目　录

致谢

当前文档 《GitBook文档（中文版）》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-05-27。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常生活、工作和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地址：http://www.bookstack.cn/books/gitbook-documentation

书栈官网：http://www.bookstack.cn

书栈开源：https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

介绍

GitBook文档（中文版）

这本书包含了GitBook的所有文档（平台和工具链）

开源工具链

这个工具链（GitBook）是一个使用 Git 和 Markdown 来构建书籍的工具。它可以将你的书输出很多格式：PDF，ePub，mobi，或者输出为静态网页。

GitBook工具链是开源并且完全免费的，它的源码可以在 GitHub 上获取。

与格式和工具链相关的问题被发表在 github.com/GitbookIO/gitbook/issues。

GitBook.com

GitBook.com 是一个使用工具链来创建和托管书籍的在线平台（www.gitbook.com）。

其他文档

开发文档（API & 插件）地址 developer.gitbook.com。

企业版本的安装向导和手册的地址 hep.enterprise.gitbook.com。

帮助和支持

我们很乐意帮助你的书本摆脱困难或者解决任何其他你可能遇到的问题。你可以通过这个表单 gitbook.com/contact 来提问题或者报告一个错误。

贡献文档

你可以贡献自己的一份力量来完善这个托管在在 `GitHub` 上的文档。

来源(书栈小编注)

> *https://github.com/chrisniael/gitbook-documentation*

格式

格式

格式主要注重简单和易读性

GitBook约定了下面这些文件的作用:

- README: 书本的介绍
- SUMMARY: 章节结构
- LANGS: 多语言书籍
- GLOSSARY: 术语描述的清单

至少需要一个 `README` 和 `SUMMARY` 文件来构建一本书。

输出

输出格式

GitBook可以把你的书本生成为不同格式的电子书。

静态网站

这是默认的格式。它生成一个可交互的静态站点。

PDF (Portable Document Format)

Portable Document Format (PDF) 是一以一种独立于软硬件，以及操作系统的方式来保存文档的格式。这是一种很普遍的格式。文件拥有的扩展名为 `.pdf` 。

ePub (electrontic publication)

EPUB (electrontic publicaton的简称，有时称它为epub) 是一个由国际电子出版物论坛 (IDPF) 制定的免费并开放的电子书标准。文件拥有的扩展名为 `.epub` ，苹果和谷歌的设备支持ePub格式。

Mobi (Mobipocket)

Mobipocket电子书格式是基于使用XHTML的开放电子书标准，并且可以包含JavaScript以及框架。亚马逊的设备 (Kindle) 支持这样的格式。

自述和介绍

自述和介绍

书本的第一页内容是从文件 `README.md` 中提取的。如果这个文件名没有出现在 `SUMMARY` 中，那么它会被添加为章节的第一个条目。

使用其他文件替代README.md

一些托管在GitBook上的书更加喜欢将README.md文件作为项目的介绍而不是书的介绍。

从GitBook `>2.0.0` 起，就可以在 `book.json` 中定义某个文件作为README。

```
{
    "structure" : {
        "readme" : "myIntro.md"
    }
}
```

章节和子章节

章节和子章节

GitBook使用文件 `SUMMARY.md` 来定义书本的章节和子章节的结构。文件 `SUMMARY.md` 被用来生成书本内容的预览表。

`SUMMARY.md` 的格式是一个简单的链接列表，链接的名字是章节的名字，链接的指向是章节文件的路径。

子章节被简单的定义为一个内嵌于父章节的列表。

简单的例子

```
# 概要

* [章节 1](chapter1.md)
* [章节 2](chapter2.md)
* [章节 3](chapter3.md)
```

包含子章节的例子

```
# 概要

* [卷 I](part1/README.md)
    * [写作很棒](part1/writing.md)
    * [GitBook很酷](part1/gitbook.md)
* [卷 II](part2/README.md)
    * [期待反馈](part2/feedback_please.md)
    * [更好的写作工具](part2/better_tools.md)
```

Markdown

Markdown

GitBook默认使用Markdown语法。

下面这些可以作为一个快速参考和展示。更多完整的信息，请参考 John Gruber's original spec 和 Github-flavored Markdown info page。

标题

```
# H1
## H2
### H3
#### H4
##### H5
###### H6
```

对于H1和H2，有下划线样式可供选择：

```
Alt-H1
======

Alt-H2
------
```

强调

```
强调，又叫做斜体，使用 *星号* 或 _下划线_。

重点强调，又叫做粗体，使用 **星号** 或 __下划线__。

使用 **星号和_下划线_** 组合使用强调。

删除线使用两个波浪线。~~划掉这个~~
```

列表

在这个例子中，前导空格和尾部空格显示为圆点 ： ·)

```
1. 有序列表的第一项
2. 另外一个项
..* 无序子列表
1. 事实上序号不起作用，那只是一个数字而已
..1. 有序子列表
4. 最后一个项

...你可以适当的缩紧列表项中的段落。注意上面的空行和前导空格（至少一个，但是这里我们使用三个来对齐原始的Markdown内容）。

...换行而不形成段落，你需要使用两个尾部空格。..
...注意这行是分开的，但还在同一个段落中。..
...（这个违背了不需要尾部空格的典型的GFM换行行为）。

* 无序列表可以使用星号
- 或者减号
+ 或者加号
```

链接

有两种创建链接的方式。

```
[内嵌式链接](https://www.google.com)

[带标题的内嵌式链接](https://www.google.com "谷歌的主页")

[引用式链接][arbitrary case-insensitive reference text]

