基于 Django 快速开发社会化网络书签系统（一）

——数据模型的定义

刘　班

摘　要：利用 Django 进行 Web 应用开发简洁、清晰、高效、安全。以连载形式详细讲解基于 Django 开发一个典型的社会化网络书签系统的主要过程。本期重点讲解该系统数据模型的建立过程。

关键词：Web 开发框架；Python；Django；网络书签系统

1　引言

传统的桌面型书签系统可以使用户在本地方便地保存网络书签信息，但用户在异地访问这些书签信息却非常困难。网络书签的出现很好地解决了这个问题，用户能通过任何一台联网计算机访问存储在网络数据库中的书签信息。近年来，随着 Web2.0 的流行，新一代社会化网络书签系统应运而生。这类书签系统除具有传统网络书签系统的主要功能外，还包含了很多 Web2.0 特色功能，比如标签云生成功能、RSS 订阅功能等。这些功能的加入，使网络书签系统的开发变得更为复杂。为了减轻开发人员的负担，加快开发的速度，可以采用目前非常流行的基于框架的软件开发技术。以连载形式详细讲解了在 Windows 下基于 Web2.0 开发框架 Django 构建一个典型的社会化网络书签系统的主要过程。这一期重点讲解该系统数据模型的建立过程。

2　准备工作

在进行正式开发之前，需要先建立一个名为 mysite 的项目，然后将本系统添加到其中，接着对项目配置文件 settings.py 进行如下修改：

（1）在 settings.py 文件的开头顺序添加用来指定该文件编码方式的代码 # -*- coding: utf-8 -*-和导入 os 模块的语句 import os。

（2）本系统采用 sqlite3 作为后端数据库，因此要将数据库引擎配置项 DATABASE_ENGINE 和数据库名称配置项 DATABASE_NAME 的值分别修改为 'sqlite3' 和 os.path.join (os.path.dirname (_file_) , 'bookmarks/bookmarksdb')。

（3）将时区配置项 TIME_ZONE 和语言代码配置项 LANGUAGE_CODE 的值分别修改为 'CCT' 和 'zh-cn'。

（4）将模板路径配置项 TEMPLATE_DIRS 的值修改为 (os.path.join (os.path.dirname (_file_) , 'bookmarks/templates/'))。

（5）分别将 Django 内置的后台管理应用、Django 内置的评论应用以及本系统（应用）的激活代码添加到配置项 INSTALLED_APPS 之中，如下所示：

```
INSTALLED_APPS = (
  'django.contrib.auth',
 'django.contrib.contenttypes',
 'django.contrib.sessions',
 'django.contrib.sites',
  """激活 Django 内置的后台管理应用。"""
 'django.contrib.admin',
  """激活 Django 内置的评论应用。"""
 'django.contrib.comments',
  """激活本系统(应用)。"""
 'mysite.bookmarks',
)
```

（6）将如下所示的配置代码添加到 settings.py 文件的末尾：

```
"""从 Django 默认的全局项目配置文件 global_settings.py 中导入模板内容处理器配置项。"""
from django.conf.global_settings import TEMPLATE_CONTEXT_PROCESSORS
"""激活 django.core.context_processors.request 模板内容处理器。"""
TEMPLATE_CONTEXT_PROCESSORS += (
  'django.core.context_processors.request',
)
"""指定本系统所在服务器主机名。"""
SITE_HOST = '127.0.0.1:8000'
"""指定电子邮件发件服务器地址，这里使用 163 邮箱的发件服务器地址。"""
EMAIL_HOST = 'smtp.163.com'
"""指定电子邮件发件服务器的端口号，不填写表示使用缺省的端口号 25。"""
EMAIL_PORT = ''
"""指定登录电子邮件发件服务器的用户名，这里要填写读者自己的 163 电子邮箱地址。"""
EMAIL_HOST_USER = '******@163.com'
"""指定登录电子邮件发件服务器的用户密码，这里要填写读者自己的 163 电子邮箱登录密码。"""
```

```
EMAIL_HOST_PASSWORD = '*********'
"""指定电子邮件发件人信息,其中的电子邮件地址填写读者自己的 163 电子邮箱地址。"""
DEFAULT_FROM_EMAIL = 'Django 社会化网络书签系统<******@163.com>'
```

