Nuxt

> 讲师：李忠国

一、客户端和服务器端渲染

1. 客户端渲染

客户端 vs 服务器端
客户端
- 优点：减轻了服务器的压力，爬虫爬取不到页面数据
- 缺点：因为没有初始源代码数据，百度等搜索引擎抓取到的没有数据的空页面，不利于SEO优化

服务器端
- 优点：解决SEO优化
- 缺点：会被爬虫爬取到页面数据

客户端渲染的特点

- `JS` 代码在浏览器中运行
- 查看源代码时，无法查看渲染出来的数据（因为数据是 `JS` 运行之后渲染的，所以页面无初始数据）

2. SEO

> SEO(搜索引擎优化)：让我们的网站，在百度，谷歌里排到前面

SEO原理

"爬虫"程序抓取每个页面核心数据到自己的数据库，当用户搜索关键字，会和数据库中数据进行匹配，匹配度好就排在前面，但注意它抓取时页面不会执行 `JS`

3. 服务器端渲染

服务器端渲染(SSR: ServerSide Render)：在服务端运行的Vue代码，然后将结果返回给前端

> 用途：解决SEO
> 特点：Vue代码在服务器端运行，将结果返回给前端，前端源代码中可以查看到数据

4. Nuxt.js

- `Nuxt.js` 是一个基于 `Vue.js` 的**通用**[1]应用框架
- 通过对 客户端/服务端基础架构的抽象组织，`Nuxt.js` 主要关注的是应用的 **UI 渲染**
- `Nuxt.js` 预设了利用 `Vue.js` 开发**服务端渲染**的应用所需要的各种配置。

二、Nuxt安装

1. 安装指令

```
1. npx create-nuxt-app <项目名>
#or
2. yarn create nuxt-app <项目名>
```

2. 运行

```
1. cd 项目目录
2. yarn dev / npm run dev

注意：
   开发时：
        yarn dev         npm run dev         ---> 开启开发服务器

   开发完：
        yarn build       npm run build       ---> 打包
        yarn start       npm run start       ---> 开启正式服务器

        yarn generate    npm run generate    ---> 生成静态页
                特点：    1. 生成dist
                         2. 生成.html的页面
```

三、脚手架项目结构

1. 目录结构

目录名称	描述
assets	资源目录，用于存放**需要编译**的静态资源。例如：LESS、SASS等 默认情况下，Nuxt使用Webpack若干加载器处理目录中的文件
components	vue组件目录，Nuxt.js 不会增强该目录，及不支持SSR
layouts	布局组件目录
pages	页面目录，所有的vue视图，nuxt根据目录结构自动生成对应的路由。
middleware	中间件目录
plugins	插件目录
static	静态文件目录，**不需要编译**的文件
store	vuex目录
nuxt.config.js	nuxt个性化配置文件，内容将覆盖默认
package.json	项目配置文件

2. 页面组成

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

  > 注：别忘了在布局文件中添加 `<nuxt/>` 组件用于显示页面的主体内容。

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

3. 自定义布局

> `layouts` 目录中的每个文件 (顶级) 都将创建一个可通过页面组件中的 `layout` 属性访问的自定义布局。

```
<!-- layouts/blog.vue -->
<template>
  <div>
    <div>博客导航栏在这里</div>
    <!-- page页面内容 -->
    <nuxt />
  </div>
</template>
```

```
<!-- pages/posts.vue -->
<template>
  <!-- Your template -->
</template>
<script>
  export default {
    layout: 'blog'  //指定当前页面的布局
    // page component definitions
  }
</script>
```

4. 错误页面

> 通过编辑 `layouts/error.vue` 文件来定制化错误页面。
> 注：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)**。

```
<!-- layouts/error.vue -->
<template>
  <div class="container">
    <h1 v-if="error.statusCode === 404">页面不存在</h1>
    <h1 v-else>应用发生错误异常</h1>
    <nuxt-link to="/">首 页</nuxt-link>
  </div>
</template>

<script>
  export default {
    props: ['error'],
```

```
    layout: 'blog'  //可以为错误页面指定自定义的布局
  }
</script>
```

四、路由

- `Nuxt` 中已经内置了vue-router组件，所以可以直接使用，而不需要写任何代码
- `Nuxt` 中不需要自己配置路由，`Nuxt` 会根据 `pages` 目录中的文件结构自动生成路由的配置

1. 基础路由

> 要在页面之间使用路由，我们建议使用 `nuxt-link` 标签。

```
<template>
  <nuxt-link to="/">首页</nuxt-link>
</template>
```

```
/* 假设 pages 的目录结构如下 */
pages/
--| user/
-----| index.vue
-----| one.vue
--| index.vue
```

```
/* Nuxt.js 自动生成的路由配置如下 */
routes: [
  {
    name: 'index',
    path: '/',
    component: 'pages/index.vue'
  },
  {
    name: 'user',
    path: '/user',
    component: 'pages/user/index.vue'
  },
  {
    name: 'user-one',
    path: '/user/one',
    component: 'pages/user/one.vue'
  }
]
```

2. 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。

```
pages/
--| _slug/
-----| comments.vue
-----| index.vue
--| users/
-----| _id.vue
--| index.vue
```

```
/* Nuxt.js 生成对应的路由配置表为 */
routes: [
  {
    name: 'index',
    path: '/',
    component: 'pages/index.vue'
  },
  {
    name: 'users-id',
    path: '/users/:id?',  //代表该路由是可选的
    component: 'pages/users/_id.vue'
  },
  {
    name: 'slug',
    path: '/:slug',
    component: 'pages/_slug/index.vue'
  },
  {
    name: 'slug-comments',
    path: '/:slug/comments',
    component: 'pages/_slug/comments.vue'
  }
]
```

