

RESTful Web Services (in Java)

Intertech's Oxygen Blast Series
April 2010

An Intertech Course ▶▶

Agenda

- Welcome to Intertech
- What are RESTful Web services?
 - Comparing/Contrasting with SOAP-based Web services
- How RESTful Web services work
- How RESTful Web services work in Java
 - JAX-RS
 - Jersey (JAX-RS) clients in Java
- Considerations before adopting REST
- Question/Answer Session
 - How to submit questions in Live Meeting

*10-15 Minute Break
somewhere
in the middle*



Available On Line

- Slides
- www.intertech.com/downloads/JAXRS-OxyBlast.pdf
- Demo Code
- www.intertech.com/downloads/JAXRS-OxyBlast.zip

Who is this guy?



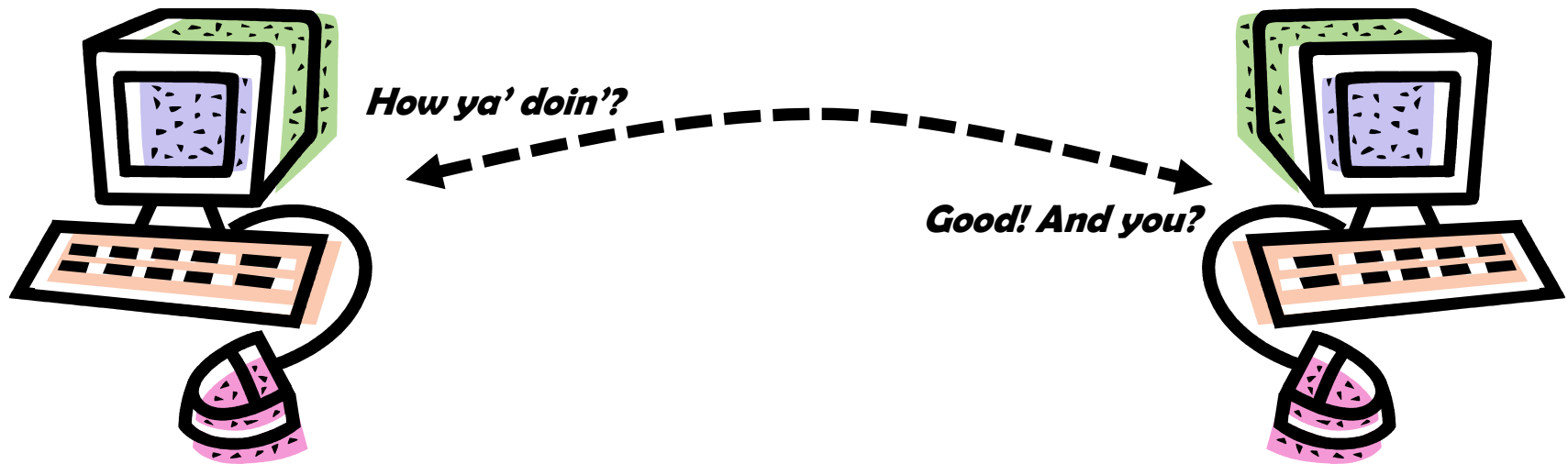
- Jim White – jwhite@intertech.com
- Intertech Instructor & Director of Training
- Author J2ME, Java in Small Things – 2002, Manning
- International Speaker
 - Including JavaOne
- Contributor to many journals including:
 - JDJ
 - DevX.com
 - JavaPro
- Consultant, engineer and architect with several companies

REST vs. SOAP Web Services

FUNDAMENTALS OF REST

Web Service

A software system designed to support interoperable machine-to-machine interaction¹.



