

# 目录

README	1.1
Docsify使用	2.1

## 入门

定制化	3.1
配置项	3.1.1
主题	3.1.2
插件列表	3.1.3
开发插件	3.1.4
Markdown 配置	3.1.5
代码高亮	3.1.6
指南	3.2
部署	3.2.1
文档助手	3.2.2
CDN	3.2.3
离线模式(PWA)	3.2.4
服务端渲染 (SSR)	3.2.5
文件嵌入	3.2.6
Awesome docsify	3.3

# 知识库

- [项目环境说明](#)
- [项目开发手册](#)
- [项目笔记](#)

# docsify

一个神奇的文档网站生成器。

## 概述

docsify 可以快速帮你生成文档网站。不同于 **GitBook**、**Hexo** 的地方是它不会生成静态的 `.html` 文件，所有转换工作都是在运行时。如果你想要开始使用它，只需要创建一个 `index.html` 就可以开始编写文档并直接部署在 [GitHub Pages](#)。

查看[快速开始](#)了解详情。

## 特性

- 无需构建，写完文档直接发布
- 容易使用并且轻量 (压缩后 ~21kB)
- 智能的全文搜索
- 提供多套主题
- 丰富的 API
- 支持 Emoji
- 兼容 IE11
- 支持服务端渲染 SSR ([示例](#))

## 示例

可以查看 [Showcase](#) 来了解更多在使用 docsify 的文档项目。

## 捐赠

如果你觉得 **docsify** 对你有帮助，或者想对我微小的工作一点资助，欢迎给我[捐赠](#)。

## 社区

在[Gitter](#)的社区里可以找到**docsify**的用户和开发者团队。

# 配置项

你可以配置在 `window.$docsify` 里。

```
<script>
window.$docsify = {
  repo: 'docsifyjs/docsify',
  maxLevel: 3,
  coverpage: true,
};
</script>
```

## el

- 类型: `String`
- 默认值: `#app`

`docsify` 初始化的挂载元素，可以是一个 CSS 选择器，默认为 `#app` 如果不存在就直接绑定在 `body` 上。

```
window.$docsify = {
  el: '#app',
};
```

## repo

- 类型: `String`
- 默认值: `null`

配置仓库地址或者 `username/repo` 的字符串，会在页面右上角渲染一个 [GitHub Corner](#) 挂件。

```
window.$docsify = {
```

```
repo: 'docsifyjs/docsify',  
// or  
repo: 'https://github.com/docsifyjs/docsify/',  
};
```

## maxLevel

- 类型: `Number`
- 默认值: `6`

默认情况下会抓取文档中所有标题渲染成目录，可配置最大支持渲染的标题层级。

```
window.$docsify = {  
  maxLevel: 4,  
};
```

## loadNavbar

- 类型: `Boolean|String`
- 默认值: `false`

加载自定义导航栏，参考[定制导航栏](#)了解用法。设置为 `true` 后会加载 `_navbar.md` 文件，也可以自定义加载的文件名。

```
window.$docsify = {  
  // 加载 _navbar.md  
  loadNavbar: true,  
  
  // 加载 nav.md  
  loadNavbar: 'nav.md',  
};
```

## loadSidebar

- 类型: `Boolean|String`
- 默认值: `false`

加载自定义侧边栏，参考[多页文档](#)。设置为 `true` 后会加载 `_sidebar.md` 文件，也可以自定义加载的文件名。

```
window.$docsify = {  
  // 加载 _sidebar.md  
  loadSidebar: true,  
  
  // 加载 summary.md  
  loadSidebar: 'summary.md',  
};
```

## hideSidebar

- 类型: `Boolean`
- 默认值: `true`

这个选项用来完全隐藏侧边栏，侧边栏的任何内容都不会被渲染。

```
window.$docsify = {  
  hideSidebar: true,  
};
```

## subMaxLevel

- 类型: `Number`
- 默认值: `0`

自定义侧边栏后默认不会再生成目录，你也可以通过设置生成目录的最大层级开启这个功能。

```
window.$docsify = {  
  subMaxLevel: 2,
```

```
};
```

## auto2top

- 类型: `Boolean`
- 默认值: `false`

切换页面后是否自动跳转到页面顶部。

```
window.$docsify = {  
  auto2top: true,  
};
```

## homepage

- 类型: `String`
- 默认值: `README.md`

设置首页文件加载路径。适合不想将 `README.md` 作为入口文件渲染，或者是文档存放在其他位置的情况使用。

```
window.$docsify = {  
  // 入口文件改为 /home.md  
  homepage: 'home.md',  
  
  // 文档和仓库根目录下的 README.md 内容一致  
  homepage:  
    'https://raw.githubusercontent.com/docsifyjs/docsify/master/  
README.md',  
};
```

## basePath

- 类型: `String`



文档加载的根路径，可以是二级路径或者是其他域名的路径。

```
window.$docsify = {  
  basePath: '/path/',  
  
  // 直接渲染其他域名的文档  
  basePath: 'https://docsify.js.org/',  
  
  // 甚至直接渲染其他仓库  
  basePath:  
    'https://raw.githubusercontent.com/ryanmcdermott/clean-code-javascript/master/',  
};
```

## relativePath

- 类型: `Boolean`
- 默认值: `false`

如果该选项设为 **true**，那么链接会使用相对路径。

例如，像这样的目录结构：

```
.  
├── docs  
│   ├── README.md  
│   ├── guide.md  
│   └── zh-cn  
│       ├── README.md  
│       ├── guide.md  
│       └── config  
│           └── example.md
```

如果 启用了相对路径，当前链接是 `http://domain.com/zh-cn/README`，对应的链接会解析为：

```
guide.md          => http://domain.com/zh-cn/guide
```

```
config/example.md    => http://domain.com/zh-cn/config/example
../README.md         => http://domain.com/README
/README.md           => http://domain.com/README
```

```
window.$docsify = {
  // 启用相对路径
  relativePath: true,

  // 禁用相对路径（默认值）
  relativePath: false,
};
```

## coverpage

- 类型: `Boolean|String`
- 默认值: `false`

启用封面页。开启后是加载 `_coverpage.md` 文件，也可以自定义文件名。

```
window.$docsify = {
  coverpage: true,

  // 自定义文件名
  coverpage: 'cover.md',

  // 多个封面页
  coverpage: ['/', '/zh-cn/'],

  // 多个封面页，并指定文件名
  coverpage: {
    '/': 'cover.md',
    '/zh-cn/': 'cover.md',
  },
};
```

## logo

- 类型: `String`

在侧边栏中出现的网站图标，你可以使用 `css` 来更改大小

```
window.$docsify = {  
  logo: '/_media/icon.svg',  
};
```

