Document-Oriented SOAP

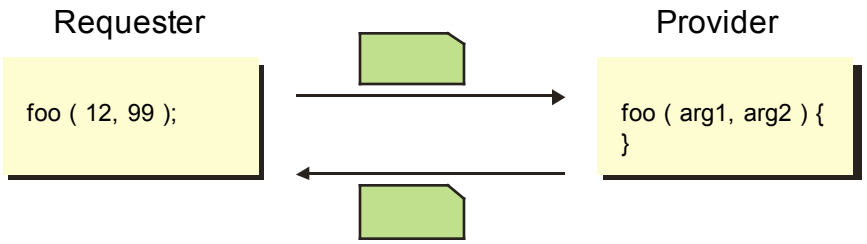
- Document Oriented versus RPC
- Writing a Provider

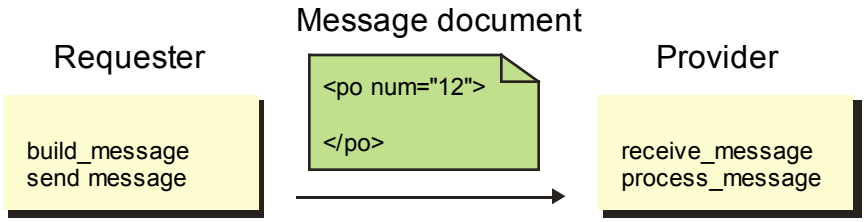
Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Document-Oriented versus RPC

- In an RPC style service, SOAP messages correspond to remote functions with parameters
- In a document-oriented service, SOAP messages contain XML documents





Document-oriented services always use literal SOAP messages, typically defining the XML payload in an XML Schema.

Document-oriented systems tend to be more loosely coupled than RPC systems, sometimes using asynchronous invocation and alternative transports, such as JMS.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Document-Oriented Java APIs

- JAX-WS document-oriented services can work with documents in several ways:
  - "Transparent" JAX-B
  - SAAJ
  - Standard XML APIs like DOM, StAX and SAX
  - Other APIs/frameworks by treating the XML as plain text
- Transparent JAX-B endpoints use the @WebService annotation; all others use @WebServiceProvider

"Transparent" JAX-B is not a term from the specification, but it describes how behind the scenes, JAX-WS delegates Java/XML mapping to JAX-B.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Transparent Document-Oriented Services

- JAX-WS services with @WebService and @SOAPBinding with **parameterStyle** as **BARE** expose document-oriented service methods
- JAX-WS delegates to JAX-B to map to/from Java and XML
- BARE methods must have at most one input parameter and at most one return value that act as the document

```
1    @WebService
2    @SOAPBinding(style=Style.DOCUMENT,
3      use=Use.LITERAL,
4      parameterStyle=ParameterStyle.BARE)
5    public class Endpoint10
6    {
7      public MyBean1 method12(MyBean2 mb) {...}
8    }
```

JAX-WS/JAX-B provide a high-level view of the document so the programmer doesn't need to work with XML, but can instead work solely with Java objects.

The types used as documents must adhere to the JAX-B specification.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Low Level Document-Oriented Services

- For services that need to access messages at a low level, you can write a service endpoint annotated with **@WebServiceProvder** and implement the **javax.xml.ws.Provider** generic interface

```
1    @WebServiceProvider
2    @ServiceMode(Mode.MESSAGE)
3    public class SAAJEndpoint
4      implements Provider<SOAPMessage>
5    {
6      public SOAPMessage invoke(SOAPMessage request)
7      {
8        . . .
9      }
10   }
```

You can annotate an endpoint class with either @WebService or @WebServiceProvider, but not both. @WebService is for endpoints written at a high level with JAX-B XML binding. @WebServiceProvider is for endpoints that access the message with no data binding.

