

# JAX- RS

The Java API for RESTful Web Services

Paul Sandoz  
Sun Microsystems  
5251

**JAZOON08**

THE INTERNATIONAL CONFERENCE ON JAVA TECHNOLOGY  
JUNE 23 - 26, 2008 ZURICH



# Learn how to build RESTful Web services using JAX- RS

Paul Sandoz  
Sun Microsystems  
5251

**JAZOON08**

THE INTERNATIONAL CONFERENCE ON JAVA TECHNOLOGY  
JUNE 23 - 26, 2008 ZÜRICH



# AGENDA

- > **REST Primer**
  - **REST in Five Steps**
  - **Common Patterns and Best Practices**
  - **Key Benefits**
- > RESTful Design and API Elements
- > Building a Simple Service
- > Deployment Options
- > Status
- > Q & A

# Short Primer – REST in Five Steps\*

- > Give everything an ID
- > Link things together
- > Use standard methods
- > Multiple representations
- > Stateless communication

\* Inspired by Stefan Tilkov

<http://www.innoq.com/blog/st/presentations/2008/2008-03-13-REST-Intro--QCon-London.pdf>

**JAZOON08**

THE INTERNATIONAL CONFERENCE ON JAVA TECHNOLOGY  
JUNE 23 - 26, 2008 ZÜRICH



# 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`

# 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>
```

# Use Standard Methods

Method	Purpose
GET	Read, possibly cached
POST	Update or create without a known ID
PUT	Update or create with a known ID
DELETE	Remove

# 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`

# Stateless Communication

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

# Common Patterns: Container, Item

## Server in control of URI path space

- > List container contents: **GET** /container
- > Add item to container: **POST** /container
  - with item in request
  - URI of item returned in HTTP response header
  - e.g. **Location: http://host/container/item**
- > Read item: **GET** /container/item
- > Update item: **PUT** /container/item
  - with updated item in request
- > Remove item: **DELETE** /container/item
- > Good example: Atom Publishing Protocol

# Common Patterns: Map, Key, Value

## Client in control of URI path space

- > List key-value pairs: `GET /map`
- > Put new value to map: `PUT /map/{key}`
  - with entry in request
  - e.g. `PUT /map/dir/contents.xml`
- > Read value: `GET /map/{key}`
- > Update value: `PUT /map/{key}`
  - with updated value in request
- > Remove value: `DELETE /map/{key}`
- > Good example: Amazon S3

# Key Benefits

- > Server side
  - Horizontal scaling
  - Straightforward failover
  - Cacheable
  - Reduced coupling
  - Works well with existing Web infrastructure
- > Client side
  - Bookmarkable
  - Easy to experiment in browser
  - Broad programming language support
  - Choice of data formats

# AGENDA

- > REST Primer
- > **RESTful Design and API Elements**
  - **Give everything an ID**
  - **Link things together**
  - **Use standard methods**
  - **Multiple representations**
- > Building a Simple Service
- > Deployment Options
- > Status
- > Q & A

# Give Everything an ID

- > “Thing” == resource class
  - POJO
  - No required interfaces
- > ID provided by `@Path` annotation
  - Relative to deployment context
  - Embedded parameters for non-fixed parts of the URI
  - Annotate class or “sub-resource locator” method

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

    @Path("customer")
    CustomerResource getCustomer(...) {...}
}
```

# Link Things Together

- > `UriInfo` provides information about deployment context, the request URI and the route to the resource
- > `UriBuilder` provides facilities to easily construct URIs for resources

```
@Context UriInfo i;
```

```
OrderResource r = ...
```

```
UriBuilder b = i.getBaseUriBuilder();
```

```
URI u = b.path(OrderResource.class).build(r.id);
```

```
List<URI> ancestors = i.getAncestorResourceURIs();
```

```
URI parent = ancestors.get(ancestors.size()-1);
```

# Use Standard Methods

- > Annotate resource class methods with standard method
  - @GET, @PUT, @POST, @DELETE, @HEAD
  - @HttpMethod meta- annotation allows extensions, e.g. WebDAV
- > Flexible method signatures, annotations on parameters specify mapping from request data
- > Return value mapped to response

```
@Path("orders/{order_id}")
public class OrderResource {
    @GET
    Order getOrder(@PathParam("order_id") String id) {
        ...
    }
}
```

