Introduction to JAX-RS

- Resource Classes
- Annotations
- Paths
- Context

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JAX-RS

- JAX-RS is an API for providing RESTful services in Java
- JAX-RS went through the Java Community Process as JSR-311 and is a part of JEE Version 6



If you don't have a JEE V6 container, you can use the reference implementation of JAX-RS, Jersey, which you can download and read about at:

`http://jersey.java.net`

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JAX-RS Annotations

```
@Path specifies a relative URI for a resource class or method
@ApplicationPath specifies a relative URI for application class
@GET, @PUT, @POST, @DELETE and @HEAD specify the HTTP method
@Produces specifies response Media Type(s)
@Consumes specifies accepted request Media Type(s)
@PathParam binds a parameter based on URI
@QueryParam binds a parameter based on HTTP query parameter
@MatrixParam binds a parameter based on HTTP matrix parameter
@HeaderParam binds a parameter based on HTTP header
@CookieParam binds a parameter based on cookie
@FormParam binds a parameter based on form value
@DefaultValue specifies a default value
@Context binds a field or parameter to an environmental value
```

All of these annotations are from the javax.ws.rs package.

We will see examples of many of these annotations in this chapter.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a Resource Class

- A **resource class** defines methods that respond to RESTful GET, POST or PUT methods, decorated with JAX-RS annotations
- Resource classes must provide a public constructor that JAX-RS can use, either a zero-argument constructor or a constructor with arguments annotated with @Context

```
1    @Path("/student")
2    public class RosterService
3    {
4      @GET
5      @Produces("text/xml")
6      public String studentAsXml(
7       @QueryParam("studentID") String id)
8      {
9        . . .
10     }
11   }
```

According to the JAX-RS specification, implementations will instantiate a separate resource object for each request, at least by default for non-singleton objects.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing an Application Class

- An **application class** configures the resource classes (and providers) that make up a JAX-RS application
- Such classes extend **javax.ws.rs.core.Application** and typically override the **getClasses** method

```
1    @ApplicationPath("/roster/")
2    public class RosterApplication extends Application
3    {
4      public Set<Class<?>> getClasses()
5      {
6        Set<Class<?>> s = new HashSet<Class<?>>();
7        s.add(RosterService.class);
8        return s;
9      }
10   }
```

The JAX-RS specification notes that implementations can provide other ways of configuration, but the Application subclass is the only portable technique.

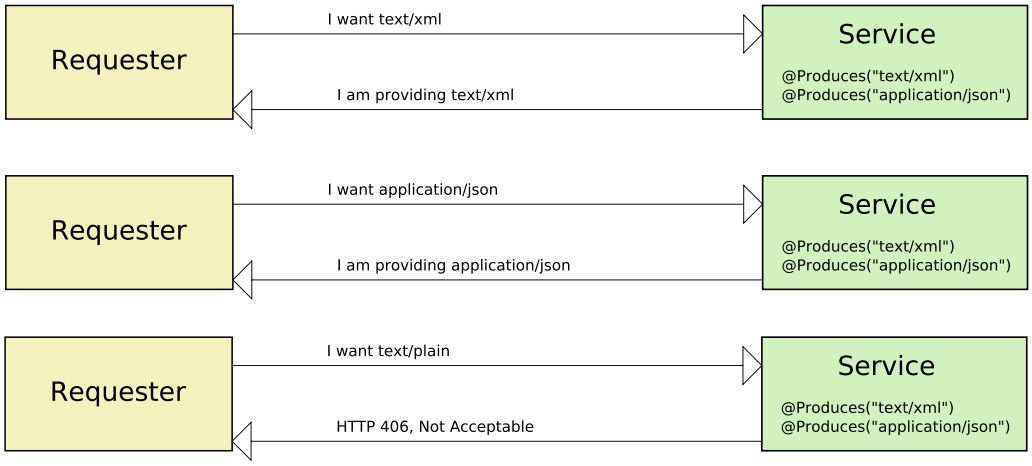
Instead (or in addition to) overriding getClasses, you can override getSingletons and return a Set of objects. If you do so, JAX-RS will not create a separate instance for the resource class for each request.

