day23 Django Channels、Async 和 Celery 的協同之舞: DocuMind 專案介紹

先前文章的重點放在Django對於資料庫的ORM（Object Relational Mapping），後台應用，還有Django REST framework（DRF）等API操作。這些主要都是單純的數據更新、單向的服務器推送需求。但是如果想要用Django進行其他的Web應用，我們需要更好的解決方案與應用

在這個系列文章中，我們會透過DocuMind這個專案來探討Django在面對需要非同步、實時雙向通訊以及長時間任務下的情境，應該如何來進行實踐

今日重點：

- 專案介紹：DocuMind
- 探討啟動Django專案的不同起手式
    - Django runserver
    - WSGI (Web Server Gateway Interface)啟動Django
    - ASGI (Asynchronous Server Gateway Interface)啟動Django
    - 適用場景
- 建立專案

專案介紹：DocuMind

來為大家介紹這個文章系列的主角：DocuMind，取名自Document與Mind的結合

這個專案最主要的功能就是個人的知識庫，具有以下功能：

- 用戶能夠自己輸入相關的文章，又或是透過上傳PDF檔案的方式到知識庫中
- 知識庫除了會將文檔存在資料庫之外，也會定期將文章內容傳到向量資料庫中，轉換成向量資料
- 用戶能傳訊息給知識庫，知識庫根據訊息來從向量資料庫返回接近的答案返回給客戶端

撇除轉換成向量資料，跟判斷用戶訊息與文庫的相關程度之外，我們需要來看一下我們需要克服哪些技術面的需求：

1. 定期將資料庫資料轉換成向量資料，除了需要安排排程處理之外，這種長時間的任務勢必會長期阻塞我們的服務
2. 用戶跟服務端這樣來回交互訊息，勢必要有實時通訊以及維持通訊的機制
3. 服務端收到訊息，經過一些業務處理，最後返回相關資料給客戶端。如果只用單純的同步處理流程，除了造成阻塞之外，耗時也會更長，可能需要一些非同步的手段

綜上所述，我們勢必要引入一些新技術了！

- 使用Django celery來幫助我們處理長時間任務
- 使用Django channel來支持WebSocket，以及實時通訊
- 使用Django async視圖來讓整體業務流程更有效率及順暢

在目前爲止，我們單純的使用Django提供的runserver來作為快速開發的後端伺服器。但是為了滿足更多的需求，我們需要進一部探討如果要成功建立起Web應用，還有哪些啟動專案的方式

探討啟動Django專案的不同起手式

這邊來深入了解 Django runserver、WSGI 和 ASGI 這三種不同的伺服器啟動方式，幫助理解它們的區別、優缺點以及適用場景

Django runserver

最快速，且適合開發中除錯的啟動方式

特點：

- 在開發環境的輕量級伺服器，透過Django內置的Web伺服器實現
- 在默認情況下是多線程，可以透過添加 `--nothreading` 來變成單線程
- 檢測到程式碼更改時會自動重啟
- 提供詳細的錯誤頁面與日誌

缺點：

- 沒有經過安全性審核與性能測試，不適合生產環境

這讓我想到初學時還妄想可以在背景執行runserver然後不用進行部署XD 對於程式小白來說部署真的是惡夢

WSGI (Web Server Gateway Interface)啟動Django

生產環境中的標準接口

特點：

- 通過WSGI伺服器（uWSGI或是Gunicorn等）來運行Django
- 通常是多進程或是再額外配置多線程
- 同步處理：每個請求在一個請求中完整處理
- 具備良好的性能與廣範的部署選項
- 在CPU密集型的應用性能較佳

缺點：

- 不支持WebSocket等長連接協議
- 對於高併發I/O密集型應用可能不夠高效，即是配置了多線程處理

uWSGI的設定檔配置如下（不含所有常見配置）：

```
[uwsgi]
socket = 127.0.0.1:xxxx
processes = 2 # 設定多少進程
threads = 4 # 設定多少進程
master = true
vacuum = true
buffer-size=65536
pidfile=uwsgi.pid
uid=1000
gid=1000
max-requests = 5000 # 限制每個worker處理5000個請求後重啟 防止記憶體泄
reload-on-rss = 300 # 當worker使用超過350MB記憶體時重啟
```

ASGI (Asynchronous Server Gateway Interface)啟動Django

支持非同步處理的新一代接口標準

特點：

- 支持非同步處理
- 通過ASGI伺服器（Daphne或是Uvicorn）來運行Django
- 一些ASGI伺服器支持多進程+協程的混合模式，結合多進程的CPU利用率與協程的高效I/O處理
- 支持WebSocket等長連接協議

缺點：

- 對於CPU密集型的任務，可能表現不如WSGI伺服器有優勢

Uvicorn的啟動文檔範例如下：

```
# 綁定的 IP 和端口
bind = "0.0.0.0:8000"
```

```
# 工作進程數
workers = 3

# 使用支持 ASGI 的 worker 類
worker_class = "uvicorn.workers.UvicornWorker"

# WebSocket 相關設置
timeout = 60
keepalive = 65
graceful_timeout = 300
max_requests = 1000
max_requests_jitter = 1000



# 根據環境變數動態設置工作進程數
import os

workers = int(os.environ.get("GUNICORN_WORKERS", "3"))
```

適用場景

- runserver適合本地開發與測試→開發時的優先選擇
- WSGI則是應用於傳統的Web應用→部落格等不需要實時更新的應用
- ASGI適合需要實時功能或是高併發I/O的現代Web應用→電商或是聊天室，需要實時接收並更新消息

我們可以用更貼近生活的例子來說明：

Django runserver

想像一下這是一家 24/7 營業的小咖啡館

- 隨叫隨到，菜單天天更新（自動重載）

- 老闆（你）親自沖泡每一杯咖啡（處理請求）
- 有最直觀的事故報告書（錯誤頁面）

優點：

- 老闆就是你，想怎麼改菜單就怎麼改
- 不需要考慮開店位置，準備好就開

缺點：

- 要是來了 100 個顧客，店家可能會崩潰
- 絕對不要在真正的商業街上開這種店XD

WSGI

這是一家連鎖咖啡店

- 多個咖啡師（進程/線程）同時工作
- 服務雖然普通，但是穩定值得信賴
- 對於高峰時段的客流能夠應付（高併發）
- 每個咖啡師都有自己的工作台（進程隔離）

缺點：

- 不太靈活，想要增加新品種（如奶茶）需要大工程

ASGI

這是一家使用最新科技的現代化咖啡館

- 高科技設備，能夠快速處理各種飲品訂單
- 一台超級智能咖啡機（事件循環）同時處理多個訂單
- 無論是美式、拿鐵還是奶茶，都能同時搞定
- 即使來了 1000 個顧客點 1000 種不同的飲料，也能從容應對

- 不只做咖啡，還能製作各種特殊飲品（WebSocket 等）

缺點：

- 操作可能有點複雜，需要專業培訓。
- 對於只想喝普通美式的顧客可能有點小題大做。

可能會有疑問？既然都使用了多線程，那這樣怎麼不好處理高併發？除了不支持WebSocket外，ASGI的啟動方式跟WSGI還差在哪裡？

我們還是可以拿這個咖啡廳當舉例：

因為python有具備GIL（全局鎖）的特性，所以即使是多線程，一次能站在櫃檯的工作人員只有一人。並且因為線程的特性，即使知道這是只要等咖啡滴完就能泡好的咖啡，咖啡師也會在那邊等咖啡滴完而不去做其他事情。但是在ASGI使用協程的特性，所以即使還是有GIL的限制，每一個工作人員都能好好的填滿所有的工作時間，不會有乾等的狀況發生

不過使用uWSGI等服務，還是能夠有效的緩解乾等的狀況，會有Master（店經理）來指揮那些只會乾等的咖啡師，在一些I/O密集型，會釋放GIL的任務中，uWSGI的確很夠用。但是多線程之間的切換還是比協程耗資源，同時使用協程不用像多線程那樣，每一個任務（每一杯咖啡）都要分一個線程（咖啡師）

經過上面的比較下來，顯然根據我們DocuMind專案的需求，使用ASGI來啟動我們的專案是更適合的選項！當然這是指最後階段，開發過程中還是使用Django runserver來進行開發

建立專案

專案的程式碼：https://github.com/class83108/DocuMind

而今天的範例程式碼則在分支中：https://github.com/class83108/DocuMind/tree/init

目前專案只完成到celery的部分，所以README等都還不會是最終版本。如果最後看完系列文章，覺得這個專案很有趣的話，請不要吝嗇給予星星喔！

- 安裝套件

```
# 建立專案
poetry new DocuMind
cd DocuMind

poetry add Django==4.2
poetry add psycopg2-binary
poetry add load-dotenv
poetry add redis
poetry add django-redis
```

- 建立專案

```
# 啟動虛擬環境
poetry shell

# 建立專案
django-admin startproject documind
```

- 建立環境變數

在與`manage.py`同級目錄建立.env，並配置相對應的參數

```
# .env

DB_HOST=127.0.0.1
DB_USER=<DB_USER>
DB_PWD=<DB_PWD>
```

```
DB_NAME=<DB_NAME>

SECRET_KEY='<your key>'

REDIS_HOST=127.0.0.1
REDIS_PORT=6379
REDIS_PASSWORD=<REDIS_PASSWORD>

DEBUG=true
```

```
# settings.py

from dotenv import load_dotenv

import os

ENV_FILE_PATH = BASE_DIR / ".env"

# 加載 .env 文件
load_dotenv(ENV_FILE_PATH)


# postgreSQL 相關設定
DB_HOST = os.getenv("DB_HOST")
DB_USER = os.getenv("DB_USER")
DB_PWD = os.getenv("DB_PWD")
DB_NAME = os.getenv("DB_NAME")

# REDIS 相關
REDIS_HOST = os.getenv("REDIS_HOST")
REDIS_PORT = os.getenv("REDIS_PORT")
REDIS_PASSWORD = os.getenv("REDIS_PASSWORD")

TEMPLATES = [
    {
```

```
        "BACKEND": "django.template.backends.django.DjangoTempla
        "DIRS": [os.path.join(BASE_DIR, "templates/")],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.mess
            ],
        },
    },
]

DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql",
        "NAME": DB_NAME,
        "USER": DB_USER,
        "PASSWORD": DB_PWD,
        "HOST": DB_HOST,
        "PORT": "5432",
    },
}

CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": f"redis://:{REDIS_PASSWORD}@{REDIS_HOST}:{RE
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        },
    }
}

STATIC_URL = "static/"
```

```
STATICFILES_DIRS = [os.path.join(BASE_DIR, "static")]
STATIC_ROOT = os.path.join(BASE_DIR, "staticfiles")


MEDIA_ROOT = os.path.join(BASE_DIR, "uploads")
MEDIA_URL = "/media/"
```

- 建立相關應用

```
python3 manage.py startapp articles
```

- 建立相關模型與註冊

```
# articles.models.py

from django.db import models
from django.contrib.auth import get_user_model

User = get_user_model()


class Article(models.Model):
    title = models.CharField(max_length=255)
    content = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.title
```

```
# settings.py
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "articles",
]
```

- 資料遷移

```
python3 manage.py makemigrations
python3 manage.py migrate
```

今天的專案就先告一個段落，明天會開始建立相關的核心功能

今日總結

我們今天了解了幾種啟動Django專案的方式：

- 開發階段最好用與方便的Django runserver
- 足以應付一般Web應用需求的WSGI
- 整體來說最適合現代Web應用的ASGI

最後開始建構我們這次有趣的專案DocuMind，明天準備建立專案最重要的幾個功能：

- 上傳PDF與解析

- 將文章內容轉換成向量資料
- 根據用戶問題，拿到向量資料並且與prompt輸入到LLM，返回對應的解答
- 長任務交給celery處理