**一、前言**

**MVC=MVT设计模式**

**(models=models,views=template,control=views)**

模型(数据库模型ORM)

视图(路由函数)

模版(html模版)

控制器(control)

学习的Django版本2.2(python3.5以上)

安装Django pip install django==2.2 -i <https://mirrors.aliyun.com/pypi/simple/>

**二、开始项目 第一个Django项目**

**创建应用**

python manage.py startapp App

**项目目录结构**

可以有多个App，这样便于管理

除此之外还有**static文件夹**



**2.1 settings.py文件解读 以及修改settings文件**

import os  
  
# 获取项目根目录  
# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)  
BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
  
  
# Quick-start development settings - unsuitable for production  
# See <https://docs.djangoproject.com/en/2.2/howto/deployment/checklist/>  
  
# session加密  
# SECURITY WARNING: keep the secret key used in production secret!  
SECRET\_KEY = '=#fo83a0z4n8i6s&$0#$78lg0ic-ayc2tmr@\_nfn&7k1\*==7ne'  
  
# 调试模式  
# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = True  
  
# 允许局域网主机访问  
ALLOWED\_HOSTS = ['\*']  
  
  
# Application definition  
# 安装应用  
INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 # 把自己的应用安装,不然无法使用  
 'App'  
]  
  
MIDDLEWARE = [  
 'django.middleware.security.SecurityMiddleware',  
 'django.contrib.sessions.middleware.SessionMiddleware',  
 'django.middleware.common.CommonMiddleware',

# 如果需要使用postman调试需要注释掉这个中间件  
 'django.middleware.csrf.CsrfViewMiddleware',  
 'django.contrib.auth.middleware.AuthenticationMiddleware',  
 'django.contrib.messages.middleware.MessageMiddleware',  
 'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]  
  
# 根路由  
ROOT\_URLCONF = 'study\_django.urls'  
  
# 模版配置 路径等  
TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 'DIRS': [os.path.join(BASE\_DIR, 'templates')]  
 ,  
 'APP\_DIRS': True,  
 'OPTIONS': {  
 'context\_processors': [  
 'django.template.context\_processors.debug',  
 'django.template.context\_processors.request',  
 'django.contrib.auth.context\_processors.auth',  
 'django.contrib.messages.context\_processors.messages',  
 ],  
 },  
 },  
]  
  
WSGI\_APPLICATION = 'study\_django.wsgi.application'  
  
  
# Database  
# <https://docs.djangoproject.com/en/2.2/ref/settings/#databases>  
# 数据库配置  
DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.sqlite3',  
 'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),  
 }  
}  
  
  
# Password validation  
# <https://docs.djangoproject.com/en/2.2/ref/settings/#auth-password-validators>  
  
AUTH\_PASSWORD\_VALIDATORS = [  
 {  
 'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',  
 },  
]  
  
  
# Internationalization  
# <https://docs.djangoproject.com/en/2.2/topics/i18n/>  
  
# 语言  
LANGUAGE\_CODE = 'zh-hans'  
# 时区  
TIME\_ZONE = 'Asia/Shanghai'  
  
USE\_I18N = True  
  
USE\_L10N = True  
  
# 不使用国际标准时间  
USE\_TZ = False  
  
  
# Static files (CSS, JavaScript, Images)  
# <https://docs.djangoproject.com/en/2.2/howto/static-files/>  
  
# 静态资源请求路径  
STATIC\_URL = '/static/'

**2.2 启动Django python manage.py runserver + 可指定ip和端口（0.0.0.0：9000）**

**2.3根路由**

在urls.py创建根路由

from django.contrib import admin  
from django.urls import path, include  
from App import views  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 # 创建自己的根路由  
 path('', views.index, name='index'),  
 # 关联子路由  
 path('user/', include("App.urls"))  
]

**2.4视图函数**

在App views.py中创建视图函数

from django.http import HttpResponse  
from django.shortcuts import render  
  
def index(request):  
 return HttpResponse("Hello world")

def index(request):  
 return HttpResponse("Hello world")

