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。



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

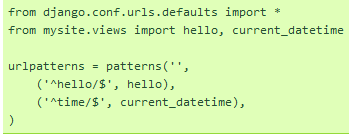
，对应文件是mysite/urls.py

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



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，

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。

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：

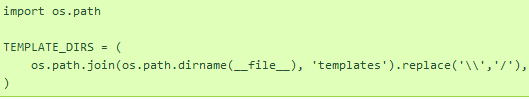
{{ name|lower }}

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

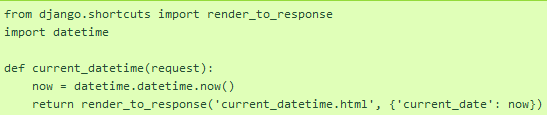


1. 在setting.py中设置TEMPLATE\_DAIRS



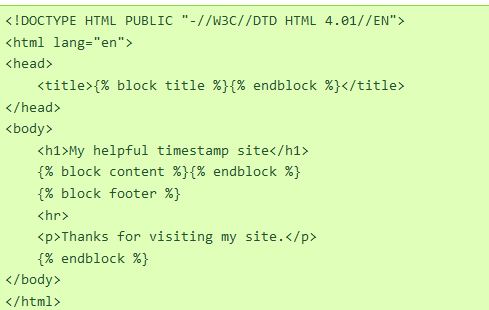
`` os.path.dirname(\_\_file\_\_)`` 将会获取自身所在的文件，即settings.py 所在的目录

1. 经过逻辑处理获得模板变量，即{{}}中的内容
2. Render\_to\_response(),加载模板，渲染，response

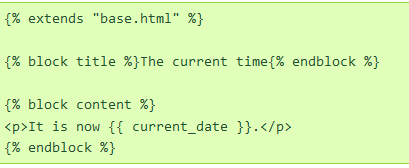


模板继承

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：

子模板(继承而来)



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](http://baike.baidu.com/view/1458.htm" \t "_blank)：**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：

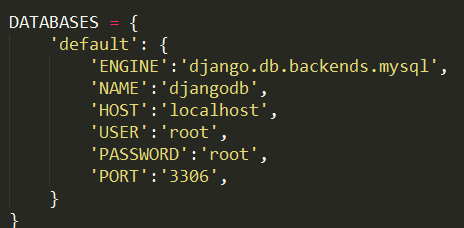
*M* 代表模型（Model），即数据存取层。 该层处理与数据相关的所有事务： 如何存取、如何验证有效性、包含哪些行为以及数据之间的关系等。

*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

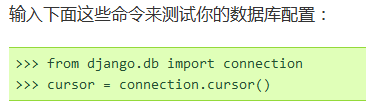
*V* 代表视图（View），即业务逻辑层。 该层包含存取模型及调取恰当模板的相关逻辑。 你可以把它看作模型与模板之间的桥梁。，展示那些数据

步骤：

1. settings.py中对DATABASES进行配置

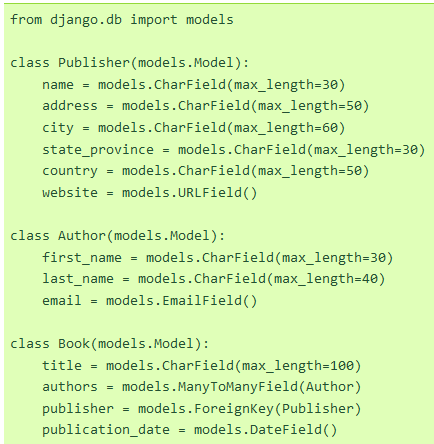


1. python manege.py shell命令，进入交互解释器



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



导入django.db.models

创建类，并定义相应属性

类对应的是数据库中的表

属性对应的是表中的字段

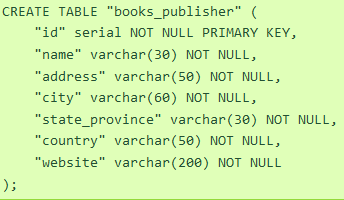
Charfield varchar

DateField 日期

URLField URL

ManyToManyField表示多对多关系

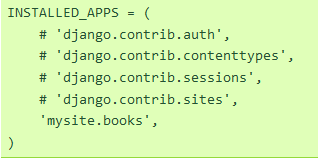
ForeignKey表示外键

以上等同于SQL语言，

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



(2)检验model有效性，若没有错误，则将在数据库中生成相应表



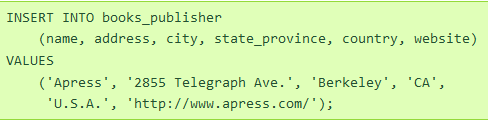
python manage.py makemigrations之后不会创建表，python manage.py migrate才会创建表

基本的数据访问



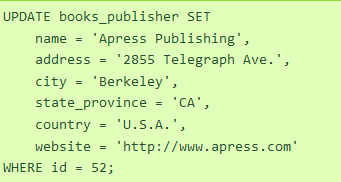
增删改，需要更改数据表，提交时要调用save方法

插入：



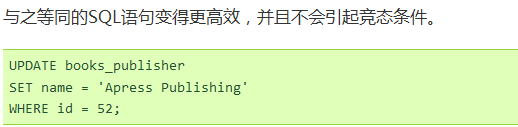


更新数据：

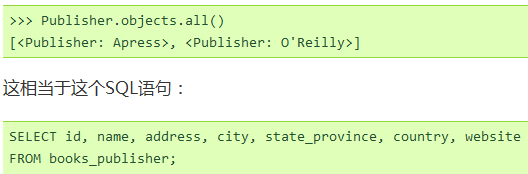


注意，并不是只更新修改过的那个字段，所有的字段都会被更新。 这个操作有可能引起竞态条件





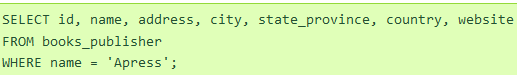
update()方法会返回一个整型数值，表示受影响的记录条数

查询：

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

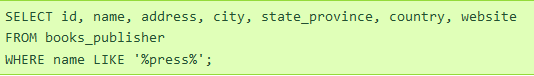
以使用`` filter()`` 方法对数据进行过滤







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：



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：

删除：