編寫你的第一個 Django 應用，第 3 部分

這一篇從 教學第 2 部分 結尾的地方繼續講起。我們將繼續編寫投票應用，並且專注於如何建立公有界面 – 也被稱為“視圖”。

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。

概況

Django 中的「視圖」是 Django 應用程式中網頁的「樣式(type)」，通常具有特定功能並具有特定範本。例如，在一個部落格應用中，你可能會建立如下幾個視圖：

- 部落格首頁 – 顯示最近的幾項內容。
- 項目 “詳情” 頁 – 單個項目的永久連結頁面。
- 以年為單位的歸檔頁 – 顯示取得的年份裡各個月份建立的內容。
- 以月為單位的歸檔頁 – 顯示取得的月份裡各天建立的內容。
- 以天為單位的歸檔頁 – 顯示取得天裡建立的所有內容。
- 評論行為 – 用於對回應特定項目發布評論的處理。

而在我們的投票應用中，我們需要下欄幾個視圖：

- 問題 “索引” 頁 – 顯示最近的幾個投票問題。
- 問題 “詳情” 頁 – 顯示一個問題本文，不帶結果，但有一個投票表單。
- 問題 “結果” 頁 – 顯示某個投票的結果。
- 投票行為 – 用於對特定問題的特定選擇的投票處理。

在 Django 中，網頁和其他內容都是經由視圖表達的。每一個視圖均由一個 Python 函數（或者說方法，如果是在基於類別的視圖裡的話）來陳述內容。Django 將會根據使用者請求的 URL 來選擇使用哪個視圖（更準確的說，是根據 URL 中域名之後的部分）。

現在，在您上網的時候，您可能會遇到像是 **ME2/Sites/dirmod.htm?sid=&type=gen&mod=Core+Pages&gid=A6CD4967199A42D9B65B1B**之類優美語法。 您會很高興知道 Django 允許我們提供比這更優雅的 *URL 模式 (URL patterns)*。

URL模式是URL的一般形式 - 例如: **/newsarchive/<year>/<month>/**.

為了將 URL 和視圖關聯起來，Django 使用了 'URLconfs' 來設定。URLconf 將 URL 模式映射到視圖。

本教學只會介紹 URLconf 的基礎內容，你可以看看 URL調度器 以取得更多內容。

編寫更多視圖

現在讓我們向 **polls/views.py** 裡增加更多視圖。這些視圖有一些不同，因為他們接收參數：

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)
```

把這些新視圖增加進 **polls.urls** 模組裡，只要增加幾個 **url()** 函數呼叫就行：

polls/urls.py

```
from django.urls import path

from . import views

urlpatterns = [
    # ex: /polls/
    path('', views.index, name='index'),
    # ex: /polls/5/
    path('<int:question_id>/', views.detail, name='detail'),
    # ex: /polls/5/results/
    path('<int:question_id>/results/', views.results, name='results'),
    # ex: /polls/5/vote/
    path('<int:question_id>/vote/', views.vote, name='vote'),
]
```

然後看看你的瀏覽器，如果你轉到 "/polls/34/" ，Django 將會執行 **detail()** 方法並且顯示你在 URL 裡提供的問題 ID。再試試 "/polls/34/vote/" 和 "/polls/34/vote/" — 你將會看到暫時用於佔位的結果和投票頁。

當某人請求你網站的某一頁面時 — 例如像是， "/polls/34/" ，Django 將會載入 **mysite.urls** 模組，因為這在設定項 **ROOT_URLCONF** 中設定了。然後 Django 尋找名為 **urlpatterns** 變數並且按序比對正規表達式。在找到比對項 **'polls/'**，它切掉了比對的文字（**"polls/"**），將剩餘文字 — **"34/"**，發送至 'polls.urls' URLconf 做進一步處理。在這裡剩餘文字比對了 **'<int:question_id>/'**，使得我們 Django 以如下形式呼叫 **detail()**:

```
detail(request=<HttpRequest object>, question_id=34)
```

question_id=34 由 **<int:question_id>** 比對產生。使用尖括號"擷取"這部分 URL，且以關鍵字參數的形式發送給視圖函數。上述字串的 **:question_id>** 部分定義了將被用於區分比對模式的變數名，而 **int:** 則是一個轉換器決定了應該以什麼變數類型比對這部分的 URL 路徑。

為每個 URL 加上不必要的東西，例如 **.html** ，是沒有必要的。不過如果你非要加的話，也是可以的:

```
path('polls/latest.html', views.index),
```

但是，別這樣做，這太傻了。

寫一個真正有用的視圖

每個視圖必須要做的只有兩件事中的其中一個：回傳一個包含被請求頁面內容的 **HttpResponse** 物件，或者拋出一個類似於 **Http404** 的異常狀況，而剩下的就取決於你。

你的視圖可以從資料庫裡讀取記錄，可以使用一個範本引擎（例如 Django 內建的，或者其他第三方的），可以產生一個 PDF 文件，可以輸出一個 XML，建立一個 ZIP 文件，你可以做任何你想做的事，使用任何你想用的 Python 庫。

Django 只要求回傳的是一個 **HttpResponse** ，或者拋出一個異常。

因為 Django 內建的資料庫 API 很方便，我們曾在 教學第 2 部分 中學過，所以我們試試在視圖裡使用它。我們在 **index()** 函數裡插入了一些新內容，讓它能顯示資料庫裡以發布日期排序的最近 5 個投票問題，以空格分割：

polls/views.py

```python
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
```

這裡有個問題：頁面的設計直接以程式碼寫在視圖函數的程式裡的。如果你想改變頁面的樣子，你需要編輯 Python 程式。所以讓我們使用 Django 的範本系統，只要建立一個視圖，就可以將頁面的設計從程式中分離出來。

首先，在你的 **polls** 目錄裡建立一個 **templates** 目錄。Django 將會在這個目錄裡尋找範本文件。

你專案的 **TEMPLATES** 設定項描述了 Django 如何載入和實現範本。預設的設定文件設定了 **DjangoTemplates** 後端，並將 **APP_DIRS** 設定成了 True。這一選項將會讓 **DjangoTemplates** 在每個 **INSTALLED_APPS** 文件夾中尋找 "templates" 子目錄。這就是為什麼儘管我們沒有像在第二部分中那樣修改 DIRS 設定，Django 也能正確找到 polls 的範本位置的原因。

在你剛剛建立的 **templates** 目錄裡，再建立一個目錄 **polls**，然後在其中新建一個文件 **index.html** 。換句話說，你的範本文件的路徑應該是 **polls/templates/polls/index.html** 。因為 **app_directories** 範本載入器是透過上述描述的方法執行的，所以 Django 可以引用得到 **polls/index.html** 這一範本了。

範本命名空間

雖然我們現在可以將範本文件直接放在 **polls/templates** 文件夾中（而不是再建立一個 **polls** 子文件夾），但是這樣做不太好。Django 將會選擇第一個比對的範本文件，如果你有一個範本文件正好和另一個應用中的某個範本文件重名，Django 沒有辦法 *區分* 它們。我們需要協助 Django 選擇正確的範本，最好的方法就是把他們放入各自的 *命名空間* 中，也就是把這些範本放入一個和 *自身* 應用重名的子文件夾裡。

將下面的程式輸入到剛剛建立的範本文件中：

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 %}
```

注解

為了讓教學看起來不那麼長，所有的範本文件都只寫出了核心程式。在你自己建立的專案中，你應該使用 完整的 HTML 文件.

然後，讓我們更新一下 **polls/views.py** 裡的 **index** 視圖來使用範本：

polls/views.py

```python
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))
```

上述程式的作用是，載入 **polls/index.html** 範本文件，並且向它傳遞一個上下文(context)。這個上下文是一個字典，它將範本內的變數映射為 Python 物件。

用你的瀏覽器開啟 "/polls/" ，你將會看見一個無序清單，列出了我們在 教學第 2 部分 中增加的 “What's up” 投票問題，連結指向這個投票的詳情頁。

一個快捷函數： render()

載入範本、填入上下文再回傳由它產生的 **HttpResponse** 物件是一個非常常用的操作流程。於是 Django 提供了一個快捷函數，我們用它來重寫 **index()** 視圖：

polls/views.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)
```

注意到，我們不再需要匯入 **loader** 和 **HttpResponse** （不過如果你還有其他函數例如像是 **detail**, **results**, 和 **vote** 需要用到它的話，就需要保留匯入 **HttpResponse** ）。

此 **render()** 函數將對網頁的請求物件做為其第一個參數、一個範本名稱做為第二個參數，以及一個字典做為選擇性的第三個參數。它會將你所傳遞的上下文實現在範本後回傳一個 **HttpResponse** 物件。

拋出 404 錯誤

現在，我們來處理投票詳情視圖 – 它會顯示指定投票的問題本文。下面是這個視圖的程式：

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})
```

這裡的新觀念是：如果指定問題 ID 所對應的問題不存在，這個視圖就會拋出一個 **Http404** 異常。

我們稍後再討論你需要在 **polls/detail.html** 裡輸入什麼，但是如果你想試試上面這段程式是否正常運作的話，你可以暫時把下面這段寫進去：

polls/templates/polls/detail.html

```
{{ question }}
```

這樣你就能測試了。

一個快捷函數： get_object_or_404()

嘗試用 **get()** 函數取得一個物件，如果不存在就拋出 **Http404** 錯誤也是一個普遍的流程。Django 也提供了一個快捷函數，下面是修改後的詳情 **detail()** 視圖程式：

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})
```

此 **get_object_or_404()** 函數將 Django 模型作為第一個參數，並將任意數量的關鍵字參數傳遞給模型管理器的 **get()** 函數。 如果物件不存在，則會引發 **Http404** 異常狀況。

> **設計哲學**
>
> 為什麼我們使用輔助函數 **get_object_or_404()** 而不是自己擷取 **ObjectDoesNotExist** 異常呢？還有，為什麼模型 API 不直接拋出 **ObjectDoesNotExist** 而是拋出 **Http404** 呢？
>
> 因為那樣會將模型層耦合到視圖層。在 Django 的首要設計目標之一就是保持鬆散耦合。有些受控管的耦合則於 **django.shortcuts** 模組中採用。

另外也有 **get_list_or_404()** 函數，除了 **get()** 函數被換成了 **filter()** 函數，工作原理和 **get_object_or_404()** 一樣。如果清單為空的話會拋出 **Http404** 異常。

使用範本系統

回過頭去看看我們的 **detail()** 視圖。它向範本傳遞了上下文變數 **question** 。下面是 **polls/detail.html** 範本裡正式的程式：

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>
```

在範本系統中使用點符號來存取變數的屬性。在範例 **{{ question.question_text }}** 中，首先 Django 嘗試對 **question** 物件使用字典尋找（也就是使用 obj.get(str) 作業），如果失敗了就嘗試屬性尋找（也就是 obj.str 作業），結果是成功了。如果這一作業也失敗的話，將會

嘗試清單尋找（也就是 obj[int] 操作）。

方法（函數）呼叫發生在 **{% for %}** 循環中：**question.choice_set.all** 被解釋為 Python 程式 **question.choice_set.all()**，它會回傳一個可迭代的 **Choice** 物件，且該物件可以在 **{% for %}** 標籤內部使用。

查看 範本指南 可以了解關於範本的更多資訊。

去除範本中用程式直接編寫的 URL

還記得嗎，我們在 **polls/index.html** 裡編寫投票連結時，連結是用程式直接編寫的：

```
<li><a href="/polls/{{ question.id }}/">{{ question.question_text }}</a></li>
```

問題在於，這樣用程式直接編寫成緊密耦合的超連結，對於一個包含很多應用的專案來說，修改這樣的 URLs 是十分困難的。不過，由於你在 **polls.urls** 的 **url()** 函數中透過 name 參數為 URL 定義了名字，你可以使用 **{% url %}** 標籤替代它：

```
<li><a href="{% url 'detail' question.id %}">{{ question.question_text }}</a></li>
```

它的工作方式是透過查詢 **polls.urls** 模組中指定的 URL 定義來完成的。你可以在下面看到具有 'detail' 名稱的 URL 的明確定義：

```
...
# 由 {% url %} 範本標籤呼叫使用的 'name' 值
path('<int:question_id>/', views.detail, name='detail'),
...
```

如果你想改變投票詳情視圖的 URL，例如想改成 **polls/specifics/12/**，你不用在範本裡修改任何東西（包括其它範本），只要在 **polls/urls.py** 裡稍微修改一下就行：

```
...
# 新增 'specifics' 這個字
path('specifics/<int:question_id>/', views.detail, name='detail'),
...
```

為 URL 名稱增加命名空間

教學專案只有一個應用，**polls**。在一個真實的 Django 專案中，可能會有五個，十個，二十個，甚至更多應用。Django 如何分辨重名的 URL 呢？舉個例子，**polls** 應用有 **detail** 視圖，可能另一個部落格應用也有同名的視圖。Django 如何知道 **{% url %}** 標籤到底對應哪一個應用的 URL 呢？

答案是：在根 URLconf 中增加命名空間。在 **polls/urls.py** 文件中稍作修改，加上 **app_name** 設定命名空間：

polls/urls.py

```
from django.urls import path

from . import views

app_name = 'polls'
urlpatterns = [
    path('', views.index, name='index'),
    path('<int:question_id>/', views.detail, name='detail'),
```

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

現在，編輯 **polls/index.html** 文件，從：

polls/templates/polls/index.html

```
<li><a href="{% url 'detail' question.id %}">{{ question.question_text }}</a></li>
```

修改為指向具有命名空間的詳細視圖：

polls/templates/polls/index.html

```
<li><a href="{% url 'polls:detail' question.id %}">{{ question.question_text }}</a></li>
```

當你對你寫的視圖感到滿意後，請閱讀 教學的第 4 部分 了解基礎的表單處理和通用視圖。

‹ 編寫你的第一個 Django 應用，第 2 部分

編寫你的第一個 Django 應用，第 4 部分 ›