第 5 章

开发“公司简介”模块

本章将开发的“公司简介”模块承接第 4 章的内容，对应 hengDaProject 项目中的 aboutApp 应用。该模块一共包含两个子页面：“企业概况”和“荣誉资质”，主要用于介绍企业的基本情况和展示企业获得的荣誉。图 5-1 和图 5-2 分别显示了两个子页面最终的效果图。“企业概况”页面与第 4 章制作的“科研基地”页面基本相同，页面全部由静态资源构成，包括说明文字和图片等，所有数据不需要从后台数据库读取。开发“企业概况”页面主要学习如何通过 Bootstrap 制作侧边导航栏。相对于“企业概况”页面，“荣誉资质”页面则采用动态数据嵌入的方式生成页面，其页面文字和图像均需要从后台读取，其好处在于可以让网站管理员方便地对企业荣誉进行添加、修改和删除。在“荣誉资质”页面开发的过程中，会详细阐述 Django 数据模型概念和基本使用方法。另外，Django 提供了现成的后台管理系统，本章会阐述如何使用该后台管理系统以及如何对后台管理系统进行优化，方便管理员管理网站数据。

视频讲解

5.1 继承模板

本节先来制作“公司简介”模块的基础页面。根据如图 5-1 所示页面效果，其与第 4 章制作的“科研基地”页面基本相同，仅在页面主体部分左侧多出一个侧边导航栏。下面进入具体的制作步骤。

按照第 4 章渲染页面的方法，首先打开 aboutApp 应用，在该应用下创建一个 templates 文件夹，然后在该文件夹下创建一个名为 survey.html 的网页文件。根据第 4 章创建的页面模板，以继承方式继承页面内容，包括：页面头部、导航栏、页脚。具体代码如下。

```
{% extends "base.html" %}
{% load staticfiles %}

{% block title %}
```

图 5-1　“企业概况”页面效果

图 5-2 “荣誉资质”页面效果

```
    企业概况
{ % endblock % }

{ % block content % }
<! -- 广告横幅 -- >
```

```
<div class = "container - fluid">
    <div class = "row">
        < img class = "img - responsive model - img" src = "{ % static 'img/about.jpg' % }">
    </div >
</div >
<! -- 主体内容 -- >
{ % endblock % }
```

上述代码通过{% extends "base.html" %}来继承模板 base.html 文件。对于广告横幅,修改相应的静态图片文件路径即可。主体部分暂时不填入内容,详细的主体内容设计将在下一小节进行阐述。

为了能够有效渲染 survey.html 文件,打开 aboutApp 应用下的 views.py 文件,修改视图处理函数 survey 如下。

```
def survey(request):
    return render(request, 'survey.html',{'active_menu': 'about',})
```

此处 render()函数第一个参数直接返回请求,第二个参数传入欲渲染的 HTML 文件名,第三个参数是为了在页面切换到“公司简介”时能够同步地切换导航栏激活状态,因此需要向模板添加 active_menu 参数。按 Ctrl+S 组合键保存所有修改的文件然后运行项目,单击“公司简介”下的“企业概况”链接,页面效果图如图 5-3 所示。

图 5-3　“企业概况”初始页面效果

按照上述开发方式，在 aboutApp 应用的 templates 文件夹下开发荣誉资质模块对应的页面 honor.html，并修改其视图处理函数 honor：

```
def honor(request):
    return render(request, 'honor.html',{'active_menu': 'about',})
```

至此，“公司简介”模块下的两个页面框架均已制作完成。可以看到，通过模板的复用，仅需使用几行继承代码，就可以将之前制作的子页面完整地导入进来，可以大幅提高项目开发效率。

最后还遗留一个小问题，如果当前已经在“企业概况”页面内，此时光标移到“荣誉资质”导航链接上会发现两个链接颜色是一样的。为了能够有效进行区分，在 style.css 文件中添加如下代码。

```
/* 二级菜单鼠标移过时属性 */
.navbar-default .navbar-nav li ul a:hover{
    color:#fff;
    background-color:#005197;
}
/* 一级菜单激活时，二级菜单鼠标移过时属性 */
.navbar-default .navbar-nav li.active ul a:hover{
    color:#fff;
    background-color:#022a4d;
}
```

通过上述设置，两个子模块链接在鼠标移过时就会呈现不同的颜色。5.2 节将开始制作页面主体部分。

5.2 制作侧边导航栏

5.1 节完成了“公司简介”模块下两个页面的基本框架设计，并修改了对应的视图处理函数，本节将制作主体部分中的侧边导航栏以方便用户切换子页面。

根据如图 5-1 所示效果，页面主体可以分为左右两部分，左边是侧边导航栏，右边是固定位置的图片和介绍性文字。根据页面结构，设计 HTML 基本结构如下。

```
<div class="container">
    <div class="row row-3">
        <!-- 侧边导航栏 -->
        <div class="col-md-3">
        </div>
        <!-- 说明文字和图片 -->
        <div class="col-md-9">
        </div>
    </div>
</div>
```

上述代码对页面采用 3-9 栅格布局，侧边导航栏占 3 个栅格，右边内容占 9 个栅格。采用 container 将整个页面主体内容限制在指定宽度内，并包含在 class＝row 的 div 中，占满一行。其中，为行 div 添加额外的样式“.row-3”，编辑 style.css 文件添加样式设置：

```
.row-3{
    margin-top:30px;  /* 设置顶部边距 */
}
```