## name

- 类型: `String`

文档标题，会显示在侧边栏顶部。

```
window.$docsify = {  
  name: 'docsify',  
};
```

`name` 项也可以包含自定义 `HTML` 代码来方便地定制。

```
window.$docsify = {  
  name: '<span>docsify</span>',  
};
```

## nameLink

- 类型: `String`
- 默认值: `window.location.pathname`

点击文档标题后跳转的链接地址。

```
window.$docsify = {  
  nameLink: '/',  
  
  // 按照路由切换
```

```
nameLink: {
  '/zh-cn/': '/zh-cn/',
  '/': '/',
},
};
```

## markdown

- 类型: `Object|Function`

参考 [Markdown 配置](#)。

```
window.$docsify = {
  // object
  markdown: {
    smartypants: true,
    renderer: {
      link: function() {
        // ...
      },
    },
  },
  // function
  markdown: function(marked, renderer) {
    // ...
    return marked;
  },
};
```

## themeColor

- 类型: `String`

替换主题色。利用 [CSS3 支持变量](#)的特性，对于老的浏览器有 `polyfill` 处理。

```
window.$docsify = {  
  themeColor: '#3F51B5'  
};
```

## alias

- 类型: `Object`

定义路由别名, 可以更自由的定义路由规则。支持正则。

```
window.$docsify = {  
  alias: {  
    '/foo/(.*)': '/bar/$1', // supports regexp  
    '/zh-cn/changelog': '/changelog',  
    '/changelog':  
      'https://raw.githubusercontent.com/docsifyjs/docsify/master/CHANGELOG',  
    '/*/_sidebar.md': '/_sidebar.md', // See #301  
  },  
};
```

## autoHeader

- 类型: `Boolean`

同时设置 `loadSidebar` 和 `autoHeader` 后, 可以根据 `_sidebar.md` 的内容自动为每个页面增加标题。 [#78](#)

```
window.$docsify = {  
  loadSidebar: true,  
  autoHeader: true,  
};
```

## executeScript

- 类型: Boolean

执行文档里的 `script` 标签里的脚本，只执行第一个 `script` ([demo](#))。如果 Vue 存在，则自动开启。

```
window.$docsify = {  
  executeScript: true,  
};
```

```
## This is test
```

```
<script>  
  console.log(2333)  
</script>
```

注意如果执行的是一个外链脚本，比如 `jsfiddle` 的内嵌 `demo`，请确保引入 `external-script` 插件。

## noEmoji

- 类型: Boolean

禁用 emoji 解析。

```
window.$docsify = {  
  noEmoji: true,  
};
```

?> 如果该选项设为 `false`，但是你不希望解析一些特别的表情符，[参考这里](#)

## mergeNavbar

- 类型: Boolean

小屏设备下合并导航栏到侧边栏。

```
window.$docsify = {  
  mergeNavbar: true,  
};
```

## formatUpdated

- 类型: `String|Function`

我们可以通过 **{docsify-updated}** 变量显示文档更新日期. 并且通过 `formatUpdated` 配置日期格式。参考

<https://github.com/lukeed/tinydate#patterns>

```
window.$docsify = {  
  formatUpdated: '{MM}/{DD} {HH}:{mm}',  
  
  formatUpdated: function(time) {  
    // ...  
  
    return time;  
  },  
};
```

## externalLinkTarget

- 类型: `String`
- 默认: `_blank`

外部链接的打开方式。默认为 `_blank` （在新窗口或者标签页中打开）

```
window.$docsify = {  
  externalLinkTarget: '_self', // default: '_blank'  
};
```

## cornerExternalLinkTarget

- 类型: String
- 默认值: `_blank`

右上角链接的打开方式。默认为 `_blank`（在新窗口或者标签页中打开）

```
window.$docsify = {  
  cornerExternalLinkTarget: '_self', // default: '_blank'  
};
```

## externalLinkRel

- 类型: String
- 默认值: `noopener`

默认为 `noopener` (no opener) 可以防止新打开的外部页面（当 `externalLinkTarget` 设为 `_blank` 时）能够控制我们的页面，没有设为 `_blank` 的话就不需要设置这个选项了。

```
window.$docsify = {  
  externalLinkRel: '', // default: 'noopener'  
};
```

## routerMode

- 类型: String
- 默认: `hash`

```
window.$docsify = {  
  routerMode: 'history', // default: 'hash'  
};
```



## crossOriginLinks

- type: `Array`

当设置了 `routerMode: 'history'` 时，你可能会面临跨域的问题，参见 [#1379](#)。在 Markdown 内容中，有一个简单的方法可以解决，参见 [helpers](#) 中的跨域链接。

```
window.$docsify = {  
  crossOriginLinks: ['https://example.com/cross-origin-link'],  
};
```

## noCompileLinks

- 类型: `Array`

有时我们不希望 docsify 处理我们的链接。参考 [#203](#)

```
window.$docsify = {  
  noCompileLinks: ['/foo', '/bar/.*'],  
};
```

## onlyCover

- 类型: `Boolean`

只在访问主页时加载封面。

```
window.$docsify = {  
  onlyCover: false,  
};
```

## requestHeaders

- 类型: `Object`

设置请求资源的请求头。

```
window.$docsify = {  
  requestHeaders: {  
    'x-token': 'xxx',  
  },  
};
```

例如设置缓存

```
window.$docsify = {  
  requestHeaders: {  
    'cache-control': 'max-age=600',  
  },  
};
```

## ext

- 类型: `String`

资源的文件扩展名。

```
window.$docsify = {  
  ext: '.md',  
};
```

## fallbackLanguages

- 类型: `Array<string>`

一个语言列表。在浏览这个列表中的语言的翻译文档时都会在请求到一个对应语言的翻译文档，不存在时显示默认语言的同名文档

Example:

- 尝试访问 `/de/overview` ， 如果存在则显示
- 如果不存在则尝试 `/overview` （取决于默认语言）， 如果存在即显示
- 如果也不存在， 显示404页面

```
window.$docsify = {  
  fallbackLanguages: ['fr', 'de'],  
};
```

## notFoundPage

- 类型: `Boolean` | `String` | `Object`

在找不到指定页面时加载 `_404.md` :

```
window.$docsify = {  
  notFoundPage: true,  
};
```

加载自定义404页面:

```
window.$docsify = {  
  notFoundPage: 'my404.md',  
};
```

加载正确的本地化过的404页面:

```
window.$docsify = {  
  notFoundPage: {  
    '/': '_404.md',  
    '/de': 'de/_404.md',  
  },  
};
```

