



J-Spring

16 april 2008 Spant! - Bussum



# JSR-311: JAX-RS

## Building RESTful Web Services in Java

Mario Klaver  
([mario.klaver@endpoint.nl](mailto:mario.klaver@endpoint.nl))





## Who is Mario Klaver?

- Software Architect at Endpoint ICT ([www.endpoint.nl](http://www.endpoint.nl)).
- Over 11 years of experience.
- Involved in knowledge network Know-IT
- Strong focus on application integration and Service Oriented Architectures, but also
- REST Believer



## Flamewar SOAP vs REST

“Web services are too complex”

“How do you deal with things  
like transactions, reliability, etc?”

“Web services are reinventing  
the Web through HTTP POST”

“Foolish to model everything as GET,  
POST, PUT and DELETE operations”

“Losing benefits of the Web”



## Contents

- Why SOAP is evil
- REST comes to the rescue!
  - REST concepts
  - REST in Java...the hard way
- REST in Java...the JAX-RS way
  - Explaining JAX-RS
  - What is not covered
- REST pro's and con's
  - SOAP and REST a happy marriage?
- Conclusion
- Q&A



## Why SOAP is evil - Example

- Issue comes from example 1 in the SOAP 1.1 specification:

```
POST /StockQuote HTTP/1.1
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope xmlns:SOAP-ENV="...">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



## Why SOAP is evil - Overview

- Goes against the Web architecture: simple information retrieval is done with GET
  - <http://stockquoteserver.example/query?symbol=DIS>
- One URI for every message
  - Not bookmarkable
- Potentially performance issues
  - Not cacheable
  - Potentially big payload



## REST comes to the rescue!

- **RE**presentational **S**tate **T**ransfer
- Architectural style for distributed hypermedia systems, introduced by Roy Fielding (2000)
  - Client-Server
  - Stateless communication
  - Cacheability
  - Uniform interface for resources
  - Layered System
  - Code-on-demand (optional)



## REST Concepts

- Resources (nouns)
  - Identified by a URI, For example:
    - <http://www.parts-depot.com/parts>
  - Connectedness
- Uniform interface (verbs)
  - Small fixed set:
    - Create, Read, Update, Delete
- State Representations
  - data and state transferred between client and server
    - XML, JSON, Atom, XHTML, ...



# HTTP Example

## Request

```
GET /music/artists/beatles/recordings HTTP/1.1  
Host: media.example.com  
Accept: application/xml
```

Method

Resource

## Response

```
HTTP/1.1 200 OK  
Date: Tue, 08 May 2007 16:41:58 GMT  
Server: Apache/1.3.6  
Content-Type: application/xml; charset=UTF-8
```

State  
transfer

```
<?xml version="1.0"?>  
<recordings xmlns="...">  
  <recording>...</recording>  
  ...  
</recordings>
```

Representation



## Examples on the web

- Google AJAX Search API
  - <http://code.google.com/apis/ajaxsearch/>
- Amazon S3
  - <http://aws.amazon.com/s3>
- Services exposing Atom Publishing Protocol or GData
  - i.e. Google apps like Google Calendar
- Accidentally RESTful
  - Flickr, Del.icio.us API



## Uniform interface

<http://www.records.com/music/artists/beatles/recordings>

- Create POST/ PUT
- Read GET
- Update PUT
- Delete DELETE
  
- Besides these operations, we can use
  - HEAD
  - OPTIONS



## URI Templates

- Music Collection
  - **/music/artists/** - container for artist
  - **/music/artists/{id}** - 1 artist
  - **/music/artists/{id}/recordings**
- URI Templates are URIs with variables within the URI syntax.
- Records can be returned in any standard web format



# REST in Java...the hard way

```
public class Artist extends HttpServlet {  
    public enum SupportedOutputFormat {XML, JSON};  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        String accept = request.getHeader("accept").toLowerCase();  
        String acceptableTypes[] = accept.split(",");  
        SupportedOutputFormat outputType = null;  
        for (String acceptableType: acceptableTypes)  
            if (acceptableType.contains("/*/*") || acceptableType.contains("application/*") ||  
                acceptableType.contains("application/*+json"))  
                outputType=SupportedOutputFormat.XML;  
            break;  
        } else if (acceptableType.contains("application/*+json"))  
            outputType=SupportedOutputFormat.JSON;  
            break;  
        }  
    }  
    if (outputType==null)  
        response.sendError(406);  
    String path = request.getPathInfo();  
    String pathSegment[] = path.split("/");  
    String artist = pathSegments[1];  
    if (pathSegments.length < 2 || pathSegments.length > 2)  
        response.sendError(404);  
    else if (pathSegments.length == 2 && pathSegment[1].equals("recordings")) {  
        if (outputType == SupportedOutputFormat.XML)  
            writeRecordingsForArtistAsXml(response, artist);  
        else  
            writeRecordingsForArtistAsJson(response, artist);  
    } else  
        if (outputType == SupportedOutputFormat.XML)  
            writeArtistAsXml(response, artist);  
        else  
            writeArtistAsJson(response, artist);  
    }  
  
    private void writeRecordingsForArtistAsXml(HttpServletRequest response, String artist) { ... }  
    private void writeRecordingsForArtistAsJson(HttpServletRequest response, String artist) { ... }  
    private void writeArtistAsXml(HttpServletRequest response, String artist) { ... }  
    private void writeArtistAsJson(HttpServletRequest response, String artist) { ... }  
}
```



## How did we do it with JAX-WS?

```
@WebService(serviceName = "ArtistService")
public interface ArtistServiceWS {
    @WebMethod @WebResult(name="Artist")
    public Collection<String> getAllArtists();

    @WebMethod @WebResult(name="Artist")
    public Artist getArtist(@WebParam(name="name")
        String name);

    @WebMethod @WebResult(name="Record")
    public Collection<Record> getRecords(
        @WebParam(name="name") String name);
}
```

- So is this where the REST euphoria stops?



## REST Framework alternatives

- Restlet (opensource client and server API)
  - <http://www.restlet.org/>
- CXF
  - HttpBinding
  - JAX-WS Provider/Dispatch API
- Axis2
  - HttpBinding (WSDL 2.0)
- JAX-RS



## REST in Java...the JAX-RS way

- Standardized in the JCP
  - JSR 311
  - Will be included in Java EE 6
- Group started in April 2007
  - Marc Hadley, Paul Sandoz
- EG members
  - Alcatel-Lucent, BEA, Day Software, Fujitsu, innoQ, Nortel, Red Hat
  - Experts in Atom, AtomPub, WebDAV, HTTP, REST, Restlet



## JAX-RS Introduction

- Set of Java APIs for development of web services built according to the REST principals
- Annotations
- Status: In progress  
(Early Draft Review: 23 November 2007)



## Resource Methods (1)

- Method of a resource class annotated with `@HttpMethod`
- Standard implementation for `@GET`, `@POST`, `@PUT`, `@DELETE`, `@HEAD`
- Extendable
  - i.e. WebDav methods Move, Copy, Search
- Parameters can be annotated
  - Examples: `@PathParam`, `@CookieParam`, `@HeaderParam`,...



## Resource Methods (2)

- Return Types
  - Resource methods MAY return **void**, **Response** or another Java type
  - Conversion from Java type to entity body by an entity provider
- Exceptions
  - **WebApplicationException**



## Resource Methods - example

```
public interface ArtistResource {  
    @HttpMethod("GET")  
    public Artist getArtist();  
  
    @DELETE  
    public void deleteArtist();  
  
    @PUT  
    public void updateArtist(Artist artist);  
}
```



## Resources

- Resource classes are annotated with `@Path`
- URI is relative to deployment context
- Methods can also be annotated with `@Path`
  - Sub-resource methods
    - In combination with request method designator (`@HttpMethod`)
  - Sub-resource locators
    - returned object is treated as a resource class



## Resources - example

```
@Path("/music/artists")
public interface ArtistsResource {
    @POST
    public Response createArtist(Artist artist);

    @GET
    public ArtistList getAllArtists();

    @GET
    @Path("top40")
    public ArtistList getTop40();

    @Path("/{id}")
    public ArtistResource getUser(@PathParam("id")
        int id);
}
```



## Representations

- Supported requests and response media types can be declared by
  - `@ConsumeMime("application/xml")`
  - `@ProduceMime("application/json")`
- Annotations may be applied to
  - Resource classes
  - Resource methods
  - Entity provider for method arguments or return type



## Representations - example

```
@ProduceMime ({ "application/xml", "application/json" })  
public interface ArtistResource {  
  
    @GET  
    public Artist getArtist ();  
  
    @GET  
    @ProduceMime ("text/html")  
    public String getArtistAsHtml ();  
  
    ...  
}
```



## Entity Providers

- Used to map representations to their associated Java types
- Classes annotated by `@Provider`
- Implementation of interface
  - `MessageBodyReader<T>`
  - `MessageBodyWriter<T>`



## Entity Providers - example

```
@Provider
@ProduceMime("text/html")
public class ArtistHtmlProvider implements
    MessageBodyWriter<Artist> {

    public long getSize(Artist a) { ... }

    public boolean isWriteable(...) { ... }

    public void writeTo(...) throws IOException { ... }

}
```



## What is not covered in JAX-RS

- REST is an architecture, JAX-RS a possible implementation in Java
- JAX-RS is bound to HTTP
- No description, registration and discovery (You can use WADL or WSDL)
- Very little media types (representations)
- Connectedness, caching, partial-GET must be implemented by developer
- No client APIs



## REST – pro's and con's (1)

- Pro's:
  - Simplicity
  - Uniformity
  - Different representations
  - Cacheable
  - Scalable
  - Connectedness



## REST – pro's and con's (2)

- Con's:
  - Not transport protocol independent
    - JMS, HTTP, Email,...
  - No standards for 2-phase commit transactions
    - WS-AtomicTransaction
  - No standards for security except for https
    - Signatures, encryption



## SOAP 1.2 / WSDL 2.0 features supporting REST

- WSDL 2.0 is tailored for SOAP-based services, but
- HTTP binding allows for good description of HTTP services

```
<binding name="HttpBinding" interface="m:Artists" type="...">  
  <operation ref="m:getArtists" whttp:method="GET"  
    whttp:location="music/artists/" />  
  <operation ref="m:updateArtist" whttp:method="PUT"  
    whttp:location="music/artists/{id}" />  
</binding>
```

- Supports all HTTP methods



## Conclusion

- With JAX-RS, REST will even be a more serious alternative for SOAP.
- Only a few situations thinkable where SOAP is the only alternative.
- It is possible to use SOAP and REST together, but when using one, stick to it.



## Further reading

- R. Fielding's thesis
  - <http://roy.gbiv.com/pubs/dissertation/top.htm>
- REST Wiki
  - <http://rest.blueoxen.net/cgi-bin/wiki.pl>
- Wikipedia
  - [http://en.wikipedia.org/wiki/Representational\\_State\\_Transfer](http://en.wikipedia.org/wiki/Representational_State_Transfer)
- <http://jcp.org/en/jsr/detail?id=311>
- Stacks: Jersey (RI), CXF, Axis



.nl.  
jug



Q&A

