# Django

# 目录结构

Cat——— **version2.X**

├── beetle **#功能目录**

│ ├── admin.py **#控制台目录，类似phpadmin**

│ ├── apps.py

│ ├── \_\_init\_\_.py

│ ├── migrations #迁移目录

│ ├── models.py **#模型目录**

│ ├── urls.py **#路由目录**

│ ├── tests.py #测试目录

│ └── views.py **#视图目录**

├── cat #系统配置目录

│ ├── \_\_init\_\_.py

│ ├── settings.py #**配置**

│ ├── urls.py #**路由**

│ └── wsgi.py #框架协议

├── temple **#模板文件夹**

├── db.sqlite3 #数据库

└── manage.py #主控

# 开发流程

1. 创建虚拟环境
2. 安装django
3. 创建项目
4. 创建应用

-------------------**以上查看 配置->命令** -------------

1. 在models.定义模型类
2. 迁移
3. 视图
4. 路由
5. 模板
6. Setting.py设置模板路径

View：接受请求，逻辑处理，调用数据，响应请求

Model：负责与数据库交互

Temple：模板

# MVT

## 配置

**cat/setting.py**

INSTALLED\_APPS 中添加模块

### 路由、模板

**cat/setting.py**

ROOT\_URLCONF 指定了url配置文件

'DIRS': [os.path.join(BASE\_DIR, 'templates')], #配置模板目录

**cat/urls.py**

**path需要导入**

**Path(“”,include(...)) #空字符串=^$，django2填写方法**

《《根据内容提示添加url解析，django2与之前版本不同

Url适用于django1.X

Path适用于django2.x》》

这里也可以将url分离开，分离出多个url文件,每个模块处理自己url文件

From django.urls import path,include

path('', include('beetle.urls'))

**Cat/beetle/urls.py**

from . import views

path('', views.index)

Path(“index/”, views.index)#这里是二级域名

### 迁移

1. 迁移目的主要是生成表的结构！！！
2. 前置条件：配置model.py类；
3. **如果以后要修改数据库结构，应直接修改表结构，再修改models中表的映射结构。不应再次迁移**

2、Python manage.py makemigrations

3、python manage.py migrate [指定model]

修改后

修改model中的字段后，

makemigrations

migrate

### 配置admin.py

**手册--->django--->高级--->admin站点**

1. 前置条件：model.py映射，迁移成功
2. 注册admin类
3. Admin.py上填写：

from django.contrib import admin

from models import BookInfo

admin.site.register(BookInfo)

1. 定义显示效果

from django.contrib import admin

from models import BookInfo

Class BookInfoAdmin(admin.Admin):

List\_display = [pk, ‘key’] #这里定义效果,显示字段

admin.site.register(BookInfo, BookInfoAdmin)#添加第2个参数

### 定义template

### 命令

创建：mkvirtualenv [虚拟环境名称]

删除：rmvirtualenv [虚拟环境名称]

进入：workon [虚拟环境名称]

退出：deactivate

python manage.py createsuperuser

创建项目：django-admin startproject 项目名 #这个命令需要ln 到系统目录

创建应用：python manage.py startapp ***pack\_name***

创建迁移：python manage.py makemigrations

生成迁移：python manage.py migrate

开启：Python manage.py runserver 0.0.0.0：8000

### 配置mysql数据库

**/setting.py**

DATABASES = {

'default': {

**'ENGINE': 'django.db.backends.mysql',**

'NAME': 'TakePicture',

'USER': 'root',

'PASSWORD': 'humingfei212697~',

'HOST': '127.0.0.1',

'PORT': '3306'

}

}

### （7）参数

通过view函数传递过来的 reuqest，使用request.META.get("header key")来获取

注意：

header key必须大写，前缀必须是"HTTP",后面如果连接符是横线“-”，要改成下划线“\_”。例如你的header的key为api\_auth，那在Django中应该使用request.META.get("HTTP\_API\_AUTH")来获取请求头的数据。