注意: 配置过 `fallbackLanguages` 这个选项的页面与这个选项 `notFoundPage` 冲突。

## topMargin

- 类型: `Number`
- 默认值: `0`

让你的内容页在滚动到指定的锚点时，距离页面顶部有一定空间。当你使用 `固定页头` 布局时这个选项很有用，可以让你的锚点对齐标题栏。

```
window.$docsify = {  
  topMargin: 90, // default: 0  
};
```

## vueComponents

- type: `Object`

创建并注册全局 `Vue` 组件。组件是以组件名称为键，以包含 `Vue` 选项的对象为值来指定的。组件 `data` 对每个实例都是唯一的，并且在用户浏览网站时不会持久。

```
window.$docsify = {  
  vueComponents: {  
    'button-counter': {  
      template: `  
        <button @click="count += 1">  
          You clicked me {{ count }} times  
        </button>  
      `,  
      data() {  
        return {  
          count: 0,  
        };  
      },  
    },  
  },  
};
```

```
    },  
  },  
};
```

```
<button-counter></button-counter>
```

## vueGlobalOptions

- type: `Object`

指定 [Vue选项](#)，用于未明确使用[vueMounts](#)、[vueComponents](#)或[markdown脚本](#)挂载的 **Vue** 内容。对全局 `data` 的更改将持续存在，并在任何使用全局引用的地方得到反映。

```
window.$docsify = {  
  vueGlobalOptions: {  
    data() {  
      return {  
        count: 0,  
      };  
    },  
  },  
};
```

```
<p>  
  <button @click="count -= 1">-</button>  
  {{ count }}  
  <button @click="count += 1">+</button>  
</p>
```



## vueMounts

- type: Object

指定要挂载为 **Vue实例** 的 DOM 元素及其相关选项。挂载元素使用 **CSS选择器** 作为键，并使用包含 **Vue** 选项的对象作为其值。每次加载新页面时，**Docsify** 将挂载主内容区域中第一个匹配的元素。挂载元素 **data** 对每个实例来说都是唯一的，并且不会在用户浏览网站时持久存在。

```
window.$docsify = {  
  vueMounts: {  
    '#counter': {  
      data() {  
        return {  
          count: 0,  
        };  
      },  
    },  
  },  
};
```

```
<div id="counter">  
  <button @click="count -= 1">-</button>  
  {{ count }}  
  <button @click="count += 1">+</button>  
</div>
```



# 主题

目前提供三套主题可供选择，模仿 [Vue](#) 和 [buble](#) 官网订制的主题样式。还有 [@liril-net](#) 贡献的黑色风格的主题。

```
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify/themes/vue.css">
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify/themes/buble.css">
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify/themes/dark.css">
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify/themes/pure.css">
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify/themes/dolphin.css">
```

!> CSS 的压缩文件位于 `/lib/themes/`

```
<!-- compressed -->

<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify/lib/themes/vue.css">
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify/lib/themes/buble.css">
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify/lib/themes/dark.css">
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify/lib/themes/pure.css">
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify/lib/themes/dolphin.css">
```

如果你有其他想法或者想开发别的主题，欢迎提 [PR](#)。

点击切换主题

[vue.css](#) [buble.css](#) [dark.css](#) [pure.css](#) [dolphin.css](#)

## 其他主题

- [docsify-themeable](#) 一个用于docsify的，简单到令人愉悦的主题系统.



# 插件列表

## 全文搜索 - Search

全文搜索插件会根据当前页面上的超链接获取文档内容，在 `localStorage` 内建立文档索引。默认过期时间为一天，当然我们可以自己指定需要缓存的文件列表或者配置过期时间。

```
<script>
window.$docsify = {
  search: 'auto', // 默认值

  search : [
    '/',           // => /README.md
    '/guide',      // => /guide.md
    '/get-started', // => /get-started.md
    '/zh-cn/',     // => /zh-cn/README.md
  ],

  // 完整配置参数
  search: {
    maxAge: 86400000, // 过期时间，单位毫秒，默认一天
    paths: [], // or 'auto'
    placeholder: 'Type to search',

    // 支持本地化
    placeholder: {
      '/zh-cn/': '搜索',
      '/': 'Type to search'
    },

    noData: 'No Results!',

    // 支持本地化
    noData: {
      '/zh-cn/': '找不到结果',
      '/': 'No Results'
    }
  }
}
```

```

    },

    // 搜索标题的最大层级, 1 - 6
    depth: 2,

    hideOtherSidebarContent: false, // 是否隐藏其他侧边栏内容

    // 避免搜索索引冲突
    // 同一域下的多个网站之间
    namespace: 'website-1',

    // 使用不同的索引作为路径前缀 (namespaces)
    // 注意: 仅适用于 paths: 'auto' 模式
    //
    // 初始化索引时, 我们从侧边栏查找第一个路径
    // 如果它与列表中的前缀匹配, 我们将切换到相应的索引
    pathNamespaces: ['/zh-cn', '/ru-ru', '/ru-ru/v1'],

    // 您可以提供一个正则表达式来匹配前缀。在这种情况下,
    // 匹配到的字符串将被用来识别索引
    pathNamespaces: /^(\/(zh-cn|ru-ru))?(\/(v1|v2))?/
  }
}
</script>
<script src="//cdn.jsdelivr.net/npm/docsify/lib/docsify.min.js">
</script>
<script src="//cdn.jsdelivr.net/npm/docsify/lib/plugins/search.min.js"></script>

```

当执行全文搜索时, 该插件会忽略双音符 (例如, "cafe" 也会匹配 "café")。像 IE11 这样的旧版浏览器需要使用以下 [String.normalize\(\)](#) polyfill 来忽略双音符:

```

<script src="//polyfill.io/v3/polyfill.min.js?features=String.prototype.normalize"></script>

```

## 谷歌统计 - Google Analytics


需要配置 track id 才能使用。

```
<script>
  window.$docsify = {
    ga: 'UA-XXXXX-Y'
  }
</script>
<script src="//cdn.jsdelivr.net/npm/docsify/lib/docsify.min.js">
</script>
<script src="//cdn.jsdelivr.net/npm/docsify/lib/plugins/ga.min.js"
"></script>
```

也可以通过 `data-ga` 配置 id。

```
<script src="//cdn.jsdelivr.net/npm/docsify/lib/docsify.min.js"
data-ga="UA-XXXXX-Y"></script>
<script src="//cdn.jsdelivr.net/npm/docsify/lib/plugins/ga.min.js"
"></script>
```

## emoji

默认是提供 emoji 解析的，能将类似 `:100:` 解析成 。但是它不是精准的，因为没有处理非 emoji 的字符串。如果你需要正确解析 emoji 字符串，你可以引入这个插件。

```
<script src="//cdn.jsdelivr.net/npm/docsify/lib/plugins/emoji.min.js"></script>
```

