# React学习文档

学习文档地址：http://www.css88.com/react/docs/introducing-jsx.html

## 一、快速开始

npm install -g create-react-app

create-react-app my-app

npm start

## 二、JSX语法



JSX语法是JavaScript的一种扩展语法。

JSX语法可以生产React元素。

### 1、JSX嵌入表达式

ReactDOM.render(

<ul>

<li>插入变量：{user.name}</li>

<li>插入方法：{getUserName(user)}</li>

<li>插入表达式：{1 + 1}</li>

</ul>,

document.getElementById('root')

);

### 2、JSX也是一个表达式

编译之后，JSX 表达式就变成了常规的 JavaScript 对象。

这意味着你可以在 if 语句或者是 for 循环中使用 JSX，用它给变量赋值，当做参数接收，或者作为函数的返回值。

function getGreeting(user) {

if (user) {

return <h1>Hello, {formatName(user)}!</h1>;

}

return <h1>Hello, Stranger.</h1>;

}

### 3、用JSX制定属性值

使用双引号来指定字符串字面量作为属性值

const element = <div tabIndex="0"></div>;

用花括号嵌入一个 JavaScript 表达式作为属性值

const element = <img src={user.avatarUrl}></img>;

在属性中嵌入 JavaScript 表达式时，不要使用引号来包裹大括号。否则，JSX 将该属性视为字符串字面量而不是表达式。对于字符串值你应该使用引号，对于表达式你应该使用大括号，但两者不能同时用于同一属性。

### 4、用JSX指定子元素

如果是空标签，您应该像 XML 一样，使用 />立即闭合它

const element = <img src={user.avatarUrl} />;

JSX 标签可能包含子元素：

const element = (

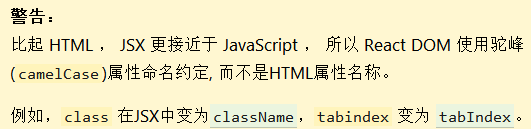
<div>

<h1>Hello!</h1>

<h2>Good to see you here.</h2>

</div>

);



### 5、JSX防止注入攻击

在JSX中嵌入用户输入是安全的：

const title = response.potentiallyMaliciousInput;

// 这样是安全的:

const element = <h1>{title}</h1>;

默认情况下， 在渲染之前, React DOM 会格式化([escapes](http://stackoverflow.com/questions/7381974/which-characters-need-to-be-escaped-on-html)) JSX中的所有值. 从而保证用户无法注入任何应用之外的代码. 在被渲染之前，所有的数据都被转义成为了字符串处理。 以避免 [XSS(跨站脚本)](https://en.wikipedia.org/wiki/Cross-site_scripting) 攻击。

### 6、JSX表示对象

Babel 将JSX编译成 React.createElement() 调用。

下面的两个例子是是完全相同的：

const element = (

<h1 className="greeting">

Hello, world!

</h1>

);

const element = React.createElement(

'h1',

{className: 'greeting'},

'Hello, world!'

);

React.createElement() 会执行一些检查来帮助你编写没有bug的代码，但基本上它会创建一个如下所示的对象：

// 注意: 这是简化的结构

const element = {

type: 'h1',

props: {

className: 'greeting',

children: 'Hello, world'

}

};

**这些对象被称作“React元素”。**

你可以把他们想象成为你想在屏幕上显示内容的一种描述。React会读取这些对象，用他们来构建DOM，并且保持它们的不断更新。

## 三、元素渲染

**元素(Elements)是 React 应用中最小的构建部件（或者说构建块，building blocks）。**

### 渲染一个元素到DOM

我们假设你的 HTML 文件中的什么地方有这么一个<div>：

<div id="root"></div>

我们称这个是一个 “root” DOM 节点，因为该节点内的所有内容都由 React DOM 管理。

要渲染一个 React 元素到一个 root DOM 节点，把它们传递给 ReactDOM.render()方法：

const element = <h1>Hello, world</h1>;

ReactDOM.render(

element,

document.getElementById('root')

);

### 2、更新已渲染的元素

React 元素是 [不可突变（immutable）](https://en.wikipedia.org/wiki/Immutable_object) 的. 一旦你创建了一个元素, 就不能再修改其子元素或任何属性。一个元素就像电影里的一帧: 它表示在某一特定时间点的 UI 。

就我们所知, 更新 UI 的唯一方法是创建一个新的元素, 并将其传入 ReactDOM.render() 方法.

function tick() {

const element = (

<div>

<h1>Hello, world!</h1>

<h2>It is {new Date().toLocaleTimeString()}.</h2>

</div>

);