# Multiple Representations

## Static and dynamic content negotiation

- > Annotate methods or classes with static capabilities
  - `@ProduceMime`, `@ConsumeMime`
- > Use `Variant`, `VariantListBuilder` and `Request.selectVariant` for dynamic capabilities

**@GET**

```
@ProduceMime ( { "application/xml", "application/json" } )
Order getOrder(@PathParam("order_id") String id) {
    ...
}
```

**@GET**

```
@ProduceMime ( "text/plain" )
String getOrder(@PathParam("order_id") String id) {
    ...
}
```

# AGENDA

- > REST Primer
- > RESTful Design and API Elements
- > **Building a Simple Service**
- > Deployment Options
- > Status
- > Q & A

# AGENDA

- > REST Primer
- > RESTful Design and API Elements
- > Building a Simple Service
- > **Deployment Options**
  - **Java SE**
  - **Servlet**
  - **Java EE**
  - **Other**
- > Status
- > Q & A

# Java SE

- > `RuntimeDelegate` used to create instances of a desired endpoint class
  - > Application supplies configuration information
    - List of resource classes and providers as subclass of `ApplicationConfig`
  - > Implementations can support any Java type
    - Jersey supports Grizzly (see below) and LW HTTP server
- ```
ApplicationConfig config = ...
RuntimeDelegate rd = RuntimeDelegate.getInstance();
Adapter a = rd.createEndpoint(config, Adapter.class);
SelectorThread st = GrizzlyServerFactory.create(
    "http://127.0.0.1:8084/", a);
```

# Servlet

- > JAX- RS application packaged in **WAR** like a servlet
- > For JAX- RS aware containers
  - **web.xml** can point to **ApplicationConfig** subclass
- > For non- JAX- RS aware containers
  - **web.xml** points to implementation- specific **Servlet**; and
  - an **init-param** identifies the **ApplicationConfig** subclass
- > Resource classes and providers can access **Servlet** request, context, config and response via injection

# Java EE

- > Applications deployed in an EE Web container have access to additional resources and capabilities:
  - Resources (`@Resource`, `@Resources`)
  - Web Services (`@WebServiceRef`, `@WebServiceRefs`)
  - EJB (`@EJB`, `@EJBs`)
  - Persistence (`@PersistenceContext`, `@PersistenceUnit`, ...)
  - Lifecycle management (`@PostConstruct`, `@PreDestroy`)
  - Security (`@RolesAllowed`, `@RunAs`, `@PermitAll`, ...)
- > Hoping to leverage Web Beans (JSR 299) for much of this

# Other

- > Easy to implement for other containers
- > Implementation work underway for:
  - Restlet
  - JBoss RESTEasy
  - Apache CXF
  - Jersey

# AGENDA

- > REST Primer
- > RESTful Design and API Elements
- > Building a Simple Service
- > Deployment Options
- > **Status**
- > Q & A

# Status

- > Early Draft Review completed November 2007
- > Currently in Public Review
- > July: Proposed Final Draft
- > September: Final Release

# AGENDA

- > REST Primer
- > RESTful Design and API Elements
- > Building a Simple Service
- > Deployment Options
- > Status
- > **Q & A**

# More Information

- > Official JSR Page
  - <http://jcp.org/en/jsr/detail?id=311>
- > JSR Project
  - <http://jsr311.dev.java.net/>
- > Reference Implementation
  - <http://jersey.dev.java.net/>
- > Marc's Blog
  - <http://weblogs.java.net/blog/mhadley/>
- > Paul's Blog
  - <http://blogs.sun.com/sandoz/>
- > Jakub's Blog
  - <http://blogs.sun.com/japod/>

**Paul Sandoz**

<http://blogs.sun.com/sandoz/>

**Sun Microsystems**

**Paul.Sandoz@Sun.Com**

**JAZOON08**

THE INTERNATIONAL CONFERENCE ON JAVA TECHNOLOGY  
JUNE 23 - 26, 2008 ZURICH