¹http://en.wikipedia.org/wiki/Web_service

SOAP Web Services

- SOAP/XML-Based Web services
 - Complex
 - Exchange Bloat

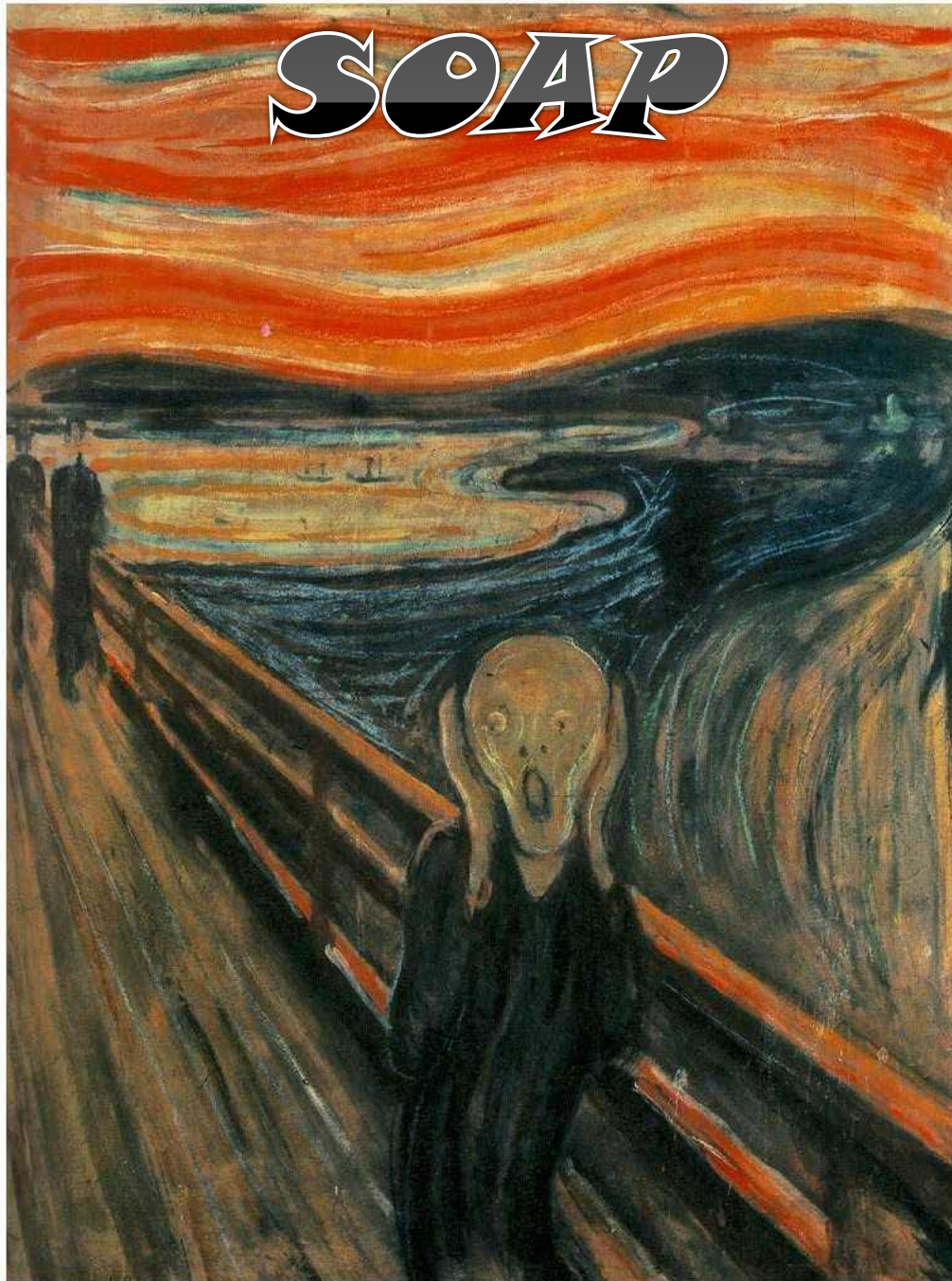
```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ws="http://ws.cdyne.com/"
  xmlns:StockService="http://www.cdyne.com/StockService"
  xmlns:LinkService="http://www.cdyne.com/LinkService">
  <StockService:LinkService>
    <LinkServiceRequest>
      <LinkServiceRequestKey>
        <LinkServiceRequestKey>
      </LinkServiceRequestKey>
    </LinkServiceRequest>
  </StockService:LinkService>
</soapenv:Envelope>
```

