

Table of Contents

hexo中文文档	1.1
开始使用	1.2
概述	1.2.1
建站	1.2.2
配置	1.2.3
命令	1.2.4
迁移	1.2.5
基本操作	1.3
写作	1.3.1
Front-matter	1.3.2
标签插件	1.3.3
资源文件夹	1.3.4
数据文件夹	1.3.5
服务器	1.3.6
生成器	1.3.7
部署	1.4
Github Pages	1.4.1
Github Pages	1.4.2
一键部署	1.4.3
自定义	1.5
永久链接	1.5.1
主题	1.5.2
模板	1.5.3
变量	1.5.4
辅助函数	1.5.5
国际化(i18n)	1.5.6
插件	1.5.7
其他	1.6
问题解答	1.6.1
贡献	1.6.2

hexo中文文档

欢迎使用 **Hexo**，本文档将帮助您快速上手。如果您在使用过程中遇到问题，请查看 [问题解答](#) 中的解答，或者在 [GitHub](#)、[Google Group](#) 上提问。

什么是 **Hexo**？

Hexo 是一个快速、简洁且高效的博客框架。**Hexo** 使用 [Markdown](#)（或其他渲染引擎）解析文章，在几秒内，即可利用靓丽的主题生成静态网页。

文档

欢迎使用 Hexo，本文档将帮助您快速上手。如果您在使用过程中遇到问题，请查看 [问题解答](#) 中的解答，或者在 [GitHub](#)、[Google Group](#) 上提问。

什么是 Hexo?

Hexo 是一个快速、简洁且高效的博客框架。Hexo 使用 [Markdown](#)（或其他渲染引擎）解析文章，在几秒内，即可利用靓丽的主题生成静态网页。

安装

安装 Hexo 只需几分钟时间，若您在安装过程中遇到问题或无法找到解决方式，请[提交问题](#)，我会尽力解决您的问题。

安装前提

安装 Hexo 相当简单，只需要先安装下列应用程序即可：

- [Node.js](#) (Node.js 版本需不低于 8.6，建议使用 Node.js 10.0 及以上版本)
- [Git](#)

如果您的电脑中已经安装上述必备程序，那么恭喜您！接下来只需要使用 npm 即可完成 Hexo 的安装。

```
$ npm install -g hexo-cli
```

如果您的电脑中尚未安装所需要的程序，请根据以下安装指示完成安装。

Mac 用户

您在编译时可能会遇到问题，请先到 App Store 安装 Xcode，Xcode 完成后，启动并进入 **Preferences -> Download -> Command Line Tools -> Install** 安装命令行工具。

For Mac / Linux 用户

如果你是 macOS 用户，或者是通过软件管理器从默认软件仓库安装 Node.js 的 Linux 用户，在使用 npm 的 `-g` 参数时可能会遇到一些权限相关的问题。请遵循 [由 npmjs 发布的指导](#) 修复该问题，并且不要使用 `root`、`sudo` 等方法覆盖权限

安装 Git

- Windows: 下载并安装 [git](#).
- Mac: 使用 [Homebrew](#), [MacPorts](#) : `brew install git` ;或下载 [安装程序](#) 安装。
- Linux (Ubuntu, Debian): `sudo apt-get install git-core`
- Linux (Fedora, Red Hat, CentOS): `sudo yum install git-core`

Windows 用户

由于众所周知的原因，从上面的链接下载git for windows最好挂上一个代理，否则下载速度十分缓慢。也可以参考[这个页面](#)，收录了存储于百度网盘的下载地址。

安装 Node.js

安装 Node.js 的最佳方式是使用 [nvm](#)。nvm 的开发者提供了一个自动安装 nvm 的简单脚本：

cURL:

```
$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | sh
```

Wget:

```
$ wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | sh
```

安装完成后，重启终端并执行下列命令即可安装 **Node.js**。

```
$ nvm install node
```

Windows 用户

对于 Windows 用户我们推荐使用 **nvs** 而不是 **nvm**。**nvs** 具有和 **nvm** 相似的命令行语法，并且可以通过 Windows Installer (MSI) 安装包进行安装。

或者您也可以下载 [安装程序](#) 来安装。

Windows 用户

对于windows用户来说，建议使用安装程序进行安装。安装时，请勾选**Add to PATH**选项。另外，您也可以使用**Git Bash**，这是git for windows自带的一组程序，提供了Linux风格的shell，在该环境下，您可以直接用上面提到的命令来安装**Node.js**。打开它的方法很简单，在任意位置单击右键，选择“Git Bash Here”即可。由于Hexo的很多操作都涉及到命令行，您可以考虑始终使用**Git Bash**来进行操作。

Linux 用户

大部分 Linux 发行版都会在它们的默认软件包仓库中分发 **Node.js**。第三方仓库 **NodeSource** 通常能分发最新版本的 **Node.js**。

安装 Hexo

所有必备的应用程序安装完成后，即可使用 **npm** 安装 **Hexo**。

```
$ npm install -g hexo-cli
```

建站

安装 Hexo 完成后，请执行下列命令，Hexo 将会在指定文件夹中新建所需要的文件。

```
$ hexo init <folder>
$ cd <folder>
$ npm install
```

新建完成后，指定文件夹的目录如下：

```
.
├─ _config.yml
├─ package.json
├─ scaffolds
├─ source
│   └─ _drafts
│       └─ _posts
└─ themes
```

_config.yml

网站的 [配置](#) 信息，您可以在这里配置大部分的参数。

package.json

应用程序的信息。[EJS](#), [Stylus](#) 和 [Markdown renderer](#) 已默认安装，您可以自由移除。

```
package.json
{
  "name": "hexo-site",
  "version": "0.0.0",
  "private": true,
  "hexo": {
    "version": ""
  },
  "dependencies": {
    "hexo": "^3.8.0",
    "hexo-generator-archive": "^0.1.5",
    "hexo-generator-category": "^0.1.3",
    "hexo-generator-index": "^0.2.1",
    "hexo-generator-tag": "^0.2.0",
    "hexo-renderer-ejs": "^0.3.1",
    "hexo-renderer-stylus": "^0.3.3",
    "hexo-renderer-marked": "^0.3.2",
    "hexo-server": "^0.3.3"
  }
}
```

scaffolds

[模版](#) 文件夹。当您新建文章时，Hexo 会根据 scaffold 来建立文件。

Hexo的模板是指在新建的文章文件中默认填充的内容。例如，如果您修改scaffold/post.md中的Front-matter内容，那么每次新建一篇文章时都会包含这个修改。

source

资源文件夹是存放用户资源的地方。除 `_posts` 文件夹之外，开头命名为 `_` (下划线)的文件 / 文件夹和隐藏的文件将会被忽略。Markdown 和 HTML 文件会被解析并放到 `public` 文件夹，而其他文件会被拷贝过去。

themes

[主题](#) 文件夹。Hexo 会根据主题来生成静态页面。

配置

您可以在 `_config.yml` 中修改大部分的配置。

网站

参数	描述
<code>title</code>	网站标题
<code>subtitle</code>	网站副标题
<code>description</code>	网站描述
<code>keywords</code>	网站的关键词。使用半角逗号 ， 分隔多个关键词。
<code>author</code>	您的名字
<code>language</code>	网站使用的语言
<code>timezone</code>	网站时区。Hexo 默认使用您电脑的时区。 时区列表 。比如说: <code>America/New_York</code> , <code>Japan</code> , 和 <code>UTC</code> 。

其中, `description` 主要用于SEO, 告诉搜索引擎一个关于您站点的简单描述, 通常建议在其中包含您网站的关键词。 `author` 参数用于主题显示文章的作者。

网址

参数	描述	默认值
<code>url</code>	网址	
<code>root</code>	网站根目录	
<code>permalink</code>	文章的 永久链接 格式	<code>:year/:month/:day/:title/</code>
<code>permalink_defaults</code>	永久链接中各部分的默认值	
<code>pretty_urls</code>	改写 permalink 的值来美化 URL	
<code>pretty_urls.trailing_index</code>	是否在永久链接中保留尾部的 <code>index.html</code> , 设置为 <code>false</code> 时去除	<code>true</code>

网站存放在子目录

如果您的网站存放在子目录中, 例如 `http://yoursite.com/blog` , 则请将您的 `url` 设为 `http://yoursite.com/blog` 并把 `root` 设为 `/blog/` 。

例如:

```
# 比如, 一个页面的永久链接是 http://example.com/foo/bar/index.html
pretty_urls:
  trailing_index: false
# 此时页面的永久链接会变为 http://example.com/foo/bar/
```

目录

参数	描述	默认值
<code>source_dir</code>	资源文件夹, 这个文件夹用来存放内容。	<code>source</code>
<code>public_dir</code>	公共文件夹, 这个文件夹用于存放生成的站点文件。	<code>public</code>

<code>tag_dir</code>	标签文件夹	<code>tags</code>
<code>archive_dir</code>	归档文件夹	<code>archives</code>
<code>category_dir</code>	分类文件夹	<code>categories</code>
<code>code_dir</code>	Include code 文件夹， <code>source_dir</code> 下的子目录	<code>downloads/code</code>
<code>i18n_dir</code>	国际化（i18n）文件夹	<code>:lang</code>
<code>skip_render</code>	跳过指定文件的渲染。匹配到的文件将会被不做改动地复制到 <code>public</code> 目录中。您可使用 glob 表达式 来匹配路径。	

例如：

```
skip_render: "mypage/**/*"
# 将会直接将 `source/mypage/index.html` 和 `source/mypage/code.js` 不做改动地输出到 'public' 目录
# 你也可以用这种方法来跳过对指定文章文件的渲染
skip_render: "_posts/test-post.md"
# 这将会忽略对 'test-post.md' 的渲染skip_render: "mypage/**/*"# 将会直接将 `source/mypage/index.html` 和 `source
/mypage/code.js` 不做改动地输出到 'public' 目录# 你也可以用这种方法来跳过对指定文章文件的渲染skip_render: "_posts/t
est-post.md"# 这将会忽略对 'test-post.md' 的渲染
```

提示

如果您刚刚开始接触 Hexo，通常没有必要修改这一部分的值。

文章

参数	描述	默认值
<code>new_post_name</code>	新文章的文件名称	<code>:title.md</code>
<code>default_layout</code>	预设布局	<code>post</code>
<code>auto_spacing</code>	在中文和英文之间加入空格	<code>false</code>
<code>titlecase</code>	把标题转换为 title case	<code>false</code>
<code>external_link</code>	在新标签中打开链接	<code>true</code>
<code>external_link.enable</code>	在新标签中打开链接	<code>true</code>
<code>external_link.field</code>	对整个网站（ <code>site</code> ）生效或仅对文章（ <code>post</code> ）生效	<code>site</code>
<code>external_link.exclude</code>	Exclude hostname. Specify subdomain when applicable, including <code>www</code>	<code>[]</code>
<code>filename_case</code>	把文件名称转换为 (1) 小写或 (2) 大写	<code>0</code>
<code>render_drafts</code>	显示草稿	<code>false</code>
<code>post_asset_folder</code>	启动 Asset 文件夹	<code>false</code>
<code>relative_link</code>	把链接改为与根目录的相对位址	<code>false</code>
<code>future</code>	显示未来的文章	<code>true</code>
<code>highlight</code>	代码块的设置	
<code>highlight.enable</code>	开启代码块高亮	<code>true</code>
<code>highlight.auto_detect</code>	如果未指定语言，则启用自动检测	<code>false</code>
<code>highlight.line_number</code>	显示行数	<code>true</code>
<code>highlight.tab_replace</code>	用 <code>n</code> 个空格替换 <code>tabs</code> ；如果值为空，则不会替换 <code>tabs</code>	<code>''</code>

相对地址

默认情况下，Hexo 生成的超链接都是绝对地址。例如，如果您的网站域名为 `example.com`，您有一篇文章名为 `hello`，那么绝对链接可能像这样：`http://example.com/hello.html`，它是绝对于域名的。相对链接像这样：`/hello.html`，也就是说，无论用什么域名访问该站点，都没有关系，这在进行反向代理时可能用到。通常情况下，建议使用绝对地址。