对 settings.py 文件修改完成之后，还需要在本系统所在文件夹 bookmarks 中建立如图 1 所示的文件夹层次结构。

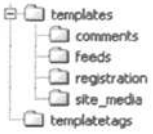

图 1 文件夹 bookmarks 中的子文件夹及其层次结构

3 数据模型定义

本系统的后端数据库 bookmarksdb 中包含以下 8 张数据表。

（1）bookmarks_link：链接信息表，用于存放用户书签中包含的链接地址（网址）信息。

（2）bookmarks_tag：标签信息表，用于存放系统中的标签信息。

（3）bookmarks_bookmark：书签信息表，用于存放用户提交（收藏）的书签信息。

（4）bookmarks_bookmark_tags：书签标签联系表，用于存放书签与标签之间的对应关系信息。

（5）bookmarks_invitation：邀请信息表，用于存放用户发出的站外好友邀请信息。

（6）bookmarks_friendship：用户好友信息表，用于存放用户之间的好友关系信息。

（7）auth_user：用户信息表，用于存放用户信息。此表对应的数据模型 User 由 Django 提供。

（8）comments_comment：书签评论信息表，用于存放用户对书签的评论信息。此表对应的数据模型 Comment 由 Django 提供。

这些数据表之间的关系如图 2 所示。

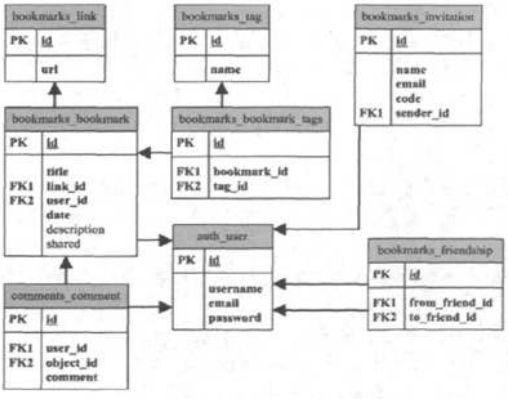

图 2 本系统的数据库中数据表之间的关系图

为了在本系统的数据库中生成以上这些数据表，首先应将与这些数据表相对应的数据模型定义添加到 models.py 文件中，如下所示：

```
"""以 utf-8 编码保存该文件。"""
# -*- coding: utf-8 -*-
"""导入建立各种数据模型所需的类。"""
from django.db import models
"""导入用于访问 settings.py 中特定配置项的 settings 对象。"""
from django.conf import settings
"""导入 connection 对象,该对象代表当前正在使用的数据库连接。"""
from django.db import connection
"""导入 Context 类,该类对象可用于封装模板变量及其值。"""
from django.template import Context
"""导入用于发送邮件的 send_mail 函数。"""
from django.core.mail import send_mail
""" 导入 Django 内置用户认证应用中包含的用户数据模型 User,该数据模型与用户信息表相对应。"""
from django.contrib.auth.models import User
"""导入用于建立模板对象的 get_template 函数。"""
from django.template.loader import get_template
""" 导入 Django 内置评论应用中包含的数据模型 Comment,该数据模型与书签评论信息表相对应。"""
from django.contrib.comments.models import Comment
"""导入 Django 内置 contenttypes 应用中包含的数据模型 ContentType。"""
from django.contrib.contenttypes.models import ContentType
""" 定义链接信息表数据模型 Link，该数据模型从 Model 类继承。"""
class Link(models.Model):
    """url 属性是一个 URLField 对象,对应于链接信息表的 url 字段,用于存取书签中包含的网址信息。该属性的值要求唯一。参数 help_text 用于设置后台管理应用中在该属性值输入文本框下的提示信息。"""
    url = models.URLField(
        unique=True,
        help_text='必须以 http://或 https://开头。'
        )
    """定义该数据模型对象的字符串表示。"""
    def __str__(self):
        return self.url
    """定义 Admin 内部类,使该数据模型能够被 Django 内置的后台管理应用管理。"""
    class Admin:
        pass
    """定义 Meta 内部类,指定该数据模型的一些元数据。"""
    class Meta:
        """此属性用于指定该数据模型在后台管理应用中的显示名称。"""
        verbose_name_plural = '链接信息'
```

```
"""定义管理者类 TagManager,该类从 Manager 类继承。"""
class TagManager(models.Manager):
  """定义一个管理者方法 user_tag_count,该方法以列表形式返回与特定用户(由参数 username 指定)相关的每一个标签对象(即 Tag 数据模型对象)的被引用次数。"""
  def user_tag_count(self, username):
    """得到当前数据库连接的一个游标对象,并保存到 cursor 变量中。"""
    cursor = connection.cursor()
    """通过游标对象执行 SQL 语句,得到一个包含与特定用户相关的每一个标签被引用次数的查询结果集。"""
    cursor.execute('select t.id, t.name, count(*) from bookmarks_bookmark b, bookmarks_tag t, bookmarks_bookmark_tags bt, auth_user u where b.id=bt.bookmark_id and b.user_id = u.id and t.id =bt.tag_id and u.username = "' + username + '" group by t.id order by t.name')
    """result_list 变量用于保存 Tag 数据模型对象的列表,其初始值为空列表。"""
    result_list = []
    """用 for 循环遍历上面的 SQL 语句执行后返回的查询结果集中包含的每一个结果行。"""
    for row in cursor.fetchall():
      """用一个查询结果行数据实例化一个 Tag 数据模型对象,并将该对象保存到变量 t 中。"""
      t = self.model(id=row[0], name=row[1])
      """为刚实例化的 Tag 数据模型对象添加一个 count 属性,将此属性的值设置为该对象的引用次数。"""
      t.count = row[2]
      """ 将经过以上步骤处理之后的 Tag 数据模型对象添加到 result_list 列表中。"""
      result_list.append(t)
    """返回变量 result_list 的值。"""
    return result_list
```