ReactDOM.render(

element,

document.getElementById('root')

);

}

setInterval(tick, 1000);

### 3、React只更新必需要更新的部分

React DOM 会将元素及其子元素与之前版本逐一对比, 并只对有必要更新的 DOM 进行更新, 以达到 DOM 所需的状态。

## 四、组件（Component）和属性（props）

组件使你可以将 UI 划分为一个一个独立，可复用的小部件，并可以对每个部件进行单独的设计。

从定义上来说， 组件就像JavaScript的函数。组件可以接收任意输入(称为”props”)， 并返回 React 元素，用以描述屏幕显示内容。

### 1、函数式组件和类组件

最简单的定义组件的方法是写一个 JavaScript 函数:

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

这个函数是一个有效的 React 组件，因为它接收一个 props 参数, 并返回一个 React 元素。 我们把此类组件称为”函数式(Functional)“组件， 因为从字面上看来它就是一个 JavaScript 函数。

你也可以用一个 [ES6 的 class](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Classes) 来定义一个组件:

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

### 2、渲染一个组件

DOM 标签的 React 元素：

const element = <div />;

元素也可以代表用户定义的组件：

const element = <Welcome name="Sara" />;

当 React 遇到一个代表用户定义组件的元素时，它将 JSX 属性以一个单独对象的形式传递给相应的组件。 我们将其称为 “props” 对象。

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

const element = <Welcome name="Sara" />;

ReactDOM.render(

element,

document.getElementById('root')

);

**警告：**

**组件名称总是以大写字母开始。**

**举例来说, <div /> 代表一个 DOM 标签，而 <Welcome /> 则代表一个组件，并且需要在作用域中有一个 Welcome 组件。**

### 3、构成组件

**组件可以在它们的输出中引用其它组件**。这使得我们可以使用同样的组件来抽象到任意层级。一个按钮，一个表单，一个对话框，一个屏幕：在 React 应用中，所有这些都通常描述为**组件**。

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

function App() {

return (

<div>

<Welcome name="Sara" />

<Welcome name="Cahal" />

<Welcome name="Edite" />

</div>

);

}

ReactDOM.render(

<App />,

document.getElementById('root')

);

**警告：**

**组件必须返回一个单独的根元素。这就是为什么我们添加一个 <div> 来包含所有 <Welcome /> 元素的原因。**

### 4、提取组件

**不要害怕把一个组件分为多个更小的组件。**

### 5、Props是只读的

**无论你用**[**函数或类**](http://www.css88.com/react/docs/components-and-props.html#functional-and-class-components)**的方法来声明组件, 它都无法修改其自身 props.**

**所有 React 组件都必须是纯函数，并禁止修改其自身 props 。**

纯函数：

function sum(a, b) {

return a + b;

}

非纯函数

function withdraw(account, amount) {

account.total -= amount;

}

## 五、状态（state）和生命周期

**state 和 props 类似，但是它是私有的，并且由组件本身完全控制。**

### 1、类组件专有特征：state

（**类组件**有专有特征：局部状态）

（**函数式组件**则不具备）

**类允许我们在其中添加本地状态(state)和生命周期钩子。**

class Clock extends React.Component {

/\*\*

\* 构造函数

\* @param {\*} props

\*/

constructor(props) {

super(props);

this.state = {date: new Date()};

}

/\*\*

\* 挂载

\*/

componentDidMount() {

// timeID直接存放于this中

// 注意：如果需要存储一些不用于视觉输出的内容，则可以手动向类中添加额外的字段

// 注意：如果在 render() 方法中没有被引用, 它不应该出现在 state 中

this.timerID = setInterval(

() => this.tick(),

1000

);

}

/\*\*

\* 卸载

\*/

componentWillUnmount() {

clearInterval(this.timerID);

}

/\*\*

\* 使用 this.setState() 来来周期性地更新组件本地状态

\*/

tick() {

this.setState({

date: new Date()

});

}

render() {

return (

<div>

<h1>Hello, World!</h1>

<h2>It is {this.state.date.toLocaleTimeString()}.</h2>

</div>

);

}

}

ReactDOM.render(

<Clock />,

document.getElementById('root')

);

**注意：**

**如果需要存储一些不用于视觉输出的内容，则可以手动向类中添加额外的字段，例如：this.name = ‘yang’;**

**如果在 render() 方法中没有被引用, 它不应该出现在 state 中。**

### 2、正确使用state

#### 1）不要直接修改state

// 错误

this.state.comment = 'Hello';

