

# Jekyll 小书

安道





---

# Jekyll 小书

安道 著

---

rev 3.2.1.p1, 2016-08-04T12:00:00+08:00

---



---

# 目录

第 1 章 简介 .....	1
1.1 Jekyll 是什么	1
1.2 静态网站的好处	2
1.3 我该不该使用静态网站生成工具	3
1.4 预备知识	4
1.5 排版约定	4
1.6 随书源码	4
1.7 反馈	5
1.8 致谢	5
1.9 版权	5
第 2 章 安装 Jekyll .....	7
2.1 安装 Ruby	7
2.2 安装 Jekyll	8
第 3 章 创建第一个网站 .....	9
3.1 命令行界面	9
3.2 新建网站	10
3.3 网站的文件结构	11
3.4 查看网站	13
3.5 设置网站	14
第 4 章 文章 .....	17
4.1 文章文件的组成	17
4.2 发布新文章	18

4.3 新建文章的 Rake 命令	19
4.4 文章元信息的默认值	21
4.5 文章的固定链接	22
4.6 草稿	23
第 5 章 页面 .....	<b>27</b>
5.1 页面文件的组成	27
5.2 页面的存放位置	28
第 6 章 编写内容 .....	<b>29</b>
6.1 图片	29
6.2 链接到其他文章	30
6.3 代码高亮	30
第 7 章 开发一个简单的主题 .....	<b>33</b>
7.1 主题的组成	33
7.2 Liquid 模板系统	35
7.3 创建一个新主题	35
7.4 编写页头	38
7.5 编写页脚	40
7.6 文章列表	42
7.7 侧边栏	44
7.8 单篇文章页面的模板	45
7.9 页面的模板	47
7.10 404 页面	47
7.11 小结	48
第 8 章 添加评论系统 .....	<b>49</b>
8.1 Disqus	49
8.2 多说	52
第 9 章 Jekyll 的插件系统 .....	<b>57</b>

9.1 显示 Jekyll 版本信息	57
9.2 文章分页	58
9.3 生成网站地图	61
9.4 文章所属的分类	61
9.5 生成归档页面	63
9.6 分类列表	68
9.7 钩子	70
9.8 寻找插件	73
<b>第 10 章 数据文件 .....</b>	<b>75</b>
10.1 分类的名称	75
10.2 作者的信息	78
<b>第 11 章 把网站部署到服务器 .....</b>	<b>83</b>
11.1 GitHub Pages	83
11.2 虚拟主机	93
11.3 再次修改网站配置	96
<b>附录 A: 设置 Jekyll 网站 .....</b>	<b>97</b>
<b>附录 B: Markdown 句法简介 .....</b>	<b>105</b>
<b>附录 C: Liquid 模板引擎简介 .....</b>	<b>123</b>
<b>附录 D: 修订日志 .....</b>	<b>133</b>



---

# 第 1 章 简介

这是一本关于 Jekyll（发音 /dʒi:kil/）的书。我希望通过这本书告诉你 Jekyll 是什么，以及如何使用 Jekyll。首先，我们要弄明白 Jekyll 是什么。

## 1.1 Jekyll 是什么

### 1.1.1 小说中的人物

Jekyll 是英国新浪漫主义小说家 [罗伯特·路易斯·史蒂文森](#) 所著小说《[化身博士](#)》中的角色。《化身博士》讲述了体面绅士 Jekyll 博士喝了自己配制的药剂后化身邪恶的 Hyde 先生的故事。Jekyll 因抵挡不了潜藏在天性中邪恶、狂野因子的耸动，发明了一种药水，将平时被压抑在虚伪表相下的心性，毫无保留地展露出来，同时随着人格心性的转变，身材样貌也会随之改变。原本一个社会公众认为行善不遗余力的温文儒雅之士，一旦喝下药水，即转身一变，成为邪恶、毫无人性且人人憎恶的猥亵男子 Hyde。Jekyll 是善的代表，Hyde 则是恶魔的化身。后来，Jekyll 因控制不了自己内心的恶魔，以自尽的方法来终止自己以 Hyde 的身分作恶。

《[化身博士](#)》被多次改编为影视作品，最近一次是由 BBC 出品的 6 集迷你剧。

### 1.1.2 一个小岛

Jekyll 是美国东南沿海一带群岛中的[一个小岛](#)，隶属佐治亚州格林县。

### 1.1.3 静态网站生成工具

当然，本书要讲的并不是前面两种 Jekyll。我们要讲的 [Jekyll](#) 是一个静态网站生成工具（Static Site Generator）。Jekyll 这个名字取自《[化身博士](#)》，这一点从 Jekyll 以前的网站中可以窥见，如图 1.1 所示。



图 1.1: Jekyll 以前的网站

Jekyll 使用 [Ruby](#) 编程语言开发，用来生成纯静态网站。Jekyll 按照一定规则，把使用 Markdown、Textile、Asciidoc 等书写语言编写的文章转换成 HTML，最终组成一个完整的网站。

那么问题来了，为什么要搭建静态网站呢？

## 1.2 静态网站的好处

接触过博客的读者可能知道，搭建网站有很多知名的程序，例如 [WordPress](#) 和 [Moveable Type](#)，还有很多 CMS（Content Management System，内容管理系统）。为什么要搭建静态网站呢？

其实原因很多，我觉得比较重要的有两点。

## 历史诟病

WordPress 等程序出现的时间都很久了，甚至超过了十年。在这么长时间的开发中，架构再好的程序也难免有被人诟病的地方。就拿 WordPress 来说，其强大功能的背后，是性能的低下，为人所诟病。

## 名人效应

Jekyll 最初的开发者是 Tom Preston-Werner。他是 Gravatar（后来卖给运营 WordPress 的公司 Automattic）的创始人，而后又参与创建了 GitHub。这两个网站可以说在一定程度上改变了网络。很多人都是奔着 Tom 的名气才使用 Jekyll 的，这在一定程度上导致了静态网站工具的流行。Jekyll 出现之前，已经有一些静态网站生成工具，例如 nanoc。但在 Jekyll 之后，各种静态网站生成工具如雨后春笋般兴起，使用各种编程语言编写的都有。具体有多少，可以看一下 Static Site Generators 网站的统计。

抛开这两点，我们来看一下静态网站有什么好处：

- 安全性高。因为静态网站中全是静态内容，服务器端不涉及任何动态语言，所以很少会被攻击。就算被攻击，网站本身也没什么损失。
- 性能高。静态网站都是静态的文件，如 HTML 文件、CSS 样式表、JavaScript 脚本和各种图片，无需在服务器端处理，也不用连接数据库。
- 对服务器的要求低。可以说，静态网站对服务器没有任何要求，不要求安装哪个具体版本的 PHP，不要求安装 MySQL 等数据库——只要服务器能伺服静态网页即可。而且，你甚至不需要自己购买服务器，可以把网站托管到 GitHub Pages 等平台中。
- 入门容易。说得简单点儿，静态网站就是把数据套入模板后生成 HTML 文件，没有复杂的插件机制，模板系统也容易理解，对新手而言，入门非常容易。
- 便于做版本控制。因为静态网站中所有内容都是以文件的形式存储，所以特别易于纳入版本控制系统，如 Git。
- 有极客范儿。不知从何时起，“极客”（geek）这个词开始流行了。如果网站没使用静态网站生成工具搭建，“出门都不好意思跟人打招呼”。

## 1.3 我该不该使用静态网站生成工具

“动态”还是“静态”完全是个人喜好。如果你觉得自己更适合使用 WordPress，那就继续使用。如果你想体验一下新鲜事物，可以试一下静态网站生成工具，说不定会爱上她。

如果你恰好是 Ruby 程序员，或者对 Ruby 感兴趣，那么 Jekyll 定是不二选择。

决定之后，下面开始安装 Jekyll 吧。

## 1.4 预备知识

这是一本面向初学者的书。我说的“初学者”是指刚接触 Jekyll。Jekyll 并不是“零基础”，你需要具备一些基础知识。具体而言，你要掌握：

- 基本的命令行知识，至少要知道什么是命令行，怎么在命令行中执行命令；
- 基本的 HTML 和 CSS 知识，如果懂一点 JavaScript 更好；
- 如果想自己编写插件，或者想深入理解 Jekyll 的运作机制，需要一定的 Ruby 知识；

## 1.5 排版约定

本书没有使用特殊的排版方式，相信多数情况下你都能理解，不过还是要做些说明。

### 等宽字体

表示代码，文件和文件夹名称。

### 斜体字

表示 URL。

### \$ 符号

表示命令行提示符。如果你在代码中看到 \$ 符号，说明随后的内容是在命令行中执行的命令。如果 \$ 符号之前有内容，例如 ~/blog，说明随后的命令要在指定的文件夹中执行。

## 1.6 随书源码

书中用到的代码托管在 GitHub 中，地址为 <https://github.com/jekyll-book>。各章的代码放在单独的仓库中，例如第 3 章的代码在 `jekyll-book/chapter03` 仓库中。

## 1.7 反馈

如果你在阅读本书的过程中发现了错误，或者觉得有些地方没讲清楚，或者想知道其他的用法和功能，或者有好的建议，都可以反馈给我。你可以到本书的[讨论页面](#)发帖，我会尽快回复。也可以直接发电子邮件给我，我的邮箱地址是 [andor.chen.27@gmail.com](mailto:andor.chen.27@gmail.com)。

本书的官方网站：<http://jekyll-china.com>。

## 1.8 致谢

感谢 Tom 和 Jekyll 开发团队开发了这个灵活好用的静态网站生成工具。感谢我的妻子忍受我在写这本书的过程中敲击键盘发出的声音。感谢给我提供反馈的各位读者。感谢所有购买这本书的读者。

## 1.9 版权

本书内容由[安道](#)原创编写，受版权法保护。

书中代码基于[MIT 协议](#)发布。



---

# 第 2 章 安装 Jekyll

Jekyll 是使用 Ruby 编程语言开发的程序，以 gem<sup>1</sup> 的形式分发，所以在安装 Jekyll 之前，要先在系统中安装 Ruby。

## 2.1 安装 Ruby

Mac OS X 和某些基于 Linux 的系统中都已经预装了 Ruby。如果不确定自己的系统中是否安装有 Ruby，可以执行下面的命令检查。在系统的命令行中输入以下命令：

```
$ ruby -v
```

如果看到 Ruby 的版本信息，说明系统中已经安装了 Ruby。如果提示找不到命令，说明系统中没有安装 Ruby，需要自己动手安装。

就算系统中已经安装 Ruby，版本也可能太旧。如果看到的版本号小于 2.0.0，需要安装最新版。

具体怎么安装 Ruby 已经超出了本书范畴，你可以参考 [InstallRails 网站](#) 中的说明。这个网站说明了如何在 Windows、Mac OS X，以及流行的 Linux 发行版中安装 Ruby。后面介绍如何安装 Rails 的步骤可以忽略，不过也可以看一下，因为 Jekyll 的安装方式和 Rails 基本一样。

---

### 使用国内镜像安装 Ruby

由于国内特殊的网络环境，安装 Ruby 时可能发现速度很慢，或者根本无法安装。这时，可以选择使用国内的镜像。

从源码编译安装 Ruby 的用户，可以直接从 [Ruby China 的镜像](#) 中下载相应的版本。使用 rbenv 管理 Ruby 的用户，可以使用笔者开发的 [rbenv-china-mirror 插件](#)。

---

---

1. gem 是 Ruby 语言的包管理和包分发工具，详情参见 <http://rubygems.org>。

## 2.2 安装 Jekyll

Jekyll 以 gem 的形式分发，安装起来十分简单。确认系统中有 Ruby 之后，在命令行中执行以下命令即可：

```
$ gem install jekyll
```

在某些系统中，执行这个命令时可能需要使用 sudo。上述命令会在系统中安装最新版 Jekyll，目前，最新版是 3.2.1。安装完成后，在命令行中执行以下命令，检查 Jekyll 是否正确安装：

```
$ jekyll -v  
jekyll 3.2.1
```

如果输出了 Jekyll 的版本号，说明安装是成功的。

---

### 使用国内镜像安装 gem

由于国内特殊的网络环境，安装 gem 时可能发现速度很慢，或者根本无法安装。这时，可以选择使用国内的镜像。设置方法如下：

```
$ gem sources --add https://gems.ruby-china.org/ --remove  
https://rubygems.org/  
$ gem sources -l  
https://gems.ruby-china.org  
# 确保只有 gems.ruby-china.org
```

---

---

# 第3章 创建第一个网站

安装好 Ruby 和 Jekyll 之后，我们该创建网站了。不过在此之前，我们要知道如何使用 Jekyll 的命令行界面。

## 3.1 命令行界面

Jekyll 是命令行工具，几乎所有操作都在命令行中完成，没有图形化界面（GUI），也没有网页界面（例如 WordPress 的后台）。所以你至少要知道怎么在命令行中运行命令。

在命令行中执行 `jekyll help` 命令，查看帮助信息：

```
$ jekyll _3.2.1_ help

jekyll 3.2.1 -- Jekyll is a blog-aware, static site generator in Ruby

Usage:

jekyll <subcommand> [options]

Options:
  ...
  ...
  ...

Subcommands:
  docs
  import
  build, b           Build your site
  clean              Clean the site ...
  doctor, hyde       Search site and print specific deprecation warnings
  help               Show the help message, optionally for a given subcommand.
  new                Creates a new Jekyll site scaffold in PATH
```

```
new-theme           Creates a new Jekyll theme scaffold  
serve, server, s   Serve your site locally
```

现在我们只关注“Subcommands”（子命令）部分，其他内容先不看。

Jekyll 默认提供了 9 个子命令，各个命令的作用如表 3.1 所示。

表 3.1: Jekyll 子命令及其作用

子命令	作用
docs	在本地查看 Jekyll 文档，必须先安装 <a href="#">jekyll-docs</a> 插件
import	把其他博客程序中的内容导入 Jekyll 网站，必须先安装 <a href="#">jekyll-import</a> 插件
build 或 b	构建网站
clean	清理网站（删除生成的网站和元信息文件）
doctor 或 hyde	扫描整个网站，查找是否用到了弃用的功能，如果有，显示提醒消息
help	显示帮助信息
new	新建一个 Jekyll 网站
new-theme	新建一个 Jekyll 主题（3.2 版新增）
serve, server 或 s	启动本地服务器，预览网站

现在，系统中已经安装好了 Jekyll，下面我们来新建一个网站。根据上表，我们需要使用的子命令是 `new`。

## 3.2 新建网站

在命令行中执行以下命令，新建一个 Jekyll 网站：

```
$ jekyll _3.2.1_ new /path/to/site
```

传给 `new` 命令的参数是保存网站的文件夹路径。假如我们要在家目录中的 `blog` 文件夹中新建一个网站，可以执行下述命令：

```
$ cd ~                      # 进入家目录
~ $ jekyll _3.2.1_ new blog # 新建网站
New jekyll site installed in ~/blog.
```

上述命令在指定的文件夹中创建网站的骨架结构。如果 `blog` 文件夹已经存在，而且其中有内容，这个命令会提醒“Conflict: `~/blog` exists and is not empty”。如果一定要在已经有内容的文件夹中创建 Jekyll 项目，可以使用 `-f` 旗标：

```
~ $ jekyll _3.2.1_ new blog -f
New jekyll site installed in ~/blog.
```

---

### 注意

执行 `jekyll help` 和 `jekyll new` 命令时，我们在命令行中指定了 Jekyll 的版本号。这么做是为了使用指定版本的 Jekyll。因为执行这两个命令时没有 Bundler 环境（[3.3 节](#)），无法限定 Jekyll 的版本，所以我们直接在命令行中指定版本号。

---

## 3.3 网站的文件结构

现在，进入 `~/blog` 文件夹，我们来了解一下 Jekyll 网站的文件结构。

```
~/blog
├── _posts/
├── css/
├── .gitignore
├── _config.yml
├── about.md
├── feed.xml
├── Gemfile
└── index.html
```

各个文件夹和文件的作用如表 3.2 所示。

表 3.2: Jekyll 网站的文件结构

文件（夹）	作用
<code>_posts/</code>	存放博客文章文件
<code>css/</code>	存放样式表

.gitignore	把指定的文件排除在 Git 仓库之外
_config.yml	网站的配置文件
about.md	“关于”页面
feed.xml	网站的订阅源文件
Gemfile	列出网站使用的 gem 和 Jekyll 插件（即依赖）
index.html	网站的入口文件，即首页

你可以在文本编辑器中打开各个文件，看看里面的内容。本书后文会分别讲解各个文件（夹）。

其中需要特别说明的是 `Gemfile` 文件。这个文件用于列出网站使用的 gem 和 Jekyll 插件（也以 gem 的形式分发），供 `Bundler` 使用。`Bundler` 是依赖管理工具，在 Ruby 社区中广泛使用。`Bundler`、`Gemfile` 文件（以及即将提到的 `Gemfile.lock` 文件）组合在一起能确保不同人使用的项目依赖始终相同，从而避免因版本不同而导致的问题。

`Bundler` 本身也是一个 gem，因此要执行下述命令安装：

```
$ gem install bundler
```

执行 `jekyll new` 命令生成网站后，还要运行 `Bundler`，让它读取 `Gemfile` 文件中声明的依赖，安装它们，然后生成 `Gemfile.lock` 文件。注意，我们执行的命令是 `bundle`，而不是 `bundler`，后面没有“r”。

```
~ $ cd blog
~/blog $ bundle install
Fetching gem metadata from https://gems.ruby-china.org/.....
Fetching version metadata from https://gems.ruby-china.org/...
Fetching dependency metadata from https://gems.ruby-china.org/..
Resolving dependencies...
...
...
...
Bundle complete! 2 Gemfile dependencies, 18 gems now installed.
Use `bundle show [gemname]` to see where a bundled gem is installed.
```

---

## 让 Bundler 使用国内镜像

由于国内特殊的网络环境，安装 gem 时可能发现速度很慢，或者根本无法安装。这时，可以选择使用国内的镜像。使用 Bundler 安装 gem 时，可以执行下述命令，设为 [Ruby China 的镜像](#)：

```
$ bundle config mirror.https://rubygems.org https://gems.ruby-china.org
```

当然，也可以直接把 `Gemfile` 文件中的 `source` 改成镜像地址。

---

## 3.4 查看网站

新建网站项目后，你可能很心急，想看一下网站是什么样子。没问题，在命令行中执行下述命令，启动本地服务器：

```
~/blog $ bundle exec jekyll server
Configuration file: ~/blog/_config.yml
  Source: ~/blog
  Destination: ~/blog/_site
Incremental build: disabled. Enable with --incremental
  Generating...
          done in 0.672 seconds.
Auto-regeneration: enabled for '~/blog'
Configuration file: ~/blog/_config.yml
  Server address: http://127.0.0.1:4000/
  Server running... press ctrl-c to stop.
```

注意，我们在 `jekyll` 命令前加上了 `bundle exec`，这么做是为了使用 `Gemfile` 文件中指定的 Jekyll 版本。

然后，打开浏览器，在地址栏中输入 <http://localhost:4000>，就能看到网站的首页了，如图 3.1 所示。

如果想关闭本地服务器，在命令行中按 `Ctrl-C` 键。

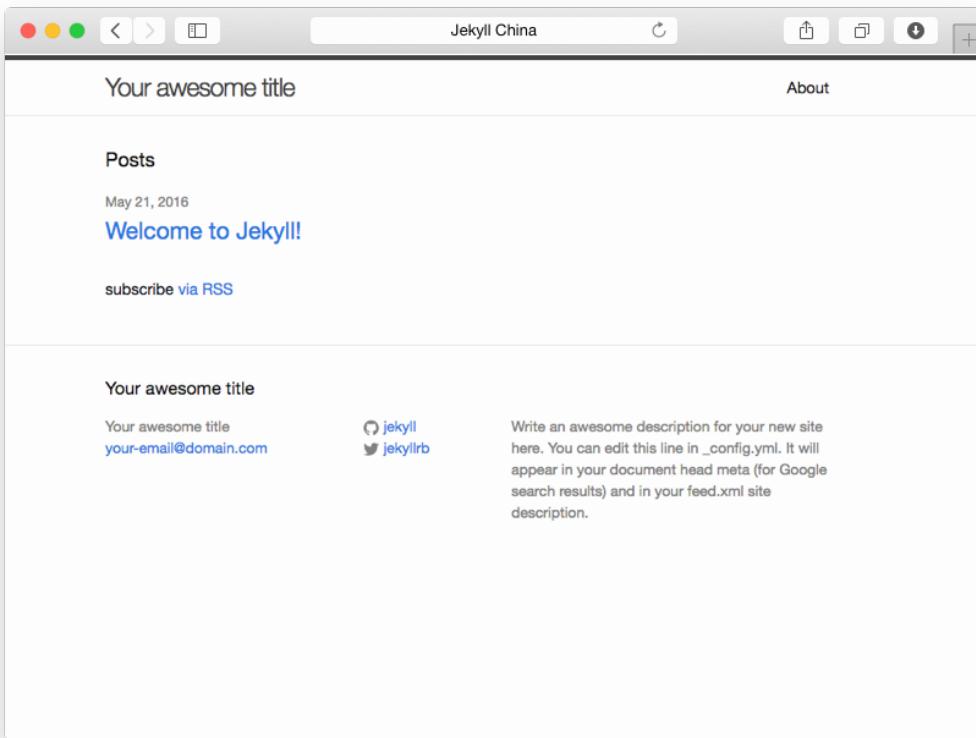


图 3.1：网站的首页

### 3.5 设置网站

仔细观察首页，我们发现有很多信息是占位用的，例如网站的标题和描述信息等，需要根据需求修改。首页的右下角有一段话，从中得知，这些占位信息在 `_config.yml` 文件中。

在文本编辑器中打开 `_config.yml` 文件。这是个 `YAML` 文件，网站所有的配置都保存在这里。文件中以 `#` 号开头的行是注释，通过 Jekyll 生成的这些注释我们可以初步了解这个文件的作用。

你可以根据自己的需求修改网站的标题（`title`）、电子邮件地址（`email`）、Twitter 用户名（`twitter_username`）和 GitHub 用户名（`github_username`）等。例如，我修改后的设置如下：

```
title: Andor Chen
email: andor.chen.27@gmail.com
...
...
url: "http://jekyll-china.com"
twitter_username: andor_chen
github_username: andorchen
```

修改完这些设置之后，我们来看一下生成的网站。进入 `_site` 文件夹，你会看到很多文件（夹），其中有两个文件并不是网站的一部分，即 `Gemfile` 和 `Gemfile.lock`。为了不让这两个文件出现在生成的网站中，我们要使用 `exclude` 设置。在 `_config.yml` 文件中添加下述设置：

```
exclude: ["Gemfile", "Gemfile.lock"]
```

修改设置后，要重启本地服务器才能生效：先按 `Ctrl-C` 键关闭本地服务器，然后执行 `bundle exec jekyll server` 命令重启。



# 第 4 章 文章

在本地预览网站时可以看到，新建的网站中已经有了一篇文章，就像 WordPress 网站中的第一篇文章“Hello, world!”一样。点击“Welcome to Jekyll！”，打开这篇文章，阅读其中的内容。

但是，这篇文章在哪儿？如果想发布新文章该怎么做呢？本章为你解答这些疑问。

## 4.1 文章文件的组成

你应该还记得网站的文件结构（3.3 节）中有一个名为 `_posts` 的文件夹，打开它，可以看到第一篇文章对应的文件。我的文件名是 `2016-05-21-welcome-to-jekyll.markdown`，你的文件名可能和我不一样，具体来说，是数字部分不一样。

