The Java™ API for XML Web Services (JAX-WS) 2.0

Roberto Chinnici/ Rajiv Mordani/ Doug Kohlert

Senior Staff Engineers
Sun Microsystems, Inc.
http://java.sun.com/webservices/jaxws/

Session 1194

Learn about the next generation of Web services technologies for the Java™ platform

Agenda

Introduction to Java™ API for XML—Web Services 2.0

Annotation-based programming model

Advanced features: The messaging layer

Implementing web service endpoints on the Java™ SE platform

Demo

Conclusion

Agenda

Introduction to Java™ API for XML—Web Services 2.0

Annotation-based programming model

Advanced features: The messaging layer

Implementing web service endpoints on the Java™ SE platform

Demo

Conclusion

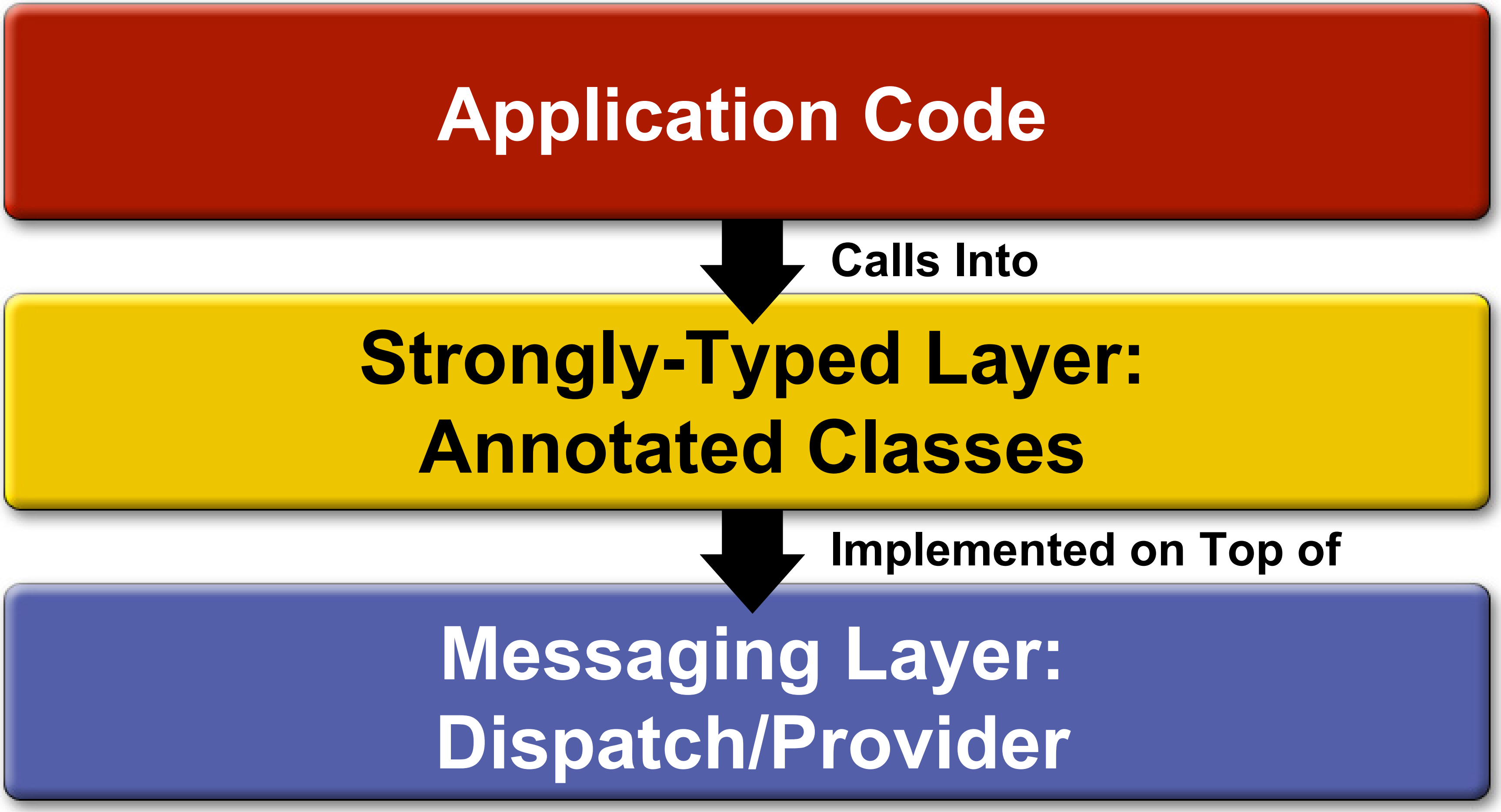
JAX-WS 2.0

- New, easy to use web services API
- Embrace plain old Java object (POJO) concepts
- Descriptor-free programming
- Layered architecture
- Protocol and transport independence
- Integrated data binding via Java Architecture for XML Binding (JAXB) 2.0
- Part of Java SE 6 and Java EE 5 platforms

Interoperability

- Standards-compliant
  - W3C/WS-I SOAP 1.1/1.2, WSDL 1.1, BP 1.0/1.1
- Foundation for full WS-* web services stack
  - Project Tango
  - Interoperability testing with Windows Communication Foundation (WCF, aka “Indigo”)
- New JSRs adding support for more WS-* technologies over time
  - JSR 261 Java API for XML Web Services Addressing (JAX-WSA)
  - JSR 265 API for Utilizing Web Services Policy

Layered Architecture



Application Code

Calls Into

Strongly-Typed Layer: Annotated Classes

Implemented on Top of

Messaging Layer: Dispatch/Provider

What Does It Mean?

- Upper layer uses annotations extensively
  - Easy to use
  - Great toolability
  - Fewer generated classes
- Lower layer is more traditional
  - API-based
  - For advanced scenarios
- Most application will use the upper layer only
- Either way, portability is guaranteed

Agenda

Introduction to Java™ API for XML—Web Services 2.0

Annotation-based programming model

Advanced features: The messaging layer

Implementing web service endpoints on the Java™ SE platform

Demo

Conclusion

Server-Side Programming Model

1. Write a POJO implementing the service
2. Add @WebService to it
3. Optionally, inject a WebServiceContext
4. Deploy the application
5. Point your clients at the WSDL
   - e.g. http://myserver/myapp/MyService?WSDL

Example: Servlet-Based Endpoint

```
@WebService
public class Calculator {
    public int add(int a, int b) {
        return a+b;
    }
}
```

- All public methods become web service operations
- Default values for service name, etc.
- WSDL/ Schema generated automatically

Example: Enterprise JavaBeans™ 3.0-Based Endpoint

```
@WebService
@Stateless
public class Calculator {
    @Resource
    WebServiceContext context;

    public int add(int a, int b) {
        return a+b;
    }
}
```

- It’s a regular EJB 3.0 component, so it can use any EJB features
  - Transactions, security, interceptors...

Infinite Customizability

Via Annotations

```
@WebService(name=”CreditRatingService”,
          targetNamespace=”http://example.org”)
  public class CreditRating {

    @WebMethod(operationName=”getCreditScore”)
    public Score getCredit(
        @WebParam(name=”customer”)
        Customer c) {
      // ... implementation code ...
   }
}
```

Data Binding

JAXB Integrated With JAX-WS

- Lower layer in JAX-WS
- JAX-WS 2.0 delegates all data binding functionality to JAXB 2.0
- One mapping, one set of annotations
- XML Schema 100% supported
- Attachment support (MTOM/XOP)
- Richer type mapping via Java API for XML Processing (JAXP)
    - e.g. javax.xml.datatype.XMLGregorianCalendar

Data Binding Tips

- Use regular Java classes as data types
- Follow JavaBeans™-based property pattern:

```
public String getName() { ... }
public void setName(String name) {...}
```

- Or use public fields:

```
public String name;
```

- Use enumerated types and collections:

```
public enum Color {RED, WHITE, BLUE};
public Color garmentColor;
public List<Person> contacts;
```

Java SE Client-Side Programming

1. Point a tool at the WSDL for the service
   ```
   wsimport http://example.org/calculator.wsdl
   ```
2. Generate annotated classes and interfaces
3. Call new on the service class
4. Get a proxy using a getPort method
5. Invoke any remote operations

Example: Java SE-Based Client

Look, Mom... No Factories!

```
CalculatorService svc = new CalculatorService();
Calculator proxy = svc.getCalculatorPort();
int answer = proxy.add(35, 7);
```

- No factories yet the code is fully portable
  - CalculatorService is defined by the specification
  - Internally it uses a delegation model

Java EE Client-Side Programming

1. Point a tool at the WSDL for the service

```
wsimport http://example.org/calculator.wsdl
```

2. Generate annotated classes and interfaces
3. Inject a WebServiceReference of the appropriate type
4. Invoke any remote operations

Example: Java EE-Based Client

Still No Factories and No Java Naming and Directory Interface™ API Either!

```
@Stateless
public class MyBean {

   @WebServiceRef(CalculatorService.class)
   Calculator proxy;

   public int mymethod() {
      return proxy.add(35, 7);
}
```

Can I Rename The Generated Classes?

Using The Binding Customization Language

- You can rename pretty much everything
- When you run the tool to import a WSDL, specify some customizations
- Customizations are written in XML
- Two models:
  - Embedded in WSDL/Schema
  - As a separate customization file
- JAXB customizations work the same way

Example: Customization File

```
<jaxws:bindings
       wsdlLocation="http://example.org/calculator.wsdl">
  <jaxws:package name="org.example.calculator"/>
  <jaxws:bindings
     node="wsdl:portType[@name=’Calculator’]">
    <jaxws:bindings node="wsdl:operation[@name=’add’]">
      <jaxws:method name="performAddition"/>
    </jaxws:bindings>
  </jaxws:bindings>
    ...additional binding declarations....
</jaxws:bindings>
```

Protocol and Transport Independence

No SOAP In Sight

- Typical application code is protocol-agnostic
- Default binding in use is SOAP 1.1/HTTP
- Server can specify a different binding, e.g.

  `@BindingType(SOAPBinding.SOAP12HTTP_BINDING)`
- Client must use binding specified in WSDL
- Bindings are extensible, expect to see more of them
  - e.g. SOAP/Java Message Service(JMS) or XML/SMTP

Using the SOAP 1.2 or REST Binding

```
@WebService
@BindingType(SOAPBinding.SOAP12HTTP_BINDING)
public class Calculator {
    public int add(int a, int b) {
        return a+b;
    }
}

@WebService
@BindingType(HTTPBinding.HTTP_BINDING)
public class Calculator {
    public int add(int a, int b) {
        return a+b;
    }
}
```

Agenda

Introduction to Java™ API for XML—
Web Services 2.0

Annotation-based programming model

Advanced features: The messaging layer

Implementing web service endpoints on the Java™ SE platform

Demo

Conclusion

Messaging in JAX-WS 2.0

- Lower layer in JAX-WS
- Mostly out of view until you need it
- Many more control knobs → more complexity
- Motivated by advanced applications:
  - Dynamic clients (e.g. a management console)
  - Dynamic servers (e.g. a gateway)
  - Protocols without an established description language

What Is a WebServiceContext?

@Resource WebServiceContext Context;

- Used on the server
- WebServiceContext gives access to
  - Security information (e.g. getUserPrincipal method)
  - The message context
  - In the future, other information on the client/service
- MessageContext is a bag of properties
  - Data which is not part of the XML payload for a message ends up in the context
  - E.g. HTTP query string

    http://myserver/myapp/MyService?format=image/jpeg

Breaking Up a Request Message

HTTP Headers Method Query String

Message Context Properties

javax.xml.ws.http.request.headers
javax.xml.ws.http.request.method
javax.xml.ws.http.request.querystring

Primary MIME Part
or
HTTP Request Body

Method Arguments

void foo(Bar b);

Attachments
(If Any)

Message Context Property

javax.xml.ws.binding.attachments.inbound

Client-Side RequestContext

BindingProvider.getRequestContext()

- Bag of properties for use by the application
- Any data is copied to the message context before each invocation
- Useful to configure a message on-the-fly
  - Endpoint address
  - Username/password
  - Attachments
  - HTTP query string
  - SOAP action...

Example: Accessing a REST Service

The Short Version

1. Create a service
2. Add a HTTP port to it
3. Create a Dispatch object for that port
4. Call the invoke method
5. Process the result

Example: Accessing a REST Service

The Code

```
String ns = "urn:yahoo:yn";
QName serviceName = new QName("yahoo",nsURI.toString());
QName portName = new QName("yahoo_port",nsURI.toString());
Service s = Service.create(serviceName);
String address =
"http://api.search.yahoo.com/NewsSearchService/V1/newsSear
ch?appid=jaxws_restful_sample&type=all&results=10&sort=dat
e&language=en&query=java";
s.addPort(portName, HTTPBinding.HTTP_BINDING, address);
Dispatch<Source> d = s.createDispatch(portName,
Source.class, Service.Mode.PAYLOAD);
Map<String, Object> requestContext=d.getRequestContext();
requestContext.put(MessageContext.HTTP_REQUEST_METHOD,
               "GET");
Source result = d.invoke(null);
// use the source...
```

Benefits Over Network APIs

- Higher-level (no sockets)
- JAXB support built in
- No need to parse MIME multipart packages
- Can plug in message handlers
- Bindings get tested for interoperability
- Extensible to new protocols/transports

Client-side Messaging API: Dispatch

```
// T is the type of the message
public interface Dispatch<T> {

   // synchronous request-response
   T invoke(T msg);

   // async request-response
   Response<T> invokeAsync(T msg);
   Future<?> invokeAsync(T msg, AsyncHandler<T> h);

   // one-way
   void invokeOneWay(T msg);
}
```

Choosing a Message Type

1. Do you want to see the whole protocol message?

   If yes, use MESSAGE mode and the appropriate message type (e.g. SOAPMessage for SOAP 1.1/1.2)

   If not, use PAYLOAD mode and answer the next question

2. Do you want to use JAXB?

   If yes, use java.lang.Object

   Otherwise use javax.xml.transform.Source

3. Pass message type and mode to:

   ```
   Service.createDispatch(port, type, mode)
   ```

Examples

Dispatch<T> → T invoke(T msg) “T in, T out”

- Payload mode with JAXB:

```
Dispatch<Object>
```

- SOAP Message mode:

```
Dispatch<SOAPMessage>
```

- HTTP binding payload mode without JAXB:

```
Dispatch<Source>
```

- HTTP binding message mode:

```
Dispatch<DataSource>
```

Server-side Messaging API: Provider

```
// T is the type of the message
public interface Provider<T> {

   T invoke(T msg, Map<String,Object> context);

}
```

- The same considerations for mode and message type apply here
- Use @ServiceMode to select a mode

Example: Payload Mode, No JAXB

```
@ServiceMode(Service.Mode.PAYLOAD)
public class MyProvider
       implements Provider<Source> {
  public Source invoke(Source request,
                       Map<String,Object> context) {
   // process the request using XML APIs, e.g. DOM
   Source response = ...

   // return the response message payload
   return response;
  }
}
```

Example: Message Mode, SOAP 1.1/1.2

```
@ServiceMode(Service.Mode.MESSAGE)
public class SOAPProvider
       implements Provider<SOAPMessage> {
  public SOAPMessage invoke(SOAPMessage request,
                       Map<String,Object> context) {
   // process the request using SAAJ
   SOAPMessage response = ...

   // return the response message payload
   return response;
  }
}
```

Message Handlers

An Application-Structuring Device

- Different degrees of visibility into messages:
  - Protocol handlers see the protocol message
  - Logical handlers see XML data but not the message
  - Application code sees Java objects (usually)
- Golden rule: Place the code in the safest place where it can still do its job
  - Example: do not give a SOAP message to application code that only needs unmarshalled Java objects
- Fewer bugs, fewer headaches

Handler API

Parameterized in the Kind of Messages it Can Handle

```
public interface Handler<C extends MessageContext> {

    // handle a regular message
    boolean handleMessage(C context);

    // handler a fault message (e.g. SOAP fault)
    boolean handleFault(C context);

    // called to terminate an exchange
    void close(C context);
}
```

Handler Decision Process

If Your Code Deals With an Aspect of Your Application…

1. Does it need to see the whole protocol message?

   Use a protocol handler

2. Does it need to see the XML payload for a message?

   Use a logical handler

3. (EJB™ technology only) Does it need to see Java objects?

   Use an interceptor

4. Otherwise, put it in the endpoint class

Agenda

Introduction to Java™ API for XML—Web Services 2.0

Annotation-based programming model

Advanced features: The messaging layer

Implementing web service endpoints on the Java™ SE platform

Demo

Conclusion

Web Service Endpoints on the Java SE Platform

- New in Mustang
- Endpoint classes are annotated POJOs
- Application creates an instance and publishes it
- Easy and error-free
- Lots of defaults applied automatically
  - WSDL, data binding, port number, threading...

Publishing a POJO

```
@WebService
public class Calculator {
    @Resource
    WebServiceContext context;

    public int add(int a, int b) {
        return a+b;
    }
}

// create and publish an endpoint
Calculator calculator = new Calculator();
Endpoint endpoint =
   Endpoint.publish(“http://localhost/calculator”,
                    calculator);
```

What Happens Behind the Scenes

1. Endpoint implementation class is introspected
2. Annotations are found and processed
3. A WSDL is generated automatically

   If you want manual generation, you can do:

   ```
   wsgen -cp calc.jar org.example.Calculator
   ```
4. HTTP server in Mustang is started (if needed)
5. Context is injected
6. New endpoint is active

Endpoint.publish Is All it Takes!

- Really!
- Simple HTTP server embedded in Mustang
- Reasonable defaults for threading, etc.
- WSDL created and published on the fly:

  `http://localhost/calculator?WSDL`

- Optionally, applications can control low-level functionality, e.g.
  - Threading via an Executor object
  - WSDL/XML Schema via metadata

Using the SOAP 1.2 or REST Binding

```
@WebService
@BindingType(SOAPBinding.SOAP12HTTP_BINDING)
public class Calculator {
    public int add(int a, int b) {
        return a+b;
    }
}

@WebService
@BindingType(HTTPBinding.HTTP_BINDING)
public class Calculator {
    public int add(int a, int b) {
        return a+b;
    }
}
```

“Doesn’t That Make My Endpoint Class Protocol-Specific?”

- Choose a binding when you create the endpoint

```
@WebService
public class Calculator {
    public int add(int a, int b) {
        return a+b;
    }
}

Calculator calculator = new Calculator();
Endpoint endpoint =
   Endpoint.create(SOAPBinding.SOAP12HTTP_BINDING,
                                    calculator);
endpoint.publish(“http://localhost/calculator”);
```

Publishing a Specific WSDL/ Schema

- Just put the WSDL and any schemas it refers to inside your application jar

  /META-INF/wsdl/my.wsdl
  /META-INF/wsdl/schema1.xsd
  /META-INF/wsdl/schema2.xsd...

```
@WebService(wsdlLocation=”/META-INF/wsdl/my.wsdl”)
public class Calculator {
    public int add(int a, int b) {
        return a+b;
    }
}
```

Agenda

Introduction to Java™ API for XML—Web Services 2.0

Annotation-based programming model

Advanced features: The messaging layer

Implementing web service endpoints on the Java™ SE platform

Demo

Conclusion

DEMO

Web Services on Project GlassFish and Mustang

Agenda

Introduction to Java™ API for XML—
Web Services 2.0

Annotation-based programming model

Advanced features: The messaging layer

Implementing web service endpoints on the Java™ SE platform

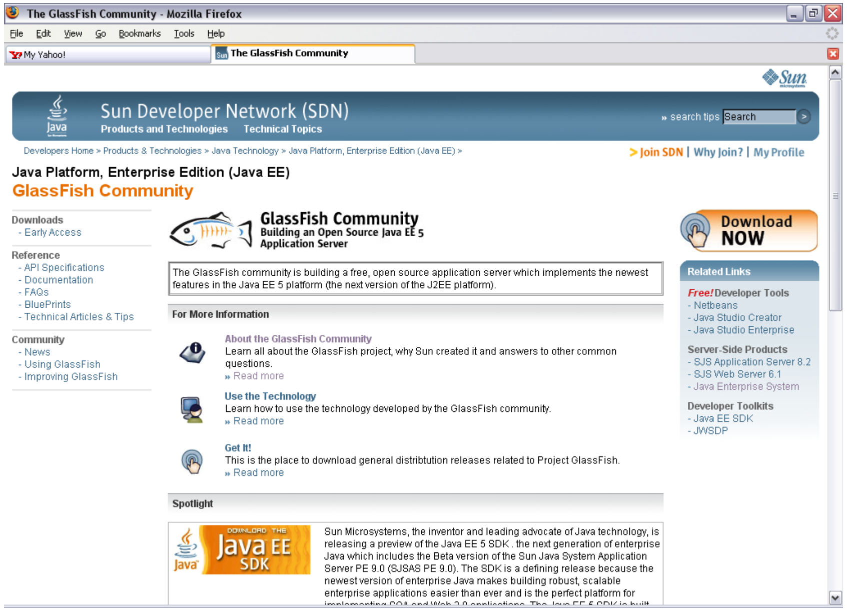
Demo

Conclusion

Summary

- JAX-WS 2.0 is easier to use and more powerful than its predecessor Java API for XML-based RPC (Remote Procedure Calls) 1.1
- Layered design hides the complexity
- Extensible at the protocol and transport level
- Fully interoperable with other WS-* stacks
- Part of the Java EE 5 and Java SE 6 platforms

Project GlassFish



Developers Home > Products & Technologies > Java Technology > Java Platform, Enterprise Edition (Java EE) >

> Join SDN | Why Join? | My Profile

Java Platform, Enterprise Edition (Java EE)

GlassFish Community

Downloads
- Early Access

Reference
- API Specifications
- Documentation
- FAQs
- BluePrints
- Technical Articles & Tips

Community
- News
- Using GlassFish
- Improving GlassFish

GlassFish Community
Building an Open Source Java EE 5 Application Server

The GlassFish community is building a free, open source application server which implements the newest features in the Java EE 5 platform (the next version of the J2EE platform).

For More Information

About the GlassFish Community
Learn all about the GlassFish project, why Sun created it and answers to other common questions.
» Read more

Use the Technology
Learn how to use the technology developed by the GlassFish community.
» Read more

Get It!
This is the place to download general distribtution releases related to Project GlassFish.
» Read more

Spotlight

DOWNLOAD THE Java EE SDK

Sun Microsystems, the inventor and leading advocate of Java technology, is releasing a preview of the Java EE 5 SDK . the next generation of enterprise Java which includes the Beta version of the Sun Java System Application Server PE 9.0 (SJSAS PE 9.0). The SDK is a defining release because the newest version of enterprise Java makes building robust, scalable enterprise applications easier than ever and is the perfect platform for implementing SOA and Web 2.0 applications. The Java EE 5 SDK is built

Related Links

Free! Developer Tools
- Netbeans
- Java Studio Creator
- Java Studio Enterprise

Server-Side Products
- SJS Application Server 8.2
- SJS Web Server 6.1
- Java Enterprise System

Developer Toolkits
- Java EE SDK
- JWSDP

Building a Java EE 5 Open Source Application Server

Simplifying Java application Development with **Java EE 5 technologies**

Includes JWSDP, EJB 3.0, JSF 1.2, JAX-WS and JAX-B 2.0

Supports > **20** frameworks and apps

Basis for the **Sun Java System Application Server PE 9**

Free to download and **free** to deploy

Over **1200** members and **200,000** downloads

Integrated with **NetBeans**

java.sun.com/javaee/GlassFish

Source: Sun 2/06—See website for latest stats

For More Information

JavaOne^{SM} conference sessions and BOFs

- TS-3274 Project Glassfish
- TS-1222 RESTful Web Services with JAX-WS 2.0
- TS-1607 Deep Dive into JAXB 2.0
- TS-4661 Project Tango
- TS-1603 Project Tango Interop with WCF
- BOF-2526 JAX-WS 2.0 Performance

Links

- http://java.sun.com/webservices/jaxws/
- http://blogs.sun.com/theaquarium

Q&A

The Java™ API for XML Web Services (JAX-WS) 2.0

Roberto Chinnici/ Rajiv Mordani/ Doug Kohlert

Senior Staff Engineers
Sun Microsystems, Inc.
http://java.sun.com/webservices/jaxws/

Session 1194