編寫你的第一個 Django 應用，第 2 部分

這部分教學從 教學第 1 部分 結尾的地方繼續講起。我們將建立資料庫，建立您的第一個模型，並主要說明 Django 提供的自動產生的管理頁面。

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。

資料庫設定

現在，打開 **mysite/settings.py** 。這是個包含了 Django 專案設定的 Python 模組。

通常，這個設定文件使用 SQLite 作為預設資料庫。如果你不熟悉資料庫，或者只是想嘗試一下 Django，這是最簡單的選擇。Python 內置 SQLite，所以你無需安裝額外東西來使用它。當你開始一個真正的專案時，你可能更傾向使用一個更具擴展性的資料庫，例如 PostgreSQL，避免中途切換資料庫這個令人頭疼的問題。

如果你想使用其他資料庫，你需要安裝合適的 database bindings ，然後改變設定文件中 **DATABASES** **'default'** 專案中的一些鍵值：

- **ENGINE** -- 可選值有 **'django.db.backends.sqlite3'**， **'django.db.backends.postgresql'**， **'django.db.backends.mysql'**，或 **'django.db.backends.oracle'**。其它 可使用的後端資料庫。
- **NAME** - 資料庫的名稱。如果使用的是 SQLite，資料庫將是你電腦上的一個文件，在這種情況下， **NAME** 應該是此文件的絕對路徑，包括文件名稱。預設值 **os.path.join(BASE_DIR, 'db.sqlite3')** 將會把資料庫文件儲存在專案的根目錄。

如果你不使用 SQLite，則必須增加一些額外設定，例如 **USER** 、 **PASSWORD** 、 **HOST** 等等。想了解更多資料庫設定方面的內容，請參考文件： **DATABASES** 。

SQLite 以外的其它資料庫

如果你使用了 SQLite 以外的資料庫，請確認在使用前已經建立了資料庫。你可以透過在你的資料庫交互式命令欄中使用 "**CREATE DATABASE database_name;**" 命令來完成這件事。

另外，還要確保該資料庫用戶中提供 **mysite/settings.py** 具有 "create database" 權限。這使得自動建立的 test database 能被以後的教學使用。

如果你使用 SQLite，那麼你不需要在使用前做任何事——資料庫會在需要的時候自動建立。

編輯 **mysite/settings.py** 文件前，先設定 **TIME_ZONE** 為你自己時區。

此外，說明一下文件頂部的 **INSTALLED_APPS** 設定項。這裡包括了會在你專案中啟用的所有 Django 應用程式。應用程式能在多個專案中使用，你也可以包裝並且發佈應用程式，讓別人使用它們。

通常， **INSTALLED_APPS** 預設包括了以下 Django 的內建應用：

- **django.contrib.admin** -- 管理員網站， 你很快就會使用它。
- **django.contrib.auth** -- 認證授權系統。

- **django.contrib.contenttypes** -- 內容類型框架。
- **django.contrib.sessions** -- 會議框架。
- **django.contrib.messages** -- 訊息框架。
- **django.contrib.staticfiles** -- 管理靜態文件的框架。

這些應用程式被預設啟用是為了給常見的專案提供方便。

預設開啟的某些應用程式需要至少一個資料表，所以，在使用他們之前需要在資料庫中建立一些表。請執行以下命令：

```
$ python manage.py migrate
```

這個 **migrate** 命令檢查 **INSTALLED_APPS** 設定，為其中的每個應用程式建立需要的資料表，至於具體會建立什麼，這取決於你的 **mysite/settings.py** 設定文件和每個應用程式的資料庫遷移文件（我們稍後會介紹這個）。這個命令所執行的每個遷移操作都會在終端中顯示出來。如果你感興趣的話，執行你資料庫的命令欄工具，並輸入 **\dt** (PostgreSQL)， **SHOW TABLES;** (MariaDB,MySQL)， **.schema** (SQLite)或者 **SELECT TABLE_NAME FROM USER_TABLES;** (Oracle) 來看看 Django 到底建立了哪些表。

