RESTful Web Services

M.I. Capel

ETS Ingenierías Informática y
Telecomunicación
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Granada
Email: manuelcapel@ugr.es
http://lsi.ugr.es/mcapel/

November, 8th 2017
Máster Universitario en Ingeniería Informática

Representational State Transfer (REST)

Historic outline

Initially proposed by Roy Thomas Fielding in his PhD dissertation book: *Architectural Styles and the Design of Network-based Software Architectures*(2000)

Fundamental characteristics

- REST-based notation to be used is mainly based on the 1996 `Http 1.0` standard
- Client applications communicate with serversa by using *Http verbs*: `GET, POST, DELETE, PUT, PATCH`
- The server can access *resources* that are identified by `URI` (*Uniform Resource Identifier*)
- Resources can have several textual representations: `XML, JSON, HTML, ...`

Http methods

Recommended return values for primary HTTP methods which are combined with URI resources

No	**Verb**	**CRUD**	**Entire Collection**	**Specific Item**
1	POST	Create	201 (Created)	404 (Not Found), 409 (Conflict) if resource exists.
2	GET	Read	200 (OK)	200 (OK)
3	PUT	Update/Replace	404 (Not Found)	200 (OK) or 204 (No Content). 404 (Not Found *)
4	PATCH	Update/Modify	404 (Not Found)	200 (OK) or 204 (No Content). 404 (Not Found *)
5	DELETE	Delete	404 (Not Found)	200 (OK). 404 (Not Found *)

(*):404 (Not Found), if ID not found or invalid.

Explanation

1. 'Location' header with link to /customers/id containing new ID.
2. List of customers. Use pagination, sorting and filtering to navigate big lists.
3. Single customer. 404 (Not Found), if ID not found or invalid, unless you want to update/replace every resource in the entire collection.
4. if ID not found or invalid, unless you want to modify the collection itself.
5. if ID not found or invalid, unless you want to delete the whole collection—not often desirable.

Definition of the base URL of a resource

Idea fundamental

A SW implemented with RESTful technology must define the *base direction* of each one of the services that offers to its clients

Example:

```
1 com.sun.jersey.api.client.config.ClientConfig
2       config = new DefaultClientConfig();
3 com.sun.jersey.api.client.WebResource
4       service =
5 com.sun.jersey.api.client.Client.create(config).resource(
      getBaseURI());
```

Data exchange between the client and the *service*

`accept` protocol

```
service.accept(MediaType.TEXT_XML).get(String.class));
service.accept(MediaType.APPLICATION_XML).get(String.class));
service.accept(MediaType.APPLICATION_JSON).get(String.class));
```

We have to program with the prior pattern each one of the read(GET()), write(PUT()), update(PATCH(),POST()) operations..., which are going to be supported by the service

JAXB

Fundamental idea

- This is about a specific standard (*Java Architecture for XML Binding*) of use for obtaining a correspondence between 'regular' data objects (*POJO*) and their representation in XML
- The associated framework allow us to read/write from/in Java objects and in/from XML documents

JAXB annotations

`@XmlRootElement(namespace = "space_of_names")`	Root element of an "XML tree"
`@XmlType(propOrder = "field1",... )`	writting order for class fields into the XML
`@XmlElement(name = "newName")`	The XML element that is used instead[a]

[a] It only needs to be used if it is different from the name assigned by the JavaBeans framework

DAO

Definition

DAO or “data access object” is an object that provides an abstract interface to a DB or any other mechanism for persistence of entities of software applications

- DAO provide us with some operations on specific data without disclosing, however, the supporting DB low-level details to the user applications
- It also provide us a mapping between operation calls performed in an application to the *persistence layer* of a Web service

DAO Todo

```java
import java.util.HashMap;
import java.util.Map;
//import the data domain model
public enum TodoDao {
  INSTANCE;//for singleton.
  private Map<String, Todo> contentsProvider = new HashMap<
      String, Todo>();
  private TodoDao() {
    Todo todo = new Todo("1", "Learn REST");
    todo.setDescription("Read http://lsi.ugr.es/dsbcs/Documentos
        /Practica/practica3.html");
    contentsProvider.put("1", todo);
    todo = new Todo("2", "Learn something about DSBCS");
    todo.setDescription("Read all the material placed at http://
        https://prado1718.ugr.es/moodle/course/view.php?id
        =63658");
    contentsProvider.put("2", todo);  }
  public Map<String, Todo> getModel(){
    return contentsProvider;  }
}
```

Data domain

```java
@XmlRootElement
public class Todo{
  private String id;
  private String summary;
  private String description;

  public Todo(){
  }
  public Todo (String id, String summary){
    this.id = id;
    this.summary = summary;
  }
  public String getId() {
    return id;
  }
  public void setId(String id) {
    this.id = id;
  }
  ...
}
```

