# Writing a JAX-WS Endpoint

- The Endpoint Interface
- The Endpoint Class
- Annotations
- Exceptions



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

## What is JAX-WS?

- The Java API for XML-Based Web Services **(JAX-WS)** is the standard Java API for programming Web services - both services and clients
- This chapter focuses on the services side of JAX-WS
- A major goal of JAX-WS is source-code portability across implementations and runtime interoperability with services implemented with other technologies



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# JAX-WS Programming Features

- JAX-WS defines:
  - Programming models for Java Web services and clients
  - APIs for services and clients
  - **Annotations** to provide metadata
  - How to define **handlers** on the client and server
- JAX-WS delegates to JAX-B to map to/from Java/XML



---

JAX-WS differs from its predecessor, JAX-RPC in that it delegates to JAX-B for XML binding. Due to historical considerations, JAX-RPC contained its own binding framework that was less capable than JAX-B.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

## Annotations

- JAX-WS makes heavy use of **annotations** to provide metadata
- JAX-WS defines its own annotations and "borrows" annotations from *JSR 181: Web Services Metadata for the Java Platform*
- This chapter covers both JAX-WS-specific annotations and JSR-181 annotations
    - JAX-WS annotations are in the *javax.xml.ws* package
    - JSR-181 annotations are in the *javax.jws* package



The rationale for having two separate specifications for Web services annotations is a bit murky, but JSR-181 came first, and could conceivably be used with JAX-RPC implementations.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

## JAX-B Supported Types

- Since JAX-WS delegates XML to Java mapping (and vice versa) to JAX-B, Web service method parameters and return values must be compatible with JAX-B
- Supported types:
    - Java primitives (except for **char**)
    - Selected standard classes from the Java class library
    - User-written classes that are compatible with JAX-B (e.g. JavaBeans with get/set methods)
    - Arrays of supported types
    - Exceptions



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

## What is an Endpoint?

- The **endpoint**, sometimes referred to as the **service implementation bean**, is a Java class that that provides one or more Web service methods
- Endpoint classes must be annotated with at least the **@WebService** annotation
- JAX-WS supports both POJO and EJB-based endpoints (EJBs covered later)

```
1    package service;
2
3    @WebService
4    public class MyEndpoint
5    {
6        . . .
7    }
```



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

## Rules For a POJO Endpoint Class

- The implementation class must:
  - Be a **public**, **non-inner**, **non-final** and **non-abstract** class
  - Include the @WebService annotation
  - Define a public, zero-argument constructor
  - NOT define a **finalize()** method



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# The Optional Service Endpoint Interface

- If you wish, you can explicitly define a **service endpoint interface** (SEI) that specifies the methods available on a Web service
- The SEI maps to the **port type** in the service's WSDL
- Rules for the SEI:
  - It must carry a @WebService annotation
  - Method parameters and returns must be compatible with JAX-B

```
1    package service;
2
3    @WebService
4    public interface MySEI
5    {
6       public int method1(String s);
7    }
```



---

In JAX-RPC, the SEI was required, but JAX-WS makes it optional.

Some developers prefer to design interfaces first and then classes that implement the interfaces, so the SEI supports that approach.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# Referencing the SEI

- If you wish to have an explicit SEI, the endpoint class must reference the SEI via the @WebService annotation
- If the endpoint class does NOT reference an SEI and no methods are annotated with @WebMethod, then all non-**java.lang.Object** methods are exposed as service methods

```
1    package service;
2
3    @WebService(endpointInterface="service.MySEI")
4    public class MyEndpoint
5    {
6       public int method1(String s) {...}
7       public double method2(int x) {...}
8    }
```



---

In this case, method2 would NOT be exposed as a service method, since it's not in the SEI.

However, if this class had not referenced an SEI at all, both methods would've been exposed, absent any @WebMethod annotations.

