第一部分

Python 3.x

Django1.8.2

目标：

* 1个投票应用

包含两部分：

* 一个公开的网站，可以让访客查看投票的结果并让他们进行投票
* 一个后台管理网站，你可以添加、修改和删除选票

**$** python -c "import django; print(django.get\_version())"

如果Django已经安装，可以看一下安装的版本号。如果没有安装，你会看到一个“No module named django”的错误。

创建一个项目

初始化设置，一个Django框架开发的网站，创建项目后我们需要的配置的东西，包括数据库的配置、针对Django的配置选项和APP的配置选项。

在命令终端中，用c 到你想要用来保存代码的目录。

**$** django-admin startproject mysite

运行以后生成一个mysite目录。

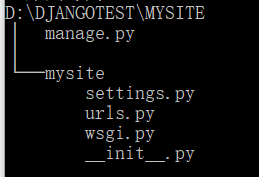
注意：

给项目命名的时候，不能和python或Django的内部组件名称相同。

代码应该放在哪里？

你可能习惯于将代码放在Web服务器的文档根目录下（例如**/var/www**）。但是对于Django，你不该这么做。将Python代码放在你的Web服务器的根目录不是个好主意，因为它可能会有让别人在网上看到你的代码的风险。这样不安全。将你的代码放置在Web服务器根目录**以外**的地方，例如**/home/mycode**。

生成了什么：



* 外层的mysite/根目录仅仅是一个容器，它的命名对Django无关紧要；可以换成任何名字。
* Manage.py：一个命令行工具，可以使用多种方式对Django项目进行交互。
* 内层的mysite/目录是你的项目中的真正的python包。它是你导入任何东西时需使用的Python包的名字（如：mysite.urls）。
* Mysite/\_\_init\_\_.py：一个空文件，他告诉Python这个目录应该被看做一个Python包。
* Mysite/settings.py：该Django项目的设置/配置。
* Mysite/urls.py：该Django项目的URL声明；你的Django站点的“目录”。
* Mysite/wsgi.py：用于你的项目的与WSGI兼容的web服务器入口。

数据库的建立

现在编辑mysite/settings.py。它是一个用模块级别变量表示Django配置的普通Python模块。

默认数据库是SQLite。SQLite包含在Python中，所以使用的时候不需要安装其它的东西来支持数据库。

当你编辑**mysite/settings.py**时，请设置[**TIME\_ZONE**](http://python.usyiyi.cn/documents/django_182/ref/settings.html#std:setting-TIME_ZONE)为你自己的时区

另外，请注意文件顶端的**[INSTALLED\_APPS](http://python.usyiyi.cn/documents/django_182/ref/settings.html" \l "std:setting-INSTALLED_APPS)**设置。它保存这个Django实例中激活的所有的Django应用的名字。 应用可以在多个项目中使用，而且你可以将这些应用打包和分发给其他人在他们的项目中使用。

默认情况下，[**INSTALLED\_APPS**](http://python.usyiyi.cn/documents/django_182/ref/settings.html#std:setting-INSTALLED_APPS)包含下面的应用，它们都是Django 与生俱来的：

* [**django.contrib.admin**](http://python.usyiyi.cn/documents/django_182/ref/contrib/admin/index.html#module-django.contrib.admin) —— 管理站点。你将在[本教程的第2部分](http://python.usyiyi.cn/documents/django_182/intro/tutorial02.html)使用到它。
* [**django.contrib.auth**](http://python.usyiyi.cn/documents/django_182/topics/auth/index.html#module-django.contrib.auth) —— 认证系统。
* [**django.contrib.contenttypes**](http://python.usyiyi.cn/documents/django_182/ref/contrib/contenttypes.html#module-django.contrib.contenttypes) —— 用于内容类型的框架。
* [**django.contrib.sessions**](http://python.usyiyi.cn/documents/django_182/topics/http/sessions.html#module-django.contrib.sessions) —— 会话框架。
* [**django.contrib.messages**](http://python.usyiyi.cn/documents/django_182/ref/contrib/messages.html#module-django.contrib.messages) —— 消息框架。
* [**django.contrib.staticfiles**](http://python.usyiyi.cn/documents/django_182/ref/contrib/staticfiles.html#module-django.contrib.staticfiles) —— 管理静态文件的框架。

