第 16 章 服务器端渲染

人工智能学院 吴承鑫

- 初识服务器端渲染
- webpack 搭建服务器端渲染
- 服务器端渲染的简单实现
- Nuxt.js 服务器端渲染框架

学习目标

了解

1. 了解客户端渲染和服务器端渲染的区别
2. 了解服务器端渲染的优点和不足
3. 掌握服务器端渲染的基本实现方法
4. 了解如何利用 webpack 搭建服务器端渲染

目录

目录

8.4 Nuxt.js 服务器端渲染框架

☞点击查看本节相关知识点

返回目录

1 客户端渲染与服务器端渲染的区别

2 服务器端渲染的注意事项

返回目录

1. 创建 vue-ssr 项目
2. 渲染 vue 实例
3. Express 搭建 SSR
4. Koa 搭建 SSR

返回目录

返回目录

8.1 初识服务器端渲染

1

客户端渲染与服务器端渲染的区别

① 客户端渲染

客户端渲染，即传统的单页面应用（ SPA ）模式， Vue.js 构建的应用程序默认情况下是一个 HTML 模板页面，只有一个 id 为 app 的 根容器，然后通过 webpack 打包生成 css 、 js 等资源文件，浏览器加载、解析来渲染 HTML 。

8.1 初识服务器端渲染

1

客户端渲染与服务器端渲染的区别

在客户端渲染时，一般使用的是 webpack-dev-server 插件，它可以帮助用户自动开启一个服务器端，主要作用是监控代码并打包，也可以配合 webpack-hot-middleware 来进行热更替（ HMR ），这样能提高开发效率。

8.1 初识服务器端渲染

1

客户端渲染与服务器端渲染的区别

提示

在 webpack 中使用模块热替换（ HMR ），能够使得应用在运行时，无须开发者重新运行 npm run dev 命令来刷新页面，便能更新更改的模块，并且将效果及时展示出来，这极大地提高了开发效率。

8.1 初识服务器端渲染

1

客户端渲染与服务器端渲染的区别

② 服务器端渲染

Vue 进行服务器端渲染时，需要利用 Node.js 搭建一个服务器，并添加服务器端渲染的代码逻辑。使用 webpack-dev-middleware 中间件对更改的文件进行监控，使用 webpack-hot-middleware 中间件进行页面的热更新，使用 vue-server-renderer 插件来渲染服务器端打包的 bundle 文件到客户端。

8.1 初识服务器端渲染

1

客户端渲染与服务器端渲染的区别

③ 服务器端渲染的优点

如果网站对 SEO （搜索引擎优化）要求比较高，页面又是通过异步来获取内容，则需要使用服务器渲染来解决此问题。

8.1 初识服务器端渲染

1

客户端渲染与服务器端渲染的区别

服务器端渲染相对于传统的 SPA （单页面应用）来说，主要有以下优势。

- 利于 SEO
- 首屏渲染速度快

8.1 初识服务器端渲染

1

客户端渲染与服务器端渲染的区别

④ 服务器端渲染的不足

虽然服务器端渲染有首屏加载速度快和有利于 SEO 的优点，但是在使用服务器端渲染的时候，还需要注意以下两点事项。

- 服务器端压力增加
- 涉及构建设置和部署的要求

8.1 初识服务器端渲染

2

服务器端渲染的注意事项

① 版本要求

Vue 2.3.0+ 版本的服务器端渲染（ SSR ），要求 vue-server-renderer （服务端渲染插件）的版本要与 Vue 版本相匹配。需要的最低 Vue 相关插件版本如下。

- vue & vue-server-renderer 2.3.0+
- vue-router 2.5.0+
- vue-loader 12.0.0+ & vue-style-loader 3.0.0+

8.1 初识服务器端渲染

2

服务器端渲染的注意事项

② 路由模式

Vue 有两种路由模式，一种是 hash （哈希）模式，在地址栏 URL 中会自带 # 符号，例如， http://localhost/#/login ， #/login 就是 hash 值。需要注意的是，虽然 hash 模式会出现在 URL 中，但不会被包含在 HTTP 请求中，改变 hash 不会重新加载页面。

8.1 初识服务器端渲染

2

服务器端渲染的注意事项

另一种路由模式是 history 模式，与 hash 模式不同的是， URL 中不会自带 # 号，看起来比较美观，如 http://localhost/login 。 history 模式利用 history.pushState API 来完成 URL 跳转而无须重新加载页面。由于 hash 模式的路由提交不到服务器上，因此服务器端渲染的路由需要使用 history 模式。

