Advanced Java API for RESTful Web Services (JAX-RS)

Jakub Podlešák
Jersey Tech Lead, Oracle

Agenda

- The current status of JAX-RS
- JAX-RS basics in one slide
- Selected topics
- The future of JAX-RS
- Q&A

What this Presentation is Not About

- The REST style per se
- The basics of JAX-RS
  - Bootstrap slide for those new to JAX-RS

The Current Status of JAX-RS, The Java API for RESTful Web Services

- JSR 311: JAX-RS 1.1 released 23rd Nov 2009
- Part of Java EE 6; Not part of the Web profile!
- Specifies integration with:
  - CDI 1.0
  - EJB 3.1
  - Servlet 3.0
- Jersey 1.8 shipped with GlassFish 3.1.1
- 7 implementations (Apache CXF, Apache Wink, eXo, Jersey, RESTEasy, Restlet, Triaxrs)

JAX-RS Basics in One Slide

```
@Path("widgets")
@Produces("application/xml, application/json")
public class WidgetsResource {

    @GET
    public WidgetsRepresentation getWidgetList() {
    ...
    }

    @GET @Path("{id}")
    public WidgetRespresentation getWidget(PathParam("id") String widgetId) {
    ...
    }
}
```

Selected Topics

- Runtime resource resolution
- Runtime content negotiation
- Conditional HTTP requests
- Dealing with type erasure
- Pluggable exception handling

goto; conference prague

Runtime Resource Resolution

Runtime Resource Resolution

- Root resources can declare sub-resources that will match the unmatched part of the URI path
- Root resources implement **sub-resource locator** methods

```
@Path("foo")
public class Foo {
    @Path(“bar”)  public Bar getBar() { return new Bar(); }
}
public class Bar {
    @Path("baz") @GET public String get() { … }
}
```

Runtime Resource Resolution

GET http://example.com/foo/bar/baz HTTP/1.1

```
@Path("foo")
public class Foo {
    @Path(“bar”)  public Bar getBar() { return new Bar(); }
}
public class Bar {
    @Path("baz") @GET public String get() { … }
}
```

Runtime Resource Resolution

GET http://example.com/foo/bar/baz HTTP/1.1

```
@Path("foo")
public class Foo {
    @Path(“bar”)  public Bar getBar() { return new Bar(); }
}
public class Bar {
    @Path("baz") @GET public String get() { … }
}
```

Runtime Resource Resolution

GET http://example.com/foo/bar/baz HTTP/1.1

```
@Path("foo")
public class Foo {
    @Path("bar")  public Bar getBar() { return new Bar(); }
}
public class Bar {
    @Path("baz") @GET public String get() { … }
}
```

Runtime Resource Resolution

GET http://example.com/foo/bar/**baz** HTTP/1.1

```
@Path("foo")
public class Foo {
    @Path(“bar”)  public Bar getBar() { return new Bar(); }
}
public class Bar {
    @Path("baz") @GET public String get() { … }
}
```

Runtime Resource Resolution

GET http://example.com/foo/bar/baz HTTP/1.1

```
@Path("foo")
public class Foo {
    @Path(“bar”)  public Bar getBar() { return new Bar(); }
}
public class Bar {
    @Path("baz") @GET public String get() { … }
}
```

Runtime Resource Resolution

- Sub-resource classes are processed at runtime

```
@Path("sub")
public Object getSubResource() {
    if ( /bar is preferred/)  return new Bar();
            else return new Baz();
}
```

- The sub-resource locator method controls life-cycle of sub-resource
  - Can combine with injection for use with EJBs or CDI-managed beans
- Warning: easy to confuse sub-resource methods with sub-resource locators

goto; conference prague

Runtime Content Negotiation

Runtime Content Negotiation

- Static content/media type negotiation of representation supported with **@Produces**
- Runtime content negotiation of representation supports 4 dimensions
  - Media type, character set, language, encoding
- Each representation has a Variant which is a point in the 4 dimensional space

Runtime Content Negotiation

```
List<Variant> vs = Variant.
    mediaTypes("application/xml", "application/json").
    languages("en", "cs").build()

assert va.size() == 4;
// [["application/xml", "en"] ["application/xml", "cs"]
// ["application/json", "en"] ["application/json", "cs"]]
```

- **List<Variant>** may be pre-calculated
- **Variant** or its extensions may be instantiated directly

Runtime Content Negotiation

```
@GET public Response get(@Context Request r) {
    List<Variant> vs = ...
    Variant v = r.selectVariant(vs);
    if (v == null) {
        return Response.notAcceptable(vs).build();
    } else {
        Object rep = selectRepresentation(v);
        return Response.ok(rep, v);
    }
}
```

- Selection will compare the list of variants with the correspond acceptable values in the client request
  - Accept, Accept-Language, Accept-Encoding, Accept-Charset

goto; conference prague

Conditional HTTP Requests

Conditional HTTP Requests

- Save bandwidth and client processing
  - A GET response can return 304 (Not Modified) if the representation has not changed since the previous request
- Avoid lost update problem
  - A PUT response can return 412 (Precondition Failed) if the resource state has been modified since the previous request
- A date and/or entity tag can be used
  - **Last-modified** and **Etag** headers
  - HTTP dates have a granularity of 1 second; entity tags are better for use with PUT

Conditional HTTP Request

```
@GET
public Response get(@Context Request r) {
    EntityTag et = ...
    Response.ResponseBuilder rb = r.evaluatePreconditions(et);
    if (rb != null) {
        // Return 304 (Not Modified)
        return rb.build();
    }
    String rep = ...
    return Response.ok(rep).tag(et).build();
}
```

goto; conference prague

Dealing with Type Erasure

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

How many of you think you understand Java Generics?

Dealing with Type Erasure

If you think you understand Java Generics you probably do not understand Java Generics.

Dealing with Type Erasure

Well the JSR-311 expert group thought they did...

Dealing with Type Erasure

- Resource method can return an entity

```
@GET public List<Bean> getBeanList() { ...}
```

- Type information is lost when returning **Response**

```
@GET
public Response getBeanList() {
  List<Bean> list = …
  return Response.ok(list).build();
}
```

- **MessageBodyWriter** supporting **List<Bean>** will not work when type information is lost

Dealing with Type Erasure

- Use **GenericEntity** to preserve type information at runtime

```
@GET
public Response getBeanList() {

  List<Bean> list = …
  GenericEntity<List<Bean>> ge =
                              new   GenericEntity<List<Bean>>(list) {};

  return Response.ok(ge).build();
}
```

goto; conference prague

Pluggable Exception Handling

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

Pluggable Exception Handling

- Propagation of unmapped exceptions to the Web container
  - A runtime exception thrown by the JAX-RS implementation or application is propagated as is
  - A checked exception thrown by the application is propagated as the cause of a ServletException
  - Propagated exceptions can be mapped to error pages
- Runtime/checked exceptions can be “caught” and mapped to a **Response** using **ExceptionMapper**

Pluggable Exception Handling

```
class A extends RuntimeException { …}

class B extends A { … }

class C extends B { … }
```

Pluggable Exception Handling

```
@Provider
public class AMapper  implements
ExceptionMapper<A> {
    public Response toResponse(A a) { ... }
}

@Provider
public class BMapper  implements
ExceptionMapper<B> {
    public Response toResponse(B b) { ... }
}
```

Pluggable Exception Handling

```
// Throwing A maps to Response of AMapper
@GET public String a() throws A { … }

// Throwing B maps to Response of BMapper
@GET public String a() throws B { … }

// Throwing C maps to Response of BMapper
@GET public String a() throws C { … }
```

Pluggable Exception Handling

- **ExceptionMapper** “closest” to exception class is selected to map exception
- Can map **Throwable**

```
@Provider public class CatchAll implements
ExceptionMapper<Throwable> {
    public Response toResponse(Throwable t) {
        // Internal Server Error
        return Response.status(500).build();
    }
}
```

- Inject **Providers** to delegate

The Future of JAX-RS

The Future of JAX-RS

- JSR-339: JAX-RS 2.0
- Expert group formed in February 2011
- Early draft published on Oct 21
- Final release anticipated in Q2 2012

JAX-RS 2.0

- Client API
- Asynchronous processing
- Filters, Handlers
- Declarative hyperlinking
- Parameter Validation
- Improved integration with JSR-330
- Quality of source

goto; conference prague

Questions?

References

- http://jersey.java.net (Jersey project, JAX-RS 1.1 RI)
- http://jsr311.java.net (JAX-RS API)
- http://jax-rs-spec.java.net/ (JAX-RS 2.0 API)
- http://marek.potociar.net (JAX-RS 2.0 spec co-lead blog)
  - See the recent post on Devoxx there
- http://www.java.net/blogs/spericas
  - (Santiago Pericas-Geertsen, JAX-RS 2.0 spec co-lead)
- http://blogs.oracle.com/japod (my blog)