Vue3+Nuxt3(基础)

刘军　liujun

目录
content

什么是Nuxt?

- 在了解 Nuxt 之前，我们先来了解一下创建一个现代应用程序，所需的技术：
  - 支持数据双向绑定 和 组件化（ Nuxt 选择了Vue.js ）。
  - 处理客户端的导航（ Nuxt 选择了vue-router ）。
  - 支持开发中热模块替换和生产环境代码打包（ Nuxt支持webpack 5和Vite ）。
  - 兼容旧版浏览器，支持最新的 JavaScript 语法转译（ Nuxt使用esbuild ）。
  - 应用程序支持开发环境服务器，也支持服务器端渲染 或 API接口开发。
  - Nuxt 使用 h3来实现部署可移植性（h3是一个极小的高性能的http框架）
    - ✓ 如：支持在 Serverless、Workers 和 Node.js 环境中运行。

```
import { createServer } from 'http'
import { createApp, eventHandler, toNodeListener } from 'h3'

const app = createApp()
app.use('/', eventHandler(() => 'Hello world!'))

createServer(toNodeListener(app)).listen(process.env.PORT || 3000)
```

- Nuxt 是一个 **直观的 Web 框架**
  - 自 2016 年 10 月以来，Nuxt专门负责集成上述所描述的事情 ， 并提供前端和后端的功能。
  - Nuxt 框架可以用来快速构建下一个 Vue.js 应用程序，如支持 CSR 、SSR、SSG 渲染模式的应用等。

Nuxt 发展史

- Nuxt.js
  - 诞生于 2016 年 10 月 25号，由 Sebastien Chopin 创建，主要是基于Vue2 、Webpack2 、Node 和 Express。
  - 在2018 年 1 月 9 日， Sebastien Chopin 正式宣布，发布 Nuxt.js 1.0 版本。
    - ✓ 重要的变化是**放弃了对 node < 8 的支持**
  - 2018 年 9 月 21 日， ， Sebastien Chopin 正式宣布，发布 Nuxt.js 2.0 版本。
    - ✓ 开始使用 Webpack 4 及其技术栈， 其它的并没有做出重大更改。
  - 2021年8月12日至今，Nuxt.js 最新的版本为：Nuxt.js 2.15.8

Nitro

Build and Deploy Universal JavaScript Servers

- Nuxt3版本：
  - 经过 16 个月的工作，Nuxt 3 beta 于 2021 年 10 月 12 日发布，引入了基于 Vue 3、Vite 和 Nitro(服务引擎) 。
  - 六个月后， 2022 年 4 月 20 日，Pooya Parsa 宣布 Nuxt 3 的第一个候选版本，代号为 “Mount Hope”
  - 在2022年11月16号， Pooya Parsa 再次宣布 Nuxt3 发布为第一个正式稳定版本。

Announcing Nuxt 3.0 stable

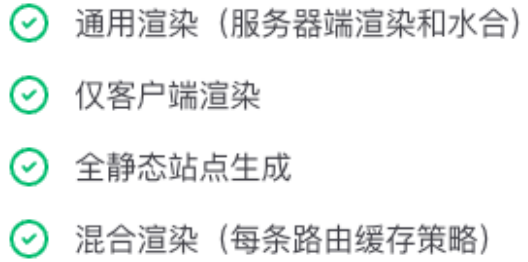
We are thrilled to announce the first stable version of Nuxt 3.0.0 ✨

Nov 16, 2022 Pooya Parsa

- 官网地址： https://nuxt.com/

Nuxt3 特点

- Vue技术栈
  - Nuxt3 是基于 Vue3 + Vue Router + Vite 等技术栈，全程 Vue3+Vite 开发体验（Fast）。
- 自动导包
  - Nuxt 会自动导入辅助函数、组合 API和 Vue API，无需手动导入。
  - 基于规范的目录结构，Nuxt 还可以对自己的组件、 插件使用自动导入。
- 约定式路由（目录结构即路由）
  - Nuxt 路由基于vue-router，在 pages/ 目录中创建的每个页面，都会根据目录结构和文件名来自动生成路由
- 渲染模式：Nuxt 支持多种渲染模式（SSR、CSR、SSG等）
- 利于搜索引擎优化：服务器端渲染模式，不但可以提高首屏渲染速度，还利于SEO
- 服务器引擎
  - 在开发环境中，它使用 Rollup 和 Node.js 。
  - 在生产环境中，使用 Nitro 将您的应用程序和服务器构建到一个通用.output目录中。
    - ✓ Nitro服务引擎提供了跨平台部署的支持，包括 Node、Deno、Serverless、Workers等平台上部署。



Nuxt.js VS Nuxt3

Feature / Version	Nuxt 2	Nuxt Bridge	Nuxt 3
Vue	2	2	3
Stability	😊 Stable	😌 Semi-stable	😊 Stable
Performance	🏎️ Fast	✈️ Faster	🚀 Fastest
Nitro Engine	❌	✅	✅
ESM support	🌙 Partial	👍 Better	✅
TypeScript	☑️ Opt-in	🚧 Partial	✅
Composition API	❌	🚧 Partial	✅
Options API	✅	✅	✅
Components Auto Import	✅	✅	✅
`<script setup>` syntax	❌	🚧 Partial	✅
Auto Imports	❌	✅	✅
webpack	4	4	5
Vite	⚠️ Partial	🚧 Partial	✅
Nuxi CLI	❌ Old	✅ nuxi	✅ nuxi
Static sites	✅	✅	✅

Nuxt3 环境搭建

- 在开始之前，请确保您已安装推荐的设置：
  - Node.js （最新 LTS 版本，或 16.11以上）
  - VS Code
    - ✓ Volar、ESLint、Prettier
- 命令行工具，新建项目（hello-nuxt ）
  - 方式一：npx nuxi init hello-nuxt
  - 方式二：pnpm dlx nuxi init hello-nuxt
  - 方式三：npm install –g nuxi && nuxi init hello-nuxt

```
liujundeMacBook-Pro:code liujun$ nuxi init 09-hello-nuxt
Nuxi 3.0.0-rc.13
✨ Nuxt project is created with v3 template. Next steps:
 › cd 09-hello-nuxt
 › Install dependencies with npm install or yarn install or
 › Start development server with npm run dev or yarn dev or
```

- 运行项目: cd hello-nuxt
  - yarn install
  - pnpm install --shamefully-hoist（创建一个扁平的 node_modules 目录结构，类似npm 和 yarn）
  - yarn dev

```
# .npmrc
shamefully-hoist=true
```

创建项目报错

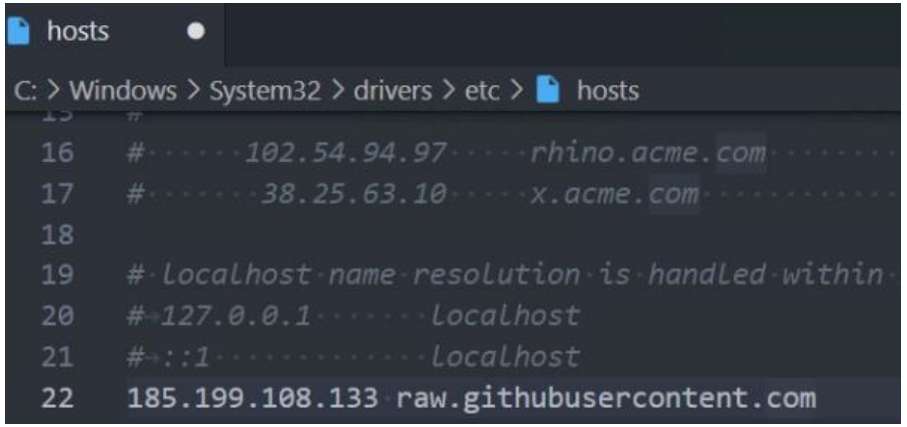
- 执行 npx nuxi init 01-hello-nuxt 报如下错误，主要是网络不通导致：

```
$ npx nuxi init 01-hello-nuxt
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.
Nuxi 3.0.0                                                                    20:18:22

ERROR  Failed to download template from registry: request to https://raw.githubusercontent.com/nuxt/starter/templates/templates/v3.json failed, reason: getaddrinfo ENOENT raw.githubusercontent.com

  at /C:/Users/DELL/AppData/Local/npm-cache/_npx/a95e0f536cf9a537/node_modules/nuxi/dist/chunks/init.mjs:15133:11
  at processTicksAndRejections (node:internal/process/task_queues:96:5)
  at async downloadTemplate (/C:/Users/DELL/AppData/Local/npm-cache/_npx/a95e0f536cf9a537/node_modules/nuxi/dist/chunks/init.mjs:15132:20)
  at async Object.invoke (/C:/Users/DELL/AppData/Local/npm-cache/_npx/a95e0f536cf9a537/node_modules/nuxi/dist/chunks/init.mjs:15200:15)
  at async _main (/C:/Users/DELL/AppData/Local/npm-cache/_npx/a95e0f536cf9a537/node_modules/nuxi/dist/cli.mjs:50:20)
```

