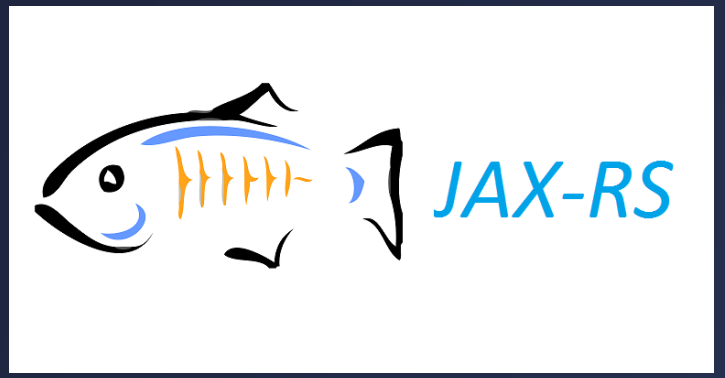


Web Service and JAX-RS

Sadegh Aliakbary

Agenda

- What Are RESTful Web Services?
- JAX-RS Standard
- Restful Design and API Elements
- Building Simple Web-Services

Web Services: Definition

- (W3C) A Web service is:
  - A software system
    designed to support interoperable machine-to-machine interaction
    over a network

- A Web service is:
  - a collection of functions
    that are packaged as a single entity
    and published to the network
    for use by other programs

Why Web Services ?

- **Distributed** architecture
- **Interoperability**
  - Based on open standards
  - Utilizes existing infrastructure
  - Can be accessed from **any programming language**
- Based on internet **standards**
  - XML, XSD, http, etc.
- Ease of use
  - Easy to understand concepts
  - Easy to implement

Java Standards for Web Services

- **JAX-WS** (Java API for XML-Based Web Services)
  - A standard way to develop a Web- Services in **SOAP** notation
  - SOAP: Simple Object Access Protocol
  - SOAP Web-Services need a **WSDL** (web service definition language)
- **JAX-RS** (Java API for RESTful Web Services)
  - Typically used for XML or JSON data exchange
  - RESTful Web Services are represented as resources
    - and can be identified by Uniform Resource Identifiers (URI).
  - Remote procedure call over HTTP (HTTP Get, POST, DELETE, etc.)

RESTful Web Services

What Are RESTful Web Services?

- Representational State Transfer (**REST**)
  - an architectural style of client-server application
  - centered around the **transfer** of **representations** of **resources** through requests and responses.
- **RESTful web services** are loosely coupled, lightweight web services
  - for creating APIs for clients
- In the REST architectural style, data and functionality are considered **resources** and
  - Resources accessed using **Uniform Resource Identifiers (URIs)**
- The resources are represented by documents, and are requested using well-defined operations (**get, delete**, ...).

REST

- REST = Representational State Transfer
- Data oriented – not procedure (RPC) oriented
- Resources
  - Identified by URI
  - Accessed by HTTP methods
    - GET
    - PUT
    - POST
    - DELETE

RESTful Web Services with JAX-RS

- **Java API for RESTful Web Services**
- **JAX-RS** is a Java programming language API
- Make it easy to develop applications that use the **REST architecture**
- JAX-RS uses Java **annotations**
  - to simplify the development of RESTful web services
- Developers decorate Java class files with JAX-RS annotations
  - to define **resources** and the **actions** on those resources
- JAX-RS services are based on **Servlet** standard.
  - A JAX-RS service may access servlet features, such as HttpServletRequest

JAX-RS: Example

```
@Path("helloworld")
public class HelloWorld {

    @GET
    @Produces("text/html")
    @Path("show")
    public String getHtml() {
        return "<html lang=\"en\"><body><h1>Hello, World!!</h1></body></html>";
    }

    @POST
    @Consumes("text/plain")
    @Path("store")
    public void postClickedMessage(String message) {
       //store the message
    }
}
```

JAX-RS Annotations

Annotation	Description
@Path	A relative **URI** path indicating where the Java class will be hosted
@ApplicationPath	The base URI for all resource URIs specified by **@Path** annotations
@GET, @POST, @PUT, @DELETE, @HEAD	Request method (HTTP methods). The annotated method will process HTTP requests.
@Consumes	The **MIME media** types a resource can consume (that were sent by the client) e.g., JSON, XML, plain text, …
@Produces	The **MIME media** types a resource can produce (and send back to the client) e.g., JSON, XML, plain text, …

JAX-RS Annotations

Annotation	Description
@PathParam	Binds the method parameter to a path segment. Extracted from the request URI. Parameter names correspond to URI path template variable names specified in the @Path class-level annotation. e.g., `@Path("/users/{username}")`
@QueryParam	Binds the method parameter to an HTTP **query parameter.** e.g., http://example.com/service?name=ferret

A JAX-RS Example (revisited)

```
@Path("helloworld")
public class HelloWorld {

    @GET
    @Produces("text/html")
    @Path("show")
    public String getHtml() {
        return "<html lang=\"en\"><body><h1>Hello, World!!</h1></body></html>";
    }

    @POST
    @Consumes("text/plain")
    @Path("store")
    public void postClickedMessage(String message) {
       //store the message
    }
}
```

The @Produces Annotation

- **@Produces**
- The **MIME media** types a resource can produce and send back to the client.
- The value of @Produces is:
  - an array of String of MIME types,
  - or a comma-separated list of MediaType constants.
- For example:

```
@Produces({"image/jpeg,image/png"})
```

```
@Produces(MediaType.APPLICATION_XML)
```

The @Consumes Annotation

- **@Consumes**
- specify which **MIME media** types a resource can accept from the client
- If a resource is unable to consume the MIME type of a client request: The JAX-RS runtime sends back an **HTTP 415** ("**Unsupported Media Type**") error.
- The value of @Consumes is an array of String of acceptable MIME types, or a comma-separated list of MediaType constants.
- For example:

```
@Consumes({"text/plain,text/html"})
```

```
@Consumes({MediaType.APPLICATION_JSON, MediaType.TEXT_PLAIN})
```

Extracting Request Parameters

- Parameters of a resource method may be annotated with parameter-based annotations to extract information from a request.
- You can extract the following types of parameters for use in your resource class:
  - **Query**
  - **URI path**
  - **Form**
  - **Cookie**
  - **Header**
  - **Matrix**

@PathParam and URI Path Templates

- URI path templates are URIs with variables embedded within the URI syntax.
- Variables are denoted by braces ({ **and** }).
- For example:

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

```
"http://example.com/users/Ali"
```

```
public class UserResource {
    @GET
    @Produces("text/xml")
    @Path("/users/{username}")
    public String getUser(@PathParam("username") String userName) {
        ...
    }
}
```

Examples of URI Path Templates

URI Path Template	URI After Substitution
http://example.com/{name1}/{name2}/	http://example.com/james/gatz/
http://example.com/{year}/{month}/{day}/	http://example.com/2015/11/21/
http://example.com/maps/{location}	http://example.com/maps/Main%20Street

URI Path

```
@Path("/users")
public class UserRestService {
    @GET
    @Path("getById/{id}")
    public Response getUserById(@PathParam("id") String id) {
        return Response.status(200).
               entity("getUserById is called, id : " + id).build();
    }

    @GET
    @Path("getHistory/{year}/{month}/{day}")
    public Response getUserHistory(@PathParam("year") int year,
           @PathParam("month") int month, @PathParam("day") int day) {
        String date = year + "/" + month + "/" + day;
        return Response.status(200).
               entity("getUserHistory is called, year/month/day : " + date).
               build();
    }
}
```

/users/getById/22667788

/users/getHistory/2016/12/21

Query parameters

- **Query parameters** are extracted from the request URI query parameters
- Example:

/smooth?step=5&max-m=false

```
@Path("smooth")
@GET
public Response smooth(
        @DefaultValue("2") @QueryParam("step") int step,
        @DefaultValue("true") @QueryParam("min-m") boolean hasMin,
        @DefaultValue("true") @QueryParam("max-m") boolean hasMax,
        @DefaultValue("true") @QueryParam("last-m") boolean hasLast)
{
         ...
}
```

Query parameters

```
@Path("/employee")
public class EmployeeWebService {
    @GET
    @Path("/query")
    public Response getEmployee(@QueryParam("from") int from,
                                @QueryParam("to") int to,
                                @QueryParam("orderBy") List<String> orderBy) {
        return Response
                .status(200)
                .entity("getEmployee is called, from : " + from + ", to : " + to
                        + ", orderBy" + orderBy.toString()).build();
    }


}
```

"employee/query?from=100&to=200&orderBy=age&orderBy=name"

Other Parameters, Example:

```
@POST
@Path("/store")
@Consumes("application/x-www-form-urlencoded")
public void post(@FormParam("name") String name) {
    //store the message
}
```

```
@GET
   @Path("/get")
   public Response removeEmployee(@HeaderParam("user-agent") String userAgent) {
       return Response.status(200)
               .entity("remove is called, employeeId : " + userAgent)
               .build();
   }
```

@Context Parameters

- Injects a variety of resources in RESTful services.
- Some of the commonly injected components are:
  - HTTP headers
  - HTTP URI related information
  - HTTP Request

@Context Example

```
@Path("testinject")
public class InjectURIDetails{
 //localhost:8080/<root-context>/testinject/httpheaders
 @GET
 @Path("httpheaders")
 public void test(@Context HttpHeaders headers){
   MultivaluedMap<String, String> list = headers.getRequestHeaders();
   String accept = headers.getHeaderString("Accept");
   String test = headers.getCookies().get("TestCookie").getValue();
}
```

@Context Example

```
@Path("testinject")
public class InjectURIDetails{
  //localhost:8080/<root-context>/testinject/uriinfo
  @GET
  @Path("uriinfo")
  public void test(@Context UriInfo uriDetails){
     MultivaluedMap<String, String> params = uriDetails.getQueryParameters();
     List<String> id = uriDetails.getQueryParameters().get("id");
     URI uri = uriDetails.getRequestUri();
}
```

@Context Example

```
@Path("testinject")
public class InjectURIDetails{
 @GET
 @Path("httpheaders")
 public void test(@Context HttpServletRequest request){
     String name = request.getParameter("name");
     Integer count = (Integer)request.getAttribute("count");
     User u = request.getSession().getAttribute("user");
 }
}
```

JSON and XML Data Binding in JAX-RS Services

Using JSON and XML with JAX-RS and JAXB

- JAX-RS can **automatically** read and write XML and JSON
  - using JAXB
- **JSON** is a simple and popular text-based format for data exchange
- For example, the XML & JSON representation of a product instance is:

```
public class Product {
    String id;
    String name;
    String description;
    int price;
    //other function
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<product>
    <id>1</id>
    <name>Pride</name>
    <description>Pride Car</description>
    <price>12</price>
</product>
```

```
{
  "id":"1",
  "name":"Pride",
  "description":"Pride Car",
  "price":12
}
```

Using JSON with JAX-RS

- You can add the format **application/json** or **MediaType.APPLICATION_JSON** to the **@Produces** annotation in resource methods to produce responses with JSON data:

```
@GET
@Path("/get")
@Produces("application/json")
public Employee getEmployee() { ... }
```

- Fortunately, java-objects can be automatically converted to JSON texts using JAXB libraries

- Similarly, JAX-RS services may consume json (or xml) data, seamlessly as java objects (thanks JAXB standard)

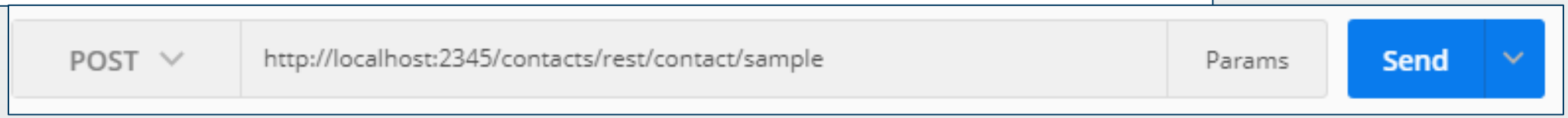
```
@POST
@Path("/create")
@Consumes("application/json")
public Response createEmployee(Employee emp) { ... }
```

Using JSON and XML with JAX-RS and JAXB

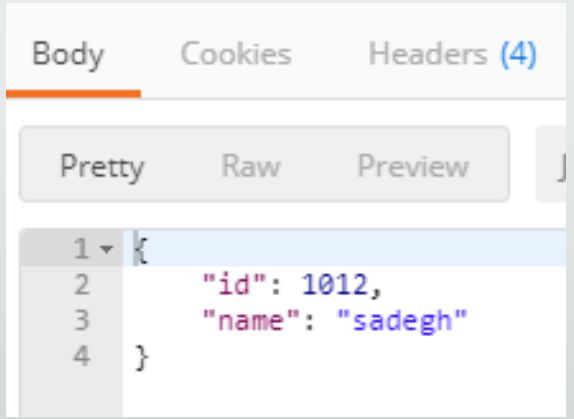
```
@Path("/employee")
public class EmployeeService {
    @GET
    @Path("/get")
    @Produces(MediaType.APPLICATION_JSON)
    public Employee getEmployeeInJSON() {
        Employee employee = new Employee();
        employee.setName("ali");
        employee.setAge(28);
        return employee;
    }

    @POST
    @Path("/post")
    @Consumes(MediaType.APPLICATION_JSON)
    public Response createEmployeeInJSON(Employee employee) {
        String result = "Employee saved : " + employee;
        return Response.status(201).entity(result).build();
    }
}
```

Example

```java
@POST
@Path("/sample")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public ContactEntity sample(ContactEntity entity){
    entity.setId(1000+entity.getId());
    return entity;
}
```



Response:



Integrating JAX-RS with Spring

```
@Named("employeeRestService")
@Path("/employee")
public class EmployeeApiService {

    @Inject
    EmployeeManager manager;

    @GET
    @Path("/add")
    @Consumes(MediaType.APPLICATION_JSON)
    public Response saveEmployee(Employee employee) {
        manager.save(employee);
        return Response.status(200).entity("employee is inserted").build();
    }
}
```

The End