```

```
[相对引用一个库文件](../blob/master/LICENSE)

[你可以在引用式链接定义中使用数字][1]

或者空着什么都不写 [link text itself]

用来说明引用链接的文字可以放在后面。

[arbitrary case-insensitive reference text]: https://www.mozilla.org
[1]: http://slashdot.org
[link text itself]: http://www.reddit.com
```

脚注

Markdown默认使用的脚注样式链接不会在页面中显示。有时包含一个读者可见的非超链接注脚很有用。对于这样的注脚，GitBook支持的一种简单的语法。

```
Text prior to footnote reference.[^2]
[^2]: Comment to include in footnote.
```

图片

```
这是我们的logo（停留查看标题）

内嵌式
![alt text](https://github.com/adam-p/markdown-here/raw/master/src/common/images/icon48.png "Logo Title Text 1")

引用式
![alt text][logo]

[logo]: https://github.com/adam-p/markdown-here/raw/master/src/common/images/icon48.png "Logo Title Text 2"
```

代码和语法高亮

代码块是Markdown规范的一部分，但是语法高亮不是。然而，很多渲染器，比如GitHub的和这里的*Markdown*，都支持语法高亮。支持那种语言以及语言的名字应该怎样写随渲染器的不同而变化。这里的*Markdown*支持许语言；查看完整的列表，以及如何写语言的名字，请参考 highlight.js 演示页.

```
1. 内嵌 `代码` 有 `反引号` 包含它.
```

代码块要么被包含三个反引号 ``` 的行包围，要么被四个空格缩进。推荐仅仅使用被包围的代码块，它们使用方便并且只有它们支持语法高亮。

```
1. ```javascript
2. var s = "JavaScript语法高亮";
3. alert(s);
4. ```
5.
6. ```python
7. s = "Python语法高亮"
8. print s
9. ```
10.
11. ```
12. 没有指明语言，所有没有语法高亮。
13. 让我们随便写个标签试试 <b>tag</b>
14. ```
```

表格

表格不是Markdown规范的核心部分，但是它是GFM的一部分，这里的*Markdown*也支持它。

```
1. 冒号可以用来对其列。
2.
3. | Tables        | Are           | Cool  |
4. | ------------- |:-------------:| -----:|
5. | col 3 is      | right-aligned | $1600 |
6. | col 2 is      | centered      |   $12 |
7. | zebra stripes | are neat      |    $1 |
8.
9. The outer pipes (|) are optional, and you don't need to make the raw Markdown line up prettily.
   You can also use inline Markdown.
10.
11. 外部的管道符 (|) 是可选的，而且不需要优雅的排列Markdown。你还可以在表格中内嵌其他Markdown。
12.
13. Markdown | Less | Pretty
14. --- | --- | ---
15. *Still* | `renders` | **nicely**
16. 1 | 2 | 3
```

块引用

```
> 在邮件中块引用中很方便用来仿真文本的回复。
> 这行是同一个块的一部分。

引用结束

> 当这行很长的文字被包裹的时候，它依然会被恰当的引用。让我们继续写下去来确保包裹它时对于每个人来说它足够长。你可以*在*块引用中使用其它**Markdown**。
```

内嵌HTML

You can also use raw HTML in your Markdown, and it’ll mostly work pretty well.

你同样可以在Markdown中使用HTML，并且它能很好的工作。

```
<dl>
  <dt>Definition list</dt>
  <dd>Is something people use sometimes.</dd>

  <dt>Markdown in HTML</dt>
  <dd>Does *not* work **very** well. Use HTML <em>tags</em>.</dd>
</dl>
```

水平线

```
三个或者更多...

---

连字符

***

星号

___

下划线
```

换行符

关于学习换行符是如何工作的，我的建议是去亲身实践并总结 — 敲击 键一次（也就是插入一个换行符），然后再敲击它两次（也就是插入两个换行），看一下发生了什么。不久你就能学会它。”Markdown Toggle” 是你的朋友。

这里有一些东西可以尝试一下：

```
1. 我们以这行作为开始。
2.
3. This line is separated from the one above by two newlines, so it will be a *separate paragraph*.
4.
5. 这行与上面那行被两个换行符分隔，所以它会成为一个 *单独的段落*。
6.
7. This line is also a separate paragraph, but...
8. This line is only separated by a single newline, so it's a separate line in the *same
   paragraph*.
9.
10. 这行同样是一个单独的段落，但是...
11. 这行仅仅被一个换行符分隔，所以它是一个在 *同一段落* 中的单独的行。
```

Youtube视频

视频不能被直接添加，但你可以添加一个链接至视频的图片，像这样：

```
1. <a href="http://www.youtube.com/watch?feature=player_embedded&v=YOUTUBE_VIDEO_ID_HERE
2. " target="_blank"><img src="http://img.youtube.com/vi/YOUTUBE_VIDEO_ID_HERE/0.jpg"
3. alt="IMAGE ALT TEXT HERE" width="240" height="180" border="10" /></a>
```

或者，使用纯Markdown，但是会丢失掉图片的大小和边框：

```
1. [![IMAGE ALT TEXT HERE](http://img.youtube.com/vi/YOUTUBE_VIDEO_ID_HERE/0.jpg)]
   (http://www.youtube.com/watch?v=YOUTUBE_VIDEO_ID_HERE)
```

AsciiDoc

AsciiDoc

从 `2.0.0` 版本起，GitBook也能支持AsciiDoc作为输入的格式了。

关于这个格式的更多信息请参考 AsciiDoc Syntax Quick Reference。

和markdown一样，GitBook也使用了一些特殊的文件来提取书本的结构： `README.adoc` ， `SUMMARY.adoc` 和 `GLOSSARY.adoc` 。

