Jekyll教程

Author williamfan 2017-05-25 Jekyll

website pengfeifan.github.io

- Jekyll教程
  - 安装Jekyll
  - Jekyll入门
    - 全局配置
    - 目录结构
    - YAML头信息
    - 撰写博客
    - 使用草稿
    - 创建页面
    - 常用变量
  - Jekyll定制
    - 过滤器
    - 标签
    - 永久链接
    - 分页功能
  - GitHub Pages
    - 将 Jekyll 部署到 Github Pages 上

安装Jekyll

Runing Jekyll on Windows

http://jekyllrb.com/
安装Ruby
http://download.csdn.net/detail/fanpengfei0/9851722
https://cache.ruby-china.org/pub/ruby/
C:\Ruby23-x64\bin 添加到环境变量

安装RubyGems
http://download.csdn.net/detail/fanpengfei0/9851784
//在RubyGems官网上下载压缩包，解压到你的本地任意位置
//在Terminal中
cd yourpath to RubyGems //你解压的位置

```
ruby setup.rb
```

```
D:\apps\rubygems-2.6.12>ruby setup.rb
RubyGems 2.6.12 installed
Parsing documentation for rubygems-2.6.12
Installing ri documentation for rubygems-2.6.12

=== 2.6.12 / 2017-04-30

Bug fixes:
```

安装jekyll

```
gen install jekyll
gem install bundler
gem install jekyll-paginate
```

使用Jekyll模板并生成静态页面

```
jekyll new myblog
cd muyblog
jekyll server
http://127.0.0.1:4000/
```

```
D:\app\jekyll\myblog>jekyll server
Configuration file: D:/app/jekyll/myblog/_config.yml
Configuration file: D:/app/jekyll/myblog/_config.yml
            Source: D:/app/jekyll/myblog
       Destination: D:/app/jekyll/myblog/_site
 Incremental build: disabled. Enable with --incremental
      Generating...
                    done in 2.435 seconds.
  Please add the following to your Gemfile to avoid polling for changes:
    gem 'wdm', '>= 0.1.0' if Gem.win_platform?
 Auto-regeneration: enabled for 'D:/app/jekyll/myblog'
Configuration file: D:/app/jekyll/myblog/_config.yml
    Server address: http://127.0.0.1:4000/
  Server running... press ctrl-c to stop.
[2017-05-25 14:01:02] ERROR `/favicon.ico' not found.
```

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

```
jekyll build
//当前文件夹中的内容将会生成到 ./site 文件夹中。
jekyll build --destination <destination>
//当前文件夹中的内容将会生成到目标文件夹<destination>中。
jekyll build --source <source> --destination <destination>
//指定源文件夹<source>中的内容将会生成到目标文件夹<destination>中。
jekyll build --watch
//当前文件夹中的内容将会生成到 ./site 文件夹中，
//查看改变，并且自动再生成。
```

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

```
jekyll serve
//一个开发服务器将会运行在 http://localhost:4000/
jekyll serve --detach
//功能和`jekyll serve`命令相同，但是会脱离终端在后台运行。
//如果你想关闭服务器，可以使用`kill -9 1234`命令，"1234" 是进程号（PID）。
//如果你找不到进程号，那么就用`ps aux | grep jekyll`命令来查看，然后关闭服务
器。[更多](http://unixhelp.ed.ac.uk/shell/jobz5.html).
jekyll serve --watch
//和`jekyll serve`相同，但是会查看变更并且自动再生成。
```

Jekyll入门

jekyll配置http://jekyll.com.cn/docs/configuration/

全局配置

全局配置

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

设置	配置项 和 参数
Site Source 改变 Jekyll 读取文件的目录	`source: DIR` `-s, --source DIR`
Site Destination 改变 Jekyll 写入文件的目录	`destination: DIR` `-d, --destination DIR`
Safe 禁用 自定义插件。	`safe: BOOL` `--safe`
Exclude 转换时排除某些文件夹或文件。被排除的文件或文件夹的路径是相对于网站源码目录的，源码目录以外不受影响。	`exclude: [DIR, FILE, ...]`
Include 转换时强制包含某些文件、文件夹。 `.htaccess` 是个典型的例子，因为默认排除 . (dot，英文中的句号) 开头的文件。	`include: [DIR, FILE, ...]`
Time Zone 设置时区，这个设置作用于 `TZ` 变量， Ruby 用它来处理日期和时间。 IANA Time Zone Database 里边的都有效，比如 `America/New_York` 。默认值为操作系统的时区。	`timezone: TIMEZONE`

