

Next Generation Web Services in the Java™ Platform

Roberto Chinnici
Marc Hadley

Sun Microsystems, Inc.
<http://java.sun.com/webservices>

Session 7230

Goal of Your Talk

What your audience will gain

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

Agenda

Overview of JAX-WS 2.0

Annotation-based architecture

Data binding using JAXB 2.0

Asynchronous Web services

Dispatch/provider API

Java SE 6.0 platform-based Web services

Web services security

Future steps and roadmap

Agenda

Overview of JAX-WS 2.0

Annotation-based architecture

Data binding using JAXB 2.0

Asynchronous Web services

Dispatch/provider API

Java SE 6.0 platform-based Web services

Web services security

Future steps and roadmap

The Present

The current state of Web services in the Java platform

- Java API for XML-based RPC (JAX-RPC) 1.1
 - Support for SOAP/HTTP(S)
 - WS-I Basic Profile 1.0 compliance
- SOAP with Attachments API for Java (SAAJ) 1.2
 - Low-level access to SOAP messages
- Java 2 Platform, Enterprise Edition (J2EE™) 1.4
 - Servlet and stateful session bean endpoints
 - Full packaging/deployment model
- Java Architecture for XML Binding (JAXB) 1.0
 - Extensive (but not complete) data binding support

The Challenges

Or where we fell short of the mark

- Technologies not fully integrated
 - They work together well, but you get to see the seams
 - e.g., Using JAXB from JAX-RPC
- Complexity
 - Different tools, multiple configuration files
- World is not standing still
 - SOAP 1.2, WSDL 2.0
 - WS-I Attachment Profile and Basic Security Profile
 - Overwhelming number of WS-* specifications
 - Lack of support for REST-ful Web services

The Solution

One integrated Web services stack

- Next generation Web services developed as a coordinated suite of new JSRs
- JAX-WS 2.0 (JSR 224)
- JAXB 2.0 (JSR 222)
- JSR 181 1.1
- Java APIs for XML-based Web Services Addressing (JAX-WSA) (JSR 261)
- Designed to work together
- Exploit new language features in Java 2 Platform, Standard Edition (J2SE™) 5.0
- Significant ease-of-development benefits
- All part of Java EE 5.0 and Java SE 6.0 platforms

JAX-WS 2.0

JSR 224 in the Java Community ProcessSM Program

- Successor to JAX-RPC 1.1
- Retains the natural RPC programming model
- Innovates on many fronts, including
 - Data binding
 - Protocol and transport independence
 - Ease-of-use
- Available online
 - Public draft of the specification
 - Early Access 2 reference implementation

The Inevitable Caveat

Can't say we didn't warn you...

- The API is not final yet, so everything you'll see today may change
- On the plus side, if you send us your feedback, it may change the way **YOU** want!

jsr224-spec-comments@sun.com

Agenda

Overview of JAX-WS 2.0

Annotation-based architecture

Data binding using JAXB 2.0

Asynchronous Web services

Dispatch/provider API

Java SE 6.0 platform-based Web services

Web services security

Future steps and roadmap

Why Annotations?

@WebService @WebMethod @SOAPBinding

- Put the information next to the program element it affects
 - Class to portType mapping
 - Method to operation mapping
 - Data type to XML mapping
- Avoids the use of lengthy descriptors
- Makes customization easier for humans and tools
- Easily extensible in the future

Example

```
@WebService (name="CreditRatingService",  
             targetNamespace="http://example.org")  
public interface CreditRating {  
  
    @WebMethod (operationName="getCreditScore")  
    public Score getCredit(  
        @WebParam (name="customer")  
        Customer c);  
  
}
```

More Benefits of Annotations

It gets better!

- No more annoying RMI-like markers
 - `extends java.rmi.Remote`
 - `throws java.rmi.RemoteException`
- At compile-time, apt tool can check the validity of annotations and report line number information for errors
- At runtime, marshalling and dispatching are entirely annotation-driven
 - The annotations are the **TRUTH** about the Java technology/WSDL mapping

Standard Customizations

`<jaxws:class/>` `<jaxws:method/>`

- XML-based
- Very similar to JAXB 2.0 customizations
- Two ways to use them
 - Embedded in a WSDL document
 - As a standalone external binding file
- Defined and mandated by the JAX-WS 2.0 spec
- Supported by all JAX-WS 2.0 tools

