第一章 Nuxt.js 简介

官网：https://zh.nuxtjs.org/

需求分析

- 采用 vue.js 开发的应用系统SEO不友好，需要解决SEO的问题
- 比如：新闻门户、博客系统有SEO的需求，希望被搜索引擎收录，百度排名靠前等。

了解搜索引擎优化 SEO

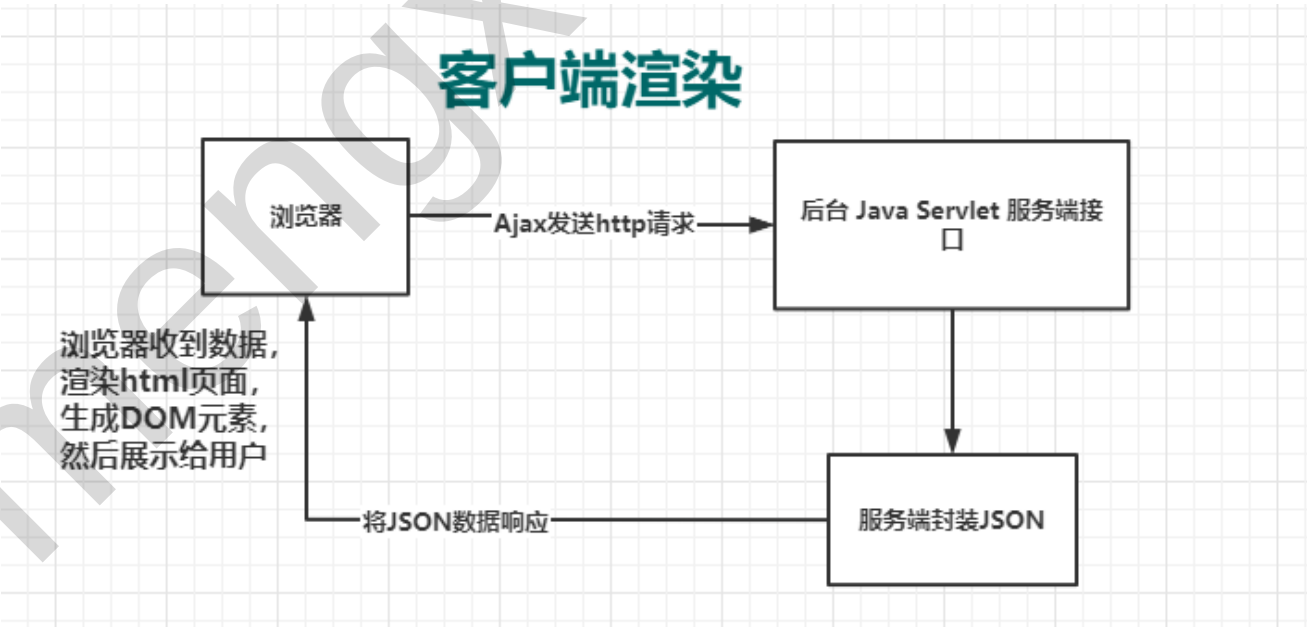
利用搜索引擎的规则提高网站在有关搜索引擎内的自然排名。目的是让其在行业内占据领先地位，获得品牌收益。很大程度上是网站经营者的一种商业行为，将自己或自己公司的排名前移。

采用什么技术可以利于 SEO 呢？下面要先理解服务端渲染和客户端渲染

服务端渲染和客户端渲染（SSR）

什么是客户端渲染

1. 浏览器（客户端）通过 AJAX 向服务端(java servlet)发送 http请求数据接口。
2. 服务端将获取的接口数据封装成 JSON ，响应给浏览器。
3. 浏览器拿到JSON就进行渲染html页面，生成DOM元素，然后将页面展示给用户。

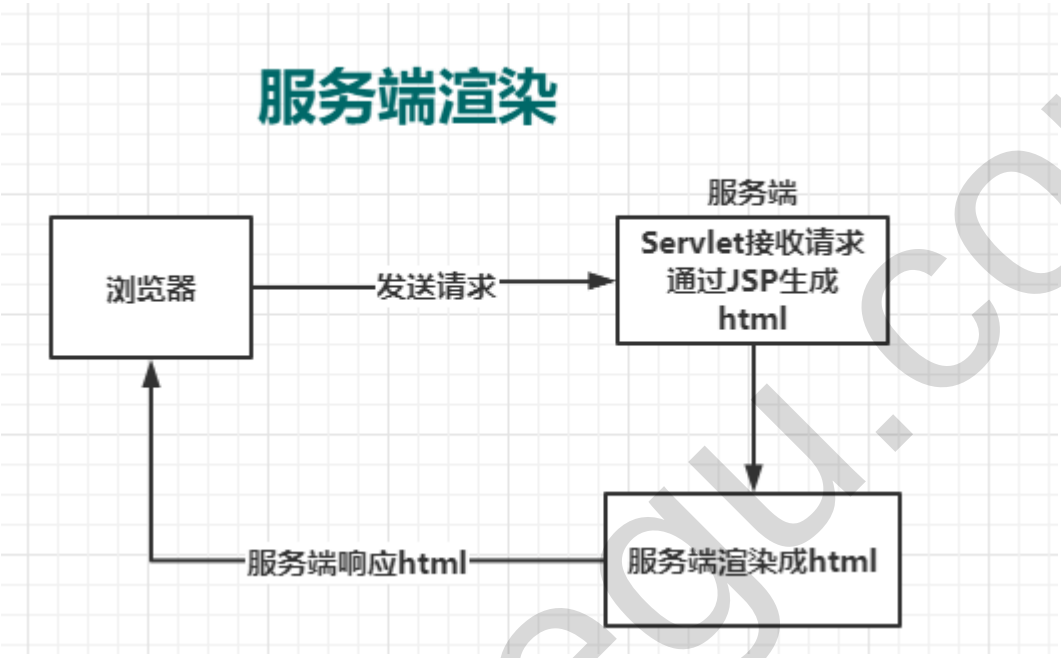


客户端渲染特点：

1. 服务端只响应数据，不生成 html 页面
2. 浏览器（客户端）负责发送请求获取服务端的数据，然后渲染成 html 页面。

什么是服务端渲染（SSR）

1. 浏览器（客户端）向服务端(java servlet)发送 http请求数据接口。
2. 服务端 （servlet ）会生成html页面，响应给浏览器。
3. 浏览器直接将接收到html页面进行展示。



服务端渲染特点：

1. 在服务端生成 html 页面。
2. 浏览器（客户端）只负责显示 html 元素内容。

为什么使用服务器端渲染（SSR）

与传统的SPA（单页面应用程序）相比，服务端渲染的优势主要有哪些：

- 更好的 SEO，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。
  如果你的应用程序初始展示Loading菊花图，然后通过Ajax获取内容，抓取工具并不会等待异步完成后再行抓取页面内容。也就是说，如果SEO对你的站点至关重要，而你的页面又是异步获取内容，则你可能需要服务器端渲染(SSR)解决此问题。
- 更快的内容到达时间（首屏加载更快），因为服务端只需要返回渲染好的HTML，这部分代码量很小的，所以用户体验更好。

使用服务器端渲染(SSR)时还需要有一些权衡之处：

- 首先就是开发成本比较高，比如某些声明周期钩子函数（如beforeCreate、created）能同时运行在服务端和客户端，因此第三方库要做特殊处理，才能在服务器渲染应用程序中运行。
- 由于服务端渲染要用Nodejs做中间层，所以部署项目时，需要处于Node.js server运行环境。在高流量环境下，还要做好服务器负载和缓存策略。

Vue.js 如何实现 SSR

1. 基于官方Vue SSR指南文档的官方方案，官方方案具有更直接的控制应用程序的结构，更深入底层，更加灵活，同时在使用官方方案的过程中，也会对Vue SSR有更加深入的了解。
2. vue.js 通用应用框架 `NUXT` ， `NUXT` 提供了平滑的开箱即用的体验，它建立在同等的 Vue.js 技术栈之上，但抽象出很多模板，并提供了一些额外的功能，例如静态站点生成。通过NUXT可以根据约定的规则，快速的实现Vue SSR。

什么是 Nuxt.js

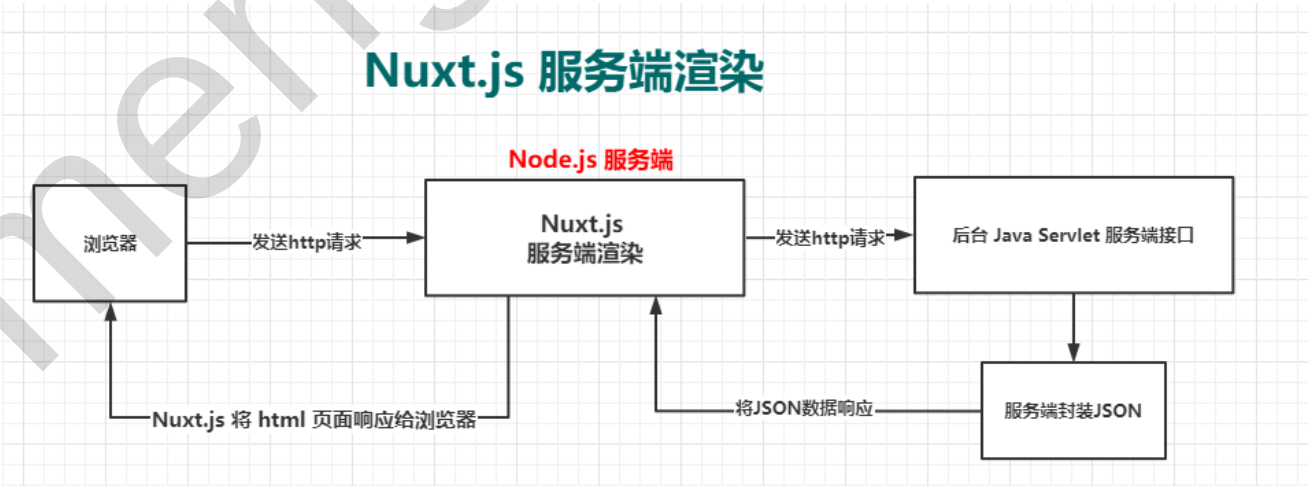
- Nuxt.js是一个基于Vue.js的通用应用框架。
- 通过对 客户端/服务端 基础框架的抽象组织，Nuxt.js主要关注的是应用的UI渲染。
- Nuxt.js的目标是创建一个灵活的应用框架，你可以基于它初始化新项目的基础代码结构。或者在已有的Node.js项目中使用Nuxt.js。其中预设了利用Vue.js开发服务端渲染的应用所需的各种配置。
- 作为框架Nuxt.js为 客户端/服务端 这种典型的应用架构模式提供了许多有用的特性，例如异步数据加载、中间件支持、布局支持等。

