# 第5章 事件处理

对用户界面而言，展示只占整体设计因素的一半。另一半则是响应用户输入，即通过JavaScript处理用户产生的事件。

React通过将事件处理器绑定到组件上来处理事件。在事件被触发的同时，更新组件的内部状态。组件的内部状态的更新会触发组件重绘。因此，如果视图层想要渲染出事件触发后的结果，它所要做的就是在渲染函数中读取组件的内部状态。

尽管简单地根据正在处理中的事件类型来更新内部状态的做法很常见，但还是有必要使用事件的额外信息来判断如何更新状态。在此情况下，传递给处理器的事件对象还会额外提供与事件相关的信息，方便在更改组件内部状态时使用。

借助这些技术以及React高效的渲染，我们能够更容易地响应用户的输入并根据输入内容来更新用户界面。

## 绑定事件处理器

React处理的事件本质上和原生JavaScript事件一样：MouseEvents事件用于点击处理器，Change事件用于表单元素变化，等等。所有的事件在命名上与原生JavaScript规范一致，并且会在相同的情景下被触发。

React绑定事件处理器的语法和HTML语法非常类似。比如，在我们的问卷制作工具示例中包含了下面的代码，在Save按钮上绑定onClick事件处理器。

<button className=”btn btn-save” onClick={this.handleSaveClicked}>Save</button>

用户点击这个按钮时，组件的handleSaveClicked方法会被调用。这个方法会包含处理Save行为的逻辑。

注意，这份代码在写法上类似普遍不推荐的HTML内联事件处理器属性，比如onClick,但其实在底层实现上并没有使用HTML的onClick属性。React只是用这种写法来绑定事件处理器，其内部则按照需要高效地维护着事件处理器。

如果不用JSX，你可以选择在参数对象的属性上指定事件处理器。比如：

React.DOM.button({className:”btn btn-save”,onClick:this.handleSaveClicked},”save”);

React对处理各种事件类型提供了友好的支持，具体的支持类型列在了其文档的事件系统中。

其中绝大部分事件不需要额外的处理就能工作，但是触控事件需要通过调用以下的代码手动启用。

React.initializeTouchEvents(true);

## 事件和状态

设想你需要让一个组件随着用户的输入而改变，比如在问卷编辑器中，你想要让用户从一个问题类型的菜单当中拖拽问题。

首先，在渲染函数内部基于HTML5拖放API注册事件处理器，代码如下。

Var SurveyEditor=React.createClass({

Render:function(){

return (

<div className=’survey-editor’>

<div className=’row’>

<aside className=’sidebar col-md-3’>

<h2>Modules</h2>

<DraggableQuestions />

</aside>

<div className=”survey-canvas col-md-9”>

<div

className={‘drop-zone well well-drop-zone’}

onDrapOver={this.handleDragOver}

onDragEnter={this.handleDragEnter}

onDragLeave={this.handleDrapLeave}

onDrop={this.handleDrop}

</div>

</div>

</div>

)

}

});

这个DraggableQuestions组件将会渲染问题类型的菜单，绑定的事件处理器方法负责处理拖放行为。

### 根据状态进行渲染

事件处理器方法还需要完成一件事——展开当前已经加入的题目清单。为了实现该功能，你需要充分利用每个React组件的内部状态。组件状态默认是null，但是可以通过它的getInitialState方法将其初始化为合理的值，比如：

getInitialState：function(){

return{

dropZoneEntered:false,

title:’’,

introduction:’’,

questions:[]

};

}

以上代码为组件状态初始化了默认值：一个空标题、一个空的介绍、一组空的题目以及一个值为false的dropZoeEntered属性，用于表示当前用户没有拖拽任何内容到放置区域。

到这里，你已经可以在render方法当中读取this.state,以便向用户展示当前表单中的所有数据了。

render:function(){

var questions=this.state.questions;

var dropZoneEntered=’’;

if(this.state.dropZoneEntered){

dropZoneEntered=”drap-enter”;

}

Return (

<div className=”survey-editor”>

<div className=”row”>

<aside className=”sidebar col-md-3”>

<h2>Modules</h2>

<DraggableQuestions />

</aside>

<div className=’survey-canvas col-md-9’>

<SurveyForm

title={this.state.title}

introduction={this.state.introduction}

onChange={this.handleFormChange}

/>

<Divider>Questions</Divider>

<ReactCSSTransitionGroup transitionName=’question’>

{questions}

</ReactCSSTransitionGroup>

<div

className={‘drop-zone well well-drop-zone’+dropZoneEntered}

onDragOver={this.handleDragOver}

onDragEnter={this.handleDragEnter}

onDragLeave={this.handleDragLeave}

onDrop={this.handleDrop}

>

Drag and frop a module from the left

</div>

<div className=’actions’>

<button className=”btn btn-save” onClick={this.handleSaveClicked} >Save</button>

</div>

</div>

</div>

</div>

)

}

与this.props类似，render函数可能有或多或少的变化，这完全取决于this.state值。它可以渲染出属性上稍有差异的相同元素，或者渲染出完全不同的元素集。无论哪种方式，效果都一样好。

### 更新状态

更新组件的内部状态会触发组件重绘，所以接下来要做的事情就是在拖拽的事件处理器方法中更新状态。然后再次运行render函数，它会从this.state中读取新数据来显示标题、介绍及题目，用户将看到所有内容被正确地更新。

更新组件状态有两种方案：组件的setState方法和replaceState方法。replaceState用一个全新的state对象完整的替换掉原有的state。使用不可变数据结构来表示状态时，这种方式很有效，不过很少应用于其他场景下。更多的情况下会使用setState，它仅仅是把传入的对象合并到已有的state对象。

比如说，假设下面的代码表示当前状态：

getInitialState:function(){

return {

dropZoneEntered:false,

title:’Fantastic:Survey’,

introduction:’This survey is fantastic!’,

questions:[]

};

}

这是，调用this.setState({title:”Fantastic Survey 2.0”})仅仅影响this.state.title的值，而this.state.dropZoneEntered、this.state.introduction及this.state.questions不会受影响。

而如果调用this.replaceState({title:”Fantastic Survey 2.0”})，则会用新的对象{title:”Fantastic Survey 2.0”}替换掉整个state对象，同时把this.state.dropZoneEntered、this.state.introduction和this.state.questions全部清除掉。这样做很可能中断render函数的执行，因为它期望this.state.questions是一个数组而不是undefined。

现在使用this.setState可以实现上文提到的事件处理器方法。

handleFormChange:function(formData){

this.setState(formData);

}

有一点很重要，永远不要尝试通过setState或者replaceState以外的方式去修改state对象。类似this.state.saveInProgress=true通常不是一个好主意，因为它无法通知React是否需要重新渲染组件，而且可能会导致下次调用setsSate时出现意外。

## 事件对象

很多事件处理器只要触发就会完成功能，但有时也会需要关于用户输入的更多信息。

我们来看一下问卷制作工具示例应用中的AnswerEssayQuestion类。

var AnswerEssayQuestion=React.createClass({

handleComplete:function(event){

this.callMethodOnProps(‘onCompleted’,event.target.value);

},

render:function(){

return (

<div className=”form-group”>

<label className=”survey-item-label”>{this.props.label}</label>

<div className=”survey-item-content”>

<textarea className=”form-control” rows=”3” onBlur={this.handleComplete} />

</div>

</div>

);

}

});

通常会有一个事件对象传入到React的事件处理器函数中，类似原生JavaScript事件监听器的写法。这里的handleComplete方法会接受一个事件对象，并通过存取event.target.value值为textarea赋值。在事件处理器中，使用event.target.value获取表单中input值是一种常规的方法，尤其在onChange事件处理器中。

注意，callMethodOnProps是由一个叫作PropsMethodMixin的mixin提供的。此外，这个mixin还提供了一些处理父子组件之间通信的简便方法。

React把原生的事件封装在一个SyntheticEvent实例当中，而不是直接把原生的浏览器事件对象传给事件处理器。SyntheticEvent在表现和功能上都与浏览器的原生事件对象一致，并且消除了某些跨浏览器差异，因此你应该可以像使用普通的事件一样使用SyntheticEvent。对于那些需要浏览器原生事件的场景，则可以通过SyntheticEvent的nativeEvent属性访问到它。

### 总结

从用户输入到更新用户界面，处理步骤非常简单：

1、在React组件上绑定事件处理器。

2、在事件处理器当中更新组件的内部状态。组件状态的更新会触发重绘。

3、实现组件的render函数用来渲染this.state的数据。

到这里我们已经学会用单个组件来响应用户交互了。接下来将继续学习怎样将多个组件复合在一起，构建功能复杂的界面。