[ReactDOM.render() 2](#_Toc17551)

[JSX 语法 2](#_Toc8807)

[组件 2](#_Toc8091)

[React.createClass 2](#_Toc28151)

[this.props.children 2](#_Toc21095)

[PropTypes 2](#_Toc4670)

[获取真实的DOM节点 3](#_Toc14389)

[state 3](#_Toc20235)

[表单 4](#_Toc1226)

[Ajax 4](#_Toc30586)

[@connect 装饰器 4](#_Toc7059)

[Redux-saga 4](#_Toc21245)

[JSX 简介 4](#_Toc14603)

[在 JSX 中使用表达式 5](#_Toc15280)

[JSX 属性 5](#_Toc18499)

[JSX 代表 Objects 5](#_Toc3128)

[元素渲染 5](#_Toc19897)

[将元素渲染到 DOM 中 6](#_Toc16837)

[更新元素渲染 6](#_Toc23014)

[组件 & Props 6](#_Toc6889)

[函数定义/类定义组件 6](#_Toc18721)

[组件渲染 6](#_Toc609)

[Props的只读性 6](#_Toc981)

[State & 生命周期 7](#_Toc19726)

[为一个类添加局部状态 7](#_Toc18708)

[正确地使用状态 7](#_Toc19617)

[事件处理 7](#_Toc7947)

[向事件处理程序传递参数 8](#_Toc1231)

[条件渲染 8](#_Toc8171)

[阻止组件渲染 8](#_Toc32524)

[列表 & Keys 8](#_Toc20287)

[Keys 9](#_Toc22802)

[用keys提取组件 9](#_Toc17095)

[表单 9](#_Toc19657)

[受控组件 9](#_Toc848)

[textarea 标签 9](#_Toc9290)

[select 标签 9](#_Toc27934)

[多个输入的解决方法 10](#_Toc25359)

[状态提升 10](#_Toc21734)

[组合 vs 继承 10](#_Toc30095)

[React理念 10](#_Toc7539)

# ReactDOM.render()

将模板转为 HTML 语言，并插入指定的 DOM 节点。

ReactDOM.render(

<h1>Hello, world!</h1>,

document.getElementById('example') //插入到id=example中

);

# JSX 语法

允许 HTML 与 JavaScript 的混写。

遇到 HTML 标签（以 < 开头），就用 HTML 规则解析；遇到代码块（以 { 开头），就用 JavaScript 规则解析。

JSX 允许直接在模板插入 JavaScript 变量。如果这个变量是一个数组，则会展开这个数组的所有成员

# 组件

React 允许将代码封装成组件（component），然后像插入普通 HTML 标签一样，在网页中插入这个组件。

# React.createClass

生成一个组件类。使用标签时生成组件类实例。

所有组件类都必须有render 方法，用于输出组件。

组件类的首字母必须大写，否则会报错。

组件类只能包含一个顶层标签(顶层标签不能有平行标签)，否则会报错。

组件的用法与原生的 HTML 标签完全一致，可以任意加入属性(使用时传值)；组件的属性在组件类的 this.props 对象上获取。

注意：保留字class 属性需要写成 className ，保留字for 属性需要写成 htmlFor 。

# this.props.children

表示组件的所有子节点。

如果当前组件没有子节点，它就是 undefined ;

如果有一个子节点，数据类型是 object ；

如果有多个子节点，数据类型就是 array

React.Children.map(this.props.children, function (child){ } ) 来遍历子节点。

# PropTypes

因组件的属性可以接受任意值；使用PropTypes属性验证组件实例的属性是否符合要求。

var MyTitle = React.createClass({

propTypes: {

title: React.PropTypes.string.isRequired,

},

render: function() {

return <h1> {this.props.title} </h1>;

}

});

getDefaultProps 方法可以用来设置组件属性的默认值。

var MyTitle = React.createClass({

getDefaultProps : function () {

return {

title : 'Hello World'

};

},

render: function() {

return <h1> {this.props.title} </h1>;

}

});

# 获取真实的DOM节点

ref 属性

var MyComponent = React.createClass({

handleClick: function() {

this.refs.myTextInput.focus();

},

render: function() {

return (

<div>

<input type="text" ref="myTextInput" />

<input type="button" value="Focus the text input" onClick={this.handleClick} />

</div>

);

}

});

# state

状态；getInitialState，setState；通过handleClick等调用setState改变state。

# 表单

通过this.state.value获取。通过onChang调用handleChange实现。

# Ajax

组件的数据来源，通常是通过 Ajax 请求从服务器获取，可以使用 componentDidMount 方法设置 Ajax 请求，等到请求成功，再用this.setState 方法重新渲染 UI。

# @connect 装饰器

@connect(({ chart, loading }) => ({

chart,

loading: loading.effects['chart/fetch'],

}))

@connect，dva 所封装的 react-redux 的 @connect 装饰器

list，绑定到此model的redux store。app.state.list。

loading，app.state.loading，来源是 src/index.js 中调用的 dva-loading；app.use(createLoading());。

payload 向 store 进行了一个类型为 chart/fetch 的 dispatch；src/models/chart.js 中就可以找到具体的对应操作。

# Redux-saga

是Redux的一个中间件，主要集中处理react架构中的异步处理工作，被定义为generator(ES6)的形式，采用监听的形式进行工作。

Effect 是一个 javascript 对象，可以通过 yield 传达给 sagaMiddleware 进行执行在， 如果我们应用redux-saga，所有的 Effect 都必须被 yield 才会执行。

put，触发某个action， 作用和dispatch相同：

yield put({ type: 'CLICK' });

call，有阻塞地调用 saga 或者返回 promise 的函数，只在触发某个动作。

状态与属性十分相似，但是状态是私有的，完全受控于当前组件。

使用类就允许我们使用其它特性，例如局部状态、生命周期钩子。

# JSX 简介

JSX 用来声明 React 当中的元素。

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

因为 JSX 的特性更接近 JavaScript 而不是 HTML , 所以 React DOM 使用 camelCase 小驼峰命名 来定义属性的名称，而不是使用 HTML 的属性名称。

例如，class 变成了 className，而 tabindex 则对应着 tabIndex。

## 在 JSX 中使用表达式

可以任意地在 JSX 当中使用 JavaScript 表达式，在 JSX 当中的表达式要包含在大括号里。

推荐在 JSX 代码的外面扩上小括号，这样可以防止 分号自动插入 的 bug。

## JSX 属性

你可以使用引号来定义以字符串为值的属性：

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

也可以使用大括号来定义以 JavaScript 表达式为值的属性：

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

## JSX 代表 Objects

Babel 转译器会把 JSX 转换成一个名为 React.createElement() 的方法调用。

const element = (

<h1 className="greeting">

Hello, world!

</h1>

);

// 注意: 以下示例是简化过的（不代表在 React 源码中是这样）

const element = {

type: 'h1',

props: {

className: 'greeting',

children: 'Hello, world'

}

};

# 元素渲染

元素是构成 React 应用的最小单位。

元素用来描述你在屏幕上看到的内容：

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

与浏览器的 DOM 元素不同，React 当中的元素事实上是普通的对象，React DOM 可以确保 浏览器 DOM 的数据内容与 React 元素保持一致。

## 将元素渲染到 DOM 中

ReactDOM.render()

## 更新元素渲染

React 只会更新必要的部分

React DOM 首先会比较元素内容先后的不同，而在渲染过程中只会更新改变了的部分。

# 组件 & Props

组件从概念上看就像是函数，它可以接收任意的输入值（称之为“props”）。

## 函数定义/类定义组件

使用JavaScript函数

function Welcome(props) {

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

}

使用 ES6 class

class Welcome extends React.Component {

render() {

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

}

}

## 组件渲染

当React遇到的元素是用户自定义的组件，它会将JSX属性作为单个对象传递给该组件，这个对象称之为“props”。

组件的返回值只能有一个根元素。

组件名称必须以大写字母开头。

建议从组件自身的角度来命名props，而不是根据使用组件的上下文命名。

## Props的只读性

无论是使用函数或是类来声明一个组件，它决不能修改它自己的props。

所有的React组件必须像纯函数那样使用它们的props。即不能修改输入值props。

# State & 生命周期

状态与属性十分相似，但是状态是私有的，完全受控于当前组件。

定义为类的组件特性：局部状态，、生命周期钩子。

## 为一个类添加局部状态

添加一个类构造函数来初始化状态 this.state。类组件应始终使用props调用基础构造函数。

在生命周期函数中调用setState()更新this.state，并调度UI更新。

虽然 this.props 由React本身设置以及this.state 具有特殊的含义，但如果需要存储不用于视觉输出的东西，则可以手动向类中添加其他字段。如果你不在 render() 中使用某些东西，它就不应该在状态中。