Note that the JAX-B approach is generally far simpler than using this non-data-binding approach. The trade-off is that you get more control.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Provider Interface

- **Provider** is a *generic* interface that you must parameterize
- The JAX-WS specification mandates that all implementations support at least the following Provider types:
    - **javax.xml.transform.Source** Lets services use standard XML APIs like DOM and SAX to work with documents
    - **javax.activation.DataSource** Lets HTTP (not SOAP) services work with the raw content of a document
    - **javax.xml.soap.SOAPMessage** Lets SOAP services use SAAJ to work with documents

---

The Provider interface contains a single method, "invoke" that's defined as:

```
public <T> invoke(<T> msg)
```

The "T" in the invoke method must match the type used to parameterize the interface itself.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @WebServiceProvider Annotation

- The **@WebServiceProvider** annotation lets you provide configuration details for a low-level service

Element	Description
portName	The WSDL "port" name
serviceName	The WSDL "service" name
targetNamespace	The WSDL targetNamespace
wsdlLocation	Location of actual WSDL

```
@WebServiceProvider(wsdlLocation="WEB-INF/wsdl/my.wsdl",
  serviceName="MyService",
  targetNamespace="http://bigcompany.com/test",
  portName="MyPort")
```


Most low-level document services are written in top-down fashion, where someone writes the WSDL instead of having it generated in the bottom-up approach. So when you write the endpoint class, you specify the location of the WSDL and important elements within the WSDL.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The @ServiceMode Annotation

- The **@ServiceMode** annotation lets you configure which part of a SOAP message the endpoint works with

Element	Values
value	Service.Mode.MESSAGE Service.Mode.PAYLOAD

```
@ServiceMode(Mode.MESSAGE)
```

You must use "MESSAGE" mode if the endpoint needs to access SOAP headers.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Steps for Writing a Document-Oriented Service

- Write a top-down WSDL for the service, including document schemas
- Determine the Provider type and the ServiceMode
- Write and deploy the service

test.wsdl

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2    <wsdl:definitions xmlns:tns="http://bigcompany.com/test"
3        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
4        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5        name="test" targetNamespace="http://bigcompany.com/test"
6        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
7        <wsdl:types>
8            <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
9                targetNamespace="http://bigcompany.com/test">
10               <xsd:element name="InputDocument">
11                   <xsd:complexType>
12                       <xsd:sequence>
13                           <xsd:element name="element1" type="xsd:string"/>
14                           <xsd:element name="element2" type="xsd:int"/>
15                       </xsd:sequence>
16                   </xsd:complexType>
17               </xsd:element>
18               <xsd:element name="OutputDocument">
19                   <xsd:complexType>
20                       <xsd:sequence>
21                           <xsd:element name="element3" type="xsd:double"/>
22                           <xsd:element name="element4" type="xsd:double"/>
23                       </xsd:sequence>
24                   </xsd:complexType>
25               </xsd:element>
26           </xsd:schema>
27       </wsdl:types>
28       <wsdl:message name="processDocumentRequest">
29           <wsdl:part name="inputDoc" element="tns:InputDocument"></wsdl:part>
30       </wsdl:message>
31       <wsdl:message name="processDocumentResponse">
32           <wsdl:part name="outputDoc" element="tns:OutputDocument"></wsdl:part>
33       </wsdl:message>
34       <wsdl:portType name="MyPortType">
35           <wsdl:operation name="processDocument">
36               <wsdl:input message="tns:processDocumentRequest"></wsdl:input>
37               <wsdl:output message="tns:processDocumentResponse"></wsdl:output>
38           </wsdl:operation>
39       </wsdl:portType>
40       <wsdl:binding name="MyBinding" type="tns:MyPortType">
41           <soap:binding style="document"
42               transport="http://schemas.xmlsoap.org/soap/http" />
43           <wsdl:operation name="processDocument">
44               <soap:operation
45                   soapAction="http://bigcompany.com/test/processDocument" />
```