文章对应的文件，其文件名由三部分组成：日期，别名（slug）和扩展名。对默认生成的第一篇文章来说，日期是“2016-05-21”，别名是“welcome-to-jekyll”，扩展名是“.markdown”。日期要满足特定的条件：年必须使用四位数字表示，例如 `2016`；月必须使用两位数字表示，例如 `02, 11`；日也必须使用两位数字表示，例如 `03, 27`。别名就是在浏览器中显示的这篇文章的 URL，一般使用英文，单词之间使用“-”（连字符）连接。扩展名表示这篇文章使用什么书写语言，“`.markdown`”表示 `Markdown`。



图 4.1：文章文件名的构成

在文本编辑器中打开这个文件，看一下里面的内容。文章对应的文件由两部分组成：头部元信息（frontmatter）和正文。

元信息包围在两行 `---`（三个连字符）之间，使用 `YAML` 格式定义。在默认生成的第一篇文章中，元信息为：

```
---
layout: post
```

```
title: "Welcome to Jekyll!"  
date: 2016-05-21 15:32:06 +0800  
categories: jekyll update  
---
```

其中：

- `layout` 指定文章使用哪个布局渲染；
- `title` 是文章的标题；
- `date` 是文章的发布日期、时间和时区；
- `categories` 是文章所属的分类，多个分类之间使用空格或英文逗号分开。

默认生成的第一篇文章使用 Markdown 命法编写。Markdown 命法的简介参见[附录 B](#)。

## 4.2 发布新文章

了解文章的基本结构之后，我们来发布一篇新文章。如果你觉得默认生成的第一篇文章没用，可以把它删掉。在 `_posts` 文件夹中新建一个文件，文件名随意取，能表明文章的主旨即可，但要符合前面所讲的格式。例如，`2016-05-21-first-post.markdown`。

在文本编辑器中打开新建的文件，写入元信息。例如：

```
---  
layout: post  
title: "我的第一篇文章"  
date: 2016-05-21 16:45:27 +0800  
categories: news  
---
```

然后，把你想要发布的内容用 Markdown 命法写出来。写完后，在命令行中执行下述命令，启动本地服务器（同时也会生成网站）：

```
~/blog $ bundle exec jekyll server
```

---

### 提示

你可能觉得每次启动本地服务器都输入 `server` 子命令有点麻烦，想知道有没有快捷方式。当然有，`server` 的简写是 `s`，如表 3.1 所示。因此，启动本地服务器时，为了少输入几个字符，可以执行 `jekyll s` 命令。

---

然后在浏览器中打开 <http://localhost:4000>，就能看到这篇文章出现在首页中。点击首页中这篇文章的标题，即可打开这篇文章。

这就是日常发布新文章的流程：在 `_posts` 文件夹中新建一个文件，写入元信息和内容，启动本地服务器预览。当然，后面还有一步——部署，不过现在先不管，第 11 章再介绍。

## 4.3 新建文章的 Rake 命令

如果经常发布文章，你会觉得每次都自己动手新建文件，再写入元信息有点麻烦。有没有办法可以把这个操作交给电脑自动完成呢？<sup>1</sup> 答案是肯定的，我们可以把这个操作交给 `rake` 任务完成。

### rake 是什么

简单来说，`rake`<sup>2</sup> 是 Ruby 界的 `make`，一种构件和自动化工具。`rake` 定义了一套简单的领域特定语言（Domain-Specific Language，简称 DSL），简化了任务的编写。使用 `rake` 任务可以自动完成一些操作。这里，我们要让 `rake` 帮我们新建文件，并在文件中写入一些默认的内容。

`rake` 也是以 `gem` 的形式分发的，我们将其添加到 `Gemfile` 文件中：

```
...
gem "jekyll", "3.2.1"
gem "rake"
...
```

---

1. 人和电脑都有自己存在的意义。电脑出现的目的之一是，帮助人类完成重复的操作。不要心存疑虑，这就是电脑的使命。能交给电脑完成的操作绝不要自己动手。

2. 2014 年 2 月，`rake` 的核心开发者 Jim Weirich 不幸离世，他是很活跃的 Ruby 开发者，开发了很多 Ruby 程序员常用的程序。R.I.P

然后，在命令行中执行 `bundle install` 命令，安装 rake。

rake 任务在一个特殊的文件中定义，名为 `Rakefile`。下面，在网站的根目录中新建文件 `Rakefile`，然后在文本编辑器中打开，写入下述内容：

代码清单 4.1：新建文章的 rake 任务

```
require 'time'

# Usage: rake post title="A Title" [date="2014-04-14"]
desc "Create a new post"
task :post do
  unless FileTest.directory?( './_posts' )
    abort("rake aborted: '_posts' directory not found.")
  end
  title = ENV["title"] || "new-post"
  slug = title.downcase.strip.gsub(' ', '-').gsub(/[^\w-]/, '')
  begin
    datetime = (ENV['date'] ? Time.parse(ENV['date']) : Time.now)
      .strftime('%Y-%m-%d %H:%M:%S %z')
    date = datetime.split.first
  rescue Exception => e
    puts "Error: date format must be YYYY-MM-DD!"
    exit -1
  end
  filename = File.join('.', '_posts', "#{date}-#{slug}.md")
  if File.exist?(filename)
    abort("rake aborted: #{filename} already exists.")
  end

  puts "Creating new post: #{filename}"
  open(filename, 'w') do |post|
    post.puts "---"
    post.puts "layout: post"
    post.puts "title: \"#{title.gsub(/-/,' ')}\""
    post.puts "date: #{datetime}"
    post.puts "categories:"
    post.puts "---"
  end
end
```

```
end  
end
```

`Rakefile` 使用 Ruby 语言编写。上述代码定义了一个 `rake` 任务，名为 `post`。这个任务的作用是，在 `_posts` 文件夹中新建一个文件，写入一些默认内容。我们来看一下如何执行这个任务。

```
~/blog $ bundle exec rake post title="Hello, again"  
Creating new post: ./_posts/2016-05-21-hello-again.md  
  
# date 选项是可选的  
~/blog $ bundle exec rake post title="Hello, there" date="2016-05-21"  
Creating new post: ./_posts/2016-05-21-hello-there.md
```

你可以自己执行试一下，然后进入 `_posts` 文件夹确认有没有生成文件，再打开新建的文件，看看其中是不是有元信息。

细心的读者可能发现了，新建的文件扩展名为 `.md`，和默认生成的第一篇文章不一样。这没关系，Jekyll 也会把扩展名为 `.md` 的文件识别为使用 Markdown 句法编写的文章。除此之外，Jekyll 默认能识别的 Markdown 文件扩展名还有 `.mkdown`、`.mkdn` 和 `.mkd`。你可以根据自己的喜好选择使用某个扩展名。不过简单起见，一般都使用 `.md`。

---

### 提示

如 3.5 节所述，为了不让 `Rakefile` 文件出现在生成的网站中，我们要把它添加到 `exclude` 设置中：

```
exclude: ["Gemfile", "Gemfile.lock", "Rakefile"]
```

---

## 4.4 文章元信息的默认值

有了前面的 `rake` 任务，创建新文章的过程大大简化了，不过还有一个问题。如果你是程序员，一定知道 **DRY 原则**，即不要重复表述相同的信息。在文章的元信息中就有信息重复——布局。一般情况下，所有文章都会使用相同的布局渲染，偶尔可能会使用特别的布局渲染一两篇特殊的文章。既然如此，为什么每篇文章都指定相同的布局信息呢？！Jekyll 允许我们为元信息设置默认值，从而避免这个问题。

打开 `_config.yml` 文件，写入下述设置：

```
defaults:
```

```
scope:  
  path: ""  
  type: "posts"  
values:  
  layout: "post"
```

保存文件，然后重启本地服务器。打开一篇文章，去掉元信息中的 `layout` 设置。你会发现，那篇文章仍会使用 `post` 布局渲染。

## 4.5 文章的固定链接

在浏览器中打开一篇文章后，注意地址栏中的地址。可以看出，Jekyll 默认使用的固定链接格式是“分类-日期-别名”，这种格式的代号是 `date`。`date` 只是一个代号，具体的内容是：`/:categories/:year/:month/:day/:title.html`。除了 `date` 之外，Jekyll 还内置了另外两种固定链接格式：

- `pretty` → `/:categories/:year/:month/:day/:title/`
- `none` → `/:categories/:title.html`

固定链接的格式在 `_config.yml` 文件中设置。如果不想使用默认的固定链接格式，可以在这个文件中设置。在文本编辑器中打开 `_config.yml`，新起一行写入如下内容：

```
permalink: 'pretty'
```

保存 `_config.yml` 文件。如果已经启动本地服务，在命令行中按 `Ctrl-C` 键关闭服务器，然后再执行 `jekyll s` 命令重启服务器。在浏览器中打开网站首页，点击某篇文章的标题，留意地址栏中显示的地址，会发现格式变了：后面的 `.html` 没了。

如果 Jekyll 内置的固定链接格式无法满足你的需求，还可以自定义。方法一样，只不过不再使用内置的代号，我们要使用固定链接变量。

如果只想显示标题，可以这样设置：

```
permalink: '/:title.html'
```

如果还想包括年份，可以这样设置：

```
permalink: '/:year/:title.html'
```

如果不想要后面的 `.html`，可以这样设置：

```
permalink: '/:title/'
```

---

## 注意

修改 `_config.yml` 文件之后必须重启本地服务器，设置才能生效。还记得方法吗？先按 `Ctrl-C` 键，然后执行 `bundle exec jekyll s` 命令。

---

Jekyll 支持很多固定链接变量，如表 4.1 所示，你可以根据自己的需求灵活使用。

表 4.1：Jekyll 支持的固定链接变量

变量	说明	举例
<code>year</code>	从文章文件名中读取的年份，四位数字	1999, 2014
<code>short_year</code>	从文章文件名中读取的年份，只有后两位数字	99, 14
<code>month</code>	从文章文件名中读取的月份，两位数字	03, 11
<code>i_month</code>	从文章文件名中读取的月份，没有前导零	3, 11
<code>day</code>	从文章文件名中读取的日，两位数字	05, 27
<code>i_day</code>	从文章文件名中读取的日，没有前导零	5, 27
<code>title</code>	从文章文件名中读取的标题	first-post
<code>slug</code>	与 <code>title</code> 类似，不过字母和数字之外的字符会被替换成连字符	first-post
<code>categories</code>	文章所属的分类	jekyll update

注意，设置固定链接时，变量前面一定要加上：`:`（英文冒号），否则会被识别为纯文本，不会被 Jekyll 代换。

## 4.6 草稿

如果想记录一件事，但一时半会儿写不完，不想发布，那么可以先把文章保存为草稿。Jekyll 支持草稿功能。草稿保存在网站根目录中的 `_drafts` 文件夹里。执行 `jekyll new` 命令新建项目时，没有生成这个文件夹，因此需要自己动手创建。

打开命令行，进入网站所在的根目录，执行下述命令，创建 `_drafts` 文件夹：

```
~/blog $ mkdir _drafts
```

---

### 注意

新建文件夹的操作也可以在 GUI（图形界面）中完成，但是为了熟悉命令行操作，建议你在命令行中完成。书中的操作，请尽量都在命令行中完成。用得多了，终有一天你会克服对命令行的恐惧。

---

草稿和文章相比，除了存放位置不一样之外，文件的标题也不一样——草稿文件的文件名中没有日期。其他都和文章一样。

如果草稿写好了，决定发布，可以把草稿文件移到 `_posts` 文件夹中，然后在文件名中加入发布的日期。

在本地预览时，如果想查看草稿，要指定 `--drafts` 选项，例如 `bundle exec jekyll s --drafts`。

如果你感兴趣，可以参照前面新建文章的 `rake` 命令，自己试着编写一个 `rake` 任务，用来新建草稿。如果你对 Ruby 语言不熟悉，可以直接把下面的代码添加到 `Rakefile` 文件中。

代码清单 4.2：新建草稿的 `rake` 任务

```
# Usage: rake draft title="A Title"
desc "Create a new draft"
task :draft do
  unless FileTest.directory?( './_drafts' )
    abort("rake aborted: '_drafts' directory not found.")
  end
  title = ENV["title"] || "new-post"
  slug = title.downcase.strip.gsub(' ', '-').gsub(/[^\w-]/, '')
  filename = File.join('.', '_drafts', "#{slug}.md")
  if File.exist?(filename)
    abort("rake aborted: #{filename} already exists.")
  end

  puts "Creating new draft: #{filename}"
  open(filename, 'w') do |post|
    post.puts "---"
    post.puts "layout: post"
```

```
post.puts "title: \"#{title.gsub(/-/,' ')}\""
post.puts "date:"
post.puts "categories:"
post.puts "---"
end
end
```

这个 rake 任务的用法和新建文章的任务类似，但不用指定 date 参数：

```
~/blog $ bundle exec rake draft title="A fake draft"
```



---

# 第 5 章 页面

启动本地浏览器，在浏览器中打开 <http://localhost:4000>。网页右上角有一个链接“About”，点击，打开那个网页。这个网页显示的就是“页面”。页面一般用于发布“关于”、“联系方式”等内容。

在文件浏览器中打开网站的根目录，其中有个文件名为 `about.md`，这就是生成刚才那个页面的文件。在文本编辑器中打开这个文件，看一下其中的内容。

## 5.1 页面文件的组成

可以看到，页面文件的组成和[文章文件](#)相同：头部元信息和正文。

页面一般不关注时效性，所以页面文件的文件名中没有日期，而且元信息中也没有定义 `date`。

对比页面文件的元信息和文章文件的元信息，可以发现有几处不同。首先，`layout` 不同。页面使用的布局是 `page`，而文章使用的布局是 `post`。这很容易理解，因为页面和文章需要显示的内容不同，因此使用两个不同的布局渲染。

其次，没有 `date` 和 `categories`。因为页面不关注时效性，也不需要分类。

再者，多了 `permalink`。这个元信息设置页面的固定链接，是可选的。如果不指定，就按照页面文件的路径生成固定链接。例如，`about.md` 在根目录中，所以它的固定链接是 `/about.html`。如果是其他文件夹中的文件，例如 `projects/about.md`，那么固定链接就是 `/projects/about.html`。

如果不想使用这种默认的固定链接格式，就可以定义 `permalink`，例如定义为 `/about/`。其实固定链接是什么，可以随意设置，没必要和文件名保持一致，如果愿意，甚至可以定义成 `/cong-jia-wang/`。

如果觉得默认生成的“About”页面有用，可以保留，修改正文，介绍一下自己和这个网站，继续使用。下面我们来看把页面文件放在哪里。

## 5.2 页面的存放位置

你可能注意到了，项目的根目录中有很多以下划线开头的文件和文件夹。这些是特殊的文件和文件夹，不会出现在最终生成的网站中。不信的话，打开`_site`文件夹看看有没有`_posts`文件夹，有没有`_config.yml`文件。但是，`_site`文件夹中有`css`文件夹。这说明，Jekyll会特殊处理以下划线开头的文件和文件夹。

这跟页面有什么关系呢？当然有关系。你可以把所有页面都放在根目录中，保存为一个个文件。但是，如果想集中保存一类页面，就可以放在文件夹中。这样也可以形成一种父子关系。假如我是一名设计师，我为很多公司设计了海报，我想做一个展示页面，显示我设计的所有海报，而且点击每个海报后，会打开一个页面，详细介绍海报。这就是一种父子关系。我可以在网站的根目录中创建一个文件夹，命名为`posters`。然后在这个文件夹中新建一个页面，命名为`index.md`，在这个文件中列出所有海报。然后在`posters`文件夹中创建很多页面，分别介绍每张海报。

假如`posters`文件夹的结构为：

```
posters
├── index.md
├── saving.md
└── no-smoking.md
```

那么在生成的网站中，会得到如下的结构：

```
posters
├── index.html
├── saving.html
└── no-smoking.html
```

然后我就可以把网址<http://example.com/posters/>（这个地址显示的是`index.html`）告诉别人，其他人访问这个网址就能看到我设计的所有海报，点击每张海报之后还会显示介绍这张海报的页面，比如说<http://example.com/posters/no-smoking.html>。

由此可见，除了以下划线开头的特殊文件或文件夹之外，其他文件和文件夹都会被Jekyll识别为页面文件（资源文件除外，例如样式表、图片和JavaScript文件等）。知道这一点之后，你就可以按照自己的需求自由组织页面文件了。

---

# 第 6 章 编写内容

Jekyll 默认支持使用 [Markdown](#) 编写内容。Markdown 只是一种书写语言，若想变成网页中显示的内容，要经过转换程序将其转换成 HTML，然后再使用 CSS 修饰外观。Markdown 转换程序有很多，在 Ruby 领域比较流行的有：[redcarpet](#), [kramdown](#), [rdiscount](#)。这几种转换程序 Jekyll 都支持。

在文本编辑器中打开 `_config.yml` 文件，你会看到一个设置是 `markdown`。它的作用就是设置使用哪种转换程序。Jekyll 默认使用 kramdown。

Markdown 的句法以及 kramdown 在标准句法之外做的扩展在此不一一介绍。如果想要全面了解，可以阅读[附录 B](#) 或者 [kramdown 的文档](#)。本章重点介绍一些常用的或者容易出错的内容编写方式。

## 6.1 图片

Markdown [插入图片的句法](#)很简单：

```
![Alt text](/path/to/img.jpg "Optional title")
```

其中，Alt 和 Title 都是可选的。下面我们来看一下如何在 Jekyll 网站中正确插入图片。我所说的“正确”，是指图片的路径正确。

假如我们把网站中用到的图片统一放在根目录中的 `images` 文件夹里，其中有个图片 `hello.jpg`。若想在某篇文章中插入这张图片，应该这么写：

```

```

注意，图片的路径前面加上了 `{{ site.baseurl }}`。这么做是为了防止你把网站部署到子文件夹时出错，例如 `http://yourdomain.com/blog/`。如果不加 `baseurl`，直接写成 `/images/hello.jpg`，图片的地址就变成了 `http://yourdomain.com/images/hello.jpg`，无法加载。你可能想，写成 `images/hello.jpg`（前面没有 `/`）可以吗？也不可以。假如你设置的[固定链接格式](#)是 `date`（默认值），那么图片的地址会变成 `http://yourdomain.com/blog/jekyll/update/2016/02/03/images/hello.jpg`（以默认生成的第一篇文章为例），这样显然无法加载图片。

## 6.2 链接到其他文章

如果你想在某篇文章中链接到网站中的其他文章，可以使用 Jekyll 提供的模板标签 `post_url`。例如，链接到默认生成的第一篇文章可以这么写：

```
[第一篇文章]({% post_url 2016-02-03-welcome-to-jekyll %})
```

可以看出，只需指定文章文件的名字，不用指定扩展名。当然，你也可以直接写某篇文章的地址，但是如果修改了[固定链接格式](#)，链接就会失效。

## 6.3 代码高亮

在 Jekyll 网站中高亮代码有两种方式，一种是使用 `highlight` 模板标签，另一种是使用 kramdown 提供的高亮句法。

### 6.3.1 `highlight` 模板标签

Jekyll 提供了一个模板标签，用于高亮代码块：`{% highlight %}`。这个标签默认使用 `rouge`<sup>1</sup> 进行高亮。基本用法如下：

```
{% highlight ruby %}
def print_hi(name)
  puts "Hi, #{name}"
end
{% endhighlight %}
```

其中，`ruby` 是这段代码的编程语言。上述代码生成的 HTML 如下（排版需要，做了缩进和断行），你可以根据所用的 CSS 类为代码块编写样式：

```
<div class="highlight">
<pre>
  <code class="language-ruby" data-lang="ruby">
    <span class="k">def</span>
    <span class="nf">print_hi</span>
    <span class="p">>(</span>
```

---

1. Jekyll 以前默认使用 [Pygments](#) 高亮代码，但这是个 Python 程序。`rouge` 算是 Pygments 的 Ruby 移植版，高亮的方式和最终输出的 HTML 与 Pygments 基本相同，大多数情况下可以代替 Pygments。

```
<span class="nb">name</span>
<span class="p">) </span>
<span class="nb">puts</span>
<span class="s2">"Hi, </span>
<span class="si">#{</span>
<span class="nb">name</span>
<span class="si">} </span>
<span class="s2">" </span>
<span class="k">end</span>
</code>
</pre>
</div>
```

### 6.3.2 kramdown 原生支持的句法

kramdown 内置了代码块句法，写法如下：

```
~~~ruby
def print_hi(name)
  puts "Hi, #{name}"
end
~~~
```

kramdown 默认使用 [CodeRay](#) 高亮代码，但是笔者不建议使用它。从 kramdown 1.5.0 开始，已经集成了对 rouge 的支持。不过，Jekyll 已经做了内部处理，我们无需做任何设置，就能使用 rouge 高亮代码。

### 6.3.3 GitHub 使用的句法

除了 `~~~` 句法，kramdown 也支持 [GitHub 使用的高亮句法](#)，即 `````。基本用法如下：

```
```ruby
def print_hi(name)
  puts "Hi, #{name}"
end
```
```

但是，若想使用这种句法，要做个设置——在 `_config.yml` 中写入：

```
kramdown:  
  input: GFM
```

修改设置之后，记得要重启服务器！同样，Jekyll 已经做了内部处理，我们无需做任何设置，就能使用 rouge 高亮代码。

### 6.3.4 禁止过度解析代码块

很多模板引擎，比如说 Liquid、Handlebars、Jinja2，使用 `{{...}}` 带入变量的值。如果在代码块中出现这样的字符，Jekyll 会将其解析成 Liquid 标签（[7.2 节](#)），导致输出的内容与预期不符。比如说我写了一篇介绍 Liquid 的文章，在文章中不可避免地要讲到如何带入变量。此时，我可能会使用下面这个代码块做演示：

```
User Name: {{ user.username }}  
Password: {{ user.password }}
```

这是我想显示的内容。在文章中，Markdown 源码要这样写：

```
```  
User Name: {{ user.username }}  
Password: {{ user.password }}  
```
```

写好文章后保存，然后在本地预览。你会发现，网页中显示的内容是：

```
User Name:  
Password:
```

`{{ user.username }}` 和 `{{ user.password }}` 不见了，为什么呢？这是因为 Jekyll 的模板引擎 Liquid（参见[第 7 章](#)）也使用 `{{ }}` 标记变量，在解析文章时，Jekyll 把代码块里的 `{{...}}` 当成模板标签解析了，而解析的结果是空 (`nil`)，因此在转换得到的文章中，那两处显示为空。

这个问题的解决方法很简单，把包含这些特殊字符的代码块放入 `{% raw %}` 标签里即可。例如：

```
{% raw %}  
```  
User Name: {{ user.username }}  
Password: {{ user.password }}  
```  
{% endraw %}
```

---

# 第7章 开发一个简单的主题

Jekyll 为我们提供了一个默认主题，而且看起来还不错。但是我们需要个性，不想和别人使用相同的主题。想要个性，只能自己动手，开发属于自己的独一无二的主题。

Jekyll 的主题机制非常简单，简单到只需一个模板即可。我们先来看一下默认的主题，了解 Jekyll 的主题机制。

---

## 提醒

Jekyll 3.2 增加了主题分发机制——主题包。主题包以 gem 的形式分发，便于传播和使用。但是，这个功能不支持在主题包中内置插件，所以我们暂且不用，还是直接把主题写在网站中。

