# 特性

组件化，安全

# 语法

var Nav;

// 输入 (JSX):

var app = <Nav color="blue" />;

// 输出 (JS):

var app = React.createElement(Nav, {color:"blue"});

# 组件

## 自定义的 React 类名以大写字母开头

//组件

var CommentBox = React.createClass({

render: function(){

return (

<div className = "commentBox">

Hello,world!I am a commentBox!

</div>

);

}

});

通过 JavaScript 对象传递一些方法到 React.createClass() 来创建一个新的 React 组件。其中最重要的方法是 render ，该方法返回一棵 React 组件树，这棵树最终将会渲染成 HTML。

这里的div不是DOM节点div，只是React的组件实例，可以简单认为是一些标记或数据。

ReactDOM.render() 实例化根组件，启动框架，把标记注入到第二个参数指定的原生的 DOM 元素中

# 使用属性（props）

从父组件传入的数据会做为子组件的 属性（ property ） ，这些 属性（ properties ） 可以通过 this.props 访问到。使用属性（ props ），我们就可以读到从 CommentList 传到 Comment 的数据，然后渲染一些标记

如：

子组件：

var Comment = React.createClass({

render: function() {

return (

<div className="comment">

<h2 className="commentAuthor">

{this.props.author} //获取author属性

</h2>

{this.props.children} //获取子节点

</div>

);

}

});

父组件：

var CommentList = React.createClass({

render: function(){

return (

<div className="commentList" >

<Comment author="ybb"><div>第一条评论</div></Comment>

<Comment author="ybb1"><div>第二条评论</div></Comment>

<Comment author="ybb2"><div>第三条评论</div></Comment>

<Comment author="ybb3"><div>第四条评论</div></Comment>

</div>

);

}

});

this.props.author 获取comment中author=”ybb”的值

this.props.children获取comment下的组件，类似jQ的children()方法

# 使用Markdown语法

var Comment = React.createClass({

rawMarkup: function() {

var rawMarkup = marked(this.props.children.toString(), {sanitize: true});

return { \_\_html: rawMarkup };

},

render: function(){

return (

<div className="comment" >

<h1>{this.props.author}</h1>

<span dangerouslySetInnerHTML={this.rawMarkup()} />

</div>

);

}

});

sanitize: true告诉 marked 转义掉评论文本中的 HTML 标签而不是直接原封不动地返回这些标签

# 接入数据类型

## 自定义数据

var data = [

{author: "Pete Hunt", text: "This is one comment"},

{author: "Jordan Walke", text: "This is \*another\* comment"}

];

<CommentBox data={data}/> 添加数据

<CommentList data={this.props.data}/> 在CommentBox里面的CommentList

var commentNode = this.props.data.map(function(comment){

return <Comment author={comment.author}>{comment.text}</Comment>

})

return (

<div className="commentList" >

{commentNode}

</div>

); 在CommentList中处理一下数据

## 从服务器获取数据

<CommentBox url = “/api/comments”>

### 响应状态变化（ Reactive state ）

this.state 是组件私有的，可以通过调用 this.setState() 来改变它。当 state 更新之后，组件就会重新渲染自己.

render() 方法依赖于 this.props 和 this.state ，框架会确保渲染出来的 UI 界面总是与输入（ this.props 和 this.state ）保持一致.

给CommentBox添加state

var CommentBox = React.createClass({

getInitialState: function() {

return {data: []};

},

componentDidMount: function() {

$.ajax({

url: this.props.url,

dataType: 'json',

cache: false,

success: function(data) {

this.setState({data: data});

}.bind(this),

error: function(xhr, status, err) {

console.error(this.props.url, status, err.toString());

}.bind(this)

});

},

render: function() {

return (

<div className="commentBox">

<h1>Comments</h1>

<CommentList data={this.state.data} />

<CommentForm />

</div>

);

}

});

**getInitialState()** 在组件的生命周期中仅执行一次，用于设置组件的初始化 state 。

**componentDidMount** 是一个组件渲染的时候被 React 自动调用的方法。

# 事件

## onsubmit

form表单提交事件

render: function() {

return (

<form className="commentForm" onSubmit={this.handleSubmit}>

<input type="text" placeholder="Your name" ref="author" />

<input type="text" placeholder="Say something..." ref="text" />

<input type="submit" value="Post" />

</form>

);

}

handleSubmit: function(e) {

e.preventDefault(); //避免浏览器默认地提交表单

},

input中利用 **ref** 属性给子组件命名，通过 **this.refs** 引用 DOM 节点