RESTful Web Services

Laboratory of Service Design and Engineering

2011/2012

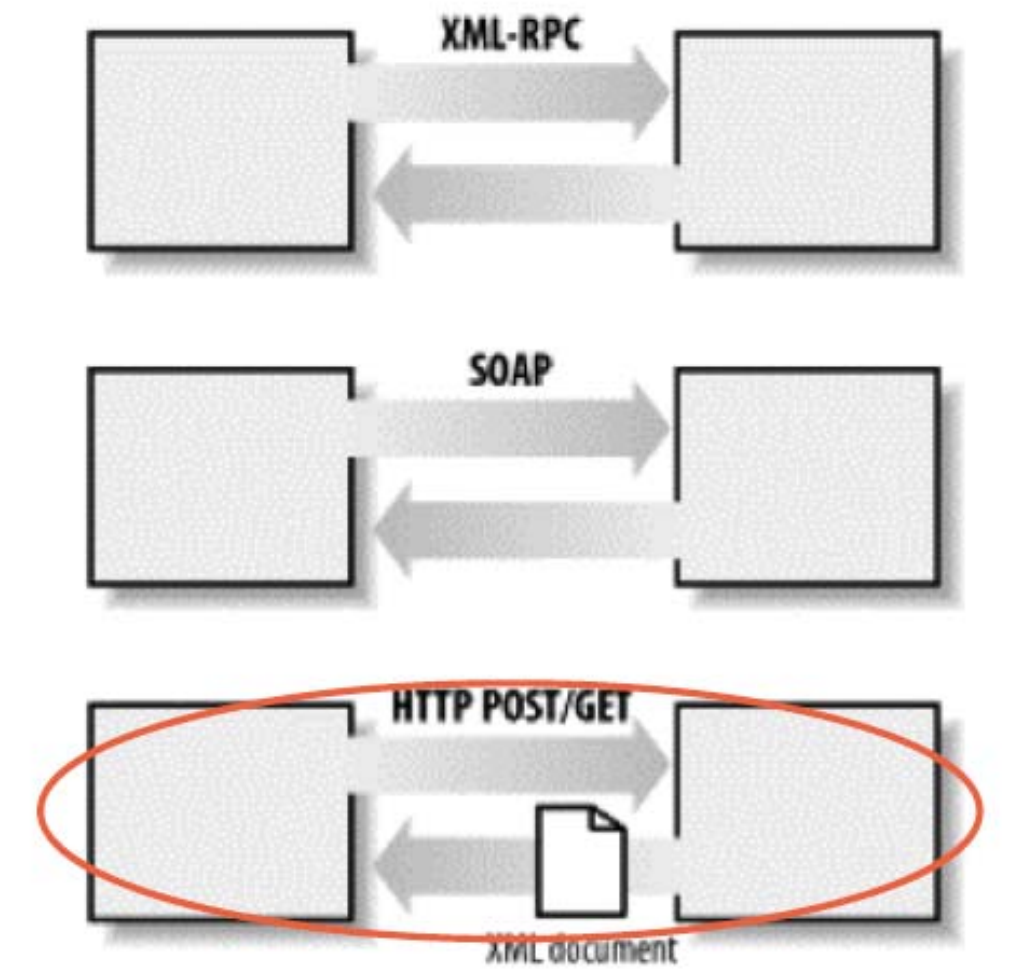
Outline

- Introduction to REST
- HTTP Protocols
- Atom & RSS
- JAX-RS
- Examples

What is REST?

- The term Representational State Transfer (REST) was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation
- REST is a methodology to creating Web Services
- REST services are built around Resources
- REST services are Stateless
- REST uses a Uniform Interface

Web Services Messaging



HTTP Protocol

- Document based protocol
- Request/response stateless protocol
  - A client sends a request to a server. The HTTP message contains request method, path and protocol version, followed by request headers, containing meta-information, and possible entity body content.
  - The server responds with a HTTP response code, followed by response headers, containing meta information, and possible entity-body content.

HTTP methods

HTTP Method	CRUD Action	Description
POST	CREATE	Create a new resource
GET	RETRIEVE	Retrieve a representation of a resource
PUT	UPDATE	Create or update a resource
DELETE	DELETE	Delete a resource

RESTFul Web Services

- How can a client convey his intentions to the server?
  - It puts the method information in the HTTP method
    - “retrieve some data” (GET)
    - “delete data” (DELETE)
    - “overwrite it with different data” (PUT)
- How does client tells the server which part of the data set to operate on?
  - It puts the scoping information into the URI

REST: Noun (Resource) Oriented

- About resources
- The operations are standard via HTTP
- Resources can be cached, bookmarked, saved via standard mechanisms

Customer

http://example.com/customer/123

http://example.com/order/555/customer

{POST, GET, DELETE}

Examples

- March 1, 2005 Yahoo! Launches REST API
- On December 5, 2006, Google stopped accepting new signups for SOAPSearch
- Public REST APIs:

Example: Flickr

- Flickr supports “REST request/response format”
  - http://www.flickr.com/services/api/
- Suppose we request to Flickr all photos tagged penguin. We can use two URIs:
  - http://flickr.com/photos/tags/penguin
  - http://api.flickr.com/services/rest/?method=flickr.photos.search&tags=penguin
- The web site (1) and the web service (2) do the same thing but with a different design!

