



Jekyll教程

极客学院出版

前言

Jekyll 是一个简单的博客形态的静态站点生产机器。它有一个模版目录，其中包含原始文本格式的文档，通过 [Markdown](#)（或者 [Textile](#)）以及 [Liquid](#) 转化成一个完整的可发布的静态网站，你可以发布在任何你喜爱的服务器上。Jekyll 也可以运行在 [GitHub Page](#) 上，也就是说，你可以使用 GitHub 的服务来搭建你的项目页面、博客或者网站，而且是完全免费的。

内容

该教程的目标是成为 Jekyll 的全面指南。包括一些内容如：搭建和运行你的站点、创建以及管理内容、定制站点的展现和外观、在不同的环境中发布、以及参与到 Jekyll 将来的开发的一些建议。

目录

前言	1
第 1 章 开始旅程	4
快速指南	5
安装	6
基本用法	8
目录结构	10
配置	12
第 2 章 编辑内容	19
头信息	20
撰写博客	23
使用草稿	27
创建页面	28
常用变量	30
Data Files	34
Assets	36
博客迁移	37
第 3 章 定制	38
模板	39
永久链接	45
分页功能	47
插件	51
附加功能	62
第 4 章 部署	64
GitHub Pages	65

	部署方法	67
第 5 章	杂项	70
	常见问题	71
	使用 Jekyll 的站点	74
	相关资源	75
	升级	77
第 6 章	元信息	80
	贡献	81



开始旅程



快速指南

以下是一个获取最简单 Jekyll 模板并生成静态页面的方法。

```
~ $ gem install jekyll
~ $ jekyll new myblog
~ $ cd myblog
~/myblog $ jekyll serve
# => Now browse to http://localhost:4000
```

就是这么简单。从现在开始，你可以通过创建文章、改变头信息来控制模板和输出、修改 Jekyll 设置来使你的站点变得更有趣～

新站点的默认 Markdown 引擎是 Redcarpet, 在 Jekyll 1.1 中，我们改变了默认的 Markdown 引擎，对于 `jekyll new` 产生的新站点，引擎将采用 Redcarpet。 安装过程中有问题？请确认是否安装了所有 [依赖的工具](#)。

安装

安装完成 Jekyll 需要几分钟的时间。如果你觉得安装对你来说并不方便，请 [file an issue](#)（或者提交一个 pull request）来描述一下你的遭遇并告诉我们如何使这个安装过程更加便捷。

事先准备

安装 Jekyll 相当简单，但是你得先做好一些准备工作。开始前你需要确保你在系统里已经有如下配置。

- [Ruby](#)
- [RubyGems](#)
- Linux, Unix, or Mac OS X > 在 Windows 下使用 Jekyll 你可以使用 [Jekyll running on Windows](#)，但是官方文档并不建议你在 Windows 平台上安装 Jekyll。

借助 RubyGems 安装 Jekyll

安装 Jekyll 的最好方式就是使用 [RubyGems](#)。你只需要打开终端输入以下命令就可以安装了：

```
$ gem install jekyll
```

所有的 Jekyll 的 gem 依赖包都会被自动安装，所以你完全不用去担心。如果你在安装中碰到了问题，请查看 [troubleshooting](#) 或者 [report an issue](#) 那么 Jekyll 社区就会帮助你和其他用户解决问题了。

安装 Xcode Command-Line Tools

如果你是 Mac 用户，你就需要安装 Xcode 和 Command-Line Tools 了。下载方式 [Preferences → Downloads → Components](#)。

附加功能

根据每个人使用方式的不同，Jekyll 还支持你安装一些附加功能。包括了对 LaTeX 的支持，以及使用动态内容渲染引擎。查看 [the extras page](#) 获得更多信息。

ProTip™: 启用代码高亮

如果你是一个使用 Jekyll 的程序猿，用 Pygments 来支持代码高亮吧。当然，使用前请先查看 [how to do that](#)。哦耶～你已经安装了所有需要的东西了，开始玩转 Jekyll 博客吧！

基本用法

安装了 Jekyll 的 Gem 包之后，就可以在命令行中使用 Jekyll 命令了。有以下这些用法：

```
$ jekyll build
# => 当前文件夹中的内容将会生成到 ./site 文件夹中。

$ jekyll build --destination <destination>
# => 当前文件夹中的内容将会生成到目标文件夹<destination>中。

$ jekyll build --source <source> --destination <destination>
# => 指定源文件夹<source>中的内容将会生成到目标文件夹<destination>中。

$ jekyll build --watch
# => 当前文件夹中的内容将会生成到 ./site 文件夹中，

# 查看改变，并且自动再生成。
```

Jekyll 同时也集成了一个开发用的服务器，可以让你使用浏览器在本地进行预览。

```
$ jekyll serve
# => 一个开发服务器将会运行在 http://localhost:4000/

$ jekyll serve --detach
# => 功能和`jekyll serve`命令相同，但是会脱离终端在后台运行。

# 如果你想关闭服务器，可以使用`kill -9 1234`命令，“1234”是进程号（PID）。

# 如果你找不到进程号，那么就用`ps aux | grep jekyll`命令来查看，然后关闭服务器。[更多](http://unixhelp.ed.ac.uk/sho

$ jekyll serve --watch
# => 和`jekyll serve`相同，但是会查看变更并且自动再生成。
```

还有一些可以配置的[配置选项](#)。很多配置选项既可以在命令行中作为标识(flags)设定，也可以在源文件根目录中的 `_config.yml` 文件中进行设定。Jekyll 会自动加载这些配置。比如你在你的 `_config.yml` 文件中添加了下面几行：

```
source:      _source
destination: _deploy
```

那么就等价于执行了以下两条命令：

```
$ jekyll build
$ jekyll build --source _source --destination _deploy
```

有关配置选项的更详细说明，请查看[配置](#)页面。

目录结构

Jekyll 的核心其实是一个文本转换引擎。它的概念其实就是：你用你最喜欢的标记语言来写文章，可以是 Markdown，也可以是 Textile，或者就是简单的 HTML，然后 Jekyll 就会帮你套入一个或一系列的布局中。在整个过程中你可以设置 URL 路径，你的文本在布局中的显示样式等等。这些都可以通过纯文本编辑来实现，最终生成的静态页面就是你的成品了。

一个基本的 Jekyll 网站的目录结构一般是像这样的：

```
.
├── _config.yml
├── _drafts
│   ├── begin-with-the-crazy-ideas.textile
│   └── on-simplicity-in-technology.markdown
├── _includes
│   ├── footer.html
│   └── header.html
├── _layouts
│   ├── default.html
│   └── post.html
├── _posts
│   ├── 2007-10-29-why-every-programmer-should-play-nethack.textile
│   └── 2009-04-26-barcamp-boston-4-roundup.textile
├── _data
│   └── members.yml
├── _site
└── index.html
```

来看看这些都有什么用：

文件/目录	描述
<code>_config.yml</code>	保存配置数据。很多配置选项都会直接从命令行中进行设置，但是如果你把那些配置写在这儿，你就不用非要去记住那些命令了。
<code>_drafts</code>	<code>drafts</code> 是未发布的文章。这些文件的格式中都没有 <code>title.MARKUP</code> 数据。学习如何使用 <code>drafts</code> 。
<code>_includes</code>	你可以加载这些包含部分到你的布局或者文章中以便重用。可以用这个标签 <code>{% include file.ext %}</code> 来把文件 <code>_includes/file.ext</code> 包含进来。
<code>_layouts</code>	<code>layouts</code> 是包裹在文章外部的模板。布局可以在 <code>YAML 头信息</code> 中根据不同文章进行选择。这将在下一个部分进行介绍。标签 <code>{{ content }}</code> 可以将 <code>content</code> 插入页面中。
<code>_posts</code>	这里放的就是你的文章了。文件格式很重要，必须要符合： <code>YEAR-MONTH-DAY-title.MARKUP</code> 。The <code>permalinks</code> 可以在文章中自己定制，但是数据和标记语言都是根据文件名来确定的。
<code>_data</code>	Well-formatted site data should be placed here. The jekyll engine will autoload all yaml files (ends with <code>.yml</code> or <code>.yaml</code>) in this directory. If there's a file <code>members.yml</code> under the directory, then you can access contents of the file through <code>site.data.members</code> .
<code>_site</code>	一旦 Jekyll 完成转换，就会将生成的页面放在这里（默认）。最好将这个目录放进你的 <code>.gitignore</code> 文件中。
<code>index.html</code> and other HTML, Markdown, Textile files	如果这些文件中包含 <code>YAML 头信息</code> 部分，Jekyll 就会自动将它们进行转换。当然，其他的如 <code>.html</code> ， <code>.markdown</code> ， <code>.md</code> ，或者 <code>.textile</code> 等在你的站点根目录下或者不是以上提到的目录中的文件也会被转换。
Other Files/Folders	其他一些未被提及的目录和文件如 <code>css</code> 还有 <code>images</code> 文件夹， <code>favicon.ico</code> 等文件都将被完全拷贝到生成的 <code>site</code> 中。这里有一些使用 Jekyll 的站点，如果你感兴趣就来看看吧。

配置

Jekyll 帮你轻松的搭建你的网站，这很大程度上归功于灵活强大的配置功能。既可以在网站根目录下的 `_config.yml` 文件中进行配置，也可以作为命令行参数来配置。

配置设置

全局配置

下表中列举了所有 Jekyll 可用的设置，和多种多样的 `配置项`（配置文件中）及 `参数`（命令行中）。

设置	配置项和参数
Site Source 改变 Jekyll 读取文件的目录	<code>source: DIR</code> <code>-s, --source DIR</code>
Site Destination 改变 Jekyll 写入文件的目录	<code>destination: DIR</code> <code>-d, --destination DIR</code>
Safe 禁用 自定义插件。	<code>safe: BOOL</code> <code>--safe</code>
Exclude 转换时排除某些文件夹或文件。被排除的文件或文件夹的路径是相对于网站源码目录的，源码目录以外不受影响。	<code>exclude: [DIR, FILE, ...]</code>
Include 转换时强制包含某些文件、文件夹。 <code>.htaccess</code> 是个典型的例子，因为默认排除 <code>.</code> (dot, 英文中的句号) 开头的文件。	<code>include: [DIR, FILE, ...]</code>
Time Zone 设置时区，这个设置作用于 <code>TZ</code> 变量，Ruby 用它来处理日期和时间。 IANA Time Zone Database 里边的都有效，比如 <code>America/New_York</code> 。默认值为操作系统的时区。	<code>timezone: TIMEZONE</code>
Encoding 设置文件的编码，仅 Ruby 1.9 以上可用。默认值为 <code>nil</code> ，使用 Ruby 默认的 <code>ASCII-8BIT</code> 。可以用命令 <code>ruby -e 'puts Encoding::list.join("\n")'</code> 查看 Ruby 可用的编码。	<code>encoding: ENCODING</code>

编译选项

设置	配置项和参数
Regeneration 允许文件修改时自动重新生成网站。	<code>-w, --watch</code>
Configuration 手动设置配置文件并替代 <code>_config.yml</code> , 可设置多个, 且后边的设置会覆盖前边的。	<code>--config FILE1[,FILE2,...]</code>
Drafts 处理草稿	<code>--drafts</code>
Future 用将来的日期发布文章	<code>future: BOOL</code> <code>--future</code>
LSI 为相关文章生成索引	<code>lsi: BOOL</code> <code>--lsi</code>
Limit Posts 限制文章的数量	<code>limit_posts: NUM</code> <code>--limit_posts NUM</code>

服务选项

除了下边的选项, `serve` 命令还可以接收 `build` 的选项, 当运行网站服务之前的编译时候使用。

设置	配置项和参数
Local Server Port 监听所给的端口	<code>port: PORT</code> <code>--port PORT</code>
Local Server Hostname 监听所给的主机名	<code>host: HOSTNAME</code> <code>--host HOSTNAME</code>
Base URL 网站的根路径	<code>baseurl: URL</code> <code>--baseurl URL</code>
Detach 从终端命令行中分离出来	<code>detach: BOOL</code> <code>-B, --detach</code>

不要在配置文件中使用 `tab` 制表符

这将造成解析错误，或倒回到默认设置。请使用空格替代。

默认配置

Jekyll 默认使用以下的配置。除非在配置文件中存在相同的配置项或在命令行中指定参数，否则 Jekyll 将使用以下配置运行。

有两个 `kramdown` 的选项不被支持

注意 Jekyll 目前不支持 `remove_block_html_tags` 和 `remove_span_html_tags`，因为没有被包含到 `kramdown HTML 转换器`中。

```
source:      .
destination: ./_site
plugins:     ./_plugins
layouts:     ./_layouts
include:     ['.htaccess']
exclude:     []
keep_files:  ['.git', '.svn']
gems:        []
timezone:    nil
encoding:    nil

future:      true
```



```

show_drafts: nil
limit_posts: 0
highlighter: pygments

relative_permalink: true

permalink:      date
paginate_path: 'page:num'
paginate:       nil

markdown:       maruku
markdown_ext:   markdown, mkd, mkdn, md
textile_ext:    textile

excerpt_separator: "\n\n"

safe:           false
watch:          false # deprecated
server:         false # deprecated
host:           0.0.0.0
port:           4000
baseurl:        /
url:            http://localhost:4000
lsi:            false

maruku:
  use_tex:      false
  use_divs:     false
  png_engine:   blahtex
  png_dir:      images/latex
  png_url:      /images/latex
  fenced_code_blocks: true

rdiscount:
  extensions: []

redcarpet:
  extensions: []

kramdown:
  auto_ids: true
  footnote_nr: 1
  entity_output: as_char
  toc_levels: 1..6
  smart_quotes: lsquo, rsquo, ldquo, rdquo

```

```

use_coderay: false

coderay:
  coderay_wrap: div
  coderay_line_numbers: inline
  coderay_line_numbers_start: 1
  coderay_tab_width: 4
  coderay_bold_every: 10
  coderay_css: style

redcloth:
  hard_breaks: true

```

Markdown 选项

Jekyll 支持的 Markdown 渲染器中有的有额外的选项。

Redcarpet

Redcarpet 支持设置 `extensions`，值为一个字符串数组，每个字符串都是 `Redcarpet::Markdown` 类的扩展，相应的扩展就会设置为 `true`。

Jekyll handles two special Redcarpet extensions: - `no_fenced_code_blocks` — 默认的，Jekyll 设置扩展 `fenced_code_blocks`（用三个波浪线或重音线标记代码区间）为 `true`，这或许是跟 GitHub 积极的采用有关。当使用 Jekyll 的时候，Redcarpet 的扩展 `fenced_code_blocks` 无效，作为替代方案，你可以这样做：注意你还可以这样来配置语言以支持语法高亮：

```

ruby
# ...ruby code

```

有了 both fenced code blocks 和 pygments，就会直接高亮代码了；如果没有 pygments，将增加一个 `class="LANGUAGE"` 属性到 `<code>` 元素，用于给不同的 JavaScript 代码高亮库做后续处理。- `smart` — 打开 Smartypants，将引号转为 ”、连字符转为 em（`---`）和 en（`--`）破折号。Redcarpet 所有其他扩展保持他们本来的名字，并且在 Jekyll 中不能给 `smart` 加渲染选项。[Redcarpet 的 README 中有可用扩展的列表](#)。确保你看的 README 是正确的版本：Jekyll 当前用的是 v2.2.x，其中 `footnotes` 和 `highlight` 在 3.0.0 以后才会支持。最常用的扩展是如下：

- `tables`
- `no_intra_emphasis`

- `autolink`

Kramdown

除了上述提到的默认配置，你还可以给 Kramdown 传递值为 “GFM” 的 `input` 选项，从而启用 Github 扩展的 Markdown 语法。

例如，在 `_config.yml` 文件中加入以下配置项：`kramdown: input: GFM`



编辑内容



头信息

正是头信息开始让 Jekyll 变的很酷。任何只要包含 [YAML](#) 头信息的文件在 Jekyll 中都能被当做一个特殊的文件来处理。头信息必须在文件的开始部分，并且需要按照 YAML 的格式写在两行三虚线之间。下面是一个基本的例子：

```
---
layout: post
title: Blogging Like a Hacker
---
```

在这两行的三虚线之间，你可以设置一些预定义的变量（下面这个例子可以作为参考）或者甚至创建一个你自己定义的变量。这样在接下来的文件和任意模板中或者在包含这些页面或博客的模板中都可以通过使用 Liquid 标签来访问这些变量。

UTF-8 编码方式警告

如果你使用 UTF-8 编码，那么在你的文件中一定不要出现 `BOM` 头字符，否则你会碰上非常糟糕的事情，尤其当你在 Windows 上使用 Jekyll 的时候。

提示™：头信息变量是可选的

如果你想使用 [Liquid 标签和变量](#)但是在头信息中又不需要任何定义，那么你可以将头信息设置为空！在头信息为空的情况下，Jekyll 仍然能够处理文件。（这对于一些像 CSS 和 RSS 的文件非常有用）

预定义的全局变量

你可以在页面或者博客的头信息处使用一些已经预定义好的全局变量。

变量名称	描述
<code>layout</code>	如果设置的话，会指定使用该模板文件。指定模板文件时候不需要扩展名。模板文件需要放在 <code>_layouts</code> 目录下。
<code>permalink</code>	如果你需要让你的博客中的URL地址不同于默认值 <code>/year/month/day/title.html</code> 这样，那么当你设置这个变量后，就会使用最终的URL地址。
<code>published</code>	当站点生成的时候，如果你不需要展示一个具体的博文，可以设置这个变量为 <code>false</code> 。
<code>category</code> <code>categories</code>	除过将博客文章放在某个文件夹下面外，你还可以根据文章的类别来给他们设置一个或者多个分类属性。这样当你的博客生成的时候这些文章就可以根据这些分类来阅读。在一个文章中多个类别可以通过 YAML list 来指定，或者用空格隔开。
<code>tags</code>	类似分类，一篇文章也可以给它增加一个或者多个标签。同样多个标签之间可以通过 YAML 列表 或者空格隔开。

自定义变量

在头信息中没有预先定义的任何变量都会在数据转换中通过 Liquid 模板被调用。例如，在头信息中你设置一个 `title`，然后就可以在你的模板中使用这个 `title` 变量来设置页面的 `title` 属性：

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>{{ page.title }}</title>
  </head>
  <body>
    ...
```

在文章中预定义的变量

在文章中使用这些在头信息变量列表中未包含的变量。

变量名称	描述
date	这里的日期会覆盖文章名字中的日期。这样就可以用来确定文章分类的正确。

撰写博客

Jekyll 的一个最好的特点是“关注 blog 本身”。这是指什么呢？简单的说就是写博客的过程被铸造进了 Jekyll 的功能中。你只需简单的管理你电脑中的一个文件夹下的文本文件就可以写文章并方便的在线上发布。与繁琐的配置和维护数据库和基于网站的内容管理系统（CMS）相比，这是一个非常受欢迎的改变。

文章文件夹

在[目录结构](#)介绍中说明过，所有的文章都在 `_posts` 文件夹中。这些文件可以用 Markdown 编写，也可以用 Textile 格式编写。只要文件中有 YAML 头信息，它们就会从源格式转化成 HTML 页面，从而成为你的静态网站的一部分。

创建文章的文件

发表一篇新文章，你所需要做的就是 在 `_posts` 文件夹中创建一个新的文件。文件名的命名非常重要。Jekyll 要求一篇文章的文件名遵循下面的格式：

```
年-月-日-标题.MARKUP
```

在这里，`年` 是4位数字，`月` 和 `日` 都是2位数字。`MARKUP` 扩展名代表了这篇文章是用什么格式写的。下面是一些合法的文件名的例子：

```
2011-12-31-new-years-eve-is-awesome.md
2012-09-12-how-to-write-a-blog.textile
```

内容格式

所有博客文章顶部必须有一段 YAML 头信息 (YAML front-matter)。在它下面，就可以选择你喜欢的格式来写文章。Jekyll 支持2种流行的标记语言格式：Markdown 和 Textile。这些格式都有自己的方式来标记文章中不同类型的内容，所以你首先需要熟悉这些格式并选择一种最符合你需求的。

```
Be aware of character sets
```

```
Content processors can modify certain characters to make them look nicer. For example, the smart extension in Redcarpet converts standard, ASCII quotation characters to curly, Unicode ones. In or
```


der for the browser to display those characters properly, define the charset meta value by including `<meta charset="utf-8">` in the `<head>` of your layout.

引用图片和其它资源

很多时候，你需要在文章中引用图片、下载或其它数字资源。尽管 Markdown 和 Textile 在链接这些资源时的语法并不一样，但你只需要关心在站点的哪些地方保存这些文件。

由于 Jekyll 的灵活性，有很多方式可以解决这个问题。一种常用做法是在工程的根目录下 创建一个文件夹，命名为 `assets` 或者 `downloads`，将图片文件，下载文件或者其它的 资源放到这个文件夹下。然后在任何一篇文章中，它们都可以用站点的根目录来进行引用。 这和你站点的域名/二级域名和目录的设置相关，下面有一些例子（Markdown 格式） 来演示怎样利用 `site.url` 变量来解决这个问题。

在文章中引用一个图片

… 从下面的截图可以看到：
 ![有帮助的截图]({{ site.url }}/assets/screenshot.jpg)

链接一个读者可下载的 PDF 文件：

… 你可以直接 [下载 PDF]({{ site.url }}/assets/mydoc.pdf).

提示™：链接只使用站点的根 URL

如果你确信你的站点只在域名的根 URL 下做展示，你可以不使用 `{{ site.url }}` 变量。在这种情况下， 直接使用 `/path/file.jpg` 即可。

文章的目录

所有文章都在一个目录中是没有问题的，但是如果你不将文章列表列出来博客文章是不会被人看到 。在另一个页面上创建文章的列表（或者使用模版）是很简单的。 感谢 Liquid 模版语言和它的标记，下面 是如何创建文章列表的简单例子：

```
<ul>
  {% for post in site.posts %}
    <li>
      <a href="{{ post.url }}">{{ post.title }}</a>
    </li>
  {% endfor %}
</ul>
```

当然，你可以完全控制怎样（在哪里）显示你的文章，如何管理你的站点。如果你想了解 更多你需要读一下 [Jekyll 的模版是怎样工作的](#)这篇文章。

文章摘要

Jekyll 会自动取每篇文章从开头到第一次出现 `excerpt_separator` 的地方作为文章的摘要，并将此内容保存到变量 `post.excerpt` 中。拿上面生成文章列表的例子，你可能想在每个标题下给出文章内容的提示，你可以在每篇文章 的第一段加上如下的代码：

```
<ul>
  {% for post in site.posts %}
    <li>
      <a href="{{ post.url }}">{{ post.title }}</a>
      {{ post.excerpt }}
    </li>
  {% endfor %}
</ul>
```

Because Jekyll grabs the first paragraph you will not need to wrap the excerpt in `p` tags, which is already done for you. These tags can be removed with the following if you'd prefer:

```
{{ post.excerpt | remove: '<p>' | remove: '</p>' }}
```

如果你不喜欢自动生成摘要，你可以在文章的 YAML 中增加 `excerpt` 来覆盖它。完全禁止掉可以将 `excerpt_separator` 设置成 `""`。

Also, as with any output generated by Liquid tags, you can pass the `| strip_html` flag to remove any html tags in the output. This is particularly helpful if you wish to output a post excerpt as a `meta="description"` tag within the post `head`, or anywhere else having html tags along with the content is not desirable.

高亮代码片段

Jekyll 自带语法高亮功能，它是由 Pygments 来实现的。在文章中插入一段高亮代码非常容易，只需使用下面的 Liquid 标记：

```
{% highlight ruby %}
def show
  @widget = Widget(params[:id])
  respond_to do |format|
```

```

    format.html # show.html.erb
    format.json { render json: @widget }
  end
end
{% endhighlight %}

```

将输出下面的效果：

```

def show
  @widget = Widget(params[:id])
  respond_to do |format|
    format.html # show.html.erb
    format.json { render json: @widget }
  end
end
end

```

提示™：显示行数

你可以在代码片段中增加关键字 `linenos` 来显示行数。这样完整的高亮开始标记将会是：`{% highlight ruby linenos %}`。有了这些基础知识就可以开始你的第一篇文章了。当你准备更深入的了解还可以做什么的时候，你可能会对如何[定制文章的永久链接](#)或在文章和站点的其它位置中使用[定制变量](#)感兴趣。

使用草稿

草稿是没有日期的文章。它们是你还在创作中而暂时不想发表的文章。想要开始使用草稿，你需要在网站根目录下创建一个名为 `_drafts` 的文件夹（如在[目录结构](#)章节里描述的），并新建你的第一份草稿：

```
|-- _drafts/  
|   |-- a-draft-post.md
```

为了预览你拥有草稿的网站，运行 `jeekyll serve` 或者带有 `--drafts` 配置选项的 `jeekyll build`。此两种方法皆会将 `Time.now` 的值赋予草稿文章，作为其发布日期，所以你将看到草稿文章作为最新文章被生成。

创建页面

作为[写文章](#)的补充，Jekyll 还可以创建静态页面。利用 Jekyll 带来的便利，你只需要复制文件或文件夹，就是这么简单。

主页

像任何网站的配置一样，需要按约定在站点的要目录下找到 `index.html` 文件，这个文件将被做为主页显示出来。除非你的站点设置了其它的文件作为默认文件，这个文件就将是你的 Jekyll 生成站点的主页。

提示™：在主页上使用布局

站点上任何 HTML 文件，包括主页，都可以使用布局和 `include` 中的内容一般共用的内容，如页面的 `header` 和 `footer`。将合适的部分抽出放到布局中。

其它的页面的位置

将 HTML 文件放在哪里取决于你想让它们如何工作。有两种方式可以创建页面：

- 命名 HTML 文件：将命名好的为页面准备的 HTML 文件放在站点的根目录下。
 - 命名文件夹：在站点的根目录下为每一个页面创建一个文件夹，并把 `index.html` 文件放在每个文件夹里。
- 这两种方法都可以工作（并且可以混合使用），它们唯一的区别就是访问的 URL 样式不同。

命名 HTML 文件

增加一个新页面的最简单方法就是把给 HTML 文件起一个适当的名字并放在根目录下。一般来说，一个站点下通常会有：主页（homepage），关于（about），和一个联系（contact）页。根目录下的文件结构和对应生成的 URL 会是下面的样子：

```
.
|-- _config.yml
|-- _includes/
|-- _layouts/
|-- _posts/
|-- _site/
|-- about.html    # => http://example.com/about.html
```

```
|-- index.html    # => http://example.com/
|   |-- contact.html # => http://example.com/contact.html
```

命名一个文件夹并包含一个 index.html 文件

上面的方法可以很好的工作，但是有些人不喜欢在 URL 中显示文件的扩展名。用 Jekyll 达到这种效果，你只需要为每个顶级页面创建一个文件夹，并包含一个 `index.html` 文件。这样，每个 URL 就将以文件夹的名字作为结尾，网站服务器会将对应的 `index.html` 展示给用户。下面是一个示例来展示这种结构的样子：

```
.
|-- _config.yml
|-- _includes/
|-- _layouts/
|-- _posts/
|-- _site/
|-- about/
|   |-- index.html # => http://example.com/about/
|-- contact/
|   |-- index.html # => http://example.com/contact/
|-- index.html    # => http://example.com/
```

这种方式可能不适合每个人，对那些喜欢干净 URL 的人这是一种简单有效的方法。最终选择哪种方法完全由你来决定！

常用变量

Jekyll 会遍历你的网站搜寻要处理的文件。任何有 YAML 头信息的文件都是要处理的对象。对于每一个这样的文件，Jekyll 都会通过 Liquid 模板工具来生成一系列的数据。下面就是这些可用数据变量的参考和文档。

全局(Global)变量

变量	说明
<code>site</code>	来自 <code>_config.yml</code> 文件，全站范围的信息 + 配置。详细的信息请参考下文
<code>page</code>	页面专属的信息 + YAML 头文件信息 。通过 YAML 头文件自定义的信息都可以在这里被获取。详情请参考下文。
<code>content</code>	被 layout 包裹的那些 Post 或者 Page 渲染生成的内容。但是又没定义在 Post 或者 Page 文件中的变量。
<code>paginator</code>	每当 <code>paginate</code> 配置选项被设置了的时候，这个变量就可用了。详情请看， 分页 。

全站(site)变量

变量	说明
<code>site.time</code>	当前时间（跑 <code>jeekyll</code> 这个命令的时间点）。
<code>site.pages</code>	所有 Pages 的清单。
<code>site.posts</code>	一个按照时间倒叙的所有 Posts 的清单。
<code>site.related_posts</code>	如果当前被处理的页面是一个 Post，这个变量就会包含最多10个相关的 Post。默认的情况下，相关性是低质量的，但是能被很快的计算出来。如果你需要高相关性，就要消耗更多的时间来计算。用 <code>jeekyll</code> 这个命令带上 <code>--lsi</code> (latent semantic indexing) 选项来计算高相关性的 Post。
<code>site.categories.CATEGORY</code>	所有的在 <code>CATEGORY</code> 类别下的帖子。
<code>site.tags.TAG</code>	所有的在 <code>TAG</code> 标签下的帖子。
<code>site.[CONFIGURATION_DATA]</code>	所有的通过命令行和 <code>_config.yml</code> 设置的变量都会存到这个 <code>site</code> 里面。举例来说，如果你设置了 <code>url: http://mysite.com</code> 在你的配置文件中，那么在你的 Posts 和 Pages 里面，这个变量就被存储在了 <code>site.url</code> 。Jekyll 并不会把对 <code>_config.yml</code> 做的改动放到 <code>watch</code> 模式，所以你每次都要重启 Jekyll 来让你的变动生效。

页面 (page) 变量

变量	说明
<code>page.content</code>	页面内容的源码。
<code>page.title</code>	页面的标题。
<code>page.excerpt</code>	页面摘要的源码。
<code>page.url</code>	帖子以斜线打头的相对路径，例子： <code>/2008/12/14/my-post.html</code> 。
<code>page.date</code>	帖子的日期。日期的可以在帖子的头信息中通过用以下格式 <code>YYYY-MM-DD HH:MM:SS</code> (假设是 UTC), 或者 <code>YYYY-MM-DD HH:MM:SS +/-TTTT</code> (用于声明不同于 UTC 的时区，比如 <code>2008-12-14 10:30:00 +0900</code>) 来显示声明其他日期/时间的方式被改写，
<code>page.id</code>	帖子的唯一标识码（在RSS源里非常有用），比如 <code>/2008/12/14/my-post</code>
<code>page.categories</code>	这个帖子所属的 Categories。Categories 是从这个帖子的 <code>_posts</code> 以上的目录结构中提取的。距离来说，一个在 <code>/work/code/_posts/2008-12-24-closures.md</code> 目录下的 Post，这个属性就会被设置成 <code>['work', 'code']</code> 。不过 Categories 也能在 YAML 头文件信息 中被设置。
<code>page.tags</code>	这个 Post 所属的所有 tags。Tags 是在 YAML 头文件信息 中被定义的。
<code>page.path</code>	Post 或者 Page 的源文件地址。举例来说，一个页面在 GitHub 上得源文件地址。这可以在 YAML 头文件信息 中被改写。

ProTip™: Use custom front-matter 任何你自定义的头文件信息都会在 `page` 中可用。距离来说，如果你在一个 Page 的头文件中设置了 `custom_css: true`，这个变量就可以这样被取到 `page.custom_css`。

分页器(Paginator)

变量	说明
<code>paginator.per_page</code>	每一页Posts的数量。
<code>paginator.posts</code>	这一页可用的Posts。
<code>paginator.total_posts</code>	Posts 的总数。
<code>paginator.total_pages</code>	Pages 的总数。
<code>paginator.page</code>	当前页号。
<code>paginator.previous_page</code>	前一页的页号。
<code>paginator.previous_page_path</code>	前一页的地址。
<code>paginator.next_page</code>	下一页的页号。
<code>paginator.next_page_path</code>	下一页的地址。

分页器变量的可用性

这些变量仅在首页文件中可以，不过他们也会存在于子目录中，就像 `/blog/index.html`。

Data Files

In addition to the [built-in variables](#) available from Jekyll, you can specify your own custom data that can be accessed via the [Liquid templating system](#).

Jekyll supports loading data from [YAML](#) files located in the `_data` directory.

This powerful feature allows you to avoid repetition in your templates and to set site specific options without changing `_config.yml`.

Plugins/themes can also leverage Data Files to set configuration variables.

The Data Folder

As explained on the [directory structure](#) page, the `_data` folder is where you can store additional data for Jekyll to use when generating your site. These files must be YAML files (using either the `.yml` or `.yaml` extension) and they will be accessible via `site.data`.

Example: List of members

Here is a basic example of using Data Files to avoid copy-pasting large chunks of code in your Jekyll templates:

In `_data/members.yml`:

```
- name: Tom Preston-Werner
  github:mojombo

- name: Parker Moore
  github: parkr

- name: Liu Fengyun
  github: liufengyun
```

This data can be accessed via `site.data.members` (notice that the filename determines the variable name).

You can now render the list of members in a template:

```
<ul>
{% for member in site.data.members %}
  <li>
    <a href="https://github.com/{{ member.github }}">
      {{ member.name }}
    </a>
  </li>
{% endfor %}
</ul>
```

Assets

Jekyll provides built-in support for Sass and CoffeeScript. In order to use them, create a file with the proper extension name (one of `.sass`, `.scss`, or `.coffee`) and start the file with two lines of triple dashes, like this:

```
##
---
// start content
.my-definition
  font-size: 1.2em
```

Sass/SCSS

Jekyll allows you to customize your Sass conversion in certain ways.

If you are using Sass `@import` statements, you'll need to ensure that your `sass_dir` is set to the base directory that contains your Sass files. You can do that thusly:

```
sass:
  sass_dir: _sass
```

The Sass converter will default to `_sass`.

You may also specify the output `style` with the `style` option in your `_config.yml` file:

```
sass:
  style: :compressed
```

These are passed to Sass, so any output style options Sass supports are valid here, too.

博客迁移

如果你要从其他博客系统迁移到 Jekyll，Jekyll 的导入器可以帮助你。请参考 [jekyll-import 文档](#) 以获取更多站点迁移的知识。



T



3

定制



模板

Jekyll 使用 [Liquid](#) 模板语言，支持所有标准的 Liquid [标签](#) 和 [过滤器](#)。Jekyll 甚至增加了几个过滤器和标签，方便使用。

过滤器

描述	过滤器和输出
日期转化为 XML 模式 将日期转化为 XML 模式 (ISO 8601) 的格式。	<pre>{{ site.time date_to_xmlschema }}</pre> <pre>2008-11-17T13:07:54-08:00</pre>
日期转化为 RFC-822 格式 将日期转化为 RFC-822 格式，用于 RSS 订阅。	<pre>{{ site.time date_to_rfc822 }}</pre> <pre>Mon, 17 Nov 2008 13:07:54 -0800</pre>
日期转化为短格式 将日期转化为短格式。	<pre>{{ site.time date_to_string }}</pre> <pre>17 Nov 2008</pre>
日期转化为长格式 将日期转化为长格式。	<pre>{{ site.time date_to_long_string }}</pre> <pre>17 November 2008</pre>
XML 转码 对一些字符串转码，已方便显示在 XML。	<pre>{{ page.content xml_escape }}</pre>
CGI 转码 CGI 转码，用于 URL 中，将所有的特殊字符转化为 %XX 的形式。	<pre>{{ "foo,bar;baz?" cgi_escape }}</pre> <pre>foo%2Cbar%3Bbaz%3F</pre>
URI 转码 URI 转码。	<pre>{{ "'foo, bar \\baz?" uri_escape }}</pre> <pre>foo,%20bar%20%5Cbaz?</pre>
统计字数 统计文章中的字数。	<pre>{{ page.content number_of_words }}</pre> <pre>1337</pre>
数组转换为句子 将数组转换为句子，列举标签时尤其有用。	<pre>{{ page.tags array_to_sentence_string }}</pre> <pre>foo, bar, and baz</pre>
Textile 支持 将 Textile 格式的字符串转换为 HTML，使用 RedCloth	<pre>{{ page.excerpt textilize }}</pre>
Markdown 支持 将 Markdown 格式的字符串转换为 HTML。	<pre>{{ page.excerpt markdownify }}</pre>
Data To JSON	<pre>{{ site.data.projects jsonify }}</pre>

标签

引用

如果你需要在多个地方引用一小代码片段，可以使用 `include` 标签。

```
{% include footer.html %}
```

Jekyll 要求所有被引用的文件放在根目录的 `_includes` 文件夹，上述代码将把 `<source>/_includes/footer.html` 的内容包含进来。

ProTip™: Use variables as file name

The name of the file you wish to embed can be literal (as in the example above), or you can use a variable, using liquid-like variable syntax as in `{% include {{ my_variable }} %}`.

你还可以传递参数：

```
{% include footer.html param="value" %}
```

这些变量可以通过 Liquid 调用：

```
{{ include.param }}
```

Code snippet highlighting

Jekyll 已经支持 [超过 100 种语言](#) 代码高亮显示，在此感谢 [Pygments](#)。要使用 Pygments，你必须安装 Python 并且在配置文件中设置 `pygments` 为 `true`。

Alternatively, you can use [Rouge](#) to highlight your code snippets. It doesn't support as many languages as Pygments does but it should fit in most cases and it's written in pure Ruby ; you don't need Python on your system!

使用代码高亮的例子如下：

```
{% highlight ruby %}
def foo
  puts 'foo'
end
{% endhighlight %}
```

`highlight` 的参数（本例中的 `ruby`）是识别所用语言，要使用合适的识别器可以参照 [Lexers 页](#) 的 “short name”。

行号

`highlight` 的第二个可选参数是 `linenos`，使用了 `linenos` 会强制在代码上加入行号。例如：

```
{% highlight ruby linenos %}
def foo
  puts 'foo'
end
{% endhighlight %}
```

代码高亮的样式

要使用代码高亮，你还需要包含一个样式。例如你可以在 [syntax.css](#) 找到，这里有跟 GitHub 一样的样式，并且免费。如果你使用了 `linenos`，可能还需要在 `syntax.css` 加入 `.lineno` 样式。

Post URL

如果你想使用你某篇文章的链接，标签 `post_url` 可以满足你的需求。

```
{% post_url 2010-07-21-name-of-post %}
```

If you organize your posts in subdirectories, you need to include subdirectory path to the post:

```
{% post_url /subdir/2010-07-21-name-of-post %}
```

当使用 `post_url` 标签时，不需要写文件后缀名。

还可以用 Markdown 这样为你的文章生成超链接：

```
[Name of Link]({% post_url 2010-07-21-name-of-post %})
```

Gist

使用 `gist` 标签可以轻松的把 GitHub Gist 签入到网站中：

```
{% gist 5555251 %}
```

你还可以配置 `gist` 的文件名，用以显示：

```
{% gist 5555251 result.md %}
```

`gist` 同样支持私有的 gists，这需要 gist 所属的 github 用户名：

```
{% gist parkr/931c1c8d465a04042403 %}
```

私有的 gist 同样支持文件名。

永久链接

Jekyll 支持以灵活的方式管理你网站的链接，你可以通过 [Configuration](#) 或 [YAML 头信息](#) 为每篇文章设置永久链接。你可以随心所欲的选择你自己的 格式，即使自定义。默认配置为 `date`。

永久链接的模板用以冒号为前缀的关键词标记动态内容，比如 `date` 代表 `/:categories/:year/:month/:day/:title.html`。

模板变量

变量	描述
<code>year</code>	文章所在文件的年份
<code>month</code>	文章所在文件的月份，格式如 <code>`01, 10`</code>
<code>i_month</code>	文章所在文件的月份，格式如 <code>`1, 10`</code>
<code>day</code>	文章所在文件的日期，格式如 <code>`01, 20`</code>
<code>i_day</code>	文章所在文件的日期，格式如 <code>`1, 20`</code>
<code>title</code>	文章所在文件的标题
<code>categories</code>	为文章配置的目录，Jekyll 可以自动将 <code>//</code> 转换为 <code>/</code> ，所以如果没有目录，会自动忽略

已经建好的链接类型

链接类型	URL 模板
date	<code>/:categories/:year/:month/:day/:title.html</code>
pretty	<code>/:categories/:year/:month/:day/:title/</code>
none	<code>/:categories/:title.html</code>

举例

比如文件名: `/2009-04-29-slap-chop.textile`

设置	对应的 URL
没有配置或 <code>permalink: date</code>	<code>/2009/04/29/slap-chop.html</code>
<code>permalink: pretty</code>	<code>/2009/04/29/slap-chop/index.html</code>
<code>permalink: /:month-:day-:year/:title.html</code>	<code>/04-29-2009/slap-chop.html</code>
<code>permalink: /blog/:year/:month/:day/:title</code>	<code>/blog/2009/04/29/slap-chop/index.html</code>

分页功能

对于大多数网站（尤其是博客），当文章越来越多的时候，就会有分页显示文章列表的需求。Jekyll 已经自建分页功能，你只需要根据约定放置文件即可。

分页功能只支持 HTML 文件

Jekyll 的分页功能不支持 Markdown 或 Textile 文件，而是只支持 HTML 文件。当然，这不会让你不爽。

开启分页功能

开启分页功能很简单，只需要在 `_config.yml` 里边加一行，并填写每页需要几行：

```
paginate: 5
```

下边是对需要带有分页页面的配置：

```
paginate_path: "blog/page:num"
```

`blog/index.html` 将会读取这个设置，把他传给每个分页页面，然后从第 2 页开始输出到 `blog/page:num`，`:num` 是页码。如果有 12 篇文章并且做如下配置 `paginate: 5`，Jekyll 会将前 5 篇文章写入 `blog/index.html`，把接下来的 5 篇文章写入 `blog/page2/index.html`，最后 2 篇写入 `blog/page3/index.html`。

与 paginator 相同的属性

属性	描述
<code>page</code>	当前页码
<code>per_page</code>	每页文章数量
<code>posts</code>	当前页的文章列表
<code>total_posts</code>	总文章数
<code>total_pages</code>	总页数
<code>previous_page</code>	上一页页码 或 <code>nil</code>
<code>previous_page_path</code>	上一页路径 或 <code>nil</code>
<code>next_page</code>	下一页页码 或 <code>nil</code>
<code>next_page_path</code>	下一页路径 或 <code>nil</code>

不支持对“标签”和“类别”分页

分页功能仅仅遍历文章列表并计算出结果，并无读取 YAML 头信息，现在不支持对“标签”和“类别”分页。

生成带分页功能的文章

接下来要做的事情就是展现在页面上了，下边是一个简单的例子：

```
---
layout: default
## title: My Blog
<!-- 遍历分页后的文章 -->
{% for post in paginator.posts %}
  <h1><a href="{{ post.url }}">{{ post.title }}</a></h1>
  <p class="author">
    <span class="date">{{ post.date }}</span>
  </p>
{% endfor %}
```

```

<div class="content">
    {{ post.content }}
</div>
{% endfor %}

<!-- 分页链接 -->
<div class="pagination">
    {% if paginator.previous_page %}
        <a href="/page{{ paginator.previous_page }}" class="previous">Previous</a>
    {% else %}
        <span class="previous">Previous</span>
    {% endif %}
    <span class="page_number">Page: {{ paginator.page }} of {{ paginator.total_pages }}</span>
    {% if paginator.next_page %}
        <a href="/page{{ paginator.next_page }}" class="next">Next</a>
    {% else %}
        <span class="next">Next</span>
    {% endif %}
</div>

```

注意首尾页

Jekyll 没有生成文件夹 ‘page1’，所以上边的代码有 bug，下边的代码解决了这个问题。下边的 HTML 片段是第一页，他除自己外，为每个页面生成了链接。

```

{% if paginator.total_pages > 1 %}
<div class="pagination">
    {% if paginator.previous_page %}
        <a href="{{ paginator.previous_page_path | prepend: site.baseurl | replace: '/', '/' }}">&laquo; Prev</a>
    {% else %}
        <span>&laquo; Prev</span>
    {% endif %}

    {% for page in (1..paginator.total_pages) %}
        {% if page == paginator.page %}
            <em>{{ page }}</em>
        {% elsif page == 1 %}
            <a href="{{ '/index.html' | prepend: site.baseurl | replace: '/', '/' }}">{{ page }}</a>
        {% else %}
            <a href="{{ site.paginate_path | prepend: site.baseurl | replace: '/', '/' | replace: ':num', page }}">{{ page }}</a>
        {% endif %}
    {% endfor %}

    {% if paginator.next_page %}
        <a href="{{ paginator.next_page_path | prepend: site.baseurl | replace: '/', '/' }}">Next &raquo;</a>
    {% else %}

```

```
    <span>Next &raquo;</span>
  {% endif %}
</div>
{% endif %}
```

插件

Jekyll 支持插件功能，你可以很容易的加入自己的代码。

在 GitHub Pages 使用插件

[GitHub Pages](#) 是由 Jekyll 提供技术支持的，考虑到安全因素，所有的 Pages 通过 `--safe` 选项禁用了插件功能，因此如果你的网站部署在 Github Pages，那么你的插件不会工作。

不过仍然有办法发布到 GitHub Pages，你只需在本地做一些转换，并把生成好的文件上传到 Github 替代 Jekyll 就可以了。

安装插件

有两种安装插件的方式：

1. 在网站根下目录建立 `_plugins` 文件夹，插件放在这里即可。Jekyll 运行之前，会加载此目录下所有以 `*.rb` 结尾的文件。
2. 在 `_config.yml` 文件中，添加一个以 `gems` 作为 key 的数组，数组中存放插件的 gem 名称。例如：

```
gems: [jekyll-test-plugin, jekyll-jsonify, jekyll-assets]
# This will require each of these gems automatically.
```

`_plugins` and `gems` 可以同时使用。

You may use both of the aforementioned plugin options simultaneously in the same site if you so choose. Use of one does not restrict the use of the other

通常，插件最终会被放在以下的目录中：

1. Generators
2. Converters
3. Tags

生成器

You can create a generator when you need Jekyll to create additional content based on your own rules.

A generator is a subclass of `Jekyll::Generator` that defines a `generate` method, which receives an instance of `Jekyll::Site`.

Generation is triggered for its side-effects, the return value of `generate` is ignored. Jekyll does not assume any particular side-effect to happen, it just runs the method.

Generators run after Jekyll has made an inventory of the existing content, and before the site is generated. Pages with YAML front-matters are stored as instances of `Jekyll::Page` and are available via `site.pages`. Static files become instances of `Jekyll::StaticFile` and are available via `site.static_files`. See `Jekyll::Site` for more details.

For instance, a generator can inject values computed at build time for template variables. In the following example the template `reading.html` has two variables `ongoing` and `done` that we fill in the generator:

```
module Reading
  class Generator < Jekyll::Generator
    def generate(site)
      ongoing, done = Book.all.partition(&:ongoing?)

      reading = site.pages.detect {|page| page.name == 'reading.html'}
      reading.data['ongoing'] = ongoing
      reading.data['done'] = done
    end
  end
end
```

This is a more complex generator that generates new pages:

```
module Jekyll
  class CategoryPage < Page
    def initialize(site, base, dir, category)
      @site = site
      @base = base
      @dir = dir
      @name = 'index.html'
    end
  end
end
```

```

    self.process(@name)
    self.read_yaml(File.join(base, '_layouts'), 'category_index.html')
    self.data['category'] = category

    category_title_prefix = site.config['category_title_prefix'] || 'Category: '
    self.data['title'] = "#{category_title_prefix}#{category}"
  end
end

class CategoryPageGenerator < Generator
  safe true

  def generate(site)
    if site.layouts.key? 'category_index'
      dir = site.config['category_dir'] || 'categories'
      site.categories.keys.each do |category|
        site.pages << CategoryPage.new(site, site.source, File.join(dir, category), category)
      end
    end
  end
end
end
end
end

```

本例中，生成器在 `categories` 下生成了一系列文件。并使用布局 `category_index.html` 列出所有的文章。

生成器只需要实现一个方法：

METHOD	DESCRIPTION
generate	Generates content as a side-effect.

转换器

如果想使用一个新的标记语言，可以用你自己的转换器实现，Markdown 和 Textile 就是这样实现的。

记住你的 YAML 头信息

Jekyll 只会转换带有 YAML 头信息的文件，即使你使用了插件也不行。

下边的例子实现了一个转换器，他会用 `UppcaseConverter` 来转换所有以 `.upcase` 结尾的文件。

```

module Jekyll
  class UppcaseConverter < Converter
    safe true
  end
end

```

```

priority :low

def matches(ext)
  ext =~ /\.\upcase$/i
end

def output_ext(ext)
  ".html"
end

def convert(content)
  content.upcase
end
end
end

```

转换器需要最少实现以下 3 个方法：

方法	描述
matches	检查文件后缀名是否是所要的，传入的参数是文件的后缀名（包括点号），接受的返回值是 true 或 false 。
output_ext	生成文件的后缀名（包括点号），通常是 ".html" 。
convert	转换逻辑，传入原始文件内容（不包含YAML头信息），返回值需要是 String 。

在上边的例子中，`UppcaseConverter#matches` 检查文件后缀名是不是 `.upcase`；`UppcaseConverter#convert` 会处理检查成功文件的内容，即将所有的字符串变成大写；最终，保存的结果 将以作为后缀名 `.html`。

标记

如果你想使用 liquid 标记，你可以这样做。Jekyll 官方的例子有 `highlight` 和 `include` 等标记。下边的例子中，自定义了一个 liquid 标记，用来输出当前时间：

```

module Jekyll
  class RenderTimeTag < Liquid::Tag

    def initialize(tag_name, text, tokens)
      super
      @text = text
    end
  end
end

```

```

end

def render(context)
  "#{@text} #{Time.now}"
end

end

end

Liquid::Template.register_tag('render_time', Jekyll::RenderTimeTag)

```

liquid 标记最少需要实现如下方法：

方法	描述
render	输出标记的内容。

你必须同时用 Liquid 模板引擎注册自定义标记，比如：

```
Liquid::Template.register_tag('render_time', Jekyll::RenderTimeTag)
```

对于上边的例子，你可以把如下标记放在页面的任何位置：

```
<p>{% render_time page rendered at: %}</p>
```

我们在页面上会得到如下内容：

```
<p>page rendered at: Tue Jun 22 23:38:47 -0500 2010</p>
```

Liquid 过滤器

你可以像上边那样在 Liquid 模板中加入自己的过滤器。过滤器会把自己的方法暴露给 liquid。所有的方法都必须至少接收一个参数，用来传输入内容；返回值是过滤的结果。

```

module Jekyll
  module AssetFilter
    def asset_url(input)
      "http://www.example.com/#{input}?#{Time.now.to_i}"
    end
  end
end

Liquid::Template.register_filter(Jekyll::AssetFilter)

```

提示™：用 Liquid 访问 site 对象

Jekyll 允许通过 Liquid 的 `context.registers` 特性来访问 `site` 对象。比如可以用 `context.registers.config` 访问配置文件 `_config.yml`。

Flags

当写插件时，有两个标记需要注意：

标记	描述
safe	告诉 Jekyll 此插件是否可以安全的执行任意代码。GitHub Pages 用他来决定那个插件可以使用，哪些不可以使用。如果你的插件不允许执行任意代码，把它设为 true 即可。GitHub Pages 仍然不会加载你的插件，但是如果你把他夹杂到核心中，最后保证此值设置的正确！
priority	此标记决定加载插件的顺序。可以是这些值： :lowest ， :low ， :normal ， :high ，还有 :highest 。优先级高的先执行，优先级低的后执行。

已上边例子的插件为例，应该这样设置这两个标记：

```
module Jekyll
  class UppcaseConverter < Converter
    safe true
    priority :low
    ...
  end
end
```

可用的插件

下边的插件，你可以按需所取：

生成器

- [ArchiveGenerator](#) by Ilkka Laukkanen: 用[这里的方法](#)生成档案。
- [LESS.js Generator](#) by Andy Fowler: 生成的时候产生 LESS.js 文件。
- [Version Reporter](#) by Blake Smith: 创建包含 Jekyll 版本的文件 `version.html`。
- [Sitemap.xml Generator](#) by Michael Levin: 遍历所有的页面和文章，生成 `sitemap.xml`。

- [Full-text search by Pascal Widdershoven](#): 全文搜索。
- [AliasGenerator by Thomas Mango](#): 根据YAML头信息中的 `alias` 生成跳转页面。
- [Pageless Redirect Generator by Nick Quinlan](#): 根据Jekyll跟路径做出跳转，支持分布式。
- [Projectlist by Frederic Hemberger](#): 一个文件夹生成一个页面
- [RssGenerator by Assaf Gelber](#): 自动生成 RSS 2.0 。
- [Monthly archive generator by Shigeya Suzuki](#): Generator and template which renders monthly archive like MovableType style, based on the work by Ilkka Laukkanen and others above.
- [Category archive generator by Shigeya Suzuki](#): Generator and template which renders category archive like MovableType style, based on Monthly archive generator.
- [Emoji for Jekyll](#): Seamlessly enable emoji for all posts and pages.
- [Compass integration for Jekyll](#): Easily integrate Compass and Sass with your Jekyll website.

转换器

- [Jade plugin by John Papandriopoulos](#): Jade 转换器。
- [HAML plugin by Sam Z](#): HAML转换器。
- [HAML-Sass Converter by Adam Pearson](#): HAML-Sass 转换器。 [Fork](#) by Sam X.
- [Sass SCSS Converter by Mark Wolfe](#): 在Sam X 的基础上，一个兼容 CSS 的 Sass 转换器。
- [LESS Converter by Jason Graham](#): 将 LESS 转换为 CSS。
- [LESS Converter by Josh Brown](#): 简单的 LESS 转换器。
- [Uppcase Converter by Blake Smith](#): 一个例子 。
- [CoffeeScript Converter by phaser](#): [CoffeeScript](#) 转换到 JavaScript 。
- [Markdown References by Olov Lassus](#): 记录所有的超链接到 `_references.md` 文件。
- [Stylus Converter](#): 将 `.styl` 转换为 `.css` 。
- [ReStructuredText Converter](#): 用 Pygments 语法将 ReST 文档转换为 HTML 。
- [Jekyll-pandoc-plugin](#): 用 pandoc 转换 markdown 。
- [Jekyll-pandoc-multiple-formats](#)by [eds1](#):用 pandoc 生成网站，支持多种格式，并支持 pandoc 的后缀名。
- [ReStructuredText Converter](#): 又一个用 Pygments 语法将 ReST 文档转换为 HTML 。
- [Transform Layouts](#): 允许使用 HAML 布局（需要 HAML 转换器的配合）

- [Org-mode Converter](#): Org-mode converter for Jekyll.

过滤器

- [Truncate HTML](#) by [Matt Hall](#): 为保持 markup 结构, 删除 HTML 标签。
- [Domain Name Filter](#) by [Lawrence Woodman](#): 过滤出域名。
- [Summarize Filter](#) by [Mathieu Arnold](#): 去掉 `<div id="extended">` 后边的内容。
- [URL encoding](#) by [James An](#): 为地址编码, 如 `' ' #=> '%20'`。
- [JSON Filter](#) by [joelverhagen](#): 转换为 JSON 格式。
- [i18n_filter](#): 实现了 I18n 国际化的 Liquid 过滤器。
- [Smilify](#) by [SaswatPadhi](#): 将表情符号转换为表情图片 ([例子](#))。
- [Read in X Minutes](#) by [zachleat](#): 估计读完文章需要的时间。
- [Jekyll-timeago](#): 把时间转换为 time ago 格式。
- [pluralize](#): 根据单词前边的数字按需转换成复数形式。
- [reading_time](#): 统计字数, 并估计需要读的时间 (已忽略HTML标签)。
- [Table of Content Generator](#): 生成包含表格 (TOC) 的 HTML 代码, 支持自定义。
- [jekyll-humanize](#): This is a port of the Django app humanize which adds a “human touch” to data. Each method represents a Fluid type filter that can be used in your Jekyll site templates. Given that Jekyll produces static sites, some of the original methods do not make logical sense to port (e.g. naturaltime).

标签

- [Asset Path Tag](#) by [Sam Rayner](#): Allows organisation of assets into subdirectories by outputting a path for a given file relative to the current post or page.
- [Delicious Plugin](#) by [Christian Hellsten](#): 从 delicious.com 获取书签并展示。
- [Ultraviolet Plugin](#) by [Steve Alex](#): [Ultraviolet](#) 插件。
- [Tag Cloud Plugin](#) by [Ilkka Laukkanen](#): 生成云状的标签列表。
- [GIT Tag](#) by [Alexandre Girard](#): 添加 Git activity。
- [MathJax Liquid Tags](#) by [Jessy Cowan-Sharp](#): 用合适的 MathJax 标签替代相应的数学公式或方程式。

- [Non-JS Gist Tag by Brandon Tilley](#): 嵌入 Gists , 显示给禁用 JavaScript 的用户。
- [Render Time Tag by Blake Smith](#): 显示页面的创建时间。
- [Status.net/OSTatus Tag by phaer](#): 显示 status.net/ostatus 的通知。
- [Raw Tag by phaer](#): 阻止转换 raw 标签的内容。
- [Embed.ly client by Robert Böhnke](#): Embed.ly 的实现。
- [Logarithmic Tag Cloud](#): 支持对数。
- [oEmbed Tag by Tammo van Lessen](#): 通过 oEmbed 支持内嵌第三方代码 (例如 YouTube, Flickr, Slideshare) 。
- [FlickrSetTag by Thomas Mango](#): 将 Flickr 的图片生成画廊。
- [Tweet Tag by Scott W. Bradley](#): [Tweets](#) 支持 。
- [Jekyll-contentblocks](#): 支持使用 Rails 风格的 content_for 标签。
- [Generate YouTube Embed by joelverhagen](#): 支持以 YouTube ID 嵌入 YouTube 视频, 宽高可配置。
- [Jekyll-beastiepress](#): 可轻松连接到 FreeBSD 官网。
- [Jsonball](#): 读取 json 文件并生成地图。
- [Bibjekyll](#): Render BibTeX-formatted bibliographies/citations included in posts and pages using bibtex2html.
- [Jekyll-citation](#): 生成 BibTeX 格式 (Holy shit! 还有多少要翻译) 。
- [Jekyll Dribbble Set Tag](#): 生成 Dribbble 画廊。
- [Debbugs](#): 可以轻松链接到 Debian BTS 。
- [Refheap_tag](#): 支持 [refheap](#).
- [Jekyll-devonly_tag](#): 仅在开发环境使用的情况下可以考虑一下。
- [JekyllGalleryTag](#) by [redwallhp](#): 生成缩略图, 并展示。
- [Youku and Tudou Embed](#): 支持内嵌 Youku 和 Tudou 视频。
- [Jekyll-swfobject](#): 通过 [SWFObject](#) 支持内嵌 flash 。
- [Jekyll Picture Tag](#): 相应式的图片, 推荐 , 改编自 Scott Jehl 的 [Picturefill](#).
- [Jekyll Image Tag](#): 更好的图片处理插件, 预先保存, 生成调整大小后的图片, 并且加好 `classes` 和 `alt` 等属性。
- [Dittaa Tag](#) by [matze](#): 将 ditaa 格式的 ASCII 图片转换成 PNG。
- [Good Include](#) by [Anatol Broder](#): 去掉文件末尾的空行空格。

- [Jekyll Suggested Tweet](#) by [David Ensinger](#): 通过 Twitter 的 API 支持内嵌感兴趣的 tweets。
- [Jekyll Date Chart](#) by [GSI](#): Block that renders date line charts based on textile-formatted tables.
- [Jekyll Image Encode](#) by [GSI](#): Tag that renders base64 codes of images fetched from the web.
- [Jekyll Quick Man](#) by [GSI](#): Tag that renders pretty links to man page sources on the internet.
- [jekyll-font-awesome](#): Quickly and easily add Font Awesome icons to your posts.

集合

- [Jekyll Plugins by Recursive Design](#): 生成 readme 说明文档，列表页以及网站地图。
- [Company website and blog plugins](#) by Flatterline, a [Ruby on Rails development company](#): Portfolio/project 的生成器，team/individual 的生成器，等等。
- [Jekyll plugins by Aucor](#): 移除不需要的空格空行，并根据 weight 属性给页面排序。

其他

- [Pygments Cache Path](#) by [Raimonds Simanovskis](#): 缓存高亮代码核心模块。
- [Draft/Publish Plugin](#) by [Michael Ivey](#): 保存到草稿。
- [Growl Notification Generator](#) by [Tate Johnson](#): Jekyll 通知发送到 Growl 。
- [Growl Notification Hook](#) by [Tate Johnson](#): 同上，推荐指数更高一点，但是需要他的 “hook” 。
- [Related Posts](#) by [Lawrence Woodman](#): 重新实现关联，覆盖 site.related_posts 。
- [Tiered Archives](#) by [Eli Naeher](#): 创建 tiered 模板变量，允许按年月分组。
- [Jekyll-localization](#): 支持本地化。
- [Jekyll-rendering](#): 一个渲染引擎。
- [Jekyll-pagination](#): 支持分页。
- [Jekyll-tagging](#): 生成云状标签。
- [Jekyll-scholar](#): 为学者定制。
- [Jekyll-asset_bundler](#): 将 JavaScript 和 CSS 最小化。
- [Jekyll-assets](#) by [ixti](#): Rails 风格的 assets pipeline（支持的资源 CoffeeScript, Sass, LESS 等等；设置依赖；最小化压缩；JST 模板；缓存处理等等）。

- [File compressor](#) by [mytharcher](#): 压缩 HTML 和 JavaScript 。
- [Jekyll-minibundle](#): 根据你选择的最小化工具绑定资源和处理缓存。
- [Singlepage-jekyll](#) by [JCB-K](#): 转换为单页网站。
- [generator-jekyllrb](#): [Yeoman](#) 的包装, 一个工具集, 还有工作流, 用来创建现代化的网站。
- [grunt-jekyll](#): [Grunt](#) 插件。
- [jekyll-postfiles](#): 添加目录 `_postfiles` 和标签 `{{ postfile }}` 以保证所有的指向正确。

期待你的作品

如果你有一个 Jekyll 插件并且愿意加到这个列表中来, 可以[阅读此须知](#), 并参照着来做。

附加功能

Jekyll 提供了诸多（可任选）的附加功能，你可以依据你使用 Jekyll 的需求来选择安装它们。

LaTeX 支持

Maruku 自带了将 LaTeX 渲染成 PNG 的功能可供选择，此功能使用 `blahtex`（版本 0.6），必须和 `dvips` 一起被置于你的 `$PATH` 中。如果你需要 Maruku 不调用默认位置的 `dvips`，请查看 [Remi 的 Maruku fork](#)。

可选的 Markdown 处理器

虽然 Jekyll 默认使用 Maruku 来转换 Markdown，你还可以使用以下三个预定义的 markdown 解析器中的任意一个，或者你也可以自己实现一个。

RDiscount

如果你更喜欢使用 [RDiscount](#) 来替代 [Maruku](#) 解析 Markdown，你只需确认已将其安装：

```
$ [sudo] gem install rdiscount
```

然后在你的 `_config.yml` 文件内选择 RDiscount 作为 Markdown 引擎，使 Jekyll 可以读取该选项来运行。

```
# _config.yml 中

markdown: rdiscount
```

Kramdown

你还可以选择 Kramdown 来替代 Maruku 解析 Markdown，你只需确认 Kramdown 已被安装：

```
$ [sudo] gem install kramdown
```

然后在你的 `_config.yml` 文件内选择 Kramdown 作为 Markdown 引擎。

```
# _config.yml 中

markdown: kramdown
```

Kramdown 提供了各种选项来自定义其 HTML 的输出。[配置](#)页面列出了 Jekyll 所使用的默认选项。一份完整的选项列表也可见于 [Kramdown 网站](#)。

自定义

如果你对以上四个内置的 markdown 解析器都不满意，没关系，你还可以自己写个插件：

```
require 'jekyll'
require 'some_renderer'

class Jekyll::Converters::Markdown::MyCustomParser
  def initialize(config)
    @site_config = config
  end

  def convert(content)
    # (this _must_ return the resulting String after the rendering)
    SomeRenderer.new(@site_config).to_html(content)
  end
end
```

一旦你的解析器完成了，就可以在 `_config.yml` 文件中告诉 Jekyll 使用你自己的 markdown 解析器了：

```
markdown: MyCustomParser
```

(注意，这是 大小写敏感的，并且，只需指定 `Jekyll::Converters::Markdown` 后面的部分就可以。) 仅此而已！



4

部署



GitHub Pages

[Github Pages](#) 是面向用户、组织和项目开放的公共静态页面搭建托管服务，站点可以被免费托管在 Github 上，你可以选择使用 Github Pages 默认提供的域名 [github.io](#) 或者自定义域名来发布站点。Github Pages 支持自动利用 Jekyll 生成站点，也同样支持纯 HTML 文档，将你的 Jekyll 站点托管在 Github Pages 上是一个不错的选择。

将 Jekyll 部署到 Github Pages 上

Github Pages 依靠 Github 上项目的某些特定分支来工作。Github Pages 分为两种基本类型：用户/组织的站点和项目的站点。搭建这两种类型站点的方法除了一小些细节之外基本一致。

用户和组织的站点

用户和组织的站点被放置在一个特殊的专用仓库中，在该仓库中只存在 Github Pages 的相关文件。这个仓库应该根据用户/组织的名称来命名，例如：[@mojombo 的用户站点仓库](#) 应该被命名为 `mojombo.github.io`。

仓库中 `master` 分支里的文件将会被用来生成 Github Pages 站点，所以请确保你的文件储存在该分支上。

自定义域名不影响仓库命名

Github Pages 初始被设置部署在 `username.github.io` 子域名上，这就是为什么即使你使用自定义域名仓库还需要这样命名。

项目的站点

不同于用户和组织的站点，项目的站点文件存放在项目本身仓库的 `gh-pages` 分支中。该分支下的文件将会被 Jekyll 处理，生成的站点会被部署到你的用户站点的子目录上，例如 `username.github.io/project`（除非指定了一个自定义的域名）。

Jekyll 项目本身就是一个很好的例子，Jekyll 项目的代码存放在 [master 分支](#)，而 Jekyll 的项目站点（就是你现在看见的网页）包含在同一仓库的 [gh-pages 分支](#) 中。

项目站点的网址结构

你最好在将 Jekyll 站点提交到 `gh-pages` 之前先预览一下。因为 Github 上项目站点的子目录结构会使站点的网址结构变得复杂。这里有一些处理 Github Pages 子目录结构（`username.github.io/project-name/`）的方法使你本地浏览的站点和部署在 Github Pages 上的站点一致，方便你的维护。

1. 在 `_config.yml` 中，设置 `baseurl` 选项为 `/project-name` - 注意必须存在头部的斜杠以及不能有尾部的斜杠。
2. JS 或者 CSS 文件的引用格式应该如下： `{{ site.baseurl }}/path/to/css.css` - 注意斜杠之后必须紧随变量（在 “Path” 之后）。
3. 创建固定链接和内部链接的格式应该如下： `{{ site.baseurl }}{{ post.url }}` - 注意两个变量之间不存在斜杠。
4. 最后，如果你想在提交/部署之前浏览的话，请使用 `jekyll serve --baseurl ''` 命令，请确定在 `--baseurl` 的选项之后存在空串，这样的话你就可以在 `localhost:4000` 看到你的站点（站点根地址不存在 `/project-name`）。用这种方法你就可以在本地从根地址预览站点，而在 Github 上以 `gh-pages` 分支生成站点的时候能以 `/project-name` 为根地址并且正确地显示。

GitHub Pages 的文档，帮助和支持

想获得关于 Github Pages 的更多信息和解决方案，你应该访问 [GitHub's Pages 帮助部分](#)。如果无法找到解决方案，你可以联系 [GitHub 支持](#)。

部署方法

Jekyll 生成的网站是静态的，因此有很多种部署方法。下面列出了一些常见的部署方法。

网站托管服务商 (FTP)

传统的网络托管服务商允许你使用 FTP 上传文件到他们的服务器。想通过 FTP 上传一个 Jekyll 站点，只需要运行 `jekyll` 命令然后复制生成的 `_site` 目录到你的托管账号根目录。多数托管服务商的跟目录会是 `httpdocs` 或 `public_html` 目录。

使用 Glynn 进行 FTP 上传

有一个叫 [Glynn](#) 的项目，可以帮助你简单的生成 Jekyll 站点并通过 FTP 发送到你的主机。

自己的网络服务器

如果你能够直接连接到部署的网络服务器，你可能有其它的方法传输文件（如 `scp`，或者直接操作文件系统），而其它过程都一样。要记住保证生成的 `_site` 目录放到网络服务器正确的根目录下。

自动化部署

也有一些自动化部署 Jekyll 站点的方法。下面列出了几种，如果你还有其它的，欢迎[贡献](#)，这样其它人就也能知道它了。

Git post-update 钩子

如果你使用 [Git](#) 管理你的 jekyll 站点，自动化部署非常简单，只需要给你的 Git 仓库设置一个 post-update 钩子，[就像这样](#)

Git post-receive 钩子

要让一个远程服务器在你每次用 Git 推送修改时进行部署，可以创建一个拥有所有要部署机器公钥的账号，然后设置 post-receive 钩子，其余的跟上面方法一样。

```
laptop$ ssh deployer@example.com
server$ mkdir myrepo.git
server$ cd myrepo.git
server$ git --bare init
server$ cp hooks/post-receive.sample hooks/post-receive
server$ mkdir /var/www/myrepo
```

接着，添加下面的代码到 `hooks/post-receive`，并保证服务器上已安装 Jekyll：

```
GIT_REPO=$HOME/myrepo.git
TMP_GIT_CLONE=$HOME/tmp/myrepo
PUBLIC_WWW=/var/www/myrepo

git clone $GIT_REPO $TMP_GIT_CLONE
jekyll build -s $TMP_GIT_CLONE -d $PUBLIC_WWW
rm -Rf $TMP_GIT_CLONE
exit
```

最后，在任意可以通过此钩子部署的用户机器上运行下面的命令：

```
laptops$ git remote add deploy deployer@example.com:~/myrepo.git
```

剩下的就是告诉 `nginx` 或 `Apache` 监听 `/var/www/myrepo` 目录，然后运行下面的命令：

```
laptops$ git push deploy master
```

Rake

另一个部署 Jekyll 站点的方法是使用 `Rake`，`HighLine`，和 `Net::SSH`。一个比较复杂的使用 `Rake` 部署多个分支的例子可以参考 `Git Ready`。

rsync

假如你已经生成了 `_site` 目录，就可以使用一个像 部署脚本 这样的 shell 脚本 `tasks/deploy` `rsync` 到服务器了。当然需要修改你的站点相应的值。甚至还有一个 `TextMate` 匹配命令 可以帮你在 `Textmate` 中运行这个脚本。 this script from within Textmate.

Rack-Jekyll

`Rack-Jekyll` 是一个部署站点到任意 `Rack` 服务的简单方法，如 `Amazon EC2`，`Slicehost`，`Heroku` 等。它也可以 `shotgun`，`rackup`，`mongrel`，`unicorn`，and `others` 一起运行。

可以阅读 [这篇文章](#) 了解如何使用 Rack-Jekyll 部署到 Heroku 。

Jekyll-Admin for Rails

如果想在 Rails 中维护 Jekyll 站点，Jekyll-Admin 包含了实现此功能直接可用的代码。详细可查看 Jekyll-Admin 的 README 。

Amazon S3

如果要在 Amazon S3 上托管你的站点，可以使用 s3_website 。它会推送你的站点到 Amazon S3 上，Amazon S3 跟任意网络服务器一样，却能够动态扩容到几乎无限流量。这种方式适用小流量博客站点，因为你只需要为你使用的流量付费。

OpenShift

如果你希望将网站部署到 OpenShift gear 上面，这里有一份教程 一份教程。

ProTip™: 使用 GitHub Pages 零麻烦托管

GitHub Pages 内部由 Jekyll 驱动，所以如果你想找个零麻烦、零花费解决方案，Github Pages 是托管 Jekyll 驱动站点的首选。



T



5

杂项



常见问题

如果你在安装或者使用 Jekyll 的过程中遇到了问题，这里有一些建议也许可以帮助你。如果你所遇到的问题没有包含在下面，[请提交一个 issue](#)，这样 Jekyll 团队才能让更多的人有更好的使用体验。

安装问题

如果你在安装 gem 的过程中遇到问题，可能你需要安装为 ruby 1.9.1 的拓展模块编译所需要的头文件，在 Ubuntu 或 Debian 系统中安装可以通过运行：

```
sudo apt-get install ruby1.9.1-dev
```

在 RHEL, CentOS 和 Fedora 系统中安装你可以通过运行：

```
sudo yum install ruby-devel
```

在 [NearlyFreeSpeech](#) 中你需要在运行命令的时候添加下面的环境变量：

```
RB_USER_INSTALL=true gem install jekyll
```

在 OSX 系统中你可能需要升级 RubyGems：

```
sudo gem update --system
```

如果你还是遇到问题，你可能需要[使用 XCode 来安装命令行工具](#)

```
sudo gem install jekyll
```

在 Gentoo 上安装 RubyGems：

```
sudo emerge -av dev-ruby/rubygems
```

在 Windows 下你可能需要安装 [RubyInstaller DevKit](#)。

运行 Jekyll 时的问题

在 Debian 或者 Ubuntu 系统中，你可能需要在 path 里添加 `/var/lib/gems/1.8/bin/` 来使 `jekyll` 命令可以在终端中执行。

Base-URL 问题

如果你正在这样使用 `base-url` 选项：

```
jeekyll serve --baseurl '/blog'
```

… 那么你需要在访问网页的时候使用：

```
http://localhost:4000/blog/index.html
```

这样访问会出现错误：

```
http://localhost:4000/blog
```

配置问题

冲突的配置设置的优先顺序如下：

1. 命令行标志
2. 配置文件设置
3. 默认配置

也就是说，默认配置会被 `_config.yml` 中指定的选项所覆盖，而在命令行中指定的参数配置会覆盖其它地方的配置。

Markup 问题

Jekyll 所使用的不同的 Markup 引擎可能会有一些问题。下面的文件可能会帮助你如果你遇到类似的问题。

Maruku

如果你的链接中有一些需要避免的词，你需要这样写：

```
![Alt text](http://yuml.me/diagram/class/[Project]->[Task])
```

如果你有一个空的标签，比如 `<script src="js.js"></script>`，Maruku 会将它转换成 `<script src="js.js">/>`。这将会在火狐或者其它浏览器中出现问题，而且[在 XHTML 中不推荐使用](#)。一个简单的避免方法就是在起始标签和结束标签之间放一个空格。

RedCloth

4.1.1 和更高的版本将不支持 `notextile` 标签。[这是一个已知的 bug](#)可能有希望在 4.2 版本中得到修复。你可以继续使用 4.1.9 版本，但是测试套件需要安装 4.1.0 版本。如果使用一个不支持 `notextile` 标签的版本，你可能需要注意 Pygments 的语法高亮格式会不正确，还有其它一些可能的问题。如果你遇到这个问题你只需要安装 4.1.0 版本。

Liquid

最新的 2.0 版本似乎打破了 `{{` 在模板中的使用，不再类似以前的版本，在 2.0 版本使用 `{{` 会出现以下问题：

```
'{' was not properly terminated with regexp: /\}\}/ (Liquid::SyntaxError)
```

摘要

从 V1.0.0 版本开始，Jekyll 已经可以自动生成文章摘要。一直到 v1.1.0 版本，Jekyll 仍使用 Liquid 来传递摘要，这将会在引用不存在或标记没有被关闭时造成奇怪的问题。如果你遇到了这些问题，你可以尝试将在