?> 如果你不想解析成表情符号，可以使用 `_colon_` 或 `&#58;`。如果你需要在标题中使用，我们建议使用 `&#58;`。例如， `&#58;100:`。

## 外链脚本 - External Script

如果文档里的 `script` 是内联脚本，可以直接执行；而如果是外链脚本（即 `js` 文件内容由 `src` 属性引入），则需要使用此插件。

```
<script src="//cdn.jsdelivr.net/npm/docsify/lib/plugins/external-script.min.js"></script>
```

## 图片缩放 - Zoom image

Medium's 风格的图片缩放插件. 基于 [medium-zoom](#)。

```
<script src="//cdn.jsdelivr.net/npm/docsify/lib/plugins/zoom-image.min.js"></script>
```

忽略某张图片

```

```

## 在 Github 上编辑

在每一页上添加 `Edit on github` 按钮. 由第三方库提供, 查看 [document](#)

## 代码即时预览和 jsfiddle 集成

通过这个插件，示例代码可以在页面上即时呈现，让读者可以立即看到预览。当读者展开演示框时，源码和说明就会显示在那里，如果点击 `Try in Jsfiddle` 按钮，`jsfiddle.net` 就会打开这个例子的代码，让读者自己修改代码和测试。

docsify同时支持[Vue](#)和[React](#)版本的插件。

## 复制到剪贴板

在所有的代码块上添加一个简单的 `Click to copy` 按钮来允许用户从你的文档中轻易地复制代码。由@jperasmus提供。

```
<script src="//cdn.jsdelivr.net/npm/docsify-copy-code/dist/docsify-copy-code.min.js"></script>
```

详情可参考 [README.md](#) 。

## Disqus

Disqus评论系统支持。 <https://disqus.com/>

```
<script>
  window.$docsify = {
    disqus: 'shortname'
  }
</script>
<script src="//cdn.jsdelivr.net/npm/docsify/lib/plugins/disqus.min.js"></script>
```

## Gitalk

[Gitalk](#)，一个现代化的，基于Preact和Github Issue的评论系统。

```
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/gitalk/dist/gitalk.css">

<script src="//cdn.jsdelivr.net/npm/docsify/lib/plugins/gitalk.min.js"></script>
<script src="//cdn.jsdelivr.net/npm/gitalk/dist/gitalk.min.js"></script>
<script>
  const gitalk = new Gitalk({
    clientId: 'Github Application Client ID',
    clientSecret: 'Github Application Client Secret',
    repo: 'Github repo',
```

```
    owner: 'Github repo owner',
    admin: ['Github repo collaborators, only these guys can initialize github issues'],
    // facebook-like distraction free mode
    distractionFreeMode: false
  })
</script>
```

## Pagination

docsify的分页导航插件，由[@imyelo](#)提供。

```
<script src="//cdn.jsdelivr.net/npm/docsify/lib/docsify.min.js">
</script>
<script src="//cdn.jsdelivr.net/npm/docsify-pagination/dist/docsify-pagination.min.js"></script>
```

从[这里](#)获取更多信息。

## 字数统计

这是一款为docsify提供文字统计的插件。 [@827652549](#)提供

它提供了统计中文汉字和英文单词的功能，并且排除了一些markdown语法的特殊字符例如\*、-等

### Add JS

```
<script src="//unpkg.com/docsify-count/dist/countable.js"></script>
```

### Add settings

```
window.$docsify = {
```

```
count:{
  countable:true,
  fontsize:'0.9em',
  color:'rgb(90,90,90)',
  language:'chinese'
}
```

check [document](#)

## Code Fund

帮你快速接入[Code Fund](#)的插件, 由[@njleonzhang](#)提供。

Code Fund 以前叫 codesponsor

```
<script src="//cdn.jsdelivr.net/npm/docsify/lib/docsify.min.js">
</script>

window.$docsify = {
  plugins: [
    DocsifyCodefund.create('51d43327-eea3-4e27-bd44-e075e67a84fb
  ') // 把这个id改成你的codefund id
  ]
}
```

## Tabs

这个插件用来在 **Markdown** 中显示选项卡。

- [文档和示例](#)

开发: [@jhildenbiddle](#).

## 更多插件

参考 [awesome-docsify](#)



# 自定义插件

docsify 提供了一套插件机制，其中提供的钩子（hook）支持处理异步逻辑，可以很方便的扩展功能。

## 完整功能

```
window.$docsify = {
  plugins: [
    function(hook, vm) {
      hook.init(function() {
        // 初始化完成后调用，只调用一次，没有参数。
      });

      hook.beforeEach(function(content) {
        // 每次开始解析 Markdown 内容时调用
        // ...
        return content;
      });

      hook.afterEach(function(html, next) {
        // 解析成 html 后调用。
        // beforeEach 和 afterEach 支持处理异步逻辑
        // ...
        // 异步处理完成后调用 next(html) 返回结果
        next(html);
      });

      hook.doneEach(function() {
        // 每次路由切换时数据全部加载完成后调用，没有参数。
        // ...
      });

      hook.mounted(function() {
        // 初始化并第一次加载完成数据后调用，只触发一次，没有参数。
      });
    }
  ]
};
```

```

    hook.ready(function() {
      // 初始化并第一次加载完成数据后调用，没有参数。
    });
  }
]
};

```

!> 如果需要用 docsify 的内部方法，可以通过 `window.Docsify` 获取，通过 `vm` 获取当前实例。

## 例子

### footer

给每个页面的末尾加上 `footer`

```

window.$docsify = {
  plugins: [
    function(hook) {
      var footer = [
        '<hr/>',
        '<footer>',
        '<span><a href="https://github.com/QingWei-Li">cinwell</a> &copy;2017.</span>',
        '<span>Proudly published with <a href="https://github.com/docsifyjs/docsify" target="_blank">docsify</a>.</span>',
        '</footer>'
      ].join('');

      hook.afterEach(function(html) {
        return html + footer;
      });
    }
  ]
};

```

### Edit Button

```

window.$docsify = {
  plugins: [
    function(hook, vm) {
      hook.beforeEach(function(html) {
        var url =
          'https://github.com/docsifyjs/docsify/blob/master/docs
/' +
          vm.route.file;
        var editHtml = '[ EDIT DOCUMENT](' + url + ')\n';

        return (
          editHtml +
          html +
          '\n----\n' +
          'Last modified {docsify-updated} ' +
          editHtml
        );
      });
    }
  ]
};