管理者类 Manager 中定义了在数据模型层面对数据表进行查询操作的接口。在默认情况下，Django 会自动为每个数据模型建立一个 Manager 类的对象，并将其保存到数据模型的 objects 属性中。用户可以根据需要对 Manager 类进行扩展（继承），上面定义的 TagManager 类便是如此，该类从 Manager 类继承，并添加了一个新方法 user_tag_count，这个方法用于返回特定用户相关标签的引用次数，使系统可以据此为特定用户生成相应的标签云图。

```
"""定义标签信息表数据模型 Tag。"""
class Tag(models.Model):
  """ name 属性是一个 CharField 对象，对应于标签信息表的 name 字段,用于存取标签名称信息。该属性值的最大长度为 16 个中英文字符且要求唯一。"""
  name = models.CharField(
    maxlength=16, unique=True,
    help_text='长度在 1-16 个中英文字符之间。'
    )
  """将属性 objects 中默认保存的 Manager 类对象修改为功能更强大的 TagManager 类对象。"""
  objects = TagManager()
  def __str__(self):
    return self.name
  """定义 get_absolute_url 方法,此方法用于返回该数据模型对象页面的 url 地址。"""
  def get_absolute_url(self):
    return '/tag/%s/' % self.id
  class Admin:
    pass
  class Meta:
    verbose_name_plural = '标签信息'
"""定义管理者类 BookmarkManager。"""
class BookmarkManager(models.Manager):
  """ 定义一个管理者方法 get_first_bookmark_ids_for_link,该方法用于返回所有特定共享书签对象 ID 的列表。"""
  def get_first_bookmark_ids_for_link(self):
    cursor = connection.cursor()
    """通过游标对象执行 SQL 语句,先将书签信息表中的所有共享书签记录按其包含的链接地址(网址)信息进行分组,然后取出每组中提交时间最早的共享书签 id 值组成查询结果集。"""
    cursor.execute('select b.id from bookmarks_bookmark b, (select link_id, min (date) md from bookmarks_bookmark where shared=1 group by link_id) bb where b.link_id =bb.link_id and b.date=bb.md')
    """将查询结果集中的共享书签 id 信息以列表形式保存到变量 result_list 中。"""
    result_list =[row[0] for row in cursor.fetchall()]
    return result_list
"""定义书签信息表数据模型 Bookmark。"""
class Bookmark(models.Model):
  """title 属性是一个 CharField 对象,对应于书签信息表的 title 字段,用于存取书签名称信息。该属性值的最大长度为 32 个中英文字符。"""
  title = models.CharField(
    maxlength=32,
    help_text='长度在 1-32 个中英文字符之间。'
    )
  """link 属性是一个 ForeignKey 对象，对应于书签信息表的 link_id 字段。该字段值必须引用链接信息表的 id 字段值,是书签信息表的一个外键。"""
  link = models.ForeignKey(Link)
  """user 属性是一个 ForeignKey 对象，对应于书签信息表的 user_id 字段。该字段值必须引用用户信息表的 id 字段值,是书签信息表的另一个外键。"""
  user = models.ForeignKey(User)
  """date 属性是一个 DateTimeField 对象，对应于书签信息表的 date 字段，用于存取书签建立时间信息。其中的参数 auto_now_add 表示在新建一个该数据模型对象时,自动设置此属性值为当前时间。"""
  date = models.DateTimeField(auto_now_add=True)
```

```
    """tags 属性是一个 ManyToManyField 对象，Django 会根据建立该对象时传递的 Tag 数据模型自动生成一张书签标签联系表 bookmarks_bookmark_tags。在书签信息表中没有直接和 tags 属性相对应的字段。"""