README.adoc

这是书本的最重要项目：介绍。这个文件是必须的。

SUMMARY.adoc

这个文件定义了章节和子章节的列表。和 markdown 文件一样， `SUMMARY.adoc` 的格式是简单的链接列表，链接的名字是章节的名字，链接的目标是章节文件的路径。

子章节被简单的定义成添加到父章节的内嵌列表。

```
= Summary

. link:chapter-1/README.adoc[Chapter 1]
.. link:chapter-1/ARTICLE1.adoc[Article 1]
.. link:chapter-1/ARTICLE2.adoc[Article 2]
... link:chapter-1/ARTICLE-1-2-1.adoc[Article 1.2.1]
. link:chapter-2/README.adoc[Chapter 2]
. link:chapter-3/README.adoc[Chapter 3]
. link:chapter-4/README.adoc[Chapter 4]
.. Unfinished article
. Unfinished Chapter
```

LANGS.adoc

对于 多语言 书，这个文件是用来定义支持的语言和翻译。

这个文件遵循 `SUMMARY.adoc` 一样的语法：

```
= Languages

. link:en/[English]
. link:fr/[French]
```

GLOSSORY.adoc

这个文件是用来定义术语的。请查看 术语表 那章节。

```
= Glossary

== Magic
Sufficiently advanced technology, beyond the understanding of the observer producing a sense of wonder.

== PHP
+A popular web programming language, used by many large websites such as Facebook. Rasmus Lerdorf originally created PHP in 1994 to power his personal homepage (PHP originally stood for "Personal Home Page" but now stands for "PHP: Hypertext Preprocessor").
```

封面

封面

为了让你的书本在GitBook上更加优雅，你可以指定一个封面。

封面由 **`cover.jpg`** 文件指定，**`cover_small.jpg`** 同样可以作为小版本封面存在。封面应该是 **`JPEG`** 格式的文件。

最佳尺寸

	大	小
文件	`cover.jpg`	`cover_small.jpg`
大小（像素）	1800x2360	200x262

指南

一个好的封面遵循下面的指南：

- 没有边框
- 清晰可见的书本标题
- 任何重要的文字在小版本中应该可见

自动封面

GitBook插件（ `autocover` ）同样可以用来为你生成一个封面文件，或者仅仅从大封面中生成 `cover_small.jpg` 。这个插件默认会被添加到托管的书本中。

阅读更多关于autocover的信息

多语言

多语言

GitBook支持使用多语言来构建书本。按照GitBook的标准格式，每个语言应该作为一个子目录，命名为 `LANGS.md` 的文件应该遵循下面的格式并出现在仓库的根目录下：

```
1. * [英语](en/)
2. * [法语](fr/)
3. * [西班牙语](es/)
```

你可以从 `Learn Git` 这本书中看到一个完整的例子。

术语表

术语表

允许你指定术语并且在术语表中显示它们各自的定义。基于这些术语，GitBook会自动建立索引并高亮这些在文中的术语。

`GLOSSORY.md` 的格式非常简单：

```
# 术语
这个术语的定义

# 另外一个术语
它的定义可以包含粗体和其他所有类型的内嵌式标记...
```

模板

模板

这是GitBook可使用的模板特性的简要概述。GitBook使用 Nunjucks 和 Jinja2 的语法。

转义

如果你想要输出任何特殊的目标标签，你可以使用raw，任何在其中的内容都会原样输出。

```
{% raw %}
  这 {{ 不会被处理 }}
{% endraw %}
```

变量

变量会从书本内容中寻找对应的值。

变量被定义在 `book.json` 文件中：

```
{
    "variables": {
        "myVariable": "Hello World"
    }
}

#### 显示变量

定义在 `book.json` 中的变量可以在 `book` 作用域下被访问：
```

{{ book.myVariable }}

```

这会从书本的变量中寻找 `myVariable` 并显示它。变量的名字可以存在点（dot）来查找属性。你同样可以使用方括号语法。
```

```
{{ book.foo.bar }}
{{ book[“bar”] }}
```

```
1.
2. 如果对应的值没有定义，那么什么也不会显示。下面这些语句不会输出任何东西，如果 `foo` 没有定义的话：`{{ book.foo }}`, `{{ book.foo.far }}`, `{{ book.foo.bar.baz }}`。
3.
4. #### 上下文变量
5.
6. 一些变量也可以用来获取当前文件或GitBook实例的信息。
7.
8. | Name | Description |
9. | ---- | ----------- |
10. | `file.path` | Path of the file relative to the book |
11. | `file.mtime` |Last modified date of the file |
12.
13. ## 标签
14.
15. 标签是在章节和模板中执行操作的特殊块
16.
17. #### If
18.
19. **if** 测试一个条件并让你选择性的显示内容。它的行为的和编程语言中的if一样。
20.
21. `
```

```
{% if variable %}
It is true
{% endif %}
```

```
1.
2. 如果 `variable` 被定义了并且是真的，那么 "It is true" 就会被显示出来。否则，没有任何东西会被显示。
3.
4. 你可以使用elif和else来指定选择性条件：
```

```
{% if hungry %}
I am hungry
{% elif tired %}
I am tired
{% else %}
I am good!
{% endif %}
```

```
1.
```

```
#### for

**for** 迭代数组和字典。

让我们来看一下 `book.json` 中的变量：
```

```
{
“variables”: {
“authors”: [
{ “name”: “Samy” },
{ “name”: “Aaron” }
]
}
}
```

```

```

Authors

{% for author in book.authors %}

- {{ author.name }}
  {% endfor %}
  ```

上面的例子使用 `authors` 数组的每项的 `name` 属性作为显示的值，列出了所有的作者 (author) 。

## include

Include 的详细描述在文章 内容引用 中。
  ```

内容引用

内容引用

内容参考（conref）是便于用来重复使用其他文件和书本内容。

导入本地文件

使用 `include` 标签导入其他文件的内容真的很简单：

```
1. {% include "./test.md" %}
```

从其他书本导入文件

GitBook 同样能处理使用了git协议的include路径：

```
1. {% include "git+https://github.com/GitbookIO/documentation.git/README.md#0.0.1" %}
```

git的url格式是：

```
1. git+https://user@hostname/project/blah.git/file#commit-ish
```

真实的git url应该以 `.git` 结尾，导入的文件名从 `.git` 之后的url片段提取。

`commit-ish` 可以是任何可以作为 `git checkout` 参数的标签，sha，或分支。默认是 `master` 。

继承

模板继承是一种重复使用模板的简单方式。当写完一个模板，你可以定义 “block” 让字模板来替换。继承链可以任意长。

`block` 在模板中定义了一个区域并用一个名字标识了它。基类模板可以指定一些块，而子类可以用新的内容替换它们。

```
{% extends "./mypage.md" %}

{% block pageContent %}
# This is my page content
{% endblock %}
```

在文件 `mypage.md` 中，你应该指定用来替换内容的块。

```
{% block pageContent %}
This is the default content
{% endblock %}

# License

{% import "./LICENSE" %}
```

忽略文件和文件夹

忽略文件和文件夹

GitBook会读取 `.gitignore` , `.bookignore` , `.ignore` 文件来获取忽略的文件和目录的列表。

这些文件的格式，遵循和 `.gitignore` 同样的约定：

```
# 这是一个注释

# 忽略文件test.md
test.md

# 忽略文件夹 "bin" 中的所有内容
bin/*
```

配置

配置

所有的配置都以JSON格式存储在名为 `book.json` 的文件中。

你可以粘贴你的 `book.json` 去 jsonlint.com 验证JSON语法。

字段

gitbook

```
1. { "gitbook": ">=2.0.0" }
```

这个选项是用来探测用来生成书本的GitBook的版本的。格式是一个 SEMVER 条件。

在 **gitbook.com** 中，这个值是根据平台中输入的标题定义的。

description

```
1. { "description": "This is such a great book!" }
```

这个选项定义了书本的描述，默认是从 **README**（第一段）中提取的。

在 **gitbook.com** 中，这个值是根据平台输入的描述定义的。

isbn

```
{ "isbn": "978-3-16-148410-0" }
```

这个选项定义了你书本的ISBN。

language

```
{ "language": "fr" }
```

这个选项定义了你书本的语言，默认值是 `en` 。

这个值是用来做国际化和本地化的，它改变网站的文字。

在 **gitbook.com** 中，这个值是根据探测到的语言或指定的设置定义的。

direction

```
{ "direction": "rtl" }
```

这个选项是用来重新设置语言的文字方向的。建议将 `language` 字段设置为带有正确的文字方向的语言。

styles

这个选项是用来自定义书本的css的。

例子:

```
{
    "styles": {
        "website": "styles/website.css",
        "ebook": "styles/ebook.css",
        "pdf": "styles/pdf.css",
        "mobi": "styles/mobi.css",
        "epub": "styles/epub.css"
    }
}
```

plugins

```
{ "plugins": ["mathjax"] }
```

书本使用的插件列表被定义在 `book.json` 的配置中。

pluginsConfig

```
{
    "plugins": ["myplugin"],
    "pluginsConfig": {
        "myPlugin": {
            "message": "Hello World"
        }
    }
}
```

structure

这个选项是用来覆盖GitBook使用的路径的。

例如你想要使用 `INTRO.md` 代替 `README.md`：

```
{
    "structure": {
        "readme": "INTRO.md"
    }
}
```

variables

```
{
    "variables": {
        "myTest": "Hello World"
    }
}
```

这个选项定义在 模板 中使用的变量值。

插件

插件

插件是扩展 GitBook 功能（电子书和网站）最好的方式。现在插件可以做很多事：支持数学公式的显示，使用 Google Analytic 追踪访问，…

如何查找插件？

可以在 plugins.gitbook.com 上很轻松的查找插件。

如何安装插件？

当你找到你要安装的插件后，你需要将它们添加至 `book.json` 中：

```
{
"plugins": ["myPlugin", "anotherPlugin"]
}
```

你可以使用：`"myPlugin@0.3.1"` 来指定一个特定版本，当你使用旧版本的 GitBook 时，这个很有用。

如果你打算本地 build 你的书，可以运行 `gitbook install` 来下载和准备插件。

书本

扩展网站和电子书样式

你可以通过在 `styles` 目录中建立文件来扩展电子书和网站使用的css:

类型	文件
`website`	`styles/website.css
`pdf`	`styles/pdf.css`
`epub`	`styles/epub.css`
`mobi`	`styles/mobi`
`pdf`, `epub`, `mobi`	`styles/ebook.css

这些路径可以在 `book.json` 设置中改变，例如使用根文件夹中的文件:

```
{
    "styles": {
        "website": "website.css",
        "ebook": "ebook.css",
        "pdf": "pdf.css",
        "mobi": "mobi.css",
        "epub": "epub.css"
    }
}
```

使用 CSS 预处理

这些插件是用来让你更简单的使用 CSS 预处理:

- styles-less
- styles-sass

数学 & TeX

数学 & Tex

GitBook 可以使用插件支持数学公式和 Tex。当前有两个官方的插件用来显示数学公式：mathjax 和 katex。

开启数学插件

为了开启数学公式支持，你需要选择（mathjax 或 katex）一个插件添加到 `book.json` 中：

```
{
    "plugins": ["mathjax"]
}
```

MathJax 和 KaTeX 的区别

`mathjax` 和 `katex` 插件是 Tex 公式绘制的不同实现，它们基于各自的开源库：KaTeX 和 MathJax 。

MathJax 支持整个 Tex 语法，但是在制作电子书版本时不是很完美。
KaTex 在所有格式（网页和电子书）的绘制上都很完美，但是还不支持 所有的语法。

添加数学公式

```
这里是一些内嵌的数学公式：$$a \ne 0$$
```

数学公式块需要另起新行：

```
这里是一个数学公式块

$$
a \ne 0
```

```
5. $$
```

构建

构建和历史记录

在你使用**git**或**editor**推送完内容后，GitBook.com就会开始不同的构建：

- 网站：它会生成网站
- **json**：它会提取书本的元数据（概要，介绍，等等）
- **epub**：它会生成epub格式的下载文件
- **pdf**：它会生成pdf格式的下载文件

这些构建的记录和详情可以在你书本的 **History** 选项卡中可以看到。

触发构建失败

如果你的书本链接到了GitHub，但GitHub并没有通知GitBook内容改变了（例如：授权被重置了）。关于如何修复这个问题的内容在 集成GitHub章节。

修复构建错误

如果你构建失败，你可以使用日志来调试问题并发布修复后的内容。

阅读更多关于常见的构建错误的内容

使用GIT更新

使用Git更新书本

当你在**gitbook.com**上创建了书本后，你需要推送一些内容给它。你可以使用网页编辑器或者命令行来做这件事。

如果你想要通过命令行来更新你的书本的话，你可以使用 GIT 来推送你的内容。

GIT Url

每本书本都有一个相关联的Git HTTPS url。GitBook的git服务器暂时还不支持ssh协议。

git url的格式是:

```
1. https://git.gitbook.com/{{UserName}}/{{Book}}.git
```

认证

git服务器使用你基本的GitBook登录来认证你。当提示的时候，输入你的GitBook用户名和密码（你同样可以使用你的API token）。

保存你的凭证

To avoid having to enter your password on each new push, you can add your GitBook credentials to your `~/.netrc` file. Create or append to an existing `~/.netrc` file something like below:
为了避免每次 push 的时候输入密码，你可以将你的 GitBook 凭证添加到 `~/.netrc` 文件里。将下面内容添加到 `~/.netrc` 文件中:

```
1. machine git.gitbook.com
```

```
2.  login 用户名或邮箱
3.  password API-TOKEN或密码
```

为了安全起见，我们推荐你使用 **API TOKEN**，你可以在 in your settings under “API” 找到它。

在命令行创建一个新的仓库

```
1. touch README.md SUMMARY.md
2. git init
3. git add README.md SUMMARY.md
4. git commit -m "first commit"
5. git remote add gitbook https://git.gitbook.com/{{UserName}}/{{Book}}.git
6. git push -u gitbook master
```

推送一个已存在的仓库

```
1. git remote add gitbook https://git.gitbook.com/{{UserName}}/{{Book}}.git
2. git push -u gitbook master
```

常见错误

常见错误

下面是常见错误的列表以及对应的修复方法。

```
Error loading plugins: plugin1, ...
```

发生这个错误是因为不能解析这个插件（或者这个插件是无效的）。外部插件需要使用 `gitbook install` 来安装。

一些插件不能加载的原因还有可能是因为他们需要另外一个版本的GitBook。如果是这样的话，你需要找到正确的插件版本来支持你使用的GitBook版本，并在 `book.json` 中定义它。

```
{
    "plugins": ["myPlugin@1.2.3"]
}
```

```
Need to install ebook-convert from Calibre
```

当你尝试把你的项目构建成为PDF，ePub或者mobi格式的电子书时，为了避开错误，你必须安装了 Calibre 电子书阅读/管理器和命令行工具。

从Mac版的Calibre安装命令行工具时，从菜单中选择: **calibre - Preferences - Miscellaneous - Install command line tools**

一旦完成了这个，你的电子书就会构建成功。

---

ebook-convert

安装ebook-convert

生成电子书 (epub, mobi, pdf) 时需要ebook-convert。

Mac

下载 Calibre.app。移动 `calibre.app` 到你的应用程序文件夹中后，给 `ebook-convert` 工具创建一个符号链接。

```
1. $ sudo ln -s ~/Applications/calibre.app/Contents/MacOS/ebook-convert /usr/bin
```

你可以把 `/usr/bin` 替换为 `$PATH` 中的任何的文件夹。

导入

导入文档

将文档导入 GitBook 很简单。当你创建一本新书的时候，使用 `Import` 标签页就可以上传文档。

支持的格式

类型	后缀名
Microsoft Word 文档	`.docx`
DocBook v5.x	`.xml`
HTML 文件	`.html`

为了充分使用它的特性，我们推荐你使用：

- Microsoft Word 2007+ 文档
- DocBook v5.x 文档

If you have existing DocBook documents in version 4, consider converting it automatically to version 5 for optimized compatibility.
如果你有 DocBook v4.x 的文档，可以考虑将它们 转化 成 DocBook v5.x 来解决兼容性问题。

