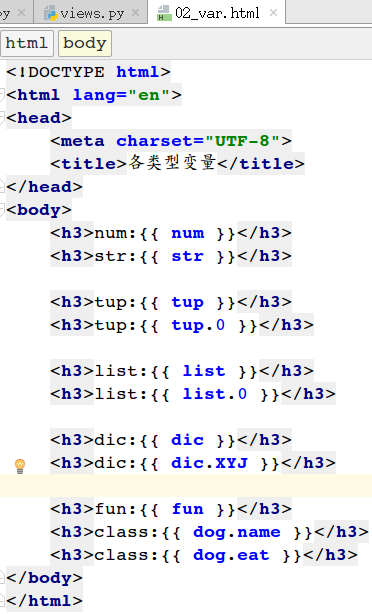
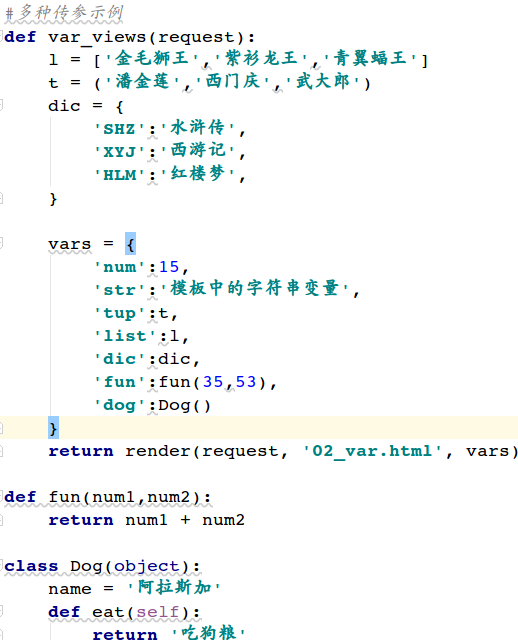
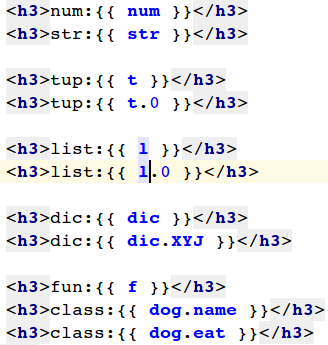
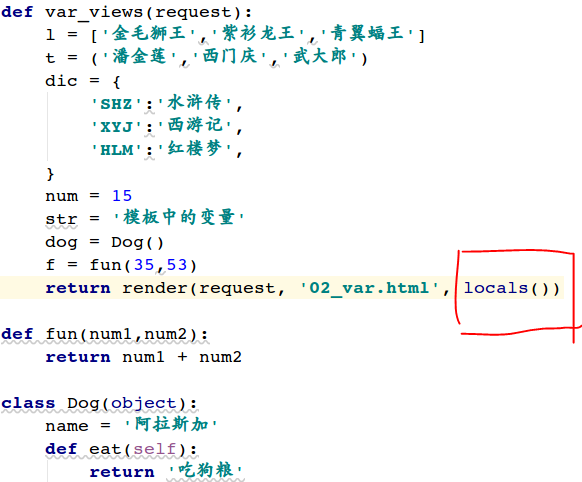
July22 Django

1 模板

1.1 变量



============================推荐写法==========================



1.2 标签

1什么是标签？

允许将服务器端的功能嵌入到模板中

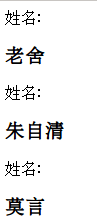
2语法

{%标签内容%}

3常用标签

for

语法：



{% for 变量 in 列表|元组|字典 %}

<h3>{{变量}}</h3>

<h3>{{变量.方法}}</h3>

<h3>{{变量.属性}}</h3>

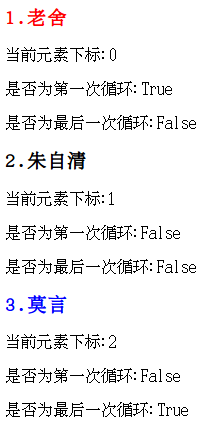
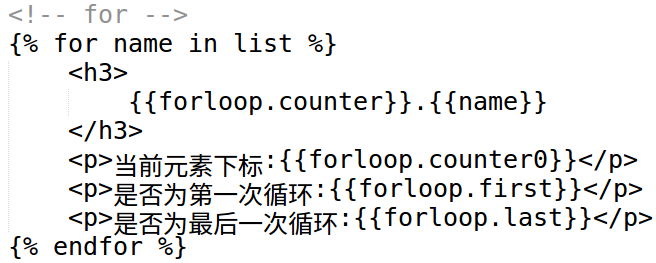
{%endfor%}



允许使用内置变量-forloop获取循环的信息

forloop在{%for中是免声明就可以直接使用的%}

|  |  |
| --- | --- |
| forloop.counter | 获取当前循环的次数 |
| forloop.counter0 | 获取当前循环的下标 |
| forloop.first | 是否为第一次循环 |
| forloop.last | 是否为最后一次循环 |



comment标签

{%comment%}

{%endcomment%}

作用：

在服务器端就被注释的内容，不会被渲染到客户端

if

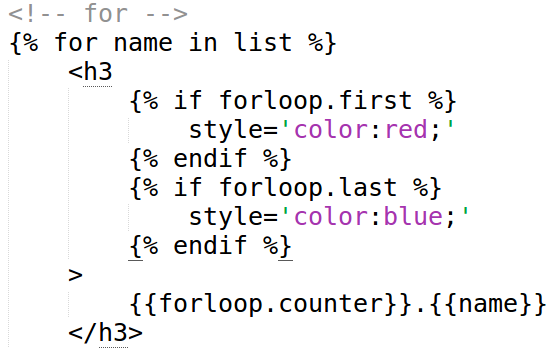
if标签

语法：

{% if 条件 %}

满足条件时要运行的内容

{% endif %}



if…else…标签

语法：

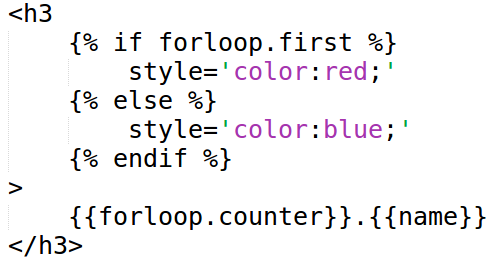
{% if 条件 %}

满足条件时要执行的操作

{% else 条件 %}

不满足条件时要执行的操作

{% endif %}



if…elif…else

语法：

{% if 条件1 %}

满足条件1时运行的内容

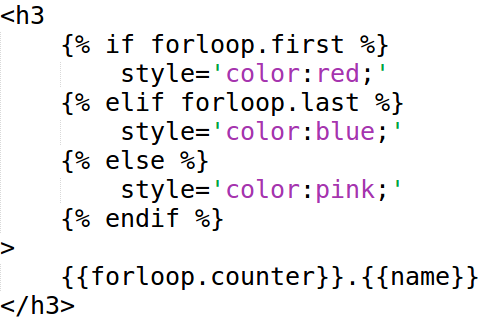
{% elif 条件2 %}

满足条件2时运行的内容

{% else %}

以上条件都不满足时运行的内容

{% endif %}



1.3 过滤器

作用：

在显示变量数据之前，允许对数据进行筛选或改变

过滤器的语法：

{{变量|过滤器}}

常用过滤器：

{{value|upper}}

将value变为大写字符输出

{{value|lower}}

将value变为小写字符输出

{{value|add:num}}

将num累加到value后

{{value|floatformat:n}}

将value四舍五入到n位小数

{{value|truncatechars:n}}

将字符串截取到n位字符（包含…）

{{}}

1.4 url( )中的name属性

url(regex,views,kwargs=None,name=None)

name:

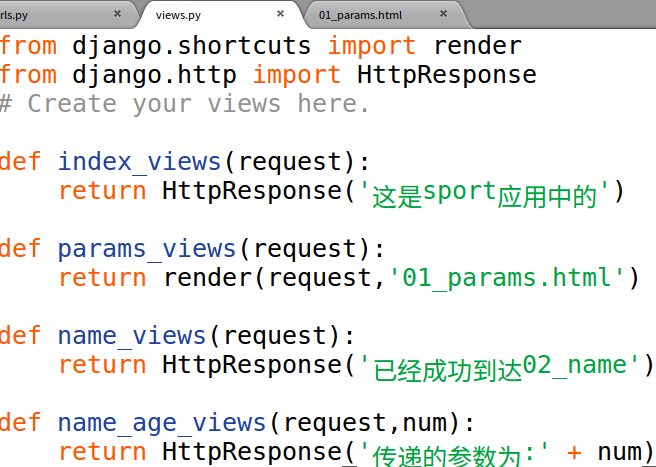
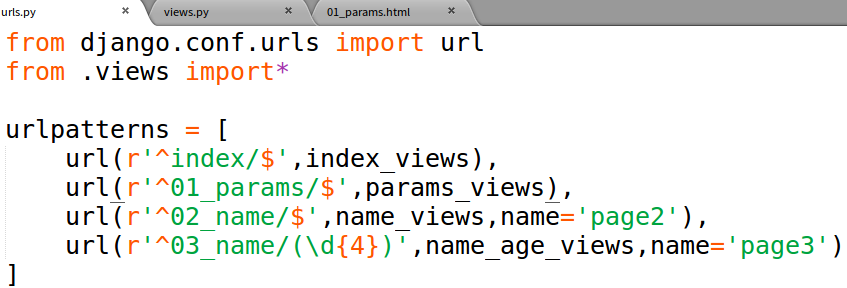
允许为当前的url设置别名，可以在Template中使用别名找到对应的url

语法：

url(regex,views,name=’别名’)

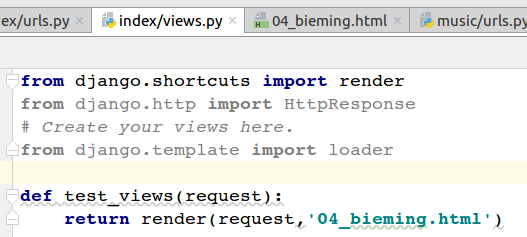
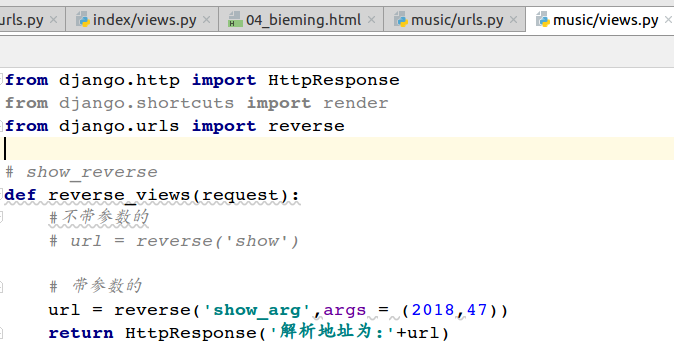
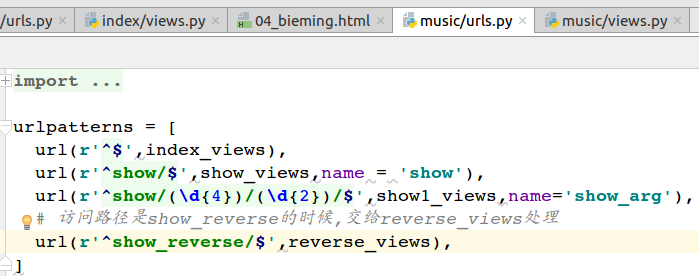
在模板中使用对应的别名找到url

{% url ‘别名’ %}



**1.4.2在视图中实现name别名反向解析**

url = reverse()



1.5 静态文件

不与用户发生动态交互的文件，称之为静态文件

如：css,js,image,audio,video,静态的html

静态文件不能用物理路径查找，需要逻辑路径

2.Django中静态文件的处理

需要在settings.py中设置静态文件的访问路径 和 存储路径

1.访问路径

STATIC\_URL=’/static/’

作用：

当访问路径是localhost:8000/static/\*\*\*

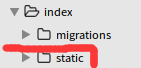
一律静态文件存储路径中搜索静态文件

2.存储路径

STATICFILES\_DIRS=(BASE\_DIR,’静态目录名称’)

在项目根目录中以及各个应用中都可以创建对应名称一个目录，来表示项目或应用中所存储静态文件的目录





3.访问静态文件

1.直接使用localhost:8000/static/\*\*\*

<img src=’http://localhost:8000/static/\*\*\*’>

<img src=’static/\*\*\*’>

2.使用{% static %}访问静态文件

1.在模板的最顶层增加

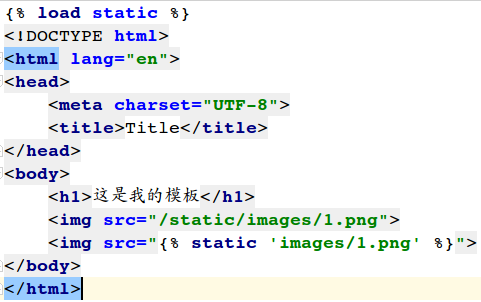
{% load static %}

2.在使用静态文件时

<img src=’{% static ‘具体路径’ %}’>

ex:

<img src=’{% static ‘images/huiyuan.jpg.. %}’>



1.6 模板的继承

当多个模板具备大部分相同的内容时，就可以使用继承的方式，将相同的内容继承过来，再增加/修改属于自己的内容即可。

**1.6.1模板继承的语法规范**

1.在父模板中

必须标识出来哪些内容在子模板中是允许被修改的

标签：

{% block 名称 %}

父模板中要显示的内容

{% endblock %}

block的作用：

1.在父模板中，正常显示

2.在子模板中，如果不修改block中的内容的话则会按照父模板中的内容进行显示



2.在子模板中

①在最顶层增加一个标签来表示继承关系

{% extends ‘父模板的名称’ %}

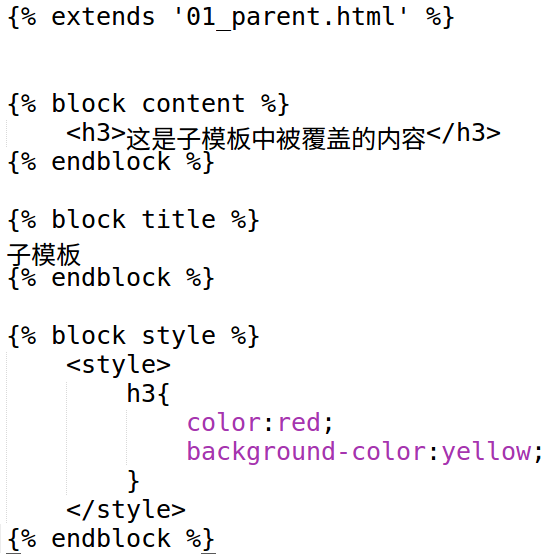
②增加block标签，改写属于自己的内容

{% block 名称 %}

子模板中的内容

{% endblock %}

在子模板中的block会覆盖父模板中的同名的block的内容



2 模型

他是根据数据库中表的结构来创建出来的class。每一张表映射到编程语言中就是一个class，表中的每一个列，到编程语言中就是class中的一个属性。在模型中还能完成对数据的CRUD操作

