
Java EE 7 Web TomEE 與 Restful API

鄭安翔

ansel_cheng@hotmail.com

課程大綱

- 1) **TomEE**
- 2) **Restful API**

Tomcat vs. TomEE

■ Tomcat 為開發、測試用的伺服器

- ❑ 實際的維運環境使用大型應用程式伺服器
 - Oracle WebLogic、IBM WebSphere
- ❑ 不支援 Java EE 6.0定義的 Web Profile
 - 容器中嵌入Embedded EJB Container，可直接開發EJB元件
 - 簡化Java EE應用程式的開發

■ TomEE

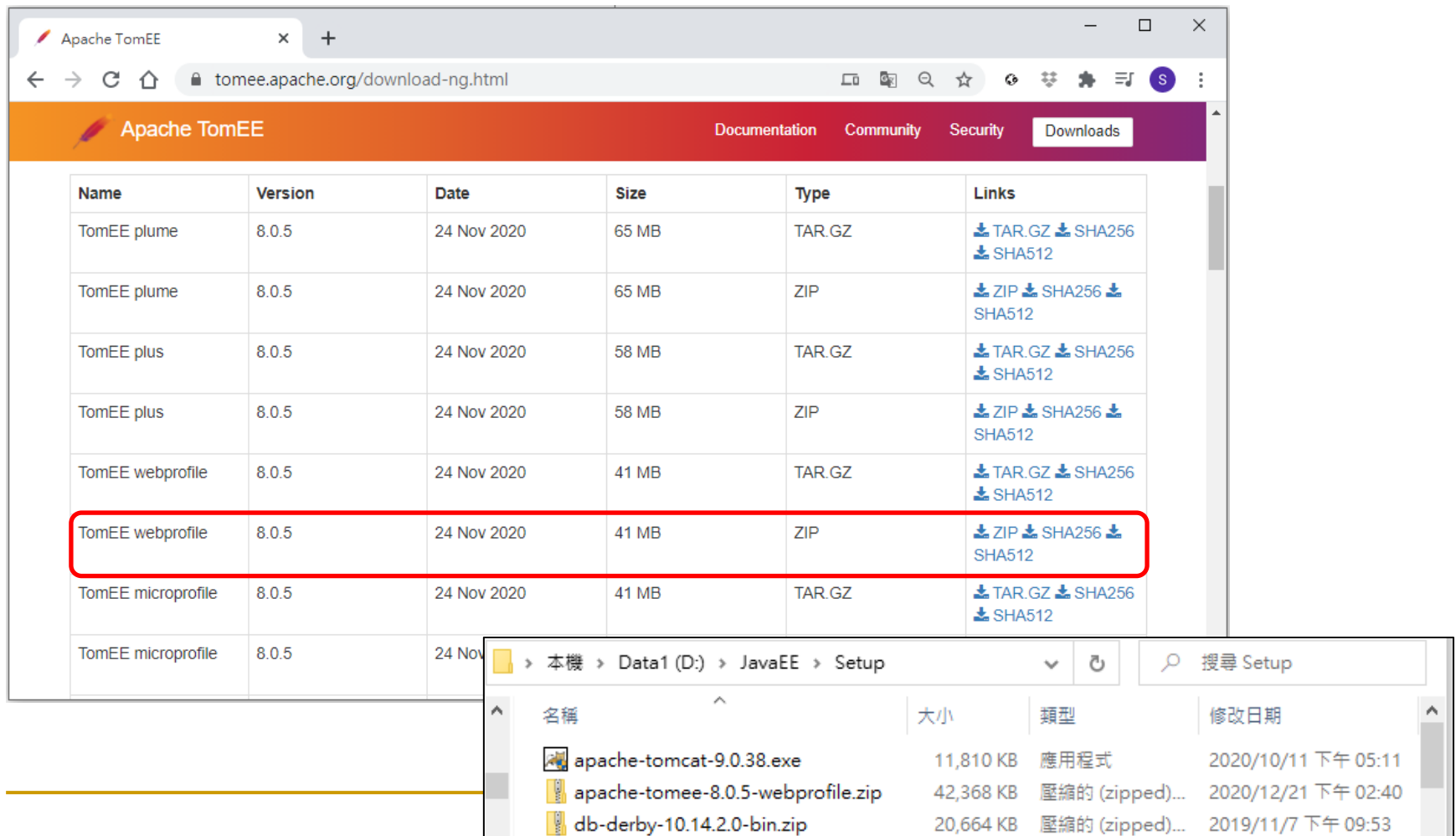
- ❑ 以Tomcat為核心
 - 具羽量、敏捷、啟動速度快的特性
- ❑ 集合Apache上相關Java EE的OpenSource實作

Java EE 容器比較

| | Tomcat | TomEE | TomEE JAXRS | TomEE+ | TomEE PluME | OpenEJB |
|--|--------|-------|-------------|--------|-------------|---------|
| Java Servlets | ✓ | ✓ | ✓ | ✓ | ✓ | |
| Java ServerPages (JSP) | ✓ | ✓ | ✓ | ✓ | ✓ | |
| Java ServerFaces (JSF) | | ✓ | ✓ | ✓ | ✓ | |
| Java Transaction API (JTA) | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Java Persistence API (JPA) | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Java Contexts and Dependency Injection (CDI) | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Java Authentication and Authorization Service (JAAS) | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Java Authorization Contract for Containers (JACC) | | ✓ | ✓ | ✓ | ✓ | ✓ |
| JavaMail API | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Bean Validation | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Enterprise JavaBeans | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Java API for RESTful Web Services (JAX-RS) | | | ✓ | ✓ | ✓ | ✓ |
| Java API for XML Web Services (JAX-WS) | | | | ✓ | ✓ | ✓ |
| Java EE Connector Architecture | | | | ✓ | ✓ | ✓ |
| Java Messaging Service (JMS) | | | | ✓ | ✓ | ✓ |
| EclipseLink | | | | | ✓ | |
| Mojarra | | | | | ✓ | |

下載 TomEE webprofile

- <https://tomee.apache.org/download-ng.html>



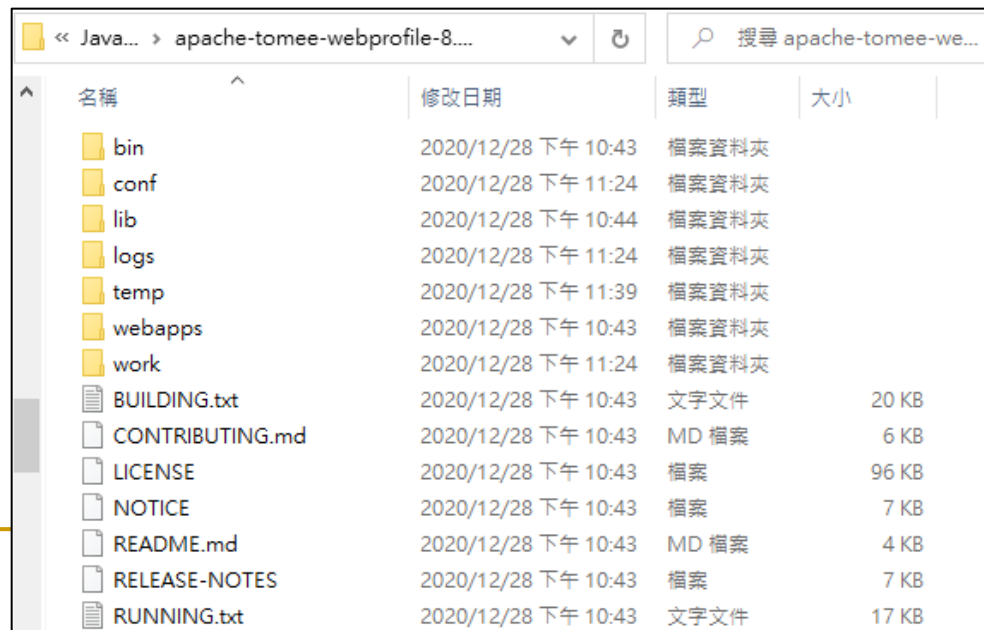
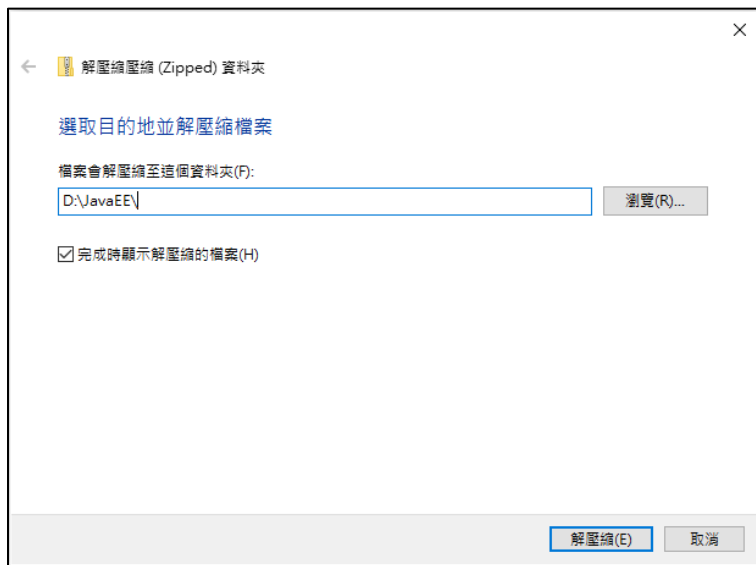
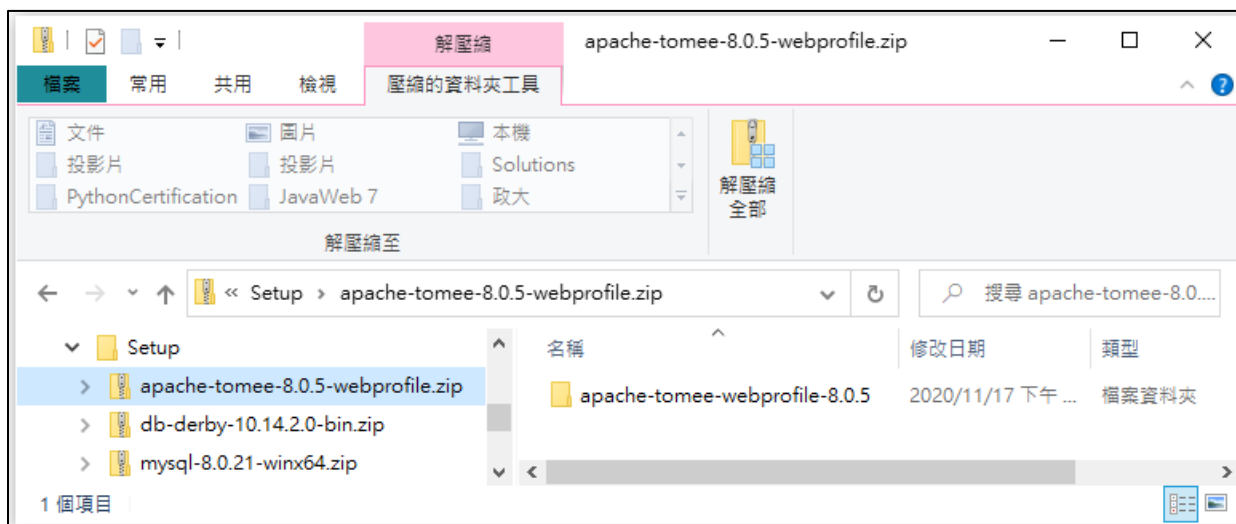
The screenshot shows the Apache TomEE download page in a web browser. The page has a navigation bar with links to Documentation, Community, Security, and Downloads. Below the navigation bar is a table listing various TomEE distributions. The table has columns for Name, Version, Date, Size, Type, and Links. The row for TomEE webprofile (ZIP) is highlighted with a red rectangle. Below the table, a file explorer window is open, showing the contents of the 'Data1 (D:) > JavaEE > Setup' directory. The file explorer shows three files: apache-tomcat-9.0.38.exe, apache-tomee-8.0.5-webprofile.zip, and db-derby-10.14.2.0-bin.zip. The file apache-tomee-8.0.5-webprofile.zip is highlighted.

| Name | Version | Date | Size | Type | Links |
|--------------------|---------|-------------|-------|--------|--|
| TomEE plume | 8.0.5 | 24 Nov 2020 | 65 MB | TAR.GZ | TAR.GZ SHA256 SHA512 |
| TomEE plume | 8.0.5 | 24 Nov 2020 | 65 MB | ZIP | ZIP SHA256 SHA512 |
| TomEE plus | 8.0.5 | 24 Nov 2020 | 58 MB | TAR.GZ | TAR.GZ SHA256 SHA512 |
| TomEE plus | 8.0.5 | 24 Nov 2020 | 58 MB | ZIP | ZIP SHA256 SHA512 |
| TomEE webprofile | 8.0.5 | 24 Nov 2020 | 41 MB | TAR.GZ | TAR.GZ SHA256 SHA512 |
| TomEE webprofile | 8.0.5 | 24 Nov 2020 | 41 MB | ZIP | ZIP SHA256 SHA512 |
| TomEE microprofile | 8.0.5 | 24 Nov 2020 | 41 MB | TAR.GZ | TAR.GZ SHA256 SHA512 |
| TomEE microprofile | 8.0.5 | 24 Nov 2020 | 41 MB | TAR.GZ | TAR.GZ SHA256 SHA512 |

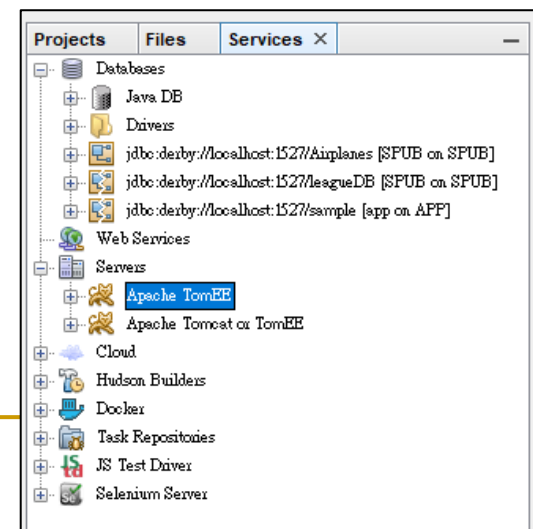
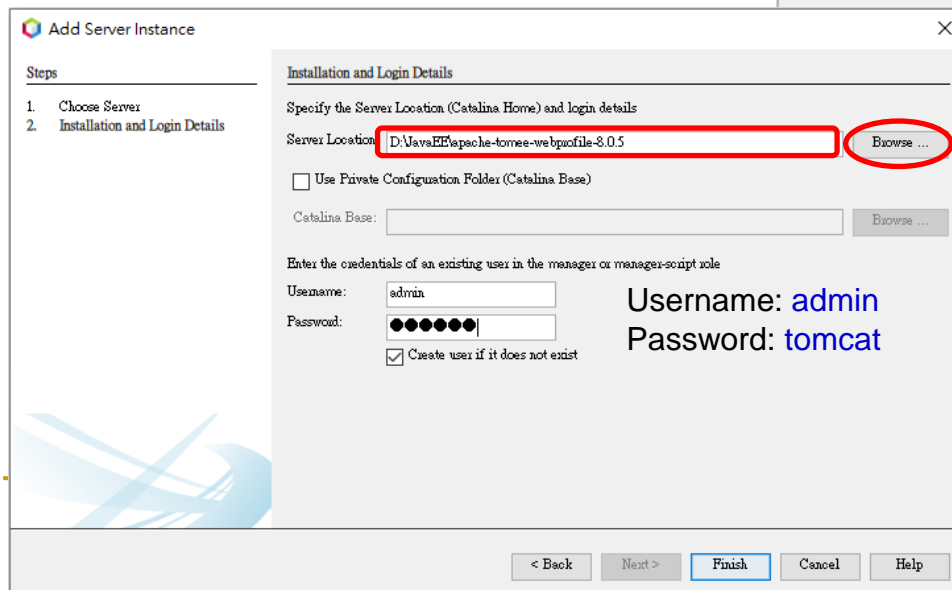
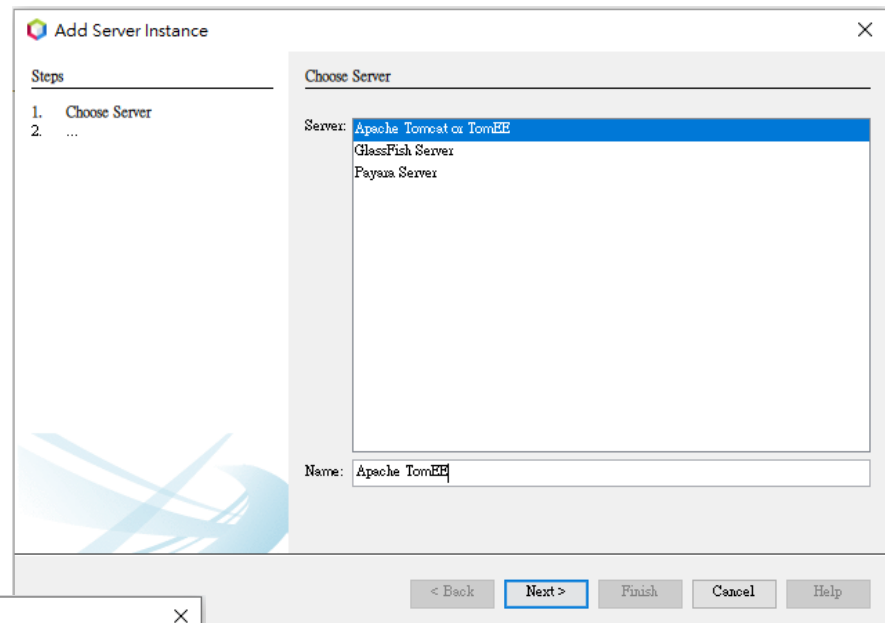
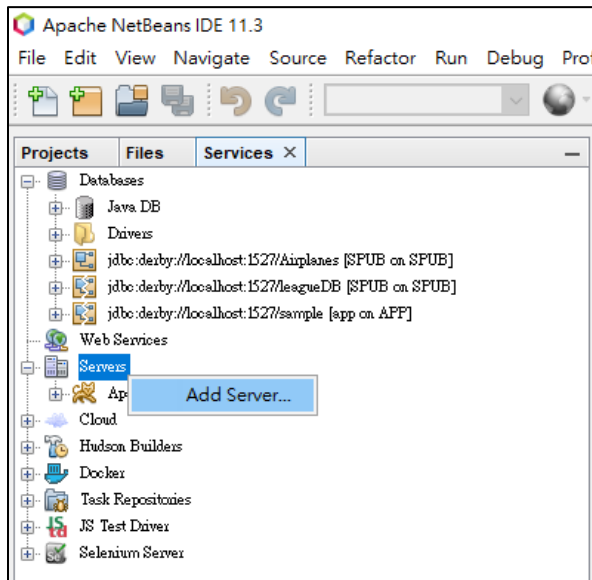
File Explorer: Data1 (D:) > JavaEE > Setup

| 名稱 | 大小 | 類型 | 修改日期 |
|-----------------------------------|-----------|-----------------|---------------------|
| apache-tomcat-9.0.38.exe | 11,810 KB | 應用程式 | 2020/10/11 下午 05:11 |
| apache-tomee-8.0.5-webprofile.zip | 42,368 KB | 壓縮的 (zipped)... | 2020/12/21 下午 02:40 |
| db-derby-10.14.2.0-bin.zip | 20,664 KB | 壓縮的 (zipped)... | 2019/11/7 下午 09:53 |

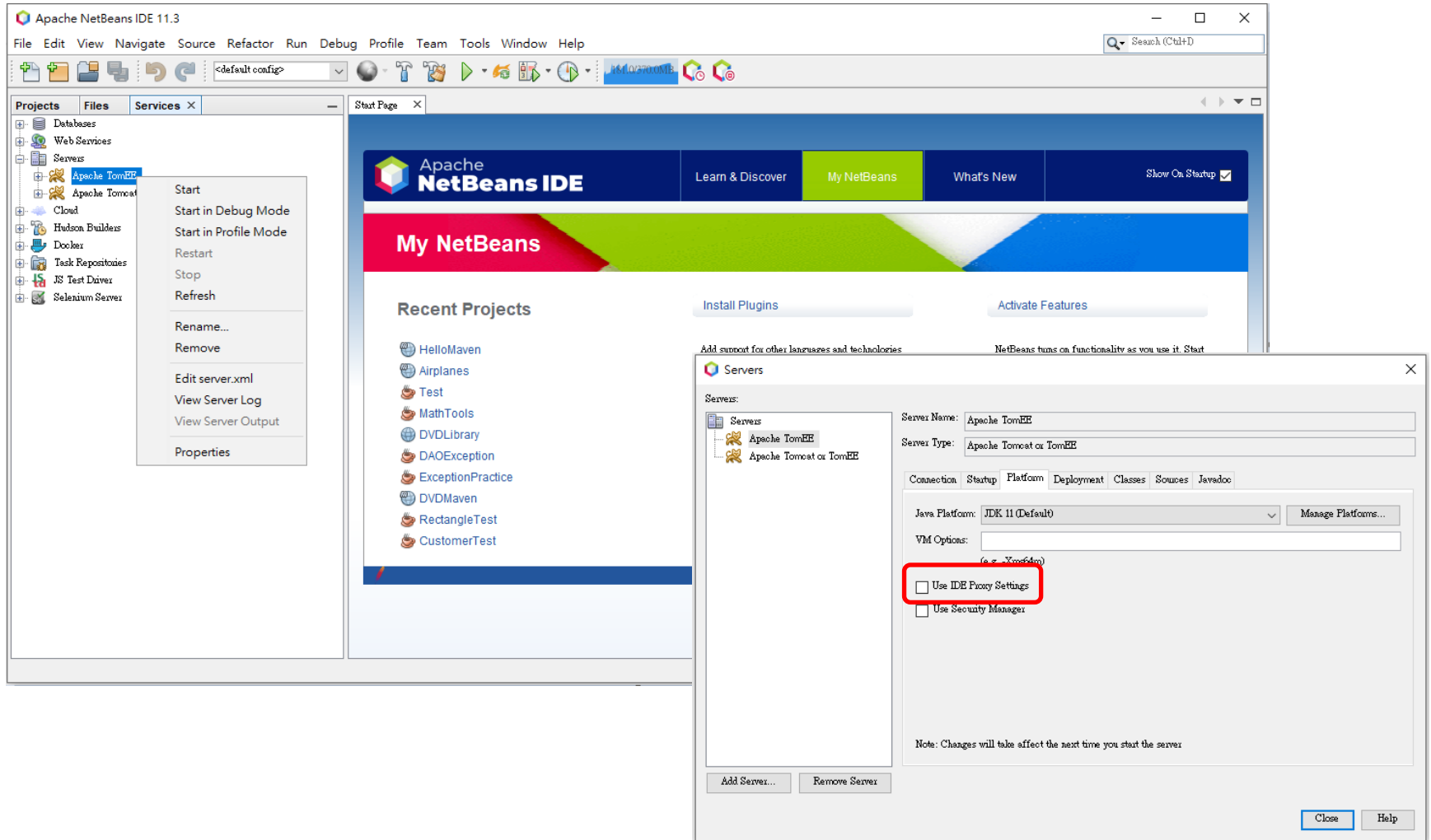
解壓縮 TomEE webprofile



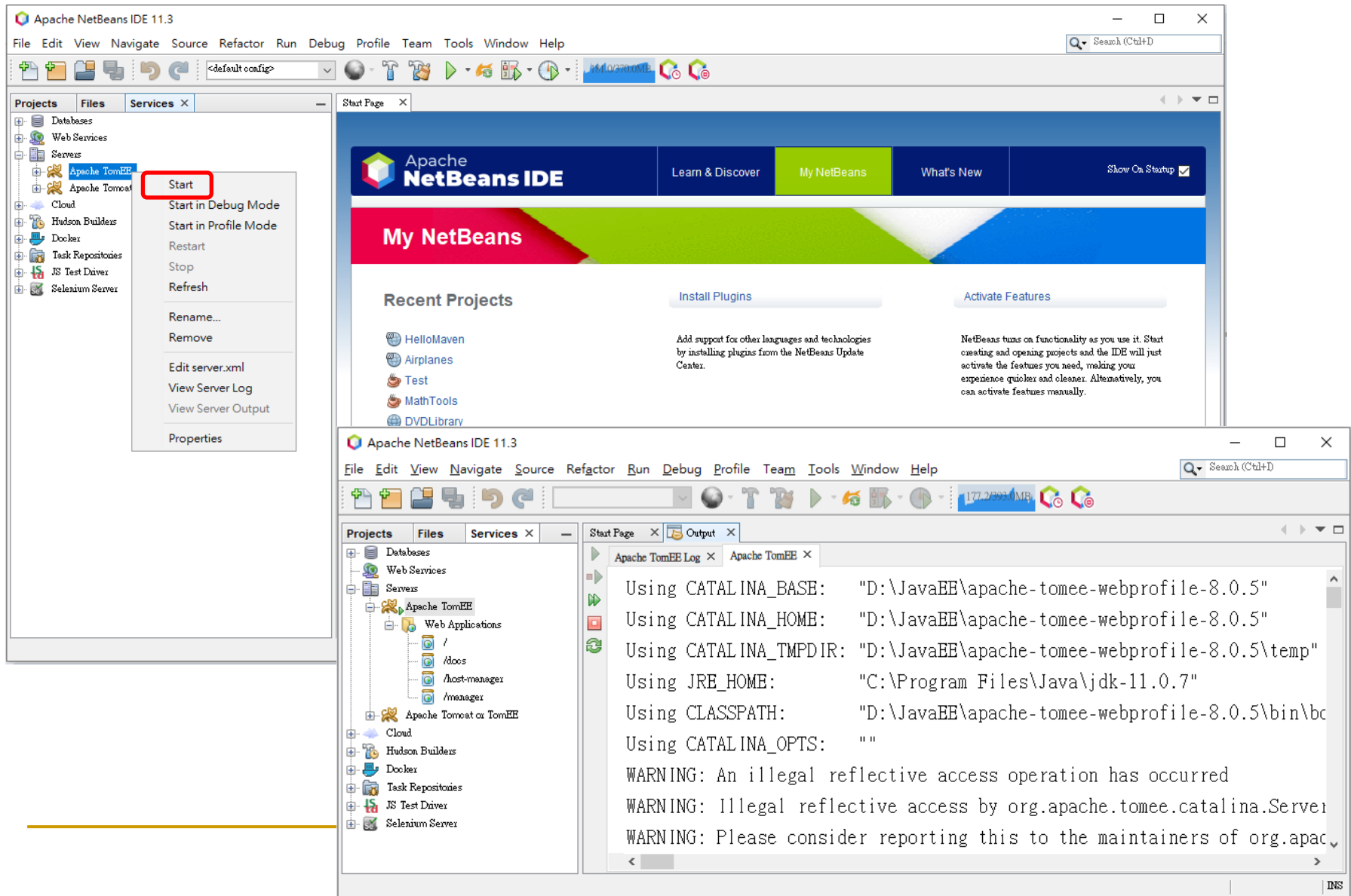
NetBeans 設定 TomEE



No Proxy 設定



啟動 TomEE



課程大綱

- 1) TomEE
- 2) **Restful API**

REST/RESTful 風格

- **R**epresentational **S**tate **T**ransfer(REST)
 - Roy Thomas Fielding 2000年提出
 - 一種網路服務應用程式的設計風格
 - HTTP通訊協定
 - 便於不同軟體/程式在網路中互相傳遞資訊
 - 符合此架構風格的網站稱為 **REST / RESTful** 網路服務
 - 用戶端發出存取和操作網路資源的請求**URI**
 - 代表預先定義好某無狀態操作

REST 架構應用情境

■ REST 架構應用情境

- 客戶端-伺服器 (Client-Server)
- 無狀態 (Stateless)
- 可快取 (Cacheability)
- 統一介面 (Uniform Interface)
- 分層系統 (Layered System)
- 依需求載入程式 (Code-On-Demand)

RESTful API

- 符合REST設計風格的Web API稱為RESTful API
- 特色
 - 直觀精簡的資源位址：
 - URI，比如：`http://example.com/resources`。
 - 傳輸的資源：
 - Web服務接受與返回的網際網路媒體類型
 - JSON，XML，YAML等。
 - 對資源的操作：
 - 支援Web服務在該資源上所的一系列請求方法
 - POST，GET，PUT或DELETE

網路API風格

■ 傳統

- ❑ 取得所有使用者 GET /getAllUsers
- ❑ 取得單筆使用者 GET /getUser?id=1
- ❑ 新增使用者資料 POST /createUser
- ❑ 更新使用者資料 POST /updateUser/
- ❑ 刪除使用者資料 POST /deleteUser/ (GET /deleteUser?id=1)

■ REST

- ❑ 取得所有使用者 GET /rest/users
- ❑ 取得單筆使用者 GET /rest/user/1
- ❑ 新增使用者資料 POST /rest/user
- ❑ 更新使用者資料 PUT /rest/user/1
- ❑ 刪除使用者資料 DELETE /rest/user/1

RESTful API Request Method應用

| 資源 | 一組資源的URI https://www.gjun.com.tw/resources | 單個資源的URI https://www.gjun.com.tw/resources/007 |
|--------|---|---|
| GET | 列出URI，以及該資源組中每個資源資訊 | 取得指定007的資源的資訊 格式為網路媒體類型（XML、JSON等） |
| POST | 在本組資源中建立或者新增一個新的資源 | 將指定007的資源當做一個資源組，在其下建立新元素，使其隸屬於目前資源 |
| PUT | 使用給定的一組資源置換目前整組資源 | 置換或者建立指定的007資源 |
| DELETE | 刪除整組資源。 | 刪除指定的元素 |

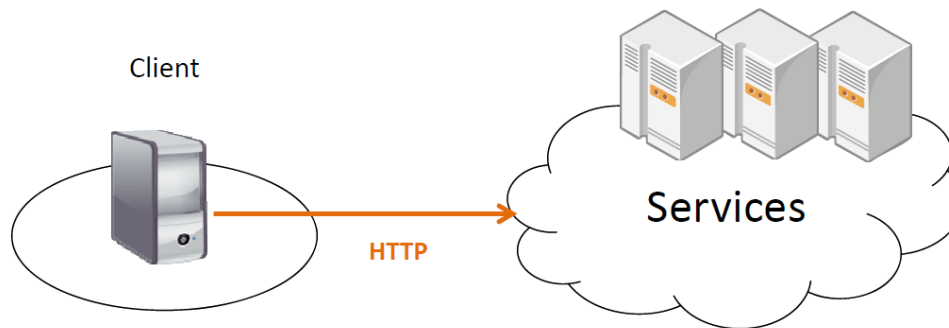
Restful Web Service

- Web Service

- 透過Internet (HTTP)存取的遠端業務邏輯

- Restful Web Service

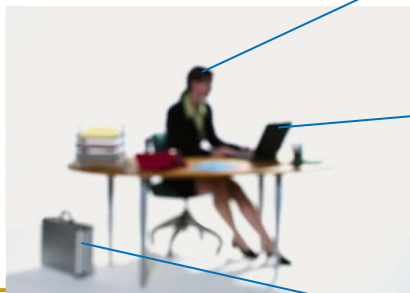
- 將HTTP的精神套用到Web Services上
 - HTTP作為應用程式平台，將服務視為物件(資源)



Restful Web Service

■ Restful Web Service

- REST 思維由data出發
- 萬物皆視被為具有URI的資源(Resources)
- 使用HTTP方法操作資源
 - GET/POST/PUT/DELETE
 - 訊息格式(XML、JSON)



<http://gjun.com.tw/dept1/employee/Helen>

<http://gjun.com.tw/dept1/laptop>

<http://gjun.com.tw/dept1/suitcase>

<http://gjun.com.tw/classrooms/101/sensors/1>



```
{  
  id: "1",  
  type: "temperature",  
  value: 28.9,  
  unit: celsius  
}
```

取得感測器資訊

GET <http://gjun.com.tw/classrooms/101/sensors/1/type>
GET <http://gjun.com.tw/classrooms/101/sensors/1/value>
GET <http://gjun.com.tw/classrooms/101/sensors/1/unit>

部署感測器

POST <http://gjun.com.tw/classrooms/101/sensors/2>
`id=2&type=humidity&value=0&unit=percent`

<http://gjun.com.tw/classrooms/101/aircon/2>



部署冷氣

POST <http://gjun.com.tw/classrooms/101/aircon/2>
`id=2&type=aircon&powerOn=true&fanSpeed=5`

取得冷氣開啟狀態

GET <http://gjun.com.tw/classrooms/101/aircon/2/powerOn>

關閉冷氣

PUT <http://gjun.com.tw/classrooms/101/aircon/2>
`powerOn=false`

調整風量

PUT <http://gjun.com.tw/classrooms/101/aircon/2>
`fanSpeed=3`

移除冷氣

DELETE <http://gjun.com.tw/classrooms/101/aircon/2>

```
{  
  id: "2",  
  type: "aircon",  
  powerOn: true,  
  fanSpeed: 5  
}
```

JAX-RS API

- Java API for RESTful Web Services (JAX-RS)
 - Java EE 支援 REST Web Service 開發的API
 - 使用Contexts and Dependency Injection(CDI) 技術
 - Annotation 注入環境及相依性資訊
 - 簡化客戶端和服務端的開發和部署工作

