HW #3 – Web Service Programming

RESTful Web Service

Hooman Peiro Sajjad (shps@kth.se)
KTH – ICT School
VT 2015

RESTFul WebService

This tutorial is based on the following resources:

1-Lecture notes on RESTful Web services :http://www.ccil.org/~cowan

2- http://www.ibm.com/developerworks/web/library/wa-aj-tomcat/index.html

What is REST?

- REpresentational State Transfer
- Resources:
  - Resources are identified by uniform resource identifiers (URIs)
  - Resources are manipulated through their Representations
  - Messages are self-descriptive and stateless
  - Multiple representations are accepted or sent
  - Hypertext is the engine of application state

Example of Resource Representations:

- A Web page , a file , a record in database,…

More about Resources

- Resources are just concepts
- URIs tell a client that there's a concept somewhere
- Clients can then request a specific representation of the concept from the representations the server makes available
- State of resource is maintained by being transferred from clients to servers and back to clients

Communication style in REST

- REST can support any media type, but XML the most popular transport for structured information

- Unlike SOAP and XML-RPC, REST does not really require a new message format

- HTTP 1.1 was designed to conform to REST

This means that you can use GET / POST / DELETE / PUT/… in REST. This means that for example:

You can do a GET on a URIs that you do POST to!

RESTful Web Service programming

Define Resource Class

1- Resource Class : to be defined as plain old java object style.

2- Add Annotations (according to your requirement) to the Class and Methods to make it RESTful resource:
Example:

@Path: resource base URI .

Resource Identifier=

HostName + Context Root + url-pattern + resource base URI

@GET- to get (retrieve) resource contents

@PUT- to update resource contents

@DELETE- to remove resource contents

.....

Annotations

@Context: Use this annotation to inject contextual information objects like (Request, Response,etc) to your resource class

@PathParam(“contact”)- to inject parameters into the path, in this case “contact”

@Produces- It specifies the response type (Plain Text, XML, MIME Types, JAXB Elements,..)

@Consumer- It indicate the request type (Plain Text, XML, MIME Types, JAXB Elements,..)

And more.....

A Simple RESTful Service

```
@Path("/hello")
public class HelloResource {
@GET
@Produces(MediaType.TEXT_PLAIN)
public String sayHello() {
        return "Hello Jersey";
    }
}
```

Client Side Code

```
Client cln = Client.create();
WebResource r = cln.resource("http://localhost:8080/Jersey/rest/hello");

String xmlRes = r.accept(MediaType.TEXT_PLAIN).get(String.class);

System.out.println(xmlRes);
```

“Client Side Project is a Normal Java Project, just include the Jersey jar file(s). “

More Advanced Example

Idea:

1- Accessing collection of objects as Resource. In our case <ContactsResource> is collection of <ContactResource>.

2- To simplify the application, just assume that we keep the content of the objects in a HashMap, instead of a file, or database

3- Neither <ContactsResource> Nor <ContactResource> does not store the Real Content of information to be stored/retrieved. They are just kind of References to those data

ContactsResource Class

```
@Path("/contacts")
public class ContactsResource {

@Context
UriInfo uriInfo;
@Context
Request request;

//Reading All objects in the Collection
@GET
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public List<Contact> getContacts() {
      List<Contact>   contacts = new ArrayList<Contact>();
      contacts.addAll( ContactStore.getStore().values() );
      return contacts;
}

//Reading a Specific Contact {contact} (contact identifier) from Collection
@Path("{contact}")
public ContactResource getContact(@PathParam("contact") String contact) {
      return new ContactResource(uriInfo, request, contact);
}
```

ContactResource Class -(1)

```
public class ContactResource {
@Context
UriInfo uriInfo;
@Context
Request request;
String contact;

public ContactResource (UriInfo uriInfo, Request request, String contact) {
 this.uriInfo = uriInfo; this.request = request; this.contact = contact;
}

// Reading a Contract Content
@GET
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public Contact getContact() {
	Contact cont = ContactStore.getStore().get(contact);
	if(cont==null)
		throw new NotFoundException("No such Contact.");
	return cont;
}
```

ContactResource Class -(2)

```
@PUT
@Consumes(MediaType.APPLICATION_XML)
public Response putContact(JAXBElement<Contact>  jaxbContact)
 {
//read content of the object
Contact c = jaxbContact.getValue();
Response res;

// Build the responce
if(ContactStore.getStore().containsKey(c.getId())) {
            res = Response.noContent().build();
} else {
      res = Response.created(uriInfo.getAbsolutePath()).build();
}
// Update the object content
      ContactStore.getStore().put(c.getId(), c);

return res;
}
```

Contact Store

```
public class ContactStore {
private static Map<String,Contact> store;
private static ContactStore instance = null;

private ContactStore() {
store = new HashMap<String,Contact>();
initOneContact();
}
public static Map<String,Contact> getStore() {
if(instance==null)
            instance = new ContactStore();
return store;
}
private static void initOneContact() {
Address[] addrs = {
new Address("Shanghai", "Long Hua Street"),
new Address("Shanghai", "Dong Quan Street")
};
Contact cHuang = new Contact("huangyim", "Huang Yi Ming",
Arrays.asList(addrs));
store.put(cHuang.getId(), cHuang);
}
```

Contact

```
@XmlRootElement
public class Contact {
private String id;
private String name;
private List<Address> addresses;

public Contact() {}

public Contact(String id, String name, List<Address> addresses) {
	this.id = id;
	this.name = name;
	this.addresses = addresses;
}
public String getId() {
	return id;
}
public void setId(String id) {
	this.id = id;
}
.....
....
}
```

Client Side Code

```
// Get a Reference to the RESTFul Resource
Client c = Client.create();
WebResource r = c.resource("http://localhost:8080/Jersey/rest/contacts");

//Create JAXB Element
GenericType<JAXBElement<Contact>> generic = new
GenericType<JAXBElement<Contact>>() {};

//For example, we would like get the contract with id “huangyim”
String id = “huangyim”;

//GET the resource
JAXBElement<Contact> jaxbContact =
r.path(id).accept(MediaType.APPLICATION_XML).get(generic);

//Raed JAXB Element Content
Contact contact = jaxbContact.getValue();

System.out.println(contact.getId() + ": " + contact.getName());
```

Setting up the Environment -1

(Eclipse + Tomcat)

Use Netbeans 6.9, 7.x*

OR

1- Eclipse IDE for J2EE

2- JAVA 5 or above

3- Apache Tomcat 6.x

4- Jersey libraries (Jersey 1.0.3 archive)

https://jersey.java.net/

Jersey is Implementation of JAVA API for RESTful web service

5- Support libraries : activation.jar, sax-api.jar, wstx-asl.jar

* *(Version suggested are tested)*

Creating RESTful Service project (server side)

1- Create Dynamic Web Application. Specify Context for example: *Jersey*

2- Specify the Target run-time container to be Apache Tomcat (point it to the installation path of Apache Tomcat)

3- After creating the project, configure servlet dispatcher in web.xml file to redirect all REST requests to your Jersey container (Apache Tomcat)

4- Put the library files (*.jar) into ./WEB-INF/lib folder

Setting up the Environment -2

(Defining Jersey Servlet Dispatcher in web.xml)

```
<display-name>Jersey</display-name>
  ....
 <servlet>
  <servlet-name>Jersey REST Service</servlet-name>
  <servlet-class>
          com.sun.jersey.spi.container.servlet.ServletContainer
</servlet-class>
  <init-param>
   <param-name>com.sun.jersey.config.property.packages</param-name>
   <param-value>sample.hello.resources</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
 </servlet>

 <servlet-mapping>
  <servlet-name>Jersey REST Service</servlet-name>
  <url-pattern>/rest/*</url-pattern>
 </servlet-mapping>
```

When you are done with programming export it:

- Right Click on project and select Export from menu, and export entire project as *WAR* file and put it into *.../apache-tomcatxxx/web-apps/*

- *Start the Apache Tomcat*

(e.g sh startup.sh, startup.bat...)

More Links

Look at the IBM tutorial.
It includes both tutorial details and source code:
http://www.ibm.com/developerworks/web/library/wa-aj-tomcat/index.html

Tasks

1- Implement ALL web services, you developed in HW2, in RESTful web service.

2- Develop client side to test GET/POST/PUT/DELETE operations.

Try to use at least one case when you do these operations on the collection of resources.

Deliverable

1- Textual report document explaining what did you do.

2- Source Code + Instructions on How to Depoly and Run the services. Show your running system in the Homework Demonstration Session!

Send your deliverables by email to both of us:

shps@kth.se, misha@kth.se

Subject : PWS15-HW3

Don’t forget to put your fullname in the email !

Deadline: 16 Feb

GOOD LUCK!