

NSD Devweb DAY05

1. [案例1：熟悉API](#)
2. [案例2：编写视图](#)
3. [案例3：创建模板](#)
4. [案例4：完成投票系统](#)

1 案例1：熟悉API

1.1 问题

1. 进入python shell模式
2. 导入Question和Choice类
3. 创建Question和Choice对象
4. 通过id等条件查找对象
- 5.

1.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：进入python shell 模式

使用如下命令来调用Python shell：

```
01 (django_env) [root@localhost my site] # python manage.py shell
```

在启动解释器之前，它告诉Django使用哪个设置文件。Django框架的大部分子系统，包括模板系统，都依赖于配置文件；如果Django不知道使用哪个配置文件，这些系统将不能工作。

Django搜索DJANGO_SETTINGS_MODULE环境变量，它被设置在settings.py中。因为manage.py 设置了DJANGO_SETTINGS_MODULE 环境变量，该环境变量告诉Django导入mysite/settings.py文件的路径。

当你运行命令：python manage.py shell，它将自动帮你处理DJANGO_SETTINGS_MODULE。这样可以免去你大费周章地去配置那些你不熟悉的环境变量。

步骤二：导入Question和Choice类

首先要导入前面编写的Question和Choice数据库模块

```
01 >>> from polls.models import Question, Choice
```

查看所有的Question

[Top](#)

```
01 >>> Question.objects.all()
```

02. <Query Set []>

因为还没有创建任何问题，所以返回的是一个空查询集。

步骤三：创建Question和Choice对象

1)Question模型中需要时间，可以使用django工具

```
01. >>> from django.utils import timezone
```

2)创建Question对象

```
01. >>> q = Question( question_text="你希望进入哪个公司工作?", pub_date=timezone.now() )
02. >>> q.save()
```

保存这个对象到数据库中。必须显示地调用save()

3)查看所有的Question

```
01. >>> Question.objects.all()
02. <Query Set [ <Question: Question object>, <Question: Question object> ]>
```

此时，<Question object> 完全是这个对象无意义的表示。

4)修复Question对象无意义显示，修改models.py文件：

```
01. class Question( models.Model ) :
02.     question_text = models.CharField( max_length=200 )
03.     pub_date = models.DateTimeField( 'date published' )
04.     def __str__( self ) :
05.         return self.question_text
06. class Choice( models.Model ) :
07.     question = models.ForeignKey( Question, on_delete=models.CASCADE )
08.     choice_text = models.CharField( max_length=200 )
09.     votes = models.IntegerField( default=0 )
10.     def __str__( self ) :
11.         return self.choice_text
```

[Top](#)

5)修改完毕，重新加载

```
01. (django_env) [root@localhost my site] # python manage.py shell
02. >>> from polls.models import Question, Choice
03. >>> Question.objects.all()
04. <Query Set [ <Question: 你希望进入哪个公司工作?>]>
```

6)创建Choice对象

由于存在外键关系，django通过Question对象可以反向得到Choice对象集

```
01. >>> q.choice_set.all()
02. <Query Set [ ]>
03. >>> q.choice_set.create(choice_text='阿里巴巴', votes=0)
04. <Choice: 阿里巴巴>
05. >>> q.choice_set.create(choice_text='华为', votes=0)
06. <Choice: 华为>
```

7)一旦创建好对象，就可以通过模型创建的字段对其进行访问了

```
01. >>> q.id
02. 1
03. >>> q.question_text
04. '你希望进入哪个公司工作?'
05. >>> q.pub_date
06. datetime.datetime(2018, 8, 30, 7, 27, 40, 626426, tzinfo=<UTC>)
07. >>> q.choice_set.count() #通过count()函数可以取得选项数量。
08. 2
09. >>> q.choice_set.all() #从关联对象设置中显示任意选项
10. <Query Set [ <Choice: 阿里巴巴>, <Choice: 华为>]>
11. >>> Question.objects.all()
12. <Query Set [ <Question: 你希望进入哪个公司工作?>]>
```

步骤四：通过id等条件查找对象

1)Django 提供了丰富的数据库查询 API，通过关键字查询

```
01. >>> from polls.models import Question, Choice
```