分类 & 标签

参数	描述	默认值
<code>default_category</code>	默认分类	<code>uncategorized</code>
<code>category_map</code>	分类别名	
<code>tag_map</code>	标签别名	

日期 / 时间格式

Hexo 使用 [Moment.js](#) 来解析和显示时间。

参数	描述	默认值
<code>date_format</code>	日期格式	<code>YYYY-MM-DD</code>
<code>time_format</code>	时间格式	<code>HH:mm:ss</code>
<code>use_date_for_updated</code>	Use the date of the post in <code>post.updated</code> if no updated date is provided in the front-matter. Typically used with Git workflow	<code>true</code>

分页

参数	描述	默认值
<code>per_page</code>	每页显示的文章量 (0 = 关闭分页功能)	<code>10</code>
<code>pagination_dir</code>	分页目录	<code>page</code>

扩展

参数	描述
<code>theme</code>	当前主题名称。值为 <code>false</code> 时禁用主题
<code>theme_config</code>	主题的配置文件。在这里放置的配置会覆盖主题目录下的 <code>_config.yml</code> 中的配置
<code>deploy</code>	部署部分的设置
<code>meta_generator</code>	Meta generator 标签。值为 <code>false</code> 时 Hexo 不会在头部插入该标签

包括或不包括目录和文件

在 Hexo 配置文件中，通过设置 `include/exclude` 可以让 Hexo 进行处理或忽略某些目录和文件夹。你可以使用 [glob 表达式](#) 对目录和文件进行匹配。

参数	描述
<code>include</code>	Hexo 默认会忽略隐藏文件和文件夹（包括名称以下划线和 <code>.</code> 开头的文件和文件夹，Hexo 的 <code>_posts</code> 和 <code>_data</code> 等目录除外）。通过设置此字段将使 Hexo 处理它们并将它们复制到 <code>source</code> 目录下。
<code>exclude</code>	Hexo 会忽略这些文件和目录

举例：

```
# Include/Exclude Files/Folders
include:
  - ".nojekyll"
  # 包括 'source/css/_typing.css'
  - "css/_typing.css"
  # 包括 'source/_css/' 中的任何文件，但不包括子目录及其其中的文件。
  - "_css/*"
  # 包含 'source/_css/' 中的任何文件和子目录下的任何文件
  - "_css/**/*"

exclude:
  # 不包括 'source/js/test.js'
  - "js/test.js"
  # 不包括 'source/js/' 中的文件、但包括子目录下的所有目录和文件
  - "js/*"
  # 不包括 'source/js/' 中的文件和子目录下的任何文件
  - "js/**/*"
  # 不包括 'source/js/' 目录下的所有文件名以 'test' 开头的文件，但包括其它文件和子目录下的单文件
  - "js/test*"
  # 不包括 'source/js/' 及其子目录中任何以 'test' 开头的文件
  - "js/**/test*"
  # 不要用 exclude 来忽略 'source/_posts/' 中的文件。你应该使用 'skip_render'，或者在要忽略的文件的文件名之前加一个下划线 '_'
  # 在这里配置一个 - "_posts/hello-world.md" 是没有用的。
```

列表中的每一项都必须用单引号或双引号包裹起来。

`include` 和 `exclude` 并不适用于 `themes/` 目录下的文件。如果需要忽略 `themes/` 目录下的部分文件或文件夹，可以在文件名之前添加下划线 `_`。

使用代替配置文件

可以在 `hexo-cli` 中使用 `--config` 参数来指定自定义配置文件的路径。你可以使用一个 `YAML` 或 `JSON` 文件的路径，也可以使用逗号分隔（无空格）的多个 `YAML` 或 `JSON` 文件的路径。例如：

```
# use 'custom.yml' in place of '_config.yml'
$ hexo server --config custom.yml

# use 'custom.yml' & 'custom2.json', prioritizing 'custom3.yml', then 'custom2.json'
$ hexo generate --config custom.yml,custom2.json,custom3.yml
```

当你指定了多个配置文件以后，`Hexo` 会按顺序将这部分配置文件合并成一个 `_multiconfig.yml`。如果遇到重复的配置，排在后面的文件的配置会覆盖排在前面的文件的配置。这个原则适用于任意数量、任意深度的 `YAML` 和 `JSON` 文件。

例如，使用 `--options` 指定了两个自定义配置文件：

```
$ hexo generate --config custom.yml,custom2.json
```

如果 `custom.yml` 中指定了 `foo: bar`，在 `custom2.json` 中指定了 `"foo": "dinosaur"`，那么在 `_multiconfig.yml` 中你会得到 `foo: dinosaur`。

覆盖主题配置

通常情况下，`Hexo` 主题是一个独立的项目，并拥有一个独立的 `_config.yml` 配置文件。你可以在站点的 `_config.yml` 配置文件中配置你的主题，这样你就不需要 `fork` 一份主题并维护主题独立的配置文件。

以下是一个覆盖主题配置的例子：

```
# _config.yml
theme_config:
  bio: "My awesome bio"
```

```
# themes/my-theme/_config.yml
bio: "Some generic bio"
logo: "a-cool-image.png"
```

最终主题配置的输出是：

```
{
  bio: "My awesome bio",
  logo: "a-cool-image.png"
}
```

指令

init

```
$ hexo init [folder]
```

新建一个网站。如果没有设置 `folder`，Hexo 默认在目前的文件夹建立网站。

new

```
$ hexo new [layout] <title>
```

新建一篇文章。如果没有设置 `layout` 的话，默认使用 `_config.yml` 中的 `default_layout` 参数代替。如果标题包含空格的话，请使用引号括起来。

```
$ hexo new "post title with whitespace"
```

参数	描述
<code>-p</code> ， <code>--path</code>	自定义新文章的路径
<code>-r</code> ， <code>--replace</code>	如果存在同名文章，将其替换
<code>-s</code> ， <code>--slug</code>	文章的 Slug ，作为新文章的文件名和发布后的 URL

默认情况下，Hexo 会使用文章的标题来决定文章文件的路径。对于独立页面来说，Hexo 会创建一个以标题为名字的目录，并在目录中放置一个 `index.md` 文件。你可以使用 `--path` 参数来覆盖上述行为、自行决定文件的目录：

```
hexo new page --path about/me "About me"
```

以上命令会创建一个 `source/about/me.md` 文件，同时 **Front Matter** 中的 **title** 为 `"About me"`

注意！**title** 是必须指定的！如果你这么做并不能达到你的目的：

```
hexo new page --path about/me
```

此时 Hexo 会创建 `source/_posts/about/me.md`，同时 `me.md` 的 **Front Matter** 中的 **title** 为 `"page"`。这是因为在上述命令中，`hexo-cli` 将 `page` 视为指定文章的标题、并采用默认的 `layout`。

generate

```
$ hexo generate
```

生成静态文件。

选项	描述
<code>-d</code> ， <code>--deploy</code>	文件生成后立即部署网站
<code>-w</code> ， <code>--watch</code>	监视文件变动
<code>-b</code> ， <code>--</code>	生成过程中如果发生任何未处理的异常则抛出异常

<code>bail</code>	生成过程中如果发生任何未处理的异常则抛出异常
<code>-f</code> , <code>--force</code>	强制重新生成文件 Hexo 引入了差分机制, 如果 <code>public</code> 目录存在, 那么 <code>hexo g</code> 只会重新生成改动的文件。使用该参数的效果接近 <code>hexo clean</code> && <code>hexo generate</code>
<code>-c</code> , <code>--concurrency</code>	最大同时生成文件的数量, 默认无限制

该命令可以简写为

```
$ hexo g
```

publish

```
$ hexo publish [layout] <filename>
```

发表草稿。

server

```
$ hexo server
```

启动服务器。默认情况下, 访问网址为: `http://localhost:4000/`。

选项	描述
<code>-p</code> , <code>--port</code>	重设端口
<code>-s</code> , <code>--static</code>	只使用静态文件
<code>-l</code> , <code>--log</code>	启动日记记录, 使用覆盖记录格式

deploy

```
$ hexo deploy
```

部署网站。

参数	描述
<code>-g</code> , <code>--generate</code>	部署之前预先生成静态文件

该命令可以简写为:

```
$ hexo d
```

render

```
$ hexo render <file1> [file2] ...
```

渲染文件。

参数	描述
<code>-o</code> , <code>--output</code>	设置输出路径

migrate

```
$ hexo migrate <type>
```

从其他博客系统 [迁移内容](#)。

clean

```
$ hexo clean
```

清除缓存文件 (`db.json`) 和已生成的静态文件 (`public`)。

在某些情况（尤其是更换主题后），如果发现您对站点的更改无论如何也不生效，您可能需要运行该命令。

list

```
$ hexo list <type>
```

列出网站资料。

version

```
$ hexo version
```

显示 Hexo 版本。

选项

安全模式

```
$ hexo --safe
```

在安全模式下，不会载入插件和脚本。当您在安装新插件遭遇问题时，可以尝试以安全模式重新执行。

调试模式

```
$ hexo --debug
```

在终端中显示调试信息并记录到 `debug.log` 。当您碰到问题时，可以尝试用调试模式重新执行一次，并 [提交调试信息到 GitHub](#)。

简洁模式

```
$ hexo --silent
```

隐藏终端信息。

自定义配置文件的路径

```
# 使用 custom.yml 代替默认的 _config.yml
$ hexo server --config custom.yml

# 使用 custom.yml 和 custom2.json, 其中 custom2.json 优先级更高
$ hexo generate --config custom.yml,custom2.json,custom3.yml
```

自定义配置文件的路径，指定这个参数后将不再使用默认的 `_config.yml`。你可以使用一个 YAML 或 JSON 文件的路径，也可以使用逗号分隔（无空格）的多个 YAML 或 JSON 文件的路径。例如：

```
# 使用 custom.yml 代替默认的 _config.yml
$ hexo server --config custom.yml

# 使用 custom.yml, custom2.json 和 custom3.yml, 其中 custom3.yml 优先级最高，其次是 custom2.json
$ hexo generate --config custom.yml,custom2.json,custom3.yml
```

当你指定了多个配置文件以后，Hexo 会按顺序将这部分配置文件合并成一个 `_multiconfig.yml`。如果遇到重复的配置，排在后面的文件的配置会覆盖排在前面的文件的配置。这个原则适用于任意数量、任意深度的 YAML 和 JSON 文件。

显示草稿

```
$ hexo --draft
```

显示 `source/_drafts` 文件夹中的草稿文章。

自定义 CWD

```
$ hexo --cwd /path/to/cwd
```

自定义当前工作目录（Current working directory）的路径。

迁移

RSS

首先，安装 `hexo-migrator-rss` 插件。

```
$ npm install hexo-migrator-rss --save
```

插件安装完成后，执行下列命令，从 RSS 迁移所有文章。`source` 可以是文件路径或网址。

```
$ hexo migrate rss <source>
```

Jekyll

把 `_posts` 文件夹内的所有文件复制到 `source/_posts` 文件夹，并在 `_config.yml` 中修改 `new_post_name` 参数。

```
new_post_name: :year-:month-:day-:title.md
```

Octopress

把 Octopress `source/_posts` 文件夹内的所有文件转移到 Hexo 的 `source/_posts` 文件夹，并修改 `_config.yml` 中的 `new_post_name` 参数。

```
new_post_name: :year-:month-:day-:title.md
```

WordPress

首先，安装 `hexo-migrator-wordpress` 插件。

```
$ npm install hexo-migrator-wordpress --save
```

在 WordPress 仪表盘导出数据(“Tools” → “Export” → “WordPress”)（详情参考[WP支持页面](#)）。

插件安装完成后，执行下列命令来迁移所有文章。`source` 可以是 WordPress 导出的文件路径或网址。

```
$ hexo migrate wordpress <source>
```