**2.5在App下创建urls.py 创建子路由**

# 添加子路由  
from django.urls import path  
from App import views  
  
# 路由列表  
urlpatterns = {  
 # 不能以/开头  
 path('home/', views.home, name='home')  
}

总结:可以创建多个App(python manage.py startapp App) 自动生成项目结构,创建的App中可以创建子路由urls.py,并且要在在根目录中关联子路由,在views中创建视图

**在 Django 中，每一个应用都是一个 Python （App）包，并且遵循着相同的约定。Django 自带一个工具，可以帮你生成应用的基础目录结构，这样你就能专心写代码，而不是创建目录了。**

**根路由可以关联视图,子路由也可以关联视图, 根路由关联子路由,**

**2.6模版渲染**

(大致和Flask相同)

在App-views.py中

from django.http import HttpResponse  
from django.shortcuts import render  
  
  
# Create your views here.  
def index(request):  
 data = {  
 "title": "主页",  
 "name": "Elric"  
 }  
 return render(request, "index.html", context=data)

**2.7模型**

App/models.py

创建模型类

from django.db import models  
  
  
# Create your models here.  
class User(models.Model):  
 username = models.CharField(max\_length=255)  
 password = models.CharField(max\_length=255)  
  
 class Meta:  
 # 指定表名  
 db\_table = 'user'

**生成数据库迁移文件 python manage.py makemigrations  
生成数据库表执行迁移 python manage.py migrate**

**尽量减少迁移 使用反向生成**

**反向生成模型 python manage.py inspectdb > models.py**

**python 测试数据库查询 python manage.py shell**

**其中使用 python manage.py 会显示当前所有指令**

**三、路由**

from django.urls import path, include  
from App import views

urlpatterns = [  
 path()  
]

**3.1 path的参数**

模式串:匹配用户请求路径的字符串

视图函数:匹配上用户指定请求路径后调用的视图函数名

kwargs:可选参数,需要额外传递的参数,是一个字典

name:给路由命名,在代码中可以使用name进行反向解析,和视图函数参数无关

例如:

path('user/', views.user, name='user')

如果访问[http://127.0.0.1:8000/HYPERLINK "http://127.0.0.1:8000/user"user](http://127.0.0.1:8000/user)

带了/的话会重定向到user/

如果user后面不带/, 访问user/会报错。

**3.2带参数的路由**

str如果没有指定参数类型默认是字符串类型,字符串参数可以匹配除了/和空字符外的其他字符串

int匹配0和正整数，视图函数得到一个int类型

slug匹配由数字 字母 - \_ 组成的字符串参数

path匹配任何非空字符串包括/(注意使用时要放在最后)

**路由是按照顺序匹配的,**

**可以多个路由对应一个视图函数,但是要注意参数的默认值**

**视图中的参数(id)要和路由中的参数(int:id)一样,路由名没有关系**

例如:

path('id=<int:id>', views.id, name='id')

def id(request, id):  
 return HttpResponse(str(id))

访问地址 <http://127.0.0.1:8000/id=1000>

显示1000

**3.3 re\_path**

正则规则匹配url

匹配qq号码

无命名组

re\_path(r'(\d{5,11})', views.qq, name='qq')

def qq(request, qq):  
 return HttpResponse(qq)

命名组 视图中的参数必需为qq

re\_path(r'(?P<qq>\d{5,11})', views.qq, name='qq')