寫給極簡主義者

就像之前說的，為了方便大多數專案，我們預設啟用了一些應用程式，但並不是每個人都需要它們。如果你不需要某個或某些應用程式，你可以在執行 **migrate** 前毫無顧慮地從 **INSTALLED_APPS** 裡注釋或者刪除掉它們。 **migrate** 命令只會為在 **INSTALLED_APPS** 裡聲明了的應用程式進行資料庫遷移。

建立模型

在 Django 裡寫一個資料庫驅動的 Web 應用程式的第一步是定義模型 - 也就是資料庫結構設計和附加的其它資料摘要。

設計哲學

模型是真實資料的簡單明確的描述。它包含了儲存的資料所必要的欄位和行為。Django 遵循 DRY Principle 。它的目標是在一個地方定義資料模型，然後其相關的程式由模型自動產生。

來介紹一下遷移 - 舉個例子，不像 Ruby On Rails，Django 的遷移程式是由你的模型文件自動產生的，它本質上是個歷史記錄，Django 可以用它來進行資料庫的滾動更新，透過這種方式使其能夠和當前的模型比對。

在這個投票應用程式中，需要建立兩個模型：問題 **Question** 和選項 **Choice**。**Question** 模型包括問題描述和發佈時間。**Choice** 模型有兩個欄位，選項描述和當前得票數。每個選項屬於一個問題。

這些概念可以透過一個 Python 類別來描述。按照下面的例子來編輯 **polls/models.py** 文件：

polls/models.py

```
from django.db import models


class 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)
```

每個模型被表示為 **django.db.models.Model** 類別的子類別。每個模型有許多類別變數，它們都表示模型裡的一個資料庫欄位。

每個欄位都是 **Field** 類別的實例 - 例如，字串欄位被表示為 **CharField** ，日期時間欄位被表示為 **DateTimeField** 。這將告訴 Django 每個欄位要處理的資料類型。

每個 **Field** 類別實例變數的名字（例如 **question_text** 或 **pub_date** ）也是欄位名稱，所以最好使用對機器友好的格式。你將會在 Python 程式裡使用它們，而資料庫會將它們作為欄名稱。

你可以使用可選的選項來為 **Field** 定義一個人類可讀的名字。這個功能在很多 Django 內部組成部分中都被使用了，而且作為文件的一部分。如果某個欄位沒有提供此名稱，Django 將會使用對機器友好的名稱，也就是變數名稱。在上面的例子中，我們只為 **Question.pub_date** 定義了對人類友好的名字。對於模型內的其它欄位，它們的機器友好名稱也會被作為人類友好名稱使用。

定義某些 **Field** 類別實例需要參數。例如 **CharField** 需要一個 **max_length** 參數。這個參數的用途不僅止於用來定義資料庫結構，也用於驗證資料，我們稍後將會看到這方面的內容。

Field 也能夠接收多個可選參數；在上面的例子中：我們將 **votes** 的 **default** 也就是預設值，設為0。

注意在最後，我們使用 **ForeignKey** 定義了一個關聯。這將告訴 Django，每個 **Choice** 物件都關聯到一個 **Question** 物件。Django 支援所有常用的資料庫關聯：多對一、多對多和一對一。

啟用模型

上面的一小段用於建立模型的程式給了 Django 很多資訊，透過這些資訊，Django 可以：

- 為這個應用程式建立資料庫 schema（產生 **CREATE TABLE** 語句）。
- 建立可以與 **Question** 和 **Choice** 物件進行交互的 Python 資料庫 API。

但是首先得把 **polls** 應用程式安裝到我們的專案裡。

> **設計哲學**
>
> Django 應用程式是“可插拔”的。你可以在多個專案中使用同一個應用程式。除此之外，你還可以發佈自己的應用程式，因為它們並不會被綁定到當前的 Django 安裝上。

為了在我們的建置中包含這個應用程式，我們需要在設定類別 **INSTALLED_APPS** 中增加設定。因為 **PollsConfig** 類別寫在文件 **polls/apps.py** 中，所以它的點式路徑是 **'polls.apps.PollsConfig'**。在文件 **mysite/settings.py** 中 **INSTALLED_APPS** 子項增加點式路徑後，它看起來像這樣：

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',
]
```

現在你的 Django 專案會包含 **polls** 應用程式。接著執行下面的命令：

