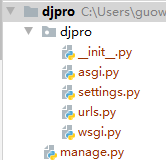
## 1、项目创建

django-admin startproject 项目名称



文件说明：

manage.py 项目管理文件

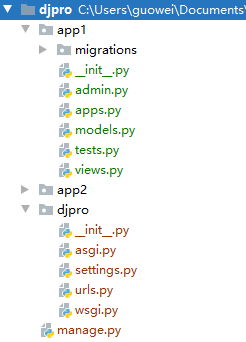
settings.py 项目配置文件

wsgi.py：是web服务和django交互的入口

一个项目由多个应用组成。需要创建多个app。

### 1.1、创建应用

python manage.py startapp 应用名称



注册应用

项目settings.py配置文件添加下面

INSTALLED\_APPS = [  
 ...  
 **'django.contrib.staticfiles'**,

'app1' # 应用名称  
]

### 1.2、启动项目

python manage.py runserver 0.0.0.0:8888

## 2、ORM框架

对象关系映射框架，将类和数据库表对应起来，只要通过类和对象就可以操作数据库。

根据设计的类生成数据库表。

## 3、模型类

### 3.1、模型类设计

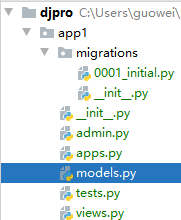
app/models.py

**from** django.db **import** models**class** BookInfo(models.Model):  
 *"""图书模型类"""  
 # 图书名称 名称长度255* btitle = models.CharField(max\_length=255)  
 *# 日期类型 出版日期* bpub\_date = models.DateField()

**def** \_\_str\_\_(self):  
 **return** self.btitle

### 3.2、生成迁移文件

python manage.py makemigrations



### 3.3、生成表

python manage.py migrate

### 3.4、模型类操作数据库

python manage.py shell

可进入命令行操作数据库

增

from app1.models import Bookinfo

from datetime import date

b = Bookinfo()

b.btitle = "天龙八部"

b.bpub\_date = date(2021,1,1)

b.save()

查

b2 = Bookinfo.object.get(id=1)

print(b2.btitle,b2.bpub\_date)

改

b2 = Bookinfo.object.get(id=1)

b2.bpub\_date=date(2021,2,21)

b2.save()

删

b2 = Bookinfo.object.get(id=1)

b2.delete()

### 3.5、表关联和关联操作

外键关联

**class** BookInfo(models.Model):  
 *"""图书模型类"""  
 # 图书名称* btitle = models.CharField(max\_length=255)  
 *# 日期类型 出版日期* bpub\_data = models.DateField()  
  
  
**class** Heroinfo(models.Model):  
 *"""英雄类"""* hname = models.CharField(max\_length=25)  
 hsex = models.BooleanField(default=**False**)  
 hbook = models.ForeignKey(**'BookInfo',** on\_delete=models.CASCADE)

on\_delete=models.CASCADE 在django2.0之后需要加上此选项

执行生成表操作

生成外键名称

hbook\_id

增

**from** app1.models **import** BookInfo, Heroinfo  
**from** datetime **import** date

b = BookInfo()  
b.btitle = **"天龙八部"**b.bpub\_data = date(2021, 2, 21)  
b.save()  
  
h = Heroinfo()  
h.hname = **"乔峰"**h.hsex = **True**

**// 插入一个对象**h.hbook = b  
h.save()

查

h2 = Heroinfo.objects.get(id=2)

查询所有

h2\_all = Heroinfo.objects.all()  
*# 英雄名称*h2.hname  
*# 图书名称*h2.hbook.btitle  
*# 图书编号*h2.hbook\_id

查看和图书关联的所有英雄

ball = b.heroinfo\_set.all()

ball[0].hname

### 3.6、mysql数据库配置

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql', # 数据库引擎

'NAME': 'django\_mysql',

'USER': 'root',

'PASSWORD': 'root',

'HOST': '118.24.40.136',

'POST': 3306,

}

}

pip install pymysql

python3

项目\_\_init\_\_.py，添加

**import** pymysql  
pymysql.install\_as\_MySQLdb()

### 3.7、字段属性和选项

BooleanField

CharField(max\_length)

TextField 大文本字段，超过4k字符使用

IntegerField

DecimalField(max\_digits=3,decimal\_places=2) digits小数前数字长度,place小数点后长度

FloatField 浮点数

DateField(auto\_now=True,auto\_now\_add=True) 可选参数，不可组合使用auto\_now\_add创建时间，auto\_now修改时间。

TimeField 同上

DateTimeField 同上

FileField 文件

ImageField 图片

选项

default

primary\_key

unique 唯一

db\_index 索引

db\_column 字段名称 指定生成表的字段名称，默认数据库表字段使用属性名称

null 字段空

blank 字段空白 用于后台管理界面的设置，True就是允许不填写内容。

### 3.8、查询

查询函数

get

返回一条，只可以有一条模型类对象。查询多条或者查不到都会报异常。

all

返回所有数据。返回查询集。QuerySet

filter

返回满足条件的数据。返回查询集。QuerySet

exclude

返回不满足条件的数据。返回查询集。QuerySet

Bookinfo.objects.exclude(id=3) //id不为3

order\_by

对结果进行排序。

Bookinfo.objects.all().order\_by("id","btitle") //从小到大

Bookinfo.objects.all().order\_by("-id") // 从大到小

所有查询的话all()可以省略。

Bookinfo.objects.filter(id\_\_gte=1).order\_by("")