侧边导航栏部分采用 Bootstrap 提供的列表组件 list-group 实现，其中每一个链接列表项用样式 list-group-item 表示，具体代码如下。

```
<!-- 侧边导航栏 -->
<div class="col-md-3">
    <div class="model-title">
        公司简介
    </div>
    <div class="model-list">
        <ul class="list-group">
            <li class="list-group-item" id="survey">
                <a href="{% url 'aboutApp:survey' %}">企业概况</a>
            </li>
            <li class="list-group-item" id="honor">
                <a href="{% url 'aboutApp:honor' %}">荣誉资质</a>
            </li>
        </ul>
    </div>
</div>
```

上述代码中"{% url 'aboutApp:survey' %}"和"{% url 'aboutApp:honor' %}"用于逆向寻找路由，方便后期部署。model-title 和 model-list 样式类分别用于定制导航栏标题和列表样式，编辑 style.css 文件，添加对应的样式。

```
/*侧边导航栏标题样式*/
.model-title {
    text-align: center;
    color: #fff;
    font-size: 22px;
    padding: 15px 0px;
    background: #005197;
    margin-top: 25px;
}
/*侧边导航栏列表项样式*/
.model-list li{
    text-align:center;
    background-color:#f6f6f6;
    font-size:16px;
}
```

```
/* 侧边导航栏列表项链接样式 */
.model-list li a{
    color:#545353;
}
/* 侧边导航栏列表项链接激活样式 */
.model-list li a:hover{
    text-decoration:none;
    color:#005197;
}
```

通过上述样式设置，可以完成侧边导航栏的基本设计。接下来针对侧边导航栏的页面切换链接进行样式设计。大致思路与设计一级导航栏链接一致，在侧边导航栏链接切换时处于激活状态的链接文字呈现蓝色，其他链接文字呈现灰色，这样用户切换侧边导航栏之后就可以明显地看到当前处于哪个子页面。实现方法可以参照一级导航栏的设计流程，只需要由后台向模板传递二级菜单变量，然后前端通过 JavaScript 脚本的 addClass()函数动态地对<li>标签添加 active 类即可实现。首先，编辑 style.css 文件，添加代码如下。

```
/* 侧边导航栏激活样式 */
.model-list li.active{
    text-align:center;
    background-color:#f6f6f6;
    font-size:16px;
    border-color: #ddd;
}
/* 侧边导航栏激活状态下鼠标移过时样式 */
.model-list li.active:hover{
    text-align:center;
    background-color:#f6f6f6;
    font-size:16px;
    border-color: #ddd;
}
/* 侧边导航栏激活状态时链接样式 */
.model-list li.active a{
    color:#005197;
}
```

上述 CSS 代码可以使得当侧边导航栏中的某一链接处于激活状态时其链接文字显示蓝色，方便用户浏览和辨识。接下来在 base.html 文件的<body>标签最后添加 JavaScript 代码：

```
<script type="text/JavaScript">
    $('#{{sub_menu}}').addClass("active");
</script>
```

最后修改 aboutApp 应用中的 views.py 文件，重新编辑 survey()和 honor()函数，在最后 render()函数返回时添加额外的 submenu 变量。

```
def survey(request):
    return render(request, 'survey.html', {
        'active_menu': 'about',
        'sub_menu': 'survey',
    })

def honor(request):
    return render(request, 'honor.html', {
        'active_menu': 'about',
        'sub_menu': 'honor',
    })
```

通过上述修改，即可实现侧边导航栏二级菜单切换。至此，已完成侧边导航栏的开发任务。接下来继续完善“企业概况”页面。主体右边部分是位置固定的介绍性文字和图片，其设计与“科研基地”页面基本一致，包括标题、下画线、段落文字和图片，这里不再过多阐述，详细设计代码可以参照 4.1 节。可以直接将 4.1 节中的代码复制到此处，然后替换文字和图片路径即可。“企业概况”最终效果如图 5-1 所示。

从最终效果来看，“企业概况”页面主体部分的文字和图片均为静态资源文件，即不需要与后台数据库进行交互，直接在 HTML 文件中调用，这种页面称为静态页面，其开发相对较为简单。在 5.3 节，将学习如何通过 Django 数据库模型来构建动态页面。

5.3　Django 数据库模型

本节将会通过 Django 数据库模型来渲染“荣誉资质”页面，其基本实现流程如下。

(1) 用户通过浏览器请求页面。

(2) 服务器收到浏览器请求，根据 URL 路由找到匹配的视图处理函数。

(3) 视图处理函数首先找到需要返回的 HTML 模板文件，然后从数据库中取出数据(图片数据对应的是图片的存储路径)，然后将数据过滤后以模板变量形式插入到模板文件中，最后通过 render()函数返回生成的页面。