注意

这个插件并不能完美地实现WordPress->Hexo的数据转换，尤其是在处理WordPress的分类方面存在问题（见[Front-matter中的分类与标签](#)）。因此，建议您在迁移完成后，手工审阅所有生成的markdown文件，检查其中是否有错误。对于文章数量较大的WordPress站点，这项工作可能要花很长的时间。

Joomla

首先，安装 `hexo-migrator-joomla` 插件。

```
$ npm install hexo-migrator-joomla --save
```


使用 [J2XML](#) 组件导出 Joomla 文章。插件安装完成后，执行下列命令来迁移所有文章。`source` 可以是 Joomla 导出的文件路径或网址。

```
$ hexo migrate joomla <source>
```

写作

你可以执行下列命令来创建一篇新文章或者新的页面。

```
$ hexo new [layout] <title>
```

您可以在命令中指定文章的布局（layout），默认为 `post`，可以通过修改 `_config.yml` 中的 `default_layout` 参数来指定默认布局。

布局（Layout）

Hexo 有三种默认布局：`post`、`page` 和 `draft`。在创建者三种不同类型的文件时，它们将会被保存到不同的路径；而您自定义的其他布局和 `post` 相同，都将储存在 `source/_posts` 文件夹。

布局	路径
<code>post</code>	<code>source/_posts</code>
<code>page</code>	<code>source</code>
<code>draft</code>	<code>source/_drafts</code>

不要处理我的文章

如果你不想你的文章被处理，你可以将 Front-Matter 中的 `layout:` 设为 `false`。

文件名称

Hexo 默认以标题做为文件名称，但您可编辑 `new_post_name` 参数来改变默认的文件名称，举例来说，设为 `:year-:month-:day-:title.md` 可让您更方便的通过日期来管理文章。

变量	描述
<code>:title</code>	标题（小写，空格将会被替换为短杠）
<code>:year</code>	建立的年份，比如， <code>2015</code>
<code>:month</code>	建立的月份（有前导零），比如， <code>04</code>
<code>:i_month</code>	建立的月份（无前导零），比如， <code>4</code>
<code>:day</code>	建立的日期（有前导零），比如， <code>07</code>
<code>:i_day</code>	建立的日期（无前导零），比如， <code>7</code>

草稿

刚刚提到了 Hexo 的一种特殊布局：`draft`，这种布局在建立时会被保存到 `source/_drafts` 文件夹，您可通过 `publish` 命令将草稿移动到 `source/_posts` 文件夹，该命令的使用方式与 `new` 十分类似，您也可在命令中指定 `layout` 来指定布局。

```
$ hexo publish [layout] <title>
```

草稿默认不会显示在页面中，您可在执行时加上 `--draft` 参数，或是把 `render_drafts` 参数设为 `true` 来预览草稿。

模版（Scaffold）

在新建文章时，Hexo 会根据 `scaffolds` 文件夹内相对应的文件来建立文件，例如：

```
$ hexo new photo "My Gallery"
```

在执行这行指令时，Hexo 会尝试在 `scaffolds` 文件夹中寻找 `photo.md`，并根据其内容建立文章，以下是您可以在模版中使用的变量：

变量	描述
<code>layout</code>	布局
<code>title</code>	标题
<code>date</code>	文件建立日期

Front-matter

Front-matter 是文件最上方以 `---` 分隔的区域，用于指定个别文件的变量，举例来说：

```
---title: Hello Worlddate: 2013/7/13 20:46:25---
```

以下是预先定义的参数，您可在模板中使用这些参数值并加以利用。

参数	描述	默认值
<code>layout</code>	布局	
<code>title</code>	标题	文章的文件名
<code>date</code>	建立日期	文件建立日期
<code>updated</code>	更新日期	文件更新日期
<code>comments</code>	开启文章的评论功能	<code>true</code>
<code>tags</code>	标签（不适用于分页）	
<code>categories</code>	分类（不适用于分页）	
<code>permalink</code>	覆盖文章网址	
<code>keywords</code>	仅用于 <code>meta</code> 标签和 <code>Open Graph</code> 的关键词（不推荐使用）	

分类和标签

只有文章支持分类和标签，您可以在 **Front-matter** 中设置。在其他系统中，分类和标签听起来很接近，但是在 **Hexo** 中两者有着明显的差别：分类具有顺序性和层次性，也就是说 `Foo, Bar` 不等于 `Bar, Foo`；而标签没有顺序和层次。

```
categories:
- Diary
tags:
- PS3
- Games
```

分类方法的分歧

如果您有过使用 **WordPress** 的经验，就很容易误解 **Hexo** 的分类方式。**WordPress** 支持对一篇文章设置多个分类，而且这些分类可以是同级的，也可以是父子分类。但是 **Hexo** 不支持指定多个同级分类。下面的指定方法：

```
categories:
- Diary
- Life
```

会使分类 `Life` 成为 `Diary` 的子分类，而不是并列分类。因此，有必要为您的文章选择尽可能准确的分类。

如果你需要为文章添加多个分类，可以尝试以下 `list` 中的方法。

```
categories:
- [Diary, PlayStation]
- [Diary, Games]
- [Life]
```

此时这篇文章同时包括三个分类：`PlayStation` 和 `Games` 分别都是父分类 `Diary` 的子分类，同时 `Life` 是一个没有子分类的分类。

JSON Front-matter

除了 YAML 外，你也可以使用 JSON 来编写 **Front-matter**，只要将 `---` 代换成 `;;;` 即可。

```
"title": "Hello World",  
"date": "2013/7/13 20:46:25"  
;;;
```

标签插件（Tag Plugins）

注：代码块中的\%应为%

标签插件和 **Front-matter** 中的标签不同，它们是用于在文章中快速插入特定内容的插件。

引用块

在文章中插入引言，可包含作者、来源和标题。

别号： **quote**

```
{\% blockquote [author[, source]] [link] [source_link_title] \%}  
content  
\% endblockquote \%}
```

样例

没有提供参数，则只输出普通的 **blockquote**

```
{\% blockquote \%}  
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque hendrerit lacus ut purus iaculis feugiat.  
Sed nec tempor elit, quis aliquam neque. Curabitur sed diam eget dolor fermentum semper at eu lorem.  
\% endblockquote \%}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque hendrerit lacus ut purus iaculis feugiat. Sed nec tempor elit, quis aliquam neque. Curabitur sed diam eget dolor fermentum semper at eu lorem.

引用书上的句子

```
{\% blockquote David Levithan, Wide Awake \%}  
Do not just seek happiness for yourself. Seek happiness for all. Through kindness. Through mercy.  
\% endblockquote \%}
```

Do not just seek happiness for yourself. Seek happiness for all. Through kindness. Through mercy.

David LevithanWide Awake

引用 **Twitter**

```
{\% blockquote @DevDocs https://twitter.com/devdocs/status/356095192085962752 \%}  
NEW: DevDocs now comes with syntax highlighting. http://devdocs.io  
\% endblockquote \%}
```

NEW: DevDocs now comes with syntax highlighting. <http://devdocs.io>

[@DevDocs](https://twitter.com/devdocs/status/356095192085962752)twitter.com/devdocs/status/356095192085962752

引用网络上的文章

```
{\% blockquote Seth Godin http://sethgodin.typepad.com/seths_blog/2009/07/welcome-to-island-marketing.html We  
lcome to Island Marketing \%}  
Every interaction is both precious and an opportunity to delight.  
\% endblockquote \%}
```

Every interaction is both precious and an opportunity to delight.

Seth Godin [Welcome to Island Marketing](#)

代码块

在文章中插入代码。

别名: `code`

```
{\% codeblock [title] [lang:language] [url] [link text] \%}  
code snippet  
{\% endcodeblock \%}
```

样例

普通的代码块

```
{\% codeblock \%}  
alert('Hello World!');  
{\% endcodeblock \%}
```

```
alert('Hello World!');
```

指定语言

```
{\% codeblock lang:objc \%}  
[rectangle setX: 10 y: 10 width: 20 height: 20];  
{\% endcodeblock \%}
```

```
[rectangle setX: 10 y: 10 width: 20 height: 20];
```

附加说明

```
{\% codeblock Array.map \%}  
array.map(callback[, thisArg])  
{\% endcodeblock \%}
```

```
Array.map  
array.map(callback[, thisArg])
```

附加说明和网址

```
{\% codeblock _.compact http://underscorejs.org/#compact Underscore.js \%}  
_.compact([0, 1, false, 2, '', 3]);  
=> [1, 2, 3]  
{\% endcodeblock \%}
```

```
_.compact  
_.compact([0, 1, false, 2, '', 3]);  
=> [1, 2, 3]
```

[Underscore.js](#)

反引号代码块

另一种形式的代码块，不同的是它使用三个反引号来包裹。

```
``` [language] [title] [url] [link text] code snippet ```
```

## Pull Quote

在文章中插入 Pull quote。

```
{\% pullquote [class] \%}
content
\% endpullquote \%}
```

## jsFiddle

在文章中嵌入 jsFiddle。

```
{\% jsfiddle shorttag [tabs] [skin] [width] [height] \%}
```

## Gist

在文章中嵌入 Gist。

```
{\% gist gist_id [filename] \%}
```

## iframe

在文章中插入 iframe。

```
{\% iframe url [width] [height] \%}
```

## Image

在文章中插入指定大小的图片。

```
{\% img [class names] /path/to/image [width] [height] "title text 'alt text'" \%}
```

## Link

在文章中插入链接，并自动给外部链接添加 `target="_blank"` 属性。

```
{\% link text url [external] [title] \%}
```

## Include Code



插入 `source/downloads/code` 文件夹内的代码文件。`source/downloads/code` 不是固定的，取决于你在配置文件中 `code_dir` 的配置。

```
{\% include_code [title] [lang:language] [from:line] [to:line] path/to/file \%}
```

## 样例

嵌入 **test.js** 文件全文

```
{\% include_code lang:javascript test.js \%}
```

只嵌入第 **3** 行

```
{\% include_code lang:javascript from:3 to:3 test.js \%}
```

嵌入第 **5** 行至第 **8** 行

```
{\% include_code lang:javascript from:5 to:8 test.js \%}
```

嵌入第 **5** 行至文件结束

```
{\% include_code lang:javascript from:5 test.js \%}
```

嵌入第 **1** 行至第 **8** 行

```
{\% include_code lang:javascript to:8 test.js \%}
```

## Youtube

在文章中插入 Youtube 视频。

```
{\% youtube video_id \%}
```

## Vimeo

在文章中插入 Vimeo 视频。

```
{\% vimeo video_id \%}
```

## 引用文章

引用其他文章的链接。

```
{\% post_path slug \%}
{\% post_link slug [title] [escape] \%}
```

在使用此标签时可以忽略文章文件所在的路径或者文章的永久链接信息、如语言、日期。

例如，在文章中使用 `{\% post_link how-to-bake-a-cake %}` 时，只需有一个名为 `how-to-bake-a-cake.md` 的文章文件即可。即使这个文件位于站点文件夹的 `source/posts/2015-02-my-family-holiday` 目录下、或者文章的永久链接是 `2018/en/how-to-bake-a-cake`，都没有影响。

默认链接文字是文章的标题，你也可以自定义要显示的文本。此时不应该使用 Markdown 语法 `[]()`。

链接使用文章的标题

```
{\% post_link hexo-3-8-released \%}
```

## Hexo 3.8.0 Released

链接使用自定义文字

```
{\% post_link hexo-3-8-released '通往文章的链接' \%}
```

通往文章的链接

## 引用资源

引用文章的资源。

```
{\% asset_path slug \%}
{\% asset_img slug [title] \%}
{\% asset_link slug [title] [escape] \%}
```

## Raw

如果您想在文章中插入 **Swig** 标签，可以尝试使用 **Raw** 标签，以免发生解析异常。

```
{\% raw \%}
content
{\% endraw \%}
```

