Práctica 3-Servicios Web RESTful

M.I. Capel

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

DSBCS

Máster en Ingeniería Informática

15 de noviembre de 2016

Índice

Índice

Representational State Transfer (REST)

Reseña histórica

Inicialmente propuesto por Roy Thomas Fielding su tesis doctoral: *Architectural Styles and the Design of Network-based Software Architectures*(2000)

Características fundamentales

- La notación que utiliza está basada en el estándar `Http 1.0` de 1996
- Las aplicaciones cliente se comunicarán con los servidores utilizando los *verbos-Http*: `GET, POST, DELETE, PUT, PATCH`
- El servidor accede a *recursos* identificados mediante `URI`
- Los recursos pueden tener varias representaciones textuales: `XML, JSON, HTML, ...`

Métodos Http

Valores de retorno recomendados para los métodos HTTP primarios combinados con los URI de los recursos

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

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

Definición de URL de base del recurso

Idea fundamental

Un SW implementado con tecnología RESTful ha de definir la dirección de base de cada uno de los servicios que ofrece a sus clientes

Ejemplo:

```
com.sun.jersey.api.client.config.ClientConfig
    config = new DefaultClientConfig();
com.sun.jersey.api.client.WebResource
    servicio =
com.sun.jersey.api.client.Client.create(config).resource
    (getBaseURI());
```

Intercambio de datos entre el cliente y el *servicio*

protocolo `accept`

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

Se ha de programar con la plantilla anterior para cada una de las operaciones de lectura (GET()), escritura (PUT()), actualización (PATCH(),POST()),... que vayan aser soportadas por el servicio

JAXB

Idea fundamental

- Se trata de un estándar para obtener una correspondencia (*Java Architecture for XML Binding*) entre los objetos de Java (*POJO*) y su representación con XML
- El marco de trabajo asociado permite leer/escribir de/en objetos Java y en/desde documentos XML

Anotaciones de JAXB

`@XmlRootElement(namespace = "espacio_nombres")`	Elemento raíz de un "árbol XML"
`@XmlType(propOrder = "campo1",... )`	Orden escritura campos en el XML
`@XmlElement(name = "nuevoNombre")`	El elemento XML que será usado[a]

[a] Sólo necesita ser utilizado si es diferente del nombre que le asigna el marco de trabajo JavaBeans

DAO

Definition

Un DAO o “objeto de acceso a datos” es un objeto que proporciona una interfaz abstracta a algún tipo de base de datos u otro mecanismo de persistencia.

- El DAO nos proporciona algunas operaciones sobre datos específicos sin que resulten visibles para las aplicaciones los detalles de la base de edtos que actúa como soporte de dichas operaciones.
- También se proporciona una correspondencia entre las llamadas a operaciones desde una aplicación a la *capa de persistencia* del servicio Web.

DAO Todo

```
import java.util.HashMap;
import java.util.Map;
//importar el modelo del dominio de datos
public enum TodoDao {
  INSTANCE;//para implementar el patron singleton.
  private Map<String, Todo> proveedorContenidos = new
      HashMap<String, Todo>();
  private TodoDao() {
    Todo todo = new Todo("1", "Aprender␣REST");
    todo.setDescripcion("Leer␣http://lsi.ugr.es/dsbcs/
        Documentos/Practica/practica3.html");
    proveedorContenidos.put("1", todo);
    todo = new Todo("2", "Aprender␣algo␣sobre␣DSBCS");
    todo.setDescripcion("Leer␣todo␣el␣material␣de␣http
        ://lsi.ugr.es/dsbcs");
    proveedorContenidos.put("2", todo);  }
  public Map<String, Todo> getModelo(){
    return proveedorContenidos;  }
}
```

Dominio de datos

```
@XmlRootElement
public class Todo{
  private String id;
  private String resumen;
  private String descripcion;

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

Recurso

```
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.PUT;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Request;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriInfo;
import javax.xml.bind.JAXBElement;
```

Recurso II

```
import javax.servlet.http.HttpServletResponse;
@Path("/todos")//Correspond. del recurso al URL: todos
public class TodosRecurso {
// Permite insertar objetos contextuales en la clase,
//por ejemplo, ServletContext, Request, Response,
    UriInfo
          @Context
          UriInfo uriInfo;
          @Context
          Request request;
// Devolvera la lista de todos los elementos contenidos
          @GET
          @Produces(MediaType.TEXT_XML)
          public List<Todo> getTodosBrowser() {
            List<Todo> todos = new ArrayList<Todo>();
            todos.addAll(TodoDao.INSTANCE.getModel().
                values());
            return todos;
          }
```

Recurso III

```
...// Para obtener el numero total de elementos en el
    servicio
          @GET
          @Path("cont")
          @Produces(MediaType.TEXT_PLAIN)
          public String getCount() {
            int cont = TodoDao.INSTANCE.getModel().size
                ();
            return String.valueOf(cont);
          }
          @Path("{todo}")
          public TodoRecurso getTodo(@PathParam("todo")
              String id) {
            return new TodoRecurso(uriInfo, request, id)
                ;
          }
}
```

Descripción del despliegue del servicio

```
<?xml version="1.0" encoding="UTF-8"?>
<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">
  <display-name>Servidor de Contenidos REST</display-
      name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
```

Descripción del despliegue del servicio-II

```
  <servlet>
    <servlet-name>Servicio REST de Jersey</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.
        ServletContainer</servlet-class>
     <!-- Registra recursos que estan ubicados dentro de
          mio.jersey.primero-->
    <init-param>
        <param-name>jersey.config.server.provider.
            packages</param-name>
        <param-value>mio.jersey.primero</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Servicio REST de Jersey</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Clase de prueba del servicio Web

```
public class Probador {
  public static void main(String[] args) {
// TODO Auto-generated method stub
        ClientConfig config = new DefaultClientConfig();
        Client cliente = Client.create(config);
WebResource servicio = cliente.resource(getBaseURI());
//crearse un tercer "objeto" todo, aparte de los otros 2
        Todo todo = new Todo("3","Este es el resumen del
            tercer registro");
ClientResponse respuesta = servicio.path("rest").path("
    todos").path(todo.getId()).accept(MediaType.
    APPLICATION_XML).put(ClientResponse.class, todo);
        System.out.print("Codigo devuelto: ");
//El codigo devuelto deberia ser: 201 == created
        System.out.println(respuesta.getStatus());
//Mostrar el contenido del recurso Todos como texto XML
System.out.println("Mostrar como Texto XML Plano");
    System.out.println(servicio.path("rest").path("todos"
    ).accept(MediaType.TEXT_XML).get(String.class));
```

Clase de prueba del servicio Web-II

```
// Crear un cuarto recurso Todo con un formulario Web
System.out.println("Creacion de 1 formulario");
Form form = new Form(); form.add("id", "4");
form.add("resumen", "Demostracion de la biblioteca-
    cliente para formularios");
respuesta = servicio.path("rest").path("todos").type(
    MediaType.APPLICATION_FORM_URLENCODED).post(
    ClientResponse.class, form);
System.out.println("Respuesta con el formulario" +
    respuesta.getEntity(String.class));
// Se ha debido crear el elemento con id = 4
System.out.println("Contenidos del recurso, despues de
    enviar el elemento id=4");
System.out.println(servicio.path("rest").path("todos").
accept(MediaType.APPLICATION_XML).get(String.class));}
private static URI getBaseURI() {
  return UriBuilder.fromUri("http://localhost:8080/mio.
      jersey.p3").build();  }
}//fin de la clase
```

Programa para eliminar un objeto del recurso

```
// Ahora vamos a eliminar el "objeto" con id=1 del
    recurso
servicio.path("rest").path("todos/1").delete();
// Mostramos el contenido del recurso Todos, el elemento
     con id=1
// deberia haber sido eliminado
System.out.println("El elemento con id = 1 del recurso
    se ha eliminado");
System.out.println(servicio.path("rest").path("todos")
.accept(MediaType.APPLICATION_XML).get(String.class));
```

Servicio CRUD desplegado en un servidor Tomcat