# Java API for XML-based Remote Procedure Call (JAX-RPC)

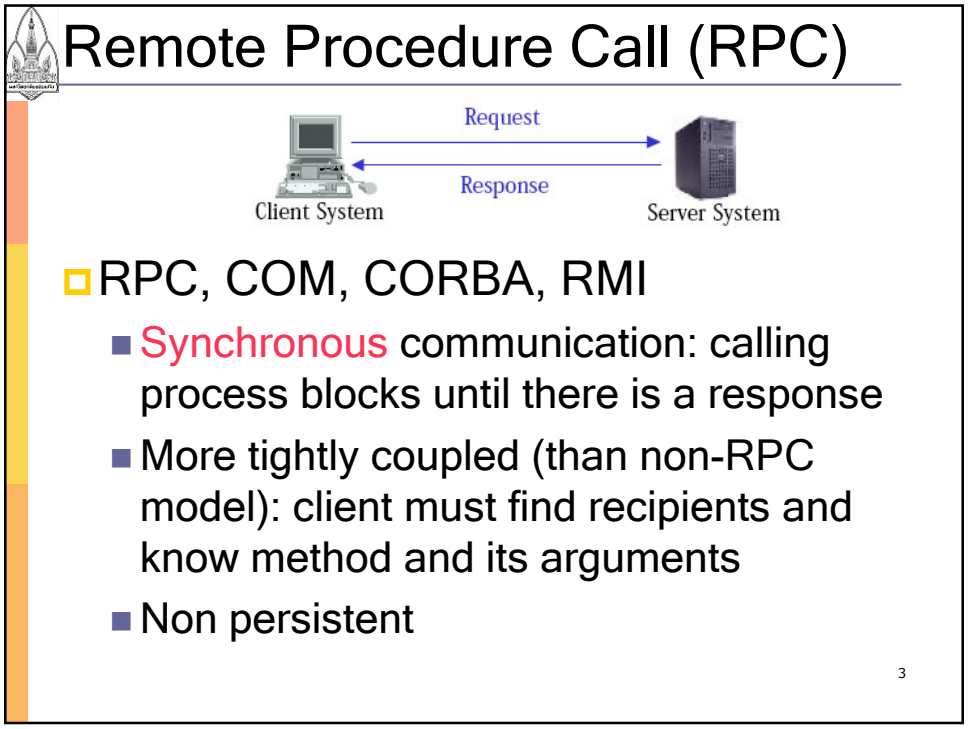
Dr. Kanda Runapongsa
(krunapon@kku.ac.th)
Department of Computer Engineering
Khon Kaen University

1

## Agenda

- Background on remote communication
- What is and Why JAX-RPC?
- Development steps of a JAX-RPC Service
- Type Mapping
- Client Programming
- Service Endpoint Model

2

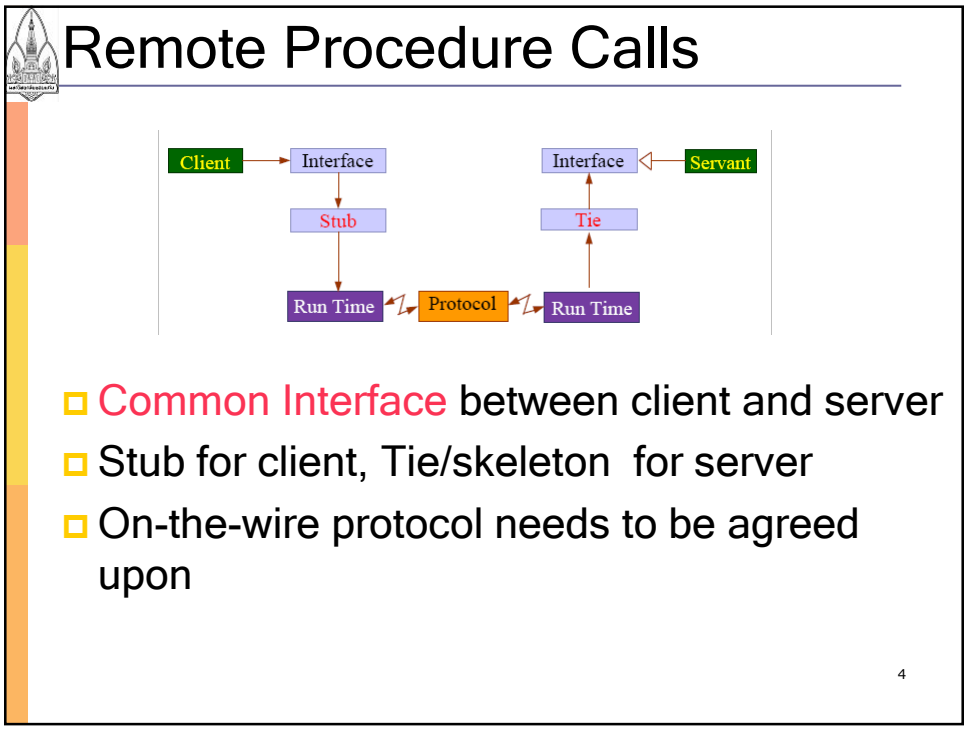


# Remote Procedure Call (RPC)




- RPC, COM, CORBA, RMI
  - Synchronous communication: calling process blocks until there is a response
  - More tightly coupled (than non-RPC model): client must find recipients and know method and its arguments
  - Non persistent

3

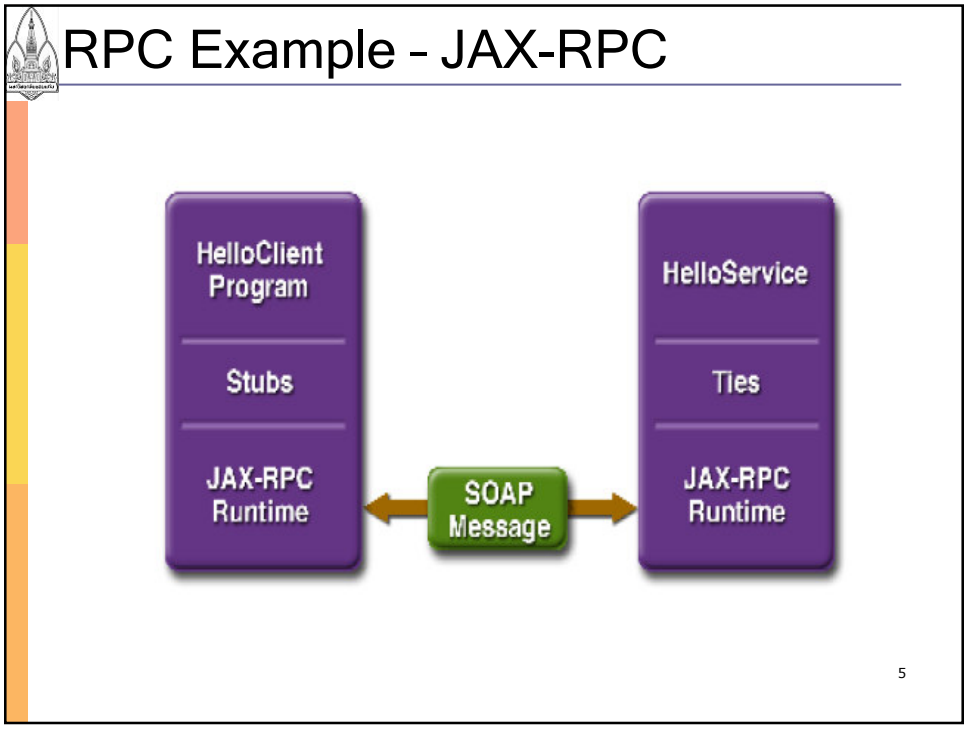


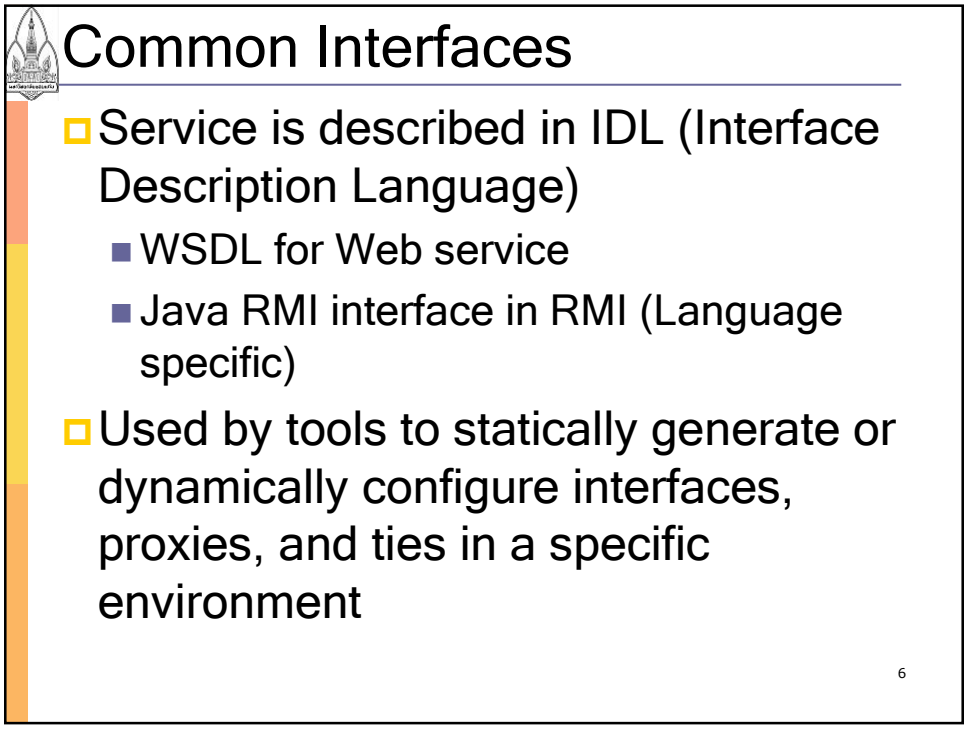
# Remote Procedure Calls

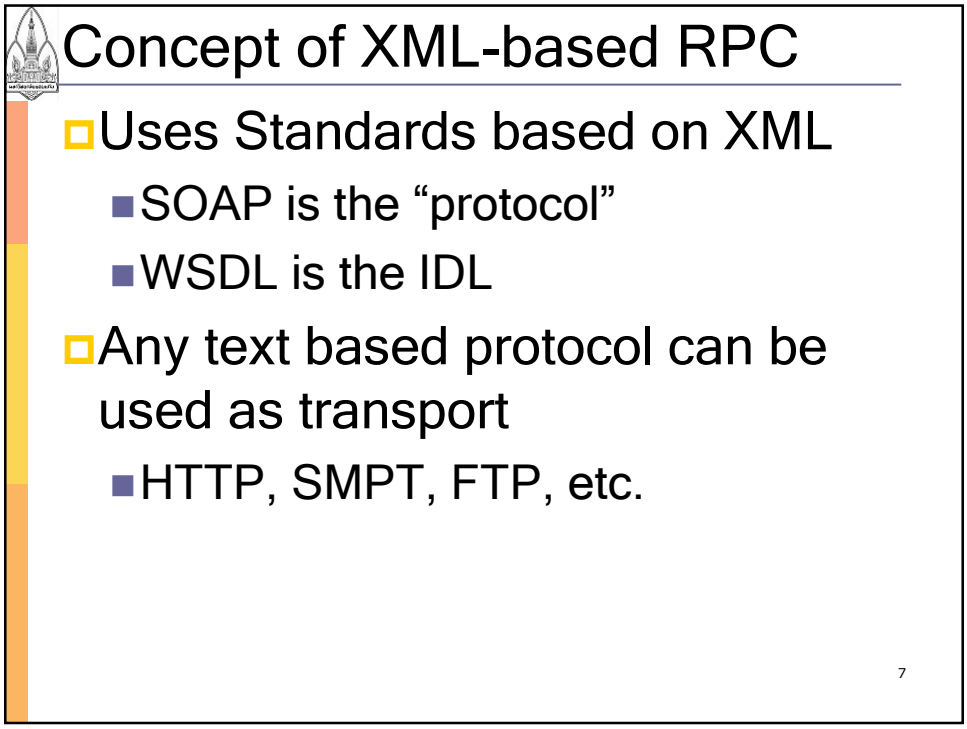


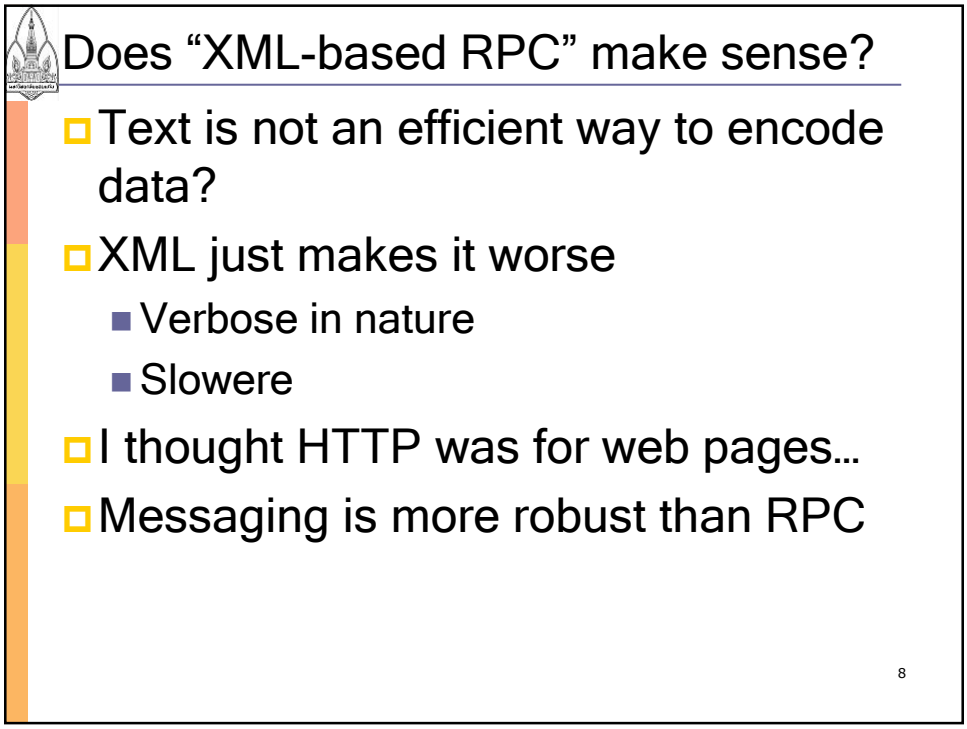

- Common Interface between client and server
- Stub for client, Tie/skeleton for server
- On-the-wire protocol needs to be agreed upon

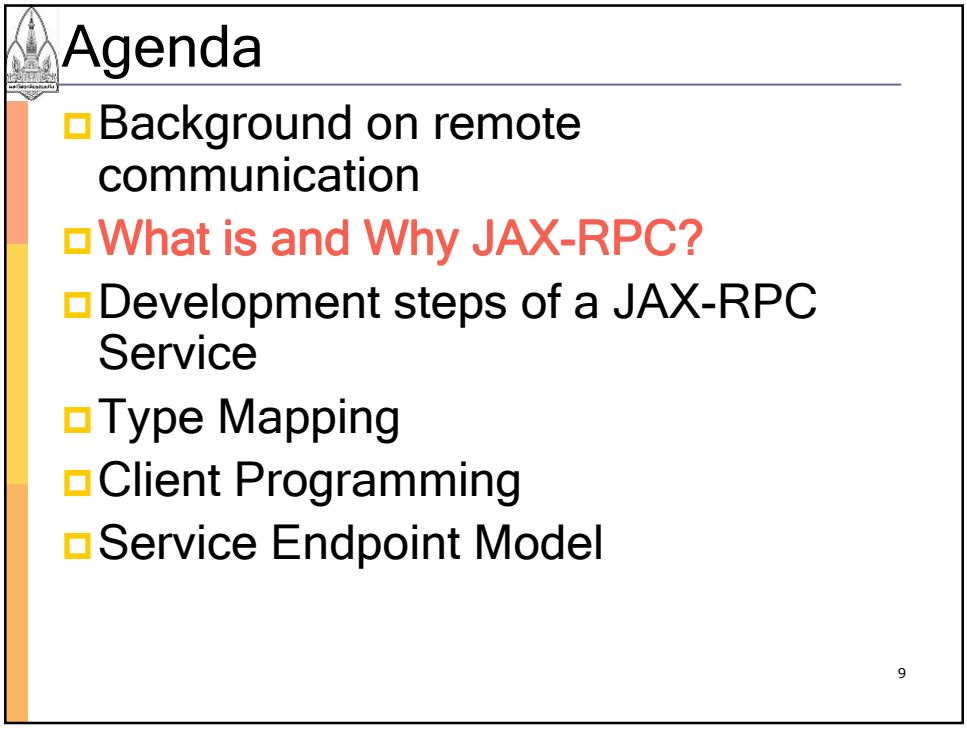
4

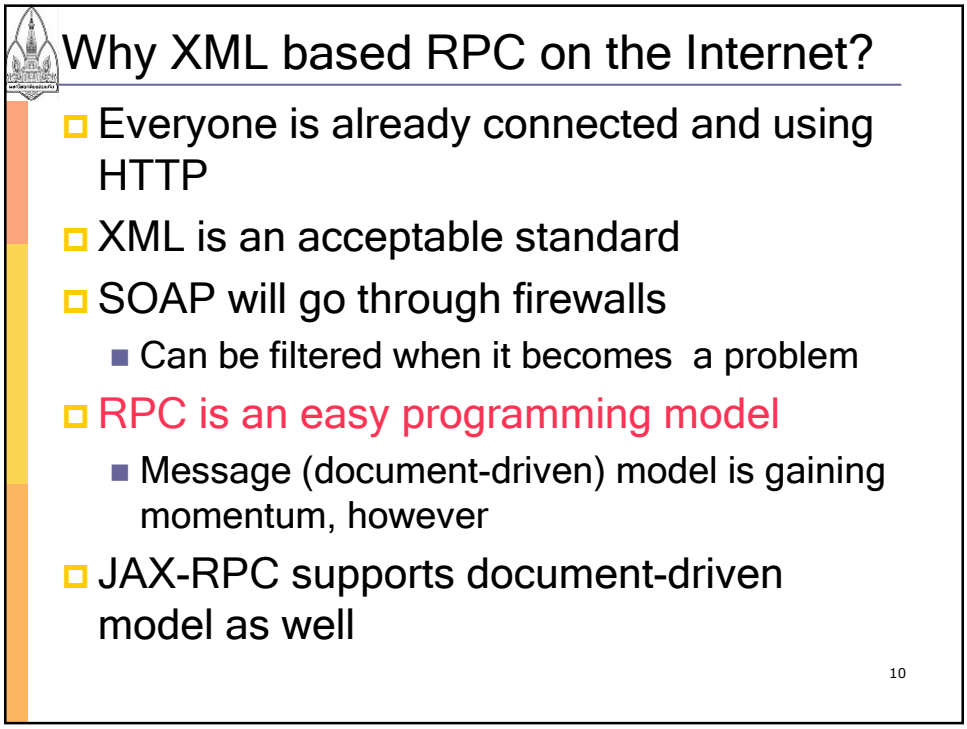

RPC Example – JAX-RPC
HelloClient Program
Stubs
JAX-RPC Runtime
SOAP Message
HelloService
Ties
JAX-RPC Runtime
5


