vuepress-next

start vuepress-next

Get Started | Introduction

Writing a Theme

> **TIP**
>
> Before reading this guide, you'd better learn the guide of Writing a Plugin first.

Create a Theme

A VuePress theme is a special plugin, which should satisfy the Theme API. Like plugins, a theme should also be a *Theme Object* or a *Theme Function*, and could be wrapped with a function to receive options:

```js
const { path } = require("@vuepress/utils");

const fooTheme = (options) => {
  return {
    name: "vuepress-theme-foo",
    layouts: {
      Layout: path.resolve(__dirname, "layouts/Layout.vue"),
      404: path.resolve(__dirname, "layouts/404.vue"),
    },
    // ...
  };
};

const barTheme = (options) => {
  return (app) => {
    return {
      name: "vuepress-theme-bar",
      layouts: {
        Layout: path.resolve(__dirname, "layouts/Layout.vue"),
        404: path.resolve(__dirname, "layouts/404.vue"),
      },
      // ...
    };
  };
};
```

The `layouts` field declares the layouts provided by your theme. A theme must provide at least two layouts: `Layout` and `404`. The former is to provide default layout for common pages, while the latter is to provide layout for 404 page.

The `Layout` layout should contain the **Content** component to display the markdown content:

```vue
<template>
  <div>
    <Content />
  </div>
</template>
```

The `404` layout will be used for the `404.html` page:

```vue
<template>
  <div>404 Not Found</div>
</template>
```

You can provide more layouts, and users can change layout via **layout** frontmatter.

Publish to NPM

Also, there are some conventions for theme in **package.json**:

```json
{
  "name": "vuepress-theme-foo",
  "keywords": [
    "vuepress-theme"
  ]
}
```

- Set `name` to follow the naming convention: `vuepress-theme-xxx` or `@org/vuepress-theme-xxx`, which should be consistent with the **name** field of the *Theme Object*.
- Set `keywords` to include `vuepress-theme`, so that users can search your theme on NPM.

Introduction

VuePress is a markdown-centered static site generator. You can write your content (documentations, blogs, etc.) in Markdown, then VuePress will help you to generate a static site to host them.

The purpose of creating VuePress was to support the documentation of Vue.js and its sub-projects, but now it has been helping a large amount of users to build their documentation, blogs, and other static sites.

How It Works

A VuePress site is in fact a single-page application (SPA) powered by Vue and Vue Router.

Routes are generated according to the relative path of your markdown files. Each Markdown file is compiled into HTML with markdown-it and then processed as the template of a Vue component. This allows you to directly use Vue inside your Markdown files and is great when you need to embed dynamic content.

During development, we start a normal dev-server, and serve the VuePress site as a normal SPA. If you've used Vue before, you will notice the familiar development experience when you are writing and developing with VuePress.

During build, we create a server-rendered version of the VuePress site and render the corresponding HTML by virtually visiting each route. This approach is inspired by Nuxt's `nuxt generate` command and other projects like Gatsby.

Why Not ...?

Nuxt

Nuxt is an outstanding Vue SSR framework, and it is capable of doing what VuePress does. But Nuxt is designed for building applications, while VuePress is more lightweight and focused on content-centric static sites.

VitePress

VitePress is the little brother of VuePress. It's also created and maintained by our Vue.js team. It's even more lightweight and faster than VuePress. However, as a tradeoff, it's more opinionated and less configurable. For example, it does not support plugins. But VitePress is powerful enough to make your content online if you don't need advanced customizations.

It might not be an appropriate comparison, but you can take VuePress and VitePress as Laravel and Lumen.

Docsify / Docute

Both are great projects and also Vue-powered. Except they are both fully runtime-driven and therefore not SEO-friendly. If you don’t care for SEO and don’t want to mess with installing dependencies, these are still great choices.

Hexo

Hexo has been serving the Vue 2.x docs well. The biggest problem is that its theming system is static and string-based - we want to take advantage of Vue for both the layout and the interactivity. Also, Hexo’s Markdown rendering isn’t the most flexible to configure.

GitBook

We’ve been using GitBook for most of our sub project docs. The primary problem with GitBook is that its development reload performance is intolerable with a large amount of files. The default theme also has a pretty limiting navigation structure, and the theming system is, again, not Vue based. The team behind GitBook is also more focused on turning it into a commercial product rather than an open-source tool.

Bundler

VuePress has been using Webpack as the bundler to dev and build sites. Since VuePress v2, other bundlers are also supported, and now we are using Vite as the default bundler. Of course, you can still choose to use Webpack.

Choose a Bundler

When using the vuepress package, Vite bundler is installed and used automatically.

If you want to use Webpack bundler instead, you can remove it and install the vuepress-webpack package instead:

YARN NPM

```sh
yarn remove vuepress
yarn add -D vuepress-webpack@next
```

> **TIP**
>
> In fact, the vuepress package is just a wrapper of the vuepress-vite package.

Configure Bundler

Generally, you could use a bundler without extra configuration, because we have already configured them properly to work with VuePress.

You can configure bundler for advanced usage via the bundler option:

```js
const { viteBundler } = require("vuepress");
// const { webpackBundler } = require('vuepress-webpack')

module.exports = {
  bundler: viteBundler({
    vuePluginOptions: {
      template: {
```

```
        compilerOptions: {
          isCustomElement: tag => tag === "center",
        },
      },
    },
  }),
};
```

You can refer to Bundlers > Vite and Bundlers > Webpack to check out all options of the corresponding bundler.

