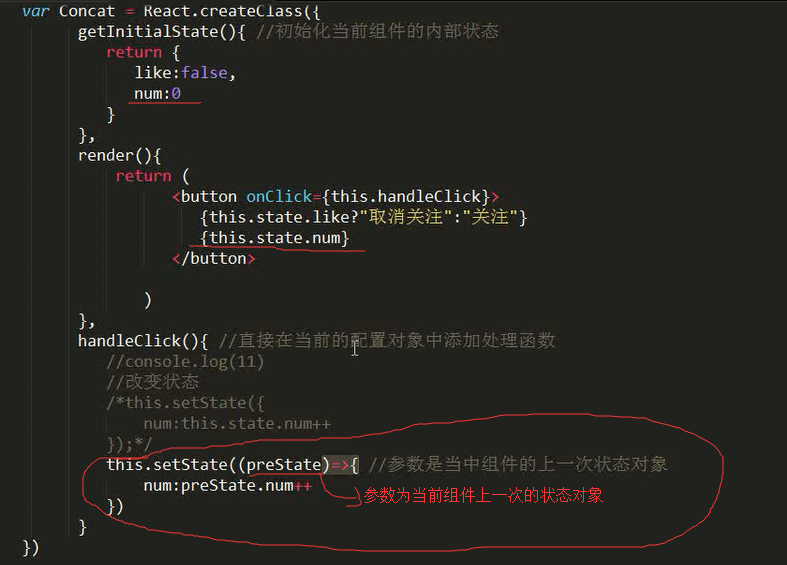


下面我们以点击按钮改变关注和取消关注为例，这时就需要改变状态（数据）用this.setState方法，如下：

1. 绑定点击事件
2. 直接在当前的配置对象中添加处理函数
3. 用this.setState 方法修改自定义状态



1. 如果想使用上一个状态作为赋值给下一个状态，再渲染页面，就需要传一个函数，参数表示当前组件的上一次的状态对象，如下：



var Concat = React.createClass({

getInitialState(){ //初始化当前组件的内部状态

return {

like:false,

num:0

}

},

render(){

return (

<button onClick={this.handleClick}>

{this.state.like?"取消关注":"关注"}

{this.state.num}

</button>

)

},

handleClick(){ //直接在当前的配置对象中添加处理函数

//改变状态

this.setState((preState)=>{ //参数是当中组件的上一次状态对象

num:preState.num++

})

}

})

ReactDOM.render(

<div>

<Concat />

</div>

,

document.getElementById('app')

); ****

使用ES6 class类的继承创建组件

1. 创建一个名称扩展为 React.Component 的[ES6 类](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Classes)
2. 调用render函数
3. 在类中初始化状态使用constructor函数

**说明：**在类中定义的都是实例方法，采用ES6方法进行状态初始化时使用constructor，中间在定义继承时没有逗号，当前指类默认情况下是没有自己的this对象，如果想拿到this对象则必须显示定义constructor函数并调用super函数，这是下面才能拿到this（此时this是父节点的this），当我们new类名时，this指向的是这个类名（组件名），this.state={}

1. 我们通过类的继承，在实例上面增加方法时，默认情况下我们拿不到实例函数体内的this，在模版中不知道this是谁，想拿到this则要在类初始化时定义this.xxx=this.xxx.bind(this)bind函数返回的是一个函数,综合代码如下：



事件的处理，通过在虚拟dom上绑定 onclick={this.handleClick}方法,注意此时handleclick函数内部是没有this的，则需要在constructor中手动绑定this ： this.handleClick = this.handleClick.bind(this)，如下：

class Concat（类名） extends React.Component {

constructor(){//初始化状态

super();

this.state={

like:false,

num:0

};

this.handleClick = this.handleClick.bind(this);//返回一个函数

}

render(){

return (

<button onClick={this.handleClick}>

{this.state.like?"取消关注":"关注"}

{this.state.num}

</button>

)

}

handleClick(){

//console.log(this)

this.setState((preState)=>{ //参数是当中组件的上一次状态对象

num:preState.num++

})

}

}

ReactDOM.render(

<div>

<Concat />

</div>

,

document.getElementById('app')

);