Common Interfaces
Service is described in IDL (Interface Description Language)
WSDL for Web service
Java RMI interface in RMI (Language specific)
Used by tools to statically generate or dynamically configure interfaces, proxies, and ties in a specific environment
6


Concept of XML-based RPC
Uses Standards based on XML
SOAP is the “protocol”
WSDL is the IDL
Any text based protocol can be used as transport
HTTP, SMPT, FTP, etc.
7


Does “XML-based RPC” make sense?
Text is not an efficient way to encode data?
XML just makes it worse
Verbose in nature
Slowere
I thought HTTP was for web pages…
Messaging is more robust than RPC
8


Agenda
Background on remote communication
What is and Why JAX-RPC?
Development steps of a JAX-RPC Service
Type Mapping
Client Programming
Service Endpoint Model
9


Why XML based RPC on the Internet?
Everyone is already connected and using HTTP
XML is an acceptable standard
SOAP will go through firewalls
Can be filtered when it becomes a problem
RPC is an easy programming model
Message (document-driven) model is gaining momentum, however
JAX-RPC supports document-driven model as well
10

# What is JAX-RPC?

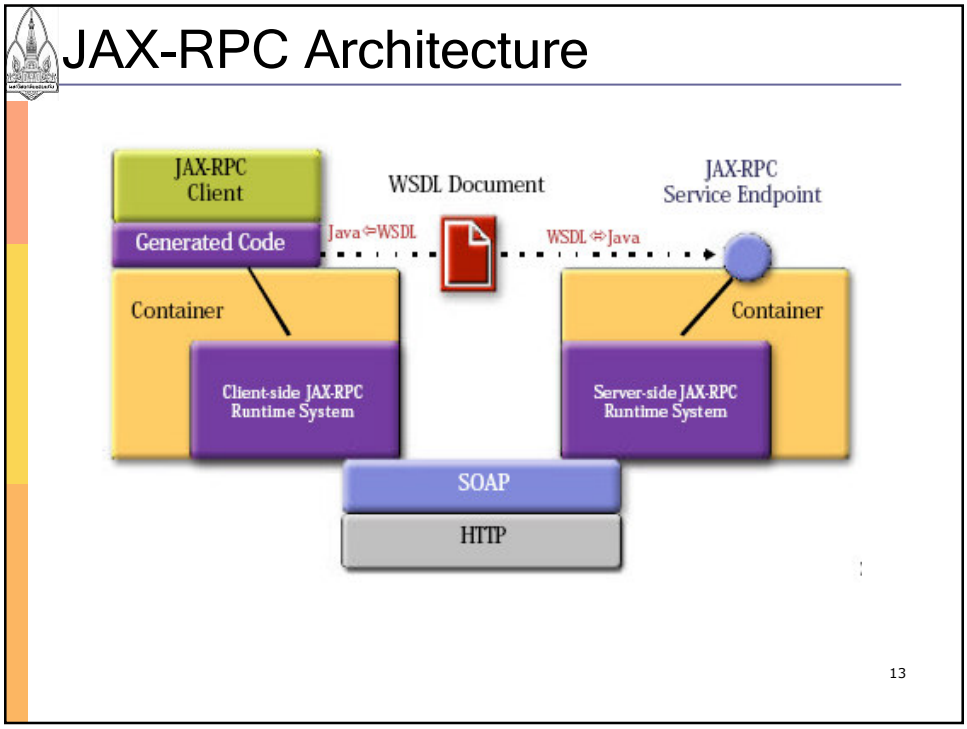
- Java API for XML-based RPC
  - Web services operations are performed by exchanging SOAP 1.1 messages
- Services are described using WSDL
  - WSDL is the contract between service provider and client
- Web service endpoints and clients use JAX-RPC programming model
- Key technology for Web Services in the upcoming J2EE 1.4 platform

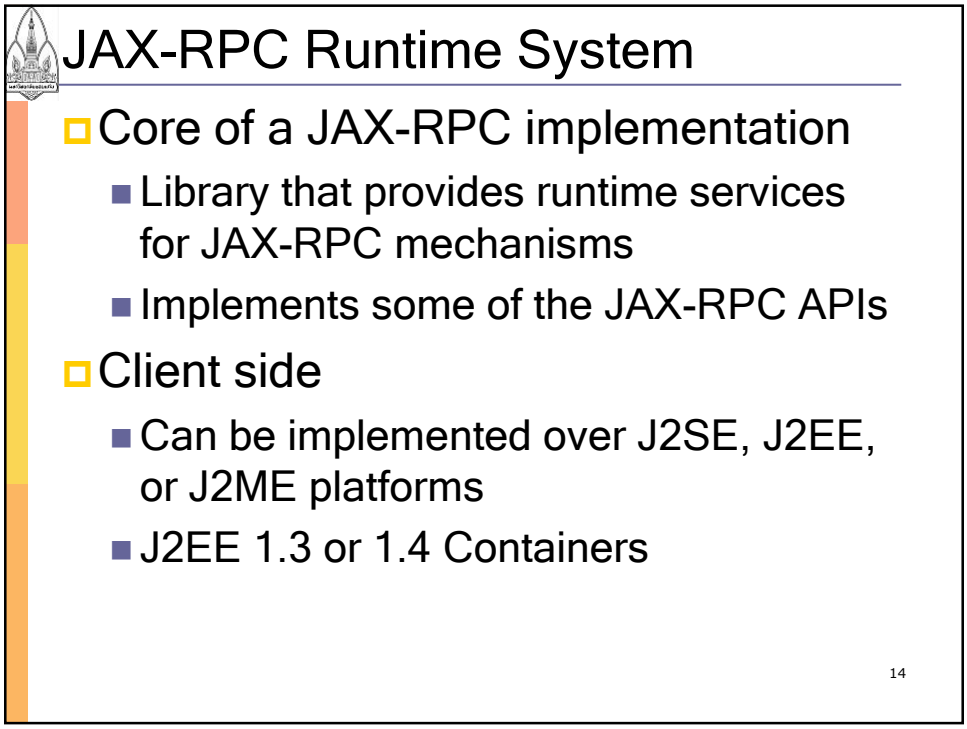
11

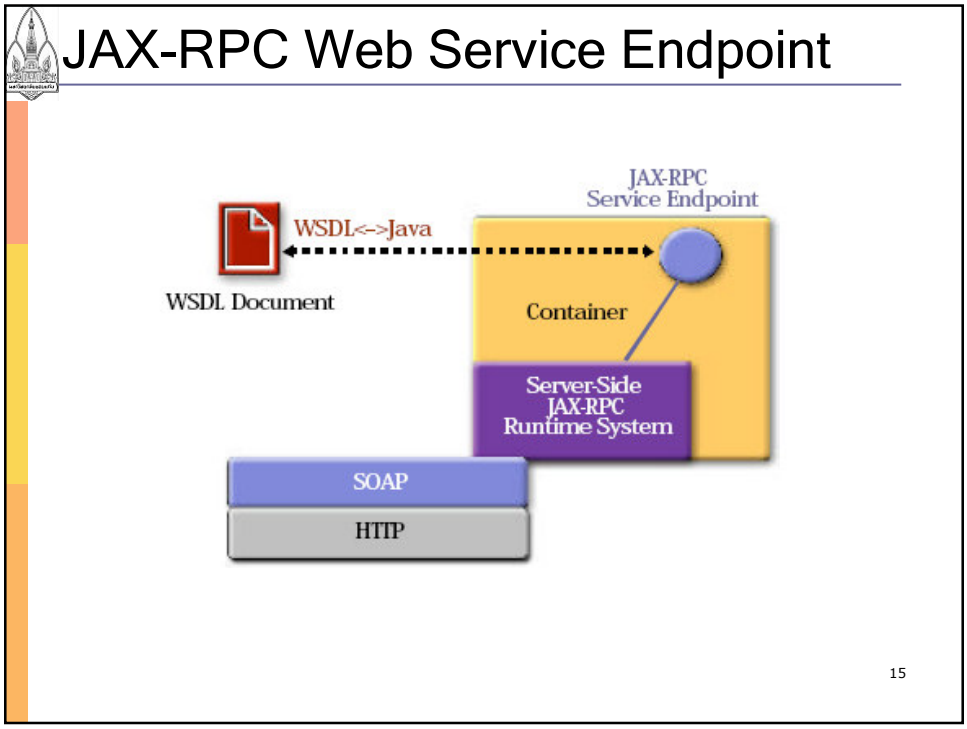
# JAX-RPC Design Goals

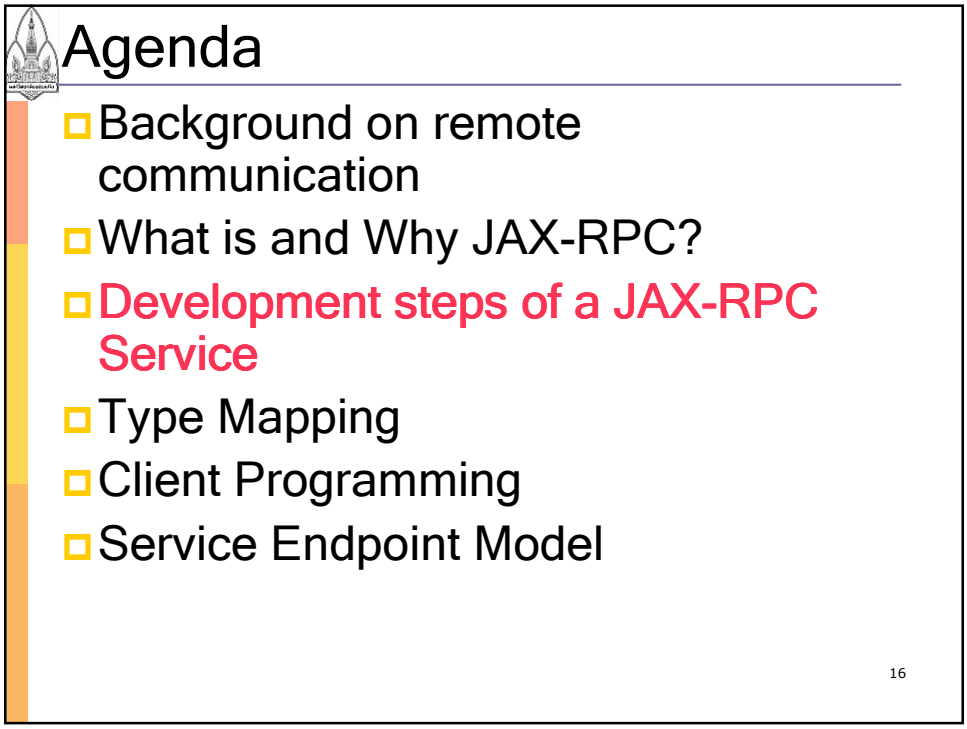
- Easy to use programming model
  - For both defining & using a service
- Hides all the plumbing
  - You don't have to create SOAP messages yourself
- SOAP and WSDL-based interoperability
  - Interoperate with any SOAP 1.1 compliant peers
- Extensibility and Modularity
  - Support future versions of XML specification

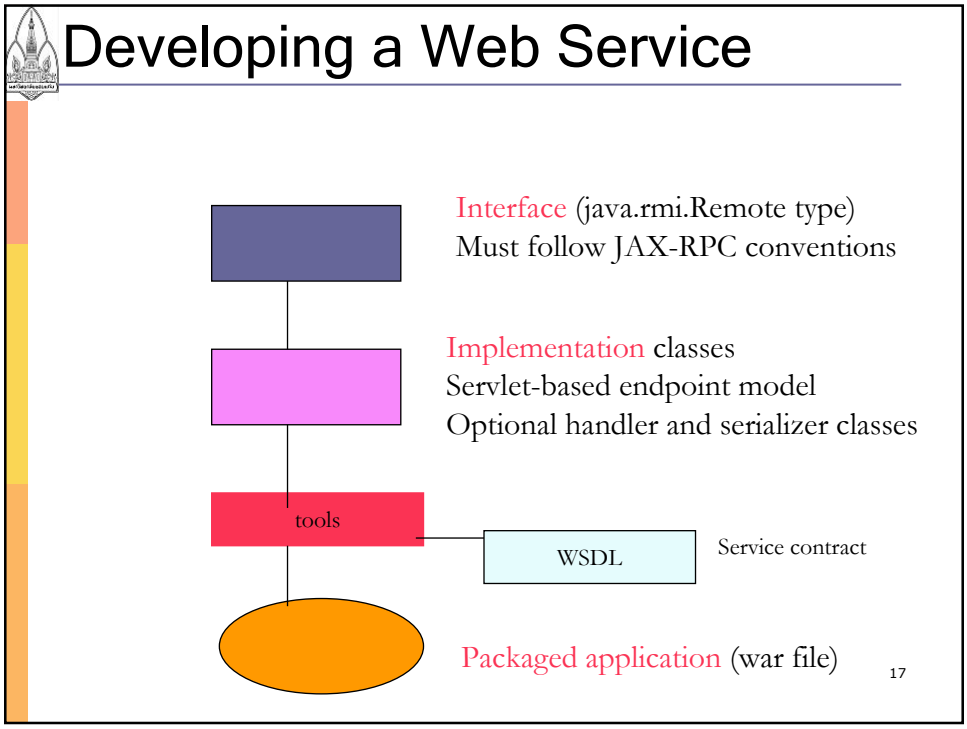
12

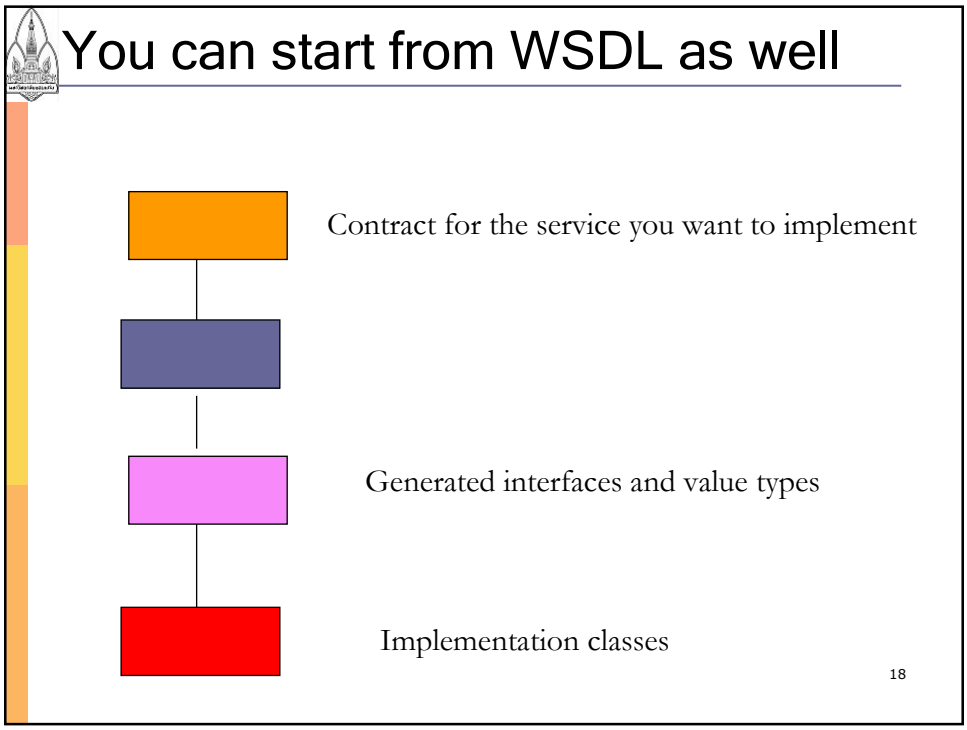

JAX-RPC Architecture
JAX-RPC Client
Generated Code
WSDL Document
JAX-RPC Service Endpoint
Java⇨WSDL
WSDL⇨Java
Container
Container
Client-side JAX-RPC Runtime System
Server-side JAX-RPC Runtime System
SOAP
HTTP
13


JAX-RPC Runtime System
Core of a JAX-RPC implementation
Library that provides runtime services for JAX-RPC mechanisms
Implements some of the JAX-RPC APIs
Client side
Can be implemented over J2SE, J2EE, or J2ME platforms
J2EE 1.3 or 1.4 Containers
14


JAX-RPC Web Service Endpoint
JAX-RPC
Service Endpoint
WSDL<->Java
WSDL Document
Container
Server-Side
JAX-RPC
Runtime System
SOAP
HTTP
15


Agenda
Background on remote communication
What is and Why JAX-RPC?
Development steps of a JAX-RPC Service
Type Mapping
Client Programming
Service Endpoint Model
16


Developing a Web Service
Interface (java.rmi.Remote type)
Must follow JAX-RPC conventions
Implementation classes
Servlet-based endpoint model
Optional handler and serializer classes
tools
WSDL
Service contract
Packaged application (war file)
17


You can start from WSDL as well
Contract for the service you want to implement
Generated interfaces and value types
Implementation classes
18

## Steps for Developing a JAX-RPC Web Service

1. Code the Service Endpoint Interface (SEI) and implementation class and interface configuration file
2. Compile the SEI and implementation class
3. Use wscompile utility program to generate the WSDL and other files required to deploy the service
4. Package the files into a WAR file
5. Deploy the WAR file

19

## 1.a. Code Service Endpoint Interface

- Declares the methods that a remote client may invoke on the service
- Rules
  - It extends the java.rmi.Remote interface
  - It must not have constant declarations, such as public final static
  - The methods must throw the java.rmi.RemoteException or one of its subclasses
  - Method parameters and return types must be supported JAX-RPC types

20

## Example: Service Definition Interface (HelloIF.java from "helloservice")

```
package helloservice;

 import java.rmi.Remote;
import java.rmi.RemoteException;

public interface HelloIF extends
   Remote {
   public String sayHello(String s)
   throws RemoteException;
}
```

21

## 1.B Code Service Implementation

- Service implementation class is an ordinary Java class (for servlet-based Web service endpoint) – helloservice example
- Service implementation class is a stateless session bean (for Stateless session bean based Web service endpoint)

22

## Example: Service Implementation (HelloImp.java from “helloservice”)

```
package helloservice;

public class HelloImpl implements
  HelloIF {
  public String sayHello(String s) {
      return message + s;
  }
}
```