## 文章摘要和截断

在文章中使用 `<!-- more -->`，那么 `<!-- more -->` 之前的文字将会被视为摘要。首页中将只出现这部分文字，同时这部分文字也会出现在正文之中。

例如：

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
<!-- more -->
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

首页中将只会出现

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
```

正文中则会出现

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepte ur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

注意，摘要可能会被 **Front Matter** 中的 `excerpt` 覆盖。

## 资源文件夹

资源（**Asset**）代表 `source` 文件夹中除了文章以外的所有文件，例如图片、CSS、JS 文件等。比方说，如果你的Hexo项目中只有少量图片，那最简单的方法就是将它们放在 `source/images` 文件夹中。然后通过类似于 `` 的方法访问它们。

## 文章资源文件夹

对于那些想要更有规律地提供图片和其他资源以及想要将他们的资源分布在各个文章上的人来说，Hexo也提供了更组织化的方式来管理资源。这个稍微有些复杂但是管理资源非常方便的功能可以通过将 `config.yml` 文件中的 `post_asset_folder` 选项设为 `true` 来打开。

```
_config.yml
post_asset_folder: true
```

当资源文件管理功能打开后，Hexo将会在你每一次通过 `hexo new [layout]` 命令创建新文章时自动创建一个文件夹。这个资源文件夹将会有与这个文章文件一样的名字。将所有与你的文章有关的资源放在这个关联文件夹中之后，你可以通过相对路径来引用它们，这样你就得到了一个更简单而且方便得多的工作流。

## 相对路径引用的标签插件

通过常规的 `markdown` 语法和相对路径来引用图片和其它资源可能会导致它们在存档页或者主页上显示不正确。在Hexo 2时代，社区创建了很多插件来解决这个问题。但是，随着Hexo 3 的发布，许多新的标签插件被加入到了核心代码中。这使得你可以更简单地在文章中引用你的资源。

```
{% asset_path slug %}
{% asset_img slug [title] %}
{% asset_link slug [title] %}
```

比如说：当你打开文章资源文件夹功能后，你把一个 `example.jpg` 图片放在了你的资源文件夹中，如果通过使用相对路径的常规 `markdown` 语法 ``，它将不会出现在首页上。（但是它会在文章中按你期待的方式工作）

正确的引用图片方式是使用下列的标签插件而不是 `markdown`：

```
{% asset_img example.jpg This is an example image %}
```

通过这种方式，图片将会同时出现在文章和主页以及归档页中。

## 数据文件

有时您可能需要在主题中使用某些资料，而这些资料并不在文章内，并且是需要重复使用的，那么您可以考虑使用 **Hexo 3.0** 新增的「数据文件」功能。此功能会载入 `source/_data` 内的 **YAML** 或 **JSON** 文件，如此一来您便能在网站中复用这些文件了。

举例来说，在 `source/_data` 文件夹中新建 `menu.yml` 文件：

```
Home: /
Gallery: /gallery/
Archives: /archives/
```

您就能在模板中使用这些资料：

```
<% for (var link in site.data.menu) { %>
 <a href="<%= site.data.menu[link] %%"> <%= link %>
<% } %>
```

渲染结果如下：

```
 Home
 Gallery
 Archives
```

## 服务器

### hexo-server

Hexo 3.0 把服务器独立成了个别模块，您必须先安装 `hexo-server` 才能使用。

```
$ npm install hexo-server --save
```

安装完成后，输入以下命令以启动服务器，您的网站会在 `http://localhost:4000` 下启动。在服务器启动期间，Hexo 会监视文件变动并自动更新，您无须重启服务器。

```
$ hexo server
```

如果您想要更改端口，或是在执行时遇到了 `EADDRINUSE` 错误，可以在执行时使用 `-p` 选项指定其他端口，如下：

```
$ hexo server -p 5000
```

### 静态模式

在静态模式下，服务器只处理 `public` 文件夹内的文件，而不会处理文件变动，在执行时，您应该先自行执行 `hexo generate`，此模式通常用于生产环境（`production mode`）下。

```
$ hexo server -s
```

### 自定义 IP

服务器默认运行在 `0.0.0.0`，您可以覆盖默认的 IP 设置，如下：

```
$ hexo server -i 192.168.1.1
```

指定这个参数后，您就只能通过该IP才能访问站点。例如，对于一台使用无线网络的笔记本电脑，除了指向本机的 `127.0.0.1` 外，通常还有一个 `192.168.*.*` 的局域网IP，如果像上面那样使用 `-i` 参数，就不能用 `127.0.0.1` 来访问站点了。对于有公网IP的主机，如果您指定一个局域网IP作为 `-i` 参数的值，那么就无法通过公网来访问站点。

## Pow

**Pow** 是一个 Mac 系统上的零配置 Rack 服务器，它也可以作为一个简单易用的静态文件服务器来使用。

### 安装

```
$ curl get.pow.cx | sh
```

### 设置

在 `~/.pow` 文件夹建立链接（`symlink`）。

```
$ cd ~/.pow$ ln -s /path/to/myapp
```

您的网站将会在 `http://myapp.dev` 下运行，网址根据链接名称而定。



## 生成文件

使用 **Hexo** 生成静态文件快速而且简单。

```
$ hexo generate
```

## 监视文件变动

**Hexo** 能够监视文件变动并立即重新生成静态文件，在生成时会比对文件的 **SHA1 checksum**，只有变动的文件才会写入。

```
$ hexo generate --watch
```

## 完成后部署

您可执行下列的其中一个命令，让 **Hexo** 在生成完毕后自动部署网站，两个命令的作用是相同的。

```
$ hexo generate --deploy
$ hexo deploy --generate
```

简写

上面两个命令可以简写为

```
$ hexo g -d
```

```
$ hexo d -g
```



## 将 Hexo 部署到 GitHub Pages

在本教程中，我们将会使用 [Travis CI](#) 将 Hexo 博客部署到 GitHub Pages 上。Travis CI 对于开源 repository 是免费的，但是这意味着你的站点文件将会是公开的。如果你希望你的站点文件不被公开，请直接前往本文 [Private repository](#) 部分。

1. 新建一个 repository。如果你希望你的站点能通过 `<你的 GitHub 用户名>.github.io` 域名访问，你的 repository 应该直接命名为 `<你的 GitHub 用户名>.github.io`。
2. 将你的 Hexo 站点文件夹推送到 repository 中。默认情况下不应该 `public` 目录将不会被推送到 repository 中，你应该检查 `.gitignore` 文件中是否包含 `public` 一行，如果没有请加上。
3. 将 [Travis CI](#) 添加到你的 GitHub 账户中。
4. 前往 GitHub 的 [Applications settings](#)，配置 Travis CI 权限，使其能够访问你的 repository。
5. 你应该会被重定向到 Travis CI 的页面。如果没有，请 [手动前往](#)。
6. 在浏览器新建一个标签页，前往 GitHub [新建 Personal Access Token](#)，只勾选 `repo` 的权限并生成一个新的 Token。Token 生成后请复制并保存好。
7. 回到 Travis CI，前往你的 repository 的设置页面，在 **Environment Variables** 下新建一个环境变量，**Name** 为 `GH_TOKEN`，**Value** 为刚才你在 GitHub 生成的 Token。确保 **DISPLAY VALUE IN BUILD LOG** 保持不被勾选 避免你的 Token 泄漏。点击 **Add** 保存。
8. 在你的 Hexo 站点文件夹中新建一个 `.travis.yml` 文件：

```
sudo: false
language: node_js
node_js:
 - 10 # use nodejs v10 LTS
cache: npm
branches:
 only:
 - master # build master branch only
script:
 - hexo generate # generate static files
deploy:
 provider: pages
 skip-cleanup: true
 github-token: $GH_TOKEN
 keep-history: true
 on:
 branch: master
 local-dir: public
```

1. 将 `.travis.yml` 推送到 repository 中。Travis CI 应该会自动开始运行，并将生成的文件推送到同一 repository 下的 `gh-pages` 分支下
2. 在 GitHub 中前往你的 repository 的设置页面，修改 [GitHub Pages](#) 的部署分支为 `gh-pages`。
3. 前往 <https://<你的 GitHub 用户名>.github.io> 查看你的站点是否可以访问。这可能需要一些时间。

## Project page

如果你更希望你的站点部署在 `<你的 GitHub 用户名>.github.io` 的子目录中，你的 repository 需要直接命名为子目录的名字，这样你的站点可以通过 <https://<你的 GitHub 用户名>.github.io/> 访问。你需要检查你的 Hexo 配置文件，将 `url` 修改为 `https://<你的 GitHub 用户名>.github.io/`、将 `root` 的值修改为 `//`

## Private repository

The following instruction is adapted from [one-command deployment](#) page.

1. Install [hexo-deployer-git](#).

2. Add the following configurations to **\_config.yml**, (remove existing lines if any)

```
deploy:
 type: git
 repo: https://github.com/<username>/<project>
 # example, https://github.com/hexojs/hexojs.github.io
 branch: gh-pages
```

3. Run `hexo clean && hexo deploy` .
4. Check the webpage at `username.github.io`.

## 有用的参考链接

- [GitHub Pages 使用文档](#)
- [Travis CI 使用文档](#)
- [Awesome Hexo](#)
- 在百度上搜索 “Hexo GitHub”

## 将 Hexo 部署到 GitLab Pages

在本教程中，我们将会使用 GitLab CI 将 Hexo 博客部署到 GitLab Pages 上。

1. 新建一个 repository。如果你希望你的站点能通过 `<你的 GitLab 用户名>.gitlab.io` 域名访问，你的 repository 应该直接命名为 `<你的 GitLab 用户名>.gitlab.io`。
2. 将你的 Hexo 站点文件夹推送到 repository 中。默认情况下不应该 `public` 目录将不会被推送到 repository 中，你应该检查 `.gitignore` 文件中是否包含 `public` 一行，如果没有请加上。
3. 在你的站点文件夹中新建 `.gitlab-ci.yml` 文件：

```
image: node:10-alpine # use nodejs v10 LTS
cache:
 paths:
 - node_modules/

before_script:
 - npm install hexo-cli -g
 - npm install

pages:
 script:
 - hexo generate
 artifacts:
 paths:
 - public
 only:
 - master
```

1. GitLab CI 应该会自动开始运行，构建成功以后你应该可以在 `https://<你的 GitLab 用户名>.gitlab.io` 查看你的网站。
2. (可选) 如果你需要查看生成的文件，可以在 [job artifact](#) 中找到。

在 GitLab.com 上，GitLab CI 是默认启用的。如果你使用的是自托管的 GitLab，你可能需要在 `Settings -> CI / CD -> Shared Runners` 启用 GitLab CI。

## Project page

如果你更希望你的站点部署在 `<你的 GitLab 用户名>.gitlab.io` 的子目录中，你的 repository 需要直接命名为子目录的名字，这样你的站点可以通过 `https://<你的 GitLab 用户名>.gitlab.io/` 访问。你需要检查你的 Hexo 配置文件，将 `url` 的值修改为 `https://<你的 GitLab 用户名>.gitlab.io/`、将 `root` 的值修改为 `//`

## Useful links

- [GitLab Pages 相关文档](#)
- [GitLab CI 相关文档](#)
- 在百度上搜索 “Hexo GitLab”

## 部署

Hexo 提供了快速方便的一键部署功能，让您只需一条命令就能将网站部署到服务器上。

```
$ hexo deploy
```

在开始之前，您必须先在 `_config.yml` 中修改参数，一个正确的部署配置中至少要有 `type` 参数，例如：

```
deploy:
 type: git
```

您可同时使用多个 `deployer`，Hexo 会依照顺序执行每个 `deployer`。

```
deploy:
- type: git
 repo:
- type: heroku
 repo:
```

### 缩进

YAML 依靠缩进来确定元素间的从属关系。因此，请确保每个 `deployer` 的缩进长度相同，并且使用空格缩进。

## Git

安装 [hexo-deployer-git](#)。

```
$ npm install hexo-deployer-git --save
```

修改配置。

```
deploy:
 type: git
 repo: <repository url> #https://bitbucket.org/JohnSmith/johnsmith.bitbucket.io
 branch: [branch] #published
 message: [message]
```

参数	描述
<code>repo</code>	库（Repository）地址
<code>branch</code>	分支名称。如果不指定，则默认值为 <code>master</code>
<code>message</code>	自定提交信息

生成站点文件并推送至远程库。执行 `hexo clean && hexo deploy` 命令（如果您并未全局安装 Hexo Cli，则需要执行 `./node_modules/.bin/hexo clean && ./node_modules/.bin/hexo deploy`）

登入 Github/BitBucket/Gitlab，请在库设置（Repository Settings）中将默认分支设置为 `_config.yml` 配置中的分支名称。稍等片刻，您的站点就会显示在您的 Github Pages 中。

## 这一切是如何发生的？

当执行 `hexo deploy` 时，Hexo 会将 `public` 目录中的文件和目录推送到 `_config.yml` 中指定的远端仓库和分支中，并且完全覆盖该分支下的已有内容。

For 使用 Git 管理站点目录的用户

由于 Hexo 的部署默认使用分支 `master`，所以如果你同时正在使用 Git 管理你的站点目录，你应当注意你的部署分支应当不同于写作分支。一个好的实践是将站点目录和 Pages 分别存放在两个不同的 Git 仓库中，可以有效避免相互覆盖。Hexo 在部署你的站点生成的文件时并不会更新你的站点目录。因此你应该手动提交并推送你的写作分支。

此外，如果您的 Github Pages 需要使用 CNAME 文件自定义域名，请将 CNAME 文件置于 `source` 目录下，只有这样 `hexo deploy` 才能将 CNAME 文件一并推送至部署分支。

## Heroku

安装 [hexo-deployer-heroku](#)。

```
$ npm install hexo-deployer-heroku --save
```

修改配置。

```
deploy:
 type: openshift
 repo: <repository url>
 message: [message]
```

参数	描述
<code>repo</code>	Heroku 库 (Repository) 地址
<code>message</code>	自定提交信息

## Netlify

[Netlify](#) 是一个提供网络托管的综合平台。它集持续集成 (CI) CDN 自定义域名 HTTPS 持续部署 (CD) 等诸多功能于一身。您可以通过以下两种方式将Hexo站点部署到Netlify。

首先，也是最通用的方式，就是使用Netlify提供的网页端用户界面。前往[新建一个网站页面](#)，选择需要关联的 Github/BitBucket/Gitlab 库，然后遵循网站提示。

另一种方式是使用Netlify提供的命令行客户端工具 [Node based CLI](#) 管理和部署您的站点。

此外，您还可以在项目的README中增加一个 [部署至Netlify按钮](#)，这样其他用户在fork或clone了您的项目之后可以方便快捷地一键部署。

## Rsync

安装 [hexo-deployer-rsync](#)。

```
$ npm install hexo-deployer-rsync --save
```

修改配置。

```
deploy:
 type: ftpsync
 host: <host>
 user: <user>
 pass: <password>
 remote: [remote]
 port: [port]
 ignore: [ignore]
```

```
connections: [connections]
verbose: [true|false]
```

参数	描述	默认值
host	远程主机的地址	
user	使用者名称	
root	远程主机的根目录	
port	端口	22
delete	删除远程主机上的旧文件	true
verbose	显示调试信息	true
ignore_errors	忽略错误	false

rsync部署模块的工作方式

需要注意的是，要求您提供的实际上是一个能通过SSH登陆远程主机的Linux用户。Hexo会自动处理关于rsync使用的一切操作。因此，您需要在远程主机上为您的Hexo站点建立一个用户，并允许其通过SSH登陆。不过，这里的 `port`，的确是指rsync监听的端口，请确保防火墙打开了该端口。

## OpenShift

安装 [hexo-deployer-openshift](#)。

```
$ npm install hexo-deployer-openshift --save
```

修改配置。

```
deploy:
 type: openshift
 repo: <repository url>
 message: [message]
```

参数	描述
repo	OpenShift 库（Repository）地址
message	自定提交信息

## FTPSync

安装 [hexo-deployer-ftpsync](#)。

```
$ npm install hexo-deployer-ftpsync --save
```

修改配置。

```
deploy:
 type: ftpsync
 host: <host>
 user: <user>
 pass: <password>
 remote: [remote]
 port: [port]
 ignore: [ignore]
```

```
connections: [connections]
verbose: [true|false]
```

参数	描述	默认值
host	远程主机的地址	
user	使用者名称	
pass	密码	
remote	远程主机的根目录	/
port	端口	21
ignore	忽略的文件或目录	
connections	使用的连接数	1
verbose	显示调试信息	false

#### FTP部署可能出现的问题

您可能需要预先通过其他方式将所有文件上传到远程主机中。否则初次使用ftpsync插件就可能出现报错。另外，由于FTP协议的特征，它每传送一个文件就需要一次握手，相对速度较慢。

## SFTP

安装 [hexo-deployer-sftp](#)。

```
$ npm install hexo-deployer-sftp --save
```

修改配置。

```
deploy:
 type: sftp
 host: <host>
 user: <user>
 pass: <password>
 remotePath: [remote path]
 port: [port]
 privateKey: [path/to/privateKey]
 passphrase: [passphrase]
 agent: [path/to/agent/socket]
```

参数	描述	默认值
host	远程主机的地址	
user	使用者名称	
pass	密码	
remotePath	远程主机的根目录	/
port	端口	22
privateKey	ssh私钥的目录地址	
passphrase	(可省略) ssh私钥的密码短语	
agent	ssh套接字的目录地址	SSH_AUTH_SOCK

## 其他方法

Hexo 生成的所有文件都放在 `public` 文件夹中，您可以将它们复制到您喜欢的地方。



## 永久链接（Permalinks）

您可以在 `_config.yml` 配置中调整网站的永久链接或者在每篇文章的 **Front-matter** 中指定。

### 变量

除了下列变量外，您还可使用 **Front-matter** 中的所有属性。

变量	描述
<code>:year</code>	文章的发表年份（4 位数）
<code>:month</code>	文章的发表月份（2 位数）
<code>:i_month</code>	文章的发表月份（去掉开头的零）
<code>:day</code>	文章的发表日期（2 位数）
<code>:i_day</code>	文章的发表日期（去掉开头的零）
<code>:title</code>	文件名称
<code>:post_title</code>	文章标题
<code>:id</code>	文章 ID
<code>:category</code>	分类。如果文章没有分类，则是 <code>default_category</code> 配置信息。

您可在 `permlink_defaults` 参数下调整永久链接中各变量的默认值：

```
permlink_defaults:
 lang: en
```

### 示例

假设 `source/_posts` 文件夹中有个 `hello-world.md`，包含以下内容：

```
title: Hello World
date: 2013-07-14 17:01:34
categories:
- foo
- bar
```

参数	结果
<code>:year/:month/:day/:title/</code>	2013/07/14/hello-world
<code>:year-:month-:day-:title.html</code>	2013-07-14-hello-world.html
<code>:category/:title</code>	foo/bar/hello-world

### 多语种支持

若要建立一个多语种的网站，您可修改 `new_post_name` 和 `permlink` 参数，如下：

```
new_post_name: :lang/:title.md
permlink: :lang/:title/
```

当您建立新文章时，文章会被储存在：

```
$ hexo new "Hello World" --lang tw
=> source/_posts/tw/Hello-World.md
```

而网址会是:

```
http://localhost:4000/tw/hello-world/
```

## 主题

创建 Hexo 主题非常容易，您只要在 `themes` 文件夹内，新增一个任意名称的文件夹，并修改 `_config.yml` 内的 `theme` 设定，即可切换主题。一个主题可能会有以下的结构：

```
.
├─ _config.yml
├─ languages
├─ layout
├─ scripts
└─ source
```

### `_config.yml`

主题的配置文件。修改时会自动更新，无需重启服务器。

### languages

语言文件夹。请参见 [国际化 \(i18n\)](#)。

### layout

布局文件夹。用于存放主题的模板文件，决定了网站内容的呈现方式，Hexo 内建 [Swig](#) 模板引擎，您可以另外安装插件来获得 [EJS](#)、[Haml](#) 或 [Jade](#) 支持，Hexo 根据模板文件的扩展名来决定所使用的模板引擎，例如：

```
layout.ejs - 使用 EJS
layout.swig - 使用 Swig
```

您可参考 [模板](#) 以获得更多信息。

### scripts

脚本文件夹。在启动时，Hexo 会载入此文件夹内的 JavaScript 文件，请参见 [插件](#) 以获得更多信息。

### source

资源文件夹，除了模板以外的 Asset，例如 CSS、JavaScript 文件等，都应该放在这个文件夹中。文件或文件夹开头名称为 `_`（下划线）或隐藏的文件会被忽略。

如果文件可以被渲染的话，会经过解析然后储存到 `public` 文件夹，否则会直接拷贝到 `public` 文件夹。

## 发布

当您完成主题后，可以考虑将它发布到 [主题列表](#)，让更多人能够使用您的主题。在发布前建议先进行 [主题单元测试](#)，确保每一项功能都能正常使用。发布主题的步骤和 [更新文档](#) 非常类似。

1. Fork [hexojs/site](#)
2. 把库（repository）复制到电脑上，并安装所依赖的插件。

```
$ git clone https://github.com/<username>/site.git
$ cd site
$ npm install
```

3. 编辑 `source/_data/themes.yml`，在文件中新增您的主题，例如：

```
- name: landscape
 description: A brand new default theme for Hexo.
 link: https://github.com/hexojs/hexo-theme-landscape
 preview: http://hexo.io/hexo-theme-landscape
 tags:
 - official
 - responsive
 - widget
 - two_column
 - one_column
```

4. 在 `source/themes/screenshots` 新增同名的截图档案，图片必须为 800x500 的 PNG 文件。
5. 推送（push）分支。
6. 建立一个新的合并申请（pull request）并描述改动。

## 模版

模板决定了网站内容的呈现方式，每个主题至少都应包含一个 `index` 模板，以下是各页面相对应的模板名称：

模板	用途	回退
<code>index</code>	首页	
<code>post</code>	文章	<code>index</code>
<code>page</code>	分页	<code>index</code>
<code>archive</code>	归档	<code>index</code>
<code>category</code>	分类归档	<code>archive</code>
<code>tag</code>	标签归档	<code>archive</code>

## 布局（Layout）

如果页面结构类似，例如两个模板都有页首（**Header**）和页脚（**Footer**），您可考虑通过「布局」让两个模板共享相同的结构。一个布局文件必须要能显示 `body` 变量的内容，如此一来模板的内容才会被显示，举例来说：

```
index.ejs
index
```

```
layout.ejs
<!DOCTYPE html>
<html>
 <body><%= body %></body>
</html>
```

生成：

```
<!DOCTYPE html>
<html>
 <body>index</body>
</html>
```

每个模板都默认使用 `layout` 布局，您可在 `front-matter` 指定其他布局，或是设为 `false` 来关闭布局功能，您甚至可在布局中再使用其他布局来建立嵌套布局。

## 局部模版（Partial）

局部模板让您在不同模板之间共享相同的组件，例如页首（**Header**）、页脚（**Footer**）或侧边栏（**Sidebar**）等，可利用局部模板功能分割为个别文件，让维护更加便利。举例来说：

```
partial/header.ejs
<h1 id="logo"><%= config.title %></h1>
```

```
index.ejs
<%= partial('partial/header') %>
<div id="content">Home page</div>
```

生成：

```
<h1 id="logo">My Site</h1>
<div id="content">Home page</div>
```

## 局部变量

您可以在局部模板中指定局部变量并使用。

```
partial/header.ejs
<h1 id="logo"><%= title %></h1>
```

```
index.ejs
<%= partial('partial/header', {title: 'Hello World'}) %>
<div id="content">Home page</div>
```

生成:

