

NUXT Docs - Português



Table of contents

• Get started - Installation	5
• Get started - Routing	9
• Get started - Directory structure	11
• Get started - Commands	13
• Get started - Conclusion	16
• Get started - Upgrading	17
• Concepts - Views	18
• Concepts - Context helpers	22
• Concepts - Server side rendering	27
• Concepts - Static site generation	29
• Concepts - Nuxt lifecycle	31
• Features - Rendering modes	36
• Features - Transitions	37
• Features - Live preview	41
• Features - Deployment targets	43
• Features - File system routing	45
• Features - Data fetching	53
• Features - Meta tags seo	60
• Features - Configuration	63
• Features - Loading	72
• Features - Nuxt components	77
• Features - Component discovery	84
• Directory structure - Nuxt	87
• Directory structure - Plugins	89
• Directory structure - Static	96
• Directory structure - Store	98
• Directory structure - Nuxt config	103
• Directory structure - Assets	110
• Directory structure - Components	115
• Directory structure - Content	118
• Directory structure - Dist	126
• Directory structure - Layouts	128
• Directory structure - Middleware	130
• Directory structure - Modules	132

• Directory structure - Pages	141
• Configuration glossary - Configuration alias	147
• Configuration glossary - Configuration extend plugins	149
• Configuration glossary - Configuration generate	150
• Configuration glossary - Configuration global name	157
• Configuration glossary - Configuration head	158
• Configuration glossary - Configuration hooks	159
• Configuration glossary - Configuration ignore	162
• Configuration glossary - Configuration loading	164
• Configuration glossary - Configuration loading indicator	166
• Configuration glossary - Configuration mode	168
• Configuration glossary - Configuration modern	169
• Configuration glossary - Configuration build	171
• Configuration glossary - Configuration modules	189
• Configuration glossary - Configuration modulesdir	191
• Configuration glossary - Configuration plugins	192
• Configuration glossary - Configuration render	194
• Configuration glossary - Configuration rootdir	201
• Configuration glossary - Configuration router	202
• Configuration glossary - Configuration runtime config	210
• Configuration glossary - Configuration server	211
• Configuration glossary - Configuration servermiddleware	213
• Configuration glossary - Configuration srkdir	216
• Configuration glossary - Configuration builddir	218
• Configuration glossary - Configuration ssr	219
• Configuration glossary - Configuration target	220
• Configuration glossary - Configuration telemetry	221
• Configuration glossary - Configuration transition	223
• Configuration glossary - Configuration vue config	225
• Configuration glossary - Configuration watch	226
• Configuration glossary - Configuration watchers	227
• Configuration glossary - Configuration cli	228
• Configuration glossary - Configuration css	230
• Configuration glossary - Configuration components	231
• Configuration glossary - Configuration dev	236
• Configuration glossary - Configuration dir	237

• Configuration glossary - Configuration env	238
• Internals glossary - Context	240
• Internals glossary - Nuxt render	244
• Internals glossary - Nuxt render route	245
• Internals glossary - Nuxt render and get window	246
• Internals glossary - \$nuxt	247
• Internals glossary - Internals	249
• Internals glossary - Internals nuxt	251
• Internals glossary - Internals renderer	252
• Internals glossary - Internals module container	253
• Internals glossary - Internals builder	256
• Internals glossary - Internals generator	257
• Internals glossary - Nuxt	258
• Components glossary - Fetch	259
• Components glossary - Watchquery	261
• Components glossary - Head	262
• Components glossary - Key	263
• Components glossary - Layout	264
• Components glossary - Loading	265
• Components glossary - Middleware	266
• Components glossary - Transition	267
• Components glossary - Scrolltotop	270
• Components glossary - Validate	271

Instalação

Aqui, você vai encontrar informações sobre como configurar e iniciar um projeto Nuxt em 4 passos.

Online playground

Você pode divertir-se com o Nuxt de forma online, diretamente pelo CodeSandbox ou StackBlitz: :app-button[Divertir-se no CodeSandbox] :app-button[Divertir-se no StackBlitz]

Pré-requisitos

- [node](#) - *Nós recomendamos que você tenha a última versão LTS instalada.*
- Um editor de texto, nós recomendamos [VS Code](#) com a extensão [Vetur](#) ou [WebStorm](#)
- Um terminal, nós recomendamos utilizar o [terminal integrado](#) do VS Code ou [terminal WebStorm](#).

Usando create-nuxt-app

Para iniciar rapidamente, você pode utilizar [create-nuxt-app](#).

Tenha certeza que você possui instalado o yarn, npx (incluído por padrão no npm v5.2+) ou npm (v6.1+).

```
yarn create nuxt-app <project-name>
```

```
npx create-nuxt-app <project-name>
```

```
npm init nuxt-app <project-name>
```

O terminal irá fazer algumas perguntas (nome, opções do Nuxt, framework de UI, TypeScript, linter, framework de testes, etc). Para encontrar mais detalhes sobre todas essas opções veja a [documentação create-nuxt-app](#).

Uma vez que todas as questões foram respondidas, será instalada todas as dependências. O próximo passo é navegar para a pasta do projeto e inicia-lo:

```
cd <project-name>
yarn dev
```

```
cd <project-name>
npm run dev
```

A aplicação agora estará rodando em <http://localhost:3000>. Parabéns!

Info

Outra forma de iniciar o Nuxt é utilizando [CodeSandbox](#) o qual é uma ótima maneira de iniciar rapidamente no Nuxt e/ou compartilhar o seu código com outras pessoas.

Instalação manual

Criar um projeto Nuxt do início vai necessitar apenas de um arquivo e uma pasta.

Nós vamos usar o terminal para criar os diretórios e arquivos, sinta-se livre para criar usando um editor de sua escolha.

Configurando o seu projeto

Crie uma pasta vazia com o nome do seu projeto e navegue para dentro dele :

```
mkdir <project-name>
cd <project-name>
```

Substitua `<project-name>` com o nome do seu projeto.

Crie o arquivo `package.json` :

```
touch package.json
```

Preencha o conteúdo do seu `package.json` com :

```
{
  "name": "my-app",
  "scripts": {
    "dev": "nuxt",
    "build": "nuxt build",
    "generate": "nuxt generate",
    "start": "nuxt start"
  }
}
```

`scripts` define os comandos do Nuxt que serão executados com o comando `npm run <command>` ou `yarn <command>`.

O que é o arquivo `package.json` ?

O arquivo `package.json` é como o documento de identificação do seu projeto. Ele irá conter todas as dependências do projeto e muito mais. Se você não sabe o que é o arquivo `package.json`, nós recomendamos imensamente a leitura da documentação do [npm](#).

Instalando Nuxt

Uma vez que o `package.json` foi criado, adicione `nuxt` no seu projeto via `npm` or `yarn` utilizando um dos comandos abaixo:

```
yarn add nuxt
```

```
npm install nuxt
```

Esse comando vai adicionar `nuxt` como uma dependência para o seu projeto e vai adiciona-lo no seu `package.json`. A pasta `node_modules` também será criada, é aonde todos os seus pacotes instalados e dependencias serão salvas.

Info

Um arquivo `yarn.lock` ou `package-lock.json` também será criado, o qual garante consistênci a e a compatibilidade das dependências instaladas no seu projeto.

Criando a sua primeira página

Nuxt transforma todos arquivos `*.vue` dentro da pasta `pages` em rotas para a aplicação.

Crie a pasta `pages` dentro do seu projeto :

```
mkdir pages
```

Então, crie um arquivo `index.vue` dentro da pasta `pages` :

```
touch pages/index.vue
```

É importante que essa página seja nomeada `index.vue` dessa forma ela será a página principal que o Nuxt irá mostrar quando a aplicação abrir.

Abra o arquivo `index.vue` no seu editor e adicione o seguinte conteúdo :

```
<template>
  <h1>Hello world!</h1>
</template>
```

Iniciando o projeto

Inicie o seu projeto executando um dos seguintes comandos abaixo :

```
yarn dev
```

```
npm run dev
```

Info

Nós usamos o comando `dev` quando iniciamos a nossa aplicação no modo de desenvolvimento.

A aplicação está agora rodando em <http://localhost:3000>.

Abra no seu navegador clicando no link do seu terminal e você deverá ver o texto "Hello world!" que nós copiamos no passo anterior.

Info

Quando iniciado o Nuxt no modo de desenvolvimento, ele estará escutando por mudanças nos arquivos dentro dos demais diretórios, então não há necessidade de reiniciar a aplicação quando por exemplo adicionar páginas novas.

Warning

Quando você rodar o comando dev será criado uma pasta chamada `.nuxt`. Essa pasta deve ser ignorada pelo controle de versionamento. Você ignorar arquivos criando um arquivo `.gitignore` na raiz do projeto e adicionando `.nuxt` ao arquivo.

Passo extra

Crie uma pagina chamada `fun.vue` dentro da pasta `pages`.

Adicione um `<template></template>` e inclua um cabeçalho com uma frase engraçada dentro.

Então, vá até o seu navegador e veja a sua nova página em localhost:3000/fun.

Info

Criar uma pasta `more-fun` e colocar um arquivo `index.vue` dentro, irá dar o mesmo resultado que criar um arquivo `more-fun.vue`.

[Go to TOC](#)

Roteamento

A maioria dos sítios da web têm mais que uma página. Por exemplo uma página inicial, uma página sobre, página de contacto etc. No sentido de exibir essas páginas nós precisamos de um Roteador.

Rotas automáticas

A maioria dos websites irão ter mais que uma página (por exemplo página inicial, página sobre, formas de contato etc.). Para mostrar essas páginas nós precisamos de um Router. É aonde o `vue-router` entra. Quando se trabalha com aplicações Vue, você tem que iniciar o arquivo de configuração (exemplo `router.js`) e adicionar todas as rotas manualmente nele. O Nuxt gera automaticamente as configurações do `vue-router` para você, com base nos ficheiros de Vue dentro da pasta `pages`. Isso significa que você nunca mais terá que escrever as configurações do roteador! O Nuxt também oferece para você a separação automática de código para todas suas rotas.

Em outras palavras, tudo o que você tem que fazer para ter uma rota na sua aplicação é criar ficheiros `.vue` dentro da pasta `pages`.

Next

Saiba mais sobre o [Roteamento](#)

Navegação

Para navegar entre as páginas da sua aplicação, você deve usar o componente [NuxtLink](#). Esse componente é incorporado no Nuxt, portanto você não tem que importar ele assim como você faz com os outros componentes. É similar ao marcador `<a>` do HTML, exceto que ao invés de usar um `href="/about"` nós usamos `to="/about"`. Se você já usou `vue-router` antes, você pode pensar no componente `<NuxtLink>` como uma substituto do componente `<RouterLink>`.

Uma simples ligação para a página `index.vue` dentro da sua pasta `pages`:

```
<template>
  <NuxtLink to="/">Página inicial</NuxtLink>
</template>
```

Para todas ligações para as páginas dentro do seu sítio, use `<NuxtLink>`. Se você tiver ligações para outros sítios na web você deve usar o marcador `<a>`. Para ter uma ideia consulte o exemplo abaixo:

```
<template>
  <main>
    <h1>Página Inicial</h1>
    <NuxtLink to="/about">
      Sobre (ligação interna que pertence a aplicação do Nuxt)
    </NuxtLink>
```

```
<a href="https://nuxtjs.org">Ligaçāo externa para outra pāgina</a>
</main>
</template>
```

Next

Saiba mais sobre o [Componente NuxtLink](#).

[Go to TOC](#)

Estrutura de Diretórios

A estrutura padrão de uma aplicação Nuxt oferece um excelente ponto de partida tanto para aplicações pequenas, quanto para aplicações grandes. Você está livre para organizar a sua aplicação da forma que desejar e criar outros diretórios conforme precisar.

Vamos criar os diretórios e ficheiros que ainda não existem no seu projeto.

```
mkdir components assets static  
touch nuxt.config.js
```

Esses são os diretórios e os ficheiros principais que nós usamos quando estamos construindo uma aplicação Nuxt. Você vai encontrar uma explicação para cada um deles abaixo.

Info

Criar diretórios com esses nomes ativa as funcionalidades no seu projeto Nuxt.

Diretórios

O diretório pages

O diretório `pages` contém as apresentações e rotas da sua aplicação. Conforme você aprendeu no último capítulo, Nuxt lê todos os arquivos `.vue` dentro da pasta e os utiliza para criar as rotas da aplicação.

Next

Aprender mais sobre o [diretório pages](#)

O diretório components

O diretório `components` é aonde você vai colocar todos os seus componentes Vue.js os quais serão importados dentro das suas páginas.

Com o Nuxt você pode criar os seus componentes e importar eles automaticamente dentro de seus ficheiros `.vue`, querendo dizer que não é mais necessário importar eles manualmente na secção de `script`. Nuxt examinará e importará eles automaticamente por você uma vez que "components" está definido como "true".

Next

Aprenda mais sobre o [diretório components](#)

O diretório assets

O diretório `assets` contém seus ficheiros não compilados como folhas de estilo, imagens, ou fontes.

Next

Aprenda mais sobre o [diretório assets](#)

O diretório static

O diretório `static` é mapeado diretamente para a raiz do servidor e contém arquivos que precisam manter os seus nomes (por exemplo `robots.txt`) ou semelhantes que não irão mudar (exemplo o favicon)

Next

Aprenda mais sobre o [diretório static](#)

O ficheiro nuxt.config.js

O ficheiro `nuxt.config.js` é o único ponto de configuração para o Nuxt. Se você quiser adicionar módulos ou sobrescrever as configurações padrão, esse é o lugar para aplicar as mudanças.

Next

Aprenda mais sobre o [ficheiro nuxt.config.js](#)

O ficheiro package.json

O ficheiro `package.json` contém todas as dependências e scripts para a sua aplicação.

Mais sobre a estrutura de projeto

Existem outros diretórios e ficheiros úteis, como `content`, `layouts`, `middleware`, `modules`, `plugins` e `store`. Como eles não são necessários para pequenas aplicações, eles não são explicados aqui.

Next

Para aprender mais sobre todas as pastas em detalhes, sinta-se livre para ler [O guia de estrutura de diretórios](#).

[Go to TOC](#)

Comandos e desdobramento

O Nuxt oferece um conjunto de comandos úteis, tanto para fins de desenvolvimento como para produção.

Usando no package.json

Você deve ter esses comandos no seu `package.json`:

```
"scripts": {
  "dev": "nuxt",
  "build": "nuxt build",
  "start": "nuxt start",
  "generate": "nuxt generate"
}
```

Você pode executar os seus comandos via `yarn <command>` ou `npm run <command>` (exemplo: `yarn dev` / `npm run dev`).

Ambiente de desenvolvimento

Para iniciar o Nuxt em modo de desenvolvimento com a [substituição instantânea de módulo](#) em `http://localhost:3000`:

`yarn dev`

`npm run dev`

Lista de comandos

Você pode executar diferentes comandos dependendo do [alvo](#):

target: `server` (valor padrão)

- **nuxt dev** - Inicia o servidor de desenvolvimento.
- **nuxt build** - Constrói e otimiza a sua aplicação com webpack para produção.
- **nuxt start** - Inicia o servidor de produção (depois de rodar `nuxt build`). Use ele para hospedagem de Node.js como Heroku, Digital Ocean, etc.

target: `static`

- **nuxt dev** - Inicia o servidor de desenvolvimento.
- **nuxt generate** - Constrói a aplicação (se necessário), gera todas as rotas como um arquivo HTML e exporta estaticamente para o diretório `dist/` (usado para hospedagem estática).
- **nuxt start** - Serve o diretório `dist/` como sua hospedagem estática faria (Netlify, Vercel, Surge, etc), ótimo para testar antes do desdobramento.

Inspeção da configuração do Webpack

Você pode inspecionar a configuração do webpack usada pelo Nuxt para construir o projeto (similar ao [vue inspect](#)).

- `nuxt webpack [query...]`

Argumentos:

- `--name` : Nome do pacote a inspecionar. (client, server, modern)
- `--dev` : Inspeciona a configuração do webpack para modo de desenvolvimento
- `--depth` : Inspeção profunda. O valor padrão é 2 para evitar saída verbose.
- `--no-colors` : Desativa as cores ANSI (desativado por padrão quando o TTY não está disponível ou quando estiver canalizando para um ficheiro)

Exemplos:

- `nuxt webpack`
- `nuxt webpack devtool`
- `nuxt webpack resolve alias`
- `nuxt webpack module rules`
- `nuxt webpack module rules test=.jsx`
- `nuxt webpack module rules test=.pug oneOf use.0=raw`
- `nuxt webpack plugins constructor.name=WebpackBar options reporter`
- `nuxt webpack module rules loader=vue-`
- `nuxt webpack module rules "loader=.*-loader"`

Desdobramento de produção

Nuxt deixa você escolher entre os desdobramentos de servidor ou estático.

Desdobramento de servidor

Para desdobrar uma aplicação renderizada no lado do servidor (SSR) nós usamos `target: 'server'`, onde `server` é o valor padrão.

```
yarn build
```

```
npm run build
```

Nuxt criará um diretório `.nuxt` com tudo pronto para ser desdoblado na sua hospedagem de.

Info

Nós recomendamos adicionar `.nuxt` ao `.npmignore` ou `.gitignore`.

Uma vez que sua aplicação está construída você pode usar o comando `start` para ver a versão de produção da sua aplicação.

`yarn start``npm run start`

Desdobramento estático (Pré-renderizado)

O Nuxt dá para você habilidade de hospedar a sua aplicação web em qualquer hospedagem estática.

Para desdobrar um sitio estático gerado, certifique-se de ter o `target: 'static'` dentro do seu ficheiro `nuxt.config.js` (para versões do Nuxt maiores ou igual a 2.13):

```
export default {
  target: 'static'
}
```

`yarn generate``npm run generate`

O Nuxt criará um diretório `.nuxt` com tudo pronto para ser hospedado em um serviço de hospedagem de sítios estáticos.

A partir da versão 2.13 do Nuxt, existe um rastreador instalado que agora rastreará seus marcadores de ligação e gerará suas rotas baseadas naquelas ligações quando estiver usando o comando `nuxt generate`.

Warning

Aviso: Rotas dinâmicas são ignoradas pelo comando `generate` quando estiver usando versões do Nuxt menores ou igual a 2.12: [API de Configuração de generate](#)

Info

Quando estiver gerando a sua aplicação web com o `nuxt generate`, o contexto dado para `asyncData` e `fetch` não terá `req` e `res`.

Falha sobre erro

Para retornar um código de estado diferente de zero quando um erro de página for encontrado e permitir o CI/CD falhar o desdobramento ou a construção, você pode usar o argumento `--fail-on-error`.

`yarn generate --fail-on-error``npm run generate --fail-on-error`

Qual é o próximo?

Next

Leia o nosso [Guia de Desdobramentos](#) para encontrar exemplos de desdobramentos em hospedeiros populares.

[Go to TOC](#)

Conclusão

Parabéns você acabou de criar a sua primeira aplicação Nuxt e você agora deve se considerar um Nuxter. Mais existe ainda muita coisa para aprender e muito mais que você pode fazer com o Nuxt. Aqui temos algumas recomendações.

Next

Dê uma olhada no [Livro de conceitos](#)

Next

Trabalhando com [asyncData](#)

Next

Escolhendo entre os diferentes [modos de renderização](#)

`alert{type="star"}` Você têm gostado do Nuxt? não se esqueça de dar uma [estrela no nosso projeto](#) on GitHub

[Go to TOC](#)

Atualizando

Atualizar o Nuxt é rápido, mas tem mais coisas envolvidas que o seu package.json

Se você estiver atualizando para Nuxt v2.14 e você quer usar hospedagem estática então você precisa adicionar `target:static` no seu arquivo `nuxt.config.js` para conseguir gerar o comando e funcionar apropriadamente.

```
export default {  
    target: 'static'  
}
```

Iniciando

1. Olhe a [notas de lançamento](#) para a versão que você deseja atualizar, e valide se existem instruções adicionais para aquela release em particular.
2. Atualize a versão especificada para o pacote `nuxt` no seu arquivo `package.json`.

Após esta etapa, as instruções variam dependendo se você está usando Yarn ou npm. [Yarn] (<https://yarnpkg.com/en/docs/usage>) é o gerenciador de pacotes preferido para trabalhar com Nuxt, pois é a ferramenta de desenvolvimento que os testes foram escritos .

Yarn

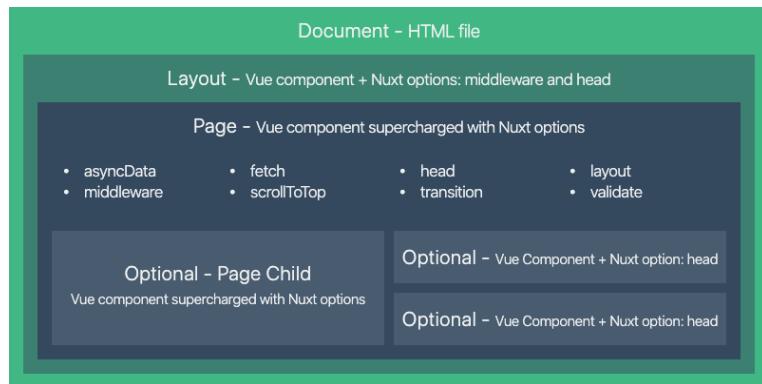
3. remova o arquivo `yarn.lock`
4. remova a pasta `node_modules`
5. Execute o comando `yarn`
6. Após a instalação ser concluída e você ter executado seus testes, considere atualizar também outras dependências. O comando `yarn outdated` pode ser usado para isso.

npm

3. remova o arquivo `package-lock.json`
4. remova o diretório `node_modules`
5. Execute o comando `npm install`
6. Após a instalação ser concluída e você ter executado seus testes, considere atualizar também outras dependências. O comando `npm outdated` pode ser usado.

Views

The Views section describes all you need to know to configure data and views for a specific route in your Nuxt Application. Views consist of an app template, a layout, and the actual page.



Composition of a View in Nuxt

Composition of a View in Nuxt

Pages

Every Page component is a Vue component but Nuxt adds special attributes and functions to make the development of your application as easy as possible.

```

<template>
  <h1 class="red">Hello World</h1>
</template>

<script>
  export default {
    head() {
      // Set Meta Tags for this Page
    }
    // ...
  }
</script>

<style>
  .red {
    color: red;
  }
</style>
  
```

Properties of a page component

There are many properties of the page component such as the `head` property in the example above.

Next

See the [Directory Structure book](#) to learn more about all the properties can use on your page

Layouts

Layouts are a great help when you want to change the look and feel of your Nuxt app. For example you want to include a sidebar or have distinct layouts for mobile and desktop.

Default Layout

You can define a default layout by adding a `default.vue` file inside the layouts directory. This will be used for all pages that don't have a layout specified. The only thing you need to include in the layout is the `<Nuxt />` component which renders the page component.

```
<template>
  <Nuxt />
</template>
```

Next

Learn more about the [Nuxt component](#) in the components chapter

Custom Layout

You can create custom layouts by adding a `.vue` file to the layouts directory. In order to use the custom layout you need to set the `layout` property in the page component where you want to use that layout. The value will be the name of the custom layout that you have created.

To create a blog layout add a `blog.vue` file to your layouts directory `layouts/blog.vue`:

```
<template>
  <div>
    <div>My blog navigation bar here</div>
    <Nuxt />
  </div>
</template>
```

Warning

Make sure to add the `<Nuxt/>` component when creating a layout to actually include the page component.

We then use the layout property with the value of 'blog' in the page where we want that layout to be used.

```
<template>
  <!-- Your template -->
</template>
<script>
  export default {
    layout: 'blog'
    // page component definitions
  }
</script>
```

Info

If you don't add a layout property to your page, e.g. `layout: 'blog'`, then the `default.vue` layout will be used.

Error Page

The error page is a *page component* which is always displayed when an error occurs (that does not happen while server-side rendering).

Warning

Although this file is placed in the `layouts` folder, it should be treated as a page.

As mentioned above, this layout is special, since you should not include the `<Nuxt/>` component inside its template. You must see this layout as a component displayed when an error occurs (`404`, `500`, etc.). Similar to other page components, you can set a custom layout for the error page as well in the usual way.

You can customize the error page by adding a `layouts/error.vue` file:

```
<template>
<div>
  <h1 v-if="error.statusCode === 404">Page not found</h1>
  <h1 v-else>An error occurred</h1>
  <NuxtLink to="/">Home page</NuxtLink>
</div>
</template>

<script>
export default {
  props: ['error'],
  layout: 'error' // you can set a custom layout for the error page
}
</script>
```

Document: App.html

The app template is used to create the actual HTML frame of your document for your Nuxt application which injects the content as well as variables for the head and body. This file is created automatically for you and in general rarely needs to be modified. You can customize the HTML app template used by Nuxt to include scripts or conditional CSS classes by creating an `app.html` file in the source directory of your project which by default is the root directory.

The default template used by Nuxt is:

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

```
  {{ APP }}  
  </body>  
</html>
```

One use case of using a custom app template is to add conditional CSS classes for IE:

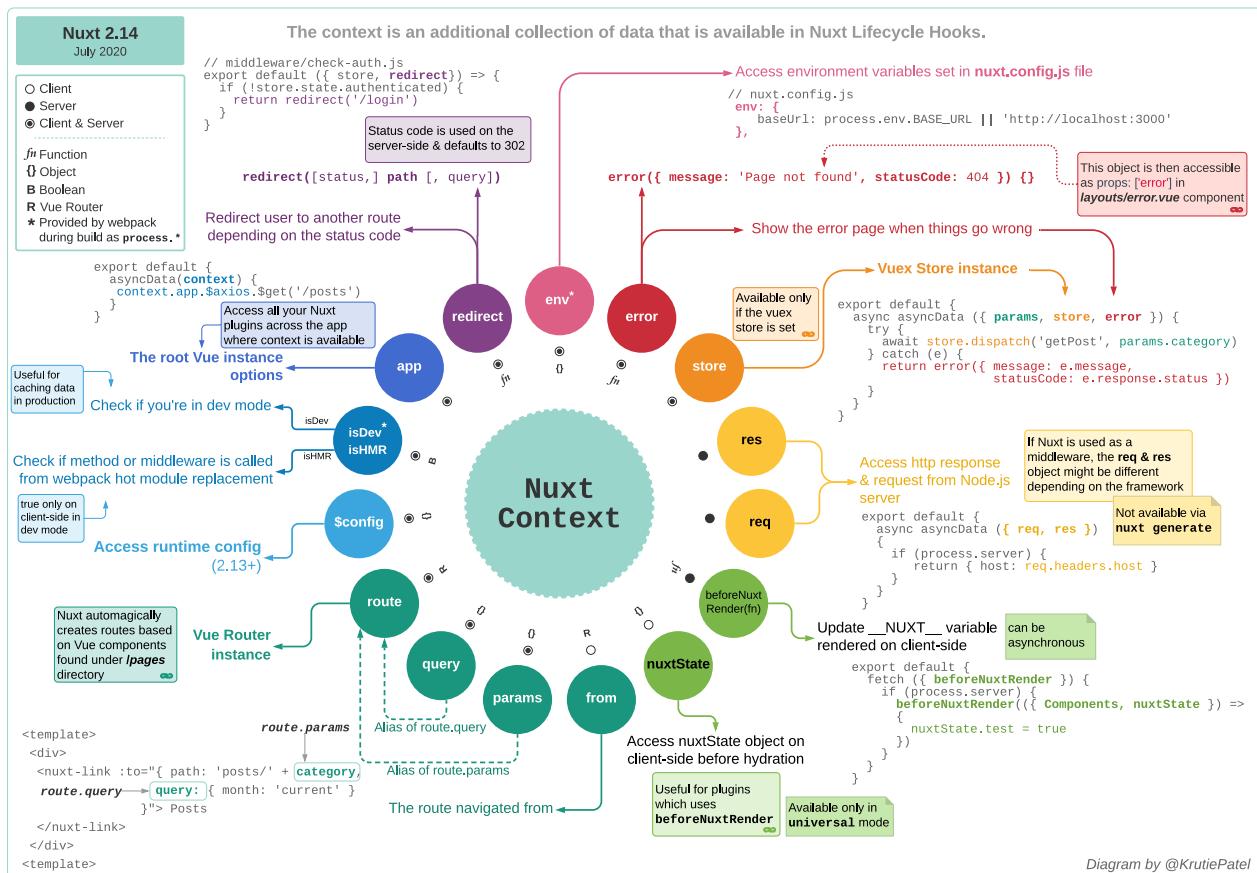
```
<!DOCTYPE html>  
<!--[if IE 9]><html class="lt-ie9 ie9" {{ HTML_ATTRS }}><![endif]-->  
<!--[if (gt IE 9) | !(IE)]><!--><html {{ HTML_ATTRS }}><!--<![endif]-->  
  <head {{ HEAD_ATTRS }}>  
    {{ HEAD }}  
  </head>  
  <body {{ BODY_ATTRS }}>  
    {{ APP }}  
  </body>  
</html>
```

Info

While you can add JavaScript and CSS files in the `app.html`, it is recommended to use the `nuxt.config.js` for these tasks instead!

Context and helpers

The context provides *additional* and often optional information about the current request to the application.



The `context` object is available in specific Nuxt functions like `asyncData`, `plugins`, `middleware` and `nuxtServerInit`. It provides *additional* and often optional information about the current request to the application.

First and foremost, the context is used to provide access to other parts of the Nuxt application, e.g. the Vuex store or the underlying `connect` instance. Thus, we have the `req` and `res` objects in the context available on the server side and `store` always available. But with time, the context was extended with many other helpful variables and shortcuts. Now we have access to HMR (Hot Module Reload / Replacement) functionalities in `development` mode, the current `route`, page `params` and `query`, as well as the option to access environment variables through the context. Furthermore, module functions and helpers can be exposed through the context to be available on both - the client and the server side.

All context keys that are present by default

```

function (context) { // Could be asyncData, nuxtServerInit, ...
  // Always available
  const {
    app,
    store,
    route,
    params,
    query,
    env,
    isDev,
    isHMR,
    redirect,
    error,
    $config
  } = context

  // Only available on the Server-side
  if (process.server) {
    const { req, res, beforeNuxtRender } = context
  }

  // Only available on the Client-side
  if (process.client) {
    const { from, nuxtState } = context
  }
}

```

Warning

The `context` we refer to here is not to be confused with the `context` object available in [Vuex Actions](#) or the one available in the `build.extend` function in your `nuxt.config.js`. These are not related to each other!

Learn more about the different context keys in our [Internals Glossary](#)

Using page parameters for your API query

The context directly exposes possible dynamic parameters of the route via `context.params`. In the following example, we call an API via the `nuxt/http` module using a dynamic page parameter as part of the URL. Modules, like the `nuxt/http` module, can expose own functions which are then available through the `context.app` object.

Also, we wrap the API call in a `try/catch` statement to handle potential errors. With the `context.error` function, we can directly show Nuxt's error page and pass in the occurred error.

```

export default {
  async asyncData(context) {
    const id = context.params.id
    try {
      // Using the nuxtjs/http module here exposed via context.app
      const post = await context.app.$http.$get(
        `https://api.nuxtjs.dev/posts/${id}`
      )
      return { post }
    } catch (e) {
      context.error(e) // Show the nuxt error page with the thrown error
    }
  }
}

```

With [ES6](#) you can use this syntax to destructure your context object. You can pass in the objects you want to have access to and then you can use them in the code without using the word context.

```
export default {
  async asyncData({ params, $http, error }) {
    const id = params.id

    try {
      // Using the nuxtjs/http module here exposed via context.app
      const post = await $http.$get(`https://api.nuxtjs.dev/posts/${id}`)
      return { post }
    } catch (e) {
      error(e) // Show the nuxt error page with the thrown error
    }
  }
}
```

Want to use query parameters instead? You then use [context.query.id](#).

Redirecting users & accessing the store

Accessing the [Vuex store](#) (when you have it set up through the `store` directory) is also possible through the context. It provides a `store` object which can be treated as `this.$store` in Vue components. In addition, we use the `redirect` method, an exposed helper through the context, to redirect the user in case the `authenticated` state is `falsy`.

```
export default {
  middleware({ store, redirect }) {
    // retrieving keys via object destructuring
    const isAuthenticated = store.state.authenticated
    if (!isAuthenticated) {
      return redirect('/login')
    }
  }
}
```

Next

Check out the Internals Glossary book for more examples of the [redirect](#) method

Helpers

Besides the shortcuts in the context, there are also other tiny helpers present in your Nuxt application.

\$nuxt : The Nuxt helper

`$nuxt` is a helper designed to improve the user experience and to be an escape hatch in some situations. It is accessible via `this.$nuxt` in Vue components and via `window.$nuxt` otherwise on the client side.

Connection checker

The `$nuxt` helper provides a quick way to find out whether the internet connection of a user is present or not: It exposes the boolean values `isOffline` and `isOnline`. We can use these to show a message as soon as the user is offline (for example).

```
<template>
  <div>
    <div v-if="$nuxt.isOffline">You are offline</div>
    <Nuxt />
  </div>
</template>
```

Accessing the root instance

Besides providing DX/UX (Developer Experience / User Experience) features, the `$nuxt` helper also provides a shortcut to the root instance of your application from every other component. But that's not everything — you can also access the `$nuxt` helper through `window.$nuxt` which can be used as an escape hatch to gain access to module methods like `$axios` from outside your Vue components. This should be used wisely and **only as last resort**.

Refreshing page data

When you want to refresh the current page for the user, you don't want to fully reload the page because you might hit the server again and at least re-initialize the whole Nuxt application. Instead, you often only want to refresh the data, provided by `asyncData` or `fetch`.

You can do so, by using `this.$nuxt.refresh()`!

```
<template>
  <div>
    <div>{{ content }}</div>
    <button @click="refresh">Refresh</button>
  </div>
</template>

<script>
  export default {
    asyncData() {
      return { content: 'Created at: ' + new Date() }
    },
    methods: {
      refresh() {
        this.$nuxt.refresh()
      }
    }
  }
</script>
```

Controlling the loading bar

With `$nuxt`, you can also control Nuxt's loading bar programmatically via `this.$nuxt.$loading`.

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

Read more in the corresponding [loading feature chapter](#)

onNuxtReady helper

If you want to run some scripts *after* your Nuxt application has been loaded and is ready, you can use the `window.onNuxtReady` function. This can be useful when you want to execute a function on the client-side without increasing the time to interactive of your site.

```
window.onNuxtReady(() => {
  console.log('Nuxt is ready and mounted')
})
```

Process helpers

Nuxt injects three boolean values (`client`, `server`, and `static`) into the global `process` object which will help you to determine whether your app was rendered on the server or fully on the client, as well as checking for static site generation. These helpers are available across your application and are commonly used in `asyncData` userland code.

```
<template>
  <h1>I am rendered on the {{ renderedOn }} side</h1>
</template>

<script>
  export default {
    asyncData() {
      return { renderedOn: process.client ? 'client' : 'server' }
    }
  }
</script>
```

In the example, `renderedOn` will evaluate to `'server'` when using server-side rendering and a user accesses the page directly. When the user would navigate to the page from another part of the application, e.g. by click on a `<NuxtLink>`, it will evaluate to `client`.

[Go to TOC](#)

Server side rendering

Server-side rendering (SSR), is the ability of an application to contribute by displaying the web-page on the server instead of rendering it in the browser. Server-side sends a fully rendered page to the client; the client's JavaScript bundle takes over which then allows the Vue.js app to [hydrate](#).

Node.js server required

A JavaScript environment is required to render your web page.

A Node.js server needs to be configured to execute your Vue.js application.

Extend and control the server

You can extend the server with `serverMiddleware` and control routes with middleware.

```
export default function (req, res, next) {
  console.log(req.url)
  next()
}

export default {
  serverMiddleware: ['~/server-middleware/logger']
}
```

Server vs Browser environments

Because you are in a Node.js environment you have access to Node.js objects such as `req` and `res`. You do not have access to the `window` or `document` objects as they belong to the browser environment. You can however use `window` or `document` by using the `beforeMount` or `mounted` hooks.

```
beforeMount () {
  window.alert('hello');
}
mounted () {
  window.alert('hello');
}
```

Server-side rendering steps with Nuxt

Step 1: Browser to Server

When a browser sends the initial request, it will hit the Node.js internal server. Nuxt will generate the HTML and send it back to the browser with results from executed functions, e.g. `asyncData`, `nuxtServerInit` or `fetch`. Hooks functions are executed as well.

Step 2: Server to Browser

The browser receives the rendered page from the server with the generated HTML. The content is displayed and the Vue.js hydration kicks in, making it reactive. After this process, the page is interactive.

Step 3: Browser to Browser

Navigating between pages with `<NuxtLink>` is done on the client side so you don't hit the server again unless you hard refresh the browser.

Caveats

window or document undefined

This is due to the server-side rendering. If you need to specify that you want to import a resource only on the client-side, you need to use the `process.client` variable.

For example, in your `.vue` file:

```
if (process.client) {
  require('external_library')
}
```

iOS and phone numbers

Some mobile Safari versions will automatically transform phone numbers into links. This will trigger a `No-deMismatch` warning as the SSR content doesn't match the website content anymore. This can make your app unusable on these Safari versions.

If you include telephone numbers in your Nuxt page, you have two options.

Use a meta tag to stop the transformation

```
<meta name="format-detection" content="telephone=no" />
```

Wrap your phone numbers in links

```
<!-- Example phone number: +7 (982) 536-50-77 -->
<template>
  <a href="tel: +7 (982) 536-50-77">+7 (982) 536-50-77</a>
</template>
```

Static Site Generation

With static site generation you can render your application during the build phase and deploy it to any static hosting services such as Netlify, GitHub pages, Vercel etc. This means that no server is needed in order to deploy your application.

Generating your site

When deploying your site in with `target:static` all your `.vue` pages will be generated into HTML and JavaScript files. All calls to APIs will be made and cached in a folder called `static` inside your generated content so that no calls to your API need to be made on client side navigation.

Step 1: Browser to CDN

When a browser sends the initial request, it will hit the CDN.

Step 2: CDN to Browser

The CDN will send the already generated HTML, JavaScript and static assets back to the browser. The content is displayed and the Vue.js hydration kicks in, making it reactive. After this process, the page is interactive.

Step 3: Browser to Browser

Navigating between pages with `<NuxtLink>` is done on the client side so you don't hit the CDN again and all API calls will be loaded from the already cached static folder even if you hard refresh the browser.

SPA Fallback

Pages that have been excluded from generation, by using the `generate.exclude` property will fallback to being a single page application. These pages will therefore not exist in the CDN and will be rendered on client side in the browser once the user navigates to that page.

Next

To learn more about the [generate property](#)

Updating your content

In order to get new content to your site from your API you will need to regenerate your site again. With most static sites hosting providers you can do this by pushing your changes to your main branch via the git command or via a pull request.

Preview Mode

The Preview mode will call your API or your CMS so you can see the changes live before deploying. See the [preview mode](#) on how to enable this feature.

[Go to TOC](#)

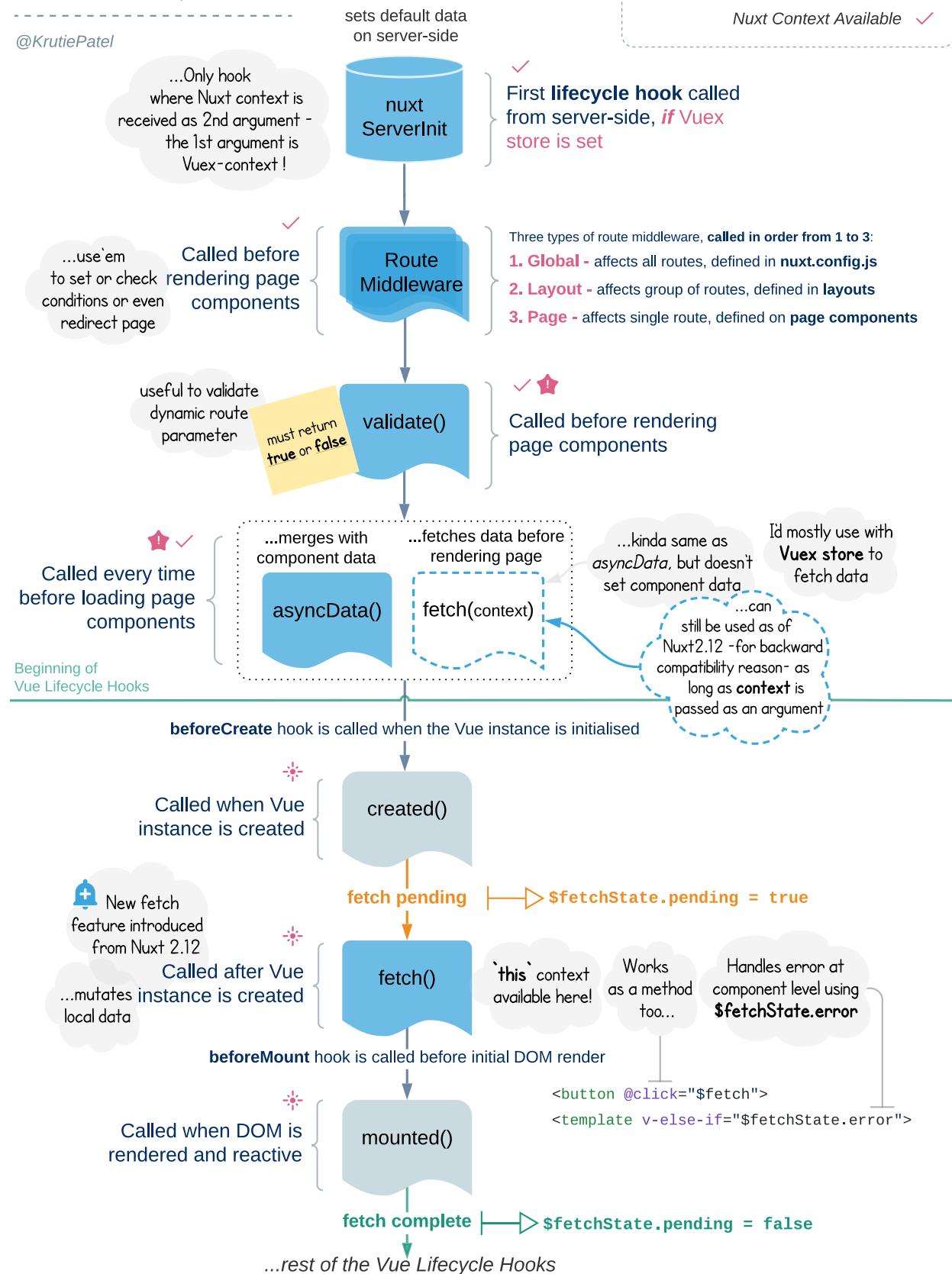
Nuxt Lifecycle

No matter which tool you use, you will always feel more confident when you understand how the tool works under the hood. The same applies to Nuxt.