注意：对于查询结果是QuerySet的，可以继续调用下面操作。

#### 模糊查询

get、filter、exclude都可以使用模糊查询

// 包含

Bookinfo.objects.filter(bname\_\_contains='传')

// 开头结尾 endswith startswith

BookInfo.objects.filter(bname\_\_endswith=‘传’)

#### 不为空查询isnull

不为空false，为空true

Bookinfo.objects.filter(bname\_\_isnull=False)

#### 范围查询in

Bookinfo.objects.filter(id\_\_in=[1,2,3])

#### 比较查询gt lt

gt、lt、gte、lte 大于、小于、大于等于、小于等于

Bookinfo.objects.filter(id\_\_gt=1) // id>1

Bookinfo.objects.filter(id\_\_gte=1) // id>=1

#### 日期查询

Bookinfo.objects.filter(bpub\_date\_\_year=2021) // 查询年份2021

Bookinfo.objects.filter(bpub\_date\_\_gte=date(2021,2,21)) //日期大于等于20210221年

### 3.9、F对象

属性之间的比较

阅读量>下载量

from django.db.models import F

Bookinfo.objects.filter(bcomment\_\_gt=F('bdowncount'))

阅读量>2倍下载量

Bookinfo.objects.filter(bcomment\_\_gt=F('bdowncount')\*2)

### 3.10、Q对象

查询之间的逻辑关系not、and、or Q对象使用&|~

from django.db.models import Q

Bookinfo.objects.filter(Q(id\_\_gt=1)&Q(bname\_\_contains='传'))//&表示且

Bookinfo.objects.filter(~Q(id=3)) // id不等于3

### 3.11、聚合函数

先导入聚合函数

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

Bookinfo.objects.all().aggregate(Count("id")) // 返回字典

返回值

"id\_\_count":5

Bookinfo.objects.filter(id\_\_gt=1).aggregate(Count("id"))

count函数

返回满足数量的条目

Bookinfo.objects.all().count()

Bookinfo.objects.filter(id\_\_gt=3).count()

### 3.12、查询集

查询集特性

1、惰性查询：实际使用中才会对数据库进行查询。

b = Bookinfo.objects.all() //没有进行数据库查询

print(b) //进行数据库查询

2、缓存

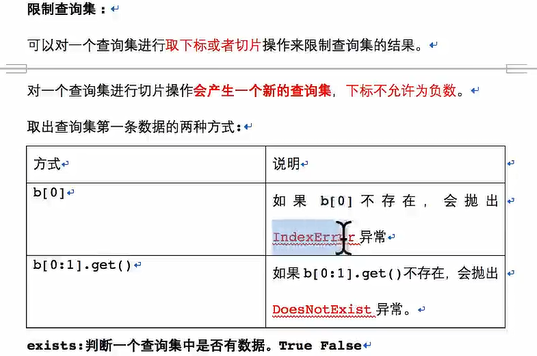
使用同一个查询及，第一次查询会把查询结果缓存起来，再次使用是缓存中的数据。

b = Bookinfo.objects.all() //没有进行数据库查询

print(b)

print(b) // 使用缓存

限制查询集



### 3.13、模型类关系

#### 一对多

models.ForeignKey() 定义在多类

#### 一对一

models.OneToOneField() 定义在哪个类都可以

type\_news = models.OneToOneField(**"BookInfo"**)

#### 多对多

models.ManyToManyField() 定义在哪个类都可以

type\_News=models.ManyToManyField(**"BookInfo"**)

### 3.14、关联查询 一对多

#### 通过对象查询

查询id为1的图书的英雄信息

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

b.heroinfo\_set.all()

查询id为1的英雄图书信息

h = Heroinfo.objects.filter(id=1)

h.hbook

#### 通过模型类查询

查询图书信息，要求英雄描述包含‘八’

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

查询天龙八部的所有英雄

Heroinfo.objects.filter(hbook\_\_btitle="天龙八部")

### 3.15、自关联



是一种特殊的一对多关系。

**class** Areas(models.Model):  
 atitle = models.CharField(max\_length=255)  
 aparent = models.ForeignKey(**"self"**, null=**True**, blank=**True**)

def index(request):

a=Areas.objects.get(atitle="广州市")

# 上级地区

parent = a.aparent

# 下级地区

children = a.Areas\_set.all()

### 3.16、管理器

django自带的模型管理器objects对象。

自定义模型管理器

1. 为了改变查询结果集.
2. 封装函数

**class** HeroInfoManager(models.Manager):  
 *# 1、改变查询的结果集* **def** all(self):  
 hero = super().all()  
 *# 查询标记为未删除的数据* hero1 = hero.filter(isDelete=1)  
 **return** hero1  
 2、封装函数：操作模型类增删改查  
 **def** save\_data(self, hname, hsex):