---

## 7.1 主题的组成

\_config.yml 文件中有这样一个设置：theme: minima。这个设置的作用是指定网站使用名为 minima 的主题包。这个主题包以 gem 的形式分发，项目的地址是 <https://github.com/jekyll/minima>。在浏览器中打开这个网址，我们来了解一下主题的结构。

可以看到，根目录中有两个以下划线开头的文件夹，一个是 \_layouts，一个是 \_includes。主题所需的模板文件都放在这两个文件夹中。此外，还有一个 \_sass 文件夹，存放着 Sass 样式表文件。本书不会展开说明 Sass。

在浏览器中打开 \_layouts/default.html。可以看到，这个文件的内容非常少，算上空行只有 20 行，如代码清单 7.1 所示。这就是网站使用的默认模板。注意，其中有三行 {% include ... %} (L4, L8 和 L16)，作用是“导入”\_includes 文件夹中对应的文件：第 4 行导入 \_includes/header.html，第 8 行导入 \_includes/header.html，第 16 行导入 \_includes/footer.html。

代码清单 7.1：默认主题中 \_layouts/default.html 文件的内容

```
1  <!DOCTYPE html>
2  <html lang="{{ page.lang | default: site.lang | default: "en" }}>
```

```
3
4     {% include head.html %}
5
6     <body>
7
8         {% include header.html %}
9
10        <main class="page-content" aria-label="Content">
11            <div class="wrapper">
12                {{ content }}
13            </div>
14        </main>
15
16        {% include footer.html %}
17
18    </body>
19
20 </html>
```

为什么要导入其他文件？这么做是为了把一个完整的 HTML 文件拆分为不同的部分，保持结构清晰：把页头放在单独的文件 `header.html` 中，把页脚放在单独的文件 `footer.html` 中。而且这么做还便于代码重用。

分拆不分拆（或者用不用 `_includes` 文件夹）完全由你自己决定。如果觉得把所有代码都写在 `default.html` 中更好，就不用分拆。

然后，在浏览器中打开 `_layouts/post.html`。这个模板用于渲染文章。注意，这个文件的头部有元信息：

```
...
layout: default
...
```

这个元信息的作用是指定这个模板继承哪个模板。也就是说，`post.html` 模板继承自 `default.html`，重用后者的一些 HTML 代码。再看一下 `default.html`，注意第 12 行：`{{ content }}`。对 `post.html` 来说，这行代码的作用是，把 `post.html` 中的代码“代入”这里。也就是说，`content` 的值就是渲染 `post.html` 得到的结果。

同样地，`_layouts/page.html` 中也有元信息，指明继承 `default.html`。其中的内容会替换 `default.html` 中的 `{{ content }}`。

细心的读者可能发现了，`post.html` 和 `page.html` 中也都有 `{{ content }}`。这和 `default.html` 中的 `{{ content }}` 作用不一样。以 `post.html` 为例，其中的 `{{ content }}` 是指文章的内容。

你可能还记得，在文章文件的元信息中有 `layout: post`，其作用是设定文章使用哪个模板渲染。由此我们可以推断出，Jekyll 并不假定文章一定使用 `post.html` 模板渲染。也就是说，文章和 `post.html` 没有内在联系。如果愿意，你可以编写一个模板，命名为 `foobar.html`，然后在文章的元信息中设定 `layout: foobar`，使用这个模板渲染文章。页面同理。

现在，你应该对 Jekyll 的模板有了大致的了解。那么，静态文件（CSS，JavaScript 和图片）怎么办，放在哪儿？随便。根目录中随便放，只要不放在以下划线开头的文件夹里就行。

## 7.2 Liquid 模板系统

你可能还有个疑问：`{% include ... %}` 和 `{{ content }}` 这种句法是什么意思？Jekyll 的模板使用 [Liquid](#) 模板系统编写，这些就是 Liquid 提供的句法。

`{% %}` 执行一个语句，例如：

```
{% if is_iphone6 %}
    "Biger than bigger"
{% endif %}
```

`{{ }}` 的作用是把语句执行的结果输出到所在位置，例如：

我的电子邮件是：{{ site.email }}

（`email` 的值在 `_config.yml` 文件中设定。）

Liquid 模板系统的功能还有很多，在此不一一介绍，详情参见[附录 C](#)。

## 7.3 创建一个新主题

好了，现在我们已经大概了解了 Jekyll 的主题是怎么回事。那就动手实践，创建一个新主题吧。我们以本书创建的第一个网站（“blog”）为例。

在自己编写主题之前，我们要把 Jekyll 默认使用的 `minima` 主题包删掉。在文本编辑器中打开网站根目录里的 `Gemfile` 文件，把 `gem "minima"` 那行删掉。保存文件，在命令行中执行 `bundle`

`install` 命令。最后，在文本编辑器中打开网站根目录里的 `_config.yml` 文件，把 `theme: minima` 那行删掉。

---

### 提醒

此时，如果执行 `bundle exec jekyll s` 命令预览网站，会看到报错信息。这是因为 `about.md` 文件引用了 `minima` 主题包中 `_includes` 文件夹里的几个文件。所以，在继续之前，我们要修改 `about.md` 文件中的内容：可以把里面的内容全部删掉，然后写一些文字介绍自己和这个网站；当然，如果用不到这个文件，也可以把它删掉。

这里，我们把它修改成下述内容：

```
---
layout: page
title: 关于
permalink: /about/
---
```

这是 Jekyll 小书的演示网站。

---

现在可以开始开发主题了。我们不完全从零开始，而是使用一个前端框架——[Bootstrap](#)。[Bootstrap](#) 框架由 Twitter 开发，提供了很多实用的前端组件（CSS 和 JavaScript），而且能实现[响应式布局](#)。

---

### 说明

我知道，Bootstrap 已经烂大街了。但是对于一本入门书而言，使用框架更利于专注问题本身，不用花时间编写样板代码，可以集中精力介绍如何开发 Jekyll 主题。

况且，“烂大街”的另一层意思不就是“受欢迎”吗，这也说明我们选择 Bootstrap 是对的。

---

我们要开发的是一个两栏主题，构思图如下所示。



图 7.1：主题构思图

先编写整体结构，也就是 `default.html`。在网站根目录中新建 `_layouts` 文件夹，然后在里面新建 `default.html` 文件。在文本编辑器中打开，写入如下内容：

**代码清单 7.2：\_layouts/default.html 文件的内容**

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

然后，我们要在 `_includes` 文件夹中创建两个文件：`header.html` 和 `footer.html`。先来看 `header.html`。

## 7.4 编写页头

在网站根目录中新建 `_includes` 文件夹，在里面新建文件 `header.html`，参照 Bootstrap 的基本模板，写入如下内容：

代码清单 7.3: `_includes/header.html` 文件的内容

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap 101 Template</title>

    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet">

    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media
        queries -->
    <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
    <!--[if lt IE 9]>
      <script src="https://oss.maxcdn.com/html5shiv/3.7.2/
      html5shiv.min.js"></script>
      <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
    <![endif]-->
  </head>
  <body>
```

然后，我们要修改几个地方。首先是 `title` 元素。我们要使用模板标签替换这个元素的内容，改成 `{{ site.title }}`。然后修改样式表的地址。我们要使用 CDN，<sup>1</sup> 而不自己伺服。

我们将使用 Bootstrap 默认提供的 CDN。替换后的结果如下：

代码清单 7.4: 换用 CDN 提供的静态资源文件

```
<!DOCTYPE html>
<html lang="en">
```

---

1. 为什么要用 CDN？可以参考[这篇文章](#)。

```

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>{{ site.title }}</title>

    <!-- Bootstrap -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
bootstrap.min.css" rel="stylesheet">

    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media
queries -->
    <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
    <!--[if lt IE 9]>
        <script src="https://oss.maxcdn.com/html5shiv/3.7.2/
html5shiv.min.js"></script>
        <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
    <![endif]-->
</head>
<body>

```

参照主题的构思图（图 7.1），接下来编写“页头”。根据 [Bootstrap 文档](#)，在 `header.html` 文件中写入如下 HTML 代码：

代码清单 7.5：网站的“页头”

```

...
</head>
<body>

<nav class="navbar navbar-default">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse"
                data-target="#navbar-collapse">
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="{{ site.baseurl }}/">{{ site.title }}</a>

```

```
</div>

<div class="collapse navbar-collapse" id="navbar-collapse">
  <ul class="nav navbar-nav navbar-right">
    <li><a href="{{ site.baseurl }}/">首页</a></li>
    <li><a href="{{ site.baseurl }}/about/">关于</a></li>
  </ul>
</div>
</div>
</nav>
```

写入之后，我们发现，现在 `header.html` 文件的内容有点儿多，那我们就把这段代码拿出来，写入单独的文件吧。在 `_includes` 文件夹中新建文件 `navigation.html`，从 `header.html` 中剪切整个 `nav` 元素，粘贴到这个文件中，然后在 `header.html` 的末尾引入：

```
...
</head>
<body>
  {% include navigation.html %}
```

接下来编写“页脚”。

## 7.5 编写页脚

网站的页脚一般放置版权声明等信息。在 `_includes` 文件夹中新建文件 `footer.html`，写入下述 HTML 代码：

代码清单 7.6：网站的“页脚”

```
<footer class="footer">
  <p>&copy; {{ site.time | date: '%Y' }} {{ site.title }}</p>
</footer>

</body>
</html>
```

这里，我们使用 `site.time` 获取生成网站时的时间，然后使用 `date` 过滤器从中获取年份。`date` 是 Liquid 提供的过滤器，使用特定的占位符格式化日期，可以使用的占位符参见 [Ruby 文档](#)。

接下来，打开 `css` 文件夹里的 `main.scss` 文件。这个文件的扩展名是 `.scss`，表明这是一个 `Sass` 文件。Sass 是一个 CSS 预处理器，提供了一套句法，用于简化样式表的编写。不过，Sass 也兼容普通的 CSS 句法。<sup>2</sup>为了不涉及太多知识，我们不会使用 Sass 的句法，而是直接使用常规的 CSS 句法。如果你对 Sass 感兴趣，请阅读[它的文档](#)。

把 `main.scss` 文件中的内容全部删掉，然后写入下述代码：

#### 代码清单 7.7：“页脚”的 CSS 样式

```
---
---

.footer {
  border-top: 1px solid #e5e5e5;
  padding: 20px 0;
  text-align: center;
}
```

注意，前两行是头部元信息。我们只需两行分隔符即可，里面不用设置 `layout` 等。如果没有头部元信息，Jekyll 不会把 Sass 文件转换成 CSS 文件。

我们还没在模板中引入这个样式表。在文本编辑器中打开 `header.html` 文件，在引入 Bootstrap 的样式表那行后面，引入这个样式表：

```
<!-- Bootstrap -->
<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
      rel="stylesheet">
<link href="{{ site.baseurl}}/css/main.css" rel="stylesheet">
```

注意，样式表的扩展名是 `.css`，而不是 `.scss`，因为我们要引入的是编译 SCSS 得到的 CSS 文件。

现在执行 `bundle exec jekyll s` 命令，预览网站。可以看到，页头和页脚都创建好了。

---

2. 确切的说，是 Sass 的一种句法——SCSS——兼容普通的 CSS 句法。

---

## 提示

如果你在命令行中看不到报错（醒目的红色和棕黄色文字），但是执行某些操作（例如修改样式表）之后，网站没有变化，首先应该想到重启服务器——大多数情况下，这么做都能解决问题。

---

## 7.6 文章列表

现在，页头和页脚有了，下面我们来开发“主栏”部分，也就是 `_layouts/default.html` 中 `{{ content }}` 表示的内容。如果不出意外，你在浏览器中看到的页面应该是首页。首页是一个网站的入口，这个页面的内容在根目录里的 `index.html` 文件中编写。

在文本编辑中打开这个文件，我们发现这个文件中已经存在一些内容。这些内容是创建网站时自动生成的，我们在这个基础上编写需要的内容。其实，现在的内容基本能满足需求，不过我们还要做些调整：

代码清单 7.8：修改 `index.html` 文件

```
<div class="home">

    <h1 class="page-heading">最新文章</h1>

    {% for post in site.posts %}
        <article class="post">
            <header class="post-header">
                <h1 class="post-title"><a href="{{ post.url | prepend: site.baseurl }}>{{ post.title }}</a></h1>
                <p class="post-meta">{{ post.date | date: "%Y年%m月%d日" }}</p>
            </header>

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

</div>
```

我们做了几处汉化，调整了显示文章列表的方式，还把 RSS 订阅的链接删掉了。在原先的内容中，文章的发布日期是英文格式，我们使用 `date` 过滤器将其改成了中国的习惯表示方法。

内容有了，但样式还不符合我们的要求。下面，我们来调整一下页面结构。在文本编辑器中打开 `_layouts/default.html` 文件，把内容改为：

代码清单 7.9：调整 `_layouts/default.html` 布局的结构

```
{% include header.html %}

<div class="content">
  <div class="container">
    <div class="row">
      <div class="main col-xs-12 col-md-8">
        {{ content }}
      </div>
    </div>
  </div>
</div>

{% include footer.html %}
```

其中，`container`、`row` 和几个 `col-*` 类都是 Bootstrap 提供的。现在首页显示了几篇文章，但还不够多，我们可以再创建几篇文章，看看整体的显示效果。如果对效果不满意，可以在 `css/main.scss` 中编写样式调整。下面列出我编写的一些样式，仅供参考：

代码清单 7.10：文章的 CSS 样式

```
...
.

.page-heading {
  font-size: 20px;
  margin-top: 0;
  margin-bottom: 30px;
}

.post {
  margin-bottom: 30px;
}
```

```
.post-header {  
    border-bottom: 1px solid #ccc;  
    margin-bottom: 10px;  
    padding-bottom: 10px;  
}  
  
.post-meta {  
    margin-bottom: 0;  
}  
  
.post-title {  
    font-size: 28px;  
    margin-top: 0;  
}  
  
.footer {  
    ...  
}
```

## 7.7 侧边栏

我们构思的主题（图 7.1）只剩下侧边栏没有实现，下面就来实现。

在文本编辑器中打开 `_layouts/default.html` 文件，添加下述代码清单中的第 10-12 行：

代码清单 7.11：修改 `_layouts/default.html` 布局，引入侧边栏

```
1      {% include header.html %}  
2  
3      <div class="content">  
4          <div class="container">  
5              <div class="row">  
6                  <div class="main col-xs-12 col-md-8">  
7                      {{ content }}  
8                  </div>  
9  
10                 <div class="sidebar col-xs-12 col-md-4">  
11                     {% include sidebar.html %}
```

```
12      </div>
13    </div>
14  </div>
15 </div>
16
17  {% include footer.html %}
```

注意，我们使用了`{% include sidebar.html %}`，也就是说，侧边栏要放在单独的文件`_includes/sidebar.html`中。现在这个文件还不存在，先新建，然后写入如下内容：

代码清单 7.12: `_includes/sidebar.html` 文件的内容

```
<aside class="about">
  <h3 class="aside-title">关于</h3>
  <div class="aside-content">
    <p>{{ site.description }}</p>
  </div>
</aside>

<aside class="subscribe">
  <h3 class="aside-title">订阅</h3>
  <div class="aside-content">
    <p>订阅 <a href="{{ "/feed.xml" | prepend: site.baseurl }}">RSS</a></p>
  </div>
</aside>
```

我们在侧边栏中放入了网站的介绍，以及原本在“正文”部分显示的 RSS 订阅链接。当然，这只是举个例子，你可以根据自己的需求在侧边栏中添加所需的内容。方法很简单，把内容写入`_includes/sidebar.html`文件即可。本书后面还会在侧边栏中添加分类列表等。

## 7.8 单篇文章页面的模板

至此，网站的整体布局基本完成，下面我们要编写单篇文章页面的模板。单篇文章页面的模板在`_layouts/post.html`文件中（因为我们在文章的元信息中把`layout`设为`post`）。

首先，在`_layouts`文件夹中新建`post.html`文件，然后写入如下内容：

代码清单 7.13: \_layouts/post.html 文件的内容

```
---
layout: default
---



< HEADER class="post-header">


# <a href="{{ page.url | prepend: site.baseurl }}>{{ page.title }}</a></h1> {{ page.date | date: "%Y年%m月%d日" }}</p> </header> <div class="post-content"> {{ content }} </div> </article>


```

注意，上述代码和 index.html 文章列表中的代码既有相同点也有不同点。首先，元信息表示这个模板继承 default.html。其次，现在文章的信息都通过 page 获取，这个模板变量在单篇文章和页面中都可用。最后，文章的内容直接通过 content 获取。

然后，在这段代码后面添加文章导航，也就是前一篇和后一篇文章的链接：

代码清单 7.14: 文章导航

```
{% if page.previous or page.next %}
<nav>



{% if page.previous %}
- &laquo; 前一篇</a>

{% endif %}

{% if page.next %}
- 后一篇 &raquo;</a>

{% endif %}


</nav>
```

```
</ul>
</nav>
{% endif %}
```

## 7.9 页面的模板

然后，按照前一节的方式编写页面的模板。页面的模板一般在 `_layouts/page.html` 文件中。首先，在 `_layouts` 文件夹中新建 `page.html` 文件，然后写入如下内容：

代码清单 7.15: `_layouts/page.html` 文件的内容

```
---
layout: default
---



< HEADER class="post-header">


# <a href="{{ page.url | prepend: site.baseurl }}>{{ page.title }}</a></h1>



{{ content }}


```

元信息表明，这个模板也继承 `default.html`。因为“页面”一般不关注时效性，所以没有显示发布时间。

## 7.10 404 页面

GitHub Pages 支持一个特殊的文件，在找不到内容时（也就是 404）显示。这个文件放在网站的根目录中，名为 `404.md`。下面我们来编写这个文件的内容：

代码清单 7.16: `404.md` 文件的内容

```
---
layout: page
title: 404 - 未找到
---
```

未找到你要查看的内容，请返回[主页]({{ site.baseurl }})。

在本地可以访问 <http://localhost:4000/404/> 查看这个页面的内容及样式。如果把网站部署到自己的服务器中，请参照 [11.2.1 节](#) 设置 .htaccess 文件，以便出现 404 错误时显示这个页面。

## 7.11 小结

本章介绍了如何从零开始创建一个简单的 Jekyll 主题，介绍了 Jekyll 的模板机制，Liquid 模板标签和常用模板的编写。现在，这个主题还略显简单，从下一章开始，我们会借助第三方服务和插件增强主题的功能。

---

# 第 8 章 添加评论系统

如果你用过其他博客系统，或者访问过别人的博客，一定会留意到，一般的博客中都有评论系统，允许访客对博主发布的文章发表意见，或附和或抨击。但 Jekyll 生成的是纯静态网站，无法像 WordPress 等程序那样使用数据库设计一套评论系统。不过，无须担心，有人已经为我们解决了这个问题——我们可以使用第三方评论系统。

第三方评论系统有很多，做得较早也较好的是 [Disqus](#)。国内也有模仿品，例如[多说](#)。这种系统会提供给你一段 JavaScript 代码，把它插入网站之后，就可以在自己的网站中加入评论系统。由此可以看出，在纯静态网站中使用这种评论系统最合适，因为静态网站并不限制使用 JavaScript。

本章会介绍如何在 Jekyll 网站中使用 Disqus 或多说实现评论功能。首先介绍 Disqus。

## 8.1 Disqus

在 Jekyll 网站中添加 Disqus 的步骤如下：

1. 如果还没有 Disqus 的账户，先[注册一个新账户](#)；
2. 注册完成后，[登录 Disqus](#)；
3. 点击头部导航右侧的齿轮图标，在下拉菜单中点击“Add Disqus to Site”；
4. 把打开的网页拉到底部，点击“GET STARTED”按钮；
5. 填写“Site name”（网站名）和“Disqus URL”，再选择网站的类别（“Category”），然后点击“Next”按钮，注册你的网站，如图 8.1 所示；
6. 在打开的页面中选择“Universal Code”（通用代码）。

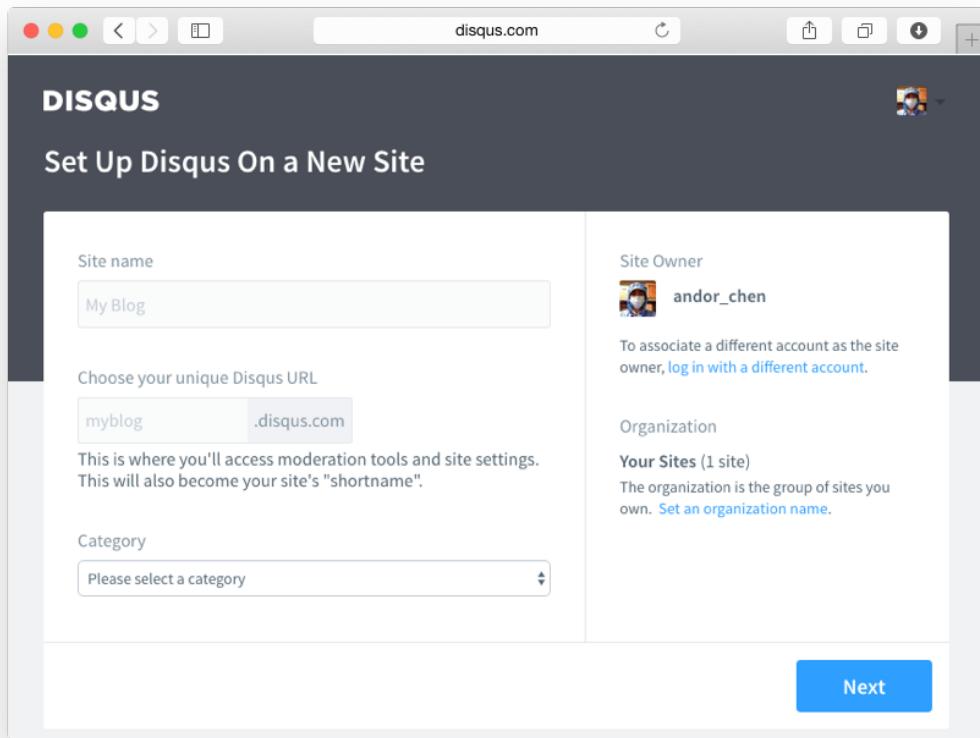


图 8.1：在 Disqus 中注册网站

现在会打开一个网页，说明如何把 Disqus 提供的 JavaScript 代码添加到网站中。我们来看一下如何把这段 JavaScript 代码添加到 Jekyll 网站中。

进入网站根目录中的 `_includes` 文件夹，新建一个文件，命名为 `disqus.html`。在文本编辑器中打开这个文件，复制 Disqus 提供的 JavaScript 代码，粘贴到这个文件中。Disqus 提供的 JavaScript 代码示例如下：

代码清单 8.1： Disqus 提供的 JavaScript 代码