NUXT.JS LIFECYCLE HOOKS

Nuxt >= 2.12 - April 2020

@KrutiePatel



The goal of this chapter is to give you a high-level overview of the different parts of the framework, their order of execution and how they work together.

The Nuxt lifecycle describes what happens after the build phase, where your application is bundled, chunked and minified. What happens after this phase depends on whether you have server-side rendering enabled or not. And if you do, it further depends on the type of server-side rendering you have chosen:

Dynamic SSR (`nuxt start`)

or Static Site Generation (`nuxt generate`).

Lifecycle

Server

For SSR, these steps will be executed for every initial request to your app

- The server starts (`nuxt start`)

When using static site generation, the server steps are only executed on build time, but once for every page that will be generated

- The generation process starts (`nuxt generate`)
- Nuxt hooks
- serverMiddleware
- Server-side Nuxt plugins
 - in order as defined in `nuxt.config.js`
- `nuxtServerInit`
 - Vuex action that is called only on server-side to pre-populate the store
 - First argument is the **Vuex context**, second argument is the **Nuxt context**
 - Dispatch other actions from here → only "entry point" for subsequent store actions on server-side
 - can only be defined in `store/index.js`
- Middleware
 - Global middleware
 - Layout middleware
 - Route middleware
- `asyncData`
- `beforeCreate` (Vue lifecycle method)
- `created` (Vue lifecycle method)
- The new fetch (top to bottom, `siblings = parallel`)

- Serialization of state (`render:routeContext` Nuxt hook)
- the HTML rendering happens (`render:route` Nuxt hook)
- `render:routeDone` hook when HTML has been sent to the browser
- `generate:before` Nuxt hook
- HTML files are generated
 - **Full static generation**
 - e.g. static payloads are extracted
 - `generate:page` (HTML editable)
 - `generate:routeCreated` (Route generated)
- `generate:done` when all HTML files have been generated

Client

This part of the lifecycle is fully executed in the browser, no matter which Nuxt mode you've chosen.

- Receives the HTML
- Loading assets (e.g. JavaScript)
- client-side Nuxt plugin
 - in order as defined in `nuxt.config.js`
- Vue Hydration
- Middleware
 - Global middleware
 - Layout middleware
 - Route middleware
- `asyncData` (blocking)
- `beforeCreate` (Vue lifecycle method)
- `created` (Vue lifecycle method)
- The new fetch (top to bottom, `siblings = parallel`) (non-blocking)
- `beforeMount` (Vue lifecycle method)
- `mounted` (Vue lifecycle method)

Navigate using the `NuxtLink` component

Same as for the *client* part, everything is happening in the browser but only when navigating via `<NuxtLink>`. Furthermore, no page content is displayed until all *blocking* tasks are fulfilled.

Info

Check out the component chapter to see more info on `<NuxtLink>`

- middleware (blocking)
 - Global middleware
 - Layout middleware

- Route middleware
- asyncData (blocking) [or full static payload loading]
- beforeCreate & created (Vue lifecycle methods)
- fetch (non-blocking)
- beforeMount & mounted

What's next

Next

Check out the [Features book](#)

Renderização

Sites Renderizados no Lado do Servidor e Sites Estáticos

Sites Renderizados no Lado do Servidor são renderizados no servidor a cada momento que o usuário requisitar uma página, então é necessário um servidor para encarregar-se de servir a página a cada requisição.

Sites Estáticos são muito similares as aplicações renderizadas no lado do servidor com a diferença principal sendo que sites estáticos são renderizados no momento de construção, então o servidor não é necessário. A navegação de uma página para outra está então no lado do cliente.

Consulte [alvos do deployment](#) para obter informações mais detalhas sobre hospedagem de sites estáticos e renderizados no servidor.

```
export default {  
  ssr: true // valor padrão  
}
```

Info

Você não precisa adicionar `ssr: true` a sua configuração do Nuxt com o fim de ativar a renderização no lado do servidor porque esta opção está ativada por padrão.

Renderizando Apenas no Lado do Cliente

Com a renderização apenas no lado do cliente não há renderização no lado do servidor. Renderização no lado do cliente significa renderizar o conteúdo no navegador usando JavaScript. Ao invés de receber todo o conteúdo a partir do HTML nós apenas recebemos um documento HTML básico com um ficheiro JavaScript que depois renderizará o resto no navegador. Para renderizar no lado do cliente defina o `ssr` para `false`.

```
export default {  
  ssr: false  
}
```

Next

[A propriedade ssr](#)

[Go to TOC](#)

Transições

O Nuxt usa o [componente de transição](#) para permitir que você criar animações e transições incríveis entre suas rotas.

Para definir uma transição personalizada para uma rota específica, adiciona a chave `transition` ao componente da página.

```
export default {
  // Pode ser uma Sequência de Caracteres
  transition: ''
  // Ou um Objeto
  transition: {}
  // Ou uma Função
  transition (to, from) {}
}
```

Sequência de Caracteres

Se a chave `transition` for definida como uma sequência de caracteres, ela será usada como o `transition.name`.

```
export default {
  transition: 'home'
}
```

O Nuxt usará essas configurações para definir o componente da seguinte forma:

```
<transition name="home"></transition>
```

Warning

Isto é feito automaticamente e você não precisa adicionar o componente `<transition>` a sua página ou esquemas (layouts).

Agora tudo o que você precisa fazer é criar a nova classe para suas transições.

```
<style>
  .home-enter-active, .home-leave-active { transition: opacity .5s; }
  .home-enter, .home-leave-active { opacity: 0; }
</style>
```

Objeto

Se a chave `transition` for definida como um objeto:

```
export default {
  transition: {
    name: 'home',
```

```
        mode: 'out-in'
    }
}
```

O Nuxt usará essas configurações para definir o componente da seguinte forma:

```
<transition name="home" mode="out-in"></transition>
```

O objeto `transition` pode ter várias propriedades tais como `name`, `mode`, `css`, `duration` e muitas mais. Consulte a documentação do vue para ter informações mais detalhadas.

Você também pode definir métodos dentro da propriedade `transition`, para saber mais sobre consulte os [Gatilhos de JavaScript](#) na documentação do vue.

```
export default {
  transition: {
    afterLeave(el) {
      console.log('afterLeave', el)
    }
  }
}
```

Modo de Transição

Warning

O modo de transição padrão para páginas diferem do modo padrão no Vue.js. O modo `transition` está por padrão definido como `out-in`. Se você quiser executar as transições de entrada e saída simultaneamente, você tem de definir o `mode` para uma string vazia `mode: ''`.

```
export default {
  transition: {
    name: 'home',
    mode: ''
  }
}
```

Função

Se a chave `transition` for definida como uma função:

```
export default {
  transition(to, from) {
    if (!from) {
      return 'slide-left'
    }
    return +to.query.page < +from.query.page ? 'slide-right' : 'slide-left'
  }
}
```

Transições aplicadas na navegação:

```
/ to /posts => slide-left, /posts to /posts?page=3 => slide-left, /posts?page=3 to /posts?page=2 => slide-right.
```

Configuração Global

O nome de transição padrão do Nuxt é `"page"`. Para adicionar uma transição fade para todas páginas da sua aplicação, tudo o que você precisa é um ficheiro CSS que é compartilhado por todas rotas.

Nosso css global dentro de `assets/main.css`:

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

Depois adicionarmos seu caminho ao array `css` dentro do nosso ficheiro de configuração `nuxt.config.js`:

```
export default {
  css: ['~/assets/main.css']
}
```

Configuração das Configurações

A Propriedade `layoutTransition`

A transição do esquema (ou layout) é usada para definir as propriedades da transições de esquema (layout).

A configuração padrão para transições de esquema são:

```
{
  name: 'layout',
  mode: 'out-in'
}
```

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

Se você quiser mudar as configurações padrões para as transições do seu esquema você pode fazer isso dentro do ficheiro `nuxt.config.js`.

```
export default {
  layoutTransition: 'my-layouts'
  // Ou
  layoutTransition: {
    name: 'my-layouts',
    mode: 'out-in'
  }
}
```

```

        mode: 'out-in'
    }
}
```

```

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

A Propriedade pageTransition

A configuração padrão para transições de página são:

```
{
    name: 'page',
    mode: 'out-in'
}
```

Deseja você modificar a configuração padrão, você pode fazer isso dentro do `nuxt.config.js`

```

export default {
    pageTransition: 'my-page'
    // Ou
    pageTransition: {
        name: 'my-page',
        mode: 'out-in',
        beforeEnter (el) {
            console.log('Before enter...');
        }
    }
}
```

Se você modificar o nome da transição da página você também terá de renomear a classe css.

```

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

[Go to TOC](#)

Modo de Pré-visualização

Pré-visualização ao vivo para alvo estático com o modo de pré-visualização

Com o Nuxt e completamente estático você pode agora usar a pré-visualização ao vivo fora da caixa que chamará sua API ou CMS, assim você pode visualizar as mudanças ao vivo antes de desdobrar.

Warning

Disponível apenas quando estiver usando `target:static`

O modo de pré-visualização irá atualizar os dados da página automaticamente visto que ele usa `$nuxt.refresh` nos bastidores e portanto chama `nuxtServerInit`, `asyncData` e `fetch` no lado do cliente.

No sentido de ativar a pré-visualização ao vivo você precisará adicionar o seguinte plugin:

```
export default function ({ query, enablePreview }) {
  if (query.preview) {
    enablePreview()
  }
}
```

Warning

O `enablePreview` está disponível apenas dentro do objeto de contexto dos plugins. Pré-visualizações são manipuladas no lado do cliente e desta maneira o plugin deve ser executado no cliente: `preview.client.js`

```
export default {
  plugins: ['~/plugins/preview.client.js']
}
```

Uma vez que você adicionou o plugin você pode agora gerar seu site e servir ele.

```
yarn generate
yarn start
```

```
npx nuxt generate
npx nuxt start
```

Em seguida você pode visualizar sua página de pré-visualização ao adicionar o parâmetro de consulta no final da página que você queira visualizar imediatamente:

```
?preview=true
```

Warning

O `enablePreview` deve ser testado localmente com o `yarn start` e não `yarn dev`

Pré-visualizando Páginas Que Ainda Não Foram Geradas

Para páginas que ainda não foram geradas, mesmo na queda do SPA elas ainda continuarão a chamar a API antes de exibir a página 404 como se essas páginas existissem na API mas que ainda não foram geradas.

Se você tiver definido um gatilho validate, você provavelmente precisará modificar ele, assim para que ele não redirecione para a página 404 no modo de pré-visualização.

```
validate({ params, query }) {
  if (query.preview) {
    return true
  }
}
```

Passando Dados ao enablePreview

Você pode passar dados para função `enablePreview`. Esses dados irão depois estar disponíveis no auxiliar de contexto `$preview` e no `this.$preview`.

O Que Se Segue

Next

Consulte o [livro da Estrutura do Diretório](#)

[Go to TOC](#)

Alvos do Desdobramento

Hospedagem Estática

O Nuxt também funciona como um gerador de sítio estático. Renderiza estaticamente a sua aplicação Nuxt e receba todos os benefícios de uma aplicação universal sem um servidor. O comando `nuxt generate` gerará um versão estática do seu website. Ele gerará um HTML para cada rota sua e a põe dentro do seu próprio ficheiro dentro do diretório `dist/`. Isto melhora o desempenho como também o SEO e refinar o suporte do offline.

Info

Rotas dinâmicas são também geradas graças ao [Nuxt Crawler](#)

Para sítios estáticos o alvo `static` precisa ser adicionado ao seu ficheiro `nuxt.config`.

```
export default {
  target: 'static' // o padrão é 'server'
}
```

Executar o `nuxt dev` com o alvo estático melhorara a experiência do desenvolvedor:

- Remove `req` e o `res` do `context`
- Recua para renderização no lado cliente no 404, erros e redireciona [consulte o recuo da aplicação de página única](#)
- `$route.query` será sempre igual ao `{}` na renderização no lado do servidor
- `process.static` é true

Info

Nós também estamos expondo `process.target` para os autores do módulo para adicionar a lógica dependendo do alvo do usuário.

Hospedagem de Servidor

Hospedagem do servidor significa executar o Nuxt em um servidor Node.js. Sempre que os usuários abrirem a sua página, os navegadores deles irão pedir aquela página a partir servidor. Nuxt manipulará a requisição, renderizará a página e enviará de volta a página resultante com todo o seu conteúdo.

Você pode precisar hospedar o servidor se você quiser renderizar o HTML a cada requisição de preferência ao momento de produção, ou se você precisar [serverMiddleware](#).

Info

Você pode continuar a executar o Nuxt com a hospedagem do servidor com `ssr: false` mas o Nuxt não renderizará de todo o HTML para cada página - deixando esse trabalho para o navegador. Você poderia escolher essa opção se você precisar do serverMiddleware mas não deseja que o HTML seja completamente renderizado no lado do servidor.

Para hospedagem do servidor, `target: 'server'` é usado, que é o valor padrão. Você usará o comando `build` para construir sua aplicação.

```
export default {  
  target: 'server'  
}
```

Sistema de Roteamento de Ficheiro

O Nuxt gera automaticamente a configuração do vue-router baseado na sua árvore de ficheiro de ficheiros Vue dentro do diretório pages. Sempre que você criar um ficheiro .vue dentro do seu diretório pages você terá o roteamento básico funcionando sem a necessidade de configurações extras.

Algumas vezes você pode precisar criar rotas dinâmicas ou rotas aninhadas ou você pode precisar avançar para configurar a propriedade router. Este capítulo irá através de tudo que você precisa saber no sentido de tirar o melhor proveito do seu roteador.

Info

O Nuxt oferece a você separação automática de código para suas rotas, sem precisar de configuração

Info

Use o [componente NuxtLink](#) para navegar entre páginas

```
<template>
  <NuxtLink to="/">Home page</NuxtLink>
</template>
```

Rotas Básicas

Esta árvore de ficheiro:

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

automaticamente gerará:

```
router: {
  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'
    }
  ]
}
```

```
    }
]
```

Rotas Dinâmicas

Algumas vezes não é possível saber o nome da rota tal como quando fazemos uma chamada para uma API para receber uma lista de usuários ou publicações de um blogue. Nós chamamos esses de rotas dinâmicas. Para criar uma rota dinâmica você precisa adicionar um sublinhado (`_`) antes do nome do ficheiro `.vue` ou antes do nome do diretório. Você pode nomear o ficheiro ou diretório com o que você quiser mas você deve prefixar ele com sublinhado.

Esta árvore de ficheiro:

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

automaticamente gerará:

```
router: {
  routes: [
    {
      name: 'index',
      path: '/',
      component: 'pages/index.vue'
    },
    {
      name: 'users-id',
      path: '/users/:id?',
      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'
    }
  ]
}
```

Info

Como você pode ver a rota nomeada `users-id` tem o caminho `:id?` o que a torna opcional, se você quiser torna ela obrigatória, crie um ficheiro `index.vue` dentro do diretório `users/_id`.

Info

Visto que desde o Nuxt \geq v2.13 há um rasteador instalado que agora irá rastrear em suas tags de ligação e gerar suas rotas dinâmicas baseadas naquelas ligações. No entanto se você tiver páginas que não estão ligadas tais como uma página secreta, então você precisará manualmente gerar aquelas rotas dinâmicas.

Next

[Gerar rotas dinâmicas](#) para sites estáticos.

Acessando Localmente Parâmetros de Rota

Você pode acessar os parâmetros da rota atual dentro da sua página local ou componente ao referenciar `this.$route.params.{parameterName}`. Por exemplo, se você tinha uma página de usuários dinâmica (`users/_id.vue`) e quis acessar o parâmetro `id` para carregar o usuário ou processar informações, você poderia acessar a variável como: `this.$route.params.id`.

Rotas Aninhadas

O Nuxt permite você criar rotas aninhadas ao usar rotas filhas do vue-router. Para definir o componente pai de uma rota alinhada, você precisa criar um ficheiro Vue com o mesmo nome que o diretório o qual contém suas views filhas.

Warning

Não esqueça de incluir o [componente NuxtChild](#) dentro do componente pai (ficheiro `.vue`).

Esta árvore de ficheiro:

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

automaticamente gerará:

```
router: {
  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'
        }
      ]
    }
  ]
}
```

```

        }
    ]
}
```

Rotas Dinâmicas Aninhadas

Este não é um quadro comum, mas é possível com o Nuxt ter filhos dinâmicos dentro de pais dinâmicos.

Esta árvore de ficheiro:

```

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

automaticamente gerará:

```

router: {
  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'
            }
          ]
        }
      ]
    }
  ]
}
```

Rotas Dinâmicas Aninhadas Desconhecidas

Se você não sabe a profundidade da sua estrutura de URL, você pode usar `_.vue` para casar dinamicamente caminhos aninhados. Isto irá manipular requisições que não casarem uma rota *mais específica*.

Esta árvore de ficheiro:

```
pages/
--| people/
-----| _id.vue
-----| index.vue
--| _.vue
--| index.vue
```

automaticamente gerará:

```
/ -> index.vue
/people -> people/index.vue
/people/123 -> people/_id.vue
/about -> _.vue
/about/careers -> _.vue
/about/careers/chicago -> _.vue
```

Info

Manipulação de páginas 404 agora sobre a lógica da página `_.vue`.

Estendendo o Roteador

Há várias maneiras de estender o roteamento com o Nuxt:

- `router-extras-module` para personalizar os parâmetros dentro da página
- componente `@nuxtjs/router` para sobrescrever o router do Nuxt e escrever seu próprio ficheiro `router.js`
- Usar a propriedade `router.extendRoutes` dentro do seu `nuxt.config.js`

A Propriedade router

A propriedade `router` permite você personalizar o router do Nuxt (`vue-router`).

```
export default {
  router: {
    // personalize o router do Nuxt
  }
}
```

Base:

A URL base da aplicação. Por exemplo, se a aplicação de página única inteira estiver servida debaixo de `/app/`, então a base deve usar o valor `'/app/'`.

Next

[Propriedade base do roteador](#)

extendRoutes

Você pode querer estender as rotas criadas pelo Nuxt. Você pode fazer então via opção `extendRoutes`.

Exemplo de adição de uma rota personalizada:

```
export default {
  router: {
    extendRoutes(routes, resolve) {
      routes.push({
        name: 'custom',
        path: '*',
        component: resolve(__dirname, 'pages/404.vue')
      })
    }
  }
}
```

Se você quiser organizar suas rotas, você pode usar a função `sortRoutes(routes)` do `@nuxt/utils`:

```
import { sortRoutes } from '@nuxt/utils'
export default {
  router: {
    extendRoutes(routes, resolve) {
      // Adicione algumas rota aqui ...

      // e depois organize elas
      sortRoutes(routes)
    }
  }
}
```

Warning

O estrutura de rotas deve respeitar a estrutura do [vue-router](#).

Warning

Quando estiver adicionando rotas que usam [Views Nomeadas](#), não esqueça de adicionar `chunkNames` correspondente dos `components` nomeados.

```
export default {
  router: {
    extendRoutes(routes, resolve) {
      routes.push({
        path: '/users/:id',
        components: {
          default: resolve(__dirname, 'pages/users'), // or
          routes[index].component
          modal: resolve(__dirname, 'components/modal.vue')
        },
        chunkNames: {
          modal: 'components/modal'
        }
      })
    }
  }
}
```

```

        }
    })
}
}
```

Next[Propriedade extendRoutes](#)**fallback**

Controla se o roteador deveria recuar para o modo hash quando o browser não suportar o `history.pushState` mas o modo está definido para history.

Next[Propriedade fallback](#)**mode**

Configura o modo do roteador, não é recomendado mudar ele por causa da renderização do lado do servidor.

Next[Propriedade mode](#)**parseQuery / stringifyQuery**

Fornece as funções parse / stringify da string de consulta personalizada

Next[Propriedade parseQuery / stringifyQuery](#)**routeNameSplitter**

Você pode querer mudar o separador entre os nomes de rota que o Nuxt usa. Você pode fazer isso via a opção `routeNameSplitter` dentro do seu ficheiro de configuração. Imagine que nós temos ao ficheiro de página `pages/posts/_id.vue`. O Nuxt gerará a nome da rota programaticamente, neste caso `posts-id`. Mudando a configuração do `routerNameSplitter` para `/` o nome irá portanto mudar para `posts/id`.

```

export default {
  router: {
    routeNameSplitter: '/'
  }
}
```

scrollBehavior

A opção `scrollBehavior` permite você definir um comportamento personalizado para a posição da rolagem entre as rotas. Este método é chamado toda vez que a página é renderizada.

Next

Para aprender mais sobre ele, veja a [documentação do vue-router para scrollBehavior](#).

Disponível desde a versão 2.9.0:

No Nuxt você pode usar o ficheiro para sobrescrever o scrollBehavior do router. Este ficheiro deve ser posto dentro da uma pasta chamada app.

```
~/app/router.scrollBehavior.js .
```

Exemplo de como forçar a posição da rolagem para o topo de toda rota:

```
export default function (to, from, savedPosition) {
  return { x: 0, y: 0 }
}
```

Next

Ficheiro `router.scrollBehavior.js` padrão do Nuxt.

Next

[Propriedade scrollBehavior](#)

trailingSlash

Disponível desde a versão: 2.10

Se este opção estiver definida para true, rastros de barras serão somados a toda rota. Se definida para false, eles serão removidos,

```
export default {
  router: {
    trailingSlash: true
  }
}
```

Warning

Esta opção não deve ser definida sem preparação e ter sido testado cuidadosamente. Quando estiver definindo `router.trailingSlash` para alguma coisa diferente de `undefined` (o qual é o valor padrão), a rota oposta irá parar de funcionar. Assim redireções 301 devem estar no lugar e sua *ligação interna* tem de ser adaptada corretamente. Se você definir `trailingSlash` para `true`, só assim o `example.com/abc/` irá funcionar mas não `example.com/abc`. No false, é vice-versa.

Next

[Propriedade trailingSlash](#)

[Go to TOC](#)

Requisição de Dados

Dentro do Nuxt, nós temos duas maneiras de receber dados de uma API. Podemos usar o método `fetch` ou o método `asyncData`.

O Nuxt suporta os padrões tradicionais do Vue para o carregamento de dados em sua aplicação no lado do cliente, tais como requisição de dados dentro de um gatilho `mounted()` do componente. As aplicações universais, por outro lado, precisam usar gatilhos específicos do Nuxt para serem capazes de renderizar dados durante a renderização no lado do servidor. Isto permite a sua página renderizar com todos os seus dados obrigatórios presente.

O Nuxt tem dois gatilhos para carregamento assíncrono de dados:

- `asyncData`. Este gatilho só pode ser colocado nos componentes `page`. Ao contrário do `fetch`, este gatilho não exibindo um placeholder de carregamento durante a renderização no lado do cliente: ao invés disso, este gatilho bloqueia a navegação da rota até estiver resolvida, exibindo uma página de erro se ele falhar.
- `fetch` (Nuxt 2.12+). Este gatilho pode ser colocado em qualquer componente, e fornece atalhos para renderização para estados do carregamentos (durante a renderização no lado do cliente) e erros.

Estes gatilhos podem ser usados com *qualquer biblioteca de requisição de dados* que você escolher. Nós recomendamos usar [@nuxt/http](#) ou [@nuxt/axios](#) para fazer requisições para APIs em HTTP. Mais informações sobre estas bibliotecas, tais como guias para configuração de cabeçalhos de autenticação, podem ser achadas em suas respectivas documentações.

Info

Se você definir `fetch` ou `asyncData` dentro de um mixin e também tem ele definido dentro de uma página/componente, a função mixin será sobreescrita ao invés de ser chamada.

O Gatilho Fetch

Info

Antes do Nuxt 2.12, havia um gatilho `fetch` diferente que só funcionava para componentes `page` e não tinha acesso a instância do componente.

Se o seu `fetch()` aceitar um argumento `context`, ele será tratado como um gatilho `fetch` legado. Esta funcionalidade está depreciada, e deve ser substituídos tanto com o `asyncData` ou um [intermediário anônimo](#).

`fetch` é um gatilho chamado durante a renderização no lado do cliente depois da instância do componente ser criada, e no cliente quando estiver navegando. O gatilho `fetch` deve retornar uma promessa (seja explicitamente, ou implicitamente usando `async/await`) que será resolvida:

- No servidor, antes da página inicial ser renderizada

- No cliente, algum tempo depois que o componente ser montado

Info

Para [hospedagem estática](#), o gatilho `fetch` é somente chamado durante a geração da página, e o resultado é depois cacheado para ser usado no cliente. Para evitar conflitos, pode ser necessário especificar um nome para o seu componente, ou alternativamente fornecer uma implementação única `fetchKey`.

Uso

Requisitando Dados

Dentro do gatilho `fetch`, você terá acesso a instância do componente via `this`.

Info

Certifique-se de que quaisquer propriedades que você queira modificar tenha já sido declarada dentro do `data()`. Então os dados que vierem da requisição possam ser atribuídas a essas propriedades.

Mudando o Comportamento do Fetch

`fetchOnServer`: `Boolean` ou `Function` (padrão: `true`), chama `fetch()` sempre que estiver renderizando a página no lado do servidor

`fetchKey`: `String` ou `Function` (padrões para o escopo ID do componente ou nome do componente), uma chave (ou uma função que produz uma chave única) que identifica o resultado deste fetch do componente (disponível no Nuxt 2.15+). Sempre que estiver hidratando uma página renderizada no lado do servidor, esta chave é será para mapear o resultado do `fetch()` no lado do servidor para os dados do componente no lado do cliente. [Mais informações estão disponíveis dentro do PR original](#)

`fetchDelay`: `Integer` (padrão: `200`), define o tempo mínimo de execução em milissegundos (para evitar piscadas)

Sempre que o `fetchOnServer` for falsy (`false` ou retorna `false`), `fetch` será chamado somente no lado do cliente e `$fetchState.pending` retornará `true` quando estiver renderizando o componente no lado do servidor.

```
export default {
  data: () => ({
    posts: []
  }),
  async fetch() {
    this.posts = await this.$http.$get('https://api.nuxtjs.dev/posts')
  },
  fetchOnServer: false,
  // dois ou vários componentes podem retornar a mesma `fetchKey` e o Nuxt irá
  // rastrear elas ambas separadamente
  fetchKey: 'site-sidebar',
  // alternativamente, para mais controle, uma função pode ser passado com acesso
  // a instância do componente
  // Ele será chamado dentro do `created` e não deve depender dos dados
  // requisitados
  fetchKey(getCounter) {
    // getCounter é um método que pode ser chamado para receber o próximo número
  }
}
```

```

dentro de uma sequência
    // como parte da geração de uma fetchKey única.
    return this.someOtherData + getCounter('sidebar')
}
}

```

Acessando o Estado do Fetch

O gatilho `fetch` expõem `this.$fetchState` no nível do componente com as seguintes propriedades:

- `pending` é um `Boolean` que permite você exibir um placeholder quando `fetch` estiver sendo chamado *no lado do cliente*.
- `error` é um `null` ou um `Error` lançando pelo gatilho `fetch`
- `timestamp` é uma referência a data e hora do ultimo `fetch`, útil para [cacheamento com keep-alive](#)

Adicionalmente ao `fetch` ser chamado pelo Nuxt, você pode manualmente chamar o `fetch` dentro do seu componente (para por exemplo recarregar seus dados assincronamente) ao chamar `this.$fetch()`.

```

<template>
  <div>
    <p v-if="$fetchState.pending">Fetching mountains...</p>
    <p v-else-if="$fetchState.error">An error occurred :(</p>
    <div v-else>
      <h1>Nuxt Mountains</h1>
      <ul>
        <li v-for="mountain of mountains">{{ mountain.title }}</li>
      </ul>
      <button @click="$fetch">Refresh</button>
    </div>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        mountains: []
      }
    },
    async fetch() {
      this.mountains = await fetch(
        'https://api.nuxtjs.dev/mountains'
      ).then(res => res.json())
    }
  }
</script>

```

Info

Você pode acessar o `contexto` do Nuxt dentro do gatilho `fetch` ao usar `this.$nuxt.context`.

Observando as Mudanças na Sequência de Caracteres de Consulta

O gatilho `fetch` não é chamado sobre as mudanças da string de consulta por padrão. Para vigiar as mudanças na consulta você pode adicionar um vigia sobre `$route.query` e depois chamar `$fetch`:

```
export default {
  watch: {
    '$route.query': '$fetch'
  },
  async fetch() {
    // Chamado também sobre as mudanças da consulta
  }
}
```

Cacheamento

Você pode usar a diretiva `keep-alive` dentro do componente `<nuxt/>` e `<nuxt-child/>` para guardar as chamados do `fetch` na páginas que você já visitou:

```
<template>
  <nuxt keep-alive />
</template>
```

Você pode também especificar as `props` passadas para o `<keep-alive>` ao passar uma propriedade `keep-alive-props` para o componente `<nuxt>` :

```
<nuxt keep-alive :keep-alive-props="{ max: 10 }" />
```

Mantenha apenas 10 componentes de página dentro da memória.

Manipulação de Erro

Warning

Se houver um erro durante a requisição dos dados, a página erro normal do Nuxt não será carregado - e você não deve usar o `redirect` do Nuxt ou o método `error` dentro do `fetch()`. Ao invés disso, você precisará manipular ele dentro do seu componente usando `$fetchState.error`.

Podemos verificar o `$fetchState.error` e exibir uma mensagem de erro se houver um erro na requisição dos dados.

```
<template>
  <div>
    <p v-if="$fetchState.pending">Loading....</p>
    <p v-else-if="$fetchState.error">Error while fetching mountains</p>
    <ul v-else>
      <li v-for="(mountain, index) in mountains" :key="index">
        {{ mountain.title }}
      </li>
    </ul>
  </div>
</template>
<script>
  export default {
    data() {
      return {
        mountains: []
      }
    },
    async fetch() {
      this.mountains = await fetch(
```

```

        'https://api.nuxtjs.dev/mountains'
    ).then(res => res.json())
}
</script>

```

Usando o gatilho `activated`

O Nuxt irá diretamente preencher o `this.$fetchState.timestamp` (data e hora) do última chamada do `fetch` (inclusive no SSR). Você pode usar esta propriedade combinada com o gatilho `activated` para adicionar um cache de 30 segundos para `fetch`:

```

<template> ... </template>

<script>
export default {
  data() {
    return {
      posts: []
    }
  },
  activated() {
    // Chama o fetch novamente se o último fetch foi a mais de 30 segundos
    // atrás.
    if (this.$fetchState.timestamp <= Date.now() - 30000) {
      this.$fetch()
    }
    async fetch() {
      this.posts = await fetch('https://api.nuxtjs.dev/posts').then(res =>
        res.json()
      )
    }
  }
}
</script>

```

A navegação para a mesma página não chamará o `fetch` se a última chamada do `fetch` antes de 30 segundos atrás.

Dados Assíncronos

Warning

`asyncData` está apenas disponível para `páginas` e você não tem acesso ao `this` dentro do gatilho.

`asyncData` é outro gatilho para requisição de dados universal. Ao contrário do `fetch`, que requer que você defina propriedades na instância do componente (ou despachar ações do Vuex) para guardar seu estado assíncrono, `asyncData` simplesmente funde seu valor de retorno dentro do seu estado local do componente. Aqui está um exemplo usando a biblioteca `@nuxt/http`:

```

<template>
<div>
  <h1>{{ post.title }}</h1>
  <p>{{ post.description }}</p>
</div>
</template>

```

```
<script>
  export default {
    async asyncData({ params, $http }) {
      const post = await $http.$get(`https://api.nuxtjs.dev/posts/${params.id}`)
      return { post }
    }
  }
</script>
```

Ao contrário do `fetch`, a promessa retornada pelo gatilho `asyncData` é resolvida durante a transição da rota. Isto significa que nenhum "placeholder de carregamento" está visível durante as transições do lado do cliente (se bem que a barra de carregamento pode ser usada para indicar um estado de carregamento ao usuário). O Nuxt irá ao invés disso esperar que o `asyncData` seja terminado antes de navegar para a próxima página ou exibir a [página de erro](#).

Este gatilho pode somente ser usado para componentes do nível de página. Ao contrário do `fetch`, `asyncData` não consegue acessar a instância do componente (`this`). Ao invés disso, ele recebe o contexto como seu argumento. Você pode usar ele para pedir alguns dados e o Nuxt irá automaticamente fazer uma fusão superficial do objeto retornado com os dados do componente.

Dentro dos exemplos que estão por vir, estamos usando o [@nuxt/http](#) o qual recomendamos para requisição de dados de uma API.

Dados Assíncrono em Componentes?

Visto que os componentes não têm um método `asyncData`, você não consegue diretamente pedir dados assíncronos do lado do servidor dentro de um componente. No sentido de dar a volta a esta limitação você tem três opções básicas:

1. Usar o novo gatilho `fetch` que está disponível no Nuxt 2.12 e posteriores versões.
2. Fazer a API chamar dentro do gatilho `mounted` e definir propriedades de dados sempre que for carregado. *Desvantagem: Não funcionará para a renderização no lado do servidor*
3. Fazer a API chamar dentro do método `asyncData` do componente da página e passar os dados como propriedades para os subcomponentes. Funcionará bem no lado do servidor. *Desvantagem: o `asyncData` da página pode ser menos legível porque está carregando os dados para outros componentes.*

Ouvindo Mudanças na Consulta

O método `asyncData` não é chamado sobre as mudanças da string de consulta por padrão. Se você quiser mudar este comportamento, por exemplo quando estiver construindo um componente de paginação, você pode configurar parâmetros que devem ser ouvidos com a propriedade `watchQuery` do seu componente de página.

Warning

A propriedade `watchQuery` não funcionará apropriadamente para sitios gerados estáticos. Para fazer isso funcionar use o gatilho `fetch` e observe as mudanças na sequência de caracteres de consulta.

Next

Aprenda mais sobre a [propriedade watchQuery](#) e veja a lista de [chaves disponíveis dentro do contexto](#).

Marcadores de Meta e SEO

O Nuxt dá para você 3 maneiras diferentes de adicionar meta dados a sua aplicação:

```
div{.d-heading-description .leading-6}
```

- Usando o `nuxt.config.js` globalmente
- Usando o `head` como um objeto localmente
- Localmente usando o `head` como uma função assim você tem acesso ao dados ou propriedades computadas.

Configuração Global

O Nuxt permite você definir todos marcadores de metas padrão para a sua aplicação dentro do ficheiro `nuxt.config.js` com o uso da propriedade `head`. Isto é muito útil para a adição dos marcadores de título e descrição com o fim de SEO ou configuração da janela de visualização ou adição do favicon.

```
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'
      }
    ],
    link: [{ rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' }]
  }
}
```

Info

Isto lhe dará o mesmo `title` e `description` em todas as páginas

Configuração Local

Você pode também adicionar títulos e meta para cada página pela configuração da propriedade `head` dentro da sua marcação de script em cada página:

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

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

Info

Use o `head` como um objeto para definir o `title` e a `description` somente para a página inicial (home)

```
<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>
```

Info

Use o `head` como uma função para definir um `title` e uma `description` somente para a página inicial (home). Ao usar uma função você tem acesso aos dados e propriedades computadas.

O Nuxt usa `vue-meta` para atualizar o `head` e atributos meta do documento da sua aplicação.

Warning

Para evitar qualquer duplicação sempre que usado em componentes filhos, dê um identificador único com a chave `hid` para a descrição de meta. Desta maneira `vue-meta` saberá que ele tem de sobrescrever o marcador padrão.

Next

Aprenda mais sobre as opções disponíveis para `head`, na [documentação do vue-meta](#).

Recursos Externos

Você pode incluir recursos externos tais como scripts e fontes ao adicionar eles globalmente ao `nuxt.config.js` ou localmente dentro do objeto ou função `head`.

Info

Você também pode passar para cada recurso um `body: true` opcional para incluir os recursos antes do fechamento da tag `</body>`.

Configuração Global

```
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'
      }
    ]
  }
}
```

Configuração Local

```
<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>
```

[Go to TOC](#)

Configuração

Por padrão, o Nuxt está configurado para cobrir a maioria dos casos de uso. Esta configuração padrão pode ser sobreescrita com o ficheiro `nuxt.config.js`.

A Propriedade CSS

O Nuxt permite você definir os ficheiros/módulos/bibliotecas CSS que você quiser definir globalmente (incluído dentro de cada página).

Warning

Neste caso você querer usar `sass` certifique-se de que você tem instalado os pacotes `sass` e `sass-loader`.

Dentro do `nuxt.config.js`, adicione os recursos do CSS:

```
export default {
  css: [
    // Carrega um módulo Node.js diretamente (aqui está um ficheiro Sass)
    'bulma',
    // ficheiro CSS dentro do projeto
    '~/assets/css/main.css',
    // ficheiro SCSS dentro do projeto
    '~/assets/css/main.scss'
  ]
}
```

Warning

O Nuxt irá automaticamente deduzir o tipo do ficheiro pela sua extensão e usar o carregador de pré-processador apropriado para o webpack. Você continuará a precisar instalar o carregador solicitado se você precisar usar eles.

Extensões do Estilo

Você pode omitir a extensão do ficheiro para os ficheiros CSS/SCSS/PostCSS/Less/Stylus... listados dentro do array `css` dentro do seu ficheiro de configuração.

```
export default {
  css: ['~/assets/css/main', '~/assets/css/animations']
}
```

Warning

Se você tem dois ficheiros com o mesmo nome, por exemplo `main.scss` e `main.css`, e não especifica uma extensão dentro o array de entrada `css`, por exemplo `css: ['~/assets/css/main']`, então somente um ficheiro será carregado dependendo da ordem do `styleExtensions`. Neste caso apenas o ficheiro `css` será carregado e o ficheiro `scss` será ignorado porque o `css` vem em primeiro dentro do array `styleExtensions` padrão.

Ordem padrão: `['css', 'pcss', 'postcss', 'styl', 'stylus', 'scss', 'sass', 'less']`

Pré-processadores

Graças ao [Vue Loader](#), você pode usar qualquer tipo de pré-processador para o seu `<template>` ou `<style>`: use o atributo `lang`.

Exemplo do nosso `pages/index.vue` usando [Pug](#) e [Sass](#):

```
<template lang="pug">
  h1.red Hello {{ name }}!
</template>

<style lang="scss">
  .red {
    color: red;
  }
</style>
```

Para usar estes pré-processadores, precisamos instalar seus carregadores para o webpack:

```
yarn add --dev pug pug-plain-loader
yarn add --dev sass sass-loader@10
```

```
npm install --save-dev pug pug-plain-loader
npm install --save-dev sass sass-loader@10
```

Recursos Externos

Configuração Global

Você pode incluir seus recursos externos dentro da função ou objeto `head`. Assim como é descrito dentro da [documentação da API head](#), os exemplos seguintes mostram o uso do `head` como um objeto e como uma função. Se você quiser usar valores do seu componente Vue como propriedades computadas ou dados, você pode usar a função `head`, retornando o objeto `head` final. Você pode também passar cada recurso um `body: true` opcional para incluir o recurso antes do fechamento da tag `</body>`.

Inclua os seus recursos dentro do `nuxt.config.js` (aqui dentro do objeto `head`).

```
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'
            }
        ]
    }
}

```

Configuração Local

Inclua os seus recursos dentro do seu ficheiro `.vue` dentro do diretório `pages/` (aqui dentro da função `head`):

```

<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>

```

Plugins do PostCSS

Se apresentar, renomear ou eliminar o `postcss.config.js` dentro do diretório do projeto. Então, dentro do seu ficheiro `nuxt.config.js` para adicione o seguinte:

```

export default {
  build: {
    postcss: {
      // Adicione os nomes do plugins como chaves e argumentos como valores
      // Instale eles antes como dependências com o npm ou yarn
      plugins: {
        ...
      }
    }
  }
}

```

```
// Desative um plugin ao passar false como valor
  'postcss-url': false,
  'postcss-nested': {},
  'postcss-responsive-type': {},
  'postcss-hexrgba': {}
},
preset: {
  // Mude as configurações do postcss-preset-env
  autoprefixer: {
    grid: true
  }
}
}
```

JSX

O Nuxt usa [@nuxt/babel-preset-app](#), que é baseado [@vue/babel-preset-app](#) oficial para a configuração padrão do babel, assim você pode usar JSX dentro dos seus componentes.

Você pode também usar JSX dentro do método `render` dos seus componentes:

```
export default {
  data () {
    return { name: 'World' }
  },
  render (h) {
    return <h1 class="red">{this.name}</h1>
  }
}
```

Apelidar de `createElement` para `h` é uma convenção comum que você verá dentro do ecossistema porém é atualmente opcional para o JSX desde que ele [injetado automaticamente](#) `const h = this.$createElement` dentro de qualquer método e recebedor (getter) (não em funções ou funções em seta) declarado dentro da sintaxe ES2015 que tem o JSX assim você pode eliminar o parâmetro (`h`).

Você pode aprender mais sobre como usar ele dentro da [seção JSX](#) da documentação do Vue.js.

Ignorando Ficheiros

.nuxtignore

Você pode usar um ficheiro `.nuxtignore` para deixar o Nuxt ignorar `layout`, `page`, `store` e ficheiros `middleware` dentro do diretório raiz do seu projeto (`rootDir`) durante a fase de construção. O ficheiro `.nuxtignore` está sujeito a mesma especificação que os ficheiros `.gitignore` e `.eslintignore`, no qual cada linha é um padrão global indicando quais ficheiros devem ser ignorados.

```
# ignore o esquema foo.vue
layouts/foo.vue
# ignore os ficheiros de esquemas os quais o nome termina com -ignore.vue
```

```

layouts/*-ignore.vue
# ignore a página bar.vue

pages/bar.vue
# ignore a página dentro da pasta ignore

pages/ignore/*.vue
# ignore a memória baz.js

store/baz.js
# ignore os ficheiros de memória correspondente _.test._

store/ignore/_.test._

# ignore ficheiros intermediários dentro da pasta foo exceto foo/bar.js

middleware/foo/*.js !middleware/foo/bar.js

```

A Propriedade ignorePrefix

Qualquer ficheiro dentro pages/, layout/, middleware/ ou store/ será ignorado durante a construção se seu nome de ficheiro com o prefixo especificado pelo ignorePrefix.

Por padrão todos ficheiros que começam com `-` serão ignorados, tais como `store/-foo.js` e `pages/-bar.vue`. Isto permite colocar-se testes, utilidades, e componentes com seus chamadores sem eles mesmos serem convertidos em rotas, memórias etc.

A Propriedade ignore

Mais personalizável do que o ignorePrefix: todos ficheiros correspondentes aos padrões globais dentro de ignore serão ignorados durante a construção.

```

export default {
  ignore: 'pages/bar.vue'
}

```

A propriedade ignoreOptions

O `nuxtignore` está usando `node-ignore` nos bastidores, `ignoreOptions` pode ser configurado como `options` de `node-ignore`.