第二种处理事件的方式，绑定箭头函数，如下：



**注意：**当有多个按钮时，我们不可能给每个都添加一个事件，这时可以给每个DOM添加一个自定义属性，使用event.target.dataset来获取添加的自定义属性的对象，如下： 

**props和state的区别**

props只能从父组件向子组件传递数据，有两种方式，第一种是使用组件的 props 通过 this.props.xx来获取；第二种是通过组件的innerHTML 来传递 ，使用 this.props.children 来获取，并提供propTypes属性验证的方法

state是组件内部的状态，供组件内部使用，并且这些状态只能在组件内部进行修改

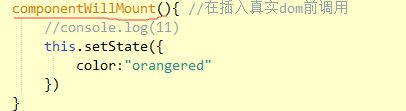
**react组件生命周期（钩子函数）**

组件的生命周期可分成三个阶段：

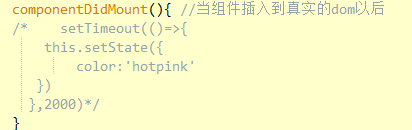
1. Mounting：已插入真实 DOM（第一阶段）
2. Updating：正在被重新渲染（第二阶段）
3. Unmounting：已移出真实 DOM（第三阶段）
4. 三个阶段的方法都应写在组件内

第一阶段Mounting的方法：

（1）componentWillMount 在插入真实DOM前调用在客户端也在服务端，方法如下：

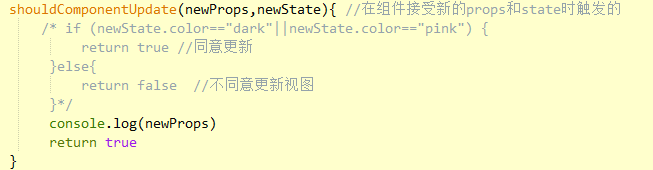


（2）componentDidMount 在组件插入到真实的DOM以后调用，只在客户端，发放如下：



第二阶段Updating的方法：

（1）shouldComponentUpdate 返回一个布尔值，在组件接收到新的props或者state时被调用，方法如下：



（2）componentWillUpdate在组件接收到新的props或者state但还没有render时被调用。在初始化时不会被调用，方法如下：



（3）componentDidUpdate 在组件完成更新后立即调用。在初始化时不会被调用，方法如下：



第三阶段Unmounting的方法：

（1）componentWillUnmount在组件从 DOM 中移除的时候立刻被调用，方法如下：



属性的验证，代码是单独独立的。如下：



**React条件渲染**

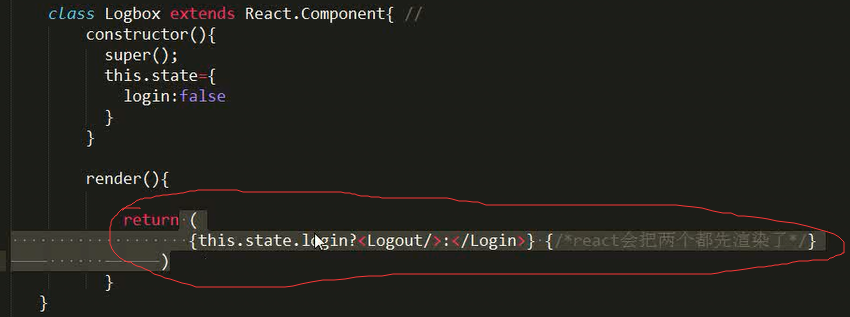
**说明：**我们根据不同的情景来展示不同的组件。以登录和登出为例，如下：

1. 先创建两个组件，分别是Login和Logout，如下：



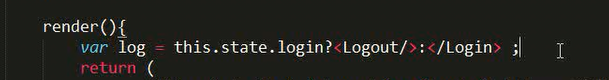
1. 创建一个大的组件Logbox来装Login和Logout两个组件，并判断显示哪个组件，如下：

此法不对

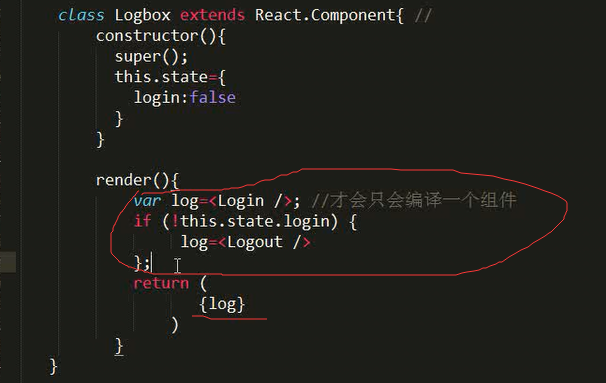


正确方法如下：

1）



2）



**React列表渲染**

说明：外面是一个大组件，里面包含一个个的小组件形成的列表。

先创建一个大组件Joblist和一个小组件Jobitem,在大组件中constructor初始化状态（数据），然后在render函数中用map（）进行遍历（遍历不要放在return中）map函数是遍历数组执行某个函数并返回新的数组，具体如下： 



**React表单和ref（直接操作DOM）**