Содержание

Предисловие

Flask отличается от других фреймворков тем, что позволяет разработчику сесть на место водителя и получить полный контроль над его приложением. Возможно, вам уже доводилось слышать фразу: «бороться с фреймворком». Такое происходит с большинством фреймворков при попытке реализовать нестандартное решение. Это может быть попытка использовать другой механизм управления базами данных или иной способ аутентификации пользователей. Отклонение от пути, предусмотренного разработчиками фреймворка, приносит массу неприятностей.

Фреймворк Flask не такой. Хотите использовать реляционную базу данных? Отлично, Flask поддерживает их. Предпочитаете базу данных NoSQL? Нет проблем, Flask способен работать и с ними. Хотите использовать механизм хранения данных собственной разработки или вообще решили обойтись без базы данных? Замечательно. Используя Flask, можно выбирать, какие его компоненты будут применяться в приложении, и даже писать собственные. Все в ваших руках!

Такая свобода объясняется тем, что фреймворк Flask изначально задумывался расширяемым. Он включает надежное ядро, обеспечивающее основные функциональные возможности, необходимые в любых веб-приложениях, и предполагает, что остальное будет реализовано сторонними разработчиками в форме расширений и, конечно же, вами.

В этой книге я расскажу о моих подходах к разработке веб-приложений с применением фреймворка Flask. Я не претендую на истину в последней инстанции, и вы должны рассматривать мои слова как рекомендации, а не как непреложное руководство к действию.

В большинстве книг, посвященных программированию, приводятся короткие фрагменты кода, демонстрирующие разные особенности обсуждаемых технологий по отдельности, оставляя за рамками «связывающий» код, необходимый для объединения этих разных фрагментов в действующее приложение. Я предпочел избрать иной подход. Все примеры, представленные в этой книге, являются частями единого приложения – сначала очень простого, а затем постепенно усложняющегося в каждой последующей главе. В начале пути это приложение состоит всего из нескольких строк кода, а к концу оно превращается в полноценное приложение социальных сетей и блогинга.

Кому адресована эта книга

Чтобы извлечь выгоду из этой книги, необходимо иметь некоторый опыт программирования на языке Python. Эта книга не предполагает предварительного знакомства с фреймворком Flask, но вы должны быть знакомы с такими понятиями языка Python, как пакеты, модули, функции, декораторы и объектно-ориентированное программирование. Нелишними будут также знакомство с исключениями и умение диагностировать проблемы по трассировке стека.

Следуя за примерами в книге, вы проведете немало времени в командной строке, поэтому вам также потребуются навыки работы в командной строке своей операционной системы.

Современные веб-приложения немыслимы без использования HTML, CSS и JavaScript. Приложение, разрабатываемое на протяжении всей книги, также использует их, но в самом тексте книги не приводятся подробности, касающиеся этих технологий. Знакомство с этими языками совершенно необходимо, если предполагаете писать законченные приложения, не прибегая к помощи разработчика, искушенного в клиентских технологиях.

Исходные тексты приложения, написанного для этой книги, я выложил в открытый доступ на сайте GitHub. Несмотря на то что GitHub позволяет загружать приложения в виде обычных ZIP- или TAR-архивов, я настоятельно рекомендую установить клиента Git и познакомиться с системой управления версиями, хотя бы с основными командами, позволяющими извлекать различные версии приложения непосредственно из репозитория. Короткий список команд, которые вам понадобятся, представлен в разделе «Использование программного кода примеров» ниже. Вы наверняка пожелаете применить систему управления версиями для собственных проектов, поэтому используйте эту книгу как повод изучить Git!

Наконец, не следует рассматривать эту книгу как полное и исчерпывающее руководство по фреймворку Flask. Здесь охватываются многие его особенности, но не упускайте из виду официальную документацию с описанием фреймворка[1].

Структура книги

Эта книга делится на три части.

---

[1] http://flask.pocoo.org/.

В первой части «Введение в Flask» исследуются основы разработки веб-приложений с применением фреймворка Flask и некоторых его расширений:

- глава 1 описывает установку и настройку фреймворка Flask;
- глава 2 погружает читателя в разработку простого приложения с помощью Flask;
- глава 3 знакомит с поддержкой шаблонов в приложениях Flask;
- глава 4 – с поддержкой веб-форм;
- глава 5 – с поддержкой баз данных;
- глава 6 – с поддержкой электронной почты;
- глава 7 описывает типичную структуру крупных и средних приложений.

Во второй части «Пример: приложение социального блогинга» описывается разработка открытого приложения социальных сетей и блогинга на основе фреймворка Flask, которое я создал для этой книги:

- глава 8 описывает реализацию системы аутентификации пользователей;
- глава 9 – реализацию системы ролей и привилегий пользователей;
- глава 10 – реализацию страниц профилей пользователей;
- глава 11 – создание интерфейса для блогинга;
- глава 12 – реализацию поддержки последователей;
- глава 13 – реализацию поддержки комментариев пользователей;
- глава 14 описывает реализацию прикладного программного интерфейса (Application Programming Interface, API).

В третьей части «Последняя миля» раскрываются некоторые важные задачи, не связанные с разработкой приложений непосредственно, которые необходимо решать перед публикацией приложения:

- глава 15 подробно описывает разные стратегии модульного тестирования;
- глава 16 представляет обзор приемов анализа производительности;
- глава 17 описывает варианты развертывания приложений на основе Flask в традиционных облачных окружениях;
- глава 18 перечисляет дополнительные источники информации.

Как работать с примерами программного кода

Примеры программного кода, описываемые в этой книге, доступны на сайте GitHub по адресу: **https://github.com/miguelgrinberg/flasky**.

История изменений в этом репозитории в точности соответствует порядку представления понятий в этой книге. Рекомендуется извлекать код из репозитория, начиная с самых ранних версий, и затем двигаться вперед по списку изменений, по мере чтения книги. Желающим сайт GitHub предоставляет возможность загружать каждое изменение в виде ZIP- или TAR-архива.

Если кто-то предпочтет извлекать исходный код примеров из репозитория Git, ему придется установить программу-клиента Git, которую можно бесплатно загрузить по адресу: **http://git-scm.com**. Ниже приводится команда, которая загрузит исходный код примеров из репозитория:

```
$ git clone https://github.com/miguelgrinberg/flasky.git
```

Команда `git clone` загрузит исходный код в каталог *flasky*, который будет создан в текущем каталоге. Этот каталог содержит не только исходный код; в него будет скопирован весь репозиторий Git с полной историей изменений приложения.

В первой главе будет предложено *извлечь* (*check out*) первоначальную версию приложения, и в соответствующих местах в книге будет предлагаться переходить дальше по истории изменений. Для извлечения исходного кода и перемещения по истории изменений используется команда `git checkout`, например:

```
$ git checkout 1a
```

Здесь `1a` в команде – это *тег*, именованная точка в истории развития проекта. Репозиторий размечен такими точками в соответствии с главами в книге, то есть тег `1a` соответствует начальной версии файлов приложения, описываемой в главе 1. Большинству глав соответствует более одного тега. Например, в репозитории имеются теги `5a`, `5b` и т. д., соответствующие последовательности версий, представленных в главе 5.

Помимо извлечения файлов из репозитория, может также потребоваться выполнить некоторые настройки. Например, иногда бывает необходимо установить дополнительные пакеты Python или внести

изменения в базу данных. Я буду сообщать об этом в соответствующие моменты.

В процессе чтения вам не придется изменять исходные файлы приложения, но если вы сделаете это, Git не позволит вам извлечь из репозитория следующую версию, чтобы не затереть локальных изменений. В этой ситуации, чтобы перейти к следующей версии, необходимо вернуть файлы в исходное состояние. Проще всего это сделать с помощью команды `git reset`:

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

Эта команда затрет все локальные изменения, поэтому, если вы не желаете потерять их, сохраните резервные копии файлов перед запуском этой команды.

Иногда бывает желательно обновлять локальную копию репозитория из внешнего, расположенного на сервере GitHub, куда разработчики могут добавлять улучшения, расширения и исправления ошибок. Сделать это можно следующей последовательностью команд:

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

Команды `git fetch` обновят историю изменений и список тегов в локальной копии репозитория в соответствии с удаленным репозиторием GitHub, но ни одна из них не оказывает влияния на сами файлы с исходным кодом. Обновление этих файлов осуществляется командой `git reset`. И снова имейте в виду, что команда `git reset` затрет все локальные изменения, которые вы могли выполнить.

Еще одной полезной операцией является получение различий между двумя версиями приложения. Она может пригодиться всем, кто пожелает лучше разобраться в изменениях. Выполняется эта операция командой `git diff`. Например, чтобы увидеть различия между ревизиями `2a` и `2b`:

```
$ git diff 2a 2b
```

Различия будут сохранены в формате файла «заплаты» (patch). Этот формат не очень удобен для просмотра изменений, особенно если прежде вам не приходилось сталкиваться с подобными файлами. Гораздо удобнее в этом отношении инструменты с графическим интерфейсом, предлагаемые проектом GitHub. Например, различия между ревизиями `2a` и `2b` можно посмотреть на странице GitHub: **https://github.com/miguelgrin berg/flasky/compare/2a…2b**.

Использование программного кода примеров

Данная книга призвана оказать вам помощь в решении ваших задач. Вы можете свободно использовать примеры программного кода из этой книги в своих приложениях и в документации. Вам не нужно обращаться в издательство за разрешением, если вы не собираетесь воспроизводить значительные по объему части программного кода. Например, если вы разрабатываете программу и используете в ней несколько отрывков программного кода из книги, вам не нужно обращаться за разрешением. Однако в случае продажи или распространения компакт-дисков с примерами из этой книги вам необходимо получить разрешение от издательства O'Reilly. Если вы отвечаете на вопросы, цитируя данную книгу или примеры из нее, получение разрешения не требуется. Но при включении существенных объемов программного кода примеров из этой книги в вашу документацию вам необходимо будет получить разрешение издательства.

Мы приветствуем, но не требуем добавлять ссылку на первоисточник при цитировании. Под ссылкой на первоисточник мы подразумеваем указание авторов, издательства и ISBN. Например: «Flask Web Development by Miguel Grinberg (O'Reilly). Copyright 2014 Miguel Grinberg, 978-1-449-3726-2».

Если вам кажется, что использование примеров из книги нарушает правила добросовестного применения или условия, сформулированные выше, обращайтесь по адресу **permissions@oreilly.com**.

Типографские соглашения

В книге приняты следующие соглашения:

Курсив
: Применяется для выделения новых терминов, имен файлов и их расширений.

`Моноширинный шрифт`
: Применяется для представления листингов программного кода, а также элементов программ, таких как имена переменных и функций, баз данных, типов данных, переменных окружения, инструкций и ключевых слов.

Моноширинный жирный

Используется для выделения команд или других фрагментов текста, которые вводятся пользователем.

Моноширинный курсив

Используется для выделения текста, который должен замещаться значениями, предоставляемыми пользователями, или значениями, определяемыми контекстом.

 Так выделяются советы и предложения.

 Так выделяются примечания общего характера.

 Так выделяются предупреждения и предостережения.

Safari® Books Online

Safari Books Online (**www.safaribooksonline.com**) – это виртуальная библиотека, содержащая авторитетную информацию в виде книг и видеоматериалов, созданных ведущими специалистами в области технологий и бизнеса.

Профессионалы в области технологий, разработчики программного обеспечения, веб-дизайнеры, а также бизнесмены и творческие работники используют Safari Books Online как основной источник информации для проведения исследований, решения проблем, обучения и подготовки к сертификационным испытаниям.

Библиотека Safari Books Online предлагает широкий выбор продуктов и тарифов для организаций, правительственных учреждений и физических лиц. Подписчики имеют доступ к поисковой базе данных, содержащей информацию о тысячах книг, видеоматериалов и рукописей от таких издателей, как O'Reilly Media, Prentice Hall Professional, Addison-Wesly Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, и десятков других. За подробной информацией о Safari Books Online обращайтесь по адресу: **http://www.safaribooksonline.com/**.

Ждем ваших отзывов

Направляйте свои комментарии и вопросы, касающиеся этой книги, по обычной почте:

O'Reilly Media
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в Соединенных Штатах Америки или в Канаде)
707-829-0515 (международный)
707-829-0104 (факс)

Список опечаток, файлы с примерами и другую дополнительную информацию вы найдете на сайте книги: **http://www.bit.ly/flask-web-dev**.

Свои пожелания и вопросы технического характера отправляйте по адресу: **bookquestions@oreilly.com**.

Дополнительную информацию о наших книгах, курсах, конференциях и новостях вы найдете на веб-сайте издательства O'Reilly: **http://www.oreilly.com**.

Ищите нас на Facebook: **http://facebook.com/oreilly**.

Читайте нас в Твиттере: **http://twitter.com/oreillymedia**.

Смотрите нас на YouTube: **http://www.youtube.com/oreillymedia**.

Благодарности

Едва ли я смог бы написать эту книгу в одиночку. Огромную помощь оказали мне моя семья, коллеги, старые друзья и новые, с которыми я познакомился попутно.

Я хотел бы поблагодарить Брендана Кехлера (Brendan Kohler) за подробный технический отзыв и за его помощь в работе над главой о прикладных программных интерфейсах. Я также многим обязан Дэвиду Баумголду (David Baumgold), Тодду Бранхоффу (Todd Brunhoff), Сесиль Рок (Cecil Rock) и Мэтью Хьюгасу (Matthew Hugues), просматривавшим рукопись на разных стадиях готовности и дававшим мне очень полезные советы относительно того, о чем писать и как организовать материал.

Разработка примеров для этой книги потребовала значительных усилий. Я благодарен Даниэлю Хофманну (Daniel Hofmann), который просмотрел весь код приложения и внес множество советов по его улучшению. Я также хочу сказать спасибо моему сыну-подростку Дилану Гринбергу (Dylan Grinberg), который забросил свою игру на несколько выходных и помогал мне тестировать код на разных платформах.

Издательство O'Reilly запустило замечательную программу под названием «Early Release» (ранний выпуск), позволяющую нетерпе-

ливым читателям получить доступ к книге в процессе ее создания. Некоторые из читателей первых выпусков моей книги приняли участие в полезных обсуждениях и обмене опытом, что помогло существенно улучшить ее. Мне хотелось бы поблагодарить Сандипа Гупту (Sundeep Gupta), Дэна Кэрона (Dan Caron), Брайана Уисти (Brian Wisti) и Коди Скотта (Cody Scott) за их помощь в работе над этой книгой.

Сотрудники O'Reilly Media всегда благоволили мне. Прежде всего я хотел бы поблагодарить моего прелестного редактора Меган Бланшетт (Meghan Blanchette) за ее поддержку, советы и помощь с самой первой нашей встречи. Благодаря Мег я получил незабываемый опыт работы над моей первой книгой.

В заключение я хотел бы поблагодарить замечательное сообщество пользователей Flask.

Часть I

Введение в Flask

Глава 1

Установка

Flask[1] – это очень маленький фреймворк, такой маленький, что его часто называют «микрофреймворк». Он настолько мал, что после непродолжительного знакомства с ним вы сможете читать и понимать его исходный код.

Но быть маленьким не означает давать меньше, чем дают другие фреймворки. Flask изначально проектировался как расширяемый фреймворк – он имеет монолитное ядро, реализующее основные службы, а все остальное поддерживается посредством расширений. Поскольку вы можете выбирать только необходимые пакеты, в результате получается достаточно ограниченный комплект программных средств, не поддающийся неконтролируемому разбуханию и в точности соответствующий вашим потребностям.

Фреймворк Flask имеет две основные зависимости. Подсистемы маршрутизации, отладки и интерфейса WSGI (Web Server Gateway Interface) заимствованы из проекта Werkzeug, а поддержка шаблонов – из проекта Jinja2. Проекты Werkzeug и Jinja2 были созданы основными разработчиками Flask.

Flask не имеет встроенной поддержки доступа к базам данных, проверки веб-форм, аутентификации пользователей или других высокоуровневых задач. Существует и множество иных ключевых служб, необходимых большинству веб-приложений и доступных в виде расширений, интегрируемых с основными пакетами. Как разработчик вы можете выбирать расширения, лучше всего подходящие для вашего проекта, или даже писать собственные, если чувствуете, что вам это удастся лучше. Этим Flask отличается от крупных фреймворков, где выбор уже сделан за вас, который очень сложно изменить, если вообще возможно.

В данной главе вы узнаете, как установить Flask. Единственное предварительное условие – наличие компьютера с установленным языком Python.


Код примеров был протестирован с Python 2.7 и Python 3.3, поэтому рекомендуется использовать одну из этих двух версий.

[1] http://flask.pocoo.org.

Использование виртуальных окружений

Самый удобный способ установки Flask – воспользоваться виртуальным окружением. Виртуальное окружение – это отдельная копия интерпретатора Python, в которую можно установить пакеты, не оказывая влияния на глобальный интерпретатор Python, установленный в системе.

Виртуальные окружения удобны тем, что предотвращают конфликты между версиями и захламление системного интерпретатора Python посторонними пакетами. Создание виртуального окружения для каждого приложения гарантирует доступность только необходимых пактов, при этом системная установка интерпретатора остается чистой и служит всего лишь ресурсом для создания виртуальных окружений. Как дополнительное преимущество виртуальные окружения не требуют наличия у вас прав администратора.

Виртуальные окружения можно создавать с помощью сторонней утилиты `virtualenv`. Чтобы проверить наличие утилиты в системе, введите следующую команду:

```
$ virtualenv --version
```

Если в ответ вы получите сообщение об ошибке, значит, вам придется установить утилиту.

Python 3.3 включает встроенную поддержку виртуальных окружений в виде модуля venv и команды `pyvenv`. Команда `pyvenv` может использоваться вместо утилиты `virtualenv`, но имейте в виду, что виртуальные окружения, созданные с помощью `pyvenv` из установки Python 3.3, не включают утилиту `pip`, которую необходимо установить вручную. Это ограничение устранено в Python 3.4, где `pyvenv` можно использовать как полноценную замену `virtualenv`.

Большинство дистрибутивов Linux позволяет установить `virtualenv` в виде отдельного пакета. Например, пользователи Ubuntu могут установить эту утилиту командой:

```
$ sudo apt-get install python-virtualenv
```

В Mac OS X ее можно установить командой `easy_install`:

```
$ sudo easy_install virtualenv
```

Пользователям Microsoft Windows и других операционных систем, официально не поддерживающих пакета `virtualenv`, может потребоваться пройти более сложную процедуру установки.

Откройте в браузере адрес **https://bitbucket.org/pypa/setuptools** – домашнюю страницу дистрибутива `setuptools`. На этой странице най-

дите ссылку для загрузки сценария установки. Этот сценарий имеет имя `ez_setup.py`. Сохраните файл сценария во временной папке на своем компьютере, затем выполните следующие команды в этой папке:

```
$ python ez_setup.py
$ easy_install virtualenv
```

Предыдущие команды должны выполняться с привилегиями администратора. Для этого в Microsoft Windows запустите командную строку, выбрав пункт **Run as Administrator** (Запуск от имени администратора). В Unix-подобных системах этим двум командам должна предшествовать команда `sudo`, или они должны вызываться с привилегиями пользователя root. После установки утилиту `virtualenv` можно вызывать с привилегиями обычного пользователя.

Теперь нужно создать папку, где будут храниться файлы с исходным кодом примеров, загруженные из репозитория на сайте GitHub. Как говорилось в разделе «Как работать с примерами программного кода» выше, проще всего загрузить файлы, извлекая их непосредственно из репозитория с помощью клиента Git. Следующие команды загрузят примеры из GitHub и инициализируют папку для версии «1a» – начальной версии приложения:

```
$ git clone https://github.com/miguelgrinberg/flasky.git
$ cd flasky
$ git checkout 1a
```

Следующий шаг – создание виртуального окружения Python внутри папки *flasky* с использованием команды `virtualenv`. Эта команда имеет единственный обязательный аргумент: имя виртуального окружения. Она создаст в текущем каталоге папку с выбранным именем и скопирует в нее все файлы, необходимые для образования виртуального окружения. Часто виртуальному окружению присваивается имя *venv*:

```
$ virtualenv venv
New python executable in venv/bin/python2.7
Also creating executable in venv/bin/python
Installing setuptools.............done.
Installing pip...............done.
```

Теперь внутри папки *flasky* у вас имеется папка *venv* с совершенно новым виртуальным окружением, содержащим собственный интерпретатор Python. Чтобы начать использовать виртуальное окружение, его необходимо «активировать». Пользующиеся командной оболочкой bash (это относится к пользователям Linux и Mac OS X) могут активировать виртуальное окружение командой:

```
$ source venv/bin/activate
```

Пользователи Microsoft Windows могут выполнить команду:

```
$ venv\Scripts\activate
```

После активации виртуального окружения каталог с копией интерпретатора Python будет добавлен в переменную окружения `PATH`, но это изменение носит временный характер; оно продолжает действовать только в текущем сеансе командной оболочки. Для напоминания, что вы активировали виртуальное окружение, команда активации изменяет строку приглашения к вводу в командной оболочке, включая в нее имя окружения:

```
(venv) $
```

По окончании работы с виртуальным окружением, чтобы вернуться к использованию глобального интерпретатора Python, введите команду `deactivate`.

Установка пакетов Python с помощью pip

Большинство пакетов Python устанавливается с помощью утилиты `pip`, которую команда `virtualenv` автоматически добавляет во все создаваемые виртуальные окружения. В момент активации виртуального окружения местоположение утилиты `pip` автоматически добавляется в переменную окружения `PATH`.

Если виртуальное окружение было создано с помощью команды `pyvenv`, входящей в состав Python 3.3, утилиту `pip` придется установить вручную. Инструкции по установке доступны на веб-сайте **http://bit.ly/pip-install**. В версии Python 3.4 команда `pyvenv` устанавливает `pip` автоматически.

Чтобы установить Flask в виртуальное окружение, выполните команду:

```
(venv) $ pip install flask
```

Она установит фреймворк Flask и все его зависимости в виртуальное окружение. Чтобы убедиться в успешности установки, запустите интерпретатор Python и попробуйте импортировать фреймворк:

```
(venv) $ python
>>> import flask
>>>
```

Если на экране не появится сообщение об ошибке, можете поздравить себя: вы готовы перейти к следующей главе, где вы напишете свое первое веб-приложение.

Глава 2

Структура простого приложения

В этой главе вы познакомитесь с разными частями приложений на основе Flask, а также напишете и запустите свое первое веб-приложение.

Инициализация

Любое приложение на основе Flask должно создать *экземпляр приложения*. Веб-сервер будет передавать все запросы, принимаемые от клиентов, этому объекту через протокол, который называется Web Server Gateway Interface (WSGI). Экземпляр приложения – это объект класса `Flask`, который обычно создается так:

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

Единственным обязательным аргументом конструктора класса `Flask` является имя главного модуля или пакета приложения. Для большинства приложений на Python это имя хранится в переменной `__name__`.

Аргумент `name`, который передается конструктору `Flask` приложения, часто является источником недопонимания для начинающих разработчиков. Фреймворк Flask использует этот аргумент для определения пути к корневому каталогу приложения, чтобы позднее с его помощью находить файлы ресурсов.

Далее мы увидим более сложные примеры инициализации, но для простых приложений этого вполне достаточно.

Маршруты и функции представлений

Клиенты, такие как веб-браузеры, отправляют *запросы* веб-серверу, который, в свою очередь, отправляет их экземпляру приложения на основе Flask. Экземпляр приложения должен определить, какой код

должен быть выполнен для обработки обращения к адресу URL в запросе, поэтому он должен хранить отображение адресов URL в функции на языке Python. Ассоциацию между адресом URL и функцией называют *маршрутом* (*route*).

Проще всего определить маршрут в приложении на основе Flask с помощью декоратора `app.route`, экспортируемого экземпляром приложения. Этот декоратор регистрирует декорируемую функцию как маршрут. Ниже показан пример объявления маршрута с помощью декоратора:

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

Декораторы – это стандартная особенность языка Python, они способны изменять поведение функций. Часто декораторы используются для регистрации функций в качестве обработчиков событий.

В предыдущем примере функция `index()` регистрируется как обработчик запросов к корневому адресу URL приложения. Если бы это приложение было развернуто на сервере с доменным именем *www.example.com*, тогда попытка открыть страницу с адресом *http://www.example.com* в браузере привела бы к вызову `index()` на сервере. Возвращаемое значение этой функции называется *ответом* (*response*) – это то, что получит клиент. Если клиентом является веб-браузер, ответ будет интерпретироваться как документ, который требуется отобразить на экране.

Функции, такие как `index()`, называют *функциями представления* (*view functions*). Ответ, возвращаемый функцией представления, может быть простой строкой с разметкой HTML или иметь более сложную форму, как будет показано позднее.

Встраивание строк ответов непосредственно в программный код на Python усложняет его сопровождение, и в данном случае это было сделано, только чтобы познакомить вас с понятием ответов. Правильный способ создания ответов будет представлен в главе 3.

Если вы внимательно рассмотрите структуру некоторых URL-служб, которыми пользуетесь ежедневно, вы заметите, что многие из них имеют переменные разделы. Например, URL к странице профиля на Facebook имеет вид: **http://www.facebook.com/<имя-пользователя>**, то есть имя пользователя является частью URL. Flask поддерживает подобные адреса URL с помощью специального

синтаксиса в декораторе маршрута. Например, ниже определяется маршрут, включающий переменный компонент `name`:

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

Фрагмент в угловых скобках – это переменная часть, то есть любой URL, совпадающий с постоянной частью, будет отображен в этот маршрут. При вызове функции представления Flask передаст ей динамический компонент в виде аргумента. В предыдущем примере аргумент `name` используется функцией представления для создания персонализированного ответа.

Переменные элементы в маршрутах по умолчанию интерпретируются как строки, но могут также иметь другие типы. Например, маршрут `/user/<int:id>` будет соответствовать адресам URL, содержащим целое число в переменном сегменте `id`. Flask поддерживает в маршрутах типы `int`, `float` и `path`. Тип `path` также является строковым, но при его интерпретации символы косой черты не рассматриваются как разделители и включаются в переменный компонент.

Запуск сервера

Экземпляр приложения имеет метод `run`, который запускает интегрированный веб-сервер, используемый для разработки:

```
if __name__ == '__main__':
    app.run(debug=True)
```

Идиома `__name__ == '__main__'`, широко используемая в языке Python, гарантирует, что веб-сервер для разработки будет запускаться только при непосредственном выполнении сценария. Когда сценарий импортируется другим сценарием, предполагается, что родительский сценарий запустит другой сервер, поэтому вызов `app.run()` пропускается.

После запуска сервер входит в цикл ожидания и обработки запросов. Этот цикл продолжается, пока приложение не будет остановлено, например нажатием комбинации **Ctrl-C**.

Метод `app.run()` имеет несколько необязательных аргументов для настройки режима работы веб-сервера. В процессе разработки довольно удобно бывает включить режим отладки, в котором, кроме всего прочего, активируются *отладчик* и *перезагрузчик*. Для этого следует передать в аргументе `debug` значение `True`.

Веб-сервер, входящий в состав фреймворка Flask, не предназначен для использования в промышленном окружении. С настоящими, промышленными веб-серверами мы познакомимся в главе 17.

Законченное приложение

В предыдущих разделах мы познакомились с разными частями веб-приложения на основе Flask, и теперь пришло время написать такое приложение. Весь сценарий *hello.py* приложения состоит точно из трех частей, описанных выше и объединенных в один файл. Исходный код приложения приводится в примере 2.1.

Пример 2.1 ❖ hello.py: законченное приложение на основе фреймворка Flask

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

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

if __name__ == '__main__':
    app.run(debug=True)
```

Если у вас уже есть копия репозитория с исходными текстами приложений с сайта GitHub, вы можете выполнить команду `git checkout 2a` и получить эту версию приложения.

Чтобы запустить приложение, активируйте виртуальное окружение, созданное ранее. Убедитесь, что фреймворк Flask установлен. Запустите приложение командой:

```
(venv) $ python hello.py
* Running on http://127.0.0.1:5000/
* Restarting with reloader
```

Откройте веб-браузер и введите адрес **http://127.0.0.1:5000/** в адресной строке. На рис. 2.1 показано, как выглядит окно браузера после соединения с приложением.

Если ввести любой другой адрес URL, приложение не будет знать, как его обработать, и вернет код ошибки 404 – широко известный код, который возвращается при попытке перейти к несуществующей странице.

В примере 2.2 приводится расширенная версия приложения, в которой добавлен второй маршрут с переменной частью. При обращении к этому адресу URL приложение выведет персонализированное приветствие.

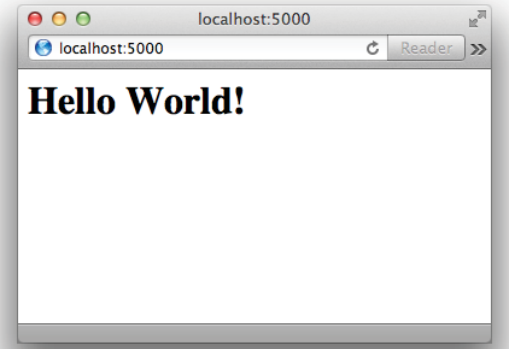

Рис. 2.1 ❖ Flask-приложение hello.py

Пример 2.2 ❖ hello.py: Flask-приложение с динамическим маршрутом

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

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

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

if __name__ == '__main__':
    app.run(debug=True)
```

Если у вас уже есть копия репозитория с исходными текстами приложений с сайта GitHub, вы можете выполнить команду `git checkout 2b` и получить эту версию приложения.

Чтобы протестировать работу динамического маршрута, запустите приложение и откройте в браузере страницу с адресом: **http://localhost:5000/user/Dave**. Приложение ответит приветствием, включив в него переменную часть маршрута `name`. Попробуйте подставлять разные имена, чтобы убедиться, что функция представления всегда возвращает ответ, содержащий указанное вами имя. Пример показан на рис. 2.2.

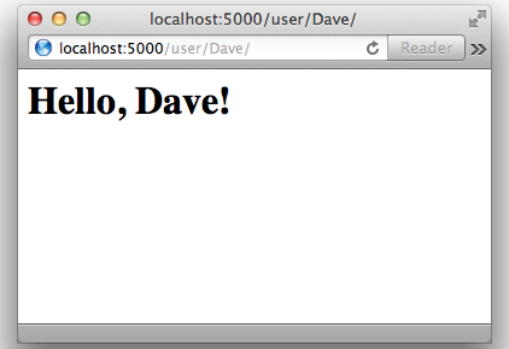

Рис. 2.2 ❖ Динамический маршрут

Цикл запрос–ответ

Теперь, поэкспериментировав с простым приложением на основе Flask, у вас может появиться желание узнать больше о том, как работает Flask. В следующих разделах описываются некоторые архитектурные аспекты фреймворка.

Контексты приложения и запроса

Когда фреймворк Flask принимает запрос от клиента, он должен обеспечить доступ к нескольким объектам из функции представления, которая будет обрабатывать запрос. Хорошим примером может служить *объект запроса*, содержащий HTTP-запрос, отправленный клиентом.

Очевидный способ обеспечить доступ к объекту запроса – передать его в виде аргумента, но для этого потребовалось бы, чтобы каждая функция представления в приложении принимала дополнительный аргумент. Ситуация становится еще более сложной, если принять во внимание, что объект запроса – не единственный объект, доступ к которому должна иметь функция представления, чтобы суметь обработать запрос.

Чтобы избежать захламления функций представления большим числом параметров, которые могут потребоваться или нет, и времен-