Note that it's not a requirement for the endpoint class to use the Java "implements" keyword, but it's OK to do so.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# Endpoint Class Lifecycle

- If the implementation wishes to be notified of lifecycle events (i.e. object creation and destruction), it can provide methods annotated with:
  - @PostConstruct
  - @PreDestroy

```
1   package service;
2
3   @WebService
4   public class MyEndpoint
5   {
6       . . .
7       @PostConstruct
8       public void init() {...}
9       @PreDestroy
10      public void destroy() {...}
11  }
```



---

These two annotations are not part of JAX-WS or JSR-181, but are standard JEE annotations from the javax.annotation package.

These methods must return "void" and accept no input arguments.

In a given class, at most one method can be annotated with @PreDestroy and at most one method with @PostConstruct.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# The @WebService Annotation

- This annotation identifies either an SEI or an endpoint class as a Web service
- Optional Elements (parameters):

| Element | Description |
|---|---|
| endpointInterface | Fully qualified SEI name |
| name | Maps to WSDL portType |
| portName | Maps to WSDL port |
| serviceName | Maps to WSDL service |
| targetNamespace | Defines WSDL targetNamespace |
| wsdlLocation | References a top-down WSDL |



---

This annotation is from the javax.ws package (defined by JSR-181).

In annotation lingo, an "element" is a value passed to an annotation. For example:

```
@WebService(name="MyService")
```

In the above, "name" is an element with value "MyService".

The defaults for the elements are as follows:

| Element | Default Value |
|---|---|
| endpointInterface | None |
| name | Class or interface name |
| portName | serviceName + "Port" |
| serviceName | Class or interface name + "Service" |
| targetNamespace | None |
| wsdlLocation | None |

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# @WebService Annotation Examples

```
@WebService(name="MyGreatService")
public interface MySEI {...}

@WebService(endpointInterface="service.MySEI")
public class MyEndpoint {...}

@WebService
  (endpointInterface="mypackage.OrderStatus"
 , targetNamespace="http://big.com/orderstatus"
 , serviceName="OrderStatus"
 , portName="OrderStatusPort"
 , wsdlLocation="WEB-INF/wsdl/orderstatus.wsdl")
public class OrderStatus {...}
```



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# The @WebMethod Annotation

- This annotation customizes a Web service method
- Optional Elements (parameters):

| Element | Description |
|---|---|
| action | For SOAP, the SOAPAction value |
| exclude | Do not expose this method |
| operationName | The WSDL operation name |



The "exclude" element lets you explicity configure a method, perhaps one inherited from a superclass, as being a service method (you could override the inherited method and annotate it).

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# @WebMethod Annotation Examples

```
1    @WebService()
2    public class MyEndpoint
3    {
4       @WebMethod(action="myaction",
5          operationName="METHOD1")
6       public int method1(int x) {...}
7
8       @WebMethod(exclude=true)
9       public int method2(int x) {...}
10   }
```



In this case, only "method1" would be exposed as a service method, and the WSDL would define the operation name as METHOD1 (upper case) rather than the default of "method1" (taken from the method name).

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# Which Methods Are Exposed?

- According to JSR-181, Version 2:

> If the implementation bean does not implement a service endpoint interface and there are no @WebMethod annotations in the implementation bean (excluding @WebMethod annotations used to exclude inherited @WebMethods), all public methods other than those inherited from java.lang.Object will be exposed as Web Service operations, subject to the inheritance rules specified in Common Annotations for the Java Platform, section 2.1.



---

This is a bit vague: what happens if a you DO write @WebMethod on a method? Are other public methods exposed?

Because of the lack of clarity here (and the fact that this wording has changed somewhat since the first version of JSR-181), some developers consider it a best practice to explicity annotate ALL public methods in an endpoint class.

Note that if you DO reference an SEI, you should write any @WebMethod annotations in the interface, not in the endpoint class. Some implementations will ignore such annotations in the class.

