

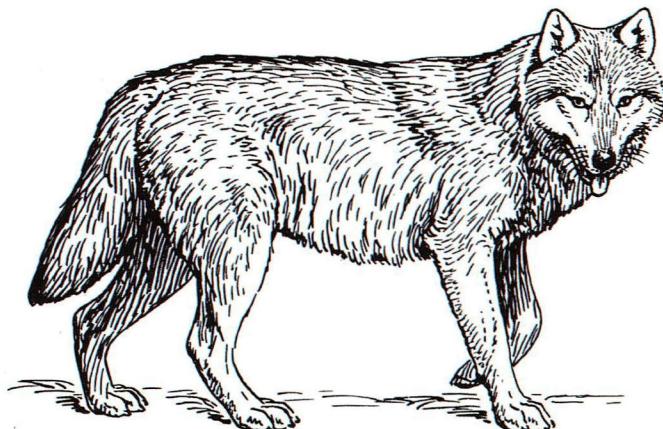


Flask开发团队成员撰写，Flask开发团队核心维护者高度评价并推荐

基于全新Flask技术版本，从基础知识到进阶实战，再到源码分析，提供完善的Flask学习路径



华章 IT



Flask Web开发实战

入门、进阶与原理解析

PYTHON WEB DEVELOPMENT WITH FLASK

李辉 著

机械工业出版社
China Machine Press





| 内容简介 |

本书由Flask官方团队的开发成员撰写，得到了Flask项目核心维护者的高度认可。

内容上，本书从基础知识到进阶实战，再到Flask原理和工作机制解析，涵盖完整的Flask Web开发学习路径，非常全面。

实战上，本书从开发环境的搭建、项目的建立与组织到程序的编写，再到自动化测试、性能优化，最后到生产环境的搭建和部署上线，详细讲解完整的Flask Web程序开发流程，用5个综合性案例将不同难度层级的知识点及具体原理串联起来，让你在开发技巧、原理实现和编程思想上都获得相应提升。

技术上，不仅Flask使用的是新发布的稳定版，而且连相关的Python工具包使用的也是全新的版本，同时对这些技术未来可能有变化的地方进行了说明，以此确保本书内容在一定时间内不会过时。

本书共16章，分为三部分。

第一部分 基础篇（第1~6章）

通过大量的程序实例详细介绍了Flask的所有基础知识，同时在每章中提供了一些进阶技巧，供进阶读者学习。

第二部分 实战篇（第7~11章）

通过5个真实的项目案例来串联和阐释不同的知识点，难度逐渐递增。这5个案例分别为留言板SayHello、个人博客Bluelog、图片社交网站Albumy、待办事项程序Todoism和聊天室CatChat。

第三部分 进阶篇（第12~16章）

首先介绍了Flask程序的部署流程，如测试、性能优化和部署上线；然后通过一个实例讲解了Flask扩展的开发，最后以源代码为切入点深入剖析了Flask的实现原理与主要工作机制。





Flask Web开发实战

入门、进阶与原理解析

PYTHON WEB DEVELOPMENT WITH FLASK

李辉 著

 机械工业出版社
China Machine Press





图书在版编目 (CIP) 数据

Flask Web 开发实战：入门、进阶与原理解析 / 李辉著. —北京：机械工业出版社，2018.8
(Web 开发技术丛书)

ISBN 978-7-111-60659-8

I. F… II. 李… III. 软件工具 - 程序设计 IV. TP311.561

中国版本图书馆 CIP 数据核字 (2018) 第 183791 号

Flask Web 开发实战：入门、进阶与原理解析

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：李 艺

责任校对：李秋荣

印 刷：北京市荣盛彩色印刷有限公司

版 次：2018 年 9 月第 1 版第 1 次印刷

开 本：186mm×240mm 1/16

印 张：44

书 号：ISBN 978-7-111-60659-8

定 价：129.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88379426 88361066

投稿热线：(010) 88379604

购书热线：(010) 68326294 88379649 68995259

读者信箱：hzit@hzbook.com

版权所有 · 侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光 / 邹晓东





Preface 前言

Flask 是目前最流行的 Python Web 框架之一。自 2010 年开源以来，Flask 受到了越来越多的 Python 开发者的喜欢，其受欢迎程度不输于 Django。截至 2018 年 6 月，它在 GitHub 上已有近 36 000 个 Star，2000 多位 Watcher，是目前 GitHub 中 Star 数最多的 Python Web 框架。



Flask 的 logo



附注 Flask 的图标虽然看起来很像辣椒，但其实它是角状的容器 (powder horn)。

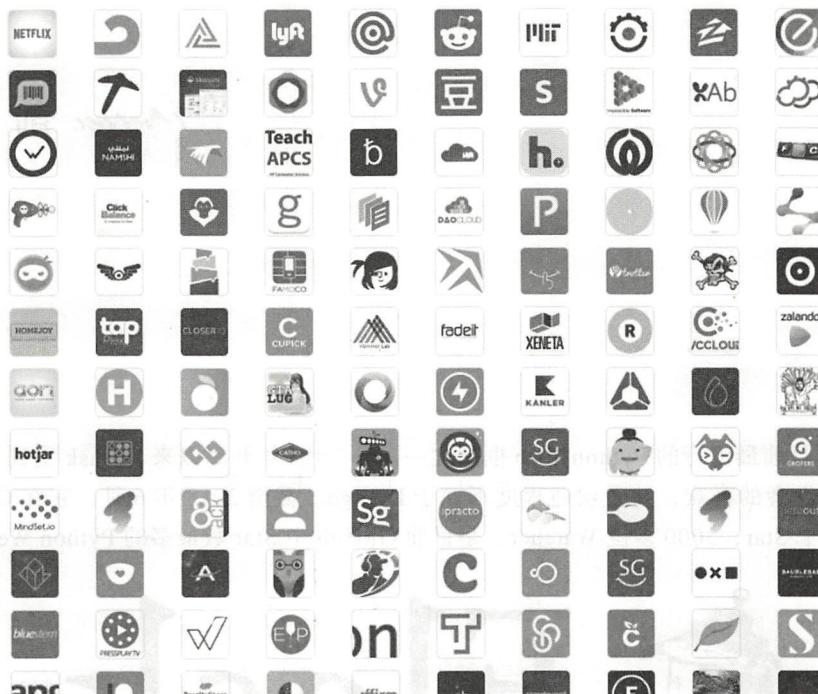
Flask 仅保留了 Web 框架的核心，其他的功能都交给扩展实现。如果没有合适的扩展，你甚至可以自己编写。Flask 不会替你做决定，也不会限制你的选择。它足够轻量，你可以只用 5 行就编写出一个最简单的 Web 程序，但并不简陋，它能够适应各类项目的开发。

因为 Flask 的灵活性，越来越多的公司选择 Flask 作为 Web 框架，甚至开始从 Django 迁移到 Flask。使用 Flask 的公司在国外有 Netflix、Reddit、Twilio、Mailgun 等，在国内则有豆瓣、果壳、下厨房等，这说明 Flask 能经受大型项目的挑战，能够适应各种需求。下图列出了部分使用 Flask 的公司。



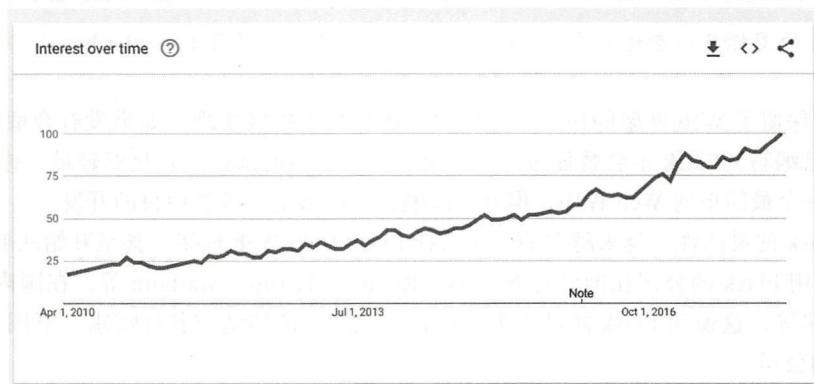
附注 你可以在 StackShare 上查看完整的使用 Flask 的公司列表 (<https://stackshare.io/flask>)。





使用 Flask 的公司

在国内，越来越多的 Python 程序员开始关注和学习 Flask。对于国内的程序员来说，相关书籍仅有一两本，内容上也过于陈旧和单薄，希望本书可以填补这一空白。本书提供了学习 Flask 的完整路径，从基础内容到进阶实践，再到源码分析。同时也安排了丰富的示例程序，让读者可以通过亲自实践来更快地掌握 Flask 开发。

Flask 自 2010 年开源以来在 Google 上的搜索趋势[⊖]

[⊖] 参考来源：Google Trends (<https://trends.google.com/trends/explore?date=2010-04-01%202018-04-01&q=%2Fm%2F0dgs72v>)。





目标读者

在技术层面，本书适合所有 Python 程序员（了解 Python 即可）阅读，包括已经学习过其他 Python Web 框架（比如 Django）的读者和没有接触过 Web 框架的读者。

在难度水平层面，本书适合新手以及中级读者阅读。新手会在这里学到 Flask 的基础内容，并且通过丰富、完善的实例学习 Flask 开发的方方面面；中级读者则可以通过阅读和实践进阶内容来进一步提高 Flask 开发能力。

综上所述，本书主要适合以下几类读者：

- 了解 Python 基本语法，想要自己动手做网站的编程爱好者。
- 熟悉 Python，想要从事 Python Web 开发的后端工程师、运维工程师和爬虫工程师。
- 想要从 Django 等其他 Python Web 框架转向 Flask 的 Python 工程师。

本书主要特点

本书主要有三个显著的特点：

(1) 内容全面

本书内容覆盖了 Flask Web 开发的完整路径：从基础知识的学习，到不同类型和复杂程度的程序的编写，再到代码的测试优化以及 Flask 源码分析；从基础的内容管理，到用户认证和权限管理，再到 Flask 与 JavaScript 的数据交互、Web API 的编写以及 WebSocket 的应用等。