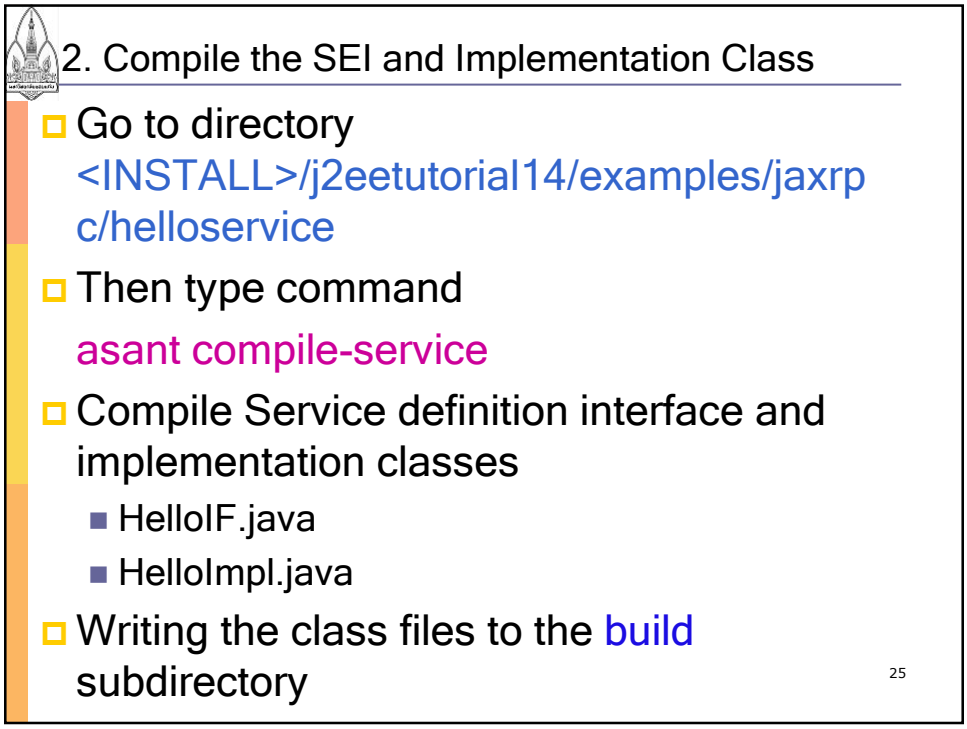
23

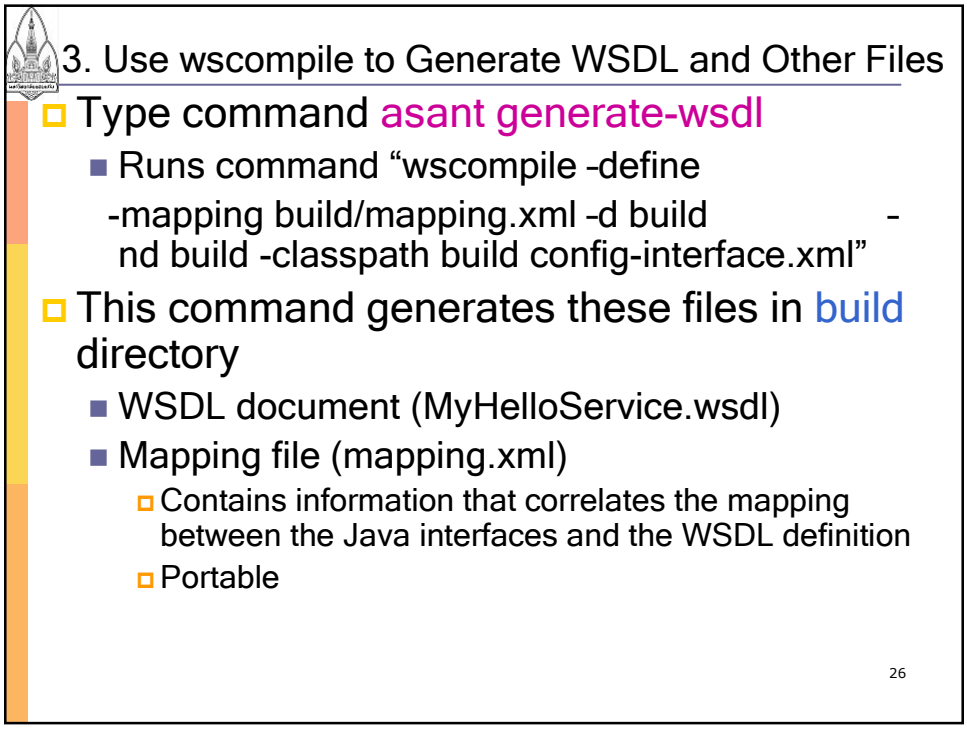
## 1.C Interface Configuration File

- Specifies information about the SEI
- Used by wscompile to generate WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
 xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
 <service
    name="MyHelloService"
    targetNamespace="urn:Foo"
    typeNamespace="urn:Foo"
    packageName="helloservice">
    <interface name="helloservice.HelloIF"/>
 </service>
</configuration>
```

24


2. Compile the SEI and Implementation Class
Go to directory <INSTALL>/j2eetutorial14/examples/jaxrpc/helloservice
Then type command
asant compile-service
Compile Service definition interface and implementation classes
HelloIF.java
HelloImpl.java
Writing the class files to the build subdirectory
25


3. Use wscompile to Generate WSDL and Other Files
Type command asant generate-wsdl
Runs command "wscompile -define -mapping build/mapping.xml -d build -nd build -classpath build config-interface.xml"
This command generates these files in build directory
WSDL document (MyHelloService.wsdl)
Mapping file (mapping.xml)
Contains information that correlates the mapping between the Java interfaces and the WSDL definition
Portable
26

## 4. Package the Files into WAR File

- To package the files into WAR file, run command asant create-war
- This command generates hello-jaxrpc.war in directory <INSTALL>/j2eetutorial14/examples/jaxrpc/helloservice

27

## 5. Deploy the WAR File

- Make sure the Application Server is started
- Run asant deploy-war
  - hello-jaxrpc.war is deployed
- The tie classes (which are used to communicate with clients) are generated by the Application server during deployment

28

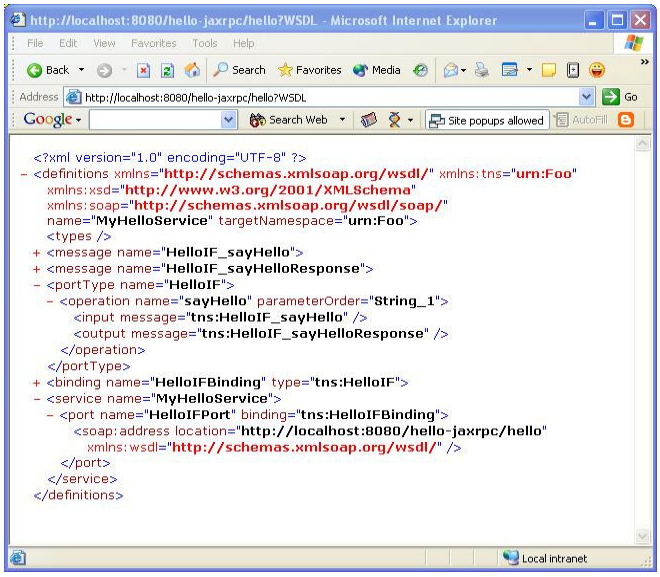
# Verify Service

- Deploy as a Web application
- Verify the service from a browser

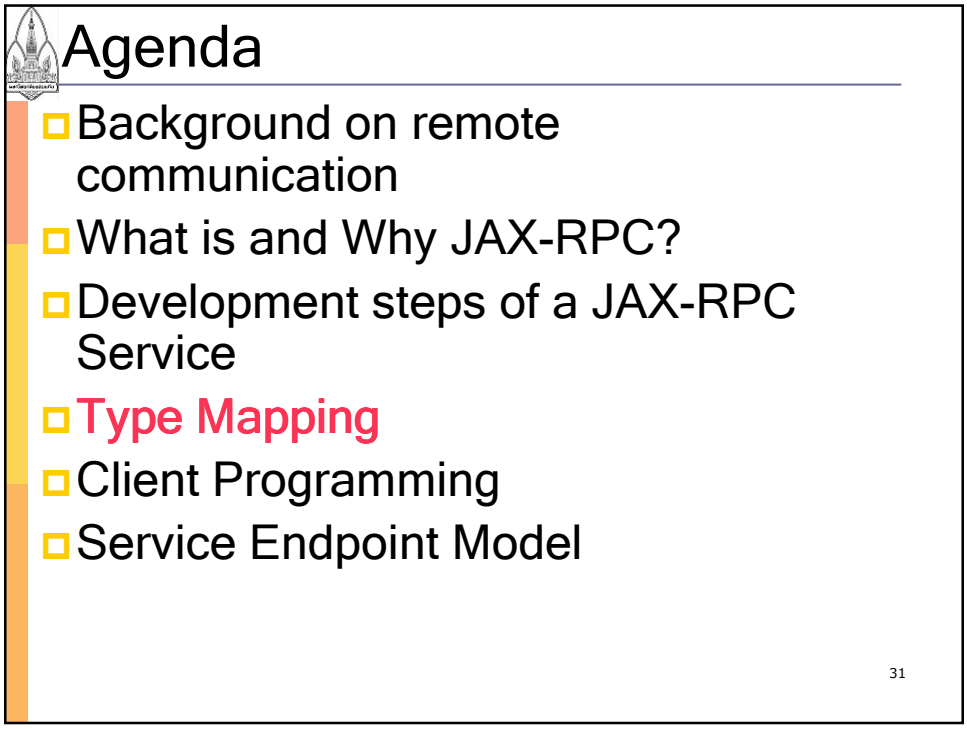
  http://localhost:8080/hello-jaxrpc/hello?WSDL

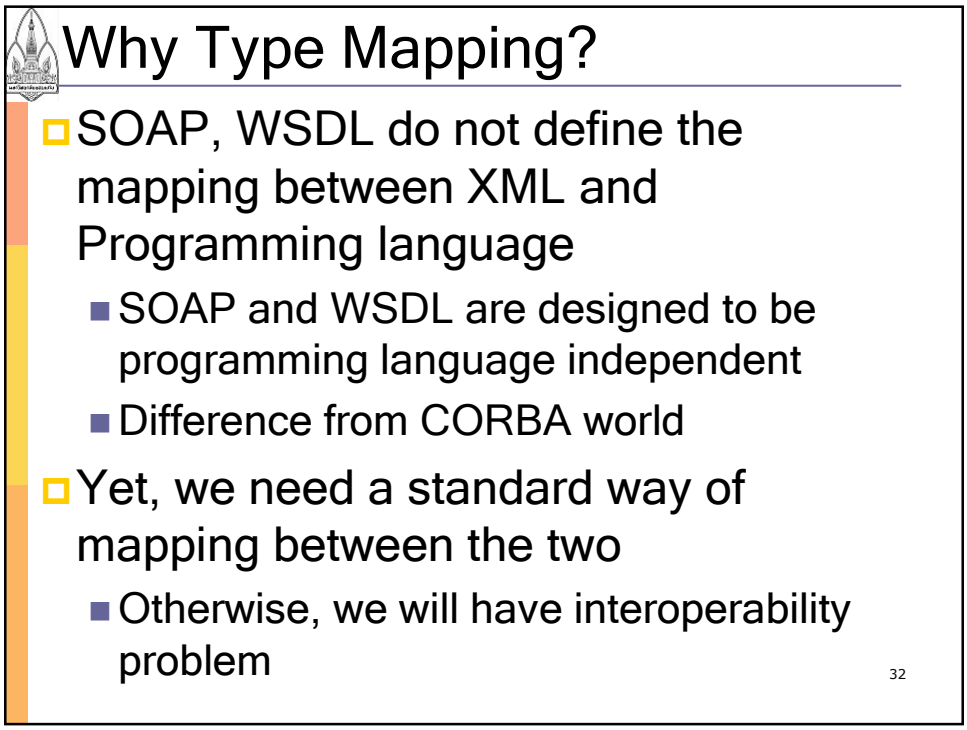
29

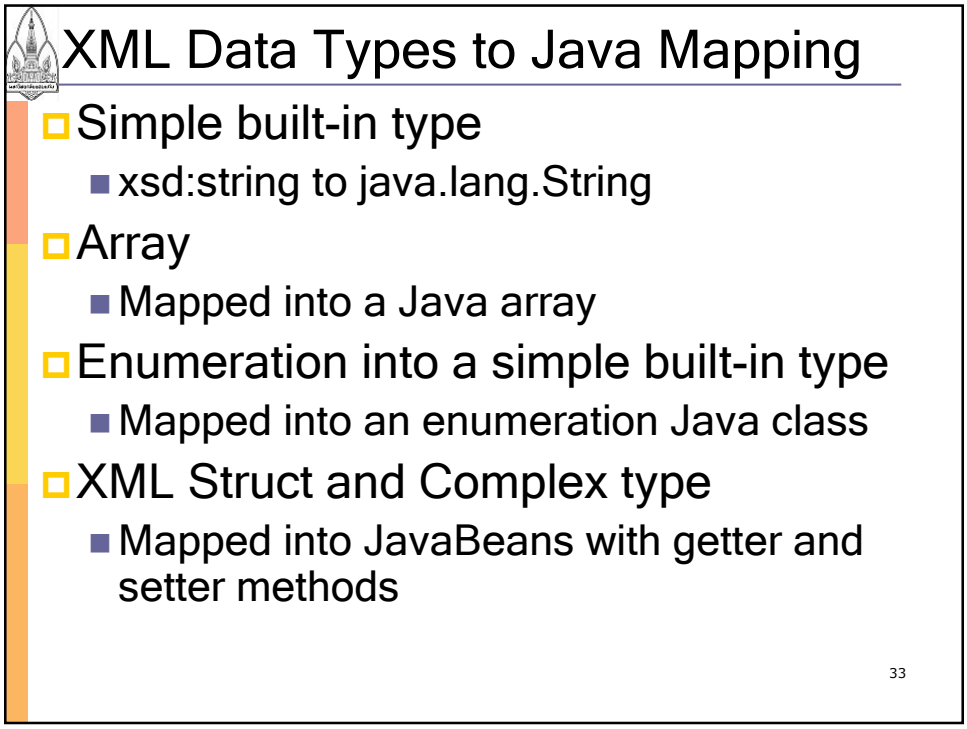
# WSDL of the helloservice Service

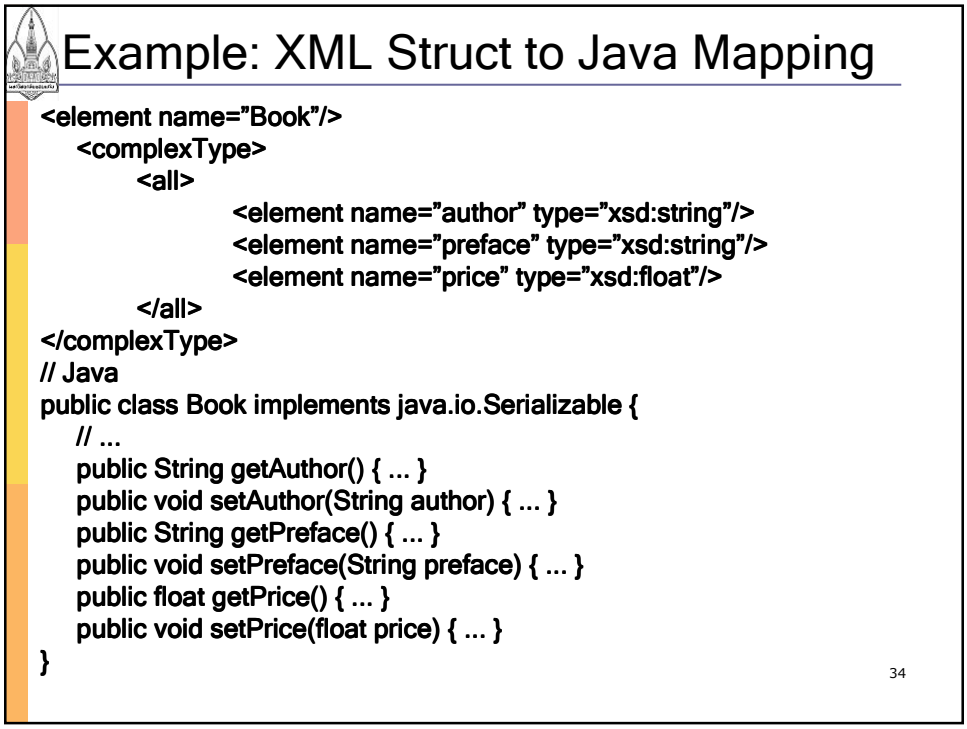


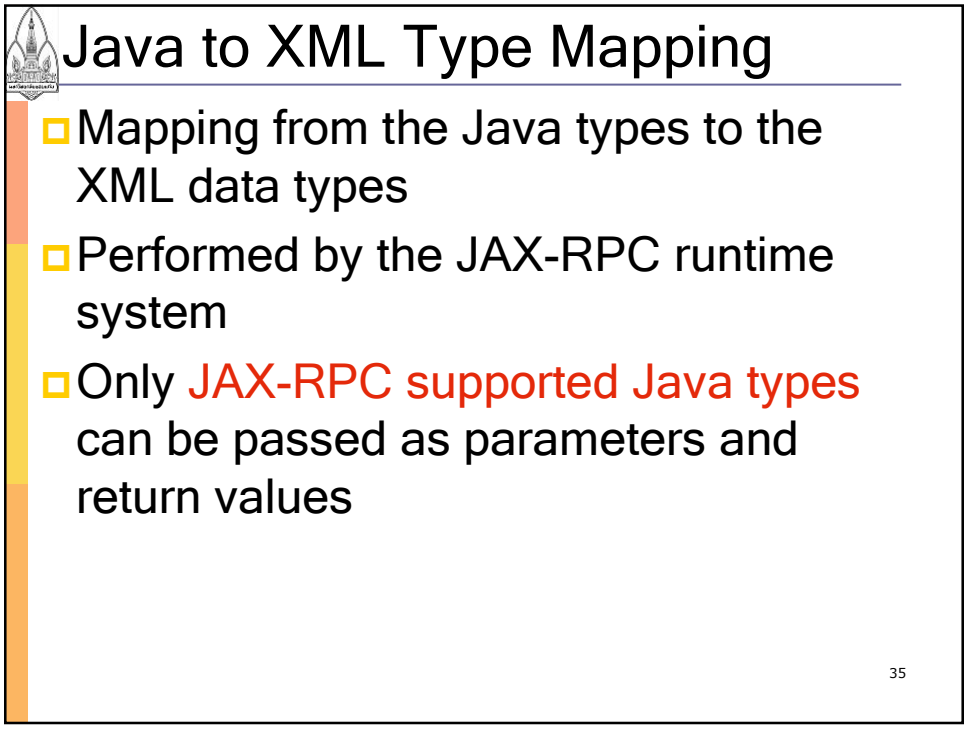

30

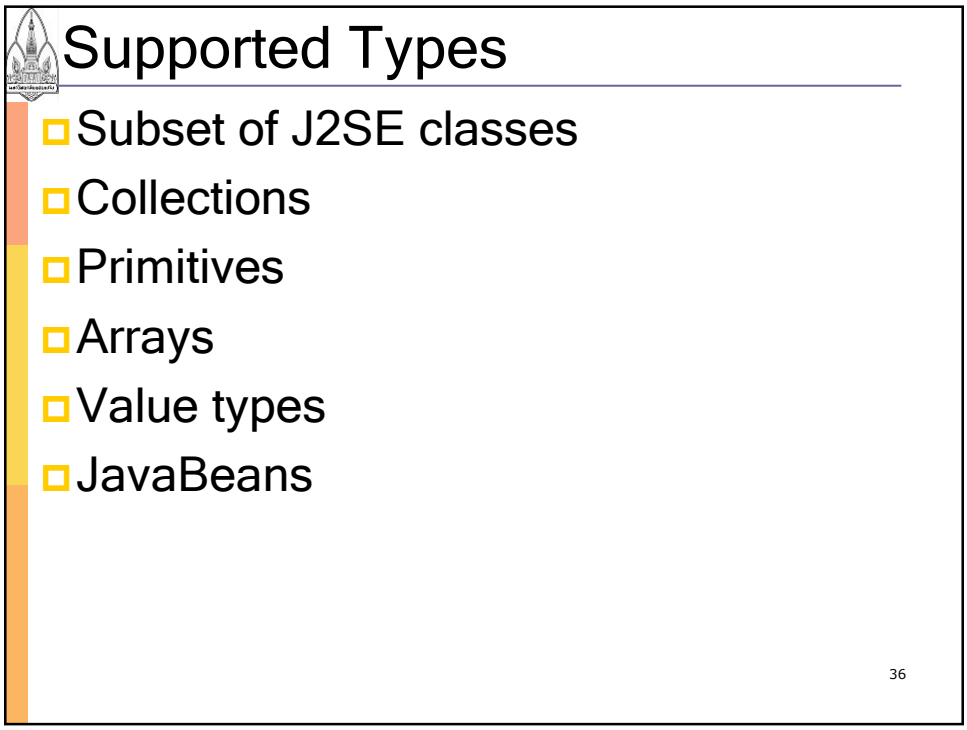

Agenda
Background on remote communication
What is and Why JAX-RPC?
Development steps of a JAX-RPC Service
Type Mapping
Client Programming
Service Endpoint Model
31


Why Type Mapping?
SOAP, WSDL do not define the mapping between XML and Programming language
SOAP and WSDL are designed to be programming language independent
Difference from CORBA world
Yet, we need a standard way of mapping between the two
Otherwise, we will have interoperability problem
32


XML Data Types to Java Mapping
Simple built-in type
xsd:string to java.lang.String
Array
Mapped into a Java array
Enumeration into a simple built-in type
Mapped into an enumeration Java class
XML Struct and Complex type
Mapped into JavaBeans with getter and setter methods
33


Example: XML Struct to Java Mapping
<element name="Book"/>
<complexType>
<all>
<element name="author" type="xsd:string"/>
<element name="preface" type="xsd:string"/>
<element name="price" type="xsd:float"/>
</all>
</complexType>
// Java
public class Book implements java.io.Serializable {
// ...
public String getAuthor() { ... }
public void setAuthor(String author) { ... }
public String getPreface() { ... }
public void setPreface(String preface) { ... }
public float getPrice() { ... }
public void setPrice(float price) { ... }
}
34


Java to XML Type Mapping
Mapping from the Java types to the XML data types
Performed by the JAX-RPC runtime system
Only JAX-RPC supported Java types can be passed as parameters and return values
35


Supported Types
Subset of J2SE classes
Collections
Primitives
Arrays
Value types
JavaBeans
36

## Subset of J2SE classes

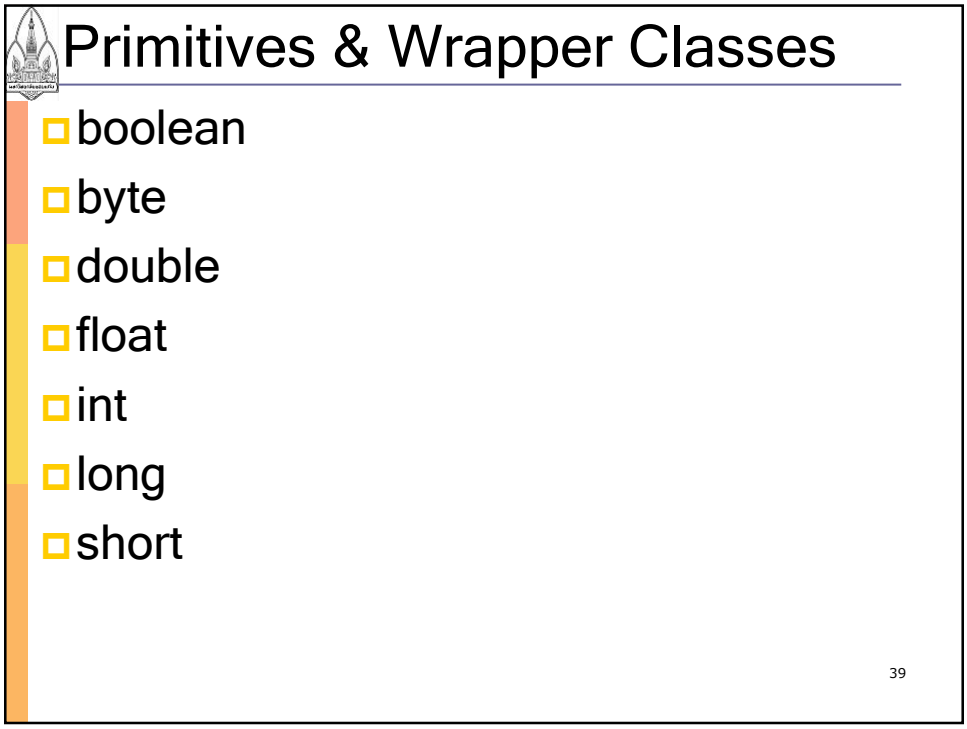
- java.lang.Boolean, java.lang.Byte, java.lang.Double, java.lang.Float, java.lang.Integer, java.lang.Long, java.lang.Short, java.lang.String
- java.math.BigDecimal, java.math.BigInteger
- java.net.URI
- java.util.Calendar, java.util.Date

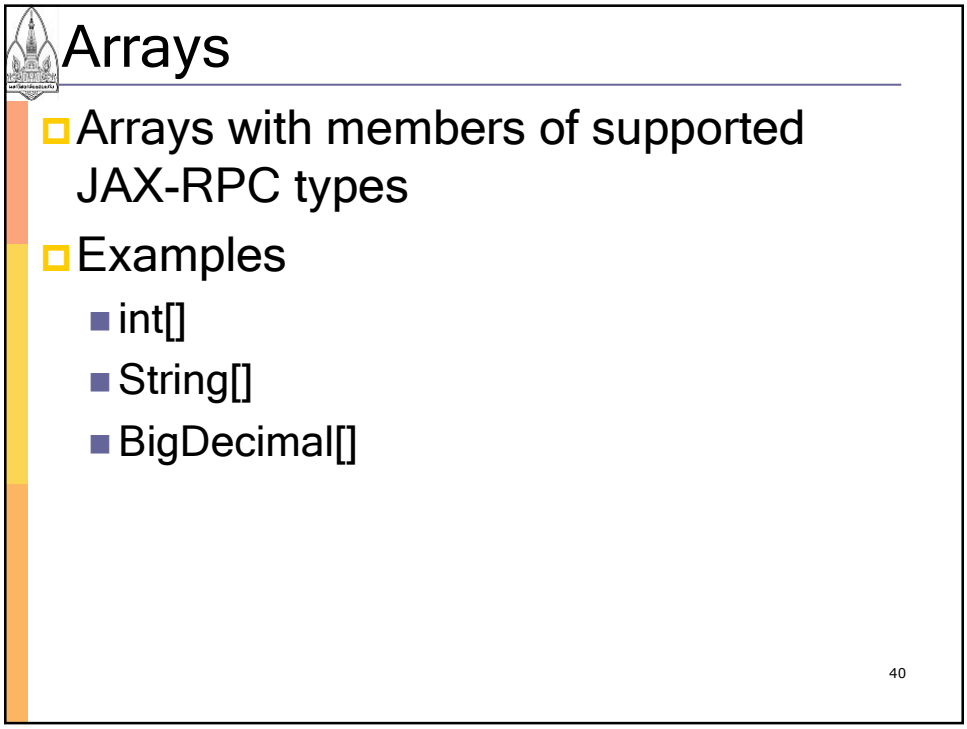
37

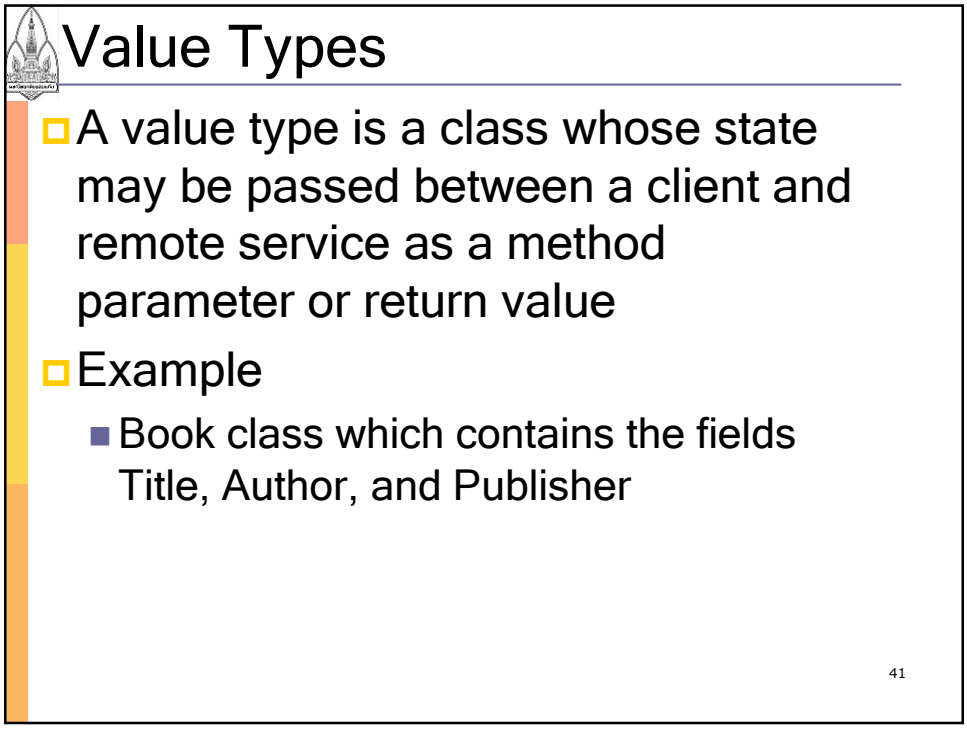
## Collections

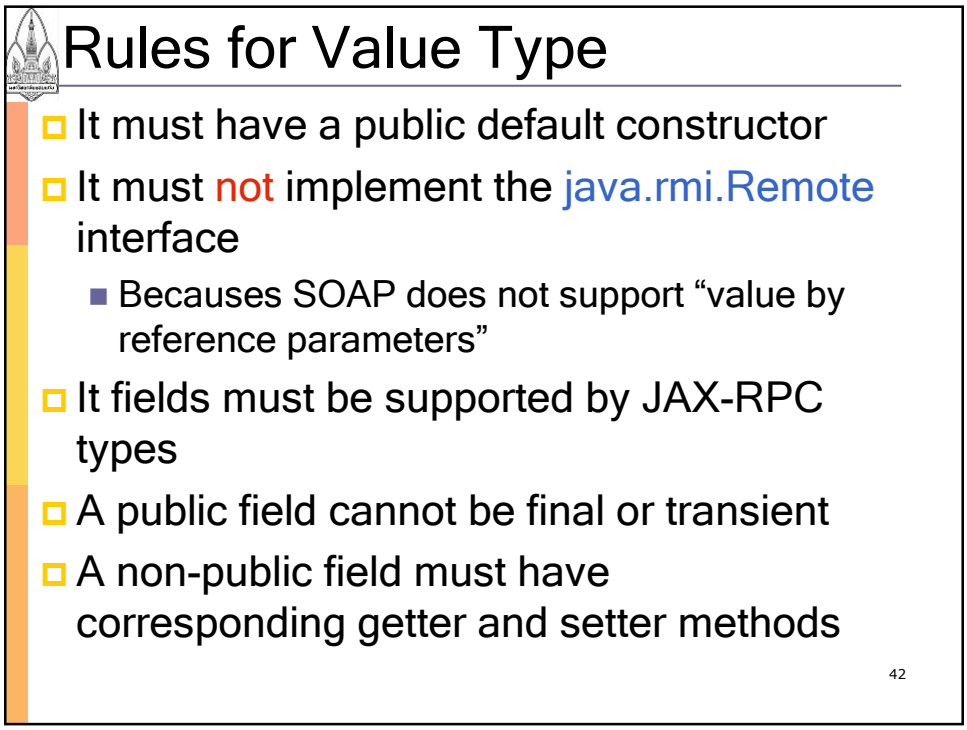
- List
  - ArrayList, LinkedList, Stack, Vector
- Map
  - HashMap, Hashtable, Properties, TreeMap
- Set
  - HashSet, TreeSet

38


Primitives & Wrapper Classes
boolean
byte
double
float
int
long
short
39


Arrays
Arrays with members of supported JAX-RPC types
Examples
int[]
String[]
BigDecimal[]
40


Value Types
A value type is a class whose state may be passed between a client and remote service as a method parameter or return value
Example
Book class which contains the fields Title, Author, and Publisher
41


Rules for Value Type
It must have a public default constructor
It must not implement the java.rmi.Remote interface
Becauses SOAP does not support “value by reference parameters”
It fields must be supported by JAX-RPC types
A public field cannot be final or transient
A non-public field must have corresponding getter and setter methods
42

## Example: Value Types

