React State(状态) React 组件 API

React Props

state 和 props 主要的区别在于 **props** 是不可变的，而 state 可以根据与用户交互来改变。这就是为什么有些容器组件需要定义 state 来更新和修改数据。 而子组件只能通过 props 来传递数据。

使用 Props

以下实例演示了如何在组件中使用 props：

React 实例

```
function HelloMessage(props) {
return <h1>Hello {props.name}!</h1>;
}
const element = <HelloMessage name="Runoob"/>;
ReactDOM.render(
element,
document.getElementById('example')
);
```

尝试一下 »

实例中 name 属性通过 props.name 来获取。

默认 Props

你可以通过组件类的 defaultProps 属性为 props 设置默认值，实例如下：

React 实例

```
class HelloMessage extends React.Component {
render() {
return (
<h1>Hello, {this.props.name}</h1>
);
}
}
HelloMessage.defaultProps = {
name: 'Runoob'
};
const element = <HelloMessage/>;
ReactDOM.render(
element,
document.getElementById('example')
);
```

尝试一下 »

State 和 Props

以下实例演示了如何在应用中组合使用 state 和 props 。我们可以在父组件中设置 state， 并通过在子组件上使用 props 将其传递到子组件上。在 render 函数中, 我们设置 name 和 site 来获取父组件传递过来的数据。

React 实例

```
class WebSite extends React.Component {
constructor() {
super();
this.state = {
name: "菜鸟教程",
site: "https://www.runoob.com"
}
}
render() {
return (
<div>
<Name name={this.state.name} />
<Link site={this.state.site} />
</div>
);
}
}
class Name extends React.Component {
render() {
return (
<h1>{this.props.name}</h1>
);
}
}
class Link extends React.Component {
render() {
return (
<a href={this.props.site}>
{this.props.site}
</a>
);
}
}
ReactDOM.render(
<WebSite />,
document.getElementById('example')
);
```

尝试一下 »

Props 验证

React.PropTypes 在 React v15.5 版本后已经移到了 prop-types 库。

```
<script src="https://cdn.bootcss.com/prop-types/15.6.1/prop-types.js"></script>
```

Props 验证使用 **propTypes**，它可以保证我们的应用组件被正确使用，React.PropTypes 提供很多验证器 (validator) 来验证传入数据是否有效。当向 props 传入无效数据时，JavaScript 控制台会抛出警告。

以下实例创建一个 Mytitle 组件，属性 title 是必须的且是字符串，非字符串类型会自动转换为字符串 ：

React 16.4 实例

```
var title = "菜鸟教程";
// var title = 123;
class MyTitle extends React.Component {
render() {
return (
<h1>Hello, {this.props.title}</h1>
);
}
}
MyTitle.propTypes = {
title: PropTypes.string
};
ReactDOM.render(
<MyTitle title={title} />,
document.getElementById('example')
);
```

尝试一下 »

React 15.4 实例

```
var title = "菜鸟教程";
// var title = 123;
var MyTitle = React.createClass({
propTypes: {
title: React.PropTypes.string.isRequired,
},
render: function() {
return <h1> {this.props.title} </h1>;
}
});
ReactDOM.render(
<MyTitle title={title} />,
document.getElementById('example')
);
```

尝试一下 »

更多验证器说明如下：

```
MyComponent.propTypes = {
// 可以声明 prop 为指定的 JS 基本数据类型，默认情况，这些数据是可选的
optionalArray: React.PropTypes.array,
optionalBool: React.PropTypes.bool,
optionalFunc: React.PropTypes.func,
```

```
optionalNumber: React.PropTypes.number,
optionalObject: React.PropTypes.object,
optionalString: React.PropTypes.string,
// 可以被渲染的对象 numbers, strings, elements 或 array
optionalNode: React.PropTypes.node,
// React 元素
optionalElement: React.PropTypes.element,
// 用 JS 的 instanceof 操作符声明 prop 为类的实例。
optionalMessage: React.PropTypes.instanceOf(Message),
// 用 enum 来限制 prop 只接受指定的值。
optionalEnum: React.PropTypes.oneOf(['News', 'Photos']),
// 可以是多个对象类型中的一个
optionalUnion: React.PropTypes.oneOfType([
React.PropTypes.string,
React.PropTypes.number,
React.PropTypes.instanceOf(Message)
]),
// 指定类型组成的数组
optionalArrayOf: React.PropTypes.arrayOf(React.PropTypes.number),
// 指定类型的属性构成的对象
optionalObjectOf: React.PropTypes.objectOf(React.PropTypes.number),
// 特定 shape 参数的对象
optionalObjectWithShape: React.PropTypes.shape({
color: React.PropTypes.string,
fontSize: React.PropTypes.number
}),
// 任意类型加上 `isRequired` 来使 prop 不可空。
requiredFunc: React.PropTypes.func.isRequired,
// 不可空的任意类型
requiredAny: React.PropTypes.any.isRequired,
// 自定义验证器。如果验证失败需要返回一个 Error 对象。不要直接使用 `console.warn` 或抛异常，因为这样 `oneOfType` 会失效。
customProp: function(props, propName, componentName) {
if (!/matchme/.test(props[propName])) {
return new Error('Validation failed!');
}
}
}
}
```

React State(状态) React 组件 API