Example

```
<jaxws:bindings
    wsdlLocation="http://example.org/foo.wsdl">
  <jaxws:package name="com.acme.foo"/>
  <jaxws:bindings node="wsdl:types/xs:schema
    [targetNamespace='http://example.org/bar']">
    <jaxb:bindings>
      ...some JAXB binding declarations...
    </jaxb:bindings>
  </jaxws:bindings>
</jaxws:bindings
  node="wsdl:portType[@name='StockQuoteUpdater']">
  <jaxws:bindings node="wsdl:operation
    [@name='setLastTradePrice']">
    <jaxws:method name="updatePrice"/>
  </jaxws:bindings>
</jaxws:bindings>
  ...additional binding declarations....
</jaxws:bindings>
```

Agenda

Overview of JAX-WS 2.0

Annotation-based architecture

Data binding using JAXB 2.0

Asynchronous Web services

Dispatch/provider API

Java SE 6.0 platform-based Web services

Web services security

Future steps and roadmap

Data Binding With JAXB 2.0

Outsourcing in action ;-)

- JAX-WS 2.0 delegates all data binding functionality to JAXB 2.0
- Learn just one mapping and one set of annotations
- XML schema 100% supported
- Homogeneous customization
 - Customize WSDL and the schemas in it in one go
- Richer type mapping via Java API for XML Processing (JAXP)
 - e.g., `javax.xml.datatype.XMLGregorianCalendar`

Example

`@XmlType`

```
public class Trade {  
    @XmlElement(name="tickerSymbol")  
    public String symbol;  
    @XmlAttribute  
    int getQuantity() {...}  
    void setQuantity() {...}  
}
```

```
<xs:complexType name="trade">  
    <xs:sequence>  
        <xs:element name="tickerSymbol" type="xs:string"/>  
    </xs:sequence>  
    <xs:attribute name="quantity" type="xs:int"/>  
</xs:complexType>
```