```

export default {
  ignoreOptions: {
    ignorecase: false
  }
}

```

Estender a Configuração do Webpack

Você pode estender a configuração do webpack do Nuxt via opção `extend` dentro do seu `nuxt.config.js`. A opção `extend` da propriedade `build` é um método que aceita dois argumentos. O primeiro argumento é o objeto `config` do webpack exportado a partir da configuração do webpack do Nuxt. O segundo parâmetro é um objeto de contexto com as seguintes propriedades booleanas: `{ isDev, isClient, isServer, loaders }`.

```
export default {
  build: {
    extend(config, { isDev, isClient }) {
      // ...
      config.module.rules.push({
        test: /\.(ttf|eot|svg|woff(2)?)(\?|[a-z0-9=&.]*)?$/,
        loader: 'file-loader'
      })
      // Defina o modo do webpack para desenvolvimento se a variável `isDev` for
      true.
      if (isDev) {
        config.mode = 'development'
      }
    }
  }
}
```

O método `extend` é chamada duas vezes - uma vez para o pacote do cliente e outra para o pacote do servidor.

Personalize Pedaços de Configuração

Você pode querer ajustar levemente a [configuração de otimização](#), evitando uma rescrita do objeto padrão.

```
export default {
  build: {
    extend(config, { isClient }) {
      if (isClient) {
        config.optimization.splitChunks.maxSize = 200000
      }
    }
  }
}
```

Inspecionar A Configuração do Webpack

Para projetos complexos e depurações faz-se útil verificar algumas vezes como será a configuração final do webpack. Felizmente você pode executar o comando `nuxt webpack` a partir de dentro do seu projeto para a saída da configuração. Para obter mais detalhes consulte esta PR [#7029](#).

Adicionar Plugins Ao Webpack

Dentro do seu ficheiro `nuxt.config.js`, por baixo da opção `build`, você pode passar `plugins` ao webpack, da mesma maneira que você faria dentro de [um ficheiro webpack.config.js](#).

Neste exemplo adicionamos o método `ProvidePlugin` embutido ao webpack para automaticamente carregar módulos do JavaScript (`lodash` and `jQuery`) ao invés de ter de os `import` ou os `require` por todos os lugares.

```
import webpack from 'webpack'

export default {
  build: {
    plugins: [
      new webpack.ProvidePlugin({
        // módulos globais
        $: 'jquery',
        _: 'lodash'
      })
    ]
  }
}
```

Note que: você pode não precisar do JQuery dentro de uma aplicação baseada no Vue.

Com o Nuxt, você pode também controlar o contexto de execução dos plugins: se eles serão executados nas construções no `client` ou dentro do `server` (ou diferenciando construções do `dev` e `prod`) dentro do `build.extend`, onde você pode também manualmente passar plugins ao webpack.

Estender o Webpack para Carregar Ficheiros de Audio

Ficheiros de audio devem ser processados pelo `file-loader`. Este carregador já está incluído dentro da configuração padrão do Webpack, mas não está configurado para lidar com ficheiros de audio. Você precisa estender sua configuração padrão dentro do `nuxt.config.js`:

```
export default {
  build: {
    extend(config, ctx) {
      config.module.rules.push({
        test: /\.(ogg|mp3|wav|mp3|mpe?g)$/.i,
        loader: 'file-loader',
        options: {
          name: '[path][name].[ext]'
        }
      })
    }
  }
}
```

Você pode agora importar ficheiros de audio como isto `<audio :src="require('@/assets/water.mp3')" controls></audio>`.

Se você apenas quiser escrever: `<audio src="@/assets/water.mp3" controls></audio>`, você precisar dizer ao `vue-loader` para automaticamente requerer ficheiros de audio sempre que você fizer referência a eles com o atributo `src`:

```

export default {
  build: {
    loaders: {
      vue: {
        transformAssetUrls: {
          audio: 'src'
        }
      }
    },
    extend(config, ctx) {
      config.module.rules.push({
        test: /\.(ogg|mp3|wav|mpe?g)$/.i,
        loader: 'file-loader',
        options: {
          name: '[path][name].[ext]'
        }
      })
    }
  }
}

```

Editar o Hospedeiro e a Porta

Por padrão, o servidor hospedeiro de desenvolvimento do Nuxt é `localhost` o qual somente é acessível a partir de dentro da máquina hospedeira. No sentido de ver sua aplicação em outro dispositivo você precisa modificar o hospedeiro. Você pode modificar o hospedeiro dentro do seu ficheiro `nuxt.config.js`.

O hospedeiro `'0.0.0.0'` está designado para dizer ao Nuxt para resolver o endereço do hospedeiro, o qual está acessível para conexões *fora* da máquina hospedeira (exemplo, LAN). Se ao hospedeiro é atribuído o valor string `'0'` (não 0, o qual é falsy), ou `0.0.0.0` o seu endereço de IP local será atribuído a sua aplicação Nuxt.

```

export default {
  server: {
    host: '0' // padrão: localhost
  }
}

```

Você pode também mudar a número da porta da porta padrão de 3000.

```

export default {
  server: {
    port: 8000 // padrão: 3000
  }
}

```

Info

Se a porta é atribuída o valor string de `'0'` (não 0, o qual é falsy) um número de porta aleatório será atribuído a sua aplicação Nuxt.

Se bem que você pode modificar isto dentro do ficheiro `nuxt.config.js`, não é aconselhado visto que isto pode causar a você problemas quando estiveres hospedando o seu site. É muito melhor modificar o hospedeiro e a porta direto dentro do comando `dev`.

```
HOST=0 PORT=8000 npm run dev
```

ou criar um script dentro do package.json

```
"scripts": {
  "dev:host": "nuxt --hostname '0' --port 8000"
}
```

Configuração Assíncrona

Embora seja melhor usar a configuração normal `export default {}` você pode ter uma configuração assíncrona ao exportar uma função assíncrona que retorna um objeto de configuração.

```
import axios from 'axios'

export default async () => {
  const data = await axios.get('https://api.nuxtjs.dev/posts')
  return {
    head: {
      title: data.title
      //... resto da configuração
    }
  }
}
```

Warning

O módulo axios não pode ser usado dentro do `nuxt.config.js`. Você precisará importar o axios e configurar ele novamente.

Configurações Avançadas

Next

O `nuxt.config.js` tem mais opções de personalização e configurações! Consulte todas suas chaves dentro do [glossário de configuração](#).

Carregamento

Fora da caixa, o Nuxt dá para você o seu próprio componente de barra de progresso de carregamento que é exibido entre as rotas. Você pode personalizar ele, desativar ele ou até criar seu próprio componente de carregamento.

Personalizando a Barra de Progresso

Entre outras propriedades, a color, o tamanho, a duração e a direção da barra de progresso podem ser personalizados para enquadram-se as necessidades da sua aplicação. Isto é feito pela atualização da propriedade `loading` do `nuxt.config.js` com as propriedades correspondentes.

Por exemplo, para definir uma barra de progresso azul com a altura de 5px, atualizamos o `nuxt.config.js` para o seguinte:

```
export default {
  loading: {
    color: 'blue',
    height: '5px'
  }
}
```

Lista de propriedades para personalizar a barra de progresso.

| Chave | Tipo | Padrão | Descrição | | | | color | String | 'black' | Cor CSS da barra de progresso | | | failedColor | String | 'red' | Cor CSS da barra de progresso quando um erro é anexado durante a renderização da rota (se os dados ou requisição enviada voltar com um erro por exemplo). | | | height | String | '2px' | Altura da barra de progresso (usada dentro da propriedade de estilo da barra de progresso) | | | throttle | Number | 200 | Em ms, espera pelo tempo especificado antes de exibir a barra de progresso. Útil para impedir a barra de piscar. | | | duration | Number | 5000 | Em ms, a duração máxima da barra de progresso, o Nuxt assume que a rota será renderizada antes de 5 segundos. | | | continuous | Boolean | false | Mantém a barra de progresso animando sempre que o carregamento levar mais tempo que a duração. | | | css | Boolean | true | Define para false para remover os estilos padrão da barra de progresso (e adicione você mesmo). | | | rtl | Boolean | false | Define a direção da barra de progresso da direita para a esquerda. | |

Desativar a Barra de Progresso

Se você não quiser exibir a barra de progresso entre as rotas adicione `loading: false` dentro do seu ficheiro `nuxt.config.js`:

```
export default {
  loading: false
}
```

A propriedade `loading` dá para você a opção de desativar a barra de progresso padrão em uma página específica.

```
<template>
  <h1>My page</h1>
</template>

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

Começar Programaticamente a Barra de Carregamento

A barra de carregamento também pode ser programaticamente iniciada dentro dos seus componentes ao chamar `this.$nuxt.$loading.start()` para iniciar a barra de carregamento e `this.$nuxt.$loading.-finish()` para terminar ele.

Durante o processo de montagem dos componentes da sua página, a propriedade `$loading` pode não estar imediatamente disponível para acessar. Para dar a volta a isso, se você quiser iniciar o carregador dentro do método `mounted`, certifique-se de envolver a sua chamada para o método `$loading` dentro de `this.$nextTick` como é mostrado abaixo.

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

Os Interiores da Barra de Progresso

Infelizmente, não é possível para o componente de carregamento saber de antemão quanto tempo uma nova página levará para carregar. Portanto, não é possível de maneira precisa animar a barra de progresso até 100% do tempo de carregamento.

O componente de carregamento do Nuxt resolve parcialmente isso ao deixar você definir a `duration`, isso deve ser definido para uma estimativa de quanto tempo o processo de carregamento levará. A menos que você use um componente de carregamento personalizado, a barra de progresso irá sempre mover de 0% para 100% em tempo de `duration` (independentemente da progressão atual). Sempre que o carregamento levar mais tempo do que o tempo em `duration`, a barra de progresso continuará em 100% até o carregamento terminar.

Você pode mudar o comportamento padrão ao configurar `continuous` para `true`, então depois de alcançar o 100% a barra de progresso começará reduzindo novamente de volta para 0% em tempo de `duration`. Sempre que o carregamento não estiver terminado depois de alcançar 0% ele começará crescendo novamente de 0% até 100%, isto é repetido até o carregamento terminar.

```
export default {
  loading: {
    continuous: true
  }
}
```

Exemplo de uma barra de progresso contínua:



.../.../.../static/img/docs/api-continuous-loading.gif

Usando um Componente de Carregamento Personalizado

Você pode também criar seu próprio componente que o Nuxt chamará no lugar do componente padrão para a barra de progresso. Para fazer isso no entanto, você precisa dar um caminho para o seu componente dentro da opção `loading`. Depois, o seu componente será chamado diretamente pelo Nuxt.

O seu componente tem que expor alguns desses métodos:

| Método | Obrigatório | Descrição | | | | start() | Obrigatório | Chamado sempre que uma rota muda, é aqui onde você exibe o seu componente. | | finish() | Obrigatório | Chamado sempre que uma rota é carregada (e dados são pedidos), é aqui onde você esconde o seu componente. | | fail(error) | Opcional | Chamado sempre que uma rota não puder ser carregada (falhou ao pedir os dados por exemplo). | | increase(num) | Opcional | Chamado durante o carregamento do componente de rota, num é um Inteiro menor que 100 (Integer < 100). |

Você pode criar o seu componente personalizado dentro de `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>

```

Depois, você atualiza seu `nuxt.config.js` para dizer ao Nuxt para usar o seu componente:

```

export default {
  loading: '~/components>LoadingBar.vue'
}

```

A Propriedade `loadingIndicator`

Não há conteúdo vindo do lado do servidor no primeiro carregamento da página, quando o Nuxt estiver executando no modo SPA. Então, ao invés de mostrar uma página em branco enquanto a página carrega, o Nuxt dá para você um rodopiador (spinner) o qual você pode personalizar para adicionar suas próprias cores, fundo e até mudar o indicador.

```

export default {
  loadingIndicator: {
    name: 'circle',
    color: '#3B8070',
    background: 'white'
  }
}

```

Indicadores Embutidos

Esses indicadores são importados do incrível projeto [SpinKit](#). Você pode consultar a sua página de demonstração para pré-visualizar os rodopiadores (spinners). No propósito de usar um desses rodopiadores tudo o que você tem de fazer é adicionar seu nome a propriedade `name`. Não precisa importar ou instalar nada. Aqui está uma lista dos indicadores embutidos para você usar.

- circle
- cube-grid
- fading-circle
- folding-cube
- chasing-dots
- nuxt
- pulse
- rectangle-bounce
- rotating-plane
- three-bounce

- wandering-cubes

Os indicadores embutidos suportam as opções `color` e `background`.

Indicadores Personalizados

Se você precisar seu próprio indicador especial, um valor do tipo String ou chave Name pode ser um caminho para o modelo (template) HTML do código-fonte do indicador! Todas as opções são passadas para o modelo também.

[Código-fonte](#) do componente Loading embutido do Nuxt!

[Go to TOC](#)

Componentes do Nuxt

O Nuxt vem com alguns componentes importantes incluídos fora da caixa, os quais serão úteis quando estiver construindo sua aplicação. Os componentes estão disponíveis globalmente, o que significa que você não precisa importar eles no sentido de usar eles.

Dentro dos parágrafos seguintes, cada componente incluído é explicado.

O Componente Nuxt

O componente `<Nuxt>` é o componente que você usa para exibir seus componentes de página. Basicamente, este componente é substituído pelo que está dentro dos componentes de página dependendo da página que está sendo exibida. Daí, é importante que você adicione o componente `<Nuxt>` aos seus esquemas.

```
<template>
  <div>
    <div>My nav bar</div>
    <Nuxt />
    <div>My footer</div>
  </div>
</template>
```

Warning

O componente `<Nuxt>` apenas pode ser usado dentro dos [esquemas](#).

O componente `<Nuxt>` pode carregar a propriedade `nuxt-child-key`. Esta propriedade será passada para `<RouterView>`, assim suas transições funcionarão corretamente dentro das páginas dinâmicas.

Há 2 maneiras de manipular a propriedade interna `key` de `<RouterView>`.

1. Use uma propriedade `nuxtChildKey` no seu componente `<Nuxt>`

```
<template>
  <div>
    <Nuxt :nuxt-child-key="someKey" />
  </div>
</template>
```

2. Adicione a opção `key` como `string` ou `função` dentro dos componentes de *página*

```
export default {
  key(route) {
    return route fullPath
  }
}
```

O Componente NuxtChild

Este componente é usado para exibição dos componentes filhos dentro de uma rota aninhada.

Example:

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

Esta árvore de ficheiro gerará estas rotas:

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

Para exibir o componente `child.vue`, você tem de inserir o componente `<NuxtChild>` dentro de `pages/parent.vue`:

```
<template>
  <div>
    <h1>I am the parent view</h1>
    <NuxtChild :foobar="123" />
  </div>
</template>
```

keep-alive

Ambos, o componente `<Nuxt>` e o componente `<NuxtChild/>`, aceitam `keep-alive` e `keep-alive-props`.

Info

Para aprender mais sobre o `keep-alive` e `keep-alive-props` consulte a [documentação do vue](#)

```
<template>
  <div>
    <Nuxt keep-alive :keep-alive-props="{ exclude: ['modal'] }" />
  </div>
</template>

<!-- será convertido em alguma coisa parecida com isto --&gt;
&lt;div&gt;</pre>

```

```
<KeepAlive :exclude="['modal']">
  <RouterView />
</KeepAlive>
</div>
```

```
<template>
  <div>
    <NuxtChild keep-alive :keep-alive-props="{ exclude: ['modal'] }" />
  </div>
</template>

<!-- será convertido em alguma coisa parecida com isto --&gt;
&lt;div&gt;
  &lt;KeepAlive :exclude="['modal']"&gt;
    &lt;RouterView /&gt;
  &lt;/KeepAlive&gt;
&lt;/div&gt;</pre>

```

Os componentes `<NuxtChild>` pode também receber propriedades como um componente normal do Vue.

```
<template>
  <div>
    <NuxtChild :key="$route.params.id" />
  </div>
</template>
```

Pare ver um exemplo, dê uma vista de olhos ao [exemplo de rotas-aninhadas](#).

:code-sandbox{src="csb_link_nuxt"}

O Componente NuxtLink

Para navegar entre as páginas da sua aplicação, você deve usar o componente `<NuxtLink>`. Este componente está incluído com Nuxt e assim você não tem de importar ele como você faz com outros componentes. É similar a tag `<a>` do HTML exceto que ao invés de usar o `href="/about"` você usa `to="/about"`. Se você já usou o `vue-router` antes, você pode pensar do `<NuxtLink>` como uma substituição do `<RouterLink>`.

Uma simples ligação para a página `index.vue` dentro da sua pasta `pages` :

```
<template>
  <NuxtLink to="/">Home page</NuxtLink>
</template>
```

O componente `<NuxtLink>` deve ser usado para todas ligações internas. Isso significa que todas as ligações para as páginas dentro do seu site você deve usar o `<NuxtLink>`. A tag `<a>` deve ser usada para todas ligações externas. Isso significa que se você tiver ligações para outros websites você deve usar a tag `<a>` para esses.

```
<template>
  <div>
    <h1>Home page</h1>
    <NuxtLink to="/about">
      >About (internal link that belongs to the Nuxt App)</NuxtLink>
    </NuxtLink>
  </div>
</template>
```

```
>
  <a href="https://nuxtjs.org">External Link to another page</a>
</div>
</template>
```

Info

Se você quiser saber mais sobre `<RouterLink>`, sinta-se livre para ler a [documentação do Vue Router](#) para obter mais informações.

Info

O `<NuxtLink>` também vem com a [pré-requisição inteligente](#) fora da caixa.

Pré-requisitar Ligações

O Nuxt automaticamente inclui pré-requisição inteligente. Isso significa que ele deteta quando uma ligação está visível, seja dentro da viewport ou quando estiver rolando na tela e pré-requisita o JavaScript para aquelas páginas assim que eles estiverem prontas quando o usuário clicar na ligação. O Nuxt apenas carrega os recursos quando o browser não estiver ocupado e pula a pré-requisição se a sua conexão estiver offline ou se você apenas tiver uma conexão 2G.

Desativa a Pré-requisição para Ligações Específicas

No entanto algumas vezes você pode querer desativar a pré-requisição em algumas ligações se sua página tiver muito JavaScript ou você tiver muitas páginas diferentes que seriam pré-requisitadas ou você tiver muitos scripts de terceiros que precisam ser carregados. Para desativar a pré-requisição em uma ligação específica, você pode usar a propriedade `no-prefetch`. Desde o Nuxt v2.10.0, você também pode usar a propriedade `prefetch` definida para `false`.

```
<NuxtLink to="/about" no-prefetch>About page not pre-fetched</NuxtLink>
<NuxtLink to="/about" :prefetch="false">About page not pre-fetched</NuxtLink>
```

Desativar a Pré-requisição Globalmente

Para desativar a pré-requisição em todas ligações, defina o `prefetchLinks` para `false`:

```
export default {
  router: {
    prefetchLinks: false
  }
}
```

Desde o Nuxt v2.10.0, se você tiver definido `prefetchLinks` para `false` mas você quiser pré-requisitar uma ligação específica, você pode usar a propriedade `prefetch`:

```
<NuxtLink to="/about" prefetch>About page pre-fetched</NuxtLink>
```

linkActiveClass (classe da ligação ativa)

A `linkActiveClass` funciona da mesma forma que a classe do `vue-router` para ligações ativas. Se quisermos mostrar quais ligações são ativadas, tudo que você tem de fazer é criar algum CSS para a classe `nuxt-link-active`.

```
.nuxt-link-active {
  color: red;
}
```

O CSS pode ser adicionado ao componente de navegação ou para uma página específica ou esquema ou dentro de seu ficheiro `main.css`.

Se você quiser, você pode também configurar o nome da classe para alguma coisa que você queira. Você pode fazer isso ao modificar a propriedade `linkActiveClass` dentro da propriedade `router` dentro do seu ficheiro `nuxt.config.js`.

```
export default {
  router: {
    linkActiveClass: 'my-custom-active-link'
  }
}
```

Info

Esta opção é dada diretamente ao `linkActiveClass` do `vue-router`. Veja a [documentação do vue-router](#) para obter mais informações.

linkExactActiveClass (classe da ligação exata ativa)

O `linkExactActiveClass` funciona da mesma forma que a classe do `vue-router` para ligações exatas ativas. Se quisermos mostrar quais ligações são ativas com um correspondente exato, todo que você tem de fazer é criar algum CSS para a classe `nuxt-link-exact-active`.

```
.nuxt-link-exact-active {
  color: green;
}
```

Info

O CSS pode ser adicionado ao componente de navegação ou para uma página específica ou esquema ou dentro de seu ficheiro `main.css`.

Se você quiser, você pode também configurar o nome da classe para alguma coisa que você queira. Você pode fazer isso ao modificar a propriedade `linkExactActiveClass` dentro da propriedade `router` dentro do seu ficheiro `nuxt.config.js`.

```
export default {
  router: {
    linkExactActiveClass: 'my-custom-exact-active-link'
  }
}
```

Info

Esta opção é dada diretamente ao linkExactActiveClass do `vue-router`. Veja a [documentação do vue-router](#) para obter mais informações.

linkPrefetchedClass (classe da ligação pré-requisitada)

O linkPrefetchedClass permitirá você adicionar estilos para todas ligações que têm de ser pré-requisitados. Isto é genial para testar quais ligações estão sendo pré-requisitadas depois de modificar o comportamento padrão. O linkPrefetchedClass está desativado por padrão. Se você quiser ativar ele você precisa adicionar ele à propriedade router dentro do seu ficheiro `nuxt.config.js`.

```
export default {
  router: {
    linkPrefetchedClass: 'nuxt-link-prefetched'
  }
}
```

Depois você pode adicionar estilos para aquela classe.

```
.nuxt-link-prefetched {
  color: orangeRed;
}
```

Info

Neste exemplo temos usado a classe `nuxt-link-prefetched` mas você pode nomear ele para o que achar melhor.

:code-sandbox{src="csb_link_nuxt_link"}

O Componente client-only

Este componente é usado para propositadamente renderizar um componente somente no lado do cliente. Para importar um componente somente no cliente, registe o componente dentro de um plugin somente no lado do cliente.

```
<template>
<div>
  <sidebar />
  <client-only placeholder="Loading...">
    <!-- este componente somente será renderizado no lado do cliente -->
    <comments />
```

```
</client-only>
</div>
</template>
```

Use um slot como um placeholder até `<client-only />` ser montado no lado do cliente.

```
<template>
  <div>
    <sidebar />
    <client-only>
      <!-- este componente somente será renderizado no lado do cliente -->
      <comments />

      <!-- indicador de carregamento, renderizado no lado do servidor -->
      <template #placeholder>
        <comments-placeholder />
      </template>
    </client-only>
  </div>
</template>
```

Info

Algumas vezes dentro das páginas renderizadas no servidor `$refs` dentro `<client-only>` pode não estar pronto mesmo com `$.nextTick`, o truque pode ser chamar o `$.nextTick` umas poucas vezes:

```
mounted() {
  this.initClientOnlyComp()
},
methods: {
  initClientOnlyComp(count = 10) {
    this.$nextTick(() => {
      if (this.$refs.myComp) {
        //...
      } else if (count > 0) {
        this.initClientOnlyComp(count - 1);
      }
    });
  },
}
```

`alert` Se você estiver usando a versão do Nuxt inferior a v2.9.0, use o `<no-ssr>` ao invés de `<client-only>`

Ativando a Descoberta Automática

Desde a versão `v2.13`, o Nuxt pode importar automaticamente seus componentes sempre que usados nos seus modelos (templates):

```
export default {
  components: true
}
```

Info

Saiba [como configurar a auto-descoberta de componente](#).

Usando Componentes

Depois de você criar os seus componentes dentro do diretório de componentes, eles estarão disponíveis em toda a sua aplicação sem a necessidade de importar eles.

```
| components/
--| TheHeader.vue
--| TheFooter.vue
```

```
<template>
  <div>
    <TheHeader />
    <Nuxt />
    <TheFooter />
  </div>
</template>
```

Info

Consulte a [demonstração ao vivo](#) ou [vídeo de exemplo](#).

Nomes do Componente

Se você tem componentes dentro de diretórios aninhados tais como:

```
| components/
--| base/
----| foo/
| Button.vue
```

O nome do componente será baseado no seu próprio diretório e nome de ficheiro. Então, o componente será:

```
<BaseFooButton />
```

Por questão de clareza, é recomendado que o nome do ficheiro do componente corresponde ao seu nome. (Assim, no exemplo acima, você poderia renomear `Button.vue` para ser `BaseFooButton.vue`.)

Se você quiser usar uma estrutura de diretório personalizada que não deve fazer parte do nome do componente, você pode explicitamente especificar esses diretórios:

```
| components/
--| base/
----| foo/
| Button.vue
```

```
components: {
  dirs: [
    '~/components',
    '~/components/base'
  ]
}
```

E agora dentro do seu modelo você pode usar `FooButton` ao invés de `BaseFooButton`.

```
<FooButton />
```

Info

Considere nomear o seu componentes e diretórios seguindo o [Guia de Estilo do Vue](#).

Importação Dinâmica

Para importar dinamicamente um componente (também conhecido como carregar preguiçosamente (lazy-loading) um componente) tudo que você precisa fazer é adicionar o prefixo `Lazy` para o componente do nome.

```
<template>
  <div>
    <TheHeader />
    <Nuxt />
    <LazyTheFooter />
  </div>
</template>
```

Isto é particularmente útil se o componente não é sempre necessário. Ao usar o prefixo `Lazy` você pode adiar o carregamento do código do componente até o momento correto, o que pode ser útil para otimizar o tamanho do seu pacote JavaScript.

```
<template>
  <div>
    <h1>Mountains</h1>
    <LazyMountainsList v-if="show" />
    <button v-if="!show" @click="show = true">Show List</button>
  </div>
</template>

<script>
export default {
  data() {
    return {
      show: false
    }
  }
}</script>
```

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

Cheatsheet

```
:modal{src="img" alt="imgAlt"}
```

[Go to TOC](#)

O diretório de construção do Nuxt

O diretório `.nuxt` é o chamado *diretório de construção (build)*. Ele é dinamicamente gerado e escondido por padrão. Dentro do diretório você pode encontrar ficheiros gerados automaticamente sempre que `nuxt dev` é usado ou seus artefactos de construção sempre que `nuxt build` é usado. A modificação desses ficheiros é uma grande forma de depuração mas lembre que eles são ficheiros gerados e uma vez que você executar o comando `dev` ou `build` novamente, qualquer coisa que foi guardada aqui será regenerada.

Warning

O diretório `.nuxt` não deve ser enviado para o seu sistema de controle de versão e deve ser ignorado através do seu `.gitignore` já que ele será gerado automaticamente quando você executar o comando `nuxt dev` ou `nuxt build`.

A propriedade buildDir

Por padrão, várias ferramentas assumem que `.nuxt` é um diretório oculto, pelo seu nome começar com um ponto. Você pode usar a opção `buildDir` para previr isso. Se você mudar o nome lembre de adicionar o novo nome dentro do seu ficheiro `.gitignore`.

```
export default {
  buildDir: 'nuxt-dist'
}
```

Dentro da pasta `.nuxt`:

- O ficheiro `router.js` é o ficheiro de roteador gerado que o Nuxt gera por você quando você coloca ficheiros `.vue` dentro da pasta `pages`. Você pode usar este ficheiro para fazer depuração para quando você quiser procurar quais rotas são geradas para o `vue-router` e encontrar os nomes de uma rota específica.
- O `route.scrollBehavior.js` o qual é o seu comportamento de rolagem do roteador (Router ScrollBehavior)
- A pasta de componentes tem todos os seus componentes do Nuxt tais como `NuxtChild` e `NuxtLink`. Ele também contém o `nuxt-build-indicator` o qual é a página que vemos quando sua aplicação está construindo e `nuxt-loading` o qual é seu componente de carregamento que é visto quando estamos esperando a sua página carregar. Você também encontrará a página `nuxt-error` aqui dentro a qual contém a página de erro padrão do Nuxt.
- A pasta de `mixins` tem os ficheiros necessários para o método `$fetch` do Nuxt.
- A pasta de `views` contém o modelo (template) da sua aplicação e sua página de erro do servidor.
- O ficheiro `app.js` é o ficheiro principal da sua aplicação.
- O ficheiro `client.js` é o ficheiro do cliente necessário para tudo que acontece no lado do cliente.
- O ficheiro `vazio` é intencionalmente deixado vazio para apelidos não operacionais
- O ficheiro `index.js` molda sua aplicação.
- O `loading.html` é o ficheiro que é usado quando a página está carregando.

- O ficheiro `middleware` é onde o seu intermediário é conservado
- O ficheiro `server.js` é todo código que é executado no servidor
- As `utilities` contém as utilidades que o Nuxt precisa para ele funcionar.

Desdobrando

A pasta `.nuxt` é parte dos ficheiros necessários para desdobrar a sua aplicação SSR. Ela não é necessário para desdobrar a sua aplicação Nuxt estática porque usamos a pasta `dist` para isso.

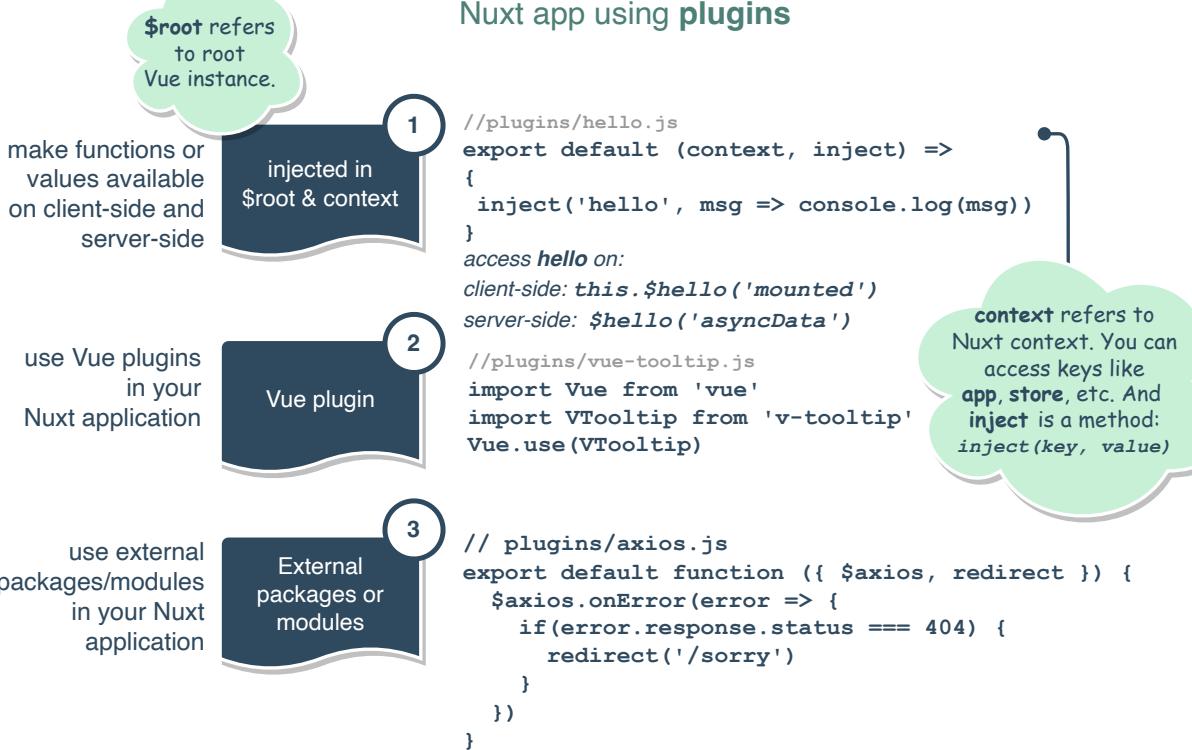
O diretório `plugins`

O diretório `plugins` contém seus plugins escritos em JavaScript que você deseja executar antes da instância da raiz da aplicação Vue.js.

Nuxt Plugins

Nuxt v2.14.1
August 2020

3 ways to extend functionalities of Nuxt app using plugins



Include Nuxt plugin using **plugins** property



Este é o lugar para adicionar os plugins do Vue e injetar funções ou constantes. Toda vez que você precisar usar o `Vue.use()`, você deve criar um ficheiro dentro de `plugins/` e adicionar seu caminho ao `plugins` dentro do `nuxt.config.js`.

Pacotes externos

Você pode querer usar pacotes/módulos externos dentro da sua aplicação (um grande exemplo é o `axios`) para fazer requisições HTTP para ambos servidor e cliente.

Primeiro, instale ele via npm ou yarn.

```
yarn add @nuxtjs/axios
```

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

Você pode configurar por exemplo os intercectores para reagir a possíveis erros da chamadas da sua API através da aplicação. Dentro deste exemplo nós redirecionamos o usuário para uma página de erro personalizada chamada desculpa (sorry) quando nós recebemos um estado de erro 500 por parte da nossa API.

```
export default function ({ $axios, redirect }) {
  $axios.onError(error => {
    if (error.response.status === 500) {
      redirect('/sorry')
    }
  })
}
```

Por último mas não menos importante, adicionar o módulo e o recentemente criado plugin à configuração do projeto.

```
module.exports = {
  modules: ['@nuxtjs/axios'],
  plugins: ['~/plugins/axios.js']
}
```

Depois podemos usar ele diretamente dentro do seu componentes de página:

```
<template>
  <h1>{{ post.title }}</h1>
</template>

<script>
export default {
  async asyncData ({ $axios, params }) {
    const post = await
    $axios.$get(`https://api.nuxtjs.dev/posts/${params.id}`)
    return { post }
  }
}
</script>
```

Uma outra maneira de usar o `axios` sem instalar o módulo é importando `axios` diretamente dentro da tag `<script>`.

```
<script>
import axios from 'axios'

export default {
  async asyncData ({ params }) {
    const { data: post } = await
    axios.get(`https://api.nuxtjs.dev/posts/${params.id}`)
    return { post }
  }
}</script>
```

Info

Se você receber um erro *Cannot use import statement outside a module* (*Você não pode usar a declaração de importação fora de um módulo*), você pode precisar adicionar o seu pacote à opção `build > transpile` dentro do `nuxt.config.js` para o carregador do webpack tornar o seu plugin disponível.

```
build: {
  // Você pode estender a configuração do webpack aqui
  transpile: ['npm-package-name'],
},
```

Plugins de Vue

Se você quiser usar plugins do vue, como `v-tooltip` para exibir dicas da ferramenta (tooltips) dentro da sua aplicação, nós precisamos configurar o plugin antes do lançamento da aplicação.

Primeiro, nós precisamos instalar ele

```
yarn add v-tooltip
```

```
npm install v-tooltip
```

Depois criamos o ficheiro `plugins/vue-tooltip.js`

```
import Vue from 'vue'
import VTooltip from 'v-tooltip'

Vue.use(VTooltip)
```

A propriedade `plugins`

Então nós adicionamos o caminho do ficheiro dentro a chave `plugins` do nosso `nuxt.config.js`. A propriedade `plugins` permite você adicionar plugins do Vue.js à sua aplicação principal. Todos os caminhos definidos dentro da propriedade `plugins` serão importados antes da inicialização da aplicação principal.

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

Plugins do ES6

Se o plugin está localizado dentro do `node_modules` e exporta um módulo ES6, você pode precisar adicionar ele à opção de construção `transpile`:

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

Você pode recorrer a documentação da [configuração do build](#) para saber mais sobre a opção `build`.

Apenas no lado do cliente ou servidor

Alguns plugins podem funcionar apenas dentro do browser por causa da falta suporte ao SSR.

Nomear de maneira convencional o plugin

Se é suposto um plugin ser executado apenas no lado do cliente ou servidor, `.client.js` ou `.server.js` podem ser aplicados como uma extensão do ficheiro do plugin. O ficheiro será automaticamente incluído apenas no respetivo lado (do cliente ou servidor).

```
export default {
  plugins: [
    '~/plugins/foo.client.js', // apenas no lado do cliente
    '~/plugins/bar.server.js', // apenas no lado do servidor
    '~/plugins/baz.js' // ambos cliente e servidor
  ]
}
```

A sintaxe de objeto

Você pode também usar a sintaxe de objeto com a propriedade `mode` com o valor (`'client'` ou `'server'`) dentro de `plugins`.

```
export default {
  plugins: [
    { src: '~/plugins/both-sides.js' },
    { src: '~/plugins/client-only.js', mode: 'client' }, // apenas no lado do cliente
    { src: '~/plugins/server-only.js', mode: 'server' } // apenas no lado do servidor
  ]
}
```

Injetar dentro do `$root` e contexto

Algumas vezes você deseja tornar as funções ou valores disponíveis ao longo da sua aplicação. Você pode injetar essas variáveis dentro das instâncias do Vue (lado do cliente), o contexto (lado do servidor) e até dentro da memória do Vuex. É uma convenção prefixar essas funções com um `$`.

O Nuxt fornece a você uma maneira de fazer isso facilmente com um método `inject(key, value)`. O `inject` é dado como segundo parâmetro quando exportamos uma função. O `$` será pré-adicionado automaticamente a chave.

Info

É importante saber que de todo [ciclo de vida da instância do Vue](#), apenas os gatilhos `beforeCreate` e `created` são chamados ambos, no lado do cliente e no lado do servidor. Todos os outros gatilhos são apenas chamado no lado do cliente.

```
export default ({ app }, inject) => {
  // Injete $hello(msg) dentro do Vue, contexto e memória.
  inject('hello', msg => console.log(`Hello ${msg}!`))
}

export default {
  plugins: ['~/plugins/hello.js']
}
```

Agora o serviço `$hello` pode ser acessado a partir do `context` e `this` dentro das páginas, componentes, plugins, e ações da memória.

```
export default {
  mounted() {
    this.$hello('mounted')
    // irá exibir no terminal 'Hello mounted!'
  },
  asyncData({ app, $hello }) {
    $hello('asyncData')
    // Se estiver usando Nuxt <= 2.12, use ⚡
    app.$hello('asyncData')
  }
}
```

```
export const state = () => ({
  someValue: ''
})

export const actions = {
  setSomeValueToWhatever({ commit }) {
    this.$hello('store action')
    const newValue = 'whatever'
    commit('changeSomeValue', newValue)
  }
}
```

Warning

Não use o `Vue.use()`, `Vue.component()` e globalmente não conecte nada ao Vue **dentro** desta função, dedicada a injeção do Nuxt. Isso causará vazamento de memória no lado do servidor.

A propriedade extendPlugins

Você talvez queira estender os plugins ou mudar as ordem dos plugins criada pelo Nuxt. Esta função aceita um array de objetos de [plugin](#) e deve retornar um array de objetos de plugin.

Exemplo de mudança da ordem de plugins:

```
export default {
  extendPlugins(plugins) {
    const pluginIndex = plugins.findIndex(
      ({ src }) => src === '~/plugins/shouldBeFirst.js'
    )
    const shouldBeFirstPlugin = plugins[pluginIndex]

    plugins.splice(pluginIndex, 1)
    plugins.unshift(shouldBeFirstPlugin)

    return plugins
  }
}
```

Mixins global

Mixins global pode ser facilmente adicionado com plugins do Nuxt mas pode causar problemas e vazamento de memória quando não manipulado corretamente. Sempre que você adicionar um mixin global a sua aplicação, você deve usar um bandeira para evitar registrar ele várias vezes:

```
import Vue from "vue"

// Certifique-se de pegar um nome único para a bandeira
// assim ele não irá entrar em conflito com qualquer outro mixin.
if (!Vue.__my_mixin__) {
  Vue.__my_mixin__ = true
  Vue.mixin({ ... }) // Depois configure o seu mixin
}
```

O diretório static

O diretório `static` é diretamente mapeado para a raiz do servidor () e contém ficheiros que raramente se- rão mudados. Todos ficheiros incluídos serão automaticamente servidos pelo Nuxt e estão acessíveis através da URL raiz do seu projeto.

`/static/robots.txt` estará disponível em `http://localhost:3000/robots.txt`

`/static/favicon.ico` estará disponível em `http://localhost:3000/favicon.ico`

Esta opção é útil para ficheiros como `robots.txt`, `sitemap.xml` ou `CNAME` (o qual é importante para o processo de deploy no GitHub Pages).

Warning

Este diretório não pode ser renomeado sem configuração extra.

Os recursos estáticos

Se você não quiser usar os recursos do Webpack do diretório `assets`, você pode adicionar imagens ao di- retório static.

Dentro do seu código, você pode então referenciar esses ficheiros relativamente a raiz (`/`):

```
<!-- Imagem estática do diretório static -->


<!-- imagem empacotada pelo webpack do diretório assets -->
<img src "~/assets/my-image-2.png" />
```

Info

O Nuxt não muda este caminho, assim se você personalizar a sua `router.base` então você precisará fazer questão de adicionar ele manualmente aos seus caminhos. Por exemplo:

```

```

A configuração do diretório static

Se você precisar você pode configurar o comportamento do diretório `static/` dentro do ficheiro `nuxt.- config.js`.

O prefixo do recurso estático

Se você desdobrar o Nuxt para uma sub-pasta, exemplo `/blog/`, a base do router será adicionada ao ca- minho do recurso estático por padrão. Se você quiser desativar este comportamento, você pode definir `static.prefix` para `false` dentro do `nuxt.config.js`.

```
export default {  
  static: {  
    prefix: false  
  }  
}
```

Padrão: /blog/my-image.png

Com o static.prefix desativado: /my-image.png

O diretório store

O diretório `store` contém seus ficheiros de memória do Vuex. A memória do Vuex vem fora da caixa com o Nuxt mas está desativada por padrão. A criação de um ficheiro `index.js` dentro deste diretório ativa a memória.

Warning

Este diretório não pode ser renomeado sem configuração extra.

Usar uma memória para gerir o estado é importante para toda grande aplicação. Isto é o porquê do Nuxt implementar o Vuex dentro do seu núcleo.

Ativar a memória

O Nuxt irá procurar pelo diretório `store`. Se ele conter um ficheiro, que não seja um ficheiro oculto ou um ficheiro `README.md`, então a memória será ativada. Isto significa que o Nuxt irá:

1. Importar o Vuex
2. Adicionar a opção `store` à instância raíz do Vue.

Os módulos

Todo ficheiro `.js` dentro do diretório `store` é transformado em um [módulo com nome reservado](#)(com o `index` sendo a raiz do módulo). O seu valor de `state` sempre será uma `function` para evitar estado *partilhado* indesejado no lado do servidor.

Para começar, exporte o estado (`state`) como uma função, e as mutações (`mutations`) e ações (`actions`) como objetos.

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

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

Depois, você pode ter um ficheiro `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
}
}
}

```

A memória será criada tal e qual:

```

new Vuex.Store({
  state: () => ({
    counter: 0
}),
  mutations: {
    increment(state) {
      state.counter++
    }
},
  modules: {
    todos: {
      namespaced: true,
      state: () => ({
        list: []
      }),
      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
        }
      }
    }
  }
})
}

```

E dentro do seu `pages/todos.vue`, ao usar o módulo `todos`:

```

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

<script>

```

```

import { 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>

```

O método do módulo também funciona para definições de alto-nível sem a implementação de um sub-diretório dentro do diretório store.