[Top](#)

- ```
02. >>> Question.objects.get(id=1)
03. <Question: 你希望进入哪个公司工作?>
```

此时，如果id不存在将引发异常

2)通过主键查询数据是常见的情况，因此 Django 提供了精确查找主键的快捷方式，下面的这个代码和Question.objects.get(id=1)结果相同

- ```
01. >>> Question.objects.get(pk=1)
02. <Question: 你希望进入哪个公司工作?>
```

确认我们自定义的方法was_published_recently()生效。

- ```
01. >>> q = Question.objects.get(pk=1)
02. >>> q.was_published_recently()
03. True
```

3)filter(id=1)用来筛选指定参数。

- ```
01. >>> Question.objects.filter(id=1)
02. <Query Set [ <Question: 你期待哪个公司给你发offer?> ]>
```

4)Django通过灵活的双下划线实现属性查找

- ```
01. >>> Question.objects.filter(question_text__startswith='你')
02. <Query Set [<Question: 你期待哪个公司给你发offer?>]>
```

question\_text\_\_startswith由两部分组成，字段：question\_text 后缀关键字：\_\_startswith。  
question\_text\_\_startswith='你'作用是：筛选指定字段，以 '你' 开头的内容。

## 2 案例2：编写视图

### 2.1 问题

1. 为投票系应用编写投票功能视图、问题详情视图以及问题结果视图
2. 为第1步的三个视图编写URLCONF，通过相应的URL可以调用对应的视图函数

[Top](#)

### 2.2 步骤

实现此案例需要按照如下步骤进行。

### 步骤一：为投票系应用编写投票功能视图、问题详情视图以及问题结果视图

引入HttpResponse，它是用来向页面返回内容的，就想python中print一样，只不过HttpResponse是把内容显示到网页上，

我们定义三个函数，第一个参数必须是request，与网页发来的请求有关，request变量里面包含get或post的内容，用户浏览器，系统等信息。

函数最终返回一个HttpResponse对象，经过处理后，最终将显示几个字到网页上。

#### 1) 编写投票功能视图：

编写视图文件views.py，处理对Question中Choice的投票

```
01. from django.http import HttpResponse
02.
03. def vote(request, question_id):
04. return HttpResponse("您正在为[%s] 投票。")
```

#### 2)问题详情视图：

编写视图文件views.py，显示单个Question的具体内容，不显示该议题的当前投票结果，而是提供一个投票的表单

```
01. def detail(request, question_id):
02. return HttpResponse("你正在查看的问题是：%s。" % question_id)
```

#### 3) 问题结果视图：

编写视图文件views.py，显示特定的Question的投票结果

```
01. def result(request, question_id):
02. response = "你正在查看问题[%s] 的结果。"
03. return HttpResponse(response % question_id)
```

### 步骤二：为第1步的三个视图编写URLCONF，通过相应的URL可以调用对应的视图函数：

1) 定义视图函数相关URL（网址），首先我们打开mysite/urls.py这个文件，规则修改为如下形式，将视图和polls.urls模块关联：

```
01. from django.conf.urls import url, include
02. from django.contrib import admin
03.
```

[Top](#)

```

04.
05. urlpatterns = [
06. url(r'^admin/', admin.site.urls),
07. url(r'^polls/', include('polls.urls'))
08.]

```

当客户端向你的网站请求一个页面时，Django将加载mysite.urls Python模块

因为它被指向ROOT\_URLCONF设置，它寻找名为urlpatterns的变量并按顺序匹配其中的正则表达式

2) 在 '^polls/' 找到匹配后，它将取消匹配的文本（“polls/”），并发送剩余的文本到'polls.urls'URLconf进行进一步处理。

polls.urls.py规则修改为如下形式：

```

01. from django.conf.urls import url
02. from . import views
03.
04. urlpatterns = [
05. url(r'^$', views.index, name='index'),
06. url(r'^(?P<question_id>[0-9]+)/$', views.detail, name='detail'),
07. url(r'^(?P<question_id>[0-9]+)/results/$', views.result, name='result'),
08. url(r'^(?P<question_id>[0-9]+)/vote/$', views.vote, name='vote'),
09.]