### ****（8）request对象的属性****

request.scheme

代表请求的方案,http或者https

request.path

请求的路径,比如请求127.0.0.1/org/list,那这个值就是/org/list

request.method

表示请求使用的http方法,GET或者POST请求

request.encoding

表示提交数据的编码方式

request.GET

<QueryDict: **{'aaa': ['123']**}> #加粗的是携带的参数

获取GET请求

request.POST

获取post的请求,比如前端提交的用户密码,可以通过request.POST.get()来获取

另外：如果使用 POST 上传文件的话，文件信息将包含在 FILES 属性中

request.COOKIES

包含所有的cookie

request.META

一个标准的Python 字典，包含所有的HTTP 首部。具体的头部信息取决于客户端和服务器，下面是一些示例：

* CONTENT\_LENGTH —— 请求的正文的长度（是一个字符串）。
* CONTENT\_TYPE —— 请求的正文的MIME 类型。
* HTTP\_ACCEPT —— 响应可接收的Content-Type。
* HTTP\_ACCEPT\_ENCODING —— 响应可接收的编码。
* HTTP\_ACCEPT\_LANGUAGE —— 响应可接收的语言。
* HTTP\_HOST —— 客服端发送的HTTP Host 头部。
* HTTP\_REFERER —— Referring 页面。
* HTTP\_USER\_AGENT —— 客户端的user-agent 字符串。
* QUERY\_STRING —— 单个字符串形式的查询字符串（未解析过的形式）。
* REMOTE\_ADDR—— 客户端的IP 地址。
* REMOTE\_HOST—— 客户端的主机名。
* REMOTE\_USER—— 服务器认证后的用户。
* REQUEST\_METHOD —— 一个字符串，例如"GET" 或"POST"。
* SERVER\_NAME —— 服务器的主机名。
* SERVER\_PORT —— 服务器的端口（是一个字符串）

request.user

一个 AUTH\_USER\_MODEL 类型的对象，表示当前登录的用户。

如果用户当前没有登录，user 将设置为 django.contrib.auth.models.AnonymousUser 的一个实例。你可以通过 is\_authenticated() 区分它们

把request传给前端的时候,前端可以通过 {%  if request.user.is\_authenticated  %}判断用户时候登录

request.session

一个既可读又**可写**的类似于字典的对象，表示当前的会话

### 修改后迁移

## view视图配置

类比TP模块中的contronal

**Cat/beetle/views.py**

**----使用模板必须配置模板路径-----**

**-------------------**

1. 编写视图（1,3,4）
2. 配置url（2）
3. 创建模板
4. 技巧（5,6,7,8）

**-------------------**

### 基础使用方法

Request对象由django自动创建（全局），使用函数第一个参数接收

Request不用定义，不用引入

Request可以被重命名

-----------

Httpresponse对象需要引入

方法1：

from django.http import HttpResponse

def index(resquest):

return HttpResponse("hello world")

方法2：

from django.shortcuts import render

from django.http import HttpResponse

def index(resquest):

return render(resquest, 'beetle/index.html', {})

方法3：

from django.template import RequestContext, loader

from django.http import HttpResponse

Def index(request):

Template = loader.get\_template(“beetle/index.html”)

Context = RequestContext(request, {‘list’:value})

Return HttpResponse(Template.render())

### 获取路由参数

一般用法

Path(‘admin/<ad>/’,view.index) #最后一个 / 必须加

Path(‘’,view.index) #url不加地址时的访问路径

Def index(request, ad): #使用<>获取参数，且必须与获取名一致

--------------转换器 <int:ad>---------------

str,匹配除了路径分隔符（/）之外的非空字符串，这是默认的形式

int,匹配正整数，包含0。

slug,匹配字母、数字以及横杠、下划线组成的字符串。

uuid,匹配格式化的uuid，如 075194d3-6885-417e-a8a8-6c931e272f00。

path,匹配任何非空字符串，包含了路径分隔符

----------------

反向解析：

Name字段用于反向解析

path("admin/", view.index, name="index")

path("", include("Ex.url"), namespace="Ex")

### GET

from django.http import HttpResponse

def get(request):

a = request.GET['value\_key'] #接收一个值

A = request.GET.get('value\_key',defind) #接收一个值,默认值defind

b = request.GET.getlist('key') #接受一key多个值

### POST

**Csrf如果没配置会报错（权限不够）**

**/setting.py -->注释csrf项**

def get(request):

a = request.POST['value\_key'] # 接收一个值

A = request.POST.get('value\_key',defind) #接收一个值,默认值defind

b = request.POST.getlist('key') # 接受一key多个值

### Cookies

---->游览器会自动生成带Cookie信息

---->夸域名不能共享cookie，由浏览器确定

def cookieTest(request):

response = HttpResponse() # 对象

cookie = ***request***.COOKIES # 请求cookie对象

if cookie.has\_key("t1"):

