Routing

Docusaurus' routing system follows single-page application conventions: one route, one component. In this section, we will begin by talking about routing within the three content plugins (docs, blog, and pages), and then go beyond to talk about the underlying routing system.

Routing in content plugins

Every content plugin provides a `routeBasePath` option. It defines where the plugins append their routes to. By default, the docs plugin puts its routes under `/docs`; the blog plugin, `/blog`; and the pages plugin, `/`. You can think about the route structure like this:

Any route will be matched against this nested route config until a good match is found. For example, when given a route `/docs/configuration`, Docusaurus first enters the `/docs` branch, and then searches among the subroutes created by the docs plugin.

Changing `routeBasePath` can effectively alter your site's route structure. For example, in Docs-only mode, we mentioned that configuring `routeBasePath: '/'` for docs means that all routes that the docs plugin create would not have the `/docs` prefix, yet it doesn't prevent you from having more subroutes like `/blog` created by other plugins.

Next, let's look at how the three plugins structure their own "boxes of subroutes".

Pages routing

Pages routing are straightforward: the file paths directly map to URLs, without any other way to customize. See the pages docs for more information.

The component used for Markdown pages is `@theme/MDXPage`. React pages are directly used as the route's component.

Blog routing

The blog creates the following routes:

- **Posts list pages**: `/`, `/page/2`, `/page/3`...
  - The component is `@theme/BlogListPage`.
- **Post pages**: `/2021/11/21/algolia-docsearch-migration`, `/2021/05/12/announcing-docusaurus-two-beta`...
  - Generated from each Markdown post.
  - The routes are fully customizable through the `slug` front matter.
  - The component is `@theme/BlogPostPage`.
- **Tags list page**: `/tags`
  - The route is customizable through the `tagsBasePath` option.
  - The component is `@theme/BlogTagsListPage`.
- **Tag pages**: `/tags/adoption`, `/tags/beta`...
  - Generated through the tags defined in each post's front matter.
  - The routes always have base defined in `tagsBasePath`, but the subroutes are customizable through the tag's `permalink` field.
  - The component is `@theme/BlogTagsPostsPage`.
- **Archive page**: `/archive`
  - The route is customizable through the `archiveBasePath` option.
  - The component is `@theme/BlogArchivePage`.

Docs routing

The docs is the only plugin that creates **nested routes**. At the top, it registers **version paths**: `/`, `/next`, `/2.0.0-beta.13`... which provide the version context, including the layout and sidebar. This ensures that when switching between individual docs, the sidebar's state is preserved, and that you can switch between versions through the navbar dropdown while staying on the same doc. The component used is `@theme/DocPage`.

The individual docs are rendered in the remaining space after the navbar, footer, sidebar, etc. have all been provided by the `DocPage` component. For example, this page, `/docs/advanced/routing`, is generated from the file at `./versioned_docs/version-2.4.0/advanced/routing.md`. The component used is `@theme/DocItem`.

The doc's `slug` front matter customizes the last part of the route, but the base route is always defined by the plugin's `routeBasePath` and the version's `path`.

File paths and URL paths

Throughout the documentation, we always try to be unambiguous about whether we are talking about file paths or URL paths. Content plugins usually map file paths directly to URL paths, for example, `./docs/advanced/routing.md` will become `/docs/advanced/routing`. However, with `slug`, you can make URLs totally decoupled from the file structure.

When writing links in Markdown, you could either mean a *file path*, or a *URL path*, which Docusaurus would use several heuristics to determine.

- If the path has a `@site` prefix, it is *always* an asset file path.
- If the path has an `http(s)://` prefix, it is *always* a URL path.
- If the path doesn't have an extension, it is a URL path. For example, a link `[page](../plugins)` on a page with URL `/docs/advanced/routing` will link to `/docs/plugins`. Docusaurus will only detect broken links when building your site (when it knows the full route structure), but will make no assumptions about the existence of a file. It is exactly equivalent to writing `<a href="../plugins">page</a>` in a JSX file.
- If the path has an `.md(x)` extension, Docusaurus would try to resolve that Markdown file to a URL, and replace the file path with a URL path.
- If the path has any other extension, Docusaurus would treat it as an asset and bundle it.

The following directory structure may help you visualize this file → URL mapping. Assume that there's no slug customization in any page.

- A sample site structure

So much about content plugins. Let's take one step back and talk about how routing works in a Docusaurus app in general.

