|  |
| --- |
|  |
| Spring Boot - API 開發 |
|  |

|  |
| --- |
| 黃懷慶  2024/12/29 |

目錄

[第一章 API 開發的層級架構 2](#_Toc186350375)

[第二章 Entity 3](#_Toc186350376)

[第三章 Repository 6](#_Toc186350377)

[第四章 DTO 7](#_Toc186350378)

[第五章 Mapper 8](#_Toc186350379)

[第六章 Service 9](#_Toc186350380)

[第七章 Controller 12](#_Toc186350381)

[第八章 Util 14](#_Toc186350382)

[第九章 Constants 15](#_Toc186350383)

[第十章 無主鍵talbe 執行 INSERT/UPDATE/DELETE (動態SQL) 16](#_Toc186350384)

# API 開發的層級架構



|  |  |
| --- | --- |
| 架構 | 說明 |
| Controller | 功能：負責接受 API 請求，並調用 Service 進行 業務邏輯處理，返回結果。 角色：API請求的出入口。 |
| Service | 功能：負責 業務邏輯處理，API 需要的 回傳資料 都在這邊完成。 角色：Spring的處理核心。 |
| Dto | 功能：定義 數據的傳輸對象。 角色：集中管理 數據傳輸對象 的地方。 |
| Util | 功能：定義 靜態工具類方法，如：日期處理、外部API請求。 角色：集中管理 工具類方法 的地方。 |
| Constants | 功能：定義 靜態常量，如：保單狀態及其對應的中文名稱。 角色：集中管理 各種靜態常量 的地方。 |
| Entity | 功能：Spring Data JPA 中，設定talbe的地方。 |
| Repository | 功能：Spring Data JPA 中，設定SQL語法的地方。  (包含 自動設定 和 原生SQL設定) |
| Mapper | 功能：定義 Entity 轉換為 Dto 的方法。 |

# Entity

Entity 是用來表示 數據庫表 的Java類，在這邊的設定 必須與 DB 中的 table 結構對應。  
如果要使用JpaRepository進行 SQL方法 的自動生成，就必須要 設定主鍵。

1. Class 註解

|  |  |
| --- | --- |
| 註解 | 說明 |
| @Entity | 標記為 entity 類 |
| @Table(name = "clnt") | DB對應的table 名稱 |
| @Schema(description = "客戶資料檔") | Swagger要顯示的中文說明 |

1. 欄位 註解

|  |  |
| --- | --- |
| 註解 | 說明 |
| @Schema(description = "客戶證號") | Swagger要顯示的中文說明 |
| @Column(name = "client\_id",  length = 10, nullable = false) | DB對應的 table 欄位 相關設定  name : table欄位名稱  length : 長度 (字串才需要)  nullable = false : 不允許null |
| @Id | 設定為 主鍵 |
| @IdClass(Addr.AddrKey.class) | 設定 主鍵的 崁套類 (多重主鍵使用) |

備註：  
單一主鍵：需要使用註解 @Id 標示主鍵。  
多重主鍵：需要使用註解 @Id 標示主鍵 + @IdClass 標示 多主鍵的崁套類。  
　　　　　多重主鍵的崁套類必須要有「無參數的建構式」、「equals()」和「hashCode()」。

1. 單一主鍵 範例



1. 多重主鍵 範例



# Repository

Repository 是用來 定義 與DB的交互操作，必須是 interface，  
繼承JpaRepository就可以有 Spring Data JPA 自動生成的 增刪改查方法；  
簡單的原生SQL也可以在這邊定義。

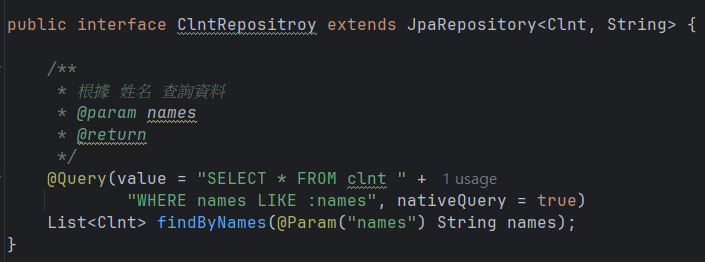
1. JpaRepository自動生成的方法 (介紹常用的)

|  |  |
| --- | --- |
| 方法 | 說明 |
| save | 新增/修改資料 (單筆)  無資料=新增 / 有資料=修改 (根據主鍵判斷) |
| saveAll | 新增/修改資料 (多筆)  無資料=新增 / 有資料=修改 (根據主鍵判斷) |
| findById | 根據主鍵 查詢 一筆資料 (單一主鍵) |
| findAllById | 根據主鍵 查詢 多筆資料 (主鍵清單) |
| deleteById | 根據主鍵 刪除 一筆資料 (單一主鍵) |
| deleteAllById | 乾鉅主鍵 刪除 多筆資料 (主鍵清單) |

＊於Service中有使用範例。

1. 簡單的原生SQL
2. @Query(value = "SQL語句", nativeQuery = true)
3. SQL語句中，變數要用「:」標示，如：「:names」。
4. 傳入參數 的地方，也要使用「@Param("SQL變數")」標示，如：「@Param("names")」。

＊複雜的原生SQL會寫在Service中，於Service進行介紹。



# DTO

DTO 是用來 定義 封裝數據的對象。

1. 註解說明

|  |  |
| --- | --- |
| 註解 | 說明 |
| @Schema(description = "客戶證號") | Swagger要顯示的中文說明 |
| @Schema(hidden = true) | Swagger 隐藏此方法  (自定義方法時使用，避免 swagger 判為為 欄位) |
| @JsonIgnore | JSON 隱藏此方法  (自定義方法時使用，避免 swagger 判為為 欄位) |

備註：  
自定義的 get方法，為了避免swagger誤認為 欄位，  
需要加上 @Schema(hidden = true) 和 @JsonIgnore這兩個註解。

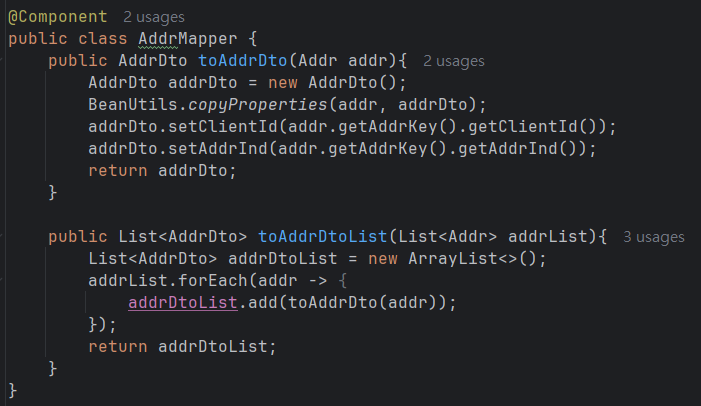


# Mapper

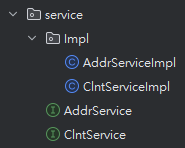
Mapper 是用來 定義 Entity 和 DTO 之間互相轉換方法的地方。

1. 註解說明

|  |  |
| --- | --- |
| 註解 | 說明 |
| @Component | 標記為 Spring 的 Bean 使其可以透過 @ Autowired 進行 注入依賴 |



# Service

Service 是用來 進行 業務邏輯撰寫 的地方，也是 整個Spring 邏輯處理的核心。

建議使用 Service Interface 和 Service Implementation 的  
設計模式進行撰寫，優點是 解偶 與 提高可維護性。

1. 註解說明

|  |  |
| --- | --- |
| 註解 | 說明 |
| @Service | 標記為 Service |
| @Autowired | Spring 自動注入方法 (可以不用 new 就進行使用) |
| @Override | 標註為 重寫(override) 的方法 |
| @Transactional | 標註為 交易，用於 DB 的 增刪改 操作，失敗會進行回朔處理。 |

1. 複雜的原生SQL

|  |  |
| --- | --- |
| 方法 | 說明 |
| createNativeQuery | 參數1: SQL語句 ; 參數2: 資料儲存的物件(欄位名稱要相同) |
| unwrap(NativeQuery.class) | 為了要使用 Hibernate 的方法  必須 import hibernate 設定的方法(不要用 spring 的) |
| setParameter | 變數 的 數值設定 |
| getResultList() | 查詢結果 用 List 型態回傳 |





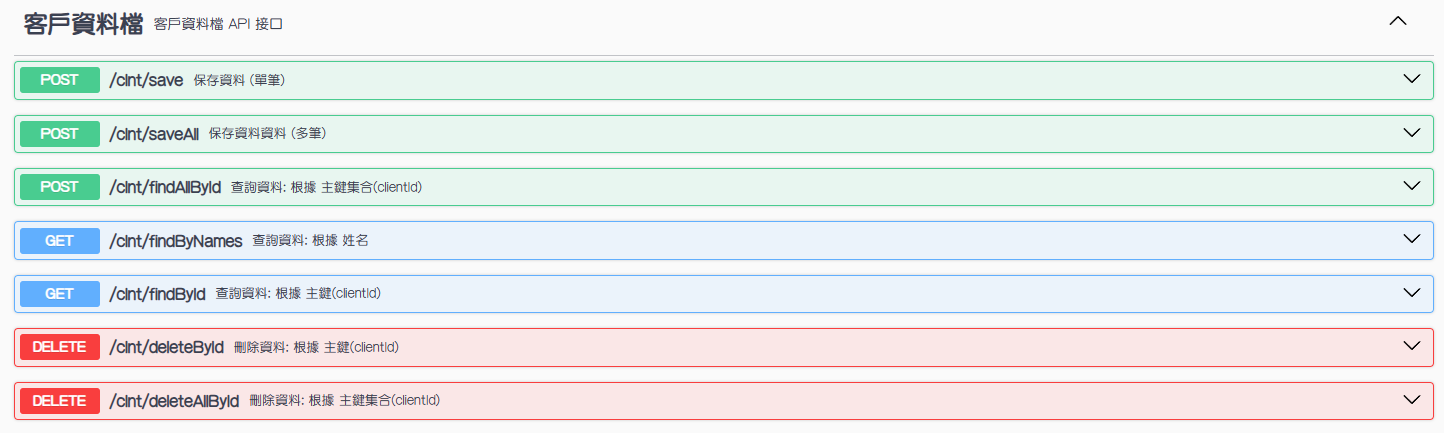


# Controller

Controller 是負責處理 用戶請求 的組件，也就是 Spring 的 入口。

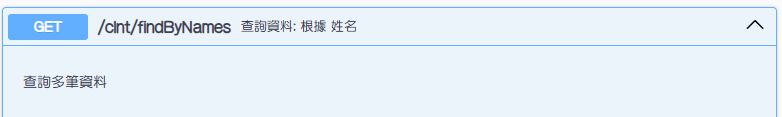
1. Class 註解

|  |  |
| --- | --- |
| 註解 | 說明 |
| @RestController | 標記為 Controller 並 返回 JSON 或其他數據格式 |
| @RequestMapping("/clnt") | 定義 Controller 接口的 URL路徑 |
| @Tag(name = "客戶資料檔", description = "客戶資料檔 API 接口") | Swagger 的 說明文件 |



1. 方法 註解

|  |  |
| --- | --- |
| 註解 | 說明 |
| @Autowired | Spring 自動注入方法 (可以不用 new 就進行使用) |
| @Operation  (summary = "查詢資料: 根據 姓名 ", description = "刪除多筆資料") | Swagger 的 說明文件 |
| @GetMapping("/findById") | 標記為 Get 請求，並設定 請求的URL |
| @PostMapping("/findAllById") | 標記為 Post 請求，並設定 請求的URL |
| @DeleteMapping("/deleteAllById") | 標記為 Delete 請求，並設定 請求的URL |
| @RequestBody | 用於 Post 請求，參數放於 body 中 |
| @RequestParam | 多用於 Get請求，參數放於 網址列 中 |





# Util

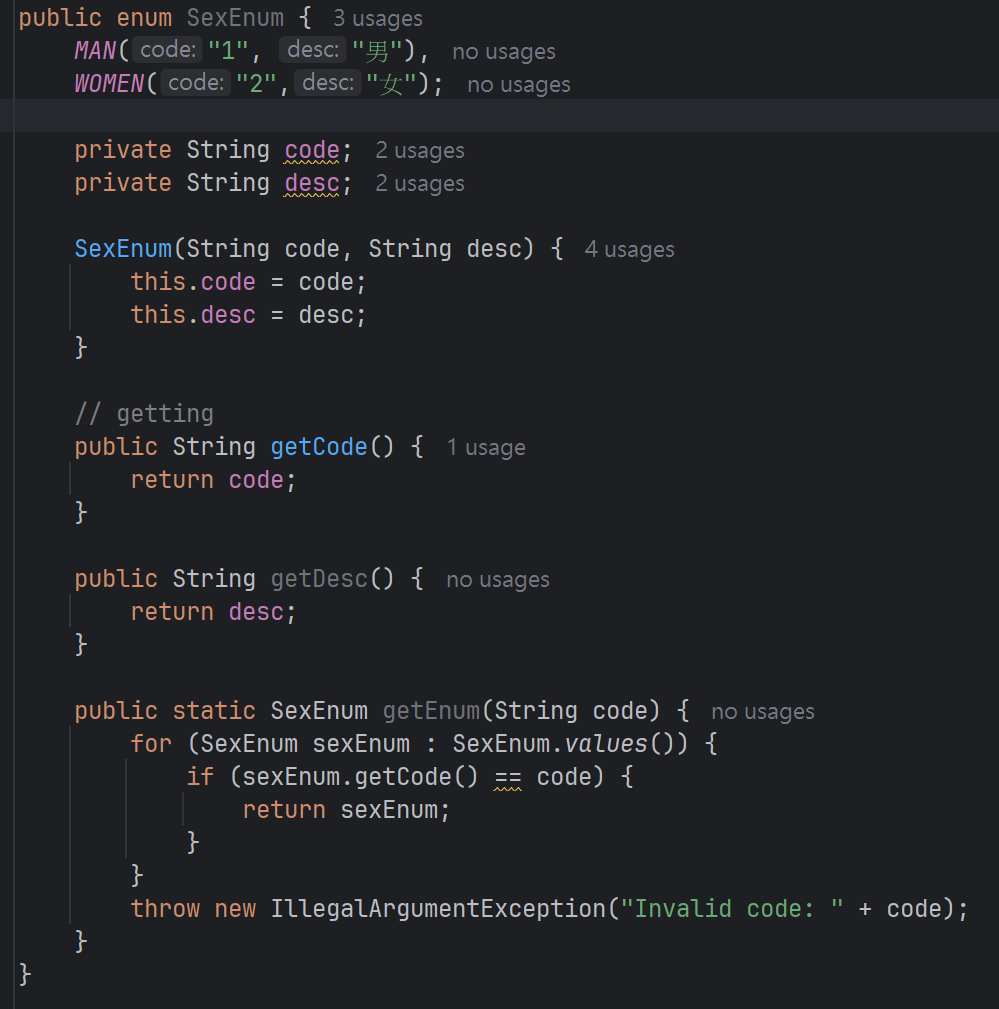
Util 是用來 定義 工具類 的 靜態組件 的 地方。

基本上 會放置 各種與邏輯無關的公用方法，如：日期工具。



# Constants

Constants 是用來 定義 Enum 和 常數 的地方。



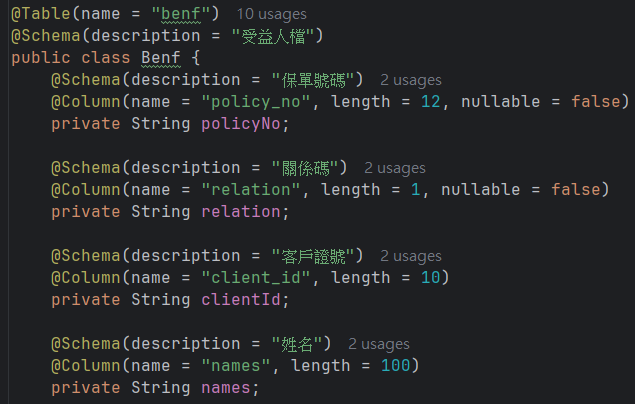
# 透過 動態SQL 製作 原生SQL 共用模組

透過 Spring Data JPA，我們可以自動生成各種 增刪改查 操作。然而，這要求table必須具備主鍵。  
對於沒有主鍵的table，我們需要借助 Hibernate 的原生 SQL 來實現 增刪改查 操作。

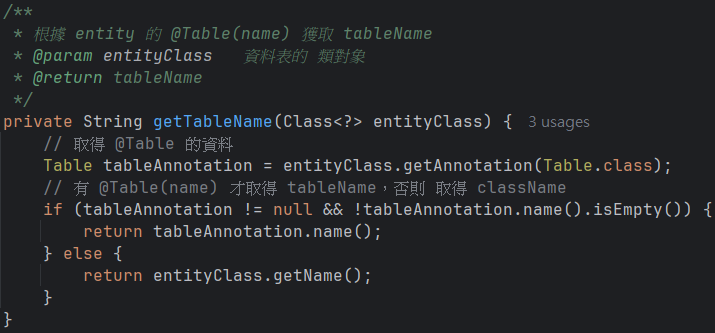
如果將條件設定在 單一table的 INSERT, UPDATE, DELETE 和 簡單的SELECT，  
由於 這些操作 需要寫的程式都是相同的，因此我們可以將其 設計成 共用模組，來方便我們使用。

1. Entity

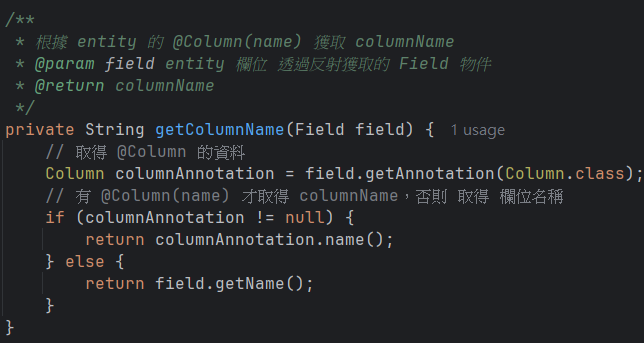
* 無主鍵 的 table 不可以設定 @Entity
* 必須要有 @Table(name) 和 @Column(name)



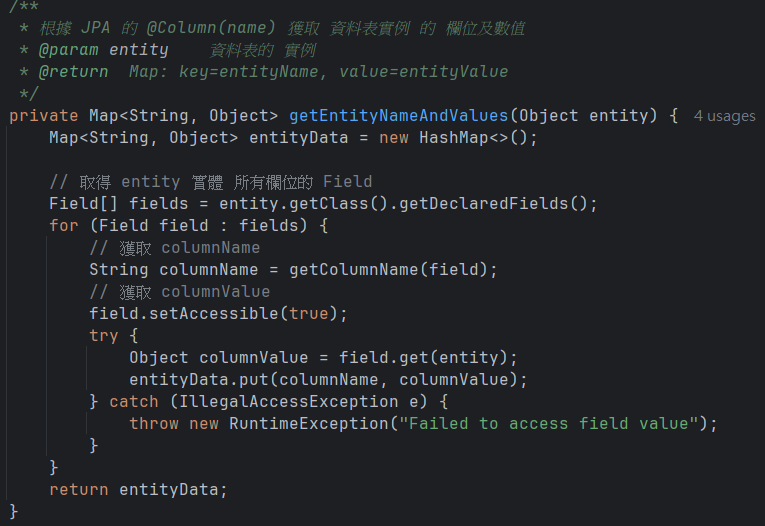
1. Service (靜態方法)
2. 獲取 Table Name



1. 獲取 Column Name



1. 取得 entity 實體 的 columnName 和 value



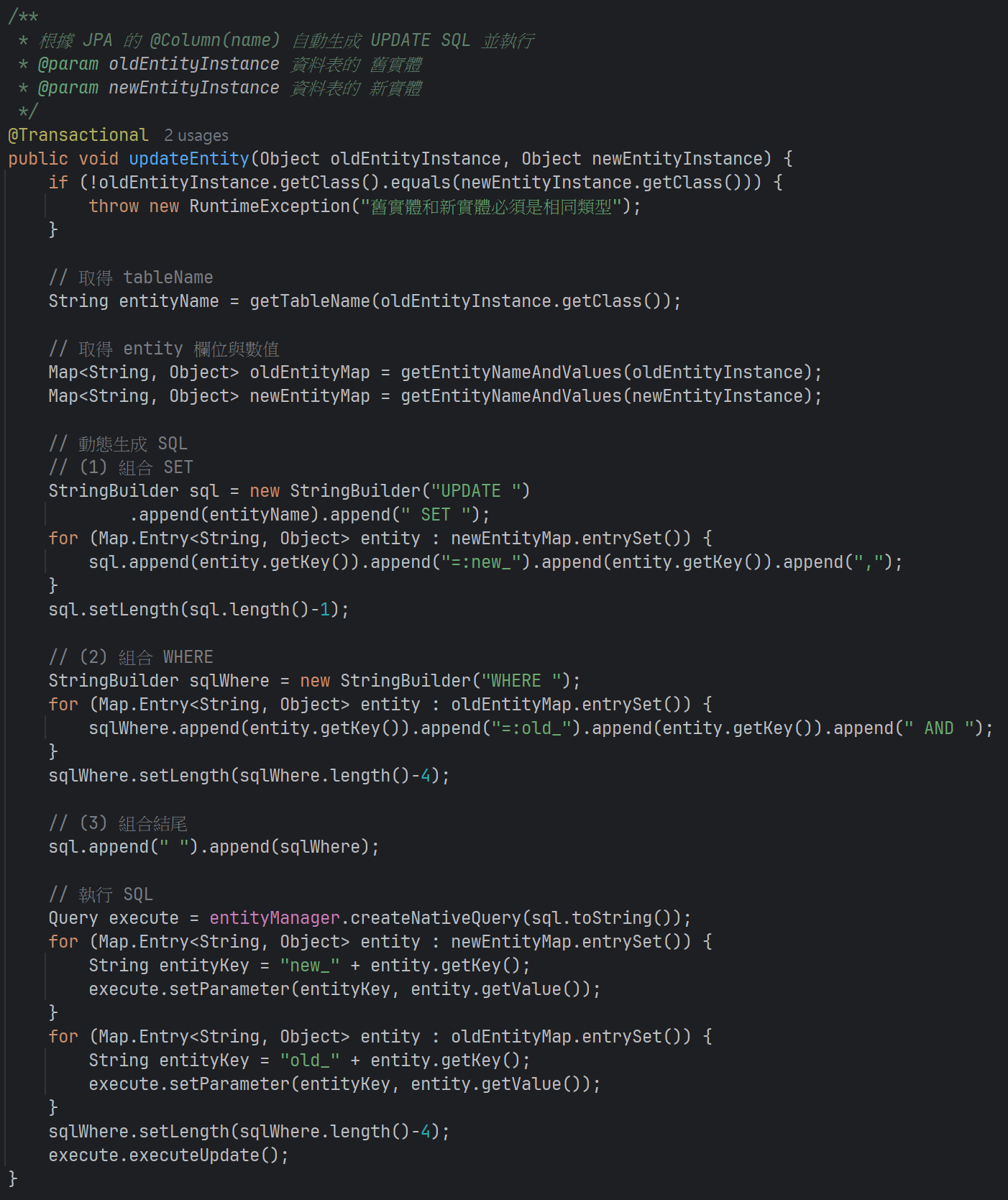
1. Service (SQL操作)
2. INSERT

透過 動態SQL 組合出INSERT的 SQL語句，並透過 setParameter 設定參數值。



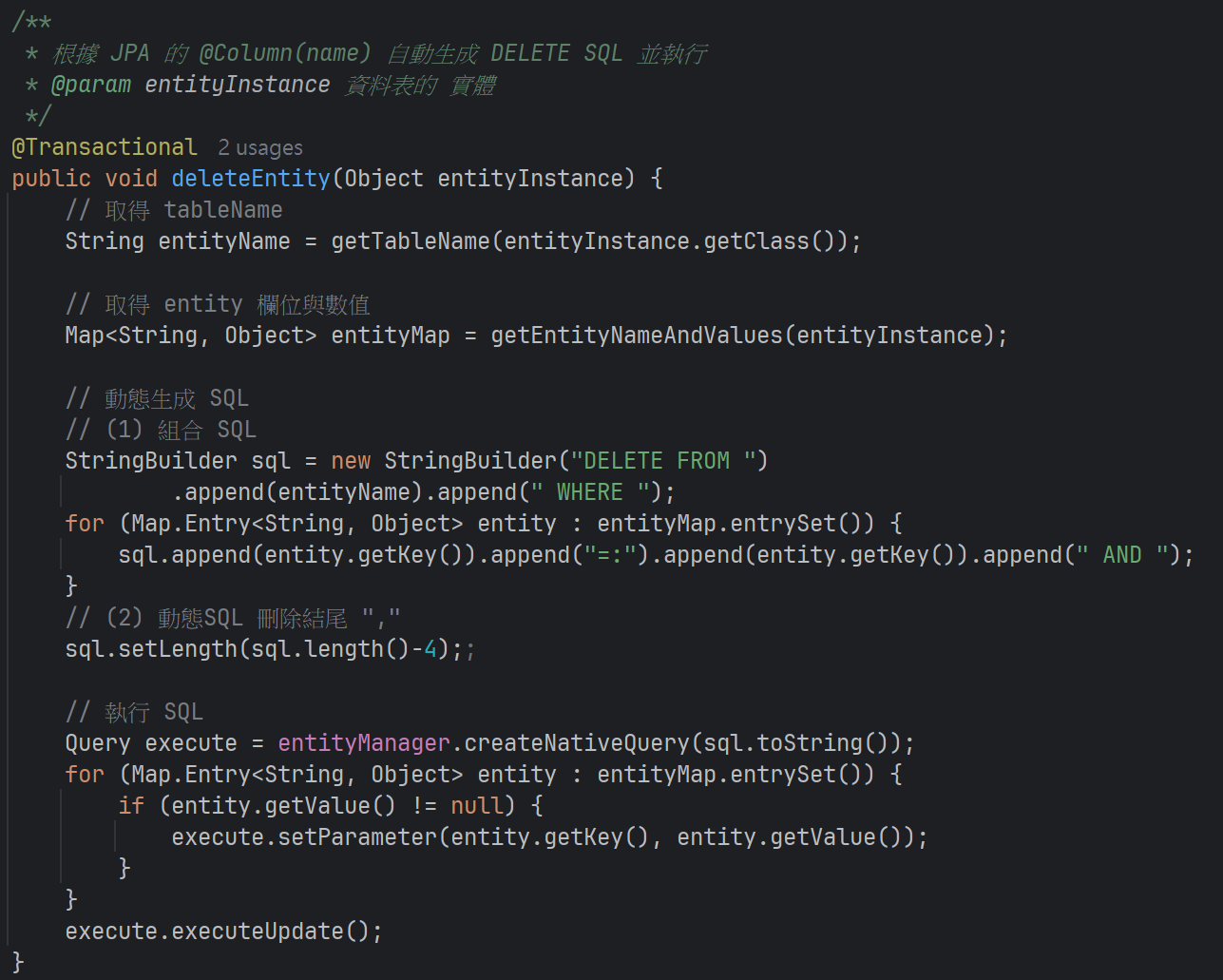
1. UPDATE

透過 動態SQL 組合出UPDATE的 SQL語句，並透過 setParameter 設定參數值。



1. DELETE

透過 動態SQL 組合出DELETE的 SQL語句，並透過 setParameter 設定參數值。

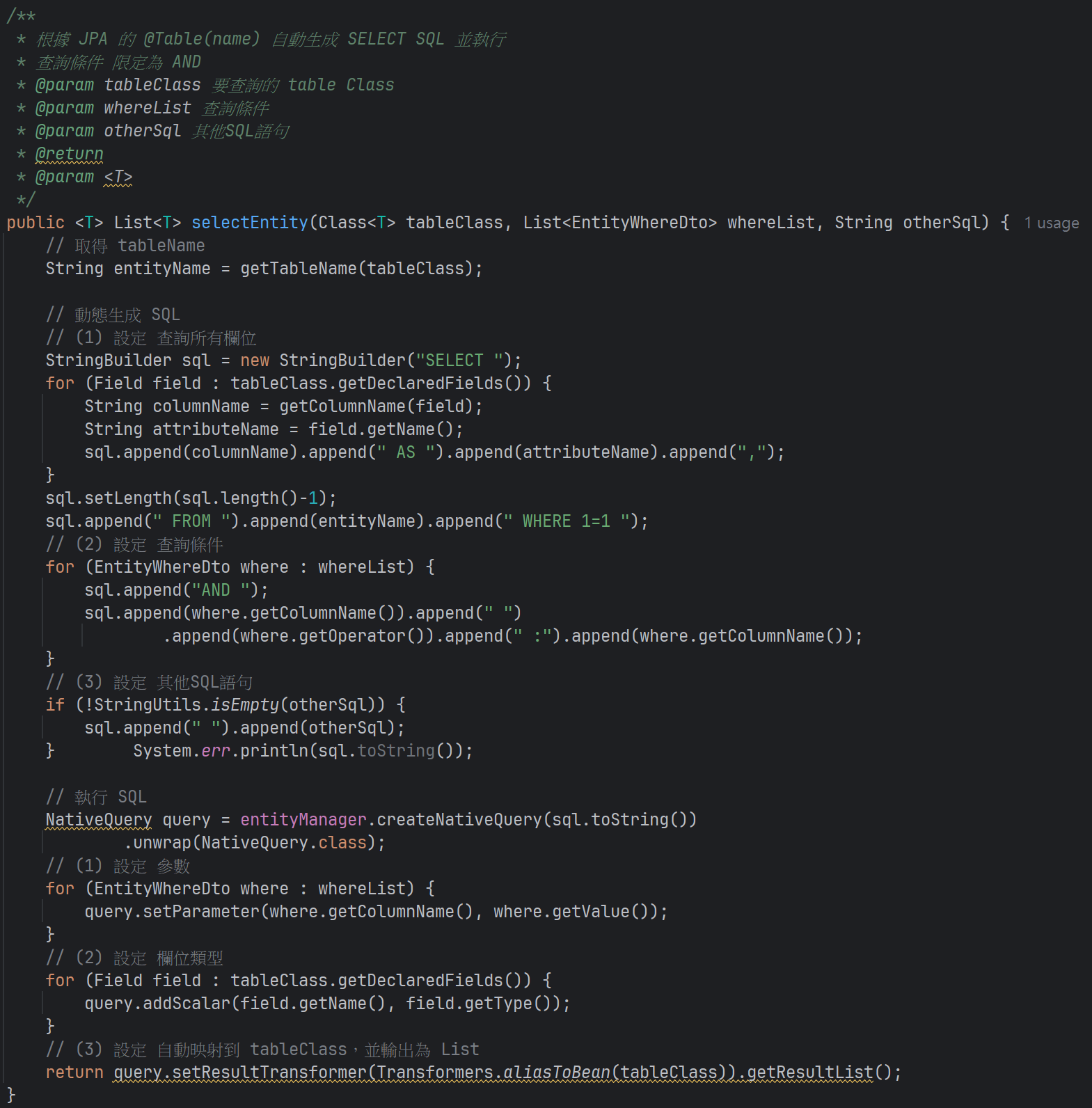


1. SELECT

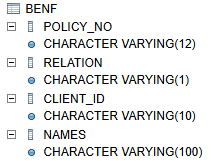
* 輸入查詢條件的DTO



* 藉由 泛型 設定 原生SQL 的查詢方法
* 透過 解析 DTO Class 來設定 SELECT 欄位 與 別名
* 透過「輸入查詢條件的DTO」設定 簡單的條件 (單參數條件)  
  ＊@Column(name) 取得 欄位名稱
* 透過 otherSql 來容許 使用者 自行輸入 其他 SQL語句 (會放在最後面)
* 原生SQL：  
  tableClass.getDeclaredFields() 可取得 DTO的 欄位List (格式= Field)  
  field.getName() 可取得 欄位名稱  
  field.getType() 可取得 欄位值

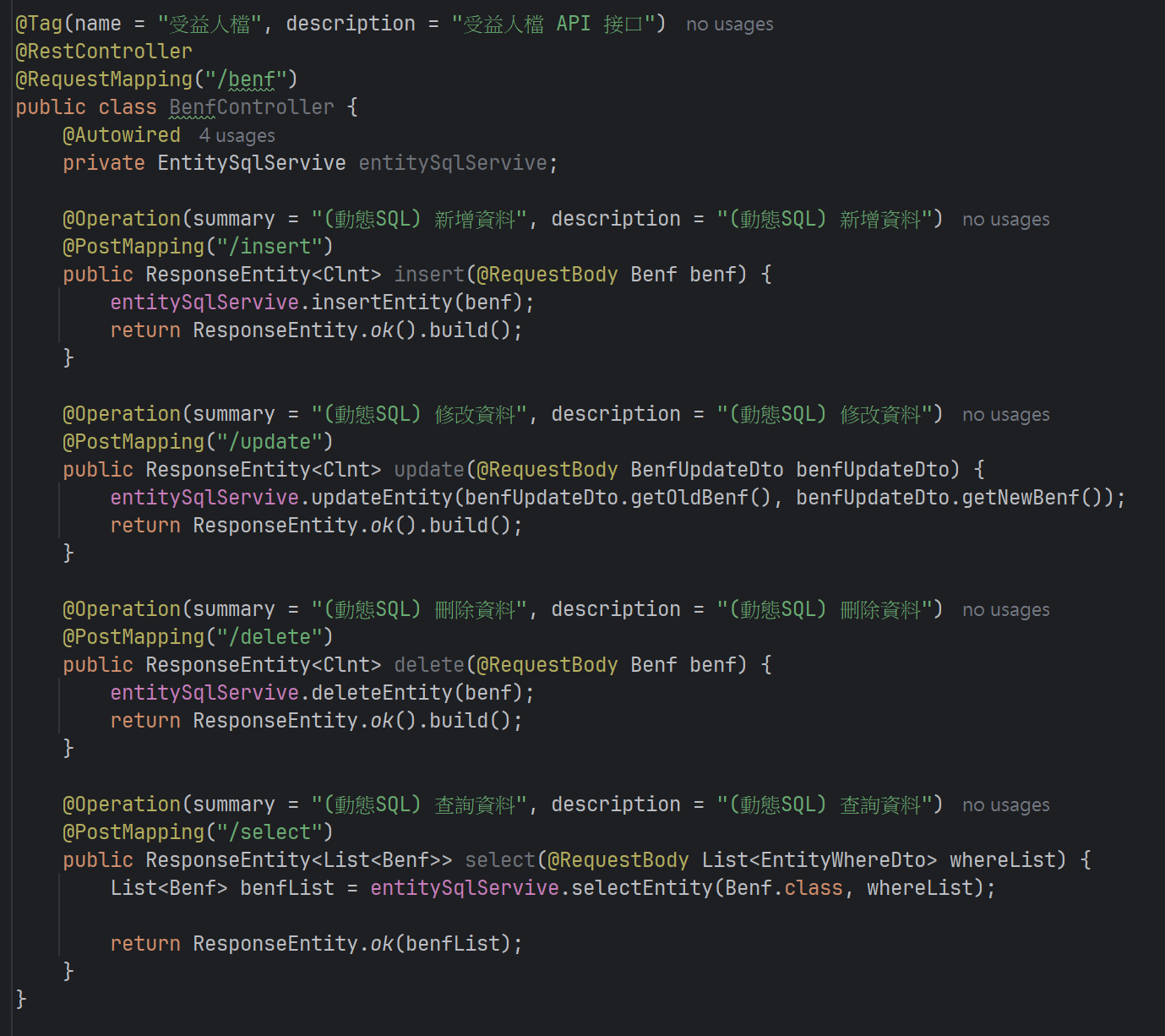


1. 使用範例

* 先開啟 H2 Database 手動建立 benf

CREATE TABLE benf (  
 policy\_no VARCHAR(12),  
 relation VARCHAR(1),  
 client\_id VARCHAR(10),  
 names VARCHAR(100)  
)

* Controller



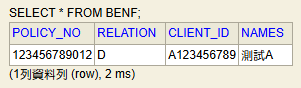
* INSERT

SQL：  
INSERT INTO benf (policy\_no,names,client\_id,relation)   
VALUES (:policy\_no,:names,:client\_id,:relation)

設定參數後

INSERT INTO benf (policy\_no,names,client\_id,relation)   
VALUES ('123456789012','測試A','A123456789','D')



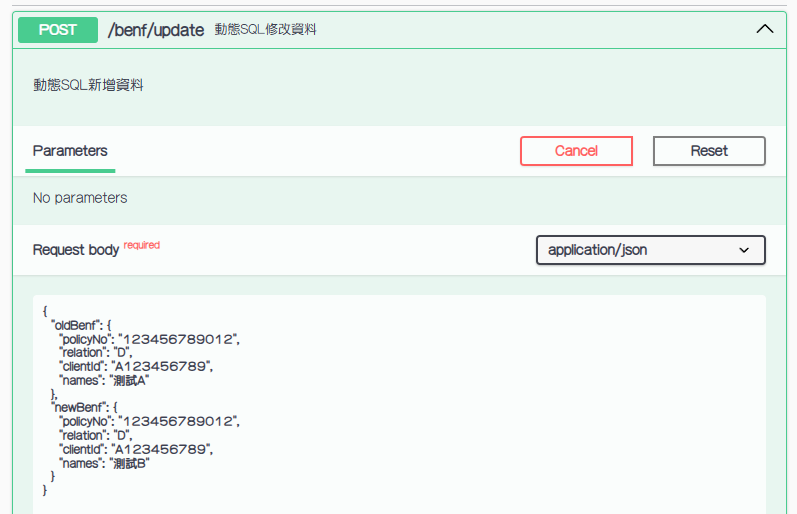


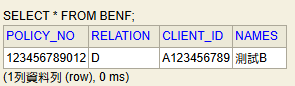
* UPDATE

SQL：  
UPDATE benf   
SET policy\_no=:new\_policy\_no,names=:new\_names,client\_id=:new\_client\_id  
 ,relation=:new\_relation   
WHERE policy\_no=:old\_policy\_no   
 AND names=:old\_names   
 AND client\_id=:old\_client\_id   
 AND relation=:old\_relation

設定參數後

UPDATE benf   
SET policy\_no='123456789012',names='測試B',client\_id='A123456789'  
 ,relation='D'   
WHERE policy\_no='123456789012'   
 AND names='測試A'   
 AND client\_id='A123456789'   
 AND relation='D'





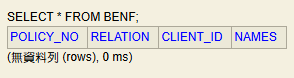
* DELETE

SQL：  
DELETE FROM benf   
WHERE policy\_no=:policy\_no   
 AND names=:names   
 AND client\_id=:client\_id   
 AND relation=:relation

設定參數後

DELETE FROM benf   
WHERE policy\_no='123456789012'   
 AND names='測試B'   
 AND client\_id='A123456789'   
 AND relation='D'

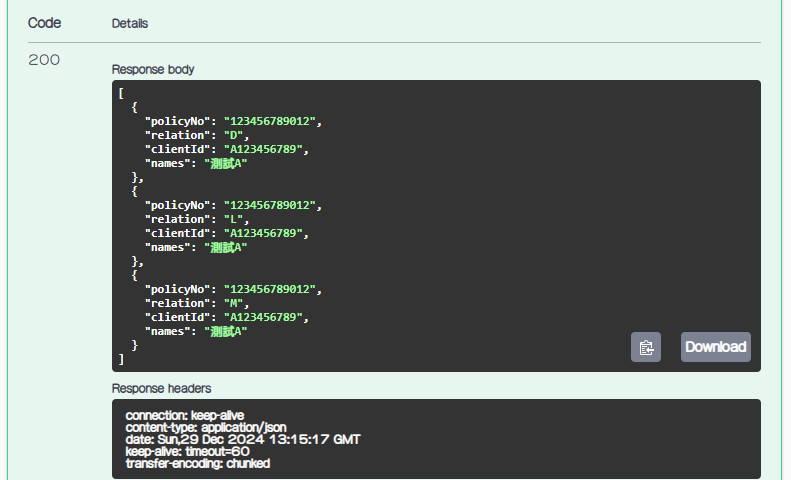




* SELECT

SQL：  
SELECT policy\_no AS policyNo,  
 relation AS relation,  
 client\_id AS clientId,  
 names AS names   
FROM benf   
WHERE 1=1   
AND policy\_no = :policy\_no





1. 程式碼