The Common Annotations specification is JSR-250.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# The @WebParam Annotation

- This annotation customizes a parameter to a Web service method
- You can use this in an interface or endpoint class
- Optional Elements (parameters):

| Element | Description |
|---|---|
| header | If true, the parameter is from a header rather than the Body |
| mode | Mode.IN, Mode.OUT or Mode.INOUT |
| name | The parameter's WSDL "part" name |
| partName | The parameter's WSDL "part" name |
| targetNamespace | For doc style services, the parameter's namespace URI |



---

The "name" and "partName" elements are similar and somewhat redundant. According to the JSR-181 spec, for the "name" element: If the operation is rpc style and @WebParam.partName has not been specified, this is name of the wsdl:part representing the parameter. If the operation is document style or the parameter maps to a header, this is the local name of the XML element representing the parameter. A name MUST be specified if the operation is document style, the parameter style is BARE, and the mode is OUT or INOUT. And for "partName": The name of the wsdl:part representing this parameter. This is only used if the operation is rpc style or if the operation is document style and the parameter style is BARE. See the JSR-181 spec for the defaults, but note in the absence of "name" or "partName", JAX-WS assigns part names like "arg0", "arg1" and so forth.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# @WebParam Annotation Examples

```
1    @WebService()
2    @SOAPBinding(style=Style.RPC)
3    public class MyEndpoint
4    {
5      public int method6(
6        @WebParam(mode=Mode.IN) int x,
7        @WebParam(name="QQQ", partName="YYY") int y,
8        @WebParam(name="ZZZ") int z,
9        @WebParam(header=false) int q)
10      {...}
11   }
```



In this case, the service is an RPC/literal service (we will cover @SOAPBinding later). Here's a fragment of the generated WSDL:

```
<message name="method6">
  <part name="arg0" type="xsd:int" />
  <part name="YYY" type="xsd:int" />
  <part name="ZZZ" type="xsd:int" />
  <part name="arg3" type="xsd:int" />
</message>
```

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

## The @WebResult Annotation

- This annotation customizes the mapping of a Web service method's return value
- @WebResult is similar to @WebParam, but it's for the return, rather than parameters
- Optional Elements (parameters):

| Element | Description |
|---|---|
| header | If true, the return value is a header rather than in the Body |
| name | The return's WSDL "part" name |
| partName | The return's WSDL "part" name |
| targetNamespace | For doc style services, the return's namespace URI |



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# @WebResult Annotation Examples

```
1    @WebService()
2    @SOAPBinding(style=Style.RPC)
3    public class MyEndpoint
4    {
5      @WebResult(header=false, name="myval")
6      public int method7(int q) {...}
7    }
```



In this case, the service is an RPC/literal service (we will cover @SOAPBinding later). Here's a fragment of the generated WSDL:

```
<message name="method7Response">
  <part name="myval" type="xsd:int" />
</message>
```

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

## The @Oneway Annotation

- This annotation indicates that a service method has only an input message (no output message)
- Oneway methods must return **void** and can have no OUT or INOUT parameters

```
1    @WebService()
2    public class MyEndpoint
3    {
4      @Oneway
5      public void method8(int q) {...}
6    }
```



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# The @SOAPBinding Annotation

- This annotation specifies the mapping of a Web service onto SOAP
- You can apply this annotation on interfaces, classes or methods
- Optional Elements (parameters):

```
Element           Values
-------           -----------
parameterStyle    ParameterStyle.BARE
                  ParameterStyle.WRAPPED *
style             Style.DOCUMENT *
                  Style.RPC
use               Use.ENCODED
                  Use.LITERAL *

* - default if not specified
```



---

This annotation, and the enumerations that provide values (e.g. ParameterStyle) are from the javax.jws.soap package (defined by JSR-181).

Note that the ENCODED "use" is not WS-I compliant and may not be supported by all implementations.

Also note that it's forbidden to specify this annotation with Style=RPC on a method (it's legal on a class or an interface).