(4) 浏览器收到请求页面并显示。

在 2.8 节中已经阐述了如何通过 Python 操作数据库，一般步骤为创建数据库、设计表结构和字段、使用 SQLite(MySQL、PostgreSQL 等)引擎来连接数据库、编写数据访问层代码、业务逻辑层调用数据访问层执行数据库操作。这里注意到上述数据库操作流程较为烦琐，需要开发人员直接面向数据库进行数据的增删查改，开发效率较低。是否可以让开发人员直接面向代码中的对象来操作数据库呢？答案是可以的。这种面向对象的数据库编程方式即为对象关系映射(Object Relational Mapping，ORM)。具体地，在 ORM 中类名对应数据库中的表名，类属性对应数据库里的字段，类实例对应数据库表里的某一行数据。Django 中内嵌了 ORM 框架，不需要直接面向数据库编程，而是通过模型类和对象完成数据表的增删改查操作。

使用 Django 进行数据库开发的步骤如下。

（1）配置数据库连接信息。

（2）在模型文件 models.py 中定义模型类。

（3）数据库模型迁移。

（4）通过类和对象完成数据增删改查操作。

具体地，Django 已经在项目创建时自动地提供了一个 SQLite 数据库用于为项目提供数据库操作，同时已为该数据库的使用配置好了参数。打开配置文件夹 hengDaProject 下的 settings.py 文件，找到其中的 DATABASES 字段，默认配置如下。

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

可以看到，该项目默认数据库引擎 ENGINE 为 django.db.backends.sqlite3，数据库路径为当前项目根目录下的 db.sqlite3 数据库文件（SQLite 数据库本质上是一个文件）。如果需要采用其他数据库，那么数据库的配置就在该字段进行设置。在实际 Web 站点部署时，一般不会采用 SQLite 数据库，因为该数据库的并发量、响应速度等具有较大的限制，但是在开发阶段可以采用该数据库进行开发测试。在第 11 章项目部署环节将会详细阐述如何在 Django 中配置和使用 MySQL 数据库。

在了解 Django 数据库的基本概念和配置后，5.3.1 节将会通过 Django 数据库模型来进行具体的开发。

5.3.1 创建荣誉模型

参照如图 5-2 所示效果，可以看到企业获得的每一项荣誉均采用“1 张图片＋1 段简要文字描述”这种形式进行展示。为了能够在后期方便管理人员对荣誉信息进行管理，需要抽象出当前的荣誉数据，并在数据库中生成相应的数据模型。

Django 数据模型是与数据库相关的，与数据库相关的代码通常写在 models.py 文件中。首先打开 aboutApp 中的 models.py 文件，在该文件中添加“荣誉”（Award）模型。

```
from django.db import models

class Award(models.Model):   #荣誉模型
    description = models.TextField(max_length=500, blank=True,
                                   null=True)   #文字描述
    photo = models.ImageField(upload_to='Award/', blank=True) #图片
```

首先导入 django.db 中的 models 模块来方便创建数据库字段。接下来定义了一个 Award 类，对应“荣誉”模型，该类继承自 models.Model。在 Award 类中定义了两个字段：description 和 photo，分别对应“荣誉”模型的文字描述和图片。文字描述采用 models.TextField 来进行字段声明，并且使用 max_length 参数来设置该字段允许的最大长度。另

外，通过设置参数 blank＝True，null＝True 表示该字段允许为空。图片信息采用 models.ImageField 来声明，通过设置 upload_to 参数来定义图片的上传目录，上传的图像信息会存储在服务器媒体资源路径的 Award 文件夹下面。

在本节实例中仅使用了 Django 数据库模型中的文本和图像字段，除此以外，Django 还提供了很多其他有用的字段，包括整型数据、字符型数据、浮点型数据等，具体调用形式和参数如表 5-1 所示。

表 5-1　Django 数据库模型常用字段

CharField	字符串字段，用于较短的字符串。CharField 要求必须有一个参数 maxlength，用于从数据库层和 Django 校验层限制该字段所允许的最大字符数
IntegerField	用于保存一个整数
DecimalField	用于保存一个浮点数。使用时必须提供以下两个参数。 max_digits：总位数(不包括小数点和符号)。 decimal_places：小数位数
AutoField	用于保存一个整型数据，添加记录时它会自动增长。通常不需要直接使用这个字段
BooleanField	用于保存真、假逻辑数据
TextField	用于保存一个容量很大的文本字段
EmailField	一个带有检查邮件合法性的字符字段，不接受 maxlength 参数
DateField	用于保存一个日期字段。可选参数主要有以下两个。 auto_now：当对象被保存时，自动将该字段的值设置为当前时间。 auto_now_add：当对象首次被创建时，自动将该字段的值设置为当前时间
DateTimeField	用于保存一个日期时间字段，功能类似 DateField，支持同样的附加选项
FileField	用于上传文件，在声明时必须指定参数 upload_to，该参数表明用于保存文件的本地路径
ImageField	功能类似 FileField，不过要校验上传对象是否是一个合法图片。除了 upload_to 以外它还有两个可选参数：height_field 和 width_field，如果提供这两个参数，则图片将按提供的高度和宽度规格进行保存
URLField	用于保存 URL。若 verify_exists 参数为 True (默认)，给定的 URL 会预先检查是否存在，即检查 URL 是否被有效装入且没有返回 404 响应
NullBooleanField	类似 BooleanField，不过允许 NULL 作为其中一个选项

注意，在使用表 5-1 中的 Django 模型字段时，其中关于文件上传的两个字段 ImageField 和 FileField 需要额外进行路径设置，即需要在项目中指定媒体资源文件存储目录。由于本节实例也采用了图像 ImageField 字段，因此需要进行配置。具体地，打开配置文件夹 hengDaProject 下的 settings.py 文件，在文件末尾添加代码：

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

上述配置可以告诉解释器当前项目的媒体资源文件(此处指图像)的存储根路径为/media/，即将所有上传的图片存储在项目根目录下的 media 文件夹中，结合数据库 ImageField 字段中参数 upload_to 的设置，最终上传的图片存储路径为/media/Award。

模型创建完成后需要将创建的模型同步到数据库系统中。在终端中首先输入命令：

```
python manage.py makemigrations
```

此时，会在终端中输出结果：

```
Migrations for 'aboutApp':
  aboutApp\migrations\0001_initial.py
    - Create model Award
```

此时，已经做好将模型数据存入数据库的准备。打开 aboutApp 应用中的 migrations 文件夹可以查看当前文件夹下已经创建了 0001_initial.py 文件，此文件即为在本地创建好的需要同步的数据模型文件。但是注意，此时并没有真正进行数据库同步，仅仅完成了同步的准备工作。接下来输入命令：

```
python manage.py migrate
```

控制台输出结果为：

```
Operations to perform:
  Apply all migrations: aboutApp, admin, auth, contenttypes, sessions
Running migrations:
  Applying aboutApp.0001_initial... 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 auth.0008_alter_user_username_max_length... OK
  Applying sessions.0001_initial... OK
```

此时才真正完成了数据库模型的同步操作，即在数据库中已经为 Award 模型创建好了对应的数据表。接下来可以对该数据库模型进行数据增删查改操作。可以直接在 Python Shell 中通过代码编辑数据库中的数据，但是这种通过代码对数据库进行增删查改的方式并不方便也不直观，而 Django 自带强大的后台管理系统，通过该后台管理系统可以方便地对创建的数据库模型进行管理和操作。

5.3.2 Django 后台管理系统

一个企业门户网站分为前台和后台两部分。前台主要为普通用户提供常规页面的访问，可以浏览基本信息。后台由网站的管理员负责网站数据的查看、添加、修改和删除。开

发一套完整的后台管理系统是一件异常烦琐的工作，为此，Django 提供了现成高效的后台管理系统，在我们创建项目的过程中已经自动生成了这样一个便捷的后台。

具体地，Django 能够根据定义的模型自动地生成管理模块，使用 Django 的管理功能只需要以下两步操作。

（1）创建超级管理员。

（2）注册模型类。

要使用 Django 的后台管理系统首先需要创建一个超级管理员账户来登录后台系统。具体地，在终端中输入命令：

```
python manage.py createsuperuser
```

此时会弹出提示需要输入超级管理员用户名：

```
Username (leave blank to use 'administrator'):
```

输入完成后按 Enter 键，会出现提示需要输入邮箱：

```
Email address:
```

输入完成后按 Enter 键，会出现提示需要输入超级管理员账户密码：

```
Password:
```

在输入过程中由于是密码因此输入时不会显示当前输入的字符。输入完成后直接按 Enter 键，会要求再次输入密码以完成密码的前后一致性检查。

```
Password (again):
```

输入完成并且两次密码输入正确后会出现：

```
Superuser created successfully.
```

此时表明超级管理员已经成功创建。接下来启动项目并访问 http://127.0.0.1:8000/admin，访问效果如图 5-4 所示。

图 5-4　Django 后台管理系统登录页面

如图 5-4 所示页面即为后台管理系统的登录页面，输入创建的超级管理员用户名和密码即可登录后台管理系统。登录后的页面如图 5-5 所示。

进入后台管理系统后可以看到当前已经注册在系统中的模型，包括 Groups 和 Users，这是系统默认提供的账户管理模型。单击 Users 可以看到如图 5-6 所示页面，在该页面中列出了当前的所有用户数据，此处由于我们并没有创建

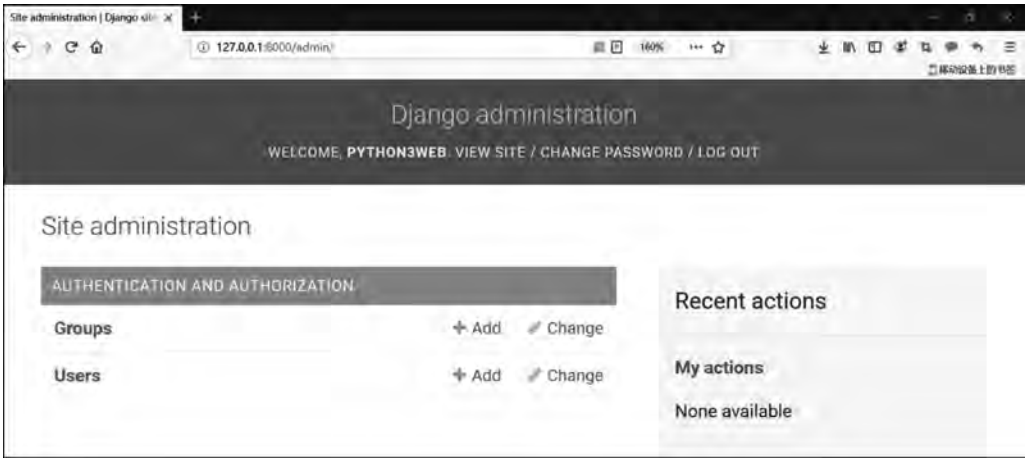

图 5-5　Django 后台管理系统主页面

图 5-6　Django 后台管理系统 Users 列表

其他用户数据，因此仅可看到超级管理员账户。

在后台管理系统中，可以通过可视化按钮以及系统提供的默认表单方便地对数据库模型进行操作。

下面继续完成本章开发任务。从图 5-5 中可以看到，当前管理系统主页面并没有 Award 模型的操作设置，这是因为我们在数据库中创建了 Award 模型，但是并没有将该模型注册到后台管理系统中，因此后台管理系统也就无法操作该模型。如果需要将模型注册到后台管理系统，只需要在 admin.py 文件中添加模型对应的注册信息即可。具体地，打开 aboutApp 中的 admin.py 文件，编辑代码如下。

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

class AwardAdmin(admin.ModelAdmin):
```

```
    list_display = ['description','photo']

admin.site.register(Award, AwardAdmin)
```

首先引入 Django 中提供的管理员模块 admin，然后从 models.py 文件中导入前面创建的 Award 类。接下来定义了一个名为 AwardAdmin 的荣誉管理类，该类继承自 admin 模块中的 ModelAdmin 类。在 AwardAdmin 中设置了展示列表 list_display 参数，在该参数中将需要编辑的模型字段添加进来，此处包括文字描述字段 description 和图像字段 photo。最后通过 admin.site.register()函数将 AwardAdmin 类与 Award 类进行绑定并实现注册。保存所有修改，刷新浏览页面可以看到在后台管理系统中新增加了 ABOUTAPP 组（对应 AboutApp 应用），在该组中可以操作 Award 类，如图 5-7 所示。

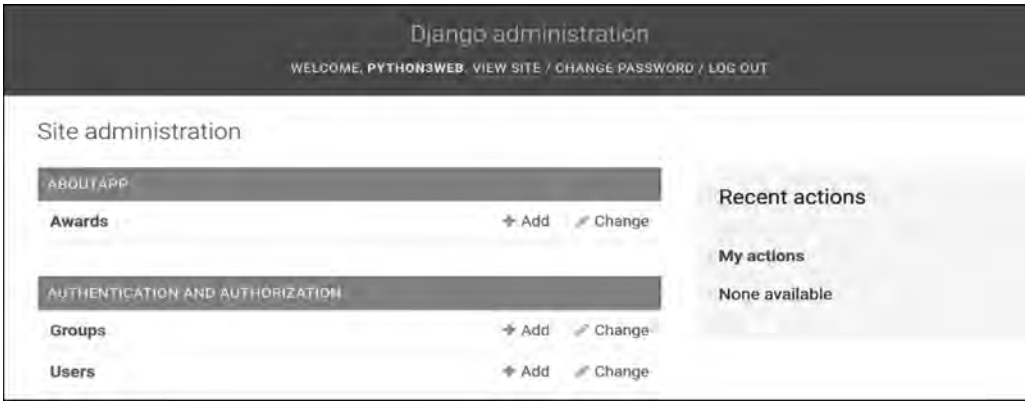

图 5-7　Django 后台管理系统添加注册模型

接下来，通过后台管理系统向 Awards 模型中添加几条数据用于后期使用。单击 Award 所在行对应的 Add 按钮，进入数据添加界面，如图 5-8 所示。

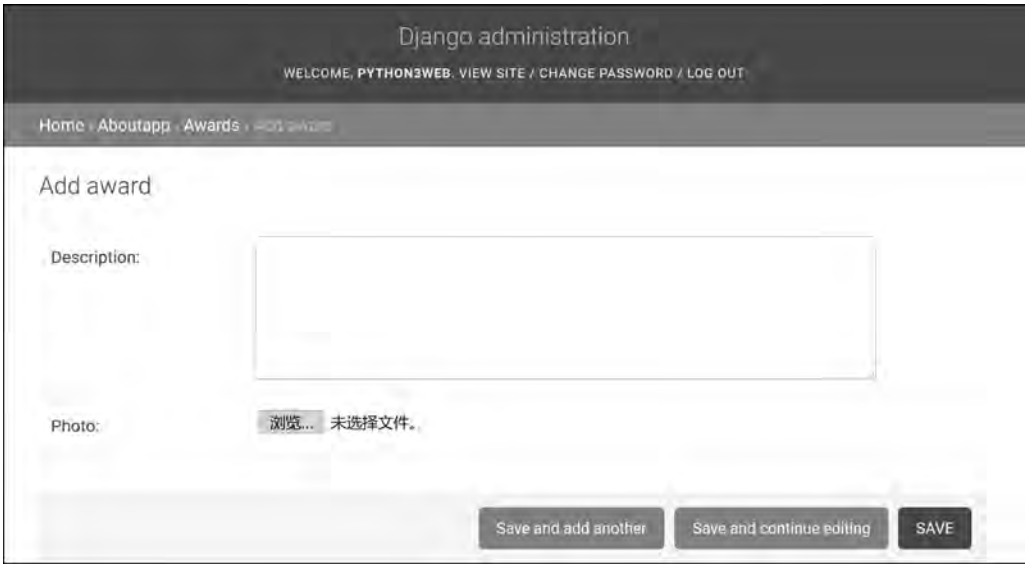

图 5-8　Django 后台管理系统向模型中添加数据

在定义 Award 模型时共有两个字段：description 和 photo，与后台数据添加界面中的两个字段一一对应。由于 description 采用了长文本字段 TextField 来声明，因此后台管理系统自动地为该字段形成多行输入框便于用户输入数据，而 photo 声明时使用了 ImageField 字段，因此后台管理系统自动形成文件上传按钮来执行数据添加操作。按照输入格式，在 description 中添加文字描述，然后单击“浏览”按钮选中一张图片（注意上传的照片名中不要含中文），最后单击右侧的 SAVE 按钮保存数据。

按照上述方式继续添加几条类似的 Award 数据，最终 Award 列表如图 5-9 所示。

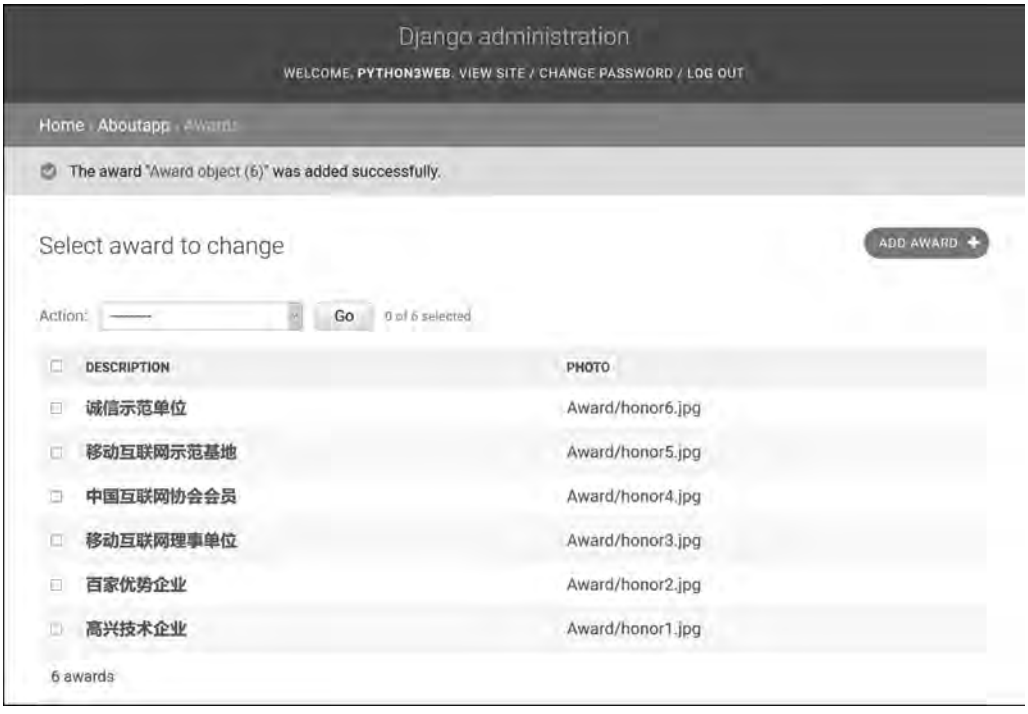

图 5-9　Django 后台管理系统 Award 模型数据列表

此时，Award 模型中所有数据均已存入数据库文件中。值得注意的是，对于图像 photo 字段，在数据库中并不是直接存储了该图像数据，而是存储了对应的图像路径。图像的真实存储路径根据 settings 文件的 MEDIA_URL 和 MEDIA_ROOT 来设置。具体地针对本实例项目，可以查看当前项目根目录下的 media 文件夹中是否存在对应的图片（media/Award 路径下）。

除了向模型添加数据以外，同样可以通过后台管理系统实现模型数据的删、改、查操作，具体的内容此处就不再一一展开演示。在数据库中有了上述的 Award 模型数据以后，就可以在用户请求页面的过程中动态地向页面嵌入数据实现动态页面的访问。5.3.3 节将会阐述如何通过 Django 从数据库中取出模型数据并插入到模板中进行动态页面渲染。

5.3.3　动态页面渲染

Django 提供了方便的 ORM 操作来对数据库模型进行管理。当用户在访问荣誉资质页面时，请求通过路由 URL 分发至 views.py 文件中的 honor()函数进行处理，该函数收到

请求后首先从数据库中取出 Award 模型的所有数据，然后将数据嵌入到 honor.html 模板文件中。

重新编辑 views.py 文件中的 honor()函数如下。

```
from .models import Award

def honor(request):
    awards = Award.objects.all()
    return render(request, 'honor.html', {
        'active_menu': 'about',
        'sub_menu': 'honor',
        'awards': awards,
    })
```

首先从当前应用下的 model.py 文件中导入 Award 模型，然后在 honor()函数中通过模型的 objects.all()函数得到一个查询集并存放于变量 awards 中。在页面渲染时通过 render()函数将 awards 变量以参数形式添加到页面中。本实例通过 Django 模型管理器的 objects.all()函数来获取模型数据信息。除此以外，Django 还提供了很多其他的查询功能，具体见表 5-2。此处，并不需要一次性地掌握所有查询语句，只需要在后续章节中通过项目实例逐步学会常见的调用方式即可。

表 5-2　Django 数据库常用查询方法

管理器方法	返回类型	说　明
模型类.objects.all()	QuerySet	返回表中所有数据
模型类.objects.filter()	QuerySet	返回符合条件的数据
模型类.objects.exclude()	QuerySet	返回不符合条件的数据
模型类.objects.order_by()	QuerySet	对查询结果集进行排序
模型类.objects.values()	QuerySet	返回一个列表，每个元素为一个字典
模型类.objects.reverse()	QuerySet	对排序的结果反转
模型类.objects.get()	模型对象	返回一个满足条件的对象； 如果没有找到符合条件的对象，会引发模型类.DoesNotExist 异常；如果找到多个，会引发模型类.MultiObjectsReturned 异常
模型类.objects.count()	int	返回查询集中对象的数目
模型类.objects.first()	模型对象	返回第一条数据
模型类.objects.last()	模型对象	返回最后一条数据
模型类.objects.exists()	bool	判断查询的数据是否存在

下面开始编辑 honor.html 页面。根据如图 5-2 所示效果，“荣誉资质”页面的整体设计和“企业概况”页面一致，不同之处在于主体部分采用图片列表的形式展示企业所获荣誉。在布局时可以采用 Bootstrap 现成的缩略图组件(样式类 thumbnail)略加修改即可实现。初始设计方案如下。

```
<div class="col-md-9">
    <div class="model-details-title">
```

```
            荣誉资质
        </div>
        <div class = "row">
            <div class = "col - sm - 6 col - md - 4">
                <div class = "thumbnail">
                    <img src = "{ % static 'img/honor1.jpg' % }">
                    <div class = "caption">
                        <p>2011 年加入互联网协会</p>
                    </div>
                </div>
            </div>
            <div class = "col - sm - 6 col - md - 4">
                <div class = "thumbnail">
                    <img src = "{ % static 'img/honor2.jpg' % }">
                    <div class = "caption">
                        <p>2012 年加入互联网协会</p>
                    </div>
                </div>
            </div>
        </div>
    </div>
```

上述代码共包含两个缩略图，每个缩略图包括一个<img>标签用于显示图像以及一个class="caption"的 标签用于显示图片对应的描述信息。每个缩略图在大屏浏览器下占4个栅格，在小屏浏览器上占6个栅格。上述代码采用了静态页面方式将图像路径和文字描述信息显式地写在HTML中，无法根据数据库信息动态地修改图片和文字数据。接下来需要实现根据后台honor()函数传入的awards参数将数据动态地写入HTML中，主要通过模板标签{% for %}{% endfor %}来实现。该模板标签可以动态地遍历传入的变量，实现页面数据循环写入。具体代码如下。

```
    <div class = "col - md - 9">
        <div class = "model - details - title">
            荣誉资质
        </div>
        <div class = "row">
            { % for award in awards % }
            <div class = "col - sm - 6 col - md - 4">
                <div class = "thumbnail">
                    <img src = "{{award.photo.url}}">
                    <div class = "caption">
                        <p>{{award.description}}</p>
                    </div>
                </div>
            </div>
            { % endfor % }
        </div>
    </div>
```

上述代码通过{% for award in awards %}语句逐个地取出 awards 中的每一条数据并赋值到新的临时变量 award 中，每个 award 都包含一项 photo 和 description 数据。在缩略图中分别使用模板变量{{award.photo.url}}和{{award.description}}对<img>的 src 属性以及段落<p>赋予动态内容。这种方式可以根据 awards 中的实际的条目数来生成缩略图。对该页面数据的增删查改不再需要开发人员变更代码实现，只需要通过后台管理系统操作数据即可完成数据更新。刷新页面后浏览效果如图 5-10 所示。

图 5-10　动态页面效果

可以看到每个缩略图的文字信息均已正确显示，但是图片信息没有显示出来。主要原因在于当前 debug 模式下没有将动态资源路径 MEDIA_URL 添加到静态路由 static 下。编辑配置文件夹 hengDaProject 下的 urls.py 文件，添加代码如下。

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

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
                          document_root = settings.MEDIA_ROOT)
```

保存修改后，重新刷新网页即可看到正确的效果图。至此，本节完成了数据库模型的导出和渲染。通过本节“荣誉资质”子模块的开发，相信读者已经掌握动态页面的制作流程和基本的数据库操作方法。后续章节的其他页面均采用这种动态页面的制作方式来实现。因此，希望读者能够牢牢掌握本节内容的知识点，多动手多实践，对于其中不清楚的地方可以结合本书配套资源代码来分析。

5.4　优化后台管理系统

在 5.3 节中介绍了 Django 的后台管理系统的使用方法，对于开发人员来说，掌握该后台管理系统的使用是必需的，但是该系统对于非开发人员来说其交互方式并不友好，包括语言（Django 后台管理系统默认语言为英文）和界面设计等。本节重点阐述如何对后台管理系统进行优化以方便今后将网站交付给实际客户使用。

5.4.1 登录界面优化

1. 界面汉化

Django 在创建项目时默认将英文作为项目主要语言，因此，在后台管理系统中大部分字段都是英文。下面首先对登录界面进行汉化处理。

打开项目配置文件 settings.py，找到其中的 LANGUAGE_CODE 字段，该字段用于设置整个项目的语言，这里需要改为中文支持。另外，需要修改时区字段 TIME_ZONE 为中国时区，对其进行修改如下。

```
LANGUAGE_CODE = 'zh-Hans'      #设置语言为中文
TIME_ZONE = 'Asia/Shanghai'    #设置中国时区
```

