7

Creating RESTFul Clients in Java

Objectives

After completing this lesson, you should be able to do the following:

- Use Java SE APIs to make HTTP requests
- Use Jersey Client APIs to make HTTP requests
- Process XML and JSON in a RESTful web service client

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 1: Introduction to Web Services
- Lesson 2: Creating XML Documents
- Lesson 3: Processing XML with JAXB
- Lesson 4: SOAP Web Services Overview
- Lesson 5: Creating JAX-WS Clients

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 6: Exploring REST Services
- **Lesson 7: Creating REST Clients** — **You are here!**
- Lesson 8: Bottom Up JAX Web Services
- Lesson 9: Top Down JAX Web Services
- Lesson 10: Implementing JAX RS Web Services

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 11: Web Service Error Handling
- Lesson 12: Java EE Security and Securing JAX WS

Communicating with Web Servers

- Java provides a simple mechanism for communicating with HTTP servers via `URL` objects and their associated `URLConnections`.
- Jersey provides a client API for convenient access to JAX-RS web services.
- Third-party libraries, such as Apache's `HttpClient`, provide finer-grained access to HTTP servers.

Simplest Java Client

```
1  public class SimplestClient {
2   static public void main( String[] args )
3          throws Exception {
4    String contextURL = "http://localhost:8080/jaxrs";
5    String resourcePath = "/airports";
6    String requestPath = "/numAirports";
7    String urlString =
8       contextURL + resourcePath + requestPath;
9    URL url = new URL( urlString );
10   InputStream result = (InputStream) url.getContent();
11   Scanner scanner = new Scanner( result );
12   System.out.println( "Result:_" + scanner.next() );
13  }
14 }
```

PathParam Java Client

```
1  public class PathParamClient {
2   static public void main( String[] args )
3         throws Exception {
4    String contextURL = "http://localhost:8080/jaxrs";
5    String resourcePath = "/airports";
6    String requestPath = "/nameByCode/";
7    String param = "LGA"; // need URL-encoding
8    String urlString =
9      contextURL + resourcePath + requestPath + param;
10   URL url = new URL( urlString );
11   InputStream result = (InputStream) url.getContent();
12   BufferedReader reader =
13    new BufferedReader(new InputStreamReader(result));
14   System.out.println( "Result:_" + reader.readLine() );
15  }
16 }
```

FormParam Java Client

```
1  public class FormParamClient {
2   static public void main( String[] args )
3         throws Exception {
4     String contextURL = "http://localhost:8080/jaxrs";
5     String resourcePath = "/airports";
6     String requestPath = "/add";
7     String code = "LGA";          // need URL-encoding
8     String name = "LaGuardia"; // need URL-encoding
9     String urlString =
10          contextURL + resourcePath + requestPath;
11    URL url = new URL( urlString );
12    HttpURLConnection connection =
13          (HttpURLConnection) url.openConnection();
```

- `URLConnection` provides more control.

```
14    connection.setRequestMethod( "POST" );
15    connection.setAllowUserInteraction( true );
16    connection.setDoOutput( true );
17    connection.setDoInput( true );
18    connection.connect();
19    OutputStream os = connection.getOutputStream();
20    PrintWriter writer = new PrintWriter( os );
21    writer.print( "code=" + code + "&name=" + name );
22    writer.close();
23    InputStream result = connection.getInputStream();
24    BufferedReader reader =
25     new BufferedReader( new InputStreamReader(result) );
26    System.out.println( "Result: " + reader.readLine() );
27   }
28 }
```

Drawbacks of the Simple Approach

- Requires explicit matching of URL rewrite rules:
  - To avoid invalid URLs, parameters may need to be provided using URL-encoding.
- Requires some awareness of the structure of HTTP messages
- Requires low-level I/O programming

Method Chaining

Method chaining is a programming idiom commonly used by Jersey. For any method that returns `void`, instead return `this`.

```
Person p1 = new Person();
p1.setFirstName("Sherlock");
p1.setLastName("Holmes");
p1.setAddress("221B Baker Street");
```

Can be replaced with:

```
Person p2 = new Person()
        .setFirstName("Sherlock")
        .setLastName("Holmes")
        .setAddress("221B Baker Street");
```

The Jersey Client API

The Jersey Client API revolves around two entities:

- `WebResource` instances represent JAX-RS resources.
  - Communications between the client and the JAX-RS resource are encapsulated within these instances.
- `Client` defines a configuration point for the Jersey run time. It also acts as a factory for `WebResources`.

Simplest Jersey Client

```
1  public class SimplestJerseyClient {
2   static public void main( String[] args ) {
3    String contextURL = "http://localhost:8080/jaxrs";
4    String resourcePath = "/airports";
5    String requestPath = "/numAirports";
6    String urlString =
7          contextURL + resourcePath + requestPath;
8    Client client = Client.create();
9    WebResource resource =
10    client.resource( urlString );
11   String result = resource.get( String.class );
12   System.out.println( "Result: " + result );
13  }
14 }
```

Customizing Request Message

`WebResource` allows the application to customize:

- An HTTP method used by request
  - Including payload, when the method allows it
- Request Headers
- Query Parameters
- Request Cookies

QueryParam Jersey Client

```
1  public class QueryParamJerseyClient {
2   static public void main( String[] args ) {
3    String contextURL = "http://localhost:8080/jaxrs";
4    String resourcePath = "/airports";
5    String requestPath = "/codeByName";
6    String name = "LaGuardia"; // No URL-Encoding!
7    String urlString =
8          contextURL + resourcePath + requestPath;
9    Client client = Client.create();
10   WebResource resource =
11    client.resource( urlString );
12   String result =
13    resource.queryParam("name",name).get(String.class);
14   System.out.println( "Result: " + result );
15  }
16 }
```

WebResource.get

- `WebResource.get` accepts a type (or a list of types) to use when creating the value to return to the caller. The types accepted as parameters include:
  - Classes with a constructor that accepts a single String
  - JAXB classes
- It is also possible to specify the representation that the client expects for the payload of the reply message.

Specifying Expected Return Type

```
1  public class JSONObjectJerseyClient {
2   static public void main( String[] args ) {
3    String urlString =
4     "http://localhost:8080/jaxrs/airports/byCode/LGA";
5    Client client = Client.create();
6    WebResource resource =
7     client.resource(urlString);
8    Airport result =
9     resource
10    .accept( "application/json" )
11    .get( Airport.class );
12   System.out.println( "Result: " + result );
13  }
14 }
```

Submitting Form Data

```
1  public class FormSubmitJerseyClient {
2   static public void main( String[] args ) {
3    String url = "http://localhost:8080/jaxrs/airports/add";
4    Client client = Client.create();
5    WebResource resource = client.resource(url);
6    MultivaluedMap<String,String> params =
7     new MultivaluedMapImpl();
8    params.add( "code", "JFK" );
9    params.add( "name", "John_F._Kennedy_Airport" );
10   String result =
11     resource
12     .type( "application/x-www-form-urlencoded" )
13     .post( String.class, params );
14   System.out.println( "Result:_" + result );
15  }
16 }
```

Obtaining Reply Metadata

`ClientResponse` represents the complete reply message received by the client. Its API allows the application to access:

- Status code
- Message payload
- Response Headers
- Response Cookies

Obtaining Reply Metadata

```
1  public class ClientResponseJerseyClient {
2   static public void main( String[] args ) {
3    String urlString =
4      "http://localhost:8080/jaxrs/airports/byCode/LGA";
5    Client client = Client.create();
6    WebResource resource = client.resource(urlString);
7    ClientResponse response =
8       resource.accept( "application/json" )
9       .get( ClientResponse.class );
10   System.out.println("Code: " +
11                 response.getStatus());
12   System.out.println("Result: " +
13                 response.getEntity(Airport.class));
14  }
15 }
```

Jersey Client API Filters

- Filters allow clients to perform common operations on any JAX-RS interaction, independent of which particular interaction they each are.
- Filters are similar to `ServletFilter` in the Servlet specification, or `Handler` in the JAX-WS specification.

Simple Client with Logging Filter

```
1  public class LoggingClient {
2   static public void main( String[] args ) {
3    String urlString =
4       "http://localhost:8080/jaxrs/airports/byCode/LGA";
5    Client client = Client.create();
6    client.addFilter( new LoggingFilter() );
7    WebResource resource = client.resource(urlString);
8    Airport result =
9       resource
10      .accept( "application/json" )
11      .get( Airport.class );
12   System.out.println( "Result:_" + result );
13  }
14 }
```

Jersey and JSON

- The Jersey client supports marshalling and unmarshalling JAXB objects as shown in the previous slides. The Jersey client can use the Jackson JSON library to support JSON.

```
Quote quote = resource
    .accept(MediaType.APPLICATION_JSON)
    .get(Quote.class);
```

- The support of JSON is not limited to JAXB classes.

```
ClientConfig clientConfig = new DefaultClientConfig();
clientConfig.getFeatures().put(JSONConfiguration.FEATURE_POJO_MAPPING,
Boolean.TRUE);
Client client = Client.create(clientConfig);
```

Reading and Writing JSON

Java does not have a standard for reading and writing JSON (yet). The Jackson JSON library can be used stand-alone or with a Jersey JSONJAXBContext.

```
InputStream in = new FileInputStream("src/sample.json");
```

Reading a POJO with Jackson:

```
ObjectMapper mapper = new ObjectMapper();
PojoQuote pojoQuote = mapper.readValue(in, PojoQuote.class);
```

Reading a JAXB class with Jersey:

```
JSONJAXBContext jsonJaxbContext =
    new JSONJAXBContext(Quote.class);
JSONUnmarshaller u =
jsonJaxbContext.createJSONUnmarshaller();
Quote quote = u.unmarshalFromJSON(in, Quote.class);
```

The JAX-RS 1.1 specification defines a standard client API.

a. True

b. False

Which Jersey class represents a REST resource that is located at a URL?

a. Client
b. URL
c. WebResource
d. ClientResponse

Resources

Topic	Website
HttpURLConnection	http://docs.oracle.com/javase/7/docs/api/java/net/HttpURLConnection.html
Jersey Client API	http://jersey.java.net/nonav/documentation/latest/client-api.html
Jackson JSON Processor	http://wiki.fasterxml.com/JacksonHome

Summary

In this lesson, you should have learned how to:

- Use Java SE APIs to make HTTP requests
- Use Jersey Client APIs to make HTTP requests
- Process XML and JSON in a RESTful web service client

Practice 7 : Overview

This practice covers the following topics:

- Calling REST Services with URLConnection
- Using the Jersey Client API
- Modifying a JavaScript (jQuery) REST Client
- Properties of a RESTful Web Service