四、视图

视图本质上是一个函数（类）。

**4.1HttpRequest**

HttpRequest是从web服务器传递过来的请求对象，**经过Django封装**产生的，封装了原始的Http请求。

视图的第一个参数必须是request，在django.http模块中定义了HttpRequest对象的API

使用HttpRequest对象的不同属性值,可以获取请求中多种信息

**属性**

content-type:请求的mime类型

GET:一个类似于字典的QueryDict对象,包含get请求方式的所有参数,也就会'?'后面的内容

POST:一个类似于字典的QueryDict对象,包含post请求方式的所有参数

COOKIES:字典,包含所有的cookie,键和值都是字符串

SESSION:一个类似于字典的对象,表示当前的会话,**只有当Django启用会话支持时才可以用**

path:字符串,其表示请求页面的完整路径,不包含域名

method:字符串,表示请求使用的HTTP方法

FILES:类似于字典的QueryDict对象,包含所有的上传文件

META:请求头信息

encoding:字符编码

scheme:协议

request.GET.get()

request.GET.getlist()

**4.2HttpResponse**

属性

content

charset

status\_code

content\_type

4.2.1调用模版返回

render是本质还是Httpresponse

renturn render()

4.2.2JsonResponse

可以返回json字符串,传入字典直接转换为json

如果参数不是字典,必须把safe设置为False

return JsonResponse({[1,2,3,4],safe=False)

4.3重定向 HttpresponseRedirect

def red(request):  
 #return HttpResponseRedirect("/user/home/")  
 return redirect('/')

4.4反向定位

根路由urls.py中

app\_name = "Hello"

def rev(request):  
  
 # 无参数  
 # return redirect(reverse('App:id', args=('23456')))  
  
 # 有参数  
 return redirect(reverse('App:id', kwargs={'id': 123456}))

**五、模版**

**一.模版位置**

settings.py

在应用中建立templates目录,好处不需要注册,不好的地方,有多个应用的时候不能复用(不推荐)

第二种是放在工程的目录下,好处是如果有多个应用,可以调用相同的页面,需要注册,如下

TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 'DIRS': [os.path.join(BASE\_DIR, 'templates')]  
 ,  
 'APP\_DIRS': True,  
 'OPTIONS': {  
 'context\_processors': [  
 'django.template.context\_processors.debug',  
 'django.template.context\_processors.request',  
 'django.contrib.auth.context\_processors.auth',  
 'django.contrib.messages.context\_processors.messages',  
 ],  
 },  
 },  
]

二、模版的渲染

2.1loader加载

好处是可以加载一次模版，然后进行多次渲染

2.2render

2.3变量

略

2.4过滤器

**自定义过滤器**

**在App下新建指定文件夹名称templatetags**

**在新建任意名称模块**

模块内:

from django import template  
  
register = template.Library()

@register.filter('hello')  
def hello(string):  
 return 'Hello ' + string + ' !'

**导入模块**

{% load time\_ago%}

**调用**

{{ user|hello }}

2.5内置标签

if 和 for等内置标签

2.6 csrf

csrf的意义在于给每一个表单都设置一个唯一的csrf值,并且cookie也存储一份,当提交表单过来的时候.判断cookie中的值,和csrf\_token中的值是否都为本网站生成的,如果验证通过则提交,否则403

settings文件中csrf的中间件是默认开启的,会生成csrf\_cookie

详情可以看这个博主的解答<https://www.cnblogs.com/renpingsheng/p/9756051.html>

在使用post,put,delete请求时 要在form表单中加上{% csrf\_token %}

四/模版继承

和jinjia2相同

**五/静态资源配置**

5.1创建static文件夹,通常放在根目录下

5.2需要在settings注册

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [

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

]

5.3在模版中使用静态资源

{% load static% }# 放到模版开头

<img src="/static/img/img.jpg" alt="">

<img src=" {% static 'img/img.jpg' alt="" %}">

**六/使用jinja2模版**

6.1pip install jinja2

6.2设置jinja2环境变量

在应用目录下,创建jinja2\_env.py配置文件

from django.contrib.staticfiles.storage import staticfiles\_storage  
from django.urls import reverse  
from jinja2 import Environment  
  
  
def environment(\*\*options):  
 env = Environment(\*\*options)  
 env.globals.update({  
 'static': staticfiles\_storage.url,  
 'url': reverse,  
 })  
 return env