8.2 服务器端渲染的简单实现

1

创建 vue-ssr 项目

服务端渲染的实现，通常有 3 种方式，第 1 种是手动进行项目的简单搭建，第 2 种是使用 Vue CLI 3 脚手架进行搭建，第 3 种是利用一些成熟框架来搭建（如 Nuxt.js ）。本节讲解第一种方式，手动搭建项目实现简单的服务器端渲染。

8.2 服务器端渲染的简单实现

1

创建 vue-ssr 项目

在项目存储目录下，使用命令行工具创建一个 vue-ssr 项目，执行完命令后，会在 vue-ssr 目录下生成一个 package.json 文件。

```
mkdir vue-ssr
cd vue-ssr
npm init -y
```

8.2 服务器端渲染的简单实现

1

创建 vue-ssr 项目

在 Vue 中使用服务器端渲染，需要借助 Vue 的扩展模块 vue-server-renderer 。

下面在 vue-ssr 项目中使用 npm 来安装 vue-server-renderer ，具体命令如下。

```
npm install vue@2.6.x vue-server-renderer@2.6.x --save
```

vue-server-renderer 是 Vue 中处理服务器加载的一个模块，给 Vue 提供在 Node.js 服务器端渲染的功能。 vue-server-renderer 依赖一些 Node.js 原生模块，所以目前只能在 Node.js 中使用。

8.2 服务器端渲染的简单实现

2

渲染 Vue 实例

将 vue-server-renderer 安装完成后，创建服务器脚本文件 test.js ，实现将 Vue 实例的渲染结果输出到控制台中，具体代码如下。

8.2 服务器端渲染的简单实现

2

渲染 Vue 实例

```
// ① 创建一个 Vue 实例
const Vue = require('vue')
const app = new Vue({
  template: '<div>SSR 的简单使用 </div>'
})
// ② 创建一个 renderer 实例
const renderer = require('vue-server-renderer').createRenderer()
// ③ 将 Vue 实例渲染为 HTML
renderer.renderToString(app, (err, html) => {
  if (err) { throw err }
  console.log(html)
})
```

8.2 服务器端渲染的简单实现

2

渲染 Vue 实例

在命令行中执行 node test.js，可以在控制台中看出此时在 标签中添加了一个特殊的属性 data-server-rendered，该属性是告诉客户端的 Vue 这个标签是由服务器渲染的，运行结果如下所示。

```
<div data-server-rendered="true">SSR 的简单使用 </div>
```

8.2 服务器端渲染的简单实现

3

Express 搭建 SSR

Express 是一个基于 Node.js 平台的 Web 应用开发框架，用来快速开发 Web 应用。下面我们将会讲解如何在 Express 框架中实现 SSR，具体步骤如下。

① 在 vue-ssr 项目中执行如下命令，安装 Express 框架

```
npm install express@4.17.x --save
```

8.2 服务器端渲染的简单实现

3

Express 搭建 SSR

② 创建 template.html 文件，编写模板页面，具体代码如下

```
<!DOCTYPE html>
<html>
  <head><title>Hello</title></head>
  <body>
    <!-- vue-ssr-outlet-->
  </body>
</html>
```

上述代码中的注释部分是 HTML 注入的地方，该注释不能删除，否则会报错。

8.2 服务器端渲染的简单实现

3

Express 搭建 SSR

③ 在项目目录下创建 server.js 文件，具体代码如下

```
// ① 创建 Vue 实例
const Vue = require('vue')
const server = require('express')()
// ② 读取模板
const renderer = require('vue-server-renderer').createRenderer({
  template: require('fs').readFileSync('./template.html', 'utf-8')
})
```

8.2 服务器端渲染的简单实现

3

Express 搭建 SSR

```
// ③ 处理 GET 方式请求
server.get('*', (req, res) => {
  res.set({'Content-Type': 'text/html; charset=utf-8'})
  const vm = new Vue({
    data: {
      title: ' 当前位置 ',
      url: req.url
    },
    template: '<div>{{title}} ： {{url}}</div>',
  })
```

