Developing RESTful Web Services using JAX-RS

Topics

- Goals of JAX-RS
- Address-ability
- Methods
- Representations (Formats)
- Returning status
- Statelessness
- Connectedness
- Status of JAX-RS
- Tooling

Goals of JAX-RS

Problem in Using Servlet API For Exposing a Resource (Too much coding)

```
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/xml")) {
                outputType=SupportedOutputFormat.XML;
                break;
            } else if (acceptableType.contains("application/json")) {
                outputType=SupportedOutputFormat.JSON;
                break;
            }
        }
        if (outputType==null)
            response.sendError(415);
        String path = request.getPathInfo();
        String pathSegments[] = path.split("/");
        String artist = pathSegments[1];
        if (pathSegments.length < 2 && pathSegments.length > 3)
            response.sendError(404);
        else if (pathSegments.length == 3 && pathSegments[2].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);
```

There Must Be a Better Way: Server Side API Wish List for Exposing a Resource

- High level and Declarative
  - Use @ annotation in POJOs
- Clear mapping to REST concepts
  - Address-ability through URI, HTTP methods
- Takes care of the boilerplate code
  - No need to write boilerplate code
- Graceful fallback to low-level APIs when required
  - Provides ease of development with flexibility for fine-tuning

Address-ability

Clear mapping to REST concepts: Address-ability

- A Web service exposes data as resources
- A resource is exposed through a URI
- JAX-RS:
  - Resources are “plain old” Java classes and methods
  - The annotation `@Path` exposes a resource
  - Think resources and URIs using Java classes and `@Path`

Clear mapping to REST concepts

- Resources: what are the URIs?

```
@Path("/employees/{id}")
```

- Design the resource URI
  - /employees – container for 'employees'
  - /employees/123456 – one 'employee'

http://www.sun.com/employees/123456
http://www.sun.com/employees/chuk

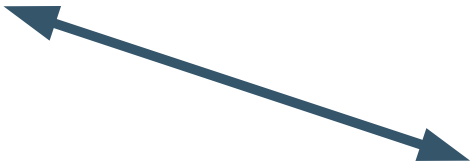
variable

Mapping URIs to Classes

```
@Path("/employees")
public class Employees {
        …
}
```

```
@Path("/employees/{id}")
public class Employee {

        public String getEmployee(@UriParam("id") int id) {
                …
        }
}
```



Methods

Clear mapping to REST concepts: Methods

- Methods: what are the HTTP methods?
- HTTP methods implemented as Java methods annotated with

```
@HEAD
@GET
@PUT
@DELETE
@POST
```

Uniform interface: methods on root resources

```
@Path("/employees")
class Employees {
   @GET <type> get() { ... }
   @POST <type> create(<type>) { ... }
}
```

```
@Path("/employees/{eid}")
class Employee {
   @GET <type> get(...) { ... }
   @PUT void update(...) { ... }
   @DELETE void delete(...) { ... }
}
```

Java method name is not significant
The HTTP method is the method

Representations (Formats)

Clear mapping to REST concepts: Formats

- Representations: what are the formats?

```
@Consumes("application/xml")
@Produces("application/json")
```

Formats in HTTP

Request

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

Format

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

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

State transfer

Representation

Multiple Representation

- Resources can have multiple representation
  - Specified through 'Content-type' HTTP header
  - Acceptable format through 'Accept' HTTP header
- A web page can be represented as
  - text/html – regular web page
  - application/xhtml+xml – in XML
  - application/rss+xml – as a RSS feed
  - application/octet-stream – an octet stream
  - application/rdf+xml – RDF format

Supported Media Types

- Think what media is consumed and produced...
- ...then think of the Java types associated
- “Out-of-the-box” support for the following
  - ***/*** – byte[], InputStream, File, DataSource
  - **text/*** – String
  - **text/xml**, **application/xml**, – JAXBElement, Source
  - **application/x-www-form-urlencoded** – MultivalueMap<String, String>

Uniform Interface

Uniform interface: JAX-RS Consuming

- Specify input format with `@Consumes`
- Annotated method parameters extract client request information
  - `@UriParam` extracts information from the URI
- Single un-annotated method parameter is the representation of the request
  - e.g. `String` or JAXB bean

