

# React 官方文档翻译

林晨<lc1990linux@gmail.com>

May 30, 2016

## Contents

<b>1 案例学习</b>	<b>1</b>
1.1 想跳过所有的直接看全部源代码?	1
1.2 注释: 下面是我的源代码简单研读:	1
1.2.1 server.js	2
1.2.2 example.js	2
1.3 运行server	2
1.4 开始	2
1.5 第一个组件	3
1.6 JSX语法:	4
1.7 发生了什么	4
1.8 Composing 组件	5
1.9 使用porps	6
1.10 组件属性	6
1.11 添加Markdown支持	7
1.12 连接数据模型	8
1.13 从服务器获取数据	10
1.14 React的state	10
1.14.1 更新state	11
1.14.2 添加新的评论	13
1.14.3 事件	14
1.14.4 提交组件	14
1.14.5 回调作为属性	16
1.15 优化: 优化更新	19
1.16 恭喜	21
<b>2 React编程思想</b>	<b>21</b>
2.1 拆分UI成组件结构	23

## 1 案例学习

我们将构建一个简单实用的评论箱，你可以把它放到博客里。并提供实时评论。我们将提供：

- 一个查看全部评论的视图
- 一个提交评论的表单
- 为你提供自定义后台的接口

它也有一些整洁的特性：

- 优化的评论：评论在被保存到服务器之前就会在页面上列出来，这样会显得很快。
- 实时更新：其他用户的评论会实时的跳进来
- Markdown格式：用户可以用markdown来格式化他们的文本。

### 1.1 想跳过所有的直接看全部源代码？

[全部在github上](#)

### 1.2 注释：下面是我的源代码简单研读：

主要是两个源文件：server.js和example.js。

#### 1.2.1 server.js

主要是nodejs，express提供的文件存储，读取的后台数据api，demo没有使用数据库，而是使用了简单的json文件来做数据存储。主要提供了2个api，get所有的comments和post一批comments。

#### 1.2.2 example.js

Comment类作为一个评论实体。CommentList类用来列出所有的Comments CommentForm类处理一个comment的提交表单。CommentBox把所有的封装起来，处理数据交互逻辑。

### 1.3 运行server

为了运行这个例子，我们将需要一个运行的server。它将提供API端为我们获取和传递数据。为了尽量简化，我们创建一个几行脚本语言就完成的简单服务器。你可以看server.js的代码。为了简化，运行的server使用json文件来做存储数据库。你不应该在生产环境运行它，但这很简单地模拟了你想做的。当你开启服务器，它会提供API端和我们需要的静态页面。

### 1.4 开始

这个案例里我们将尽量简化他，我们用一个HTML文件来引入我们前面讨论的服务器包。打开 /public/index.html，

```
<!-- index.html -->
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>React Tutorial</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react/0.14.7/react.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react/0.14.7/react-dom.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/browser.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.0/jquery.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/marked/0.3.5/marked.min.js"></script>
  </head>
  <body>
    <div id="content"></div>
    <script type="text/babel" src="scripts/example.js"></script>
    <script type="text/babel">
      // To get started with this tutorial running your own code, simply remove
      // the script tag loading scripts/example.js and start writing code here.
    </script>
  </body>
</html>
```

本文余下部分，我们在script标签下写JS代码。我们没有使用任何先进的

实时重载技术，所以，你需要手动刷新你的浏览器来看更新。打开浏览器输入 `http://localhost:3000`（在开启server之后），当你第一次不做任何修改载入时，你会看到我们想构建的完成后的产品。你想开始工作时，只需删掉预先放置的<script>标签即可。:: 注意 我们使用了JQuery来简化ajax调用，但这并不是react必须的选择。

## 1.5 第一个组件

React全是模块化的，组件化的组件。对于我们的评论箱，我们需要以下组件

- CommentBox
  - CommentList
  - \* Comment
- CommentForm

让我们来构建CommentBox组件。他是个简单的<div>:

```
// tutorial1.js
var CommentBox = React.createClass({
  render: function() {
    return (
      <div className="commentBox">
        Hello, world! I am a CommentBox.
      </div>
    );
  }
});
ReactDOM.render(
  <CommentBox />,
  document.getElementById('content')
);
```

注意原生html元素的名字都以小写字母开头，但自定义的React类名字以大写字母开头，

## 1.6 JSX语法:

首先你注意到的是XML风格的语法在你的JS里，我们有一个简单的预编译器把这些语法糖转化成普通的js。被转化成的js code

```
1 // tutorial1-raw.js
2 var CommentBox = React.createClass({displayName: 'CommentBox',
3   render: function() {
4     return (
5       React.createElement('div', {className: "commentBox"},
6         "Hello, world! I am a CommentBox."
7       )
8     );
9   }
10 });
11 ReactDOM.render(
12   React.createElement(CommentBox, null),
13   document.getElementById('content')
14 );
```

使用jsx语法不是必须的，你可以做选择，但我们发现jsx语法比不同js更容易使用，[这里](#)可以读到更多JSX语法文件。

## 1.7 发生了什么

我们在js对象里传递一些方法到React.createClass()里去创建一个React组件。这些方法里最重要的是一个叫 **render** 的方法。它返回一个React组件树，最终会被渲染成HTML。<div>标签并非实际的DOM节点，他们是React的<div>组件的实例，你可以把这些看做是React可以处理的符号或数据块，React是安全的，我们不会生成HTML字符串，所以会默认有XSS保护。你不需要返回基本HTML，你可以返回一棵全是自己创建的组件树，正是这一点使得React是 **组件化**：这是可维护前端的一个关键信条。ReactDOM.render() 实例化根组件，启动框架，并且根据第二个参数把这些标记注入成原生DOM元素。ReactDOM模块提供DOM相关的方法，而React模块为React及其相关平台提供和谐工具集（如React-Native，使用react框架来写ios和android原生应用）把ReactDOM.render放在脚本的最后是非常重要的，ReactDOM.render 应该在组件定义之后被调用。

## 1.8 Composing 组件

下面来构建 **CommentList** 和 **\*CommentForm**” 的框架。他们还是简单的<div>。添加这两个组件到你的文件里，保持已有的CommentBox声明和ReactDOM.render调用。

```
1 // tutorial2.js
```

```
2 var CommentList = React.createClass({
3   render: function() {
4     return (
5       <div className="commentList">
6         Hello, world! I am a CommentList.
7       </div>
8     );
9   }
10 });
11
12 var CommentForm = React.createClass({
13   render: function() {
14     return (
15       <div className="commentForm">
16         Hello, world! I am a CommentForm.
17       </div>
18     );
19   }
20 });
```

接下来，用这些新组建更新CommentBox组件。

```
1 // tutorial3.js
2 var CommentBox = React.createClass({
3   render: function() {
4     return (
5       <div className="commentBox">
6         <h1>Comments</h1>
7         <CommentList />
8         <CommentForm />
9       </div>
10     );
11   }
12 });
```

注意我们是怎么混合html标签和组件的。HTML组件是普通React组件，和你自定义的组件一样，唯一不同的是JSX编译器自动重写HTML标签到React.createElement(tagName)语句，和其他所有的东西都隔离开。这样是为了避免全局命名空间的污染。

## 1.9 使用props

我们创建Comment组件，会依赖从父组件传来的数据。从父组件传入的数据，子组件可以通过‘Property’来访问。这些‘Property’存在this.props。使用props，我们能读取从CommentList传来Comment的数据，并渲染一些符号：

```
1 // tutorial4.js
2 var Comment = React.createClass({
3   render: function() {
4     return (
5       <div className="comment">
6         <h2 className="commentAuthor">
7           {this.props.author}
8         </h2>
9         {this.props.children}
10      </div>
11    );
12  }
13 });
```

JSX里把JS语句括在大括号里（像一个属性或一个子语句，你可以丢文本或者React组件到标签树上。我们获取通过this.props来获取传来的命名属性，通过this.props.children来获取任何封装元素。

## 1.10 组件属性

现在我们有定义好的Comment组件，我们想传进作者名字和评论文本。我们可以在每一个不同的评论重用相同代码，我们来给CommentList加一些评论。

```
1 // tutorial5.js
2 var CommentList = React.createClass({
3   render: function() {
4     return (
5       <div className="commentList">
6         <Comment author="Pete Hunt">This is one comment</
7       Comment>
8         <Comment author="Jordan Walke">This is *another* comment</
9       Comment>
10      </div>
11    );
12  }
13 });
```

```
9     );  
10  }  
11  });
```

我们从父组件CommentList给子组件Comment传入了一些数据。例如我们传入Pete Hunt（属性），这是一个Comment（通过XML类似的子节点）。

### 1.11 添加Markdown支持

Markdown是最简单的方法来格式化内嵌文本的，例如用星号包裹的文本会被加粗。我们用了第三方库 **marked** 来把markdown格式的文本转换成原生HTML。我们已经添加了这个库，所有直接使用就可以了。我们来转换并输出它。

```
1  // tutorial6.js  
2  var Comment = React.createClass({  
3    render: function() {  
4      return (  
5        <div className="comment">  
6          <h2 className="commentAuthor">  
7            {this.props.author}  
8          </h2>  
9          {marked(this.props.children.toString())}  
10         </div>  
11       );  
12     }  
});
```

我们做的只是直接使用这个库,我们需要把this.props.children从React封装文本转换成原始字符串,所以我们要调用toString()方法. 但有一个问题! 我们的渲染的评论在浏览器里大概是这样:” <p>This is <em>another</em> comment</p>”. 我们想这些标签可以确切地渲染成HTML. 这是为了防止你遭受XSS 攻击, 有个方法可以避免, 但框架会警告你不要使用它。

```
1  // tutorial7.js  
2  var Comment = React.createClass({  
3    rawMarkup: function() {  
4      var rawMarkup = marked(this.props.children.toString(), {sanitize: true});  
5      return { __html: rawMarkup };  
6    },  
7  });
```



```
8   render: function() {
9     return (
10      <div className="comment">
11        <h2 className="commentAuthor">
12          {this.props.author}
13        </h2>
14        <span dangerouslySetInnerHTML={this.rawMarkup()} />
15      </div>
16    );
17  }
18 });
```

这个特别的API故意让插入原始HTML变得困难。但对于marked库，我们会故意利用这个后门。记住：使用这个特性，你将依靠marked来保证安全。这个例子里，我们传入 `sanitize: true` 告诉marked 跳过源代码里的任何HTML符号，而不是不加修改地全部传进去。

## 1.12 连接数据模型

目前为止，我们已经在源码直接插入评论，我们要渲染一个一团JSON数据到评论列表中，实际上这些应该来自于服务器，为了简化现在我们把json手写进代码里。

```
1 // tutorial8.js
2 var data = [
3   {id: 1, author: "Pete Hunt", text: "This is one comment"},
4   {id: 2, author: "Jordan Walke", text: "This is *another* comment"}
5 ];
```

我们需要用模块化的方法把这些数据获取进评论列表里。修改 `CommentBox` 和 `ReactDOM.render()` 方法,通过props传入这些数据到 `CommentList` 里。

```
1 // tutorial9.js
2 var CommentBox = React.createClass({
3   render: function() {
4     return (
5       <div className="commentBox">
6         <h1>Comments</h1>
7         <CommentList data={this.props.data} />
8         <CommentForm />
9       </div>
10     );
11   }
12 });
```

```
10     );
11   }
12 });
13
14 ReactDOM.render(
15   <CommentBox data={data} />,
16   document.getElementById('content')
17 );
```

现在，这些数据可以在 `CommentList` 中使用了，我们可以动态渲染这些评论了。

```
1 // tutorial10.js
2 var CommentList = React.createClass({
3   render: function() {
4     var commentNodes = this.props.data.map(function(comment) {
5       return (
6         <Comment author={comment.author} key={comment.id}>
7           {comment.text}
8         </Comment>
9       );
10    });
11    return (
12      <div className="commentList">
13        {commentNodes}
14      </div>
15    );
16  }
17 });
```

就这样。

### 1.13 从服务器获取数据

现在我们用服务器获取的动态数据来替换写死的数据，我们删除 `data` 属性，替换成 `URL` 来获取

```
1 // tutorial11.js
2 ReactDOM.render(
3   <CommentBox url="/api/comments" />,
4   document.getElementById('content')
5 );
```

这个组件不同于之前的，他需要重渲染自己。组件没有任何数据，直到它向服务器请求到数据，这时，这个组件需要去渲染一些新组件。>注意：现在这些代码还不能正常工作

### 1.14 React的state

目前为止，基于props，每个组件都已经渲染他自己，props不可改变，它从父组件传来，被父组件拥有，为了实现交互，我们引入组件可变的 **state**。`=this.state=` 是组件私有的，可以通过`this.setState()` 来改变。当state更新，组件就会重新渲染。`render`方法是`this.props`和`this.state`的函数，框架保证UI始终和输入一致。当服务器获取数据，我们将修改评论数据，让我们来添加一组评论数据作为state到评论箱中。

```
// tutorial12.js
var CommentBox = React.createClass({
  getInitialState: function() {
    return {data: []};
  },
  render: function() {
    return (
      <div className="commentBox">
        <h1>Comments</h1>
        <CommentList data={this.state.data} />
        <CommentForm />
      </div>
    );
  }
});
```

`getInitialState()` 在组件生命周期内仅会执行一次，设置组件的初始状态。

#### 1.14.1 更新state

当组件第一次被创建，我们想从服务器上GET一些JSON数据，并更新state以反映最新的数据。我们将用jQuery框架来向服务器发送异步请求我们需要的数据，数据包含在`common.json`中，一旦获取，`this.state.data`看起来会想这样

```
[
  {"author": "Pete Hunt", "text": "This is one comment"},
  {"author": "Jordan Walke", "text": "This is *another* comment"}
```

```
]

// tutorial13.js
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>
    );
  }
});
```

componentDidMount会在组件第一次渲染之后被自动调用。动态更新的关键是调用this.setState(), 我们用新数组替换旧的,UI就会随之自动更新。因为这个相关性, 它只是一个小改变去添加实时更新, 我们这里将用简单的ajax获取但你可以更简单的websockets或其他技术。

```
// tutorial14.js
var CommentBox = React.createClass({
  loadCommentsFromServer: 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)
  });
},
getInitialState: function() {
  return {data: []};
},
componentDidMount: function() {
  this.loadCommentsFromServer();
  setInterval(this.loadCommentsFromServer, this.props.pollInterval);
},
render: function() {
  return (
    <div className="commentBox">
      <h1>Comments</h1>
      <CommentList data={this.state.data} />
      <CommentForm />
    </div>
  );
}
});

ReactDOM.render(
  <CommentBox url="/api/comments" pollInterval={2000} />,
  document.getElementById('content')
);
```

这里我们只是把ajax调用单独放到一个函数,当组件首次加载,之后每隔2s就重新调用它.在你的浏览器里运行它,并修改你的json文件,每隔2s,这些改变就会出现在页面上.

### 1.14.2 添加新的评论

现在可以构建表单了,评论表单组件应该输入评论者姓名和评论内容来向服务器发送请求存储这条评论.

```
// tutorial15.js
var CommentForm = React.createClass({
  render: function() {
    return (
      <form className="commentForm">
        <input type="text" placeholder="Your name" />
        <input type="text" placeholder="Say something..." />
        <input type="submit" value="Post" />
      </form>
    );
  }
});
```

受控的组件 传统DOM里, input 元素被呈现, 浏览器管理他的 state (也就是他索要渲染的值) 结果实际DOM的state可能与组件里保存的state不同. 这不理想, 因为视图的state与组件的状态不同. 在React里, 组件应该始终代表试图的state, 而不仅仅在初始化的时刻. 因此, 我们将用this.state来存储用户的输入. 我们用 author 和 text 两个内容为空字符串的属性定义一个初始化的state. 在<input>元素中, 我们设置 value 属性来反应组件状态, 并且绑定 onChange 回调函数, 这些<input> 元素带有一个设定的值 被称为受控组件. 关于更多受控组件的信息可以读[这里](#)

```
// tutorial16.js
var CommentForm = React.createClass({
  getInitialState: function() {
    return {author: '', text: ''};
  },
  handleAuthorChange: function(e) {
    this.setState({author: e.target.value});
  },
  handleTextChange: function(e) {
    this.setState({text: e.target.value});
  },
  render: function() {
    return (
      <form className="commentForm">
```

```
        <input
          type="text"
          placeholder="Your name"
          value={this.state.author}
          onChange={this.handleAuthorChange}
        />
        <input
          type="text"
          placeholder="Say something..."
          value={this.state.text}
          onChange={this.handleTextChange}
        />
        <input type="submit" value="Post" />
      </form>
    );
  }
});
```

### 1.14.3 事件

React 对绑定组件的事件用驼峰命名法。我们绑定onChange回调函数到两个<input>元素,现在,当用户录入文本到<input>区域绑定的onChange回调函数会被触发,组件的state会被修改,随即,input元素的渲染值会被更新以反映当前组件的state.

### 1.14.4 提交组件

我们来让组件可交互,当用户提交组件,我们应该清楚它,提交请求到服务器 并刷新评论列表。我们收听组件提交时间并清理它。

```
// tutorial17.js
var CommentForm = React.createClass({
  getInitialState: function() {
    return {author: '', text: ''};
  },
  handleAuthorChange: function(e) {
    this.setState({author: e.target.value});
  },
  handleTextChange: function(e) {
    this.setState({text: e.target.value});
  },
});
```

```
    },
    handleSubmit: function(e) {
      e.preventDefault();
      var author = this.state.author.trim();
      var text = this.state.text.trim();
      if (!text || !author) {
        return;
      }
      // TODO: send request to the server
      this.setState({author: '', text: ''});
    },
    render: function() {
      return (
        <form className="commentForm" onSubmit={this.handleSubmit}>
          <input
            type="text"
            placeholder="Your name"
            value={this.state.author}
            onChange={this.handleAuthorChange}
          />
          <input
            type="text"
            placeholder="Say something..."
            value={this.state.text}
            onChange={this.handleTextChange}
          />
          <input type="submit" value="Post" />
        </form>
      );
    }
  });
```

我们在表单上绑定一个onSubmit回调函数并且在提交表单的时候清除表单域。对事件调用 preventDefault 是为了阻止浏览器在提交表单时的默认行为。

#### 1.14.5 回调作为属性

当用户提交评论，我们将需要刷新评论列表以包含这个信提交的评论。这些逻辑应该写在 CommentBox 里，因为CommentBox拥有表示评论列表



的state。我们需要传递数据从子组件回到他的父组件。我们在父组件的render方法里通过传递一个新的回调函数到子组件中,绑定到子组件的onCommentSubmit方法上.只有事件触发,回调就会被唤起。

```
// tutorial18.js
var CommentBox = React.createClass({
  loadCommentsFromServer: 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)
    });
  },
  handleCommentSubmit: function(comment) {
    // TODO: submit to the server and refresh the list
  },
  getInitialState: function() {
    return {data: []};
  },
  componentDidMount: function() {
    this.loadCommentsFromServer();
    setInterval(this.loadCommentsFromServer, this.props.pollInterval);
  },
  render: function() {
    return (
      <div className="commentBox">
        <h1>Comments</h1>
        <CommentList data={this.state.data} />
        <CommentForm onCommentSubmit={this.handleCommentSubmit} />
      </div>
    );
  }
});
```

注意CommentBox已经使得这个回调函数对CommentForm可以使用, 通

过onCommentSubmit属性。CommentForm在用户提交表单的时候调用这个回调函数。

```
// tutorial19.js
var CommentForm = React.createClass({
  getInitialState: function() {
    return {author: '', text: ''};
  },
  handleAuthorChange: function(e) {
    this.setState({author: e.target.value});
  },
  handleTextChange: function(e) {
    this.setState({text: e.target.value});
  },
  handleSubmit: function(e) {
    e.preventDefault();
    var author = this.state.author.trim();
    var text = this.state.text.trim();
    if (!text || !author) {
      return;
    }
    this.props.onCommentSubmit({author: author, text: text});
    this.setState({author: '', text: ''});
  },
  render: function() {
    return (
      <form className="commentForm" onSubmit={this.handleSubmit}>
        <input
          type="text"
          placeholder="Your name"
          value={this.state.author}
          onChange={this.handleAuthorChange}
        />
        <input
          type="text"
          placeholder="Say something..."
          value={this.state.text}
          onChange={this.handleTextChange}
        />
        <input type="submit" value="Post" />
      </form>
    );
  }
});
```

```
        </form>
      );
    }
  });
```

现在回调函数已经归位了,我们需要做的仅仅是提交到服务器并刷新列表.

```
// tutorial20.js
var CommentBox = React.createClass({
  loadCommentsFromServer: 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)
    });
  },
  handleCommentSubmit: function(comment) {
    $.ajax({
      url: this.props.url,
      dataType: 'json',
      type: 'POST',
      data: comment,
      success: function(data) {
        this.setState({data: data});
      }.bind(this),
      error: function(xhr, status, err) {
        console.error(this.props.url, status, err.toString());
      }.bind(this)
    });
  },
  getInitialState: function() {
    return {data: []};
  },
  componentDidMount: function() {
    this.loadCommentsFromServer();
```

```
    setInterval(this.loadCommentsFromServer, this.props.pollInterval);
  },
  render: function() {
    return (
      <div className="commentBox">
        <h1>Comments</h1>
        <CommentList data={this.state.data} />
        <CommentForm onSubmit={this.handleCommentSubmit} />
      </div>
    );
  }
});
```

### 1.15 优化：优化更新

我们的app现在功能已经完全了,但反应缓慢,在你的评论出现在列表中之前你需要等待请求.我们可以优化添加这个评论到列表中让app感觉更快.

```
// tutorial21.js
var CommentBox = React.createClass({
  loadCommentsFromServer: 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)
    });
  },
  handleCommentSubmit: function(comment) {
    var comments = this.state.data;
    // Optimistically set an id on the new comment. It will be replaced by an
    // id generated by the server. In a production application you would likely
    // not use Date.now() for this and would have a more robust system in place.
    comment.id = Date.now();
    var newComments = comments.concat([comment]);
```

```
this.setState({data: newComments});
$.ajax({
  url: this.props.url,
  dataType: 'json',
  type: 'POST',
  data: comment,
  success: function(data) {
    this.setState({data: data});
  }.bind(this),
  error: function(xhr, status, err) {
    this.setState({data: comments});
    console.error(this.props.url, status, err.toString());
  }.bind(this)
});
},
getInitialState: function() {
  return {data: []};
},
componentDidMount: function() {
  this.loadCommentsFromServer();
  setInterval(this.loadCommentsFromServer, this.props.pollInterval);
},
render: function() {
  return (
    <div className="commentBox">
      <h1>Comments</h1>
      <CommentList data={this.state.data} />
      <CommentForm onSubmit={this.handleCommentSubmit} />
    </div>
  );
}
});
```

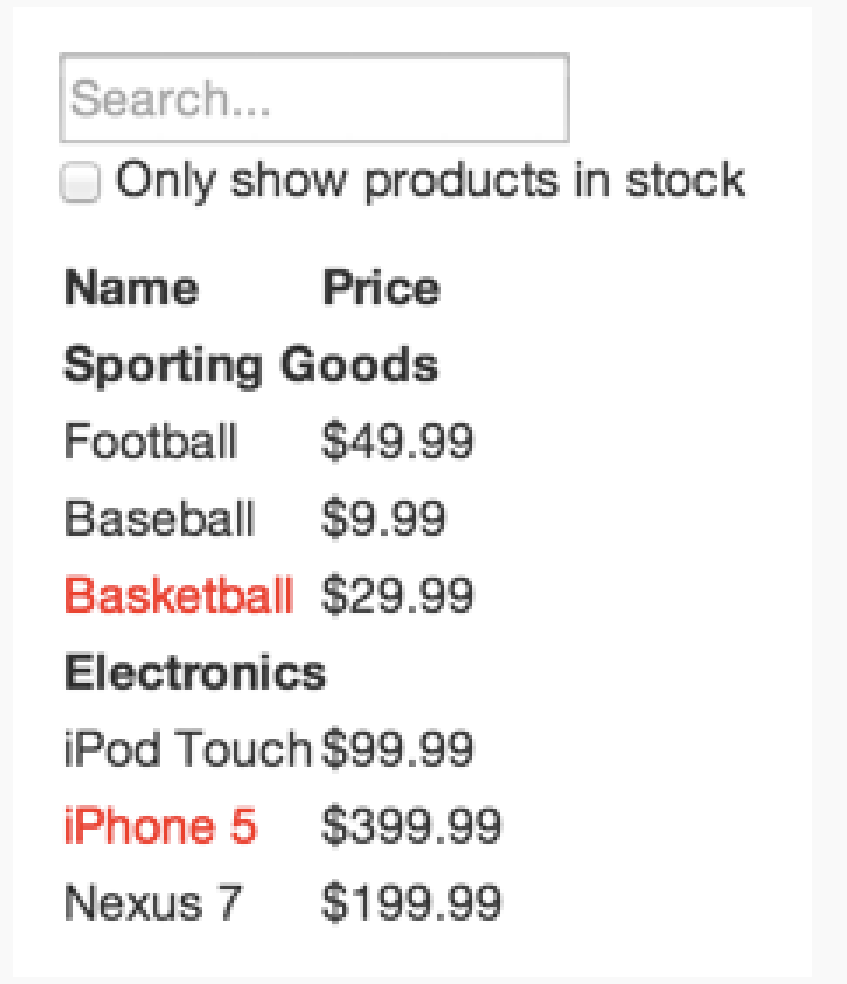
## 1.16 恭喜

你已经在几步简单的步骤下构建了一个评论箱，想学习更多React，可以读[为什么使用React](#)或深入研究[API参考](#)。祝你好运！

## 2 React编程思想

在我看来，React是用JS构建大型快速的Web app最重要的方法。他在Facebook和Instagram工作的非常好。React的一个很棒的地方在于他帮助你思考你的apps，当你构建他们的时候。我讲带你走一遍这个过程，我们用React来构建一个可搜索的产品数据表。

我们先从模拟开始。假设我们从设计师那里有个JSON api和一个模拟器，们的设计师显然不是特别好，因为模拟器长成这个样子。



Name	Price
<b>Sporting Goods</b>	
Football	\$49.99
Baseball	\$9.99
<b>Basketball</b>	\$29.99
<b>Electronics</b>	
iPod Touch	\$99.99
<b>iPhone 5</b>	\$399.99
Nexus 7	\$199.99

我们的JSON API返回的数据大概是这个样子。

```
[  
  {category: "Sporting Goods", price: "$49.99", stocked: true, name: "Football"},
```

```
{category: "Sporting Goods", price: "$9.99", stocked: true, name: "Baseball"},  
{category: "Sporting Goods", price: "$29.99", stocked: false, name: "Basketball"},  
{category: "Electronics", price: "$99.99", stocked: true, name: "iPod Touch"},  
{category: "Electronics", price: "$399.99", stocked: false, name: "iPhone 5"},  
{category: "Electronics", price: "$199.99", stocked: true, name: "Nexus 7"}  
];
```

## 2.1 拆分UI成组件结构

首先你会想在每个组件和子组件上画框框。并且给他们命名。如果你和一个设计师一起工作，他们可能已经帮你做了这些。所以，走去和他们交流。他们的PS图层名称可以作为你的React组件名。

但你怎么知道哪些应该成为它自己的组件。同样的原理，何时你觉得应该创建一个函数和对象。其中一个原理就是 **单一职责原则** 也就是，一个组件最好只做一件事，如果它持续增长，它就应该拆分成更小的子组件。因为你经常展示JSON数据，你会发现如果你的model构建正确，你的UI（就是你的组件结构）也会同样的美观。因为UI和data model将粘附在同样的信息结构上，这意味着