第 1 章 初识 Flask（部分内容试读）

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，更方便高效的编写 Web 程序。因为 Flask 保持核心简单而易于扩展，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 不会给你做决定，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克隆到本地，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）页面点击了 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，这将克隆你自己的一份派生仓库，你可以自由的修改和提交代码。

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，可以看做是 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>=2.7.9 或 Python 3>=3.4，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，截止 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 如果你不想全局安装，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 enviroment）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，并且可以方便的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，但是为了更方便的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。如果你想将虚拟环境文件夹在项目目录内创建，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，我们通常会显式的激活虚拟环境。在 Pipenv 中，可以使用 pipenv shell 命令显式的激活虚拟环境：

```
$ 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。

除了显式的激活虚拟环境，Pipenv 还提供了一个 pipenv run 命令，这个命令允许你不显式的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，和显式的激活/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的依赖包，然后使用 pip 或 Pipenv 安装，这些重复无用的工作当然应该被避免。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 会被自动更新。

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，Flask 使用这些包提供的功能我会直接称之为 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 Enviroment，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的开发 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/），点击你使用的操作系统选项卡，然后点击下载按钮。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。这里可以点击“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。点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）。

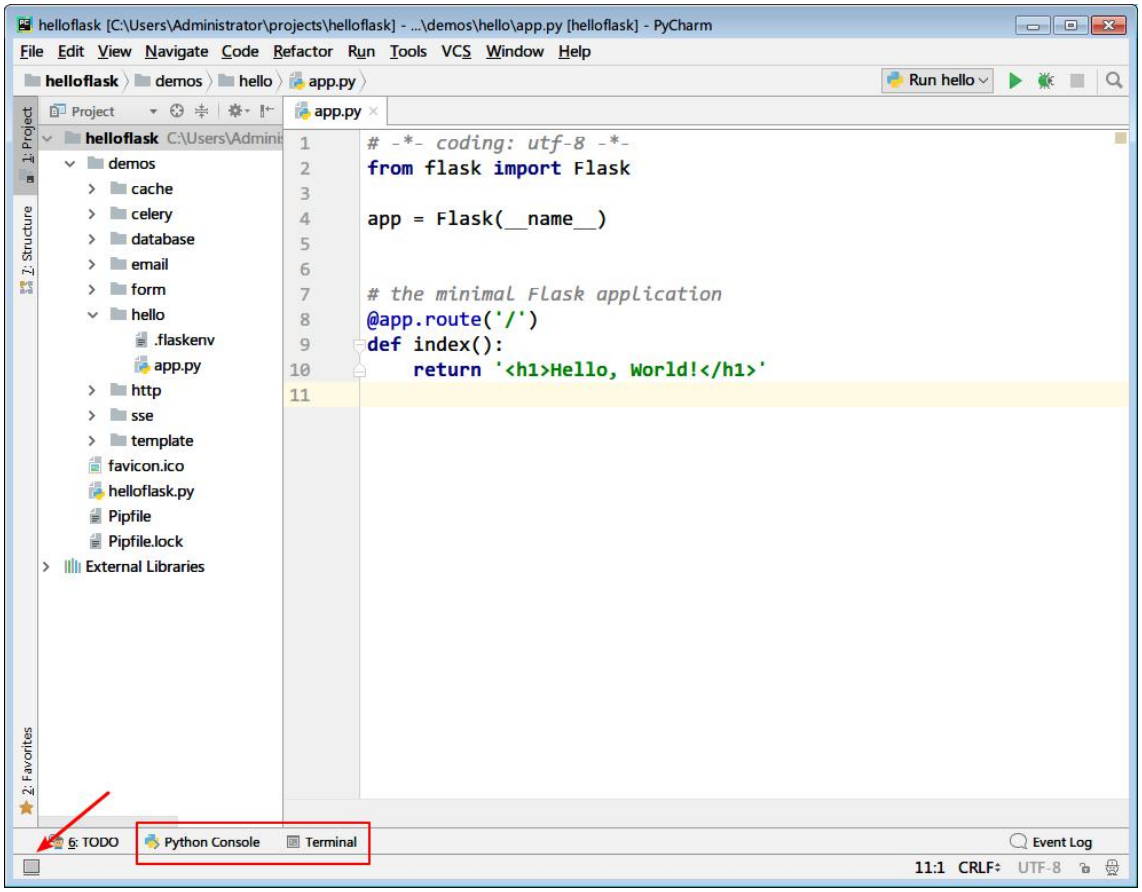

图 1-2 PyCharm 主界面

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。点击菜单栏中的 File - Settings 打开设置，然后点击 Project: helloflask - Project Interpreter 选项打开项目 Python 解释器设置窗口，如图 1-3 所示。

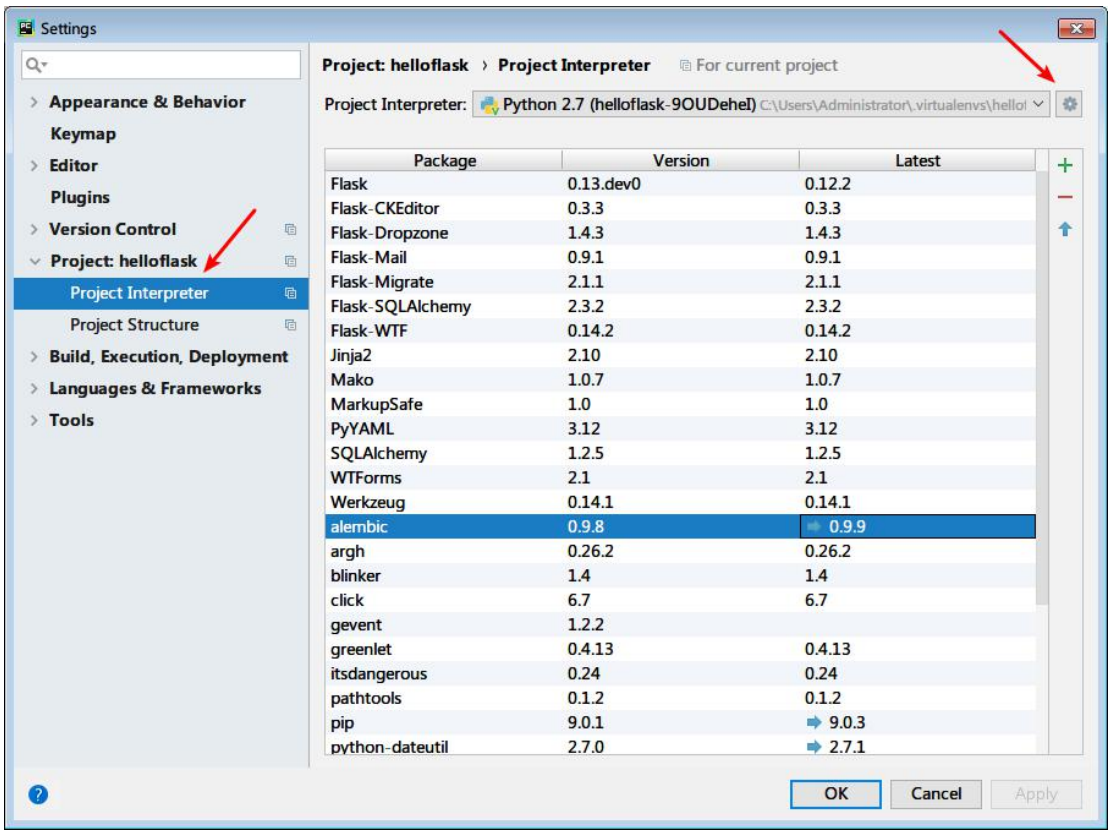

图 1-3 设置项目 Python 解释器

点击选择字段右侧的设置图标，然后点击“Add Local Python Interpreter”，在弹出的窗口选择 Virtualenv Enviroment - Existing enviroment，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。

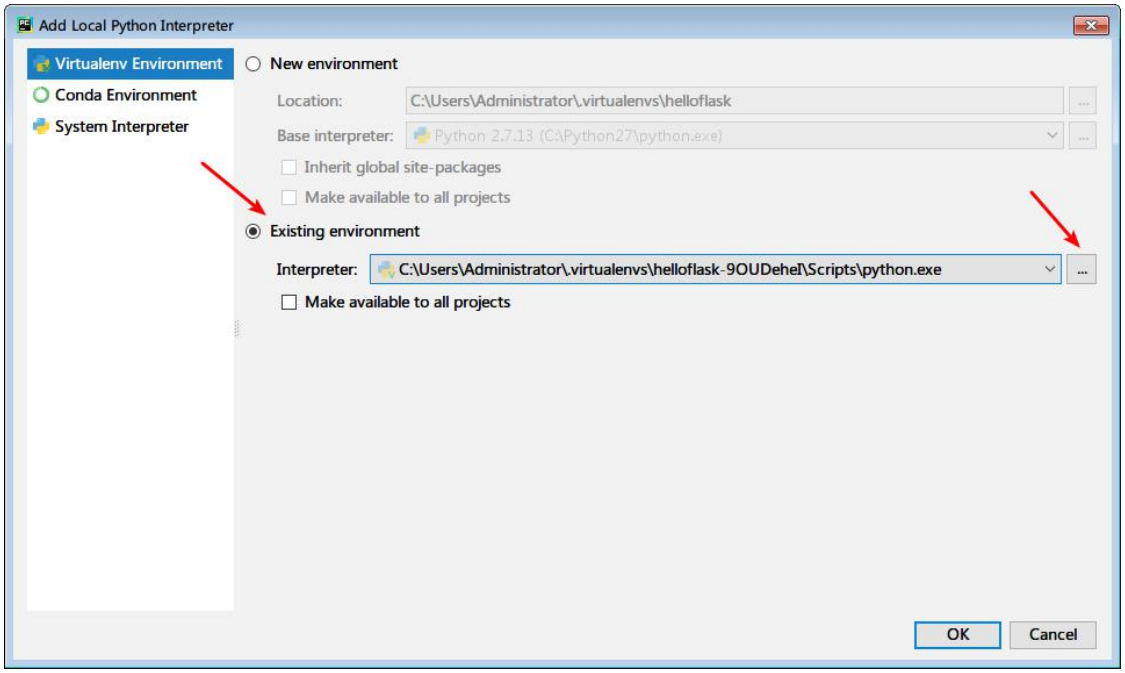

图 1-4 选择虚拟环境中的 Python 解释器