Resource

```
1  import javax.ws.rs.Consumes;
2  import javax.ws.rs.DELETE;
3  import javax.ws.rs.GET;
4  import javax.ws.rs.PUT;
5  import javax.ws.rs.Produces;
6  import javax.ws.rs.core.Context;
7  import javax.ws.rs.core.MediaType;
8  import javax.ws.rs.core.Request;
9  import javax.ws.rs.core.Response;
10 import javax.ws.rs.core.UriInfo;
11 import javax.xml.bind.JAXBElement;
```

Resource II

```
1  import javax.servlet.http.HttpServletResponse;
2  @Path("/todos")//Mapping of resource into URL: todos
3  public class TodosRecurso {
4  //It allows inserting contextual objects in the class,
5  //for instance, ServletContext, Request, Response, UriInfo
6            @Context
7            UriInfo uriInfo;
8            @Context
9            Request request;
10 //Returns the list of all contained elements
11           @GET
12           @Produces(MediaType.TEXT_XML)
13           public List<Todo> getTodosBrowser() {
14             List<Todo> todos = new ArrayList<Todo>();
15             todos.addAll(TodoDao.INSTANCE.getModel().values());
16             return todos;
17           }
```

Recurso III

```
1  ...// To obtain the total number of elements stored in the
       service's data base
2            @GET
3            @Path("cont")
4            @Produces(MediaType.TEXT_PLAIN)
5            public String getCount() {
6              int cont = TodoDao.INSTANCE.getModel().size();
7              return String.valueOf(cont);
8            }
9            @Path("{todo}")
10           public TodoRecurso getTodo(@PathParam("todo") String
                 id) {
11             return new TodoRecurso(urilnfo, request, id);
12           }
13 }
```

Service deployment description

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:
       schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://
       xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID"
        version="3.1">
3    <display-name>Contents server with REST technology</display-
         name>
4    <welcome-file-list>
5      <welcome-file>index.html</welcome-file>
6      <welcome-file>index.htm</welcome-file>
7      <welcome-file>index.jsp</welcome-file>
8      <welcome-file>default.html</welcome-file>
9      <welcome-file>default.htm</welcome-file>
10     <welcome-file>default.jsp</welcome-file>
11   </welcome-file-list>
```

Service deployment description-II

```
1   <servlet>
2     <servlet-name>Jersey-implemented REST service</servlet-name>
3     <servlet-class>org.glassfish.jersey.servlet.ServletContainer
          </servlet-class>
4      <!-- It records resources held in mio.jersey.primero -->
5     <init-param>
6         <param-name>jersey.config.server.provider.packages</
              param-name>
7         <param-value>mio.jersey.primero</param-value>
8      </init-param>
9     <load-on-startup>1</load-on-startup>
10   </servlet>
11   <servlet-mapping>
12     <servlet-name>Jersey-implemented REST service</servlet-name>
13     <url-pattern>/rest/*</url-pattern>
14   </servlet-mapping>
15 </web-app>
```

Test class for the implemented Web service

```
1  public class Tester {
2    public static void main(String[] args) {
3  // TODO Auto-generated method stub
4          ClientConfig config = new DefaultClientConfig();
5          Client client = Client.create(config);
6  WebResource service = client.resource(getBaseURI());
7  //create a third "todo" object, in addition to the other 2
8          Todo todo = new Todo("3","This is the summary of the
               third record");
9  ClientResponse response = service.path("rest").path("todos").
       path(todo.getId()).accept(MediaType.APPLICATION_XML).put(
       ClientResponse.class, todo);
10         System.out.print("Returned code: ");
11 //The code must be: 201 == created
12         System.out.println(response.getStatus());
13 //Shows the contents of the resource "Todos" as XML text
14 System.out.println("To show as XML Plain text");          System.
       out.println(service.path("rest").path("todos").accept(
       MediaType.TEXT_XML).get(String.class));
```

Test class for the implemented Web service-II

```
1  // Create a fourth "Todo" resource by a Web form
2  System.out.println("Form creation");
3  Form form = new Form(); form.add("id", "4");
4  form.add("summary", "Demonstration of the form client-library");
5  response = service.path("rest").path("todos").type(MediaType.
       APPLICATION_FORM_URLENCODED).post(ClientResponse.class,
       form);
6  System.out.println("Response with the form" + response.getEntity
       (String.class));
7  // An element with id = 4 must have been created
8  System.out.println("Resource contents, after sending the element
       with id=4");
9  System.out.println(service.path("rest").path("todos").
10 accept(MediaType.APPLICATION_XML).get(String.class));}
11 private static URI getBaseURI() {
12   return UriBuilder.fromUri("http://localhost:8080/mio.jersey.p3
         ").build();  }
13 }//the class ends
```

Program for deleting an object from the resource

```
1 // We are going to delete the "objects" with id=1 from the
      resource
2 service.path("rest").path("todos/1").delete();
3 // We show the contents of the resource "Todos", the element
      with id=1
4 // must have been deleted already
5 System.out.println("The element with id = 1 in the resource has
      been deleted");
6 System.out.println(service.path("rest").path("todos")
7 .accept(MediaType.APPLICATION_XML).get(String.class));
```

CRUD service deployed in a Tomcat server