Routes become HTML files

Because Docusaurus is a server-side rendering framework, all routes generated will be server-side rendered into static HTML files. If you are familiar with the behavior of HTTP servers like Apache2, you will understand how this is done: when the browser sends a request to the route `/docs/advanced/routing`, the server interprets that as request for the HTML file `/docs/advanced/routing/index.html`, and returns that.

The `/docs/advanced/routing` route can correspond to either `/docs/advanced/routing/index.html` or `/docs/advanced/routing.html`. Some hosting providers differentiate between them using the presence of a trailing slash, and may or may not tolerate the other. Read more in the trailing slash guide.

For example, the build output of the directory above is (ignoring other assets and JS bundle):

- Output of the above workspace

If `trailingSlash` is set to `false`, the build would emit `intro.html` instead of `intro/index.html`.

All HTML files will reference its JS assets using absolute URLs, so in order for the correct assets to be located, you have to configure the `baseUrl` field. Note that `baseUrl` doesn't affect the emitted bundle's file structure: the base URL is one level above the Docusaurus routing system. You can see the aggregate of `url` and `baseUrl` as the actual location of your Docusaurus site.

For example, the emitted HTML would contain links like `<link rel="preload" href="/assets/js/runtime~main.7ed5108a.js" as="script">`. Because absolute URLs are resolved from the host, if the bundle placed under the path `https://example.com/base/`, the link will point to `https://example.com/assets/js/runtime~main.7ed5108a.js`, which is, well, non-existent. By specifying `/base/` as base URL, the link will correctly point to `/base/assets/js/runtime~main.7ed5108a.js`.

Localized sites have the locale as part of the base URL as well. For example, `https://docusaurus.io/zh-CN/docs/advanced/routing/` has base URL `/zh-CN/`.

Generating and accessing routes

The `addRoute` lifecycle action is used to generate routes. It registers a piece of route config to the route tree, giving a route, a component, and props that the component needs. The props and the component are both provided as paths for the bundler to `require`, because as explained in the architecture overview, server and client only communicate through temp files.

All routes are aggregated in `.docusaurus/routes.js`, which you can view with the debug plugin's routes panel.

On the client side, we offer `@docusaurus/router` to access the page's route. `@docusaurus/router` is a re-export of the `react-router-dom` package. For example, you can use `useLocation` to get the current page's location, and `useHistory` to access the history object. (They are not the same as the browser API, although similar in functionality. Refer to the React Router documentation for specific APIs.)

This API is **SSR safe**, as opposed to the browser-only `window.location`.

myComponent.js

```
import React from 'react';
import {useLocation} from '@docusaurus/router';

export function PageRoute() {
  // React router provides the current component's route, even in SSR
  const location = useLocation();
  return (
    <span>
      We are currently on <code>{location.pathname}</code>
    </span>
  );
}
```

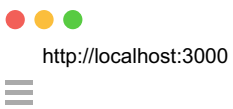
We are currently on `/docs/advanced/routing`

Escaping from SPA redirects

Docusaurus builds a single-page application, where route transitions are done through the `history.push()` method of React router. This operation is done on the client side. However, the prerequisite for a route transition to happen this way is that the target URL is known to our router. Otherwise, the router catches this path and displays a 404 page instead.

If you put some HTML pages under the `static` folder, they will be copied to the build output and therefore become accessible as part of your website, yet it's not part of the Docusaurus route system. We provide a `pathname://` protocol that allows you to redirect to another part of your domain in a non-SPA fashion, as if this route is an external link. Try the following two links:

```
- [/pure-html](/pure-html)
- [pathname:///pure-html](pathname:///pure-html)
```



- /pure-html
- pathname:///pure-html

TIP

The first link will **not** trigger a "broken links detected" check during the production build, because the respective file actually exists. Nevertheless, when you click on the link, a "page not found" will be displayed until you refresh.

The `pathname://` protocol is useful for referencing any content in the static folder. For example, Docusaurus would convert all Markdown static assets to require() calls. You can use `pathname://` to keep it a regular link instead of being hashed by Webpack.

my-doc.md

```
![An image from the static](pathname:///img/docusaurus.png)

[An asset from the static](pathname:///files/asset.pdf)
```

Docusaurus will only strip the `pathname://` prefix without processing the content.

Edit this page

*Last updated on **Mar 23, 2023** by **Sébastien Lorber***

Previous
« Plugins

Next
Static site generation »