

Node服务器端渲染(SSR)

刘军 liujun

目录

content



1 邂逅SPA和SSR

2 Node服务搭建

3 Vue3+SSR搭建

4 SSR + Hydration

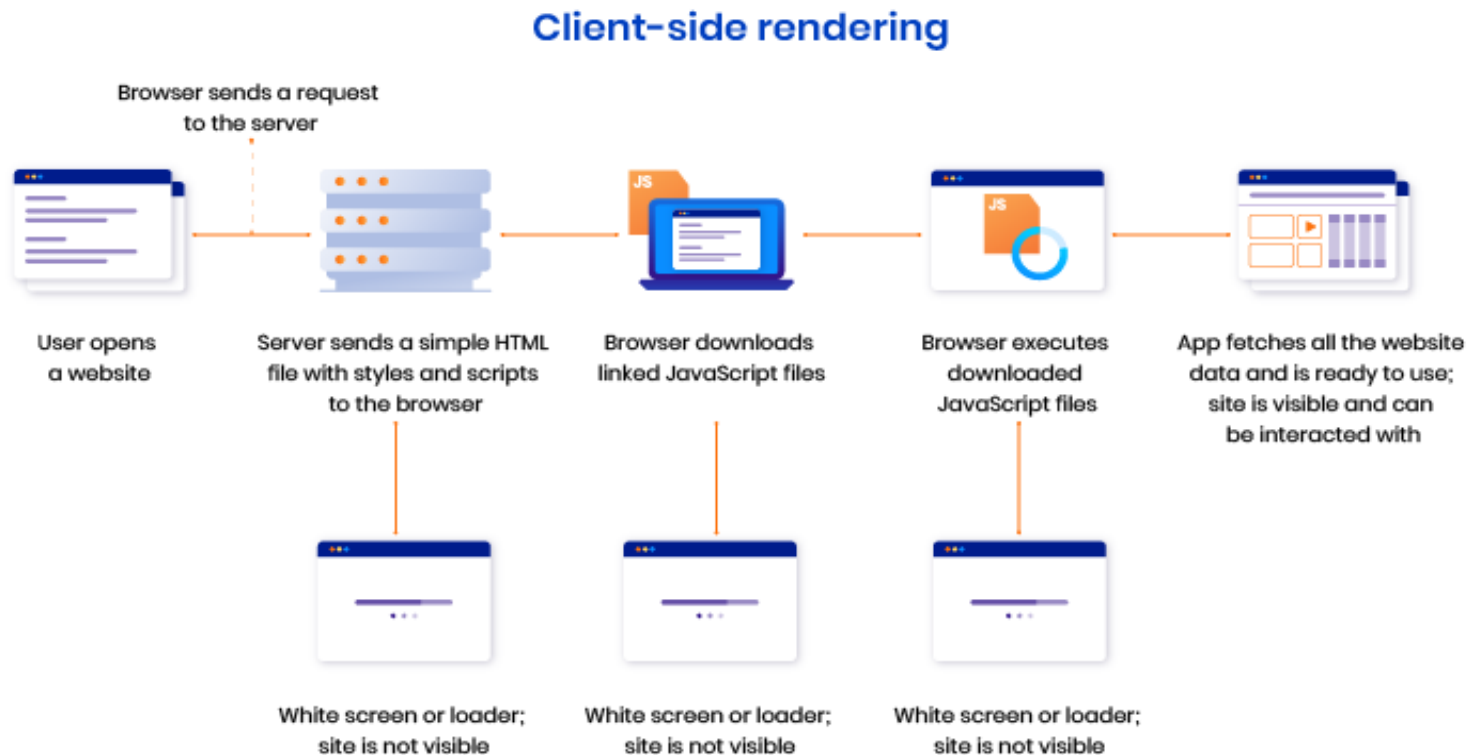
5 Vue SSR + Router

6 Vue SSR + Pinia

单页应用程序 (SPA)

