目录

**[Django 3](#_Toc28883)**

**[Django的MTV模型 4](#_Toc20873)**

**[创建app 4](#_Toc30618)**

**[路由 4](#_Toc29247)**

**[错误重定向 5](#_Toc11128)**

**[静态文件 5](#_Toc30022)**

**[response响应 5](#_Toc9375)**

**[Django 通用视图 6](#_Toc26811)**

**[跨站请求伪造保护 6](#_Toc31908)**

**[加载静态资源load+static 6](#_Toc23580)**

**[for循环变量 6](#_Toc30789)**

**[自定义过滤器 6](#_Toc3979)**

**[Models 7](#_Toc29739)**

[增删改查 7](#_Toc15847)

[模型关系 9](#_Toc24836)

[复合类型 10](#_Toc32155)

**[Manager 10](#_Toc2119)**

**[ajax用户名验证 11](#_Toc8480)**

**[发送邮件 12](#_Toc23915)**

**[session和cookie 12](#_Toc16310)**

**[中间件 13](#_Toc4165)**

**[缓存 14](#_Toc16758)**

[数据库缓存 14](#_Toc27063)

[redis缓存 14](#_Toc4133)

[文件系统缓存 15](#_Toc1849)

[本地内存缓存 15](#_Toc19224)

[缓存参数 15](#_Toc2969)

[站点缓存 15](#_Toc20330)

[视图缓存 15](#_Toc14059)

[模板片段缓存 16](#_Toc24253)

**[分页 16](#_Toc9914)**

**[Django后台管理 17](#_Toc16663)**

**[forms 17](#_Toc18680)**

**[验证码 18](#_Toc25245)**

**[方法一． 18](#_Toc8004)**

**[方法二． 18](#_Toc13373)**

**[celery异步 20](#_Toc7543)**

**[restful接口rest\_framework序列化 21](#_Toc3715)**

**[1.定制create：重写父类的create方法 24](#_Toc1443)**

[关系型模型序列化 25](#_Toc4136)

**[1.序列化 25](#_Toc29917)**

[最终 25](#_Toc32292)

[Session认证 权限设置 27](#_Toc12173)

[token认证 27](#_Toc6473)

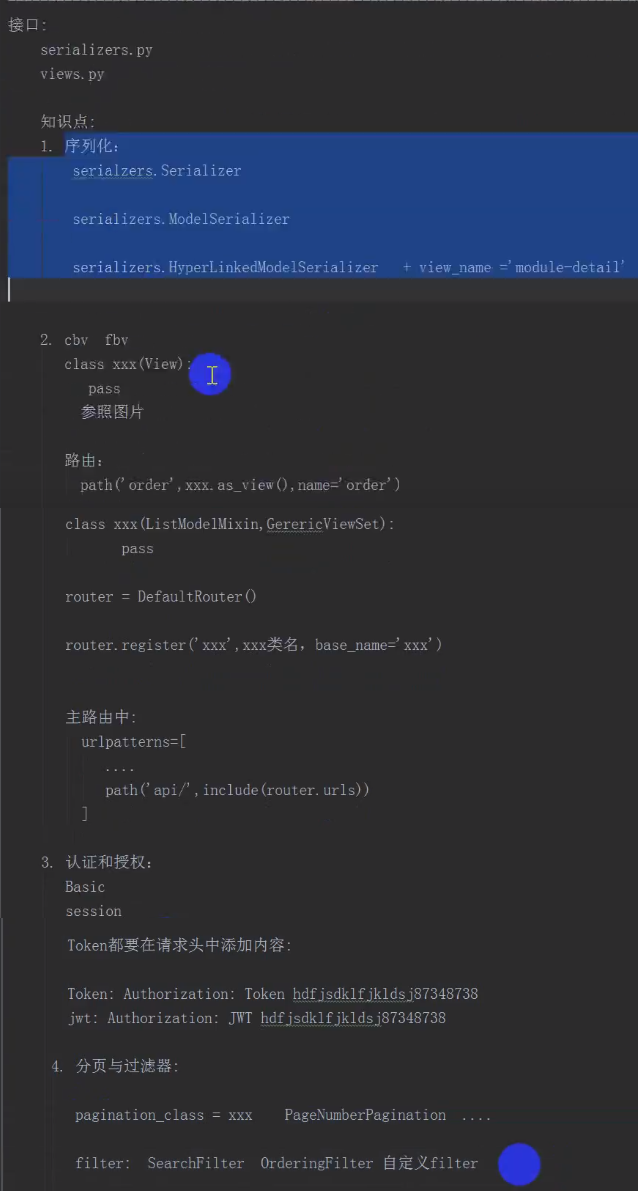
**[1 setting。py添加 27](#_Toc28958)**

[JWT认证 28](#_Toc15791)

[分页 29](#_Toc25151)

[配置 30](#_Toc628)

[筛选过滤 30](#_Toc12821)

**[ 32](#_Toc1189)**

**[跨域 33](#_Toc29194)**

**[4 setting配置 33](#_Toc7756)**

**[发送短信验证码模块 34](#_Toc4390)**

# Django

文档：https://docs.djangoproject.com/zh-hans/3.1/

虚拟环境(windown)：pip install virtualenvwrapper-win -i https://mirrors.aliyun.com/pypi/simple/

查看：workon 新建：mkvirtualenv 名称

进入：workon 名称 退出：deactivate

删除：rmvirtualenv 名称

新建：django-admin startproject project\_name

启动：python manage.py runserver 8000

设置templates 路径：

settings.py -> TEMPLATES -> DIRS：[BASE\_DIR / 'templates']

设置 static 路径:

settings.py ->末尾新增STATICFILES\_DIRS = [BASE\_DIR / 'statics']

mysql数据库配置：

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / 'db.sqlite3',

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

# 'NAME':'django01',

# 'HOST':'127.0.0.1',

# 'POST':3306,

# 'USER':'root',

# 'PASSWORD':'123'

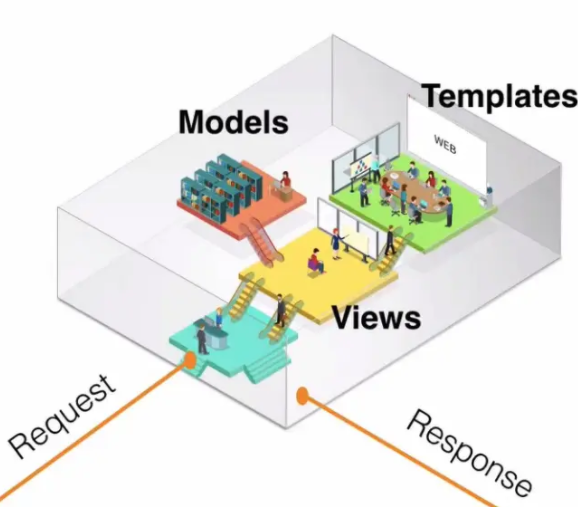
}

}

将app整合到一个文件里要在setting.py加：sys.path.insert(0,os.path.join(BASE\_DIR,'apps'))

其他同理

# Django的MTV模型

1.用户提交Request请求：把浏览器中url地址提交给Django服务器

2.Request首先到达路由urls.py 文件，在此文件中会去根据你事先在 urlpatterns 列表中定义好的url路径去确定指向哪个 Views文件views.py

3.views.py文件中的函数将会获取数据模型：Models和模板网页：Templates,在三者的共同渲染下，给客户端返回Response，呈现出完整网页。

# 创建app

创建：python manage.py startapp appname

# 路由

urlpatterns = [

path('articles/2003/', views.special\_case\_2003),

re\_path('articles/(?p<name>\w+)/', views.year\_archive),

path('articles/<int:year>/<str:month>/', views.month\_archive),

self.kwargs.get('pk', None)

path('community/', include('aggregator.urls')),]

# 错误重定向

主目录urls添加错误重定向视图

handler500='hello.views.err\_500'

# 静态文件

static

设置路径:settings.py ->末尾新增STATICFILES\_DIRS = [BASE\_DIR / 'statics']

media

**if** settings.DEBUG:

urlpatterns += [

re\_path(r'^media/(?P<path>.\*)$', serve, {

'document\_root': settings.MEDIA\_ROOT,

}),

]

setting中加MEDIA\_ROOT=os.path.join(BASE\_DIR,’media’)

MEDIA\_URL=’/media/’

# response响应

return HttpResponse('aaaaaaa')

return redirect(reverse(‘namespace:name’))

return render(request,'index.html')

# Django 通用视图

1. django.views.generic.base.View
2. django.views.generic.base.TemplateView
3. django.views.generic.detail.DetailView
4. django.views.generic.list.ListView

# 跨站请求伪造保护

<form method="post">{% csrf\_token %}

# 加载静态资源load+static

{% load staticfiles %}

<img src="{% static 'image/a1.jpg' %}" alt="">

# for循环变量

[https://docs.djangoproject.com/zh-hans/3.1/ref/templates/builtins/#std:templatetag-for](https://docs.djangoproject.com/zh-hans/3.1/ref/templates/builtins/" \l "std:templatetag-for)

# 自定义过滤器

自定义过滤器必须放在app中，且app必须在INSTALLED\_APPS中。在app下创建包templatetags（名字固定）。

新建templatetags包

建立py 文件

导入模块from django import template

注册过滤器对象：register = template.Library()

方法一register.filter("my\_template",my\_template)

