React.js 小书

<-- 返回首页

渲染列表数据

- 作者：胡子大哈
- 原文链接： http://huziketang.com/books/react/lesson13

列表数据在前端非常常见，我们经常要处理这种类型的数据，例如文章列表、评论列表、用户列表…一个前端工程师几乎每天都需要跟列表数据打交道。

React.js 当然也允许我们处理列表数据，但在使用 React.js 处理列表数据的时候，需要掌握一些规则。我们这一节会专门讨论这方面的知识。

渲染存放 JSX 元素的数组

假设现在我们有这么一个用户列表数据，存放在一个数组当中：

```
const users = [
  { username: 'Jerry', age: 21, gender: 'male' },
  { username: 'Tomy', age: 22, gender: 'male' },
  { username: 'Lily', age: 19, gender: 'female' },
  { username: 'Lucy', age: 20, gender: 'female' }
]
```

如果现在要把这个数组里面的数据渲染页面上要怎么做？开始之前要补充一个知识。之前说过 JSX 的表达式插入 `{}` 里面可以放任何数据，如果我们往 `{}` 里面放一个存放 JSX 元素的数组会怎么样？

```
...

class Index extends Component {
  render () {
    return (
      <div>
        {[
          <span>React.js </span>,
          <span>is </span>,
          <span>good</span>
        ]}
      </div>
    )
  }
}
```

```
ReactDOM.render(
  <Index />,
  document.getElementById('root')
)
```

我们往 JSX 里面塞了一个数组，这个数组里面放了一些 JSX 元素（其实就是 JavaScript 对象）。到浏览器中，你在页面上会看到：

审查一下元素，看看会发现什么：

```
▼<div data-reactroot class="wrapper">
  ▼<div>
      <span>React.js </span>
      <span>is </span>
      <span>good</span>
    </div>
  </div>
```

React.js 把插入表达式数组里面的每一个 JSX 元素一个个罗列下来，渲染到页面上。所以这里有个关键点：**如果你往 `{}` 放一个数组，React.js 会帮你把数组里面一个个元素罗列并且渲染出来。**

使用 map 渲染列表数据

知道这一点以后你就可以知道怎么用循环把元素渲染到页面上：循环上面用户数组里面的每一个用户，为每个用户数据构建一个 JSX，然后把 JSX 放到一个新的数组里面，再把新的数组插入 `render` 方法的 JSX 里面。看看代码怎么写：

```
const users = [
  { username: 'Jerry', age: 21, gender: 'male' },
  { username: 'Tomy', age: 22, gender: 'male' },
  { username: 'Lily', age: 19, gender: 'female' },
  { username: 'Lucy', age: 20, gender: 'female' }
]

class Index extends Component {
  render () {
    const usersElements = [] // 保存每个用户渲染以后 JSX 的数组
```

```
    for (let user of users) {
      usersElements.push( // 循环每个用户，构建 JSX，push 到数组中
        <div>
          <div>姓名：{user.username}</div>
          <div>年龄：{user.age}</div>
          <div>性别：{user.gender}</div>
          <hr />
        </div>
      )
    }

    return (
      <div>{usersElements}</div>
    )
  }
}

ReactDOM.render(
  <Index />,
  document.getElementById('root')
)
```

这里用了一个新的数组 `usersElements`，然后循环 `users` 数组，为每个 `user` 构建一个 JSX 结构，然后 push 到 `usersElements` 中。然后直接用表达式插入，把这个 `userElements` 插到 return 的 JSX 当中。因为 React.js 会自动化帮我们把数组当中的 JSX 罗列渲染出来，所以可以看到页面上显示：

但我们一般不会手动写循环来构建列表的 JSX 结构，可以直接用 ES6 自带的 `map`（不了解 `map` 函数的同学可以先了解相关的知识再来回顾这里），代码可以简化成：

```
class Index extends Component {
  render () {
    return (
      <div>
        {users.map((user) => {
          return (
            <div>
              <div>姓名：{user.username}</div>
              <div>年龄：{user.age}</div>
              <div>性别：{user.gender}</div>
              <hr />
            </div>
          )
        })}
      </div>
    )
  }
}
```

这样的模式在 JavaScript 中非常常见，一般来说，在 React.js 处理列表就是用 `map` 来处理、渲染的。现在进一步把渲染单独一个用户的结构抽离出来作为一个组件，继续优化代码：

```
const users = [
  { username: 'Jerry', age: 21, gender: 'male' },
  { username: 'Tomy', age: 22, gender: 'male' },
  { username: 'Lily', age: 19, gender: 'female' },
  { username: 'Lucy', age: 20, gender: 'female' }
]

class User extends Component {
  render () {
    const { user } = this.props
    return (
      <div>
        <div>姓名：{user.username}</div>
        <div>年龄：{user.age}</div>
        <div>性别：{user.gender}</div>
        <hr />
      </div>
    )
  }
}

class Index extends Component {
  render () {
    return (
      <div>
        {users.map((user) => <User user={user} />)}
      </div>
    )
  }
```

```
}

ReactDOM.render(
  <Index />,
  document.getElementById('root')
)
```

这里把负责展示用户数据的 JSX 结构抽离成一个组件 `User` ，并且通过 `props` 把 `user` 数据作为组件的配置参数传进去；这样改写 `Index` 就非常清晰了，看一眼就知道负责渲染 `users` 列表，而用的组件是 `User`。

key! key! key!

现在代码运作正常，好像没什么问题。打开控制台看看：

React.js 报错了。如果需要详细解释这里报错的原因，估计要单独写半本书。但可以简单解释一下。

React.js 的是非常高效的，它高效依赖于所谓的 Virtual-DOM 策略。简单来说，能复用的话 React.js 就会尽量复用，没有必要的话绝对不碰 DOM。对于列表元素来说也是这样，但是处理列表元素的复用性会有一个问题：元素可能会在一个列表中改变位置。例如：

```
<div>a</div>
<div>b</div>
<div>c</div>
```

假设页面上有这么3个列表元素，现在改变一下位置：

```
<div>a</div>
<div>c</div>
<div>b</div>
```

`c` 和 `b` 的位置互换了。但其实 React.js 只需要交换一下 DOM 位置就行了，但是它并不知道其实我们只是改变了元素的位置，所以它会重新渲染后面两个元素（再执行 Virtual-DOM 策略），这样会大大增加 DOM 操作。但如果给每个元素加上唯一的标识，React.js 就可以知道这两个元素只是交换了位置：

```
<div key='a'>a</div>
<div key='b'>b</div>
<div key='c'>c</div>
```

这样 React.js 就简单的通过 `key` 来判断出来，这两个列表元素只是交换了位置，可以尽量复用元素内部的结构。

这里没听懂没有关系，后面有机会会继续讲解这部分内容。现在只需要记住一个简单的规则：**对于用表达式套数组罗列到页面上的元素，都要为每个元素加上 `key` 属性，这个 `key` 必须是每个元素唯一的标识**。一般来说，`key` 的值可以直接后台数据返回的 `id`，因为后台的 `id` 都是唯一的。

在上面的例子当中，每个 `user` 没有 `id` 可以用，可以直接用循环计数器 `i` 作为 `key`：

```
...
class Index extends Component {
  render () {
    return (
      <div>
        {users.map((user, i) => <User key={i} user={user} />)}
      </div>
    )
  }
}
...
```

再看看，控制台已经没有错误信息了。但这是不好的做法，这只是掩耳盗铃（具体原因大家可以自己思考一下）。记住一点：在实际项目当中，如果你的数据顺序可能发生变化，标准做法是最好是后台数据返回的 `id` 作为列表元素的 `key`。

课后练习

- 打印章节标题

> 因为第三方评论工具有问题，对本章节有任何疑问的朋友可以移步到 React.js 小书的论坛 发帖，我会回答大家的疑问。

下一节：实战分析：评论功能（一）

上一节：state vs props

或者传播一下知识也是一个很好的选择

3 条评论，3 人参与。 ★ 4

我有话说...

使用社交帐号登录 发布前先点击左边的按钮登录

最新评论

鸡冷 • 4月27日 23:05

map是ES5的方法

1顶 • 回复 • 分享»

果冻 • 4月24日 10:33

为什么是 掩耳盗铃 。。。

顶 • 回复 • 分享»

H_ai_ 果冻 • 4月25日 01:10

i是循环生成的，b,c对调如果key是abc就可以看出来key变成acb了， 如果key是下标i，对调前后key都是012，没啥变化。粗浅理解

1顶 • 回复 • 分享»

友言?