```

Django中的urls.py用的是正则进行匹配。

在浏览器打开 “/polls/12/” 。它会运行views.py文件中的detail()方法，并显示任何提供的URL内容。

再次尝试访问 “/polls/12/results/” 和 “/polls/12/vote/” –这将显示占位符结果和投票页面。

如果用户进入 “/polls/12/” ，在这个系统中Django会找到匹配'^polls/'，

然后Django会去掉匹配的文本("polls/"), 并发送剩余的文本（"12/"）到'polls.urls.py'文件。

URL配置用于进一步处理相匹配r'^(?P<question\_id>[0-9]+)/\$'从而调用detail()视图，如下所示：

```

01. detail(request=<HttpRequest object>, question_id='12')

```

question\_id='12' 是来自 (?P<question\_id>[0-9]+)的一部分，用周围的模式括号“[捕获](#)”匹配该模式文本，并将其作为参数传递给视图函数; ?P<question\_id> 定义了将被用来识别所述匹配的模式名称;以及[0-9]+正则表达式匹配一个数字序列(在一个数字)。

3)运行服务器，访问http://127.0.0.1:8000/polls/12/，结果显示如图-1所示：

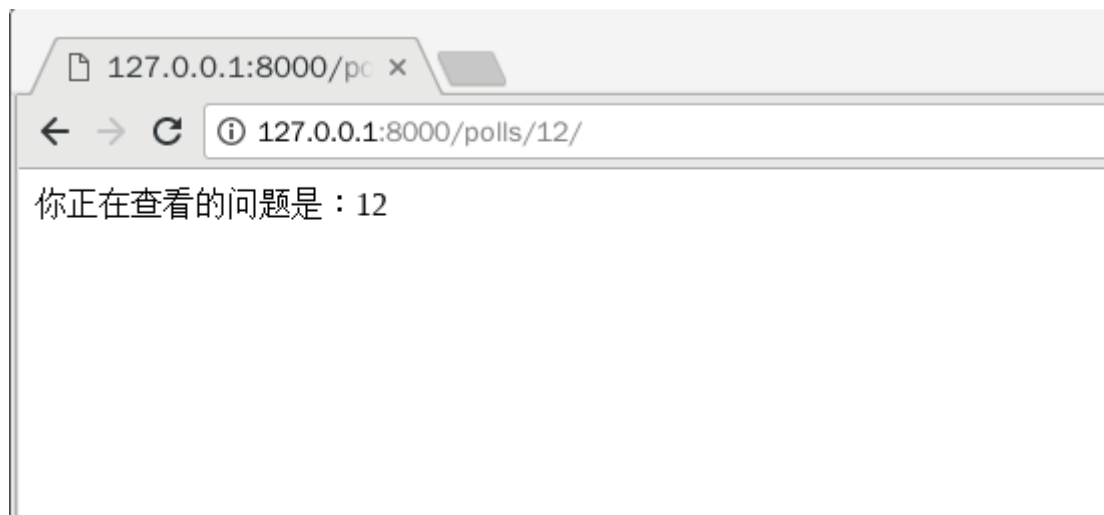


图-1

4)运行服务器，访问/http://127.0.0.1:8000/polls/12/results/，结果显示如图-2所示：



图-2

3)运行服务器，访问http://127.0.0.1:8000/polls/12/vote/，结果显示如图-3所示：

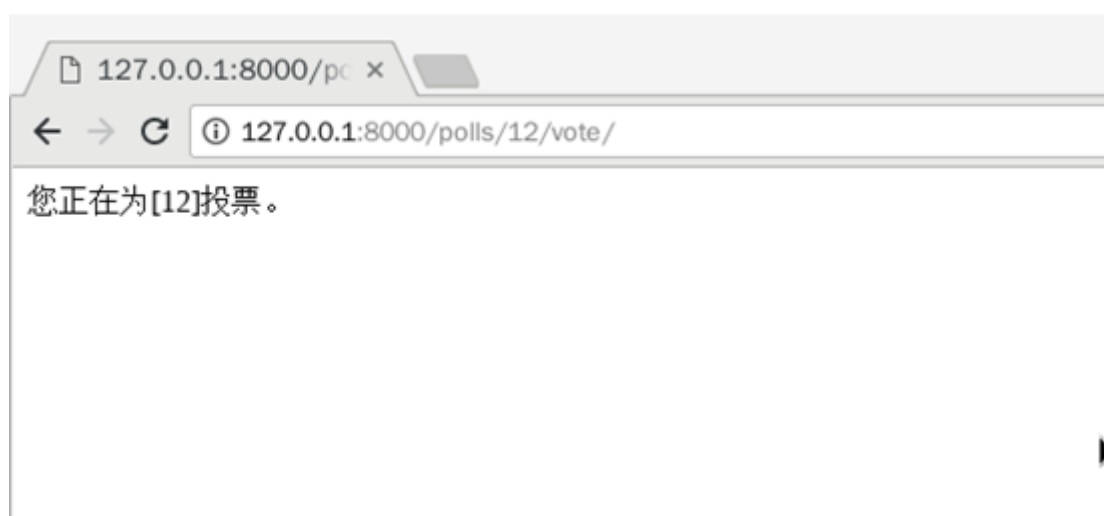


图-3

### 3 案例3：创建模板

[Top](#)

#### 3.1 问题

1. 为投票、投票结果、问题详情编写视图
2. 为投票、投票结果、问题详情编写模板
3. 访问相应url，观察是否是用到了正确的模板

## 3.2 步骤

实现此案例需要按照如下步骤进行。

### 步骤一：为投票、投票结果、问题详情编写视图

#### 1)创建模板工作目录

```
01 [root@localhost my site] # mkdir -p polls/templates/polls
```

此时，模板位于polls/templates/polls/目录下，目录结构如图-4所示：

