
一个特殊的 Python 提示符

如果你曾经使用过 Python，你一定好奇，为什么我们运行 python manage.py shell 而不是 python。这两个命令都会启动交互解释器，但是 manage.py shell 命令有一个重要的不同： 在启动解释器之前，它告诉 Django 使用哪个设置文件。 Django 框架的大部分子系统，包括模板系统，都依赖于配置文件；如果 Django 不知道使用哪个配置文件，这些系统将不能工作。

如果你想知道，这里将向你解释它背后是如何工作的。 Django 搜索 DJANGO_SETTINGS_MODULE 环境变量，它被设置在 settings.py 中。例如，假设 mysite 在你的 Python 搜索路径中，那么 DJANGO_SETTINGS_MODULE 应该被设置为：’mysite.settings’。

当你运行命令：python manage.py shell，它将自动帮你处理 DJANGO_SETTINGS_MODULE。 在当前的这些示例中，我们鼓励你使用`` python manage.py shell``这个方法，这样可以免去你大费周章地去配置那些你不熟悉的环境变量。

随着你越来越熟悉 Django，你可能会偏向于废弃使用`` manage.py shell`` ，而是在你的配置文件.bash_profile 中手动添加 DJANGO_SETTINGS_MODULE 这个环境变量。

==========

project 和 app 的区别：它们的区别就是一个是配置另一个是 代码：

一个 project 包含很多个 Django app 以及对它们的配置。
技术上，project 的作用是提供配置文件，比方说哪里定义数据库连接信息，安装的 app 列表，TEMPLATE_DIRS，等等。
一个 app 是一套 Django 功能的集合，通常包括模型和视图，按 Python 的包结构的方式存在。
例如，Django 本身内建有一些 app，例如注释系统和自动管理界面。 app 的一个关键点是它们是很容易移植到其他 project 和被多个 project 复用。

```
python manage.py startapp books
```

---

django-admin.py 和 manage.py 是 Django 执行一些管理任务的命令行，他们两者执行的命令是一样的，区别的是当你安装 Django 的时候，最先安装了 django-admin.py，使用它生成一个 project 后，project 才会产生一个 manage.py。所以在一个 project 中把 django-admin.py 看成是全局的，即在生成项目之前，也可以使用它完成一些任 务；而 manage.py 只能在 project 生成后，才用得上

它。可能我们在开发环境的时候，习惯使用 manage.py；但他们的用法是一样的。

==========

视图：每个视图函数至少要有一个参数，通常被叫作 request。 这是一个触发这个视图、包含当前 Web 请求信息的对象，是类 django.http.HttpRequest 的一个实例，一个视图就是 Python 的一个函数。这个函数第一个参数的类型是 HttpRequest；它返回一个 HttpResponse 实例。为了使一个 Python 的函数成为一个 Django 可识别的视图，它必须满足这两个条件。

Django 处理请求流程：

当运行 python manage.py runserver 命令时，脚本将查找名为 setting.py 的文件，在 setting.py 中查找 ROOT_URLCONF 的配置，用来指向自动产生的 urls.py

`ROOT_URLCONF = 'mysite.urls'`，对应文件是 mysite/urls.py

当访问 特定 URL 时，Django 根据 ROOT_URLCONF 的设置装载 URLconf 。 然后按顺序逐个匹配 URLconf 里的 URLpatterns，直到找到一个匹配的。 当找到这个匹配 的 URLpatterns 就调用相关联的 view 函数，并把 HttpRequest 对象作为第一个参数

总结一下：

1. 进来的请求转入/hello/.
2. Django 通过在 ROOT_URLCONF 配置来决定根 URLconf.
3. Django 在 URLconf 中的所有 URL 模式中，查找第一个匹配/hello/的条目。
4. 如果找到匹配，将调用相应的视图函数
5. 视图函数返回一个 HttpResponse
6. Django 转换 HttpResponse 为一个适合的 HTTP response， 以 Web page 显示出来

