

NUXT Docs - 日本語



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	59
• Features - Configuration	62
• Features - Loading	71
• Features - Nuxt components	75
• Features - Component discovery	82
• Directory structure - Nuxt	85
• Directory structure - Plugins	87
• Directory structure - Static	94
• Directory structure - Store	96
• Directory structure - Nuxt config	101
• Directory structure - Assets	108
• Directory structure - Components	113
• Directory structure - Content	116
• Directory structure - Dist	123
• Directory structure - Layouts	125
• Directory structure - Middleware	127
• Directory structure - Modules	129

• Directory structure - Pages	138
• Configuration glossary - Configuration alias	143
• Configuration glossary - Configuration extend plugins	145
• Configuration glossary - Configuration generate	146
• Configuration glossary - Configuration global name	153
• Configuration glossary - Configuration head	154
• Configuration glossary - Configuration hooks	155
• Configuration glossary - Configuration ignore	158
• Configuration glossary - Configuration loading	160
• Configuration glossary - Configuration loading indicator	162
• Configuration glossary - Configuration mode	163
• Configuration glossary - Configuration modern	164
• Configuration glossary - Configuration build	166
• Configuration glossary - Configuration modules	183
• Configuration glossary - Configuration modulesdir	185
• Configuration glossary - Configuration plugins	186
• Configuration glossary - Configuration render	188
• Configuration glossary - Configuration rootdir	195
• Configuration glossary - Configuration router	196
• Configuration glossary - Configuration runtime config	204
• Configuration glossary - Configuration server	205
• Configuration glossary - Configuration servermiddleware	207
• Configuration glossary - Configuration srkdir	210
• Configuration glossary - Configuration builddir	212
• Configuration glossary - Configuration ssr	213
• Configuration glossary - Configuration target	214
• Configuration glossary - Configuration telemetry	215
• Configuration glossary - Configuration transition	217
• Configuration glossary - Configuration vue config	219
• Configuration glossary - Configuration watch	220
• Configuration glossary - Configuration watchers	221
• Configuration glossary - Configuration cli	222
• Configuration glossary - Configuration css	224
• Configuration glossary - Configuration components	225
• Configuration glossary - Configuration dev	230
• Configuration glossary - Configuration dir	231

• Configuration glossary - Configuration env	232
• Internals glossary - Context	234
• Internals glossary - Nuxt render	238
• Internals glossary - Nuxt render route	239
• Internals glossary - Nuxt render and get window	240
• Internals glossary - \$nuxt	241
• Internals glossary - Internals	243
• Internals glossary - Internals nuxt	245
• Internals glossary - Internals renderer	246
• Internals glossary - Internals module container	247
• Internals glossary - Internals builder	250
• Internals glossary - Internals generator	251
• Internals glossary - Nuxt	252
• Components glossary - Fetch	253
• Components glossary - Watchquery	255
• Components glossary - Head	256
• Components glossary - Key	257
• Components glossary - Layout	258
• Components glossary - Loading	259
• Components glossary - Middleware	260
• Components glossary - Transition	261
• Components glossary - Scrolltotop	264
• Components glossary - Validate	265

インストール

ここでは、4つのステップで、Nuxt プロジェクトを立ち上げて実行する方法について説明します。

オンラインプレイグランド

CodeSandbox や StackBlitz で直接 Nuxt をオンラインで試すことができます：
:app-button[CodeSandbox で試す]
:app-button[StackBlitz で試す]

前提条件

- [Node.js](#) - 最新の LTS バージョンのインストールを推奨します。
- テキストエディタ、[VS Code](#) と [Vetur](#) 拡張、または [WebStorm](#) を推奨します。
- ターミナル、VS Code に [統合されたターミナル](#) または [WebStorm ターミナル](#) を推奨します。

create-nuxt-app の使用

video-player

src: <https://www.youtube.com/watch?v=wHkPjOmJTt0>

素早く始めるために、[create-nuxt-app](#) を使用できます。

yarn か npx のインストール (npx は npm 5.2 からデフォルト提供されています)、または npm v6.1 以上がインストールされていることを確認してください。

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

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

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

始めると、いくつかの質問が表示されます（名前、Nuxt オプション、UI フレームワーク、TypeScript、リンター、テストフレームワーク、その他）。すべてのオプションについては、[create-nuxt-app documentation](#) を確認してください。

一度全ての質問に答えると、全ての依存関係がインストールされます。次のステップは、プロジェクトフォルダに移動して、プロジェクトを起動します：

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

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

アプリケーションは、今 <http://localhost:3000> で動作しています。おつかれさまでした！

Info

Nuxt で始めるための他の方法は、[CodeSandbox](#) を使うことです。これは、Nuxt を使って素早く試したり、自分のコードを他の人と共有したりするのに最適な方法です。

手動インストール

video-player

src: https://www.youtube.com/watch?v=mKV_9AIG70E

1 つのファイルと 1 つのディレクトリを用意するだけで Nuxt プロジェクトをスクラッチで作れます。

今回の例では、ターミナルを使ってディレクトリとファイルを作成しますが、お好みのエディタで作成して構いません。

プロジェクトのセットアップ

始めるにはプロジェクトの名前をつけた空のディレクトリを作成し、作成したディレクトリに移動します：

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

<project-name> はご自身のプロジェクト名に置き換えてください。

`package.json` を作成します：

```
touch package.json
```

`package.json` をエディタで開き、以下の JSON の内容を入力します：

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

`scripts` に `npm run <command>` または `yarn <command>` で起動される Nuxt コマンドを定義します。

package.json ファイルとは何ですか？

`package.json` はプロジェクトにおける ID カードのようなものです。もし `package.json` ファイルがどんなものか知らない場合は [npm ドキュメント](#) を一読してください。

Nuxt のインストール

`package.json` を作成後、以下のように、`npm` または `yarn` 経由でプロジェクトに `nuxt` を追加します：

```
yarn add nuxt
```

```
npm install nuxt
```

このコマンドは `nuxt` を依存関係としてプロジェクトに追加します。`package.json` にも自動的に追加します。また、`node_modules` ディレクトリが作成され、インストールしたパッケージと依存関係が保存されます。

Info

`yarn.lock` または `package-lock.json` も作成され、プロジェクトにインストールされたパッケージの一貫したインストールと互換性のある依存関係を保証します。

最初のページを作成

Nuxt は `pages` ディレクトリ内にある全ての `*.vue` ファイルをアプリケーションのルートとして変換します。

プロジェクトに `pages` ディレクトリを作成します：

```
mkdir pages
```

そして、`pages` ディレクトリ内に `index.vue` ファイルを作成します：

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

このページを `index.vue` と名付けることが重要です。これはアプリケーションが開いた際に Nuxt がデフォルトで表示するページになります。これがホームページであり、`index` と名づけなればなりません。

`index.vue` ファイルをエディタで開き、次の内容を追加します：

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

プロジェクトの開始

ターミナルで以下のコマンドのどちらかを入力してプロジェクトを実行します：

```
yarn dev
```

```
npm run dev
```

Info

アプリケーションを開発モードで起動する場合は、`dev` コマンドを使用します。

今、アプリケーションは <http://localhost:3000> で動作しています。

ターミナルのリンクをクリックしてブラウザを開くと、前のステップでコピーした「Hello World」のテキストが表示されます。

Info

Nuxt を開発モードで起動すると、ほとんどのディレクトリでファイルの変更を監視しているので、新しいページを追加する際などにアプリケーションを再起動する必要はありません。

Warning

dev コマンドを実行すると、`.nuxt` フォルダが作成されます。このフォルダはバージョン管理の `ignore` 対象にしてください。ルートレベルで `.gitignore` ファイルを作成し、`.nuxt` を追加することで `.nuxt` フォルダ内にあるファイルを無視することができます。

ボーナスステップ

`pages` ディレクトリに `fun.vue` という名前のページを作成しましょう。

`<template></template>` を追加し、タグ内におもしろい文章を入れた見出しを入れます。

そして、ブラウザで新しいページ `localhost:3000/fun` にアクセスします。

Info

`more-fun` という名前のディレクトリを作成し、その中に `index.vue` ファイルを入れます。これは、`more-fun.vue` ファイルを作成したのと同じ結果が得られます。

[Go to TOC](#)

ルーティング

多くのウェブサイトでは単一のページではなく、複数ページを持っています。例えばホームページ、概要ページ、お問い合わせページなどです。これらのページを表示するにはルーターが必要です。

video-player

src: <https://www.youtube.com/watch?v=cKutrcn-hdE>

自動的に生成されるルート

多くのウェブサイトでは単一のページではなく、複数ページ（例えばホームページ、概要ページ、お問い合わせページなど）を持っています。これらのページを表示するにはルーターが必要です。そこで `vue-router` の出番です。Vue アプリケーションで動作させる際に、設定ファイル（例えば、`router.js`）を設定し、全てのルートを手動で追加しなければなりません。Nuxt では `pages` ディレクトリにある Vue ファイルに基づいて、`vue-router` の設定を自動的に生成します。つまり、ルーターの設定を書く必要はもうありません！ Nuxt はまた全てのルートに対して自動でコード分割を行います。

つまり、アプリケーションでルーティングを行うのに必要なことは `pages` フォルダに `.vue` ファイルを作成するだけです。

Next

詳細は [Routing](#) を参照してください。

ナビゲーション

アプリケーションのページ間を遷移するには、[NuxtLink](#) コンポーネントを使用する必要があります。このコンポーネントは Nuxt に含まれているため、他のコンポーネントのようにインポートする必要はありません。HTML の `<a>` タグに似ていますが、`href="/about"` の代わりに `to="/about"` を使用します。もし以前に `vue-router` を使ったことがある場合は、`<NuxtLink>` を `<RouterLink>` の代わりと考えることができます。

`pages` フォルダにある `index.vue` へのシンプルなリンク：

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

サイト内の全てのページへのリンクに `<NuxtLink>` を使ってください。他のウェブサイトへのリンクがある場合は `<a>` タグを使ってください。例は以下を参照してください：

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

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

Next

詳細は [NuxtLink コンポーネント](#) を参照してください。

[Go to TOC](#)

ディレクトリ構造

デフォルトの Nuxt のアプリケーション構造は小規模なものから大規模なものまで両方のアプリケーションにとって素晴らしい出発点を提供することを目的としています。アプリケーションは自由に構成でき、必要に応じて他のディレクトリを作成することができます。

プロジェクトにまだ存在しないディレクトリとファイルを作成してみましょう。

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

これらは Nuxt アプリケーションを構築する際に使用する主なディレクトリとファイルです。それについての説明は後述します。

Info

これらの名前のディレクトリを作成することで、Nuxt プロジェクトの機能を利用できるようになります。

ディレクトリ

pages ディレクトリ

`pages` ディレクトリにはアプリケーションのビューとルートが含まれています。前の章で学んだように、Nuxt はこのディレクトリ内の `.vue` ファイルをすべて読み込み、アプリケーションのルーターを作成します。

Next

[pages ディレクトリ](#) についてより理解する

components ディレクトリ

`components` ディレクトリにはページにインポートするすべての Vue.js のコンポーネントファイルを入れます。

Nuxt を使用すると、作成したコンポーネントを `.vue` ファイルに自動でインポートすることができます。（訳注：`nuxt.config.js` 内の）`components` を `true` に設定すると、Nuxt がスキャンして自動でインポートしてくれます。

Next

[components ディレクトリ](#) についてより理解する

assets ディレクトリ

`assets` ディレクトリにはスタイルや画像、フォントなどコンパイルされていないアセットを入れます。

Next

[assets ディレクトリ](#) についてより理解する

static ディレクトリ

`static` ディレクトリは直接サーバのルートに配置され、名前を保持しなければいけないファイル（例えば `robots.txt`）または 変更されない可能性の高いファイルが含まれています（例えば、`favicon` など）。

Next

[static ディレクトリ](#) についてより理解する

nuxt.config.js ファイル

`nuxt.config.js` ファイルは Nuxt の設定を行う唯一の場所です。モジュールの追加やデフォルトの設定を上書きしたい場合にここで変更を適用します。

Next

[nuxt.config.js ファイル](#) についてより理解する

package.json ファイル

`package.json` ファイルには、アプリケーションのすべての依存関係とスクリプトが含まれています。

プロジェクト構造についての詳細

`content`、`layouts`、`middleware`、`modules`、`plugins` そして `store` など、さらに役立つディレクトリやファイルがあります。これらは小規模なアプリケーションには必要ないのでここでは説明しません。

Next

すべてのディレクトリについての詳細は [Directory Structure book](#) を参照してください。

[Go to TOC](#)

コマンドと開発

Nuxt は開発と運用両方に役立つ一連のコマンドを用意しています。

video-player

src: <https://www.youtube.com/watch?v=hYdXzIGDIYA>

package.json を使う

これらのコマンドは `package.json` に記述する必要があります:

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

次に `yarn <command>` または `npm run <command>` を介してコマンドを起動できます (例: `yarn dev` / `npm run dev`)。

開発環境

`http://localhost:3000` で `hot module replacement` を使って開発モードで Nuxt を起動するためには、以下のコマンドを実行します:

`yarn dev`

`npm run dev`

コマンド一覧

ターゲットに応じて異なるコマンドを実行できます:

target: server (デフォルト値)

- **nuxt dev** - 開発サーバーを起動します。
- **nuxt build** - 本番用の webpack を使用してアプリケーションをビルドおよび最適化します。
- **nuxt start** - (`nuxt build` を実行した後に) 本番サーバーを起動します。Heroku や Digital Ocean などの Node.js ホスティングに使用します。

target: static

- **nuxt dev** - 開発サーバーを起動します。

- **nuxt generate** - (必要に応じて) アプリケーションをビルドし、すべてのルートを HTML ファイルとして生成し、`dist/` ディレクトリに静的にエクスポートします (静的ホスティングに使用されます)。
- **nuxt start** - 静的ホスティング (Netlify、Vercel、Surge など) と同じように `dist/` ディレクトリを提供します。デプロイ前のテストに最適です。

Webpack の設定を検査

`vue inspect` のように、nuxt がプロジェクトをビルドする際に使用する webpack の設定を検査することができます。

- **nuxt webpack [query...]**

引数:

- `--name` : 検査するバンドル名。(client、server、modern)
- `--dev` : 開発用の webpack の設定を検査します。
- `--depth` : 検査で出力される深さ。冗長に出力されないように、デフォルトでは 2 が設定されています。
- `--no-colors` : ANSI カラーを無効にします。(TTY が使用できない場合、またはファイルへパイプしている場合は、デフォルトで無効になります)

例:

- `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"`

プロダクション開発

Nuxt では、サーバーデプロイと静的デプロイのどちらかを選択することができます。

サーバー開発

SSR アプリケーションをデプロイするためにデフォルト値である `target: 'server'` を使います。

```
yarn build
```

```
npm run build
```

Nuxt はすべてのものが含まれる `.nuxt` ディレクトリを作成するので、ホスティングしているサーバーにデプロイする準備ができます。

Info

`.npmignore` または `.gitignore` に `.nuxt` を入れることをおすすめします。

一度アプリケーションがビルドされると、`start` コマンドを使ってアプリケーションの本番バージョンを確認することができます。

```
yarn start
```

```
npm run start
```

静的デプロイ(プリレンダリング)

Nuxt を使うと、どんな静的なホスティング上でもウェブアプリケーションをホストすることができます。

静的に生成されたサイトをデプロイするには、`nuxt.config.js` に `target: 'static'` を指定していることを確認してください (Nuxt v2.13 以上) :

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

```
yarn generate
```

```
npm run generate
```

Nuxt はすべてのものが含まれる `dist/` ディレクトリを作成するので、静的なホスティングサービスにデプロイする準備ができます。

Nuxt v2.13 ではリンクタグをクロールするクローラーがインストールされているので、`next generate` コマンドを使う際それらのリンクに基づいてルートを生成します。

Warning

警告: Nuxt v2.12 以下を使う場合、動的ルートは `generate` コマンドによって無視されます: [API Configuration generate](#)

Info

`nuxt generate` を使ってウェブアプリケーションを生成する場合、`asyncData` と `fetch` に与えられる `context` は `req` と `res` を持ちません。

エラー時の失敗 (Fail on Error)

ページエラーが発生した際にゼロ以外のステータスコードを返し、CI/CD のデプロイまたはビルドに失敗するようにするには `--fail-on-error` 引数を使います。

```
yarn generate --fail-on-error
```

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

この次は？

Next

[デプロイガイド](#)で人気のホストへのデプロイ例を参照してください。

おわりに

おめでとうございます！最初の Nuxt アプリケーションを作成し、これで Nuxter になったと思うかもしれません。しかし、Nuxt で学ぶことやできることはまだまだたくさんあります。いくつかおすすめのページを紹介します：

Next

[コンセプトを確認](#)

Next

[asyncDataの操作](#)

Next

異なる [レンダリングモード](#) の選択

`alert{type="star"}` Nuxt は気に入りましたか？ GitHub で[私たちのプロジェクトにスターをつけることを忘れないでください。](#)

[Go to TOC](#)

アップグレード

Nuxt のアップグレードは一瞬ですが、`package.json` のアップデートより複雑です

Nuxt v2.14 にアップグレードしてスタティックホスティングを使用したい場合、`generate` コマンドを正しく動作させるために `nuxt.config.js` ファイルに `target:static` を追加する必要があります。

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

はじめに

1. アップグレードしたいバージョンの[リリースノート](#)を確認し、そのリリースで追加の説明がないか確認してください。
2. `package.json` ファイルの `nuxt` パッケージに指定したバージョンを更新します。

このステップの後の手順は Yarn を使っているか npm を使っているかによって異なります。Yarn はテストが書かれている開発ツールなので、Nuxt を使うのに適したパッケージマネージャです。

Yarn

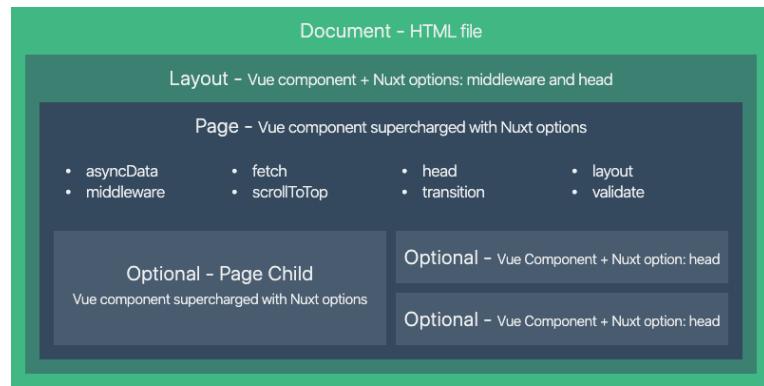
3. `yarn.lock` ファイルを削除します。
4. `node_modules` ディレクトリを削除します。
5. `yarn` コマンドを実行します。
6. インストールが完了しテストをしたら他の依存関係のアップグレードも検討してください。`yarn outdated` コマンドが使えます。

npm

3. `package-lock.json` ファイルを削除します。
4. `node_modules` ディレクトリを削除します。
5. `npm install` コマンドを実行します。
6. インストールが完了しテストをしたら他の依存関係のアップグレードも検討してください。`npm outdated` コマンドが使えます。

ビュー

ビュー (Views) セクションでは、Nuxt アプリケーション内の特定のルートのデータとビューを設定するために必要なことを全て説明しています。ビューは、アプリテンプレート、レイアウト、ページで構成されています。



Nuxt でのビューの構成

Nuxt でのビューの構成

ページ

すべてのページコンポーネントは Vue コンポーネントですが、Nuxt はアプリケーションの開発を可能な限り容易にするための特殊な属性や関数が追加されています。

```

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

<script>
  export default {
    head() {
      // このページ向けにメタタグを設定します
    }
    // ...
  }
</script>

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

ページコンポーネントのプロパティ

上の例の head プロパティのように、ページコンポーネントには多くのプロパティがあります。

Next

ページで使用できるすべてのプロパティの詳細については、[ディレクトリ構造のドキュメント](#)を参照してください。

レイアウト

サイドバーを含めたり、モバイルとデスクトップに異なるレイアウトを使用したりするなど Nuxt アプリの外観を変更したい場合にレイアウトは非常に役立ちます。

デフォルトレイアウト

`layouts` ディレクトリに `default.vue` ファイルを追加することでデフォルトレイアウトを定義することができます。これは特定のレイアウトを持たないすべてのページで使用されます。レイアウトに含める必要があるのは、ページコンポーネントをレンダリングする `<Nuxt />` コンポーネントだけです。

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

Next

詳細は [Nuxt コンポーネント](#) のコンポーネントの章を参照してください。

カスタムレイアウト

`layouts` ディレクトリに `.vue` ファイルを追加することで、カスタムレイアウトを作成できます。カスタムレイアウトを使用するには、そのレイアウトを使用したいページコンポーネントで `layout` プロパティを設定する必要があります。値は作成したカスタムレイアウトの名前になります。

`blog` レイアウトを作成するには、`blog.vue` ファイルを `layouts` ディレクトリ `layouts/blog.vue` に追加します：

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

Warning

実際にページコンポーネントを含めるレイアウトを作成するときは、必ず `<Nuxt />` コンポーネントを追加してください。

次に、そのレイアウトを使用したいページで 'blog' の値を持つ `layout` プロパティを使用します。

```
<template>
  <!-- テンプレート -->
</template>
<script>
  export default {
    layout: 'blog'
    // ページコンポーネントの定義
  }
</script>
```

Info

ページに `layout: 'blog'` のような layout プロパティを追加しない場合、`default.vue` レイアウトが使用されます。

エラーページ

エラーページは、エラーが発生したときに常に表示される ページコンポーネント です（サーバーサイドレンダリングでは発生しません）。

Warning

このファイルは `layouts` フォルダに配置されますが、ページとして扱う必要があります。

上記のように、このレイアウトは特別です。なぜなら、テンプレート内に `< Nuxt />` コンポーネントを含めるべきではないからです。このレイアウトは、エラー（`404`、`500` など）が発生したときに表示されるコンポーネントとしてみなさなければなりません。他のページコンポーネントと同様に、エラーページにも通常の方法でカスタムレイアウトを設定することができます。

`layouts/error.vue` ファイルを追加することでエラーページをカスタマイズすることができます：

```
<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' // エラーページ用のカスタムレイアウトを指定できます
}
</script>
```

ドキュメント: App.html

アプリテンプレートは、Nuxt アプリケーションのドキュメントの実際の HTML フレームを作成するために使用され、コンテンツだけでなく、ヘッドとボディに変数も注入します。このファイルは自動的に作成されるので、通常はほとんど変更する必要はありません。プロジェクトのソースディレクトリ（デフォルトではルートディレクトリ）に `app.html` ファイルを作成することで、スクリプトや条件付き CSS クラスを含むように Nuxt が使用する HTML アプリテンプレートをカスタマイズすることができます。

Nuxt で使用されるデフォルトのテンプレートは次のとおりです：

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

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

カスタムアプリテンプレートを使うユースケースの 1 つとして、IE 用の条件付き CSS クラスを追加します：

```
<!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>
```

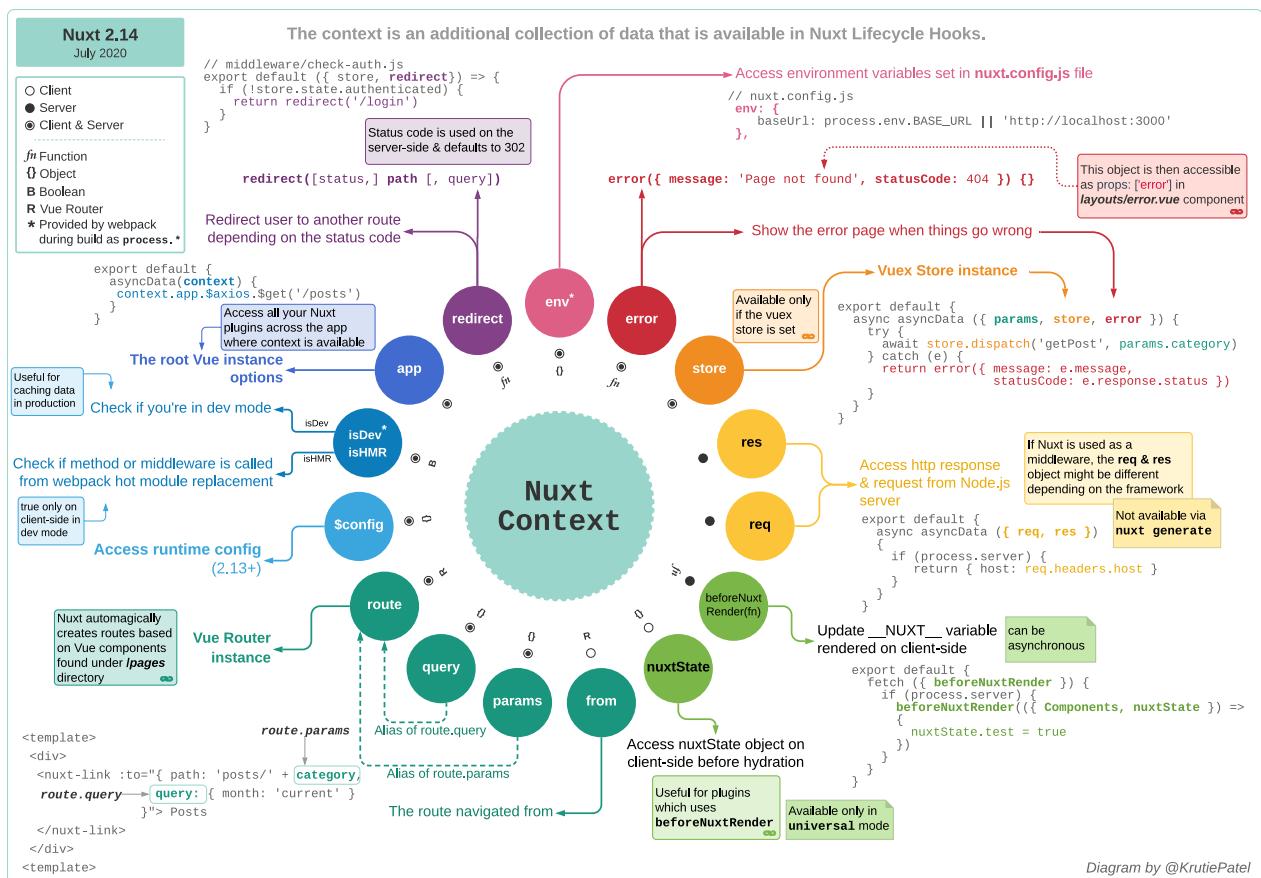
Info

JavaScript や CSS ファイルを `app.html` に追加することもできますが、代わりに `nuxt.config.js` を使用することをお勧めします！

[Go to TOC](#)

コンテキストとヘルパー

コンテキストは、アプリケーションへの現在のリクエスト (request) に関する追加の情報とオプション情報を提供します。



`context` オブジェクトは、`asyncData`、`plugins`、`middleware` そして `nuxtServerInit` といった特定の Nuxt 関数で利用できます。これは、アプリケーションへの現在のリクエストに関する追加の情報とオプション情報を提供します。

何よりもまず、コンテキストは Nuxt アプリケーションの他の部分（例えば Vuex ストアや基盤となる `connect` インスタンス）へのアクセスを提供するために使用されます。そのため、サーバーサイドと `store` で使用可能なコンテキストの `req` オブジェクトと `res` オブジェクトは常に使用可能です。しかし、時間が経つにつれコンテキストは他の多くの役立つ変数やショートカットで拡張されました。今では `development` モードの HMR（ホットモジュールリロード、またはリプレイスメント）機能、現在の `route`、ページの `params`、そして `query` にアクセスできるほか、コンテキストを介して環境変数にアクセスするオプションもあります。さらに、モジュール関数とヘルパーは、クライアントサイドとサーバーサイドの両方で使用できるようにコンテキストを通じて公開できます。

デフォルトで存在するすべてのコンテキストキー

```

function (context) { // asyncData, nuxtServerInit, ...
  // 常に利用可能
  const {
    app,
    store,
    route,
    params,
    query,
    env,
    isDev,
    isHMR,
    redirect,
    error,
    $config
  } = context

  // サーバーサイドでのみ利用可能
  if (process.server) {
    const { req, res, beforeNuxtRender } = context
  }

  // クライアントサイドでのみ利用可能
  if (process.client) {
    const { from, nuxtState } = context
  }
}

```

Warning

ここで参照する コンテキスト を [Vuex アクション](#)で利用可能な `context` オブジェクトや `nuxt.config.js` の `build.extend` 関数で利用可能な `context` オブジェクトと混同しないでください。これらは互いに関連していません！

他のコンテキストキーについての詳細は[コンテキストのドキュメント](#)を参照してください。

API クエリ用にページパラメータを使う

コンテキストは `context.params` を介してルートのとりうる動的パラメータを直接公開します。次の例では、URL の一部として動的ページパラメータを使用して、`nuxt/http` モジュールを介して API を呼び出します。`nuxt/http` モジュールは、`context.app` オブジェクトを介して利用できる独自の関数を公開できます。

また、潜在的なエラーを処理するために API の呼び出しを `try/catch` 構文でラップします。`context.error` 関数を使用すると、Nuxt のエラーページを直接表示して発生したエラーを渡すことができます。

```

export default {
  async asyncData(context) {
    const id = context.params.id
    try {
      // nuxtjs/http モジュールをここで使い context.app を介して公開します
      const post = await context.app.$http.$get(
        `https://api.nuxtjs.dev/posts/${id}`
      )
      return { post }
    } catch (e) {
      context.error(e) // スローしたエラーと一緒に nuxt エラーページを表示します
    }
  }
}

```

[ES6](#) を使用すると、この構文を使用してコンテキストオブジェクトを分解できます。アクセスしたいオブジェクトを渡すと、コンテキストという単語を使用せずにコードでそれらを使用できます。

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

    try {
      // nuxtjs/http モジュールをここで使い context.app を介して公開します
      const post = await $http.$get(`https://api.nuxtjs.dev/posts/${id}`)
      return { post }
    } catch (e) {
      error(e) // スローしたエラーと一緒に nuxt エラーページを表示します
    }
  }
}
```

代わりにクエリパラメータを使いたい場合は、[context.query.id](#) を使ってください。

ユーザをリダイレクトさせストアへアクセスする

[Vuex store](#) ストアへのアクセス (`store` ディレクトリを介して設定した場合) もコンテキストを介して可能です。これは、Vue コンポーネントで `this.$store` として扱うことができる `store` オブジェクトを提供します。さらに、コンテキストを介して公開されたヘルパーである `redirect` メソッドを使用して `authenticated` 状態が `falsey` な場合にユーザーをリダイレクトします。

```
export default {
  middleware({ store, redirect }) {
    // 分解した (destructuring) オブジェクトからキーを取得する
    const isAuthenticated = store.state.authenticated
    if (!isAuthenticated) {
      return redirect('/login')
    }
  }
}
```

Next

その他の例についての詳細は `redirect` メソッドのドキュメントを参照してください。

ヘルパー

コンテキスト内のショートカットに加えて、Nuxt アプリケーションには他の小さなヘルパーもあります。

\$nuxt: Nuxt ヘルパー

`$nuxt` はユーザー体験向上を目的としたヘルパーです。Vue コンポーネントでは `this.$nuxt` を介してアクセスでき、それ以外の場合はクライアント側で `window.$nuxt` を介してアクセスできます。

コネクションチェック

`$nuxt` ヘルパーはユーザーのインターネット接続があるかどうかをすばやく確認する方法を提供します。布尔値の `isOffline` と `isOnline` を公開します。これらを使用して例えば、ユーザーがオフラインになるとすぐにメッセージを表示できます。(以下の例)

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

ルートインスタンスへアクセスする

DX/UX（デベロッパーエクスペリエンス / ユーザーエクスペリエンス）機能を提供することに加え、`$nuxt` ヘルパーは他のすべてのコンポーネントからアプリケーションのルートインスタンスへのショートカットも提供します。しかし、それだけではありません。Vue コンポーネントの外部から `$axios` などのモジュールメソッドにアクセスするためのエスケープハッチとして使用できる `window.$nuxt` を介して `$nuxt` ヘルパーにアクセスすることもできます。賢明に使う必要があり最後の手段としてしてください。

ページデータのリフレッシュ

ユーザーの現在のページを更新したい場合、サーバーに再度アクセスして少なくとも Nuxt アプリケーション全体を再初期化する可能性があるのでページを完全にリロードさせたくないでしょう。リロードの代わりに `asyncData` または `fetch` によって提供されるデータのみを更新したい場合がよくあります。

これは `this.$nuxt.refresh()` を使って行えます！

```
<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>
```

ローディングバーの制御

`$nuxt` を使うと、Nuxt のローディングバーを `this.$nuxt.$loading` を介してプログラムで制御できます。

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

詳細については[ローディング機能のドキュメント](#)を参照してください。

onNuxtReady ヘルパー

Nuxt アプリケーションがロードされて準備ができた 後 、いくつかのスクリプトを実行する場合は、`window.onNuxtReady` 関数を使用できます。これはサイトのインタラクティブ化にかかる時間を増やすことなく、クライアントサイドで関数を実行したい場合に便利です。

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

プロセスヘルパー

Nuxt はグローバルな `process` オブジェクトに 3 つの布尔値 (`client`、`server` および `static`) を挿入します。これはアプリケーションがサーバーでレンダリングされたか、完全にクライアントでレンダリングされたかを判断し、静的サイトの生成を確認するのに便利です。これらのヘルパーはアプリケーション全体で利用でき、`asyncData` ユーザーランドコードで一般的に使用されています。

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

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

この例ではサーバーサイドレンダリングを使用し、ユーザーが直接ページにアクセスすると `renderedOn` は `'server'` と評価されます。ユーザーがアプリケーションの別の部分からページに遷移する場合（例えば `<NuxtLink>` をクリックする場合）クライアントに評価されます。

[Go to TOC](#)

サーバーサイドレンダリング

サーバーサイドレンダリング (SSR) はウェブページのレンダリングをブラウザの代わりにサーバー上で行う、画面の表示において有用なアプリケーションの機能です。サーバーサイドは完全にレンダリングされたページをクライアントに送信します。クライアントの JavaScript バンドルが引継ぎ Vue.js のアプリがハイドレーションできるようにします。

Node.js サーバー環境が必須

ウェブページをレンダリングするためには JavaScript の環境が必要です。

Vue.js のアプリケーションを実行するには Node.js サーバーの環境構築が必要です。

サーバーの拡張と制御

serverMiddleware でサーバーを拡張しルートを middleware で制御することができます。

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

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

サーバー環境とブラウザ環境

Node.js の環境を利用するため `req` や `res` といった Node.js のオブジェクトにアクセス可能です。また `window` や `document` といったブラウザ環境に帰属するオブジェクトにはアクセスできません。ただし `beforeMount` や `mounted` などのフックを使うと `window` や `document` を使うことができます。

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

Nuxt でのサーバーサイドレンダリングのステップ

ステップ 1: ブラウザからサーバーへ

ブラウザが最初のリクエストを送ると Node.js の内部サーバーへ到達します。Nuxt は HTML を生成し `asyncData`、`nuxtServerInit`、`fetch` といった関数を実行した結果と共にブラウザへ送り返します。フック関数も同様に実行されます。

ステップ 2: サーバーからブラウザへ

サーバーによって生成された HTML と共にレンダリングされたページをブラウザが受け取ります。その内容が表示され Vue.js のハイドレーションが作用し始めます。この工程の後、ページがインタラクティブになります。

ステップ 3: ブラウザからブラウザへ

`<NuxtLink>` によるページ間の遷移はクライアントサイドで行われるためブラウザをハード再読み込みをしない限りサーバーへリクエストを送りません。

警告

window または document undefined

これはサーバーサイドレンダリングによるものです。クライアントサイドでのみリソースをインポートするように指定する必要がある場合は `process.client` 変数を使用する必要があります。

たとえば `.vue` ファイルは次のようにになります:

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

iOS と電話番号

モバイル Safari の一部のバージョンでは、電話番号を自動的にリンクに変換します。これにより SSR コンテンツがウェブサイトのコンテンツと一致しなくなるため、`NodeMismatch` の警告が引き起こされます。そのためこれらの Safari バージョンでアプリが使用できなくなる可能性があります。