// 正确

this.setState({comment: 'Hello'});

唯一可以分配 this.state 的地方是构造函数。

constructor(props) {

super(props);

this.state = {date: new Date()};

}

#### 2）state更新可能是异步的

**React 为了优化性能，有可能会将多个 setState() 调用合并为一次更新。**

this.props 和 this.state 可能是异步更新的，所以不能依赖他们的值计算下一个state(状态)。

// 错误

this.setState({

counter: this.state.counter + this.props.increment,

});

要弥补这个问题，使用另一种 setState() 的形式，它接受一个函数而不是一个对象。这个函数将接收前一个状态作为第一个参数，应用更新时的 props 作为第二个参数：

// 正确

this.setState((prevState, props) => ({

counter: prevState.counter + props.increment

}));

#### 3）state更新会被合并

调用独立的setState()， React 将合并你提供的对象到当前的状态中。

constructor(props) {

super(props);

this.state = {

posts: [],

comments: []

};

}

componentDidMount() {

// 第一次调用，将更新this.state.posts

fetchPosts().then(response => {

this.setState({

posts: response.posts

});

});

// 第二次调用，将更新this.state.comments，不影响第一更新的this.state.posts

fetchComments().then(response => {

this.setState({

comments: response.comments

});

});

}

### 3、数据向下流

无论作为父组件还是子组件，**它都无法获悉一个组件是否有状态**，同时也**不需要关心另一个组件是定义为函数组件还是类组件**。

这就是 state(状态) 经常被称为 **本地状态** 或 **封装状态**的原因。 它不能被拥有并设置它的组件 以外的任何组件访问。

一个组件可以选择将 state(状态) 向下传递，作为其子组件的 props(属性)：

<h2>It is {this.state.date.toLocaleTimeString()}.</h2>

同样适用于用户定义组件:

<FormattedDate date={this.state.date} />

FormattedDate 组件通过 props(属性) 接收了 date 的值，但它仍然不能获知该值是来自于 Clock的 state(状态) ，还是 Clock 的 props(属性)，或者是直接手动创建的：

function FormattedDate(props) {

return <h2>It is {props.date.toLocaleTimeString()}.</h2>;

}

这通常称为一个**“从上到下”**，或者**“单向”**的数据流。任何 state(状态) 始终由某个特定组件所有，并且从该 state(状态) 导出的任何数据 或 UI **只能影响树中 “下方” 的组件**。

## 六、处理事件

### 1、React 事件使用驼峰命名，而不是全部小写。

// HTML

<button onclick="activateLasers()">

Activate Lasers

</button>

//React

<button onClick={activateLasers}>

Activate Lasers

</button>

### 2、通过 JSX , 传递一个函数作为事件处理程序，而不是一个字符串。

### 3、在 React 中不能通过返回 false来阻止默认行为，必须明确调用 preventDefault

// HTML

<a href="#" onclick="console.log('The link was clicked.'); return false">

Click me

</a>

//React

function ActionLink() {

function handleClick(e) {

e.preventDefault();

console.log('The link was clicked.');

}

return (

<a href="#" onClick={handleClick}>

Click me

</a>

);

}

### 4、关于绑定this（bind(this)）

**在JSX回调中必须注意 this 的指向。**

class Toggle extends React.Component {

constructor(props) {

super(props);

this.state = {isToggleOn: true};

// 这个绑定是必要的，使`this`在回调中起作用

this.handleClick = this.handleClick.bind(this);

}

handleClick() {

this.setState(prevState => ({

isToggleOn: !prevState.isToggleOn

}));

}

// 这个语法确保 `this` 绑定在 handleClick 中。

// 警告：这是 \*实验性的\* 语法。

// handleClick = () => {

// console.log('this is:', this);

// }

render() {

return (

<button onClick={this.handleClick}>

{this.state.isToggleOn ? 'ON' : 'OFF'}

</button>

);

// 这个语法确保 `this` 被绑定在 handleClick 中

// 这个语法问题是：每次Toggle渲染时都创建一个不同的回调，若回调被作为props传递给子组件时，子组件可能需要额外的重复渲染

// return (

// <button onClick={(e) => this.handleClick(e)}>

// {this.state.isToggleOn ? 'ON' : 'OFF'}

// </button>

// );

}

}

ReactDOM.render(

<Toggle />,

document.getElementById('root')

);