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ws="http://ws.cdyne.com/"
  xmlns:QuickQuoteService="http://www.cdyne.com/QuickQuoteService">
  <QuickQuoteService:QuickQuoteResponse>
    <QuickQuoteResponse>
      <QuickQuoteResult>50.64</QuickQuoteResult>
    </QuickQuoteResponse>
  </QuickQuoteService:QuickQuoteResponse>
</soap:Envelope>
```

WSDL
JAX-WS
SOAP
XML
DOM
JAX-RPC
XML Namespaces
SAX
Schema



SOAP



RESTful Web Services

- A SOAP Alternative
- Why?
 - Still language/platform independent = Interoperability
 - Simpler/Easier - based on well known Web paradigm
 - Bring back the “Web” in Web services
 - Can be more concise
 - Clients can choose their data format (XML, XHTML, JSON, etc.)
 - Cacheable results

SOA vs. other-OAs

- Comparisons to SOAP-based Web Services may be inappropriate. Both address particular needs.
 - SOAP based Web services are Service Oriented Architecture
 - SOA in WS-* form
 - An architecture loaded by standards
 - RESTful Web services are REST or Web Oriented Architecture²
 - ROA or WOA
 - An architectural style
 - WOA = SOA + REST + WWW
 - Web sites for machines vs. Web sites for humans



²WOA coined by Nick Gall of Gartner

Representation State Transfer

- REST = Representation State Transfer
 - Term and idea introduced in 2000 in PhD dissertation by Roy Fielding.
 - Fielding was a coauthor of HTTP.
 - Co-founder of Apache HTTP Server project.
 - REST concepts originated with WWW and pre-date SOAP Web services.

Resources

- At the heart of REST is the concept of a *Resource*.
- A resource is a thing – anything you can conceive.
 - “Any coherent and meaningful concept”
 - You, me, all of us
 - An account
 - An order
 - A stock
 - A health record
 - A hotel reservation
 - A book →

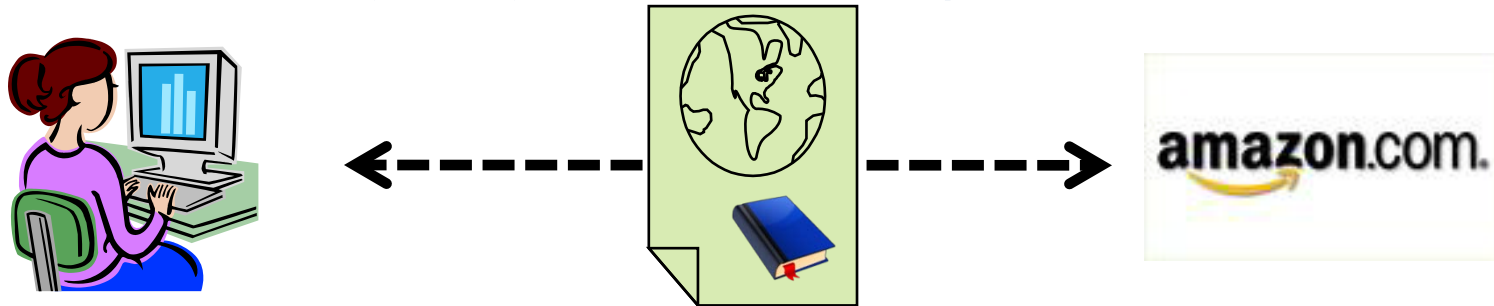


Resource = Noun

- RESTful Web service developers are fond of suggesting resources are typically expressed by nouns.

Representation

- A representation is a document that captures the current state of a resource.
- For example, an HTML Web page document with information (state) about a book you are interested in.



- So REST is about the *transfer* (request and response) of *representations of resources*

URI

- Each resource must be identified.
- Uniform Resource Identifiers (URIs) are used to identify resources.
- More specifically, URL's are used in Web-based REST systems.
 - They identify **where** and **what** resource is available.
 - <http://www.intertech.com/students/jamesBond>

The WWW and REST



- So, you already use REST today!
- When you are browsing the WWW, you request the representation of a resource state via URL.
- Resource state like:
 - Stock prices for stocks
 - Cost/availability of a room
 - Course catalog from a school
- Typically, you (or more precisely your browser) asks for the representation in the form of:
 - HTML
 - Images
 - Other MIME types.

Representation Formats

- While humans are interested in browser understood and displayable representations, machines are not.
- Machines are more typically interested in more structured document representations of resources.
 - XML
 - XHTML
 - JavaScript Object Notation (JSON)
 - Could even be SOAP (but not usually)
