



Python web 开发与接口测试

---基于 Django 和 Requests

作者：虫师



前言

为什么学习本书:

是否想学习 web 开发而找不到很好的入门教程。

是否想做一个漂亮的网站出来炫耀。

是否很想知道当你点击一个按钮后，程序到底经过哪些过程把你想要的页面展现在你面前。

是否想告诉别人，不就是开发嘛，我也会。

是否想知道接口到底是什么，如何对它进行测试。

为什么是 Django?

我们总是很难去决定一件事情。比如，我到底该学 Java 呢还是 C# 呢？到底测试的工资高还是开发的高？我学习了 Python 之后，想学习 web 开发，是 Django 还是 Flask 好学、强大和灵活？

搞 Java 开发的老程序员会告诉你，他只需要几周时间就可以学会用 C# 开发程序。反之也是一样。一直用 Flask 写 web 的应用同学有一天突然想换成 Django 来开发 web 应用，结果看了一下 Django 的文档就开始写代码了。

好吧！选择总会有理由，我的理由只是因为 Django 资料更丰富，遇到了问题更容易找到答案。

为什么是接口测试？

接口常被开发人员挂在嘴边，在开发过程中无处不在，但对于测试人员来说，它又如此朦胧，无形无色无味，难以触碰。但它到底是什么？如何对它进行测试？一直是困扰许多测试新手的问题。之所以看不清接口是什么，主要是因为我们不了解应用是如何被开发出来的。

所以，对于想学接口测试的同学，我都建议他们很学习一下 web 开发，当然，我们目的不是想抢程序员的饭碗，如果，你愿意，在学完本书后也未尝不可。

时间在哪里？

不要总是想着，等我有时间了可以去好好的学学 xx，时间是挤出来的。不要只停留在想上面，从现在开始动手开始学习吧。

第一章 Django 入门

Django 是在 BSD 许可证下的开源项目。我们建议使用 Python3 的最新版本，但你也可以使用 Python2.7。

在 Python 的哪些版本下可以使用 Django：

Django version	Python versions
1.8	2.7, 3.2 (until the end of 2016), 3.3, 3.4, 3.5
1.9, 1.10	2.7, 3.4, 3.5

1.1 Django 开发环境搭建

Django 就基于 Python 的 web 框架，所以，要想使用它的前提是安装 Python。当然，在此之前，我们又将面临另一个选择，到底是选择 Python2.x 还是 Python3.x 的问题？既然 Django 官网已经告诉我们，它已经支持了 Python3.x 了，那我们就用 Python3.x 吧！

1.1.1 安装 Python

访问 Python 官方网站：<https://www.Python.org/>

找到下载页面下载最新版本的 Python3，目前最新版本为 Python3.5。读者可根据自己的平台选择相应的版本进行下载。对于 Windows 用户来说，如果 32 位系统是则选择 x86 版本；如果是 64 位系统，则选择 64 版本。下载完成后会得到一个以 .msi 为后缀名的文件，双击进行安装，如图 2.1 所示。



图 1.1 Python 安装界面

安装过程与一般的 Windows 程序类似。安装完成，将在开始菜单中看到安装好的 Python 目录，如图 2.2 所示。

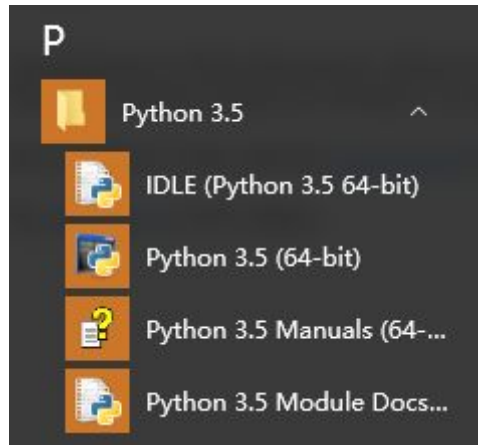


图 1.2 Python 目录

打开 Python 自带的 IDLE，就可以编写 Python 程序了，Python Shell 界面。如图 2.3 所示。

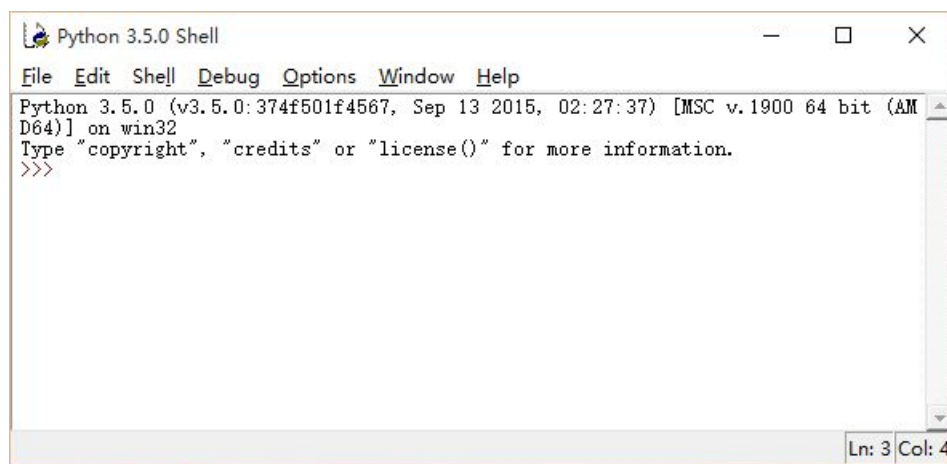


图 1.3 Python Shell 界面

或者通过在 Windows 命令提示符下输入“python”命令，也可以进入 Python Shell 模式，如图 2.4 所示。

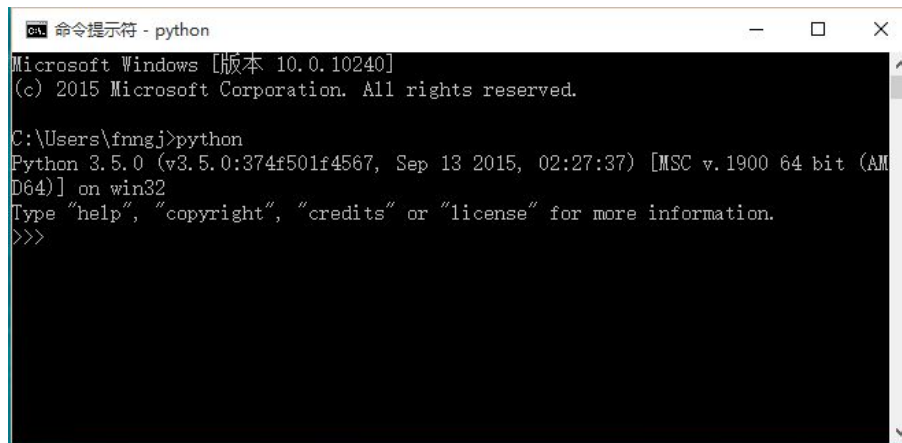


图 1.4 命令提示符下进入 Python Shell 界面

小提示 如果提示 Python 不是内部或外部命令！别急，把 Python 的安装目录添加到系统环境变量的 Path 下面。右击桌面“我的电脑”，打开右键菜单，在属性→高级→环境变量→系统变量的 Path 中添加：

变量名：PATH

变量值：;C:\Python35

也可以在图 2.1 所示 Python 安装界面，勾选 Add Python 3.5 to PATH 复选框，在安装完成后自动完成 PATH 配置工作。

1.1.2 安装 Django

访问 Django 官方网站：<https://www.djangoproject.com/>

在 Django 的下载页面告诉我们两种安装 Django 的方式。

第二种安装方法，通过 pip 安装。pip 是一个安装和管理 Python 包的工具，通过 pip 来安装 Python 包将变得十分简单，我们将省去搜索→查找版本→下载→安装等烦琐的过程。

通过 CMD 命令打开 Windows 提示符，输入 pip 或 php3 命令。

cmd.exe

C:\Users\fnngj>pip

Usage:

pip <command> [options]

Commands:

install Install packages.

uninstall Uninstall packages.

```
freeze          Output installed packages in requirements format.
list            List installed packages.
show           Show information about installed packages.
search         Search PyPI for packages.
wheel          Build wheels from your requirements.
zip            DEPRECATED. Zip individual packages.
unzip          DEPRECATED. Unzip individual packages.
bundle         DEPRECATED. Create pybundles.
help           Show help for commands.
```

.....

如果出现 `pip` 命令的说明信息，则说明我们已经安装成功。如果提示 `pip` 不是内部或外部命令，则可以手动将 `C:\Python35\Scripts\` 目录添加到系统环境变量下的 `Path` 下面，重新打开 `cmd` 命令行验证。

通过 `pip` 命令安装 `django`。

cmd.exe

```
C:\Users\fnngj>pip install django    #安装 django
```

.....

```
C:\Users\fnngj>python3 -m pip install django    #或者通过这种方式安装 django
```

.....

`pip` 下面包含了很多命令，正如我们前面只输入一个有 `pip` 后回车所得到的提示。

1.1.2 Ubuntu 下安装 Django

Linux 操作系统的版本很多，这里以流行的 Ubuntu 系统为例，介绍在其下面的安装过程。

因为 Ubuntu 系统本身对 Python 有很强的依赖，所以 Ubuntu 自带的就有 Python。

当前的在 Ubuntu 系统中已经同时集成了 Python2 与 Python3，打开终端，输入 “python” 或 “Python3” 命令回车，即可进入相应版本的 Python Shell 模式。

ubuntu 终端

```
fnngj@fnngj-PC:~$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
```

```
fnngj@fnngj-PC:~$ python3
Python 3.4.3 (default, Jul 28 2015, 18:20:59)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
```

下面我们在 Ubuntu 的 Python3 下安装 Django。

ubuntu 终端

```
fnngj@fnngj-PC:~$ python3 -m pip install django
```

1.2 快速搭建一个 blog

按照惯例，本书应该开始大谈 Django 的特点和架构。但理论的东西是我们在用了之后的理解和总结。所以，抛开这些内容，跟着我动手来搭建一个 blog，并体会 Django 开发 web 应用的套路。

本例操作的环境：

```
=====
windows 10
python 3.5
Django 1.9.1
=====
```

为尽量避免你在操作过程中遇到问题，而又无从解决，从而造成挫败感，请尽量与我的环境保持一致（尤其是 Django 版本）。

我们在 D 盘下创建一下 pydj 目录，用于存放我们的项目练习。当然你可以选择任意目录。

1.2.1、创建项目与应用

创建项目和应用：

```
cmd.exe
```

```
D:\pydj>django-admin startproject myweb    #创建 myweb 项目
D:\pydj>cd myweb        #进入项目目录
D:\pydj\myweb>python3 manage.py startapp blog #创建 blog 应用
```

经过上面的这几步操作之后，你在一好奇在 `pydj` 目录下产生了什么？

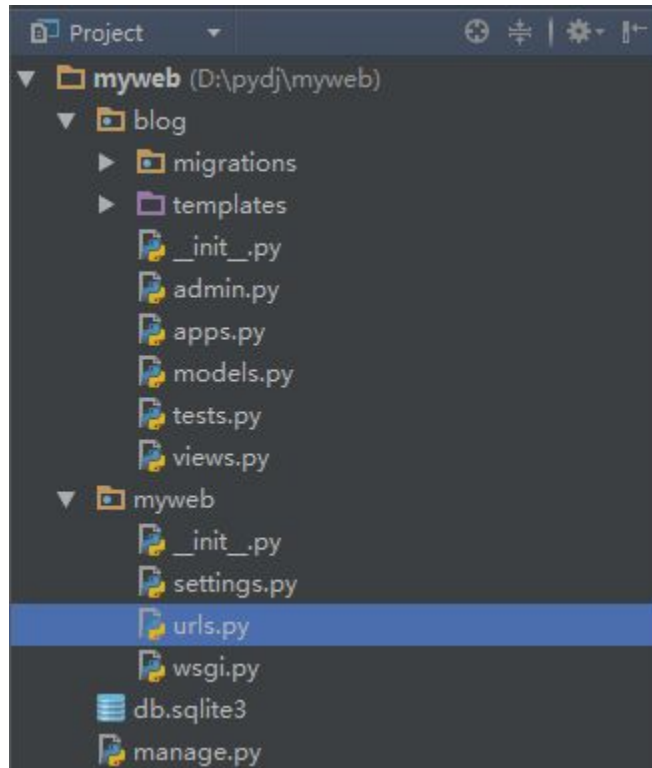


图 1.5 Django 项目目录结构

如图 1.5，Django 项目的目录结构（截图是通过 PyCharm 编辑器打开）。

`.../myweb/myweb/`目录下的 `settings.py` 文件为整个项目的配置文件，例如，应用模块的添加，数据库的连接与配置，模板静态目录的配置等。

下面添加 `blog` 应用到项目，找到文件第 39 行的位置，添加 `blog` 应用：

```
.....
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',    #admin 应用
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
```



```
'django.contrib.staticfiles',
'blog',    # 添加 blog 应用
```

这个“blog”正是我们执行“python manage.py startapp blog”命令所创建的应用。

1.2.2、admin 后台

Django 中一个最强大的部是自带了一个 admin 后台，在模型中读取数据库数据来提供一个强大的接口，使内容提供者能立即用它向站点中添加内容，可以让我们非常方便的操作数据库的数据。

首先需要进行数据库同步：

cmd.exe

```
D:\pydj\myweb>python3 manage.py migrate
Operations to perform:
  Apply all migrations: contenttypes, sessions, auth, admin
Running migrations:
  Rendering model states... DONE
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying sessions.0001_initial... OK
```

“migrate”查看 myweb/settings.py 文件中的 INSTALLED_APPS 设置并根据数据库设置创建任何必要的数据库表，数据库的迁移还会跟踪应用的变化。

首先，我们需要创建一个能够登录管理站点的用户。运行如下命令：

cmd.exe

```
D:\pydj\myweb>python3 manage.py createsuperuser
Username (leave blank to use 'fnngj'): fnngj    #输入登录用户名
Email address: fnngj@126.com    #输入用户邮箱
Password:    #输入登录密码
Password (again):    #再次输入用户密码
Superuser created successfully.
```

Django 对密码做了加强的要求，至少需要 8 个字符，并且不能为全数字。

现在可以去登录 **admin** 后台了，不过，在此之前需要先把服务启动起来。运行以下命令启动服务：

cmd.exe

```
D:\pydj\myweb>python3 manage.py runserver # 运行 server 服务
Performing system checks...
System check identified no issues (0 silenced).
January 05, 2016 - 22:30:38
Django version 1.9.1, using settings 'myweb.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

通过浏览器访问：<http://127.0.0.1:8000/>

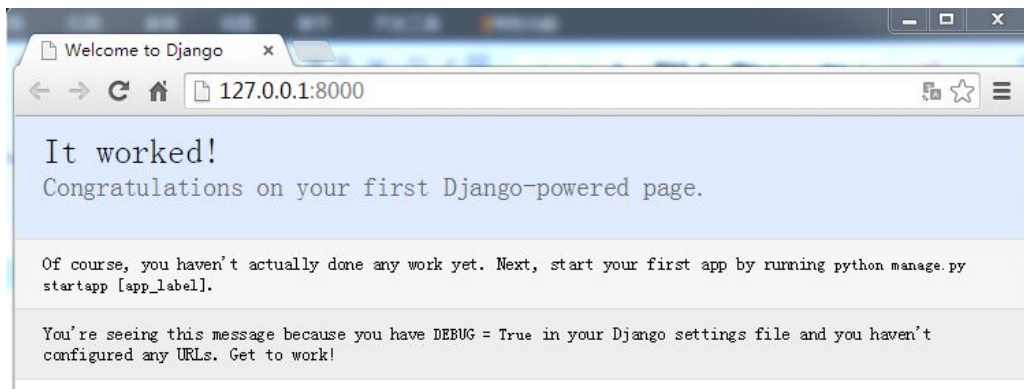


图 1.6 Django 默认首页

如果看到如图 1.6 页面，说明 Django 已经可以工作了。

接着访问 admin 后台：<http://127.0.0.1:8000/admin>

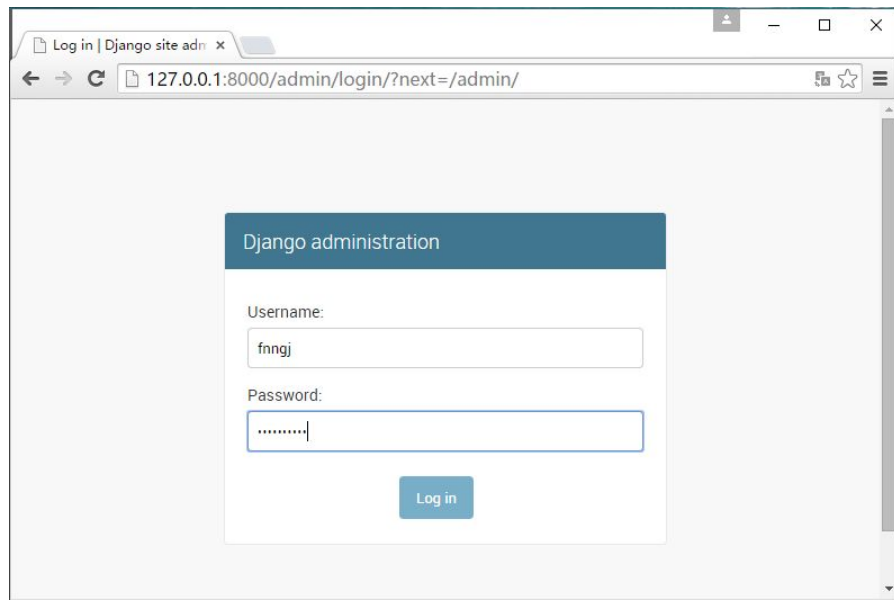


图 1.7 admin 管理后台登录

还记得前面通过“createsuperuser”命令所创建的用户名和密码么？用创建的用户名登录 admin。

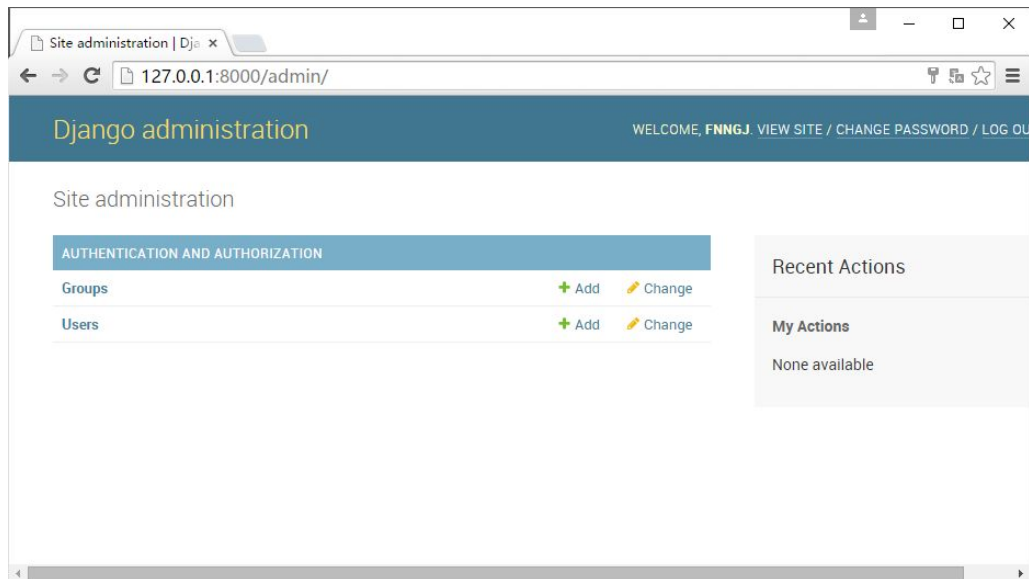


图 1.8 admin 管理后台界面

1.2.3、创建 blog 模型

Django 提供了一个抽象层（模型），对您的 Web 应用中的数据进行构建及操作。

我们在创建一篇 blog 时是怎样的，输入标题和编写正文，对吧！那我们来创建一个存放标题和正文的数据库表吧。

打开 myweb/blog/models.py 文件，编写如下代码。

```
from django.db import models
from django.contrib import admin

# Create your models here.
class Blog(models.Model):
    title = models.CharField(max_length=150)
    body = models.TextField()
    timestamp = models.DateTimeField()

class BlogAdmin(admin.ModelAdmin):
    list_display = ('title', 'timestamp')

admin.site.register(Blog, BlogAdmin)
```

通过模型的方式创建表，表名为 **Blog**，字段分别为 **title**（char 类型，最长 150 个字符），**body**（text 文本类型）和 **timestamp**（datetime 日期时间类型）。

创建对应的数据库表，执行以下命令：

cmd.exe

```
D:\pydj\myweb>python3 manage.py makemigrations blog
Migrations for 'blog':
  0001_initial.py:
  - Create model Blog

D:\pydj\myweb>python3 manage.py migrate
Operations to perform:
  Apply all migrations: auth, contenttypes, blog, sessions, admin
Running migrations:
  Rendering model states... DONE
  Applying blog.0001_initial... OK
```

注意，在执行完数据库的创建后，需要重新执行“migrate”命令。

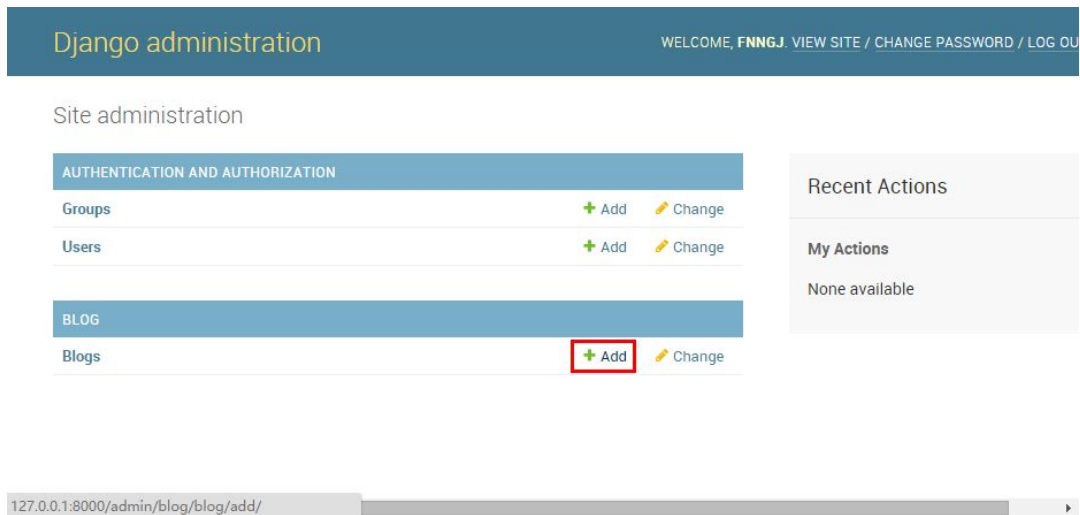


图 1.9 查看重新创建的 blog 表

如图 1.9，点击“add”按钮，添加一篇 Blog。

The image shows the 'Add blog' form in the Django administration interface. The breadcrumb trail at the top is 'Home > Blog > Blogs > Add blog'. The form has two main sections: 'Title:' with a text input field containing 'hello Django', and 'Body:' with a large text area containing the text '这是我们第一次用Django开发blog应用，跟着虫师一步一步的创建的自己的blog吧！'. Below these, there's a 'Timestamp:' section with 'Date:' set to '2016-01-05' and 'Time:' set to '23:03:52'. At the bottom, there are three buttons: 'Save and add another', 'Save and continue editing', and 'SAVE'.

图 1.10 添加一篇 blog

如图 1.10，添加一篇 Blog，并点击“Save”，那么就可以看到我们的博客列表了，如图 1.11。

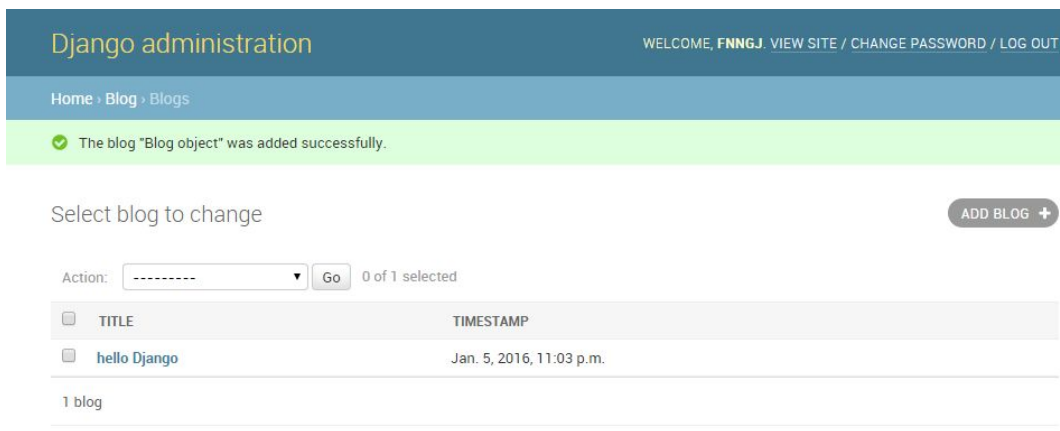


图 1.11 blog 列表

1.2.4、将 blog 展示到网页上

现在 blog 后台已经写好了，那么如何把 blog 显示到前端页面上呢？接下来完成剩下的工作。

创建模板

在../myweb/blog/目录下创建 templates 目录，该目录用来存放 web 页面，在 templates 目录下创建 index.html 文件：

```
{% for blog in blogs %}
<h2>{{ blog.title }}</h2>
<p>{{ blog.timestamp }}</p>
<p>{{ blog.body }}</p>
{% endfor %}
```

使用 Django 自带的模板语言向页面中循环输出每一篇 blog。

编写视图

打开.../myweb/blog/views.py 文件：

```
from blog.models import Blog
from django.shortcuts import render_to_response

# Create your views here.
def index(request):
```

```
blog_list = Blog.objects.all()
return render_to_response('index.html', {'blogs':blog_list})
```

通过 `Blog.objects.all()` 得到 `blog` 表中的所有数据，并赋值给 `blog_list` 变量。最终将 `blog_list` 发送到 `index.html` 页面。

配置 url

打开.../myweb/myweb/urls.py

```
from django.conf.urls import url
from django.contrib import admin

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

浏览器访问: <http://127.0.0.1:8000/index/>



图 1.12 blog 首页

1.2.5、给 blog 加点样式

到此，我们的 **blog** 已经搭建完成，从后台添加 **blog**，前端页面就可以显示新添加的 **blog**。但是，这太丑了！那就稍微加点样式吧。

创建基础模板

在.../myweb/blog/templates 目录里创建 **base.html** 的模板：

```
<html><head>
  <style type="text/css">
    body{color:#0A0A0A;background:#FAFAFA;padding:0 5em;margin:0}
    h1{padding:2em 1em;background:#7171C6}
    h2{color:#8B1A1A;border-top:1px dotted #8B2500;margin-top:2em}
    p{margin:1em 0}
  </style>
</head>
<body>
  <h1>Django blog</h1>
  {% block content %}
  {% endblock %}
</body>
</html>
```

修改 **index.html** 模板，让它引用 **base.html** 模板。

```
{% extends "base.html" %}
{% block content %}

  {% for blog in blogs %}
    <h2>{{ blog.title }}</h2>
    <p>{{ blog.timestamp }}</p>
    <p>{{ blog.body }}</p>
  {% endfor %}
{% endblock %}
```