Finally, note that if you specify ParameterStyle.BARE, then any service method must accept at most one parameter.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# @SOAPBinding Annotation Examples

```
@WebService
@SOAPBinding(parameterStyle=ParameterStyle.BARE)
public interface MySEI {...}

@WebService
public class OrderStatus
{
  @SOAPBinding(style=Style.RPC, use=Use.LITERAL)
  public int method1(String s) {...}
}
```



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# The @BindingType Annotation

- This annotation specifies the binding for an endpoint
- You can apply this annotation on interfaces or classes
- Optional Elements (parameters):

| Element | Values |
| --- | --- |
| value | SOAPBinding.SOAP11HTTP_BINDING * |
| | SOAPBinding.SOAP12HTTP_BINDING |
| | SOAPBinding.SOAP11HTTP_MTOM_BINDING |
| | SOAPBinding.SOAP12HTTP_MTOM_BINDING |
| | HTTPBinding.HTTP_BINDING |

* - default if not specified



---

The HTTP_BINDING is typically used with RESTful services that don't use SOAP.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# Input, Output and Input/Output Parameters

- Even though Java methods can explicitly return only one thing, WSDL allows for **in**, **out** and **inout** parameters

SomeService.wsdl

```
. . .
<message name="myRequest">
  <part name="p1" type="xsd:double"/>
  <part name="p2" type="xsd:double"/>
</message>
<message name="myResponse">
  <part name="p2" type="xsd:double"/>
  <part name="p3" type="xsd:double"/>
</message>
<portType type="myPort">
 <operation name="myMethod" parameterOrder="p1 p2 p3">
   <input message="tns:myRequest"/>
   <output message="tns:myResponse"/>
 </operation>
</portType>
. . .
```



Some programming languages (e.g. C and C++) allow for passing of parameters by address so that the caller can pass a parameter and have a function modify the actual parameter, rather than a copy of the parameter. WSDL and SOAP supports this style of parameter passing by allowing for "out" and "inout" parameters.

In this WSDL, p1 is an input parameter, p2 is "in-out" and p3 is "out".

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# The Holder Class

- To support **out** and **inout** parameters, JAX-WS provides the parameterized (generic) **Holder** class that "wrap" a parameter value

```
1    Holder<Double> myHolder = new Holder<Double>(22.33);
2
3    . . .
4
5    System.out.println(myHolder.value);
6
7    . . .
8
9    myHolder.value = 66.66;
```



---

This is different than in JAX-RPC, which provided a whole set of Holder classes. This new approach is possible because the generics support in JSE 5 and later.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# OUT and INOUT Example: The Service

```
1    @WebService(name="myPort")
2    @SOAPBinding(style=Style.RPC)
3    public class Endpoint7
4    {
5      public void myMethod(
6        @WebParam(name="p1", mode=Mode.IN) double p1,
7        @WebParam(name="p2", mode=Mode.INOUT) Holder<Double> p2,
8        @WebParam(name="p3", mode=Mode.OUT) Holder<Double> p3)
9      {
10       System.out.println("Input: " + p1);
11       System.out.println("Input/Output: " + p2);
12
13       p2 = new Holder<Double>(44.44);
14       p3.value = 55.55;
15     }
16   }
```



---

The @WebParam annotations configure the method so it matches the WSDL shown two pages ago.

Note how the service can only "read" p1, but can read and write p2. Furthermore, p3 acts as an output only.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# OUT and INOUT Example: The Client

```
1    MyPort stub = . . .;
2
3    Holder<Double> p2 = new Holder<Double>(11.11);
4    Holder<Double> p3 = new Holder<Double>();
5
6    stub.myMethod(88.88, p2, p3);
7    System.out.println("Input/Output: " + p2.value);
8    System.out.println("Output: " + p3.value);
```



Note the client must create an "empty" Holder object for p3, but doesn't need to initialize its value.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# Exceptions and SOAP Faults

- The JAX-WS runtime maps Java exceptions to SOAP Faults

MyException.java

```
package exceptions;

public class MyException extends Exception
{
  public MyException()
  {
    super ("This is my exception!");
  }
}
```