- The type or *format* of requested representation is often suggested in the REST request URL.
 - <http://www.intertech.com/students/jamesBond/json>

REST Verbs

- When doing Web-based REST, HTTP methods are used to identify what you want done with/to the resource state.
 - GET, the most common, suggests the client would like to receive the latest representation of the resource state.
 - POST provides the state for the creation of a new resource.
 - Typically, the state data is provided in body of the HTTP POST request message.
 - DELETE requests the removal of a resource.
 - PUT provides the new state for the update of a resource's state.
 - Again, typically, state data is also provided in the body of the HTTP PUT request message.
- RESTful developers akin the HTTP methods to the verbs acting on the nouns (resources)

Some REST Examples

- Stock Quote fetch (XML)
 - <http://ws.cdyne.com/delayedstockquote/delayedstockquote.asmx/GetQuote?StockSymbol=MMM&LicenseKey=0>
 - <http://ws.cdyne.com/delayedstockquote/delayedstockquote.asmx/GetQuote?StockSymbol=GOOG&LicenseKey=0>
- GeoName – Country Data fetch
 - XML
 - <http://ws.geonames.org/countryInfo?country=us>
 - <http://ws.geonames.org/countryInfo?country=es>
 - JSON
 - <http://ws.geonames.org/countryInfoJSON?country=us>
 - <http://ws.geonames.org/countryInfoJSON?country=fr>
 - CSV
 - <http://ws.geonames.org/countryInfoCSV?country=gb>

JAX-RS

RESTFUL IN JAVA

RESTful Web Services in Java

- Implementation Options

- Servlets

- Roll your own
 - Restlet Open Source project

- JAX-WS

- Was designed to process SOAP messages.
 - Retrofitted to allow most of the SOAP message processing to be bypassed. Consider a clumsy and inconvenient solution.

- JAX-RS

- It is a relatively recent addition to the Java community. The final release of the specification (JAX-1.0) occurred in October 2008.
 - JAX-RS maps URIs, HTTP methods and MIME content type negotiation to annotated POJOs.

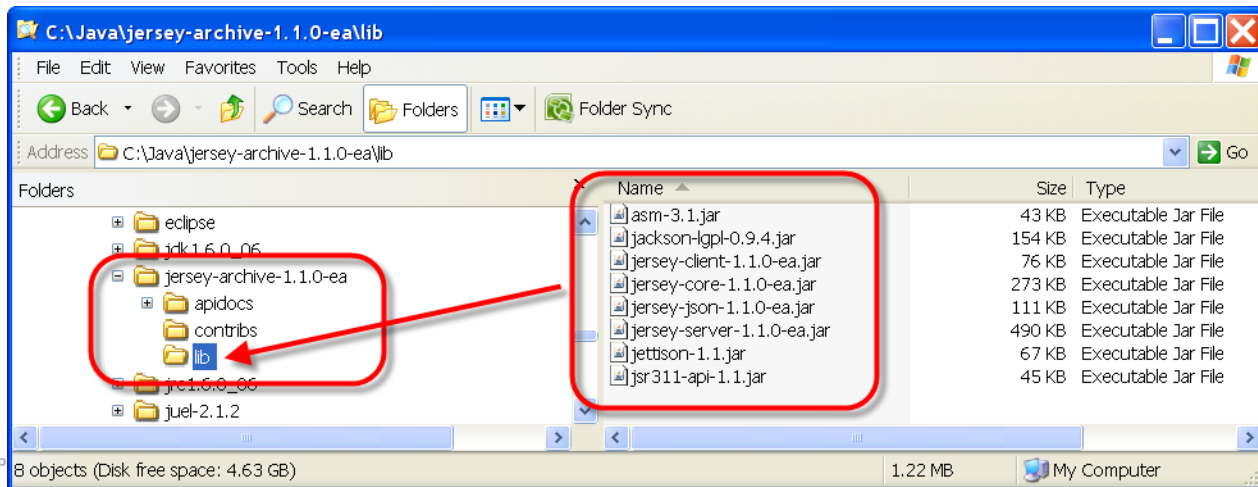


Jersey

- As with all Java specifications, JAX-RS must be implemented.
- The Jersey project (<https://jersey.dev.java.net/>) is Sun's reference implementation of JAX-RS 1.0.
- Jersey is already part of Glassfish.
- Jersey (and by default the JAX-RS specification/API) are part of Java EE 6.
- Jersey is not, however, the only JAX-RS implementation.
- Resteasy is an implementation of JAX-RS by JBoss.

Jersey Setup

- As Jersey is not a part of most Java runtime environments at this time, you must first get Jersey and make it a part of your project.
 - Download Jersey from <https://jersey.dev.java.net/>.
 - The download includes several Jersey libraries (see the lib folder) to include the Jersey core, core server, Jersey client, etc.
 - Jersey is also dependent on many third party jars too like asm.jar (a bytecode manipulation framework) and jettison.jar (APIs which read and write JSON).
 - The required JAR files are all in the downloaded zip file.



Jersey Setup Cont.

- With Jersey added to your project, you must add the Jersey adapter servlet to your web.xml file.
 - The adapter servlet routes REST traffic to the appropriate RESTful Web service POJO/JavaBean.
 - Open your application's web.xml file and add the following `<servlet>` element.