# 类名发生变化就需要多出改，使用self.model

# h1=self.model

# obj=h1()

# 上面3句和下面一样作用  
 obj = Heroinfo()  
 obj.hname = hname  
 obj.hsex = hsex  
 obj.save()  
 **return** obj  
  
**class** Heroinfo(models.Model):  
 *"""英雄类"""* hname = models.CharField(max\_length=25)  
 hsex = models.BooleanField(default=**False**)  
 hbook = models.ForeignKey(**'BookInfo'**, on\_delete=models.CASCADE)  
 *# 自定义类的模型管理器* object = HeroInfoManager()  
  
 **def** \_\_str\_\_(self):  
 **return** self.hname

模型类使用

Heroinfo.object.save\_data("超人",True)

### 3.17、元选项

模型类表名不依赖应用名

**class** Heroinfo(models.Model):  
 *"""英雄类"""* hname = models.CharField(max\_length=25)  
 hsex = models.BooleanField(default=**False**)  
 hbook = models.ForeignKey(**'BookInfo'**, on\_delete=models.CASCADE)  
 *# 自定义类的模型管理器* object = HeroInfoManager()  
  
 **def** \_\_str\_\_(self):  
 **return** self.hname  
  
 **class** Meta:  
 db\_table = **"heroinfo"**

## 4、后台管理

### 4.1、本地化

语言和时区

LANGUAGE\_CODE = **'zh-hans'**TIME\_ZONE = **'Asia/Shanghai'**

### 4.2、管理员创建

python manage.py createsuperuser

登录

localhost:8000/admin

### 4.3、注册模型类

在admin.py里面

**from** django.contrib **import** admin  
**from** app1.models **import** BookInfo,Heroinfoadmin.site.register(BookInfo)  
admin.site.register(Heroinfo)

需要页面显示指定内容，在模型类添加\_\_str\_\_方法

### 4.4、自定义管理界面

admin.py

**from** django.contrib **import** admin  
**from** app1.models **import** BookInfo, Heroinfo**class** BookinfoAdmin(admin.ModelAdmin):  
 list\_display = [**'id'**, **'btitle'**, **'bpub\_data'**]  
  
**class** HeroinfoAdmin(admin.ModelAdmin):  
 list\_display = [**'id'**, **'hname'**, **'hsex'**, **'hbook'**]  
  
admin.site.register(BookInfo, BookinfoAdmin)  
admin.site.register(Heroinfo, HeroinfoAdmin)



## 5、视图

视图处理请求。

视图函数使用过程

1、创建视图函数在views.py

**from** django.http **import** HttpResponse  
**from** django.shortcuts **import** render  
**def** login(request):  
 *# 返回HttpResponse对象* **return** HttpResponse(**"hello world"**)

2、url配置,建立url地址和视图的关系

项目/urls.py

urlpatterns = [  
path(**'app1/'**, include(**'app1.urls'**))

]

应用/urls.py  
**from** app1 **import** views  
  
urlpatterns = [  
 path(**'login'**, views.login)  
]

## 6、模板

### 6.1、使用模板

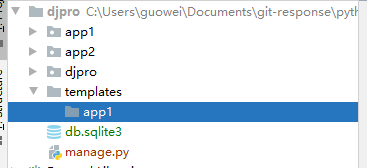
1、项目目录下创建templates文件夹

2、配置模板目录

在settings.py文件下

TEMPLATES = [  
 {  
 **'DIRS'**: [os.path.join(BASE\_DIR, **'templates'**)],  
 },  
]

3、templates下面创建应用模板的文件夹appname



4、使用模板文件

1)、加载模板文件，获取html的内容,得到模板对象

2)、定义模板文件上下文，向模板传递数据

3)、渲染模板，得到标准html内容

**def** login(request): *# 加载模板文件* templates\_cl = loader.get\_template(**'app1/index.html'**)  
 *# 定义模板上下文  
 # context = RequestContext(request, {"name": "张三"})* context = {}  
 *# 渲染* resp\_html = templates\_cl.render(context)  
 **return** HttpResponse(resp\_html)

简写方式 render已经封装好了

**def** login(request):

hero1=HeroInfo.objects.all()  
 **return** render(request, **"app1/index.html"**, {"name":"wdd","lists":[1,2,43,5],"heros":hero1})

模板使用变量

<**h1**>hello {{ **name** }}</**h1**>

循环

{% for i in list%}

{{ i }}

{% endfor %}

遍历对象

{% for i in heros %}

{{ i.hname }}{{ i.hage }}

{% endfor %}

### 6.2、重定向

跳转到url

**from** django.template **import** loader, RequestContext

**def** index(request, bid):  
 print(bid)**return** HttpResponseRedirect(**'/app1/login'**)

## 7、视图

### 自定义错误页面

在templates下面新建404.html和500页面

在seetings.py设置

DEBUG=False

ALLOWED\_HOST=['\*']

### 获取url参数