"Providers" are optional classes that you can write to extend the JAX-RS framework. For example, you can write an "entity provider" that changes how JAX-RS maps between Java types and resource representations. See the JAX-RS specification for more details.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Content Negotiation

- The client and server must agree on a data **representation**, which is usually a standard format such as text, XML or JSON using a **MediaType**
- Ultimately, the service decides on which representations it will provide



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Negotiation via HTTP Accept Header

- The HTTP specification allows for an **Accept** header, in which the user agent client provide a list of MIME types
- For GET requests, you use the **@Produces** annotation and the **@Consumes** annotation for POST and PUT requests
- JAX-RS will map requests to methods based on the Accept header and the annotations

```
curl -H "Accept: application/json, application/xml"
 http://localhost:8080/TestRESTWeb/rest/advisor/12

{"id":12,"name":"Sue Joens","salary":50000.0}
```


---

cURL is a popular and powerful tool for sending network requests - it's very useful for testing RESTful services. cURL is included with many operating systems, but not Windows. You can download it from;

`http://curl.haxx.se/download.html`

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Content Negotiation Example #1

```
1    // JavaScript Ajax requester
2
3    req.open("GET", url, true);
4    req.setRequestHeader("Accept","application/json");
5    req.onreadystatechange = callback;
6    req.send(null);
```

```
1    // JAX-RS resource class
2
3    @GET
4    @Produces("application/json")
5    public String studentAsJSON(
6       @QueryParam("studentID") String id)
7    {
8    . . .
9    }
```


Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Content Negotiation Example #2

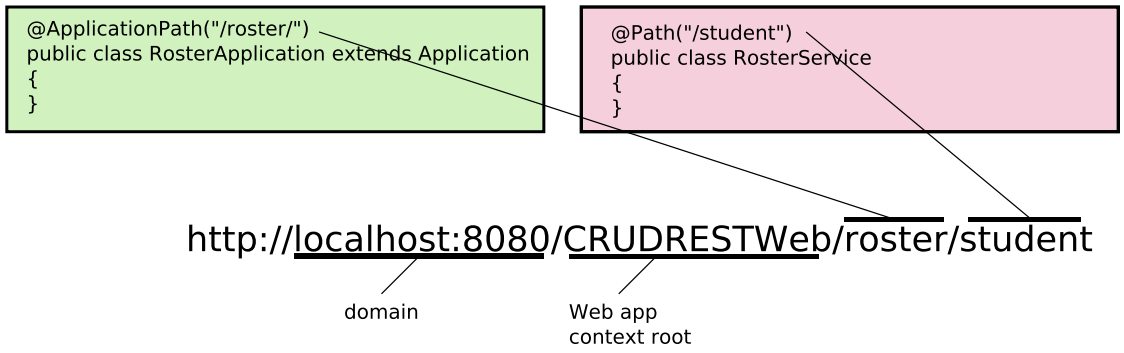
```
1    // JavaScript Ajax requester
2
3    req.open("POST", url, true);
4    req.setRequestHeader("Content-type","text/xml");
5    req.onreadystatechange = callback;
6    req.send(xml);
```

```
1    // JAX-RS resource class
2
3    @POST
4    @Consumes("text/xml")
5    public String createStudentFromXml(String xml)
6    {
7    . . .
8    }
```


Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Assigning a URI Path

- To make methods available to respond to REST requests, you use the **Path** annotation to assign URIs
- You also can assign an overall application URI



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing Request Parameters

- HTTP **GET** requests can contain parameters as part of the URL
- JAX-RS methods can bind to such parameters using the **@QueryParam** annotation

```
1    // JavaScript Ajax client
2
3    // url like http://...../roster/student?studentID=12
4    var url="roster/student?studentID=" + escape(studentID);
```

```
1    // JAX-RS resource
2    @GET
3    @Produces("text/xml, application/json")
4    public String studentAsXml(
5       @QueryParam("studentID") String id)
6    {
7     . . .
8    }
```


Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing URI Parameters

- Instead of, or in addition to, HTTP GET request parameters, RESTful applications can use parameters from the URI itself
- JAX-RS methods can bind to such parameters using the **@PathParam** annotation

```
1    // JAX-RS resource
2    @Path("/student/{studentId}")
3    public class StudentService
4    {
5        @GET
6        @Produces("text/xml, application/json")
7        public Response find(@PathParam("studentId")int id)
8        {
9            . . .
10       }
11   }
```


Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @Context Annotation

- You can use the **javax.ws.rs.core.Context** annotation to decorate fields or method parameters to **inject** context information of the following types:
  - **Application** – Reference to the Application subclass
  - **UriInfo** – Request URI information
  - **HttpHeaders** – HTTP headers
  - **Request** – Content negotiation information
  - **SecurityContext** – User principal, etc.
  - **ServletConfig**, **ServletContext**, **HttpServletRequest**, **HttpServletResponse** – Standard JEE objects