Encoding 设置文件的编码，仅 Ruby 1.9 以上可用。默认值为 nil ，使用 Ruby 默认的 `ASCII-8BIT` 。可以用命令 `ruby -e 'puts Encoding::list.join("\n")'` 查看 Ruby 可用的编码。	`encoding: ENCODING`

目录结构

文件 / 目录	描述
`_config.yml`	保存配置数据。很多配置选项都会直接从命令行中进行设置，但是如果你把那些配置写在这儿，你就不用非要去记住那些命令了。
`_drafts`	drafts 是未发布的文章。这些文件的格式中都没有 `title.MARKUP` 数据。学习如何使用 drafts.
`_includes`	你可以加载这些包含部分到你的布局或者文章中以方便重用。可以用这个标签 `{% include file.ext %}` 来把文件 `_includes/file.ext` 包含

	进来。
`_layouts`	layouts 是包裹在文章外部的模板。布局可以在 YAML 头信息中根据不同文章进行选择。 这将在下一个部分进行介绍。标签 `{{ content }}` 可以将content插入页面中。
`_posts`	这里放的就是你的文章了。文件格式很重要，必须要符合: `YEAR-MONTH-DAY-title.MARKUP` 。 The permalinks 可以在文章中自己定制，但是数据和标记语言都是根据文件名来确定的。
`_data`	Well-formatted site data should be placed here. The jekyll engine will autoload all yaml files (ends with `.yml` or `.yaml`) in this directory. If there's a file `members.yml` under the directory, then you can access contents of the file through `site.data.members` .
`_site`	一旦 Jekyll 完成转换，就会将生成的页面放在这里（默认）。最好将这个目录放进你的 `.gitignore` 文件中。
`index.html` and other HTML, Markdown, Textile files	如果这些文件中包含 YAML 头信息 部分，Jekyll 就会自动将它们进行转换。当然，其他的如 `.html` ， `.markdown` ， `.md` ，或者 `.textile` 等在你的站点根目录下或者不是以上提到的目录中的文件也会被转换。
Other Files/Folders	其他一些未被提及的目录和文件如 `css` 还有 `images` 文件夹， `favicon.ico` 等文件都将被完全拷贝到生成的 site 中。 这里有一些使用 Jekyll 的站点，如果你感兴趣就来看看吧。

YAML头信息

预定义的全局变量

你可以在页面或者博客的头信息处使用一些已经预定义好的全局变量。

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

撰写博客

文章文件夹

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

创建文章的文件

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

```
年-月-日-标题.MARKUP
```

> 在这里，年是4位数字，月和日都是2位数字。MARKUP扩展名代表了这篇文章 是用什么格式写的。下面是一些合法的文件名的例子：

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

内容格式

> 所有博客文章顶部必须有一段 YAML 头信息(YAML front- matter)。 在它下面，就可以选择你喜欢的格式来写文章。Jekyll 支持2种流行的标记语言格式： Markdown 和 Textile. 这些格式都有自己的方式来标记文章中不 同类型的内容，所以你首先需要熟悉这些格式并选择一种最符合你需求的。
>
> Be aware of character sets
>
> Content processors can modify certain characters to make them look nicer. For example, the smart extension in Redcarpet converts standard, ASCII quotation characters to curly, Unicode ones. In order for the browser to display those characters properly, define the charset meta value by including `<meta charset="utf-8">` in the `<head>` of your layout.