Nuxt ページに電話番号を含める場合、2 つのオプションがあります。

メタタグを使用して変換を停止する

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

電話番号をリンクにラップする

```
<!-- 電話番号の例: +7 (982) 536-50-77 -->

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

静的サイト生成

静的サイト生成を使用すると、ビルドフェーズでアプリケーションをレンダリングし、Netlify、GitHub pages、Vercelなどの静的ホスティングサービスにデプロイすることができます。これは、アプリケーションをデプロイするためにサーバーが必要ないことを意味します。

サイトを生成する

`target:static` でサイトをデプロイすると、すべての `.vue` のページが HTML と JavaScript ファイルに生成されます。API へのすべての呼び出しは、生成されたコンテンツ内の static と呼ばれるフォルダにキャッシュされ、クライアントサイドの遷移で API を呼び出す必要がなくなります。

ステップ 1: ブラウザから CDN へ

ブラウザが最初のリクエストを送信すると、CDN にヒットします。

ステップ 2: CDN からブラウザへ

CDN は、すでに生成された HTML、JavaScript、静的アセットをブラウザに送り返します。コンテンツが表示され、`Vue.js` のハイドレーションが作動してリアクティブになります。この処理の後、ページはインタラクティブになります。

ステップ 3: ブラウザからブラウザへ

ページ間の遷移は `<NuxtLink>` を使ってクライアントサイドで行われるので、CDN に再度ヒットすることはありません。そしてブラウザをハードリフレッシュしても、すべての API への呼び出しはすでにキャッシュされている静的フォルダから読み込まれます。

SPA フォールバック

`generate.exclude` プロパティを使用して生成から除外されたページは、シングルページアプリケーションにフォールバックします。そのため、これらのページは CDN には存在せず、ユーザーがそのページに遷移するとクライアントサイドのブラウザでレンダリングされます。

Next

より深く理解するには [generate プロパティ](#) を参照してください。

コンテンツを更新する

API から新しいコンテンツを取得するには、サイトを再生成する必要があります。ほとんどの静的サイトのホスティングプロバイダでは、git コマンドやブルリクエストで変更を `master` ブランチにプッシュすることで再生成することができます。

プレビューモード

プレビューモードは API や CMS を呼び出して、デプロイ前に変更点をライブで確認できるようにします。この機能を有効にする方法については [プレビューモード](#) を参照してください。

[Go to TOC](#)

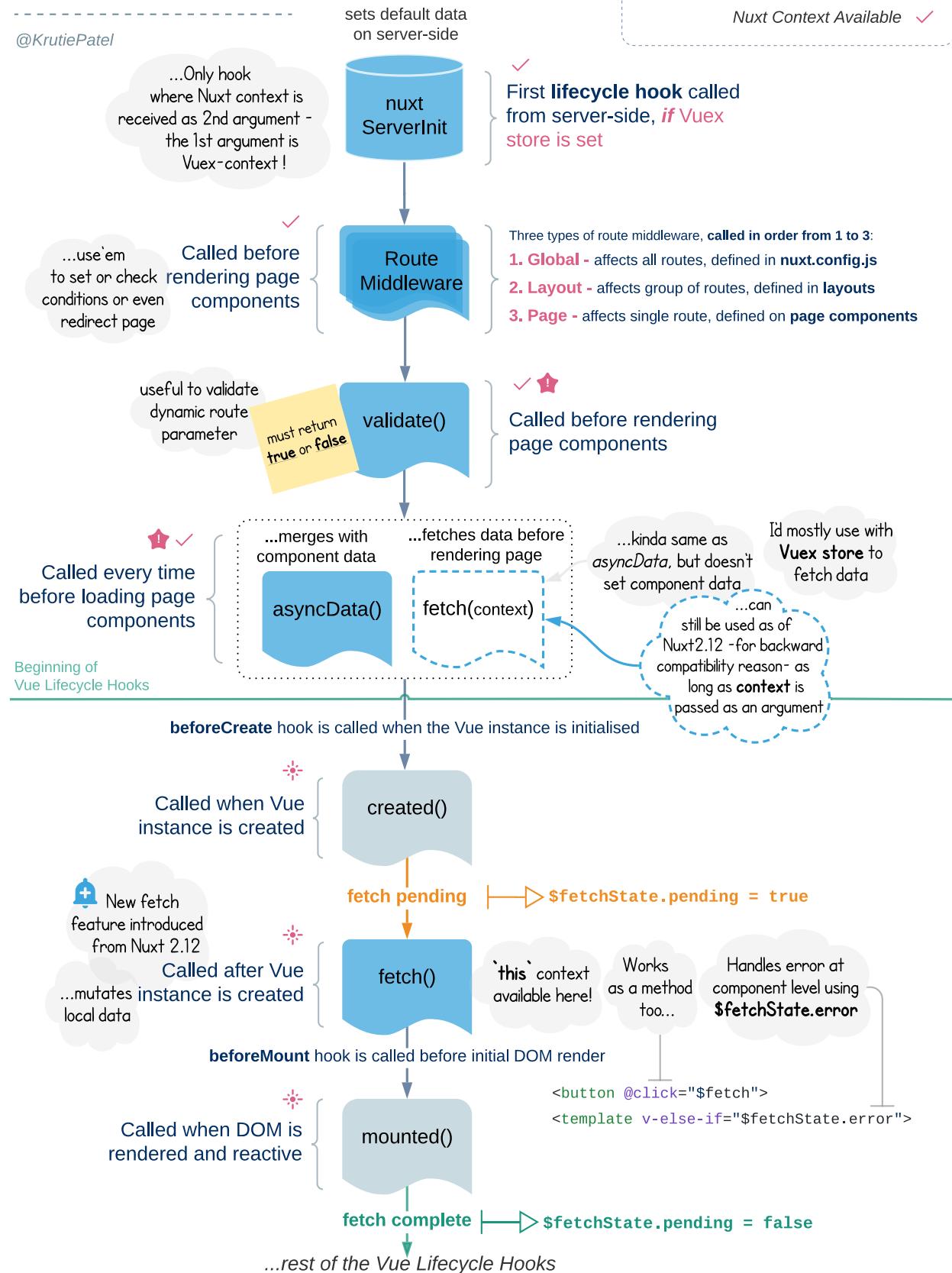
Nuxt ライフサイクル

どのツールを使う場合でも、ツールが内部でどのように機能するかを理解すると常に自信が持てるようになります。同じことが Nuxt にも当てはまります。

NUXT.JS LIFECYCLE HOOKS

Nuxt >= 2.12 - April 2020

@KrutiePatel



この章の目的は、フレームワークのさまざまな部分とそれらの実行順序、およびどのように連携するかについての概要を説明することです。

Nuxt ライフサイクルはアプリケーションがバンドルされ、チャンク化され、縮小されるビルドフェーズの後に何が起こるかを説明します。このフェーズの後に何が起こるかは、サーバーサイドのレンダリングが有効になっているかどうかによって異なります。有効になっている場合は、選択したサーバーサイドレンダリングのタイプによってさらに異なります：

動的な SSR (`nuxt start`)

または、静的サイト生成 (`nuxt generate`)

ライフサイクル

サーバー

SSR の場合、これらのステップはアプリケーションへの最初のリクエストごとに実行されます。

- サーバーを起動します (`nuxt start`)

静的サイトの生成を行う場合、サーバーステップはビルト時にのみ実行されますが、生成されるページごとに 1 回実行されます。

- 生成処理を開始します (`nuxt generate`)
 - Nuxt フック
 - サーバーミドルウェア
 - サーバーサイドの Nuxt プラグイン
 - `nuxt.config.js` で定義されている順序
- `nuxtServerInit`
 - ストアに事前にデータを詰めるためにサーバーサイドでのみ呼び出される Vuex アクション
 - 第 1 引数は **Vuex context**、第 2 引数は **Nuxt context** です
 - ここから他のアクションをディスパッチします → サーバーサイドの後続ストアアクションの「エントリポイント」のみです
 - `store/index.js` でのみ定義できます
- ミドルウェア
 - グローバルミドルウェア
 - レイアウトミドルウェア
 - ルートミドルウェア
- `asyncData`
- `beforeCreate` (Vue ライフサイクルメソッド)
- `created` (Vue ライフサイクルメソッド)

- 新しい fetch (上から下、ノード = 並列)
- 状態のシリアル化 (`render:routeContext` Nuxt フック)
- HTML レンダリングの開始 (`render:route` Nuxt フック)
- `render:routeDone` フック、HTML がブラウザに送信されたとき
- `generate:before` Nuxt フック
- HTML ファイルの生成
 - 完全な静的生成
 - 例えば静的ペイロードが抽出された場合
 - `generate:page` (HTML が編集可能)
 - `generate:routeCreated` (ルートが生成されます)
- `generate:done` すべての HTML ファイルが生成されたとき

クライアント

ライフサイクルのこの部分は選択した Nuxt モードに関係なくブラウザーで完全に実行されます。

- HTML を受け取り
- アセットのロード (例えば JavaScript)
- Vue ハイドレーション
- ミドルウェア
 - グローバルミドルウェア
 - レイアウトミドルウェア
 - ルートミドルウェア
- クライアントサイド Nuxt プラグイン
 - `nuxt.config.js` で定義されている順序
- `asyncData` (ブロッキング)
- `beforeCreate` (Vue ライフサイクルメソッド)
- `created` (Vue ライフサイクルメソッド)
- あたらしい fetch (上から下、ノード = 並列) (ノンブロッキング)
- `beforeMount` (Vue ライフサイクルメソッド)
- `mounted` (Vue ライフサイクルメソッド)

NuxtLink コンポーネントを使って遷移する

クライアント の章と同様に `<NuxtLink>` を介して遷移する場合だけすべてのステップがブラウザで発生します。さらにすべての ブロッキングタスクが実行されるまでページコンテンツは表示されません。

Info

`<NuxtLink>` のドキュメントで詳細を確認してください

- ミドルウェア
 - グローバルミドルウェア

- レイアウトミドルウェア
- ルートミドルウェア
- `asyncData` (ブロッキング)
- `asyncData` (ブロッキングまたは完全静的なペイロードをロードします)
- `beforeCreate & created` (Vue ライフサイクルメソッド)
- `fetch` (ノンブロッキング)
- `beforeMount & mounted`

この次は?

Next

[機能](#) について確認しましょう。

レンダリング

サーバーサイドでレンダリングされるサイトと静的サイト

サーバーサイドでレンダリングされるサイトは、ユーザーがページをリクエストするたびにサーバー上でレンダリングされるため、リクエストのたびにページを提供できるサーバーが必要です。

静的サイトは、サーバーサイドでレンダリングされるアプリケーションと非常によく似ていますが、主な違いとして静的サイトはビルド時にレンダリングされるため、サーバーを必要としません。あるページから別のページへの遷移はクライアントサイドで行います。

静的およびサーバーでのホスティングについては[デプロイターゲット](#)を参照してください。

```
export default {
  ssr: true // デフォルト値
}
```

Info

デフォルトでサーバサイドレンダリングが有効になっているので、サーバサイドレンダリングを有効にするために `ssr: true` を `nuxt` の設定に追加する必要はありません。

クライアントサイドレンダリングのみ

クライアントサイドレンダリングのみではサーバーサイドレンダリングをすることはできません。クライアントサイドレンダリングとは、JavaScript を使用してブラウザでコンテンツをレンダリングすることです。HTML からすべてのコンテンツを取得する代わりに、基本的な HTML ドキュメントと JavaScript ファイルを取得し、ブラウザを使用してサイトの残りの部分をレンダリングします。クライアントサイドレンダリングをするためには `ssr` を `false` に設定します。

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

Next

[ssr プロパティ](#)

[Go to TOC](#)

トランジション

Nuxt は `transition` コンポーネント を使って、ページ間を遷移する際のトランジション/アニメーションを行うことができます。

特定のルートに対してカスタムトランジションを設定するには、ページコンポーネントに `transition` キーを追加してください。

```
export default {
  // 文字列を指定できます
  transition: ''
  // またはオブジェクト
  transition: {}
  // または関数
  transition (to, from) {}
}
```

文字列

`transition` キーに文字列がセットされたときは `transition.name` として用いられます。

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

上のように設定すると、Nuxt はコンポーネントに次のようにセットします：

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

Warning

これは自動的に行われるため、ページやレイアウトに `<transition>` コンポーネントを追加する必要はありません。

これで、あとはトランジションのための新しいクラスを作るだけです。

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

オブジェクト

`transition` キーにオブジェクトがセットされたとき：

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

上のように設定すると、Nuxt はコンポーネントに次のようにセットします：

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

`transition` オブジェクトは `name`、`mode`、`css`、`duration` などの多くのプロパティを持つことができます。詳細は vue のドキュメントを参照してください。

ページ内の `transition` プロパティで関数を定義することもできます。 詳細は vue のドキュメントの [JavaScript フック](#) を参照してください。

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

トランジションモード

Warning

デフォルトのトランジションモードは Vue.js とは異なります。`transition` モードはデフォルトで `out-in` に設定されています。leaving と entering トランジションを同時に実行したい場合、モードプロパティに空文字列をセット (`mode: ''`) する必要があります。

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

関数

`transition` キーに関数がセットされたとき：

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

トランジションは各ページ遷移時に次のように適用されます：

/ から /posts => `slide-left` に, /posts から /posts?page=3 => `slide-left` に, /posts?page=3 から /posts?page=2 => `slide-right` に。

グローバルな設定

Nuxt のデフォルトのトランジション名は "page" です。 アプリケーションのすべてのページでフェードさせるトランジションを追加するには、ルーティング全体に適用されている CSS ファイルが必要です。

`assets/main.css` 内にグローバルな CSS を書きます：

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

`nuxt.config.js` ファイルの `css` 配列に CSS ファイルのパスを追加します：

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

構成設定

layoutTransition プロパティ

レイアウトトランジションのデフォルト設定を指定するために使われます。

デフォルトの設定は以下の通りです：

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

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

デフォルトの設定を変更したい場合、`nuxt.config.js` ファイルに以下のように記述してください。

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

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

pageTransition プロパティ

デフォルトの設定は以下の通りです：

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

デフォルトの設定を変更したい場合、nuxt.config.js ファイルに以下のように記述してください。

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

ページトランジションの名前を変更したい場合、css のクラス名も変更する必要があります。

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

プレビューモード

プレビューモードを使った静的ホスティング向けのライブプレビュー

Nuxt とフルスタティックモジュールを使うことで、API や CMS を呼び出すライブプレビューをすぐに利用することができます。そして、デプロイする前に変化を確認することができます。

Warning

`target:static` を利用しているときのみ使用可能です

内部では `$nuxt.refresh` を使っており、クライアントサイドで `nuxtServerInit` や `asyncData`、`fetch` を呼び出しているため、プレビューモードでは自動的にページのデータが更新されます。

ライブプレビューを有効化するためには、次のプラグインを追加する必要があります：

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

Warning

`enablePreview` はプラグインの `context` オブジェクトでのみ利用可能です。プレビューはクライアントサイドで処理されるため、プラグイン(今回の例では `preview.client.js`)はクライアントでのみ実行される必要があります。

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

一度そのプラグインを追加すれば、すぐにサイトを生成し配信することができます。

```
yarn generate
yarn start
```

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

そして以下のクエリパラメータを確認したいページの最後に追加することでプレビューを見るることができます：

```
?preview=true
```

Warning

`enablePreview` は `yarn dev` ではなく `yarn start` を用いてローカルでテストしてください。

まだ生成されていないページをプレビューする場合

まだ生成されていないページは API 上に存在しますが静的なページとして生成されてはいないため、SPA フォールバックは 404 ページを表示する前に API を呼び出します。

もし validate フックを設定している場合、プレビューモードで 404 ページへリダイレクトしないように validate フックを修正する必要があるでしょう。

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

enablePreview にデータを渡す場合

`enablePreview` 関数にデータを渡すことができます。そのデータは `$preview` コンテキストヘルパーと `this.$preview` で利用できます。

この次は？

Next

[ディレクトリ構造のドキュメント](#)について確認しましょう。

[Go to TOC](#)

デプロイターゲット

静的ホスティング

Nuxt は静的サイトジェネレータとしても機能します。Nuxt アプリケーションを静的にレンダリングし、サーバーなしでユニバーサルアプリケーションのいいところが全て使えます。`nuxt generate` コマンドはウェブサイトの静的バージョンを生成します。ルートごとに HTML を生成し、それを `dist/` ディレクトリ内に配置します。これによりパフォーマンスと SEO が向上し、オフラインサポートが向上します。

Info

[Nuxt Crawler](#) のおかげで動的なルートも生成されます。

For static sites the target of `static` needs to be added to your `nuxt.config` file.

```
export default {
  target: 'static' // デフォルトは 'server'
}
```

target を `static` にした状態で `nuxt dev` を実行すると、開発者の体験は向上するでしょう：

- `context` から `req` と `res` を削除します
- クライアントサイドレンダリングの 404、エラー、リダイレクトをフォールバックします [SPA フォールバックを参照](#)
- サーバーサイドレンダリングでは常に `$route.query` と `{}` は等しくなります
- `process.static` は `true` になります

Info

また、モジュールの製作者がユーザーターゲットに応じてロジックを追加するために `process.target` を公開しています。

サーバーホスティング

サーバーホスティングとは、Node.js サーバー上で Nuxt を動かすことです。ユーザーがあなたのページを開くと、ユーザーのブラウザはサーバーからそのページをリクエストします。Nuxt はそのリクエストを処理し、ページをレンダリングして、結果のページをすべてのコンテンツとともに送り返します。

生成時にあらかじめ HTML をレンダリングするのではなく、リクエストごとにレンダリングしたい場合や、[serverMiddleware](#)が必要な場合は、サーバーホスティングが必要かもしれません。

Info

`ssr: false` を指定してサーバーホスティングで Nuxt を実行することはできますが、Nuxt は各ページの HTML を完全にはレンダリングしません。サーバーミドルウェアは必要だが、サーバーサイドで完全にレンダリングされた HTML は必要ないという場合に、このオプションを選択することができます。

サーバーホスティングの場合は、デフォルトである `target: 'server'` が使用されます。アプリケーションをビルドするには、`build` コマンドを使用します。

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

[Go to TOC](#)

ファイルシステムルーティング

Nuxt は pages ディレクトリにある Vue ファイルの木構造に基づいて、vue-router の設定を自動的に生成します。pages ディレクトリに .vue ファイルを作成すると、特別な設定をせずとも基本的なルーティングが動作するようになります。

動的なルーティングやネストされたルートを生成したり、ルーターのプロパティをより細かく設定したりする必要がある場合があるでしょう。この章では、ルーターを最大限に活用するために知るべきことすべてを説明します。

Info

Nuxt は設定をしなくても自動的にコード分割を行います。

Info

ページ間を遷移するためには [NuxtLink コンポーネント](#) を使ってください。

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

ルーティングの基礎

下記のような木構造のとき：

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

自動的に以下が生成されます：

```
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'
    }
  ]
}
```

動的なルーティング

ユーザー や ブログ記事の一覧を API から取得する場合など、ルートの名前を決定することができない場合があります。私たちはこれを動的なルーティングと呼びます。動的なルーティングを生成するには、`.vue` ファイル名またはディレクトリ名の前にアンダースコア(`_`)を付ける必要があります。ファイルやディレクトリには好きな名前をつけることができますが、アンダースコアのプレフィックスを必ず付ける必要があります。

下記のような木構造のとき：

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

自動的に以下が生成されます：

```
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

`users-id` と名付けられたルートに `:id?` というパスがありますが、これはこの `:id` が必須ではないことを表します。もし必須にしたい場合は、代わりに `users/_id` ディレクトリ内に `index.vue` ファイルを作成してください。

Info

Nuxt \geq v2.13 ではクローラーがインストールされており、リンクタグをクロールしてそれらのリンクに基づいた動的なルートを生成します。しかしシークレットページのようなリンクが存在しないページがある場合、それらの動的なルートは手動で生成する必要があります。

Next

静的なサイトのために動的ルーティングを生成する

ルートパラメーターへのローカルアクセス

ローカルページやコンポーネント内の現在のルートパラメータは、`this.$route.params.{parameterName}` を参照することができます。例えば、動的なユーザーページ (`users/_id.vue`) があり、ユーザーまたはプロセス情報を読み込むために `id` パラメーターにアクセスしたい場合、次のような変数にアクセスできます：
`this.$route.params.id`

ネストされたルート

Nuxt は vue-router の子ルートを使ってルートをネストさせることができます。ネストされたルートの親コンポーネントを定義するには、子ビューを含むディレクトリと同じ名前の Vue ファイルを作成する必要があります。

Warning

親コンポーネント (`.vue` ファイル) 内に **NuxtChild コンポーネント** を含めることを忘れないでください。

下記のような木構造のとき：

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

自動的に以下が生成されます：

```
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'
        }
      ]
    }
  ]
}
```

動的でネストされたルート

あまり頻繁に使う場面はありませんが、Nuxt は動的な親ルーティングの中に動的な子ルーティングを持つことが可能です。

下記のような木構造のとき：

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

自動的に以下が生成されます：

```
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'
            }
          ]
        }
      ]
    }
  ]
}
```

未知の動的でネストされたルート

もし URL 構造の深さが不明な場合は、ネストされたパスに動的にマッチさせる `_.vue` ファイルを使うことができます。これはより詳細なルートにマッチしなかったリクエストをハンドリングします。

下記のような木構造のとき：

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

次のようにリクエストをハンドリングします：

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

Info

404 ページのハンドリングは `_.vue` ページのロジックに依存します。

ルーターの拡張

Nuxt のルーティングを拡張する方法はいくつかあります：

- `router-extras-module` を使ってページのルートパラメータをカスタマイズする
- `@nuxtjs/router` コンポーネントを使い、独自の `router.js` で Nuxt router を上書きする
- `nuxt.config.js` 内で `router.extendRoutes` プロパティを使う

router プロパティ

`router` プロパティを使って Nuxt のルーター (vue-router) をカスタマイズできます。

```
export default {
  router: {
    // Nuxt router をカスタマイズする
  }
}
```

Base:

アプリケーションのベース URL です。例えばシングルページアプリケーション全体を `/app/` 配下で配信したい場合、`base` に `/app/` を設定します。

Next

[Router Base プロパティ](#)

extendRoutes

Nuxt によって生成されるルーティングを拡張したい場合があるでしょう。それは `extendRoutes` オプションで実現できます。

独自のルートを追加する例：

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

ルートをソートしたい場合、`@nuxt/utils` の `sortRoutes(routes)` 関数を使うことができます：

```
import { sortRoutes } from '@nuxt/utils'
export default {
  router: {
    extendRoutes(routes, resolve) {
      // ここにルートを追加します ...

      // それらをソートします
      sortRoutes(routes)
    }
  }
}
```

Warning

ルートのスキーマは `vue-router` のスキーマを尊重すべきです。

Warning

名前付きビューを使うルートを追加する場合、対応する名前付き `components` の `chunkNames` を追加することを忘れないでください。

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

```
        modal: 'components/modal'  
    }  
})  
}  
}  
}
```

Next

extendRoutes プロパティ

fallback

`history.pushState` をサポートしていないブラウザにおいて、モードが `history` に設定されている場合に、ルーターを `hash` モードにフォールバックするかどうかを制御します。

Next

fallback プロパティ

mode

ルーティングのモードを設定します。サーバーサイドレンダリングのため、この設定を変更することは非推奨です。

Next

mode プロパティ

parseQuery / stringifyQuery

カスタムクエリ構文解析関数 / 文字列化関数を提供します。

Next

parseQuery / stringifyQuery プロパティ

routeNameSplitter

Nuxt が使うルート名の区切り文字を変更したい場合があるでしょう。それは設定ファイル内の `routeNameSplitter` オプションを使用して実現できます。`pages/posts/_id.vue` というページファイルがあるとします。Nuxt はプログラムに従ってルート名を生成します。この場合は `posts-id` です。`routeNameSplitter` の設定を `/` に変更することによって `posts/id` へ変更されます。

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

scrollBehavior

`scrollBehavior` オプションを使って、ページ間遷移のスクロール位置について独自の振る舞いを定義することができます。このメソッドはページがレンダリングされるたびに毎回呼び出されます。

Next

詳細は [vue-vue-router のスクロールの振る舞いのドキュメント](#) を参照してください。

v2.9.0 以降で利用可能：

ファイルを利用してルーターの scrollBehavior を上書きすることができます。このファイルは app フォルダに配置する必要があります。

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

すべてのルートに置いて強制的にトップまでスクロールさせる例：

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

Next

[Nuxt のデフォルトの router.scrollBehavior.js ファイル](#)

Next

[scrollBehavior プロパティ](#)

trailingSlash

v2.10 以降で利用可能：

このオプションを true に設定した場合、すべてのルートの末尾にスラッシュが追加されます。false に設定した場合、それらは取り除かれます。

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

Warning

このオプションを準備なしに設定しないでください。徹底的にテストする必要があります。`router.trailingSlash` に `undefined` (デフォルトの値) 以外の値を設定した場合、どちらかのルートは機能しなくなります。したがって、301 リダイレクトは適切に行われ、内部リンクは正しく適応される必要があります。`trailingSlash` を `true` に設定する場合、`example.com/abc/` のみが機能し、`example.com/abc` は機能しません。`false` に設定する場合はその逆になります。

Next

[trailingSlash プロパティ](#)

[Go to TOC](#)

データの取得

Nuxt では、API からデータを取得する方法が 2 つあります。fetch メソッドまたは asyncData メソッドを使用できます。

Nuxt はコンポーネントの `mounted()` フックでデータを取得するなど、クライアントサイドアプリケーションにおける従来の Vue のデータの取得パターンをサポートしています。しかしユニークなアプリケーションでは、サーバーサイドレンダリング中にデータをレンダリングするために Nuxt 特有のフックを使う必要があります。これにより、必要なデータがすべて存在する状態でページをレンダリングすることができます。

Nuxt には、データを非同期に読み込むためのフックが 2 つあります。

- `asyncData`。このフックは、ページコンポーネントでのみ使用できます。`fetch` とは異なり、クライアントサイドレンダリング中にローディングプレースホルダーを示しません。そのかわり、フックが解決するまでルートナビゲーションをブロックし、失敗した場合はエラーページを表示します。
- `fetch` (Nuxt 2.12 以降)。どのコンポーネントにでも配置することができ、(クライアントサイドレンダリング中の)ローディング状態やエラーをレンダリングするショートカットを提供します。

これらのフックは、あなたが選ぶどのようなデータ取得ライブラリとでも一緒に使うことができます。HTTP API ヘリエストを送るために `@nuxt/http` または `@nuxt/axios` を使用することをおすすめします。認証ヘッダーの設定のような、これらのライブラリのより詳しい情報はそれぞれのドキュメントで見つけることができるでしょう。

Info

ミックスイン内で `fetch` または `asyncData` を定義し、それをコンポーネントやページでも定義すると、ミックスイン関数は上書きされ呼び出されません。

fetch フック

Info

Nuxt 2.12 より前は、ページコンポーネントに対してのみ機能し、コンポーネントのインスタンスにアクセスできない別の `fetch` フックがありました。

もし `fetch()` が `context` 引数を受け取るように書かれているなら、それはレガシーな `fetch` フックとして扱われます。この機能は非推奨なので、`asyncData` または [無名ミドルウェア \(anonymous middleware\)](#) に置き換えてください。

`fetch` はサーバーサイドレンダリングではコンポーネントのインスタンスが作成された後に呼び出され、クライアントサイドでは遷移時に呼び出されるフックです。`fetch` フックは下記のタイミングで解決される promise を（明示的に、または `async/await` を使って暗黙的に）返却するべきです：

- サーバー上では、初期ページのレンダリングよりも前に
- クライアント上では、コンポーネントのマウントよりも後

Info

静的ホスティングでは、fetch フックはページ生成時にのみ呼び出され、その結果はクライアントで使用するためにキャッシュされます。キャッシュの競合を避けるために、コンポーネントの名前を指定するか、代わりにユニークな fetchKey の実装を提供する必要があるかもしれません。

使い方

データの取得 (Fetching)

fetch フック内では、`this` を介してコンポーネントインスタンスにアクセスできます。

Info

更新したいプロパティがすでに `data()` で宣言されていることを確認してください。宣言されている場合、取得したデータをこれらのプロパティに割り当てることができます。

fetch 動作の変更

`fetchOnServer: Boolean` または `Function` (デフォルト: `true`)。サーバーがページをレンダリングする際に `fetch()` を呼び出します。

`fetchKey: String` または `Function` (デフォルトはコンポーネントのスコープ ID またはコンポーネント名)、コンポーネントの `fetch` 結果を識別するキー (または一意のキーを生成する関数)。(Nuxt 2.15 以降で有効、詳細情報の [GitHub プルリクエスト](#))。サーバーでレンダリングされたページをハイドレートする際、このキーはサーバーサイド `fetch()` の結果をクライアントサイドのコンポーネントデータにマッピングするために使用されます。 詳細は [PR を参照してください](#)。

`fetchDelay: Integer` (デフォルト: `200`)。最小実行時間をミリ秒単位で設定します (高速な画面ちらつきを防ぐため)。

`fetchOnServer` が `false` (`false` または `false` と評価される値) の場合、`fetch` はクライアントサイドでのみ呼び出され、サーバーでコンポーネントをレンダリングしている間は `$fetchState.pending` は `true` を返します。

```
export default {
  data: () => ({
    posts: []
  }),
  async fetch() {
    this.posts = await this.$http.$get('https://api.nuxtjs.dev/posts')
  },
  fetchOnServer: false,
  // 複数のコンポーネントは同じ `fetchKey` を返すことができ、Nuxt はそれら両方を別々に追跡します
  fetchKey: 'site-sidebar',
  // 他の手段として、もっとコントロールしたい場合は、コンポーネントのインスタンスにアクセスできる関数を渡すこともできます
  // これは `created` で呼び出され、フェッチされたデータに依存してはいけません
  fetchKey(getCounter) {
    // getCounterは、ユニークな fetchKey を生成する際に、シーケンス内の次の番号を
    // 取得するために呼び出すことができるメソッドです
    return this.someOtherData + getCounter('sidebar')
  }
}
```

fetch 状態へのアクセス

`fetch` フックは、以下のプロパティを持つ `this.$fetchState` をコンポーネントレベルで公開します：

- `pending` は `fetch` がクライアントサイドで呼び出されたときにプレースホルダーを表示するかを表す `Boolean` です
- `error` は `null` もしくは `fetch` フックで発生した `Error` です
- `timestamp` は最後に `fetch` した時刻で、`keep-alive` によるキャッシングのために使えます

Nuxt が呼び出す `fetch` に加え、`this.$fetch()` を使うことでコンポーネント内から手動（例として非同期データの再読み込み）で `fetch` を呼び出すことができます。

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

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

Info

`fetch` フック内では `this.$nuxt.context` を使うことで、Nuxt `context` にアクセスできます。

クエリ文字列の変化のリスニング

デフォルトでは、クエリ文字列の変化で `fetch` フックは呼び出されません。クエリ文字列の変化を監視するには、ウォッチャに `$route.query` を追加して `$fetch` を呼び出します：

```
export default {
  watch: {
    '$route.query': '$fetch'
  },
  async fetch() {
    // クエリ文字列の変化時にも呼び出される
  }
}
```

キャッシング

`<nuxt/>` や `<nuxt-child/>` コンポーネントで `keep-alive` ディレクティブを使うと、すでに訪れたページの `fetch` 呼び出しを保存することができます：

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

また、`<nuxt>` コンポーネントへ `keep-alive-props` プロパティを渡すことで、`<keep-alive>` に渡す `props` を指定することもできます。

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

ページコンポーネントを 10 ページ分だけメモリに保存します。

エラーハンドリング

Warning

データをフェッチングするときエラーが発生した場合は、通常 Nuxt エラーページはロードされません。そして、`fetch()` 内で Nuxt `redirect` または `error` メソッドを使うべきではありません。代わりに、`$fetchState.error` を使ったコンポーネント内でエラー処理する必要があります。

データのフェッチングでエラーが発生した場合、`$fetchState.error` でチェックし、エラーメッセージを表示します。

```
<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>
```

activated フックを使う

Nuxt は最後に `fetch` を呼び出した時刻 (SSR も含む) を `this.$fetchState.timestamp` に直接格納します。このプロパティを `activated` フックと組み合わせることで、`fetch` に 30 秒のキャッシュを追加することができます：

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

<script>
  export default {
    data() {
      return {
        posts: []
      }
    },
    activated() {
      // 最後の fetch から30秒以上経っていれば、fetch を呼び出します
      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>
```

最後の `fetch` の呼び出しが 30 秒以内であれば、同じページへの遷移で `fetch` は呼ばれません。

Async Data

Warning

`asyncData` は `pages` でのみ使用可能で、このフック内では `this` にアクセスすることはできません。

`asyncData` はユニバーサルなデータ取得のためのもう 1 つのフックです。非同期な状態を保存するために、コンポーネントのインスタンスにプロパティをセットする（または Vuex アクションをディスパッチする）必要がある `fetch` とは異なり、`asyncData` は単にその返却された値をコンポーネントのデータにマージします。以下は、[@nuxt/http](#) ライブラリを使った例です：

```
<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>
```

`fetch` と異なり、`asyncData` フックから返却される promise は ルートの遷移の間に解決されます。つまり、"loading placeholder" はクライアントサイドの遷移で表示されないということです（ただし読み込み中の状態をユーザーに示すために ローディングバー を使うことができます。Nuxt は代わりに `asyncData` フックの終了を待ってから、次のページへ移動したり エラーページ を表示したりします）。

このフックはページレベルのコンポーネントのためだけに使うことができます。`fetch` と異なり、`asyncData` はコンポーネントインスタンス (`this`) にアクセスすることはできません。そのかわりに、`context` を引数として受け取ります。`asyncData` をデータの取得のために使うことができ、Nuxt は返却されたオブジェクトとコンポーネントのデータとの浅いマージ (shallow merge) を自動的に行います。

今後追加される例では、API からのデータの取得におすすめの `@nuxt/http` を使用します。

コンポーネントの非同期データ？

コンポーネントには `asyncData` メソッドがないため、コンポーネント内でサーバーから非同期データを直接取得することはできません。この制限を回避するには、3 つの基本的なオプションがあります：

1. Nuxt 2.12 以降のバージョンで有効になった 新しい `fetch` フックを使う
2. `mounted` フックで API を呼び出し、ロード時にデータプロパティを設定します。欠点：サーバーサイドレンダリングでは機能しません。
3. ページコンポーネントの `asyncData` メソッドで API を呼び出し、データをプロパティとしてサブコンポーネントに渡します。サーバーのレンダリングは正常に機能します。欠点：ページの `asyncData` は他のコンポーネントのデータを読み込むため読みにくい可能性があります。

クエリ文字列の変化のリスニング

デフォルトでは、クエリ文字列の変化で `asyncData` メソッドは呼び出されません。ページネーションコンポーネントを作成するなどにこの挙動を変えたい場合は、監視するパラメータをページコンポーネントの `watchQuery` プロパティに設定することができます。

Next

`watchQuery` プロパティについてより学び、利用可能な `context` 内のキーのリストを見てみましょう。

メタタグと SEO

Nuxt はアプリケーションにメタデータを追加するために 3 つの方法を用意しています：

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

- nuxt.config.js を使用してグローバルに設定する
- head をオブジェクトとして使用してローカルに設定する
- data と computed プロパティにアクセスするために、head を関数として使用してローカルに設定する

グローバルな設定

Nuxt では、nuxt.config.js 内の head プロパティを利用してすることで、アプリケーションにおけるデフォルトの `<meta>` タグをすべて定義することができます。SEO のためのデフォルトの title と description タグの追加や、ビューポートのセット、ファビコンの追加が簡単にできます。

```
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

この設定により、すべてのページで共通のタイトルと説明が設定されます。

ローカルな設定

各ページのスクリプトタグ内で `head` プロパティを設定することで、ページ別のタイトルとメタデータを追加することもできます。

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

```

        }
    }
</script>

