# 一.django 的安装:

解压之后,运行 sudo python setup.py install进行安装

安装结束后: 进入ipython交互式界面,引入django后(import django),使用 django.VERSION 命令 来验证安装是否完好

# 二.创建工程

创建工程:

django-admin startproject nba //nba为工程名称,创建工程的路径不能有中文

查看django-admin的命令: django-admin help,列出所有命令列表{查看django-admin 与manage.py 的所有命令,见附录一}

查看具体属性的使用方法: django-admin help functionname

nba中的文件:

① settings.py 项目的初始化配置文件

② urls.py url查找的根文件,

③ \_\_init\_\_.py 包中的文件,不用搭理

④ wsgi.py 一般不用,不用搭理

⑤ settings.pyc 字节码文件,不用搭理

⑥ \_\_init\_\_.pyc 字节码文件,不用搭理

在工程中,根目录为nba,要引入模块时,从根目录开始输入路径即可,父目录与子文件之间使用’.’号连接, eg:如果要引入manage.py,则直接引入就可以

# 三 开启测试服务器

## 3.1启动本地服务器:

./manage.py runserver 启动测试服务器, runserver 后可加端口号 默认为8000,只能本机来访问

## 3.2 启动局域网服务器

./manage.py runserver 0.0.0.0:8888 可以通过外部浏览器来访问

运行完毕后,当前文件夹会产生一个db.sqlite3 的数据库文件,不用搭理

# 四 更改settings设置

4.1 初步更改

更改语言: LANGUAGE\_CODE=’zh-Hans’

更改时区:TIME\_ZONE=’Asia/Shanghai’

# 五 创建应用

## 5.1创建应用:

./manage.py startapp myapp

## 5.2 配置文件

1) 创建urls.py: touch urls.py

2) 配置urls :

第一步: 在根目录下的urls.py中,将myapp中的urls.py添加进去:

① 引入include: from django.conf.urls import url,include

② 添加myapp中的urls.py:

urlpatterns += [url('^myapp/',include(‘myapp.urls’)),]

第二步:编辑myapp中的urls.py:

from django.conf.urls import \*

from views import \*

urlpatterns =[ url( '^model/$',model\_fun ), ]

第三步: 根据urls.py 编辑views.py,添加如下代码:

from django.http import HttpResponse

def model\_fun(request):

return HttpResponse('nihao') (返回给前端的具体方法,见附录二)

第四步: 在setttings中,进行注册:

在settings.py中的INSTALLED\_APPS中添加 ‘myapp’

第五步: 在views.py 中使用模板:

在settings.py中的TEMPLATES中的DIR中,,添加’templates’(需要在末尾加逗号)

## 5.3自定义 过滤器:

1创建templatetags文件夹,在该文件夹中,创建\_\_init\_\_.py(空文件即可),和extra.py(用于创建自定义过滤器,名称随意).

2.在templatetags文件夹中,创建 \_\_init\_\_.py(空文件即可,名字固定)和poll\_extra.py(名字随意)

3.在poll\_extra.py 中,写入:

from django import template

register = template.Library()

@register.filter(name = 'cut') #cut,名称为自定义的修饰器名称

def cut(value,arg):

return value.replace(arg,'##') #自定义的修饰器操作

4.在setting.py中的 INSTALLED\_APPS 中,添加先前创建的 myapp 文件夹

5.在views.py中的视图函数中添加相应的操作,然后通过urls.py进行捕捉即可

## 5.4配置数据库:

1.在mysql中.创建utf-8格式的数据库: create database Nbadb default character set utf8 collate utf8\_general\_ci;

2. 在settings中的DATEBASES中,修改为:

’ENGINE’:'django.db.backends.mysql',

‘NAME’:’nbadb’,

'USER':'root', # NOTE: 数据库用户名

"PASSWORD":'123', # NOTE: 数据库密码

'HOST':'', # NOTE: 主机IP

'PORT':'', # NOTE: 主机端口号

(各类数据库的是设置方法,见附录五)

配置完成数据库后,进行数据库配置检测:

./manage.py shell #进入项目中的ipython

from django.db import connection

Cursor = connection.cursor()

如果没有报错,则说明数据库配置成功

3.在models.py中添加如下代码:

class Publisher(models.Model):

name = models.CharField(max\_length = 30)

address = models.CharField(max\_length = 50)

country = models.CharField(max\_length = 30)

province\_state = models.CharField(max\_length = 30)

city = models.CharField(max\_length = 20)

website = models.UrlField(max\_length=30,blank = True,null = True)

class Author(models.Model):

first\_name = models.CharField(max\_length = 20)

last\_name = models.Charfield(max\_length = 20)

email = models.EmailField()

class Book(models.Model):

title = models.CharField(max\_length=100)

Publication\_date = models.DateField()

authors = models.ManyToManyField(Author)

publisher = models.ForeignKey(Publisher)

每一个类,就是数据库中的一张表格,类中的每一个变量,即为表中的每一个字段

1. 生成数据库日志文件并同步数据库:

