BITS Pilani
Pilani | Dubai | Goa | Hyderabad

RESTful Web Services

Srikanth Gunturu

Guest Faculty
BITS, WILP

In this segment

RESTful Web Services

- REST – Architectural Constraints
- JAVA API FOR RESTFUL WEB SERVICES (JAX-RS)
  - Annotations
  - Server deployment
  - Client API
  - JSON API

RESTful Web Services

REST – Architectural Constraints

- Client-server model - "*Servers and clients may also be replaced and developed independently, as long as the interface between them is not altered*"
- Statelessness - "*No client context shall be stored on the server between requests. Client is responsible for managing the state of application.*"
- Cacheability – "*Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance*"
- Layered system – *"Client need not be aware of underlying layers managed by the service provider"*
- Code-on-demand (optional) – "*..when you need to, you are free to return executable code to support a part of your application..*"
- Uniform interface – "*Once a developer become familiar with one of your API, he should be able to follow similar approach for other APIs*"
  - Resource identification in requests
  - Resource manipulation through representations
  - Self-descriptive messages
  - Hypermedia As The Engine Of Application State (HATEOAS)

RESTful Web Services

JAVA API FOR RESTFUL WEB SERVICES (JAX-RS)

- Java based middleware for creating RESTful web services
- Reference Implementations – Jersey, Apache CXF, Spring
- Commonly deployed on a servlet container, or EJB / OSGi container
- Method Annotations
  - @GET
  - @POST
  - @PUT
  - @DELETE
- MIME Annotations
  - @Consumes
  - @Produces
- Parameter Annotations
  - @PathParam
  - @QueryParam
  - @FormParam

```
@Path("/hello")
public class Hello {
    @GET
  public String method_that_handles_GETs() {
   …useful stuff...
  }
}
```

```
@Path("/hello")
public class Hello {
    @POST
    @Consumes(MediaType.TEXT_XML)
    public String method_that_handles_POSTs() {
     …useful stuff...
    }
}
```

http://localhost:8080/project1/sample/student?classname="..."&name="..."

RESTful Web Services

JAX-RS – Sample code

```
@Path("/hello")
public class Hello {
        // called when request has accept text/plain header
        @GET
        @Produces(MediaType.TEXT_PLAIN)
        public String return_plaintext() {
            return "Hello World\n\n";
        }
         // called when request has accept text/html header
        @GET
        @Produces(MediaType.TEXT_HTML)
        public String return_html() {
        return "<html> " + "<body>" + "<h1>" + "Hello" +"</h1>"+"<h2>"+"to" + "</h2>"
         + "<h2>" + "all the World" + "</h2>"+"</body>" + "</html> "+"\n";
        }
         // called when request has accept text/xml header
        @GET
```

RESTful Web Services

JAX-RS – Sample code

```
    @Produces(MediaType.TEXT_XML)
    public String return_XML() {
        return "<?xml version=\"1.0\"?>" + "<thedata> Hello World" + "</thedata>"+"\n";
    }
    // called when request sends a POST with data in format "variable=value"
    @POST
    public String received_POST(String msg) {
        return "POST: got a string "+msg+"\n";
    }
    // called when request sends a PUT with JSON data
    @PUT
    @Consumes(MediaType.APPLICATION_JSON)
    public String received_JSON_PUT(String msg) {
        return "PUT: got a JSON file "+msg+"\n";
    }
    // called when request sends a POST with JSON data and
    // must return more interesting response
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public Response received_JSON_POST(String msg) {
      if (!msg.isEmpty())
        {
          String mymessage = "POST: got a JSON file " + msg + "\n";
          return Response.status(200).entity(mymessage).build();
        }
      else {
        return Response.status(Response.Status.NOT_FOUND).entity("Null String Received\n
          \n").build();
        }
    }
}
```

RESTful Web Services

JAX-RS – Sample code execution

- GET
  - Media type - text/plain

```
curl --request GET -H "Accept: text/plain" http://localhost:8080/myrestful/restful/hello

Output produced is:

Hello World
```

  - Media type - text/html

```
curl --request GET -H "Accept: text/html" http://localhost:8080/myrestful/restful/hello

Output produced is:

<html> <body><h1>Hello</h1><h2>to</h2><h2>all the World</h2></body></html>
```

- POST

```
curl -X POST --data "mystuff=32" http://localhost:8080/myrestful/restful/hello

Output produced is:

POST: got a string mystuff=32
```

- PUT

```
curl -X PUT -H "Content-Type: application/json" -d @mydata
http://localhost:8080/myrestful/restful/hello

Output produced is:

PUT: got a JSON file { "firstName":"Joe", "lastName":"Blow",
"education":{"bachelors":"BSCS","masters":"MSCS"}, "employer":"ACME Software" }
```

RESTful Web Services

Deploying JAX-RS server

- Using Deployment Descriptor → web.xml

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
        version="3.1">
        <!-- load application class
         all @Path and @Provider classes included in application will be
         automatically registered in the application -->
        <servlet>
                <servlet-name>javax.ws.rs.core.Application</servlet-name>
                <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
                <servlet-name>javax.ws.rs.core.Application</servlet-name>
                <url-pattern>/restful/*</url-pattern>
        </servlet-mapping>
</web-app>
```

RESTful Web Services

Deploying JAX-RS server

- Using ApplicationConfig class

```
package example.restful;
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;
@ApplicationPath("/restful")
public class ApplicationConfig extends Application
{
    // empty class
}
```

RESTful Web Services

JAX-RS Client API

- Using Generic Invocation

```
@GET
@Produces(MediaType.TEXT_PLAIN)
public String return_plaintext() {
  //  a "generic invocation"

  // Create a Client and a target URL
  Client client=ClientBuilder.newClient();
  WebTarget theURL = client.target("http://localhost:8080/myrestclient/restclient/hello");

 // Build the request
  final String mediaType=MediaType.TEXT_PLAIN;
  Invocation theInvoke = theURL.request(mediaType).buildGet();

  // Do actual invocation
  // this invocation could have been done much later
  Response theresponse = theInvoke.invoke();

  String thestuff = theresponse.readEntity(String.class);
  System.out.println(thestuff);

  return thestuff;
}
```

RESTful Web Services

JAX-RS Client API

- Using Specific Invocation

```
Client client=ClientBuilder.newClient();
WebTarget theURL = client.target("http://localhost:8080/myrestful/restful/hello");
```

```
Response theGetResponse=theURL.request(mediaType).get();
thestuff = theGetResponse.readEntity(String.class);
```

RESTful Web Services

JAX-RS JSON API

- Building a nested JSON object

```
JsonObject myobject =
        Json.createObjectBuilder()
                .add("hi","y'all")
                        .add("location",
                                Json.createObjectBuilder()
                                        .add("city","Huntsville")
                                        .add("state","Alabama"))
                . build();
```

Thank You!

In our next session:

Middleware for Cloud applications