Python Web框架Django课程

Django第3天

项目目录结构

./manage.py 用于管理项目

Django02/settings.py

BASE_DIR 项目文件夹的路径
DEBUG 调试模式
ALLOW_HOST 允许哪些主机访问
INSTALLED_APPS 安装的应用
MIDDLEWARE 中间件
ROOT_URLCONF 项目跟路由配置文件
TEMPLATES 模板引擎配置
DATABASES 数据库设置

Django02/urls.py

路由配置信息

Library/models.py

定义模型

Library/views.py

定义路由对应的处理函数

./manage.py 参数说明

./manage.py —help
查看帮助信息

./mange.py help 子命令
查看子命令帮助信息

./manage.py startapp app名称
新建app，然后将app名称添加到INSTALLED_APPS中

./manage.py runserver [ip:port]
如果让其他人连接需要把ip改为0.0.0.0，并且修改settings文件中ALLOW_HOST

./manage.py makemigrations app名称 迁移文件的名字
构建迁移模型

./manage.py sqlmigrate app名称 迁移文件的名字
查看迁移模型将要执行的sql语句

./manage.py migrate app名称
执行迁移模型到数据库中

./manage.py shell 打开交互环境

新建数据库用户:grant all on *.* to ‘django’@‘%’ identified by ‘djangopwd’
新建数据库:create database django;

新建模型

```
class Student(models.Model):
    name = models.CharField(max_length=32)
    age = models.IntegerField(default=16)
    klass = models.CharField(max_length=32)
    grade = models.IntegerField(default=1)
    klass_id=models.ForeignKey("Klass",null=True,)

class Klass(models.Model):
    title=models.CharField(max_length=16)
    student_numbers=models.IntegerField()
```

```
./manage.py makemigrations app名称
./manage.py migrate app名称
./manage.py shell
```

创建数据

```
s=Student()
s.属性=值
s.save()
```

查询 Student.objects.[all,filter,get,exclude,last,first,latest](查询条件)

更新

```
s.属性=值
s.save()
Student.objects.查询函数(查询条件).update(属性=值)
```

删除 s.delete() 或者 Student.objects.查询函数(查询条件).delete()

查询字段__[lt,gt,lte,gte,contains,isnull,in,endswith,startswith]

查询参数中的查询字段是以and逻辑连接

```
Student.objects.filter(name__contains='Abc',age__gt=20)
```

关系字段的读取
多对一读取
```
st.klass_id.title
```
一对多读取
```
klass1.student_set.查询函数,默认是:类名称小写_set
Klass1.student_set.filter()
```

今天内容

查询函数可以级联操作

Student.objects.filter(age__lt=10).exclude(grade=20).filter(name__contains='002')

关联对象的查询
Student.objects.filter(klass_id__title='1801')

```
Student.objects.filter(klass__floor__name__contains='python')
```

F对象:

对象的A属性与B属性进行比较,并且支持运算 Video.objects.fillter(like__lt=F('unlike')+5)

Q函数

Q(age__lt=20) | Q(grade__lt=5)

filter(~Q(age__lt=20)) filter(age__gte=20)

exclude(age__lt=20) filter(age__gte=20)

where查询除了支持and 还支持or not逻辑判断

Django是使用Q(djang.db.models.Q)来支持or和not查询

Student.objects.filter(Q(age__lt=20) | Q(grade=2)) 表示OR

Student.objects.filter(~Q(age__lt=20)) 表示not

排序

对查询语句进行排序
Student.objects.all().order_by(字段) 默认排序方式是升序,如果需要降序字段前面使用 ' - '

values方法

Students.objects.all()返回的是对象结果集QuerySet<QuerySet [<Student: Student object>, <Student: Student object>,]

Students.objects.all().values() 返回的是字典结果集<QuerySet [{'id': 1, 'name': '0001', 'age': 16, 'klass': '1802', 'grade': 4, 'klass_id_id': 11}, {'id': 2, 'name': '0000', 'age': 16, 'klass': '1803', 'grade': 3, 'klass_id_id': 11}]

count方法

Students.objects.查询函数().count() 返回符合条件数量

exists方法
Student.objects.exsits(查询条件) 返回True或False

Limit offset

Student.objects.all()[5:100] 类似于offset limit 但是不能使用负数

聚合函数 aggregate
Avg,Count,Max,Min,Sum

Student.objects.aggregate(AVG('age'))

objects
objects是Manager对象,由django默认提供,我们可以自定义

```
class Student(models.Model):
    name = models.CharField(max_length=32)
    age = models.IntegerField(default=16)
    klass = models.CharField(max_length=32)
    grade = models.IntegerField(default=1)
    klass_id=models.ForeignKey("Klass",null=True,)

    st_manager=models.Manager()
```

Student.st_manager.all()

自定义Manager

```
 class StudentManager(models.manager.Manager):
    def get_lower_grade(self):
        return self.get_queryset().filter(grade__lt=3).count()
```

多对多关系

作者与书

book.author.add(a1,a2,a3)
book.author.remove(a1)
book.author.查询函数
book.author.set(Author.objects.all()[2:5])替换原来所有的关联对象

author.[类名小写]_set.查询函数()

一对一
学生和课桌
desk.student=Student.objects.last()
desk.save()

访问关联对象
desk.student 查看desk是哪个学生的
student.desk 查看学生的课桌

普通字段:

IntegerField
4字节整数,范围从 -2147483648 ~ 2147483647

PositiveIntegerField:
类似于IntegerField,但是存储的都是正整数0~2147483647

PositiveSmallIntegerField:
类似于IntegerField,但是存储的都是正整数0~32767

SmallIntegerField:
类似于IntegerField,存储范围是-32768 ~ 32767

BigIntegerField
8字节整数,范围是:-9223372036854775808 ~ 9223372036854775807

AutoField:
存储的是整数,但是会自动增长,通常情况下我们不会直接使用这个字段,model当中有一个默认字段 id 就是AutoField

BigAutoField
8字节整数,类似于AutoField,但是存储的是1~9223372036854775807的整数

BinaryField
存储的是原始的二进制数据

BooleanField
赋值的时候用True或False,数据库中实际保存的是0,1

CharField

存储字符串,有个特殊的属性:max_length在使用这个类型字段时,必须指定max_length,否则会报错

URLField
继承自CharField,默认max_length=200,判断存储的字符串是否是一个URL

TextField
存储的一大段文字

DateField
存储日期,特殊属性:auto_now,每次调用save方法时都会把该字段改为当前时间,默认值的False

DateTimeField
存储日期和时间

TimeField:
存储的是时间

DecimalField
存储10进制小数,特殊的属性:decimal_places小数点位置,max_digits数字最多位数;两个属性必须都指定

DurationField
存储的是时间段,使用timedelta来赋值

EmailField
继承自CharField,会检查要存储的字符串是不是正确的email地址

FloatField
存储的是float类型数据

FileField
文件上传字段,用于保存上传的文件,两个可选属性:
upload_to,文件保存到哪里
models.FileField(upload_to='files/') 会保存到MEDIA_ROOT/files下MEDIA_ROOT配置在settings里面
storage:文件存储对象,默认是FileSystemStorage
相关的对象时FieldFile,当我们访问FileField中保存的文件时,时通过FieldFile来访问的,FieldFile的属性:

- name:保存的文件的名字
- size:文件的大小
- url:该文件的URL

ImageField
继承自FileField,他会检查文件是否是图片,他有两个属性,height_field和width_field,每次执行save函数的时候都会计算这两个值,这个字段依赖于python的Pillow图形库

GenericIPAddressField
保存的数据格式是IP地址,可以是ipv4也可以是ipv6,ipv4通常是xxx.xxx.xxx.xxx ipv6通常是xxxx:xxxx:xxxx:xxxx

UUIDField
UUID是指在一台机器上生成的数字，它保证对在同一时空中的所有机器都是唯一的
数据库中以char(32)来保存,我们可以直接使用python的UUID类来创建

关系字段

ForeignKey(m:1)

ForeignKey(*to*, *on_delete*, ***options*)

ForeignKey(default='')
to
表示与哪一个model是多对一关系,
on_delete
是当外键删除的时候,执行什么操作,支持的值为:
models.CASCADE 级联删除
models.PROTECT 删除外键是抛出异常ProtectedError
models.SET_NULL 将该字段设置为null,同时还要设置null=True
models.SET_DEFAULT 将该字段设置为默认值
models.SET(function_name) 调用一个函数,并用该函数的返回值作为字段的值
models.DO_NOTHING 什么都不做,保持原样

related_name
默认情况下是model的小写名称加上_set, student_set

to_field
默认情况下,是直接使用关联表的主键,例如默认的id字段,也可以使用其他字段,但是该字段必须是
unique=True

ManyToManyField(n:m)

ManyToManyField(*to*, ***options*)
to
表示与哪一个model是多对多关系

Django通过创建一个中间表来记录两张表数据的对应关系

OneToOneField(1:1)

字段通用属性

null

默认是False,数据库字段定义时,如果定义了null,那么在创建一行新的数据时,允许不对该字段进行赋值,日期类型,时间类型,数字类型字段当中是无法存储空字符的,如果不需要强制给该字段赋值,那么要设置null为True,这样数据会使用null存储在该字段中

blank
默认是False,表示这个字段不可为空,当我们使用CharField和TextField时,如果想要存储空字符串”,那么必须设置blank为True,否则会存储失败.

blank是在我们的django中做的判断,null是在数据库层面做的转换

choices
接受一个可迭代的对象,对象中的元素是一个元组,每一个元组包含两个元素(itemA,itemB) itemA是存储到数据库中的值,itemB是对对itemA的说明,不会存储到数据库中,可以简单理解为给人看的

```
gender_choices=[('male','男'),('female','女')]
gender=models.CharField(choices=gender_choices,max_length=1)
```

db_column
对应的数据表中的字段名称,默认是属性名

db_index
默认False,如果设置为True,那么数据库会创建这个字段的索引

default
字段的默认值

help_text
帮助信息,可以看做字段的注释,在定义属性的时候尽量写一下help_text

primary_key
是否是主键

unique
这个字段存储的值是惟一的,整个表中该列没有重复的值