HW #4 – Web Service Programming

RESTful Web Service

RESTFul WebService

This tutorial is based on the following link from IBM web site:
http://www.ibm.com/developerworks/web/library/wa-aj-tomcat/index.html

Setting up the Environment

1- Eclipse IDE for JEE

2- JAVA 5 or above

3- Apache Tomcat 6.x

4- Jersey libraries (Jersey 1.0.3 archive)

Jersey is Implementation of JAVA API for RESTful web service

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

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

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>
```

Defining Web Service Classes

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(“contract”)- to inject parameters into the path, in this case “contract”

@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";
        }
}
```

1- Right Click on project and select select Export from menu, and export entire project as *WAR* file and put it into *.../apache-tomcatxxx/web-apps/ and Start the Apache Tomcat (e.g sh startup.sh, startup.bat...)*

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 <ContractsResource> is collection of <ContractResorce>.

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 <ContractsResource> Nor <ContractResource> 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 Contract {contact} from Collection
@Path("{contact}")
public ContactResource getContact(@PathParam("contact") String
contact) {
       return new ContactResource(uriInfo, request, contact);
    }
```

ContractResource 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;
}
```

ContractResource 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);
}
```

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());
```

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- Choose ONE web service from those you developed in HW2 /HW3 and implement it Entirely in RESTful web service . Each service should provide access to a collection of resources (e.g. transcripts, company info, profiles, job info, etc)

2- Develop client side to test GET/PUT/DELETE operations on the the collection of resources provided by the aforementioned web service.

Deliverable

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

Send your deliverables by eamil to both of us:

shahabm@kth.se , nimad@kth.se

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

Deadline: 21 March

Presentation: To Be Decided

GOOD LUCK!