React 元素渲染 React 条件渲染

React 事件处理

React 元素的事件处理和 DOM 元素类似。但是有一点语法上的不同:

- React 事件绑定属性的命名采用驼峰式写法，而不是小写。
- 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，而不是一个字符串(DOM 元素的写法)

HTML 通常写法是：

```
<button onclick="activateLasers()">
  激活按钮
</button>
```

React 中写法为：

```
<button onClick={activateLasers}>
  激活按钮
</button>
```

在 React 中另一个不同是你不能使用返回 **false** 的方式阻止默认行为， 你必须明确的使用 `preventDefault`。

例如，通常我们在 HTML 中阻止链接默认打开一个新页面，可以这样写：

```
<a href="#" onclick="console.log('点击链接'); return false">
  点我
</a>
```

在 React 的写法为：

```
function ActionLink() {
function handleClick(e) {
e.preventDefault();
console.log('链接被点击');
}
return (
<a href="#" onClick={handleClick}>
点我
</a>
);
}
```

实例中 e 是一个合成事件。

使用 React 的时候通常你不需要使用 addEventListener 为一个已创建的 DOM 元素添加监听器。你仅仅需要在这个元素初始渲染的时候提供一个监听器。

当你使用 ES6 class 语法来定义一个组件的时候，事件处理器会成为类的一个方法。例如，下面的 Toggle 组件渲染一个让用户切换开关状态的按钮：

实例

```
class Toggle extends React.Component {
constructor(props) {
super(props);
this.state = {isToggleOn: true};
// 这边绑定是必要的，这样 `this` 才能在回调函数中使用
this.handleClick = this.handleClick.bind(this);
}
handleClick() {
this.setState(prevState => ({
isToggleOn: !prevState.isToggleOn
}));
}
render() {
return (
<button onClick={this.handleClick}>
{this.state.isToggleOn ? 'ON' : 'OFF'}
</button>
);
}
}
ReactDOM.render(
<Toggle />,
document.getElementById('example')
);
```

尝试一下 »

你必须谨慎对待 JSX 回调函数中的 this，类的方法默认是不会绑定 this 的。如果你忘记绑定 this.handleClick 并把它传入 onClick, 当你调用这个函数的时候 this 的值会是 undefined。

这并不是 React 的特殊行为；它是函数如何在 JavaScript 中运行的一部分。通常情况下，如果你没有在方法后面添加 () ，例如 `onClick={this.handleClick}` ，你应该为这个方法绑定 this。

如果使用 bind 让你很烦，这里有两种方式可以解决。如果你正在使用实验性的属性初始化器语法，你可以使用属性初始化器来正确的绑定回调函数：

```
class LoggingButton extends React.Component {
// 这个语法确保了 `this` 绑定在 handleClick 中
// 这里只是一个测试
handleClick = () => {
console.log('this is:', this);
}
render() {
return (
<button onClick={this.handleClick}>
Click me
</button>
);
```

```
}
}
```

如果你没有使用属性初始化器语法，你可以在回调函数中使用 箭头函数：

```
class LoggingButton extends React.Component {
handleClick() {
console.log('this is:', this);
}
render() {
// 这个语法确保了 `this` 绑定在 handleClick 中
return (
<button onClick={(e) => this.handleClick(e)}>
Click me
</button>
);
}
}
```

使用这个语法有个问题就是每次 LoggingButton 渲染的时候都会创建一个不同的回调函数。在大多数情况下，这没有问题。然而如果这个回调函数作为一个属性值传入低阶组件，这些组件可能会进行额外的重新渲染。我们通常建议在构造函数中绑定或使用属性初始化器语法来避免这类性能问题。

向事件处理程序传递参数

通常我们会为事件处理程序传递额外的参数。例如，若是 id 是你要删除那一行的 id，以下两种方式都可以向事件处理程序传递参数：

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

上述两种方式是等价的。
上面两个例子中，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，但是通过 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。
值得注意的是，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 要排在所传递参数的后面，例如:

```
class Popper extends React.Component{
constructor(){
super();
this.state = {name:'Hello world!'};
}
preventPop(name, e){ //事件对象e要放在最后
e.preventDefault();
alert(name);
}
render(){
return (
<div>
<p>hello</p>
{/* 通过 bind() 方法传递参数。 */}
```

```
<a href="https://reactjs.org" onClick={this.preventPop.bind(this,this.state.name)}>Click</a>
</div>
);
}
}
```

React 元素渲染 React 条件渲染

2 篇笔记

写笔记

React 点击事件的 bind(this) 如何传参?

需要通过 bind 方法来绑定参数，第一个参数指向 this,第二个参数开始才是事件函数接收到的参数:

```
<button onClick={this.handleClick.bind(this, props0, props1, ...}></button>

handleClick(porps0, props1, ..., event) {
    // your code here
}
```

事件：`this.handleclick.bind(this，要传的参数)`

函数：`handleclick(传过来的参数，event)`

从 **render** 中传递参数到 外部函数 `one()`:

```
class Ex extends React.Component{

....  this.state={name:'Stupid Dog'};

... ... ...

function one(para){
    console.log('parameter in one Func',para);
}

... ... ...

render(){
    var mm='AABB';
    return (<div><button onClick={this.one.bind(this,mm)}> test</button>
    <button onClick={this.one.bind(this,this.state.name)}> test</button></div>);

}}
```

阿珂 8个月前 (07-26)

理解 React 中 es6 创建组件 this 的方法

在JavaScript中，this对象是运行时基于函数的执行环境（也就是上下文）绑定的。

从 react 中的 demo 说起

Facebook最近一次更新react时，将es6中的class加入了组件的创建方式当中。Facebook也推荐组件创建使用通过定义一个继承自 React.Component 的class来定义一个组件类。官方的demo：

```
class LikeButton extends React.Component {
  constructor() {
    super();
    this.state = {
      liked: false
    };
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    this.setState({liked: !this.state.liked});
  }
  render() {
    const text = this.state.liked ? 'liked' : 'haven\'t liked';
    return (
      <div onClick={this.handleClick}>
        You {text} this. Click to toggle.
      </div>
    );
  }
}
ReactDOM.render(
  <LikeButton />,
  document.getElementById('example')
);
```

上面的demo中有大量this的使用，在 class LikeButton extends React.Component 中我们是声明该class，因为this具体是由其上下文决定的，因此在类定义中我们无法得知this用法。 相当于是new了上面定义的类，首先调用 constructor() 函数， this.state 的this上下文就是该实例对象；同理，render() 函数中 this.state.liked 的this上下文也是该对象。问题在于 onClick={this.handleClick} ，获取该函数引用是没有问题，这里的this上下文就是该对象。

这时问题来了，在原来 React.createClass 中， handleClick() 在onClick事件触发的时候，会自动绑定到LikeButton实例上，这时候该函数的this的上下文就是该实例。不过在ES6的class的写法中，Facebook取消了自动绑定，实例化LikeButton后，handleClick()的上下文是div的支撑实例（ backing instance ），而 handleClick() 原本要绑定的上下文是LikeButton的实例。对于该问题，我们有多种解决方案。

使用 bind() 函数改变 this 的上下文

可以在class声明中的constructor()函数中，使用：

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

该方法是一个bind()绑定，多次使用。在该方法中，我们在声明该实例后，可以在该实例任何地方使用 handleClick() 函数，并且该 handleClick() 函数的this的上下文都是LikeButton实例对象。

除此我们也可以在具体使用该函数的地方绑定this的上下文到LikeButton实例对象，代码如下：

```
<div onClick={this.handleClick.bind(this)}>
  You {text} this. Click to toggle.
</div>
```

这种方法需要我们每次使用bind()函数绑定到组件对象上。

es6的箭头函数

es6中新加入了箭头函数=>，箭头函数除了方便之外还有而一个特征就是将函数的this绑定到其定义时所在的上下文。这个特征也可以帮助我们解决这个问题。使用如下代码：

```
<div onClick={() => this.handleClick()}>
  You {text} this. Click to toggle.
</div>
```

这样该 this.handleClick() 的上下文就会被绑定到 LikeButton 的实例对象上。个人认为，使用箭头函数使得 JavaScript 更加接近面向对象的编程风格。

this 的总结

this 的本质就是：this跟作用域无关的，只跟执行上下文有关。

YuCun Zhong 5个月前 (10-20)