8.2 服务器端渲染的简单实现

3

Express 搭建 SSR

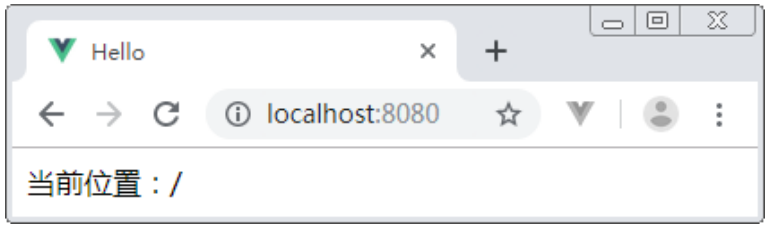
```
// ④ 将 Vue 实例渲染为 HTML 后输出
renderer.renderToString(vm, (err, html) => {
  if (err) {
    res.status(500).end('err: ' + err)
    return
  }
  res.end(html)
 })
})
server.listen(8080, function () {
 console.log('server started at localhost:8080')
})
```

8.2 服务器端渲染的简单实现

3

Express 搭建 SSR

④ 执行 node server.js 命令，启动服务器，在浏览器中访问 http://localhost:8080 ，结果如下图所示。



Express 搭建 SSR

浏览器输出结果

8.2 服务器端渲染的简单实现

4

Koa 搭建 SSR

Koa 是一个基于 Node.js 平台的 Web 开发框架，致力于成为 Web 应用和 API 开发领域更富有表现力的技术框架。 Koa 能帮助开发者快速地编写服务器端应用程序，通过 async 函数很好地处理异步的逻辑，有力地增强错误处理。下面我们讲解如何在 Koa 中搭建 SSR 。

① 在 vue-ssr 项目中安装 Koa ，具体命令如下

```
npm install koa@2.8.x --save
```

8.2 服务器端渲染的简单实现

4

Koa 搭建 SSR

② 创建 koa.js 文件，编写服务器端逻辑代码，具体代码如下

```
// ① 创建 vue 实例
const Vue = require('vue')
const Koa = require('koa')
const app = new Koa()
// ② 读取模板
const renderer = require('vue-server-
renderer').createRenderer({
  template: require('fs').readFileSync('./template.html', 'utf-8')
})
```

8.2 服务器端渲染的简单实现

4

Koa 搭建 SSR

```
// ③ 添加一个中间件来处理所有请求
app.use(async (ctx, next) => {
  const vm = new Vue({
    data: {
      title: ' 当前位置 ',
      url: ctx.url   //  这里的 ctx.url 相当于 ctx.request.url
    },
    template: '<div>{{title}} ： {{url}}</div>'
  })
```

8.2 服务器端渲染的简单实现

4

Koa 搭建 SSR

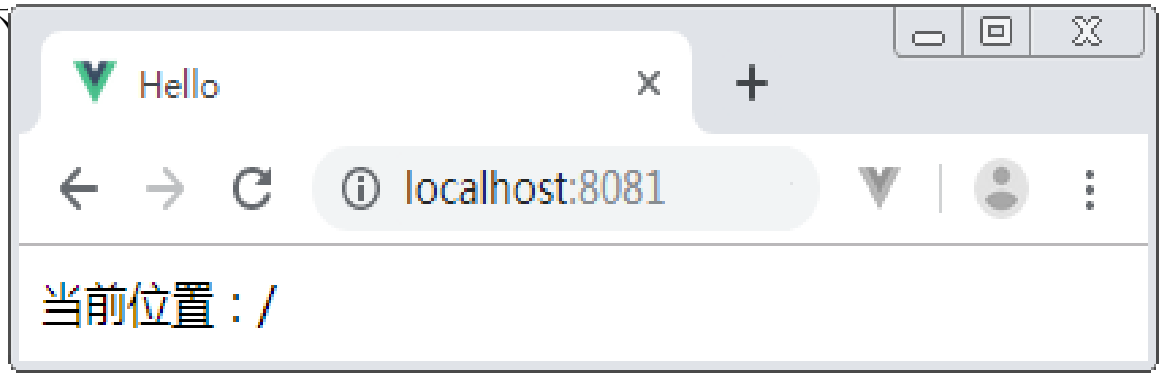
```
// ④ 将 Vue 实例渲染为 HTML 后输出
renderer.renderToString(vm, (err, html) => {
  if (err) {
    ctx.res.status(500).end('err: ' + err)
    return
  }
  ctx.body = html
 })
})
app.listen(8081, function () {
 console.log('server started at localhost:8081')
})
```

8.2 服务器端渲染的简单实现

4

Koa 搭建 SSR

③ 执行 node koa.js 命令启动项目，在浏览器中访问 http://localhost:8081 ，结果如下图所示



Koa 搭建 SSR

8.3 webpack 搭建服务器端渲染

1

基本流程

本节将使用 Vue CLI 3+webpack 来搭建服务端渲染，这种方式相对上一节介绍的方式来说比较难， Vue 在官方文档中进行了较深入的介绍，对于初学者来说可能并不容易理解，适合具有一定技术功底的读者阅读。如果只是利用服务器端渲染来快速搭建项目，读者可以选择学习 8.4 节讲解的 Nuxt.js 框架，用这个框架可以轻松实现服务器端渲染。

8.3 webpack 搭建服务器端渲染

1

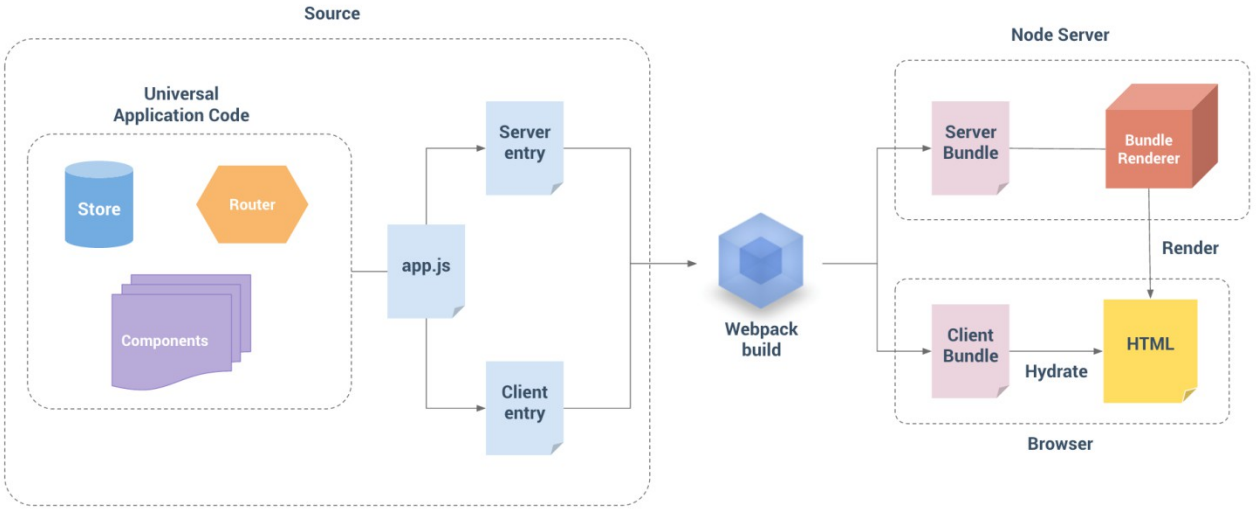
基本流程

webpack 服务器端渲染需要使用 entry-server.js 和 entry-client.js 两个入口文件，两者通过打包生成两份 bundle 文件。其中，通过 entry-server.js 打包的代码是运行在服务器端，而通过 entry-client.js 打包的代码运行在客户端。

8.3 webpack 搭建服务器端渲染

1 基本流程

在 Vue 官方文档中提供了 webpack 服务器端渲染的流程图，如下图所示。



服务器端渲染流程图

8.3 webpack 搭建服务器端渲染

1

基本流程

上图中， Source 表示 src 目录下的源代码文件， Node Server 表示 Node 服务器，Browser 表示浏览器， Universal Application Code 是服务器端和浏览器端共用的代码。

Server entry 和 Client entry 分别包含了服务器端应用程序（仅运行在服务器）和客户端应用程序（仅运行在浏览器），对应 entry-server.js 和 entry-client.js 两个入口文件， webpack 将这两个入口文件分别打包成给服务器端用的 Server Bundle 和给客户端用的 Client Bundle 。 app.js 是通用入口文件，它用来编写两个入口文件中的相同部分的代码。

8.3 webpack 搭建服务器端渲染

1

基本流程

当服务器接收到了来自客户端的请求之后，会创建一个 Bundle Renderer 渲染器，这个渲染器会读取 Server Bundle 文件，并且执行它的代码，然后发送一个生成好的 HTML 到浏览器。