### 5、将参数传递给事件处理程序

在循环内部，通常需要将一个额外的参数传递给事件处理程序。 例如，如果 id 是一个内联 ID，则以下任一方式都可以正常工作：

// 参数 e 作为 React 事件对象将会被作为第二个参数进行传递

// 通过箭头函数的方式，事件对象必须显式的进行传递

<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>

// 通过 bind 的方式，事件对象以及更多的参数将会被隐式的进行传递

<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>

## 七、条件渲染

**在 React 中，可以创建不同的组件封装所需要的行为。然后，只渲染它们之中的一些，取决于应用的状态。**

function Greeting(props) {

const isLoggedIn = props.isLoggedIn;

if (isLoggedIn) {

return <UserGreeting />;

}

return <GuestGreeting />;

}

ReactDOM.render(

// 修改为 isLoggedIn={true} 试试:

<Greeting isLoggedIn={false} />,

document.getElementById('root')

);

### 1、元素变量

可以用变量来存储元素。有条件地渲染组件的一部分，而输出的其余部分不会更改。

### 2、使用逻辑 && 操作符的内联 if 用法

可以在JSX语法嵌入任何表达式，方法是将其包裹在花括号中。

function Mailbox(props) {

const unreadMessages = props.unreadMessages;

return (

<div>

<h1>Hello!</h1>

{unreadMessages.length > 0 &&

<h2>

You have {unreadMessages.length} unread messages.

</h2>

}

</div>

);

}

const messages = ['React', 'Re: React', 'Re:Re: React'];

ReactDOM.render(

<Mailbox unreadMessages={messages} />,

document.getElementById('root')

);

### 3、防止组件渲染

class Page extends React.Component {

constructor(props) {

super(props);

this.state = {showWarning: true}

this.handleToggleClick = this.handleToggleClick.bind(this);

}

handleToggleClick() {

this.setState(prevState => ({

showWarning: !prevState.showWarning

}));

}

render() {

return (

<div>

<WarningBanner warn={this.state.showWarning} />

<button onClick={this.handleToggleClick}>

{this.state.showWarning ? 'Hide' : 'Show'}

</button>

</div>

);

}

}

ReactDOM.render(

<Page />,

document.getElementById('root')

);

从组件的 render 方法返回 null 不会影响组件生命周期方法的触发。例如， componentWillUpdate 和 componentDidUpdate 仍将被调用。

## 八、列表Lists和键Keys

### 1、多组件渲染

const numbers = [1, 2, 3, 4, 5];

const listItems = numbers.map((number) =>

<li>{number}</li>

);

// 把整个 listItems 数组包含到一个 <ul> 元素，并渲染到 DOM：

ReactDOM.render(

<ul>{listItems}</ul>,

document.getElementById('root')

);

### 2、基本列表组件

function NumberList(props) {

const numbers = props.numbers;

const listItems = numbers.map((number) =>

<li key={number.toString()}>

{number}

</li>

);

return (

<ul>{listItems}</ul>

);

}

const numbers = [1, 2, 3, 4, 5];

ReactDOM.render(

<NumberList numbers={numbers} />,

document.getElementById('root')

);

### 3、键Keys

**键(Keys) 帮助 React 标识哪个项被修改、添加或者移除了。**数组中的每一个元素都应该有一个唯一不变的键(Keys)来标识：

const numbers = [1, 2, 3, 4, 5];

const listItems = numbers.map((number, index) =>

<li key={number.toString()}>

{number}

</li>

);

keys 只在数组的上下文中存在意义。

### 4、keys 在同辈元素中必须是唯一的

**在数组中使用的 keys 必须在它们的同辈之间唯一。然而它们并不需要全局唯一。**

键是React的一个内部映射，但其不会传递给组件的内部。如果你需要在组件中使用相同的值，可以明确使用一个不同名字的 prop 传入。

const content = posts.map((post) =>

<Post

key={post.id}

id={post.id}

title={post.title} />

);

## 九、表单Forms

### 1、受控组件：Controlled Components

在 HTML 中，表单元素如 <input>，<textarea> 和 <select> 表单元素通常保持自己的状态，并根据用户输入进行更新。而在 React 中，可变状态一般保存在组件的 state(状态) 属性中，并且只能通过 [setState()](http://www.css88.com/react/docs/react-component.html" \l "setstate) 更新。

我们可以通过使 React 的 state 成为 “单一数据源原则” 来结合这两个形式。然后渲染表单的 React 组件也可以控制在用户输入之后的行为。这种形式，其值由 React 控制的输入表单元素称为“**受控组件**”。

class NameForm extends React.Component {

constructor(props) {

super(props);

this.state = {value: ''};

this.handleChange = this.handleChange.bind(this);

this.handleSubmit = this.handleSubmit.bind(this);

