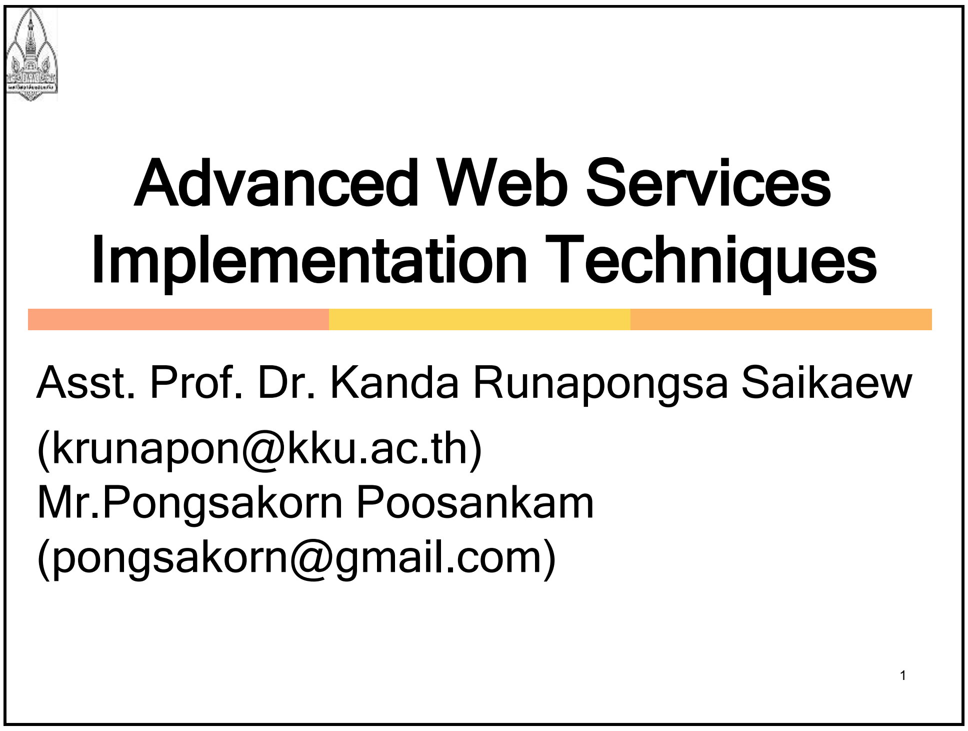
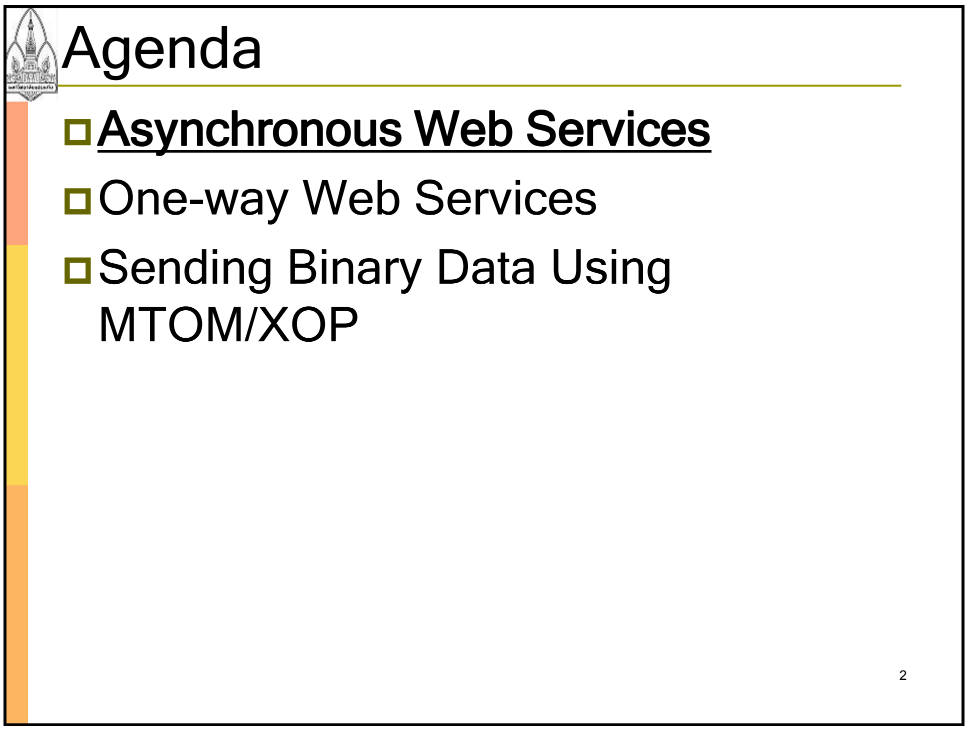


Advanced Web Services Implementation Techniques

Asst. Prof. Dr. Kanda Runapongsa Saikaew
(krunapon@kku.ac.th)
Mr.Pongsakorn Poosankam
(pongsakorn@gmail.com)

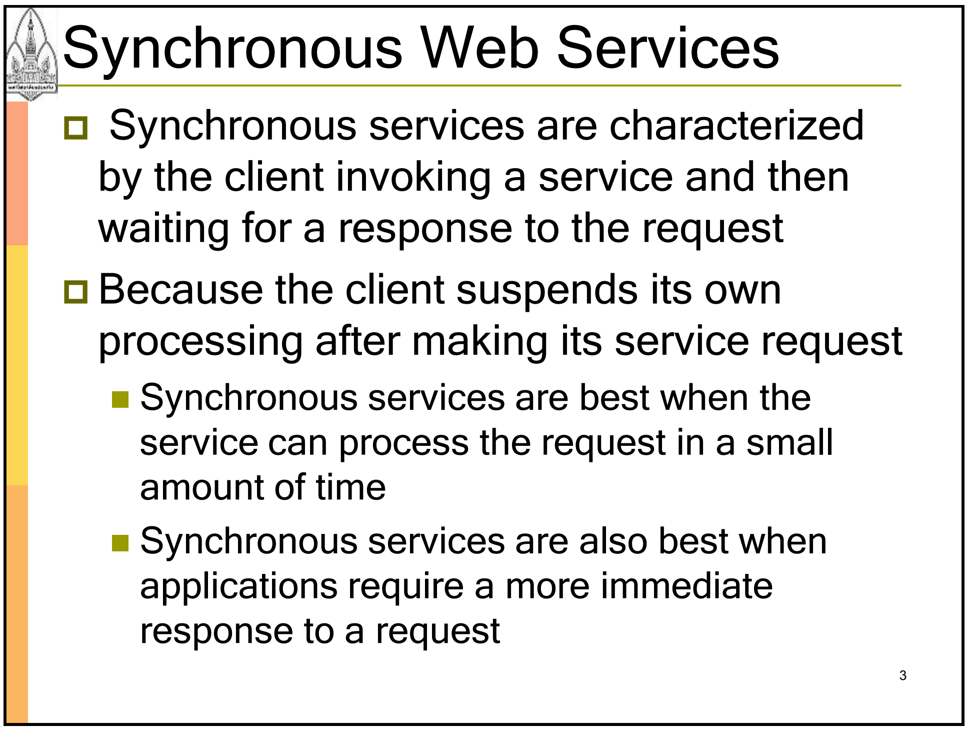
1



Agenda

- **Asynchronous Web Services**
- One-way Web Services
- Sending Binary Data Using MTOM/XOP

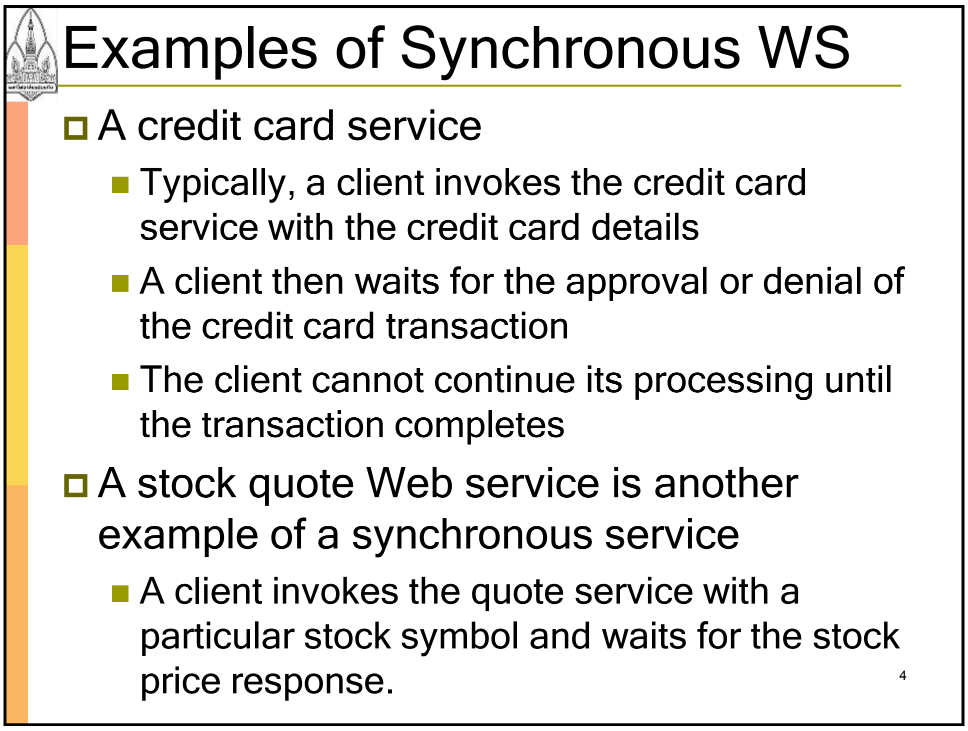
2



Synchronous Web Services

- Synchronous services are characterized by the client invoking a service and then waiting for a response to the request
- Because the client suspends its own processing after making its service request
  - Synchronous services are best when the service can process the request in a small amount of time
  - Synchronous services are also best when applications require a more immediate response to a request

3

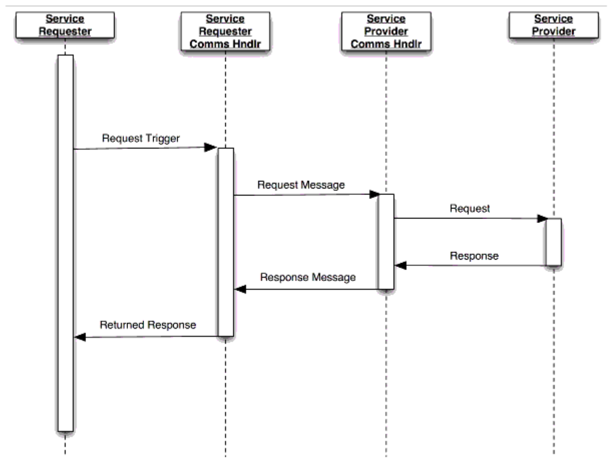


Examples of Synchronous WS

- A credit card service
  - Typically, a client invokes the credit card service with the credit card details
  - A client then waits for the approval or denial of the credit card transaction
  - The client cannot continue its processing until the transaction completes
- A stock quote Web service is another example of a synchronous service
  - A client invokes the quote service with a particular stock symbol and waits for the stock price response.