```
public class MeetingInfo {
   // private fields
   private String id;

   // public fields - does not need getter
   // setter methods
   public String address;

   // has to have getter and setter for
   // non-public fields
   public String getID() { …}
   public void setID(String id) {…}
}
```

43

## JavaBeans

- Must follow the same rules for Value types
- Must have a getter and setter method for each bean property
- The type of the bean property must be a supported JAX-RPC type

44

# Example: JavaBean Type

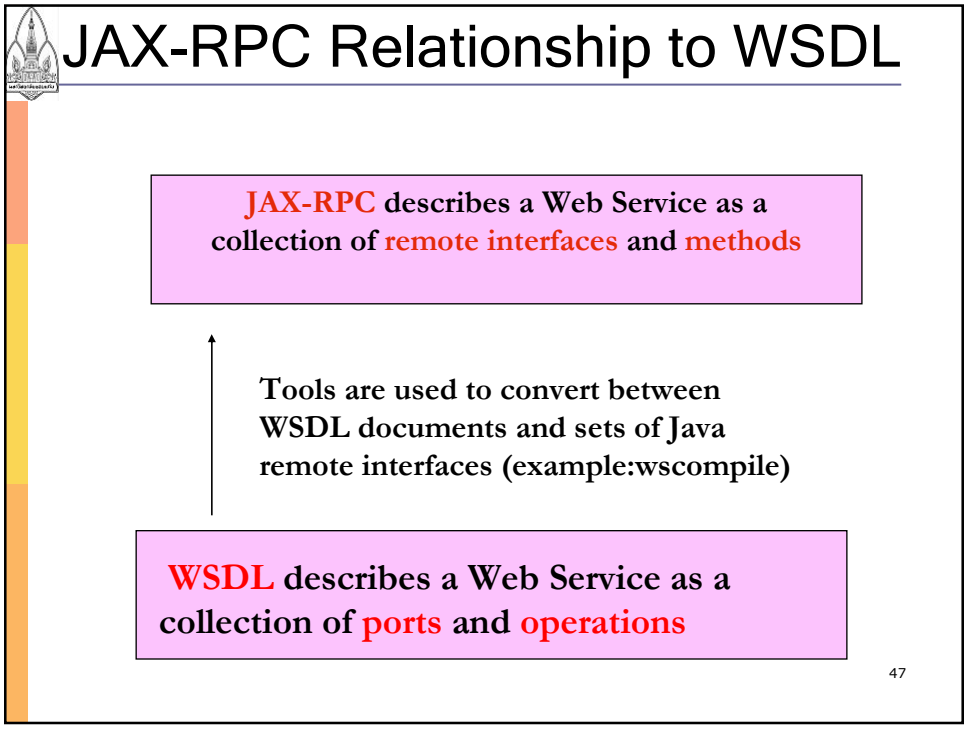
```
public class AddressBean implements java.io.Serializable {

    private String street;private String city;private String
    state;private String zip;