```
46              <wsdl:input>
47                  <soap:body use="literal" />
48              </wsdl:input>
49              <wsdl:output>
50                  <soap:body use="literal" />
51              </wsdl:output>
52          </wsdl:operation>
53      </wsdl:binding>
54      <wsdl:service name="MyServiceSAAJ">
55          <wsdl:port name="MyPortSAAJ" binding="tns:MyBinding">
56              <soap:address
57                location="http://localhost:8080/testDocumentServiceWeb/MyServiceSAAJ
58          </wsdl:port>
59      </wsdl:service>
60      <wsdl:service name="MyServiceDOM">
61          <wsdl:port name="MyPortDOM" binding="tns:MyBinding">
62              <soap:address
63                location="http://localhost:8080/testDocumentServiceWeb/MyServiceDOM"
64          </wsdl:port>
65      </wsdl:service>
66  </wsdl:definitions>
```

SAAJ Endpoint Example

```
1    <!-- Sample Request Document -->
2    <m:InputDocument xmlns:m='http://bigcompany.com/test'>
3      <element1>abcd</element1>
4      <element2>42</element2>
5    </m:InputDocument>
6
7    <!-- Sample Response Document -->
8    <m:OutputDocument xmlns:m='http://bigcompany.com/test'>
9      <element3>123.44</element3>
10     <element4>567.89</element4>
11   </m:OutputDocument>
```

SAAJEndpoint.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package service;
2
3    import java.util.Iterator;
4
5    import javax.xml.namespace.QName;
6    import javax.xml.soap.MessageFactory;
7    import javax.xml.soap.SOAPBody;
8    import javax.xml.soap.SOAPElement;
9    import javax.xml.soap.SOAPEnvelope;
10   import javax.xml.soap.SOAPException;
11   import javax.xml.soap.SOAPMessage;
12   import javax.xml.soap.SOAPPart;
13   import javax.xml.ws.Provider;
14   import javax.xml.ws.Service.Mode;
15   import javax.xml.ws.ServiceMode;
16   import javax.xml.ws.WebServiceProvider;
17
18   @WebServiceProvider(wsdlLocation = "WEB-INF/wsdl/test.wsdl",
19     serviceName = "MyServiceSAAJ",
20     targetNamespace = "http://bigcompany.com/test",
21     portName = "MyPortSAAJ")
22   @ServiceMode(Mode.MESSAGE)
23   public class SAAJEndpoint implements Provider<SOAPMessage>
24   {
25       @Override
26       public SOAPMessage invoke(SOAPMessage request)
27       {
28           SOAPMessage reply = null;
29
30           try
31           {
32               processMessage(request);
33
34               reply = createMessage();
35           }
36           catch (SOAPException e)
37           {
38               e.printStackTrace();
39           }
40
41           return reply;
42       }
43       private void processMessage(SOAPMessage request)
44           throws SOAPException
45       {
```

```
46          SOAPBody body = request.getSOAPBody();
47          Iterator iter = body.getChildElements();
48
49          while (iter.hasNext())
50          {
51              Object o = iter.next();
52              if (o instanceof SOAPElement)
53              {
54                  SOAPElement payload = (SOAPElement) o;
55
56                  Iterator childIter = payload.getChildElements();
57                  while (childIter.hasNext())
58                  {
59                      o = childIter.next();
60                      if (o instanceof SOAPElement)
61                      {
62                          SOAPElement anElement = (SOAPElement) o;
63                          String val = anElement.getValue();
64                          if (anElement.getNodeName().equals("element1"))
65                          {
66                              System.out.println("Element 1: " + val);
67                          }
68                          else if (anElement.getNodeName().equals("element2"))
69                          {
70                              int n = Integer.parseInt(val.trim());
71                              System.out.println("Element 2: " + n);
72                          }
73                      }
74                  }
75              }
76          }
77      }
78
79      private SOAPMessage createMessage() throws SOAPException
80      {
81          MessageFactory fact = MessageFactory.newInstance();
82          SOAPMessage msg = fact.createMessage();
83
84          SOAPPart soapPart = msg.getSOAPPart();
85          SOAPEnvelope env = soapPart.getEnvelope();
86          SOAPBody body = env.getBody();
87
88          QName payloadQname = new QName("http://bigcompany.com/test",
89                  "OutputDocument", "m");
90          SOAPElement payload = body.addChildElement(payloadQname);
```

```
91
92            QName element3QName = new QName("element3");
93            SOAPElement element3 = payload.addChildElement(element3QName);
94
95            element3.addTextNode("44.55");
96
97            QName element4QName = new QName("element4");
98            SOAPElement element4 = payload.addChildElement(element4QName);
99
100           element4.addTextNode("66.77");
101
102           return msg;
103       }
104   }
```