- 解决方案：
  - 第一步：ping raw.githubusercontent.com 检查是否通
  - 第二步：如果访问不通，代表是网络不通
  - 第三步：配置 host，本地解析域名
    - ✓ Mac电脑 host 配置路径： /etc/hosts
    - ✓ Win 电脑 host 配置路由：c:/Windows/System32/drivers/etc/hosts

  - 第四步：在host文件中新增一行 ， 编写如下配置：
    - ✓ 185.199.108.133 raw.githubusercontent.com
  - 第五步：重新ping域名，如果通了就可以用了
  - 第六部：重新开一个终端创建项目即可

```
185.199.108.133
185.199.109.133
185.199.110.133
185.199.111.133
2606:50c0:8000::154
2606:50c0:8001::154
2606:50c0:8002::154
2606:50c0:8003::154
```



Nuxt3 目录结构

```
├── hello-nuxt  # Nuxt3项目名称
    ├── assets  # 资源目录
    ├── components  # 组件目录
    ├── composables  # 组合 API 目录
    ├── layout  # 布局 目录
    ├── pages  # 定义页面文件夹，路由会根据该页面目录结构和文件名自动生成
    |   └── index.vue # 项目的首页
    ├── plugins  # 插件目录
    ├── public # 静态资源目录，不参与打包
    ├── app.vue # 整个应用程序
    ├── app.config.ts # 应用程序的配置
    ├── nuxt.config.js # 可定制 Nuxt 框架的配置，比如：css、ssr、vite、app、modules等
    ├── package-lock.json # 项目依赖库版本的锁定
    ├── package.json # 项目的描述文件
    ├── README.md  # 项目简介
    └── tsconfig.json # TypeScript的配置文件
```

应用入口（App.vue）

- 默认情况下，Nuxt 会将此文件视为**入口点**，并为应用程序的每个路由呈现其内容，常用于：
  - 定义页面布局Layout 或 自定义布局，如：NuxtLayout
  - 定义路由的占位，如：NuxtPage
  - 编写全局样式
  - 全局监听路由 等等

```
<template>
  <NuxtLayout>
    <!-- 页面内容 -> vue-router -->
    <NuxtPage></NuxtPage>
  </NuxtLayout>
</template>
```

```
<script lang="ts" setup>
const router = useRouter();
router.beforeEach((to, form) => {
  console.log("beforeEach", to, form);
});
</script>
<style scoped></style>
```

Nuxt 配置（nuxt.config）

- nuxt.config.ts 配置文件位于项目的根目录，可对Nuxt进行自定义配置。比如，可以进行如下配置：
  - runtimeConfig：运行时配置，即定义环境变量
    - ✓ 可通过.env文件中的环境变量来覆盖，优先级（.env > runtimeConfig）
      - ➢ .env的变量会打入到process.env中，符合规则的会覆盖runtimeConfig的变量
      - ➢ .env一般用于某些终端启动应用时动态指定配置，同时支持dev和pro
  - appConfig： 应用配置，定义在构建时确定的公共变量，如：theme
    - ✓ 配置会和 app.config.ts 的配置合并（优先级 app.config.ts > appConfig）
  - app：app配置
    - ✓ head：给每个页面上设置head信息，也支持 useHead 配置和内置组件。
  - ssr：指定应用渲染模式
  - router：配置路由相关的信息，比如在客户端渲染可以配置hash路由
  - alias：路径的别名，默认已配好
  - modules：配置Nuxt扩展的模块，比如：@pinia/nuxt @nuxt/image
  - routeRules：定义路由规则，可更改路由的渲染模式或分配基于路由缓存策略（公测阶段）
- builder：可指定用 vite 还是 webpack来构建应用，默认是vite。如切换为 webpack 还需要安装额外的依赖。

.env

```
> NUXT_API_SECRET=api_secret_token
> NUXT_PUBLIC_API_BASE=https://nuxtjs.org
```

nuxt.config.ts

```
export default defineNuxtConfig({
  runtimeConfig: {
    apiSecret: '', // can be overridden by NUXT_API_SECRET environment variable
    public: {
      apiBase: '', // can be overridden by NUXT_PUBLIC_API_BASE environment variab
    }
  },
})
```

```
<script setup>
const config = useRuntimeConfig()
console.log('Runtime config:', config)
if (process.server) {
  console.log('API secret:', config.apiSecret)
}
</script>
```

应用配置（app.config）

- Nuxt 3 提供了一个 app.config.ts 应用配置文件，用来定义在构建时确定的公共变量，例如：
  - 网站的标题、主题色 以及任何不敏感的项目配置
- app.config.ts 配置文件中的选项不能使用env环境变量来覆盖，与 runtimeConfig 不同
- 不要将秘密或敏感信息放在 app.config.ts 文件中，该文件是客户端公开

```ts
export default defineAppConfig({
  theme: {
    primaryColor: '#ababab'
  }
})
```

```ts
const appConfig = useAppConfig()

console.log(appConfig.theme)
```

runtimeConifg vs app.config

- runtimeConfig 和 app.config都用于向应用程序公开变量。要确定是否应该使用其中一种，以下是一些指导原则：
  - runtimeConfig：定义环境变量，比如：运行时需要指定的私有或公共token。
  - app.config：定义公共变量，比如：在构建时确定的公共token、网站配置。

Feature	`runtimeConfig`	`app.config`
Client Side	Hydrated	Bundled
Environment Variables	✅ Yes	❌ No
Reactive	✅ Yes	✅ Yes
Types support	✅ Partial	✅ Yes
Configuration per Request	❌ No	✅ Yes
Hot Module Replacement	❌ No	✅ Yes
Non primitive JS types	❌ No	✅ Yes

Nuxt3 内置组件

- Nuxt3 框架也提供一些内置的组件，常用的如下：
  - SEO组件： Html、Body、Head、Title、Meta、Style、Link、NoScript、Base
  - NuxtWelcome：欢迎页面组件，该组件是 @nuxt/ui的一部分
  - NuxtLayout：是 Nuxt 自带的页面布局组件
  - NuxtPage：是 Nuxt 自带的页面占位组件
    - ✓ 需要显示位于目录中的顶级或嵌套页面 pages/
    - ✓ 是对 router-view 的封装
  - ClientOnly：该组件中的默认插槽的内容只在客户端渲染
    - ✓ 而fallback插槽的内容只在服务器端渲染
  - NuxtLink：是 Nuxt 自带的页面导航组件
    - ✓ 是 Vue Router<RouterLink>组件 和 HTML<a>标签的封装。
  - 等等

```vue
<template>
  <NuxtLayout :name="layout">
    <NuxtPage />
  </NuxtLayout>
</template>

<script setup>
// layouts/custom.vue
const layout = 'custom'
</script>
```

```vue
<ClientOnly>
  <!-- ... -->
  <template #fallback>
    <!-- this will be rendered on server side -->
    <p>Loading comments...</p>
  </template>
</ClientOnly>
```

全局样式

- 编写全局样式步骤
  - 1.在assets中编写全局样式，比如：globel.scss
  - 2.接着在nuxt.config中的css选项中配置
  - 3.接着执行npm i –D sass 即可

```
css: [
  // Load a Node.js module directl
  'bulma',
  // CSS file in the project
  '@/assets/css/main.css',
  // SCSS file in the project
  '@/assets/css/main.scss'
]
```

```
assets/_colors.scss  assets/_colors.sass

$primary: #49240F;
$secondary: #E4A79D;
```

- 定义全局变量步骤
  - 1.在assets中编写全局样式变量，比如：_colors.scss
  - 2.接着在nuxt.config中的vite选项中配置
  - 3.然后就可以在任意组件中或scss文件中直接使用全局变量

```
export default defineNuxtConfig({
  vite: {
    css: {
      preprocessorOptions: {
        scss: {
          additionalData: '@use "@/assets/_colors.scss" as *;'
        }
      }
    }
  }
})
```

资源的导入

- **public目录**
  - 用作静态资产的公共服务器，可在应用程序上直接通过 URL 直接访问
  - 比如：引用public/img/ 目录中的图像文件
    - ✓ 在静态 URL 中可用 /img/nuxt.png，如右图
    - ✓ 静态的URL也支持在背景中使用

```
<template>
  <img src="/img/nuxt.png" alt="Discover Nuxt 3" />
</template>
```

- **assets目录**
  - assets经常用于存放如样式表、字体或 SVG的资产
  - 可以使用 ~/assets/ 路径引用位于assets目录中的资产文件
  - ~/assets/ 路径也支持在背景中使用

```
<template>
  <img src="~/assets/img/nuxt.png" alt="Discover Nuxt 3" />
</template>
```

字体图标

- **字体图标使用步骤**
  - 1.将字体图标存放在assets目录下
  - 2.字体文件可以使用 ~/assets/ 路径引用。
  - 3.在nuxt.config配置文件中导入全局样式
  - 4.在页面中就可以使用字体图标了

```
export default defineNuxtConfig({
  // 导入的是全局样式
  css: [
    '~/assets/css/variable.css',
    '~/assets/css/globel.scss',
    '~/assets/custom-font/iconfont.css' // 导入字体图标
  ],
```

新建页面

- Nuxt项目中的页面是在 pages目录 下创建的
- 在pages目录创建的页面，Nuxt会根据该页面的目录结构和其文件名来自动生成对应的路由。
- 页面路由也称为文件系统路由器（file system router），路由是Nuxt的核心功能之一
- **新建页面步骤**
  - 1.创建页面文件，比如： pages/index.vue
  - 2.将<NuxtPage /> 内置组件添加到 app.vue
  - 3.页面如果使用scss那么需要安装：npm i sass -D
- **命令快速创建页面**
  - npx nuxi add page home # 创建home页面
  - npx nuxi add page detail/[id] # 创建detail页面
  - npx nuxi add page user-[role]/[id] # 创建user页面

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

```
> # Generates `pages/about.vue`
> npx nuxi add page about
```

```
> # Generates `pages/category/[id].vue`
> npx nuxi add page "category/[id]"
```

组件导航（NuxtLink）

- <NuxtLink>是Nuxt内置组件，是对 RouterLink 的封装，用来实现页面的导航。
  - 该组件底层是一个<a>标签，因此使用 a + href 属性也支持路由导航
  - 但是用a标签导航会有触发浏览器默认刷新事件，而 NuxtLink 不会，NuxtLink还扩展了其它的属性和功能
- 应用Hydration后（已激活，可交互），页面导航会通过前端路由来实现。这可以防止整页刷新
- 当然，手动输入URL后，点击刷新浏览器也可导航，这会导致整个页面刷新
- NuxtLink 组件属性：
  - to：支持路由路径、路由对象、URL
  - href：to的别名
  - activeClass：激活链接的类名
  - target：和a标签的target一样，指定何种方式显示新页面
  - 等等

```
<template>
  <header>
    <nav>
      <ul>
        <li><NuxtLink to="/about">About</NuxtLink></li>
        <li><NuxtLink to="/posts/1">Post 1</NuxtLink></li>
        <li><NuxtLink to="/posts/2">Post 2</NuxtLink></li>
      </ul>
    </nav>
  </header>
</template>
```

编程导航 （一）

- Nuxt3除了可以通过<NuxtLink>内置组件来实现导航，同时也支持编程导航：navigateTo 。
- 通过编程导航，在应用程序中就可以轻松实现动态导航了，但是编程导航不利于SEO。
- navigateTo 函数在服务器端和客户端都可用，也可以在插件、中间件中使用，也可以直接调用以执行页面导航，例如：
  - 当用户触发该goToProfile()方法时，我们通过navigateTo函数来实现动态导航。
  - 建议： goToProfile方法总是返回 navigateTo 函数（该函数不需要导入）或 返回异步函数
- navigateTo(to , options) 函数:
  - to: 可以是纯字符串 或 外部URL 或 路由对象，如右图所示：
  - options: 导航配置，可选
    - ✓ replace：默认为false，为true时会替换当前路由页面
    - ✓ external：默认为false，不允许导航到外部连接，true则允许
    - ✓ 等等

```
<script setup>
// passing 'to' as a string
await navigateTo('/search')

// ... or as a route object
await navigateTo({ path: '/search' })

// ... or as a route object with query parameters
await navigateTo({
  path: '/search',
  query: {
    page: 1,
    sort: 'asc'
  }
})
</script>
```

编程导航（二）

- Nuxt3z中的编程导航除了可以通过 navigateTo 来实现导航，同时也支持 useRouter（或 Options API 的 this.$router）
- useRouter常用的API
  - back：页面返回，和 一样 router.go(-1)
  - forward：页面前进，同 router.go(1)
  - go：页面返回或前进，如 router.go(-1) or router.go(1)
  - push：以编程方式导航到新页面。建议改用 navigateTo 。支持性更好
  - replace：以编程方式导航到新页面，但会替换当前路由。建议改用 navigateTo 。支持性更好
  - beforeEach：路由守卫钩子，每次导航前执行（用于全局监听）
  - afterEach：路由守卫钩子，每次导航后执行（用于全局监听）
  - .....

```
const router = useRouter();
router.back();
router.forward();
router.go();
router.push({ path: "/home" });
router.replace({ hash: "#bio" });
```

