第12章 Vue服务端渲染

课程提要

- 什么是服务器端渲染
- 为什么使用服务器端渲染
- 利用SSR实现服务器端渲染
- 利用Nuxt.js实现服务器端渲染

12.1 什么是服务器端渲染

使用 Vue.js 构建客户端应用程序时，默认情况下是在浏览器中输出Vue组件，进行生成DOM和操作DOM。而使用服务器端渲染SSR（Server-Side Rendering）可以将同一个组件渲染为服务器端的HTML字符串，然后将它们直接发送到浏览器，最后将静态标记"混合"为客户端上完全交互的应用程序。

这里说的渲染，就是指生成HTML文档的过程，和之前浏览器的CSS+HTML渲染没有关系。简单来说，浏览器端渲染，指的是用JS去生成HTML。

使用Vue.js的服务器端渲染，Vue.js会联络服务器，然后服务器把HTML发送给客户端，因此浏览器可以立即显示页面，SSR对用户体验更好。

12.2 为什么使用服务器端渲染

1、更好的SEO（SEO是由英文Search Engine Optimization缩写而来，中文意译为：搜索引擎优化），由于搜索引擎爬虫抓取工具可以直接查看完全渲染的页面。

2、产生更好的用户体验，更快的首屏渲染更快的内容到达时间（time-to-content），特别是对于缓慢的网络情况或运行缓慢的设备。无需等待所有的 JavaScript 都完成下载并执行，才显示服务器渲染的标记，所以你的用户将会更快速地看到完整渲染的页面。

12.3 SSR

12.3.1 SSR的弊端

- 开发条件受限，Vue组件的某些生命周期钩子函数不能使用；
- 开发环境基于Node.js；
- 会造成服务端更多的负载。

在Node.js中渲染完整的应用程序，显然会比仅仅提供静态文件server更加占用CPU资源，因此如果你在预料在高流量下使用，请准备响应的服务负载，并明智的采用缓存策略。

12.3.2 SSR基本使用

Vue.js本身没有附带SSR，但是有很好的库可以很容易地将SSR添加到我们的Web应用程序中。最受欢迎的两个库是vue-server-renderer和Nuxt.js。

1、vue-server-renderer

（1）安装依赖

新建一个文件夹，安装Vue与SSR依赖包vue-server-renderer。

```
mkdir ssr-test // 新建空文件夹
cd ssr-test // 进入目录
npm init // 初始化，生成 package.json
npm install vue vue-server-renderer express --save// 安装
```

生成如下文件：

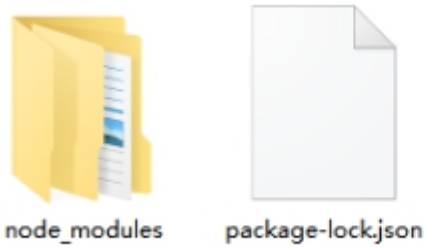

图12-1 生成依赖

（2）创建启动文件

在当前目录下创建一个app.js文件，并编写如下内容：

```
const Vue = require('vue')
const server = require('express')()
const fs = require('fs')
//读取模板
const renderer = require('vue-server-renderer').createRenderer({
  template: fs.readFileSync('./index.html', 'utf-8')
})
//此参数是vue生成Dom之外位置的数据，如vue生成的dom一般位于body中的某个元素容器中
//此数据可在header标签等位置渲染，是renderer.renderToString()的第二个参数
//第一个参数是vue实例，第三个参数是一个回调函数
const context = {
  title: 'Vue-ssr-demo',
  meta: `<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">`
}
server.get('*', (req, res) => {
  //创建vue实例，主要用于替换index.html中body注释地方的内容和index.html中<!--vue-ssr-outlet-->的地方是对应的
  const app = new Vue({
    data: {
      url: req.url,
      data: ['这是一个ssr示例'],
      title: '学习vue-ssr服务端渲染'
    },
    //template中的文本最外层一定要有容器包裹，和vue的组件中是一样的，
```

```
    //只能有一个父级元素，万万不能忘了!
    template: `
            <div>
            <p>{{title}}</p>
            <p v-for='item in data'>{{item}}</p>
            </div>`
  })
  //将Vue实例渲染为字符串（其他的API自己看用法是一样的）
  renderer.renderToString(app, context, (err, html) => {
    if (err) {
      res.status(500).end('err:' + err)
      return
    }
    //将模板发送给浏览器
    res.end(html)
  })
})

server.listen(8080)
```

（3）引入模板

在app.js的平级目录处创建一个index.html，并编写如下内容：

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>{{ title }}</title>
        {{{ meta }}}
    </head>
    <body>
        <!--vue-ssr-outlet--> <!--注意：这是注入应用程序标记的位置。-->
    </body>
</html>
```

（4）启动项目

在命令行中进入项目目录，执行如下命令启动项目：