    public AddressBean() { }
        public AddressBean(String street, String city) {
        this.street = street;
        this.city = city;
    }
    public String getStreet() { return street; }
    public void setStreet(String street) { this.street = street; }
    public String getCity() { return city; }
    public void setCity(String city) { this.city = city; }
    …
}
```

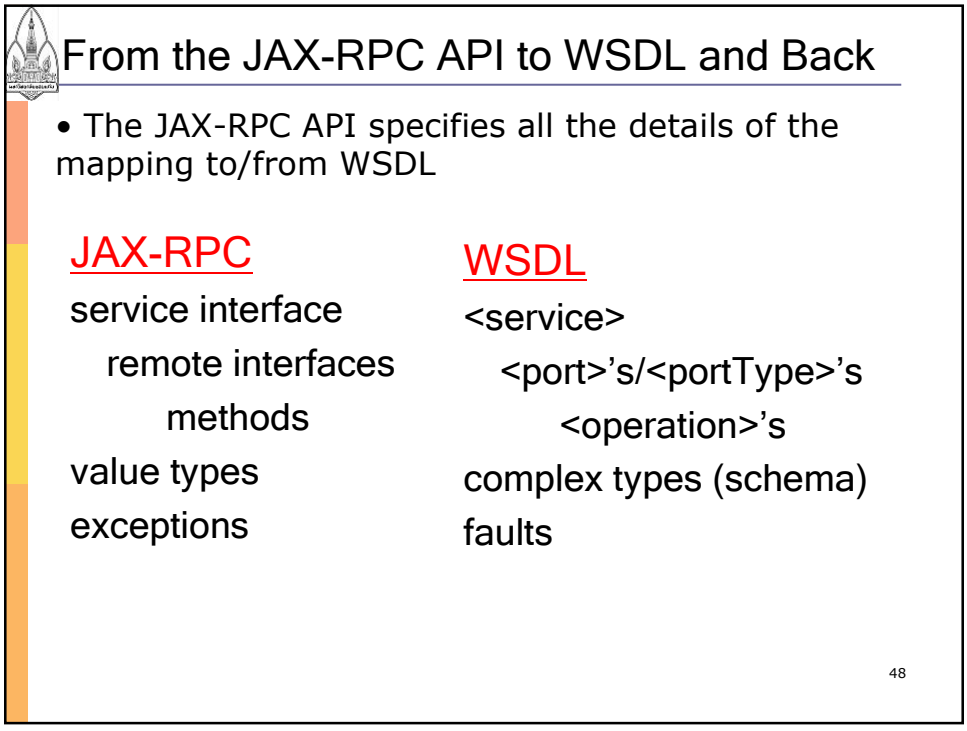
45

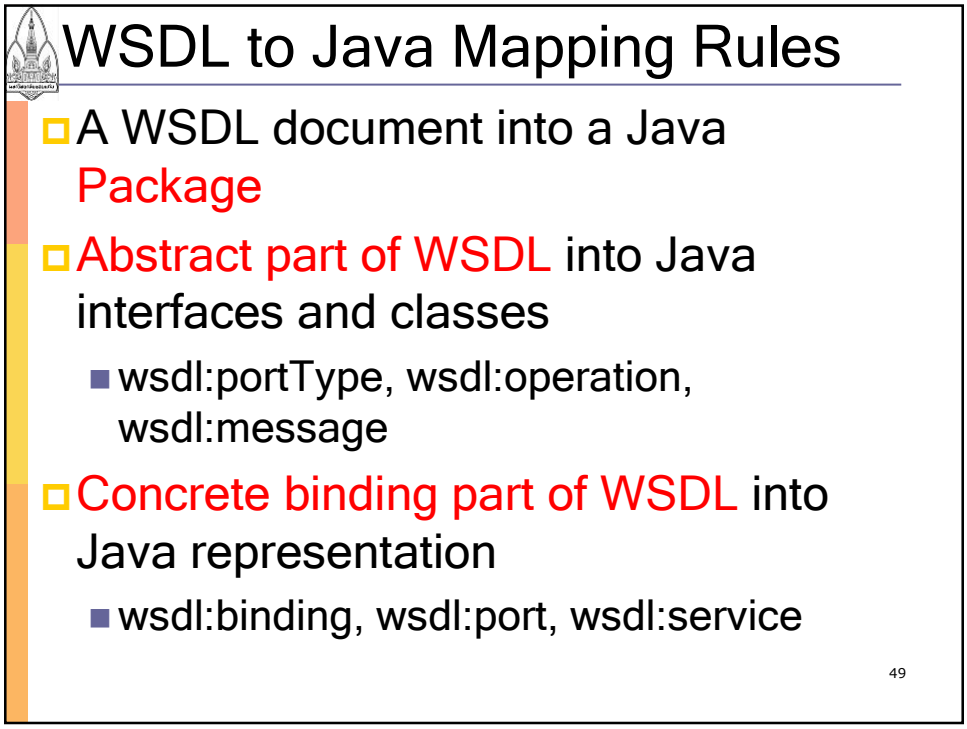
# WSDL & JAX-RPC

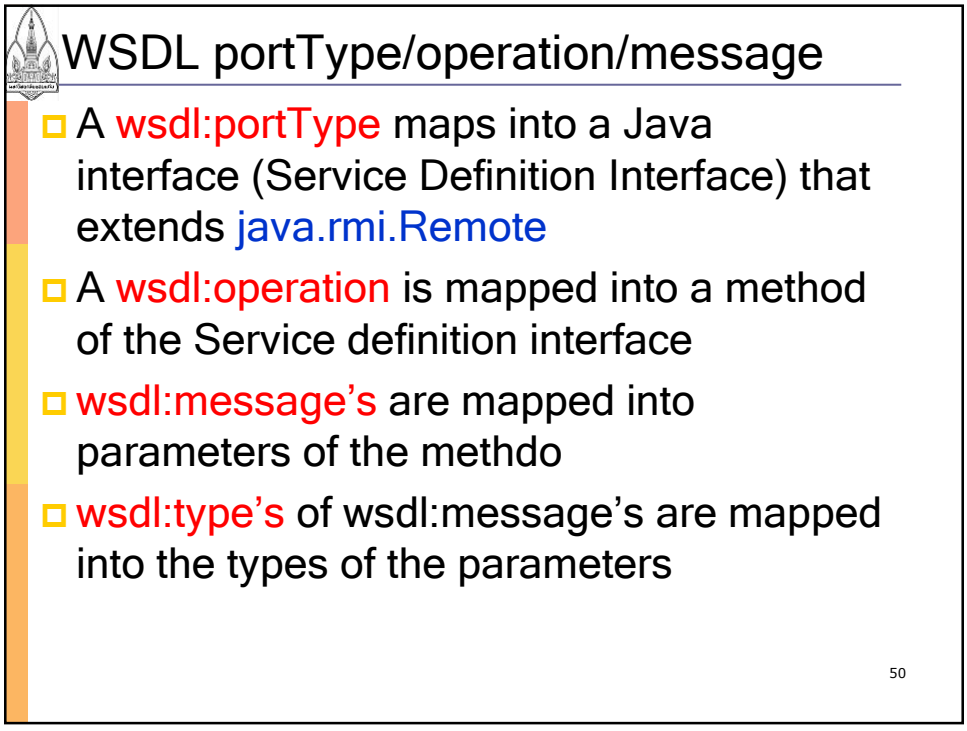
- Services are described using WSDL
- WSDL is the only contract needed between service provider and client
- JAX-RPC tools in J2EE 1.4 SDK
  - wscompile tool creates “abstract part of WSDL” from Service definition interface (server side)
  - wsdeploy tool creates “complete WSDL” including port address and tie classes (server side)
  - wscompile tool creates stubs (client side)

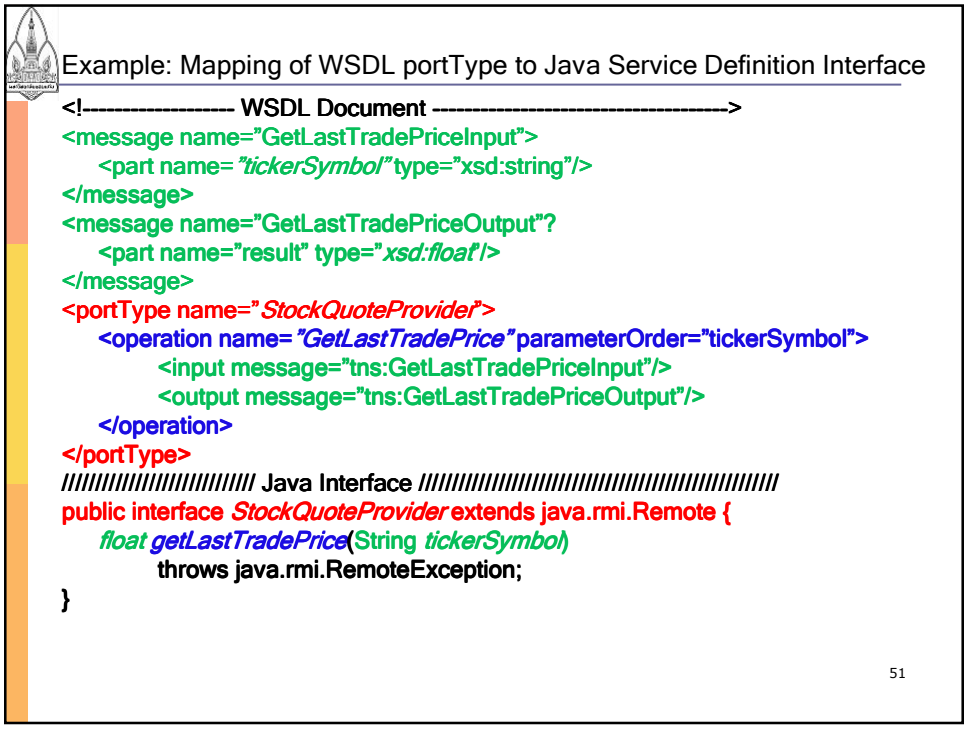
46

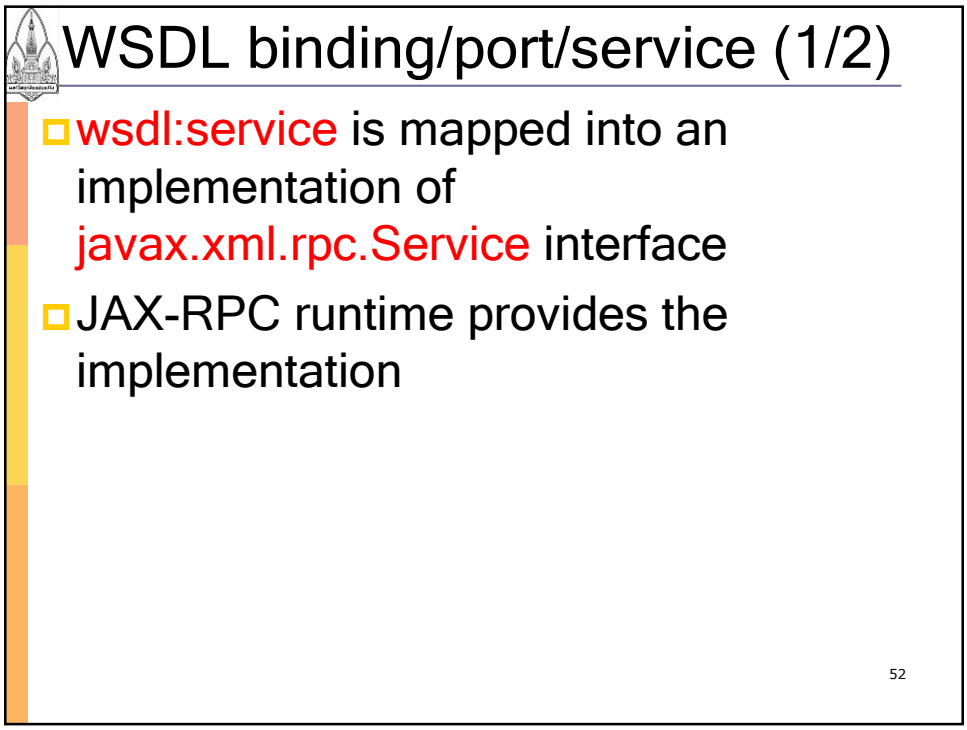

JAX-RPC Relationship to WSDL
JAX-RPC describes a Web Service as a collection of remote interfaces and methods
Tools are used to convert between WSDL documents and sets of Java remote interfaces (example:wscompile)
WSDL describes a Web Service as a collection of ports and operations
47

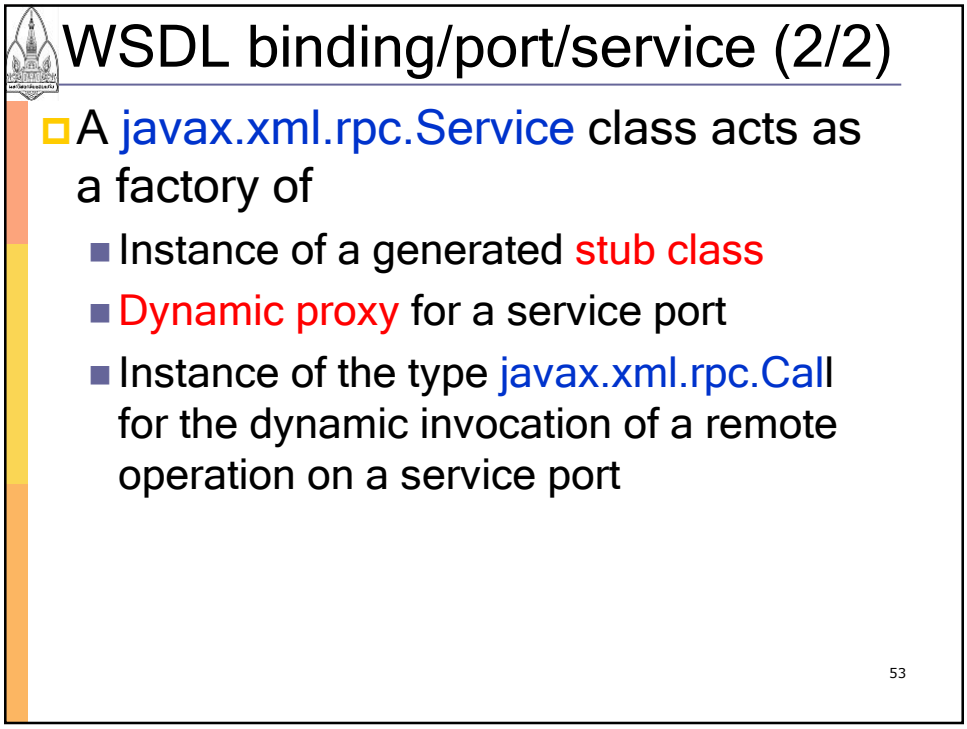

From the JAX-RPC API to WSDL and Back
• The JAX-RPC API specifies all the details of the mapping to/from WSDL
JAX-RPC
service interface
remote interfaces
methods
value types
exceptions
WSDL
<service>
<port>'s/<portType>'s
<operation>'s
complex types (schema)
faults
48

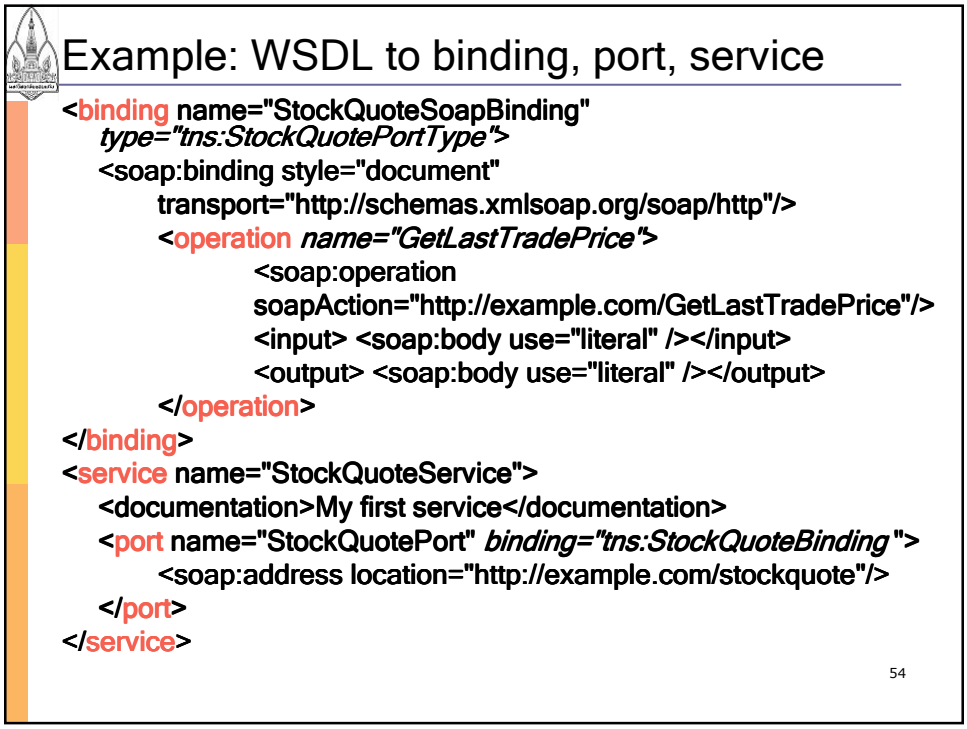

WSDL to Java Mapping Rules
A WSDL document into a Java Package
Abstract part of WSDL into Java interfaces and classes
wsdl:portType, wsdl:operation, wsdl:message
Concrete binding part of WSDL into Java representation
wsdl:binding, wsdl:port, wsdl:service
49


WSDL portType/operation/message
A wsdl:portType maps into a Java interface (Service Definition Interface) that extends java.rmi.Remote
A wsdl:operation is mapped into a method of the Service definition interface
wsdl:message's are mapped into parameters of the methdo
wsdl:type's of wsdl:message's are mapped into the types of the parameters
50


Example: Mapping of WSDL portType to Java Service Definition Interface
<!---------- WSDL Document ---------->
<message name="GetLastTradePriceInput">
<part name="tickerSymbol" type="xsd:string"/>
</message>
<message name="GetLastTradePriceOutput"?
<part name="result" type="xsd:float"/>
</message>
<portType name="StockQuoteProvider">
<operation name="GetLastTradePrice" parameterOrder="tickerSymbol">
<input message="tns:GetLastTradePriceInput"/>
<output message="tns:GetLastTradePriceOutput"/>
</operation>
</portType>
////////////////////////// Java Interface ////////////////////////////////////////////////////
public interface StockQuoteProvider extends java.rmi.Remote {
float getLastTradePrice(String tickerSymbol)
throws java.rmi.RemoteException;
}
51


WSDL binding/port/service (1/2)
wsdl:service is mapped into an implementation of javax.xml.rpc.Service interface
JAX-RPC runtime provides the implementation
52


WSDL binding/port/service (2/2)
A javax.xml.rpc.Service class acts as a factory of
Instance of a generated stub class
Dynamic proxy for a service port
Instance of the type javax.xml.rpc.Call for the dynamic invocation of a remote operation on a service port
53


Example: WSDL to binding, port, service
<binding name="StockQuoteSoapBinding"
type="tns:StockQuotePortType">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="GetLastTradePrice">
<soap:operation
soapAction="http://example.com/GetLastTradePrice"/>
<input> <soap:body use="literal" /></input>
<output> <soap:body use="literal" /></output>
</operation>
</binding>
<service name="StockQuoteService">
<documentation>My first service</documentation>
<port name="StockQuotePort" binding="tns:StockQuoteBinding ">
<soap:address location="http://example.com/stockquote"/>
</port>
</service>
54

## javax.xml.rpc.Service

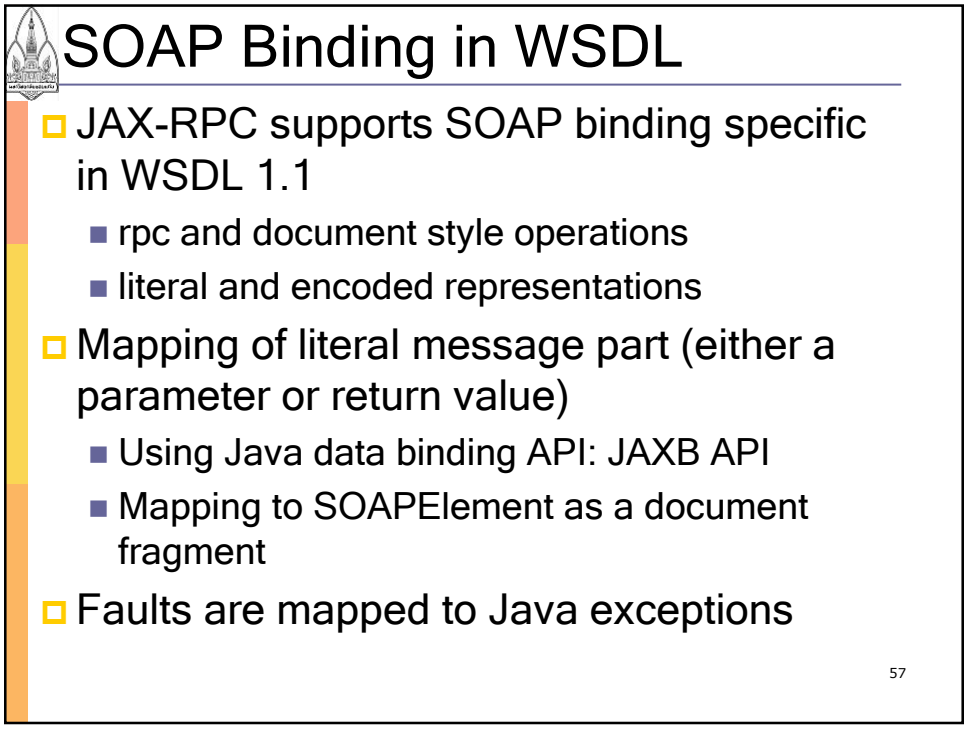
```
package javax.xml.rpc;
public interface Service {
    public java.rmi.Remote getPort(QName portName,
        Class proxyInterface)
        throws JAXRPCException;
    public Call createCall(QName portName)
        throws JAXRPCException;
    public Call createCall(QName portName, String
    operationName)
        throws JAXRPCException;
    public Call createCall() throws JAXRPCException;
    public java.net.URL getWSDLDocumentLocation();
    public QName getServiceName();
    public java.util.Iterator getPorts();
}
```

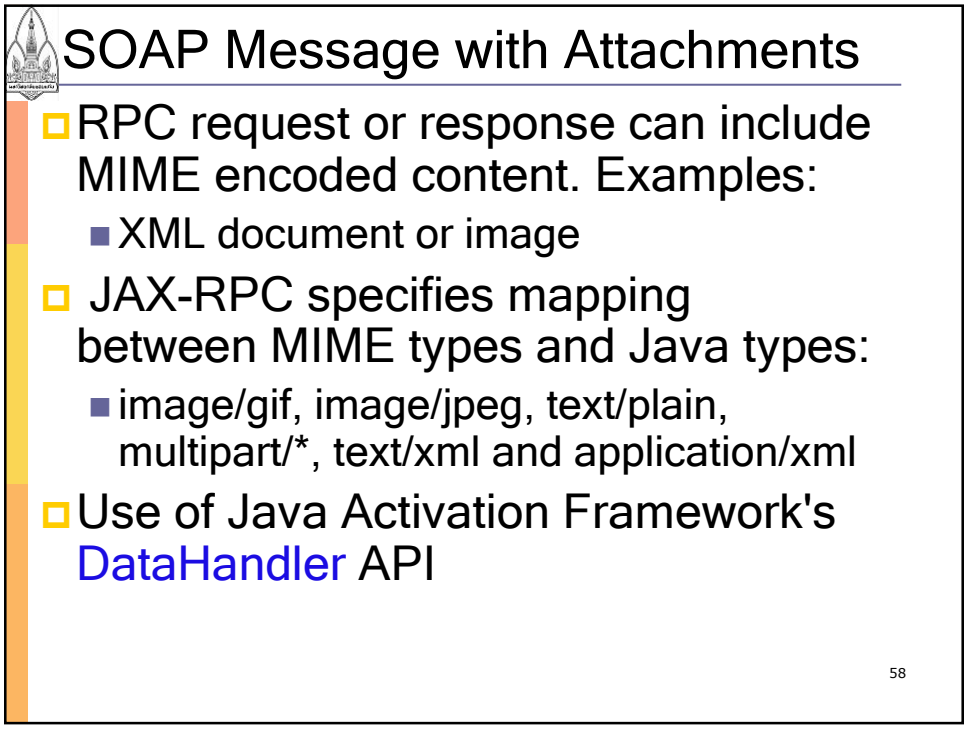
55

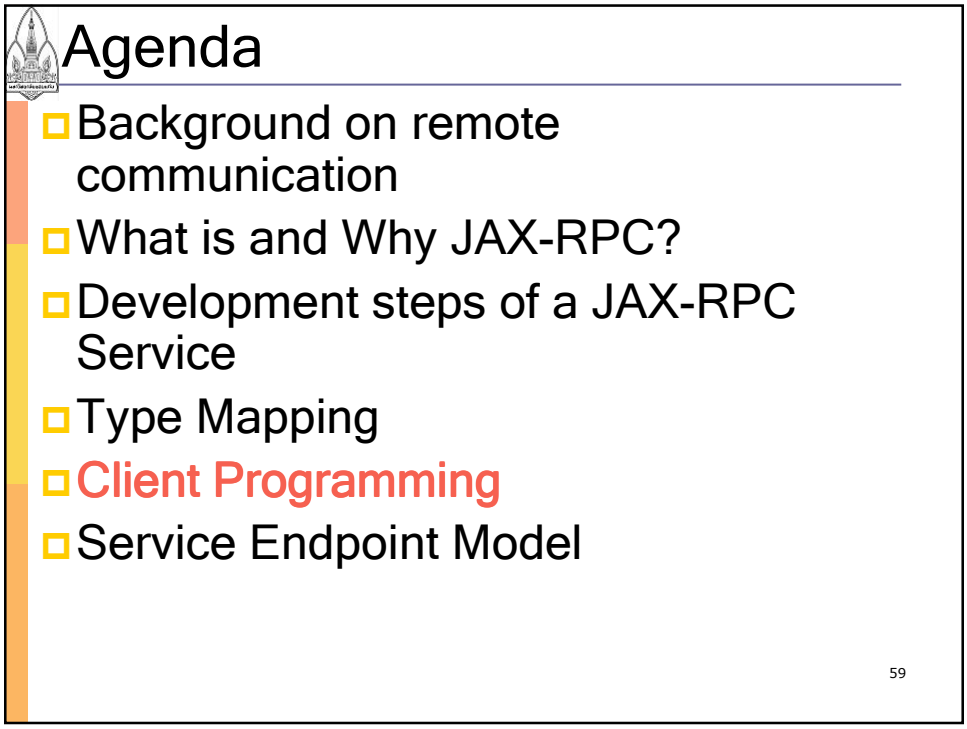
## Typical WSDL to Java Mapping Tool

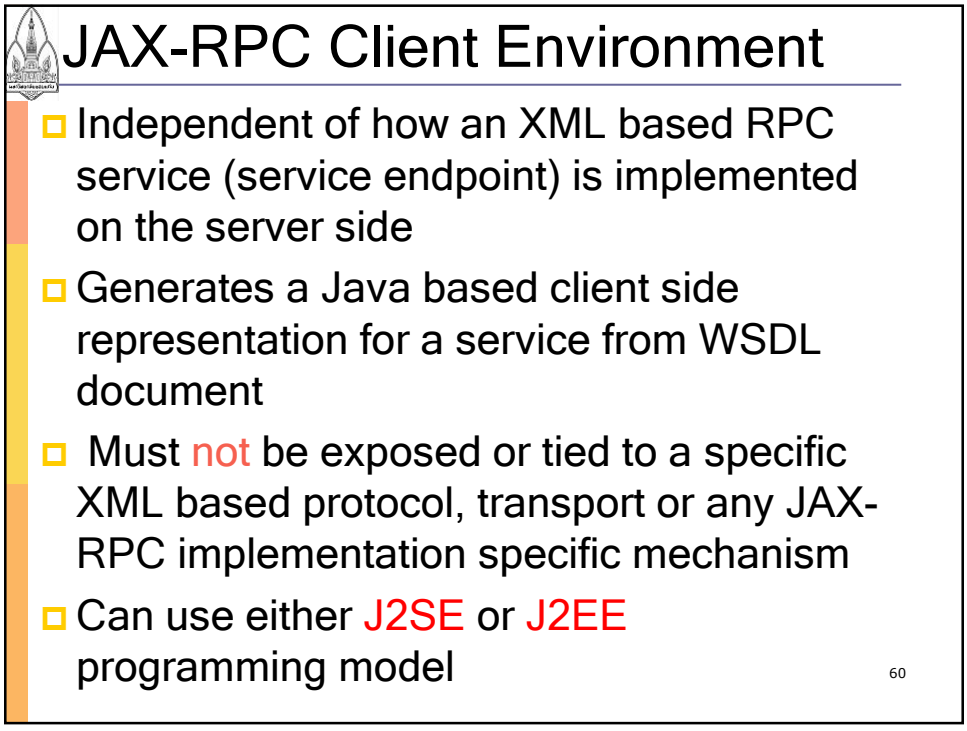
- Read WSDL document and then generates
  - Service interface (javax.xml.rpc.Service) and its implementation
  - Service definition interface (Extension of java.rmi.Remote)