MyService.java

```
public class MyService
{
  public int myMethod(int x) throws MyException
  {
    if (x < 0) throw new MyException();
   else return x;
  }
}
```

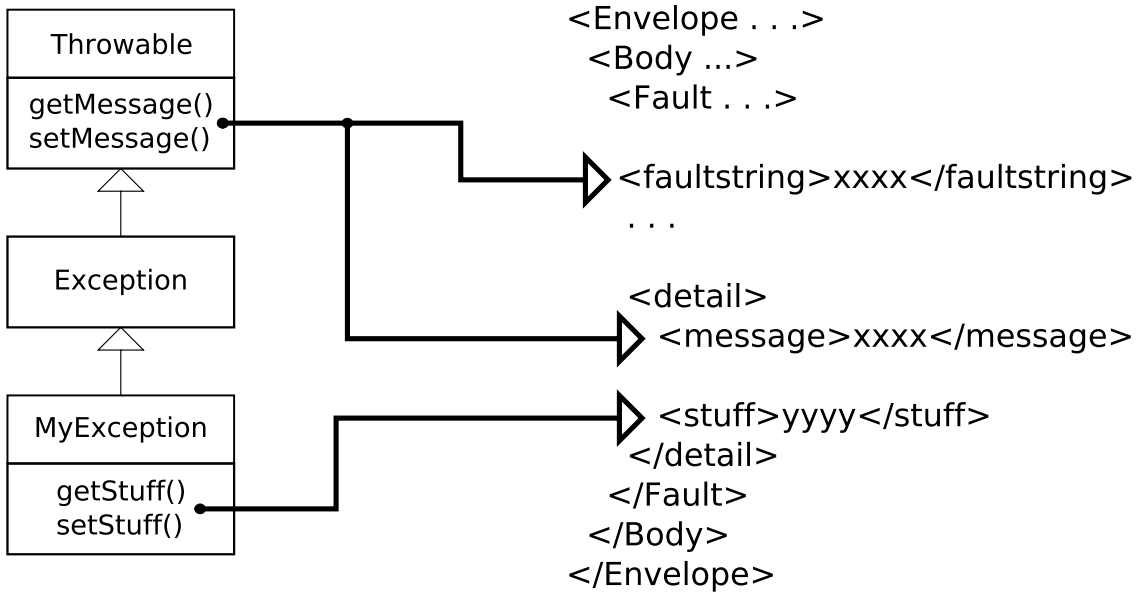


Java clients can handle Faults using try-catch blocks.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# Fault Details

- In a bottom-up approach, JAX-WS will map **properties** in an exception class to **details** in the Fault message






Properties are exposed in the normal JavaBean fashion via get/set methods.

Note that all exceptions ultimately derive from java.lang.Throwable, and that type exposes a "message" property.

If an exception class doesn't provide a "message" (i.e., the string is null), then an implementation can substitute the exception class's fully-qualified class name as the faultstring.

In a top-down approach, or when generating client stubs, JAX-WS will generate TWO classes for each Fault: a so-called "fault bean" that maps the Fault's detail element, and an exception class that references the "fault bean".

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# The @WebFault Annotation

- This annotation lets you customize the Fault message

| Element | Description |
|---|---|
| faultBean | The name of the "fault bean" that describes the Fault's detail element |
| name | The Fault's WSDL "part" name |
| targetNamespace | The "part" element's namespace URI |

```
1    @WebFault(name="MyFault")
2    public class MyException
3      extends Exception
4    {
5     . . .
6    }
```



---

Unfortunately, in JAX-WS version 2.1, there's no way to specify the element name of the Fault, only its child "part" element.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# The WebServiceContext