(2) 实践丰富

本书包含大量代码片段，并附带多个完整可运行的示例程序。在本书第一部分的第 2~6 章均分别提供一个示例程序；第二部分则会通过介绍 5 个比较完善的 Flask 项目来讲解各个方面的进阶知识；在第三部分还会通过一个真实的扩展来讲解 Flask 扩展开发。通过将各类知识融入实际的项目开发实践中，可以让你更直观地了解具体的代码实现，并且快速应用到实际开发中。

(3) 内容最新

本书的另一个特点就是内容保证最新。书中的代码和示例程序都基于 Flask 最新发布的稳定版 1.0。书中涉及的其他 Python 包和前端框架（Bootstrap、Materialize 等）全部使用最新版本，并且对未来可能会有的变化会加以说明。这些特点可以保证书中的内容在一定时间内不会过时。对于其他书籍或教程中存在的关于 Flask 的误区，本书也会逐一纠正说明。

除了使用的工具保持最新，本书还引入了 Python 和 Flask 开发中的新变化，比如 Flask 的命令行系统、新的 Python 包管理工具（Pipenv）、新的包上传工具（twine）、新的 PyPI 站点（<https://pypi.org>）、在 PyPI 上使用 Markdown 格式的 README……

本书核心内容

本书由三部分组成，分别为基础篇、实战篇、进阶篇，共 16 章。本书章节经过精心设计，力求让读者可以循序渐进地掌握 Flask 开发的基础知识和技巧。





第一部分：基础篇。介绍 Flask 开发相关的基础知识。

- 第 1 章：搭建开发环境，编写一个最小的 Flask 程序并运行它，了解 Flask 基本知识。
- 第 2 章：介绍 Flask 与 HTTP 的交互方式以及相关的 Flask 功能。
- 第 3 章：介绍 Jinja2 模板的使用。
- 第 4 章：介绍 Web 表单的创建和表单数据的验证。
- 第 5 章：介绍在 Flask 程序中使用数据库进行 CRUD 操作。
- 第 6 章：介绍在 Flask 程序中发送电子邮件的几种方式。

第二部分：实战篇。通过几个示例程序来介绍 Flask 开发中各类功能的实现方法和技巧。

- 第 7 章：通过一个简单的留言板程序 SayHello 介绍 Web 开发的基本流程和基本的项目管理方式，对第一部分的基础知识进行简单回顾。
- 第 8 章：通过个人博客程序 Bluelog 介绍 CRUD 操作、用户认证、文章评论、管理后台等功能。
- 第 9 章：通过图片社交程序 Albulmy 介绍用户注册和认证、用户权限管理、图片上传与处理、用户头像、复杂的数据库关系、复杂的数据库查询、全文搜索等内容。
- 第 10 章：通过待办事项程序 Todoism 介绍单页应用、国际化与本地化、Web API、OAuth 服务器端实现等内容。
- 第 11 章：通过聊天室程序 CatChat 介绍 Websocket 应用、OAuth 客户端实现（第三方登录）、Markdown 支持、代码语法高亮等内容。

第三部分：进阶篇。介绍 Flask 程序的部署流程，如测试、性能优化、部署上线；介绍 Flask 开发的进阶话题，如 Flask 扩展开发、Flask 源码与机制分析。

- 第 12 章：介绍 Flask 程序的自动化测试，包括单元测试和 UI 测试的编写、计算测试覆盖率和代码质量检查。
- 第 13 章：介绍对 Flask 程序进行性能优化的主要措施，包括函数与数据库查询的性能分析、缓存的使用、静态文件优化。
- 第 14 章：介绍部署 Flask 程序前的准备，以及部署到 Linux 服务器和云平台 Heroku、PythonAnywhere 的完整流程。
- 第 15 章：通过扩展 Flask-Share 来介绍编写 Flask 扩展的完整流程，从创建项目到上传到 PyPI。
- 第 16 章：介绍 Flask 的一些设计理念，包括底层 WSGI 的相关实现，并对各个主要功能点进行源码分析。

此外，书的最后还提供了附录 A，补充介绍一些 Flask 学习相关的资源。

阅读前的准备

在开始我们的 Flask 之旅前，还有一些准备工作要做。首先，你要有一台安装了 Python (<https://www.python.org/>) 的电脑，并且，你要了解 Python 的基础知识。





 提示 本书中所有示例程序的代码均通过了 Python 2.7 和 Python 3.6 的测试，建议你选用这两个版本。因为大多数 Python 包（包括 Flask）已经不再支持 Python 2.6 及以下版本，以及 Python 3.3 及以下版本，确保不要使用这些版本。另外，Python 官方社区将于 2020 年 1 月 1 日停止对 Python 2.X 的维护，这或许可以作为你选择 Python 版本时的考量之一。

其次，本书有大量操作需要在命令行（CLI, Command Line Interface）下进行，所以你要熟悉你所在操作系统的命令行。书中会在涉及操作系统特定的命令时给出提示，Windows 系统给出的命令对应的是 CMD.exe，Linux 和 macOS 系统则对应的是 Bash。

最后，HTML、CSS、JavaScript 分别作为一个 Web 页面的结构层、表现层和行为层，是 Web 开发的基础，你需要对它们有基本的了解。任何一个 Web 程序都是由单个或多个 Web 页面，页面上包含的内容，以及按钮、表单等交互组件构成的。在本书中，我们会使用 Flask 操作 HTML 页面；为了让 HTML 页面更加美观，我们会使用 CSS 定义样式，为了简化编写样式的操作，我们会使用 CSS 框架，比如 Bootstrap (<http://getbootstrap.com/>)；为了让某些操作更加合理和方便，或为了给程序增加动画效果，我们会使用 JavaScript 来操作页面元素，为了简化编写 JavaScript 的工作，我们会使用 JavaScript 库 jQuery (<https://jquery.com/>)。

 附注 在 Web 开发中，大部分程序离不开 JavaScript，JavaScript 可以很方便、简洁地实现很多页面逻辑和功能。为了更多地介绍 Flask，本书将尽量避免使用过多的 JavaScript 代码。

如果你还不熟悉这些内容，那么可以通过下面的网站来快速入门：

- W3Schools (<https://www.w3schools.com>)。
- MDN Web 文档 (<https://developer.mozilla.org/docs/Web>)。
- Codecademy (<https://www.codecademy.com>)。

使用示例程序

示例程序均使用 Git 来管理程序版本，为了便于大家获取示例程序，代码均托管在 GitHub (<https://github.com/>) 上。Git (<https://git-scm.com>) 是最流行的开源 VCS (Version Control System，版本控制系统)，大多数项目都使用它来追踪文本文件（代码）的变化。Git 非常易于上手，如果你还不熟悉它，可以访问 Try Git (<https://try.github.io>) 来快速了解 Git。

你可以访问 Git 官网的下载页面 (<https://git-scm.com/downloads>) 了解不同操作系统下 Git 的安装方法，安装成功后即可使用它来获取示例程序。下面介绍了两种使用示例程序的方式。

1. 阅读示例程序

因为示例程序都托管在 GitHub 上，所以阅读示例程序最简单的方式是在浏览器中阅读。在对应的章节，我们会给出示例程序在 GitHub 上的仓库链接。

如果要在本地阅读，那么首先使用 git clone 命令把 GitHub 上的示例程序克隆（即复制）到本地，以本书的项目仓库为例：



VIII

```
$ git clone https://github.com/greyli/helloflask.git
```

 提示 clone 命令后面的参数是远程 Git 仓库的 URL，最后的“.git”后缀可以省略。这里的 URL 中的传输协议使用了 http(s):// 协议，你也可以使用 git:// 协议，即 git://github.com/greyli/helloflask.git。

使用 ls (即 List) 命令 (Windows 下使用 dir 命令) 列出当前目录下的文件信息，你会看到当前目录中多了一个 helloflask 文件夹，这就是我们刚刚复制下来的项目仓库。下面使用 cd (即 change directory) 命令切换进这个文件夹：

```
$ cd helloflask
```

现在你可以使用你喜欢的文本编辑器打开项目文件夹并准备阅读了。建议使用轻量的文本编辑器来阅读示例代码，比如 Atom (<https://atom.io/>)、Sublime Text (<https://www.sublimetext.com/>) 或 Notepad++ (<https://notepad-plus-plus.org/>)。

在对应章节的开始处都会包含从 GitHub 复制程序、创建虚拟环境并运行程序的基本步骤，你可以一边阅读源码，一边实际尝试使用对应的程序功能。

示例程序根据章节内容设置了对应的标签，每个标签都对应了一个程序版本。你可以使用 git tag -n 命令查看当前项目仓库中包含的所有标签：

```
$ git tag -n
```

使用 git checkout 命令即可签出对应标签版本的代码，添加标签名作为参数，比如：

```
$ git checkout foo
```

在后面，书中会在每一次包含更改文件的章节提示应该签出的标签名。如果在执行新的签出命令之前，你对文件做了修改，那么需要使用 git reset 命令来撤销改动：

```
$ git reset --hard
```

 注意 git reset 命令会删除本地修改，如果你希望修改示例程序源码并保存修改，可以参考后面的“改造示例程序部分”。

如果你想比较两个版本之间的变化，可以使用 git diff 命令，添加比较的两个标签作为参数，比如：

```
$ git diff foo bar
```

如果你想更直观地查看版本变化，可以使用下面的命令打开内置的 Git 浏览客户端：

```
$ gitk
```

除了内置的 Git 客户端，还有大量的第三方客户端可以使用，详情可以访问 <https://git-scm.com/downloads/guis> 查看。另外，你也可以访问 GitHub 的 Web 页面查看不同版本（标签）的变化，即查看某项目两个版本之间的变化可以访问 <https://github.com/用户名/仓库名/compare/标签A...标签B>，比如对 foo 和 bar 标签进行比较可以访问 <https://github.com/greyli/helloflask/compare/foo...bar>。