方法二通过装饰器注册过滤器@register.filter

在模板中导入过滤器py文件{% load my\_filter %}

# Models

mysqlclient

mysql数据库配置：

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / 'db.sqlite3',

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

# 'NAME':'django01',

# 'HOST':'127.0.0.1',

# 'POST':3306,

# 'USER':'root',

# 'PASSWORD':'123'

}

}

定义：

class User(models.Model):  
 username = models.CharField(max\_length=12, unique=True, null=False,error\_messages={'unqiue': '用户名不能重复'})  
 password=models.CharField(max\_length=100,null=False)  
 phone=models.CharField(max\_length=11)  
 add\_time=models.DateTimeField(default=datetime.now)  
  
 def \_\_str\_\_(self):  
 return self.username  
  
 class Meta:  
 db\_table='user'

迁移与同步：python manage.py makemigrations

python manage.py migrate

元类：Meta

<https://docs.djangoproject.com/zh-hans/3.1/ref/models/options/>

增删改查

增，改：

user = User()  
user.username = username

user.save()

User.objects.create(username=username)

查：

User.objects.filter(username=username).first()

User.objects.all()

User.objects.get(username=username)

查询条件

等于：id\_\_exact=

像。。值：id\_\_iexact=

包含：contains 包含(不区分大小写)：icontains

在\*\*里：id\_\_in=[‘’,’’]

以\*\*开始：startswith istartswith 以\*\*结束： endswith iendswith

大于：gt大于等于gte：小于等于：lte小于：lt为空:isnull

date：日期

year：年 month ：月 day：日

hour/minute/second：时分秒 week/week\_day：星期

外键查询user\_\_username

删：

user=User.objects.get(pk=id)  
if user:  
 user.delete()

from django.shortcuts import render, redirect  
from django.http import HttpResponse  
from django.urls import reverse  
  
from user.models import User  
def index(request):  
 return render(request, 'base.html')  
  
def login(request, ):  
 if request.method == 'POST':  
 username = request.POST.get('username')  
 password = request.POST.get('password')  
 if username and password:  
 user = User.objects.filter(username=username).first()  
 if user and user.\_password == User.checkpwd(password):  
 return redirect(reverse('user:show'))  
 return render(request, 'user/login.html', context={'msg': '用户名或密码错误！'})  
 else:  
 return render(request, 'user/login.html', context={'msg': '用户名和密码不能为空！'})  
 return render(request, 'user/login.html')

def register(request):  
 if request.method == 'POST':  
 username = request.POST.get('username')  
 password = request.POST.get('password')  
 repassword = request.POST.get('repassword')  
 phone = request.POST.get('phone')  
 if username and password and repassword:  
 if password == repassword:  
 # 添加到数据库  
 user = User()  
 user.username = username  
 user.password = password  
 if phone:  
 user.phone = phone  
 user.save()  
 return render(request, 'base.html', context={'msg': '注册成功！,点击登录吧~'})  
 else:  
 return render(request, 'user/register.html', context={'msg': '两次密码不一致！'})  
 else:  
 return render(request, 'user/register.html', context={'msg': '用户名和密码不能为空！'})  
 return render(request, 'user/register.html')  
  
  
def show\_all(request):  
 users = User.objects.all()  
 return render(request, 'user/show\_all.html', context={'users': users})  
  
def delete(request):  
 id = request.GET.get('id')  
 # 根据id删除  
 user = User.objects.get(pk=id)  
 if user:  
 user.delete()  
 return redirect(reverse('user:show'))  
 else:  
 return render(request, 'user/show\_all.html', context={'msg': '删除失败！'})  
  
def update(request):  
 if request.method == 'GET':  
 id = request.GET.get('id')  
 user = User.objects.get(pk=id)  
 return render(request, 'user/update.html', context={'user': user})  
 else:  
 username = request.POST.get('username')  
 phone = request.POST.get('phone')  
 id = request.POST.get('id')  
 check = request.POST.get('check')  
 user = User.objects.get(pk=id)  
 user.username = username  
 user.phone = phone  
 user.is\_delete = 1 if check else 0  
 user.save()  
 return redirect(reverse('user:show'))

## 模型关系

related\_name= 反向引用

1对1：user=models.OneToOneField(to=表,on\_delete=[ ])

on\_delete:CASCADE:这就是默认的选项，级联删除.

PROTECT: 保护模式，如果采用该选项，删除的时候，会抛出ProtectedError错误。

SET\_NULL: 置空模式，外键字段被设置为空，定义该字段时要允许为空。

SET\_DEFAULT: 置默认值，设置为默认值，定义外键时要加上默认值。

SET(): 自定义一个值，该值当然只能是对应的实体了

分别添加，可一起查询

1对多：user=models.ForeignKey(to=表,on\_delete=)

1——》多 :1.多\_set.all()

多——》1：多。外键。属性

多1对多2 ：goods=models.ManyToManyField(to=表)

多1.外键。add（多2） 多2.多1\_set.add(多1)

多1.外键。remove（多2

## 复合类型

多个表订单表，用一个表放评价

class Order(models.Model):  
 *'''订单表'''* 。。。  
 #反向引用  
 comments=GenericRelation(to='Comment',related\_query\_name='order\_comments')  
  
class Comment(models.Model):  
 *'''评论表'''  
 。。。*

#外键绑定系统的ContentType表，提供对应的表名  
 content\_type=models.ForeignKey(ContentType,on\_delete=models.CASCADE)  
 #各表中的id  
 object\_id=models.PositiveIntegerField()  
 #用来查询，将表名和id绑定起来  
 content\_object=GenericForeignKey('content\_type','object\_id')

## 聚合函数

Book.objects.all().aggregate(Avg('price'))返回一个结果

Book.objects.annotate(Sum('price'))返回多个

## 事务

装饰器用法：整个视图所有数据库操作都看做一个事务，范围大，不够灵活。无法作用于类视图

from django.db import transaction

@transaction.atomic

with 语句用法：with transaction.atomic():

手动控制: （加装饰器）

transaction.set\_autocommit(False) #关闭自动提交

transaction.rollback() #回滚

transaction.commit() #提交

PS：三句话使用

from django.db import transaction

# 创建保存点save\_id = transaction.savepoint()

# 回滚到保存点transaction.savepoint\_rollback(save\_id)

# 提交事务操作transaction.savepoint\_commit(save\_id)

## 案例：统计报表

view.py

def get\_data\_count(start=None, end=None):  
 query = Q()  
 if start:  
 query = query & Q(created\_at\_\_gte=start)  
 if end:  
 query = query & Q(created\_at\_\_lte=start)  
 order\_list = Order.objects.filter(status=OrderStatus.PAID).filter(query)  
 user\_list = Profile.objects.select\_related('user').filter(user\_\_is\_active=True).filter(query)  
 return {  
 # 订单销售额  
 'order\_amount': order\_list.aggregate(amount=Sum('buy\_amount'))['amount'],  
 # 订单数  
 'order\_count': order\_list.count(),  
 # 新增用户数  
 'user\_add\_count': user\_list.count(),  
 # 下单用户数 distinct去重  
 'order\_user\_count': order\_list.values('user').distinct().count()  
 }  
  
  
def get\_latest\_order\_stats(days=7):  
 now\_time = now()  
 date\_array, amount\_array, count\_array = [], [], []  
 for i in range(days, 0, -1):  
 # 日期  
 calc\_time = now\_time + timedelta(hours=-i \* 24)  
 date\_array.append(f'{calc\_time.day}号')  
  
 # 订单金额 created\_at\_\_date，创建日期  
 queryset = Order.objects.filter(status=OrderStatus.PAID, created\_at\_\_date=calc\_time.date())  
 result = queryset.aggregate(amount=Sum('buy\_amount'))['amount']  
 amount\_array.append(result or 0)  
  
 # 订单数量  
 count\_array.append(queryset.count())  
 return {  
 'date': date\_array,  
 'amount': amount\_array,  
 'count': count\_array  
 }  
  
  
@login\_required(login\_url='/admin/login/') # 使用自带的登录验证  
def index(request):  
 # //数据统计  
 total\_stats = {  
 'sight\_count': Sight.objects.filter(is\_valid=True).count(),  
 'comment\_count': Comment.objects.filter(is\_valid=True).count(),  
 'user\_count': User.objects.filter(is\_active=True).count(),  
 'order\_count': Order.objects.filter(status=OrderStatus.PAID).count(),  
 }  
 # //实时数据  
 now\_time = now() # 获取现在时间  
 # 传入今天0点的时间  
 now\_stats = get\_data\_count(start=datetime(now\_time.year, now\_time.month, now\_time.day))  
 # 昨天数据  
 yesterday = now\_time + timedelta(hours=-24) # 获取昨天时间  
 yesterday\_stats = get\_data\_count(start=datetime(yesterday.year, yesterday.month, yesterday.day),  
 end=datetime(now\_time.year, now\_time.month, now\_time.day))  
 # 数据走势  
 latest\_stats = get\_latest\_order\_stats()  
 return render(request, 'master/index.html', locals())