注：名称为 `users-id` 的路由路径带有 `:id?` 参数，表示该路由是可选的。如果想将它设置为必选的路由，需要在 `users/_id` 目录内创建一个 `index.vue` 文件。

> 警告：`generate` 命令会忽略动态路由: API Configuration generate

3. 路由参数校验

> `Nuxt.js` 可以让你在动态路由组件中定义参数校验方法。

```
// 例：ages/users/_id.vue
export default {
  validate({ params }) {
    // 必须是number类型
    return /^\d+$/.test(params.id)
  }
}
```

如果校验方法返回的值不为 `true` 或 `Promise` 中 `resolve` 解析为 `false` 或抛出 `Error`， `Nuxt.js` 将自动加载显示 `404` 错误页面或 `500` 错误页面。

4. 嵌套路由

> 可以通过 `vue-router` 的子路由创建 `Nuxt.js` 应用的嵌套路由。
> 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。

警告：别忘了在父组件(`.vue` 文件) 内增加 `nuxt-child/` 用于显示子视图内容。

```
pages/
--| users/
-----| _id.vue
-----| index.vue
--| users.vue
```

```
/* Nuxt.js 自动生成的路由配置如下 */
routes: [
  {
    path: '/users',
    component: 'pages/users.vue',
    children: [
      {
        path: '',
        component: 'pages/users/index.vue',
        name: 'users'
      },
      {
        path: ':id',
        component: 'pages/users/_id.vue',
        name: 'users-id'
      }
    ]
  }
]
```

5. 动态嵌套路由

> 这个应用场景比较少见，但是 Nuxt.js 仍然支持：在动态路由下配置动态子路由。

```
pages/
--| _category/
-----| _subCategory/
--------| _id.vue
--------| index.vue
-----| _subCategory.vue
-----| index.vue
--| _category.vue
--| index.vue
```

```
/* Nuxt.js 自动生成的路由配置如下 */
routes: [
  {
    path: '/',
    component: 'pages/index.vue',
    name: 'index'
  },
  {
    path: '/:category',
    component: 'pages/_category.vue',
    children: [
      {
        path: '',
        component: 'pages/_category/index.vue',
        name: 'category'
      },
      {
        path: ':subCategory',
        component: 'pages/_category/_subCategory.vue',
        children: [
          {
            path: '',
            component: 'pages/_category/_subCategory/index.vue',
            name: 'category-subCategory'
          },
          {
            path: ':id',
            component: 'pages/_category/_subCategory/_id.vue',
            name: 'category-subCategory-id'
          }
        ]
      }
    ]
  }
]
```

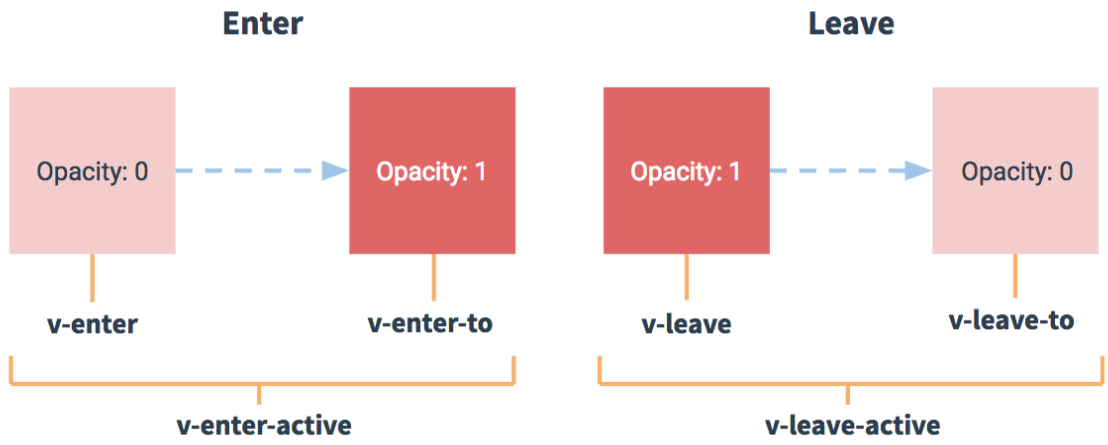
6. 过渡和动画

> `Nuxt.js` 使用 `Vue.js` 的 组件来实现路由切换时的过渡动效。

提示：Nuxt.js 默认使用的过渡效果名称为 `page`

6.1 `transition` 属性

> *Nuxt.js* 使用 *Vue.js* 的 组件来实现路由切换时的过渡动效 详见更多
>
> - **类型：** `String` 或 `Object` 或 `Function`
> - `layoutTransition` 属性：设置布局转换的默认属性（默认 `page`）




```
// 自定义过渡特效：给指定页面配置 transition 字段
export default {
  // 可以是字符
  transition: ''
  // 或对象
  transition: {}
  // 或函数
  transition (to, from) {}
}
```

6.2 全局过渡效动效设置

```
/* 例：在全局样式文件 assets/main.css 里添加 */
.page-enter-active, .page-leave-active {
  transition: opacity 0.5s;
}
.page-enter, .page-leave-active（or .page-leave-to） {
  opacity: 0;
}
```

```
module.exports = {
  css: ['assets/main.css']
}
```

6.3 页面过渡动效设置

```
// 例：将页面组件中的 transition 属性的值设置为 test
export default {
  transition: 'test'
}
```

```
.test-enter-active, .test-leave-active {
  transition: opacity 0.5s;
}
.test-enter, .test-leave-active {
  opacity: 0;
}
```

五、asyncData

> 获取的数据会显示到页面源代码中，有利于SEO

1. 使用时机

- 只能在页面文件中使用（page目录下）
- 获取页面初始化异步数据时使用

2. 特点

- 需要 `return` 一个数据，然后这个数据可以在页面中使用
- 有提供 `context`，包含 `store`,`query`,`params`,`route`,`redirect`等
- 可以在服务端或路由更新之前被调用
  - `asyncData` 函数**默认在服务端渲染**
  - `asyncData` 函数在当前所在页面刷新后**在服务端渲染**
  - `asyncData` 函数在路由跳转时**在客户端渲染**（可通过 `process.server`[2] 判断）

3. 使用方式

3.1 普通 `return`

> 用法类似 `data`，`Nuxt.js` 会将 `asyncData` 返回的数据融合组件 `data` 方法返回的数据一并返回给当前组件。

```
<template>
     <div class="home">
         <h1>{{ msg }}</h1>
     </div>
</template>

<script>
export default {
    asyncData () {
        return {
```

```
            msg:'Hello Nuxt.js~'
        }
    }
}
</script>
```

3.2 返回 `promise`

> `Nuxt.js` 会等待 `Promise` 被解析之后才会设置组件的数据，从而渲染组件。
> 接口示例：https://cnodejs.org/api/v1/topics

```
<template>
  <div class="home">
    <ul>
      <li v-for="item in list" :key="item.id">
        <a href="#">{{ item.title }}</a>
      </li>
    </ul>
  </div>
</template>

<script>
import axios from "axios";
export default {
  async asyncData() {
    // 解构data中的data，赋予别名list
    const { data: {data: list} } = await
axios.get("https://cnodejs.org/api/v1/topics");
    return {
      list,
    };
  },
};
</script>
```

4. 服务器端运行

> `Nuxt` 为了实现 `SSR`，在原Vue中添加了一些额外的功能，这些功能都会在服务器端执行。

功能	执行的位置
中间件	服务器 或者 路由更新
asyncData	服务器 或者 路由更新
fetch	服务器 或者 路由更新
beforeCreated, created	服务器 或者 路由更新
nuxtServerInt	服务器端

4.1 区分代码执行位置

- `console.log()`
  为了区分代码执行的位置，`Nuxt` 会把服务器执行的代码输出到 `Nuxt SSR` 中，可以在浏览器的工具中查看
- `process.server`[2]
  通过其返回的布尔值可以判断出当前的环境，针对不同的环境（客户端or服务器端）执行不同的代码

六、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。

> 概念：就是一个函数，会在每一次请求路由之前被执行。
> 应用场景：可以用来做权限校验等功能（加载页面之前执行）

```
// 一个中间件接收 context 作为第一个参数
export default function (context) {
  context.userAgent = process.server
    ? context.req.headers['user-agent']
    : navigator.userAgent
}
```

中间件执行流程顺序：

- `nuxt-config-js`
- 匹配布局
- 匹配页面

```
// 中间件可以异步执行,只需要返回一个 Promise 或使用async/await:
/* middleware/stats.js */
import axios from 'axios'

export default function ({ route }) {
  return axios.post('http://my-stats-api.com', {
    url: route.fullPath
  })
}
```

全局配置

```
/* nuxt.config.js */
module.exports = {
  router: {
    middleware: 'stats'
  }
}
```

布局或页面配置

```
export default {
  middleware: ['auth', 'stats']
}
```

匿名中间件

> 若只需要为特定页面使用中间件，则可以直接为其使用一个函数（或函数数组）：

```
<template>
  <h1>Secret page</h1>
</template>

<script>
  export default {
    middleware({ store, redirect }) {
      // If the user is not authenticated
      if (!store.state.authenticated) {
        return redirect('/login')
      }
    }
  }
</script>
```

七、资源文件

1. css/html预处理器

> 目的：为了更高效或者简洁的编写 `css`，`html` 代码，一般会使用 `css` 或者 `html` 预处理器。

- `css` 常见预处理器
  - less
  - sass
  - stylus
- `html` 预处理器
  - pug

1.1 less/css

- (1) 安装依赖包(nuxt内置webpack4)

  ```
  yarn add -D less less-loader@7.3.0  #指定版本
  ```

- (2) 创建公共类库文件

```
// 例：assets/less/base.less
*{
    margin: 0;
    padding: 0;
    color: @globalColor;
}
```

- (3) 创建全局变量文件

```
// 例：assets/less/variables.less
@globalColor: #222;
```

- (4) 安装 `@nuxtjs/style-resources`

```
yarn add -D @nuxtjs/style-resources
```

- (5) 配置 `nuxt.config.js`

```
css: [
    '~/assets/less/base.less'
],
buildModules: [
    '@nuxtjs/style-resources'
],
styleResources: {
    less: [
        '~/assets/less/variables.less'
    ]
},
```

1.2 scss/css

- (1) 安装依赖包(nuxt内置webpack4)

```
yarn add -D node-sass sass-loader@10  #指定版本
```

- (2) 创建公共类库文件

```
// 例：assets/scss/base.scss
*{
    margin: 0;
    padding: 0;
    color: $globalColor;
}
```

- (3) 创建全局变量文件

```
// 例：assets/scss/variables.scss
$globalColor: #222;
```

- (4) 安装 `@nuxtjs/style-resources`

```
yarn add -D @nuxtjs/style-resources
```

- (5) 配置 `nuxt.config.js`

```
css: [
    '~/assets/scss/base.scss'
],
buildModules: [
    '@nuxtjs/style-resources'
],
styleResources: {
    less: [
        '~/assets/scss/variables.scss'
    ]
},
```

1.3 pug/html

> pug：`html` 模板引擎，作用是让 `html` 代码更加精简

- 安装

```
yarn add -D pug@2.0.3 pug-plain-loader
```

- 使用 [中文文档]

```
<!-- 例：pug结合vue使用 -->
<template lang="pug">
  ul
    li(v-for="item in list" :key="item")
      {{item}}
</template>

<script>
export default {
  data() {
    return {
      list: [1, 2, 3, 4, 5]
    }
  }
};
</script>
```

2. assets

- `assets` 目录下的图片，less，scss等资源会被 `webpack` 编译
- 行内样式使用图片时，不会被 `webpack` 编译，需要手动使用 `require` 加载图片

```
<template>
  <div class="home">
    <h3>使用img标签显示图片</h3>
```

```
    <img src="~/assets/imgs/1.jpg" alt="" />

    <h3>使用类名显示图片</h3>
    <div class="pic"></div>

    <h3>使用行内样式显示图片</h3>
    <div class="stylePic"
:style="`background:url(${require('~/assets/imgs/3.jpg')})`"></div>
  </div>
</template>

<style  scoped>
img {
  width: 200px;
}
.pic {
  width: 200px;
  height: 200px;
  background: url(~/assets/imgs/2.jpg) no-repeat 100%;
}
.stylePic {
  width: 200px;
  height: 200px;
}
</style>
```

3. static

- `Nuxt` 直接使用资源文件，不做任何处理

```
<template>
  <div class="home">
    <h3>使用img标签显示图片</h3>
    <img src="imgs/1.jpg" alt="" />

    <h3>使用类名显示图片</h3>
    <div class="pic"></div>

    <h3>使用行内样式显示图片</h3>
    <div class="stylePic"
:style="`background:url(${require('~/static/imgs/3.jpg')})`"></div>
  </div>
</template>

<style scoped>
img {
  width: 200px;
}
.pic {
  width: 200px;
  height: 200px;
  /* 类名加载static下的图片需要补全路径 */
  background: url(~/static/imgs/2.jpg) no-repeat 100%;
}
.stylePic {
```

```
  width: 200px;
  height: 200px;
}
</style>
```

八、插件

> *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。

1. 客户端和服务端插件

> 默认插件，在客户端和服务器端都会自动执行

```
// 自定义两端插件  默认会在'服务端'和'客户端'都执行
/* both.js */
export default () => {
    const envm = process.server ? '服务端' : '客户端'
    console.log("自定义两端插件", `当前环境：${envm}`);
}
```

```
plugins: [
    '@/plugins/both.js'
]
```

2. 指定端插件

2.1 通过变量判断

> 可以通过检测`process.server`这个变量来控制插件中的某些脚本库只在服务端使用。当值为`true`表示是当前执行环境为服务器中。此外，可以通过检查`process.static`是否为`true`来判断应用是否通过`nuxt generator`生成。您也可以组合`process.server`和`process.static`这两个选项，确定当前状态为服务器端渲染且使用`nuxt generate`命令运行。

2.2 配置指定端插件

> 设置`mode`属性，`client`代表客户端;`server`代表服务端

```
export default {
  plugins: [
    { src: '~/plugins/both-sides.js' },
    { src: '~/plugins/client-only.js', mode: 'client' },
    { src: '~/plugins/server-only.js', mode: 'server' }
  ]
}
```

2.3 传统命名指定插件

> 通过对插件文件进行规范命名：`xxx.server.js` or `xxx.client.js`

```
export default {
  plugins: [
    '~/plugins/foo.client.js', // only in client side
    '~/plugins/bar.server.js', // only in server side
    '~/plugins/baz.js' // both client & server
  ]
}
```

3. vue插件

```
// 例：显示应用的通知信息，需要在程序运行前配置好这个插件
/* plugins/vue-tooltip */
import Vue from 'vue'
import tooltip from 'v-tooltip';

Vue.use(tooltip)
```

```
/* nuxt.config.js */
module.exports = {
  plugins: ['~/plugins/vue-tooltip']
}
```

```
<!-- 使用示例：v-tooltip -->
<template>
  <div class="home">
    <h2 v-tooltip="msg">鼠标悬浮显示提示</h2>
  </div>
</template>

<script>
export default {
  data() {
    return {
      msg: "hello Nuxt👏",
    };
  },
};
</script>
```

ES6 插件

> 如果插件位于`node_modules`并导出模块，需要将其添加到`transpile`构建选项：参考构建配置

```
module.exports = {
  build: {
    transpile: ['vue-notifications']
  }
}
```

4. 注入插件（$root 和 context）

> 有时希望在整个应用程序中使用某个函数或属性值，需要将它注入到 `Vue实例`（客户端），`context`（服务器端）甚至 `store(Vuex)`。（惯例：新增的属性或方法名使用 `$` 作为前缀）

4.1 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.$myInjectedFunction = (string) => {
    console.log('This is an example', string)
}
```

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

```
/* 使用示例：example-component.vue */
export default {
  mounted() {
    this.$myInjectedFunction('test')
  }
}
```

4.2 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) => {
  // Set the function directly on the context.app object
  app.myInjectedFunction = string =>
    console.log('Okay, another function', string)
}
```

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

```
/* 使用示例：ctx-example-component.vue: */
export default {
  // 有context皆可使用，例如`asyncData`和`fetch`
  asyncData(context) {
    context.app.myInjectedFunction('ctx!')
  }
}
```

4.3 同时注入

> 在`context`，`Vue`实例，甚至`Vuex`中同时注入，可以使用`inject`方法，它是 plugin 导出函数的第二个参数。系统会自动将`$`添加到方法名的前面。

```
/* plugins/axios.js */
import axios from "axios"

export default ({ app }, inject) => {
  axios.defaults.baseURL = 'https://cnodejs.org/api/v1'
  inject('api', {
    getData(path) {
        return axios.get(path)
    }
  })
}
```

```
/* nuxt.config.js */
export default {
  plugins: ['~/plugins/axios.js']
}
```

```
/* ctx-example-component.vue */
export default {
  async mounted(){
    await this.$api.getData('/topics')
  },
  async asyncData({ app}) {
    const {
      data: { data: topics },
    } = await app.$api.getData("/topics");
    return {
      topics,
    };
  },
};
```

5. ui-vant插件

> UI组件采用插件的方式配置，以 `vant` 为例

```
/* plugins/vant.js */
import Vue from 'vue';
// import vant from 'vant';
import {Button,Grid,GridItem} from 'vant';
import 'vant/lib/index.css';
Vue.use(Button);
Vue.use(Grid);
Vue.use(GridItem);
```

```
/* vant.js */
plugins: [
 '~/plugins/vant'
],
```

九、Vuex

> 对于每个大项目来说，使用状态树 *(store)* 管理状态 *(state)* 十分有必要。

1. 使用状态树

- 引用 `vuex` 模块
- 将 `vuex` 模块 加到 `vendors` 构建配置中去
- 设置 `Vue` 根实例的 `store` 配置项
- 目录中的每个 `.js` 文件都被 `store` 转换为一个 `命名空间模块` （`index` 作为根模块）您的 `state` 值应始终为 a `function` 以避免 在服务器端出现不需要的 共享状态

2. 普通方式

> `Nuxt.js` 允许您拥有一个 `store` 目录，其中包含与模块对应的每个文件。
> 状态导出为函数，变量和操作作为对象导出。`Nuxt` 将会自动创建配置 `Vuex`

```
/* store/index.js */
const state = () => ({
  count: 0
})

const mutations = {
  incremetn(state, payload) {
    state.count += payload
  }
}

const actions = {}
const getters = {}

export {state, mutations, actions, getters}
```

```
/* store/todos.js */
export const state = () => ({
  list: []
})

export const mutations = {
  add(state, text) {
    state.list.push({
      text,
      done: false
    })
  },
  remove(state, { todo }) {
    state.list.splice(state.list.indexOf(todo), 1)
  },
  toggle(state, todo) {
    todo.done = !todo.done
  }
}
```

```
<template>
  <ul>
    <li v-for="todo in todos">
      <input type="checkbox" :checked="todo.done" @change="toggle(todo)" />
      <span :class="{ done: todo.done }">{{ todo.text }}</span>
    </li>
    <li>
      <input placeholder="What needs to be done?" @keyup.enter="addTodo" />
    </li>
  </ul>
</template>

<script>
  // 导入vuex辅助方法
  import { mapState, mapMutations } from 'vuex'

  export default {
    computed: {
      todos() {
        return this.$store.state.todos.list
      }
    },
    methods: {
      addTodo(e) {
        this.$store.commit('todos/add', e.target.value)
        e.target.value = ''
      },
      ...mapMutations({
        toggle: 'todos/toggle'
      })
    }
  }
</script>

<style>
  .done {
```

```
    text-decoration: line-through;
  }
</style>
```

3. 模块文件

> 将模块文件分解为单独的文件：`state.js`，`mutations.js`，`actions.js` 和 `getters.js`
> 注意：在使用拆分文件模块时，必须记住使用**箭头函数功能**，`this` 在词法上可用。

```
/* 复杂的商店设置文件/文件夹结构可能如下所示 */
store/
--| index.js
--| ui.js
--| shop/
----| cart/
------| actions.js
------| getters.js
------| mutations.js
------| state.js
----| products/
------| mutations.js
------| state.js
------| itemsGroup1/
--------| state.js
```

4. 插件

> 可以将其他插件添加到 `store`（在模块模式下），将其放入 `store/index.js` 文件中 详见更多

```
import myPlugin from 'myPlugin'

export const plugins = [myPlugin]

export const state = () => ({
  counter: 0
})

export const mutations = {
  increment(state) {
    state.counter++
  }
}
```

5. nuxtServerInit 方法

> `nuxtServerInit` 作为一个 `Action`，Nuxt.js 调用它会将页面的上下文对象作为第2个参数传给它（仅在服务端）。当想从服务端传递数据给客户端时，它很好用。

```
// 例：假设在服务器端有会话，可以通过访问连接的用户 `req.session.user`。要将经过身份验证的用户添加到我们的商店，store/index.js 代码如下：
actions: {
  nuxtServerInit ({ commit }, { req }) {
    if (req.session.user) {
      commit('user', req.session.user)
    }
  }
}
```

注意：只有主模块（在 `store/index.js` 中）会收到此操作。您需要从那里链接您的模块操作。

案例：用户登录Cookie操作

```
<!-- pages/login.vue -->
<template>
  <div>
    <h2>【请登录】</h2>
    <button @click="login">登录</button>
  </div>
</template>

<script>
// 导入js-cookie
import Cookie from 'js-cookie'

export default {
  methods: {
    // 登录
    login() {
      setTimeout(() => {
        const auth = 'qwertyuisdfghjzxcvbn'
        // 保存用户信息
        this.$store.commit('updateAuth', auth)
        // 设置Cookie
        Cookie.set('auth', auth)
        // 跳转首页
        this.$router.push('/')
      }, 1000)
    }
  }
}
</script>

<style>
</style>
```

```
/* store/index.js */
// 导入cookie解析包
const cookieparser = require('cookieparser')

const store = () => ({
    auth: ''
})
```

```
const mutations = {
    updateAuth(state, payload) {
        state.auth = payload
    }
}

const actions = {
    // 只执行一次（服务端或页面刷新）
    nuxtServerInit({ commit }, { req }) {
        let auth = ''
        // console.log(req.headers.cookie, 'cookie');
        if (req.headers.cookie) {
            // 解析出cookie
            const parser = cookieparser.parse(req.headers.cookie)
            // 将服务端数据存储到客户端
            auth = parser.auth
        }
        commit('updateAuth', auth)
    }
}

export { store, mutations, actions }
```

十、Loading

1. 自定义进度条

```
/* 例如，要设置一个高度为 5px 的蓝色进度条，我们将其更新 nuxt.config.js 为以下内容 */
export default {
  loading: {
    color: 'blue',
    height: '5px'
  }
}
```

Key	Type	Default	Description
color	String	'black'	CSS color of the progress bar
failedColor	String	'red'	CSS color of the progress bar when an error appended while rendering the route (if data or fetch sent back an error for example).
height	String	'2px'	Height of the progress bar (used in the style property of the progress bar)
throttle	Number	200	In ms, wait for the specified time before displaying the progress bar. Useful for preventing the bar from flashing.
duration	Number	5000	In ms, the maximum duration of the progress bar, Nuxt assumes that the route will be rendered before 5 seconds.
continuous	Boolean	false	Keep animating progress bar when loading takes longer than duration.
css	Boolean	true	Set to false to remove default progress bar styles (and add your own).
rtl	Boolean	false	Set the direction of the progress bar from right to left.

2. 禁用进度条

```
/* nuxt.config.js全局设置禁用 */
export default {
  loading: false
}
```

```
<!-- 给特定页面设置禁用 -->
<template>
  <h1>My page</h1>
</template>

<script>
  export default {
    loading: false
  }
</script>
```

3. 编程方式启动

```
export default {
  mounted() {
    this.$nextTick(() => {
      this.$nuxt.$loading.start()
      setTimeout(() => this.$nuxt.$loading.finish(), 500)
    })
  }
}
```

4. 自定义加载组件

方法	必需的	描述
start()	Required	Called when a route changes, this is where you display your component.
finish()	Required	Called when a route is loaded (and data fetched), this is where you hide your component.
fail(error)	Optional	Called when a route couldn't be loaded (failed to fetch data for example).
increase(num)	Optional	Called during loading the route component, num is an Integer < 100.

```
<!-- components/LoadingBar.vue -->
<template>
  <div v-if="loading" class="loading-page">
    <p>Loading...</p>
  </div>
</template>

<script>
  export default {
    data: () => ({
      loading: false
    }),
    methods: {
      start() {
        this.loading = true
      },
      finish() {
        this.loading = false
      }
    }
  }
</script>

<style scoped>
  .loading-page {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
```

```
    background: rgba(255, 255, 255, 0.8);
    text-align: center;
    padding-top: 200px;
    font-size: 30px;
    font-family: sans-serif;
  }
</style>
```

```
/* nuxt.config.js */
export default {
  loading: '~/components/LoadingBar.vue'
}
```

5. 加载指示器属性

> 在 SPA 模式下运行 Nuxt 时，第一页加载时没有来自服务器端的内容。因此，Nuxt 不会在页面加载时显示空白页面，而是为您提供了一个微调器，您可以对其进行自定义以添加自己的颜色或背景，甚至更改指示器。

```
/* nuxt.config.js */
export default {
  loadingIndicator: {
    name: 'circle',
    color: '#3B8070',
    background: 'white'
  }
}
```

十一、Head

1. 全局设置

```
export default {
  head: {
    title: 'my website title',
    meta: [
      { charset: 'utf-8' },
      { name: 'viewport', content: 'width=device-width, initial-scale=1' },
      {
        hid: 'description',
        name: 'description',
        content: 'my website description'
      },
      { name: 'keywords', content:"website keywords" }
    ],
    link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
  }
}
```

2. 本地设置

> 使用 `head` 作为一个对象或函数来设置标题和描述仅为主页。

```
<script>
export default {
  head: {
    title: 'Home page',
    meta: [
      {
        hid: 'description',
        name: 'description',
        content: 'Home page description'
      }
    ],
  }
}
</script>
```

```
<template>
  <h1>{{ title }}</h1>
</template>
<script>
  export default {
    data() {
      return {
        title: 'Home page'
      }
    },
    head() {
      return {
        title: this.title,
        meta: [
          {
            hid: 'description',
            name: 'description',
            content: 'Home page description'
          }
        ]
      }
    }
  }
</script>
```

3. 外部资源

- 全局设置

```
export default {
  head: {
    script: [
      {
        src:
'https://cdnjs.cloudflare.com/ajax/libs/jquery/3.1.1/jquery.min.js'
      }
```

```
    ],
    link: [
      {
        rel: 'stylesheet',
        href: 'https://fonts.googleapis.com/css?family=Roboto&display=swap'
      }
    ]
  }
}
```

- 本地设置

```
<template>
  <h1>About page with jQuery and Roboto font</h1>
</template>

<script>
  export default {
    head() {
      return {
        script: [
          {
            src:

 'https://cdnjs.cloudflare.com/ajax/libs/jquery/3.1.1/jquery.min.js'
          }
        ],
        link: [
          {
            rel: 'stylesheet',
            href: 'https://fonts.googleapis.com/css?
family=Roboto&display=swap'
          }
        ]
      }
    }
  }
</script>

<style scoped>
  h1 {
    font-family: Roboto, sans-serif;
  }
</style>
```

十二、fetch

> `fetch` 方法用于渲染页面前填充应用的状态树（`store`）数据，与 `asyncData` 类似，不同的是它无法设置组件数据。

```
<template>
  <h1>Stars: {{ $store.state.stars }}</h1>
</template>

<script>
  export default {
    fetch({ store, params }) {
      return axios.get('http://my-api/stars').then(res => {
        store.commit('setStars', res.data)
      })
    }
  }
</script>
```

Vuex

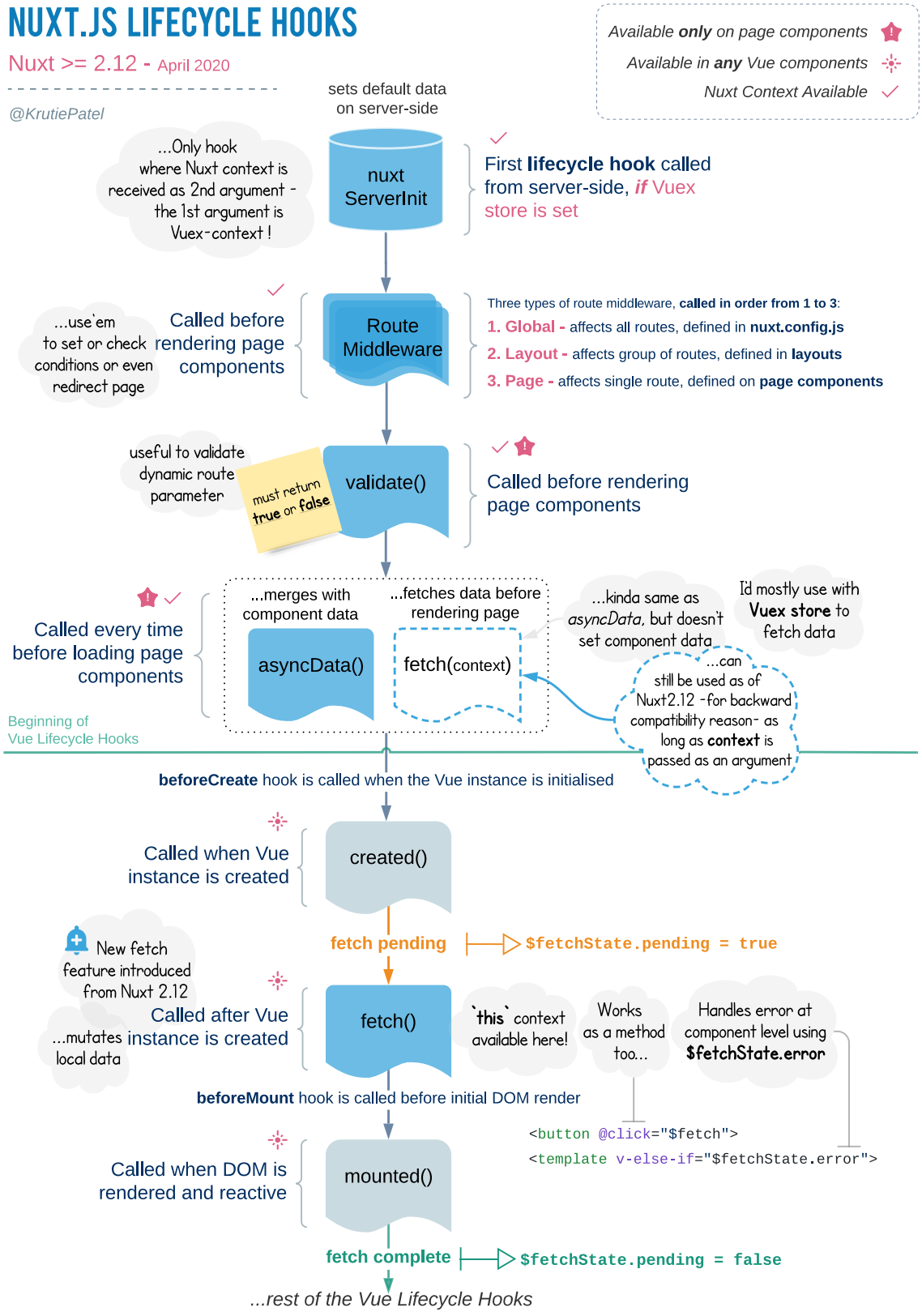
> 如果要在 `fetch` 中调用并操作 `store`，请使用 `store.dispatch`，但是要确保在内部使用 `async/await` 等待操作结束

```
<script>
  export default {
    async fetch({ store, params }) {
      await store.dispatch('GET_STARS')
    }
  }
</script>
```

```
/* store/index.js */
// ...
export const actions = {
  async GET_STARS({ commit }) {
    const { data } = await axios.get('http://my-api/stars')
    commit('SET_STARS', data)
  }
}
```

