

4. Controller-建立RESTful API

上個部分有提到 RESTful API，這邊來提一下 HTTP request、response 的結構，所謂的 HTTP 全名是 Hypertext Transfer Protocol (超文本傳輸協定)，是一種網路溝通的概念與過程，既然都說是協定了，那 request 跟 response 當然具有既定的格式，整個格式大概像是下面這張表：

HTTP Request	HTTP Response
Request Headers	Response Headers
Request Method	Response Status
Request URL	Response Body
Request Body	

這邊先有個概念即可，之後會嘗試操作 `RequestEntity`、`ResponseEntity` 這兩個類別，就可以拿到 Headers、Body 的內容。

把 `TestController.java`，改成下面的形式：

```

package com.example.demospringboot.controller;

import java.util.HashMap;
import java.util.Map;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

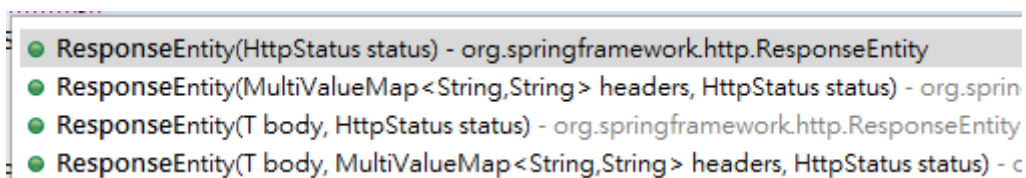
@Controller
public class TestController {
    @RequestMapping(value = "/map", method = RequestMethod.GET)
    public ResponseEntity<Map<String, String>> testMap() {
        Map<String, String> map = new HashMap<>();
        map.put("key", "value");
        return new ResponseEntity<>(map, HttpStatus.OK);
    }

    @RequestMapping(value = "/string", method = RequestMethod.GET)
    public ResponseEntity<String> testString() {
        String string = "testString";
        return new ResponseEntity<>(string, HttpStatus.OK);
    }
}

```

在 `@RequestMapping` 裡面，加入了兩個設定，一個是 `value`，它用來設定 `url` 的位置，另一個則是 `method`，它用來限制 `HTTP method`，這邊都先限制只能使用 `GET` 方法，如果沒有設定 `HTTP method`，任何一種 `HTTP method` 都可以打進來。另外除了 `@RequestMapping` 外，也可使用 `@PostMapping` 或是 `@GetMapping`，這種寫法不需再額外設定 `HTTP method`。

而回傳的物件就使用 `ResponseEntity`，首先看一下這物件的建構子，有圖片中的四種方式，這邊採用第三種方式來創建 `ResponseEntity`，有興趣的人可自行查看其他種在建構物件上該怎麼使用。第一個參數用來設定 `Response Body`，`Response Body` 要放入 `ResponseEntity` 中定義的泛型，上面的範例是採用 `String` 跟 `Map`，第二個參數則是用來設定 `Response Status`，最常見的是 `404 Not Found` 的錯誤，那這邊的情況會是正常拿到 `Response`，設定成 `200` 就可以。



```

• ResponseEntity(HttpStatus status) - org.springframework.http.ResponseEntity
• ResponseEntity(MultiValueMap<String,String> headers, HttpStatus status) - org.sprin
• ResponseEntity(T body, HttpStatus status) - org.springframework.http.ResponseEntity
• ResponseEntity(T body, MultiValueMap<String,String> headers, HttpStatus status) - c

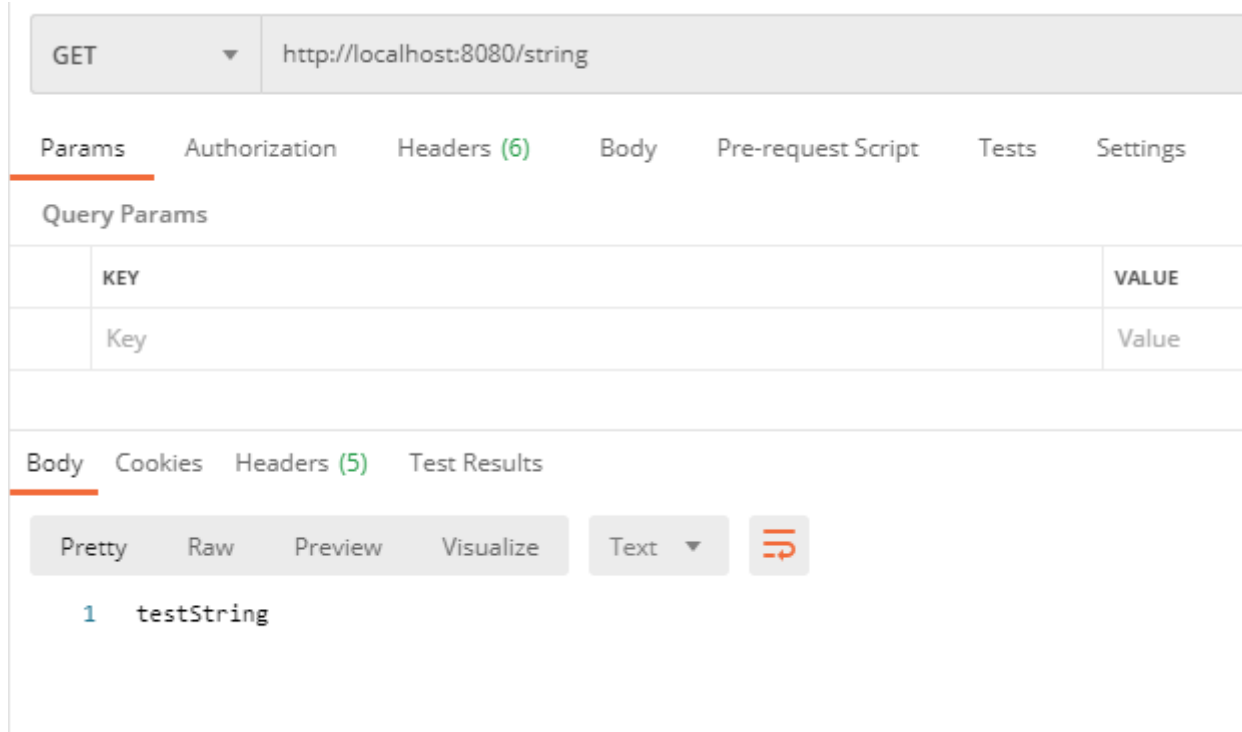
```

T body能放的東西
可以是任意物件

跟前面建構的 `TestController.java` 相比，有人可能會好奇，為什麼 `method` 不需加上 `@ResponseBody`，原因在於之前的例子希望回傳的東西就是 `Response Body`、`Response Headers`、`Response Status`

也不是不需要，只是用預設的就可以。而這邊是直接回傳整個 `ResponseEntity` 物件，因此就不需要加上 `@ResponseBody`。

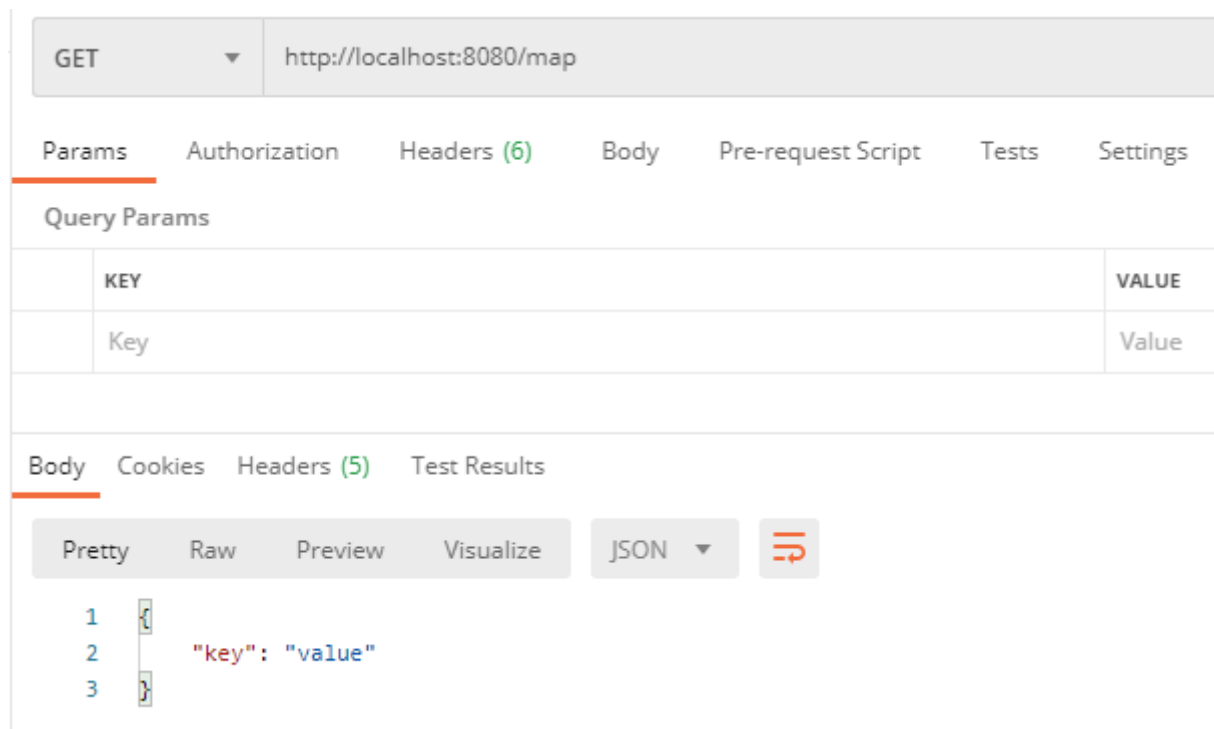
建立好之後，嘗試使用 Postman 測試自己的 API：



A screenshot of the Postman application showing a GET request to `http://localhost:8080/string`. The interface includes tabs for Params, Authorization, Headers (6), Body, Pre-request Script, Tests, and Settings. The Params tab is active, displaying a table for Query Params with columns KEY and VALUE. The table contains one entry: Key with value Value. Below the table, there are tabs for Body, Cookies, Headers (5), and Test Results. The Body tab is active, showing a text response `testString` with a line number 1. The response is displayed in a 'Text' format.

KEY	VALUE
Key	Value

Body: `1 testString`



A screenshot of the Postman application showing a GET request to `http://localhost:8080/map`. The interface is similar to the previous one, with the Params tab active and a table for Query Params containing one entry: Key with value Value. The Body tab is active, showing a JSON response with a line number 1. The response is displayed in a 'JSON' format.

KEY	VALUE
Key	Value

Body: `1 {
2 "key": "value"
3 }`

操作完 `ResponseEntity` 後，下一步嘗試操作 `RequestEntity`，程式碼如下：

```

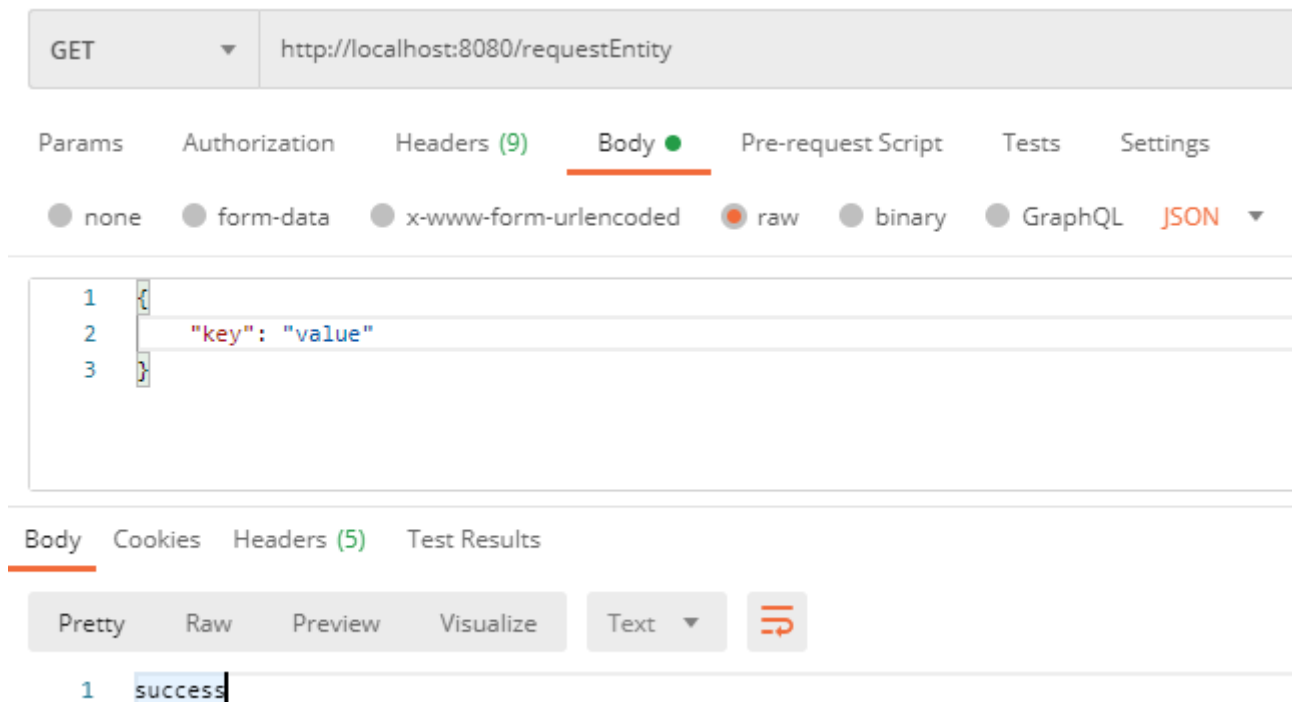
@RequestMapping(value = "/requestEntity", method = RequestMethod. POST)
public ResponseEntity<String> requestEntity(RequestEntity<Map<String, String>> requestEntity) {

    System.out.println(requestEntity.getHeaders());
    System.out.println(requestEntity.getBody());
    return new ResponseEntity<>("success", HttpStatus.OK);

}

```

同樣可用Postman來進行測試，記得發 Request Headers 的 Content-Type 要改成 application/json，然後在 Body 的部分新增一筆 Json 資料：



在 eclipse 的 console 中，可以看到 Request Headers 跟 Request Body 的內容。



很多時候不會用到 Header 裡所塞的資訊，只會用 Body 的內容，這時可用 @RequestBody 取出 Request 中 Body 的內容，除此之外，可自行定義一個 class，用來規定 Request Body 裡應有的格式，程式碼如下 Product.java：

```
|--com.example.demospringboot
    |--DemospringbootApplication.java
|--com.example.demospringboot.configuration
    |--SwaggerConfig.java
|--com.example.demospringboot.controller
    |--TestController.java
|--com.example.demospringboot.model
    |--Product.java // 新增的檔案
```

```
public class Product {
    public String id;

    public String name;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Controller 的部分如下：

```
|--com.example.demospringboot
    |--DemospringbootApplication.java
|--com.example.demospringboot.configuration
    |--SwaggerConfig.java
|--com.example.demospringboot.controller
    |--TestController.java
    |--ProductController.java // 新增的檔案
|--com.example.demospringboot.model
    |--Product.java
```

參考：<https://mvnrepository.com/artifact/org.projectlombok/lombok/1.18.20> 避免產生太多getter and setter影響閱讀的方式而使用的dependency

在class前一行加入@Getter & @Setter，可自動產生getter and setter，當然還有其他annotation可以使用，例如：@Data、@ToString...

其中一種安裝方式：

1. 一開始先放入dependency

2. 之後記得在eclipse.ini中加入設定檔

舉例：-javaagent:C:\Users\Steven\eclipse\jee-2022-06\eclipse\lombok-1.18.20.jar

@RestController

```
public class ProductController {  
    private static Map<String, Product> productRepo = new HashMap<>();  
    static {  
        Product honey = new Product();  
        honey.setId("1");  
        honey.setName("Honey");  
        productRepo.put(honey.getId(), honey);  
  
        Product almond = new Product();  
        almond.setId("2");  
        almond.setName("Almond");  
        productRepo.put(almond.getId(), almond);  
    }  
}
```

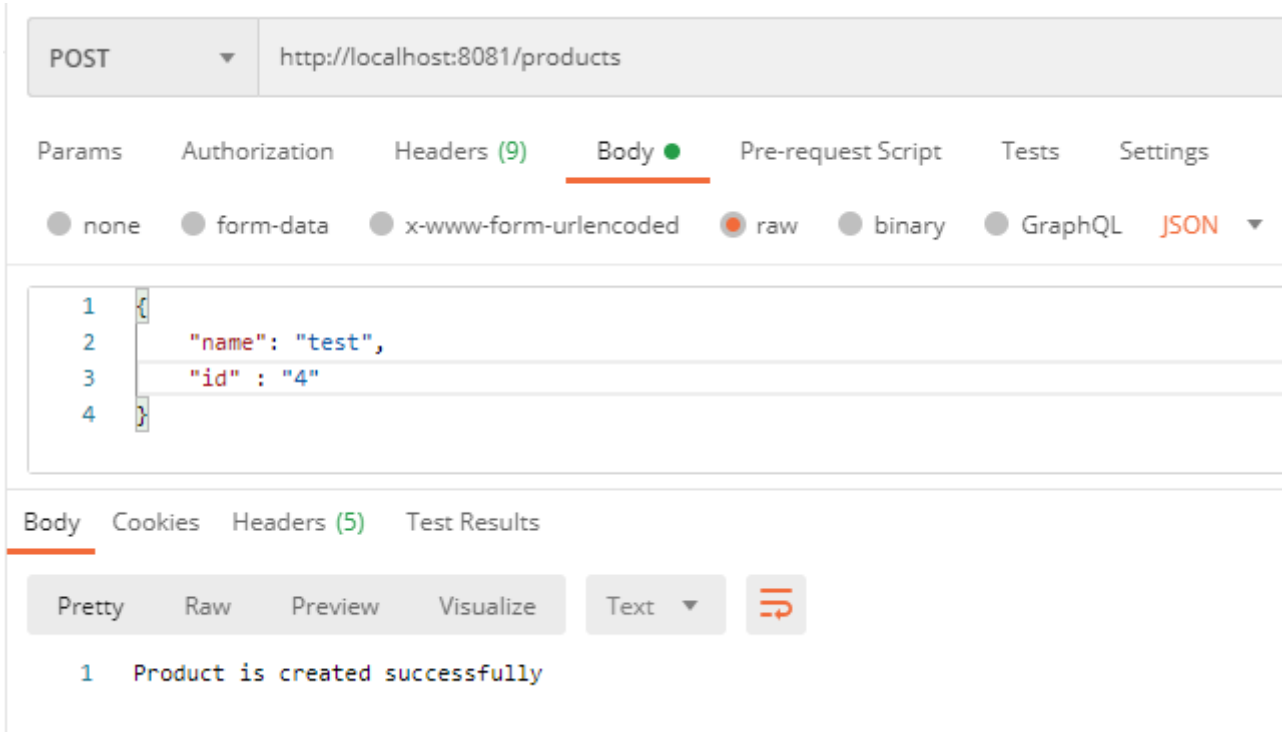
@PathVariable通常用在只傳一個參數進來的狀況下，且不常用

```
@RequestMapping(value = "/products/{id}", method = RequestMethod.DELETE)  
public ResponseEntity<Object> delete(@PathVariable("id") String ID) {  
    productRepo.remove(ID);  
    return new ResponseEntity<>("Product is deleted successssfully", HttpStatus.OK);  
}
```

```
@RequestMapping(value = "/products/{id}", method = RequestMethod.PUT)  
public ResponseEntity<Object> updateProduct(@PathVariable("id") String Id  
    , @RequestBody Product product) {  
    productRepo.remove(Id);  
    productRepo.put(product.getId(), product);  
    return new ResponseEntity<>("Product is updated successssfully", HttpStatus.OK);  
}
```

```
@RequestMapping(value = "/products", method = RequestMethod.POST)  
public ResponseEntity<Object> createProduct(@RequestBody Product product) {  
    productRepo.put(product.getId(), product);  
    return new ResponseEntity<>("Product is created successfully", HttpStatus.CREATED);  
}
```

```
@RequestMapping(value = "/products", method = RequestMethod.GET)  
public ResponseEntity<Object> getProduct() {  
    return new ResponseEntity<>(productRepo.values(), HttpStatus.OK);  
}  
}
```



目前先把業務邏輯寫在 `@Controller` 中，資料庫的操作也不管，先使用 `static block` 假裝有資料庫，下個章節會把資料的操作放在 `Service` 層中。這邊有個 annotation `@PathVariable`，用途是抓 url 裡大括號的字串當成參數，像 `delete method` 就把大括號裡 `id` 的字串當成 `method` 的其中一項參數。

`@RestController` 、 `@Controller` 差在哪？

參考

<https://medium.com/pierceshih/筆記-何謂-http-傳輸協定-1d9b5be3fd24>

https://www.tutorialspoint.com/spring_boot/spring_boot_consuming_restful_web_services.htm

<https://www.baeldung.com/spring-controller-vs-restcontroller>