RESTEasy

Distributed peace of mind

Agenda

- Why REST?
- Writing RESTFul Web Services in Java
  - JAX-RS
- RESTEasy features
- RESTEasy Roadmap

Speaker's Qualifications

- RESTEasy project lead
  - Fully certified JAX-RS implementation
- JAX-RS JSR member
  - Also served on EE 5 and EJB 3.0 committees
- JBoss contributor since 2001
  - Clustering, EJB, AOP
- Published author
  - Books, articles

What are the goals of SOA?

SOA Goals

- Reusable
- Interoperable
- Evolvable
  - Versioning
- Governable
  - Standards
  - Architectural Guidelines and Constraints
  - Predictable
- Scalable
- Manageable

What system has these properties?

The Web!

What is REST?

- REpresentational State Transfer
  - PhD by Roy Fielding
- REST answers the questions of
  - Why is the Web so prevalent and ubiquitous?
  - What makes the Web scale?
  - How can I apply the architecture of the web to my applications?

What is REST?

- It can mean a simple, “lightweight”, distributed interfac over HTTP
- REST is really a set of architectural principles
  - Principles that make the Web unique
- REST isn’t protocol specific
  - But, usually REST == REST + HTTP
- A different way to look at writing Web Services
  - Many say it’s the anti-WS-*
- Rediscovery of HTTP

REST Architectural Principles

- Addressable Resources
- Representation Oriented
- Constrained interface
- Hypermedia and Link Driven
- Communicate statelessly

Why REST?

- HTTP is everywhere
- “Lightweight” interoperability
- “Lightweight” stack
- Evolvability
  - Link driven systems allow you to redirect easily
  - Content negotiation allows you to support old and new formats

JAX-RS

RESTFul Web Services in Java

JAX-RS

- JCP Specification
- Required part of Java EE 6
- Annotation Framework
- Allows you to map HTTP requests to Java method invocations

JAX-RS: GET /orders/3323


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

    @Path("/{order-id}")
    @GET
    @Produces("application/xml")
    Order getOrder(@PathParam("order-id") int id) {
      …
    }
}
```

JAX-RS Annotations

- @Path
  - Defines URI mappings and templates
- @Produces, @Consumes
  - What MIME types does the resource produce and consume
- @GET, @POST, @DELETE, @PUT, @HEAD
  - Identifies which HTTP method the Java method is interested in

JAX-RS Parameter Annotations

- @PathParam
  - Allows you to extract URI parameters/named URI template segments
- @QueryParam
  - Access to specific parameter URI query string
- @HeaderParam
  - Access to a specific HTTP Header
- @CookieParam
  - Access to a specific cookie value

JAX-RS: GET /orders/3323

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @Produces("application/xml")
    Order getOrder(@PathParam("order-id") int id) {
      …
    }
}
```

Base URI path to resource

JAX-RS: GET /orders/3323

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @ProduceMime("application/xml")
    Order getOrder(@PathParam("order-id") int id) {
      …
    }
}
```

Additional URI pattern that getOrder() method maps to

JAX-RS: GET /orders/3323

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @Produces("application/xml")
    Order getOrder(@PathParam("order-id") int id) {
      …
    }
}
```

Defines a URI path segment pattern

JAX-RS: GET /orders/3323

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @Produces("application/xml")
    Order getOrder(@PathParam("order-id") int id) {
      …
    }
}
```

HTTP method Java getOrder() maps to

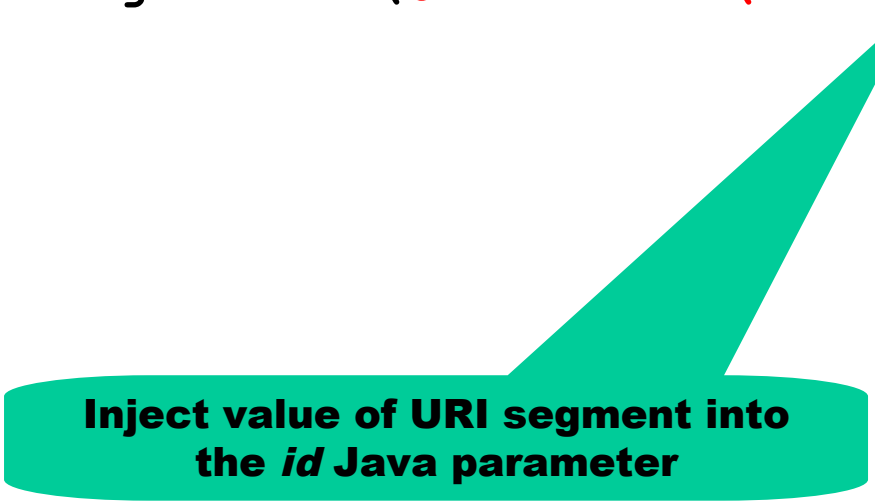
JAX-RS: GET /orders/3323

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @Produces("application/xml")
    Orer getOrder(@PathParam("order-id") int id) {
      …
    }
}
```

What's the CONTENT-TYPE returned?

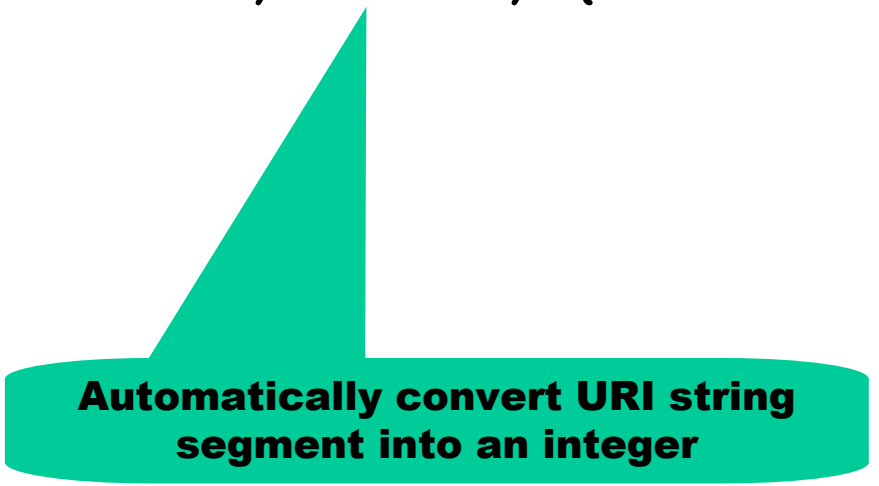
JAX-RS: GET /orders/3323

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @Produces("application/xml")
    Order getOrder(@PathParam("order-id") int id) {
      …
    }
}
```



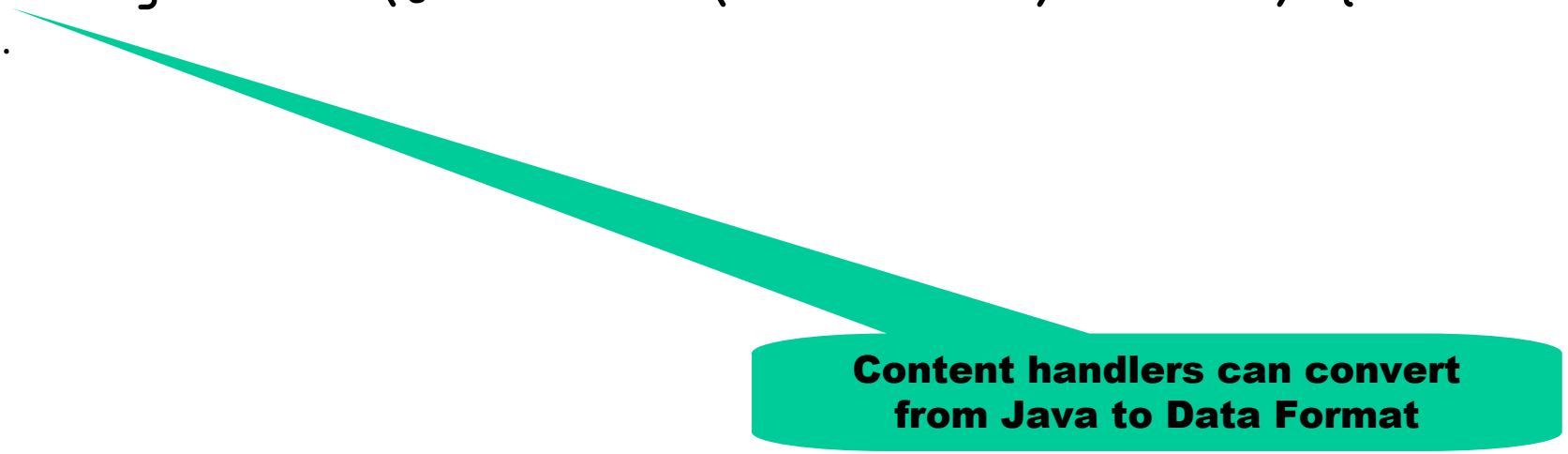
JAX-RS: GET /orders/3323

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @Produces("application/xml")
    Order getOrder(@PathParam("order-id") int id) {
      …
    }
}
```



JAX-RS: GET /orders/3323

```
@Path("/orders")
public class OrderService {

    @Path("/{order-id}")
    @GET
    @Produces("application/xml")
    Order getOrder(@PathParam("order-id") int id) {
      …
    }
}
```



JAX-RS: POST /orders


```
@Path("/orders")
public class OrderService {

    @POST
    @Consumes("application/xml")
    void submitOrder(Order orderXml) {
      …
    }
}
```

What CONTENT-TYPE is this method expecting from client?

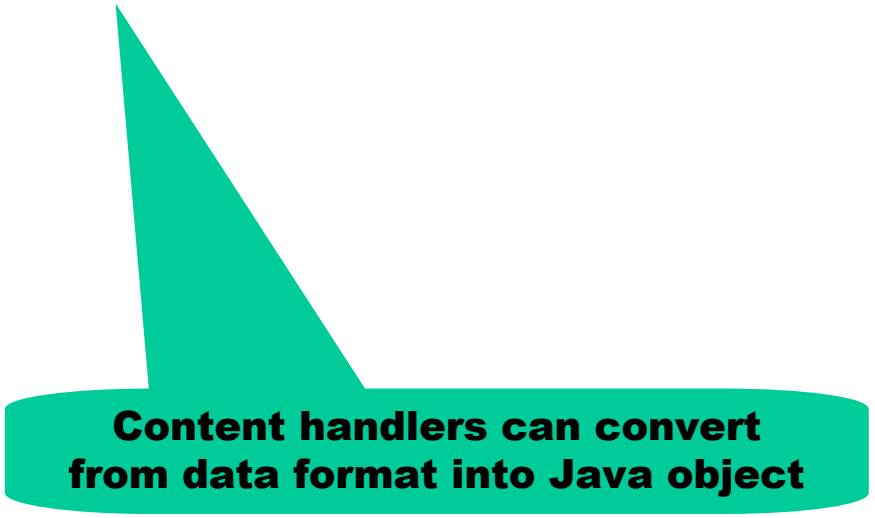
JAX-RS: POST /orders

```
@Path("/orders")
public class OrderService {

    @POST
    @Consumes("application/xml")
    void submitOrder(Order orderXml) {
      …
    }
}
```

Un-annotated parameters assumed to be incoming message body. There can be only one!

JAX-RS: POST /orders

```
@Path("/orders")
public class OrderService {

    @POST
    @Consumes("application/xml")
    void submitOrder(Order orderXml) {
      …
    }
}
```



More on Content Handlers

- Media type, annotations, object type are all used to fir a handler

```
@XmlElement
public class Order {
…
}

@Path("/orders")
public class OrderService {

    @POST
    @Consumes("application/xml")
    void submitOrder(Order orderXml) {
      …
    }
}
```

More on Content Handlers

- JAXB and other simple types required by specificatior
- JSON? Jackson project is a great provider
- Jettison can output/input JSON from JAXB annotation
  - A little buggy on JSON output
  - Jackson is implementing JAXB support (try that)
- Atom, multipart, XOP and other formats available
- You can write your own custom ones

Response Object

- JAX-RS has a Response and ResponseBuilder class
  - Customize response code
  - Specify specific response headers
  - Specify redirect URLs
  - Work with variants

```
@GET
Response getOrder() {
   ResponseBuilder builder =
        Response.status(200, order);
   builder.type("text/xml")
          .header("custom-header", "33333");
   return builder.build();
}
```

JAX-RS Content Negotiation

- Matched up and chosen based on request ACCEPT header
  - Accept: application/json;q=1.0,application/xml;q=0.5

```
@GET
@Produces("application/xml")
String getXmlOrder() {…}

@GET
@Produces("application/json")
String getJsonOrder() {…}
```

ExceptionMappers

- Map application thrown exceptions to a Response object
  - Implementations annotated by @Provider

```
public interface ExceptionMapper<E>
{
   Response toResponse(E exception);
}
```

JAX-RS Conclusions

- Mapping HTTP requests using annotations
- A la carte HTTP information
- Nice content handlers
- Nice routing

RESTEasy Features

RESTEasy

- Embeddable
- CDI, Spring, EJB, Guice, and Seam integration
- Client Framework
- Asynchronous HTTP (COMET)
- Asynchronous Jobs
- Client and Server Side Caching
- Interceptor model
- Automatic GZIP encoding support
- Data format support
  - Atom, JAXB, JSON, Multipart, XOP

Embeddable

- Very fast unit testing
  - Web embed a fork of TJWS
- Can run within IDE
  - No special plugins

Embeddable

```
public class SimpleTest extends BaseResourceTest  {

   @BeforeClass
   public static void setUp() throws Exception  {
      addPerRequestResource(Resource.class);
   }


   @Test
   public void testEcho()  {
      String url =
             TestPortProvider.generateURL("/my/resource");
      … call the client
   }


}
```

Client Framework

- JAX-RS aware
- Layer over
  - Apache HttpClient 3.1 (very stable)
  - Apache HttpClient 4.x (latest and greatest)
  - java.net.URL (for use within GAE)
- Proxy framework for ease of use
  - Use JAX-RS annotations on the client side

Client Framework


```
ClientRequest request = new ClientRequest("http://…");
request.accept("application/xml");

ClientResponse<Customer> response =
           request.get(Customer.class);

Assert.assertEquals(200, response.getStatus());

Customer cust = response.getEntity();
```

Client Proxy Framework


```
@Path("/customers")
public interface CustomerService {

    @GET
    @Path("{id}")
    @Produces("application/xml")
    public Customer getCustomer(
          @PathParam("id") String id);
}


CustomerService service =
        ProxyFactory(CustomerService.class,
                        "http://example.com");

Customer cust = service.getCustomer("3322");
```

Client Javascript Framework

- Special Javascript Servlet
  - Scans a RESTEasy deployment
  - Builds downloadable Javascript proxies

Client Javascript Framework

```
@Path("/customers")
public class Customers {

    @GET
    @Path("{id})
    @Produces("application/json")
    public Customer getCustomer(
          @PathParam("id") String id) {…}
}
```

```
var customer = Customers.getCustomer({id : "42"})
```

Asynchronous HTTP (COMET)

- “Pushing” events to the browser
  - Really just blocking AJAX HTTP clients
- Servlets generally thread/request
  - 1000 blocking connections, 1000 threads
- Async HTTP
  - Detaching request thread from response
  - Different thread can service response
    - 1 thread can service multiple responses

Asynchronous HTTP (COMET)

- Different Async HTTP implementations
  - Tomcat 6
  - JBossWeb
  - Servlet 3.0
- RESTEasy Async HTTP
  - JAX-RS integration
  - Synchronous façade (for use in non-async-http platforms)
  - Abstraction for different async implementations

Async HTTP


```
@Path("/")
public class MyResource  {
    @GET
    public void get(
      final @Suspend(1000) AsynchronousResponse async)
    {
       Thread thread = new Thread() {
             public void run()
             {
                Response jaxrs = Response.ok("hello")
                                 .type(MediaType.TEXT_PLAIN)
                                 .build();
                async.setResponse(jaxrs);
             }
       };
       thread.start();
    }
}
```

Job Framework

HTTP and ACCEPTED

- Server is allowed to send a 202, “Accepted” response
  - Request was received but not processed yet
- A design pattern
  - Server sends 202 response code
  - Server sends a Location header
    - Location header is an HTTP redirect
    - Location header has a URI that will hold our response

Job Framework

- Any invocation can be made asynchronous
  - uri?asynch=true - creates a job
  - uri?oneway=true - fire and forget
- Returns a Location that can be viewed and deleted
  - GET and DELETE
  - /jobs/{job-id}?wait={time}&nowait=true
  - Returns 410, “Gone” if job doesn’t exist anymore
  - Returns 202, Accepted if job exists but isn’t complete

RESTEasy Caching

Client Cache

- Acts like a browser minus persistence
  - In memory only
- Does validation and conditional gets
- Sharable “Browser” cache instances
- Works with raw request or proxy framework

Client Cache

```
@Path("/orders")
public interface OrderServiceProxy {

    @GET
    @Produces("application/xml")
    @Path("{id"}
    Order getOrder(@PathParam("id") int id);
}
```

…

```
OrderServiceProxy proxy = ProxyFactory.create(
                        "http://example.com");
BrowserCache cache =
CacheFactory.makeCacheable(proxy);

// proxy instance now will cache responses and do
// conditional GETs
```

Server Cache

- Local in-memory cache
- Sits in front of JAX-RS service
- Caches marshalled data
- Auto generates Cache-Control headers
- Generates ETag headers
- Automatically handles conditional gets

RESTEasy Roadmap

RESTEasy Roadmap

- Java EE 6 integration
  - Automatic scanning
  - Zero web.xml configuration
  - Complete CDI and EJB integration
  - JBoss AS6 Milestone 4

RESTEasy Roadmap

- Client Response mapping

```
@ResponseMapping
@ExpectedCode
public class MyResponse {

    @Body
    Customer getCustomer() {…}

    @Header
    public String getEtag() {…}
}

@Path
public interface MyProxy {
    @GET
    MyResponse getCustomer(@PathParameter("id") int
id);
}
```

RESTEasy Roadmap

- Client Response Exception mapping

```
@ResponseMapping
@ExpectedCode(400)
public class ErrorException {
    …
}

@Path
public interface MyProxy {
    @GET
    MyResponse getCustomer(@PathParameter("id") int id)
throws ErrorException
}
```

RESTEasy Roadmap

- REST + SOA + Security
  - Oauth integration for running “on behalf of”
  - Integration with Picketlink and Secure Token Service

RESTEasy Roadmap

- RESTEasy Middleware Interfaces
  - REST-*.org
  - HornetQ (close to release)
  - Transactions (close to iteration)
  - Workflow (prototype only)
  - (See my jboss world talk)

References

- Links
  - http://jsr311.dev.java.net/
  - http://jboss.org/resteasy
  - http://rest-star.org
- O'Reilly Books
  - "RESTFul Java with JAX-RS" by me
  - "RESTful Web Services"
  - "RESTful Web Services Cookbook"