記事
Toshihiko Minamoto · 2022年10月19日 9m read

Open Exchange

Django 入門 パート 1

しばらく前、IRIS 用 Django の新しいドライバーを導入しました。 そこで実際に、IRIS で Django を使用する方法を見てみましょう。

重要な注意事項: IRIS Community Edition で Django を使用してもほぼまったく動作しません。Community Edition には利用できる接続が 5 つしかなく、Django がすぐに使い果たしてしまうためです。 残念ながらこの理由により、ライセンスの使用状況を予測するのが難しいため、新しいアプリケーションの開発にはこの方法をお勧めできません。

Django プロジェクトを開始する

新しい Django プロジェクトを開始しましょう。そのためにはまず、Django そのものをインストールする必要があります。

```
pip install django
```

次に、demo という名前のプロジェクトを作成します。プロジェクトフォルダが同じ名前で作成されます。

```
django-admin startproject demo
```

```
cd demo
```

または、既存のフォルダを使用することもできます。

```
django-admin startproject main .
```

このコマンドにより、いくつかの Python ファイルが作られます。

ファイルの説明:

- **manage.py:** この Django プロジェクトを様々な方法で操作するためのコマンドラインユーティリティ
- **main** ディレクトリ: プロジェクトの実際の Python パッケージ
- **main/init.py:** このディレクトリを Python パッケージと見なすように Python に伝達する空のファイル
- **main/settings.py**: この Django プロジェクトの設定/構成
- **main/urls.py**: この Django プロジェクトの URL 宣言。Django で作られたサイトの「目次」。
- **main/asci.py**: プロジェクトにサービスを提供するための ASGI 対応ウェブサーバーのエントリポイント。
- **main/wsci.py**: プロジェクトにサービスを提供するための WSGI 対応ウェブサーバーのエントリポイント。

このポイントからでもプロジェクトを開始することが可能で、何とか機能します。

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

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced). You have 18 unapplied migration(s). Y
our project may not work properly until you apply the migrations for app(s): admin, a
uth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
July 22, 2022 - 15:24:12
Django version 4.0.6, using settings 'main.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

ブラウザに移動して、URL http://127.0.0.1:8000 を開きます。

IRIS を追加する

IRIS へのアクセスを追加しましょう。それには、プロジェクトに依存関係をいくつかインストールする必要があります. 正しい方法は、*以下の内容で、django を依存関係として追加する requirements.txt* というファイルに定義することです。

```
# Django itself
django>=4.0.0
```

次は、公開されている Django の IRIS 用ドライバーです。 *残念ながら、InterSystems には独自のドライバーを PyPI に公開する意向がないため、以下の方法でそれを定義しなければなりません。*
これがいつ削除されるかわからないため、将来的には動作しない可能性もあるので注意してください。 （pypi にあるのであれば、django-iris
の依存関係としてインストールされ、明示的に定義される必要がないかもしれません）

```
# InterSystems IRIS driver for Django, and DB-API driver from InterSystems
django-iris==0.1.13
https://raw.githubusercontent.com/intersystems-community/iris-driver-
distribution/main/DB-API/intersystems_irispython-3.2.0-py3-none-any.whl
```

このファイルに定義されている依存関係を次のコマンドでインストールします。

```
pip install -r requirements.txt
```

次に、プロジェクトが IRIS を使用するように構成できます。それには、*settings.py* ファイルの **DATABASES** パラメーターを以下のような行で更新する必要があります。NAME は IRIS のネームスペース、PORT は IRIS を利用できる SuperPort を指します。

```
DATABASES = {
    'default': {
        'ENGINE': 'django_iris',
        'NAME': 'USER',
        'USER': '_SYSTEM',
        'PASSWORD': 'SYS',
        'HOST': 'localhost',
        'PORT': 1982,
    }
}
```

Django には ORM と、プロジェクトに格納されたモデルがあり、Django モデルと Database をテーブルとして同期する必要があります。 デフォルトでは、auth に関連するモデルがいくつかあります。 ここで、migrate を実行できます。

```
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

IRIS に移動すると、追加のテーブルがあります。

さらにモデルを定義する

では、モデルを追加しましょう。 それには、次のような内容で *models.py,* という新しいファイルを追加します。

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    dob = models.DateField()
    sex = models.BooleanField()
```

見てのとおり、様々な型のフィールがあります。 次に、このモデルを Database 用に準備する必要がありますが、 その前に、**main** プロジェクトを *settings.py* の **INSTALLED_APPS** に追加します。

```
INSTALLED_APPS = [
    ....
    'main',
]
```

次に、makemigrations を行います。 このコマンドは、モデルに変更があった後に呼び出される必要があります。 モデルの過去の変更に対応し、インストールされているアプリケーションのバージョンに関係なく、migration はデータベースのスキーマをどのように更新すべきかを認識しています。

```
$ python manage.py makemigrations main
Migrations for 'main':
  main/migrations/0001_initial.py
    - Create model Person
```

もう一度 migrate を実行しますが、以前の migrations
がすでに実行されたことをわかっているため、新しいものだけが実行されます。

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

そして実際に、SQL ビューから migration がどのように見えるかを確認できます。

```
$ python manage.py sqlmigrate main 0001
--
-- Create model Person
--
CREATE TABLE "main_person" ("id" BIGINT AUTO_INCREMENT NOT NULL PRIMARY KEY, "first_n
ame" VARCHAR(30) NULL, "last_name" VARCHAR(30) NULL, "dob" DATE NOT NULL, "sex" BIT N
OT NULL);
```

ただし、作業アプリケーションがすでに存在する場合などには、データベースにすでに存在するテーブルにアクセスすることが可能です。 zpm package posts-and-tags がインストールされているので、community.posts
テーブルのモデルを作成しましょう。

```
$ python manage.py inspectdb community.post
# This is an auto-generated Django model module.
# You'll have to do the following manually to clean this up:
#   * Rearrange models' order
#   * Make sure each model has one field with primary_key=True
#   * Make sure each ForeignKey and OneToOneField has `on_delete` set to the desired
behavior
#   * Remove `managed = False` lines if you wish to allow Django to create, modify, a
nd delete the table
# Feel free to rename the models, but don't rename db_table values or field names.
from django.db import models

class CommunityPost(models.Model):
    id = models.AutoField(db_column='ID')  # Field name made lowercase.
    acceptedanswerts = models.DateTimeField(db_column='AcceptedAnswerTS', blank=True,
 null=True)  # Field name made lowercase.
    author = models.CharField(db_column='Author', max_length=50, blank=True, null=Tru
e)  # Field name made lowercase.
    avgvote = models.IntegerField(db_column='AvgVote', blank=True, null=True)  # Fiel
d name made lowercase.
    commentsamount = models.IntegerField(db_column='CommentsAmount', blank=True, null
=True)  # Field name made lowercase.
    created = models.DateTimeField(db_column='Created', blank=True, null=True)  # Fie
ld name made lowercase.
    deleted = models.BooleanField(db_column='Deleted', blank=True, null=True)  # Fiel
d name made lowercase.
    favscount = models.IntegerField(db_column='FavsCount', blank=True, null=True)  #
Field name made lowercase.
```

```
    hascorrectanswer = models.BooleanField(db_column='HasCorrectAnswer', blank=True,
null=True)  # Field name made lowercase.
    hash = models.CharField(db_column='Hash', max_length=50, blank=True, null=True)
# Field name made lowercase.
    lang = models.CharField(db_column='Lang', max_length=50, blank=True, null=True)
# Field name made lowercase.
    name = models.CharField(db_column='Name', max_length=250, blank=True, null=True)
 # Field name made lowercase.
    nid = models.IntegerField(db_column='Nid', primary_key=True)  # Field name made l
owercase.
    posttype = models.CharField(db_column='PostType', max_length=50, blank=True, null
=True)  # Field name made lowercase.
    published = models.BooleanField(db_column='Published', blank=True, null=True)  #
Field name made lowercase.
    publisheddate = models.DateTimeField(db_column='PublishedDate', blank=True, null=
True)  # Field name made lowercase.
    subscount = models.IntegerField(db_column='SubsCount', blank=True, null=True)  #
Field name made lowercase.
    tags = models.CharField(db_column='Tags', max_length=350, blank=True, null=True)
 # Field name made lowercase.
    text = models.CharField(db_column='Text', max_length=-1, blank=True, null=True)
# Field name made lowercase.
    translated = models.BooleanField(db_column='Translated', blank=True, null=True)
# Field name made lowercase.
    type = models.CharField(db_column='Type', max_length=50, blank=True, null=True)
# Field name made lowercase.
    views = models.IntegerField(db_column='Views', blank=True, null=True)  # Field na
me made lowercase.
    votesamount = models.IntegerField(db_column='VotesAmount', blank=True, null=True)
  # Field name made lowercase.

    class Meta:
        managed = False
        db_table = 'community.post'
```

managed = False にマークされているため、makemigrations と migrate はこのテーブルで動作しません。
テーブル名を省略すると、Django
が以前に作成したテーブルも含む、大規模なモジュールのリストが生成されます。

#Python #InterSystems IRIS
InterSystems Open Exchangeで関連アプリケーションを確認してください

ソースURL:
https://jp.community.intersystems.com/post/django-%E5%85%A5%E9%96%80-%E3%83%91%E3%83%BC%E3%83%88-1