Leif Azzopardi & David Maxwell 著 / 安道 译

Django 基础教程

Tango with Django

样章

兼容 django 1.9/1.10/1.11

www.tangowithdjango-china.com

Django 基础教程

Tango with Django

Leif Azzopardi & David Maxwell 著

安道 译

rev 0.0.2, 2018-05-23T20:09:04+08:00

目录

本书是《Tango with Django》的简体中文版，由 Leif Azzopardi 和 David Maxwell 授权翻译。

1 导言

这一本学做结合的指南，旨在教你使用 Django 和 Python 做 Web 开发。本书主要针对学生，因此会详解使用 Django 开发 Web 应用过程中的每个步骤。

Django 官方提供了一份教程，而且网上也有很多优秀的教程，本书的目标是填补一些空白，通过实例开发学习 Django 框架。此外，本书还会介绍开发 Web 应用所需掌握的其他知识，例如 HTML、CSS、JavaScript 等等。

1.1 本书特色

❑ 事半功倍

笔者见过很多聪明的学生陷入僵局，浪费几小时的时间尝试解决遇到的 Django 或其他 Web 开发问题。这些问题往往是由于抓不住重点，或者所用的材料言语不详。有时，你可能灵光一现，在十几分钟之内解决问题，但是更多的时候要耗费几小时。笔者结合自己的经验，力求消除这些障碍，让你远离坎坷，在开发应用的过程中一帆风顺。

❑ 平缓学习曲线

Web 应用框架省时省力，但前提是你知道怎么使用框架。框架的学习曲线往往陡峭。本书力求平缓学习曲线，详解方方面面，让你快速掌握框架的用法。

❑ 改进工作流程

使用 Web 应用框架要遵守特定的设计模式，你只需在特定的位置放置特定的代码。但是，与很多学生交流之后，笔者发现他们经常抱怨，不知如何从框架那里夺回控制权（即控制反转）。为此，笔者制定了几个工作流程，帮你在开发过程中重获控制权，以自己的方式构建 Web 应用。

❑ 边学边做

无论如何，不要只是看看内容而已。这是一本实作指南，你要自己动手使用 Django 构建 Web 应用。动眼不动手可不行！若想有所获益，请跟着本书一起开发应用。而且，在开发的过程中，不

要复制粘贴书中的代码。自己动手输入，想想代码的作用，然后再阅读书中给出的说明。如果依旧不解，查阅 Django 文档，到 Stack Overflow 或其他网站中寻求帮助，一定要把不理解的地方弄明白。如果你确实陷入僵局了，可以联系笔者，以便笔者改进内容——已经有多位读者为本书做出了贡献。

1.2 你将学到

本书以一个示例为主线，说明如何开发一个名为 Rango 的 Web 应用。在这个过程中将讲解如何完成下述关键任务：

- **搭建开发环境**，包含学习使用终端、虚拟环境、pip 安装程序和 Git，等等。
- **创建 Django 项目**和简单的 Django 应用。
- **配置 Django 项目**，伺服静态媒体和其他媒体文件。
- 使用 Django 的“**模型-视图-模板**”设计模式。
- **创建数据库模型**，使用 Django 提供的*对象关系映射*（Object Relational Mapping，ORM）功能。
- **创建表单**，利用数据库模型生成动态网页。
- 使用 Django 提供的**用户身份验证**服务。
- 在 Django 应用中融合**外部服务**。
- 在 Web 应用中引入*层叠样式表*（Cascading Styling Sheet，CSS）和 JavaScript。
- 使用 CSS 为应用提供专业的外观。
- 在 Django 应用中处理 **cookie 和会话**。
- 在应用中使用 Ajax 等高级技术。
- 把应用**部署**到 PythonAnywhere 上。

每一章结尾都有几道练习题，旨在加强你对知识的掌握，也检验你能不能学以致用。后面几章还有开放性开发练习，而且提供了参考代码和说明。

</> 这样的区域是练习

每一章都有练习题，旨在检查你对知识和技能的掌握情况。你必须解答这些练习，因为后面的章节建立在这些题目之上。

别担心自己做不出来，本书的 GitHub 仓库中有所有练习题的解答。

1.3 用到的技术和服务

本书将使用的技术和外部服务如下：

- Python 编程语言
- pip 包管理器
- Django 框架
- Git 版本控制器系统
- GitHub
- HTML
- CSS
- JavaScript 编程语言
- jQuery 库
- Twitter Bootstrap 框架
- Webhose API（即后文所用的搜索 API）
- PythonAnywhere 托管服务

这些技术和服务是笔者精选的，其中某些是 Web 开发的基础。Twitter Bootstrap 为 Web 应用提供样式，Webhose API 是一个外部服务，PythonAnywhere 则能简化部署应用的过程。

1.4 Rango 的初步设计和客户要求

本书的主线是开发一个名为 Rango 的应用。在这个过程中，我们将涵盖构建 Web 应用所需掌握的重要知识。如果想查看这个应用的最终版本，请访问 *http://rangodemo.pythonanywhere.com/rango/*。

设计概要

客户要求你开发一个名为 Rango 的网站，让用户按分类浏览不同的网页。在西班牙语中，“rango”的意思是“等级”或“显要地位”。

- 用户访问 Rango 网站的首页时，客户想让访客看到：

    - 访问次数最多的 5 个网页
    - 查看次数最多的 5 个分类
    - 浏览或搜索分类的不同方式

- 用户访问分类页面时，客户想让 Rango 显示：
    - 分类的名称、查看次数、点赞次数，以及分类下的网页（显示网页的标题，并链接到网页的 URL）
    - 一定的搜索功能（通过搜索 API 实现），找出可以归入当前分类的网页
- 客户要求记录分类的名称，分类页面被查看的次数，以及多少用户点击了“点赞”按钮（即用户对分类打分，投票排名）。
- 各分类能通过友好的 URL 访问，例如 */rango/books-about-django/*。
- 只有注册用户能搜索，能把网页添加到分类中。因此要让网站的访客能注册账户。

这么一看，我们要开发的应用并不复杂，不过是列出一些网页的分类而已。然而，这其中有些复杂问题需要解决，可能并不像想的那样简单。着手开发之前，我们先规划一下整体设计。

</> 练习

继续阅读之前，请根据客户要求绘制下述设计图：

- N 层或系统架构图
- 主页和分类页面的线框图
- 应用的 URL 映射
- 实体关系图（Entity-Relationship diagram，ER 图），描述要实现的数据模式

在继续阅读之前，请试着完成这几题。即使你不熟悉系统架构图、线框图或 ER 图也没关系，这几题的重点是让你思考如何说明和描述即将构建的应用。

N 层架构

多数 Web 应用的整体架构是一种 3 层结构。Rango 也采用这种架构，不过稍有不同，因为它还要

与外部服务交互。

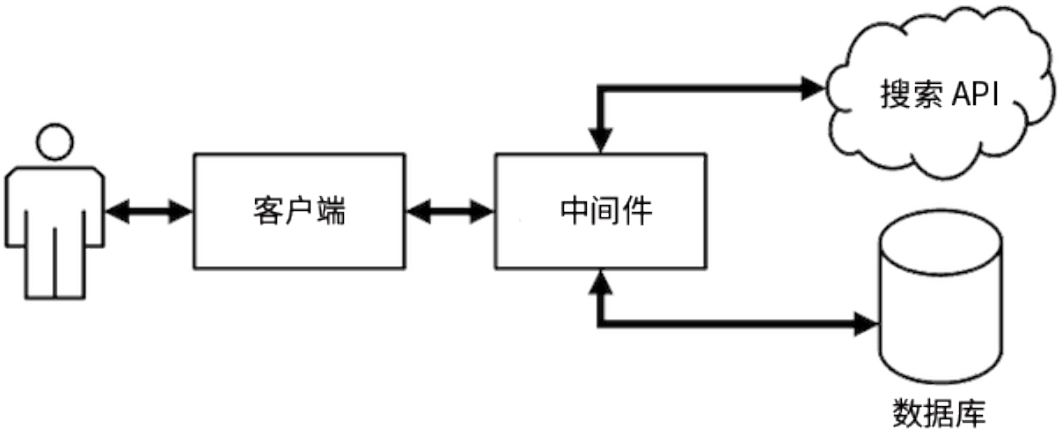

图 1-1：Rango 的 3 层系统架构

因为我们将使用 Django 构建这个 Web 应用，所以各层用到的技术如下：

- 客户端是 Web 浏览器（例如 Chrome、Firefox 或 Safari），负责渲染 HTML/CSS 页面。
- 中间件是一个 Django 应用程序，在开发过程中由 Django 内置的开发 Web 服务器负责调度。
- 数据库使用基于 Python 的 SQLite3 数据库引擎。
- 搜索 API 使用 Webhose API。

本书基本上将集中精力开发中间件，不过从图 1-1 中不难看出，我们也将与其他组件交互。

线框图

线框图可以让客户一览最终完成的应用是什么样子。线框图能节省大量时间，不过具体形式各种各样，有手绘的，也有使用专用工具的。Rango 应用首页的设计稿见图 1-2，分类页面的设计稿见图 1-3。

