8. JPA-Service & Controller & Request

建好操作資料庫的 `@Repository` 後，再來要在 `@Service` 中使用它，並進行業務邏輯處理，先用全查當作範例：

```
|--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
|--com.example.demospringboot.service
   |--CarService.java // 新增的檔案
   |--ProductService.java
|--com.example.demospringboot.service.impl
   |--CarServiceImpl.java // 新增的檔案
   |--ProductServiceImpl.java
```

```java
import java.util. List;

import com.example.demo.entity. Car;

public interface CarService {

    List<Car> queryAllCar();

}
```

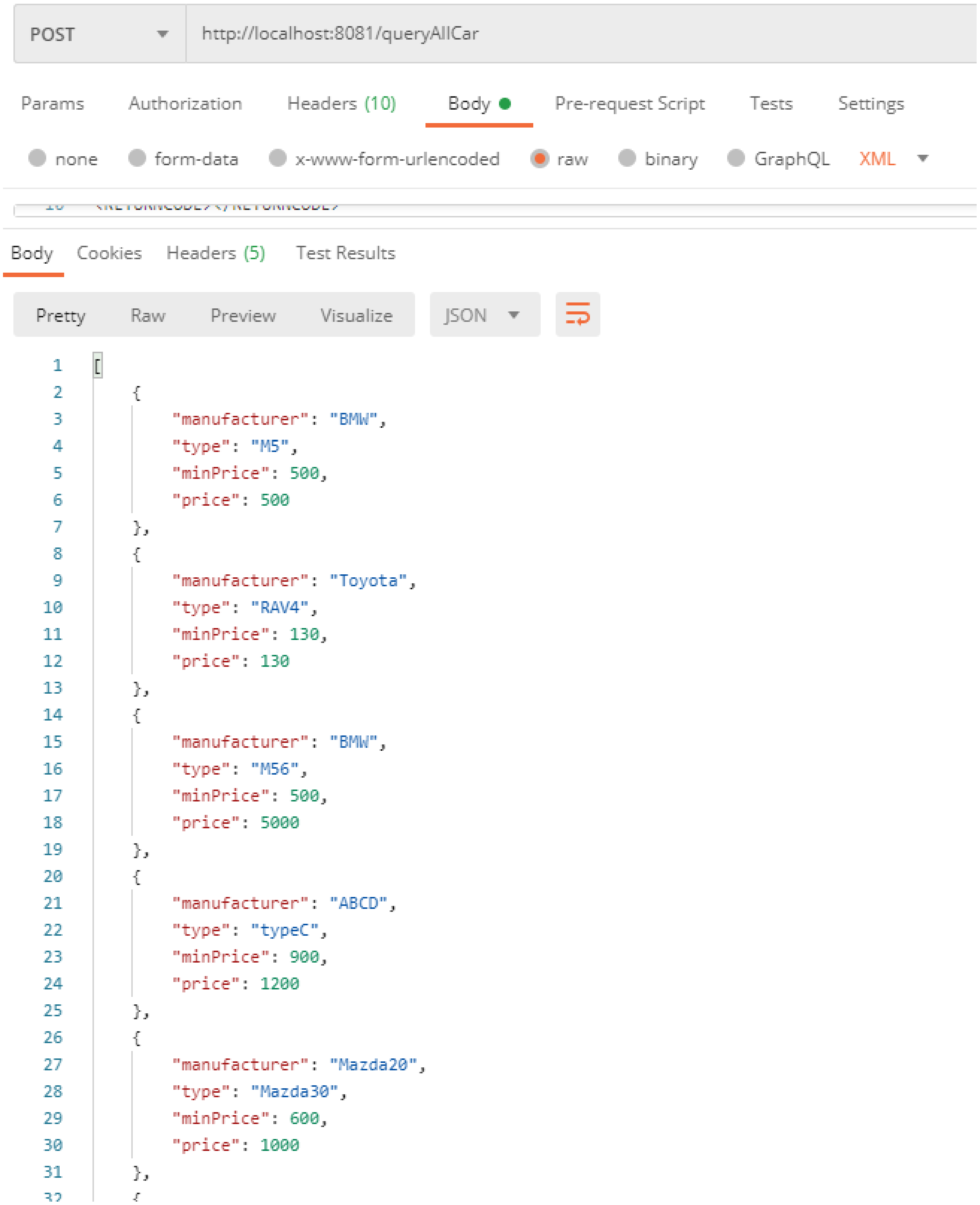
```java
@Service
public class CarServiceImpl implements CarService {
    @Autowired
    private CarRepository carRepository;

    @Override
    public List<Car> queryAllCar() {
        return carRepository.findAll();
    }
}
```

在 `@Service` 做好查詢的邏輯後，下一步是在 `@Controller` 呼叫：

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

```java
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.example.demo.entity.Car;
import com.example.demo.service.CarService;

@RestController
public class CarController {
    @Autowired
    CarService carService;

    @RequestMapping(value = "/queryAllCar", method = RequestMethod.GET)
    public List<Car> queryAllCar() {
        return carService.queryAllCar();
    }
}
```

用 Postman 進行測試：



```
[
    {
        "manufacturer": "BMW",
        "type": "M5",
        "minPrice": 500,
        "price": 500
    },
    {
        "manufacturer": "Toyota",
        "type": "RAV4",
        "minPrice": 130,
        "price": 130
    },
    {
        "manufacturer": "BMW",
        "type": "M56",
        "minPrice": 500,
        "price": 5000
    },
    {
        "manufacturer": "ABCD",
        "type": "typeC",
        "minPrice": 900,
        "price": 1200
    },
    {
        "manufacturer": "Mazda20",
        "type": "Mazda30",
        "minPrice": 600,
        "price": 1000
    },
    {
```

基本上這就完成整個 Web Service 的操作，用 Postman 發送 request 會進到 `@Controller` 接口，`@Controller` 負責將打進來的 request 導到正確的 `@Service` ，`@Service` 會做業務邏輯操作，如果要用到 DB 的資料，再交由 `@Repository` 來負責。

那如果用 `findByManufacturerAndType(String manu, String type)` 這個方法呢？也就是說需要使用傳進來的參數來進行查詢，這時可以跟之前一樣，先定義好傳進來的資料格式該有的樣子：

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

```java
import java.math.BigDecimal;

import com.fasterxml.jackson.annotation.JsonProperty;

public class CarRequest {
    @JsonProperty("Manufacturer")
    private String manufacturer;

    @JsonProperty("Type")
    private String type;

    public String getManufacturer() {
        return manufacturer;
    }

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

    public String getType() {
        return type;
    }

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

```java
import java.math.BigDecimal;

import com.fasterxml.jackson.annotation.JsonProperty;

public class CarResponse {
    @JsonProperty
    private String manufacturer;

    @JsonProperty
    private String type;

    @JsonProperty
    private BigDecimal minPrice;

    @JsonProperty
    private BigDecimal price;

    public String getManufacturer() {
        return manufacturer;
    }

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

    public String getType() {
        return type;
    }

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

    public BigDecimal getMinPrice() {
        return minPrice;
    }

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

    public BigDecimal getPrice() {
        return price;
    }

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

`CarRequest.java` 是定義進來 request 該有的格式， `CarResponse.java` 則是定義回傳 response 該有的格式，另外每個屬性上面的 `@JsonProperty` ，則是定義傳進來的 json 欄位名稱，好比以上面建好的 `CarRequest.java` 來說，格式應該如下：

```
{
    Manufacturer:"Audi",
    Type:"A5",
    Min_Price:500,
    Price:500
}
```

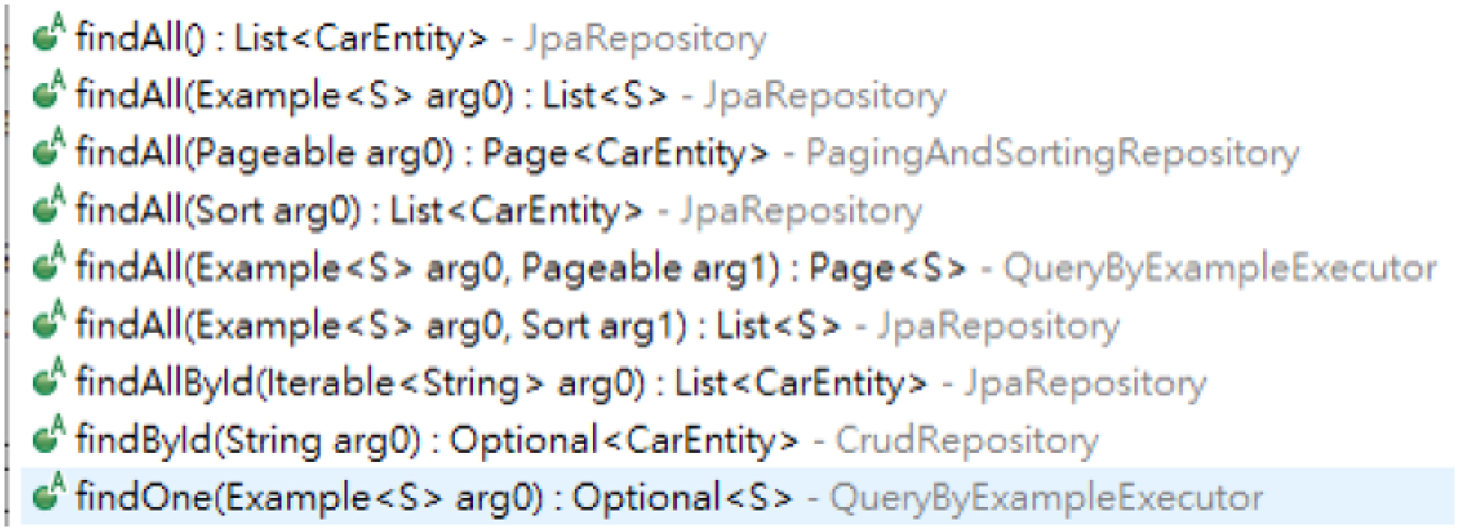
如果 `@JsonProperty` 沒寫 value，那麼 json 欄位的名稱就會跟 class 中的屬性相同：

```
{
    manufacturer:"Audi",
    type:"A5",
    minPrice:500,
    price:500
}
```

建立好 `CarRequest.java` 、 `CarResponse.java` 的資料格式後，可以再次嘗試把整個流程串起來，先定義 `@Controller` 導向 `@Service` ，然後寫下 `@Service` 要做的業務邏輯，如果要用到 DB 操作再使用 `@Repository` ：

```
@RestController
public class CarController {
    @Autowired
    CarService carService;

    @RequestMapping(value = "/queryAllCar", method = RequestMethod.POST)
    public List<Car> queryAllCar() {
        return carService.queryAllCar();
    }

    @RequestMapping(value = "/query", method = RequestMethod.POST)
    public CarResponse queryCar(@RequestBody CarRequest carRequest) {
        return carService.queryCar(carRequest);
    }
}
```

在 Repository 的部分，預設的方法其實沒有 `findByManufacturerAndType` 這個方法，所以我們需要在 Repository 內依照命名規範來新增需要的查詢語句，命名規範的部分可以參考講義最後下面的連結。



```
findAll() : List<CarEntity> - JpaRepository
findAll(Example<S> arg0) : List<S> - JpaRepository
findAll(Pageable arg0) : Page<CarEntity> - PagingAndSortingRepository
findAll(Sort arg0) : List<CarEntity> - JpaRepository
findAll(Example<S> arg0, Pageable arg1) : Page<S> - QueryByExampleExecutor
findAll(Example<S> arg0, Sort arg1) : List<S> - JpaRepository
findAllById(Iterable<String> arg0) : List<CarEntity> - JpaRepository
findById(String arg0) : Optional<CarEntity> - CrudRepository
findOne(Example<S> arg0) : Optional<S> - QueryByExampleExecutor
```

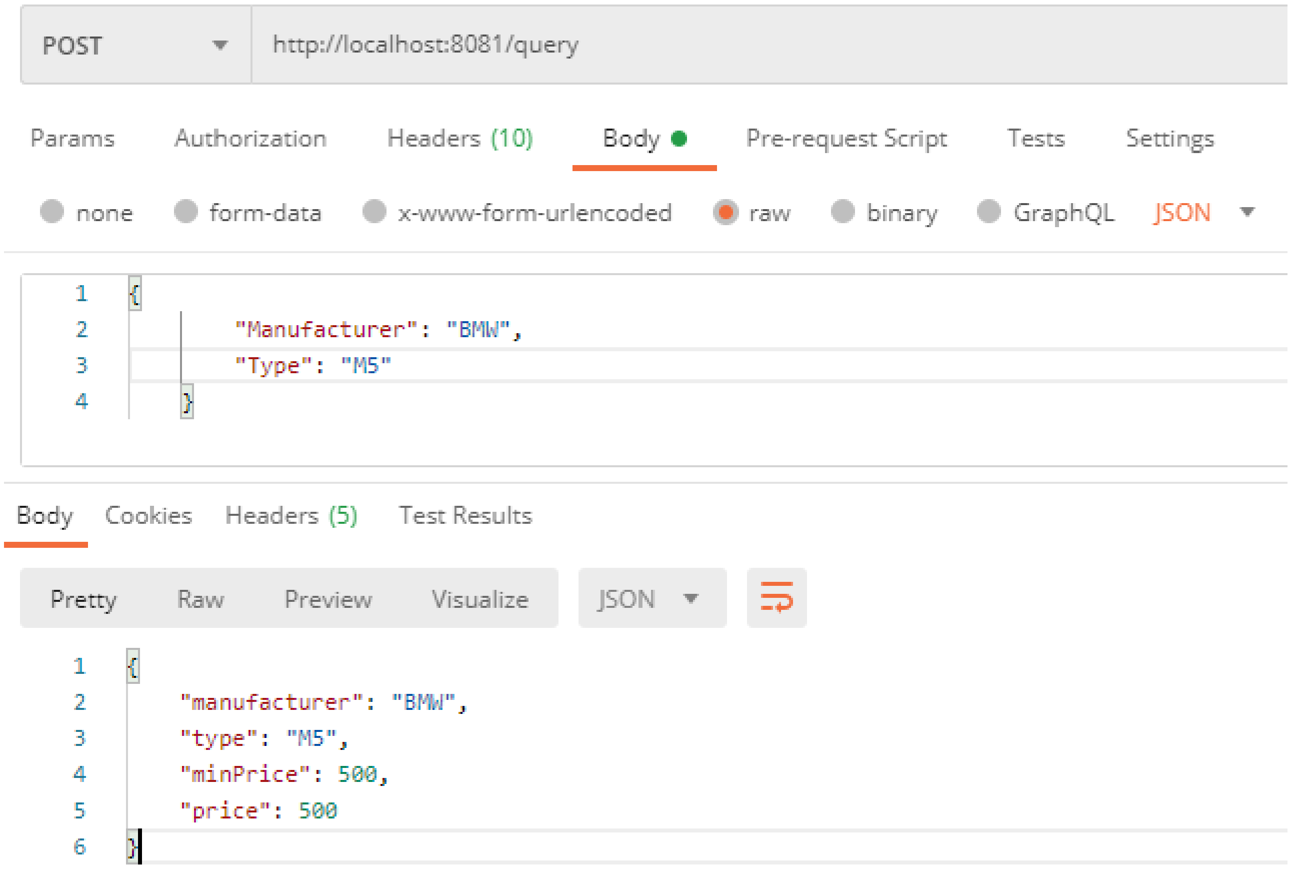
```
@Repository
public interface CarRepository extends JpaRepository<CarEntity, String> {
    List<CarEntity> findByManufacturerAndType(String manu, String type);
}
```

新增後就可以在 ServiceImpl 內調用這個方法。

```
public interface CarService {

    List<Car> queryAllCar();

    CarResponse queryCar(CarRequest carRequest);

}
```

```
@Service
public class CarServiceImpl implements CarService {
    @Autowired
    private CarRepository carRepository;

    @Override
    public List<Car> queryAllCar() {
        return carRepository.findAll();
    }

    @Override
    public CarResponse queryCar(CarRequest carRequest) {
        String manufacturer = carRequest.getManufacturer();
        String type = carRequest.getType();
        List<Car> list = carRepository.findByManufacturerAndType(manufacturer, type);

        Car car = list.get(0);
        CarResponse carResponse = new CarResponse();
        carResponse.setManufacturer(car.getManufacturer());
        carResponse.setType(car.getType());
        carResponse.setPrice(car.getPrice());
        carResponse.setMinPrice(car.getMinPrice());
        return carResponse;
    }
}
```

用 Postman 進行測試：


POST http://localhost:8081/query

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{
    "Manufacturer": "BMW",
    "Type": "M5"
}
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "manufacturer": "BMW",
    "type": "M5",
    "minPrice": 500,
    "price": 500
}
```

> 嘗試做做看 save 的操作。

```java
@RequestMapping(value = "/create", method = RequestMethod.POST)
    public String createCar() {
```

這邊最好還是用GET

```java
        CarPK pk = new CarPK();
        pk.setManufacturer("Boyee");
        pk.setType("Man");

        // 找目標
        Optional<Car> car = carRepository.findById(pk);

        // 沒有則新增
        if(car.isEmpty()) {
                Car newCar = new Car();
                newCar.setManufacturer("Boyee");
                newCar.setType("Man");
                carRepository.save(newCar);
                return "create succesful";
        }

        return "create fail";
    }

    @RequestMapping(value = "/put", method = RequestMethod.POST)
    public String putCar() {
```

這邊最好還是用GET

```java
        CarPK pk = new CarPK();
        pk.setManufacturer("Boyee");
        pk.setType("Man");

        // 找目標
        Optional<Car> car = carRepository.findById(pk);

        // 有則修改
        if(car.isPresent()) {
                car.get().setPrice(new BigDecimal("10"));
                carRepository.save(car.get());
                return "put succesful";
        }

        return "put fail";
    }
```

這邊最好還是用GET

```java
    @RequestMapping(value = "/delete", method = RequestMethod.POST)
    public String deleteCar() {
        CarPK pk = new CarPK();
        pk.setManufacturer("Boyee");
        pk.setType("Man");
```

補充：
雖然此處用不到，但若遇到當Option<T>內部為空時需要丟出例外的情況，可以加上orElseThrow(() -> new Exception)，簡化程式碼

```java
        // 找目標
        Optional<Car> car = carRepository.findById(pk);

        // 有則刪除
        if(car.isPresent()) {
                carRepository.delete(car.get());
                return "delete succesful";
        }

        return "delete fail";
    }
```

参考

https://blog.csdn.net/sbin456/article/details/53304148