# 使用

## 配置

配置 `doom` 文档工具

- 配置文件
- 基础配置
- API 文档配置
- 权限说明文档配置
- 引用文档配置
- 发行说明配置
- 左导航配置
- 内部文档路由配置
- 仅包含文档路由配置
- 语言高亮插件配置
- `sites.yaml` 配置
- 翻译配置
- 在代码仓库编辑文档
- 文档导出配置
- 文档检查配置
- Algolia 搜索配置
- Sitemap 配置

## 约定

基于“约定大于配置”的理念，我们约定文档

- 目录结构
- 元数据
- 排序
- 预览

## 国际化

在可复用组件中使用国际化文本

- `i18n.json`
- `.ts/.tsx`
- `.mdx`

## 权限说明文档

- `props`
- 示例

## Markdown

- Callouts
- Mermaid

## MDX

使用 MDX 可以

- rspress 组件
- doom 组件

## API 文档

- K8S API
- 高级 API

## 部署

文档项目开发完成后我们可以将项目部署到 ACP 平台

- 构建与预览

置

多版本构建

合并目录结构

动态挂载配置文件

# 配置

## 目录


- 配置文件
- 基础配置
- API 文档配置
- 权限说明文档配置
- 引用文档配置
  - `frontmatterMode`
- 发行说明配置
- 左导航配置
- 内部文档路由配置
- 仅包含文档路由配置
- 语言高亮插件配置
- `sites.yaml` 配置
- 翻译配置
- 在代码仓库编辑文档
- 文档导出配置
- 文档检查配置
- Algolia 搜索配置
- Sitemap 配置

# 配置文件

大部分情况下，我们只需要使用静态 `yaml` 配置文件即可，支持 `doom.config.yaml` 或 `doom.config.yml`，对于复杂场景，比如需要动态配置或自定义 `rspress` 插件时，可以使用 `js/ts` 配置文件，支持 `.js/.ts/.mjs/.mts/.cjs/.cts` 多种文件格式。

对于 `js/ts` 配置文件，我们需要导出配置，可以配合 `@alauda/doom/config` 中导出的 `defineConfig` 函数实现类型辅助：

```
import { defineConfig } from '@alauda/doom/config'

export default defineConfig({})
```

# 基础配置

- `lang`：默认文档语言，为方便大部分项目使用，我们默认支持中英文文档，默认语言为 `en`，如果当前文档项目不需要多语言支持，可以将此项配置为 `null` 或 `undefined`
- `title`：文档标题，会显示在浏览器标签页上
- `logo`：文档左上角 logo，支持图片链接、文件路径，绝对路径代表 `public` 目录下的文件，相对路径代表相对于当前工具目录的文件，默认使用 `doom` 包内置的 alauda logo
- `logoText`：文档标题，会显示在左上角的 logo 处
- `icon`：文档 favicon，默认同 `logo`
- `base`：文档基础路径，用于部署到非根路径，如 `product-docs`，默认为 `/`
- `outDir`：构建产物目录，默认为 `dist/{base}/{version}`，如果指定此项，则变更为 `dist/{outDir}/{version}`，其中 `version` 可选，参考多版本构建

# API 文档配置

```yaml
api:
  # CRD 定义文件路径，相对于 doom.config.* 所在目录，支持 glob 匹配，json/yaml 文件
  crds:
    - docs/shared/crds/*.yaml
  # OpenAPI 定义文件路径，相对于 doom.config.* 所在目录，支持 glob 匹配，json/yaml 文件
  openapis:
    - docs/shared/openapis/*.json
  # 渲染 openapi 相关的资源定义时，默认会在页面内联，如果需要将相关联的资源定义单独提取到文件中，可以配置以下选项
  # 参考 https://doom.alauda.cn/apis/references/CodeQuality.html#v1alpha1.CodeQualitySpec
  references:
    v1alpha1.CodeQualityBranch: /apis/references/CodeQualityBranch#v1alpha1.CodeQualityBranch
  # 可选，API 文档路径前缀，如果当前业务使用 gateway 等代理服务，可以配置此项
  pathPrefix: /apis
```

文档编写参考 API 文档

# 权限说明文档配置

```yaml
# 以下资源文件路径，相对于 doom.config.* 所在目录，支持 glob 匹配，json/yaml 文件
permission:
  functionresources:
    # `kubectl get functionresources`
    - docs/shared/functionresources/*.yaml
  roletemplates:
    # `kubectl get roletemplates -l auth.cpaas.io/roletemplate.official=true`
    - docs/shared/roletemplates/*.yaml
```