./manage.py makemigretions

./manage.py migrate

此时,数据库已经构建完毕,可以直接在数据库中查看数据列表

1. 给数据库中增添数据:

count = Book.objects.title\_count('shoot-skill')

return render(request,'model.html',locals())

增加

Publisher.objects.create(name = 'jason.ltd',address = 'First Street',city = 'XIAN',state\_province = 'SHAANXI',country = 'China',website = 'www.baidu.com')

dic = {'first\_name':'Hao','last\_name':'Guo','email':'guohao@126.com'}

Author.objects.create(\*\*dic)

obj = Book(title = 'basketballskill',publication\_date = datetime.now(),publisher\_id = 2)

obj.save()

6 删除

Publisher.objects.filter(id = 2).delete()

7 改

Author.objects.filter(id=2).update(first\_name = 'miao',last\_name = 'zhang')

obj = Book.objects.get(id=3)

obj.title = 'shootskill'

obj.save()

8 查

book = Book.objects.all()

a = Publisher.objects.all().values('name')

b = Author.objects.get(id = 1)

c = Publisher.objects.filter(id\_\_gt=0).count()

return render(request,'model.html',locals())

# 6 站点管理:

## 6.1 创建超级用户

使用 ./manage.py createsuperuser 来创建超级用户, 根据提示输入用户名,密码邮箱即可.

然后在admin.py的文件中,添加一下代码:

from myapp.models import \*

admin.site.register(Publisher)

admin.site.register(Author,AuthorAdmin)

admin.site.register(Book,BookAdmin)

admin.site.register(User)

如果需要返回修改前的状态,可以使用 ./manage.py migrate myapp '0001' ,来返回.

其中 myapp为应用文件夹 '0001' 为前一个状态的数据库的编号,这两个需要根据实际情况来填写

manage命令:

1.生成model文件: ./manage.py inspectdb > test (test为导出的model文件.可以自行命名)

2 显示执行的sql语句 ./manage.py sqlmagrite myapp '0001'

3.进入当前项目的ipython ./manage.py shell

4 清除数据库数据 flush

5.进入当前项目的mysql: dbshell

可以使用 help 来查询这些命令

给数据库填写数据:

在应用文件夹里 新建 urls.py 和 templates(模板)文件夹

在模板文件夹里,新建模板文件

填写urls.py中的内容(仿照根目录中的urls.py)(from django.conf.urls import \* )

from django.conf.urls import \*

from myapp.views import \*

urlpatterns = patterns('',

(r'^models/$',model\_fun)

在views.py中,仿照根目录下的 views.py 来填写

from django.shortcuts import rander,render\_to\_response

....

然后,将根目录中的urls.py 设置成跳转到当前的urls文件:

在根目录中的urls.py 中 添加 (r'^',include('myapp.urls')) (需要在根urls.py的 django.conf.urls中引入include)

urlpatterns += patterns('',

('^app/',include('myapp.urls')),

)

数据操作:

增:

首先,在views中引入 models:

from myapp.models import \* (model为创建的html模板名称)

删:

改:

查:

admin: 数据库后台管理

1.创建超级用户:

./manage.py createsuperuser

将 models中的类注册到admin.py中:

先在admin.py中引入 models模块

然后注册:

admin.site.register(类名)

可以在models.py在进行相应操作

# :jdango相关知识:

MTV 开发模式

Django 的设计鼓励松耦合及对应用程序中不同部分的严格分割。把数据存取逻辑、业务逻辑和表现逻辑组合在一起的概念有时被称为软件架构的 Model-View-Controller

(MVC)模式。 在这个模式中:

Model 代表数据存取层,

View 代表的是系统中选择显示什么和怎么显示的部分,由视图和模板处理.

Controller 指的是系统中根据用户输入并视需要访问模型,以决定使用哪个视图的那部分。由django框架根据URLconf设置,对给定的URL调用适当的pyghon函数.

Controller由django框架自行处理,dango中更关注的是模型(model),模板(Template)和视图(Views),所以django也成为MTV框架:在MTV框架中:

M为模型(Model),即数据存储,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库之间的关系.