```
$ python manage.py makemigrations polls
```

你將會看到類似於下面這樣的輸出：

```
Migrations for 'polls':
  polls/migrations/0001_initial.py
    - Create model Question
    - Create model Choice
```

透過執行 **makemigrations** 命令，Django 會檢測你對模型文件的修改（在目前這個情況下，你已經剛建立了一個新的模型），並且把修改的部分儲存為一次 遷移。

遷移是 Django 對於模型定義（也就是你的資料庫結構）的變化的儲存形式 - 它們其實也只是一些你磁碟上的文件。如果你想的話，你可以閱讀一下你模型的遷移資料，它被儲存在 **polls/migrations/0001_initial.py** 裡。別擔心，你不需要每次都閱讀遷移文件，但是它們被設計成人類可讀的形式，這是為了便於你手動調整 Django 的修改方式。

Django 有一個自動執行資料庫遷移並同步管理你的資料庫結構的命令 - 這個命令是 **migrate**，我們馬上就會接觸它 - 但是首先，讓我們看看遷移命令會執行哪些 SQL 語句。**sqlmigrate** 命令取得遷移的名稱，然後回傳對應的 SQL 語句：

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

你將會看到類似下面這樣的輸出（我把輸出重組成了人類可讀的格式）：

```
BEGIN;
--
-- 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
);
--
```

```
-- Create model Choice
--
CREATE TABLE "polls_choice" (
    "id" serial NOT NULL PRIMARY KEY,
    "choice_text" varchar(200) NOT NULL,
    "votes" integer NOT NULL,
    "question_id" integer NOT NULL
);
ALTER TABLE "polls_choice"
  ADD CONSTRAINT "polls_choice_question_id_c5b4b260_fk_polls_question_id"
    FOREIGN KEY ("question_id")
    REFERENCES "polls_question" ("id")
    DEFERRABLE INITIALLY DEFERRED;
CREATE INDEX "polls_choice_question_id_c5b4b260" ON "polls_choice" ("question_id");

COMMIT;
```

請注意以下幾點：

- 輸出的內容和你使用的資料庫有關，上面的輸出範例使用的是 PostgreSQL。
- 資料庫的表名稱是結合了應用程式名稱 (**polls**) 和資料庫模型的英文小寫名稱 – **question** 和 **choice** 組合而成的。（如果需要，你可以覆寫這個定義。）
- 主鍵 (IDs) 會被自動建立。(你也可以覆寫這個定義。)
- 依照慣例，Django 會在外鍵欄位名稱後追加字串 **"_id"**。（是，這個定義也可以覆寫。）
- 外鍵關聯由 **FOREIGN KEY** 產生。你不用關心 **DEFERRABLE** 部分，它只是告訴 PostgreSQL，請在整個交易過程全部執行完之後再建立外鍵關聯。
- 產生的 SQL 語句是使用你選用的資料庫客製化成的，所以那些和資料庫有關的欄位類型，例如 **auto_increment** (MySQL)、 **serial** (PostgreSQL)和 **integer primary key autoincrement** (SQLite)，Django 會幫你自動處理。那些和引號相關的事情 – 例如，是使用單引號還是雙引號 – 也一樣會被自動處理。
- 這個 **sqlmigrate** 命令並沒有真正在你的資料庫中的執行遷移 - 相反，它只是把命令輸出到螢幕上，讓你看看 Django 認為需要執行哪些 SQL 語句。這在你想看看 Django 到底準備做什麼，或者當你是資料庫管理員，需要寫腳本來批量處理資料庫時會很有用。

如果你有興趣，你也可以試試看執行 **python manage.py check**；這個命令協助你檢查專案中的問題，並且在檢查過程中不會對資料庫進行任何操作。

現在，再次執行 **migrate** 命令，在資料庫裡建立新定義的模型的資料表：