DOM Endpoint Example

DOMEndpoint.java

This DOM example uses the same request and response documents as the SAAJ example.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package service;
2
3    import java.io.IOException;
4    import java.io.StringReader;
5
6    import javax.xml.parsers.DocumentBuilder;
7    import javax.xml.parsers.DocumentBuilderFactory;
8    import javax.xml.parsers.ParserConfigurationException;
9    import javax.xml.transform.Source;
10   import javax.xml.transform.Transformer;
11   import javax.xml.transform.TransformerConfigurationException;
12   import javax.xml.transform.TransformerException;
13   import javax.xml.transform.TransformerFactory;
14   import javax.xml.transform.TransformerFactoryConfigurationError;
15   import javax.xml.transform.dom.DOMResult;
16   import javax.xml.transform.dom.DOMSource;
17   import javax.xml.ws.Provider;
18   import javax.xml.ws.Service.Mode;
19   import javax.xml.ws.ServiceMode;
20   import javax.xml.ws.WebServiceProvider;
21
22   import org.w3c.dom.DOMException;
23   import org.w3c.dom.Document;
24   import org.w3c.dom.Element;
25   import org.w3c.dom.NodeList;
26   import org.xml.sax.InputSource;
27   import org.xml.sax.SAXException;
28
29   @WebServiceProvider(wsdlLocation = "WEB-INF/wsdl/test.wsdl",
30           serviceName = "MyServiceDOM",
31           targetNamespace = "http://bigcompany.com/test",
32           portName = "MyPortDOM")
33   @ServiceMode(Mode.PAYLOAD)
34   public class DOMEndpoint implements Provider<Source>
35   {
36       @Override
37       public DOMSource invoke(Source request)
38       {
39           DOMSource reply = null;
40
41           processDocument(request);
42
43           reply = createDocument();
44
45           return reply;
```

```
46      }
47
48      private void processDocument(Source request)
49      {
50          try
51          {
52              DOMResult dr = new DOMResult();
53              Transformer trans = TransformerFactory.newInstance()
54                      .newTransformer();
55              trans.transform(request, dr);
56
57              Document payload = (Document) dr.getNode();
58              NodeList nodes = payload.getElementsByTagName("element1");
59              if (nodes.getLength() != 0)
60              {
61                  Element element1 = (Element) nodes.item(0);
62                  String val = element1.getTextContent();
63                  System.out.println("Element 1: " + val);
64              }
65
66              nodes = payload.getElementsByTagName("element2");
67              if (nodes.getLength() != 0)
68              {
69                  Element element2 = (Element) nodes.item(0);
70                  String val = element2.getTextContent();
71                  System.out.println("Element 2: " + Integer.parseInt(val));
72              }
73          }
74          catch (NumberFormatException e)
75          {
76              e.printStackTrace();
77          }
78          catch (TransformerConfigurationException e)
79          {
80              e.printStackTrace();
81          }
82          catch (DOMException e)
83          {
84              e.printStackTrace();
85          }
86          catch (TransformerFactoryConfigurationError e)
87          {
88              e.printStackTrace();
89          }
90          catch (TransformerException e)
```

```
91          {
92              e.printStackTrace();
93          }
94      }
95
96      private DOMSource createDocument()
97      {
98          DOMSource retVal = null;
99
100         try
101         {
102             String xml =
103                 "<m:OutputDocument xmlns:m='http://bigcompany.com/test'>\n" +
104                 "<element3>123.44</element3>\n" +
105                 "<element4>567.89</element4>\n" +
106                 "</m:OutputDocument>";
107
108             System.out.println(xml);
109
110             StringReader sr = new StringReader(xml);
111             InputSource is = new InputSource(sr);
112
113             DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
114             dbf.setNamespaceAware(true);
115             DocumentBuilder db = dbf.newDocumentBuilder();
116
117             Document doc = db.parse(is);
118
119             retVal = new DOMSource(doc);
120         }
121         catch (ParserConfigurationException e)
122         {
123             e.printStackTrace();
124         }
125         catch (SAXException e)
126         {
127             e.printStackTrace();
128         }
129         catch (IOException e)
130         {
131             e.printStackTrace();
132         }
133
134         return retVal;
135     }
```

```
136   }
```

Dispatch Clients

- Proxy-based clients use a high-level interface to the service: they don't need to work with the raw message itself
- Sometimes clients need more low-level control, so JAX-WS provides the **dispatch** client programming model
- Dispatch clients can work in one of two modes:
    - **Message** - Lets SOAP clients work with raw SOAP message
    - **Payload** - Clients work with just the contents of the message, not the entire message

The Dispatch API replaces the Dynamic Invocation Interface found in JAX-RPC.

Note that JAX-WS supports other "protocols" other than SOAP, for example plain HTTP for RESTful services. Here were are only covering SOAP-based services.

Also note that proxy-based clients can work with document/literal (BARE) services - the JAX-WS/JAX-B infrastructure on the client maps the document to/from Java objects.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Dispatch Interface

- The **javax.xml.ws.Dispatch** interface uses JSE **generics** so clients can work with messages
- The JAX-WS specification requires support for at least the following types:
    - **javax.xml.transform.Source** - Lets clients use standard XML APIs like DOM and SAX passed as the payload to a service
    - **JAX-B objects** - Lets clients use an object graph of JAX-B objects passed as the payload to a service
    - **javax.xml.soap.SOAPMessage** - Lets clients use SAAJ to build SOAP messages
    - **javax.activation.DataSource** - Lets clients access HTTP-based (non-SOAP) services

We will cover the JAX-B approach in a later chapter.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

SAAJ Client Example

SAAJClientForSAAJEndpoint.java

This client uses the same request and response documents shown earlier.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package client;
2
3    import java.io.IOException;
4    import java.net.URL;
5    import java.util.Iterator;
6
7    import javax.xml.namespace.QName;
8    import javax.xml.soap.MessageFactory;
9    import javax.xml.soap.SOAPBody;
10   import javax.xml.soap.SOAPElement;
11   import javax.xml.soap.SOAPEnvelope;
12   import javax.xml.soap.SOAPException;
13   import javax.xml.soap.SOAPFault;
14   import javax.xml.soap.SOAPMessage;
15   import javax.xml.soap.SOAPPart;
16   import javax.xml.ws.Dispatch;
17   import javax.xml.ws.Service;
18
19   public class SAAJClientForSAAJEndpoint
20   {
21       private static SOAPMessage sendMessageSAAJ(Dispatch<SOAPMessage> dispatch)
22       {
23           SOAPMessage responseMsg = null;
24           try
25           {
26               String targetNamespace = "http://bigcompany.com/test";
27               SOAPMessage requestMsg = MessageFactory.newInstance()
28                       .createMessage();
29
30               SOAPPart reqSoapPart = requestMsg.getSOAPPart();
31               SOAPEnvelope reqEnv = reqSoapPart.getEnvelope();
