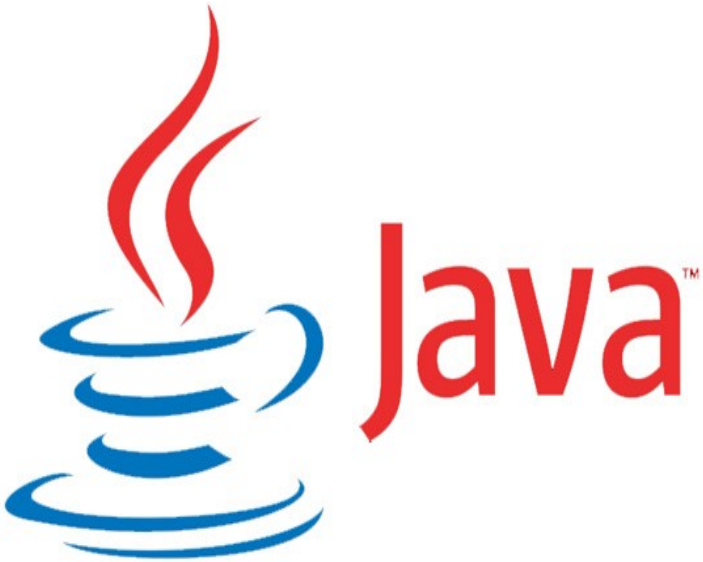


Developing RESTful Web services with JAX-RS

Sabyasachi Ghosh, Senior Application Engneer
Oracle India, @neilghosh

Java API for RESTful Web Services (JAX-RS)

Standard annotation-driven API that aims to help developers build RESTful Web services in Java

RESTful Application Cycle

Resources are identified by URIs

↓

Clients communicate with resources via requests using a standard set of methods

↓

Requests and responses contain resource representations in formats identified by media types

↓

Responses contain URIs that link to further resources

- Give everything an ID
- Standard set of methods
- Link things together
- Multiple representations
- Stateless communications

- ID is a URI

http://example.com/widgets/foo

http://example.com/customers/bar

http://example.com/customers/bar/orders/2

http://example.com/orders/101230/customer

Resources are identified by URIs

- Resource == Java class
  - POJO
  - No required interfaces
- ID provided by `@Path` annotation
  - Value is relative URI, base URI is provided by deployment context or parent resource
  - Embedded parameters for non-fixed parts of the URI
  - Annotate class or “sub-resource locator” method

Resources are identified by URIs

```
@Path("orders/{order_id}")
public class OrderResource {

  @GET
  @Path("customer")
  CustomerResource
getCustomer(@PathParam(“order_id”)int id) {...}
}
```

Standard Set of Methods

Method	Purpose
GET	Read, possibly cached
POST	Update or create without a known ID
PUT	Update or create with a known ID
DELETE	Remove

Standard Set of Methods

- Annotate resource class methods with standard method
  - `@GET`, `@PUT`, `@POST`, `@DELETE`, `@HEAD`
  - `@HttpMethod` meta-annotation allows extensions, e.g. WebDAV
- JAX-RS routes request to appropriate resource class and method
- Flexible method signatures, annotations on parameters specify mapping from request
- Return value mapped to response

Standard Set of Methods

```
@Path("properties/{name}")
public class SystemProperty {

  @GET
  Property get(@PathParam("name") String name)
    {...}

  @PUT
  Property set(@PathParam("name") String name,
    String value) {...}

}
```

Parameters

Template parameters and Regular expressions

```
@Path("customers/{firstname}-{lastname}")
```

```
@Path("{id : \\d+}")
```

Matrix Parameters

```
example.cars.com/mercedes/e55;color=black/2006
```

Query Paremetsrs

```
DELETE /orders/233?cancel=true
```

Multiple Representations

- Offer data in a variety of formats
  - XML
  - JSON
  - (X)HTML
- Maximize reach
- Support content negotiation
  - Accept header
    ```
    GET /foo
    Accept: application/json
    ```
  - URI-based
    ```
    GET /foo.json
    ```

Content Negotiation: Accept Header

Accept: application/xml
Accept: application/json;q=1.0, text/xml;q=0.5, application/xml;q=0.5,

```
@GET
@Produces({"application/xml","application/json"})
Order getOrder(@PathParam("order_id") String id) {
   ...
}

@GET
@Produces("text/plain")
String getOrder2(@PathParam("order_id") String id) {
   ...
}
```

Content Negotiation: URL-based

```
@Path("/orders")

public class OrderResource {

    @Path("{orderId}.xml")
    @Produces(“application/xml”)
    @GET
    public Order getOrderInXML(@PathParam("orderId") String
orderId) {

      . . .
    }



    @Path("{orderId}.json")
    @Produces(“application/json”)
    @GET
    public Order getOrderInJSON(@PathParam("orderId") String
orderId) {

      . . .
    }

}
```

Content Negotiation

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.PathParam;
import javax.inject.Inject;
import javax.enterprise.context.RequestScoped;

@Path("/actor/{id}")
@RequestScoped
public class ActorResource {
    @Inject DatbaseBean db;

    @GET
    @Produces("application/json")
    public Actor getActor(@PathParam("id") int id) {
        return db.findActorById(id);
    }
}
```

Link Things Together

```
<order self="http://example.com/orders/101230">
  <customer ref="http://example.com/customers/bar">
  <product ref="http://example.com/products/21034"/>
  <amount value="1"/>
</order>
```

Responses Contain Links

```
HTTP/1.1 201 Created
Date: Wed, 03 Jun 2009 16:41:58 GMT
Server: Apache/1.3.6
Location: http://example.com/properties/foo
Content-Type: application/order+xml
Content-Length: 184

<property self="http://example.com/properties/foo">
  <parent ref="http://example.com/properties/bar"/>
  <name>Foo</name>
  <value>1</value>
</order>
```

Responses Contain Links

- **`UriInfo`** provides information about deployment context, the request URI and the route to the resource
- **`UriBuilder`** provides facilities to easily construct URIs for resources

Responses Contain Links

```
@Context UriInfo i;

SystemProperty p = ...
UriBuilder b = i.getBaseUriBuilder();
URI u = b.path(SystemProperties.class)
   .path(p.getName()).build();

List<URI> ancestors = i.getMatchedURIs();
URI parent = ancestors.get(1);
```

Responses Contain Links

```
UriBuilder builder = UriBuilder.fromPath("/customers/
{id}");

builder.scheme("http")
.host("{hostname}")
.queryParam("param={param}");
```

```
http://{hostname}/customers/{id}?param={param}
```

```
UriBuilder clone = builder.clone();
URI uri = clone.build("example.com", "333",
"value");
```

```
http://example.com/customers/333?param=value
```

Response codes and Exception

```
Successful HTTP : 200 to 399
200 – OK
204 – No Content

Standard HTTP error : 400 to 599
404 – Not Found
406 - Not Acceptable
405 - Method Not Allowed

java.lang.Exception
java.lang.RuntimeException
javax.ws.rs.WebApplicationException

throw new
WebApplicationException(Response.Status.NOT_FOUND);
```

Stateless Communications

- Long lived identifiers
- Avoid sessions
- Everything required to process a request contained in the request

Deployment

- JAX-RS application packaged in **`WAR`** like a servlet
- For JAX-RS aware containers
  - **`web.xml`** can point to **`Application`** subclass
- For non-JAX-RS aware containers
  - **`web.xml`** points to implementation-specific **`Servlet`**; and
  - an **`init-param`** identifies the **`Application`** subclass
- Resource classes and providers can access **`Servlet`** request, context, config and response via injection

JAX-RS 1.1

Integration with Java EE 6 – Servlets 3.0

- No or Portable “web.xml”

```
<web-app>
  <servlet>
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>

com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <init-param>
      <param-name>javax.ws.rs.Application</param-name>
      <param-value>com.foo.MyApplication</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/resources/*</url-pattern>
  </servlet-mapping>
</web-app>
```

```
@ApplicationPath(“resources”)
public class MyApplication
   extends
   javax.ws.rs.core.Application {
}
```

JAX-RS Summary

- Java API for building RESTful Web Services
- POJO based
- Annotation-driven
- Server-side API
- HTTP-centric

JAX-RS 1.1

Jersey Client-side API

- Consume HTTP-based RESTful Services
- Easy-to-use
  - Better than HttpURLConnection!
- Reuses JAX-RS API
  - Resources and URI are first-class citizens
- Not part of JAX-RS yet
  - com.sun.jersey.api.client

JAX-RS 1.1

Jersey Client API – Code Sample

```
Client client = Client.create();

WebResource resource = client.resource(“...”);

//curl  http://example.com/base
String s = resource.get(String.class);

//curl -HAccept:text/plain http://example.com/base
String s = resource.
     accept(“text/plain”).
     get(String.class);
```

http://blogs.oracle.com/enterprisetechtips/entry/consuming_restful_web_services_with

JAX-RS 1.1

WADL Representation of Resources

- Machine processable description of HTTP-based Applications
- Generated OOTB for the application

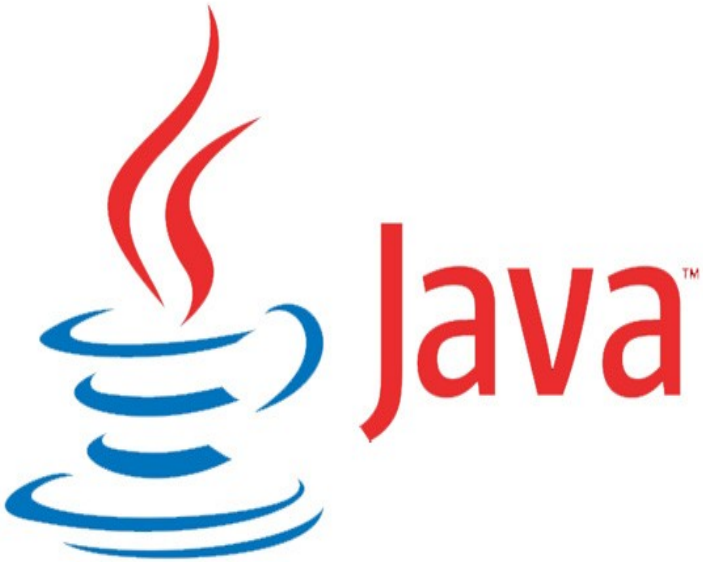
```
<application xmlns="http://research.sun.com/wadl/2006/10">
  <doc xmlns:jersey="http://jersey.dev.java.net/"
       jersey:generatedBy="Jersey: 1.1.4.1 11/24/2009 01:37
AM"/>
  <resources base="http://localhost:8080/HelloWADL/resources/">

    <resource path="generic">
      <method id="getText" name="GET">
        <response>
          <representation mediaType="text/plain"/>
        </response>
      </method>
      <method id="putText" name="PUT">
        <request>
          <representation mediaType="text/plain"/>
        </request>
      </method>
    </resource>
  </resources>
</application>
```

References

- oracle.com/javaee
- download.oracle.com/javaee/6/tutorial/doc/
- jersey.java.net
- jax-ws.java.net/

Demo



Thank You

Developing RESTful Web services with JAX-RS

Sabyasachi Ghosh, Senior Application Engineer
Oracle India, @neilghosh