```
_config.yml
```

 中设置 `excerpt_separator: ""` 或设置成不敏感的字符。

使用 Jekyll 的站点

看看别人想出了什么设计与特性是十分有趣的。下面列出了一些可供学习的博客。

- [Tom Preston-Werner](#) (源代码)
- [Nick Quaranto](#) (源代码)
- [Roger Chapman](#) (源代码)
- [GitHub Official Teaching Materials](#) (源代码)
- [Rasmus Andersson](#) (源代码)
- [Scott Chacon](#) (源代码)

如果你想查看更多的示例，你可以在 `Jekyll wiki` 的 “Sites” 页面 找到另一些站点和源代码的列表。

相关资源

随着 Jekyll 被更广泛的使用，一系列的教程、框架、扩展、示例和其他各种有用的资源也相继出现。下面列出了一些最流行的 Jekyll 资源合辑的链接。

Jekyll 提示和技巧，以及示例

- [与 GitHub Pages 集成的技巧](#) 重用代码并保持文档同步至最新的示例。
- [使用 Simple Form 来集成一个简单的联系表单](#)
- [JekyllBootstrap.com](#) 提供了详细的解释，例子和辅助代码让大家上手 Jekyll 变得更容易。

教程

集成 Jekyll 和 Git

- [使用 Git、Emacs 和 Jekyll 写博客](#)

其他技巧

- [集成 Twitter 和 Jekyll](#)

“在将 Justkez.com 转换成基于 Jekyll 的网站后，我仔细思索我该如何添加我最近的 Twitter 条目至网站的主页上。在 WordPress 的世界里，这可以通过一个插件来完成，其结果可能会或不会阻塞主页加载，也可能提供了缓存功能，但同时肯定也会有各种额外支出。… 但 Jekyll 不会这样。”

- [‘我的 Jekyll Fork’，Mike West](#)

“Jekyll 是一个构架良好的复古产物，让时光倒退回 WordPress 之前，那时人还是人，而 HTML 却是静态的。我钟意它的理念，故提供了少量对其内核的改进。这里，我将展现出我的 fork 的特色，并希望能激发一些与众不同的启示。”

- [‘关于本网站’，Carter Allen](#)

“Jekyll 拥有了我认为一个博客引擎所应该具有的一切。千真万确。虽然它还不完美，但这正是他的魅力之所在，如果某处出了问题，我知道它应该如何工作，并知道如何去修复它。它可以只在你的电脑里运行，给你和你

的浏览器之间添加了一个“运行”的步骤。我完全只使用了 TextMate 和基本的 HTML5 和 CSS3，然后在最后给标记语言添加了几个变量。突然之间，我的网站便已大功告成。”

- [在 Jekyll 里生成一个标签云 \(Tag Cloud\)](#)

一个通过 Jekyll 来实现标签云和单独标签列表页面的教程。

- [Jekyll 扩展 == 痛苦](#)

一种可以[扩展 Jekyll](#)，而不去 forking 和修改 Jekyll gem 源代码的方法，以及一些可供复用和分享的[便携 Jekyll 扩展](#)。

- [在 Jekyll 里使用你的 Rails 布局](#)

升级

是时候该升级你的 Jekyll 了，升级之前为你介绍一下版本 1.0 的更新内容。

首先，我们需要获取 Jekyll 的最新版本：

```
$ gem update jekyll
```

提示

想要迅速建立起一个 Jekyll 站点并跑起来吗？只需要输入命令 `jekyll new SITENAME`，该命令将创建一个包含基本功能的 Jekyll 站点。

Jekyll 命令

Jekyll 现在支持命令 `build` 和 `serve`，使用起来更加清晰。以前你或许会使用命令 `jekyll` 生成一个网站并用 `jekyll --server` 在本地浏览，现在可以用子命令 `jekyll build` 和 `jekyll serve` 代替。如果当一个文件改变时，你希望 Jekyll 自动做出相应的更新，只需要在命令的末尾加上 `--watch` 即可。

Watching 和 Serving

使用新的子命令，网站的操作方式也有一些变化。以前的做法是在网站配置文件中加上 `server: true`，现在用 `jekyll serve` 即可。同样的 `watch: true` 也是如此，在 `jekyll serve` 或 `jekyll build` 后边加上 `--watch` 即可。

绝对地址

在 Jekyll v1.0 中，我们引入了“绝对地址”。v1.1 之前，使用 `opt-in`。从 v1.1 开始，将使用 `opt-out`，这意味着 Jekyll 将使用绝对地址代替相对地址。

- 如果要使用绝对地址，需要在配置文件中加上 `relative_permalinks: false`。
- 如果要继续使用相对地址，在配置文件中加上 `relative_permalinks: true`。

在 v1.1 中，绝对地址将成为默认配置

从 Jekyll v1.1.0 开始，`relative_permalinks` 默认为 `false`，这意味着所有页面默认为绝对地址，该配置会一直保留到 v2.0。

草稿箱

Jekyll 现在支持草稿箱，并且可以很容易的在发布前预览。想要开始编写草稿，只需要在项目 中建立 `_drafts` 文件夹（和 `_posts` 在同一目录），然后新建一个 `markdown` 文件即可。想要预览你的草稿，只需要在命令 `jekyll serve` 后边加上 `--drafts`。

草稿没有日期

跟文章不同，草稿没有日期，因为还没有发布。只需用标题（比如 `my-draft-post.md`）做为文件名，而不是 `2013-07-01-my-draft-post.md`。

自定义配置文件

不仅可以通过在命令行末加标志，还能够使用一个 `Jekyll` 自定义配置文件。这样可以帮助区分不同的环境，或以编程方式覆盖用户指定的配置。只需要在命令 `jekyll` 后加上 `--config`，然后输入一个或多个配置文件的路径（以逗号隔开，不能有空格）。

所以，不再建议使用以下这些命令：

- `--no-server`
- `--no-auto`
- `--auto`（现在的 `--watch`）
- `--server`
- `--url=`
- `--maruku`，`--rdiscount`，和 `--redcarpet`
- `--pygments`
- `--permalink=`
- `--paginate`

显式指定配置文件

如果你使用了标志 `--config`，Jekyll 将忽略文件 `_config.yml`。想对个配置文件中组合使用吗？没问题，Jekyll 支持通过命令行指定多个配置文件。越往右，配置文件优先级越高。如果我运行 `jekyll serve --config _config.yml,_config-dev.yml`，并且他们包含同一个配置项，那么这个配置项的结果将是右边 `_config-dev.yml` 的值，而非左边 `_config.yml`。

新的配置选项

Jekyll 1.0 引进了几个新的配置选项。在升级之前，你应该检查一下在 pre-1.0 的配置文件中是否有这些，如果有，确保正确配置了：

- `excerpt_separator`
- `host`
- `include`
- `keep_files`
- `layouts`
- `show_drafts`
- `timezone`
- `vurl`

根路径

通常，你想要在不同的地方运行你的 Jekyll 站点，比如发布前在本地预览。Jekyll 1.0 中使用标志 `--baseurl` 即可。要使用这个特写，首先在网站的 `_config.yml` 中写入生产环境的 `baseurl`；然后，遍历一遍代码，对所有相对地址加上前缀 `{{ site.baseurl }}`。当你想在本地测试的时候，在 `jekyll serve` 后传入标志 `--baseurl` 并跟上本地地址即可（可能是 `/`）。

所有的地址包含斜杠

如果你按照上边的方法做了，记得所有的地址前有一个斜杠。因此，`site.baseurl = /` 和 `post.url = /2013/06/05/my-fun-post/` 最终形成的地址有两个斜杠开头。所以建议在 `baseurl` 不是 `/` 时使用 `site.baseurl`。



元信息



贡献

是不是有个点子想实现到 Jekyll 。太好了，请参照如下： Great! Please keep the following in mind:

- 如果你要在已有的特性上做一个小修补，只需要写一个简单的 test 就可以了。在当前测试中使用 [Shoulda](#) 和 [RR](#).
- 如果是一个新特性，请写一个新的 [Cucumber](#) 并在 适当的地方重用步骤。同样，你也可以大胆的修改你对本网站的拷贝，一旦被合并掉，就会展示到网站 [jekyllrb.com](#) 。
- 如果你改变了 Jekyll 的习惯，不要忘了及时更新文档。在 `site/docs` 里边。如果发现文档中缺失的信息， 赶快加上吧。伟大的文档早就伟大的项目！
- 当修改 Ruby 代码的时候，请遵照 [GitHub Ruby 编码规范](#)。
- 请尽可能的提交 小的 pull request 。修改内容看起来越简单，就越可能被合并到主分支。
- 当提交 pull request 时，要知道什么地方放什么东西。描述一下做了哪些修改，背后的动机以及 [完成了什么任务或有待完成的](#)都会加快复核。

不接受没有测试的代码

如果你要在已有的特性上做一个小修补，只需要写一个简单的 test 就可以了。

测试依赖

想要跑测试用例和编译 gem 的话，你需要安装 Jekyll 的依赖包。Jekyll 支持 Bundler ，所以只需要运行一下 `bundle` 就可以了。

```
$ bundle
```

在开始之前，跑一下测试代码以确信全部通过（确定一下你的环境配置好了）：

```
$ bundle exec rake test
$ bundle exec rake features
```

Workflow

这是最直接的途径： the most direct way to get your work merged into the project:

- Fork 本项目。
- 从你的fork下载到本地：

```
git clone git://github.com/<username>/jekyll.git
```

- 创建一个分支，包含要修改的内容：

```
git checkout -b my_awesome_feature
```

- 添加测试。
- 通过命令 `rake` 确定所有测试依然全部通过。
- 如果有必要，将你的提交合并到逻辑块里边，不能有错误。
- 可以推送本分支了：

```
git push origin my_awesome_feature
```

- 同 `mojombo/jekyll:master` 对比并创建一个 `pull request`，描述一下你改了些什么还有你为什么认为他们会合并你的代码。

更新文档

我们希望 Jekyll 的文档尽可能的优秀。我们已经开源了所有文档，欢迎提交修改。

你可以在[这里](#)找到 `jekyllrb.com` 的文档。

所有针对文档的 `pull requests` 都要放在 `master`。不允许提交到其他分支。

Github 上的 [Jekyll wiki](#)可以自由更新，不需要 `pull request`，任何人都可以修改。

陷阱

- 如果你想修改 `gem` 版本，请放在一个独立的提交里边。如此，维护人员方便管理一些。
- 尽量让你分支中的代码是最新的。
- 不要在你的 GitHub issue 用 `[fix]`，`[feature]` 等标记。维护人员会积极的阅读 issues，一旦碰到他们会主动标记。

帮助我们做的更好

Both 不管使用还是为 Jekyll 贡献代码，都应该是有兴趣的、简单的、轻松的，所以如果你发现有什么不适，请在 Github 上提交一个 issue。

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台



更多信息请访问 

<http://wiki.jikexueyuan.com/project/jekyll/>