```
<h1 id="logo">Hello World</h1>
<div id="content">Home page</div>
```

## 优化

如果您的主题太过于复杂，或是需要生成的文件量太过于庞大，可能会大幅降低性能，除了简化主题外，您可以考虑 **Hexo 2.7** 新增的局部缓存（**Fragment Caching**）功能。

本功能借鉴于 [Ruby on Rails](#)，它储存局部内容，下次便能直接使用缓存内容，可以减少文件夹查询并使生成速度更快。

它可用于页首、页脚、侧边栏等文件不常变动的位置，举例来说：

```
<%= fragment_cache('header', function(){
 return '<header></header>';
});
```

如果您使用局部模板的话，可以更简单：

```
<%= partial('header', {}, {cache: true});
```

`fragment_cache()` 将会缓存第一次的渲染结果，并在之后直接输出缓存的结果。因此只有在不同页面的渲染结果都相同时才应使用局部缓存。比如，在配置中启用了 `relative_link` 后不应该使用局部缓存，因为相对链接在每个页面可能不同。

## 变量

### 全局变量

变量	描述	类型
<code>site</code>	<a href="#">网站变量</a>	<code>object</code> ; 见 <a href="#">网站变量</a>
<code>page</code>	针对该页面的内容以及 <code>front-matter</code> 中自定义的变量。	<code>object</code> ; 见 <a href="#">页面变量</a>
<code>config</code>	网站配置	<code>object</code> (站点的配置文件)
<code>theme</code>	主题配置。继承自网站配置。	<code>object</code> (主题配置文件)
<code>_</code> (单下划线)	<a href="#">Lodash</a> 函数库	<a href="#">Lodash</a> 文档
<code>path</code>	当前页面的路径 (不含根路径)	<code>string</code>
<code>url</code>	当前页面的完整网址	<code>string</code>
<code>env</code>	环境变量	???

### 网站变量

变量	描述	类型
<code>site.posts</code>	所有文章	<code>array of post objects</code>
<code>site.pages</code>	所有分页	<code>array of page objects</code>
<code>site.categories</code>	所有分类	<code>object</code> , 包含了站点全部的分类
<code>site.tags</code>	所有标签	<code>array</code> , 包含了站点全部的标签

### 页面变量

页面 ( `page` )

变量	描述	类型
<code>page.title</code>	页面标题	<code>string</code>
<code>page.date</code>	页面建立日期	<a href="#">Moment.js</a> 对象
<code>page.updated</code>	页面更新日期	<a href="#">Moment.js</a> 对象
<code>page.comments</code>	留言是否开启	<code>boolean</code>
<code>page.layout</code>	布局名称	<code>string</code>
<code>page.content</code>	页面的完整内容	<code>string</code>
<code>page.excerpt</code>	页面摘要	<code>string</code>
<code>page.more</code>	除了页面摘要的其余内容	<code>string</code>
<code>page.source</code>	页面原始路径	<code>string</code>
<code>page.full_source</code>	页面的完整原始路径	<code>string</code>
<code>page.path</code>	页面网址 (不含根路径)。我们通常在主题中使用 <code>url_for(page.path)</code> 。	<code>string</code>
<code>page.permalink</code>	页面的完整网址	<code>string</code>
<code>page.prev</code>	上一个页面。如果此为第一个页面则为 <code>null</code> 。	<code>string</code> or <code>null</code>
<code>page.next</code>	下一个页面。如果此为最后一个页面则为 <code>null</code> 。	<code>string</code> or <code>null</code>

<code>page.raw</code>	文章的原始内容	???
<code>page.photos</code>	文章的照片（用于相簿）	array
<code>page.link</code>	文章的外部链接（用于链接文章）	string

文章 ( `post` ): 与 `page` 布局相同，但新增以下变量。

变量	描述	类型
<code>page.published</code>	如果该文章已发布则为 <code>true</code>	boolean
<code>page.categories</code>	该文章的所有分类	array of ???
<code>page.tags</code>	该文章的所有标签	array of ???

首页 ( `index` )

变量	描述	类型
<code>page.per_page</code>	每页显示的文章数量	number
<code>page.total</code>	总文章数	number
<code>page.current</code>	目前页数	number
<code>page.current_url</code>	目前分页的网址	string
<code>page.posts</code>	本页文章 ( <a href="#">Data Model</a> )	object
<code>page.prev</code>	上一页的页数。如果此页是第一页的话则为 <code>0</code> 。	number
<code>page.prev_link</code>	上一页的网址。如果此页是第一页的话则为 <code>''</code> 。	string
<code>page.next</code>	下一页的页数。如果此页是最后一页的话则为 <code>0</code> 。	number
<code>page.next_link</code>	下一页的网址。如果此页是最后一页的话则为 <code>''</code> 。	string
<code>page.path</code>	当前页面的路径（不含根目录）。我们通常在主题中使用 <code>url_for(page.path)</code> 。	string

归档 ( `archive` ): 与 `index` 布局相同，但新增以下变量。

变量	描述	类型
<code>page.archive</code>	等于 <code>true</code>	boolean
<code>page.year</code>	年份归档 (4位)	number
<code>page.month</code>	月份归档 (没有前导零的2位数)	number

分类 ( `category` ): 与 `index` 布局相同，但新增以下变量。

变量	描述	类型
<code>page.category</code>	分类名称	string

标签 ( `tag` ): 与 `index` 布局相同，但新增以下变量。

变量	描述	类型
<code>page.tag</code>	标签名称	string



## 辅助函数（Helpers）

辅助函数帮助您在模版中快速插入内容。辅助函数不能在源文件中使用。

### 网址

#### url\_for

在路径前加上根路径，从 Hexo 2.7 开始您应该使用此函数而不是 `config.root + path`。

```
<%- url_for(path, [option]) %>
```

参数	描述	默认值
<code>relative</code>	是否输出相对链接	<code>config.relative_link</code> 的值

示例：

```
_config.yml
root: /blog/
```

```
<%- url_for('/a/path') %>
// /blog/a/path
```

是否输出相对链接，默认遵循配置文件中 `relative_link` 的值 例如，`post/page` 的相对路径值可能是 `/foo/bar/index.html`

```
_config.yml
relative_link: true
```

```
<%- url_for('/css/style.css') %>
// ../../css/style.css
/* 覆盖配置
 * 即使配置文件中启用了 relative_link，你也可以使用 relative 参数禁用相对链接输出，反之亦然
 */
<%- url_for('/css/style.css', {relative: false}) %>
// /css/style.css
```

#### relative\_url

取得与 `from` 相对的 `to` 路径。

```
<%- relative_url(from, to) %>
```

示例：

```
<%- relative_url('foo/bar/', 'css/style.css') %>
// ../../css/style.css
```

#### gravatar

根据邮箱地址返回 Gravatar 头像 URL。

如果你不指定 `options` 参数，将会应用默认参数。否则，你可以将其设置为一个数字，这个数字将会作为 Gravatar 的大小参数。最后，如果你设置它一个对象，它将会被转换为 Gravatar 的一个查询字符串参数。

```
<%- gravatar(email, [options]) %>
```

参数	描述	默认值
s	图片大小	80
d	默认头像	
f	强制使用默认图象	
r	头像等级限制	

访问 [Gravatar](#) 了解更多。

示例：

```
<%- gravatar('a@abc.com') %>
// https://www.gravatar.com/avatar/b9b00e66c6b8a70f88c73cb6bdb06787

<%- gravatar('a@abc.com', 40) %>
// https://www.gravatar.com/avatar/b9b00e66c6b8a70f88c73cb6bdb06787?s=40

<%- gravatar('a@abc.com' {s: 40, d: 'https://via.placeholder.com/150'}) %>
// https://www.gravatar.com/avatar/b9b00e66c6b8a70f88c73cb6bdb06787?s=40&d=https%3A%2F%2Fvia.placeholder.com%2F150
```

## full\_url\_for

在路径前加上根路径和域名。输出会被自动转码。

```
<%- full_url_for(path) %>
```

示例：

```
_config.yml
url: https://example.com/blog # example
```

```
<%- full_url_for('/a/path') %>
// https://example.com/blog/a/path
```

## HTML 标签

### CSS

载入 CSS 文件。`path` 可以是数组或字符串，如果 `path` 开头不是 `/` 或任何协议，则会自动加上根路径；如果后面没有加上 `.css` 扩展名的话，也会自动加上。

```
<%- css(path, ...) %>
```

示例：

```
<%- css('style.css') %>
// <link rel="stylesheet" href="/style.css">

<%- css(['style.css', 'screen.css']) %>
// <link rel="stylesheet" href="/style.css">
// <link rel="stylesheet" href="/screen.css">
```

## js

载入 **JavaScript** 文件。`path` 可以是数组或字符串，如果 `path` 开头不是 `/` 或任何协议，则会自动加上根路径；如果后面没有加上 `.js` 扩展名的话，也会自动加上。

```
<%- js(path, ...) %>
```

示例：

```
<%- js('script.js') %>
// <script src="/script.js"></script>

<%- js(['script.js', 'gallery.js']) %>
// <script src="/script.js"></script>
// <script src="/gallery.js"></script>
```

## link\_to

插入链接。

```
<%- link_to(path, [text], [options]) %>
```

参数	描述	默认值
<code>external</code>	在新视窗打开链接	<b>false</b>
<code>class</code>	Class 名称	
<code>id</code>	ID	

示例：

```
<%- link_to('http://www.google.com') %>
// http://www.google.com

<%- link_to('http://www.google.com', 'Google') %>
// Google

<%- link_to('http://www.google.com', 'Google', {external: true}) %>
// Google
```

## mail\_to

插入电子邮箱链接。

```
<%- mail_to(path, [text], [options]) %>
```

参数	描述
----	----

class	Class 名称
id	ID
subject	邮件主题
cc	抄送 (CC)
bcc	密送 (BCC)
body	邮件内容

示例：

```
<%- mail_to('a@abc.com') %>
// a@abc.com

<%- mail_to('a@abc.com', 'Email') %>
// Email
```

## image\_tag

插入图片。

```
<%- image_tag(path, [options]) %>
```

参数	描述
alt	图片的替代文字
class	Class 名称
id	ID
width	图片宽度
height	图片高度

## favicon\_tag

插入 favicon。

```
<%- favicon_tag(path) %>
```

## feed\_tag

插入 feed 链接。

```
<%- feed_tag(path, [options]) %>
```

参数	描述	默认值
title	Feed 标题	
type	Feed 类型	atom

## 条件函数

### is\_current

检查 `path` 是否符合目前页面的网址。开启 `strict` 选项启用严格比对。

```
<%- is_current(path, [strict]) %>
```

## is\_home

检查当前页面是否为首页。

```
<%- is_home() %>
```

## is\_post

检查当前页面是否为文章。

```
<%- is_post() %>
```

## is\_page

检查当前页面是否为独立页面。

```
<%- is_page() %>
```

## is\_archive

检查当前页面是否为存档页面。

```
<%- is_archive() %>
```

## is\_year

检查当前页面是否为年度归档页面。

```
<%- is_year() %>
```

## is\_month

检查当前页面是否为月度归档页面。

```
<%- is_month() %>
```

## is\_category

检查当前页面是否为分类归档页面。如果给定一个字符串作为参数，将会检查目前是否为指定分类。

```
<%- is_category() %>
<%- is_category('hobby') %>
```

## is\_tag

检查当前页面是否为标签归档页面。如果给定一个字符串作为参数，将会检查目前是否为指定标签。

```
<%- is_tag() %>
<%- is_tag('hobby') %>
```

## 字符串处理

### trim

清除字符串开头和结尾的空格。

```
<%- trim(string) %>
```

### strip\_html

清除字符串中的 HTML 标签。

```
<%- strip_html(string) %>
```

示例：

```
<%- strip_html('It\'s not important anymore!') %>
// It's not important anymore!
```

### titlecase

把字符串转换为正确的 Title case。

```
<%- titlecase(string) %>
```

示例：

```
<%- titlecase('this is an apple') %>
This is an Apple
```

### markdown

使用 Markdown 解析字符串。

```
<%- markdown(str) %>
```

示例：

```
<%- markdown('make me **strong**') %>
// make me strong
```

### render

解析字符串。

```
<%- render(str, engine, [options]) %>
```

## word\_wrap

使每行的字符串长度不超过 `length`。`length` 预设值为 80。

```
<%- word_wrap(str, [length]) %>
```

示例：

```
<%- word_wrap('Once upon a time', 8) %>
// Once upon\n a time
```

## truncate

移除超过 `length` 长度的字符串。`length` 的默认值是 30。

```
<%- truncate(text, length) %>
```

示例：

```
<%- truncate('Once upon a time in a world far far away', {length: 17}) %>
// Once upon a ti...