```
node app.js
```

然后在浏览器中打开http://localhost:8080。

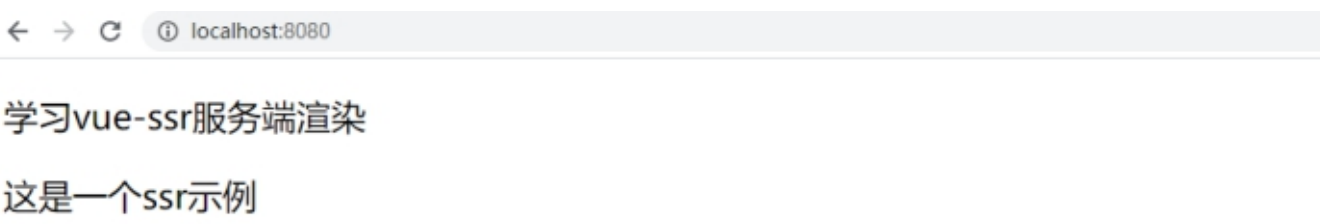

图12-2 ssr服务器渲染页面

然后查看页面源代码，会发现和vue的不一样了。

图12-3 ssr服务器渲染页面源代码

12.4 Nuxt.js

（1）简介

Nuxt.js 是一款建立在Vue之上的一款更高级的js框架，可以帮助我们快速搭建SSR应用。Nuxt.js 主要采用webpack + vue-loader + babel-loader去打包应用。

（2）安装

为了简化用户安装 Nuxt.js，其团队制作了一个脚手架可以很方便创建 Nuxt.js 应用。

```
npx create-nuxt-app <项目名称>
```

使用npx可以免去你在系统全局安装create-nuxt-app这个npm包。

运行创建项目命令，等待一会之后，就会出现和vue-cli脚手架相似的一个过程，如下图：

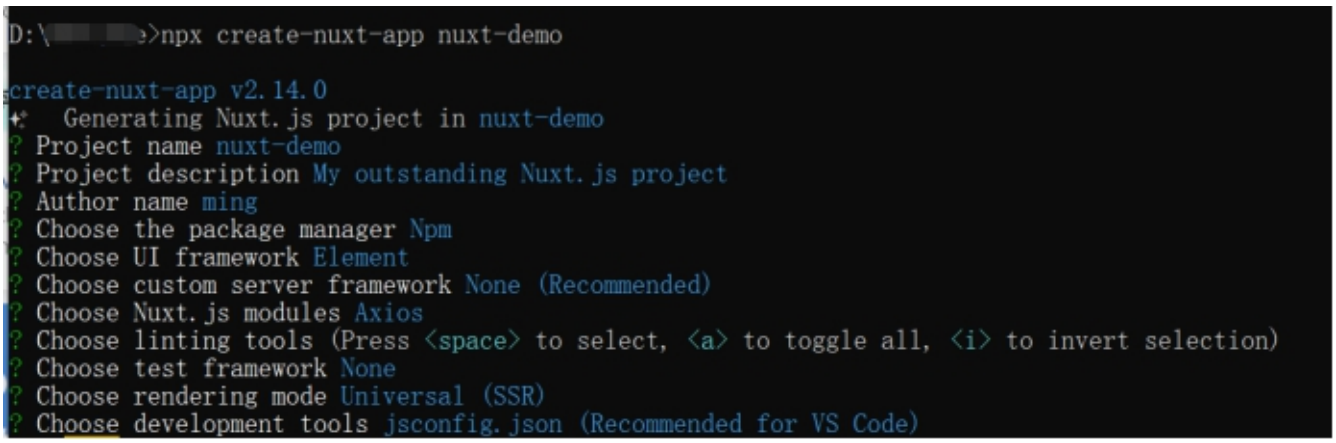

图12-4 初始化nuxt项目

需要进行配置的选项主要包括：

服务端框架、前端 UI 组件库、单元测试框架、构建模式【启用服务端渲染，还是简单的单页应用】、异步请求库、代码检查库、代码格式化库等。

初始化完成后，会出现如下图的提示：

```
Successfully created project nuxt-demo

To get started:

      cd nuxt-demo
      npm run dev

To build & start for production:

      cd nuxt-demo
      npm run build
      npm run start
```

图12-5 初始化nuxt完成

（3）运行项目

根据提示，进入到刚刚初始化好的项目目录中，然后执行npm run dev 命令即可运行项目。

图12-6 nuxt项目默认欢迎页面

启动项目后，通过localhost:3000地址就可以访问Nuxt.js项目了。

（4）目录结构

```
nuxt-demo
.nuxt             Nuxt自动生成，临时用于编辑的文件
assets            资源文件目录
components        全局公用组件目录
layouts           模板文件目录
middleware        中间件目录，在每次进行路由切换的时候，就会调用中间件。
node_modules      项目依赖包目录
pages             页面目录，在此目录放好文件后，nuxt.js会自动配置好路由
plugins           第三方插件目录
static            静态资源目录
store             vuex仓库目录
```

（5）SSR方法

asyncData和fetch方法都是在服务端进行请求数据的方法。

①asyncData

这个方法主要是获取组件内部的数据。一般来说只有子组件间需要传递的数据，我们才会放到 store 中，否则是没有必要的。如果某个数据只在当前组件中使用，那么我们就可以使用该方法来获取。

```
<script>
    export default {
        data() {
            return {
                banners: []
            };
        },
            async asyncData({ $axios }){
                const { data } = await $axios.get('http://localhost:5000/banner');
                return{
                    banners:data.banners
                }
            }
    }
</script>
```

②fetch

该方法主要是获取store中的数据。用法和asyncData差不多，它是一个异步方法。

```
export default {
    async fetch({store, $axios}) {
        const {data} = await $axios.$get('http://localhost:3000/banner')
        store.commit('setBanner', data.banners)
    }
}
```

12.5 课程小结

- 什么是服务器端渲染
- 为什么使用服务器端渲染
- 利用SSR实现服务器端渲染
- 利用Nuxt.js实现服务器端渲染

12.6 实训

完成上面所有案例。