Developing RESTful Web services with JAX-RS

Arun Gupta, Java EE & GlassFish Guy
blogs.oracle.com/arungupta, @arungupta

REST is an Architectural Style

Style of software architecture for distributed hypermedia systems such as World Wide Web

RESTful Web Services

Application of REST architectural style to services that utilize Web standards (URIs, HTTP, HTML, XML, Atom, RDF etc.)

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

Principles of REST

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

Give Everything an ID

- 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")
  Customer getCustomer(@PathParam(“order_id”)int id {
    …
  }
}
```

Standard Set of Methods

Method	Purpose	Annotation
GET	Read, possibly cached	@GET
POST	Update or create without a known ID	@POST
PUT	Update or create with a known ID	@PUT
DELETE	Remove	@DELETE
HEAD	GET with no response	@HEAD
OPTIONS	Supported methods	@OPTIONS

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) {...}

}
```

Binding Request to Resource

Annotation	Sample
@PathParam	Binds the value from URI, e.g. @PathParam("id")
@QueryParam	Binds the value of query name/value, e.g. @QueryParam("name")
@CookieParam	Binds the value of a cookie, e.g. @CookieParam("JSESSIONID")
@HeaderParam	Binds the value of a HTTP header , e.g. @HeaderParam("Accept")
@FormParam	Binds the value of an HTML form, e.g. @FormParam("name")
@MatrixParam	Binds the value of a matrix parameter, e.g. @MatrixParam("name")

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
    ```

Resource Representations

- Representation format identified by media type. E.g.:
  - XML - application/properties+xml
  - JSON - application/properties+json
  - (X)HTML+microformats - application/xhtml+xml
- JAX-RS automates content negotiation

  -
    ```
    GET /foo
    Accept: application/properties+json
    ```

Multiple Representations

- Static - Annotate methods or classes with static capabilities
  - `@Produces`, `@Consumes`
- Dynamic - Use `Variant`, `VariantListBuilder` and `Request.selectVariant` for dynamic capabilities

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) {
  ...
}
```

ORACLE

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) {
          . . .
    }
}
```

JAX-RS 1.1
Code Sample

```
import javax.inject.Inject;
import javax.enterprise.context.RequestScoped;

@RequestScoped
public class ActorResource {
    @Inject DatbaseBean db;

    public Actor getActor(int id) {
        return db.findActorById(id);
    }
}
```

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);
    }
}
```

http://blogs.oracle.com/arungupta/entry/totd_124_using_cdi_jpa

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);
```

Stateless Communications

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

JAX-RS 1.1

More Code Samples

- Processing POSTed HTML Form

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

- Sub-Resources

```
@Singleton
@Path("/printers")
public class PrintersResource {

  @GET @Path("/list")
  @Produces({"application/json", "application/xml"})
  public WebResourceList getListOfPrinters() { ... }

  @GET @Path("/ids/{printerid}")
  @Produces({"application/json", "application/xml"})
  public Printer getPrinter(@PathParam("printerid") String printerId) { ...
}
```

ORACLE

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 {
}
```

Mapping Application Exceptions

- Map checked or runtime exception to `Response`

```
public class OrderNotFoundException extends RuntimeException {

  public OrderNotFoundException(int id)
    super(id + " order not found");
  }

}
```

```
@Path("{id}")
public Order getOrder(@PathParam("id")int id)
  Order order = null;
  . . .
  return order;
  if (order == null)
    throw new OrderNotFoundException(id);
}
```

ORACLE

Mapping Application Exceptions

```
@Provider
public class OrderNotFoundExceptionMapper implements
ExceptionMapper<OrderNotFoundException> {

  @Override
  public Response toResponse(OrderNotFoundException exception)
{
    return Response
        .status(Response.Status.PRECONDITION_FAILED)
        .entity("Response not found")
        .build();
  }

}
```

ORACLE

JAX-RS Summary

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

JAX-RS 1.1
Integration with Java EE 6 – EJB 3.1

- Use stateless or singleton EJBs in the WAR as resource and provider classes

```
@Path(“stateless”)
@Stateless
public class MyStatelessRootResource {
  @Context UriInfo ui;

  @GET
  public String get() { return “GET”; }

  @EJB MyStatelessResource subResource;

  @Path(“sub-resource”)
  public MyStatelessResource sub() {
    return subResource;
  }
}
```

```
@Singleton
public class MyStatelessResource {
     @Context UriInfo ui;

   …
}
```