```
1 <div id="disqus_thread"></div>
2 <script>
3 /*
```

```

* RECOMMENDED CONFIGURATION VARIABLES: EDIT AND UNCOMMENT THE SECTION BELOW
TO INSERT DYNAMIC VALUES FROM YOUR PLATFORM OR CMS.
* LEARN WHY DEFINING THESE VARIABLES IS IMPORTANT: https://disqus.com/admin/universalcode/#configuration-variables
4 */
5 /*
6 /*
7 var disqus_config = function () {
8     this.page.url = PAGE_URL; // Replace PAGE_URL with your page's canonical URL
9     variable
10    this.page.identifier = PAGE_IDENTIFIER; // Replace PAGE_IDENTIFIER with your
11    page's unique identifier variable
12 };
13 */
14 (function() { // DON'T EDIT BELOW THIS LINE
15     var d = document, s = d.createElement('script');
16
17     s.src = '//<example>.disqus.com/embed.js';
18
19     s.setAttribute('data-timestamp', +new Date());
20     (d.head || d.body).appendChild(s);
21 })();
22 </script>
<noscript>Please enable JavaScript to view the <a
href="https://disqus.com/?ref_noscript" rel="nofollow">comments powered by
Disqus.</a></noscript>

```

这是 Disqus 为我们提供的模板，我们要根据里面的说明，做几处修改。首先，去掉第 6 行和 13 行的注释；然后，把 PAGE\_URL 改成 '{{ page.url | prepend: site.baseurl | prepend: site.url }}'，把 PAGE\_IDENTIFIER 改成 '{{ page.id }}'。（注意，别忘了引号！）

如果你直接复制上述代码，除了 PAGE\_URL 和 PAGE\_IDENTIFIER 之外，还要修改第 17 行，改成前面填写的 Disqus URL。

然后在文本编辑器中打开 \_layouts/post.html，在最后一行插入 {% include disqus.html %}：

```

.....
</nav>
{% endif %}

```

```
{% include disqus.html %}
```

保存所有文件。执行 `bundle exec jekyll s` 命令，启动本地服务器，然后在浏览器中访问网站，随便打开一篇文章，你会看到正文下面显示了一个评论框（图 8.2），访客和你可以在这里输入评论——这就是 Disqus 为我们提供的评论系统。

如果想在页面中也加入评论功能，可以参照前面的方法修改 `_layouts/page.html` 文件。

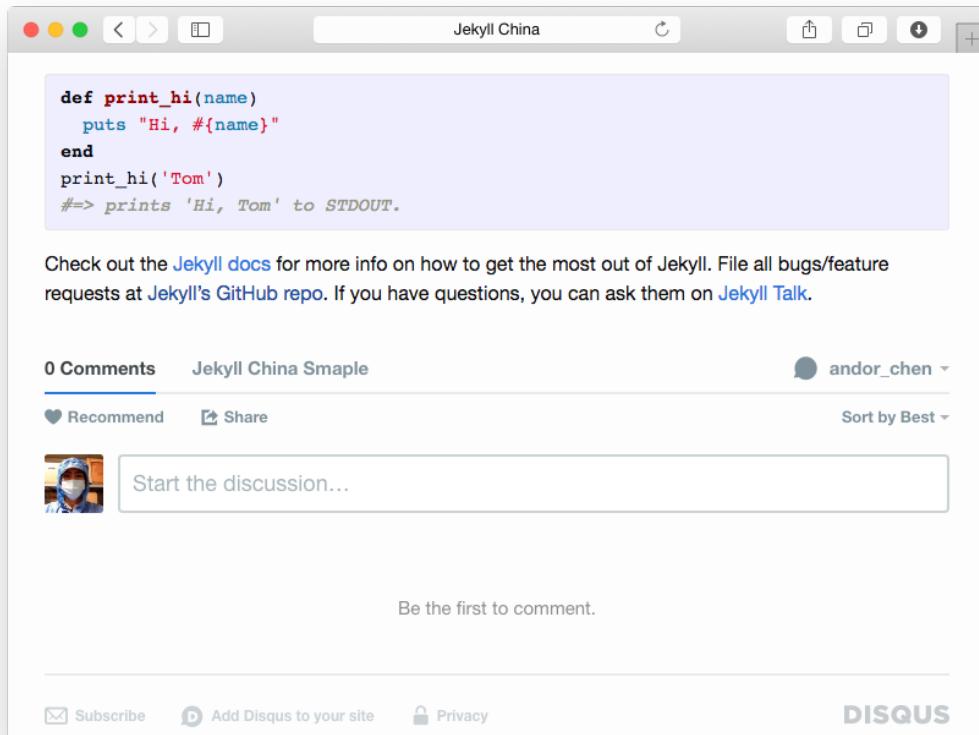


图 8.2：文章底部显示了 Disqus 的评论系统

## 8.2 多说

如果想使用多说提供的评论系统，方法与 Disqus 类似，具体而言有下面几步：

1. 使用第三方账户登录[多说](#);
2. 登录后点击首页中的“我要安装”按钮;
3. 填写信息，在多说中注册你的网站，填完后点击“创建”按钮;
4. 在打开的网页中，复制多说提供的“通用代码”。

进入网站根目录中的`_includes`文件夹，新建文件`duoshuo.html`。在文本编辑器中打开这个文件，粘贴多说提供的代码：

代码清单 8.2：多说提供的 JavaScript 代码

```
1  <!-- 多说评论框 start -->
2  <div class="ds-thread" data-thread-key="请将此处替换成文章在你的站点中的ID"
3      data-title="请替换成文章的标题" data-url="请替换成文章的网址"></div>
4  <!-- 多说评论框 end -->
5  <!-- 多说公共JS代码 start (一个网页只需插入一次) -->
6  <script type="text/javascript">
7  var duoshuoQuery = {short_name:"yousitename"};
8  (function() {
9      var ds = document.createElement('script');
10     ds.type = 'text/javascript';ds.async = true;
11     ds.src = (document.location.protocol == 'https:' ? 'https:' : 'http:') +
12             '//static.duoshuo.com/embed.js';
13     ds.charset = 'UTF-8';
14     (document.getElementsByTagName('head')[0]
15      || document.getElementsByTagName('body')[0]).appendChild(ds);
16 })();
17 </script>
18 <!-- 多说公共JS代码 end -->
```

根据这段代码中的注释，我们要修改几个属性的值——把第 2 行和 3 行替换成：

```
<div class="ds-thread"
    data-thread-key="{{ page.id }}"
    data-title="{{ page.title }}"
    data-url="{{ page.url | prepend: site.baseurl | prepend: site.url }}>
</div>
```

为了排版需要，我加入了换行。再把第 7 行中的`short_name` 的值设为你填写的多说域名。



图 8.3: 文章底部显示了多说的评论系统

然后，在文本编辑器中打开 `_layouts/post.html`，在最后一行插入 `{% include duoshuo.html %}`:

```
....  
</nav>  
{% endif %}  
  
{% include duoshuo.html %}
```

保存所有文件。执行 `bundle exec jekyll s` 命令，启动本地服务器，然后在浏览器中访问网站，随便打开一篇文章，你会看到正文下面显示了一个评论框（图 8.3），访客和你可以在这里输入这篇文章的评论——这就是多说为我们提供的评论系统。

如果想在页面中也加入评论功能，可以参照前面的方法修改 `_layouts/page.html` 文件。



---

# 第 9 章 Jekyll 的插件系统

任何程序提供的功能都不可能满足所有人的需求，因此，多数程序都会提供插件系统，让用户或其他开发者扩展原程序的功能。Jekyll 也不例外。Jekyll 的插件系统非常简单，但又足够强大。

Jekyll 的插件分为五类：生成器，转换器，子命令，标签和钩子。生成器用于生成页面，或者把信息插入页面；转换器用于转换某种书写语言；子命令用于向 `jekyll` 可执行文件添加命令；标签用于自定义模板标签和过滤器；钩子的作用是在 Jekyll 运行过程的特定时刻执行指定的操作。

若想编写插件，需要对 Jekyll 的内部运作机制和 Ruby 有一定的了解。因此，如果你对这些都不了解，知道怎么使用他人编写的插件就行了。

Jekyll 对插件的安装方式没做十分严格的限制，提供了三种方式。插件可以写入 `.rb` 文件，保存在根目录中的 `_plugins` 文件夹里，Jekyll 会自动加载这个文件夹中的所有 Ruby 文件。插件也可以打包成 `gem`，然后在 `_config.yml` 文件中的 `gems` 设置项中指定需要使用的插件名，Jekyll 会自动加载指定的插件。最后，`gem` 形式的插件还可以写入 `Gemfile` 中专门的 `jekyll_plugins` 分组，让 `Bundler` 自动安装和加载。先来看第一种形式。

---

## 注意

为了服务器的安全，GitHub Pages 不支持随意安装和使用插件。GitHub Pages 支持的插件参见[这个页面](#)。如果不想受此限制，可以先在本地生成网站，然后再上传到 GitHub Pages，方法参见[11.1.1 节](#)。

---

## 9.1 显示 Jekyll 版本信息

如果网站的根目录中没有 `_plugins` 文件夹，先创建。然后，在里面新建文件 `version_tag.rb`，写入如下代码：

### 代码清单 9.1：定义 `jekyll_version` 标签

```
module Jekyll
  class VersionTag < Liquid::Tag

    def render(context)
      "Jekyll #{Jekyll::VERSION}"
    end
  end
end

Liquid::Template.register_tag('jekyll_version', Jekyll::VersionTag)
```

我们自己编写并安装了一个插件。这是一个标签类插件，作用是显示 Jekyll 的版本信息。下面我们来看如何使用这个插件，以及它的效果。在文本编辑器中打开 `_includes/footer.html`，在版权信息后插入这个标签：

```
...
<p>&copy; {{ site.time | date: '%Y' }} {{ site.title }}</p>
<p>% jekyll_version %</p>
...
```

重启本地服务器，然后刷新网页，在页脚会看到显示了 Jekyll 的版本信息：“Jekyll 3.2.1”。

---

#### 注意

这里编写的 `jekyll_version` 标签只是做个演示，其实在模板中直接使用 `{{ jekyll.version }}` 就能获取 Jekyll 的版本号。

---

## 9.2 文章分页

当文章越来越多时，首页会变得越来越长，滚动条要拉动很久才能到达底部。我们都知道，一个页面不能显示过多的文章，一般最多显示 10 篇。Jekyll 和 WordPress 等程序一样，也支持分页显示文章。下面我们来看具体怎么做。

从 Jekyll 3.0 开始，分页功能从核心代码中移除，制成了插件——`jekyll-paginate`。所以，首先要安装这个插件。在 `Gemfile` 文件末尾中添加下述代码：

```
group :jekyll_plugins do
  gem 'jekyll-paginate'
end
```

然后在命令行中执行 `bundle install` 命令，安装 `jekyll-paginate` 插件。

接下来，我们要设置一页显示几篇文章。这个数量没有标准值，你可以根据需要，或者参考其他网站设定。这里，我们设为 10 篇。在文本编辑器中打开根目录中的 `_config.yml` 文件，把下面这个设置添加到文件末尾：

```
paginate: 10
```

然后修改 `index.html` 文件，把循环的开头改成：

```
{% for post in paginator.posts %}
```

注意，现在我们不再通过 `site` 变量获取网站中的文章，而是通过 `paginator` 变量获取。然后，在循环的后面添加分页链接：

#### 代码清单 9.2：分页链接

```
...
</article>
{% endfor %}

<nav>
  <ul class="pager">
    {% if paginator.previous_page %}
      <li class="previous">
        <a href="{{ paginator.previous_page_path | prepend: site.baseurl |
replace: '/index.html', '/' }}>
          &laquo; 前一页</a>
        </li>
    {% endif %}

    {% if paginator.next_page %}
      <li class="next">
        <a href="{{ paginator.next_page_path | prepend: site.baseurl }}>
          后一页 &raquo;</a>
        </li>
    {% endif %}
```

```
</ul>
</nav>

</div>
```

上述代码中的几个 CSS 类由 Bootstrap 提供，详情参见[文档](#)。注意，我们在“前一页”的链接中使用了 `replace` 过滤器，把 `/index.html` 替换成 `/`。因为默认生成的第一页链接后面带有 `index.html`，不雅观，所以我们使用 `replace` 过滤器把它去掉。

现在，如果文章的数量超过了所设的 `paginate` 值，文章列表的下面会显示一个精美的分页链接，如图 9.1 所示。

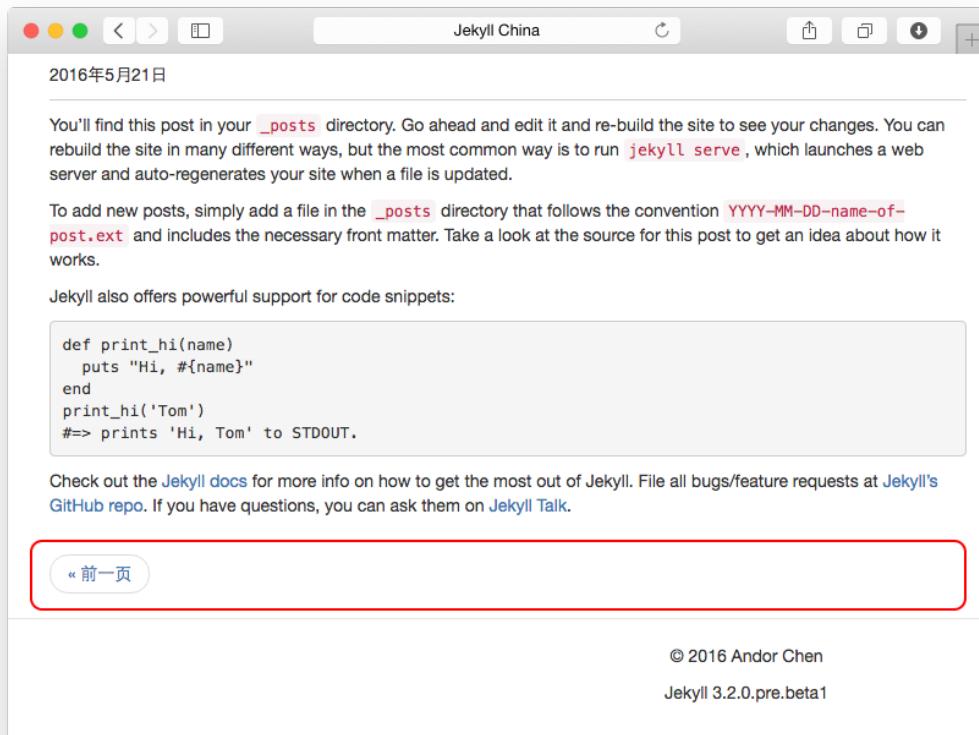


图 9.1：分页链接（图中红色圈出的部分）

## 9.3 生成网站地图

网站地图是一个 XML 文件（文件名一般是 `sitemap.xml`），使用特定的格式列出网站中的页面链接（不一定包含全部页面）。搜索引擎的爬虫往往会根据网站地图中的链接爬取网站的内容，因此网站地图有利于 SEO。

如果想生成网站地图，可以根据协议指定的规则手动编写，当然也可以使用现有的插件。我们要使用的是 Jekyll 团队开发的 [jekyll-sitemap 插件](#)。

首先，在 `Gemfile` 文件中添加 `gem 'jekyll-sitemap'`：

```
...
group :jekyll_plugins do
  gem 'jekyll-paginate'
  gem 'jekyll-sitemap'
end
```

然后，在命令行中执行 `bundle install` 命令。安装这个插件后，重新生成网站时，会自动在 `_site` 文件夹中生成 `sitemap.xml` 文件。

这个插件还提供有一个设置，如果不想要某篇文章或某个页面出现在网站地图中，可以在头部元信息中设置 `sitemap: false`。

前面几节介绍了安装插件的不同方式，而且也动手编写了一个插件。下面我们再来编写几个插件，增强前一章所开发主题的功能。

## 9.4 文章所属的分类

你可能还记得，我们在文章的元信息中设置了分类信息，例如在默认生成的第一篇文章中，分类信息是：`categories: jekyll update`。这一节，我们的目的是在文章标题下面显示文章的分类。为此，我们要编写一个插件。首先，在 `_plugins` 文件夹中新建 `category_links.rb` 文件，然后写入如下代码：

代码清单 9.3：定义 `category_links` 过滤器

```
module Jekyll
  module CategoryLinksFilter
    def category_links(categories)
      return '' if categories.empty?
```

```

    output = []
  categories.each do |cat|
    output << cat
  end
  configs = @context.registers[:site].config
  separator = configs['categories_seperator'] || ','

  output.join(separator)
end

end

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

```

上述代码定义的是一个标签类插件，严格来说定义的是过滤器，名为 `category_links`。在这段代码中，我们先检查文章有没有设置分类信息，如果没设置，直接返回空字符串。否则，迭代分类数组，获取全部分类名。注意，在 `category_links` 方法的最后，我们使用了一个自定义的设置——`categories_separator`，设置连接各个分类的字符，其默认值为 `' , '`。如果需要使用其他值，例如中文逗号，可以在 `_config.yml` 文件中设置：`categories_separator: ', '`。

写好这个插件后，接下来要修改 `post.html` 模板，在文章发布日期后加上分类信息：

```
<p class="post-meta">{{ page.date | date: "%Y年%-m月%-d日" }} {{ page.categories | category_links }}</p>
```

然后，修改 `index.html`，在文章列表中也显示分类信息：

```
<p class="post-meta">{{ post.date | date: "%Y年%-m月%-d日" }} {{ post.categories | category_links}}</p>
```

重启服务器后，就能看到，在文章标题下面，发布日期后面显示了分类，如下图所示（以默认生成的第一篇文章为例）。

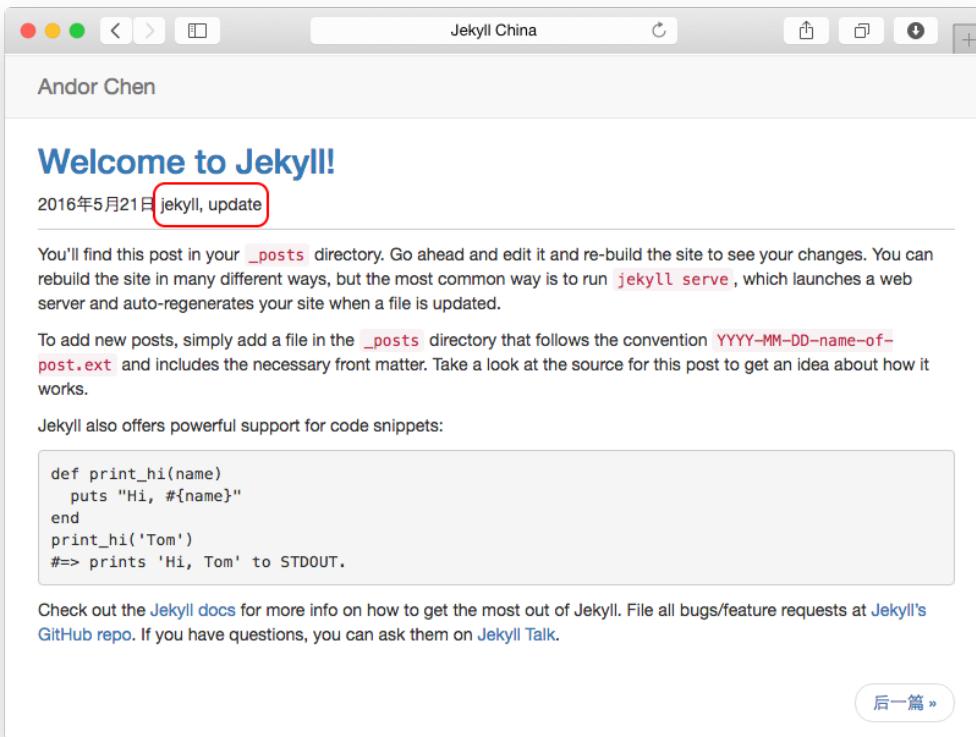


图 9.2: 文章的分类（图中红色圈出的部分）

## 9.5 生成归档页面

目前，我们的主题实现了基本功能，不过好像还少点儿什么。如果以前用过 WordPress 等程序，你应该知道，WordPress 会生成各种归档页面，例如分类归档、标签归档和年月日归档等。Jekyll 默认没有提供这种归档页面，不过我们可以使用插件生成。

我们要使用的插件是 `jekyll-archives`，这个插件由 Jekyll 团队开发，以 gem 的形式分发，安装和使用都很简单，把 `gem 'jekyll-archives'` 写入 `Gemfile` 文件中的 `jekyll_plugins` 分组，然后执行 `bundle install` 命令：

```
...
group :jekyll_plugins do
```

```
gem 'jekyll-paginate'  
gem 'jekyll-sitemap'  
gem 'jekyll-archives'  
end
```

然后，我们还要做些设置，告诉 `jekyll-archives` 生成哪些归档页面。在文本编辑器中打开 `_config.yml` 文件，在末尾添加如下设置：

```
jekyll-archives:  
  enabled: all
```

我们告诉 `jekyll-archives`，我们要生成所有归档页面（`all`），包括分类归档（`categories`）、标签归档（`tags`）、年份归档（`year`）、月份归档（`month`）和单日归档（`day`）。如果只想生成分类归档和年份归档，可以这样设置：

```
jekyll-archives:  
  enabled: ['categories', 'year']
```

现在，重启服务器（安装插件后都要重启）。我们看到命令行中有很多提醒：

```
Build Warning: Layout 'archive' requested in category/jekyll/index.html does not exist.  
Build Warning: Layout 'archive' requested in category/update/index.html does not exist.  
Build Warning: Layout 'archive' requested in 2016/index.html does not exist.  
....
```

这是因为归档页面使用的模板 `archive.html` 不存在。`jekyll-archives` 没有提供默认的 `archive.html` 模板，因此我们要自己编写。在 `_layouts` 文件夹中新建 `archive.html` 文件，写入如下代码：

代码清单 9.4: `archive.html` 布局的内容

```
---  
layout: default  
---  
{% if page.type == 'category' %}  
  {% capture page_title %}“{{ page.title }}”分类的归档{% endcapture %}  
{% elsif page.type == 'tag' %}  
  {% capture page_title %}“{{ page.title }}”标签的归档{% endcapture %}  
{% elsif page.type == 'year' %}  
  {% assign page_title = page.date | date: '%Y年' | append: '归档' %}
```

```

{% elif page.type == 'month' %}
  {% assign page_title = page.date | date: '%Y年%-m月' | append: '归档' %}
{% elif page.type == 'day' %}
  {% assign page_title = page.date | date: '%Y年%-m月%-d日' | append: '归档' %}
{% endif %}

<h1 class="page-heading">{{ page_title }}</h1>

<ul class="post-list">
  {% for post in page.posts %}
    <li>
      <span class="post-date">{{ post.date | date: "%Y年%-m月%-d日" }}</span>
      <a href="{{ post.url | prepend: site.baseurl }}">{{ post.title }}</a>
    </li>
  {% endfor %}
</ul>

```

这段代码的前半部分是一个较大的 `if` 语句，作用是根据归档的类型创建标题。后半部分遍历 `page.posts`，列出当前归档中的文章。我们没有显示文章的内容，也没有使用分页，因为 `jekyll-archives` 目前对此还欠缺支持。

### 9.5.1 重构 `archive.html` 模板

从上面的代码可以看出，`archive.html` 模板的前半部分都是用来创建页面标题的 `if` 语句，有点扰乱这个模板的主体逻辑。所以，这一节我们把这一部分抽出来，形成单独的逻辑部分。也借此再次说明如何开发插件。

我们要把这个插件放在 `_plugins` 文件夹中。首先，在 `_plugins` 文件夹中新建 `archive_page_title.rb` 文件，然后写入如下代码：

代码清单 9.5：定义 `archive_page_title` 标签

```

module Jekyll
  class ArchivePageTitleTag < Liquid::Tag
    ARCHIVE_PAGE_TYPES = %w(category tag year month day)

    def render(context)
      # 检查 jekyll-archives 插件是否已经安装
      unless Jekyll.const_defined?('Archives', false)
        return "" # 沉默是金
      end
    end
  end
end

```

```

end

page = context.environments.first['page']

# 检查当前页面是否为五个归档页面之一
page_type = page['type']
unless ARCHIVE_PAGE_TYPES.include? page_type
  return "" # 沉默是金
end

case page_type
when 'category'
  "#{page['title']}”分类的归档”
when 'tag'
  "#{page['title']}”标签的归档”
when 'year'
  page['date'].strftime('%Y年') << '归档'
when 'month'
  page['date'].strftime('%Y年%b月') << '归档'
when 'day'
  page['date'].strftime('%Y年%b月%d日') << '归档'
end

end
end
end

```

```
Liquid::Template.register_tag('archive_page_title', Jekyll::ArchivePageTitleTag)
```

从上述代码可以看出，这是个标签类插件，定义的模板标签是 `archive_page_title`。这个插件不仅把 `archive.html` 模板中的 `if` 语句提取出来了，还适当增强了功能。首先，我们声明一个数组常量 `ARCHIVE_PAGE_TYPES`，定义属于归档的五个页面类型。然后，在 `render` 方法中，满足特定的条件时才构建页面的标题。第一个条件是，已经安装 `jekyll-archives` 插件。因为只有安装了这个插件，才能生成归档页面，因而才能在归档页面中使用这个插件定义的模板标签。第二个条件是，使用这个标签的页面必须是五个归档页面之一。也就是说，在其他页面，例如首页，不能使用这个标签。

插件编写好之后，我们来修改 `archive.html` 模板：删掉整个 `if` 语句，然后把 `h1` 标签中的 `{{ page_title }}` 换成 `{% archive_page_title %}`。注意，这里必须使用 `{% %}`。因为 `{}{}` 相当于“变量”，而 `{% %}` 相当于“函数”。

## 9.5.2 添加分类链接

有分类归档之后，我们可以为文章标题下面显示的分类列表添加链接，指向各分类的归档页面。

打开 `_plugins/category_links.rb` 文件，在 `category_links` 方法之后添加一个方法，名为 `category_link`，如下所示：

```
...
end

def category_link(category, context)
  site = context.registers[:site]
  archive = Archives::Archive.new(site, category, 'category', [])
  url = site.config['baseurl'] + archive.url

  %(<a href="#{url}" title="#{category}">#{category}</a>
end

end
end

Liquid::Template.register_filter(Jekyll::CategoryLinksFilter)
```

然后修改 `category_links` 方法，改成下面这样：

```
...
def category_links(categories)
  return '' if categories.empty?

  output = []
  jekyll_archives_installed = Jekyll.const_defined?('Archives', false)
  categories.each do |cat|
    output << (jekyll_archives_installed ? category_link(cat, @context) : cat)
  end
  configs = @context.registers[:site].config
  separator = configs['categories_seperator'] || ', '

  output.join(separator)
end
...
```

在上述代码清单中，我们先检查文章有没有设置分类信息，如果没设置，直接返回空字符串。否则，迭代分类数组，如果安装了 `jekyll-archives` 插件，调用前面定义的 `category_link` 方法，显示指向各分类归档页面的链接；否则，只显示分类名。

这样修改之后，无需修改模板，因为我们使用的还是 `category_links` 过滤器。重启本地服务器，查看效果。

## 9.6 分类列表

现在，文章所属的分类能显示出来了，不过我们还想列出网站中的所有分类。网站中的所有分类可以使用 `site.categories` 获取，得到的是一个 Hash，键是分类名，值由属于该分类的文章对象组成。因此，我们要找到一种合适的方式处理这个 Hash，列出所有分类。如果可能的话（例如，安装了 `jekyll-archives` 插件），还要把各个分类链接到各自的归档页面。

为此，我们还要编写一个插件，这次定义的是一个标签。先在 `_plugins` 文件夹中新建 `categories_list.rb` 文件，然后写入以下代码：

代码清单 9.6：定义 `categories_list` 标签

```
require_relative 'category_links'

module Jekyll
  class CategoriesListTag < Liquid::Tag
    include CategoryLinksFilter

    def render(context)
      site = context.registers[:site]
      show_count = site.config['show_count_in_categories_list'] || true
      jekyll_archives_installed = Jekyll.const_defined?('Archives', false)

      output = '<ul>'
      site.categories.each do |cat, posts|
        output << '<li>'

        if jekyll_archives_installed
          output << category_link(cat, context)
        else
          output << cat
        end
      end
    end
  end
end
```

```

    if show_count
      output << " (#{posts.size})"
    end

    output << ''
end

output << ''
end

end
end

Liquid::Template.register_tag('categories_list', Jekyll::CategoriesListTag)

```

首先，我们加载同在 `_plugins` 文件夹中的 `category_links.rb` 文件，因为我们要使用这个文件中定义的 `category_link` 方法，创建指向分类归档页面的链接。在 `render` 方法中，我们遍历网站中的所有分类，然后在一个无序列表中列出这些分类。如果安装了 `jekyll-archives` 插件，还会把列表中的每个分类链接到各自的归档页面。除此之外，我们还使用了一个自定义的设置 —— `show_count_in_categories_list`，设置是否显示各分类中的文章数量（默认显示）。如果不显示数量，可以在 `_config.yml` 文件中把这个设置设为 `false`。

插件编写好之后，我们要修改模板，把分类列表显示出来。我们要在侧边栏中显示分类列表，所以要修改的文件是 `_includes/sidebar.html`。在这个文件中的两个 `aside` 标签之间添加如下代码：

代码清单 9.7：修改 `_includes/sidebar.html` 文件，显示分类列表

```

<aside class="categories_list">
  <h3 class="aside-title">分类</h3>
  <div class="aside-content">
    {% categories_list %}
  </div>
</aside>

```

然后重启本地服务器，刷新页面，此时就能看到侧边栏中显示的分类列表了，如图 9.3 所示。

## 关于

Write an awesome description for your new site here.  
You can edit this line in `_config.yml`. It will appear in your  
document head meta (for Google search results) and in  
your `feed.xml` site description.

## 分类

- [jekyll \(3\)](#)
- [update \(1\)](#)

图 9.3: 分类列表

## 9.7 钩子

Jekyll 3.0 新增了一种插件——钩子。借助钩子，我们可以在 Jekyll 运作过程的特定时刻执行指定的操作。Jekyll 支持[多个钩子](#)，这里我们举一个例子，说明如何使用 `site.post_write` 钩子。

为了减少网页的下载时间，提升用户体验，网站一般会简化（minify）HTML 页面，即把多余的空格去掉。由于 Liquid（Jekyll 使用的模板引擎）的局限，如果模板中有条件判断等语句，生成的 HTML 中会有大量空白，为了去掉这些多余的空白，我们可以对 Jekyll 生成的 HTML 做简化。

为此，我们要在 Jekyll 生成网站之后执行简化操作，此时就可以使用 `site.post_write` 钩子，在 Jekyll 把整个网站写入硬盘之后执行简化操作。

我们要使用的简化程序是 [html-minifier](#)。这是运行在 Node.js 平台中的程序，因此要先[安装 Node.js](#)。

---

### 说明

其实有个 Jekyll 插件也能简化 HTML——[octopress-minify-html](#)。这个插件纯粹使用 Ruby 实现，而且也利用了 Jekyll 钩子。但是，这个插件简化的结果不准确。

---

首先，在 Jekyll 网站的根目录中新建文件 `package.json`，写入一对空花括号 {}。然后，执行下述命令，安装 Gulp：<sup>1</sup>

```
$ npm install gulp --save-dev
```

然后，执行下述命令，安装 `gulp-htmlmin`：

```
$ npm install gulp-htmlmin --save-dev
```

---

### 提示

`npm` 安装的 Node.js 模块保存在当前目录中的 `node_modules` 文件夹里，这些模块都可以通过 `npm install` 命令重新安装，因此不用纳入版本控制系统。为了排除这个文件夹，要把 `node_modules` 添加到 `.gitignore` 文件中。

---

安装完成后，在 Jekyll 网站的根目录中新建 `gulpfile.js` 文件。这个文件用于存放 Gulp 任务，可以类比 `Rakefile` 文件理解。在文本编辑器中打开 `gulpfile.js` 文件，写入下述代码：

```
var gulp = require('gulp');
var minifyHTML = require('gulp-htmlmin');

const DEST = './_site';
const HTML_PATH = './_site/**/*.{html};
```

前两行代码引入 `gulp` 和 `gulp-htmlmin`。后两行代码定义两个常量，第一个常量是 Jekyll 生成的网站所在的目录，第二常量是 HTML 文件的通配模式，用于匹配 `_site` 文件夹中的全部 HTML 文件。

然后，定义一个 Gulp 任务，用于简化 HTML 文件。把下述代码添加到 `gulpfile.js` 文件中：

```
gulp.task('minifyHTML', function(){
  return gulp.src(HTML_PATH)
    .pipe(minifyHTML({collapseWhitespace: true}))
    .pipe(gulp.dest(DEST));
});
```

此时，我们可以执行 Gulp 命令，试试这个任务的效果：

```
$ gulp minifyHTML
```

---

1. Gulp 类似于 `rake`，是一种自动化构建工具，只不过运行在 Node.js 平台上。

---

## 注意

如果没有全局安装 Gulp (`npm install gulp -g`)，要执行 `./node_modules/.bin/gulp minifyHTML` 命令。

---

如果一切顺利，`_site` 文件夹中的所有 HTML 文件应该都简化了。但是现在有个问题：Jekyll 每次构建网站后都要自己动手输入上述命令。我们需要一种机制，每次构建网站后自动调用 `minifyHTML` 任务。这个需求有多种实现方法，例如使用 `browser-sync`。但是，这里方便起见，也为了介绍一种新的 Jekyll 插件机制，我们将使用钩子。

为了在构建网站之后执行某项操作，我们要使用 `site.post_write` 钩子。首先，在网站根目录中的 `_plugins` 文件夹里新建一个文件，将其命名为 `minify_html.rb`，写入下述 Ruby 代码：

```
Jekyll::Hooks.register :site, :post_write do |site|
  # 构建网站后简化 HTML 文件
  gulp = File.join(site.source, 'node_modules', '.bin', 'gulp')
  system "#{gulp} minifyHTML --silent"
end
```

保存文件后，重启 Jekyll 本地服务器。这时，再看 `_site` 文件夹里的 HTML 文件，应该都简化了。你还可以改动一个文件，例如某篇文章，再看看 HTML 文件有没有简化。

最后，我们要做一项优化。现在，不管修改哪个文件，`gulp-htmlmin` 会简化 `_site` 文件夹中的所有 HTML 文件。为了提高效率，如果只简化有变动的文件就好了。当然可以，我们需要借助一个 Gulp 插件——`gulp-changed-in-place`。

首先，安装这个插件：

```
$ npm install gulp-changed-in-place --save-dev
```

然后，编辑 `gulpfile.js` 文件，引入 `gulp-changed-in-place`：

```
var gulp = require('gulp');
var minifyHTML = require('gulp-htmlmin');
var changedInPlace = require('gulp-changed-in-place');
...

```

最后，编辑 `minifyHTML` 任务，在简化 HTML 文件之前调用 `changedInPlace()`：

```
...
gulp.task('minifyHTML', function(){

```

```
return gulp.src(HTML_PATH)
  .pipe(changedInPlace({firstPass: true}))
  .pipe(minifyHTML({collapseWhitespace: true}))
  .pipe(gulp.dest(DEST));
});
```

这样，Jekyll 每次构建之后都会自动调用 `minifyHTML` 任务，而且只简化有变动的 HTML 文件。

---

#### 提示

别忘了把 `package.json`、`gulpfile.js` 和 `node_modules` 添加到 `exclude` 设置中，否则它们会出现在生成的网站中。

```
exclude: ["Gemfile", "Gemfile.lock", "Rakefile", "package.json",
  "gulpfile.js", "node_modules"]
```

---

## 9.8 寻找插件

本章我们自己动手编写了几个插件，看起来都不错，既熟悉了 Jekyll 的插件系统，又增强了我们自己开发的主题的功能。不过，现实中，Jekyll 用户可能不具备自己开发插件的知识，或者有基础，但不想重复造轮子。这时，使用他人开发好的插件最合适了。可是到哪寻找插件呢？

首先，[Jekyll 文档](#)中列出了一些插件，可以根据需要选择使用。其次，可以到社区维护的 [Jekyll-Plugins](#) 网站中寻找。这个网站所含的插件比较全面，而且搜索方便。

找到合适的插件后，使用前面说明的方式安装即可。



---

# 第 10 章 数据文件

Jekyll 还有一个很强大的功能，数据文件。这个术语由两部分组成，一是“数据”，二是“文件”。我们使用这个功能存放的是“数据”，而存储介质是“文件”。如果是在传统的动态博客中，这些数据可能存放在数据库中。但 Jekyll 是静态网站生成工具，没有数据库，所以要把“数据”放在这些文件中。

通过数据文件，我们可以为网站提供一些额外的信息，或者提供一些数据让 Jekyll 生成网页。你可能会想，网站所需的数据放在文章、页面或者 `_config.yml` 文件中不行吗，为什么要放在单独的“数据文件”中？放在单独的文件中，自然有好处。其一，有助于重用数据。需要多次使用的数据不必每次都自己动手插入文章或页面。其二，内容和表现分离。我们存储在数据文件中的仅仅是“数据”而已，至于以什么方式把这些数据显示出来，显示成什么样子，那就是模板的事了。其三，不放在 `_config.yml` 文件中，是为了避免把这个文件的内容搅乱，毕竟正如这个文件的文件名所示，它是用来保存“设置”的。

数据文件要放在单独的文件夹中，Jekyll 生成网站骨架时没有生成这个文件夹，因此我们要自己动手创建。这个文件夹是根目录中的 `_data`（注意，前面有下划线）。下面举几个实例，说明如何使用数据文件。

## 10.1 分类的名称

前面，我们在文章的标题下面显示了所属的分类。不过，在使用的过程中你可能会发现有几个问题。第一，不管在文章的元信息中设定的分类是大写还是小写，最终显示出来的都是全部小写形式。第二，如果分类名是中文，无法正确生成分类的归档页面（所有分类名为中文的分类归档页面都指向同一个页面）。为了解决这些问题，我们可以使用数据文件，提供一些关于分类的额外信息。

首先，在 `_data` 文件夹中新建文件 `categories.yml`。Jekyll 支持的数据文件类型包括 [YAML](#)、[JSON](#) 和 [CSV](#)，不过 Ruby 社区都喜欢使用 YAML。然后，在文本编辑器中打开这个文件，写入一些关于分类的信息，例如：

### 代码清单 10.1：分类的信息

```
- slug: jekyll  
name: Jekyll  
  
- slug: update  
name: 更新
```

我们写入的两个条目对应于默认生成的第一篇文章的两个分类，仅作示例。其中，`slug` 表示分类的别名，只能使用英文字母，而且单词之间要使用连字符（-）连接；`name` 表示分类的名称，可以使用非 ASCII 字符，例如中文。有了这些信息之后，我们还要修改显示分类的方式，即 [9.4 节](#) 编写的 `category_links` 过滤器。在文本编辑器中打开 `_plugins/category_links.rb`，删除全部内容，替换成下述代码：

### 代码清单 10.2：显示分类“真正的”名称

```
module Jekyll  
  module CategoryLinksFilter  
    def category_links(categories)  
      return '' if categories.empty?  
  
      output = []  
      jekyll_archives_installed = Jekyll.const_defined?('Archives', false)  
      categories.each do |cat|  
        cat_data = category_data(cat, @context)  
        output << (jekyll_archives_installed ? category_link(cat_data, @context)  
                  : cat_data['name'])  
    end  
    configs = @context.registers[:site].config  
    separator = configs['categories_seperator'] || ','  
  
    output.join(separator)  
  end  
  
  def category_link(category_data, context)  
    site = context.registers[:site]  
    archive = Archives::Archive.new(site, category_data['slug'], 'category', [])  
    url = site.config['baseurl'] + archive.url  
  
    %(<a href="#{url}"  
      title="#{category_data['name']}">#{category_data['name']})  
  end  
end
```

```

    end

    def category_data(category, context)
      categories_data = context.registers[:site].data['categories']
      slugs = categories_data.map { |e| e['slug'] }

      return {'slug' => category, 'name' => category } unless
      slugs.include?(category)

      categories_data.select { |e| e['slug'] == category }.first
    end

  end
end

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

```

在上述代码中，我们对 `category_links` 和 `category_link` 两个方法做了一些修改，这些改动都是为了配合新添加的 `category_data` 方法。在 `category_data` 方法中，我们先读取 `_data/categories.yml` 中的数据，然后获取一个由分类别名组成的数据。得到别名数组后，我们判断这个数据文件中是否有指定分类的数据：如果没有就新构建一个，把别名和名称都设为传入的分类名；如果有，就返回相应的数据。

此时，如果重新构建网站，会发现侧边栏中的分类列表没有名称了，只有数字。这是因为侧边栏中显示的分类列表用到了 `category_link` 方法，而我们修改了这个方法。所以，我们还要相应地修改 `categories_list` 标签。在文本编辑器中打开 `_plugins/categories_list.rb`，按照下述方式修改：

```

...
  site.categories.each do |cat, posts|
    cat_data = category_data(cat, context)
    output << '<li>

    if jekyll_archives_installed
      output << category_link(cat_data, context)
    else
      output << cat_data['name']
    end
  end

```

```
if show_count  
...  
...
```

保存后再启动服务器就一切正常了。可以看到，文章标题下面和侧边栏中的分类列表中都显示了分类“真正的”名称，如图 10.1 所示。



图 10.1：分类“真正的”名称

### 提示

以后发布新文章时要这么做：在文章的元信息中使用“别名”设定分类，然后在 `_data/categories.yml` 文件中添加新条目，分别写入“别名”和对应的“名称”。当然，如果所用的别名已经存在，就可以省略这一步。

不过，现在还有一个问题。如果点击“更新”打开这个分类的归档页面，你会发现标题中显示的还是“update”。有了前面的基础，我想你应该知道如何显示“真正的”分类名了。提示：修改 `archive_page_title` 标签。下面给出具体的修改方法。在文本编辑器中打开 `_plugins/archive_page_title.rb`。首先，在第一行前面插入一行，写入：`require_relative 'category_links'`。然后，在 `class ArchivePageTitleTag < Liquid::Tag` 之后新起一行，添加 `include CategoryLinksFilter`。最后，修改 `render` 方法中的 `case` 语句，把 `'category'` 分支改为：“`#{category_data(page['title'], context)['name']}`”分类的归档”。保存文件后，重启服务器，就能看到分类归档页面的标题中显示了分类“真正的”名称。

如果你的网站使用标签（tag）的话，可以按照上述方式设置各个标签的“别名”和“名称”。

## 10.2 作者的信息

这一节再举个例子，说明如何使用数据文件。假设你的博客是团队的博客，有多位作者，你想在文章标题下显示作者的名称，而且想在文章的内容下面显示作者的信息。此时，我们可以把各个作者的信息保存在数据文件中，然后在 `post.html` 模板中显示出来。

Jekyll 的数据文件除了可以放在单个文件中之外，还可以组织在文件夹中。这次，我们要把各个作者的信息放入单独的文件夹里。首先，在 `_data` 文件夹中新建子文件夹，名为 `authors`，然后再新建多个文件，一个文件对应一个作者。假如我们的博客有两位作者：张三和李四。那么，我们要在 `authors` 子文件夹中新建两个文件，分别命名为 `zhangsan.yml` 和 `lisi.yml`。然后，在这两个文件中写入下述内容：

代码清单 10.3：张三的信息 (`zhangsan.yml`)

```
name: 张三
email: zhangsan@example.com
bio: 团队 CEO，负责团队管理。他是一名 Ruby 开发者，开发了很多程序。
```

代码清单 10.4：李四的信息 (`lisi.yml`)

```
name: 李四
email: lisi@example.com
bio: 团队 CTO，负责团队的技术指导。他是一名 Ruby 开发者，开发了很多程序。
```

现在，作者的信息有了，我们还要指定文章的作者。文章的作者在文章的元信息中指定，例如，在默认生成的第一篇文章中把作者设为“张三”：

代码清单 10.5：把文章的作者设为“张三”

```
---
layout: post
title: "Welcome to Jekyll!"
date: 2016-05-21 15:32:06 +0800
categories: jekyll update
author: zhangsan
---
```

注意，`author` 的值不能直接写成“张三”，而要使用数据文件的文件名，这样在代码中才能读取“张三”的信息。下面，我们要修改 `post.html` 模板，在文章标题下面和文章内容下部显示作者的信息。

代码清单 10.6：显示作者的信息

```
---
layout: default
---
{% if page.author %}
  {% assign the_author = site.data.authors[page.author] %}
```

```

{%- endif %}

<article class="post">
  <header class="post-header">
    <h1 class="post-title"><a href="{{ page.url | prepend: site.baseurl }}>{{ page.title }}</a></h1>
    <p class="post-meta">
      {% if the_author %}{% the_author.name %}{% endif %}
      {{ page.date | date: "%Y年%m月%d日" }}
      {{ page.categories | category_links}}
    </p>
  </header>

  <div class="post-content">
    {{ content }}
  </div>
</article>

{% if the_author %}
  <div class="alert alert-success author-info">
    <h5>{{ the_author.name }}</h5>
    <p>{{ the_author.bio }}</p>
  </div>
{% endif %}

{% if page.previous or page.next %}
  ....

```

保存这个模板后，在浏览器中访问默认生成的第一篇文章会看到标题下面和内容下部显示了作者的信息。如果你还想在文章列表中显示作者信息，例如首页，可以参照上述代码，修改 index.html 文件：

```

...
<h1 class="page-heading">最新文章</h1>

{% for post in paginator.posts %}
  <article class="post">
    <header class="post-header">
      <h1 class="post-title"><a href="{{ post.url | prepend: site.baseurl }}>{{ post.title }}</a></h1>
```

```
<p class="post-meta">
  {% if post.author %}{{ site.data.authors[post.author].name }}{% endif %}
  {{ post.date | date: "%Y年%m月%d日" }}
  {{ post.categories | category_links}}
</p>
</header>

