

20.2 部署“学习笔记”

至此，项目“学习笔记”的外观显得很专业，下面将其部署到服务器，让任何有互联网连接的人都能够使用它。为此，我们将使用Heroku。这是一个基于Web的平台，供我们管理Web应用程序的部署。我们将让“学习笔记”在Heroku上运行起来。

20.2.1 建立Heroku账户

要建立账户，请访问Heroku官方网站，并单击其中一个注册链接。注册账户是免费的，Heroku提供的免费试用服务（free tier）让你能够将项目部署到服务器并对其进行测试。

注意 Heroku提供的免费试用服务存在一些限制，如可部署的应用程序数量以及用户访问应用程序的频率。但这些限制都很宽松，让你能够在不支付任何费用的情况下练习部署应用程序。

20.2.2 安装Heroku CLI

要将项目部署到Heroku的服务器并对其进行管理，需要使用Heroku CLI（Command Line Interface，命令行界面）提供的工具。要安装最新的Heroku CLI版本，请访问Heroku Dev Center网站的The Heroku CLI页面，并根据你使用的操作系统按相关的说明做：使用只包含一行的终端命令或下载并运行安装程序。

20.2.3 安装必要的包

我们还需安装三个包，以便在服务器上支持Django项目提供的服务。为此，在活动状态的虚拟环境中执行如下命令：

```
(11_env)learning_log$ pip install psycopg2==2.7.*  
(11_env)learning_log$ pip install django-heroku  
(11_env)learning_log$ pip install gunicorn
```

为管理Heroku使用的数据库，`psycopg2` 包必不可少。`django-heroku` 包用于管理应用程序的各种配置，使其能够在Heroku服务器上正确地运行。这包括管理数据库，以及将静态文件存储到合适的地方，以便能够妥善地提供它们。静态文件包括样式规则和JavaScript文件。`gunicorn` 包让服务器能够实时地支持应用程序。

20.2.4 创建文件requirements.txt

Heroku需要知道项目依赖于哪些包，因此我们使用pip生成一个文件，在其中列出这些包。同样，进入活动虚拟环境，并执行如下命令：

```
(11_env)learning_log$ pip freeze > requirements.txt
```

命令freeze 让pip将项目中当前安装的所有包的名称都写入文件 requirements.txt。请打开文件requirements.txt，查看项目中安装的包及其版本：

requirements.txt

```
dj-database-url==0.5.0
Django==2.2.0
django-bootstrap4==0.0.7
django-heroku==0.3.1
gunicorn==19.9.0
psycopg2==2.7.7
pytz==2018.9
sqlparse==0.2.4
whitenoise==4.1.2
```

“学习笔记”依赖于8个特定版本的包，因此在相应的环境中才能正确地运行。在这8个包中，有4个是我们手工安装的，余下的4个是作为依赖的包自动安装的。

我们部署“学习笔记”时，Heroku将安装requirements.txt列出的所有包，从而创建一个环境，其中包含在本地使用的所有包。有鉴于此，我们可以深信在部署到Heroku后，项目的行为将与在本地系统上完全相同。当你在自己的系统上开发并维护各种项目时，这将是一个巨大的优点。

注意 如果在你的系统中，requirements.txt列出的包的版本与上面列出的不同，请保留原来的版本号。

20.2.5 指定Python版本

如果没有指定Python版本，Heroku将使用其当前的Python默认版本。下面来确保Heroku使用我们使用的Python版本。为此，在活动状态的虚拟环境中，执行命令python --version：

```
(11_env)learning_log$ python --version
Python 3.7.2
```

上述输出表明，我使用的是Python 3.7.2。请在manage.py所在的文件夹中新建一个名为runtime.txt的文件，并在其中输入如下内容：

runtime.txt

```
python-3.7.2
```

这个文件应只包含一行内容，以上面所示的格式指定你使用的Python版本。请确保输入小写的python，在它后面输入一个连字符，再输入由三部分组成的版本号。

注意 如果出现错误消息，指出不能使用指定的Python版本，请访问Heroku Dev Center网站的Language Support页面，再单击到Specifying a Python Runtime的链接。浏览打开的文章，了解支持的Python版本，并使用与你使用的Python版本最接近的版本。

20.2.6 为部署到Heroku而修改settings.py

现在需要在settings.py末尾添加一个片段，在其中指定一些Heroku环境设置：

settings.py

```
--snip--  
# 我的设置  
LOGIN_URL = 'users:login'  
  
# Heroku设置  
import django_heroku  
django_heroku.settings(locals())
```