## 数据

from django.db import models  
  
class CommonModel(models.Model):  
 updated\_at = models.DateTimeField('更新时间', auto\_now=True)  
 created\_at = models.DateTimeField('创建时间', auto\_now\_add=True)  
  
 class Meta:  
 abstract = True

class User(CommonModel):  
 *'''用户信息'''* USER\_STATUS = (('0', '删除'), ('1', '正常'))  
 username = models.CharField('用户名', max\_length=20, unique=True)  
 password = models.CharField('密码', max\_length=256)  
 nickname = models.CharField('昵称', max\_length=20, null=True)  
 avatar = models.ImageField('头像', null=True, upload\_to='avatar')  
 status = models.CharField('用户状态', max\_length=3, choices=USER\_STATUS, default=1)  
 is\_super = models.BooleanField('超级用户', default=0)  
 is\_delete = models.BooleanField('已删除', default=0)  
  
 class Meta:  
 db\_table = 'accounts\_user'  
  
 def \_\_str\_\_(self):  
 return self.username  
  
class UserProfile(CommonModel):  
 *'''用户详情'''* SEX = (('0', '女'), ('1', '男'), ('2', '隐藏'))  
 user = models.OneToOneField(verbose\_name='关联用户', to='User', on\_delete=models.CASCADE, related\_name='profile')  
 username = models.CharField('用户名', max\_length=20, unique=True)  
 real\_name = models.CharField('真实姓名', max\_length=20, null=True)  
 sex = models.CharField('性别', max\_length=2, choices=SEX, default=2)  
 maxim = models.CharField('用户格言', max\_length=128, null=True)  
 address = models.CharField('用户地址', max\_length=128, null=True)  
  
 class Meta:  
 db\_table = 'accounts\_user\_profile'  
  
 def \_\_str\_\_(self):  
 return self.username  
  
class LoginHistory(models.Model):  
 *'''用户登录历史'''* user = models.ForeignKey(verbose\_name='关联用户', to='User', on\_delete=models.CASCADE, related\_name='login\_history')  
 username = models.CharField('用户名', max\_length=20, null=True)  
 login\_type = models.CharField('登录平台', max\_length=128, null=True)  
 ip = models.CharField('IP', max\_length=32, null=True)  
 ua = models.CharField('登录来源', max\_length=128, null=True)  
 created\_at = models.DateTimeField('创建时间', auto\_now\_add=True)  
  
 class Meta:  
 db\_table = 'accounts\_login\_history'  
 ordering = ['-created\_at']  
  
 def \_\_str\_\_(self):  
 return self.username

# Manager

重写可以方便查询

class User(models.Model):

objects=UserManager()

。。。

class UserManager(models.Manager):  
 #重写查询方法  
 def get\_queryser(self):  
 return super(UserManager, self).get\_queryser().filter(is\_delete=0)

# ajax用户名验证

register.js

var username=$('#username').val();  
$.getJSON('/user/check\_user',{username:username},function(data){  
 if(data.status=='success'){  
 err\_name=true;  
 };  
 $('#username').next().html(data.msg);  
 $('#username').next().show();  
});

$('#reg\_form').submit(function(){  
 check\_username();  
 check\_password();  
 if(err\_name && err\_pwd){  
 return true;  
 }else{  
 return false;  
 }  
});

views

def check\_user(request):  
 username=request.GET.get('username')  
 user=User.objects.filter(username=username).first()  
 if user:  
 return JsonResponse({'status':'fail','msg':'用户名已存在!'})  
 else:  
 return JsonResponse({'status':'success','msg': '用户名可以使用!'})

js密码验证

function check\_password(){  
 var pwd1=$('#password').val()  
 var pwd2=$('#repassword').val()  
 if (~pwd1){  
 $('#repassword').next().html('密码不能为空！');  
 $('#repassword').next().show();  
 err\_pwd=false;  
 }  
 else if (pwd1 != pwd2){  
 $('#repassword').next().html('两次密码不一样！');  
 $('#repassword').next().show();  
 err\_pwd=false;  
 }  
 else{  
 err\_pwd=true  
 $('#repassword').next().hide();  
 }  
};

# 发送邮件

在settings.py设置

# 邮箱设置  
EMAIL\_HOST = 'smtp.qq.com'  
EMAIL\_PORT = 465 # 465 or 587  
EMAIL\_HOST\_USER = '804436125@qq.com'  
EMAIL\_HOST\_PASSWORD = 'zusvpklituttbbfa'  
EMAIL\_USE\_SSL = True  
EMAIL\_USE\_TLS = False

send\_mail(subject=标题, message=内容, from\_email=settings.EMAIL\_HOST\_USER,recipient\_list=[email,],html\_message=转义的内容

)

send\_mail(subject, message, from\_email, recipient\_list,  
 fail\_silently=False, auth\_user=None, auth\_password=None,  
 connection=None, html\_message=None):

# session和cookie

cookie：

设置：response.set\_cookie(‘key’, value, max\_age=)

获取：request.COOKIES.get(‘key’)

删除：response.delete\_cookie(‘key’)

session：

设置：request.session[‘key’]=value

request.session.set\_expiry(0) ——》过期时间，0关闭过期

获取：request.session.get('key')

删除：request.session.flush() ——》session和cookie的

request.session.delete() ——》当前会话的

request.session.pop(‘key’) ——》某一个

# 中间件

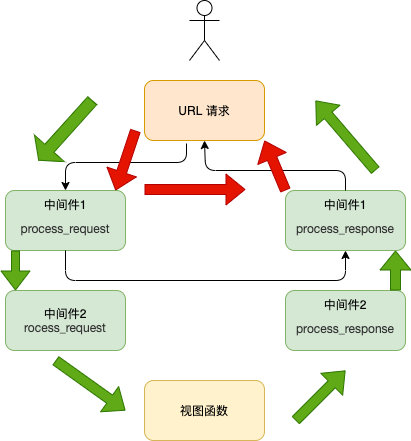
中间件组件配置在 settings.py 文件的 MIDDLEWARE 选项列表中。

步骤：

新建一个 py ，并导入 MiddlewareMixin

自定义中间件类，要继承父类 MiddlewareMixin如：class MD1(MiddlewareMixin)

在 settings.py 中的 MIDDLEWARE 里注册自定义中间件

中间件可以定义五个方法，分别是：

process\_request(self,request)

process\_view(self, request, view\_func, view\_args, view\_kwargs)

process\_template\_response(self,request,response)

process\_response(self, request, response)

process\_exception(self, request, exception)

**process\_request(self,request)**

在视图函数之前执行，和视图函数中的 request 是一样的。

返回 None 的话，按正常流程继续走

返回 HttpResponse 对象，Django 将不再前进，以该中间件为起点，倒序执行视 图函数之后的中间件方法。

**process\_view(self, request, view\_func, view\_args, view\_kwargs)**

Django会在调用视图函数之前调用process\_view方法。

request是HttpRequest对象。

view\_func是即将使用的视图函数。（obj）

view\_args是将传递给视图的参数的列表.

view\_kwargs是将传递给视图的关键字参数的字典。

**process\_template\_response(self,request,response)**

是在视图函数后执行，前提条件，视图函数返回的对象有一个render()方法

**process\_response(self, request, response)**

是在视图函数之后执行的，倒序执行！

**process\_exception(self, request, exception)**

一个HttpRequest对象，一个exception是视图函数异常产生的Exception对象。

在视图函数中出现异常才执行，倒序执行！

# 缓存

## 数据库缓存

settings.py中设置

CACHES = {

'default': {

'BACKEND': 'django.core.cache.backends.db.DatabaseCache',

'LOCATION': 'my\_cache\_table',

}}

输入：python manage.py createcachetable创建表

设置：cache.set(key,value)

获取：cache.get(key)

删除：cache.delete(key)

## redis缓存

安装：pip install django-redis

设置

CACHES = {

'default': {

'BACKEND': 'django\_redis.cache.RedisCache',

'LOCATION': 'redis://127.0.0.1:6379/1', #’redis://:密码@redis数据库地址:6379/1号数据库’

}}

开启redis：cmd进入redis文件夹(C:\Program Files\Redis)，redis-server redis.windows.conf

redis客户端：redis-cli

## 文件系统缓存

CACHES = {

'default': {

'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',

'LOCATION': 'c:/foo/bar',

}}

## 本地内存缓存

CACHES = {

'default': {

'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',

'LOCATION': 'unique-snowflake',

}}

## 缓存参数

TIMEOUT ：超时时间（以秒为单位）。默认为 300 秒。None，永不过时。 0 立刻过期（就是不缓存）。

**[OPTIONS](https://docs.djangoproject.com/zh-hans/3.1/ref/settings/" \l "std:setting-CACHES-OPTIONS)** ：以下选项：

MAX\_ENTRIES ：删除旧值前允许缓存的最大条目。默认是 300 。

CULL\_FREQUENCY ：达到 MAX\_ENTRIES 时被淘汰的条目。比率为 1 / CULL\_FREQUENCY，设置为2就淘汰 一半条目。一个整数，默认为3。 0 ，整个缓存都会被清空。

**[KEY\_PREFIX](https://docs.djangoproject.com/zh-hans/3.1/ref/settings/" \l "std:setting-CACHES-KEY_PREFIX)** ：将自动添加前缀，字符串。

**[VERSION](https://docs.djangoproject.com/zh-hans/3.1/ref/settings/" \l "std:setting-CACHES-VERSION)** ：版本号。

## 站点缓存

<https://docs.djangoproject.com/zh-hans/3.1/topics/cache/>

## 视图缓存

装饰器，它将自动缓存视图的响应：@cache\_page(60 \* 15)

在 URLconf 中指定视图缓存：

urlpatterns = [ path('foo/<int:code>/', cache\_page(60 \* 15)(视图名)),]