8.3 webpack 搭建服务器端渲染

2

项目搭建

① 创建项目

如果安装了 @vue/cli 脚手架就不需要再安装了，如果没有全局安装 @vue/cli 脚手架的在这里需要进行安装，用于搭建开发模板。 通过 npm 全局安装 @vue/cli 脚手架，命令如下。

```
npm install @vue/cli@3.10 -g
```

8.3 webpack 搭建服务器端渲染

2

项目搭建

在项目存储目录下创建一个名称为 ssr-project 的项目，命令如下。

```
vue create ssr-project
```

执行完上述命令，会进入一个交互界面，选择 default 默认即可。

8.3 webpack 搭建服务器端渲染

2

项目搭建

在 ssr-project 项目中安装依赖，具体命令如下

```
cd ssr-project
npm install vue-router@3.1.x koa@2.8.x vue-server-renderer@2.6.x --save
```

8.3 webpack 搭建服务器端渲染

2

项目搭建

② 配置 vue.config.js

```
const VueSSRServerPlugin = require('vue-server-renderer/server-plugin')
module.exports = {
  configureWebpack: () => ({
    entry: './src/entry-server.js',
    devtool: 'source-map', //  对 bundle renderer 提供 source map 支持
    target: 'node',
    output: { libraryTarget: 'commonjs2  ' },
    plugins: [ new VueSSRServerPlugin() ]
  }),
  chainWebpack: config => { }
}
```

8.3 webpack 搭建服务器端渲染

2

项目搭建

③ 编写项目代码

首先删除 src 目录中所有的文件，然后重新创建项目文件，然后创建 src\app.js 文件，具体代码如下。

```
import Vue from 'vue'
import App from './App.vue'
import { createRouter } from './router'
Vue.config.productionTip = false
```

8.3 webpack 搭建服务器端渲染

2

项目搭建

```
export function createApp() {   //  导出 createApp() 函数
  const router = createRouter()  //  创建 router 实例
  const app = new Vue({
    router,
    render: h => h(App)
  })
  return { app, router }
}
```

8.3 webpack 搭建服务器端渲染

2

项目搭建

创建 src\router.js 文件，具体代码如下。

```
import Vue from 'vue'
import Router from 'vue-router'
Vue.use(Router)
export function createRouter () {
  return new Router({
    mode: 'history',
    routes: [
      {path: '/',name: 'home',component: () => import('./App.vue')}
    ]
  }) }
```

8.3 webpack 搭建服务器端渲染

2

项目搭建

创建 App.vue 文件，具体代码如下。

```
<template>
  <div id="app">test</div>
</template>
<script>
export default {
  name: 'app'
}
</script>
```

8.3 webpack 搭建服务器端渲染

2

项目搭建

创建 entry-server.js 文件，该文件是服务器端打包入口文件，在 Vue 官方文档中提供了该文件的示例，可以直接复制到项目中使用。

8.3 webpack 搭建服务器端渲染

2

项目搭建

```
import { createApp } from './app' //  导入 createApp 函数
export default context => {
  return new Promise((resolve, reject) => {
    const { app, router } = createApp()
    router.push(context.url)
    router.onReady(() => {
      const matchedComponents = router.getMatchedComponents()
      if (!matchedComponents.length) {
        return reject(new Error('no components matched'))
      }
      resolve(app)
    }, reject) }) }
```

8.3 webpack 搭建服务器端渲染

2

项目搭建

④ 生成 vue-ssr-server-bundle.json

修改 package.json 文件，在 scripts 脚本命令中添加如下内容。

```
"build:server": "vue-cli-service build --mode server"
```

执行如下命令，生成 vue-ssr-server-bundle.json 文件。

```
npm run build:server
```

上述命令执行后，在 dist 目录中可以看到生成后的 vue-ssr-server-bundle.json 文件。

8.3 webpack 搭建服务器端渲染

2

项目搭建

⑤ 编写服务器端代码

服务器端代码主要是通过 Koa 、 vue-server-renderer 来实现，这部分代码可以参考官方文档中的介绍。创建 server.js 文件，具体代码如下。

8.3 webpack 搭建服务器端渲染

2

项目搭建

```
const Koa = require('koa')
const app = new Koa()
const bundle = require('./dist/vue-ssr-server-bundle.json')
const { createBundleRenderer } = require('vue-server-renderer')
const renderer = createBundleRenderer(bundle, {
  template: require('fs').readFileSync('./template.html', 'utf-8'),
})
function renderToString (context) {... //  接下来添加该处代码  }
app.use(async (ctx, next) => {{... //  接下来添加该处代码 }
app.listen(8080, function() {{... //  接下来添加该处代码 }
```