  - Stub and tie classes
  - Additional classes
    - Serializers, deserializers

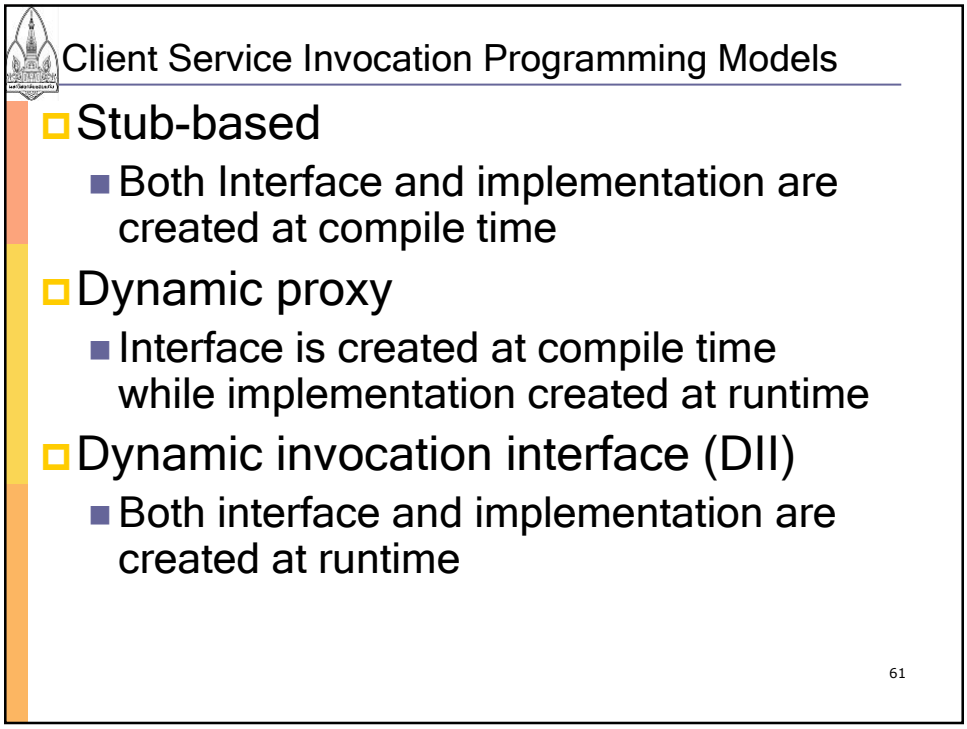
56

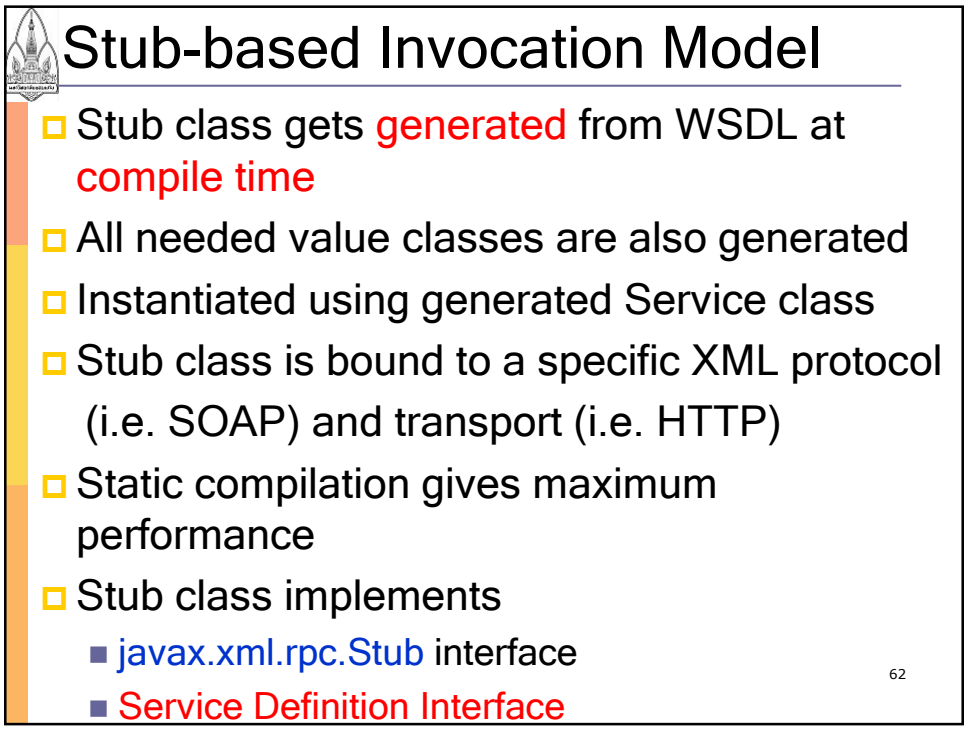

SOAP Binding in WSDL
JAX-RPC supports SOAP binding specific in WSDL 1.1
rpc and document style operations
literal and encoded representations
Mapping of literal message part (either a parameter or return value)
Using Java data binding API: JAXB API
Mapping to SOAPElement as a document fragment
Faults are mapped to Java exceptions
57


SOAP Message with Attachments
RPC request or response can include MIME encoded content. Examples:
XML document or image
JAX-RPC specifies mapping between MIME types and Java types:
image/gif, image/jpeg, text/plain, multipart/*, text/xml and application/xml
Use of Java Activation Framework's DataHandler API
58


Agenda
Background on remote communication
What is and Why JAX-RPC?
Development steps of a JAX-RPC Service
Type Mapping
Client Programming
Service Endpoint Model
59


JAX-RPC Client Environment
Independent of how an XML based RPC service (service endpoint) is implemented on the server side
Generates a Java based client side representation for a service from WSDL document
Must not be exposed or tied to a specific XML based protocol, transport or any JAX-RPC implementation specific mechanism
Can use either J2SE or J2EE programming model
60


Client Service Invocation Programming Models
Stub-based
Both Interface and implementation are created at compile time
Dynamic proxy
Interface is created at compile time while implementation created at runtime
Dynamic invocation interface (DII)
Both interface and implementation are created at runtime
61


Stub-based Invocation Model
Stub class gets generated from WSDL at compile time
All needed value classes are also generated
Instantiated using generated Service class
Stub class is bound to a specific XML protocol (i.e. SOAP) and transport (i.e. HTTP)
Static compilation gives maximum performance
Stub class implements
javax.xml.rpc.Stub interface
Service Definition Interface
62

## Steps of Coding Static Stub Client

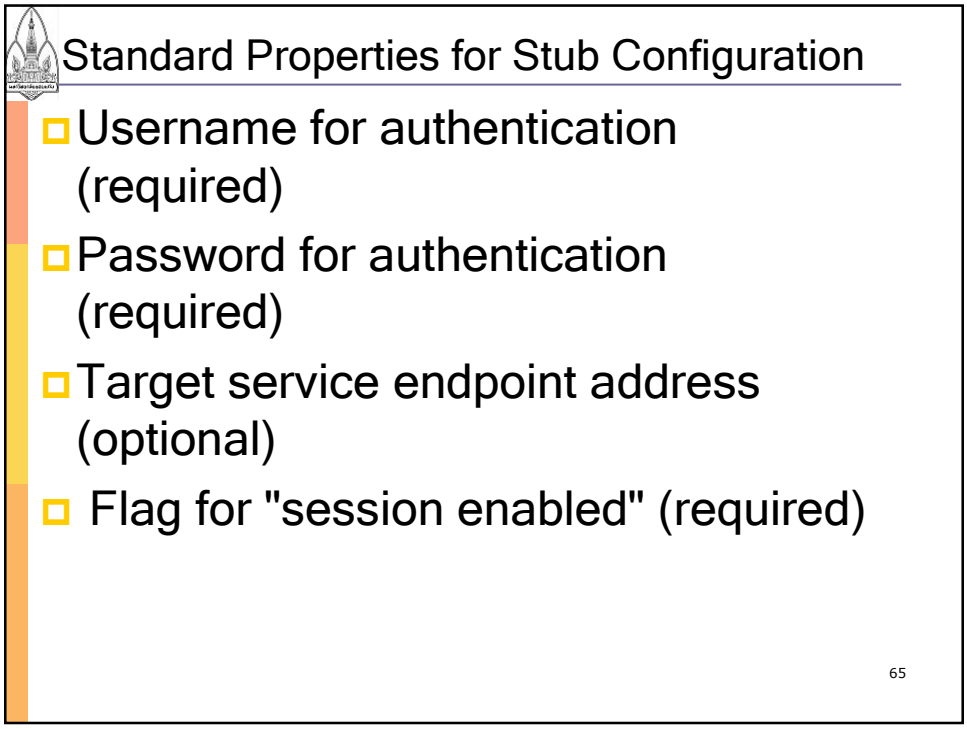
- Creates a Stub object
  - (Stub)(new MyHelloService_Impl().getHelloIFPort())
- Sets the endpoint address that the stub uses to access the service
  - stub._setProperty (javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY, args[0]);
- Casts stub to the service endpoint interface, HelloIF
  - HelloIF hello = (HelloIF)stub

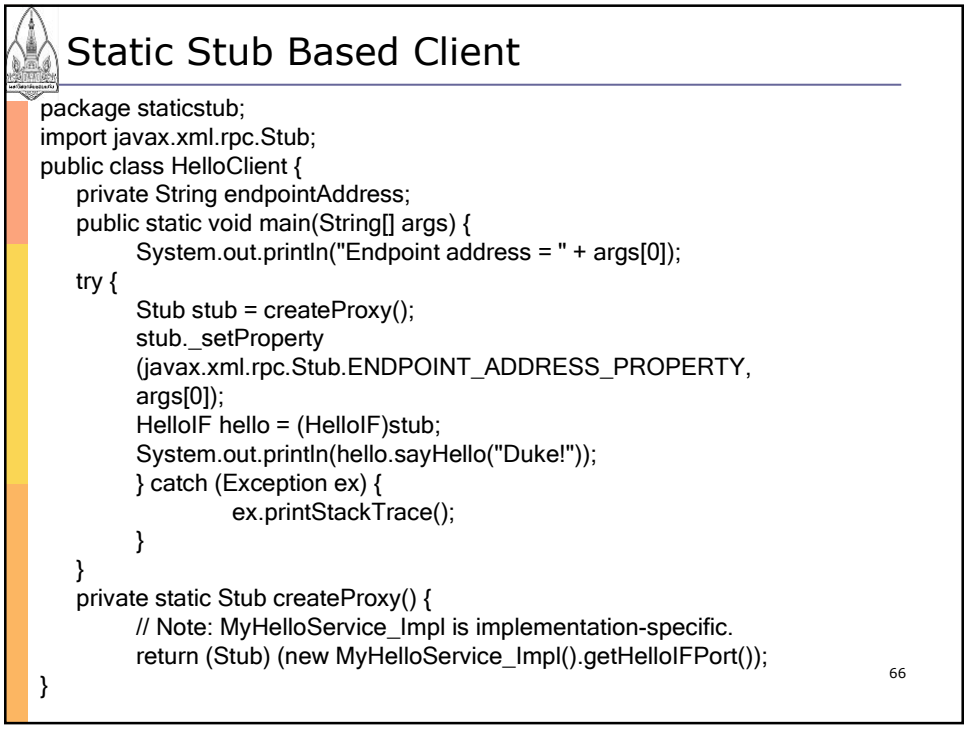
63

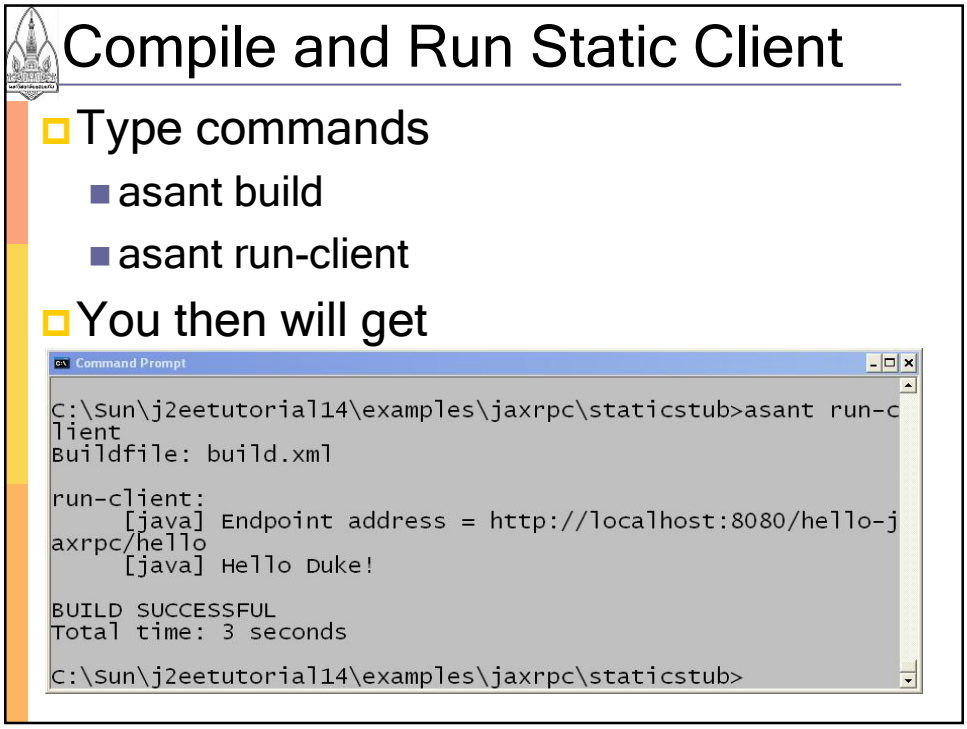
## Stub Configuration

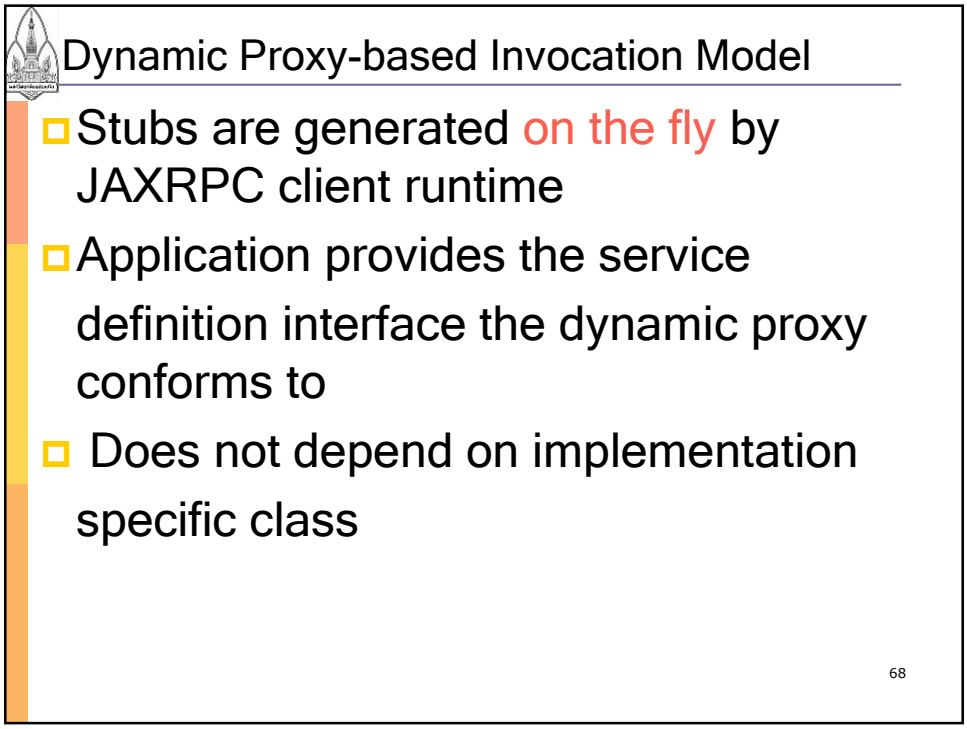
- Stub instance must be configured
  - XML protocol binding
  - endpoint address
- Can be configured in two ways
  - Static configuration based on the WSDL description of a target service endpoint
  - *wsdl:binding, soap:binding, wsdl:port*
- Runtime configuration using the *javax.xml.rpc.Stub* API

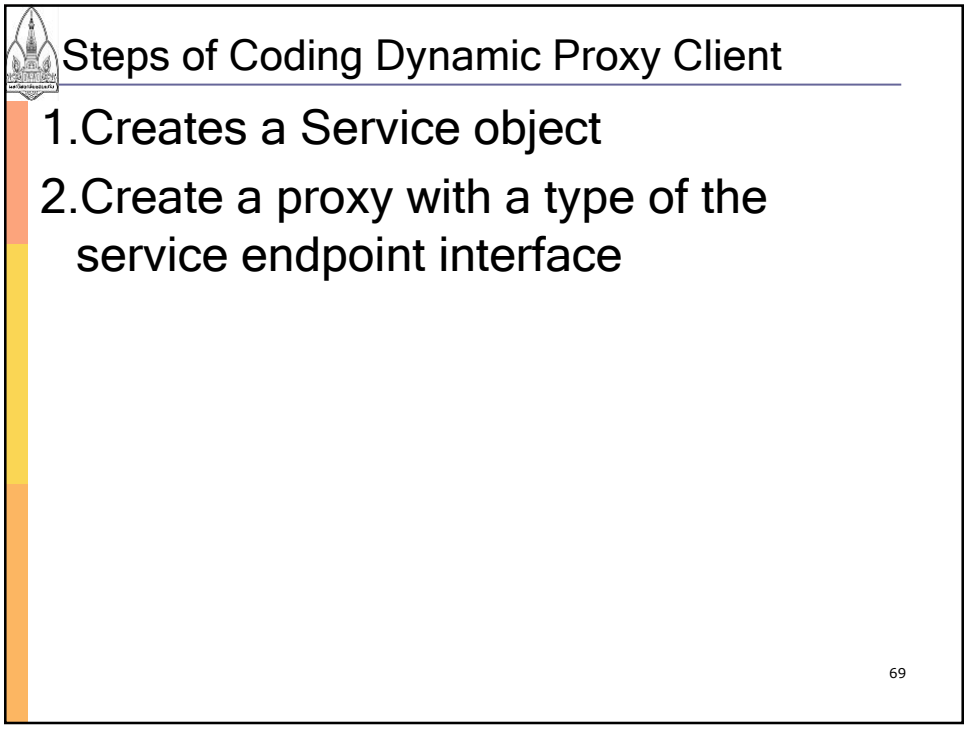
64

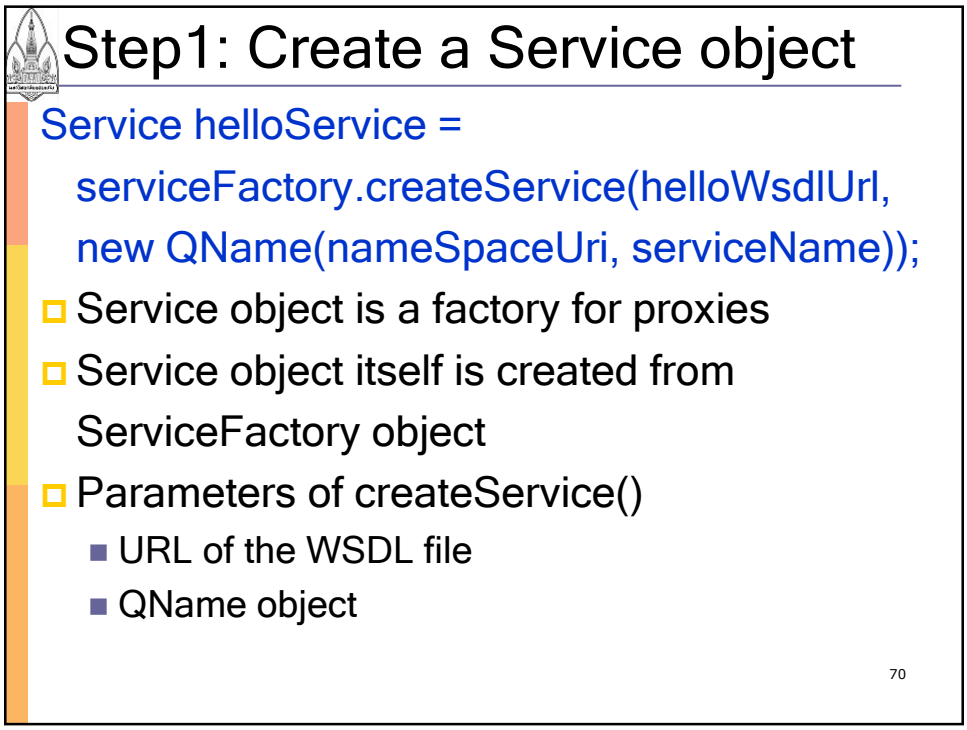

Standard Properties for Stub Configuration
Username for authentication (required)
Password for authentication (required)
Target service endpoint address (optional)
Flag for "session enabled" (required)
65


Static Stub Based Client
package staticstub;
import javax.xml.rpc.Stub;
public class HelloClient {
private String endpointAddress;
public static void main(String[] args) {
System.out.println("Endpoint address = " + args[0]);
try {
Stub stub = createProxy();
stub._setProperty
(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
args[0]);
HelloIF hello = (HelloIF)stub;
System.out.println(hello.sayHello("Duke!"));
} catch (Exception ex) {
ex.printStackTrace();
}
}
private static Stub createProxy() {
// Note: MyHelloService_Impl is implementation-specific.
return (Stub) (new MyHelloService_Impl().getHelloIFPort());
}
66


Compile and Run Static Client
Type commands
asant build
asant run-client
You then will get
Command Prompt
C:\Sun\j2eetutorial14\examples\jaxrpc\staticstub>asant run-client
Buildfile: build.xml
run-client:
[java] Endpoint address = http://localhost:8080/hello-jaxrpc/hello
[java] Hello Duke!
BUILD SUCCESSFUL
Total time: 3 seconds
C:\Sun\j2eetutorial14\examples\jaxrpc\staticstub>


Dynamic Proxy-based Invocation Model
Stubs are generated on the fly by JAXRPC client runtime
Application provides the service definition interface the dynamic proxy conforms to
Does not depend on implementation specific class
68


Steps of Coding Dynamic Proxy Client
1.Creates a Service object
2.Create a proxy with a type of the service endpoint interface
69


Step1: Create a Service object
Service helloService =
serviceFactory.createService(helloWsdlUrl,
new QName(nameSpaceUri, serviceName));
Service object is a factory for proxies
Service object itself is created from ServiceFactory object
Parameters of createService()
URL of the WSDL file
QName object
70

## Step2: Create a Dynamic Proxy object