为什么要使用 Nuxt.js

主要是为了实现服务端的渲染（SSR），利用Nuxt.js的服务端渲染能力来解决Vue项目的SEO问题。

Nuxt.js 工作原理

1. 浏览器（客户端）发送 http 请求到 Node.js 服务端 。
2. 部署在 Node.js 的应用 Nuxt.js 接收到浏览器请求，它会去请求后台接口服务端。
3. 后台接口服务端会响应 JSON 数据，Nuxt.js 获取数据后进行服务端渲染成 html 。
4. 然后 Nuxt.js 将 html 页面响应给浏览器。
5. 浏览器直接将接收到html页面进行展示。



Nuxt.js 使用了 vue.js + webpack + babel 三大技术架构，集成了 Vue.js 中以下组件/框架，用于开发完整而强大的 Web 应用：

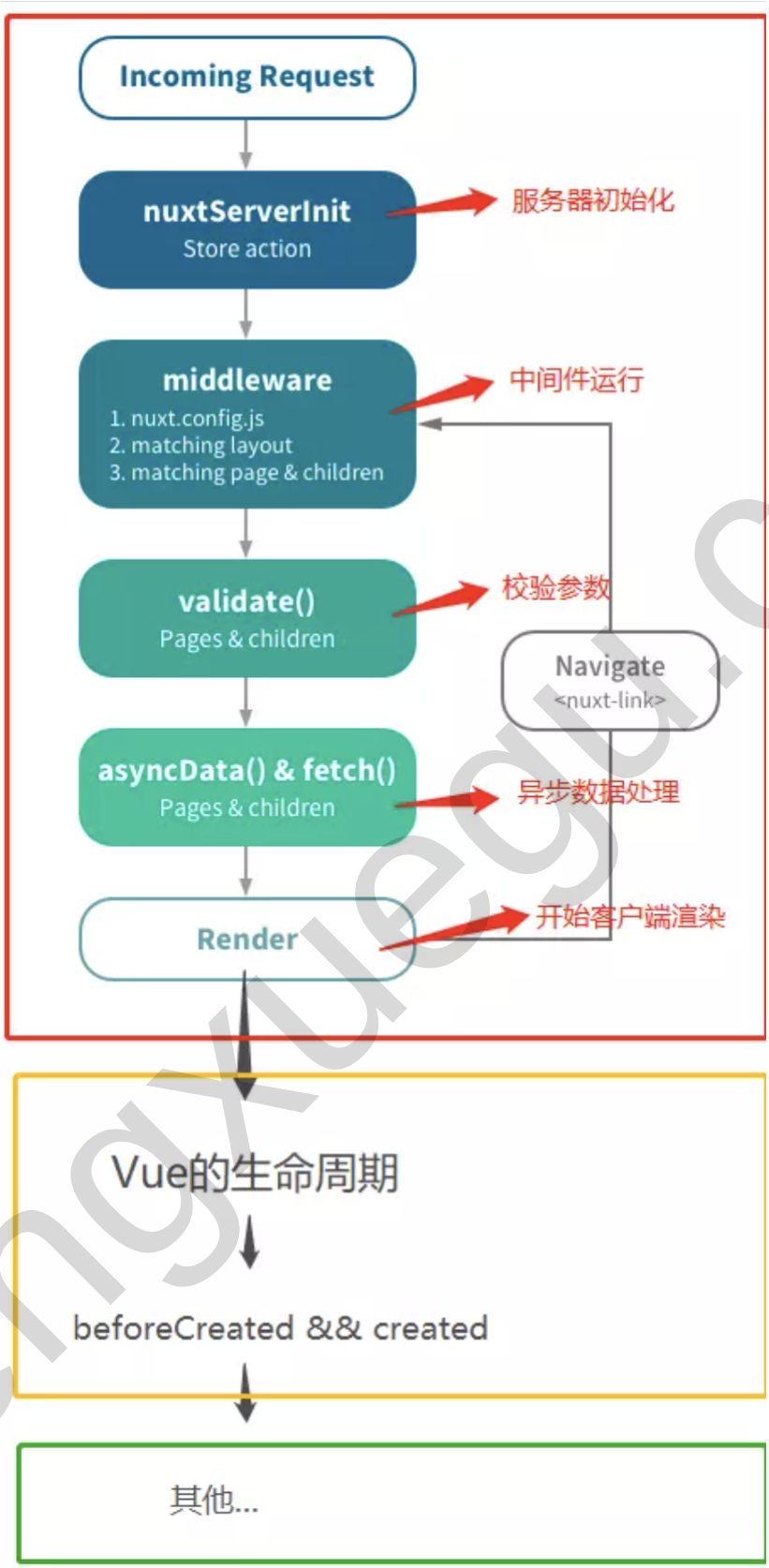
- Vue 2
- Vue-Router
- Vuex (当配置了 Vuex 状态树配置项 时才会引入)
- Vue 服务器端渲染 (排除使用 `mode: 'spa'`)
- Vue-Meta

压缩并 gzip 后，总代码大小为：**57kb** （如果使用了 Vuex 特性的话为 60kb）。

另外，Nuxt.js 使用 Webpack 和 vue-loader 、 babel-loader 来处理代码的自动化构建工作（如打包、代码分层、压缩等等）。

Nuxt 生命周期流程图分析

下图是 Nuxt.js 应用一个完整的服务器请求到渲染（或用户通过 `<nuxt-link>` 切换路由渲染页面）的流程：



红框内的是Nuxt的生命周期(运行在服务端)，黄框内同时运行在服务端&&客户端上，绿框内则运行在客户端

红框、黄框周期内的都不存在Window对象。

第二章 搭建 Nuxt 环境和创建项目

开发环境

1. 安装 node.js & npm

   https://nodejs.org/zh-cn/

2. 安装 Vscode

   https://code.visualstudio.com/

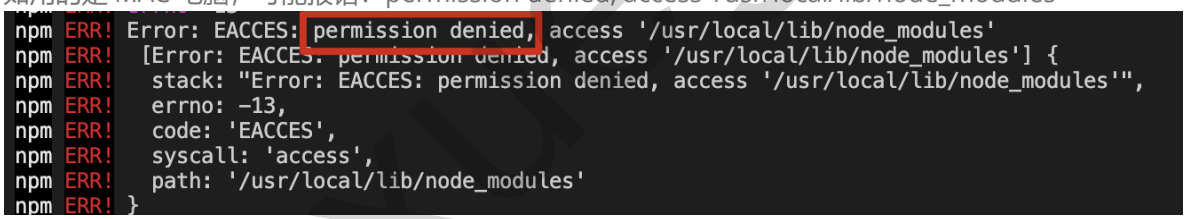
安装脚手架工具与创建项目

为了快速入门，Nuxt.js团队创建了脚手架工具 create-nuxt-app。

NPM 方式

1. 全局安装 `create-nuxt-app`

```
$ npm install -g create-nuxt-app
```

> 如用的是 MAC 电脑，可能报错：permission denied, access '/usr/local/lib/node_modules'
>
> 
>
>
> 原因: 执行命令行命令时没有获得管理员权限，输入电脑的管理员密码（开机密码） 解决：命令前加上 `sudo`
>
> ```
> $ sudo npm install -g create-nuxt-app
> ```

2. 然后用 `create-nuxt-app` 初始化项目

```
$ create-nuxt-app <项目名>
```

示例: 创建 `nuxt-demo1` 项目

```
$ create-nuxt-app nuxt-demo1
```

3. 如需卸载 create-nuxt-app， 可执行下面命令

```
# windows
$ npm uninstall -g create-nuxt-app

# MAC
$ sudo npm uninstall -g create-nuxt-app
```

NPX 方式

确保安装了npx（npx在NPM版本5.2.0默认安装了），npx 是 npm 的高级版本，npx 具有更强大的功能。

作用是避免全局安装模块：npx 临时安装一个 create-nuxt-app 模块，来初始化项目，使用过后会自动删除 create-nuxt-app 模块(下面不需要全局安装)

1. 直接初始化项目

```
$ npx create-nuxt-app <项目名>
```

示例: 创建 `nuxt-demo2` 项目

```
$ npx create-nuxt-app nuxt-demo2
```

```
mengxuegu@mengxuegu nuxt % npx create-nuxt-app nuxt-demo2

create-nuxt-app v2.14.0
✨  Generating Nuxt.js project in nuxt-demo2
? Project name nuxt-demo2
? Project description My transcendent Nuxt.js project
? Author name
? Choose the package manager Npm
? Choose UI framework None
? Choose custom server framework None (Recommended)
? Choose Nuxt.js modules (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Choose linting tools (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Choose test framework None
? Choose rendering mode Universal (SSR)
? Choose development tools (Press <space> to select, <a> to toggle all, <i> to invert selection)
⠸ Installing packages with npm
```