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。这个过程我们称之为注册路由（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，被点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）则是 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 Lacator，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/。简单的说，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()的字符串称之为 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。

附注 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，这个参数接受字典作为输入，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。

提示 如果执行 flask run 命令后显示命令未找到提示（command not found）或其他错误，可以尝试使用 python -m flask run 启动服务器，其他命令亦同。

flask run 命令运行的开发服务器默认会监听 http://127.0.0.1:5000/地址（按 Crtl+C 退出），并开启多线程支持。当我们打开浏览器访问这个地址时，会看到网页上显示“Hello, World!”，如图 1-5 所示。

图 1-5 “Hello, World!”程序主页

提示 http://127.0.0.1 即 localhost，是指向本地机的 IP 地址，一般用来测试。Flask 默认使用 5000 端口，对于上面的地址，你也可以使用 http://localhost:5000/。在本书中这两者会交替使用，除了地址不同外，两者没有实际区别，即域名和 IP 地址的映射关系。

提示 旧的启动开发服务器的方式是使用 app.run()方法，目前已不推荐使用（deprecated）。

1. 自动发现程序实例

一般来说，在执行 flask run 命令运行程序前，我们需要提供程序实例所在模块的位置。我们在上面可以直接运行程序，是因为 Flask 会自动探测程序实例，自动探测存在下面这些规则：

- 从当前目录寻找 app.py 和 wsgi.py 模块，并从中寻找名为 app 或 application 的程序实例。
- 从环境变量 FLASK_APP 对应的值寻找名为 app 或 application 的程序实例。

因为我们的程序主模块命名为 app.py，所以 flask run 命令会自动在其中寻找程序实例。如果你的程序主模块是其他名称，比如 hello.py，那么需要设置环境变量 FLASK_APP，将包含程序实例的模块名赋值给这个变量。Linux 或 macOS 系统使用 export 命令：

```
$ export FLASK_APP=hello
```

在 Windows 系统中使用 set 命令：

```
> set FLASK_APP=hello
```

2. 管理环境变量

Flask 的自动发现程序实例机制还有第三条规则：如果安装了 python-dotenv，那么在使用 flask run 或其他命令时会使用它自动从.flaskenv 文件和.env 文件中加载环境变量。

附注 当安装了 python-dotenv 时，Flask 在加载环境变量的优先级是：手动设置的环境变量>.env 中设置的环境变量>.flaskenv 设置的环境变量。

除了 FLASK_APP，在后面我们还会用到其他环境变量。环境变量在新创建命令行窗口或重启电脑后就清除了，每次都要重设变量有些麻烦。而且如果你同时开发多个 Flask 程序，这个 FLASK_APP 就需要在不同的值之间切换。为了避免频繁设置环境变量，我们可以使用 python-dotenv 管理项目的环境变量，首先使用 Pipenv 将它安装到虚拟环境：

```
$ pipenv install python-dotenv
```

我们在项目根目录下分别创建两个文件：.env 和.flaskenv。.flaskenv 用来存储和 Flask 相关的

公开环境变量，比如 FLASK_APP。而.env 用来存储包含敏感信息的环境变量，比如后面我们会用来配置 Email 服务器的账户名与密码。在.flaskenv 或.env 文件中，环境变量使用键值对的形式定义，每行一个，以#开头的为注释，如下所示：

```
SOME_VAR=1

# 这是注释

FOO="BAR"
```

注意 .env 包含敏感信息，除非是私有项目，否则绝对不能提交到 Git 仓库中。当你开发一个新项目时，记得把它的名称添加到.gitignore 文件中，这会告诉 Git 忽略这个文件。gitignore 文件是一个名为.gitignore 的文本文件，它存储了项目中 Git 提交时的忽略文件规则清单。Python 项目的.gitignore 模板可以参考 https://github.com/github/gitignore/blob/master/Python.gitignore。使用 PyCharm 编写程序时会产生一些配置文件，这些文件保存在项目根目录下的.idea 目录下，关于这些文件的忽略设置可以参考 https://www.gitignore.io/api/pycharm。

3. 使用 PyCharm 运行服务器

在 PyCharm 中，虽然我们可以使用内置的命令行窗口执行命令以启动开发服务器，但是在开发时使用 PyCharm 内置的运行功能更加方便。在 2018.1 版本后的专业版添加了 Flask 命令行支持，在旧版本或社区版中，如果要使用 PyCharm 运行程序，还需要进行一些设置。

首先，在 PyCharm 中，点击菜单栏中的 Run - Edit Configurations 打开运行配置窗口。在 PyCharm 中设置一个运行配置具体的步骤序号已经在图 1-6 中标出。

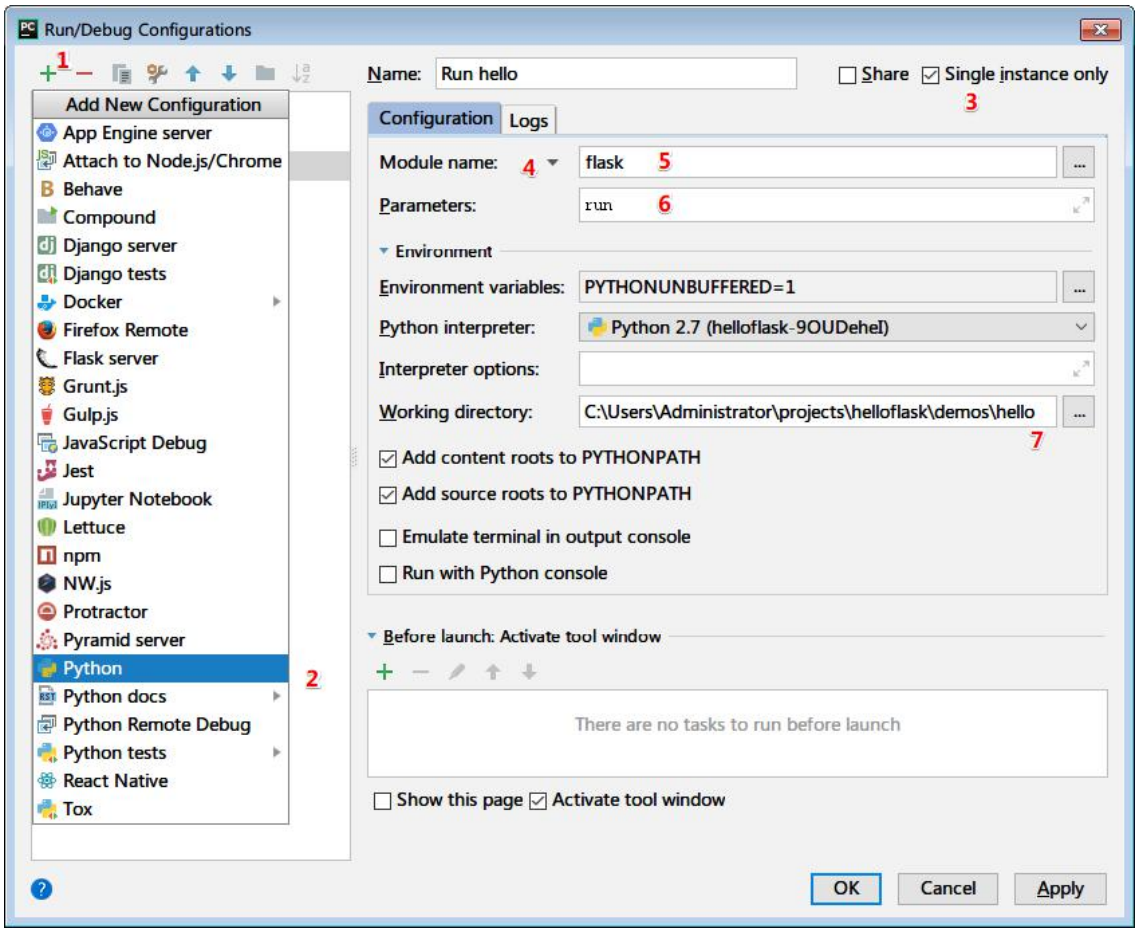

图 1-6 运行 Flask 程序的配置

打开新建配置窗口后，具体的步骤如下所示：

步骤 1 点击左侧的“+”符号打开下拉列表

步骤 2 新建一个 Python 类型的运行配置（如果你使用的是专业版，则可以直接选择 Flask server），并在右侧的 Name 字段输入一个合适的名称，比如“Run hello”。

步骤 3 勾选“Single instance only”。

步骤 4 将第一项配置字段通过下列选项选为“Module Name”

步骤 5 填入模块名称 flask。

步骤 6 第二栏的“Parameters”填入要执行的命令 run，你也可以附加其他启动选项。

步骤 7 在“Working directory”字段中选择程序所在的目录作为工作目录。

提示 我们可以点击左上方的复制图标复制一份配置，然后稍加修改就可以用于其他 flask 命令，包括扩展提供的命令，或是我们自定义的命令。

现在点击 Apply 或 OK 保存并关闭窗口。在 PyCharm 右上方选择我们创建的运行配置，然后

点击绿色三角形的运行按钮即可启动开发服务器。

注意 因为本章示例程序的模块名称为 app.py，Flask 会自动从中寻找程序实例，所以我们在 PyCharm 中的运行设置可以正确启动程序。如果你不打算使用 python-dotenv 来管理环境变量，那么需要修改 PyCharm 的运行配置：在 Enviroment variable 字段中添加环境变量 FLASK_APP 并设置正确的值。

1.3.2 更多的启动选项

1. 使服务器外部可见

我们在上面启动的 Web 服务器默认是对外不可见的，可以在 run 命令后添加--host 选项将主机地址设为 0.0.0.0 使其对外可见：

```
$ flask run --host=0.0.0.0
```

这会让服务器监听所有外部请求。个人计算机（主机）一般没有公网 IP（公有地址），所以你的程序只能被局域网内的其他用户通过你的个人计算机的内网 IP（私有地址）访问，比如你的内网 IP 为 192.168.191.1。当局域网内的其他用户访问 http://192.168.191.1:5000 时，也会看到浏览器里显示一行“Hello, Flask!”。

提示 把程序安装在拥有公网 IP 的服务器上，让互联网上的所有人都可以访问是我们最后要介绍的程序部署部分的内容。如果你迫切的想把你的程序分享给朋友们，可以考虑使用 ngrok（https://ngrok.com/）、Localtunnel（https://localtunnel.github.io/www/）等内网穿透/端口转发工具。

2. 改变默认端口

Flask 提供的 Web 服务器默认监听 5000 端口，你可以在启动时传入参数来改变它：

```
$ flask run --port=8000
```

这时服务器会监听来自 8000 端口的请求，程序的主页地址也相应变成了 http://localhost:8000/。

附注 执行 flask run 命令时的 host 和 port 选项也可以通过环境变量 FLASK_RUN_HOST 和 FLASK_RUN_PORT 设置。事实上，Flask 内置的命令都可以使用这种模式定义默认选项值，即“FLASK_<COMMAND>_<OPTION>”，你可以使用 flask --help 命令查看所有可用的命令。

1.3.3 设置运行环境

开发环境（Development Enviroment）和生产环境（Production Enviroment）是我们后面会频繁接触到的概念。开发环境是指在我们在本地编写和测试程序时的计算机环境，而生产环境与在开发环境相对，它指的是网站部署上线供用户访问时的服务器环境。

根据运行环境的不同，Flask 程序、扩展以及其他程序会改变相应的行为和设置。为了区分程序运行环境，Flask 提供了一个 FLASK_ENV 环境变量用来设置环境，默认为 production（生产）。在开发时，我们可以将其设为 development（开发），这会开启所有支持开发的特性。为了方便管理，我们将把环境变量 FLASK_ENV 的值写入.flaskenv 文件中：

```
FLASK_ENV=development
```

现在启动程序，你会看到下面的输出提示：

```
$ flask run
 * Environment: development
 * Debug mode: on
 * Debugger is active!
 * Debugger PIN: 202-005-064
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

在开发环境下，调试模式（Debug Mode）将被开启，这时执行 flask run 启动程序会自动激活 Werkzeug 内置的调试器（debugger）和重载器（reloader），它们会为开发带来很大的帮助。

提示 如果你想单独控制调试模式的开关，可以通过 FLASK_DEBUG 环境变量设置，设为 1 开启，设为 0 关闭，不过通常不推荐手动设置这个值。

注意 在生产环境中部署程序时，绝不能开启调试模式。尽管 PIN 码可以避免用户任意执行代码，提高攻击者利用调试器的难度，但并不能确保调试器完全安全，会带来巨大的安全隐患。而且攻击者可能会通过调试信息获取你的数据库结构等容易带来安全问题的信息。另一方面，调试界面显示的错误信息也会让普通用户感到困惑。

1. 调试器

Werkzeug 提供的调试器（debugger）非常强大，当程序出错时，我们可以在网页上看到详细的错误追踪信息，这在调试错误时非常有用。运行中的调试器如图 1-7 所示。

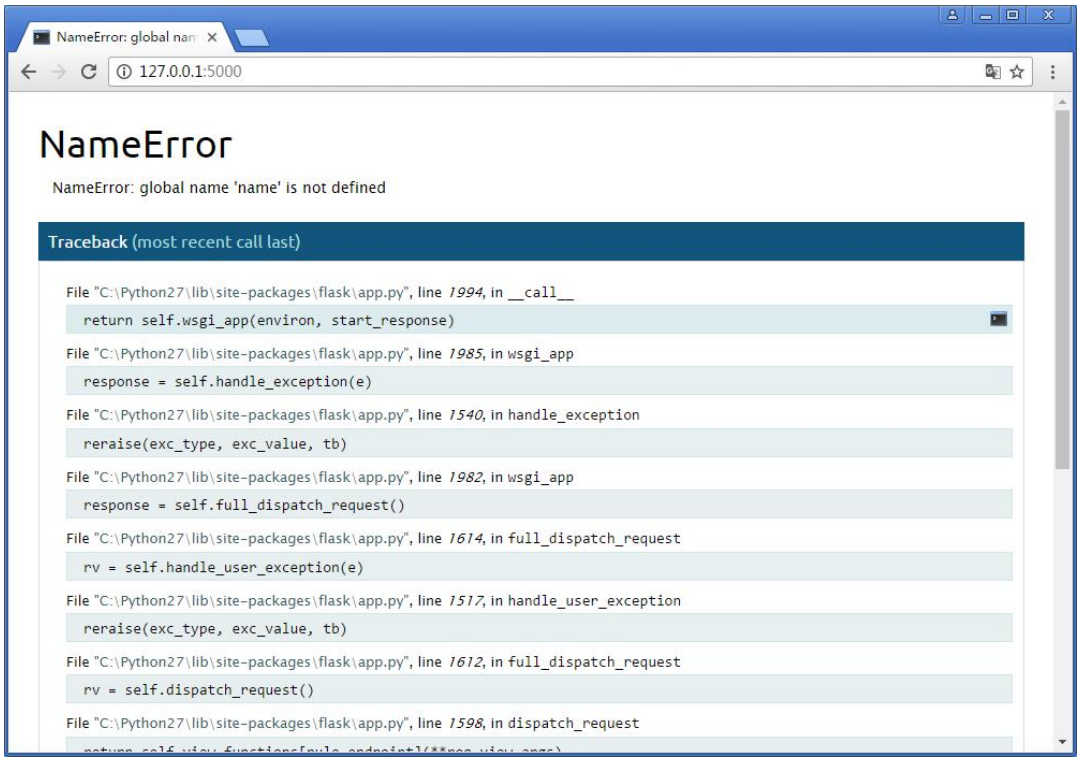

图 1-7 调试器界面

调试器允许你在错误页面上执行 Python 代码。点击错误信息右侧的命令行图标，会弹出窗口要求输入 PIN 码，也就是在启动服务器时命令行窗口打印出的调试器 PIN 码（Debugger PIN）。输入 PIN 码后，我们可以点击错误堆栈的某个节点右侧的命令行界面图标，这会打开一个包含代码执行上下文信息的 Python Shell，我们可以利用它来进行调试。

2. 重载器

当我们对代码做了修改后，期望的行为是这些改动立刻作用到程序上。重载器（Reloader）的作用就是监测文件变动，然后重新启动开发服务器。当我们修改了脚本内容并保存后，会在命令行看到下面的输出：

```
Detected change in '/path/to/app.py', reloading
```

```
 * Restarting with stat
```

默认会使用 Werkzeug 内置的 stat 重载器，它的缺点是耗电较严重，而且准确性一般。为了获得更优秀的体验，我们可以安装另一个用于监测文件变动的 Python 库 Watchdog，安装后 Werkzeug 会自动使用它来监测文件变动：

```
$ pipenv install watchdog --dev
```

因为这个包只在开发时才会用到，所以我们在安装命令后添加了一个--dev 选项，这用来把这个包声明为开发依赖。在 Pipfile 文件中，这个包会被添加到 dev-packages 部分。

不过，如果项目中使用了单独的 CSS 或 JavaScript 文件时，那么浏览器可能会缓存这些文件，从而导致对文件作出的修改不能立刻生效。在浏览器中，我们可以按下 Crtl+F5 或 Shift+F5 执行硬重载（hard reload），即忽略缓存并重载（刷新）页面。

提示 当在一个新电脑创建运行环境时，使用 pipenv install 命令时需要添加额外的--dev 选项才会安装 dev-packages 部分定义的开发依赖包。

1.4 Python Shell

本书有许多操作需要在 Python Shell（即 Python 交互式解释器）里执行。在开发 Flask 程序时，我们并不会直接使用 python 命令启动 Python Shell，而是使用 flask shell 命令：

```
$ flask shell
App: app [development]
Instance: Path/to/your/helloflask/instance
>>>
```

注意 和其他 flask 命令相同，执行这个命令前我们要确保程序实例可以被正常找到。

在本书中，如果代码片段前的提示符提示符为三个大于号，即“>>>”，那么就表示这些代码需要在使用 flask shell 命令打开的 Python Shell 中执行。

提示 Python Shell 可以执行 exit()或 quit()退出，在 Windows 系统上可以使用 Crtl+Z 并按 Enter 退出；在 Linux 和 macOS 则可以使用 Crtl+D 退出。

使用 flask shell 命令打开的 Python Shell 自动包含程序上下文，并且已经导入了 app 实例：

```
>>> app
<Flask 'app'>
>>> app.name
 'app'
```

附注 上下文（Context）可以理解为环境。为了正常运行程序，一些操作相关的状态和数据需要被临时保存下来，这些状态和数据被统称为上下文。在 Flask 中，上下文有两种，分别为程序上下文和请求上下文，后面我们会详细了解。

试读部分完结。

请访问本书主页 http://helloflask.com/book 了解更多信息。