# 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。