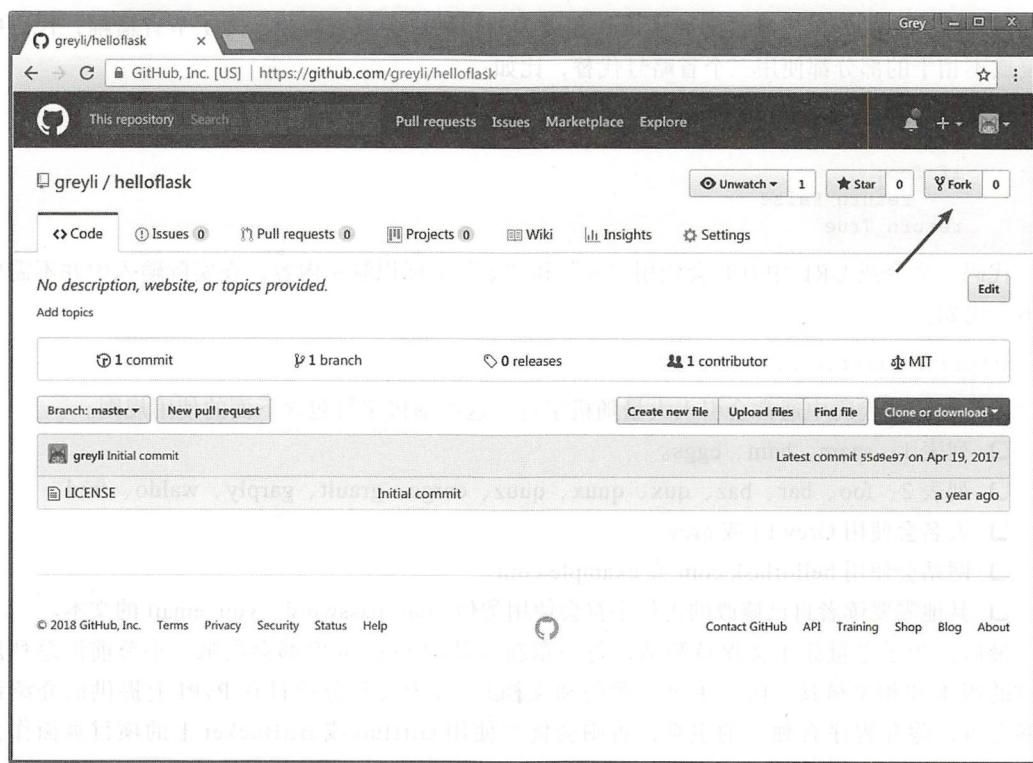
最后，你可以定期使用 git fetch 命令来更新本地仓库：

```
$ git fetch --all
$ git fetch --tags
$ git reset --hard origin/master
```

2. 改造示例程序

只看菜谱是不能学会烹饪的，自己动手编写代码才是学习 Flask 最有效的途径。你可以在阅读示例程序的同时编写自己的 Flask 程序，将书中介绍的内容和实际的示例程序代码作为参照。另外，你也可以创建一份示例程序的拷贝（派生，fork），这样你就可以自由地修改示例程序的源码，改造成你自己的示例程序。创建派生仓库的主要步骤如下：

- 1) 注册一个 GitHub 账号 (<https://github.com>)。
- 2) 访问示例程序的 GitHub 仓库页面（比如 <https://github.com/greyli/helloflask>），单击右上角的 Fork 按钮创建一个派生仓库，如下图所示。



创建派生仓库

- 3) 在本地使用 git clone 命令复制新创建的派生仓库，使用你的用户名构建 URL：

```
$ git clone https://github.com/你的用户名/helloflask.git
```

现在你可以在本地自由修改实例程序，并提交到你的 GitHub 账户的远程仓库中了。

排版约定

Windows 中的命令提示符为“>”，而 Linux 或 macOS 中的命令提示符为“\$”，本书中将统一使用美元符号（即“\$”）作为命令提示符，比如：

```
$ cd hello
```

命令提示符为三个大于号（即“>>>”）表示 Python Shell 中输入的代码，比如：

```
>>> import os
```

“\$”或“>>>”标记的文本下方没有命令提示符的文字表示输出的字符，不需要打出，比如：

```
$ cat hello.txt
Hello, Flask!
```

为了节省篇幅，本书中的代码片段没有严格遵照 PEP8 的约定，比如类和函数之间的空行被缩减为 1 行。另外，出现过的导入语句和无关的代码块会被省略掉。为了节省篇幅，代码中重复或不相干的部分都使用三个省略号代替，比如：

```
def do_something():
    ...
    if foo:
        return False
    return True
```

代码、命令或 URL 中有时会使用“<”和“>”来标识演示内容，在实际输入中并不需要写出，比如：

```
https://github.com/<你的用户名>
```

因为在示例代码中通常会引入大量随机字符，这些随机字符包含下面的使用规则。

- 列表 1：spam、ham、eggs。
- 列表 2：foo、bar、baz、qux、quux、quuz、corge、grault、garply、waldo、fred。
- 人名会使用 Grey Li 或 grey。
- 网站会使用 helloflask.com 或 example.com。
- 其他需要读者自己修改的占位字符会使用类似 your_password、you_email 的文本。

最后，为了尽量让正文保持简洁，每一章新涉及的 Python 库都会在第一小节前汇总列出对应的版本和相关链接（比如主页、源码和文档）。因为大部分项目在 PyPI 上提供的介绍都不够完善，除非程序有独立的主页，否则会优先使用 GitHub 或 BitBucket 上的项目页面作为主页。

读者反馈与疑问

由于笔者水平有限，编写时间也比较仓促，书中难免有错误或者不全面的地方，在此恳请读者朋友批评指正。

关于本书的疑问和反馈可以到本书在 GitHub 上的项目仓库 HelloFlask (<https://github.com/greylly/helloflask>) 中创建 Issue 并提交；书中的错误或笔误可以修改仓库中的勘误文件 (Errata.md) 并提交 Pull Request。

对于示例程序的疑问、反馈和改进建议可以到示例程序在 GitHub 上的项目仓库提交 Issue 或 Pull Request，具体的网址可以在对应的章节看到。

当然，你也可以直接发邮件与笔者联系，笔者的邮箱是 withlihui@gmail.com。

本书的配套资源索引可以在本书的主页 <http://helloflask.com/book> 上看到。另外，你可以访问笔者的博客 (<http://greylly.com>) 或是知乎专栏“Hello, Flask!” (<https://zhuanlan.zhihu.com/flask>) 阅读更多与 Flask 相关的文章。

致谢

首先，感谢机械工业出版社华章公司的杨福川老师和李艺老师。因为杨老师的信任，才让笔者有幸写作这本书。本书能够顺利完成，离不开两位老师的悉心指导，更离不开其他编辑的辛苦工作。

其次，感谢 Flask 社区和其他开源项目的贡献者们创造了这一切，也感谢在 Stack Overflow、GitHub、Reddit 和 Wikipedia 等网站贡献知识的开发者们。

最后，感谢父母和奶奶这段时间的支持和帮助，也感谢女友魏瑶和弟弟家辉给予的鼓励和陪伴。

目 录 *Contents*

前言

第一部分 基础篇

第1章 初识 Flask	2
1.1 搭建开发环境	3
1.1.1 Pipenv 工作流	3
1.1.2 安装 Flask	7
1.1.3 集成开发环境	8
1.2 Hello, Flask!	11
1.2.1 创建程序实例	11
1.2.2 注册路由	12
1.3 启动开发服务器	14
1.3.1 Run, Flask, Run!	14
1.3.2 更多的启动选项	18
1.3.3 设置运行环境	18
1.4 Python Shell	20
1.5 Flask 扩展	21
1.6 项目配置	22
1.7 URL 与端点	23
1.8 Flask 命令	23
1.9 模板与静态文件	24
1.10 Flask 与 MVC 架构	25

1.11 本章小结	26
-----------------	----

第2章 Flask 与 HTTP

2.1 请求响应循环	27
2.2 HTTP 请求	29
2.2.1 请求报文	29
2.2.2 Request 对象	31
2.2.3 在 Flask 中处理请求	34
2.2.4 请求钩子	37
2.3 HTTP 响应	38
2.3.1 响应报文	39
2.3.2 在 Flask 中生成响应	40
2.3.3 响应格式	43
2.3.4 来一块 Cookie	46
2.3.4 session: 安全的 Cookie	49
2.4 Flask 上下文	54
2.4.1 上下文全局变量	54
2.4.2 激活上下文	55
2.4.3 上下文钩子	56
2.5 HTTP 进阶实践	57
2.5.1 重定向回上一个页面	57
2.5.2 使用 AJAX 技术发送异步请求	60
2.5.3 HTTP 服务器端推送	64