    tags = models.ManyToManyField(Tag)
    """description 属性是一个 TextField 对象，对应于书签信息表的 description 字段，用于存取书签的描述信息。该属性值的最大长度为 100 个中英文字符且允许为空字符串。"""
    description = models.TextField(
      blank=True, maxlength=100,
      help_text='长度在 0-100 个中英文字符之间。'
      )
    """shared 属性是一个 BooleanField 对象，对应于书签信息表的 shared 字段。该属性的默认值为 False，表示不共享书签对象。"""
    shared = models.BooleanField(default=False)
    """将属性 objects 中默认保存的 Manager 类对象修改为功能更强大的 BookmarkManager 类对象。"""
    objects = BookmarkManager()
    def _str_(self):
      return '%s, %s' % (self.user.username, self.link.url)
    def get_absolute_url(self):
      return '/bookmark/%s/' % self.id
    class Admin:
      """此属性用于指定在后台管理应用中显示该数据模型的哪些属性。"""
      list_display = ('title', 'link', 'user', 'date')
      """此属性用于指定在后台管理应用中按该数据模型的哪些属性对其对象进行筛选。"""
      list_filter = ('user',)
      """此属性用于指定在后台管理应用中按该数据模型的哪些属性对其对象进行排序。"""
      ordering = ('title', 'date')
      """此属性用于指定在后台管理应用中按该数据模型的哪些属性对其对象进行搜索。"""
      search_fields = ('title',)
    class Meta:
      verbose_name_plural = '书签信息'
    """定义一个用于返回书签对象用户评分均值的方法 get_avg_rating1。此方法可以被该数据模型的任何一个对象调用。"""
    def get_avg_rating1(self):
      """得到数据模型 Bookmark 所对应的 ContentType 数据模型对象，然后将该对象保存到变量 ctype 中。"""
      ctype = ContentType.objects.get_for_model(self)
      """得到与调用 get_avg_rating1 方法的书签对象（Bookmark 数据模型对象）相关的所有评论对象列表，并将该列表保存到变量 comments 中。"""
      comments = Comment.objects.filter (content_type = ctype, object_id=self.id)
      rating_list = []
      rating_total = 0
      """用 for 循环遍历变量 comments 中的每一个评论对象。"""
      for comment in comments:
        """如果用户在评论的同时也对书签进行了评分。"""
        if comment.rating1:
          """将用户评分添加到评分列表变量 rating_list 中。"""
          rating_list.append(comment.rating1)
          """将用户评分累加到变量 rating_total 中。"""
          rating_total += comment.rating1
      """如果变量 rating_total 的值不为 0。"""
      if rating_total:
        """计算用户评分均值，然后将该值保存到变量 avg 中。"""
        avg = round(rating_total * 1.0 / len(rating_list), 1)
        """返回变量 avg 的值。"""
        return avg
      """如果变量 rating_total 的值为 0（说明没有用户对书签进行评分）。"""
      else:
        return 0
```