2. 当运行完时，它将安装所有依赖项，然后启动项目:

```
$ cd nuxt-demo2
$ npm run dev
```

打开浏览器访问 http://localhost:3000 效果如下:

> 注意：Nuxt.js 会监听 `pages` 目录中的文件更改，因此在添加新页面时无需重新启动应用程序。

Nuxt运行命令

查看 package.json

```
"scripts": {
  //  开发环境
  "dev": "nuxt",
  //  打包
  "build": "nuxt build",
  //  在服务端运行
  "start": "nuxt start",
  //  生成静态页面
  "generate": "nuxt generate"
},
```

更改端口号与主机

如果默认 `3000` 端口号被占用，可在 `nuxt.config.js` 中更改端口号：

> 参考：https://zh.nuxtjs.org/api/configuration-server

```
export default {
  mode: 'universal', // 代表SSR，还可取值 `spa` 单体应用
  server: {
    port: 8000,
    host: '0.0.0.0' // 默认：localhost
  },
  // 其他配置项
}
```

更改后重启项目，然后访问 http://localhost:8000/

详解项目目录结构

目录结构

- `.nuxt` ：执行 `npm run dev` 命令后编译的目录文件。
- `assets` ：用于组织未编译的静态资源如 `LESS` 、`SASS` 或 `JavaScript` 。
- `components` ：用于组织应用的 Vue.js 组件。Nuxt.js 不会扩展增强该目录下 Vue.js 组件，即这些组件不会像页面组件那样有 `asyncData` 方法的特性。
- `layouts` ：用于组织应用的布局组件。
- `middleware` ：用于存放应用的中间件。
- `node_modules` ：用于存放项目的依赖包
- `pages` ：用于组织应用的路由及视图。Nuxt.js 框架读取该目录下所有的 `.vue` 文件并**自动生成对应的路由配置。**
- `plugins` ：用于组织那些需要在 `根vue.js应用` 实例化之前需要运行的 Javascript 插件
- `static` ：静态文件目录 `static` 用于存放应用的静态文件，此类文件不会被 Nuxt.js 调用 Webpack 进行构建编译处理。 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。

  举个例子: `/static/robots.txt` 映射至 `/robots.txt`
- `store` ：用于组织应用的 Vuex 状态树 文件。 Nuxt.js 框架集成了 Vuex 状态树 的相关功能配置，在 `store` 目录下创建一个 `index.js` 文件可激活这些配置。
- `.editorconfig` ：用于指定编辑器编写项目的代码风格

```
# editorconfig.org
#项目里读editorcongig文件时，读到此文件即可，停止向上寻找配置文件
root = true

[*]
#缩进风格：空格
indent_style = space
#缩进大小2

indent_size = 2
```

```
#换行符lf
end_of_line = lf
#字符集utf-8
charset = utf-8
#是否删除行尾的空格
trim_trailing_whitespace = true
#是否在文件的最后插入一个空行
insert_final_newline = true

[*.md]
# 删除行尾空格 = 否
trim_trailing_whitespace = false
```

- `.gitignore` 文件用于指定哪些文件不被提交到Git仓库中。
- `nuxt.config.js` 文件用于组织 Nuxt.js 应用的个性化配置，以便覆盖默认配置。
- `package-lock.json` 文件用于描述应用具体的依赖版本
- `package.json` 文件用于描述应用的依赖关系和对外暴露的脚本接口。

别名 ~ @

别名	目录
`~` 或 `@`	srcDir
`~~` 或 `@@`	rootDir

默认情况下， `srcDir` 和 `rootDir` 相同。

> **提示:** 在您的 `vue` 模板中, 如果你需要引入 `assets` 或者 `static` 目录, 使用 `~/assets/your_image.png` 和 `~/static/your_image.png` 方式。

第三章 布局与错误页

默认页面布局

1. 当执行 `npm run dev` 后，在会生成一个默认的 html 模板页面 `.nuxt/views/app.template.html` 内容如下：

```
<!DOCTYPE html>
<html {{ HTML_ATTRS }}>
  <head {{ HEAD_ATTRS }}>
    {{ HEAD }}
  </head>
  <body {{ BODY_ATTRS }}>
    {{ APP }}
  </body>
</html>
```

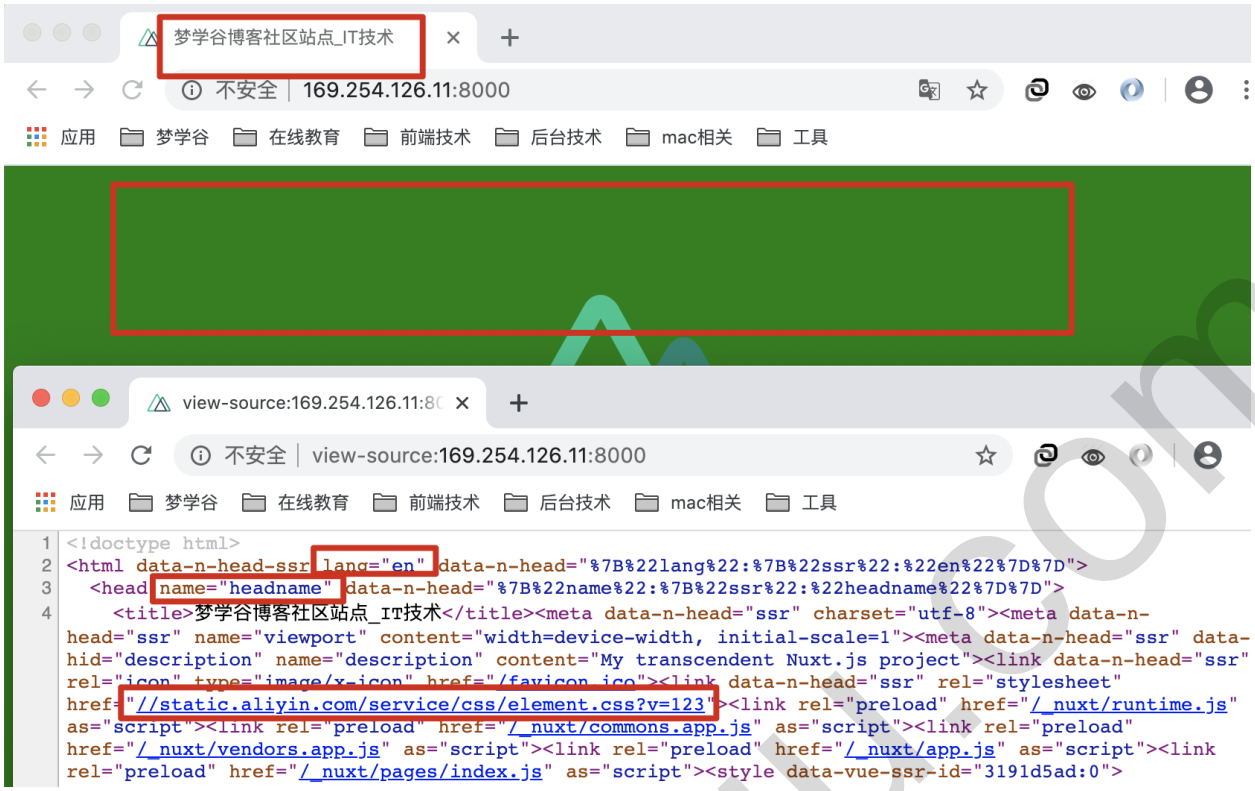
> {{ APP }} 渲染的是主体内容，就是 nuxt/nuxt-demo2/pages 下的页面组件

2. 页面中的表达式引用的是 nuxt.config.js 文件中 `head` 属性值：

```
  head: {
    htmlAttrs: { // 对应 {{ HTML_ATTRS }} ，html标签属性
      lang: 'en',
    },
    headAttrs: { // 对应 {{ HEAD_ATTRS }}，head标签属性
      // 一般没有什么可设置的，随便设置个
      name: 'headname'
    },
    bodyAttrs: { // 对应 {{ HOBDY_ATTRS }} body标签属性
      style: 'background-color: green'
    },
    // 对应 {{ HEAD }} 即 head标签体内容----start
    title: "梦学谷博客社区站点_IT技术", // process.env.npm_package_name || '', // package.js
的 name
    meta: [
      { charset: 'utf-8' },
      { name: 'viewport', content: 'width=device-width, initial-scale=1' },
      { hid: 'description', name: 'description', content:
process.env.npm_package_description || '' }
    ],
    link: [
      { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
      { rel: 'stylesheet', href: '//static.aliyin.com/service/css/element.css?v=123' }
    ],
    // 对应 {{ HEAD }}即 head标签体内容----end
  },
```

3. 重新运行，访问 http://localhost:8000/ ，查看对应的都生效了。

> 访问 http://localhost:8000/ 对应渲染的主体内容就是 `pages/index.vuex`

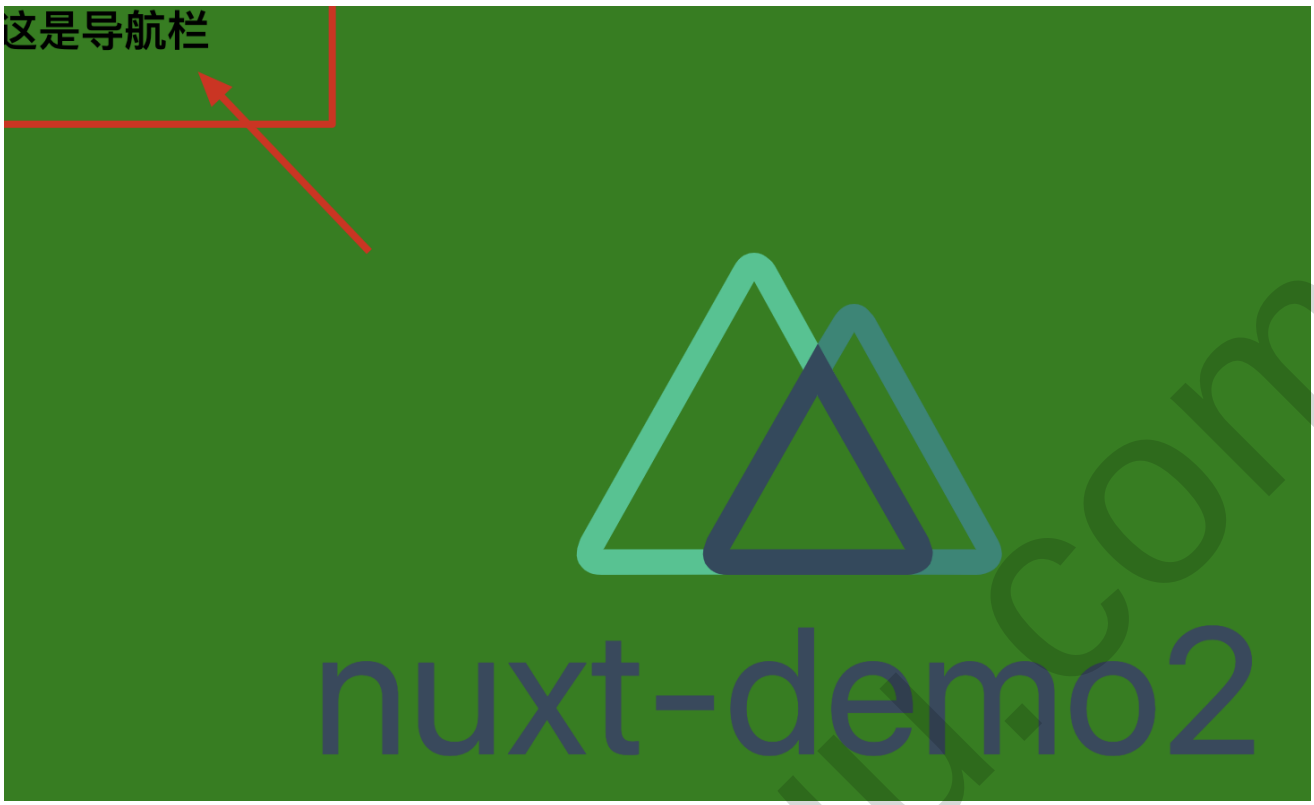


自定义页面布局

我们也可以定制化模板页面，只需要在应用根目录下创建一个 `app.html` 的文件，就可以覆盖 Nuxt.js 默认模板。

```
<!DOCTYPE html>
<html {{ HTML_ATTRS }}>
  <head {{ HEAD_ATTRS }}>
    {{ HEAD }}
  </head>
  <body {{ BODY_ATTRS }}>
      <h2>这是导航栏</h2>
    <!-- 主体内容 -->
    {{ APP }}
  </body>
</html>
```

创建 app.html 文件后，注意要重启项目。 重新访问右上角多有文字出来。



上面这种 app.html 页面布局一般不能，因为实际项目中基本上使用 `.vue` 组件化文件开发。

组件布局

默认组件布局

Nuxt.js 可通过添加 `layouts/default.vue` 文件来扩展应用的默认布局。

nuxt 官方提供的默认布局源码如下：

```
<template>
  <div>
    <nuxt />
  </div>
</template>
```

> 在布局文件中添加 `<nuxt />` 组件用于显示页面的主体内容。

自定义组件布局

在 `layouts` 目录中的创建 `.vue` 文件来自定义布局， 然后通过页面组件（pages目录下的文件）中的 `layout` 属性来引用自定义布局。

示例：

1. 下面我们创建 `layouts/blog.vue` 文件，在其中定义一个博客上下结构的布局文，如下：

> 注意：在布局文件中添加 `<nuxt/>` 组件，用于显示页面的主体内容。

```
<template>
  <div>
    <!-- 头部导航 -->
    <div >
      <h1>博客导航栏</h1>
    </div>
    <!-- 主体内容 -->
    <nuxt />
  </div>
</template>
```

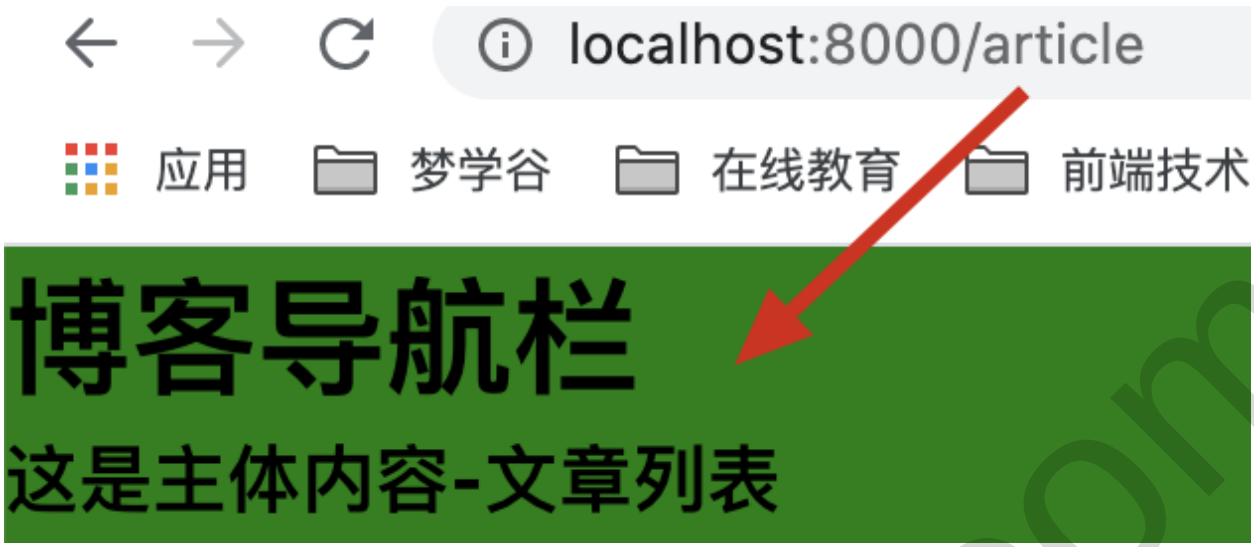
2. 在 pages 目录下面的页面组件通过 `layout` 属性引用自定义组件布局，如 `pages/article.vue`

> `layout` 默认值是 `default`

```
<template>
    <div>
        <h3>这是主体内容-文章列表</h3>
    </div>
</template>
<script>
export default {

    // 引用的是 /layouts/blog.vue 布局
    // layout: 'blog',
    // 或者
    layout(context) {
        console.log('context', context)
        return 'blog'
    }
}
</script>
```

- 注意不写 `layout` ，则页面组件默认采用的是 default.vue 布局
- 上面参数 `context` 上下文对象可获取的属性参考：https://zh.nuxtjs.org/api/context/

3. 删除根目录下创建的 app.html 页面布局文件，组件布局才会生效。
4. 重启项目，再访问http://localhost:8000/article 注意访问 /article



5. **注意：如果是 `layouts/blog` 目录下放了 `index.vue`，直接通过 `layout: 'blog'` 是找不到这个页面的，需要指定文件名 index 才找得到，即 `layout: 'blog/index'` 。如下图：**

```
<template>
    <div>
        <h3>这是主体内容-文章列表</h3>
    </div>
</template>
<script>
export default {
    // 引用的是 /layouts/blog.vue 布局
    // layout: 'blog',
    // 或者
    layout(context) {
        console.log('context', context)
        // 要写完整路径，加上 index，才可引用 blog/index.vue 布局
        return 'blog/index'
    }
}
</script>
```

重构默认组件布局 default.vue

1. 修改 layouts/default.vue 默认布局，分为头部导航、和主体内容

```
<template>
  <div>
    <!-- 头部导航 -->
    <div class="header">
      <ul>
        <li><a href="">首页</a></li>
        <li><a href="">文章</a></li>
        <li><a href="">问答</a></li>
        <li><a href="">个人中心</a></li>
      </ul>
    </div>
    <!-- 主体内容 -->
    <nuxt />
  </div>
</template>
```

2. 为 layouts/default.vue 默认布局添加样式

```
<style>
  * {
      margin: 0 auto;
      font-size: 18px;
  }
  .header {
    width: 100%;
    height: 50px;
    background-color: black;
  }
  .header ul {
    margin-left: 80px;
  }
  .header li {
    display: inline;
    line-height: 50px;
    margin-left: 60px;
  }
  .header a {
    color: #fff;
    text-decoration: none; /* 去除下划线 */
  }
  .header a:hover {
    color: aqua;
  }
</style>
```

3. 访问首页 http://localhost:8000/ 浏览效果

定制错误页面

1. 通过编辑 `layouts/error.vue` 文件来定制化错误页面（404，500等）。

   警告: 虽然此文件放在 `layouts` 文件夹中, 但应该将它看作是一个**页面(page)**，而不是布局使用。

2. 示例 `layouts/error.vue`

```
<template>
  <div >
    <h1 v-if="error.statusCode === 404">
      您访问的页面不存在：{{error.message}}
    </h1>
    <h1 v-else>请求接口失败或超时！请刷新重试</h1>
  </div>
</template>

<script>
export default {
  props: ['error'], // 直接引用Error错误对象
  // 可只为 blog 布局定制的错误页面，默认 default 所有布局的错误页
  // layout: 'blog'
}
</script>
```

3. 发送一个错误的请求：http://localhost:8000/test 响应401

设置当前页面头部标签

使用 `head` 方法设置当前页面的头部标签。

在 `head` 方法里可通过 `this` 关键字来获取组件的数据，你可以利用页面组件的数据来设置个性化的 `meta` 标签。

1. 修改首页组件 page/index.vue

```
<template>
    <div>
        <h3>这是主体内容-首页</h3>
    </div>
</template>
<script>
export default {

  data() {
    return {
      title: '博客文章列表页'
    }
  },

  // 使用 head 方法设置当前页面的头部标签。
  // 在 head 方法里可通过 this 关键字来获取组件的数据，你可以利用页面组件的数据来设置个性化的 meta 标签
  head () {
    return  {
      bodyAttrs: { // 对应 {{ HOBDY_ATTRS }} body标签属性
        style: 'background-color: #fff'
      },
      title: this.title,
      meta: [
        // hid 指定标识，防止不能覆盖父组件的 <meta name="description" > 标签，
        // hid: 'description' 就会覆盖父组件中的属性为 name="description" 的meta标签
        { hid: 'description', name: 'description', content: 'nuxt.js技术栈' }
      ]
    }
  }
}

</script>
```

注意：为了避免子组件中的meta标签，不能正确覆盖父组件中相同的name名称的meta标签，而产生重复meta的现象，可使用 `hid` 键指定要覆盖的父组件name值与hid相同的meta标签。请阅读关于 `vue-meta` 的更多信息。

如上面 `hid: 'description'` 就会覆盖父组件中的属性为 `name="description"` 的meta标签，不会出现多个的 `<meta name="description">`

第四章 插件 Plugin

Nuxt.js允许您在运行Vue.js应用程序之前执行js插件。这在您需要使用自己的库或第三方模块时特别有用。

可以将自定义插件注入到 Vue 实例（客户端），context（服务器端）、store(Vuex)。

新增的属性或方法名使用 `$` 作为前缀。

注入 Vue 实例

将内容注入Vue实例，避免重复引入，在Vue原型上挂载注入一个函数，所有组件内都可以访问(**不包含服务器端**)。

`plugins/vue-inject.js`：

```
import Vue from 'vue'

Vue.prototype.$myVueFunction = string => console.log('绑定到Vue实例的方法参数', string)
```

`nuxt.config.js`：

```
export default {
  plugins: ['~/plugins/vue-inject.js']
}
```

这样，您就可以在所有Vue组件中使用该函数。

`plugin-vue.vue`：

```
export default {
  mounted () {
    this.$myVueFunction('test')
  }
}
```

注入 context

context注入方式和在其它vue应用程序中注入类似。

`plugins/ctx-inject.js`:

```
export default ({ app }, inject) => {
  app.myContextFunction = string => console.log('绑定到Context的方法参数', string)
}
```

`nuxt.config.js`:

```
export default {
  plugins: ['~/plugins/ctx-inject.js']
}
```

现在，只要您获得context，你就可以使用该函数。

`plugin-ctx.vue`:

```
export default {
  asyncData (context) {
    context.app.myContextFunction('ctx!')
  }
}
```

同时注入在`context`，`Vue`，`Vuex`实例

如果您需要同时在`context`，`Vue`实例，甚至`Vuex`中同时注入，您可以使用`inject`方法,它是plugin导出函数的第二个参数。 将内容注入Vue实例的方式与在Vue应用程序中进行注入类似。系统会自动将`$`添加到方法名的前面。

`plugins/all-inject.js`:

```
export default ({ app }, inject) => {
  inject('myAllFunction', string => console.log('绑定到Context和Vue的方法参数', string))
}
```

`nuxt.config.js`：

```
export default {
  plugins: ['~/plugins/all-inject.js']
}
```

现在您就可以在`context`，或者`Vue`实例中的`this`，或者`Vuex`的`actions/mutations`方法中的`this`来调用`myAllFunction`方法。

`plugin-all.vue`：

```
export default {
  mounted () {
    this.$myAllFunction('mounted钩子调用myAllFunction')
  },
  asyncData (context) {
    context.app.$myAllFunction('asyncData调用myAllFunction')
  }
}
```

只在客户端使用的插件

```
export default {
  plugins: [
    { src: '~/plugins/combined-inject.js' },
    { src: '~/plugins/combined-inject.js', mode: 'client' }, // 插件只会在客户端运行。
    { src: '~/plugins/combined-inject.js', mode: 'server' }, // 插件只会在服务端运行。
  ]
}
```

第五章 异步加载数据 asyncData

Nuxt.js 扩展了Vue.js，增加了一个叫`asyncData`的方法，使得我们可以在渲染组件之前异步获取数据。

`asyncData`方法会在组件（**限于页面组件**）每次加载之前被调用。它可以在服务端或路由更新之前被调用。 在这个方法被调用的时候，第一个参数`context`被设定为当前页面的上下文对象，你可以利用`asyncData`方法来获取数据，Nuxt.js 会将`asyncData`返回的数据与`data`方法返回的数据一起合并后返回给当前组件。

调用后台数据接口我们采用 axios 异步发送请求，所以下面我们要先安装它。

安装 `@nuxtjs/axios`

参考官网：https://axios.nuxtjs.org/

Nuxt.js 官方提供了 `@nuxtjs/axios` 模块，此模块中还包含了 axios 、@nuxtjs/proxy 模块。其中 @nuxtjs/proxy 是解决 Nuxt 中跨域问题，进行代理转发请求。

所以我们只要安装 `@nuxtjs/axios` 即可：

```
npm install @nuxtjs/axios
```

2. 在 nuxt.config.js 引入 @nuxtjs/axios 模块

```
  modules: [
    '@nuxtjs/axios',
  ],
```

在 easymock 上创建数据接口

访问 http://mock.mengxuegu.com/ 创建数据接口

get 请求 /test 接口，响应如下数据

```
{
  "code": 20000,
  "message": "成功",
  "data": {
    "title": "@ctitle",
    "content": "@cparagraph",
    "author": "@cname"
  }
}
```

归属 / 项目名

1586841388312 / 博客门户网站数据接口

项目基础 URL

/ blog-web

项目描述

博客门户网站数据接口

Swagger Docs API (可选)

URL

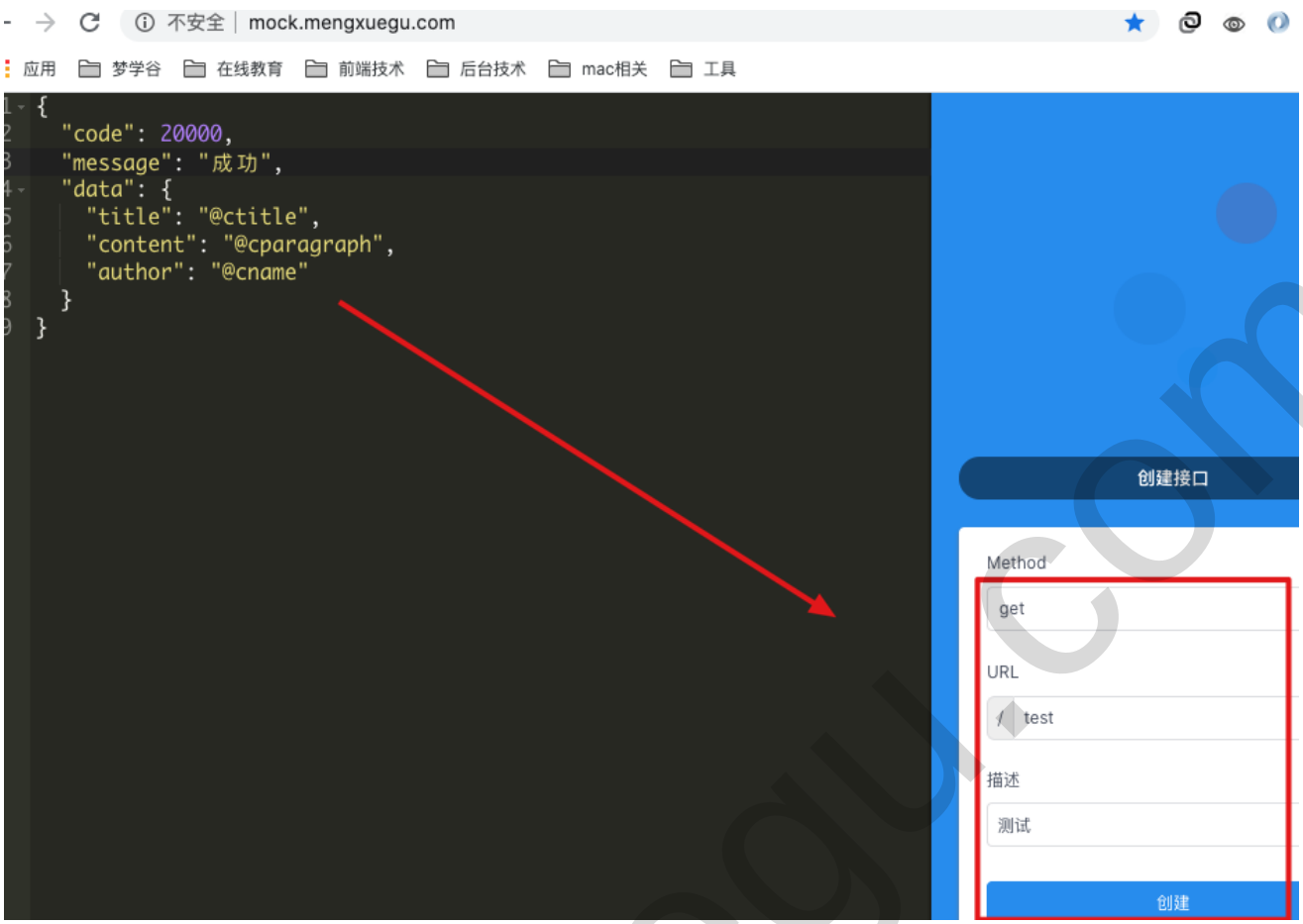
http://example.com/swagger.json

如果后台有提供 Swagger 文档（并且没有验证授权的问题），于是我们可以在此处填写 Swagger 的接口地址, Easy Mock 会自动基于此接口创建 Mock 接口.

邀请成员 协同编辑 (可选)

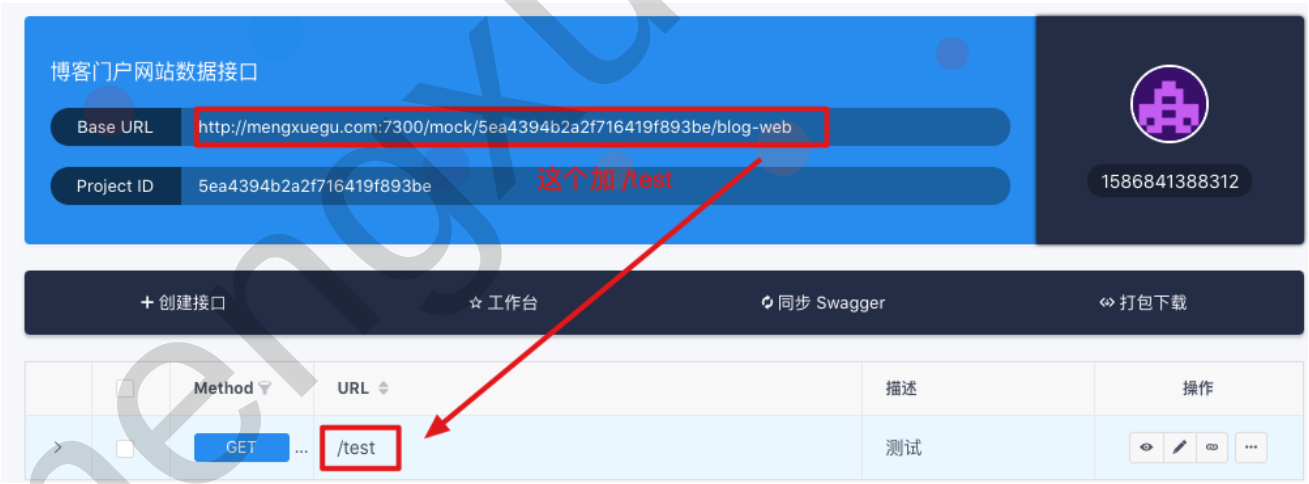
用户昵称、用户名，支持模糊匹配

创建



Base URL + 接口 URL 就是调用地址

http://mengxuegu.com:7300/mock/5ea4394b2a2f716419f893be/blog-web/test



首页请求数据接口

Nuxt.js 提供了几种不同的方法来使用 `asyncData` 方法，你可以选择自己熟悉的一种来用：

1. 返回一个 `Promise`，nuxt.js会等待该 `Promise` 被解析之后才会设置组件的数据，从而渲染组件.
2. 使用 async 或 await (了解更多)

使用 axios 返回 `Promise`

1. 注意： 因为 asyncData方法是在当前组件加载之前被调用，不能在 asyncData 方法体中用 this 传值
2. 在 nuxt-demo2/pages/index.vue 添加如下代码：

```
<template>
    <div class="container">
        <h2> 首页标题： {{title}}</h2>
        <ul>
          <li>标题: {{data.title}}</li>
          <li>内容: {{data.content}}</li>
          <li>作者: {{data.author}}</li>
        </ul>
      </div>
</template>

<script>
export default {

  data() {
    return {
      title: '博客文章列表页'
    }
  },

    // 加载组件之前服务端会调用
    // 方式1: 使用了两个return
  asyncData({$axios}) {
    return $axios.$get('http://mengxuegu.com:7300/mock/5ea4394b2a2f716419f893be/blog-
web/test')
    .then(response => {
      console.log('response', response)
      const data = response.data
      return { data }// {data: data}
    })
  }
  // head省略

}
</script>
```

如果一直渲染不出数据，检查接口地址，或者 `$axios` 前面少了 `return` ，正常的是 `return $axios.$get`

3. 访问首页 http://localhost:8000/ 效果

这是主体内容-首页

- 标题：律先照目
- 内容：只调且业克展民多风意党术题人同周建老。速离发识色化全如例变难说相四使。
  眼点往记连气形包图你列。
- 作者：万军

使用 async与await

```
export default {

  // 方式2 请求数据接口 async await
  async asyncData( {$axios} ) {
    const response = await
$axios.$get('http://mengxuegu.com:7300/mock/5ea4394b2a2f716419f893be/blog-web/test')
    console.log('response', response)
    return {data: response.data}
  }

}
```

访问首页 http://localhost:8000/ 效果一样。

Nuxt.js中配置代理解决跨域

如果调用后台接口有跨域问题，需要对请求地址进行代理转发才可解决。

我们知道在vue-cli中配置代理很方便，只需要在 `vue.config.js` 中找到 `proxyTable` 添加即可，而在nuxt中同样需要修改 `nuxt.config.js` 配置文件，前提要安装了 @nuxtjs/axios 。参考 https://axios.nuxtjs.org/

1. 在 nuxt.config.js 中开启代理配置

```
  modules: [ // 数组
    '@nuxtjs/axios', // 引用模块
  ],

  axios: { // 对象
    proxy: true // 开启代理
    prefix: '/api' // 请求前缀

  },
```

```
  proxy:{ // 对象
    // 将 /api 替换为 '', 然后代理转发到 target 指定的 url
    '/api': {
       target: 'http://mengxuegu.com:7300/mock/5ea4394b2a2f716419f893be/blog-web',
       pathRewrite: {'^/api': ''}
    }
  }, // 逗号

}
```

> `pathRewrite` 选项(重写地址)
>
> `/api/` 将被添加到API端点的所有请求中。可以使用 `pathRewrite` 选项删除。
>
> 因为在 ajax 的 url 中加了前缀 `/api`，而原本的接口是没有这个前缀的。
>
> 所以需要通过 pathRewrite 来重写地址，将前缀 `/api` 转为 `/` 或者是 `''`。
>
> 如果本身的接口地址就有 `/api` 这种通用前缀，就可以把 `pathRewrite` 删掉。

2. 在 nuxt-demo2/pages/index.vue 修改请求接口 `/test` ：

```
export default {

  // 方式2 请求数据接口 async await
  async asyncData({$axios, error}) {
    try {
      const response = await $axios.$get('/test')
      return {data: response.data.data}
    } catch (e) {
       error({statusCode: 404, message: '未找到请求资源。'})
    }
  },

}
```

3. 请求 http://localhost:8000/

$axios 拦截器

1. 创建 plugins/interceptor.js 请求拦截器

```
export default ({ store, route, redirect, $axios}) => {

  $axios.onRequest( config => {
      console.log('请求拦截器')
      return config
```

```
  })

  $axios.onResponse(
    response => {
      console.log('响应拦截器', response)
      return response
    },
  )

  $axios.onError(
    error => {
        console.log('error.response.status', error.response.status)
    }
  )

}
```

2. 引入插件

```
plugins: ['~/plugins/interceptor.js']
```

第六章 插件方式解耦在 NuxtJs 中调用 api

创建api插件 api/test.js

```
export default ({ $axios }, inject) => {

    inject('test', data => $axios.$get( `/test` ) )

}
```

引入插件

```
plugins: ['~/api/test.js']
```

组件中调用

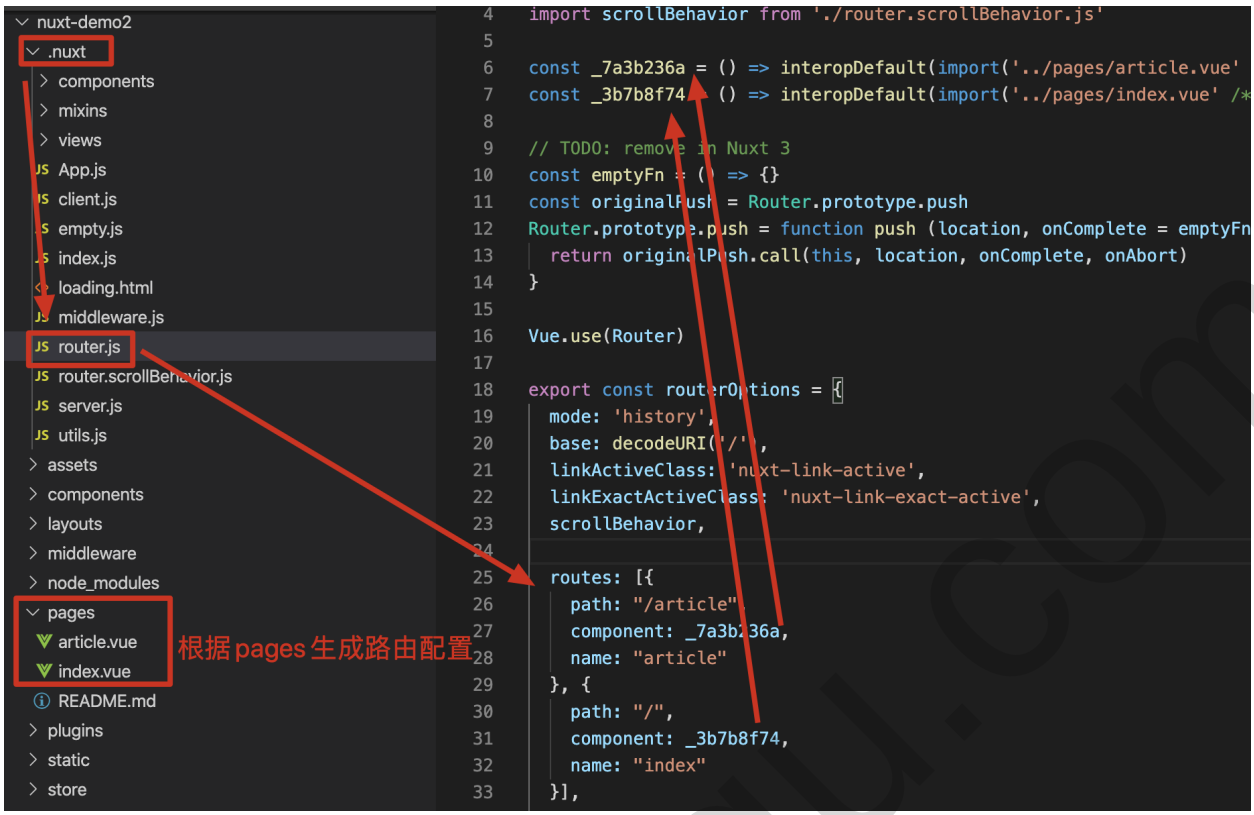
```
async asyncData({ app }) {
    const response = await app.$test()
    console.log(response)
},

methods: {
  testApiPlugin() {
    const response = this.$test()
  }
}
```

第七章 路由 Vue-Router

Nuxt.js 依据 `pages` 目录结构自动生成 vue-router 模块的路由配置，不需要我们去额外的配置路由。

1. Nuxt 根据 pages 下面的 article.vue 和 index.vue 页面组件，生成如下路由配置：

```
routes: [{
  path: "/article",
  component: "pages/article.vue",
  name: "article"
}, {
  path: "/",
  component: "pages/index.vue",
  name: "index"
}],
```



2. 在页面之间使用路由跳转，官方建议使用 `<nuxt-link>` 标签。

   `<nuxt-link>` 的作用与Vue的 `<router-link>` 一致

   - 在 layouts/default.vue 中将 `a` 标签改为 `<nuxt-link>` ， `to` 指定要跳转的路由地址。

```
<template>
  <div>
    <!-- 头部导航 -->
    <div class="header">
      <ul>
        <!-- <li><a href="">首页</a></li>
        <li><a href="">文章</a></li>
        <li><a href="">问答</a></li>
        <li><a href="">个人中心</a></li> -->
        <li><nuxt-link to="/">首页</nuxt-link></li>
        <li><nuxt-link to="/article">文章</nuxt-link></li>
        <li><nuxt-link to="">问答</nuxt-link></li>
        <li><nuxt-link to="">个人中心</nuxt-link></li>
      </ul>
    </div>
    <!-- 主体内容 -->
    <nuxt />
  </div>
</template>
```

点击 *文章*，跳到了 /pages/article.vue 页面

基础路由

1. 在 `pages` 下创建 user 目录，user目录下创建 index.vue 和 one.vue 结构如下：

```
pages/
--| user/     ++++++
-----| index.vue  ++++++
-----| one.vue    ++++++
--| article.vue
--| index.vue
```

2. 那么，Nuxt.js 自动生成的路由配置如下：

```
router: {
  routes: [
    {
      path: '/user',
      component: 'pages/user/index.vue',
      name: 'user',
    },
    {
      path: '/user/one',
      component: 'pages/user/one.vue',
      name: 'user-one',
    },
    {
      path: "/article",
      component: "pages/article.vue",
      name: "article"
    },
    {
      path: "/",
      component: "pages/index.vue",
      name: "index"
    }
  ]
}
```

2. 在 layouts/default.vue 对个人中心添加路由跳转：

```
<template>
  <div>
    <!-- 头部导航 -->
    <div class="header">
      <ul>
        <li><nuxt-link to="/">首页</nuxt-link></li>
        <li><nuxt-link to="/article">文章</nuxt-link></li>
        <li><nuxt-link to="">问答</nuxt-link></li>

        <li><nuxt-link to="/user">个人中心</nuxt-link></li>
```

```
        <li><nuxt-link to="/user/one">UserOne</nuxt-link></li>
      </ul>
    </div>
    <!-- 主体内容 -->
    <nuxt />
  </div>
</template>
```

3. 点击个人中心

动态路由

在 Nuxt.js 里面定义带参数的动态路由，需要创建对应的**以 _ 下划线作为前缀**的 Vue 文件 或 目录。

下划线为前缀的Vue文件

1. 在 pages/user 目录下创建**以 _ 下划线作为前缀**的 Vue 文件 `_id.vue`

   以下目录结构：

```
pages/
--| user/
-----| _id.vue # 通过id查看用户详情，++++++
-----| index.vue
-----| one.vue
--| article.vue
--| index.vue
```

2. Nuxt.js 生成对应的路由配置表为：

```
router: {
  routes: [

    {
      path: "/user/:id",
      component: "pages/user/_id.vue",
      name: "user-id"
    },

  ]
}
```

3. 路由地址中 `:id` 用于匹配请求参数。
   - 在 `user/_id.vue` 组件中使用 validate 校验参数值是否合法。
   - asyncData 通过 `{params}` 获取参数值，不能通过 this.$route.params.id 这种获取方式获取。

```
<template>
    <div>
        <h1>
            获取到的用户id是: {{ $route.params.id }}
        </h1>
    </div>
</template>

<script>
export default {
  // 校验参数值
  validate ({ params }) {
    // 必须是number类型
    return /^\d+$/.test(params.id)
  },
  // 查询数据
  asyncData( { params } ) {
    // this.$route.params.id 这种获取方式是错的
    const id = params.id
    console.log(`查询id=${id}的用户数据`)
  },
}
</script>
```

4. 测试，请求 http://localhost:8000/user/2

5. 发送一个参数是字母的请求 http://localhost:8000/user/aa ，user/_id.vue 组件不匹配，nuxt.js 就会去找 user/aa.vue 组件，但是当前没有，就会抛出异常。

下划线为前缀的目录

1. 在 pages 目录下创建 **以 _ 下划线作为前缀**的问答模块目录 `_question` ，并创建两个文件 index.vue 和 view.vue

   以下目录结构：

```
pages/
--| _question/ # 问答模块，热门hot，最新new，待回答wait
-----| index.vue # 问答列表，根据不同类型问题，显示不同列表
-----| view.vue # 问题详情
--| user/
-----| _id.vue
-----| index.vue
-----| one.vue
--| article.vue
--| index.vue
```

2. Nuxt.js 生成对应的路由配置表为：

```
router: {
    routes: [

    // http://localhost:8000/hot
    {
      path: "/:question",
      component: "pages/_question/index.vue",
      name: "question"
    },

    // http://localhost:8000/hot/view
    {
            path: "/:question/view",
      component: "pages/_question/view.vue",

      name: "question-view"
```

```
    }

  ]
}
```

3. 路由地址中 `:question` 用于匹配请求参数。
   - 在 `pages/_question/index.vue` 组件中使用 validate 校验参数值是否合法。
   - asyncData 通过 `{params}` 获取参数值，不能通过 this.$route.params.id 这种获取方式获取。

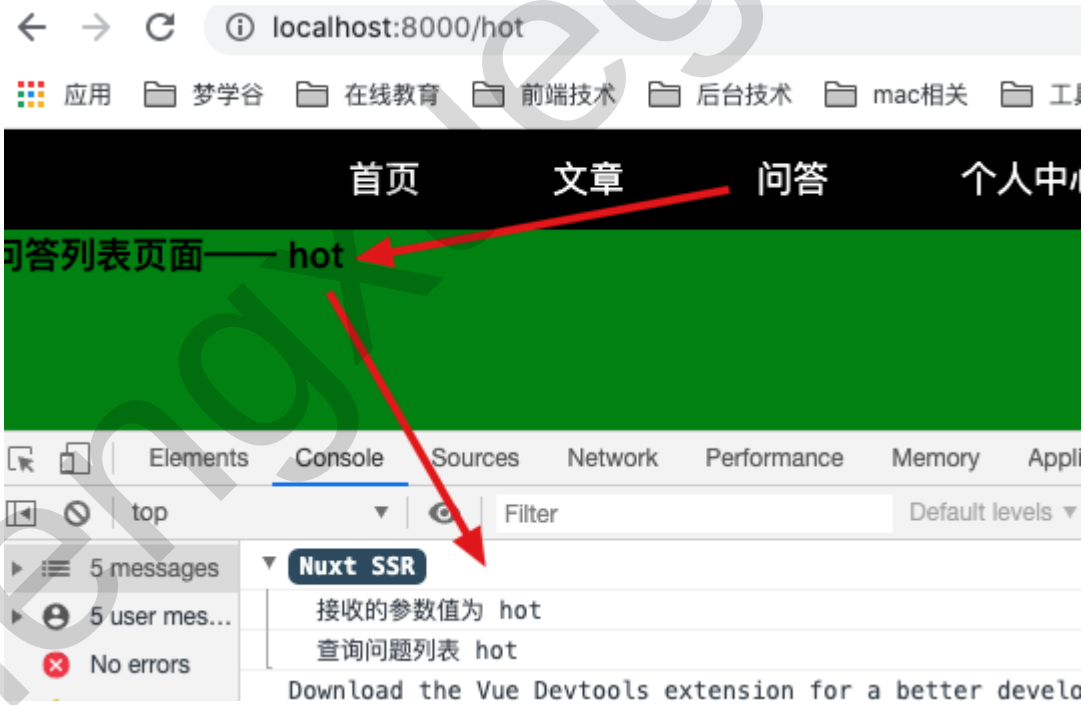
```
<template>
    <div>
        <h1>问答列表页面— {{ $route.params.question }}</h1>
    </div>
</template>
<script>
export default {
    validate({params}) {
        console.log('接收的参数值为', params.question)
        const arr = ['hot', 'wait', 'red']
        // 必须是为数组中的某一个元素
        return arr.indexOf(params.question) != -1
    },
    asyncData({params}) {
        console.log('查询问题列表', params.question)
    }
}
</script>
```