2.5.4 Web 安全防范	65
2.6 本章小结	74
第 3 章 模板	75
3.1 模板基本用法	75
3.1.1 创建模板	76
3.1.2 模板语法	77
3.1.3 渲染模板	78
3.2 模板辅助工具	80
3.2.1 上下文	80
3.2.2 全局对象	82
3.2.3 过滤器	83
3.2.4 测试器	85
3.2.5 模板环境对象	87
3.3 模板结构组织	88
3.3.1 局部模板	88
3.3.2 宏	88
3.3.3 模板继承	90
3.4 模板进阶实践	93
3.4.1 空白控制	93
3.4.2 加载静态文件	94
3.4.3 消息闪现	98
3.4.4 自定义错误页面	100
3.4.5 JavaScript 和 CSS 中的 Jinja2	101
3.5 本章小结	103
第 4 章 表单	104
4.1 HTML 表单	104
4.2 使用 Flask-WTF 处理表单	106
4.2.1 定义 WTForms 表单类	106
4.2.2 输出 HTML 代码	109
4.2.3 在模板中渲染表单	110
4.3 处理表单数据	112
4.3.1 提交表单	112
4.3.2 验证表单数据	113
4.3.3 在模板中渲染错误消息	117
4.4 表单进阶实践	118
4.4.1 设置错误消息语言	118
4.4.2 使用宏渲染表单	120
4.4.3 自定义验证器	121
4.4.4 文件上传	122
4.4.5 使用 Flask-CKEditor 集成富文本编辑器	129
4.4.6 单个表单多个提交按钮	132
4.4.7 单个页面多个表单	133
4.5 本章小结	137
第 5 章 数据库	138
5.1 数据库的分类	139
5.1.1 SQL	139
5.1.2 NoSQL	139
5.1.3 如何选择?	140
5.2 ORM 魔法	140
5.3 使用 Flask-SQLAlchemy 管理数据库	142
5.3.1 连接数据库服务器	142
5.3.2 定义数据库模型	144
5.3.3 创建数据库和表	145
5.4 数据库操作	146
5.4.1 CRUD	147
5.4.2 在视图函数里操作数据库	151
5.5 定义关系	156
5.5.1 配置 Python Shell 上下文	157
5.5.2 一对多	158

5.5.3 多对一	164	7.1.1 配置文件	196
5.5.4 一对一	165	7.1.2 创建程序实例	197
5.5.5 多对多	166	7.2 Web 程序开发流程	198
5.6 更新数据库表	168	7.2.1 程序功能设计	199
5.6.1 重新生成表	168	7.2.1 前端页面开发	200
5.6.2 使用 Flask-Migrate 迁移 数据库	169	7.2.3 后端程序开发	202
5.6.3 开发时是否需要迁移?	171	7.3 使用 Bootstrap-Flask 简化页面 编写	206
5.7 数据库进阶实践	172	7.3.1 加载资源文件	207
5.7.1 级联操作	172	7.3.2 快捷渲染表单	207
5.7.2 事件监听	175	7.4 使用 Flask-Moment 本地化日期 和时间	209
5.8 本章小结	177	7.4.1 本地化前的准备	209
第 6 章 电子邮件	178	7.4.2 使用 Flask-Moment 集成 Moment.js	209
6.1 使用 Flask-Mail 发送电子邮件	179	7.4.3 渲染时间日期	210
6.1.1 配置 Flask-Mail	179	7.5 使用 Faker 生成虚拟数据	213
6.1.2 构建邮件数据	182	7.6 使用 Flask-DebugToolbar 调试 程序	215
6.1.3 发送邮件	182	7.7 Flask 配置的两种组织形式	216
6.2 使用事务邮件服务 SendGrid	183	7.7.1 环境变量优先	217
6.2.1 注册 SendGrid	183	7.7.2 实例文件夹覆盖	217
6.2.2 SendGrid SMTP 转发	185	7.8 本章小结	218
6.2.3 SendGrid Web API 转发	185	第 8 章 个人博客	219
6.3 电子邮件进阶实践	188	8.1 大型项目结构	220
6.3.1 提供 HTML 正文	188	8.1.1 使用蓝本模块化程序	221
6.3.2 使用 Jinja2 模板组织邮件正文	189	8.1.2 使用类组织配置	227
6.3.3 异步发送邮件	191	8.1.3 使用工厂函数创建程序实例	228
6.4 本章小结	192	8.2 编写程序骨架	232
第二部分 实战篇		8.2.1 数据库	233
第 7 章 留言板	194	8.2.2 模板	240
7.1 使用包组织代码	195		

8.2.3 表单	246	9.1.3 混合式架构	303
8.2.4 视图函数	249	9.1.4 如何选择	303
8.2.5 电子邮件支持	249	9.2 编写程序骨架	303
8.3 编写博客前台	251	9.2.1 数据库模型与虚拟数据	305
8.3.1 分页显示文章列表	251	9.2.2 模板与静态文件	307
8.3.2 显示文章正文	258	9.3 高级用户认证	308
8.3.3 文章固定链接	259	9.3.1 用户注册	309
8.3.4 显示分类文章列表	262	9.3.2 验证邮箱地址	311
8.3.5 显示评论列表	263	9.3.3 使用装饰器过滤未确认用户	315
8.3.6 发表评论与回复	266	9.3.4 密码重置	316
8.3.7 支持回复评论	267	9.4 基于用户角色的权限管理	319
8.3.8 网站主题切换	269	9.4.1 角色与权限模型	319
8.4 初始化博客	271	9.4.2 设置角色与权限	320
8.4.1 安全存储密码	271	9.4.3 写入角色与权限	321
8.4.2 创建管理员用户	273	9.4.4 验证用户权限	323
8.5 使用 Flask-Login 管理用户认证	275	9.5 使用 Flask-Dropzone 优化文件上传	325
8.5.1 获取当前用户	276	9.5.1 配置 Flask-Dropzone	326
8.5.2 登入用户	277	9.5.2 渲染上传区域	328
8.5.3 登出用户	278	9.5.3 处理并保存上传图片	329
8.5.4 视图保护	279	9.6 使用 Flask-Avatars 处理用户头像	334
8.6 使用 CSRFProtect 实现 CSRF 保护	281	9.6.1 默认头像	335
8.7 编写博客后台	283	9.6.2 生成随机头像	335
8.7.1 文章管理	284	9.7 图片展示与管理	337
8.7.2 评论管理	291	9.7.1 在用户主页显示图片列表	338
8.7.3 分类管理	297	9.7.2 图片详情页	341
8.8 本章小结	298	9.7.3 上一张下一张跳转	342
第 9 章 图片社交网站	299	9.7.4 删除确认模态框	344
9.1 项目组织架构	300	9.7.5 举报图片	346
9.1.1 功能式架构	300	9.7.6 图片描述	347
9.1.2 分区式架构	302	9.7.7 图片标签	349
9.8 收藏图片	358	9.7.8 用户资料弹窗	353

9.8.1 使用关联模型表示多对多关系	358	9.14 编写网站后台	407
9.8.2 添加和取消收藏	360	9.14.1 用户管理	408
9.8.3 收藏者和收藏页面	362	9.14.2 资源管理	411
9.9 用户关注	365	9.14.3 面向管理员的用户资料编辑	412
9.9.1 自引用多对多关系	365	9.15 本章小结	413
9.9.2 关注与取消关注	366		
9.9.3 显示关注用户列表	369		
9.9.4 使用 AJAX 在弹窗中执行关注操作	371		
9.10 消息提醒	378		
9.10.1 提醒消息在数据库中的表示	379		
9.10.2 创建提醒	379		
9.10.3 显示和管理提醒	380		
9.10.4 通过轮询实时更新未读计数	382		
9.11 用户资料与账户设置	384		
9.11.1 编辑个人资料	385		
9.11.2 自定义头像	386		
9.11.3 更改密码	392		
9.11.4 提醒消息开关	394		
9.11.5 将收藏设为仅自己可见	395		
9.11.6 注销账户	396		
9.12 首页与探索	397		
9.12.1 获取正在关注用户的图片	399		
9.12.2 使用联结和分组查询获取热门标签	401		
9.12.3 使用数据库通用函数获取随机图片	402		
9.13 使用 Flask-Whooshee 实现全文搜索	403		
9.13.1 创建索引	404		
9.13.2 搜索表单	405		
9.13.3 显示搜索结果	406		
		第 10 章 待办事项程序	415
		10.1 使用 JavaScript 和 AJAX 编写单页程序	417
		10.1.1 单页程序的模板组织	418
		10.1.2 在根页面内切换子页面	421
		10.1.3 生成测试账户	423
		10.1.4 添加新待办条目	424
		10.2 国际化与本地化	426
		10.2.1 使用 Flask-Babel 集成 Babel	427
		10.2.2 区域和语言	427
		10.2.3 文本的国际化	432
		10.2.4 文本的本地化	433
		10.2.5 时间与日期的本地化	438
		10.3 设计并编写 Web API	440
		10.3.1 认识 Web API	441
		10.3.2 设计优美实用的 Web API	443
		10.3.3 使用 Flask 编写 Web API	446
		10.3.4 使用 OAuth 认证	453
		10.3.5 资源的序列化	461
		10.3.6 资源的反序列化	465
		10.3.7 Web API 的测试与发布	470
		10.4 本章小结	473
		第 11 章 在线聊天室	474
		11.1 编写程序骨架	476
		11.2 Gravatar 头像	477

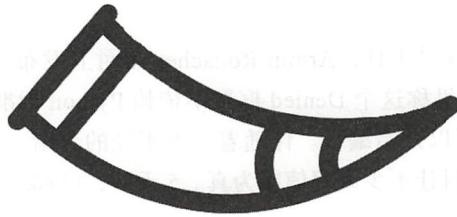
11.3 使用 Flask-SocketIO 建立实时 双向通信.....	480	12.4.1 安装浏览器与驱动.....	537
11.3.1 建立 Socket.IO 连接.....	481	12.4.2 准备测试环境.....	538
11.3.2 一条消息的旅程.....	482	12.4.3 编写测试代码.....	539
11.3.3 在线人数统计.....	485	12.5 使用 Coverage.py 计算测试 覆盖率.....	541
11.3.4 通信频道分离.....	486	12.5.1 基本用法.....	542
11.4 使用 Flask-OAuthlib 实现第三方 登录.....	489	12.5.2 获取测试覆盖率.....	543
11.4.1 编写 OAuth 客户端.....	490	12.6 使用 Flake8 检查代码质量	544
11.4.2 注册 OAuth 程序.....	491	12.7 本章小结.....	546
11.4.3 处理 OAuth2 授权.....	493		
11.4.4 处理 OAuth1 授权.....	505		
11.5 聊天室功能增强	507		
11.5.1 无限滚动加载历史消息.....	507		
11.5.2 Markdown 支持	510		
11.5.3 代码语法高亮.....	512		
11.5.4 标签页消息提醒.....	515		
11.5.5 浏览器桌面通知.....	517		
11.5.6 消息管理.....	519		
11.6 本章小结.....	519		
第三部分 进阶篇			
第 12 章 自动化测试	522		
12.1 认识自动化测试	522		
12.2 Flask 测试客户端	523		
12.3 使用 unittest 编写单元测试	525		
12.3.1 Flask 程序的测试固件.....	525		
12.3.2 编写测试用例.....	527		
12.3.3 组织测试.....	533		
12.3.4 运行测试.....	535		
12.4 使用 Selenium 进行用户界面测试.....	536		
第 13 章 性能优化	547		
13.1 程序性能分析.....	548		
13.1.1 函数性能分析.....	548		
13.1.2 数据库查询分析.....	549		
13.2 使用 Flask-Caching 设置缓存	551		
13.2.1 缓存视图函数.....	552		
13.2.2 缓存其他函数.....	554		
13.2.3 更新缓存.....	554		
13.2.4 使用 Redis 作为缓存后端	556		
13.3 使用 Flask-Assets 优化静态资源	557		
13.3.1 注册资源集.....	558		
13.3.2 生成资源集文件.....	559		
13.3.3 在模板中加载资源集.....	559		
13.4 本章小结.....	562		
第 14 章 部署上线	563		
14.1 传统部署 VS 云部署	563		
14.2 基本部署流程	564		
14.3 部署前的准备	566		
14.3.1 更新程序配置.....	566		
14.3.2 创建生产环境专用的程序 实例.....	567		