十三、生命周期

> nuxtServerInit → RouteMiddleware → validate() → asyncData(ctx) & fetch(ctx) → created() → fetch() → mounted()



十四、集成TypeScript和Composition-api

集成 TypeScript

- 下包

```
yarn add vue-property-decorator vue-class-component
```

- 基本应用

```
<template>
  <div>
    <h2>{{ count }}</h2>
    <button @click="increment">加</button>
    <button @click="decrement">减</button>
  </div>
</template>

<script lang="ts">
import { Vue, Component } from "vue-property-decorator";

@Component
export default class Index extends Vue {
  count: number = 1;

  increment() {
    this.count++;
  }

  decrement() {
    this.count--;
  }
}
</script>
```

集成 Composition-api

- 下包

```
yarn add @vue/composition-api
```

- 注册插件

```
/* plugins/composition-api.js */
import Vue from 'vue';
import VueComposition  from '@vue/composition-api'

Vue.use(VueComposition);
```

- 全局配置

```
/* nuxt.config.js */
plugins: [
  '@/plugins/componsition-api'
],
```

- 基本应用

```
<template>
  <div>
    <h2>{{ count }}</h2>
    <button @click="increment">加</button>
    <button @click="decrement">减</button>
  </div>
</template>

<script lang="ts">
import { defineComponent, ref } from "@vue/composition-api";

export default defineComponent({
  setup() {
    // 使用ref
    const count = ref(0);

    const increment = () => count.value++;
    const decrement = () => count.value--;

    return {
      count,
      increment,
      decrement,
    };
  },
});
</script>

<style>
</style>
```

十五、数据获取

> 在 Nuxt 中，我们有两种从 API 获取数据的方法。我们可以使用 fetch 方法或 asyncData 方法。

- `asyncData`. 这个钩子只能放在页面组件上。与 不同的是`fetch`，此钩子在客户端渲染期间不会显示加载占位符：相反，此钩子会阻止路由导航，直到解决为止，如果失败则显示页面错误。
- `fetch`（Nuxt 2.12+）。这个钩子可以放在任何组件上，并提供渲染加载状态（在客户端渲染期间）和错误的快捷方式。

数据请求内置模块

```
/* nuxt.config.js */
export default {
  http: {
    baseURL: 'https://cnodejs.org/api/v1'
  },
  axios: {
    baseURL: 'https://cnodejs.org/api/v1'
  }
}
```

`@nuxt/http`

> @nuxt/http：基于 ky-universal 的轻量级和通用的 `HTTP` 请求

- 安装

```
yarn add @nuxt/http -D
```

- 基本使用

```
<script>
export default {
  async asyncData({ $http }) {
    // console.log($http, 'http');
    const { data: list } = await $http.$get('/topics')
    console.log(list, 'server-list');

    return {
      list
    }
  },
}
</script>
```

`@nuxtjs/axios`

> @nuxtjs/axios：安全和使用简单 `Axios` 与 `Nuxt.js` 集成用来请求 `HTTP`

- 安装

```
yarn add @nuxtjs/axios -D
```

- 基本使用

```
<script>
export default {
  async asyncData({ $axios }) {
    // console.log($axios, 'axios');
    // const { data: { data: list } } = await $axios.get('/topics')
    const { data: list } = await $axios.$get('/topics')
    console.log(list, 'server-list');

    return {
      list
    }
```

```
  },
}
</script>
```

项目部署

Nginx

1. 下载&解压

> Nginx官网：http://nginx.org/en/download.html

2. 启动nginx

- `cmd` 进入 `nginx` 根目录
- 输入命名启动

```
nginx -c conf/nginx.conf
or
start nginx
```

- 访问网址，默认：http://localhost/

3. 常用命令

指令作用	指令
验证配置是否正确	`nginx -t`
查看Nginx的详细的版本号	`nginx -V`
查看Nginx的简洁版本号	`nginx -v`
启动Nginx	`start nginx`
快速停止或关闭Nginx	`nginx -s stop`
正常停止或关闭Nginx	`nginx -s quit`
配置文件修改重装载命令	`nginx -s reload`
载入指定配置文件	`nginx.exe -c conf/xxxx.conf`
测试配置文件	`nginx.exe -t -c conf/xxxx.conf`

4. 配置

> nginx配置文件目录下的 `conf/nginx.conf`

```
...

    server {
        listen       80;  #指定端口
        server_name localhost;

        location / {
            root    html;  #运行文件路径
            index   index.html index.htm;  #运行文件主页面
        }
    }
```

项目打包

> 1、静态项目 `yarn generate`
> 2、动态项目 `yarn build`

- 静态部署

  > 网站页面全部静态化
  > 优点：访问速度快 缺点：不能实时更新（可以周期性重新更新静态网站）
  >
  > **推荐使用路径参数（非查询参数（`?name=zs&age=8`）），打包时会将页面拆分打包**

- 动态部署

  > 开启一个node服务器，通过node启动项目 `yarn start`
  > 优点：数据实时更新 缺点：访问速度变慢，数据需要调接口动态渲染

---

1. 1.代替Vue客户端开发 2.服务器端开发 3.通过不过钩子函数,客户端和服务器端结合开发 ↩
2. true代表服务器端，false代表客户端 ↩ ↩