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

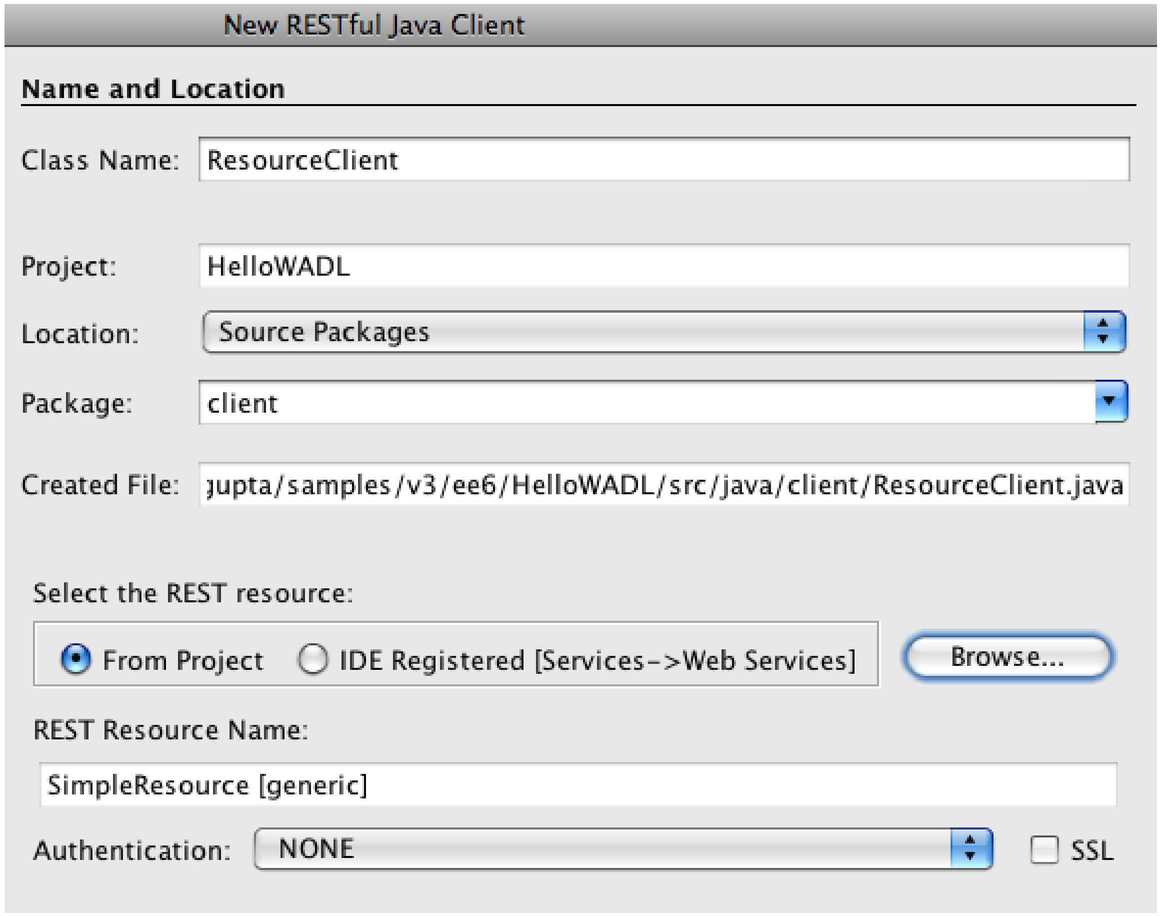
Jersey Client API – Code Sample

```
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

Jersey Client API – NetBeans Code Generation



New RESTful Java Client

Name and Location

Class Name:	ResourceClient
Project:	HelloWADL
Location:	Source Packages
Package:	client
Created File:	jupta/samples/v3/ee6/HelloWADL/src/java/client/ResourceClient.java

Select the REST resource:

[✓ From Project] IDE Registered [Services->Web Services] Browse...

REST Resource Name:

SimpleResource [generic]

Authentication: NONE ☐ SSL

ORACLE

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>
```

Java SE Deployment

- **`RuntimeDelegate`** is used to create instances of a desired endpoint class
- Application supplies configuration information
  - List of resource classes and providers as subclass of **`Application`**
- Implementations can support any Java type
  - Jersey supports Grizzly (see below), LW HTTP server and JAX-WS Provider

Example Java SE Deployment

```
Application app = ...
RuntimeDelegate rd = RuntimeDelegate.getInstance();
Adapter a = rd.createEndpoint(app, Adapter.class);

SelectorThread st = GrizzlyServerFactory.create(
    "http://127.0.0.1:8084/", a);
```

ORACLE

Servlet

- 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 status

- JAX-RS 1.0: 18th October 2008
- JAX-RS 1.1: 23rd November 2009
  - Aligned with Java EE 6, but not in the Web profile!
- JAX-RS 2.0: **`Future<?>`**
  - 2.0 M3 Available
- Implementations
  - Apache CXF, Apache Wink, eXo, Jersey, RESTEasy, Restlet, Triaxrs

GET /Samples

- Many samples are provided with the release
  - Atom, JAXB, JSON, Scala, Spring, WADL
  - Using GlassFish (+embedded) and Grizzly
- Download the 1.1.0 samples
  - Samples are maven-based
  - Works with NetBeans 6.x + maven plugin
  - Individual sample zip files are also available
    - e.g. Sparklines, Mandel

JAX-RS 2.0 (JSR 339)

http://jcp.org/en/jsr/detail?id=339
http://jax-rs-spec.java.net

- Client API
  - Low level using Builder pattern, Higher-level
- Hypermedia
- ~~MVC Pattern~~
  - ~~Resource controllers, Pluggable viewing technology~~
- Bean Validation
  - Form or Query parameter validation
- Closer integration with @Inject, etc.
- Server-side asynchronous request processing
- Server-side content negotiation

References

- oracle.com/javaee
- glassfish.org
- oracle.com/goto/glassfish
- blogs.oracle.com/theaquarium
- youtube.com/GlassFishVideos
- Follow @glassfish

ORACLE

Developing RESTful Web services with JAX-RS

Arun Gupta, Java EE & GlassFish Guy
blogs.oracle.com/arungupta, @arungupta