8.3 webpack 搭建服务器端渲染

2

项目搭建

renderToString() 函数用于将 Vue 实例渲染成字符串。

```
function renderToString (context) {
  return new Promise((resolve, reject) => {
    renderer.renderToString(context, (err, html) => {
      err ? reject(err) : resolve(html)
    })
  })
}
```

8.3 webpack 搭建服务器端渲染

2

项目搭建

```
app.use(async (ctx, next) => {
  const context = {
    title: 'ssr project',
    url: ctx.url
  }
  const html = await renderToString(context)
  ctx.body = html
})
app.listen(8080, function() {
  console.log('server started at localhost:8080')
})
```

8.3 webpack 搭建服务器端渲染

2

项目搭建

创建 template.html 文件，具体代码如下。

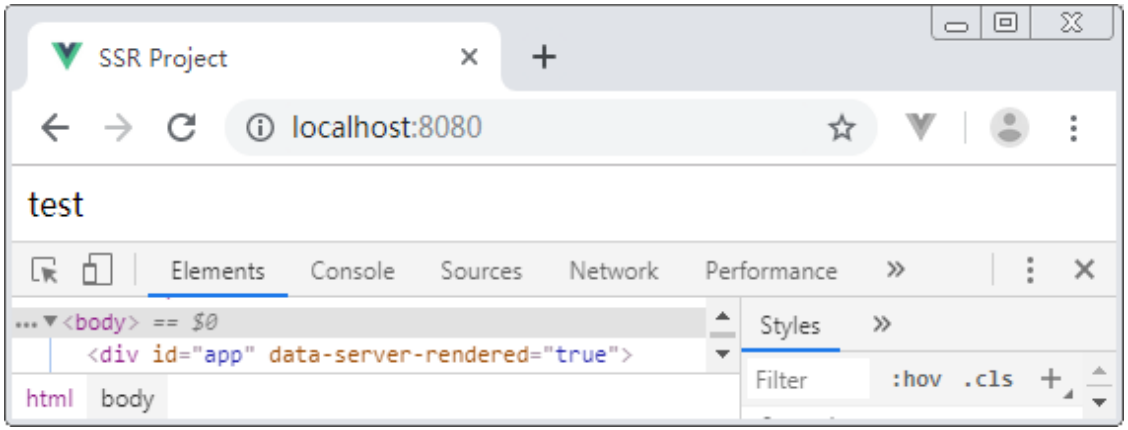
```
<!DOCTYPE html>
<html>
  <head><title>SSR Project</title></head>
  <body>
    <!--vue-ssr-outlet-->
  </body>
</html>
```

8.3 webpack 搭建服务器端渲染

2

项目搭建

执行 node server.js 命令，启动服务器，通过浏览器访问 http://localhost:8080，运行结果如下图所示。



服务器端渲染结果

8.4 Nuxt.js 服务器端渲染框架

1

创建 Nuxt.js 项目

Nuxt.js 是一个基于 Vue.js 的轻量级应用框架，可用来创建服务端渲染应用，也可充当静态站点引擎生成静态站点应用，具有优雅的代码结构分层和热加载等特性。本节将会讲解如何利用 Nuxt.js 创建服务器端渲染项目。

8.4 Nuxt.js 服务器端渲染框架

1

创建 Nuxt.js 项目

Nuxt.js 提供了利用 vue.js 开发服务端渲染的应用所需要的各种配置，为了快速入门， Nuxt.js 团队创建了脚手架工具 create-nuxt-app ，具体使用步骤如下。

8.4 Nuxt.js 服务器端渲染框架

1

创建 Nuxt.js 项目

① 确保已经安装好了 node.js 和 vue-cli 脚手架。

② 全局安装 create-nuxt-app 脚手架工具。

```
npm install create-nuxt-app@2.9.x -g
```

③ 在项目存储目录下执行以下命令，创建项目。

```
npx create-nuxt-app my-nuxt-dome
```

8.4 Nuxt.js 服务器端渲染框架

1

创建 Nuxt.js 项目

④ 在创建项目过程中，会询问选择哪个包管理器，在这里选择使用 npm 。

```
? Choose the package manager (Use arrow keys)
  Yarn
> Npm
```