4. 修改 layouts/default.vue 默认布局文件中的*回答*路由地址

```
<template>
  <div>
    <!-- 头部导航 -->
    <div class="header">
      <ul>
        <li><nuxt-link to="/">首页</nuxt-link></li>
        <li><nuxt-link to="/article">文章</nuxt-link></li>

        <li><nuxt-link to="/hot">问答</nuxt-link></li>

        <li><nuxt-link to="/user">个人中心</nuxt-link></li>
        <li><nuxt-link to="/user/one">UserOne</nuxt-link></li>
      </ul>
    </div>
    <!-- 主体内容 -->
    <nuxt />
  </div>
</template>
```

```vue
<template>
  <div>
    <!-- 头部导航 -->
    <div class="header">
      <ul>
        <li><nuxt-link to="/">首页</nuxt-link></li>
        <li><nuxt-link to="/article">文章</nuxt-link></li>
        <li><nuxt-link to="/hot">问答</nuxt-link></li>
        <li><nuxt-link to="/user">个人中心</nuxt-link></li>
        <li><nuxt-link to="/user/one">UserOne</nuxt-link></li>
      </ul>
    </div>
    <!-- 主体内容 -->
    <nuxt />
  </div>
</template>
```

5. 请求 http://localhost:8000/hot



针对 `path: "/:question/view"` 也是一样的，请求 http://localhost:8000/hot/view

嵌套路由

1. 创建内嵌子路由，你需要添加一个 Vue 文件，同时添加一个**与该文件同名**的目录用来存放子视图组件。
2. 在父组件(`.vue` 文件) 内增加 `<nuxt-child/>` 用于显示子视图内容。

比如：

针对首页 pages/index.vue 添加子路由，则文件结构如下：

```
pages/
--| index/ # 创建一个 index 同名的目录名
-----| _id.vue # 子路由组件
--| index.vue
```

Nuxt.js 自动生成的路由配置如下：

```
router: {
    routes: [
    // http://localhost:8000/
    {
      path: "/",
      component: 'pages/index.vue',
      name: "index",
      children: [{
        // ?表示可传可不传，如果一定要，则index目录下创建一个 index.html
        // http://localhost:8000/1
        path: ":id?",
        component:"pages/index/_id.vue",
        name: "index-id"
      }]
    }

  ]
}
```

第八章 中间件

什么是中间件

> 中间件允许您定义一个自定义函数运行在一个页面或一组页面渲染之前。可用于权限判断，有权限才可访问对应页面。

每一个中间件应放置在 `middleware/` 目录。文件名的名称将成为中间件名称 (`middleware/auth.js` 将成为 `auth` 中间件)。

在 `nuxt.config.js` 、 layouts 或者 pages 中使用中间件。

创建权限中间件

1. 创建 nuxt-demo2\middleware\auth.js ， 其中文件名 auth 就是中间件名称 。

   一个中间件接收 context 作为第一个参数。

```
export default ({ store, redirect }) => {
    console.log('auth.js认证中间件被执行')
    // 有store实例，且有userInfo
    if(!store || !store.state.userInfo) {
        return redirect("/")
    }
}
```

使用中间件

在 `nuxt.config.js` 、 layouts 或者 pages 中使用中间件。

1. 在 nuxt-demo2\pages\user\index.vue 页面组件中使用中间件，

```
<script>
    export default {
        middleware: 'auth'
    }
</script>
```

浏览器访问 http://localhost:3000/user ， 重写向到首页 http://localhost:3000/

2. 在 nuxt-demo2\layouts\default.vue 默认布局中使用中间件

```
<script>
export default {
    middleware: 'auth'
}
</script>
```

只要引用了这个布局的页面组件，当浏览器访问时都先执行中间件。

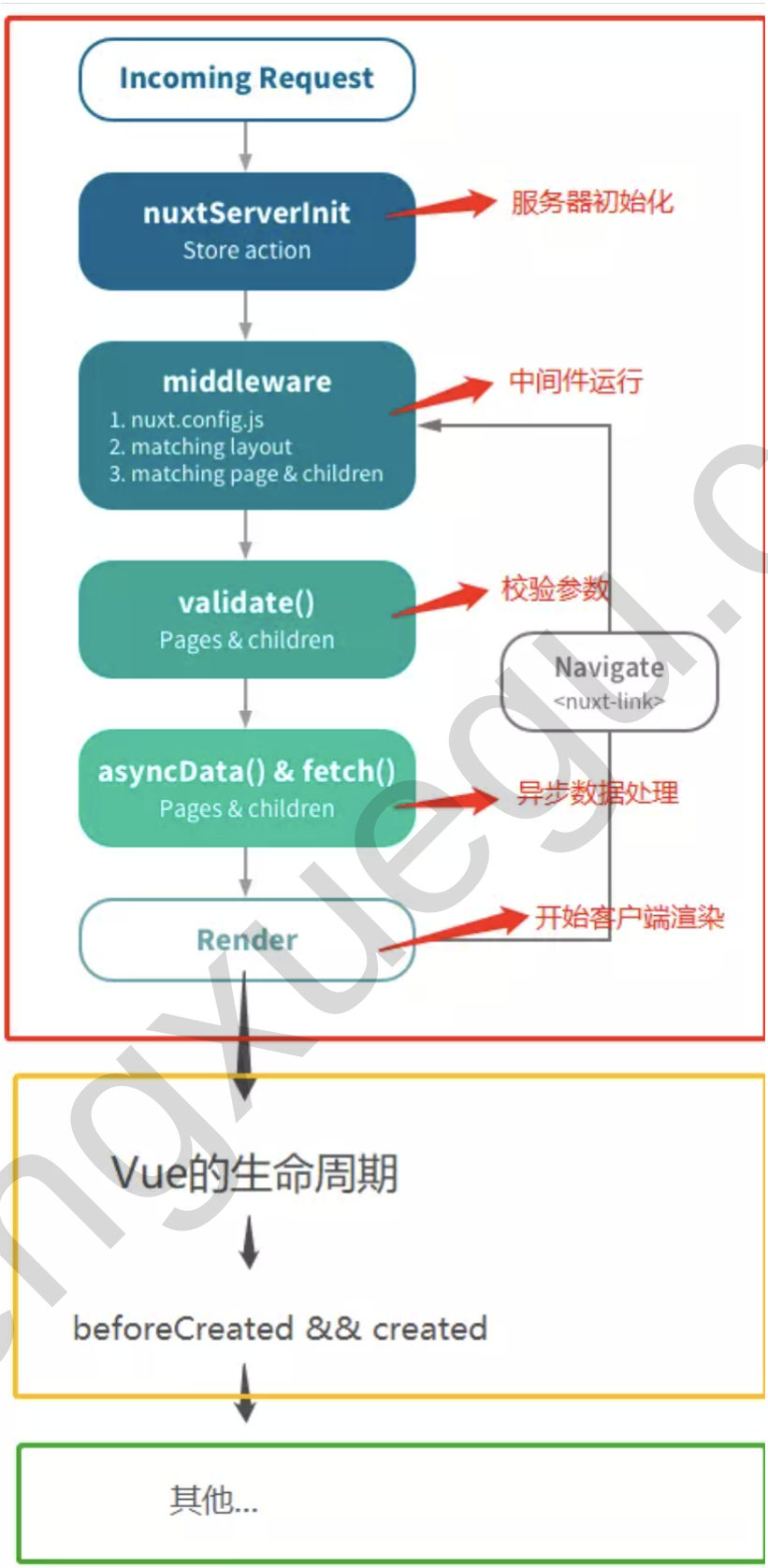
3. 在 `nuxt.config.js` 中使用中间件

```
module.exports = {
  router: {
    middleware: 'auth'
  }
}
```

`auth` 中间件在每个路由改变时被调用 。

Nuxt 生命周期流程图分析

下图是 Nuxt.js 应用一个完整的服务器请求到渲染（或用户通过 `<nuxt-link>` 切换路由渲染页面）的流程：



分析上面 Nuxt 流程图：

红框内的是Nuxt的生命周期(运行在服务端)，黄框内同时运行在服务端&&客户端上，绿框内则运行在客户端

红框、黄框周期内的都不存在Window对象。

- 当一个客户端请求进入的时候，服务端有通过 `nuxtServerInit` 这个命令执行在 `Store` 的 `action` ，在这里接收到客户端请求的时候，可以将一些客户端信息存储到 `Store` 中，也就是说可以把在服务端存储的一些客户端登录信息存储到 `Store` 中。
- 接着使用了 `中间件` 机制，中间件其实就是一个函数，会在每个路由执行之前去执行，在这里可以做很多事情，或者说可以理解为是路由器的拦截器的作用。
- 再接着使用 `validate` 对客户端携带的参数进行校验。
- 然后在 `asyncData` 与 `fetch` 进入正式的渲染周期， `asyncData` 向服务端获取数据，把请求到的数据合并到 `Vue` 中的 `data` 中。
- 最后 Render 将渲染组件。