response.write(cookie['t1']) # 返回给游览器信息

# response.set\_cookie('t1','abc') #设置cookie，会自动写入客户端游览器

### 重定向

**手册-->django-->视图-->Response-->HttpResponseRedirect**

### （7）返回json

**手册-->django-->视图-->Response-->JsonResponse**

### Session

**手册-->django-->视图-->状态保持**

启动SESSION

/setting.py

项INSTALLED\_APPS列表中添加：

'django.contrib.sessions',

项MIDDLEWARE\_CLASSES列表中添加：

'django.contrib.sessions.middleware.SessionMiddleware',

状态保持：

#django默认保存至数据库，必须迁移

#可修改默认存放位置，redis

服务器保存session，就自动状态保持，要设置过期时间等

def session(request):

name = request.POST['name'] # 保存session

request.session['name'] = name # 保存session

return redirect('index/') # 重定向至其他页

def session(request):

name = request.session.get('name','默认参数') # 得到session,这个方法会返回Null

## model模型配置

连接数据库-->在cat/setting.py中配置

必须继承django.db.models

--------------------------------------

1. 根据表填写映射关系、表结构（1,2,3,4）
2. 数据迁移
3. 自定义管理器（5）
4. 6,7查询技巧

----------------------------------

### 方法详解

**1、all() **--> 查询所有，返回一个集合****

**2、filter(‘key’=value,’key’=value) **--> 返回符合条件的的集合****

**3、exclude([条件]) **--> 返回满足条件外的所有数据****

**4、order\_by() **--> 根据条件排序，并返回  （可以用'-'实现倒序）****

**5、values(‘key’) **--> 返回指定字段的值****

****------------一下返回单个数据****

****例：objects.filter().get()/exists()/count()/first()/last()/exists()****

**6、get(id=2) **--> 查询只能返回一个对象，多了少了都会引发DoesNotExist 异常，如果找到的是多条将抛出异常：MultipleObjectsReturned****

**7、exists() **-->  判断查询集中是否有数据，有为True****

**8、count() --> **返回当前查询的总条数****

**9、first() --> **返回第一个对象****

**10、last() --> **返回最后一个对象****

**11、exists() --> **判断查询集中是否有数据，如果有则返回True****

### （1）字段说明：

查看手册

id = models.IntegerField(primary\_key=True) #自增主键

### 实例

1、创建model类

from django.db import models

Class Ex(models.Model): #Ex表名 Model必须大写

#最长20字节，可以为空，

One = models.CharField(max\_length=20,null=True)

Two = models.IntertField(null = False)

2、使用管理器----->**查询**

可以更改默认查询集；**管理器可以后期添加代码**

**当自定义管理器后，默认管理器object就会没有**

**---------------------**

**管理器可以有多个**

from django.db import models

class MyManage(models.Manager):

def get\_info(self):

“””

这是一个自定一个的方法

”””

...

#查看手册，filter/count/all等方法

return self.filter(title\_\_icontains=keyword).count()

def get\_queryset(self):

“”“

使用Ex.objects.all()将调用我

”“”

return super(Ex, self).get\_queryset().filter(isDelete=False)

class Ex(models.Model):

#类属性

objects = models.Manager() # 自定义管理器时要设置，不然程序中默认的objects将不能使用

two = MyManage() # 自定义管理器

**在view中调用**：

from models import Ex

**Ex.objdects.all()** #调用默认的管理器方法

Ex.two.get\_info() #调用自定义的管理器方法

3、使用管理器--->**添加**

from django.db import models

class MyManage(models.Manager):

#函数名可以自定义

Def create(self,value\_one,value\_two):

B = Ex() #必须项获取模型类

B.value\_one = ‘one’ #模型中的属性

B.save() #保存方法-->添加到数据库

Return B #必须返回

class Ex(models.Model):

value\_one = models.CharField(max\_length=20,null=True)

### limit

4、模型查询

**手册-》模型-》模型查询**

**其他模块使用类**

------------------------

from beetle.models import Entry #从models包中导入类

querylist=Entry.objects.all() ***#这里查到的是一个对象集合，需要根据数据库中的字段进行读取***

Query[0,10] #切片

Query[11,20] #不会缓存，用子集不会缓存

使用缓存

querylist=Entry.objects.all() #直接保存

print([e.title for e in querylist])

print([e.title for e in querylist])

根据Id查询

Querylist.objects.get(pk=1) #pk代表主键

修改

book = BookInfo.objects.get(btitle='雪山飞狐')

book.bisDelete = 1

book.save()

### （3）关系相互访问

BookInfo():

Pass:

HeroInfo():

Book = models.ForeignKey(BookInfo)

BookInfo.heroinfo\_set #获取所有对应的HeroInfo

heroInfo.book\_id #获取书的Id

### （4）元选项

自定义表名；默认为 （应用名\_模型名）

from django.db import models

Class Ex(models.Model):

One = models.CharFile(max\_length=20)

Class Meta:

Db\_table=”name”

Ordering = [‘id’] #默认排序，物理磁盘会变大

### （7）F对象，Q对象

**F对象2个字段进行比较**

BookInfo.object.filter(bread\_\_gt=F(‘bcommet’)) #查询bcommet字段大于bread字段的值

对不同表字段进行关联

--------------

**Q对象体现：逻辑或关系**

**Ex.object.filter(Q(id\_\_lt=4) | Q(name\_\_contains=’a’) #id大于4,name 包含a**

### 外键

必须设置on\_delete

models.ForeignKey(Goods, verbose\_name=”名称”, on\_delete=models.CASCADE)

### 使用userModel

setting.py配置

<app.model.model类名>

AUTH\_USER\_MODEL = “USER.USERModel”

引入类

from django.contrib.auth import get\_user\_model

使用

User = get\_user\_model()

## temple模板配置

类比TP模块中的view

--------------

1. 配置模板（1）
2. 编写模板（2,5）
3. 其他功能（3,4,6）

------------

### 配置

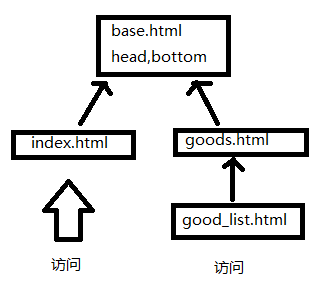
DIRS定义了一个目录列表，模板引擎按列表顺序搜索这些目录以查找模板源文件

APP\_DIRS告诉模板引擎是否应该在每个已安装的应用中查找模板

### 模板继承

**手册-->django-->模板-->定义模板**

**路径基于templates开始找模板位置**



### 过滤器

**手册-->django-->模板-->定义模板**

实现加减乘除等运算方式等其他运算，关键词 | 管道符

### 反向解析

在模板中根据路由自动生成url

#组成www.11.com/111/222/333

#使用参数，url必须获取参数

{% url ‘namespace:name’ 参数1 参数2 %}

### 模板语言

**手册-->django-->模板-->定义模板**

### HTML转义

django会自动转义部分html标签

不适用转义使用safe

### 富文本

### Csrf跨站请求伪造

只有post请求才有csrf验证

1. 启用csrf模块
2. 在form表单中添加：

<from action=”1.html”/>

{% csrf\_token %}

</from>

1. 或取消验证csrf

@csrf\_exempt

def csrf2(request):

原理：

1. 客户端第一次访问服务器，模板中{% csrf\_token %}标签会转义成input标签value是csrf cookie
2. 服务器view会要求客户端服务设置一个cookie

-----以上完成首页（第一次）请求--------

1. 当提交表单时会将客户端cookie与input标签的csrf值提交服务器
2. 服务器会对比两个值是否一致

# 安装流程

1. 应先安装sqlite数据库，因为django默认支持sqlite

Yum install sqlite3

1. 安装python3

ln -s /python3目录 /usr/bin/python 替换系统中默认的python2

1. 使用python3安装pip

如果使用pip -V 查看版本，确定是使用想要的版本python安装

替换默认pip位置:

Ln -s /pip位置 /usr/bin/pip

-----------------------------安装virtualenv-------------------------------

1. Pip install virtualenv
2. Pip install virtualenvwrapper 安装python虚拟环境
3. 修改~/.bashrc文件,**没有创建文件**

**添加:**

#virtualenvwrapper存放虚拟环境目录，可自定义目录

export WORKON\_HOME=/home/virtualenvwrapper

# 查找virtualenvwrapper.sh文件

# find / -name virtualenvwrapper.sh

source /目录/virtualenvwrapper.sh

1. source ~/.bashrc # 执行

---------------------------------------------------------------------------------

8、修复yum

Find / -name yum

修改yum文件开头的python注释为#! python2.7

9、Mkvirtualenv 环境名

10、Workon 环境名

11、Pip install django 在虚拟环境中安装django

12、Django-admin startproject 项目名 #创建项目

13、Python manage.py startapp pack\_name #创建包名

14、-----修改配置----填写代码----------

15、Python manage.py runserver 开启服务

# 其他

## 静态文件

**注意：**

**使用nginx时，请求静态文件找不到，需要设置nginx请求路由**

**这个配置是在开发过程中的配置方法，部署查看部署文件**

**--------------------------**

**更多内容从手册查找**

### 配置：

#路径路径与磁盘路径可以不一样

/setting.py

#url请求路径；决定请求位置，确保这个路径与html文件中的url路径一致

#不定义此项，django会出错

STATIC\_URL = ‘/static/’

#物理路径，决定文件在哪里

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, 'static'),

]

### 位置

Static目录与template目录同级（配置决定）

模板：从template位置开始

### 动态生成路径

**模板中添加**

{% load static from staticfiles %}

{% static ‘book/ad.png’ %}

## 中间件

/Setting.py MIDDLEWARE\_CLASSES中间件选项

插件系统；干预不同的执行流程6个方法

---面向切面编程--

## 上传图片

1、图片类需要pillow类支持

2、直接在setting.py添加MEDIA\_ROOT路径

**#这是为了确定上传文件保存位置**

3、Html需要上传插件

**<Html>**

<form method=”post” action=”upload/” enctype=”multipart/form-data”>

<input type=”file” name=”key” / >

</form>

1. view接受上传的文件

接收request对象中的files，将文件写入本地（就是文件读写过程）

From django.conf import settings

Import os

Def up(request):

Pic = request.FILES[‘key’] #**接收**文件

picName = os.path.join(settings.MEDIA\_ROOT,pic.name) #拼接路径

With open(picName, ‘w’) as pic:

For c in pic.chuncks():

Pic.write(c)

1. .Files[‘key’] 打印文件名

.content\_type #文件类型

.name#文件名

.size#文件大小（以字节为单位）

.read#文件内容

## 分页

1. 获取需要分页的所有数据
2. 使用paginator类对数据进行分类
3. 获取第n页的数据

from django.core.paginator import Paginator

def paginator(request, pageNum): # 请求对象，请求的第几页，设置默认值为1

# 判断页面是否为空,如果为空，则给1

if pageNum == "":

pageNum = 1

list = HeroInfo.object.all() # 使用数据查询，查找所有数据，惰性加载

pageObject = paginator(list, 5) # 获取数据，5条

page = pageObject.page(int(pageNum)) # 获取某一页的数据

context = {'page': page} # 当前分页的数据（内容）

return render(request, 'book/herolist.html', context)

# 1、路由要给一个参数

# 2、路由可以强制给一参数，只要统一口径，就可没有问题

# herolist.html

<html >

<!--这里是调用通过page调回pagintor对象，获取pagintor对象中的属性-->

{% for index in page.paginator.page\_range % }

<a href = "/herolist/{{ index }}" > {{index}} < /a >

< % endfor % >

</html >

## 5、Ajax

Jquery

## 6、全文索引

## 7、缓存

## 8、celery

## 9、跨域

**使django能接收跨域请求**

1、安装django-cors-headers

pip install django-cors-headers

2、修改settings.py

INSTALLED\_APPS = [

'corsheaders', #添加模块

]

MIDDLEWARE = [

......

'corsheaders.middleware.CorsMiddleware', #注册中间件

'django.middleware.common.CommonMiddleware', #要在这个模块上面

......

]

CORS\_ORIGIN\_ALLOW\_ALL = True #允许所有请求

CORS\_ALLOW\_CREDENTIALS = True

## 10、下载

1、在setting.py中设置下载目录

2、view：

from django.http import FileResponse

from django.conf import settings

def download(request, filename):

"""

下载功能

filename:下载的文件名

"""

downpath = settings.MEDIA\_ROOT + "/down/" + filename

#核心（读取文件，返回给浏览器）

file = open(downpath, 'rb')

response = FileResponse(file)

response['Content-Type'] = 'application/octet-stream'

response['Content-Disposition'] = 'attachment;filename="' + filename + '"

return response

3、html代码中从新打开一页

## 11、设置cookie/记住我

HttpReponse中有设置cookie方法

re =HttpReponseRedirect(“/usr/info”) #跳转的页面

re.set\_cookie(“name”, value) #设置cookie ，name的值为value

re.set\_cookie(“name”, “” ,max\_age=-1) #设置cookie立即过期

*request.session[‘name’]=name #服务器保存session，非必需*

return re #返回cookie，必须

#登陆方法--记录cookie

name=request.COOKIES.get(“name”, “”) #取得客户端cookie

context = {‘name’:name} #页面中使用{{name}}

return render(request, “login/index”,context)