```

Info

`head` をオブジェクトとして使用して、ホームページのみに title と description を設定する例。

```

<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

`head` を関数として使用して、ホームページのみに title と description を設定する例。関数を使用することで `data` と `computed` プロパティにアクセスすることができます。

Nuxt はアプリケーションの `document head` とメタ要素を更新するために、[vue-meta](#) を利用しています。

Warning

子コンポーネントを使用したときにメタタグの重複を避けるために、`hid` キーを利用してメタデータに一意な識別子を与えてください。こうすることで `vue-meta` は、デフォルトのタグを上書きすべきということを知ることができます。

Next

`head` の利用可能なプロパティについての詳細は、[vue-meta のドキュメント](#) を参照してください。

外部リソース

スクリプトやフォントのような外部のリソースを含めるには、`nuxt.config.js` にグローバルに追加する、もしくは `head` オブジェクトまたは関数の中にローカルに追加する必要があります。

Info

各リソースにオプションの `body: true` を渡すことで、`</body>` の終了タグの前にリソースを含めることもできます。

グローバルな設定

```
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'
      }
    ]
  }
}
```

ローカルな設定

```
<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>
```

設定

Nuxt ではデフォルトの設定でほとんどのユースケースをカバーしています。`nuxt.config.js` を使ってこの設定を上書きすることができます。

css プロパティ

グローバルに（すべてのページで）適用したい CSS ファイル/モジュール/ライブラリを設定できます。

Warning

`sass` を利用したい場合は `node-sass` と `sass-loader` パッケージをインストールしてください。

`nuxt.config.js` で CSS リソースを追加するには：

```
export default {
  css: [
    // Node.js モジュールを直接ロードする（ここでは Sass ファイル）
    'bulma',
    // プロジェクト内の CSS ファイル
    '~/assets/css/main.css',
    // プロジェクト内の SCSS ファイル
    '~/assets/css/main.scss'
  ]
}
```

Warning

Nuxt は拡張子から自動的にファイルタイプを推測して Webpack の適切なプリプロセッサローダを使用します。それらを使用する場合は各自で必要なローダをインストールしてください。

スタイルの拡張

`nuxt` の設定ファイルで、`css` プロパティの配列に記述する CSS/SCSS/Postcss/Less/Stylus/ などのファイルの拡張子は省略することができます。

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

Warning

`main.scss` と `main.css` のような同じ名前の 2 つのファイルが存在し、`css: ['~/assets/css/main']` のように `css` 配列中で拡張子を明示しなかった場合、`styleExtensions` の順番に応じて 1 つのファイルだけが読み込まれます。このケースの場合、`css` がデフォルトの `styleExtension` の配列の中で最初に登場するため、`css` ファイルだけが読み込まれ `scss` ファイルは無視されます。

デフォルトの順番： `['css', 'pcss', 'postcss', 'styl', 'stylus', 'scss', 'sass', 'less']`

プリプロセッサ

Vue Loaderのおかげで、`lang` 属性を使うだけで `<template>` や `<style>` などのためのさまざまなプリプロセッサを使うことができます。

`Pug` と `Sass` を使った `pages/index.vue` の例：

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

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

これらのプリプロセッサを使うために Webpack のローダーをインストールする必要があります：

```
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
```

外部のリソース

グローバルの設定

ヘッドオブジェクトまたは関数に外部リソースを含めることができます。head API ドキュメントで説明したように、次の例ではオブジェクトや関数として `head` の使い方を示します。もし計算されたプロパティやデータなど、Vue コンポーネントの値を使いたい場合は、`head()` 関数を使って最終的な `head` オブジェクトを返すことができます。オプションの `body: true` を各リソースに渡して、閉じる `</body>` タグの前にリソースを含めることもできます。

`nuxt.config.js` (ここでは `head` オブジェクト) にリソースを含めます：

```
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'
      }
    ]
  }
}
```

ローカルの設定

pages/ ディレクトリ（ここでは head 関数）内の .vue ファイルにリソースを含めます：

```
<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>
```

PostCSS プラグイン

もし存在する場合、プロジェクトディレクトリにある postcss.config.js の名前を変更するか削除します。そして nuxt.config.js ファイルに以下を追加します：

```
export default {
  build: {
    postcss: {
      // プラグイン名をキーとして、引数を値として追加します
      // npm または yarn を使って依存関係としてこれらを事前にインストールします
      plugins: {
        // false を値として渡しプラグインを無効にします
        'postcss-url': false,
        'postcss-nested': {},
        'postcss-responsive-type': {},
        'postcss-hexrgba': {}
      },
      preset: {
        // false を値として渡しプラグインを無効にします
        autoprefixer: {
          grid: true
        }
      }
    }
  }
}
```

```

    }
}
}
```

JSX

Nuxt は babel のデフォルトの設定用の公式の [@vue/babel-preset-app](#) をベースにした [@nuxt/babel-preset-app](#) を使用しています。そのため、コンポーネントに JSX を使うことができます。

コンポーネントの `render` メソッド内で JSX が使えます：

```

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

`createElement` を `h` にエイリアスすることは、Vue のエコシステムで見られる共通の慣例です。しかしその慣例は JSX では任意です。なぜなら ES2015 の構文で宣言された（関数またはアロー関数ではない）JSX で書かれた任意のメソッドやゲッターには `const h = this.$createElement` が自動的に注入されるためです。よって `(h)` パラメータは削除することができます。

JSX の使い方をより深く理解するには Vue.js ドキュメントの [JSX セクション](#)を参照してください。

ファイルの無視

.nuxtignore

`.nuxtignore` を使用することで、ビルド時にプロジェクトルート (`rootDir`) にある `layout`、`page`、`store` そして `middleware` のファイルを Nuxt に無視させることができます。`.nuxtignore` ファイルは `.gitignore` や `.eslintignore` と同じ仕様に従います。各行はどのファイルを無視するかを示す glob パターンです。

```

# foo.vue レイアウトを無視する
layouts/foo.vue

# ファイル名が -ignore.vue で終わるレイアウトファイルを無視する
layouts/*-ignore.vue

# bar.vue ページを無視する
pages/bar.vue

# ignore フォルダにあるページを無視する
pages/ignore/*.vue

# bar.js ストアを無視する
store/baz.js
```

```
# _.test._ にマッチするストアファイルを無視する
store/ignore/_.test._

# foo フォルダにある foo/bar.js 以外のミドルウェアファイルと無視する
middleware/foo/*.js !middleware/foo/bar.js
```

ignorePrefix プロパティ

`pages/`、`layout/`、`middleware/` や `store/` ディレクトリに含まれる、ファイル名が `ignorePrefix` プロパティで指定された接頭辞から始まっているファイルはビルド時に無視されます。

デフォルトでは `-` から始まる `store/-foo.js` や `pages/-bar.vue` のようなファイルはすべて無視されます。これにより、ルートやストア等に変換されることなく、呼び出し元と同じ場所にテスト、ユーティリティ、コンポーネントなどのファイルを置くことができます。

ignore プロパティ

`ignorePrefix` よりカスタマイズしやすいです: `ignore` プロパティに指定した `glob` パターンと一致するすべてのファイルがビルド時に無視されます。

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

ignoreOptions

`nuxtignore` は内部で `node-ignore` を使用しています。`ignoreOptions` は `node-ignore` の `options` として設定されます。

```
export default {
  ignoreOptions: {
    ignorecase: false
  }
}
```

webpack config を拡張する

`nuxt.config.js` 内の `extend` オプションを通して nuxt の webpack 設定を拡張できます。`build` プロパティの `extend` オプションは 2 つの引数を受け取る関数です。第一引数は nuxt の webpack 設定からエクスポートされた webpack `config` オブジェクトです。第二引数はこれらの Boolean 型のプロパティを持つ `context` オブジェクトです: `{ 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'
      })
      // `isDev` が true の場合、webpack を開発モードに設定します。
      if (isDev) {
        config.devtool = 'source-map'
      }
    }
  }
}
```

```
    config.mode = 'development'  
  }  
}  
}
```

`extend` メソッドは 2 回呼び出されます - 1 回目はクライアントのバンドルため、もう 1 回はサーバーのバンドルのためです。

チャンク設定をカスタマイズする

デフォルトのオブジェクトを書き換えずに最適化設定を微調整することができます。

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

webpack 設定の検査

複雑なプロジェクトやデバッグの場合、最終的な webpack 構成がどのようになるかを確認すると便利な場合があります。幸い、プロジェクトから `nuxt webpack` コマンドを実行して設定を出力できます。詳細については、GitHub のプルリクエスト [#7029](#) を確認してください。

webpack プラグインの追加

`nuxt.config.js` ファイルの `build` オプションを使うと、`webpack.config.js` ファイルと同じ方法で `webpack plugins` を渡すことができます。

この例では JavaScript モジュール (*lodash* と *jQuery*) を `import` や `require` するかわりに自動的にロードする webpack ビルトインの `ProvidePlugin` を追加します。

```
import webpack from 'webpack'

export default {
  build: {
    plugins: [
      new webpack.ProvidePlugin({
        // グローバルモジュール
        $: 'jquery',
        _: 'lodash'
      })
    ]
  }
}
```

注意: Vue ベースのアプリケーションでは jQuery は必要ないかもしれません。

Nuxt を使うと、プラグインの実行コンテキストを制御することもできます。もしプラグインが `build.extend` 内の `client` または `server` ビルド（または `dev` ビルドと `prod` ビルドを区別する）で実行されることを意図している場合、webpack プラグインを手動で渡すこともできます。

Webpack を拡張してオーディオファイルをロードする

オーディオファイルは `file-loader` で処理する必要があります。このローダーはすでにデフォルトの Webpack 設定に含まれていますが、オーディオファイルを処理するように設定されていません。`nuxt.config.js` でデフォルトの設定を拡張する必要があります：

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

これでオーディオファイルを `<audio :src="require('@/assets/water.mp3')" controls></audio>` のようにインポートできます。

`<audio src="@/assets/water.mp3" controls></audio>` だけ書きたい場合、`src` 属性でこれらを参照する際に `vue-loader` に自動的にオーディオファイルを要求するように支持する必要があります：

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

ホストとポート番号を編集する

デフォルトでは、Nuxt の開発サーバーのホストは `localhost`（ホストマシン内からのみアクセス可能）です。アプリケーションを他のデバイスで確認するにはホストを修正する必要があります。`nuxt.config.js` ファイルでホストを修正することができます。

ホストの `'0.0.0.0'` は、ホストマシンの外部からの接続（例えば LAN）でもアクセスが可能なホストアドレスを解決するように Nuxt に指示します。ホストに `'0'`（0 ではありません）や `'0.0.0.0'` という文字列が指定された場合、一口カル IP アドレスが Nuxt に割り振られます。

```
export default {
  server: {
    host: '0' // デフォルト: localhost
  }
}
```

ポート番号をデフォルトの 3000 から変更することもできます。

```
export default {
  server: {
    port: 8000 // デフォルト: 3000
  }
}
```

Info

ポート番号に `'0'`（0 ではなく falsy）という文字列の値が指定された場合、ランダムなポートが Nuxt アプリケーションに割り振られます。

`nuxt.config.js` ファイル内で変更することはできますが、サイトをホスティングする際に問題が発生する場合があるのでおすすめしません。`dev` コマンドで直接ホストとポートを指定することをおすすめします。

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

または `package.json` 内でスクリプトを作成します。

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

非同期な設定

通常の `export default {}` を使用した設定をするのが望ましいですが、config オブジェクトを返却する非同期な関数をエクスポートすることで非同期な設定をすることができます。

```
import axios from 'axios'

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

```
//... 残りの設定
}
}
```

Warning

`nuxt.config.js` 内で `axios-module` を使うことはできません。`axios` をインポートして再度設定する必要があります。

さらなる設定

Next

`nuxt.config.js` には、カスタマイズや設定オプションがたくさんあります！ [build の設定](#) すべてのキーを確認しましょう。

[Go to TOC](#)

ローディング

Nuxt は、遷移中に表示する独自のローディングプログレスバー・コンポーネントを提供します。これをカスタマイズしたり、無効にしたり、独自のコンポーネントを作成したりすることができます。

プログレスバーをカスタマイズする

他のプロパティの中でも、プログレスバーの色、サイズ、期間、方向はアプリケーションのニーズに合わせてカスタマイズできます。`nuxt.config.js` の `loading` プロパティに関連するプロパティを更新することでカスタマイズができます。

例えば、高さ 5px の青いプログレスバーを設定するには、`nuxt.config.js` を次のように更新します：

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

プログレスバーをカスタマイズするために使えるプロパティ一覧。

| キー | 型 | デフォルト | 説明 | | | | color | String | 'black' | プログレスバーの CSS カラー | | | failedColor | String | 'red' | ルートをレンダリング中にエラーが発生した場合のプログレスバーの CSS カラー。(例えば `data` または `fetch` がエラーを返したとき) | | | height | String | '2px' | プログレスバーの高さ。(プログレスバーの `style` プロパティで使われます) | | | throttle | Number | 200 | プログレスバーを表示するまでに待つ時間(ミリ秒単位)。プログレスバーの点滅を防ぐことに役立ちます | | | duration | Number | 5000 | プログレスバーを表示する時間の最大値(ミリ秒単位)。Nuxt は各ルートが 5 秒以内にレンダリングされると想定しています | | | continuous | Boolean | false | ローディングが `duration` で指定した時間より長くかかる場合にプログレスバーのアニメーションを継続します | | | css | Boolean | true | デフォルトのプログレスバーのスタイルを削除(そして独自に追加)する場合には `false` を設定します | | | rtl | Boolean | false | プログレスバーの向きを右から左にします | |

プログレスバーを無効にする

ルートから別のルートへ遷移する間にプログレスバーを表示したくないときは `nuxt.config.js` ファイルに `loading: false` を追加します：

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

デフォルトのローディングプログレスバーを特定のページで無効にすることも可能です。

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

<script>
  export default {
```

```
    loading: false
}
</script>
```

ローディングバーをプログラムで起動する

ローディングバーは `this.$nuxt.$loading.start()` を呼び出してローディングバーを開始させたり、`this.$nuxt.$loading.finish()` を呼び出して停止させたりなど、コンポーネント内でプログラムで起動することもできます。

ページコンポーネントのマウントプロセス中に、`$loading` プロパティにすぐアクセスできない場合があります。これを回避するために、`mounted` メソッドでローダーを起動したい場合は、以下のように `this.$nextTick` 内で `$loading` メソッドの呼び出しを行うようにしてください。

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

プログレスバーの内部

残念ながら、ローディングコンポーネントは新しいページを読み込むのにかかる時間を事前に知ることはできません。したがって、ローディング時間の 100% ちょうどにプログレスバーをアニメーションさせることはできません。

Nuxt のローディングコンポーネントは `duration` を設定することでこれを部分的に解決します。これはローディングプロセスにかかる時間の推定値として設定すべきです。カスタムローディングコンポーネントを使わない限り、プログレスバーは（実際の進行にかかわらず）`duration` の時間内で常に 0% から 100% まで移動します。ローディングが `duration` で設定された時間より長くかかる場合は、プログレスバーはローディングが完了するまで 100% の位置にとどまります。

`continuous` を `true` に設定することでこのデフォルトの挙動を変更することができます。プログレスバーが 100% に達した後に再び `duration` で設定した時間で 0% に向かって縮小し始めます。0% に達してもまだローディングが完了していない場合は、再び 0% から 100% に向かって伸び始めます。ローディングが完了するまでこれを繰り返します。

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

`continuous` を設定したプログレスバーの例：



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

独自のローディングコンポーネントを使う

Nuxt がデフォルトのプログレスバーコンポーネントの代わりに呼び出す、独自のコンポーネントを作成することができます。そのためには `loading` オプション内に独自コンポーネントへのパスを指定する必要があります。こうすることで Nuxt は独自コンポーネントを直接呼び出します。

独自コンポーネントはこれらのメソッドを備えている必要があります：

| メソッド | 必須か否か | 説明 | | | start() | 必須 | ルートが変更されたときに呼び出されます。このときコンポーネントが表示されます | | finish() | 必須 | ルートがロード（そしてデータが取得）されたときに呼び出されます。このときコンポーネントの表示が終了します | | fail(error) | 任意 | ルートがロードできなかったときに呼び出されます（例えばデータの取得に失敗したときなどです） | | increase(num) | 任意 | ルートのコンポーネントがロードされている間に呼び出されます。num は 100 未満の整数です |

`components>LoadingBar.vue` に独自のカスタムコンポーネントを作成することができます：

```
<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>
```

それから `nuxt.config.js` を編集して、独自のコンポーネントを使うことを Nuxt に伝えます：

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

loading indicator プロパティ

SPA モードで Nuxt を実行すると、初回のページ読み込み時にはサーバーからのコンテンツがありません。そのためページが読み込まれている間に、空白のページの代わりとして独自の色や背景を加えたりインジケータを変更したりできるスピナーを提供しています。

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

ビルトインインジケータ

これらのインジケータは、素晴らしい [SpinKit](#) プロジェクトからインポートされています。Spinkit のデモページでスピナーを試してみることができます。これらのスピナーを使うために必要なのは、`name` プロパティに使いたいスピナーの名前を指定することだけです。これらは使うことができるビルトインのインジケータのリストです。

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

ビルトインインジケータは、`color` および `background` オプションをサポートしています。

カスタムインジケータ

もし独自の特別なインジケータが必要な場合は、String 値もしくは Name キーにインジケータのソースコードを HTML テンプレートへのパスとして設定することができます！その際、すべてのオプションもテンプレートへと渡されます。

ベースが必要な場合は、Nuxt のビルトイン[ソースコード](#)を使うこともできます。

[Go to TOC](#)

Nuxt で提供されるコンポーネント

Nuxt はいくつかの重要なコンポーネントをいっしょに提供しており、アプリケーションを構築する際に役立ちます。これらのコンポーネントはグローバルで利用可能になっており、それらを使うにあたってインポートする必要ないことを意味します。

以降では、組み込まれているコンポーネントそれぞれについて説明します。

Nuxt コンポーネント

<Nuxt> コンポーネントはページコンポーネントで表示するために使います。基本的にこのコンポーネントは、表示されているページに応じて、ページコンポーネントの中にあるものに置き換えられます。それゆえ、<Nuxt> コンポーネントをレイアウトに追加することは重要です。

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

Warning

<Nuxt> コンポーネントはレイアウト 内部でのみ使うことができます。

<Nuxt> コンポーネントは nuxt-child-key のプロパティを取得できます。このプロパティは <RouterView> に渡されるため、トランジションは動的なページ内で正しく動作するようになります。

<RouterView> の内部の key プロパティで処理されるためには 2 つの方法があります。

1. <Nuxt> コンポーネントで nuxtChildKey プロパティを使う

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

2. ページコンポーネントで key オプションを string または function として追加する

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

NuxtChild コンポーネント

このコンポーネントは、ネストされたルートで子コンポーネントを表示するために使われます。

例：

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

このファイルツリーは以下のルートが生成されます：

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

`child.vue` コンポーネントを表示するために、`pages/parent.vue` 内部で `<NuxtChild>` コンポーネントを挿入する必要があります：

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

keep-alive

`<Nuxt>` コンポーネント、そして `<NuxtChild>` コンポーネント両方は、`keep-alive` そして `keep-alive-props` を受け付けます。

Info

`keep-alive` そして `keep-alive-props` について詳細は、[Vue.js のドキュメント](#) を参照してください。

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

<!-- 以下のようなものに変換されます -->
<div>
```

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

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

<!-- 以下のようなものに変換されます -->
<div>
  <KeepAlive :exclude="['modal']">
    <RouterView />
  </KeepAlive>
</div>
```

`<NuxtChild>` コンポーネントはまた通常の Vue コンポーネントのようなプロパティも受け付けます。

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

例を確認するには、[nested-routes の例](#)を参照してください。

:code-sandbox{src="csb_link_nuxt"}

NuxtLink コンポーネント

アプリケーションのページ間を遷移するためには、`<NuxtLink>` コンポーネントを使用する必要があります。このコンポーネントは、Nuxt に含まれているため、他のコンポーネントといっしょにインポートする必要はありません。HTML の`<a>` タグに似ていますが、`href="/about"` を使う代わりに、`to="/about"` を使うことができます。以前に`vue-router`を使用したことがある場合は、`<RouterLink>` の代わりに`<NuxtLink>` を使用することができます。

`pages` フォルダの`index.vue` ページにリンクするシンプルな例です：

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

`<NuxtLink>` コンポーネントは全ての内部リンクに対して使う必要があります。つまり、サイト内のページへ全てのリンクに対して`<NuxtLink>` を使用する必要があります。`<a>` タグは全ての外部リンクに対して使用する必要があります。つまり、他のウェブサイトにリンクがある場合は、それらに対して`<a>` タグを使用する必要があります。

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

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

Info

`<RouterLink>` についてもっと知りたい場合は、[Vue Router のドキュメント](#)を読んで参考にしてください。

Info

`<NuxtLink>` にはスマートプリフェッ칭もサポートしています。

prefetchLinks

Nuxt は自動的にスマートプリフェッ칭をサポートしています。つまり、ビューポートまたはスクロール時にリンクが表示されたことを検知し、そしてユーザーがリンクをクリックしたときにすぐに使えるよう、ページにおいて JavaScript をプリフェッチします。Nuxt はブラウザがビジー状態でないときにのみリソースをロードし、接続がオフラインの場合や 2g の接続しかできない場合はプリフェッチをスキップします。

特定のページに対するプリフェッチングの無効

しかし、一部のリンクでプリフェッチを無効にしたい場合があります。ページに多くの JavaScript が存在する、またはプリフェッチされるページが多数ある場合、ロードする必要のあるサードパーティスクリプトが多数ある場合などです。特定のリンクのプリフェッチを無効にするには、`no-prefetch` プロパティを使用できます。Nuxt v2.10.0 以降では、`prefetch` プロパティを `false` に設定して使用することもできます。

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

グローバルなプリフェッチングの無効

全てのリンクでプリフェッチングを無効にするために、`prefetchLinks` に `false` を設定します：

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

Nuxt v2.10.0 からは、`prefetchLinks` に `false` を設定していても、特定のリンクをプリフェッチしたい場合は、`prefetch` プロパティを使用できます：

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

linkActiveClass

`linkActiveClass` はアクティブなリンクにおいて `vue-router` の `class` と同じように動作します。どのリンクがアクティブなのかを表示したい場合は、`nuxt-link-active` という `class` の CSS を作成するとよいでしょう。

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

この CSS は遷移コンポーネントや、特定のページやレイアウト、または main.css ファイルに追加することができます。

必要であれば、class 名を別ものに設定することもできます。これを行うには、`nuxt.config.js` ファイルの `router` プロパティで `linkActiveClass` を変更することができます。

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

Info

このオプションは `vue-router` の `linkActiveClass` に直接設定されます。詳細については[Vue Router のドキュメント](#)を参照してください。

linkExactActiveClass

`linkExactActiveClass` は、`vue-router` の class と同じように、完全にアクティブなリンクに対して動作します。どのリンクが完全に一致してアクティブになっているかを表示したい場合は、`nuxt-link-exact-active` という class の CSS を作成するだけです。

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

Info

この CSS は遷移コンポーネントや、特定のページやレイアウト、または main.css ファイルに追加することができます。

必要であれば、class 名を別ものに設定することもできます。これを行うには、`nuxt.config.js` ファイルの `router` プロパティで `linkExactActiveClass` を変更することができます。

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

Info

このオプションは `vue-router` の `linkExactActiveClass` に直接設定されます。詳細については[Vue Router のドキュメント](#)を参照してください。

linkPrefetchedClass

`linkPrefetchedClass` を使うと、プリフェッチされたすべてのリンクにスタイルを追加することができます。これは、デフォルトの動作を変更した後、どのリンクがプリフェッチされているかをテストするのに最適です。`linkPrefetchedClass` は、デフォルトでは無効になっています。これを有効にするには、`nuxt-config.js` ファイルの `router` プロパティに追加する必要があります。

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

そして、そのクラスに対してスタイルを追加することができます

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

Info

この例では、`nuxt-link-prefetched` class を使いますが、好きな名前を付けることができます。

:code-sandbox{src="csb_link_nuxt_link"}

client-only コンポーネント

このコンポーネントは、コンポーネントを意図的にクライアントサイドでのみレンダリングするために使用します。クライアント側でのみコンポーネントをインポートするには、クライアントサイド専用のプラグインにコンポーネントを登録します。

```
<template>
  <div>
    <sidebar />
    <client-only placeholder="Loading...">
      <!-- このコンポーネントはクライアント側のみレンダリングされます -->
      <comments />
    </client-only>
  </div>
</template>
```

クライアント側で `<client-only />` がマウントされるまで、スロットをプレースホルダーとして使用します。

```
<template>
  <div>
    <sidebar />
    <client-only>
      <!-- このコンポーネントはクライアント側のみレンダリングされます -->
      <comments />
      <!-- ローディングインジケータはサーバー側でレンダリングされます -->
      <template #placeholder>
        <comments-placeholder />
      </template>
    </client-only>
  </div>
</template>
```

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

Info

ときどき、サーバーでレンダリングされたページでは、`<client-only>` 内の `$refs` が `$nextTick` を使用しても準備できなことがあります。その場合は `$nextTick` を何度も呼び出す必要があります：

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

alert Nuxt < v2.9.0 なバージョンを使っている場合は、`<client-only>` の代わりに `<no-ssr>` を使います。

自動探索の有効

v2.13 から、Nuxt はテンプレートで使用されているコンポーネントを自動的にインポートすることができます。この機能を有効にするには、設定ファイルで `components: true` を設定してください：

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

Info

詳細は[how to configure component auto-discovery](#)を確認してください。

コンポーネントを使う

一度 `components` ディレクトリにコンポーネントを作成すると、インポートしなくてもアプリ全体で利用できるようになります。

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

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

Info

[ライブデモ](#) または[ビデオ](#)の例を確認してください。

コンポーネントの名前

次のようなネストされたディレクトリにコンポーネントがある場合：

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

コンポーネント名はそれ自信のパスディレクトリとファイル名をベースにします。したがって、以下のコンポーネントは：

```
<BaseFooButton />
```

alert わかりやすくするために、コンポーネントのファイル名はその名前と一致させることをお勧めします。(つまり、上記の例では、`Button.vue` を `BaseFooButton.vue` にリネームすることができます。)

コンポーネント名に含まれないカスタムディレクトリ構造を使用したい場合は、これらのディレクトリを明示的に指定することができます：

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

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

そして、テンプレートでは、`BaseFooButton` の代わりに `FooButton` を使用することができます。

```
<FooButton />
```

Info

コンポーネントやディレクトリの名前は、[Vue.js のスタイルガイド](#)を検討してください。

動的インポート

コンポーネントを動的にインポート(コンポーネントの遅延ロードとして知られている)するためには、コンポーネント名に `Lazy` という接頭辞を追加するだけです。

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

これは、コンポーネントが常に必要なわけではない場合に特に便利です。`Lazy` という接頭辞を使えば、コンポーネントのコードの読み込みを適切なタイミングまで遅らせることができるので、JavaScript のバンドルサイズを最適化するのに役立ちます。

```
<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>
```

チートシート

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

[Go to TOC](#)

Nuxt ビルドディレクトリ

`.nuxt` ディレクトリはいわゆる *build* ディレクトリです。このディレクトリは動的に生成され、デフォルトでは非表示になっています。ディレクトリの中には、`nuxt dev` を使っているときには自動生成されたファイルが、`nuxt build` を使っているときにはビルトの成果物が入っています。これらのファイルを変更することはデバッグに最適ですが、これらは生成されたファイルであり、`dev` や `build` コマンドを再度実行すると、ここに保存されていたものはすべて再生成されることを覚えておいてください。

Warning

`.nuxt` ディレクトリはバージョン管理システムにコミットされるべきではありません。`nuxt dev` または `nuxt build` を実行したときに自動的に生成されるので、`.gitignore` で無視してください。

buildDir プロパティ

デフォルトでは `.nuxt` のディレクトリ名がドットで始まるため、多くのツールが隠しディレクトリと見なします。これを防ぐには `buildDir` オプションを使用します。名前を変更した場合は、新しい名前を `.gitignore` ファイルに追加することを忘れないようにしてください。

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

.nuxt フォルダの中身 :

- `router.js` ファイルは、ページフォルダ内に `.vue` ファイルを配置したときに、Nuxt が生成してくれるルーターファイルです。このファイルは、`vue-router` 用に生成されたルートを調べたり、特定のルートの名前を調べたりしたいときのデバッグに使うことができます。
- `router.scrollBehavior.js` は、ルーターの `ScrollBehavior` です。 `components` フォルダには、`NuxtChild` や `NuxtLink` などの Nuxt コンポーネントが入っています。また、アプリケーションのビルト中に表示されるページである `nuxt-build-indicator` や、ページの読み込みを待っているときに表示されるロードコンポーネントである `nuxt-loading` も含まれています。Nuxt のデフォルトのエラーページを含む `nuxt-error` ページもここにあります。
- `mixins` フォルダには、Nuxt の `$fetch` メソッドに必要なファイルが含まれています。
- `views` フォルダには、アプリのテンプレートとサーバーエラーページが含まれます。
- `app.js` は、メインのアプリケーションファイルです。
- `client.js` ファイルは、クライアントサイドで発生するすべてのことに必要なクライアントファイルです。
- `empty` ファイルは何も処理しないエイリアスのために意図的に空のままになっています。
- `index.js` ファイルは、アプリケーションを起動します。
- `loading.html` は、ページの読み込み時に使用するファイルです。
- `middleware` ファイルは、ミドルウェアが保存されている場所です。
- `server.js` ファイルは、サーバー上で実行されるすべてのコードです。
- ユーティリティには、Nuxt が動作するために必要なユーティリティが含まれています。

デプロイする

.nuxt フォルダは SSR アプリケーションのデプロイに必要なファイルの一部です。ただし、静的 Nuxt アプリのデプロイには dist フォルダを使用するため必要ありません。

[Go to TOC](#)

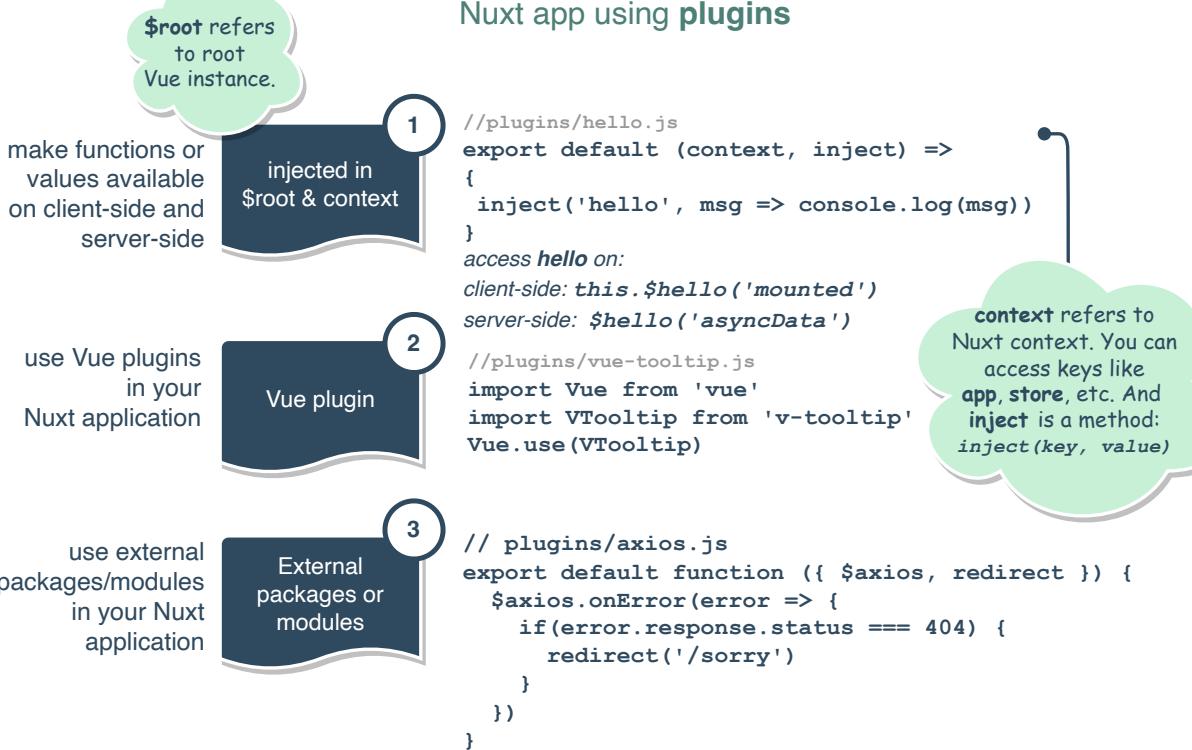
プラグインディレクトリ

`plugins` ディレクトリにはルート `Vue.js` アプリケーションがインスタンス化する前に実行する Javascript プラグインが含まれています。

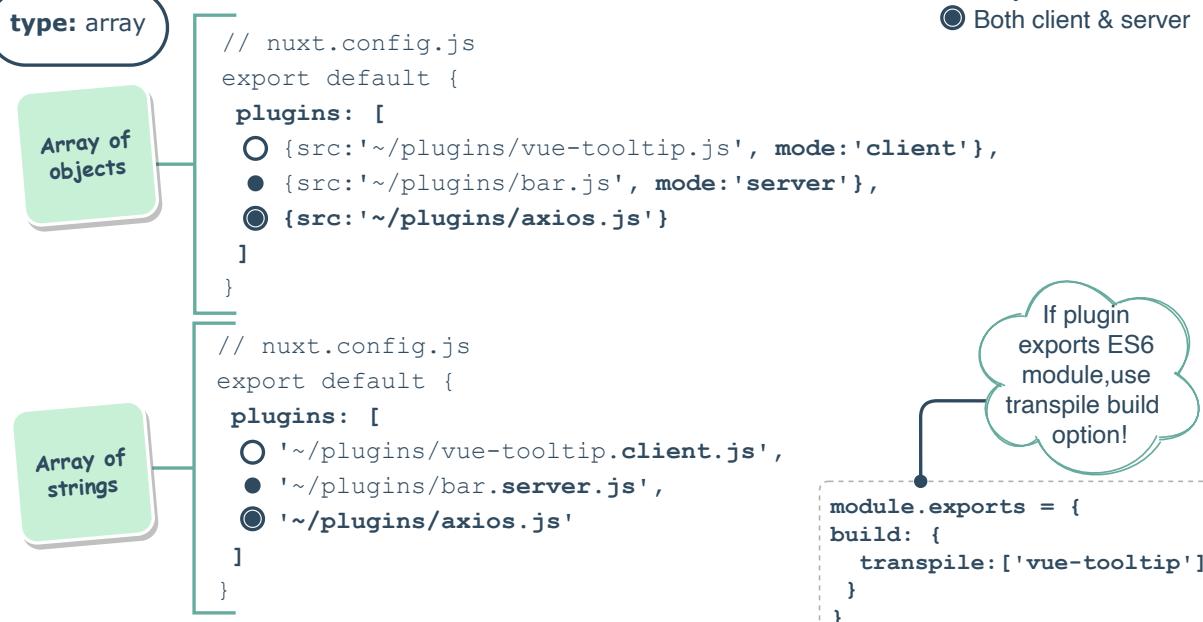
Nuxt Plugins

Nuxt v2.14.1
August 2020

3 ways to extend functionalities of Nuxt app using plugins



Include Nuxt plugin using **plugins** property



これは Vue プラグインを追加し、関数や定数を導入する場所です。 `Vue.use()` を使用する必要がある時は都度 `plugins/` ファイルを作成し、`nuxt.config.js` の `plugins` にそのパスを追加する必要があります。

外部パッケージ

サーバーとクライアント両方に HTTP リクエストを行うためには、アプリケーションで外部パッケージ/モジュール（著名な例は `axios` です）を使用することをお勧めします。

はじめに、npm か Yarn を介してインストールします。

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

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

例えば、アプリケーション全体の API 呼び出しから発生する可能性のあるエラーに対応するよう、`axios` インターセプターを設定できます。この例では、API から 500 ステータスエラーが発生した時、`sorry` というカスタムエラーページにユーザをリダイレクトします。

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

最後になりますが、モジュールと新しく作成したプラグインをプロジェクト構成に追加します。

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

その後ページコンポーネントで直接使用できます：

```
<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>
```

モジュールのインストールなしで `axios` を使うための他の方法は、`<script>` タグで直接 `axios` をインポートすることができます。

```
<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

Cannot use import statement outside a module (モジュールの外部でインポートステートメントを使用できない) というエラーが発生した場合、プラグインを使用できるようにするために、webpack ローダーの `nuxt.config.js` の `build > transpile` オプションにパッケージを追加する必要があります。

```
build: {
  // ここでwebpack設定を拡張できます
  transpile: ['npm-package-name'],
},
```

Vue プラグイン

`v-tooltip` などの Vue プラグインを使用してアプリケーションにツールチップを表示する場合、アプリを起動する前にプラグインを設定する必要があります。

はじめにインストールをする必要があります

```
yarn add v-tooltip
```

```
npm install v-tooltip
```

次にファイル `plugins/vue-tooltip.js` を作成します

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

Vue.use(VTooltip)
```

plugins プロパティ

次に `nuxt.config.js` の `plugins` キー内にファイルパスを追加します。`plugins` プロパティを使用すると、簡単に Vue.js プラグインをメインアプリケーションに追加できます。`plugins` プロパティで定義されているすべてのパスは、メインアプリケーションを初期化する前にインポートされます。

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

ES6 プラグイン

プラグインが `node_modules` にあり、ES6 モジュールをエクスポートする場合、`transpile` ビルドオプションに追加する必要がある場合があります：

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

その他のビルドオプションについては、[configuration build](#) のドキュメントを参照してください。

クライアントまたはサーバーサイドのみ

一部のプラグインは SSR がサポートされていないため、ブラウザのみで機能する場合があります。

従来のプラグインに名前を付ける

プラグインがクライアント、もしくはサーバーサイドのみで実行されると想定される場合、`.client.js` または `.server.js` は、プラグインファイルの拡張子として適用できます。ファイルはそれぞれの（クライアントまたはサーバー）サイドのみに自動的に含まれます。

```
export default {
  plugins: [
    '~/plugins/foo.client.js', // クライアントサイドのみ
    '~/plugins/bar.server.js', // サーバーサイドのみ
    '~/plugins/baz.js' // クライアントサイドとサーバーサイド両方
  ]
}
```

オブジェクト構文

`plugins` の `mode` プロパティ (`'client'` または `'server'`) でオブジェクト構文を使用することもできます。

```
export default {
  plugins: [
    { src: '~/plugins/both-sides.js' },
    { src: '~/plugins/client-only.js', mode: 'client' }, // クライアントサイドのみ
    { src: '~/plugins/server-only.js', mode: 'server' } // サーバーサイドのみ
  ]
}
```

\$root とコンテキストの挿入

ときどき、アプリケーション内で関数や値を利用できるようにしたい場合があります。これらの変数を Vue インスタンス（クライアントサイド）、コンテキスト（サーバーサイド）、さらには Vuex ストアに挿入できます。これらの関数の前に `$` を付けるのが慣例です。

Nuxt はこれを簡単に行うための `inject (key, value)` メソッドを提供します。関数をエクスポートするとき、2 番目のパラメーターとして `Inject` が指定されます。`$` は、キーの先頭に自動的に追加されます。

Info

Vueインスタンスのライフサイクルでは、`beforeCreate` フックと `created` フックのみがクライアント、サーバーサイド両方から呼び出されることを把握しておくことが重要です。他のすべてのフックはクライアントサイドからのみ呼び出されます。

```
export default ({ app }, inject) => {
  // Vue、コンテキスト、そしてストアに $hello(msg) を挿入します。
  inject('hello', msg => console.log(`Hello ${msg}!`))
}
```

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

`$hello` サービスは、ページ、コンポーネント、プラグイン、ストアアクションの `context` と `this` にアクセスできるようになりました。

```
export default {
  mounted() {
    this.$hello('mounted')
    // console.log 'Hello mounted!' が呼び出される
  },
  asyncData({ app, $hello }) {
    $hello('asyncData')
    // Nuxt <= 2.12 を使用する場合は ↗
    app.$hello('asyncData')
  }
}
```

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

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

Warning

`Vue.use()`、`Vue.component()` を使用しないでください、またグローバルに、Nuxt インジェクション専用のこの関数内に `Vue` を接続しないでください。サーバーサイドでメモリリークが発生します。

extendPlugins プロパティ

プラグインを拡張したり、Nuxt によって作成されたプラグインの順序を変更したりすることができます。この関数は `plugin` オブジェクトの配列を受け入れ、プラグインオブジェクトの配列を返す必要があります。

プラグインの順序を変更する例：

```
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
  }
}
```

グローバルミックスイン

グローバルミックスインは Nuxt プラグインで簡単に追加できますが、正しく処理しないとトラブルやメモリリークが発生する可能性があります。アプリケーションにグローバルミックスインを追加するときは、常にフラグを使用して複数回登録しないようにする必要があります：

```
import Vue from "vue"

// 他の mixin と衝突しないように
// フラグの名前は必ずユニークなものにしてください。
if (!Vue.__my_mixin__) {
  Vue.__my_mixin__ = true
  Vue.mixin({ ... }) // ミックスインを設定する
}
```

静的ディレクトリ

`static` ディレクトリはサーバルートに直接マッピングされ、また変更されない可能性があるコンテンツファイルが含まれています。含まれている全てのファイルは Nuxt によって自動で提供され、プロジェクトのルート URL からアクセスできます。

`/static/robots.txt` は、`http://localhost:3000/robots.txt` で利用できます

`/static/favicon.ico` は、`http://localhost:3000/favicon.ico` で利用できます

このオプションは `robots.txt`、`sitemap.xml`、`CNAME` (GitHub Pages のデプロイに重要です) などのファイルに役立ちます。

Warning

このディレクトリは追加の設定をせずに名前を変更することはできません。

静的アセット

`assets` ディレクトリの Webpack assets を使用したくない場合は、画像を `static` ディレクトリに追加できます。

コードではルート（/）を基準にしてこれらのファイルを参照できます：

```
<!--static ディレクトリからの static 画像-->


<!--assets ディレクトリからの webpack でバンドルされた 画像-->
<img src("~/assets/my-image-2.png" />
```

Info

Nuxt はこのパスを変更しないため `router.base` をカスタマイズする場合、手動でパスを追加する必要があります。例：

```

```

静的ディレクトリの設定

必要に応じて `nuxt.config.js` ファイルで `static/` ディレクトリの動作を設定できます。

静的アセットのプレフィックス

Nuxt をサブフォルダにデプロイする場合（例：`/blog/` の場合）ルータベースはデフォルトで `static` アセットパスに追加されます。この動作を無効にしたいなら、`nuxt.config.js` で `static.prefix` を `false` に設定します。

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

デフォルト : `/blog/my-image.png`

`static.prefix` を無効にした場合 : `/my-image.png`

[Go to TOC](#)

ストアディレクトリ

`store` ディレクトリには Vuex ストアに関するファイルが含まれています。Vuex ストアは Nuxt に付属していくつすぐに使えますが、デフォルトでは無効化されています。このディレクトリ内に `index.js` ファイルを作成することで、ストアが有効化されます。

Warning

このディレクトリは追加の設定無しにリネームすることはできません。

ストアを使って状態を管理するのは、どういった大きなアプリケーションを作るにあたっても重要なことです。それがなぜ Nuxt がそのコアに Vuex を組み込んでいるかの理由です。

ストアを有効化する

Nuxt は `store` ディレクトリを確認します。もし隠しファイルや `README.md` ではないファイルが含まれていたら、ストアは有効化されます。これは Nuxt が以下を行うことを意味しています：

1. Vuex をインポートする
2. ルートの Vue インスタンスに `store` オプションを追加します。

モジュール

`store` ディレクトリの中にあるすべての `.js` ファイルは名前空間付きモジュールに変換されます（`index` はルートモジュールになります）。`state` の値は必要に状態がサーバサイドで共有されてしまうことを避けるため、常に `function` であるべきです。

はじめに、ステートを関数で、ミューテーションとアクションをオブジェクトでエクスポートしましょう。

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

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

次に、`store/todos.js` ファイルを作成しましょう：

```
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
    }
}
}

```

ストアは以下のように生成されるでしょう：

```

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
        }
      }
    }
  }
})

```

また、`pages/todos.vue` では `todos` モジュールを以下のように利用することができます：

```

<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>
  </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>

```

モジュールメソッドはストアディレクトリにサブディレクトリを追加することなく、トップレベルでの定義に利用することができます。

ステートの例: 以下のように `store/state.js` を作成します。

```

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

```

対応するミューテーションは `store/mutations.js` に置くことができます

```

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

```

例のフォルダ構造

複雑なストア設定のファイル/フォルダ構造は次のようになります:

```

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

```

```
| state.js
| itemsGroup1/
| state.js
```

ストアでのプラグイン

`store/index.js` ファイルの中に置くことで、ストアにプラグインを追加することができます：

```
import myPlugin from 'myPlugin'

export const plugins = [myPlugin]

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

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

プラグインについてより詳細な情報はこちら: [Vuex のドキュメント](#)

nuxtServerInit アクション

ストアに `nuxtServerInit` アクションが定義されていて、かつ `universal` モードの場合、Nuxt はコンテキストを渡してこれを呼び出します（サーバサイドのみ）。これはサーバにある何らかのデータを直接クライアントサイドに渡すときに便利です。

例えば、サーバサイドにセッションがあり、`req.session.user` で接続したユーザにアクセスできるとしましょう。認証されたユーザをストアに渡すには、`store/index.js` を以下のように変更します：

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

Warning

このアクションは受け取るのは (`store/index.js` 内の) プライマリモジュールだけです。ここから他のモジュールのアクションに呼び出しを繋いでいく必要があります。

コンテキストは `asyncData` メソッドのように、`nuxtServerInit` の第 2 引数として渡されます。

`nuxt generate` が走った場合、`nuxtServerInit` は全ての動的に生成されたルートで実行されます。

Info

非同期の `nuxtServerInit` アクションは、Promise を返すか `async/await` を利用して nuxt サーバが待機できるようにしなければなりません。

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

Vuex 厳格モード

厳格モードは dev モードではデフォルトで有効になっていて、production モードでは無効になっています。 厳格モードを dev モードでも無効化するには、`store/index.js` を下記の例に従ってください：

```
export const strict = false
```

[Go to TOC](#)

Nuxt 設定ファイル

Nuxt ではデフォルトの設定でほとんどのユースケースをカバーしています。`nuxt.config.js` を使ってこの設定を上書きすることができます。

nuxt.config.js

alias

このオプションで JavaScript や CSS で利用可能なエイリアスを定義できます。

```
import { resolve } from 'path'

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

Next

[alias プロパティ](#) の詳細を見る

build

このオプションで、`loaders`、`filenames` や `webpack` の設定、`transpilation` を含む `build` ステップにおけるさまざまな設定を行うことができます。

```
export default {
  build: {
    /*
      ** ここで webpack の設定を拡張することができます。
    */
    extend(config, ctx) {}
  }
}
```

Next

[build プロパティ](#) の詳細を見る

CSS

このオプションで、グローバルに（すべてのページで）利用したい CSS ファイル/モジュール/ライブラリを指定できます。

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

nuxt の設定ファイルで、css プロパティの配列に記述する CSS、SCSS、Postcss、Less、Stylus などのファイルの拡張子は省略することができます。

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

拡張子を省略することで、例えば css ファイルを使用していて sass を使うように変更した場合でも、ファイル名が同じままであれば新しい拡張子が使用されるので、nuxt.config を更新する必要はありません。

Next

[css プロパティ](#) の詳細を見る

dev

このオプションで、Nuxt の `development` または `production` モードを定義できます。(Nuxt をプログラム上で使う際に重要です)

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

Next

[dev プロパティ](#) の詳細を見る

env

このオプションを使うと、`NODE_ENV=staging` や `VERSION=1.2.3` のように、ビルト時（ランタイムではなく）に必要な環境変数を定義することができます。ただし、ランタイム環境変数の場合は `runtimeConfig` が必要です。

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

runtimeConfig

ランタイム設定にはより良いセキュリティと高速な開発のため `dotenv` サポートが組み込まれています。ランタイム設定は Nuxt ペイロードに追加されるので、開発中、サーバーサイドレンダリング、またはクライアントサイドのみのアプリケーションで作業する際に、ランタイム設定を更新するためにリビルドする必要はありません。（静的サイトでは変更を確認するためにサイトをリビルドする必要があります）

.env サポート

もしプロジェクトのルートディレクトリに `.env` ファイルがあるなら、自動的に `process.env` にロードされ、`nuxt.config` / `serverMiddleware` またはそれらがインポートする別のファイル内からアクセスできます。

`--dotenv <file>` を使用してパスをカスタマイズすることができます、また `--dotenv false` を使用して完全に無効にすることができます。例えば本番、ステージング、開発環境で違う `.env` ファイルを指定することができます。

publicRuntimeConfig

- フロントエンドに公開されるのでパブリックな env 変数はすべて保持しなければなりません。例として公開 URL への参照を含めることができます。
- サーバーとクライアントの両方で `$config` を使って利用できます。

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

privateRuntimeConfig

- プライベートでフロントエンドでは公開されてはいけないすべての env 変数を保持しなければなりません。例として API のシークレットトークンへの参照を含めることができます。
- サーバーのみで同じく `$config` を使って利用できます。(publicRuntimeConfig を上書きします)

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

設定値を使用する:

ページ、ストア、コンポーネント、プラグインのコンテキストを使って、`this.$config` や `context.$config` を使用することで、どこからでもこれらの値にアクセスすることができます。

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

テンプレートの中では、`$config.*` を使って直接ランタイム設定にアクセスできます。

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

Warning

サーバーのみのコンテキスト以外で `$config` を使用した場合（例えば `fetch`、`asyncData`、あるいはテンプレート内で直接 `$config` を使用した場合など）、プライベートな設定が公開される可能性があります。

Next

[runtimeConfig の詳細を見る](#)

Next

[@nuxtjs/dotenv からランタイム設定への移行](#) のブログ記事を見る

Next[env プロパティの詳細を見る](#)**generate**

このオプションで、Nuxt によって HTML ファイルに変換されるアプリケーション内のすべての動的ルートのパラメータを設定することができます。

```
export default {
  generate: {
    dir: 'gh_pages', // dist/ の代わりに gh_pages/ を設定する
    subFolders: false // HTML ファイルがルートパスに従って生成されます
  }
}
```

Next[generate プロパティの詳細を見る](#)**head**

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

このオプションで、アプリケーションのデフォルトのメタタグを全て指定できます。

Next[head インテグレーションの詳細を見る](#)**loading**

このオプションで、Nuxt のデフォルトのローディングコンポーネントをカスタマイズできます。

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

Next[loading インテグレーションの詳細を見る](#)**modules**

このオプションで、プロジェクトに Nuxt モジュールを追加できます。

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

Next[modules プロパティの詳細を見る](#)**modulesDir**

modulesDir プロパティは、モジュールディレクトリの設定でパス解決のために使用します。例えば Webpack の resolveLoading、nodeExternals や postcss です。設定パスは `options.rootDir`（デフォルト: `process.cwd()`）からの相対パスになります。

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

プロジェクトが Yarn ワークスペーススタイルのモノリソジトリで構成されている場合はこのフィールドが必要になるかもしれません。

Next[modulesDir プロパティの詳細を見る](#)**plugins**

このオプションで、ルートの Vue.js アプリケーションをインスタンス化する前に実行したい JavaScript plugin を指定できます。

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

Next[plugins プロパティの詳細を見る](#)**router**

`router` オプションで、Nuxt のデフォルトの Vue Router 設定を上書きできます。

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

Next[router プロパティの詳細を見る](#)

server

このオプションで、Nuxt アプリケーションのサーバーインスタンスにおける接続変数を設定できます。

```
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

[server プロパティ](#) の詳細を見る

srcDir

このオプションで、Nuxt アプリケーションのソースディレクトリを指定できます。

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

`client` ディレクトリにある Nuxt アプリケーションのプロジェクト構造の例です。

```
**- app/
--- node_modules/
--- nuxt.config.js
--- package.json
--- client/
| assets/
| components/
| layouts/
| middleware/
| pages/
| plugins/
| static/
| store/**
```

dir

このオプションで、Nuxt のディレクトリのカスタムネームを指定できます。

```
export default {
  dir: {
    pages: 'views' // Nuxt は pages/ フォルダの代わりに views/ を探します。
  }
}
```

Next

[dir プロパティ](#) の詳細を見る

pageTransition

このオプションで、ページトランジションのデフォルトプロパティを指定できます。

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

Next

[transition プロパティ](#) の詳細を見る

その他の設定ファイル

`nuxt.config.js` の他にもプロジェクトのルートに `.eslintrc`、`prettier.config.json` や `.gitignore` などの設定ファイルがあるかもしれません。これらは linter やコードフォーマッタ、git リポジトリなどの他のツールを設定するために使われ、`nuxt.config.js` から切り離されています。

.gitignore

`.gitignore` ファイルに以下の項目を追加して、バージョン管理から無視かつ追加されないようにする必要があります。
`node_modules` フォルダはインストールしたモジュールがすべて入っているフォルダです。`nuxt` フォルダは、dev コマンドや build コマンドを実行したときに作成されるフォルダです。`dist` フォルダは generate コマンドの実行時に作成されるフォルダです。

```
node_modules .nuxt dist
```

この次は？

Next

[設定用語集](#) を見てみましょう。

[Go to TOC](#)

アセットディレクトリ

`assets` ディレクトリには、Stylus や Sass ファイル、画像、フォントなどコンパイルされていないアセットが含まれます。

画像

`vue` テンプレートの中で `assets` ディレクトリにリンクする必要がある場合は、`assets` の前にスラッシュを付け `~/assets/your_image.png` と記述します。

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

`css` ファイルの中で `assets` ディレクトリにリンクする必要がある場合は、スラッシュなしで `~assets/your_image.png` と記述します。

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

動的イメージを使う場合は、`require` を使う必要があります。

```

```

Next

[webpack Assets](#) の詳細についてはこちらをご覧ください。

スタイル

Nuxt では、グローバル（すべてのページ）に設定したい CSS ファイルやモジュール、ライブラリを定義することができます。`nuxt.config` 内で CSS プロパティを使って簡単にスタイルを追加することができます。

```
export default {
  css: [
    // 直接 Node.js モジュールをロード（ここは Sass ファイル）
    'bulma',
    // プロジェクト内の CSS ファイル
    '~/assets/css/main.css',
    // プロジェクト内の SCSS ファイル
    '~/assets/css/main.scss'
  ]
}
```

Sass

`sass` を使用したい場合、`sass` と `sass-loader` プラグインがインストールされているか確認してください。

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

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

Nuxt は拡張子からファイルの種類を自動的に推測して webpack に適したプリプロセッサローダーを使用します。それでも必要な場合は、必要なローダーをインストールする必要があります。

フォント

ローカルフォントをアセットフォルダに追加することで使用できるようになります。追加したら @font-face を使用して css からアクセスすることができます。

```
-l assets
---| 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

CSS ファイルは自動的にロードされません。それらを追加するには [CSS 設定プロパティ](#) を使用してください。

Next

Google フォントのような外部フォントを追加する場合は [Meta Tags and SEO chapter](#) を確認してください。

Webpack アセット

デフォルトでは、Nuxt は webpack の vue-loader、file-loader、url-loader を使用してアセットを提供します。また webpack を通して実行されるべきではないアセットのために、静的ディレクトリを使用することもできます。

Webpack

[vue-loader](#) は、[css-loader](#) と Vue Template Compiler を用いて、スタイルやテンプレートファイルを自動的に処理します。このコンパイル処理の中で、`` や `background: url(...)` や CSS `@import` などのすべてのアセット URL はモジュールの依存関係として解決されます。

例えば、次のようなファイル構成があるとします：

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

CSS で `url('~assets/image.png')` と記述した場合、それは `require('~/assets/image.png')` に変換されます。

Warning

`~` エイリアスは CSS ファイルで正しく解決されないでしょう。CSS の `url` の参照には、`~assets` (**スラッシュなし**) を使わなければなりません。例: `background: url(~assets/banner.svg")`

`pages/index.vue` で画像を参照する場合 :

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

次のようにコンパイルされます :

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

`.png` は JavaScript ファイルではないため、Nuxt は `file-loader` (<https://github.com/webpack/file-loader>) と `url-loader` を使ってそれらを処理できるよう `webpack` を設定します。

これらのローダーを利用する利点 :

`file-loader` は、アセットファイルをコピー・配置する場所と、キャッシュ改善のためにバージョンハッシュを用いてファイル名を指定することができます。本番環境では、デフォルトで長期的なキャッシングの恩恵を受けることができます。

`url-loader` は、指定した閾値よりも小さい場合に、Base64 データ URL として条件付きでファイルに埋め込むことができます。これにより、小さなファイル取得のための HTTP リクエスト数を減らすことができます。もし閾値よりも大きい場合は、`file-loader` に自動的にフォールバックします。

これら 2 つのローダーのデフォルトの設定は次の通りです :

```
// 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]'
      }
    }
]
}

```

つまり、1 KB 未満のすべてのファイルは Base64 データ URL としてインライン化されます。それ以外の場合、画像やフォントは、対応するフォルダ（`.nuxt` ディレクトリの下）にコピーされ、より良いキャッシュのためにバージョンハッシュを含む名前が付けられます。

`nuxt` コマンドでアプリケーションを起動するときの `pages/index.vue` のテンプレートは以下の通りです：

```

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

```

次のように生成されます：

```

```

ローダの設定を変更したい場合は `build.extend` を使用してください。

エイリアス

デフォルトでは、ソースディレクトリ (`srcDir`) とルートディレクトリ (`rootDir`) は同じディレクトリを意味します。ソースディレクトリには `~` のエイリアスを使うことができます。`../assets/your_image.png` のような相対パスを記述する代わりに、`~/assets/your_image.png` と記述することができます。

どちらも同じ結果になります。

```

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

```

エイリアスとして `~` を使うことをお勧めします。`@` はまだサポートされていますが、CSS の `background image` などすべてのケースで機能するわけではありません。

ルートディレクトリには、エイリアスの `~~` や `@@` を使用することができます。

Info

Tip: スペイン語のキーボードでは、Mac OSでは (`Option + ñ`)、または Windows では (`Alt gr + 4`) で `~` にアクセスできます。

[Go to TOC](#)

コンポーネントディレクトリ

`components` ディレクトリには、Vue.js コンポーネントが含まれています。コンポーネントは、ページのさまざまな部分を構成するものであり、再利用して、ページ、レイアウト、さらには他のコンポーネントにインポートすることができます。

データの取得

コンポーネントの API から非同期データにアクセスするには、`fetch()` を使用できます。

`$fetchState.pending` をチェックすることで、データがロードされるのを待っている時にメッセージを表示することができます。`$fetchState.error` を確認し、データの取得中にエラーが発生した場合はエラーメッセージを表示することもできます。`fetch()` を使用する場合、`data()` で適切なプロパティを宣言する必要があります。フェッチから取得したデータは、これらのプロパティに割り当てることができます。

```
<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

フェッチの動作の詳細については、[fetch\(\)](#) の章を参照してください

コンポーネントの探索

:prose-img{src="../../../../../static/img/docs/components.png"}

v2.13 以降、Nuxt はテンプレートで使用される時にコンポーネントを自動でインポートできます。この機能をアクティブにするには、構成を `components: true` と設定します：

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

`~/components` ディレクトリにあるコンポーネントは、明示的にインポートしなくても、ページやレイアウト（および他のコンポーネント）全体で使用することができます。

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

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

Next

componentsモジュールの詳細は、[コンポーネントの探索 ドキュメント](#)と、[お知らせ記事](#)でご覧いただけます。

動的インポート

コンポーネントを動的にインポートするには、コンポーネントの遅延読み込みともいわれる、`Lazy` プレフィックスをテンプレートに追加するだけです。

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

`lazy` プレフィックスを使用すると、イベントが発火された時にコンポーネントを動的にインポートすることもできます。

```
<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>
```

ネストされたディレクトリ

次のようにネストされたディレクトリにコンポーネントがある場合：

```
components/
base/
  foo/
    CustomButton.vue
```

コンポーネント名は独自のパスディレクトリとファイル名に基づいています。したがってコンポーネントは次のようにになります：

```
<BaseFooCustomButton />
```

ディレクトリ構造を保ったまま、`<CustomButton />` として使用したい場合は、`nuxt.config.js` に `CustomButton.vue` のディレクトリを追加します。

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

そして、`<BaseFooCustomButton />` の代わりに `<CustomButton />` を使うことができます。

```
<CustomButton />
```

Next

コンポーネント名を制御する他の方法については、[components プロパティ](#) を参照してください。

Info

[components モジュール](#)についてはこちらをご覧ください。

[Go to TOC](#)

コンテンツディレクトリ

`@nuxt/content` モジュールを使うことで Nuxt アプリケーションを強化できます。このモジュールは `content/` ディレクトリへの書き込みや、**Git ベースのヘッドラス CMS** のように動作する MongoDB のような API を介してマークダウン、JSON、YAML や CSV ファイルを取得できます。

Nuxt Content Module

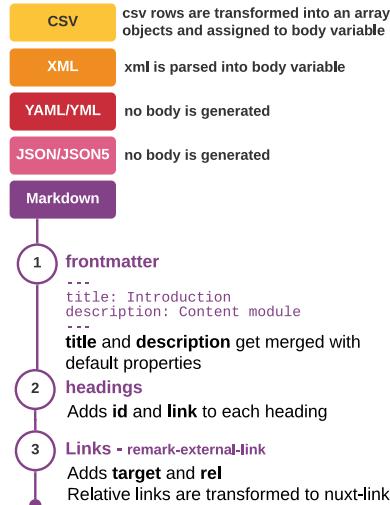
1. WRITE

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

Nuxt 2.13+ | @nuxt/content 1.4.1

July 2020 - Page 1

Created by @KrutiePatel

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
path is file, path is dir
fetch returns Promise<Object>, fetch returns Promise<Array>

Example:

```
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
  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>

開発時のホットリロード

マークダウンファイルに変更があった場合 webpack を介する必要がないので、開発時における content モジュールのホットリロードは非常に高速です。また、`content:update` イベントのリッスンやプラグインを作ることもできるので content 内のファイルを更新するたびに例えば `fetchCategories` メソッドをディスペッチすることができます。

Next

詳細は [content モジュールのドキュメントのホットリロードの取り扱い](#) を参照してください

content の表示

ページの本文を表示するために `<nuxt-content>` コンポーネントをテンプレートで直接使えます。

```
<template>
  <article>
    <nuxt-content :document="article" />
  </article>
</template>
```

Next

詳細は [content モジュールのドキュメントのコンポーネント](#) を参照してください。

content のスタイル

アプリケーションの設計に使っているものによってはマークダウンを適切に表示するために何らかのスタイルを作成する必要があるかもしれません。

`<nuxt-content>` コンポーネントは自動的に `.nuxt-content` class を追加します。この class を使ってスタイルをカスタマイズできます。

```
<style>
  .nuxt-content h2 {
    font-weight: bold;
    font-size: 28px;
  }
  .nuxt-content p {
    margin-bottom: 20px;
  }
</style>
```

Next

詳細は [content モジュールのドキュメントのスタイル](#) を参照してください。

マークダウン、CSV、YAML、JSON(5)を扱う

このモジュールは `.md` ファイルを JSON AST ツリー構造に変換し `body` 変数に格納します。 YAML front matter ブロックをマークダウンファイルまたはドキュメントに挿入される `.yaml` ファイルに追加することもできます。また、同様にドキュメントに挿入できる `json/json5` ファイルや行が `body` 変数に割り当てられる `.csv` ファイルを追加することもできます。

```
---  
title: My first Blog Post  
description: Learning how to use @nuxt/content to create a blog  
---
```

Next

詳細は [content モジュールのドキュメントのマークダウン](#) を参照してください。

マークダウン内の Vue コンポーネント

マークダウンファイルで Vue コンポーネントを直接使えます。ただし、コンポーネントをケバブケースで参照する必要があり自動終了タグは使えません。

```
<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

詳細は [content モジュールのドキュメントの vue コンポーネント](#) を参照してください。

完全検索可能な API

`$content()` を使って content を簡単に一覧表示したりフィルタしたり検索したりできます。

```
<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>
```

Next

詳細は [content モジュールのドキュメントのメソッド一覧](#) を参照してください。

前の記事と次の記事

content モジュールには `.surround(slug)` が含まれているので前の記事と次の記事を簡単に取得できます。

```

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

詳細は [content モジュールのドキュメントの surroundslug オプション](#)を参照してください。

全文検索

content モジュールには全文検索が備わっているので何もインストールしなくてもマークダウンファイルを簡単に検索できます。

```

<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

詳細は [content モジュールのドキュメントの search](#) を参照してください。

シンタックスハイライト

このモジュールは自動的にコードブロックをラップし、[Prism class](#) を適用します。異なる Prism テーマを使ったり無効化したりすることもできます。

```
yarn add prism-themes
```

```
npm install prism-themes
```

```
content: {
  markdown: {
    prism: {
      theme: 'prism-themes/themes/prism-material-oceanic.css'
    }
  }
}
```

Next

詳細は [content モジュールのドキュメントのシンタックスハイライト](#) を参照してください。

マークダウンのパースを拡張する

もともとマークダウンはコードブロック内の行やファイル名の強調表示をサポートしていません。content モジュールでは独自のカスタム構文を使えます。行番号は `data-line` 属性の `pre` タグに追加され、ファイル名は `filename class` の `span` に変換されるのでスタイルを設定できます。

Next

詳細は [content モジュールのドキュメントのコードブロック](#) を参照してください。

目次の生成

TOC (Table of Contents/ 目次) 配列プロパティがドキュメントに挿入され、すべての見出しとそのタイトル、ID の一覧が表示されるのでそれらにリンクできます。

```
<nav>
  <ul>
    <li v-for="link of article.toc" :key="link.id">
      <NuxtLink :to="`#${link.id}`">{{ link.text }}</NuxtLink>
    </li>
  </ul>
</nav>
```

Next

詳細は [content モジュールのドキュメントのテーブルコンテンツ](#) を参照してください。

強力なクエリビルダー API

content モジュールには MongoDB に似た強力なクエリビルダー API が備わっており、
http://localhost:3000/_content/ で各ディレクトリの JSON を簡単に確認できます。エンドポイントは GET や POST リクエストでアクセスできるのでクエリパラメータを使えます。

```
http://localhost:3000/_content/articles?only=title&only=description&limit=10
```

Next

詳細は [content モジュールのドキュメントの API エンドポイント](#) を参照してください。

フックを使った拡張

フックを使用してモジュールを拡張し、ドキュメントを保存する前にデータをドキュメントに追加できます。

Next

詳細は [content モジュールのドキュメントのフック](#) を参照してください。

@nuxtjs/feed を使った統合

記事の場合、content は [@nuxtjs/feed](#) モジュールを使ってニュースフィードを生成することに使えます。

Next

詳細は [content モジュールのドキュメントの @nuxtjs/feed との統合](#) を参照してください。

静的サイト生成のサポート

content モジュールは `nuxt generate` を使って静的サイト生成ができます。nuxt クローラ機能のおかげですべてのルートが自動的に生成されます。

Warning

もし Nuxt v2.12 以下を使っていて動的ルートを指定する必要がある場合は、generate プロパティを使ってプログラムから `@nuxt/content` を使って指定できます。

Next

プログラムでの扱い方についての詳細は [content モジュールのドキュメントのプログラマティックな利用方法](#) を参照してください

この次は

Next

チュートリアル [How to Create a Blog with Nuxt Content](#) を参照してください。

Next

高度な使用方法と例については [content モジュールのドキュメント](#) を参照してください。

[Go to TOC](#)

配布ディレクトリ

`dist` は *distribution* の略です。`dist` フォルダは `nuxt generate` コマンドを使用すると動的に生成されます。`dist` フォルダ内には、静的に生成された Nuxt アプリケーションをデプロイ、実行するために必要となる本番用の HTML ファイルとアセットが含まれています。

デプロイ

これは静的ホスティングのためにアップロードする必要があるフォルダで、本番用の HTML ファイルとアセットを含んでいます。

Warning

`dist` ディレクトリはバージョン管理システムにコミットされるべきではありません。`nuxt generate` を行うたびに自動的に生成されるので、`.gitignore` で無視してください。

dir プロパティ

`dist` フォルダは、デフォルトでは `dist` という名前になっていますが `nuxt.config` ファイルで設定することができます。

```
generate: {
  dir: 'my-site'
}
```

Warning

もし `dist` フォルダの名前を変更する場合は、`git` がそれを無視するようにバージョン管理に追加する必要があります。

subFolders プロパティ

Nuxt は、デフォルトでは生成された全てのページをフォルダ内に配置しますが、`nuxt.config` ファイルを変更して `subFolders` を `false` にすることでこれを変更することができます。

```
generate: {
  subFolders: false
}
```

fallback プロパティ

サイトをデプロイする際、フォールバックの `html` パスが正しくセットされていることを確認する必要があります。不明なルートが Nuxt 経由でレンダリングされるようにエラーページとして設定する必要があります。もし設定されていない場合、Nuxt は `200.html` のデフォルト値を使用します。

シングルページアプリケーションを実行しているときは、他のルートが生成されず、必要なファイルは `200.html` だけなのでそれを使用するほうが理にかなっています。

静的に生成されたページで作業する場合、エラーページ用に `404.html` を使用することをおすすめします。

Warning

ホスティングサービスによっては 200.html もしくは 404.html を使わなければならないかもしれません。ホスティングプロバイダで確認してください。例えば Netlify は 404.html を使用しています。

```
export default {
  generate: {
    fallback: '404.html'
  }
}
```

excludes プロパティ

generate の excludes プロパティを使用することで、生成されるページを除外することができます。静的ページとして生成される代わりにシングルページアプリケーションへとフォールバックされ、クライアントサイドのみでレンダリングされます。

```
generate: {
  exclude: [/admin/]
}
```

Info

また、正規表現を使用して、特定の単語で始まるページや終わるページ除外することもできます。

[Go to TOC](#)

レイアウトディレクトリ

レイアウトは Nuxt アプリケーションのルック&フィールを変えるとき、とても役に立ちます。例えばサイドバーを含めたいときや、モバイルとデスクトップのための別々のレイアウトを持ちたいときなどに有用です。

Warning

このディレクトリは追加の設定なしに名前を変更できません

デフォルトレイアウト

`layouts/default.vue` ファイルを追加することでメインレイアウトを設定できます。これはレイアウトが指定されていない全てのページで使用されます。ページコンポーネントを含むレイアウトを作成するときには `<Nuxt>` コンポーネントを記述することを忘れないようにしてください。

レイアウトに必要なのは、ページコンポーネントをレンダリングするための 3 行のコードだけです。

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

ナビゲーションやヘッダー、フッターのようなコンポーネントもここに追加できます。

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

Info

もし `components` プロパティを `true` に設定していたら、コンポーネントを追加するために `import` 文は必要ないです。

カスタムレイアウト

`layout` ディレクトリの全てのファイル（トップレベル）は、ページコンポーネントの `layout` プロパティで使えるカスタムレイアウトが作られます。

ブログレイアウトを作成して `layouts/blog.vue` に保存しましょう：

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

その後、カスタムレイアウトを使用するためにページに教えなければなりません。

```
<script>
export default {
  layout: 'blog',
  // OR
  layout (context) {
    return 'blog'
  }
}
</script>
```

エラーページ

エラーページはエラーが発生したときいつも表示される ページコンポーネント です（サーバーサイドで発生したエラーではない）。

Warning

このファイルは `layout` フォルダに置かれますが、ページとして扱われるべきです。

上述したように、このレイアウトは特殊でテンプレートの中に `<Nuxt>` を含めるべきではありません。このレイアウトは、エラーが発生したときに表示されるコンポーネントとして表示されなければなりません（`404`、`500` など）。他のページコンポーネントと同様に、エラーページにも通常の方法でカスタムレイアウトを設定することができます。

`layouts/error.vue` ファイルを追加することで、エラーページをカスタマイズできます：

```
<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' // エラーページ用のカスタムレイアウトを設定できます
}
</script>
```

Info

デフォルトのエラーページのソースコードは [GitHub](#) で参照できます。

[Go to TOC](#)

ミドルウェアディレクトリ

`middleware` ディレクトリには、アプリケーションミドルウェアが含まれています。ミドルウェアを使用すると、ページまたはページのグループ（レイアウト）をレンダリングする前に実行できる、カスタム関数を定義できます。

共有ミドルウェアは、`middleware/` ディレクトリに配置する必要があります。ファイル名はミドルウェアの名前になります（`middleware/auth.js` は `auth` ミドルウェアになります）。関数を直接使用して、ページ固有のミドルウェアを定義することもできます。[匿名ミドルウェア](#)を参照してください。

ミドルウェアは最初の引数として [コンテキスト](#) を受け取ります。

```
export default function (context) {
  // userAgentプロパティをコンテキストに追加します
  context.userAgent = process.server
    ? context.req.headers['user-agent']
    : navigator.userAgent
}
```

ユニバーサルモードでは、ミドルウェアはサーバ側（Nuxt アプリへの最初のリクエスト時、たとえばアプリに直接アクセスするときやページを更新するとき）で一度呼び出され、クライアント側でさらにルートに移動するときに呼び出されます。`ssr: false` を使用すると、どちらの状況でもクライアント側でミドルウェアが呼び出されます。

ミドルウェアは以下の順序で連続して実行されます：

1. `nuxt.config.js` (ファイル内の順序)
2. マッチしたレイアウト
3. マッチしたページ

ルーターミドルウェア

ミドルウェアは非同期にすることができます。これを行うには `Promise` を返すか、`async/await` を使用します。

```
import http from 'http'

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

次に、`nuxt.config.js` で、`router.middleware` キーを使用します。

```
export default {
  router: {
    middleware: 'stats'
  }
}
```

これで、ルートが変更されるたびに `stats` ミドルウェアが呼び出されます。

ミドルウェア（複数でも）を特定のレイアウト、またはページに追加することもできます。

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

名前付きミドルウェア

`middleware/` ディレクトリ内にファイルを作成することで、名前付きミドルウェアを作成できます。ファイル名はミドルウェア名になります。

```
export default function ({ store, redirect }) {
  // ユーザーが認証されていない場合
  if (!store.state.authenticated) {
    return redirect('/login')
  }
}
```

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

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

匿名ミドルウェア

特定のページにのみミドルウェアを使用する必要がある場合は、そのミドルウェアの関数（または関数の配列）を直接使用できます：

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

<script>
  export default {
    middleware({ store, redirect }) {
      // ユーザーが認証されていない場合
      if (!store.state.authenticated) {
        return redirect('/login')
      }
    }
  }
</script>
```

モジュールディレクトリ

Nuxt は高度な (higher-order) モジュールシステムを提供し、コアの拡張を可能にしています。モジュールは、Nuxt の起動時に順次呼び出される関数です。

Nuxt モジュールを探す

Nuxt プロジェクトを強化するために、Nuxt チームとコミュニティによって作成された[モジュールの一覧](#)を見てください。

- 165 以上のモジュール
- 105 人以上のメンテナ

Next

modules.nuxtjs.org を確認してください。

Nuxt Modules

modules are Nuxt extensions that extend Nuxt's core functionalities

Nuxt v2.14
September 2020

property type:
array

modules

Called once, in sequence,
while booting Nuxt app
i.e. npm run dev

```
// nuxt.config.js
export default {
  modules: [
    // Using package name
    '@nuxtjs/axios',
    // Relative to project srcDir
    '~/modules/simple',
    // Providing options
    ['~/modules/simple', { token: '123' }],
    // Inline definition
    function () {}
  ]
}
```

array item can be
string, array with
options or function

Write your own module

Combine multiple options
sources

Top-level options

```
// nuxt.config.js
export default {
  modules: [
    ['@nuxtjs/axios', { anotherOption: true }],
    // axios module is aware of this by using `this.options.axios`
    axios: { option1, option2 }
  ]
}
```

Access Nuxt options, including
module's options
i.e. add a CSS library or
emit assets

this.options

```
// modules/module.js
export default function (moduleOptions) {
  // `options` will contain option1,option2 & anotherOption
  const options = Object.assign({}, this.options.axios,
    moduleOptions
  )
}
```

Refers to the current Nuxt
instance, allowing us
to register hooks on certain
life-cycle events
i.e. provide plugins or
register custom webpack loaders

this.nuxt

```
// modules/module.js
export default function (moduleOptions) {
  this.nuxt.hook('ready', async nuxt => {})
  this.nuxt.hook('error', async error => {})
  this.nuxt.hook('close', async nuxt => {})
  this.nuxt.hook('listen', async(server, {host, port})) => {}
}
```

this refers to the
moduleContainer
instance

create async modules that
return a promise or
call a callback

async module

```
// modules/mymodule.js
// Use async/await
export default async function asyncModule() {
  const pages = await ...
}
// Return a promise
export default function asyncModule() {
  return fetch().then(response => ...)
}
```

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

Nuxt を使って本番相当のアプリケーションを開発しているときに、フレームワークのコア機能が十分でないことに気付くかもしれません。Nuxt は設定オプションとプラグインで拡張できますが、複数のプロジェクト間でこれらのカスタマイズを維持するのは面倒で反復的で時間がかかります。一方、すべてのプロジェクトのニーズをすぐにサポートすると、Nuxt は非常に複雑になり使いにくくなります。

これが Nuxt がコア機能を簡単に拡張できるようにするために、より高度なモジュールシステムを導入する理由の 1 つです。モジュールは Nuxt 起動時に順番に呼び出される関数です。フレームワークは Nuxt が処理を続ける前に各モジュールが処理を完了するまで待機します。このようにして、モジュールは Nuxt のほとんどすべての項目をカスタマイズできます。Webpack の [Tapable](#) に基づいた Nuxt のモジュール設計のおかげで、モジュールは例えればビルドの初期化のような特定のエントリーポイントにフックを簡単に登録できます。また、モジュールはテンプレートの上書き、webpack ローダーの設定、CSS ライブラリの追加、その他多くの便利なタスクを実行することができます。

素晴らしいことに Nuxt モジュールは npm パッケージに組み込めます。これによりプロジェクト間で再利用したり、コミュニティでシェアすることができるようになります。高品質なアドオンのエコシステムを作成できます。

modules プロパティ

モジュールは Nuxt の拡張機能であり、フレームワークのコア機能を拡張し無限に組み込めます。一度モジュールをインストールすると、`nuxt.config.js` ファイルの `modules` プロパティにモジュールを追加できます。

```
export default {
  modules: [
    // 使うパッケージ名
    '@nuxtjs/axios',

    // プロジェクトのソースディレクトリの相対パス
    '~/modules/awesome.js',

    // オプションの提供
    ['@nuxtjs/google-analytics', { ua: 'X1234567' }],

    // インライン定義
    function () {}
  ]
}
```

Info

モジュール開発者は通常、使い方について追加で必要な手順と詳細を提供します。

Nuxt は node が要求するパス（`node_modules` 内）を使ってモジュール配列内の各項目を解決しようとします。@ エイリアスが使われている場合はプロジェクトの `srcDir` から解決します。

Warning

モジュールは順番に実行されるため、順序が重要です。

モジュールはビルトや実行を強化する関数をエクスポートし、オプションでジョブが終了するまで promise を返す必要があります。実行時にインポートされるので、最新の ES6 機能を使っている場合は事前にトランスペイルする必要があることに注意してください。

独自のモジュールを書く

モジュールは関数です。モジュールは npm モジュールとしてパッケージ化することもプロジェクトのソースコードに直接含めることができます。

```
export default {
  exampleMsg: 'hello',
  modules: [
    // シンプルな使い方
    '~/modules/example',
    // オプションを直接渡す
    ['~/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')
  })
}

// モジュールを npm パッケージとして公開する場合は必須
module.exports.meta = require('./package.json')
```

1) ModuleOptions

`moduleOptions`：これは `modules` の配列を利用するため、モジュールの利用者から渡されるオブジェクトです。これを使うことで `modules` のふるまいをカスタマイズすることができます。

トップレベルのオプション

`nuxt.config.js` にモジュールを登録するときにトップレベルでオプションを使えると便利な場合があります。このオプションにより、複数のオプションソースを組み合わせることができます。

```
export default {
  modules: [['@nuxtjs/axios', { anotherOption: true }]],
  // axios モジュールは `this.options.axios` を使ってこのオプションを認識します
  axios: {
    option1,
    option2
  }
}
```

2) this.options

`this.options`：このリファレンスを使って Nuxt オプションに直接アクセスできます。これはすべてのデフォルトオプションが割り当てられたユーザーの `nuxt.config.js` のコンテンツです。モジュール間の共有オプションに使用えます。

```

export default function (moduleOptions) {
  // `options` には option1, option2, anotherOption が含まれます
  const options = Object.assign({}, this.options.axios, moduleOptions)

  // ...
}

```

CSS ライブラリの追加

モジュールが CSS ライブラリを提供する場合は、重複を避けるためにユーザーがすでにライブラリを含んでいるかどうかを確認し、モジュールで CSS ライブラリを無効にするオプションを追加してください。

```

export default function (moduleOptions) {
  if (moduleOptions.fontAwesome !== false) {
    // Font Awesome を追加
    this.options.css.push('font-awesome/css/font-awesome.css')
  }
}

```

アセットの出力

ビルド中にアセットを出力するために webpack プラグインを登録できます。

```

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) => {
        // info 変数の内容を含む `nuxt/dist/info.txt` が生成されます。
        // ソースもバッファにすることができます
        compilation.assets['info.txt'] = {
          source: () => info,
          size: () => info.length
        }

        cb()
      })
    }
  })
}

```

3) this.nuxt

`this.nuxt`：これは現在の Nuxt インスタンスへの参照です。特定のライフサイクルイベントにフックを登録できます。

- **Ready**：Nuxt は動作する準備ができています（ModuleContainer とレンダラの準備ができます）。

```

nuxt.hook('ready', async nuxt => {
  // カスタムコードはここに記述する
})

```

- **Error**：フックを呼び出すときに捕まえられなかったエラーです。

```
nuxt.hook('error', async error => {
  // カスタムコードはここに記述する
})
```

- **Close:** Nuxt インスタンスは正常に終了します。

```
nuxt.hook('close', async nuxt => {
  // カスタムコードはここに記述する
})
```

- **Listen:** Nuxt の内部サーバーがリッスンを開始します (nuxt start または nuxt dev を使います)

```
nuxt.hook('listen', async (server, { host, port }) => {
  // カスタムコードはここに記述する
})
```

`this`: モジュールのコンテキストです。すべてのモジュールは、ModuleContainer インスタンスのコンテキスト内で呼び出されます。

利用可能なメソッドについては [ModuleContainer class](#) のドキュメントを参照してください。

特定のフックでタスクを実行する

モジュールは Nuxt の初期化中だけでなく、特定の条件でのみ処理を実行する必要がある場合があります。強力な Nuxt フックを使って特定のイベントでタスクを実行できます ([Hookable](#) に基づきます)。Nuxt は Promise か `async` として定義された関数が返されるのを待ちます。

基本的な例をいくつか提示します：

```
export default function myModule() {
  this.nuxt.hook('modules:done', moduleContainer => {
    // これはすべてのモジュールのロードが終了したときに呼び出されます
  })

  this.nuxt.hook('render:before', renderer => {
    // レンダラが作成された後に呼び出されます
  })

  this.nuxt.hook('build:compile', async ({ name, compiler }) => {
    // コンパイラ（デフォルト： webpack）が起動する前に呼び出されます
  })

  this.nuxt.hook('generate:before', async generator => {
    // これは Nuxt がページを生成する前に呼び出されます
  })
}
```

提供されるプラグイン

モジュールが追加されると、1つ以上のプラグインを提供するのが一般的です。たとえば、[bootstrap-vue](#) モジュールは自分自身を Vue に登録する必要があります。このような状況では `this.addPlugin` ヘルパーが使えます。

```

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) {
  // `plugin.js` テンプレートを登録する
  this.addPlugin(path.resolve(__dirname, 'plugin.js'))
}

```

注意: モジュールによって注入されたプラグインは、プラグインリストの 先頭 に追加されます。オプションとして：

- 自分のプラグインをプラグインリストの最後に手動で追加します（`this.nuxt.options.plugins.push(...)`）
- 他のモジュールに依存している場合、モジュールの順序を逆にします

テンプレートプラグイン

登録済みのテンプレートとプラグインは [lodash テンプレート](#) を利用して登録済みのプラグインの出力を条件付きで変更できます。

```

// Google Analytics の UA を設定
ga('create', '<%= options.ua %>', 'auto')

<% if (options.debug) { %>
// 開発向けのコードのみ
<% } %>

import path from 'path'

export default function nuxtGoogleAnalytics(moduleOptions) {
  // `plugin.js` テンプレートを登録する
  this.addPlugin({
    src: path.resolve(__dirname, 'plugin.js'),
    options: {
      // プラグインをプロジェクトにコピーするとき Nuxt は `options.ua` を `123` に置き換えます
      ua: 123,
      // dev を含む条件付きパートは本番ビルトのプラグインコードから削除されます
      debug: this.options.dev
    }
  })
}

```

カスタム webpack ローダーの登録

`this.extendBuild` を使って `nuxt.config.js` の `build.extend` と同じことが行えます。

```

export default function (moduleOptions) {
  this.extendBuild((config, { isClient, isServer }) => {
    // `.foo` ローダー
    config.module.rules.push({
      test: /\.foo$/,
      use: [...]
    })
}

```

```
// 既存のローダーをカスタマイズします
// Nuxt 内部のソースコードを参照してください：
//
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')
}
}
```

非同期モジュール

すべてのモジュールがすべてを同期的に実行するわけではありません。たとえば、API でデータを取得したり非同期操作を実行したりする必要があるモジュールを開発したい場合があります。このため、Nuxt は Promise を返すか、コールバックを呼び出すことができる非同期モジュールをサポートしています。

async/await を使う

```
import fse from 'fs-extra'

export default async function asyncModule() {
  // ここで `async` / `await` を使って非同期作業が行えます
  const pages = await fse.readJson('./pages.json')
}
```

Promise を返す

```
export default function asyncModule($http) {
  return $http
    .get('https://jsonplaceholder.typicode.com/users')
    .then(res => res.data.map(user => '/users/' + user.username))
    .then(routes => {
      // Nuxt ルートを拡張して何かをする
    })
}
```

Info

モジュールにはもっと多くのフックと可能性があります。Nuxt 内部の API についての詳細は [Nuxt Internals](#) を参照してください。

モジュールを公開する

`module.exports.meta`：モジュールを npm パッケージとして公開する場合はこの行が必要です。Nuxt は内部的にメタを使ってパッケージをより適切に処理します。

```
module.exports.meta = require('./package.json')
```

buildModules

一部のモジュールは開発およびビルド時にのみインポートされます。`buildModules` を使うと本番環境での起動が速くなり、本番環境での `node_modules` のサイズが大幅に削減されます。各モジュールのドキュメントを参照し `modules` または `buildModules` の使用が推奨されているかどうかを確認してください。

使い方の違いは以下の通りです：

- `nuxt.config.js` に `modules` を追加するかわりに `buildModules` を使います

```
export default {
  buildModules: ['@nuxtjs/eslint-module']
}
```

- `package.json` に `dependencies` を追加するかわりに `devDependencies` を使います

```
yarn add --dev @nuxtjs/eslint-module
```

```
npm install --save-dev @nuxtjs/eslint-module
```

Info

モジュール作成者の場合、パッケージを `devDependency` としてインストールし、`nuxt.config.js` では `modules` のかわりに `buildModules` を使うことをユーザーに提案することを強くおすすめします。

次の場合を除いてモジュールは `buildModule` です：

- `serverMiddleware` を提供している
- Node.js ランタイムフックを登録する必要がある（例えば Sentry）
- `vue-renderer` の動作に影響しているか、`server:` または `vue-renderer:` 名前空間からのフックを使っている
- webpack スコープ外のその他のもの（ヒント：プラグインとテンプレートはコンパイルされ、webpack スコープ内にある）

Warning

`buildModules` を使って提供する場合、この機能は Nuxt v2.9 以降から利用可能であることに注意してください。昔からのユーザーは、Nuxt をアップグレードするか `modules` セクションを使う必要があります。

ページディレクトリ

`pages` ディレクトリには、アプリケーションのビューとルートが格納されています。Nuxt はこのディレクトリ内のすべての `.vue` ファイルを読み込んで、ルーターの設定を自動的に作成します。

Info

`.js` ファイルや `.ts` ファイルを使ってルートを作成できます。

すべてのページコンポーネントは Vue コンポーネントですが、Nuxt は特殊な属性や機能を追加し、ユニバーサルアプリケーションの開発をできるだけ簡単にします。

```
<template>
  <h1 class="red">Hello {{ name }}!</h1>
</template>

<script>
  export default {
    // ページプロパティはここに
  }
</script>

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

動的なページ

動的なページは、API からの出力でページ名がわからない場合や、同じページを何度も作成したくないときに作成できます。動的なページを作成するには、`.vue` ファイル名の前にアンダースコアを追加します。ディレクトリを動的にしたい場合はディレクトリ名の前にアンダースコアを追加します。ファイル名やディレクトリ名は自由に指定できますが、名前の前にアンダースコアを付ける必要があります。

例えば、ページフォルダ内に `_slug.vue` というファイルがある場合、context の `params.slug` で値にアクセスできます。

```
<template>
  <h1>{{ this.slug }}</h1>
</template>

<script>
  export default {
    async asyncData({ params }) {
      const slug = params.slug // "/abc" パスにアクセスすると、slug は "abc" になります。
      return { slug }
    }
  }
</script>
```

`/_book` フォルダ内に `/_slug.vue` ファイルを作成した場合は、context の `params.slug` および `params.book` で値にアクセスできます。

```
<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>
```

プロパティ

asyncData

`asyncData` はコンポーネントをロードする前に毎回呼び出されます。これは非同期にすることができます、引数として context を受け取ります。`asyncData` の結果は `data` とマージされます。

```
export default {
  asyncData(context) {
    return { name: 'World' }
  }
}
```

Next

`asyncData` の詳細は[データの取得](#)の章を参照してください。

fetch

非同期データの取得には `fetch` が使えます。`fetch` はサーバーサイドではルートをレンダリングするときに呼び出され、クライアントサイドでは遷移するときに呼び出されます。

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

Next

fetch の詳細は[データの取得](#)の章を参照してください。

head

現在のページに特定の タグを設定します。Nuxt は `vue-meta` を使用してアプリケーションのドキュメントヘッドとメタ属性を更新します。

```
export default {
  head() {
    // このページの meta タグを設定する
  }
}
```

Next

詳細は[メタタグと SEO](#)の章を参照してください。

layout

layouts ディレクトリで定義されているレイアウトを指定します。

```
export default {
  layout: 'blog'
}
```

Next

レイアウトの詳細は[ビュー](#)の章を参照してください。

loading

`loading` を `false` に設定すると、ページに入るときに `this.$nuxt.$loading.finish()` が自動的に呼び出されるのを防ぎ、ページを離れるときに `this.$nuxt.$loading.start()` が自動的に呼び出されるのを防ぎます。

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

Info

`nuxt.config.js` で `loading` が設定されている場合のみ適用されます。

Next

詳細は[ローディング](#)の章を参照してください。

transition

このページにカスタムトランジションを設定します。

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

Next

詳細は[トランジション](#)の章を参照してください。

scrollToTop

`scrollToTop` は、ページをレンダリングする前に Nuxt に一番上にスクロールするように指示できます。デフォルトでは別ページに移動すると一番上にスクロールしますが、子ルートの場合はスクロール位置を維持します。子ルートをレンダリングする際に Nuxt に一番上までスクロールするように指示したい場合は、`scrollToTop` を `true` に設定します。

```
export default {
  scrollToTop: true
}
```

逆に、親ルートでも `scrollToTop` を手動で `false` に設定することができます。

スクロールについて Nuxt のデフォルトの挙動を上書きしたいときは、[scrollBehavior オプション](#)を参照してください。

middleware

このページのためのミドルウェアを定義します。このミドルウェアは、ページをレンダリングする前に呼び出されます。

```
export default {
  middleware: 'auth'
}
```

Next

詳細は[ミドルウェア](#)の章を参照してください。

watchQuery

`watchQuery` キーを設定し、監視するクエリ文字列を設定します。定義した文字列が変更されると、すべてのコンポーネントメソッド (`asyncData`、`fetch`、`validate`、`layout`、...) が呼ばれます。パフォーマンス向上のため、監視はデフォルトで無効になっています。

```
export default {
  watchQuery: ['page']
}
```

Warning

警告: 2.12 で導入された新しい `fetch` フックは `watchQuery` によって影響を受けません。より詳細情報については[クエリ文字列の変化のリスニング](#)を参照してください。

```
export default {
  watchQuery: true
}
```

より洗練された監視のために、`watchQuery(newQuery, oldQuery)` 関数を使用することもできます。

```
export default {
  watchQuery(newQuery, oldQuery) {
    // 古いクエリ文字列に `bar` が含まれ、新しいクエリ文字列に `foo` が含まれている場合のみ、
    // コンポーネントメソッドを実行します
    return newQuery.foo && oldQuery.bar
  }
}
```

Next

詳細は[データの取得](#)の章を参照してください。

key

テンプレート内の Vue コンポーネントで仮想 DOM のヒントとして使用できる `key` プロパティと同様に、このプロパティでは、（親コンポーネントではなく）ページ自体からキー値を定義することができます。

Nuxt のデフォルトでは、この値は `$route.path` になります。つまり、別のルートに移動すると、クリーンなページコンポーネントが作成されることになります。論理的には以下と同等です：

```
<router-view :key="$route.path" />
```

このプロパティは、`String` または `Function` で、ルートを第一引数として受け取ります。

ページを無視する

ページを無視したい場合は、ファイルの先頭に `-` を付けると `router.js` ファイルに含まれなくなります。

例えば、`pages/-about.vue` は無視されます。

Next

詳細は[ignore オプション](#)を参照してください。

設定

`dir.pages` オプションを設定すると、`pages/` ディレクトリの名前を別の名前に変更できます：

```
export default {
  dir: {
    // `pages` ディレクトリの名前を `routes` に変更します。
    pages: 'routes'
  }
}
```

Next

詳細は[dir オプション](#)を参照してください。

[Go to TOC](#)

alias プロパティ

Nuxt では、エイリアスを使って JavaScript や CSS でカスタムディレクトリにアクセスできます。

- 型: `Object`
- デフォルト:

```
{
  '~~': `<rootDir>`,
  '@@': `<rootDir>`,
  '~': `<srcDir>`,
  '@': `<srcDir>`,
  'assets': `<srcDir>/assets`, // (`dir.assets` を設定している場合を除く)
  'static': `<srcDir>/static`, // (`dir.static` を設定している場合を除く)
}
```

このオプションでは、(上記のディレクトリに加えて) プロジェクト内のディレクトリにエイリアスを定義できます。これらのエイリアスは、JavaScript や CSS で使用できます。

```
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

Webpack のコンテキスト内 (画像ソース、CSS - ただし JavaScript は 除く) では、(上記の例のように) エイリアスのプレフィックスとして `~` を付ける必要があります。

Info

TypeScript を使っていて、自分で定義したエイリアスを TypeScript ファイル内で使用したい場合は、`tsconfig.json` 内の `paths` オブジェクトにエイリアスを追加する必要があります。

[Go to TOC](#)

extendPlugins プロパティ

extendPlugins プロパティで Nuxt プラグインのカスタマイズができます。(options.plugins)。

- 型: `Function`
- デフォルト: `undefined`

プラグインを拡張したり、Nuxt で作成したプラグインの順序を変更したい場合があるでしょう。この関数は[プラグインオブジェクト](#)の配列を受け取り、プラグインオブジェクトの配列を返します。

プラグインの順序を変更する例：

```
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
  }
}
```

generate プロパティ

ユニバーサルなウェブアプリケーションや静的なウェブアプリケーションの生成について設定します。

- 型: `Object`

`nuxt.generate()` を呼び出した際、Nuxt は `generate` プロパティで定義された設定を使います。

```
export default {
  generate: {
    ...
  }
}
```

cache

v2.14.0 で導入されました

- 型: `Object` または `false`

このオプションは追跡したファイルが変更されていない場合にリビルドを避けるために `nuxt generate` の `static target` で使われます。

デフォルト:

```
{
  ignore: [
    '.nuxt', // buildDir
    'static', // dir.static
    'dist', // generate.dir
    'node_modules',
    '**/*',
    '*',
    'README.md'
  ]
}
```

設定ファイル変更時にリビルドを避けたい場合は、`cache.ignore` オプションに対象となる設定ファイルのリストを追加してください：

```
export default {
  generate: {
    cache: {
      ignore: ['renovate.json'] // このファイルに適用された変更を無視する
    }
  }
}
```

concurrency

- 型: `Number`
- デフォルト: `500`

ルート生成は並行で行われ、`generate.concurrency` は 1 スレッドで実行されるルートの数を指定します。

crawler

- 型: `boolean`
- デフォルト: `true`

Nuxt v2.13 以降、Nuxt には相対リンクをクロールし、クロールしたリンクに基づいて動的リンクを生成するクローラーがインストールされています。この機能を無効にしたい場合は `false` に設定してください。

```
export default {
  generate: {
    crawler: false
  }
}
```

dir

- 型: `String`
- デフォルト: `'dist'`

`nuxt generate` コマンドを使ってウェブアプリケーションがビルトされる時に作成されるディレクトリ名です。

devtools

- 型: `boolean`
- デフォルト: `false`

`vue-devtools` による検査を許可するかどうかを設定します。

もしすでに `nuxt.config.js` か何かで有効にしている場合はこのフラグに関係なく `devtools` が有効になります。

exclude

- 型: `Array`
 - 要素: `String` または `RegExp`

配列の文字列か正規表現を指定することができます、一致するルートの生成を防ぎます。`generate.fallback` が使われている場合はルートは引き続きアクセスすることができます。

この構造の例を用いると：

```
-| pages/
---| index.vue
---| admin/
-----| about.vue
-----| index.vue
```

デフォルトでは `nuxt generate` を実行するとそれぞれのルートに対してファイルが生成されます。

```
-| dist/
---| index.html
---| admin/
-----| about.html
-----| index.html
```

"ignore" で全てのルートに一致する正規表現を追加した場合はこれらのルート生成が防止されます。

```
export default {
  generate: {
    exclude: [
      '/^\/admin/' // /admin で始まるパス
    ]
  }
}
```

```
-| dist/
---| index.html
```

文字列で特定のルートを除外することもできます：

```
export default {
  generate: {
    exclude: ['/my-secret-page']
  }
}
```

fallback

- 型: `String` または `Boolean`
- デフォルト: `200.html`

```
export default {
  generate: {
    fallback: '404.html'
  }
}
```

フォールバックする HTML ファイルのパスです。エラーページとして設定する必要があります。この設定により、不明なルートも Nuxt を介してレンダリングされます。未設定または falsy を取り扱える値が設定されている場合、フォールバック HTML ファイルの名前は `200.html` になります。もし、`true` を設定すると、ファイル名は `404.html` になります。値として文字列を指定すると、その文字列が代わりに使用されます。

```
fallback: false;
```

静的に生成されたページを運用する場合、エラーページと `excludes` でカバーされるページに `404.html` を使うことをおすすめします（静的ページとして生成してほしくない場合）。

`fallback: true`

しかし、Nuxt では任意のページを設定できるため、`200.html` または `404.html` を使用したくない場合は代わりに文字列を追加して、そのページにリダイレクトができます。これはもちろん必須ではなく、`200.html` / `404.html` にリダイレクトするのがベストです。

`fallback: 'fallbackPage.html'`

注意: 複数のサービス（例えば Netlify）では、`404.html` を自動的に検出します。ウェブサーバーを独自に設定する場合は、ドキュメントを参照してエラーページの設定方法を確認してください（そして、エラーページを `404.html` ファイルに設定してください）

interval

- 型: `Number`
- デフォルト: `0`

2 つのレンダーの間でのミリセカンドのインターバルで、ウェブアプリケーションからの潜在的な API に対して溢れでないようにするためのものです。

minify

- 非推奨!**
- 代わりに `build.html.minify` を使ってください

routes

- 型: `Array`

Info

Nuxt v2.13 から `nuxt generate` を実行すると、リンクタグをクロールしてルートを生成するクローラがインストールされています。

リンクされていないページ（シークレットページなど）も生成したい場合は `generate.routes` プロパティを使うことができます。

Warning

Nuxt v2.12 以下 で `generate` コマンドを使うと、動的ルートは無視されます。

例：

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

ルート `/` のみが Nuxt によって生成されます。

Nuxt で動的なパラメータをもつルートを生成したい場合は `generate.routes` プロパティに動的なルートの配列を設定する必要があります。

`/users/:id` のルートを追加します：

```
export default {
  generate: {
    routes: ['/users/1', '/users/2', '/users/3']
  }
}
```

そして `nuxt generate` を実行した場合：

```
[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
```

すばらしい、ですが、**動的なパラメータ**がある場合はどうでしょうか？

1. `Promise` を返す `Function` を使う。
2. `callback(err, params)` と一緒に `Function` を使う。

Promise を返す関数

```
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
        })
      })
    }
  }
}
```

callback 関数

```
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)
    }
  }
}
```

payload による動的ルーティング生成の高速化

上記の例では、サーバーから `user.id` を使ってルーティングを生成しますが、残りのデータは破棄しています。通常はそのような場合は `/users/_id.vue` 内から再度データを取得する必要があります。再度取得することは可能ですが、その場合は `generate.interval` に 100 のような値を設定してサーバー呼び出しが殺到しないようにする必要があります。このような実装は生成時間の増加へつながるため、`user` オブジェクト自体を `_id.vue` コンテキストに渡すことが好ましいです。上記のコードを以下のように変更することで実現できます：

```
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
          }
        })
      })
    }
  }
}
```

これで `/users/_id.vue` から `payload` へアクセスできます：

```
async asyncData ({ params, error, payload }) {
  if (payload) return { user: payload }
  else return { user: await backend.fetchUser(params.id) }
}
```

subFolders

- 型: `Boolean`
- デフォルト: `true`

デフォルトでは `nuxt generate` を実行すると、Nuxt はルートごとにディレクトリを作成し `index.html` ファイルを提供します。

例：

```
- | dist/
---| index.html
---| about/
-----| index.html
---| products/
-----| item/
| index.html
```

`false` を設定した場合、HTML ファイルはルートパスに従って生成されます：

```
export default {
  generate: {
    subFolders: false
  }
}
```

```
- | dist/
---| index.html
---| about.html
---| products/
-----| item.html
```

globalName プロパティ

Nuxt ではメインの HTML テンプレートで使用されるグローバルな ID とメインとなる Vue インスタンス名、その他オプションをカスタマイズできます。

- 型: `String`
- デフォルト: `nuxt`

```
{
  globalName: 'myCustomName'
}
```

Warning

`globalName` は有効な JavaScript 識別子である必要があるので、変更することで Nuxt の名前がついた関数に依存する特定のプラグインが機能しなくなる可能性があります。表示されている `__nuxt` HTML ID を変更するだけの場合は `globals` プロパティを使ってください。

globals プロパティ

デフォルトで `globalName` に基づいた特定のグローバル名をカスタマイズします。

- 型: `Object`
- デフォルト:

```
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`
},
```

head プロパティ

Nuxt ではアプリケーションのデフォルトメタを全て `head` プロパティを使って `nuxt.config.js` 内で定義することができます

- 型: `Object` または `Function`

```
export default {
  head: {
    titleTemplate: '%s - Nuxt',
    meta: [
      { charset: 'utf-8' },
      { name: 'viewport', content: 'width=device-width, initial-scale=1' },

      // `hid` は一意の識別子として使用されます。`vmid` は動作しないので使わないでください
      { hid: 'description', name: 'description', content: 'Meta description' }
    ]
  }
}
```

`head` に指定できるオプションリストについては [vue-meta ドキュメント](#) を参照してください。

コンポーネントの関数として `head` を使用して `this` を介してコンポーネントデータにアクセスすることもできます（[詳細はこちら](#)）。

Info

子コンポーネントを使用した際にメタタグの重複を避けるために、meta 要素に `hid` キーで一意な識別子を設定してください（[詳細はこちら](#)）。

[Go to TOC](#)

hooks プロパティ

フックは Nuxt モジュールで一般的に使われる [Nuxt イベントリスナー](#)だけでなく `nuxt.config.js` でも利用できます。

- 型: `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')
      }
    }
  }
}
```

内部的には、フックはコロン（例えば `build:done`）を使った命名パターンに従います。設定を簡単にするため、上記の例のように独自のフックを `nuxt.config.js` に設定すると、それらを階層オブジェクトとして構造化できます。仕組みの詳細については [Nuxt の内部](#)を参照してください。

フックのリスト

- [Nuxt フック](#)
- [Renderer フック](#)
- [ModulesContainer フック](#)
- [Builder フック](#)
- [Generator フック](#)

例

root でない場合は `router.base` にリダイレクトさせる

ページを `/` のかわりに `/portal` として提供するとします。

これはおそらくエッジケースであり、`nuxt.config.js` の `router.base` のポイントは web サーバーがドメインルート以外の場所で Nuxt を提供するときのものです。

しかしローカル開発中に `localhost` にアクセスすると `router.base` が `/` でない場合は 404 が返されます。フックを設定することでこれを防げます。

リダイレクトはプロダクション用の Web サイトでは最適なユースケースではないかもしれませんのがフックを活用するのに役立ちます。

まずははじめに、`router.base` を変更できます。`nuxt.config.js` を更新してみましょう：

```
import hooks from './hooks'
export default {
  router: {
    base: '/portal'
  }
  hooks: hooks(this)
}
```

そしていくつかファイルを作成します。

1. フックモジュールである `hooks/index.js`

```
import render from './render'

export default nuxtConfig => ({
  render: render(nuxtConfig)
})
```

2. レンダーフックである `hooks/render.js`

```
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. ミドルウェア自体である `hooks/route-redirect-portal.js`

```
/**
 * /portalから / ヘリダイレクトするための Nuxt ミドルウェアフック（または nuxt.config.js
 * の router.base で設定したもの）
 *
 * connect 同じバージョンにしてください
 * {@link node_modules/connect/package.json}
 */
import parseurl from 'parseurl'

/**
 * 目的の Web アプリケーションコンテキストルートへのリダイレクト処理をするためのミドルウェアに接続します。
 *
 * Nuxt のドキュメントにはフックの使い方の説明が欠けていることに注意してください
 * これは補足説明として役立フルーターのサンプルです。
 *
 * インスピレーションのために素晴らしい実装を見てみましょう：
 * - 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の内部的なクライアントリクエストオブジェクト
[http_class_http_clientrequest]
 * @param {http.ServerResponse} res Node.jsの内部的なレスポンス
[http_class_http_serverresponse]
 * @param {Function} next ミドルウェアのコールバック
*/
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();
  }
}
```

これで、開発中の Web サービスで同僚が誤って / にアクセスしても Nuxt は自動的に /portal にリダイレクトするでしょう。

ignore プロパティ

Nuxt アプリケーションに対して無視するファイルを定義します

.nuxtignore

`.nuxtignore` ファイルを使うことで、ビルト時にプロジェクトのルートディレクトリ (`rootDir`) にある `layout`、`page`、`store` そして `middleware` ファイルを Nuxt に無視させることができます。`.nuxtignore` ファイルは `.gitignore` ファイルと `.eslintignore` ファイルと同じ仕様に従います。各行は無視するファイルを示す glob パターンです。

例：

```
# foo.vue レイアウトを無視する
layouts/foo.vue
# ファイル名の末尾が -ignore.vue となるレイアウトを無視する
layouts/*-ignore.vue

# bar.vue ページを無視する
pages/bar.vue
# ignore フォルダにあるページを無視する
pages/ignore/*.vue

# baz.js ストアを無視する
store/baz.js
# *.test.* にマッチするストアファイルを無視する
store/ignore/*.test.*

# foo フォルダにある foo/bar.js 以外のミドルウェアファイルを無視する
middleware/foo/*.js
!middleware/foo/bar.js
```

仕様に関する詳細は [gitignore のドキュメント](#) を参照してください

ignorePrefix プロパティ

- 型: `String`
- デフォルト: `'-'`

`pages/`、`layouts/`、`middleware/` や `store/` ディレクトリに含まれるファイルの中で、ファイル名が `ignorePrefi`
`x` プロパティで指定された接頭辞から始まる場合にはビルト時に無視されます。

デフォルトでは `-` で始まる `store/-foo.js` や `pages/-bar.vue` のようなファイルは全て無視されます。これによりルートやストア等に変換されることなく、呼び出し元と同じ場所にテスト、ユーティリティ、コンポーネント等のファイルを置くことができます。

ignore プロパティ

- 型: `Array`
- デフォルト: `['**/*.test.*']`

`ignorePrefix` よりもカスタマイズ: `ignore` で指定された `glob` パターンにマッチするすべてのファイルはビルド時に無視されます。

ignoreOptions

`nuxtignore` は内部で `node-ignore` を使っており、`ignoreOptions` は `node-ignore` の `options` として設定できます。

```
export default {
  ignoreOptions: {
    ignorecase: false
  }
}
```

loading プロパティ

Nuxt はルート間遷移時にローディングプログレスバーを表示するコンポーネントを用意しています。このコンポーネントをカスタマイズしたり、無効にしたりできます。また、独自のコンポーネントの作成もできます。

- 型: `Boolean` または `Object` または `String`

```
export default {
  mounted() {
    this.$nextTick(() => {
      this.$nuxt.$loading.start()

      setTimeout(() => this.$nuxt.$loading.finish(), 500)
    })
  }
}
```

プログレスバーを無効にする

- 型: `Boolean`

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

プログレスバーをカスタマイズする

- 型: `Object`

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

プログレスバーをカスタマイズするためのプロパティのリスト。 | キー | 型 | デフォルト | 説明 | | | | `color` | `String` | `'black'` | プログレスバーの CSS カラー | | `failedColor` | `String` | `'red'` | ルートをレンダリング中にエラー（例えば `data` や `fetch` がエラーを返したとき）が発生した場合のプログレスバーの CSS カラーです | | `height` | `String` | `'2px'` | プログレスバーの高さ（プログレスバーの `style` プロパティで使われる） | | `throttle` | `Number` | `200` | ミリ秒単位で指定された時間待ったのちにプログレスバーを表示します。 プログレスバーの点滅を防ぐために利用します | | `duration` | `Number` | `5000` | プログレスバーを表示する時間の最大値をミリ秒で指定します。 Nuxt は各ルートが 5 秒以内にレンダリングされると想定しています | | `continuous` | `Boolean` | `false` | ローディングが `duration` で指定した時間より長くかかる場合は、プログレスバーのアニメーションを表示したままにします | | `css` | `Boolean` | `true` | デフォルトのプログレスバーのスタイルを削除（そして、独自に追加）する場合には `false` に設定します | | `rtl` | `Boolean` | `false` | プログレスバーの向きを右から左に設定します |

独自のローディングコンポーネントを使う

- 型: `String`

Your component has to expose some of these methods:

メソッド	必須かどうか	説明			
`start()`	必須	ルートが変更されたときに呼び出されます。このときに独自コンポーネントの表示が開始されます			
`finish()`	必須	ルートがロード（及びデータ取得）されたときに呼び出されます。このときに独自コンポーネントが表示が終了します			
`fail(error)`	任意	ルートがロードできなかったときに呼び出されます（例えばデータの取得に失敗したなど）			
`increase(num)`	任意	ルートのコンポーネントがロードされている間に呼び出されます。`num` は 100 未満の整数です			

```
<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'
}
```

loading indicator プロパティ

ページ読み込み中に素敵なローディングインジケータを表示します！

サーバーサイドレンダリングでない (`ssr` オプションが `false`) 場合、最初のページ読み込み時にはサーバーサイドからのコンテンツがありません。そのためページ読み込み中に空白のページのかわりにスピナーを表示することができます。

このプロパティには `string`、`false`、`object` の 3 つの異なる型を指定できます。文字列が指定された場合はオブジェクトスタイルに変換されます。

デフォルト値：

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

内蔵されているインジケータ

これらのインジケータは素晴らしい [SpinKit](#) プロジェクトからインポートされています。SpinKit のデモページを使ってスピナーをプレビューできます。

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

内蔵インジケータは `color` と `background` オプションをサポートしています。

カスタムインジケータ

独自の特別なインジケータが必要な場合は文字列または Name キーをインジケータのソースコードの HTML テンプレートへのパスにすることもできます。すべてのオプションもテンプレートに渡されます。

もしベースが必要な場合は[ソースコード](#)も利用できます！

[Go to TOC](#)

mode プロパティ

デフォルト nuxt モードに変更します

- 型: `string`
 - デフォルト: `universal`
 - 設定できる値:
 - `'spa'`: サーバーサイドレンダリングを行いません（クライアントでの遷移のみ）
 - `'universal'`: アイソモーフィックなアプリケーション（サーバーサイドレンダリングに加え、クライアントでの遷移を行う）になります

`nuxt.config.js` でこのオプション使うとデフォルトの `nuxt` の `mode` を変更できます。

Warning

非推奨: `mode: spa` のかわりに `ssr: false` を使ってください

Next

`ssr` オプションについての詳細は [ssr プロパティ](#) を参照してください。

Next

`mode` オプションについての詳細は [レンダリングモードのドキュメント](#) を参照してください。

[Go to TOC](#)

modern プロパティ

ビルドとモダンなバンドルの提供

この機能は [vue-cli の modern mode](#) にインスパイアされています

- 型: `String` または `Boolean`
 - デフォルト: `false`
 - 設定できる値:
 - `'client'` : モダンバンドル `<script type="module">` とレガシーバンドル `<script nomodule>` スクリプトの両方を提供し、モダンバンドル用の `<link rel="modulepreload">` も提供します。module タイプを理解するすべてのブラウザではモダンバンドルをロードしますが、古いブラウザではレガシー（トランスペイルされた）バンドルにフォールバックします。
 - `'server'` または `true` : Node.js サーバーは、ユーザーエージェントに基づいてブラウザのバージョンをチェックし、対応するモダンバンドルまたはレガシーバンドルを提供します。
 - `false` : モダンビルトを無効にします

2 つのバージョンのバンドルについて:

1. モダンバンドル: ES modules をサポートするモダンなブラウザを対象にしています。
2. レガシーバンドル: babel config (デフォルトでは IE9 互換) をベースにした古いブラウザを対象にしています。

情報:

- モダンバンドルでビルド/スタートするにはコマンドオプション `--modern | -m=[mode]` を使ってください:

```
{
  "scripts": {
    "build:modern": "nuxt build --modern=server",
    "start:modern": "nuxt start --modern=server"
  }
}
```

nuxt generate に関する注意: `modern` プロパティは `nuxt generate` コマンドでも動作しますが、この場合は `client` オプションだけが優先され `nuxt generate --modern` コマンドを値なしで実行した際に自動的に選択されます。

- `modern` が指定されていない場合、Nuxt は `nuxt start` の `modern` ビルドを自動的に検出します。自動検出モードは次のとおりです:

ssr
<code>true</code>
<code>false</code>

- `nuxt generate` のモダンモードは `client` のみになります。
- `render.crossorigin` を使って `<link>` と `<script>` タグに `crossorigin` 属性を設定します。

モダンビルトについての詳細は [Phillip Walton さんの素晴らしい投稿](#)を参照してください。

build プロパティ

Nuxt では、webpack の設定をカスタマイズして web アプリケーションを思いのままに構築することができます。

analyze

Nuxt では [webpack-bundle-analyzer](#) を使ってバンドルファイルと最適化の仕方を視覚化できます。

- 型: `Boolean` または `Object`
- デフォルト: `false`

オブジェクトの場合、利用できるプロパティについて[こちら](#)を参照してください。

```
export default {
  build: {
    analyze: true,
    // または
    analyze: {
      analyzerMode: 'static'
    }
  }
}
```

Info

情報: `yarn nuxt build --analyze` または `yarn nuxt build -a` コマンドを使って、アプリケーションをビルドし、バンドルアナライザを <http://localhost:8888> で起動できます。 `yarn` を使っていない場合は `npx` を使って実行できます。

corejs

Nuxt@2.14 以降、Nuxt は `core-js` の現在のバージョンを自動的に検出します。また、使用するバージョンを指定することもできます。

- 型: `number | string` (有効な値は `'auto'`、`2` そして `3` です)
- デフォルト: `'auto'`

babel

JavaScript や Vue ファイル用に Babel の設定をカスタマイズします。デフォルトでは `.babelrc` は無視されます。

- 型: `Object`
- `babel-loader` オプションと `babel` オプションを参照してください
- デフォルト:

```
{
  babelrc: false,
  cacheDirectory: undefined,
  presets: ['@nuxt/babel-preset-app']
}
```

`@nuxt/babel-preset-app` のデフォルトのターゲットは `client` ビルドでは `ie: '9'`、`server` ビルドでは `node: 'current'` です。

presets

- 型: `Function`
- 引数:
 - `Object` : `isServer: true`
 - `Array` :
 - プリセット名 `@nuxt/babel-preset-app`
 - `@nuxt/babel-preset-app` のオプション

注意: `build.babel.presets` で設定されたプリセットはクライアントとサーバー両方のビルドに適用されます。ターゲットは（クライアント/サーバー）それに応じて Nuxt によって設定されます。クライアントビルドとサーバービルドで異なるプリセットの設定をしたい場合は、関数として `presets` を使用してください：

以下のカスタマイズの代わりにデフォルトのプリセットを使用することを強くおすすめします

```
export default {
  build: {
    babel: {
      presets({ isServer }, [ preset, options ]) {
        // change options directly
        options.targets = isServer ? ... : ...
        options.corejs = ...
        // return nothing
      }
    }
  }
}
```

またはプリセットのリスト全体を返すことによってデフォルトの値を上書きします：

```
export default {
  build: {
    babel: {
      presets({ isServer }, [preset, options]) {
        return [
          [
            preset,

```

```
{
  targets: isServer ? ... : ...,
  ...options
}
],
[
  // 他のプリセット
]
]
}
}
```

cache

- 型: `Boolean`
- デフォルト: `false`
- △ 実験的機能

`terser-webpack-plugin` と `cache-loader` を使ってキャッシュを有効化します

cssSourceMap

- 型: `boolean`
- デフォルト: 開発モードでは `true`、プロダクションモードでは `false`

CSS ソースマップのサポートを有効にします

devMiddleware

- 型: `Object`

利用できるオプションは [webpack-dev-middleware](#) を参照してください。

devtools

- 型: `boolean`
- デフォルト: `false`

`vue-devtools` の検査を許可するかどうかを設定します。

既に `nuxt.config.js` などで有効化している場合は、このフラグに関係なく `devtools` が有効になります。

extend

クライアント及びサーバーのバンドルについて webpack の設定を手動で拡張します。

- 型: `Function`

`extend` メソッドは一度はサーバーのバンドルのため、一度はクライアントのバンドルのため、計二度呼び出されます。メソッドの引数は次のとおりです：

1. webpack 設定オブジェクト
2. 次のキーを持つオブジェクト (`loaders` を除きすべて真偽値) : `isDev`、`isClient`、`isServer`、`loaders`

Warning

警告: 提供される `isClient` と `isServer` は `context` で利用可能なキーとは別物です。これらのキーは非推奨ではあります。ここでは、`process.client` および `process.server` は `undefined` となるため使用しないでください

```
export default {
  build: {
    extend(config, { isClient }) {
      // クライアントのバンドルの webpack 設定のみを拡張する
      if (isClient) {
        config.devtool = 'source-map'
      }
    }
  }
}
```

デフォルトの webpack の設定についてもう少し見たい場合は [webpack ディレクトリ](#)を参照してください。

extend での loaders

`loaders` は `build.loaders` と同じオブジェクト構造を持っているため、`extend` 内部の `loaders` のオプションを変えることができます。

```
export default {
  build: {
    extend(config, { isClient, loaders: { vue } }) {
      // クライアントのバンドルの webpack 設定のみを拡張する
      if (isClient) {
        vue.transformAssetUrls.video = ['src', 'poster']
      }
    }
  }
}
```

extractCSS

Vue サーバーサイドレンダリングガイドラインを使って共通の CSS を抽出できるようにします。

- 型: `Boolean` または `Object`
- デフォルト: `false`

内部で `extract-css-chunks-webpack-plugin` が使われ、全ての CSS は別々のファイルに、通常はコンポーネントごとに 1 つ抽出されます。これは CSS と JavaScript を別々にキャッシュすることを可能にし、多くのグローバルまたは共通 CSS が存在する場合には試してみる価値があります。

例 (`nuxt.config.js`) :

```
export default {
  build: {
    extractCSS: true,
    // または
    extractCSS: {
      ignoreOrder: true
    }
  }
}
```

Info

警告: Vue 2.5.18 以前では、このオプションを使用したときに重要な CSS のインポートが削除されるバグがありました。

全ての CSS を 1 つのファイルに抽出したい場合があります。このための回避策があります：

Warning

全てのファイルを 1 つのファイルに抽出することはおすすめしません。複数の CSS ファイルに抽出する方が、キャッシングとプリロードの分離には適しています。また、必要なリソースのみをダウンロードして解決することによって、ページのパフォーマンスを向上させることもできます。

```
export default {
  build: {
    extractCSS: true,
    optimization: {
      splitChunks: {
        cacheGroups: {
          styles: {
            name: 'styles',
            test: /\.(\css|vue)$/,
            chunks: 'all',
            enforce: true
          }
        }
      }
    }
  }
}
```

filenames

バンドルのファイル名をカスタマイズします。

- 型: `Object`

- デフォルト:

```
{
  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]`
}
```

この例ではチャンク名を数値の `id` に変更します：

```
export default {
  build: {
    filenames: {
      chunk: ({ isDev }) => (isDev ? `[$name].js` : `[$id].[contenthash].js`)
    }
  }
}
```

`manifest` の使い方をより理解するためにこの [webpack ドキュメント](#) を参照してください。

Warning

ハッシュ化されていないベースのファイル名を本番環境で使用する場合、ほとんどのブラウザはアセットをキャッシュし、最初のロード時には変更を検出しないので注意してください。

friendlyErrors

- 型: `Boolean`
- デフォルト: `true` (上書きが有効)

[FriendlyErrorsWebpackPlugin](#) を使って上書きを有効にするか無効にするかを設定します。

hardSource

- 型: `Boolean`
- デフォルト: `false`

- △ 実験的機能

キャッシュを改善するために [HardSourceWebpackPlugin](#) を有効にします。

hotMiddleware

- 型: `Object`

利用できるオプションは [webpack-hot-middleware](#) を参照してください。

html.minify

- 型: `Object`
- デフォルト:

```
{
  collapseBooleanAttributes: true,
  decodeEntities: true,
  minifyCSS: true,
  minifyJS: true,
  processConditionalComments: true,
  removeEmptyAttributes: true,
  removeRedundantAttributes: true,
  trimCustomFragments: true,
  useShortDoctype: true
}
```

注意: `html.minify` に変更を加えても、それらはデフォルトとマージされません！

ビルドプロセス中に作成された HTML ファイルを最小化するために使用される [html-minifier](#) プラグインの設定（すべてのモードに適用されます）。

indicator

開発モードにおいて hot module replacement のビルドインジケータを表示する（v2.8.0 以上から利用可能）

- 型: `Boolean`
- デフォルト: `true`



nuxt-build-indicator

loaders

Nuxt に統合された webpack loaders のカスタマイズオプション

- 型: `Object`
- デフォルト:

```
{  
  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: {}  
}
```

注意: `nuxt.config.js` の設定で指定するだけでなく、`build.extend` でも設定を変更することができます

loaders.file

詳細は [file-loader オプション](#)を参照してください。

loaders.fontUrl と loaders.imgUrl

詳細は [url-loader オプション](#)を参照してください。

loaders.pugPlain

詳細は [pug-plain-loader](#) または [Pug compiler オプション](#)を参照してください。

loaders.vue

詳細は [vue-loader オプション](#) を参照してください。

loaders.css と loaders.cssModules

詳細は [css-loader オプション](#)を参照してください。注意: cssModules は [CSS Modules](#)を使うための loader オプションです。

loaders.less

Less specific オプションは `loaders.less` を介して `less-loader` に渡すことができます。dash-case で利用可能な全てのオプションについては [Less のドキュメント](#)を参照してください。

loaders.sass と loaders.scss

利用可能な Sass のオプションについては [Sass ドキュメント](#)を参照してください。注意: `loaders.sass` は [Sass Indented Syntax](#)用です。

loaders.vueStyle

詳細は [vue-style-loader オプション](#)を参照してください。

optimization

- 型: `Object`
- デフォルト:

```
{  
  minimize: true,  
  minimizer: [  
    // ...  
  ]  
}
```

```
// terser-webpack-plugin
// optimize-css-assets-webpack-plugin
],
splitChunks: {
  chunks: 'all',
  automaticNameDelimiter: '.',
  name: undefined,
  cacheGroups: {}
}
}
```

`dev` または `analyze` モードでは `splitChunks.name` のデフォルト値は `true` です。

カスタマイズされたプラグインの配列に `minimizer` を設定するか、`minimize` を `false` にすることで全ての `minimizer` を無効にできます。（`minimize` はデフォルトで開発用に無効になっています）

[Webpack の最適化](#)を参照してください。

optimizeCSS

- 型: `Object` または `Boolean`
- デフォルト:
 - `false`
 - `extractCSS` が有効になるときは `{}`

OptimizeCSSAssets プラグインのオプションです。

[NMFR/optimize-css-assets-webpack-plugin](#) を参照してください。

parallel

- 型: `Boolean`
- デフォルト: `false`
- △ 実験的機能

webpack のビルドで `thread-loader` を有効にする

plugins

webpack プラグインを追加する

- 型: `Array`
- デフォルト: `[]`

```
import webpack from 'webpack'
import { version } from './package.json'
export default {
  build: {
    plugins: [
      new webpack.DefinePlugin({
        'process.VERSION': version
      })
    ]
  }
}
```

postcss

[PostCSS Loader](#) プラグインをカスタマイズします

- 型: `Array` (レガシーな場合はデフォルトで上書きします)、`Object` (推奨)、`Function` または `Boolean`

注意: Nuxt は [PostCSS Preset Env](#)を適用しました。デフォルトでは、[Stage 2 機能](#) と [Autoprefixer](#)が有効になっていますが、`build.postcss.preset` を使って設定できます。

- デフォルト:

```
{
  plugins: {
    'postcss-import': {},
    'postcss-url': {},
    'postcss-preset-env': this.preset,
    'cssnano': { preset: 'default' } // 開発モードでは無効化されています
  },
  order: 'presetEnvAndCssnanoLast',
  preset: {
    stage: 2
  }
}
```

カスタムプラグイン設定は、デフォルトのプラグイン設定にマージされます (`Object` のかわりに `Array` を使っている場合を除きます)。

```
export default {
  build: {
    postcss: {
      plugins: {
        // `postcss-url` の無効化
        'postcss-url': false,
        // プラグインの追加
        'postcss-nested': {},
        'postcss-responsive-type': {},
        'postcss-hexrgba': {}
      },
      preset: {
        autoprefixer: {
          grid: true
        }
      }
    }
  }
}
```

```

        }
    }
}
```

postcss の設定が `Object` の場合、プラグインの順番の定義に `order` が利用できます：

- 型: `Array` (順序付けされたプラグイン名)、`String` (順序プリセット名)、`Function`
- デフォルト: `cssnanoLast` (最後に `cssnano` を配置する)

```

export default {
  build: {
    postcss: {
      // プリセット名
      order: 'cssnanoLast',
      // 順序付けされたプラグイン名
      order: ['postcss-import', 'postcss-preset-env', 'cssnano']
      // プラグインの順番を算出するための関数
      order: (names, presets) => presets.cssnanoLast(names)
    }
  }
}
```

postcss plugins と @nuxtjs/tailwindcss

@nuxtjs/tailwindcss の設定で postcss プラグイン（例えば `postcss-pxtorem`）を適用したい場合、順番を変更して最初に tailwindcss を読み込む必要があります。

この設定は `nuxt-purgecss` には影響しません。

```

import { join } from 'path'

export default {
  // ...
  build: {
    postcss: {
      plugins: {
        tailwindcss: join(__dirname, 'tailwind.config.js'),
        'postcss-pxtorem': {
          propList: ['*', '!border*']
        }
      }
    }
  }
}
```

profile

- 型: `Boolean`
- デフォルト: コマンドライン引数 `--profile` で有効にします

[WebpackBar](#) の profiler で有効にします

publicPath

Nuxt ではパフォーマンスの最大化のため dist ディレクトリ内のファイルを CDN にアップロードすることも可能です。そのためにはまず publicPath に CDN を指定します。

- 型: `String`
- デフォルト: `'/_nuxt/'`

```
export default {
  build: {
    publicPath: 'https://cdn.nuxtjs.org'
  }
}
```

設定後、`nuxt build` を実行する際に `.nuxt/dist/client` ディレクトリの内容を CDN にアップロードしてください。

Nuxt v2.15 以降では、実行時にこのプロパティの値を変更すると、既にビルドされているアプリケーションの設定が上書きされます。

quiet

ビルド出力ログの大半を抑制します

- 型: `Boolean`
- デフォルト: `std-env`によって `CI` または `test` 環境で検出された際に有効になります

splitChunks

- 型: `Object`
- デフォルト:

```
export default {
  build: {
    splitChunks: {
      layouts: false,
      pages: true,
      commons: true
    }
  }
}
```

`layout`、`pages` や `commons` で分割したコードの場合（共通ライブラリ:`vue|vue-loader|vue-router|vuex...`）

ssr

SSR レンダラー用の webpack バンドルを作成します。

- 型: `Boolean`
- デフォルト: ユニバーサルモードでのデフォルト値は `true`、spa モードでのデフォルト値は `false`

このオプションは、指定されていない場合は `mode` 値に基づいて自動的に設定されます。

standalone

サーバービルドにおいて依存関係をインラインでバンドル（アドバンスド）

- 型: `Boolean`
- デフォルト: `false`

このモードは通常サーバービルドで外部として保存される `node_modules` をバンドルします（[詳細](#)）

Warning

ランタイムの依存関係（モジュール、`nuxt.config`、サーバーミドルウェア、静的ディレクトリ）はバンドルされていません。この機能は、`server-bundle` での `webpack-externals` の使用のみを無効にします。

Info

コマンド `yarn nuxt build --standalone` を使用して、コマンドラインでこのモードを有効にすることができます。（`yarn` を使用していない場合は、`npx` を使用してコマンドを実行できます）

styleResources

- 型: `Object`
- デフォルト: `{}`

Warning

このプロパティは非推奨です。パフォーマンスおよび開発体験の向上のために、代わりに `style-resources-module` を使ってください。

毎回インポートせずに変数やミックスインをページに挿入する必要がある場合に便利です。

Nuxt はこの動作を実現するために <https://github.com/yenshih/style-resources-loader> を使用しています。

特定のプリプロセッサ（`less`、`sass`、`scss` または `stylus`）に含めるパターン/パスを指定する必要があります：

ここではパスのエイリアス (~ や @) を使用することができないため、相対パスまたは絶対パスを使用する必要があります。

```
{
  build: {
    styleResources: {
      scss: './assets/variables.scss',
      less: './assets/*.less',
      // sass: ....,
      // scss: ...
      options: {
        // https://github.com/yenshih/style-resources-loader#options の
        // `patterns` プロパティ以外を参照してください
      }
    }
  }
}
```

templates

Nuxt では設定に基づいてレンダリングされる独自のテンプレートを提供できます。この機能はモジュールを使う場合にとりわけ便利です。

- 型: `Array`

```
export default {
  build: {
    templates: [
      {
        src: '~/modules/support/plugin.js', // `src` は絶対パスもしくは相対パスで指定してください
        dst: 'support.js', // `dst` は `~.nuxt` ディレクトリからみた相対パスです
        options: {
          // Options は `options` キーとしてテンプレートに提供されます
          live_chat: false
        }
      }
    ]
  }
}
```

テンプレートは `lodash.template` を使ってレンダリングされます。[こちら](#)でより詳細な使い方を知ることができます。

terser

- 型: `Object` または `Boolean`
- デフォルト:

```
{
  parallel: true,
  cache: false,
  sourceMap: false,
  extractComments: {
```

```

        filename: 'LICENSES'
    },
    terserOptions: {
        output: {
            comments: /^ !*** !@preserve!@license!@cc_on/
        }
    }
}

```

Terser プラグインのオプションです。`false` を設定するとこのプラグインは無効になります。

`sourceMap` を有効にすると、webpack の `config.devtool` が `source-map` に設定されている場合、各出力ファイルの最後に `//# sourceMappingURL` というリンクコメントが残ります。

[webpack-contrib/terser-webpack-plugin](#)を参照してください

transpile

- 型: `Array<String | RegExp | Function>`
- デフォルト: `[]`

特定の依存関係を Babel でトランスパイルしたい場合、`build.transpile` を追加することができます。`transpile` の項目はパッケージ名や依存関係のファイル名にマッチする文字列または正規表現オブジェクトになります。

`v2.9.0` からは条件付きでトランスパイルするために関数を使うこともできます。関数はオブジェクト (`{ isDev, isServer, isClient, isModern, isLegacy }`) を受け取ります：

```
{
  build: {
    transpile: [( { isLegacy } ) => isLegacy && 'ky' ]
  }
}
```

この例では、Nuxt が`modern mode`でない場合、`ky` は Babel によってトランスパイルされます。

vueLoader

注意: この設定は Nuxt 2.0 から削除されました。代わりに `build.loaders.vue` を使ってください。

- 型: `Object`
- デフォルト:

```
{
  productionMode: !this.options.dev,
  transformAssetUrls: {
    video: 'src',
    source: 'src',
    object: 'src',
  }
}
```

```
    embed: 'src'  
  }  
}
```

Vue Loader オプションを指定します。

watch

監視や変更後に再生成を行うカスタムファイルを提供することができます。この機能は[モジュール](#)を使用する場合にとりわけ便利です。

- 型: `Array<String>`

```
export default {  
  build: {  
    watch: ['~/.nuxt/support.js']  
  }  
}
```

followSymlinks

デフォルトでは、ビルドプロセスはシンボリックリンク内のファイルをスキャンしません。この boolean 値はスキヤンするかどうかを含むため、`followSymlinks` を `true` に設定すると例えば "pages" フォルダなどでフォルダ内のシンボリックリンクを使うことができます。

- 型: `Boolean`

```
export default {  
  build: {  
    followSymlinks: true  
  }  
}
```

[Go to TOC](#)

modules プロパティ

モジュールは Nuxt のコア機能を拡張し、無限のインテグレーションを加える Nuxt の拡張機能です。詳細は[こちら](#)

- 型: `Array`

例 (`nuxt.config.js`) :

```
export default {
  modules: [
    // パッケージ名を使う
    '@nuxtjs/axios',

    // プロジェクトの srcDir からの相対パス
    '~/modules/awesome.js',

    // オプションを渡す
    ['@nuxtjs/google-analytics', { ua: 'X1234567' }],

    // インラインで定義
    function () {}
  ]
}
```

モジュール開発者は通常、追加で必要なステップや使用方法の詳細を提供します。

Nuxt は `node` の読み込みパス（`node_modules` の中）を用いてモジュール配列内の各要素を解決しようとします。`~` エイリアスが使用されている場合、プロジェクトの `srcDir` から解決されます。モジュールは順番に実行されるため、順序が重要です。

注意: モジュールによって注入されたプラグインは、プラグインリストの `先頭` に追加されます。オプションとして：

- 自分のプラグインをプラグインリストの最後に手動で追加します（`this.nuxt.options.plugins.push(...)`）
- 他のモジュールに依存している場合、モジュールの順序を逆にします

モジュールは `nuxt` のビルト/ランタイムを強化する機能をエクスポートし、必要に応じてそれらのジョブが完了するまで `promise` を返却する必要があります。実行時に必要とされるので、最新の ES6 機能に依存する場合には、すでにトランスペイルされている必要があることに注意して下さい。

モジュールの仕組みや独自のモジュール開発に興味がある場合、より詳細な情報については[モジュールガイド](#)を参照して下さい。また、公式の[モジュール](#)セクションでは Nuxt Community によって作られた数多くの本番用モジュールの一覧を提供しています。

buildModules

Info

この機能は Nuxt v2.9 以降で使うことができます

一部のモジュールは開発時およびビルド時にのみ必要になります。`buildModules` を使うことで本番環境の起動を速くし、本番環境にデプロイされる `node_modules` のサイズを大幅に減らすことができます。それぞれのモジュールのドキュメントを参照して、`modules` と `buildModules` どちらを使うのが推奨されているかを確認してください。

使用方法の違いは以下のとおりです：

- `nuxt.config.js` に `modules` を追加するかわりに `buildModules` を使います
- `package.json` に `dependencies` を追加するかわりに `devDependencies` を使います（`yarn add --dev` または `npm install --save-dev`）

[Go to TOC](#)

modulesDir プロパティ

Nuxt アプリケーションに対してモジュールディレクトリを定義します

- 型: `Array`
- ディレクトリ: `['node_modules']`

モジュールディレクトリの設定でパス解決のために使用します。例えば Webpack の `resolveLoading`、`nodeExternals` や `postcss` です。設定パスは `options.rootDir` (デフォルト: `process.cwd()`) からの相対パスになります。

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

プロジェクトが Yarn ワークスペーススタイルのモノリポジトリで構成されている場合はこのフィールドが必要になるかもしれません。

plugins プロパティ

Nuxt の `plugins` オプションで Vue.js プラグインを使用します。

注意: Nuxt 2.4 以降、プラグインのタイプを指定するために `plugins` のオプションとして `mode` が導入されました。指定可能な値は `client` または `server` です。`ssr: false` は `mode: 'client'` の改良に伴い次のメジャーリリースで非推奨になります。

- 型: `Array`
 - 要素: `String` または `Object`

要素がオブジェクトの場合、プロパティは次のとおりです：

- `src: String` (ファイルパス)
- `mode: String` (`client` または `server`) もし定義されている場合はファイルはそれぞれの (クライアントまたはサーバー) 側にのみ含まれます。

注意: 古いバージョン

- 型: `Array`
 - 要素: `String` または `Object`

要素がオブジェクトの場合、プロパティは次のとおりです：

- `src: String` (ファイルパス)
- `ssr: Boolean` (デフォルトは `true`) もし `false` の場合はクライアントサイドでのみファイルがインクルードされます。

`plugins` プロパティを使うと Vue.js プラグインをメインアプリケーションに簡単に追加できます。

```
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' // Ant Design ドキュメンテーションごと
Vue.use(Antd)
```

css は [Ant Design ドキュメント](#)に従ってインポートされていることに注意してください。

`plugins` プロパティに定義されたパスはすべてメインアプリケーションが初期化される前にインポートされます。

render プロパティ

Nuxt はページレンダリング用のランタイムオプションをカスタマイズできます。

bundleRenderer

- 型: `Object`

このオプションを使って Vue SSR のバンドルレンダラーをカスタマイズします。`ssr: false` が設定されている場合このオプションはスキップされます。

```
export default {
  render: {
    bundleRenderer: {
      directives: {
        custom1(el, dir) {
          // 何かの処理 ...
        }
      }
    }
  }
}
```

利用可能なオプションについての詳細は [Vue SSR API リファレンス](#) を参照してください。Nuxt はすでに最高の SSR のデフォルト設定を提供しており誤った設定は SSR の問題を引き起こすので、このオプションは使用しないことをおすすめします。

etag

- 型: `Object`
 - デフォルト: `{ weak: true }`

ページの `etag` を無効にするには `etag: false` を設定してください。

利用可能なオプションは [etag](#) のドキュメントを参照してください。

`etag.hash` を指定することで、独自のハッシュ関数を使用することができます：

```
import { murmurHash128 } from 'murmurhash-native'

export default {
  render: {
    etag: {
      hash: html => murmurHash128(html)
    }
  }
}
```

この場合、大きい HTML の body サイズに対して高速な `murmurhash-native` を使います。独自のハッシュ関数を指定する場合、`weak` オプションは無視されることに注意してください。

compressor

- 型: `Object`
 - デフォルト: `{ threshold: 0 }`

オブジェクトを提供する場合、`compression` ミドルウェアがそれぞれのオプションと共に使われます。

独自の圧縮ミドルウェアを使用したい場合は、直接参照することができます（例: `otherComp({ myOptions: 'example' })`）。

圧縮を無効にするには `compressor: false` を設定します。

fallback

- 型: `Object`
 - デフォルト: `{ dist: {}, static: { skipUnknown: true } }`
 - `dist` キーは `publicPath` に一致するルート用です（例: `/_nuxt/*`）
 - `static` キーは `/*` に一致するルートに一致するルート用です

`dist` と `static` の値は `serve-placeholder` ミドルウェアに転送されます。

もしこれらのうち 1 つか両方を無効にする場合は、偽をとり得る値を渡すことができます。

ルーティングに `.js` 拡張子を許可する例（例: `/repos/nuxt.js`）：

```
export default {
  render: {
    fallback: {
      static: {
        // これらの拡張子に対し 404 の送信を避けます
        handlers: {
          '.js': false
        }
      }
    }
  }
}
```

http2

- 型: `Object`
 - デフォルト: `{ push: false, pushAssets: null }`

HTTP2 プッシュヘッダーを有効にします。

`pushAssets` 関数を使ってプッシュするリンクを制御できます。

例：

```
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}`)
```

独自のアセットを配列に追加することもできます。`req` と `res` を使ってリクエストヘッダーを元にプッシュするリンクを決めることができます。例えばアプリケーションのバージョンでクッキーを使います。

アセットは `,` で結合され、1 つの `Link` ヘッダーとして渡されます。

asyncScripts

- 型: `Boolean`
 - デフォルト: `false`

Nuxt バンドル用に `<script>` タグに `async` 属性を追加し、パースと並行してフェッチできるようにします（v2.14.8 以上 から利用可能）。[詳細はこちら](#)を参照してください。

injectScripts

- 型: `Boolean`
 - デフォルト: `true`

Nuxt のバンドルに `<script>` を追加します（v2.8.0 以降 で利用可能）。JS を除く純粋な HTML を表示する場合は `false` に設定してください。

resourceHints

- 型: `Boolean`
 - デフォルト: `true`

初期ページの読み込み時間をより早くするために `prefetch` と `preload` リンクを追加します。

ページとルートが多い場合にのみ、このオプションを無効にすることをおすすめします。

ssr

- 型: `Boolean`
 - デフォルト: `true`
 - クライアントサイドのレンダリングの場合のみ `false`

SSR レンダリングを有効にします。

このオプションが指定されていない場合、グローバルな `ssr` の値に基づいて自動的に設定されます。これは（例えば Docker で）イメージビルド後にランタイムで SSR を動的に有効/無効にするのに便利です。

crossorigin

- 型: `String`
 - デフォルト: `undefined`

生成された HTML の `<link rel="stylesheet">` と `<script>` タグに `crossorigin` 属性を設定します。

詳細: [CORS 属性](#)

ssrLog

- 型: `Boolean | String`
 - デフォルト: 開発モードでは `true`、本番では `false`

デバックしやすいように、サーバーサイドのログをブラウザに転送します（開発モードのみ利用可能）。

ログを折りたたむには `'collapsed'` を設定します。

static

- 型: `Object`
 - デフォルト: `{}`

`static/` ディレクトリの振る舞いを設定します

利用可能なオプションは [serve-static](#) のドキュメントを参照してください。

それらに加えて、デフォルト `true` の `prefix` オプションを導入しました。静的なアセットに `router base` を追加します。

例s:

- アセット: `favicon.ico`
- Router base: `/t`
- `prefix: true` の場合 () デフォルト: `/t/favicon.ico`
- `prefix: false` の場合: `/favicon.ico`

警告:

一部の URL の書き換えではプレフィックスが守られないかもしれません。

dist

- 型: `Object`
 - デフォルト: `{ maxAge: '1y', index: false }`

配布ファイルのデプロイに使用されるオプションです。本番でのみ適用されます。

利用可能なオプションは [serve-static](#) のドキュメントを参照してください。

csp

- 型: `Boolean` または `Object`
 - デフォルト: `false`

これを使用して、外部リソースを読み込むように Content-Security-Policy を構成します

前提条件:

これらの CSP 設定は、Nuxt に `target: 'server'` を設定して SSR アプリケーションを提供する場合にのみ有効です。
`csp.policy` で定義されたポリシーはレスポンスの `Content-Security-Policy` HTTP ヘッダーに追加されます。

設定の更新:

これらの設定は Nuxt サーバーが `nuxt.config.js` から直接読み込みます。つまり、これらの設定の変更にはサーバー再起動時に有効になります。CSP 設定を更新するためにアプリケーションをリビルドする必要はありません。

HTML メタタグ:

`<meta http-equiv="Content-Security-Policy"/>` を `<head>` に追加するには、`csp.addMeta` を `true` に設定する必要があります。この機能は `csp.policies` の設定とは無関係であることに注意してください:

- `script-src` タイプのポリシーを追加するだけで
- `script-src` ポリシーにはインラインの `<script>` タグのハッシュのみが含まれます。

`csp.addMeta` を `true` に設定した場合でも、定義されたポリシーの完全なセットが HTTP レスポンスヘッダーに追加されます。

`script-src` ポリシーに `'unsafe-inline'` が含まれている場合、CSP ハッシュは `<meta>` として追加されないことに注意してください。これはハッシュが存在する場合、ブラウザが `'unsafe-inline'` を無視するためです。CSPv1 との互換性のために `'unsafe-inline'` とハッシュの両方が必要な場合は `unsafeInlineCompatibility` オプションを `true` に設定してください。その場合 `<meta>` タグにはインラインの `<script>` タグのハッシュのみが含まれ、`csp.policies` で定義されたポリシーは `Content-Security-Policy` HTTP レスポンスヘッダーで使用されます。

```
export default {
  render: {
    csp: true
  }
}

// または

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
    }
  }
}

// または
/*
次の例では、Google Analytics、LogRocket.io、および Sentry.io で
ロギング、トラッキングの分析が行えます。

どのトラッキングリンクを使用すべきか学ぶためにSentry.io のブログを確認してください
https://blog.sentry.io/2018/09/04/how-sentry-captures-csp-violations
*/
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>`  
    ]  
}  
}  
}
```

rootDir プロパティ

Nuxt アプリケーションのワークスペースを定義します

-
- 型: `String`
 - デフォルト: `process.cwd()`

nuxt commands (nuxt start、nuxt build など) に引数が渡された場合、このプロパティは上書きされます。例えば `nuxt ./my-app/` を実行すると `rootDir` はカレント/ワーキングディレクトリである `./my-app/` の絶対パスが設定されます。

[Nuxt をプログラムで使用しない限り、このオプションを指定する必要はありません。](#)

Info

[依存関係を解決](#)するためには、`node_modules` ディレクトリを含むパッケージルートと `rootDir` が同じディレクトリツリー内にある必要があります。そうでない場合のディレクトリ構造の例については、[srcDir オプション](#)を参照してください。

router プロパティ

router プロパティを使って Nuxt ルーター ([vue-router](#))をカスタマイズできます。

base

- 型: `String`
- デフォルト: `'/'`

アプリケーションのベースの URL です。例えば、単一ページアプリケーション全体が `/app/` 下に配置される場合、`base` の値は `'/app/'` を指定します。

これは、より大きな Web サイト内から、Nuxt を別のコンテキストルートとして提供する必要がある場合に役立ちます。フロントプロキシウェブサーバーを設定する場合としない場合があることに注意してください。

`router.base` にリダイレクトしたい場合、[Hook を使ってリダイレクトすることができます。ルートでない場合 router.base にリダイレクトさせるを参照してください。](#)

Nuxt v2.15 以降では、実行時にこのプロパティの値を変更すると、既にビルドされているアプリケーションの設定が上書きされます。

```
export default {
  router: {
    base: '/app/'
  }
}
```

Info

`base` が設定されている場合、Nuxt はドキュメントヘッダー `<base href="{{ router.base }}"/>` も追加します。

このオプションは直接 `vue-router` の `base` に渡されます。

routeNameSplitter

- 型: `String`
- デフォルト: `'-'`

Nuxt が使うルート名間の区切り文字を変更したいかもしれません。その場合は設定ファイルの `routeNameSplitter` オプションを介して変更できます。ページファイル `pages/posts/_id.vue` があると仮定します。Nuxt はプログラムでルート名（今回の場合は `posts-id`）を生成します。`routeNameSplitter` の設定を `/` に変更するとルート名は `posts/id` に変更されます。

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

extendRoutes

- 型: `Function`

Nuxt によって作成されたルートを拡張したいことがあるかもしれません。その場合は `extendRoutes` オプションを介して拡張できます。

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

ルートをソートしたい場合、`@nuxt/utils` の `sortRoutes(routes)` 関数を使えます：

```
import { sortRoutes } from '@nuxt/utils'
export default {
  router: {
    extendRoutes(routes, resolve) {
      // ルートをここに追加する ...

      // それからソートする
      sortRoutes(routes)
    }
  }
}
```

ルートのスキーマは [vue-router](#) スキーマを尊重してください。

Warning

名前付きビューを使うルートを追加する場合、対応する名前付き `components` の `chunkNames` の追加を忘れないでください。

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

```
        modal: 'components/modal'  
    }  
})  
}  
}  
}
```

fallback

- 型: boolean
 - デフォルト: false

`history.pushState` がサポートされていないブラウザにおいて、モードが `history` に設定されているとき、ルーターを `hash` モードにフォールバックするかどうか制御します。 Controls whether the router should fallback to hash mode when the browser does not support `history.pushState` but mode is set to `history`.

これを `false` に設定すると、基本的にすべてのルーターリンクのナビゲーションが IE9 でフルページリフレッシュされるようになります。これは、アプリがサーバレンダリングされており、ハッシュモードのURLが SSR で動作しないため、IE9 で動作させる必要がある場合に便利です。

このオプションは直接 vue-router の `fallback` に渡されます。

linkActiveClass

- 型: `String`
 - デフォルト: `'nuxt-link-active'`

`<nuxt-link>` のデフォルトの active class をグローバルに設定します。

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

このオプションは直接 vue-router の `linkActiveClass` に渡されます。

linkExactActiveClass

- 型: `String`
 - デフォルト: `'nuxt-link-exact-active'`

`<nuxt-link>` のデフォルトの active class をグローバルに設定します。

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

このオプションは直接 vue-router の `linkExactActiveClass` に渡されます。

linkPrefetchedClass

- 型: `String`
- デフォルト: `false`

`<nuxt-link>` の prefetch クラスをグローバルに設定します（デフォルトは無効の機能）。

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

middleware

- 型: `String` または `Array`
 - 要素: `String`

アプリケーションのページごとにデフォルトのミドルウェアを設定します。

```
export default {
  router: {
    // ページごとに middleware/user-agent.js を実行します
    middleware: 'user-agent'
  }
}
```

```
export default function (context) {
  // userAgent プロパティを context (`asyncData` と `fetch` メソッドで利用できます) 内に追加します
  context.userAgent = process.server
    ? context.req.headers['user-agent']
    : navigator.userAgent
}
```

ミドルウェアについてより深く理解するために、[ミドルウェアのドキュメント](#)を参照してください。

mode

- 型: `String`
- デフォルト: `'history'`

ルーティングのモードを設定します。サーバーサイドレンダリングのためにこの設定を変更することはおすすめしません。

```
export default {
  router: {
    mode: 'hash'
  }
}
```

このオプションは直接 vue-router の `mode` に渡されます。

parseQuery / stringifyQuery

- 型: `Function`

カスタムクエリ構文解析関数/文字列化関数を提供します。デフォルトを上書きします。

このオプションは直接 vue-router の `parseQuery / stringifyQuery` に渡されます。

prefetchLinks

この機能は Nuxt v2.4.0 で追加されました

- 型: `Boolean`
- デフォルト: `true`

`viewport` (ブラウザの表示領域) 内でリンクが検出された時に コード分割された ページを先読みする `<nuxt-link>` の設定をします。`IntersectionObserver` がサポートされている必要があります ([CanIuse](#) を参照してください)。

この機能を [Polyfill.io](#) のようなサービスに条件付きで埋め込むことをおすすめします：

```
export default {
  head: {
    script: [
      {
        src: 'https://polyfill.io/v3/polyfill.min.js?features=IntersectionObserver',
        body: true
      }
    ]
  }
}
```

特定のリンクで先読みを無効にしたい場合、`no-prefetch` プロパティが使えます。Nuxt v2.10.0 からは `prefetch` プロパティ (`false` に設定) も使えます：

```
<nuxt-link to="/about" no-prefetch>About page not prefetched</nuxt-link>
<nuxt-link to="/about" :prefetch="false">About page not prefetched</nuxt-link>
```

全てのリンクで先読みを無効にしたい場合は、`prefetchLinks` を `false` に設定してください：

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

Nuxt v2.10.0 からは `prefetchLinks` を `false` に設定した上で特定のリンクを先読みしたい場合、`prefetch` プロパティが使えます：

```
<nuxt-link to="/about" prefetch>About page prefetched</nuxt-link>
```

prefetchPayloads

v2.13.0 で追加された機能で、[静的なターゲット](#)でのみ利用できます。

- 型: `Boolean`
- デフォルト: `true`

`target: 'static'` に設定した上で `nuxt generate` を使う場合、Nuxt はページごとに `payload.js` を生成します。

このオプションを有効にすると、viewport に `<nuxt-link>` が表示された時に Nuxt はリンク先ページのペイロードを自動的に先読みしインスタントナビゲーションを作成します。

Info

このオプションの有効化は `prefetchLinks` オプションに依存します。

`prefetchPayloads` を `false` に設定することで無効にできます：

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

scrollBehavior

- 型: `Function`

`scrollBehavior` オプションを使って、ページ間のスクロール位置についての独自の振る舞いを定義できます。このメソッドはページがレンダリングされるたびに毎回呼び出されます。詳細は [vue-router のスクロールの振る舞いのドキュメント](#) を参照してください。

v2.9.0 以降、ファイルを使用してルーターの `scrollBehavior` を上書きすることができます。このファイルは `~/app/router.scrollBehavior.js` に配置する必要があります（注意：Windows ではファイル名の大文字と小文字が区別されます）。

Warning

`router.scrollBehavior.js` ファイルはプロジェクトルートの `app` フォルダー内に置く必要があります。

Nuxt のデフォルトの `router.scrollBehavior.js` ファイルは次の場所にあります：

[packages/vue-app/template/router.scrollBehavior.js](#)

すべてのルートにおいて強制的にトップまでスクロールさせる例：

`app/router.scrollBehavior.js`

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

trailingSlash

- 型: `Boolean` または `undefined`
- デフォルト: `undefined`
- 利用できる最小バージョン: v2.10

このオプションを `true` に設定した場合、末尾のスラッシュがすべてのルートに追加されます。もし `false` に設定した場合は末尾のスラッシュは削除されます。

注意: このオプションは準備なしに設定すべきではなく、徹底的にテストする必要があります。`router.trailingSlash` に `undefined` 以外の値を設定すると、反対のルートは機能しなくなります。したがって、301 リダイレクトを設定し、内部リンクを正しく適応させる必要があります。`trailingSlash` を `true` に設定した場合、`example.com/abc/` のみが動作し `example.com/abc` は動作しません。`false` に設定する場合はその逆になります。

動作例(子ルートあり)

ディレクトリ構造：

```
- pages/
  --- index.vue
  --- posts.vue
  --- posts/
    ----- _slug.vue
    ----- index.vue
```

これは `trailingSlash` の設定がそれぞれ有効な場合の動作です：

Route	Page
/	<code>~/pages/index.vue</code>
/posts	<code>~/pages/posts.vue</code> (parent) + <code>~/pages/posts.vue</code> (child route)
/posts/	<code>~/pages/posts.vue</code> (parent) + <code>~/pages/posts.vue</code> (child route)
/posts/foo	<code>~/pages/posts.vue</code> (parent) + <code>~/pages/_slug.vue</code> (child route)
/posts/foo/	<code>~/pages/posts.vue</code> (parent) + <code>~/pages/_slug.vue</code> (child route)

Route	Page
/	<code>~/pages/index.vue</code>
/posts	<code>404</code>

/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

Runtime config プロパティ

RuntimeConfig を使うと、動的な設定や環境変数を nuxt コンテキストに渡すことができます。より詳細な使い方については [runtime config のドキュメント](#) を参照してください。

publicRuntimeConfig

- 型: `Object`

このオブジェクトの値は `$config` を使って **クライアントとサーバー両方からアクセスできます**。

privateRuntimeConfig

- 型: `Object`

このオブジェクトの値は `$config` を使って **サーバーからのみアクセスできます**。サーバー用の `publicRuntimeConfig` を上書きします。

[Go to TOC](#)

server プロパティ

Nuxt では、アプリケーションのサーバー接続用の変数を `nuxt.config.js` 内に定義できます。

- 型: `Object`

基本的な例:

```
export default {
  server: {
    port: 8000, // デフォルト: 3000
    host: '0.0.0.0', // デフォルト: localhost,
    timing: false
  }
}
```

これで Nuxt のサーバーインスタンスの `host` と `port` を指定できます。

HTTPS 設定を用いる例

```
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'))
    }
  }
}
```

サーバーの鍵と証明書を `localhost` 上で作成する方法については [certificates for localhost](#) の記事を参照してください。

sockets 設定を用いる例

```
export default {
  server: {
    socket: '/tmp/nuxt.socket'
  }
}
```

timing

- 型: `Object` または `Boolean`
- デフォルト: `false`

`server.timing` オプションを有効にすると、サーバーサイドレンダリング中に経過した時間を計測するミドルウェアが追加され、「Server-Timing」としてヘッダーに追加されます。

timing 設定用いる例

`server.timing` はオプションを提供するためのオブジェクトです。現在、`total` のみがサポートされています（これはサーバーサイドレンダリングで費やした全ての時間を直接追跡します）。

```
export default {
  server: {
    timing: {
      total: true
    }
  }
}
```

timing API を使う

`timing` API は `server.time` が有効な場合にサーバーサイドの `response` にも注入されます。

構文

```
res.timing.start(name, description)
res.timing.end(name)
```

サーバーミドルウェアで timing を用いる例

```
export default function (req, res, next) {
  res.timing.start('midd', 'Middleware timing description')
  // サーバーサイドの処理..
  // ...
  res.timing.end('midd')
  next()
}
```

そして `server-timing` は以下のようにレスポンスヘッダーに含まれます：

```
Server-Timing: midd;desc="Middleware timing description";dur=2.4
```

詳細は [MDN の Server-Timing](#) を参照してください。

[Go to TOC](#)

serverMiddleware プロパティ

サーバーサイドミドルウェアを定義します。

- 型: `Array`
 - 要素: `String` または `Object` または `Function`

Nuxt はカスタムミドルウェアを追加できる `connect` インスタンスを内部で作ります。これにより外部サーバーを必要とせずに通常は `/api` ルート) を登録できます。

`connect` 自体はミドルウェアなので、登録されたミドルウェアは `nuxt start` と `express-template` のようなプログラムで使われるミドルウェアとして使う場合の両方で動作します。Nuxt モジュールもまた `this.addServerMiddleware()` を使って `serverMiddleware` を提供できます。

それらに加え、デフォルト値が `true` の `prefix` オプションを導入しました。サーバーミドルウェアに `router base` を追加します。

例:

- サーバーミドルウェアパス: `/server-middleware`
- Router base: `/admin`
- `prefix: true` の場合 (デフォルト): `/admin/server-middleware`
- `prefix: false` の場合: `/server-middleware`

サーバーミドルウェア vs ミドルウェア!

クライアントサイドやサーバーサイドレンダリングで Vue によって各ルートの前に呼び出されるルートのミドルウェアと混同しないでください。`serverMiddleware` プロパティに列挙されているミドルウェアは `vue-server-renderer` の前にサーバーサイドで実行され、API リクエストの処理やアセットの提供などのサーバー固有のタスクに使用できます。

Warning

サーバーミドルウェアを `middleware/` ディレクトリに追加しないでください。

ミドルウェアは `webpack` によって本番バンドルにバンドルされ `beforeRouteEnter` で実行されます。`serverMiddleware` を `middleware/` ディレクトリに追加すると、Nuxt によってミドルウェアとして誤って選択され、誤った依存関係がバンドルに追加されたり、エラーが発生したりします。

使い方

もしミドルウェアが文字列の場合、Nuxt はそのミドルウェアを自動的に解決し要求します。

```
import serveStatic from 'serve-static'

export default {
  serverMiddleware: [
    // redirect-ssl npm パッケージを登録します
    'redirect-ssl',
  ],
}
```

```
// /server-middleware/* を処理するために、プロジェクトの server-middleware ディレクト  
リからファイルを登録します  
{ path: '/server-middleware', handler: '~/server-middleware/index.js' },  
  
// カスタムインスタンスを作成することもできます  
{ path: '/static2', handler: serveStatic(__dirname + '/static2') }  
]  
}
```

Warning

もしミドルウェアをすべてのルートに登録したくない場合は、特定のパスでオブジェクトフォームを使用する必要があります。そうしないと nuxt のデフォルトハンドラは機能しません。

カスタムサーバーミドルウェア

カスタムミドルウェアの作成も可能です。詳細については [Connect Docs](#) を参照してください。

ミドルウェア (`server-middleware/logger.js`) :

```
export default function (req, res, next) {  
  // req は Node.js の HTTP リクエストオブジェクトです  
  console.log(req.url)  
  
  // res は Node.js の HTTP レスポンスオブジェクトです  
  
  // next は 次のミドルウェアを呼び出すための関数です。  
  // ミドルウェアがエンドポイントでない場合、関数の最後で next を呼び出すのを忘れないでください！  
  next()  
}  
  
serverMiddleware: ['~/server-middleware/logger']
```

カスタム API エンドポイント

サーバーミドルウェアも Express を拡張できます。これにより REST エンドポイントを作成できます。

```
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" },  
],
```

オブジェクトの構文

パスにマッピングされた関数のリストでサーバーミドルウェアが構成されている場合：

```
export default {
  serverMiddleware: [
    { path: '/a', handler: '~/server-middleware/a.js' },
    { path: '/b', handler: '~/server-middleware/b.js' },
    { path: '/c', handler: '~/server-middleware/c.js' }
  ]
}
```

あるいは、次のとおりオブジェクトを渡して定義することもできます：

```
export default {
  serverMiddleware: {
    '/a': '~/server-middleware/a.js',
    '/b': '~/server-middleware/b.js',
    '/c': '~/server-middleware/c.js'
  }
}
```

srcDir プロパティ

Nuxt アプリケーションのソースディレクトリを定義します。

- 型: `String`
- デフォルト: `rootDir` 値

相対パスを指定すると、`rootDir` からの相対パスになります。

例 1: 前提条件 :

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

```
"script": {
  "dev": "yarn nuxt"
}
```

次のフォルダ構成で動作します (`nuxt.config` は `app` ディレクトリにあることに注意してください)。

```
- app/
--- node_modules/
--- nuxt.config.js
--- package.json
--- client/
| assets/
| components/
| layouts/
| middleware/
| pages/
| plugins/
| static/
| store/
```

例 2 :

例 1 の代わりに `nuxt.config` を `client` フォルダに移動することもできます。この場合 `client` を `rootDir` として指定するだけでよく、`srcDir` は空にできます :

前提条件 :

```
export default {
  srcDir: '' // または単に削除
}
```

```
"script": {
  "dev": "yarn nuxt client" // client を rootDir として設定
}
```

次のフォルダ構成で動作します (`nuxt.config` は `client` ディレクトリにあることに注意してください)。

```
- | app/
---| node_modules/
---| package.json
---| client/
| nuxt.config.js
| assets/
| components/
| layouts/
| middleware/
| pages/
| plugins/
| static/
| store/
```

buildDir プロパティ

Nuxt アプリケーションの dist ディレクトリを定義します。

- 型: `String`
- デフォルト: `.nuxt`

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

名前がドットで始まるため、デフォルトでは多くのツールが `.nuxt` を隠しディレクトリとみなします。このオプションを使うことでそれを防ぐことができます。

[Go to TOC](#)

ssr プロパティ

デフォルト nuxt ssr の値を変更します。

- 型: `boolean`
- デフォルト: `true`
- 設定できる値:
 - `true`: サーバーサイドレンダリングが有効になります
 - `false`: サーバーサイドレンダリングを行いません（クライアント側のレンダリングのみとなります）

クライアント側のレンダリングのみ行いたい場合にオプションを `false` に設定できます

```
export default {  
  ssr: false // サーバーサイドレンダリングを無効化  
}
```

Next

以前はサーバーサイドレンダリングを有効にするか無効にするかの設定に `mode` プロパティが使われていました。[mode ドキュメント](#) はこちらです。

target プロパティ

デフォルト nuxt ターゲットを変更します

Nuxt v2.13 以上でのデプロイターゲット：

- 型: `string`
 - デフォルト: `server`
 - 設定できる値:
 - `'server'` : サーバーサイドレンダリング用
 - `'static'` : 静的サイト用

`nuxt.config.js` でこのオプションを使ってプロジェクトのデフォルトの nuxt target を変更します。

target オプションの詳細については[デプロイターゲットのドキュメント](#)を参照してください。

[Go to TOC](#)

telemetry プロパティ

Nuxt は、一般的な使用状況に関する匿名のテレメトリデータを収集します。これにより、すべてのユーザーにおける Nuxt 機能の使用状況やカスタマイズ状況を正確に把握します。

telemetry プロパティ

一般的な使用状況に関する匿名のテレメトリデータを収集するために Nuxt Telemetry が Nuxt v2.13.0 で導入されました。これによりすべてのユーザーの Nuxt 機能の使用状況やカスタマイズ状況を正確に把握できます。

- 型: `Boolean`
- デフォルト値はユーザーの設定に基づきます

なぜテレメトリデータを集めるのか

Nuxt は初期リリース (2016/11/7) から大きく成長しており、改善のため [コミュニティのフィードバック](#) に耳を傾けています。

しかしこの手動プロセスでは、記入に時間のかかる issue テンプレートからユーザーのサブセットによるフィードバックを収集するだけであり、人によってニーズやユースケースが異なる場合があります。

Nuxt Telemetry は一般的な使用状況に関する匿名のテレメトリデータを収集します。これはすべてのユーザーの機能の使用状況やカスタマイズ状況を正確に把握することに役立ちます。このデータは Nuxt が世界的にどのように使われているかや改善点 (DX とパフォーマンス) とその関連性を測定し、よりよく理解することに役立ちます。

複数のイベントを収集します：

- コマンドの呼び出し (nuxt dev、nuxt build など)
- Nuxt と Node.js のバージョン
- 一般的なマシン情報 (MacOS/Linux/Windows や CI 内で実行されるコマンド、ci 名)
- Webpack のビルド時間とアプリケーションの平均サイズ、および生成統計 (nuxt generate 使用時)
- プロジェクトのパブリックな依存関係は何か (Nuxt モジュール)

コードはオープンソースであり <https://github.com/nuxt/telemetry> で入手できます。

オプトアウト

Nuxt Telemetry を無効化する方法はいくつかあります：

1. `npx nuxt telemetry disable` を使う

```
npx nuxt telemetry [status|enable|disable] [-g,--global] [dir]
```

2. 環境変数を使う

NUXT_TELEMETRY_DISABLED=1

3. `nuxt.config.js` で `telemetry: false` を設定する：

```
export default {  
  telemetry: false  
}
```

Nuxt Telemetry と送信されるイベントの詳細は[こちら](#)を参照してください。

[Go to TOC](#)

Transition プロパティ

ページのデフォルトプロパティとレイアウトトランジションを設定します。

pageTransition プロパティ

Nuxt v2.7.0 では "transition" キーに代わり "pageTransition" キーが導入され、名前が layout transition キーに統合されています。

- 型: `String` または `Object`

ページトランジションのデフォルトプロパティを設定するために使われます。

デフォルト：

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

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

`nuxt.config.js` 内の `transition` キーはページトランジションのデフォルトプロパティを指定するために使われます。`transition` キーがオブジェクトのときに利用可能なキーの詳細については[ページトランジションのプロパティ](#)を参照してください。

layoutTransition プロパティ

- 型: `String` または `Object`

レイアウトトランジションのデフォルトプロパティを設定するために使われます。`name` オプションで指定された値は `layouts` フォルダの `layout` で指定された名前で動くように設定されています。

デフォルト：

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

```
export default {  
  layoutTransition: 'layout'  
  // or  
  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](#)

vue.config プロパティ

Vue.config 向けの設定オブジェクト

- 型: `Object`
- デフォルト: `{ silent: !isDev, performance: isDev }`

vue.config プロパティは `Vue.config` 用に直接設定できる橋渡しを提供します。

例

```
export default {
  vue: {
    config: {
      productionTip: true,
      devtools: false
    }
  }
}
```

この設定により次の `Vue.config` が作成されます：

```
Vue.config.productionTip // true
Vue.config.devtools // false
Vue.config.silent // !isDev [デフォルト値]
Vue.config.performance // isDev [デフォルト値]
```

`Vue.config` API の詳細については[公式の Vue のドキュメント](#)を参照してください。

watch プロパティ

watch プロパティはサーバー再起動用にカスタムファイルを監視することができます。

- 型: `Object`
- デフォルト: `{}`

`watch: ['~/custom/*.js']`

`chokidar` は watcher の設定に使われます。 `chokidar` のパターンオプションの詳細は [chokidar API](#) を参照してください。

[Go to TOC](#)

watchers プロパティ

nuxt.config.js の watchers プロパティは watcher の設定を上書きできます。

- 型: `Object`
- デフォルト: `{}`

chokidar

- 型: `Object`
- デフォルト: `{}`

chokidar オプションの詳細は [chokidar API](#) を参照してください。

webpack

- 型: `Object`
- デフォルト:

```
watchers: {  
  webpack: {  
    aggregateTimeout: 300,  
    poll: 1000  
  }  
}
```

webpack の watchOptions の詳細は [webpack のドキュメント](#) を参照してください。

この次は?

Next

[Internals Glossary のドキュメント](#) を参照してください。

cli プロパティ

Nuxt では CLI の設定をカスタマイズできます。

badgeMessages

- 型: `Array`

CLI バナーにメッセージを追加します。

```
cli: {  
  badgeMessages: ['Hello World!']  
}
```



bannerColor

- 型: `String`
- デフォルト: `'green'`

CLI バナーの「Nuxt」タイトルの色を変更します。

利用可能な色:

`black`、`red`、`green`、`yellow`、`blue`、`magenta`、`cyan`、`white`、`gray`、`redBright`、`greenBright`、`yellowBright`、`blueBright`、`magentaBright`、`cyanBright`、`whiteBright`

```
export default {
  cli: {
    bannerColor: 'yellow'
  }
}
```

[Go to TOC](#)

CSS プロパティ

Nuxt ではグローバルに適用したい（すべてのページにインクルードしたい）CSS ファイル/モジュール/ライブラリを設定できます。

`sass` を利用したい場合は `sass` および `sass-loader` パッケージをインストールしてください。もしインストールしていない場合は以下のようにインストールしてください。

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

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

- 型: `Array`
 - 要素: `string`

```
export default {
  css: [
    // Node.js モジュールを直接ロードする（ここでは Sass ファイル）
    'bulma',
    // プロジェクト内の CSS ファイル
    '@/assets/css/main.css',
    // プロジェクト内の SCSS ファイル
    '@/assets/css/main.scss'
  ]
}
```

Nuxt は拡張子から自動的にファイルタイプを推測して webpack のための適切なプリプロセッサローダを使用します。ただし使用する必要のあるローダーは各自でインストールしてください。

スタイルの拡張子

`nuxt.config.js` の `css` 配列に指定するファイル（CSS/SCSS/Postcss/Less/Stylus/...）の拡張子を省略できます。

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

Warning

`main.scss` と `main.css` のように同じ名前のファイルを 2 つ持っていて `css` 配列に拡張子を指定しない（例えば `css: ['~/assets/css/main']`）場合、`styleExtensions` の順序に応じて 1 つのファイルしか読み込まれません。デフォルトの `styleExtension` 配列の 1 番目が `css` なので、今回の場合 `css` ファイルは読み込まれますが `scss` ファイルは無視されます。

デフォルトの順序: `['css', 'pcss', 'postcss', 'styl', 'stylus', 'scss', 'sass', 'less']`

[Go to TOC](#)

components プロパティ

Nuxt v2.13 以上では `@nuxt/components` モジュールを使ってコンポーネントのスキャンと自動インポートができます。

`components` この機能は、Nuxt 2.10 - 2.12で使用することができます。手動でインストールして、`@nuxt/components` を `nuxt.config` 内の `buildModules` に追加するだけです。

- 型: `Boolean` または `Array`
- デフォルト: `false`

`true` またはオブジェクトオプションを設定したとき、Nuxt は `@nuxt/components` を組み込んでそして `pages`、`layouts`（および他のコンポーネント）内でコンポーネントを使用する際には、自動的にインポートされます。

Info

詳しい使い方は、[自動検索の有効 ドキュメント](#)を参照してください。

設定

```
export default {
  // これは自動的に、`~/components` からコンポーネントを読み込みます
  components: true
}
```

`components: true` を設定すると、デフォルトでは、`~/components` ディレクトリが組み込まれます。

しかしながら、スキャンするディレクトリを追加することで、自動検測の動作をカスタマイズすることができます：

```
export default {
  components: [
    // { path: '~/components' } に相当
    '~/components',
    { path: '~/components/other', extensions: ['vue'] }
  ]
}
```

パス

各項目は、文字列またはオブジェクトのいずれかです。文字列の値は、`{ path }` に対するショートカットです。

Info

ディレクトリの順番や重複を心配する必要はありません。コンポーネントモジュールが処理してくれます。(各ファイルは最長のパスで一度だけ照合されます)

path

- 必須項目
- 型: `String`

コンポーネントを含むディレクトリへのパス（絶対または相対）。

Nuxt のエイリアス (~ や @) を使ってプロジェクト内のディレクトリを参照したり、npm パッケージのパスを直接使用することができます（プロジェクト内で `require` を使用するのと同様です）。

extensions

- 型: `Array<string>`
- デフォルト:
 - Nuxt ビルダー (`builder.supportedExtensions`) によって拡張子がサポート済み
 - デフォルトは拡張子 `['vue', 'js']` がサポート、または `['vue', 'js', 'ts', 'tsx']` は環境次第

例: 複数ファイルコンポーネント構造をサポート

SFC を `.js`、`.vue`、`.css` に分割したい場合は、`.vue` ファイルのみをスキャンすることも可能です：

```
| components
---| componentC
| componentC.vue
| componentC.js
| componentC.scss
```

```
// nuxt.config.js
export default {
  components: [{ path: '~/components', extensions: ['vue'] }]
}
```

pattern

- 型: `string` (glob パターン)
- デフォルト: `**/*.{${extensions.join(',')}}}`

指定された `path` において、このパターンにマッチするファイルのみが含まれます。

ignore

- 型: `Array`
- 要素: `string` (glob パターン)
- デフォルト: `[]`

指定された `path` 内のファイルを除外するパターン

prefix

- 型: `String`
- デフォルト: `''` (プリフィックスなし)

マッチするすべてのコンポーネントのプリフィックス

以下の例では、`awesome/` ディレクトリ内のコンポーネントの名前に、`awesome-` / `Awesome` というプレフィックスを追加しています。

```
// 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>
```

pathPrefix

- 型: Boolean
- デフォルト: true

コンポーネント名の前に、そのパスを付加する。

watch

- 型: Boolean
- デフォルト: true

指定された path を監視して、ファイルの追加や削除などの変更をする。

transpile

- 型: Boolean
- デフォルト: 'auto'

指定された path を build.transpile を使ってトランスペイルする。デフォルト ('auto') では、 node_modules/ が path にある場合、 transpile: true を設定します。

level

- 型: Number
- デフォルト: 0

レベルは、2つの異なるディレクトリにある同じ名前のコンポーネントの上書きを許可するために使用されます。これは、ユーザーが自分のコンポーネントを上書きできるようにしたいライブラリの作成者や、カスタムテーマの作成者にとって便利です。

```
export default {
  components: [
    '~/components', // デフォルトレベルは 0
```

```

        { path: 'my-theme/components', level: 1 }
    ]
}

```

`~/components` にあるコンポーネントは、`my-theme/components` にある同じ名前のコンポーネントを上書きします。値の小さいものが優先されます。

高度

コンポーネントの上書き

`level` オプションを使ってコンポーネントを上書きする方法があります。これは、モジュールやテーマの作者にとって非常に便利です。

以下の構造を考え：

```

| node_modules/
---| my-theme/
| components/
| Header.vue
| components/
---| Header.vue

```

そして、`nuxt.config` で定義すると：

```

components: [
  '~/components', // デフォルトルベルは 0
  { path: 'node_modules/my-theme/components', level: 1 }
]

```

`components/Header.vue` は、最下層が優先されるので、テーマコンポーネントを上書きします。

ライブラリ作者

tree-shaking とコンポーネントの自動登録機能を備えた Vue コンポーネントライブラリの作成が超簡単になりました！。

このモジュールは、`components:dirs` という名前のフックを公開しているので、Nuxtモジュールでのユーザー設定を必要とせずに、ディレクトリリストを簡単に拡張することができます。

以下のようなディレクトリ構造を想像してみてください：

```

| node_modules/
---| awesome-ui/
| components/
| Alert.vue
| Button.vue
| nuxt.js
| pages/
---| index.vue
| nuxt.config.js

```

それから、`awesome-ui/nuxt.js` で、`components:dir` というフックを使うことができます：

```

import { join } from 'path'

export default function () {
  this.nuxt.hook('components:dirs', dirs => {
    // ./components の dir をリストに追加する
    dirs.push({
      path: join(__dirname, 'components'),
      prefix: 'awesome'
    })
  })
}

```

これで完成です。これで、あなたのプロジェクトでは、`nuxt.config.js` で ui ライブリを Nuxt モジュールとしてインポートすることができます：

```

export default {
  buildModules: ['@nuxt/components', 'awesome-ui/nuxt']
}

```

そして、モジュールのコンポーネント（プレフィックスは `awesome-`）を直接使用して、私たちの `pages/index.vue` を作成します：

```

<template>
  <div>
    My <AwesomeButton>UI button</AwesomeButton>!
    <awesome-alert>Here's an alert!</awesome-alert>
  </div>
</template>

```

これは、使用されている場合にのみ自動的にコンポーネントをインポートし、`node_modules/awesome-ui/components/` でコンポーネントを更新する際に HMR をサポートします。

次：作成した `awesome-ui` モジュールを `npm` に公開して、他の Nuxters と共有しましょう！

dev プロパティ

開発モードかプロダクションモードか定義します。

- 型: Boolean
- デフォルト: true

このプロパティは nuxt コマンドによって上書きされます :

- nuxt コマンドを使う場合は dev は強制的に true になります
- nuxt build、nuxt start そして nuxt generate を使う場合は dev は強制的に false になります

このプロパティは [Nuxt をプログラム](#) で使うときに設定します :

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

```
const { Nuxt, Builder } = require('nuxt')
const app = require('express')()
const port = process.env.PORT || 3000

// オプションを使って Nuxt をインスタンス化する
const config = require('./nuxt.config.js')
const nuxt = new Nuxt(config)
app.use(nuxt.render)

// 開発モードのときのみビルドする
if (config.dev) {
  new Builder(nuxt).build()
}

// サーバーを Listen する
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](#)

dir プロパティ

Nuxt アプリケーションに対してカスタムディレクトリを定義します

- 型: `Object`
- デフォルト:

```
{  
  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'  
  }  
}
```

env プロパティ

クライアントサイドとサーバーサイドで環境変数を共有できます。

- 型: `Object`

Nuxt ではクライアントサイドとサーバーサイドで共有される環境変数を作成できます。

`env` プロパティはクライアントサイドで使用できる環境変数を定義します。これらはサーバーサイド環境変数や [dotenv モジュール](#)などを使用して割り当てることができます。

`alert nuxt` のバージョン > 2.12+ では、(ビルト時ではなく) ランタイムに環境変数が必要な場合、`env` プロパティを [Runtime config プロパティ](#) の `publicRuntimeOptions` と `privateRuntimeOptions` で代替することができます。

詳細は、チュートリアルの [Moving from `@nuxtjs/dotenv` to runtime config] (/tutorials/moving-from-nuxtjs-dotenv-to-runtime-config/) をご覧ください。

より良いトラブルシューティングのために、後述の `process.env` と `process.env == {}` を必ず読んでください。

```
export default {
  env: {
    baseUrl: process.env.BASE_URL || 'http://localhost:3000'
  }
}
```

これによりサーバーサイドの環境変数 `BASE_URL` が利用可能または定義されていれば、`BASE_URL` と等しい `baseUrl` プロパティを作成できます。もし `BASE_URL` が利用不可であったり定義されていなければ、クライアントサイドの `baseUrl` は `'http://localhost:3000'` になります。サーバーサイドの環境変数 `BASE_URL` は `nuxt.config.js` の `env` プロパティを経由してクライアントサイドにコピーされます。または、他の値が定義されています。

そして `baseUrl` 変数にアクセスするには 2 つの方法があります：

- `process.env.baseUrl` 経由でアクセスする。
- `context.env.baseUrl` 経由でアクセスする。詳細は [context API](#) を参照してください。

例えば `env` プロパティを使って公開トークンを付与できます。

上記の例として、`env` プロパティを使って `axios` を設定できます。

```
import axios from 'axios'

export default axios.create({
  baseURL: process.env.baseUrl
})
```

このように記述するとページ内で `import axios from '~/plugins/axios'` という具合に `axios` をインポートできます。

環境変数の自動注入

ビルドフェーズで `NUXT_ENV_` で始まる環境変数を定義（例えば `NUXT_ENV_COOL_WORD=freezing nuxt build` または Windows の `console` に対して `SET NUXT_ENV_COOL_WORD=freezing & nuxt build`）した場合、この環境変数はプロセス環境に自動的に注入されます。`nuxt.config.js` 内に定義した同名変数よりも優先されるので注意してください。

`process.env ==`

Nuxt は webpack の `definePlugin` を用いて環境変数を定義することに注意してください。つまり、Node.js の実際の `process` または `process.env` が利用可能でもなければ定義されてもいいことを意味します。`nuxt.config.js` で定義された `env` プロパティは `process.env.xxxx` へそれぞれ個別にマップされコンパイル時に変換されます。

つまり、`console.log(process.env)` は `{}` を出力しますが、`console.log(process.env.your_var)` は定義された値を出力します。コードが webpack でコンパイルされると、`process.env.your_var` と記述されたすべての箇所が定義した値に置き換えられます。例えば、`env.test = 'testing123'` と定義すると、コード中に `process.env.test` と記述してある箇所がすべて '`testing123`' へ置き換えられます。

前

```
if (process.env.test == 'testing123')
```

後

```
if ('testing123' == 'testing123')
```

serverMiddleware

`serverMiddleware` はメインの Nuxt ビルドから分離されているので、`nuxt.config.js` に定義された `env` 変数は使用できません。

コンテキスト

`context` は、Nuxt から Vue コンポーネントに追加のオブジェクト/パラメータを提供し、`asyncData`、`fetch`、`plugins`、`middleware` そして `nuxtServerInit` のような特別な Nuxt ライフサイクル内で使用できます。

注意: ここで言う "コンテキスト" と、 Vuex のアクション の中で使用できる `context` オブジェクトを混同しないでください。2 つは無関係です。

```
function (context) {
  // ユニバーサルなキー
  const {
    app,
    store,
    route,
    params,
    query,
    env,
    isDev,
    isHMR,
    redirect,
    error,
    $config
  } = context
  // サーバーサイド
  if (process.server) {
    const { req, res, beforeNuxtRender } = context
  }
  // クライアントサイド
  if (process.client) {
    const { from, nuxtState } = context
  }
}
```

ユニバーサルなキー

これらのキーは、クライアントサイドとサーバーサイドの両方で使用できます。

app

`app` (*NuxtAppOptions*)

すべてのプラグインを含むルート Vue インスタンスのオプション。たとえば `i18n` を使用している場合は `context.app.i18n` で `$i18n` にアクセスできます。

store

`store` (*Vuex Store*)

Vuex の Store インスタンス。[Vuex ストア](#)が設定されている場合のみ使用できます。

route

`route` ([Vue Router Route](#))

Vue Router の route インスタンス。

params

`params` (*Object*)

`route.params` のエイリアス。

query

`query` (*Object*)

`route.query` のエイリアス。

env

`env` (*Object*)

`nuxt.config.js` で設定された環境変数。 [env API](#)を参照してください。

isDev

`isDev` (*Boolean*)

開発モードであるかどうかを示す真偽値で、本番環境で一部のデータをキャッシュするのに役立ちます。

isHMR

`isHMR` (*Boolean*)

メソッド/ミドルウェアが webpack の hot module replacement から呼び出されたかどうかを示す真偽値（開発モードのクライアントサイドでのみ `true`）。

redirect

`redirect` (*Function*)

ユーザーを別のルートにリダイレクトするには、このメソッドを使用してください。ステータスコードはサーバーサイドで使用され、デフォルトは `302` です。 `redirect([status,] path [, query])`。

例：

```
 redirect(302, '/login')
 redirect({ name: 'slug', params: { slug: mySlug } })
 redirect('https://vuejs.org')
```

Location プロパティの詳細については、[Vue Router のドキュメント](#)を参照してください。

Info

ハイドレーションエラー（クライアントの内容が、サーバーが返すものと異なってしまう）のため、[クライアントサイド限定の Nuxt プラグイン](#)の中では `redirect` や `error` を使用することはできません。

有効な回避策は、`window.onNuxtReady(() => { window.$nuxt.$router.push('/your-route') })` を使用することです。

error

`error` (*Function*)

エラーページを表示するには、このメソッドを使用してください。例: `error(params)`。`params` には `statusCode` と `message` プロパティが必要です。

\$config

`$config` (*Object*)

実際の[ランタイム設定](#)。

サーバーサイドのキー

これらのキーはサーバーサイドでのみ使用できます。

req

`req` ([http.Request](#))

Node.js サーバーからのリクエスト。Nuxt がミドルウェアとして使われる場合、リクエストオブジェクトは使用しているフレームワークによって異なる可能性があります。

`nuxt generate` では使用できません。

Res

`res` ([http.Response](#))

Node.js サーバーからのレスポンス。Nuxt がミドルウェアとして使われる場合、レスポンスオブジェクトは使用しているフレームワークによって異なる可能性があります。

`nuxt generate` では使用できません。

beforeNuxtRender

`beforeNuxtRender(fn)` (*Function*)

クライアントサイドでレンダリングされた `__NUXT__` 変数を更新するには、このメソッドを使用してください。`fn` (非同期にもできます) は `{ Components, nuxtState }` で呼び出されます。[例](#)を参照してください。

クライアントサイドのキー

これらのキーはクライアントサイドでのみ使用できます。

from

`from` (*Vue Router Route*)

遷移元の route。

nuxtState

`nuxtState` (*Object*)

Nuxt の状態。ハイドレーション前のクライアントサイドで Nuxt の状態を取得するために `beforeNuxtRender` を使うプラグインに便利です。`universal` モードでのみ使用できます。

nuxt.render(req, res)

`nuxt.render` を使うと Node.js サーバーのミドルウェアとして Nuxt を使うことができます。

- 型: `Function`
- 引数:
 - `Request`
 - `Response`
- 戻り値: `Promise`

[Express](#) による例 :

```
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() {
  // Nuxt インスタンスを取得
  const nuxt = await loadNuxt(isDev ? 'dev' : 'start')

  // すべてのルートを Nuxt でレンダリング
  app.use(nuxt.render)

  // ホットリローディングつきの開発モードの場合のみビルド
  if (isDev) {
    build(nuxt)
  }
  // サーバーをリッスン
  app.listen(port, '0.0.0.0')
  console.log(`Server listening on `localhost:${port}``)
}

start()
```

Warning

ミドルウェアの終わりに `nuxt.render` を呼び出すことをお勧めします。`nuxt.render` は Web アプリケーションのレンダリングを処理し `next()` を呼び出さないからです。

[Go to TOC](#)

nuxt.renderRoute(route, context)

渡されたコンテキストを使い、特定のルートをレンダリングします。

- 型: `Function`
- 引数:
 - `String` : レンダリングするルート
 - オプション `Object`、渡されたコンテキスト、利用可能なキー: `req` と `res`
- 戻り値: `Promise`
 - `html` : `String`
 - `error` : `null` または `Object`
 - `redirected` : `false` または `Object`

このメソッドはほとんどの場合 `nuxt.renderAndgetWindow` とともにテストする目的で使われます。

Warning

`nuxt.renderRoute` はプロダクションモードではビルド処理の後に実行するといいでしょう。

```
const { loadNuxt, build } = require('nuxt')

async function start() {
  // 起動用の nuxt インスタンスを取得します（プロダクションモード）
  // このスクリプトを実行する前に `nuxt build` を実行していることを確認してください
  const nuxt = await loadNuxt({ for: 'start' })

  const { html, error, redirected } = await nuxt.renderRoute('/')

  // `html` は常に文字列になります

  // エラーレイアウトが表示されるときは `error` は null ではありません。エラーフォーマットは下記:
  // { statusCode: 500, message: 'My error message' }

  // `asyncData()` または `fetch()` 内で `redirect()` が使われたときは `redirected` は
  // `false` ではありません
  // { path: '/other-path', query: {}, status: 302 }
}

start()
```

nuxt.renderAndGetWindow(url, options)

Nuxt アプリケーションの与えられた URL から `window` を取得します。

- 型: `Function`
- 引数: `String`
 1. `String`: レンダリングする URL
 2. オプション `Object`: オプション
 - `virtualConsole`: `Boolean` (デフォルト: `true`)
- 戻り値: `Promise`
 - 戻り値: `window`

Warning

このメソッドはテスト目的で作られました。

この関数を使うためには `jsdom` をインストールする必要があります：

```
npm install --save-dev jsdom
```

例：

```
const { loadNuxt } = require('nuxt')

async function init() {
  // すでにプロジェクトをビルド済みと仮定します
  const nuxt = await loadNuxt({ for: 'start' })
  await nuxt.listen(3000)
  const window = await nuxt.renderAndGetWindow('http://localhost:3000')
  // head `<title>` を表示する
  console.log(window.document.title)
  nuxt.close()
}

init()
```

この次は？

Next

[コンポーネント用語集](#)をチェックしてみましょう。

[Go to TOC](#)

\$nuxt: Nuxt ヘルパー

\$nuxt はユーザーエクスペリエンスを向上するために設計されたヘルパーです。

Nuxt ヘルパーの詳細については [コンテキストとヘルパーのドキュメント](#) を参照してください。

Connection checker

- `isOffline`
 - 型: `Boolean`
 - 説明: ユーザーのインターネット接続環境がオフラインになった時に `true`
- `isOnline`
 - 型: `Boolean`
 - 説明: `Opposite of isOffline` の反対

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

ページデータのリフレッシュ

- `refresh()`
 - `asyncData` または `fetch` が提供するデータのみをリフレッシュしたいとき

```
<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>
```

ローディングバーの制御

- `$loading`
 - Nuxt のローディングバーをプログラムで制御したいとき

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

Nuxt モジュールの紹介

Nuxt 内部の理解を深める

Nuxt は完全にモジュール化されたアーキテクチャを備えているため、開発者は柔軟な API を使用して Nuxt Core の任意の部分を拡張できます。

独自のモジュール開発に興味がある場合は[モジュールガイド](#) のドキュメントを参照してください。

このセクションは Nuxt 内部に詳しくなるために役立ち、独自のモジュールを作成する際に Nuxt 内部をよりよく理解するためのリファレンスとして使えます。

コア

これらのクラスは Nuxt の心臓部であり、実行時とビルド時の両方に存在する必要があります。

Nuxt

- [Nuxt クラス](#)
- ソース: [core/nuxt.js](#)

Renderer

- [Renderer クラス](#)
- ソース: [vue-renderer/renderer.js](#)

ModuleContainer

- [ModuleContainer クラス](#)
- ソース: [core/module.js](#)

Build

これらのクラスは、ビルドモードまたは開発モードでのみ必要です。

Builder

- [Builder クラス](#)
- ソース: [builder/builder.js](#)

Generator

- [Generator クラス](#)
- ソース: [generator/generator.js](#)

共通

Utils

- ソース: [utils/src](#)

Options

- ソース: [config/options.js](#)

パッケージング&使用方法

Nuxt はデフォルトですべてのクラスをエクスポートします。それらをインポートするには：

```
import { Nuxt, Builder, Utils } from 'nuxt'
```

共通のパターン

すべての Nuxt クラスには `nuxt` インスタンスとオプションへの参照を持ちます。そのため `options` と `nuxt` にアクセスするためのクラス間で一貫した API を常に持っています。

```
class SomeClass {
  constructor(nuxt) {
    super()
    this.nuxt = nuxt
    this.options = nuxt.options
  }

  someFunction() {
    // `this.nuxt` と `this.options` へアクセスできます
  }
}
```

クラスは プラガブル なのでより多くのフックを登録するには `nuxt` コンテナにプラグインを登録する必要があります。

```
class FooClass {
  constructor(nuxt) {
    super()
    this.nuxt = nuxt
    this.options = nuxt.options

    this.nuxt.callHook('foo', this)
  }
}
```

以下のように `foo` モジュールにフックできます：

```
nuxt.hook('foo', foo => {
  // ...
})
```

[Go to TOC](#)

Nuxt クラス

すべてのモジュールとクラスが通信できるようにするためのコアコンテナです。すべてのモジュールは `this.nuxt` を使って Nuxt インスタンスにアクセスできます。

- ソース: [core/nuxt.js](#)

フック

特定のライフサイクルイベントでフックを登録できます。

```
nuxt.hook('ready', async nuxt => {
  // ここにコードを記述してください
})
```

| プラグイン | 引数 | タイミング | | | `ready` | (`nuxt`) | Nuxt (`ModuleContainer` と `Renderer`) の準備が整ったとき || `error` | (`error`) | フックの呼び出し時に未処理のエラーが発生したとき || `close` | (`nuxt`) | Nuxt インスタンスが正しく終了したとき || `listen` | (`server, {host, port}`) | Nuxt 内部のサーバーがリッスンを始めたとき (`nuxt start` または `nuxt dev` を使用時に発生します) |

[Go to TOC](#)

Renderer クラス

このクラスは、すべての SSR とアセットのリクエストを処理して提供する connect ミドルウェアをエクスポートしています。

-
- ソース: [vue-renderer/renderer.js](#)

フック

特定のライフサイクルイベントでフックを登録できます。

| フック | 引数 | タイミング | | | | `render:before` | (renderer, options) | Renderer クラスのミドルウェアとリソースを設定する前、メソッドやオプションのオーバーロードに役立ちます。 | | `render:setupMiddleware` | (app) `connect instance` | Nuxt が追加する前のミドルウェアスタックです。カスタムしたサーバーサイドミドルウェアを登録するために使用できます。 | | `render:errorMiddleware` | (app) `connect instance` | Nuxt のエラーミドルウェアを追加する前、Nuxt のミドルウェアを使用する前に独自のミドルウェアを追加することに役立ちます。詳細については [Sentry モジュール](#) を参照してください。 | | `render:resourcesLoaded` | (resources) | レンダラーのリソースがロードされた後に呼び出されます（クライアントマニフェスト、サーバーバンドルなど）。 | | `render:done` | (renderer) | SSR ミドルウェアとすべてのリソースの準備がおわったとき（Renderer の準備完了） | | `render:routeContext` | (context.nuxt) | `render:route` フックの前に route が サーバーレンダリングされるたび。Nuxt コンテキストを `window.__NUXT__` にシリアル化する前に呼び出され、クライアントサイドでフェッチできるデータを追加することに役立ちます。 | | `render:route` | (url, result, context) | route が サーバーレンダリングされるたび。リクエストをブラウザに送り返す前に呼び出されます。 | | `render:routeDone` | (url, result, context) | route が サーバーレンダリングされるたび。レスポンスがブラウザに送られた後に呼び出されます。 |

[Go to TOC](#)

ModuleContainer クラス

- ソース: [core/module.js](#)

すべてのモジュールは `ModuleContainer` インスタンスのコンテキストの中で呼ばれます。

Tapable プラグイン

特定のライフサイクルイベントでフックを登録できます。

```
nuxt.moduleContainer.plugin('ready', async moduleContainer => {
  // すべてのモジュールの準備ができたらこの処理を実行します
})
```

モジュールコンテキストの中では代わりに以下のようにできます：

```
this.plugin('ready', async moduleContainer => {
  // すべてのモジュールの準備ができたらこの処理を実行します
})
```

| プラグイン | 引数 | タイミング | | | `ready` | `moduleContainer` | `nuxt.config.js` | にあるすべてのモジュールが初期化されたとき |

メソッド

addVendor (vendor)

`vendor` は使われていないので非推奨です

`options.build.vendor` に追加し、一意なフィルタを適用します。

addTemplate (template)

- template:** `String` または `Object`

- `src`
- `options`
- `fileName`

プロジェクトの `buildDir` (`.nuxt`) ヘビルド中に、[lodash template](#) を使って与えられたテンプレートをレンダリングします。

`fileName` を与えないか `template` が文字列の場合、ターゲットのファイル名はデフォルトで `[dirName].[fileName].[pathHash].[ext]` になります。

このメソッドは最終的な `{ dst, src, options }` オブジェクトを返します。

addPlugin (template)

- **template:** オブジェクトプロパティ (`src`、`options`、`fileName`、`mode`)。

`addTemplate` を使ってプラグインを登録し、`plugins` 配列の先頭に追加します。

```
this.addPlugin({
  src: path.resolve(__dirname, 'templates/foo.js'),
  fileName: 'foo.server.js' // [optional] はサーバーバンドルにのみ含まれます
  options: moduleOptions
})
```

注意: プラグインをクライアントサイドまたはサーバーサイドでのみ使う場合は `mode` または `.client` と `.server` 修飾子を `fileName` オプションと共に使えます（利用可能なオプションについては `plugins` を参照してください）。

`fileName` を指定する場合、`fileName` にカスタムパスを設定できます。名前の衝突を防ぐため `.nuxt` フォルダ内のフォルダ構成を選択できます：

```
{
  fileName: path.join('folder', 'foo.client.js'), // 結果は
  '.nuxt/folder/foo.client.js` になります
}
```

addServerMiddleware (middleware)

`options.serverMiddleware` にミドルウェアをプッシュします。

extendBuild (fn)

`options.build.extend` 関数をつなげることで webpack のビルド設定を簡単に拡張できます。

extendRoutes (fn)

`options.build.extendRoutes` 関数をつなげることでルートを簡単に拡張できます。

addModule (moduleOpts, requireOnce)

非同期関数

モジュールを登録します。`moduleOpts` は文字列または配列 (`[src, options]`) です。`requireOnce` が `true` で解決されたモジュールが `meta` をエクスポートしている場合に、同じモジュールが二度登録されるのを回避します。

requireModule (moduleOpts)

非同期関数

`addModule(moduleOpts, true)` の短縮形です。

フック

特定のライフサイクルイベントでフックを登録できます。

| フック | 引数 | タイミング | | | | `modules:before` | (`moduleContainer, options`) | ModuleContainer クラスが作
られる前に呼ばれ、メソッドとオプションのオーバーロードに役立ちます。 | | `modules:done` | (`moduleContainer`) |
すべてのモジュールがロードされたときに呼ばれます。 |

Builder クラス

- ソース: [builder/builder.js](#)

フック

特定のライフサイクルイベントにフックを登録できます。

```
// ビルド用にフックを追加
this.nuxt.hook('build:done', (builder) => {
  ...
})
```

| フック | 引数 | タイミング | | | build:before | (nuxt, buildOptions) | Nuxt のビルド開始前 ||
builder:prepared | (nuxt, buildOptions) | ビルドディレクトリが作成された時 | | builder:extendPlugins |
(plugins) | プラグイン生成時 | | build:templates | ({ templatesFiles, templateVars, resolve }) | .nuxt テンプ
レートファイル生成時 | | build:extendRoutes | (routes, resolve) | ルーティング生成時 | | webpack:config |
(webpackConfigs) | コンパイラの設定前 | | build:compile | ({ name, compiler }) | webpack コンパイル前
(コンパイラは webpack Compiler インスタンス) でもし universal モードの場合 'client' と 'server' の名前で二
度呼び出されます。 | | build:compiled | ({ name, compiler, stats }) | webpack のビルド終了時 | |
build:done | (nuxt) | Nuxt のビルド終了時 |

[Go to TOC](#)

Generator クラス

-
- ソース: [generator/generator.js](#)

フック

`generate:` フック:

| フック | 引数 | タイミング | | | | `generate:before` | (generator, generateOptions) | 生成前のフック | |
`generate:distRemoved` | (generator) | ビルド先のフォルダが削除されるときのフック | | `generate:distCopied` |
(generator) | 静的ファイルとビルドされたファイルがコピーされるときのフック | | `generate:route` | ({ route,
setPayload }) | ページ生成前のフック。動的ペイロードに便利です。#7422 を参照してください。Nuxt v2.13 以上で
利用可能 | | `generate:page` | ({ route, path, html }) | ユーザーが生成後のパスと HTML を更新するときのフック | |
`generate:routeCreated` | ({ route, path, errors }) | 生成されたページの保存に成功したときのフック | |
`generate:extendRoutes` | (routes) | ユーザーが生成するルートを更新をするとときのフック | | `generate:routeFailed`
| ({ route, errors }) | 生成されたページの保存に失敗したときのフック | | `generate:done` | (generator, errors) |
生成完了時のフック |

Nuxt をプログラムで使う

Nuxt をプログラムで使用することで、ミドルウェアとして利用することができ、Web アプリケーションをレンダリングするための独自のサーバーを自由に作成することができます。

あなた自身のサーバーと共にミドルウェアや API を使いたい場合もあるかもしれません。そういった場合に Nuxt をプログラムから利用することができます。

Nuxt コンストラクタ

Nuxt に渡すことができるオプション一覧を見るには、設定のセクションを参照してください。

```
const { loadNuxt, build } = require('nuxt')

// Nuxt を開発モードで実行する必要があるかどうかをチェックします
const isDev = process.env.NODE_ENV !== 'production'

// Nuxt のインスタンスを取得します
const nuxt = await loadNuxt(isDev ? 'dev' : 'start')

// 開発モードの場合にライブビルドとライブリローディングを有効化します
if (isDev) {
  build(nuxt)
}

// `nuxt.render(req, res)` または `nuxt.renderRoute(route, context)` が使えます
```

手っ取り早く始めるために [nuxt-express](#) や [adonuxt](#) スターターを参照できます。

デバッグログ

Nuxt のログを表示したいときはファイルの始めに次のコードを追加してください：

```
process.env.DEBUG = 'nuxt:*
```

[Go to TOC](#)

Fetch フック

`fetch` フックは、非同期にデータを取得するためのものです。ルートをレンダリングする際にはサーバーサイドで、遷移する際にはクライアントサイドで呼び出されます。

Nuxt >= 2.12

Nuxt v2.12 から、すべての Vue コンポーネントに対して `fetch` という新しいフックが導入されています。非同期データを取得する必要があるたびに `fetch` を使います。`fetch` はサーバーサイドではルートをレンダリングするときに、クライアントサイドでは遷移するときに呼び出されます。

コンポーネントレベルで `$fetchState` を公開します：

- `$fetchState.pending` : Boolean。`fetch` が クライアントサイドで 呼び出されているときにプレースホルダーを表示する用途などで使うことができます。
- `$fetchState.error` : null または Error。エラーメッセージを表示する用途で使うことができます。
- `$fetchState.timestamp` : Integer。最後に `fetch` したタイムスタンプです。`keep-alive` を用いたキャッシングに便利です。

テンプレートから `fetch` フックを呼び出したい場合は、以下のようにします：

```
<button @click="$fetch">Refresh</button>
```

また、コンポーネントメソッドでは下記のとおりです：

```
// スクリプトセクションのコンポーネントメソッドより
export default {
  methods: {
    refresh() {
      this.$fetch()
    }
  }
}
```

`fetch` フック内では `this.$nuxt.context` を使用して、Nuxt `context` にアクセスできます。

オプション

- `fetchOnServer` : Boolean または Function (デフォルト: `true`)。サーバーがページをレンダリングする際に `fetch()` を呼び出します。
- `fetchKey` : String または Function (デフォルトはコンポーネントスコープ ID またはコンポート名) でこのコンポーネントの取得結果を識別するキー (または一意のキーを生成する関数) です (Nuxt v2.15 以上で利用可能)。詳細は PR を参照してください。
- `fetchDelay` : Integer (デフォルト: `200`)。最小実行時間をミリ秒単位で設定します (極短期間での画面切り替えによるちらつきを防ぐため)。

`fetchOnServer` が偽の値をとり得る場合(`false` または、`false` になりえる値を返す)、`fetch` はクライアントサイドでのみ呼び出され、サーバーでコンポーネントをレンダリングする際には `$fetchState.pending` は `true` を返します。

```
<script>
  export default {
    data() {
      return {
        posts: []
      }
    },
    async fetch() {
      this.posts = await this.$http.$get('https://api.nuxtjs.dev/posts')
    },
    fetchOnServer: false,
    // 複数のコンポーネントが同じ `fetchKey` を返却することができ、Nuxt は両方を別々に追跡します。
    fetchKey: 'site-sidebar',
    // あるいは、より詳細に制御するためにコンポーネントインスタンスにアクセスして関数を渡すことができます。
    // 関数は `created` で呼び出され、取得したデータに依存してはいけません。
    fetchKey(getCounter) {
      // getCounter は呼び出し可能なメソッドで、一意な fetchKey の生成結果として
      // シーケンス内の次の番号を取得できます。
      return this.someOtherData + getCounter('sidebar')
    }
  }
</script>
```

Next

fetch フックチェックアウトの詳細については[データの取得](#)のドキュメントを参照してください

watchQuery プロパティ

クエリ文字列を監視し、変更時にコンポーネントメソッドを実行します (asyncData、fetch、validate、layout など)

- 型: `Boolean` または `Array` または `Function` (デフォルト: `□`)

`watchQuery` キーを設定し、監視するクエリ文字列を設定します。定義した文字列に変更が生じると、全てのコンポーネントメソッド (asyncData、fetch、validate、layout など) が呼ばれます。パフォーマンス向上のため、監視はデフォルトで無効になっています。

すべてのクエリ文字列に対して監視を設定したい場合は `watchQuery: true` を設定してください。

```
export default {
  watchQuery: ['page']
}
```

より洗練された監視のために `watchQuery(newQuery, oldQuery)` 関数を使用することもできます。

```
export default {
  watchQuery(newQuery, oldQuery) {
    // 古いクエリ文字列に `bar` が含まれている場合コンポーネントメソッドのみを実行し、
    // そして、新しいクエリ文字列には `foo` が含まれる
    return newQuery.foo && oldQuery.bar
  }
}
```

Warning

警告: v2.12 で導入された新しい `fetch` フックは `watchQuery` の影響を受けません。詳しくは[クエリ文字列の変化のリスニング](#)を参照してください。

Head メソッド

Nuxt はアプリケーションの `headers` 及び `html attributes` を更新するために `vue-meta` を使います。

- 型: `Object` または `Function`

現在のページの HTML の `head` タグを設定するために `head` メソッドを使います。

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

<script>
  export default {
    data() {
      return {
        title: 'Hello World!'
      }
    },
    head() {
      return {
        title: this.title,
        meta: [
          // hid は一意の識別子として使用されます。`vmid` は動作しないので使わないでください。
          {
            hid: 'description',
            name: 'description',
            content: 'My custom description'
          }
        ]
      }
    }
  }
</script>
```

Info

子コンポーネントで使用するときにメタタグが重複しないようにするために、メタ要素の `hid` キーを使って一意の識別子を設定します（[詳細はこちら](#)）。

[Go to TOC](#)

key プロパティ

内部の `<router-view>` コンポーネントに `key` プロパティを設定します

- 型: `String` または `Function`

`key` プロパティは `<router-view>` に渡され、動的ページでのルートトランジションに使われます。`key` が異なる場合はページコンポーネントが再レンダリングされます。

`key` を設定する方法は他もあります。詳細については [nuxt コンポーネント](#) の `nuxtChildKey` プロパティを参照してください。

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

layout プロパティ

layouts ディレクトリの（第一階層の）ファイルはカスタムレイアウトになります。これらはページコンポーネントの layout プロパティで指定して利用できます

- 型: `String` または `Function` (デフォルト: `'default'`)

どのレイアウトを使うか指定するために、ページコンポーネントで `layout` キーを使ってください： Use the `layout` key in your pages components to define which layout to use:

```
export default {
  layout: 'blog',
  // または
  layout(context) {
    return 'blog'
  }
}
```

[Go to TOC](#)

loading プロパティ

loading プロパティは特定のページに対してデフォルトの loading プログレスバーを無効にするオプションを提供します。

- 型: `Boolean` (デフォルト: `true`)

デフォルトでは、Nuxt はルート間の遷移の際に自身で持っているコンポーネントでプログレスバーを表示します。

[loading オプションの設定](#) でグローバルに無効化やカスタマイズできるだけでなく、特定のページに対しても `loading` プロパティを `false` に設定することで無効化できます：

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

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

middleware プロパティ

アプリケーションの特定のページにミドルウェアを設定します。

- 型: `String` または `Array` または `Function`
 - 要素: `String` または `Function`

名前付きミドルウェア

`middleware/` ディレクトリ内にファイルを作成することで名前付きミドルウェアを作成できます。ファイル名がミドルウェア名になります。

```
export default function ({ store, redirect }) {
  // ユーザーが認証されていない場合
  if (!store.state.authenticated) {
    return redirect('/login')
  }
}
```

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

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

無名ミドルウェア

特定のページにだけミドルウェアを使用する必要がある場合は、関数（もしくは関数の配列）を直接使用できます：

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

<script>
  export default {
    middleware({ store, redirect }) {
      // ユーザーが認証されていない場合
      if (!store.state.authenticated) {
        return redirect('/login')
      }
    }
  }
</script>
```

[Go to TOC](#)

page transition プロパティ

Nuxt では `<transition>` コンポーネントを使って、ページ間を遷移する際のトランジション/アニメーションの作成することができます。

- 型: `String` または `Object` または `Function`

特定のルートに対してカスタムトランジションを設定するには、ページコンポーネントに `transition` キーを追加してください。

```
export default {
  // 文字列を指定できます
  transition: ''
  // またはオブジェクト
  transition: {}
  // または関数
  transition (to, from) {}
```

文字列

もし `transition` キーが文字列で設定された場合は `transition.name` として用いられます。

```
export default {
  transition: 'test'
```

Nuxt はこれらの設定を使ってコンポーネントを以下のように設定します：

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

オブジェクト

もし `transition` キーがオブジェクトで設定された場合：

```
export default {
  transition: {
    name: 'test',
    mode: 'out-in'
  }
```

Nuxt はこれらの設定を使ってコンポーネントを以下のように設定します：

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

`transition` オブジェクトが持てるプロパティは以下の通りです：

| キー | 型 | デフォルト | 定義 | | | | name | String | "page" | すべてのルートトランジションに適用されるトランジション名です。 | | mode | String | "out-in" | すべてのルートに適用されるトランジションモードです。詳細は [Vue.js のドキュメント](#) を参照してください。 | | css | Boolean | true | CSS トランジションクラスを適用するかどうかを設定します。デフォルトは true です、もし false に設定した場合コンポーネントのイベント経由で登録された JavaScript フックのみがトリガーとなります。 | | duration | Integer | n/a | トランジションが適用される時間（ミリ秒）です。 [Vue.js のドキュメント](#) を参照してください。 | | type | String | n/a | トランジション終了のタイミングを判定するために待ち受けるトランジションのイベントタイプを指定します。指定可能な値は "transition" または "animation" です。デフォルトではより期間の長いほうのタイプが自動的に検出されます。 | | enterClass | String | n/a | トランジションクラスの開始状態です。 [Vue.js のドキュメント](#) を参照してください。 | | enterToClass | String | n/a | トランジションの終了状態です。 [Vue.js のドキュメント](#) を参照してください。 | | enterActiveClass | String | n/a | トランジション中に適用されるクラスです。 [Vue.js のドキュメント](#) を参照してください。 | | leaveClass | String | n/a | トランジションクラスの開始状態です。 [Vue.js のドキュメント](#) を参照してください。 | | leaveToClass | String | n/a | トランジションの終了状態です。 [Vue.js のドキュメント](#) を参照してください。 | | leaveActiveClass | String | n/a | トランジション中に適用されるクラスです。 [Vue.js のドキュメント](#) を参照してください。 |

page transition プロパティでメソッドを定義することもできます。これらは [JavaScript フック用](#) です：

- 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)
    }
  }
}
```

注意: *JavaScript* のみのトランジションに css: false を明示的に追加して *Vue* が *CSS* の検出をスキップできるようにすることもいい考えです。これにより *CSS* ルールが誤ってトランジションに干渉するのを防ぎます。

トランジションモード

ページ用のデフォルトのトランジションモードは [Vue.js のデフォルトモード](#) とは異なります。 transition モードはデフォルトで out-in が設定されます。もしトランジションの開始と終了を同時に実行したい場合モードを空文字 mode: '' に設定する必要があります。

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

関数

もし `transition` キーが関数で設定された場合：

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

トランジションは各ページ遷移時に適用されます：

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

scrollToTop プロパティ

`scrollToTop` プロパティで、ページをレンダリングする前にトップまでスクロールか否かを指定できます。

- 型: `Boolean` (デフォルト: `false`)

別のページへ遷移する際にトップまでスクロールしますが、子ルートがあるときはスクロール位置をキープする、というのが Nuxt のデフォルトの挙動です。子ルートをレンダリングするときにトップまでスクロールさせたいときは `scrollToTop` を `true` に設定してください：

```
<template>
  <h1>My child component</h1>
</template>

<script>
  export default {
    scrollToTop: true
  }
</script>
```

逆に、親ルートは手動で `scrollToTop` を `false` に設定することもできます。

スクロールについて Nuxt のデフォルトの挙動を上書きしたいときは [scrollBehavior オプション](#)を参照してください。

[Go to TOC](#)

validate メソッド

Nuxt では動的なルーティングを行うコンポーネント内でバリデーションメソッドを定義できます。

- 型: `Function` または `Async Function`

`validate` は新しいルートに移動する前に毎回呼び出されます。サーバーサイドでは（Nuxt アプリケーションへの最初のリクエストで）1 度、クライアントサイドでは別のルートに遷移する際に呼び出されます。このメソッドは `context` オブジェクトを引数として受け取ります。

```
validate({ params, query, store }) {
  return true // params が有効な場合
  return false // Nuxt にルートをレンダリングさせるのを止め、エラーページを表示させます
}
```

```
async validate({ params, query, store }) {
  // 非同期操作
  return true // params が有効な場合
  return false // Nuxt にルートをレンダリングさせるのを止め、エラーページを表示させます
}
```

プロミスを返すこともできます：

```
validate({ params, query, store }) {
  return new Promise((resolve) => setTimeout(() => resolve()))
}
```

Nuxt では動的なルーティングを行うコンポーネント内でバリデーションメソッドを定義できます（以下の例では `pages/users/_id.vue` です）。

バリデーションメソッドが `true` を返さない場合 Nuxt は自動的に 404 エラーページをロードします。

```
export default {
  validate({ params }) {
    // Must be a number
    return /^[^d+$/.test(params.id)
  }
}
```

例えばストア内的一部のデータ（アクションの前に `nuxtServerInit` で格納されます）を確認することもできます：

```
export default {
  validate({ params, store }) {
    // `params.id` が存在しているカテゴリかどうかを確認します
    return store.state.categories.some(category => category.id === params.id)
  }
}
```

バリデーション関数を実行中に想定されるエラーや想定外のエラーを投げることもできます：

```
export default {
  async validate({ params, store }) {
    // カスタムメッセージと共に 500 internal server error を投げます
    throw new Error('Under Construction!')
  }
}
```

Colophon

This book is created by using the following sources:

- Nuxt - 日本語
- GitHub source: [nuxt/nuxtjs.org](https://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>