Django 内置的 contenttypes 应用用于跟踪记录各种基于 Django 构建的应用中包含的数据模型，并且为访问这些数据模型提供了一个高层次的通用接口。contenttypes 应用包含一个数据模型 ContentType，该数据模型的每一个对象实例都与一个具体的数据模型相对应。

```
"""定义用户好友信息表数据模型 Friendship。"""
class Friendship(models.Model):
  """from_friend 属性是一个 ForeignKey 对象，该属性对应于用户好友信息表的 from_friend_id 字段。该字段值必须引用用户信息表的 id 字段值，是用户好友信息表的一个外键。from_friend 属性用于表示将 to_friend 属性指定用户添加为好友的用户。参数 related_name 会使每个 User 数据模型对象都包含一个 friend_set 属性，表示该用户对象全部好友的集合。"""
  from_friend = models.ForeignKey(
    User, related_name='friend_set'
    )
  """to_friend 属性是一个 ForeignKey 对象，该属性对应于用户好友信息表的 to_friend_id 字段。该字段值必须引用用户信息表的 ID 字段值，是用户好友信息表的另一个外键。to_friend 属性用于表示被 from_friend 属性指定用户添加为好友的用户。参数 related_name 会使每个 User 数据模型对象都包含一个 to_friend_set 属性，表示将该用户对象添加为好友的所有用户对象的集合。"""
  to_friend = models.ForeignKey(
    User, related_name='to_friend_set'
    )
  def_str_(self):
    return '%s, %s' % (
      self.from_friend.username,
      self.to_friend.username
      )
  class Admin:
    pass
  class Meta:
```

```
    """此属性用于限定 from_friend 和 to_friend 两个属性值组合的唯一性。"""
    unique_together = (('from_friend', 'to_friend'),)
    verbose_name_plural = '用户好友信息'
```