14.3.3 设置迁移工具	568	15.5 实现扩展功能	610
14.3.4 程序日志	568	15.5.1 加载静态资源	611
14.3.5 手动导入环境变量	572	15.5.2 创建分享组件	612
14.3.6 HTTPS 转发	572	15.5.3 在移动设备上隐藏	613
14.4 部署到 Linux 服务器	573	15.6 开源发布前的准备	614
14.4.1 使用 OpenSSH 登录远程主机	573	15.6.1 添加文档字符串与注释	614
14.4.2 安装基本库和工具	574	15.6.2 编写 README 与文档	615
14.4.3 安全防护措施	575	15.6.3 为打包做准备	616
14.4.4 推送代码并初始化程序环境	578	15.6.4 编写示例程序	621
14.4.5 使用 Gunicorn 运行程序	579	15.6.5 编写单元测试	621
14.4.6 使用 Nginx 提供反向代理	580	15.7 发布到 PyPI	622
14.4.7 使用 Supervisor 管理进程	584	15.7.1 创建 PyPI 账号	623
14.4.8 更新部署后的程序	586	15.7.2 使用 setuptools 打包	623
14.5 部署到 PythonAnywhere	587	15.7.3 使用 twine 上传	625
14.5.1 反向代理设置	587	15.8 编写良好的扩展	625
14.5.2 创建 PythonAnywhere 程序	588	15.9 本章小结	627
14.5.3 推送代码并初始化程序环境	588		
14.5.4 创建数据库	590		
14.5.5 设置虚拟环境	591		
14.5.6 静态文件	592		
14.5.7 运行和更新程序	592		
14.6 部署到 Heroku	594		
14.6.1 通过 Heroku Git 部署	595		
14.6.2 使用 GitHub 部署	601		
14.7 下一步做什么？	603		
14.8 本章小结	604		
第 15 章 Flask 扩展开发	605		
15.1 扩展的命名	606	16.1 阅读 Flask 源码	628
15.2 扩展项目骨架	606	16.1.1 获取 Flask 源码	628
15.3 编写扩展类	608	16.1.2 如何阅读源码	629
15.4 添加扩展配置	610	16.1.3 Flask 发行版本分析	637
		16.2 Flask 的设计理念	638
		16.2.1 “微” 框架	638
		16.2.2 两个核心依赖	638
		16.2.3 显式程序对象	638
		16.2.4 本地上下文	639
		16.2.5 三种程序状态	640
		16.2.6 丰富的自定义支持	641
		16.3 Flask 与 WSGI	641
		16.3.1 WSGI 程序	642
		16.3.2 WSGI 服务器	643

16.3.3 中间件.....	644
16.4 Flask 的工作流程与机制	646
16.4.1 Flask 中的请求响应循环.....	646
16.4.2 路由系统.....	649
16.4.3 本地上下文.....	653
16.4.4 请求与响应对象.....	665
16.4.5 session	667
16.4.6 蓝本.....	674
16.4.7 模板渲染.....	677
16.5 本章小结.....	680
附录 A Flask 资源	681

第一部分 *Part 1*

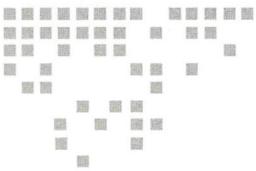
基础篇



注：Icons made by Nikita Golubev www.flaticon.com is licensed by CC 3.0 BY

在本书的第一部分，我们会学习 Flask 开发的基础知识，包括视图函数的编写、HTTP 交互、模板渲染、表单、数据库和电子邮件。

每一章都会提供一个示例程序，其中包含每章涉及的大部分代码。在第一章的开始，我们会介绍如何在本地复制包含所有示例的项目仓库。在后面的每一个章节，我们都会提示如何找到并运行对应的示例程序。此外，第 2 章～第 6 章均有一节是进阶实践部分，这里会介绍对应主题的一些进阶技巧和实践。



Chapter 1

第1章

初识 Flask

全
部
出
售

这一切开始于 2010 年 4 月 1 日，Armin Ronacher 在网上发布了一篇关于“下一代 Python 微框架”的介绍文章，文章里称这个 Denied 框架不依赖 Python 标准库，只需要复制一份 `deny.py` 放到你的项目文件夹就可以开始编程。伴随着一本正经的介绍、名人推荐语、示例代码和演示视频，这个“虚假”的项目让不少人都信以为真。5 天后，Flask (<http://flask.pocoo.org/>) 就从这么一个愚人节玩笑诞生了。

Flask 是使用 Python 编写的 Web 微框架。Web 框架可以让我们不用关心底层的请求响应处理，更方便高效地编写 Web 程序。因为 Flask 核心简单且易于扩展，所以被称作微框架（micro framework）。Flask 有两个主要依赖，一个是 WSGI（Web Server Gateway Interface，Web 服务器网关接口）工具集——Werkzeug（<http://werkzeug.pocoo.org/>），另一个是 Jinja2 模板引擎（<http://jinja.pocoo.org/>）。Flask 只保留了 Web 开发的核心功能，其他的功能都由外部扩展来实现，比如数据库集成、表单认证、文件上传等。如果没有合适的扩展，你甚至可以自己动手开发。Flask 不会替你做决定，也不会限制你的选择。总之，Flask 可以变成任何你想要的东西，一切都由你做主。

 **附注** Flask（瓶子，烧瓶）的命名据说是对另一个 Python Web 框架——Bottle 的双关语 / 调侃，即另一种容器（另一个 Python Web 框架）。Werkzeug 是德语单词“工具（tool）”，而 Jinja 指日本神社，因为神社（庙）的英文 temple 与 template（模板）相近而得名。

 **附注** WSGI（Web Server Gateway Interface）是 Python 中用来规定 Web 服务器如何与 Python Web 程序进行沟通的标准，在本书的第三部分将进行详细介绍。

本章将会对 Flask 的主要基础概念进行一些介绍，并通过一个最简单的 Flask 程序来了解一些核心概念。如果你对某些概念感到疑惑，不用担心，我们会在后面深入学习这些内容。

在本书的第一部分，每一章都有一个对应的示例程序，章节中的大部分示例代码均可以在

示例程序中找到。首先，请打开命令行窗口，切换到合适的目录，然后使用下面的命令把本书的示例程序仓库复制到本地，并切换进项目根目录：

```
$ git clone https://github.com/greyli/helloflask.git  
$ cd helloflask
```

 提示 如果你在 HelloFlask 的 GitHub 页面 (<https://github.com/greyli/helloflask>) 单击了 Fork 按钮，那么可以使用你自己的 GitHub 用户名来替换掉上面的 greyli，这将复制一份派生仓库，你可以自由地修改和提交代码。

本章新涉及的 Python 包如下所示：

❑ Flask (1.0.2)

主页：<http://flask.pocoo.org/>

源码：<http://github.com/pallets/flask>

文档：<http://flask.pocoo.org/docs/>

❑ pip (10.0.1)

主页：<https://github.com/pypa/pip>

文档：<https://pip.pypa.io>

❑ Pipenv (v2018.05.18)

主页：<https://github.com/pypa/pipenv>

文档：<http://pipenv.readthedocs.io/>

❑ Virtualenv (15.1.0)

主页：<https://github.com/pypa/virtualenv>

文档：<https://virtualenv.pypa.io>

❑ Pipfile (0.0.2)

主页：<https://github.com/pypa/pipfile>

❑ python-dotenv (0.8.2)

主页：<https://github.com/theskumar/python-dotenv>

❑ Watchdog (0.8.3)

主页：<https://github.com/gorakhargosh/watchdog>

文档：<https://pythonhosted.org/watchdog/>

1.1 搭建开发环境

在前言中，我们已经简单介绍了阅读本书所需要的基础知识，现在我们开始正式的搭建开发环境。

1.1.1 Pipenv 工作流

Pipenv 是基于 pip 的 Python 包管理工具，它和 pip 的用法非常相似，可以看作 pip 的加强

版，它的出现解决了旧的 pip+virtualenv+requirements.txt 的工作方式的弊端。具体来说，它是 pip、Pipfile 和 Virtualenv 的结合体，它让包安装、包依赖管理和虚拟环境管理更加方便，使用它可以实现高效的 Python 项目开发工作流。如果你还不熟悉这些工具，不用担心，我们会在下面逐一进行介绍。

1. 安装 pip 和 Pipenv

pip 是用来安装 Python 包的工具。如果你使用 Python 2.7.9 及以上版本或 Python 3.4 及以上版本，那么 pip 已经安装好了。可以使用下面的命令检查 pip 是否已经安装：

```
$ pip --version
```

如果报错，那么你需要自己安装 pip。最简单的方式是下载并使用 Python 执行 get-pip.py 文件 (<https://bootstrap.pypa.io/get-pip.py>)。

下面这条命令你经常在各种文档中见到：

```
$ pip install <某个包的名称>
```

这会从 PyPI (Python Package Index, Python 包索引) 上下载并安装指定的包。



附注 PyPI (<https://pypi.org>) 是一个 Python 包的在线仓库，截至 2018 年 5 月，共有 13 万多个包存储在这里。后面我们会学习如何编写自己的 Flask 扩展，并把它上传到 PyPI 上。到时你就可以使用上面这条命令安装自己编写的包。

现在使用 pip 安装 Pipenv：

```
$ pip install pipenv
```

在 Linux 或 macOS 系统中使用 sudo 以全局安装：

```
$ sudo pip install pipenv
```



附注 如果你不全局安装，可以添加 --user 选项执行用户安装（即 pip install --user pipenv），并手动将用户基础二进制目录添加到 PATH 环境变量中，具体可参考 <https://docs.pipenv.org/install/#installing-pipenv>。



提示 PyPI 中的包名称不区分大小写。出于方便的考虑，后面的安装命令都将使用小写名称。

可以使用下面的命令检查 Pipenv 是否已经安装：

```
$ pipenv --version  
pipenv, version 11.10.4
```

2. 创建虚拟环境

在 Python 中，虚拟环境 (virtual environment) 就是隔离的 Python 解释器环境。通过创建虚拟环境，你可以拥有一个独立的 Python 解释器环境。这样做的好处是可以为每一个项目创建独

立的 Python 解释器环境，因为不同的项目常常会依赖不同版本的库或 Python 版本。使用虚拟环境可以保持全局 Python 解释器环境的干净，避免包和版本的混乱，并且可以方便地区分和记录每个项目的依赖，以便在新环境下复现依赖环境。

虚拟环境通常使用 Virtualenv 来创建，但是为了更方便地管理虚拟环境和依赖包，我们将会使用集成了 Virtualenv 的 Pipenv。首先确保我们当前工作目录在示例程序项目的根目录，即 helloflask 文件夹中，然后使用 pipenv install 命令为当前的项目创建虚拟环境：

```
$ pipenv install
Creating a virtualenv for this project...
...
Virtualenv location: /path/to/virtualenv/helloflask-5Pa0ZfZw
...
```

这会为当前项目创建一个文件夹，其中包含隔离的 Python 解释器环境，并且安装 pip、wheel、setuptools 等基本的包。因为示例程序仓库里包含 Pipfile 文件，所以这个文件中列出的依赖包也会一并被安装，下面会具体介绍。

 **附注** 默认情况下，Pipenv 会统一管理所有虚拟环境。在 Windows 系统中，虚拟环境文件夹会在 C:\Users\Administrator\.virtualenvs\ 目录下创建，而 Linux 或 macOS 会在 ~/.local/share/virtualenvs/ 目录下创建。如果你想在项目目录内创建虚拟环境文件夹，可以设置环境变量 PIPENV_VENV_IN_PROJECT，这时名为 .venv 的虚拟环境文件夹将在项目根目录被创建。

虚拟环境文件夹的目录名称的形式为“当前项目目录名 + 一串随机字符”，比如 helloflask-5Pa0ZfZw。

 **提示** 你可以通过 --three 和 --two 选项来声明虚拟环境中使用的 Python 版本（分别对应 Python3 和 Python2），或是使用 --python 选项指定具体的版本号。同时要确保对应版本的 Python 已经安装在电脑中。

在单独使用 Virtualenv 时，我们通常会显式地激活虚拟环境。在 Pipenv 中，可以使用 pipenv shell 命令显式地激活虚拟环境：

```
$ pipenv shell
Loading .env environment variables...
Launching subshell in virtual environment. Type 'exit' to return.
```

 **提示** 当执行 pipenv shell 或 pipenv run 命令时，Pipenv 会自动从项目目录下的 .env 文件中加载环境变量。

Pipenv 会启动一个激活虚拟环境的子 shell，现在你会发现命令行提示符前添加了虚拟环境名“(虚拟环境名称) \$”，比如：

```
(helloflask-5Pa0ZfZw) $
```

这说明我们已经成功激活了虚拟环境，现在你的所有命令都会在虚拟环境中执行。当你需要退出虚拟环境时，使用 exit 命令。

 **注意** 在 Windows 系统中使用 pipenv shell 激活虚拟环境时，虽然激活成功，但是命令行提示符前不会显示虚拟环境名称。

除了显式地激活虚拟环境，Pipenv 还提供了一个 pipenv run 命令，这个命令允许你不显式激活虚拟环境即可在当前项目的虚拟环境中执行命令，比如：

```
$ pipenv run python hello.py
```

这会使用虚拟环境中的 Python 解释器，而不是全局的 Python 解释器。事实上，和显式激活 / 关闭虚拟环境的传统方式相比，pipenv run 是更推荐的做法，因为这个命令可以让你在执行操作时不用关心自己是否激活了虚拟环境。当然，你可以自由选择你偏爱的用法。

 **注意** 为了方便书写，本书后面涉及的诸多命令会直接写出，省略前面的虚拟环境名称。在实际执行时，你需要使用 pipenv shell 激活虚拟环境后执行命令，或是在命令前加入 pipenv run，后面不再提示。

3. 管理依赖

一个程序通常会使用很多的 Python 包，即依赖（dependency）。而程序不仅仅会在一台电脑上运行，程序部署上线时需要安装到远程服务器上，而你也许会把它分享给朋友。如果你打算开源的话，就可能会有更多的人需要在他们的电脑上运行。为了能顺利运行程序，他们不得不记下所有依赖包，然后使用 pip 或 Pipenv 安装，这些重复无用的工作当然应该避免。在以前我们通常使用 pip 搭配一个 requirements.txt 文件来记录依赖。但 requirements.txt 需要手动维护，在使用上不够灵活。Pipfile 的出现就是为了替代难于管理的 requirements.txt。

在创建虚拟环境时，如果项目根目录下没有 Pipfile 文件，pipenv install 命令还会在项目文件夹根目录下创建 Pipfile 和 Pipfile.lock 文件，前者用来记录项目依赖包列表，而后者记录了固定版本的详细依赖包列表。当我们使用 Pipenv 安装 / 删除 / 更新依赖包时，Pipfile 以及 Pipfile.lock 会自动更新。

 **附注** 你可以使用 pipenv graph 命令查看当前环境下的依赖情况，或是在虚拟环境中使用 pip list 命令查看依赖列表。

当需要在一个新的环境运行程序时，只需要执行 pipenv install 命令。Pipenv 就会创建一个新的虚拟环境，然后自动从 Pipfile 中读取依赖并安装到新创建的虚拟环境中。

 **提示** 在本书撰写时，Pipfile 项目还处于活跃的开发阶段，有很多东西还没有固定，所以这里不会过多介绍，具体请访问 Pipfile 主页了解。

1.1.2 安装 Flask

下面使用 pipenv install 命令在我们刚刚创建的虚拟环境里安装 Flask：

```
$ pipenv install flask
Installing flask...
...
Successfully installed Jinja2-2.10 MarkupSafe-1.0 Werkzeug-0.14.1 click-6.7
flask-1.0.2 itsdangerous-0.24
```

 提示 Pipenv 会自动帮我们管理虚拟环境，所以在执行 pipenv install 安装 Python 包时，无论是否激活虚拟环境，包都会安装到虚拟环境中。后面我们都将使用 Pipenv 安装包，这相当于在激活虚拟环境的情况下使用 pip 安装包。只有需要在全局环境下安装 / 更新 / 删除包，我们才会使用 pip。

从上面成功安装的输出内容可以看出，除了 Flask 包外，同时被安装的还有 5 个依赖包，它们的主要介绍如表 1-1 所示。

表 1-1 Flask 的依赖包

名称与版本	说 明	资 源
Jinja2 (2.10)	模板渲染引擎	主页: http://jinja.pocoo.org/ 源码: https://github.com/pallets/jinja 文档: http://jinja.pocoo.org/docs/
MarkupSafe (1.0)	HTML 字符转义 (escape) 工具	主页: https://github.com/pallets/markupsafe
Werkzeug (0.14.1)	WSGI 工具集，处理请求与响应，内置 WSGI 开发服务器、调试器和重载器	主页: http://werkzeug.pocoo.org/ 源码: https://github.com/pallets/werkzeug 文档: http://werkzeug.pocoo.org/docs/
click (6.7)	命令行工具	主页: https://github.com/pallets(click 文档: http://click.pocoo.org/6/
itsdangerous (0.24)	提供各种加密签名功能	主页: https://github.com/pallets/itsdangerous 文档: https://pythonhosted.org/itsdangerous/

在大部分情况下，为了方便表述，我会直接称 Flask 使用这些包提供的功能为 Flask 提供的功能，必要时则会具体说明。这里仅仅是打个照面，后面我们会慢慢熟悉这些包。

 附注 包括 Flask 在内，Flask 的 5 个依赖包都由 Pocoo 团队 (<http://www.pocoo.org/>) 开发，主要作者均为 Armin Ronacher (<http://lucumr.pocoo.org/>)，这些项目均隶属于 Pallets 项目 (<https://www.palletsprojects.com/>)。

本书使用了最新版本的 Flask (1.0.2)，如果你还在使用旧版本，请使用下面的命令进行更新：

```
$ pipenv update flask
```

另外，本书涉及的所有 Python 包都将使用当前发布的最新版本，在每一章的开始我们都会

列出新涉及的 Python 包的版本及 GitHub 主页。如果你使用旧版本，请使用 pipenv update 命令更新版本。

 提示 如果你手动使用 pip 和 virtualenv 管理包和虚拟环境，可以使用 --upgrade 或 -U 选项（简写时 U 为大写）来更新包版本：pip install -U <包名称>