```
dynamicproxy.HelloIF myProxy =
  (dynamicproxy.HelloIF)helloService.getPort(
      new QName(nameSpaceUri, portName),
          dynamicproxy.HelloIF.class);
```

- HelloIF class is generated by wscompile
- The port name (HelloIFPort) is specified by the WSDL file

71

## Dynamic Proxy Client (1/2)

```
package dynamicproxy;
import java.net.URL;
import javax.xml.rpc.Service;
import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceFactory;
import dynamicproxy.HelloIF;
public class HelloClient {
public static void main(String[] args) {
   try {
        String UrlString = args[0] + "?WSDL";
        String nameSpaceUri = "urn:Foo";
        String serviceName = "MyHelloService";
        String portName = "HelloIFPort";
        System.out.println("UrlString = " + UrlString);
        URL helloWsdlUrl = new URL(UrlString
```

72

## Dynamic Proxy Client (2/2)

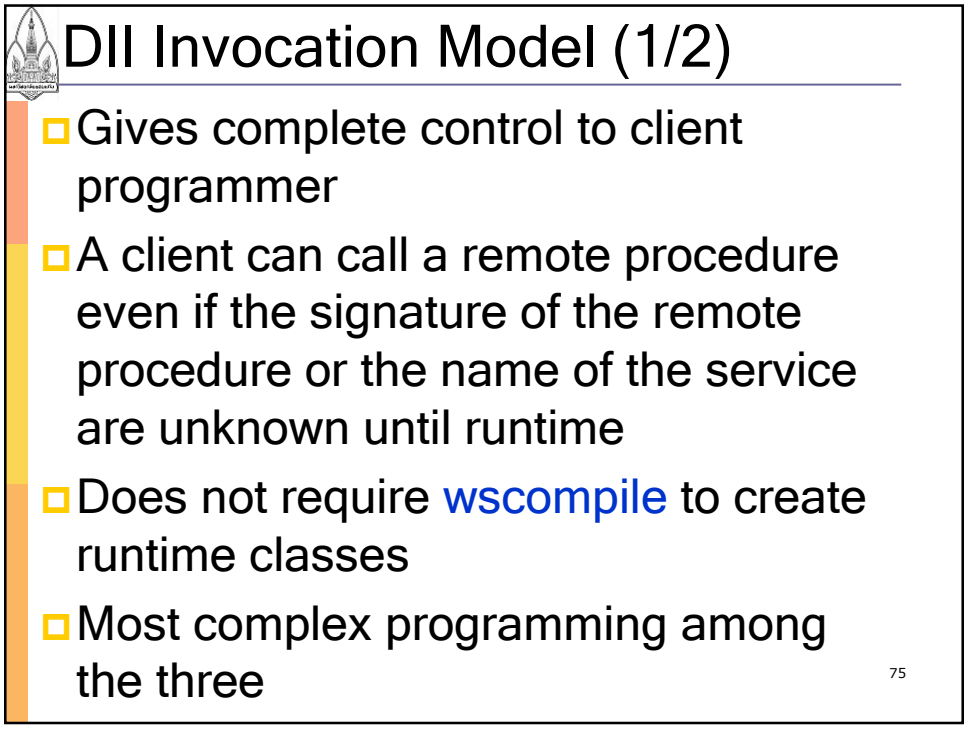
```
        ServiceFactory serviceFactory =
                ServiceFactory.newInstance();
        Service helloService =
        serviceFactory.createService(helloWsdlUrl,
                new QName(nameSpaceUri, serviceName));
        dynamicproxy.HelloIF myProxy =
                (dynamicproxy.HelloIF)
                helloService.getPort(
                new QName(nameSpaceUri, portName),
                dynamicproxy.HelloIF.class);
        System.out.println(myProxy.sayHello("Buzz"));
        } catch (Exception ex) {
                ex.printStackTrace();
        }
    }
}
```

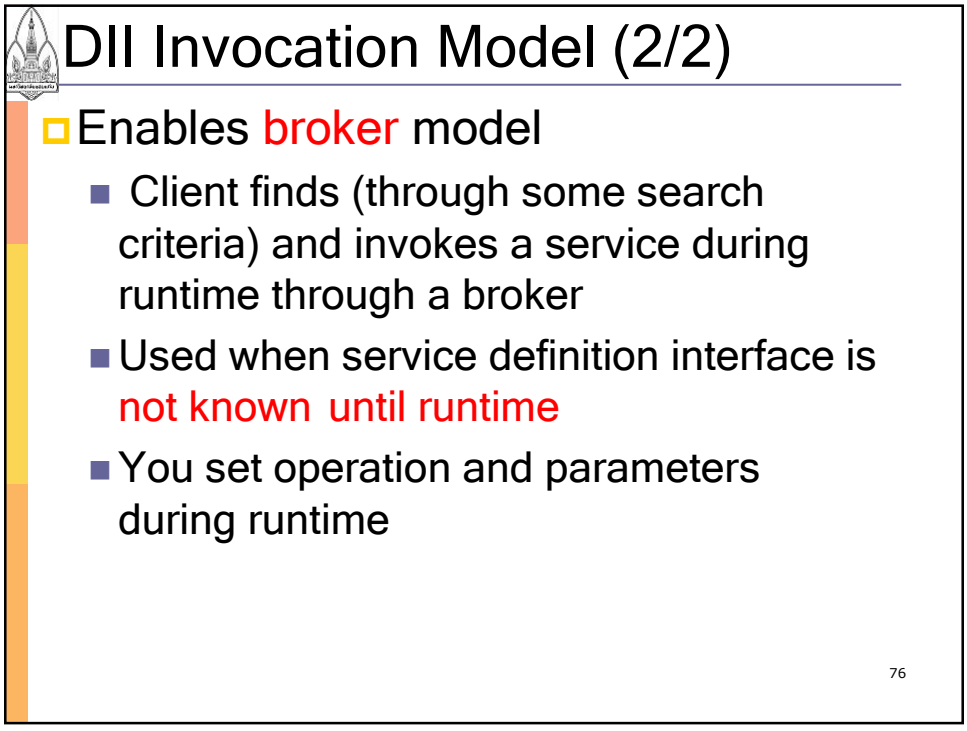
73

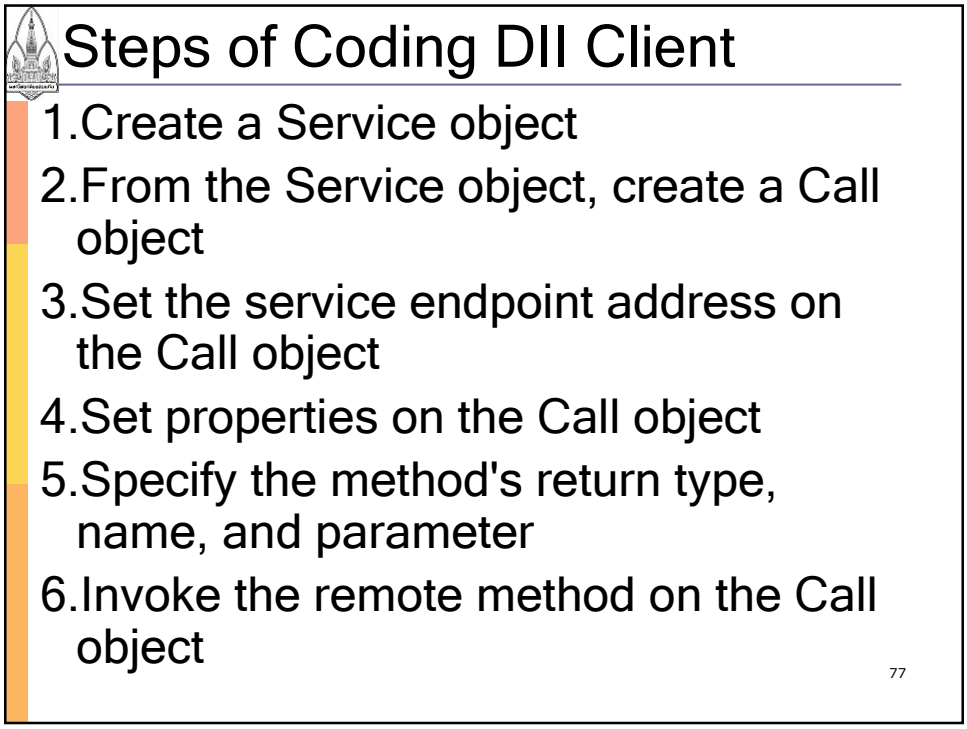
## Compile and Run Dynamic Client

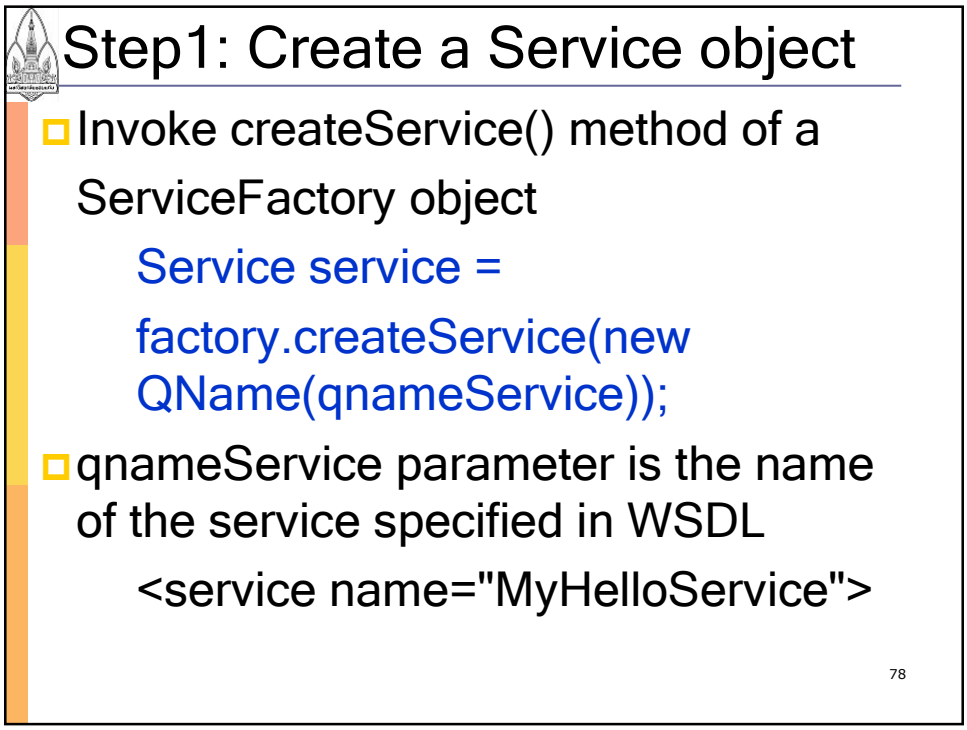
- Type commands
  - asant build
  - asant run-client
- You then will get

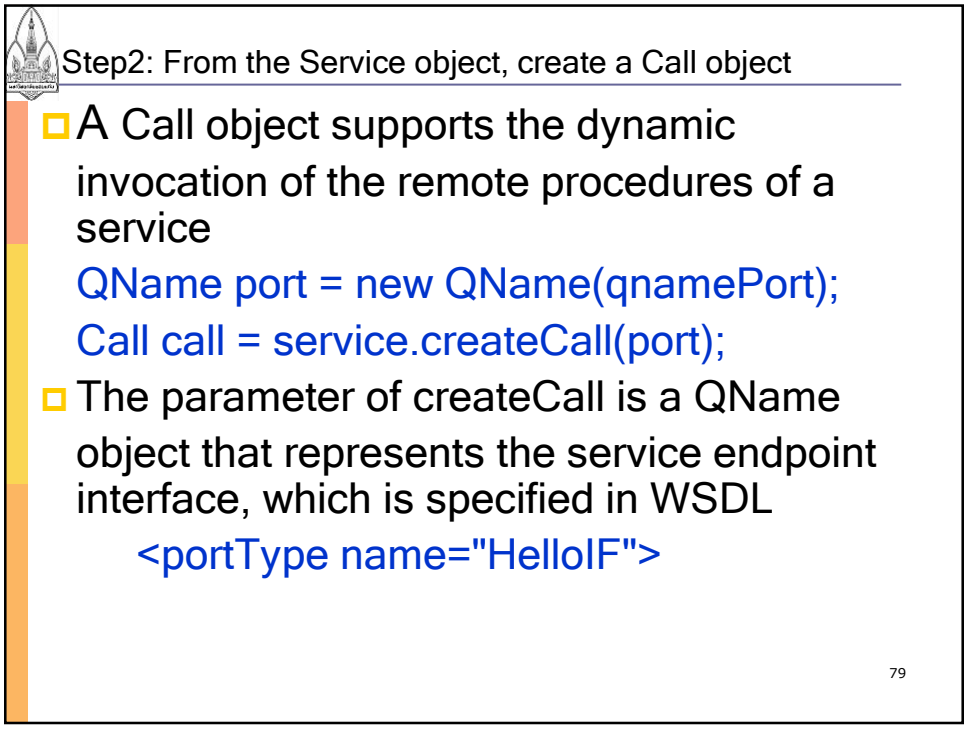
```
Command Prompt