文档编写参考权限说明文档

# 引用文档配置

```yaml
reference:
  - repo: alauda-public/product-doc-guide # 可选，引用文档仓库地址，如果不填写，则默认使用当前文档仓库地址
    branch: # [string] 可选，引用文档仓库分支
    publicBase: # [string] 可选，使用远程仓库时使用绝对路径 /images/xx.png 对应的静态资源所在目录，默认为 docs/public
    sources:
      - name: anchor # 引用文档名称，用于在文档中引用，全局唯一
        path: docs/index.mdx#介绍 # 引用文档路径，支持锚点定位，远程仓库相对于仓库根目录，本地相对于 doom.config.* 所在目录
        ignoreHeading: # [boolean] 可选，是否忽略标题，如果为 true，则不会在引用文档中显示锚点的标题
        processors: # 可选，引用文档内容处理器
          - type: ejsTemplate
            data: # ejs 模板参数，使用 `<%= data.xx %>` 访问
        frontmatterMode: merge # 可选，引用文档处理 frontmatter 模式，默认为 ignore，可选值为 ignore/merge/replace/remove
```

### `frontmatterMode`

- `ignore`：忽略引用文档的 frontmatter，保留使用当前文档的 frontmatter
- `merge`：合并引用文档的 frontmatter，如果有相同的 key，引用文档的值会覆盖当前文档的值
- `replace`：使用引用文档的 frontmatter 替换当前文档的 frontmatter
- `remove`：移除当前文档的 frontmatter

文档编写参考引用文档

## 发行说明配置

```yaml
releaseNotes:
  queryTemplates:
    fixed: # 可包含 ejs 模板的 jql 语句
    unfixed:
```

release-notes.md

```
<!-- release-notes-for-bugs?template=fixed&project=DevOps -->
```

release-notes.mdx

```
{/* release-notes-for-bugs?template=fixed&project=DevOps */}
```

以上述 `template=fixed&project=DevOps` 为例，`fixed` 为 `queryTemplates` 中定义的模板名称，剩余的 `query` 参数 `project=DevOps` 将作为 `ejs` ↗ 模板参数传递给 `fixed` 模板处理后作为 jira `jql` ↗ 发起 `https://jira.alauda.cn/rest/api/2/search?jql=<jql>` 请求，此 API 要求鉴权，须提供 `JIRA_USERNAME` 和 `JIRA_PASSWORD` 环境变量才能预览生效

## 左导航配置

```
sidebar:
  collapsed: false # 可选，是否默认折叠左导航，默认折叠，文档内容不多时可以考虑设置为 false
```

## 内部文档路由配置

```
internalRoutes: # 可选，支持 glob 匹配，相对于 docs 目录，在 cli 启用 `-i, --ignore` 选项时匹配到的路由/文件会被忽略
  - '*/internal/**'
```

## 仅包含文档路由配置

```yaml
onlyIncludeRoutes: # 可选，支持 glob 匹配，相对于 docs 目录，在 cli 启用 `-i, --ignore` 选项时只有此配置下的路由/文件会被启用，可同时配合 `internalRoutes` 进一步排除其中的部分路由
  - '*/internal/**'
internalRoutes:
  - '*/internal/overview.mdx'
```

## 语言高亮插件配置

```yaml
shiki:
  theme: # optional, https://shiki.style/themes
  langs: # optional, https://shiki.style/languages
  transformers: # optional, only available in js/ts config, https://shiki.style/guide/transformers
```

**WARNING**

未配置的语言将在命令行提示告警，并回退到 `plaintext` 渲染

## `sites.yaml` 配置

`sites.yaml` 配置文件用于配置当前文档站点关联的子站点信息，引用外部站点组件和构建单版本文档时会用到此处定义的信息。

```yaml
- name: connectors # 全站唯一名称
  base: /devops-connectors # 站点访问基础路径
  version: v1.1 # 构建多版本站点时 ExternalSite/ExternalSiteLink 跳转的版本

  displayName: # 站点显示名称，如果不填写或未匹配到语言，则默认使用 name
    en: DevOps Connectors
    zh: DevOps 连接器

  # 以下属性用于构建全站点时拉取镜像，如果不填写则在最终打包完整网站时将忽略此项
  # 一般对子站点引用需要配置相关信息，对父站点引用不需要配置
  repo: https://github.com/AlaudaDevops/connectors-operator # 站点仓库地址，如果是内部 gitlab 仓库，可以直接使用相关 slug，如 `alauda/product-docs`
  image: devops/connectors-docs # 站点构建镜像，用于构建全站点时拉取镜像
```