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

@Context Example

```
1    @Path("/student")
2    public class RosterService
3    {
4      @Context
5       HttpServletRequest request;
6
7      @GET
8      @Produces("text/xml, application/json")
9      public String studentAsXml(
10            @Context HttpHeaders headers)
11     {
12       System.out.println("User-agent: " +
13          headers.getRequestHeader("user-agent"));
14       HttpSession session = request.getSession();
15
16       . . .
17      }
18   }
```


Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mapping Java Objects to Content Types

- Many JAX-RS implementations, including Jersey, can automatically map Java objects to/from common types such as XML or JSON
- You must decorate the Java classes with **JAX-B** annotations

```
1    // JAX-RS resource
2    @GET
3    @Produces("text/xml, application/json")
4    public Response find(@PathParam("studentId")int id)
5    {
6        Student s = . . .;
7        return Response.ok(s).build();
8    }
9
10   @XmlRootElement
11   public class Student{. . .}
```

The javax.ws.rs.core.Response lets you build responses with well known HTTP status codes and response body. Here we send back HTTP 200 OK and the response body with a Student object serialized into XML.

Note for this to work in Jersey, the application's CLASSPATH must include optional JARs including jersey-json*.jar and jackson-*.jar.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a JAX-RS Application

- Create one or more **resource** classes that produce or consume RESTful data and annotate them with JAX-RS annotations
- Create a class that extends **javax.ws.rs.core.Application** and overrides the **getClasses** method
- Configure JAX-RS libraries into Web application (if not using a full JEE application server)
- Deploy Web application
- Write client(s) that invoke the service

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example

Student.java

Roster.java

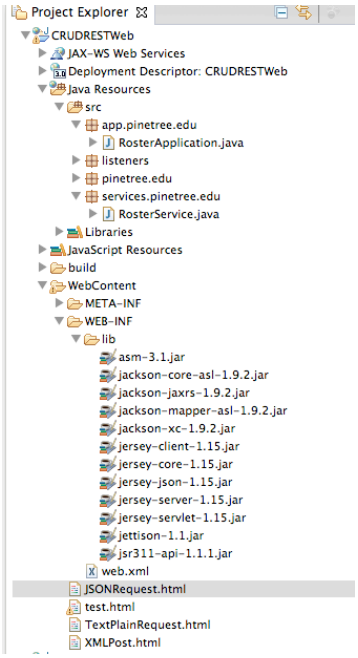
MyContextListener.java

RosterApplication.java

RosterService.java

GETRequest.html

XMLPost.html



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package pinetree.edu;
2
3    import javax.xml.bind.annotation.XmlRootElement;
4
5    @XmlRootElement
6    public class Student
7    {
8        private int studentID;
9        private String name;
10       private double gpa;
11
12       @Override
13       public int hashCode()
14       {
15           final int prime = 31;
16           int result = 1;
17           result = prime * result + studentID;
18           return result;
19       }
20
21       @Override
22       public boolean equals(Object obj)
23       {
24           if (this == obj)
25               return true;
26           if (obj == null)
27               return false;
28           if (getClass() != obj.getClass())
29               return false;
30           Student other = (Student) obj;
31           if (studentID != other.studentID)
32               return false;
33           return true;
34       }
35
36       public int getStudentID()
37       {
38           return studentID;
39       }
40
41       public void setStudentID(int studentID)
42       {
43           this.studentID = studentID;
44       }
45
```

```
46      public String getName()
47      {
48          return name;
49      }
50
51      public void setName(String name)
52      {
53          this.name = name;
54      }
55
56      public double getGpa()
57      {
58          return gpa;
59      }
60
61      public void setGpa(double gpa)
62      {
63          this.gpa = gpa;
64      }
65
66      @Override
67      public String toString()
68      {
69          return "Student [studentID=" + studentID + ", name=" + name + ", gpa="
70                  + gpa + "]";
71      }
72
73  }
```

```
1    package pinetree.edu;
2
3    import java.util.ArrayList;
4    import java.util.List;
5
6    import javax.xml.bind.annotation.XmlRootElement;
7
8    @XmlRootElement
9    public class Roster
10   {
11       private List<Student> students = new ArrayList<Student>();
12
13       public List<Student> getStudents()
14       {
15           return students;
16       }
17
18       public void setStudents(List<Student> students)
19       {
20           this.students = students;
21       }
22   }
```