## 模板片段缓存

<https://docs.djangoproject.com/zh-hans/3.1/topics/cache/>

# 分页

**paginator = Paginator(object\_list, per\_page)** # 先拿到分页器对象，第一个：对象列表，第二个：每页显示的条数

paginator.count # 总条数

paginator.num\_pages # 总页数

paginator.page\_range # 页码数列表

**page\_obj=paginator.page(page)** # 取某一页，返回一个对象

page\_obj**.**paginator.xx #可访问**paginator** 的属性

page\_obj = paginator.page(5) # 取某一页，返回一个对象

page\_obj.object\_list # 某一页里所有数据

page\_obj.has\_next() # 是否有下一页

cpage\_obj.has\_previous() # 是否有上一页

page\_obj.next\_page\_number() # 下一页的页码数

page\_obj,previous\_page\_number() # 上一页的页码数

PY

paginator=Paginator(users,2)  
try:  
 page=request.GET.get('page',1)  
except Exception:  
 page=1  
page\_obj=paginator.page(page)

HTML

<nav aria-label="Page navigation">  
 <ul class="pagination pagination-sm">  
 {% if page\_obj.has\_previous %}  
 <li>  
 <a href="{% url 'user:show' %}?page={{page\_obj.previous\_page\_number}}" aria-label="Previous">  
 <span aria-hidden="true">上一页</span>  
 </a>  
 </li>  
 {% endif %}  
 {% for i in page\_obj.paginator.page\_range %}  
 <li {% if page\_obj.number == i %}class="active" {% endif%}><a  
 href="{% url 'user:show' %}?page={{i}}">{{i}}</a></li>  
 {% endfor %}  
 {% if page\_obj.has\_next %}  
 <li>  
 <a href="{% url 'user:show' %}?page={{page\_obj.next\_page\_number}}" aria-label="Next">  
 <span aria-hidden="true">下一页</span>  
 </a>  
 </li>  
 {% endif %}  
 </ul>  
</nav>

## ListView分页（面向对象）

class UserListView(ListView):  
 template\_name = 'user\_liat,html' #模板名  
 model = User #模型名  
 paginate\_by = 20 #每页条数  
 page\_kwart = 'p' #分页变量名

在html可调用两个

page\_obj：分页数据，如页码，第几页

object\_list：当前页码数据

# Django后台管理

创建管理员：python manage.py createsuperuser

## 步骤

### 框架部分配置

通常已配置好

1. setting.py - > INSTALLED\_APPS 添加 'django.contrib.admin',
2. setting.py -> TEMPLATES -> OPTIONS 添加

'context\_processors': [  
 。。。  
 'django.contrib.auth.context\_processors.auth',  
 'django.contrib.messages.context\_processors.messages',  
],

1. setting.py -> MIDDLEWARE 添加

MIDDLEWARE = [  
 。。。  
 'django.contrib.auth.middleware.AuthenticationMiddleware',  
 'django.contrib.messages.middleware.MessageMiddleware', <-这是倒数第二行  
 。。。  
]

1. 添加地址 path('admin/', admin.site.urls),

### ORM模型配置

1. 新建admin.py

继承admin.ModelAdmin 如果是系统用户表，继承UserAdmin

@admin.register(Profile) #注册模型类  
class ProfileAdmin(admin.ModelAdmin):  
 list\_display = ['username','sex'] #列表中显示字段  
 list\_per\_page = 15 #每页条数  
 search\_fields = ['username','sex'] #可搜索字段  
 list\_filter = ['username','sex'] #过滤器

fields/exclude 需要/不需要 的字段

form 自定义表单

ordering 排序

2.setting.py - > INSTALLED\_APPS 添加app(一般可忽略),就可

## 优化配置

### 本地化

**中文**：

1. setting.py -> LANGUAGE\_CODE = 'zh-hans'

时区：setting.py -> TIME\_ZONE = 'Asia/Shanghai'

2.apps.py -> verbose\_name = '用户账户'

class AccountsConfig(AppConfig):  
 name = 'accounts'  
 verbose\_name = '用户账户'

3.模型 -> verbose\_name = '用户表'

class User(models.Model):  
 nickname = models.CharField(verbose\_name='密码', max\_length=32)  
 class Meta:  
 db\_table = 'account\_user'  
 verbose\_name = '用户表'  
 verbose\_name\_plural = verbose\_name

### 查询速度优化

减少每页大小：list\_per\_page默认100

减少查询次数：list\_select\_related 把外键关联一并查出

@admin.register(Profile)  
class ProfileAdmin(admin.ModelAdmin):  
 list\_display = ['username','sex']  
 list\_per\_page = 15

list\_select\_related = ['user']

### 格式化字段内容

直接写个函数后，添加到list\_display就可

@admin.register(Profile)  
class ProfileAdmin(admin.ModelAdmin):  
 list\_display = ['username', 'format\_username', 'sex']  
  
 def format\_username(self, obj): #obj:Profile  
 return obj.username[:3] + '\*\*\*' + obj.username[-4:]  
 format\_username.short\_description = '用户名' #修改表头名字



### 快捷搜索

list\_filter

@admin.register(Profile)  
class ProfileAdmin(admin.ModelAdmin):  
 list\_display = ['username', 'sex']  
 list\_filter = ['sex']



### 关联搜索

search\_fields

跨表关联搜索：user\_\_nickname 外键字段双下划线加字段名

user = models.OneToOneField(to='User', related\_name='profile')

@admin.register(Profile)  
class ProfileAdmin(admin.ModelAdmin):  
 list\_display = ['username', 'format\_username', 'sex']  
 search\_fields = ['username', 'sex', 'user\_\_nickname']

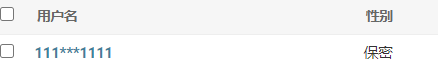
IMG_256

## 自定义模型配置

### 定制列表显示字段

list\_display

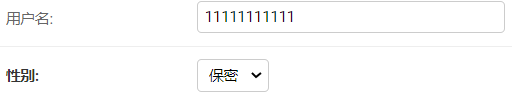
@admin.register(Profile)  
class ProfileAdmin(admin.ModelAdmin):  
 list\_display = ['format\_username', 'sex']



### 限制可编辑字段

fields/exclude 需要/不需要 的字段

@admin.register(Profile)  
class ProfileAdmin(admin.ModelAdmin):  
 list\_display = ['format\_username', 'sex']  
 fields = ['username','sex']



或 模型类字段添加editable=True

username = models.CharField('用户名', max\_length=64, editable=True)

### 自定义表单验证

form = ProfildEditForm，新建form验证

@admin.register(Profile)  
class ProfileAdmin(admin.ModelAdmin):  
 fields = ['username', 'sex','age']  
 form = ProfildEditForm

form.py

class ProfildEditForm(forms.ModelForm):  
 class Meta:  
 model = Profile  
 fields = ('username', 'sex', 'age')  
 def clean\_age(self):  
 age = self.cleaned\_data.get('age', None)  
 if 0 <= int(age) <= 120:  
 return age  
 return forms.ValidationError('年龄不在0~120之间')

### 重写保存方法，执行其他业务

在form中重写save方法，commit改false

form.py

def save(self, commit=False):  
 obj = super().save(commit)  
 # 在此执行其他操作  
 obj.save()

### 增加批量操作

定义方法xxx(self,request,queryset),后添加到actions

@admin.register(User)  
class MyUserAdmin(UserAdmin):  
 actions=['disable\_user']  
   
 def disable\_user(self,request,queryset):  
 queryset.update(is\_active=False)

disable\_user.short\_description = '名称' #显示中文名

## 用户登录

认证用户：user = authenticate(username, password)

登录：login(request,user,backend=None)

需要登录才可访问 @login\_required装饰函数def my\_view(request)

setting.py -> LOGIN\_URL ='/accounts/user/login/'

class LoginForm(forms.Form):  
 username = forms.CharField(label='用户名',required=True)  
 password = forms.CharField(label='密码',widget=forms.PasswordInput,required=True)  
  
 def clean\_username(self):  
 username = self.cleaned\_data.get('username', None)  
 if not re.search(r'1\d{10}', username):  
 raise forms.ValidationError('手机号格式不正确')  
 return username  
  
 def clean(self):  
 data = super().clean()  
 if self.errors:  
 return  
 username = data.get('username', None)  
 password = data.get('password', None)  
 user = authenticate(username=username, password=password)  
 if not user:  
 raise forms.ValidationError('用户名或密码不正确')  
 elif not user.is\_active:  
 raise forms.ValidationError('该用户被禁用')

login(request, self.user)  
 return data

# 富文本拓展

安装：pip install django-ckeditor

配置：

1. setting.py -> INSTALLED\_APPS添加'ckeditor'
2. setting.py添加STATIC\_ROOT = os.path.join(BASE\_DIR, 'static')(先创建'static'文件夹)
3. 执行：python manage.py collectstatic

文件上传配合使用时需要配置：

1. setting.py -> INSTALLED\_APPS添加'ckeditor\_uploader'
2. setting.py添加CKEDITOR\_UPLOAD\_PATH = 'uploads'
3. setting.py添加MEDIA\_ROOT=os.path.join(BASE\_DIR,'medias')(先创建'medias'文件夹)
4. 根目录urls.py添加path('ckeditor/', include('ckeditor\_uploader.urls'))

使用：

将模型中的models.TextField替换成RichTextField，就可

content = RichTextField('详细',null=True,blank=True)

# echarts

<http://echarts.apache.org/zh/index.html>

下载后放进static中就可使用

如果没有显示，在setting.py添加STATICFILES\_DIRS = [BASE\_DIR / 'static']，不能和STATIC\_ROOT同时使用