```
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Rendering model states... DONE
  Applying polls.0001_initial... OK
```

這個 **migrate** 命令取得所有還沒有執行過的遷移（Django 透過在資料庫中建立一個特殊的表 **django_migrations** 來追蹤執行過哪些遷移）並套用在資料庫上 - 也就是將你對模型的更改同步到資料庫結構上。

遷移是非常強大的功能，它能讓你在開發過程中持續的改變資料庫結構而不需要重新刪除和建立表 - 它專注於使資料庫即時更新而不會遺失資料。我們會在後面的教學中更加深入的學習這部分內容，現在，你只需要記住，改變模型需要這三個步驟：

- 編輯 **models.py** 文件，改變模型。
- 執行 **python manage.py makemigrations** 為模型的改變產生遷移文件。
- 執行 **python manage.py migrate** 來套用資料庫遷移。

資料庫遷移被分解成產生和套用兩個命令是因為你將可以在版本控制系統上與你的應用程式一起進行遷移確認；這不僅僅會讓開發更加簡單，也給別的開發者和生產環境中的使用帶來方便。

透過閱讀文件 Django 管理文件，你可以取得關於 **manage.py** 工具的更多資訊。

初試 API

現在讓我們進入交互式 Python 命令欄，嘗試一下 Django 為你建立的各種 API。透過以下命令打開 Python 命令欄：

```
$ python manage.py shell
```

我們改用這個命令而不是簡單的輸入 "Python" 是因為 **manage.py** 會設定 **DJANGO_SETTINGS_MODULE** 環境變數，這個變數會讓 Django 根據 **mysite/settings.py** 文件來設定 Python 套件的導入路徑。

當你成功進入命令欄後，來試試 database API 吧：

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

# 系統中還沒有問題。
>>> Question.objects.all()
<QuerySet []>

# 建立一個新的問題。
# 在預設的設定文件中啟用了對時區的支援，因此
# Django 預期 pub_date 與 tzinfo 的日期時間。使用 timezone.now()
# 而不是 datetime.datetime.now() 然後它就會它會做出正確的事情。
>>> from django.utils import timezone
>>> q = Question(question_text="What's new?", pub_date=timezone.now())

# 將物件儲存到資料庫中。 您必須明確地呼叫 save() 函式。
>>> q.save()

# 現在它有一個 ID.
>>> q.id
1

# 透過 Python 屬性存取模型字串值。
>>> q.question_text
"What's new?"
```

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

# 透過更改屬性來更改值，然後呼叫 save()函式。
>>> q.question_text = "What's up?"
>>> q.save()

# objects.all() 顯示資料庫中的所有的問題。
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
```

等等。**<Question: Question object (1)>** 對於我們了解這個物件的細節沒什麼協助。讓我們透過編輯 **Question** 模型的程式（位於 **polls/models.py** 中）來修正這個問題。給 **Question** 和 **Choice** 增加 **__str__()** 方法。

polls/models.py

```
from django.db import models

class Question(models.Model):
    # ...
    def __str__(self):
        return self.question_text

class Choice(models.Model):
    # ...
    def __str__(self):
        return self.choice_text
```

給模型增加 **__str__()** 方法是很重要的，這不僅僅能給你在命令欄裡使用帶來方便，Django 自動產生的 admin 裡也使用這個方法來表示物件。

讓我們再為此模型增加一個自定義方法：

polls/models.py

```
import datetime

from django.db import models
from django.utils import timezone


class Question(models.Model):
    # ...
    def was_published_recently(self):
        return self.pub_date >= timezone.now() - datetime.timedelta(days=1)
```

新加入的 **import datetime** 和 **from django.utils import timezone** 分別導入了 Python 的標準 **datetime** 模組和 Django 中和時區相關的 **django.utils.timezone** 工具模組。如果你不太熟悉 Python 中的時區處理，看看 時區支援文件 吧。

儲存文件然後透過 **python manage.py shell** 命令再次打開 Python 交互式命令欄：

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

# 確認我們新增的 __str__() 有發揮功效。
>>> Question.objects.all()
<QuerySet [<Question: What's up?>]>

# Django 提供了一個豐富的資料庫查找 API，該 API 完全由
# 關鍵字參數來驅動。
>>> Question.objects.filter(id=1)
<QuerySet [<Question: What's up?>]>
>>> Question.objects.filter(question_text__startswith='What')
<QuerySet [<Question: What's up?>]>

# 取得今年發佈的 question。
>>> from django.utils import timezone
>>> current_year = timezone.now().year
>>> Question.objects.get(pub_date__year=current_year)
<Question: What's up?>

# 要求一個不存在的 ID，這將引起異常狀況。
>>> Question.objects.get(id=2)
Traceback (most recent call last):
    ...
DoesNotExist: Question matching query does not exist.

# 透過主鍵搜尋是最常見的情況，因此 Django 提供了一個
# 主鍵精確搜尋的捷徑。
# 以下與 Question.objects.get(id=1) 是相同的。
>>> Question.objects.get(pk=1)
<Question: What's up?>

# 確認我們的自定方法有運作。
>>> q = Question.objects.get(pk=1)
>>> q.was_published_recently()
True

# 給 Question 幾個 Choices。 這個 create 呼叫會構造一個新的
# Choice 物件，執行 INSERT 語句，將 choice 新增到一個
# 可用選項並回傳新的 Choice 物件集合。 Django 建立
# 用於儲存 ForeignKey 關係的 "另一面" 的集合
# (例如問題的選擇)，因而可以透過 API 進行存取。
>>> q = Question.objects.get(pk=1)

# 顯示相關物件集合的所有 choices -- 到目前為止還沒有。
>>> q.choice_set.all()
<QuerySet []>

# 建立三個 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 物件可以透過 API 存取其相關的 Question 物件。
>>> c.question
<Question: What's up?>

# 反之亦然：Question 物件可以存取 Choice 物件。
>>> q.choice_set.all()
<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]>
>>> q.choice_set.count()
3

# API 會根據您的需要自動遵循關係。
# 使用雙下劃線分隔關係。
# 這可以根據需要進行任意深度的工作；沒有限制。
# 查詢今年 pub_date 為任何 Question 的所有 Choice
# (重複使用我們上面建立的 'current_year' 變數)。
>>> Choice.objects.filter(question__pub_date__year=current_year)
<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]>

# 讓我們刪除其中一個選項。 為此我們使用 delete() 函式。
>>> c = q.choice_set.filter(choice_text__startswith='Just hacking')
>>> c.delete()
```

閱讀 開啟關聯物件 文件可以獲取關於資料庫關聯的更多內容。想知道關於雙下劃線的更多用法，參見 查找欄位 文件。資料庫 API 的所有細節可以在 資料庫 API 參考 文件中找到。

介紹 Django 管理頁面

設計哲學

為你的員工或客戶產生一個用戶增加，修改和刪除內容的管理是一項缺乏創造性和乏味的工作。因此，Django 全自動地根據模型建立管理界面。

Django 產生於一個公眾頁面和內容發佈者頁面完全分離的新聞類網站的開發過程中。網站管理人員使用管理系統來增加新聞、事件和體育時訊等，這些增加的內容被顯示在公眾頁面上。Django 透過為網站管理人員建立統一的內容編輯界面解決了這個問題。

管理界面不是為了網站的開啟者，而是為管理者準備的。

建立一個管理員賬號

首先，我們得建立一個能登錄管理頁面的用戶。請執行下面的命令：

```
$ python manage.py createsuperuser
```

鍵入你想要使用的使用者，然後按下 Enter 鍵：

```
Username: admin
```

然後提示你輸入想要使用的郵件地址：

```
Email address: admin@example.com
```

最後一步是輸入密碼。你會被要求輸入兩次密碼，第二次的目的是為了確認第一次輸入的確實是你想要的密碼。

```
Password: **********
Password (again): *********
Superuser created successfully.
```

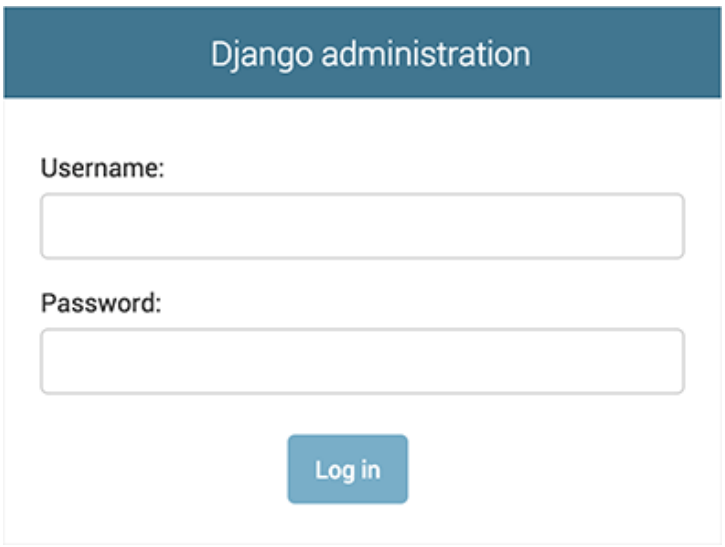
啟動開發伺服器

Django 的管理界面預設就是啟用的。讓我們啟動開發伺服器，看看它到底是什麼樣子。

如果開發伺服器未啟動，用以下命令啟動它：

```
$ python manage.py runserver
```

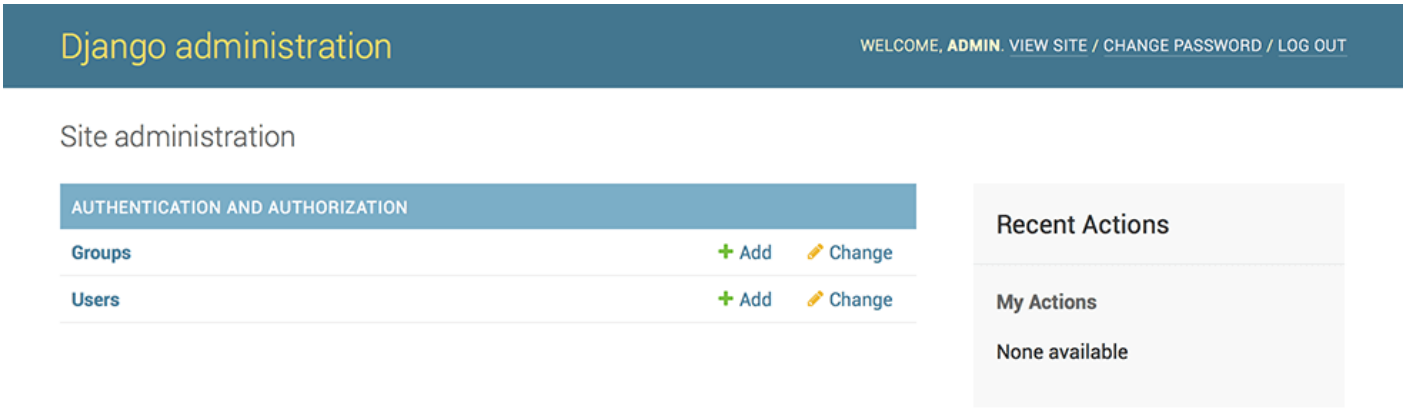
現在，打開瀏覽器，轉到你本地域名稱的 "/admin/" 目錄， -- 例如 "http://127.0.0.1:8000/admin/" 。你應該會看見管理員登錄界面：



由於 翻譯功能 預設為打開狀態，因此如果您設定了 **LANGUAGE_CODE**，則登錄畫面將會以指定的語言顯示(如果 Django 有適當的翻譯)。

進入管理網站頁面

現在，試著使用你在上一步中建立的超級用戶來登錄。然後你將會看到 Django 管理頁面的索引頁：



你將會看到幾種可編輯的內容：群組和使用者。它們是由 **`django.contrib.auth`** 提供的，這是 Django 開發的認證框架。

向管理頁面中加入投票應用程式

但是我們的投票應用程式在哪呢？它沒在索引頁面裡顯示。

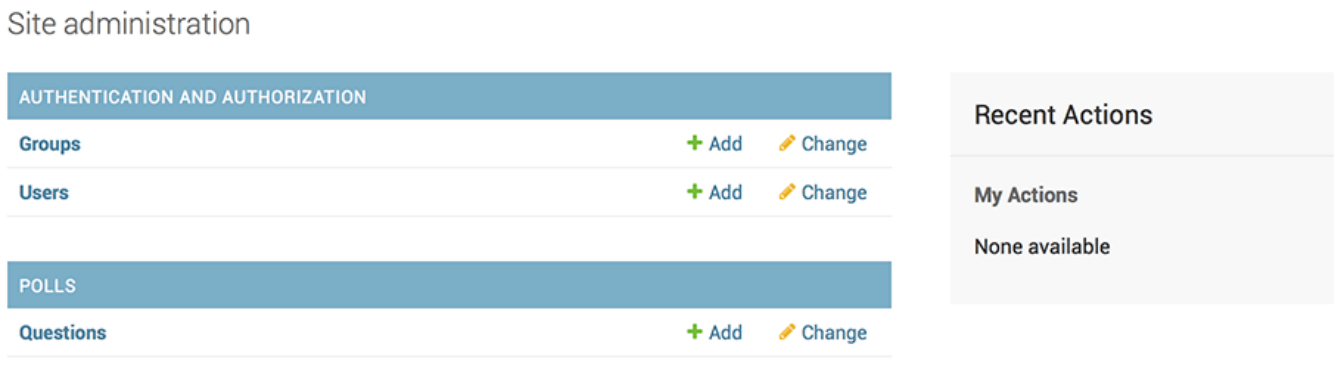
只需要再做一件事：我們得告訴管理員，問題 **`Question`** 物件需要一個管理接口。打開 **`polls/admin.py`** 文件，把它編輯成下面這樣：

polls/admin.py

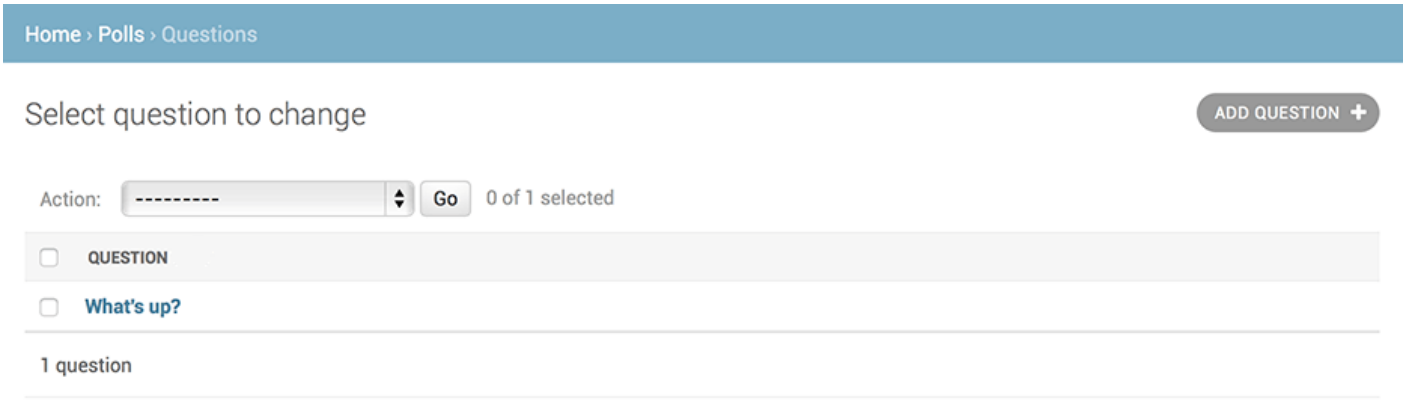
```
from django.contrib import admin

from .models import Question

admin.site.register(Question)
```

體驗便捷的管理功能

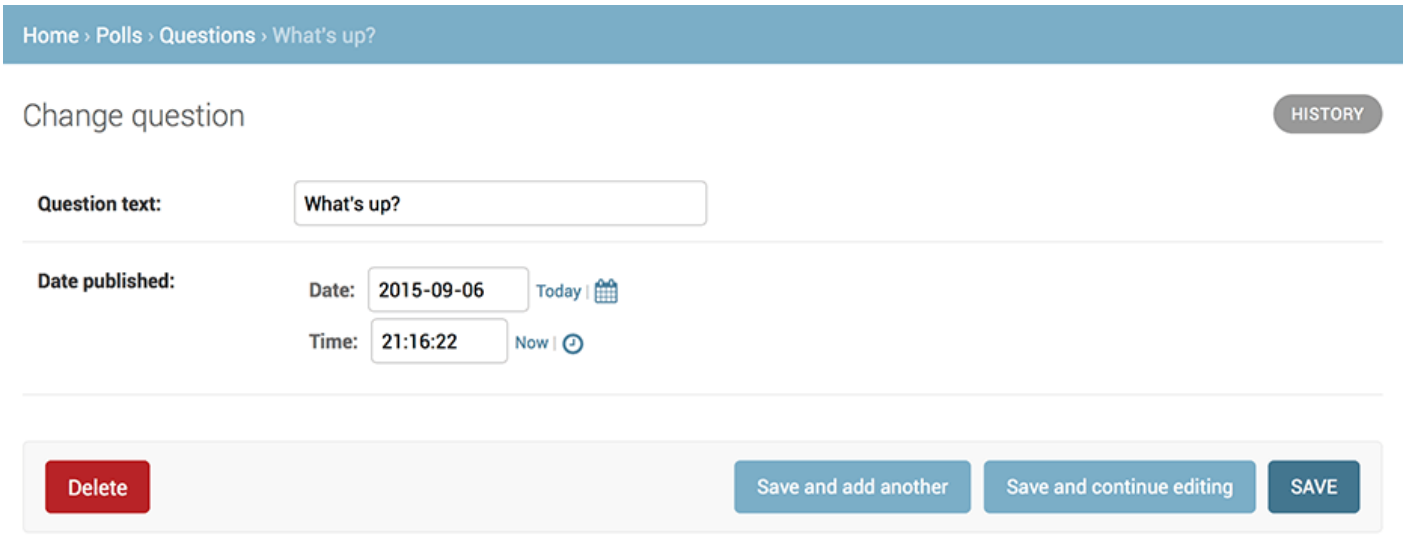
現在我們向管理頁面注冊了問題 **`Question`** 類別。Django 知道它應該被顯示在索引頁裡：



點擊 "Questions" 。現在看到是問題 "Questions" 物件的欄表 "change list" 。這個界面會顯示所有資料庫裡的問題 Question 物件，你可以選擇一個來修改。這裡現在有我們在上一部分中建立的 “What's up?” 問題。



點擊 “What's up?” 來編輯這個問題（Question）物件：



注意事項：

- 這個表單是從問題 **`Question`** 模型中自動產生的
- 不同的欄位類型（日期時間欄位 **`DateTimeField`** 、字串欄位 **`CharField`**）會產生對應的 HTML 輸入控件。每個類型的欄位都知道它們該如何在管理頁面裡顯示自己。
- 每個日期時間欄位 **`DateTimeField`** 都有 JavaScript 寫的快捷按鈕。日期有轉到今天（Today）的快捷按鈕和一個彈出式日曆界面。時間有設為現在（Now）的快捷按鈕和一個欄出常用時間的方便的彈出式欄表。

頁面的底部提供了幾個選項：

- 儲存（Save） - 儲存改變，然後回傳物件欄表。
- 儲存並繼續編輯（Save and continue editing） - 儲存改變，然後重新載入當前物件的修改界面。
- 儲存並新增（Save and add another） - 儲存改變，然後增加一個新的空物件並載入修改界面。
- 刪除（Delete） - 顯示一個確認刪除頁面。

如果顯示的 “發佈日期(Date Published)” 和你在 教學 1 裡建立它們的時間不一致，這意味著你可能沒有正確的設定 **`TIME_ZONE`** 。改變設定，然後重新載入頁面看看是否顯示了正確的值。

透過點擊 “今天(Today)” 和 “現在(Now)” 按鈕改變 “發佈日期(Date Published)”。然後點擊 “儲存並繼續編輯(Save and add another)”按鈕。然後點擊右上角的 “歷史(History)”按鈕。你會看到一個列出了所有透過 Django 管理頁面對當前物件進行的改變的頁面，其中列出了時間戳和進行修改操作的使用者：

當你熟悉了資料庫 API 之後，你就可以開始閱讀 教學第 3 部分 ，下一部分我們將會學習如何為投票應用程式增加更多視圖。

‹ 編寫你的第一個 Django 應用程式，第 1 部分

編寫你的第一個 Django 應用程式，第 3 部分 ›