Uniform interface: consuming

```
@Path("/employees/{eid}")
@ConsumeMime("application/xml")
class Employee{
   @GET
   <type> get(@UriParam("eid") String eid)
   { ... }
   @PUT
   <type> update(@UriParam("eid") String eid,
        Ent e)
   { ... }
   @DELETE
   <type> delete(@UriParam("eid") eid) { ... }
}
```

Uniform interface: JAX-RS producing

- A HTTP method classifies the response type with `@Produces`
- The method return type is the response
  - Java type that is the representation
  - Or `void` if no representation

Producing - annotate on methods

```
@Path("/employees")
class Employees {
  @GET @ProduceMime("application/xml")
  Col get() { ... }

  @POST
  @ConsumeMime("application/xml")
  @ProduceMime("application/xml")
  Ent create(Ent e) { ... }
}
```

Producing - annotate on the class

```
@Path("/employees")
@ConsumeMime("application/xml")
@ProduceMime("application/xml")
class Employees {
  @GET Col get() { ... }

  @POST Ent create(Ent e) { ... }
}
```

Uniform interface: JAX-RS producing

- Instance of **`Response`** may contain a Java type
- A response builder can produce arbitrary responses
  - e.g. created or redirected responses

Uniform interface: build a response

```
@Path("/employees")
@ConsumeMime("application/xml")
@ProduceMime("application/xml")
class Employees {
  @GET Col get() { ... }

  @POST Response create(Ent e) {
    // create and persist the new entry
    // create entry resource URI
    URI u = ...
    // build response and return
    return Response.created(u).build();
  }
}
```

Uniform interface: HTTP request and response

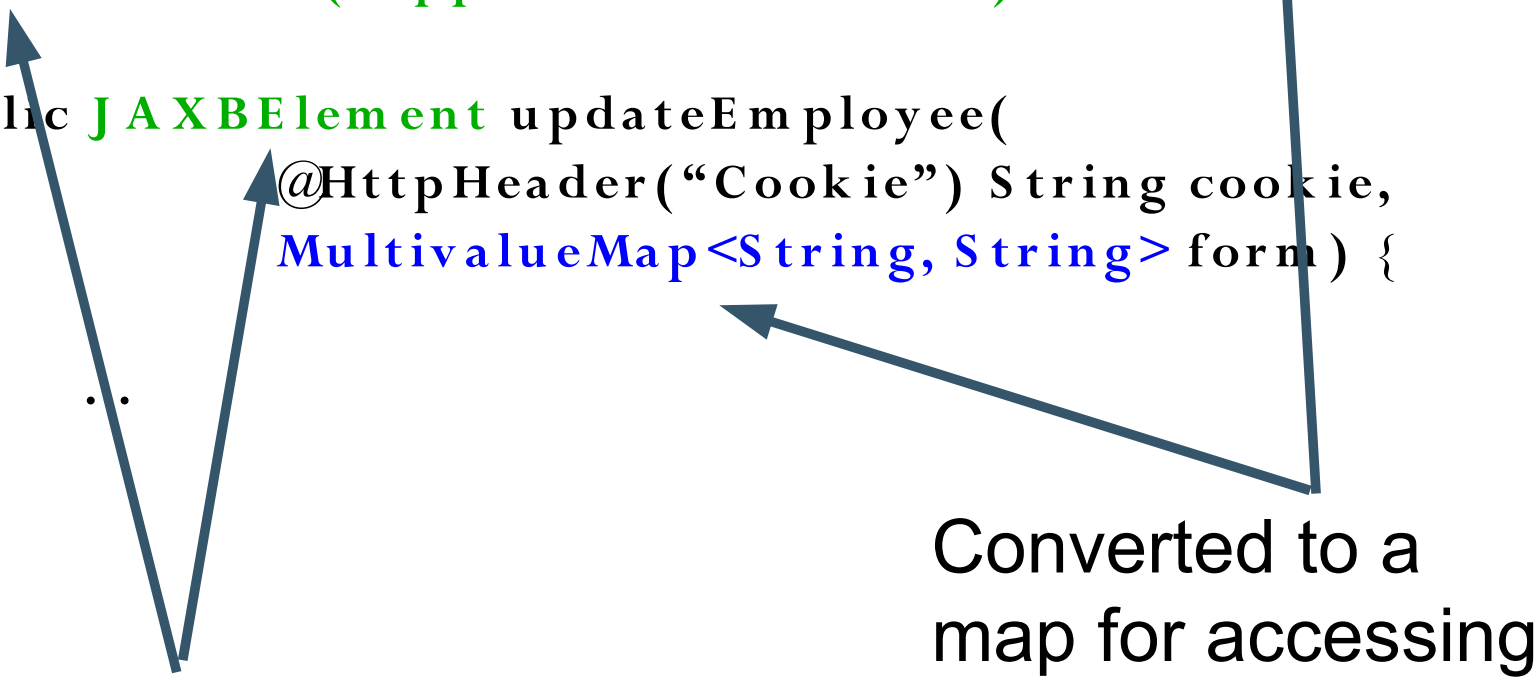
```
C: POST /employees HTTP/1.1
C: Host: host.com
C: Content-Type: application/xml
C: Content-Length: 35
C:
C: <employee><name>carol</name></employee>

S: HTTP/1.1 201 Created
S: Location: http://host.com/employees/1234
S: Content-Length: 0
```

Working with Media Types

```
@Post
@ConsumeMime("application/x-www-form-urlencoded")
@ProduceMime("application/rss+xml")

public JAXBElement updateEmployee(
                @HttpHeader("Cookie") String cookie,
                MultivalueMap<String, String> form) {
    ...
```



Serialized to a XML stream

Converted to a map for accessing form's field

Distinct URI for content

```
@Get @ProduceMime("application/xml")
@Path("/employees/xml")
public JAXBElement getXML() {
                ...
@Get @ProduceMime("application/json")
@Path("/employees/json")
public JSONObject getJSON() {
                ...
```

Example adapted from http://blogs.sun.com/sandoz/entry/philosphical_content_negotiation

Clear mapping to REST concepts

- Resources: what are the URIs?

```
@Path("/artists/{id}")
```

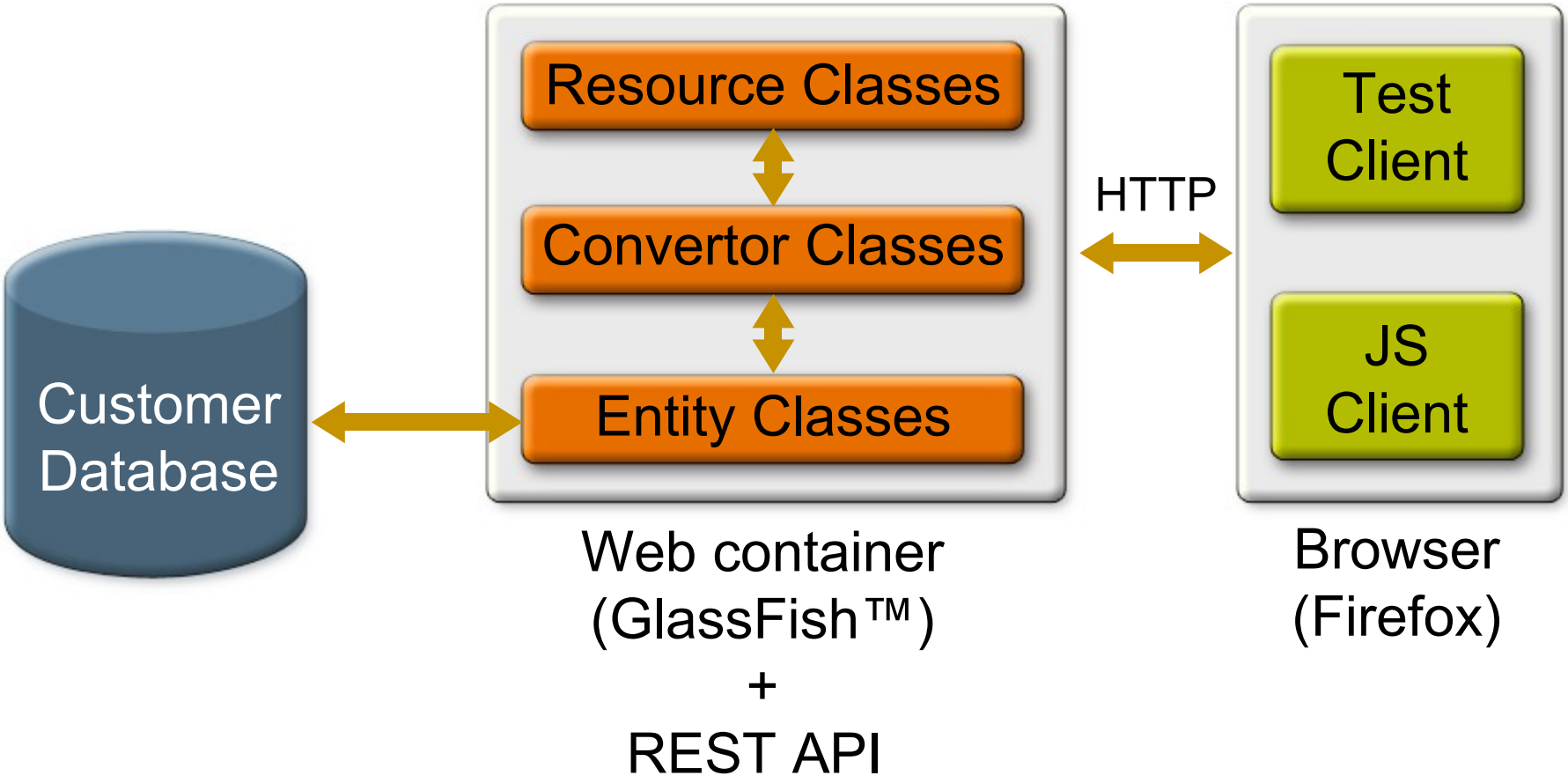
- Methods: what are the HTTP methods?

```
@GET
public XXX find()
```

- Representations: what are the formats?

```
@Consumes("application/xml")
@Produces("application/json")
```

Customer Service Overview



Returning Status

Returning Status Code

- Mainly for error conditions
- For example
  - 405 Method Not Allowed
  - 415 Unsupported Media Type
- See http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
- Status can be returned either with WebApplicationException or Response

Return Code Examples

```
@HttpMethod("GET")
public Employee getEmployee(
                @UriParam("id") int id) {
        if (!doesEmployeeExist(id))
                throw new WebApplicationException(410);
```

- Use it when the method has a return type
- Simple error indicator

```
@HttpMethod("GET")
public Response getEmployee(
                @UriParam("id") int id) {
        if (!doesEmployeeExist(id))
                return (Response.Builder.status(410).build());
```

- Response is used to construct more sophisticated return result
- Can build responses for redirects, errors, ok, setting headers, etc

Statelessness

Statelessness

- HTTP protocol is stateless
  - Service should not store session from previous requests
  - Eliminates many failure conditions
- States of Web service are resources
- Client responsible for application state
- Service responsible for resource state

Sessions are Irrelevant

- REST is the transfer of states
- Simple, visible, reusable, cacheable
- Eg. Booking travel
  - Create itinerary resource, fill itinerary, post itinerary
  - All held on client as **not** on **server session**

Statelessness: JAX-RS

- Default per-request life-cycle for root resource classes
  - A new instance created for every request
  - Constructor/fields used like plain old Java objects
  - Reduces concurrency issues

Statelessness: per-request life-cycle

```
@Path("/employees/{eid}")
@ConsumeMime("application/xml")
@ProduceMime("application/xml")
class Employee {
  String eid;
  EntryResource(@UriParam("eid") String eid)
  { this.eid = eid; }

  @GET Ent get(){ ... }

  @PUT void update(Ent e) { ... }

  @DELETE void delete() { ... }
}
```

Statelessness: constructor can check for errors

```
@Path("/collection/{eid}")
@ConsumeMime("application/xml")
@ProduceMime("application/xml")
class EntryResource {
  String eid;
  EntryResource(@UriParam("eid") String eid) {
    this.eid = eid;
    if ("eid does not exist")
      // Not found
      throw new WebApplicationException(404);
  }

}
```

Statelessness: JAX-RS guiding principles

- HTTP session life-cycle is not supported
- Developer must model state
  - As resources; and
  - As application state in the representations

Connectedness

Connectedness

- Representations contain links to other resources
- Put a service in different states by following links and filling in forms
  - "Hypermedia as the engine of application state"
  - HTML, XML, JSON, N3 can all be Hypermedia
- Client is in charge
  - Service offers potential states
  - Service does not impose state transition order

Connectedness

- Representations contain links to other resources
- Example
  - Eg. http://blogs.sun.com/chuk may return a JSON object with pointers to my other blog resource
    - http://blogs.sun.com/chuk/01102007/0 - first blog for 01102007
    - http://blogs.sun.com/chuk/2007/january - all blogs for January

From http://www.prescod.net/rest/mistakes/

JAX-RS

RESTful Web Services API: JAX-RS

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

JAX-RS = Easy REST Way

- JAX-RS Reference Implementation: Jersey
  - Open Source
  - http://jersey.dev.java.net
  - Also Stable builds at GlassFish Update Center

Jersey, the Reference Implementation and more...

- Java.net project
  - CDDL license, will be duel licensed with GPL
  - Also available as NetBeans 6 and GlassFish plugins
- New stable release ~every 6 weeks
  - On java.net maven repository
- Cooperation and contributions welcome!
  - A number of external (outside of Sun) contributors

Jersey features

- Runtime discovery of resource classes
  - No tooling pre-compile step
- Runtime generation of WADL
- API and SPI for HTTP containers
  - Support for HTTP server, Grizzly, JAX-WS Provider, Servlet
  - Easy to plug-in your own
- SPI for IoC frameworks
  - Examples using Spring and Guice

Jersey planned features

- Improved plugable Java type support
  - Work with IoC frameworks
  - Register dynamically
- Model View Controller
  - HTTP method returns a model
  - SPI for template engines
    - Freemarker, Velocity
- Improved JSON support
  - Using JAXB

Status and Schedule

- Early draft review completed
  - EG making good progress
- Stable 0.5 JAX-RS API and Jersey made available on Jan 18th 2008
- Will be standard component of JavaEE 6
- Plan to finalize end of Q3 2008
  - Dependencies on other specifications could impact schedule

Tooling

RESTful Tooling

- Rapid RESTful application development
  - Generate, deploy, test, edit, deploy, test, edit, ...
- Generate Web service from database
  - Expose existing data onto the Web in a few clicks
  - Container/item resource pattern maps to DB tables
- Generate Client artifacts from Web service
  - For client applications or for testing

Summary

Summary

- JAX-RS guides
  - Clearer division of responsibility
- JAX-RS makes it easier
  - Performs common RESTful tasks
- JAX-RS and Jersey available now

Summary

- REST architecture is gaining popularity
  - Simple, scalable and the infrastructure is already in place
- JAX-RS (JSR-311) provides a high level declarative programming model
  - http://jersey.dev.java.net
- NetBeans provices necessary tool

Resources

- JAX-RS (JSR-311)
  - https://jsr311.dev.java.net/
- Jersey
  - http://jersey.dev.java.net
  - Downloads
    - https://jersey.dev.java.net/servlets/ProjectDocumentList
    - http://download.java.net/maven/1/javax.ws.rs/
    - http://download.java.net/maven/1/jersey/
  - Schedule
    - http://wikis.sun.com/display/Jersey/Schedule

Project GlassFish

Building a Java EE 5 Open Source Application Server

- Simplifying Java application Development with **Java EE 5 technologies**
- Includes JWSDP, EJB 3.0, JSF 1.2, JAX-WS and JAX-B 2.0
- Supports > **20** frameworks and apps
- Basis for the **Sun Java System Application Server PE 9**
- **Free** to download and **free** to deploy
- Over **1200** members and **200,000** downloads
- Integrated with **NetBeans**

java.sun.com/javaee/GlassFish

Source: Sun 2/06—See website for latest stats

Developing RESTful Web Services using JAX-RS