1.1.3 集成开发环境

如果你还没有顺手的文本编辑器，那么可以尝试一下 IDE (Integrated Development Environment，集成开发环境)。对于新手来说，IDE 的强大和完善会帮助你高效开发 Flask 程序，等到你熟悉了整个开发流程，可以换用更加轻量的编辑器以避免过度依赖 IDE。下面我们将介绍使用 PyCharm 开发 Flask 程序的主要准备步骤。

步骤 1 下载并安装 PyCharm

打开 PyCharm 的下载页面 (<http://jetbrains.com/pycharm/download/>)，单击你使用的操作系统选项卡，然后单击下载按钮。你可以选择试用专业版 (Professional Edition)，或是选择免费的社区版 (Community Edition)，如图 1-1 所示。

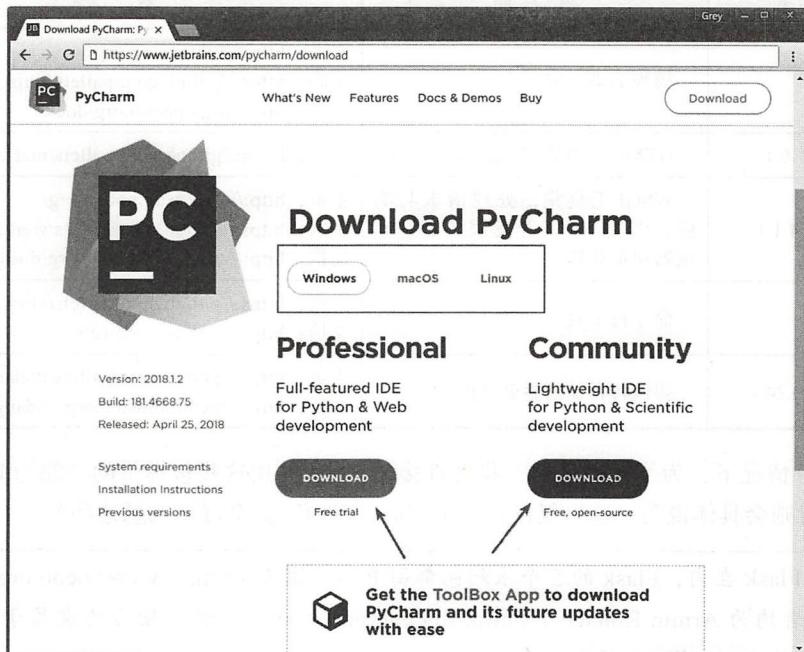


图 1-1 下载 PyCharm



附注 专业版有一个月的免费试用时间。如果你是学生，可以申请专业版的免费授权 (<https://www.jetbrains.com/student/>)。专业版提供了更多针对 Flask 开发的功能，比如创建 Flask 项目模板，Jinja2 语法高亮，与 Flask 命令行功能集成等。

PyCharm 的安装过程比较简单，这里不再详细说明，具体可以参考 <https://www.jetbrains.com/help/pycharm/requirements-installation-and-launching.html>。

步骤 2 创建项目

安装成功后，初始界面提供了多种方式创建新项目。这里可以单击“Open”，选择我们的 helloflask 文件夹。打开项目后的界面如图 1-2 所示，左边是项目目录树，右边是代码编辑区域。单击左下角的方形图标可以隐藏和显示工具栏，显示工具栏后，可以看到常用的 Python 交互控制台（Python Console）和终端（Terminal，即命令行工具）。

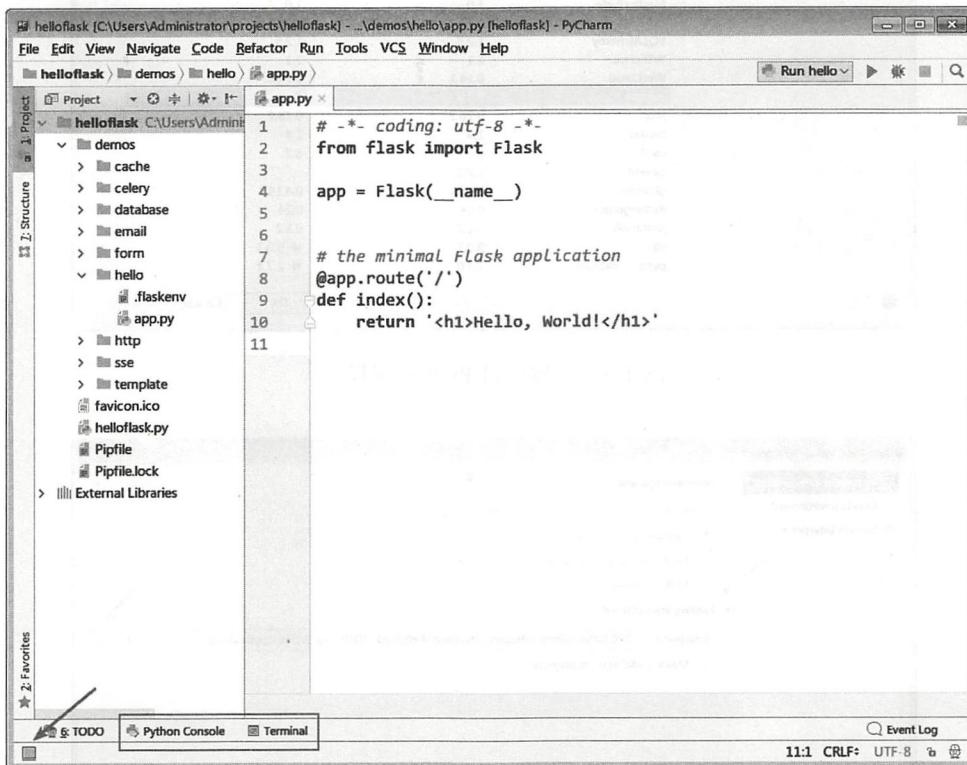


图 1-2 PyCharm 主界面

步骤 3 设置 Python 解释器

因为 PyCharm 目前还没有添加 Pipenv 集成支持（最新进展可访问 <https://youtrack.jetbrains.com/issue/PY-26492> 查看），我们需要手动使用 pipenv 命令安装依赖，同时还需要为项目设置正确的 Python 解释器。单击菜单栏中的 File→Settings 打开设置，然后单击 Project: helloflask-Project Interpreter 选项打开项目 Python 解释器设置窗口，如图 1-3 所示。

单击选择字段右侧的设置图标，然后单击“Add Local Python Interpreter”，在弹出的窗口选择 Virtualenv Environment→Existing environment，在下拉框或是自定义窗口找到我们之前创建的虚拟环境中的 Python 解释器路径，如图 1-4 所示。

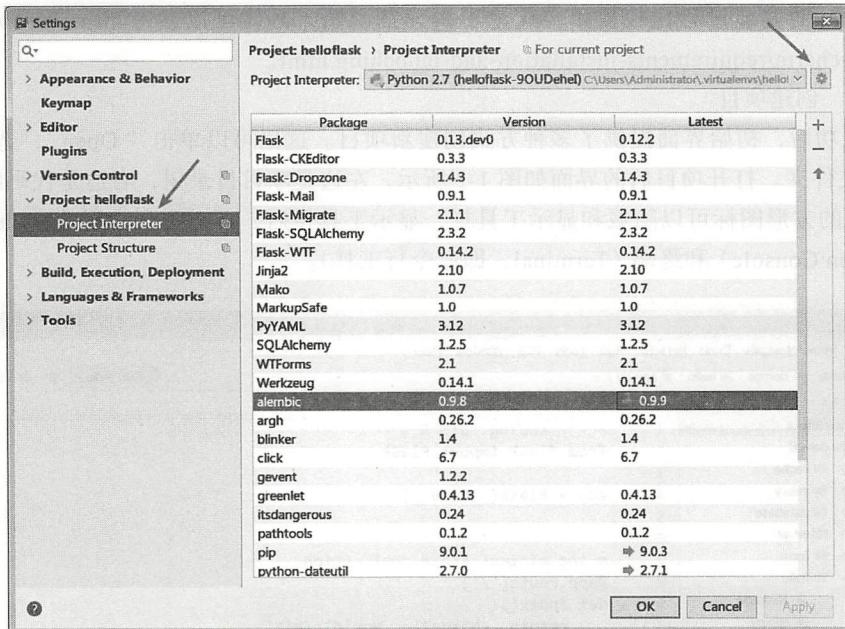


图 1-3 设置项目 Python 解释器

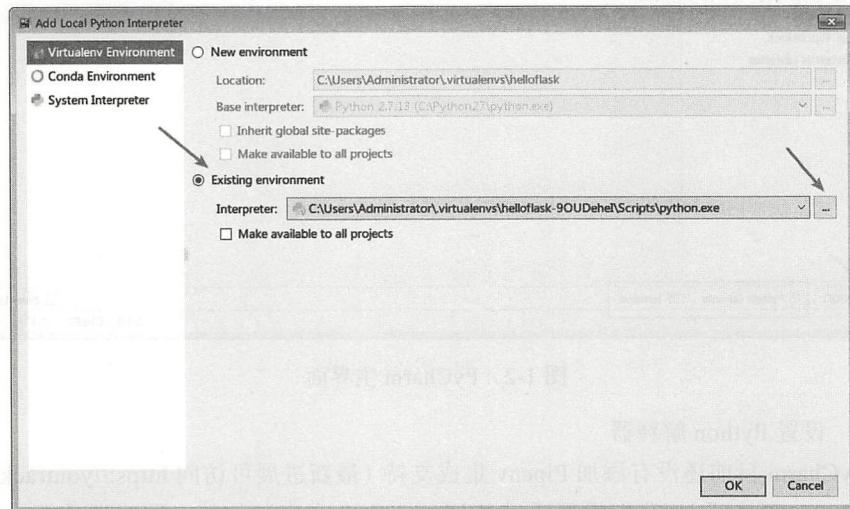


图 1-4 选择虚拟环境中的 Python 解释器

使用 `pipenv --venv` 命令可以查看项目对应的虚拟环境路径。Linux 或 macOS 系统下的路径类似 `~/.local/share/virtualenvs/helloflask-kSN7ec1K/bin/python` 或 `~/.virtualenvs/helloflask-kSN7ec1K/bin/python`, Windows 系统的路径类似 `C:\Users\Administrator\virtualenvs\helloflask-5Pa0ZfZw\Scripts\python.exe`。

正确设置以后，重新创建一个 Terminal 会话，你会发现命令行提示符前出现了虚拟环境名称，说明虚拟环境已经激活。以后每次打开项目，PyCharm 都会自动帮你激活虚拟环境，并且把工作目录定位到项目根目录。具体行为你也可以在 Settings→Tools→Terminal 中设置。

 **附注** 你可以通过 PyCharm 提供的入门教程 (<https://www.jetbrains.com/pycharm/documentation/>) 来了解 PyCharm 的更多用法。

最后，我们的 Web 程序需要在 Web 浏览器中访问，所以你还需要安装一个 Web 浏览器，推荐使用 Firefox (<https://www.mozilla.org/firefox/>) 或 Chrome (<https://www.google.com/chrome/>)。现在你已经做好了一切准备，Flask 之旅正式启程了！下面我们会通过一个最小的 Flask 程序来了解 Flask 的基本运作方式。

1.2 Hello, Flask!

本书的第一部分每一章对应一个示例程序，分别存储在 demos 目录下的不同文件夹中。本章的示例程序在 helloflask/demos/hello 目录下，使用下面的命令切换到该目录：

```
$ cd demos/hello
```

在 hello 目录下的 app.py 脚本中包含一个最小的 Flask 程序，如代码清单 1-1 所示。

代码清单 1-1 hello/app.py：最小的 Flask 程序

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return '<h1>Hello Flask!</h1>'
```

你也许已经猜到它能做什么了，下面让我们先来一步步分解这个程序。

 **提示** 对于简单的程序来说，程序的主模块一般命令为 app.py。你也可以使用其他名称，比如 hello.py，但是要避免使用 flask.py，因为这和 Flask 本身冲突。

1.2.1 创建程序实例

我们安装 Flask 时，它会在 Python 解释器中创建一个 flask 包，我们可以通过 flask 包的构造文件导入所有开放的类和函数。我们先从 flask 包导入 Flask 类，这个类表示一个 Flask 程序。实例化这个类，就得到我们的程序实例 app：

```
from flask import Flask
app = Flask(__name__)
```

传入 Flask 类构造方法的第一个参数是模块或包的名称，我们应该使用特殊变量 __name__。

Python 会根据所处的模块来赋予 `_name_` 变量相应的值，对于我们的程序来说（`app.py`），这个值为 `app`。除此之外，这也会帮助 Flask 在相应的文件夹里找到需要的资源，比如模板和静态文件。

 提示 Flask 类是 Flask 的核心类，它提供了很多与程序相关的属性和方法。在后面，我们经常会直接在程序实例 `app` 上调用这些属性和方法来实现相关功能。在第一次提及 Flask 类中的某个方法或属性时，我们会直接以实例方法 / 属性的形式写出，比如存储程序名称的属性为 `app.name`。

1.2.2 注册路由

在一个 Web 应用里，客户端和服务器上的 Flask 程序的交互可以简单概括为以下几步：

- 1) 用户在浏览器输入 URL 访问某个资源。
- 2) Flask 接收用户请求并分析请求的 URL。
- 3) 为这个 URL 找到对应的处理函数。
- 4) 执行函数并生成响应，返回给浏览器。
- 5) 浏览器接收并解析响应，将信息显示在页面中。

在上面这些步骤中，大部分都由 Flask 完成，我们要做的只是建立处理请求的函数，并为其定义对应的 URL 规则。只需为函数附加 `app.route()` 装饰器，并传入 URL 规则作为参数，我们就可以让 URL 与函数建立关联。这个过程我们称为注册路由（route），路由负责管理 URL 和函数之间的映射，而这个函数则被称为视图函数（view function）。

 附注 路由的含义可以从字面意义理解，作为动词时，它的含义是“按某路线发送”，即调用与请求 URL 对应的视图函数。

在这个程序里，`app.route()` 装饰器把根地址 / 和 `index()` 函数绑定起来，当用户访问这个 URL 时就会触发 `index()` 函数。这个视图函数可以像其他普通函数一样执行任意操作，比如从数据库中获取信息，获取请求信息，对用户输入的数据进行计算和处理等。最后，视图函数返回的值将作为响应的主体，一般来说，响应的主体就是呈现在浏览器窗口的 HTML 页面。在最小程序中，视图函数 `index()` 返回一行问候：

```
@app.route('/')
def index():
    return '<h1>Hello, World!</h1>'
```

虽然这个程序相当简单，但它却是大部分 Flask 程序的基本模式。在复杂的程序中，我们会有许多个视图函数分别处理不同 URL 的请求，在视图函数中会完成更多的工作，并且返回包含各种链接、表单、图片的 HTML 文件，而不仅仅是一行字符串。返回的页面中的链接又会指向其他 URL，被单击后触发对应的视图函数，获得不同的返回值，从而显示不同的页面，这就是我们浏览网页时的体验。

 提示 route() 装饰器的第一个参数是 URL 规则，用字符串表示，必须以斜杠 (/) 开始。这里的 URL 是相对 URL (又称为内部 URL)，即不包含域名的 URL。以域名 www.helloflask.com 为例，“/”对应的是根地址 (即 www.helloflask.com)，如果把 URL 规则改为 “/hello”，则实际的绝对地址 (外部地址) 是 www.helloflask.com/hello。

假如这个程序部署在域名为 www.helloflask.com 的服务器上，当启动服务器后，只要你在浏览器里访问 www.helloflask.com，就会看到浏览器上显示一行“Hello, Flask!”问候。

 附注 URL (Uniform Resource Locator, 统一资源定位符) 正是我们使用浏览器访问网页时输入的网址，比如 http://helloflask.com/。简单来说，URL 就是指向网络中某个资源的地址。

1. 为视图绑定多个 URL

一个视图函数可以绑定多个 URL，比如下面的代码把 /hi 和 /hello 都绑定到 say_hello() 函数上，这就会为 say_hello 视图注册两个路由，用户访问这两个 URL 均会触发 say_hello() 函数，获得相同的响应，如代码清单 1-2 所示。

代码清单 1-2 hello/app.py：绑定多个 URL 到同一视图函数

```
@app.route('/hi')
@app.route('/hello')
def say_hello():
    return '<h1>Hello, Flask!</h1>'
```

2. 动态 URL

我们不仅可以为视图函数绑定多个 URL，还可以在 URL 规则中添加变量部分，使用“<变量名>”的形式表示。Flask 处理请求时会把变量传入视图函数，所以我们可以添加参数获取这个变量值。代码清单 1-3 中的视图函数 greet()，它的 URL 规则包含一个 name 变量。

代码清单 1-3 hello/app.py：添加 URL 变量

```
@app.route('/greet/<name>')
def greet(name):
    return '<h1>Hello, %s!</h1>' % name
```

因为 URL 中可以包含变量，所以我们将传入 app.route() 的字符串称为 URL 规则，而不是 URL。Flask 会解析请求并把请求的 URL 与视图函数的 URL 规则进行匹配。比如，这个 greet 视图的 URL 规则为 /greet/<name>，那么类似 /greet/foo、/greet/bar 的请求都会触发这个视图函数。

 顺便说一句，虽然示例中的 URL 规则和视图函数名称都包含相同的部分 (greet)，但这并不是必须的，你可以自由修改 URL 规则和视图函数名称。

这个视图返回的响应会随着请求 URL 中的 name 变量而变化。假设程序运行在 http://helloflask.com 上，当我们在浏览器里访问 http://helloflask.com/hello/Grey 时，可以看到浏览器

上显示“Hello, Grey!”。

当 URL 规则中包含变量时，如果用户访问的 URL 中没有添加变量，比如 /greet，那么 Flask 在匹配失败后会返回一个 404 错误响应。一个很常见的行为是在 app.route() 装饰器里使用 defaults 参数设置 URL 变量的默认值，这个参数接收字典作为输入，存储 URL 变量和默认值的映射。在下面的代码中，我们为 greet 视图新添加了一个 app.route() 装饰器，为 /greet 设置了默认的 name 值：

```
@app.route('/greet', defaults={'name': 'Programmer'})
@app.route('/greet/<name>')
def greet(name):
    return '<h1>Hello, %s!</h1>' % name
```

这时如果用户访问 /greet，那么变量 name 会使用默认值 Programmer，视图函数返回 <h1>Hello, Programmer! </h1>。上面的用法实际效果等同于：

```
@app.route('/greet')
@app.route('/greet/<name>')
def greet(name='Programmer'):
    return '<h1>Hello, %s!</h1>' % name
```

1.3 启动开发服务器

Flask 内置了一个简单的开发服务器（由依赖包 Werkzeug 提供），足够在开发和测试阶段使用。

 **注意** 在生产环境需要使用性能够好的生产服务器，以提升安全和性能，具体在本书第三部分会进行介绍。

1.3.1 Run , Flask , Run!

Flask 通过依赖包 Click 内置了一个 CLI (Command Line Interface，命令行交互界面) 系统。当我们安装 Flask 后，会自动添加一个 flask 命令脚本，我们可以通过 flask 命令执行内置命令、扩展提供的命令或是我们自己定义的命令。其中，flask run 命令用来启动内置的开发服务器：

```
$ flask run
 * Environment: production
WARNING: Do not use the development server in a production environment.
Use a production WSGI server instead.
 * Debug mode: off
      * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

 **注意** 确保执行命令前激活了虚拟环境 (pipenv shell)，否则需要使用 pipenv run flask run 命令启动开发服务器。后面将不再提示。

你可以执行 flask --help 查看所有可用的命令。