# 如何在列表、字典、集合中更具条件筛选数据？

解决方案

1. 列表：
   1. filter函数 filter(lambda x: x>=0, data)

## 列表推导式 [x for x in data if x >= 0]

1. 字典：
   1. 字典推导式 {k: v for k, v in d.items() if v >90 }
2. 集合：
   1. 集合推导式 {x for x in s if x >90}

## 默认字**典defaultdict**

默认给字典的值设置类型，其他与字典差不多

>>> from collections import defaultdict

>>> dic = defaultdict(list)

>>> dic['key'].append('value')

>>> dic

defaultdict(<class 'list'>, {'key': ['value']})

# 可命名元组

解决方案

1. 定义类似其他语言的枚举类型，也就是定义一系列数值常量

>>> student = ("Jim", 15, "male", "1@1.com")

>>> NAME AGE, SEX, EMAIL = range(4)

>>> print student[NAME]

SyntaxError: Missing parentheses in call to 'print'

>>> print (student[NAME])

Jim

>>> print (student[AGE])

15

>>> print (student[SEX])

male

>>> print (student[EMAIL])

1@1.com

>>>

1. 使用标准库中collections.namedtuple替代tuple

>>> from collections import namedtuple

>>> Student = namedtuple("Student", ["NAME", "AGE", "SEX", "EMAIL"]) #创建类

>>> s = Student("Jim", 16, "mail", "1@1.com")

>>> s

Student(NAME='Jim', AGE=16, SEX='mail', EMAIL='1@1.com')

>>> s2 = Student(NAME='Jim', AGE=16, SEX='mail', EMAIL='1@1.com')

>>> s.NAME

'Jim'

还可以用访问类属性的方法访问Namedtuple

>>> import collections

>>> MyTupleClass = collections.namedtuple("MyTupleClass", ['x', 'y', 'z'])

>>> obj = MyTupleClass(11,22,33)

>>> obj.x

11

>>> obj.y

22

>>> obj.z

33

**双向队列deque, 单向队列Queue**

双向队列

>>> import collections

>>> deq = collections.deque()

>>> deq.append('1')

>>> deq

deque(['1'])

>>> deq.appendleft('10')

>>> deq

deque(['10', '1'])

>>> deq.appendleft('1')

>>> deq

deque(['1', '10', '1'])

>>> deq.count('1')

2

>>> deq.extend(['yy','uu','ii'])

>>> deq

deque(['1', '10', '1', 'yy', 'uu', 'ii'])

>>> deq.extendleft(['yy1','uu1','ii1'])

>>> deq

deque(['ii1', 'uu1', 'yy1', '1', '10', '1', 'yy', 'uu', 'ii'])

>>> deq.rotate(1)

>>> deq

deque(['ii', 'ii1', 'uu1', 'yy1', '1', '10', '1', 'yy', 'uu'])

单向队列

>>> import queue

>>> que = queue.Queue()

>>> que.put('123')

>>> que.qsize()

1

>>> que.get()

**Count – 字典扩充**

1. 统计列表中每个元素出现的次数

方法1：字典构造法

>>> from random import randint

>>> [randint(0, 20) for \_ in range(30)]

[3, 7, 6, 7, 13, 8, 10, 7, 14, 20, 10, 4, 10, 6, 18, 1, 2, 5, 5, 2, 1, 14, 20, 3, 3, 17, 10, 15, 11, 3]

>>> data = [randint(0, 20) for \_ in range(30)]

>>> c = dict.fromkeys(data, 0)

>>> c

{1: 0, 2: 0, 4: 0, 5: 0, 7: 0, 8: 0, 9: 0, 11: 0, 12: 0, 15: 0, 16: 0, 17: 0, 19: 0, 20: 0}

>>> for x in data:

c[x] += 1

>>> c

{1: 3, 2: 1, 4: 1, 5: 5, 7: 1, 8: 3, 9: 2, 11: 3, 12: 3, 15: 1, 16: 2, 17: 1, 19: 2, 20: 2}

然后按上题方法进行排序

方法2使用collections.Counter对象

>>> from collections import Counter

>>> c2 = Counter(data) #看c是一样的

>>> c2[10]