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,相当于MVC模式中的视图

V为视图(VIEWS),业务逻辑层,该层包含存取模型及调用恰当的模板的相关逻辑,可以看做是模型和模板之间的桥梁,相当于MVC模式中的控制器

# 附录一: django-admin命令与manage命令

|  |  |  |
| --- | --- | --- |
| django-admin 命令 | 意思 | manege 命令(带#为与admin相同命令) |
| django-admin help | <-----查看命令列表命令-----> | manage.py help |
| check | 检查django项目完整性 | # check |
| compilemessages | 编译语言文件 | # compilemessages |
| createcachetable | 在数据库中创建缓存表 | # createcachetable |
| dbshell | 进入djangodbshell | # dbshell |
| diffsettings | 查看自己的配置和django默认配置的不同之处 | # diffsettings |
| dumpdata | 导出数据 | # dumpdata |
| flush | 清空数据库 | # flush |
| inspectdb | 将数据库导出为模型文件 | # inspectdb |
| loaddata | 导入数据 | # loaddata |
| makemessages | 创建语言文件 | # makemessages |
| makemigrations | 生成数据库同步脚本 | # makemigrations |
| migrate | 同步数据库 | # migrate |
| runserver | 运行开发服务器 | # sendtestemail |
| sendtestemail | 发送测试邮件 | # shell |
| shell | 进入djangoshell | # showmigrations |
| showmigrations | 查看生成的数据库同步脚本 | # sqlflush |
| sqlflush | 查看生成清空数据库的脚本 | # sqlmigrate |
| sqlmigrate | 查看数据库同步的sql语句 | # sqlsequencereset |
| sqlsequencereset |  | # squashmigrations |
| squashmigrations | 数据库迁移 | # startapp |
| startapp | 创建一个app | # startproject |
| startproject | 创建一个项目 | # test |
| test |  | # testserver |
| testserver |  | # runserver |
|  | 修改密码 | changepassword |
|  | 创建超级管理员 | createsuperuser |
|  | 清除session | clearsessions |
|  | 静态文件复制 | collectstatic |
|  | 静态文件查找 | findstatic |

# 附录二:

使用views.py向前端返回信息的方法:

方法一:

t = loader.get\_template('template1.html')

html = t.render({'list':l,'dict':d,'class':c,'value':1,'message':message})

return HttpResponse(html)

方法二:

# return render\_to\_response('template1.html',{'list':l,'dict':d,'class':c,'value':1,'message':message})

方法三:

# return render\_to\_response('template1.html',locals())

方法四:

# return render(request,'template1.html',locals())

方法五:

return render(request,'template1.html',{'list':l,'dict':d,'class':c,'value':1,'message':message})

# 附录三:

传参方式:

1.位置传参 直接使用()

2.字典传参

# 附录四:

渲染:

1. 基本的模板标签和过滤器

标签

1) if/else

{% if today\_is\_weekend %}

<p>Welcome to the weekend!</p>

{% endif %}

{% else %} 标签是可选的

2) for

<ul>

{% for athlete in athlete\_list %}

<li>{{ athlete.name }}</li>

{% endfor %}

</ul>

3) forlooplast

forloop.last 是一个布尔值;在最后一次执行循环时被置为True。 一个常见的用法是在一系列的链接之间

放置管道符(|)

{% for link in links %}{{ link }}{% if not forloop.last %} | {% endif %}{% endfor %}

上面的模板可能会产生如下的结果:

Link1 | Link2 | Link3 | Link4

4) ifequal/ifnotequal

2. 注释

就像HTML或者Python,Django模板语言同样提供代码注释。 注释使用 {# #} :

{# This is a comment #}

如果要实现多行注释,可以使用`` {% comment %}`` 模板标签,就像这样:

{% comment %}

This is a

multi‐line comment.

{% endcomment %}

3.过滤器

就象本章前面提到的一样,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

# 附录五:

|  |  |  |
| --- | --- | --- |
| 数据库引擎设置 | | |
| 设置 | 数据库 | 所需配置器 |
| `` postgresql`` | PostgreSQL | psycopg 1.x 版,http://www.djangoproject.com/r/python- pgsql/1/ |
| postgresql\_psycopg2 | PostgreSQL | psycopg 2.x 版, http://www.djangoproject.com/r/python- pgsql/ 。 |
| mysql | MySQL | MySQLdb , http://www.djangoproject.com/r/python-mysql/. |
| sqlite3 | SQLite | sqlite3 SQLite 如果使用 Python 2.5+ 则不需要适配器。 否则就使用 pysqlite , |
| oracle | Oracle | cx\_Oracle , http://www.djangoproject.com/r/python-oracle/. |

常见错误及解决方法:

1.You haven’t set the DATABASE\_ENGINE settingyet:

不要以空字符串配置`` DATABASE\_ENGINE`` 的值。 表格 5-1 列出可用的值。

2.Environment variableDJANGO\_SETTINGS\_MODULE is undefined:

使用`` python manager.py shell`` 命令启动交互解释器,不要以`` python`` 命令直接启动交互解释器。

3.Error loading \_\_\_\_\_ module:No module named \_\_\_\_\_:

未安装合适的数据库适配器 (例如, psycopg 或 MySQLdb )。Django并不自带适配器,所以你得自己下载安装。

4.\_\_\_\_\_ isn’t an available database backend:

把DATABASE\_ENGINE 配置成前面提到的合法的数据库引擎。 也许是拼写错误?

5.database \_\_\_\_\_ does not exist:

设置`` DATABASE\_NAME`` 指向存在的数据库,或者先在数据库客户端中执行合适的``

CREATE DATABASE`` 语句创建数据库。

6.role \_\_\_\_\_ does not exist:

设置`` DATABASE\_USER`` 指向存在的用户,或者先在数据库客户端中执创建用户。

7.could not connect to server:

查看DATABASE\_HOST和DATABASE\_PORT是否已正确配置,并确认数据库服务器是否已正常运行。