## 正确地使用状态

不要直接更新状态，构造函数是唯一能够初始化 this.state 的地方。

this.state.comment = 'Hello'; // Wrong，不会重新渲染组件

状态更新可能是异步的

this.props 和 this.state 可能是异步更新的，你不应该依靠它们的值来计算下一个状态。

使用第二种形式的 setState() 来接受一个函数而不是一个对象。 该函数将接收先前的状态作为第一个参数，将此次更新被应用时的props做为第二个参数

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

counter: prevState.counter + props.increment

}));

状态更新合并

当你调用 setState() 时，React 将你提供的对象合并到当前状态。

# 事件处理

React事件绑定属性的命名采用驼峰式写法

如果采用 JSX 的语法你需要传入一个函数作为事件处理函数

<button onClick={activateLasers}>

Activate Lasers

</button>

不能使用返回 false 的方式阻止默认行为；必须明确的使用 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>

);

}

## 向事件处理程序传递参数

通过 arrow functions为事件处理函数传递参数。

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

通过 Function.prototype.bind 为事件处理函数传递参数。

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

参数 e 作为 React 事件对象将会被作为第二个参数进行传递。箭头函数方式，事件对象必须显式的进行传递，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。

通过 bind 方式向监听函数传参，在类组件中定义的监听函数，事件对象 e 要放在最后。

# 条件渲染

使用 JavaScript 操作符 if 或条件运算符来创建表示当前状态的元素，然后让 React 根据它们来更新 UI。

if，&&，三木运算符？。

## 阻止组件渲染

让 render 方法返回 null 而不是它的渲染结果即可实现。

组件的 render 方法返回 null 并不会影响该组件生命周期方法的回调。例如，componentWillUpdate 和 componentDidUpdate 依然可以被调用。

# 列表 & Keys

通过map()方法，将数组、Iterator接口的对象映射为列表。

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

const doubled = numbers.map((number) => number \* 2);

console.log(doubled);

## Keys

Keys可以在DOM中的某些元素被增加或删除的时候帮助React识别哪些元素发生了变化。

key最好是这个元素在列表中拥有的一个独一无二的字符串。通常使用数据的id作为元素的key。

### 用keys提取组件

元素的key只有在它和它的兄弟节点对比时才有意义。

提取出一个ListItem组件，你应该把key保存在数组中的这个<ListItem />元素上，而不是放在<li>元素上。

在map()方法的内部调用元素时，记得为每一个元素加上一个独一无二的key。

元素的key在他的兄弟元素之间应该唯一，不需要是全局唯一。

key会作为给React的提示，但不会传递给你的组件。如需要使用和key相同的值，将其作为属性传递

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

<Post

key={post.id}

id={post.id}

title={post.title} />

);

# 表单

## 受控组件

在React中，可变的状态通常保存在组件的状态属性中，并且只能用 setState() 方法进行更新。

通过绑定状态改变事件函数，使用e.target.value传递值；调用setState()重置state数据，刷新显示。

## textarea 标签

在HTML当中，<textarea> 元素通过子节点来定义它的文本内容。

在React中，<textarea>会用value属性来代替。处理方法同上<input type="text"/>。

## select 标签

HTML在子标签使用selected表示选中。

在React中，并不使用之前的selected属性，而在根select标签上用value属性来表示选中项。

## 多个输入的解决方法

当你有处理多个受控的input元素时，你可以通过给每个元素添加一个name属性，来让处理函数根据 event.target.name的值来选择做什么。

使用ES6当中的计算属性名语法来更新与给定输入名称相对应的状态键：

this.setState({

[name]: value //计算name的值，转换为属性名

});

# 状态提升

在React中，状态分享是通过将state数据提升至离需要这些数据的组件最近的父组件来完成的。这就是所谓的状态提升。

调用父组件的setState()方法更新state，并将更新后的state赋给子组件，刷新显示。

# 组合 vs 继承

React 具有强大的组合模型，我们建议使用组合而不是继承来复用组件之间的代码。

如果要在组件之间复用 UI 无关的功能，我们建议将其提取到单独的 JavaScript 模块中。这样可以在不对组件进行扩展的前提下导入并使用该函数、对象或类。

# React理念

props 是一种从父级向子级传递数据的方法

在创建静态版本的时候不要使用 state。State 只在交互的时候使用，即随时间变化的数据。

确定state：通过props从父级传来？随时间推移(用户点击、输入等操作)不变？能否根据state/props算出？