根据以上流程可以发现，添加视图(即在 views.py 中添加视图函数)，需要两步，第一步在 views.py 中添加视图函数，第二步，在 urls.py 中 import 所添加的视图函数，并配置相应的 url 和视图的映射

```
from django.conf.urls.defaults import *
from mysite.views import hello, current_datetime

urlpatterns = patterns('',
    ('^hello/$', hello),
    ('^time/$', current_datetime),
)
```

模板

模板是一个文本，用于分离文档的表现形式和内容，模板定义了占位符以及各种用于规范文档该如何显示的各部分基本逻辑（模板标签）

创建模板，
```
from django.template import Template
t = Template('My name is {{ name }}.')
```

渲染，一旦你创建一个 Template 对象，你可以用 ***context*** 来传递数据给它。 一个 context 是一系列变量和它们值的集合。
```
>>> c = Context({'name': 'Stephane'})
```

模板标签，系统会显示在 {% if %} 和 {% endif %} 之间的任何内容

{% for %}，{% endfor %}

注释使用 {# #}

过滤器 http://www.lidongkui.com/django-template-filter-table

模板过滤器是在变量被显示前修改它的值的一个简单方法。 过滤器使用管道字符，如下所示：

{{ name|lower }}

显示的内容是变量 {{ name }} 被过滤器 lower 处理后的结果，它功能是转换文本为小写。过滤管道可以被* 套接* ，既是说，一个过滤器管道的输出又可以作为下一个管道的输入，如此下去

在视图中使用模板，先编写好相应的 html

```
<html><body>It is now {{ current_date }}.</body></html>
```

1. 在 setting.py 中设置 TEMPLATE_DAIRS

```
import os.path

TEMPLATE_DIRS = (
    os.path.join(os.path.dirname(__file__), 'templates').replace('\\','/'),
)
```

`` os.path.dirname(__file__)`` 将会获取自身所在的文件，即 settings.py 所在的目录

2. 经过逻辑处理获得模板变量，即{{}}中的内容

3. Render_to_response(),加载模板，渲染，response

```
from django.shortcuts import render_to_response
import datetime

def current_datetime(request):
    now = datetime.datetime.now()
    return render_to_response('current_datetime.html', {'current_date': now})
```

模板继承

第一步是定义 ***基础模板*** ， 该框架之后将由 ***子模板*** 所继承

基础模板：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    <h1>My helpful timestamp site</h1>
    {% block content %}{% endblock %}
    {% block footer %}
    <hr>
    <p>Thanks for visiting my site.</p>
    {% endblock %}
</body>
</html>
```

子模板(继承而来)

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

{% block title %}The current time{% endblock %}

{% block content %}
<p>It is now {{ current_date }}.</p>
{% endblock %}
```

所有的 {% block %} 标签告诉模板引擎，子模板可以重载这些部分。 每个{% block %}标签所要做的是告诉模板引擎，该模板下的这一块内容将有可能被子模板覆盖，若子模板没有覆盖，则用父模板的值，如果在模板中使用 {% extends %} ，必须保证其为模板中的第一个模板标记。 否则，模板继承将不起作用。

使用继承的一种常见方式是下面的三层法：

1.创建 base.html 模板，在其中定义站点的主要外观感受。 这些都是不常修改甚至从不修改的部分。

2.为网站的每个区域创建 base_SECTION.html 模板(例如, base_photos.html 和 base_forum.html)。这些模板对 base.html 进行拓展，并包含区域特定的风格与设计。

3.为每种类型的页面创建独立的模板，例如论坛页面或者图片库。 这些模板拓展相应的区域模板。这个方法可最大限度地重用代码，并使得向公共区域（如区域级的导航）添加内容成为一件轻松的工作

模型

ORM:对象关系映射（英语：Object Relational Mapping，简称ORM，或O/RM，或O/R mapping），是一种程序技术，用于实现面向对象编程语言里不同类型系统的数据之间的转换