- 单页应用程序 (SPA) 全称是：[Single-page application](#)，SPA应用是在客户端呈现的（术语称：CRS）。
 - SPA应用默认只返回一个空HTML页面，如：body只有<div id= "app" ></div>。
 - 而整个应用程序的内容都是通过 Javascript 动态加载，包括应用程序的逻辑、UI 以及与服务器通信相关的所有数据。
 - 构建 SPA 应用常见的库和框架有：[React](#)、[AngularJS](#)、[Vue.js](#) 等。

■ 客户端渲染原理



■ SPA的优点

□ 只需加载一次

- ✓ SPA应用程序只需要在第一次请求时加载页面，页面切换不需重新加载，而传统的Web应用程序必须在每次请求时都得加载页面，需要花费更多时间。因此，SPA页面加载速度要比传统 Web 应用程序更快。

□ 更好的用户体验

- ✓ SPA 提供类似于桌面或移动应用程序的体验。用户切换页面不必重新加载新页面
- ✓ 切换页面只是内容发生了变化，页面并没有重新加载，从而使体验变得更加流畅

□ 可轻松的构建功能丰富的Web应用程序

■ SPA的缺点

□ SPA应用默认只返回一个空HTML页面，不利于SEO（search engine optimization）

□ 首屏加载的资源过大时，一样会影响首屏的渲染

□ 也不利于构建复杂的项目，复杂 Web 应用程序的大文件可能变得难以维护

爬虫-工作流程

■ Google 爬虫的工作流程分为 3 个阶段，并非每个网页都会经历这 3 个阶段：

□ 抓取：

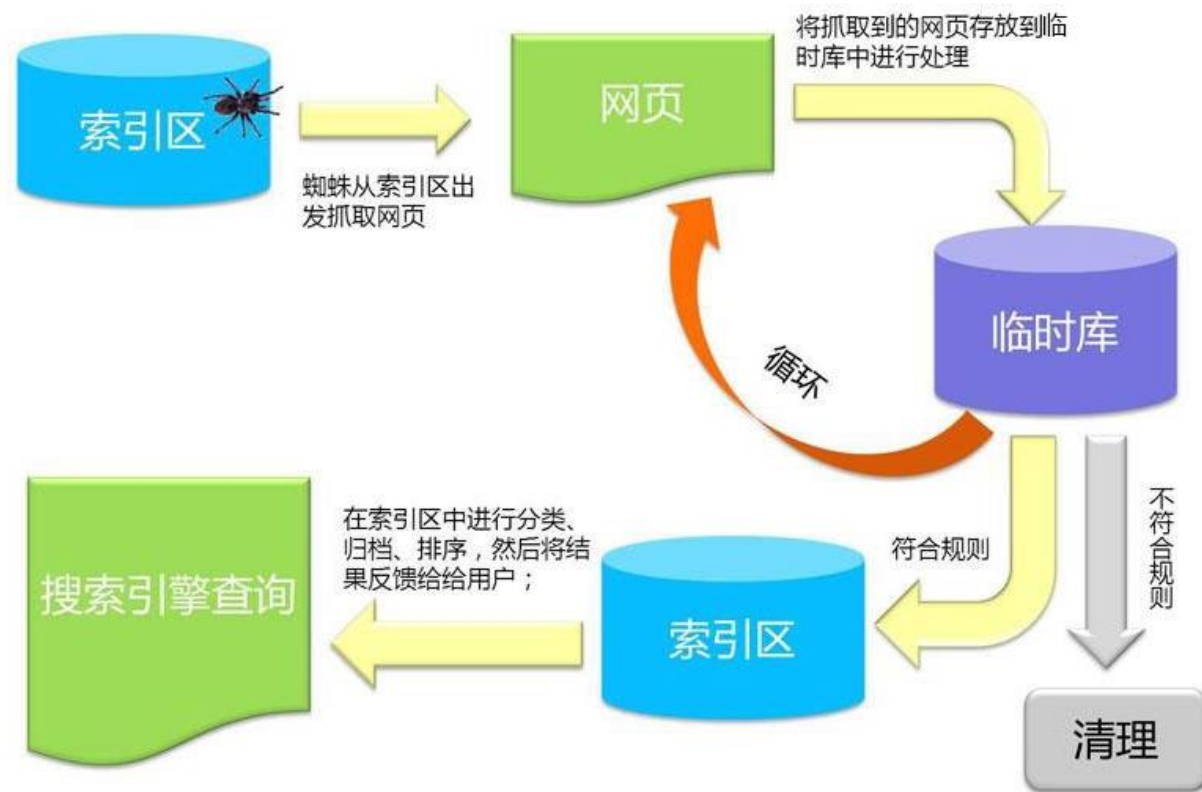
- ✓ 爬虫（也称蜘蛛），从互联网上发现各类网页，网页中的外部连接也会被发现。
- ✓ 抓取数以十亿被发现网页的内容，如：文本、图片和视频