6.3配置(settings.py)

**单独使用时**

**要记得注释掉django自带的admin,admin使用的是django模版,**

**相同的根路由urls.py下也要注释掉**

**同时可以再pycharm中settings,修改使用的模版**



INSTALLED\_APPS = [  
 # 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'App'  
]

添加引擎

TEMPLATES = [  
 {  
 'BACKEND':'django.template.backends.jinja2.Jinja2',

# 指定模版所在文件夹位置   
'DIRS': [os.path.join(BASE\_DIR, 'jinja2\_templates')],  
 'APP\_DIRS': True,  
 'OPTIONS': {  
 'context\_processors': [  
 'django.template.context\_processors.debug',  
 'django.template.context\_processors.request',  
 'django.contrib.auth.context\_processors.auth',  
 'django.contrib.messages.context\_processors.messages',  
 ],  
 # 配置环境,jinja的配置文件位置  
 'environment': 'App.jinja2\_env.environment',  
 },  
 },  
]

**七、数据库模型**

7.1配置数据库

DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.mysql',  
 'NAME': '链家二手房',  
 'HOST': '127.0.0.1',  
 'USER': 'root',  
 'PASSWORD': '123456',  
 'PORT': 3306  
 }  
}

7.2models.py定义模型类

from django.db import models

class User(models.Model):  
 uid = models.AutoField(primary\_key=True)  
 username = models.CharField(max\_length=30, unique=True)  
 password = models.CharField(max\_length=128)  
 regtime = models.DateTimeField(auto\_now\_add=True)  
 email = models.CharField(max\_length=255)  
  
 class Meta:  
 managed = False  
 db\_table = 'user'

7.3激活模型

**生成数据库迁移文件 python manage.py makemigrations  
生成数据库表执行迁移 python manage.py migrate**

**尽量减少迁移 使用反向生成**

**反向生成模型 python manage.py inspectdb > models.py**

**python 测试数据库查询 python manage.py shell**

**其中使用 python manage.py 会显示当前所有指令**

# 在这里遇到一个问题 使用python manage.py makemigrations生成迁移文件的时候  
# Meta中managed设置为False 导致无法生成对应表  
# 然而反向生成模型时managed=False(自动创建的)是为了防止对数据库误操作

7.4使用模型

# 增加数据  
# 方法一  
# user = User(username='Elric', password='123456')  
# user.save()  
# 方法二  
user = {'username': 'jojo', 'password': '123456'}  
User.objects.create(\*\*user)  
# 方法三 批量增加  
user = [User(username='Zero'), User(username='Two')]  
User.objects.bulk\_create(user)  
  
# 删除数据  
try:  
 user=User.objects.get(pk=2)  
 print(user, type(user))  
 if user:  
 user.delete()  
except Exception as e:  
 print(e)  
  
# 删除多条  
users = User.objects.filter(pk\_\_gte=3)  
users.delete()  
  
# 查  
find\_username = User.objects.get(pk=1)  
print(find\_username.username)  
  
# 改  
find\_username.password = '654321'  
find\_username.save()

小知识:逻辑删除(假删),添加字段用于判断是否被查询,如果用户删除,实际上没有被删除,只是查询不到了

7.4.1查询

过滤器:返回queryset (可遍历/切片(不支持-下标))

models.object.all() 返回所有数据

models.object.filter()返回符合条件的数据

models.object.exclude()返回不符合条件的数据

models.object.order\_by()对查询结果集进行排序

models.object.values()返回一个queryset,其中每个对象为一个字典

models.object.values\_list()和values()基本相同,但每个对象是一个元组

models.object.reverse()对排序的结果反转

models.object.only(字段)只显示指定字段

models.object.defer(字段)去除指定字段

.distinct()去重

对象:返回模型对象

models.object.get()返回**一个**满足条件的对象;如果没有会引发异常,如果找到多个也会引发异常

models.object.first()返回第一条

models.object.last()返回最后一条

models.object.earliest()根据指定字段返回最早增加的记录

models.object.latest(field)根据field字段返回最近增加记录

models.object.exists()判断是否存在 返回布尔值

models.object.count()返回查询集中对象的数目 返回int

**filter中操作符**

例如 filter(uid\_\_gt=9)

gt/gte 大于/大于等于

lt/lte 小于/小于等于

ne 不等于(not equal)

集合运算

in filter(uid\_\_in=[1,2,3])

filter(uid=1,name='x')

contains 包含某字符串

icontains (不区分大小写)包含某字符串

startswith 以..开头

istartswith (不区分大小写)以..开头

endswith 以..结尾

iendswith (不区分大小写)以..结尾

正则匹配

regex

filter(username\_regex=r'')

日期查询

filter(regtime\_\_year=2020)

**7.4统计查询**

from django.db.models import Max, Min, Avg, Sum, Count

聚合查询:

models.objects.aggregate(Max('uid'))

分组查询:

models.objects.values(分组字段).annotate(Count('uid'))

该方法可以使用filter过滤查询结果

Q和F对象:

(本身就是逻辑与不需要用到Q对象)

Q对象:构造逻辑或、逻辑非

例如 查询uid>30或 sex=1

filter(Q(uid\_gt=30)|Q(sex=1))

逻辑非

filter(~Q(sex=1)) 无法处理NULL值

F对象:

F对象;把sex看成models的一个列名

models.objects.filter.\(uid=F('sex'))

原生sql

models.objects.raw(sql)

**防止sql注入**(不能使用format拼接查询字符串)

sql = "SELECT \* FROM USER WHERE YEAR='{}'".format(year)

应该使用:

sql = "SELECT \* FROM USER WHERE YEAR=%s"

data =User.objects.raw(sql, [year])

自定义sql语句使用cursor(类似于flask)

from django.db import connection

with connection.cursor() as cursor:

curcor.excute('')

cursor.excute('')

row = cursor.fetchone()

7.5模型成员



重命名Manager管理器:

在models模型类里

class User(models.Model):  
 author = models.CharField(max\_length=255, blank=True, null=True)  
 title = models.CharField(max\_length=255)  
 release\_time = models.DateTimeField(auto\_now\_add=True)  
 url = models.CharField(max\_length=255)  
 like = models.IntegerField(blank=True, null=False, default=0)  
 # 自定义一个新的管理器,名字为new\_manage  
 new\_manage = Manager()

class Meta:  
 managed = False  
 db\_table = 'user'

使用data = User.new\_manage.all()

重命名是无意义的,关键在于**扩展Manager管理器功能,**

一个model可以使用多个管理器

这个类必须放在模型之前,要不然无法调用该管理器类

自定义管理器可以实现一些功能(假删之类)

# 自定义管理器  
class MyManager(Manager):  
  
 def get\_queryset(self):  
 data = super().get\_queryset().filter(like=0)  
 return data

**在需要自定义管理器中混用到自带的object管理器的时候,需要手动实例化一下**

**objects = Manager()**

**7.6 django orm模型之一对多关系（企业开发不推荐使用外键做一对多关系）**

**使用外键会影响查询速度**

class Book(models.Model):  
 book\_name = models.CharField(max\_length=200, blank=True, null=True)  
 # 外键  
 # 第一个参数:所参照的模型(如果不写参照模型,需要把参照的模型写在这个模型类前面)  
 # 第二个参数:on\_delete=models.DO\_NOTHING 说明详见图  
 # db\_column 表中字段名  
 # related\_name指的是从出版社(主表)查询图书(从表)引用的名字  
 pid = models.ForeignKey('Publisher', on\_delete=models.CASCADE, related\_name='books', db\_column='pid', null=True)  
  
 class Meta:  
 managed = False  
 db\_table = 'book'  
  
  
class Publisher(models.Model):  
 pname = models.CharField(max\_length=100, null=True)  
  
 class Meta:  
 managed = False  
 db\_table = 'publisher'

一是主表,多是从表

例如一个出版社对应多本出版的图书

外键参数on\_delete的选项:

级联删除：是指 当删除主表的内容时,对应从表的内容也会被删除,即删除出版社时,出版社对应的图书也会被删除。

保护模式：即是删除出版社时，会抛出异常，因为对应的图书表还有数据，如果要删除出版社表必须先删除图书表中对应的数据。



从表外键关联主键,crud都可以通过 对象.对象.属性 (pub.book.book\_name)

先查询: pub = Publisher.objects,get()

book = Book.objects.get()

再关联:(传递对象)

book.pid = pub





7.7一对一关系使用OneToOneField字段

(例如学生表和学生档案表)

class Author(models.Model):

name = models.CharField(max\_length=32)

age = models.IntegerField()

email = models.CharField(max\_length=32)

ad = models.OneToOneField(to="AuthorDetail", on\_delete=models.CASCADE)

class AuthorDetail(models.Model):

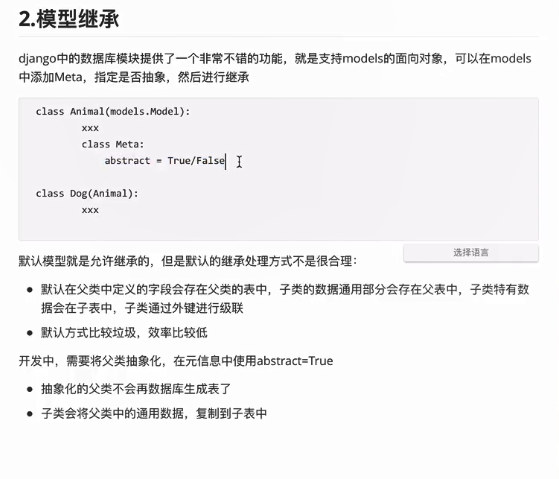
address = models.CharField(max\_length=32)

telephone = models.CharField(max\_length=11)

7.8多对多(顾客和商家和商品)

不建议使用多对多,可以加一个中间表变成一对多和一对多

7.9模型继承



1. Cookie Session

8.1

登录时设置cookies

response = redirect('/')  
# 设置cookie过期时间  
future = datetime.datetime.now() + datetime.timedelta(days=1)  
response.set\_cookie('username', username, expires=future)  
  
return response

视图函数中获取cookie

request.COOKIES.get('username')

登出时删除cookie

response = redirect('/')

response.delete\_cookie('username')

return response

8.2

Session设置在settings中

8.3提高网站安全 使用表单验证(主要判断字段长度)

App下新建form.py 用于定制表单验证

# 用户自定义表单  
from django import forms  
  
class RegisterForm(forms.Form):  
 # 用户名最少三个字符 required(html5输入框,必选项)  
 username = forms.CharField(min\_length=3, required=True, error\_messages={  
 'required': '用户名必须输入',  
 'min\_length': '用户名至少3个字符'  
 })

password = forms.CharField(min\_length=3, required=True, error\_messages={  
 'required': '密码名必须输入'  
 'min\_length' '密码至少为6位'  
 })

视图中使用

def register(request):  
 if request.method == 'POST':  
 form = RegisterForm(request.POST)  
 # 进行验证  
 if form.is\_valid():  
 # data是字典  
 data = form.cleaned\_data  
 print(data)  
 else:  
 # 验证不成功,把错误信息渲染到前端页面  
 return render(request, 'register.html', {'form': form})  
 return render(request, 'register.html')

Html页面

<form action="" method="post">  
 {% csrf\_token %}  
 用户名:<input type='text' name='username'>  
 {# 报错信息 #}  
 {{ form.username.errors}}  
 <br>  
 密码:<input type='password' name='password'> <br>  
 确认密码:<input type="password" name="password"><br>  
 <input type="submit" value="注册">

</form>