## 翻译配置

```
translate:
  # 系统提示语，ejs 模板，传入的参数有 `sourceLang`, `targetLang`, `userPrompt`, `additionalPrompts`, `terms`, `titleTranslationPrompt`
  # 其中 `sourceLang` 和 `targetLang` 是 `中文` 和 `英文` 两个字符串，
  #     `userPrompt` 为下述用户全局配置，可能为空
  #     `additionalPrompts` 为文档 `frontmatter.i18n` 中的 `additionalPrompts` 配置，可能为空
  #     `terms` 和 `titleTranslationPrompt` 为根据文档内容包含的术语和标题动态生成的提示语，用于让 AI 根据术语对照表和标题对照表进行翻译，可能为空
  # 默认的系统提示语如下，可以根据实际情况进行修改
  systemPrompt: |
You are a professional technical documentation engineer, skilled in writing high-quality technical documentation in <%= targetLang %>. Please accurately translate the following text from <%= sourceLang %> to <%= targetLang %>, maintaining the style consistent with technical documentation in <%= sourceLang %>.

## Baseline Requirements
- Sentences should be fluent and conform to the expression habits of the <%= targetLang %> language.
- Input format is MDX; output format must also retain the original MDX format. Do not translate the names of jsx components such as <Overview />, and do not wrap output in unnecessary code blocks.
- **CRITICAL**: Do not translate or modify ANY link content in the document. This includes:
  - URLs in markdown links: [text](URL) - keep URL exactly as is
  - Reference-style links: [text][ref] and [ref]: URL - keep both ref and URL unchanged
  - Inline URLs: https://example.com - keep completely unchanged
  - Image links: ![alt](src) - keep src unchanged, but alt text can be translated
  - Anchor links: [text](#anchor) - keep #anchor unchanged
  - Any href attributes in HTML tags - keep unchanged
- Do not translate professional technical terms and proper nouns, including but not limited to: Kubernetes, Docker, CLI, API, REST, GraphQL, JSON, YAML, Git, GitHub, GitLab, AWS, Azure, GCP, Linux, Windows, macOS, Node.js, React, Vue, Angular, TypeScript, JavaScript, Python, Java, Go, Rust, etc. Keep these terms in their original form.
- The title field and description field in frontmatter should be translated, other frontmatter fields should retain and do not translate.
- Content within MDX components needs to be translated, whereas MDX component names and parameter keys do not.
- Do not modify or translate any placeholders in the format of __ANCHOR_N
```

```
__ (where N is a number). These placeholders must be kept exactly as they appear in the source text.
- Keep original escape characters like backslash, angle brackets, etc. unchanged during translation.
- Do not add any escape characters to special characters like [], (), {}, etc. unless they were explicitly present in the source text. For example:
  - If source has "Architecture [Optional]", keep it as "Architecture [Optional]" (not "Architecture \\[Optional]")
  - If source has "Function (param)", keep it as "Function (param)" (not "Function \\(param)")
  - Only add escape characters if they were present in the original text
- Preserve and do not translate the following comments, nor modify their content:
  - {/* release-notes-for-bugs */}
  - <!-- release-notes-for-bugs -->
- Remove and do not retain the following comments:
  - {/* reference-start */}
  - {/* reference-end */}
  - <!-- reference-start -->
  - <!-- reference-end -->
- Ensure the original Markdown format remains intact during translation, such as frontmatter, code blocks, lists, tables, etc.
- Do not translate the content of the code block.
<% if (titleTranslationPrompt) { %>
<%- titleTranslationPrompt %>
<% } %>
<% if (terms) { %>
<%- terms %>
<% } %>

<% if (userPrompt || additionalPrompts) { %>
## Additional Requirements
These are additional requirements for the translation. They should be met along with the baseline requirements, and in case of any conflict, the baseline requirements should take precedence.

The text for translation is provided below, within triple quotes:
"""
<% if (userPrompt) { %>
<%- userPrompt %>
<% } %>

<% if (additionalPrompts) { %>
<%- additionalPrompts %>
```

```
<% } %>
"""
<% } %>
```

## 在代码仓库编辑文档

```
editRepoBaseUrl: alauda/doom/tree/main/docs # https://github.com/ 前缀可以省略，仅当启用 `-R, --edit-repo` 命令行标志符时生效
```

## 文档导出配置

```
export:
  - name: Concepts # 可选，全局唯一 pdf 名称，默认为文档标题
    scope: '*/concepts' # 必填，字符串或数组，文档范围，支持 glob 匹配，相对于 docs 目录
```