Examples

- Twitter:
  - http://search.twitter.com/search.json?q=elkstein&count=5
  - http://search.twitter.com/search.atom?q=elkstein&count=5
- Yahoo Search supports “REST request/response format”
  - http://developer.yahoo.com/search/rest.html
- Suppose we request to "yahoo search" a list of search results for the query ‘dog’. We may use two URIs:
  - http://search.yahoo.com/search?p=dog
  - http://api.search.yahoo.com/WebSearchService/V1/webSearch?appid=restbook&query=dog

Atom vs RSS

- Atom intended as a replacement for RSS, published under IETF
  - Clarify ambiguities
  - Unify
  - Extend capabilities

Rome: Atom & RSS

- https://**rome**.dev.java.net
- Makes it easy to work with RSS and Atom in Java
- Include parsers and generators for a variety of feeds
  - Get Java objects representing specific feed types
  - Or, work normalized objects (SyndFeed)

JAX-RS

JAX-RS

- Java API for RESTful Web Services
- Help developers to quickly write RESTful applications
- API Expressed in Annotations
- Now part of Java EE 6

Resources

- In JAX-RS, a Resource is a POJO
  - No interface to implement
  - Just express the matching URI
- `@Path`
  - The value is a relative path
  - The base URI is provided by the either
    - Deployment Context
    - Parent Resource

JAX-RS Methods

- If your method returns `void`, JAX-RS returns a 204 (successfully processed, no message body)
- Automatic encoding
  - `@Path("product list")` is identical to `@Path("product%20list).`

Hello Jersey!

```
import javax.ws.rs.Path;
import javax.ws.rs.GET;
import javax.ws.rs.ProduceMime;

@Path("/helloRest")
public class HelloRest {
  @GET
  @Produces("text/html")
  public String sayHello() {
      return "<html><body><h1>Hello from
            Jersey!</body></h1></html>";
  }
}
```

Uniform Interface

- **Annotate methods**
  - **`@GET, @PUT, @POST,`**
  - **`@DELETE`**
- JAX-RS forwards to correct method based on request method

URI Templates

- At class level, assign a Root Resource with `@Path`
- Dynamic resources assigned using `@PathParam(paramName)`
- Can use Regular Expressions to match
  - `@Path("products/{id}:[a-zA-Z][a-zA-Z_0-9]}"}`
  - Non-matches return 404

URI Template Example

```
@Path("/products/{id}")
public class ProductResource {
   @Context
   private UriInfo context;
   /** Creates a new instance of ProductResource */
   public ProductResource() { }

   @GET
   @ProduceMime("text/plain")
   public String getProduct(@PathParam("id") int productId) {
      switch (productId) {
      case 1: return "A Shiny New Bike";
      case 2: return "Big Wheel";
      case 3: return "Taser: Toddler Edition";
      default: return "No such product";
      }
   }
}
```

Variable Resources of the Same Type

- Map path elements using `@PathParam`:

```
@Path(“customer/{name}")
public class Customer {
   @GET
   String get(@PathParam("name") String
     name) { … }
   @PUT
   Void put(@PathParam(“name”) String name,
     String value) { … }
```

Regular Expressions in URI Template

```
@Path("/products/{id: \\d{3}}")
public class ProductResource {
      public ProductResource() { }
      @GET
      @Produces("text/plain")
      public String getProductPlainText(@PathParam("id") int productId) {
      return "Your Product is: " + productId;
      }
}
```

//constrained to 3 digits:

http://localhost:8080/jrs/resources/products/555

works

http://localhost:8080/jrs/resources/products/7

returns 404

Accessing Query Parameters

- Use `@QueryParam` on your method parameter
- Optionally include `@DefaultValue`

```
@GET
@Produces("text/xml")
public String getProducts(
    @PathParam("id") int productId,
    @QueryParam("results")
    @DefaultValue("5") int numResults)
    //.../resources/products?results=3
```

Accessing Request Headers

```
@GET public String doGet(@Context
  HttpHeaders headers) {
//list all incoming headers
MultivaluedMap<String,String> h =
  headers.getRequestHeaders();
for (String header : h.keySet()) {
System.out.println(header + "=" +
  h.get(header));
}
```

Representation Formats

- Identified by media type
  - `text/xml, application/json`
- Content negotiation is automatically handled by JAX-RS
  - Annotate with `@Produces` or `@Consumes` to indicate static content capabilities

```
@Path("/emps")
public class EmployeeService {
   @GET
   @Path("{id}")
   @Produces("application/xml")
   public Employee getEmployee(@PathParam("id") int empId) {
      return emps.get(empId);
}

//this example uses JAXB on the Employee POJO for XML:
@XmlRootElement(name="employee")
public class Employee { …id, name }
```

Produces/Consumes

- **`@Produces`**
  - Specify the MIME media types of representations a resource can produce and send back to the client.
  - Applied at Class or Method level
- **`@Consumes`**
  - Specify MIME media types of representations a resource can consume that were sent by the client
  - Applied at Class or Method level
  - One method can consume more than one media type

Adding Metadata to Responses

```
Response response = Response.noContent()
.header("MY_KEY", "MY_VALUE")
.cacheControl(cacheCtl)
.expires(expy)
.language(Locale.ENGLISH)
.type(MediaType.TEXT_HTML)
.build();

HTTP/1.1 204 No Content
Server: Apache-Coyote/1.1
Cache-Control: no-store, no-transform, must-
  revalidate, max-age=500
Expires: Sat, 10 Oct 2009 16:41:49 GMT
MY_KEY: MY_VALUE
Content-Language: en
Date: Sat, 08 Nov 2008 16:41:49 GMT
```

Security

- Available via the `SecurityContext` from `@Context`
- Same as security in `HttpServletRequest:`

```
@Path(“cart")
public ShoppingBasketResource
  get(@Context SecurityContext sc) {
   if(sc.isUserInRole(“GoldMember")
   { //…
```

Exercise

- Create first REST web service
  - Service accepts integer ID of the student
  - Service should return student details
- Use proper HTTP methods for specific task
- Use proper JAX-RS annotation
- Use any IDE (NetBeans or Eclipse)

- *For deployment, use GlassFish server (Recommended)*