如果在 FriendShip 数据模型中定义属性 from_friend 和 to_friend 时不指定 related_name 参数，Django 则默认给每个 User 数据模型对象都添加两个 friendship_set 属性。为了避免属性名称冲突，必须指定 related_name 参数。

```
"""定义邀请信息表数据模型 Invitation。"""
class Invitation(models.Model):
  """name 属性是一个 CharField 对象，对应于邀请信息表的 name 字段，用于存取被邀请人的姓名或昵称信息。该属性值的最大长度为 16 个中英文字符。"""
  name = models.CharField(
    maxlength=16,
    help_text='长度在 1-16 个中英文字符之间。'
    )
  """email 属性是一个 EmailField 对象，对应于邀请信息表的 e-mail 字段，用于存取被邀请人的电子邮件地址信息。"""
  email = models.EmailField()
  """code 属性是一个 CharField 对象，对应于邀请信息表的 code 字段，用于存取邀请码信息。该属性值的最大长度为 20 个字符。"""
  code = models.CharField(
    maxlength=20,
    help_text='长度在 1-20 个字符之间。'
    )
  """sender 属性是一个 ForeignKey 对象，对应于邀请信息表的 sender_id 字段。sender_id 字段值必须引用用户信息表的 ID 字段值，是邀请信息表的一个外键。"""
  sender = models.ForeignKey(User)
  def _str_(self):
    return '%s, %s' % (self.sender.username, self.email)
  class Admin:
    pass
  class Meta:
    verbose_name_plural = '邀请信息'
  """定义一个向站外好友发送邀请邮件的方法 send，此方法可以被该数据模型的任何一个对象调用。"""
  def send(self):
    """将邀请邮件主题保存到变量 subject 中。"""
    subject = '邀请您加入 Django 社会化网络书签系统'
    """ 生成接受邀请链接字符串，然后将其保存到变量 link 中。"""
    link = 'http://%s/friend/accept/%s/' % (
      """从 settings.py 中取出本系统所在服务器主机名。"""
      settings.SITE_HOST,
      """得到为本次邀请生成的邀请码。"""
      self.code
      )
    """将 get_template 函数返回的邀请邮件模板对象保存到变量 template 中。"""
    template = get_template('invitation_email.html')
    """将邀请邮件模板中要用到的模板变量及其值封装到一个 Context 对象中，然后将该对象保存到变量 context 中。"""
    context = Context({
      'name': self.name,
      'link': link,
      'sender': self.sender.username
      })
    """ 用模板变量值替换邀请邮件模板中相应的模板变量，生成邀请邮件正文，然后将其保存到变量 message 中。"""
    message = template.render(context)
    """调用 send_mail 函数正式将邀请邮件发送到被邀请站外好友的邮箱中。send_mail 函数的参数 settings.DEFAULT_FROM_EMAIL 表示电子邮件发件人信息；参数 self.e-mail 表示收件人电子邮件地址。"""
    send_mail(
      subject, message,
      settings.DEFAULT_FROM_EMAIL, [self.email]
      )
```

从以上 send 方法的定义可知，在将邀请邮件发送给被邀请人之前，需要先用存放在 templates 文件夹（该文件夹用于存放本系统的模板文件）中的邀请邮件模板 invitation_email.html 生成具体的邀请邮件。该模板的内容如下所示：

```
<! --此文件要以 utf-8 编码保存-->
{{name}},您好!
{{sender}} 诚邀您加入 Django 社会化网络书签系统。
  加入本系统之后，可以将您喜爱的网址以书签形式保存到其中，并可以和您的好友分享您的网址书签!
  您可以单击下面的链接，接受{{sender}}的邀请:
  {{link}}
(如果您的 Email 客户端程序未将上面的链接转换为可单击形式，那么请将该链接拷贝到浏览器的地址栏进入本系统的注册页面。)
--Django 社会化网络书签系统
```

该模板中的 {{name}}、{{sender}} 以及 {{link}} 用于在将模板 invitation_email.html 转换为一封具体的邀请邮件时分别得到模板变量 name、sender 和 link 的值。

在邀请邮件的正文中包含了一种称为邀请码的随机码，设置此邀请码有两个作用：(1) 可以使用该邀请码验证邀请信息是否从本系统发出。(2) 被邀请人接受邀请后能通过邀请码从邀请信息表中得到发出邀请的用户信息，从而当被邀请人注册后，可以建立发出邀请用户和接受这一邀请用户之间的好友关系。

数据模型编写完之后，就可以通过执行命令 manage.py syncdb 为本系统生成后端数据库 bookmarksdb，并在该数据库中添加支持本系统运行的数据表（注意：在该命令执行过程中要按提示输入一个后台管理用到的管理员帐号和密码）。

以上，全面介绍了系统数据模型的建立过程。

（收稿日期：2011-04-11）