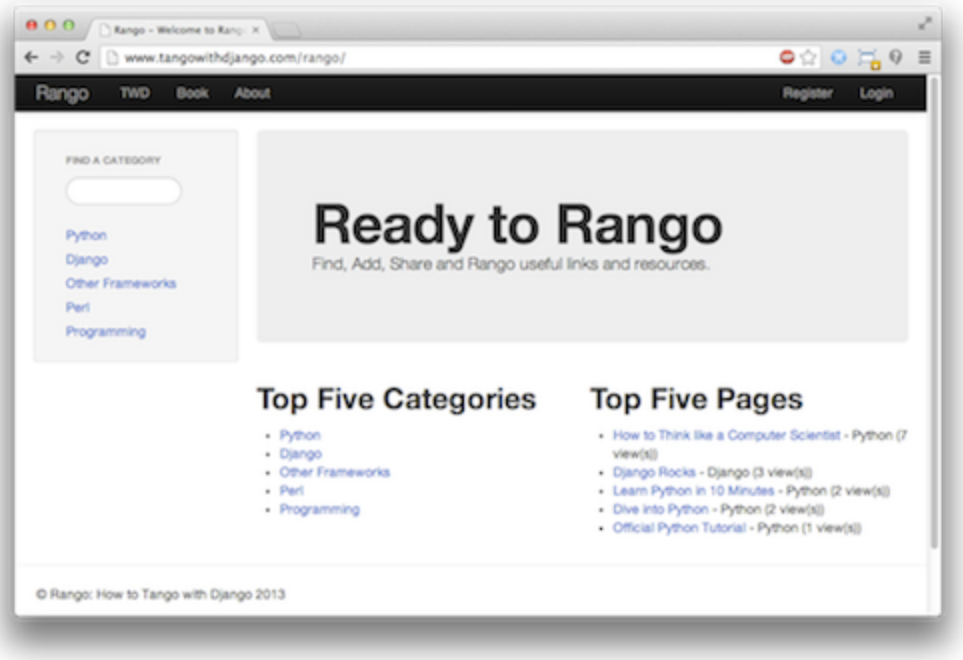

图 1-2：首页，左边是分类搜索框，右边是查看次数最多的 5 个分类和 5 个网页

页面和 URL 映射

从客户的要求中我们知道，应用至少要有两个页面。为了能访问各个页面，我们要描述 URL 映射。你可以把 URL 映射理解为访问页面时在浏览器地址栏中输入的文本。Rango 应用的基本 URL 映射如下：

- / 或 */rango/* 指向主页
- */rango/about/* 指向关于页面
- */rango/category/<category_name>/* 指向名为 `<category_name>`（例如 games、python-recipes 或 code-and-compilers）的分类页面

这只是开始，在构建应用的过程中，可能还要添加其他 URL 映射。本书将带领你使用 Django 框架和“模型-视图-模板”设计模式逐渐丰富这些页面的内容。对 URL 映射和页面的设计有了大致了解之后，我们要定义数据模型，存储应用将用到的数据。

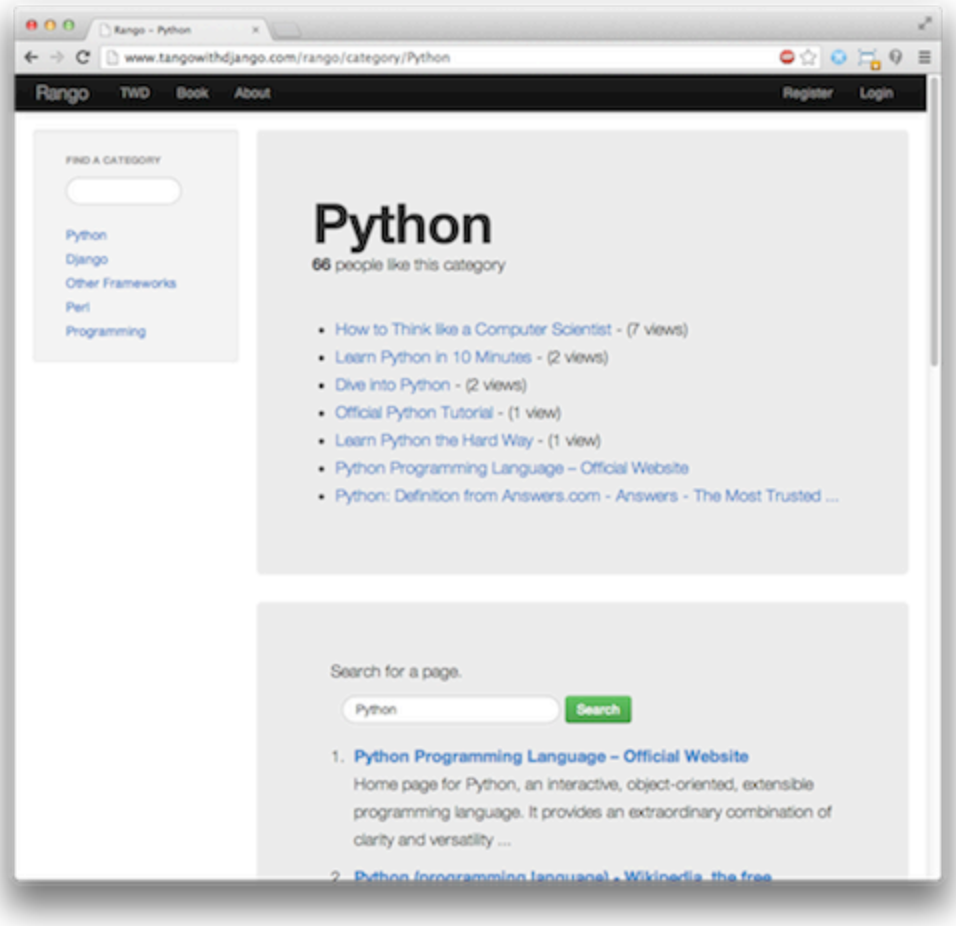

图 1-3：分类页面，显示分类中的网页（带有访问次数），以及搜索“Python”得到的结果

实体关系图

根据客户的要求，很明显我们至少需要两个实体：分类（category）和网页（page）。而且，一个分类中可以有多个网页。这个简单的数据模型可以通过下述 ER 图描述。

注意，这个关系并不十分明确。理论上，一个网页可以归入多个分类。鉴于此，分类和网页之间可以通过多对多关系建模。但是这样处理太过复杂，因此我们将假定一个分类中有多个网页，而一个网页只能属于一个分类。这样并不妨碍把一个网页归入不同的分类，只不过要多次输入相同的网页，不是太理想而已。

图 1-4：Rango 应用中两个主要实体的 ER 图

■ 做好记录 ■

要养成习惯，把所做的假设记录下来，例如这里假设的一对多关系，以防以后出问题。有了记录，你便可以与开发团队沟通，确保他们考虑到了你的假设。

根据这一假设，我们可以通过表格列出各实体的详细内容，指明各实体中有哪些字段。我们使用 Django 的 `ModelField` 类型定义各字段的类型（即 `IntegerField`、`CharField`、`URLField` 或 `ForeignKey`）。注意，在 Django 中主键（primary key）是隐含的，Django 会为各模型添加 `id` 字段（详情参见第 5 章）。

表 1-1：Category 模型

字段	类型
`name`	`CharField`
`views`	`IntegerField`
`likes`	`IntegerField`

表 1-2：Page 模型

字段	类型
`category`	`ForeignKey`
`title`	`CharField`
`url`	`URLField`
`views`	`IntegerField`

此外，还需要一个 `User` 模型，以便用户注册和登录。这里没有给出 `User` 模型，讨论用户身份验证时再介绍。后面的章节将说明如何在 Django 中定义这些模型，以及如何使用内置的 ORM 连接数据库。

1.5 小结

这些整体上的设计和要求将在构建 Web 应用的过程中为我们提供参考。虽然我们将集中精力说明如何使用特定的技术，但是多数数据库驱动的网站都是这么规划的。因此，你最好能熟练地阅读和制定这样的要求和设计，以便与人顺畅沟通。本书将使用 Django 和相关的技术实现这里制定的

要求。

◆ 复制粘贴代码 ◆

阅读本书的过程中，你可能会把书中的代码复制粘贴到代码编辑器中。然而，笔者建议自己动手输入代码。我们知道自己输入稍显麻烦，但是这样有助于你理解整个过程，还能让你记住以后用得到的命令。

此外，复制粘贴 Python 代码简直就是自找麻烦。复制的空白可能变成空格、制表符或二者混杂。这样可能导致各种奇怪的问题，而且不一定是由缩进引起的。如果你确实想复制粘贴代码，一定要留神这样的问题。如果你使用的是 Python 3，对这样的问题要格外小心，因为混用制表符和空格缩进会导致 `TabError`。

多数代码编辑器都能显示空白，而且会区分制表符和空格。如果你使用的编辑器有这样的功能，一定要启用，免得分不清。

2 前期准备工作

开始编程之前，我们要搭建好 Django 所需的开发环境。我们要在自己的电脑中安装所需的全部组件。本章概述要安装和使用的 5 个关键组件。

- 终端或命令提示符
- Python
- Python 包管理器和虚拟环境
- 集成开发环境（如果你想用的话）
- 版本控制系统 Git

如果你的电脑中已经安装了 Python 2.7/3.4/3.5 和 Django 1.9/1.10，而且熟悉前述工具，可以直接跳到第 3 章。否则，请阅读下面对这些组件的介绍，以及它们在开发过程中的重要性。我们还将指出这些组件的安装方式。