## 文档检查配置

```
lint:
  cspellOptions: # 可选，cspell 配置项，参考 https://github.com/streetsidesoftware/cspell/tree/main/packages/cspell-eslint-plugin#options
```

## Algolia 搜索配置

```
algolia: # 可选，Algolia 搜索配置，仅当启用 `-a, --algolia` 命令行标志符时生效
  appId: # Algolia 应用 ID
  apiKey: # Algolia API Key
  indexName: # Algolia 索引名称
```

请使用 `public/robots.txt` 进行 Algolia 爬虫验证

**Info**

由于 `rspress` 当前架构限制，使用 Algolia 搜索功能需通过自定义主题↗ 实现，因此为统一使用相关主题功能，我们提供了 `@alauda/doom/theme` 主题入口，请添加以下主题配置文件启用：

theme/index.ts

```
export * from '@alauda/doom/theme'
```

# Sitemap 配置

```
siteUrl: https://docs.alauda.cn # 可选，站点 URL，用于生成 sitemap，仅当启用 `-S, --site-url` 命令行标志符时生效
```

# 约定

## 目录

目录结构
元数据
排序
预览

## 目录结构

左侧边栏默认基于文件目录结构自动生成，一级目录中 `index` 文件即文档首页，将展示为左导航首项，子文件夹中可以使用 `index.md` 或 `index.mdx` 并定义文档一级标题来设置左侧边栏分组标题，其他子文档将自动归并到当前分组下，嵌套子文件夹也遵循相同规则。

```
├── index.md
├── start.mdx
└── usage
    ├── index.mdx
    └── convention.md
```

同时我们约定

1. `public` 目录用于存放静态资源，如图片、视频等
2. `public/_remotes` 用于存放远程引用文档关联的静态资源，请勿直接依赖该目录的资源，可以将 `*/public/_remotes` 加入 `.gitignore` 避免提交到代码仓库

3. `shared` 目录用于存放公共组件、可复用的文档等，不会自动生成文档数据。

## 元数据

在文档的开头，可以通过 `frontmatter` 来定义文档的元数据，如标题、描述、作者、分类等。

```
---
title: 标题
description: 描述
author: 作者
category: 分类
---
```

在文档正文中，参考 MDX 使用 `.mdx` 文件时可以使用 `frontmatter` 来访问这些元数据。

## 排序

除 `index.md` 或 `index.mdx` 外，其他文档将默认按照文件名排序，可以通过自定义 `frontmatter` 中的 `weight` 值来调整文档在左侧边栏中的排序（`weight` 值越小排序越靠前）。

```
---
weight: 1
---
```

> **Warning**
>
> 注意：目前左导航配置的变更需要重启服务才能生效，一般开发时不用过多关注。

## 预览

有时在分组首页中我们不需要显示特别的内容，这时可以使用 `index.mdx` 文件并使用 `Overview` 组件来展示当前分组的文档列表，将展示分组列表文件的标题、描述和二级标题信息。

```
# 使用

<Overview />
```

效果可以参考使用。

# Markdown

除了标准的 gfm↗ 语法外，Doom 内置了一些额外的 Markdown 扩展功能。

## 目录

Callouts

Mermaid

## Callouts

源码标注组件

**Note**

1. 请根据实际语言使用行内代码注释，如 `;`，`%`，`#`，`//`，`/** */`，`--` 和 `<!-- -->` 等
2. 如果需要将其视为代码注释，请使用 `[\!code callout]` 进行转义
3. 有时，`:::callouts` 由于嵌套缩进导致解析显示异常，可以使用 `<div class="doom-callouts">` 或 `<Callouts>` 组件代替

```sh
Memory overhead per virtual machine ≈ (1.002 × requested memory) \
              + 218 MiB \  # [!code callout]
              + 8 MiB × (number of vCPUs) \  # [!code callout]
              + 16 MiB × (number of graphics devices) \  # [!code callou
t]
              + (additional memory overhead) # [!code callout]
```

:::callouts

1. Required for the processes that run in the `virt-launcher` pod.
2. Number of virtual CPUs requested by the virtual machine.
3. Number of virtual graphics cards requested by the virtual machine.
4. Additional memory overhead:
   - If your environment includes a Single Root I/O Virtualization (SR-IO
V) network device or a Graphics Processing Unit (GPU), allocate 1 GiB add
itional memory overhead for each device.
   - If Secure Encrypted Virtualization (SEV) is enabled, add 256 MiB.
   - If Trusted Platform Module (TPM) is enabled, add 53 MiB.

:::

```sh
Memory overhead per virtual machine ≈ (1.002 × requested memory) \
              + 218 MiB \  1
              + 8 MiB × (number of vCPUs) \  2
              + 16 MiB × (number of graphics devices) \  3
              + (additional memory overhead)  4
```

1. Required for the processes that run in the `virt-launcher` pod.
2. Number of virtual CPUs requested by the virtual machine.
3. Number of virtual graphics cards requested by the virtual machine.
4. Additional memory overhead:

   - If your environment includes a Single Root I/O Virtualization (SR-IOV) network device or a Graphics Processing Unit (GPU), allocate 1 GiB additional memory overhead for each device.
   - If Secure Encrypted Virtualization (SEV) is enabled, add 256 MiB.

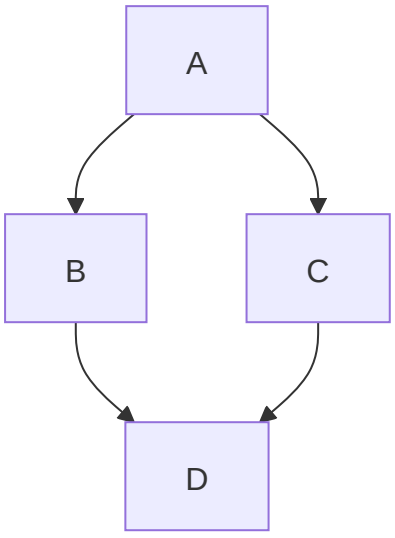
- If Trusted Platform Module (TPM) is enabled, add 53 MiB.

更多源码转换功能请参考 Shiki Transformers↗。

## Mermaid↗

图表绘制工具

```mermaid
graph TD;
    A-->B;
    A-->C;
    B-->D;
    C-->D;
```




配合 Markdown Preview Mermaid↗ 可以在 VSCode 中实时预览。

# MDX

MDX↗ 是一种 Markdown 的扩展语法，允许在 Markdown 中使用 JSX 语法，使用方式可以参考 rspress MDX↗。

---

## 目录


- rspress 组件
- doom 组件
  - `Overview`
  - `Directive`
  - `ExternalSite`
  - `ExternalSiteLink`
  - `AcpApisOverview` 与 `ExternalApisOverview`
  - Term
    - `props`
  - `TermsTable`
    - `props`
  - `JsonViewer`
- 自定义组件复用


---

## rspress 组件

`rspress` 主题提供的内置组件↗大部分已调整为全局组件，可以在 `.mdx` 文件中无需导入直接使用，包括：

- `Badge`
- `Card`
- `LinkCard`
- `PackageManagerTabs`
- `Steps`
- `Tab/Tabs`
- `Toc`

其他不常用的组件可以通过 `rspress/theme` 导入使用，例如：

preview.mdx

```
import { SourceCode } from '@rspress/core/theme'

<SourceCode href="/" />
```

# doom 组件

`doom` 提供了一些全局组件来辅助文档编写，不需要导入即可直接使用，目前包括：

## `Overview`

文档概览组件，用于展示文档目录

## `Directive`

有时，由于嵌套缩进，自定义容器↗语法可能失效，可以使用 `Directive` 组件代替

```
- 多语言文档(`doc/en`)的目录结构需要与 `doc/zh` 目录下的文档完全一致，保证多语言文档的链接除了语言标识外完全相同。

  <Directive type="danger" title="注意">
    如果是使用自动化翻译工具进行翻译，则无需关心该问题，自动化翻译工具会自动根据 `doc/zh` 生成目标语言文档的目录结构。
  </Directive>
```

- 多语言文档( `doc/en` )的目录结构需要与 `doc/zh` 目录下的文档完全一致，保证多语言文档的链接除了语言标识外完全相同。

  > **注意**
  >
  > 如果是使用自动化翻译工具进行翻译，则无需关心该问题，自动化翻译工具会自动根据 `doc/zh` 生成目标语言文档的目录结构。

## `ExternalSite`

引用外部站点组件

```
<ExternalSite name="connectors" />
```

> **Note**
>
> 因为 DevOps 连接器的发版周期与灵雀云容器平台不同，所以 DevOps 连接器的文档现在作为独立的文档站点托管在 DevOps 连接器↗。

## `ExternalSiteLink`

引用外部站点链接组件

```
<ExternalSiteLink name="connectors" href="link.mdx#hash" children="Content" />
```

Content↗

**TIP**

在 mdx 中 `<ExternalSiteLink name="connectors" href="link" children="Content" />` 与下面的内容含义不同

```
<ExternalSiteLink name="connectors" href="link">
  Content {/* 将渲染在 `p` 元素内 */}
</ExternalSiteLink>
```