4

Synchronous Web Service Model

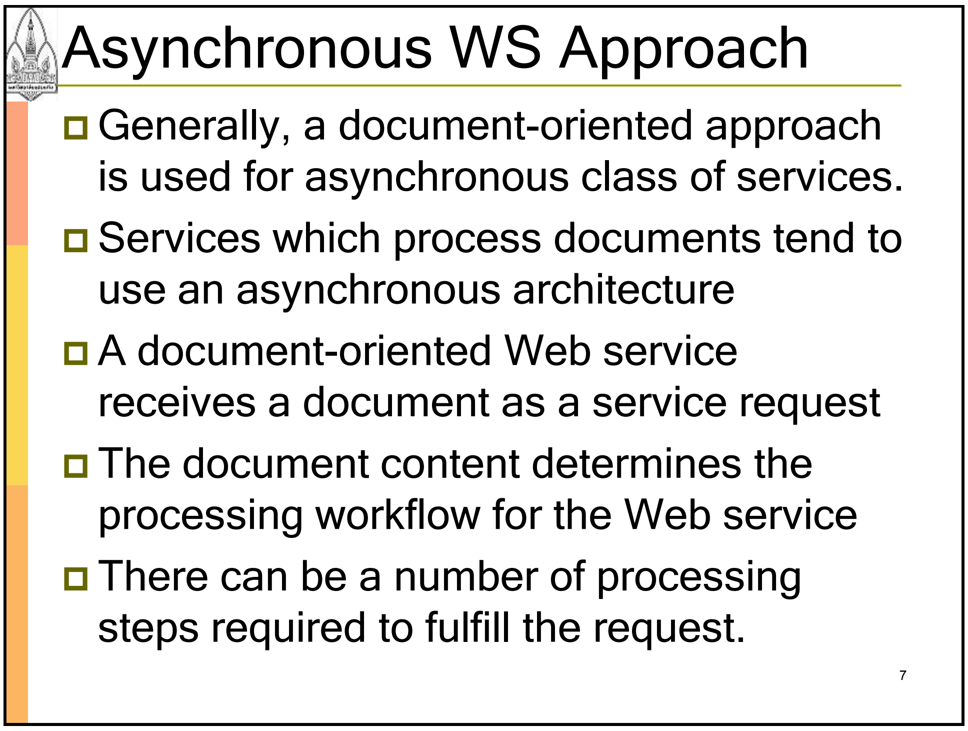


5

Asynchronous Web Services

- With asynchronous services, the client invokes the service but does not -- or cannot -- wait for the response
- The client often does not want to wait for the response because it may take a significant amount of time
- The client can continue with some other processing rather than wait for the response
- Later, when it does receive the response, it resumes whatever processing initiated the service request

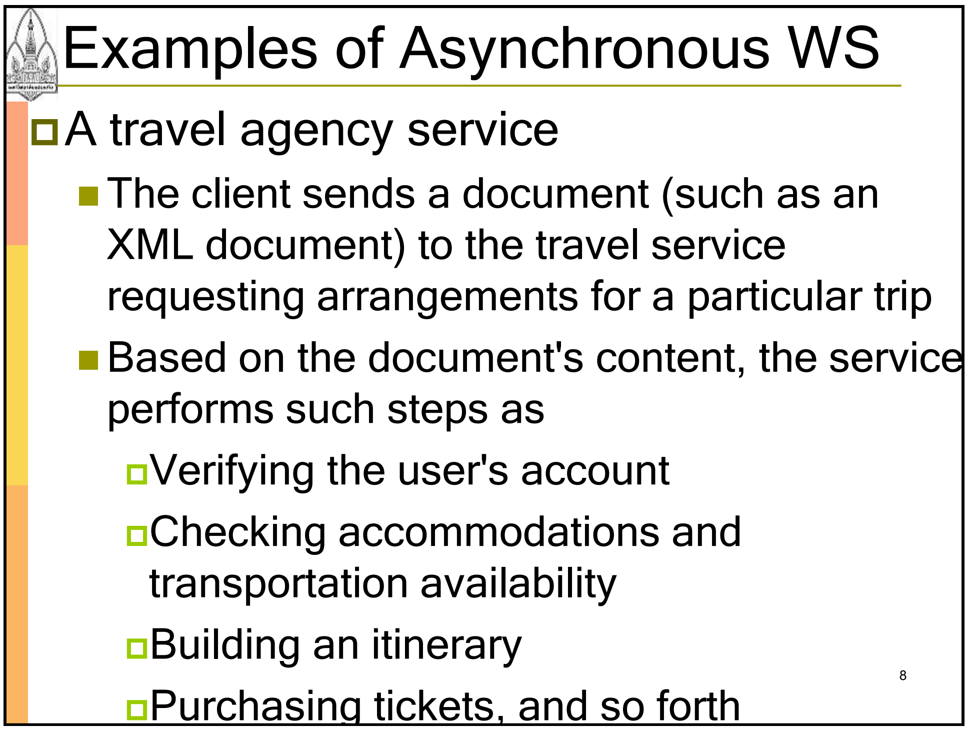
6



Asynchronous WS Approach

- Generally, a document-oriented approach is used for asynchronous class of services.
- Services which process documents tend to use an asynchronous architecture
- A document-oriented Web service receives a document as a service request
- The document content determines the processing workflow for the Web service
- There can be a number of processing steps required to fulfill the request.

7



Examples of Asynchronous WS

- A travel agency service
  - The client sends a document (such as an XML document) to the travel service requesting arrangements for a particular trip
  - Based on the document's content, the service performs such steps as
    - Verifying the user's account
    - Checking accommodations and transportation availability
    - Building an itinerary
    - Purchasing tickets, and so forth

8

Asynchronous Patterns

- Fire and Forget
- Callbacks
- Polling

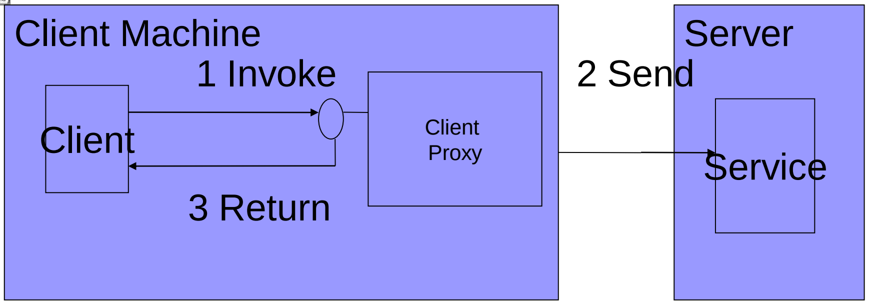
9

Fire and Forget Pattern

- Invokes the service and return immediately without ever bothering about a response.
- No waiting. Client can immediately resume the thread.
- Easy for the developers
- Loosely coupled
- No way to verify whether the request has been sent or not.
- Not Recommend

10

Fire and Forget Code Snippet



- Code Snippet

```
localhost.Service1 service = new localhost.Service1();
if(txtName.Text!=""||txtPhoneNumber.Text!="")
{
    service.BeginsendPhoneDetails(txtName.Text,txtPhoneNumber.Text,null,null);
}
else{
```

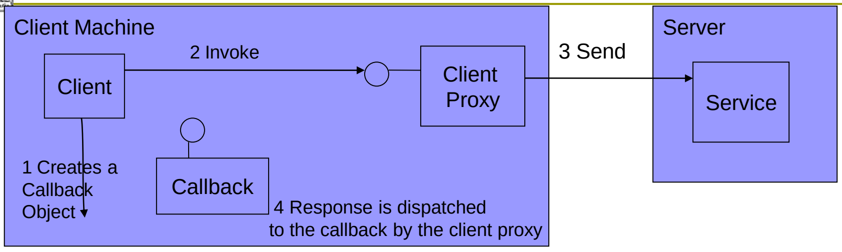
11

Callbacks Pattern

- Client provides a callback method
- Proxy will dispatch the result using the callback method
- Client has to provide the callback method
- Client has to handle the additional complexity
- Response can be retrieved asynchronously

12

Callbacks Snippet Code


Client Machine
Client
2 Invoke
Client Proxy
3 Send
Server
Service
1 Creates a Callback Object
Callback
4 Response is dispatched to the callback by the client proxy

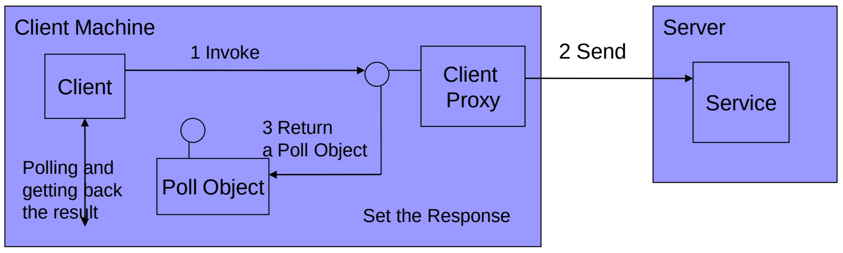
```
localhost.Service1 service = new localhost.Service1();

if(txtName.Text!="")
{
      service.BeginfindPhoneNumberForName(txtName.Text,
            new AsyncCallback(callBackMethod),null);

      for(int i=0;i<400;i++){
            progressBar1.Increment(1);
            if( progressBar1.Value==50)
                  progressBar1.Value=0;
            Thread.Sleep(20);
      }

}
    private void callBackMethod(IAsyncResult iar)
    {
         localhost.Service1 service = new localhost.Service1();
         string number =service.EndfindPhoneNumberForName(iar);
         MessageBox.Show("The Phone Number:"+number, "Phone Number");
```

Polling Pattern

- Client repeatedly polls
- Client has to wait while keep polling
- Client has to handle the complexity of this polling operation
- Response can be retrieved asynchronously

14

Polling Snippet Code

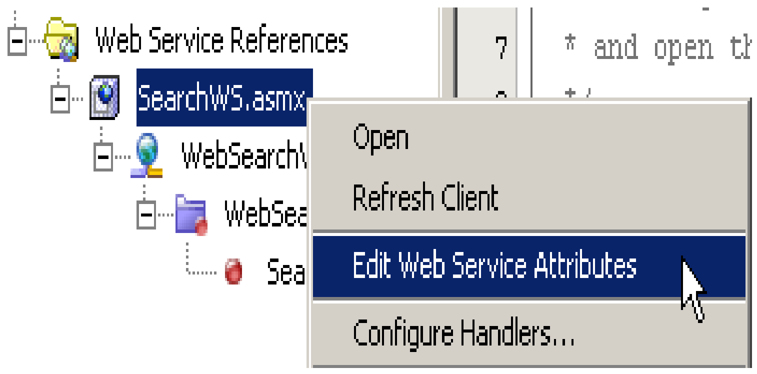


```
localhost.Service1 service = new localhost.Service1();
if(txtName.Text!="")
{    IAsyncResult iAsyncResult=null;
     iAsyncResult=service.BeginfindPhoneNumberForName(txtName.Text,
          null,null);

     while(iAsyncResult.IsCompleted!=true){
          progressBar1.Increment(1);
          Thread.Sleep(100);
     }
     txtPhoneNumber.Text=(string)service.EndfindPhoneNumberForName(iAsyncResult);
```

15

Creating Asynchronous with NetBeans (1/3)

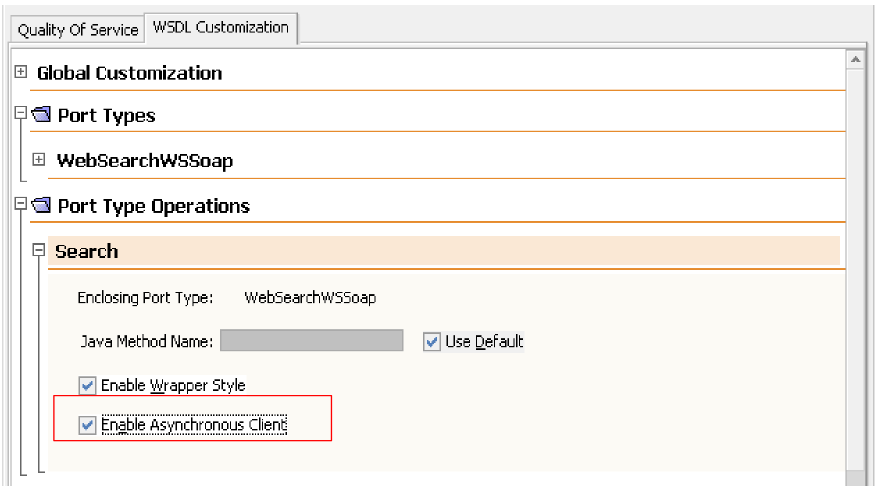
- Right Click in Web Services
- Select “Edit Web Services Attributes”