Exemplo para estado (state): você cria um ficheiro `store/state.js` e adiciona o seguinte.

```

export default () => ({
  counter: 0
})

```

E os recuperadores (getters) podem estar no ficheiro `store/getter.js`:

```

export default {
  getCounter(state) {
    return state.counter
  }
}

```

E as mutações (mutations) correspondentes podem estar no ficheiro `store/mutations.js`

```

export default {
  increment(state) {
    state.counter++
  }
}

```

E as ações (actions) correspondentes podem estar no ficheiro `store/actions.js`:

```

export default {
  async fetchCounter({ state }) {
    // fazer requisição
    const response = { data: 10 };
    state.counter = response.data;
  }
}

```

```

        return response.data;
    }
}

```

Exemplo de estrutura de pasta

Uma configuração complexa da estrutura de ficheiro/pasta memória pode parecer com isto:

```

store/
--| index.js
--| ui.js
--| shop/
----| cart/
| actions.js
| getters.js
| mutations.js
| state.js
----| products/
| mutations.js
| state.js
| itemsGroup1/
| state.js

```

Os plugins na memória

Você pode adicionar plugins adicionais à memória ao por eles dentro do ficheiro `store/index.js`:

```

import myPlugin from 'myPlugin'

export const plugins = [myPlugin]

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

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

```

Mais informações sobre os plugins na: [documentação do Vuex](#).

A ação nuxtServerInit

Se a ação `nuxtServerInit` estiver definida dentro da memória (store) e o modo estiver em `universal`, o Nuxt irá chamar ele com o contexto (apenas a partir do lado do servidor). É útil quando nós temos algum dado no servidor que nós queremos dar diretamente para o lado do cliente.

Por exemplo, vamos dizer que temos sessões no lado do servidor e podemos acessar o usuário conectado através do `req.session.user`. Para adicionar o usuário autenticado a nossa memória (store), nós atualizamos o nosso `store/index.js` para o seguinte:

```
actions: {
  nuxtServerInit ({ commit }, { req }) {
    if (req.session.user) {
      commit('user', req.session.user)
    }
  }
}
```

Warning

Somente o módulo primário (dentro do `store/index.js`) será receber esta ação. Você precisará encadear as ações do módulo a partir de lá.

O `contexto` é dado ao `nuxtServerInit` como o segundo argumento dentro do método `asyncData`.

Se o `nuxt generate` for executado, o `nuxtServerInit` será executado para cada rota dinâmica gerada.

Info

Ações do `nuxtServerInit` assíncronas devem retornar uma promessa (Promise) ou usar o `async/await` para permitir o servidor do nuxt esperar neles.

```
actions: {
  async nuxtServerInit({ dispatch }) {
    await dispatch('core/load')
  }
}
```

O modo estrito do Vuex

O modo estrito está ativado por padrão no modo de desenvolvimento e desativado no modo de produção. Para desativar o modo estrito em desenvolvimento, siga o exemplo abaixo dentro do `store/index.js`:

```
export const strict = false
```

O ficheiro de configuração do Nuxt

Por padrão, o Nuxt está configurado para cobrir a maior parte dos casos de uso. Esta configuração pode ser sobreescrita com o ficheiro `nuxt.config.js`.

nuxt.config.js

A propriedade alias

Esta opção permite você definir apelidos que estarão disponíveis dentro do seu JavaScript e CSS.

```
import { resolve } from 'path'

export default {
  alias: {
    'style': resolve(__dirname, './assets/style')
  }
}
```

Next

Para saber mais consulte pela [propriedade alias](#)

A propriedade build

Esta opção permite você configurar várias definições para o passo `build`, incluindo `loaders`, `filenames`, a configuração do `webpack` e `transpilation`.

```
export default {
  build: {
    /*
      ** Você pode estender a configuração do webpack aqui
    */
    extend(config, ctx) {}
  }
}
```

Next

Para saber mais consulte pela [propriedade build](#)

A propriedade css

Esta opção permite você definir os ficheiros CSS, módulos e bibliotecas que você quiser incluir globalmente (em todas páginas).

```
export default {
  css: ['~/assets/css/main.css', '~/assets/css/animations.scss']
}
```

Você pode omitir a extensão para CSS, SCSS, PostCSS, LESS, Stylus, ... ficheiros listados dentro o array `css` dentro do seu ficheiro de configuração do nuxt.

```
export default {
  css: ['~/assets/css/main', '~/assets/css/animations']
}
```

Ao omitir a extensão, se você tiver um ficheiro `css` e decidir mudar para usar `sass` por exemplo, você não terá de atualizar o seu `nuxt.config.js` assim ele usará a nova extensão uma vez que nome do ficheiro permanece o mesmo.

Next

Para saber mais consulte pela [propriedade css](#)

A propriedade dev

Esta opção permite você definir o modo de `development` (desenvolvimento) ou `production` (produção) do Nuxt (importante para quando você usar o Nuxt programaticamente).

```
export default {
  dev: process.env.NODE_ENV !== 'production'
}
```

Next

Para saber mais consulte pela [propriedade dev](#)

A propriedade env

Esta opção permite você definir variáveis de ambientes que são requeridas no momento da construção (ao invés do momento de execução) tais como `NODE_ENV=staging` ou `VERSION=1.2.3`. Contudo, para as variáveis do tempo de execução o `runtimeConfig` é requerido.

```
export default {
  env: {
    baseURL: process.env.BASE_URL
  }
}
```

A propriedade runtimeConfig

A configuração do tempo de execução tem suporte embutido ao `dotenv` para uma segurança melhor e rápido desenvolvimento. A configuração do tempo de execução é adicionada ao payload do Nuxt assim não há necessidade de reconstruir no sentido de atualizar a configuração do tempo de execução quando estiver trabalhando em desenvolvimento ou com a renderização no lado do servidor ou com apenas aplicações no lado do cliente. (Para sites estáticos você continuará a precisar regerar o seu site para ver as mudanças).

Suporte ao ponto env

Se você tiver um ficheiro `.env` dentro do diretório raiz do seu projeto, ele será automaticamente carregado dentro do `process.env` e acessível dentro do seu `nuxt.config / serverMiddleware` e quaisquer outros ficheiros que eles importar.

Você pode personalizar o caminho ao usar `--dotenv <file>` ou desativar completamente com `--dotenv false`. Por exemplo, você pode especificar um ficheiro diferente `.env` em ambientes de produção, montagem ou desenvolvimento.

A propriedade publicRuntimeConfig

- deve carregar todas variáveis de ambientes que são públicas assim esses serão expostas no frontend.
Isto pode incluir uma referência para sua URL pública por exemplo.
- está disponível ao usar `$config` dentro de ambos servidor e cliente.

```
export default {
  publicRuntimeConfig: {
    baseURL: process.env.BASE_URL || 'https://nuxtjs.org'
  }
}
```

A propriedade privateRuntimeConfig

- deve carregar todas variáveis de ambiente que são privadas e que não devem ser expostas no frontend.
Isto pode incluir uma referência para os seus símbolos secretos da API por exemplo.
- apenas está disponível no servidor ao usar o mesmo `$config` (ele sobrescreve o `publicRuntimeConfig`)

```
export default {
  privateRuntimeConfig: {
    apiSecret: process.env.API_SECRET
  }
}
```

Usando seus valores de configuração:

Você pode então acessar esses valores em qualquer lugar ao usar o contexto dentro de suas páginas, memória, componentes e plugins ao usar `this.$config` ou `context.$config`.

```
<script>
  asyncData ({ $config: { baseURL } }) {
    const posts = await fetch(`#${baseURL}/posts`)
      .then(res => res.json())
  }
</script>
```

Dentro dos seus modelos você pode acessar suas configurações de tempo de execução diretamente usando `$config.*`

```
<template>
  <p>Our Url is: {{ $config.baseURL }}</p>
</template>
```

Warning

A sua configuração privada pode ser exposta se você usar `$config` fora de um contexto de apenas servidor (por exemplo, se você usar `$config` dentro do `fetch`, `asyncData` ou diretamente dentro do seu modelo).

Next

Para saber mais consulte por [runtimeConfig](#)

Next

Consulte a publicação do nosso blogue em [Migrando de @nuxtjs/dotenv para configuração do tempo de execução](#)

Next

Para saber mais consulte pela [propriedade env](#)

A propriedade generate

Esta opção permite você definir valores de parâmetros para cada rota dinâmica dentro da sua aplicação que será transformada em ficheiros HTML pelo Nuxt.

```
export default {
  generate: {
    dir: 'gh_pages', // gh_pages/ ao invés de dist/
    subFolders: false // Ficheiros HTML são gerados de acordo com caminho da rota
  }
}
```

Next

Para saber mais consulte pela [propriedade generate](#)

head

```
export default {
  head: {
    title: 'my title',
    meta: [
      { charset: 'utf-8' },
      ....
    ]
  }
}
```

Esta opção permite você definir todas as metas tags padrão para sua aplicação.

Next

Para saber mais consulte pela [integração do head](#)

A propriedade loading

Esta opção permite você personalizar o componente de carregamento que o Nuxt usa por padrão.

```
export default {
  loading: {
    color: '#fff'
  }
}
```

Next

Para saber mais consulte pela [integração do loading](#)

A propriedade modules

Com esta opção você pode adicionar módulos de Nuxt ao seu projeto.

```
export default {
  modules: ['@nuxtjs/axios']
}
```

Next

Para saber mais consulte pela [propriedade modules](#)

A propriedade modulesDir

A propriedade modelesDir é usada para definir os diretórios dos módulos para resolução do caminho. Por exemplo: resolveLoading, nodeExternals e postcss do webpack. O caminho de configuração é relativo ao `options.rootDir` (padrão: `process.cwd()`).

```
export default {
  modulesDir: ['../../node_modules']
}
```

Configurar este campo pode ser necessário se seu projeto está organizado como um espaço de trabalho de mono-repositório com Yarn.

Next

Para saber mais consulte pela [propriedade modulesDir](#)

A propriedade plugins

Esta opção permite você definir plugins escritos em JavaScript que devem ser executados antes da inicialização da raiz da aplicação Vue.js.

```
export default {
  plugins: ['~/plugins/url-helpers.js']
}
```

Next

Para saber mais consulte pela [propriedade plugins](#)

A propriedade router

Com a opção `router` você pode sobrescrever a configuração padrão do Nuxt do Vue Router.

```
export default {
  router: {
    linkExactActiveClass: 'text-primary'
  }
}
```

Next

Para saber mais consulte pela [propriedade router](#)

A propriedade server

Esta opção permite você configurar as variáveis de conexão para a instância da sua aplicação Nuxt.

```
import path from 'path'
import fs from 'fs'

export default {
  server: {
    https: {
      key: fs.readFileSync(path.resolve(__dirname, 'server.key')),
      cert: fs.readFileSync(path.resolve(__dirname, 'server.crt'))
    }
  }
}
```

Next

Para saber mais consulte pela [propriedade server](#)

A propriedade srcDir

Esta opção permite você definir o diretório fonte da sua aplicação Nuxt.

```
export default {
  srcDir: 'client/'
}
```

Exemplo da estrutura do projeto com a sua aplicação Nuxt dentro do diretório `client`.

```
**- app/
--- node_modules/
--- nuxt.config.js
--- package.json
---- client/
| assets/
| components/
| layouts/
```

```
middleware/
pages/
plugins/
static/
store/**
```

A propriedade dir

Esta opção permite você definir nomes personalizados dos seus diretórios do Nuxt.

```
export default {
  dir: {
    pages: 'views' // O Nuxt buscará pela pasta views/ ao invés da pasta pages/
  }
}
```

Next

Para saber mais consulte pela [propriedade dir](#)

A propriedade pageTransition

Esta opção permite você definir as propriedades de transições da página.

```
export default {
  pageTransition: 'page'
}
```

Next

Para saber mais consulte pela [propriedade transition](#)

Outros ficheiros de configuração

Além do `nuxt.config.js` podem haver outros ficheiros de configuração dentro da raíz do seu projeto, tais como o `.eslintrc`, `prettier.config.json` ou `.gitignore`. Estes são usados para configurar outras ferramentas tais como o seu linter, formatador de código ou o seu repositório do git e independente do `nuxt.config.js`.

O ficheiro .gitignore

Dentro do seu ficheiro `.gitignore` você precisará adicionar os seguintes diretórios, assim eles serão ignorados e não adicionados ao controlo de versão. `node_modules` o qual é onde todos os seus módulos instalados estão. A pasta `.nuxt` a que é criada quando estiver executando os comandos `dev` ou `build`. A pasta `dist` é a pasta que é criada quando estiver executando o comando `generate`.

```
node_modules .nuxt dist
```

O que segue

Next

Para saber mais consulte o [glossário da configuração](#)

O diretório assets

O diretório `assets` contém recursos não compilados tais como ficheiros Stylus ou Sass, imagens, ou fontes.

As imagens

Dentro dos seus modelos `vue`, se você precisar ligar ao seu diretório `assets`, use `~/assets/your_image.png` com uma barra antes do assets.

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

Dentro dos seus ficheiros `css`, se você precisar referenciar o seu diretório `assets`, use `~assets/your_image.png` (sem uma barra).

```
background: url('~assets/banner.svg');
```

Quando estiveres trabalhando com imagens dinâmicas você precisará usar a função `require()`

```

```

Next

Aprenda mais sobre [os recursos do webpack](#)

Os estilos

O Nuxt permite você definir ficheiros/módulos/bibliotecas CSS que você quiser definir globalmente (incluído dentro de cada página). Dentro do ficheiro `nuxt.config.js` você pode facilmente adicionar seus estilos usando a propriedade CSS.

```
export default {
  css: [
    // Carrega um módulo do Node.js diretamente (aqui está um ficheiro Sass)
    'bulma',
    // Ficheiro CSS dentro do projeto
    '~/assets/css/main.css',
    // Ficheiro SCSS dentro do projeto
    '~/assets/css/main.scss'
  ]
}
```

O Sass

No caso de você querer usar `sass` certifique-se que você tem os pacotes `sass` e `sass-loader` instalado.

```
yarn add --dev sass sass-loader@10
```

```
npm install --save-dev sass sass-loader@10
```

O Nuxt irá automaticamente adivinhar o tipo de ficheiro pela sua extensão e usar o carregador do pré-processador adequado para o webpack. Você continuará a precisar instalar o carregador exigido se você precisar usar eles.

As fontes

Você pode usar fontes locais ao adicionar elas à sua pasta `assets`. Uma vez que eles têm sido adicionados você pode então acessar eles através do seu CSS usando o `@font-face`.

```
-l assets
----l fonts
| DMSans-Regular.ttf
| DMSans-Bold.ttf
```

```
@font-face {
  font-family: 'DM Sans';
  font-style: normal;
  font-weight: 400;
  font-display: swap;
  src: url('~assets/fonts/DMSans-Regular.ttf') format('truetype');
}

@font-face {
  font-family: 'DM Sans';
  font-style: normal;
  font-weight: 700;
  font-display: swap;
  src: url('~assets/fonts/DMSans-Bold.ttf') format('truetype');
}
```

Info

Os ficheiros CSS não são automaticamente carregados. Adicione eles usando a [propriedade de configuração do CSS](#).

Next

Para adicionar fontes externas tais como as do Google Fonts consulte o [capítulo Meta Tags e SEO](#)

Os recursos do webpack

Por padrão, o Nuxt usa o `vue-loader` do webpack, `file-loader`, e o `url-loader` para servir seus assets. Você também pode usar o diretório `static` para os recursos que não devem ser processados pelo webpack

Webpack

O `vue-loader` processa automaticamente seu estilo e ficheiros de modelos com o `css-loader` e o compilador de modelos do Vue. Neste processo de compilação, todas URLs do recurso tais como ``, `background: url(...)`, e os `@import` do CSS são resolvidos como módulo de dependências.

Por exemplo, nós temos esta árvore de ficheiro:

```
-l assets/
  ---l image.png
-l pages/
  ---l index.vue
```

Se você usar `url('~assets/image.png')` dentro do seu CSS, ele será traduzido para `require('~assets/image.png')`.

Warning

O apelido `~/` não será resolvido corretamente dentro de ficheiros CSS. Você deve usar `~assets` (**sem uma barra**) dentro das referências de `url` de CSS, exemplo `background: url("~assets/banner.svg")`

Se você referenciar aquela imagem dentro do seu `pages/index.vue`:

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

Ela será compilada para:

```
createElement('img', { attrs: { src: require('~assets/image.png') } })
```

Por `.png` não ser um ficheiro JavaScript, o Nuxt configura o webpack para usar `file-loader` e `url-loader` para manipular eles por você.

Os benefícios desses carregadores são:

O `file-loader` permite você designar onde copiar e colocar o ficheiro de recurso, e como nomear ele usando a versão com hash para um cacheamento melhor. Em produção, você tirará proveito de cacheamento de longo período por padrão!

O `url-loader` permite você condicionalmente embutir os ficheiros como URLs de dados em base64 se eles forem menores do que um determinado limite. Isto pode reduzir o número de requisições HTTP para ficheiros triviais.

Para estes dois carregadores, a configuração padrão é:

```
// https://github.com/nuxt/nuxt.js/blob/dev/packages/webpack/src/config/base.js#L382-L411
{
  test: /\.(png|jpe?g|gif|svg|webp|avif)$/i,
```

```

use: [{  
  loader: 'url-loader',  
  options: {  
    esModule: false,  
    limit: 1000, // 1kB  
    name: 'img/[name].[contenthash:7].[ext]'  
  }]  
},  
{  
  test: /\.(woff2?|eot|ttf|otf)(\?.*)?$/i,  
  use: [{  
    loader: 'url-loader',  
    options: {  
      esModule: false,  
      limit: 1000, // 1kB  
      name: 'fonts/[name].[contenthash:7].[ext]'  
    }]  
},  
{  
  test: /\.(webm|mp4|ogv)$/i,  
  use: [{  
    loader: 'file-loader',  
    options: {  
      esModule: false,  
      name: 'videos/[name].[contenthash:7].[ext]'  
    }]  
}
]

```

O que significa que cada ficheiro abaixo de 1kb será embutido como URL de dado na base64. Caso contrário, a imagem/fonte serão copiados para suas pastas correspondentes (dentro do diretório `.nuxt`) com o nome contendo a versão em hash para um cacheamento melhor.

Quando estiver lançando a sua aplicação com `nuxt`, o seu modelo dentro de `pages/index.vue`:

```

<template>  
    
</template>

```

Será transformado em:

```



```

Se você quiser mudar as configurações do carregador, use o `build.extend`.

Os apelidos

Por padrão o diretório fonte (`srcDir`) e diretório raiz (`rootDir`) são o mesmo. Você pode usar o apelido `~` para o diretório fonte. Ao invés de escrever caminhos relativos como `../assets/your_image.png` você pode usar `~/assets/your_image.png`.

Ambos alcançarão os mesmos resultados.

```
<template>
  <div>
    
    <img src("~/assets/your_image.png" />
  </div>
</template>
```

Nós recomendamos usar o `~` como um apelido. `@` continua sendo suportado mas não funcionará em todos casos tais como com imagens de fundo dentro do seu css.

Você pode usar os apelidos `~~` ou `@@` para o diretório raiz.

Info

Dica: No teclado Espanhol você pode acessar o `~` com (`Option` + `ñ`) no Mac OS, ou (`Alt Gr` + `4`) no Windows

[Go to TOC](#)

O diretório components

O diretório `components` contém seus componentes Vue.js. Os componentes são os que compõe as diferentes partes da sua página e podem ser reutilizados e importados dentro das suas páginas, dos esquemas e até dentro de outros componentes.

Requisição de dados

Para acessar dados assíncronos a partir de uma API dentro dos seus componentes você pode usar o método `fetch()` do Nuxt.

Ao verificar o `$fetchState.pending`, nós podemos exibir uma mensagem quando os dados estiverem esperando para serem carregados. Nós podemos também consultar o `$fetchState.error` e exibir uma mensagem de erro se houver um erro na requisição dos dados. Sempre que estivermos usando o método `fetch()`, nós devemos declarar as propriedades apropriadas dentro do método `data()`. Os dados que vierem da requisição podem então ser atribuídos à estas propriedades.

```
<template>
  <div>
    <p v-if="$fetchState.pending">Loading....</p>
    <p v-else-if="$fetchState.error">Error while fetching mountains</p>
    <ul v-else>
      <li v-for="(mountain, index) in mountains" :key="index">
        {{ mountain.title }}
      </li>
    </ul>
  </div>
</template>
<script>
  export default {
    data() {
      return {
        mountains: []
      }
    },
    async fetch() {
      this.mountains = await fetch(
        'https://api.nuxtjs.dev/mountains'
      ).then(res => res.json())
    }
  }
</script>
```

Next

Consulte o capítulo no método `fetch()` para obter mais detalhes sobre como o fetch funciona.

Descoberta de componentes

:prose-img{src="../../../../static/img/docs/components.png"}

A partir da versão `v2.13`, o Nuxt pode importar automaticamente os componentes que você usar. Para ativar esta funcionalidade, defina `components: true` dentro da sua configuração:

```
export default {
  components: true
}
```

Qualquer componente dentro do diretório `~/components` podem então ser usados em todas as suas páginas, esquemas (e outros componentes) sem a necessidade de explicitamente importar eles.

```
| components/
--| TheHeader.vue
--| TheFooter.vue
```

```
<template>
  <div>
    <TheHeader />
    <Nuxt />
    <TheFooter />
  </div>
</template>
```

Next

Aprenda mais sobre o módulo de componentes [dentro da documentação da descoberta de componentes](#) e [dentro deste artigo de anúncio](#).

Importação dinâmica

Para importar dinamicamente um componente, também conhecido como carregamento preguiçoso de um componente, tudo o que você precisa fazer é adicionar o prefixo `Lazy` dentro dos seus modelos.

```
<template>
  <div>
    <TheHeader />
    <Nuxt />
    <LazyTheFooter />
  </div>
</template>
```

Ao usar o prefixo `lazy` você pode também dinamicamente importar um componente quando um evento for acionado.

```
<template>
  <div>
    <h1>Mountains</h1>
    <LazyMountainsList v-if="show" />
    <button v-if="!show" @click="show = true">Show List</button>
  </div>
</template>

<script>
export default {
  data() {
    return {
      show: false
    }
  }
}
```

```

        }
    }
</script>

```

Diretórios aninhados

Se você tiver componentes dentro de diretórios aninhados tais como:

```

components/
base/
foo/
CustomButton.vue

```

O nome do componente será baseado no seu próprio caminho do diretório e nome de ficheiro. Portanto, o componente será:

```
<BaseFooCustomButton />
```

Se nós quisermos usar ele como `<CustomButton />` enquanto mantém a estrutura do diretório, nós podemos adicionar o diretório de `CustomButton.vue` dentro de `nuxt.config.js`.

```

components: {
  dirs: [
    '~/components',
    '~/components/base/foo'
  ]
}

```

E agora podemos usar o `<CustomButton />` ao invés de `<BaseFooCustomButton />`.

```
<CustomButton />
```

Next

Consulte a documentação da [propriedade components](#) para conhecer outros métodos de controle de nome do componente.

Info

Saiba mais sobre o [módulo de componentes](#).

O diretório content

Dê poderes sua aplicação Nuxt com o módulo `@nuxt/content` onde você pode escrever dentro de um diretório `content/` e requisitar os seus ficheiros markdown, json, yaml e csv através de uma API parecida com o MongoDB, agindo como um **CMS Headless baseado em Git**.

Nuxt Content Module

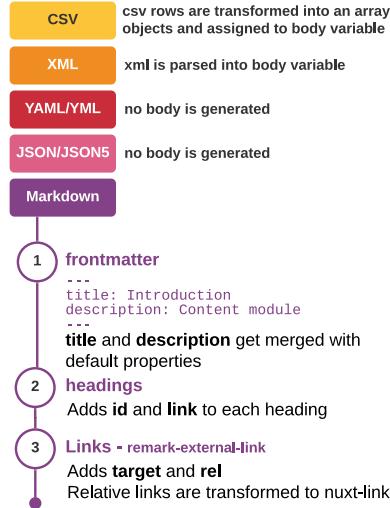
1. WRITE

Getting started:

- Install module `npm install @nuxt/content`
- Create `content/` directory in your Nuxt project.

Module will parse `.md`, `.yaml`, `.yml`, `.csv`, `.json`, `.json5`, `.xml` and generate following default properties:

- dir - extension (`.md`)
- path - `createdAt`
- slug - `updatedAt`



1. Write continues...

- Footnotes - remark-footnote**: Add footnotes like, `[^1]`, then define that footnote like, `[^1]: This is my first footnote`
- Codeblocks**: Add codeblocks
- Syntax highlight - Prism Js**: data line, file name
`...js{1, 3-5}[server.js]`
- HTML**: Add html inside markdown file
- Vue Components**: Add Vue component inside markdown file
✓ `<my-component>` - only kebab case
✗ `<my-component/>` - no self-closing
- Global components**: Put components in `components/global/` and access them directly inside markdown file
- TOC**: - id of titles create links
- h2 & h3 are used to create toc

2. FETCH

Fetch content on:

- Client-side using `this.$content`
- Server-side using `context.$content`

```
content(path, options).fetch()
```

Example:

```
$content('articles', params.slug).fetch()
== /articles/${params.slug}
```

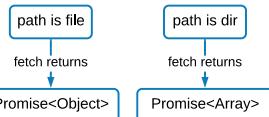
Chainable methods:

- only(keys)**: Select a subset of fields
 - without(keys)**: Remove a subset of fields
 - where(query)**: Filter results by query
 - sortBy(key, direction)**: Sort results by key
 - limit(n)**: Limit number of results
 - skip(n)**: Skip results
 - search(field, value)**: Perform a full-text search on a field
- Default fields:
['title', 'description', 'slug', 'text']

Nuxt Content Module

2. Fetch continues...

- surround(slug, options)**: Get prev and next results around a specific slug
search, limit and skip are ineffective when using this method
- fetch()**: Ends chain sequence and collects data



Example:

```
const articles =
  const articles = await this.$content('articles')
    .only(['title', 'date', 'authors'])
    .sortBy('date', 'asc')
    .limit(5)
    .skip(10)
    .where({tags: 'testing',
      isArchived: false,
      date: { $gt: new Date(2020) },
      rating: { $gte: 3 }
    })
    .search('welcome')
    .fetch()
```

Example with options:

```
Fetch files from subdirectories
const options = {deep:true, text:true}
const options = {deep:true, text:true}
await this.$content('articles', params.slug, options)
```

Fetch files from subdirectories

Returns markdown content in a text variable

3. DISPLAY

Use `<nuxt-content>` component directly in template area to display the page body. No import required.

```
<nuxt-content :document="page" />
```

`<nuxt-content>` accepts page object using :document prop.

Live editing (>= v1.4.0): In development, double-click on the `<nuxt-content>` component to live edit the content directly in the browser!

Custom style: `<nuxt-content>` component adds `.nuxt-content` class. `.nuxt-content` class can be used to add custom styles for the content.

```
.nuxt-content h1 {
  // add custom style for h1 here
}
```

Example:

```
<template>
  <article>
    <h1>{{ page.title }}</h1>
    <nuxt-content :document="page" />
  </article>
</template>
<script>
export default {
  async asyncData ({ $content }) {
    const page = await $content('home').fetch()

    return {
      page
    }
  }
}</script>
```

Default configuration

Configure `@nuxt/content` with the `content` property in `nuxt.config.js`.

- Customise api url using `apiPrefix`. By default, `$content` api will be served on `http://localhost:3000/_content/articles`
- Provide different directory for writing content.
- Customise searchable fields.
- Provide custom PrismJS theme.

```
// nuxt.config.js
export default {
  content: {
    1 apiPrefix: '_content',
    2 dir: 'content',
    3 fullTextSearchFields:
      ['title', 'description', 'slug', 'text'],
    nestedProperties: ['categories.slug'],
    markdown: {
      remarkPlugins: [
        'remark-squeeze-paragraphs',
        'remark-slug',
        'remark-autolink-headings',
        'remark-external-links',
        'remark-footnotes'
      ],
      rehypePlugins: [
        'rehype-minify-whitespace',
        'rehype-sort-attribute-values',
        'rehype-sort-attributes',
        'rehype-raw'
      ],
      4 prism: {
        theme: 'prismjs/themes/prism.css'
      }
    },
    yaml: {},
    csv: {},
    xml: {}
  }
}
```

Learn more about custom configuration:
<https://content.nuxtjs.org/configuration>

Recarregamento instantâneo em desenvolvimento

O módulo de conteúdo é extremamente rápido quando vem para o recarregamento instantâneo em desenvolvimento por não ter que passar pelo webpack sempre que você fizer mudanças em seus ficheiros markdown. Você pode também ouvir o evento `content:update` e criar um plugin que em todo momento que você atualizar um ficheiro dentro do seu diretório de conteúdo ele despachará por exemplo um método `fetchCategories`.

Next

[Consulte a documentação do módulo de conteúdo para obter mais detalhes](#)

Exibindo o conteúdo

Você pode diretamente usar o componente `<nuxt-content>` dentro do seu modelo para exibir o corpo da página.

```
<template>
  <article>
    <nuxt-content :document="article" />
  </article>
</template>
```

Next

[Consulte a documentação do módulo de conteúdo para obter mais detalhes](#)

Estilizando o seu conteúdo

Dependendo do que você está usando para desenhar a sua aplicação, você pode precisar escrever alguns estilos para exibir o markdown de maneira decente.

O componente `<nuxt-content>` irá automaticamente adicionar uma classe `.nuxt-content`, você pode usá-la para personalizar seus estilos.

```
<style>
  .nuxt-content h2 {
    font-weight: bold;
    font-size: 28px;
  }
  .nuxt-content p {
    margin-bottom: 20px;
  }
</style>
```

Next

[Consulte a documentação do módulo de conteúdo para obter mais detalhes](#)

Manipular ficheiros markdown, csv, yaml, json(5)

Este módulo converte o seus ficheiros `.md` em uma estrutura de árvore JSON AST, guardada dentro de uma variável `body`. Você pode também adicionar um bloco de assunto dianteiro em YAML aos seus ficheiros markdown ou um ficheiro `.yaml` que será injetado dentro do documento. Você pode também adicionar um ficheiro json/json5 que pode também ser injetado dentro do documento. E você pode usar um ficheiro `.csv` onde as linhas serão atribuídas a variável `body`.

```
---  
title: My first Blog Post  
description: Learning how to use @nuxt/content to create a blog  
---
```

Next

Consulte a [documentação do módulo de conteúdo](#) para obter mais detalhes

Componentes do vue dentro do markdown

Você pode usar componentes Vue diretamente dentro do seus ficheiros markdown. No entanto você precisará usar seus componentes em kebab case e não pode usar marcadores com auto fechamento.

```
<template>  
  <div class="p-4 mb-4 text-white bg-blue-500">  
    <p><slot name="info-box">default</slot></p>  
  </div>  
</template>
```

```
<info-box>  
  <template #info-box>  
    This is a vue component inside markdown using slots  
  </template>  
</info-box>
```

Next

Consulte a [documentação do módulo de conteúdo](#) para obter mais detalhes

API totalmente pesquisáveis

Você pode usar `$content()` para listar, filtrar e pesquisar seus conteúdo facilmente.

```
<script>  
export default {  
  async asyncData({ $content, params }) {  
    const articles = await $content('articles', params.slug)  
      .only(['title', 'description', 'img', 'slug', 'author'])  
      .sortBy('createdAt', 'asc')  
      .fetch()  
  
    return {  
      articles  
    }  
  }  
</script>
```

```

        }
    }
</script>

```

Next

Consulte a [documentação do módulo de conteúdo](#) para obter mais detalhes

Artigos anteriores e próximos artigos

O módulo de conteúdo inclui um método `.surround(slug)`, é assim que você recebe com facilidade os artigos anteriores e próximos.

```

async asyncData({ $content, params }) {
  const article = await $content('articles', params.slug).fetch()

  const [prev, next] = await $content('articles')
    .only(['title', 'slug'])
    .sortBy('createdAt', 'asc')
    .surround(params.slug)
    .fetch()

  return {
    article,
    prev,
    next
  }
},

```

```
<prev-next :prev="prev" :next="next" />
```

Next

Consulte a [documentação do módulo de conteúdo](#) para obter mais detalhes

Pesquisa completa de texto

O módulo de conteúdo vem com uma pesquisa completa de texto assim você pode facilmente pesquisar ao longo de seus ficheiros markdown sem ter que instalar nada.

```

<script>
export default {
  data() {
    return {
      searchQuery: '',
      articles: []
    }
  },
  watch: {
    async searchQuery(searchQuery) {
      if (!searchQuery) {
        this.articles = []
        return
      }
      this.articles = await this.$content('articles')
        .limit(6)
        .search(searchQuery)
    }
  }
}

```

```

        .fetch()
    }
}
</script>

```

Next

Consulte a [documentação do módulo de conteúdo](#) para obter mais detalhes

Destacando a sintaxe

Este módulo envolve automaticamente o bloco de código e aplica as classes do [Prism](#). Você pode também adicionar uma tema diferente do Prism ou desativar ele totalmente.

```
yarn add prism-themes
```

```
npm install prism-themes
```

```

content: {
  markdown: {
    prism: {
      theme: 'prism-themes/themes/prism-material-oceanic.css'
    }
  }
}

```

Next

Consulte a [documentação do módulo de conteúdo](#) para obter mais detalhes

Estender o parseamento do markdown

Originalmente o markdown não suporta destaqueamento de linhas dentro de um bloco de código nem de nomes de ficheiros. O módulo de conteúdo permite isto com sua própria sintaxe personalizada. Linhas numeradas são adicionadas a tag `pre` dentro dos atributos `data-line` e o nome do ficheiro será convertido para um `span` com uma classe `filename`, assim você pode estilizar ela.

Next

Consulte a [documentação do módulo de conteúdo](#) para obter mais detalhes

Geração de tabela de conteúdos

Uma propriedade do tipo array `toc`(Table of Contents, Table de Conteúdos) será injetada dentro do seu documento, listando todos os cabeçalhos com seus títulos e ids, assim você pode ligar a eles.

```

<nav>
  <ul>
    <li v-for="link of article.toc" :key="link.id">
      <NuxtLink :to="`#${link.id}`>{{ link.text }}</NuxtLink>
    </li>
  </ul>
</nav>

```

Next

Consulte a [documentação do módulo de conteúdo](#) para obter mais detalhes

A poderosa API construtora de consulta (parecida com MongoDB)

O módulo de conteúdo vem com uma poderosa API construtora de consulta semelhante ao MongoDB que permite você facilmente ver o JSON de cada diretório em `http://localhost:3000/_content/`. O alvo final é acessível as requisições GET e POST, assim você pode usar parâmetros de consulta.

```
http://localhost:3000/_content/articles?only=title&only=description&limit=10
```

Next

Consulte a [documentação do módulo de conteúdo](#) para obter mais detalhes

Estender com gatilhos

Você pode usar gatilhos para estender o módulo, assim você pode adicionar dados a um documento antes dele ser guardado.

Next

Consulte a [documentação do módulo de conteúdo](#) para obter mais detalhes

Integração com o `@nuxtjs/feed`

No caso de artigos, o conteúdo pode ser usado para gerar alimentador de notícias usando o módulo [@nuxtjs/feed](#).

Next

Consulte a [documentação do módulo de conteúdo](#) para obter mais detalhes

Suporte a geração de site estático

O módulo de conteúdo funciona com geração de site estático usando o `nuxt generate`. Todas as rotas serão automaticamente gerada graças ao funcionalidade rastreadora do Nuxt.

Warning

Se estiveres usando o Nuxt < 2.13 e você precisa especificar as rotas dinâmicas, você pode fazer isso usando programaticamente a propriedade `generate` e `@nuxt/content`.

Next

Consulte a [documentação do módulo de conteúdo](#) para obter mais detalhes sobre o uso programático

O que segue

Next

Consulte nosso tutorial em [How to Create a Blog with Nuxt Content](#)

Next

Consulte a [documentação do módulo de conteúdo](#) para uso mais avançados e exemplos

O diretório dist

A pasta `dist`, forma abreviada para pasta de *distribuition (distribuição)*, é dinamicamente gerada quando se usa os comandos `nuxt generate` e inclui os ficheiros HTML e assets gerados e prontos para produção que são necessários para desdobrar e executar sua aplicação Nuxt estaticamente gerada.

Desdobrando

Esta é a pasta que você precisa **carregar para hospedagem estática** visto que ele contém seus ficheiros HTML e assets gerados e prontos para produção.

Warning

O diretório `dist` não deve ser enviado para o seu sistema de controlo de versão e deve ser ignorado através do seu `.gitignore` visto que ele será gerado automaticamente toda vez que você executar o `nuxt generate`.

A propriedade dir

A pasta dist é nomeada dist por padrão mas pode ser configurada dentro do seu ficheiro `nuxt.config.js`.

```
generate: {
  dir: 'my-site'
}
```

Warning

Se você mudar a sua pasta dist então você precisará adicionar ele ao seu sistema de controlo de versão assim o git ignorará ele.

A propriedade subFolders

O Nuxt coloca todas as suas páginas geradas dentro de uma pasta por padrão, no entanto você pode mudar isso se você quiser ao modificar o `nuxt.config.js` e mudando o subFolders para ser false.

```
generate: {
  subFolders: false
}
```

A propriedade fallback

Quando estiver fazendo o deploy do seu site você precisará certificar-se que o caminho para o HTML do fallback está corretamente definido. Ele deve ser definido como a página de erro uma vez que as rotas desconhecidas são renderizadas via Nuxt.

Quando estiver executando uma aplicação de página única faz mais sentido usar o `200.html` assim ele é o único ficheiro necessário, visto que nenhuma das outras rotas é gerada.

Quando estiver trabalhando com páginas geradas estaticamente, é recomendado usar o 404.html para páginas de erro.

Warning

Dependendo de onde você estiver hospedando o seu sítio, você pode ter que usar o 200.html ou o 404.html. Consulte com o seu provedor de hospedagem. O Netlify, por exemplo usa o 404.html.

```
export default {
  generate: {
    fallback: '404.html'
  }
}
```

A propriedade exclude

Você pode excluir páginas de serem geradas ao usar a propriedade excludes do generate. Ao invés de serem geradas como uma página estática ela voltará a ser uma página de uma aplicação de página única e somente será renderizada no lado do cliente.

```
generate: {
  exclude: [/admin/]
}
```

Info

Você pode também usar aqui uma expressão regular para excluir páginas que começam ou terminam com uma palavra em particular

O diretório layouts

Os esquemas (layouts) são de uma grande ajuda sempre que você quiser mudar a aparência e a emoção da sua aplicação Nuxt. Seja quando você quiser incluir uma barra lateral ou ter um esquema distinto para o mobile e desktop.

Warning

Este diretório não pode ser renomeado sem configuração extra.

O esquema padrão

Você pode estender o esquema principal ao adicionar um ficheiro `layouts/default.vue`. Ele será usado para todas as páginas que não tiverem um esquema especificado. Certifique-se de adicionar o componente `<Nuxt>` quando estiver criando um esquema para efetivamente incluir o componente de página.

Todo o que você precisa dentro do seu esquema são três linhas de código que renderizarão o componente da página.

```
<template>
  <Nuxt />
</template>
```

Você pode adicionar aqui mais componentes tais como Navegação (Navigation), Cabeçalho (Header), Rodapé (Footer) etc.

```
<template>
  <div>
    <TheHeader />
    <Nuxt />
    <TheFooter />
  </div>
</template>
```

Info

Se você tiver a propriedade `components` definida para `true` então não há necessidade de adicionar qualquer declaração `import` para os seus componentes.

O esquema personalizado

Todo ficheiro de (*alto-nível*) dentro do diretório `layouts` criará um esquema personalizado acessível com a propriedade `layout` dentro dos componentes da página.

Vamos dizer que queremos criar um esquema de blogue e guardar ele em `layouts/blog.vue`:

```
<template>
  <div>
    <div>My blog navigation bar here</div>
```

```
<Nuxt />
</div>
</template>
```

Depois você tem de dizer para as páginas usarem o seu esquema personalizado

```
<script>
export default {
  layout: 'blog',
  // Ou
  layout (context) {
    return 'blog'
  }
}</script>
```

A página de erro

A página de erro é um *componente de página* que é sempre exibido quando um erro ocorre (que não seja lançado no lado do servidor).

Warning

Embora este ficheiro esteja posto dentro da pasta `layouts`, ele deve ser tratado como uma página.

Como já foi mencionado acima, este esquema é especial e você não deve incluir o `<Nuxt>` dentro do seu modelo (template). Você deve ver este esquema como um componente exibido quando um erro ocorre (`404`, `500` etc). Similar aos outros componentes de página, você pode definir um esquema personalizado para a página de erro.

Você pode personalizar a página de erro ao adicionar um ficheiro `layouts/error.vue`:

```
<template>
<div class="container">
  <h1 v-if="error.statusCode === 404">Page not found</h1>
  <h1 v-else>An error occurred</h1>
  <NuxtLink to="/">Home page</NuxtLink>
</div>
</template>

<script>
export default {
  props: ['error'],
  layout: 'blog' // você pode definir um esquema personalizado para a página de
erro
}
</script>
```

Info

O código-fonte da página de erro padrão está [disponível no GitHub](#).

[Go to TOC](#)

O diretório middleware

O diretório `middleware` contém suas aplicações intermediárias. O intermediário permite você definir funções personalizadas que podem ser executadas antes da renderização, seja de uma página ou um grupo de páginas (esquema).

O intermediário partilhado deve ser colocado dentro do diretório `middleware/`. O nome do ficheiro será o nome do intermediário (`middleware/auth.js` será o intermediário `auth`). Você pode também definir intermediário específico da página ao usar uma função diretamente, vé [intermediário anônimo](#).

Um intermediário recebe [o contexto](#) como o primeiro argumento.

```
export default function (context) {
  // Adiciona a propriedade `userAgent` ao contexto
  context.userAgent = process.server
    ? context.req.headers['user-agent']
    : navigator.userAgent
}
```

No modo universal, os intermediários serão chamados uma vez no lado do servidor (na primeira requisição para a aplicação Nuxt, exemplo de quando estiverem diretamente acessando a aplicação ou atualizado a página) e no lado do cliente quando estiverem navegando para as rotas adiante. Com `ssr: false`, o intermediário será chamado no lado do cliente em ambas situações.

