7. JPA-Entity & Repository

除了打別人的 API 以外，有時資料會從資料庫來，這時會用 JPA 來對資料庫進行操作，JPA 全名是 Java Persistence API，是一個標準規範及接口 (API) 來實現 ORM (object-relational mapping) 框架，它的作用是在關聯資料庫和業務實體對象之間作一個映射，這樣在具體的操作業務對象的時候，就不需要再寫複雜的 SQL 語句，只需簡單的操作對象的屬性和方法。

以上是網路上對於 ORM 比較淺顯易懂的定義，現在可能沒什麼感覺，但實作時就能夠理解，**不用寫 SQL 但還是能夠對資料庫做操作**，另外有人會好奇什麼是 Persistence 持久層，目前先把它想成對資料庫做操作的層級即可，有興趣的人可以再去查詢表現層和業務層是什麼。

使用 JPA 時，先在 `pom.xml` 中加入以下設定：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>

<dependency>
    <groupId>com.oracle.jdbc</groupId>
    <artifactId>ojdbc8</artifactId>
    <version>12.2.0.1</version>
</dependency>
```

yml與properties的最大差異在於前者有固定格式；後者則沒有

然後設定資料庫的連線設定，因為只對一個 DB 做連接，直接設定在 `application.yml` 即可，當然也可以設定在 `application.properties` ，這邊範例使用 `application.yml` ：

```
spring:
  datasource:
    driverClassName: oracle.jdbc.OracleDriver
    url: jdbc:oracle:thin:@88.8.125.32:1521/xe
    username: STUDENT
    password: CATHAYBK654321
```

基本設定做好後，再來是程式的部分：

```
|--com.example.demospringboot
   |--DemospringbootApplication.java
|--com.example.demospringboot.configuration
   |--SwaggerConfig.java
   |--RestConfiguration.java
|--com.example.demospringboot.controller
   |--TestController.java
   |--ProductController.java
|--com.example.demospringboot.entity
   |--Car.java // 新增的檔案
   |--CarPK.java // 新增的檔案
|--com.example.demospringboot.model
   |--Product.java
|--com.example.demospringboot.service
   |--ProductService.java
|--com.example.demospringboot.service.impl
   |--ProductServiceImpl.java
```

```java
import java.io.Serializable;
import java.math.BigDecimal;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.IdClass;
import javax.persistence.Table;

@Entity
@IdClass(value = CarPK.class)
@Table(name = "CARS")
public class Car implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "MANUFACTURER")
    private String manufacturer;

    @Id
    @Column(name = "TYPE")
    private String type;

    @Column(name = "MIN_PRICE")
    private BigDecimal minPrice;

    @Column(name = "PRICE")
    private BigDecimal price;

    public BigDecimal getMinPrice() {
        return this.minPrice;
    }

    public void setMinPrice(BigDecimal minPrice) {
        this.minPrice = minPrice;
    }

    public BigDecimal getPrice() {
        return this.price;
    }

    public void setPrice(BigDecimal price) {
        this.price = price;
    }

    public String getManufacturer() {
        return manufacturer;
    }

    public void setManufacturer(String manufacturer) {
```

當資料做轉移(從一個環境到另一個環境時)通常都會做序列化

```
        this.manufacturer = manufacturer;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

}
```

```java
import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Embeddable;

@Embeddable 在執行時會被當作是entity的一個內嵌物件
public class CarPK implements Serializable {
    private static final long serialVersionUID = 1L;

    @Column(name = "MANUFACTURER")
    private String manufacturer;

    @Column(name = "TYPE")
    private String type;

    public CarPK() {
    }

    public String getManufacturer() {
        return this.manufacturer;
    }

    public void setManufacturer(String manufacturer) {
        this.manufacturer = manufacturer;
    }

    public String getType() {
        return this.type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public boolean equals(Object other) {
        if (this == other) {
            return true;
        }
        if (!(other instanceof CarPK)) {
            return false;
        }
        CarPK castOther = (CarPK) other;
        return this.manufacturer.equals(castOther.manufacturer)
                && this.type.equals(castOther.type);
    }

    public int hashCode() {
        final int prime = 31;
        int hash = 17;
        hash = hash * prime + this.manufacturer.hashCode();
```

```
        hash = hash * prime + this.type.hashCode();

        return hash;
    }
}
```

這邊範例特別使用複合主鍵的 table，所以有兩個 java 檔，先看 `Car.java` ，這 class 上有兩個必要的 annotation `@Entity` 跟 `@Table` ， `@Entity` 告訴 Spring Boot 要使用 ORM 對資料庫做操作， `@Table` 則告訴 Spring Boot 究竟要對應到哪一張 table， `@IdClass` 在做單主鍵的 ORM 時並不需要存在，然而如果是雙主鍵的話，就有存在的必要，這部分會對應到另一個 `CarPK.java` 。

`Car.java` class 的內部會寫下各種屬性以及 getter & setter，跟一般的 class 不同的是，class 中屬性要加上 `@Column` 跟 `@Id` ， `@Column` 對應到資料庫的欄位，而 `@Id` 則加在 PK 的欄位上。這部分建好後，再建一個 `CarPK.java` ，這個 class 除了屬性和一般的 getter & setter 外，class 上面要加 `@Embeddable` ，表示它可以內嵌在 `Car.java` 中，另外還要覆寫 `hashCode()` 和 `equals()` 方法。

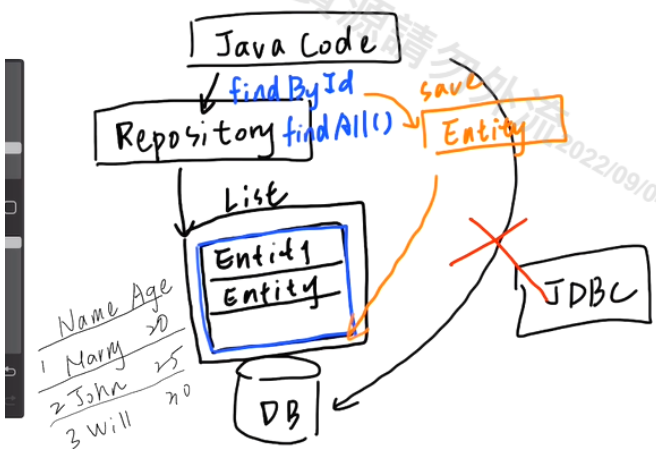
> 為什麼要覆寫 `hashCode()` 和 `equals()` ，除了 JPA 外，什麼樣的情況會覆寫這兩個方法？
> 如果資料庫沒有 PK，是不是就不能加 `@Id` ？

否，若沒加則無法判斷撈出來的資料是不是同一筆，但通常幾乎都有PK

先用hashcode判斷是否是同一筆資料(計算速度較快)，若無法判斷再用equals來判斷

Entity 建好後，下一步來建 Repository :

```
|--com.example.demospringboot
   |--DemospringbootApplication.java
|--com.example.demospringboot.configuration
   |--SwaggerConfig.java
   |--RestConfiguration.java
|--com.example.demospringboot.controller
   |--TestController.java
   |--ProductController.java
|--com.example.demospringboot.entity
   |--Car.java
   |--CarPK.java
|--com.example.demospringboot.model
   |--Product.java
|--com.example.demospringboot.repository
   |--CarRepository.java // 新增的檔案
|--com.example.demospringboot.service
   |--ProductService.java
|--com.example.demospringboot.service.impl
   |--ProductServiceImpl.java
```



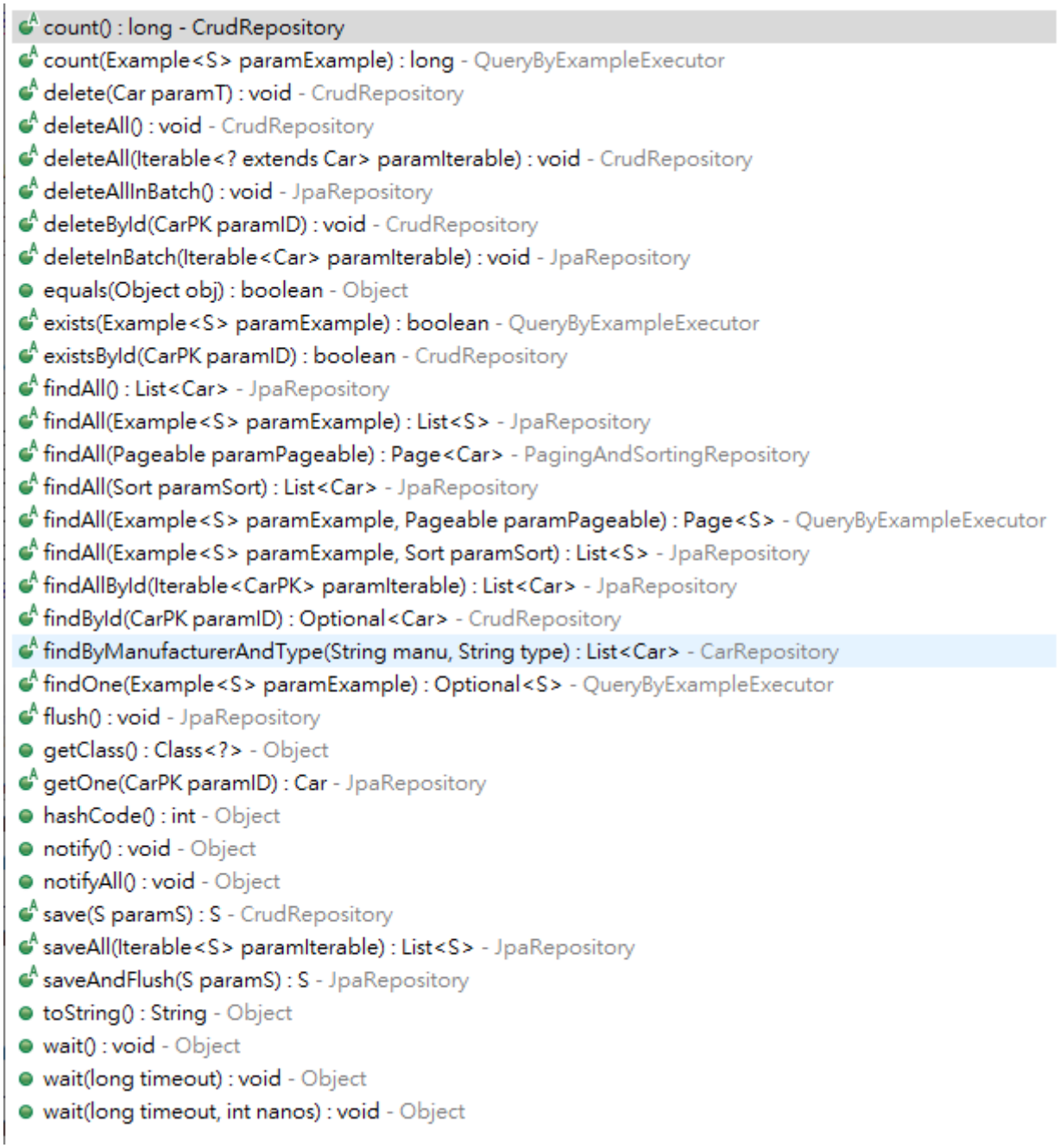
```java
import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.example.demo.entity.Car;

@Repository
public interface CarRepository extends JpaRepository<Car, CarPK> {
    List<Car> findByManufacturerAndType(String manu, String type);
}
```

這邊建立 `CarRepository.java` ，它繼承了 `JpaRepository.java` ，後面的泛型要放入對應到的 table 以及 PK 類別。如果點進去看 `JpaRepository.java` ，可以看到裡面其實還有很多預設方法：



以 `findAll()` 來說，它可以直接查詢到資料庫欄位所有的資料，注意這邊沒有寫下任何的 SQL 語句，只是單純的使用某個方法而已，卻可以對資料庫進行操作。

很多時候預設的方法並不符合查詢的需求，這時必需自行寫方法，範例中寫了一個 `findByManufacturerAndType(String manu, String type)` ，從它的名稱可以推測這個方法是想用 Manufacturer 跟 Type 進行查詢，這邊一樣沒有在程式碼裡寫下任何的 SQL，而是單純地用方法命名，去決定 SQL 要做的事情。不過，如果想偷懶不寫 SQL 就得遵守規則，可以看到方法名稱為了符合想做的 SQL 語句，名稱特別長，如果有想做其他的 SQL，可以在自行上網查詢 JPA 的方法命名規則，或是參考資料夾中的 JPA 文件。

最後 class 上面有個 annotation `@Repository` ，目的也跟前面的 `@Service` 相同，主要是為了讓 Spring Boot 的 `@ComponentScan` 掃到，並且把這個 Component 放到 Spring Boot的 Bean Pool 裡，因此改寫成 `@Component` 也可以。

> @Service 跟 @Repository 有什麼不同嗎？如果可以抽換成 @Component，為什麼要有這兩個 annotation？

參考

https://www.cnblogs.com/xiaowuzi/p/3485302.html