8.4 Nuxt.js 服务器端渲染框架

1

创建 Nuxt.js 项目

⑤ 当询问选择哪个渲染模式时，在这里选择使用 SSR 。

```
? Choose rendering mode (Use arrow keys)
> Universal (SSR)
  Single Page App
```

⑥ 安装配置完成后，启动项目，命令如下。

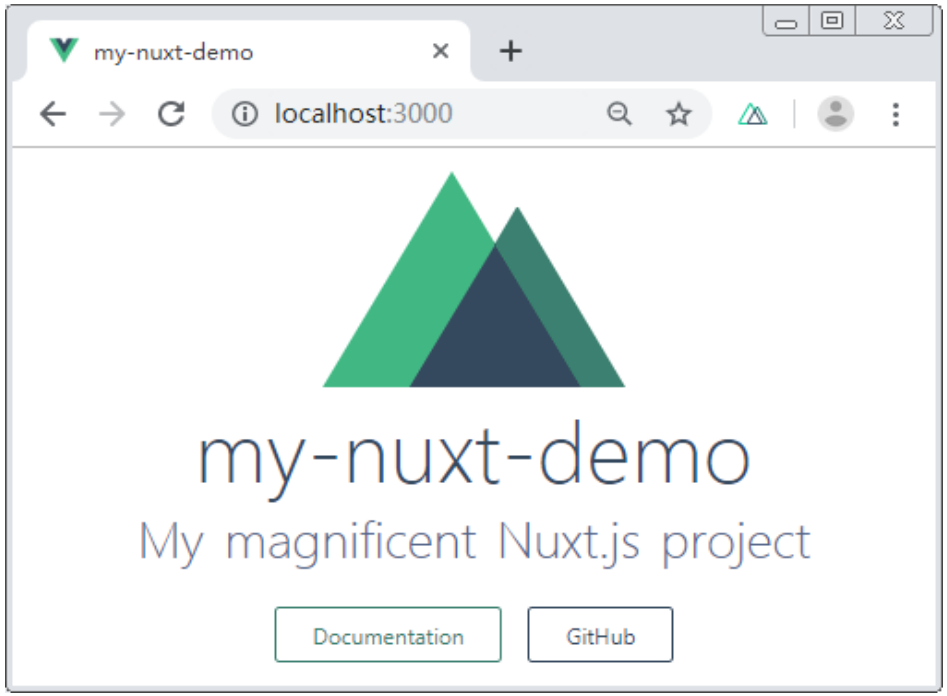
```
cd my-nuxt-demo
npm run dev
```

8.4 Nuxt.js 服务器端渲染框架

1

创建 Nuxt.js 项目

⑦ 通过浏览器访问： http://localhost:3000/ ，运行结果如下图所示。



my-nuxt-demo 项目

8.4 Nuxt.js 服务器端渲染框架

1

创建 Nuxt.js 项目

接下来对 my-nuxt-demo 项目中的关键文件进行说明，详细描述如下表所示。

文件	说明
assets	存放待编译的静态资源，如 Less 、 Sass
static	存放不需要 webpack 编译的静态文件，服务器启动的时候，该目录下的文件会映射至应用的根路径“ /” 下
components	存放自己编写的 Vue 组件
layouts	布局目录，用于存放应用的布局组件
middleware	用于存放中间件
pages	用于存放应用的路由及视图， Nuxt.js 会根据该目录结构自动生成对应的路由配置
plugins	用于存放需要在根 Vue 应用实例化之前需要运行的 JavaScript 插件
nuxt.config.js	用于存放 Nuxt.js 应用的自定义配置，以便覆盖默认配置

8.4 Nuxt.js 服务器端渲染框架

2

页面和路由

在项目中， pages 目录用来存放应用的路由及视图，目前该目录下有两个文件，分别是 index.vue 和 README.md ，当直接访问根路径“ /” 的时候，默认打开的是 index.vue 文件。 Nuxt.js 会根据目录结构自动生成对应的路由配置，将请求路径和 pages 目录下的文件名映射，例如，访问“ /test” 就表示访问 test.vue 文件，如果文件不存在，就会提示“ This page could not be found”（该页面未找到）错误。

