

JAX-RS

Oxford University
Software Engineering Programme
Dec 2014



© Paul Fremantle 2012. Portions © Jeremy Gibbons 2010, © WSO2 2005-2012 used with permission of the author(s).
Licensed under the Creative Commons 3.0 BY-SA (Attribution-Sharealike) license.
See <http://creativecommons.org/licenses/by-sa/3.0/>

Introducing JAX-RS Model

- JAX-RS uses Java annotations to map an incoming HTTP request to a Java method.
- To use JAX-RS you annotate your class with the `@Path` annotation to indicate the relative URI path.
- Then annotate one or more of your class's methods with `@GET`, `@POST`, `@PUT`, `@DELETE`, or `@HEAD` to indicate which HTTP method you want dispatched to a particular method.



© Paul Fremantle 2012. Portions © Jeremy Gibbons 2010, © WSO2 2005-2012 used with permission of the author(s).

Licensed under the Creative Commons 3.0 BY-SA (Attribution-Sharealike) license.

See <http://creativecommons.org/licenses/by-sa/3.0/>

Multiple implementations

- Jersey (the “reference implementation”)
 - Apache CXF
 - Apache Wink
 - RESTlet
 - RESTEasy
-
- We will use Apache CXF for the labs



Understanding JAX-RS better



© Paul Fremantle 2012. Portions © Jeremy Gibbons 2010, © WSO2 2005-2012 used with permission of the author(s).
Licensed under the Creative Commons 3.0 BY-SA (Attribution-Sharealike) license.
See <http://creativecommons.org/licenses/by-sa/3.0/>

An Example

- `@Path ("/accounts")`
- `public class`
`AccountEntryService {`
- `@GET`
- `public String getAccounts ()`
`{ . . . }`
- `}`



Query Parameters

- `getAccounts()` method could return thousands of accounts in our system.
- To limit the size of the result set, the client could send a URI query parameter to specify how many results it wanted
 - `http://somewhere.com/accounts?size=50`.
- To extract this information from the HTTP request, JAX-RS has `@QueryParam` annotation:



© Paul Fremantle 2012. Portions © Jeremy Gibbons 2010, © WSO2 2005-2012 used with permission of the author(s).

Licensed under the Creative Commons 3.0 BY-SA (Attribution-Sharealike) license.

See <http://creativecommons.org/licenses/by-sa/3.0/>

Accessing Query Parameters

```
@Path("/accounts")
public class AccountEntryService {
    @GET
    public String getAccounts(
        @QueryParam("size")
        @DefaultValue("50")
        int size)
    {
        ... method body ...
    }
}
```



Other Ways for Parameter Passing

- Other parameter annotations like `@HeaderParam`, `@CookieParam`, and `@FormParam` allow you to extract additional information from the HTTP request to inject into parameters of your Java method.
- `@Context UriInfo` gets information about the URI the request came in on
- `@FormParam` allows you to pull in parameters from an application/x-www-form-urlencoded request body (an HTML form).
- Pretty much behave in the same way as `@QueryParam` does.



Path Parameters

```
@Path("/accounts")
public class AccountEntryService {
    @GET
    @Path("/{id}")
    public String getAccount(
        @PathParam("id") int
        accountId) {
        ... method body ...
    }
}
```



More on Path Parameters

- The {id} string represents our path expression.
- The @PathParam annotation will pull in the info from the incoming URI and inject it into the accountId parameter.
 - For example, if our request is <http://somewhere.com/accounts/111>, accountId would get the value 111 injected into it.
- Complex path expressions are also supported. Use Java regular expressions as follows:
 - @Path("{id: \\d+}")



Handling Content Types

- The String passed back from `getAccount()` could be any mime type: plain text, HTML, XML, JSON, YAML.
- You can specify which mime type the method return type provides with the `@Produces` annotation. For example, let's say `getAccounts()` method actually returns an XML string.
- Also the `@Consumes` can direct different incoming content types to different methods



© Paul Fremantle 2012. Portions © Jeremy Gibbons 2010, © WSO2 2005-2012 used with permission of the author(s).

Licensed under the Creative Commons 3.0 BY-SA (Attribution-Sharealike) license.

See <http://creativecommons.org/licenses/by-sa/3.0/>

Response Content Type

```
@Path("/accounts")
public class AccountEntryService {
    @GET
    @Path("{id}")
    @Produces("application/xml")
    public String
    getAccount(@PathParam("id") int accountId)
    {
        . . .
    }
}
```



Content Negotiation

- HTTP clients use the HTTP Accept header to specify a list of mime types they would prefer the server to return to them.
- Firefox browser sends this Accept header with every request:
 - Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
- JAX-RS understands the Accept header and will use it when dispatching to JAX-RS annotated methods.



Request Content Type

```
@Path("/accounts")
public class AccountEntryService {
    @GET
    @Path("/{id}")
    @Produces("application/xml")
    public String getAccount(@PathParam("id") int
accountId) {...}

    @GET
    @Path("/{id}")
    @Produces("text/html")
    public String getAccountHtml(@PathParam("id") int
accountId) {...}
}
```



Content Marshalling

- JAX-RS allows you to write HTTP message body readers and writers that know how to marshall a specific Java type to and from a specific mime type.
- The JAX-RS specification has some required built-inmarshallers. For instance, vendors are required to provide support for marshalling JAXB annotated classes.



© Paul Fremantle 2012. Portions © Jeremy Gibbons 2010, © WSO2 2005-2012 used with permission of the author(s).

Licensed under the Creative Commons 3.0 BY-SA (Attribution-Sharealike) license.

See <http://creativecommons.org/licenses/by-sa/3.0/>

Response Codes

- The HTTP specification defines what HTTP response codes should be on a successful request.
 - GET should return 200 OK
 - POST should return 201 Created
- You can expect JAX-RS to return the same default response codes.
- Sometimes, however, you need to specify your own response codes, or simply to add specific headers or cookies to your HTTP response. JAX-RS provides a Response class for this.



Examples of creating Responses

200 OK:

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

201 Created

```
return Response.created(  
    URI.create  
    ("orders/" + uuid)).build();
```

404 Not Found

```
return  
Response.status(Status.NOT_FOUND).  
build();
```



© Paul Fremantle 2012. Portions © Jeremy Gibbons 2010, © WSO2 2005-2012 used with permission of the author(s).

Licensed under the Creative Commons 3.0 BY-SA (Attribution-Sharealike) license.

See <http://creativecommons.org/licenses/by-sa/3.0/>

Deployment

- JAX-RS applications are packaged in a WAR like a servlet.
- For CXF the “normal” approach is to use a Spring context to specify which JAX-RS classes should be exposed
- Simply copy the generated WAR to:

`<wso2as>/repository/deployment/server/webapps`



© Paul Fremantle 2012. Portions © Jeremy Gibbons 2010, © WSO2 2005-2012 used with permission of the author(s).

Licensed under the Creative Commons 3.0 BY-SA (Attribution-Sharealike) license.

See <http://creativecommons.org/licenses/by-sa/3.0/>

Creating a JAX-RS project

- `mvn archetype:generate -Dfilter=org.apache.cxf.archetype:`
- Choose a version of CXF
- Choose properties for your project (name, group, version, etc)
- `mvn eclipse:eclipse`
- Import into Eclipse



Questions?



© Paul Fremantle 2012. Portions © Jeremy Gibbons 2010, © WSO2 2005-2012 used with permission of the author(s).
Licensed under the Creative Commons 3.0 BY-SA (Attribution-Sharealike) license.
See <http://creativecommons.org/licenses/by-sa/3.0/>