[ReactDOM.render() 2](#_Toc11165)

[JSX 语法 2](#_Toc1525)

[组件 2](#_Toc26866)

[React.createClass 2](#_Toc15194)

[this.props.children 2](#_Toc7605)

[PropTypes 2](#_Toc18141)

[获取真实的DOM节点 3](#_Toc1003)

[state 3](#_Toc26528)

[表单 4](#_Toc24218)

[Ajax 4](#_Toc31567)

[@connect 装饰器 4](#_Toc10357)

# 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。