8.4 Nuxt.js 服务器端渲染框架

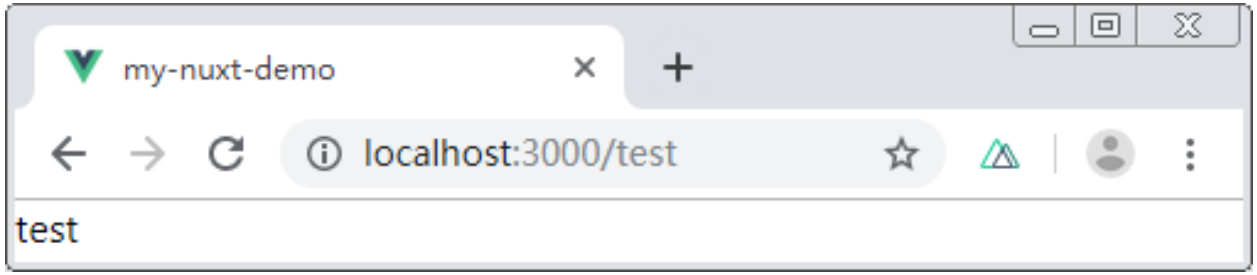
2

页面和路由

接下来我们创建 pages\test.vue 文件，具体代码如下。

```
<template>
  <div>test</div>
</template>
```

通过浏览器访问 http://localhost:3000/test 地址，运行结果如下图所示。



访问 test 组件

8.4 Nuxt.js 服务器端渲染框架

2

页面和路由

pages 目录下的 vue 文件也可以放在子目录中，在访问的时候也要加上子目录的路径。例如，创建 pages\sub\test.vue 文件，具体代码如下。

```
<template>
  <div>sub/test</div>
</template>
```

在浏览器中可以通过 http://localhost:3000/sub/test 地址，来访问 pages\sub\test.vue 文件。

8.4 Nuxt.js 服务器端渲染框架

2

页面和路由

Nuxt.js 提供了非常方便的自动路由机制，当它检测到 pages 目录下的文件发生变更时，就会自动更新路由。通过查看“ .nuxt\router.js” 路由文件，可以看到 Nuxt.js 自动生成的代码，如下所示。

8.4 Nuxt.js 服务器端渲染框架

2

页面和路由

```
routes: [{
  path: "/test",
  component: _5c170d74,
  name: "test"
}, {
  path: "/sub/test",
  component: _c12b6364,
  name: "sub-test"
}, {
  path: "/",
  component: _f51f2a64,
  name: "index"
}],
```

8.4 Nuxt.js 服务器端渲染框架

3

页面跳转

Nuxt.js 中使用 <nuxt-link> 组件来完成页面中路由的跳转，它类似于 Vue 中的路由组件 <router-link> ，它们具有相同的属性，并且使用方式也相同。需要注意的是，在 Nuxt.js 项目中不要直接使用 <a> 标签来进行页面的跳转，因为 <a> 标签是重新获取一个新的页面，而 <nuxt-link> 更符合 SPA 的开发模式。下面我们介绍在 Nuxt.js 中页面跳转的两种方式。

8.4 Nuxt.js 服务器端渲染框架

3

页面跳转

① 声明式路由

以 pages\test.vue 页面为例，在页面中使用 <nuxt-link> 完成路由跳转，具体代码如下。

```
<template>
  <div>
    <nuxt-link to="/sub/test"> 跳转到 sub/test</nuxt-link>
  </div>
</template>
```

8.4 Nuxt.js 服务器端渲染框架

3

页面跳转

② 编程式路由

编程式路由，就是在 JavaScript 代码中实现路由的跳转。以 pages\sub\test.vue 页面为例，示例代码如下。

html 代码

```
<template>
  <div>
    <button @click="jumpTo"> 跳转到 test</button>
    <div>sub/test</div>
  </div>
</template>
```

8.4 Nuxt.js 服务器端渲染框架

3

页面跳转

js 代码

```
<script>
export default {
  methods: {
    jumpTo () {
      this.$router.push('/test')
      //  使用 this.$router.push('/test') 导航到 test 页面
    }
  }
}
</script>
```

本章小结

本章主要讲解了服务器端渲染的概念及使用、客户端渲染和服务端渲染的区别等。在讲解了服务器端渲染的基本知识后，通过案例的形式讲解了如何手动搭建服务器端渲染项目。大家应重点理解服务器端渲染的概念，理解服务器端渲染的优缺点，能够利用服务器端渲染技术完成项目开发中的需求。

Thank You!