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、如何验证有效性、包含哪些行为以及数据之间的关系等。

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。如何展示数据

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。，展示那些数据

步骤：

1. settings.py 中对 DATABASES 进行配置

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

2. python manege.py shell 命令，进入交互解释器

输入下面这些命令来测试你的数据库配置：

```
>>> from django.db import connection
>>> cursor = connection.cursor()
```

3.python manage.py startapp books,打开由上述命令创建的models.py

这一步是创建model

```
from django.db import models

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()

class Author(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=40)
    email = models.EmailField()

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

以上等同于SQL语言，

```
CREATE TABLE "books_publisher" (
    "id" serial NOT NULL PRIMARY KEY,
    "name" varchar(30) NOT NULL,
    "address" varchar(50) NOT NULL,
    "city" varchar(60) NOT NULL,
    "state_province" varchar(30) NOT NULL,
    "country" varchar(50) NOT NULL,
    "website" varchar(200) NOT NULL
);
```

Django操作数据库，不用SQL语言，直接用Python编码的方式

4.model创建完毕之后，在数据库中创建这些表

(1)settings.py,找到INSATLLED_APPS设置，将app添加到应用列表中

```
INSTALLED_APPS = (
    # 'django.contrib.auth',
    # 'django.contrib.contenttypes',
    # 'django.contrib.sessions',
    # 'django.contrib.sites',
    'mysite.books',
)
```

(2)检验 model 有效性`python manage.py validate`，若没有错误，则将在数据库中生成相应表

```
注意：Django 1.7.1及以上的版本需要用以下命令
python manage.py makemigrations
python manage.py migrate 或 python manage.py syncdb
```

python manage.py makemigrations 之后不会创建表，python manage.py migrate 才会创建表

基本的数据访问

```
from books.models import Publisher
```

增删改，需要更改数据表，提交时要调用 save 方法

插入：

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

```
p.save()
```

更新数据：

```
p.name = 'Apress Publishing'
p.save()
```

注意，并不是只更新修改过的那个字段，所有的字段都会被更新。 这个操作有可能引起竞态条件

```
Publisher.objects.filter(id=52).update(name='Apress Publishing')
```

与之等同的SQL语句变得更高效，并且不会引起竞态条件。

```
UPDATE books_publisher
SET name = 'Apress Publishing'
WHERE id = 52;
```

update()方法会返回一个整型数值，表示受影响的记录条数

```
>>> Publisher.objects.all()
[<Publisher: Apress>, <Publisher: O'Reilly>]
```

查询：这相当于这个SQL语句：

```
SELECT id, name, address, city, state_province, country, website
FROM books_publisher;
```

objects 属性。 它被称为管理器，管理器管理着所有针对数据包含、还有最重要的数据查询的表格级操作，所有的模型都自动拥有一个 objects 管理器；你可以在想要查找数据时使用它

以使用`` filter()`` 方法对数据进行过滤

filter() 根据关键字参数来转换成 WHERE SQL语句.

```
>>> Publisher.objects.filter(name='Apress')
```

```
SELECT id, name, address, city, state_province, country, website
FROM books_publisher
WHERE name = 'Apress';
```

```
Publisher.objects.filter(name__contains="press")
```

在 *name* 和 contains 之间有双下划线。和 Python 一样，Django 也使用双下划线来表明会进行一些魔术般的操作。这里，contains 部分会被 Django 翻译成 LIKE 语句：

```
SELECT id, name, address, city, state_province, country, website
FROM books_publisher
WHERE name LIKE '%press%';
```

filter()`` 函数返回一个记录集，这个记录集是一个列表。 相对列表来说，有些时候我们更需要获取单个的对象， `` get()`` 方法

排序：`Publisher.objects.order_by("address")`

删除：`Publisher.objects.all().delete()`