16

Creating Asynchronous with NetBeans (2/3)

- Enable Asynchronous Client



Quality Of Service | WSDL Customization

Global Customization

Port Types

WebSearchWSSoap

Port Type Operations

Search

Enclosing Port Type: WebSearchWSSoap

Java Method Name: [✓ Use Default]

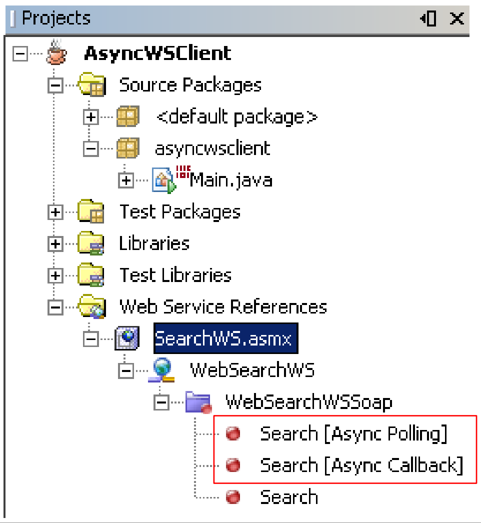
[✓ Enable Wrapper Style]

[✓ Enable Asynchronous Client]

17

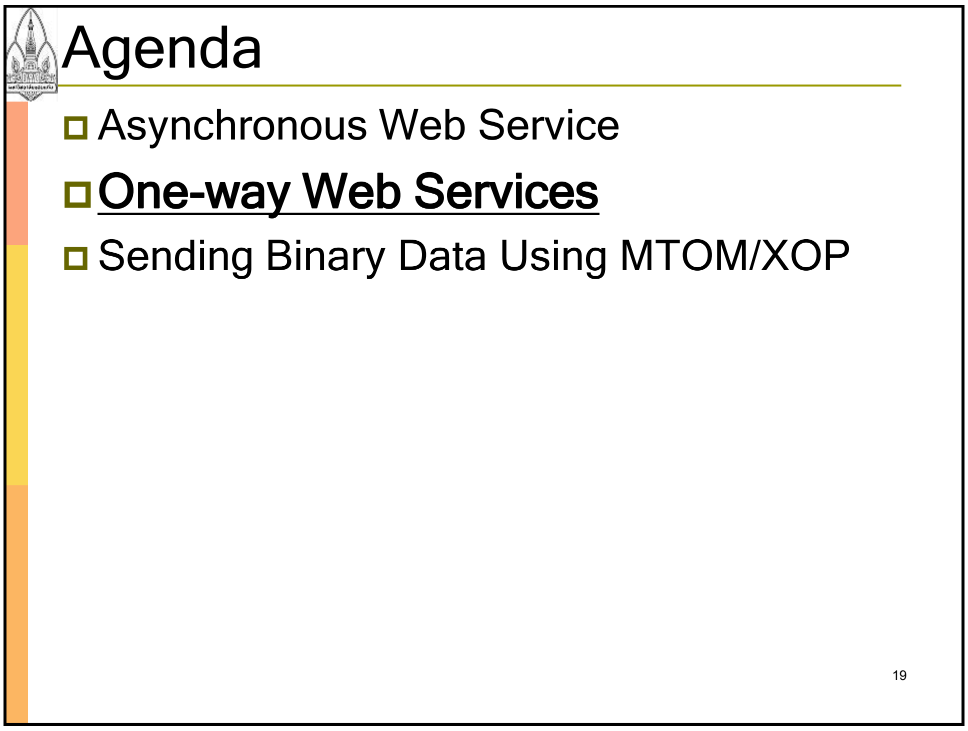
Creating Asynchronous with NetBeans (3/3)

- Appearing Async Polling and Async Callback



```
Projects
AsyncWSClient
  Source Packages
    <default package>
    asyncwsclient
      Main.java
  Test Packages
  Libraries
  Test Libraries
  Web Service References
    SearchWS.asmx
      WebSearchWS
        WebSearchWSSoap
          Search [Async Polling]
          Search [Async Callback]
          Search
```

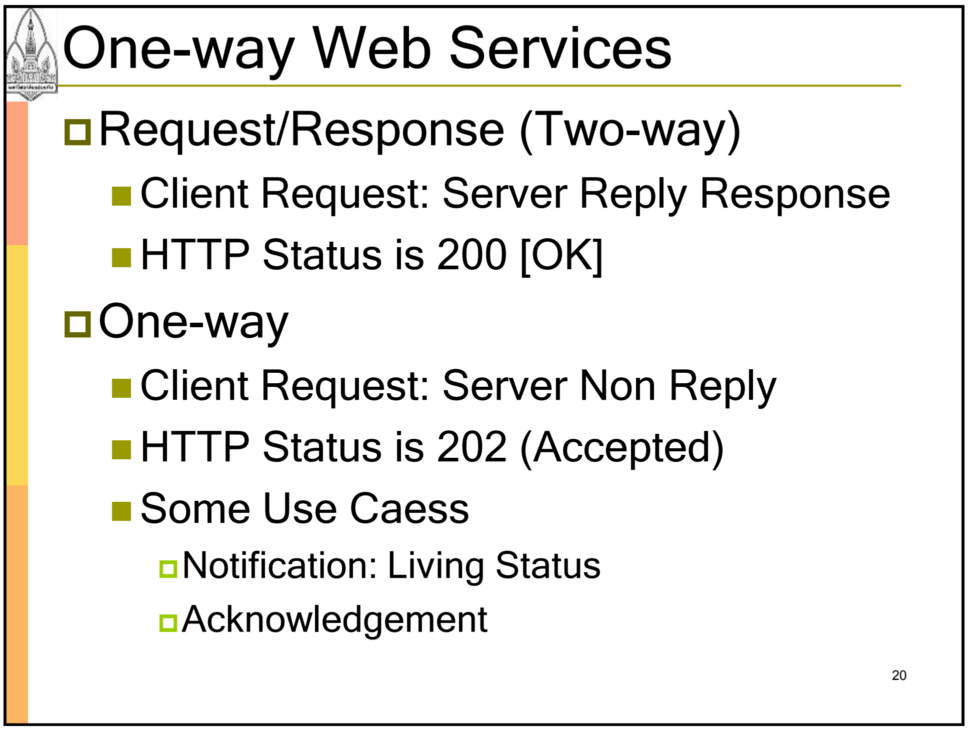
18



Agenda

- Asynchronous Web Service
- **One-way Web Services**
- Sending Binary Data Using MTOM/XOP

19



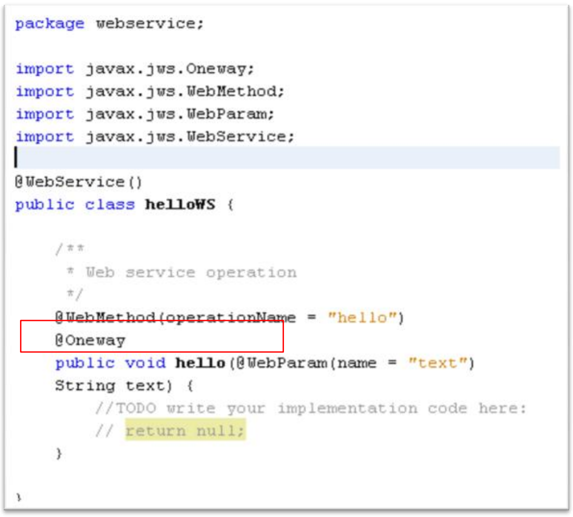
One-way Web Services

- Request/Response (Two-way)
  - Client Request: Server Reply Response
  - HTTP Status is 200 [OK]
- One-way
  - Client Request: Server Non Reply
  - HTTP Status is 202 (Accepted)
  - Some Use Caess
    - Notification: Living Status
    - Acknowledgement

20

Creating One-Way Web Services with NetBeans

- Add @Oneway annotation on top of public *void* method



```
package webservice;

import javax.jws.Oneway;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

@WebService()
public class helloWS {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "hello")
    @Oneway
    public void hello(@WebParam(name = "text")
    String text) {
        //TODO write your implementation code here:
        // return null;
    }

}
```

21

Agenda

- Asynchronous Web Services
- One-way Web Services
- **Sending Binary Data Using MTOM/XOP**

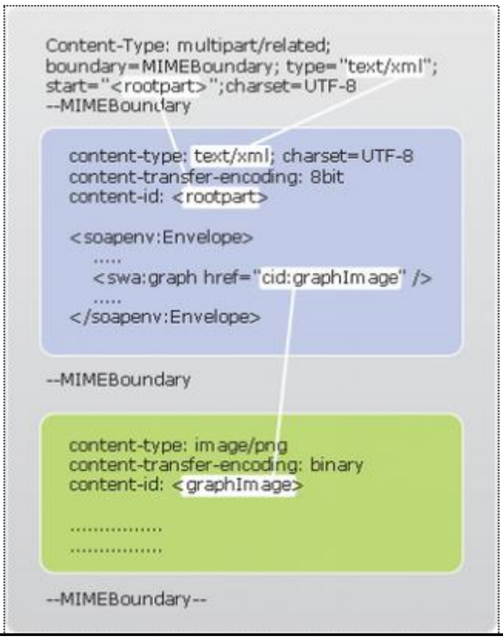
22

Motivation for MTOM/XOP

- Two approaches of sending binary data via XML
  - Embedding - Base64 encoding
  - Referencing - SOAP with Attachment
- Problem of Base64 encoding
  - Increased size
  - Added overhead
- Problem of SOAP with Attachment
  - Data is external to the document, and it isn't part of the message Infoset, thus requires two different ways of processing data

23

SOAP with Attachment



```
Content-Type: multipart/related;
boundary=MIMEBoundary; type="text/xml";
start="<rootpart>";charset=UTF-8
--MIMEBoundary

  content-type: text/xml; charset=UTF-8
  content-transfer-encoding: 8bit
  content-id: <rootpart>

  <soapenv:Envelope>
     .....
     <swa:graph href="cid:graphImage" />
     .....
  </soapenv:Envelope>

--MIMEBoundary

  content-type: image/png
  content-transfer-encoding: binary
  content-id: <graphImage>

  ................
  ................

--MIMEBoundary--
```

24

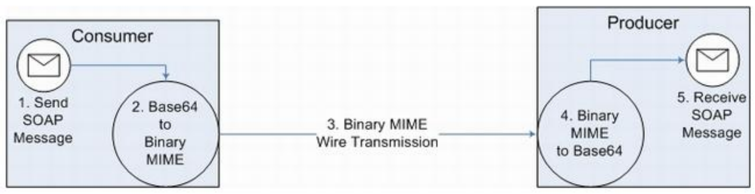
XOP (XML-Binary Optimized Packaging)

- An alternate serialization of XML that just happens to look like a MIME multipart/related package, with an XML document as the root part
- That root part is very similar to the XML serialization of the document, except that base64-encoded data is replaced by a reference to one of the MIME parts, which isn't base64 encoded.
- Let you avoid the bulk and overhead in processing associated with base64 encoding
- Can be used for any XML-based format

MTOM/XOP

- SOAP Message Transmission Optimization Mechanism/XML-binary Optimized Packaging (MTOM/XOP) defines a method for optimizing the transmission of XML data in SOAP messages
- When the transport protocol is HTTP, MIME attachments are used to carry that data while at the same time allowing both the sender and the receiver direct access to the XML data in the SOAP message
  - Without having to be aware that any MIME artifacts were used to marshal the base64Binary or hexBinary data