转化

导入文档的特性使用了 gitbook-convert 命令行程序。这个组件负责将原始文档转化到 makrdown 文件，以及 `SUMMARY.md` 的创建。

gitbook-convert 根据文章的结构，将文章划分成章节和子章节。因此，原始文档的一级标题都会被转化成一个章节。如果一个章节包含二级标题，则会为这个章节创建一个目录来包含每个子章节。

在章节目录里，会创建一个包含章节前言的 `README.md` 文件。第一个二级标题前的所有内容会被认为是一个章节的前言。这样的规则同样适用于第一个一级标题前的所有内容，它们会成为为书的前

言。

对于 `.docx` 文档，gitbook-convert 会将文档包含的所有图片导出到 `assets/` 目录中。

如果你需要更多的灵活性，可以考虑在本地使用 gitbook-convert，然后用 Git 或 GitHub 新建一个仓库并导入转化后的内容。

疑难解答

我们很愿意帮忙解决你在使用 GitBook 时遇到的问题。你可以在 github.com/GitbookIO/gitbook-convert 提问或标注一个问题。

集成GitHub

集成GitHub

GitBook完美的集成了 GitHub 来托管你书本的源代码。

连接你的账号/权限

在集成你的书本和GitHub前，你需要授予GitBook访问你的GitHub账号的权限。

在你的 账号设置 里，使用正确的权限连接你的GitHub账号：

- 默认的权限：仅仅在登陆的时候访问你的GitHub账号来验证你
- 访问**webhook**：访问你的GitHub账号来在指定的仓库中创建webhook（查看webhooks）
- 访问公开的仓库：从网页编辑器中访问你的GitHub仓库，你可以很容易的在GitBook中编辑你的书本（仅仅公共仓库）
- 访问私有的仓库：和上面一项目一样，但是只能访问私有仓库

从 GitHub 导入书

创建一本新书的时候， `GitHub` 标签页让你选择一个 GitHub 仓库导入。

新创建的书会使用你仓库的内容，webhook 也会自动添加。

Webhooks

当你的GitHub的仓库改变时，Webhooks会通知GitBook。

如果你的GitHub仓库改变时，GitBook没有收到通知，这个问题的主要原因是webhook。你可以检查仓库设置中webhook的状态。

连接书本和GitHub仓库

当你的GitHub账号正确地连接到你的GitBook账号后，将一本书链接至一个仓库很简单。

警告：当你在你书本的设置中指定了一个GitHub仓库，这个仓库会优先于GitBook的git仓库，这意味着编辑器会直接编辑GitHub中的内容。

1. 打开你书本设置中的GitHub部分
2. 输入你的仓库id（例如：`YourGitHubUserName/RepoName`）
3. 保存你的设置
4. 点击**Add a deployment webhook**按钮

你现在使用网页编辑器来编辑你的GitHub仓库（如果你授权了正确的权限）,并且你在GitHub上的提交会触发GitBook来构建书本。

常见的错误

书本没有被更新/没有任何的构建

如果你连接了你的GitHub仓库至一个GitBook项目，但是编辑它的内容却没有触发GitBook任何的构建操作。验证webhook被正确的添加到了你的GitHub仓库中（GitHub仓库设置->Webhook），如果webhook不存在或无效，之后添加它到你书本的设置中。

改变书本的名字/拥有者

如果你把你的书本转移给了新的拥有者，之前的wehook会失效，你需要之后重新添加它。

迁移内容至GitHub

迁移内容至GitHub

如果你开始在GitBook上写书了，但现在你想把它的源代码托管到GitHub上，不用担心，这很简单：

使用GitHub导入工具

1. 使用GitHub的导入工具：import.github.com/new。
2. 输入你的GitBook的git url，例如：`https://git.gitbook.com/MyName/MyBook.git`（这个url可以在你书本的设置中找到）。
3. 输入你的GitHub仓库。
4. 当提示时输入你的GitBook凭证（你可以使用你的API token代替你的密码）。

当你的内容迁移到GitHub后，你可以建立集成来让GitBook依然能从GitHub中构建你的书本：集成GitHub

使用命令行

在GitHub上创建完仓库后。

注意：这个操作会覆盖你的git历史记录。

```
# Clone你的GitBook仓库（你需要输入你的用户名和密码）
$ git clone https://git.gitbook.com/MyName/MyBook.git ./mybook

# 进入Clone的书本
cd ./mybook

# Push它至GitHub
$ git push https://github.com/username/repo.git master --force
```

编辑器

编辑器

可以使用在线编辑器来编辑你的书本。每次你保存一个文件的时候，都会构建书本一次。如果想要写一份草稿而不构建书本，请参考“草稿式工作流程”。

如何访问编辑器

每本书都可以使用编辑器。在书本的控制面板中，点击“编辑”图标，就会新建一个新标签页来打开编辑器。

编辑器兼容于所有现代的网页浏览器：Google Chrome，Safari，Firefox 和 Internet Explore；确保你使用的浏览器是最新的版本。

草稿式工作流程

草稿式工作流程

每次你保存一个文件（或者当你编辑术语表或概要时），都会触发GitBook构建一次书本。

但是正确的工作流程是，先在草稿上撰写，完成后再构建书本。

1. 从分支菜单中创建一个新的分支
    i. 输入一个描述你修改内容的名字，例如：“第一个草稿”
   ii. 选择”master”作为原始分支
2. 现在激活的分支应该是你刚刚创建的那个
3. 正常编辑你的书本
4. 当草稿完成时，打开分支菜单，点击“合并分支”
5. 合并你的草稿分支至主分支
6. 删除刚刚合并完的草稿分支
7. 完成！

