第9章　程式

9-1 元件清單及其規格描述

▼表9-1-2 元件清單及其規格描述表(後端)

|  |  |  |
| --- | --- | --- |
| Python檔案(後端)－ChatGPT API | | |
| 編號 | 檔案名稱 | 功能 |
| 2-5-1 | house\_describe\_with\_neo4j.py  house\_describe\_with\_neo4j.ipynb | 連接neo4j給chatgpt當作回答的依據。 |
| 2-5-2 | chatgpt\_question.py  chatgpt\_question. ipynb | 連接chatgpt，且提供neo4j的房屋資料當作回答的參考依據。 |
| 2-5-3 | Line\_all\_function.py | Line聊天機器人連接到chatgpt做智慧問答。 |
| 2-5-4 |  |  |
| 2-5-5 |  |  |

9-2 其他附屬之各種元件

▼表9-2-1 部分程式碼－house\_describe\_with\_neo4j.py

|  |  |  |  |
| --- | --- | --- | --- |
| 編號 | 2-5-1 | 程式名稱 | house\_describe\_with\_neo4j.py |
| 目的 | 連接neo4j給chatgpt當作回答的依據。 | | |
| 部分程式碼 | | | |
| 1.連接到 Neo4j 資料庫   * 使用 py2neo 模組連接到 Neo4j 資料庫，指定連接的 URI 和授權憑據。   2.定義函數 generate\_description   * 輸入參數：hid（房屋 ID） * 建立查詢語句：   + 匹配房屋節點 h，該節點的 hid 等於輸入參數 hid。   + 可選匹配房屋節點 h 與店家節點 s 之間的 NEAR\_STORE 關係。   + 返回房屋節點 h 和所有匹配的店家節點 s（以列表形式返回）。 * 執行查詢，將結果存儲到 result。   3.檢查查詢結果   * 如果 result 為空：   + 返回 "未找到與該HID相關的房屋資料。" * 否則：   + 獲取房屋資訊 house\_info 和店家資訊 stores。   4.初始化描述列表   * 創建一個空的描述列表 descriptions。   5.生成房屋屬性描述   * 如果 house\_info 包含 address：   + 添加描述 "房屋地址位於 {address}。" * 如果 house\_info 包含 type：   + 添加描述 "該房屋類型為 {type}。" * 如果 house\_info 包含 pattern：   + 添加描述 "房屋格局為 {pattern}。" * 如果 house\_info 包含 size：   + 添加描述 "房屋大小為 {size} 坪。" * 如果 house\_info 包含 layer：   + 添加描述 "樓層為 {layer}。" * 如果 house\_info 包含 price：   + 添加描述 "房屋租金為 {price} 元。" * 如果 house\_info 包含 subway：   + 添加描述 "距離最近的地鐵站為 {subway}。" * 如果 house\_info 包含 bus：   + 添加描述 "附近的公車站包括 {bus}。"   6.生成店家描述   * 如果 stores 不為空：   + 從 stores 中提取每個店家的 name。   + 如果 store\_names 不為空：     - 添加描述 "附近有以下店家：{店家清單}。"   7.返回完整描述   * 將所有描述合併成一個字符串並返回。   8.主程式   * 獲取用戶輸入的 hid。 * 調用 generate\_description 函數，將 hid 傳入。 * 打印生成的描述。 | | | |

▼表9-2-2 部分程式碼－xxx.py

|  |  |  |  |
| --- | --- | --- | --- |
| 編號 | 2-5-2 | 程式名稱 | chatgpt\_question.py |
| 目的 | 連接chatgpt，且提供neo4j的房屋資料當作回答的參考依據。 | | |
| 部分程式碼 | | | |
| 1.**初始化與設置**   * 設定 OpenAI 的 API 密鑰。 * 使用 py2neo 模組連接到 Neo4j 資料庫，指定連接的 URI 和授權憑據。   2.**定義函數 generate\_description**   * **輸入參數**：hid（房屋 ID） * **建立查詢語句**：   + 匹配房屋節點 h，該節點的 hid 等於輸入參數 hid。   + 可選匹配房屋節點 h 與店家節點 s 之間的 NEAR\_STORE 關係。   + 返回房屋節點 h 和所有匹配的店家節點 s（以列表形式返回）。 * **執行查詢**，將結果存儲到 result。   3.**檢查查詢結果**   * 如果 result 為空：   + 返回 "未找到與該HID相關的房屋資料。" * 否則：   + 獲取房屋資訊 house\_info 和店家資訊 stores。   4.**初始化描述列表**   * 創建一個空的描述列表 descriptions。   5.**生成房屋屬性描述**   * 如果 house\_info 包含 address：   + 添加描述 "房屋地址位於 {address}。" * 如果 house\_info 包含 type：   + 添加描述 "該房屋類型為 {type}。" * 如果 house\_info 包含 pattern：   + 添加描述 "房屋格局為 {pattern}。" * 如果 house\_info 包含 size：   + 添加描述 "房屋大小為 {size} 坪。" * 如果 house\_info 包含 layer：   + 添加描述 "樓層為 {layer}。" * 如果 house\_info 包含 price：   + 添加描述 "房屋租金為 {price} 元。" * 如果 house\_info 包含 subway：   + 添加描述 "距離最近的捷運站為 {subway}。" * 如果 house\_info 包含 bus：   + 添加描述 "附近的公車站包括 {bus}。"   6.**生成店家描述**   * 如果 stores 不為空：   + 從 stores 中提取每個店家的 name。   + 如果 store\_names 不為空：     - 添加描述 "附近有以下店家：{店家清單}。"   7.**返回完整描述**   * 將所有描述合併成一個字符串並返回。   8.**定義函數 gpt\_analyze\_input**   * **輸入參數**：hid（房屋 ID），message（使用者提問） * 調用 generate\_description 函數以生成房屋描述。 * 如果描述中包含 "未找到與該HID相關的房屋資料"：   + 返回該訊息。 * 否則，構建 OpenAI GPT 的 prompt，包含房屋描述和使用者提問。 * 使用 OpenAI GPT 生成回答：   + 發送 prompt 給 OpenAI GPT-3.5-turbo-16k 模型。   + 使用系統訊息設定模型角色為專業房屋中介助手，並指定回答語言為繁體中文。 * 返回模型生成的回覆。   9.**例外處理**   * 如果調用 OpenAI API 或其他操作發生異常，捕捉異常並打印錯誤訊息。 * 返回 "在處理您的請求時發生錯誤。"   10.**主程式**   * 獲取使用者輸入的 hid 和 question。 * 打印輸入訊息。 * 調用 gpt\_analyze\_input 函數，傳入 hid 和 question。 * 打印生成的回答。 * 如果未收到回答，則打印錯誤訊息。 | | | |

▼表9-2-3 部分程式碼－xxx.py

|  |  |  |  |
| --- | --- | --- | --- |
| 編號 | 2-5-3 | 程式名稱 | Line\_all\_function.py |
| 目的 | Line聊天機器人連接到chatgpt做智慧問答。 | | |
| 部分程式碼 | | | |
| 1. **連接到 Neo4j 資料庫**    * 使用 py2neo 模組連接到 Neo4j 資料庫，指定連接的 URI (bolt://localhost:7687) 和授權憑據（使用者名稱和密碼）。 2. **設置 Line Bot API 和 Webhook Handler**    * 初始化 LineBotApi 和 WebhookHandler，使用各自的金鑰和通行碼進行身份驗證。 3. **啟用 ngrok 並創建公共 URL**    * 使用 ngrok 來建立本地伺服器的公共 URL。    * 設置 ngrok 認證令牌。    * 斷開任何現有的隧道連接。    * 建立新隧道並打印公共 URL。 4. **初始化 Flask 應用**    * 創建 Flask 應用實例，並設置隨機生成的秘密金鑰。 5. **定義全局字典用於存儲使用者狀態**    * 創建一個全局字典 user\_states 用於儲存使用者狀態。 6. **定義函數 generate\_description**    * **輸入參數**：hid（房屋 ID）    * **查詢語句**：      + 匹配房屋節點 h，該節點的 hid 等於輸入參數 hid。      + 可選匹配房屋節點 h 與店家節點 s 之間的 NEAR\_STORE 關係。      + 返回房屋節點 h 和所有匹配的店家節點 s（以列表形式返回）。    * **執行查詢**：將結果存儲到 result。    * **生成描述**：      + 檢查 result 是否為空，如果是，返回"未找到與該HID相關的房屋資料"。      + 從 result 提取房屋信息和店家列表。      + 根據房屋信息生成描述，並將描述存入列表 descriptions。      + 返回連接所有描述的字符串。 7. **定義函數 gpt\_analyze\_input**    * **輸入參數**：message（使用者消息），user\_id（使用者 ID）    * **專業系統消息**：設置系統消息，定義 GPT 回應的範圍和語言風格。    * **判斷使用者消息類型**：      + 如果消息包含 "租屋知識" 或 "租屋相關"，則生成對應的 prompt 用於租屋知識查詢。      + 否則，提取 hid 並生成房屋描述，生成對應的 prompt 用於房屋信息查詢。    * **調用 OpenAI API**：      + 使用 openai.ChatCompletion.create 調用 OpenAI API，傳入系統消息和 prompt 生成回應。      + 將生成的回應提取並返回給使用者。 8. **設置 Linebot 回調函數 callback**    * **獲取簽名和請求體**：      + 從請求頭獲取 X-Line-Signature。      + 以文本形式獲取請求體。    * **處理請求**：使用 WebhookHandler 處理請求，並返回 'OK'。    * **處理異常**：如果簽名無效，則返回 400 錯誤。 9. **定義函數 handle\_message**    * **獲取使用者 ID 和消息**：從事件中提取使用者 ID 和消息。    * **檢查消息格式**：      + 使用正則表達式檢查消息是否符合格式：member\_id: <數字> hid: <數字>。      + 如果格式正確，更新使用者狀態並回應驗證成功。      + 否則，檢查使用者狀態是否已驗證，如果已驗證則分析輸入，否則提示重新驗證。 10. **啟動 Flask 應用**  * 使用 app.run() 啟動 Flask 應用。 | | | |