```
<servlet>
  <servlet-name>Jersey Adapter Servlet</servlet-name>
  <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
</servlet>
```

Jersey Setup Cont.

- You must also add a `<servlet-mapping>` element to map the appropriately designated requests to the servlet and thus to the RESTful Web service beans.
 - The URL pattern is up to you, however many developers choose the `/resources/*` pattern (as shown below) which aligns with the idea of requesting REST resources.

```
<servlet-mapping>  
  <servlet-name>Jersey Adapter Servlet</servlet-name>  
  <url-pattern>/resources/*</url-pattern>  
</servlet-mapping>
```

- An additional `<init-param>` element must also be added to the `<servlet>` element above.
- However, a better understanding of the RESTful services is needed before adding this element.

JAX-RS Services

- Using JAX-RS, REST resources are Plain Old Java Objects.
 - The POJOs are annotated with `@Path`.
 - `@Path` takes one argument. The argument identifies the relative request URI path to which the resource responds.

```
@Path("/helloworld")
```

```
public class HelloWorldResource {  
    ...  
}
```

- The `@Path` annotation and most annotations discussed here can be found in the `javax.ws.rs` package.

RESTful Service Path

- The URI path to the RESTful POJO is relative to the base URI of the server, context root of the WAR, and the URL pattern for the Jersey adapter servlet.

```
http://<server>:<port>/<context>/<Jersey servlet mapping>/<Path>
```

- So, assuming the POJO was in a MyWebApp WAR, a URL request to this HelloWorldResource might look like the following:

```
http://localhost:8080/MyWebApp/resources/helloworld
```

- Paths are automatically encoded. So a `@Path("/hello world")` is the same as `@Path("/hello%20world")`.
- Paths may use wild cards and regular expressions to assist in mapping.

RESTful Operations

- Add annotations (@GET, @POST, @PUT, @DELETE, @HEAD) to the resource methods to indicate what should be invoked for each type of HTTP request.

```
@Path("/helloworld")
public class HelloWorldResource {
    @GET
    public String sayHello() {
        return "Hello World";
    }
    ...
}
```

- Methods that are annotated with @GET, @POST, @PUT, @DELETE and @HEAD are called resource methods.
- These methods must be declared public.
- If a resource method returns void, the response to the client will be an empty HTTP method body with a 204 status code (request successfully processed).

DEMO

Hello World – REST style

An Intertech Course ▶▶

RESTful Operations Cont.

- By default, the resource returns text/plain. However, the `@Produces` annotation can be used to specify the MIME type of response.
 - The `@Produces` can annotate the resource to provide the default return type for all resource methods.

```
@Path("/helloworld")
@Produces("text/html")
public class HelloWorldResource {
    ...
}
```

- It can also be placed on the resource methods to override any class `@Produces` setting.

```
@Path("/helloworld")
@Produces("text/html")
public class HelloWorldResource {
    @GET
    @Produces("text/plain")
    public String sayHello() {
        return "Hello World";
    }
    ...
}
```

Representations

- Resources may offer multiple *MIME* types for any given request.