章节

- 章节

章节

你可以在编辑器里简单地右击一个章节并选择 `添加章节` 来添加章节。可以通过拖拽的方式来重新组织章节和子章节。

搜索

搜索GitBook

你可以使用我们强大的搜索工具来在GitBook的上千本书中查找你需要的内容。

当你搜索GitBook的时候，你可以构造一个匹配指定数字和单词的查询语句。

查询文字

GitBook默认会使用查询中的关键字来搜索书本。例如 `javascript angular` 会返回所有包含单词”javascript”和”angular”的书本。

排除结果中特定的单词

你同样可以通过 `NOT` 语法排除单词来缩小你的搜索结果。搜索 `Hello` 会返回一大堆”Hello World”的项目，但是把搜索语句改成 `hello NOT world` 则会返回更高少的结果。

查询特定域的值

你可以通过指定一个特定域的值来筛选书本。例如：`license:apache-2` 会返回所有使用 Apache 2 许可证的书本。

查询小于/大于某个数字的值

你可以分别使用 `>` 和 `>=` 来表示 “大于” 和 “大于等于”。 例如，下面的搜索查询是等价的： `cats stars:">10"` 和 `cats stars:">=11"`

你可以分别使用 `<` 他 `<=` 来表示 “小于” 和 “小于等于”。

可见性

可见性

公共的/私有的

你的书本可以是 公共的，也可以是 私有的。公共的书本对所有人可见，但是只有合作者才可以更改它。私有书本只对合作者可见。

你可以将公共的书本变为私有，也可以将私有书本变成公共的。

收费书本

收费书本只可以是 公共的。

主页/浏览页面

主页和浏览页面仅仅会展示已经成功构建的书本。建议给书本设置一张封面图片。

自定义域名

自定义域名

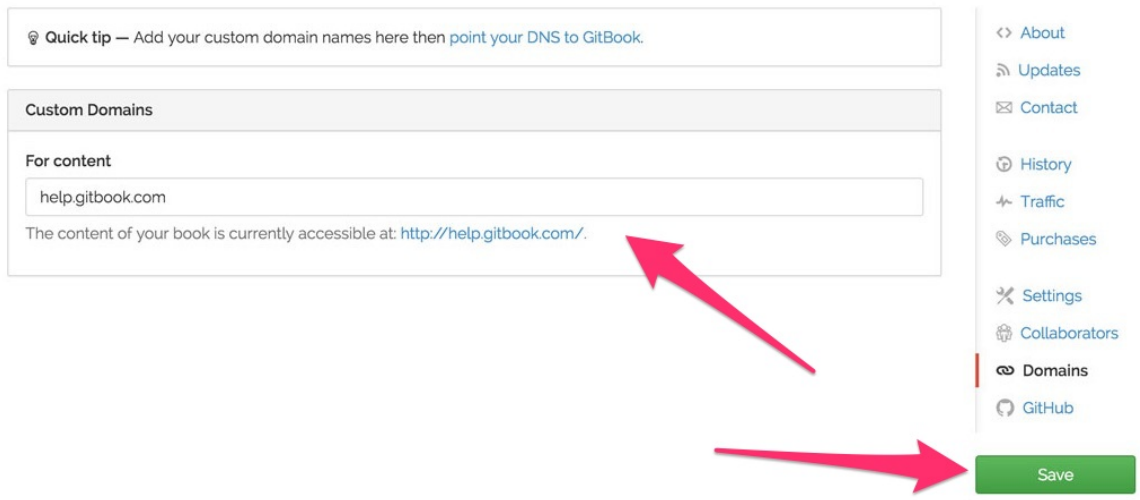
所有在 **Gitbook.com** 上的书的http地址为 `http://{author}.gitbooks.io/{book}/` ，而书内容的地址是 `http://{author}.gitbooks.io/{book}/content/` 。

但是你也可以使用你自定义的域名（GitBook的免费功能）。域名可以绑定到你的主页或者内容上（或两者都）。

很容易就可以添加一个自定义域名。

GitBool.com 设置

前往你书本的 设置 页面，点击 域名。然后输入你的域名并保存。



域名提供商设置

为了完成域名绑定，你需在你的域名提供商那边做一些设置：

1. 登陆你的域名注册商的网站，找到允许你 添加/编辑 主机记录的页面，通常这个页面会在 `编辑DNS` ， `主机记录` 或者 `域文件控制` 的设置里。

2. 设置一个值为 `www` 的 **CNAME** 记录，URL域指向： `www.gitbooks.io` 。

3. 为了 重定向 顶级域名（ `yourdomain.com` ）到 `www.yourdomain.com` 上，你需要开启 “域名转发”。这个功能通常在 `转发` ， `URL 转发` 或者 `URL 重定向` 中（大陆域名提供商已于2009年12月29日起禁用域名转发功能）。

DNS 解析的转播可能需要花上几个小时的时间。

账号

账号

每个 GitBook 用户/作者都拥有一个 GitBook 的账号。你可以使用你的邮箱或社交账号（例如 **Facebook**，**Twitter**，**Google** 或者 **GitHub**）来创建一个账号。

删除账号

你可以在 www.gitbook.com/settings 中删除你的个人账号。

这个操作是 不能 撤销的。它会永久地删除你所有的书籍，构建，和其他事务。一旦你删除了你的账号，就不能恢复了。请三思而行。

账号复制

复制账号

当你使用社交账号（Facebook，Twitter，GitHub 或者 Google）注册时会出现复制账号的界面，然后使用这个表单重新注册

这样子，你就可以创建两个不同的账号（一个关联你的邮箱，另一个关联你的社交账号）。

你可以像下面这样来 删除 一个账号：

1. 从关联你邮箱的账号登出
2. 使用社交账号登录
3. 在 www.gitbook.com/settings 里删除和这个社交账号关联的账号
4. 重新使用你的邮箱登陆
5. 在 www.gitbook.com/settings 里重新将你的社交账号和你的邮箱关联

组织

组织

你可以创建一个新的组织或将现有的个人账号转化为一个组织账号。

组织账号适合企业和那些需要大量成员和管理员的项目。更多关于组织账号如何帮助你合作一个项目的信息，请查看 个人账号与组织账号的区别。

创建一个组织有多种方式。

创建一个组织

当你随便创建一个组织的时候，它还没有他任何书本关联。拥有写权限的组织成员可以在任何时候新建书本，或者转让已存在的书本。

变更个人账号为组织账号

如果你想把你当前账号下所有的仓库变为组织的，你可以变更你的账号为组织账号。

用户的差异

个人账号和组织账号的区别

你的个人账号是你在 GitBook 上的身份。你的个人账号可以成为好多组织的成员，无论这个个人账号加入的免费的计划还是付费计划。

个人账号

每个使用 GitBook 的用户都拥有他们自己的账号。这些账号包括：

- 无限制的公共书本和合作者
- 个人账户可以创建发布私有书本
- 可以无限制的添加仓库和合作者

组织

组织适合那些需要大量拥有者和管理员的企业和大型项目。他们包括：

- 计划创建和发行私有书本的企业
- 基于合作者的访问权限
- 团队模式下无限制的管理员和合作者
- 账单和收据可以被发送到备用（second）邮件地址

转变用户

变更账号为组织

如果你需要为访问书本赋予更多的粒度权限，你可以将你的个人账号转化为组织。

警告：在变更账号为组织之前，请注意下面几点：

- 变更之后，你就不能再登陆这个转化后的账号了。
- 组织账号不能变更为个人账号。

1. 创建一个个人账号

除非你是组织的成员，否则你没有权限访问组织内的书本。因此在变更账号后，你需要创建一个新账号来访问组织的内容。

2. 退出所有已加入的组织

如果你变更的账号已经是某些组织的成员，你首先一定要退出这些组织。

3. 变更账号为组织

- 打开你的 账号设置，点击 `Organizations`。
- 在 `Turn account nto an organization` 下，输入你个人账号的用户名，勾选 `confirm` 并点击 `Go!`。

4. 登陆组织账号

上一步的转化操作中，如果你把你的第二个账号设置成了拥有者，那么登陆这个账号，在账号上下文切换栏，你可以访问你的组织。

转移组织拥有权

转移组织拥有权

为了让其他人成为组织账号的拥有者，你需要添加一个新的拥有者，确保修改了账单信息，然后把你自己从账户中移除。

> 从组织中移除自己的时候没有修改组织账户的账单信息。新拥有者一定要修改账单信息来移除你的信用卡或 *PayPal* 信息。

1. 如果你是拥有者团队唯一的成员，添加另一个组织成员进入这个团队。
2. 联系新的拥有者，确保他或她能访问组织的设置功能。
3. 在你的组织里，如果你是当前负责支付 GitBook 的人话，你还需要修改组织的账单信息来关联新的拥有者。
4. 从组织里移除你自己。

主页主题

主页主题

主题可以使用预定义或自定义的HTML模板来定制你**gitbook.com**上书本的主页。

预定义的主题

GitBook预定义主题是开源的并可以从GitHub获取。

书本主页的变量

变量	类型	描述
`book`	`<book>`	要显示的书本

书本的表示

变量	类型	描述
`id`	`string`	书本的完整的id（例如: `Username/Test` ）
`name`	`string`	书本的名字
`title`	`string`	书本的标题
`description`	`string`	书本的描述
`public`	`boolean`	书本是私有的则为False
`price`	`number`	书本的价格（0 等价于免费）
`githubId`	`string`	书本在GitHub上的ID（例如: `Username/Test` ）
`categories`	`array[string]`	与书本相关的标签的列表
`author`	`<user>`	创建书本的用户
`collaborators`	`array[]	书本的合作者的数组（不包括作者）
`cover.small`	`string`	封面的Url（小尺寸）

`cover.large`	`string`	封面的Url（大尺寸）
`urls.access`	`string`	书本主页的Url
`urls.homepage`	`string`	主页的Url（使用定制的域名）
`urls.read`	`string`	阅读书本的Url
`urls.reviews`	`string`	评论页面的Url
`urls.subscribe`	`string`	提交邮件订阅的Url
`urls.download.epub`	`string`	下载电子书的Url（EPUB格式）
`urls.download.mobi`	`string`	下载电子书的Url（Mobi格式）
`urls.download.pdf`	`string`	下载电子书的Url（PDF格式）

用户的表示

变量	类型	描述
`username`	`string`	用户的用户名
`name`	`string`	用户的的名字
`urls.profile`	`string`	主页设置的Url
`urls.avatar`	`string`	头像的Url
`accounts.twitter`	`string`	Twitter上的用户名（如果关联了的话）
`accounts.github`	`string`	GitHub上的用户名（如果关联了的话）

Webhooks

Webhooks

Webhooks 提供了一种任何时候只要有操作作用于书上就通知外部服务器的方式。

很多作用于书本的操作都会触发 Webhooks。

为了建立一个新的 webhook，你需要访问外部服务器，并且熟悉技术规格。如果需要帮助，请访问开发者指南，它包含了你可以关联的所有的操作。

配置 webhook

你可以在书本的设置里配置 webhook：