MTOM Case Study (1/2)



❑The Consumer application begins by sending a SOAP Message that contains complex data in Base64Binary encoded format.
❑The Base64Binary data type represents arbitrary data (e.g. Images, PDF files, Word Docs) in 65 textual characters that can be displayed as part of a SOAP Message element

27

MTOM Case Study

- A sample SOAP Body with Base64Binary encoded element <tns:data> is as follows

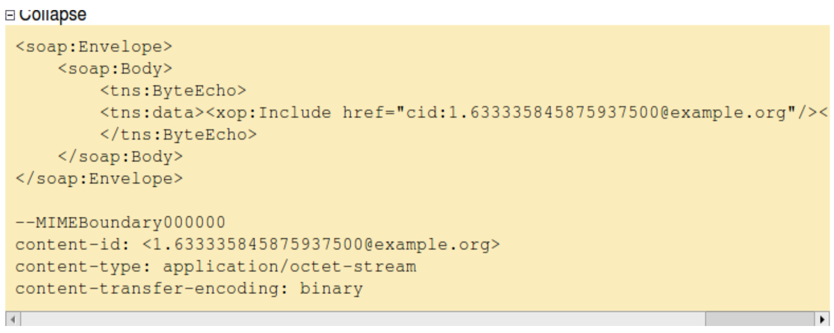
⊟ Collapse
```
<soap:Body>
    <tns:ByteEcho>
        <tns:data>JVBERi0xLjYNJeLjz9MNCjE+DQpzdGFyNCjEx0YNCg==</tns:data>
    </tns:ByteEcho>
</soap:Body>
```

- An MTOM-aware web services engine detects and converts Base64Binary data to MIME data with an XML-binary Optimization Package (xop) content type

28

Data Conversion Results

- The data conversion results in replacing the Base64Binary data with an <xop:Include> element that references the original raw bytes of the document being transmitted



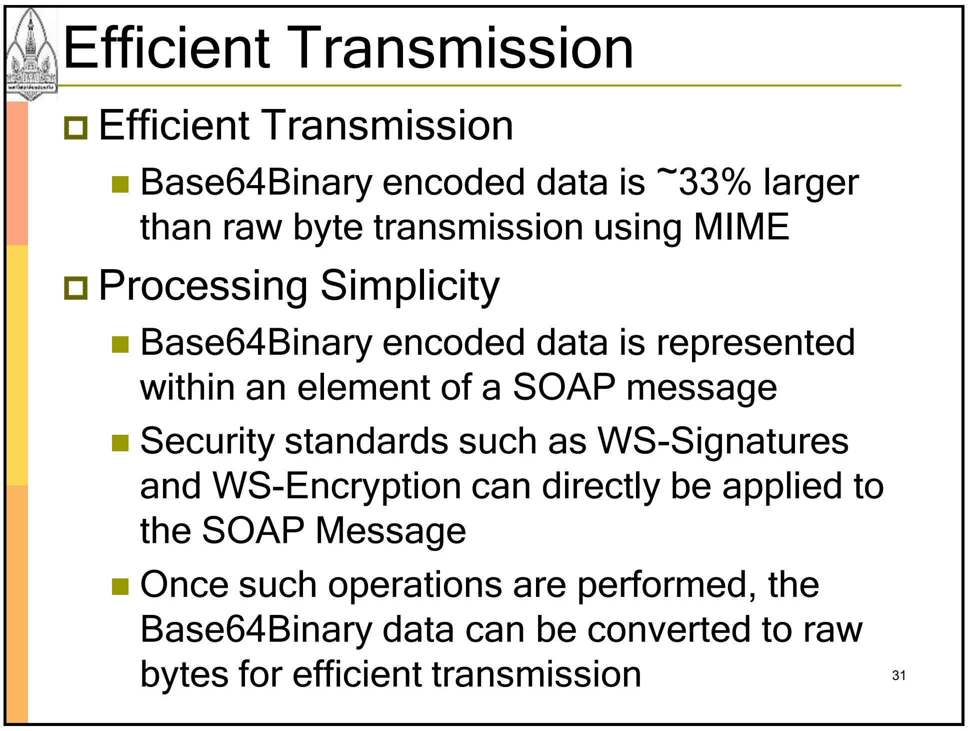
```
<soap:Envelope>
    <soap:Body>
        <tns:ByteEcho>
        <tns:data><xop:Include href="cid:1.633335845875937500@example.org"/><
        </tns:ByteEcho>
    </soap:Body>
</soap:Envelope>

--MIMEBoundary000000
content-id: <1.633335845875937500@example.org>
content-type: application/octet-stream
content-transfer-encoding: binary
```

29

Base64Binary & Raw MIME Byte

- The raw binary data, along with the SOAP Message and the MIME Boundary, is transmitted over the wire to the Producer
- The Producer then changes the raw binary data back to Base64Binary encoding for further processing
- With this conversion between Base64Binary and raw binary MIME types, MTOM provides two significant advantages

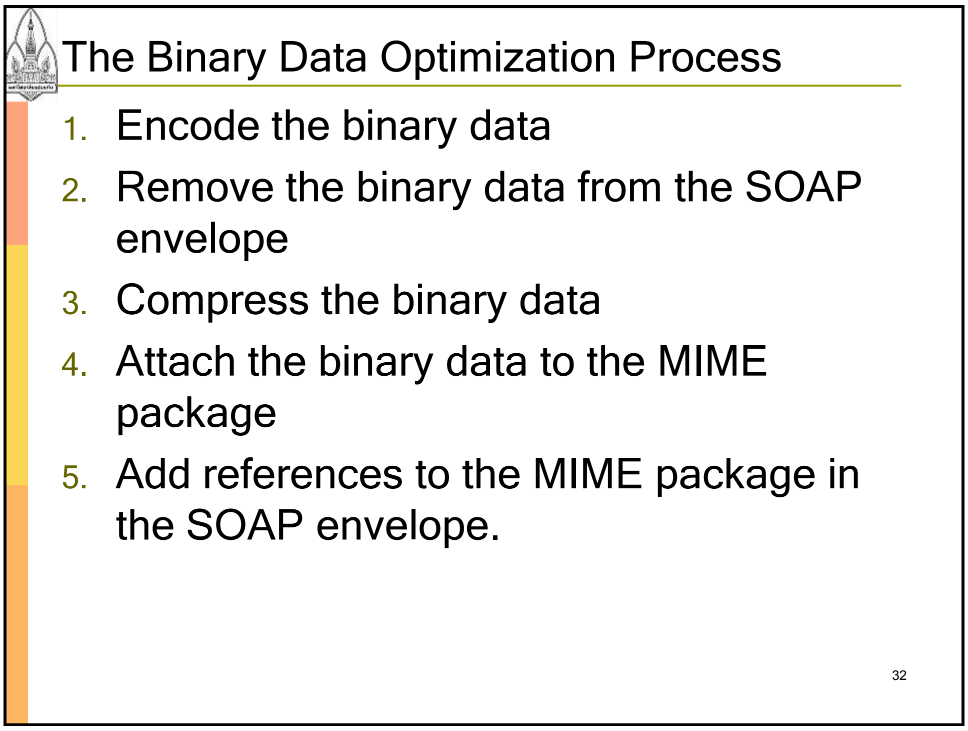
30



Efficient Transmission

- Efficient Transmission
  - Base64Binary encoded data is ~33% larger than raw byte transmission using MIME
- Processing Simplicity
  - Base64Binary encoded data is represented within an element of a SOAP message
  - Security standards such as WS-Signatures and WS-Encryption can directly be applied to the SOAP Message
  - Once such operations are performed, the Base64Binary data can be converted to raw bytes for efficient transmission

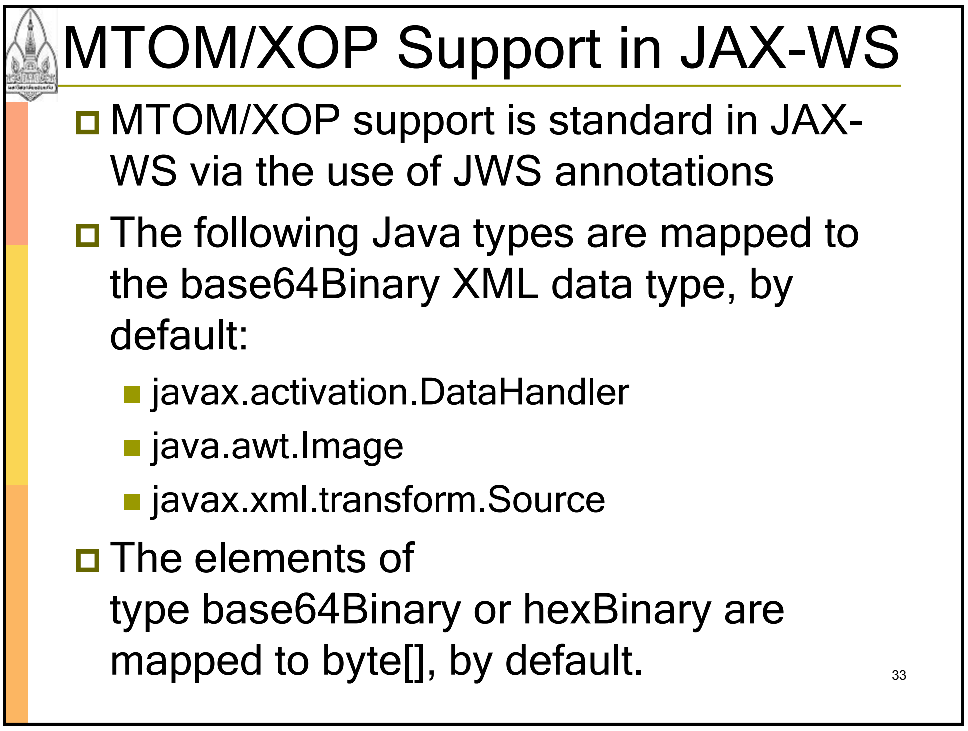
31



The Binary Data Optimization Process

1. Encode the binary data
2. Remove the binary data from the SOAP envelope
3. Compress the binary data
4. Attach the binary data to the MIME package
5. Add references to the MIME package in the SOAP envelope.

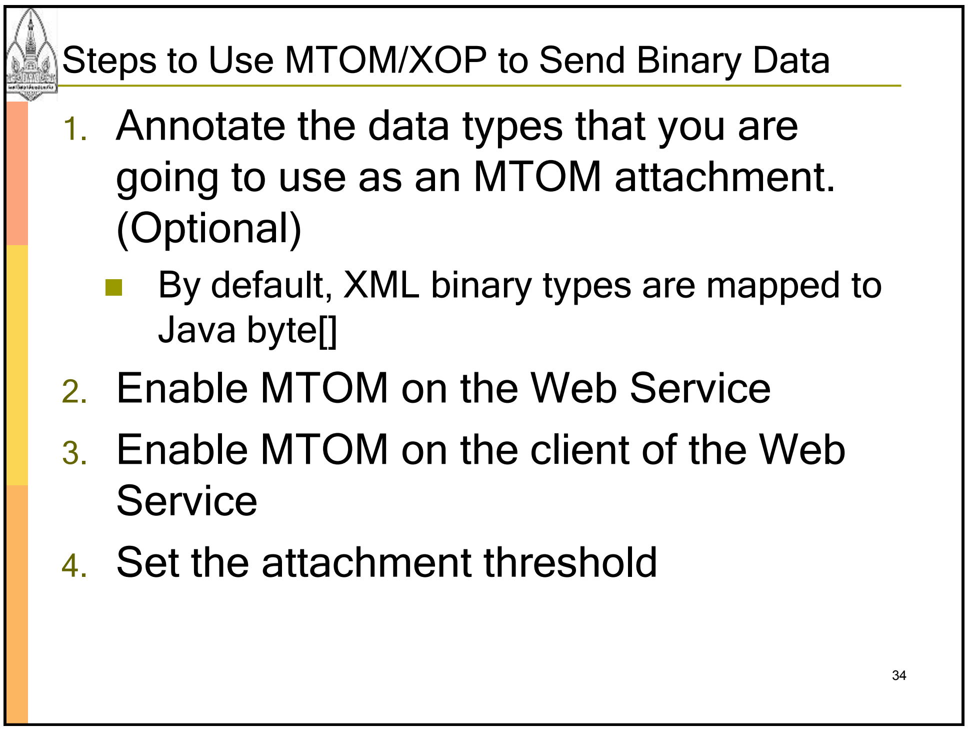
32



MTOM/XOP Support in JAX-WS

- MTOM/XOP support is standard in JAX-WS via the use of JWS annotations
- The following Java types are mapped to the base64Binary XML data type, by default:
  - javax.activation.DataHandler
  - java.awt.Image
  - javax.xml.transform.Source
- The elements of type base64Binary or hexBinary are mapped to byte[], by default.

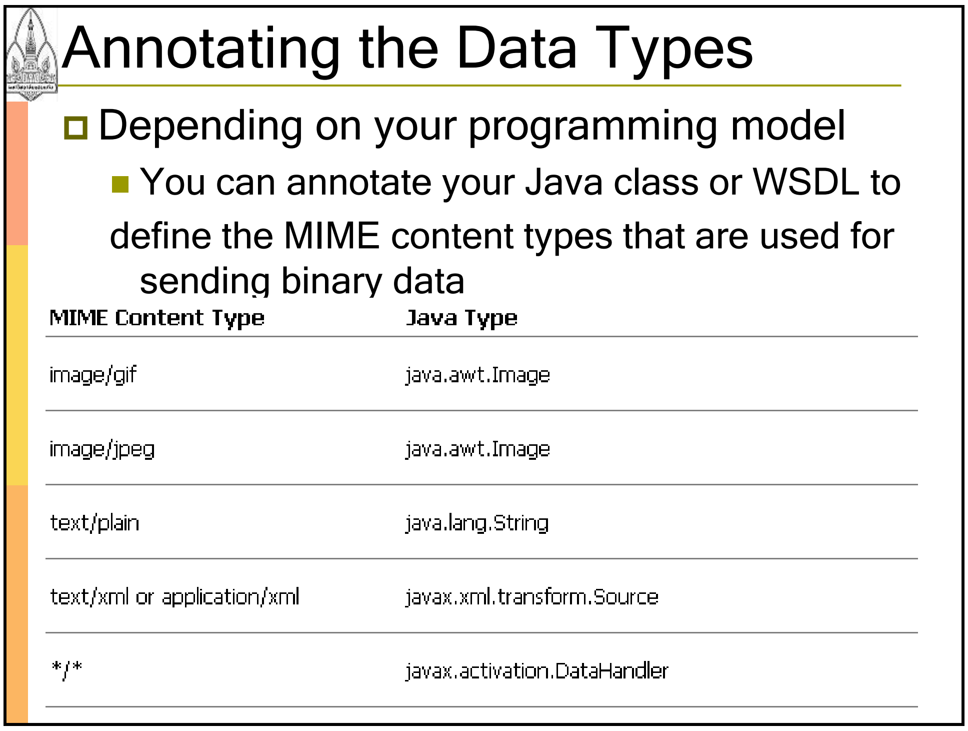
33



Steps to Use MTOM/XOP to Send Binary Data

1. Annotate the data types that you are going to use as an MTOM attachment. (Optional)
   - By default, XML binary types are mapped to Java byte[]
2. Enable MTOM on the Web Service
3. Enable MTOM on the client of the Web Service
4. Set the attachment threshold

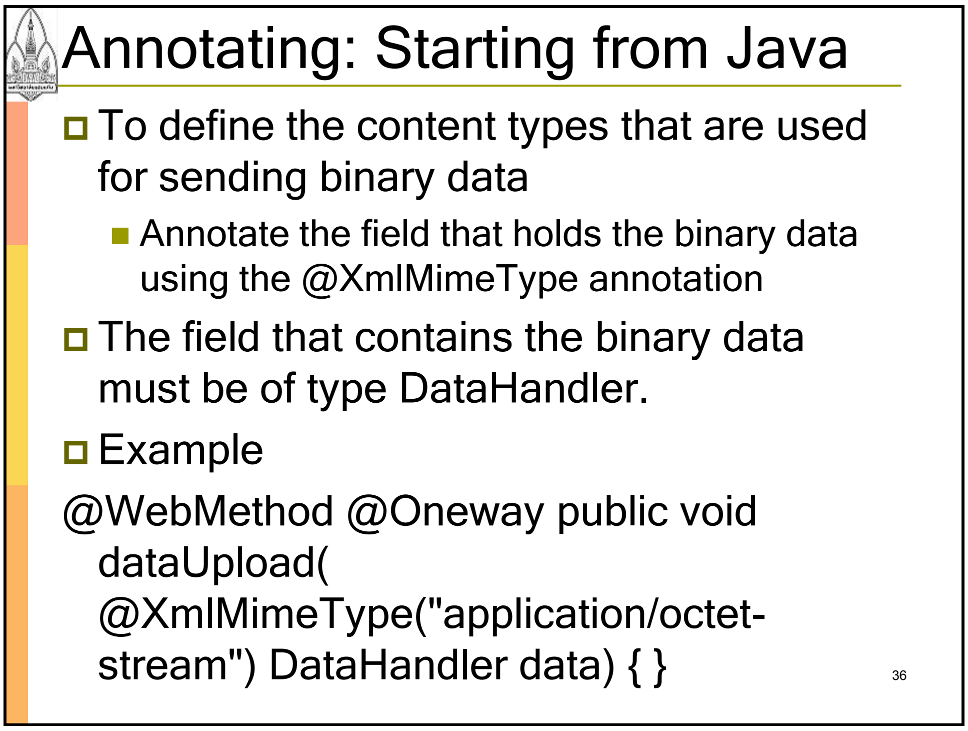
34



Annotating the Data Types

- Depending on your programming model
  - You can annotate your Java class or WSDL to define the MIME content types that are used for sending binary data

MIME Content Type	Java Type
image/gif	java.awt.Image
image/jpeg	java.awt.Image
text/plain	java.lang.String
text/xml or application/xml	javax.xml.transform.Source
/	javax.activation.DataHandler



Annotating: Starting from Java

- To define the content types that are used for sending binary data
  - Annotate the field that holds the binary data using the @XmlMimeType annotation
- The field that contains the binary data must be of type DataHandler.
- Example

```
@WebMethod @Oneway public void
  dataUpload(
  @XmlMimeType("application/octet-
  stream") DataHandler data) { }
```

36

Annotating: Starting from WSDL

- To define the content types that are used for sending binary data, annotate the WSDL element of type xs:base64Binary or xs:hexBinary using one of the following attributes:
  - xmime:contentType - Defines the content type of the element.
  - xmime:expectedContentType - Defines the range of media types that are acceptable for the binary data.

37

Example Annotating from WSDL

- The following example maps the image element of type base64binary to image/gif MIME type (which maps to the java.awt.Image Java type)
- <element name="image" type="base64Binary" xmime:expectedContentTypes="image/gif" xmlns:xmime="http://www.w3.org/2005/05/xmlmime"/>

38

Enabling MTOM on the Web Service

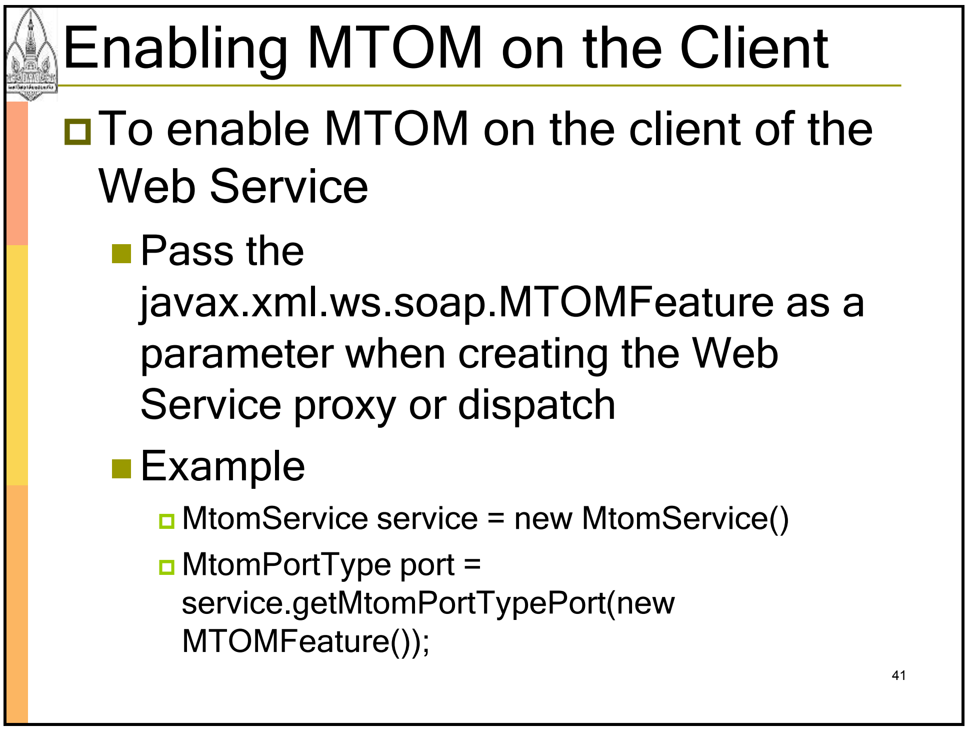
- You can enable MTOM on the Web Service using an annotation or WS-Policy file
  - Enabling MTOM on the Web Service Using Annotation
    - Specify the @java.xml.ws.soap.MTOM annotation on the service endpoint implementation class
  - Enabling MTOM on the Web Services Using WS-Policy File

39

Enabling MTOM Using Annotation

```
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.soap.MTOM;
@MTOM @WebService(name="MtomPortType",
   serviceName="MtomService",
   targetNamespace="http://example.org") public
   class MTOMImpl {
    @WebMethod public String
   echoBinaryAsString(byte[] bytes) {
        return new String(bytes);
   }
}
```

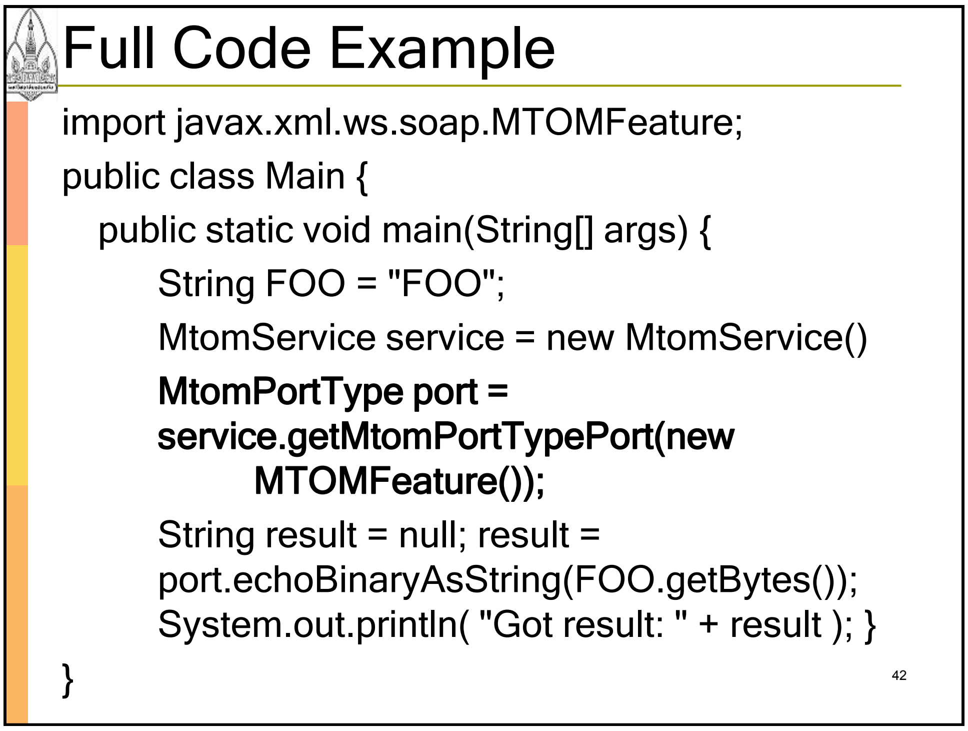
40



Enabling MTOM on the Client

- To enable MTOM on the client of the Web Service
  - Pass the javax.xml.ws.soap.MTOMFeature as a parameter when creating the Web Service proxy or dispatch
  - Example
    - MtomService service = new MtomService()
    - MtomPortType port = service.getMtomPortTypePort(new MTOMFeature());

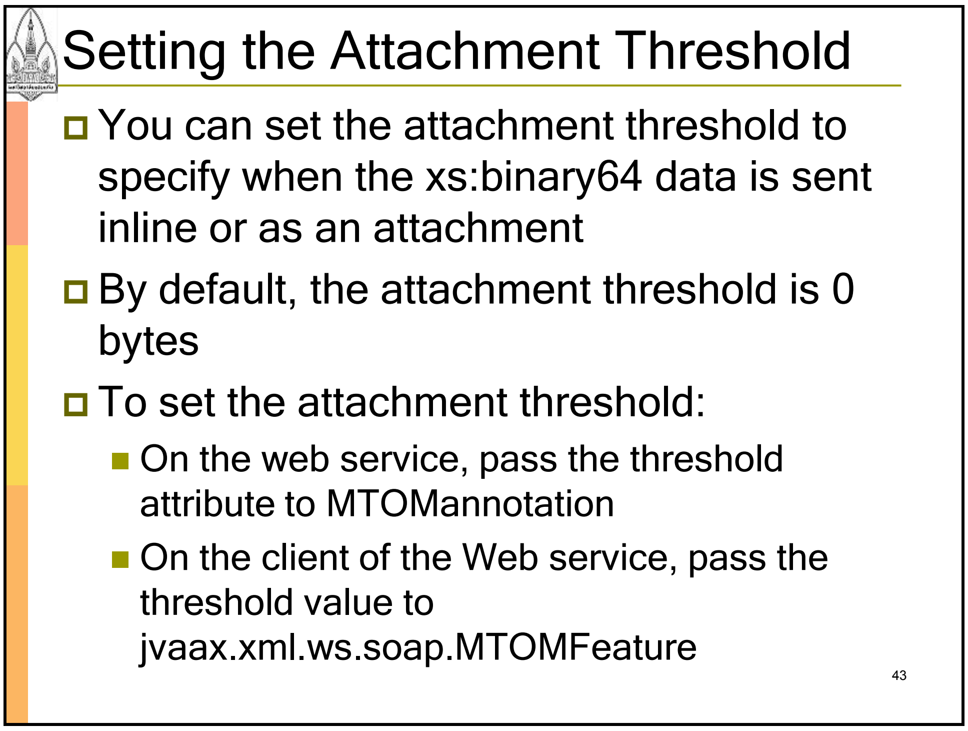
41



Full Code Example

```
import javax.xml.ws.soap.MTOMFeature;
public class Main {
   public static void main(String[] args) {
        String FOO = "FOO";
        MtomService service = new MtomService()
        MtomPortType port =
        service.getMtomPortTypePort(new
                MTOMFeature());
        String result = null; result =
        port.echoBinaryAsString(FOO.getBytes());
        System.out.println( "Got result: " + result ); }
}
```

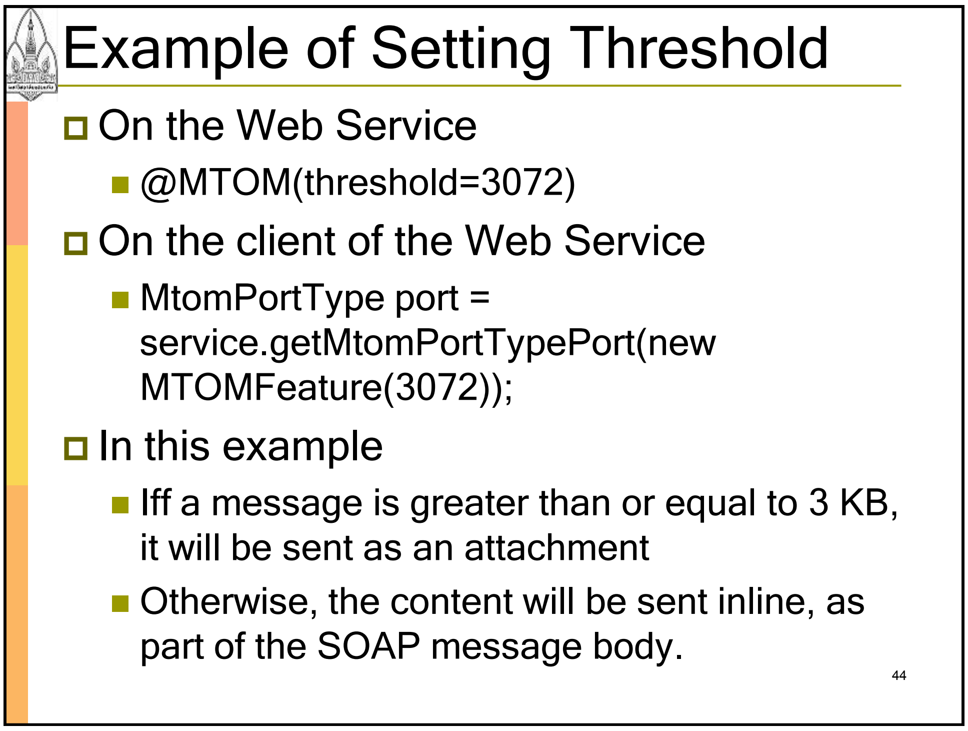
42



Setting the Attachment Threshold

- You can set the attachment threshold to specify when the xs:binary64 data is sent inline or as an attachment
- By default, the attachment threshold is 0 bytes
- To set the attachment threshold:
  - On the web service, pass the threshold attribute to MTOMannotation
  - On the client of the Web service, pass the threshold value to jvaax.xml.ws.soap.MTOMFeature

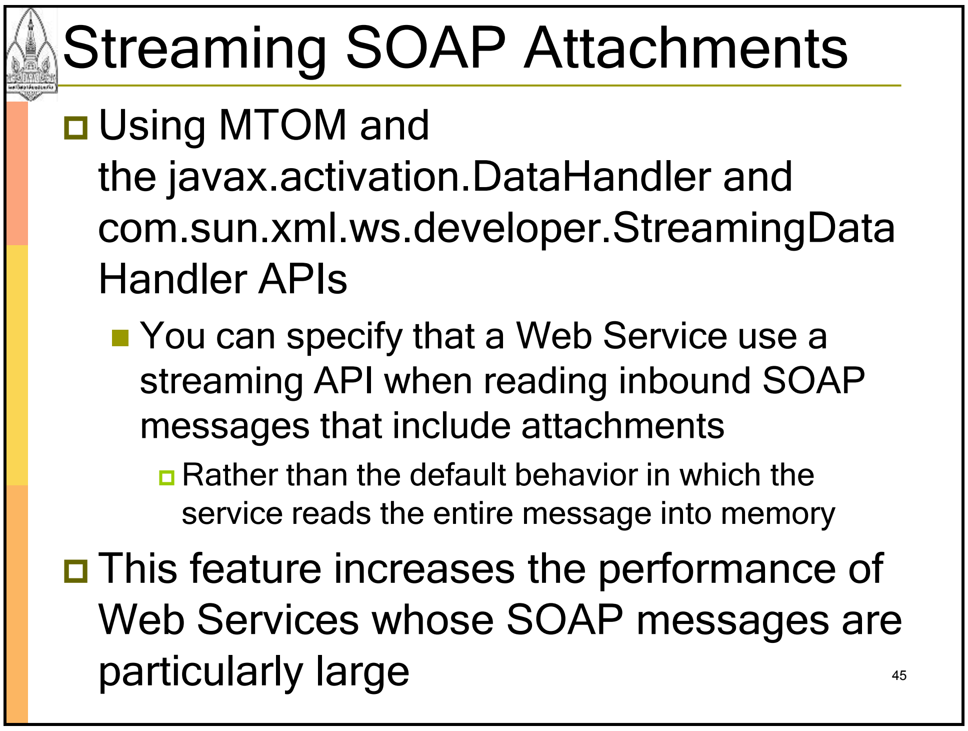
43



Example of Setting Threshold

- On the Web Service
  - @MTOM(threshold=3072)
- On the client of the Web Service
  - MtomPortType port = service.getMtomPortTypePort(new MTOMFeature(3072));
- In this example
  - Iff a message is greater than or equal to 3 KB, it will be sent as an attachment
  - Otherwise, the content will be sent inline, as part of the SOAP message body.

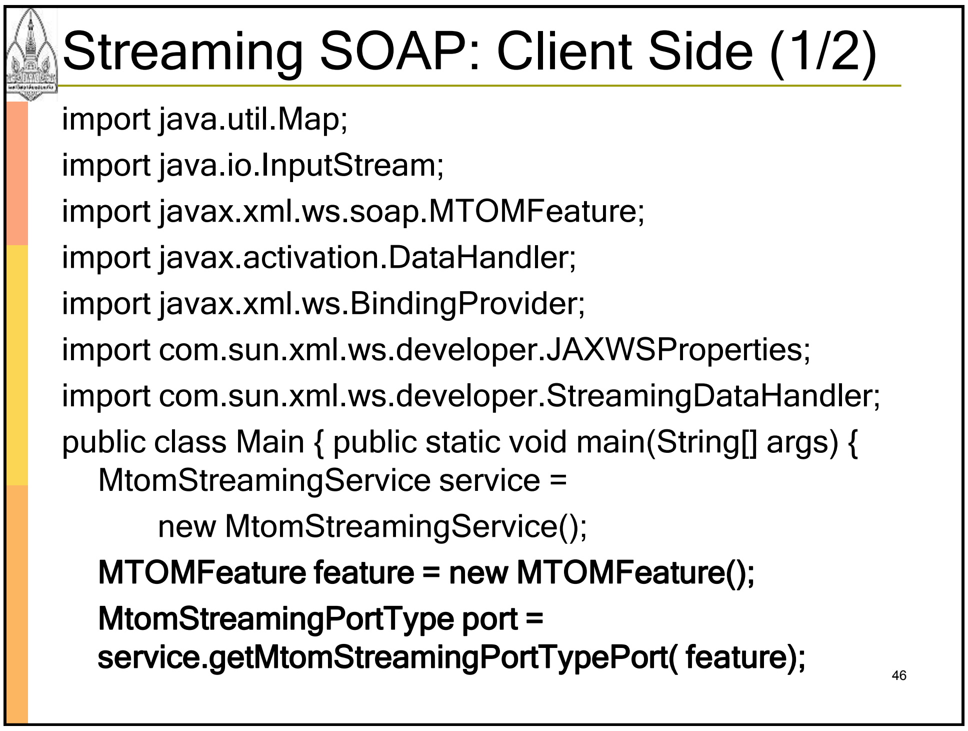
44



Streaming SOAP Attachments

- Using MTOM and the javax.activation.DataHandler and com.sun.xml.ws.developer.StreamingDataHandler APIs
  - You can specify that a Web Service use a streaming API when reading inbound SOAP messages that include attachments
    - Rather than the default behavior in which the service reads the entire message into memory
- This feature increases the performance of Web Services whose SOAP messages are particularly large

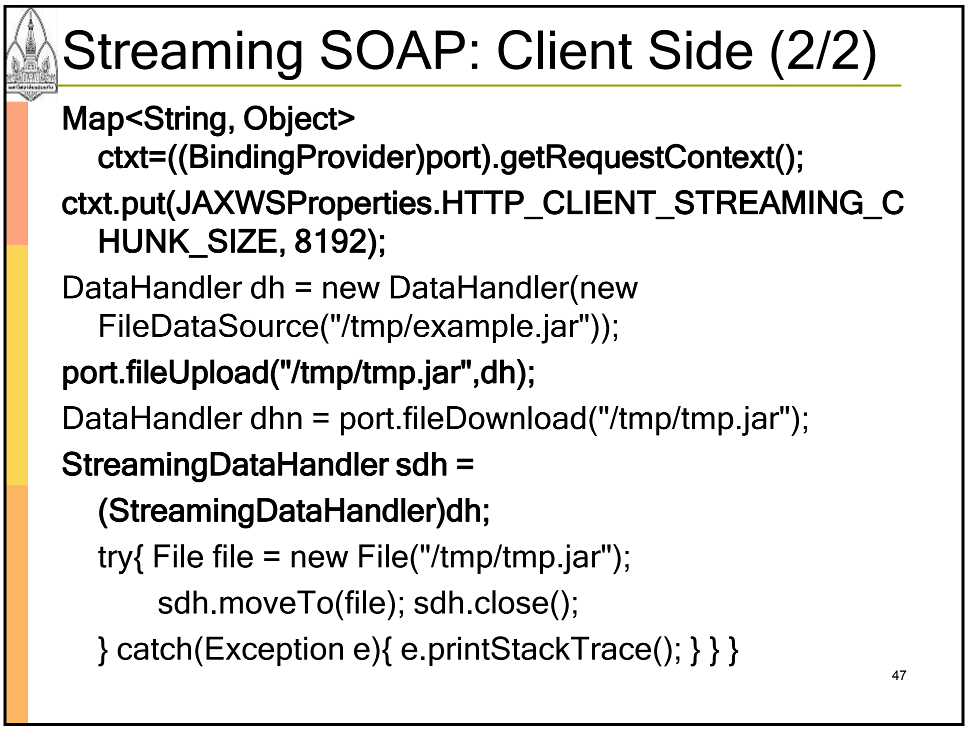
45



Streaming SOAP: Client Side (1/2)

```
import java.util.Map;
import java.io.InputStream;
import javax.xml.ws.soap.MTOMFeature;
import javax.activation.DataHandler;
import javax.xml.ws.BindingProvider;
import com.sun.xml.ws.developer.JAXWSProperties;
import com.sun.xml.ws.developer.StreamingDataHandler;
public class Main { public static void main(String[] args) {
   MtomStreamingService service =
        new MtomStreamingService();
   MTOMFeature feature = new MTOMFeature();
   MtomStreamingPortType port =
   service.getMtomStreamingPortTypePort( feature);
```

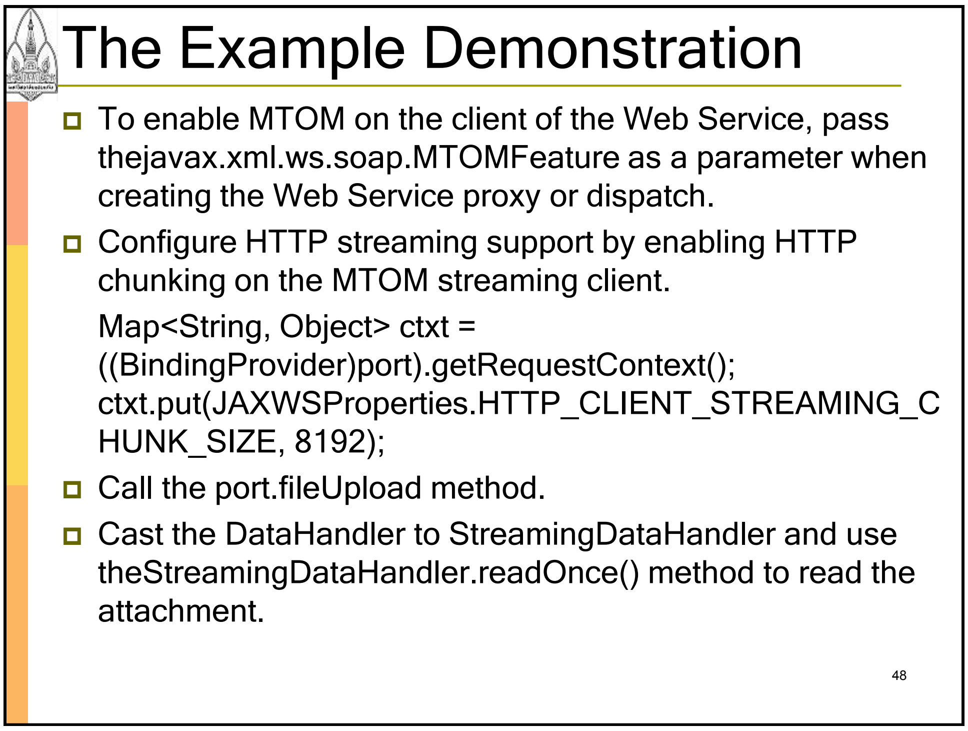
46



Streaming SOAP: Client Side (2/2)

```
Map<String, Object>
   ctxt=((BindingProvider)port).getRequestContext();
ctxt.put(JAXWSProperties.HTTP_CLIENT_STREAMING_C
   HUNK_SIZE, 8192);
DataHandler dh = new DataHandler(new
   FileDataSource("/tmp/example.jar"));
port.fileUpload("/tmp/tmp.jar",dh);
DataHandler dhn = port.fileDownload("/tmp/tmp.jar");
StreamingDataHandler sdh =
   (StreamingDataHandler)dh;
   try{ File file = new File("/tmp/tmp.jar");
        sdh.moveTo(file); sdh.close();
   } catch(Exception e){ e.printStackTrace(); } } }
```

47

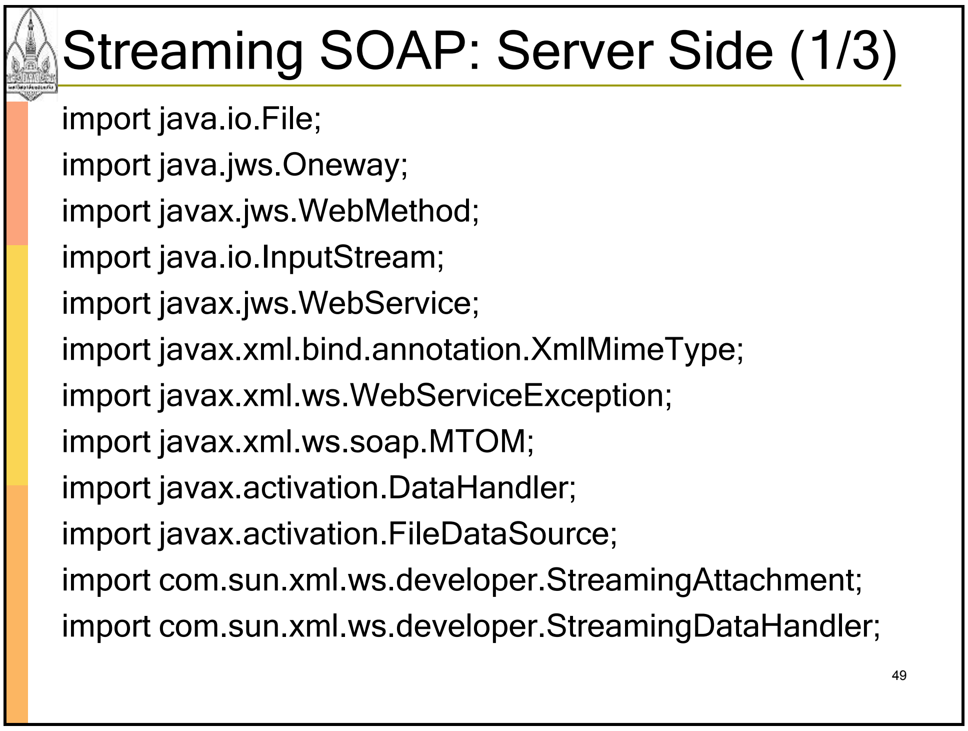


The Example Demonstration

- To enable MTOM on the client of the Web Service, pass thejavax.xml.ws.soap.MTOMFeature as a parameter when creating the Web Service proxy or dispatch.
- Configure HTTP streaming support by enabling HTTP chunking on the MTOM streaming client.

  ```
  Map<String, Object> ctxt =
  ((BindingProvider)port).getRequestContext();
  ctxt.put(JAXWSProperties.HTTP_CLIENT_STREAMING_C
  HUNK_SIZE, 8192);
  ```
- Call the port.fileUpload method.
- Cast the DataHandler to StreamingDataHandler and use theStreamingDataHandler.readOnce() method to read the attachment.

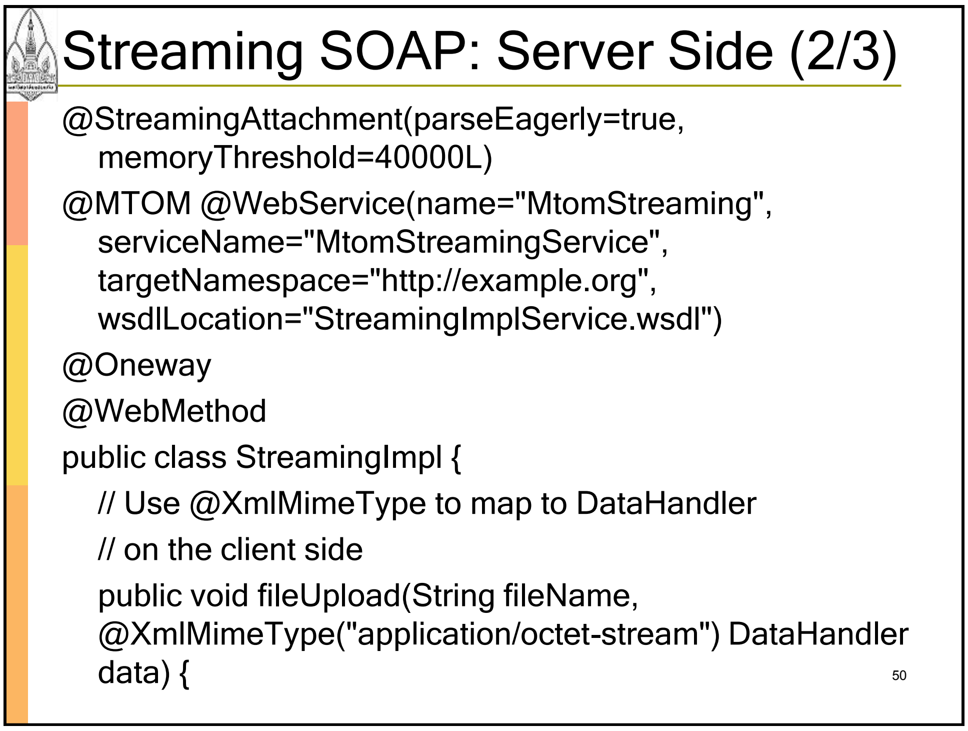
48



Streaming SOAP: Server Side (1/3)

```
import java.io.File;
import java.jws.Oneway;
import javax.jws.WebMethod;
import java.io.InputStream;
import javax.jws.WebService;
import javax.xml.bind.annotation.XmlMimeType;
import javax.xml.ws.WebServiceException;
import javax.xml.ws.soap.MTOM;
import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import com.sun.xml.ws.developer.StreamingAttachment;
import com.sun.xml.ws.developer.StreamingDataHandler;
```

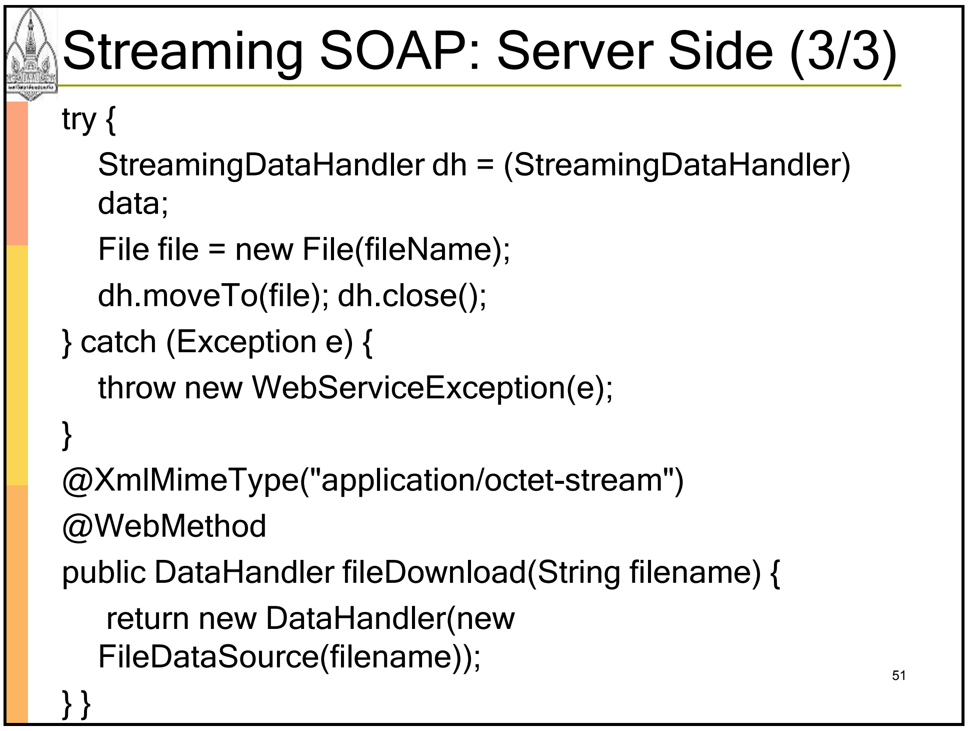
49



Streaming SOAP: Server Side (2/3)

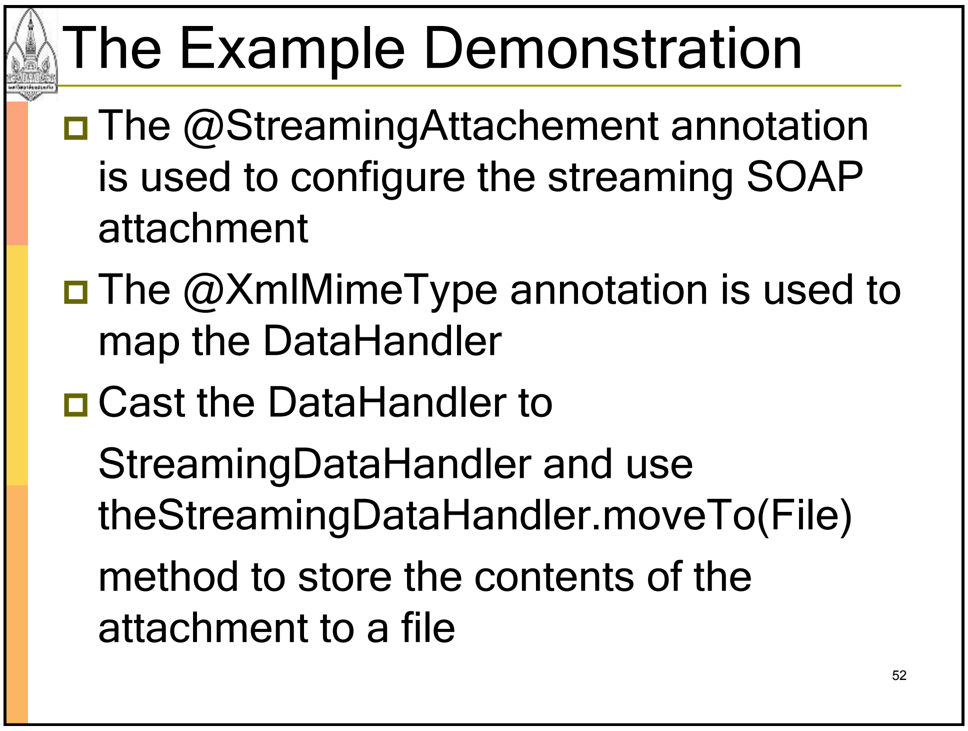
```
@StreamingAttachment(parseEagerly=true,
   memoryThreshold=40000L)
@MTOM @WebService(name="MtomStreaming",
   serviceName="MtomStreamingService",
   targetNamespace="http://example.org",
   wsdlLocation="StreamingImplService.wsdl")
@Oneway
@WebMethod
public class StreamingImpl {
   // Use @XmlMimeType to map to DataHandler
   // on the client side
   public void fileUpload(String fileName,
   @XmlMimeType("application/octet-stream") DataHandler
   data) {
```

50



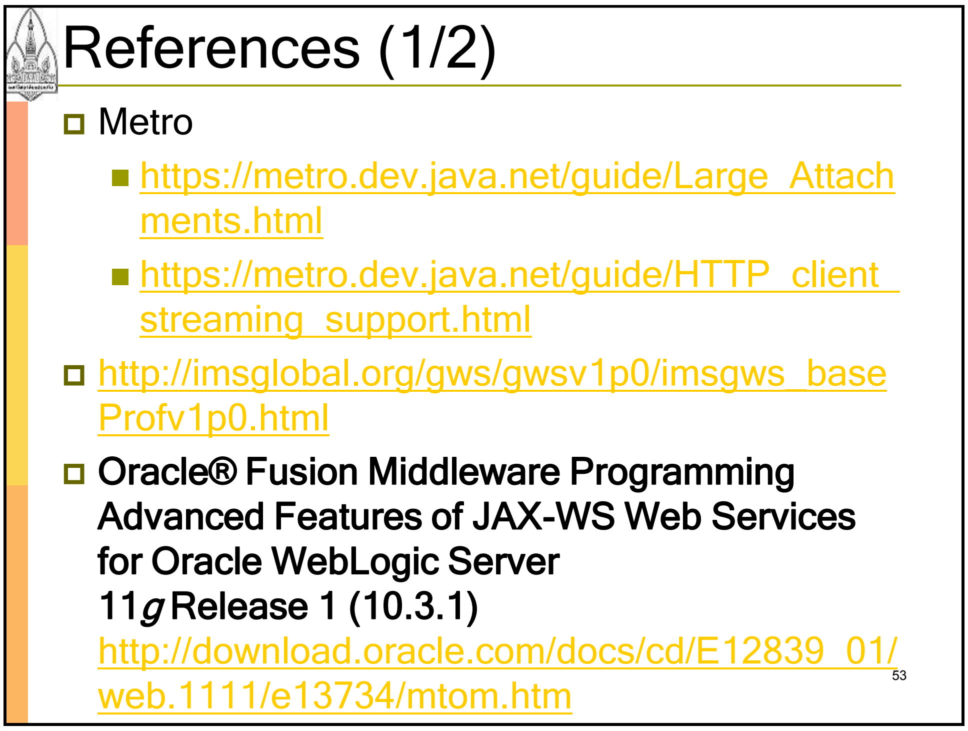
Streaming SOAP: Server Side (3/3)

```
try {
   StreamingDataHandler dh = (StreamingDataHandler)
   data;
   File file = new File(fileName);
   dh.moveTo(file); dh.close();
} catch (Exception e) {
   throw new WebServiceException(e);
}
@XmlMimeType("application/octet-stream")
@WebMethod
public DataHandler fileDownload(String filename) {
    return new DataHandler(new
   FileDataSource(filename));
} }
```



The Example Demonstration

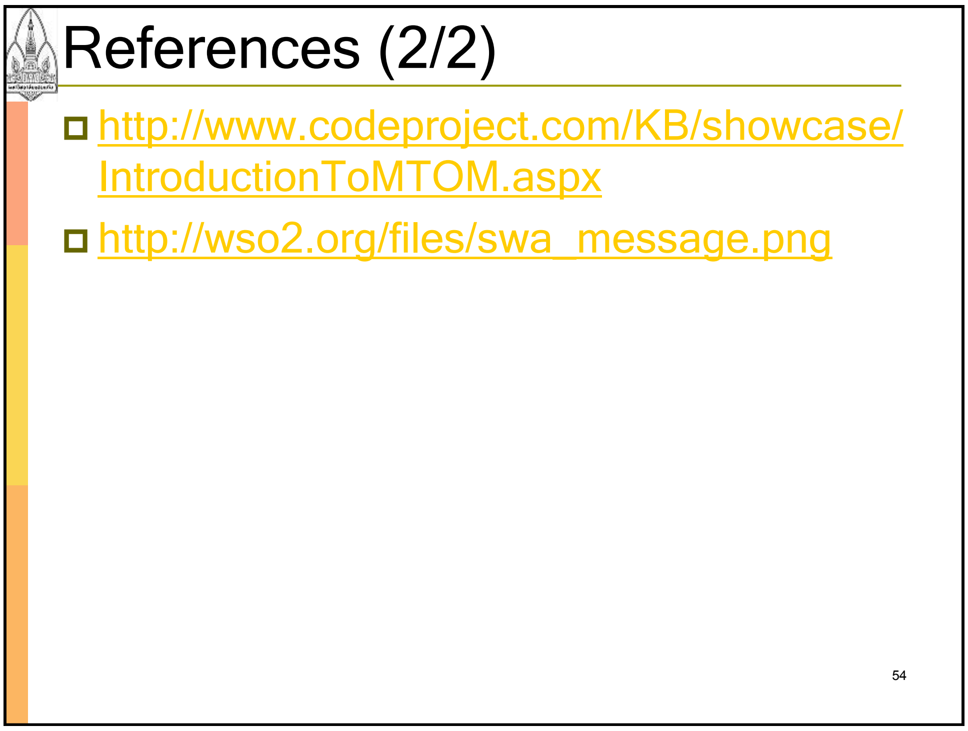
- The @StreamingAttachement annotation is used to configure the streaming SOAP attachment
- The @XmlMimeType annotation is used to map the DataHandler
- Cast the DataHandler to StreamingDataHandler and use theStreamingDataHandler.moveTo(File) method to store the contents of the attachment to a file



References (1/2)

- Metro
  - https://metro.dev.java.net/guide/Large_Attachments.html
  - https://metro.dev.java.net/guide/HTTP_client_streaming_support.html
- http://imsglobal.org/gws/gwsv1p0/imsgws_baseProfv1p0.html
- **Oracle® Fusion Middleware Programming Advanced Features of JAX-WS Web Services for Oracle WebLogic Server 11*g* Release 1 (10.3.1)** http://download.oracle.com/docs/cd/E12839_01/web.1111/e13734/mtom.htm

53



References (2/2)

- http://www.codeproject.com/KB/showcase/IntroductionToMTOM.aspx
- http://wso2.org/files/swa_message.png

54