开发主题

> **TIP**
>
> 在阅读该指南之前，你最好先了解一下 **开发插件** 指南。

创建一个主题

VuePress 主题是一个特殊的插件，它应该符合 **主题 API** 。和插件一样，主题可以是一个 *主题对象* 或一个 *主题函数* ，并且通常通过一个函数来接收配置项：

```js
const { path } = require("@vuepress/utils");

const fooTheme = (options) => {
  return {
    name: "vuepress-theme-foo",
    layouts: {
      Layout: path.resolve(__dirname, "layouts/Layout.vue"),
      404: path.resolve(__dirname, "layouts/404.vue"),
    },
    // ...
  };
};

const barTheme = (options) => {
  return (app) => {
    return {
      name: "vuepress-theme-bar",
      layouts: {
        Layout: path.resolve(__dirname, "layouts/Layout.vue"),
        404: path.resolve(__dirname, "layouts/404.vue"),
      },
      // ...
    };
  };
};
```

`layouts` 字段声明了你的主题提供的布局。一个主题必须提供至少两个布局：`Layout` 和 `404`。前者用于提供一般页面的默认布局，后者用于提供 404 页面的布局。

`Layout` 布局应该包含 Content 组件来展示 Markdown 内容：

vue

```vue
<template>
  <div>
    <Content />
  </div>
</template>
```

`404` 布局会被用于 `404.html` 页面：

vue

```vue
<template>
  <div>404 Not Found</div>
</template>
```

你可以提供多个布局，用户可以通过 layout Frontmatter 来修改布局。

发布到 NPM

同样的，对于主题也有 package.json 相关的约定：

json

```json
{
  "name": "vuepress-theme-foo",
  "keywords": [
    "vuepress-theme"
  ]
}
```

- 将 `name` 按照约定命名：`vuepress-theme-xxx` 或 `@org/vuepress-theme-xxx`，它应该和 *主题对象* 的 name 字段保持一致。
- 在 `keywords` 中包含 `vuepress-theme`，这样用户可以在 NPM 上搜索到你的主题。

静态资源

相对路径

你可以在你的 Markdown 内容中使用相对路径来引用静态资源：

```md
![图片](./image.png)
```

一般情况下，我们推荐你使用这种方式来引用图片，因为人们通常会把图片放在引用它的 Markdown 文件附近。

Public 文件

你可以把一些静态资源放在 Public 目录中，它们会被复制到最终生成的网站的根目录下。

默认的 Public 目录是 `.vuepress/public` ，可以通过 public 配置项来修改。

在下列这些情况中，你可能会用到它：

- 你可能需要提供一些静态资源，但是它们并不直接被你的 Markdown 文件引用，比如 favicon 和 PWA 图标。
- 你可能想要托管一些共享的静态资源，甚至可能需要在你的网站外部引用它，比如 Logo 图片。
- 你可能想在你的 Markdown 内容中通过绝对路径来引入图片。

以我们文档的源文件为例，我们把 VuePress 的 Logo 放在了 Public 目录下：

```sh
└─ docs
   ├─ .vuepress
   |  └─ public
   |     └─ images
   |        └─ hero.png  # <- Logo 文件
   └─ guide
      └─ assets.md       # <- 我们在这里
```

我们可以这样在当前页面引用 Logo：

Input

```md
1 ![VuePress Logo](/images/hero.png)
```

Output

Base Helper

如果你的网站部署在非根路径下，例如 `https://foo.github.io/bar/` ，那么你应该把 base 设置为 `'/bar/'` 。显然，此时你的 Public 文件会被部署在 `https://foo.github.io/bar/images/hero.png` 这样的链接下。

在大多数情况下，你不需要担心这些 Public 文件的引用路径，因为 VuePress 会自动帮你处理 `base` 前缀：

```md
1 <!-- 你不需要给 `/images/hero.png` 手动添加 `/bar/` 前缀 -->
2 ![VuePress Logo](/images/hero.png)
```

然而，有些情况下，你可能会有一些指向 Public 文件的动态路径，尤其是在你开发一个自定义主题的时候。在这种情况下， `base` 无法被自动处理。为了解决这个问题，VuePress 提供了 withBase 工具函数，它可以帮助你添加 `base` 前缀：

```vue
<template>
  <img :src="withBase(logoPath)">
</template>

<script setup>
import { ref } from 'vue'
import { withBase } from '@vuepress/client'

const logoPath = ref('/images/hero.png')
</script>
```

你也可以通过 `$withBase` 来直接使用这个工具函数：

```md
<img :src="$withBase('/images/hero.png')" alt="VuePress Logo">
```

依赖包和路径别名

尽管这不是常见用法，但是你可以从依赖包中引用图片：

```sh
npm install -D package-name
```

```md
![来自依赖包的图片](package-name/image.png)
```

在配置文件中设置的路径别名也同样支持：

```js
module.exports = {
  alias: {
    "@alias": path.resolve(__dirname, "./path/to/some/dir"),
  },
};
```

```md
![来自路径别名的图片](@alias/image.png)
```

TIP

配置参考： alias

404

> There's nothing here.

Take me home