```

## 小技巧

### 获取 **docsify** 版本

```
console.log(window.Docsify.version)
```

当前版本: 正在加载

# Markdown 配置

内置的 Markdown 解析器是 [marked](#)，可以修改它的配置。同时可以直接配置 `renderer`。

```
window.$docsify = {  
  markdown: {  
    smartypants: true,  
    renderer: {  
      link: function() {  
        // ...  
      }  
    }  
  }  
}
```

?> 完整配置参数参考 [marked 文档](#)

当然也可以完全定制 Markdown 解析规则。

```
window.$docsify = {  
  markdown: function(marked, renderer) {  
    // ...  
  
    return marked  
  }  
}
```

## 支持 mermaid

```
// Import mermaid  
// <link rel="stylesheet" href="//cdn.jsdelivr.net/npm/mermaid/  
dist/mermaid.min.css">  
// <script src="//cdn.jsdelivr.net/npm/mermaid/dist/mermaid.min
```

```

.js"></script>

var num = 0;
mermaid.initialize({ startOnLoad: false });

window.$docsify = {
  markdown: {
    renderer: {
      code: function(code, lang) {
        if (lang === "mermaid") {
          return (
            '<div class="mermaid">' + mermaid.render('mermaid-sv
g-' + num++, code) + "</div>"
          );
        }
        return this.origin.code.apply(this, arguments);
      }
    }
  }
}

```

# 代码高亮

**docsify**内置的代码高亮工具是 **Prism**。Prism 默认支持的语言如下：

- Markup - markup , html , xml , svg , mathml , ssml , atom , rss
- CSS - css
- C-like - clike
- JavaScript - javascript , js

添加额外的语法支持需要通过CDN添加相应的语法文件：

```
<script src="//cdn.jsdelivr.net/npm/prismjs@1/components/prism-bash.min.js"></script>
<script src="//cdn.jsdelivr.net/npm/prismjs@1/components/prism-php.min.js"></script>
```

要使用语法高亮，需要在代码块第一行添加对应的语言声明，示例如下：

```
```html
<p>This is a paragraph</p>
<a href="//docsify.js.org/">Docsify</a>
```

```bash
echo "hello"
```

```php
function getAdder(int $x): int
{
    return 123;
}
```
```

上面代码的渲染结果:

```
<p>This is a paragraph</p>  
<a href="//docsify.js.org/">Docsify</a>
```

```
echo "hello"
```

```
function getAdder(int $x): int  
{  
    return 123;  
}
```

# 部署

和 GitBook 生成的文档一样，我们可以直接把文档网站部署到 GitHub Pages 或者 VPS 上。

## GitHub Pages

GitHub Pages 支持从三个地方读取文件

- docs/ 目录
- master 分支
- gh-pages 分支

我们推荐直接将文档放在 docs/ 目录下，在设置页面开启 **GitHub Pages** 功能并选择 **master branch /docs folder** 选项。



!> 可以将文档放在根目录下，然后选择 **master** 分支 作为文档目录。你需要在部署位置下放一个 **.nojekyll** 文件（比如 /docs 目录或者 gh-pages 分支）

## GitLab Pages

如果你正在部署你的主分支，在 **.gitlab-ci.yml** 中包含以下脚本：

?> **.public** 的解决方法是这样的，**cp** 不会无限循环的将 **public/** 复制到自身。

```
pages:
  stage: deploy
  script:
    - mkdir .public
    - cp -r * .public
```



```
- mv .public public
artifacts:
  paths:
    - public
only:
  - master
```

!> 你可以用 `- cp -r docs/. public` 替换脚本, 如果 `./docs` 是你的 docsify 子文件夹。

## Gitee Pages

在对应的 Gitee 仓库服务中选择 **Gitee Pages** , 选择您要部署的分支, 填写您要部署的分支上的目录, 例如 `docs` , 填写完成之后点击启动即可。

## Firebase 主机

!> 你需要先使用谷歌账号登陆 [Firebase 控制台](#) , 然后使用 `npm i -g firebase-tools` 命令安装 Firebase CLI 。

使用命令行浏览到你的 Firebase 项目目录, 大致是 `~/Projects/Docs` 等等。在这里执行 `firebase init` 命令, 从菜单中选择 **Hosting** (使用空格键选择, 方向键切换选项, 回车键确认。遵照安装说明。

然后你会有个 `firebase.json` 文件, 内容大致如下 (我把部署目录从 `public` 改为 `site` 了):

```
{
  "hosting": {
    "public": "site",
    "ignore": ["firebase.json", "**/.*", "**/node_modules/**"]
  }
}
```

完成后，执行 `docsify init ./site` 构建起始模板（将 `site` 替换为你在运行 `firebase init` 时确定的部署目录 - 默认情况下为 `public` ）。添加/编辑文档，然后在项目根目录执行 `firebase deploy` 。

## VPS

和部署所有静态网站一样，只需将服务器的访问根目录设定为 `index.html` 文件。

例如 `nginx` 的配置

```
server {  
    listen 80;  
    server_name your.domain.com;  
  
    location / {  
        alias /path/to/dir/of/docs/;  
        index index.html;  
    }  
}
```

## Netlify

1. 登陆你的[Netlify](#)账号
2. 在[dashboard](#)页上点击 **New site from Git**
3. 选择那个你用来存储文档的git仓库，将 **Build Command** 留空, 将 **Publish directory** 区域填入你的 `index.html` 所在的目录，例如：填入 `docs` (如果你的 `index.html` 的相对路径是 `docs/index.html` 的话)

## HTML5 路由

当使用HTML5路由时，你需要设置一条将所有请求重定向到你的 `index.html` 的重定向规则。当你使用Netlify时这相当简单，在你的 **Publish Directory**下创建一个 `_redirects` 文件，写进以下内容就可以了 🎉

```
/* /index.html 200
```

## ZEIT Now

1. 安装 [Now CLI](#) : `npm i -g now`
2. 切换到你的 docsify 网站的文档目录, 例如 `cd docs`
3. 用一个指令来部署: `now`

## AWS Amplify

1. 在 Docsify 项目的 `index.html` 中设置 `routerMode` 为 *history* 模式:

```
<script>
  window.$docsify = {
    loadSidebar: true,
    routerMode: 'history'
  }
</script>
```

1. 登录到你的 [AWS 控制台](#)。
2. 到 [AWS Amplify 仪表盘](#)。
3. 选择 **Deploy** 路线来设置你的项目。
4. 若有提示, 如果你希望在项目根目录下保存你的文档, 保持构建设置为空; 如果你想保存文档到其它目录, 修改 `amplify.yml` :

```
version: 0.1
frontend:
  phases:
    build:
      commands:
        - echo "Nothing to build"
  artifacts:
    baseDirectory: /docs
    files:
```

```
- '**/*'
cache:
  paths: []
```