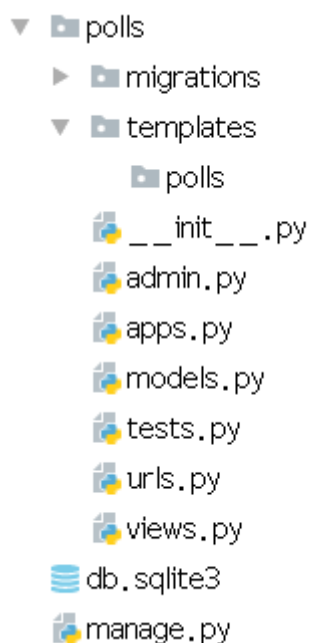


图-4

#### 2)配置文件设置，如图-5所示：

```
TEMPLATES = [
 {
 'BACKEND': 'django.template.backends.django.DjangoTemplates',
 'DIRS': [],
 'APP_DIRS': True,
 'OPTIONS': {
 'context_processors': [
 'django.template.context_processors.debug',
 'django.template.context_processors.request',
 'django.contrib.auth.context_processors.auth',
 'django.contrib.messages.context_processors.messages',
],
 },
 },
]
```

图-5

[Top](#)



设置文件settings.py配置了一个DjangoTemplates后端，其中将APP\_DIRS选项设置为True，此时，DjangoTemplates在INSTALLED\_APPS所包含的每个应用的目录下查找名为"templates"子目录

### 3)修改polls/views.py的detail函数，编写视图

当请求一个不存在的对象时，django将会抛出异常，一种常见的习惯是使用get()并在对象不存在时引发Http404。Django为此提供一个快捷方式，如下：

```
01. from django.shortcuts import render, get_object_or_404
02.
03. def detail(request, question_id):
04. question = get_object_or_404(Question, pk=question_id)
05. return render(request, 'polls/detail.html', {'question': question})
06.
```

get\_object\_or\_404()函数接受一个Django模型作为第一个参数和关键字任意参数数量，它传递到模型管理的get()函数。

如果对象不存在将引发HTTP404。

还有一个get\_list\_or\_404()函数，它的工作原理就像get\_object\_or\_404()-除了使用filter()而不是get()方法。如果列表是空的它会引发HTTP404。

。

## 步骤二：为投票、投票结果、问题详情编写模板

创建模板文件polls/templates/polls/detail.html，编写投票详情模板

```
01. <h1>{{ question.question_text }}</h1> #获取问题文本
02.
03. {% for choice in question.choice_set.all %}
04. {{ choice.choice_text }} #通过循环获取结果文本
05. {% endfor %}
06.
```

模板系统采用点查询语法来访问变量属性。

在这个例子{{question.question\_text}}，第一个Django确实在question对象字典查找。如果找不到，它再尝试属性查询，如果属性查找失败，它会尝试一个列表索引查找。

## 步骤三：访问相应url，观察是否是用到了正确的模板

1)运行服务器，访问http://127.0.0.1:8000/polls/3/，没有异常发生时的页面显示如图-6所示：

[Top](#)



图-6

2)运行服务器，访问`http://127.0.0.1:8000/polls/12/`，引发 404 错误，现在我们请求一个不存在问题，如图-7所示：

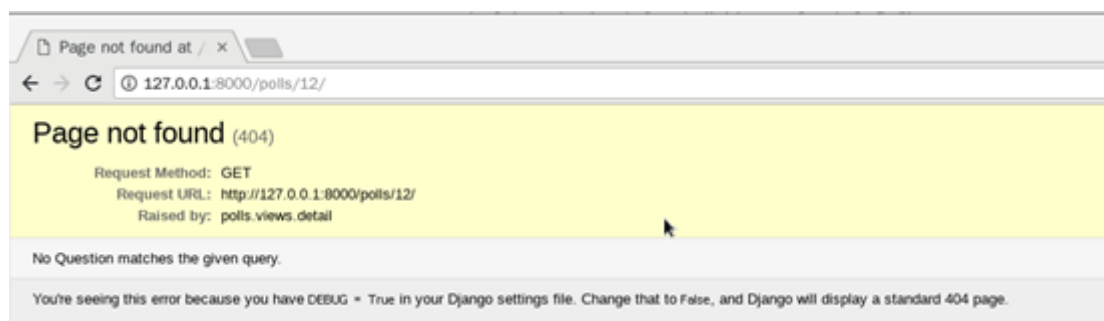


图-7

## 4 案例4：完成投票系统

### 4.1 问题

1. 为投票应用增加表单，使用户可以完成投票操作
2. 修改投票的视图函数
3. 修改模板文件
4. 使投票应用成为真正可用的程序

### 4.2 步骤

实现此案例需要按照如下步骤进行。

#### 步骤一：为投票应用增加表单，使用户可以完成投票操作

更新投票详细页面的模板`detail.html`表单

HTTP协议以“请求-回复”的方式工作。客户发送请求时，可以在请求中附加数据。服务器通过解析请求，就可以获得客户传来的数据，并根据URL来提供特定的服务。

```
01. <h1>{{ question.question_text }}</h1>
02. {% if error_message %} <p>{{ error_message }}</p>{% endif %}
03. <form action="/polls/{{ question.id }}/vote/" method="post">
```

[Top](#)

```

04. {% csrf_token %}
05. {% for choice in question.choice_set.all %}
06. <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}" %>
07. <label for="choice{{ forloop.counter }}">{{ choice.choice_text }} </label>

08. {% endfor %}
09. <input type="submit" value="投票" />
10. </form>

```

上面的模板显示每个问题选择一个单选按钮。每个单选按钮的值相联问题的选择编号。每个单选按钮的名称是“choice”。这意味着，当有人选择了其中一个单选按钮并提交表单，它会发送POST数据choice=#，其中#是被选择的选择的ID。这是HTML表单的基本概念。

我们设置表单的动作{%/polls/{{ question.id }}/vote/}，以及设置 method="post"。使用 method="post" (相对于 method="get") 是非常重要的，因为提交此表将改变服务器端数据的行为。当创建一个改变数据服务器端表单形式，使用 method="post"。

forloop.counter指示for标签已经循环多少次

因为我们正在创建一个POST形式（可以有修改数据的影响），我们需要担心跨站点请求伪造。但是也不必担心，因为Django自带了保护对抗的一个非常容易使用的系统。总之，这是针对内部URL所有的POST形式应该使用{% csrf\_token %}模板标签。

## 步骤二：修改投票的视图函数

### 1)修改polls/views.py的vote函数，编写视图

```

01. def vote(request, question_id):
02. question = get_object_or_404(Question, pk=question_id)
03. try:
04. choices = question.choice_set.get(pk=request.POST['choice'])
05. except (KeyError, Choice.DoesNotExist):
06. return render(request, 'polls/detail.html', {
07. 'question': question,
08. 'error_message': '您没有做出任何选择',
09. })
10. else:
11. choices.votes += 1
12. choices.save()
13. return redirect('result', question_id=question_id)

```

request.POST是一个类似于字典的对象，使您可以通过键名访问提交的数据。在这种情况下，request.POST['choice']返回被选择的choice的ID，作为字符串。request.POST的值总是字符串。[Top](#)

注意：Django还提供request.GET以相同的方式访问GET数据—但我们明确使用request.POST在我们的代码，以确保数据只能通过POST调用修改。

如果POST数据未提供choice，request.POST['choice']将引发KeyError异常。上面的代码检查KeyError异常和错误消息显示问题的表单，如果没有给出choice。

选择choice计数递增后，代码返回redirect重定向，的一个参数：用户将被重定向到URL。

2)修改polls/views.py的result函数，编写视图

```
01. def result(request, question_id):
02. question = get_object_or_404(Question, pk=question_id)
03. return render(request, 'polls/results.html', {'question': question})
```

### 步骤三：修改模板文件

创建polls/templates/polls/results.html模板文件

```
01. <h1>{{ question.question_text }}</h1>
02.
03. {% for choice in question.choice_set.all %}
04. {{ choice.choice_text }} : {{ choice.votes }}
05. {% endfor %}
06.
07. 投票首页
```

步骤四：投票程序展示如图-9、图-10：

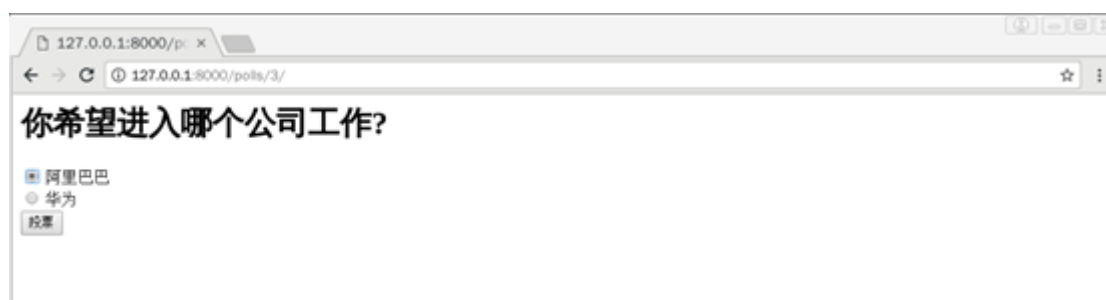


图-9

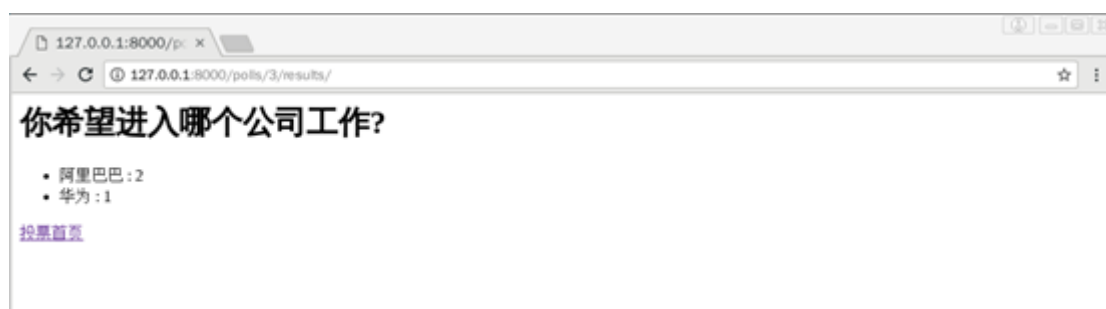


图-10

[Top](#)