□ 索引编制：

- ✓ 爬虫程序会分析网页上的文本、图片和视频文件
- ✓ 并将信息存储在大型数据库（索引区）中。
- ✓ 例如 <title> 元素 和 Alt 属性、图片、视频等
- ✓ 爬虫会对内容类似的网页归类分组
- ✓ 不符合规则内容和网站会被清理
 - 如：禁止访问 或 需要权限网站等等

□ 呈现搜索结果：

- ✓ 当用户在 Google 中搜索时，搜索引擎会根据内容的类型，选择一组网页中最具代表性的网页进行呈现



搜索引擎的优化 (SEO)

■ 语义性HTML标记

- 标题用<h1>，一个页面只有一个；副标题用<h2>到<h6>。
- 不要过度使用h标签，多次使用不会增加 SEO (search engine optimization) 。
- 段落用<p>，列表用，并且li只放在 ul 中 等等。

■ 每个页面需包含：标题 + 内部链接

- 每个页面对应的title，同一网站所有页面都有内链可以指向首页

■ 确保链接可供抓取，如右图所示：

■ meta标签优化：设置 description keywords 等

■ 文本标记和img：

- 比如和加粗文本的标签，爬虫也会关注到该内容
- img标签添加 alt 属，图片加载失败，爬虫会取alt内容。

■ robots.txt 文件：规定爬虫可访问您网站上的哪些网址。

■ sitemap.xml站点地图：在站点地图列出所有网页，确保爬虫不会漏掉某些网页

■ 更多查看：<https://developers.google.com/search/docs/crawling-indexing/valid-page-metadata>

HTML标签	含义	SEO权重
<title>	标题	10分
<a>	内部链接文字	10分
< h1 >/< h2 >	标题，其实是H1 ~ H6，不过H1和H2是最重要的，一般一个页面最好一个，不要过多使用	5分
<p>	每段首句	5分
<p>	每句开头	1.5分
/	加粗	1分
	title属性	1分
	img alt标记	0.5分
<meta description=''>	meta标签的网站description	0.5分
<meta keywords=''>	meta标签的网站keywords	0.05分

✓

✓

✗

✗

✗

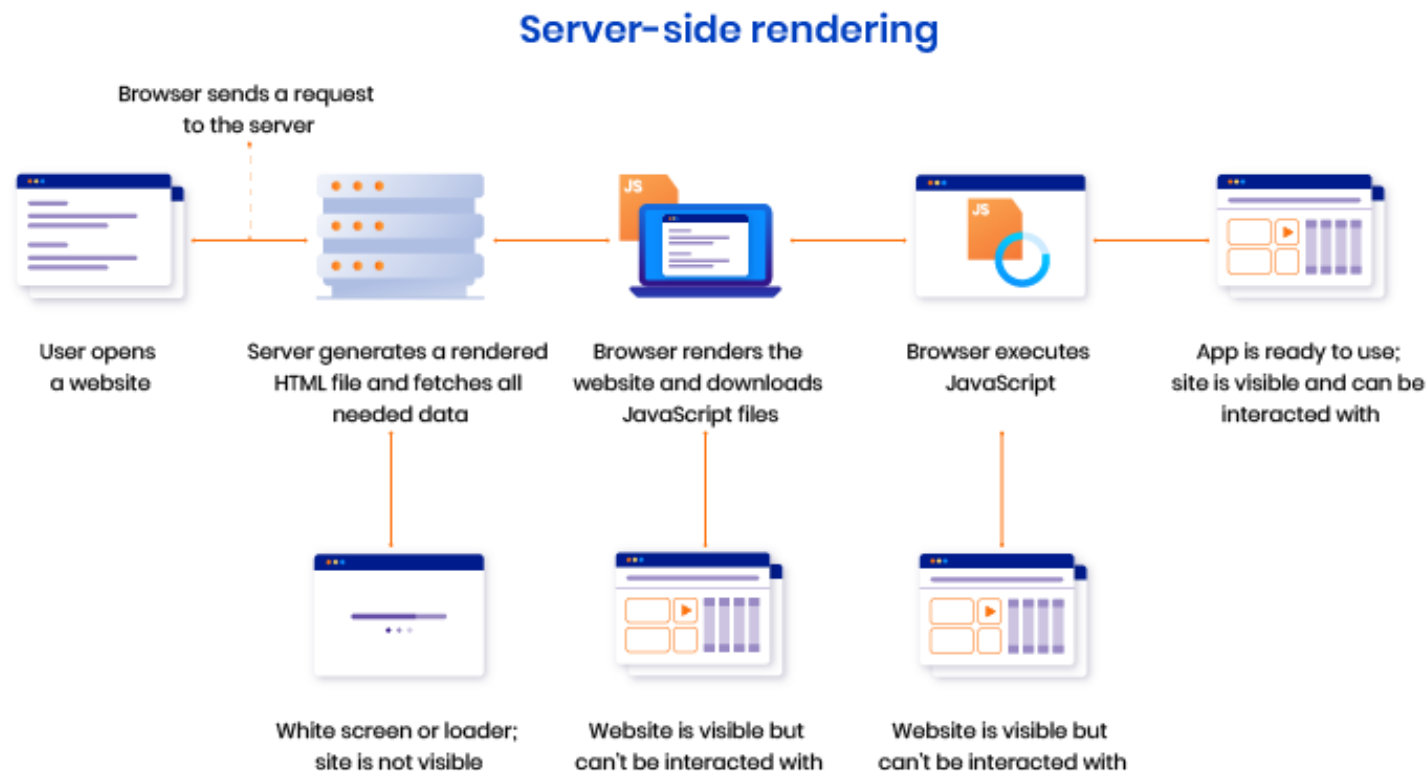
静态站点生成 (SSG)

- 静态站点生成(SSG) 全称是：Static Site Generate，是预先生成好的静态网站。
 - SSG 应用一般在构建阶段就确定了网站的内容。
 - 如果网站的内容需要更新了，那必须得重新再次构建和部署。
 - 构建 SSG 应用常见的库和框架有：Vue Nuxt、React Next.js 等。
- SSG的优点：
 - 访问速度非常快，因为每个页面都是在构建阶段就已经提前生成好了。
 - 直接给浏览器返回静态的HTML，**也有利于SEO**
 - SSG应用依然保留了SPA应用的特性，比如：前端路由、响应式数据、虚拟DOM等
- SSG的缺点：
 - 页面都是静态，**不利于展示实时性的内容，实时性的更适合SSR。**
 - 如果站点内容更新了，那必须得**重新再次构建和部署。**

服务器端渲染 (SSR)

- 服务器端渲染全称是：Server Side Render，在服务器端渲染页面，并将渲染好HTML返回给浏览器呈现。
 - SSR应用的页面是在服务端渲染的，用户每请求一个SSR页面都会先在服务端进行渲染，然后将渲染好的页面，返回给浏览器呈现。
 - 构建 SSR 应用常见的库和框架有：Vue Nuxt、React Next.js 等（SSR应用也称同构应用）。

服务器端渲染原理



■ SSR的优点

□ 更快的首屏渲染速度

- ✓ 浏览器显示静态页面的内容要比 JavaScript 动态生成的内容快得多。
- ✓ 当用户访问首页时可立即返回静态页面内容，而不需要等待浏览器先加载完整个应用程序。

□ 更好的SEO

- ✓ 爬虫是最擅长爬取静态的HTML页面，服务器端直接返回一个静态的HTML给浏览器。
- ✓ 这样有利于爬虫快速获取网页内容，并编入索引，有利于SEO。

□ SSR应用程序在 Hydration 之后依然可以保留 Web 应用程序的交互性。比如：前端路由、响应式数据、虚拟DOM等。

■ SSR的缺点

□ SSR 通常需要对服务器进行更多 API 调用，以及在服务器端渲染需要消耗更多的服务器资源，成本高。

□ 增加了一定的开发成本，用户需要关心哪些代码是运行在服务器端，哪些代码是运行在浏览器端。

□ SSR 配置站点的缓存通常会比SPA站点要复杂一点。

■ SSR的解决方案：

- ❑ 方案一： php、jsp ...
- ❑ 方案二： 从零搭建 SSR 项目（ Node+webpack+Vue/React ）
- ❑ 方案三： 直接使用流行的框架（推荐）
 - ✓ React : Next.js
 - ✓ Vue3 : Nuxt3 || Vue2 : Nuxt.js
 - ✓ Angular : Angular Universal

■ SSR应用场景非常广阔，比如：

- ❑ SaaS产品，如：电子邮件网站、在线游戏、客户关系管理系统（CRM）、采购系统等
- ❑ 门户网站、电子商务、零售网站
- ❑ 单个页面、静态网站、文档类网站
- ❑ 等等

Server Side Rendering (Choose One)

You can also run frameworks like React and Vue on the server. There are advantages to this such as better SEO, easy routing, etc

✦ [Next.js](#) (React)

✦ [Nuxt.js](#) (Vue)

✦ [Angular Universal](#) (Angular)

✦ [Sapper](#) (Svelte)



邂逅Vue3 + SSR

- Vue除了支持开发SPA应用之外，其实也是支持开发SSR应用的。
- 在Vue中创建SSR应用，需要调用createSSRApp函数，而不是createApp
 - createApp：创建应用，直接挂载到页面上
 - createSSRApp：创建应用，是在激活的模式下挂载应用
- 服务端用 @vue/server-renderer 包中的 renderToString 来进行渲染。

// app.js (在服务器和客户端之间共享的应用实例)

```
import { createSSRApp } from "vue";

export function createApp() {
  // 为了在激活模式下挂载应用，我们应该使用 createSSRApp() 而不是 createApp()
  const app = createSSRApp({
    data: () => ({ count: 1 }),
    template: `<button @click="count++">{{ count }}</button>`,
  });
  return app;
}
```

```
import { renderToString } from "@vue/server-renderer";
import { createApp } from "../app.js";
const app = createApp();
renderToString(app)
```

Node Server 搭建

■ 需安装的依赖项：

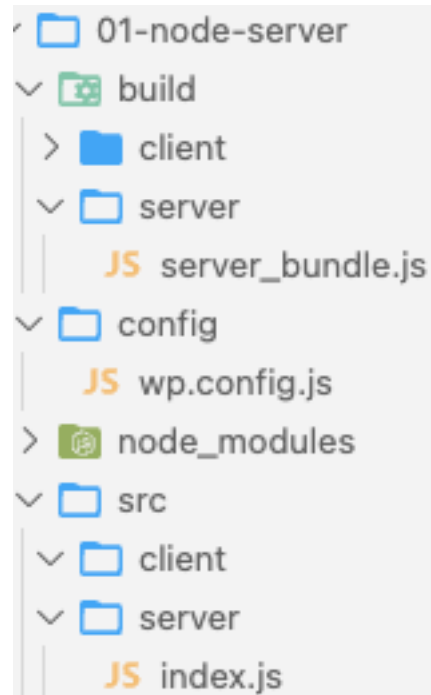
- ❑ npm i express
- ❑ npm i -D nodemon
- ❑ npm i -D webpack webpack-cli webpack-node-externals

■ nodemon:

- ❑ 启动Node程序时并监听文件的变化，变化即刷新

■ webpack-node-externals:

- ❑ 排除掉 node_modules 中所有的模块



```
01-node-server
├── build
│   ├── client
│   └── server
│       ├── JS server_bundle.js
│       └── config
│           ├── JS wp.config.js
│           └── node_modules
├── src
│   ├── client
│   └── server
│       └── JS index.js
```

Vue3 + SSR 搭建

■ 需安装的依赖项:

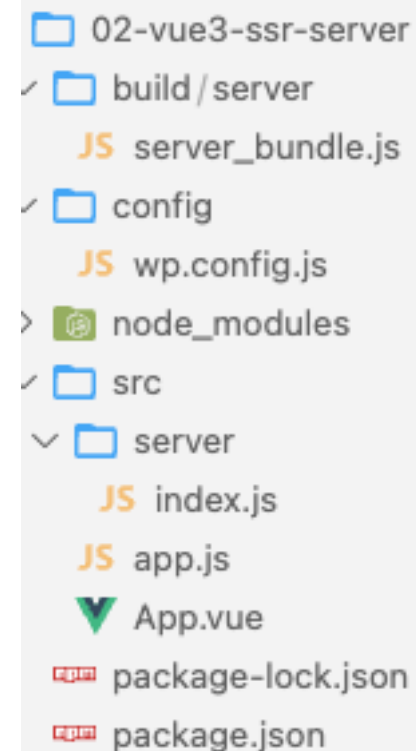
- ❑ npm i express
- ❑ npm i -D nodemon
- ❑ npm i vue
- ❑ npm i -D vue-loader
- ❑ npm i -D babel-loader @babel/preset-env
- ❑ npm i -D webpack webpack-cli
- ❑ npm i -D webpack-merge webpack-node-externals

■ vue-loader: 加载.vue文件

■ webpack-merge: 用来合并webpack配置

■ babel-loader、@babel/preset-env

- ❑ 加载JS文件, 转换新语法



```
02-vue3-ssr-server
├── build/server
│   └── JS server_bundle.js
├── config
│   └── JS wp.config.js
├── node_modules
├── src
├── server
│   ├── JS index.js
│   ├── JS app.js
│   └── App.vue
├── package-lock.json
└── package.json
```

跨请求状态污染

- 在SPA中，整个生命周期中只有一个App对象实例 或 一个Router对象实例 或 一个Store对象实例都是可以的，因为每个用户在使用浏览器访问SPA应用时，应用模块都会重新初始化，这也是一种**单例模式**。
- 然而，在 SSR 环境下，App应用模块通常只在服务器启动时初始化一次。同一个应用模块会在多个服务器请求之间被复用，而我们的单例状态对象也一样，也会在多个请求之间被复用，比如：
 - 当某个用户对共享的单例状态进行修改，那么这个状态可能会意外地泄露给另一个在请求的用户。
 - 我们把这种情况称为：**跨请求状态污染**。
- 为了避免这种跨请求状态污染，SSR的解决方案是：
 - 可以在每个请求中为整个应用**创建一个全新的实例**，包括后面的 router 和全局 store等实例。
 - 所以我们在创建App 或 路由 或 Store对象时都是使用一个函数来创建，保证每个请求都会创建一个全新的实例。
 - 这样也会有缺点：需要消耗更多的服务器的资源。