再次刷新页面，你将看到下面的效果：



图 1.13 加了样式的 blog 首页

好了，现在可以登录 admin 后台继续添加文章了。

1.3 django workflow

blog 我们博客已经开发完成了，下面来梳理一下 Django 的开发流程。

一张流程图告诉你，django 的处理流程：

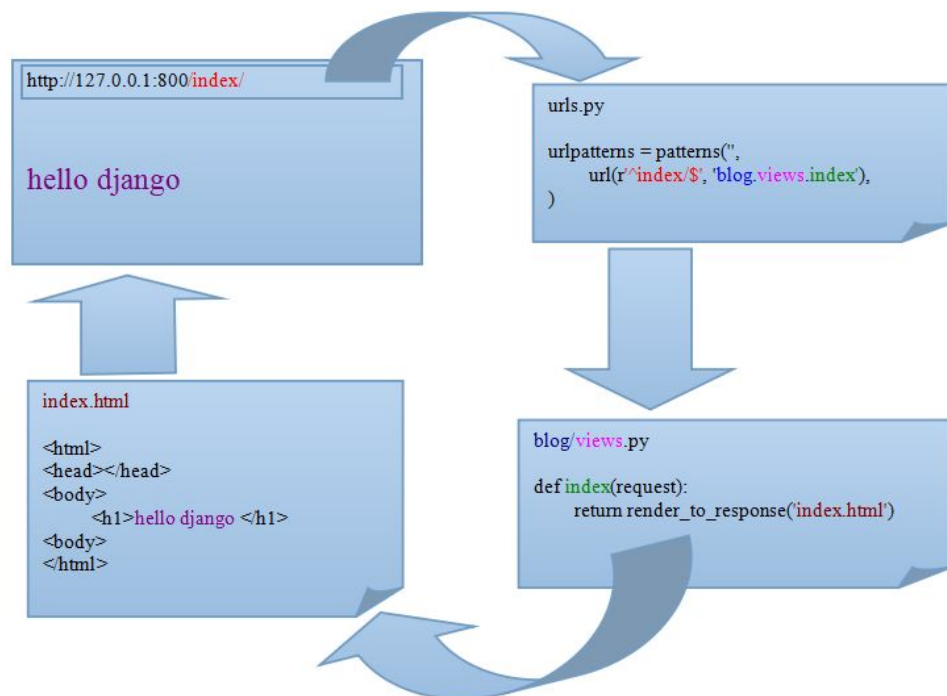


图 1.14 Django 开发流

1.3.1、URL 组成

作为用户，我们首先在浏览器的输入框内输入：<http://127.0.0.1:8000/index/>

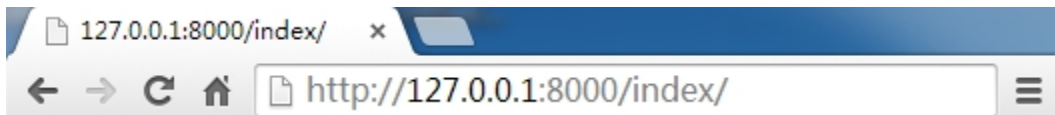


图 1.15 URL 地址

URL 地址由以下几部分组成：

协议类型： HTTP ， FTP

HTTP 协议（HyperText Transfer Protocol，超文本传输协议）是用于从 WWW 服务器传输超文本到本地浏览器的传送协议。它可以使浏览器更加高效，使网络传输减少。它不仅保证计算机正确快速地传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示等。

HTTPS（全称：Hyper Text Transfer Protocol over Secure Socket Layer），是以安全为目标的 HTTP 通道，简单讲是 HTTP 的安全版。

主机地址： itest.info ， 127.0.0.1

itest.info 为一个网址，网址通过域名解析服务器会找到对应的 IP 主机。

127.0.0.1 为一个 IP 地址，不过，该 IP 地址比较特殊，用来指向的本机。

有同学一定见过在浏览器中输入 <http://localhost> 就可以访问到本机。这是因为在 Windows 下的 hosts 文件中做了指向。

打开本 C:\Windows\System32\drivers\etc\ 目录下的 hosts 文件（可以用记事本打开）。

127.0.0.1 localhost

因为我们在 hosts 文件里设置的 localhost 与 127.0.0.1 的对应关系，所以在浏览器输入框输入 localhost 时，指向的就是本机的 IP 地址。

端口号： 8000

一台主机上有很多应用，不同的应用占用不同的端口号，除了要指定主机（网址或 IP 地址）之外，还要进一步指定相应的端口号才能指到具体的应用。

前面我们在运行 Django 服务器，默认使用 8000 的端口号，所以，在浏览器除了输入 IP 地址之后，还要指向端口号，才能找到相应的应用。

路径： /index/ 、 /admin

一般用来表示主机上的一个目录或文件地址。

1.3.2、urls 的配置

当 Django 拿到浏览器 URL 有地址之后，取端口号后面的路径 “/index” 、 “/admin” 。然后在 urls.py 文件中匹配。

```
from django.conf.urls import url
from django.contrib import admin

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

`r'^index/$'`

这里使用了 python 的正则表达式。

匹配符	含义
r	字符串有前面加 “ r ” 是为了防止字符串中出现类似 “\t” 字符时被转义。
^	匹配字符串开头；在多行模式中匹配每一行的开头。 ^abc abc
\$	匹配字符串末尾；在多行模式中匹配每一行末尾。 abc\$ abc

结果`^index/$` 可以对这个文件路径进行匹配。那么将指向 `blog.views.index` 这个地址。

`blog` 是文件夹，`views` 是试图文件，`index` 则文件中的函数名。

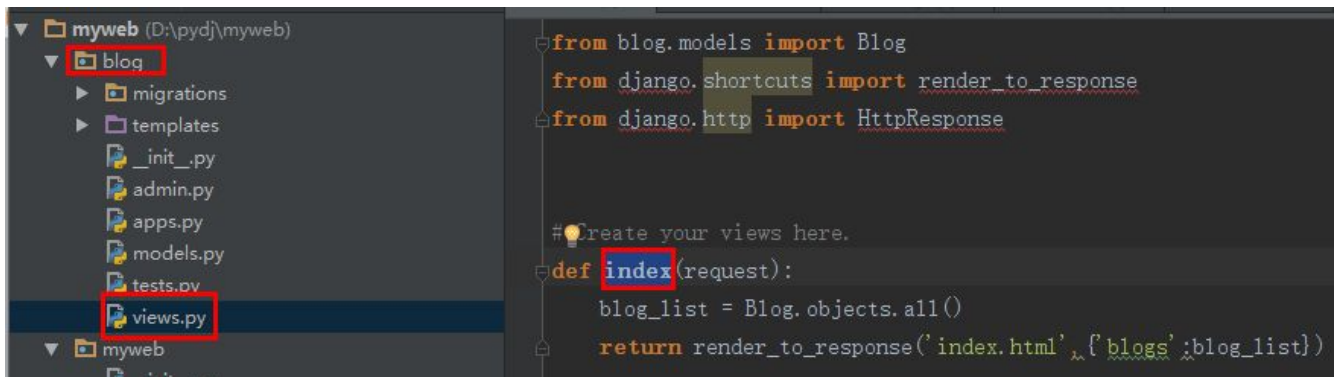


图 1.16 urls.py 配置的路径

1.3.3、views 视图

../blog/views.py

```
from blog.models import Blog
from django.shortcuts import render_to_response

# Create your views here.
def index(request):
    blog_list = Blog.objects.all()
    return render_to_response('index.html', {'blogs':blog_list})
```

index()函数做了两件事儿:

```
blog_list = BlogPost.objects.all()
```

查询到 BlogPost 数据库里的所有数据，赋值给 blog_list 变量。

```
return render_to_response('index.html', {'blog_list':blog_list})
```

通过 render_to_response() 返回给浏览器一个 index.html 页面，并且将 blog_list 变量的值也返回给 index.html。

1.3.4、templates 模板

模板就是我们所熟悉的页面了，django 自带的有模板系统。它的主要作用是如何展示数据，比如视图返回了一堆数据过来。是都循环显示出来呢？还通过判断只显示你认为有用的呢？当然，这里为了使页面更漂亮需要借助前端技术，比如 css、JavaScript、bootstrap 等。

然后，我们就在浏览器上看到了 index.html 页面了：



图 1.17 Django 模板

1.4 MTV 开发模式

了解了 django 的组成部分之间，我们再来深入的探讨一下 django 的开发模式。

MTV 开发模式

在钻研更多代码之前，让我们先花点时间考虑下 Django 数据驱动 Web 应用的总体设计。我们在前面章节提到过，Django 的设计鼓励松耦合及对应用程序中不同部分的严格分割。遵循这个理念的话，要想修改应用的某部分而不影响其它部分就比较容易了。在视图函数中，我们已经讨论了通过模板系统把业务逻辑和表现逻辑分隔开的重要性。在数据库层中，我们对数据访问逻辑也应用了同样的理念。把数据存取逻辑、业务逻辑和表现逻辑组合在一起的概念有时被称为软件架构的 Model-View-Controller(MVC)模式。在这个模式中，Model 代表数据存取层，View 代表的是系统中选择显示什么和怎么显示的部分，Controller 指的是系统中根据用户输入并视需要访问模型，以决定使用哪个视图的那部分。

为什么用缩写？

像 MVC 这样的明确定义模式的主要用于改善开发人员之间的沟通。比起告诉同事，“让我们采用抽象的数据存取方式，然后单独划分一层来显示数据，并且在中间加上一个控制它的层”，一个通用的说法会让你收益，你只需要说：“我们在这里使用 MVC 模式吧。”。Django 紧紧地遵循这种 MVC 模式，可以称得上是一种 MVC 框架。以下是 Django 中 M、V 和 C 各自的含义：

M，数据存取部分，由 django 数据库层处理，本章要讲述的内容。

V，选择显示哪些数据要显示以及怎样显示的部分，由视图和模板处理。

C，根据用户输入委派视图的部分，由 Django 框架根据 URLconf 设置，对给定 URL 调用适当的 Python 函数。

由于 C 由框架自行处理，而 Django 里更关注的是模型（Model）、模板(Template)和视图（Views），Django 也被称为 MTV 框架。在 MTV 开发模式中：

M 代表模型（Model），即数据存取层。该层处理与数据相关的所有事务：如何存取、如何验证有效

T 代表模板(Template)，即表现层。该层处理与表现相关的决定：如何在页面或其他类型文档中进行显示。

V 代表视图（View），即业务逻辑层。该层包含存取模型及调取恰当模板的相关逻辑。你可以把它看作模型与模板之间的桥梁。

如果你熟悉其它的 MVC Web 开发框架，比方说 Ruby on Rails，你可能会认为 Django 视图是控制器，而 Django 模板是视图。很不幸，这是对 MVC 不同诠释所引起的错误认识。在 Django 对 MVC 的诠释中，视图用来描述要展现给用户的数据；不是数据 如何展现,而且展现 哪些 数据。相比之下，Ruby on Rails 及一些同类框架提倡控制器负责决定向用户展现哪些数据，而视图则仅决定 如何展现数据，而不是展现 哪些 数据。

两种诠释中没有哪个更加正确一些。重要的是要理解底层概念。

第二章 Django 视图

问题驱动学习，不仅可以充分调用我们的兴趣，也可以锻炼我们解决问题的思路与方法。这一章，我们继续以开发具体功能为导向进行 Django 的学习。

2.1 来写个登录

你现在一定跃跃欲试的想用 Django 写点常见的功能，比如用户的登录、注册等。这次，我们先从前端页面写起。（注：此处在第一章 1.2 小节的基础上开发。）

打开.../blog/templates 目录，修改 index.html 文件，添加如下代码。

```
.....  
<h2>login</h2>  
<form>
```

```

<input name="username" type="text" placeholder="username">
<input name="password" type="password" placeholder="password">
<button id="btn" type="submit" >login</button>
</form>
.....

```

启动 Django 服务，访问：<http://127.0.0.1:8000/index/>

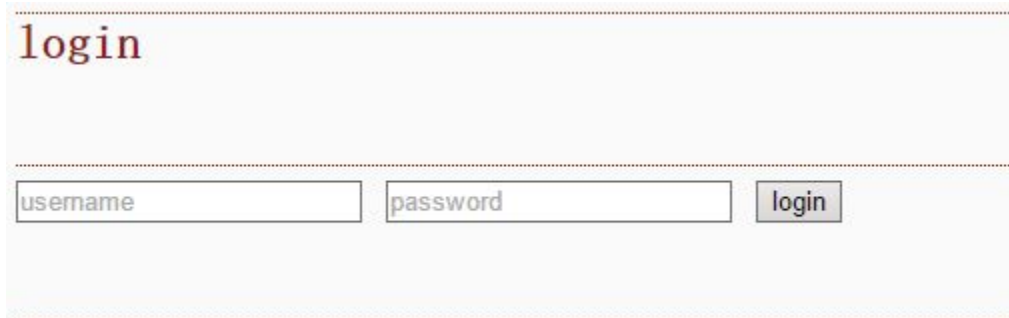


图 2.1 登录功能

虽然在页面上已经看到了一个登录功能，但当前还并没有真正的实现登录功能。真正的要实现登录我还需要考虑以一些问题。当点击“login”按钮时，表单（form）要以什么方式提交（get 还是 post）？后台如何校验用户名密码是否成功，如果成功应该跳到什么页面？失败如何提示用户？

2.1.1、get 与 post 请求

常见的两种 HTTP 请求方法：GET 和 POST

在客户机和服务器之间进行请求-响应时，两种最常被用到的方法是：GET 和 POST。

- GET - 从指定的资源请求数据。
- POST - 向指定的资源提交要被处理的数据

先来看看 get 方法是如何传参数，给 form 添加属性 `method="get"`。

```

<form method="get" >

```

然后保存 index.html 文件，刷新页面。输入用户名、密码，点击保存。

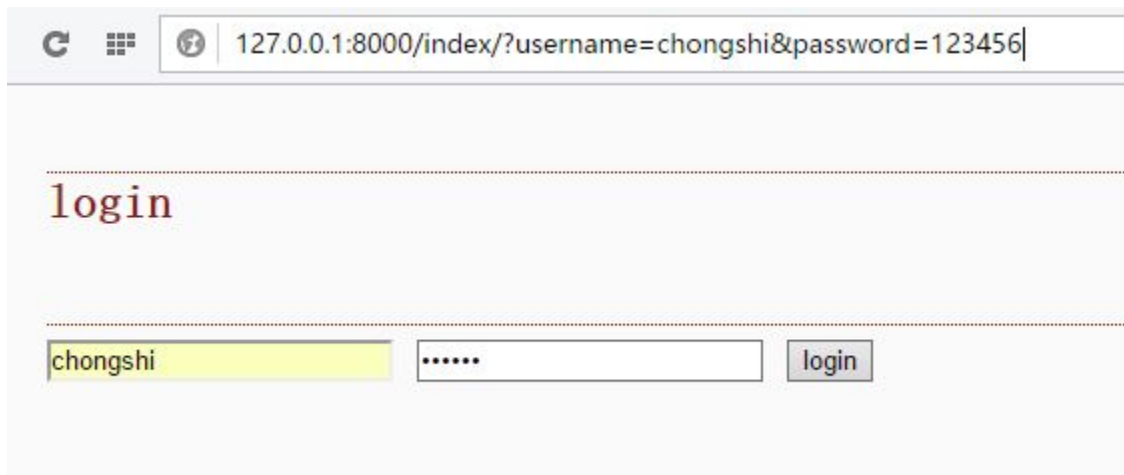


图 2.2 get 请求

<http://127.0.0.1:8000/index/?user=chongshi&passwd=123456>

get 传参方式，会将提交的数据跟在 URL 地址的后面，地址后面跟“？”，user 和 passwd 为输入的 name 属性值，[user=chongshi](#) 表示用户名输入框所接收到的信息为 chongshi。密码框 [passwd=123456](#) 同理。不同参数用“&”符号隔开。

同样是上面的代码我们将 form 的属性值换成 `method="post"`。再次刷新页面，输入用户名密码，点击登录。

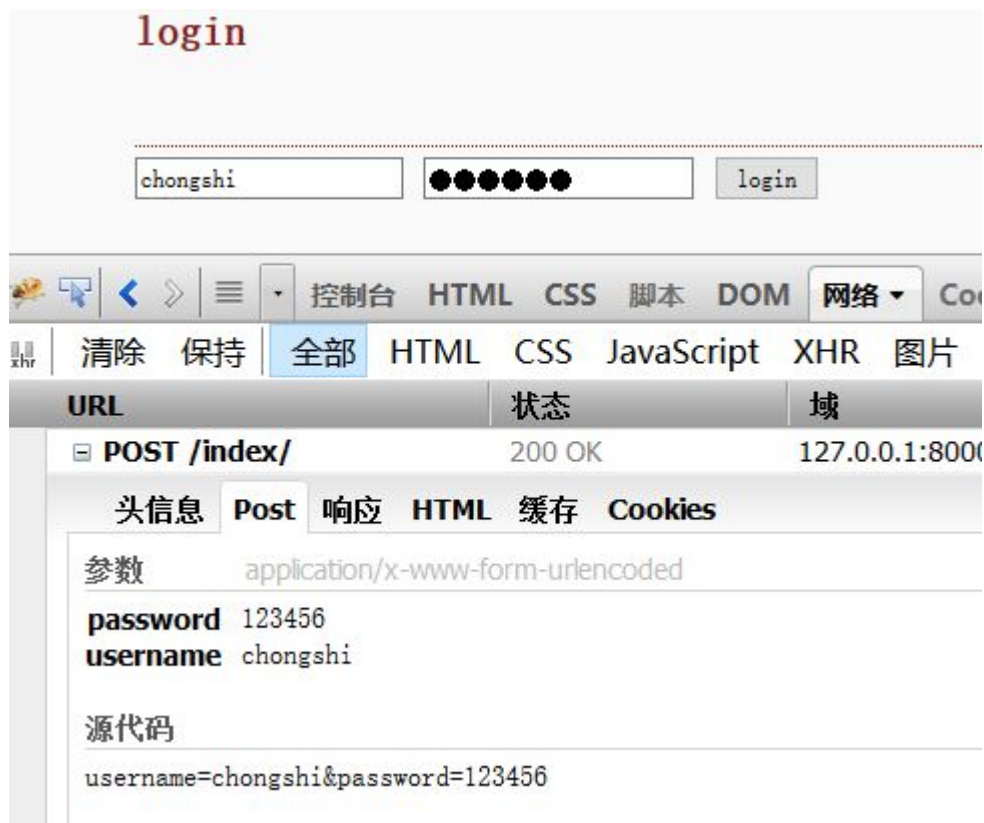


图 2.3 通过 FireBug 查看 post 请求

这一次的提交借助 firebug 前端调试工具进行查看。我们发起的是一个 post 请求，提交的数据将做为一个单独的请求包进行发送。

2.1.2、from 表单提交

现在知道了如何通过 get 或 post 将表单中的数据提交给服务器，那么提交服务器的哪里呢？我们在访问 blog 页面时（<http://127.0.0.1:8000/index/>）所以传的 URL 之所以能正确的解析，最后重要的是尾部的文件目录（/index/），那么 Django 拿到这个 URL 之后通过 urls.py 文件进行匹配解析。那么表单如何给它指定一个文件目录呢？

```
<form method="post" action="/login/">
```

form 表单有一个 action 属性可以用于指定文件目录。我这里指向一个/login/目录。

urls 配置

打开 myweb/urls.py 文件添加一条配置。

```
from django.conf.urls import url
from django.contrib import admin
from blog.views import index, login

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

此处为 Django1.9.1 的新写法，我们可以将视图路径导入，直接指定视图的函数名。

2.1.3、创建视图

在 urls.py 文件中，login/的路径指向的 views.py 文件中的 login 函数，那我们现在就来创建 login 函数。

```
from blog.models import Blog
from django.shortcuts import render_to_response
from django.http import HttpResponseRedirect

# Create your views here.
def index(request):
    blog_list = Blog.objects.all()
    return render_to_response('index.html', {'blogs':blog_list})

def login(request):
    username = request.POST.get('username', '')
    password = request.POST.get('password', '')
    if username == 'chongshi' and password == '123456':
        return HttpResponseRedirect('login success!')
    else:
        return render_to_response('index.html', {'error':'username or password error!'})
```

创建 login 的函数虽然也只有几行代码，但这它却有不少知识点需要我们了解。

```
def login(request):
    username = request.POST.get('username', '')
    password = request.POST.get('password', '')
```

定义一个 login() 函数，这个函数需要接收一个参数 request，这个参数就是通过前端所传过来的请求（数据）。寻找用户名为 'username' 的 POST 参数，而且如果参数没有提交，返回一个空的字符串("")。寻找密码为 'password' 的 POST 参数，而且如果参数没有提交，返回一个空的字符串("")。

```
<form method="post" action="/login/">
    <input name="username" type="text" placeholder="username">
    <input name="password" type="password" placeholder="password">
    <button id="btn" type="submit">login</button>
</form>
```

通过前端页面上的输入框的 name 属性值用来区分不同输入框传来的值。

那么如果 from 表单用的是 get 方法（method="get"）呢？那在视图函数中用 GET 方法去接收就可以了。

```
username = request.GET.get('username', '')
password = request.GET.get('password', '')
```

从前端获取到用户名和密码信息之后，接下来的处理是这样的：

```
if username == 'chongshi' and password == '123456':
    return HttpResponse('login success!')
else:
    return render_to_response('index.html', {'error': 'username or password error!'})
```

我们做了一个简单的判断，如果 username == 'chongshi' 并且 password == '123456' 的话，调用 HttpResponse 对象，返回前端页面一个字符串('login success!')。如果不相等，则返回 index.html 页面上一个提示信息“username or password error!”。

HttpResponse 对象：

HttpResponse 类存在于 `django.http.HttpResponse` 中，以字符串的形式传递给前端页面数据。用前记得先将包导入。

```
from django.http import HttpResponse
```



图 2.4 登录成功提示

如图 2.4，这就是我们输入正确的用户名密码所得到的页面信息。

如果不满足前面的条件将调用 `render_to_response()` 函数。

render_to_response 函数：

`render_to_response` 函数包含两个参数，第一个参数是一个 `html` 页面，它会去当前应用下面的 `templates/` 目录查找有没有一个叫 `index.html` 的页面。

第二个参数默认可以为空。它是一个字典（`{'error': 'username or password error!'}`），如假如用户输入了错误的用户名和密码，那么我们需要告诉他。

我们需要将这个信息显示在什么位置，那么需要修改前端代码。

```
<form method="post" action="/login/">
  <input name="username" type="text" placeholder="username">
  <input name="password" type="password" placeholder="password">
  <button id="btn" type="submit">login</button>
  <br>{{ error }}
</form>
```

输入错误的用户名密码信息，点击“登录”看看效果：



login

username password login

username or password error!

图 2.5 登录错误提示

等等，貌似哪儿不太对！我的文章怎么没了？这是因为我们在返回 index.html 页面时，只传了 error 信息，并没有传 blog 信息，当然就看不到了。修改 views.py 文件中的 login()函数：

```
def login(request):
    blog_list = Blog.objects.all()
    username = request.POST.get('username', '')
    password = request.POST.get('password', '')
    if username == 'chongshi' and password == '123456':
        return HttpResponse('login success!')
    else:
        return render_to_response('index.html', {'error': 'username or password error!',
        'blogs': blog_list})
```

再来试试错误的用户名和密码的登录。



login

DDD ●●●● login

username or password error!

hello Django

Jan. 5, 2016, 11:03 p.m.

这是我们第一次用Django开发blog应用，跟着虫师一步一步的创建的自己的blog吧！

图 2.6 登录错误与文章都被显示

啊哈！现在就正常了。错误提示和博客都出来了。

回味：

我们在登录之前访问的是：<http://127.0.0.1:8000/index/>

点击登录登录之后，页面就跳到了：<http://127.0.0.1:8000/login/>

虽然在“登录前”和“登录失败”我们都返回的 index.html 页面，render_to_response()函数的第二个传参有区别，从而使页面的显示结果不完全一样。

```
# 登录前
def index(request):
    blog_list = Blog.objects.all()
    return render_to_response('index.html', {'blogs': blog_list})

# 登录处理
def login(request):
    blog_list = Blog.objects.all()
    username = request.POST.get('username', '')
    password = request.POST.get('password', '')
    if username == 'chongshi' and password == '123456':
        return HttpResponse('login success!')
    else:
        return render_to_response('index.html', {'error': 'username or password error!',
        'blogs': blog_list})
```

2.2 cookie 和 session

我们继续另外一个有意思的问题，在不考虑数据库验证的情况下，假如我输入张三用户登录，然后，它就告诉我“嘿，张三你好！”，那么换成李四登录，提示信息就变成“嘿，李四你好！”。

cookie 与 session

cookie 机制。正统的 cookie 分发是通过扩展 HTTP 协议来实现的，服务器通过在 HTTP 的响应头中加上一行特殊的指示以提示浏览器按照指示生成相应的 cookie。然而纯粹的客户端脚本如 JavaScript 或者 VBScript 也可以生成 cookie。而 cookie 的使用是由浏览器按照一定的原则在后台自动发送给服务器的。浏览器检查所

有存储的 cookie，如果某个 cookie 所声明的作用范围大于等于将要请求的资源所在的位置，则把该 cookie 附在请求资源的 HTTP 请求头上发送给服务器。

session 机制。session 机制是一种服务器端的机制，服务器使用一种类似于散列表的结构（也可能就是使用散列表）来保存信息。

2.2.1、用 cookie 保存信息

继续修改 views.py 文件：

```
from blog.models import Blog
from django.shortcuts import render_to_response
from django.http import HttpResponseRedirect, HttpResponseRedirect
.....

# 登录处理
def login(request):
    blog_list = Blog.objects.all()
    username = request.POST.get('username', '')
    password = request.POST.get('password', '')
    # 判断用户名和密码不为空
    if username != '' and password != '':
        # return HttpResponse('login success!')
        response = HttpResponseRedirect('/login_ok/')
        response.set_cookie('username', username, 3600) # 用户名 cookie
        return response
    else:
        return render_to_response('index.html', {'error': 'username or password error!',
        'blogs': blog_list})

# 登录成功
def login_ok(request):
    blog_list = Blog.objects.all()
    username = request.COOKIE.get('username', '') # 读取浏览器 cookie
    return render_to_response('login_ok.html', {'user': username, 'blog_list':
    blog_list})
```

在之前的基础中，我们修改 login 函数，对用户名或密码判断，只判断不能为空。如果都不为空，则调用 HttpResponseRedirect() 函数。HttpResponseRedirect() 方法将请求定向到新 URL，将指定新的 URL。它和我们前

面学过的 URL 地址中的文件目录: http://127.0.0.1:8000/login_ok/

在指定新路径（'/login_ok/'）的时候，通过 `set_cookie()` 方法将 `username` 写到浏览器的 `cookie` 中。

`set_cookie()`

我们给 `set_cookie()` 传了三个参数，第一个参数（'username'）是个字符串用于描述数据，第二个参数（`username`）是从前端用户名输入框中接收到的用户信息，第三个参数（3600）用于标识这个数据保存在浏览器多久后删除，默认以秒为单位。

最终的 `response` 变量带上了文件地址，带上了 `cookie` 信息，`return` 给 `urls.py`。

在 `urls.py` 文件中曾加一条指引：

```
from django.conf.urls import url
from django.contrib import admin
from blog.views import index, login, login_ok

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

接着回到 `views.py` 文件中看看 `login_ok()` 函数做了什么处理：

```
# 登录成功
def login_ok(request):
    blog_list = Blog.objects.all()
    username = request.COOKIES.get('username', '') # 读取浏览器 cookie
    return render_to_response('login_ok.html', {'user': username, 'blog_list':
blog_list})
```

通过 `request.COOKIES.get()` 方法读取浏览器中 `username` 的一条 `cookie` 信息。将信息返回给 `login_ok.html` 页面。

接下来我们需要在 `blog/templates/` 目录下创建 `login_ok.html`

```
{% extends "base.html" %}
{% block content %}
    <h2>嘿! {{ user }}</h2>
```



```
{% for blog in blog_list %}
    <h3>{{ blog.title }}</h3>
    <p>{{ blog.timestamp }}</p>
    <p>{{ blog.body }}</p>
{% endfor %}
{% endblock %}
```

浏览器访问首页：<http://127.0.0.1:8000/index/>

输入用户名密码登录，这里使用了简单的验证，只要用户名和密码不为空即可。

通过 firebug 工具可以看包存在浏览器中的 cookie 信息。通过 `set_cookie()` 方法，我们设置的失效时间是 3600 秒，3600 秒后刷新页面，我们将看不到用户信息；或者直清楚浏览器 cookie 再刷新页面也将看不到用户信息。



图 2.7 查看用户登录 cookie

删除 cookie

当用户正确的点击“退出”按钮后，程序需要将浏览器的 cookie 删除。这也是所谓的“安全退出”。在 `login_ok.html` 页面添加“退出”按钮：

```
<h2>嘿! {{ user }}
<a href="/logout/">退出</a>
</h2>
```

urls.py 添加:

```
from blog.views import logout

urlpatterns = [
    .....
    url(r'^logout/$', logout),
]
```

views.py 添加 logout 函数:

```
# 退出登录
def logout(request):
    response = HttpResponseRedirect('/index/') # 返回首页
    response.delete_cookie('username') # 清理 cookie 里保存 username
    return response
```

delete_cookie() 用于删除指定的 cookie 信息 ('username')。

2.2.2、用 session 保存信息

cookie 固然好,但存在一定的安全隐患,当然我们可以通过信息加密技术增加安全性。cookie 像我们以前用的存折,用户的存钱、取钱都会记录在这张存折上,那么对于有非分之想的人可能会去修改存折上的数据。

相对于存折,银行卡要安全得多,客户拿到的只是一个银行卡号,那么用户的存钱、取钱都会记录在银行里面的,客户只要提供卡号,银行来查询这个卡号上存款的记录。session 的就类似于我们的银行卡。

在 Django 中使用 session 和 cookie 类似。我们只用将 cookie 的几步操作替换成 session 即可。

修改 views.py 代码:

```

.....