如果不希望文本渲染在 `p` 元素内，可以像上面的示例一样使用 `children` 属性传递

## `AcpApisOverview` 与 `ExternalApisOverview`

引用外部站点 API 概览组件

```
<AcpApisOverview />
{/* same as following */}
<ExternalApisOverview name="acp" />

<ExternalApisOverview name="connectors" />
```

**Note**

关于 ACP APIs 的使用方法介绍请参考 ACP APIs 指南↗。

**Note**

关于 DevOps 连接器 APIs 的使用方法介绍请参考 DevOps 连接器 APIs 指南↗。

# Term

术语组件，纯文本，动态挂载注入

```
<Term name="company" textCase="capitalize" />
<Term name="product" textCase="lower" />
<Term name="productShort" textCase="upper" />
<Term name="alaudaCloudLink" />
```

灵雀云 灵雀云容器平台 ACP Alauda Cloud↗

## `props`

- `name`: 内置术语名称，参考动态挂载配置文件
- `textCase`: 文本大小写转换，可选值为 `lower`, `upper`, `capitalize`

## `TermsTable`

内置术语列表展示组件

```
<TermsTable />
```

| 名称 | 中文 | 中文反例 | 英文 | 英文反例 | 描述 |
| --- | --- | --- | --- | --- | --- |
| company | 灵雀云 | - | Alauda | - | 公司品牌 |
| product | 灵雀云容器平台 | - | Alauda Container Platform | - | 产品品牌 |
| productShort | ACP | - | ACP | - | 产品品牌简称 |

| 名称 | 中文 | 中文反例 | 英文 | 英文反例 | 描述 |
| --- | --- | --- | --- | --- | --- |
| alaudaCloudLink | Alauda Cloud ↗ | - | Alauda Cloud ↗ | - | - |

### `props`

- `terms`: `NormalizedTermItem[]`，可选，自定义术语列表，方便内部文档渲染自定义术语时复用

### `JsonViewer`

```
<JsonViewer value={{ key: 'value' }} />
```

yaml json

```
key: value
```

## 自定义组件复用

根据约定，我们可以将需要复用的内容抽取到 `shared` 目录中，然后在需要的地方引入即可，比如：

```
import CommonContent from './shared/CommonContent.mdx'

<CommonContent />
```

如果需要使用更多 runtime ↗ 相关的 API，可以使用 `.jsx/.tsx` 实现组件，然后在 `.mdx` 文件中引入使用。

```tsx
// shared/CommonContent.tsx
export const CommonContent = () => {
  const { page } = usePageData()
  return <div>{page.title}</div>
}

// showcase/content.mdx
import { CommonContent } from './shared/CommonContent'
;<CommonContent />
```

**WARNING**

注意：目前 `.mdx` 导出的组件不支持 `props` 传参，参考此 issue，因此需要传递 `props` 的场景请使用 `.jsx/.tsx` 组件进行开发

# 国际化

`alauda` 内部大部分文档都是中英文双语，因此我们默认支持使用 `en` / `zh` 两个子文件夹来存放不同语言的文档，推荐在 `public` 目录下也按 `en` / `zh` 子文件夹存放静态资源，这样可以方便文档内容和静态资源的管理。

---

## 目录


- `i18n.json`
- `.ts/.tsx`
- `.mdx`


---

## `i18n.json`

对于可复用组件，如果需要在同一个组件中同时支持中英文，那么就需要先在 `docs` 目录下创建 `i18n.json` 文件，然后在组件中通过 `useI18n` 来获取当前语言的文本，比如：

```
docs/i18n.json
```

```
{
  "title": {
    "zh": "标题",
    "en": "Title"
  },
  "description": {
    "zh": "描述",
    "en": "description"
  }
}
```

## .ts/.tsx

```
import { useI18n } from '@rspress/core/runtime'

export const CommonContent = () => {
  const t = useI18n()
  return <h1>{t('title')}</h1>
}
```

## .mdx

```
import { useI18n } from '@rspress/core/runtime'

# {useI18n()('title')}

{useI18n()('description')}
```

# API 文档

根据实际业务，我们一般会将 API 分为标准 K8S API, 高级 API 和 CRD (Custom Resource Definition) 三种，因此在目录结构上一般分为：

```
├── apis
│   ├── advanced_apis # 高级 API
│   ├── crds # CRDs
│   ├── kubernetes_apis # K8S API
│   └── references # 公共引用
```

## 目录


K8S API

`props`

高级 API

`props`

CRD (deprecated)

`props`

公共引用

`props`

指定 openapi 路径


## K8S API

kubernetes_apis/workload/daemonset.mdx

```
# DaemonSet [apps/v1]

<K8sAPI
  name="io.k8s.api.apps.v1.DaemonSet"
  pathPrefix="/kubernetes/{cluster}"
/>
```

参考 DaemonSet。

crds/ArtifactCleanupRun.mdx

```
# ArtifactCleanupRun

<K8sAPI name="artifactcleanupruns.artifacts.katanomi.dev" />
```

参考 ArtifactCleanupRun。

### `props`

- `name`: OpenAPI schema `definitions`(v2) or `components/schemas`(v3) 下的引用名称或CRD `metadata.name`
- `namespaced`: 指示资源是否为命名空间级别，默认为 `true`，即 API Endpoints 是否包含命名空间路径参数 `namespaces/{namespace}`
- `pathPrefix`: 可以用于覆盖全局配置中的 `api.pathPrefix`
- `filepath`: 类似指定 openapi 路径，用于指定特定的 openapi 或 CRD 文件
- `apiGroup`: 可选，指定 API 组，openapi 会尝试读取引用的 `x-kubernetes-group-version-kind`，下同
- `apiVersion`: 可选，指定 API 版本，CRD 会默认使用 `spec.versions` 中第一个版本
- `apiKind`: 可选，指定 API 资源类型

## 高级 API

advanced_apis/codeQualityTaskSummary.mdx

```
# CodeQualityTaskSummary

<OpenAPIPath path="/plugins/v1alpha1/template/codeQuality/task/{task-id}/summary" />
```

参考 CodeQualityTaskSummary。

### `props`

- `path`: OpenAPI schema `paths` 下的路径
- `pathPrefix`: 可以用于覆盖全局配置中的 `api.pathPrefix`
- `openapiPath`: 参考指定 openapi 路径

## CRD (deprecated)

> **WARNING**
>
> 请使用 `K8sAPI` 组件替代 `K8sCrd` 组件，`K8sCrd` 组件将在未来版本中移除。

crds/ArtifactCleanupRun-K8sCrd.mdx

```
# ArtifactCleanupRun - K8sCrd

<K8sCrd name="artifactcleanupruns.artifacts.katanomi.dev" />
```

参考 ArtifactCleanupRun-K8sCrd。

### `props`

- `name`: CRD `metadata.name`
- `crdPath`: 类似指定 openapi 路径，用于指定特定的 CRD 文件

# 公共引用

references/CodeQuality.mdx

```
# CodeQuality

<OpenAPIRef schema="v1alpha1.CodeQuality" />
```

参考 CodeQuality。

## `props`

- `schema`: OpenAPI schema `definitions`(v2) or `components/schemas`(v3) 下的引用名称
- `openapiPath`: 参考指定 openapi 路径

# 指定 openapi 路径

对于 `OpenAPIPath` 和 `OpenAPIRef` 组件，默认会在所有 openapi 定义文件中查找至匹配，如果需要指定特定的 openapi 文件，可以使用 `openapiPath` 属性指定：

```
<OpenAPIPath
  path="/plugins/v1alpha1/template/codeQuality/task/{task-id}/summary"
  openapiPath="shared/openapis/katanomi.json"
/>
```

# 权限说明文档

```
<K8sPermissionTable functions={['devops-testplans', 'devops-testmodules']} />
```

## 目录


`props`

示例


## `props`

- `functions`: `string[]` - 必填，需要展示的 `FunctionResource` 资源名称数组

## 示例

| 功能 | 操作 | 平台管理员 | 平台审计人员 | 项目管理员 | 命名空间管理员 | 开发人员 | 集群管理员 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 测试计划 `devops-testplans` | 查看 | ✓ | ✓ | ✓ | ✓ | ✓ | × |
| | 创建 | ✓ | × | ✓ | ✓ | ✓ | × |

| 功能 | 操作 | 平台管理员 | 平台审计人员 | 项目管理员 | 命名空间管理员 | 开发人员 | 集群管理员 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| | 更新 | ✓ | × | ✓ | ✓ | ✓ | × |
| | 删除 | ✓ | × | ✓ | ✓ | ✓ | × |
| 测试模块 `devops-testmodules` | 查看 | ✓ | ✓ | ✓ | ✓ | ✓ | × |
| | 创建 | ✓ | × | ✓ | ✓ | ✓ | × |
| | 更新 | ✓ | × | ✓ | ✓ | ✓ | × |
| | 删除 | ✓ | × | ✓ | ✓ | ✓ | × |

# 引用文档

在 Markdown 文件中：

```
<!-- reference-start#name -->

<!-- reference-end -->
```

在 MDX 文件中：

```
{/* reference-start#name */}

{/* reference-end */}
```

上述 `name` 为引用文档的名称，参考引用文档配置，如果引用的文档内容使用了远程仓库图片静态资源，相关静态资源将自动存储在本地 `<root>/public/_remotes/<name>` 目录下。

以下为使用 `<!-- reference-start#ref -->` 的实例：

---

## 目录


- 引用文档配置
  - `frontmatterMode`


---

## 引用文档配置

```
reference:
  - repo: alauda-public/product-doc-guide # 可选，引用文档仓库地址，如果不填写，则默认使用当前文档仓库地址
    branch: # [string] 可选，引用文档仓库分支
    publicBase: # [string] 可选，使用远程仓库时使用绝对路径 /images/xx.png 对应的静态资源所在目录，默认为 docs/public
    sources:
      - name: anchor # 引用文档名称，用于在文档中引用，全局唯一
        path: docs/index.mdx#介绍 # 引用文档路径，支持锚点定位，远程仓库相对于仓库根目录，本地相对于 doom.config.* 所在目录
        ignoreHeading: # [boolean] 可选，是否忽略标题，如果为 true，则不会在引用文档中显示锚点的标题
        processors: # 可选，引用文档内容处理器
          - type: ejsTemplate
            data: # ejs 模板参数，使用 `<%= data.xx %>` 访问
        frontmatterMode: merge # 可选，引用文档处理 frontmatter 模式，默认为 ignore，可选值为 ignore/merge/replace/remove
```

## `frontmatterMode`

- `ignore`：忽略引用文档的 frontmatter，保留使用当前文档的 frontmatter
- `merge`：合并引用文档的 frontmatter，如果有相同的 key，引用文档的值会覆盖当前文档的值
- `replace`：使用引用文档的 frontmatter 替换当前文档的 frontmatter
- `remove`：移除当前文档的 frontmatter

文档编写参考引用文档

# 部署

## 目录

构建与预览
多版本构建
合并目录结构
动态挂载配置文件

## 构建与预览

在部署之前，我们需要先对项目进行生产环境的构建，并在本地进行预览，以确保项目能够正常运行：

```
doom build # 构建静态产物
doom serve # 以生产模式预览构建产物
```

## 多版本构建

默认情况下，`doom build` 会将构建产物输出到 `dist` 目录，如果需要构建多个版本的文档，可以通过 `-v` 参数指定版本号，例如：

```
# 一般由分支名确定，如 release-4.0 对应 4.0 版本
doom build -v 4.0 # 构建 4.0 版本，产物输出到 dist/4.0，文档访问路径为 {base}/4.0
doom build -v master # 构建 master 版本，产物输出到 dist/master，文档访问路径为 {base}/master
doom build -v {other} # 构建其他版本，产物输出到 dist/{other}，文档访问路径为 {base}/{other}

# unversioned 和 unversioned-x.y 为特殊版本号，用于构建无版本号前缀的文档
doom build -v unversioned # 构建无版本号前缀的文档，产物输出到 dist/unversioned，文档访问路径为 {base}
doom build -v unversioned-4.0 # 构建无版本号前缀但导航栏展示版本号 4.0 的文档，产物输出到 dist/unversioned，文档访问路径为 {base}
```

## 合并目录结构

```
├── console-platform
│   ├── 4.0
│   ├── 4.1
│   ├── index.html
│   ├── overrides.yaml
│   └── versions.yaml
├── console-devops-docs
│   ├── 4.0
│   ├── 4.1
│   ├── index.html
│   ├── overrides.yaml
│   └── versions.yaml
├── console-tekton-docs
│   ├── 1.0
│   ├── 1.1
│   ├── index.html
│   ├── overrides.yaml
│   └── versions.yaml
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Redirecting...</title>
    <meta http-equiv="refresh" content="0; url=/console-docs/4.1" />
  </head>
  <body>
    <p>Redirecting to <a href="/console-docs/4.1">/console-docs/4.1</a></p>
  </body>
</html>
```

# 动态挂载配置文件

overrides.yaml

```
# 文档信息，每个文档都可以挂载覆盖默认配置
title:
  en: Doom - Alauda
  zh: Doom - 灵雀云
logoText:
  en: Doom - Alauda
  zh: Doom - 灵雀云
```

versions.yaml

```
- '4.1'
- '4.0'
```