編寫你的第一個 Django 應用，第 4 部分

這一篇從 教學第 3 部分 結尾的地方繼續講起。我們將繼續編寫投票應用，專注於表單處理並且精簡我們的程式。

> **從哪裡取得協助：**
>
> 如果你在閱讀本教學的過程中有任何疑問，可以前往FAQ的:doc:Getting Help</faq/help> 的小節。

編寫一個簡單的表單

讓我們更新一下在上一個教學中編寫的投票詳細頁面的範本 ("polls/detail.html")，讓它包含一個 HTML **<form>** 元素：

polls/templates/polls/detail.html

```
<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>
```

很簡單地概要說明：

- 上面的範本在 Question 的每個 Choice 前增加一個單選按鈕。 每個單選按鈕的 **value** 屬性是對應的各個 Choice 的 ID。每個單選按鈕的 **name**是**"choice"**。意思是，當有人選擇一個單選按鈕並提交表單提交時，它將發送一個 POST 資料**choice=#**，其中 # 為選擇的 Choice 的 ID。這是 HTML 表單的基本概念。
- 我們設定表單的 **action** 為 **{% url 'polls:vote' question.id %}** ，並設定 **method="post"** 。使用 **method="post"**（與其相對的是 **method="get"**）是非常重要的，因為這個提交表單的行為會改變伺服器端的資料。無論何時，當你需要建立一個改變伺服器端資料的表單時，使用 **method="post"**。這不是 Django 的特定技巧；這是優秀的網站開發技巧。
- **forloop.counter** 指示 **for** 標籤已經循環多少次。
- 由於我們建立一個 POST 表單（它具有修改資料的作用），所以我們需要小心跨網站請求偽造。這就要感謝由於 Django 內建了一個非常有用的防禦系統，因此你並不需要太過擔心。簡而言之，所有針對內部 URL 的 POST 表單都應該使用 **{% csrf_token %}** 範本標籤。

現在，讓我們來建立一個 Django 視圖來處理提交的資料。記住，在 教學第 3 部分 中，我們為投票應用建立了一個 URLconf，包含這一行：

polls/urls.py

```
path('<int:question_id>/vote/', views.vote, name='vote'),
```

我們還建立了一個 **vote()** 函數的虛擬實現。讓我們來建立一個真實的版本。 將下面的程式增加到 **polls/views.py**：

polls/views.py

```
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
```

```python
# ...
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):
        # 重新顯示問題投票表單。
        return render(request, 'polls/detail.html', {
            'question': question,
            'error_message': "You didn't select a choice.",
        })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        # 在成功處理 POST 資料後，緊跟著就回應 HttpResponseRedirect。
        # 這樣可以防止如果使用者按下 "上一頁" 按鈕時，
        # 資料會傳送兩次的情況。
        return HttpResponseRedirect(reverse('polls:results', args=(question.id,)))
```

以上程式中有些內容還未在本教學中提到過：

- **request.POST** 是一個類字典物件，讓你可以透過關鍵字的名字取得提交的資料。 這個例子中， **request.POST['choice']** 以字串形式回傳選擇的 Choice 的 ID。 **request.POST** 的值永遠是字串。

  注意，Django 還以同樣的方式提供 **request.GET** 用於開啟 GET 資料 – 但我們在程式中明確地使用 **request.POST** ，以保證資料只能透過 POST 呼叫改動。

- 如果在 **request.POST['choice']** 資料中沒有提供 **choice** ， POST 將引發一個 **KeyError** 。上面的程式檢查 **KeyError** ，如果沒有指定 **choice** 將重新顯示 Question 表單和一個錯誤資訊。

- 在增加 Choice 的得票數之後，程式回傳一個 **HttpResponseRedirect** 而不是常用的 **HttpResponse** 。 **HttpResponseRedirect** 只接收一個參數：用戶將要被重定向的 URL（請繼續看下去，我們將會解釋如何構造這個例子中的 URL）。

  正如上面的 Python 註釋所指出的那樣，在成功處理 POST 資料後，你應該總是返回一個 **HttpResponseRedirect** 。這個技巧並非專門針對 Django；總體而言，這是一項很好的 Web 開發實踐。

- 在這個例子中，我們在 **HttpResponseRedirect** 的構造函數中使用 **reverse()** 函數。這個函數避免了我們在視圖函數中用程式直接編寫 URL。它需要我們給出我們想要跳轉的視圖的名字和該視圖所對應的 URL 模式中需要給該視圖提供的參數。 在本例中，使用在 教學第 3 部分 中設定的 URLconf， **reverse()** 呼叫將回傳一個這樣的字串：

  ```
  '/polls/3/results/'
  ```

  其中 **3** 是 **question.id** 的值。重定向的 URL 將呼叫 **'results'** 視圖來顯示最終的頁面。

正如在 教學第 3 部分 中提到的， **HttpRequest** 是一個 **HttpRequest** 物件。更多關於 **HttpRequest** 物件的內容，請參見 請求和回應的文件 。

當有人對 Question 進行投票後， **vote()** 視圖將請求重定向到 Question 的結果界面。讓我們來編寫這個視圖：

polls/views.py

```python
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})
```

這和 教學第 3 部分 中的 **detail()** 視圖幾乎一模一樣。唯一的不同是範本的名字。 我們將在稍後解決這個重複項目。

現在，建立一個 **polls/results.html** 範本：

polls/templates/polls/results.html

```
<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>
```

現在，在你的瀏覽器中開啟 **/polls/1/** 然後為 Question 投票。你應該看到一個投票結果頁面，並且在你每次投票之後都會更新。 如果你提交時沒有選擇任何 Choice，你應該看到錯誤資訊。

注解

我們的 **vote()** 視圖程式有一個小問題。程式首先從資料庫中取得了 **selected_choice** 物件，接著計算 **vote** 的新值，最後把值存回資料庫。如果網站有兩個使用者可在 *同一時間* 同時投票，可能會導致問題。同樣的值，42，會被 **votes** 回傳。然後，對於兩個使用者，新值43計算完畢，並被儲存，但是期望值是44。

這個問題被稱為 *競爭條件*。如果你對此有興趣，你可以閱讀 Avoiding race conditions using F() 來學習如何解決這個問題。

使用通用視圖：程式還是少點好

detail()（在 教學第3部分 中）和 **results()** 視圖都很精簡 – 並且，像上面提到的那樣，存在重複項目。用來顯示一個投票欄表的 **index()** 視圖（也在 教學第 3 部分 中）和它們類似。

這些視圖反映基本的 Web 開發中的一個常見情況：根據 URL 中的參數從資料庫中取得資料、載入範本文件然後回傳實現後的範本。 由於這種情況特別常見，Django 提供一種快捷方式，叫做 “通用視圖” 系統。

通用視圖將常見的模式抽象化，可以使你在編寫應用時甚至不需要編寫 Python 程式。

讓我們將我們的投票應用轉換成使用通用視圖系統，這樣我們可以刪除許多我們的程式。我們僅僅需要做以下幾步來完成轉換，我們將：

1. 轉換 URLconf。
2. 刪除一些舊的、不再需要的視圖。
3. 基於 Django 的通用視圖引入新的視圖。

請繼續閱讀來了解詳細資訊。

為什麼要重構程式?

一般來說，當編寫一個 Django 應用時，你應該先評估一下通用視圖是否可以解決你的問題，你應該在一開始使用它，而不是進行到一半時重構程式。本教學目前為止是有意將重點放在以 “艱難的方式” 編寫視圖，這是為了將重點放在核心概念上。

就像在使用計算機之前你需要掌握基礎數學一樣。

改良 URLconf

首先，打開 **polls/urls.py** 這個 URLconf 並將它修改成：

polls/urls.py

```
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，第二個和第三個比對準則中，路徑字串中比對模式的名稱已經由 **<question_id>** 改為 **<pk>**。

改良視圖

下一步，我們將刪除舊的 **index**, **detail**, 和 **results** 視圖，並用 Django 的通用視圖代替。打開 **polls/views.py** 文件，並將它修改成：

polls/views.py

```
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):
        """傳回最後五個發佈的問題。"""
        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):
    ... # 沒有需要變更，和上面的內容相同。
```

我們在這裡使用兩個通用視圖： **ListView** 和 **DetailView** 。這兩個視圖分別抽象 “顯示一個物件欄表” 和 “顯示一個特定類型物件的詳細資訊頁面” 這兩種概念。

- 每個通用視圖需要知道它將作用於哪個模型。 這由 **model** 屬性提供。
- **DetailView** 期望從 URL 中擷取名為 **"pk"** 的主鍵值，所以我們為通用視圖把 **question_id** 改成 **pk** 。

預設情況下，通用視圖 **DetailView** 使用一個叫做 **<app name>/<model name>_detail.html** 的範本。在我們的例子中，它將使用 **"polls/question_detail.html"** 範本。**template_name** 屬性是用來告訴 Django 使用一個指定的範本名字，而不是自動產生的預設名

字。 我們也為 **`results`** 欄表視圖指定了 **`template_name`** – 這確保 results 視圖和 detail 視圖在實現時具有不同的外觀，即使它們在底層都是使用同一個 **`DetailView`** 。

類似地，**`ListView`** 使用一個叫做 **`<app name>/<model name>_list.html`** 的預設範本；我們使用 **`template_name`** 來告訴 **`ListView`** 使用我們建立的已經存在的 **`"polls/index.html"`** 範本。

在之前的教學中，提供範本文件時都帶有一個包含 **`question`** 和 **`latest_question_list`** 變數的 context。對於 **`DetailView`** ，**`question`** 變數會自動提供 – 因為我們使用 Django 的模型 (Question)，Django 能夠為 context 變數決定一個合適的名字。然而對於 ListView，自動產生的 context 變數是 **`question_list`**。為了覆蓋這個行為，我們提供 **`context_object_name`** 屬性，表示我們想使用 **`latest_question_list`**。作為一種替換方案，你可以改變你的範本來比對新的 context 變數 – 這是一種更便捷的方法，告訴 Django 使用你想使用的變數名。

啟動伺服器，使用一下基於通用視圖的新投票應用。

更多關於通用視圖的詳細資訊，請查看 通用視圖的文件

當你對你所寫的表單和通用視圖感到滿意後，請閱讀 教學的第 5 部分 來了解如何測試我們的投票應用。