1. 依次添加如下跳转规则。注意第二条的 **PNG** 是图片格式，如果你要使用其它图片格式，可以相应修改。

Source address	Target address	Type
/<*>.md	/<*>.md	200 (Rewrite)
/<*>.png	/<*>.png	200 (Rewrite)
/<*>	/index.html	200 (Rewrite)

## Docker

- 创建 docsify 的文件

你需要准备好初始文件，而不是在容器中制作。请参阅 [快速开始](#) 部分，了解如何手动或使用 [docsify-cli](#) 创建这些文件。

```
index.html
README.md
```

- 创建 Dockerfile

```
FROM node:latest
LABEL description="A demo Dockerfile for build Docsify."
WORKDIR /docs
RUN npm install -g docsify-cli@latest
EXPOSE 3000/tcp
ENTRYPOINT docsify serve .
```

创建成功后当前的目录结构应该是这样的：

```
index.html
```

README.md  
Dockerfile

- 构建 docker 镜像

```
docker build -f Dockerfile -t docsify/demo .
```

- 运行 docker 镜像

```
docker run -itp 3000:3000 --name=docsify -v $(pwd):/docs docsify  
/demo
```

# 文档助手

docsify 扩展了一些 Markdown 语法，可以让文档更易读。

## 强调内容

适合显示重要的提示信息，语法为 `!> 内容`。

```
!> 一段重要的内容，可以和其他 **Markdown** 语法混用。
```

`!>` 一段重要的内容，可以和其他 **Markdown** 语法混用。

## 普通提示

普通的提示信息，比如写 TODO 或者参考内容等。

```
?> _TODO_ 完善示例
```

`?> TODO` 完善示例

## 忽略编译链接

有时候我们会把其他一些相对路径放到链接上，你必须告诉 docsify 你不需要编译这个链接。例如：

```
[link](/demo/)
```

它将被编译为 `<a href="/#/demo/">link</a>` 并将加载 `/demo/README.md`。可能你想跳转到 `/demo/index.html`。

现在你可以做到这一点

```
[link] (/demo/ ':ignore')
```

即将会得到 `<a href="/demo/">link</a>` html 代码。不要担心，你仍然可以为链接设置标题。

```
[link] (/demo/ ':ignore title')  
  
<a href="/demo/" title="title">link</a>
```

## 设置链接的 **target** 属性

```
[link] (/demo ':target=_blank')  
[link] (/demo2 ':target=_self')
```

## 禁用链接

```
[link] (/demo ':disabled')
```

## 跨域链接

只有当你同时设置了 `routerMode: 'history'` 和 `externalLinkTarget: '_self'` 时，你需要为这些跨域链接添加这个配置。

```
[example.com] (https://example.com/ ':crossorigin')
```

## Github 任务列表

```
- [ ] foo
- bar
- [x] baz
- [ ] bam <~ not working
  - [ ] bim
  - [ ] lim
```

- [ ] foo
- bar
- [x] baz
- [ ] bam <~ not working
  - [ ] bim
  - [ ] lim

## 图片处理

### 缩放

```
![logo](https://docsify.js.org/_media/icon.svg ':size=WIDTHxHEIGHT')
![logo](https://docsify.js.org/_media/icon.svg ':size=50x100')
![logo](https://docsify.js.org/_media/icon.svg ':size=100')

<!-- 支持按百分比缩放 -->

![logo](https://docsify.js.org/_media/icon.svg ':size=10%')
```



### 设置图片的 **Class**



```
![logo](https://docsify.js.org/_media/icon.svg ':class=someCssClass')
```

## 设置图片的 ID

```
![logo](https://docsify.js.org/_media/icon.svg ':id=someCssId')
```

## 设置标题的 id 属性

```
### 你好，世界! :id=hello-world
```

## html 标签中的 Markdown

你需要在 html 和 Markdown 内容中插入空行。当你在 `details` 元素中渲染 Markdown 时很有用。

```
<details>
<summary>自我评价（点击展开）</summary>

- Abc
- Abc

</details>
```

► 自我评价（点击展开）

Markdown 内容也可以被 html 标签包裹。

```
<div style='color: red'>

- listitem
- listitem
- listitem
```

```
</div>
```

- Abc - Abc

# CDN

推荐使用 [jsDelivr](#)，能及时获取到最新版。你也可以在[cdn.jsdelivr.net/npm/docsify/](https://cdn.jsdelivr.net/npm/docsify/)中浏览npm包的源代码。

## 获取最新版本

不指定特定版本号时将引入最新版。

```
<!-- 引入 css -->
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify/themes/vue.css">

<!-- 引入 script -->
<script src="//cdn.jsdelivr.net/npm/docsify/lib/docsify.js"></script>
```

也可以使用 [压缩版文件](#)。

## 获取指定版本

如果担心频繁地版本更新又可能引入未知 **Bug**，我们也可以使用具体的版本。规则是 `//cdn.jsdelivr.net/npm/docsify@VERSION/`

```
<!-- 引入 css -->
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify@4.10.2/themes/vue.css">

<!-- 引入 script -->
<script src="//cdn.jsdelivr.net/npm/docsify@4.10.2/lib/docsify.js"></script>
```

!> 指定 *VERSION* 为 `latest` 可以强制每次都请求最新版本。

## 压缩版

CSS 的压缩文件位于 `/lib/themes/` 目录下，JS 的压缩文件是原有文件路径的基础上加 `.min` 后缀。

```
<!-- 引入 css -->
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify/lib/
themes/vue.css">

<!-- 引入 script -->
<script src="//cdn.jsdelivr.net/npm/docsify/lib/docsify.min.js">
</script>
```

```
<!-- 引入 css -->
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify@4.10
.2/lib/themes/vue.css">

<!-- 引入 script -->
<script src="//cdn.jsdelivr.net/npm/docsify@4.10.2/lib/docsify.m
in.js"></script>
```

## 其他 CDN

- <https://www.bootcdn.cn/docsify/> (支持国内)
- <https://cdn.jsdelivr.net/npm/docsify/> (国内外都支持)
- <https://cdnjs.com/libraries/docsify>
- <https://unpkg.com/browse/docsify/>