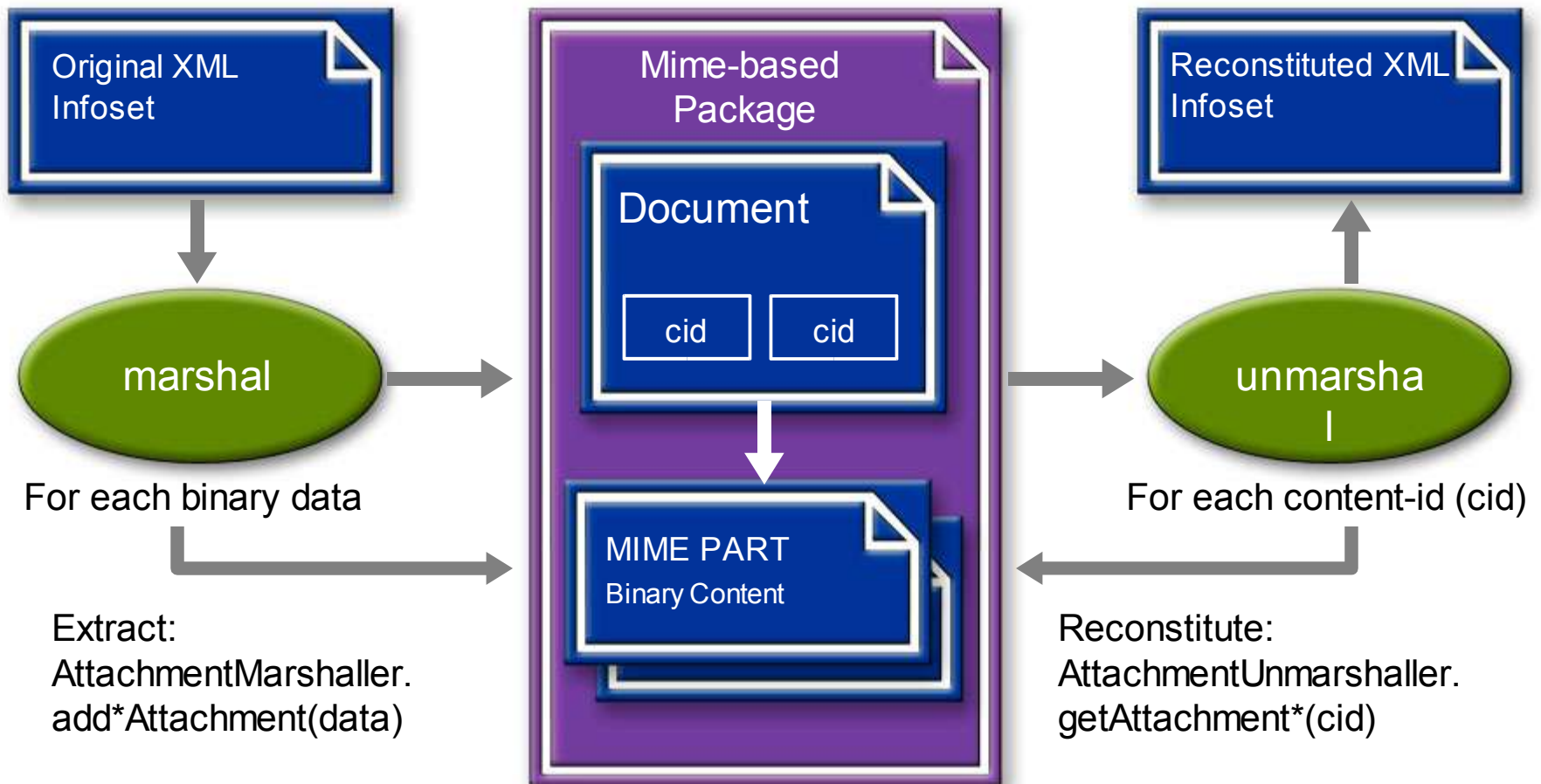
Interoperable Attachments

Using W3C MTOM / XOP

- Data of type `xs:base64Binary` can be marshalled as an attachment
- On the wire, it uses the MIME format
- Completely transparent to the user
- JAXB 2.0 is MTOM-aware
- MIME type mapping
 - `image/*` `java.awt.Image`
 - `text/xml` `javax.xml.transform.Source`
 - `*/*` `javax.activation.DataHandler`

JAX-WS 2.0/JAXB Integration

A closer look at supporting MTOM/XOP



Agenda

Overview of JAX-WS 2.0

Annotation-based architecture

Data binding using JAXB 2.0

Asynchronous Web services

Dispatch/provider API

Java SE 6.0 platform-based Web services

Web services security

Future steps and roadmap

Asynchronous Client API

When you don't want your application to freeze...

- Async methods generated on-demand
- Two models: Polling or callbacks
 - Polling: You ask for the response whenever you're ready
 - Callback: Notification as soon as the response arrives
- No need to spawn threads in the application
- No changes needed on the server

Key Types When Using Async

Using Java language generics

- **Future<T>**
 - A promise to deliver a value of type T when it becomes available
- **Response<T>**
 - Extends Future<T>
 - Contains a value of type T plus the JAX-WS 2.0 context
- **AsyncHandler<T>**
 - User-written handler for callback notifications
- **Future<?>**
 - Useful only for cancelling the invocation

Example: Polling

```
@WebService
public interface CreditRatingService {

    // sync operation
    Score getCreditScore (Customer customer) ;

    // async operation w/polling
    Response<Score>
        getCreditScoreAsync (Customer customer) ;

    // async operation w/callback
    Future<?>
        getQuoteAsync (Customer customer,
            AsyncHandler<Score> handler) ;
}
```

Example: Polling Client

```
CreditRatingService svc = ...;

Response<Score> response =
    svc.getCreditScoreAsync(customerFred);

// client app does other things...

// ready to deal with the response
// no cast needed, thanks to generics
Score score = response.get();

// or use
// Score score = response.get(10L,
// TimeUnit.SECONDS);
// to wait 10 seconds
```

Example: Callback

```
@WebService
public interface CreditRatingService {

    // sync operation
    Score getCreditScore (Customer customer) ;

    // async operation w/ polling
    Response<Score>
        getCreditScoreAsync (Customer customer) ;

    // async operation w/callback
    Future<?>
        getQuoteAsync (Customer customer,
                      AsyncHandler<Score> handler) ;
}
```

Example: Callback Client

```
CreditRatingService svc = ...;

Future<?> invocation =
    svc.getCreditScoreAsync(customerFred,
        new AsyncHandler<Score>() {
            public void handleResponse
                (Response<Score> response) {
                Score score = response.get();
                // do work here...
            }
        });

// to cancel the request, use
// invocation.cancel(true);
```

Agenda

Overview of JAX-WS 2.0

Annotation-based architecture

Data binding using JAXB 2.0

Asynchronous Web services

Dispatch/provider API

Java SE 6.0 platform-based Web services

Web services security

Future steps and roadmap

Why Dispatch/Provider?