```
@Path("/helloworld")
public class HelloWorldResource {
    @GET @PATH("/text")
    @Produces("text/html")
    public String sayHello() {
        ...
    }
    @GET @PATH("/json")
    @Produces("application/json")
    public String sayJsonHello() {
        ...
    }
}
```

- If a resource can produce more than one *MIME* type, the resource method that corresponds to the most acceptable type as requested by the client will be invoked.
- The `accept` header of the HTTP request declares what is most acceptable on the part of the client.

Representations

- Resources may also be sent information (as part of the HTTP request body) by the client.
 - The `@Consumes` annotation works in a fashion similar to `@Produces` but for incoming rather than outgoing MIME types.
 - `@Consumes` specifies which MIME types can be accepted or consumed by the resource.
 - Like `@Produces`, it can be specified on the class or method level (where it overrides the class annotation setting).
 - If a resource is unable to consume the MIME type of a client request, the runtime sends back an HTTP 415 error (Unsupported Media Type).

```
@POST
@Consumes("text/plain")
public void respondToMessage(String message) {
    ...
}
```

Path Parameters

- The URI path may contain variables. The variables are substituted at runtime in order to provide the resource with additional information.
 - Variables are denoted by curly braces in the `@Path` annotation.

```
@Path("/helloworld/{username}")  
public class HelloWorldResource {  
    ...  
}
```

- Make sure your servlet mapping takes into account any extra data passed via the URL parameters.

```
<url-pattern>/resources/*</url-pattern>
```

- Without the "*" in the url-pattern, the additional data is interpreted as part of the URL causing the request to miss its intended target resource.

Path Parameters Cont.

- Regular expressions can be used in the variable declaration to constrain valid character data.

```
@Path("/helloworld/{username: [A-Z][a-zA-Z]}")
```

- In this example, the username must start with an uppercase character and then include any number/any case of characters after that.
 - If, on request of the resource, the username provided in the URL does not match the regular expression template then a 404 (Not Found) error is the response.
 - In other words, sending the wrong data is as good as if the resource is not there.
-
- If multiple variables are needed, separate the curly braced variables with a `"/"`.

```
@Path("/helloworld/{firstname}/{lastname}")
```

Path Parameters Cont.

- In order to extract the data from a path variable (a.k.a. path parameter), use the `@PathParam` annotation in the resource method where they are needed.

```
@Path("/helloworld/{username}")
public class HelloWorldResource {
    @GET
    @Produces("text/plain")
    public String sayHello(@PathParam("username") String username) {
        return "Hello " + username;
    }
    ...
}
```

- If the `@PathParam` variable cannot be cast to the specified method parameter type, the runtime returns an HTTP 404 error (Not Found) to the client.

Query String Parameters

- Query string parameters can be extracted with the `@QueryParam` annotation.
 - In a fashion similar to the use of `@PathParam`, use the `@QueryParam` annotation in front of any method parameter.

```
@Path("/helloworld")
public class HelloWorldResource {
    @GET
    @Produces("text/plain")
    public String sayHello(@QueryParam("username") String username) {
        return "Hello " + username;
    }
    ...
}
```

- The method parameter will be set to the query string parameter by the same name.

<http://localhost:8080/MyWebApp/resources/helloworld?username=Jim>

Other Parameters

- In fact, there are six types of parameters that you can extract data from using annotations.
 - You have already seen path parameters and query parameters.
 - In addition, you can extract data from form parameters, cookie parameters, header parameters and matrix parameters using the following annotations.

Parameter	JAX-RS Annotation
HTML form parameter	@FormParam
HTTP Header parameter	@HeaderParam
HTTP Cookie parameter	@CookieParam
Matrix URL parameter	@MatrixParam

JAX-RS, JAXB and JSON

- The Java Architecture for XML Binding (JAXB) allows Java objects to be marshaled and unmarshaled in/out of XML.
 - It is used heavily in JAX-WS for SOAP-based Web services.
 - JAXB annotations can be used with JAX-RS to marshal and unmarshal your Java types to XML for RESTful Web services producing or using XML.
- For example, consider the following JAXB annotated type.

```
@XmlElement(name="student")  
public class Student {  
    private int id;  
    private String name;  
    private String grade;  
    //getters and setters removed for brevity  
}
```

JAX-RS, JAXB and JSON Cont.

- In order to return JAXB produced XML documents of the Student type, simply have your RESTful service produce an instance of that type.