这里导入了模块django_heroku 并调用了函数settings()。这个函数将一些设置修改为Heroku环境要求的值。

20.2.7 创建启动进程的Procfile

Procfile 告诉Heroku应该启动哪些进程，以便正确地提供项目需要的服务。请将这个只包含一行代码的文件命名为Procfile（首字母P为大写），但不指定文件扩展名，再将其保存到manage.py所在的目录中。

Procfile的内容如下：

Procfile

```
web: gunicorn learning_log.wsgi --log-file -
```

这行代码让Heroku将Gunicorn用作服务器，并使用learning_log/wsgi.py中的设置来启动应用程序。标识log-file 告诉Heroku应将哪些类型的事件写入日志。

20.2.8 使用Git跟踪项目文件

第17章说过，Git是一个版本控制程序，让你能够在每次成功实现新功能后都拍摄项目代码的快照。无论出现什么问题（如实现新功能时不小心引入了bug），都可轻松地恢复到最后一个可行的快照。每个快照都称为提交。

使用Git意味着在尝试实现新功能时无须担心破坏项目。将项目部署到服务器时，需要确保部署的是可行版本。要更详细地了解Git和版本控制，请参阅附录D。

a. 安装Git

在你的系统中，可能已经安装了Git。要确定是否安装了Git，可打开一个新的终端窗口，并在其中执行命令`git --version`：

```
(11_env) learning_log$ git --version  
git version 2.17.0
```

如果由于某种原因出现了错误消息，请参阅附录D中的Git安装说明。

β. 配置Git

Git跟踪是谁修改了项目，即便项目由一个人开发亦如此。为进行跟踪，Git需要知道你的用户名和电子邮箱。因此，你必须提供用户名，但对于练习项目，可以编造一个电子邮箱：

```
(11_env) learning_log$ git config --global user.name  
"ehmatthes"  
(11_env) learning_log$ git config --global user.email  
"eric@example.com"
```

如果忘记了这一步，首次提交时Git将提示你提供这些信息。

γ. 忽略文件

无须让Git跟踪项目中的每个文件，因此我们让它忽略一些文件。在`manage.py`所在的文件夹中创建一个名为`.gitignore`的文件（请注意，这个文件名以句点打头，且不包含扩展名），并在其中输入如下代码：

```
.gitignore
```

```
ll_env/  
__pycache__/  
*.sqlite3
```

这里让Git忽略目录ll_env，因为随时都可自动重新创建它。还指定不跟踪目录__pycache__，这个目录包含Django运行.py文件时自动创建的.pyc文件。我们没有跟踪对本地数据库的修改，因为这是个坏习惯：如果在服务器上使用的是SQLite，将项目推送到服务器时，可能会不小心用本地测试数据库覆盖在线数据库。
*.sqlite3让Git忽略所有扩展名为.sqlite3的文件。

注意 如果你使用的是macOS系统，请将.DS_Store添加到文件.gitignore中。文件.DS_Store存储的是有关文件夹设置的信息，与这个项目一点关系都没有。

8. 显示隐藏的文件

大多数操作系统都会隐藏名称以句点打头的文件和文件夹，如.gitignore。当你打开文件浏览器或在诸如Sublime Text等应用程序中打开文件时，默认看不到隐藏的文件。

不过作为程序员，你需要看到它们。下面说明了如何显示隐藏的文件。

- 在Windows系统中，打开资源管理器，再打开一个文件夹，如Desktop。单击标签View（查看），并确保选中了复选框File name extensions（文件扩展名）和Hidden items（隐藏的项目）。
- 在macOS系统中，要显示隐藏的文件和文件夹，可在文件浏览器窗口中按Command + Shift + .（句点）。
- 在Ubuntu等Linux系统中，可在文件浏览器中按Ctrl + H来显示隐藏的文件和文件夹。为让这种设置为永久性的，可打开文件浏览器（如Nautilus），再单击选项标签（以三条线表示），并选中复选框Show Hidden Files（显示隐藏的文件）。

ε. 提交项目

我们需要为“学习笔记”初始化一个Git仓库，将所有必要的文件都加入该仓库，并提交项目的初始状态，如下所示：

```
❶ (ll_env)learning_log$ git init  
Initialized empty Git repository in  
/home/ehmatthes/pcc/learning_log/.git/
```

```

❶ (ll_env)learning_log$ git add .
❷ (ll_env)learning_log$ git commit -am "Ready for deployment to
heroku."
[master (root-commit) 79fef72] Ready for deployment to
heroku.
 45 files changed, 712 insertions(+)
  create mode 100644 .gitignore
  create mode 100644 Procfile
  --snip--
  create mode 100644 users/views.py
❸ (ll_env)learning_log$ git status
On branch master
nothing to commit, working tree clean
(ll_env)learning_log$
```

在❶处，执行命令`git init`，在“学习笔记”所在的目录中初始化一个空仓库。在❷处，执行命令`git add .`（千万别忘了这个句点），将未被忽略的文件都加入这个仓库。在❸处，执行命令`git commit -am commit message`，其中的标志`-a`让Git在这个提交中包含所有修改过的文件，而标志`-m`让Git记录一条日志消息。

在❶处，执行命令`git status`，输出表明当前位于分支`master`，而工作树是干净（`clean`）的。每当要将项目推送到Heroku时，我们都希望看到这样的状态。

20.2.9 推送到Heroku

终于可以将项目推送到Heroku了。在活动的虚拟环境中，执行下面的命令：

```

❶ (ll_env)learning_log$ heroku login
heroku: Press any key to open up the browser to login or q to
exit:
Logging in... done
Logged in as eric@example.com
❷ (ll_env)learning_log$ heroku create
Creating app... done, ( secret-lowlands-82594
https://secret-lowlands-82594.herokuapp.com/ |
https://git.heroku.com/secret-lowlands-82594.git
❸ (ll_env)learning_log$ git push heroku master
--snip--
remote: -----> Launching...
remote: Released v5
❹ remote: https://secret-lowlands-82594.herokuapp.com/
deployed to Heroku
remote: Verifying deploy... done.
To https://git.heroku.com/secret-lowlands-82594.git
 * [new branch]      master -> master
(ll_env)learning_log$
```

首先执行命令heroku login，这将打开浏览器并在其中显示一个页面，让你能够登录Heroku（见❶）。然后，让Heroku创建一个空项目（见❷）。Heroku生成的项目名由两个单词和一串数字组成，但以后可修改这个名称。接下来，执行命令git push heroku master（见❸），让Git将项目的分支master推送到Heroku刚才创建的仓库中。Heroku将使用这些文件在其服务器上创建项目。❹处列出了用于访问这个项目的URL，但这个URL和项目名都是可以修改的。

执行这些命令后，项目就部署好了，但还未做全面配置。为核实正确地启动了服务器进程，请执行命令heroku ps：

```
(ll_env)learning_log$ heroku ps
❶ Free dyno hours quota remaining this month: 450h 44m (81%)
Free dyno usage for this app: 0h 0m (0%)
For more information on dyno sleeping and how to upgrade, see:
https://devcenter.heroku.com/articles/dyno-sleeping
❷ === web (Free): gunicorn learning_log.wsgi --log-file -
web.1: up 2019/02/19 23:40:12 -0900 (~ 10m ago)
(ll_env)learning_log$
```

输出指出了在接下来的一个月内，项目还可在多长时间内处于活动状态（见❶）。编写本书时，Heroku允许免费部署在一个月内最多有550小时处于活动状态。项目的活动时间超过这个限制后，将显示标准的服务器错误页面，我们稍后将定制这个错误页面。在❷处，我们发现启动了Procfile指定的进程。

现在，可使用命令heroku open 在浏览器中打开这个应用程序了：

```
(ll_env)learning_log$ heroku open
(ll_env)learning_log$
```

也可以启动浏览器并输入Heroku告诉你的URL，但上述命令让你无须这样做。你将看到“学习笔记”的主页，其样式设置正确无误，但还无法使用这个应用程序，因为尚未建立数据库。

注意 部署到Heroku的流程会不断变化。如果遇到无法解决问题，请通过查看Heroku文档来获取帮助。为此，可访问Heroku Dev Center网站首页，单击Python，再单击到Get Started with Python或Deploying Python and Django Apps on Heroku的链接。如果看不懂这些文档，请参阅附录C提供的建议。

20.2.10 在Heroku上建立数据库

为建立在线数据库，需要再次执行命令migrate，并应用在开发期间生成的所有迁移。要对Heroku项目执行Django和Python命令，可使用

命令heroku run 。下面演示了如何对Heroku部署执行命令migrate :

```
❶ (ll_env)learning_log$ heroku run python manage.py migrate
❷ Running 'python manage.py migrate' on ⚙ secret-lowlands-82594...
  up, run.3060
    --snip--
❸ Running migrations:
    --snip--
      Applying learning_logs.0001_initial... OK
      Applying learning_logs.0002_entry... OK
      Applying learning_logs.0003_topic_owner... OK
      Applying sessions.0001_initial... OK
(ll_env)learning_log$
```

首先执行命令heroku run python manage.py migrate（见❶）。Heroku随后创建一个终端会话来执行命令migrate（见❷）。在❸处，Django应用默认迁移以及我们在开发“学习笔记”期间生成的迁移。

现在如果访问这个部署的应用程序，将能够像在本地系统上一样使用它，但看不到在本地部署中输入的任何数据（包括超级用户账户），因为它们还没有被复制到在线服务器。通常，不将本地数据复制到在线部署中，因为本地数据通常是测试数据。

你可分享“学习笔记”的Heroku URL，让任何人都可使用它。下一节将再完成几个任务，以结束部署过程，让你能够继续开发“学习笔记”。

20.2.11 改进Heroku部署

本节将通过创建超级用户来改进部署，就像在本地一样。我们还将让这个项目更安全：将DEBUG设置为False，让用户在错误消息中看不到额外的信息，以防其利用这些信息来攻击服务器。

a. 在Heroku上创建超级用户

我们知道可以使用命令heroku run 来执行一次性命令，但也可这样执行命令：在连接到Heroku服务器的情况下，使用命令heroku run bash 打开Bash终端会话。Bash是众多Linux终端运行的语言。我们将使用Bash终端会话来创建超级用户，以便访问在线应用程序的管理网站：

```
(ll_env)learning_log$ heroku run bash
Running 'bash' on ⚙ secret-lowlands-82594... up, run.9858
❶ ~ $ ls
learning_log learning_logs manage.py Procfile
requirements.txt runtime.txt
staticfiles users
```

```
❶ ~ $ python manage.py createsuperuser
Username (leave blank to use 'u47318'): ll_admin
Email address:
Password:
Password (again):
Superuser created successfully.
❷ ~ $ exit
exit
(ll_env) learning_log$
```

在❶处，执行命令ls，以查看服务器上有哪些文件和目录。服务器包含的文件和目录应与本地系统相同。可像遍历其他文件系统一样遍历这个文件系统。

注意 即便使用的是Windows系统，也应使用这里列出的命令（如ls而不是dir），因为这里是在通过远程连接运行Linux终端。

在❷处，执行创建超级用户的命令，它像第18章在本地系统创建超级用户一样提示你输入相关的信息。在这个终端会话中创建超级用户后，执行命令exit返回到本地系统的终端会话（见❸）。

现在，你可以在在线应用程序的URL末尾添加/admin/来登录管理网站了。对我而言，这个URL为https://secret-lowlands-82594.herokuapp.com/admin/。

如果有其他人使用这个项目，别忘了你能访问他们的所有数据！千万别不把这一点当回事，否则用户就不会再将数据托付给你了。

β. 在Heroku上创建对用户友好的URL

你很可能希望URL更友好，比https://secret-lowlands-82594.herokuapp.com/admin/更好记。为此，只需使用一个命令重命名应用程序：

```
(ll_env) learning_log$ heroku apps:rename learning-log
Renaming secret-lowlands-82594 to learning-log-2e... done
https://learning-log.herokuapp.com/
https://git.heroku.com/learning-log.git
Git remote heroku updated
◆ Don't forget to update git remotes for all other local
checkouts of the app.
(ll_env) learning_log$
```

给应用程序命名时，可使用字母、数字和连字符，并且想怎么命名都可以，只要指定的名称未被别人使用就行。现在，项目的URL

变成了`https://learning-log.herokuapp.com/`。使用以前的URL再也无法访问它，命令`apps:rename` 将整个项目都移到了新的URL处。

注意 使用Heroku提供的免费服务部署项目时，如果项目在指定时间内未收到请求或过于活跃，Heroku将让项目进入休眠状态。用户访问处于休眠状态的网站时，加载时间将更长，但对于后续请求，服务器的响应速度将更快。这就是Heroku能够提供免费部署的原因所在。

20.2.12 确保项目的安全

当前，部署的项目存在严重的安全问题：`settings.py`包含设置`DEBUG=True`，指定在发生错误时显示调试信息。开发项目时，Django的错误页面显示了重要的调试信息，如果将项目部署到服务器后还保留这个设置，将给攻击者提供大量可利用的信息。

在在线项目中，我们将设置一个环境变量，以控制是否显示调试信息。**环境变量** 是在特定环境中设置的值。这是在服务器上存储敏感信息并将其与项目代码分开的方式之一。

下面来修改`settings.py`，使其在项目于Heroku上运行时检查一个环境变量：

settings.py

```
--snip--  
# Heroku设置  
import django_heroku  
django_heroku.settings(locals())  
  
if os.environ.get('DEBUG') == 'TRUE':  
    DEBUG = True  
elif os.environ.get('DEBUG') == 'FALSE':  
    DEBUG = False
```

方法`os.environ.get()` 从项目当前所处的环境中读取与特定环境变量相关联的值。如果设置了这个环境变量，就返回它的值；如果没有设置，就返回`None`。使用环境变量来存储布尔值时，必须小心应对，因为在大多数情况下，环境变量存储的都是字符串。请看下面在简单的Python终端会话中执行的代码片段：

```
>>> bool('False')  
True
```

字符串`'False'` 对应的布尔值为`True`，因为非空字符串对应的布尔值都为`True`。因此，我们将使用全大写的字符串`'TRUE'`。

和'FALSE'，以明确地指出存储的不是Python布尔值True 和False 。Django读取Heroku中键为'DEBUG' 的环境变量时，如果其值为'TRUE'，我们就将DEBUG 设置为True ；如果其值为'FALSE'，就将DEBUG 设置为False 。

20.2.13 提交并推送修改

现在需要将对settings.py所做的修改提交到Git仓库，再将修改推送 to Heroku。下面的终端会话演示了这个过程：

```
❶ (ll_env)learning_log$ git commit -am "Set DEBUG based on
environment variables."
[master 3427244] Set DEBUG based on environment variables.
 1 file changed, 4 insertions(+)
❷ (ll_env)learning_log$ git status
On branch master
nothing to commit, working tree clean
(ll_env)learning_log$
```

我们执行命令git commit，并指定一条简短而有描述性的提交消息（见❶）。别忘了，标志-am 让Git提交所有修改过的文件，并记录一条日志消息。Git找出唯一修改过的文件，并将所做的修改提交到仓库。

❷处显示的状态表明，当前位于仓库的分支master，没有任何未提交的修改。推送到Heroku前，必须检查状态并看到刚才所说的消息。如果没有看到这样的消息，就说明有未提交的修改，而这些修改将不会推送到服务器。在这种情况下，可尝试再次执行命令commit，但如果你不知道该如何解决这个问题，请阅读附录D，更深入地了解Git的用法。

下面将修改后的仓库推送到Heroku：

```
(ll_env)learning_log$ git push heroku master
remote: Building source:
remote:
remote: ----> Python app detected
remote: ----> Installing requirements with pip
--snip--
remote: ----> Launching...
remote: Released v6
remote: https://learning-log.herokuapp.com/ deployed to
Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/learning-log.git
 144f020..d5075a1 master -> master
(ll_env)learning_log$
```

Heroku发现仓库发生了变化，因此重建项目，确保所有的修改都生效。它不会重建数据库，因此这次无须执行命令migrate。

20.2.14 在Heroku上设置环境变量

现在可通过Heroku将settings.py中的DEBUG设置为所需的值了。

命令heroku config:set 设置一个环境变量：

```
(ll_env)learning_log$ heroku config:set DEBUG=FALSE
Setting DEBUG and restarting ⚡ learning-log... done, v7
DEBUG: FALSE
(ll_env)learning_log$
```

每当你在Heroku上设置环境变量时，Heroku都将重启项目，让环境变量生效。

要核实部署现在更安全了，请输入项目的URL，并在末尾加上未定义的路径，如尝试访问http://learning-log.herokuapp.com/letmein。你将看到通用的错误页面，它没有泄露任何有关该项目的具体信息。如果通过输入URL http://localhost:8000/letmein/向本地的“学习笔记”发出同样的请求，你将看到完整的Django错误页面。这样的结果非常理想：接着开发这个项目时，你可以看到信息丰富的错误消息，但访问在线项目的用户看不到有关代码的重要信息。

如果你在部署应用程序时遇到麻烦，需要排除故障，可执行命令heroku config:set DEBUG='TRUE'，以便访问在线项目时能够看到完整的错误报告。成功地排除故障后，务必将这个环境变量重置为'FALSE'。另外，请务必小心，一旦有用户经常访问这个在线项目，就不要这样做。

20.2.15 创建自定义错误页面

第19章对“学习笔记”进行了配置，使其在用户请求不属于自己的主题或条目时返回404错误。你可能还遇到过一些500错误（内部错误）。404错误通常意味着Django代码是正确的，但请求的对象不存在。500错误通常意味着代码有问题，如views.py中的函数有问题。当前，在这两种情况下，Django都返回通用的错误页面，但我们可以编写外观与“学习笔记”一致的404和500错误页面模板。这些模板必须放在根模板目录中。

a. 创建自定义模板

在最外层的文件夹learning_log中，新建一个文件夹，并将其命名为templates。然后在这个文件夹中新建一个名为404.html的文件（这个文件的路径应为learning_log/templates/404.html），并在其中输入如下内容：

404.html

```
{% extends "learning_logs/base.html" %}

{% block page_header %}
    <h2>The item you requested is not available. (404)</h2>
{% endblock page_header %}
```

这个简单的模板指定了通用的404错误页面包含的信息，但该页面的外观与网站其他部分一致。

再创建一个名为*500.html*的文件，并在其中输入如下代码：

500.html

```
{% extends "learning_logs/base.html" %}

{% block page_header %}
    <h2>There has been an internal error. (500)</h2>
{% endblock page_header %}
```

这些新文件要求对*settings.py*做细微的修改：

settings.py

```
--snip--
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        '--snip--'
    },
]
--snip--
```

这项修改让Django在根模板目录中查找错误页面模板。

β. 在本地查看错误页面

将项目推送到Heroku前，如果要在本地查看错误页面是什么样的，首先需要在本地设置中设置*Debug=False*，以禁止显示默认的Django调试页面。为此，可对*settings.py*做如下修改（请确保修改的是*settings.py*中用于本地环境的部分，而不是用于Heroku的部分）：

settings.py

```
--snip--  
# 安全警告：不要在在线环境中启用调试！  
DEBUG = False  
--snip--
```

现在，请求不属于你的主题或条目，以查看404错误页面。然后请求不存在的主题或条目，以查看500错误页面。例如，如果输入 `http://localhost:8000/topics/999/`，将出现500错误页面，除非你输入的主题已经超过了999个！

查看错误页面后，将本地DEBUG 的值重新设置为True ，为后续开发提供方便。（在管理Heroku设置的部分，确保处理DEBUG 的方式不变。）

注意 500错误页面不会显示任何有关当前用户的信息，因为发生服务器错误时，Django不会通过响应发送任何上下文信息。

γ. 将修改推送到Heroku

现在需要提交对错误页面所做的修改，并将这些修改推送到 Heroku：

```
❶ (11_env) learning_log$ git add .  
❷ (11_env) learning_log$ git commit -am "Added custom 404 and  
500 error pages."  
 3 files changed, 15 insertions(+), 10 deletions(-)  
  create mode 100644 templates/404.html  
  create mode 100644 templates/500.html  
❸ (11_env) learning_log$ git push heroku master  
--snip--  
remote: Verifying deploy.... done.  
To https://git.heroku.com/learning-log.git  
 d5075a1..4bd3b1c master -> master  
(11_env) learning_log$
```

在❶处，执行命令`git add .`，因为我们在项目中创建了一些新文件，需要让Git跟踪它们。然后，提交所做的修改（见❷），并将修改后的项目推送到Heroku（见❸）。

现在，错误页面出现时，其样式应该与网站其他部分一致。这样，在发生错误时，用户将不会感到别扭。

8. 使用方法`get_object_or_404()`

现在，如果用户手工请求不存在的主题或条目，将导致500错误。Django尝试渲染不存在的页面，但没有足够的信息来完成这项任务，进而引发了500错误。对于这种情形，将其视为404错误更合适。为此可使用Django快捷函数`get_object_or_404()`。这个函数尝试从数据库获取请求的对象，如果这个对象不存在，就引发404异常。我们在views.py中导入这个函数，并用它替换函数`get()`：

views.py

```
from django.shortcuts import render, redirect,
get_object_or_404
from django.contrib.auth.decorators import login_required
--snip--
@login_required
def topic(request, topic_id):
    """显示单个主题及其所有的条目。"""
    topic = get_object_or_404(Topic, id=topic_id)
    # 确定主题属于当前用户。
--snip--
```

现在，如果请求不存在的主题（如使用URL
`http://localhost:8000/topics/999/`），将看到404错误页面。
为部署这里所做的修改，再次提交，并将项目推送到Heroku。

20.2.16 继续开发

将项目“学习笔记”推送到服务器后，你可能想进一步开发它或开发要部署的其他项目。更新项目的过程几乎完全相同。

首先，对本地项目做必要的修改。如果在修改过程中创建了新文件，使用命令`git add .`（千万别忘记末尾的句点）将其加入Git仓库中。如果有修改要求迁移数据库，也需要执行这个命令，因为每个迁移都将生成新的迁移文件。

然后，使用命令`git commit -am "commit message"`将修改提交到仓库，再使用命令`git push heroku master`将修改推送到Heroku。如果在本地迁移了数据库，也需要迁移在线数据库。为此，可使用一次性命令`heroku run python manage.py migrate`，也可使用`heroku run bash`打开远程终端会话，并在其中执行命令`python manage.py migrate`。然后访问在线项目，确认期望看到的修改已生效。

在这个过程中很容易犯错，因此看到错误时不要太惊小怪。如果代码不能正确地工作，请重新审视所做的工作，尝试找出其中的错误。如果找不出错误，或者不知道如何撤销错误，请参阅附录C中有关如何寻

求帮助的建议。不要羞于去寻求帮助：每个学习开发项目的人都可能遇到过你面临的问题，因此总有人乐意伸出援手。通过解决遇到的每个问题，可让你的技能稳步提高，最终能够开发可靠而有意义的项目，还能解决别人遇到的问题。

20.2.17 设置`SECRET_KEY`

Django根据`settings.py`中设置`SECRET_KEY`的值来实现大量的安全协议。在这个项目中，提交到仓库的设置文件包含设置`SECRET_KEY`。对于一个练习项目而言，这足够了，但对于生产网站，应更细致地处理设置`SECRET_KEY`。如果你创建的项目的用途很重要，务必研究如何更安全地处理设置`SECRET_KEY`。

20.2.18 将项目从Heroku删除

一个不错的练习是，使用同一个项目或一系列小项目执行部署过程多次，直到对部署过程了如指掌。然而，你需要知道如何删除部署的项目。Heroku限制了可免费托管的项目数，而你也不希望自己的账户中包含大量练习项目。

在Heroku网站登录后，将重定向到一个页面，其中列出了你托管的所有项目。单击要删除的项目，你将看到另一个页面，其中显示了有关这个项目的信息。单击链接Settings，再向下滚动，找到用于删除项目的链接并单击它。这种操作是不可撤销的，因此Heroku让你手工输入要删除的项目的名称，确认你确实要删除它。

如果喜欢在终端中工作，也可使用命令`destroy` 来删除项目：

```
(ll_env)learning_log$ heroku apps:destroy --app appname
```

`appname` 是要删除的项目的名称，可能类似于`secret-lowlands-82594`，也可能类似于`learning-log`（如果你重命名了项目）。你将被要求再次输入项目名，确认你确实要删除它。

注意 删除Heroku上的项目对本地项目没有任何影响。如果没有使用你部署的项目，就尽管去练习部署过程好了，在Heroku上删除项目再重新部署完全合情合理。

动手试一试

练习20-3：在线博客 将你一直在开发的项目Blog部署到Heroku。确保将`DEBUG` 设置为`False`，以免出现错误时用户看到完整的Django错误页面。

练习20-4：在更多的情况下显示404错误页面 在视图函数 new_entry() 和edit_entry() 中，也使用函数 get_object_or_404()。完成这些修改后进行测试：输入类似于http://localhost:8000/new_entry/999/的URL，确认看到的是404错误页面。

练习20-5：扩展“学习笔记” 在“学习笔记”中添加一项功能，并将修改推送到在线部署。尝试做一项简单的修改，如在主页中对项目做更详细的描述。再尝试添加一项更高级的功能，如让用户能够将主题设置为公开的。为此，需要在模型Topic中添加一个名为public的属性（其默认值为False），并在new_topic页面中添加一个表单元素，让用户能够将私有主题改为公开。然后，需要迁移项目，并修改views.py，让未登录的用户也可以看到所有公开的主题。将修改推送到Heroku后，别忘了迁移在线数据库。