```
1    package listeners;
2
3    import javax.servlet.ServletContextEvent;
4    import javax.servlet.ServletContextListener;
5
6    import pinetree.edu.Roster;
7    import pinetree.edu.Student;
8
9    public class MyContextListener implements ServletContextListener
10   {
11       @Override
12       public void contextDestroyed(ServletContextEvent evt)
13       {
14       }
15
16       @Override
17       public void contextInitialized(ServletContextEvent evt)
18       {
19           Roster r = new Roster();
20           Student s = new Student();
21           s.setGpa(3.44);
22           s.setName("Sam B. Jones");
23           s.setStudentID(12);
24           r.getStudents().add(s);
25
26           s = new Student();
27           s.setGpa(3.45);
28           s.setName("Susan Q. Smith");
29           s.setStudentID(13);
30           r.getStudents().add(s);
31
32           s = new Student();
33           s.setGpa(2.98);
34           s.setName("Xiang Lu");
35           s.setStudentID(14);
36           r.getStudents().add(s);
37
38           System.out.println("Context initialized");
39
40           evt.getServletContext().setAttribute("roster", r);
41       }
42   }
```

```
1    package app.pinetree.edu;
2
3    import java.util.HashSet;
4    import java.util.Set;
5
6    import javax.ws.rs.ApplicationPath;
7    import javax.ws.rs.core.Application;
8
9    import services.pinetree.edu.RosterService;
10
11   @ApplicationPath("/roster/")
12   public class RosterApplication extends Application
13   {
14       @Override
15       public Set<Class<?>> getClasses()
16       {
17           Set<Class<?>> s = new HashSet<Class<?>>();
18           s.add(RosterService.class);
19           return s;
20       }
21
22   }
```

```
1    package services.pinetree.edu;
2
3    import java.net.InetAddress;
4    import java.net.URI;
5
6    import javax.servlet.ServletContext;
7    import javax.servlet.http.HttpServletRequest;
8    import javax.ws.rs.Consumes;
9    import javax.ws.rs.GET;
10   import javax.ws.rs.HeaderParam;
11   import javax.ws.rs.POST;
12   import javax.ws.rs.Path;
13   import javax.ws.rs.Produces;
14   import javax.ws.rs.QueryParam;
15   import javax.ws.rs.core.Context;
16   import javax.ws.rs.core.Response;
17
18   import pinetree.edu.Roster;
19   import pinetree.edu.Student;
20
21   @Path("/student")
22   public class RosterService
23   {
24       @Context
25       ServletContext context;
26
27       @GET
28       @Produces("text/xml, application/json")
29       public Response getStudent(@QueryParam("studentID")
30          String id, @HeaderParam("Accept") String acceptHeader)
31       {
32           System.out.println("Received GET request for student: " +
33              id + " requested type: " + acceptHeader);
34           Response retVal = null;
35
36           retVal = findStudent(id);
37
38           return retVal;
39       }
40
41       private Response findStudent(String id)
42       {
43           Response retVal;
44           int studentID = Integer.parseInt(id);
45           Roster roster = (Roster)context.getAttribute("roster");
```

```
46
47          Student found = null;
48          for (Student s : roster.getStudents())
49          {
50              if (s.getStudentID() == studentID)
51              {
52                  System.out.println("Found a student: " + s);
53                  found = s;
54                  break;
55              }
56          }
57
58          if (found == null)
59          {
60              System.out.println("No student found");
61              retVal =
62                Response.status(Response.Status.NOT_FOUND).build();
63          }
64          else
65          {
66              retVal = Response.ok(found).build();
67          }
68          return retVal;
69      }
70
71      @POST
72      @Consumes("text/xml")
73      public Response createStudentFromXml(Student s,
74              @Context final HttpServletRequest request)
75          throws Exception
76      {
77          Response retVal =
78            Response.status(Response.Status.BAD_REQUEST).build();
79
80          Roster roster = (Roster) context.getAttribute("roster");
81
82          if (s != null)
83          {
84              System.out.println("Received student XML " + s);
85              if (!roster.getStudents().contains(s))
86              {
87                  roster.getStudents().add(s);
88
89                  String newStudentURI = "http://" +
90                    InetAddress.getLocalHost().getHostName() +
```

```
91                     ":" + request.getLocalPort() + request.getContextPath() +
92                     "/roster/student?studentID=" + s.getStudentID();
93
94                  System.out.println(
95                     "Created student resource with URI: " +
96                          newStudentURI);
97
98                  // the Response.created() method sets the HTTP
99                  // Location header to the given URI
100                 retVal = Response.created(
101                    URI.create(newStudentURI)).build();
102             }
103         }
104
105         return retVal;
106     }
107 }
```