REST Application

注入Resource 配置服務

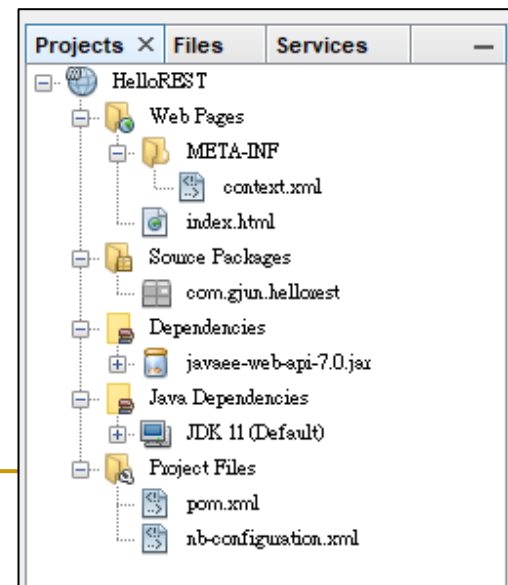
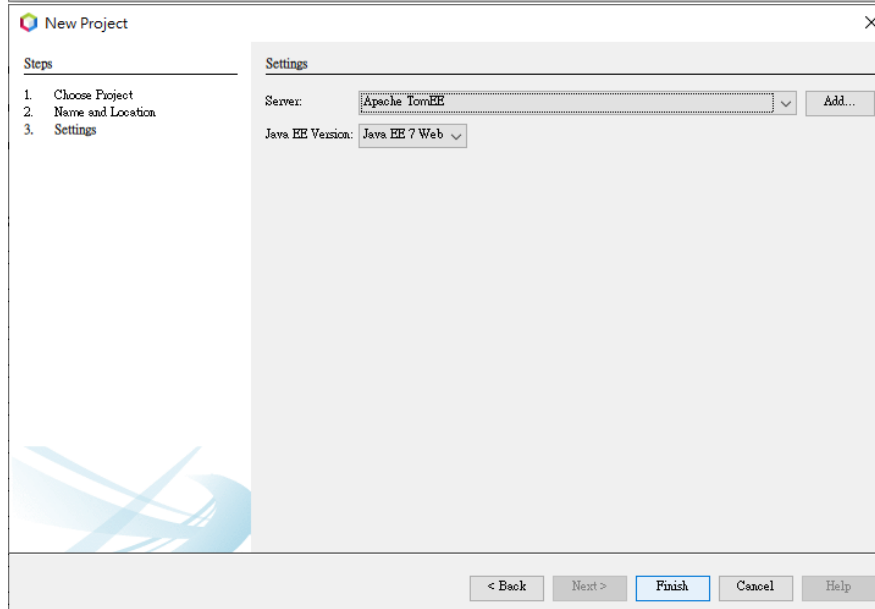
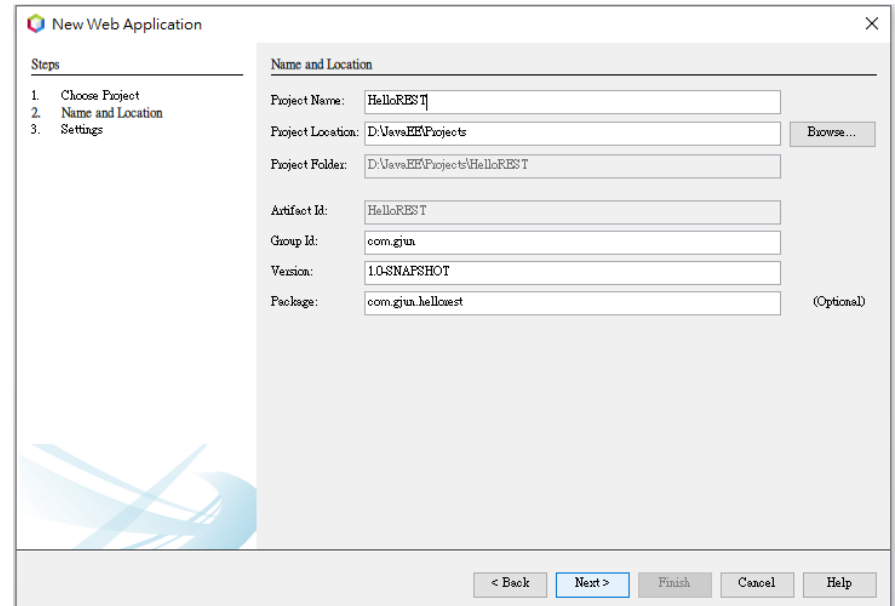
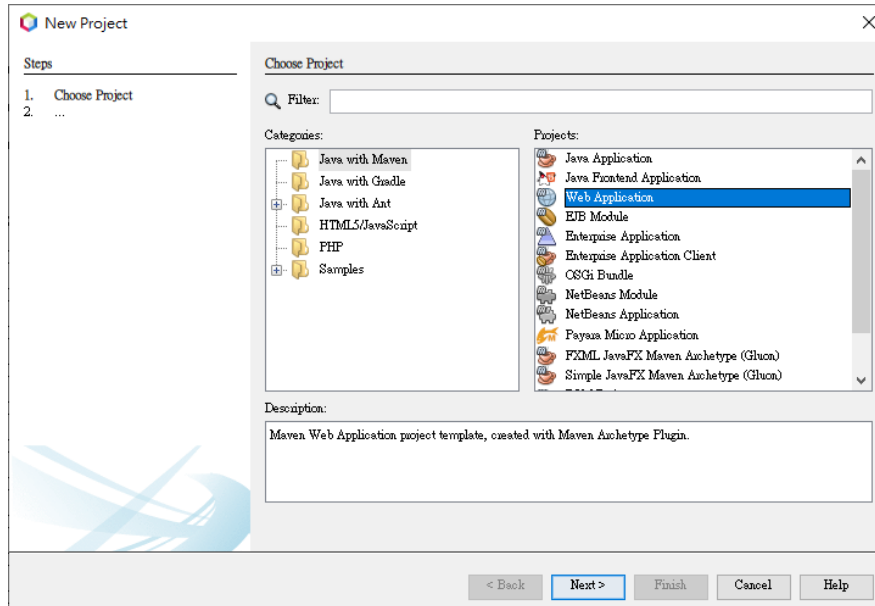
REST Resource

提供服務對外的端點與服務功能

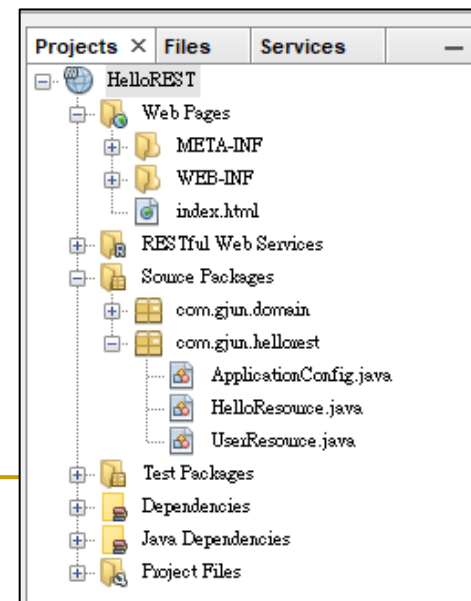
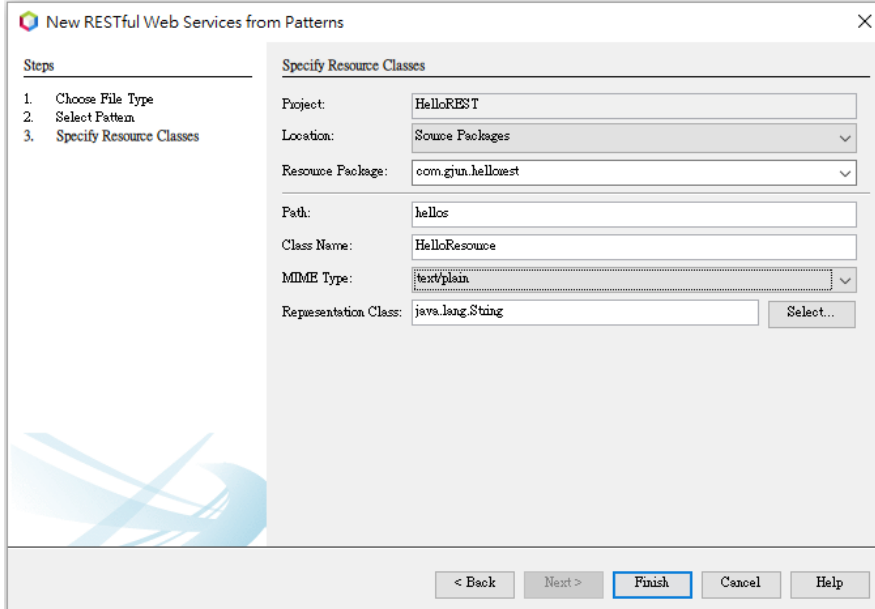
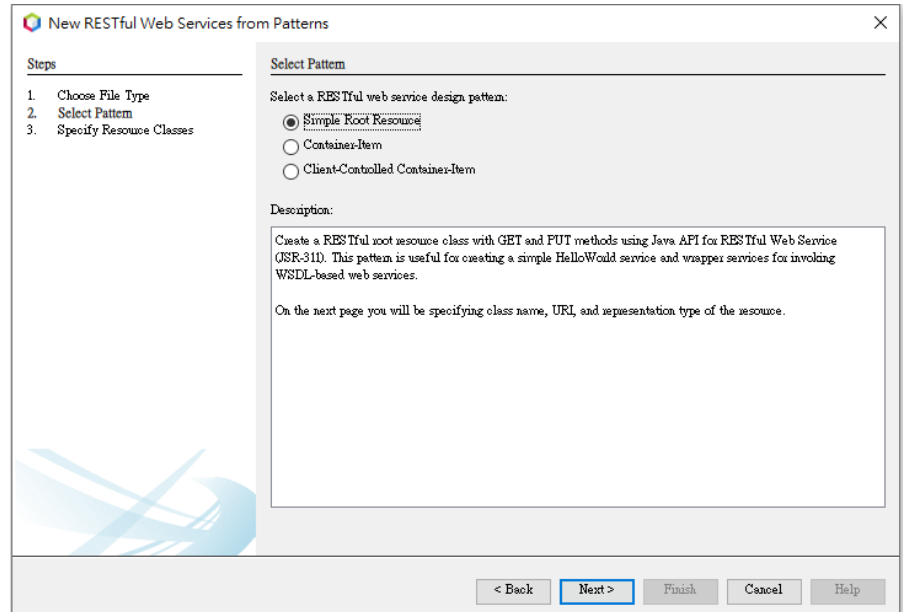
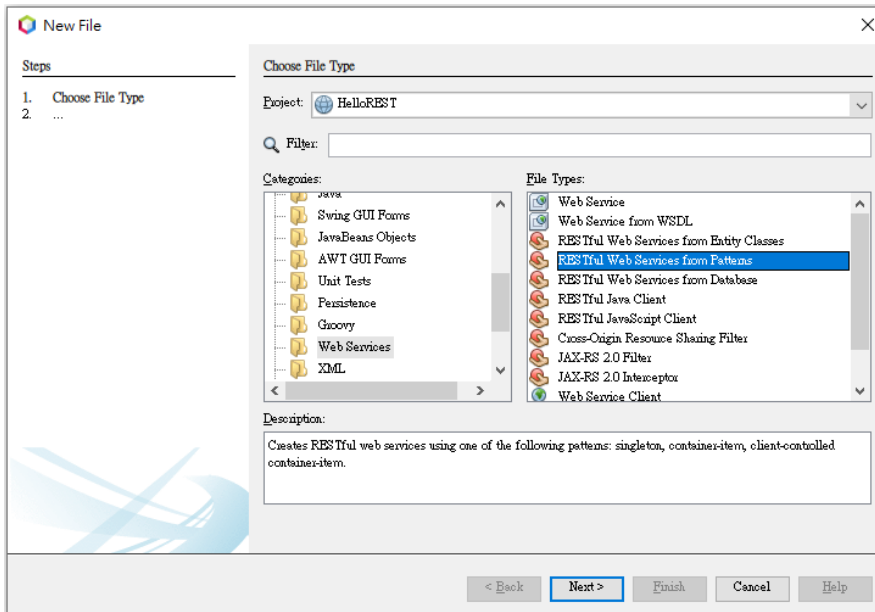
Annoation

Resource類別與方法標註
設定服務特徵

建立 Maven 網路應用程式



建立 RESTful Web Service

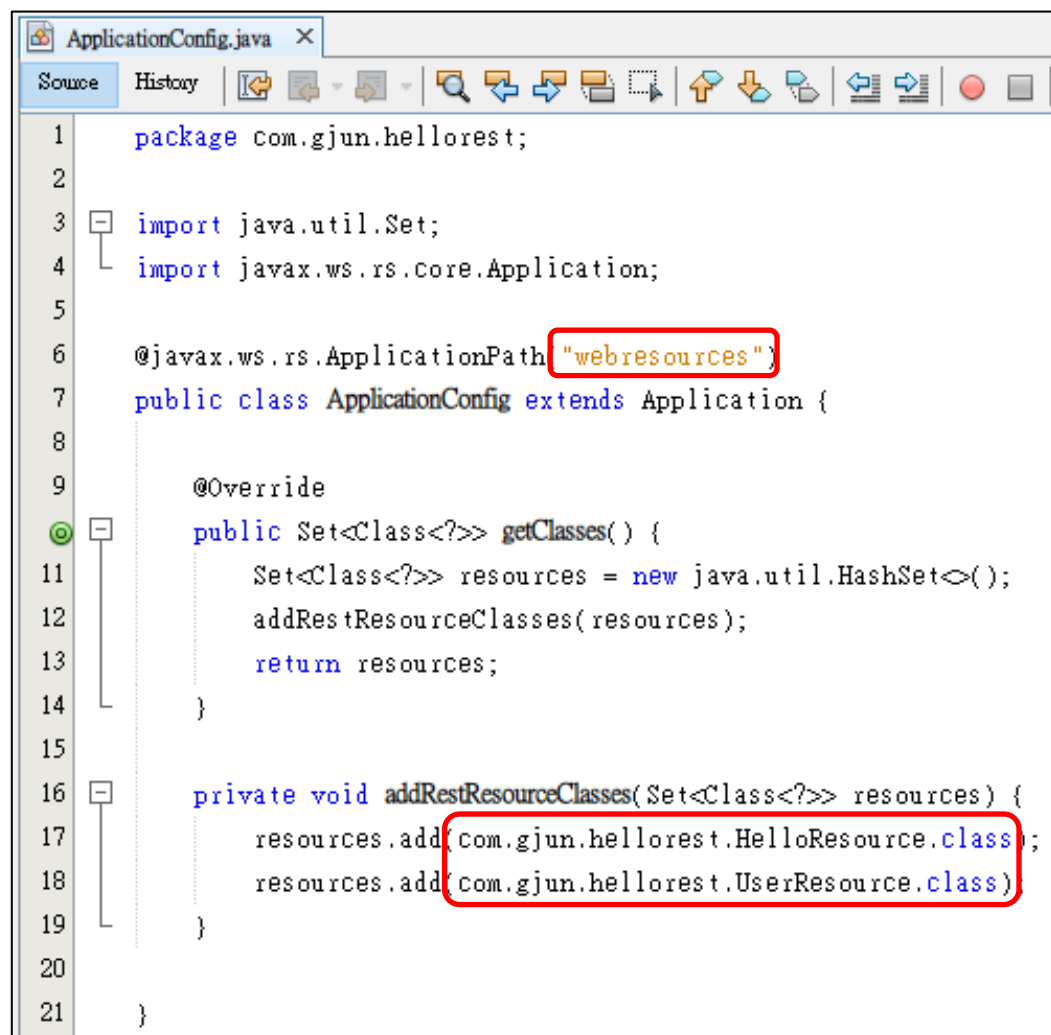


REST Application

■ REST Application

- 繼承 `javax.ws.rs.core.Application`
- 代表 REST 服務應用系統實體
 - 包含 JAX-RS 資源集合 `Resources`，型態為 `Set<Class<?>>`
- `ApplicationPath` 標註 Annotation
 - 設定網站中 REST API 進入的起始位址 (URI)
- `Resources` 資源集合
 - 加入 JAX-RS 資源類別

REST Application 類別



```
1 package com.gjun.hellorest;
2
3 import java.util.Set;
4 import javax.ws.rs.core.Application;
5
6 @javax.ws.rs.ApplicationPath("webresources")
7 public class ApplicationConfig extends Application {
8
9     @Override
10     public Set<Class<?>> getClasses() {
11         Set<Class<?>> resources = new java.util.HashSet<>();
12         addRestResourceClasses(resources);
13         return resources;
14     }
15
16     private void addRestResourceClasses(Set<Class<?>> resources) {
17         resources.add(com.gjun.hellorest.HelloResource.class);
18         resources.add(com.gjun.hellorest.UserResource.class);
19     }
20
21 }
```

REST Service Resource

- 採簡單的類別規劃
 - 繼承 Root Class(Object)
 - 服務方法使用標註注入
- 標註元素
 - @Path()
 - 設定類別或觸發服務方法的URI 路徑
 - @GET / @POST / @PUT / @DELETE
 - 設定觸發服務的 HTTP 方法
 - @Produces(Content-Type)
 - 回應的資料型態 Media Type

REST Service Resource

```
HelloResource.java x
Source History
1 package com.gjun.hellorest;
2
3 import javax.ws.rs.Produces;
4 import javax.ws.rs.GET;
5 import javax.ws.rs.Path;
6 import javax.ws.rs.PathParam;
7 import javax.ws.rs.QueryParam;
8 import javax.ws.rs.core.MediaType;
9
10 // http://localhost:8080/HelloREST/webresources/hellos
11 @Path("hellos")
12 public class HelloResource {
13
14     @GET
15     @Produces(MediaType.TEXT_HTML)
16     public String hello() {
17         return "<h3>Hello REST Web Service<h3>";
18     }
19
20     // webresources/hellos/hello?name=Sean
21     @Path("hello")
22     @GET
23     @Produces(MediaType.TEXT_HTML)
24     public String helloParam(@QueryParam("name")String who) {...3 lines }
25
26     // webresources/hellos/hi/{msg}/
27     @Path("hi/{msg}/")
28     @GET
29     @Produces(MediaType.TEXT_HTML)
30     public String helloPath(@PathParam("msg")String message) {...3 lines }
31
32 }
35
```



Resource 取得資訊

■ QueryString 請求參數

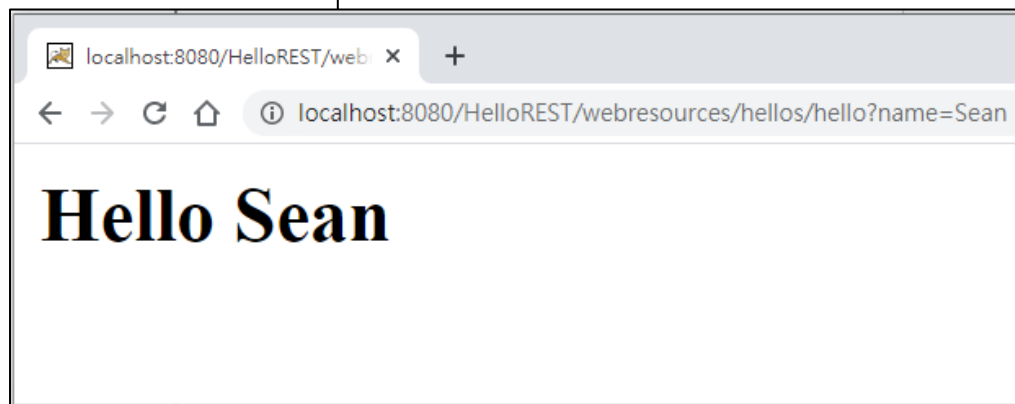
- `http://host/context/xyz/abc?param=value&...`
- Resource 方法中取得 QueryString
 - `@QueryParam("param")` 標註方法傳入參數

■ URI Path 路徑參數

- `http://host/context/xyz/abc/{path}/`
- Resource 方法中取得 URI Path 路徑參數
 - `@PathParam("path")` 標註方法傳入參數

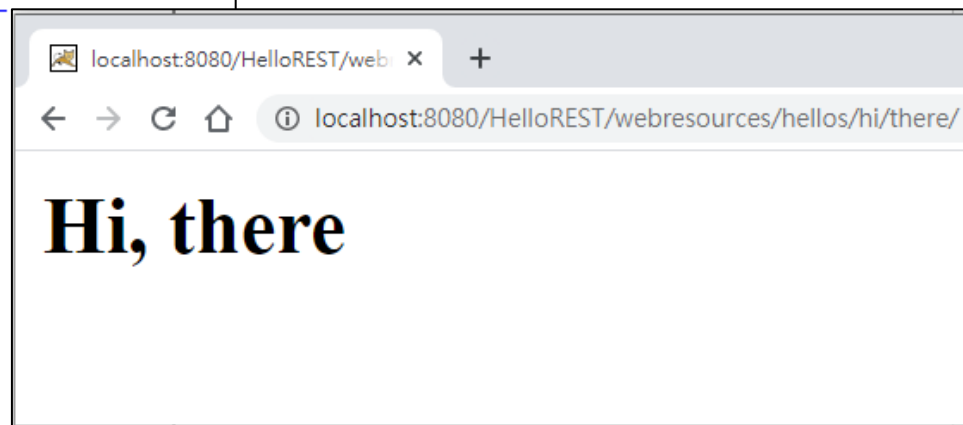
Resource 取得資訊

```
HelloResource.java x
Source History
1 package com.gjun.hellorest;
2
3 import ...6 lines
9
10 // http://localhost:8080/HelloREST/webresources/hellos
11 @Path("hellos")
12 public class HelloResource {
13
14     @GET
15     @Produces(MediaType.TEXT_HTML)
16     public String hello() {...3 lines }
17
18     // webresources/hellos/hello?name=Sean
19     @Path("hello")
20     @GET
21     @Produces(MediaType.TEXT_HTML)
22     public String helloParam(@QueryParam("name")String who) {
23         return String.format("<h3>Hello %s<h3>", who);
24     }
25
26     // webresources/hellos/hi/{msg}/
27     @Path("hi/{msg}/")
28     @GET
29     @Produces(MediaType.TEXT_HTML)
30     public String helloPath(@PathParam("msg")String message) {...3 lines }
31
32 }
33
34
35
36 }
```



Resource 取得資訊

```
HelloResource.java x
Source History
1 package com.gjun.hellorest;
2
3 import ...6 lines
9
10 // http://localhost:8080/HelloREST/webresources/hellos
11 @Path("hellos")
12 public class HelloResource {
13
14     @GET
15     @Produces(MediaType.TEXT_HTML)
16     public String hello() {...3 lines }
19
20     // webresources/hellos/hello?name=Sean
21     @Path("hello")
22     @GET
23     @Produces(MediaType.TEXT_HTML)
24     public String helloParam(@QueryParam("name")String who) {...3 lines }
27
28     // webresources/hellos/hi/{msg}/
29     @Path("hi/{msg}/")
30     @GET
31     @Produces(MediaType.TEXT_HTML)
32     public String helloPath(@PathParam("msg")String message) {
33         return String.format("<h3>Hi, %s<h3>", message);
34     }
35
36 }
```



Resource 內容回應

- Resource 內容回應
 - Resource 方法傳回JavaBean物件
 - Java EE 容器將其轉換為 JSON 字串置於回應中
- 實作步驟
 - JavaBean 類別
 - Resource 類別
 - @Produces("application/json")
 - 取得所有使用者 GET /rest/users
 - 取得單筆使用者 GET /rest/user/1

JSON文件格式

■ JSON (JavaScript Object Notation)

- 道格拉斯·克羅克福特構想和設計
- 輕量級的資料交換語言
- 易於讓人閱讀的文字為基礎
- 用來傳輸由屬性值或者序列性的值組成的資料物件。
- JavaScript的一個子集
- 獨立於語言的文字格式
 - 採用了類似於C語言家族的一些習慣。
- MIME 類型是 `application/json`，副檔名是 `.json`。

JSON 格式

■ JSON格式

□ 物件（object）

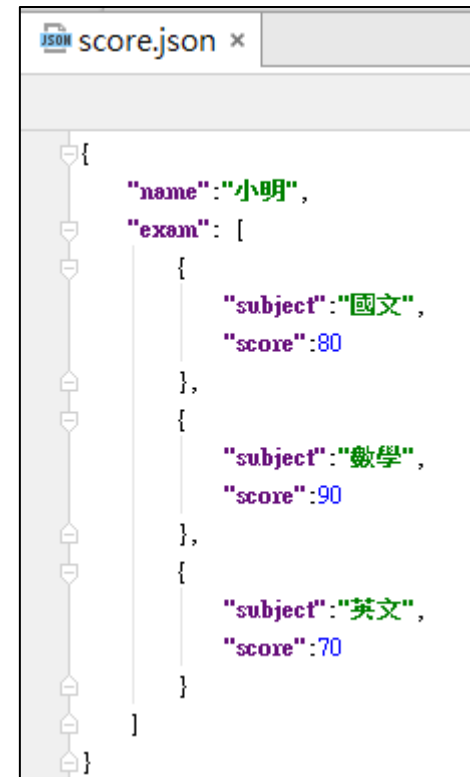
- 以大括號{ }標記
 - 可包含多組鍵值對以「，」分隔
 - 鍵和值之間使用「：」隔開
 - 鍵為字串，值則可為字串、數值、布林、物件、陣列、null
- ```
{ name1 : value1 , name2 : value2 }
```

### □ 陣列

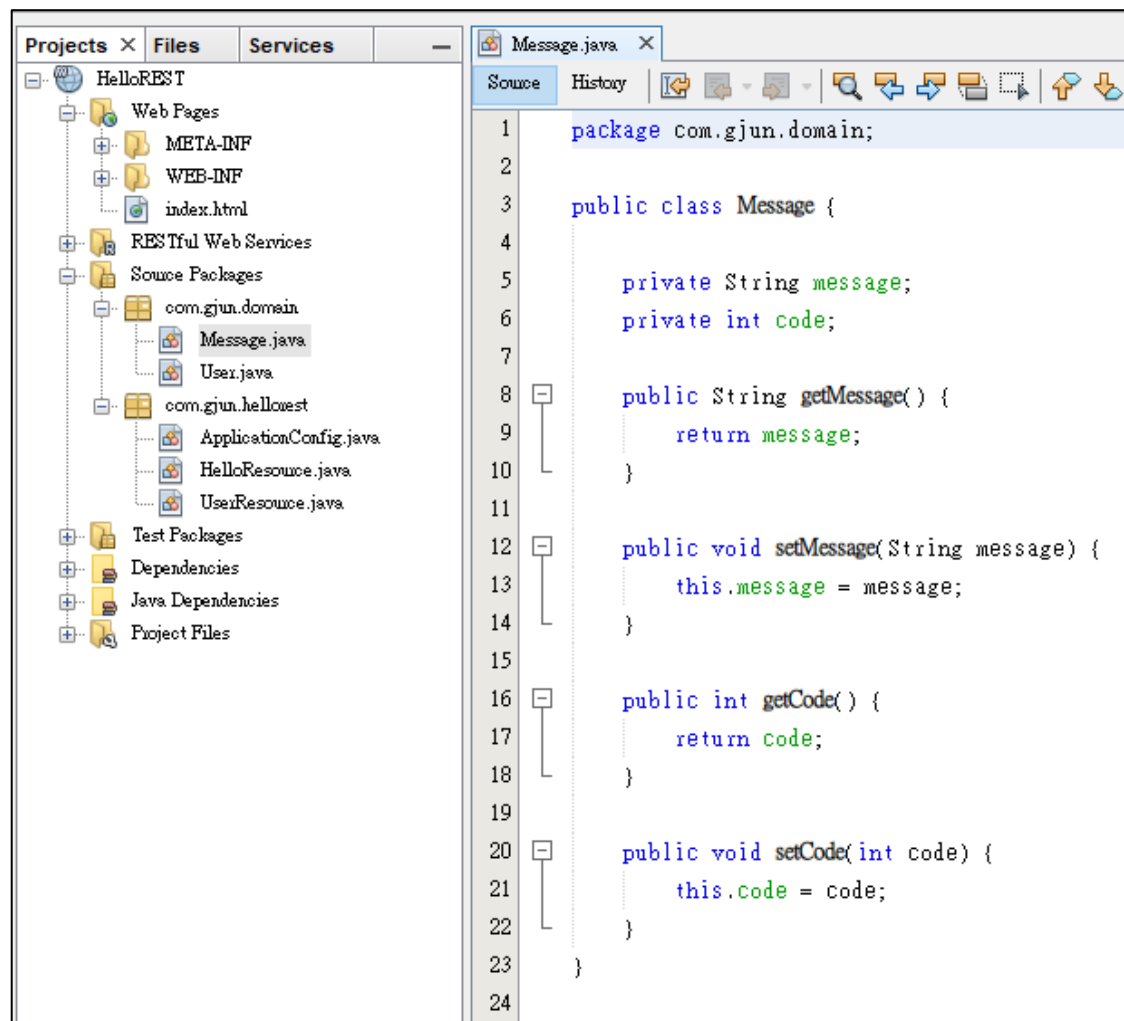
- 以中括號[ ]標記
  - 多個值以「，」分隔
  - 值可為字串、數值、布林、物件
- ```
[ value1 , value2 ]
```

JSON 格式

```
{
  "firstName": "John",
  "lastName": "Smith",
  "sex": "male",
  "age": 25,
  "address":
  {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumber":
  [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```



Java Bean 類別



Java Bean 類別

```
User.java
Source History
1 package com.gjun.domain;
2
3 import java.io.Serializable;
4
5 public class User implements Serializable {
6     private Integer id;
7     private String name;
8     private Integer age;
9
10    public User() {
11    }
12
13    public User(Integer id, String name, Integer age) {
14        this.id = id;
15        this.name = name;
16        this.age = age;
17    }
```

```
19 public Integer getId() {
20     return id;
21 }
22
23 public void setId(Integer id) {
24     this.id = id;
25 }
26
27 public String getName() {
28     return name;
29 }
30
31 public void setName(String name) {
32     this.name = name;
33 }
34
35 public Integer getAge() {
36     return age;
37 }
38
39 public void setAge(Integer age) {
40     this.age = age;
41 }
42
43 @Override
44 public String toString() {
45     return "User{" + "id=" + id + ", name=" + name + ", age=" + age + '}';
46 }
47 }
```

Resource 類別

```
UserResource.java x
Source History
1 package com.gjun.hellorest;
2
3 import ...14 lines
17
18 @Path("user")
19 public class UserResource {
20     private static Map<Integer, User> users = new TreeMap<>();
21     static{
22         users.put(1, new User(1, "Sean", 36));
23         users.put(2, new User(2, "David", 28));
24     }
25
26     // 查詢所有 User
27     @Path("users")
28     @GET
29     @Produces(MediaType.APPLICATION_JSON)
30     public Collection<User> getUsers(){...3 lines }
31
32     // 查詢指定id User
33     @Path("user/{id}")
34     @GET
35     @Produces(MediaType.APPLICATION_JSON)
36     public User getUser(@PathParam("id")Integer id){...3 lines }
```

```
42 // 新增 User
43 @Path("user")
44 @POST
45 @Consumes(MediaType.APPLICATION_JSON)
46 public Message addUser(User user){...13 lines }
47
48 // 更新 User
49 @Path("user/{id}")
50 @PUT
51 @Consumes(MediaType.APPLICATION_JSON)
52 public Message updateUser(@PathParam("id")Integer id, User user){...12 lines }
53
54 // 刪除 User
55 @Path("user/{id}")
56 @DELETE
57 @Consumes(MediaType.APPLICATION_JSON)
58 public Message deleteUser(@PathParam("id")Integer id){...12 lines }
59
60 }
```

查詢所有User

```
25
26 // 查詢所有 User
27 @Path("users")
28 @GET
29 @Produces(MediaType.APPLICATION_JSON)
30 public Collection<User> getUsers(){
31     return users.values();
32 }
33
```

localhost:8080/HelloREST/webresources/user/users

```
[{"age":36,"id":1,"name":"Sean"},
{"age":28,"id":2,"name":"David"}]
```

Postman

File Edit View Help

+ New Import Runner My Workspace Invite

GET http://localhost:8080/HelloREST/webresources/user/users

Untitled Request

GET http://localhost:8080/HelloREST/webresources/user/users

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

| KEY | VALUE | DESCRIPTION |
|-----|-------|-------------|
| Key | Value | Description |

Body Cookies Headers (6) Test Results

Status: 200 OK Time: 48 ms Size: 252 B

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "age": 36,
4     "id": 1,
5     "name": "Sean"
6   },
7   {
8     "age": 28,
9     "id": 2,
10    "name": "David"
11  }
12 ]
```

Find and Replace Console Bootcamp Build Browse

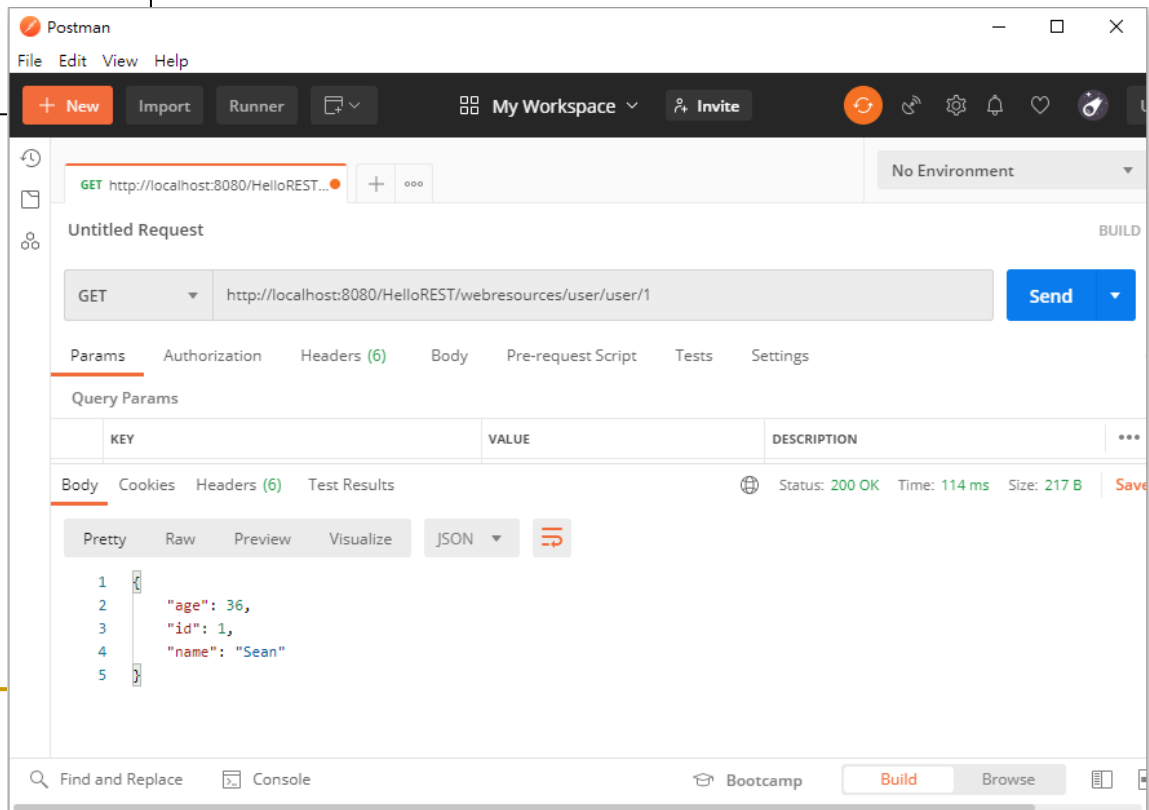
查詢指定User

```
34 // 查詢指定id User
35 @Path("user/{id}")
36 @GET
37 @Produces(MediaType.APPLICATION_JSON)
38 public User getUser(@PathParam("id") Integer id){
39     return users.get(id);
40 }
41
```

localhost:8080/HelloREST/web x +

localhost:8080/HelloREST/webresources/user/user/1

```
{"age":36,"id":1,"name":"Sean"}
```

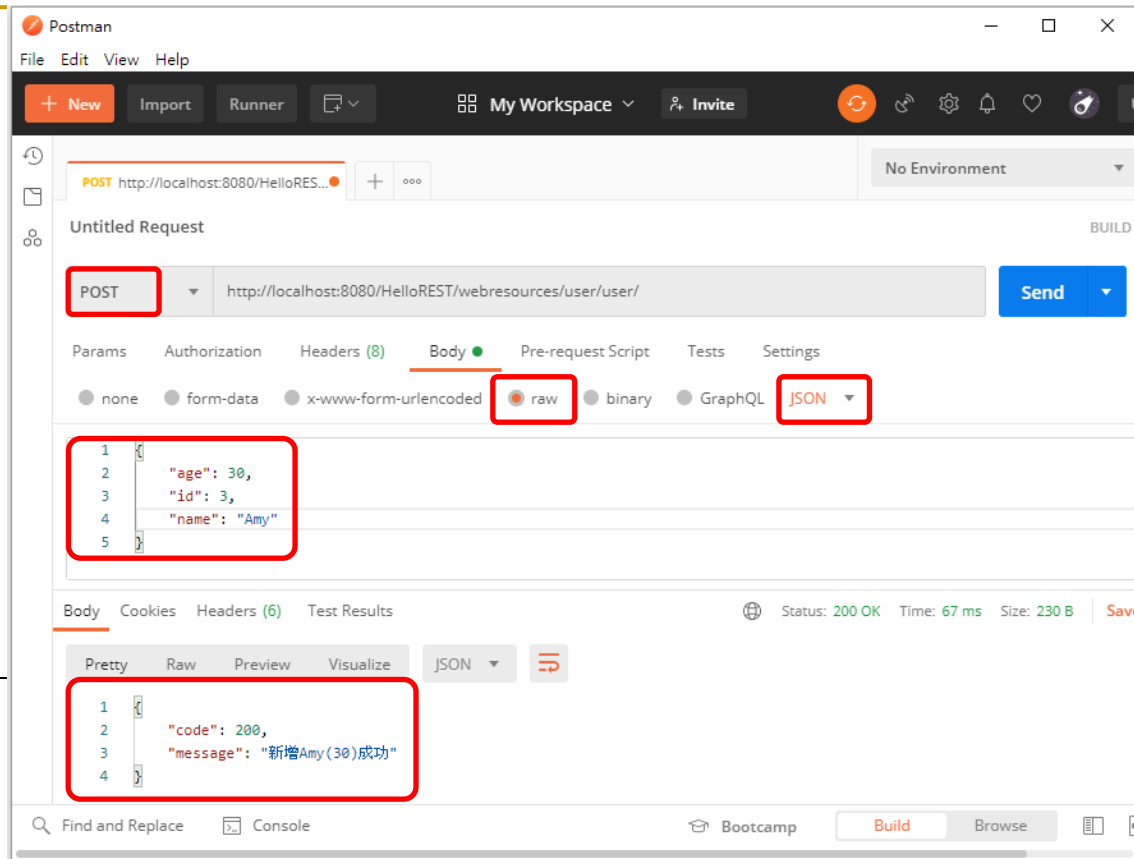


Resource 接收 JSON 資訊

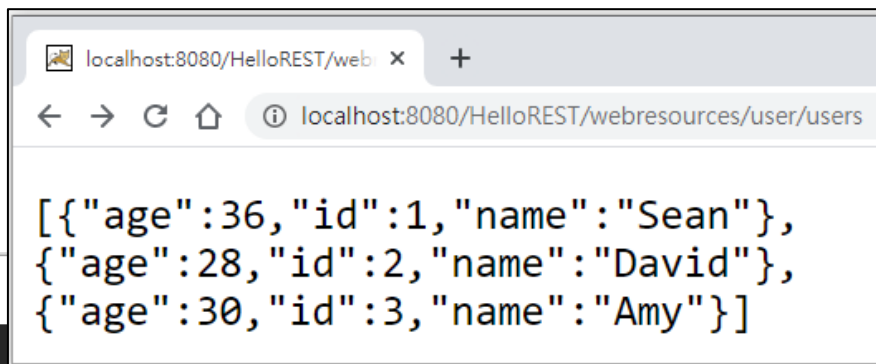
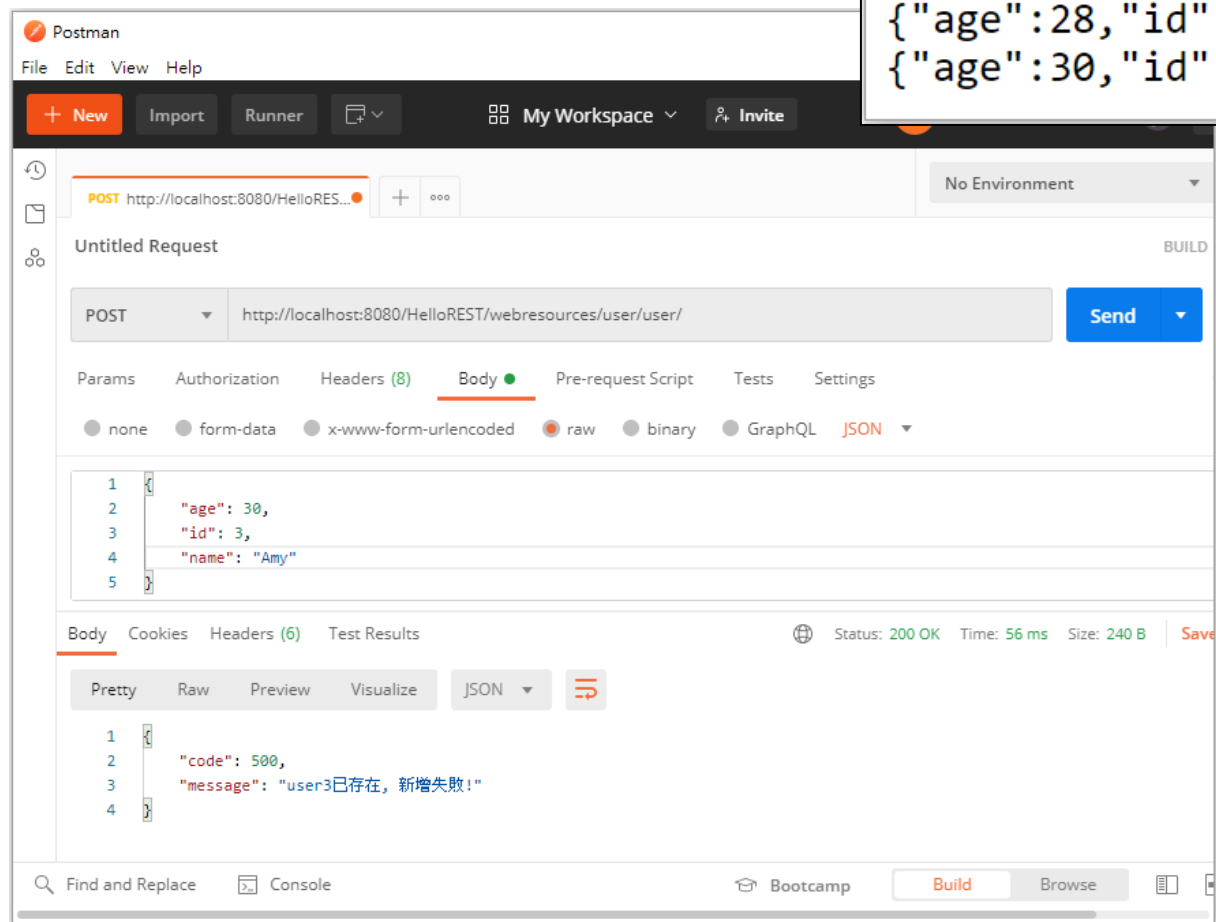
- Resource 接收 JSON 資訊
 - Resource 方法接收 JSON 字串
 - Java EE 容器將其轉換為 JavaBean 物件
- 實作步驟
 - Resource 類別
 - @Consumes("application/json")
 - 新增使用者資料 POST /rest/user
 - 更新使用者資料 PUT /rest/user/1
 - 刪除使用者資料 DELETE /rest/user/1
 - 使用PostMan進行測試
 - Body : Raw Data
 - Request Header Content-Type : application/json

新增 User

```
42 // 新增 User
43 @Path("user")
44 @POST
45 @Consumes(MediaType.APPLICATION_JSON)
46 public Message addUser(User user){
47     Message msg = new Message();
48     Integer id = user.getId();
49     if(users.containsKey(id)){
50         msg.setMessage(String.format("user%d已存在, 新增失败!", id));
51         msg.setCode(500);
52     } else {
53         users.put(id, user);
54         msg.setMessage(String.format("新增%s(%d)成功", user.getName(), user.getAge()));
55         msg.setCode(200);
56     }
57     return msg;
58 }
```

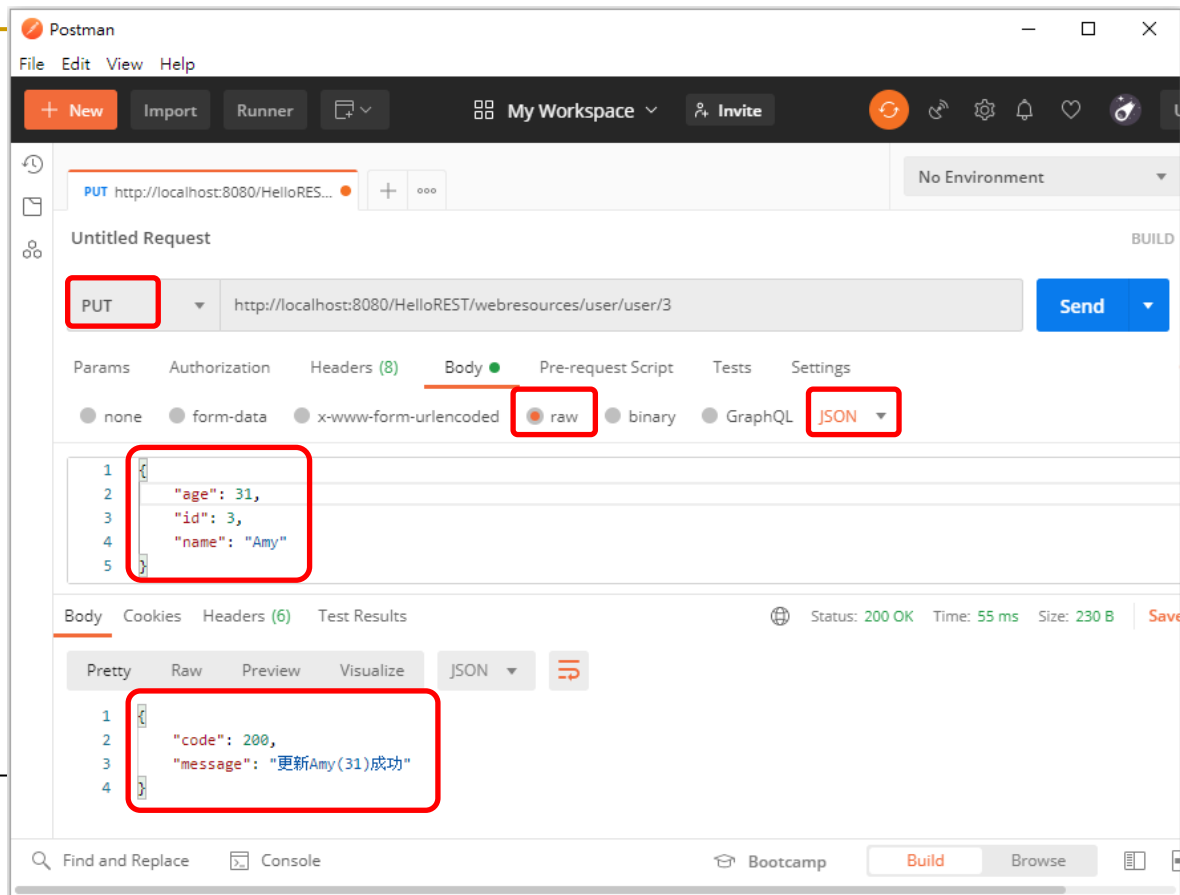


新增 User



更新 User

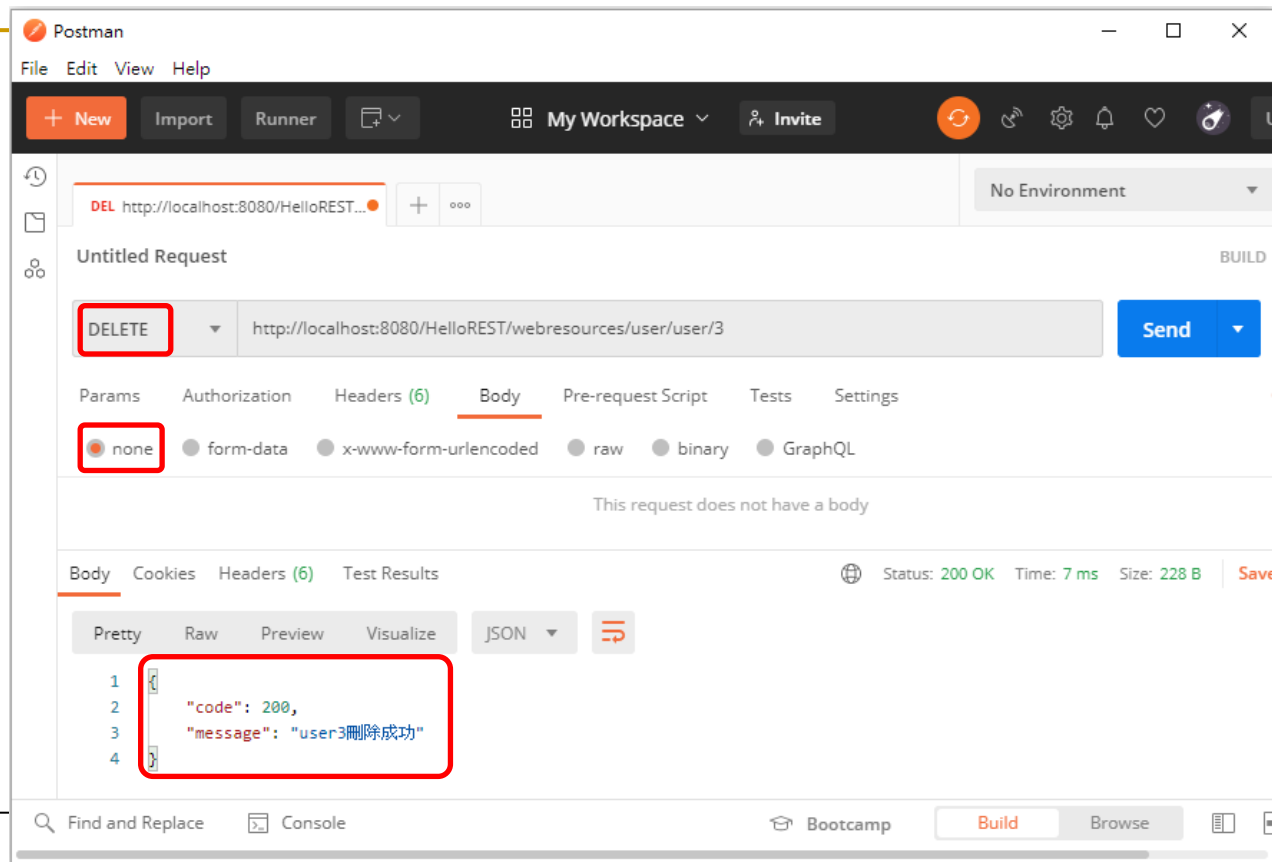
```
60 // 更新 User
61 @Path("user/{id}")
62 @PUT
63 @Consumes(MediaType.APPLICATION_JSON)
64 public Message updateUser(@PathParam("id") Integer id, User user){
65     Message msg = new Message();
66     if(users.containsKey(id) && user.getId().equals(id)){
67         users.put(user.getId(), user);
68         msg.setMessage(String.format("更新%s(%d)成功", user.getName(), user.getAge()));
69         msg.setCode(200);
70     } else {
71         msg.setMessage(String.format("user%d更新失败!", user.getId()));
72         msg.setCode(500);
73     }
74     return msg;
75 }
```



The browser displays the JSON response from the API at `localhost:8080/HelloREST/webresources/user/user/3`:

```
{"age":31,"id":3,"name":"Amy"}
```

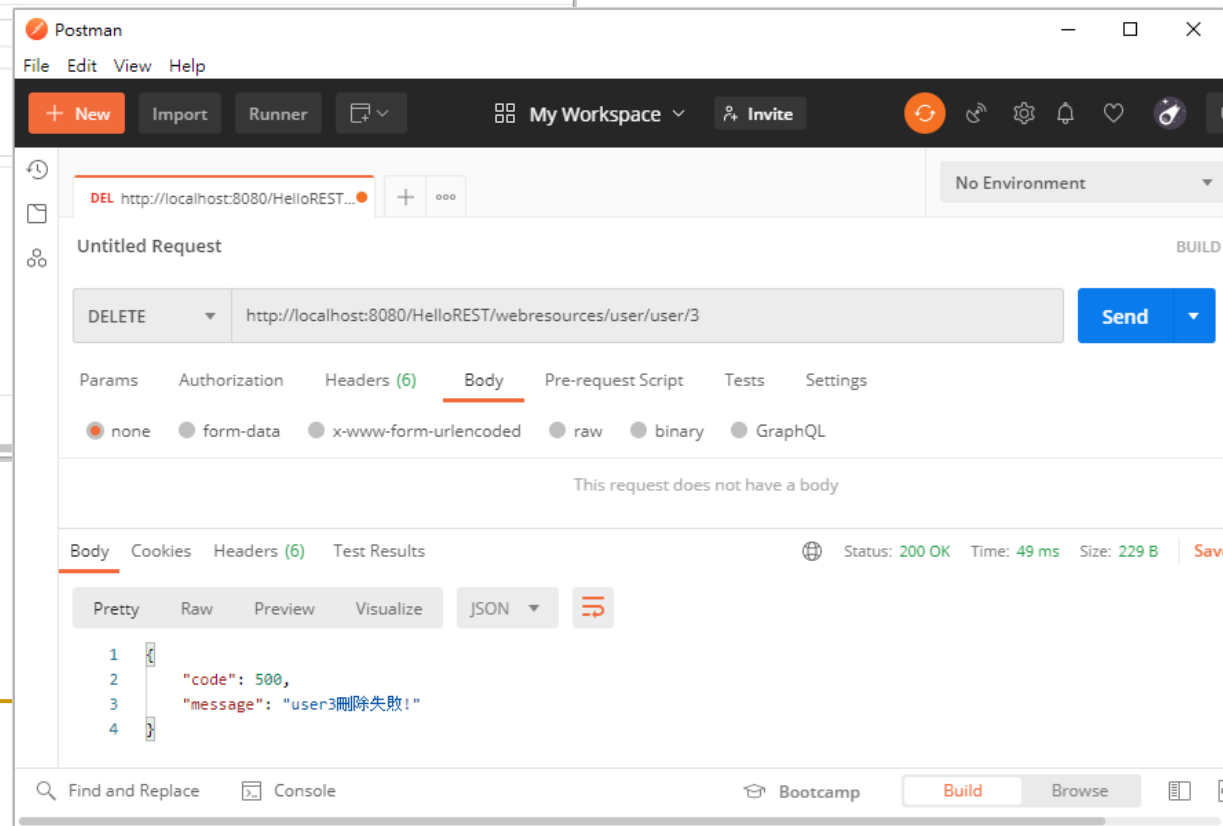
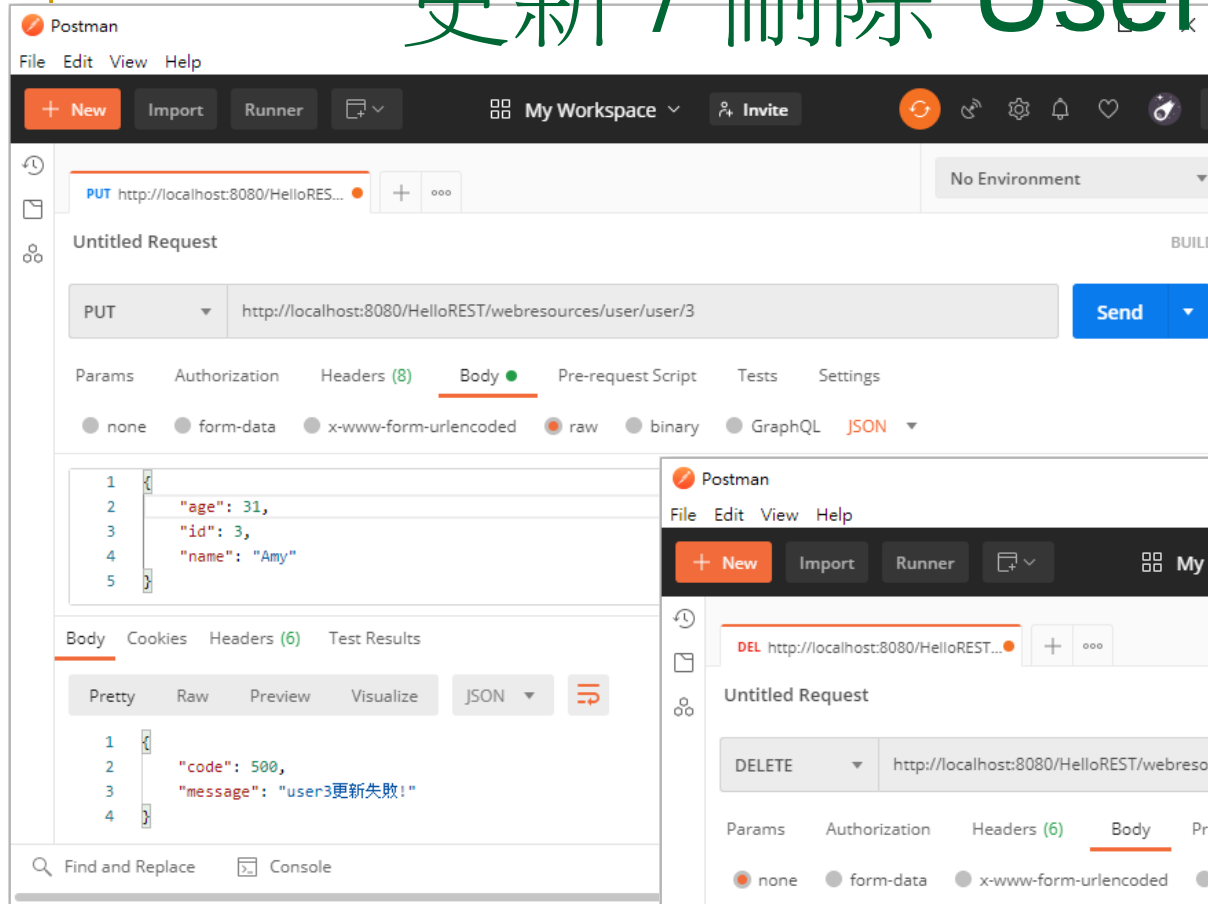
删除 User



```
77 // 删除 User
78 @Path("user/{id}")
79 @DELETE
80 public Message deleteUser(@PathParam("id") Integer id){
81     Message msg = new Message();
82     if(users.containsKey(id)){
83         users.remove(id);
84         msg.setMessage(String.format("user%d删除成功", id));
85         msg.setCode(200);
86     } else {
87         msg.setMessage(String.format("user%d删除失败!", id));
88         msg.setCode(500);
89     }
90     return msg;
91 }
```

The screenshot shows a web browser displaying the JSON response from the API: `[{"age": 36, "id": 1, "name": "Sean"}, {"age": 28, "id": 2, "name": "David"}]`.

更新 / 刪除 User 失敗



Lab

■ ContactInfo 網路應用專案

- 使用 Rest Web Service API 新增/修改/刪除客戶資料
- 不包含Filter機制 (與第九章練習完成結果相同)
- 修改網路伺服器為 TomEE

Lab

■ 修改 Customer.java

- 新增無參數建構子
- 刪除靜態測試資料及靜態方法
 - 搬移至CustomerResource.java
- 新增下列 **setter** 方法
 - `public void setOfficeAddress(Address address)`
 - `public void setBillingAddress(Address address)`
 - `public void setDeliveryAddress(Address address)`
- 刪除取得所有地址的方法
 - `public Address[] getAddresses() {`

Lab

- 修改 `customerList.jsp`
 - EL 不使用 `Addresses[0]` 顯示客戶聯絡資料
 - 改用 `OfficeAddress`, `BillingAddress`, `DeliveryAddress` 屬性
- 修改 `customerView.jsp`
 - EL 不使用 `Addresses[2]` 顯示客戶聯絡資料
 - 改用 `customer.deliveryAddress` 屬性

Lab

- Restful Web Service Resource
 - 新增 Restful Web Service Resource
 - File Type
 - Restful Web Service From Patterns
 - Pattern
 - Design Pattern: Single Root Resource
 - Resource Class
 - Package: service
 - Path: customer
 - ClassName: customerResource
 - MIME Type: application/json

Lab

- ❑ 檢視 ApplicationConfig.java

- ApplicationPath: webresources
- addRestResourceClasses() 包含
 - ❑ service.CustomerResource.class

- ❑ CustomerResource.java

- 將Customer 的靜態測試資料及靜態方法搬移過來
 - ❑ HashMap<Integer, Customer> customers
 - ❑ static {...}
 - ❑ public static Customer getCustomer(int id)
 - ❑ public static Collection<Customer> getCustomers()

Lab

- Response addCustomer(@PathParam("id")Integer id, Customer customer) 方法
 - @POST / @Path("{id}") / @Consumes(MediaType.APPLICATION_JSON)
 - 客戶編號已存在 customers 中, 產生錯誤回應, 代碼400
 - 客戶編號不存在, customers 中新增客戶 put(id, customer), 產生成功回應, 傳回 customers
- Response updateUser(@PathParam("id")Integer id, Customer customer) 方法
 - @PUT / @Path("{id}") / @Consumes(MediaType.APPLICATION_JSON)
 - 客戶編號不存在 customers 集合中, 產生錯誤回應, 代碼400
 - 客戶編號已存在, customers 中取代客戶 put(id, customer), 產生成功回應, 傳回 customers

Lab

- Response deleteUser(@PathParam("id")Integer id) 方法
 - @DELETE / @Path("{id}")
 - 客戶編號不存在 customers 集合中, 產生錯誤回應, 代碼400
 - 客戶編號不存在, customers 中 remove(id) 移除客戶), 產生成功回應, 傳回 customers
- 修改 web.Controller.java
 - CustomerResource.getCustomers() 取得所有客戶
 - CustomerResource.getCustomer(custid) 取得指定客戶
- 測試、執行