修改后保存并启动项目。访问后台管理系统，可以发现关键字段英文都已转换为中文，效果如图 5-11 所示。

2. 修改管理系统名称

Django 提供的后台管理系统默认的系统名称为“Django 管理”，而在实际交付给客户使用时需要按照网站主题定义后台系统名字，因此接下来需要完成后台管理系统名称的修改。

打开 aboutApp 应用下的 admin.py 文件，在文件末尾添加代码：

```
admin.site.site_header = '企业门户网站后台管理系统'
admin.site.site_title = '企业门户网站后台管理系统'
```

上述代码分别对管理系统头部和页面标题进行了修改，保存修改后刷新页面如图 5-12 所示。这里注意，由于我们创建的项目 hengDaProject 是一个多应用项目，上述代码修改只需要放置在任一应用下的 admin.py 文件中即可生效。

图 5-11 Django 后台管理系统登录界面汉化

图 5-12 修改 Django 后台管理系统名称

5.4.2 主界面优化

1. 模型名称修改

主界面部分首先来修改数据模型的显示。由于创建了 Award 模型，因此在 ABOUTAPP 下可以看到英文显示的“Awards”字样（默认会以模型名的复数形式表示）。尽管开发人员

知道该模型含义，但是对于非开发人员来说英文字样较为突兀，这里希望能够改成“获奖荣誉”。一种有效的解决思路就是在模型的创建过程中为模型创建一个中文别名，这样在后台管理系统中就可以用别名替代模型的真实名来显示。Django 为这种思路提供了简单的实现方法，只需要修改模型的 Meta 元信息即可。打开 aboutApp 应用下的 models.py 文件，为定义的 Award 模型添加 Meta 元信息说明，代码如下。

```
class Award(models.Model): #荣誉模型
  description = models.TextField(max_length=500, blank=True,
                                 null=True) #文字描述
  photo = models.ImageField(upload_to='Award/', blank=True) #图片

  class Meta:
      verbose_name = '获奖和荣誉'
      verbose_name_plural = '获奖和荣誉'
```

其中，verbose_name 字段即为模型定义的别名，verbose_name_plural 为别名对应的复数形式。保存后刷新页面即可看到如图 5-13 所示效果。

图 5-13　修改数据模型名称

2. 应用名称修改

接下来需要对应用名 ABOUTAPP 的显示进行修改。针对本章任务来说，需要将 ABOUTAPP 修改为“公司简介”。Django 在后台默认显示的应用的名称为创建 App 时的名称，需要修改这个 App 的名称达到定制的要求。从 Django 1.7 版本以后不再使用 app_label，修改 App 需要使用 AppConfig。这里只需要在应用的__init__.py 里面进行修改即可，打开 aboutApp 下的__init__.py 文件，添加代码如下。

```
from os import path
from django.apps import AppConfig

VERBOSE_APP_NAME = '公司简介'

def get_current_app_name(file):
    return path.dirname(file).replace('\\', '/').split('/')[-1]

class AppVerboseNameConfig(AppConfig):
    name = get_current_app_name(__file__)
    verbose_name = VERBOSE_APP_NAME

default_app_config = get_current_app_name(
    __file__) + '.__init__.AppVerboseNameConfig'
```

这里主要参考 Django 官方参考文档来实现，通过继承 AppConfig 类来设置 App 别名，该部分代码如果难以理解可以暂时不做深究，主要注意 VERBOSE_APP_NAME 字段，通

过修改该字段可以为应用添加别名。保存修改后刷新页面，效果图如图 5-14 所示。

图 5-14　修改应用名称

后续章节每个应用在后台管理界面中均按照上述方法进行名称修改，本书对此不再重复阐述。

5.4.3　列表界面优化

单击模型会进入模型列表页面，该页面中显示了模型的所有数据，注意到模型的每个字段依然是英文。接下来将对模型字段进行修改使得 description 和 photo 分别显示为“荣誉描述”和“荣誉照片”。解决方法与模型名称的修改基本一致，通过对模型每个字段取别名来进行显示。重新编辑 Award 模型中的 description 和 photo 字段，为每个字段添加 verbose_name 属性，具体修改如下。

```
description = models.TextField(max_length = 500,
                               blank = True,
                               null = True,
                               verbose_name = '荣誉描述')
photo = models.ImageField(upload_to = 'Award/',
                          blank = True,
                          verbose_name = '荣誉照片')
```

保存修改后刷新页面，效果如图 5-15 所示。

图 5-15　修改模型字段名称

小结

本章实现“公司简介”模块的开发，该模块共包含两个页面：一个静态页面（“企业概况”页面）和一个动态页面（“荣誉资质”页面）。静态页面主要学习模板的继承以及侧边导航栏的制作。动态页面则通过 Django 实现了“荣誉”模型的后台管理和页面显示，重点需要掌握 Django 数据模型创建方式、后台管理系统的使用技巧以及前端页面渲染动态数据方法。本章最后一节阐述如何对后台管理系统进行优化以方便非开发用户使用。本章内容侧重阐述后端 Python Web 编程知识，通过一个动态页面的制作来学习 Django 数据库的使用，后续章节的页面大部分均采用这种动态页面制作的方式实现。因此，读者需要牢牢掌握本章知识内容。