0

>>> c2.most\_common(3) #最频繁的三个

[(5, 5), (1, 3), (8, 3)]

>>>

文档

import re

from collections import Counter

txt = open("1.txt").read()

c3 = Counter(re.split('\W+', txt)) #把文本按照非文本的正则表达式分割

c3.most\_common(33)

# 如何根据字典中值的大小对字典的项进行排序

解决方法：

>>> d = {x:random.randint(60,100) for x in "xyzabc"}

>>> d

{'a': 91, 'x': 60, 'c': 71, 'z': 99, 'y': 77, 'b': 65}

1. 利用zip将字典数据转化成元组

>>> sorted(tuple(zip(d.values(), d.keys())))

[(60, 'x'), (65, 'b'), (71, 'c'), (77, 'y'), (91, 'a'), (99, 'z')]

1. 传递sorted函数的key参数

>>> sorted(d.items(), key=lambda x: x[1])

[('x', 60), ('b', 65), ('c', 71), ('y', 77), ('a', 91), ('z', 99)]

# 如何快速找到多个字典中的公共键

>>> from random import randint, sample

>>> s1 = {x:randint(1,4) for x in sample('abcdefg',randint(3,6))}

>>> s2 = {x:randint(1,4) for x in sample('abcdefg',randint(3,6))}

>>> s3 = {x:randint(1,4) for x in sample('abcdefg',randint(3,6))}

>>> s1

{'f': 4, 'd': 4, 'b': 2}

>>> s2

{'d': 4, 'c': 1, 'b': 1, 'e': 2, 'g': 4}

>>> s3

{'d': 3, 'a': 1, 'c': 1, 'g': 3, 'f': 1, 'b': 2}

解决方法：

>>> res = []

>>> for k in s1:

if k in s2 and k in s3:

res.append(k)

>>> res

['d', 'b']

方法2使用字典的keys()方法得到字典集合

>>> s1.keys() & s2.keys() & s3.keys()

方法3 map reduce多轮

>>> from functools import reduce

>>> reduce(lambda a, b: a & b, map(dict.keys, [s1,s2,s3]))

# 如何让字典保持有序

使用collections.OrderedDict

>>> d = OrderedDict()

>>> d['Jim'] = (1, 35)

>>> d['Leo'] = (2, 37)

>>> d['Bob'] = (3, 40)

>>> for e in d:

print (e)

from time import time

from random import randint

from collections import OrderedDict

d = OrderedDict()

players = list('ABCDEFGH')

start = time()

for i in range(8):

input()

p = players.pop(randint(0, 7-i))

end = time()

print(i+1, p, end-start)

d[p] = (i+1, end-start)

print()

print ("-" \*20)

for k in d: print(k, d[k])

>>>1 G 1.4990849494934082

2 A 2.2461278438568115

3 E 2.790158987045288

4 C 3.214184045791626

5 B 3.6762099266052246

6 H 4.067231893539429

# 可迭代对象和迭代器对象

itertools.chain函数把可迭代对象串起来

zip函数能把两个序列同一序列号的元素做成元组

reversed函数是反向迭代器

itertools.islice 可把可迭代对象变成可索引对象

# 一行式构造class

class A(object):

def \_\_init\_\_(self, a, b, c, d, e, f):

self.\_\_dict\_\_.update({k: v for k, v in locals().items() if k != 'self'})

相当于

def \_\_init\_\_(self, a, b, c, d, e, f):

self.a = a

self.b = b

self.c = c

self.d = d

self.e = e

self.f = f

# 如何处理二进制文件

## 深浅拷贝

import copy

copy.copy() -- 浅拷贝

copy.deepcopy() -- 深拷贝

id()函数可获取内存地址

对于数字和字符串，无论是浅拷贝还是深拷贝，都指向 内存同一个地址

>>> import copy

>>> a1 = "sdfdsaf"

>>> a3 = copy.copy(a1)

>>> id(a1)

43233408

>>> id(a3)

43233408

>>> a5 = copy.deepcopy(a1)

>>> id(a1)

43233408

>>> id(a5)

43233408

尝试把a的值变一下

>>> a1 = "aaa"

>>> a3

'sdfdsaf'

>>> a5

'sdfdsaf'

>>> id(a1)

43234752

>>> id(a3)

43233408

发现a1的内存地址改变了

2 对于其他数据类型

>>> #元组、列表、字典等其他数据类型

>>> n1 = {'k1':'wu', 'k2':123, 'k3':['alex',456]}

>>> n2 = n1

>>> id(n1)

42826320

>>> id(n2)

42826320

实施浅拷贝

>>> n3 = copy.copy(n1)

>>> id(n1)

42826320

>>> id(n3)

43245264

内存的地址不一样了

>>> n1

{'k1': 'wu', 'k2': 123, 'k3': ['alex', 456]}

>>> n3

{'k1': 'wu', 'k2': 123, 'k3': ['alex', 456]}

>>> id(n1['k3'])

42980496

>>> id(n3['k3'])

42980496

然而，他们内部的数据k3看起来一样，指向的内存地址也是一样的

尝试改变n1的数据值

>>> n1['k3'] = 2

>>> n1

{'k1': 'wu', 'k2': 123, 'k3': 2}

>>> n3

{'k1': 'wu', 'k2': 123, 'k3': ['alex', 456]}

>>> id(n1)

42826320

>>> id(n3)

43245264

发现n3没有变化，内存地址也不一样了，也就是n1搬家了

再试试深拷贝

>>> n1 = {'k1': 'wu', 'k2': 123, 'k3': ['alex', 456]}

>>> n7 = copy.deepcopy(n1)

>>> n7

{'k1': 'wu', 'k2': 123, 'k3': ['alex', 456]}

>>> id(n1)

43174768

>>> id(n7)

43245408

>>> id(n1['k3'])

43110568

>>> id(n7['k3'])

42980176

发送邮件smtplib

import smtplib

from email.mime.text import MIMEText

from email.utils impor formataddr

def mail():

try:

msg = MIMEText('邮件内容','plain', 'utf-8')

msg['From'] = formataddr['名字','邮箱']

msg['To'] = formataddr['名字','邮箱']

msg['Subject'] = ['主题']

server = smtplib.SMTP('smtp.126.com',25)

server.login('要登录的邮箱', '密码')

server.sendmail('邮箱地址', ['邮箱地址'], msg.as\_string())

server.quit()

# Django模块

## 第一章 创建Django项目

### 创建项目

cd 命令到文件夹下，然后django-admin startproject XXX

### 创建服务器

**$** python manage.py runserver

打开<http://127.0.0.1:8000/>即可看到成功

8000可以修改成其他端口

**$** python manage.py runserver 0:8000

这句表示使用0.0.0.0:8000作为ip和端口

It worked!

Congratulations on your first Django-powered page.

Next, start your first app by running python manage.py startapp [app\_label].

You're seeing this message because you have DEBUG = True in your Django settings file and you haven't configured any URLs. Get to work!

### 创建应用

**$** python manage.py startapp polls

### 第一个View器

Polls/views.py =>

**def** index(request):

**return** HttpResponse("Hello, world. You're at the polls index.")

**def** index(request):

**return** HttpResponse("Hello, world. You're at the polls index.")

### 创建URLconf文件

在polls文件夹下创建urls.py=>

**from** **django.conf.urls** **import** url

**from** **.** **import** views

urlpatterns = [

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

]

url(regex, view, [kwargs, name])

regex是需要请求的url形式的正则表达，不会搜索GET和POST

view是查看的视图

kwargs暂无

name从Django特别是模版内部访问

### 添加polls到mysite/urls.py中

**from** **django.conf.urls** **import** include, url

**from** **django.contrib** **import** admin

urlpatterns = [

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

url(r'^admin/', admin.site.urls),

]

此时python manage.py runserver应该显示在index视图下创建的“*Hello, world. You’re at the polls index.*”

## 第二章 创建数据库

系统默认是sqlite3，如需更改，可以到settings.py下的DATABASES ‘default’修改，

[ENGINE](https://docs.djangoproject.com/en/1.11/ref/settings/#std:setting-DATABASE-ENGINE) ：

'django.db.backends.sqlite3'

'django.db.backends.postgresql'

'django.db.backends.mysql'

'django.db.backends.oracle'

视乎实际需要

settings.py：

TIME\_ZONE时区

[INSTALLED\_APPS](https://docs.djangoproject.com/en/1.11/ref/settings/#std:setting-INSTALLED_APPS)  在本实例中使用的Django的应用{

[django.contrib.admin](https://docs.djangoproject.com/en/1.11/ref/contrib/admin/#module-django.contrib.admin) – 管理员模块

[django.contrib.auth](https://docs.djangoproject.com/en/1.11/topics/auth/#module-django.contrib.auth) – 认证系统模块

[django.contrib.contenttypes](https://docs.djangoproject.com/en/1.11/ref/contrib/contenttypes/#module-django.contrib.contenttypes) – 内容类型框架

[django.contrib.sessions](https://docs.djangoproject.com/en/1.11/topics/http/sessions/#module-django.contrib.sessions) – session 框架

[django.contrib.messages](https://docs.djangoproject.com/en/1.11/ref/contrib/messages/#module-django.contrib.messages) – messaging 框架

[django.contrib.staticfiles](https://docs.djangoproject.com/en/1.11/ref/contrib/staticfiles/#module-django.contrib.staticfiles) – 管理静态文件的框架

**$** python manage.py migrate

[migrate](https://docs.djangoproject.com/en/1.11/ref/django-admin/#django-admin-migrate)命令查看settings.py中 [INSTALLED\_APPS](https://docs.djangoproject.com/en/1.11/ref/settings/#std:setting-INSTALLED_APPS) 的设置，然后创建必要的数据库，更新已经修改的table和models

在此教程中，Question和Choice两个model被创建

polls/models.py=>

**from** **django.db** **import** models

c**lass** **Question**(models.Model):

question\_text = models.CharField(max\_length=200)

pub\_date =models.DateTimeField('date published')

**class** **Choice**(models.Model):

question = models.ForeignKey(Question, on\_delete=models.CASCADE)

choice\_text = models.CharField(max\_length=200)

votes = models.IntegerField(default=0)

每个model都由 [django.db.models.Model](https://docs.djangoproject.com/en/1.11/ref/models/instances/" \l "django.db.models.Model)体现，每个model都有一些类变量

CharField 字符

DateTimeField datetimes

把刚刚创立的polls放进

mysite/settings.py =>

INSTALLED\_APPS = [

'polls.apps.PollsConfig',

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

]

**$** python manage.py makemigrations polls

即可把polls的apps创建s

Migrations for 'polls':

polls/migrations/0001\_initial.py:

- Create model Choice

- Create model Question

- Add field question to choice

Makemigrations即告诉Django已对model进行修改，要把他变成migration。Migration是django存储model修改的方式，是磁盘上的文件**polls/**migrations**/0001\_initial.py**.

**$** python manage.py sqlmigrate polls 0001

使用这条命令可以查看即将执行的sql命令:

**BEGIN**;

*--*

*-- Create model Choice*

*--*

**CREATE** **TABLE** "polls\_choice" (

"id" serial **NOT** **NULL** **PRIMARY** **KEY**,

"choice\_text" varchar(200) **NOT** **NULL**,

"votes" integer **NOT** **NULL**

);

*--*

*-- Create model Question*

*--*

**CREATE** **TABLE** "polls\_question" (

"id" serial **NOT** **NULL** **PRIMARY** **KEY**,

"question\_text" varchar(200) **NOT** **NULL**,

"pub\_date" **timestamp** **with** time **zone** **NOT** **NULL**

);

*--*

*-- Add field question to choice*

*--*

**ALTER** **TABLE** "polls\_choice" **ADD** **COLUMN** "question\_id" integer **NOT** **NULL**;

**ALTER** **TABLE** "polls\_choice" **ALTER** **COLUMN** "question\_id" **DROP** **DEFAULT**;

**CREATE** **INDEX** "polls\_choice\_7aa0f6ee" **ON** "polls\_choice" ("question\_id");

**ALTER** **TABLE** "polls\_choice"

**ADD** **CONSTRAINT** "polls\_choice\_question\_id\_246c99a640fbbd72\_fk\_polls\_question\_id"

**FOREIGN** **KEY** ("question\_id")

**REFERENCES** "polls\_question" ("id")

**DEFERRABLE** **INITIALLY** **DEFERRED**;

**COMMIT**;

**$** [python manage.py check](https://docs.djangoproject.com/en/1.11/ref/django-admin/#django-admin-check)

查错

**$** python manage.py migrate

修改模版的三步

1. 在models.py中修改models
2. 执行python manage.py makemigrations创建变更
3. 执行python manage.py migrate把变更执行到数据库中

**$** python manage.py shell

开启ide

如果不想开启这个ide，可以把**[DJANGO\_SETTINGS\_MODULE](https://docs.djangoproject.com/en/1.11/topics/settings/" \l "envvar-DJANGO_SETTINGS_MODULE)**环境变量更改成**mysite.settings**，在manage.py同级目录下开启ide然后import django; django.setup()

**>>> from** **polls.models** **import** Question, Choice *# Import the model classes we just wrote.*

# No questions are in the system yet.

**>>>** Question.objects.all()

<QuerySet []>

# Create a new Question.

# Support for time zones is enabled in the default settings file, so

# Django expects a datetime with tzinfo for pub\_date. Use timezone.now()

# instead of datetime.datetime.now() and it will do the right thing.

**>>> from** **django.utils** **import** timezone

**>>>** q = Question(question\_text="What's new?", pub\_date=timezone.now())

# Save the object into the database. You have to call save() explicitly.

**>>>** q.save()

# Now it has an ID. Note that this might say "1L" instead of "1", depending

# on which database you're using. That's no biggie; it just means your

# database backend prefers to return integers as Python long integer

# objects.

**>>>** q.id

1

# Access model field values via Python attributes.

**>>>** q.question\_text

"What's new?"

**>>>** q.pub\_date

datetime.datetime(2012, 2, 26, 13, 0, 0, 775217, tzinfo=<UTC>)

# Change values by changing the attributes, then calling save().

**>>>** q.question\_text = "What's up?"

**>>>** q.save()

# objects.all() displays all the questions in the database.

**>>>** Question.objects.all()

<QuerySet [<Question: Question object>]>

这时发现最后一个Question.objects.all()没卵用，可修改models.py

**from** **django.db** **import** models

**from** **django.utils.encoding** **import** python\_2\_unicode\_compatible

@python\_2\_unicode\_compatible *# only if you need to support Python 2*

**class** **Question**(models.Model):

*# ...*

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

**return** self.question\_text

**def** was\_published\_recently(self):

**return** self.pub\_date >= timezone.now() - datetime.timedelta(days=1)

再度运行shell

**>>> from** **polls.models** **import** Question, Choice

# Make sure our \_\_str\_\_() addition worked.

**>>>** Question.objects.all()

<QuerySet [<Question: What's up?>]>

# Django provides a rich database lookup API that's entirely driven by

# keyword arguments.

**>>>** Question.objects.filter(id=1)

<QuerySet [<Question: What's up?>]>

**>>>** Question.objects.filter(question\_text\_\_startswith='What')

<QuerySet [<Question: What's up?>]>

# Get the question that was published this year.

**>>> from** **django.utils** **import** timezone

**>>>** current\_year = timezone.now().year

**>>>** Question.objects.get(pub\_date\_\_year=current\_year)

<Question: What's up?>

# Request an ID that doesn't exist, this will raise an exception.

**>>>** Question.objects.get(id=2)

Traceback (most recent call last):

...

DoesNotExist: Question matching query does not exist.

# Lookup by a primary key is the most common case, so Django provides a

# shortcut for primary-key exact lookups.

# The following is identical to Question.objects.get(id=1).

**>>>** Question.objects.get(pk=1)

<Question: What's up?>

# Make sure our custom method worked.

**>>>** q = Question.objects.get(pk=1)

**>>>** q.was\_published\_recently()

True

# Give the Question a couple of Choices. The create call constructs a new

# Choice object, does the INSERT statement, adds the choice to the set

# of available choices and returns the new Choice object. Django creates

# a set to hold the "other side" of a ForeignKey relation

# (e.g. a question's choice) which can be accessed via the API.

**>>>** q = Question.objects.get(pk=1)

# Display any choices from the related object set -- none so far.

**>>>** q.choice\_set.all()

<QuerySet []>

# Create three choices.

**>>>** q.choice\_set.create(choice\_text='Not much', votes=0)

<Choice: Not much>

**>>>** q.choice\_set.create(choice\_text='The sky', votes=0)

<Choice: The sky>

**>>>** c = q.choice\_set.create(choice\_text='Just hacking again', votes=0)

# Choice objects have API access to their related Question objects.

**>>>** c.question

<Question: What's up?>

# And vice versa: Question objects get access to Choice objects.

**>>>** q.choice\_set.all()

<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]>

**>>>** q.choice\_set.count()

3

## Django Admin

首先需要创建可以登录admin的superuser

**$** python manage.py createsuperuser XXX

Username: admin

Email address: admin@example.com

Password: \*\*\*\*\*\*\*\*\*\*

Password (again): \*\*\*\*\*\*\*\*\*

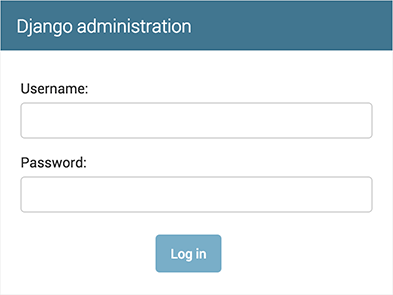
Superuser created successfully.

## 打开开发服务器

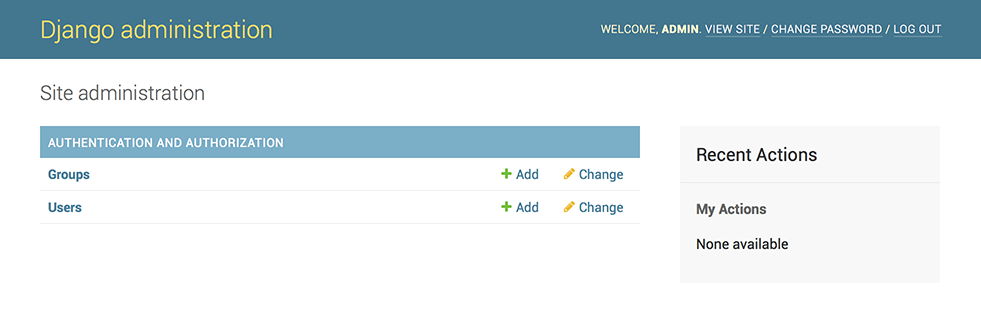
**$** python manage.py runserver

打开命令行上显示的ip地址

和端口例如<http://127.0.0.1:8000/admin/>.



输入密码后



[**django.contrib.auth**](https://docs.djangoproject.com/en/1.11/topics/auth/#module-django.contrib.auth)中存储着groups&users

使之前的poll可修改

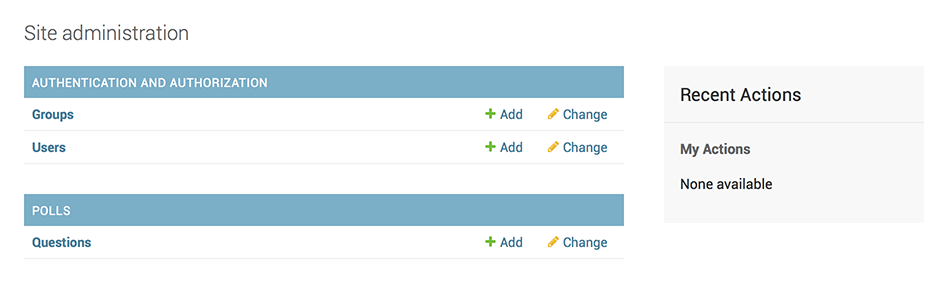
**polls/admin.py =>**

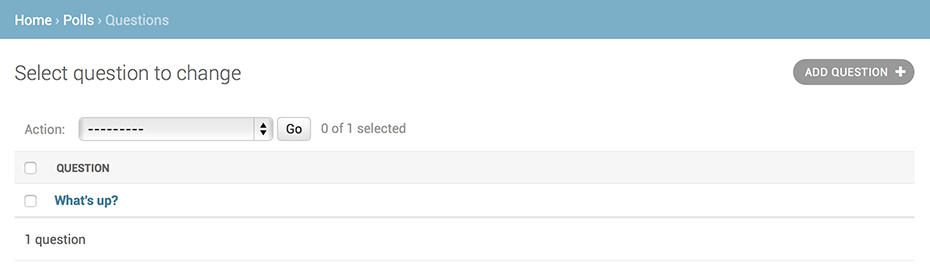
**from** **django.contrib** **import** admin

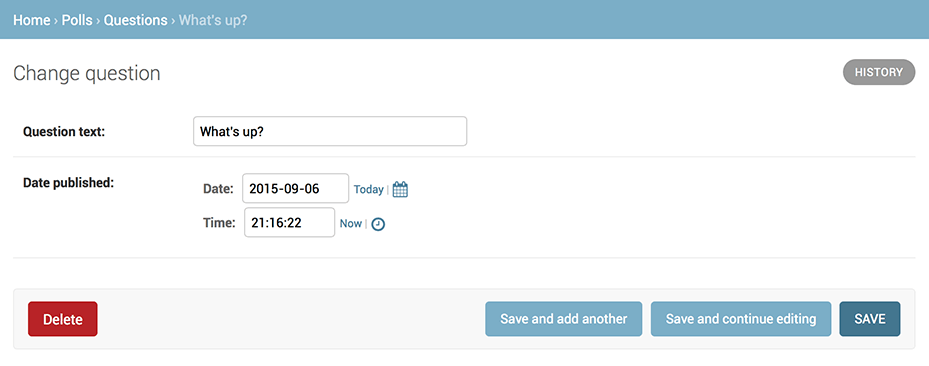
**from** **.models** **import** Question

admin.site.register(Question)

## 探索admin功能







可以修改question相关的东西

下面可以修改删除

# 第三章 视图

视图就是有功能的模板

比如在博客中，可能会有以下视图

主页

细节页

以年计算的文件夹

以月计算的文件夹

以日计算的文件夹

在本例中，我们有以下视图

index页—展示最新的问题

detail页—展示问题文本，但没有结果，只有一个用于投票的表单

result页—展示某个问题的结果

vote action –可以用于投票的

Django不采用这样的格式“ME2/Sites/dirmod.asp?sid=&type=gen&mod=Core+Pages&gid=A6CD4967199A42D9B65B1B”.

而是采用/newsarchive/<year>/<month>/这样的格式

要从url变成视图，django使用一种叫URLconfs的东西，把URL正则表达式映射到视图

## 编写更多视图

polls/views.py =>

def detail(request, question\_id):

return HttpResponse("You're looking at question %s." % question\_id)

def results(request, question\_id):

response = "You're looking at the results of question %s."

return HttpResponse(response % question\_id)

def vote(request, question\_id):

return HttpResponse("You're voting on question %s." % question\_id)

把这些新视图写入polls.url

from django.conf.urls import url

from . import views

urlpatterns = [

# ex: /polls/

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

# ex: /polls/5/

url(r'^(?P<question\_id>[0-9]+)/$', views.detail, name='detail'),

# ex: /polls/5/results/

url(r'^(?P<question\_id>[0-9]+)/results/$', views.results, name='results'),

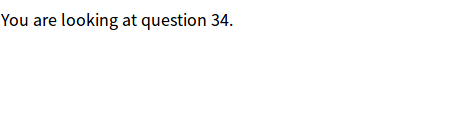
# ex: /polls/5/vote/

url(r'^(?P<question\_id>[0-9]+)/vote/$', views.vote, name='vote'),

]

此时如果runserver，浏览器中输入http://127.0.0.1:8000/polls/34/vote/

就会打开



浏览器中输入/polls/34之后，因为settings.py中ROOT\_URLCONF的设定，会打开mysite.urls，然后找到urlpatterns变量，逐个检查正则，找到’^polls’就去掉其空格，传入剩下的”34/”，然后调出detail（）视图

正则中?P<question\_id>与question\_id是同一回事

因为可以使用正则，所以url的样式可以自己去改变

## 写视图

每个视图都负责做以下两件事的其中之一：返回包含请求页面的对象HttpResponse对象，找不到就Http404。可以使用任何第三方模板系统来生成view

修改

polls/views.py =>

from django.http import HttpResponse

from .models import Question

def index(request):

latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]

output = ', '.join([q.question\_text for q in latest\_question\_list])

return HttpResponse(output)

# Leave the rest of the views (detail, results, vote) unchanged

## 模板

在polls文件夹下新建templates文件夹

settings中的TEMPLATES决定django如何渲染templates.一般来说都会在INSTALLED\_APPS所列的apps文件夹下的templates中寻找模板

在templates下新建polls文件夹，其中放一个index.html，用polls/index.html可以访问

polls/templates/polls/index.html

{% if latest\_question\_list %}

<ul>

{% for question in latest\_question\_list %}

<li><a href="/polls/{{ question.id }}/">{{ question.question\_text }}</a></li>

{% endfor %}

</ul>

{% else %}

<p>No polls are available.</p>

{% endif %}

然后修改

polls/views.py

from django.http import HttpResponse

from django.template import loader

from .models import Question

def index(request):

latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]

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

context = {

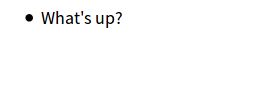
'latest\_question\_list': latest\_question\_list,

}

return HttpResponse(template.render(context, request))

这部分代码读取polls/index.html的模板并传入context，context是映射模板变量

runserver并打开http://127.0.0.1:8000/polls/



## 快捷方式render()

可以采用render()函数简化上面的东东

polls/view.py =>

from django.shortcuts import render

from .models import Question

def index(request):

latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]

context = {'latest\_question\_list': latest\_question\_list}

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

render(request, templateName, [contextDictionary])

第一个参数是请求对象，第二个是模板名称，第三个是可选参数字典，返回在context下给定Template渲染HttpResponse对象

## 404错误

polls/views.py = >

from django.http import Http404

from django.shortcuts import render

from .models import Question

# ...

def detail(request, question\_id):

try:

question = Question.objects.get(pk=question\_id)

except Question.DoesNotExist:

raise Http404("Question does not exist")

return render(request, 'polls/detail.html', {'question': question})

如果请求的问题id不存在，Http404会抛出Http404错误

快捷方式get\_object\_or\_404()

polls/views.py =>

from django.shortcuts import get\_object\_or\_404, render

from .models import Question

# ...

def detail(request, question\_id):

question = get\_object\_or\_404(Question, pk=question\_id)

return render(request, 'polls/detail.html', {'question': question})

get\_object\_or\_404(DjangoModel, [任意关键词参数])，然后传给get()（这个函数只返回一个对象，多了会出现MultiObjectReturned错误），如果对象不存在就抛出Http404，

[get\_list\_or\_404()](https://docs.djangoproject.com/en/1.11/topics/http/shortcuts/#django.shortcuts.get_list_or_404)是[get\_object\_or\_404()](https://docs.djangoproject.com/en/1.11/topics/http/shortcuts/" \l "django.shortcuts.get_object_or_404)的多元素版本，传给的函数不是[get()](https://docs.djangoproject.com/en/1.11/ref/models/querysets/" \l "django.db.models.query.QuerySet.get)而是[filter()](https://docs.djangoproject.com/en/1.11/ref/models/querysets/" \l "django.db.models.query.QuerySet.filter)

# 使用模板系统

polls/templates/polls/detail.html

<h1>{{ question.question\_text }}</h1>

<ul>

{% for choice in question.choice\_set.all %}

<li>{{ choice.choice\_text }}</li>

{% endfor %}

</ul>