路由的魔法：React Router

随着 AJAX 技术的成熟，现在单页应用（Single Page Application）已经是前端网页界的标配，名为“单页”，其实在设计概念上依然是多页的界面，只不过从技术层面上页之间的切换是没有整体网页刷新的，只需要做局部更新。

要实现“单页应用”，一个最要紧的问题就是做好“路由”（Routing)，也就是处理好下面两件事：

1. 把 URL 映射到对应的页面来处理；
2. 页面之间切换做到只需局部更新。

感谢业界前人给我们开发者铺平了道路，在 React 的世界里，上面说的问题都有成熟解法，其中最热门的工具，就是 react-router，这一节我们就来介绍这个工具。

react router v4 的动态路由

我们现在说到 react-router，基本上都是在说 react-router 的第 4 版，也就是 v4。这个 v4 很有意思，它完全推翻了之前 v3 的做法。可以说，react-router 的 v3 和 v4 版完完全全是不同的两个工具，两者差距实在太大。

其实当初 v3 也已经很优秀很热门了，但是 react-router 的开发者不满意，他们认为 v3 还是落入了“静态路由”的窠臼，所以在 v4 中 react-router 做到了“动态路由”的功能。

所谓“静态路由”，就是说路由规则是固定的，无论 express、Angular 还是 Rails 等业界响当当的框架，都用的是静态路由。以 express 为例，路由规则差不多是这么写的：

```
app.get('/', Home);
app.get('/product/:id', Product);
app.get('/about', About);
```

对于大部分应用，支持这样的路由规则真的是足够了，但是，react-router 的开发者觉得这样还不够好，要支持“动态路由”才是最好。

所谓动态路由，指的是路由规则不是预先确定的，而是在渲染过程中确定的。因为 react-router 的定位就是专供 React 应用服务，而 React 的世界中一切皆为组件，所以 react-router v4 就完全用 React 组件来实现路由功能。

不得不承认，虽然 react-router 的开发者是挺折腾的，但是他们的确是领悟了 React 的精髓，而且在 react-router 中把 React 的哲学发挥到了极致。

接下来，我们通过一个很简单的例子来说明 react-router v4 如何工作的，然后在这个例子的基础上介绍“动态路由”。

React Router 实例

安装包 react-router-dom

create-react-app 产生的应用默认为不支持多个页面，但还是在 README 文件中友情推荐了一下 react-router 来增强功能，可见 react-router 影响力之大。

不过，我们并不需要安装 react-router 这个 npm 包，因为 react-router 为了支持 Web 和 React Native 出了两个包—— react-router-dom 和 react-router-native，我们只关心 Web，所以只需要安装 react-router-dom 。这个 react-router-dom 依赖于 react-router，所以 react-router 也会被自动安装上。

```
npm install react-router-dom
```

HashRouter 还是 BrowserRouter

react-router 的工作方式，是在组件树顶层放一个 Router 组件，然后在组件树中散落着很多 Route 组件（注意比 Router 少一个“r”），顶层的 Router 组件负责分析监听 URL 的变化，在它保护伞之下的 Route 组件可以直接读取这些信息。

很明显，Router 和 Route 的配合，就是之前我们介绍过的“提供者模式”，Router 是“提供者”，Route是“消费者”。

更进一步，Router 其实也是一层抽象，让下面的 Route 无需各种不同 URL 设计的细节，不要以为 URL 就一种设计方法，至少可以分为两种。

第一种很自然，比如 `/` 对应 Home 页，`/about` 对应 About 页，但是这样的设计需要服务器端渲染，因为用户可能直接访问任何一个 URL，服务器端必须能对 `/` 的访问返回 HTML，也要对 `/about` 的访问返回 HTML。

第二种看起来不自然，但是实现更简单。只有一个路径 `/`，通过 URL 后面的 `#` 部分来决定路由，`/#/` 对应 Home 页，`/#/about` 对应 About 页。因为 URL 中 `#` 之后的部分是不会发送给服务器的，所以，无论哪个 URL，最后都是访问服务器的 `/` 路径，服务器也只需要返回同样一份 HTML 就可以，然后由浏览器端解析 `#` 后的部分，完成浏览器端渲染。

在 react-router，有 BrowserRouter 支持第一种 URL，有 HashRouter 支持第二种 URL。

因为 create-react-app 产生的应用默认不支持服务器端渲染，为了简单起见，我们在下面的例子中使用 HashRouter，在实际产品中，其实最好还是用 BrowserRouter，这样用户体验更好。

修改 `index.js` 文件，增加下面的代码：

```
import {HashRouter} from 'react-router-dom';

ReactDOM.render(
  <HashRouter>
    <App />
  </HashRouter>,
  document.getElementById('root')
);
```

把 Router 用在 React 组件树的最顶层，这是最佳实践。因为将来我们如果想把 HashRouter 换成 BrowserRouter，组件 App 以下几乎不用任何改变。

使用 Link

对于单页应用，需要在不同“页面”之间切换，往往需要一个“导航栏”，我们在这里也实现一个简单的导航栏。

在 `App.js` 中，我们让网页由两个组件 Navigation 和 Content 组成， Navigation 就是导航栏，而 Content 是具体内容。

```
class App extends Component {
  render() {
    return (
      <div className="App">
        <Navigation />
        <Content />
      </div>
    );
  }
}
```

我们计划只增加两个页面，在 Navigation 中就应该有两个链接，但是，如果我们简单使用 HTML 的 `<a>` 标签那就错了，用户点击 `<a>` 标签缺省行为是网页跳转，这违背了“单页应用”的原则。虽然对于 HashRouter 使用的是没有网页跳转的 `#`，但是为了将来可以无缝切换为 BrowserRouter，我们也不能使用 `<a href="#about">` 这样的标签。

正确的解法是用 react-router 提供的 Link 组件，虽然 Link 最终还是渲染为 `<a>` 标签，但这是有神力的 `<a>` 标签，用户点击时，react-router 可以知晓这是一个单页应用的链接，不用网页跳转只做局部页面更新。

```
const ulStyle = {
  'list-style-type': 'none',
  margin: 0,
  padding: 0,
};

const liStyle = {
  display: 'inline-block',
  width: '60px',
};

const Navigation = () => (
  <header>
    <nav>
      <ul style={ulStyle}>
        <li style={liStyle}><Link to='/'>Home</Link></li>
        <li style={liStyle}><Link to='/about'>About</Link></li>
      </ul>
    </nav>
  </header>
)
```

使用 Route 和 Switch

我们来看 Content 这个组件，这里会用到 react-router 最常用的两个组件 Route 和 Switch。

```
const Content = () => (
  <main>
    <Switch>
      <Route exact path='/' component={Home}/>
      <Route path='/about' component={About}/>
    </Switch>
  </main>
)
```

Route 组件的 path 属性用于匹配路径，因为我们需要匹配 `/` 到 `Home`，匹配 `/about` 到 `About`，所以肯定需要两个 Route，但是，我们不能这么写。

```
      <Route path='/' component={Home}/>
      <Route path='/about' component={About}/>
```

如果按照上面这么写，当访问 `/about` 页面时，不光匹配 `/about`，也配中 `/`，界面上会把 Home 和 About 都渲染出来的。

解决方法，可以在想要精确匹配的 Route 上加一个属性 `exact`，或者使用 Switch 组件。

可以把 Switch 组件看做是 JavaScript 的 switch 语句，像这样：

```
switch (条件) {
  case 1: 渲染1; break;
  case 2: 渲染2; break;
}
```

从上往下找第一个匹配的 Route，匹配中了之后，立刻就 break，不继续这个 Switch 下其他的 Route 匹配了。

可以看到，react-router 巧妙地用 React 组件实现了路由的所有逻辑，印证了那句话：React 世界里一切都是组件。

动态路由

在了解了 react-router的基本路由功能之后，再来理解“动态路由”就容易了。

假设，我们增加一个新的页面叫 `Product`，对应路径为 `/product`，但是只有用户登录了之后才显示。如果用静态路由，我们在渲染之前就确定这条路由规则，这样即使用户没有登录，也可以访问 `product`，我们还不得不在 Product 组件中做用户是否登录的检查。

如果用动态路由，则只需要在代码中的一处涉及这个逻辑：

```
    <Switch>
      <Route exact path='/' component={Home}/>
      {
        isUserLogin() &&
        <Route exact path='/product' component={Product}/>,
      }
      <Route path='/about' component={About}/>
    </Switch>
```

可以用任何条件决定 Route 组件实例是否渲染，比如，可以根据页面宽度、设备类型决定路由规则，动态路由有了最大的自由度。

小结

这一小节我们介绍了 React 世界中最热门的路由工具 react-router，读者应该能够理解：

1. 单页应用中路由功能的必要；
2. 如何使用 react-router v4 来实现路由；
3. 动态路由的意义。

留言

评论将在后台进行审核，审核通过后对所有人可见

鲜知

还缺了一种吧，MemoryRouter

▲ 0 评论 7天前

HaoliangWu 前端/全栈工程师 @ 云匠软件

> 如果用静态路由，我们在渲染之前就确定这条路由规则，这样即使用户没有登录，也可以访问 product，我们还不得不在 Product 组件中做用户是否登录的检查。

这里我觉的应该补充一点，就是静态路由一般均会提供 guard，校验的逻辑就不会放到具体的组件中去了。

▲ 3 评论 1月前

大伙子~😫

BrowserHistory也不一定非要服务端渲染吧？配置一下静态服务器的规则就行了，例如nginx的配置:

```
location / {
  try_files $uri /index.html;
}
```

▲ 0 收起评论 1月前

程墨 Hulu

是的，只是如果能够不劳烦nginx就更好了。

1月前

zhangyanling77 前端开发 @ 成都

你这样实际上也是说明需要服务端支持啊

1月前

评论审核通过后显示

评论