mkvirtualenv # 创建虚拟环境

rmvirtualenv # 删除虚拟环境

workon # 进入虚拟环境、查看所有虚拟环境

创建django虚拟环境

mkvirtualenv django\_py3\_1.11 -p python3

pip install # 安装依赖包

pip uninstall # 卸载依赖包

pip list # 查看已安装的依赖包

pip freeze # 冻结当前环境的依赖包

创建工程

django-admin startproject demo

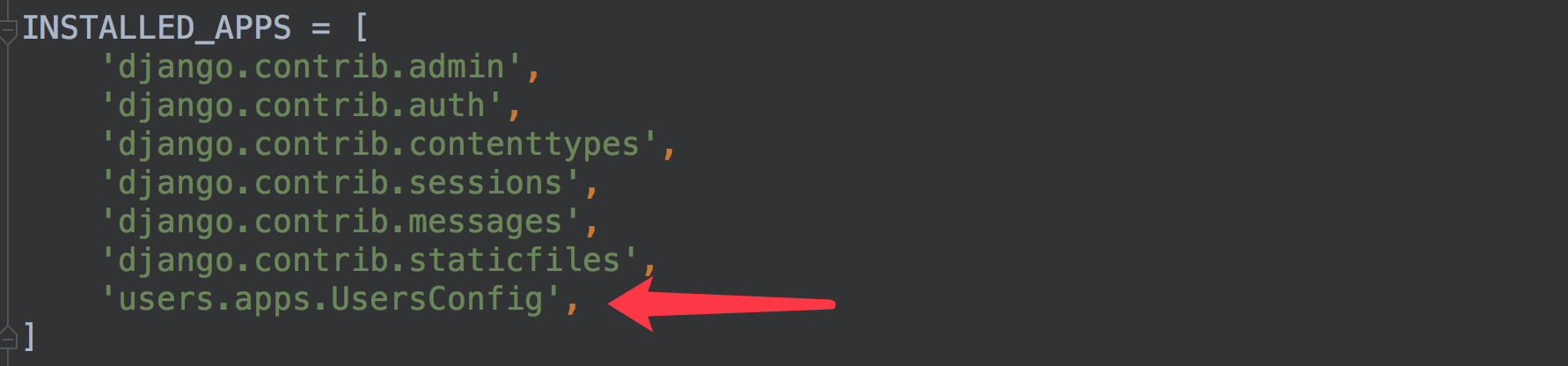
运行

python manage.py runserver ip:端口

创建子应用

python manage.py startapp 子应用名称

注册安装子应用setting中



创建视图

1.在子应用views编写视图

from django.http import HttpResponse

def index(request):

return HttpResponse("hello")

2.在子应用urls设置路由

from django.conf.urls import url

from . import views

# urlpatterns是被django自动识别的路由列表变量

urlpatterns = [

# url(路径, 视图)

url(r'^index/$', views.index),

]

#### 3.在总路由urls中添加子应用的路由数据

from django.conf.urls import url,include

urlpatterns = [

url(r'^admin/', admin.site.urls), # django默认包含的

# 添加

url(r'^users/', include('users.urls')),

]

配置静态文件(仅在debug模式下启用)

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [

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

]

reverse反解析

主路由添加namespace命名空间

rl(r'^users/', include('users.urls', namespace='users')),

普通路由加name

urlpatterns = [

url(r'^index/$', views.index, name='index'),

url(r'^say', views.say, name='say'),

]

解析路由

url = reverse('users:index') # 返回 /users/index/

app应用设置

class UsersConfig(AppConfig):

name = 'users'

verbose\_name = '用户管理'

request查询字符串

通过request.GET属性获取请求行路径中的参数

def qs(request):

#获取参数,多值参数

a = request.GET.get('a')

b = request.GET.get('b')

alist = request.GET.getlist('a')

获取表单类型数据

通过request.POST属性获取请求体中表单数据

a = request.POST.get('a')

b = request.POST.get('b')

alist = request.POST.getlist('a')

获取非表单数据

json\_str = request.body

json\_str = json\_str.decode() # python3.6 无需执行此步

req\_data = json.loads(json\_str)

请求头获取request.META

def get\_headers(request):

print(request.META['CONTENT\_TYPE'])

return HttpResponse('OK')

构造响应对象

respose=HttpResponse(content=响应体, content\_type=响应体数据类型, status=状态码)

respose.content：表示返回的内容。

respose.status\_code：返回的HTTP响应状态码。

respose.content\_type：指定返回数据的的MIME类型。

response['Head'] = 'value' # 自定义响应头Head, 值为value

构造json响应对象

JsonResponse({'city': 'beijing', 'subject': 'python'})

