Day 02 - React 服务端渲染与 Next.js

SSR 实现过程

客户端基本结构

1. 创建一个空项目，初始化 `package.json`
2. 安装 `react`, `react-dom` 模块
3. 创建基本的项目结构
    1. `src/index.js` → 应用入口
    2. `src/App.js` → 根组件
4. 安装 webpack 相关模块：`webpack`, `webpack-cli`

```
$ npm i webpack webpack-cli --save-dev
```

5. 安装 babel loader 相关模块：`babel-loader`, `@babel/core`, `@babel/preset-react`

```
$ npm i babel-loader @babel/core @babel/preset-react --save-dev
```

6. 配置 webpack 配置文件

```
module.exports = {
  mode: 'none',
  entry: './src/index.js',
  output: {
    filename: 'bundle.js'
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        loader: 'babel-loader'
      }
    ]
  }
}
```

7. 配置 babel 配置

```
{
  "presets": ["@babel/preset-react"]
}
```

8. 运行 webpack 打包

添加客户端路由实现

1. 安装 react-router 模块：`react-router-dom`
2. 添加不同的客户端页面
    1. `src/pages/Home.js`
    2. `src/pages/About.js`
    3. `src/pages/Contact.js`
3. 在根组件中使用 `react-router-dom` 模块实现页面路由切换

```
import React from 'react'
import { BrowserRouter as Router, Switch, Route, Link } from 'react-router-
dom'
import Home from './pages/Home'
import About from './pages/About'
import Contact from './pages/Contact'

export default () => (
  <Router>
    <div>
      <h1>React Application</h1>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
        <Link to="/contact">Contact</Link>
      </nav>
      <Switch>
        <Route path="/" exact component={Home} />
        <Route path="/about" exact component={About} />
        <Route path="/contact" exact component={Contact} />
      </Switch>
    </div>
  </Router>
)
```

服务端渲染实现

1. 添加服务端代码入口文件 `src/server.js`
2. 建议将客户端代码入口文件修改为 `src/client.js`
3. 安装 express 模块（处理服务端请求）
4. 处理请求
5. 使用 webpack 打包服务端代码

注意事项

因为前后端同构，所以很多时候不能使用 Web APIs

Next.js

Next.js 是 React.js 基础之上的一款应用型框架，主要特点：

1. 使用成本低
2. 开箱即用的 SSR

快速上手

创建一个基本的 Next.js 项目只需要简单的几个命令：

```
$ mkdir hello-next
$ cd hello-next
$ yarn init -y
$ yarn add react react-dom next
$ mkdir pages
# run dev
$ yarn next
```

接下来，为了便于使用，我们将 Next.js 所提供的命令行工具添加到 package.json 的 `scripts` 中：

```
{
  "scripts": {
    "dev": "next",
    "build": "next build",
    "start": "next start"
  }
}
```

在这之后我们就可以通过 `yarn dev` 的方式快速启动开发服务器了。

> Next.js 的核心特点就是：每一个 `pages` 下的 js 文件就是一个页面，文件名就是其访问路径 (index 会被省略)

动态页面

Next.js 9 之前的版本实现动态页面相对复杂

对于同一类型的多个页面，一般我们采用动态页面的方式实现，即通过 URL 接受不同参数，根据参数决定页面上所呈现的内容。

在 Next.js 中实现动态页面同样十分简单，只需要在 `Link` 组件的 `href` 属性上添加 `?` 参数，在目标页面中通过 `query` 对象获取。

URL 重写

很多时候 URL 中包含 `?` 参数是很不友好的，相比于 `/post?id=1` 更友好的形式应该是 `/post/1`。

在 Next.js 中实现 URL 的重写更为简单：

```
// 直接通过添加 as 属性指定 url 格式
<Link href={`/post?id=${id}`} as={`/post/${id}`}>
  <a>Post 1</a>
</Link>
```

getInitialProps

页面初始数据供给，此函数接收一个 `context` 参数，返回的对象会被作为 Component 的 `props`。

```
const Post = ({ post }) => {
  return (
    <div>
      <h1>{post.title}</h1>
      <p>{post.content}</p>
    </div>
  )
}
```

```
Post.getInitialProps = async context => {
  const { id } = context.query
  const post = await getPost(id)
  return { post }
}

export default Post
```

https://www.npmjs.com/package/isomorphic-unfetch

- https://juejin.im/entry/5c9cde3e6fb9a070b153fa77
- https://juejin.im/post/5d7deef6e51d453bb13b66cd

License