# 离线模式

**Progressive Web Apps(PWA)** 是一项融合 **Web** 和 **Native** 应用各项优点的解决方案。我们可以利用其支持离线功能的特点，让我们的网站可以在信号差或者离线状态下正常运行。要使用它也非常容易。

## 创建 **serviceWorker**

这里已经整理好了一份代码，你只需要在网站根目录下创建一个 `sw.js` 文件，并粘贴下面的代码。

**sw.js**

```
/* =====
 * docsify sw.js
 * =====
 * Copyright 2016 @huxpro
 * Licensed under Apache 2.0
 * Register service worker.
 * ===== */

const RUNTIME = 'docsify'
const HOSTNAME_WHITELIST = [
  self.location.hostname,
  'fonts.gstatic.com',
  'fonts.googleapis.com',
  'cdn.jsdelivr.net'
]

// The Util Function to hack URLs of intercepted requests
const getFixedUrl = (req) => {
  var now = Date.now()
  var url = new URL(req.url)

  // 1. fixed http URL
  // Just keep syncing with location.protocol
```

```

    // fetch(httpURL) belongs to active mixed content.
    // And fetch(httpRequest) is not supported yet.
    url.protocol = self.location.protocol

    // 2. add query for caching-busting.
    // Github Pages served with Cache-Control: max-age=600
    // max-age on mutable content is error-prone, with SW life of
    bugs can even extend.
    // Until cache mode of Fetch API landed, we have to workaround
    cache-busting with query string.
    // Cache-Control-Bug: https://bugs.chromium.org/p/chromium/iss
    ues/detail?id=453190
    if (url.hostname === self.location.hostname) {
        url.search += (url.search ? '&' : '?') + 'cache-bust=' + now
    }
    return url.href
}

/**
 * @Lifecycle Activate
 * New one activated when old isnt being used.
 *
 * waitUntil(): activating ==> activated
 */
self.addEventListener('activate', event => {
    event.waitUntil(self.clients.claim())
})

/**
 * @Functional Fetch
 * All network requests are being intercepted here.
 *
 * void respondWith(Promise<Response> r)
 */
self.addEventListener('fetch', event => {
    // Skip some of cross-origin requests, like those for Google A
    nalytics.
    if (HOSTNAME_WHITELIST.indexOf(new URL(event.request.url).host
    name) > -1) {
        // Stale-while-revalidate
        // similar to HTTP's stale-while-revalidate: https://www.mno
        t.net/blog/2007/12/12/stale

```

```

// Upgrade from Jake's to Surma's: https://gist.github.com/surma/eb441223daaedf880801ad80006389f1
const cached = caches.match(event.request)
const fixedUrl = getFixedUrl(event.request)
const fetched = fetch(fixedUrl, { cache: 'no-store' })
const fetchedCopy = fetched.then(resp => resp.clone())

// Call respondWith() with whatever we get first.
// If the fetch fails (e.g disconnected), wait for the cache.

// If there's nothing in cache, wait for the fetch.
// If neither yields a response, return offline pages.
event.respondWith(
  Promise.race([fetched.catch(_ => cached), cached])
    .then(resp => resp || fetched)
    .catch(_ => { /* eat any errors */ })
)

// Update the cache with the version we fetched (only for ok status)
event.waitUntil(
  Promise.all([fetchedCopy, caches.open(RUNTIME)])
    .then(([response, cache]) => response.ok && cache.put(event.request, response))
    .catch(_ => { /* eat any errors */ })
)
})
})

```

## 注册

现在，到 `index.html` 里注册它。这个功能只能工作在一些现代浏览器上，所以我们需要加个判断。

*index.html*

```

<script>
  if (typeof navigator.serviceWorker !== 'undefined') {

```

```
    navigator.serviceWorker.register('sw.js')
  }
</script>
```

## 体验一下

发布你的网站，并开始享受离线模式的魔力吧！:ghost: 当然你现在看到的 docsify 的文档网站已经支持离线模式了，你可以关掉 Wi-Fi 体验一下。



# 服务端渲染（SSR）

先看例子 <https://docsify.now.sh>

项目地址在 <https://github.com/docsifyjs/docsify-ssr-demo>



文档依旧是部署在 **GitHub Pages** 上，**Node** 服务部署在 **now.sh** 里，渲染的内容是从 **GitHub Pages** 上同步过来的。所以静态部署文档的服务器和服务端渲染的 **Node** 服务器是分开的，也就是说你还是可以用之前的方式更新文档，并不需要每次都部署。

## 什么是 SSR?

- 更好的 SEO
- 更酷的感觉

## 快速开始

如果你熟悉 `now` 的使用，接下来的介绍就很简单了。先创建一个新项目，并安装 `now` 和 `docsify-cli`。

```
npm i now docsify-cli -D
```

编辑 `package.json`。假设你的文档放在 `./docs` 子目录。

```
{
  "name": "my-project",
  "scripts": {
    "start": "docsify start . -c ssr.config.js",
    "deploy": "now -p"
  },
  "files": [
```

```

    "docs"
  ],
  "docsify": {
    "config": {
      "basePath": "https://docsify.js.org/",
      "loadSidebar": true,
      "loadNavbar": true,
      "coverpage": true,
      "name": "docsify"
    }
  }
}

```

!> 其中 `basePath` 相当于 `webpack` 的 `publicPath`，为文档所在的路径，可以填你的 `docsify` 文档网站。我们可以使用本地或者远程文件。

配置好了以后，我们可以在本地预览。

```

npm start

# open http://localhost:4000

```

发布！

```

now -p

```

现在，你有一个支持服务端渲染的文档网站了。

## 定制模板

你可以提供一个整页模板，例如：

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">

```

```

<title>docsify</title>
<meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
<link rel="stylesheet" href="//cdn.jsdelivr.net/npm/docsify/themes/vue.css" title="vue">
</head>
<body>
  <!--inject-app-->
  <!--inject-config-->
  <script src="//cdn.jsdelivr.net/npm/docsify/lib/docsify.js"></script>
  <script src="//cdn.jsdelivr.net/npm/docsify/lib/plugins/search.js"></script>
  <script src="//cdn.jsdelivr.net/npm/prismjs/components/prism-bash.min.js"></script>
  <script src="//cdn.jsdelivr.net/npm/prismjs/components/prism-markdown.min.js"></script>
  <script src="//cdn.jsdelivr.net/npm/prismjs/components/prism-nginx.min.js"></script>
</body>
</html>

```

模板可以包含占位符，会自动将渲染后的 `html` 和配置内容注入到页面上。

- `<!--inject-app-->`
- `<!--inject-config-->`

## 配置

配置可以单独写在配置文件内，然后通过 `--config config.js` 加载，或者写在 `package.json` 中。

渲染的基础模版也可以自定义，配置在 `template` 属性上。

```

module.exports = {
  template: './ssr.html',
  maxAge: 60 * 60 * 1000, // lru-cache 设置
  config: {
    // docsify 设置
  }
}

```

```
}  
}
```

## 更多玩法

你可以直接在你的 **Node** 服务器上执行 `docsify start` 。

`docsify start` 其实是依赖了 `docsify-server-renderer` 模块，如果你感兴趣，你完全可以用它自己实现一个 **server**，可以加入缓存等功能。

```
var Renderer = require('docsify-server-renderer')  
var readFileSync = require('fs').readFileSync  
  
// init  
var renderer = new Renderer({  
  template: readFileSync('./docs/index.template.html', 'utf-8'),  
  config: {  
    name: 'docsify',  
    repo: 'docsifyjs/docsify'  
  }  
})  
  
renderer.renderToString(url)  
  .then(html => {})  
  .catch(err => {})
```

当然文档文件和 **server** 也是可以部署在一起的，`basePath` 不是一个 **URL** 的话就会当做文件路径处理，也就是从服务器上加载资源。

# 文件嵌入

docsify 4.6 开始支持嵌入任何类型的文件到文档里。你可以将文件当成 `iframe` 、 `video` 、 `audio` 或者 `code block` ，如果是 Markdown 文件，甚至可以直接插入到当前文档里。

这是一个嵌入 Markdown 文件的例子。

```
[filename](../_media/example.md ':include')
```

`example.md` 文件的内容将会直接显示在这里

[filename](#)

你可以查看 [example.md](#) 原始内容对比效果。

通常情况下，这样的语法将会被当作链接处理。但是在 docsify 里，如果你添加一个 `:include` 选项，它就会被当作文件嵌入。

## 嵌入的类型

当前，嵌入的类型是通过文件后缀自动识别的，这是目前支持的类型：

- **iframe** `.html` , `.htm`
- **markdown** `.markdown` , `.md`
- **audio** `.mp3`
- **video** `.mp4` , `.ogg`
- **code** other file extension

当然，你也可以强制设置嵌入类型。例如你想将 Markdown 文件当作一个 `code block` 嵌入。

```
[filename](../_media/example.md ':include :type=code')
```

你会看到：

filename

## 嵌入代码片段

有时候你并不想嵌入整个文件，可能你只想要其中的几行代码，但你还要在 CI 系统中编译和测试该文件。

```
[filename](../_media/example.js ':include :type=code :fragment=demo')
```

在你的代码文件中，你需要用斜线 `/// [demo]` 包裹该片段（片段的前后都要有）。你也可以使用 `### [demo]` 来包裹。

示例：

filename

## 标签属性

如果你嵌入文件是一个 `iframe`、`audio` 或者 `video`，你可以给这些标签设置属性。

```
[cinwell website](https://cinwell.com ':include :type=iframe width=100% height=400px')
```

cinwell website

看见没？你只需要直接写属性就好了，每个标签有哪些属性建议你查看 [MDN 文档](#)。

## 代码块高亮

如果是嵌入一个代码块，你可以设置高亮的语言，或者让它自动识别。这里是手动设置高亮语言

```
[](../_media/example.html ':include :type=code text')
```



?> 如何高亮代码？你可以查看[这份文档](#)。

## 嵌入Gist

你可以将 **Gist** 作为 **Markdown** 内容或代码块嵌入。这是基于[嵌入文件](#)部分开头的方法，不过是嵌入一个原始的 **Gist URL**。

?> 这里不需要插件或修改配置来使其工作。事实上，即使你使用插件或修改配置来允许加载外部脚本，从 **Gist** 复制的 "Embed" `script` 标签也无法加载。

## 确定Gist的元数据

从查看 `gist.github.com` 上的 **Gist** 开始。在本指南中，我们使用这个 **Gist**：

- <https://gist.github.com/anikethsaha/f88893bb563bb7229d6e575db53a8c15>

从 **Gist** 中找出以下内容：

字段	示例	说明
<b>Username</b>	<code>anikethsaha</code>	<b>Gist</b> 的作者
<b>Gist ID</b>	<code>c2bece08f27c4277001f123898d16a7c</code>	<b>Gist</b> 的标识符。 该标识在 <b>Gist</b> 的有效期内是固定的
		在 <b>Gist</b> 中选择一

<b>Filename</b>	<code>content.md</code>	个文件名。即使是单文件的 <b>Gist</b> ，也需要这样做才能嵌入
-----------------	-------------------------	--------------------------------------

你将需要这些来为目标文件建立 *raw gist URL*。它的格式如下：

- `https://gist.githubusercontent.com/USERNAME/GIST_ID/raw/FILENAME`

下面是根据示例 **Gist** 举出的两个例子：

- <https://gist.githubusercontent.com/anikethsaha/f88893bb563bb7229d6e575db53a8c15/raw/content.md>
- <https://gist.githubusercontent.com/anikethsaha/f88893bb563bb7229d6e575db53a8c15/raw/script.js>

?> 另外你也可以直接点击 **Gist** 文件上的 **Raw** 按钮来获取原始 **URL**。但是如果你使用这种方法，请确保删除 `raw/` 和文件名之间的修订号，这样 **URL** 就会与上面的模式一致。否则当更新 **Gist** 时，你嵌入的 **Gist** 将不会显示最新的内容。

继续下面的一个部分，将 **Gist** 嵌入到 **Docsify** 页面上。

## 渲染 **Gist** 中的 **Markdown** 内容

这是一个很好的方法，可以将内容无缝地嵌入到你的文档中，而不需要将别人发送到外部链接。这种方法很适合在多个仓库的文档网站上重复使用一个 **Gist**，比如安装说明。这种方法同样适用于您的账户或其他用户拥有的 **Gist**。

格式：

```
[LABEL](https://gist.githubusercontent.com/USERNAME/GIST_ID/raw/FILENAME ':include')
```

例如：



```
[gist: content.md](https://gist.githubusercontent.com/anikethsaha/f88893bb563bb7229d6e575db53a8c15/raw/content.md ':include')
```

你会看到：

[gist: content.md](#)

**LABEL** 可以是任何你想要的文本。如果链接被破坏，它可以作为一个 *fallback* 信息。所以在这里重复文件名是很有用的，万一你需要修复一个破坏的链接。它还可以使嵌入的元素一目了然。

## 渲染 **Gist** 中的代码块

格式与上一节相同，但是在 **alt** 文本中添加了 `:type=code`。与 [嵌入的类型](#) 部分一样，语法高亮将从扩展名(如 `.js` 或 `.py`)中推断，所以你可以将 `type` 设置为 `code`。

格式：

```
[LABEL](https://gist.githubusercontent.com/USERNAME/GIST_ID/raw/FILENAME ':include :type=code')
```

例如：

```
[gist: script.js](https://gist.githubusercontent.com/anikethsaha/f88893bb563bb7229d6e575db53a8c15/raw/script.js ':include :type=code')
```

你会看到：

[gist: script.js](#)