}

handleChange(event) {

this.setState({value: event.target.value});

}

handleSubmit(event) {

alert('A name was submitted: ' + this.state.value);

event.preventDefault();

}

render() {

return (

<form onSubmit={this.handleSubmit}>

<label>

Name:

<input type="text" value={this.state.value} onChange={this.handleChange} />

</label>

<input type="submit" value="Submit" />

</form>

);

}

}

### 2、textare标签

// HTML

<textarea>

Hello there, this is some text in a text area

</textarea>

// 在 React 中，<textarea> 的赋值使用 value 属性替代。

// 这样一来，表单中 <textarea> 的书写方式接近于单行文本输入框 ：

// 注意，this.state.value 在构造函数中初始化，所以这些文本一开始就出现在文本域中。

class EssayForm extends React.Component {

constructor(props) {

super(props);

this.state = {

value: 'Please write an essay about your favorite DOM element.'

};

this.handleChange = this.handleChange.bind(this);

this.handleSubmit = this.handleSubmit.bind(this);

}

handleChange(event) {

this.setState({value: event.target.value});

}

handleSubmit(event) {

alert('An essay was submitted: ' + this.state.value);

event.preventDefault();

}

render() {

return (

<form onSubmit={this.handleSubmit}>

<label>

Name:

<textarea value={this.state.value} onChange={this.handleChange} />

</label>

<input type="submit" value="Submit" />

</form>

);

}

}

### 3、select标签

// HTML

<select>

<option value="grapefruit">Grapefruit</option>

<option value="lime">Lime</option>

<option selected value="coconut">Coconut</option>

<option value="mango">Mango</option>

</select>

// React

class FlavorForm extends React.Component {

constructor(props) {

super(props);

this.state = {value: 'coconut'};

this.handleChange = this.handleChange.bind(this);

this.handleSubmit = this.handleSubmit.bind(this);

}

handleChange(event) {

this.setState({value: event.target.value});

}

handleSubmit(event) {

alert('Your favorite flavor is: ' + this.state.value);

event.preventDefault();

}

render() {

return (

<form onSubmit={this.handleSubmit}>

<label>

Pick your favorite La Croix flavor:

<select value={this.state.value} onChange={this.handleChange}>

<option value="grapefruit">Grapefruit</option>

<option value="lime">Lime</option>

<option value="coconut">Coconut</option>

<option value="mango">Mango</option>

</select>

</label>

<input type="submit" value="Submit" />

</form>

);

}

}

**注意**

**您可以将一个数组传递给 value 属性，允许你在 select 标签中选择多个选项：**

<select multiple={true} value={['B', 'C']}>

### 4、file input 标签

在 React 中，一个 <input type =“file”/> 和一个普通的 <input /> 类似，但有一个重要的区别：**它是只读的(read-only)**。

class FileInput extends React.Component {

constructor(props) {

super(props);

this.handleSubmit = this.handleSubmit.bind(

this

);

}

handleSubmit(event) {

event.preventDefault();

alert(

`Selected file - ${

this.fileInput.files[0].name

}`

);

}

render() {

return (

<form

onSubmit={this.handleSubmit}>

<label>

Upload file:

<input

type="file"

ref={input => {

this.fileInput = input;

}}

/>

</label>

<br />

<button type="submit">

Submit

</button>

</form>

);

}

}

ReactDOM.render(

<FileInput />,

document.getElementById('root')

);

### 5、处理多个输入元素

需要处理多个受控的 input 元素时，可以为每个元素添加一个 name 属性，并且让处理函数根据 event.target.name 的值来选择要做什么。

class Reservation extends React.Component {

constructor(props) {

super(props);

this.state = {

isGoing: true,

numberOfGuests: 2

};

this.handleInputChange = this.handleInputChange.bind(this);

}

handleInputChange(event) {

const target = event.target;

const value = target.type === 'checkbox' ? target.checked : target.value;

const name = target.name;

// 使用ES6计算的属性名称语法来更新与给定输入名称相对应的 state(状态) 键：

this.setState({

[name]: value

});

}

render() {

return (

<form>

<label>

Is going:

<input

name="isGoing"

type="checkbox"

checked={this.state.isGoing}

onChange={this.handleInputChange} />

</label>

<br />

<label>

Number of guests:

<input

name="numberOfGuests"

type="number"

value={this.state.numberOfGuests}

onChange={this.handleInputChange} />

</label>

</form>

);

}

}

## 十、状态提升：Lifting State Up