O intermédio será executado em série nesta ordem:

1. `nuxt.config.js` (na ordem dentro do ficheiro)
2. Esquemas correspondidos
3. Páginas correspondidas

O intermediário do roteador

Um intermediário pode ser assíncrono. Para fazer isso retorne uma `Promise` ou usar `async/await`.

```
import http from 'http'

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

Depois, dentro do seu `nuxt.config.js`, use a chave `router.middleware`.

```
export default {
  router: {
    middleware: 'stats'
  }
}
```

Agora o intermediário `stats` será chamado para toda mudança de rota.

Você pode adicionar o seu intermediário (até mesmo vários) também para um esquema ou página específica.

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

Intermediário nomeado

Você pode criar o intermediário nomeado ao criar um ficheiro dentro do diretório `middleware/`, o nome do ficheiro será o nome do intermediário.

```
export default function ({ store, redirect }) {
  // Se o usuário não estiver autenticado
  if (!store.state.authenticated) {
    return redirect('/login')
  }
}
```

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

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

Intermediário anônimo

Se você precisar usar um intermediário somente para uma página específica, você pode diretamente usar uma função para isso (ou um arranjo de funções):

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

<script>
  export default {
    middleware({ store, redirect }) {
      // Se o usuário não estiver autenticado
      if (!store.state.authenticated) {
        return redirect('/login')
      }
    }
  }
</script>
```

O diretório `modules`

O Nuxt fornece um sistema de módulo de alta ordem que torna possível estender o núcleo. Os módulos são funções que são chamados sequencialmente sempre o Nuxt estiver iniciando.

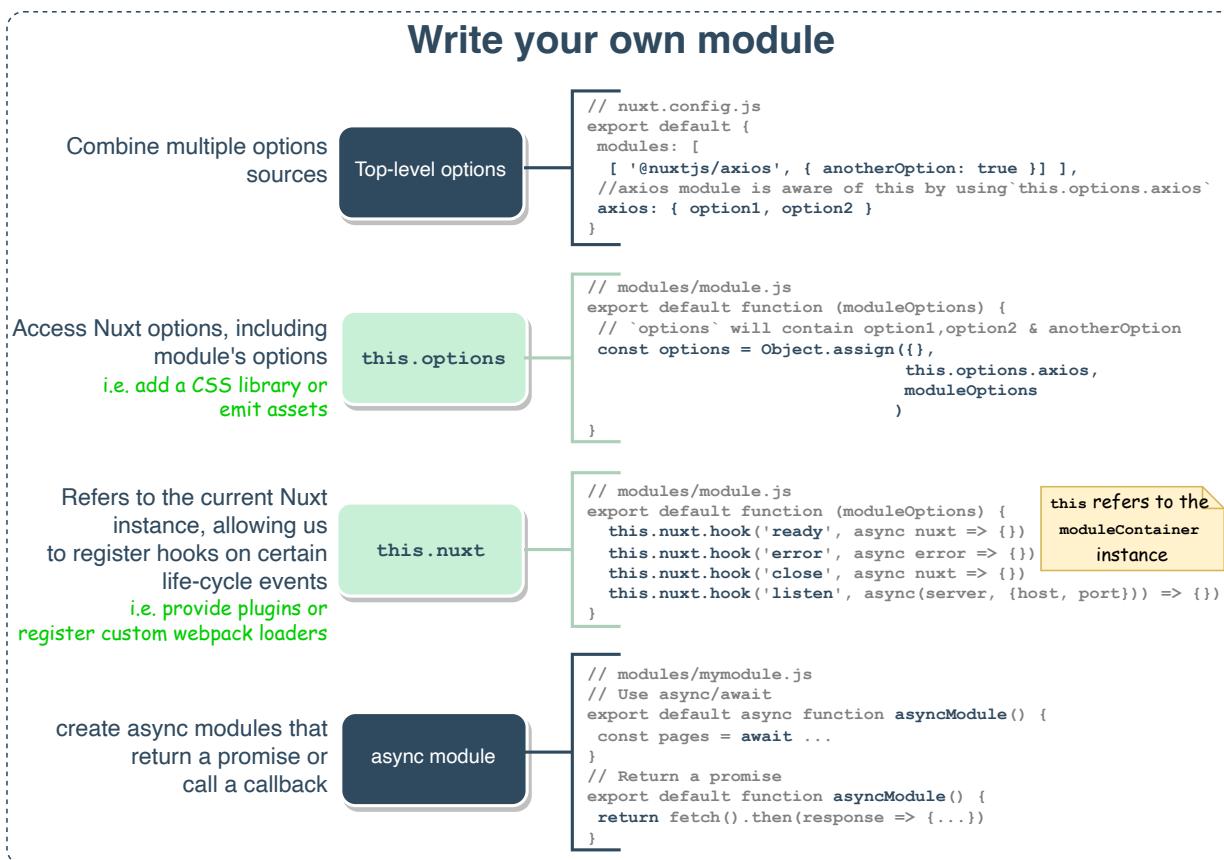
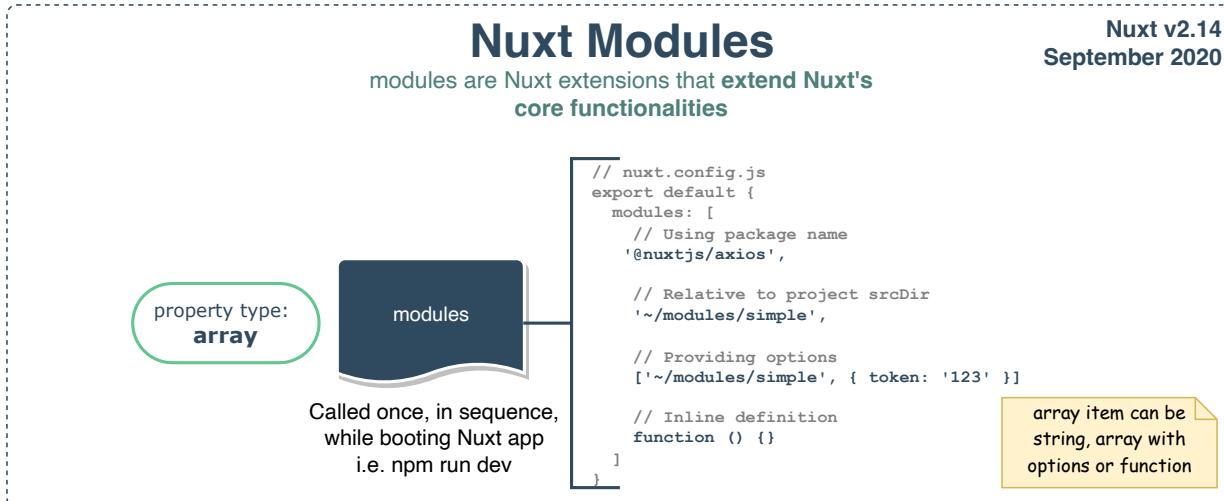
Explorando os módulos do Nuxt

Descubra nossa [lista de módulos](#) para super carregar o seu projeto Nuxt, criado pela equipa do Nuxt e pela comunidade.

- 165+ Módulos
- 105+ Mantenedores

Next

Consulte o modules.nuxtjs.org



Use modules

If module:

- provides serverMiddleware
- registers Node.js runtime hook
- affects vue-renderer behaviour
- or anything that's outside of webpack scope

```
// nuxt.config.js
export default {
  modules: ['@nuxtjs/sitemap']
}
```

Use buildModules

If module is imported during dev and build time.

simply, add dependency in devDependencies and USE buildModules:

```
// nuxt.config.js
export default {
  buildModules: ['@nuxtjs/eslint-module']
}
```

Benefits:

- ✓ Decrease node_module size
- ✓ Make production startup faster

Enquanto estiver desenvolvendo aplicações para produção com Nuxt você pode achar que o núcleo de funcionalidade do framework não é suficiente. O Nuxt pode ser estendido com opções de configuração e plugins, mas manter essas personalizações através de vários projetos é tedioso, repetitivo e consome tempo. Por outro lado, atender a cada necessidade do projeto fora da caixa tornaria o Nuxt muito complexo e difícil de usar.

Esta é uma das razões do porquê o Nuxt fornecer um sistema de módulo de alta ordem que torna possível estender o núcleo. Os módulos são funções que são chamados sequencialmente sempre o Nuxt estiver iniciando. O framework espera cada módulo terminar antes de continuar. Desta maneira, os módulos podem personalizar quase qualquer aspecto da sua aplicação. Graças o desenho modular do Nuxt (baseado no [Tapable](#) do webpack), os módulos podem facilmente registrar gatilhos para certos pontos de entrada como a inicialização do construtor. Os módulos podem também sobreescriver modelos, configurar os carregadores do webpack, adicionar bibliotecas CSS, e realizar muitas outras tarefas úteis.

O melhor de tudo, os módulos do Nuxt podem ser incorporados dentro dos pacotes npm. Isto torna possível re-usar através dos projetos e partilhar com a comunidade, ajudando a criar um ecossistema de recursos adicionáveis de alta qualidade.

A propriedade modules

Os módulos são extensões do Nuxt que podem estender a funcionalidade do núcleo do framework e adicionar integrações sem fim. Uma vez você ter instalado os módulos você pode então adicionar eles ao seu arquivo `nuxt.config.js` dentro da propriedade `modules`.

```
export default {
  modules: [
    // Usando o nome do pacote
    '@nuxtjs/axios',

    // Relativo ao diretório fonte do seu projeto (srcDir)
    '~/modules/awesome.js',

    // Fornecendo opções
    ['@nuxtjs/google-analytics', { ua: 'X1234567' }],

    // Definição em linha
    function () {}
  ]
}
```

Info

Desenvolvedores de módulo geralmente fornecem passos adicionais necessários e detalhes para utilização.

O Nuxt tenta resolver cada item dentro do array de módulos usando o caminho exigido pelo node (dentro do `node_modules`) e depois resolverá a partir do `srcDir` se o apelido `@` for usado.

Warning

Os módulos são executados sequencialmente, então a ordem é importante.

Os módulos devem exportar uma função para destacar a construção/tempo de execução e opcionalmente retorna uma promessa até que o trabalho deles esteja terminado. Repare que eles são importados em tempo de execução então eles devem já estar transpilados se estiverem usando as funcionalidades modernas do ES6.

Escreva o seu próprio módulo

Os módulos são funções. Eles podem ser empacotados como módulos npm ou diretamente incluídos dentro do código-fonte do seu projeto.

```
export default {
  exampleMsg: 'hello',
  modules: [
    // Uso simples
    '~/modules/example',
    // Passando as opções diretamente
    ['~/modules/example', { token: '123' }]
  ]
}
```

```
export default function ExampleModule(moduleOptions) {
  console.log(moduleOptions.token) // '123'
  console.log(this.options.exampleMsg) // 'hello'

  this.nuxt.hook('ready', async nuxt => {
    console.log('Nuxt is ready')
  })
}

// Obrigatório se estiver publicando o módulo como pacote npm
module.exports.meta = require('./package.json')
```

1) ModuleOptions

`moduleOptions` : este é o objeto passado usando o array `modules` pelo usuário. Podemos usar ele para personalizar seu comportamento,

Opções de alto nível

Algumas vezes é muito mais conveniente podermos usar opções de alto nível enquanto estivermos registrando módulos dentro do `nuxt.config.js`. Isto permite-nos combinar várias fontes de opções.

```
export default {
  modules: [['@nuxtjs/axios', { anotherOption: true }]],
  // o módulo axios está ciente disto ao usar `this.options.axios`
  axios: {
    option1,
    option2
  }
}
```

2) this.options

`this.options` : Você pode acessar diretamente as opções do Nuxt usando esta referência. Este é o conteúdo do `nuxt.config.js` do usuário com todas opções padrão atribuídas a ela. Ela pode ser usada para opções partilhadas entre os módulos.

```
export default function (moduleOptions) {
  // `options` conterá option1, option2 e anotherOption
  const options = Object.assign({}, this.options.axios, moduleOptions)

  // ...
}
```

Adicionar uma biblioteca CSS

Se o seu módulo fornecerá uma biblioteca CSS, certifique-se de realizar uma verificação para saber se o usuário já incluiu a biblioteca para evitar, e adicionar uma opção para desativar a biblioteca CSS dentro do módulo.

```
export default function (moduleOptions) {
  if (moduleOptions.fontAwesome !== false) {
    // Adicione o font-awesome
    this.options.css.push('font-awesome/css/font-awesome.css')
  }
}
```

Emitir os recursos

Nós podemos registrar os plugins do webpack para emitir os recursos durante a construção.

```
export default function (moduleOptions) {
  const info = 'Built by awesome module - 1.3 alpha on ' + Date.now()

  this.options.build.plugins.push({
    apply(compiler) {
      compiler.plugin('emit', (compilation, cb) => {
        // Isto gerará o `.nuxt/dist/info.txt` com os conteúdos da variável info.
        // A fonte (source) pode ser uma memória temporária também.
        compilation.assets['info.txt'] = {
          source: () => info,
          size: () => info.length
        }

        cb()
      })
    }
  })
}
```

3) this.nuxt

`this.nuxt` : Isto é uma referência a instância atual do Nuxt. Nós podemos registrar gatilhos em certos eventos do ciclo de vida.

- **Ready** : O Nuxt está pronto para trabalhar (ModuleContainer e Renderer prontos).

```
nuxt.hook('ready', async nuxt => {
  // O seu código personalizado vai aqui
})
```

- **Error**: Um erro não manipulado quando estiver chamando os gatilhos.

```
nuxt.hook('error', async error => {
  // O seu código personalizado vai aqui
})
```

- **Close**: A instância do Nuxt está fechando graciosamente.

```
nuxt.hook('close', async nuxt => {
  // O seu código personalizado vai aqui
})
```

- **Listen**: O servidor interno do Nuxt começa ouvindo. (Usando o `nuxt start` ou `nuxt dev`)

```
nuxt.hook('listen', async (server, { host, port }) => {
  // O seu código personalizado vai aqui
})
```

`this`: O contexto dos módulos. Todos os módulos são chamados dentro contexto da instância ModuleContainer.

Consulte a documentação da classe [ModuleContainer](#) para conhecer os métodos disponíveis.

Executar tarefas em gatilhos específicos

O seu módulo pode precisar fazer coisas apenas sobre condições específicas e não somente durante a inicialização do Nuxt. Nós podemos usar os poderosos gatilhos do Nuxt para realizar tarefas em eventos específicos (baseado no [Hookable](#)). O Nuxt esperará pela sua função para saber se ela retorna uma promessa ou está definida como `async`.

Here are some basic examples:

```
export default function myModule() {
  this.nuxt.hook('modules:done', moduleContainer => {
    // Isto será chamado quando todos os módulos terminarem o carregamento
  })

  this.nuxt.hook('render:before', renderer => {
    // Chamado depois do renderizador ser criado
  })

  this.nuxt.hook('build:compile', async ({ name, compiler }) => {
    // Chamado antes do compilador (padrão: webpack) começar
  })

  this.nuxt.hook('generate:before', async generator => {
    // Isto será chamado antes do Nuxt gerar suas páginas
  })
}
```

Fornecer plugins

É comum que módulos forneçam um ou mais plugins quando adicionados. Por exemplo o módulo [bootstrap-vue](#) precisaria registar a si mesmo dentro do Vue. Em tais situações nós podemos usar o auxiliar `this.addPlugin`.

```
import Vue from 'vue'
import BootstrapVue from 'bootstrap-vue/dist/bootstrap-vue.esm'

Vue.use(BootstrapVue)

import path from 'path'

export default function nuxtBootstrapVue(moduleOptions) {
  // Registe o modelo `plugin.js`
  this.addPlugin(path.resolve(__dirname, 'plugin.js'))
}
```

Observe que: Quaisquer plugins injetados pelos módulos são adicionados no **princípio** da lista de plugins. Suas opções são:

- Manualmente adicionar o plugin para o final da lista de plugins
`(this.nuxt.options.plugins.push(...))`
- Inverter a ordem dos módulos se ele depender de um outro

Os plugins do modelo

Os modelos registados e plugins podem influenciar o [os modelos do lodash](#) para condicionalmente mudar a saída dos plugins registados.

```
// Define o Google Analytics UA
ga('create', '<%= options.ua %>', 'auto')

<% if (options.debug) { %>
// Apenas código do desenvolvedor
<% } %>

import path from 'path'

export default function nuxtGoogleAnalytics(moduleOptions) {
  // Registrar o modelo `plugin.js`
  this.addPlugin({
    src: path.resolve(__dirname, 'plugin.js'),
    options: {
      // O Nuxt substituirá `options.ua` com `123` quando estiver copiando o
      // plugin para o projeto
      ua: 123,

      // partes condicionais com `dev` será desfeito do código do plugin nas
      // construções de produção
      debug: this.options.dev
    }
  })
}
```

Registrar carregadores personalizados do webpack

Nós podemos fazer o mesmo com o `build.extend` dentro do `nuxt.config.js` usando `this.extendBuild`.

```
export default function (moduleOptions) {
  this.extendBuild((config, { isClient, isServer }) => {
    // Carregador do `foo`
    config.module.rules.push({
      test: /\.foo$/,
      use: [...]
    })

    // Personalizar carregadores existentes
    // Recorra ao código-fonte para o interior do Nuxt:
    //
    https://github.com/nuxt/nuxt.js/blob/dev/packages/webpack/src/config/base.js
      const barLoader = config.module.rules.find(rule => rule.loader === 'bar-
      loader')
    })
  }
}
```

Os módulos assíncronos

Nem todos os módulos farão tudo de forma síncrona. Por exemplo, você talvez queira desenvolver um módulo que precisar requisitar alguma API ou fazer operações assíncronas. Para isto, o Nuxt suporta módulos assíncronos que podem retornar uma promessa ou chamar um callback.

Use o `async/await`

```
import fse from 'fs-extra'

export default async function asyncModule() {
  // Você pode fazer trabalho assíncrono aqui usando `async`/`await`
  const pages = await fse.readJson('./pages.json')
}
```

Retornar uma promessa

```
export default function asyncModule($http) {
  return $http
    .get('https://jsonplaceholder.typicode.com/users')
    .then(res => res.data.map(user => '/users/' + user.username))
    .then(routes => {
      // Faça alguma coisa ao estender as rotas do nuxt
    })
}
```

Info

Existem mais gatilhos e possibilidades para módulos. Consulte o [Interior do Nuxt](#) para achar mais informações sobre a API nuxt-internal.

Publicando o seu módulo

`module.exports.meta` : Esta linha é exigida se você estiver publicando o módulo como um pacote npm. O Nuxt usa internamente o `meta` para trabalhar melhor com seu pacote.

```
module.exports.meta = require('./package.json')
```

A propriedade buildModules

Alguns módulos são apenas importados durante o tempo de desenvolvimento e construção. Usar `buildModules` ajuda tornar inicio da produção rápido e também diminuir significativamente o tamanho do seu diretório `node_modules` para deployments em produção. Recorra à documentação para cada módulo para ver se é recomendado usar a propriedade `modules` ou `buildModules`.

A diferença de uso é:

- Ao invés de adicionar ao `modules` dentro do `nuxt.config.js`, adicione ao `buildModules`

```
export default {
  buildModules: ['@nuxtjs/eslint-module']
}
```

- Ao invés de adicionar ao `dependencies` dentro do `package.json`, adicione ao `devDependencies`

```
yarn add --dev @nuxtjs/eslint-module
```

```
npm install --save-dev @nuxtjs/eslint-module
```

Info

Se você é um autor de módulo, é altamente recomendado sugerir aos usuários que instalem o seu pacote como uma `devDependency` e usar o `buildModules` ao invés de `modules` para o `nuxt.config.js`.

O seu módulo é um `buildModules` a menos que:

- Ele esteja fornecendo um `serverMiddleware`
- Ele tem de registrar um gatilho para o tempo de execução do Node.js (como `sentry`)
- Ele esteja afetando o comportamento do `vue-renderer` ou usando um gatilho do espaço de nome `server:` ou `vue-renderer`
- Outra coisa que esteja fora do escopo do webpack (sugestão: plugins e modelos que são compilados e estão dentro do escopo do webpack)

Warning

Se você está oferecendo o uso do `buildModules` mencione que esta funcionalidade está apenas disponível a partir da versão 2.9 do Nuxt. Usuários antigos devem atualizar ou usar a secção `modules`.

[Go to TOC](#)

O diretório pages

O diretório `pages` contém as views e rotas da sua aplicação. O Nuxt lê todos os ficheiros `.vue` dentro desse diretório e cria automaticamente a configuração do router por você.

Info

Você pode também criar rotas com ficheiros `.js` e `.ts`

Todo componente de página é um componente Vue mas o Nuxt adiciona atributos e funções especiais para tornar o desenvolvimento da sua aplicação universal o mais fácil possível.

```
<template>
  <h1 class="red">Hello {{ name }}!</h1>
</template>

<script>
  export default {
    // as propriedades da página vão aqui
  }
</script>

<style>
  .red {
    color: red;
  }
</style>
```

Páginas dinâmicas

As páginas dinâmicas podem ser criadas quando você não sabe o nome da página devido ao fato dela vir de uma API ou você não quer ter de criar a mesma página uma vez a outra. Para criar uma página dinâmica você precisa adicionar um sublinhado antes do nome do ficheiro `.vue` ou antes do nome do diretório, se você quiser que o diretório seja dinâmico. Você pode nomear o ficheiro ou diretório com o que você quiser mas você deve adicionar um prefixo nele com um sublinhado.

Se você tiver definido um ficheiro nomeado como `_slug.vue` dentro da sua pasta `pages`, você pode acessar o valor usando o contexto com o `params.slug`.

```
<template>
  <h1>{{ this.slug }}</h1>
</template>

<script>
  export default {
    async asyncData({ params }) {
      const slug = params.slug // Quando estiver chamando /abc o slug será "abc"
      return { slug }
    }
  }
</script>
```

Se você tiver definido um ficheiro nomeado como `_slug.vue` dentro de uma pasta chamada `_hook` você pode acessar o valor usando o contexto com o `params.slug` e o `params.book`

```
<template>
  <h1>{{ this.book }} / {{ this.slug }}</h1>
</template>

<script>
  export default {
    async asyncData({ params }) {
      const book = params.book
      const slug = params.slug
      return { book, slug }
    }
  }
</script>
```

Propriedades

A propriedade `asyncData`

O `asyncData` é chamado toda vez antes do componente de carregamento. Ele pode ser assíncrono e receber o contexto como um argumento. O objeto retornado será combinado com o seu objeto `data`.

```
export default {
  asyncData(context) {
    return { name: 'World' }
  }
}
```

Next

Para saber mais sobre como o `asyncData` funciona consulte o nosso capítulo [Requisição de Dados](#)

A propriedade `fetch`

Toda vez que você precisar receber dados assíncronos você pode usar o `fetch`. O `fetch` é chamado no lado do servidor quando estiver renderizando a rota, e no lado do cliente quando estiver navegando.

```
<script>
  export default {
    data() {
      return {
        posts: []
      }
    },
    async fetch() {
      this.posts = await fetch('https://api.nuxtjs.dev/posts').then(res =>
        res.json()
      )
    }
  }
</script>
```

Next

Para saber mais sobre como o fetch funciona consulte o nosso capítulo [Requisição de Dados](#)

A propriedade head

Definir marcadores de `<meta>` específicas para página atual. O Nuxt usa o `vue-meta` para atualizar o head e meta atributos do documento da sua aplicação.

```
export default {
  head() {
    // Defina os marcadores de meta para esta página
  }
}
```

Next

Para saber mais sobre o head consulte o nosso capítulo [Marcadores de Meta e SEO](#)

A propriedade layout

Especifica um esquema definido dentro do diretório de layouts.

```
export default {
  layout: 'blog'
}
```

Next

Para saber mais sobre layouts consulte o nosso capítulo de [Apresentações](#).

A propriedade loading

Se definido para `false`, previne que uma página de automaticamente chamar `this.$nuxt.$loading.finish()` assim que você entrar nela e `this.$nuxt.$loading.start()` assim que você sair dela, permitindo que você manualmente controle o comportamento, assim como é mostrado no exemplo.

```
export default {
  loading: false
}
```

Info

Apenas se aplica se loading estiver também definido dentro de `nuxt.config.js`.

Next

Para saber mais sobre loading consulte o nosso capítulo [Carregamento](#).

A propriedade transition

Define uma transição específica para a página.

```
export default {
  transition: 'fade'
}
```

Next

Para saber mais sobre transition consulte o nosso capítulo [Transições](#).

A propriedade scrollTop

A propriedade `scrollTop` permite você dizer ao Nuxt para rolar a página até ao topo antes da renderização da página. Por padrão, o Nuxt rola para o topo quando você vai para uma outra página, mas com as rotas filhas, o Nuxt mantém a posição da rolagem. Se você quiser dizer ao Nuxt para rolar para o topo quando estiver renderizando sua rota filha, defina `scrollTop` para `true`

```
export default {
  scrollTop: true
}
```

Pelo contrário, você pode manualmente definir `scrollTop` para `false` no pai das rotas também.

Se você quiser sobrescrever o comportamento padrão de rolagem do Nuxt, de uma vista de olhos na [opção scrollBehavior](#).

A propriedade middleware

Define um intermediário para esta página. O intermediário será chamado antes da renderização da página.

```
export default {
  middleware: 'auth'
}
```

Next

Para saber mais sobre middleware consulte o nosso capítulo [Intermediário](#).

A propriedade watchQuery (observar consulta)

Use a chave `watchQuery` para definir um observador para as strings da consulta. Se a string definida mudar, todos os métodos do componente (`asyncData`, `fetch(context)`, `validate`, `layout`, ...) serão chamados. A observação está desativada por padrão para melhorar a performance.

```
export default {
  watchQuery: ['page']
}
```

Warning

Aviso: O novo gatilho `fetch` introduzido na versão 2.12 não é afetado pelo `watchQuery`. Para obter mais informações consulte [ouvindo a mudanças na string da consulta](#).

```
export default {
  watchQuery: true
}
```

Você pode também usar a função `watchQuery(newQuery, oldQuery)` para ter observadores mais refinados.

```
export default {
  watchQuery(newQuery, oldQuery) {
    // Apenas execute os métodos do componente se a antiga sequência de caracteres
    // da consulta conter `bar`
    // e a nova sequência de caracteres da consulta conter `foo`
    return newQuery.foo && oldQuery.bar
  }
}
```

Next

Para saber mais sobre `watchQuery` consulte o nosso capítulo [Requisição de Dados](#).

A propriedade key

O mesmo é com a propriedade `key` que pode ser usada nos componentes Vue dentro dos modelos como uma dica para a DOM virtual, esta propriedade permite que a chave e valor sejam definidas a partir da própria página (em vez do componente pai).

Por padrão dentro Nuxt, este valor será o `$route.path`, querendo dizer que a navegação para uma rota diferente garantirá que um componente de página limpo seja criado. Logicamente equivalente a:

```
<router-view :key="$route.path" />
```

A propriedade pode ser uma `String` ou uma `Function` que recebe uma rota como o primeiro argumento.

Ignorando páginas

Se você quiser ignorar páginas para que elas não sejam incluídas dentro do ficheiro `router.js` gerado, então você pode ignorar elas ao prefixar elas com um `-`.

Por exemplo, `pages/-about.vue` será ignorada.

Next

Consulta a [opção ignore](#) para aprender mais sobre ela.

Configuração

Você pode renomear o diretório `pages/` para alguma coisa diferente ao definir a opção `dir.pages`:

```
export default {
  dir: {
    // Renomeia o diretório `pages` para `routes`
```

```
    pages: 'routes'  
  }  
}
```

Next

Consulte a [opção dir](#) para aprender mais sobre ela.

[Go to TOC](#)

A propriedade alias

O Nuxt permite você usar apelidos para acessar diretórios personalizados dentro do seu JavaScript e do seu CSS

- Type: `Object`
- Default:

```
{
  '~~': `<rootDir>`,
  '@@': `<rootDir>`,
  '~': `<srcDir>`,
  '@': `<srcDir>`,
  'assets': `<srcDir>/assets`, // (a menos que você tenha definido um
  `dir.assets` personalizado)
  'static': `<srcDir>/static`, // (a menos que você tenha definido um
  `dir.static` personalizado)
}
```

Esta opção permite você definir apelidos para os diretórios dentro do seu projeto (além daqueles acima). Estes apelidos podem ser usados dentro do seu JavaScript e do seu CSS.

```
import { resolve } from 'path'
export default {
  alias: {
    'images': resolve(__dirname, './assets/images'),
    'style': resolve(__dirname, './assets/style'),
    'data': resolve(__dirname, './assets/other/data')
  }
}
```

```
<template>
  
</template>

<script>
import data from 'data/test.json'

// etc.
</script>

<style>
@import '~style/variables.scss';
@import '~style/utils.scss';
@import '~style/base.scss';

body {
  background-image: url('~images/main-bg.jpg');
}
</style>
```

Warning

Dentro de um contexto do Webpack (fontes de imagem, CSS - mas não JavaScript) você deve prefixar o apelidos com `~` (assim como no exemplo acima).

Info

Se você estiver usando o TypeScript e quiser usar os apelidos que você define dentro do seus ficheiros TypeScript, você precisará adicionar os apelidos ao seu objeto de `paths` dentro do ficheiro `tsconfig.json`.

[Go to TOC](#)

A propriedade extendPlugins

A propriedade extendPlugins permite você personalizar os plugins do Nuxt. ([options.plugins](#)).

- Tipo: `Function`
- Valor padrão: `undefined`

Você talvez queira estender os plugins ou mudar a ordem dos plugins criada pelo Nuxt. Esta função aceita um arranjo de objetos de [plugin](#) e deve retornar um arranjo de objetos de plugins.

Exemplo de mudança de ordem dos plugins:

```
export default {
  extendPlugins(plugins) {
    const pluginIndex = plugins.findIndex(
      plugin => (typeof plugin === 'string' ? plugin : plugin.src) ===
      '~/plugins/shouldBeFirst.js'
    )
    const shouldBeFirstPlugin = plugins[pluginIndex]

    plugins.splice(pluginIndex, 1)
    plugins.unshift(shouldBeFirstPlugin)

    return plugins
  }
}
```

A propriedade generate

Configura a geração da sua aplicação web universal para uma aplicação web estática.

- Tipo: `Object`

Quando estiver chamando o `nuxt.generate()`, o Nuxt usará a configuração definida dentro da propriedade `generate`.

```
export default {
  generate: {
    ...
  }
}
```

A propriedade cache

Introduzida dentro da versão 2.14.0

- Tipo: `Object` ou `false`

Esta opção é usada pelo comando `nuxt generate` com o [alvo estático](#) para evitar a reconstrução quando nenhum ficheiro rastreado for mudado.

Valores padrão:

```
{
  ignore: [
    '.nuxt', // buildDir
    'static', // dir.static
    'dist', // generate.dir
    'node_modules',
    '**/*',
    '*',
    'README.md'
  ]
}
```

Se você quiser evitar a reconstrução quando estiver mudando um ficheiro de configuração, apenas adicione isto a lista pelo fornecimento da opção `cache.ignore`:

```
export default {
  generate: {
    cache: {
      ignore: ['renovate.json'] // ignorar mudanças aplicadas sobre este ficheiro
    }
  }
}
```

A propriedade concurrency

- Tipo: `Number`
- Valor padrão: `500`

A geração de rotas são concorrentes, o `generate.concurrency` especifica a quantidade de rotas que executam em uma fila.

A propriedade crawler

- Tipo: `boolean`
- Valor padrão: `true`

Desde a versão `2.13` do Nuxt, o Nuxt vem com um rastreador instalado que rasteará usas ligações relativas e gerar suas ligações dinâmicas baseada nestas ligações. Se você quiser desativar esta funcionalidade você pode definir o valor para `false`

```
export default {
  generate: {
    crawler: false
  }
}
```

A propriedade dir

- Tipo: `String`
- Valor padrão: `'dist'`

O nome do diretório criado quando estiver construindo aplicações web com uso do comando `nuxt generate`.

A propriedade devtools

- Tipo: `boolean`
- Valor padrão: `false`

Configura a permissão de inspeção para `vue-devtools`.

Se você já ativou através do ficheiro `nuxt.config.js` ou outra forma, a ferramenta de desenvolvimento ativa independentemente da bandeira.

A propriedade exclude

- Tipo: `Array`
 - Itens: `String` ou `RegExp`

Ela aceita um arranjo de sequência de caracteres ou expressões regulares e prevenirá a geração de rotas que correspondem a elas. As rotas continuarão a ser acessíveis quando o `generate.fallback` for usado.

Considere este exemplo de estrutura:

```
-| pages/
---| index.vue
---| admin/
-----| about.vue
-----| index.vue
```

Por padrão, a execução do comando `nuxt generate` criará um ficheiro para cada rota.

```
-| dist/
---| index.html
---| admin/
-----| about.html
-----| index.html
```

Quando estiver adicionando uma expressão regular a qual corresponde a todas rotas com o "ignore", ela prevenirá a geração dessas rotas.

```
export default {
  generate: {
    exclude: [
      /^\/admin/ // o caminho começa com /admin
    ]
  }
}
```

```
-| dist/
---| index.html
```

Você pode também excluir uma rota específica pela atribuição de uma sequência de caracteres:

```
export default {
  generate: {
    exclude: ['/my-secret-page']
  }
}
```

A propriedade fallback

- Tipo: `String` ou `Boolean`
- Valor padrão: `200.html`

```
export default {
  generate: {
    fallback: '404.html'
  }
}
```

O caminho para o ficheiro HTML alternativo. Ele deve ser definido como a página de erro, assim também as rotas desconhecidas serão renderizadas através do Nuxt. Se for definida ou não para um valor de falsidade, o nome de um ficheiro HTML alternativo será `200.html`. Se for definido para `true`, o nome de ficheiro será `404.html`. Se você fornecer uma sequência de caracteres como um valor, ela será usada no lugar.

`fallback: false;`

Se estiver trabalhando com páginas estaticamente geradas então é recomendado usar um ficheiro `404.html` para páginas de erros e para aqueles cobertos pelo `excludes` (os ficheiros que você não quer que sejam gerados como páginas estáticas).

`fallback: true`

Contudo, o Nuxt permite você configurar qualquer página que você quiser, assim se você não quiser usar o ficheiro `200.html` ou `404.html`, você pode adicionar uma sequência de caracteres e depois você apenas tem de certificar-se de redirecionar para aquela página no lugar.

`fallback: 'fallbackPage.html'`

Nota: Vários serviços (por exemplo, Netlify) detetam um `404.html` automaticamente. Se você configura o seu servidor web por conta própria, consulte a sua documentação para saber como definir uma página de erro (e defina ela para o ficheiro `404.html`)

A propriedade interval

- Tipo: `Number`
- Valor padrão: `0`

Intervalo em milissegundos entre dois ciclos de renderização para evitar o transbordar de uma API em potencial com chamadas da aplicação web.

A propriedade minify

- **Depreciada!**
- Use o `build.html.minify` no lugar instead

A propriedade routes

- Tipo: `Array`

Info

Desde a versão `2.13` do Nuxt, existe um rastreador instalado que rastreia suas tags de ligação e gerar suas rotas quando estiver executando o comando `nuxt generate`.

Se tiver páginas desligadas (tais como páginas secretas) e você gostaria que também essas fossem geradas então você pode usar a propriedade `generate.routes`.

Warning

As rotas dinâmicas são ignoradas pelo comando `generate` quando estiver usando uma versão do Nuxt menor ou igual a `2.12`

Exemplo:

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

Apenas a rota `/` será gerada pelo Nuxt.

Se você quiser que o Nuxt gere rotas com parâmetros dinâmico, você precisa definir a propriedade `generates.routes` para um arranjo de rotas dinâmicas.

Nós adicionamos rotas para o `/users/:id`:

```
export default {
  generate: {
    routes: ['/users/1', '/users/2', '/users/3']
  }
}
```

Então quando nós executamos o comando `nuxt generate`:

```
[nuxt] Generating...
[...]
nuxt:render Rendering url / +154ms
nuxt:render Rendering url /users/1 +12ms
nuxt:render Rendering url /users/2 +33ms
nuxt:render Rendering url /users/3 +7ms
nuxt:generate Generate file: /index.html +21ms
nuxt:generate Generate file: /users/1/index.html +31ms
nuxt:generate Generate file: /users/2/index.html +15ms
nuxt:generate Generate file: /users/3/index.html +23ms
nuxt:generate HTML Files generated in 7.6s +6ms
[nuxt] Generate done
```

Genial, mas como se nós temos **parâmetros dinâmicos**?

1. Usa uma `Function` que retorna uma `Promise`.
2. Usa uma `Function` com um `callback(err, params)`.

Função que retorna uma Promessa

```
import axios from 'axios'

export default {
  generate: {
    routes() {
      return axios.get('https://my-api/users').then(res => {
        return res.data.map(user => {
          return '/users/' + user.id
        })
      })
    }
  }
}
```

Função com um callback

```
import axios from 'axios'

export default {
  generate: {
    routes(callback) {
      axios
        .get('https://my-api/users')
        .then(res => {
          const routes = res.data.map(user => {
            return '/users/' + user.id
          })
          callback(null, routes)
        })
        .catch(callback)
    }
  }
}
```

Acelerando a geração de rota dinâmica com payload

No exemplo acima, estamos usando o `user.id` do servidor para gerar as rotas mas descartando o resto dos dados. Normalmente, nós precisamos requisitar ele novamente a partir de dentro do `/users/_id.vue`. Enquanto nós podemos fazer isso, iremos provavelmente precisar definir a propriedade `generate.interval` para alguma coisa como `100` no sentido não sobrecarregar o servidor com chamadas. Porque isso aumenta o tempo de execução do roteiro `generate`, seria preferível passar ao longo do objeto `user` inteiro para o contexto dentro do `_id.vue`. Nós fazemos isso pela modificação do código acima disto:

```
import axios from 'axios'

export default {
  generate: {
    routes() {
      return axios.get('https://my-api/users').then(res => {
        return res.data.map(user => {
          return {
            route: '/users/' + user.id,
            payload: user
          }
        })
      })
    }
  }
}
```

Agora nós podemos acessar o `payload` do `/users/_id.vue` desse jeito:

```
async asyncData ({ params, error, payload }) {
  if (payload) return { user: payload }
  else return { user: await backend.fetchUser(params.id) }
}
```

A propriedade subFolders

- Tipo: `Boolean`

- Valor padrão: `true`

Por padrão, quando estiver executando o comando `nuxt generate`, o Nuxt criará um diretório para cada rota e servir um ficheiro `index.html`.

Exemplo:

```
-| dist/
---| index.html
---| about/
-----| index.html
---| products/
-----| item/
| index.html
```

Quando definida para `false`, os ficheiros HTML são gerados de acordo com o caminho da rota:

```
export default {
  generate: {
    subFolders: false
  }
}
```

```
-| dist/
---| index.html
---| about.html
---| products/
-----| item.html
```

[Go to TOC](#)

A propriedade globalName

O Nuxt permite você personalizar o identificador global usado dentro do modelo do HTML principal bem como o nome da instância principal do Vue e outras opções.

- Tipo: `String`
- Valor padrão: `nuxt`

```
{
  globalName: 'myCustomName'
}
```

Warning

A propriedade `globalName` precisa ser um identificador JavaScript, e modificar ele pode quebrar o suporte para certos plugins que dependem das funções nomeada do Nuxt. Se você estiver procurando apenas mudar o identificador `_nuxt` visível do HTML, então use a propriedade `globals`.

A propriedade globals

Personaliza nomes globais específicos os quais são baseados no `globalName` por padrão.

- Type: `Object`
- Default:

```
globals: {
  id: globalName => `__${globalName}`,
  nuxt: globalName => `__$${globalName}`,
  context: globalName => `__${globalName.toUpperCase()}__`,
  pluginPrefix: globalName => globalName,
  readyCallback: globalName => `on${_.capitalize(globalName)}Ready`,
  loadedCallback: globalName => `on${_.capitalize(globalName)}Loaded`
},
```

A propriedade head

O Nuxt permite você definir todos os atributos meta padrão para sua aplicação dentro do ficheiro `nuxt.- config.js`, usando a mesma propriedade `head`

- Tipo: `Object` ou `Function`

```
export default {
  head: {
    titleTemplate: '%s - Nuxt',
    meta: [
      { charset: 'utf-8' },
      { name: 'viewport', content: 'width=device-width, initial-scale=1' },

      // hid é usado como identificador único. Não use o `vmid` para isso visto
      que isso não funcionará
      { hid: 'description', name: 'description', content: 'Meta description' }
    ]
  }
}
```

Para conhecer a lista de opções que você pode atribuir ao `head`, consulte a [documentação do vue-meta](#).

Você pode também usar o `head` como uma função dentro de seus componentes para acessar os dados do componente através do `this` ([leia mais](#)).

Info

Para evitar marcação meta duplicadas quando for usada dentro de um componente filho, defina um identificador único com a chave `hid` para os seus elementos meta ([leia mais](#)).

A propriedade hooks

Gatilhos são [observadores para os eventos do Nuxt](#) que são normalmente são usados dentro dos módulos do Nuxt, porém estão disponíveis dentro do ficheiro `nuxt.config.js`.

- Tipo: `Object`

```
import fs from 'fs'
import path from 'path'

export default {
  hooks: {
    build: {
      done(builder) {
        const extraFilePath = path.join(
          builder.nuxt.options.buildDir,
          'extra-file'
        )
        fs.writeFileSync(extraFilePath, 'Something extra')
      }
    }
  }
}
```

Internamente, gatilhos seguem um padrão de nomeação que usa dois pontos (por exemplo, `build:done`). Para facilitação da configuração, você pode estruturar eles como um objeto hierárquico quando estiver usando um ficheiro `nuxt.config.js` (como exemplificado acima) para definir seus próprios gatilhos. Consulte o [Interior do Nuxt](#) para ter acesso a informações mais detalhadas de como eles funcionam.

Lista de Gatilhos

- [Gatilhos do Nuxt](#)
- [Gatilhos do Renderer](#)
- [Gatilhos do ModulesContainer](#)
- [Gatilhos do Builder](#)
- [Gatilhos do Generator](#)

Exemplos

Redirecionar para `router.base` quando não estiver na raiz

Vamos dizer que você quer servir páginas como `/portal` ao invés de `/`.

Isto pode ser um caso extremo, e a finalidade do `router.base` do `nuxt.config.js` é para quando um servidor web servir o Nuxt em algum lugar além da raiz do domínio.

Mas em ambiente de desenvolvimento local, acertar o `localhost`, quando o `router.base` não é `/` retorna um `404`. No sentido de prevenir isso, você pode definir um gatilho (Hook).

Talvez o redirecionamento não seja o melhor caso de uso para sítio da web em produção, mas isso ajudará você influenciar os gatilhos.