- JAX-WS uses **dependency injection** to optionally provide endpoints with a **WebServiceContext** reference

```
1    @WebService
2    public class MyEndpoint
3    {
4       @Resource
5       private WebServiceContext ctx;
6
7       . . .
8    }
```

| javax.xml.ws.WebServiceContext |
| --- |
| |
| getEndpointReference(clazz:Class, refParms:Element[]):EndpointReference<br>getEndpointReference(refParms:Element[]):EndpointReference<br>getMessageContext():MessageContext<br>getUserPrincipal():Principal<br>isUserInRole(role:String):boolean |

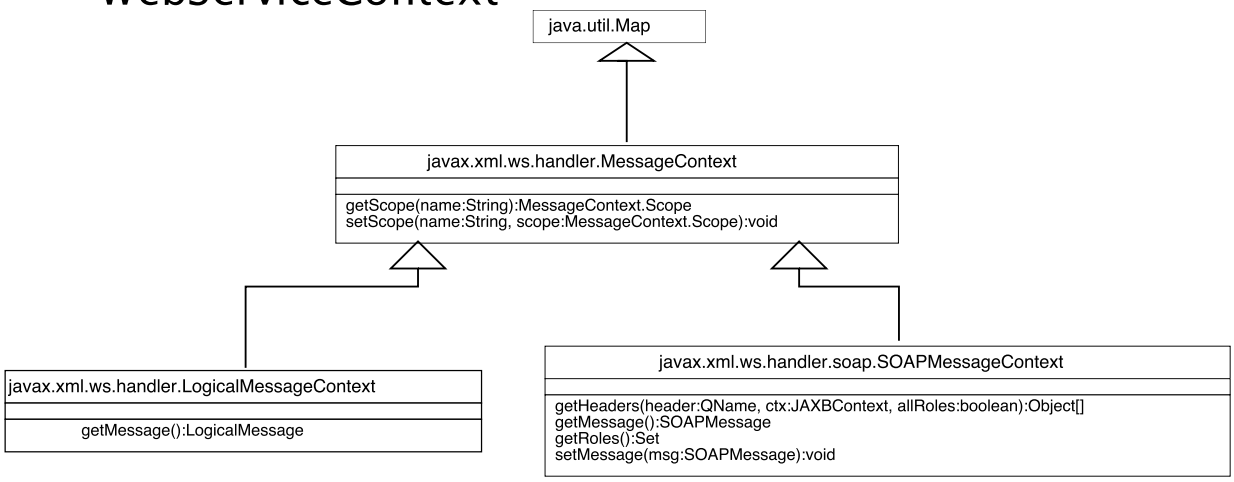


The WebServiceContext type gives endpoints access to security info and to the raw MessageContext.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# The MessageContext

- An endpoint can retrieve additional message-context references from the WebServiceContext






As shown on the last page, an endpoint class can obtain a WebServiceContext, and from it, retrieve a MessageContext as described here.

Later in the course, you will see how to use some of these methods to share data amongst JAX-WS handlers and the service endpoint.

For the SOAP-based services we are discussing, the actual type of the MessageContext is SOAPMessageContext. Note how it provides methods to access the SOAPMessage, which is an SAAJ type that represents message as a whole, including the Envelope, Header and Body.

The LogicalMessageContext gives you access to the input message without tying you to SOAP, so it could be used for RESTful services, for example.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# Accessing the Servlet Environment

- At the risk of tying your service to the servlet environment, the **MessageContext** type gives you access to several servlet and HTTP values

```
1    @WebService
2    public class MyEndpoint
3    {
4      @Resource
5      WebServiceContext ctx;
6
7       public int myMethod(int x)
8       {
9         MessageContext mc = ctx.getMessageContext();
10        HttpServletRequest req = (HttpServletRequest)
11          mc.get(MessageContext.SERVLET_REQUEST);
12        HttpSession session = req.getSession();
13        . . .
14      }
15   }
```



---

Here we show using the MessageContext to access the servlet-request and HTTP session, which would allow the service to maintain state in the session.

See the documentation for MessageContext for a complete listing of the values it provides.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

# Chapter Summary

In this chapter, you learned:

- About the JAX-WS service programming model
- About Java to XML mapping



Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum