# 组件&props

一个组件的显示形态可以由数据状态和外部参数所决定，外部参数也就是props，而数据状态就是state。

1. props（外部传入参数）

props是一个从外部传进组件的参数，主要作为就是从父组件向子组件传递数据，它具有可读性和不变性，只能通过外部组件主动传入新的props来重新渲染子组件，否则子组件的props以及展现形式不会改变。

组件无论是使用[函数声明还是通过 class 声明](https://zh-hans.reactjs.org/docs/components-and-props.html#function-and-class-components)，都决不能修改自身的 props。

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')

);

1. state（当前组件私有的，内部使用）

一个组件的显示形态可以由数据状态和外部参数所决定，外部参数也就是props，而数据状态就是state。

用法

export default class ItemList extends React.Component{

constructor(){

super();

this.state = {

itemList:'一些数据',

}

}

render(){

return (

{this.state.itemList}

)

}

}

首先，在组件初始化的时候，通过this.state给组件设定一个初始的state，在第一次render的时候就会用这个数据来渲染组件。

state不同于props的一点是，state是可以被改变的。不过，不可以直接通过this.state=的方式来修改，而需要通过this.setState()方法来修改state。

**注意：通过this.state=来初始化state，使用this.setState来修改state，constructor是唯一能够初始化的地方。**

1. setState

this.setState()方法来修改state。

export default class ItemList extends React.Component{

constructor(){

super();

this.state = {

name:'axuebin',

age:25,

}

}

componentDidMount(){

this.setState({age:18})

}

}

setState还可以接受第二个参数，它是一个函数，会在setState调用完成并且组件开始重新渲染时被调用，可以用来监听渲染是否完成：

this.setState({

name:'xb'

},()=>console.log('setState finished'))

在执行完setState之后的state应该是{name:'axuebin',age:18}。

**总结**

state的主要作用是用于组件保存、控制以及修改自己的状态，它只能在constructor中初始化，它算是组件的私有属性，不可通过外部访问和修改，只能通过组件内部的this.setState来修改，修改state属性会导致组件的重新渲染.

1. 组件

函数组件与 class 组件

**注意： 组件名称必须以大写字母开头。**

React 会将以小写字母开头的组件视为原生 DOM 标签。例如，<div /> 代表 HTML 的 div 标签，而 <Welcome /> 则代表一个组件，并且需在作用域内使用 Welcome。

你可以在[深入 JSX](https://zh-hans.reactjs.org/docs/jsx-in-depth.html" \l "user-defined-components-must-be-capitalized)中了解更多关于此规范的原因。

**State 的更新可能是异步的**

出于性能考虑，React 可能会把多个 setState() 调用合并成一个调用。

因为 this.props 和 this.state 可能会异步更新，所以你不要依赖他们的值来更新下一个状态。

1. componentWillMount 和 componentDidMount的区别

1、componentWillMount  将要装载，在render之前调用；

     componentDidMount，（装载完成），在render之后调用

2、componentWillMount  每一个组件render之前立即调用；

    componentDidMount  render之后并不会立即调用，而是所有的子组件都render完之后才可以调用

3、componentWillMount  可以在服务端被调用，也可以在浏览器端被调用；

    componentDidMount  只能在浏览器端被调用，在服务器端使用react的时候不会被调用

# 事件处理

在 React 中另一个不同点是你不能通过返回 false 的方式阻止默认行为。你必须显式的使用 preventDefault 。例如，传统的 HTML 中阻止链接默认打开一个新页面，你可以这样写：

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

Click me

</a>

在 React 中，可能是这样的：

function ActionLink() {

function handleClick(e) {

e.preventDefault();

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

}

return (

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

Click me

</a>

);

}

# 条件渲染

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')

);

之所以能这样做，是因为在 JavaScript 中，true && expression 总是会返回 expression, 而 false && expression 总是会返回 false。

因此，如果条件是 true，&& 右侧的元素就会被渲染，如果是 false，React 会忽略并跳过它。



1. 三目运算

另一种内联条件渲染的方法是使用 JavaScript 中的三目运算符 [condition ? true : false](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Conditional_Operator)。

render() {

const isLoggedIn = this.state.isLoggedIn;

return (

<div>

{isLoggedIn ? (

<LogoutButton onClick={this.handleLogoutClick} />

) : (

<LoginButton onClick={this.handleLoginClick} />

)}

</div>

);

}

1. 阻止组件渲染

在极少数情况下，你可能希望能隐藏组件，即使它已经被其他组件渲染。若要完成此操作，你可以让 render 方法直接返回 null，而不进行任何渲染。

下面的示例中，<WarningBanner /> 会根据 prop 中 warn 的值来进行条件渲染。如果 warn的值是 false，那么组件则不会渲染:

function WarningBanner(props) {

if (!props.warn) {

return null;

}

return (

<div className="warning">

Warning!

</div>

);

}

class Page extends React.Component {

constructor(props) {

super(props);

this.state = {showWarning: true};

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

}

handleToggleClick() {

this.setState(state => ({

showWarning: !state.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')

);

# 列表&key

1. 列表

下面，我们使用 Javascript 中的 [map()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map) 方法来遍历 numbers 数组。将数组中的每个元素变成 <li> 标签，最后我们将得到的数组赋值给 listItems：

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

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

<li>{numbers}</li>

);

ReactDOM.render(

<ul>{listItems}</ul>,

document.getElementById('root')

);

这段代码生成了一个 1 到 5 的项目符号列表。

1. key

key 帮助 React 识别哪些元素改变了，比如被添加或删除。因此你应当给数组中的每一个元素赋予一个确定的标识。

一个好的经验法则是：在 map() 方法中的元素需要设置 key 属性。