REST-based Web Services (II)

Introduction to Service Design and Engineering 2013/2014.

Lab session #6

University of Trento

Cristhian Parra: Lab Lectures for IntroSDE-2013/2014
Some slides from LabSDE-2012/2013

This is a DRAFT in progress

Outline

- Frameworks
- A simple REST Service example with Jersey
- A simple REST Services client

Frameworks (1)

- There is **no need of a specific client or server-side framework**.
- The only requirement is **supporting the HTTP protocol**
  - In Java: all you need are servlets and override *doGet()*, *doPost()*, *doPUT()* and *doDelete()*
  - Remember last 5 basic servlet example
- However, *just servlets* can be complicated in reality, so a framework is welcomed to reduce boilerplace-type coding

Frameworks (2)

- Standard specification in java: JAX-RS
- Some frameworks
  - Apache CXF
  - Jersey - the JAX-RS Reference Implementation from Sun.
  - RESTEasy - JBoss's JAX-RS project.
  - Restlet - probably the first REST framework, which existed prior to JAX-RS.
  - Play Framework - popular nowadays, it is an MVC framework with a heavy focus on RESTful design
  - Spring Framework - another very popular java application framework that you can use to build RESTful services

Before we start: Jersey Libraries

- Download the libraries we need from Jersey Download page (jars also available in the resources folder
- Unzip the downloaded zip file into your home directory. We will use this later through Eclipse WTP. Remove the zip file.
- **What's inside Jersey?**
  - A core server, implemented as a servlet dispatcher for REST requests
  - A core client that provides a library to communicate with the server

Example 1: Hello World with Jersey (1)

- Create the project:
  - New *Dynamic Web Project* (File -> New -> Other -> Web) and call it 'introsde.simple.rest.hello'
  - Go to the Eclipse *Java EE perspective*
  - Once the project is created, navigate to WebContent/WEB-INF/lib folder
  - While the lib folder is highlighted, do 'right mouse click' to open a menu, then choose Import -> File system -> Specify the jersey_core directory you saved the jersey library files to.
- **Note:**
  - Click and highlight the directory, but do not 'tick' the box next to the directory name. Tick the boxes next to all 8 jar files for importing.
  - When available, mark the option *Generate Web XML Descriptor*

Example 1: Hello World with Jersey (2)

- For this hello world example, we are going to create a single java class as a resource the "**helloworld**" Resource
- Create the following java class with a package name 'introsde.simple.rest.hello' (full source in Example1-SimpeRestful

```
package introsde.simple.rest.hello;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
@Path("/helloworld")
public class HelloWorld {
    @GET
    @Produces(MediaType.TEXT_HTML)
    public String sayHelloHtml() {
        return "<html> " + "<title>" + "Hello Jersey" + "</title>"
                + "<body><h1>" + "Hello World in REST" + "</body></h1>"
                + "</html> ";
    }
    @GET
    @Produces(MediaType.TEXT_XML)
    public String sayHelloXML() {
        return "<?xml version=\"1.0\"?>" + "<msg>" + "Hello World in REST"
                + "</msg>";
    }
    ...
}
```

Example 1: Hellow World with Jersey (3)

- The above resource supports two representations (XML and HTML).
- Jersey uses **content negotiation** to decide what representation to send as response.
  - JAX-RS understands the *Accept* header and will use it when dispatching the answer.
  - For example, an **Accept: text/html,application/xhtml,application/xml;q=0.9,/;q=0.8** means that the client prefers html or xhtml (q is 1 by default), raw XML second, and any other content type third * Your browser will always request HTML MIME type as the first preference.

Example 1: Register Jersey Servlet Dispatcher

- You need to add the resource you created and a jersey servlet dispatcher in the web.xml file
- The correct content of the web.xml file is shown below (the servlet and servlet-mapping below may come before the existing content in your default web.xml)
- If you don't have the web.xml, you can create a stub in eclipse by *Right click on Deployment Descriptor -> Generate Deployment Descriptor Stub
- Open it with double-click

```
<servlet>
   <servlet-name>SimpleRest</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>introsde.simple.rest.hello;introsde.simple.rest.ehealth</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
   <servlet-name>SimpleRest</servlet-name>
   <url-pattern>/rest/*</url-pattern>
 </servlet-mapping>
```

Example 1: Register Jersey Servlet Dispatcher

- **Important Note**: the 'param-name' tag "com.sun.jersey.config.property.packages" is a configuration parameter. Its corresponding 'param-value' tag value should point to the package where the resource classes are located (i.e. the package **"introsde.simple.rest.hello"** in our case). If your resource classes are located in the default package you can put either "." (without quotes) as the value or completely omit the 'param-name' and 'param-value' tags.
- Note the url-pattern in servlet-mapping. Any URL with this pattern will be handled by Jersey servlet dispatcher.

Example 1: Testing the HelloWorld service:

- **[Run the project]** Run the project (while the project is highlighted, do right mouse click, then choose 'Run As' -> 'Run on Server'. Follow the prompt to deploy the project to your Tomcat runtime.
- **[Test it first with your browser]** try 'http://localhost:8080/introsde.simple.rest.hello/rest/helloworld'. You should see the result of sayHelloHtml() (try viewing the source of the page returned).
- **[Test it with a REST-client tool]** you can use any REST client tool, below you will find links to some of them.

REST clients

- Postman [Chrome extension]
- Simple REST Client [Chrome extension]
- rest-client [Java, multi-platform]
- cocoa-rest-client [Mac OS X]
- Let me know if you find others!

Example 2: Simple client in Java

- Create a simple java program with a main as follows:

```
package introsde.simple.rest.client;
import java.net.URI;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriBuilder;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;
public class Test {
    public static void main(String[] args) {
        ClientConfig config = new DefaultClientConfig();
        // Creating the client
        Client client = Client.create(config);
        // Instantiating a web resource that will be requested from BASEURI
        WebResource service = client.resource(getBaseURI());
        // Make HTTP requests and process resources you get in response
        ...
```

Example 2: Simple client in Java

```
public class Test {
        ...
        // Making a GET request on BASEURI/rest/helloworld with
        // Accept header equal text/plain
        System.out.println(service.path("rest").path("helloworld")
                .accept(MediaType.TEXT_PLAIN).get(ClientResponse.class)
                .toString());
        // Get plain text
        System.out.println(service.path("rest").path("helloworld")
                .accept(MediaType.TEXT_PLAIN).get(String.class));
        // Get XML
        System.out.println(service.path("rest").path("helloworld")
                .accept(MediaType.TEXT_XML).get(String.class));
        // The HTML
        System.out.println(service.path("rest").path("helloworld")
                .accept(MediaType.TEXT_HTML).get(String.class));
    }
    private static URI getBaseURI() {
        return UriBuilder.fromUri(
                "http://localhost:8080/introsde.simple.rest.hello").build();
    }
}
```

- The example code is here

Brief Summary of Jersey annotations (1)

- **@PATH(your_path):** Sets the path to base URL + /your_path. The base URL is based on your application name, the servlet and the URL pattern from the web.xml configuration file.
- **@POST:** Indicates that the following method will answer to a HTTP POST request
- **@GET:** Indicates that the following method will answer to a HTTP GET request
- **@PUT:** Indicates that the following method will answer to a HTTP PUT request
- **@DELETE:** Indicates that the following method will answer to a HTTP DELETE request

Brief Summary of Jersey annotations (2)

- **@Produces(MediaType.TEXT_PLAIN [, more-types]):** defines which MIME type is delivered by a method annotated with @GET.
- **@Consumes(type [, more-types]):** defines which MIME type is consumed by this method.
- **@PathParam:** inject values from the URL into a method parameter. This way you inject for example the ID of a resource into the method to get the correct object.
- **@QueryParam:** inject values from the query parameters in the URL into a method parameter.
- The complete path to a resource is based on the base URL and the @PATh annotation in your class.

```
http://your_domain:port/display-name/url-pattern/path_from_rest_class
```

Further reading

- Take a look on deploying Jersey Services with the new Servlet 3.0 API
  - https://jersey.java.net/documentation/latest/deployment.html
- A good tutorial on REST with jersey:
  - http://www.vogella.com/articles/REST/article.html