Para começar, você [pode mudar o `router.base`](#); Atualize o seu ficheiro `nuxt.config.js`:

```
import hooks from './hooks'
export default {
  router: {
    base: '/portal'
  }
  hooks: hooks(this)
}
```

Depois, crie alguns ficheiros;

1. `hooks/index.js`, módulo de gatilhos

```
import render from './render'

export default nuxtConfig => ({
  render: render(nuxtConfig)
})
```

2. `hooks/render.js`, gatilho render

```
import redirectRootToPortal from './route-redirect-portal'

export default nuxtConfig => {
  const router = Reflect.has(nuxtConfig, 'router') ? nuxtConfig.router : {}
  const base = Reflect.has(router, 'base') ? router.base : '/portal'

  return {
    /**
     * 'render:setupMiddleware'
     * {@link node_modules/nuxt/lib/core/renderer.js}
     */
    setupMiddleware(app) {
      app.use('/', redirectRootToPortal(base))
    }
  }
}
```

3. `hooks/route-redirect-portal.js`, O próprio intermédario

```
/**
 * Intermediário de gatilho do Nuxt para redirecionar de `/' para `/portal` (ou
 * para o que definimos no router.base do nuxt.config.js)
 *
 * Deve ser a mesma versão que o connect
 * {@link node_modules/connect/package.json}
 */
import parseurl from 'parseurl'

/**
 * Conecta o intermediário para manipular o redirecionamento para Raiz do
 * Contexto da Aplicação Web desejada.
 *
 * Note que a documentação do Nuxt carece de explicação de como usar gatilhos.
*/
```

```

* Isto é um roteador de exemplo para ajudar a explicar.
*
* Consulte boas implementações para obter inspiração:
* - https://github.com/nuxt/nuxt.js/blob/dev/examples/with-
cookies/plugins/cookies.js
* - https://github.com/yyx990803/launch-editor/blob/master/packages/launch-
editor-middleware/index.js
*
* [http_class_http_clientrequest]:
https://nodejs.org/api/http.html#http_class_http_clientrequest
* [http_class_http_serverresponse]:
https://nodejs.org/api/http.html#http_class_http_serverresponse
*
* @param {http.ClientRequest} req Node.js internal client request object
[http_class_http_clientrequest]
* @param {http.ServerResponse} res Node.js internal response
[http_class_http_serverresponse]
* @param {Function} next middleware callback
*/
export default desiredContextRoot =>
  function projectHooksRouteRedirectPortal(req, res, next) {
    const desiredContextRootRegExp = new RegExp(`^${desiredContextRoot}`)
    const _parsedUrl = Reflect.has(req, '_parsedUrl') ? req._parsedUrl : null
    const url = _parsedUrl !== null ? _parsedUrl : parseurl(req)
    const startsWithDesired = desiredContextRootRegExp.test(url.pathname)
    const isNotProperContextRoot = desiredContextRoot !== url.pathname
    if (isNotProperContextRoot && startsWithDesired === false) {
      const pathname = url.pathname === null ? '' : url.pathname
      const search = url.search === null ? '' : url.search
      const Location = desiredContextRoot + pathname + search
      res.writeHead(302, {
        Location
      })
      res.end()
    }
    next()
  }
}

```

Então, sempre que um colega em desenvolvimento acidentalmente bater em / para alcançar o desenvolvimento do serviço de desenvolvimento web, o Nuxt irá redirecionar automaticamente para /portal

A propriedade ignore

Define os ficheiros a serem ignorados pela sua aplicação Nuxt

O ficheiro `.nuxtignore`

Você pode usar um ficheiro `.nuxtignore` para deixar o Nuxt ignorar ficheiros de `layout` (esquema), `page` (página), `store` (memória) e `middleware` (intermediário) dentro do diretório raiz (`rootDir`) do seu projeto durante a fase de construção. O ficheiro `.nuxtignore` está sujeito a mesma especificação que os ficheiros `.gitignore` e o `.eslintignore`, no qual cada linha é padrão global indicando quais ficheiros devem ser ignorados.

Por exemplo:

```
# ignora o esquema foo.vue
layouts/foo.vue
# ignora os ficheiros de esquema os quais os nomes termina com -ignore.vue
layouts/*-ignore.vue

# ignora a página bar.vue
pages/bar.vue
# ignora a página dentro da pasta ignore
pages/ignore/*.vue

# ignora a memória baz.js
store/baz.js
# ignora os ficheiros de memória que correspondem a *.test.*
store/ignore/*.test.*

# ignora os ficheiros de intermediários dentro da raiz da pasta foo exceto bar.js
middleware/foo/*.js
!middleware/foo/bar.js
```

Mais detalhes sobre a especificação estão dentro da [documentação do gitignore](#)

A propriedade ignorePrefix

- Tipo: `String`
- Valor padrão: `'-'`

Qualquer ficheiro dentro de `pages/`, `layouts/`, `middleware/` ou `store/` será ignorado durante a construção se o seu nome de ficheiro começar com o prefixo especificado pelo `ignorePrefix`.

Por padrão todos ficheiros os quais começam com o `_` serão ignorados, ficheiros tais como `store/-foo.js` e `pages/-bar.vue`. Isto permite levar em consideração a co-localização de testes, utilitários, e componentes com seus chamadores sem eles mesmo serem convertidos em rotas, memórias, etc.

A propriedade ignore

- Tipo: `Array`
- Valor padrão: `['**/*.test.*']`

Mais personalizável do que o `ignorePrefix`: todos ficheiros correspondentes ao padrão global especificado dentro do `ignore` será ignorado na construção.

ignoreOptions

O `nuxtignore` está usando o `node-ignore` nos bastidores, o `ignoreOptions` pode ser configurado como `options` de `node-ignore`.

```
export default {
  ignoreOptions: {
    ignorecase: false
  }
}
```

A propriedade loading

O Nuxt usa o seu próprio componente para mostrar uma barra de progresso entre as rotas. Você pode personalizar ela, desativar ela ou criar o seu próprio componente.

- Tipo: Boolean ou Object ou String

```
export default {
  mounted() {
    this.$nextTick(() => {
      this.$nuxt.$loading.start()

      setTimeout(() => this.$nuxt.$loading.finish(), 500)
    })
  }
}
```

Desativar a Barra de Progresso

- Tipo: Boolean

```
export default {  
  loading: false  
}
```

Personalizado a Barra de Progresso

- Tipo: Object

```
export default {  
  loading: {  
    color: 'blue',  
    height: '5px'  
  }  
}
```

Lista de propriedades para personalizar a barra de progresso.

| Chave | Tipo | Valor Padrão | Descrição | || | `color` | String | `'black'` | A cor de css da barra de progresso | | `failedColor` | String | `'red'` | A cor de css da barra de progresso quando um erro é acrescentado enquanto estiver renderizando a rota (se por exemplo o `data` ou `fetch` enviou um erro). | | `height` | String | `'2px'` | Altura da barra de progresso (usada dentro da propriedade `style` da barra de progresso) | | `throttle` | Number | `200` | Em milissegundos, espera pelo tempo especificado antes da exibição da barra de progresso. Útil para prevenir a barra de cintilar. | | `duration` | Number | `5000` | Em milissegundos, a duração máxima da barra de progresso, o Nuxt assume que a rota será renderizada antes de 5 segundos. | | `continuous` | Boolean | `false` | Mantém animando a barra de progresso quando o carregamento demorar mais do que a `duration` (duração) . | | `css` | Boolean | `true` | Defina para `false` para remover os estilos padrão da barra de progresso (e adiciona o seu próprio estilo) | | `rtl` | Boolean | `false` | Define a direção da barra de progresso, da direita para esquerda. |

Usando um Componente de Carregamento Personalizado

- Tipo: `String`

O seu componente tem que expor alguns desses métodos:

| Método | Obrigatório | Descrição | | | `start()` | Obrigatório | Chamado quando uma rota muda, isto é, onde você exibe o seu componente. | | `finish()` | Obrigatório | Chamado quando uma rota é carregada (e os dados requisitados), isto é, onde você esconde o seu componente. | | `fail(error)` | *Opcional* | Chamado quando uma rota não pode ser carregada (falhou em requisitar os dados por exemplo). | | `increase(num)` | *Opcional* | Chamado durante o carregamento do componente de rota, `num` é um Inteiro maior do que 100. |

```
<template lang="html">
  <div class="loading-page" v-if="loading">
    <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>
```

```
export default {
  loading: '~/components/loading.vue'
}
```

[Go to TOC](#)

A propriedade loadingIndicator

Mostra um indicador de carregamento elegante enquanto a página estiver carregando!

Sem a Renderização no Lado do Servidor (quando a opção `ssr` for `false`), não há conteúdo do lado do servidor no primeiro carregamento da página. Então, ao invés de mostrar uma página em branco enquanto a página carrega, nós podemos mostrar um rodopiador.

Esta propriedade pode ter três diferentes tipos: que podem ser `string` ou `boolean` ou `object`. Se um valor de sequência de caracteres for fornecido ele será convertido para estilo de objeto.

O valor padrão é:

```
loadingIndicator: {
  name: 'circle',
  color: '#3B8070',
  background: 'white'
}
```

Indicadores embutidos

Esses indicadores são importados do incrível projeto do [SpinKit](#). Você pode usar sua página de demonstração para pré-visualizar os rodopiadores.

- circle
- cube-grid
- fading-circle
- folding-cube
- chasing-dots
- nuxt
- pulse
- rectangle-bounce
- rotating-plane
- three-bounce
- wandering-cubes

Os indicadores embutidos suportam as opções `color` e `background`.

Indicadores personalizados

Se você precisar do seu próprio indicador especial, um valor de sequência de caracteres ou nome de chave pode também ser um caminho para um modelo HTML do código-fonte do indicador! Todas opções são passadas para o modelo também.

O [código-fonte](#) do componente de carregamento embutido do Nuxt está disponível se você precisar de uma base!

[Go to TOC](#)

A propriedade mode

Muda o modo padrão do Nuxt

- Tipo: `string`
 - Valor padrão: `universal`
 - Valores possíveis:
 - `'spa'` : Sem renderização no lado do servidor (apenas navegação no lado do cliente)
 - `'universal'` : Aplicação isomórfica (renderização no lado do servidor + navegação no lado do cliente)

Você pode usar esta opção para mudar o modo padrão do Nuxt para o seu projeto usando o ficheiro `nuxt.config.js`

Warning

Depreciado: use o `ssr: false` no lugar de `mode: spa`

Next

Para saber mais sobre a opção `ssr`, consulte a [propriedade ssr](#).

Next

Para saber mais sobre a opção `mode`, consulte a [seção de modos de renderização](#).

[Go to TOC](#)

A propriedade modern

Constrói e serve um pacote moderno

Esta funcionalidade é inspirada pelo [modo moderno do vue-cli](#)

- Tipo: `String` ou `Boolean`
 - Valor padrão: `false`
 - Valores possíveis:
 - `'client'` : Serve ambos pacotes de roteiros, o pacote moderno `<script type="module">` e o pacote legado `<script nomodule>`, também fornece um `<link rel="modulepreload">` para o pacote moderno. Todo navegador que o tipo `module` carregará o pacote moderno enquanto navegadores antigos recuarão para o legado (transpilado).
 - `'server'` ou `true` : O servidor Node.js consultará a versão do navegador baseado no agente de usuário e servirá o pacote moderno ou legado correspondente.
 - `false` : Desativa a construção moderna

As duas versões de pacotes são:

1. Pacote moderno: apontando navegadores modernos que suportem os módulos do EcmaScript
2. Pacote legado: apontando navegadores antigos baseado na configuração do Babel (compatível com IE9 por padrão).

Informação:

- Use a opção de comando `--modern | -m=[mode]` para construir/iniciar pacotes modernos:

```
{
  "scripts": {
    "build:modern": "nuxt build --modern=server",
    "start:modern": "nuxt start --modern=server"
  }
}
```

Nota sobre comando `nuxt generate`: A propriedade `modern` também funciona com o comando `nuxt generate`, mas neste caso apenas a opção `client` é honrada e será selecionada automaticamente quando executar o comando `nuxt generate --modern` sem fornecer nenhum valor.

- O Nuxt detetará automaticamente a construção `modern` no `nuxt start` quando o `modern` não for especificado, o modo de deteção automática é:

ssr
true

ssr

false

- O modo moderno para o comando `nuxt generate` apenas pode ser `client`
- Use `render.crossorigin` para definir o atributo `crossorigin` dentro dos marcadores `<link>` e `<script>`

Recorra a [exelente publicação do Phillip Walton](#) para mais conhecimentos que dizem respeito as construções modernas.

A propriedade build

O Nuxt permite você personalizar a configuração do webpack para construir a sua aplicação web como você quiser.

A propriedade analyze

O Nuxt usa o [webpack-bundle-analyzer](#) para permitir você visualizar seus pacotes e como otimizar eles.

- Tipo: `Boolean` ou `Object`
- Padrão: `false`

Se for um objeto, veja as propriedades disponíveis [aqui](#).

```
export default {
  build: {
    analyze: true,
    // ou
    analyze: {
      analyzerMode: 'static'
    }
  }
}
```

Info

Informação: você pode usar o comando `yarn nuxt build --analyze` ou `yarn nuxt build -a` para construir a sua aplicação e lançar o analisador de pacote sobre o <http://localhost:8888>. Se você não estiver usando `yarn` você pode executar o comando com `npx`.

A propriedade corejs

Desde o [Nuxt@2.14](#) o Nuxt automaticamente deteta a versão atual do `core-js` dentro do seu projeto, você também pode especificar qual versão você deseja usar.

- Tipo: `number` | `string` (os valores válidos são `'auto'`, `2` e `3`)
- Padrão: `'auto'`

A propriedade babel

Personalize a configuração do Babel para os ficheiros JavaScript e Vue. `.babelrc` é ignorado por padrão.

- Tipo: `Object`
- Consulte as [opções](#) do `babel-loader` e as [opções](#) do `babel`
- Padrão:

```
{
  babelrc: false,
  cacheDirectory: undefined,
  presets: ['@nuxt/babel-preset-app']
}
```

A alvos padrão do `@nuxt/babel-preset-app` são: `ie: '9'` na construção do `client`, e `node: 'current'` na construção do `server`.

O método presets

- Tipo: `Function`
- Argumento:
 1. `Object` : `isServer: true`
`false`
 2. `Array` :
 - nome prédefinido `@nuxt/babel-preset-app`
 - [opções](#) do `@nuxt/babel-preset-app`

Nota:: O `presets` configurado dentro do `build.babel.presets` será aplicado a ambas, construção cliente e a do servidor. O alvo será definido pelo Nuxt de acordo com o (cliente/servidor). Se você quiser configurar o `preset` diferentemente para a construção do cliente ou do servidor, use o `presets` como uma função:

Nós **recomendamos fortemente** a usar o `preset` padrão ao invés da personalização abaixo

```
export default {
  build: {
    babel: {
      presets({ isServer }, [ preset, options ]) {
        // muda as opções diretamente
        options.targets = isServer ? ... : ...
        options.corejs = ...
        // não retorna nada
      }
    }
  }
}
```

```

        }
    }
}
```

Ou sobrescreva o valor padrão ao retornar a lista de `presets` inteira:

```

export default {
  build: {
    babel: {
      presets({ isServer }, [preset, options]) {
        return [
          [
            preset,
            {
              targets: isServer ? ... : ...,
              ...options
            }
          ],
          [
            // Outras predefinições
          ]
        ]
      }
    }
  }
}
```

A propriedade cache

- Tipo: `Boolean`
- Padrão: `false`
- Δ Experimental

Ativa o cache do `terser-webpack-plugin` e do `cache-loader`

A propriedade cssSourceMap

- Tipo: `boolean`
- Padrão: `true` para o desenvolvimento e `false` para a produção.

Ativa o suporte ao CSS Source Map (Mapa da Fonte do CSS)

A propriedade devMiddleware

- Tipo: `Object`

Consulte o `webpack-dev-middleware` para conhecer as opções disponíveis.

A propriedade devtools

- Tipo: `boolean`
- Padrão: `false`

Configura-se para permitir inspeção do [vue-devtools](#).

Se você já a ativou através do ficheiro `nuxt.config.js` ou de outra maneira, a ferramenta de desenvolvimento ativa apesar da bandeira.

O método extend

Estende a configuração do webpack manualmente para os pacotes do cliente e do servidor.

- Tipo: `Function`

O `extend` é chamado duas vezes, uma vez para o pacote do servidor, e uma vez para o pacote do cliente. Os argumentos do método são:

1. O objeto de configuração do Webpack,
2. Um objeto com as seguintes chaves (todas booleanas exceto `loaders`): `isDev`, `isClient`, `isServer`, `loaders`.

Warning

Aviso: As chaves `isClient` e `isServer` fornecidas são separadas das chaves disponíveis dentro do `contexto`. Elas **não** estão depreciadas. Não use `process.client` e `process.server` aqui neste ponto quando eles forem `undefined`.

```
export default {
  build: {
    extend(config, { isClient }) {
      // Extend only webpack config for client-bundle
      if (isClient) {
        config.devtool = 'source-map'
      }
    }
  }
}
```

Se você quiser ver mais sobre nossa configuração padrão do webpack, dê uma vista de olhos no nosso [diretório do webpack](#).

O objeto loaders dentro do extend

O `loaders` tem a mesma estrutura de objeto que o `build.loaders`, assim você pode mudar as opções de `loaders` dentro do `extend`.

```
export default {
  build: {
    extend(config, { isClient, loaders: { vue } }) {
      // Estende a configuração do webpack somente para o pacote do cliente
      if (isClient) {
        vue.transformAssetUrls.video = ['src', 'poster']
      }
    }
  }
}
```

A propriedade extractCSS

Ativa a Common CSS Extraction (Extração Comum do CSS) usando as [orientações](#) do Vue Server Renderer (Servidor Renderizador do Vue).

- Tipo: `Boolean` ou `Object`
- Padrão: `false`

Ao usar o `extract-css-chunks-webpack-plugin` nos bastidores, todo o seu CSS será extraído em ficheiros separados, normalmente um por componente. Isto permite o cacheamento do seu CSS e JavaScript separadamente e vale a pena tentar no caso de você ter um grande volume ou CSS partilhado.

Exemplo (`nuxt.config.js`):

```
export default {
  build: {
    extractCSS: true,
    // ou
    extractCSS: {
      ignoreOrder: true
    }
  }
}
```

Info

Nota: Havia um bug antes do Vue 2.5.18 que removia a importação de CSS crítico quando usada esta opção.

Você talvez queira extrair todo o seu CSS em ficheiro único. Existe uma maneira de dar a volta a isto:

Warning

Não é recomendado extrair tudo em um ficheiro único. A extração em vários ficheiros CSS é melhor para o cacheamento e para isolação do precarregamento. Isto pode também melhorar o desempenho da página ao descarregar e resolver somente aqueles recursos que são necessários.

```
export default {
  build: {
    extractCSS: true,
    optimization: {
      runtimeChunk: false
    }
  }
}
```

```

    splitChunks: {
      cacheGroups: {
        styles: {
          name: 'styles',
          test: /\.css$/|\.vue$/,
          chunks: 'all',
          enforce: true
        }
      }
    }
  }
}

```

A propriedade filenames

Personaliza os nomes de ficheiro de pacote.

- Tipo: `Object`
- Padrão:

```

{
  app: ({ isDev, isModern }) => isDev ? `[_name]${isModern ? '.modern' : ''}.js` :
    `[_contenthash:7]${isModern ? '.modern' : ''}.js`,
  chunk: ({ isDev, isModern }) => isDev ? `[_name]${isModern ? '.modern' : ''}.js` :
    `[_contenthash:7]${isModern ? '.modern' : ''}.js`,
  css: ({ isDev }) => isDev ? '[name].css' : 'css/[contenthash:7].css',
  img: ({ isDev }) => isDev ? '[path][name].[ext]' : 'img/[name].
[_contenthash:7].[ext]',
  font: ({ isDev }) => isDev ? '[path][name].[ext]' : 'fonts/[name].
[_contenthash:7].[ext]',
  video: ({ isDev }) => isDev ? '[path][name].[ext]' : 'videos/[name].
[_contenthash:7].[ext]'
}

```

Este exemplo muda pedaços agradáveis de nomes para identificadores numéricos:

```

export default {
  build: {
    filenames: {
      chunk: ({ isDev }) => (isDev ? '[name].js' : '[id].[contenthash].js')
    }
  }
}

```

Para entender um pouco mais sobre o uso de manifestos, dê uma vista de olhos na [documentação do webpack](#).

Warning

Seja cuidadoso quando estiver usando nomes de ficheiros que não são baseados em hash em produção assim a maioria dos navegadores cachearão o recurso e não detectarão as mudanças no primeiro carregamento.

A propriedade friendlyErrors

- Tipo: `Boolean`
- Padrão: `true` (cobertura ativada)

Ativa ou desativa a cobertura fornecida pelo [FriendlyErrorsWebpackPlugin](#)

A propriedade hardSource

- Tipo: `Boolean`
- Padrão: `false`
- Δ Experimental

Ativa o [HardSourceWebpackPlugin](#) para melhoramento do cache

A propriedade hotMiddleware

- Tipo: `Object`

Consulte o [webpack-hot-middleware](#) para saber as opções disponíveis

A propriedade html.minify

- Tipo: `Object`
- Padrão:

```
{
  collapseBooleanAttributes: true,
  decodeEntities: true,
  minifyCSS: true,
  minifyJS: true,
  processConditionalComments: true,
  removeEmptyAttributes: true,
  removeRedundantAttributes: true,
  trimCustomFragments: true,
  useShortDoctype: true
}
```

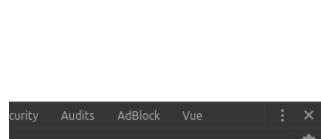
Atenção: Se você fizer mudanças ao `html.minify`, elas não serão combinadas com os padrões!

Configuração para o plugin [html-minifier](#) usada para minificar os ficheiros HTML criados durante o processo de construção (serão aplicados a *todos os modos*).

A propriedade indicator

Mostra o indicador de construção para o substituição forte de módulo em desenvolvimento (disponível nas versões `v2.8.0+`)

- Tipo: `Boolean`
- Padrão: `true`



nuxt-build-indicator

A propriedade loaders

Personaliza as opções do Nuxt integradas aos loaders (carregadores) do webpack.

- Tipo: `Object`
- Padrão:

```
{
  file: {},
  fontUrl: { limit: 1000 },
  imgUrl: { limit: 1000 },
  pugPlain: {},
  vue: {
    transformAssetUrls: {
      video: 'src',
      source: 'src',
      object: 'src',
      embed: 'src'
    },
    css: {},
    cssModules: {
      localIdentName: '[local]_[hash:base64:5]'
    },
    less: {},
    sass: {
      indentedSyntax: true
    },
    scss: {},
    stylus: {},
    vueStyle: {}
  }
}
```

Nota: Além de especificar a configuração dentro do ficheiro `nuxt.config.js`, ele pode também ser modificado pelo `build.extend`

A propriedade `loaders.file`

Mais detalhes estão dentro das [opções do file-loader](#).

As subpropriedades `loaders.fontUrl` e `loaders imgUrl`

More details are in [url-loader options](#). Mais detalhes estão dentro das [opções do url-loader](#).

A propriedade `loaders.pugPlain`

Mais detalhes estão dentro do [pug-plain-loader](#) e das [opções do compilador do Pug](#).

A propriedade `loaders.vue`

Mais detalhes estão dentro das [opções do vue-loader](#).

As subpropriedades `loaders.css` e `loaders.cssModules`

Mais detalhes estão dentro das [opções do css-loader](#). O `cssModules` é a opção do carregador para o uso de [Módulos do CSS](#)

A propriedade `loaders.less`

Você pode passar quaisqueres opções específicas do Less para o `less-loader` via `loaders.less`. Consulte a [documentação do Less](#) para saber todas opções disponíveis dentro do dash-case

As subpropriedades `loaders.sass` e `loaders.scss`

Consulte a [documentação do Sass](#) para saber todas opções disponíveis para o Sass. Nota: O `loader s.sass` é para [Sintaxe Indentada do Sass](#)

A propriedade loaders.vueStyle

Mais detalhes estão dentro das [opções do vue-style-loader](#).

A propriedade optimization

- Tipo: `Object`

- Padrão:

```
{
  minimize: true,
  minimizer: [
    // terser-webpack-plugin
    // optimize-css-assets-webpack-plugin
  ],
  splitChunks: {
    chunks: 'all',
    automaticNameDelimiter: '.',
    name: undefined,
    cacheGroups: {}
  }
}
```

O valor padrão do `splitChunks.name` é `true` no modo `dev` (desenvolvimento) ou no modo `analyze` (análise).

Você pode definir `minimizer` para um `Array` personalizado de plugins ou definir `minimize` para `false` para desativar todos minimizadores. (`minimize` está desativado para desativado para desenvolvimento por padrão)

Consulte a seção de [Otimização do Webpack](#).

A propriedade optimizeCSS

- Tipo: `Object` ou `Boolean`

- Padrão:

- `false`
- `{}` quando o `extractCSS` estiver ativado

Opções do plugin `OptimizeCSSAssets`.

Consulte o [NMFR/optimize-css-assets-webpack-plugin](#).

A propriedade parallel

- Tipo: `Boolean`
- Padrão: `false`
- Δ Experimental

Ativa o [thread-loader](#) dentro da construção do webpack

A propriedade plugins

Adiciona plugins do webpack

- Tipo: `Array`
- Padrão: `[]`

```
import webpack from 'webpack'
import { version } from './package.json'
export default {
  build: {
    plugins: [
      new webpack.DefinePlugin({
        'process.VERSION': version
      })
    ]
  }
}
```

A propriedade postcss

Personaliza os plugins do [Carregador do PostCSS](#) plugins.

- Tipo: `Array` (legado, sobrescreverá os padrões), `Object` (recomendado), `Function` ou `Boolean`

Nota: O Nuxt tem aplicado a [pré-definição de ambiente do PostCSS](#). Por padrão ele ativa o [estágio de 2 funcionalidades](#) e o [Autoprefixer](#), você pode usar o `build.postcss.preset` para configurar ele.

- Padrão:

```
{
  plugins: {
    'postcss-import': {},
    'postcss-url': {},
    'postcss-preset-env': this.preset,
    'cssnano': { preset: 'default' } // desativado no modo de desenvolvimento
  },
  order: 'presetEnvAndCssnanoLast',
  preset: {
    stage: 2
  }
}
```

As suas definições personalizadas de plugin serão combinadas com os plugins padrão (a menos que você esteja usando um `Array` ao invés de um `Object`).

```
export default {
  build: {
    postcss: {
      plugins: {
        // Desativa `postcss-url`
        'postcss-url': false,
        // Adiciona alguns plugins
        'postcss-nested': {},
        'postcss-responsive-type': {},
        'postcss-hexrgba': {}
      },
      preset: {
        autoprefixer: {
          grid: true
        }
      }
    }
  }
}
```

Se a configuração do PostCSS é um `Object`, o `order` pode ser usado para definição da ordem do plugin:

- Tipo: `Array` (ordenado por nomes de plugin), `String` (ordena a pré-definição de nome), `Function`
- Padrão: `cssnanoLast` (coloca `cssnano` em último)

```
export default {
  build: {
    postcss: {
      // pré-definição do nome
      order: 'cssnanoLast',
      // ordenado por nomes de plugin
      order: ['postcss-import', 'postcss-preset-env', 'cssnano']
      // função para calcular a ordem do plugin
      order: (names, presets) => presets.cssnanoLast(names)
    }
  }
}
```

A propriedade `plugins` do `postcss` e o módulo `@nuxtjs/tailwindcss`

Se você quiser aplicar um plugin de PostCSS (por exemplo, `postcss-pxtorem`) na configuração do módulo `@nuxtjs/tailwindcss`, você tem de mudar a ordem e carregar o `tailwindcss` primeiro.

Esta configuração não tem impacto sobre o `nuxt-purgecss`.

```
import { join } from 'path'

export default {
  // ...
  build: {
    postcss: {
      plugins: {
        tailwindcss: join(__dirname, 'tailwind.config.js'),
        'postcss-pxtorem': {
          propList: ['*', '!border*']
        }
      }
    }
  }
}
```

```

    }
}
}
```

A propriedade profile

- Tipo: `Boolean`
- Padrão: ativado pelo argumento de linha de comando `--profile`

Ativa o perfilador dentro do [WebpackBar](#)

A propriedade publicPath

O Nuxt deixa você carregar os seus ficheiros de distribuição para o seu CDN para o máximo desempenho, simplesmente defina o `PublicPath` para o seu CDN.

- Tipo: `String`
- Padrão: `'/_nuxt/'`

```

export default {
  build: {
    publicPath: 'https://cdn.nuxtjs.org'
  }
}
```

Depois, quando estiver executando o `nuxt build`, carrega o conteúdo do diretório `.nuxt/dist/client` para a sua CDN e voilà!.

No Nuxt 2.15+, mudando o valor desta propriedade em tempo de execução sobrescreverá a configuração de uma aplicação que já tem sido construída.

A propriedade quiet

Suprime a maioria do registo de saída da construção

- Tipo: `Boolean`
- Padrão: Ativado quando uma `CI` ou ambiente de `test` é detetado pelo `std-env`

A propriedade splitChunks

- Tipo: `Object`

- Padrão:

```
export default {
  build: {
    splitChunks: {
      layouts: false,
      pages: true,
      commons: true
    }
  }
}
```

Se criar ou não pedaços separados para o `layouts`, `pages` e `commons` (bibliotecas comuns: vue|vue-loader|vue-router|vuex...) Para mais informações, consulte a [documentação do webpack](#).

A propriedade ssr

Cria um pacote especial do webpack para o renderizador de SSR.

- Tipo: `Boolean`
- Padrão: `true` para o modo universal e `false` para o modo de aplicação de página única

Esta opção é automaticamente definida com base no valor de `mode` se não for fornecida.

A propriedade standalone

Dependências em linha do pacote do servidor (avançado)

- Tipo: `Boolean`
- Padrão: `false`

Este mode empacota o `node_modules` que é normalmente preservado como externo dentro da construção de servidor ([mais informações](#)).

Warning

Dependências do tempo de execução (módulos, `nuxt.config`, intermediário do servidor e diretório estático (static)) não são empacotados. Esta funcionalidade desativa o uso do `webpack-externals` para o pacote de servidor (server-bundle).

Info

Você pode usar o comando `yarn nuxt build --standalone` para desativar este modo na linha de comando. (Se você não está usando o `yarn` você pode executar o comando com o `npx`.)

A propriedade styleResources

- Tipo: `Object`
- Padrão: `{}`

Warning

Esta propriedade está depreciada. Ao invés desta use o [style-resources-module](#) para melhorado desempenho e melhor experiência do desenvolvedor!

Isto é útil quando você precisa injetar algumas variáveis e mixins dentro das suas páginas sem ter que importar elas a todo momento.

O Nuxt usa o [style-resources-loader](#) para alcançar este comportamento.

Você precisa especificar os padrões/caminhos que você quer incluir para os dados pré-processadores:

`less`, `sass`, `scss` ou `stylus`

Você não pode usar os apelidos de caminho aqui (`~` e `@`), você precisa usar caminhos relativos e caminhos absolutos.

```
{
  build: {
    styleResources: {
      scss: './assets/variables.scss',
      less: './assets/*.less',
      // sass: ....,
      // scss: ...
      options: {
        // Consulte o https://github.com/yenshih/style-resources-loader#options
        // Exceto a propriedade `patterns`
      }
    }
  }
}
```

A propriedade templates

O Nuxt permite você fornecer os seus próprios modelos os quais serão renderizados com base na configuração do Nuxt. Esta funcionalidade é especialmente útil para uso com [módulos](#).

- Tipo: `Array`

```
export default {
  build: {
    templates: [
      {
        src: '~/modules/support/plugin.js', // `src` pode ser absoluto ou relativo
        dst: 'support.js', // `dst` é relativo ao diretório `.nuxt` do projeto
        options: {
          // Opções são fornecidas ao modelo como chave `options`
          live_chat: false
        }
      }
    ]
  }
}
```

```

        }
    ]
}
}
```

Os modelos são renderizados usando o `lodash.template` que você pode aprender sobre como usar eles [aqui](#).

A propriedade `terser`

- Tipo: `Object` or `Boolean`
- Padrão:

```
{
  parallel: true,
  cache: false,
  sourceMap: false,
  extractComments: {
    filename: 'LICENSES'
  },
  terserOptions: {
    output: {
      comments: /[^*!*!@preserve!@license!@cc_on/(
    }
  }
}
```

Opções do plugin Terser. Defina para `false` para desativar este plugin.

A ativação do `sourceMap` deixará o comentário de ligação `//# sourceMappingURL` no final de cada ficheiro de saída se o webpack `config.devtool` estiver definido para `source-map`.

Consulte o [webpack-contrib/terser-webpack-plugin](#).

A propriedade `transpile`

- Tipo: `Array<String | RegExp | Function>`
- Padrão: `[]`

Se você quiser transpilar dependências específicas com Babel, você pode adicionar elas dentro do `build.transpile`. Cada item dentro da propriedade `transpile` pode ser um nome de pacote, uma sequência de caracteres e objeto de expressões regulares correspondem ao nome do ficheiro da dependência.

A partir da versão `2.9.0`, você pode também usar uma função para transpilar condicionalmente, a função receberá um objeto (`{ isDev, isServer, isClient, isModern, isLegacy }`):

```
{
  build: {
    transpile: [({ isLegacy }) => isLegacy && 'ky']
  }
}
```

Neste exemplo, o `ky` será transpilado pelo Babel se o Nuxt não estiver no [modo moderno](#).

A propriedade vueLoader

Nota: Esta configuração tem sido removida desde a versão 2.0 do Nuxt, ao invés desta use o `build.loaders.vue`.

- Tipo: `Object`

- Padrão:

```
{
  productionMode: !this.options.dev,
  transformAssetUrls: {
    video: 'src',
    source: 'src',
    object: 'src',
    embed: 'src'
  }
}
```

Especifica as [Opções do Carregador do Vue](#).

A propriedade watch

Você pode fornecer os seus ficheiros personalizados para observar e regerar depois de mudanças. Esta funcionalidade é especialmente útil para ser usada com [módulos](#).

- Tipo: `Array<String>`

```
export default {
  build: {
    watch: ['~/.nuxt/support.js']
  }
}
```

A propriedade followSymlinks

Por padrão, o processo de construção não examinar os ficheiros dentro de ligações simbólicas (symlinks). Este booleano inclui elas, permitindo assim o uso das ligações simbólicas dentro de pastas tais como a pasta "pages", por exemplo.

- **Tipo:** Boolean

```
export default {
  build: {
    followSymlinks: true
  }
}
```

A propriedade modules

Módulos são extensões do Nuxt que podem estender seu núcleo de funcionalidades e adicionar integrações intermináveis. [Saiba mais](#)

- Tipo: `Array`

Exemplo (`nuxt.config.js`):

```
export default {
  modules: [
    // Usando o nome do pacote
    '@nuxtjs/axios',

    // Relativo ao `srcDir` do seu projeto
    '~/modules/awesome.js',

    // Fornecendo opções
    ['@nuxtjs/google-analytics', { ua: 'X1234567' }],

    // Definição em linha
    function () {}
  ]
}
```

Desenvolvedores de módulo normalmente fornecem adicionais passos necessários e detalhes para uso.

O Nuxt tenta resolver cada item dentro do arranjo de módulos usando o caminho exigido do node (dentro do `node_modules`) e depois será resolvido a partir do `srcDir` do projeto se o apelido `~` for usado. Os módulos são executados sequencialmente então a ordem é importante.

Nota: Qualquer plugin injetado pelos módulos são adicionados ao **inicio** da lista de plugins. As suas opções são para:

- Manualmente adicionar o seu plugin no final da lista de plugins
`(this.nuxt.options.plugins.push(...))`
- Reverter a ordem dos módulos se um depender de um outro.

Os módulos devem exportar uma função para melhorar a construção/tempo de execução do Nuxt e opcionalmente retornar uma promessa até o trabalho deles estiver terminado. Nota que eles são obrigatórios no tempo de execução então devem já estar transpilados se dependerem de funcionalidades modernas do EcmaScript 6.

Consulte o [Guia dos Módulos](#) para mais informações detalhadas sobre como eles funcionam ou se estiver interessado em desenvolver o seu próprio módulo. Nós também fornecemos uma secção de [módulos](#) oficiais listando milhares de módulos feitos pela comunidade do Nuxt prontos para produção.

A propriedade buildModules

Info

Esta funcionalidade está disponível desde a versão 2.9 do Nuxt

Alguns módulos apenas são exigidos durante o tempo de desenvolvimento e construção. Usar o `buildModules` ajuda tornar o início em produção mais rápido e também diminuir significativamente o tamanho da pasta `node_modules` para desdobramentos em produção. Recorra a documentação de cada módulo para consultar se é recomendado usar `modules` ou `buildModules`.

A diferença de uso é:

- Ao invés de adicionar ao `modules` dentro do ficheiro `nuxt.config.js`, usa o `buildModules`
- Ao invés de adicionar ao `dependencies` dentro do `package.json`, usa o `devDependencies` (`yarn add --dev` ou `npm install --save-dev`)

[Go to TOC](#)

A propriedade modulesDir

Define o diretório dos módulos para sua aplicação Nuxt

- Tipo: [Array](#)
- Valor padrão: `['node_modules']`

Usada para definir os diretórios dos módulos para resolução de caminho, por exemplo: `resolveLoadi`
`ng`, `nodeExternals` e o `postcss` do Webpack. O caminho da configuração é relativo ao [options.ro](#)
`otDir` (valor padrão: `process.cwd()`).

```
export default {  
  modulesDir: ['../../node_modules']  
}
```

A definição deste campo pode ser necessária se o seu projeto estiver organizado como um espaço de trabalho estilizado de um repositório Yarn monólito.

A propriedade plugins

Usa os plugins de vue.js com as opções de plugins do Nuxt.

Nota: Desde a versão 2.4 do Nuxt, o `mode` tem sido introduzido como a opção de `plugins` para especificar o tipo do plugin, os possíveis valores são: `client` ou `server`. O `ssr: false` será adotado para o `mode: client` e depreciado no próximo principal lançamento.

- Tipo: `Array`
 - Itens: `String` ou `Object`

Se o item for um objeto, as propriedades são:

- `src: String` (caminho do ficheiro)
- `mode: String` (pode ser `client` ou `server`) *Se definida, o ficheiro será incluído apenas no respetivo lado do (cliente ou servidor).*

Nota: Versão antiga

- Tipo: `Array`
 - Itens: `String` ou `Object`

Se o item for um objeto, as propriedades são:

- `src: String` (caminho do ficheiro)
- `ssr: Boolean` (padrão ao `true`) *Se for `false`, o ficheiro será incluído apenas no lado do cliente.*

A propriedade `plugins` permite você adicionar plugins de Vue.js facilmente à sua aplicação principal.

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

```
export default {
  plugins: ['~/plugins/ant-design-vue']
}
```

```
import Vue from 'vue'
import Antd from 'ant-design-vue'
import 'ant-design-vue/dist/antd.css' // conforme documentação do Ant Design
Vue.use(Antd)
```

Nota que o CSS foi [importado conforme a documentação do Ant Design](#)

Todos os caminhos definidos dentro da propriedade `plugins` serão **importados** antes da inicialização da aplicação principal.

A propriedade render

O Nuxt permite você personalizar as opções do tempo de execução para renderização de páginas

A propriedade bundleRenderer

- Tipo: `Object`

Use esta opção para personalizar pacote renderizador da Renderização no Lado do Servidor do Vue. Esta opção é ignorada se houver definido `ssr: false`.

```
export default {
  render: {
    bundleRenderer: {
      directives: {
        custom1(el, dir) {
          // alguma coisa...
        }
      }
    }
}
```

Saiba mais sobre as opções disponíveis na [Referência da API de Renderização no Lado do Servidor do Vue](#). É recomendado não usar esta opção visto que o Nuxt já está fornecendo os melhores padrões de renderização no lado do servidor e uma configuração errada pode levar a problemas na Renderização no Lado do Servidor.

A propriedade etag

- Tipo: `Object`
 - Valor padrão: `{ weak: true }`

Para desativar o `etag` para páginas defina `etag: false`

Consulte a documentação do `etag` para conhecer as possíveis opções.

Você pode usar a sua própria função `hash` ao especificar `etag.hash`:

```
import { murmurHash128 } from 'murmurhash-native'

export default {
  render: {
    etag: {
      hash: html => murmurHash128(html)
    }
  }
}
```

Neste caso usamos o `murmurhash-native`, o qual é mais rápido para tamanhos de corpo de HTML muito maiores. Repare que a opção `weak` é ignorada, quando estiver especificando a sua própria função de `hash`.

A propriedade compressor

- Tipo `Object`
 - Valor padrão: `{ threshold: 0 }`

Quando estiver fornecendo um objeto, o intermediário do `compression` será usado (com as respetivas opções).

Se você quiser usar o seu próprio intermediário de compressão, você pode referenciar ele diretamente (por exemplo, `otherComp({ myOptions: 'example' })`).

Para desativar a compressão, use `compressor: false`.

A propriedade fallback

- Tipo `Object`
 - Valor padrão: `{ dist: {}, static: { skipUnknown: true } }`
 - a chave `dist` é para rotas correspondentes ao `publicPath` (por exemplo: `/_nuxt/*`)
 - a chave `static` é para rotas correspondentes ao `/*`

Os valores `dist` e `static` são encaminhados para o intermediário `serve-placeholder`.

Se você quiser desativar um deles ou ambos, você pode passar um valor de falsidade.

Exemplo permitindo a extensão `.js` para roteamento (por exemplo: `/repos/nuxt.js`):

```
export default {
  render: {
    fallback: {
      static: {
        // Evite enviar o 404 para estas extensões
        handlers: {
          '.js': false
        }
      }
    }
  }
}
```

A propriedade http2

- Tipo `Object`
 - Valor padrão: `{ push: false, pushAssets: null }`

Ativa o empurrar de cabeçalhos de HTTP2

Você pode controlar quais ligações empurrar usando a função `pushAssets`.

Exemplo:

```
pushAssets: (req, res, publicPath, preloadFiles) =>
  preloadFiles
    .filter(f => f.asType === 'script' && f.file === 'runtime.js')
    .map(f => `<${publicPath}${f.file}>; rel=preload; as=${f.asType}`)
```

Você pode adicionar os seus próprios recursos ao arranjo também. Usando o `req` e o `res` você pode decidir quais ligações para empurrar baseado nos cabeçalhos da requisição, por exemplo usando o `cookie` com a versão da aplicação.

Os recursos serão juntados com o `,` e passados como um cabeçalho `Link` único.

A propriedade `asyncScripts`

- Tipo: `Boolean`
 - Valor padrão: `false`

Adiciona um atributo `async` para os marcadores de `<script>` para pacotes do Nuxt, habilitando eles a serem requisitados em paralelo para parseamento (disponível desde a versão `2.14.8+` do Nuxt). [Mais informações](#).

A propriedade `injectScripts`

- Tipo: `Boolean`
 - Valor padrão: `true`

Adiciona o `<script>` para os pacotes do Nuxt, defina ele para `false` para renderizar HTML puro sem JavaScript (disponível desde a versão `2.14.8+` do Nuxt)

A propriedade `resourceHints`

- Tipo: `Boolean`
 - Valor padrão: `true`

Adiciona as ligações `prefetch` e `preload` para acelerar o tempo de carregamento da página.

Você talvez queira apenas desativar esta opção se você tiver muitas páginas e rotas.

A propriedade ssr

- Tipo: `Boolean`
 - Valor padrão: `true`
 - `false` only client side rendering

Ativa a Renderização no Lado do Servidor

Esta opção é automaticamente definida baseada no valor global de `ssr` se não for fornecida. Isto pode ser útil para dinamicamente ativar/desativar a Renderização no Lado do Servidor em tempo de execução depois da construção da imagem (com o docker por exemplo).

A propriedade crossorigin

- Tipo: `String`
- Valor padrão: `undefined`

Configura o atributo `crossorigin` nos marcadores `<link rel="stylesheet">` e `<script>` dentro do HTML gerado.

Mais Informações: [atributos de definições de CORS](#)

A propriedade ssrLog

- Tipo: `Boolean | String`
 - Valor padrão: `true` em modo de desenvolvimento e `false` em produção

Encaminha os regtos do lado do servidor para o navegador para melhor depuração (apenas disponível em desenvolvimento)

Para colapsar os regtos, use o valor `'collapsed'`.

A propriedade static

- Tipo: `Object`
 - Valor padrão: `{}`

Configura o comportamento do diretório `static/`

Consulta a documentação [serve-static](#) para conhecer as possíveis opções.

Adicionalmente a eles, nós introduzimos uma opção `prefix` a qual valor padrão é definida para `true`. Ela adicionará a base do roteador aos seus recursos estáticos.

Exemplo:

- Recursos: `favicon.ico`
- Base do Roteador: `/t`
- Com o `prefix: true` (valor padrão): `/t/favicon.ico`
- Com o `prefix: false`: `/favicon.ico`

Avisos:

Algumas restrições de URL podem não respeitar o prefixo.

A propriedade dist

- Tipo: `Object`
 - Valor padrão: `{ maxAge: '1y', index: false }`

Opções usadas para servir a distribuição de ficheiros. Somente aplicável em produção.

Consulte a documentação [serve-static](#) para conhecer as possíveis opções.

A propriedade csp

- Tipo: `Boolean` ou `Object`
 - Valor padrão: `false`

Use isto para configurar a Política de Segurança de Conteúdo para carregar recursos externos

Pré-requisitos:

Estas definições de Política de Segurança de Conteúdo apenas são eficazes quando estiver usando o Nuxt com o `target: 'server'` para servir a sua aplicação Renderizada no Lado do Servidor. As políticas definidas conforme o `csp.policies` são adicionadas ao cabeçalho `Content-Security-Policy` da resposta do HTTP.

Atualizando definições:

Estas definições são lidas pelo servidor do Nuxt diretamente a partir do ficheiro `nuxt.config.js`. Isto significa para mudanças para essas definições tem efeito quando o servidor é reiniciado. Não há necessidade de reconstruir a aplicação para atualizar as definições da Política de Segurança de Conteúdo.

Marcador meta do HTML:

NO sentido de adicionar `<meta http-equiv="Content-Security-Policy"/>` ao `<head>` você precisa definir o `csp.addMeta` para `true`. Repare que essa funcionalidade é independente da configuração do `csp.policies`:

- ele apenas adiciona uma política do tipo `script-src`, e
- a política `script-src` apenas contém os hashes dos marcadores `<script>` em linha.

Quando o `csp.addMeta` estiver definido para `true`, o conjunto completo das políticas definidas continuam a serem adicionadas ao cabeçalho da resposta do HTTP.

Repare que as hashes da Política de Segurança de Conteúdo não serão adicionadas como `<meta>` se a política `script-src` conter o `'unsafe-inline'`. Isto é para navegador ignorar `'unsafe-inline'` se as hashes estiverem presentes. Defina a opção `unsafeInlineCompatibility` para `true` se você quiser ambas hashes e o `'unsafe-inline'` para compatibilidade com a versão 1 da Política de Segurança de Conteúdo. Neste caso o marcador `<meta>` continuará contendo somente as hashes dos marcadores `<script>` em linha, e as políticas definidas conforme `csp.policies` serão usadas dentro do cabeçalho `Content-Security-Policy` da resposta do HTTP.

```
export default {
  render: {
    csp: true
  }
}

// OU

export default {
  render: {
    csp: {
      hashAlgorithm: 'sha256',
      policies: {
        'script-src': [
          'https://www.google-analytics.com',
          'https://name.example.com'
        ],
        'report-uri': ['https://report.example.com/report-csp-violations']
      },
      addMeta: true
    }
  }
}

// OU
/*
  O seguinte exemplo permite o Google Analytics, LogRocket.io, e o Sentry.io
  fazerem o registo e rastreio analítico.

  Consulte este blogue no Sentry.io
  https://blog.sentry.io/2018/09/04/how-sentry-captures-csp-violations

  Para saber qual ligação de rastreio você deve usar.
*/
const PRIMARY_HOSTS = `loc.example-website.com`
export default {
```

```
render: {
  csp: {
    reportOnly: true,
    hashAlgorithm: 'sha256',
    policies: {
      'default-src': ["'self'"],
      'img-src': ['https:', '*.google-analytics.com'],
      'worker-src': ["'self'", `blob:`, PRIMARY_HOSTS, '*logrocket.io'],
      'style-src': ["'self'", "'unsafe-inline'", PRIMARY_HOSTS],
      'script-src': [
        "'self'",
        "'unsafe-inline'",
        PRIMARY_HOSTS,
        'sentry.io',
        '*.sentry-cdn.com',
        '*.google-analytics.com',
        '*.logrocket.io'
      ],
      'connect-src': [PRIMARY_HOSTS, 'sentry.io', '*google-analytics.com'],
      'form-action': ["'self'"],
      'frame-ancestors': ["'none'"],
      'object-src': ["'none'"],
      'base-uri': [PRIMARY_HOSTS],
      'report-uri': [
        `https://sentry.io/api/<project>/security/?sentry_key=<key>`
      ]
    }
  }
}
```

[Go to TOC](#)

A propriedade rootDir

Define o espaço de trabalho da aplicação Nuxt

-
- Tipo: `String`
 - Valor padrão: `process.cwd()`

Esta propriedade será sobreescrita pelos comandos do Nuxt (`nuxt start`, `nuxt build`, etc) se um argumento for passado para eles. Por exemplo executar `nuxt ./my-app/` definirá o `rootDir` para o caminho absoluto de `./my-app/` do diretório atual/de trabalho.

Por causa disto ele normalmente não precisa configurar esta opção a menos que você use o [Nuxt programaticamente](#).

Info

Ambos o `rootDir` como a raiz do pacote contendo o diretório `node_modules` precisam estar dentro da mesma árvore de diretório para serem capazes de [resolver dependências](#). Consulte a opção `srcDir` para exemplos de estruturas de diretório quando esse não for o caso.

A propriedade router

A propriedade router permite você personalizar o roteador do Nuxt. ([vue-router](#)).

A propriedade base

- Tipo: `String`
- Valor padrão: `'/'`

A base da URL da aplicação. Por exemplo, se aplicação de página única inteira for servida debaixo de `/app/`, então a base deve usar o valor `'/app/'`.

Isto pode ser útil se você precisar servir o Nuxt como uma raiz de contexto diferente, a partir de dentro de um sítio maior na teia de computadores. Repare que você pode, ou não definir um Servidor de Procuração de Rede Frontal (Front Proxy Web Server).

Se você quiser ter um redirecionamento para o `router.base`, você pode fazer isso [usando um Gatilho](#), consulte o [Redirecionar para `router.base` quando não estiver na raiz](#).

Desde a versão 2.15 do Nuxt, a mudança do valor desta propriedade em tempo de execução sobrescreverá a configuração de uma aplicação que já foi construída.

```
export default {
  router: {
    base: '/app/'
  }
}
```

Info

Quando o `base` estiver definido, o Nuxt também adicionará dentro do cabeçalho do documento o `<base href="{{ router.base }}"/>`.

Esta opção é dada diretamente ao `base` do `vue-router`.

A propriedade routeNameSplitter

- Tipo: `String`
- Valor padrão: `'-'`

Você pode querer mudar o separador entre os nomes de rota que o Nuxt usa. Você pode fazer isso através da opção `routerNameSplitter` dentro do seu ficheiro de configuração. Imagina que nós temos um ficheiro de página `pages/posts/_id.vue`. O Nuxt gerará o nome da rota programaticamente, neste caso o `posts-id`. Mudando a configuração do `routerNameSplitter` para `/` o nome mudará então para `posts/id`.

```
export default {
  router: {
    routeNameSplitter: '/'
  }
}
```

A propriedade extendRoutes

- Tipo: `Function`

Você pode querer estender as rotas criadas pelo Nuxt. Você pode fazer isso através da opção `extendRoutes`.

```
export default {
  router: {
    extendRoutes(routes, resolve) {
      routes.push({
        name: 'custom',
        path: '*',
        component: resolve(__dirname, 'pages/404.vue')
      })
    }
  }
}
```

Se você quiser organizar as suas rotas, você pode usar a função `sortRoutes(routes)` do módulo `@nuxt/utils`:

```
import { sortRoutes } from '@nuxt/utils'
export default {
  router: {
    extendRoutes(routes, resolve) {
      // Adicione algumas rotas aqui...
      // e depois organize elas
      sortRoutes(routes)
    }
  }
}
```

O estrutura da rota deve respeitar o estrutura do `vue-router`.

Warning

Quando estiver adicionando rotas que usam Apresentações Nomeadas, não esqueça de adicionar o `chunkNames` correspondente dos `components` nomeados.

```
export default {
  router: {
    extendRoutes(routes, resolve) {
      routes.push({
        path: '/users/:id',
        components: {
          default: resolve(__dirname, 'pages/users'), // ou
          routes[index].component
          modal: resolve(__dirname, 'components/modal.vue')
        },
      })
    }
  }
}
```

```

        chunkNames: {
          modal: 'components/modal'
        }
      })
    }
}

```

A propriedade fallback

- Tipo: `boolean`
- Valor padrão: `false`

Controla se o roteador deve recuar para o modo de `hash` quando o navegador não suporta o modo histórico. Empurra o estado mas o modo é definido para histórico.

Definindo isto para `false` essencialmente faz um recarregamento completo da página em toda navegação do `router-link` no Internet Explorer 9. Isto é útil quando a aplicação é renderizada no servidor e precisa funcionar no Internet Explorer 9, porque o modo de `hash` da URL não funciona com a Renderização no Lado do Servidor.

Esta opção é dada diretamente ao `fallback` do `vue-router`.

A propriedade linkActiveClass

- Tipo: `String`
- Valor padrão: `'nuxt-link-active'`

Configura globalmente a classe ativa padrão do componente `<nuxt-link>`.

```

export default {
  router: {
    linkActiveClass: 'active-link'
  }
}

```

Esta opção é dada diretamente ao `linkActiveClass` do `vue-router`.

A propriedade linkExactActiveClass

- Tipo: `String`
- Valor padrão: `'nuxt-link-exact-active'`

Configura globalmente a classe ativa exata padrão do componente `<nuxt-link>`.

```
export default {
  router: {
    linkExactActiveClass: 'exact-active-link'
  }
}
```

Esta opção é dada diretamente ao `linkExactActiveClass` do `vue-router`.

A propriedade linkPrefetchedClass

- Tipo: `String`
- Valor padrão: `false`

Configura globalmente a classe de pré-requisição padrão do componente `<nuxt-link>` (funcionalidade desativada por padrão).

```
export default {
  router: {
    linkPrefetchedClass: 'nuxt-link-prefetched'
  }
}
```

A propriedade middleware

- Tipo: `String` ou `Array`
 - Itens: `String`

Define o(s) intermediário(s) padrão para cada página da aplicação.

```
export default {
  router: {
    // Executa o middleware/user-agent.js em cada página
    middleware: 'user-agent'
  }
}
```

```
export default function (context) {
  // Adiciona a propriedade `userAgent` dentro do contexto (disponível dentro do gatilho `asyncData` e `fetch`)
  context.userAgent = process.server
    ? context.req.headers['user-agent']
    : navigator.userAgent
}
```

Para saber mais sobre o intermediário, consulte o [guia do intermédario](#).

A propriedade mode

- Tipo: `String`
- Valor padrão: `'history'`

Configura o modo do roteador, não é recomendado mudar ele por causa da renderização no lado do servidor.

```
export default {
  router: {
    mode: 'hash'
  }
}
```

Esta opção é dada diretamente ao `mode` do `vue-router`.

A propriedade `parseQuery / stringifyQuery`

- Tipo: `Function`

Fornece análise personalizada de sequência de caracteres de consulta / transforma funções em sequências de caracteres. Sobrescreve o valor padrão.

Esta opção é dada diretamente ao `parseQuery / stringifyQuery` do `vue-router`.

A propriedade `prefetchLinks`

Adicionada com a versão 2.4.0 do Nuxt

- Tipo: `Boolean`
- Valor padrão: `true`

Configura o componente `<nuxt-link>` para pré-requisitar o *código separado* da página quando detetado dentro da janela de visualização. Requer o `IntersectionObserver` para ser suportado (consulte o [CanIUse](#)).

Nós recomendamos condicionalmente preencher esta funcionalidade com um serviço como o [Polyfill.io](#):

```
export default {
  head: {
    script: [
      {
        src:
          'https://polyfill.io/v3/polyfill.min.js?features=IntersectionObserver',
        body: true
      }
    ]
  }
}
```

Para desativar a pré-requisição em uma ligação específica, você pode usar o atributo `no-prefetch`. Desde a versão 2.10.0 do Nuxt, você pode também usar o atributo `prefetch` para definir para `false`:

```
<nuxt-link to="/about" no-prefetch>About page not prefetched</nuxt-link>
<nuxt-link to="/about" :prefetch="false">About page not prefetched</nuxt-link>
```

Para desativar a pré-requisição em todas as ligações, defina a propriedade `prefetchLinks` para `false`:

```
export default {
  router: {
    prefetchLinks: false
  }
}
```

Desde a versão 2.10.0 do Nuxt, se você tiver o `prefetchLinks` definido como `false` mas você quer pré-requisitar uma ligação específica, você pode usar o atributo `prefetch`:

```
<nuxt-link to="/about" prefetch>About page prefetched</nuxt-link>
```

A propriedade prefetchPayloads

Adicionada com a versão 2.13.0 do Nuxt, apenas disponível para o [alvo estático](#).

- Tipo: `Boolean`
- Valor padrão: `true`

Quando usar o comando `nuxt generate` com o `target: 'static'`, o Nuxt gerará um ficheiro `payload.js` para cada página.

Com esta opção ativada, o Nuxt pré-requisitará automaticamente o `payload` da página ligada quando o componente `<nuxt-link>` estiver visível dentro da janela de visualização, tornando a **navegação instantânea**.

Info

Esta opção precisa que a opção `prefetchLinks` esteja ativada.

Você pode desativar este comportamento ao definir `prefetchPayloads` para `false`:

```
export default {
  router: {
    prefetchPayloads: false
  }
}
```

A propriedade scrollBehavior

- Tipo: `Function`

A opção `scrollBehavior` permite você definir um comportamento personalizado para a posição da rolagem entre as rotas. Este método é chamado toda vez que uma página é renderizada. Para saber mais sobre ele, consulte a [documentação do scrollBehavior do vue-router](#).

Desde a versão 2.9.0 do Nuxt, você pode usar um ficheiro para sobrescrever o `scrollBehavior` do roteador, este ficheiro deve ser colocado dentro de `~/app/router.scrollBehavior.js` (Repare que: o nome do ficheiro é sensível a caixa dos caracteres se estiver executando no Windows).

Warning

O ficheiro `router.scrollBehavior.js` deve estar dentro da pasta `app`, a qual por sua vez está dentro da raiz do projeto.

Você pode consultar o ficheiro `router.scrollBehavior.js` padrão do Nuxt aqui: [packages/vue-app/template/router.scrollBehavior.js](#).

Exemplo de forçamento da posição da rolagem ao topo para todas as rotas:

`app/router.scrollBehavior.js`

```
export default function (to, from, savedPosition) {
  return { x: 0, y: 0 }
}
```

A propriedade trailingSlash

- Tipo: `Boolean` ou `undefined`
- Valor padrão: `undefined`
- Disponível desde versão: 2.10

Se esta opção estiver definida para `true`, rastos de barras serão acrescentados à cada rota. Se estiver definida para `false`, os rastros de barras serão removidos.

Atenção: Esta opção não deve ser definida sem preparação e tem de ser testada cuidadosamente. Quando estiver definindo `router.trailingSlash` para alguma coisa diferente de `undefined`, a rota oposta parará de funcionar. Assim os redirecionamentos de 301 deve estar no lugar e a sua *ligação interna* tem de estar corretamente adaptada. Se você definir a propriedade `trailingSlash` para `true`, então apenas o `example.com/abc/` funcionará mas não o `example.com/abc`. Se você definir para `false`, será o contrário.

Exemplo do comportamento (com as rotas filhas)

Para um diretório com esta estrutura:

```
- | pages/
---| index.vue
---| posts.vue
---| posts/
-----| _slug.vue
-----| index.vue
```

Este é o comportamento para cada possível definição da propriedade `trailingSlash`:

Route	Page
/	~/pages/index.vue
/posts	~/pages/posts.vue (parent) + ~/pages/posts.vue (child route)
/posts/	~/pages/posts.vue (parent) + ~/pages/posts.vue (child route)
/posts/foo	~/pages/posts.vue (parent) + ~/pages/_slug.vue (child route)
/posts/foo/	~/pages/posts.vue (parent) + ~/pages/_slug.vue (child route)

Route	Page
/	~/pages/index.vue
/posts	404
/posts/	~/pages/posts.vue (parent) + ~/pages/index.vue (child route)
/posts/foo	404
/posts/foo/	~/pages/posts.vue (parent) + ~/pages/_slug.vue (child route)

Route	Page
/	~/pages/index.vue
/posts	~/pages/posts.vue
/posts/	~/pages/posts.vue (parent) + ~/pages/index.vue (child route)
/posts/foo	~/pages/posts.vue (parent) + ~/pages/_slug.vue (child route)
/posts/foo/	404

As propriedades do RuntimeConfig

O runtimeConfig permite a passagem de configurações dinâmicas e variáveis de ambiente ao contexto do Nuxt. Para mais informações de uso, consulte o [guia do runtimeConfig](#)

publicRuntimeConfig

- Tipo: `Object`

O valor deste objeto é **acessível a partir de ambos cliente e servidor** com uso do `$config`.

privateRuntimeConfig

- Tipo: `Object`

O valor deste objeto é acessível apenas a partir do **servidor** com uso do `$config`. Sobrescreve o `publicRuntimeConfig` para o servidor.

A propriedade server

O Nuxt permite você definir as variáveis de conexão do servidor para a sua aplicação dentro do ficheiro `nuxt.config.js`.

- Tipo: `Object`

Exemplo Básico:

```
export default {
  server: {
    port: 8000, // valor padrão: 3000
    host: '0.0.0.0', // valor padrão: localhost,
    timing: false
  }
}
```

Isto permite você especificar o [hospedeiro e a porta](#) para a sua instância de servidor do Nuxt.

Exemplo usando a configuração do HTTPS

```
import path from 'path'
import fs from 'fs'

export default {
  server: {
    https: {
      key: fs.readFileSync(path.resolve(__dirname, 'server.key')),
      cert: fs.readFileSync(path.resolve(__dirname, 'server.crt'))
    }
  }
}
```

Você pode encontrar informações adicionais sobre a criação de chaves e certificados de servidor no [localhost](#) no artigo [certificados para o localhost](#).

Exemplo usando configuração do sockets

```
export default {
  server: {
    socket: '/tmp/nuxt.socket'
  }
}
```

A propriedade timing

- Tipo: `Object` ou `Boolean`
- Valor padrão: `false`

A ativação da opção `server.timing` adiciona um intermediário para medir o tempo decorrido durante a renderização no lado do servidor e adiciona ele aos cabeçalhos como `Server-Timing`

Exemplo usando a configuração do timing

O `server.timing` pode ser um objeto para fornecimento de opções. Atualmente, apenas o `total` é suportado (o qual rastreia diretamente o tempo todo gasto na renderização do lado do servidor)

```
export default {
  server: {
    timing: {
      total: true
    }
  }
}
```

Usando a API do timing

A API do `timing` é também injetada dentro do `response` no lado do servidor quando o `server.time` estiver ativado.

A Sintaxe

```
res.timing.start(name, description)
res.timing.end(name)
```

Exemplo usando o timing dentro do serverMiddleware

```
export default function (req, res, next) {
  res.timing.start('midd', 'Middleware timing description')
  // operação no lado do servidor...
  // ...
  res.timing.end('midd')
  next()
}
```

Depois o cabeçalho `server-timing` será incluído dentro do cabeçalho de resposta como:

```
Server-Timing: midd;desc="Middleware timing description";dur=2.4
```

Recorra ao [MDN do server-timing](#) para mais detalhes.

A propriedade serverMiddleware

Define o intermediário no lado do servidor.

- Tipo: `Array`
 - Itens: `String` ou `Object` ou `Function`

O Nuxt internamente cria um instância de `connect` que você pode adicionar ao seu próprio intermediário. Isto permite-nos registar rotas adicionais (normalmente rotas `/api`) **sem precisar de um servidor externo**.

Por `connect` mesmo ser um intermediário, o intermediário registrado funcionará com ambos `nuxt start` e também quando usado como um intermediário com uso programático como o `express-template`. Os `Módulos` do Nuxt pode também fornecer o `serverMiddleware` usando o `this.addServerMiddleware()`

Adicionalmente a eles, nós introduzimos uma opção `prefix` a qual o valor padrão será `true`. Ela adicionará uma base de roteador para os intermediários do seu servidor.

Exemplo:

- Caminho do intermediário do servidor: `/server-middleware`
- Base de roteador: `/admin`
- Com o `prefix: true` (valor padrão): `/admin/server-middleware`
- Com o `prefix: false`: `/server-middleware`

serverMiddleware vs middleware!

Não confunda ele com o `intermediários de rotas` as quais são chamadas antes de cada rota pelo Vue no lado do cliente ou na renderização no lado do servidor. O intermediário listado dentro da propriedade `serverMiddleware` executa no lado do servidor **antes** do `vue-server-renderer` e pode ser usado para tarefas específicas do servidor como manipulando de requisições de API ou servindo recursos.

Warning

Não adicione o `serverMiddleware` ao diretório `middleware/`.

Os intermediários são empacotados pelo webpack dentro do seu pacote de produção e executam sobre o gatilho `beforeRouteEnter`. Se você adicionar o `serverMiddleware` ao diretório `middleware/` ele será erradamente tomado pelo Nuxt como um intermediário comum e adicionará dependências erradas ao seu pacote ou gerar erros.

Uso

Se o intermediário for um sequência de caracteres o Nuxt tentará pedir e resolver ele automaticamente.

```

import serveStatic from 'serve-static'

export default {
  serverMiddleware: [
    // Registarão o pacote `redirect-ssl` de npm
    'redirect-ssl',

    // Registarão o ficheiro do diretório `server-middleware` do projeto manipular
    os pedidos de `/server-middleware/*`
    { path: '/server-middleware', handler: '~/server-middleware/index.js' },

    // Podemos também criar instâncias personalizadas
    { path: '/static2', handler: serveStatic(__dirname + '/static2') }
  ]
}

```

Warning

Se você não quiser o intermediário para registrar para todas rotas, você tem que usar a forma de objeto com um caminho específico, de outra maneira o manipulador padrão do Nuxt não funcionará!

Intermediário de Servidor Personalizado

É também possível escrever um intermediário personalizado. Para mais informações consulte a [Documentação do Connect](#).

Intermediário (`server-middleware/logger.js`):

```

export default function (req, res, next) {
  // O `req` é o objeto de requisição http do Node.js
  console.log(req.url)

  // O `res` é o objeto de resposta http do Node.js

  // O `next` é uma função para chamar o próximo intermediário
  // Não esqueça de chamar o `next` no final se o seu intermediário não for um
  // ponto final!
  next()
}

serverMiddleware: ['~/server-middleware/logger']

```

Ponto Final de API Personalizada

Um intermediário de servidor pode também estender o `Express`. Isto permite a criação de pontos finais de REST.

```

const bodyParser = require('body-parser')
const app = require('express')()

app.use(bodyParser.json())
app.all('/getJSON', (req, res) => {
  res.json({ data: 'data' })
})

module.exports = app

```

```
serverMiddleware: [
  { path: "/server-middleware", handler: "~/server-middleware/rest.js" },
],
```

A Sintaxe de Objeto

Se o seu intermediário de servidor compreende em uma lista de funções mapeadas para os caminhos:

```
export default {
  serverMiddleware: [
    { path: '/a', handler: '~/server-middleware/a.js' },
    { path: '/b', handler: '~/server-middleware/b.js' },
    { path: '/c', handler: '~/server-middleware/c.js' }
  ]
}
```

Você pode alternativamente passar um objeto para definir eles, como o seguinte:

```
export default {
  serverMiddleware: {
    '/a': '~/server-middleware/a.js',
    '/b': '~/server-middleware/b.js',
    '/c': '~/server-middleware/c.js'
  }
}
```

A propriedade srcDir

Define o diretório da fonte da sua aplicação Nuxt.

- Tipo: `String`
- Valor padrão: [valor do rootDir](#)

Se um caminho relativo for especificado ele será relativo ao `rootDir`.

Exemplo 1: Pré-requisitos:

```
export default {
  srcDir: 'client/'
```

```
"script": {
  "dev": "yarn nuxt"
}
```

funciona com a seguinte estrutura de pasta (nota que o ficheiro `nuxt.config` é listado dentro do diretório `app`, diretório da aplicação)

```
- app/
--- node_modules/
--- nuxt.config.js
--- package.json
--- client/
| assets/
| components/
| layouts/
| middleware/
| pages/
| plugins/
| static/
| store/
```

Exemplo 2:

No lugar do exemplo 1 você pode também mover o ficheiro `nuxt.config` dentro da sua pasta `client`. Neste caso você apenas precisa especificar o `client` como o `rootDir` e você pode deixar o `srcDir` vazio.

Pré-requisitos:

```
export default {
  srcDir: '' // ou apenas remova ele
}
```

```
"script": {
  "dev": "yarn nuxt client" // isto define a pasta `client` como o `rootDir`
}
```

funciona com a seguinte estrutura de pasta (nota que o `nuxt.config` está listado dentro do diretório `client`, diretório do cliente)

```
- | app/
---| node_modules/
---| package.json
---| client/
| nuxt.config.js
| assets/
| components/
| layouts/
| middleware/
| pages/
| plugins/
| static/
| store/
```

A propriedade buildDir

Define o diretório de dist (distribuição) para a sua aplicação Nuxt

- Tipo: `String`
- Padrão: `.nuxt`

```
export default {  
  buildDir: 'nuxt-dist'  
}
```

Por padrão, muitas ferramentas assumem que o diretório `.nuxt` é um diretório oculto, porque o seu nome começa com um ponto. Você pode usar esta opção para prevenir isso.

A propriedade ssr

Muda o valor padrão do ssr do Nuxt

- Tipo: `boolean`
- Valor padrão: `true`
- Possíveis valores:
 - `true`: Renderização no Lado do Servidor ativada
 - `false`: Sem renderização no lado do servidor (apenas renderização no lado do cliente)

Você pode definir esta opção para `false` quando você quiser **apenas a renderização no lado do cliente**

```
export default {  
  ssr: false // Desativa a renderização no lado do servidor  
}
```

Next

Anteriormente, o `mode` era usado para desativar e ativar a renderização no lado do servidor. Aqui está a [documentação do mode](#).

A propriedade target

Muda o alvo padrão do Nuxt

Alvos do desdobramento para o Nuxt na versão igual ou superior a 2.13:

- Tipo: `string`
 - Valor padrão: `server`
 - Possíveis valores:
 - `'server'` : Para renderização no lado do servidor
 - `'static'` : Para sítios estáticos

Você pode usar esta opção para mudar o alvo padrão do Nuxt para o seu projeto usando o ficheiro `nuxt.config.js`

Para saber mais sobre a opção `target` consulte a [secção de alvos do desdobramento](#).

[Go to TOC](#)

A propriedade telemetry

O Nuxt coleta dados de telemetria anónimos sobre o uso em geral. Isto ajuda-nos avaliar precisamente o uso de funcionalidade e personalização através de todos nossos usuários.

A propriedade telemetry

O versão 2.13.0 introduz o [Nuxt Telemetry](#) (Telemetria do Nuxt) para coletar dados de telemetria anónimos sobre o uso em geral. Isto ajuda-nos avaliar precisamente o uso de funcionalidade e personalização através de todos nossos usuários.

- Tipo: `Boolean`
- Valor padrão é baseado nas preferências do seu usuário

Porquê coletar a Telemetria

O Nuxt tem crescido muito desde o seu [lançamento inicial](#) (7 Nov 2016) e nós continuamos ouvindo a [reação da comunidade](#) para melhorar ele.

No entanto, este processo manual apenas coleta reações de um subconjunto de usuários que reservam tempo para preencher o modelo de questões e ele pode ter diferentes necessidades ou caso de uso do que você.

A Telemetria do Nuxt coleta **coleta dados de telemetria anónimos sobre o uso em geral**. Isto ajuda-nos avaliar precisamente o uso de funcionalidade e personalização através de todos nossos usuários. Estes dados irá permitir-nos entender melhor como o Nuxt é usado globalmente, medindo melhorias feitas (experiência do desenvolvedor e desempenhos) e suas relevâncias.

Nós coletamos vários eventos:

- Comando invocado (`nuxt dev`, `nuxt build`, etc)
- Versões do Nuxt e Node.js
- Informações da máquina em geral (MacOS/Linux/Windows e se o comando é executado dentro de um CI, o nome do CI)
- Duração da construção do Webpack e tamanho médio da aplicação, bem como as estatísticas da geração (quando estiver usando o `nuxt generate`)
- Quais são as dependências públicas do seu projeto (módulos do Nuxt)

O código é de código-aberto e está disponível no repositório [nuxt-telemetry](#).

Opção de saída

Você pode desativar a [Telemetria do Nuxt](#) para seu projeto de várias maneiras:

1. Usando o `npx nuxt telemetry disable`

```
npx nuxt telemetry [status|enable|disable] [-g,--global] [dir]
```

2. Usando uma variável de ambiente

```
NUXT_TELEMETRY_DISABLED=1
```

3. Definindo `telemetry: false` dentro do seu ficheiro `nuxt.config.js`:

```
export default {  
  telemetry: false  
}
```

Você pode saber mais sobre a Telemetria do Nuxt e os eventos enviados no repositório [nuxt-telemetry](#) no GitHub.

[Go to TOC](#)

As propriedades de transition

Define as propriedades padrão das transições de página e esquema.

A propriedade pageTransition

A versão 2.7.0 do Nuxt introduz a chave `pageTransition` em favor da chave `transition` para consolidar a nomeação com as chaves de transição de esquema.

- Tipo: `String` ou `Object`

Usada para definir as propriedades padrão das transições de página

Valor padrão:

```
{
  name: 'page',
  mode: 'out-in'
}
```

```
export default {
  pageTransition: 'page'
  // ou
  pageTransition: {
    name: 'page',
    mode: 'out-in',
    beforeEnter (el) {
      console.log('Before enter...');
    }
  }
}
```

A chave da transição dentro do ficheiro `nuxt.config.js` é usada para definir as propriedades padrão das transições de página. Para saber mais sobre as chaves disponíveis quando a chave `transition` é um objeto, consulte [propriedade da transição de páginas](#).

A propriedade layoutTransition

- Tipo: `String` ou `Object`

Usada para definir as propriedades padrão das transições de esquema. O valor fornecido dentro da opção `name` é configurado para funcionar com o nome fornecido no `layout` da sua pasta `layouts`.

Valor padrão:

```
{  
  name: 'layout',  
  mode: 'out-in'  
}
```

```
export default {  
  layoutTransition: 'layout'  
  // ou  
  layoutTransition: {  
    name: 'layout',  
    mode: 'out-in'  
  }  
}
```

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

[Go to TOC](#)

A propriedade vue.config

Um objeto de configuração para o `Vue.config`

- Tipo: `Object`
- Valor padrão: `{ silent: !isDev, performance: isDev }`

A propriedade `vue.config` fornece uma ponte de configuração direta para o `Vue.config`

Exemplo

```
export default {
  vue: {
    config: {
      productionTip: true,
      devtools: false
    }
  }
}
```

Esta configuração resultará no seguinte `Vue.config`:

```
Vue.config.productionTip // true
Vue.config.devtools // false
Vue.config.silent // !isDev [valor padrão]
Vue.config.performance // isDev [valor padrão]
```

Para saber mais sobre a API do `Vue.config`, consulte a [documentação oficial do Vue](#)

A propriedade watch

A propriedade `watch` permite você observar ficheiros personalizados para reinicialização do servidor.

- Tipo: `Object`
- Valor padrão: `[]`

`watch: ['~/custom/*.js']`

O `chokidar` é usado para definir os observadores. Para saber mais sobre as opções do padrão do chokidar, consulte a [API do chokidar](#).

A propriedade watchers

A propriedade `watchers` permite você sobrescrever a configuração dos observadores dentro do seu ficheiro `nuxt.config.js`.

- Tipo: `Object`
- Valor padrão: `{}`

A propriedade chokidar

- Tipo: `Object`
- Valor padrão: `{}`

Para saber mais sobre as opções do `chokidar`, consulte a [API do chokidar](#).

A propriedade webpack

- Tipo: `Object`
- Valor padrão:

```
watchers: {  
  webpack: {  
    aggregateTimeout: 300,  
    poll: 1000  
  }  
}
```

Para saber mais sobre as opções de observação do webpack, consulte a [documentação do webpack](#).

O que se segue

Next

Consulte o livro [Glossário de Interiores](#)

A propriedade cli

O Nuxt permite você personalizar o configuração da interface de linha de comando.

A propriedade badgeMessages

- Tipo: `Array`

Adiciona uma mensagem à faixa da interface de linha de comando.

```
cli: {  
  badgeMessages: ['Hello World!']  
}
```

Nuxt.js @ v2.14.7

- ▶ **Environment:** development
- ▶ **Rendering:** server-side
- ▶ **Target:** static

Listening:

<http://localhost:3000/>

Hello World!

A propriedade bannerColor

- Tipo: `String`
 - Padrão: `'green'`

Muda a color do título 'Nuxt' dentro da faixa da interface de linha de comando.

Cores disponíveis:

```
black, red, green, yellow, blue, magenta, cyan, white, gray, redBright, greenBright,  
yellowBright, blueBright, magentaBright, cyanBright, whiteBright
```

```
export default {  
  cli: {  
    bannerColor: 'yellow'  
  }  
}
```

A propriedade css

O Nuxt permite você definir ficheiros/módulos/bibliotecas CSS que você quiser para definir globalmente (incluído em todas páginas).

No caso de você querer usar `sass` certifique-se que você tenha instalado os pacotes `sass` e `sass-loader`. Se você não tiver, apenas

```
yarn add --dev sass sass-loader@10
```

```
npm install --save-dev sass sass-loader@10
```

- Tipo: `Array`
 - Itens: `string`

```
export default {
  css: [
    // Carrega um módulo de Node.js diretamente (aqui está um ficheiro Sass)
    'bulma',
    // Ficheiro CSS dentro do projeto
    '@/assets/css/main.css',
    // Ficheiro SCSS dentro do projeto
    '@/assets/css/main.scss'
  ]
}
```

O Nuxt irá automaticamente adivinhar o tipo do ficheiro pela sua extensão e usar o carregador do pré-processador adequado para o webpack. Você continuará a precisar instalar o carregador exibido se você precisar usar eles.

Extensões de Estilo

Você pode omitir a extensão do ficheiro para os ficheiros CSS/SCSS/PostCSS/Less/Stylus listados no array `css` dentro do seu ficheiro `nuxt.config`.

```
export default {
  css: ['~/assets/css/main', '~/assets/css/animations']
}
```

Warning

Se você tem dois ficheiros com o mesmo nome por exemplo `main.scss` e `main.css`, e não especificar uma extensão na entrada do array `css`, por exemplo `css: ['~/assets/css/main']`, então apenas um ficheiro será carregado dependendo da ordem do `styleExtensions`. Neste caso apenas o ficheiro `css` será carregado e o ficheiro `scss` será ignorado porque o `css` vem primeiro no array `styleExtension` padrão.

Ordem padrão: `['css', 'pcss', 'postcss', 'styl', 'stylus', 'scss', 'sass', 'less']`

[Go to TOC](#)

A propriedade components

O Nuxt 2.13+ pode examinar e importar automaticamente os seus componentes usando o módulo `@nuxt/components`

Alert É possível usar esta funcionalidade com Nuxt 2.10 - 2.12. Apenas instale manualmente e adicione o `@nuxt/components` ao `buildModules` dentro do ficheiro `nuxt.config.js`.

- Tipo: `Boolean` ou `Array`
- Valor padrão: `false`

Quando definir para `true` ou para um objeto de opções, o Nuxt incluirá `@nuxt/components` e importar automaticamente o seus componentes sempre que você user eles dentro das suas páginas, esquemas (e outros componentes).

Info

Para mais informações de como usar, recorra a [documentação da descoberta automática de componente](#) para mais detalhes.

Configuração

```
export default {
  // Isto irá carregar automaticamente os componentes da pasta `~/components`
  components: true
}
```

Como o `components: true`, por padrão o diretório `~/components` será incluído.

No entanto você pode personalizar comportamento da descoberta automática pelo fornecimento de diretórios adicionais para examinar:

```
export default {
  components: [
    // Equivalente a { path: '~/components' }
    '~/components',
    { path: '~/components/other', extensions: ['vue'] }
  ]
}
```

A propriedade path

Cada item pode ser tanto uma sequência de caracteres ou um objeto. Uma valor de sequência de caracteres é um atalho para `{ path: ... }`.

Info

Não se preocupe com a organização ou sobreposição de diretórios! O módulo de componentes cuidará disso. (Cada ficheiro será correspondido apenas uma vez com o caminho mais longo.)

A propriedade path

- Obrigatório
- Tipo: `String`

O caminho (absoluto ou relativo) para o diretório que contém seus componentes.

Você pode usar os apelidos do Nuxt (`~` ou `@`) para referir aos diretórios dentro do projeto ou usar diretamente um caminho de pacote npm (similar ao uso de `require` dentro do seu projeto).

A propriedade extensions

- Tipo: `Array<string>`
- Valor padrão:
 - Extensões suportadas pelo construtor do Nuxt (`builder.supportedExtensions`)
 - Extensões padrão suportadas `['vue', 'js']` ou `['vue', 'js', 'ts', 'tsx']` dependendo do seu ambiente

Exemplo: Suportar a estrutura de componente de vários ficheiros

Se você preferir separar seus componentes de ficheiro único (SFCs) em `.js`, `.vue` e `.css`, você pode escolher apenas examinar ficheiros `.vue`:

```
| components
---| componentC
| componentC.vue
| componentC.js
| componentC.scss
```

```
// nuxt.config.js
export default {
  components: [{ path: '~/components', extensions: ['vue'] }]
}
```

A propriedade pattern

- Tipo: `string` (padrão glob)
- Valor padrão: `**/*.${extensions.join(',')}`

Dentro do `path` (caminho) especificado, apenas ficheiros que corresponderem a este padrão serão incluídos.

A propriedade ignore

- Tipo: `Array`
- Itens: `string` (padrão glob)
- Valor padrão: `[]`

Padrões para excluir ficheiros dentro do `path` (caminho) especificado.

A propriedade prefix

- Tipo: `String`
- Valor padrão: `''` (sem prefixo)

Prefixa todos componentes correspondidos.

O exemplo abaixo adiciona o prefixo `awesome-` / `awesome` ao nome dos componentes dentro do diretório `awesome/`.

```
// nuxt.config.js
export default {
  components: [
    '~/components',
    { path: '~/components/awesome/', prefix: 'awesome' }
  ]
}
```

```
| components/
---| awesome/
| Button.vue
---| Button.vue
```

```
<template>
  <div>
    <AwesomeButton>Click on me 🤘 </AwesomeButton>
    <button>Click on me</button>
  </div>
</template>
```

A propriedade pathPrefix

- Tipo: `Boolean`
- Valor padrão: `true`

Prefixa o nome do componente pelo seu caminho.

A propriedade watch

- Tipo: `Boolean`
- Valor padrão: `true`

Observa mudanças no `path` (caminho) especificado, incluindo adição e eliminação de ficheiros.

A propriedade transpile

- Tipo: `Boolean`
- Valor padrão: `'auto'`

Transpila o `path` (caminho) especificado usando o `build.transpile`. Por padrão (`'auto'`) definirá `transpile: true` se o `node_modules` estiver dentro `path` (caminho).

A propriedade level

- Tipo: `Number`
- Valor padrão: `0`

Níveis são usados para definir permissão sobrescrevendo componentes que tem o mesmo nome dentro dentro de diretórios diferentes. Isto pode ser útil para autores de bibliotecas que desejam permitir os usuários sobrescrever seus componentes, ou personalizar os temas.

```
export default {
  components: [
    '~/components', // 0 nível padrão é 0
    { path: 'my-theme/components', level: 1 }
  ]
}
```

Um componente dentro de `~/components` irá então sobrescrever um componente com o mesmo nome dentro de `my-theme/components`. O valor mais baixo recebe prioridade.

Avançado

Sobrescrevendo Componentes

É possível ter uma maneira para sobrescrever componentes usando a opção `level`. Isto é muito útil para autores de módulos e temas.

Considerando esta estrutura:

```
| node_modules/
---| my-theme/
| components/
| Header.vue
| components/
---| Header.vue
```

Então definindo dentro do `nuxt.config`:

```
components: [
  '~/components', // 0 nível padrão é 0
  { path: 'node_modules/my-theme/components', level: 1 }
]
```

O nosso `components/Header.vue` sobrescreverá o nosso componente do tema visto que o valor nível mais baixo recebe prioridade.

Autores de Biblioteca

Producir bibliotecas de componentes de Vue com sacudidela de árvore (tree-shaking) automática e registo de componente é agora super fácil!

Este módulo expõe um gatilho nomeado `components:dirs` assim você pode facilmente estender a lista de diretório sem a exigência de configuração do usuário dentro do seu módulo do Nuxt.

Imagina uma estrutura de diretório como esta:

```
| node_modules/
---| awesome-ui/
| components/
| Alert.vue
| Button.vue
| nuxt.js
| pages/
---| index.vue
| nuxt.config.js
```

Depois dentro `awesome-ui/nuxt.js` você pode usar o gatilho `components:dir`:

```
import { join } from 'path'

export default function () {
  this.nuxt.hook('components:dirs', dirs => {
    // Adiciona diretório ./components à lista
    dirs.push({
      path: join(__dirname, 'components'),
      prefix: 'awesome'
    })
  })
}
```

E é isso! Agora dentro do seu projeto, você pode importar sua biblioteca de UI como um módulo do Nuxt dentro do seu `nuxt.config.js`:

```
export default {
  buildModules: ['@nuxt/components', 'awesome-ui/nuxt']
}
```

E use o módulo de componentes diretamente (prefixado com `awesome-`), em nossa `pages/index.vue`:

```
<template>
<div>
  My <AwesomeButton>UI button</AwesomeButton>!
  <awesome-alert>Here's an alert!</awesome-alert>
</div>
</template>
```

Ele importará automaticamente os componentes somente se usado e se também suportar HMR quando estiver atualizando seus componentes dentro de `node_modules/awesome-ui/components/`.

Proximo: publicar o seu módulo `awesome-ui` no `npm` e partilhar ele com outros Nuxteres ✨

[Go to TOC](#)

A propriedade dev

Define o modo de desenvolvimento ou produção.

- Tipo: Boolean
- Valor padrão: true

Esta propriedade é sobreescrita pelos comandos do Nuxt:

- dev é forçado para true com o comando nuxt
- dev é forçado para false com o comando nuxt build, nuxt start e nuxt generate

Esta propriedade deve ser usada quando estiver usando o [Nuxt programaticamente](#):

```
export default {
  dev: process.env.NODE_ENV !== 'production'
}
```

```
const { Nuxt, Builder } = require('nuxt')
const app = require('express')()
const port = process.env.PORT || 3000

// Instanciamos o Nuxt com as opções
const config = require('./nuxt.config.js')
const nuxt = new Nuxt(config)
app.use(nuxt.render)

// Construa somente no de desenvolvimento
if (config.dev) {
  new Builder(nuxt).build()
}

// Oiça o servidor
app.listen(port, '0.0.0.0').then(() => {
  console.log(`Server is listening on port: ${port}`)
})
```

```
{
  "scripts": {
    "dev": "node server.js",
    "build": "nuxt build",
    "start": "NODE_ENV=production node server.js"
  }
}
```

[Go to TOC](#)

A propriedade dir

Define diretórios personalizados para a sua aplicação Nuxt

- Tipo: `Object`
- Valor padrão:

```
{  
  assets: 'assets',  
  app: 'app',  
  layouts: 'layouts',  
  middleware: 'middleware',  
  pages: 'pages',  
  static: 'static',  
  store: 'store'  
}
```

```
export default {  
  dir: {  
    assets: 'custom-assets',  
    app: 'custom-app',  
    layouts: 'custom-layouts',  
    middleware: 'custom-middleware',  
    pages: 'custom-pages',  
    static: 'custom-static',  
    store: 'custom-store'  
  }  
}
```

A propriedade env

Partilhe variáveis de ambiente entre o cliente e o servidor.

- Tipo: `Object`

O Nuxt permite você criar variáveis de ambiente do lado do cliente, e também serem partilhadas a partir do lado do servidor.

A propriedade `env` define variáveis de ambiente que devem estar disponível no lado do cliente. Eles podem ser atribuídos usando variáveis de ambiente do lado do servidor, com o [módulo dotenv](#) ou similares.

`alert` Para versões do Nuxt > 2.12+, nos casos onde variáveis de ambiente forem exigidas em tempo de execução (não em tempo de construção) é recomendado para substituir a propriedade `env` com as [propriedades do runtimeConfig](#): `publicRuntimeOptions` e `privateRuntimeOptions`.

Aprenda mais com o nosso tutorial sobre o [movendo do @nuxtjs/dotenv para configuração do tempo de execução](#).

Certifique-se de ler sobre o `process.env` e o `process.env == {}` abaixo para melhor resolução de problemas.

```
export default {
  env: {
    baseUrl: process.env.BASE_URL || 'http://localhost:3000'
  }
}
```

Isto permite você criar uma propriedade `baseUrl` que será igual a variável de ambiente `BASE_URL` do lado do servidor se estiver disponível ou definida. Se não, O `baseUrl` no lado do cliente será igual a `'http://localhost:3000'`. A variável `BASE_URL` do lado do cliente é então copiada para o lado do cliente através da propriedade `env` dentro do ficheiro `nuxt.config.js`. Alternativamente, o outro valor é definido (`http://localhost:3000`).

Depois, Eu posso acessar minha variável `baseUrl` em duas maneiras:

1. Através do `process.env.baseUrl`.
2. Através do `context.env.baseUrl`, consulte a [API de contexto](#).

Você pode usar a propriedade `env` para atribuir uma chave pública por exemplo.

Para o exemplo acima, nós podemos usar ele para configurar o `axios`.

```
import axios from 'axios'

export default axios.create({
  baseURL: process.env.baseUrl
})
```

Então, dentro da suas páginas, você pode importar o `axios` desse jeito: `import axios from '~/plugins/axios'`

Injeção automática de variáveis de ambiente

Se você definir variáveis de ambiente começando com o `NUXT_ENV_` na fase de construção (por exemplo, `NUXT_ENV_COOL_WORD=freezing nuxt build` ou `SET NUXT_ENV_COOL_WORD=freezing & nuxt build` para a consola do Windows, eles serão automaticamente injetados dentro do ambiente de processo). Esteja ciente que eles irão potencialmente ter precedência sobre variáveis definidas dentro do seu ficheiro `nuxt.config.js` com o mesmo nome.

`process.env ==`

Nota que o Nuxt usa o `definePlugin` do webpack para definir a variável de ambiente. Isto significa que o atual `process` ou `process.env` do Node.js não está nem disponível nem definido. Cada propriedade de `env` definida dentro do ficheiro `nuxt.config.js` é individualmente mapeado para `process.env.xxxx` e convertido durante a compilação.

Querendo dizer que, o `console.log(process.env)` imprimirá `{}` mas o `console.log(process.env.your_var)` continuará a imprimir seu valor. Quando o webpack compila o seu código, ele substitui todas instâncias de `process.env.your_var` com o valor que você tem definido, por exemplo: `env.test = 'testing123'`. Se você usar `process.env.test` em algum lugar do seu código, ele é de fato traduzido para `'testing123'`.

antes

```
if (process.env.test == 'testing123')
```

depois

```
if ('testing123' == 'testing123')
```

A propriedade serverMiddleware

Visto que a propriedade `serverMiddleware` é desligada da construção principal do Nuxt, variáveis de `env` definidas dentro do ficheiro `nuxt.config.js` não estão disponíveis lá.

[Go to TOC](#)

O contexto

O contexto fornece objetos/parâmetros adicionais do Nuxt para os componentes de Vue e está disponível em áreas especiais como `asyncData`, `fetch`, `plugins`, `middleware` e o `nuxtServerInit` do ciclo de vida do Nuxt.

Note que: "O contexto" que referimos aqui não é para ser confundido com o objeto `context` disponível nas Ações do Vuex. Os dois não estão relacionados.

```
function (context) {
  // Chaves Universais
  const {
    app,
    store,
    route,
    params,
    query,
    env,
    isDev,
    isHMR,
    redirect,
    error,
    $config
  } = context
  // No Lado do Servidor
  if (process.server) {
    const { req, res, beforeNuxtRender } = context
  }
  // No Lado do Cliente
  if (process.client) {
    const { from, nuxtState } = context
  }
}
```

Chaves universais

Essas chaves estão disponíveis em ambos lado do cliente e servidor.

app

`app` (*NuxtAppOptions*)

As opções da instância raiz do Vue que incluem todos os seus plugins. Por exemplo, quando estiver usando o `i18n`, você pode obter o acesso ao `$i18n` através do `context.app.i18n`.

store

`store` (*Memória do Vuex*)

Instância da Memória do Vuex. **Apenas disponível se a memória do vuex estiver definida.**

route

`route` (*Rota do Roteador do Vue*)

Instância da Rota do Roteador do Vue.

params

`params` (*Object*)

Pseudónimo de `route.params`.

query

`query` (*Object*)

Pseudónimo de `route.query`.

env

`env` (*Object*)

Variáveis de ambiente definidas dentro do ficheiro `nuxt.config.js`, consulte a [API de env](#).

isDev

`isDev` (*Boolean*)

Booleano para permitir você saber se está em modo de desenvolvimento, pode ser útil para cacheamento de algum dado em produção.

isHMR

`isHMR` (*Boolean*)

Booleano que permite saber se o método/intermediário é chamado a partir da substituição instantânea de módulo do webpack (é `true` *apenas no lado do cliente no modo de desenvolvimento*).

redirect

`redirect` (*Function*)

Use este método para redirecionar o usuário para um outra rota, o código de estado é usado no lado do servidor, o estado padrão será `302`. `redirect([status,] path [, query])`.

Exemplos:

```
 redirect(302, '/login')
 redirect({ name: 'slug', params: { slug: mySlug } })
 redirect('https://vuejs.org')
```

Consulte a [documentação do Roteador do Vue](#) para mais informações sobre a propriedade `location`.

Info

Não é possível usar o `redirect` ou `error` dentro do [plugin do Nuxt no lado do cliente](#) por causa da hidratação de erros (o conteúdo do cliente seria diferente daquilo que se esperaria do servidor).

Um maneira válida de dar a volta a isso seria usando `window.onNuxtReady(() => { window.$nuxt.$router.push('/your-route') })`

error

`error (Function)`

Use este método para mostrar a página de erro: `error(params)`. O `params` deve possuir as propriedades `statusCode` e `message`.

\$config

`$config (Object)`

A atual [configuração de tempo de execução](#).

Chaves do Lado do Servidor

Essas chaves apenas estão disponíveis no lado do servidor.

req

`req (http.Request)`

Requisição do servidor do Node.js. Se o Nuxt for usado como um intermediário, o objeto `request` pode ser diferente dependendo do framework que você estiver usando.

Não está disponível via `nuxt generate`.

Res

`res (http.Response)`

Resposta do servidor do Node.js. Se o Nuxt for usado como um intermediário, o objeto `response` pode ser diferente dependendo do framework que você estiver usando.

Não está disponível via `nuxt generate`.

beforeNuxtRender

`beforeNuxtRender(fn) (Function)`

Use este método para atualizar a variável `__NUXT__` renderizada no lado do cliente, a função `fn` (pode ser assíncrona) é chamada com o objeto `{ Components, nuxtState }`, consulte o [exemplo](#).

Chaves do Lado do Cliente

Essas chaves estão apenas disponíveis no lado do cliente.

from

`from` (*Rota do Roteador de Vue*)

A rota de onde navegamos (onde nós saímos).

nuxtState

`nuxtState` (*Object*)

O estado do Nuxt, útil para plugins o qual usa o objeto `beforeNuxtRender` para receber o estado do Nuxt no lado do cliente antes da hidratação. **Disponível apenas no modo `universal`**.

nuxt.render(req, res)

Você pode usar o Nuxt como um intermediário com `nuxt.render` para o seu servidor Node.js.

- Tipo: `Function`
- Argumentos:
 - `Request`
 - `Response`
- Retorna: `Promise`

Exemplo com o `Express`:

```
const { loadNuxt, build } = require('nuxt')

const app = require('express')()
const isDev = process.env.NODE_ENV !== 'production'
const port = process.env.PORT || 3000

async function start() {
  // Recebemos a instância do Nuxt
  const nuxt = await loadNuxt(isDev ? 'dev' : 'start')

  // Renderiza todas rotas com o Nuxt
  app.use(nuxt.render)

  // Construa apenas em modo de desenvolvimento com o recarregamento instantâneo
  if (isDev) {
    build(nuxt)
  }
  // Ouvir o servidor
  app.listen(port, '0.0.0.0')
  console.log(`Server listening on `localhost:${port}``)
}

start()
```

Warning

É recomendado chamar o `nuxt.render` no final do seu intermediário visto que ele lida com a renderização da sua aplicação web e não chamará o `next()`

[Go to TOC](#)

nuxt.renderRoute(route, context)

Renderiza uma rota específica com um dado contexto.

- Tipo: `Function`
- Argumentos:
 1. `String` : rota para renderizar
 2. *Opcional*, `Object`, dado contexto, chaves disponíveis: `req` e `res`
- Retorna: `Promise`
 - `html` : `String`
 - `error` : `null` ou `Object`
 - `redirected` : `false` ou `Object`

Este método deve ser usado principalmente para fins de teste bem como com o `nuxt.renderAndgetWindow`.

Warning

O `nuxt.renderRoute` deve ser executado depois do processo de construção em modo de produção.

```
const { loadNuxt, build } = require('nuxt')

async function start() {
  // Recebe a instância do Nuxt para iniciar (modo de produção)
  // Certifique de executar o `nuxt build` antes da execução deste roteiro
  const nuxt = await loadNuxt({ for: 'start' })

  const { html, error, redirected } = await nuxt.renderRoute('/')

  // O `html` será sempre uma sequência de caracteres

  // O `error` não nulo quando o esquema de erro é exibido, o formato do erro é:
  // { statusCode: 500, message: 'My error message' }

  // `redirected` não é `false` quando `redirect()` tem de ser usado em
  // `asyncData()` ou em `fetch()`
  // { path: '/other-path', query: {}, status: 302 }
}

start()
```

nuxt.renderAndGetWindow(url, options)

Recebe o `window` a partir de uma dada URL de uma aplicação Nuxt.

- Tipo: `Function`
- Argumento: `String`
 1. `String`: URL para renderizar
 2. `Object`, *Opcional*: opções
 - `virtualConsole`: `Boolean` (valor padrão: `true`)
- Retorna: `Promise`
 - Retorna: `window`

Warning

Este método foi feito para fins de teste.

Para usar esta função, você precisa instalar o `jsdom`:

```
npm install --save-dev jsdom
```

Exemplo:

```
const { loadNuxt } = require('nuxt')

async function init() {
  // Assumindo que você já tem construído o seu projeto
  const nuxt = await loadNuxt({ for: 'start' })
  await nuxt.listen(3000)
  const window = await nuxt.renderAndGetWindow('http://localhost:3000')
  // Exibe o cabeçalho `<title>`
  console.log(window.document.title)
  nuxt.close()
}

init()
```

O que se segue

Next

Consulte o [livro Glossário de Componentes](#)

[Go to TOC](#)

\$nuxt: O auxiliar de Nuxt

O `$nuxt` é um auxiliar desenhado para melhorar a experiência do usuário.

Para mais informações sobre o auxiliar do Nuxt consulte o [capítulo contexto e auxiliares no livro de Conceitos](#)

Verificador de conexão

- `isOffline`
 - Tipo: `Boolean`
 - Descrição: `true` quando a conexão de internet do usuário estiver fora de linha
- `isOnline`
 - Tipo: `Boolean`
 - Descrição: O oposto de `isOffline`

```
<template>
<div>
  <div v-if="$nuxt.isOffline">You are offline</div>
  <nuxt />
</div>
</template>
```

Atualizando os dados da página

- `refresh()`
 - Quando você quiser apenas atualizar os dados fornecidos pelo `asyncData` ou pelo `fetch`

```
<template>
<div>
  <div>{{ content }}</div>
  <button @click="refresh">Refresh</button>
</div>
</template>

<script>
export default {
  asyncData() {
    return { content: 'Created at: ' + new Date() }
  },
  methods: {
    refresh() {
      this.$nuxt.refresh()
    }
  }
}
</script>
```

Controlando a barra de carregamento

- `$loading`
 - Quando você quiser controlar programaticamente a barra de carregamento do Nuxt

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

Introdução aos Módulos do Nuxt

Entenda melhor o interior do Nuxt

O Nuxt tem uma arquitetura completamente modular que permite os desenvolvedores estenderem qualquer parte do núcleo do Nuxt usando uma API flexível.

Consulte o [Guia de Módulos](#) para informações mais detalhadas caso estiver interessado em desenvolver o seu próprio módulo.

Esta secção ajuda a se familiarizar com o interior do Nuxt e pode ser usada como uma referência para entender ele melhor enquanto estiver escrevendo seus próprios módulos.

O Núcleo

Essas classes são o coração do Nuxt e devem existir em ambos tempo de execução e construção.

A classe Nuxt

- A classe `Nuxt`
- Código-fonte: [core/nuxt.js](#)

A classe Renderer

- A classe `Renderer`
- Código-fonte: [vue-renderer/renderer.js](#)

A classe ModuleContainer

- A classe `ModuleContainer`
- Código-fonte: [core/module.js](#)

A Construção

Essas classes apenas são necessárias para o modo de construção ou desenvolvimento.

A classe Builder

- A classe `Builder`
- Código-fonte: [builder/builder.js](#)

Generator

- A classe `Generator`
- Código-fonte: [generator/generator.js](#)

O Geral

Utils

- Código-fonte: [utils/src](#)

Options

- Código-fonte: [config/options.js](#)

O Empacotamento e Uso

O Nuxt exporta todas as classes por padrão. Para importar elas:

```
import { Nuxt, Builder, Utils } from 'nuxt'
```

Padrões Gerais

Todas classes do Nuxt possuem uma referência a instância `nuxt` e as opções, desta maneira nós sempre teremos uma API consistente através das classes para acessar o `options` e o `nuxt`.

```
class SomeClass {
  constructor(nuxt) {
    super()
    this.nuxt = nuxt
    this.options = nuxt.options
  }

  someFunction() {
    // Nós temos acesso ao `this.nuxt` e ao `this.options`
  }
}
```

As classes são *conectáveis* assim elas devem registrar um plugin no contentor `nuxt` principal para registrar mais gatilhos.

```
class FooClass {
  constructor(nuxt) {
    super()
    this.nuxt = nuxt
    this.options = nuxt.options

    this.nuxt.callHook('foo', this)
  }
}
```

Assim desta maneira nós podemos prender dentro do módulo `foo`:

```
nuxt.hook('foo', foo => {
  // ...
})
```

[Go to TOC](#)

A classe Nuxt

Isto é, o contentor do núcleo que permite todos módulos e classes comunicarem umas com as outras. Todos os módulos tem acesso a instância do Nuxt usando `this.nuxt`.

- Código-fonte: [core/nuxt.js](#)

Os gatilhos

Nós podemos registrar gatilhos em certos eventos do ciclo de vida.

```
this.nuxt.hook('ready', async nuxt => {
  // O seu código personalizado vem aqui
})
```

| Plugin | Argumentos | Quando | | | `ready` | (nuxt) | O Nuxt está pronto para trabalhar (`ModuleContainer` e `Renderer` prontos). | | `error` | (error) | Um erro não manipulado sempre que estiver chamando gatilhos. | | `close` | (nuxt) | A instância do Nuxt está fechando graciosamente. | | `listen` | (server, {host, port}) | O servidor **interno** do Nuxt começa escutando. (Usando `nuxt start` ou `nuxt dev`). |

A classe Renderer

Esta classe está exportando um intermediário de `connect` o qual manipula e serve toda renderização no lado do servidor e as requisições de recurso.

-
- Código-fonte: [vue-renderer/renderer.js](#)

Os gatilhos

Nós podemos registrar os gatilhos certos eventos do ciclo de vida.

| Hook | Argumentos | Quando | | | `render:before` | (renderer, options) | Antes de definir o intermediário e os recursos para a classe Renderer, útil para sobrecarregar alguns métodos ou opções. | | `render:setupMiddleware` | (app) *instância de connect* | Antes do Nuxt adicionar pilha de intermediário. Nós podemos usar ele para registrar intermediário personalizado no lado do servidor. | | `render:errorMiddleware` | (app) *instância de connect* | Antes de adicionar intermediário de erro do Nuxt, útil para adicionar o seu próprio intermediário antes do uso do Nuxt. Consulte o [módulo Sentry](#) para mais informações. | | `render:resourcesLoaded` | (resources) | Chamado depois dos recursos para o renderizador serem carregados (manifesto de cliente, pacote do servidor, etc). | | `render:done` | (renderer) | O intermediário da renderização no lado do servidor e todos os recursos estiverem prontos (Renderer está pronto) | | `render:routeContext` | (context.nuxt) | *Toda vez que uma rota é renderizada no servidor e antes do gatilho render:route*. Chamado antes da serialização do contexto do Nuxt dentro do `window.__NUXT__`, útil para adicionar alguns dados que você pode requisitar no lado do cliente. | | `render:route` | (url, result, context) | *Toda vez que uma rota é renderizada no servidor*. Chamado antes de enviar de volta a requisição para o navegador. | | `render:routeDone` | (url, result, context) | *Toda vez que uma rota é renderizada no servidor*. Chamado depois da resposta ter sido enviada para o navegador. |

A classe ModuleContainer

- Código-fonte: [core/module.js](#)

Todos os módulos serão chamados dentro do contexto da instância de `ModuleContainer`.

Plugins intercetáveis

Nós podemos registar gatilhos em certos eventos do ciclo de vida.

```
nuxt.moduleContainer.plugin('ready', async moduleContainer => {
  // Faça isso depois de todos módulos estiverem prontos
})
```

Dentro do contexto de módulos nós podemos usar isso:

```
this.plugin('ready', async moduleContainer => {
  // Faça isso depois de todos módulos estiverem prontos
})
```

| Plugin | Argumentos | Quando | | | | `ready` | `moduleContainer` | Todos módulos dentro do ficheiro `nuxt.config.js` terem sido inicializados |

Métodos

O método addVendor (vendor)

Depreciado visto que `vendor` não é mais usado

Adiciona ao `options.build.vendor` e aplica filtro único.

addTemplate (template)

- template:** `String` ou `Object`
 - `src`
 - `options`
 - `fileName`

Renderiza o modelo dado usando o [modelo do lodash](#) durante a construção dentro do projeto `buildDir` (`.nuxt`).

Se o `fileName` não for fornecido ou o `template` não for uma sequência de caracteres, o nome do ficheiro alvo é padronizado para `[dirName].[fileName].[pathHash].[ext]`.

Este método retorna o objeto final `{ dst, src, options }`.

O método addPlugin (template)

- **template:** propriedades de Objeto (`src`, `options`, `fileName`, `mode`).

Regista um plugin usando o método `addTemplate` e adiciona ele no início do arranjo `plugins[]`.

```
this.addPlugin({
  src: path.resolve(__dirname, 'templates/foo.js'),
  fileName: 'foo.server.js' // [opcional] apenas inclui o pacote do servidor
  options: moduleOptions
})
```

Repare que: Você pode usar o `mode` ou os modificadores `.client` e `.server` com a opção `fileName` para usar o plugin apenas no lado do cliente ou servidor. (Consulte a secção `plugins` para conhecer todas opções disponíveis)

Se você escolher especificar um `fileName`, você pode também configurar um caminho personalizado para `fileName`, assim você pode escolher a estrutura da pasta dentro da pasta `.nuxt` no sentido de prevenir a colisão de nome:

```
{
  fileName: path.join('folder', 'foo.client.js'), // resultará em
  `._nuxt/folder/foo.client.js`
}
```

O método addServerMiddleware (middleware)

Empurra o intermediário dentro `options.serverMiddleware`.

O método extendBuild (fn)

Permite estender facilmente a configuração da construção do webpack pela encadeamento da função `options.build.extend`.

O método extendRoutes (fn)

Permite estender facilmente as rotas pelo encadeamento da função `options.build.extendRoutes`.

O método addModule (moduleOpts, requireOnce)

Função assíncrona

Regista um módulo. O `moduleOpts` pode ser uma sequência de caracteres ou um arranjo (`[src, options]`). Se o `requireOnce` for `true` e o módulo resolvido exportar o `meta`, ele evita o registo do mesmo módulo duas vezes.

O método requireModule (moduleOpts)

Função assíncrona

É um atalho para `addModule(moduleOpts, true)`

Os gatilhos

Nós podemos registrar gatilhos em certos eventos do ciclo de vida.

| Gatilho | Argumentos | Quando | | | `modules:before` | (`moduleContainer, options`) | Chamado antes da criação da classe `ModuleContainer`, útil para sobrecarregar métodos e opções. | | `modules:done` | (`moduleContainer`) | Chamado quando todos módulos tiverem sido carregados. |

A classe Builder

- Código-fonte: [builder/builder.js](#)

Os gatilhos

Nós podemos registrar gatilhos em certos eventos do ciclo de vida.

```
// Adiciona o gatilho para construção
this.nuxt.hook('build:done', (builder) => {
  ...
})
```

| Gatilho | Argumentos | Quando | | | build:before | (nuxt, buildOptions) | Antes da construção do Nuxt ter começado| | builder:prepared | (nuxt, buildOptions) | Os diretórios de construção tiverem sido criado| | builder:extendPlugins | (plugins) | Estiver gerando plugins | | build:templates | ({ templatesFiles, templateVars, resolve }) | Estiver gerando os ficheiros de modelos do .nuxt | | build:extendRoutes | (routes, resolve) | Estiver gerando rotas | | webpack:config | (webpackConfigs) | Antes da configuração dos compiladores | | build:compile | ({ name, compiler }) | Antes do webpack compilar (compiler é uma instância de Compiler do webpack), se estiver no modo universal, é chamado duas vezes com o nome 'client' e 'server' | | build:compiled | ({ name, compiler, stats }) | Construção do webpack tiver terminado | | build:done | (nuxt) | Construção do Nuxt tiver terminado |

[Go to TOC](#)

A classe Generator

-
- Código-fonte: [generator/generator.js](#)

Os gatilhos

```
generate: gatilhos:
```

```
| Gatilho | Argumentos | Quando | | | generate:before | (generator, generateOptions) | Antes da geração | | generate:distRemoved | (generator) | A pasta de destino for limpada | | generate:distCopied | (generator) | Ao copiar os ficheiros estáticos e construidos | | generate:route | ({ route, setPayload }) | Antes da geração da página, útil para payload dinâmico, consulte o #7422, disponível a partir da versão 2.13 do Nuxt | | generate:page | ({ route, path, html }) | Ao deixar o usuário atualizar o caminho e o HTML depois da geração | | generate:routeCreated | ({ route, path, errors }) | Ao guardar a página de sucesso gerada | | generate:extendRoutes | (routes) | Ao deixar usuário atualizar as rotas para geração | | generate:routeFailed | ({ route, errors }) | Ao guardar a página de falha gerada | | generate:done | (generator, errors) | A geração estiver terminada |
```

Usando o Nuxt Programaticamente

Você pode usar o Nuxt programaticamente para usar ele como um intermediário dando a você a liberdade de criação do seu próprio servidor para renderizar suas aplicações web.

Você pode querer usar o seu próprio servidor com o seu intermediário e sua API. É por isso que você pode usar o Nuxt programaticamente.

O construtor do Nuxt

Para ver a lista de opções para dar ao Nuxt, consulte a secção de configuração.

```
const { loadNuxt, build } = require('nuxt')

// Verifica se precisamos executar o Nuxt em modo de desenvolvimento
const isDev = process.env.NODE_ENV !== 'production'

// Prepare para usar a instância do Nuxt
const nuxt = await loadNuxt(isDev ? 'dev' : 'start')

// Ativar construção ao vivo e recarregamento em modo de desenvolvimento
if (isDev) {
  build(nuxt)
}

// Nós podemos usar o `nuxt.render(req, res)` ou `nuxt.renderRoute(route,
context)`
```

Você pode consultar os inicializadores [nuxt-express](#) ou [adonuxt](#) para iniciar rapidamente.

Registros de depuração

Se você quiser exibir os registros do Nuxt, você pode adicionar o seguinte ao topo do seu ficheiro:

```
process.env.DEBUG = 'nuxt:*
```

[Go to TOC](#)

O Gatilho Fetch

O gatilho `fetch` serve para requisição assíncrona de dados. Ele é chamado no lado do servidor quando estiver renderizando a rota, e no lado do cliente quando estiver navegando entre as rotas.

Em versões do Nuxt maiores ou igual 2.12

A versão 2.12 do Nuxt introduz um novo gatilho chamado `fetch` o qual você pode usar **dentro de qualquer um de seus componentes de Vue**. Use o `fetch` toda vez que você precisar receber dados **assíncronos**. O `fetch` é chamado no lado do servidor quando estiver renderizando a rota, e no lado do cliente quando estiver navegando entre as rotas.

Ele expõe o `$fetchState` no nível de componente:

- `$fetchState.pending` : `Boolean`, permite você mostrar um segurador de espaço quando o `fetch` estiver sendo chamado *no lado do cliente*.
- `$fetchState.error` : `null` ou `Error`, permite você mostrar uma mensagem de erro
- `$fetchState.timestamp` : `Integer`, é uma referência a data e hora da última requisição, útil para cacheamento com o `keep-alive`

Se você quiser chamar o gatilho `fetch` a partir do seu modelo, use:

```
<button @click="$fetch">Refresh</button>
```

ou método de componente:

```
// a partir de métodos de componente na secção de roteiro (script)
export default {
  methods: {
    refresh() {
      this.$fetch()
    }
  }
}
```

Você pode acessar o `contexto` do Nuxt dentro do gatilho `fetch` usando o `this.$nuxt.context`.

Opções

- `fetchOnServer` : `Boolean` ou `Function` (valor padrão: `true`), chama o `fetch()` quando o servidor estiver renderizando a página
- `fetchKey` : `String` ou `Function` (padroniza para identificador do escopo de componente ou nome de componente), uma chave (ou uma função que produz uma chave única) que identifica o resultado da requisição deste componente(disponível desde a versão 2.15 do Nuxt) [Mais informações estão disponíveis na PR original](#).
- `fetchDelay` : `Integer` (valor padrão: `200`), define em milissegundos o tempo mínimo de execução (para evitar piscadelas)

Quando o `fetchOnServer` for falso (`false` ou retornar `false`), o `fetch` será chamado apenas no lado do cliente e o `$fetchState.pending` retornará `true` quando o servidor estiver renderizando o componente.

```
<script>
  export default {
    data() {
      return {
        posts: []
      }
    },
    async fetch() {
      this.posts = await this.$http.$get('https://api.nuxtjs.dev/posts')
    },
    fetchOnServer: false,
    // vários componentes podem retornar o mesmo `fetchKey` e o Nuxt rastreará
    // ambos eles separadamente
    fetchKey: 'site-sidebar',
    // alternativamente, para mais controle, uma função pode ser passada com o
    // acesso à instância do componente
    // Ela será chamada dentro do gatilho `created` e não deve depender de um dado
    // requisitado
    fetchKey(getCounter) {
      // `getCounter` é um método que pode ser chamado para receber o próximo
      // número dentro de uma sequência
      // como parte de geração de uma `fetchKey` única.
      return this.someOtherData + getCounter('sidebar')
    }
  }
</script>
```

Next

Para mais informações sobre o Gatilho Fetch consulte o capítulo [Requisição de Dados](#) no livro Funcionalidades

A propriedade watchQuery

Watch query strings and execute component methods on change (asyncData, fetch, validate, layout, ...) Observa as sequência de caracteres de consulta e executa os métodos do componente sobre as mudanças (asyncData, fetch, validate, layout, ...)

- **Tipo:** Boolean ou Array ou Function (valor padrão: [])

Use a chave `watchQuery` para definir um observador para sequência de caracteres de consulta. Se a sequência de caracteres definida mudar, todos os métodos do componente (asyncData, fetch(context), validate, layout, ...) serão chamados. A observação está desativada por padrão para melhora de desempenho.

Se você quiser definir um observador para todas sequências de caracteres de consulta, defina `watchQuery: true`.

```
export default {
  watchQuery: ['page']
}
```

Você pode também usar a função `watchQuery(newQuery, oldQuery)` para ter observadores mais refinados.

```
export default {
  watchQuery(newQuery, oldQuery) {
    // Apenas execute os métodos do componente se sequência de caracteres de consulta antiga conter `bar`
    // e a sequência de caracteres de consulta nova conter `conter`
    return newQuery.foo && oldQuery.bar
  }
}
```

Warning

Aviso: O novo gatilho `fetch` introduzido na versão 2.12 não é afetado pelo `watchQuery`. Para mais informações consulte [ouvindo a mudanças na sequência de caracteres de consulta](#).

O método head

O Nuxt usa `vue-meta` para atualizar os `cabeçalhos` e `atributos do HTML` da sua aplicação.

- **Tipo:** `Object` ou `Function`

Use o método `head` para definir os marcadores do cabeçalho do HTML para a página atual.

```
<template>
  <h1>{{ title }}</h1>
</template>

<script>
  export default {
    data() {
      return {
        title: 'Hello World!'
      }
    },
    head() {
      return {
        title: this.title,
        meta: [
          // `hid` é usado como identificador único. Não use o `vmid` para isso
          visto que ele não funcionará
          {
            hid: 'description',
            name: 'description',
            content: 'My custom description'
          }
        ]
      }
    }
  }
</script>
```

Info

Para evitar marcadores de meta duplicados quando usado em componente filho, definir um identificador único com a chave `hid` para os seus elementos de meta ([leia mais](#)).

[Go to TOC](#)

A propriedade key

Define a propriedade `key` do componente `<router-view>` interno

- **Tipo:** `String` ou `Function`

A propriedade `key` é propagada dentro do `<router-view>`, o qual é útil para realizar transições dentro de uma página dinâmica e rota diferente. Chaves diferentes resultam em re-renderização dos componentes de página.

Há várias maneiras de definir a chave. Para mais detalhes, recorra a propriedade `nuxtChildKey` dentro do componente `nuxt`.

```
export default {
  key(route) {
    return route fullPath
  }
}
```

A propriedade layout

Todo ficheiro (primeiro nível) dentro do diretório layouts criará um esquema personalizado acessível com a propriedade layout dentro do componente de página.

- **Tipo:** `String` ou `Function` (valor padrão: `'default'`)

Use a chave `layout` dentro dos componentes de páginas para definir qual esquema usar:

```
export default {
  layout: 'blog',
  // ou
  layout(context) {
    return 'blog'
  }
}
```

A propriedade loading

A propriedade `loading` dá a você a opção de desativar a barra de progresso de carregamento padrão em uma página específica.

- **Tipo:** Boolean (valor padrão: `true`)

Por padrão, o Nuxt usa o seu próprio componente para mostrar uma barra de progresso entre as rotas.

Você pode desativar ou personalizar ela globalmente através da [opção de configuração do loading](#), mas também desativar ela para páginas específicas ao definir a propriedade `loading` para `false`:

```
<template>
  <h1>My page</h1>
</template>

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

A propriedade middleware

Definir o intermediário para uma página específica da aplicação.

- Tipo: `String` ou `Array` ou `Function`
 - Itens: `String` ou `Function`

Intermediário nomeado

Você pode criar intermediário nomeado ao criar um ficheiro dentro do diretório `middleware/`, o nome do ficheiro será o nome do intermediário.

```
export default function ({ store, redirect }) {
  // Se o usuário não estiver autenticado
  if (!store.state.authenticated) {
    return redirect('/login')
  }
}
```

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

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

Intermediário anônimo

Se você precisar usar um intermediário apenas para uma página específica, você pode usar diretamente uma função para isso (ou um arranjo de funções):

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

<script>
  export default {
    middleware({ store, redirect }) {
      // Se o usuário não estiver autenticado
      if (!store.state.authenticated) {
        return redirect('/login')
      }
    }
  }
</script>
```

[Go to TOC](#)

A propriedade transition da página

O Nuxt usa o componente `<transition>` para permitir você criar transições ou animações incríveis entre suas páginas.

- **Tipo:** `String` ou `Object` ou `Function`

Para definir uma transição personalizada para uma rota específica, simplesmente adicione a chave `transition` ao componente.

```
export default {
  // Pode ser uma Sequência de Caracteres
  transition: ''
  // Ou um Objeto
  transition: {}
  // ou uma Função
  transition (to, from) {}
```

Sequência de Caracteres

Se a chave `transition` for definida como uma sequência de caracteres, ela será usada como `transition.name`.

```
export default {
  transition: 'test'
```

O Nuxt usará essas definições para definir o componente conforme a seguinte:

```
<transition name="test"></transition>
```

Objeto

Se a chave `transition` for definida como um objeto:

```
export default {
  transition: {
    name: 'test',
    mode: 'out-in'
  }
}
```

O Nuxt usará essas definições para definir o componente conforme a seguinte:

```
<transition name="test" mode="out-in"></transition>
```

O objeto `transition` pode ter as seguintes propriedades:

| key | Type | Default | definition | | | | name | String | "page" | O nome da transição aplicado sobre todas transições de rotas. || mode | String | "out-in" | O modo de transição aplicado sobre todas rotas, consulte a [documentação do Vue.js](#) || css | Boolean | true | Se aplicar a classes de transição de CSS. Padroniza para true. Se definir para false, apenas acionará gatilhos de JavaScript registrado através dos eventos de componente. || duration | Integer | n/a | A duração (em milissegundos) aplicado sobre a transição, consulte a [documentação do Vue.js](#). || type | String | n/a | Especifica o tipo dos eventos de transição, para esperar determinar o tempo final da transição. Os valores disponíveis são 'transition' e 'animation'. Por padrão, ele detetará automaticamente o tipo que tiver duração mais longa. || enterClass | String | n/a | O estado de inicial da classe de transição (classe ou transição de entrada). Consulte a [documentação do Vue.js](#). || enterToClass | String | n/a | O estado final para a transição (estado final da classe ou transição de entrada). Consulte a [documentação do Vue.js](#). || enterActiveClass | String | n/a | A classe aplicada em toda duração da transição (depois de adicionar a classe enterClass e antes de adicionar a classe enterToClass). Consulte a [documentação do Vue.js](#). || leaveClass | String | n/a | O estado inicial da transição (classe ou transição de saída). Consulte a [documentação do Vue.js](#). || leaveToClass | String | n/a | O estado final da transição (classe ou transição de saída). Consulte a [documentação do Vue.js](#). || leaveActiveClass | String | n/a | A classe aplicada em toda duração da transição (depois de adicionar a classe leaveClass e antes de adicionar a classe leaveToClass). Consulte a [documentação do Vue.js](#).

Você pode também definir métodos dentro da propriedade transition da página, esses são para os [gatilhos de JavaScript](#):

- beforeEnter(el)
- enter(el, done)
- afterEnter(el)
- enterCancelled(el)
- beforeLeave(el)
- leave(el, done)
- afterLeave(el)
- leaveCancelled(el)

```
export default {
  transition: {
    afterLeave(el) {
      console.log('afterLeave', el)
    }
  }
}
```

Nota que: é também uma boa ideia adicionar css: false explicitamente para transições de somente JavaScript assim o Vue pode pular a deteção de CSS. Isto também previne as regras de CSS de acidentalmente interferirem com a transição.

Modo de Transição

A modo de transição padrão para páginas diferem do mesmo modo padrão no Vue.js. O modo transition é por padrão definido para out-in. Se você quiser executar transições de saída e entrada simultaneamente, você tem de definir o mode para uma sequência de caracteres vazia mode: ''.

```
export default {
  transition: {
    name: 'test',
    mode: ''
  }
}
```

Função

Se a chave `transition` for definida como uma função:

```
export default {
  transition(to, from) {
    if (!from) {
      return 'slide-left'
    }
    return +to.query.page < +from.query.page ? 'slide-right' : 'slide-left'
  }
}
```

As transições aplicadas na navegação são:

- / para /posts => slide-left,
- /posts para /posts?page=3 => slide-left,
- /posts?page=3 para /posts?page=2 => slide-right.

A propriedade scrollToTop

A propriedade `scrollToTop` permite você dizer ao Nuxt para rolar até o topo antes da renderização da página.

- **Tipo:** Boolean (valor padrão: `false`)

Por padrão, Nuxt rola até ao topo quando você ir para uma outra página, mas com rotas filhas, o Nuxt mantém a posição da rolagem. Se você quiser dizer ao Nuxt para rolar até ao topo quando estiver renderizando seu rota filha, defina `scrollToTop` para `true`:

```
<template>
  <h1>My child component</h1>
</template>

<script>
  export default {
    scrollToTop: true
  }
</script>
```

Inversamente, você pode também manualmente definir o `scrollToTop` para `false` nas rotas pais.

Se você quiser sobrescrever o comportamento padrão de rolagem do Nuxt, consulte a [opção scrollBehavior](#)

O método validate

O Nuxt permite você definir um método validador dentro de um componente de rota dinâmica.

- **Tipo:** `Function` ou `Async Function`

O `validate` é chamado toda vez antes de navegar para uma nova rota. Ele será chamado no lado do servidor uma vez (na primeira requisição para aplicação Nuxt) e no lado do cliente quando estiver navegando para rotas além. Este método recebe objeto de `contexto` como um argumento.

```
validate({ params, query, store }) {
  return true // se os parâmetros forem validos
  return false // impedirá o Nuxt de renderizar a rota e mostrará a página de erro
}
```

```
async validate({ params, query, store }) {
  // espere as operações
  return true // se os parâmetros forem validos
  return false // impedirá o Nuxt de renderizar a rota e mostrará a página de erro
}
```

Você pode também retornar promessas:

```
validate({ params, query, store }) {
  return new Promise((resolve) => setTimeout(() => resolve()))
}
```

O Nuxt permite você definir um método validador dentro do seu componente de rota dinâmica (Neste exemplo: `pages/users/_id.vue`).

Se o método `validate` não retornar `true`, o Nuxt carregará automaticamente a página de erro 404.

```
export default {
  validate({ params }) {
    // Deve ser um número
    return /^\d+$/.test(params.id)
  }
}
```

Você pode também verificar alguns dados dentro da sua `memória` por exemplo (preenchido pelo `nuxtServerInit` antes da ação):

```
export default {
  validate({ params, store }) {
    // Verifica se `params.id` é uma categoria existente
    return store.state.categories.some(category => category.id === params.id)
  }
}
```

Você pode também lançar erros esperados ou inesperados durante a execução da função `validate`:

```
export default {
  async validate({ params, store }) {
    // Lança um erro de código 500, erro do servidor interno com mensagem
    // personalizada
    throw new Error('Under Construction!')
  }
}
```

Colophon

This book is created by using the following sources:

- Nuxt - Português
- GitHub source: nuxt/nuxtjs.org
- Created: 2022-12-10
- Bash v5.2.2
- Vivliostyle, <https://vivliostyle.org/>
- By: @shinokada
- Viewer: <https://read-html-download-pdf.vercel.app/>
- GitHub repo: <https://github.com/shinokada/markdown-docs-as-pdf>
- Viewer repo: <https://github.com/shinokada/read-html-download-pdf>