2.1 创建和使用模型

-ORM框架

ORM：Object Relational Mapping

简称：ORM, O/RM ,O/R Mapping

中文：对象关系映射

ORM的三大特征：

1.数据表（table）到类（class）的映射将数据表自动生成一个class类，将class类自动生成一张表；

2.数据类型的映射

允许将表中字段的数据类型自动 映射 成编程语言中的对应的数据类型

允许将编程语言中的数据类型自动 映射 成表中字段的数据类型

3.关系映射

数据库中表的关联关系：

表与表之间的关系，映射成类与类之间的关系

一对一，一对多，多对多

将表与表之间的关系也映射到编程语言中，通过创建类与类（对象与对象）之间的关系来完成

ORM的优点：

1.提高了开发效率，能够自动完成表到类的映射

2.不用SQL编码，也能够完成对数据库的CRUD操作，可以省略庞大的数据访问层

3 创建和配置数据库

3.1 创建一个数据库（支持中文）

create database 数据库名 default charset utf8 collate utf8\_general\_ci;

mysql可视化开发工具-Navicat

1.创建一个数据库 - webdb

2.在settings.py中配置数据库的信息

DATATBASES={

‘default’:{

‘ENGINE’:’…’,

‘NAME’:’…’,

}

}

连接MySQL的配置：

1.ENGINE:引擎 'ENGINE': 'django.db.backends.mysql'

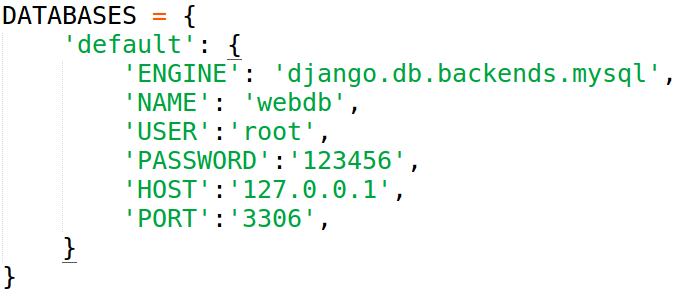
2.NAME：指定要连接到的数据库的名称

3.USER：指定用户名称，通常为root

4.PASSWORD：指定密码，通常123456

5.HOST：要连接的主机地址，本机：localhost或127.0.0.1

6.PORT：指定端口号，通常为3306



注意：

在Django中连接MySQL的话依赖于MySQLdb

解决方案：

在主目录中的\_\_init\_\_.py中

import pymysql

pymysql.install\_as\_MySQLdb()

4数据库的同步操作

./manage.py makemigrations

作用：

将每个应用下的models.py文件生成一个数据库的中间文件，并保存在migrations文件夹中

./manage.py migrate

作用：

将每个应用下的migrations文件中的中间文件同步到数据库中

5 编写Models（重点）

注意：

1.Models中的每个class都成为模型类（Models类）或实体类（Entry）

实体：就是数据表中的一条记录

实体完整性：约束表中的记录不重复

2.Models中的每个实体类，必须要继承自models.Model

示例：

在models.py中

from django.db import models

#创建Publisher模型类

#表示出版社的信息，并包含以下属性

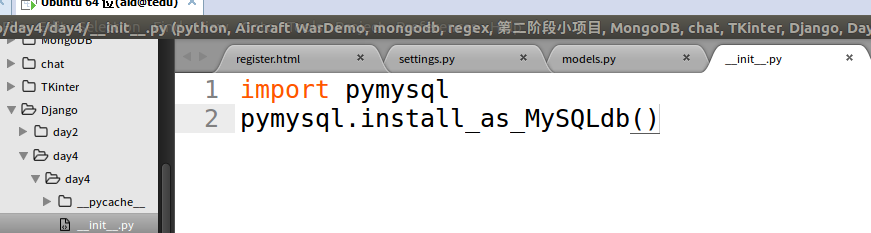
#1.name：表示出版社名称

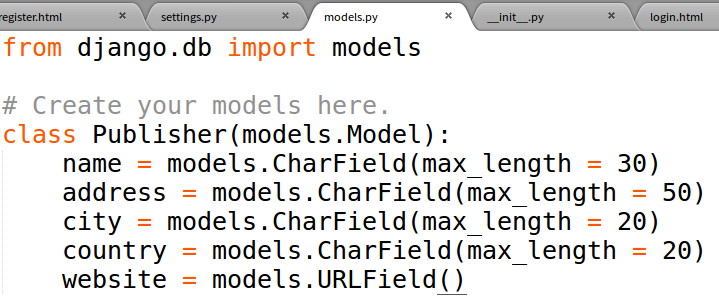
#2.address：表示出版社的地址

#3.city：出版社所在的城市名称

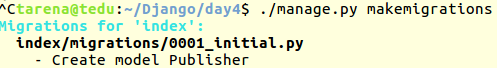
#4.country：出版社所在国家名称

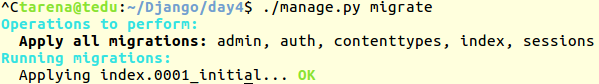
#5.websit：表示出版社的网址





生成数据库的中间文件





在mysql数据库中将应用名和表名组合成一个表名



6 Django中的数据字段 和 字段类型

语法：

属性 = models.数据字段（字段选项1，字段选项2，……）

1.字段类型（Field Type）

|  |  |
| --- | --- |
| BooleanField() | 真假 |
| CharField() | varchar |
| DateField() | 日期年月日 |
| DateTimeField() | 日期时间年月日时分秒 |
| DecimalField() | 小数 |
| EmailField() | 电子邮箱 |
| FloatField() | 小数 |
| ImageField() | 图片字段---存图片的路径--varchar |
| IntegerField() | 整数 |
| UrlField() | 网址--varchar |
| TextField() | 存大量数据--text |

models.DecimalFeid(max\_digits = 7,decimal\_places=2)

uimg = models.ImageField(upload\_to=’images/users/’)

2.字段选项（Field Options）

|  |  |
| --- | --- |
| max\_length | 设置最大长度，在models.CharField()中是必选项 |
| default | 为当前的属性指定默认值 |
| null | 指定当前属性是否允许为空，默认为False |
| unique | 指定当前属性的值是否唯一，默认为False |
| blank | 指定当前属性的值是否允许为空字符串，默认为False |

作业：

1.在day4项目中的models.py中追加两个class，并且同步到数据库中

Author-作者

name

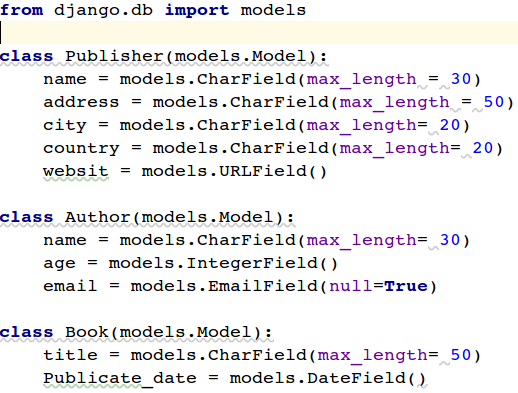
age

email

Book-图书

title-书名

publicate\_date-出版时间



2.在fruitday中，根据注册页面创建models并同步到数据库中（数据库名:fruitday）

July24 Django

1 模型

1.1 字段类型

ImageField：保存图片的路径

用法：

pic = models.ImageField(upload\_to=’static/upload/…’)

2 天天果园数据分析

商品类型Models - GoodsType

类型名称 - title

类型图片 - picture，默认上传路径:static/upload/goodstype

类型的描述 - desc

商品Models - Goods

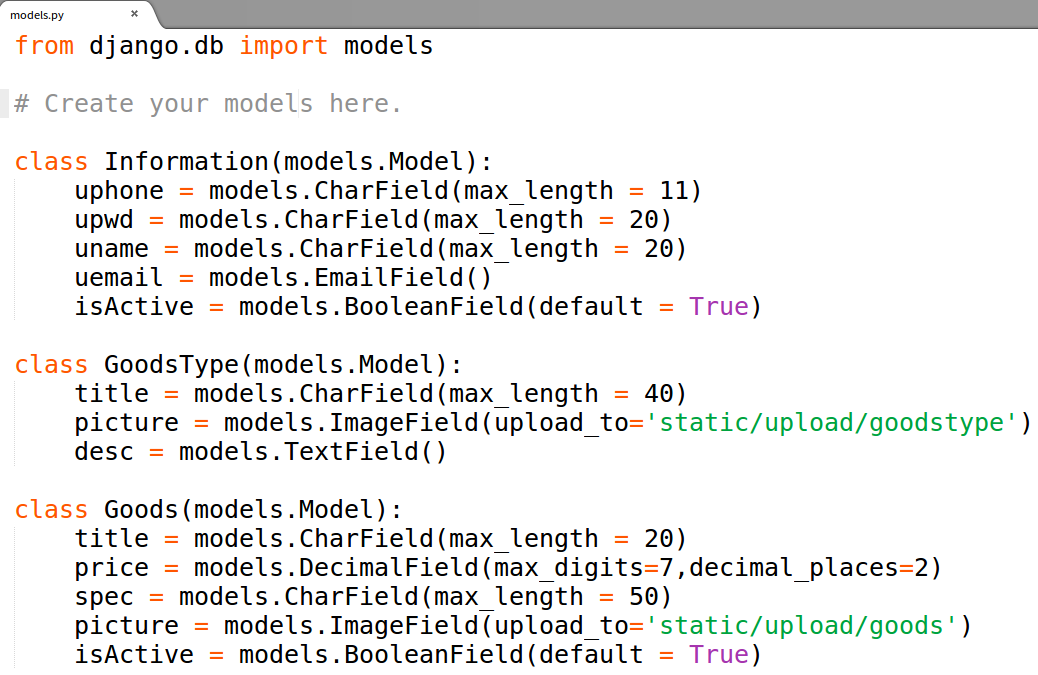
商品名称 - title

商品价格 - price(DecimalField)

商品规格 - spec(CharField)

商品图片 - picture，默认上传路径:static/upload/goods

销售状态 - isActive（默认为True）



3 模型

3.1 数据的版本切换

./manage.py migrate执行所有应用中最新版本的数据库中间文件

./manage.py migrate应用名称 版本号

./manage.py migrate index 0002

执行index应用中版本号为0002的中间文件

3.3 通过数据库自动导出Models

./manage.py inspectdb > 文件名.py



4 模型中的CRUD

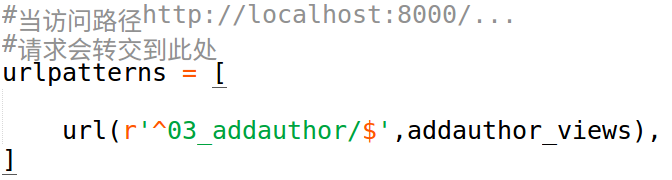
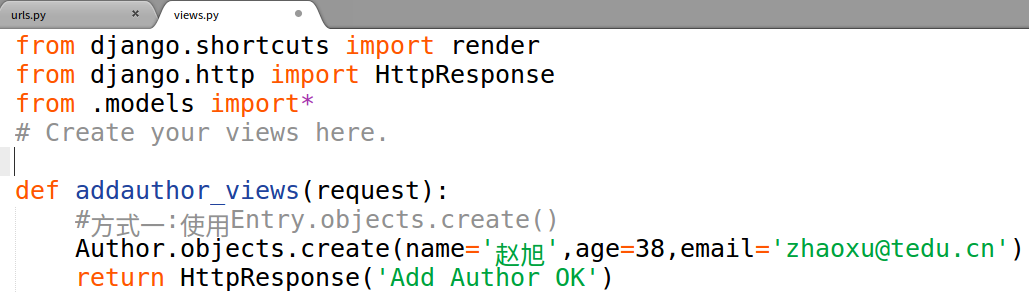
4.1 通过ORM向DB中增加数据

方法一：

Entry.objects.create(属性=值，属性=值，…)

ex：

Author.objects.create(name=’zsf’,age=85,email=’zsf@163.com’)

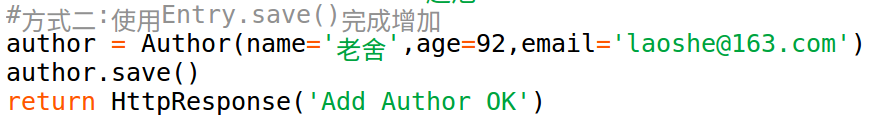


方法二：

创建一个Models对象，并通过save()完成增加

obj=Entry(属性=值，属性=值)

obj.save()



方式三：

使用字典构建对象

dic={

‘属性1’:’值1’,

‘属性2’:’值2’,

……

}

obj=Entry(\*\*dic)

obj.save()



练习：

使用三种方式，分别向index\_book,index\_publisher中，各增加三条数据

4.2 查询操作（重难点）

通过Entry.objects调用查询接口

**4.2.1基本的查询操作**

语法：

all()

用法：

Entry.objects.all()

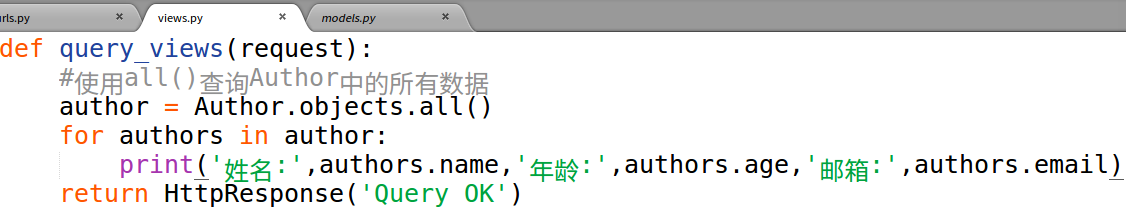
返回：

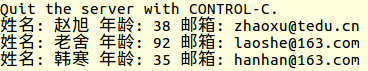
QuerySet（封装了若干对象的列表）

ex:

author = Author.objects.all();

