RESTFul Web Services in Java

Agenda

- Introduction to Jersey
  - Resource classes
  - Resource methods
- How to call a REST service
- How to create a REST resource

References:

- Learn REST: A RESTful Tutorial
- Jersey User Guide

What is Jersey?

Jersey is a framework to build RESTFul web services

- implements JAX-RS API (actually it is the reference implementation)
- provides further APIs to simplify the development of RESTful services and clients

The JAX-RS specification

- It defines a set of Java APIs for the development of RESTful web services: a set of **annotations** and **classes/interfaces** that may be used to expose POJOs as Web resources
- It assumes that HTTP is the underlying transportation protocol

Available at `https://github.com/jax-rs/spec/blob/master/spec.pdf`

POJOs: Plain Old Java Objects

A standard Java class (no Java Bean, etc.)

```
public class Calendar {
  public String now(){
    Date nowDate = new Date();
    return nowDate.toString();
  }
}
```

is transformed into ...

A POJO as a REST Resource

Using JS-RX annotation we can expose a Java class as a REST resource

```
@Path("mycalendar")
public class Calendar {

  @GET
  @Produces("text/plain")
  public String now(){
    Date nowDate = new Date();
    return nowDate.toString();
  }
}
```

Fundamental Concepts

- **Resource class**: a Java class that uses a JS-RX annotations
- **Root resource class**: A resource class annotated with `@Path`
- **Request method designator**: a Java annotation used to identify a HTTP method
- **Resource method**: a method of a resource class annotated with a request method designator

Working with Jersey

- Jersey is built using Apache Maven and all modules of the framework are available on the Central Maven Repository
- Starting from a Maven project is the most convenient way for working with Jersey, in a terminal simply run

```
mvn archetype:generate -DarchetypeArtifactId=jersey-quickstart-grizzly2 \
-DarchetypeGroupId=org.glassfish.jersey.archetypes -DinteractiveMode=false \
-DgroupId=com.example -DartifactId=simple-service -Dpackage=com.example \
-DarchetypeVersion=2.29
```

  or create a Maven project in Eclipse

Note on Java 9

To avoid a run-time exception you should add the following to your `pom.xml` file

```
<dependency>
    <groupId>javax.xml</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.1</version>
</dependency>
```

Create A Client

In the client API a resource is an instance of the Java class `WebTarget` that encapsulates an URI and allows calling HTTP methods

Basic steps to follow to perform a request

1. Create an instance of the `Client` class
2. Target a web resource obtaining an instance of `WebTarget`
3. Identify the resource and setting possibly parameters via the methods of `WebTarget`
4. Create and set up the HTTP request through an `Invocation.Builder` instance
5. Perform the request: the result is an instance of `Response` class
6. Read and process the answer

Create An Instance of `Client` class

Use the static methods of the factory class `ClientBuilder`

`Client newClient()` Create a new instance using the default client builder implementation

`Client newClient(Configuration config)` Create a new instance by providing initial configuration. A configuration allows specifying providers and setting up properties

Example

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

Target a Web Resource

We create an instance of `WebTarget` class by using the `target` method of `Client` class and specifying the URL or or the root URL of the remote resource

- `WebTarget target(String uri)`
- `WebTarget target(Link uri)`
- `WebTarget target(URI uri)`
- others see documentation

Example

```
WebTarget webTarget = client.target("http://exaple.com/rest");
```

Identify The Resource

- If we have a WebTarget pointing at the URI of the context root of the REST service, we can obtain a target to a specific resource by using the method `path`
- We can set parameters on the URI by using the method `queryParam`

Example

```
WebTarget resourceWebTarget = webTarget.path("resource");
WebTarget targetWithQueryParam =
        webTarget.queryParam("greeting", "Hi World!");
```

Prepare A Request For Invocation

- An `Invocation.Builder` instance is used to setup request specific parameters, for example cookie parameters or other header of the HTTP request
- To obtain an instance of `Invocation.Builder`, use the `request` method of the class `WebTarget`
- There are several `request` methods that are available on WebTarget, some of which accept parameters that set the media type of the representation
- The method `header` of `Invocation.Builder` allows customizing the header of the HTTP request

Example

```
Invocation.Builder invocationBuilder =
    webTargetWithQueryParam.request(MediaType.TEXT_PLAIN_TYPE);
```

We tell Jersey to add a “Accept: text/plain” HTTP header to our request

Perform A Request

Some methods in `Invocation.Builder` for synchronous invocation, see documentation for the other ones

`Response get()` perform a HTTP get request

`Response get(Class<T> responseType)` perform a get request where T is the Java type the response will be converted to

`Response delete()` perform a HTTP delete request

Example

```
Response response = invocationBuilder.get();
```

Process A Response

An instance of `Response` class represents a HTTP response

`int getStatus()` returns the status code associated with the response

`MediaType getMediaType()` returns the media type of message entity

`T readEntity(Class<T> entityType)` returns a Java object of type T that represents the content of the response

Example

```
int status = response.getStatus();
String text = response.readEntity(String.class);
```

Example Of REST API Client

We use an on-line REST API for testing available at
`https://jsonplaceholder.typicode.com/`

Available resources

`/posts`, `/comments`, `/albums`, `/photos`, `/todos`, `/users`

How to invoke

GET	/posts
GET	/posts/1
GET	/posts/1/comments
GET	/comments?postId=1
GET	/posts?userId=1

An Example Of Invocation

Retrieve all comments with `postId` equal to 1

```
Client client = ClientBuilder.newClient();
WebTarget target = client.target(BASE_URI)
                         .path(RESOURCE)
                         .queryParam("postId", "1");
Invocation.Builder request =
                   target.request(MediaType.APPLICATION_JSON);
Response res = request.get();
System.out.println("Request status" + res.getStatus());
System.out.println(res.readEntity(String.class));
```

REST Resources

- A REST resource is a Java class annotated with JS-RX annotations
- Annotations can be used to specify
  - the name (URI) of a resource
  - the Java method will process a given HTTP request
  - the parameters of a request and the mapping to the parameters of the corresponding Java method
  - the MIME media types of representations a resource can produce and send back to the client
  - the MIME media types of data that can be consumed by a resource

Example

Our calendar resource

```
@Path("mycalendar")
public class Calendar {

  @GET
  @Produces("text/plain")
  public String now(){
    Date nowDate = new Date();
    return nowDate.toString();
  }
}
```

`@Path` Annotation

- Its value is the relative URI where the Java class will be hosted
- It can also contain **URI path templates**, i.e., URI with variables between curly braces

```
@Path("/users/{username}")
```

  where `username` is a variable
- To obtain the value of a variable use the `@PathParam` annotation on method parameter of a request method

```
public String getUser(@PathParam("username") String userName) {
        ...
}
```

- To constrain the value of a variable is possible to use regular expression

```
@Path("users/{username: [a-zA-Z][a-zA-Z_0-9]}")
```

Resource Method Designators

`@GET`, `@PUT`, `@POST`, `@DELETE` and `@HEAD` decorate the Java method responsible to process the corresponding HTTP method

```
@DELETE
public void deleteRes() {
  ...
}
```

See the documentation for further details

@Produce Annotation

- It is used to specify the MIME media types of representations a resource can produce and send back to the client
- It can be applied at both the class and method levels — the method level overrides the class one
- If a resource is capable of producing more that one MIME media type then the resource method chosen will correspond to the most acceptable media type as declared by the client

```
@Path("/myResource")
@Produces("text/plain")
public class SomeResource {
    @GET
    public String doGetAsPlainText() {
        ...
    }

    @GET
    @Produces("text/html")
    public String doGetAsHtml() {
        ...
    }
}
```

`@Consumes` Annotation

- It is used to specify the MIME media type of representations that can be consumed by a resource
- It can be specified at both the class and the method level
- More than one media type can be specified in the same `@Consumes` declaration

```
@POST
@Consumes("text/plain")
public void postMessage(String message) {
    ...
}
```

`@QueryParam` Annotation

- It is used to extract query parameters from the Query component of the request URL
- When query parameter is not present in the request then value will be
  - an empty collection for `List`, `Set` or `SortedSet`
  - `null` for other object types
  - the Java-defined default for primitive types
- The annotation `@DefaultValue` allows to specify a default value to use when the parameter is not present in the request

```
@GET
public Response item(
        @DefaultValue("2") @QueryParam("index") int i ) {
...
}
```

Example of RESTful Web Service

```
@Path("/temperature")
public class TempSensor {
    // ...
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getIt(@DefaultValue(KELVIN) @QueryParam("kind") String kind) {
        double temp = currentTemp();
        switch(kind) {
        case CELSIUS:    temp = toCelsius(temp);     break;
        case FAHRENHEIT: temp = toFahrenheit(temp);  break;
        }
        return String.valueOf(temp);
    }
}
```

Conclusion

We have seen

- Introduction to Jersey
  - Resource classes
  - Resource methods
- How to call a REST service
- How to create a REST resource

References:

- RESTful Web Services: A Tutorial
- Jersey User Guide