<%- truncate('Once upon a time in a world far far away', {length: 17, separator: ' '}) %>
// Once upon a...

<%- truncate('And they found that many people were sleeping better.', {length: 25, omission: '... (continued)'}) %>
// And they f... (continued)
```

## 模板

### partial

载入其他模板文件，您可在 `locals` 设定区域变量。

```
<%- partial(layout, [locals], [options]) %>
```

参数	描述	默认值
<code>cache</code>	缓存（使用 <code>Fragment cache</code> ）	<code>false</code>
<code>only</code>	限制局部变量。在模板中只能使用 <code>locals</code> 中设定的变量。	<code>false</code>

### fragment\_cache

局部缓存。它储存局部内容，下次使用时就能直接使用缓存。

```
<%- fragment_cache(id, fn);
```

示例：

```
<%- fragment_cache('header', function(){
 return '<header></header>';
}) %>
```

## 日期与时间

### date

插入格式化的日期。`date` 可以是 UNIX 时间、ISO 字符串、`Date` 对象或 [Moment.js](#) 对象。`format` 默认为 `date_format` 配置信息。

```
<%- date(date, [format]) %>
```

示例：

```
<%- date(Date.now()) %>
// 2013-01-01

<%- date(Date.now(), 'YYYY/M/D') %>
// Jan 1 2013
```

### date\_xml

插入 XML 格式的日期。`date` 可以是 UNIX 时间、ISO 字符串、`Date` 对象或 [Moment.js](#) 对象。

```
<%- date_xml(date) %>
```

示例：

```
<%- date_xml(Date.now()) %>
// 2013-01-01T00:00:00.000Z
```

### time

插入格式化的时间。`date` 可以是 UNIX 时间、ISO 字符串、`Date` 对象或 [Moment.js](#) 对象。`format` 默认为 `time_format` 配置信息。

```
<%- time(date, [format]) %>
```

示例：

```
<%- time(Date.now()) %>
// 13:05:12

<%- time(Date.now(), 'h:mm:ss a') %>
// 1:05:12 pm
```

### full\_date

插入格式化的日期和时间。`date` 可以是 UNIX 时间、ISO 字符串、`Date` 对象或 [Moment.js](#) 对象。`format` 默认为 `date_format + time_format`。

```
<%- full_date(date, [format]) %>
```

示例：



```
<%- full_date(new Date()) %>
// Jan 1, 2013 0:00:00

<%- full_date(new Date(), 'dddd, MMMM Do YYYY, h:mm:ss a') %>
// Tuesday, January 1st 2013, 12:00:00 am
```

## moment

[Moment.js](#) 函数库。

## 列表

### list\_categories

插入分类列表。

```
<%- list_categories([options]) %>
```

参数	描述	默认值
<code>orderby</code>	分类排列方式	<code>name</code>
<code>order</code>	分类排列顺序。 <code>1</code> , <code>asc</code> 升序； <code>-1</code> , <code>desc</code> 降序。	<code>1</code>
<code>show_count</code>	显示每个分类的文章总数	<code>true</code>
<code>style</code>	分类列表的显示方式。使用 <code>list</code> 以无序列表（ <code>unordered list</code> ）方式显示。	<code>list</code>
<code>separator</code>	分类间的分隔符号。只有在 <code>style</code> 不是 <code>list</code> 时有用。	<code>,</code>
<code>depth</code>	要显示的分类层级。 <code>0</code> 显示所有层级的分类； <code>-1</code> 和 <code>0</code> 很类似，但是显示不分层级； <code>1</code> 只显示第一层的分类。	<code>0</code>
<code>class</code>	分类列表的 <code>class</code> 名称。	<code>category</code>
<code>transform</code>	改变分类名称显示方法的函数	
<code>suffix</code>	为链接添加前缀	<code>None</code>

用例：

```
<%- list_categories(post.categories, {
 class: 'post-category',
 transform(str) {
 return titlecase(str);
 }
}) %>
<%- list_categories(post.categories, {
 class: 'post-category',
 transform(str) {
 return str.toUpperCase();
 }
}) %>
```

### list\_tags

插入标签列表。

```
<%- list_tags([options]) %>
```

选项	描述	预设值
<code>orderby</code>	标签排列方式	<code>name</code>
<code>order</code>	标签排列顺序。1，asc 升序； -1，desc 降序。	1
<code>show_count</code>	显示每个标签的文章总数	true
<code>style</code>	标签列表的显示方式。使用 <code>list</code> 以无序列表（unordered list）方式显示。	list
<code>separator</code>	标签间的分隔符号。只有在 <code>style</code> 不是 <code>list</code> 时有用。	,
<code>class</code>	标签列表的 <code>class</code> 名称。	tag
<code>transform</code>	改变标签名称显示方法的函数。请查看 <a href="#">list_categories</a> 中给出的例子	
<code>amount</code>	要显示的标签数量（0 = 无限制）	0
<code>suffix</code>	为链接添加前缀	None

## list\_archives

插入归档列表。

```
<%- list_archives([options]) %>
```

参数	描述	默认值
<code>type</code>	类型。此设定可为 <code>yearly</code> 或 <code>monthly</code> 。	monthly
<code>order</code>	排列顺序。1，asc 升序； -1，desc 降序。	1
<code>show_count</code>	显示每个归档的文章总数	true
<code>format</code>	日期格式	MMMM YYYY
<code>style</code>	归档列表的显示方式。使用 <code>list</code> 以无序列表（unordered list）方式显示。	list
<code>separator</code>	归档间的分隔符号。只有在 <code>style</code> 不是 <code>list</code> 时有用。	,
<code>class</code>	归档列表的 <code>class</code> 名称。	archive
<code>transform</code>	改变归档名称显示方法的函数。请查看 <a href="#">list_categories</a> 中给出的例子	

## list\_posts

插入文章列表。

```
<%- list_posts([options]) %>
```

参数	描述	默认值
<code>orderby</code>	文章排列方式	date
<code>order</code>	文章排列顺序。1，asc 升序； -1，desc 降序。	-1
<code>style</code>	文章列表的显示方式。使用 <code>list</code> 以无序列表（unordered list）方式显示。	list
<code>separator</code>	文章间的分隔符号。只有在 <code>style</code> 不是 <code>list</code> 时有用。	,
<code>class</code>	文章列表的 <code>class</code> 名称。	post
<code>amount</code>	要显示的文章数量（0 = 无限制）	6
<code>transform</code>	改变文章名称显示方法的函数。请查看 <a href="#">list_categories</a> 中给出的例子	

## tagcloud

插入标签云。

```
<%- tagcloud([tags], [options]) %>
```

参数	描述	默认值
<code>min_font</code>	最小字体尺寸	10
<code>max_font</code>	最大字体尺寸	20
<code>unit</code>	字体尺寸的单位	px
<code>amount</code>	标签总量	40
<code>orderby</code>	标签排列方式	name
<code>order</code>	标签排列顺序。 1 , sac 升序; -1 , desc 降序	1
<code>color</code>	使用颜色	false
<code>start_color</code>	开始的顏色。您可使用十六進位值 ( #b700ff ) , <code>rgba</code> ( <code>rgba(183, 0, 255, 1)</code> ) , <code>hsla</code> ( <code>hsla(283, 100%, 50%, 1)</code> ) 或 <a href="#">顏色关键字</a> 。此变量仅在 <code>color</code> 参数开启时才有用。	
<code>end_color</code>	結束的顏色。您可使用十六進位值 ( #b700ff ) , <code>rgba</code> ( <code>rgba(183, 0, 255, 1)</code> ) , <code>hsla</code> ( <code>hsla(283, 100%, 50%, 1)</code> ) 或 <a href="#">顏色关键字</a> 。此变量仅在 <code>color</code> 参数开启时才有用。	

## 其他

### paginator

插入分页链接。

```
<%- paginator(options) %>
```

参数	描述	默认值
<code>base</code>	基础网址	/
<code>format</code>	网址格式	page/%d/
<code>total</code>	分页总数	1
<code>current</code>	目前页数	0
<code>prev_text</code>	上一页链接的文字。仅在 <code>prev_next</code> 设定开启时才有用。	Prev
<code>next_text</code>	下一页链接的文字。仅在 <code>prev_next</code> 设定开启时才有用。	Next
<code>space</code>	空白文字	...
<code>prev_next</code>	显示上一页和下一页的链接	true
<code>end_size</code>	显示于两侧的页数	1
<code>mid_size</code>	显示于中间的页数	2
<code>show_all</code>	显示所有页数。如果开启此参数的话, <code>end_size</code> 和 <code>mid_size</code> 就没用了。	false
<code>escape</code>	Escape HTML tags	true

#### Examples:

```
<%- paginator({
 prev_text: '<',
 next_text: '>'
}) %>
```

```
}) %>
```

```
<!-- Rendered as -->
<
1
2
3
>
```

```
<%- paginator({
 prev_text: '<i class="fa fa-angle-left"></i>',
 next_text: '<i class="fa fa-angle-right"></i>',
 escape: false
}) %>
```

```
<!-- Rendered as -->
<i class="fa fa-angle-left"></i>
1
2
3
<i class="fa fa-angle-right"></i>
```

## search\_form

插入 Google 搜索框。

```
<%- search_form(options) %>
```

参数	描述	默认值
<code>class</code>	表单的 class name	<code>search-form</code>
<code>text</code>	搜索提示文字	<code>Search</code>
<code>button</code>	显示搜索按钮。此参数可为布尔值（ <code>boolean</code> ）或字符串，当设定是字符串的时候，即为搜索按钮的文字。	<code>false</code>

## number\_format

格式化数字。

```
<%- number_format(number, [options]) %>
```

参数	描述	默认值
<code>precision</code>	数字精度。此选项可为 <code>false</code> 或非负整数。	<code>false</code>
<code>delimiter</code>	千位数分隔符号	<code>,</code>
<code>separator</code>	整数和小数之间的分隔符号	<code>.</code>

示例：

```
<%- number_format(12345.67, {precision: 1}) %>
// 12,345.68
```

```

<%- number_format(12345.67, {precision: 4}) %>
// 12,345.6700

<%- number_format(12345.67, {precision: 0}) %>
// 12,345

<%- number_format(12345.67, {delimiter: ''}) %>
// 12345.67

<%- number_format(12345.67, {separator: '/'}) %>
// 12,345/67

```

## open\_graph

插入 open graph 资源。

```

<%- open_graph([options]) %>

```

参数	描述	默认值
title	页面标题 ( og:title )	page.title
type	页面类型 ( og:type )	blog
url	页面网址 ( og:url )	url
image	页面图片 ( og:image )	内容中的图片
site_name	网站名称 ( og:site_name )	config.title
description	页面描述 ( og:description )	内容摘要或前 200 字
twitter_card	Twitter 卡片类型 ( twitter:card )	summary
twitter_id	Twitter ID ( twitter:creator )	
twitter_site	Twitter 网站 ( twitter:site )	
google_plus	Google+ 个人资料链接	
fb_admins	Facebook 管理者 ID	
fb_app_id	Facebook 应用程序 ID	

## toc

解析内容中的标题标签 (h1~h6) 并插入目录。

```

<%- toc(str, [options]) %>

```

参数	描述	默认值
class	Class 名称	toc
list_number	显示编号	true

示例：

```

<%- toc(page.content) %>

```



## 国际化（i18n）

若要让您的网站以不同语言呈现，您可使用国际化（internationalization）功能。请先在 `_config.yml` 中调整 `language` 设定，这代表的是预设语言，您也可设定多个语言来调整预设语言的顺位。

```
language: zh-tw

language:
- zh-tw
- en
```

## 语言文件

语言文件可以使用 YAML 或 JSON 编写，并放在主题文件夹中的 `languages` 文件夹。您可以在语言文件中使用 [printf 格式](#)。

## 模板

在模板中，透过 `__` 或 `_p` 辅助函数，即可取得翻译后的字符串，前者用于一般使用；而后者用于复数字符串。例如：

```
en.yml
index:
 title: Home
 add: Add
 video:
 zero: No videos
 one: One video
 other: %d videos
```

```
<%= __('index.title') %>
// Home

<%= _p('index.video', 3) %>
// 3 videos
```

## 路径

您可在 `front-matter` 中指定该页面的语言，也可在 `_config.yml` 中修改 `i18n_dir` 设定，让 Hexo 自动侦测。

```
i18n_dir: :lang
```

`i18n_dir` 的预设值是 `:lang`，也就是说 Hexo 会捕获网址中的第一段以检测语言，举例来说：

```
/index.html => en
/archives/index.html => en
/zh-tw/index.html => zh-tw
```

捕获到的字符串唯有在语言文件存在的情况下，才会被当作是语言，因此例二 `/archives/index.html` 中的 `archives` 就不被当成是语言。





## 插件

Hexo 有强大的插件系统，使您能轻松扩展功能而不用修改核心模块的源码。在 Hexo 中有两种形式的插件：

### 脚本（Scripts）

如果您的代码很简单，建议您编写脚本，您只需要把 JavaScript 文件放到 `scripts` 文件夹，在启动时就会自动载入。

### 插件（Packages）

如果您的代码较复杂，或是您想要发布到 NPM 上，建议您编写插件。首先，在 `node_modules` 文件夹中建立文件夹，文件夹名称开头必须为 `hexo-`，如此一来 Hexo 才会在启动时载入否则 Hexo 将会忽略它。

文件夹内至少要包含 2 个文件：一个是主程序，另一个是 `package.json`，描述插件的用途和所依赖的插件。

```
.
├─ index.js
└─ package.json
```

`package.json` 中至少要包含 `name`，`version`，`main` 属性，例如：

```
package.json
{
 "name": "hexo-my-plugin",
 "version": "0.0.1",
 "main": "index"
}
```

## 工具

您可以使用 Hexo 提供的官方工具插件来加速开发：

- [hexo-fs](#)：文件 IO
- [hexo-util](#)：工具程式
- [hexo-i18n](#)：本地化（i18n）
- [hexo-pagination](#)：生成分页资料

## 发布

当您完成插件后，可以考虑将它发布到 [插件列表](#)，让更多人能够使用您的插件。发布插件的步骤和 [更新文件](#) 非常类似。

1. Fork [hexojs/site](#)
2. 把库（repository）复制到电脑上，并安装所依赖的插件。

```
$ git clone https://github.com/<username>/site.git
$ cd site
$ npm install
```

3. 编辑 `source/_data/plugins.yml`，在档案中新增您的插件，例如：

```
- name: hexo-server
 description: Server module for Hexo.
 link: https://github.com/hexojs/hexo-server
 tags:
```

```
- official
- server
- console
```

4. 推送 (push) 分支。
5. 建立一个新的合并申请 (pull request) 并描述改动。

## 问题解答

在使用 **Hexo** 时，您可能会遇到一些问题，下列的常见问题解答可能会对您有所帮助。如果您在这里找不到解答，可以在 [GitHub](#) 或 [Google Group](#) 上提问。

## YAML 解析错误

```
JS-YAML: incomplete explicit mapping pair; a key node is missed at line 18, column 29:
 last_updated: Last updated: %s
```

如果 **YAML** 字符串中包含冒号（`:`）的话，请加上引号。

```
JS-YAML: bad indentation of a mapping entry at line 18, column 31:
 last_updated:"Last updated: %s"
```

请确认您使用空格进行缩进（**Soft tab**），并确认冒号后有加上一个空格。

您可参阅 [YAML 规范](#) 以取得更多信息。

## EMFILE 错误

```
Error: EMFILE, too many open files
```

虽然 **Node.js** 有非阻塞 **I/O**，同步 **I/O** 的数量仍被系统所限制，在生成大量静态文件的时候，您可能会碰到 **EMFILE** 错误，您可以尝试提高同步 **I/O** 的限制数量来解决此问题。

```
$ ulimit -n 10000
```

（这一命令只对Linux系统有效）

## Git 部署问题

### RPC failed

```
error: RPC failed; result=22, HTTP code = 403

fatal: 'username.github.io' does not appear to be a git repository
```

请确认您已经在电脑上 [配置 git](#)，或改用 **HTTPS** 库（**repository**）地址。

### Error: ENOENT: no such file or directory

如果你遇到了这个错误，有可能是你的文件名、分类或者标签的名字混淆了大写和小写，你可以尝试检查每一个标签和分类的名称，是否大小写一致来修复这一问题。

## 服务器问题

```
Error: listen EADDRINUSE
```

您可能同时开启两个 **Hexo** 服务器，或者有其他应用程序正在占用相同的端口，请尝试修改 `port` 参数，或是在启动 **Hexo** 服务器时加上 `-p` 选项。

```
$ hexo server -p 5000
```

## 插件安装问题

```
npm ERR! node-waf configure build
```

当您尝试安装以 **C/C++** 或其他非 **JavaScript** 语言所编写的插件时，可能会遇到此类问题，请确认您已经在电脑上安装相对应的编译器。

## DTrace 错误（Mac OS X）

```
{ [Error: Cannot find module './build/Release/DTraceProviderBindings'] code: 'MODULE_NOT_FOUND' }
{ [Error: Cannot find module './build/default/DTraceProviderBindings'] code: 'MODULE_NOT_FOUND' }
{ [Error: Cannot find module './build/Debug/DTraceProviderBindings'] code: 'MODULE_NOT_FOUND' }
```

**DTrace** 安装可能有错误，使用下列命令：

```
$ npm install hexo --no-optional
```

参考 [#1326](#)

## 在 **Jade** 或 **Swig** 遍历资料

**Hexo** 使用 [Warehouse](#) 存储资料，它不是一般数组所以必须先进行类型转型才能遍历。

```
{% for post in site.posts.toArray() %}
{% endfor %}
```

## 资料没有更新

有时资料可能没有被更新，或是生成的文件与修改前的相同，您可以尝试清除缓存并再执行一次。

```
$ hexo clean
```

## 命令没有执行

如果你除了 `hexo help`、`hexo init` 和 `hexo version` 以外不能执行任何命令、并且你的任何命令都只返回了 `hexo help` 的内容，这可能是由于 `package.json` 中缺乏 `hexo` 字段导致的。

```
{
 "hexo": {
 "version": "3.9.0"
 }
}
```

## 泄露（**Escape**）内容

Hexo 使用 [Nunjucks](#) 来解析文章（旧版本使用 [Swig](#)，两者语法类似），内容若包含 `{{ }}` 或 `{% %}` 可能导致解析错误，您可以用 `raw` 标签包裹来避免潜在问题发生。

```
{% raw %}
Hello {{ sensitive }}
{% endraw %}
```

## ENOSPC 错误（Linux）

运行 `$ hexo server` 命令有时会返回这样的错误：

```
Error: watch ENOSPC ...
```

它可以用过运行 `$ npm dedupe` 来解决，如果不起作用的话，可以尝试在 Linux 终端中运行下列命令：

```
$ echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl.conf && sudo sysctl -p
```

这将会提高你能监视的文件数量。

## EMPERM Error (Windows Subsystem for Linux)

在执行 `hexo server` 后返回如下错误：

```
Error: watch /path/to/hexo/theme/ EMPERM
```

这是由于你使用的 WSL 版本不支持监听文件系统改动。最新版的 WSL 已经解决了这一问题。

您也仍然可以通过先使用 `hexo g` 生成文件然后将其作为静态服务器运行来从 WSL 环境运行服务器：

```
$ hexo generate
$ hexo server -s
```

关于 WSL 的这一 Issue 请前往 <https://github.com/Microsoft/BashOnWindows/issues/216> 查看。目前这一问题已经得到了解决。

## 模板渲染错误

有的时候你在执行 `hexo generate` 时会返回以下错误信息：

```
FATAL Something's wrong. Maybe you can find the solution here: http://hexo.io/docs/troubleshooting.html
Template render error: (unknown path)
```

这表明你的文件中存在一些不可被识别的字符，比如不可见的零宽度字符。有可能你的新文章存在这个问题，或者你在修改配置文件时导致了这个错误。

检查你的 `_config.yml` 文件中是否漏掉了列表前的空格。你可以查阅 Wikipedia 中 [YAML](#) 相关页面来学习 YAML 语法。


这个是错误的：

```
plugins:
- hexo-generator-feed
- hexo-generator-sitemap
```

正确的应该是这样：

```
plugins:
 - hexo-generator-feed
 - hexo-generator-sitemap
```

## 贡献

We welcome you to join the development of Hexo. 

## 开发

我们非常欢迎您加入 **Hexo** 的开发，这份文件将帮助您了解开发流程。

## 开始之前

请遵守以下准则：

- 遵守 [Google JavaScript 代码风格](#)。
- 使用 2 个空格缩排。
- 不要把逗号放在最前面。

## 工作流程

1. Fork [hexojs/hexo](#)
2. 把库（repository）复制到电脑上，并安装所依赖的插件。

```
$ git clone https://github.com/<username>/hexo.git
$ cd hexo
$ npm install
$ git submodule update --init
```

3. 新增一个功能分支。

```
$ git checkout -b new_feature
```

4. 开始开发。
5. 推送（push）分支。

```
$ git push origin new_feature
```

6. 建立一个新的合并申请（pull request）并描述变动。

## 注意事项

- 不要修改 `package.json` 的版本号。
- 只有在测试通过的情况下您的合并申请才会被批准，在提交前别忘了进行测试。

```
$ npm test
```

## Updating official-plugins

我们也欢迎给 [Hexo 官方插件](#) 提交 PR 和 Issue 

## 更新文档

Hexo 文档开放源代码，您可以在 [hexojs/site](https://github.com/hexojs/site) 找到源代码。

## 工作流程

1. Fork [hexojs/site](https://github.com/hexojs/site)
2. 把库（repository）复制到电脑上，并安装所依赖的插件。

```
$ git clone https://github.com/<username>/site.git
$ cd site
$ npm install
```

3. 开始编辑文件，您可以通过服务器预览变动。

```
$ hexo server
```

4. 推送（push）分支。
5. 建立一个新的合并申请（pull request）并描述变动。

## 翻译

1. 在 `source` 资料夹中建立一个新的语言资料夹（全小写）。
2. 把 `source` 资料夹中相关的文件（Markdown 和模板文件）复制到新的语言资料夹中。
3. 在 `source/_data/language.yml` 中新增语言。
4. 将 `en.yml` 复制到 `themes/navy/languages` 中并命名为语言名称（全小写）。

## 反馈问题

当您在使用 Hexo 时遇到问题，您可以尝试在 [问题解答](#) 中寻找解答，或是在 [GitHub](#) 或 [Google Group](#) 上提问。如果你没有找到答案，请在 Github 报告它。

1. 在 [调试模式](#) 中重现问题。
2. 运行 `hexo version` 并检查版本信息。
3. 把调试信息和版本信息都贴到 [GitHub](#)。