Vue3 SSR + Hydration

■ 服务器端渲染页面 + 客户端激活页面，是页面有交互效果（这个过程称为：Hydration 水合）

■ Hydration的具体步骤如下：

- 1. 开发一个App应用，比如App.vue
- 2. 将App.vue打包为一个客户端的client_bundle.js文件
 - ✓ 用来激活应用，使页面有交互效果
- 3. 将App.vue打包为一个服务器端的server_bundle.js文件
 - ✓ 用来在服务器端动态生成页面的HTML
- 4. server_bundle.js 渲染的页面 + client_bundle.js 文件进行Hydration

```
app.get("/*", async (req, res) => {  
  const AppContent = await renderToString(App);  
  res.send(`  
    <!DOCTYPE html>  
    <html lang="en">  
      <body>  
        <h1> Vue + Server Side Render</h1>  
        <div id="app">${AppContent}</div>  
        <script src="/client/client_bundle.js"></script>  
      </body>  
    </html>  
  `);  
});
```

```
03-vue3-ssr-hydration  
├── build  
│   ├── client  
│   │   └── client_bundle.js  
│   └── server  
│       └── server_bundle.js  
├── config  
│   ├── client.config.js  
│   └── server.config.js  
├── node_modules  
├── src  
│   ├── client  
│   │   └── index.js  
│   └── server  
│       ├── index.js  
│       ├── app.js  
│       └── App.vue  
├── package-lock.json  
└── package.json
```

Vue3 SSR + Vue Router

■ 需再安装的依赖项:

□ npm i vue-router

■ 注意事项:

□ 为了避免 跨请求状态污染

□ 需在每个请求中都创建一个全新router

```
import express from "express";
import createApp from "../app";
import { renderToString } from "@vue/server-renderer";

import createRouter from "../router/index";
import { createMemoryHistory } from "vue-router";
let server = express();
server.use(express.static("build"));
server.get("/*", async (req, res) => {
  let app = createApp(); // 每次请求创建一个新的 app 实例
  // 安装路由
  const router = createRouter(createMemoryHistory());
  app.use(router);
  await router.push(req.url || "/"); // 跳转首页
  await router.isReady();

  const appContent = await renderToString(app);
  console.log(appContent);
  res.send(`...`);
});
server.listen(3000, () => {
```

04-vue3-ssr-router

- > build
- > config
- > node_modules
- ✓ src
 - ✓ client
 - JS index.js
 - ✓ router
 - JS index.js
 - ✓ server
 - JS index.js
 - ✓ views
 - ▼ about.vue
 - ▼ home.vue
 - JS app.js
 - ▼ App.vue

Vue3 SSR + Pinia

■ 需再安装的依赖项:

□ npm i pinia

■ 注意事项:

□ 为了避免 跨请求状态污染

□ 需在每个请求中都创建一个全新pinia

```
import { createPinia } from "pinia";
let server = express();
server.use(express.static("build"));
server.get("/*", async (req, res) => {
  let app = createApp(); // 每次请求创建一个新的 app 实例
  // 安装路由
  const router = createRouter(createMemoryHistory());
  app.use(router);
  await router.push(req.url || "/"); // 跳转首页
  await router.isReady(); // 在客户端和服务端我们都需要
  // 安装pinia
  const pinia = createPinia();
  app.use(pinia);

  const appContent = await renderToString(app);
```

05-vue3-ssr-pinia

- > build
- > config
- > node_modules
- ✓ src
 - ✓ client
 - JS index.js
 - > router
 - ✓ server
 - JS index.js
 - ✓ store
 - JS home.js
 - ✓ views
 - ▼ about.vue
 - ▼ home.vue
 - JS app.js
 - ▼ App.vue