32               SOAPBody reqBody = reqEnv.getBody();
33
34               QName reqPayloadQname = new QName(targetNamespace, "InputMessage",
35                       "m");
36               SOAPElement reqPayload = reqBody.addChildElement(reqPayloadQname);
37
38               QName elem1QName = new QName("element1");
39               SOAPElement elem1 = reqPayload.addChildElement(elem1QName);
40
41               elem1.addTextNode("abcd");
42
43               QName elem2QName = new QName("element2");
44               SOAPElement elem2 = reqPayload.addChildElement(elem2QName);
45
```

```
46              elem2.addTextNode("123");
47
48              responseMsg = dispatch.invoke(requestMsg);
49          }
50          catch (SOAPException e)
51          {
52              e.printStackTrace();
53          }
54
55          return responseMsg;
56      }
57
58      private static void processResponseSAAJ(SOAPMessage responseMsg)
59      {
60          try
61          {
62              SOAPPart respSoapPart = responseMsg.getSOAPPart();
63              SOAPEnvelope respEnv = respSoapPart.getEnvelope();
64              SOAPBody respBody = respEnv.getBody();
65
66              // check for SOAPFault
67              if (respBody.hasFault())
68              {
69                  SOAPFault fault = respBody.getFault();
70                  System.out.println("FAULT: " + fault.getFaultString());
71                  return;
72              }
73
74              // process document root element
75              Iterator iter = respBody.getChildElements();
76              while (iter.hasNext())
77              {
78                  Object o = iter.next();
79                  if (o instanceof SOAPElement)
80                  {
81                      // process child elements
82                      SOAPElement root = (SOAPElement) o;
83
84                      Iterator iterChild = root.getChildElements();
85                      while (iterChild.hasNext())
86                      {
87                          Object oChild = iterChild.next();
88                          if (oChild instanceof SOAPElement)
89                          {
90                              SOAPElement child = (SOAPElement) oChild;
```

```
91                              if (child.getNodeName().equals("element3"))
92                              {
93                                  double val = Double.parseDouble(child
94                                          .getValue());
95                                  System.out.println("Element 3: " + val);
96                              }
97                              if (child.getNodeName().equals("element4"))
98                              {
99                                  double val = Double.parseDouble(child
100                                         .getValue());
101                                 System.out.println("Element 3: " + val);
102                             }
103                         }
104                     }
105                 }
106             }
107         }
108         catch (NumberFormatException e)
109         {
110             e.printStackTrace();
111         }
112         catch (SOAPException e)
113         {
114             e.printStackTrace();
115         }
116     }
117
118     public static void main(String[] args) throws Exception
119     {
120         String targetNamespace = "http://bigcompany.com/test";
121
122         // create dispatch
123         String serviceName = "MyServiceSAAJ";
124         URL wsdlURL = new URL(
125                 "http://localhost:8080/testDocumentServicesWeb/MyServiceSAAJ?wsdl"
126         QName serviceQname = new QName(targetNamespace, serviceName);
127         Service service = Service.create(wsdlURL, serviceQname);
128
129         String portName = "MyPortSAAJ";
130         QName portQname = new QName(targetNamespace, portName);
131
132         Dispatch<SOAPMessage> dispatch = service.createDispatch(portQname,
133                 SOAPMessage.class, Service.Mode.MESSAGE);
134
135         SOAPMessage responseMsg = sendMessageSAAJ(dispatch);
```

```
136
137             processResponseSAAJ(responseMsg);
138         }
139     }
```

DOM Client Example

DOMClientForDOMEndpoint.java

This client uses the same request and response documents shown earlier.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package client;
2
3    import java.io.IOException;
4    import java.io.StringReader;
5    import java.net.URL;
6
7    import javax.xml.namespace.QName;
8    import javax.xml.parsers.DocumentBuilder;
9    import javax.xml.parsers.DocumentBuilderFactory;
10   import javax.xml.parsers.ParserConfigurationException;
11   import javax.xml.transform.Source;
12   import javax.xml.transform.Transformer;
13   import javax.xml.transform.TransformerConfigurationException;
14   import javax.xml.transform.TransformerException;
15   import javax.xml.transform.TransformerFactory;
16   import javax.xml.transform.TransformerFactoryConfigurationError;
17   import javax.xml.transform.dom.DOMResult;
18   import javax.xml.transform.dom.DOMSource;
19   import javax.xml.ws.Dispatch;
20   import javax.xml.ws.Service;
21
22   import org.w3c.dom.DOMException;
23   import org.w3c.dom.Document;
24   import org.w3c.dom.Element;
25   import org.w3c.dom.NodeList;
26   import org.xml.sax.InputSource;
27   import org.xml.sax.SAXException;
28
29   public class DOMClientForDOMEndpoint
30   {
31       private static Source sendMessageDOM(Dispatch<Source> dispatch)
32       {
33           Source responseMsg = null;
34
35           try
36           {
37               String xml =
38                   "<m:InputDocument xmlns:m='http://bigcompany.com/test'>\n" +
39                     "<element1>abcd</element1>\n" +
40                     "<element2>42</element2>\n" +
41                   "</m:InputDocument>";
42
43               StringReader sr = new StringReader(xml);
44               InputSource is = new InputSource(sr);
45
```

```
46              DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
47              dbf.setNamespaceAware(true);
48              DocumentBuilder db = dbf.newDocumentBuilder();
49
50              Document doc = db.parse(is);
51              Source input = new DOMSource(doc);
52
53              responseMsg = dispatch.invoke(input);
54          }
55          catch (ParserConfigurationException e)
56          {
57              e.printStackTrace();
58          }
59          catch (SAXException e)
60          {
61              e.printStackTrace();
62          }
63          catch (IOException e)
64          {
65              e.printStackTrace();
66          }
67
68          return responseMsg;
69      }
70
71      private static void processResponseDOM(Source responseMsg)
72      {
73          try
74          {
75              DOMResult dr = new DOMResult();
76              Transformer trans = TransformerFactory.newInstance()
77                      .newTransformer();
78              trans.transform(responseMsg, dr);
79
80              Document payload = (Document) dr.getNode();
81              NodeList nodes = payload.getElementsByTagName("element3");
82              if (nodes.getLength() != 0)
83              {
84                  Element element3 = (Element) nodes.item(0);
85                  String val = element3.getTextContent();
86                  System.out.println("Element 3: " + Double.parseDouble(val));
87              }
88
89              nodes = payload.getElementsByTagName("element4");
90              if (nodes.getLength() != 0)
```

```
91              {
92                  Element element4 = (Element) nodes.item(0);
93                  String val = element4.getTextContent();
94                  System.out.println("Element 4: " + Double.parseDouble(val));
95              }
96          }
97          catch (NumberFormatException e)
98          {
99              e.printStackTrace();
100         }
101         catch (TransformerConfigurationException e)
102         {
103             e.printStackTrace();
104         }
105         catch (DOMException e)
106         {
107             e.printStackTrace();
108         }
109         catch (TransformerFactoryConfigurationError e)
110         {
111             e.printStackTrace();
112         }
113         catch (TransformerException e)
114         {
115             e.printStackTrace();
116         }
117     }
118
119     public static void main(String[] args) throws Exception
120     {
121         String targetNamespace = "http://bigcompany.com/test";
122
123         // create dispatch
124         String serviceName = "MyServiceDOM";
125         URL wsdlURL = new URL(
126                 "http://localhost:8080/testDocumentServicesWeb/MyServiceDOM?wsdl")
127         QName serviceQname = new QName(targetNamespace, serviceName);
128         Service service = Service.create(wsdlURL, serviceQname);
129
130         String portName = "MyPortDOM";
131         QName portQname = new QName(targetNamespace, portName);
132
133         Dispatch<Source> dispatch = service.createDispatch(portQname,
134                 Source.class, Service.Mode.PAYLOAD);
135
```

```
136           Source responseMsg = sendMessageDOM(dispatch);
137
138           processResponseDOM(responseMsg);
139       }
140   }
```

Chapter Summary

In this chapter, you learned

- The fundamentals of document-oriented SOAP
- About various APIs for processing documents

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum