# 第9章 表单

表单是应用必不可少的一部分，只要需要用户输入，哪怕是最简单的输入，都离不开表单。一直以来，单页应用中的表单都很难处理好，因为表单中充斥着用户变化莫测的状态。要管理好这些状态很费神，也很容易出现bug。React可以帮助你管理应用中的状态，自然也包括表单在内。

现在，你应该知道React组件的核心理念就是可预知性和可测试性。给定同样的props和state，任何React组件都会渲染出一样的结果。表单也不例外。

在React中，表单组件有两种类型：约束组件和无约束组件。我们在本章中将会学习两者的差别，以及在什么场景下选择哪种组件。

本章内容包括：

A. React中表单事件的使用。

B. 使用约束的表单组件来控制数据输入。

C. 如何使用React修改表单组件界面。

D. 在React中表单组件命名的重要性。

E. 多个约束的表单组件的处理。

F. 创建自定义的可服用的表单组件。

G. 在React中使用Autofocus.

H. 创建高可用性应用的建议。

## 无约束的组件

你可能不想在很多重要的表单中使用无约束组件，但它们会帮助你更好地理解约束组件的概念。无约束组件的构造与React中大多数组件相比是反模式的。

在HTML中，表单组件与React组件的行为方式并不一致。给定HTML的<input />一个值，这个<input />的值仍是可以改变的。这正是无约束组件名称的由来，因为表单组件的值是不受React组件控制的。

在React中，这种行为与设置<input /> 的defaultValue一致。

我们可以通过defaultValue属性设置<input />的默认值。

var MyForm=React.createClass({

render:function(){

return <input type=”text” deafultValue=”Hello World” />

}

});

上面这个例子展示的就是无约束组件。组件的value并非由父组件设置，而是让<input />自己控制自己的值。

一个无约束的组件没有太大的用处，除非可以访问它的值。因此需要给<input />添加一个ref属性，以访问Dom节点的值。

Ref是一个不属于DOM属性的特殊属性，用来标记DOM节点，可以通过this上下文访问这个节点。为了便于访问，组件中所有的ref都添加到了this.refs上。

下面我们在表单中添加一个<input />,并在表单提交时访问它的值。

Var MyForm=React.createClass({

submitHandler:function(event){

event.preventDefault();

//通过ref访问输入框

Var helloTo=this.refs.helloTo.getDOMNode().value;

Alert(helloTo);

}

Render:function(){

Return (

<form onSubmit={this.submitHandler}>

<input ref=”helloTo” type=”text” defaultValue=”Hello World1” />

<button type=”submit”>Speak</button>

</form>

);

}

});

无约束组件可以用在基本的无须任何验证或者输入控制的表单中。

## 约束组件

约束组件与React其他类型组件的模式一致。表单组件的状态交由React组件控制，状态值被存储在React组件的state中。

如果想要更好地控制表单组件，推荐使用约束组件。

在约束组件中，输入框的值是由父组件设置的。

让我们把之前的例子改成约束组件：

Var MyForm=React.createClass({

getInitialState:function(){

return {

helloTo:”Hello World”

};

},

handleChange:function(event){

this.setState({

helloTo:event.target.value

});

},

submitHandler:function(event){

event.preventDefault();

alert(this.state.helloTo);

},

Render:function(){

Return (

<form onSubmit={this.submitHandler}>

<input type=”text” value={this.state.helloTo} onChange={this.handleChange} />

<br />

<button type=”submit”>Speak</button>

</form>

);

}

});

其中最显著的变化就是<input />的值存储在父组件state中。因此数据流有了清晰的定义。

A. getInitialState设置defaultValue。

B. <input />的值在渲染时被设置。

C. <input />的值onChange时，change处理器被调用。

D.change处理器更新state。

E.在重新渲染时更新<input />的值。

虽然与无约束组件相比，代码量增加不少，但是现在可以控制数据流，在用户输入数据时更新state。

示例：用户输入的时候，把字符都转成大写。

HandleChange：function(event){

This.setState({

helloTo:event.target.value.toUpperCase()

});

}

你可能会注意到，用户输入数据后，小写字符转成大写形式并添加到输入框时，并不会发生闪烁。这是因为React拦截了浏览器原生的change事件，在setState被调用后，这个组件就会重新渲染输入框。然后React计算差异，更新输入框的值。

你可以使用同样的方式来限制可输入的字符集，或者限制用户向邮件地址输入框中输入不合法的字符串。

你还可以使用同样的方式来限制用户输入数据时，把它们用在其他的组件上。例如：

A. 显示一个有长度限制的输入框还可以输入多少字符。

B. 显示输入的HEX值所代表的颜色。

C. 显示可自动匹配下拉列表的可选项。

D. 使用输入框的值更新其他UI元素。

## 表单事件

访问表单事件是控制表单不同部分的一个非常重要的方面。

React支持所有的HTML事件。这些事件遵循驼峰命名的约定，且会被转成合成事件。这些事件是标准化的，提供跨浏览器的一致接口。

所有合成事件都提供了event.target来访问触发事件的DOM节点。

HandleEvent:function(syntheticEvent){

Var DOMNode=syntheticEvent.target;

Var newValue=DOMNode.value;

}

这是访问约束组件的值的最简单方法之一。

### Label

Label是表单元素中很重要的组件，通过Label可以明确地向用户传达你的要求，提升单选框和复选框的可用性。

但Label与for属性有一个冲突的地方。因为如果使用JSX，这个属性会被转换成一个JavaScript对象，且作为第一个参数传递给组件的构造器。但由于for属于javaScript的一个保留字，所以我们无法把它作为一个对象的属性。

在React中，与class变成了className类似，for也变成了htmlFor。

// JSX

<label htmlFor=”name”>Name:</label>

//javascript

React.DOM.label({htmlFor:”name”},”Name”);

//渲染后

<label for=”name”>Name:</label>

### 文本框和Select

React对<textarea />和<select />的接口做了一些修改，提升了一致性，让它们操作起来更容易。

<textarea />被改得更像<input />了，允许我们设置value和defaultValue.

//非约束的

<textarea defaultValue=”Hello World” />

//约束的

<textarea value=”{this.state.helloTo}” onChange={this.handleChange} />

<select />现在接受value和defaultValue来设置已选项，我们可以更容易地对它的值进行操作。

//非约束的

<select defaultValue=”B”>

<option value=”A”>First Option</option>

<option value=”B”>Second Option</option>

<option value=”C”>Third Option</option>

</select>

//约束的

<select value={this.state.helloTo} onChange={this.handleChange} >

<option value=”A”>First Option</option>

<option value=”B”>Second Option</option>

<option value=”C”>Third Option</option>

</select>

React支持多选select。你需要给value和defaultValue传递一个数组。

//非约束的

<select multiple=”true” defaultValue={[“A”,”B”]} >

<option value=”A”>First Option</option>

<option value=”B”>Second Option</option>

<option value=”C”>Third Option</option>

</select>

当使用可多选的select时，select组件的值在选项被选择时不会更新，只有选项的selected属性会发生变化。你可以使用ref或者syntheticEvent.target来访问选项，检查它们是否被选中。

在下面的例子中，handleChange循环检查DOM，并过滤出那些选项被选中了。

Var MyForm=React.createClass({

getInitialState:function(){

return {

options:[“B”]

};

},

handleChange:function(event){

var checked=[];

var sel=event.target;

for (var i=0;i<sel.length;i++){

var option=sel.options[i];

if(option.selected){

checked.push(option.value);

}

}

this.setState({

options:checked

});

},

submitHandler:function(event){

event.preventDefault();

alert(this.state.options);

},

Render:function(){

Return (

<form onSubmit={this.submitHandler} >

<select multiple=”true” value={this.state.options} onChange={this.handleChange} >

<option value=”A”>First Option</option>

<option value=”B”>Second Option</option>

<option value=”C”>Third Option</option>

</select>

<br />

<button type=”submit”>Speak</button>

</form>

)

}

});

### 复选框和单选框

复选框和单选框使用的则是另外一种完全不同的控制方式。

在HTML中，类型为checkbox或radio的<input />与类型为text的<input />的行为完全不一样。通常，复选框或者单选框的值是不变的，只有checked状态会变化。要控制复选框或者单选框，就要控制它们的checked属性。你也可以在非约束的复选框或者单选框中使用defaultChecked。

//非约束的

Var MyForm=React.createClass({

submitHandler:function(event){

event.preventDefault();

alert(this.refs.checked.getDOMNode().checked);

},

Render:function(){

Return (

<form onSubmit={this.submitHandler}>

<input ref=”checked” type=”checkbox” value=”A” defaultChecked=”true” />

</br>

<button type=”submit”>Speak</button>

</form>

);

}

});

//约束的

Var MyForm=React.createClass({

getInitialState:function(){

return {

checked:true

};

},

handleChange:function(event){

this.setState({

checked:event.target.checked

});

},

submitHandler:function(event){

event.preventDefault();

alert(this.state.checked);

},

Render:function(){

Return (

<form onSubmit={this.submitHandler}>

<input type=”checkbox” value=”A” checked={this.state.checked} onChange={this.handleChange} />

<br />

<button type=”submit”>Speak</button>

</form>

);

}

});

在这两个例子中，<input />的值一直都是A，只有checked的状态在变化。

### 表单元素的name属性

在React中，name属性对于表单元素来说并没有那么重要，因为约束表单组件已经把值存储到了state中，并且表单提交的事件也会被拦截。在获取表单值的时候，name属性并不是必须的。对于非约束的表单组件来说，也可以使用refs来直接访问表单元素。

即便如此，name仍然是表单组件中非常重要的一部分。

A. name属性可以让表单序列化类库在React中正常工作。

B. 对于仍然使用传统提交方式的表单来说，name属性是必需的。

C. 在用户的浏览器中，name被用在自动填写常用信息中，比如用户地址等。

D. 对于非约束的单选框组件来讲，name是很有必要的，它可作为这些组件分组的依据，确保在同一时刻，同一表单中拥有同样name的单选框只有一个可以被选中。如果不使用name属性，这一行为可以使用约束的单选框实现。

下面这个例子把状态存储在MyForm组件中，实现了非约束单选框具备的分组功能。请注意，这里并没有使用name属性。

var MyForm=React.createClass({

getInitialState:function(){

return {

radio:”B”

};

},

handleChange:function(event){

this.setState({

radio:event.target.value

});

},

submitHandler:function(event){

event.preventDefault();

alert(this.state.radio);

},

Render:function(){

<form onSubmit={this.submitHandler}>

<input type=”radio” value=”A” checked={this.state.radio==”A”} onChange={this.handleChange} /> A <br/>

<input type=”radio” value=”B” checked={this.state.radio==”B”} onChange={this.handleChange} />B <br/>

<input type=”radio” value=”C” checked={this.state.radio==”C”} onChange={this.handleChange} />C <br/>

<input type=”radio” value=”D” checked={this.state.radio==”D”} onChange={this.handleChange} />D <br/>

</form>

}

})

### 多个表单元素与change处理器

在使用约束的表单组件时，没人愿意重复地为每一个组件编写change处理器。还好有几种方式可以在React中重用一个事件处理器。

示例一：通过.bind传递其他参数。

var MyForm=React.createClass({

getInitialState:function(){

return {

give\_name:””,

family\_name:””

};

},

handleChange:function(name,event){

var newState={};

newState[name]=event.target.value;

this.setState(newState);

},

submitHandler:function(event){

event.preventDefault();

var words=[

“Hi”,this.state.given\_name,this.state.family\_name

];

alert(words.join(“ ”));

},

Render:function(){

Return (

<form onSubmit={this.submitHandler}>

<label htmlFor=”give\_name”>Given Name:</label>

<br />

<input

type=”text”

name=”given\_name”

value={this.state.given\_name}

onChange={this.handleChange.bind(this,’given\_name’)}

/>

<br />

<label htmlFor=”family\_name”>Family Name:</label>

<br />

<input

type=”text”

name=”family\_name”

value={this.state.family\_name}

onChange={this.handleChange.bind(this,’family\_name’)}

/>

</br>

<button type=”submit”>Speak</button>

</form>

);

}

});

示例二：使用DOMNode的name属性来判断需要更新哪些组件的状态。

var MyForm=React.createClass({

getInitialState:function(){

return {

give\_name:””,

family\_name:””

};

},

handleChange:function(event){

var newState={};

newState[event.target.name]=event.target.value;

this.setState(newState);

},

submitHandler:function(event){

event.preventDefault();

var words=[

“Hi”,this.state.given\_name,this.state.family\_name

];

alert(words.join(“ ”));

},

Render:function(){

Return (

<form onSubmit={this.submitHandler}>

<label htmlFor=”give\_name”>Given Name:</label>

<br />

<input

type=”text”

name=”given\_name”

value={this.state.given\_name}

onChange={this.handleChange}

/>

<br />

<label htmlFor=”family\_name”>Family Name:</label>

<br />

<input

type=”text”

name=”family\_name”

value={this.state.family\_name}

onChange={this.handleChange}

/>

</br>

<button type=”submit”>Speak</button>

</form>

);

}

});

上面这两个例子很相似，解决同样的问题，但使用了不同的方式。React还在addon中提供了一个mixin，React.addons.LinkedStateMixin,通过另一种方式解决同样的问题。

React.addons.LinkedStateMixin为组件提供了一个linkState方法。linkState返回一个对象，包含value和requestChange两个属性。

value根据提供的name属性从state中获取对应的值。

requestChange是一个函数，使用新的值更新同名的state。

this.linkState(‘given\_name’);

// 返回

{

value:this.state.given\_name,

requestChange:function(newValue){

this.setState({

given\_name:newValue

});

}

}

需要把这个对象传递给一个React特有的非DOM属性valueLink。ValueLink使用对象提供的value更新表单域的值，并提供一个onChange处理器，当表单域更新时使用新的值调用requestChange。

Var MyForm=React.createClass({

Mixins:[React.addons.LinkedStateMixin],

getInitialState:function(){

return {

given\_name:””,

family\_name:””

};

},

submitHandler:function(event){

event.preventDefault();

var words=[

“Hi”,this.state.given\_name,this.state.family\_name

];

}

});