# 表单

内置Field类:<https://www.cnblogs.com/open-yang/p/11223142.html>

from django import forms  
class User(forms.Form):  
 username = forms.CharField(widget=forms.TextInput, min\_length=2, max\_length=10, label='用户名')  
 password = forms.CharField(widget=forms.PasswordInput, min\_length=2, max\_length=10, label='密码')  
 repassword = forms.CharField(widget=forms.PasswordInput, min\_length=2, max\_length=10, label='确认密码')  
 email = forms.CharField(widget=forms.EmailInput)

views.py

def ff(request):  
 if request.method == 'POST':  
 user\_form=Use(request.POST)  
 if user\_form.is\_valid():  
 data=user\_form.cleaned\_data  
 username=data.get('username')  
 print(username)  
 user\_form=Use()  
 return render(request,'ff.html',locals())

html

<form action="">{% csrf\_token %}  
 {{user\_form.as\_p}}  
 <input type="submit" value="注册">  
</form>

## 验证

单个字段clean\_+字段名

def clean\_phone(self):  
 phone = self.cleaned\_data.get('phone')

if not re.search(r'^1[0**-**9]{10}$',phone):  
 raise forms.ValidationError('手机号格式不正确')  
 return phone

多个字段 重写 clean(self)方法

def clean(self):  
 data = super().clean() #调用父类获取表单数据  
 username = data.get('username')  
 password = data.get('password')  
 if username and password:  
 user = User.objects.filter(username=username).first()  
 if not user:  
 raise forms.ValidationError('用户不存在')  
 elif user.password != password:  
 raise forms.ValidationError('密码错误')  
 return data

## 从ORM继承表单

class LoginForm(forms.ModelForm): # 继承forms.ModelForm  
 class Meta:  
 model = User  
 fields = ['username', 'password'] # 指定字段

exclude=[]#要排除字段

#可加labels，error\_messages，help\_texts，labels 等

重写save方法添加功能

@transaction.atomic #添加事务  
def save(self, commit=False):  
 user\_obj=super().save(commit)  
 user\_obj.save()

#在详情表一起添加数据  
 UserProfile.objects.create(user=user\_obj,username=user\_obj.username)  
 return user\_obj

## 文件上传

模板form要加enctype="multipart/form-data"

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

方法一：

def user\_login(request):  
 if request.method == 'POST':  
 file = request.FILES.get('img', None)  
 if file:  
 from django02.settings import MEDIA\_ROOT # 导入文件保存路径  
 filename = os.path.join(MEDIA\_ROOT[0], file.\_name)  
 with open(filename, 'wb+') as dest:  
 for chunk in file.chunks(): # 循环分批写入  
 dest.write(chunk)  
 print('ok')  
 else:  
 form = LoginForm()  
 return render(request, 'user\_login.html', context=locals())

方法二：用表单

def user\_login(request):  
 if request.method == 'POST':  
 form = LoginForm(data=request.POST, files=request.FILES)  
 if form.is\_valid():  
 file = form.cleaned\_data.get('img')  
 from django02.settings import MEDIA\_ROOT  
 filename = os.path.join(MEDIA\_ROOT[0], file.\_name)  
 with open(filename, 'wb+') as dest:  
 for chunk in file.chunks(): # 循环分批写入  
 dest.write(chunk)  
 print('ok')  
 else:  
 form = LoginForm()  
 return render(request, 'user\_login.html', context=locals())

结合ORM上传文件

可加/%Y/%m/%d自动生成年月日

avatar = models.ImageField('头像', null=True, upload\_to='avatar/%Y/%m/%d')

# 验证码

方法一．

pip install captcha

INSTALLED\_APPS添加captcha

迁移同步 make --》mig

方法二．

def get\_color():  
 red = random.randint(0, 256)  
 green = random.randint(0, 256)  
 blue = random.randint(0, 256)  
 return (red, green, blue)  
def get\_code():  
 x = 'qwertyuiopasdfghjklzxcvbnm123456789QWERTYUIOPASDFGHJKLZXCVBNM'  
 return random.sample(x, 4)  
def get\_point():  
 x = random.randint(0, 121)  
 y = random.randint(0, 41)  
 return (x, y)  
def draw\_code(request):  
 image\_size = (120, 40)  
 # 定义画布  
 image = Image.new('RGB', image\_size, get\_color())  
 # 定义画笔  
 draw = ImageDraw.Draw(image)  
  
 for i in range(10):  
 # 绘制线  
 draw.line((get\_point(), get\_point()), fill=get\_color())  
 # 绘制点  
 draw.point((get\_point(), get\_point()), fill=get\_color())  
  
 # 指定字体，大小  
 font = ImageFont.truetype(font=r'C:\Windows\Fonts\simsun.ttc', size=30)  
 code = get\_code()  
 for i in range(4):  
 # 每绘制一个字，x轴要变，y可不变  
 x = random.randint(30 \* i, 30 \* i + 5)  
 y = random.randint(0, 5)  
 draw.text((x, y), code[i], fill=get\_color(), font=font)  
  
 # 滤镜，边界加强  
 # image = image.filter(ImageFilter.EDGE\_ENHANCE\_MORE)  
 # 保存  
 # with open('1.png', 'wb') as f:  
 # image.save(f,'png')  
 # 显示图片  
 # image.show()  
 # 将图片放到缓存中  
 buffer = BytesIO()  
 image.save(buffer, 'jpeg')  
 # 获取缓存中的内容  
 buf\_bytes = buffer.getvalue()  
 # 将code保存到session  
 request.session['code'] = code  
 return HttpResponse(buf\_bytes, 'image/jpeg')  
def check\_code(request):  
 code = ''.join(request.session.get('code'))  
 ucode = request.GET.get('ucode')  
 if code.lower() == ucode.lower():  
 return JsonResponse({'status': 'success', 'msg': '验证码正确!'})  
 return JsonResponse({'status': 'fail', 'msg': '验证码不正确!'})

html

<label>验证码：</label>  
<input type="text" placeholder="请输入验证码" id="code"><img src="{% url 'user:getcode' %}" alt="" id="pic">  
<span class="err"></span><br>

js

#点击图片更换验证码  
$('#pic').click(function(){  
 $(this).attr('src','/user/getcode?ran='+Math.random());  
});  
#失去焦点事件  
$('#code').blur(function(){  
 check\_code();  
});  
var yzm=false  
#check验证码

function check\_code(){  
 var len=$('#code').val().length;  
 $('#pic').next().show();  
 if(len != 4){  
 $('#pic').next().html('请输入验证码！');  
 }else{  
 var ucode=$('#code').val();  
 $.getJSON('/user/check\_code',{ucode:ucode},function(data){  
 if(data.status == 'success'){  
 yzm=true;  
 };  
 $('#pic').next().html(data.msg);  
 })  
 }  
}

# celery异步

一

pip install redis

pip install celery-with-redis -i <https://mirrors.aliyun.com/pypi/simple/>

二

配置settings.py

# 配置celery  
BROKER\_URL = 'redis://192.168.109.129:6379/0' #redis://:password@hostname:port/db\_number  
# 用户储存结果  
CELERY\_RESULT\_BACKEND = 'redis://192.168.109.129:6379/1'  
# 连接超时  
BROKER\_TRANSPORT\_OPTIONS = {'visibility\_timeout': 3600}  
# 消息格式  
CELERY\_ACCEPT\_CONNECT = ['application/json', ]  
CELERY\_TASK\_SERIALIZER = 'json'  
CELERY\_RESULT\_SERIALIZER = 'json'  
CELERY\_TIMEZONE = TIME\_ZONE

三

在项目目录下新建celery.py

import os  
from celery import Celery  
  
project\_name = os.path.split(os.path.abspath('.'))[-1]  
project\_settings = '%s.settings' % project\_name  
os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', project\_settings)  
app = Celery(project\_name)  
app.config\_from\_object('django.conf:settings')

配置\_\_init\_\_.py

from \_\_future\_\_ import absolute\_import, unicode\_literals  
from .celery import app as celery\_app

四

task.py（异步程序）

import time  
from celery import shared\_task  
  
@shared\_task  
def send(name):  
 time.sleep(10)  
 print('1111111111111',name)

启动异步

send.delay(username)

五

启动测试异步

celery -A 项目名worker -l info

PS报错

Celery ValueError: not enough values to unpack (expected 3, got 0)

别人描述大概是说win10上运行celery4.x就会出现这个问题，解决办法如下,原理未知：

安装`eventlet：pip install eventlet

启动worker的时候加一个参数，如下：celery -A <mymodule> worker -l info -P eventlet

就可以正常的调用了。

# restful接口rest\_framework序列化

<https://www.django.cn/course/show-21.html>

FBV（function base views） CBV（class base views）

cbv

class name(View):  
 def get(self,request):  
 pass  
  
 def post(self,request):  
 pass

def delete(self):  
 pass

def put(self):  
 pass

路由：path('name', name.as\_view(), name='name'),

# Api

原生fbv接口

@csrf\_exempt # 取消csrf验证

def index(request):  
 if request.method == 'GET':  
 # return HttpResponse(json.dumps(data),content\_type='application/json') #序列化  
 return JsonResponse(data) # 同上  
 if request.method == 'POST':  
 j\_data = json.loads(request.body.decode('utf-8')) # 反序列化  
 # return HttpResponse(json.dumps(j\_data),content\_type='application/json')  
 return JsonResponse(j\_data, safe=False) # 同上 j\_data不是字典类型要safe=False

原生cbv接口

@method\_decorator(csrf\_exempt, name='dispatch') # 用类装饰器取消csrf验证  
class index(View): # 要继承View from django.views import View  
 def get(self, request):  
 return JsonResponse(data)  
 def post(self, request):  
 j\_data = json.loads(request.body.decode('utf-8'))  
 return JsonResponse(j\_data, safe=False)

## DRF中的Serializer

安装：pip install djangorestframework

class KCSerializer(serializers.ModelSerializer):  
 teacher = serializers.ReadOnlyField(source='teacher.username') # 外键字段 只读  
 class Meta: # 写法和form表单相似  
 model = User  
 # fields=('name', 'idu', 'teacher', 'price') # 要序列化的字段  
 fields = '\_\_all\_\_' # 全部  
 # exclude = ('id',) # 要排除序列化的字段  
class UserSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = AdminUser  
 fields = '\_\_all\_\_' # 全部

## DRF中的api\_view

## fbv

@api\_view(['GET', 'POST'])  
def index(request):  
 if request.method == 'GET':  
 s = KCSerializer(instance=User.objects.all(), many=True) # 从model中拿数据序列化 多个对象many=True  
 return Response(data=s.data, status=status.HTTP\_200\_OK)  
 elif request.method == 'POST':# 创建  
 s = KCSerializer(data=request.data, partial=True) # 从request中拿数据序列化,partial字段非必填可以不填  
 if s.is\_valid(): # 校验数据  
 s.save(teacher=request.user) # save中可加model字段  
 return Response(data=s.data, status=status.HTTP\_201\_CREATED)  
 return Response(s.errors, status=status.HTTP\_400\_BAD\_REQUEST) # s.errors数据错误处理  
@api\_view(['GET', 'POST', 'PUT', 'DELETE'])  
def detail(request, pk):  
 try:  
 course = User.objects.get(pk=pk)  
 except User.DoesNotExist:  
 return Response(data={'msg': '没有此课程信息'}, status=status.HTTP\_404\_NOT\_FOUND)  
 if request.method == 'GET':  
 s = KCSerializer(instance=course) # 从model中拿数据序列化 多个对象many=True  
 return Response(data=s.data, status=status.HTTP\_200\_OK)  
 elif request.method == 'PUT': # 更新  
 s = KCSerializer(instance=course, data=request.data)  
 if s.is\_valid():  
 s.save()  
 return Response(data=s.data,status=status.HTTP\_200\_OK)  
 elif request.method == 'DELETE':  
 course.delete()  
 return Response(status=status.HTTP\_204\_NO\_CONTENT)

## cbv

class CourseList(APIView):  
 def get(self, request):  
 queryset = User.objects.all()  
 s = KCSerializer(instance=queryset, many=True)  
 return Response(data=s.data, status=status.HTTP\_200\_OK)  
  
 def post(self, request):  
 s = KCSerializer(data=request.data)  
 if s.is\_valid():  
 s.save(teacher=self.request.user)  
 return Response(data=s.data, status=status.HTTP\_201\_CREATED)  
 return Response(s.errors, status=status.HTTP\_400\_BAD\_REQUEST)  
class Detail(APIView):  
 def get\_obj(self, pk):  
 try:  
 self.course = User.objects.get(pk=pk)  
 except User.DoesNotExist:  
 return Response(status=status.HTTP\_404\_NOT\_FOUND)  
 def get(self, request, pk):  
 self.get\_obj(pk=pk)  
 s = KCSerializer(instance=self.course)  
 return Response(data=s.data, status=status.HTTP\_200\_OK)  
 def put(self, request,pk):  
 self.get\_obj(pk=pk)  
 s = KCSerializer(instance=self.course, data=request.data)  
 if s.is\_valid():  
 s.save()  
 return Response(data=s.data, status=status.HTTP\_200\_OK)  
 return Response(status=status.HTTP\_400\_BAD\_REQUEST)  
 def delete(self, request,pk):  
 self.get\_obj(pk=pk)  
 self.course.delete()  
 return Response(status=status.HTTP\_204\_NO\_CONTENT)

## 通用类视图(简化)

class GcourseList(generics.ListCreateAPIView):  
 queryset = User.objects.all()  
 serializer\_class = KCSerializer  
 #由于要加teacher字段，所以重写父类创建方法  
 def perform\_create(self, serializer):  
 serializer.save(teacher=self.request.user)

## DRF中的viewset(最简)

视图集：一个视图搞定增删改查

class CourseViewSet(viewsets.ModelViewSet):  
 queryset = User.objects.all()  
 serializer\_class = KCSerializer  
 #由于要加teacher字段，所以重写父类创建方法  
 def perform\_create(self, serializer):  
 serializer.save(teacher=self.request.user)

路由注册

方法一：由于路由要跟pk，所以定义两个路由，as\_view里要加对应请求的方法名

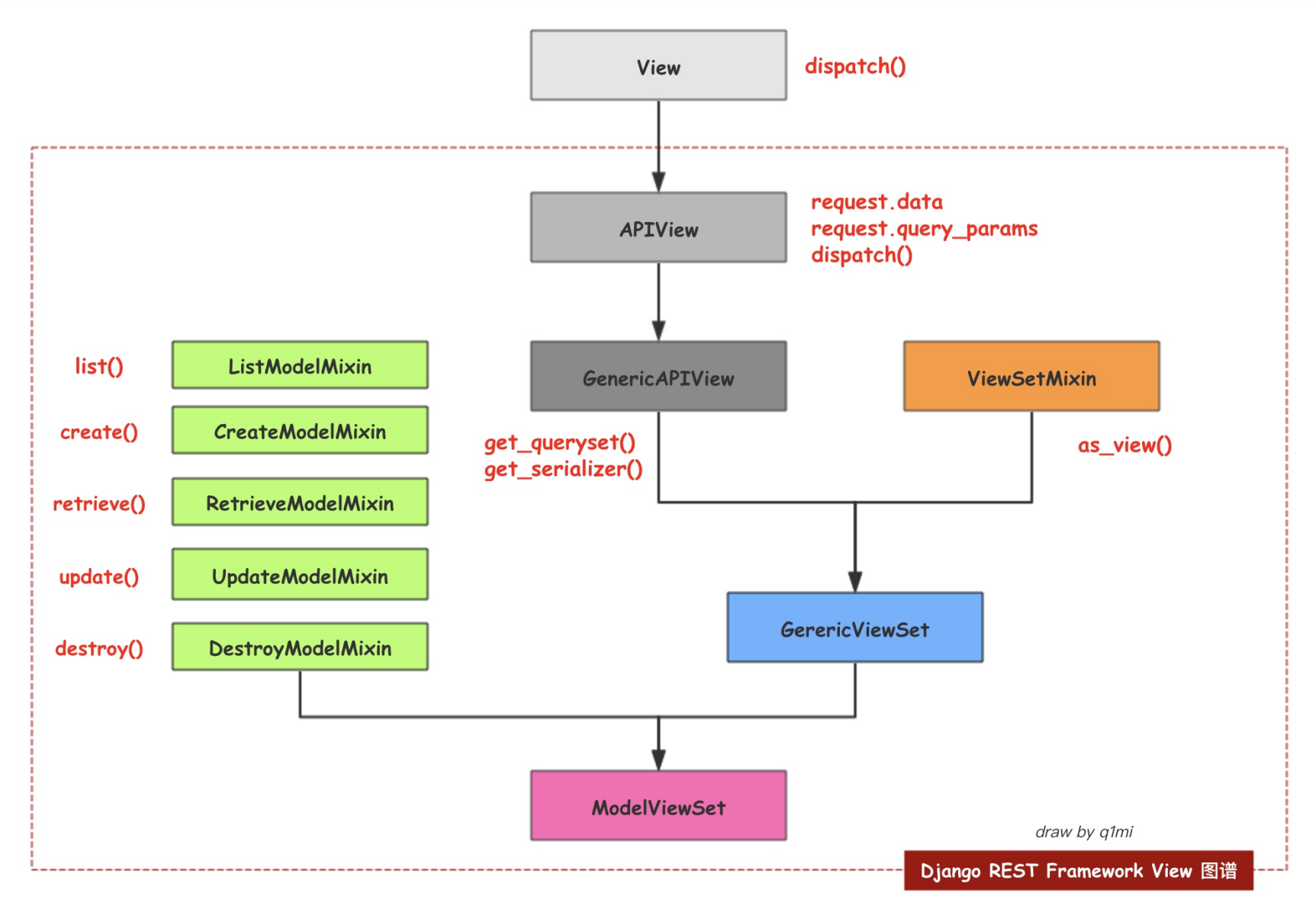
path('i/', CourseViewSet.as\_view({'get': 'list', 'post': 'create'})),  
 path('i/<int:pk>', CourseViewSet.as\_view({  
 'get': 'retrieve',  
 'put': 'update',  
 'patch': 'partial\_update',  
 'delete': 'destroy'})),  
]

方法二：

from rest\_framework.routers import DefaultRouter

router = DefaultRouter()  
router.register(prefix='i', viewset=CourseViewSet)#prefix 前缀  
  
urlpatterns = [  
 path('', include(router.urls)),  
 ]

## DRF中的认证和权限



一安装：pip install djangorestframework

**二**

将 rest\_framework添加到setting.py --> INSTALLED\_APPS

普通的models.py

from django.db import models  
  
class User2(models.Model):  
 username = models.CharField(max\_length=20, unique=True)  
 password = models.CharField(max\_length=128)  
 phone = models.CharField(max\_length=11)  
 add\_time = models.DateTimeField(auto\_now=True)  
 usertype=models.ForeignKey(to=UserType,on\_delete=models.CASCADE,related\_name='type')  
  
 class Meta:  
 db\_table = 'user2'  
  
 def \_\_str\_\_(self):  
 return self.username

创建serializers.py,定义需要序列化的models的字段

from rest\_framework import serializers  
  
  
class UserSerializer(serializers.Serializer):  
 id = serializers.IntegerField(read\_only=True)  
 username = serializers.CharField(max\_length=20, required=True)  
 password = serializers.CharField(max\_length=128, required=True)  
 phone = serializers.CharField(max\_length=11, min\_length=11)

简化：继承serializers.ModelSerializer，使用modles

class UserSerializer2(serializers.ModelSerializer):

#repassword = serializers.CharField(max\_length=128, required=True)

#可在此添加格外的字段

#repassword = serializers.CharField(max\_length=128, write\_only=True)

#如果要添加数据库没有的字段要加write\_only，表示只是在前端使用

class Meta:  
 #使用models模型  
 model=User2  
 #包含字段  
 fields=['username','password','phone']#'\_\_name\_\_'

三

视图views.py

from django.shortcuts import render  
from django.http import HttpResponse,JsonResponse  
  
from rest\_framework.views import APIView  
from bb.models import User2  
from bb.serializers import UserSerializer  
class UserViews(APIView):  
 def get(self, request):  
 return render(request,'1.html')  
  
 def post(self, request):  
 username=request.POST.get('username')  
 password = request.POST.get('password')  
 phone = request.POST.get('phone')  
 user=User2.objects.create(username=username,password=password,phone=phone)

#序列化类  
 **user\_serializer=UserSerializer(user)**  
 print(user\_serializer.data)  
 return JsonResponse({'status':200,'user':user\_serializer.data})

简化：

class UserViews(APIView):  
 def get(self, request):  
 return render(request,'1.html')  
  
 def post(self, request):  
 #传入序列化好的数据request.data

#相当于data=JSONParser().parse(request)  
 user=UserSerializer2(data=request.data)  
 #验证  
 if user.is\_valid():  
 #保存数据  
 user.save()

return JsonResponse({'status': 200,'user':user.data})  
 return JsonResponse({'status':500})

定制serializers，基于serializers.ModelSerializer

1.定制create：重写父类的create方法

def create(self, validated\_data):  
 username=validated\_data['username']  
 password=validated\_data['password']  
 #密码加密  
 password=make\_password(password)  
 user=User2.objects.create(username=username,password=password)  
 return user

2定制字段校验：对全局属性进行验证

def validate(self, attrs):

attrs就是提交数据的字典格式  
 pass

例子

class SmsSerializer(serializers.Serializer):  
 phone = serializers.CharField(max\_length=11, min\_length=11, error\_messages={  
 'min\_length': '手机号为11位！',  
 'max\_length': '手机号为11位！', })  
  
 def validate\_phone(self, phone):  
 if User.objects.filter(phone=phone).exists():  
 raise serializers.ValidationError('手机号已注册！')  
 if not re.match(r'1[0**-**9]{10}]', phone):  
 raise serializers.ValidationError('手机号格式错误！')  
 return phone

## 关系型模型序列化

UserType1 --> User 多

class User2(models.Model):  
 。。。  
 usertype=models.ForeignKey(to=UserType,on\_delete=models.CASCADE,related\_name='users')

1.序列化

class UserTypeSerializer(serializers.ModelSerializer):  
 #名称要和models中的related\_name一致  
 users=serializers.StringRelatedField(many=True)  
 class Meta:  
 model = UserTpye  
 fields = ['name', 'type\_id', 'users']#要加上自定义的users

1. views.py

class UserTypeView(APIView):  
 def get(self,request):  
 usertypes=UserType.objects.all()  
 #序列化多个要加many=True  
 serializer = UserTypeSerializer(usertypes,many=True)  
 data={  
 'status':200,  
 'types':serializer  
 }  
 return JsonResponse(data=data)

## 最终

from rest\_framework.generics import ... generics 里有各种方法

一models.py

二serializers.py 序列化类

1.class UserSerializer(serializers.Serializer):

pass

2.class UserSerializer2(serializers.ModelSerializer):

# password = serializers.CharField(max\_length=128, write\_only=True)

class Meta:

model=User2 #使用models模型

fields=['username','password','phone']#'\_\_name\_\_' #包含字段

def validate(self, attrs):

pass

三views。py

class UserView(ModelViewSet):

queryset = User.objects.all()

serializer\_class =UserSerializer2

路由：

router=DefaultRouter()

router.register(r’name’,User)

urlpatterns = [

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

]

例子

from django.contrib import admin  
from django.urls import path, include  
  
from rest\_framework.routers import DefaultRouter  
  
from user.views import SmsViewset  
  
router = DefaultRouter()  
# 用户  
router.register('code', SmsViewset, basename='code')  
  
# 电影  
  
  
# 影院  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('api/', include(router.urls)),  
]

## Session认证 权限设置

全局：setting。py

# REST\_FRAMEWORK = {

# 'DEFAULT\_AUTHENTICATION\_CLASSES': [

# 'rest\_framework.authentication.BasicAuthentication',

# 'rest\_framework.authentication.SessionAuthentication',

# ],

# 'DEFAULT\_PERMISSION\_CLASSES': [

# 'rest\_framework.permissions.IsAuthenticated',

# ]

# }

局部：views。py

class UserViewset(CreateModelMixin, UpdateModelMixin, RetrieveModelMixin, GenericViewSet):  
 queryset = User.objects.all()  
 # serializer\_class = RegisterSerializer  
 # 局部添加权限  
 authentication\_classes = [BasicAuthentication, SessionAuthentication]  
  
 # permission\_classes = [IsAuthenticated]  
 def get\_permissions(self):  
 if self.action != 'create':  
 return [IsAuthenticated(), ]

return []

## token认证

1 setting。py添加

INSTALLED\_APPS = [

'rest\_framework.authtoken',

]

2迁移同步数据库

3添加路由

from rest\_framework.authtoken import views

ulpatterns = [ url(r'^api-token-auth/', views.obtain\_auth\_token) ]

4 添加token认证

authentication\_classes 添加TokenAuthentication

class MovieListViewSet(ListModelMixin, GenericViewSet):  
 queryset = Movie.objects.all()  
 serializer\_class = MovieListSerializer  
  
 authentication\_classes = [BasicAuthentication,TokenAuthentication]  
 permission\_classes = [IsAuthenticated, ]

请求头要加上Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b

要带前缀Token

## JWT认证

token缺点：1.token 会被永久保存，不安全 2.对于分布式系统，需要同步token。

1安装 pip install djangorestframework-jwt

2配置

打开项目中settings配置，全局添加在这写REST\_FRAMEWORK

REST\_FRAMEWORK = {  
 # 设置所有接口都需要被验证  
 'DEFAULT\_PERMISSION\_CLASSES': (  
 #'rest\_framework.permissions.IsAuthenticatedOrReadOnly',  
 ),  
 # 用户登陆认证方式  
 'DEFAULT\_AUTHENTICATION\_CLASSES': (  
 'rest\_framework\_jwt.authentication.JSONWebTokenAuthentication',  
 #’rest\_framework.authentication.SessionAuthentication’,  
 #’rest\_framework.authentication.BasicAuthentication’,  
 ),}  
# jwt载荷中的有效期设置  
JWT\_AUTH = {  
 #token 有效期  
 'JWT\_EXPIRATION\_DELTA': datetime.timedelta(days=7),  
 'JWT\_ALLOW\_REFRESH': True,  
 #续期有效期（该设置可在24小时内带未失效的token 进行续期）   
 'JWT\_REFRESH\_EXPIRATION\_DELTA': datetime.timedelta(days=1),  
 # 自定义返回格式，需要手工创建  
 'JWT\_RESPONSE\_PAYLOAD\_HANDLER': 'Users.utils.jwt\_response\_payload\_handler',  
 #Token前缀  
 'JWT\_AUTH\_HEADER\_PREFIX':'JWT'  
}

局部添加在视图添加 JSONWebTokenAuthentication

class MovieListViewSet(ListModelMixin, GenericViewSet):  
 queryset = Movie.objects.all()  
 serializer\_class = MovieListSerializer  
  
 authentication\_classes = [BasicAuthentication, SessionAuthentication,

JSONWebTokenAuthentication]  
 permission\_classes = [IsAuthenticated]

3路由

from rest\_framework\_jwt.views import obtain\_jwt\_token

urlpatterns = [ path('login/',obtain\_jwt\_token) ]

## 分页

<https://www.django-rest-framework.org/api-guide/pagination/>

分页种类：

PageNumberPagination ：GET <https://api.example.org/accounts/?page=4>

HTTP 200 OK

{

"count": 1023

"next": "https://api.example.org/accounts/?page=5",

"previous": "https://api.example.org/accounts/?page=3",

"results": [

…

]

}

LimitOffsetPagination ：GET <https://api.example.org/accounts/?limit=100&offset=400>

HTTP 200 OK

{

"count": 1023

"next": "https://api.example.org/accounts/?limit=100&offset=500",

"previous": "https://api.example.org/accounts/?limit=100&offset=300",

"results": [

…

]

}

自定义分页器类

class MoviePagination(PageNumberPagination):

page\_size = 10

page\_query\_param = 'p'

max\_page\_size = 20

## 配置

全局，settings中配置

REST\_FRAMEWORK = {

#默认分页类  
 'DEFAULT\_PAGINATION\_CLASS': 'rest\_framework.pagination.LimitOffsetPagination',  
 'PAGE\_SIZE': 100 #每页条数  
}

局部，在类视图中加

class MovieListViewSet(ListModelMixin, GenericViewSet):  
 。。。  
 pagination\_class = LimitOffsetPagination #定义分页类

## 筛选过滤

安装：pip install django-filter

加入setting -> INSTALLED\_APPS = [‘django-filter’]

[https://www.django-rest-framework.org/api-guide/filtering/#djangofilterbackend](https://www.django-rest-framework.org/api-guide/filtering/" \l "djangofilterbackend)

1 DjangoFilterBackend

设置全局

REST\_FRAMEWORK = {

'DEFAULT\_FILTER\_BACKENDS': ['django\_filters.rest\_framework.DjangoFilterBackend']

}

局部

from django\_filters.rest\_framework import DjangoFilterBackend

class ProductListView(generics.ListAPIView):  
 queryset = Product.objects.all()  
 serializer\_class = ProductSerializer  
 filter\_backends = [DjangoFilterBackend] #可加多个  
 filterset\_fields = ['category', 'in\_stock'] #过滤字段

2 SearchFilter 文本类型字段，如CharField或TextField。

from rest\_framework import filters  
class UserListView(generics.ListAPIView):  
 queryset = User.objects.all()  
 serializer\_class = UserSerializer  
 filter\_backends = [SearchFilter]  
 search\_fields = ['username', 'email']

3.OrderingFilter 排序，同上

filter\_backends = [OrderingFilter]  
ordering\_fields = ['username', 'email']

4 自定义过滤类

新建filter.py

from django\_filters import FilterSet  
from django\_filters  
class OrderFilter(FilterSet):  
 totalmax=django\_filters.NumberFilter(field\_name='total',lookup\_expr='get')  
 totalmin=django\_filters.NumberFilter(field\_name='total',lookup\_expr='lt')

movie\_id=django\_filters.NumberFilter(field\_name='movie\_id')  
  
 def movie\_id\_filter(self,queryset,name,value):  
 #收索电影表的电影id或用户表的电影id  
 result=queryset.filter(Q(movie\_id=value)|Q(user\_order\_movie\_id=value))  
 return result

class Meta:  
 model=Order  
 fields=['total',]

views.py

filter\_class=OrderFilter

5

默认情况下，搜索不区分大小写的模糊查询。

可以通过在字符前面添加各种字符来限制搜索行为search\_fields。

'^'开始搜索。 '='完全匹配。

'@'全文搜索。（当前仅支持Django的PostgreSQL后端。） '$'正则表达式搜索。

例如：search\_fields = ['=username', '=email']

6

# 083017404106_0无标题1

# 跨域

1安装：pip install django-cors-headers

2加入setting -> INSTALLED\_APPS = [‘corsheaders’]

3添加中间件

MIDDLEWARE = [

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

**'corsheaders.middleware.CorsMiddleware', #加在**Session，Common的中间

'django.middleware.common.CommonMiddleware',]

4 setting配置

CORS\_ALLOW\_CREDENTIALS = True

CORS\_ORIGIN\_ALLOW\_ALL = True #允许所有

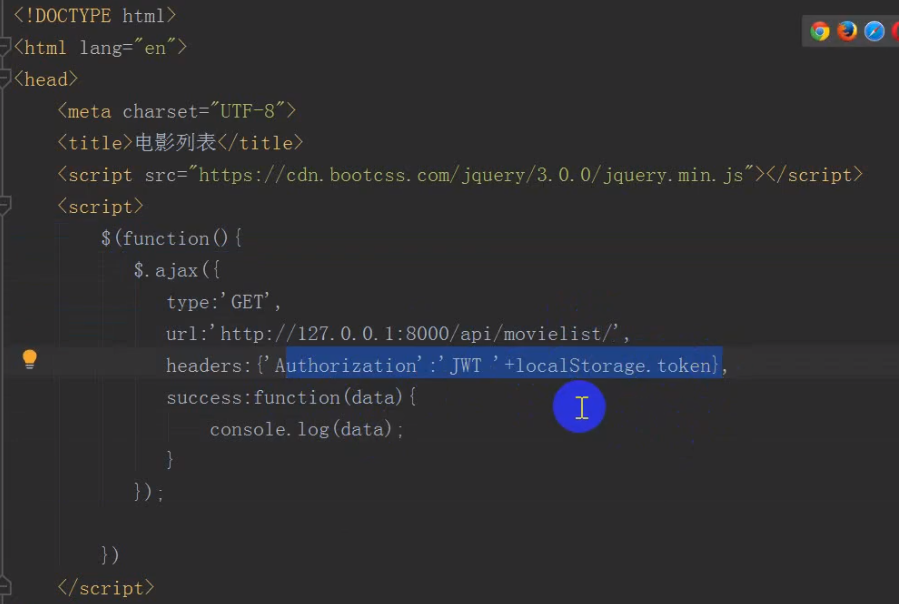
或

CORS\_ORIGIN\_WHITELIST = ( #跨域白名单

'\*', #所有

‘192.168.1.1:8000’,

)



# 发送短信验证码模块

def send\_sms(phone):  
 headers = {}  
 url = 'https://api.netease.im/sms/verifycode.action'  
 AppSecret='d6586248eca4'  
 headers['AppKey'] = '3abe0cfc4778e0980e80f8d22d47098f'  
 headers['Content-Type'] = 'application/x-www-form-urlencoded;charset=utf-8'  
 headers['Nonce'] = str(uuid.uuid4()).replace('-', '')  
 headers['CurTime'] = str(int(time.time()))  
 headers['CheckSum'] = hashlib.sha1((AppSecret+headers['Nonce']+headers['CurTime']).encode('utf-8')).hexdigest()  
 response=requests.post(url=url, data={'mobile': phone}, headers=headers)  
   
 json\_result=response.json()   
 if json\_result.get('code')==200:  
 return JsonResponse({'msg':'短信发送成功!'})  
 else:  
 return JsonResponse({'msg': '短信发送失败!'})

# 案例

## 自己云盘

form

class RegisterForm(forms.ModelForm):  
 repassword = forms.CharField(label='确认密码', widget=forms.PasswordInput(attrs={'id': 'repassword'}))  
 class Meta:  
 model = UserModel  
 fields = ['username', 'password', 'repassword', 'phone']  
 widgets = {  
 'username': forms.TextInput(attrs={'id': 'username'}),  
 'password': forms.PasswordInput(attrs={'id': 'password'}),  
 'phone': forms.TextInput(attrs={'id': 'phone'}),  
 }  
 @transaction.atomic  
 def save(self, commit=False):  
 # 保存到数据库  
 user\_obj = super(RegisterForm, self).save(commit)  
 user\_obj.save()  
 UserProfile.objects.create(user=user\_obj)  
 return user\_obj  
 def clean(self):  
 # 验证表单  
 data = super(RegisterForm, self).clean()  
 username = data.get('username')  
 password = data.get('password')  
 repassword = data.get('repassword')  
 phone = data.get('phone')  
 user = UserModel.objects.filter(username=username).exists()  
 if not user and (password == repassword):  
 return data  
 raise forms.ValidationError('数据错误')  
class LoginForm(forms.ModelForm):  
 class Meta:  
 model = UserModel  
 fields = ['username', 'password']  
 widgets = {  
 'username': forms.TextInput(attrs={'id': 'username'}),  
 'password': forms.PasswordInput(attrs={'id': 'password'})}  
class FileForm(forms.ModelForm):  
 class Meta:  
 model = FileModel  
 fields = ['path']

model

class BaseModel(models.Model):  
 create\_at = models.DateTimeField('创建时间', auto\_now\_add=True)  
 update\_at = models.DateTimeField('更新时间', auto\_now=True)  
 class Meta:  
 abstract = True

class ...  
...  
class FileModel(BaseModel):  
 TYPE = (('0', '图片'), ('1', '视频'), ('2', '文件'),)  
 user = models.ForeignKey(to=UserModel, on\_delete=models.CASCADE, related\_name='file\_user', verbose\_name='关联用户')  
 feil\_name = models.CharField('文件名', max\_length=256, null=True, blank=True)  
 path = models.FileField('地址', upload\_to='avatar/%Y/%m/%d', max\_length=256, null=True, blank=True)  
 type = models.CharField('类型', max\_length=1, choices=TYPE, null=True, blank=True)  
  
 class Meta:  
 db\_table = 'file\_model'  
 verbose\_name = '文件表'  
 verbose\_name\_plural = verbose\_name  
 def \_\_str\_\_(self):  
 return self.feil\_name

view

class Login(View):  
 def get(self, request):  
 if request.session.get('username', None):  
 return redirect(reverse('index'))  
 form = LoginForm()  
 return render(request, 'user\_login.html', context=locals())  
 def post(self, request):  
 username = request.POST.get('username', None)  
 password = request.POST.get('password', None)  
 if username and password:  
 user = UserModel.objects.filter(username=username).first()  
 if user:  
 if password == user.password:  
 request.session['username'] = user.username  
 else:  
 raise forms.ValidationError('密码错误')  
 else:  
 raise forms.ValidationError('用户不存在')  
 else:  
 raise forms.ValidationError('数据错误')  
 return redirect(reverse('index'))

...