这些应用，默认包含在Django中，以方便通用场合下使用。

然而上面的部分应用至少需要使用一个数据库表，因此我们需要在使用它们之前先在数据库中创建相应的表。要做到这一点，请运行以下命令：

**$** python manage.py migrate

migrate查看INSTALLED\_APPS设置并根据mysite/settings.py文件中的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（我们稍后会讲到）。user如果你有兴趣，可以运行你的数据库的命令行客户端并输入dt (PostgreSQL), SHOW TABLES; (MySQL)或.schema (SQLite)来显示Django创建的表。

对于极简主义者来说

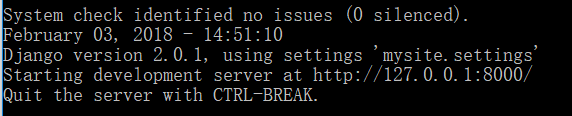
就像我们上面说到的，以上包含的默认应用用于常见的场景，但并不是每个人都需要它们。 如果你不需要它们中的任何一个或所有应用，可以在运行[**migrate**](http://python.usyiyi.cn/documents/django_182/ref/django-admin.html#django-admin-migrate)之前从[**INSTALLED\_APPS**](http://python.usyiyi.cn/documents/django_182/ref/settings.html#std:setting-INSTALLED_APPS)中自由地注释或删除相应的行。[**migrate**](http://python.usyiyi.cn/documents/django_182/ref/django-admin.html#django-admin-migrate) 命令将只为[**INSTALLED\_APPS**](http://python.usyiyi.cn/documents/django_182/ref/settings.html#std:setting-INSTALLED_APPS)中的应用运行数据库的迁移。

**开发服务器[¶](http://python.usyiyi.cn/documents/django_182/intro/tutorial01.html" \l "the-development-server" \o "Permalink to this headline)**

让我们验证一下你的Django项目是否工作。 如果你不在外层的**mysite**目录下，那么进入这个目录，然后运行以下命令：

**$** python manage.py runserver

你将看到命令行下输出了以下内容：



这表明你已经启动了Django开发服务器，一个用纯Python写的轻量级Web服务器。 我们在Django中内置了它，这样你就可以在不配置用于生产环境的服务器 —— 例如Apache —— 的情况下快速开发出产品，直到你准备好上线。

请注意：**不要**在任何生产环境使用这个服务器。它仅仅是用于在开发中使用。（我们的重点是编写Web框架，非Web服务器。）

既然服务器已经运行，请用你的浏览器访问 <http://127.0.0.1:8000/>。在淡蓝色背景下，你将看到一个“Welcome to Django”的页面。 它运行成功了！

更改端口

默认情况下，[**runserver**](http://python.usyiyi.cn/documents/django_182/ref/django-admin.html#django-admin-runserver)命令在内部IP的8000端口启动开发服务器。

如果你需改变服务器的端口，把要使用的端口作为一个命令行参数传递给它。 例如，这个命令在8080端口启动服务器：

**$** python manage.py runserver 8080

如果你需改变服务器的IP地址，把IP地址和端口号放到一起。 因此若要监听所有的外网IP，请使用（如果你想在另外一台电脑上展示你的工作，会非常有用）：

**$** python manage.py runserver 0.0.0.0:8000

开发服务器会根据需要自动重新载入Python代码。 你不必为了使更改的代码生效而重启服务器。 然而，一些行为比如添加文件，不会触发服务器的重启，所以在这种情况下你需要手动重启服务器。

## 创建模型[¶](http://python.usyiyi.cn/documents/django_182/intro/tutorial01.html#creating-models)

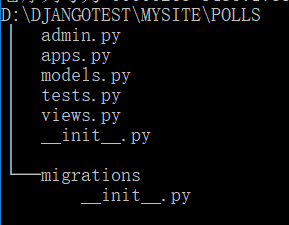
现在，你的开发环境 —— 一个“项目” —— 已经建立起来，你将开始在上面做一些东西。