# 登录处理
def login(request):
    blog_list = Blog.objects.all()
    username = request.POST.get('username', '')
    password = request.POST.get('password', '')
    # 判断用户名和密码不为空
    if username != '' and password != '':
        response = HttpResponseRedirect('/login_ok/')
        # response.set_cookie('username', username, 3600) # 用户名 cookie
        request.session['username'] = username # 将 session 信息写到服务器
        return response
    else:
        return render_to_response('index.html', {'error': 'username or password error!',
'blogs': blog_list})

# 登录成功
def login_ok(request):
    blog_list = Blog.objects.all()
    # username = request.COOKIES.get('username', '') # 读取浏览器 cookie
    username = request.session.get('username', '') # 读取用户 session
    return render_to_response('login_ok.html', {'user': username, 'blog_list':
blog_list})

# 退出登录
def logout(request):
    response = HttpResponseRedirect('/index/') # 返回首页
    # response.delete_cookie('username') # 清理 cookie 里保存 username
    del request.session['username'] # 清理用户 session
    return response

```

注释掉 cookie 的操作，选用 session，通过上面的代码可以看到，我们只做了三行代码的替换。

```
request.session['username'] = username
```

将表单里提交的用户名信息生成服务器端 session。

```
username = request.session.get('username', '')
```

获取用户名信息并在客户端生成 sessionid。

```
del request.session['username']
```

删除用户名信息的 sessionid。



图 2.8 查看用户登录 session

2.3 Django 认证系统

前面虽然实现登录，但有一个很严重的缺陷，任意页面都可以直接通过 URL 访问。这就好比一个房间，虽然为它安装了一个门，通过这个门进入房间时需要配套的钥匙才能打开门进入。但是，房间所有的窗户都大开着，我们依然可以毫无障碍的通过窗户进入房间。

例如：http://127.0.0.1:8000/login_ok/

这个页面是用户登录之后的页面，但我们现在直接输入这个地址也可以正常访问（只是没有用户信息），这一小节就来堵住可以进入房间的窗户。

2.3.1、登录

Django 提供了用户认证并且默认在进行数据库同步时已经生成了 `users` 类，所以，我们可以直接使用，下面先来改造 `login()` 函数：

```
.....  
from django.contrib import auth  
  
# 登录处理
```

```
def login(request):
    blog_list = Blog.objects.all()
    username = request.POST.get('username', '')
    password = request.POST.get('password', '')
    users_ = [username]
    user = auth.authenticate(username=username, password=password)
    if user is not None:
        auth.login(request, user) # 验证登录
        response = HttpResponseRedirect('/login_ok/')
        request.session['username'] = users_
        return response
    else:
        return render_to_response('index.html', {'error': 'username or password error!',
'blogs':blog_list})

# 登录成功
def login_ok(request):
    blog_list = Blog.objects.all()
    username = request.session.get('username', '') # 读取用户 session
    user = username[0]
    return render_to_response('login_ok.html', {'user': user, 'blog_list':
blog_list})
```

```
user = auth.authenticate(username=username,password=password)
```

认证给出的用户名和密码，使用 `authenticate()` 函数。它接受两个参数，用户名 `username` 和密码 `password`，并在密码对给出的用户名合法的情况下返回一个 `user` 对象。如果密码不合法，`authenticate()` 返回 `None`。

如果你细心还会发现，写将用户名写入 `session` 时也做了一些改变，前面是直接写入的，这里是先将其放到 `users_` 数组里，然后将数组写入 `session`，因为如果直接写入，在登录成功后，Django 会报 “<User: xxx> is not JSON serializable” 的错误，这主要和 Django 的认证系统有关。所以，在读取 `session` 对应的用户名时，有也是取数组中的第一个元素，即 `username[0]`。

登录 Django 后台，添加用户。<http://127.0.0.1:8000/admin>

Django administration

Site administration

Authentication and Authorization

Groups	+ Add	Change
Users	+ Add	Change

BLOG

Blogs	+ Add	Change
-------	-----------------------	------------------------

图 2.9 通过 users 表添加用户

Home > Authentication and Authorization > Users > Add user

Add user

First, enter a username and password. Then, you'll be able to edit more user options.

Username:
Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

Password confirmation:
Enter the same password as before, for verification.

[Save and add another](#) [Save and continue editing](#) [SAVE](#)

图 2.10 创建用户并保存

Django blog

login

[login](#)

图 2.11 通过创建的用户登录

2.3.2、认证

前面只是帮助我们实现了真正的登录，下面就来堵住敞开的窗户。

Django 提供了 `@login_required` 修饰符来修饰视图函数，如检查用户没有登录，则不允许访问这个视图。

```
.....
from django.contrib.auth.decorators import login_required
.....

# 登录成功
@login_required
def login_ok(request):
    blog_list = Blog.objects.all()
    username = request.session.get('username', '') # 读取用户 session
    user = username[0]
    return render_to_response('login_ok.html', {'user': user, 'blog_list': blog_list})

# 退出登录
@login_required
def logout(request):
    response = HttpResponseRedirect('/index/') # 返回首页
    del request.session['username'] # 清理用户 session
    return response
```

`login_required` 做下面的事情：

如果用户没有登录，重定向到 `/accounts/login/`，把当前绝对 URL 作为 `next` 在查询字符串中传递过去，例如：`/accounts/login/?next=/login_ok/`。

如果用户已经登录，正常地执行视图函数。视图代码就可以假定用户已经登录了。

当用户还没有登录的情况下，再去访问：

http://127.0.0.1:8000/login_ok

<http://127.0.0.1:8000/logout>

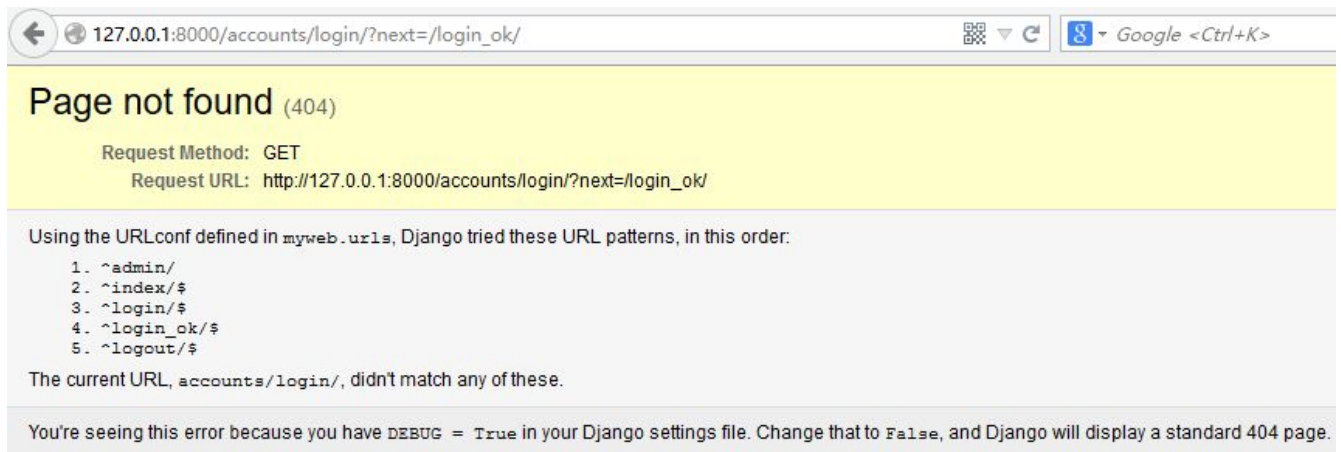


图 2.14 访问需要登录的页面

默认会跳转到 ‘/accounts/login/’，这显然不是我们希望用户看到的页面，因为它暴露了太多信息。我们希望它能自动的跳转到 404 页面，或网站首页 `index.html` 页面。当前设置跳转到首页。在 `urls.py` 中添加：

```
url(r'^accounts/login/$', index)
```

当然，如果你真访问一下不存在的页面，如：<http://127.0.0.1:8000/abc>

还是会报图 2.14 的页面，这个时候就需要设置 404 页面，我们将会在后面的章节中介绍。

第三章 Django 模型

模型（model）层主要用来创建数据的，前端丰富数据的展示，离不开数据库的支持，所以，我们有必要详细的学习在 Django 中实现数据的增、删、查、改。

3.1 制作图书管理

这里我们引用《The Django Book》中的例子，因为这个例子颇具代表性。

3.1.1、数据库配置

首先配置数据库，打开.../myweb/setting.py 文件，做如下设置。

```
# Database
# https://docs.djangoproject.com/en/1.9/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

默认配置数据库为 sqlite3，sqlite3 为 Python 内置的数据库，所以，不用我们额外安装，在后面章节我们会介绍 mysql 的安装与配置。

3.1.2、创建表的映射

打开 blog/models.py 文件，创建作者、出版社和书。

```
from django.db import models
from django.contrib import admin
```

```

# Create your models here.
# 作者
class Author(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=40)
    email = models.EmailField()

    def __unicode__(self):
        return self.first_name

# 出版商
class Publisher(models.Model):
    name = models.CharField(max_length=30)
    address = models.CharField(max_length=50)
    city = models.CharField(max_length=60)
    state_province = models.CharField(max_length=30)
    country = models.CharField(max_length=50)
    website = models.URLField()

    def __unicode__(self):
        return self.name

# 书
class Book(models.Model):
    title = models.CharField(max_length=100)
    authors = models.ManyToManyField(Author)
    publisher = models.ForeignKey(Publisher)
    publication_date = models.DateField()

    def __unicode__(self):
        return self.title

```

- 一个作者有姓，有名及 **email** 地址。
- 出版商有名称，地址，所在城市、省，国家，网站。
- 书籍有书名和出版日期。 它有一个或多个作者（和作者是多对多的关联关系[many-to-many]）， 只有一个出版商（和出版商是一对多的关联关系[one-to-many]，也被称作外键[foreign key]）

```
from django.db import models
```

引入 `django.db` 下在 `models` 模块。

```
models.CharField(max_length=30)
```

使用 `models.CharField()` 方法创建字符类型字段，字段长度为 `max_length=30`。

```
models.EmailField()
```

使用 `models.EmailField()` 方法用于创建存放 `email` 类型的字段。

```
models.URLField()
```

使用 `models.URLField()` 方法用于创建存放 `URL` 地址类型的字段。

```
authors = models.ManyToManyField(Author)
```

定义 `books` 与 `author` 是多对多的关系，一本书可以有多个作者，一个作者也可以写多本书。

```
publisher = models.ForeignKey(Publisher)
```

在 `book` 类（表）中定义出版商的外键。

```
models.DateField()
```

使用 `models.DateField()` 方法用于创建存放日期类型的数据。

3.1.3、Django 数据类型

Django 数据类型，如下表：

（依据 `C:\Python35\Lib\site-packages\django\db\models\fields__init__.py` 文件）

类型	说明
<code>AutoField</code>	用于存放 <code>integer</code> 类型的数字。

BooleanField	用于存放布尔类型的数据（True 或 False）
CharField	用于存放字符型的数据，需要指定长度 max_length。
CommaSeparatedIntegerField	用于存放用逗号隔开的 integer 类型的数据。
DateField	日期型，必须是“YYYY-MM-DD”格式
DateTimeField	日期时间型，必须是"YYYY-MM-DD HH:MM[:ss[.uuuuuu]][TZ]"格式。
DecimalField	小数型，用于存放小数的数字。
EmailField	电子邮件类型
FilePathField	文件路径类类型，FilePathFields must have either 'allow_files' or 'allow_folders' set to True.
FloatField	浮点型。用于存放浮点型数据。
IntegerField	用于存放 integer 类型的数字。
BigIntegerField	用于存放大 integer 类型的数字，最大数支持：9223372036854775807
IPAddressField	IP 地址类型，支持的 IPv4 的长度。
GenericIPAddressField	一般的 IP 地址，非 IPv4
NullBooleanField	value must be either None, True or False.
PositiveIntegerField	Positive integer
PositiveSmallIntegerField	Positive small integer
SlugField	需要定义 max_length 值。
SmallIntegerField	Small integer
TextField	用于存放文本类型的数据。
TimeField	时间类型。"HH:MM[:ss[.uuuuuu]]" 格式
URLField	用于存放 URL 地址
BinaryField	Raw binary data

```
def __unicode__(self):
    return self.first_name
```

`__unicode__()` 方法告诉 Python 如何实现对象的 unicode 表示。

下面执行数据库的生成与同步:

cmd.exe

```
D:\pydj\myweb>python3 manage.py makemigrations blog
```

```
Migrations for 'blog':
```

```
0003_auto_20160114_2321.py:
```

- Create model Author
- Create model Book
- Create model Publisher
- Add field publisher to book

```
D:\pydj\myweb>python3 manage.py migrate
```

```
Operations to perform:
```

```
Apply all migrations: admin, sessions, blog, contenttypes, auth
```

```
Running migrations:
```

```
Applying blog.0003_auto_20160114_2321... OK
```

3.1.4、创建表的后台管理

下面我们可以将创建的这几张表交给 admin 后台进行管理, 下面打开.../blog/admin.py 文件:

```
from django.contrib import admin
from blog.models import *

# Register your models here.
class AutAdmin(admin.ModelAdmin):
    list_display = ('first_name', 'last_name', 'email')

class PubAdmin(admin.ModelAdmin):
    list_display = ('name', 'address', 'city', 'state_province', 'country', 'website')

class BookAdmin(admin.ModelAdmin):
    list_display = ('title', 'publication_date')
```

```
admin.site.register(Author, AutAdmin)
admin.site.register(Publisher, PubAdmin)
admin.site.register(Book, BookAdmin)
```

```
from blog.models import *
```

导入 models.py 文件下的所有类。

```
class AutAdmin(admin.ModelAdmin):
```

AutAdmin 类继承 admin.ModelAdmin 类。

```
list_display = ('first_name', 'last_name', 'email')
```

list_display 返回作者 (Author) 的姓和名, 以及 email 地址。

```
admin.site.register(Author, AutAdmin)
```

admin.site.register() 将 Author 与 AutAdmin 对应。

登录 admin 后台: <http://127.0.0.1:8000/admin>

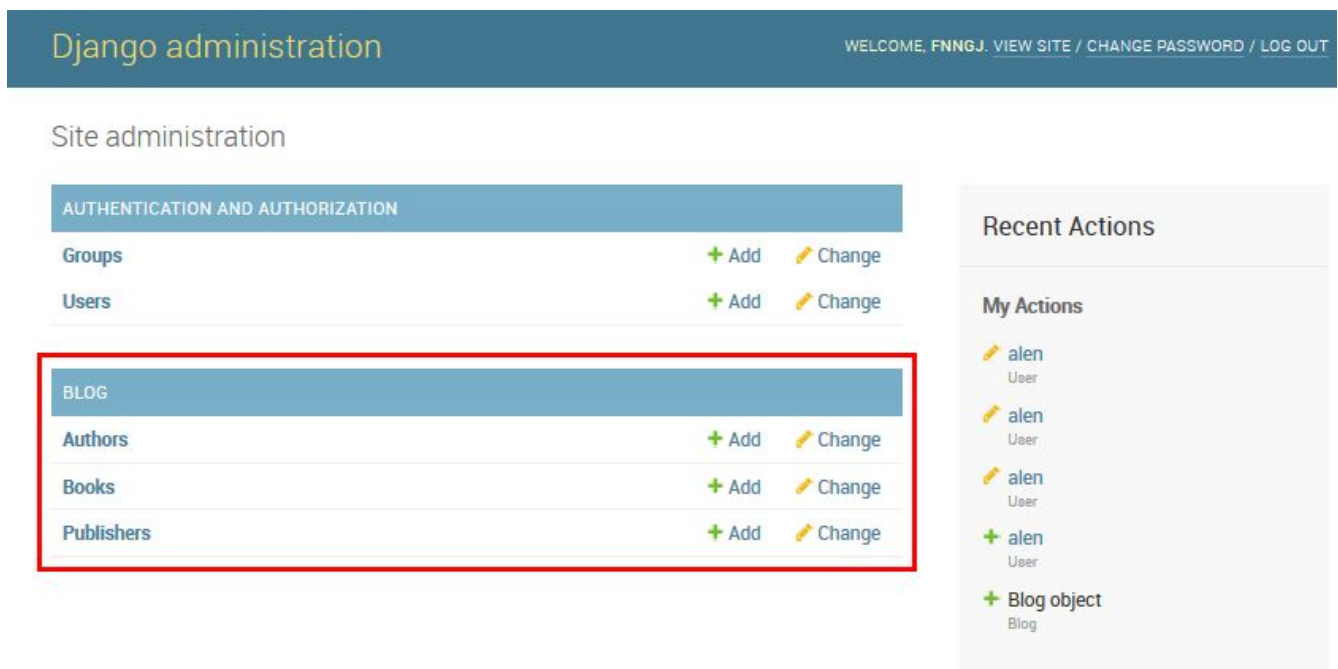


图 3.1 admin 后台管理图书相关表

如图 2.15，为 Authors、Books、Publishers 三张表中添加数据。点击“add”添加一些信息。

3.2 数据库表操作

我们可以在 django 提供的 shell 下进行数据库的相关操作：

cmd.exe

```
D:\pydj\myweb> python3 manage.py shell
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

通过“python manage.py shell”命令进入 shell（这里与我们直接敲“python”命令进入的 shell 有区别）。

cmd.exe

```
>>> from blog.models import *
>>> Author.objects.all()
[<Author: Author object>, <Author: Author object>]
>>> Book.objects.all()
[<Book: Book object>]
>>> Publisher.objects.all()
[<Publisher: Publisher object>]
```

```
from blog.models import *
```

导入 blog 目录下面，models.py 文件下的所有（*）类（表）。

```
xxx.objects.all()
```

获得 xxx 表中的所有对象。

在上一节中，我要求你们通过 admin 管理后台在这三张表中添加一些数据，这是我添加的，不知道你们有没有添加。如果没有请去添加，不然还怎么一起愉快的玩耍呢。

3.2.1、插入数据

```
cmd.exe
```

```
>>> p1 = Publisher(name='Apress', address='2855 Telegraph Avenue', city='Berkeley',  
state_province='CA', country='U.S.A.', website='http://www.apress.com/')  
  
>>> p1.save()
```

我们插入了一条出版社信息，给表中的每个字段赋值。通过 `save()` 方法进行保存。

3.2.2、查询数据

通过 `name="Apress"` 条件查询：

```
cmd.exe
```

```
>>> Publisher.objects.get(name='Apress')  
<Publisher: Publisher object>  
>>> Publisher.objects.get(name='Apress').name  
'Apress'  
>>> Publisher.objects.get(name='Apress').country  
'U.S.A.'  
>>> Publisher.objects.get(name='Apress').website  
'http://www.apress.com/'
```

或者可以借助变量来查询：

```
cmd.exe
```

```
>>> p = Publisher.objects.get(name='Apress')  
>>> p.address  
'2855 Telegraph Avenue'  
>>> p.city  
'Berkeley'
```


查看 title=selenium2 这本书的作者信息:

```
cmd.exe
```

```
>>> a = Book.objects.get(title='selenium2')
>>> a.authors.values()
[{'email': 'fnngj@126.com', 'first_name': '虫师', 'id': 2, 'last_name': '虫师'}]
```

或者我还想来个模糊查询:

```
cmd.exe
```

```
>>> Book.objects.filter(title__contains = r'(s|e)')
[<Book: Book object>, <Book: Book object>]
```

搜索条件是书名中包含“p”或“e”字母。

不过,我们查询到的都是对象(object)下面通过 for 循环打印每一个对象的 title:

```
cmd.exe
```

```
>>> books = Book.objects.filter(title__contains = 'selenium2')
>>> for b in books:
...     print(b.title)
...
selenium2 for python
selenium2
```

3.2.3、删除数据

删除 first name=hu 的作者。

```
cmd.exe
```

```
>>> authors = Author.objects.all()
>>> for a in authors:
...     print(a.first_name)
```

```
...
hu
虫师
>>> d = Author.objects.get(first_name='hu')
>>> d.delete()
(2, {'blog.Author': 1, 'blog.Book_authors': 1})
```

3.2.4、更新数据

更新 book 表中 title=selenium2 的 title 为 python。

cmd.exe

```
>>> b = Book.objects.filter(title='selenium2')
>>> b.update(title='python')
1
```

3.3 安装 SQLiteStudio

Sqlitestudio 是一款 Sqlite 数据库可视化工具，是使用 Sqlite 数据库开发应用的必备软件，软件无需安装，下载后解压即可使用，很小巧但很了用，绿色中文版本。比起其它 SQLite 管理工具，我喜欢用这个。很方便易用，不用安装的单个可执行文件，支持中文。

SQLiteStudio 是一个跨平台的 SQLite 数据库的管理工具，采用 Tcl 语言开发。

下载地址：<http://sqlitestudio.pl/>

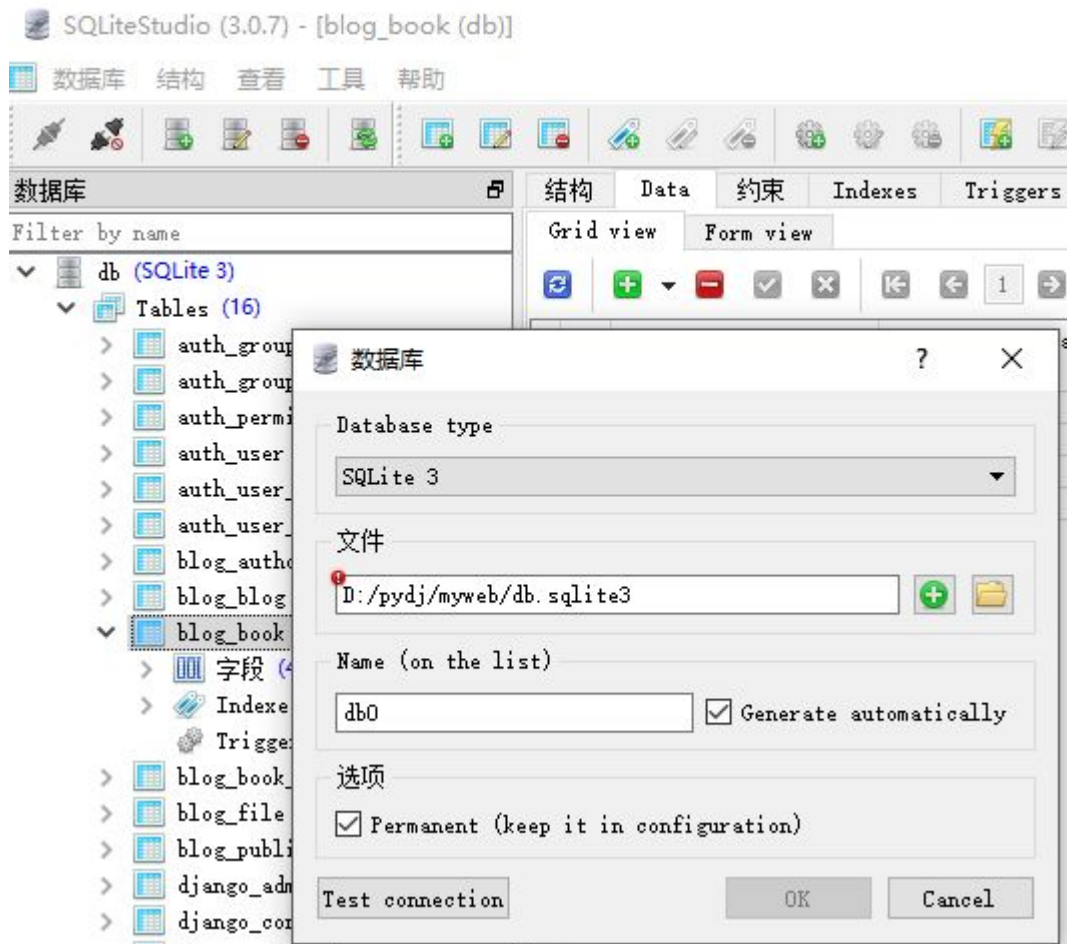


图 3.2 选择打开的 db.sqlite3 文件

3.3 配置 MySQL

前面用的数据库是 Python 自带的 sqlite3，这种数据库并不适用大型的项目。除此之外，Django 支持四种数据库，分别是：

- PostgreSQL (<http://www.postgresql.org/>)
- SQLite 3 (<http://www.sqlite.org/>)
- MySQL (<http://www.mysql.com/>)
- Oracle (<http://www.oracle.com/>)

本节以 mysql 为例，介绍 mysql 的安装，以及在 django 中的配置。

3.3.1、安装 MySQL

下载 mysql: <http://dev.mysql.com/downloads/mysql/>

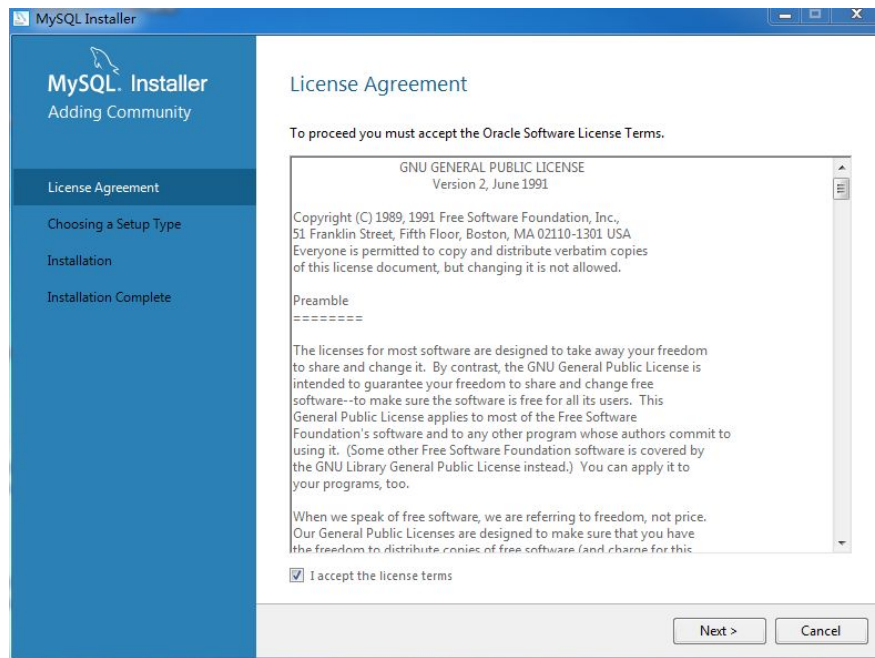


图 3.3 MySQL 数据库安装

双击运行安装程序，如图 3.2，根据提示一步一步进行安装，作者这里就不再详细介绍每一步的安装过程了，请参考其它 mysql 安装手册。

进入 mysql

通过 Windows 命令提示符下进入 mysql。

cmd.exe