Low-level messaging API

- Sometimes you need
 - More control on the messages being sent
 - Write one class that can support many endpoints
 - Perhaps for a family of similar WSDLs
 - Or a gateway passing messages on to somebody else
- Sure you could use sockets directly...
- But you also want
 - Reuse existing same message handlers
 - Support all the different protocol bindings in JAX-WS 2.0
 - Not reinvent the wheel

Dispatch Interface

Generics are your friend

```
// 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);
}
```

Supported Message Types

The choice is yours

- Can access (logical) **PAYLOAD** or **MESSAGE**
- Multiple views on a payload/message

XML	<code>javax.xml.transform.Source</code>
JAXB object	<code>java.lang.Object</code>
SOAP Message	<code>javax.xml.soap.SOAPMessage</code>
... (it's extensible)	(some other message class)

Example: Using Dispatch

```
Service svc = ...;
```

```
Dispatch<Source> d1 = svc.createDispatch(port,  
    Source.class, Service.Mode.PAYLOAD);  
Source result = d1.invoke(mySource);
```

```
Dispatch<SOAPMessage> d2 = svc.createDispatch(port,  
    SOAPMessage.class, Service.Mode.MESSAGE);  
SOAPMessage result = d2.invoke(mySOAPMessage);
```

```
Dispatch<Object> d3 = svc.createDispatch(port,  
    myJAXBContext, Service.Mode.PAYLOAD);  
Object result = d3.invoke(myJAXBObject);
```

Provider Interface

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

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

}
```

- The context has information about
 - The WSDL document
 - The port the request was sent to
 - The portType the request belongs to
 - Any properties created by handlers (e.g., session info)

Example: Two Providers

```
@ServiceMode(Service.Mode.PAYLOAD)
public Provider1 implements Provider<Source> {
    public Source invoke(Source msg,
                        Map<String, Object> context) {
        // ... implementation code ...
    }
}

@ServiceMode(Service.Mode.MESSAGE)
public Provider2 implements Provider<SOAPMessage> {
    public SOAPMessage invoke(SOAPMessage msg,
                             Map<String, Object> context) {
        // ... implementation code ...
    }
}
```

Protocol and Transport Independence

The SOAP-iness is gone

- Normally, applications work against protocol and transport independent messages
 - Payload is a Source or an object managed by JAXB
- If and only when you need it, you can drop down one level and access SOAP messages
- Binding in use is identified by URI
 - SOAP11_HTTP_BINDING
 - SOAP12_HTTP_BINDING
- Expect to see a lot more protocols/transports being supported, e.g., SOAP/Java Message Service (JMS)

SOAP 1.2 and REST Support

More power to you

- SOAP 1.1/HTTP supported with service endpoint interfaces thanks to WSDL 1.1
- New bindings supported by Dispatch/Provider
 - SOAP 1.2/HTTP
 - XML/HTTP (REST)
- Will add support for WSDL when the corresponding bindings are standardized
- WSDL 2.0 will capture many more features of the XML/HTTP binding

Agenda

Overview of JAX-WS 2.0

Annotation-based architecture

Data binding using JAXB 2.0

Asynchronous Web services

Dispatch/provider API

Java SE 6.0 platform-based Web services

Web services security

Future steps and roadmap

Java SE 6.0 Platform (Project Mustang) Inclusion

JAX-WS 2.0 will be a Mustang component

- JAX-WS 2.0 will be in Mustang along with
 - JAXB 2.0
 - SAAJ 1.3
 - New HTTP server API for Java SE platform
- Client **and** server support
- Smaller footprint for your applications
 - JAX-WS 2.0 runtime is included in the Java runtime environment (JRE)
 - Thanks to the annotation-driven runtime, no need to generate stubs or ties

Why Add Server Support in Java SE Platform?

We listened to you, the developers

- Use cases
 - Receiving notifications from a server
 - Callbacks into a client application
 - Web service-enabling an application
 - Management
 - Testing
- Previously, you had to embed a servlet container or use your own lightweight HTTP server
- Not intended for high-performance, highly scalable applications (use Java EE platform instead)

Example: Simple Publishing

One method call and you're done!

```
@WebService
public class MyEndpoint {
    public String sayHello(String s) { ... }
}
```

```
MyEndpoint impl = new MyEndpoint()
EndpointFactory f = EndpointFactory.newInstance();
f.publish("http://localhost:8080/hello", impl);
```

- No deployment step
- WSDL/schema generated at runtime
- Default binding is SOAP 1.1/HTTP

Example: Longer Version

Offers finer control on the server configuration

```
HttpServer server = HttpServer.create(
    new InetSocketAddress(8080), 10);
server.setExecutor(Executor.newFixedThreadPool(10));
HttpContext context = server.createContext(
    "http",
    "/hello");

MyEndpoint impl = new MyEndpoint()
EndpointFactory f = EndpointFactory.newInstance();
Endpoint endpoint = f.createEndpoint(
    SOAPBinding.SOAP11HTTP_BINDING,
    impl);

endpoint.publish(context);
```

Agenda

Overview of JAX-WS 2.0

Annotation-based architecture

Data binding using JAXB 2.0

Asynchronous Web services

Dispatch/provider API

Java SE 6.0 platform-based Web services

Web services security

Future steps and roadmap

Web Services Security Specifications

Read 'til you drop

- Web Services Security (WSS)
- WS-I Basic Security Profile
- SAML, X.509, SSL/TLS
- All based on SOAP 1.1, XML Dsig, XML Enc
- More of a framework than a profile
 - Support multiple security token types
- Deals with things like
 TLS_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
 (that's a cyphersuite...)

Security Annotations

Tells us what you want and we'll do it for you

- Declarative security requirements for inbound and outbound messages
 - AUTHENTICATION
 - CONFIDENTIALITY
 - INTEGRITY
- All necessary modules engaged automatically at runtime, e.g.,
 - confidentiality: Encrypt the message
 - Integrity: Sign the message
 - Authentication: Include a security token

Example

```
@WebService
public interface CreditRating {
    @MessageSecurity(
        inbound={AUTHENTICATION, CONFIDENTIALITY},
        outbound={INTEGRITY, CONFIDENTIALITY})
    public Score getCredit(Customer c);
}
```

- Using SOAP/HTTP, WS-Security headers will be generated automatically
- Other protocols/transport may support them too
 - e.g., HTTPS gives you confidentiality and authentication of the receiver (server)

Agenda

Overview of JAX-WS 2.0

Annotation-based architecture

Data binding using JAXB 2.0

Asynchronous Web services

Dispatch/provider API

Java SE 6.0 platform-based Web services

Web services security

Future steps and roadmap

JAX-WS 2.0 Roadmap

A fully integrated Web services stack

- JAX-WS 2.0 to go final in early 2006
- Will ship with Java EE 5.0 and Java SE 6.0 platforms
- Add support for WS-Addressing and WS-Policy
 - Done via JSR 261 and JSR 265
- For a future JAX-WS 2.x release
 - Standardize SOAP/JMS binding
 - Connection-less asynchronous invocations
 - Track other core WS-* specs, e.g., WS-RM, WS-MEX
 - Expand declarative programming via annotations

Summary

- Lots of new features in JAX-WS 2.0
- 100% XML Schema via JAXB 2.0
- Ease of use via annotations
- Protocol and transport independence
- Fast Web services
- Java SE 6.0 platform-based Web services

For More Information

Sessions

- TS-3477 Implementing the new Web services APIs
- TS-3636 JAXB 2.0
- TS-7187 Fast Infoset

Resources

- Spec <http://www.jcp.org/en/jsr/detail?id=224>
- RI <https://jax-rpc.dev.java.net>
- Java SE 6.0 platform <https://mustang.dev.java.net/>
- Glassfish <https://glassfish.dev.java.net>

Q&A

Submit Session Evaluations for Prizes!

Your opinions are important to Sun

- You can win a \$75.00 gift certificate to the on-site Retail Store by telling Sun what you think!
- Turn in completed forms to enter the daily drawing
- Each evaluation must be turned in the same day as the session presentation
- Five winners will be chosen each day (Sun will send the winners e-mail)
- Drop-off locations: give to the room monitors or use any of the three drop-off stations in the North and South Halls

Note: Winners on Thursday, 6/30, will receive and can redeem certificates via e-mail

Next Generation Web Services in the Java™ Platform

Roberto Chinnici
Marc Hadley

Sun Microsystems, Inc.
<http://java.sun.com/webservices>

Session 7230