★ 开发环境 ★

搭建开发环境是个繁琐的过程，很容易出错，但不是每天都要做。本章介绍这其中涉及的关键技术，以及如何安装各个组件。

根据笔者的经验，建议你把搭建的步骤记录下来。说不定有一天你会用到，例如你买了新电脑，或者帮助别人。现在做的记录能为以后节省时间和精力。不要只看眼前！

2.1 Python

本书要求你在自己的电脑中安装 Python 编程语言，2.7 系列（至少 2.7.5）或 3.4 版本以上都行。

如果你不知道如何安装 Python，需要帮助，请阅读 A.1 节。

★ 不会用 Python? ★

如果你没用过 Python，或者想提升自己的技能，笔者推荐下面几个材料：

- Stavros 写的 Python 10 分钟速成教程
- Python 官方教程
- Allen B. Downey 写的《像计算机科学家一样思考 Python》
- Jennifer Campbell 和 Paul Gries 开设的“学习编程”课程

这几个材料能让你熟悉 Python 的基础知识，以便开始使用 Django 开发应用。注意，为了使用 Django，你无需变身专家。Python 是门优秀的语言，如果你会用其他编程语言，可以边用边学。

2.2 Python 包管理器

pip 是 Python 的包管理器。你可以使用 pip 安装各种库，增强 Python 的功能。

包管理器，不管是针对 Python 的、针对操作系统的，还是针对其他环境的，是一种自动安装、升级、配置和删除包的软件，解放了你的双手，无需你自己动手下载、安装和维护软件。Python 包管理起来可不简单。多数包通常有依赖（dependency），要随包一起安装。因此，包之间可能有冲突，需要依赖特定的版本。此外，包的系统路径也要指定和维护。幸好，这些繁琐的工作都由 pip 代为处理了，解放了我们的生产力。

使用 `pip` 命令运行 pip 试试看。如果提示找不到 `pip` 命令，说明你要安装 pip。详情参见附录 A。你还要确保自己的系统中安装了下面两个包。执行下述命令，安装 Django 和 Pillow（处理图像的 Python 库）：

```
$ pip install -U django==1.9.10
$ pip install pillow
```

■ 无法成功安装 Pillow? ■

安装 Pillow 时可能会报错，提示缺少 JPEG 支持，如下所示：

```
ValueError: jpeg is required unless explicitly disabled using
            --disable-jpeg, aborting
```

如果你遇到这个问题，禁用 JPEG 支持试试：

```
$ pip install pillow --global-option="build_ext"
                     --global-option="--disable-jpeg"
```

这样虽然无法处理 JPEG 图像，但是却能成功安装 Pillow。本书只要求能安装 Pillow 就行了。详情参见 Pillow 文档。

2.3 虚拟环境

环境就快搭建好了。但是在继续之前要注意一点，目前的环境虽然可以使用了，但是有些缺点。如果另一个应用要使用 Python 的其他版本才能运行怎么办？如果你想把 Django 换成最新版，但是依然想维护使用 Django 1.9 开发的项目又怎么办？？

答案是使用虚拟环境。借助虚拟环境，我们可以让不同的 Python 版本和同一个包的不同版本和谐相处。如今，这是人们普遍使用的 Python 环境搭建方式。

虽然不必须使用虚拟环境，但是强烈建议你使用。搭建、创建和使用虚拟环境的详细说明参见 A.4 节。

2.4 集成开发环境

集成开发环境（Integrated Development Environment，IDE）虽然不是必须的，但一个支持 Python 的好 IDE 能为开发工作提供极大的帮助。支持 Python 的 IDE 很多，JetBrains 出品的 PyCharm 和 PyDev（Eclipse 的插件）或许是其中最受欢迎的。Python Wiki 中有支持 Python 的 IDE 的最新列表。

自己研究一下，找出合用的。注意，有些 IDE 需要购买许可证。如果你选择的 IDE 能集成 Django，那就更好了。

笔者使用的是 PyCharm，因为它支持虚拟环境，而且能集成 Django（不过需要配置）。这里不讨论如何集成 Django，如果你想知道，可以阅读 JetBrains 网站中介绍 PyCharm 集成 Django 的文章。

2.5 代码仓库

最后，开发过程中应该把代码纳入版本控制系统，例如 SVN 或 Git。为了不脱离主线，这里不说明如何使用版本控制器系统。若想快速掌握 Git，推荐阅读《GitHub入门与实践》一书。强烈建议你为自己的项目创建一个 Git 仓库。

</> 练习

为了熟悉搭建环境的步骤，请尝试完成以下练习：

- ❑ 安装 Python 2.7.5+/3.4+ 和 pip
- ❑ 熟悉命令行界面，创建一个名为 *workspace* 的目录，我们创建的项目将保存在这里
- ❑ 搭建虚拟环境（选做）
- ❑ 安装 Django 和 Pillow
- ❑ 如果没有 Git 仓库托管网站（例如 GitHub、BitBucket，等等）的账号，注册一个
- ❑ 下载并设置一个 IDE，例如 PyCharm

前面说过，本书的内容和示例应用在一个 GitHub 仓库中。

- ❑ 如果你发现错误或其他问题，请在 GitHub 中提出，让我们知道
- ❑ 如果碰到做不出的练习，可以参照仓库中的代码

★ 目录是什么？ ★

前面的练习中有一题让你创建一个目录（directory），可目录究竟是什么呢？如果你一直使用 Windows 系统，目录就是文件夹（folder）。二者在概念上是一样的，都是一种目录结构，列出里面的文件和目录。

3 Django 基础

下面开始学习 Django。本章介绍如何新建项目和 Web 应用，最终让一个 Django 驱动的简单网站运行起来。

3.1 检查环境

首先，我们要检查有没有正确安装 Python 和 Django。请打开一个新终端窗口，执行下述命令，查看 Python 的版本。

```
$ python --version
```

得到的结果应该是 2.7.11 或 3.5.1，不过 2.7.5+ 或 3.4+ 版的 Python 也可以。如果需要安装或升级 Python，请翻到附录 A。

如果你想使用虚拟环境，记得激活。倘若不记得怎么操作了，请翻到 A.4 节。

确认 Python 正确安装之后，还要检查 Django。在终端窗口中执行下述命令，运行 Python 解释器。

```
$ python
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

在提示符后输入下述命令：

```
>>> import django
>>> django.get_version()
'1.9.10'
```

```
>>> exit()
```

如果一切正常，应该能看到 Django 的版本号。然后输入 `exit()`，退出 Python 解释器。如果无法导入 Django，确认你是不是在虚拟环境中， 再使用 `pip list` 命令查看安装的包。

如果不知如何安装包，或者安装的版本不对，请翻到附录 A，或者阅读 Django 文档中的安装说明。

■ 提示符 ■

本书代码片段中有两种符号要注意。

以美元符号（`$`）开头的代码片段，其后的内容是要在终端或命令提示符中运行的命令。

`>>>` 表示后面的命令要输入交互式 Python 解释器。解释器打开的方法是执行 `python` 命令，退出方法是输入 `exit()`。

3.2 创建 Django 项目

进入 *workspace* 目录，执行下述命令，新建一个 Django 项目：

```
$ django-admin.py startproject tango_with_django_project
```

如果你的电脑中没有 *workspace* 目录，那就创建一个，把 Django 项目和其他项目都保存在那里。我们将在代码中使用 `<workspace>` 指代你的 *workspace* 目录。你要把 `<workspace>` 替换成 *workspace* 目录的具体路径，例如 */Users/leifos/Code/* 或 */Users/maxwelld90/Workspace/*。

★ 找不到 `django-admin.py`? ★

输入 `django-admin` 试试。根据采用的安装方式，有些系统可能无法识别 `django-admin.py`。

根据 Stack Overflow 中这个问答的建议，在 Windows 系统中可能要使用 *django-admin.py* 脚本的完整路径，例如：

```
python c:\python27\scripts\django-admin.py
       startproject tango_with_django_project
```

这个命令调用 *django-admin.py* 脚本，新建一个名为 *tango_with_django_project* 的 Django 项目。通常，笔者喜欢在 Django 项目所在的目录名后面加上 `_project`，明确表明目录中是什么。不过，具体怎么命名完全由你自己决定。

此时你会发现，你的 *workspace* 目录中出现了与项目同名的一个目录，即 *tango_with_django_project*。在这个目录中你会看到两个内容：

- 另一个与项目同名的目录
- 一个 Python 脚本，名为 *manage.py*

在本书中，我们将把内部那个 *tango_with_django_project* 目录称为项目配置目录。在这个目录中，你会看到 4 个 Python 脚本，下面简单介绍一下，后文再详细说明：

- *__init__.py*：一个空 Python 脚本，存在的目的是告诉 Python 解释器，这个目录是一个 Python 包；
- *settings.py*：存放 Django 项目的所有设置；
- *urls.py*：存放项目的 URL 模式；
- *wsgi.py*：用于运行开发服务器和把项目部署到生产环境的一个 Python 脚本。

项目目录中有个名为 *manage.py* 的文件，在开发过程中时常用到。它提供了一系列维护 Django 项目的命令，例如通过它可以运行内置的 Django 开发服务器，可以测试应用，还可以运行多个数据库命令。几乎每个 Django 命令都要调用这个脚本。

★ Django 管理脚本 ★

Django 管理脚本的详细说明参见 Django 文档。

执行 `python manage.py help` 命令可以查看可用命令列表。

你现在就可以使用 *manage.py* 脚本，执行下述命令试试：

```
$ python manage.py runserver
```

这个命令启动 Python，让 Django 运行内置的轻量级开发服务器。你在终端窗口中应该会看到类似下面的输出：

```
$ python manage.py runserver
```

```
Performing system checks...