引用图片和其它资源

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

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

在文章中引用一个图片

```
... 从下面的截图可以看到：
```

```
![有帮助的截图]({{ site.url }}/assets/screenshot.jpg)
```

链接一个读者可下载的 PDF 文件：

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

- **提示™**: 链接只使用站点的 `根 URL`

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

文章的目录

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

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

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

文章摘要

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

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

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

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

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

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

高亮代码片段

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

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

将输出下面的效果：

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

- **提示™**：显示行数

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

使用草稿

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

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

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

创建页面

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

- **主页**

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

- **提示™**: 在主页上使用布局

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

- **其它的页面的位置**

  > 将 HTML 文件放在哪里取决于你想让它们如何工作。有两种方式可以创建页面：
  > `命名 HTML 文件`：将命名好的为页面准备的 HTML 文件放在站点的根目录下。
  > `命名文件夹`：在站点的根目录下为每一个页面创建一个文件夹，并把 index.html 文件放在每个文件夹里。 这两种方法都可以工作（并且可以混合使用），它们唯一的区别就是访问的 URL 样式不同。

- **命名 HTML 文件**

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

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

- **命名一个文件夹并包含一个 `index.html` 文件**

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

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

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

常用变量

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

- 页面(page)变量

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

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

- 分页器(Paginator)

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

> **分页器变量的可用性**
> 这些变量仅在首页文件中可以，不过他们也会存在于子目录中，就像 `/blog/index.html`。

Jekyll定制

过滤器

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

标签

- **引用**

如果你需要在多个地方引用一小代码片段，可以使用`include`标签。

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

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

- ProTip™: Use variables as file name

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

你还可以传递参数：

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

这些变量可以通过 Lquid 调用：

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

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

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

使用代码高亮的例子如下：

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

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

- **行号**

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

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

- **代码高亮的样式**

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

- **Post URL**

如果你想使用你某篇文章的链接，标签 `post_url`可以满足你的需求。

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

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

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

当使用 post_url标签时，不需要写文件后缀名。

还可以用 `Markdown` 这样为你的文章生成超链接：

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

Gist

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

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

你还可以配置 `gist` 的文件名，用以显示：

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

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

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

私有的 gist 同样支持文件名。

永久链接

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

永久链接的模板用以冒号为前缀的关键词标记动态内容，比如 `date` 代表

```
/:categories/:year/:month/:day/:title.html
```

- **模板变量**

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

- **已经建好的链接类型**

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

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

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

分页功能

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

分页功能只支持 HTML 文件

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

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

  paginate: 5
  下边是对需要带有分页页面的配置：

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

与 `paginator` 相同的属性

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

不支持对“标签”和“类别”分页

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

- **生成带分页功能的文章**

接下来要做的事情就是展现在页面上了，下边是一个简单的例子：

```
---
layout: default
title: My Blog
---

<!-- 遍历分页后的文章 -->
{% for post in paginator.posts %}
  <h1><a href="{{ post.url }}">{{ post.title }}</a></h1>
  <p class="author">
    <span class="date">{{ post.date }}</span>
  </p>
  <div class="content">
    {{ post.content }}
  </div>
{% endfor %}

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

注意首尾页

> Jekyll 没有生成文件夹 ‘page1’ ，所以上边的代码有 bug ，下边的代码解决了这个问题。

下边的 HTML 片段是第一页，他除自己外，为每个页面生成了链接。

```
{% if paginator.total_pages > 1 %}
<div class="pagination">
  {% if paginator.previous_page %}
    <a href="{{ paginator.previous_page_path | prepend: site.baseur
l | replace: '//', '/' }}">« Prev</a>
  {% else %}
    <span>« Prev</span>
  {% endif %}

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

  {% if paginator.next_page %}
    <a href="{{ paginator.next_page_path | prepend: site.baseurl |
replace: '//', '/' }}">Next »</a>
  {% else %}
    <span>Next »</span>
  {% endif %}
</div>
{% endif %}
```

GitHub Pages

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

将 Jekyll 部署到 Github Pages 上

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

用户和组织的站点

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

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

自定义域名不影响仓库命名

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

项目的站点

> 不同于用户和组织的站点，项目的站点文件存放在项目本身仓库的 `gh-pages` 分支中。该分支下的文件将会被 Jekyll 处理，生成的站点会被 部署到你的用户站点的子目录上，例如 `username.github.io/project` （除非指定了一个自定义的域名）。
> Jekyll 项目本身就是一个很好的例子，Jekyll 项目的代码存放在 `master` 分支， 而 Jekyll 的项目站点（就是你现在看见的网页）包含在同一仓库的 `gh-pages` 分支中。

项目站点的网址结构

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

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

GitHub Pages 的文档，帮助和支持

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

---

http://jekyllbootstrap.com/