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

<nav>
...

```



---

# 第 11 章 把网站部署到服务器

目前为止，我们都在本地服务器中查看（预览）网站。我们搭建的网站总是要面向公众的，所以，要把网站部署到服务器上，让其他人访问。

Jekyll 生成的是纯静态网站，因此选择服务器时有很多优势。可以说，几乎所有服务器都能伺服 Jekyll 生成的网站。除了自己购买服务器之外，我们也可以使用托管服务。比如，GitHub 就提供了这类服务，叫 GitHub Pages。

本章介绍如何把 Jekyll 生成的网站部署到线上环境，包括 GitHub Pages 和自己购买的虚拟主机。

---

## 注意

本章的操作需要用到 Git，如果你的电脑中还没安装，请从官网[下载](#)并安装 Git。如果你不知道如何使用 Git，请先阅读《[精通 Git](#)》。

---

## 11.1 GitHub Pages

若想把网站部署到 GitHub Pages，首先要注册一个 GitHub 账户，免费套餐即可。然后按照[说明](#)，把 SSH 密钥添加到 GitHub。如果使用 GitHub 客户端<sup>1</sup>则不需要做这一步。

登录后，在首页的右上角点击“+”号，在下拉菜单中选择“New repository”（新建仓库）。在出现的页面中填写仓库信息：只有“Repository name”（仓库名）是必填项，其他都可以不填，或者保持默认值。假设我们填写的仓库名是“blog”。填完后，点击“Create repository”（创建仓库）按钮。现在会显示这个仓库的页面（图 11.1），其中有些说明，介绍接下来该做什么。你可以看一下这些说明，如果不理解也没关系，下面我会详细说明每一步。

---

1. GitHub 客户端 Mac 版：<https://mac.github.com/>。Windows 版：<https://windows.github.com/>。

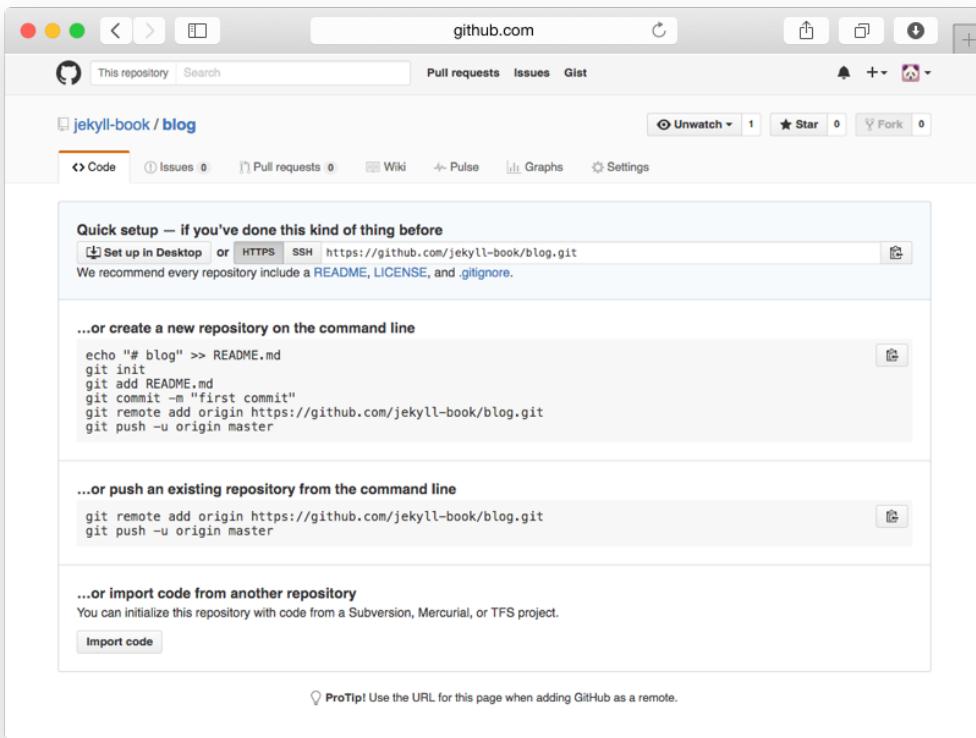


图 11.1：接下来做什么

现在回到本地，进入网站根目录，在命令行中执行下述命令，初始化 Git 仓库：

```
~/blog $ git init  
Initialized empty Git repository in ~/blog/.git/
```

然后在 GitHub 的仓库页面找到仓库的地址，例如 <https://github.com/<username>/blog.git>，回到命令行，添加这个远程仓库：

```
# 把 <username> 换成你的用户名  
~/blog (master) $ git remote add origin https://github.com/<username>/blog.git
```

然后添加根目录中的所有文件和文件夹<sup>2</sup>，做首次提交：

```
~/blog (master) $ git add -A  
~/blog (master) $ git commit -m "First commit"
```

现在我们要新建一个分支，名为 `gh-pages`:

```
# 新建并切换到 gh-pages 分支  
~/blog (master) $ git checkout -b gh-pages  
Switched to a new branch 'gh-pages'
```

现在我们已经身处 `gh-pages` 分支中，可以把本地仓库推送到 GitHub 的远程仓库了：

```
~/blog (gh-pages) $ git push origin gh-pages  
Counting objects: 27, done.  
Delta compression using up to 4 threads.  
Compressing objects: 100% (24/24), done.  
Writing objects: 100% (27/27), 10.00 KiB | 0 bytes/s, done.  
Total 27 (delta 1), reused 0 (delta 0)  
To git@github.com:<username>/blog.git  
 * [new branch]      gh-pages -> gh-pages
```

几乎立刻，你会收到一封 GitHub 发来的电子邮件，告诉你构建网站失败了，而且还指出了原因：

```
The tag `categories_list` on line 11 in `_includes/sidebar.html` is not a  
recognized Liquid tag. For more information, see https://help.github.com/articles/  
page-build-failed-unknown-tag-error.
```

我们直接把 Jekyll 网站推送到 GitHub，让 GitHub 为我们生成网站，然后再伺服，让用户访问。但是，这种方式有个最大的缺点：严重依赖 GitHub 服务器中的 Jekyll。GitHub 服务器中安装的 Jekyll 可能不是最新版，如果我们想使用新版中提供的新功能怎么办？GitHub 为了安全考虑，对服务器中安装的 Jekyll 做了限制，不能自由使用插件（[支持的数量有限](#)），如果某项功能必须使用插件实现怎么办？这里之所以构建失败，就是因为我们使用了自己编写的插件。

我们可以使用一种方法，一劳永逸地解决这些问题——把本地生成好的网站推送给 GitHub。这样我们就完全不依赖 GitHub 服务器中安装的 Jekyll 了，想使用哪个版本就使用哪个版本，想使用什么插件就使用插件，再也不用“看别人脸色”了。那么具体应该怎么做呢？

---

2. 其实不是所有文件，`_site` 文件夹中的文件不会添加。因为我们在 `.gitignore` 文件中排除了 `_site` 文件夹。`.gitignore` 文件的作用及用法参见[这里](#)。

其实很简单，想办法把 `_site` 文件夹中的内容推送到 GitHub 中即可。在这个过程中，可能有些步骤难以理解，我会尽量说得详细一点。如果读一遍不能理解，那就多读几遍。<sup>3</sup>

### 11.1.1 最佳实践

前面已经推送到远程 `gh-pages` 分支了，那我们就把它删除，同时也把本地的 `gh-pages` 分支删除：

```
# 切换到 master 分支
~/blog (gh-pages) $ git checkout master

# 删除本地 gh-pages 分支
~/blog (master) $ git branch -d gh-pages

# 删除远程 gh-pages 分支
~/blog (master) $ git push origin :gh-pages
```

然后，进入进入 `_site` 文件夹，初始化一个 Git 仓库：

```
~/blog (master) $ cd _site
~/blog/_site $ git init
```

然后，添加远程仓库：

```
~/blog/_site (master) $ git remote add origin https://github.com/<username>/blog.git
```

注意，`_site` 文件夹中的仓库与上一级目录中的不同，因此要重新添加远程仓库。接下来，创建并切换到 `gh-pages` 分支：

```
~/blog/_site (master) $ git checkout -b gh-pages
Switched to a new branch 'gh-pages'
```

现在，回到网站的根目录，重新生成网站：

```
~/blog/_site (gh-pages) $ cd ..
~/blog (master) $ bundle exec jekyll build
```

---

<sup>3</sup>. 古人云：书读百遍，其义自现。如果读了“一百遍”还是不能理解，写封信告诉我吧，我的电子邮箱是 [andor.chen.27@gmail.com](mailto:andor.chen.27@gmail.com)。

如果还想在本地预览，第二个命令可以换成 `jekyll s`。现在打开 `_site` 文件夹，你应该会看到生成的网站。接下来，我们要把生成的内容推送到 GitHub 仓库的 `gh-pages` 分支中：

```
~/blog (master) $ cd _site
~/blog/_site (gh-pages) $ git add -A
~/blog/_site (gh-pages) $ git commit -m "First deploy"
~/blog/_site (gh-pages) $ git push origin gh-pages # 注意，不是 master 分支
```

等待推送完毕。稍等片刻之后，你就应该可以通过 `http://<username>.github.io/blog/` 访问网站了。

不过，似乎有点问题——网页与本地看到的效果不同。这是什么原因呢？打开浏览器中的控制台，我们发现，是因为 CSS 样式表没加载。可是我们已经把样式表推送到 GitHub 仓库中了，为什么无法加载呢？这涉及到“路径”的问题。

从控制台中显示的结果可以看出，现在请求的 CSS 地址是 `http://<username>.github.io/css/main.css`。但我们的网站地址是 `http://<username>.github.io/blog/`。如果把这个地址理解为文件夹路径的话，`css/main.css` 这个文件在 `blog` 文件夹中，所以地址应该是 `http://<username>.github.io/blog/css/main.css`。那么为什么会请求 `http://<username>.github.io/css/main.css` 呢？

在文本编辑器中打开 `_includes/header.html` 文件，找到引入样式的元素：

```
<link href="{{ site.baseurl}}/css/main.css" rel="stylesheet">
```

其中，`href` 属性的值，也就是 CSS 文件的路径，用到了一个 Liquid 标签。`{{ site.baseurl }}` 的作用是读取 `_config.yml` 文件中设置的 `baseurl`。在文本编辑器中打开 `_config.yml`，可以看到，现在 `baseurl` 的值是空字符串，所以 `site.baseurl` 标签在 `/css/main.css` 前面只添加了空字符串，相当于什么也没加。我们要把 `baseurl` 设为 `/blog`（前面有斜线，后面没有）。然后，在命令行中执行 `bundle exec jekyll s` 命令，重新生成网站（还记得吗，每次修改 `_config.yml` 文件中的设置都要重新生成网站）：

```
~/blog (master) $ bundle exec jekyll s
Configuration file: ~/blog/_config.yml
      Source: ~/blog
    Destination: ~/blog/_site
Incremental build: disabled. Enable with --incremental
   Generating...
          done in 0.672 seconds.
Auto-regeneration: enabled for '~/blog'
Configuration file: ~/blog/_config.yml
```

```
Server address: http://127.0.0.1:4000/blog/  
Server running... press ctrl-c to stop.
```

注意看“Server address”（服务器地址）。因为我们修改了 `baseurl` 的值，所以现在在本地要访问的地址是 `http://localhost:4000/blog/`。在浏览器中访问这个地址——显示一切正常，样式表可以正常加载。

接下来，我们提交这次改动，然后推送到 GitHub 中：

```
~/blog (master) $ git add .  
~/blog (master) $ git commit -m "Change baseurl"  
~/blog (master) $ bundle exec jekyll build  
...  
~/blog (master) $ cd _site  
~/blog/_site (gh-pages) $ git add .  
~/blog/_site (gh-pages) $ git commit -m "Fix stylesheet url"  
~/blog/_site (gh-pages) $ git push origin gh-pages
```

稍等片刻，再次访问 `http://<username>.github.io/blog/`，现在网站的外观正常了。

---

### 提示

什么？网站还是老样子，外观并不正常？你可以清除浏览器的缓存后再试试。如果清除缓存也不行，那就多等一会儿。等得花儿都谢了，还是没变化？打开仓库的设置页面（在 GitHub 仓库页面的右边栏中点击“Settings”），拉倒底部，找到“GitHub Pages”区域，如果看到醒目的红色提醒，说明 GitHub 没有成功生成你的网站。这时，你可以回到本地，随便修改一下网站（例如某篇文章），然后再推送一次。一般情况下，重新推送都能解决这种问题。如果红色提醒还在，可以[联系 GitHub 的人工客服](#)，寻求帮助。别害羞，他们都很友好。<sup>4</sup>

---

其实，现在 GitHub 服务器中的 Jekyll 还会处理一遍 `gh-pages` 分支中的内容。我们可以进一步设置，完全不让它处理。在文本编辑器中打开 `_config.yml` 文件，添加下述设置：

```
keep_files: [".git", ".nojekyll"]
```

`keep_files` 设置的作用是指定生成网站时不清除 `_site` 文件夹中的某些文件（夹）。Jekyll 默认不清除保存 Git 仓库数据的 `.git` 文件夹，我们又添加了一个特殊的文件——文件名以点号开头的 `.nojekyll`。

---

4. 我联系过 GitHub 的客服，所以知道他们很友好。

然后，在`_site`文件夹中新建文件`.nojekyll`（以点号开头），什么内容都不用写入。这个文件的作用是告诉GitHub Pages，这个网站不需要使用Jekyll处理，“别来烦我”。最后，推送到GitHub仓库的`gh-pages`分支：

```
~/blog/_site (gh-pages) $ git add .nojekyll  
~/blog/_site (gh-pages) $ git commit -m "Add nojekyll"  
~/blog/_site (gh-pages) $ git push origin gh-pages # 注意，不是 master 分支
```

过程有点儿艰难，读一遍你可能无法完全理解，那就多读几遍吧。

太棒了，我们的网站终于上线了，赶快把网站的地址告诉朋友吧，和他们一起分享这份喜悦！

### 11.1.2 账户的网站

上述这种网站可以叫做“项目的网站”，因为它是针对某一个项目的。GitHub Pages还有另一种使用方式——搭建“账户的网站”。我们可以把“账户的网站”理解为“家目录”，而“项目的网站”相当于家目录中的一个文件夹。只不过这个“家”不在我们自己的电脑中，而在GitHub中。

账户的网站和项目的网站有几点区别，我们先看一下（表11.1）。

表11.1：GitHub Pages账户的网站和项目的网站对比

	账户的网站	项目的网站
仓库名	<code>&lt;username&gt;.github.io</code>	随意
网站地址	<code>http://&lt;username&gt;.github.io</code>	<code>http://&lt;username&gt;.github.io/&lt;repo-name&gt;/</code>
网站所在的分支	<code>master</code>	<code>gh-pages</code>

下面来看一下如何创建账户的网站。我们再新建一个Jekyll项目，保存在家目录的`homepage`文件夹中：

```
$ cd ~  
~ $ jekyll _3.2.1_ new homepage  
New jekyll site installed in ~/homepage.
```

在GitHub中再新建一个仓库。不过要注意，这一次仓库名要填写`<username>.github.io`（“`<username>`”换成你的用户名）。然后回到本地，按照前面介绍的方法，在命令行中初始化仓库、添加远程仓库，最后再推送：

```
~/homepage $ git init
~/homepage (master) $ git add remote origin
https://github.com/<username>/<username>.github.io.git
~/homepage (master) $ git add -A
~/homepage (master) $ git commit -m "First commit"

~/homepage (master) $ git checkout -b source # 切换到 source 分支
~/homepage (source) $ git branch -d master # 删除 master 分支
~/homepage (source) $ git push origin source # 注意, 不是 master 分支

~/homepage (source) $ bundle exec jekyll build
~/homepage (source) $ cd _site
~/homepage/_site $ git init
~/homepage/_site (master) $ git add remote origin
https://github.com/<username>/<username>.github.io.git
~/homepage/_site (master) $ git add .
~/homepage/_site (master) $ git commit -m 'First deploy'
~/homepage/_site (master) $ git push origin master # 注意, 不是 gh-pages 分支
```

可以看出，步骤跟前面差不多。根目录所在的 Git 仓库要使用 `source` 分支，这是因为账户的网站必须使用 `master` 分支。因此，在 `_site` 文件夹中，我们不用再切换到 `gh-pages` 分支，直接使用默认的 `master` 分支即可。此外，账户的网站也不用在 `_config.yml` 文件中修改 `baseurl` 的值，使用默认的空字符串即可。

稍等片刻之后，应该就可以通过 `http://<username>.github.io` 访问你的网站了。如果无法访问，可以参照前面介绍的方法解决。

### 在本地模拟 GitHub Pages 环境

GitHub Pages 使用的是特定版本的 Jekyll，此外，基于安全的考量，GitHub Pages 禁止随意使用插件，只允许使用[白名单内的插件](#)。之前查看 `Gemfile` 文件时，你可能还记得里面有这么一段注释：

```
# If you want to use GitHub Pages, remove the "gem "jekyll"" above and
# uncomment the line below. To upgrade, run `bundle update github-pages`.
# gem "github-pages", group: :jekyll_plugins
```

GitHub Pages 服务器中的环境就是使用 `github-pages` 这个 gem 搭建的，因此如果想在本地模拟 GitHub Pages 环境，可以按照上述说明，把 `gem "jekyll", "3.2.1"` 删掉，再把 `gem "github-pages", group: :jekyll_plugins` 前面的注释去掉，然后执行 `bundle install` 命令，安装 `github-pages` 及其依赖。

`github-pages` 除了用于搭建环境之外，还提供了命令行界面。例如，执行 `github-pages versions` 命令，打印出 GitHub Pages 所用的各个依赖及其版本；执行 `github-pages health-check` 命令，检查域名绑定有无问题。

### 11.1.3 绑定域名

现在我们的网站使用的是 GitHub 免费为我们提供的二级域名，例如 `http://<username>.github.io`。如果我们想使用自己的域名，GitHub Pages 也支持。

我们以账户的网站为例。在网站的根目录中新建一个文件，命名为 `CNAME`。在文本编辑器中打开这个文件，写入你想使用的域名，例如：

```
example.com
```

这里我们绑定的是“裸域”。如果用户访问 `http://www.example.com`，GitHub 会把它自动转向 `http://example.com`。类似地，如果你绑定的是 `www.example.com`，用户访问 `http://example.com` 时，GitHub 会把它自动转向 `http://www.example.com`。

带 `www` 的域名叫做“二级域名”。我们先介绍如何绑定“裸域”。在你使用的域名注册商网站中打开 DNS 管理界面，新建一条 A 记录，指向 IP `192.30.252.153` 或者 `192.30.252.154`。

如果想绑定二级域名，在域名注册商的 DNS 管理界面要新建一条 CNAME 记录，指向地址 `<username>.github.io`。记得把 `username` 换成你自己的用户名。

设置好 DNS 之后，把 `CNAME` 文件添加到仓库中，提交，然后推送到 GitHub：

```
~/homepage (source) $ git add CNAME
~/homepage (source) $ git commit -m "Add CNAME"
~/homepage (source) $ git push origin source # 注意，不是 master 分支
~/homepage (source) $ bundle exec jekyll build
~/homepage (source) $ cd _site

~/homepage/_site (master) $ git add CNAME
```

```
~/homepage/_site (master) $ git commit -m 'Add CNAME'  
~/homepage/_site (master) $ git push origin master # 注意, 不是 gh-pages 分支
```

稍等一会儿之后,<sup>5</sup> 应该就可以使用绑定的域名访问账户的网站了。

### 加不加 www, 这是个问题

你可能会纠结, 绑定域名时到底要不要加 www。GitHub 的建议是, [最好加上](#)。因为加上 www 之后, 得到的是二级域名。而根据 GitHub 的要求, 二级域名必须使用 CNAME 记录。使用 CNAME 记录好处多多:

- 可以使用 GitHub Pages 的 CDN (内容分发网络), 理论上访问速度更好;
- 如果 GitHub 更换 IP, 无需修改 DNS 记录;
- 而且还能合理利用 GitHub 提供的“拒绝访问”(DoS) 保护服务。

#### 11.1.4 用于部署的 Rake 任务

每次更新网站都这么折腾, 有点儿筋疲力尽? 那我们就把推送的过程交给电脑自动完成吧。当然, 这一次我们还要劳烦 rake。在文本编辑器中打开 `Rakefile` 文件, 在末尾写入如下代码(以部署项目的网站为例, 账户的网站请相应地修改) :

代码清单 11.1: 自动推送到 GitHub Pages 的 rake 任务

```
desc "Push to github"  
task :push do  
  puts "Pushing to `master` branch:"  
  system "git push origin master"  
  puts "'master' branch updated."  
  puts  
  
  puts "Building site...."  
  system "bundle exec jekyll build"  
  puts
```

5. DNS 生效要等一段时间, 有时可能长达 24 小时。

```
cd '_site' do
  puts "Pushing to `gh-pages` branch:"
  system "git add -A"
  system "git commit -m 'Update at #{Time.now.utc}'"
  system "git push origin gh-pages"
  puts "`gh-pages` branch updated."
end
end
```

我们定义了一个新 `rake` 任务，名为“`push`”，作用是把网站根目录中的代码推送到 GitHub 仓库的 `master` 分支，把 `_site` 文件夹中的内容推送到 `gh-pages` 分支。注意，如果你的网站不在 `gh-pages` 分支中，不要直接复制粘贴这段代码。

这个 `rake` 任务的执行方法很简单，更新网站并在本地预览之后（`master` 分支要手动提交，之所以没交给这个任务自动完成，是为了让你编写更有意义的提交说明），在命令行中执行 `rake push` 即可：

```
~/blog (master) $ bundle exec rake push
```

## 11.2 虚拟主机

如果想把 Jekyll 网站部署到虚拟主机，只要想办法把 `_site` 文件夹中的内容上传到虚拟主机的 FTP 服务器即可。一般情况下，我们可以使用 FTP 软件，例如 [FileZilla](#) 或 [Transmit](#)。FTP 软件的用法很简单，在此不做详细介绍。下面介绍一种更“极客”的方法——在命令行中上传文件。

我们要使用 `glyn` 帮我们把 `_site` 文件夹中的内容上传到虚拟主机中。首先，在本地电脑中安装 `glyn`。这也是一个 Ruby gem，所以我们把它添加到 `Gemfile` 文件中，再执行 `bundle install` 命令：

```
...
gem "glyn"
```

然后，在文本编辑器中打开 `_config.yml` 文件，写入下面几个设置：

```
ftp_host: 'example.com'
ftp_dir: '/web/site/root'
ftp_passive: false
```

其中，`ftp_host` 是虚拟主机的 FTP 服务器地址，你的主机提供商可能会为你提供；`ftp_dir` 是在虚拟主机中存放这个网站的文件夹路径；`ftp_passive` 设置使用哪种模式传输数据：`true` 表示被动

模式，`false` 表示主动模式。请根据你的虚拟主机商提供给你的信息，修改 `ftp_host` 和 `ftp_dir`。

设置好之后，每次更新网站，只需在命令行中执行下述命令，就会把 `_site` 文件夹中的内容上传到 FTP 服务器：

```
~/blog $ bundle exec glynn
```

如果要求输入密码，输入 FTP 服务器的密码。剩下的工作都交给 `glynn` 完成了。上传完毕后，访问你的网站，应该就能看到内容更新了。

### 11.2.1 设置 404 页面

把网站部署到虚拟主机中，相当于我们自己托管了。自己托管意味着 GitHub Pages 等服务提供的“开箱即用”功能失效了，比如 404 页面。GitHub 在他们的服务器中做了相关设置，当出现 HTTP 404 响应码时显示指定的网页（即 `404.html`）。而现在我们要自己动手设置。幸好，设置的方法并不难。我们要使用 `.htaccess` 文件设置 404 页面。

首先，更新 `keep_files` 设置，加入 `.htaccess` 文件：

```
keep_files: [".git", ".nojekyll", ".htaccess"]
```

然后，进入 `_site` 文件夹，新建 `.htaccess` 文件。在文本编辑器中打开，写入以下内容：

```
ErrorDocument 404 /404.html
```

这样就行了。当然，前提是，存在 `404.html` 文件。这个文件的创建方法参见 [7.10 节](#)。

### 11.2.2 使用 git-ftp 部署

如果网站的内容很多，你会发现，使用 `glynn` 部署时每次都会上传所有文件，不管文件有没有改动都会上传。这样浪费时间，不“敏捷”。如果你用过 Git，可能会有这样的疑问：“有没有一种方法能像 Git 那样知道哪些文件有变动，只上传有改动的文件？”答案是，有，[git-ftp](#)。`git-ftp` 利用 Git 的文件差异管理系统，只会上传有改动的文件。`git-ftp` 的安装方法参见[这里](#)。

如果 `_site` 文件夹里生成的网站没有纳入 Git 仓库，先初始化仓库：

```
$ cd _site
_site $ git init
_site $ git add -A
_site $ git commit -m 'init'
```

首次上传时，要执行下述命令：

```
_site $ git ftp init -u <user> -p <passwd> ftp://host.example.com/public_html
```

后续上传则要执行下述命令：