动态路由

- Nuxt3 和 Vue一样，也是支持动态路由的，只不过在Nuxt3中，动态路由也是根据目录结构和文件的名称自动生成。
- 动态路由语法：
  - 页面组件目录 或 页面组件文件都 支持 [ ] 方括号语法
  - 方括号里编写动态路由的参数。
- 例如，动态路由 支持如下写法：
  - pages/detail/[id].vue -> /detail/:id
  - pages/detail/user-[id].vue -> /detail/user-:id
  - pages/detail/[role]/[id].vue -> /detail/:role/:id
  - pages/detail-[role]/[id].vue -> /detail-:role/:id
- 注意事项：
  - 动态路由 和 index.vue 不能同时存在， Next.js则可以

```
pages/
--| about.vue
--| posts/
----| [id].vue
```

```
{
  "routes": [
    {
      "path": "/about",
      "component": "pages/about.vue"
    },
    {
      "path": "/posts/:id",
      "component": "pages/posts/[id].vue"
    }
  ]
}
```

路由参数

- 动态路由参数
  - 1.通过 [] 方括号语法定义动态路由，比如：/detail/[id].vue
  - 2.页面跳转时，在URL路径中传递动态路由参数，比如：/detail/10010
  - 3.目标页面通过 route.params 获取动态路由参数
- 查询字符串参数
  - 1.页面跳转时，通过查询字符串方式传递参数，比如：/detail/10010?name=liujun
  - 2.目标页面通过 route.query 获取查询字符串参数

```
<script setup>
const route = useRoute()

// When accessing /posts/1, route.params.id will be 1
console.log(route.params.id)
</script>
```

404 Page

- 捕获所有不配路由（即 404 not found 页面）
  - 通过在方括号内添加三个点 ，如：[...slug].vue 语法，其中slug可以是其它字符串。
  - 除了支持在 pages根目录下创建，也支持在其子目录中创建。
  - Nuxt3正式版不支持404.vue页面了，以前的候选版是支持的404.vue，但是Next.js是支持。

Catch-all Route

If you need a catch-all route, you create it by using a file named like `[...slug].vue`. This will match *all* routes under that path.

pages/[...slug].vue

```
<template>
  <p>{{ $route.params.slug }}</p>
</template>
```

Navigating to `/hello/world` would render:

```
<p>["hello", "world"]</p>
```

路由匹配规则

- 路由匹配需注意的事项
  - 预定义路由优先于动态路由，动态路由优先于捕获所有路由。请看以下示例：
    - ✓ 1.预定义路由：pages/detail/create.vue
      - ➢ 将匹配 /detail/create
    - ✓ 2.动态路由：pages/detail/[id].vue
      - ➢ 将匹配/detail/1， /detail/abc 等。
      - ➢ 但不匹配 /detail/create 、/detail/1/1、/detail/ 等
    - ✓ 3.捕获所有路由：pages/detail/[...slug].vue
      - ➢ 将匹配 /detail/1/2， /detail/a/b/c 等。
      - ➢ 但不匹配 /detail 等

嵌套路由

- Nuxt 和 Vue一样，也是支持嵌套路由的，只不过在Nuxt中，嵌套路由也是根据目录结构和文件的名称自动生成。
- 编写嵌套路由步骤：
  - 1.创建一个一级路由，如：parent.vue
  - 2.创建一个与一级路由同名同级的文件夹，如： parent
  - 3.在parent文件夹下，创建一个嵌套的二级路由
    - ✓ 如：parent/child.vue， 则为一个二级路由页面
    - ✓ 如： parent/index.vue 则为二级路由默认的页面
  - 4.需要在parent.vue中添加 NuxtPage 路由占位

```
> -| pages/
> ---| parent/
> -------| child.vue
> ---| parent.vue
```

```
[
  {
    path: '/parent',
    component: '~/pages/parent.vue',
    name: 'parent',
    children: [
      {
        path: 'child',
        component: '~/pages/parent/child.vue'
        name: 'parent-child'
      }
    ]
  }
]
```