编写一个简单的表单**[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial04/" \l "write-a-simple-form" \o "永久链接至标题)**

让我们更新一下在上一个教程中编写的投票详细页面的模板 ("polls/detail.html") ，让它包含一个 HTML **<form>** 元素：

polls/templates/polls/detail.html[**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial04/#id1)

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

{% **if** error\_message %}<**p**><**strong**>{{ error\_message }}</**strong**></**p**>{% **endif** %}

<**form** action="{% **url** 'polls:vote' question.id %}" method="post">

{% **csrf\_token** %}

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

<**input** type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}">

<**label** for="choice{{ forloop.counter }}">{{ choice.choice\_text }}</**label**><**br**>

{% **endfor** %}

<**input** type="submit" value="Vote">

</**form**>

简要说明：

* 上面的模板在 Question 的每个 Choice 前添加一个单选按钮。 每个单选按钮的 **value** 属性是对应的各个 Choice 的 ID。每个单选按钮的 **name** 是 **"choice"** 。这意味着，当有人选择一个单选按钮并提交表单提交时，它将发送一个 POST 数据 **choice=#** ，其中# 为选择的 Choice 的 ID。这是 HTML 表单的基本概念。
* 我们设置表单的 **action** 为 **{% url 'polls:vote' question.id %}** ，并设置**method="post"** 。使用 **method="post"``（与其相对的是 ``method="get"`）是非常重要的，因为这个提交表单的行为会改变服务器端的数据。 无论何时，当你需要创建一个改变服务器端数据的表单时，请使用 ``method="post"** 。这不是 Django 的特定技巧；这是优秀的网站开发技巧。
* **forloop.counter** 指示 [**for**](https://docs.djangoproject.com/zh-hans/2.1/ref/templates/builtins/#std:templatetag-for) 标签已经循环多少次。
* 由于我们创建一个 POST 表单（它具有修改数据的作用），所以我们需要小心跨站点请求伪造。 谢天谢地，你不必太过担心，因为 Django 已经拥有一个用来防御它的非常容易使用的系统。 简而言之，所有针对内部 URL 的 POST 表单都应该使用 [**{% csrf\_token %}**](https://docs.djangoproject.com/zh-hans/2.1/ref/templates/builtins/#std:templatetag-csrf_token) 模板标签。

现在，让我们来创建一个 Django 视图来处理提交的数据。记住，在 [教程第 3 部分](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/) 中，我们为投票应用创建了一个 URLconf ，包含这一行：

polls/urls.py[**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial04/#id2)

path('<int:question\_id>/vote/', views.vote, name='vote'),

我们还创建了一个 **vote()** 函数的虚拟实现。让我们来创建一个真实的版本。 将下面的代码添加到 **polls/views.py** ：

polls/views.py[**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial04/#id3)

**from** **django.http** **import** HttpResponse, HttpResponseRedirect

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

**from** **django.urls** **import** reverse

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

*# ...*

**def** vote(request, question\_id):

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

**try**:

selected\_choice = question.choice\_set.get(pk=request.POST['choice'])

**except** (**KeyError**, Choice.DoesNotExist):

*# Redisplay the question voting form.*

**return** render(request, 'polls/detail.html', {

'question': question,

'error\_message': "You didn't select a choice.",

})

**else**:

selected\_choice.votes += 1

selected\_choice.save()

*# Always return an HttpResponseRedirect after successfully dealing*

*# with POST data. This prevents data from being posted twice if a*

*# user hits the Back button.*

**return** HttpResponseRedirect(reverse('polls:results', args=(question.id,)))

以上代码中有些内容还未在本教程中提到过：

* [**request.POST**](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/#django.http.HttpRequest.POST) 是一个类字典对象，让你可以通过关键字的名字获取提交的数据。 这个例子中， **request.POST['choice']** 以字符串形式返回选择的 Choice 的 ID。 **[request.POST](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpRequest.POST" \o "django.http.HttpRequest.POST)** 的值永远是字符串。

注意，Django 还以同样的方式提供 **[request.GET](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpRequest.GET" \o "django.http.HttpRequest.GET)** 用于访问 GET 数据 —— 但我们在代码中显式地使用 **[request.POST](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpRequest.POST" \o "django.http.HttpRequest.POST)** ，以保证数据只能通过 POST 调用改动。

* 如果在 **request.POST['choice']** 数据中没有提供 **choice** ， POST 将引发一个 **[KeyError](https://docs.python.org/3/library/exceptions.html" \l "KeyError" \o "(在 Python v3.7))** 。上面的代码检查 **[KeyError](https://docs.python.org/3/library/exceptions.html" \l "KeyError" \o "(在 Python v3.7))** ，如果没有给出 **choice** 将重新显示 Question 表单和一个错误信息。
* 在增加 Choice 的得票数之后，代码返回一个 **[HttpResponseRedirect](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpResponseRedirect" \o "django.http.HttpResponseRedirect)** 而不是常用的 **[HttpResponse](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpResponse" \o "django.http.HttpResponse)** 、 **[HttpResponseRedirect](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpResponseRedirect" \o "django.http.HttpResponseRedirect)** 只接收一个参数：用户将要被重定向的 URL（请继续看下去，我们将会解释如何构造这个例子中的 URL）。

As the Python comment above points out, you should always return an **[HttpResponseRedirect](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpResponseRedirect" \o "django.http.HttpResponseRedirect)**after successfully dealing with POST data. This tip isn't specific to Django; it's just good Web development practice.

* 在这个例子中，我们在 **[HttpResponseRedirect](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpResponseRedirect" \o "django.http.HttpResponseRedirect)** 的构造函数中使用 **reverse()** 函数。这个函数避免了我们在视图函数中硬编码 URL。它需要我们给出我们想要跳转的视图的名字和该视图所对应的 URL 模式中需要给该视图提供的参数。 在本例中，使用在 [教程第 3 部分](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/) 中设定的 URLconf， **reverse()** 调用将返回一个这样的字符串：
* '/polls/3/results/'

其中 **3** 是 **question.id** 的值。重定向的 URL 将调用 **'results'** 视图来显示最终的页面。

正如在 [教程第 3 部分](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/) 中提到的，**[HttpRequest](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpRequest" \o "django.http.HttpRequest)** 是一个 **[HttpRequest](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpRequest" \o "django.http.HttpRequest)** 对象。更多关于 **[HttpRequest](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/" \l "django.http.HttpRequest" \o "django.http.HttpRequest)**对象的内容，请参见 [请求和响应的文档](https://docs.djangoproject.com/zh-hans/2.1/ref/request-response/) 。

当有人对 Question 进行投票后， **vote()** 视图将请求重定向到 Question 的结果界面。让我们来编写这个视图：

polls/views.py[**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial04/#id4)

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

**def** results(request, question\_id):

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

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

这和 [教程第 3 部分](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/) 中的 **detail()** 视图几乎一模一样。唯一的不同是模板的名字。 我们将在稍后解决这个冗余问题。

现在，创建一个 **polls/results.html** 模板：

polls/templates/polls/results.html[**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial04/#id5)

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

<**ul**>

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

<**li**>{{ choice.choice\_text }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}</**li**>

{% **endfor** %}

</**ul**>

<**a** href="{% **url** 'polls:detail' question.id %}">Vote again?</**a**>

现在，在你的浏览器中访问 **/polls/1/** 然后为 Question 投票。你应该看到一个投票结果页面，并且在你每次投票之后都会更新。 如果你提交时没有选择任何 Choice，你应该看到错误信息。

**注解**

我们的 **vote()** 视图代码有一个小问题。代码首先从数据库中获取了 **selected\_choice**对象，接着计算 **vote** 的新值，最后把值存回数据库。如果网站有两个方可同时投票在 *同一时间* ，可能会导致问题。同样的值，42，会被 **votes** 返回。然后，对于两个用户，新值43计算完毕，并被保存，但是期望值是44。

这个问题被称为 *竞争条件* 。如果你对此有兴趣，你可以阅读 [Avoiding race conditions using F()](https://docs.djangoproject.com/zh-hans/2.1/ref/models/expressions/#avoiding-race-conditions-using-f) 来学习如何解决这个问题。

## 使用通用视图：代码还是少点好[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial04/#use-generic-views-less-code-is-better)

**detail()** （在 [教程第3部分](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/) 中）和 **results()** 视图都很简单 —— 并且，像上面提到的那样，存在冗余问题。用来显示一个投票列表的 **index()** 视图（也在 [教程第 3 部分](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial03/) 中）和它们类似。

这些视图反映基本的 Web 开发中的一个常见情况：根据 URL 中的参数从数据库中获取数据、载入模板文件然后返回渲染后的模板。 由于这种情况特别常见，Django 提供一种快捷方式，叫做“通用视图”系统。

通用视图将常见的模式抽象化，可以使你在编写应用时甚至不需要编写Python代码。

让我们将我们的投票应用转换成使用通用视图系统，这样我们可以删除许多我们的代码。我们仅仅需要做以下几步来完成转换，我们将：

1. 转换 URLconf。
2. 删除一些旧的、不再需要的视图。
3. 基于 Django 的通用视图引入新的视图。

请继续阅读来了解详细信息。

**为什么要重构代码？**

一般来说，当编写一个 Django 应用时，你应该先评估一下通用视图是否可以解决你的问题，你应该在一开始使用它，而不是进行到一半时重构代码。本教程目前为止是有意将重点放在以“艰难的方式”编写视图，这是为将重点放在核心概念上。

就像在使用计算器之前你需要掌握基础数学一样。

### 改良 URLconf[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial04/" \l "amend-urlconf" \o "永久链接至标题)

首先，打开 **polls/urls.py** 这个 URLconf 并将它修改成：

polls/urls.py[**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial04/#id6)

**from** **django.urls** **import** path

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

app\_name = 'polls'

urlpatterns = [

path('', views.IndexView.as\_view(), name='index'),

path('<int:pk>/', views.DetailView.as\_view(), name='detail'),

path('<int:pk>/results/', views.ResultsView.as\_view(), name='results'),

path('<int:question\_id>/vote/', views.vote, name='vote'),

]

注意，第二个和第三个匹配准则中，路径字符串中匹配模式的名称已经由 **<question\_id>** 改为 **<pk>**。

### 改良视图[¶](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial04/#amend-views)

下一步，我们将删除旧的 **index**, **detail**, 和 **results** 视图，并用 Django 的通用视图代替。打开 **polls/views.py** 文件，并将它修改成：

polls/views.py[**¶**](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial04/#id7)

**from** **django.http** **import** HttpResponseRedirect

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

**from** **django.urls** **import** reverse

**from** **django.views** **import** generic

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

**class** **IndexView**(generic.ListView):

template\_name = 'polls/index.html'

context\_object\_name = 'latest\_question\_list'

**def** get\_queryset(self):

*"""Return the last five published questions."""*

**return** Question.objects.order\_by('-pub\_date')[:5]

**class** **DetailView**(generic.DetailView):

model = Question

template\_name = 'polls/detail.html'

**class** **ResultsView**(generic.DetailView):

model = Question

template\_name = 'polls/results.html'

**def** vote(request, question\_id):

... *# same as above, no changes needed.*

我们在这里使用两个通用视图： **[ListView](https://docs.djangoproject.com/zh-hans/2.1/ref/class-based-views/generic-display/" \l "django.views.generic.list.ListView" \o "django.views.generic.list.ListView)** 和 **[DetailView](https://docs.djangoproject.com/zh-hans/2.1/ref/class-based-views/generic-display/" \l "django.views.generic.detail.DetailView" \o "django.views.generic.detail.DetailView)** 。这两个视图分别抽象“显示一个对象列表”和“显示一个特定类型对象的详细信息页面”这两种概念。

* 每个通用视图需要知道它将作用于哪个模型。 这由 **model** 属性提供。
* [**DetailView**](https://docs.djangoproject.com/zh-hans/2.1/ref/class-based-views/generic-display/#django.views.generic.detail.DetailView) 期望从 URL 中捕获名为 **"pk"** 的主键值，所以我们为通用视图把 **question\_id** 改成 **pk** 。

默认情况下，通用视图 **[DetailView](https://docs.djangoproject.com/zh-hans/2.1/ref/class-based-views/generic-display/" \l "django.views.generic.detail.DetailView" \o "django.views.generic.detail.DetailView)** 使用一个叫做 **<app name>/<model name>\_detail.html** 的模板。在我们的例子中，它将使用 **"polls/question\_detail.html"** 模板。**template\_name** 属性是用来告诉 Django 使用一个指定的模板名字，而不是自动生成的默认名字。 我们也为 **results** 列表视图指定了 **template\_name** —— 这确保 results 视图和 detail 视图在渲染时具有不同的外观，即使它们在后台都是同一个 **[DetailView](https://docs.djangoproject.com/zh-hans/2.1/ref/class-based-views/generic-display/" \l "django.views.generic.detail.DetailView" \o "django.views.generic.detail.DetailView)** 。

类似地，[**ListView**](https://docs.djangoproject.com/zh-hans/2.1/ref/class-based-views/generic-display/#django.views.generic.list.ListView) 使用一个叫做 **<app name>/<model name>\_list.html** 的默认模板；我们使用 **template\_name** 来告诉 **[ListView](https://docs.djangoproject.com/zh-hans/2.1/ref/class-based-views/generic-display/" \l "django.views.generic.list.ListView" \o "django.views.generic.list.ListView)** 使用我们创建的已经存在的 **"polls/index.html"** 模板。

在之前的教程中，提供模板文件时都带有一个包含 **question** 和 **latest\_question\_list** 变量的 context。对于 **DetailView** ， **question** 变量会自动提供—— 因为我们使用 Django 的模型 (Question)， Django 能够为 context 变量决定一个合适的名字。然而对于 ListView， 自动生成的 context 变量是 **question\_list**。为了覆盖这个行为，我们提供 **context\_object\_name** 属性，表示我们想使用 **latest\_question\_list**。作为一种替换方案，你可以改变你的模板来匹配新的 context 变量 —— 这是一种更便捷的方法，告诉 Django 使用你想使用的变量名。

启动服务器，使用一下基于通用视图的新投票应用。

更多关于通用视图的详细信息，请查看 [通用视图的文档](https://docs.djangoproject.com/zh-hans/2.1/topics/class-based-views/)

当你对你所写的表单和通用视图感到满意后，请阅读 [教程的第 5 部分](https://docs.djangoproject.com/zh-hans/2.1/intro/tutorial05/) 来了解如何测试我们的投票应用。