```
C:\Users\fnngj>mysql -u root -p
```

```
Enter password: *****
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 5
```

```
Server version: 5.5.19 MySQL Community Server (GPL)
```

```
Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.
```

```
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.
```

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>

登录的用户名为 root ，密码在我们安装 mysql 的过程中填写，如果没填写默认为空。

mysql 基本操作

查看库与表：

cmd.exe

mysql> **show databases;** #查看当前数据库下面的所有库

```
+-----+
| Database          |
+-----+
| information_schema |
| mysql             |
| performance_schema |
| test              |
+-----+
4 rows in set (0.00 sec)
```

mysql> **use test;** # 切换到 test 库

Database changed

mysql> **show tables;** # 查看 test 库下面的表

Empty set (0.00 sec)

查看 mysql 端口号：

cmd.exe

mysql> **show global variables like 'port';**

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| port         | 3306  |
+-----+-----+
```

```
1 row in set (0.00 sec)
```

创建数据库:

```
cmd.exe
```

```
mysql> CREATE DATABASE myweb;  
Query OK, 1 row affected (0.00 sec)
```

创建 myweb 库，用于我们 Django 的 myweb 项目。

3.3.2、安装 PyMySQL

这里遇到小小的分歧,如果读者使用的 Python2.x 版本,那么连接 MySQL 数据库可以使用 MySQL-python。但是, MySQL-python 只支持 Python2.x 版本,而且如果使用的系统是 Win 64 位,还需要单独查找 64 位版本 (mysql-python-1.2.5.win-amd64-py2.7.exe)。

下载地址: <https://pypi.python.org/pypi/MySQL-python>

而我们当前使用的是 Python3.x 版本,所以需要使用 PyMySQL 驱动。通过 pip 命令进行安装:

下载地址: <https://pypi.python.org/pypi/PyMySQL>

Python2.x :

```
cmd.exe
```

```
C:\Users\fnngj>pip install PyMySQL
```

Python3.x :

```
cmd.exe
```

```
C:\Users\fnngj>python3 -m pip install PyMySQL
```

官方操作例子:

```
import pymysql  
conn = pymysql.connect(host='127.0.0.1', unix_socket='/tmp/mysql.sock', user='root',  
passwd=None, db='mysql')
```

```
# conn = pymysql.connect(host='127.0.0.1', port=3306, user='root', passwd=None,
db='mysql')
cur = conn.cursor()
cur.execute("SELECT Host, User FROM user")
# print cur.description
# r = cur.fetchall()
# print r
# ...or...
for r in cur:
    print r

cur.close()
conn.close()
```

3.3.3、Django 配置 MySQL

我们想在 django 中使用 mysql 数据库，那么需要 settings.py 中修改配置。

```
# Database
# https://docs.djangoproject.com/en/1.9/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'HOST': '127.0.0.1',
        'PORT': '3306',
        'NAME': 'myweb',
        'USER': 'root',
        'PASSWORD': '123456',
    }
}
```

配置信息从上到下依次是驱动（ENGINE），主机地址（HOST），端口号（PORT），数据库（NAME），登录用户名（USER），登录密码（PASSWORD）。

检测配置是否正确：

cmd.exe

```
D:\pydj\myweb>python3 manage.py shell
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from django.db import connection
>>> cursor = connection.cursor()
>>>
```

如果没有报错，说明我们的配置没有问题。

因为现在切换了数据库，之前在 sqlite3 下的创建的表，需要在重新在 mysql 数据库下生成。

cmd.exe

```
D:\pydj\myweb>python3 manage.py makemigrations blog
Migrations for 'blog':
  0003_auto_20160114_2321.py:
    - Create model Author
    - Create model Book
    - Create model Publisher
    - Add field publisher to book

D:\pydj\myweb>python3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, sessions, blog, contenttypes, auth
Running migrations:
  Applying blog.0003_auto_20160114_2321... OK
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
```


Applying sessions.0001_initial... OK

注意：切换了数据库后，之前数据库里的数据就没有了，需要我们重新通过 **admin** 后台去添加数据。

查看 **mysql** 数据库：

cmd.exe

```
mysql> use myweb;
Database changed
mysql> show tables;
+-----+
| Tables_in_myweb          |
+-----+
| auth_group                |
| auth_group_permissions    |
| auth_permission           |
| auth_user                 |
| auth_user_groups          |
| auth_user_user_permissions |
| blog_author               |
| blog_blogspost            |
| blog_book                 |
| blog_book_authors         |
| blog_publisher            |
| django_admin_log          |
| django_content_type        |
| django_migrations         |
| django_session            |
+-----+
15 rows in set (0.00 sec)
```

3.4 配置 Oracle

如果你想使用 Django 连接 Oracle 数据库的话，此处默认你已经安装并可以正常使用 Oracle 数据库。

3.4.1、安装 cx_Oracle

cx_Oracle 用于驱动 Python 连接 Oracle 数据库：

下载地址：https://pypi.python.org/pypi/cx_Oracle

cx_Oracle 是一个用来连接并操作 Oracle 数据库的 Python 扩展模块，支持包括 Oracle 9.2、10.2 以及 11.1 等版本。该驱动未提供 pip 的安装方式，需要下载 exe 程序，双击进行安装。

3.4.2、Django 配置 Oracle

配置 setting.py 文件：

```
# Database
# https://docs.djangoproject.com/en/1.9/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE' : 'django.db.backends.oracle', #数据库驱动
        'HOST' : '192.168.201.138', #数据库地址
        'PORT' : '1521', #端口号
        'NAME' : 'ORCL', #库名
        'USER' : 'username', #登录用户名
        'PASSWORD' : 'password', #登录密码
    }
}
```

连接数据库：

cmd.exe

```
D:\pydj\myweb> python3 manage.py shell
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
```

```
>>> from django.db import connection
>>> cur = connection.cursor()
>>> cur.execute("select * from blog")
<cx_Oracle.Cursor on <cx_Oracle.Connection to otcnewsett@ (DESCRIPTION= (ADDRESS_L
IST= (ADDRESS= (PROTOCOL=TCP) (HOST=192.168.201.138) (PORT=1521))) (CONNECT_DATA= (SI
D=ORCL)) )>>
>>> for row in cur:
...     print row
```

第四章 Django 模板

Django 提供了模板语言，该模板语言主要控制着数据如何在页面中显示。例如，在本书第一章，将 blog 循环的显示在 web 页面上。

```
{% for blog in blogs %}
    <h2>{{ blog.title }}</h2>
    <p>{{ blog.timestamp }}</p>
    <p>{{ blog.body }}</p>
{% endfor %}
```

当然，如何如何使页面显示更漂亮，还是需要依靠前端技术 HTML、CSS、JavaScript 等。

不管是 Django 的模板语言也好，还是前端技术也好，都不是本书所想介绍的重点。所以，本章直接介绍 Bootstrap 前端框架。

什么是 Bootstrap?

Bootstrap, 来自 Twitter, 是目前很受欢迎的前端框架。Bootstrap 是基于 HTML、CSS、JAVASCRIPT 的, 它简洁灵活, 使得 Web 开发更加快捷。它由 Twitter 的设计师 Mark Otto 和 Jacob Thornton 合作开发, 是一个 CSS/HTML 框架。Bootstrap 提供了优雅的 HTML 和 CSS 规范, 它即是由动态 CSS 语言 Less 写成。

Bootstrap 中文网: <http://www.bootcss.com/>

4.1 安装 Bootstrap

要想使用 Bootstrap 框架, 首先需要进行下载与安装, 接下来我们就来安装 Bootstrap 框架。

Django-bootstrap3 开源地址: <https://github.com/dyve/django-bootstrap3>

1、通过 Python 的 pip 命令安装:

cmd.exe

```
C:\Users\fnngj>python3 -m pip install django-bootstrap3
```

```
Collecting django-bootstrap3
  Downloading django-bootstrap3-6.2.2.tar.gz
Installing collected packages: django-bootstrap3
  Running setup.py install for django-bootstrap3
Successfully installed django-bootstrap3-6.2.2
```

2、在 Django 的 settings.py 文件中的 INSTALLED_APPS 模块中添加 “bootstrap3”。

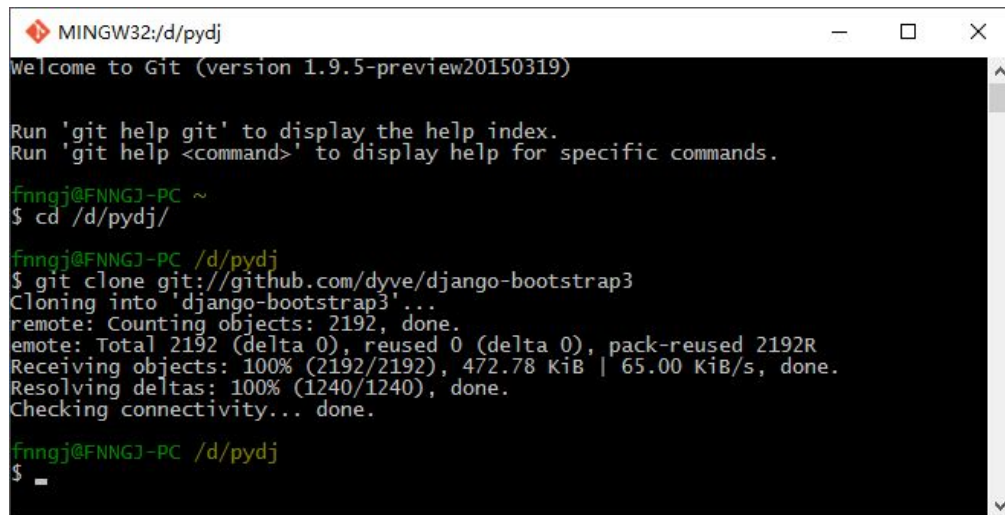
```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog',
    'bootstrap3',
]
```

3、下载并运行 demo。

在 django-bootstrap3 项目中带了完整的 demo，下面从 github 上克隆项目到本地，打开 git，通过下面的命令进行克隆。

```
$ git clone git://github.com/dyve/django-bootstrap3
```



```
MINGW32:/d/pydj
Welcome to Git (version 1.9.5-preview20150319)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

fnngj@FNNGJ-PC ~
$ cd /d/pydj/

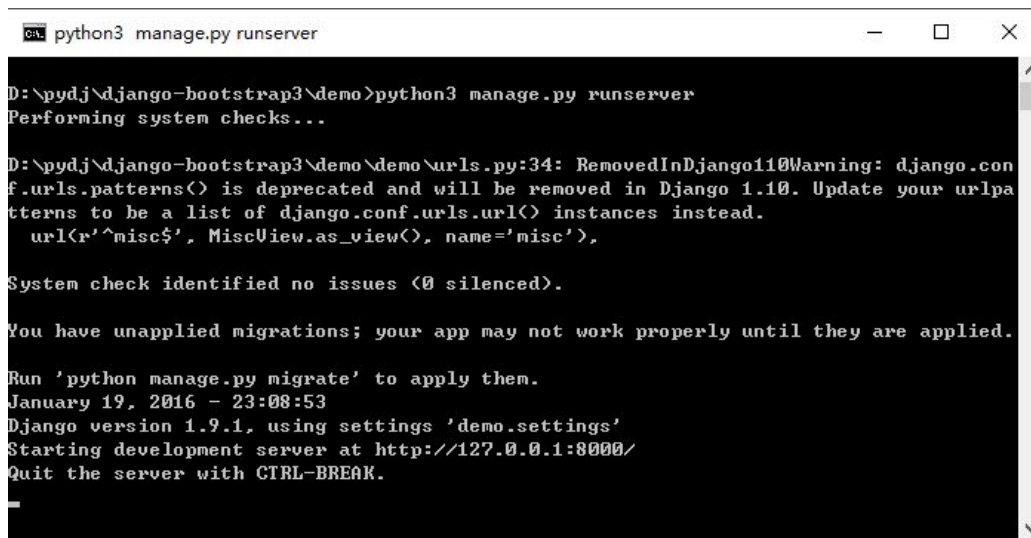
fnngj@FNNGJ-PC /d/pydj
$ git clone git://github.com/dyve/django-bootstrap3
Cloning into 'django-bootstrap3'...
remote: Counting objects: 2192, done.
remote: Total 2192 (delta 0), reused 0 (delta 0), pack-reused 2192
Receiving objects: 100% (2192/2192), 472.78 KiB | 65.00 KiB/s, done.
Resolving deltas: 100% (1240/1240), done.
Checking connectivity... done.

fnngj@FNNGJ-PC /d/pydj
$
```

图 4.1 git 克隆 django-bootstrap3

克隆下来的 django-bootstrap3 项目自带 demo 项目，demo 和我们通过 Django 生成的 myweb 项目一样，所以，我们可以进入到 demo 目录下运行该项目。

> python3 manage.py runserver



```
python3 manage.py runserver

D:\pydj\django-bootstrap3\demo>python3 manage.py runserver
Performing system checks...

D:\pydj\django-bootstrap3\demo\demo\urls.py:34: RemovedInDjango110Warning: django.conf.urls.patterns() is deprecated and will be removed in Django 1.10. Update your urlpatterns to be a list of django.conf.urls.url() instances instead.
  url(r'^misc$', MiscView.as_view(), name='misc'),

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.
January 19, 2016 - 23:08:53
Django version 1.9.1, using settings 'demo.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

-
```

图 4.2 运行 demo

访问：<http://127.0.0.1:8000/>



图 4.3 django-bootstrap3 项目首页

在该 demo 中提供了不同样式的按钮、输入框、文本框、单选框、复选框、下拉框，甚至是翻页样式，我们点击上面的链接查看。从而在页面开发过程中找到想到样式使用。

这一节我们来使用 bootstrap3 的样式制作一张表格。

4.2 制作表格

首先回到我们的 myweb 项目中，在 ../blog/ 目录下 views.py 文件中添加。

```
# 学生表
def student(request):
    student = {
        'jack': [22, 'boy', 'Programmer'],
        'alen': [27, 'boy', 'Designer'],
        'una': [23, 'girl', 'Tester'],
        'Brant': [23, 'girl', 'Tester'],
        'David': [23, 'boy', 'Tester']
    }
    return render_to_response('student.html', {'student_list': student})
```

在 myweb/urls.py 中添加：

```
.....
```

```

from blog.views import student

urlpatterns = [
.....
    url(r'^student/$', student),
]

```

最后，在.../blog/templates/目录下创建 student.html 页面。

```

<html lang="zh-CN">
<head>
    {# Load the tag library #}
    {% load bootstrap3 %}

    {# Load CSS and JavaScript #}
    {% bootstrap_css %}
    {% bootstrap_javascript %}

    {# Display django.contrib.messages as Bootstrap alerts #}
    {% bootstrap_messages %}
</head>
<body>

    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <a class="navbar-brand" href="#">Project name</a>
            </div>
        </div>
    </nav>

    <br><br>
    <div class="well">
        <h2>student information</h2>
        {% if student_list %}
            <table class="table">
                <tr>
                    <th>name</th>
                    <th>age</th>
                    <th>sex</th>
                    <th>occupation</th>
                </tr>
                {% for k,v in student_list.items %}
                    <tr>
                        <td>{{ k }}</td>
                        {% for i in v %}
                            <td>{{ i }}</td>
                        {% endfor %}
                    </tr>
                {% endfor %}
            </table>
        {% else %}
            <div class="text-center">
                <p>No student information found.</p>
            </div>
        {% endif %}
    </div>
</body>
</html>

```



```

        </tr>
    {% endfor %}
</table>
{% endif %}
</div>

<footer>
    <p>&copy; Company 2016 & chongshi</p>
</footer>

</body>
</html>

```

启动 django 的内设 server，然后访问：<http://127.0.0.1:8000/student/>

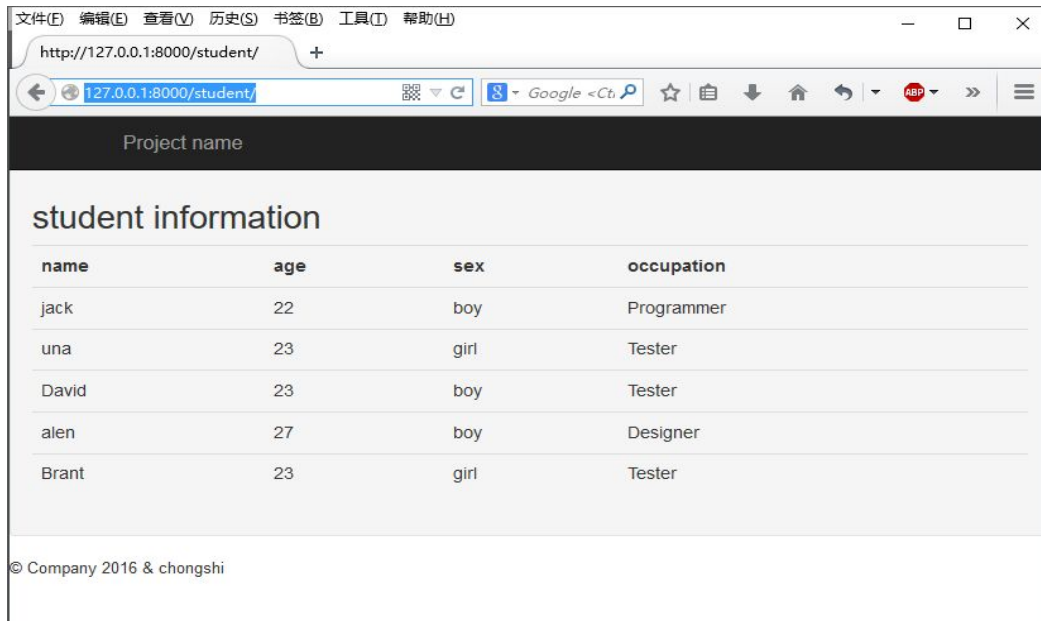


图 4.4 bootstrap3 表格

4.3 调用图书列表

在上节的例子中，我们并没有调用数据，而是在视图创建了一个数组，通过读取数组来生成页面上的表格。其实，在本书第三章中已经创建了三张关于图书的表。这一节就来读取其中的 **book** 表，并将其显示在 web 页面上。

首先，打开视图文件 **views.py**，过程一样，为了简单起见，我们直接修改 **sutdent** 函数：

```

.....
from blog.models import *
.....

# 学生表
def student(request):
    books = Book.objects.all()
    student = {
        'jack': [22, 'boy', 'Programmer'],
        'alen': [27, 'boy', 'Designer'],
        'una': [23, 'girl', 'Tester'],
        'Brant': [23, 'girl', 'Tester'],
        'David': [23, 'boy', 'Tester']
    }
    return render_to_response('student.html', {'student_list': student, 'book_list':
books})

```

读取 book 表中的所有内容，并把给 book_list 对象，显示在 student.html 上。

接着，修改 student.html 文件，并添加如下内容：

```

<div class="container">
  <!-- Example row of columns -->
  <div class="row">
    <h2>book list</h2>
    <table class="table">
      <th>name</th><th>publisher</th><th>data</th><th>authors</th>
      {% for book in book_list %}
      <tr class="success">
        <td>{{book.title}}</td>
        <td>{{ book.publisher.name }}</td>
        <td>{{book.publication_date}}</td>
        <td>
          {% for author in book.authors.values %}
            {{author.first_name}}
          {% endfor %}
        </td>
      </tr>
      {% endfor %}
    </table>
  </div>
</div>

```

```
</div>
</div>
```

循环读取 `boo_list` 内容，显示在 web 页面上，如图 4.5。

book list

name	publisher	data	authors
selenium2 for python	机械工业出版社	Jan. 17, 2016	虫师
python	Apress	Jan. 17, 2016	chongshi

图 4.5 book 表

4.4 分页功能

既然开发了列表，那么当列的数据多了之后必要要考虑分页功能，那么接下来我们继续开发分页功能。

4.4.1、Paginator 类基本使用

Django 早就为我们准备好了 `Paginator` 分页类。所以，只需要它调用它即可完成列表的分页功能。

```
cmd.exe
```

```
D:\pydj\myweb>python3 manage.py shell
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from django.core.paginator import Paginator #导入 Paginator 类
>>> from blog.models import * #blog 下的所有表
>>> book_list = Book.objects.all() #查询 Book 表的所有数据
>>> p = Paginator(book_list, 2) #创建每页 2 条数据的分页器
>>> p.count #查看共多少条数据
5
>>> p.page_range #查看共分多少页（每页 2 条数据）循环结果为 1, 2, 3（共 3 页）
range(1, 4)
>>>
```

```

#####第一页#####
>>> page1 = p.page(1) #获取第 1 页的数据
>>> page1          #当前第几页
<Page 1 of 3>
>>>
>>> page1.object_list #当前页的对象
[<Book: Book object>, <Book: Book object>]
>>>
>>> page1 = p.page(1) #循环打印第 1 页书的 titile
>>> for p in page1:
...     p.title
...
'selenium2 for python'
'python 基础教程'
>>>
#####第二页#####
>>> page2 = p.page(2) #获取第 2 页的数据
>>> page2.start_index() #本页的第一条数据
3
>>> page2.end_index() #本页的最后一条数据
4
>>> page2.has_previous() #是否有上一页
True
>>> page2.has_next() #是否有下一页
True
>>> page2.previous_page_number() #上一页是第几页
1
>>> page2.next_page_number() #下一页是第几页
3
>>>
#####第三页#####
>>> page3 = p.page(3) #获取第 3 页的数据
>>> page3.has_next() #是否有下一页
False
>>> page3.has_previous() #是否有上一页
True
>>> page3.has_other_pages() #是否有其它页

```

```
True
>>> page3.previous_page_number() #前一页是第几页
2
```

通过对 **Book** 表的练习，现在已经学会了 **Paginator** 类的基本操作，那么下面就来实现分页面吧！

4.4.2、实面分页功能

接下来，打开...blog/views.py 文件：

```
.....
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger
.....
def page(request):
    file_list = Book.objects.all()
    paginator = Paginator(file_list, 2)    # Show 2 contacts per page
    page = request.GET.get('page')
    try:
        contacts = paginator.page(page)
    except PageNotAnInteger:
        # If page is not an integer, deliver first page.
        contacts = paginator.page(1)
    except EmptyPage:
        # If page is out of range (e.g. 9999), deliver last page of results.
        contacts = paginator.page(paginator.num_pages)

    return render_to_response('book.html', {'pages': contacts})
```

打开.../myweb/urls.py 文件添加 url 地址指向：

```
.....
from blog.views import page

urlpatterns = [
    .....
```

```
url(r'^book/$', page),
```

继续，在.../blog/templates/目录下创建 book.html 页面。

```
<html lang="zh-CN">
<head>
    {# Load the tag library #}
    {% load bootstrap3 %}
    {# Load CSS and JavaScript #}
    {% bootstrap_css %}
    {% bootstrap_javascript %}
    {# Display django.contrib.messages as Bootstrap alerts #}
    {% bootstrap_messages %}
</head>
<body>
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <a class="navbar-brand" href="#">Project name</a>
            </div>
        </div>
    </nav>

    <br><br>
    <div class="container">
        <div class="row">
            <h2>book list</h2>
            <table class='table'>
                <th>name</th><th>publisher</th><th>data</th><th>authors</th>
                {% for book in pages %}
                <tr class='success' >
                    <td>{{book.title}}</td>
                    <td>{{ book.publisher.name }}</td>
                    <td>{{book.publication_date}}</td>
                    <td>
                        {% for author in book.authors.values %}
                            {{author.first_name}}
                        {% endfor %}
                    </td>
                </tr>
                {% endfor %}
            </table>
        </div>
    </div>
</body>
</html>
```

```

        </tr>
    </table>
</div>
</div>
<!-- Paging capabilities -->
<div class="pagination">
    <span class="step-links">
        {% if pages.has_previous %}
        <a href="?page={{ pages.previous_page_number }}">previous</a>
        {% endif %}
        <span class="current">
            Page {{ pages.number }} of {{ pages.paginator.num_pages }}.
        </span>
        {% if pages.has_next %}
        <a href="?page={{ pages.next_page_number }}">next</a>
        {% endif %}
    </span>
</div>

<footer>
    <p>&copy; Company 2016 & chongshi</p>
</footer>

</body>
</html>

```

如图 4.6, book 列表左下角显示分页功能, 我们可以点击 “next” 和 “previous” 进行翻页。

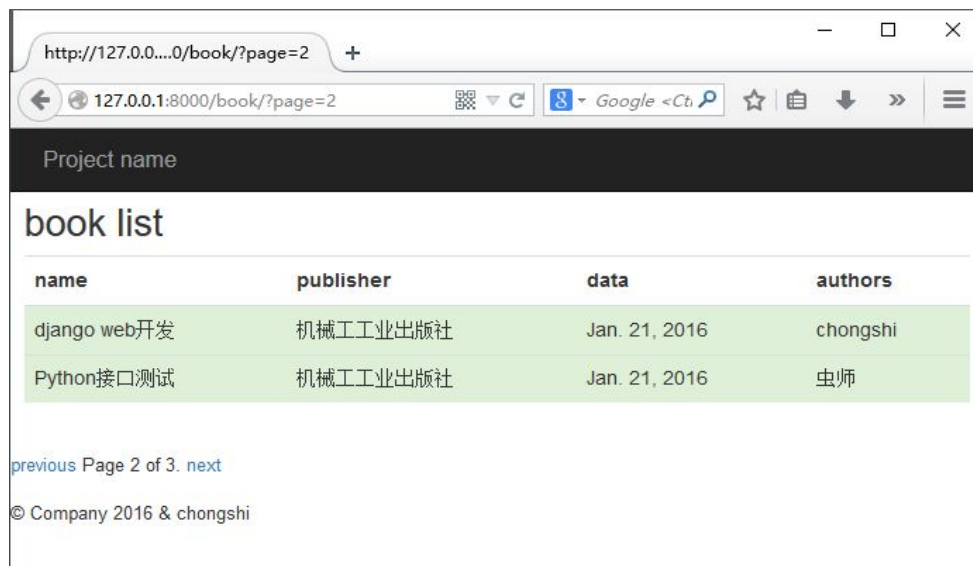


图 4.6 book 列表的分页功能

4.5 文件上传功能

接下来为什么开发一个上传功能呢？没有为什么，但这是一个非常常见功能，如果进行 web 开发工作，避免不了要开发此功能，不是吗？

4.5.1、开发上传功能

创建模型：

首先，我们要创建用于存放上传文件的表，打开...blog/models.py 文件，添加：

```
# file upload
class File(models.Model):
    filename = models.CharField(max_length=30)
    fileway = models.FileField(upload_to='./upload/')

    def __unicode__(self):
        return self.filename
```

File 表分别定义两个字段，filename 用于存放文件名（或文件描述），fileway 用于存放上传文件的路径。

接下来执行数据库同步：

cmd.exe

```
D:\pydj\myweb>python3 manage.py makemigrations blog
```

```
Migrations for 'blog':
```

```
0004_file.py:
```

```
- Create model File
```

```
D:\pydj\myweb>python3 manage.py migrate
```

```
Operations to perform:
```

```
Apply all migrations: sessions, admin, blog, auth, contenttypes
```

```
Running migrations:
```


Applying blog.0004_file... OK

创建视图:

在.../blog/views.py 文件中增加:

```
# 上传文件页面
def upload(request):
    return render_to_response('upload.html')

# 执行文件上传
def upload_save(request):
    filename = request.POST.get('filename', '') # 获得表单文件说明
    fileing = request.FILES.get('fileing', '') # 获得文件
    upload = File() # 将文件名和文件路径存放到 File 表中
    upload.filename = filename
    upload.fileway = fileing
    upload.save()
    return render_to_response('upload.html')
```

创建模板:

创建.../blog/templates/upload.html 文件并且添加如下内容:

```
<html lang="zh-CN">
<head>
    {% load bootstrap3 %}
    {% bootstrap_css %}
    {% bootstrap_javascript %}
    {% bootstrap_messages %}
</head>
<body>
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <a class="navbar-brand" href="#">Project name</a>
            </div>
        </div>
    </nav>
```

```

</nav>

<br><br>
<div class="well">
    <h3>upload file</h3>
    <form method="post" enctype="multipart/form-data" action="/upload_s/" >
        <br/>
        <input name="fileing" type="file" name="file" />
        <br/>
        <p>file describe:</p>
        <input name="filename" type="text" class="l_input account
input_with_watermark" >
        <input id="btn" type="submit" class="btn btn-lg btn-primary"
value="upload"/>
    </form>
</div>

<footer>
    <p>&copy; Company 2016 & chongshi</p>
</footer>

</body>
</html>

```

添加路径指向:

在 myweb/urls.py 文件中增加:

```

.....
from blog.views import upload, upload_save
.....
urlpatterns = [
    .....
    url(r'^upfile/$', upload),
    url(r'^upload_s/$', upload_save),
]

```

启动 server, 访问: <http://127.0.0.1:8000/upfile/>

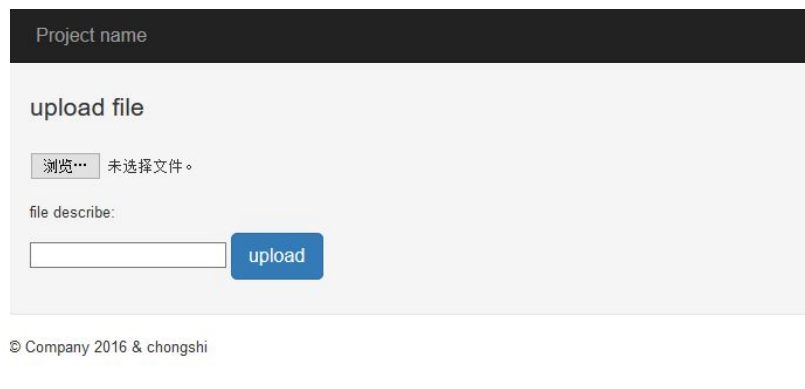


图 4.8 文件上传页面

4.5.2、加上友好提示

下接来我们希望在用户点击上传之后，能返回一个提示告诉用户上传成功了，这样做会更加友好一些。还有我们不希望用户传空的信息给我们，这样对我们没意义，不管是上传的文件还是文件描述，都不能传空信息。

打开.../blog/views.py 文件，修改 upload_s()函数逻辑：

```
# 执行文件上传
def upload_save(request):
    filename = request.POST.get('filename', '')
    fileing = request.FILES.get('fileing', '')
    if filename == '' or fileing == '':
        error = '文件与文件描述不能为空'
        return render_to_response('upload.html', {'error': error})
    else:
        upload = File()
        upload.filename = filename
        upload.fileway = fileing
        upload.save()
        return render_to_response('upload.html', {'upload_success': '上传文件成功'})
```

打开.../blog/templates/upload.html 文件，添加警告信息。


```
<div class="well">
    <h3>upload file</h3>
    <form method="post" enctype="multipart/form-data" action="/upload_s/" >
        <br/>
```

```

<input name="fileing" type="file" name="file" />
<br/>
<p>file describe:</p>
<input name="filename" type="text" class="l_input account
input_with_watermark" >
<input id="btn" type="submit" class="btn btn-lg btn-primary"
value="upload"/>
<br/>
<p class="text-warning">
    {{ upload_success }} {{ error }}
</p>
</form>
</div>

```

下面执行文件的上传。



The screenshot shows a web form titled "upload file". It contains a file selection button labeled "浏览..." with the text "未选择文件。" next to it. Below this is a text input field labeled "file describe:". To the right of the input field is a blue button labeled "upload". At the bottom of the form, a red-bordered box contains the error message "文件与文件描述不能为空" (File and file description cannot be empty).

图 4.9 文件上传错误提示



The screenshot shows the same "upload file" form as in Figure 4.9. The file selection button and "file describe:" input field are present. The blue "upload" button is visible. At the bottom of the form, a red-bordered box contains the success message "上传文件成功" (File upload successful).

图 4.10 文件上传成功

那么，现在你一定很好奇，上传的文件都在什么地方，都存到了数据库里面么？通过 Sqlitestudio 打开

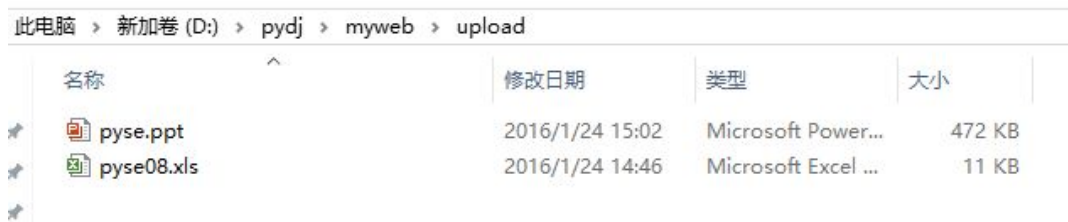
File 表查看。



	id	filename	fileway
1	1	学生名单	upload/pyse08.xls
2	2	python单元测试框架	upload/pyse.ppt

图 4.11 文件上传成功

如图表 4.11, fileway 字段中存放的是文件的路径, 并且文件本身, 那文件在哪儿呢?



名称	修改日期	类型	大小
pyse.ppt	2016/1/24 15:02	Microsoft Power...	472 KB
pyse08.xls	2016/1/24 14:46	Microsoft Excel ...	11 KB

图 4.12 上传文件的位置

如果, 你细心会发现项目下多出一个 upload/目录, 打开该目录就能看到上传的文件了, 如图 4.12。

4.5.3、制作上传文件列表

根据前面所学的知识, 为什么不把文件列显示在页面上, 这样我们上传的文件一目了然。修改.../blog/views.py 文件:

```
# 上传文件页面
def upload(request):
    files = File.objects.all()
    return render_to_response('upload.html', {'file_list': files})

# 执行文件上传
def upload_save(request):
    files = File.objects.all()
    filename = request.POST.get('filename', '')
    fileing = request.FILES.get('fileing', '')
    if filename == '' or fileing == '':
        error = '文件与文件描述不能为空'
        return render_to_response('upload.html', {'error': error, 'file_list': files})
    else:
        upload = File()
```

```

        upload.filename = filename
        upload.fileway = fileing
        upload.save()
        return render_to_response('upload.html', {'upload_success': '上传文件成功',
'file_list': files})

```

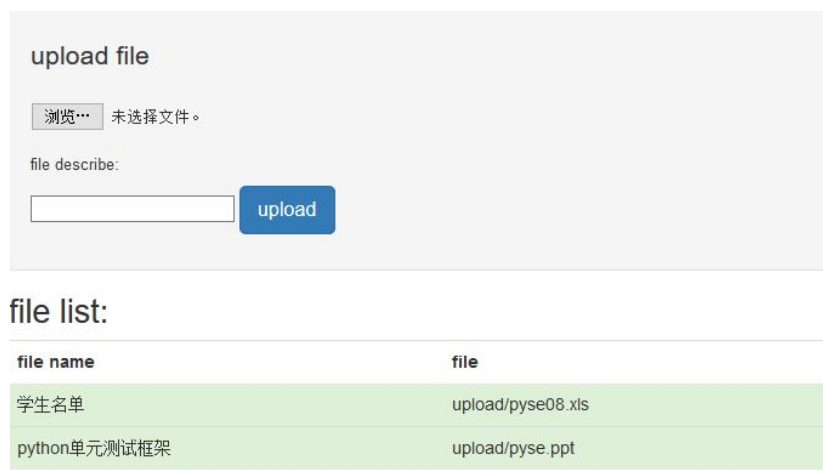
在.../blog/templates/upload.html 文件中添加:

```

.....
<h2>file list:</h2>
<table class='table'>
    <th>file name</th><th>file</th>
    {% for file in file_list %}
        <tr class='success'>
            <td>{{file.filename}}</td>
            <td>{{file.fileway}}</td>
        </tr>
    {% endfor %}
</table>
</div>
.....

```

刷新上传页面，如图 4.13。



upload file

未选择文件。

file describe:

file list:

file name	file
学生名单	upload/pyse08.xls
python单元测试框架	upload/pyse.ppt

图 4.13 上传文件列表

第五章 URL 配置

随着我们要创建的目录越来越多，那么在.../myweb/urls.py 文件下的配置也越来越多。为在项目扩展后不至于 urls.py 文件过于臃肿。我们可以将项目的 urls 与应用 urls 分开。

现在看看.../myweb/urls.py 文件的内容：

```
from django.conf.urls import url, include
from django.contrib import admin
from blog.views import index, login, login_ok, logout, bs_demo, student ,page
from blog.views import upload,upload_save

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^index/$', index),
    url(r'^login/$', login),
    url(r'^login_ok/$', login_ok),
    url(r'^logout/$', logout),
    url(r'^accounts/login/$', index),
    url(r'^bs_demo/$', bs_demo),
    url(r'^student/$', student),
    url(r'^book/$', page),
    url(r'^upfile/$', upload),
    url(r'^upload_s/$', upload_save),
]
```

在当前项目下，只有 blog 应用，随着项目的扩展，urls.py 文件下的目录指向会不断增加，给后期维护增加难度。下面把 blog 应用下的配置单独分离出来。

对.../myweb/urls.py 文件的进行修，内容如下：

```
from django.conf.urls import url, include
from django.contrib import admin
from blog import urls

urlpatterns = [
```

```
url(r'^admin/', admin.site.urls),
url(r'^blog/', include(urls)),
]
```

然后，在.../blog/目录下创建 urls.py 文件，添加如下内容：

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

app_name = 'blog'
urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^login/$', views.login, name='index'),
    url(r'^login_ok/$', views.login_ok, name='login_ok'),
    url(r'^logout/$', views.logout, name='logout'),
    url(r'^accounts/login/$', views.index, name='index'),
    url(r'^bs_demo/$', views.bs_demo, name='bs_demo'),
    url(r'^student/$', views.student, name='student'),
    url(r'^book/$', views.page, name='page'),
    url(r'^upfile/$', views.upload, name='upload'),
    url(r'^upload_s/$', views.upload_save, name='upload_save'),
]
```

通过这文件的匹配，指向 blog 应用下面的 views.py 文件里的具体函数。这么一来之前 blog 应用下的一级目录就变成了二级目录：

<http://127.0.0.1:8000/login/>

<http://127.0.0.1:8000/student/>

<http://127.0.0.1:8000/book/>

<http://127.0.0.1:8000/upfile/>

.....

修改为：

<http://127.0.0.1:8000/blog/login/>

<http://127.0.0.1:8000/blog/student/>

<http://127.0.0.1:8000/blog/book/>

<http://127.0.0.1:8000/blog/upfile/>

...

第六章 项目部署

虽然，目前的我们的项目并没有实现什么功能，其仍然包含一个 blog 展示和登录功能，一个图书列表的展示，以及一个文件的上传与展示。现在我们可以考虑将其部署到中间件中运行。

6.1 Apache 部署 Django

在此之前，我们一直使用 Django 的“runserver”命令来运行 Django 应用，但这只是我们的开发环境，当项目真正部署上线的时候这么做就不可行了，必须将我们的项目部署到特定的 web 服务器上。

6.1.1、安装 Apache

Apache 是非常有名的 web 服务器软件，如果想让我们 web 项目运行几乎离不开它。

Apache 官方网站：<http://httpd.apache.org/>

根据自己的环境，选择相应的版本进行下载。apache 官网没有 windows 64 位版本，可以通过下面的链接进行下载：win7 64 位：<http://www.apachelounge.com/download/>

下载安装完成，Apache 的目录结构如下：

bin	2014/7/19 18:27	文件夹	
cgi-bin	2014/7/19 18:27	文件夹	
conf	2014/7/19 18:27	文件夹	
error	2014/7/19 18:27	文件夹	
htdocs	2014/7/19 18:27	文件夹	
icons	2014/7/19 18:27	文件夹	
include	2014/7/19 18:27	文件夹	
lib	2014/7/19 18:26	文件夹	
logs	2014/7/19 18:27	文件夹	
manual	2014/7/19 18:27	文件夹	
modules	2014/7/19 18:26	文件夹	
ABOUT_APACHE.txt	2014/6/20 13:45	文本文档	14 KB
CHANGES.txt	2014/7/16 0:36	文本文档	165 KB
INSTALL.txt	2014/1/24 4:38	文本文档	5 KB
LICENSE.txt	2014/7/19 18:26	文本文档	38 KB
NOTICE.txt	2014/7/19 18:26	文本文档	2 KB
OPENSSE-NEWS.txt	2014/10/21 17:14	文本文档	33 KB
OPENSSE-README.txt	2014/7/19 18:26	文本文档	11 KB
README.txt	2014/1/24 0:33	文本文档	5 KB

图 6.1 Apache 目录

修改.../conf/httpd.conf 配置文件:

```
.....
ServerRoot "d:/pydj/Apache24" # 37 行
.....

Listen 127.0.0.1:8089 # 58 行 # 修改监听端口号
.....

DocumentRoot "d:/pydj/Apache24/htdocs" # 243 行
<Directory "d:/pydj/Apache24/htdocs">
.....

<Directory "d:/pydj/Apache24/cgi-bin"> # 376 行
    AllowOverride None
    Options None
    Require all granted
</Directory>
.....
```

启动.../bin/httpd.exe 程序, 通过浏览器访问: <http://127.0.0.1:8089/>

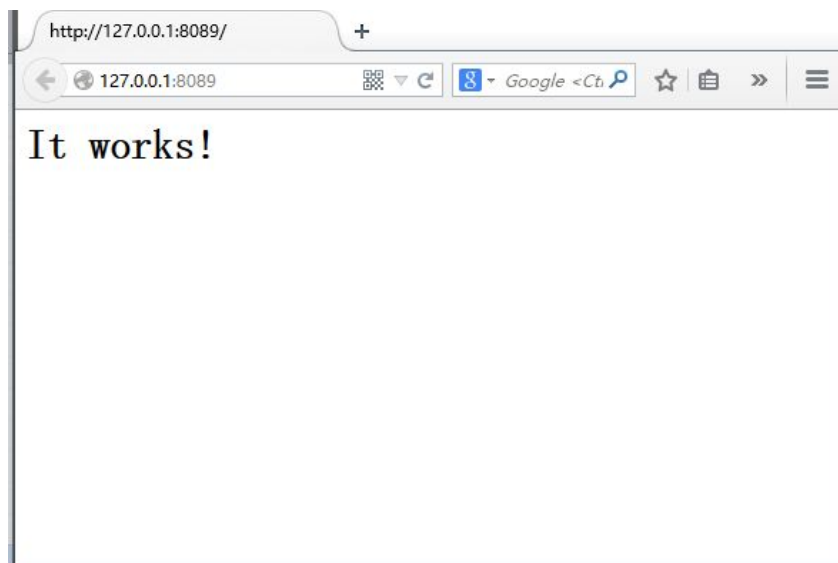


图 6.2 Apache 首页

6.1.2、安装 mod_wsgi

mod_wsgi

The aim of mod_wsgi is to implement a simple to use Apache module which can host any Python application which supports the Python WSGI interface. The module would be suitable for use in hosting high performance production web sites, as well as your average self managed personal sites running on web hosting services.

mod_wsgi 的目的是实现一个简单的使用 Apache 模块可以举办任何 Python 应用程序支持 Python 的 WSGI 接口。该模块将适用于主机的高性能生产的网站，以及一般的自我管理个人网站的网页寄存服务运行。

下载地址：http://www.lfd.uci.edu/~gohlke/pythonlibs/#mod_wsgi

如我的环境为：Windows 7 64 位、Python 3.5.0、Apache (httpd-2.4.10) 对应版本为：

mod_wsgi-4.4.22+ap24vc14-cp35-none-win_amd64.whl

.whl 就 Python 文件的一种打包格式，此处我们将该文件的后缀名改为 .zip 并解压。在 ...\\mod_wsgi-4.4.22+ap24vc14.data\\data\\ 文件里将得到一个 mod_wsgi.so 文件，将其拷贝到 Apache24\\modules\\ 目录下。

我的 Apache 和 Django 项目位置：

Apache 存放目录：D:\\pydj\\Apache24

Django 项目目录：D:\\pydj\\myweb

再次打开 Apache 的配置文件 httpd.conf，添加如下内容：

```
.....
#添加 mod_wsgi.so 模块
LoadModule wsgi_module modules/mod_wsgi.so

#指定 myweb 项目的 wsgi.py 配置文件路径
WSGIScriptAlias / D:/pydj/myweb/myweb/wsgi.py

#指定项目路径
WSGIProxyPath D:/pydj/myweb

<Directory D:/pydj/myweb/myweb>
<Files wsgi.py>
```

```
Require all granted
</Files>
</Directory>
```

参考上面的项目路径，请根据自己的实际情况进行修改。

下面查看.../myweb/wsgi.py 文件：

```
import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "myweb.settings")

application = get_wsgi_application()
```

在我们生成 Django 项目时该文件已经默认自动生成了，一般情况下不需要做任何修改。

打开.../myweb/settings.py 文件添加：

```
.....
ALLOWED_HOSTS = ['127.0.0.1', 'localhost']
```

再次双击启动 Apache24/bin/httpd.exe 程序，通过浏览器访问：<http://127.0.0.1:8089/blog/>



图 6.3 Apache 运行 Django

参考:

<https://docs.djangoproject.com/en/1.9/howto/deployment/wsgi/modwsgi/>

6.2 Nginx 部署 Django (ubuntu)

Django 的部署可以有很多方式, 采用 nginx+uwsgi 的方式是其中比较常见的一种方式。

在这种方式中, 我们的通常做法是, 将 nginx 作为服务器最前端, 它将接收 WEB 的所有请求, 统一管理请求。nginx 把所有静态请求自己来处理 (这是 NGINX 的强项)。然后, NGINX 将所有非静态请求通过 uwsgi 传递给 Django, 由 Django 来进行处理, 从而完成一次 WEB 请求。

可见, uwsgi 的作用就类似一个桥接器。起到桥梁的作用。

Linux 的强项是用来做服务器, 所以, 下面的整个部署过程我们选择在 Ubuntu 下完成。

6.2.1、安装 Nginx

Nginx 是一款轻量级的 Web 服务器/反向代理服务器及电子邮件 (IMAP/POP3) 代理服务器, 并在一个 BSD-like 协议下发行。其特点是占有内存少, 并发能力强, 事实上 nginx 的并发能力确实在同类型的网页服务器中表现较好。

Nginx 同样为当前非常流行的 web 服务器。利用其部署 Django, 我们在此也做简单的介绍。

Nginx 官网: <http://nginx.org/>

打开 ubuntu 控制台 (ctrl+alt+t) 利用 Ubuntu 的仓库安装。

```
ubuntu
```

```
fnngj@ubuntu:~$ sudo apt-get install nginx #安装
```

启动 Nginx:

```
ubuntu
```

```
fnngj@ubuntu:~$ /etc/init.d/nginx start #启动
```

```
fnggj@ubuntu:~$ /etc/init.d/nginx stop #关闭
```

```
fnggj@ubuntu:~$ /etc/init.d/nginx restart #重启
```

修改 Nginx 默认端口号，打开/etc/nginx/nginx.conf 文件，修改端口号。

```
server {  
    listen      8088;    # 修改端口号  
    server_name localhost;  
  
    #charset koi8-r;  
  
    #access_log logs/host.access.log main;  
  
    location / {  
        root    html;  
        index   index.html index.htm;  
    }  
}
```

大概在文件 36 行的位置，将默认的 80 端口号改成其它端口号，如 8088。因为默认的 80 端口号很容易被其它应用程序占用。

然后，通过上面命令重启 nginx。访问：<http://127.0.0.1:8088/>

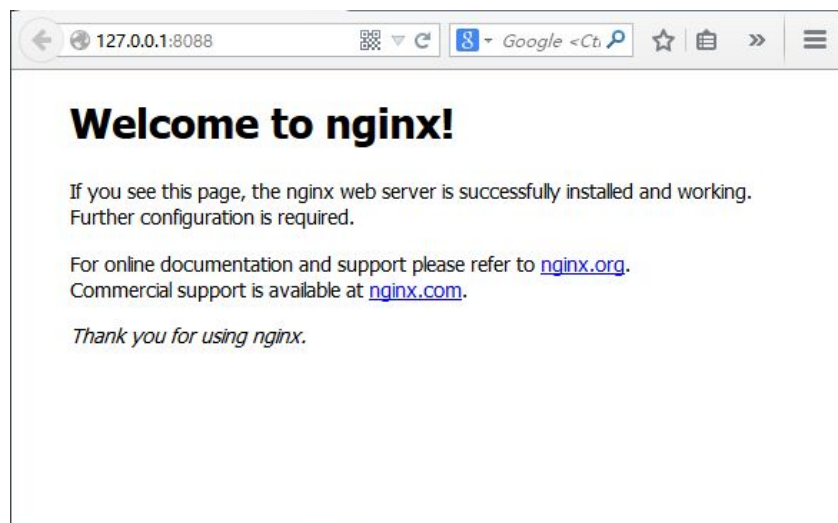


图 6.4 Nginx 默认页面

如果出现如上图，说明 Nginx 启动成功。

6.2.2、安装 uwsgi

通过 pip 安装 uwsgi。

ubuntu

```
root@ubuntu:/etc# python3 -m pip install uwsgi
```

测试 uwsgi，创建 test.py 文件：

```
def application(env, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    return [b"Hello World"]
```

通过 uwsgi 运行该文件。

ubuntu

```
fnngj@ubuntu:~/pydj$ uwsgi --http :8001 --wsgi-file test.py
```

通过，浏览器访问：<http://127.0.0.1:8001/>



图 6.5 uwsgi 当然 web 服务器

接下来配置 Django 与 uwsgi 连接。此处，假定的我的项目目录为：`/home/fnngj/pydj/myweb`

ubuntu

```
fnngj@ubuntu:~/pydj$ uwsgi --http :8001 --chdir /home/fnngj/pydj/myweb/ --wsgi-file myweb/wsgi.py
--master --processes 4 --threads 2 --stats 127.0.0.1:9191
```


- 常用选项:

http : 协议类型和端口号

processes : 开启的进程数量

workers : 开启的进程数量, 等同于 **processes** (官网的说法是 **spawn the specified number of workers / processes**)

chdir : 指定运行目录 (chdir to specified directory before apps loading)

wsgi-file : 载入 wsgi-file (load .wsgi file)

stats : 在指定的地址上, 开启状态服务 (enable the stats server on the specified address)

threads : 运行线程。由于 GIL 的存在, 我觉得这个真心没啥用。 (run each worker in prethreaded mode with the specified number of threads)

master : 允许主进程存在 (enable master process)

daemonize : 使进程在后台运行, 并将日志打到指定的日志文件或者 udp 服务器 (daemonize uWSGI)。实际上最常用的, 还是把运行记录输出到一个本地文件上。

pidfile : 指定 pid 文件的位置, 记录主进程的 pid 号。

vacuum : 当服务器退出的时候自动清理环境, 删除 unix socket 文件和 pid 文件 (try to remove all of the generated file/sockets)

6.2.3、Nginx+uwsgi+Django

接下来, 我们要将三者结合起来。

首先, 将在/uwsgi_params 文件拷贝到 myweb 项目下。也可以在这个页面下载:

https://github.com/nginx/nginx/blob/master/conf/uwsgi_params

ubuntu

```
fnngj@ubuntu:~$ cp /etc/nginx/uwsgi_params /home/fnngj/pydj/myweb/
```

然后, 在项目文件夹下创建文件 myweb_nginx.conf, 填入并修改下面内容:

这个 configuration 文件告诉 nginx 从文件系统中拉起 media 和 static 文件作为服务，同时相应 django 的 request

在/etc/nginx/sites-enabled 目录下创建本文件的连接，使 nginx 能够使用它：

```
ubuntu
```

```
fnngj@ubuntu:~$ sudo ln -s ~/home/fnngj/pydj/myweb/myweb_nginx.conf /etc/nginx/sites-enabled/
```

6.3 创建 404 页面

不管是用 Django 的“runserver”命令还是通过 Apache 运行项目时，如果访问的路径不存在，那么将会看到这样一个页面，如图 6.6。

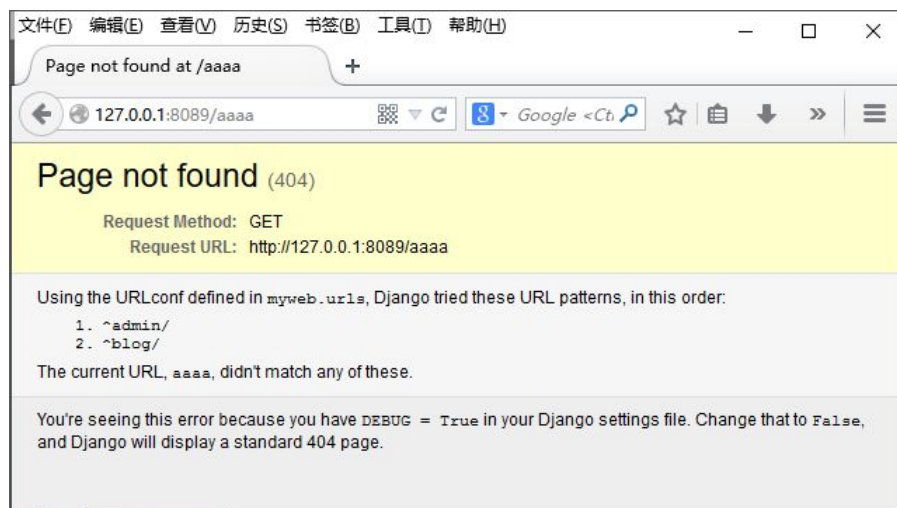


图 6.6 django debug 页面

在项目真正部署上线的时候是需要关闭 debug 状态的，这个页面不能暴露给用户看到。在.../myweb/setting.py 文件中关闭 debug 状态。

```
# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = False
```

把“Ture”改为“False”状态。

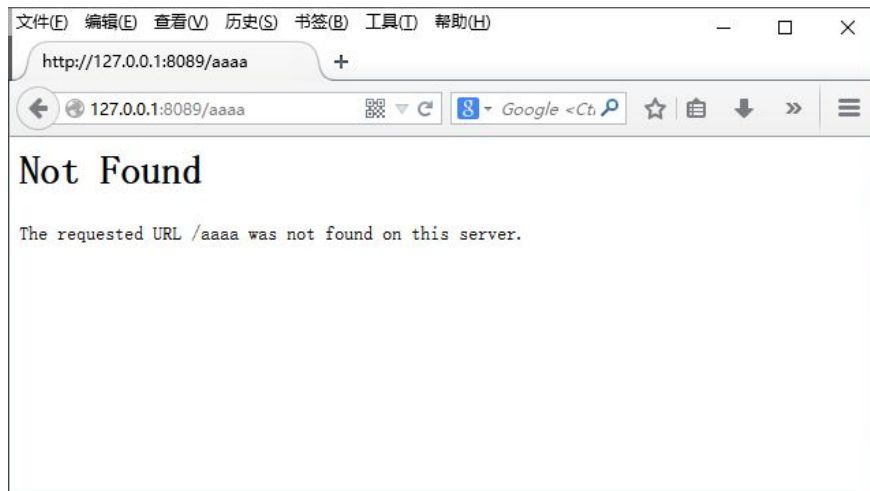


图 6.7 django 关闭 debug 页面

虽然，这个页面屏蔽了报错信息，但是，它看上去并不友好。接下来我们就来创建默认的 404 页面。在.../blog/templates/目录下创建 404.html 页面

```
<html lang="zh-CN">
<head>
    {% load bootstrap3 %}
    {% bootstrap_css %}
    {% bootstrap_javascript %}
    {% bootstrap_messages %}
</head>
<body>

    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <a class="navbar-brand" href="#">Home</a>
            </div>
        </div>
    </nav>

    <br><br><br><br><br><br><br><br>
    <div class="container">
        <div class="row">
            <h1>404!! file not found.</h1>
        </div>
    </div>
```

```
<br><br><br><br><br><br><br><br>

<div class="container">
  <footer>
    <p>&copy; Company 2016 & chongshi</p>
  </footer>
</div>
</body>
</html>
```

此时，再来访问一个不存在的路径。

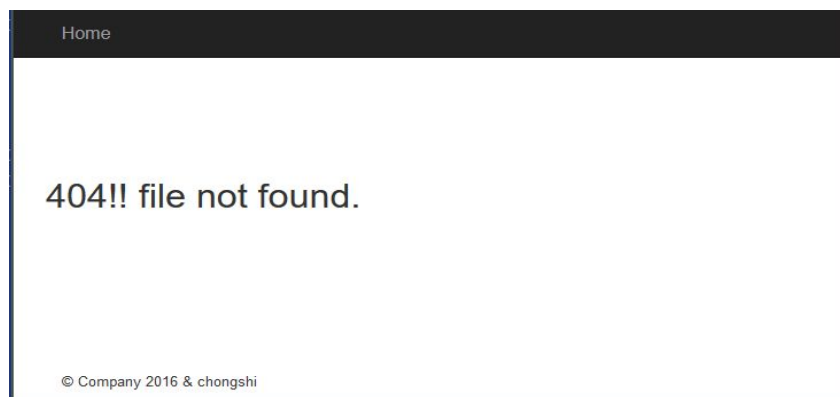


图 6.8 默认 404 页面

如图 6.8，服务器告诉我们 not found ！显然这个页面要友好些。

第七章 Django 测试

关于，Django 的 web 的开发，我们先告一段落，当然，对于 web 开发来说，还有很多问题值得我们继续学习，不管是前端页面的开发，还是后端数据的处理。但这不是本书的重点，或者并不是我擅长和重点学习的内容。我们的重点是如何对 web 项目进行测试。当然，本书并不是教你如何进行功能测试。而是打开这个封闭的黑盒子，进行接口和单元测试。

这一章我们将探讨 Django 如何进行单元测试。

7.1 Testing in Django

对于 web 开发人员来说，自动化测试是一个非常有用的发现 bug 的手段。您可以使用一个测试集合或一个测试套件来解决，或避免一些问题：

当你编写新的代码，你可以使用测试来验证你的代码是否按预期工作。

当你重构或修改旧代码，你可以使用测试，以确保您的更改不会影响您的应用程序的行为意外。

测试一个 Web 应用是一项复杂的任务，因为在 Web 应用程序是由逻辑几层 - 从 HTTP 的级请求处理，以形成验证和处理，以模板渲染。随着 Django 的测试执行框架和各种实用工具，可以模拟请求，插入测试数据，检查您的应用程序的输出，一般检查你的代码是做什么的，应该做的事情。

Django 的是使用内置于 Python 标准库中的 unittest 单元测试框架。您也可以使用任何其他的 Python 测试框架; Django 提供了一个 API 和工具，可以融合其它的单元测试框架。

7.1.1、A simple example

Django 的单元测试使用 Python 标准库模块：unittest。该模块定义使用基于类的方法测试。在我们创建 Django 应用时，默认已经帮我们创建了 tests.py 文件，打开.../blog/tests.py 文件，编写如下代码：

```
from django.test import TestCase
from blog.models import Author
```

```
# Create your tests here.
class AuthorTestCase(TestCase):

    def setUp(self):
        Author.objects.create(first_name="zhang", last_name="san",
email="zhangsan@gmail.com")
        Author.objects.create(first_name="li", last_name="si",
email="lisi@gmail.com")

    def test_add_author_email(self):
        """test ada auhor email"""
        zhang = Author.objects.get(first_name="zhang")
        li = Author.objects.get(first_name="li")
        self.assertEqual(zhang.email, "zhangsan@gmail.com")
        self.assertEqual(li.email, "lisi@gmail.com")
```

我们以测试出版社作者表为例，如果不清楚改表的，请回到本书的第三章进行学习。

分析一下测试用例的实现：

首先，创建 `AuthorTestCase` 类，继承 `TestCase` 测试类。

然后，在 `setUp()` 初始化方法中，创建 “zhangsan” 和 “lisi” 两条用户数据。

最后，通过 `test_add_author_email()` 测试方法，获取这两条用户信息，并判断他们的 `email` 是否正确。

执行测试用例，切换到项目的根目录下，通过 `manage.py` 运行测试。

cmd

```
D:\pydj\myweb>python3 manage.py test
```

```
Creating test database for alias 'default'...
```

```
.
```

```
-----
Ran 1 test in 0.002s
```

```
OK
```

```
Destroying test database for alias 'default'...
```

你一定很好奇，“test”命令是如何找到测试用例并执行的？这就要求我们创建测试用例时要遵循一定的规范了，首先创建的测试文件必须以“test”开发，测试文件中的测试类必须继承 `TestCase` 测试类，测试类

下面的测试方法必须以“test”开头。

当用例执行失败时又是怎样的呢？ 修改用例中的预期结果，把"zhangsan@gmail.com"改为"aaa@gmail.com"，再次执行测试。

```
cmd
```

```
D:\pydj\myweb>python3 manage.py test
```

```
Creating test database for alias 'default'...
```

```
F
```

```
=====
```

```
FAIL: test_add_author_email (blog.tests.AuthorTestCase)
```

```
test add auhor email
```

```
-----
```

```
Traceback (most recent call last):
```

```
  File "D:\pydj\myweb\blog\tests.py", line 16, in test_add_author_email
```

```
    self.assertEqual(zhang.email, "aaa@gmail.com")
```

```
AssertionError: 'zhangsan@gmail.com' != 'aaa@gmail.com'
```

```
- zhangsan@gmail.com
```

```
? -- --- ^
```

```
+ aaa@gmail.com
```

```
?    ^
```

```
-----
```

```
Ran 1 test in 0.002s
```

```
FAILED (failures=1)
```

```
Destroying test database for alias 'default'...
```

从上面的提示信息中，我们很容易就可以找到错误的原因。

7.1.2、Run test case

当编写完测试，最简单的方式是通过 `manage.py` 中 “test” 命令来直接执行所有测试。但是编写的测试用例越来越多的时候，测试运行的情况就复杂起，比如要指定特定的测试模块，或测试类，又或者想执行测试文件名包含了 “test” 的文件。接下来我们就要讨论这些特定的需求。

在此之前我们要重新准备好用例，以便更简单的说明问题。

首先修改.../blog/tests.py 文件如下：

```
from django.test import TestCase
from blog.models import Author

# Create your tests here.
class AuthorTestCase(TestCase):

    def setUp(self):
        Author.objects.create(first_name="zhang", last_name="san",
email="zhangsan@gmail.com")
        Author.objects.create(first_name="li", last_name="si",
email="lisi@gmail.com")

    def test_add_author_zhangsan_email(self):
        """test ada auhor email"""
        zhang = Author.objects.get(first_name="zhang")
        self.assertEqual(zhang.email, "zhangsan@gmail.com")

    def test_add_author_lisi_email(self):
        """test ada auhor lisi email"""
        li = Author.objects.get(first_name="li")
        self.assertEqual(li.email, "lisi@gmail.com")
```

这里做了简单的拆分，把一个测试用例的两个断言，分别拆分到两个测试用例里面。

接下来创建.../blog/tests2.py 文件：

```
from django.test import TestCase
from blog.models import Publisher
```



```
# Create your tests here.
class PublisherTestCase(TestCase):

    def setUp(self):
        Publisher.objects.create(name='zhongxin',
                                address='Xinyuan South Road, Chaoyang District,
Beijing on the 6th',
                                city='beijing',
                                state_province='China',
                                country='beijing',
                                website='http://www.publish.citic.com/')

    def test_add_publisher_city(self):
        """test add publisher city"""
        zhongxin = Publisher.objects.get(name="zhongxin")
        self.assertEqual(zhongxin.city, "beijing")
```

在该测试文件中，创建一条测试 Publisher（出版社）的用例。

运行 blog 应用下的所有测试用例：

cmd

```
D:\pydj\myweb>python3 manage.py test blog
```

```
Creating test database for alias 'default'...
```

```
...
```

```
-----
Ran 3 tests in 0.004s
```

```
OK
```

```
Destroying test database for alias 'default'...
```

运行 blog 应用下的 tests.py 测试文件：

cmd

```
D:\pydj\myweb>python3 manage.py test blog.tests
```

Creating test database for alias 'default'...

..

Ran 2 tests in 0.003s

OK

Destroying test database for alias 'default'...

运行 blog 应用 tests2.py 测试文件下的 PublisherTestCase 测试类:

cmd

D:\pydj\myweb>python3 manage.py test blog.tests2.PublisherTestCase

Creating test database for alias 'default'...

.

Ran 1 test in 0.002s

OK

Destroying test database for alias 'default'...

下面执行 PublisherTestCase 测试类下面的 test_add_publisher_city 测试方法（用例）：

cmd

D:\pydj\myweb>python3 manage.py test blog.tests2.PublisherTestCase.test_add_publisher_city

Creating test database for alias 'default'...

.

Ran 1 test in 0.002s

OK

Destroying test database for alias 'default'...

除此之外，我们还可以使用 -p （或 --pattern）参数指匹配某类测试文件：

cmd

```
D:\pydj\myweb>python3 manage.py test -p="test*.py"
Creating test database for alias 'default'...
```

...

```
-----
Ran 3 tests in 0.004s
```

OK

```
Destroying test database for alias 'default'...
```

指定匹配运行的测试文件--->test*.py，必须以“test”开头，以“.py”结尾。

7.1.3、Assertion methods

因为 Django 默认使用 unittest 单元测试框架，所以，这里有必要学习该框架中所提供的常用的断言方法。

在执行用例的过程中，最终用例是否执行通过，是通过判断测试得到的实际结果与预期结果相等。unittest 框架的 TestCase 类提供下面这些方法用于测试结果的判断。

方 法	检 查	版 本
assertEqual(a, b)	a == b	
assertNotEqual(a, b)	a != b	
assertTrue(x)	bool(x) is True	
assertFalse(x)	bool(x) is False	
assertIs(a, b)	a is b	3.1
assertIsNot(a, b)	a is not b	3.1
assertIsNone(x)	x is None	3.1
assertIsNotNone(x)	x is not None	3.1
assertIn(a, b)	a in b	3.1
assertNotIn(a, b)	a not in b	3.1
assertIsInstance(a, b)	isinstance(a, b)	3.2
assertNotIsInstance(a, b)	not isinstance(a, b)	3.2

为了演示断言方法的使用，我们重新创建.../blog/tests3.py 文件。

```
=====
```

```
- assertEquals(first, second, msg=None)
```

```
- assertNotEqual(first, second, msg=None)
```

```
=====
```

断言第一个参数和第二个参数是否相等，如果不相等则测试失败。msg 为可选参数，用于定义测试失败时所打印的信息。

```
from django.test import TestCase

class MyTest(TestCase):

    def setUp(self):
        number = input("Enter a number:")
        self.number = int(number)

    def test_case(self):
        self.assertEqual(self.number, 10, msg="Your input is not 10!")
```

在 setUp() 方法中要求用户输入一个数，在 test_case() 中通过 assertEquals() 比较输入的数是否等于 10，如果不相等则输出 msg 中定义的提示信息。执行结果如下。

cmd

```
D:\pydj\myweb>python3 manage.py test blog.tests3.MyTest.test_case
```

```
Creating test database for alias 'default'...
```

```
Enter a number:11    #手动输入数字
```

```
F
```

```
=====
=
```

```
FAIL: test_case (blog.tests2.MyTestCase)
```

```
-----
Traceback (most recent call last):
```

```
File "D:\pydj\myweb\blog\tests3.py", line 11, in test_case
```

```
self.assertEqual(self.number, 10, msg="Your input is not 10!")
```

```
AssertionError: 11 != 10 : Your input is not 10!
```

```
-----
Ran 1 test in 2.402s
```

```
FAILED (failures=1)
```

```
Destroying test database for alias 'default'...
```

从执行结果看到，输入了一个 11，显然与预期的 10 不相等，msg 所定义的提示信息告诉我们 “Your input is not 10!”。

assertNotEqual() 与 assertEquals() 相反，它用于断言第一个参数与第二个参数是否不相等，如果相等则测

试失败。

```
=====
- assertTrue(expr, msg=None)
- assertFalse(expr, msg=None)
=====
```

测试表达式是 `true`（或 `false`）。

下面来实现判断一个数是否为质数的功能，所谓的质数（又叫素数）是指只能被 1 和它本身整除的数。

```
from django.test import TestCase

# 用于判断质数
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

class Test(TestCase):

    def setUp(self):
        print("test start")

    def test_case(self):
        self.assertTrue(is_prime(7), msg="Is not prime!")

    def tearDown(self):
        print("test end")
```

创建 `is_prime()` 函数来实现质数的判断，当得到一个数字 `n` 后，首先判断它是否小于等于 1，如果小于等于 1，则直接返回 `False`；如果大于 1，则对其进行循环判断，若能整除 2 到其自身之间的任意一个数，则不为质数，返回 `False`，否则返回 `True`。

接下来，在调用 `is_prime()` 函数时分别传不同的值来执行测试用例，在上面的例子中传值为 7，显然是一个质数，所以通过 `assertTrue()` 断言得到的结果为 `True`。

```
=====
- assertIn(first, second, msg=None)
- assertNotIn(first, second, msg=None)
=====
```

断言第一个参数是否在第二个参数中，反过来讲，第二个参数是否包含第一个参数。

```
from django.test import TestCase

class TestIn(TestCase):

    def setUp(self):
        print("test start")

    def test_case(self):
        a = "hello"
        b = "hello world"
        self.assertIn(a, b, msg="a is not in b")

    def tearDown(self):
        print("test end")
```

这个很好理解，定义字符串 a 为 “hello”，b 为 “hello world”。通过 assertIn 判断 b 是否包含 a，如果不包含则打印 msg 定义的信息。

```
=====
- assertIs(first, second, msg=None)
- assertIsNot(first, second, msg=None)
=====
```

断言第一个参数和第二个参数是否为同一对象。

```
=====
- assertIsNone(expr, msg=None)
- assertIsNotNone(expr, msg=None)
=====
```

断言表达式是否为 None 对象。

```
=====
- assertIsInstance(obj, cls, msg=None)
- assertNotIsInstance(obj, cls, msg=None)
=====
```

断言 obj 是否为 cls 的一个实例。

在 unittest 中还提供了其他检查比较的方法，因为不常用，所以不再一一进行介绍。读者可参考 Python 官方文档 unittest 章节进行学习。

7.2 Testing Tools

Django 提供一小部分的编写测试时派上用场的工具。

7.2.1、The test client

测试客户端是一个 Python 类，它充当一个虚拟的网络浏览器，让您测试您的视图层，并与你的 Django 的应用程序编程方式进行交互。

客户端测试可以做的事情：

- 模拟 "Get" 和 "Post" 请求, 观察响应结果--从 HTTP(headers, status codes) 到页面内容。
- 检查重定向链(如果有的话)，在每一步检查 URL 和 status code。
- 用一个包括特定值的模板 context 来测试一个 request 被 Django 模板渲染。

进入 Django Shell 模式：

```
cmd.exe
```

```
D:\pydj\myweb>python3 manage.py shell
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
```

>>>

建立测试环境:

cmd.exe

```
>>> from django.test.utils import setup_test_environment
>>> setup_test_environment()
```

导入测试类:

cmd.exe

```
>>> from django.test import Client
>>> c = Client()
```

POST 请求:

cmd.exe

```
>>> response = c.post('/blog/login/', {'username':'fnngj',
                                         'password':'heng198876'})

>>> response.status_code
302
```

`post()`方法用于发起 `post` 请求。

“`/blog/login/`”为用户登录的路径。

`{'username':'fnngj','password':'heng198876'}` 字典中的内容为用户登录的用户名密码。

`status_code` 用于得到返回码。

当然，我们也可以通过 `firebug` 来捕捉登录请求与之做参考。如图 7.1。

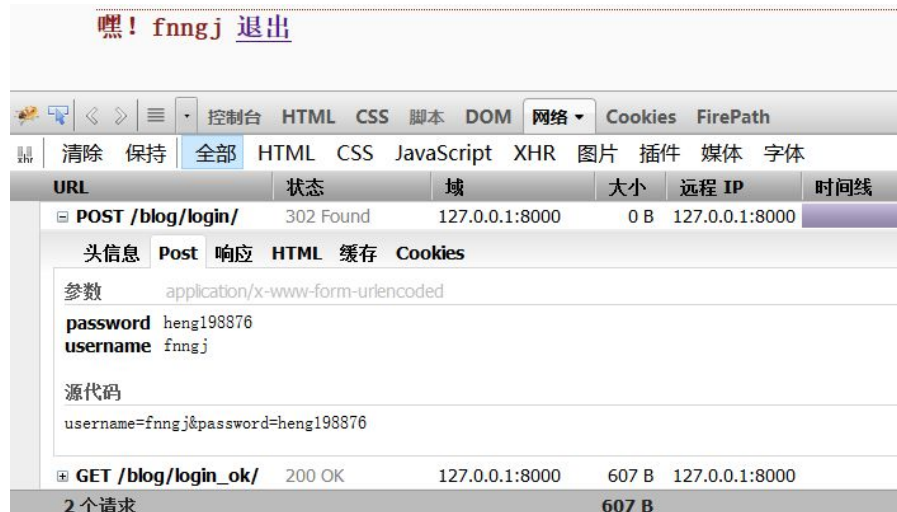


图 7.1 用户登录 post 请求

GET 请求:

cmd.exe

```
>>> response = c.get('/blog/')
>>> response.status_code
200
>>> response.content
b'<html><head>\n    <style type="text/css"> \n
body{color:#0A0A0A;background:#FAFAFA;padding:0 5em;margin:0}\n
h1{padding:2em 1em;background:#7171C6}\n
h2{color:#8B1A1A;border-top:1px dotted #8B2500;margin-top:2em}\n
p{margin:1em 0}\n    </style>\n</head>\n<body>\n    <h1>Django blog</h1>\n
\n\n    <h2>login<h2>\n    <form method="post" action=\'/blog/login/\'>\n
<input name="username" type="text" placeholder="username">\n        <input
name="password" type="password" placeholder="password">\n        <button
id="btn" type="submit" >login</button>\n        <br>\n    </form>\n\n    \n
<h2>hello Django</h2>\n        <p>Jan. 5, 2016, 11:03 p.m.</p>\n
<p>\xe8\xbf\x99\xe6\x98\xaf\xe6\x88\x91\xe4\xbb\xac\xe7\xac\xac\xe4\xb8\x80\xe6
\xac\xa1\xe7\x94\xa8Django\xe5\xbc\x80\xe5\x8f\x91blog\xe5\xba\x94\xe7\x94\xa8
\xef\xbc\x8c\xe8\xb7\x9f\xe7\x9d\x80\xe8\x99\xab\xe5\xb8\x88\xe4\xb8\x80\xe6\xad
\xa5\xe4\xb8\x80\xe6\xad\xa5\xe7\x9a\x84\xe5\x88\x9b\xe5\xbb\xba\xe7\x9a\x84\xe
8\x87\xaa\xe5\xb7\xb1\xe7\x9a\x84blog\xe5\x90\xa7\xef\xbc\x81</p>\n
\n\n</body>\n</html>'
```

get()方法用于发起 get 请求。

“/blog/” 表示获取 blog 首页。

content 方法得到整个页面的代码。

同样，我们也可以通过 firebug 来捕捉 get 请求并查看响应的信息与 content 获取的内容是否一样。如图 7.2。



图 7.2 get 请求的响应

注意：请求网页时，使用 path 而不是整个 domain。

```
cmd.exe
```

```
>>> response = c.get('/blog/')
```

是正确的。

```
cmd.exe
```

```
>>> response = c.get('http://127.0.0.1:8000/blog/')
```

是错误的。

test client 不适合操作不是由 Django 建立的网站。所以，请求其它网页时，请使用 python 的标准库--urllib 或者 urllib2。

7.2.2、Making Requests

使用 `django.test.client.Client()` 来执行请求。

```
class Client(RequestFactory):
    """
    A class that can act as a client for testing purposes.
    .....

    """
    def __init__(self, enforce_csrf_checks=False, **defaults):
        super(Client, self).__init__(**defaults)
        self.handler = ClientHandler(enforce_csrf_checks)
        self.exc_info = None
    .....
```

可以使用关键字参数来指定默认的请求报头：

cmd.exe

```
>>> c = Client(HTTP_USER_AGENT='Mozilla/5.0')
```

请求头信息	原始头信息
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding	gzip, deflate
Accept-Language	zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3
Cache-Control	max-age=0
Connection	keep-alive
Cookie	csrftoken=XT0cyB2o3BMHQM2LhhdPaLPFeMABWzW: sessionId=qfhwu86bse2ypmdzsr5coxv2w38fc4v9
Host	127.0.0.1:8000
User-Agent	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:29.0) Gecko/20100101 Firefox/29.0

图 7.3 请求头信息

为什么浏览器身份 User-Agent 有“Mozilla”字样，感兴趣的读者可以查阅相关资料。

get 方法实现：

```
def get(self, path, data=None, follow=False, secure=False, **extra):
    """
    Requests a response from the server using GET.
    """
```

```

response = super(Client, self).get(path, data=data, secure=secure, **extra)
if follow:
    response = self._handle_redirects(response, **extra)
return response

```

cmd.exe

```

>>> c = Client()
>>> c.get('/blog/login/', {'username': 'fnngj', 'password': 'heng198876'})

```

相当于在 url 中执行 get:

/blog/login/?username=fnngj&password=heng198876

post 方法实现:

```

def post(self, path, data=None, content_type=MULTIPART_CONTENT,
        follow=False, secure=False, **extra):
    """
    Requests a response from the server using POST.
    """
    response = super(Client, self).post(path, data=data,
                                       content_type=content_type,
                                       secure=secure, **extra)

    if follow:
        response = self._handle_redirects(response, **extra)
    return response

```

cmd.exe

```

>>> c = Client()
>>> with open('d://abc.txt') as fp:
...     c.post('/blog/upload_s/', {'filename': 'abc file', 'fileing': fp})
...
<HttpResponse status_code=200, "text/html; charset=utf-8">

```

通过上面的代码实现文件的上传, 如果想理解这段实现, 还要回过头去看看文件上传的表单。

```

<form method="post" enctype="multipart/form-data" action="/blog/upload_s/" >
<br/>

```

```
<input name="fileing" type="file" name="file" />
<br/>
<p>file describe:</p>
<input name="filename" type="text" class="l_input account
input_with_watermark" >
<input id="btn" type="submit" class="btn btn-lg btn-primary" value="upload"/>
<br/>
<p class="text-warning">
    {{ upload_success }}  {{ error }}
</p>
</form>
```

首先，打开 `open()` 打开上传的文件，并重命名 `fp`，调用上传接口 `"/blog/upload_s/"`，`filename` 指定上传的文件名，`fileing` 指定上传的文件。

关于，测试工具的讨论，暂时到这里，参考 Django 官方文档。

<https://docs.djangoproject.com/en/1.9/topics/testing/tools/>

第八章 接口相关概念

从本章开始，我们暂时抛开 Django 的学习，来学习有关接口的知识。

接口测试：

接口测试是测试系统组件间接口的一种测试。接口测试主要用于检测外部系统与系统之间以及内部各个子系统之间的交互点。测试的重点是要检查数据的交换，传递和控制管理过程，以及系统间的相互逻辑依赖关系等。

——百度百科

8.1 分层的自动化测试

测试金字塔的概念由敏捷大师 Mike Cohn 在他的《*Succeeding with Agile*》一书中首次提出，如图8.1所示。他的基本观点是：我们应该有更多的低级别的单元测试，而不仅仅是通过用户界面运行的高层的端到端的测试。

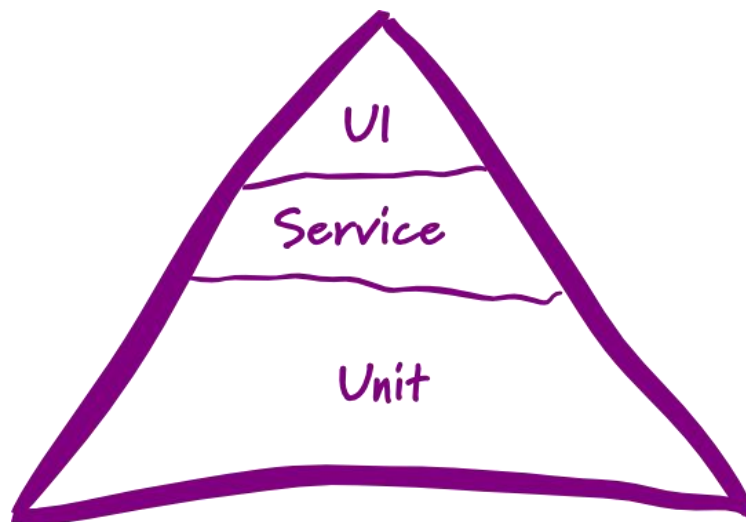


图 8.1 测试金字塔

Martin Fowler 大师在测试金字塔模型的基础上提出分层自动化测试的概念，如图1.4所示。在自动化测试之前加了一个“分层”的修饰，用来区别于“传统的”自动化测试。那么什么是传统的自动化测试？为何要提倡分层自动化测试的思想呢？

所谓传统的自动化测试我们可以理解为基于产品 UI 层的自动化测试，它是将黑盒功能测试转化为由程序或工具执行的一种自动化测试。

在目前的大多数研发组织当中，都存在开发与测试团队割裂（部门墙）、质量职责错配（测试主要对质量负责）的问题，在这种状态下，测试团队的一个“正常”反应就是试图在测试团队能够掌控的黑盒测试环节进行尽可能全面的覆盖，甚至是尽可能全面的 UI 自动化测试。

这可能会导致两个恶果：一是测试团队规模的急剧膨胀；二是所谓的全面 UI 自动化测试运动。因为 UI 是非常易变的，所以 UI 自动化测试维护成本相对高昂。

分层自动化测试倡导的是从黑盒（UI）单层到黑白盒多层的自动化测试体系，从全面黑盒自动化测试到对系统的不同层次进行自动化测试，如图 8.2 所示。

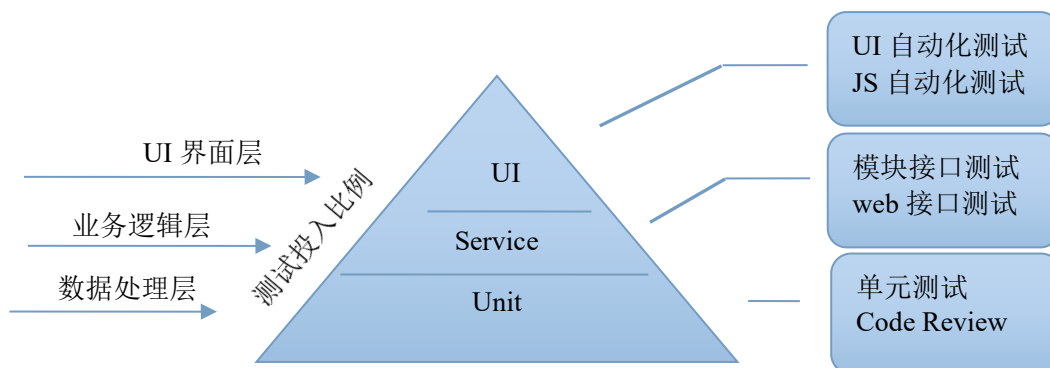


图 8.2 分层自动化测试

单元自动化测试

单元测试是指对软件中的最小可测试单元进行检查和验证。对于单元测试中单元的含义，一般来说，要根据实际情况去判定其具体含义，如 C 语言中单元指一个函数、Java 中单元指一个类、图形化的软件中单元可以指一个窗口或一个菜单等。总的来说，单元就是人为规定的最小的被测功能模块。规范地进行单元测试就需要借助单元测试框架，如 java 语言的 Junit、TestNG，C#语言的 NUnit，以及 Python 语言的 unittest、pytest 等，目前几乎所有的主流语言都会有其相应的单元测试框架。

Code Review 中文翻译为代码评审或代码审查，是指在软件开发过程中，通过对源代码进行系统性检查的过程。通常的目的是查找系统缺陷，保证软件总体质量和提高开发者自身水平。与 Code Review 相关的插件与工具有很多，例如 Java 语言中基于 Eclipse 的 Reviewclipse 和 Jupiter、主要针对 Python 语言的 Review Board 等。

接口自动化测试

根据笔者的理解，Web 应用的接口测试大体分为两类：模块接口测试和 Web 接口测试。

（1）模块接口测试，主要测试模块之间的调用与返回。当然，我们也可以将其看作是单元测试的基础。它主要强调对一个类方法或函数的调用，并对返回结果的验证，所用到的测试工具与单元测试相同。

(2) Web 接口测试又可分为两类：服务器接口测试和外部接口测试。

- 服务器接口测试：指测试浏览器与服务器的接口。我们知道 Web 开发一般分前端和后端，前端开发人员用 HTML/CSS/JavaScript 等技术，后端开发人员用 PHP/Java/C#/Python/Ruby 等各种语言。用户的操作是在前端页面上，需要后端提供服务器接口，把前端通过调用这些接口来获得需要的数据，通过 HTTP 协议来实现前后端的数据传递。
- 外部接口测试：指调用的接口由第三方系统提供。典型的例子就是第三方登录，例如新上线的产品为了免于新用户注册账号的麻烦会提供第三方登录，那么用户在登录的时候调用的就是第三方登录的接口，用户登录信息的验证由第三方完成，并返回给当前系统是否验证通过。

当然，接口测试也有相应的类库或工具，例如测试 HTTP 的有 HttpUnit、Postman 等。

UI 自动化测试

UI 层是用户使用该产品的入口，所有功能都通过这一层提供并展示给用户，所以大多测试工作都集中在这一层进行。为了减轻这一层的测试人力和时间成本，早期的自动化测试工具主要针对该层设计。目前主流的测试工具有 UFT、Watir、Robot Framework、Selenium 等。

除了 UI 层所展示的功能外，前端代码同样需要进行测试。在前端开发中最主要的莫过于 JavaScript 脚本语言，而 QUnit 就是针对 JavaScript 的一个强大的单元测试框架。

图1.4中的测试金字塔映射了不同测试阶段所投入的自动化测试的比例，UI 层被放到了塔尖，这也说明 UI 层应该投入较少的自动化比例。如果系统只关注 UI 层的自动化测试并不是一种明智的做法，因为其很难从本质上保证产品的质量。如果妄图实现全面的 UI 层的自动化测试，那么需要投入大量的人力和时间，然而，最终获得的收益可能远低于所投入的成本。因为对于一个系统来讲，越接近用户其越容易变化，为了不断适应这种变化就必然需要投入更多的成本。

既然 UI 层的自动化测试这么劳民伤财，那么我们是不是只做单元测试与接口测试就可以了呢？答案是否定的，因为不管什么样的产品，最终呈现给用户的是 UI 层的功能，所以产品才需要招聘大量的测试人员进行 UI 层的功能测试。也正是因为测试人员在 UI 层投入了大量的时间与精力，所以我们才有必要通过自动化的方式帮助测试人员解放部分重复的工作。所以，笔者更提倡“半自动化”的开展测试工作，把可以自动化测试的工作交给工具或脚本完成，这样测试人员就可以把更多的精力放在更重要的测试工作上，例如探索性测试等。

至于在金字塔中每一层测试的投入比例则要根据实际的产品特征来划分。在《Google 测试之道》一书中提到，Google 对产品测试类型划分为：小测试、中测试和大测试，采用 70%（小）/20%（中）/10%（大）的比例，大体对应测试金字塔中的 Unit、Service 和 UI 层。

在进行自动化测试中最担心的是变化，因为变化会直接导致测试用例的运行失败，所以需要对自动化脚本进行不断调整。如何控制失败，降低维护成本是对自动化测试工具及人员能力的挑战。反过来讲，一份永远都运行通过的自动化测试用例已经失去了它存在的价值。

8.2 编程语言的接口定义

在大多面向对象的编程语言中都提供了 Interface（接口）的概念。既然要讲接口，这里先简单介绍一下面向对象编程语言中的 Interface。

8.2.1、Java 中的 Interface

在 Java 中定义接口使用 interface 关键字来声明，可以看做是一种特殊的抽象类，可以指定一个类必须做什么，而不是规定它如何去做。

为什么使用接口？

大型项目开发中，可能需要从继承链的中间插入一个类，让它的子类具备某些功能而不影响它们的父类。例如 A -> B -> C -> D -> E，A 是祖先类，如果需要为 C、D、E 类添加某些通用的功能，最简单的方法是让 C 类再继承另外一个类。但是问题来了，Java 是一种单继承的语言，不能再让 C 继承另外一个父类了，只到移动到继承链的最顶端，让 A 再继承一个父类。这样一来，对 C、D、E 类的修改，影响到了整个继承链，不具备可插入性的设计。

接口是可插入性的保证。在一个继承链中的任何一个类都可以实现一个接口，这个接口会影响到此类的所有子类，但不会影响到此类的任何父类。此类将不得不实现这个接口所规定的方法，而子类可以从此类自动继承这些方法，这时候，这些子类具有了可插入性。

我们关心的不是哪一个具体的类，而是这个类是否实现了我们需要的接口。

接口提供了关联以及方法调用上的可插入性，软件系统的规模越大，生命周期越长，接口使得软件系统的灵活性和可扩展性，可插入性方面得到保证。

接口在面向对象的 Java 程序设计中占有举足轻重的地位。事实上在设计阶段最重要的任务之一就是设计出各部分的接口，然后通过接口的组合，形成程序的基本框架结构。

所以简单总结其用途为：实现类的多继承，以解决 Java 只能单继承，不支持多继承的问题。

下面通过例子介绍 Java 中接口的使用。

定义接口：

```
IAnimal.java
```

```
package mypor.interfaces.demo;
```

```
public interface IAnimal {
```

```
    public String Behavior(); //行为方法，描述各种动物的特性  
}
```

实现接口（一）：

Dog.java

```
package mypor.interfaces.demo;  
import mypor.interfaces.demo.IAnimal;  
  
//类：狗  
public class Dog implements IAnimal{  
  
    public String Behavior()  
    {  
        String ActiveTime = "我晚上睡觉,白天活动";  
  
        return ActiveTime;  
    }  
}
```

实现接口（二）：

Dog.java

```
package mypor.interfaces.demo;  
import mypor.interfaces.demo.IAnimal;  
  
//类：猫  
public class Cat implements IAnimal{  
  
    public String Behavior()  
    {  
        String ActiveTime = "我白天睡觉,晚上捉老鼠。";  
  
        return ActiveTime;  
    }  
}
```

测试接口的实现：

Dog.java

```
package mypor.interfaces.demo;  
import mypor.interfaces.demo.Dog;
```

```
import mypor.interfaces.demo.Cat;

public class Test {

    public static void main(String[] args) {
        //调用 dog 和 cat 的行为
        Dog d = new Dog();
        Cat c = new Cat();
        System.out.println(d.Behavior());
        System.out.println(c.Behavior());
    }
}
```

注意，这里的测试，并不是测试的接口，因为接口本身只是简单的定义，没什么可测试的，这里真正所测试的是继承接口的类，或者叫已经实例化的对象。

8.2.2、Python 中的 Zope.interface

那么读者好奇，Python 中是否有接口（Interface）的概念，Python 本身并不提供接口的创建和使用，但是我们可以通过第三方扩展库来使用接口，那就是 Zope.interface。

下载地址：<https://pypi.python.org/pypi/zope.interface>

来看个普通的例子：

```
class Host(object):

    def goodmorning(self, name):
        """Say gooa morning to guests"""
        return "Good morning, %s!" % name

if __name__ == '__main__':
    h = Host()
    hi = h.goodmorning('zhangsan')
    print(hi)
```

下面在这个例子的基础中使用接口：

```

from zope.interface import Interface
from zope.interface import implements

# 定义接口
class IHost(Interface):

    def goodmorning(self, guest):
        """Say gooa morning to guest"""

class Host(object):
    implements(IHost) # 实现接口

    def goodmorning(self, guest):
        """Say gooa morning to guests"""
        return "Good morning, %s!" % guest

if __name__ == '__main__':
    h = Host()
    hi = h.goodmorning('zhangsan')
    print(hi)

```

参考: <http://muthukadan.net/docs/zca.html#an-example>

当然，本书的重点并不是讨论面向对象编程语言（Interface）接口的使用，之所以花一小节介绍，是希望读者了解有这么个概念，并且把它与我们要测试的接口进行区分。

8.3 我所了解的接口测试

我所了解的模块接口测试大体分为两类：模块接口测试和 web 接口测试。

8.3.1、模块接口测试

我们先看例子，后谈概念。

实现模块功能，module.py

```
'''
Author: 虫师
Date: 2016/2/19
Describe: 实现简单计算器: +、-、*、/、
'''
```

```
class Calculator():

    def __init__(self, a, b):
        self.a = int(a)
        self.b = int(b)

    # 加法
    def add(self):
        return self.a + self.b

    # 减法
    def sub(self):
        return self.a - self.b

    # 乘法
    def mul(self):
        return self.a * self.b

    # 除法
    def div(self):
        return self.a / self.b
```

对模块 Calculator 接口的测试, test.py

```
from module import Calculator
import unittest

class TestModule(unittest.TestCase):

    def setUp(self):
        pass

    def tearDown(self):
```

```

pass

def test_add(self):
    c = Calculator(2, 3)
    self.assertEqual(c.add(), 5)

def test_sub(self):
    c = Calculator(5, 3)
    self.assertEqual(c.sub(), 2)

def test_mul(self):
    c = Calculator(2, 3)
    self.assertEqual(c.mul(), 6)

def test_div(self):
    c = Calculator(6, 3)
    self.assertEqual(c.div(), 2)

if __name__ == '__main__':
    unittest.main()

```

本例当中用到了单元测试框架--unittest。

官方文档: <https://docs.python.org/3.5/library/unittest.html>

在我出版的《Selenium2 自动化测试实战—基于 Python 语言》一书中，有一章专门介绍了该框架的使用。

那么有读者会有疑问，那这个例子是不是可以看成是一个单元测试用例？个人认为，其实它们之间并没有太清晰的界限，来看定义：

单元测试 (unit testing)，是指对软件中的最小可测试单元进行检查和验证。

模块接口测试 (Module interface testing)，它是单元测试的基础。只有在数据能正确流入、流出模块的前提下，其他测试才有意义。

从它们定义上看，单元测试强调的是程序的最小可测单元。而模块接口测试强调的是模块的输入和输出，输入测试数组，验证输出的数据是否符合语言。从这个角度来看，单元测试框架就是用来进行模块接口测试的，而且单元测试的概念比较模糊。

这里了为了做到例子的简单，所以尽量简化模块功能，实际项目中的模块会远比 Calculator 要复杂得多。假如几个开发人员去完成一个系统，他们分别开发一些功能模块，最终数据会在这些功能模块之间传递。当 A 开发好自己负责的功能模块后会提供相应的接口（类方法、函数），B 肯定需要模拟数据调用 A 写的模块接口，用来验证改接口是否符合需求。

当然，在测试的手段上需要遵循测试的一些要点。

- 1、检查接口返回的数据是否与预期的结果一致。
- 2、检查接口的容错性，假如传递数据的类型错误时是否可以处理。例如上面的例子是支持整数，传递的是小数或字符串呢？
- 3、接口参数的边界值。例如，传递的参数足够大或为负数时，接口是否可以正常处理。
- 4、接口的性能，接口处理数据的时间也是测试的一个方面。牵扯到内部就是算法与代码的优化。
- 5、接口的安全性，如果是外部接口的话，这点尤为重要。

8.3.2、web 接口测试

web 接口测试又可分为两类：服务器接口测试和外部接口测试。

服务器接口测试：是测试浏览器与服务器的接口。这个很容易理解，我们知道 web 开发一般分前端和后端，前端开发人员用 html/css/javascript 等技术。后端开发人员用 php/java/python/ruby 等各种语言。用户输入的数据是输入到的前端页面上，怎样把这些数据传递的后台的呢？通过 http 协议的 get 与 post 请求来实现前后端的数据传递。这也可认为是接口测试，调用的登录接口还是查询接口，传参的是用户密码还是搜索关键字。

外部接口测试：这个很典型的例子就是第三方登录，比如你做的新系统免于新用户重新注册的麻烦会提供第三方登录，那用户在登录的时候调用的就是第三方登录的接口，由第三方验证用户名和密码并且返回给当前系统。

对于服务器接口测试，我们来看例子，以 Django 为例。

在.../blog/view.py 文件中添加：

```
# hello
def hello(request):
    name = request.GET.get('name', '')
    return HttpResponse("Hello , %s!" % name)
```

然后通过 URL 地址发起 get 请求：<http://127.0.0.1:8000/hello/?name=zhangsan>

name 可以等于任何内容。我们将看到如图 8.3，请求 name 打印在页面上。

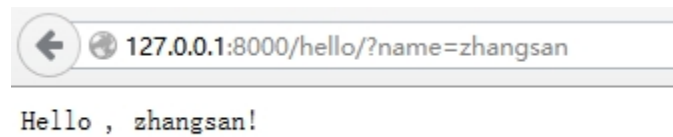


图 8.3 get 请求

对于 web 接口测试来说有哪些测试要点：

- 1、请求是否正确，默认请求成功是 200，如果请求错误也能返回 404、500 等。
- 2、检查返回数据的正确性与格式：json 是一种非常创建的格式。
- 3、接口的安全性，一般 web 都不会暴露在网上任意被调用，需要做一些限制，比如鉴权或认证。
- 4、接口的性能，web 接口同样注重性能，这直接影响用户的使用体验。如果我搜索一个关键字半天结果都没返回，果断弃用。

8.4 HTTP 协议基础

对于 web 接口测试来说，必不可少的需要了解 HTTP 协议。这里不打算花费的太多的精力对其时行介绍，一方面是它比较简单，也许我们之前并没有系统的对其时行学习，但我们只要上网就已经在使用它了；另一方面关于它的资料非常多，我们随便就可以找到一堆关于它的介绍。

超文本传输协议（HTTP，HyperText Transfer Protocol）是互联网上应用最为广泛的一种网络协议。

HTTP 协议的主要特点可概括如下：

1. 支持客户/服务器模式。
2. 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。
3. 灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。
4. 无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

5. 无状态：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

URL 结构：

http（超文本传输协议）是一个基于请求与响应模式的、无状态的、应用层的协议，常基于 TCP 的连接方式，HTTP1.1 版本中给出一种持续连接的机制，绝大多数的 Web 开发，都是构建在 HTTP 协议之上的 Web 应用。

HTTP URL (URL 是一种特殊类型的 URI，包含了用于查找某个资源的足够的信息)的格式如下：

`http://host[":"port][abs_path]`

http 表示要通过 HTTP 协议来定位网络资源；host 表示合法的 Internet 主机域名或者 IP 地址；port 指定一个端口号，为空则使用缺省端口 80；abs_path 指定请求资源的 URI；如果 URL 中没有给出 abs_path，那么当它作为请求 URI 时，必须以“/”的形式给出，通常这个工作浏览器自动帮我们完成。

1、输入：www.testpub.cn

浏览器自动转换成：`http://www.guet.edu.cn/`

2、`http:127.0.0.1:8000/hello`

HTTP 请求类型：

请求行以一个方法符号开头，以空格分开，后面跟着请求的 URI 和协议的版本，格式如下：Method Request-URI HTTP-Version CRLF

其中 Method 表示请求方法；Request-URI 是一个统一资源标识符；HTTP-Version 表示请求的 HTTP 协议版本；CRLF 表示回车和换行（除了作为结尾的 CRLF 外，不允许出现单独的 CR 或 LF 字符）。

请求方法（所有方法全为大写）有多种，各个方法的解释如下：

请求方法	说明
GET	请求获取 Request-URI 所标识的资源
POST	在 Request-URI 所标识的资源后附加新的数据
HEAD	请求获取由 Request-URI 所标识的资源的响应消息报头
PUT	请求服务器存储一个资源，并用 Request-URI 作为其标识

DELETE	请求服务器删除 Request-URI 所标识的资源
TRACE	请求服务器回送收到的请求信息，主要用于测试或诊断
CONNECT	保留将来使用
OPTIONS	请求查询服务器的性能，或者查询与资源相关的选项和需求

响应状态码：

状态代码有三位数字组成，第一个数字定义了响应的类别，且有五种可能取值：

- 1xx：指示信息--表示请求已接收，继续处理
- 2xx：成功--表示请求已被成功接收、理解、接受
- 3xx：重定向--要完成请求必须进行更进一步的操作
- 4xx：客户端错误--请求有语法错误或请求无法实现
- 5xx：服务器端错误--服务器未能实现合法的请求

常见状态代码、状态说明：

- 200 OK //客户端请求成功
- 400 Bad Request //客户端请求有语法错误，不能被服务器所理解
- 401 Unauthorized //请求未经授权，这个状态代码必须和 WWW-Authenticate 报头域一起使用
- 403 Forbidden //服务器收到请求，但是拒绝提供服务
- 404 Not Found //请求资源不存在，eg：输入了错误的 URL
- 500 Internal Server Error //服务器发生不可预期的错误
- 503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

请求头信息与响应头信息：

参数	头信息	响应	HTML	缓存
响应头信息		原始头信息		
Content-Type	text/html; charset=utf-8			
Date	Sat, 20 Feb 2016 14:34:41 GMT			
Server	WSGIServer/0.2 CPython/3.5.0			
X-Frame-Options	SAMEORIGIN			
请求头信息		原始头信息		
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8			
Accept-Encoding	gzip, deflate			
Accept-Language	zh-cn,zh;q=0.8,en-us;q=0.5,en;q=0.3			
Connection	keep-alive			
Host	127.0.0.1:8000			
User-Agent	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:29.0) Gecko/20100101 Firefox/29.0			

图 8.3 HTTP 头信息

1、请求头信息

请求报头允许客户端向服务器端传递请求的附加信息以及客户端自身的信息。

常用的请求报头：

Accept

Accept 请求报头域用于指定客户端接受哪些类型的信息。eg: Accept: image/gif, 表明客户端希望接受 GIF 图象格式的资源; Accept: text/html, 表明客户端希望接受 html 文本。

Accept-Encoding

Accept-Encoding 请求报头域类似于 Accept, 但是它是用于指定可接受的内容编码。

Accept-Language

Accept-Language 请求报头域类似于 Accept, 但是它是用于指定一种自然语言。

Connection

Connection: 允许发送指定连接的选项。例如指定连接是连续, 或者指定 “close” 选项, 通知服务器, 在响应完成后, 关闭连接。从 HTTP/1.1 起, 默认都开启了 Keep-Alive, 保持连接特性。

Host (发送请求时, 该报头域是必需的)

Host 请求报头域主要用于指定被请求资源的 Internet 主机和端口号, 它通常从 HTTP URL 中提取出来的。

User-Agent

我们上网登陆论坛的时候, 往往会看到一些欢迎信息, 其中列出了你的操作系统的名称和版本, 你所使用的浏览器的名称和版本, 这往往让很多人感到很神奇, 实际上, 服务器应用程序就是从 User-Agent 这个请求报头域中获取到这些信息。User-Agent 请求报头域允许客户端将它的操作系统、浏览器和其它属性告诉服务器。不过, 这个报头域不是必需的, 如果我们自己编写一个浏览器, 不使用 User-Agent 请求报头域, 那么服务器端就无法得知我们的信息了。

2、响应头信息

响应报头允许服务器传递不能放在状态行中的附加响应信息，以及关于服务器的信息和对 Request-URI 所标识的资源进行下一步访问的信息。

常用的响应报头

Location

Location 响应报头域用于重定向接受者到一个新的位置。Location 响应报头域常用在更换域名的时候。

Server

Server 响应报头域包含了服务器用来处理请求的软件信息。与 User-Agent 请求报头域是相对应的。

WWW-Authenticate

WWW-Authenticate 响应报头域必须被包含在 401（未授权的）响应消息中，客户端收到 401 响应消息时候，并发送 Authorization 报头域请求服务器对其进行验证时，服务端响应报头就包含该报头域。

X-Frame-Options

X-Frame-Options 有三个值：DENY 表示该页面不允许在 frame 中展示，即便是在相同域名的页面中嵌套也不允许。SAMEORIGIN 表示该页面可以在相同域名页面的 frame 中展示。ALLOW-FROM uri 表示该页面可以在指定来源的 frame 中展示。

第九章 Requests 库

Requests 是使用 Apache2 Licensed 许可证的 HTTP 库。用 Python 编写。

Requests 使用的是 urllib3，因此继承了它的所有特性。Requests 支持 HTTP 连接保持和连接池，支持使用 cookie 保持会话，支持文件上传，支持自动确定响应内容的编码，支持国际化的 URL 和 POST 数据自动编码。现代、国际化、人性化。

Requests 是以 PEP 20 的习语为中心开发的

- 1、Beautiful is better than ugly. (美丽优于丑陋)
- 2、Explicit is better than implicit. (清楚优于含糊)
- 3、Simple is better than complex. (简单优于复杂)
- 4、Complex is better than complicated. (复杂优于繁琐)
- 5、Readability counts. (重要的是可读性)

官方网站: <http://docs.python-requests.org/en/master/>

中文文档: http://cn.python-requests.org/zh_CN/latest/

9.1 安装与例子

9.1.1、安装

通过 pip 安装:

Python Shell

```
C:\Users\fnggj> python3 -m pip install requests
```

```
.....
```

Requests 一直在 Github 上被积极的开发着，你可以在此获得代码。你也可以克隆公共版本库：

9.1.2、第一个例子

Python Shell

```
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> import requests
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf-8'
>>> r.encoding
'utf-8'
>>> r.text
'{"login":"user","id":1000588,"avatar_url":.....'
>>> r.json()
{'public_gists': 0, 'id': 1000588, 'type': .....
```

此乃 Requests 官方所提供的第一个例子，不过，要想尝试执行这个例子，前提是你得有个 github 账号。

9.2 快速上手

9.2.1、发送请求

使用 Requests 发送网络请求非常简单。

一开始要导入 Requests 模块：

Python Shell

```
>>> import requests
```

然后，尝试获取某个网页。本例子中，我们来获取 **Github** 的公共时间线

Python Shell

```
>>>r = requests.get('https://github.com/timeline.json')
```

现在，我们有一个名为 **r** 的 **Response** 对象。可以从这个对象中获取所有我们想要的信息。

Requests 简便的 API 意味着所有 HTTP 请求类型都是显而易见的。例如，你可以这样发送一个 HTTP POST 请求：

Python Shell

```
>>> r = requests.post("http://httpbin.org/post")
```

漂亮，对吧？那么其他 HTTP 请求类型：**PUT**，**DELETE**，**HEAD** 以及 **OPTIONS** 又是如何的呢？都是一样的简单：

Python Shell

```
>>> r = requests.put("http://httpbin.org/put")
>>> r = requests.delete("http://httpbin.org/delete")
>>> r = requests.head("http://httpbin.org/get")
>>> r = requests.options("http://httpbin.org/get")
```

都很不错吧，但这也仅是 **Requests** 的冰山一角呢。

9.2.2、为 URL 传递参数

你也许经常想为 URL 的查询字符串(query string)传递某种数据。如果你是手工构建 URL，那么数据会以键/值 对的形式置于 URL 中，跟在一个问号的后面。例如，**httpbin.org/get?key=val**。**Requests** 允许你使用 **params** 关键字参数，以一个字典来提供这些参数。举例来说，如果你想传递 **key1=value1** 和 **key2=value2** 到 **httpbin.org/get**，那么你可以使用如下代码：

Python Shell

```
>>> payload = {'key1': 'value1', 'key2': 'value2'}
>>> r = requests.get("http://httpbin.org/get", params=payload)
```

通过打印输出该 URL，你能看到 URL 已被正确编码：

Python Shell

```
>>> print(r.url)http://httpbin.org/get?key2=value2&key1=value1
```

注意字典里值为 `None` 的键都不会被添加到 URL 的查询字符串里。

9.2.3、响应内容

我们能读取服务器响应的内容。再次以 Github 时间线为例：

Python Shell

```
>>> import requests
>>> r = requests.get('https://github.com/timeline.json')
>>> r.text
'[{"repository":{"open_issues":0,"url":"https://github.com/...
```

`Requests` 会自动解码来自服务器的内容。大多数 `unicode` 字符集都能被无缝地解码。

请求发出后，`Requests` 会基于 HTTP 头部对响应的编码作出有根据的推测。当你访问 `r.text` 之时，`Requests` 会使用其推测的文本编码。你可以找出 `Requests` 使用了什么编码，并且能够使用 `r.encoding` 属性来改变它：

Python Shell

```
>>> r.encoding
'utf-8'
>>> r.encoding = 'ISO-8859-1'
```

如果你改变了编码，每当你访问 `r.text`，`Request` 都将会使用 `r.encoding` 的新值。你可能希望在使用特殊逻辑计算出文本的编码的情况下修改编码。比如 `HTTP` 和 `XML` 自身可以指定编码。这样的话，你应该使用 `r.content` 来找到编码，然后设置 `r.encoding` 为相应的编码。这样就能使用正确的编码解析 `r.text` 了。

在你需要的情况下，`Requests` 也可以使用定制的编码。如果你创建了自己的编码，并使用 `codecs` 模块进行注册，你就可以轻松地使用这个解码器名称作为 `r.encoding` 的值，然后由 `Requests` 来为你处理编码。

9.2.4、二进制响应内容

你也能以字节的方式访问请求响应体，对于非文本请求：

Python Shell

```
>>> r.content
b' [{"repository": {"open_issues": 0, "url": "https://github.com/...
```

Requests 会自动为你解码 `gzip` 和 `deflate` 传输编码的响应数据。

例如，以请求返回的二进制数据创建一张图片，你可以使用如下代码：

Python Shell

```
>>> from PIL import Image
>>> from StringIO import StringIO
>>> i = Image.open(StringIO(r.content))
```

9.2.5、JSON 响应内容

Requests 中也有一个内置的 JSON 解码器，助你处理 JSON 数据：

Python Shell

```
>>> import requests
>>> r = requests.get('https://github.com/timeline.json')
>>> r.json()
[{'u'repository': {'u'open_issues': 0, u'url': 'https://github.com/...
```

如果 JSON 解码失败，`r.json` 就会抛出一个异常。例如，相应内容是 `401 (Unauthorized)`，尝试访问 `r.json` 将会抛出 `ValueError: No JSON object could be decoded` 异常。

9.2.6、原始响应内容

在罕见的情况下你可能想获取来自服务器的原始套接字响应，那么你可以访问 `r.raw`。如果你确实想这么干，那请你确保在初始请求中设置了 `stream=True`。具体的你可以这么做：

Python Shell

```
>>> r = requests.get('https://github.com/timeline.json', stream=True)
>>> r.raw
<requests.packages.urllib3.response.HTTPResponse object at 0x101194810>
>>> r.raw.read(10)
'\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\x03'
```

但一般情况下，你应该下面的模式将文本流保存到文件：

test_interface.py

```
with open(filename, 'wb') as fd:
    for chunk in r.iter_content(chunk_size):
        fd.write(chunk)
```

使用 `Response.iter_content` 将会处理大量你直接使用 `Response.raw` 不得不处理的。当流下载时，上面是优先推荐的获取内容方式。

9.2.7、定制请求头

如果你想为请求添加 HTTP 头部，只要简单地传递一个 `dict` 给 `headers` 参数就可以了。

例如，在前一个示例中我们没有指定 `content-type`：

Python Shell

```
>>> import json
>>> url = 'https://api.github.com/some/endpoint'
>>> payload = {'some': 'data'}
>>> headers = {'content-type': 'application/json'}
>>> r = requests.post(url, data=json.dumps(payload), headers=headers)
```

9.2.8、更复杂的 POST 请求

通常，你想要发送一些编码为表单形式的数据—非常像一个 HTML 表单。要实现这个，只需简单地传递一个字典给 `data` 参数。你的数据字典 在发出请求时会自动编码为表单形式：

Python Shell

```
>>> payload = {'key1': 'value1', 'key2': 'value2'}
>>> r = requests.post("http://httpbin.org/post", data=payload)
>>> print r.text
{ ... "form": { "key2": "value2", "key1": "value1" }, ...}
```

很多时候你想要发送的数据并非编码为表单形式的。如果你传递一个 `string` 而不是一个 `dict`，那么数据会被直接发布出去。

例如，Github API v3 接受编码为 JSON 的 POST/PATCH 数据：

Python Shell

```
>>> import json
>>> url = 'https://api.github.com/some/endpoint'
>>> payload = {'some': 'data'}
>>> r = requests.post(url, data=json.dumps(payload))
```

9.2.9、POST 一个多部分编码（Multipart-Encoded）的文件

Requests 使得上传多部分编码文件变得很简单：

Python Shell

```
>>> url = 'http://httpbin.org/post'
>>> files = {'file': open('report.xls', 'rb')}
>>> r = requests.post(url, files=files)
>>> r.text
{ ... "files": { "file": "<censored...binary...data>" }, ...}
```

你可以显式地设置文件名，文件类型和请求头：

Python Shell

```
>>> url = 'http://httpbin.org/post'
>>> files = {'file': ('report.xls', open('report.xls', 'rb'),
'application/vnd.ms-excel', {'Expires': '0'})}
>>> r = requests.post(url, files=files)
>>> r.text
{ ... "files": { "file": "<censored...binary...data>" }, ...}
```

如果你想，你也可以发送作为文件来接收的字符串：

Python Shell

```
>>> url = 'http://httpbin.org/post'
>>> files = {'file': ('report.csv', 'some,data,to,send\nanother,row,to,send\n')}
>>> r = requests.post(url, files=files)
>>> r.text
{ ... "files": { "file":
"some,data,to,send\nanother,row,to,send\n" }, ...}
```

如果你发送一个非常大的文件作为 **multipart/form-data** 请求，你可能希望流请求(?)。默认下 **requests** 不支持，但有个第三方包支持 - **requests-toolbelt**。你可以阅读 **toolbelt** 文档 来了解使用方法。

在一个请求中发送多文件参考 高级用法 一节。

9.2.10、响应状态码

我们可以检测响应状态码：

Python Shell

```
>>> r.encoding
>>> r = requests.get('http://httpbin.org/get')
>>> r.status_code
200
```

为方便引用，**Requests** 还附带了一个内置的状态码查询对象：

Python Shell

```
>>> r.status_code == requests.codes.ok
True
```

如果发送了一个失败请求(非 200 响应), 我们可以通过 `Response.raise_for_status()` 来抛出异常:

Python Shell

```
>>> bad_r = requests.get('http://httpbin.org/status/404')
>>> bad_r.status_code
404
>>> bad_r.raise_for_status()
Traceback (most recent call last):
  File "requests/models.py", line 832, in raise_for_status
raise http_errorrequests.exceptions.HTTPError: 404 Client Error
```

但是, 由于我们的例子中 `r` 的 `status_code` 是 200, 当我们调用 `raise_for_status()` 时, 得到的是:

Python Shell

```
>>> r.raise_for_status()
None
```

一切都挺和谐哈。

9.2.11、响应头

我们可以查看以一个 Python 字典形式展示的服务器响应头:

Python Shell

```
>>> r.headers
{'content-encoding': 'gzip', 'transfer-encoding': 'chunked',
'connection': 'close', 'server': 'nginx/1.0.4', 'x-runtime': '148ms',
```

```
'etag': '"e1ca502697e5c9317743dc078f67693f"', 'content-type':  
'application/json'}
```

但是这个字典比较特殊：它是仅为 HTTP 头部而生的。根据 RFC 2616，HTTP 头部是大小写不敏感的。

因此，我们可以使用任意大写形式来访问这些响应头字段：

Python Shell

```
>>>r.headers['Content-Type']  
'application/json'  
>>>r.headers.get('content-type')  
'application/json'
```

9.2.12、Cookies

如果某个响应中包含一些 Cookie，你可以快速访问它们：

Python Shell

```
>>>url='http://example.com/some/cookie/setting/url'  
>>>r=requests.get(url)  
>>>r.cookies['example_cookie_name']  
'example_cookie_value'
```

要想发送你的 cookies 到服务器，可以使用 cookies 参数：

Python Shell

```
>>>url='http://httpbin.org/cookies'  
>>>cookies=dict(cookies_are='working')  
>>>r=requests.get(url,cookies=cookies)  
>>>r.text  
'{"cookies": {"cookies_are": "working"}}'
```

9.2.13、重定向与请求历史

默认情况下，除了 HEAD, Requests 会自动处理所有重定向。

可以使用响应对象的 `history` 方法来追踪重定向。

`Response.history` 是一个 `class:Response <requests.Response>` 对象的列表，为了完成请求而创建了这些对象。这个对象列表按照从最老到最近的请求进行排序。

例如，Github 将所有的 HTTP 请求重定向到 HTTPS。：

Python Shell

```
>>>r=requests.get('http://github.com')
>>>r.url
'https://github.com/'
>>>r.status_code
200
>>>r.history
[<Response [301]>]
```

如果你使用的是 GET, OPTIONS, POST, PUT, PATCH 或者 DELETE, 那么你可以通过 `allow_redirects` 参数禁用重定向处理：

Python Shell

```
>>> r.encoding
>>>r=requests.get('http://github.com',allow_redirects=False)
>>>r.status_code
301
>>>r.history
[]
```

如果你使用的是 HEAD, 你也可以启用重定向：

Python Shell

```
>>>r=requests.head('http://github.com',allow_redirects=True)
>>>r.url
'https://github.com/'
>>>r.history
```

[<Response [301]>]

9.2.14、超时

你可以告诉 `requests` 在经过以 `timeout` 参数设定的秒数时间之后停止等待响应:

Python Shell

```
>>>requests.get('http://github.com',timeout=0.001)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>requests.exceptions.Timeout:
HTTPConnectionPool(host='github.com', port=80): Request timed out. (timeout=0.001)
```

注:

`timeout` 仅对连接过程有效, 与响应体的下载无关。 `timeout` is not a time limit on the entire response download; rather, an exception is raised if the server has not issued a response for `timeout` seconds (more precisely, if no bytes have been received on the underlying socket for `timeout` seconds).

9.2.15、错误与异常

遇到网络问题 (如: DNS 查询失败、拒绝连接等) 时, `Requests` 会抛出一个 `ConnectionError` 异常。

遇到罕见的无效 HTTP 响应时, `Requests` 则会抛出一个 `HTTPError` 异常。

若请求超时, 则抛出一个 `Timeout` 异常。

若请求超过了设定的最大重定向次数, 则会抛出一个 `TooManyRedirects` 异常。

所有 `Requests` 显式抛出的异常都继承自 `requests.exceptions.RequestException`。

第十章 项目开发 with 接口测试实战

利用 Django 开发 Web 项目我们已经掌握了不少基础知识，对于 Web 接口测试来说，基本的使用，我也已经掌握。本章将通过一个具体的实例，来深化前面的开发与测试技能。

10.1 开发投票系统

参考官网文档，创建投票系统。

<https://docs.djangoproject.com/en/1.9/intro/tutorial01/>

=====

Windows 7/10

Python 3.5.10

Django 1.9.1

=====

10.1.1、创建应用 Polls

继续在我们原先的项目（myweb）下创建应用 polls。

cmd.exe

```
D:\pydj\myweb>python3 manage.py startapp polls
```

打开.../myweb/setting.py 文件添加：

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
```

```
'django.contrib.auth',  
'django.contrib.contenttypes',  
'django.contrib.sessions',  
'django.contrib.messages',  
'django.contrib.staticfiles',  
'blog',  
'bootstrap3',  
'polls',
```

最终目录结构：

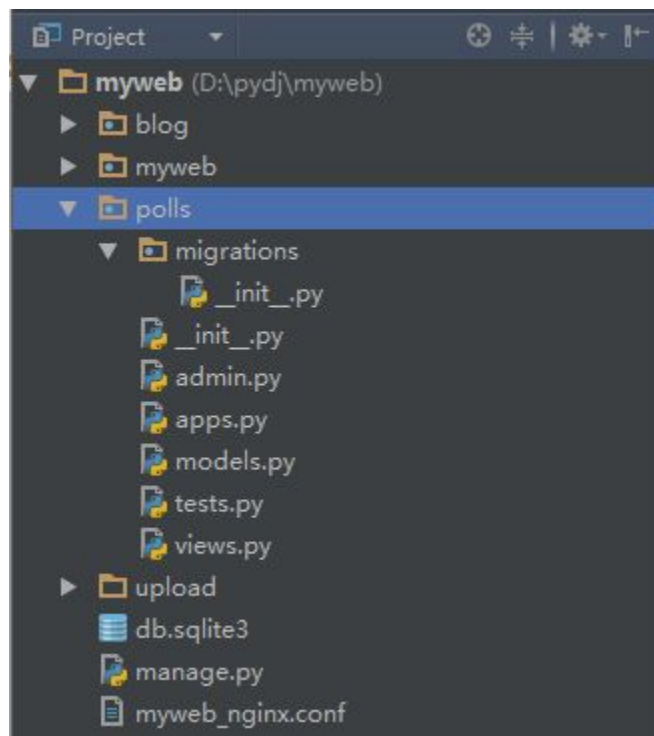


图 10.1 polls 应用目录

10.1.2、创建模型

一般 web 开发先设计数据库，数据库设计好了，项目就完了大半了，可见数据库的重要性。打开 polls/models.py 编写如下：

```
from django.db import models  
  
# Create your models here.
```

```
# 问题
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

    def __unicode__(self):
        return self.question_text

# 选择
class Choice(models.Model):
    question = models.ForeignKey(Question)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)

    def __unicode__(self):
        return self.choice_text
```

执行数据库表生成与同步。

cmd.exe

```
D:\pydj\myweb>python3 manage.py makemigrations polls
```

```
Migrations for 'polls':
```

```
0001_initial.py:
```

- Create model Choice
- Create model Question
- Add field question to choice

```
D:\pydj\myweb>python3 manage.py migrate
```

```
Operations to perform:
```

```
Apply all migrations: contenttypes, sessions, admin, polls, blog, auth
```

```
Running migrations:
```

```
Applying polls.0001_initial... OK
```

10.1.3、admin 管理

Django 提供了强大的后台管理，对于 web 应用来说，后台必不可少，例如，当前投票系统，如何添加问

题与问题选项？直接操作数据库添加，显然麻烦，不方便，也不安全。所以，管理后台就可以完成这样的工作。

打开 `polls/admin.py` 文件，编写如下内容：

```
from django.contrib import admin
from .models import Question, Choice

# Register your models here.
class ChoiceInline(admin.TabularInline):
    model = Choice
    extra = 3

class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        (None, {'fields': ['question_text']}),
        ('Date information', {'fields': ['pub_date'], 'classes': ['collapse']}),
    ]
    inlines = [ChoiceInline]
    list_display = ('question_text', 'pub_date')

admin.site.register(Choice)
admin.site.register(Question, QuestionAdmin)
```

当前脚本的作用就是将模型（数据库表）交由 `admin` 后台管理。运行 `web` 容器：

cmd.exe

```
D:\pydj\myweb>python3 manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
February 23, 2016 - 22:59:01
Django version 1.9.1, using settings 'myweb.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

登录后台: <http://127.0.0.1:8000/admin>

登录密码就是在执行数据库同步时设置的用户名和密码。



图 10.2 后台 polls 表管理

点击“add”添加问题。

Question text:

Date information (Show)

CHOICES		
CHOICE TEXT	VOTES	DELETE?
<input type="text" value="旅行"/>	<input type="text" value="0"/>	
<input type="text" value="看电影"/>	<input type="text" value="0"/>	
<input type="text" value="看书"/>	<input type="text" value="0"/>	

[+ Add another Choice](#)

图 10.3 添加投票问题

10.1.4、编写视图

视图起着承前启后的作用，前是指前端页面，后是指后台数据库。将数据库表中的内容查询出来显示到页面上。

编写 polls/views.py 文件:

```
from django.shortcuts import render, get_object_or_404
from django.http import HttpResponseRedirect
from django.core.urlresolvers import reverse
from .models import Question, Choice
```

```

# Create your views here.
# 首页展示所有问题
def index(request):
    # latest_question_list2 = Question.objects.order_by('-pub_data')[:2]
    latest_question_list = Question.objects.all()
    context = {'latest_question_list': latest_question_list}
    return render(request, 'polls/index.html', context)

# 查看单个问题选项
def detail(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/detail.html', {'question': question})

# 查看投票结果
def results(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/results.html', {'question': question})

# 选择投票
def vote(request, question_id):
    p = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice = p.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):
        # Redisplay the question voting form.
        return render(request, 'polls/detail.html', {
            'question': p,
            'error_message': "You didn't select a choice.",
        })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        # Always return an HttpResponseRedirect after successfully dealing
        # with POST data. This prevents data from being posted twice if a
        # user hits the Back button.
        return HttpResponseRedirect(reverse('polls:results', args=(p.id,)))

```

10.1.5、配置 URL

url 是一个请求配置文件，页面中的请求转交给由哪个函数处理，由该文件决定。

首先配置.../polls/urls.py（该文件需要创建）

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

urlpatterns = [
    # ex : /polls/
    url(r'^$', views.index, name='index'),
    # ex : /polls/5/
    url(r'^(?P<question_id>[0-9]+)/$', views.detail, name='detail'),
    # ex : /polls/5/results/
    url(r'^(?P<question_id>[0-9]+)/results/$', views.results, name='results'),
    # ex : /polls/5/vote
    url(r'^(?P<question_id>[0-9]+)/vote/$', views.vote, name='vote'),
]
```

接着，编辑.../myweb/urls.py 文件。

```
from django.conf.urls import url, include
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^polls/', include('polls.urls', namespace="polls")),
]
```

10.1.6、创建模板

模板就是前端页面，用来将数据显示到 web 页面上。

首先创建 polls/templates/polls/ 目录，分别在该目录下创建 index.html、detail.html 和 results.html 文件。

index.html

```

<html lang="zh-CN">
<head>
    {% load bootstrap3 %}
    {% bootstrap_css %}
    {% bootstrap_javascript %}
    {% bootstrap_messages %}
</head>
<body>
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <a class="navbar-brand" href="#">Polls System</a>
            </div>
        </div>
    </nav>

    <br><br>
    <div class="well">
        <h3>Question List:</h3>
        {% if latest_question_list %}
            <ul>
                {% for question in latest_question_list %}
                    <li><a href="{% url 'polls:detail'
question.id %}">{{ question.question_text }}</a></li>
                {% endfor %}
            </ul>
        {% else %}
            <p>No polls are available.</p>
        {% endif %}
    </div>

    <footer>
        <p>&copy; Company 2016 & chongshi</p>
    </footer>

</body>
</html>

```

detail.html

```

<html lang="zh-CN">
<head>

```



```

{% load bootstrap3 %}
{% bootstrap_css %}
{% bootstrap_javascript %}
{% bootstrap_messages %}
</head>
<body>
  <nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
      <div class="navbar-header">
        <a class="navbar-brand" href="#">Polls System</a>
      </div>
    </div>
  </nav>

  <br><br>
  <div class="well">
    <h1>{{ question.question_text }}</h1>
    {% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}
    <form action="{% url 'polls:vote' question.id %}" method="post">
      {% csrf_token %}
      {% for choice in question.choice_set.all %}
        <input type="radio" name="choice" id="choice{{ forloop.counter }}"
value="{{ choice.id }}" />
        <label
for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br />
      {% endfor %}
      <input type="submit" value="Vote" />
    </form>
  </div>

  <footer>
    <p>&copy; Company 2016 & chongshi</p>
  </footer>

</body>
</html>

```

results.html

```

<html lang="zh-CN">
  <head>
    {% load bootstrap3 %}

```

```

{% bootstrap_css %}
{% bootstrap_javascript %}
{% bootstrap_messages %}
</head>
<body>
  <nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
      <div class="navbar-header">
        <a class="navbar-brand" href="#">Polls System</a>
      </div>
    </div>
  </nav>

  <br><br>
  <div class="well">
    <h1>{{ question.question_text }}</h1>
    <ul>
      {% for choice in question.choice_set.all %}
        <li>{{ choice.choice_text }} -- {{ choice.votes }}
vote{{ choice.votes|pluralize }}</li>
      {% endfor %}
    </ul>
    <a href="{% url 'polls:detail' question.id %}">Vote again?</a>
  </div>

  <footer>
    <p>&copy; Company 2016 & chongshi</p>
  </footer>

</body>
</html>

```

10.1.7、系统功能展示

首先需要在 admin 后台添加投票问题，并设置投票选项启动 web 容器。

接下来访问：<http://127.0.0.1:8000/polls/>



图 10.4 投票活动首页

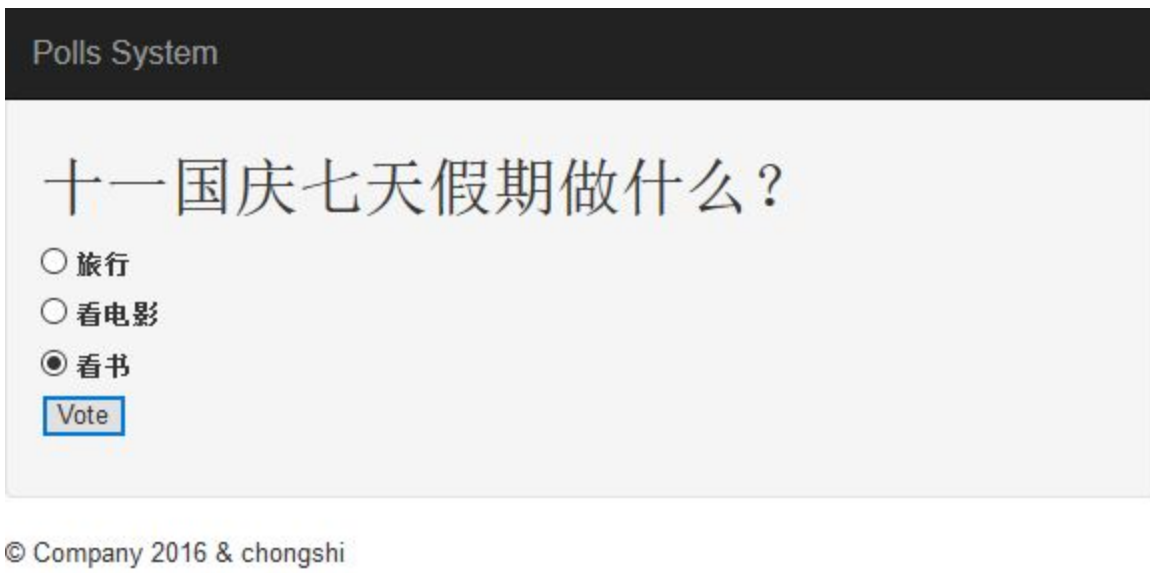


图 10.5 选择投票选项

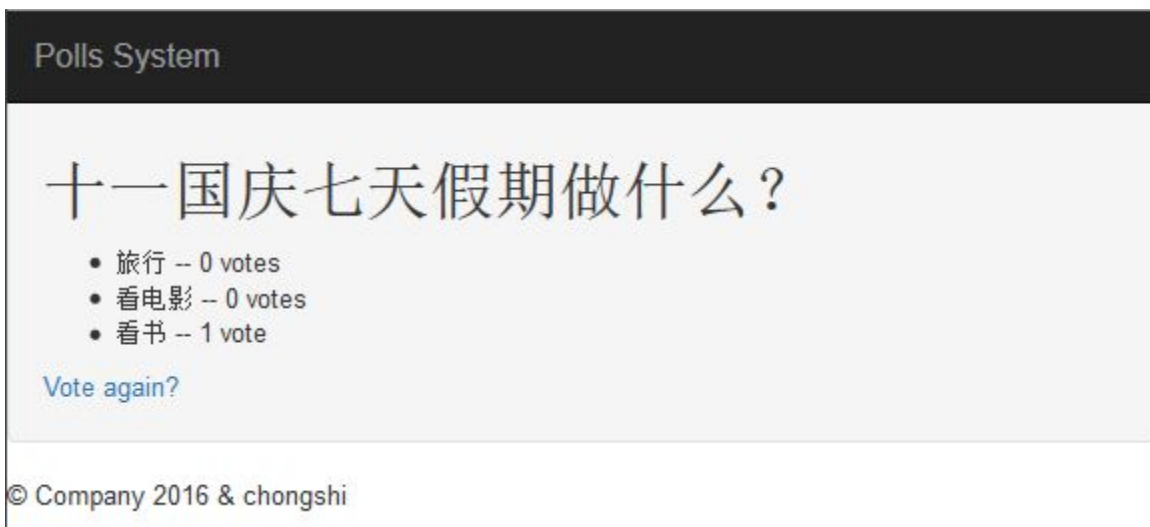


图 10.6 查看投票结构

10.2 开发投票系统接口

虽然投票系统的功能已经开发完成，但我们并没有开发专门的接口，在当前的投票系统中，在我们调用一个 `get` 或 `post` 请求时，系统会返回整个页面，并且把测试连同页面一起返回。

10.2.1、改造投票系统接口

例如，当我们要调用所有问题的接口时（`test_get.py`）：

```
import requests

base_url = 'http://127.0.0.1:8000/polls'
r = requests.get(base_url)
code = r.status_code
text = r.text
print(code)
print(text)
```

运行程序，得到如下结果：

```
200

<html lang="zh-CN">
  <head>
    <link href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css"
rel="stylesheet">
    <script
src="//maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>

  </head>
  <body>
    <nav class="navbar navbar-inverse navbar-fixed-top">
      <div class="container">
        <div class="navbar-header">
```

```

        <a class="navbar-brand" href="#">Polls System</a>
    </div>
</div>
</nav>

<br><br>
<div class="well">
    <h3>Question List:</h3>
    <ul>
        <li><a href="/polls/1/">十一国庆七天假期做什么? </a></li>
        <li><a href="/polls/2/">你最想学的自动化工具是什么? </a></li>
    </ul>
</div>

<footer>
    <p>&copy; Company 2016 & chongshi</p>
</footer>

</body>
</html>

```

而特有的接口应该返回的是数据，而不是整个页；而数据一般格式为数组、字典或 Json 格式。

所以，需要对试图层（.../polls/views.py）进行改造，使其只提供接口，并单纯的返回数据。

```

from django.shortcuts import render, get_object_or_404
from django.http import HttpResponseRedirect
from django.core.urlresolvers import reverse
from .models import Question, Choice
from django.http import HttpResponse
import json

# Create your views here.
# 查看所有问题
def index(request):
    latest_question_list = Question.objects.all()
    dicts = {}
    if latest_question_list:
        for question in latest_question_list:
            dicts[question.id] = question.question_text
    j = json.dumps(dicts)

```

```

        return HttpResponse(j)
    else:
        return HttpResponse("question list null")

# 查看单个问题选项
def detail(request, question_id):
    choices = Choice.objects.filter(question_id=question_id)
    dicts = {}
    print(question_id)
    if question_id:
        for choice in choices:
            dicts[choice.id] = choice.choice_text
        j = json.dumps(dicts)
        return HttpResponse(j)

```

为了节省时间，暂时先对查看所有问题、单个问题的所有选项两个功能进行接口改造，当然这里的改造也不够完整和健壮。例如单个问题的所有选项的接口，接收的参数 `question_id` 如果为空，应该提示，参数错误。如果查询不到相关问题，应该提示，查询结果为空，如果传的类型不为数字，应该提示，类型错误。这些都是一个健壮的接口应有的处理逻辑。

再次执行 `test_get.py` 文件。

```

200
{"1": "\u5341\u4e00\u56fd\u5e86\u4e03\u5929\u5047\u671f\u505a\u4ec0\u4e48\u5f1f",
"2":
"\u4f60\u6700\u60f3\u5b66\u7684\u81ea\u52a8\u5316\u5de5\u5177\u662f\u4ec0\u4e48\u5f1f"}

```

这一次得到的就是 `json` 类型的数据了。不过，返回值对中文进行了 `unicode` 的编码。这里提供个小技巧，将其转换成中文。

打开 `Firefox` 浏览器的 `Firebug` 工具，切换到“控制台”标签。

返回格式	json
返回结果	{ "1": "旅行", "2": "看电影", "3": "看书" }
错误类型	暂无（接口代码需要补充逻辑）

好啦！接口文档的大体结构就是上面的样子。有了这个份文档，我们接下来就很容易知道如何调用这些接口做测试了。

10.3 系统接口测试

对于编写接口测试来说，我们会涉及到两个技术。前面也都有过简单介绍，`unittest` 单元测试框架和 `request` 库。

```
import unittest
import requests

class PollsTest(unittest.TestCase):

    def setUp(self):
        self.base_url = 'http://127.0.0.1:8000/polls'

    def tearDown(self):
        pass

    def test_get_poll_index(self):
        '''测试投票系统首页'''
        r = requests.get(self.base_url)
        code = r.status_code
        text = r.text
        self.assertEqual(code, 200)
```



```
def test_get_poll_question(self):  
    '''获得问题 1 的所有选项'''  
    r = requests.get(self.base_url+' /1/')  
    code = r.status_code  
    text = r.text  
    self.assertEqual(code, 200)  
    self.assertIn("3", text)  
  
if __name__ == '__main__':  
    unittest.main()
```

接口用例测试本身的编写是简单的，我们只用调用接口，传递不同的参数。从而验证返回值是否符合预期即可。

=====

由于时间比较仓促，后面几章准备不足，还有一些内容值得我们探讨。我会在接一下来的时间继续完善，暂时先到这里。