等同于：select \* from [index\_author]





**4.2.2查询指定列的操作**

语法：

values(‘列1’,’列2’)

用法：

Entry.objects.values(‘列1‘，’列2’)

返回：

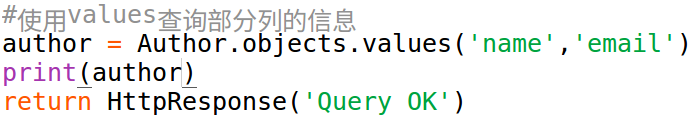
QuerySet(封装了若干字典的列表)

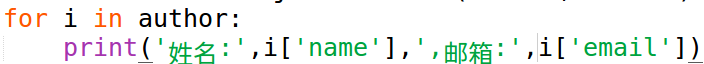
注意：

values()可以用在所有返回查询结果集的方法的后面的

Entry.objects.all().values()









语法：

values\_list()

用法：

Entry.objects.values\_list()

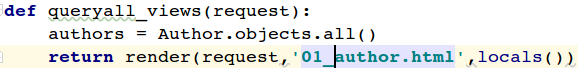
返回：

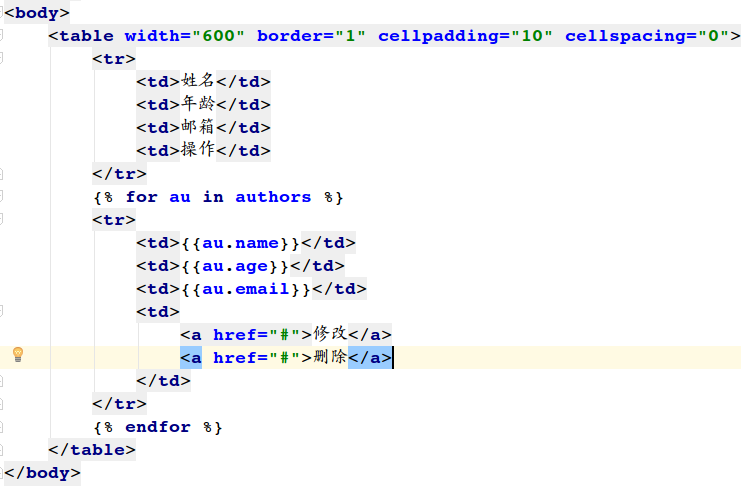
QuerySet，是一个由元组所组成的列表

作业

从数据库取出数据显示在网页





**4.2.3指定排序效果**

语法：

order\_by(‘列1’，’列2’，......)

默认升序，降序在列前加”-”号

ex:

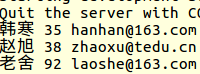
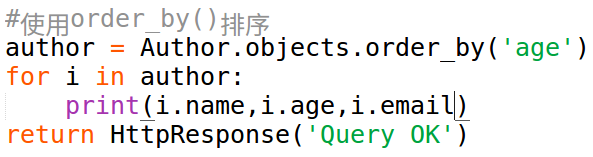
Author.objects.order\_by(‘age’)

Author.objects.all().order\_by(‘age’)

注意：

order\_by()可以用在所有返回查询结果集的方法后面





**4.2.4对条件取反**

语法：

exclude（条件）

用法：

Entry.object.exclude(属性=值，属性=值)

ex:

Author.objects.exclude(id=3)->select\*from author where not(id=3)

Author.objects.exclude(id=3,age=85)->select\*from author where not

(id=3 and age=85)

**4.2.5根据条件查询部分行数据（重难点）**

语法：

filter(条件)

用法：

Entry.objects.filter(条件)

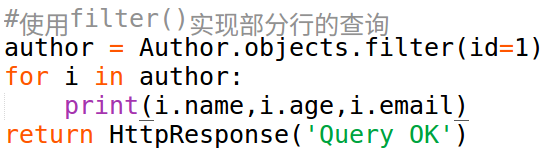
1.直接使用Entry中的属性作为查询条件

如果有多个查询条件使用” , ”隔开，使用and进行关联的

Author.objects.filter(id=1)->select\*from author where id=1;

Author.objects.filter(id=1,name=’韩寒’)->select\*from author where

id=1 and name=’韩寒’





可以查询空数据或多条数据，不报错

2.通过Field Lookups完成复杂条件查询

Field Lookup：查询表达式（查询谓词）

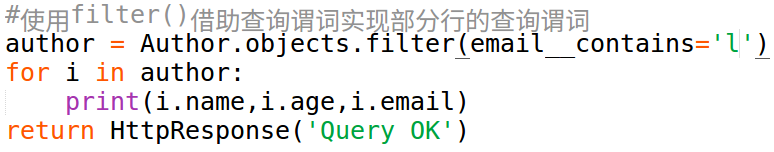
每一个查询谓词都是一个独立的查询条件，可以用在所有有查询条件的位置处

ex:

Author.objects.filter(id\_\_exact=1)

Author.objects.filter(id=1)

|  |  |  |
| --- | --- | --- |
| \_\_exact | 等值判断 | Entry.objects.filter(属性\_\_exact=值) |
| \_\_contains | 判断属性中是否包含指定关键字 | Entry.objects.filter(属性\_\_contains=值) |
| \_\_lt | 判断属性值是否小于指定的值 | Entry.objects.filter(属性\_\_lt=值) |
| \_\_lte | 判断属性值是否小于等于指定的值 |  |
| \_\_gt | 判断属性值是否大于指定的值 |  |
| \_\_gte | 判断属性是否大于等于指定的值 |  |
| \_\_startswith | 筛选属性值是以指定数值开头的 |  |
| \_\_endswith | 筛选属性值是以指定数值结尾的 |  |

**4.2.6查询只返回一条记录**

语法：

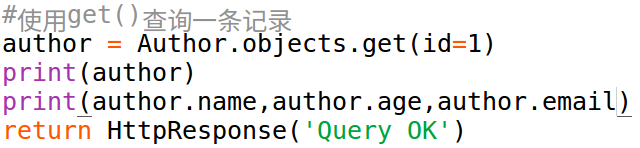
get()

用法：

Entry.objects.get(查询条件/谓词)

注意：

该函数只能用在查询返回一条记录时使用，如果返回多条数据，或者不返回数据时都会报错





**4.2.7查询结果集中记录数的**

语法：

count()

用法：

Entry.objects.count()

Entry.objects.filter(条件).count()

July25 Django

1 修改数据

1.1 修改单个数据

1查2改3保存

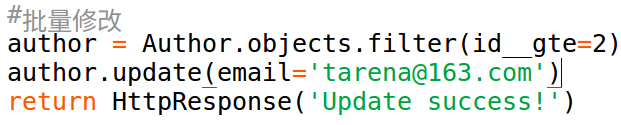
通过get()得到要修改的数据，再通过实体对象的属性修改属性值，最后通过实体对象的save()方法保存回数据库



1.2 批量修改数据

调用QuerySet的update()即可

Author.objects.filter(id\_\_gt=1).update(属性=值，属性=值)



2 删除数据

调用实体对象/查询结果集的delete()即可

1.1 删除单个对象

obj = Author.objects.get(id=1)

obj.delete()

1.2 删除多个对象

Author.objects.all().delete()

练习：

1.Author实体中增加一个列，isActive(BooleanField),默认True

2.删除：将对应author的isActive更改为False

3.查询：只查询isActive为True的信息，其余不查询

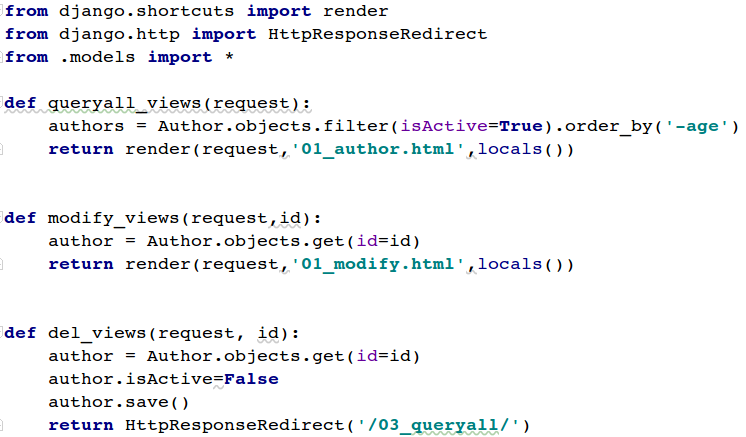
3 模型转发&重定向

3.1 转发

在一个视图中调用了另外一个视图，再将结果响应给浏览器

特点：

浏览器地址栏的地址始终是一个请求地址





3.2 重定向（重要）

重新向新的访问地址发送一个请求（由服务器端通知客户端）

语法：

from django.http import HttpResponseRedirect

return HttpRespinseRedirect(‘重定向地址’)

表现：

地址栏会实现最后一次请求地址

原因：

重定向导致浏览器向服务器发送了两次请求

4 ORM操作

4.1 F操作和Q操作

F()

更新Author表中数据，所有人的年龄+10岁

update Author set age=age+10

Author.objects.all().update(age=100) ××××××错误的写法

作用：

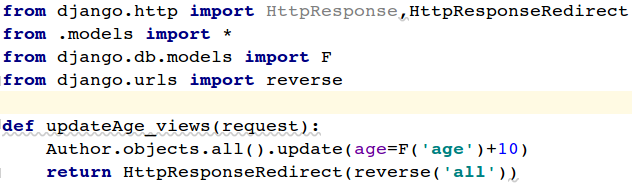
用于在执行中获取某列的值

语法：

F(‘列名’)

from django.db.models import F

Author.objects.all().update(age=F(‘age’)+10)



tip：使用反向解析地址



Q()

Author.objects.filter(id=1,age\_\_gte=50)

select \* from Author where id=1 and age>=50

作用：

在查询条件中，可以完成 或(or) 的操作

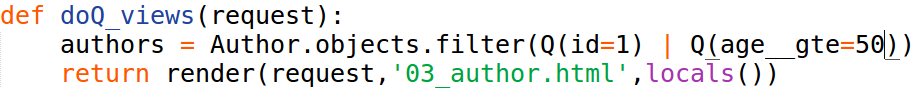
语法：

from django.db.models import Q

Q(条件1)|Q(条件2)

Author.objects.filter(Q(id=1)|Q(age\_\_gte=50))

select \* from Author where id=1 or age>=50



4.2 原生数据库操作方法

**4.2.1查询**

函数：

raw(sql语句)

语法：

Entry.objects.raw(sql语句)

返回：

QuerySet

**4.2.2增删改**

def doSQL(request):

with connection.cursor() as cursor:

sql = ‘delete from Author’

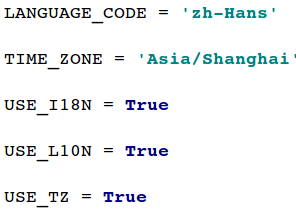
cursor.execute(sql)

return ……

5 使用后台管理Models

5.1 后台的配置

登录地址：http://localhost:8000/admin



创建后台管理员

./manage.py createsuperuser

Username:输入用户名，默认为tarena

Email Address:电子邮件

Password:密码

Password(again):重复密码

5.2 基本管理

1.在应用中的admin.py中注册要管理的Models

作用：

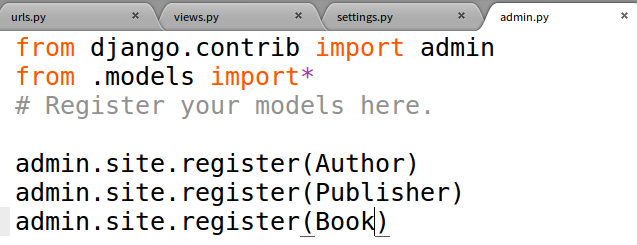
注册需要管理的Models，只有再次注册了的Models才允许被后台管理

2.注册Models

from.models import\*

admin.site.register(Entry1)

admin.site.register(Entry2)

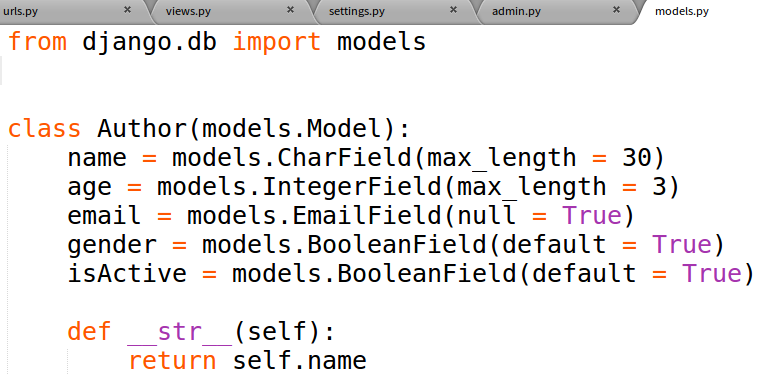


5.3 修改models.py处理显示的内容

1.在models.py中的各个class追加

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

return self.xxx



2.通过Models类的内部类Meta定义更多的展现形式

class Author(models.Model)

… …

… …

… …

class Meta:

1.db\_table

指定该实体类对应到表的名字

取值：字符串

注意：需要同步数据库才能生效

2.verbose\_name

定义该实体类在后台中的显示的名字(单数)

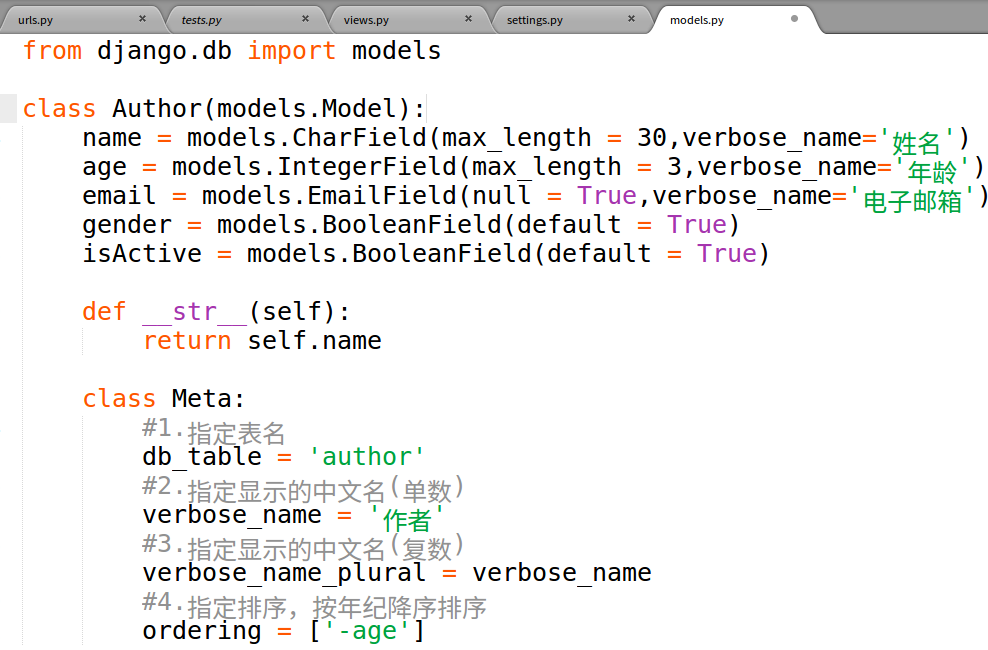
3.verbose\_name\_plural

效果同上(复数)

4.ordering

显示数据的排序规则

取值：一个列表，指定排序属性，默认是升序，如果需要降序的话属性名前加“-”号



同步数据库，修改库名由index\_author改为author

5.4 高级管理

1.在admin.py中创建高级管理类

1.定义高级管理类---EntyAdmin

需要继承自admin.ModelAdmin

ex:

class AuthorAdmin(admin.ModelAdmin):

pass

2.注册高级管理类

admin.site.register(Entry,EntryAdmin)

ex:

admin.site.register(Author,AuthorAdmin)

2.允许在EntryAdmin增加的属性（高级管理属性）

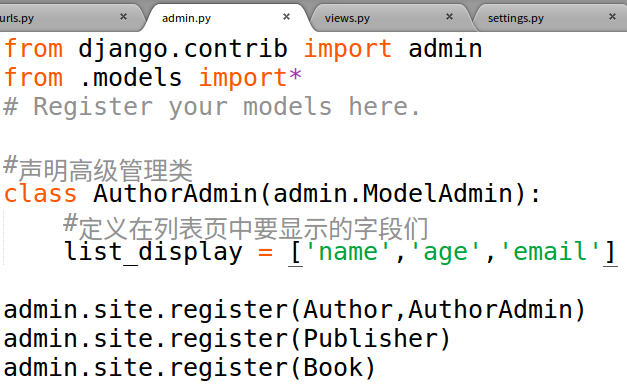
1.list\_display

作用：指定在列表页中显示的字段们

取值：由属性名称组成的列表或元组

ex：

list\_display=(‘name’,’age’,’email’)





2.list\_display\_links

作用：定义在列表页中也能够连接到详情页中的字段们

取值：有属性名组成的列表或元组

注意：取值必须要出现在list\_display中



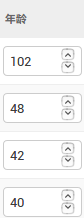
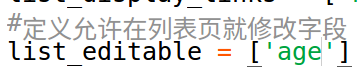


3.list\_editable

作用：指定在列表页中就允许修改的字段们

取值：由属性名组成的列表或元组

注意：取值不能出现在list\_display\_link中



4.search\_fields

作用：指定在列表页中允许被搜索的字段们

取值：由属性名组成的列表或元组



5.list\_filter

作用：在列表页中右侧增加一个过滤器，实现快速筛选

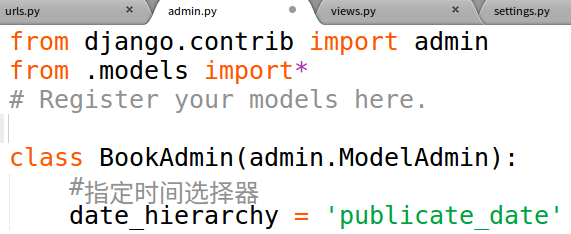
取值：由属性名组成的列表或元组



6.date\_hierarchy

作用：在列表页的顶部增加一个时间选择器

取值：必须是DateField或DateTimeFiled的列



7.fields

作用：在详情页面中，指定显示哪些字段并按照什么样的顺序显示

取值：由属性名组成的列表或元组



8.fieldsets

作用：在详情页中，对字段进行分组显示

注意：fieldsets和field是不能共存的

语法：

fieldsets = (

#分组1

(

‘分组名称’,{

‘fields’:(‘属性1’,’属性2’),

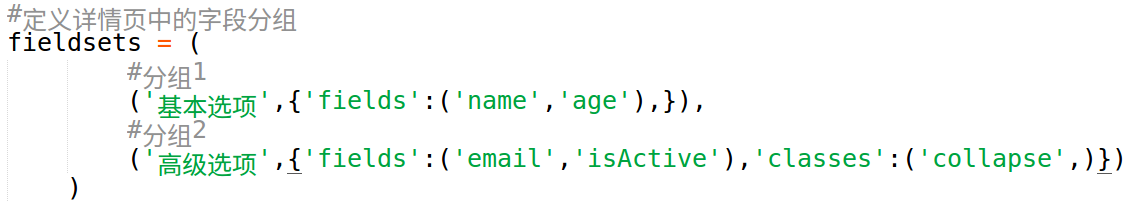
}

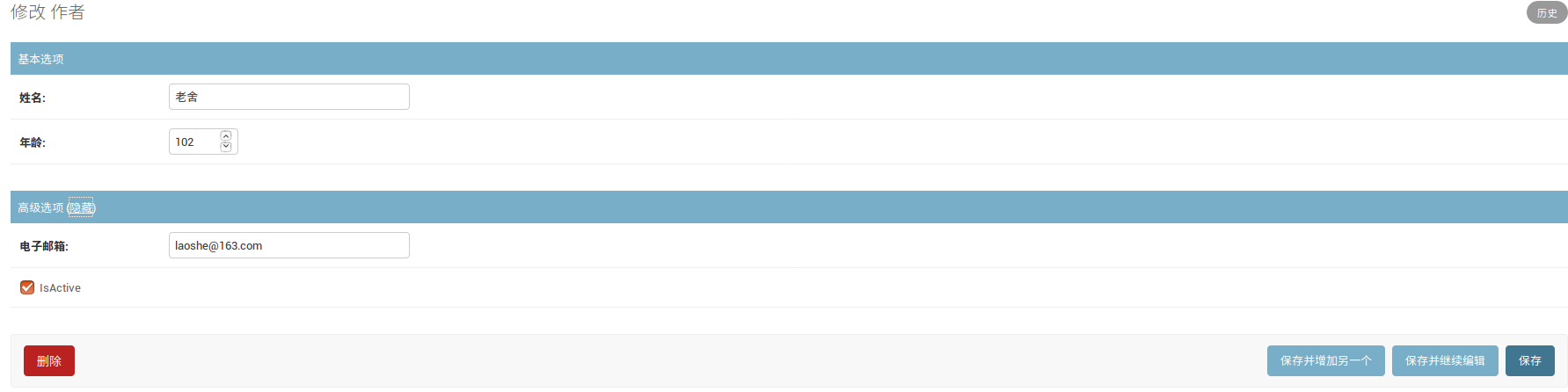
),

#分组2

(),

)





为Publisher增加管理功能

1.基本功能：指定每个列显示的名称；指定表名publish；指定每个实体对象在列表页中所显示的中文名

2.高级功能：在列表页中显示name，address，city属性值；address和city可以编辑；右侧显示过滤器，允许city和country筛选；分组显示

July26 Django

1 关系映射

1.1 一对一映射

A表中的一条记录只能与B表中的一条记录相关联

数据库中：

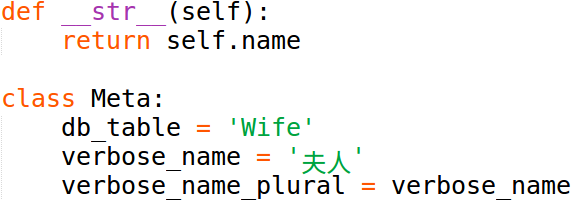
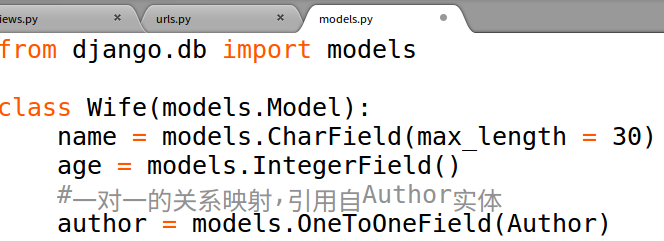
A表：设置主键

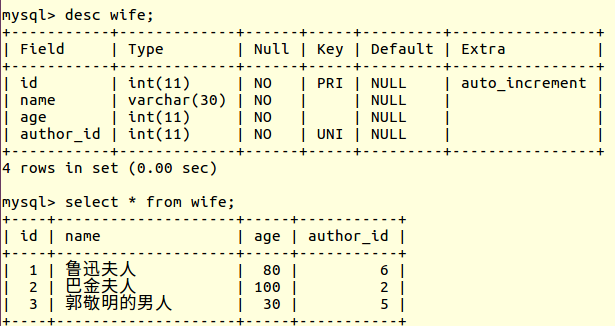
B表：增加一列，并引用自A表的主键，再增加一个唯一约束

语法：

在关联的两个类的任何一个类中实现下列操作：

属性=models.OneToOneField(Entry)





查询：

class Wife(models.Model):

…

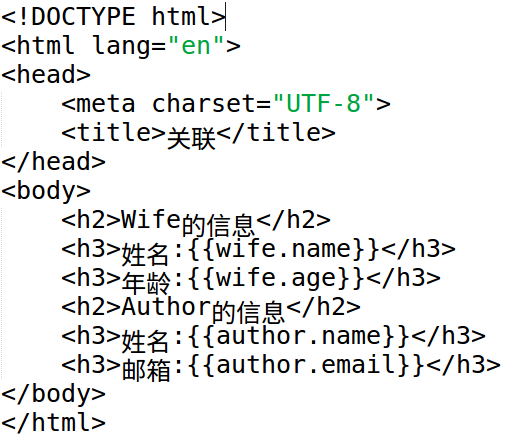
author = models.OneToOneField(Author)

正向查询：

通过wife找author

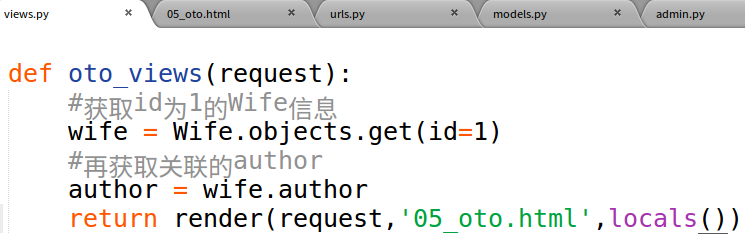
#获取id为1的wife信息

wife = Wife.objects.get(id=1)



#再通过wife找到对应的author

author = wife.author



反向查询：

通过author找wife

#获取id为3的author的信息

author = Author.objects.get(id=3)

#再通过author找到对应的wife

wife = author.wife

wife是由Django通过OneToOneField在Author中默认增加的一个隐式属性

1.2 一对多映射

A表中的一条数据可以与B表中的任意多条数据关联

B表中的一条数据只能与A表中的一条数据关联

1.在数据库中的实现

通过外键（Foreign Key）来实现一对多

在“多”表中增加外键（Foreign Key）对“一”表的主键进行引用

2.语法：

使用外键（Foreign Key）在“多”的实体中增加：

属性 = models.ForeignKey(Entry)

ex:

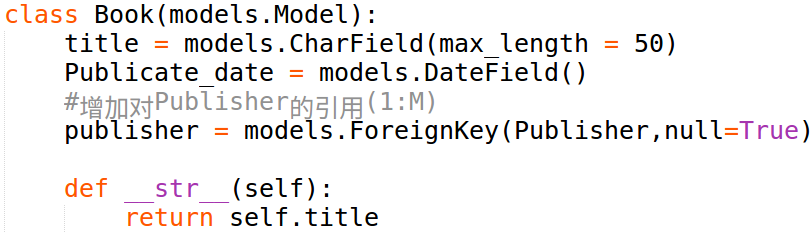
Book(多) 和 Publisher(一)

class Book(models.Model)

…

…

publisher = models.ForeignKey(Publisher)



3.查询

Book(多) 和 Publisher(一)

Book(多) 和 Publisher(一)

class Book(models.Model)

…

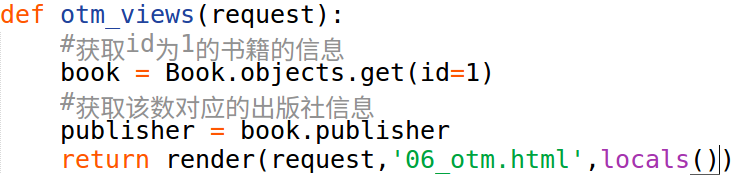
…

publisher = models.ForeignKey(Publisher)

1.正向查询---通过Book查询Publisher

book = Book.objects.get(id=1)

publisher = book.publisher



2.反向查询---通过Publisher查询Book

Django默认会在Publisher中增加一个隐式属性book\_set()

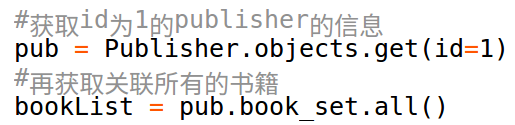
多表类名全小写\_set

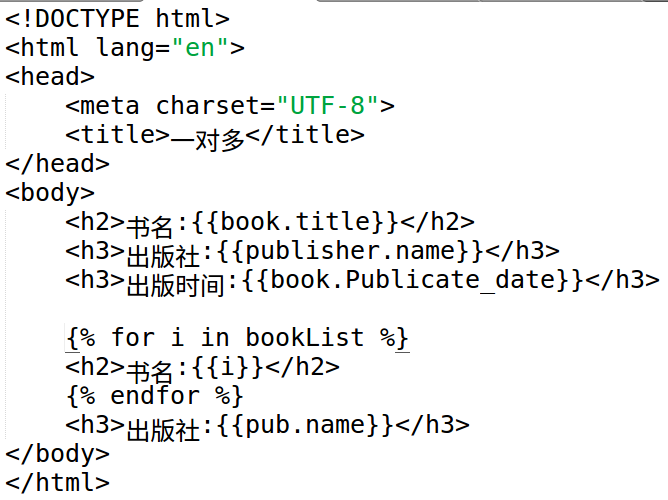
#查询id为1的出版社的信息

pub = Publisher.objects.get(id=1)

#查询出版社所关联的所有书籍

bookList = pub.book\_set.filter(条件)





1.3 多对多映射

A表中的一条记录可以与B表中的任意多条记录匹配

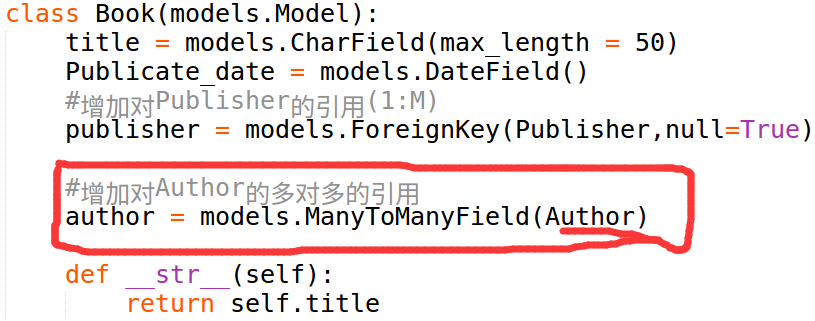
B表中的一条记录可以与A表中的任意多条记录匹配

1.在数据库中的体现

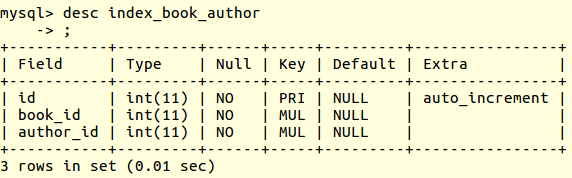
2语法：

在任何一个实体类中，增加：

entry = models.ManyToManyField(Entry)



会在数据库中自动创建一个新的id空表

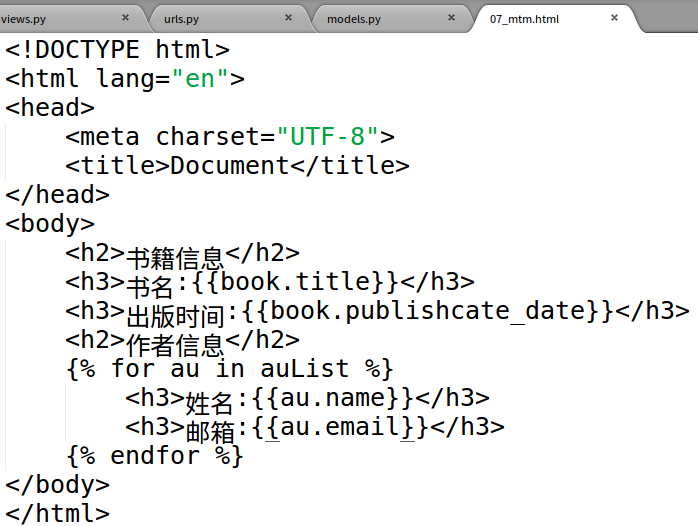


正向查询：

通过author查询所有的book

通过 关联属性.查询方法() 查询关联数据





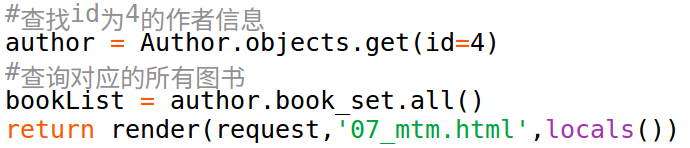
反向查询：

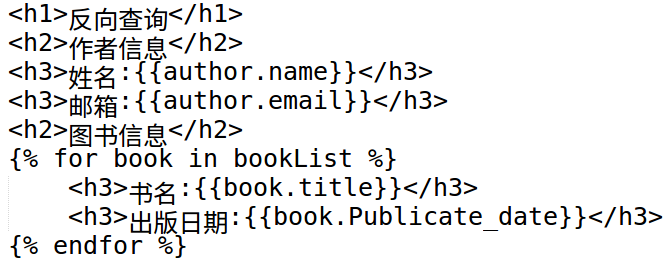
通过book查询所有的author

#django会在Book中增加一个隐式属性author\_set(对应实体的全小写\_set)

book = Book.objects.get(id=1)

auList = book.author\_set.all()





2 自定义查询对象-objects

2.1 声明

声明类EntryManager，继承自models.Manager

允许在EntryManager中添加自定义函数

class EntryManager(models.Manager):

def 函数名(self,自定义参数列表)

… …

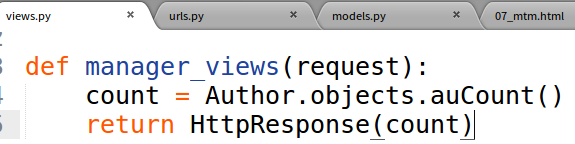
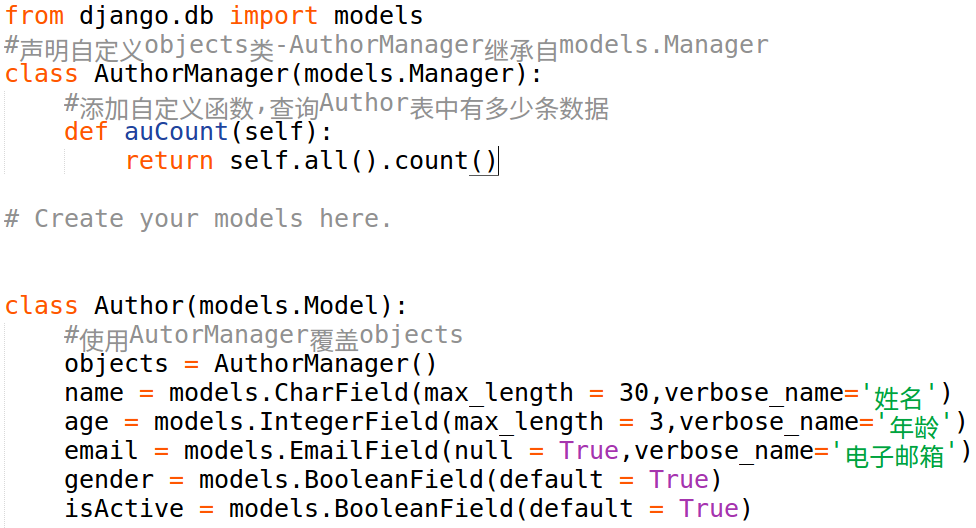
return … …

2.2 覆盖

使用EntryManager覆盖Models类中的objects

class Entry(models.Model):

objects = EntryManager()



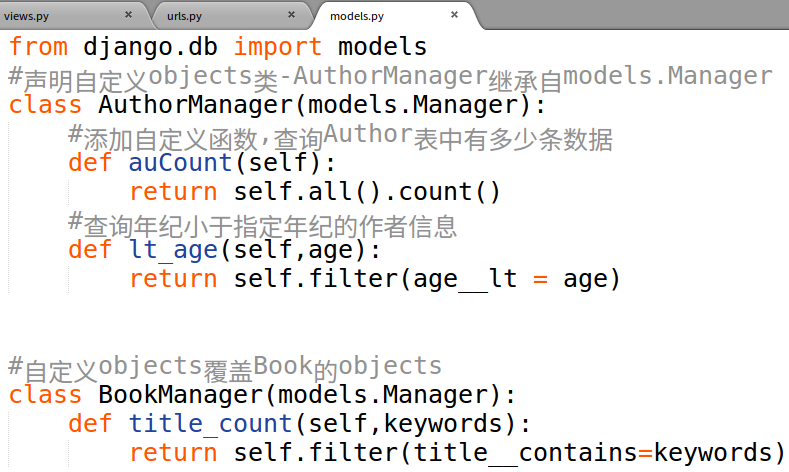
练习：

1.扩展Author的objects

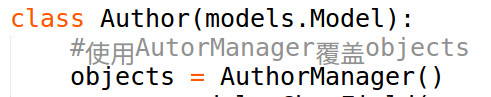
查询年级小于指定年纪的所有作者的信息

2.扩展Book的objects

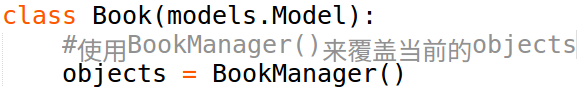
查询书名中包含指定关键字的书籍的数量



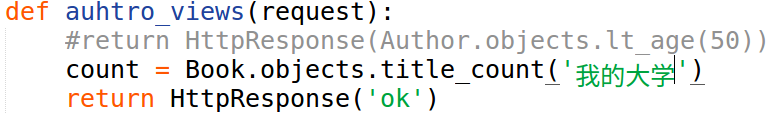
---------------------------------------------------------------------------------------------------------------



---------------------------------------------------------------------------------------------------------------



---------------------------------------------------------------------------------------------------------------



3 请求对象-HttpResquest

3.1 HTTP协议

响应和请求

3.2 请求

请求主体：

post和put两种提交方式会产生请求主体

相应消息：

由响应起始行，响应消息头，响应主体

3.3 HttpRequest介绍

HttpRequest，在django中就是请求对象的封装体现。封装了请求过程中所有信息。在django中，HttpRequest被化身了request封装到了视图处理函数中作为参数，在调用视图时自动传入

1.HttpRequest中的主要内容

|  |  |
| --- | --- |
| request.scheme | 请求协议 |
| request.body | 请求主体 |
| request.path | 请求路径 |
| request.get\_host() | 请求的主机地址/域名 |
| request.method | 请求方法 |
| request.GET | 请求方法所提交的数据 |
| request.POST | POST方法所提交的数据 |
| request.COOKIES | cookies中的数据 |
| request.META | 封装了请求的元数据 |

['COOKIES', 'FILES', 'GET', 'META', 'POST', '\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', '\_encoding', '\_get\_post', '\_get\_raw\_host', '\_get\_scheme', '\_initialize\_handlers', '\_load\_post\_and\_files', '\_mark\_post\_parse\_error', '\_messages', '\_post\_parse\_error', '\_read\_started', '\_set\_post', '\_stream', '\_upload\_handlers', 'body', 'build\_absolute\_uri', 'close', 'content\_params', 'content\_type', 'csrf\_processing\_done', 'encoding', 'environ', 'get\_full\_path', 'get\_host', 'get\_port', 'get\_raw\_uri', 'get\_signed\_cookie', 'is\_ajax', 'is\_secure', 'method', 'parse\_file\_upload', 'path', 'path\_info', 'read', 'readline', 'readlines', 'resolver\_match', 'scheme', 'session', 'upload\_handlers', 'user', 'xreadlines']

July29 Django

1 获取get请求提交的数据

1.1 form表单提交

<form method = ‘get’ action= ‘{%url ‘xxx’ %}’>

<p>

姓名:<input type = ‘text’ name = ‘uname’>

</p>

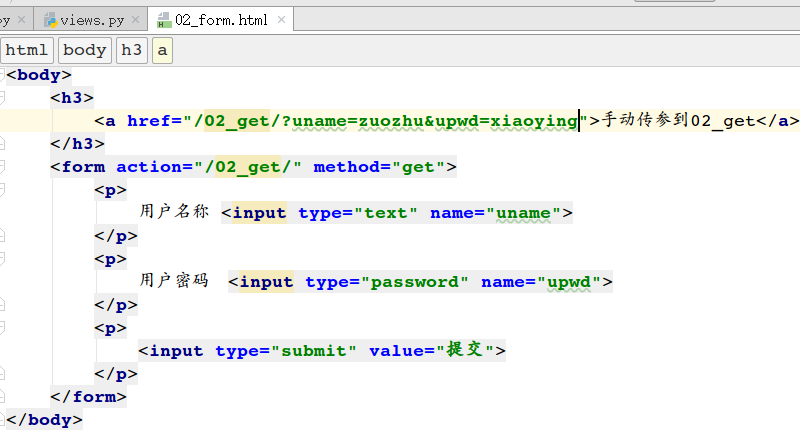
<p>

<input type=’submit’>

</p>

</form>

服务器端:request.GET[‘uname’]











1.2 通过地址发送请求

在地址栏上拼查询字符串(Query String)

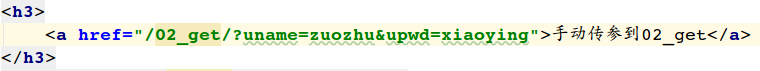
http://请求地址/?参数名=值&参数名=值

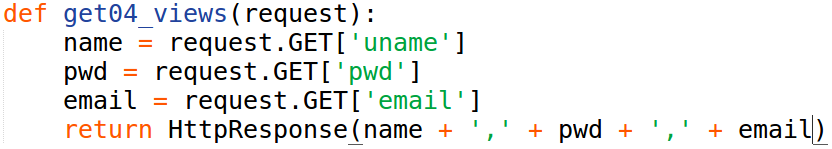
http://127.0.0.1:8000/get?name=a&pwd=b&email=c

参数1:name,值:a

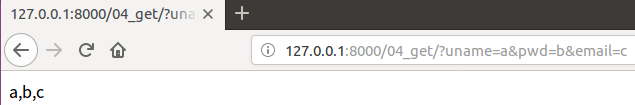
参数2pwd,值:b

参数3: email,值:c









1.发送post的方式

只有在表单的method指定为post的时候，才能发送post请求

2.获取post请求提交的数据

value = request.POST[‘key’]

3.403-Forbidden-csrf

①csrf

Cross-Site Request Forgery

理解为：跨站点攻击

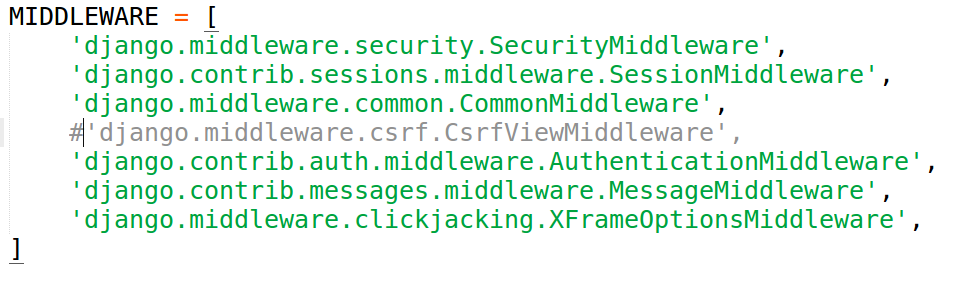
②解决方案

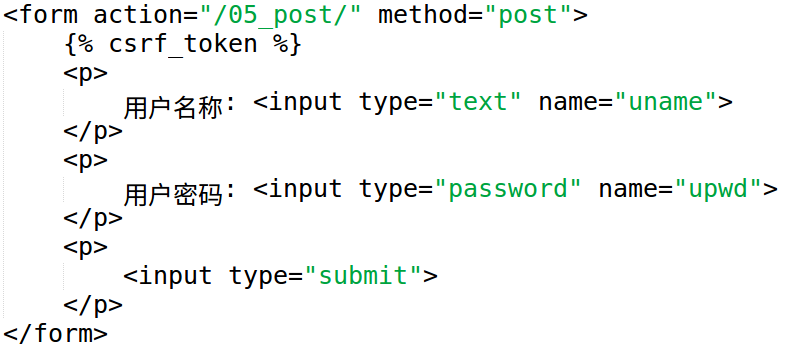
1.settings.py

删除CsrfViewMiddleware中间件，取消跨站点攻击验证

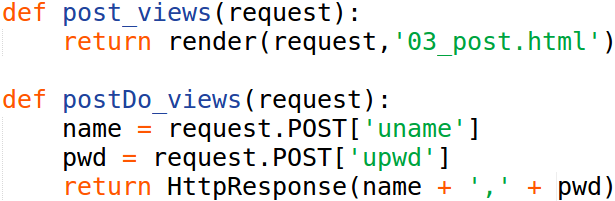
2.在处理视图上增加@csrf\_protect装饰器，开放验证权限

3.在模板中的<form>下的第一句话增加{% csrf\_token %}

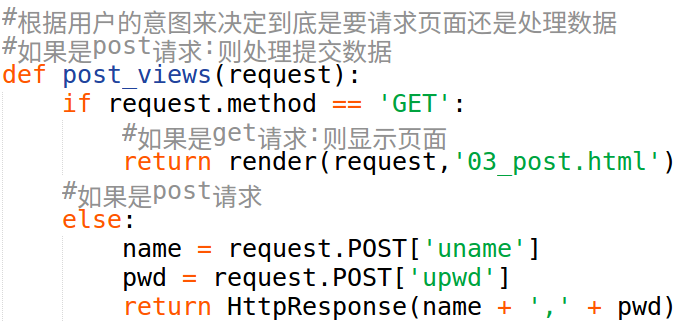


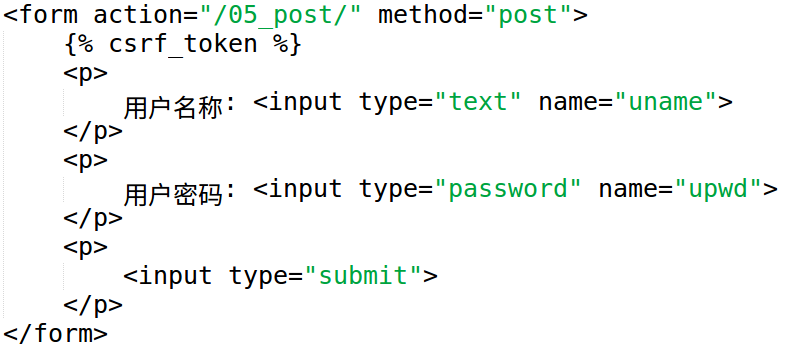






=======================修改=================================







2 使用forms处理表单

2.1 forms模块作用

通过form模块，允许将表单控件与py文件相结合，可以在网页中自动生成表单控件

2.2 使用forms模块

1.在应用中创建forms.py文件

2.导入forms

from django import forms

3.创建一个class表单，一个class对应到页面上的一个表单

该类必须继承自forms.Form

4.在class中创建属性

一个属性对应一个表单控件

3.在模板中解析forms对象

2.3 在模板中解析forms对象

注意：

需要自定义<form>

需要自定义提交按钮

1.需要自定义form

1.手动解析

ex：

form = RemarkForm()

return

render(request,’xx.html’,locals())

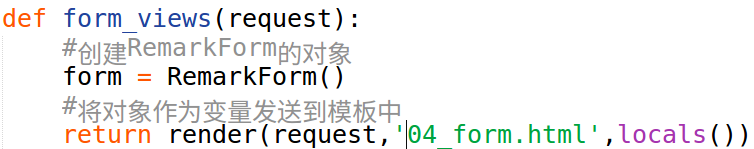
模板中：

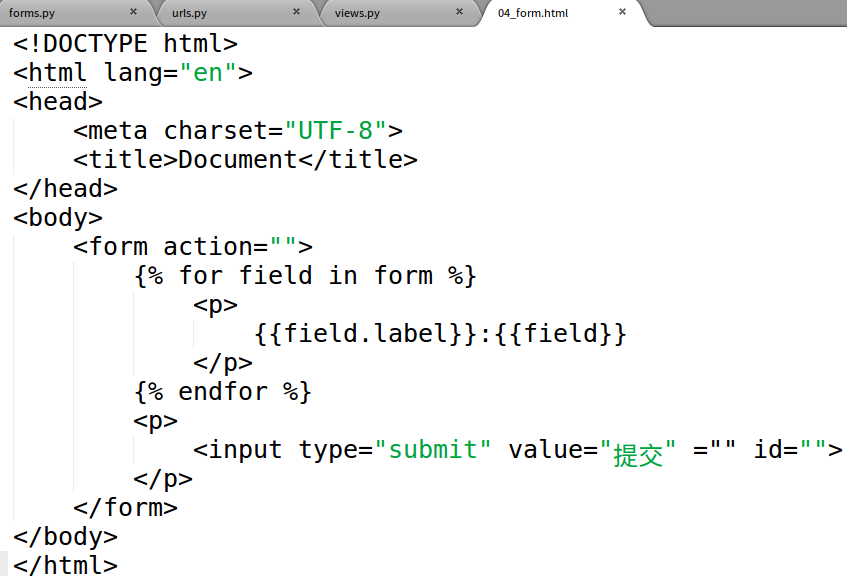
{% for field in form%}

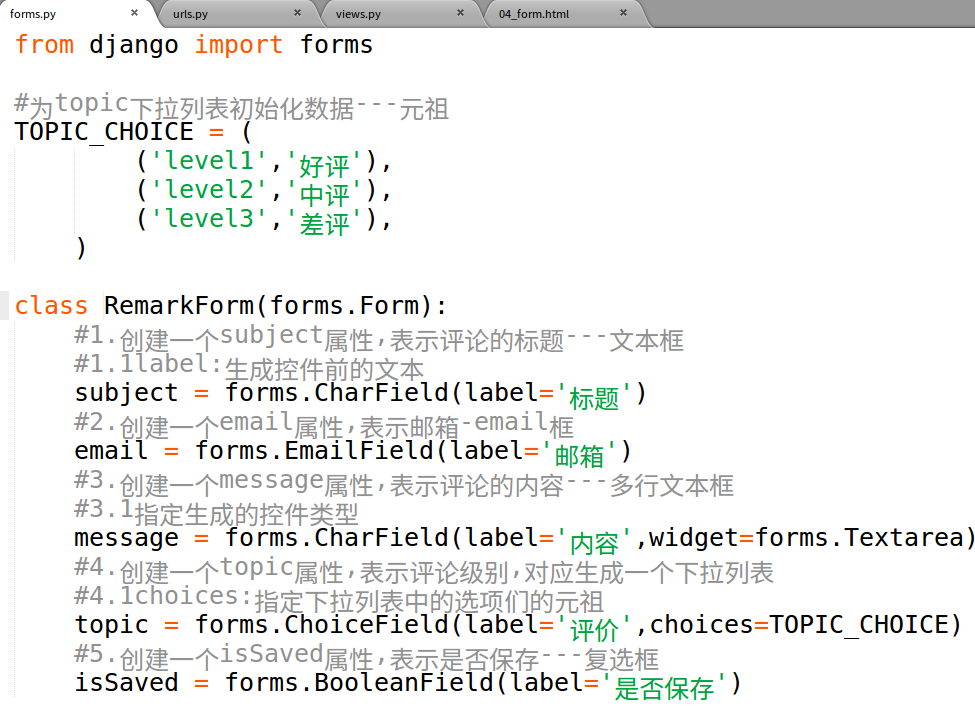
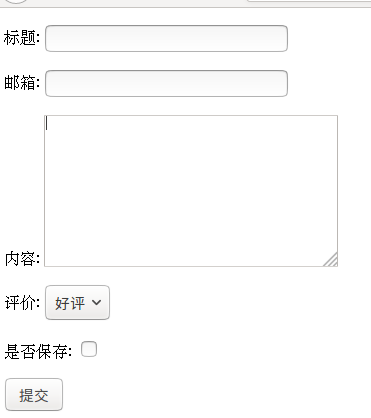
{{filed.label}}:表示的是label属性值

{{field}}:表示控件

{% endfor %}

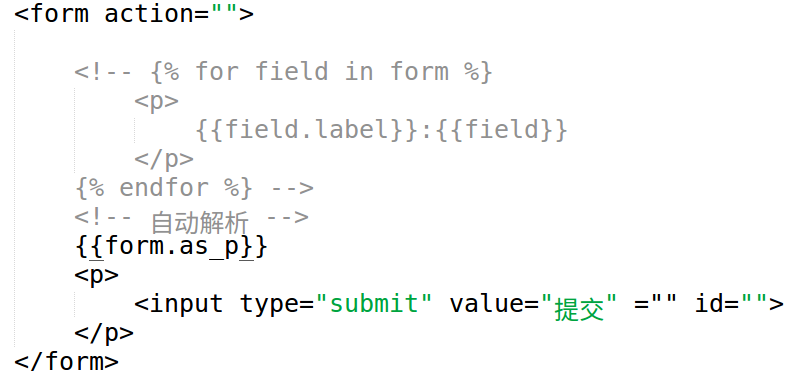


2.自动解析

①{{form.as\_p}}

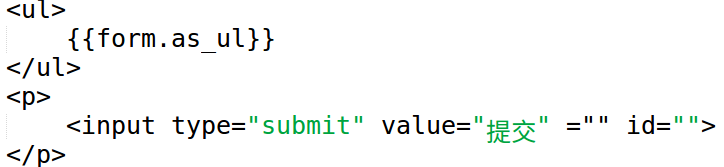
将form中每个属性都解析成一个p标记，在p标记中再显示控件以及标签



②{{form.as\_ul}}

将form中每个属性解析成一个li标记，再显示控件以及标签

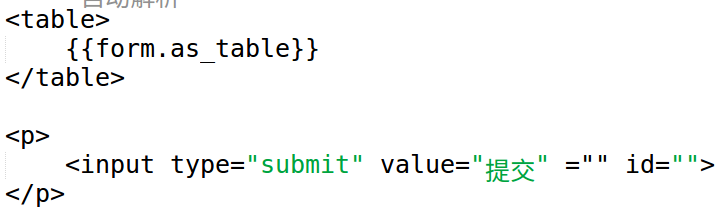
注意：需要自定义ul或ol标记的



③{{as\_table}}

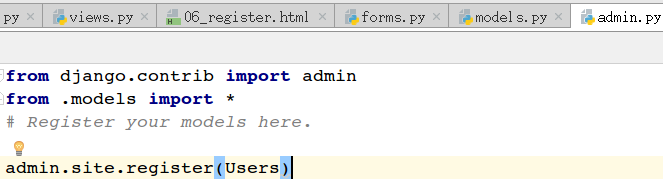
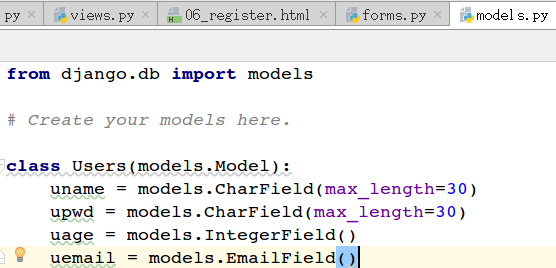
将form中每个属性解析成一个tr以及td，再显示控件以及标签

注意：需要自定义table标记



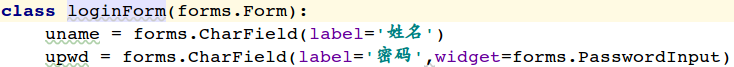
练习：

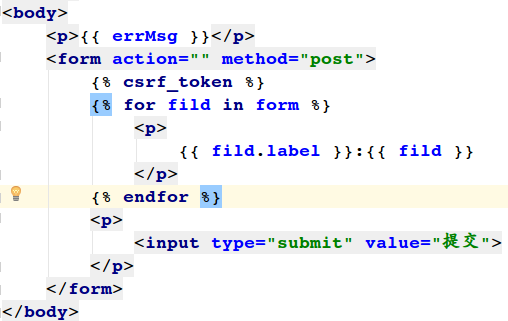
验证账号密码是否正确











作业：

1.创建一个表单类型注册RegisterForm,根据Users属性构建RegisterForm属性

2.在模板中，根据RegisterForm,创建表单控件

3.提交数据时，再交给RegisterForm接收提交的数据，通过验证后，将数据插入到数据库中



July30 Django

1 使用forms模块

1.在视图中，通过forms.Form快速获取表单数据

通过forms.Form的构造函数，接收post数据

form = xxxForm(request.POST)

2.需要让form通过验证后，再取值

目的：防止数据错误提交

form.is\_valid()

返回true：通过验证

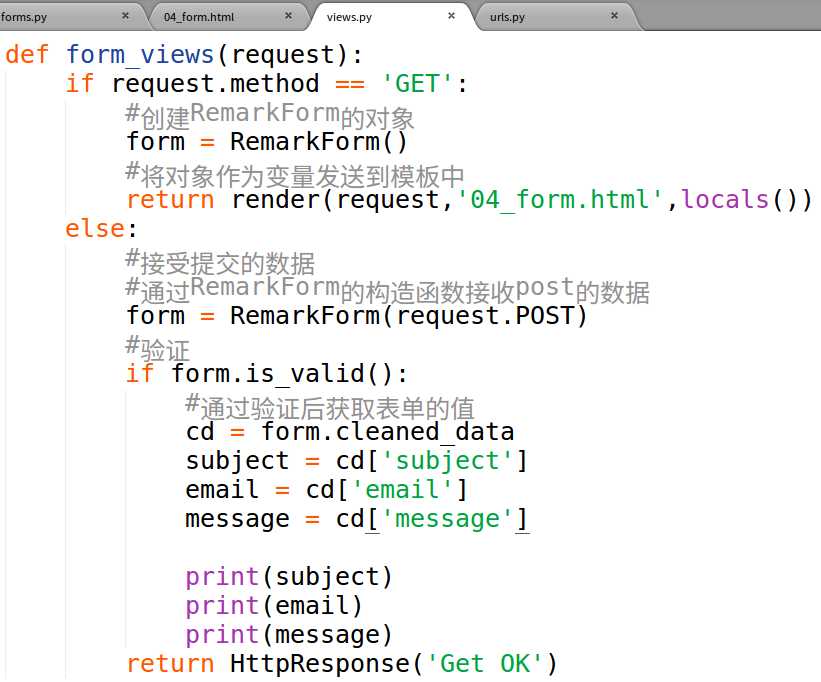
返回false：未通过验证，提交的数据与表单类型不匹配

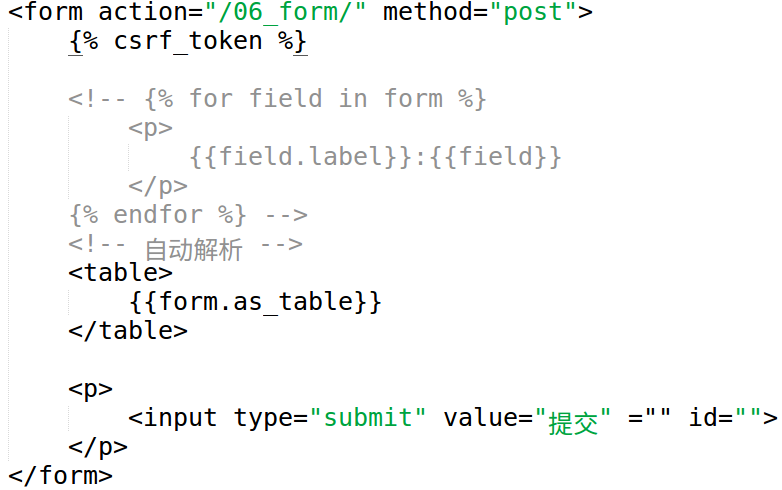
3.获取表单提交的数据

通过form.cleaned\_data(字典)获取提交的数据

cd = form.cleaned\_data

value = cd[‘name’]



练习：

1.在forms.py中追加一个RegisterForm类



2.增加以下属性，并生成到rergister.html显示成表单控件

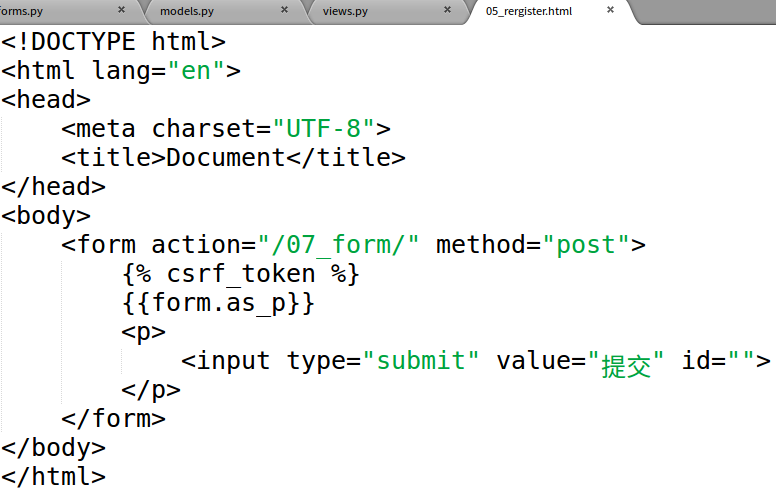
1.uname：文本框

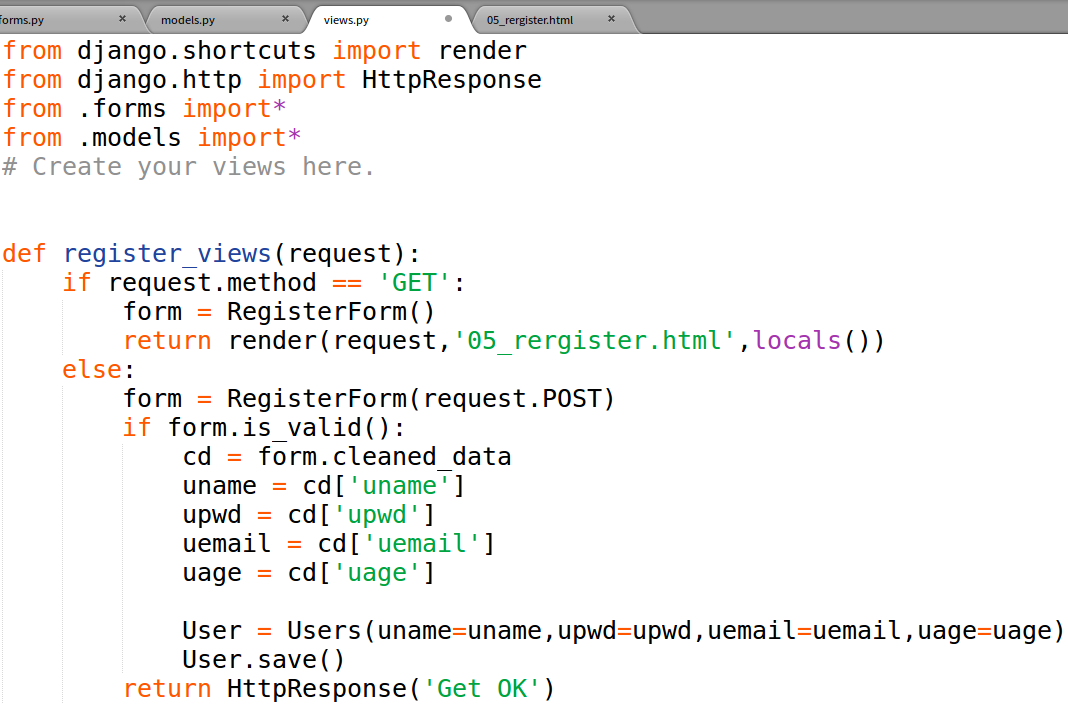
2.upwd：密码框

3.uemail：邮件框

4.uage：文本框

3.在register.html中增加form和submit

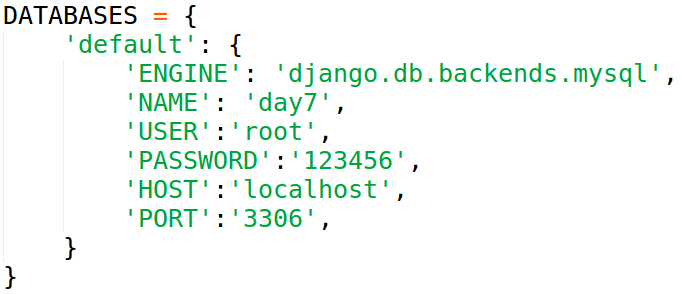


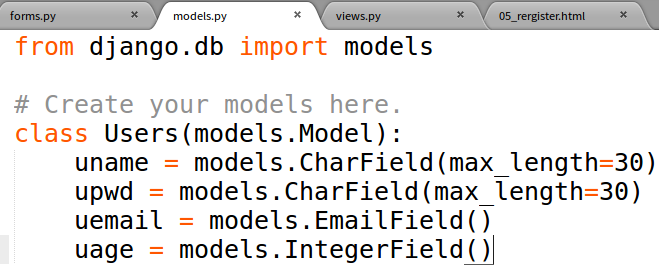


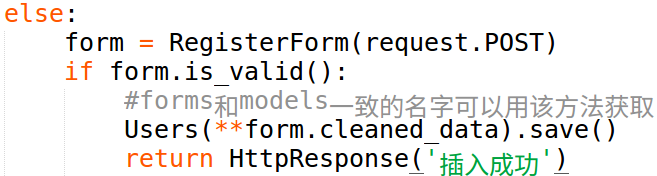
4.点击提交时，将以上数据插入到数据库中

创建数据库day7

创建models类-Users







1.1 form的高级处理

将Models和Forms结合到一起使用

1.在forms.py中，创建class，继承自forms.ModelForm

2.创建一个内部类：Meta

作用：关联当前Form和Models的

①model：指定要关联的Models类

ex：model = Users

②fields：指定从Models类中取出哪些字段生成控件

1.取值为“\_\_all\_\_”,全部属性都要生成表单控件

2.指定一个列表，声明允许生成表单控件的属性名

ex：fields=[‘uname’,’upwd’]

③labels：指定每个属性要关联的label，取值为字典

ex：

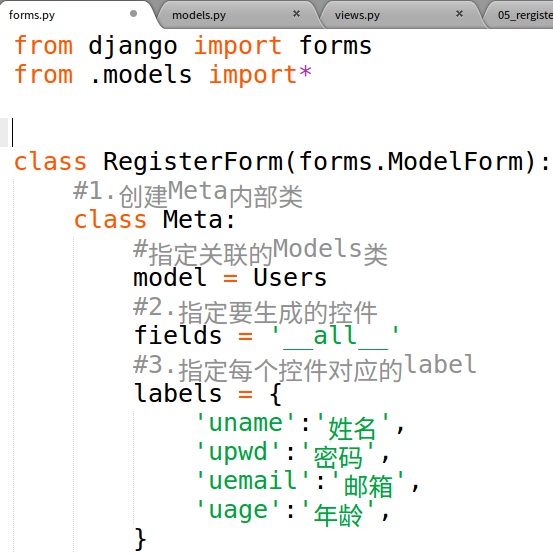
labels = {

‘属性名’:’label文本’,

‘属性名’:’label文本’,

……

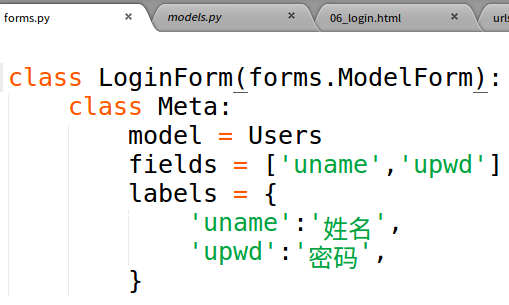
}

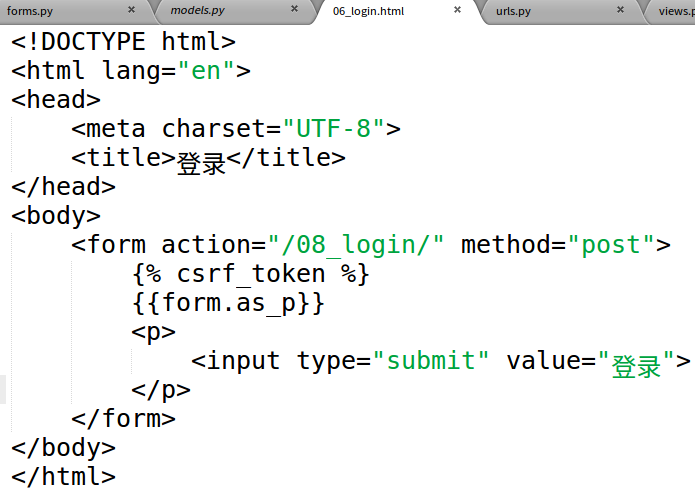


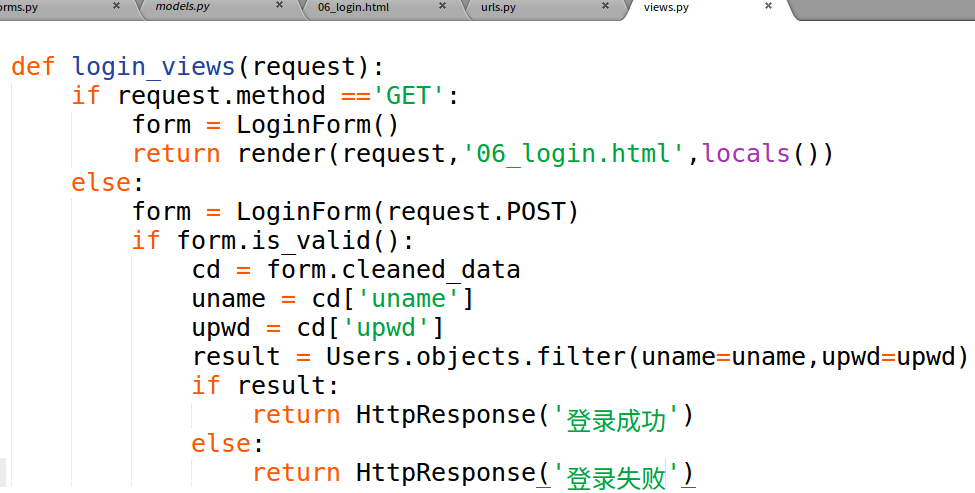
练习：

使用Models完成登录操作

创建一个登录的表单，控件从Users中获取







1.2 内部小部件

小部件（widget），表示的是生成到页面中的控件的类型以及其他属性

|  |  |
| --- | --- |
| TextInput | type=’text’ |
| NumberInput | type=’number’ |
| EmailInput | type=’email’ |
| URLInput | type=’url’ |
| PasswordInput | type=’password’ |
| HiddenInput | type=’hidden’ |
| CheckboxInput | type=’checkbox’ |
| Textarea | <textarea></textarea> |
| Select | <select></select> |

小部件的使用

1继承自forms.Form

①基本版

属性=forms.CharField(

label=’xxx’,

widget=forms.小部件类型

)

ex:

upwd = forms.CharField(

label=’用户密码’,

widget=forms.PasswordInput

)

age = forms.CharField(

label=’用户年龄’,

widget=form.NumberInput

)

②高级版

属性=forms.CharField(

label=’标签’,

widget=forms.小类型部件(

attrs={

‘html属性名1’:’html属性值1’,

‘html属性名2’:’html属性值2’,

}

)

)

ex:

upwd = forms.CharField(

label=’用户密码’,

widget=forms.PasswordInput(

attrs={

‘placeholder’:’请输入密码’,

‘class’:’form-control’,

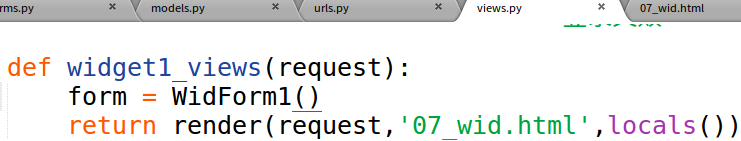
‘max-length’:’26’,

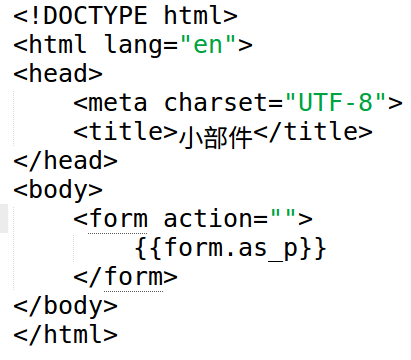
}

)

)







2.继承自forms.ModelForm

class WidForm2(forms.ModelForm):

class Meta:

model=Users

fields=’\_\_all\_\_’

labels={

‘属性1’:’标签1’,

‘属性2’:’标签2’,

}

widgets={

‘属性1’:forms.小部件类型(

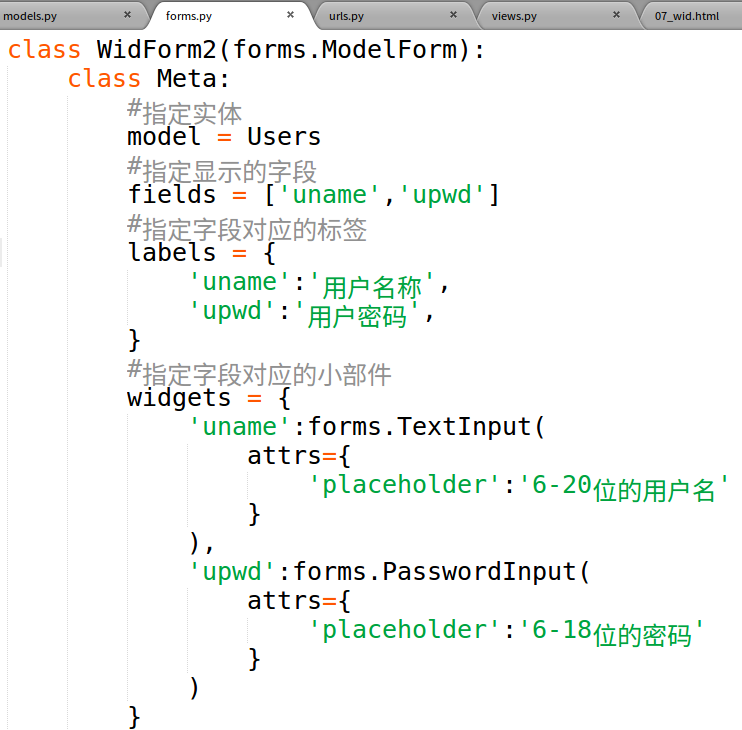
attrs={

‘属性’:’值’,

}

)

}



2 cookies

cookies是一种数据存储技术，将一段文本数据保存在客户端的一种技术，并且可以长时间保存

2.1 cookies的使用场合

1.保存登录信息

2.保存需要长期使用的数据

2.2 django中使用cookies

1.设置cookies的值（将数据保存到客户端）

cookies的设置，在服务器端是由响应对象发起的

语法：

响应对象.set\_cookie(key,value,expires)

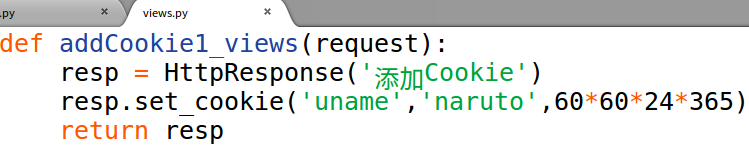
|  |  |
| --- | --- |
| key | cookie的名字 |
| value | cookie的值 |
| expires | 保存时间，以秒为单位 |

不使用模板---HttpResponse

resp=HttpResponse(‘给客户端一句话’)

resp.set\_cookie(‘key’,’value’,expires)

return resp

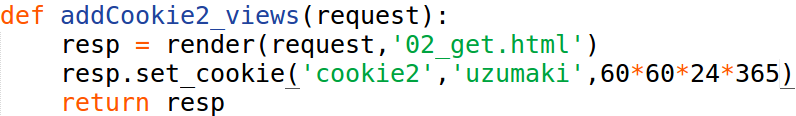


使用模板---render()

resp=render(request,’xx.html’)

resp.set\_cookie(‘key’,’value’,expires)

return resp



使用重定向---HttpResponseRedirect()

resp = HttpResponseRedirect(‘/地址/’)

resp.set\_cookie(‘key’,’value’,expires)

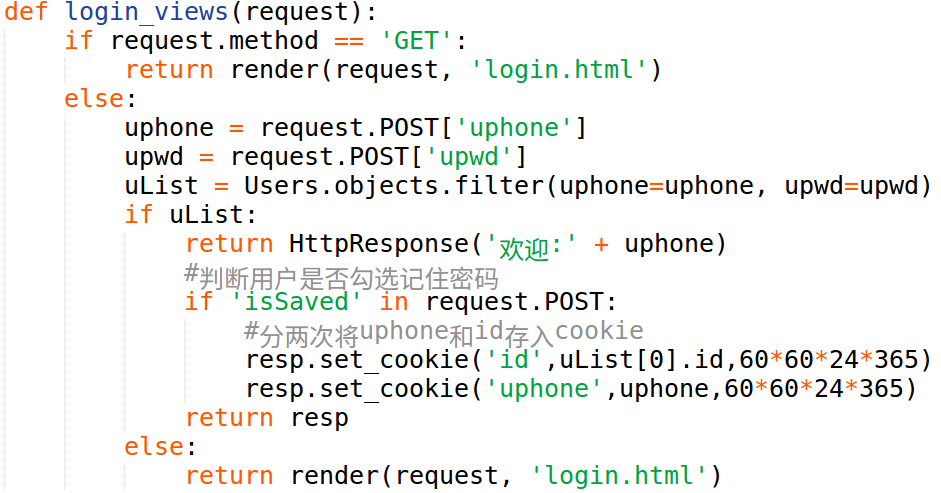
return resp

练习：

1.完成fruitday登录操作

2.判断手机号和密码是否正确

3.如果正确的话，并且勾选了记住密码的话，则登录信息（id，uname）保存进cookie，存在一年



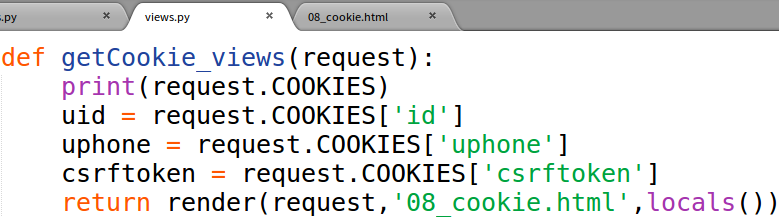
2.获取cookies（从客户端获取数据）

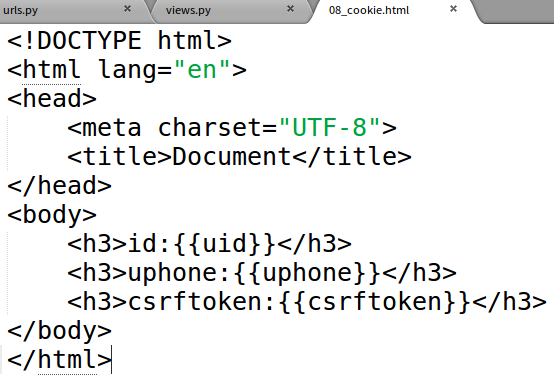
注意：

每次请求时，请求对象（request）会将当前路径下的所有cookies都带到服务器

语法：

request.COOKIES





取出的COOKIE值

{'uname': 'da', 'uid': '6',

'csrftoken': '0dcrPdcNiAZf8PdUjFwjwuBc9VPUezTgbVB1TviFfvFXqbDWBcHLbXzQBmRWkKh7'}

练习：

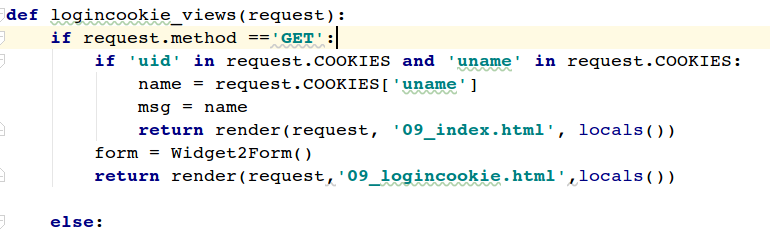
1.完善fruitday登录

2.当用户访问登录路径时：

①判断cookies中是否有登录信息

如果有：则显示欢迎：xxx

如果没有：则正常去往登录页面



存中文的COOKIES

import json

json.dumps(uname)

将Unicode码转换成中文

json.loads（uname）

July31

1 session

session，实际上是在服务器上为每个浏览器开辟一段空间，用于保存相关的请求信息

session的使用场合也是为了存放数据而存在的通常会把服务器端各个程序要用到的数据保存进去

2 Django中使用session

2.1 设置session的值

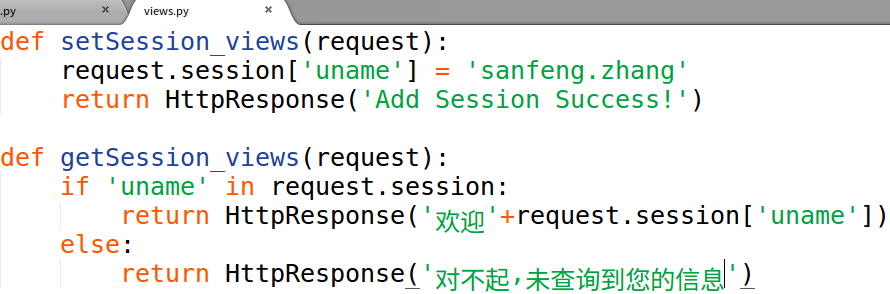
request.session[‘key’] = value

request.session.set\_expiry(time)设置session的过期时间，如果设置为0的话则表示关闭浏览器，session就失效

2.2 获取session的值

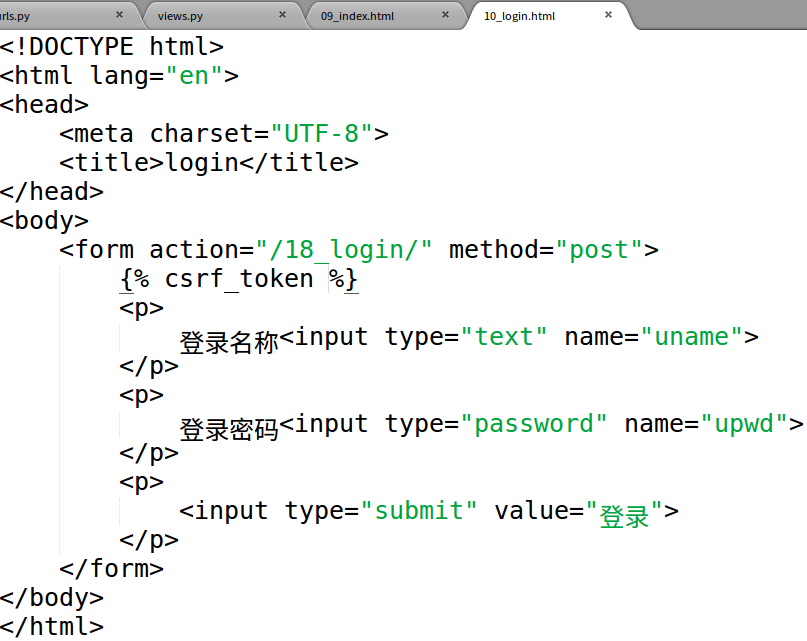
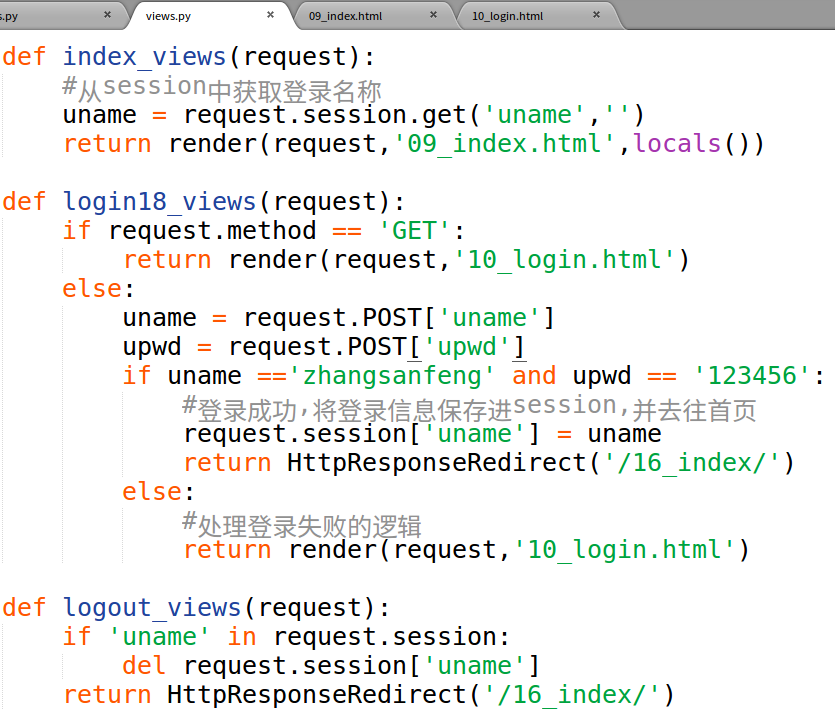
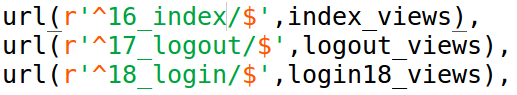
request.session[‘key’]

request.session.get(‘key’)



2.3 删除session的值

del request.session[‘key’]



3 在settings.py中有关session的设置

3.1 SESSION\_COOKIE\_AGE

作用：

设置sessionid在cookie中存活时长

ex：

SESSION\_COOKIE\_AGE=60\*30

3.2 SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE

在浏览器关闭时清除对应的session

ex:

SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE=True