System check identified no issues (0 silenced).

You have unapplied migrations; your app may
not work properly until they are applied.

Run 'python manage.py migrate' to apply them.

October 2, 2016 - 21:45:32
Django version 1.9.10, using settings 'tango_with_django_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

从这段输出可以看出几件事。首先，没有出现阻碍应用运行的问题。但是，输出中有个提醒，指出有未应用的迁移。这个问题在设置数据库时再讨论，现在暂且忽略。最后，尤为重要的是一个 URL 地址，即 *http://127.0.0.1:8000/*，这是 Django 开发服务器的地址。

打开 Web 浏览器，输入 URL *http://127.0.0.1:8000/*。你将看到类似图 3-1 的网页。

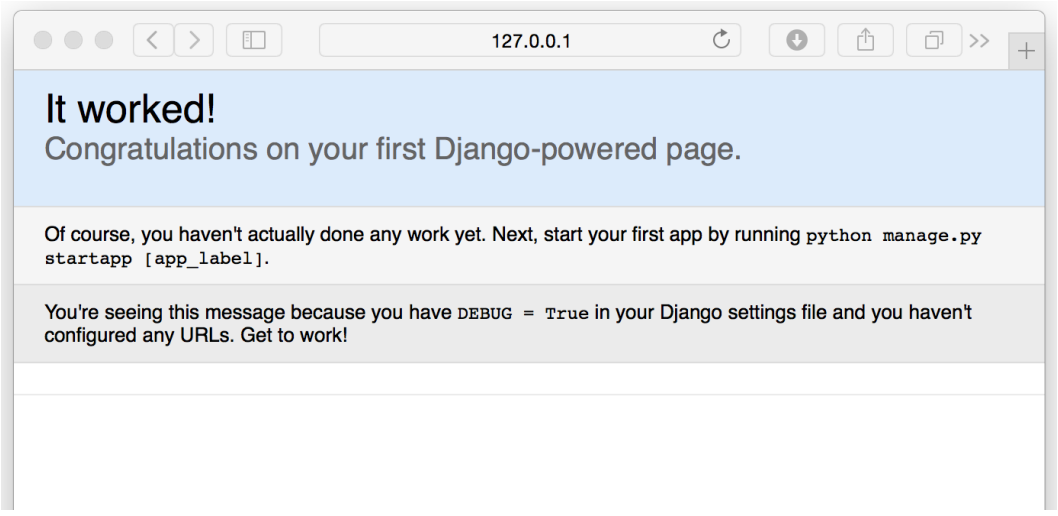

图 3-1：首次运行 Django 开发服务器时看到的页面

开发服务器随时可以停止，只需在终端或命令提示符窗口中按 CTRL+C 键。如果想在其他端口上运行开发服务器，或者允许其他设备访问，可以提供可选的参数。例如下述命令：

```
$ python manage.py runserver <your_machines_ip_address>:5555
```

这个命令强制开发服务器在 TCP 端口 5555 上响应入站请求。记得把 <your_machines_ip_address> 换成你电脑的 IP 地址或 127.0.0.1。

★ 不知道自己的 IP 地址？ ★

使用 0.0.0.0，Django 能找出你的 IP 地址。不信可以试试：

```
$ python manage.py runserver 0.0.0.0:5555
```

设置端口时，不要使用 80 或 8080，这一般是为 HTTP 保留的。此外，低于 1024 的端口是操作系统专属的。[1]

虽然部署应用时不会使用这个轻量级的开发服务器，但是能让同一网络中的其他设备访问你的应用还是有必要的。使用你的设备的 IP 地址运行服务器能让他人通过 *http://<your_machines_ip_address>:<port>* 访问你的应用。当然，这要看你的网络是怎么配置的，可能要设置代理服务器或防火墙。如果无法远程访问开发服务器，请向网络管理员寻求帮助。

3.3 创建 Django 应用

一个 Django 项目中包含一系列配置和应用，这些在一起共同构成一个完整的 Web 应用或网站。这样做便于运用优秀的软件工程实践。把一个 Web 应用分解为多个小应用的好处是，可以把那些小应用放到别的 Django 项目中，无需做多少改动就能使用。

一个 Django 应用完成一件特殊的任务。一个网站需要多少应用，要视其功能而定。例如，一个项目中可能包含一个投票应用、一个注册应用和一个与内容有关的应用。在另一个项目中，我们可能想复用投票和注册应用，因此可以把它们拿过来用。稍后再详细说明。下面创建 Rango 应用。

在 Django 项目所在的目录（例如 *<workspace>/tango_with_django_project*）中执行下述命令：

```
$ python manage.py startapp rango
```

`startapp` 目录在项目的根目录中创建一个新目录，你可能猜到了，这个目录名为 *rango*，其中包含一些 Python 脚本：

- *__init__.py*：与前面那个的作用完全一样；

1. https://www.w3.org/Daemon/User/Installation/PrivilegedPorts.html

- *admin.py*：注册模型，让 Django 为你创建管理界面；
- *apps.py*：当前应用的配置；
- *models.py*：存放应用的数据模型，即数据的实体及其之间的关系；
- *tests.py*：存放测试应用代码的函数；
- *views.py*：存放处理请求并返回响应的函数；
- *migrations* 目录：存放与模型有关的数据库信息。

views.py 和 *models.py* 是任何应用中都有的两个文件，是 Django 所采用的设计模式（即“模型-视图-模板”模式）的主要部分。如果想深入了解模型、视图和模板之间的关系，请阅读 Django 文档。

在动手创建模型和视图之前，必须告诉 Django 项目这个新应用的存在。为此，要修改项目配置目录中的 *settings.py* 文件。打开那个文件，找到 `INSTALLED_APPS` 列表，把 `rango` 添加到末尾：

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rango',
]
```

再次运行开发服务器，确认 Django 识别了这个新应用。如果能正常启动开发服务器，没有任何错误，说明新应用已经成功识别，可以进入下一步了。

★ `startapp` 的自动操作 ★

使用 `python manage.py startapp` 命令创建应用时，Django 可能会把新应用的名称自动添加到 *settings.py* 中的 `INSTALLED_APPS` 列表里。尽管如此，在继续之前自己再检查一下也没什么错。

3.4 编写视图

创建好 Rango 应用后，下面编写一个简单的视图。这是我们编写的第一个视图，简单起见，暂不

使用模型或模板，而是把一些文本发回给客户端。

在你选择的 IDE 中打开新建的 *rango* 目录里的 *views.py* 文件。把 `# Create your views here` 这行注释删掉，得到一个空文件。

然后，写入下述代码：

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Rango says hey there partner!")
```

下面分析一下这三行代码，看这个简单的视图是如何运作的：

- 首先，从 `django.http` 模块中导入 `HttpResponse` 对象。
- 在 *views.py* 文件中，一个函数就是一个视图。这里我们只编写了一个视图，即 `index`。
- 视图函数至少有一个参数，即一个 `HttpRequest` 对象，它也在 `django.http` 模块中。按约定，这个参数名为 `request`，不过你可以根据自己的意愿随意使用其他名称。
- 视图必须返回一个 `HttpResponse` 对象。简单的 `HttpResponse` 对象的参数是一个字符串，表示要发给客户端的页面内容。

有了视图还不行，为了让用户能访问视图，要把一个统一资源定位地址（Uniform Resource Locator，URL）映射到视图上。

为此，打开项目配置目录中的 *urls.py* 文件，在 `urlpatterns` 中添加一行代码：

```
from rango import views

urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^admin/', admin.site.urls),
]
```

新加的那行代码把根 URL 映射到 `rango` 应用的 `index` 视图上。启动开发服务器（`python manage.py runserver`），访问 *http://127.0.0.1:8000* 或你指定的其他地址。你将看到 `index` 视图渲染的输出。

3.5 映射 URL

为了提升模块化程度，我们可以换种方式把入站 URL 映射到视图上，而不直接在项目层设置。首先，要修改项目的 *urls.py* 文件，把针对 Rango 应用的请求交给 Rango 应用处理。然后，在 Rango 应用中指定如何处理请求。

首先，打开项目配置目录中的 *urls.py* 文件。相对 *workspace* 目录而言，这个文件的地址是 *<workspace>/tango_with_django_project/tango_with_django_project/urls.py*。把 `urlpatterns` 列表改成下面这样：

```
from django.conf.urls import url
from django.contrib import admin
from django.conf.urls import include
from rango import views

urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^rango/', include('rango.urls')),
    # 上面的映射把以 rango/ 开头的 URL 交给 rango 应用处理
    url(r'^admin/', admin.site.urls),
]
```

可以看出，`urlpatterns` 是个 Python 列表。新增的映射寻找能匹配 `^rango/` 模式的 URL。遇到这样的 URL 时，`rango/` 后面的部分传给 Rango，由 `rango.urls` 处理。这一步是通过 `django.conf.urls` 模块中的 `include()` 函数实现的。

这是一种分段处理 URL 字符串的方式，如图 3-2 所示。这里，完整的 URL 先去掉域名，余下的部分（`rango/`）传给 tango_with_django 项目，找到匹配的映射后，再把 `rango/` 去掉，把空字符串传给 `rango` 应用处理。

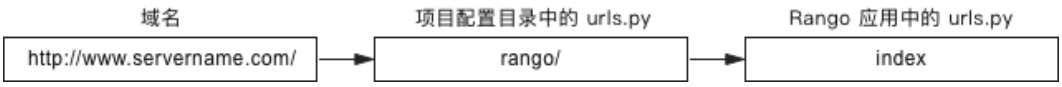

图 3-2：分段处理 URL，域名后的不同的部分由不同的 *url.py* 文件处理

根据上述设置，我们要在 `rango` 应用的目录中新建 *urls.py* 文件，让它处理余下的 URL（即把空字符串映射到 `index` 视图上）：

```
from django.conf.urls import url
from rango import views
```

```
urlpatterns = [
    url(r'^$', views.index, name='index'),
]
```

这段代码先导入 Django 处理 URL 映射的函数和 Rango 应用的 `views` 模块，然后在 `urlpatterns` 列表中调用 `url` 函数映射 `index` 视图。

本书所指的 URL 字符串，都是去掉主机地址后的部分。主机地址是指向 Web 服务器的地址或域名，例如 *http://127.0.0.1:8000* 或 *http://www.tangowithdjango-china.com*。去掉主机地址后，Django 便只需处理 URL 中余下的部分。例如，对 *http://127.0.0.1:8000/rango/about/* 这个 URL 来说，Django 得到的 URL 字符串是 */rango/about/*。

上述代码中的 URL 映射调用 Django 的 `url()` 函数，其第一个参数是正则表达式 `^$`。这个正则表达式匹配空字符串，因为 `^` 表示开头，`$` 表示结尾，而且二者之间没有任何内容，所以只能匹配空字符串。用户访问的 URL，只要匹配这个模式，Django 就会调用 `views.index()` 视图。你可能觉得匹配空 URL 没有什么意义，那为什么要这样做呢？还记得吗，匹配 URL 模式时，只会考虑原 URL 的一部分。Django 先使用项目的 URL 模式处理 URL 字符串（`rango/`），去掉 `rango/` 部分之后得到空字符。然后把空字符串传给 Rango 应用，交给 *rango/urls.py* 中的 URL 模式处理。

传给 `url()` 函数的下一个参数是 `index` 视图，指明处理入站请求的函数。后面的 `name` 参数是可选的，这里把它设为字符串 `'index'`。为 URL 命名的目的是反向解析 URL，即通过名称引用 URL 映射，而不直接使用 URL。讲到模板时再说明这一点。如果想深入了解这个话题，请阅读 Django 文档。

现在，重启 Django 开发服务器，然后访问 *http://127.0.0.1:8000/rango/*。如果一切正常，你应该能看到文本“Rango says hey there partner!”，如图 3-3 所示。

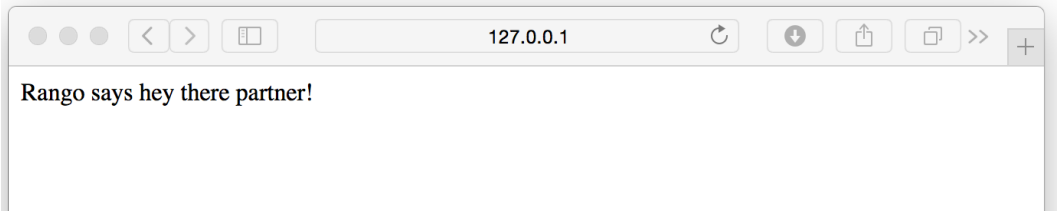

图 3-3：Web 浏览器中显示着首个由 Django 驱动的网页

每个应用中都可以有一些 URL 映射。一开始，URL 映射十分简单，不过随着开发的深入，我们将添加更多复杂的参数化 URL 映射。

你要理解 Django 处理 URL 的方式。现在你可能还有点迷糊，不过用得多了，终究会明白的。如

果想深入了解 URL 映射，想看更多的示例，请阅读 Django 文档。

★ 正则表达式 ★

Django 的 URL 模式使用正则表达式匹配 URL，因此你要熟练使用 Python 的正则表达式。Python 官方文档中有一篇关于正则表达式的指南，此外也可以查看 Regex Cheat Sheet 网站中的速查表。

如果使用版本控制系统，现在是提交改动的好时机。

3.6 基本流程

本章内容可以总结为一系列操作。这一节给出我们所执行的两个任务的操作步骤。如果以后记不得了，可以随时翻阅。

创建 Django 项目

1. 执行 `python django-admin.py startproject <name>` 命令，其中 `<name>` 是想创建的项目名称。

创建 Django 应用

1. 执行 `python manage.py startapp <appname>` 命令，其中 `<appname>` 是想创建的应用名称。
2. 把应用名称添加到项目配置目录中的 *settings.py* 文件里，放到 `INSTALLED_APPS` 列表的末尾，告诉 Django 项目这个应用的存在。
3. 在项目的 *urls.py* 文件中添加一个映射，指向新建的应用。
4. 在应用的目录中新建 *urls.py* 文件，把入站 URL 与视图对应起来。
5. 在应用的 *view.py* 文件中编写所需的视图，确保视图返回一个 `HttpResponse` 对象。

</> 练习

我们创建了一个 Django 项目，而且把新建的应用运行起来了。请试着完成以下练习，巩固所学的知识。走到这一步不简单，是学习 Django 过程中的一个重要里程碑。编写视图并把 URL 映射到视图上是开发更加复杂的 Web 应用所必须迈出的第一步。

- 回顾本章的内容，确保自己理解了 URL 是如何映射到视图上的。
- 再编写一个视图函数，名为 `about`，返回“Rango says here is the about page.”。
- 把这个视图映射到 URL */rango/about/* 上。只需编辑 Rango 应用的 *urls.py* 文件。记住，`/rango/` 部分由项目的 *urls.py* 文件处理。
- 修改 `index` 视图中的 `HttpResponse` 对象，加入一个指向关于页面的链接。
- 在 `about` 视图的 `HttpResponse` 对象中添加一个指向主页的链接。
- 既然开始阅读本书了，那就在 Twitter 上关注 @tangowithdjango 吧，告诉我们你学的怎么样。

★ 提示 ★

如果你做不出上述练习，希望下面的提示能给你一点启发。

- 在 *views.py* 文件中定义一个函数，`def about(request):`，让它返回一个 `HttpResponse` 对象，在参数中指明要返回的 HTML。
- 匹配 URL */about/* 的正则表达式是 `r'^about/'`。在 *rango/urls.py* 文件中为 `about()` 视图增加一个映射。
- 修改 `index()` 视图，添加一个指向 `about` 视图的链接。简单起见，现在可以这样写：`Rango says hey there partner! <br/> <a href='/rango/about/'>About</a>.`。
- 同样，在 `about()` 视图中添加指向首页的链接：`<a href="/rango/">Index</a>`。
- 如果你还没读过 Django 官方教程，现在请阅读第一部分。

4 模板和媒体文件

本章介绍 Django 的模板引擎，并说明如何在应用中伺服静态文件和媒体文件。

4.1 使用模板

目前，我们只把一个 URL 映射到一个视图上。然而，Django 框架采用的是“模型-视图-模板”架构。这一节说明模板与视图的关系，后面几章再讨论如何与模型联系起来。

为什么使用模板？网站中的不同页面通常使用相同的布局，提供通用的*页头*（header）和*页脚*（footer），为用户呈现导航，体现一种一致性。Django 模板能让开发者轻易实现这样的设计要求，而且还能把应用逻辑（视图代码）与表现（应用的外观）区分开。本章将创建一个简单的模板，用于生成 HTML 页面，交由 Django 视图调度。第 5 章将更近一步，结合模型动态分发数据。

★ 总结：模板是什么？ ★

Django 的模板可以理解为构建完整的 HTML 页面所需的骨架。模板中有静态内容（不变的部分），也有特殊的句法（动态内容，即模板标签）。Django 视图会把动态内容替换成真正的数据，生成最终的 HTML 响应。

配置模板目录

若想在 Django 应用中使用模板，要创建两个目录，用于存放模板文件。

在 Django 项目配置目录（*<workspace>/tango_with_django_project/*）中创建一个名为 *templates* 的目录。注意，这个目录要与项目的 *manage.py* 脚本放在同一级。在这个新目录中再创建一个目

录，名为 *rango*。我们将在 *<workspace>/tango_with_django_project/templates/rango/* 目录中存放 Rango 应用的模板。

★ 以合理的方式组织模板 ★

建议把各应用的模板放在单独的子目录中。这就是我们在 *templates* 目录中创建 *rango* 子目录的原因。如果想打包应用，把它分发给其他开发者，这样就便于区分模板属于哪个应用。

接下来要告诉 Django 你把模板放在什么位置。打开项目的 *settings.py* 文件，找到 `TEMPLATES`。如果项目是使用 Django 1.9 创建的，`TEMPLATES` 的默认内容如下：

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

为了告诉 Django，模板在何处，我们要修改 `DIRS` 列表（默认为空）。把这个键值对改成下面这样：

```
'DIRS': ['<workspace>/tango_with_django_project/templates']
```

注意，这里要使用*绝对路径*（absolute path）。然而，与团队成员协作，或者换台电脑的话，这就是个问题了。用户名和磁盘结构变了，`<workspace>` 目录的路径也就不一样了。这个问题的一种解决方法是列出每个可能的路径，例如：

```
'DIRS': [ '/Users/leifos/templates',
          '/Users/maxwelld90/templates',
          '/Users/davidm/templates', ]
```

可是这样做也有诸多问题。首先，每增加一个团队成员就要增加一个路径。其次，如果换成其他操作系统（例如 Windows），斜线可能都要换种形式。

◆ 不要硬编码路径 ◆

硬编码路径是自找麻烦。硬编码路径是软件工程的一种反模式，会导致项目不易移植，也就是说在一台电脑中运行好好的，换台设备可能就会出错。

动态路径

更好的方法是使用 Python 内置的函数自动找出 *templates* 目录的路径。这样无论你把 Django 项目的代码放在何处，最终都能得到一个绝对路径。因此，项目的可移植性更高。

settings.py 文件的顶部有个名为 `BASE_DIR` 的变量，它的值是 *settings.py* 文件所在目录的路径。这里用到了 Python 的特殊属性 `__file__`，它的值是所在文件的绝对路径。调用 `os.path.dirname()` 的作用是获取 *settings.py* 文件所在目录的绝对路径，再调用一次 `os.path.dirname()` 又去掉一层，因此 `BASE_DIR` 最终的值是 *<workspace>/tango_with_django_project/*。如果你还不太理解这个过程，可以把下面几行代码放到 *settings.py* 文件中：

```
print(__file__)
print(os.path.dirname(__file__))
print(os.path.dirname(os.path.dirname(__file__)))
```

有了 `BASE_DIR` 之后，我们便可以轻易引用 Django 项目中的文件和目录。我们可以定义一个名为 `TEMPLATE_DIR` 的变量，指向 `templates` 目录的位置。这里还要使用 `os.path.join()` 函数拼接多个路径片段。`TEMPLATE_DIR` 变量的定义如下：

```
TEMPLATE_DIR = os.path.join(BASE_DIR, 'templates')
```

我们使用 `os.path.join()` 函数把 `BASE_DIR` 变量和 `'templates'` 字符串拼接起来，得到 `<workspace>/tango_with_django_project/templates/`。如此一来，我们便可以使用 `TEMPLATE_DIR` 变量替代前面在 `TEMPLATES` 中硬编码的路径。把 `DIRS` 键值对改成下面这样：

```
'DIRS': [TEMPLATE_DIR, ]
```

★ 为什么命名为 TEMPLATE_DIR? ★

我们在 *settings.py* 文件的顶部定义了一个名为 `TEMPLATE_DIR` 的变量，采用这个名称是因为它能表明变量的作用，而且基本不会修改。在较复杂的 Django 项目中，可以在 `DIRS` 列表中指定多个模板目录，但是本书用这一个就够了。

◆ 拼接路径 ◆

拼接系统路径时一定要使用 `os.path.join()` 函数，这样为的是使用正确的路径分隔符。Unix 操作系统（及其衍生系统）使用正斜线（`/`）分隔目录，而 Windows 操作系统使用反斜线（`\`）。如果直接使用斜线，更换操作系统后可能导致路径错误，从而降低项目的可移植性。

添加一个模板

模板目录和路径设置好之后，在 *templates/rango/* 目录中创建一个文件，命名为 *index.html*。在这个新文件中写入下述 HTML 代码。

```
<!DOCTYPE html>
<html>

    <head>
        <title>Rango</title>
    </head>

    <body>
        <h1>Rango says...</h1>
        <div>
            hey there partner! <br />
            <strong>{{ boldmessage }}</strong><br />
        </div>
        <div>
            <a href="/rango/about/">About</a><br />
        </div>
    </body>

</html>
```

这段 HTML 代码很好理解，最终得到的 HTML 页面将向用户打个招呼。你可能注意到了，这里有些内容不是 HTML，而是 `{{ boldmessage }}` 形式。这是 Django 模板变量。我们可以为这样的变量设值，这样渲染模板后便会显示我们设定的值。稍后再做。

为了使用这个模板，我们要调整一下前面编写的 `index()` 视图，不再分发一个简单的响应对象，而是分发这个模板。

打开 *rango/views.py* 文件，看看文件顶部有没有下面这个 `import` 语句。如果没有，加上。

```
from django.shortcuts import render
```

然后根据下述代码片段修改 `index()` 视图函数。各行代码的作用参见注释。

```
def index(request):
    # 构建一个字典，作为上下文传给模板引擎
    # 注意，boldmessage 键对应于模板中的 {{ boldmessage }}
    context_dict = {'boldmessage': "Crunchy, creamy, cookie, candy, cupcake!"}

    # 返回一个渲染后的响应发给客户端
    # 为了方便，我们使用的是 render 函数的简短形式
    # 注意，第二个参数是我们想使用的模板
    return render(request, 'rango/index.html', context=context_dict)
```

首先，构建一个字典，设定要传给模板的数据。然后，调用 `render()` 辅助函数。这个函数的参数是 `request` 对象、模板的文件名和上下文字典。`render()` 函数将把上下文字典中的数据代入模板，生成一个完整的 HTML 页面，作为 `HttpResponse` 对象返回，分发给 Web 浏览器。

★ 模板上下文是什么？ ★

Django 的模板系统加载模板文件时会创建一个模板上下文。简单而言，模板上下文是一个 Python 字典，把模板变量名映射到值上。在前面的模板中，有个名为 `boldmessage` 的模板变量。在修改后的 `index()` 视图中，我们把 `Crunchy, creamy, cookie, candy, cupcake!` 字符串赋给模板变量 `boldmessage`。这样，渲染模板时，模板中的 `{{ boldmessage }}` 将会替换为字符串 `Crunchy, creamy, cookie, candy, cupcake!`。

我们已经更新视图，用上了模板。现在启动 Django 开发服务器，然后访问 *http://127.0.0.1:8000/rango/*。你应该能看到这个简单的 HTML 模板渲染出来了，如图 4-1 所示。

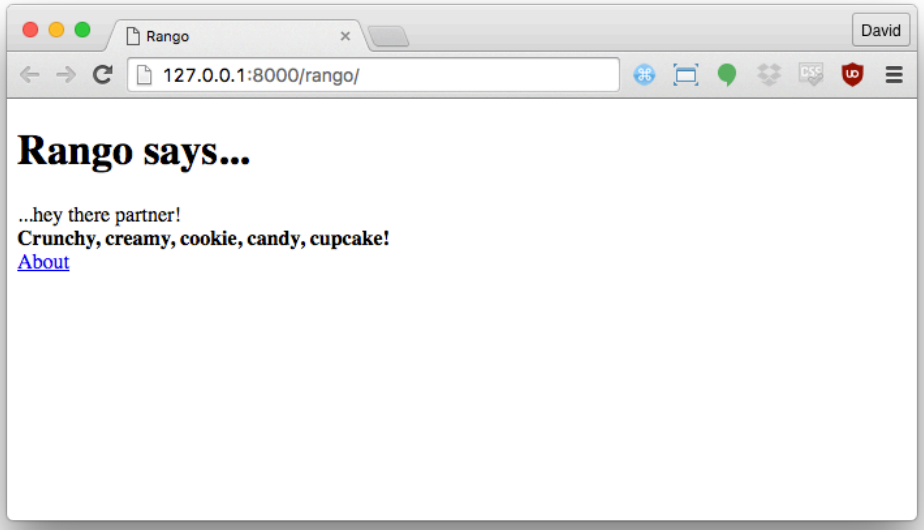

图 4-1：一切正常时应该看到的页面。注意加粗的文本，即“Crunchy, creamy, cookie, candy, cupcake!”，它们取自视图，通过模板渲染出来。

如果没看到上述页面，读一下错误消息，看看是什么地方出错了，再次确认前面所做的改动。常见的问题之一是，*settings.py* 文件中的模板路径设置的不对。可以在 *settings.py* 文件中使用 `print` 语句输出 `BASE_DIR` 和 `TEMPLATE_DIR` 的值，确认一切是否正常。

这个示例演示了如何在视图中使用模板。然而，我们只用了 Django 模板引擎丰富功能中的一点皮毛。本书后文还将使用更复杂的模板功能。如果想深入了解模板，请阅读 Django 文档。

4.2 伺服静态文件

尽管我们用上了模板，但是不得不承认，Rango 应用现在还有点简陋，没有样式也没有图像装饰。为了改善这种状况，我们可以在 HTML 模板中引用其他文件，例如层叠样式表（Cascading Style Sheet，CSS）、JavaScript 和图像。这些是静态文件（static file），因为它们不是由 Web 服务器动态生成的，而是原封不动发给 Web 浏览器。本节说明 Django 伺服静态文件的方式，以及如何在模板中添加一个图像。

配置静态文件目录

首先要指定一个目录，用于存放静态文件。在项目配置目录中新建一个目录，名为 *static*，然后在

static 目录中再新建一个目录，名为 *images*。确保 *static* 目录与前面创建的 *templates* 目录位于同一级。

然后，在 *images* 目录中放一个图像。如下图所示，我们选择的是动画电影《兰戈》中变色龙兰戈的图片。

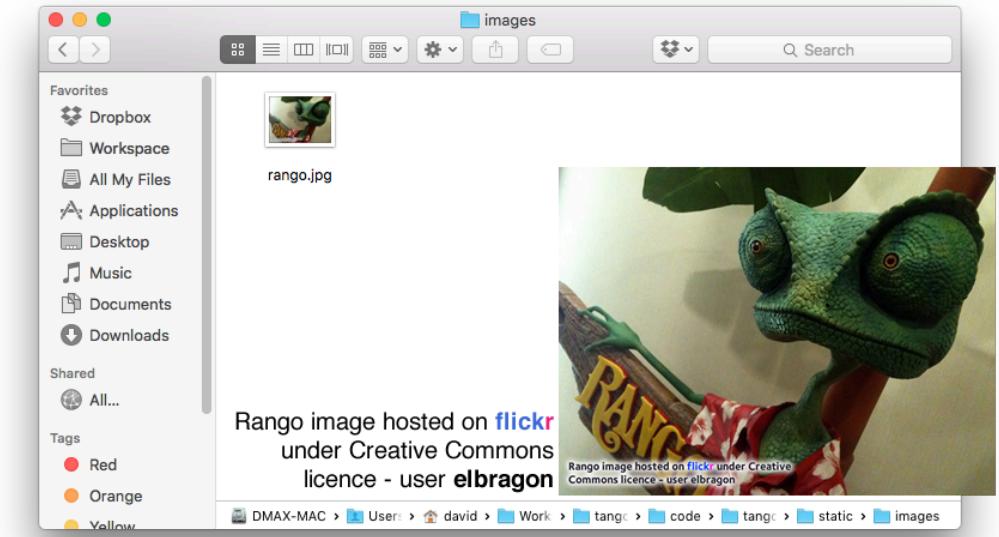

图 4-2：把变色龙兰戈的图片放到 *static/images* 目录中

与前面的 *templates* 目录一样，我们要告诉 Django 这个 *static* 目录的路径。为此，还要编辑项目的 *settings.py* 模块。在这个文件中，我们要定义一个变量，指向 *static* 目录，并在一个数据结构中告诉 Django 这个目录的路径。

首先，在 *settings.py* 文件的顶部定义一个变量，名为 `STATIC_DIR`。为了把所有路径放在一起，可以把 `STATIC_DIR` 放在 `BASE_DIR` 和 `TEMPLATE_DIR` 下面。`STATIC_DIR` 的值也应该使用前面用过的 `os.path.join`，不过这一次是指向 *static* 目录，如下所示：

```
STATIC_DIR = os.path.join(BASE_DIR, 'static')
```

这样得到的是一个绝对路径，指向 *<workspace>/tango_with_django_project/static/*。定义好这个变量之后，还要创建一个数据结构，名为 `STATICFILES_DIRS`。这个数据结构的值是一系列路径，让 Django 在其中寻找要伺服的静态文件。默认情况下，*settings.py* 文件中没有这个列表。在创建之前，确认这个列表确实不存在。如果定义两次，Django 会混淆的，你自己也是一样。

本书只在一个位置存放项目的静态文件，即 `STATIC_DIR` 定义的路径。因此，我们可以像下面这样

设置 STATICFILES_DIRS：

```
STATICFILES_DIRS = [STATIC_DIR, ]
```

★ 保持 *settings.py* 整洁有序 ★

最好保持 *settings.py* 模块整洁有序。不要随意乱放，要有组织。把定义目录位置的变量放在模块的顶部，这样便于查找。把 STATICFILES_DIRS 放在与静态文件相关的那部分（靠近底部）。这样对你和其他协作者来说都方便。

最后，检查 *settings.py* 模块中有没有定义 STATIC_URL 变量。如果没有，像下面那样定义。注意，Django 1.9 默认把这个变量放在靠近底部的位置，因此你可能要向下拉滚动条才能找到。

```
STATIC_URL = '/static/'
```

我们做了这么多，还没说为什么要这么做。简单来说，STATIC_DIR 和 STATICFILES_DIRS 两个变量设定静态文件在电脑中的位置；STATIC_URL 变量则指定启动 Django 开发服务器后通过什么 URL 访问静态文件。例如，把 STATIC_URL 设为 /static/ 后，我们可以通过 *http://127.0.0.1:8000/static/* 访问里面的静态内容。前两个变量相当于服务器端的位置，而第三个变量是客户端访问静态内容的位置。

</> 测试配置

现在，请测试一下一切是否配置正确。启动 Django 开发服务器，在浏览器中访问 *rango.jpg* 图像试试。如果把 STATIC_URL 设为 /static/，而 *rango.jpg* 的位置是 *images/rango.jpg*，想一想应该在 Web 浏览器中输入什么 URL？

在继续阅读之前，请仔细想一想。想不出来也没关系，后文会告诉你。

◆ 别忘了斜线 ◆

设置 STATIC_URL 时，确保 URL 的末尾有一个正斜线（例如，是 /static/，而不是 /static）。根据 Django 文档，缺少末尾的斜线会导致一系列问题。在末尾加上斜线的目的是把 URL 的根部（例如 /static/）与要伺服的静态内容（例如 images/rango.jpg）区分

开。

★ 伺服静态内容 ★

在开发环境中使用 Django 开发服务器伺服静态文件没问题，但是在生产环境中却不合适。Django 文档中有关于在生产环境中部署静态文件的更多信息。第 18 章将深入讨论这个问题。

如果你没想出应该通过哪个 URL 访问那个图像，请在 Web 浏览器中输入 *http://127.0.0.1:8000/static/images/rango.jpg*。

在模板中引用静态文件

我们已经做好设置，Django 项目能处理静态文件了。现在可以在模板中利用静态文件改进外观及增添功能了。

下面说明如何引用静态文件。打开 *<workspace>/templates/rango/* 目录中的 *index.html* 模板，参照下述代码修改。为了方便查找，新增的行旁边有注释。

```
<!DOCTYPE html>

{% load staticfiles %} <!-- 新增 -->

<html>
    <head>
        <title>Rango</title>
    </head>

    <body>
        <h1>Rango says...</h1>

        <div>
            hey there partner! <br />
            <strong>{{ boldmessage }}</strong><br />
        </div>
```

```
        <div>
            <a href="/rango/about/">About</a><br />
            <img src="{% static "images/rango.jpg" %}"
                 alt="Picture of Rango" /> <!-- 新增 -->
        </div>
    </body>

</html>
```

新增的第一行（`{% load staticfiles %}`）告诉 Django 模板引擎，我们将在模板中使用静态文件。这样便可以在模板中使用 `static` 模板标签引入静态目录中的文件。`static "images/rango.jpg"` 告诉 Django，我们想显示静态目录中名为 *images/rango.jpg* 的图像。`static` 标签会在 *images/rango.jpg* 前加上 `STATIC_URL` 指定的 URL，得到 */static/images/rango.jpg*。Django 模板引擎生成的 HTML 如下：

```
<img src="/static/images/rango.jpg" alt="Picture of Rango" />
```

建议为图像设定替代文本，以防图片由于某些原因无法加载。替代文本通过 `img` 标签的 `alt` 属性指定，效果如下。

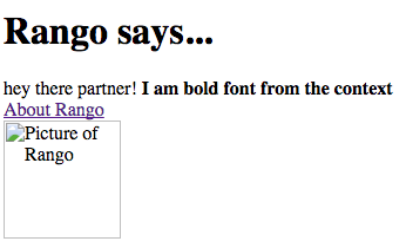

图 4-3：找不到图像时显示 alt 属性的值

修改视图之后，启动 Django 开发服务器，然后访问 *http://127.0.0.1:8000/rango*。如果一切正常，应该会看到类似图 4-4 中的网页。

★ 模板中的 <!DOCTYPE> ★

HTML 模板的第一行始终是 DOCTYPE 声明。如果把 `{% load staticfiles %}` 放在前面，渲染得到的 HTML 中，DOCTYPE 声明前面将出现空白。多出的空白会导致 HTML 标记无法通过验证。

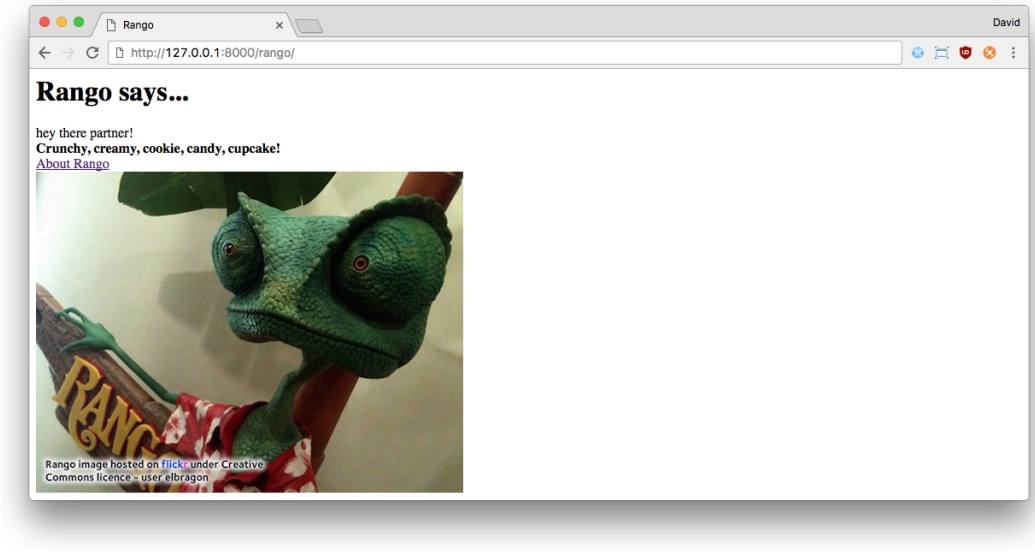

图 4-4：我们的第一个模板中显示着变色龙兰戈的图片

★ 加载其他静态文件 ★

只要想在模板中引用静态文件，就可以使用 {% static %} 模板标签。下述代码片段说明如何在模板中引入 JavaScript、CSS 和图像。

```
<!DOCTYPE html>
{% load staticfiles %}

<html>

    <head>
        <title>Rango</title>
        <!-- CSS -->
        <link rel="stylesheet" href="{% static "css/base.css" %}" />
        <!-- JavaScript -->
        <script src="{% static "js/jquery.js" %}"></script>
    </head>

    <body>
        <!-- 图像 -->
        <img src="{% static "images/rango.jpg" %}" alt="Picture of Rango" />
    </body>
</html>
```

显然，*static* 目录中要有你引用的静态文件。如果引用的文件不存在，或者未正确引用，Django 开发服务器在控制台中的输出将报告 HTTP 404 错误。你可以引用一个不存在的文件试试。注意下述输出中的最后一条，HTTP 状态码是 404。

```
[10/Apr/2016 15:12:48] "GET /rango/ HTTP/1.1" 200 374
[10/Apr/2016 15:12:48] "GET /static/images/rango.jpg HTTP/1.1" 304 0
[10/Apr/2016 15:12:52] "GET /static/images/not-here.jpg HTTP/1.1" 404 0
```

如果想深入了解引入静态文件的方式，请阅读 Django 文档。

4.3 伺服媒体文件

应用中的静态文件可以理解为不变的文件。不过，有时还要使用可变的媒体文件（media file）。这类文件可由用户或管理员上传，因此可能会变化。比如说，用户的头像就是媒体文件，电商网站中的商品图片也是媒体文件。

为了能伺服媒体文件，我们要修改 Django 项目的设置。这一节说明具体需要做哪些设置，但暂不测试，等到实现用户上传头像功能时再做检查。

★ 伺服媒体文件 ★

与静态文件一样，在 Django 开发环境中也能检查设置是否正确。当然，开发环境中伺服媒体文件的方法极其不适合在生产环境中使用，因此部署后应该寻找其他方式托管应用的媒体文件。详情参见第 18 章。

修改 *settings.py*

首先，打开 Django 项目配置目录中的 *settings.py* 模块。我们将在这个文件中添加一些内容。与静态文件一样，媒体文件也放在文件系统中专门的一个目录中。因此，要告诉 Django 这个目录的位置。

在 *settings.py* 模块的顶部，找到 `BASE_DIR`、`TEMPLATE_DIR` 和 `STATIC_DIR` 变量。在它们后面加上 `MEDIA_DIR`：

```
MEDIA_DIR = os.path.join(BASE_DIR, 'media')
```

这一行告诉 Django，媒体文件将上传到 Django 项目根目录中的 *media* 目录里，即 *<workspace>/tango_with_django_project/media/*。正如前文所说，把路径相关的变量放在 *settings.py* 模块的顶部便于以后修改。

然后在 *settings.py* 中找个地方添加两个变量：`MEDIA_ROOT` 和 `MEDIA_URL`。Django 伺服媒体文件时会用到这两个变量。

```
MEDIA_ROOT = MEDIA_DIR
MEDIA_URL = '/media/'
```

◆ 同样，别忘了斜线 ◆

与 `STATIC_URL` 变量一样，`MEDIA_URL` 变量的末尾要有一条正斜线（例如，是 `/media/`，而不是 `/media`）。在末尾加上斜线的目的是把 URL 的根部（例如 `/media/`）与用户上传的内容区分开。

`MEDIA_ROOT` 变量告诉 Django 在哪里寻找上传的媒体文件，`MEDIA_URL` 变量则指明通过什么 URL 伺服媒体文件。这样设置之后，上传的 *cat.jpg* 文件在 Django 开发服务器中将通过 *http://localhost:8000/media/cat.jpg* 访问。

后面在模板中引入上传的内容时，如果能方便引用 `MEDIA_URL` 路径就好了。为了方便这样的操作，Django 提供了模板上下文处理器。严格来说，现在无需这么做，但是趁机加上也没关系。

找到 *settings.py* 文件中的 `TEMPLATES` 列表，里面嵌套着 `context_processors` 列表。在 `context_processors` 列表中添加一个处理器，`django.template.context_processors.media`。添加之后，`context_processors` 列表应该类似下面这样：

```
'context_processors': [
    'django.template.context_processors.debug',
    'django.template.context_processors.request',
    'django.contrib.auth.context_processors.auth',
    'django.contrib.messages.context_processors.messages',
    'django.template.context_processors.media'
],
```

调整 URL

在开发环境中伺服媒体文件的最后一步是，让 Django 使用 `MEDIA_URL` 伺服媒体内容。打开项目的 *urls.py* 模块，修改 `urlpatterns` 列表，调用 `static()` 函数：

```
urlpatterns = [
    ...
    ...
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

此外，还要在 *urls.py* 模块的顶部添加下述 `import` 语句：

```
from django.conf import settings
from django.conf.urls.static import static
```

现在可以通过 */media/* URL 访问 *media* 目录中的媒体文件了。

4.4 基本流程

至此，你应该知道如何设置和创建模板，知道如何在视图中使用模板，知道如何设置并让 Django 开发服务器伺服静态媒体文件，以及如何在模板中引入图像了。本章的知识可不少！

本章的重点是知道如何创建模板，并在 Django 视图中使用模板。这其中涉及好几步，不过多做几次就会习惯的。

1. 首先，创建要使用的模板，保存到 *templates* 目录中（在项目的 *settings.py* 模块中设定）。模板中可以使用 Django 模板变量（例如 `{{ variable_name }}`）或模板标签。模板变量的值在相应的视图中设定。
2. 在应用的 *views.py* 文件中找到所需的视图，或者新建一个。
3. 把视图相关的逻辑写在视图函数中。例如，从数据库中检索数据，存到列表中。
4. 在视图中构建一个字典，通过模板上下文传给模板引擎。
5. 使用 `render()` 辅助函数生成响应。这个函数的参数是请求对象、模板文件名和上下文字典。
6. 如果还没把视图映射到 URL 上，修改项目的 *urls.py* 文件和应用的 *urls.py* 文件。

在网页中引入静态文件的步骤也很重要，应该熟练掌握。具体方法参见下述步骤。

1. 把想用的静态文件放到项目的 *static* 目录中。这个目录在项目的 *settings.py* 模块中的 `STATICFILES_DIRS` 列表中设定。
2. 在模板中引用静态文件。例如，图像通过 `<img />` 标签插入 HTML 页面。
3. 记得在模板中加上 `{% load staticfiles %}`，然后使用 `{% static "<filename>" %}` 标签引用静态文件。把 `<filename>` 替换成图像或其他资源的路径。只要想引用静态文件，使用 `static` 模板标签。

伺服媒体文件的步骤与伺服静态文件差不多。

1. 把媒体文件放到项目的 *media* 目录中。这个目录由项目的 *settings.py* 模块中的 `MEDIA_ROOT` 变量设定。
2. 在模板中使用 `{{ MEDIA_URL }}` 上下文变量引用媒体文件。例如，引用上传的图像 *cat.jpg*：`<img src="{{ MEDIA_URL}}cat.jpg" />`。

</> 练习

请完成以下练习，巩固本章所学的知识。

- ❑ 让关于页面也使用模板渲染，模板名为 *about.html*。
- ❑ 在 *about.html* 模板中引入一个图片（存储在项目的 *static* 目录中）。
- ❑ 在关于页面中添加一行：`This tutorial has been put together by <your-name>.`。
- ❑ 在 Django 项目配置目录中新建一个目录，命名为 *media*。从网上下载一张猫的图片，保存到 *media* 目录中，命名为 *cat.jpg*。
- ❑ 在关于页面中添加一个 `<img>` 标签，显示那个猫的图片，确保媒体文件能正确伺服。加上首页中的兰戈图片，现在你的应用既可以伺服静态文件，也可以伺服媒体文件。

★ 静态文件 vs. 媒体文件 ★

从名称可以看出，静态文件是不变的，是网站的核心构成组件。而媒体文件是用户提供的，时常变化。

样式表是静态文件，定义网页的外观。而用户的头像是媒体文件，用户在注册账户时上传。