```
@Path("/students")
public class StudentResource {
    @GET
    @Produces("application/xml")
    public Student getStudent(@QueryParam("id") int studentId) {
        Student student = //work to retrieve Student object by id
        return student;
    }
}
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <student>
  <grade>A+</grade>
  <id>12</id>
  <name>James Bond</name>
</student>
```

- Likewise, use any of the parameter annotations with a Student type parameter to have JAX-RS use JAXB to unmarshal XML data into object form.

JAX-RS, JAXB and JSON Cont.

- In a similar fashion, JAX-RS provides libraries to automatically marshal and unmarshal in JSON format!
 - Just change the `@Produces` to `"application/json"` and magically JAX-RS provides the same data in JSON format.

```
@GET
```

```
@Produces("application/json")
```

```
public Student getStudent(@QueryParam("id") int studentId) {
```

```
    Student student = //work to retrieve Student object by id
```

```
    return student;
```

```
}
```

```
{"grade":"A+","id":"12","name":"James Bond"}
```

- Some of the other JAX-RS implementations support additional representations.
 - For example, Resteasy supports Atom.

JSON Sidebar

- BTW - What is JSON?
- JSON stands for JavaScript Object Notation.
 - It is a lightweight computer data interchange format.
 - Like XML, it is a text-based, human-readable format.
 - Many prefer JSON over XML because the same data can be exchanged with fewer characters.
 - Typically, JSON is faster and easier to parse and work with.
 - Since many developers see RESTful as a lighter weight means of providing Web services, not surprisingly, JSON is an often favored data exchange format.

JSON Example

- To provide you a glimpse of the JSON format, here is a golfer described in XML.

```
<golfer>
  <person id="777-99-8888">
    <first>Jim</first>
    <last>White</last>
  </person>
  <handicap>
    <course id="1234">
      <name>Oneka Ridge</name>
      <rating>129/71.2</rating>
    </course>
    <index>12</index>
  </handicap>
</golfer>
```

JSON Example Cont.

- And now the same golfer describe via JSON.

```
{golfer:{
  person:{
    id:'777-99-8888',
    first:'Jim',
    last:'White'},
  handicap:{
    course:{
      id:1234,
      name:'Oneka Ridge',
      rating:'129/71.2'},
    index:12} }
}
```

- There are Java API's for parsing/working with JSON.
- There are several on-line resources for learning more about JSON.

Resource Lifecycle

- By default, the JAX-RS environment creates an instance of the resource (the JAX-RS POJO) for each request.
- The resource can be configured for two other lifecycles.
 - Use the `@Singleton` annotation on the resource to indicate a single instance should be reused for all requests.

```
import com.sun.jersey.spi.resource.Singleton;  
@Path("/helloworld")  
@Singleton  
public class HelloWorldResource {  
    ...  
}
```

Resource Lifecycle Cont.

- Use `@PerSession` to indicate that a new instance should be created and reused for each client session.

```
import com.sun.jersey.spi.container.servlet.PerSession;  
@Path("/helloworld")  
@PerSession  
public class HelloWorldResource {  
    ...  
}
```

- It should be noted that these **last two annotations are Jersey implementation specific** and not defined in JAX-RS.
- Note the `com.sun.jersey` package names!

JAX-RS Resource Deployment

- Once you have your JAX-RS resource constructed, you must inform the JAX-RS adapter servlet of your resource.
 - There are several options for identifying resources.
 - The simplest way is to provide an `<init-param>` to the Jersey Adapter Servlet.
 - The `init-param` provides the adapter with a list of packages containing JAX-RS annotated resources.

```
<servlet>
  <servlet-name>Jersey Adapter Servlet</servlet-name>
  <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
  <init-param>
    <param-name>com.sun.jersey.config.property.packages</param-name>
    <param-value>com.intertech.orders.rest;com.intertech.accounts.rest</param-value>
  </init-param>
</servlet>
```

Demo

Contact List Application– REST style

An Intertech Course ▶▶

Simple Contact List Application in JAX-RS
Access and update the database via XML and JSON

Jersey/JAX-RS Demos

- Hello World in JAX-RS
- Simple Contact List Application in JAX-RS
 - Access and update the database via XML and JSON
- All code available at:
www.intertech.com/downloads/JAXRS-OxyBlast.zip
 - Link is on the resource list given to you.

RESTful clients

JERSEY CLIENT API

JAX-RS Clients

- The JAX-RS specification does not specify how to build a client.
 - In Java, one could use existing client APIs such as `java.net.URL`, `URLConnection` or `HttpClient`.
 - Of course since REST operates simply over the Web using HTTP, etc., a simple browser can be used to test RESTful services.
 - However, many of the JAX-RS implementations provide a client API.

JAX-RS Clients Cont.

- The Jersey JAX-RS implementation comes with a client API.
 - Here is a small sample application to call on the HelloWorldResource.

```
import javax.ws.rs.core.MultivaluedMap;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.core.util.MultivaluedMapImpl;
public class StudentClient {
    public static void main(String[] args) {
        Client client = Client.create();
        WebResource webResource = client.resource(
            "http://localhost:9082/MyWebApp/resources/helloworld");
        MultivaluedMap queryParams = new MultivaluedMapImpl();
        queryParams.add("username", "James Bond");
        String s = webResource.queryParams(queryParams).get(String.class);
        System.out.println("Your RESTful response: " + s);
    }
}
```

JAX-RS Clients Cont.

- Jersey's Client object is the main configuration point for building a RESTful web service client.
- WebResource, as its name implies, represents the RESTful resource to the client.
- The client API comes replete with capability to pass all types of parameters and data in the body of the request message and to extract data from the response.
- Notice the Jersey packages. Again, the **client side API is not by specification.**

No Free Lunch

BEFORE ADOPTING REST

No REST for the Weary

- REST is not a standard. REST follows conventions.
 - It uses well understood paradigms and standards of the Web (HTTP, URLs, etc.)
 - Your REST may not be my REST.
- Assumes point-to-point communications.
 - It is a client/server model.
 - Doesn't support distributed computing model well.
- Generally speaking, it's bound to HTTP.
 - Fielding's paper didn't say it had to be.
 - Supporting WWW infrastructure makes it difficult to use elsewhere.

No REST for the Weary Cont.

- Not everything fits well into GET/POST/... (CRUD) activity.
 - REST “Verbs” don’t always fit additional application semantics.
 - Can be overcome, but not always clean.
- Asynchronous activity – ???
 - Clients can send callback location
 - Services provide drop ship location
- Transactions – ???
- Security - ???
 - HTTPS anyone?
- Service Descriptions and Registries (a la WSDL and UDDI)
 - WSDL – clumsy
 - WADL – Web Application Description Language

Resource List

- See Jim's big list-o-RESTful resources.
- RESTful basics:
www.ibm.com/developerworks/webservices/library/ws-restful/
- JAX-RS JSR Page: <https://jsr311.dev.java.net/>
- O'Reilly Book on RESTful Java:
<http://oreilly.com/catalog/9780596158057>
- Jersey Web Page: <https://jersey.dev.java.net/>
- JAX-RS with Jersey: www.javaworld.com/community/?q=node/4036
- RESTEasy: <http://www.jboss.org/resteasy>
- Intertech's Complete Java Web Services class
- jwhite@intertech.com
- Follow Intertech on   

Intertech Resources

- Intertech offers free:
 - Content packed newsletters
 - Podcasts through iTunes
 - YouTube videos
 - Free Oxygen Blast seminars
 - Whitepapers and .pdf downloads

- For the above and special offers, register for the newsletter at the bottom of our homepage



Intertech Training

- Founded in 1991, Intertech offers a full training line-up:
 - JEE, open source technologies
 - .NET, SQL Server, SharePoint
 - XML, Ajax
- Delivery formats include:
 - Instructor-led public and onsite
 - Instructor-led night and virtual
 - Self-paced study
- For advanced purchase customers, Intertech offers Elite Rewards™ — call 651-994-8558 +23



Call 800 866 9884 +23 to begin your Elite Rewards membership.

Intertech Consulting

- In addition to training, Intertech delivers consulting
- Consulting is part of our brand:
*Instructors Who Consult |
Consultants Who Teach*
- Give your project success with our consulting services
- To learn more, contact us at
651-994-8558 +11



What you have learned today

- ✓ How RESTful Web services are defined
 - ✓ What separates RESTful Web services from SOAP-based Web services.
- ✓ How RESTful Web services work.
- ✓ How RESTful Web services work in Java.
 - ✓ Java implementations/APIs
 - ✓ Tools/Frameworks
 - ✓ Jersey (JAX-RS) clients in Java.
- ✓ What to consider before adopting RESTful Web services

Q & A

An Intertech Course ▶▶

Thanks for coming!

An Intertech Course ▶▶