# 第四章数据流

在React中，数据的流向是单向的——从父节点传递到子节点，因而组件是简单且易于把握的，它们只需从父节点获取props渲染即可。如果顶层组件的某个prop改变了，React会递归地向下遍历整颗组件树，重新渲染所有使用这个属性的组件。

React组件内部还具有自己的状态，这些状态只能在组件内修改。React组件本身很简单，你可以把它们看成是一个函数，它接受props和state作为参数，返回一个虚拟的DOM表现。

在本章我们将学习：

A. props是什么。

B. state是什么。

C. 什么时候用props以及什么时候用state。

## Props

Props就是properties的缩写，你可以使用它把任意类型的数据传递给组件。

Var surveys=[{title:’Superheroes’}];

<ListSurveys surveys={surveys} />

或者通过调用组件实例的setProps方法（很少需要这样做）来设置其props：

Var surveys=[{title:’Superheroes’}];

Var listSurveys=React.render(

<ListSurveys />,

Document.querySelector(‘body’)

);

listSurvery.setProps({surveys:surveys});

你只能在子组件上或者组件树外（如上例）调用setProps。千万别调用this.setProps或者直接修改this.props,如果真的需要，请使用state，我们将会在本章的后面讨论它。

可以通过this.props访问props，但绝对不能通过这种方式修改它。一个组件绝对不可以修改自己的props。

在JSX中，可以把props设置为字符串：

<a href=”/surveys/add”>Add survey</a>

也可以使用{}语法来设置，注入JavaScript传递任意类型的变量：

<a href={‘/surveys/’+survey.id}>{survey.title}</a>

还可以使用JSX的展开语法把props设置成一个对象：

Var ListSurveys=React.createClass({

Render:function(){

Var props={

One:’foo’,

Two:’bar’

};

Return <SurveyTable {…props} />

}

});

Props还可以用来添加事件处理器：

Var SaveButton=React.createClass({

Render:function(){

Return(

<a className=”button save” onClick={this.handleClick}>Save</a>

);

}

handleClick:function(){

…

}

});

这里我们给链接标签传递了一个onClick属性，值为handleClick函数。当用户点击链接时，handleClick方法将被调用。

## PropTypes

通过在组件中定义一个配置对象，React提供了一种验证props的方式：

var SurveyTableRow=React.createClass({

propTypes:{

survey:React.PropTypes.shape({

id:React.PropTypes.number.isRequired

}).isRequired,

onClick:React.PropTypes.func

},

// …

});

组件初始化时，如果传递的属性和propTypes不匹配，则会打印一个console.warn日志。

如果是可选的配置，则可以去掉 .isRequired.

在应用中使用propTypes并不是强制性的，但这提供了一种极好的方式来描述组件的API。

## getDefaultProps

可以为组件添加getDefaultProps函数来设置属性的默认值。不过，这应该只针对那些非必需属性。

var SurveyTable = React.createClass({

getDefaultProps:function(){

return {

surveys:[]

};

}

//…

}) ;

必须要提醒的一点是，getDefaultProps并不是在组件实例化时被调用的，而是在React.createClass调用时就被调用了，返回值会被缓存起来。也就是说，不能在getDefaultProps中使用任何特定的实例数据。

## State

每一个React组件都可以拥有自己的State，state与props的区别在于前者只存在于组件的内部。

State可以用来确定一个元素的视图状态。我们来看一个自定义的<Dropdown>组件：

var CountryDropdown=React.createClass({

getInitialState:function(){

return {

showOptions:false

};

},

render:function(){

var options;

if(this.state.showOptions){

options=(

<ul className=’option’>

<li>United States of America</li>

<li>New Zealand</li>

<li>Denmark</li>

</ul>

);

}

Return (

<div className=”dropdown” onClick={this.handleClick}>

<label>Choose a country</label>

</div>

);

},

handleClick:function(){

this.setState({

showOptions:true

})

}

});

在上例中，state被用来记录是否在下拉框中显示可选项。

State可以通过setState来修改，也可以使用上面出现的getInitialState方法提供一组默认值。只要setState被调用，render就会被调用。如果render函数的返回值有变化，虚拟DOM就会更新，真实的DOM也会被更新，最终用户就会在浏览器中看到变化。

千万不能直接修改this.state,永远记得要通过this.setState方法修改。

状态总是让组件变得更加复杂，但是如果把状态针对不同的组件独立开来，应用就会更容易调试一些。

## 放在state和props的各是哪些部分

不要在state中保存计算出的值，而应该只保存最简单的数据，即那些组件正常工作时的必要数据。比如前面出现过的勾选状态，如果没有它就无法勾选（或不勾选）；比如用来表示下拉框是否显示的布尔值，又比如输入框的值，等等。

不要尝试把props复制到state中。要尽可能把props当做数据源。

## 总结

本章我们学习了：

A. 使用props在整个组件树中传递数据和配置。

B. 避免在组件内部修改this.props或调用this.setProps,请把props当做是只读。

C. 使用props来做事件处理器，与子组件通信。

D. 使用state存储简单的视图状态，比如说下拉框是否可见这样的状态。

E. 使用this.setState来设置状态，而不要使用this.state直接修改状态。

这一章中我们简单提到了事件处理器，下一章将深入到它的更多细节中。