```
<html>
  <head>
    <title>XML POST</title>
    <script>

      function sendRequest()
      {
        var url = "roster/student";
        sendAsyncPostRequest(url);
      }

      function callback()
      {
        if (req.readyState == 4)
        {
            if (req.status == 200)
            {
            var text = req.responseText;
            alert(text);
            }
            else
            {
                alert("Problem: " + req.status + ", " + req.statusText);
            }
        }
      }

      function sendAsyncPostRequest(url)
      {
        req = getXMLHttpRequest();
        if (req)
        {
        var studentID = document.getElementById("studentid").value;
        var name = document.getElementById("name").value;
        var gpa = document.getElementById("gpa").value;
        var xml = "<student><name>" + name + "</name>" +
                             "<studentID>" + studentID + "</studentID>" +
                             "<gpa>" + gpa + "</gpa>" +
                    "</student>";

          req.open("POST", url, true);
          req.setRequestHeader("Content-type","text/xml");
          req.onreadystatechange = callback;
          req.send(xml);
        }
```

```
46          }
47
48          function getXMLHttpRequest()
49          {
50            var xRequest=null;
51            if (window.XMLHttpRequest)
52            {
53                xRequest=new XMLHttpRequest();
54            }
55            else if (typeof ActiveXObject != "undefined")
56            {
57                xRequest=new ActiveXObject("Microsoft.XMLHTTP");
58            }
59            return xRequest;
60          }
61        </script>
62      </head>
63      <body>
64        <form>
65          <p>
66            Student ID:   <input type="text" name="studentID" id="studentid"
67          </p>
68          <p>
69            Name:   <input type="text" name="name" id="name">
70          </p>
71          <p>
72            GPA:   <input type="text" name="gpa" id="gpa">
73          </p>
74          <p>
75            <input type="button" value="Insert Student" onclick="sendRequest()">
76          </p>
77        </form>
78      </body>
79    </html>
```

```
1    <html>
2      <head>
3        <title>JSON Request</title>
4        <script>
5
6          function sendRequest()
7          {
8            var studentID =
9                document.getElementById("studentid").value;
10           var mediaType =
11               document.getElementById("mediaType").value;
12           var url = "roster/student?studentID=" +
13               escape(studentID);
14           sendAsyncGetRequest(url, mediaType);
15         }
16
17         function showResponse(text)
18         {
19           if (text.trim().length == 0)
20             text = "No student with that ID";
21
22           var resp = document.getElementById('output');
23
24           resp.firstChild.nodeValue=text;
25         }
26
27         function callback()
28         {
29           if (req.readyState == 4)
30           {
31               if (req.status == 200)
32               {
33                   var text = req.responseText;
34
35                   showResponse(text);
36               }
37               else
38               {
39                   alert("Problem: " + req.status + ", " +
40                      req.statusText);
41               }
42           }
43         }
44
45         function sendAsyncGetRequest(url, mediaType)
```

```
46          {
47            req = getXMLHttpRequest();
48            if (req)
49            {
50              req.open("GET", url, true);
51              req.setRequestHeader("Accept",mediaType);
52              req.onreadystatechange = callback;
53              req.send(null);
54            }
55          }
56
57          function getXMLHttpRequest()
58          {
59            var xRequest=null;
60            if (window.XMLHttpRequest)
61            {
62                xRequest=new XMLHttpRequest();
63            }
64            else if (typeof ActiveXObject != "undefined")
65            {
66                xRequest=new ActiveXObject("Microsoft.XMLHTTP");
67            }
68            return xRequest;
69          }
70        </script>
71      </head>
72      <body>
73        <h3>Service Response</h3>
74        <div id="output">Nothing yet</div>
75        <form>
76          <p>
77            Student ID:
78            <input type="text" name="studentID" id="studentid">
79          </p>
80          <p>
81            Media Type (text/xml or application/json):
82            <input type="text" name="mediaType" id="mediaType">
83          </p>
84          <p>
85            <input type="button" value="Send Request"
86               onclick="sendRequest()">
87          </p>
88        </form>
89      </body>
90    </html>
```

Chapter Summary

In this chapter, you learned:

- The fundamentals of the JAX-RS framework
- How to write **resource** classes and **application** classes
- How JAX-RS applications negotiate content representation types

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum