FastAPI 入門指南

什麼是 FastAPI?

FastAPI 是一個現代、快速的 Python Web 框架，用於建構 API

主要特色：

- 🚀 **高性能**：基於 Starlette 和 Pydantic，性能媲美 NodeJS 和 Go
- 📝 **自動文檔**：自動生成互動式 API 文檔 (Swagger UI)
- 🔍 **類型檢查**：支援 Python 類型提示，自動驗證資料
- 🛠️ **易開發**：減少人為錯誤，開發體驗佳
- ⚡ **異步支援**：原生支援 async/await

FastAPI 心智模型

```
HTTP Request → FastAPI App → Route Handler → Response
     ↓              ↓              ↓            ↓
  GET/POST       @app.get()     你的函數      JSON/HTML
 /users          @app.post()    處理邏輯      自動驗證
 /users/1        @app.put()     資料庫操作    錯誤處理
```

核心概念：

- **App**：FastAPI 應用程式實例
- **Route**：API 端點路徑
- **Handler**：處理請求的函數
- **Pydantic**：資料驗證和序列化

安裝與基本設置

```
# 安裝 FastAPI 和支援套件
pip install fastapi uvicorn[standard]

# 建立基本應用程式
# main.py
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}
```

```
# 啟動開發伺服器
uvicorn main:app --reload
```

@app.get 語法詳解

`@app.get()` 是 FastAPI 的**路徑裝飾器**，用來定義 GET 請求的處理器

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/path")                    # 基本語法
def handler_function():              # 處理函數
    return {"message": "Hello"}      # 回傳資料
```

參數說明：

- `path`：API 端點路徑字串

@app.get 基本範例

```python
from fastapi import FastAPI

app = FastAPI()

# 基本 GET 路由
@app.get("/")
def read_root():
    return {"Hello": "World"}

# 帶參數的路由
@app.get("/items/{item_id}")
def read_item(item_id: int):
    return {"item_id": item_id}

# 查詢參數
@app.get("/items/")
def read_items(skip: int = 0, limit: int = 10):
    return {"skip": skip, "limit": limit}
```

路徑參數 (Path Parameters)

```python
from fastapi import FastAPI

app = FastAPI()

# 路徑參數 - 必要參數
@app.get("/users/{user_id}")
def get_user(user_id: int):
    return {"user_id": user_id}

# 路徑參數 - 字串類型
@app.get("/posts/{post_id}")
def get_post(post_id: str):
    return {"post_id": post_id}

# 多個路徑參數
@app.get("/users/{user_id}/posts/{post_id}")
def get_user_post(user_id: int, post_id: int):
    return {"user_id": user_id, "post_id": post_id}
```

查詢參數 (Query Parameters)

```python
from fastapi import FastAPI

app = FastAPI()

# 查詢參數 - 選用參數
@app.get("/items/")
def read_items(skip: int = 0, limit: int = 10):
    return {"skip": skip, "limit": limit}

# 查詢參數 - 必要參數
@app.get("/search/")
def search_items(q: str, limit: int = 10):
    return {"q": q, "limit": limit}

# 布林值參數
@app.get("/items/")
def read_items(available: bool = True):
    return {"available": available}
```

請求主體 (Request Body)

```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

# 定義資料模型
class Item(BaseModel):
    name: str
    price: float
    is_offer: bool = None

# 使用請求主體
@app.get("/items/{item_id}")  # 等等，這是 GET 方法...
def read_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}

# POST 方法才真正使用請求主體
@app.post("/items/")
def create_item(item: Item):
    return item
```

回應模型 (Response Models)

```python
from fastapi import FastAPI
from pydantic import BaseModel
from typing import List, Optional

app = FastAPI()

class Item(BaseModel):
    name: str
    price: float
    is_offer: Optional[bool] = None

class User(BaseModel):
    username: str
    email: str
    full_name: Optional[str] = None

# 指定回應模型
@app.get("/items/", response_model=List[Item])
def read_items():
    return [
        {"name": "Item 1", "price": 10.5},
        {"name": "Item 2", "price": 20.0, "is_offer": True}
    ]

@app.get("/users/me", response_model=User)
def read_user_me():
    return {"username": "current_user", "email": "user@example.com"}
```

常用裝飾器總覽

```python
from fastapi import FastAPI

app = FastAPI()

# HTTP 方法裝飾器
@app.get("/items/")        # 取得資料
@app.post("/items/")       # 建立資料
@app.put("/items/{id}")    # 更新資料（完整替換）
@app.patch("/items/{id}")  # 更新資料（部分更新）
@app.delete("/items/{id}") # 刪除資料

# 特殊裝飾器
@app.head("/items/")       # 取得標頭資訊
@app.options("/items/")    # 取得支援的方法
```

迷你練習 (可直接複製貼上)

```python
# 建立 app.py
from fastapi import FastAPI

app = FastAPI(title="My API", description="FastAPI 練習")

# 練習 1: 基本路由
@app.get("/")
def home():
    return {"message": "歡迎使用 FastAPI!"}

# 練習 2: 路徑參數
@app.get("/hello/{name}")
def greet(name: str):
    return {"message": f"Hello, {name}!"}

# 練習 3: 查詢參數
@app.get("/items/")
def get_items(skip: int = 0, limit: int = 10):
    return {"skip": skip, "limit": limit, "items": []}

# 練習 4: 組合參數
@app.get("/users/{user_id}/posts/{post_id}")
def get_user_post(user_id: int, post_id: int, detail: bool = False):
    return {
        "user_id": user_id,
        "post_id": post_id,
        "detail": detail,
        "data": "這裡會是實際的資料"
    }
```

啟動與測試

```
# 啟動伺服器
uvicorn app:app --reload --host 0.0.0.0 --port 8000

# 測試 API
curl http://localhost:8000/
curl "http://localhost:8000/items/?skip=5&limit=20"
curl http://localhost:8000/hello/Alice

# 檢視文檔
# 打開瀏覽器訪問: http://localhost:8000/docs
```

FastAPI Cheat Sheet

```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

# 基本路由
@app.get("/")                                   # GET 請求
@app.post("/")                                  # POST 請求
@app.put("/{id}")                               # PUT 請求
@app.delete("/{id}")                            # DELETE 請求

# 參數類型
def func(item_id: int, q: str = None): # 路徑參數, 查詢參數
def func(item: Item):                           # 請求主體

# 啟動
uvicorn main:app --reload                   # 開發模式
uvicorn main:app --host 0.0.0.0             # 生產模式
```

常見問題與最佳實務

常見問題：

- 忘記 `return` 語句 → 會得到 500 錯誤
- 參數類型錯誤 → FastAPI 會自動驗證並回報
- 路徑衝突 → 越具體的路徑要放在越前面

最佳實務：

- 使用類型提示
- 定義 Pydantic 模型
- 使用有意義的狀態碼

下一步學習建議

1. **進階路由**：更多 HTTP 方法和複雜路由
2. **資料庫整合**：SQLAlchemy, Tortoise ORM
3. **認證與安全性**：OAuth2, JWT
4. **測試**：Pytest + HTTPX
5. **部署**：Docker, Kubernetes
6. **進階功能**：中介軟體、事件處理、背景任務

學習資源：

- FastAPI 官方文檔
- FastAPI 教學