你编写的每个Django应用都是遵循特定约定且包含一个Python包。 Django自带一个工具，它可以自动生成应用的基本目录结构，这样你就能专心于书写代码而不是创建目录。

项目和应用之间有什么不同？ 应用是一个Web应用程序，它完成具体的事项 —— 比如一个博客系统、一个存储公共档案的数据库或者一个简单的投票应用。 项目是一个特定网站中相关配置和应用的集合。一个项目可以包含多个应用。一个应用可以运用到多个项目中去。

你的应用可以放在Python path上的任何位置。在本教程中，我们将在你的manage.py文件同级目录创建我们的投票应用，以便可以将它作为顶层模块导入，而不是mysite的子模块。

这将创建一个目录**polls**，它的结构如下：



我们的polls应用将基于这个目录结构。

当编写一个数据库驱动的Web应用时，第一步就是定义该应用的模型 —— 本质上，就是定义该模型所对应的数据库设计及其附带的元数据。

原理

模型指出了数据的唯一、明确的真实来源。 它包含了正在存储的数据的基本字段和行为。 Django遵循[DRY (Don't repeat yourself)原则](http://python.usyiyi.cn/documents/django_182/misc/design-philosophies.html#dry)。这个原则的目标是在一个地方定义你的数据模型，并自动从它获得需要的信息。

迁移工具也符合以上哲学 —— 这不同于Ruby On Rails中的迁移；例如，迁移完全依照于你的模型文件且本质上只是一个历史记录，Django通过这个历史记录更新你的数据库模式使它与你现在的模型文件保持一致。

在这个简单的投票应用中，我们将创建两个模型： **Question**和**Choice**。**Question**对象具有一个question\_text（问题）属性和一个publish\_date（发布时间）属性。 **Choice**有两个字段：选择的内容和选择的得票统计。 每个**Choice**与一个**Question**关联。

这些概念通过简单的Python类来表示。 编辑**polls/models.py**文件，并让它看起来像这样：

**from** django.db **import** models  
**class** Question(models.Model):  
 question\_text = models.CharField(max\_length = 20)  
 pub\_date = models.DateTimeField(**'date published'**)  
*# Create your models here.***class** Choice(models.Model):  
 question = models.ForeignKey(Question)  
 choice\_text = models.CharField(max\_length = 200)  
 votes = models.IntegerField(default=0)

上述代码非常直观。每个模型都用一个类表示，该类继承自[**django.db.models.Model**](http://python.usyiyi.cn/documents/django_182/ref/models/instances.html#django.db.models.Model)。每个模型都有一些类变量，在模型中每个类变量都代表了数据库中的一个字段。

每个字段通过[**Field**](http://python.usyiyi.cn/documents/django_182/ref/models/fields.html#django.db.models.Field)类的一个实例表示 —— 例如字符字段[**CharField**](http://python.usyiyi.cn/documents/django_182/ref/models/fields.html#django.db.models.CharField)和日期字段[**DateTimeField**](http://python.usyiyi.cn/documents/django_182/ref/models/fields.html#django.db.models.DateTimeField)。这种方法告诉Django，每个字段中保存着什么类型的数据。

每个**[Field](http://python.usyiyi.cn/documents/django_182/ref/models/fields.html" \l "django.db.models.Field" \o "django.db.models.Field)** 实例的名字（例如**question\_text** 或 **pub\_date**）就是字段的名字，并且是机器可读的格式。你将在Python代码中使用到它的值，并且你的数据库将把它用作表的列名。

你可以使用[**Field**](http://python.usyiyi.cn/documents/django_182/ref/models/fields.html#django.db.models.Field)的第一个参数来指定一个人类可读的名字，这是可选的。它在Django的内省机制中有使用，而且可以兼作文档。 如果没有提供这个参数，Django将使用那个机器可读的名字（实例名）。 在这个例子中，我们只为**Question.pub\_date**定义一个人类可读的名字。 对于这个模型中其他的字段，机器可读的名字（实例名）足以充分地表达出它的含义。

某些**[Field](http://python.usyiyi.cn/documents/django_182/ref/models/fields.html" \l "django.db.models.Field" \o "django.db.models.Field)** 类具有必选的参数。例如，CharField要求您给它一个max\_length。 这个参数不仅用于数据库模式，而且数据验证中也会用到，我们稍后会看到。[**Field**](http://python.usyiyi.cn/documents/django_182/ref/models/fields.html#django.db.models.Field) 还具有各种可选参数。在这个例子中，我们设置**votes**字段的[**默认值**](http://python.usyiyi.cn/documents/django_182/ref/models/fields.html#django.db.models.Field.default) 为0。

最后，注意我们使用[**ForeignKey**](http://python.usyiyi.cn/documents/django_182/ref/models/fields.html#django.db.models.ForeignKey)定义了一个关联。它告诉Django每个**Choice**都只关联一个**Question**。Django支持所有常见的数据库关联：多对一、多对多和一对一。

## 激活模型[¶](http://python.usyiyi.cn/documents/django_182/intro/tutorial01.html#activating-models)

上面那段简短的模型代码给了Django很多信息。 有了这些代码，Django就能够自动完成以下两个功能：

* 为该应用创建数据库表（**CREATE TABLE** 语句）。
* 为**Question**对象和**Choice**对象创建一个访问数据库的python API。

但是，我们首先得告诉项目：**polls**应用已经安装。

原理:

Django 应用是可以“热插拔”的，即可以在多个项目中使用同一个应用，也可以分发这些应用， 因为它们不需要与某个特定的Django安装绑定。

再次编辑**mysite/settings.py**文件，并修改[**INSTALLED\_APPS**](http://python.usyiyi.cn/documents/django_182/ref/settings.html#std:setting-INSTALLED_APPS)设置以包含字符串**'polls'**。所以它现在是这样的：

mysite/settings.py

INSTALLED\_APPS **=** (

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'polls',

)

现在Django知道要包含**polls**应用。 让我们运行另外一个命令：

**$** python manage.py makemigrations polls

你应该看到类似下面的内容：

Migrations for 'polls':

0001\_initial.py:

- Create model Question

- Create model Choice

- Add field question to choice

通过运行**makemigrations**告诉Django，已经对模型做了一些更改（在这个例子中，你创建了一个新的模型）并且会将这些更改记录为迁移文件。

迁移是 Django 如何储存模型的变化（以及您的数据库模式），它们只是磁盘上的文件。如果愿意，你可以阅读这些为新模型建立的迁移文件；这个迁移文件就是  **polls/migrations/0001\_initial.py**。不用担心，Django不要求你在每次Django生成迁移文件之后都要阅读这些文件，但是它们被设计成可人为编辑的形式，以便你可以手工稍微修改一下Django的某些具体行为。

有一个命令可以运行这些迁移文件并自动管理你的数据库模式 —— 它叫做**[migrate](http://python.usyiyi.cn/documents/django_182/ref/django-admin.html" \l "django-admin-migrate)**，我们一会儿会用到它 —— 但是首先，让我们看一下迁移行为将会执行哪些SQL语句。**[sqlmigrate](http://python.usyiyi.cn/documents/django_182/ref/django-admin.html" \l "django-admin-sqlmigrate)**命令接收迁移文件的名字并返回它们的SQL语句：

**$** python manage.py sqlmigrate polls 0001

你应该会看到类似如下的内容（为了便于阅读我们对它重新编排了格式）：

**BEGIN**;

**CREATE** **TABLE** "polls\_choice" (

"id" serial **NOT** **NULL** **PRIMARY** **KEY**,

"choice\_text" varchar(200) **NOT** **NULL**,

"votes" integer **NOT** **NULL**

);

**CREATE** **TABLE** "polls\_question" (

"id" serial **NOT** **NULL** **PRIMARY** **KEY**,

"question\_text" varchar(200) **NOT** **NULL**,

"pub\_date" **timestamp** **with** time **zone** **NOT** **NULL**

);

**ALTER** **TABLE** "polls\_choice" **ADD** **COLUMN** "question\_id" integer **NOT** **NULL**;

**ALTER** **TABLE** "polls\_choice" **ALTER** **COLUMN** "question\_id" **DROP** **DEFAULT**;

**CREATE** **INDEX** "polls\_choice\_7aa0f6ee" **ON** "polls\_choice" ("question\_id");

**ALTER** **TABLE** "polls\_choice"

**ADD** **CONSTRAINT** "polls\_choice\_question\_id\_246c99a640fbbd72\_fk\_polls\_question\_id"

**FOREIGN** **KEY** ("question\_id")

**REFERENCES** "polls\_question" ("id")

**DEFERRABLE** **INITIALLY** **DEFERRED**;

**COMMIT**;

请注意以下几点：

* 输出的具体内容会依据你使用的数据库而不同。 以上例子使用的数据库是PostgreSQL。
* 表名是自动生成的，由app的名字（**polls**）和模型名字的小写字母组合而成 —— **question**和**choice**。（你可以重写这个行为。)
* 主键（id）是自动添加的。（你也可以重写这个行为。)
* 按照惯例，Django会在外键的字段名后面添加 **"\_id"**。（是的，你依然可以重写这个行为。）
* 外键关系由**FOREIGN KEY**约束显式声明。不用在意**DEFERRABLE**部分；它只是告诉PostgreSQL直到事务的最后再执行外键关联。
* 这些SQL语句是针对你所使用的数据库定制的，所以会为你自动处理某些数据库所特有的字段例如**auto\_increment** (MySQL)、 **serial** (PostgreSQL)或**integer primary key autoincrement** (SQLite) 。在处理字段名的引号时也是如此 —— 例如，使用双引号还是单引号。
* [**sqlmigrate**](http://python.usyiyi.cn/documents/django_182/ref/django-admin.html#django-admin-sqlmigrate)命令并不会在你的数据库上真正运行迁移文件 —— 它只是把Django 认为需要的SQL打印在屏幕上以让你能够看到。 这对于检查Django将要进行的数据库操作或者你的数据库管理员需要这些SQL脚本是非常有用的。

如果有兴趣，你还可以运行[**python manage.py check**](http://python.usyiyi.cn/documents/django_182/ref/django-admin.html#django-admin-check)；它会检查你的项目中的模型是否存在问题，而不用执行迁移或者接触数据库。

现在，再次运行[**migrate**](http://python.usyiyi.cn/documents/django_182/ref/django-admin.html#django-admin-migrate)以在你的数据库中创建模型所对应的表：

**$** python manage.py migrate

Operations to perform:

Synchronize unmigrated apps: staticfiles, messages

Apply all migrations: admin, contenttypes, polls, auth, sessions

Synchronizing apps without migrations:

Creating tables...

Running deferred SQL...

Installing custom SQL...

Running migrations:

Rendering model states... DONE

Applying <migration name>... OK

[**migrate**](http://python.usyiyi.cn/documents/django_182/ref/django-admin.html#django-admin-migrate)命令会找出所有还没有被应用的迁移文件（Django使用数据库中一个叫做**django\_migrations**的特殊表来追踪哪些迁移文件已经被应用过），并且在你的数据库上运行它们 —— 本质上来讲，就是使你的数据库模式和你改动后的模型进行同步。

迁移功能非常强大，可以让你在开发过程中不断修改你的模型而不用删除数据库或者表然后再重新生成一个新的 —— 它专注于升级你的数据库且不丢失数据。 我们将在本教程的后续章节对迁移进行深入地讲解，但是现在，请记住实现模型变更的三个步骤：

* 修改你的模型（在**models.py**文件中）。
* 运行[**python manage.py makemigrations**](http://python.usyiyi.cn/documents/django_182/ref/django-admin.html#django-admin-makemigrations) ，为这些修改创建迁移文件
* 运行[**python manage.py migrate**](http://python.usyiyi.cn/documents/django_182/ref/django-admin.html#django-admin-migrate) ，将这些改变更新到数据库中。

将生成和应用迁移文件的命令分成几个命令来执行，是因为你可能需要将迁移文件提交到你的版本控制系统中并跟随你的应用一起变化； 这样做不仅可以使开发变得更加简单，而且对其他开发者以及上线生产非常有用。

阅读[django-admin 的文档](http://python.usyiyi.cn/documents/django_182/ref/django-admin.html)来了解**manage.py** 工具能做的所有事情。

## 玩转API[¶](http://python.usyiyi.cn/documents/django_182/intro/tutorial01.html#playing-with-the-api)

现在，让我们进入Python的交互式shell，玩转这些Django提供给你的API。 使用如下命令来调用Python shell：

**$** python manage.py shell

我们使用上述命令而不是简单地键入“python”进入python环境，是因为**manage.py** 设置了**DJANGO\_SETTINGS\_MODULE** 环境变量，该环境变量告诉Django导入**mysite/settings.py**文件的路径。

绕开 manage.py

如果你不想使用**manage.py**，也没问题。只要设置[**DJANGO\_SETTINGS\_MODULE**](http://python.usyiyi.cn/documents/django_182/topics/settings.html#envvar-DJANGO_SETTINGS_MODULE) 环境变量为 **mysite.settings**，启动一个普通的Python shell，然后建立Django：

**>>> import** **django**

**>>>** django**.**setup()

如果以上命令引发了一个[**AttributeError**](https://docs.python.org/3/library/exceptions.html#AttributeError)，可能是你使用了一个和本教程不匹配的Django版本。 也许你应该切换到新一点或者旧一点的Django版本中去

您必须在与**manage.py**同一级目录中运行**python**，或确保该目录在Python Path中，以便**import mysite**能运行。

所有这些信息，请参见[django-admin 的文档](http://python.usyiyi.cn/documents/django_182/ref/django-admin.html)。

一旦你进入shell，请浏览[数据库API](http://python.usyiyi.cn/documents/django_182/topics/db/queries.html)：

**>>> from** **polls.models** **import** Question, Choice *# Import the model classes we just wrote.*

# No questions are in the system yet.

**>>>** Question**.**objects**.**all()

[]

# Create a new Question.

# Support for time zones is enabled in the default settings file, so

# Django expects a datetime with tzinfo for pub\_date. Use timezone.now()

# instead of datetime.datetime.now() and it will do the right thing.

**>>> from** **django.utils** **import** timezone

**>>>** q **=** Question(question\_text**=**"What's new?", pub\_date**=**timezone**.**now())

# Save the object into the database. You have to call save() explicitly.

**>>>** q**.**save()

# Now it has an ID. Note that this might say "1L" instead of "1", depending

# on which database you're using. That's no biggie; it just means your

# database backend prefers to return integers as Python long integer

# objects.

**>>>** q**.**id

1

# Access model field values via Python attributes.

**>>>** q**.**question\_text

"What's new?"

**>>>** q**.**pub\_date

datetime.datetime(2012, 2, 26, 13, 0, 0, 775217, tzinfo=<UTC>)

# Change values by changing the attributes, then calling save().

**>>>** q**.**question\_text **=** "What's up?"

**>>>** q**.**save()

# objects.all() displays all the questions in the database.

**>>>** Question**.**objects**.**all()

[<Question: Question object>]

先等一下。**<Question: Question object>**对于这个对象是一个完全没有意义的表示。 让我们来修复这个问题，编辑**Question**模型（在**polls/models.py**文件中）并添加一个[**\_\_str\_\_()**](http://python.usyiyi.cn/documents/django_182/ref/models/instances.html#django.db.models.Model.__str__)方法给**Question**和**Choice**：

polls/models.py

**from** **django.db** **import** models

**class** **Question**(models**.**Model):

*# ...*

**def** **\_\_str\_\_**(self): *# \_\_unicode\_\_ on Python 2*

**return** self**.**question\_text

**class** **Choice**(models**.**Model):

*# ...*

**def** **\_\_str\_\_**(self): *# \_\_unicode\_\_ on Python 2*

**return** self**.**choice\_text

给你的模型添加[**\_\_str\_\_()**](http://python.usyiyi.cn/documents/django_182/ref/models/instances.html#django.db.models.Model.__str__)方法很重要，不仅会使你自己在使用交互式命令行时看得更加方便，而且会在Django自动生成的管理界面中使用对象的这种表示。

**\_\_str\_\_** 还是 **\_\_unicode\_\_**?

对于Python 3来说，这很简单，只需使用[**\_\_str\_\_()**](http://python.usyiyi.cn/documents/django_182/ref/models/instances.html#django.db.models.Model.__str__)。

对于Python 2来说，你应该定义**[\_\_unicode\_\_()](http://python.usyiyi.cn/documents/django_182/ref/models/instances.html" \l "django.db.models.Model.__unicode__" \o "django.db.models.Model.__unicode__)**方法并返回**unicode** 值。Django 模型具有一个默认的**[\_\_str\_\_()](http://python.usyiyi.cn/documents/django_182/ref/models/instances.html" \l "django.db.models.Model.__str__" \o "django.db.models.Model.__str__)** 方法，它会调用**[\_\_unicode\_\_()](http://python.usyiyi.cn/documents/django_182/ref/models/instances.html" \l "django.db.models.Model.__unicode__" \o "django.db.models.Model.__unicode__)**并将结果转换为UTF-8 字节字符串。这意味着**unicode(p)**将返回一个Unicode 字符串，而**str(p)**将返回一个字节字符串，其字符以UTF-8编码。Python 的行为则相反：**对象**的**\_\_unicode\_\_**方法调用 **\_\_str\_\_**方法并将结果理解为ASCII 字节字符串。这个不同点可能会产生困惑。

如果以上这些令你费解的话，那就使用Python 3吧。

请注意这些都是普通的Python方法。 让我们演示一下如何添加一个自定义的方法：

polls/models.py

**import** **datetime**

**from** **django.db** **import** models

**from** **django.utils** **import** timezone

**class** **Question**(models**.**Model):

*# ...*

**def** **was\_published\_recently**(self):

**return** self**.**pub\_date **>=** timezone**.**now() **-** datetime**.**timedelta(days**=**1)

注意**import datetime** 和**from django.utils import timezone**分别引用Python 的标准[**datetime**](https://docs.python.org/3/library/datetime.html#module-datetime) 模块和Django **[django.utils.timezone](http://python.usyiyi.cn/documents/django_182/ref/utils.html" \l "module-django.utils.timezone" \o "django.utils.timezone: Timezone support.)**中时区相关的工具。如果你不了解Python中时区的处理方法，你可以在[时区支持的文档](http://python.usyiyi.cn/documents/django_182/topics/i18n/timezones.html) 中了解更多的知识。

保存这些改动，然后通过**python manage.py shell**再次打开一个新的Python 交互式shell：

**>>> from** **polls.models** **import** Question, Choice

# Make sure our \_\_str\_\_() addition worked.

**>>>** Question**.**objects**.**all()

[<Question: What's up?>]

# Django provides a rich database lookup API that's entirely driven by

# keyword arguments.

**>>>** Question**.**objects**.**filter(id**=**1)

[<Question: What's up?>]

**>>>** Question**.**objects**.**filter(question\_text\_\_startswith**=**'What')

[<Question: What's up?>]

# Get the question that was published this year.

**>>> from** **django.utils** **import** timezone

**>>>** current\_year **=** timezone**.**now()**.**year

**>>>** Question**.**objects**.**get(pub\_date\_\_year**=**current\_year)

<Question: What's up?>

# Request an ID that doesn't exist, this will raise an exception.

**>>>** Question**.**objects**.**get(id**=**2)

Traceback (most recent call last):

**...**

DoesNotExist: Question matching query does not exist.

# Lookup by a primary key is the most common case, so Django provides a

# shortcut for primary-key exact lookups.

# The following is identical to Question.objects.get(id=1).

**>>>** Question**.**objects**.**get(pk**=**1)

<Question: What's up?>

# Make sure our custom method worked.

**>>>** q **=** Question**.**objects**.**get(pk**=**1)

**>>>** q**.**was\_published\_recently()

True

# Give the Question a couple of Choices. The create call constructs a new

# Choice object, does the INSERT statement, adds the choice to the set

# of available choices and returns the new Choice object. Django creates

# a set to hold the "other side" of a ForeignKey relation

# (e.g. a question's choice) which can be accessed via the API.

**>>>** q **=** Question**.**objects**.**get(pk**=**1)

# Display any choices from the related object set -- none so far.

**>>>** q**.**choice\_set**.**all()

[]

# Create three choices.

**>>>** q**.**choice\_set**.**create(choice\_text**=**'Not much', votes**=**0)

<Choice: Not much>

**>>>** q**.**choice\_set**.**create(choice\_text**=**'The sky', votes**=**0)

<Choice: The sky>

**>>>** c **=** q**.**choice\_set**.**create(choice\_text**=**'Just hacking again', votes**=**0)

# Choice objects have API access to their related Question objects.

**>>>** c**.**question

<Question: What's up?>

# And vice versa: Question objects get access to Choice objects.

**>>>** q**.**choice\_set**.**all()

[<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]

**>>>** q**.**choice\_set**.**count()

3

# The API automatically follows relationships as far as you need.

# Use double underscores to separate relationships.

# This works as many levels deep as you want; there's no limit.

# Find all Choices for any question whose pub\_date is in this year

# (reusing the 'current\_year' variable we created above).

**>>>** Choice**.**objects**.**filter(question\_\_pub\_date\_\_year**=**current\_year)

[<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]

# Let's delete one of the choices. Use delete() for that.

**>>>** c **=** q**.**choice\_set**.**filter(choice\_text\_\_startswith**=**'Just hacking')

**>>>** c**.**delete()

For more information on model relations, see [Accessing related objects](http://python.usyiyi.cn/documents/django_182/ref/models/relations.html).更多关于如何在这些API中使用双下划线来执行字段查询的信息，请查看 [字段查询](http://python.usyiyi.cn/documents/django_182/topics/db/queries.html#field-lookups-intro)。更多关于数据库API的信息，请查看我们的 [数据库API参考](http://python.usyiyi.cn/documents/django_182/topics/db/queries.html)。

如果你适应了这些API，可以阅读[本教程的第2部分](http://python.usyiyi.cn/documents/django_182/intro/tutorial02.html)来让Django的自动管理界面工作起来。

### 目录

* [编写你的第一个Django应用，第1部分](http://python.usyiyi.cn/documents/django_182/intro/tutorial01.html)
  + [创建一个项目](http://python.usyiyi.cn/documents/django_182/intro/tutorial01.html#creating-a-project)
    - [数据库的建立](http://python.usyiyi.cn/documents/django_182/intro/tutorial01.html#database-setup)
    - [开发服务器](http://python.usyiyi.cn/documents/django_182/intro/tutorial01.html#the-development-server)
  + [创建模型](http://python.usyiyi.cn/documents/django_182/intro/tutorial01.html#creating-models)
  + [激活模型](http://python.usyiyi.cn/documents/django_182/intro/tutorial01.html#activating-models)
  + [玩转API](http://python.usyiyi.cn/documents/django_182/intro/tutorial01.html#playing-with-the-api)

### 浏览

* 上一页： [快速安装指南](http://python.usyiyi.cn/documents/django_182/intro/install.html)
* 下一页：[编写你的第一个Django应用，第2部分](http://python.usyiyi.cn/documents/django_182/intro/tutorial02.html)

### 你在这里：

* [Django 1.8.2 文档](http://python.usyiyi.cn/documents/django_182/index.html)
  + [开始](http://python.usyiyi.cn/documents/django_182/intro/index.html)
    - 编写你的第一个Django应用，第1部分

### 当前页

* [显示源](http://python.usyiyi.cn/documents/django_182/_sources/intro/tutorial01.txt)

### 快速搜索

窗体顶端

窗体底端

输入搜索字词或模块，类或函数名称。

### 最后更新：

2015年5月13日