Jakarta RESTful Web Services 3.1 Workshop Participant

Module 5: Java SE web services client.

Now that we have basic Java SE and Application Server based web services, let us look at writing client code to utilize the services. One of the strengths of web services is that they are language agnostic. If the HTTP protocols are followed, a client written in Java can utilize a web service written in C#. This also means that a client written in Java SE or, as we will see in the next module, Servlets/JSF can access a service on any host by just using the correct web service URL.

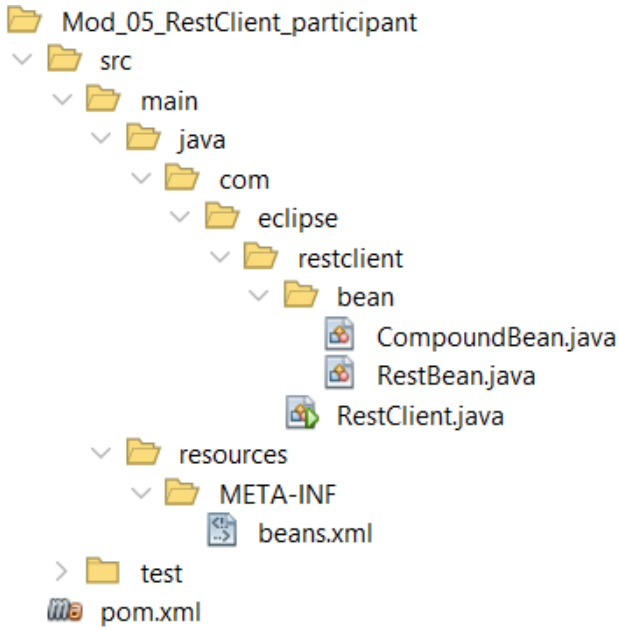
In this module we will see how to send and receive data in the JSON format and convert it to a Java object for both a server based and SeBootstrap based service. You will be working with two projects:

- mod_05_restserver
- mod_05_restclient_participant

The `Mod_05_RestServer` project is a complete `Mod_04_RestServer` with a working compound interest service. Build and deploy Mod_05_RestServer to the GlassFish server before working with the client.

The Project

This module's client is a Java SE client. The code we will examine will allow any Java SE desktop application to interact with any REST web service.



Let's look at the pom.xml file

For this module we will only work with the Jersey client from GlassFish library. We are not using GlassFish, just the libraries necessary for a client.

```
<dependency>
    <!-- Client to access a service -->
    <groupId>org.glassfish.jersey.core</groupId>
    <artifactId>jersey-client</artifactId>
    <version>3.1.3</version>
</dependency>
<dependency>
    <!-- Official ref implementation of JSON Binding API(JSR-367) -->
    <groupId>org.eclipse</groupId>
    <artifactId>yasson</artifactId>
    <version>3.0.3</version>
</dependency>
```

The `<plugins>` section is identical to the Mod 3 project where all dependencies must be added to the final jar and the `exec:exec` goal can execute the code.

Let's look at the beans.xml

This file is the same for all examples.

Let's look at the code

As we have seen in Module 4, a web service can return a JSON representation of an object if the return type of the service, GET or POST, is an object, such as a JavaBean in Java. This means that in the client code we need the same source code for the Java bean that the web service is returning as JSON, the RestBean.java. This class is already in the project.

In the RestClient.java file there is already one complete REST client method named `callHelloService`. Let us examine its code.

In this project you will be using JSON-B, B for binding. This is how we will convert to and from JSON and an object. The first line of code will create a `Jsonb` object for this purpose.

```
Jsonb jsonb = JsonbBuilder.create();
```

Now we can convert the `RestBean` that is passed to `callHelloService` into JSON.

```
String restJson = jsonb.toJson(restBean);
```

With this done we can now create the `Client` that will allow us to send and receive from a service.

```
Client client = ClientBuilder.newClient();
```

With the client object created, we need to describe where the service is, its `URI`, and any query parameters you wish to pass to the service. This will result in a `WebTarget` object.

```
WebTarget target = client.target(UriBuilder.fromUri(
                    "http://localhost:8080/Mod_05_RestServer/").
                     build());
```

In this example all services are found after the `services` path. With the `WebTarget` object, we can make a request to the `hello` service. The name of the service is the `@Path` of the service. In the case of `hello` we will need the following code added:

```
String serviceReturnJson
      = target.path("services").path("hello").request(
                MediaType.APPLICATION_JSON)
                .post(Entity.entity(restJson, MediaType.APPLICATION_JSON),
                String.class);
```

This will send the JSON representation of a `RestBean` to the service and receive in return a new bean as a JSON string. Now we need to use JSON-B to convert the JSON string into a RestBean.

```
restBean = jsonb.fromJson(serviceReturnJson, RestBean.class);
```

In the main method of this class, you call upon a service by:

```
RestClient restClient = new RestClient();
RestBean restBean = new RestBean();
restBean.setName("Ken");
restBean = restClient.callHelloService(restBean);
```

Your turn:

As the project is complete for running the hello service you can run it with the `mvn` command. Do not forget to build and deploy to GlassFish the `Mod_05_RestServer` first. As there is no UI you need to examine the log output of the program. Here are just the INFO lines from the log of my run:

```
INFO:
 ]]] Conversion of restBean to JSON= {"name":"Ken","serviceSource":"","theTime":"2023-09-
21T14:07:46.8935399"}

INFO:
 ]]] JSON of result from service call = {"name":"Ken","serviceSource":"GreetingService","theTime":"2023-
09-21T14:07:47.2445905"}

INFO:
 ]]] Conversion of JSON string to restBean = JSON{name=Ken, theTime=2023-09-21T14:07:47.244590500,
serviceSource=GreetingService}

INFO:
 ]]] restBean in main = JSON{name=Ken, theTime=2023-09-21T14:07:47.244590500,
serviceSource=GreetingService}

INFO:
 ]]] Conversion of restBean to JSON= {"name":"","serviceSource":"","theTime":"2023-09-
21T14:07:47.3194261"}
```

```
INFO:
 ]]] JSON of result from service call =
{"name":"Anonymous","serviceSource":"GreetingService","theTime":"2023-09-21T14:07:47.3304238"}

INFO:
 ]]] Conversion of JSON string to restBean = JSON{name=Anonymous, theTime=2023-09-21T14:07:47.330423800,
serviceSource=GreetingService}

INFO:
 ]]] restBean in main = JSON{name=Anonymous, theTime=2023-09-21T14:07:47.330423800,
serviceSource=GreetingService}
```

The `Mod_05_RestServer` already has a `CompoundInterest` service. Create a method to access this service. Examine the source code in the server to find the `@Path` of the service. Do not forget that the path to all services begins with `services`.

Feel free to create a UI in Swing or JavaFX for the services.

In our next module we will see how to have a client that a Servlet or a JSF page can use.