重定向

redirect('/index.html')

Cookie

设置cookie

HttpResponse.set\_cookie(cookie名, value=cookie值, max\_age=cookie有效期)

读取cookie

cookie1 = request.COOKIES.get('itcast1')

Session

SESSION\_ENGINE='django.contrib.sessions.backends.db' 默认存储

SESSION\_ENGINE='django.contrib.sessions.backends.cache' 本地缓存

SESSION\_ENGINE='django.contrib.sessions.backends.cached\_db' 混合存储

配置redis

……

session操作

1） 以键值对的格式写session。

request.session['键']=值

2）根据键读取值。

request.session.get('键',默认值)

3）清除所有session，在存储中删除值部分。

request.session.clear()

4）清除session数据，在存储中删除session的整条数据。

request.session.flush()

5）删除session中的指定键及值，在存储中只删除某个键及对应的值。

del request.session['键']

6）设置session的有效期

request.session.set\_expiry(value)

类视图导入

from django.views.generic import View

class RegisterView(View):

"""类视图：处理注册"""

def get(self, request):

"""处理GET请求，返回注册页面"""

return render(request, 'register.html')

def post(self, request):

"""处理POST请求，实现注册逻辑"""

return HttpResponse('这里实现注册逻辑')

路由

url(r'^register/$', views.RegisterView.as\_view(), name='register')

创建装饰器

def my\_decorator(func):

def wrapper(request, \*args, \*\*kwargs):

print('before')

response = func(request, \*args, \*\*kwargs)

print('after')

return response

return wrapper

在url中添加

url(r'^demo/$', my\_decorate(DemoView.as\_view()))

类内添加

全部添加

@method\_decorator(my\_decorator)

def dispatch(self, request, \*args, \*\*kwargs):

return super().dispatch(request , \*args, \*\*kwargs)

get添加

@method\_decorator(my\_decorator)

def get(self, request):

……

类上添加

全部添加

@method\_decorator(my\_decorator, name='dispatch')

get添加

@method\_decorator(my\_decorator, name='get')

构造扩展类

class MyDecoratorMixin(object):

@classmethod

def as\_view(cls, \*args, \*\*kwargs):

view = super().as\_view(\*args, \*\*kwargs)

view = my\_decorator(view)

return view

# 根据mro调用每个父类中的as\_view方法

class DemoView(MyDecoratorMixin,[其他扩展类…,] View):

def get(self, request):

……

中间件模板

新建middleware

def my\_middleware(get\_response):

# 此处编写的代码仅在Django第一次配置和初始化的时候执行一次。

def middleware(request):

# 此处编写的代码会在每个请求处理视图前被调用。

response = get\_response(request)

# 此处编写的代码会在每个请求处理视图之后被调用。

return response

return middleware

在settings.py 文件中添加注册中间件

MIDDLEWARE = [

……

'users.middleware.my\_middleware', # 添加中间件

]

执行顺序:处理前由上到下

处理后由下到上

数据库

在Django的工程同名子目录的\_\_init\_\_.py文件中添加如下语句

from pymysql import install\_as\_MySQLdb

install\_as\_MySQLdb()

修改配置文件

DATABASES = {

'default': {

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

'HOST': '127.0.0.1', # 数据库主机

'PORT': 3306, # 数据库端口

'USER': 'root', # 数据库用户名

'PASSWORD': 'mysql', # 数据库用户密码

'NAME': 'django\_demo' # 数据库名字

}

}

模型类定义

from django.db import models

#定义图书模型类BookInfo

class BookInfo(models.Model):

btitle = models.CharField(max\_length=20, verbose\_name='名称')

bpub\_date = models.DateField(verbose\_name='发布日期')

bread = models.IntegerField(default=0, verbose\_name='阅读量')

)

class Meta:

db\_table = 'tb\_books' # 指明数据库表名

verbose\_name = '图书' # 在admin站点中显示的名称

verbose\_name\_plural = verbose\_name # 显示的复数名称

def \_\_str\_\_(self):

"""定义每个数据对象的显示信息"""

return self.btitle

class HeroInfo(models.Model):

GENDER\_CHOICES = (

(0, 'male'),

(1, 'female')

)

…

hgender = models.SmallIntegerField(choices=GENDER\_CHOICES, default=0, verbose\_name='性别')

hbook = models.ForeignKey(BookInfo, on\_delete=models.CASCADE, verbose\_name='图书') # 外键

…

)

django会为表创建自动增长的主键列, 默认创建的主键列属性为id

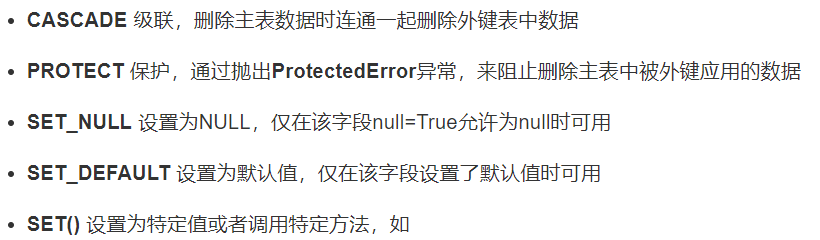
属性=models.字段类型(选项)



选项



外键



def get\_sentinel\_user():

return get\_user\_model().objects.get\_or\_create(username='deleted')[0]

class MyModel(models.Model):

user = models.ForeignKey(

settings.AUTH\_USER\_MODEL,

on\_delete=models.SET(get\_sentinel\_user),

)

数据库迁移

python manage.py makemigrations 生成迁移文件

python manage.py migrate

如果数据库迁移出错,删除migrations文件夹下的迁移文件和已建的数据库

shell命令

python manage.py shell

增删改查

添加数据

save:无返回值

hero = HeroInfo(……)

hero.save()

create:有返回值

HeroInfo.objects.create(……)

基本查询

get 查询单一结果，如果不存在会抛出模型类.DoesNotExist异常。

BookInfo.objects.get(id=3)

all 查询多个结果。

BookInfo.objects.all()

count 查询结果数量。

BookInfo.objects.count()

过滤查询

filter 过滤出多个结果

exclude 排除掉符合条件剩下的结果

get 过滤单一结果

1.相等 exact

BookInfo.objects.filter(id\_\_exact=1) 可简写为(id=1)

2.模糊查询

contains:包含

BookInfo.objects.filter(btitle\_\_contains='传')

startswith、endswith：以指定值开头或结尾

BookInfo.objects.filter(btitle\_\_endswith='部')

3.空查询 isnull

BookInfo.objects.filter(btitle\_\_isnull=False)

4.范围查询 in

BookInfo.objects.filter(id\_\_in=[1, 3, 5])

5. 比较查询

gt 大于 (greater then)

gte 大于等于 (greater then equal)

lt 小于 (less then)

lte 小于等于 (less then equal)

BookInfo.objects.filter(id\_\_gt=3)

不等于exclude()

BookInfo.objects.exclude(id=3)

6.日期查询

year、month、day、week\_day、hour、minute、second：对日期时间类型的属性进行运算

BookInfo.objects.filter(bpub\_date\_\_year=1980)

BookInfo.objects.filter(bpub\_date\_\_gt=date(1990, 1, 1))

F对象

from django.db.models import F

BookInfo.objects.filter(bread\_\_gt=F('bcomment') \* 2)

Q对象

from django.db.models import Q

与

BookInfo.objects.filter(bread\_\_gt=20,id\_\_lt=3)

BookInfo.objects.filter(Q(bread\_\_gt=20) & Q(pk\_\_lt=3))

或

BookInfo.objects.filter(Q(bread\_\_gt=20) | Q(pk\_\_lt=3))

非

BookInfo.objects.filter(~Q(pk=3))

BookInfo.objects.exclude(pk=3)

聚合函数

用aggregate()过滤器Avg()平均，Count()数量，Max()最大，Min()最小，Sum()求和

from django.db.models import Sum

BookInfo.objects.aggregate(Sum('bread'))

返回类型:字典

{'属性名\_\_聚合类小写':值}如:{'bread\_\_sum':3}

排序

BookInfo.objects.all().order\_by('bread') # 升序

BookInfo.objects.all().order\_by('-bread') # 降序

关联查询

一对多

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

b.heroinfo\_set.all()

多对一

h = HeroInfo.objects.get(id=1)

h.hbook

关联跨表查询

多查一

语法:关联模型类名小写\_\_属性名\_\_条件运算符=值

BookInfo.objects.filter(heroinfo\_\_hname='孙悟空')

BookInfo.objects.filter(heroinfo\_\_hcomment\_\_contains='八')

一查多

语法:一模型类关联属性名\_\_一模型类属性名\_\_条件运算符=值

HeroInfo.objects.filter(hbook\_\_btitle='天龙八部')

HeroInfo.objects.filter(hbook\_\_bread\_\_gt=30)

修改

1. save

hero = HeroInfo.objects.get(hname='猪八戒')

hero.hname = '猪悟能'

hero.save()

1. update

HeroInfo.objects.filter(hname='沙悟净').update(hname='沙僧')

删除

delete

1.模型对象

hero = HeroInfo.objects.get(id=13)

hero.delete()

2.模型类

HeroInfo.objects.filter(id=14).delete()

查询集QuerySet

all(),filter(),exclude(),order\_by()返回的都是查询集

特性

1. 惰性

遇到迭代、序列化、与if合用时,才去执行获得数据

1. 缓存

使用同一个查询集时,不会重复执行

对查询集切片

qs = BookInfo.objects.all()[0:2]

exists()判断查询集是否有数据

模型管理器

可以自定义模型管理器,重写其中的一些方法

1.定义一个类继承models.Manager

2.重写或新增方法

3.创建自定义模型管理器对象

模板

配置中修改DIRS

'DIRS': [os.path.join(BASE\_DIR, 'templates')], # 修改模板存放位置

模板渲染

from django.http import HttpResponse

from django.template import loader

def index(request):

# 1.获取模板 , loader.get\_template(模板文件在模板目录中的相对路径)

template=loader.get\_template('index.html')

context={'city': '北京'}

# 2.渲染模板, 模板对象.render(context=None, request=None),返回渲染后的html字符串,context 为字典变量，默认为None,request为请求对象，默认为None

return HttpResponse(template.render(context))

简写

render(request对象, 模板文件路径, 模板数据字典)

from django.shortcuts import render

def index(request):

context={'city': '北京'}

return render(request,'index.html',context)

模板语句

for

{% for item in 列表 %}

…

{{forloop.counter}}表示当前是第几次循环，从1开始

{%empty%} 列表为空或不存在时执行此逻辑

{% endfor %}

if条件

{% if ... %}

…

{% elif ... %}

…

{% else %}

…

{% endif %}

过滤器

变量|过滤器:参数

eg:safe转义 length长度 default默认值 date日期

单行注释 {#...#}

多行注释 {% comment %}...{% endcomment %}

模板继承

父模板

{% block 名称 %}

预留区域，可以编写默认内容，也可以没有默认内容

{% endblock 名称 %}

子模版

{% extends "父模板路径"%}

{% block 名称 %}

实际填充内容

{{ block.super }}用于获取父模板中block的内容

{% endblock 名称 %}

表单

模型类表单

class BookForm(forms.ModelForm):

class Meta:

model = BookInfo

fields = ('btitle', 'bpub\_date')

model 指明从属于哪个模型类

fields 指明向表单中添加模型类的哪个字段

Admin站点

创建超管python manage.py createsuperuser

在admin.py中注册模型类

from django.contrib import admin

from booktest.models import BookInfo,HeroInfo

admin.site.register(BookInfo)

admin.site.register(HeroInfo)

自定义管理模型类(2种方法)

1.class BookInfoAdmin(admin.ModelAdmin):

pass

admin.site.register(BookInfo,BookInfoAdmin) # 注册

2.@admin.register(BookInfo) # 装饰器

class BookInfoAdmin(admin.ModelAdmin):

pass

调整列表大小

list\_per\_page=100

操作选项上下的位置

actions\_on\_top = True

actions\_on\_bottom = True

列表中显示的列

list\_display=[模型字段1,模型字段2,...]

将方法作为列显示

def pub\_date(self):

return self.bpub\_date.strftime('%Y年%m月%d日')

pub\_date.short\_description = '发布日期' # 设置方法字段在admin中显示的标题

pub\_date.admin\_order\_field = 'bpub\_date' # 用于排序

admin中显示

list\_display = ['id','atitle','pub\_date']

过滤器

list\_filter = ['hbook', 'hgender']

搜索框

search\_fields = ['hname']

编辑页中显示的字段

fields = ['btitle', 'bpub\_date']

分组显示

fieldset=(

('组1标题',{'fields':('字段1','字段2')}),

('组2标题',{'fields':('字段3','字段4')}),

)

一对多的一中显示多条数据(2种)

1.块显示

class HeroInfoStackInline(admin.StackedInline):

model = HeroInfo # 要编辑的对象

extra = 1 # 附加编辑的数量

class BookInfoAdmin(admin.ModelAdmin):

...

inlines = [HeroInfoStackInline]

2.表格显示

class HeroInfoTabularInline(admin.TabularInline):

…

class BookInfoAdmin(admin.ModelAdmin):

...

inlines = [HeroInfoTabularInline]

站点信息

admin.site.site\_header = '网站页头'

admin.site.site\_title = '页面标题 '

admin.site.index\_title = '首页标语 '

上传图片

安装pillow pip install Pillow

建立图片字段

image = models.ImageField(upload\_to='booktest', verbose\_name='图片', null=True)

添加配置

MEDIA\_ROOT=os.path.join(BASE\_DIR,"static\_files/media")

显示字段

fields = ['btitle', 'bpub\_date', 'image']

重新迁移数据库

...