```
_site $ git ftp push -u <user> -p <passwd> ftp://host.example.com/public_html
```

其中，`<user>` 和 `<passwd>` 要替换成FTP的用户名和密码。如果觉得每次输入用户名、密码和FTP地址麻烦，可以编写一个 Rake 任务，例如：

代码清单 11.2：使用 git-ftp 部署网站的 Rake 任务

```
desc 'Deploy to ftp via git-ftp'
task :deploy do
  options = YAML.load_file('_ftp.yml')
  user = options['ftp_username']
  passwd = options['ftp_password']
  host = options['ftp_host'].chomp('/')
  port = options['ftp_port'] || 21
  dir = options['ftp_dir'].start_with?(' '/') ? options['ftp_dir'][1...-1] :
  options['ftp_dir']

  system "bundle exec jekyll build"
  puts
  cd '_site' do
    system "git add -A"
    system "git commit -m 'Update at #{Time.now.utc}'"
    puts
    system "git ftp push --user #{user} --passwd #{passwd}"
    ftp://#[host]:#[port]/#[dir]"
  end
end
```

FTP 服务器的相关信息保存在网站根目录中的 `_ftp.yml` 文件里，这个文件的内容如下：

代码清单 11.3：在 `_ftp.yml` 文件中设置 FTP

```
ftp_host: 'host.example.com'
ftp_dir: 'public_html'
ftp_port: 21
```

```
ftp_username: 'user'  
ftp_password: 'passwd'
```

切记，为了防止泄露密码等信息，这个文件不能纳入 Git 仓库。也就是说，要把 `_ftp.yml` 添加到 `.gitignore` 文件里。现在，需要部署时执行 `bundle exec rake deploy` 命令即可，这个命令只会上传有改动的文件。

### 11.3 再次修改网站配置

网站部署完毕之后，我们还得修改一个设置：把 `url` 改为你使用的域名。如果使用 GitHub 提供的二级域名就填写那个域名；如果使用自己的域名就填写自己的域名。但是域名末尾不能加斜线，应该写成 `http://yourdoamin.com`，而不是 `http://yourdoamin.com/`。

注意，如果是项目的网站，假如地址是 `http://<username>.github.io/hello/`，后面的 `/hello/` 要去掉，只填这之前的地址即可。也就是说，只填域名（顶级或二级等），不含路径。路径应该在 `baseurl` 中设置。

修改之后，需要重新部署一次。这项设置只需做一次，除非你更换了域名。之所以设置 `url`，是因为有些地方，例如网站地图，需要使用网站的域名。

---

# 附录 A：设置 Jekyll 网站

我们在前面做过设置，例如，让 krammdown 使用 GFM 句法。这一篇附录详细介绍各种设置。

Jekyll 网站的设置保存在根目录中的 `_config.yml` 文件里，各个顶层设置都可以通过 `site` 获取。例如，使用 `site.title` 可以获取 `title` 设置。除了 Jekyll 内置的设置之外，还可以根据自己的需求，在这个文件中自定义其他设置（后面的章节会这么做）。

## A.1 设置网站的目录结构

Jekyll 使用一个默认的目录结构生成网站，不过这个结构也可以调整，方法是使用表 A.1 列出的设置项目。执行 `jekyll build` 或 `jekyll server` 命令时会使用这些设置。

表 A.1：目录结构的设置项目

设置	默认值	作用	备注
<code>source</code>	.	网站项目的根目录	默认值为当前文件夹
<code>destination</code>	<code>./_site</code>	保存生成的网站的文件夹	
<code>plugins_dir</code>	<code>./_plugins</code>	保存插件的文件夹	
<code>layouts_dir</code>	<code>./_layouts</code>	保存布局的文件夹	
<code>data_dir</code>	<code>./_data</code>	保存数据文件的文件夹	
<code>includes_dir</code>	<code>./_includes</code>	保存引用文件的文件夹	

一般情况下无需修改这些设置，除非有十分特殊的理由。

## A.2 设置转换规则

执行 `jekyll build` 或 `jekyll server` 命令时会使用表 A.2 列出的设置项目。

表 A.2：转换规则的设置项目

设置	默认值	作用	备注
<code>include</code>	<code>[".htaccess"]</code>	在生成的网站中包含指定的文件或文件夹	Jekyll 生成的网站默认不包含以点号开头的文件或文件夹
<code>exclude</code>	<code>[]</code>	在生成的网站中排除指定的文件或文件夹	例如 <code>README</code> , <code>Gemfile</code> 等
<code>keep_files</code>	<code>[".git", ".svn"]</code>	生成网站时不清除 <code>_site</code> 中指定的文件或文件夹	第 11 章会用到
<code>encoding</code>	<code>"utf-8"</code>	读写文件时使用的编码	
<code>show_drafts</code>	<code>null</code>	是否转换草稿	
<code>limit_posts</code>	<code>0</code>	只转换指定数量的文章	<code>0</code> 表示转换所有文章
<code>future</code>	<code>true</code>	生成的网站中是否包含发布日期为未来某一天的文章	
<code>unpublished</code>	<code>false</code>	是否包含标记为“未发布”的文章	在文章的元信息中设置 <code>unpublished: false</code> , 就会把文章标记为“未发布”
<code>permalink</code>	<code>date</code>	文章固定链接的格式	参加 4.5 节
<code>paginate_path</code>	<code>/page:num</code>	分页链接的格式	
<code>timezone</code>	<code>null</code>	处理文章发布日期时使用的时区	可以使用的时区参见维基百科
<code>quiet</code>	<code>false</code>	是否静默命令行中的输出	
<code>excerpt_separator</code>	<code>\n\n"</code>	把指定分隔符之前的内容当成文章的摘要	

## A.3 设置转换程序

表 A.3 列出的设置项目用于设置 Jekyll 使用的转换程序。

表 A.3: 转换程序的设置项目

设置	默认值	作用	备注
markdown	kramdown	使用指定的程序转换使用 Markdown 编写的内容	除此之外，Jekyll 原生还支持使用 redcarpet 和 rdiscount
markdown_ext	" <code>markdown</code> , <code>mkdown</code> , <code>mkdn</code> , <code>mkd</code> , <code>md</code> "	把使用指定后缀的文件识别为 Markdown 文件	
highlighter	rouge	设置用于高亮代码的程序	
rdiscount	-	设置 rdiscount	这是个顶层设置，旗下的各个子设置参见 <a href="#">A.3.1 节</a>
redcarpet	-	设置 redcarpet	这是个顶层设置，旗下的各个子设置参见 <a href="#">A.3.2 节</a>
kramdown	-	设置 kramdown	这是个顶层设置，旗下的各个子设置参见 <a href="#">A.3.3 节</a>

### A.3.1 设置 rdiscount

rdiscount 虽然是 Ruby gem，但用到了 C 语言扩展。rdiscount 的设置如表 A.4 所示。

表 A.4: rdiscount 的设置项目

设置	默认值	作用	备注
extensions	[]	设置使用哪些扩展	可用的扩展如下表

rdiscount 支持的扩展及说明如下：

表 A.5: rdiscount 的扩展

扩展	作用
:smart	把 ' 转换成 ‘， 把 "" 转换成 “”
:filter_styles	过滤掉 <style> 标签中的内容
:filter_html	过滤掉 Markdown 中的 HTML 代码
:footnotes	把 [^] 转换成脚注，参见 <a href="#">PHP Markdown Extra</a>
:generate_toc	生成文章的目录
:no_image	不生成图片标签 (<img>)
:no_links	不生成链接标签 (<a>)
:no_tables	不生成表格
:strict	使用严格模式处理 Markdown
:autolink	自动创建链接
:safelink	不创建未知类型的链接
:no_pseudo_protocols	不支持伪协议
:no_superscript	不支持上标
:no_strikethrough	不支持使用 ~~ 生成 <del> 标签

### A.3.2 设置 redcarpet

redcarpet 和 rdiscount 一样，虽然也是 Ruby gem，但用到了 C 语言扩展。redcarpet 由 GitHub 的员工 [Vicent Marti](#) 开发，GitHub 就使用 redcarpet 转换 Markdown。Jekyll 以前默认使用 redcarpet 转换 Markdown。redcarpet 的设置如表 A.6 所示。

表 A.6: redcarpet 的设置项目

设置	默认值	作用	备注
extensions	[]	设置使用哪些扩展	可用的扩展如下表

redcarpet 支持的扩展及说明如下：

表 A.7: redcarpet 的扩展

扩展	作用
:no_intra_emphasis	不转换单词内部的着重句法
:tables	支持表格句法，参见 <a href="#">PHP Markdown Extra</a>
:fenced_code_blocks	支持使用三个或多个`或~写代码块
:autolink	自动创建链接
:disableIndentedCodeBlocks	不处理普通的 Markdown 代码块（使用四个空格表示）
:strikethrough	使用 ~~生成 <del> 标签
:lax_spacing	HTML 块两侧不用加空行
:space_after_headers	标题的起始符号后要加上空格，例如 # 一级标题
:superscript	使用 ^ 符号写上标
:underline	把原本用于生成斜体_的符号，改为生成下划线
:highlight	使用 == 生成 <mark> 标签
:quote	使用 " " 生成 <q> 标签
:footnotes	把 [^] 转换成脚注，参见 <a href="#">PHP Markdown Extra</a>

### A.3.3 设置 kramdown

kramdown 纯粹使用 Ruby 开发。因为 kramdown 的扩展机制灵活易用，越来越受欢迎，后来 Jekyll 也把默认的 Markdown 转换程序改成了 kramdown。kramdown 的设置如表 A.8 所示。

表 A.8: kramdown 的设置项目

设置	默认值	作用	备注
auto_ids	true	是否自动生成各级标题的 id 属性	

设置	默认值	作用	备注
footnote_nr	1	脚注的起始序号	
entity_output	as_char	HTML 实体的输出方式	可选值还有： as_input、numeric 和 symbolic
toc_levels	1..6	目录中包含的标题等级	
smart_quotes	lsquo,rsquo, ldquo,rdquo	把引号转换成什么 HTML 实体	
syntax_highlighter_opts	-	高亮程序的设置	这是个顶层设置，旗下的各个子设置如表 A.9 所示

`syntax_highlighter_opts` 的子设置如下表。注意，这些子设置只对 Coderay 有效。

表 A.9：高亮程序的设置项目

设置	默认值	作用	备注
wrap	div	高亮后的代码块外层使用的元素	
line_numbers	inline	行号的显示方式	可选值还有： table 和 nil
line_number_start	1	行号的起始序号	
tab_width	4	制表符的宽度	
bold_every	10	行号为指定数值的整倍数时，加粗显示	
css	style	如何加载样式	可选值还有 class

前面说过，这些设置是全局性的，无法在单个代码块中设置。前面已经介绍过如何让 kramdown 使用 rouge 高亮代码（参见 6.3.2 节）。

## A.4 设置本地服务器

执行 `jekyll server` 命令时会使用表 A.10 列出的设置项目，设置本地服务器。

表 A.10：本地服务器的设置项目

设置	默认值	作用	备注
<code>detach</code>	<code>false</code>	是否释放控制权	如果释放，服务器会在后台运行，在当前命令窗口中可以继续执行其他命令
<code>port</code>	<code>4000</code>	本地服务器使用的端口号	要大于 1024
<code>host</code>	<code>127.0.0.1</code>	本地服务器的 IP 地址	
<code>baseurl</code>	<code>" "</code>	网站的二级目录地址	没有域名，以 / 开头，但后面不加 /

在设置这些项目之前，要对服务器有一定的了解，否则不要随意修改，一旦改错，可能无法启动本地服务器。

## A.5 设置插件

Jekyll 加载和调用插件时会用到表 A.11 列出的设置项目。

表 A.11：插件的设置项目

设置	默认值	作用	备注
<code>safe</code>	<code>false</code>	设置网站所处的环境是否安全	如果设为 <code>true</code> , Jekyll 不会加载任何插件
<code>gems</code>	<code>[]</code>	网站中使用的插件列表	指定的插件必须以 <code>gem</code> 的形式分发
<code>whitelist</code>	<code>[]</code>	插件白名单	即使 <code>safe</code> 为 <code>true</code> , 也会加载这里指定的插件



---

# 附录 B: Markdown 句法简介

这篇附录中的内容编译自 [Markdown 官方网站](#)。

## B.1 概述

### B.1.1 宗旨

Markdown 的目标是“易读易写”。

无论如何，可读性是最重要的。一份使用 Markdown 格式编写的文件应该可以直接以纯文本发布，而且看起来不像是由许多标签或是格式指令所构成。Markdown 句法受到一些现有的“纯文本到 HTML”转换句法的影响，包括 [Setext](#)、[atx](#)、[Textile](#)、[reStructuredText](#)、[Grutatext](#) 和 [Et-Text](#)，而最大的灵感来源其实是纯文本电子邮件格式。

总之， Markdown 的句法全由一些符号组成，这些符号经过精挑细选，作用一目了然。比如，在文字两旁加上星号，看起来就像是在\*强调\*。Markdown 的列表看起来就是列表。Markdown 的引用块看起来就真的像是引用一段文字，就像你在电子邮件中见过的那样。

### B.1.2 兼容 HTML

Markdown 句法的目标是，成为一种适用于网络的书写语言。

Markdown 不想要取代 HTML，甚至也没有要和它相近，它的语法种类很少，只对应 HTML 标记的一小部分。Markdown 的构想不是要使 HTML 文档更容易书写。在我看来， HTML 写起来已经很容易了。Markdown 的理念是，让文档更容易读写，而且能随意修改。HTML 是一种适用于发布出去的格式，Markdown 则是一种书写格式。所以， Markdown 的句法只涵盖纯文本能解决的问题。

不在 Markdown 句法涵盖范围之内的标签，都可以直接在文档里面用 HTML 撰写。不需要额外标注这是 HTML 或是 Markdown；直接写 HTML 标签就可以了。

不过，唯有一个要求：HTML 块级元素——比如 `<div>`、`<table>`、`<pre>`、`<p>` 等，必须在前后加上空行，与其它内容分开，而且开始标签和结尾标签不能使用制表符或空格缩进。Markdown 很智能，不会在 HTML 块级标签外添加不必要的 `<p>` 标签。

假如想在使用 Markdown 编写的文章中加入表格，可以这么写：

一个普通段落。

```
<table>
<tr>
  <td>Foo</td>
</tr>
</table>
```

另一个普通段落。

注意，HTML 块级标签内的 Markdown 句法不会处理。例如，在 HTML 块级元素内使用 Markdown 的\*强调\*句法没有效果。

HTML 行间标签如 `<span>`、`<cite>` 和 `<del>` 可以在 Markdown 的段落、列表或标题里随意使用。依照个人习惯，甚至可以不用 Markdown 格式，而直接采用 HTML 标签来格式化。举例说明：如果比较喜欢 HTML 的 `<a>` 或 `<img>` 标签，可以直接使用这些标签，而不用 Markdown 提供的链接或是图像句法。

和 HTML 块级标签内的句法不同，在 HTML 块级标签之间可以使用 Markdown 句法。

### B.1.3 自动转义特殊字符

对 HTML 来说，有两个字符需要特殊处理：`<` 和 `&`。`<` 符号用于起始标签，`&` 符号则用于标记 HTML 实体。如果想显示这些字符的字面值，必须使用实体形式，写成 `&lt;` 和 `&amp;`。

`&` 字符尤其让网络写手受折磨。如果想写“AT&T”，必须要写成“AT&T”。而且网址中的 `&` 字符也要转义。比如你要链接到：

`http://images.google.com/images?num=30&q=larry+bird`

必须把网址转换成：

`http://images.google.com/images?num=30&amp;q=larry+bird`

才能放到链接的 `href` 属性里。不用说也知道，这一步容易被忽略，这可能是 HTML 验证时出现错误最多的情况。

Markdown 可以让你自然地书写字符，它会代你转义必要的字符。如果你使用的 & 字符是 HTML 字符实体的一部分，它会保留原状，否则会被转换成 `&amp;`。

所以，如果要在文档中插入一个版权符号，可以这样写：

```
&copy;
```

Markdown 不会处理。但是如果写成：

AT&T

Markdown 就会把它转换成：

```
AT&amp;T
```

类似的转换也会应用于 < 符号上，因为 Markdown 允许直接使用 HTML。如果把 < 符号作为 HTML 标签的定界符使用，那 Markdown 不会做任何转换。但是如果写成：

```
4 < 5
```

Markdown 会将其转换为：

```
4 &lt; 5
```

不过，在 `code` 标签内，不论是行间元素还是块级元素，< 和 & 都会转换成 HTML 实体。这个特性让你可以很容易地用 Markdown 写 HTML 代码。

## B.2 块级元素

### B.2.1 段落和换行

一个 Markdown 段落由一个或多个连续的文本行组成，前后要有一个以上的空行。（空行的定义是，显示上看起来像是空的，便会被视为空行。比方说，若某一行只包含空格和制表符，则该行也会被视为空行。）普通段落不能用空格或制表符缩进。

“由一个或多个连续的文本行组成”这句话其实暗示了 Markdown 允许在段落内强制换行（插入换行符）。这个特性和其他大部分“纯文本到 HTML”转换句法不一样（包括 Movable Type 的“Convert Line Breaks”选项），它们会把换行符转换成 `<br />` 标签。

如果确实想使用 Markdown 插入 `<br />` 标签的话，在插入处输入两个以上的空格之后再回车。

的确，需要多费点事（多加空格）才能得到 `<br />`，但是直接把每个换行都转换为 `<br />` 的方法在 Markdown 中并不适合。Markdown 中电子邮件格式的[引用块](#)和[多段落列表](#)使用换行排版时，不但更好用，而且更便于阅读。

## B.2.2 标题

Markdown 支持两种标题格式：Setext 式和 atx 式。

Setext 是底线形式，使用 =（一级标题）和 -（二级标题）。例如：

```
This is an H1  
=====
```

```
This is an H2  
-----
```

= 和 - 的数量不限。

Atx 式则是在行首插入 1 到 6 个 #，对应一级到六级标题。例如：

```
# This is an H1  
  
## This is an H2  
  
##### This is an H6
```

你可以选择性地“闭合”atx 式标题。这么做纯粹是为了美观，如果觉得这样看起来比较舒适，可以在行尾加上 #，而且行尾的 # 数量不用和开头一样多（行首的 # 数量决定标题的级别）：

```
# This is an H1 #  
  
## This is an H2 ##  
  
### This is an H3 #####
```

### B.2.3 引用块

Markdown 使用电子邮件中常用的 > 标记引用块。如果知道怎么在电子邮件中引用文字，就知道怎么在 Markdown 中创建引用块。如果在引用的内容中加上硬换行，然后在每一行前面加上 >，看起来就更美观了：

```
> This is a blockquote with two paragraphs. Lorem ipsum dolor sit amet,  
> consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus.  
> Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.  
>  
> Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse  
> id sem consectetur libero luctus adipiscing.
```

Markdown 也允许你偷懒，可以只在整个段落的第一行最前面加上 > :

```
> This is a blockquote with two paragraphs. Lorem ipsum dolor sit amet,  
consectetuer adipiscing elit. Aliquam hendrerit mi posuere lectus.  
Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.  
  
> Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse  
id sem consectetur libero luctus adipiscing.
```

引用块可以嵌套（即引用内的引用），只要根据层次加上不同数量的 > 即可：

```
> This is the first level of quoting.  
>  
> > This is nested blockquote.  
>  
> Back to the first level.
```

引用块内也可以使用其他 Markdown 句法，包括标题、列表、代码块等：

```
> ## This is a header.  
>  
> 1. This is the first list item.  
> 2. This is the second list item.  
>  
> Here's some example code:  
>  
>     return shell_exec("echo $input | $markdown_script");
```

任何像样的文本编辑器都能轻松地创建电子邮件格式的引用。例如，在 BBEdit 中，选取文字后  
再从“文本”菜单中选择“增加引用层级”。

#### B.2.4 列表

Markdown 支持有序列表和无序列表。

无序列表使用星号、加号或是减号标记：

- \* Red
- \* Green
- \* Blue

等价于：

- + Red
- + Green
- + Blue

也等价于：

- Red
- Green
- Blue

有序列表则在数字后加一个英文句点：

1. Bird
2. McHale
3. Parish

很重要的一点是，在列表项上使用的数字并不影响输出的 HTML，上面的列表生成的 HTML 标记为：

```
<ol>
  <li>Bird</li>
  <li>McHale</li>
  <li>Parish</li>
</ol>
```

如果把列表写成：

1. Bird
1. McHale
1. Parish

甚或是：

3. Bird
1. McHale
8. Parish

得到的 HTML 都是一样的。重点是，如果愿意，可以使用递增的数字标记列表，让 Markdown 中的序号和 HTML 中一样。不过也可以偷懒，不管数字的顺序。

偷懒是可以，不过第一个列表项还得使用 1。未来，Markdown 可能会支持以任意数字标记第一个列表项。

列表项的标记通常放在最左边，但是也可以缩进，最多 3 个空格，列表项的标记后面则一定要跟着至少一个空格或制表符。

要让列表看起来更漂亮，可以把内容用固定的缩进整理好：

- \* Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi,  
viverra nec, fringilla in, laoreet vitae, risus.
- \* Donec sit amet nisl. Aliquam semper ipsum sit amet velit.  
Suspendisse id sem consectetur libero luctus adipiscing.

如果懒，写可以这么写：

- \* Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi,  
viverra nec, fringilla in, laoreet vitae, risus.
- \* Donec sit amet nisl. Aliquam semper ipsum sit amet velit.  
Suspendisse id sem consectetur libero luctus adipiscing.

如果列表项之间用空行分开，输出 HTML 时会把列表项中的内容放到 `<p>` 标签中。例如下面的 Markdown 文本：

- \* Bird
- \* Magic

会转换为：

```
<ul>
    <li>Bird</li>
    <li>Magic</li>
</ul>
```

但这段文本：

```
* Bird

* Magic
```

会转换为：

```
<ul>
    <li><p>Bird</p></li>
    <li><p>Magic</p></li>
</ul>
```

列表项可以包含多个段落，段落都必须缩进 4 个空格或 1 个制表符：

1. This is a list item with two paragraphs. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam hendrerit mi posuere lectus.

Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus. Donec sit amet nisl. Aliquam semper ipsum sit amet velit.

2. Suspendisse id sem consectetur libero luctus adipiscing.

如果每行都有缩进，看起来更好。当然，这里也可以偷懒，写成：

```
* This is a list item with two paragraphs.
```

This is the second paragraph in the list item. You're only required to indent the first line. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

```
* Another item in the same list.
```

如果要在列表项中加入引用块，> 必须缩进：

```
* A list item with a blockquote:  
  > This is a blockquote  
  > inside a list item.
```

如果要在列表项中加入代码块，代码块必须缩进两次，即 8 个空格或 2 个制表符：

```
* A list item with a code block:  
  
<code goes here>
```

有件事值得注意一下，有时可能不小心触发一个有序列表，例如：

1986. What a great season.

也就是说，行首是“数字+点号+空格”。为了避免出现这种问题，可以转义点号：

1986\. What a great season.

## B.2.5 代码块

代码块表示程序相关的内容，或者表示源码。代码块和段落不一样，代码块中的内容是什么样，显示的就是什么样。Markdown 把代码块放在 `<pre>` 和 `<code>` 元素中。

要在 Markdown 中创建代码块很简单，只要缩进 4 个空格或是 1 个制表符就可以。例如，下面的文本：

This is a normal paragraph:

This is a code block.

生成的 HTML 是：

```
<p>This is a normal paragraph:</p>  
  
<pre><code>This is a code block.  
</code></pre>
```

一层缩进（4 个空格或 1 个制表符）不见了。例如，下面这段文本：

Here is an example of AppleScript:

```
tell application "Foo"
    beep
end tell
```

会转换成：

```
<p>Here is an example of AppleScript:</p>

<pre><code>tell application "Foo"
    beep
end tell
</code></pre>
```

代码块直到没有缩进的行才会结束（或者到达文件末尾）。

在代码块中，&、<和>会被自动转换成HTML实体。这种处理方式便于在Markdown中插入HTML源码——直接粘贴再缩进就好了。例如，下面这段HTML源码：

```
<div class="footer">
    &copy; 2004 Foo Corporation
</div>
```

会转换成：

```
<pre><code>&lt;div class="footer"&gt;
    &copy; 2004 Foo Corporation
&lt;/div&gt;
</code></pre>
```

在代码块中，不会转换常规的Markdown句法，星号就是星号。所以可以方便地使用Markdown编写介绍Markdown句法的文章。

## B.2.6 分隔线

在一行中写上三个或三个以上的星号、连字符、下划线可以生成一个分割线（`<hr />`）。连字符和星号之间还可以加入空格。下面这些写法都会生成分割线：

```
* * *
***
```

\*\*\*\*\*

- - -

-----

## B.3 行间元素

### B.3.1 链接

Markdown 支持两种链接句法：行间式和引用式。

不管哪一种，链接文字都放在方括号中（[]）。

要创建行间式链接，只要在方括号后面紧接着使用圆括号，并在圆括号中写入网址即可。如果还想要加上链接的提示文字，只要在网址后面，用双引号把提示文字包起来即可。例如：

```
This is [an example](http://example.com/ "Title") inline link.
```

```
[This link](http://example.net/) has no title attribute.
```

生成的 HTML 是：

```
<p>This is <a href="http://example.com/" title="Title">  
an example</a> inline link.</p>
```

```
<p><a href="http://example.net/">This link</a> has no  
title attribute.</p>
```

如果链接到同一台主机中的页面，还可以使用相对路径：

```
See my [About](/about/) page for details.
```

引用式链接使用两个方括号对，第二个方括号对里是这个链接的标签：

```
This is [an example][id] reference-style link.
```

两个方括号对之间也可以加入空格：

```
This is [an example] [id] reference-style link.
```

然后，在文档的任意位置，新起一行，定义这个标签的内容：

```
[id]: http://example.com/ "Optional Title Here"
```

链接内容定义的形式为：

- 方括号（前面可以选择性地加上至多三个空格来缩进），里面输入链接文字
- 接着一个冒号
- 接着一个以上的空格或制表符
- 接着链接的网址
- 可选的提示文字，可以放在单引号、双引号或是括弧中

下面这三种形式的链接是等价的：

```
[foo]: http://example.com/ "Optional Title Here"  
[foo]: http://example.com/ 'Optional Title Here'  
[foo]: http://example.com/ (Optional Title Here)
```

网址也可以用尖括号括起来：

```
[id]: <http://example.com/> "Optional Title Here"
```

提示文字也可以另起一行写，但要缩进。如果网址很长，这样写很好看：

```
[id]: http://example.com/longish/path/to/resource/here  
"Optional Title Here"
```

标签定义只在生成链接时有用，不会直接出现在文件中。

链接的标签可以包含字母、数字、空白和标点符号，而且不区分大小写。例如，下面两个链接是等价的：

```
[link text][a]  
[link text][A]
```

引用式链接也可以不指定链接的标签，第二个方括号对中什么也不写——此时，把链接文字当做标签。例如，把“Google”链接到 google.com，可以写成：

```
[Google][]
```

然后定义这个链接：

```
[Google]: http://google.com/
```

由于链接的标签中可以包含空白，所以这种简写形式也适用于包含多个单词的链接文字：

```
Visit [Daring Fireball][] for more information.
```

然后定义这个链接：

```
[Daring Fireball]: http://daringfireball.net/
```

链接的定义可以放在文件中的任意位置，我喜欢放在链接出现的段落后面。你也可以把它放在文件末尾，就像是脚注一样。

下面是一个引用式链接实例：

```
I get 10 times more traffic from [Google] [1] than from  
[Yahoo] [2] or [MSN] [3].
```

```
[1]: http://google.com/ "Google"  
[2]: http://search.yahoo.com/ "Yahoo Search"  
[3]: http://search.msn.com/ "MSN Search"
```

如果不指定链接的标签，也可以写成：

```
I get 10 times more traffic from [Google][] than from  
[Yahoo][] or [MSN][].
```

```
[google]: http://google.com/ "Google"  
[yahoo]: http://search.yahoo.com/ "Yahoo Search"  
[msn]: http://search.msn.com/ "MSN Search"
```

上面两种写法都会生成下面的 HTML：

```
<p>I get 10 times more traffic from <a href="http://google.com/"  
title="Google">Google</a> than from  
<a href="http://search.yahoo.com/" title="Yahoo Search">Yahoo</a>  
or <a href="http://search.msn.com/" title="MSN Search">MSN</a>. </p>
```

下面是使用行间式链接书写的同一段路，提供在此仅作比较：

```
I get 10 times more traffic from [Google](http://google.com/ "Google")  
than from [Yahoo](http://search.yahoo.com/ "Yahoo Search") or  
[MSN](http://search.msn.com/ "MSN Search").
```

引用式链接的好处不是写起来容易，而是为了提高文档的可读性。你可以比较一下前面两种写法：使用引用式，这段文字只有 81 个字符，但使用行间式有 176 个字符。而生成的 HTML 只有 234 个字符，可是 HTML 比纯文本中的标记要多。

使用引用式链接，文档更像是浏览器最后渲染得到的结果。把标记相关的信息移出，就不会打断阅读节奏。

### B.3.2 强调

Markdown 使用星号 (\*) 和下划线 (\_) 表示强调。一对 \* 或 \_ 中的文字生成 HTML 中的 `<em>` 元素；两对 \* 或 \_ 中的文字生成 HTML 中的 `<strong>` 元素。例如：

```
*single asterisks*
_
single underscores_
**double asterisks**
__
double underscores__
```

会转换成：

```
<em>single asterisks</em>
<em>single underscores</em>
<strong>double asterisks</strong>
<strong>double underscores</strong>
```

在 \* 和 \_ 之间，可以根据自己的喜欢选择。唯有一个限制，前面使用哪个符号，后面也要使用哪个符号。

强调符号也可以直接插在文字中间：

```
un*f*rigging*believable
```

但是如果 \* 和 \_ 两边都有空白的话，就只会被当成普通的符号。

如果要在文字前后插入普通的星号或下划线，可以使用反斜线转义：

```
\*this text is surrounded by literal asterisks\*
```

### B.3.3 代码

如果要标记行间代码，可以使用反引号（`）。和代码块不同，行间代码在段落中。例如：

```
Use the `printf()` function.
```

会生成：

```
<p>Use the <code>printf()</code> function.</p>
```

如果要在行间代码中插入反引号，可以使用多个反引号：

```
``There is a literal backtick (`) here.``
```

生成的 HTML 是：

```
<p><code>There is a literal backtick (`) here.</code></p>
```

行间代码两侧的分隔符都可以包含空格——前面的分隔符后加一个空格，后面的分隔符前加一个空格。这样就可以在行间代码的开头使用反引号了：

```
A single backtick in a code span: `` ``
```

```
A backtick-delimited string in a code span: `` `foo` ``
```

生成的 HTML 是：

```
<p>A single backtick in a code span: <code>`</code></p>
```

```
<p>A backtick-delimited string in a code span: <code>`foo`</code></p>
```

在行间代码中，& 和尖括号都会自动转换成 HTML 实体。这样便于插入 HTML 标签，例如：

```
Please don't use any `<blink>` tags.
```

生成的 HTML 为：

```
<p>Please don't use any <code>&lt;blink&gt;</code> tags.</p>
```

也可以这样写：

```
`&#8212;` is the decimal-encoded equivalent of `&mdash;`.
```

生成的 HTML 是：

```
<p><code>&#8212;</code> is the decimal-encoded  
equivalent of <code>&mdash;</code>.</p>
```

### B.3.4 图片

老实说，要找到一种自然的句法在文本中插入图片相当有难度。

Markdown 使用一种和链接相似的句法标记图片，也有两种写法：行间式和引用式。

行间式举例如下：

```
![Alt text](/path/to/img.jpg)
```

```
![Alt text](/path/to/img.jpg "Optional title")
```

详细叙述如下：

- 一个英文感叹号 !；
- 后面跟着一个方括号，里面是图片的替代文字；
- 再跟着一个圆括号，里面是图片的地址，以及可选的 `title` 属性，放在双引号或单引号中。

引用式如下所示：

```
![Alt text][id]
```

其中，“id”是图片的标签。标签的定义和链接一样：

```
[id]: url/to/image "Optional title attribute"
```

目前，Markdown 的句法还不支持指定图片的尺寸。如果尺寸很重要，可以直接使用 `<img>` 标签。

## B.4 其它

#### B.4.1 自动链接

Markdown 支持一种简写的链接格式，可以自动链接到 URL 和电子邮件地址：把 URL 或电子邮件地址放在一对尖括号中。也就是说，如果想显示链接和电子邮件的地址本身，而且要可以点击，可以这么写：

<http://example.com/>

Markdown 会将其转换成:

```
<a href="http://example.com/">http://example.com/</a>
```

自动链接电子邮件地址的方式类似，只不过 Markdown 会使用随机的十进制和十六进制实体编码电子邮件地址，防止垃圾邮件骚扰。例如，Markdown 会把下面的地址：

<address@example.com>

转换成：

<a href="#&#x6D;&#x61;i&#x6C;&#x74;&#x6F;:&#x61;&#x64;&#x64;&#x72;&#x65;  
&#115;&#115;&#64;&#101;&#120;&#x61;&#109;&#x70;&#x6C;e&#x2E;&#99;&#111;  
&#109;">&#x61;&#x64;&#x64;&#x72;&#x65;&#115;&#115;&#64;&#101;&#120;&#x61;  
&#109;&#x70;&#x6C;e&#x2E;&#99;&#111;&#109;</a>

在浏览器里中，这段字符会显示为指向“address@example.com”的链接。

(这种处理方式虽然可以糊弄不少机器人，但并不能全部挡下来，不过总比什么都不做好。不管怎样，公开电子邮件地址终究会引来垃圾邮件。)

#### B.4.2 反斜线转义

Markdown 允许使用反斜线转义在 Markdown 句法中有特殊意义的字符。例如，如果想在文字两侧加上星号（而不生成 `<em>` 元素），可以在星号前加上反斜线：

\\*literal asterisks\\*

Markdown 支持使用反斜线转义如下字符：

\	反斜线
'	反引号
*	星号
_	下划线
{}	花括号
[]	方括号
()	括号
#	井号
+	加号
-	减号
.	英文句号
!	英文惊叹号

---

## 附录 C: Liquid 模板引擎简介

Liquid 模板系统有两种标记: 输出值和标签。

### C.1 输出值

输出值使用 `{{ }}` 句法, 例如:

```
Hello {{name}}
Hello {{user.name}}
Hello {{ 'tobi' }}
```

### C.2 过滤器

在输出值后可以使用过滤器, 进一步修改输出的内容。

```
Hello {{ 'tobi' | uppercase }}
Hello tobi has {{ 'tobi' | size }} letters!
Hello {{ '*tobi*' | textile | uppercase }}
Hello {{ 'now' | date: "%Y %h" }}
```

Liquid 内建支持的过滤器如表 B.1。

表 C.1: Liquid 过滤器

过滤器	作用	举例
<code>date</code>	格式化日期, 参见 <a href="#">Ruby 文档</a>	
<code>capitalize</code>	把单词的首字母转换成大写	
<code>downcase</code>	把所有字母都转换成小写	
<code>upcase</code>	把所有字母都转换成大写	

过滤器	作用	举例
<code>first</code>	获取数组的第一个元素	
<code>last</code>	获取数组的最后一个元素	
<code>join</code>	使用指定的值连接数组中的元素	
<code>sort</code>	排序数组中的元素	
<code>map</code>	按照指定的规则处理数组中的每一个元素	
<code>size</code>	返回数组或字符串的长度	
<code>escape</code>	转义字符串	
<code>escape_once</code>	转义字符串，已经转义成 HTML 实体的不再转义	
<code>strip_html</code>	删除字符串中的 HTML 标签	
<code>strip_newlines</code>	删除字符串中的换行符 (\n)	
<code>newline_to_br</code>	把换行符替换成  	
<code>replace</code>	替换	<pre>{{ 'foofoo'     replace:'foo','bar' }} #=&gt; 'barbar'</pre>
<code>replace_first</code>	只替换匹配条件的第一个字符串	
<code>remove</code>	删除	<pre>{{ 'foobarfoobar'     remove:'foo' }} #=&gt; 'barbar'</pre>
<code>remove_first</code>	只删除匹配条件的第一个字符串	
<code>truncate</code>	截断字符串，只保留指定数量的字符	<pre>{{ 'foobarfoobar'   truncate:   5 }} #=&gt; 'foob'</pre>
<code>truncatewords</code>	截断字符串，只保留指定数量的单词数	

过滤器	作用	举例
<code>prepend</code>	在字符串之前添加内容	<code>{{ 'bar'   prepend:'foo' }}</code> <code>#=&gt; 'foobar'</code>
<code>append</code>	在字符串之后添加内容	<code>{{ 'foo'   append:'bar' }}</code> <code>#=&gt; 'foobar'</code>
<code>slice</code>	截取字符串，第一个参数是偏移值，第二个参数是长度	<code>{{ "hello"   slice: -3, 3 }}</code> <code>#=&gt; llo</code>
<code>minus</code>	减法运算	<code>{{ 4   minus:2 }}</code> => 2
<code>plus</code>	加法运算，或者连接字符串	<code>{{ '1'   plus:'1' }}</code> #=> '11'
<code>times</code>	乘法运算	<code>{{ 5   times:4 }}</code> => 20
<code>divided_by</code>	除法运算	<code>{{ 10   divided_by:2 }}</code> => 5
<code>split</code>	按照指定的模式分拆字符串	<code>{{ "a~b"   split:"~" }}</code> #=> ['a','b']
<code>modulo</code>	取余数	<code>{{ 3   modulo:2 }}</code> #=> 1

## C.2.1 Jekyll 扩展的过滤器

除了上述 Liquid 内置的过滤器之外，Jekyll 还添加了一些，如表 C.2 所示。

表 C.2: Jekyll 扩展的过滤器

过滤器	作用	举例
<code>date_to_xmlschema</code>	把日期转换成 ISO 8601 格式	<code>{{ site.time   date_to_xmlschema }}</code> <code>#=&gt; 2008-11-07T13:07:54-08:00</code>
<code>date_to_rfc822</code>	把日期转换成 RFC-822 格式	<code>{{ site.time   date_to_rfc822 }}</code> <code>#=&gt; Mon, 07 Nov 2008 13:07:54 -0800</code>

过滤器	作用	举例
<code>date_to_string</code>	把日期转换成简短形式	<pre>{{ site.time   date_to_string }} #=&gt; 07 Nov 2008</pre>
<code>date_to_long_string</code>	把日期转换成长形式	<pre>{{ site.time   date_to_long_string }} #=&gt; 07 November 2008</pre>
<code>where</code>	按指定的条件收集集合中的元素	<pre>{{ site.members   where:"graduation_year","2014" }}</pre>
<code>group_by</code>	按指定的条件分组集合中的元素	<pre>{{ site.members   group_by:"graduation_year" }} #=&gt; [{"name"=&gt;"2013", "items"=&gt;[...]}, {"name"=&gt;"2014", "items"=&gt;[...]}]</pre>
<code>xml_escape</code>	转义字符串，以便在 XML 中使用	
<code>cgi_escape</code>	把特殊字符转换成 %XX 形式	<pre>{{ "foo,bar;baz?"   cgi_escape }} #=&gt; foo%2Cbar%3Bbaz%3F</pre>
<code>uri_escape</code>	URI 转义	<pre>{{ "foo, bar \baz?"   uri_escape }} #=&gt; foo,%20bar%20%5Cbaz?</pre>
<code>number_of_words</code>	字数统计	
<code>array_to_sentence_string</code>	把数组转换成字符串形式	<pre>{{ page.tags   array_to_sentence_string }} #=&gt; foo, bar, and baz</pre>
<code>textilize</code>	把 Textile 文本转换成 HTML	
<code>markdownify</code>	把 Markdown 文本转换成 HTML	

过滤器	作用	举例
<code>scssify/sassify</code>	把 Sass 或 SCSS 转换成 CSS	
<code>slugify</code>	把空白和字母数字之外的字符替换成连字符	
<code>jsonify</code>	把 Hash 或数组转换成 JSON 格式	

## C.3 标签

标签用于处理模板中的逻辑。Liquid 支持的标签如表 C.3 所示。

表 C.3: Liquid 标签

标签	作用
<code>assign</code>	把一个值赋值给变量
<code>capture</code>	把块中的内容赋值给变量
<code>case</code>	块标签, 条件分支
<code>comment</code>	块标签, 注释
<code>cycle</code>	一般在循环中使用, 交替输出指定的值
<code>for</code>	<code>for</code> 循环
<code>if</code>	<code>if...else</code> 条件判断
<code>include</code>	引入其他模板
<code>raw</code>	输出原始内容
<code>unless</code>	和 <code>if</code> 相反

### C.3.1 注释

注释是最简单的标签, 块中的内容不会出现在网页中:

```
We made 1 million dollars {%- comment %} in losses {%- endcomment %} this year
```

### C.3.2 原始内容

包含在 `raw` 中的内容不会经由 Liquid 处理。如果其他程序（例如 Mustache 和 Handlebars）使用的句法和 Liquid 类似，可以把内容放在 `raw` 标签中，然后再传给这些程序：

```
{% raw %}
  In Handlebars, {{ this }} will be HTML-escaped, but {{{ that }}} will not.
{% endraw %}
```

### C.3.3 if 条件语句

如果你了解一种编程语言，对 `if` 语句一定不陌生。`if` 语句中还可以有 `elsif` 和 `else` 分支。

```
{% if user %}
  Hello {{ user.name }}
{% endif %}

# 和前一个语句一样
{% if user != null %}
  Hello {{ user.name }}
{% endif %}

{% if user.name == 'tobi' %}
  Hello tobi
{% elsif user.name == 'bob' %}
  Hello bob
{% endif %}

{% if user.name == 'tobi' or user.name == 'bob' %}
  Hello tobi or bob
{% endif %}

{% if user.name == 'bob' and user.age > 45 %}
  Hello old bob
{% endif %}

{% if user.name != 'tobi' %}
```

```

    Hello non-tobi
{%- endif %}

# 和前一个语句一样
{% unless user.name == 'tobi' %}
    Hello non-tobi
{%- endunless %}

# 检查数组中元素的数量
{% if user.payments == empty %}
    you never paid !
{%- endif %}

{% if user.payments.size > 0 %}
    you paid !
{%- endif %}

{% if user.age > 18 %}
    Login here
{%- else %}
    Sorry, you are too young
{%- endif %}

# array = 1,2,3
{% if array contains 2 %}
    array includes 2
{%- endif %}

# string = 'hello world'
{% if string contains 'hello' %}
    string includes 'hello'
{%- endif %}

```

### C.3.4 case 语句

如果条件很多，还可以使用 `case` 语句：

```

{%- case template %}

{%- when 'label' %}
```

```
// {{ label.title }}  
{% when 'product' %}  
// {{ product.vendor | link_to_vendor }} / {{ product.title }}  
{% else %}  
// {{page_title}}  
{% endcase %}
```

### C.3.5 cycle

有时你可能想交替显示颜色，Liquid 内建支持这种需求：

```
{% cycle 'one', 'two', 'three' %}
```

### C.3.6 for 循环

for 语句用于遍历集合：

```
{% for item in array %}  
{{ item }}  
{% endfor %}
```

如果遍历 Hash，item[0] 是键，item[1] 是键对应的值：

```
{% for item in hash %}  
{{ item[0] }}: {{ item[1] }}  
{% endfor %}
```

在 for 循环中可以使用下述辅助变量：

```
forloop.length      # => 整个循环的次数  
forloop.index      # => 当前执行循环到第几次  
forloop.index0     # => 当前执行循环到第几次（有前导零）  
forloop.rindex      # => 循环还剩多少次  
forloop.rindex0     # => 循环还剩多少次（有前导零）  
forloop.first       # => 是第一次吗？  
forloop.last        # => 是最后一次吗？
```

还有两个属性可以限制传入循环的集合：

- `limit:int`: 限制传入循环的元素数量；

- `offset:int`: 从第 N + 1 个元素开始循环。

```
# array = [1,2,3,4,5,6]
{% for item in array limit:2 offset:2 %}
{{ item }}
{% endfor %}
# 结果为 3,4
```

还可以倒序遍历：

```
{% for item in collection reversed %} {{item}} {% endfor %}
```

### C.3.7 变量赋值

你可以把数据赋值给变量，这个变量可以在其他标签中使用，或者直接输出。创建变量最简单的方法是使用 `assign` 标签，用法如下：

```
{% assign name = 'freestyle' %}

{% for t in collections.tags %}
{% if t == name %}
<p>Freestyle!</p>
{% endif %}
{% endfor %}
```

如果想把多个字符串合并为一个字符串，然后再赋值给变量，可以使用 `capture` 标签。

```
{% capture attribute_name %}{{ item.title | handleize }}-{{ i }}-color{% endcapture %}

<label for="{{ attribute_name }}>Color:</label>
<select name="attributes[{{ attribute_name }}]" id="{{ attribute_name }}>
<option value="red">Red</option>
<option value="green">Green</option>
<option value="blue">Blue</option>
</select>
```



---

## 附录 D：修订日志

### v3.2.1.p1, 2016年8月4日

- 升级到 Jekyll 3.2.1;
- 根据 Jekyll 默认的文件结构修改内容;
- 调整章节顺序;

### v3.2.0.p1, 2016年7月28日

- 增加对 `{% raw %}` 标签的说明;
- 调整章节顺序;
- 局部内容完善;
- 优化部署最佳实践;
- 全面使用 Bundler;
- 升级到 Jekyll 3.2.0;

### v1.2.0, 2016年4月19日

- 升级到 Jekyll 3.1.3;
- 增加对 Ruby 和 Rubygem 的国内镜像说明;
- 增加说明，全面使用 Bundler;
- 删除对 Textile 的简要说明;

### v1.1.0, 2016年02月19日

- 升级到 Jekyll 3.1.1;
- 删除部署到 GitCafe Pages 的说明;
- 增加对 Jekyll 钩子的说明;

## **v0.0.4, 2015年09月06日**

- 第10章“显示 Jekyll 版本信息”一节添加“注意”说明；
- 第8章“虚拟主机”一节添加“设置 404 页面”小节；
- 第8章“虚拟主机”一节添加“使用 git-ftp 部署”小节；

## **v0.0.3, 2015年02月12日**

- 修正 ePub 格式的样式；

## **v0.0.2, 2015年02月07日**

- 修正代码片段中的错误转义；
- 修正 PDF 格式中目录的错误分页；
- 修正文章导航的链接：加上 `site.baseurl`；
- 优化部分代码；
- 第 1 章添加“随书源码”一节；
- 添加第 11 章，“数据文件”；

## **v0.0.1, 2015年01月27日**

- 初始版发布。