C:\Sun\j2eetutorial14\examples\jaxrpc\dynamicproxy>asant run
-client
Buildfile: build.xml

run-client:
     [java] UrlString = http://localhost:8080/hello-jaxrpc/h
ello?WSDL
     [java] Hello Buzz

BUILD SUCCESSFUL
Total time: 3 seconds

C:\Sun\j2eetutorial14\examples\jaxrpc\dynamicproxy>
```

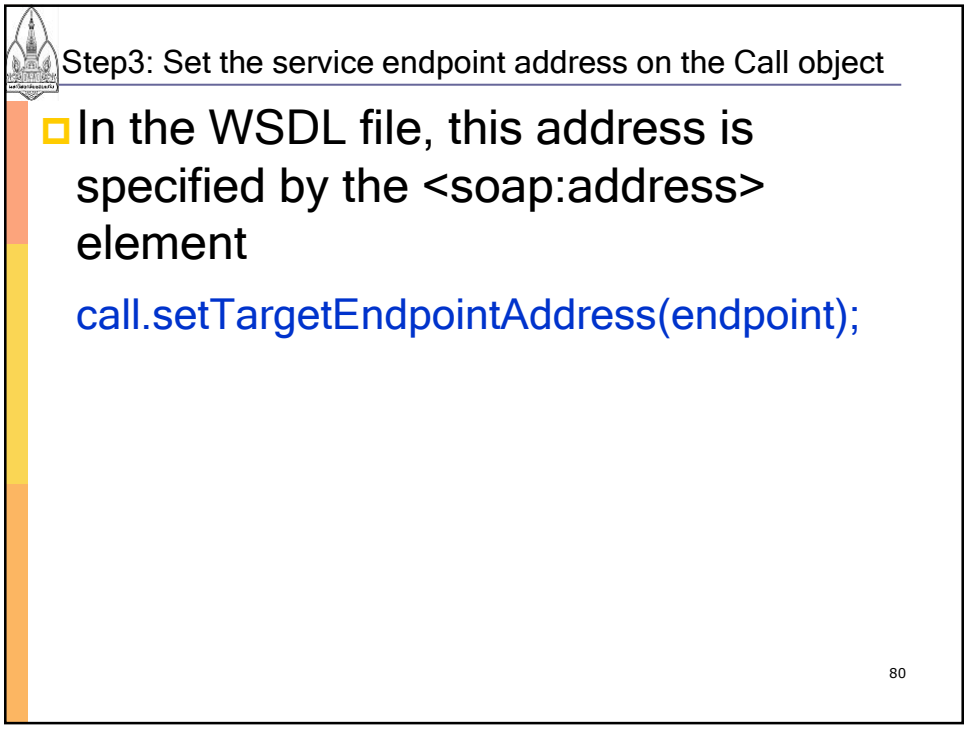

DII Invocation Model (1/2)
Gives complete control to client programmer
A client can call a remote procedure even if the signature of the remote procedure or the name of the service are unknown until runtime
Does not require wscompile to create runtime classes
Most complex programming among the three
75


DII Invocation Model (2/2)
Enables broker model
Client finds (through some search criteria) and invokes a service during runtime through a broker
Used when service definition interface is not known until runtime
You set operation and parameters during runtime
76


Steps of Coding DII Client
1.Create a Service object
2.From the Service object, create a Call object
3.Set the service endpoint address on the Call object
4.Set properties on the Call object
5.Specify the method's return type, name, and parameter
6.Invoke the remote method on the Call object
77


Step1: Create a Service object
Invoke createService() method of a ServiceFactory object
Service service =
factory.createService(new QName(qnameService));
qnameService parameter is the name of the service specified in WSDL
<service name="MyHelloService">
78


Step2: From the Service object, create a Call object
A Call object supports the dynamic invocation of the remote procedures of a service
QName port = new QName(qnamePort);
Call call = service.createCall(port);
The parameter of createCall is a QName object that represents the service endpoint interface, which is specified in WSDL
<portType name="HelloIF">
79


Step3: Set the service endpoint address on the Call object
In the WSDL file, this address is specified by the <soap:address> element
call.setTargetEndpointAddress(endpoint);
80

## Step4: Specify the method's return type, name, and parameter

- Properties to set
  - SOAPACTION_USE_PROPERTY
  - SOAPACTION_URI_PROPERTY
  - ENCODING_STYLE_PROPERTY

81

## Step5: Specify the method's return type, name, and parameter

- Return type, method name, parameter

```
QName QNAME_TYPE_STRING = new
    QName(NS_XSD, "string");
call.setReturnType(QNAME_TYPE_STRING);

call.setOperationName(new
QName(BODY_NAMESPACE_VALUE,
"sayHello"));

call.addParameter("String_1",
QNAME_TYPE_STRING,
ParameterMode.IN)
```

82

## Step6: Invoke the remote method on the Call object

- Assign the parameter value (Murphy) to a String array (params) and then executes the invoke method with the String array as an argument

```
String[] params = { "Murphy" };
String result = (String)call.invoke(params);
```

83

## Example: DII Client (1/3)

```
package dii;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.xml.rpc.ServiceFactory;
import javax.xml.rpc.ParameterMode;
public class HelloClient {
    private static String qnameService = "MyHelloService";
    private static String qnamePort = "HelloIF";
    private static String BODY_NAMESPACE_VALUE =
    "urn:Foo";
    private static String ENCODING_STYLE_PROPERTY =
    "javax.xml.rpc.encodingstyle.namespace.uri";
    private static String NS_XSD =
    "http://www.w3.org/2001/XMLSchema";
    private static String URI_ENCODING =
    "http://schemas.xmlsoap.org/soap/encoding/";
```

84

## Example: DII Client (2/3)

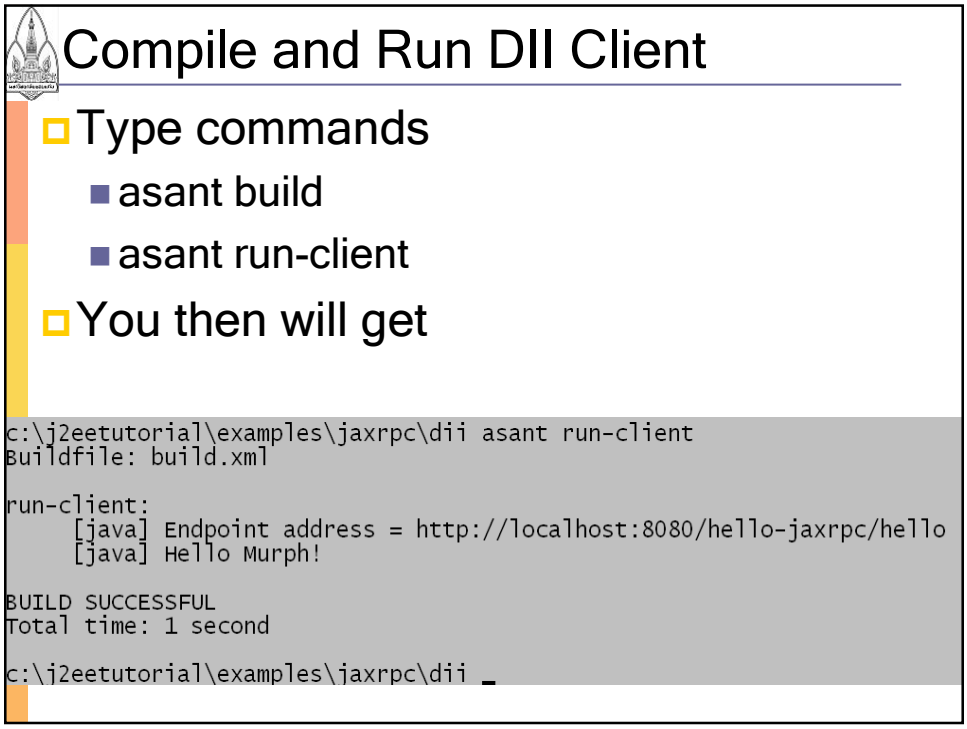
```
public static void main(String[] args) {
    System.out.println("Endpoint address = " + args[0]);
    try {
        ServiceFactory factory =
                ServiceFactory.newInstance();
        Service service =
                factory.createService(
                new QName(qnameService));
        QName port = new QName(qnamePort);
        Call call = service.createCall(port);
        call.setTargetEndpointAddress(args[0]);
        call.setProperty(Call.SOAPACTION_USE_PROPERTY,
                new Boolean(true));
        call.setProperty(Call.SOAPACTION_URI_PROPERTY, "");
        call.setProperty(ENCODING_STYLE_PROPERTY,
                URI_ENCODING);
```

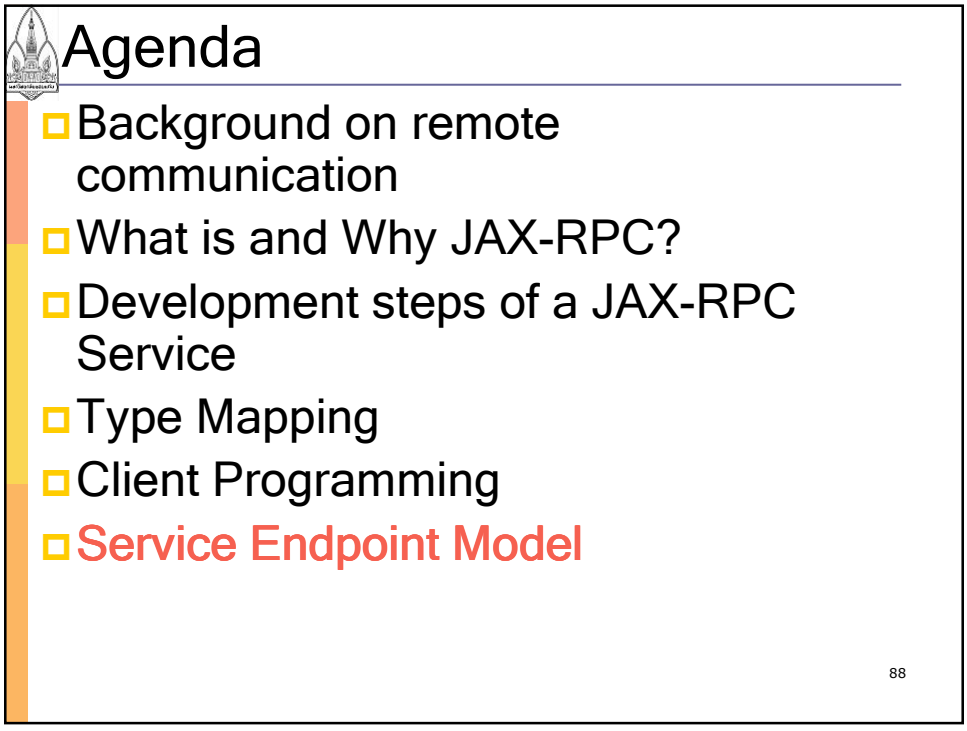
85

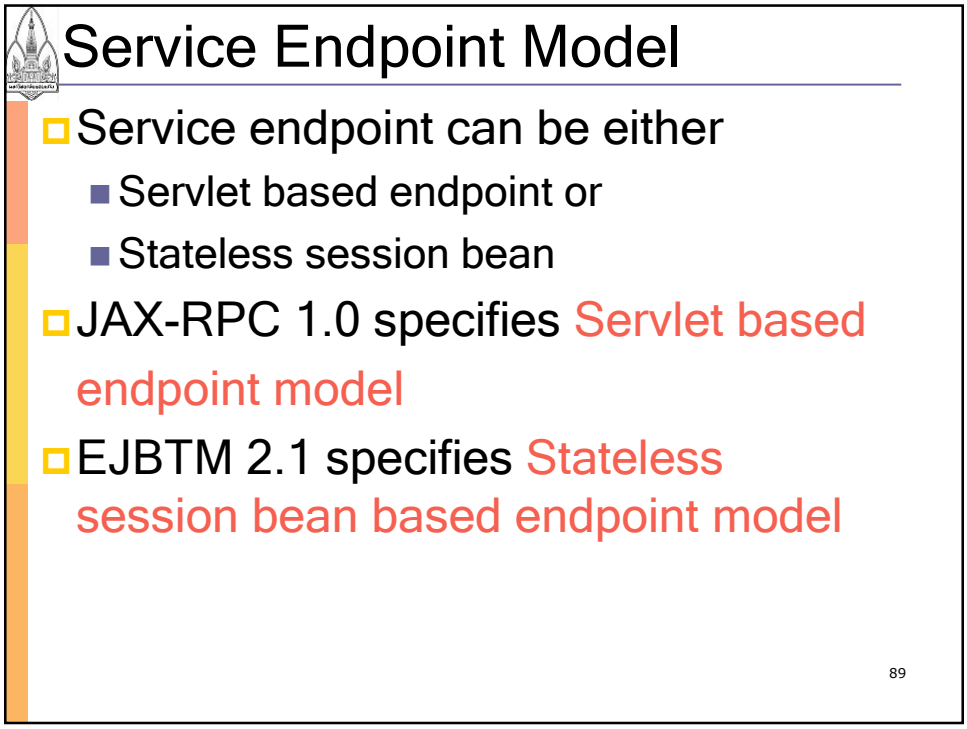
## Example: DII Client (3/3)

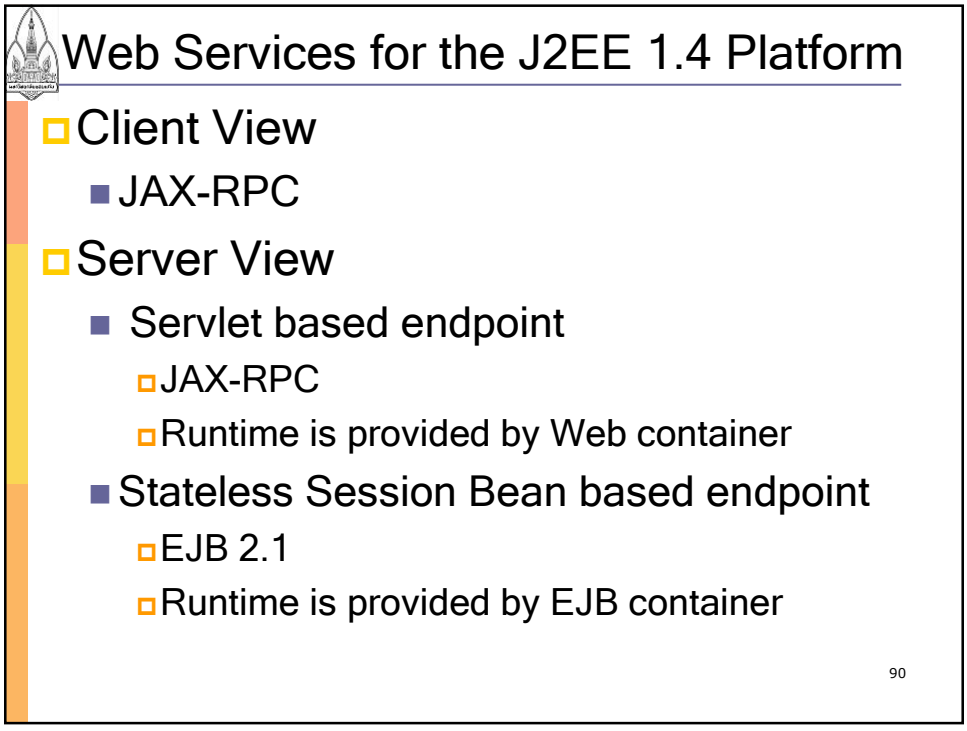
```
        QName QNAME_TYPE_STRING =
                new QName(NS_XSD, "string");
        call.setReturnType(QNAME_TYPE_STRING);
        call.setOperationName(
        new
        QName(BODY_NAMESPACE_VALUE,"sayHello"));
        call.addParameter("String_1",
                QNAME_TYPE_STRING,
                ParameterMode.IN);
        String[] params = { "Murph!" };
        String result = (String)call.invoke(params);
        System.out.println(result);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

86


Compile and Run DII Client
Type commands
asant build
asant run-client
You then will get
c:\j2eetutorial\examples\jaxrpc\dii asant run-client
Buildfile: build.xml
run-client:
[java] Endpoint address = http://localhost:8080/hello-jaxrpc/hello
[java] Hello Murph!
BUILD SUCCESSFUL
Total time: 1 second
c:\j2eetutorial\examples\jaxrpc\dii _


Agenda
Background on remote communication
What is and Why JAX-RPC?
Development steps of a JAX-RPC Service
Type Mapping
Client Programming
Service Endpoint Model
88


Service Endpoint Model
Service endpoint can be either
Servlet based endpoint or
Stateless session bean
JAX-RPC 1.0 specifies Servlet based endpoint model
EJBTM 2.1 specifies Stateless session bean based endpoint model
89


Web Services for the J2EE 1.4 Platform
Client View
JAX-RPC
Server View
Servlet based endpoint
JAX-RPC
Runtime is provided by Web container
Stateless Session Bean based endpoint
EJB 2.1
Runtime is provided by EJB container
90

## Service Developer

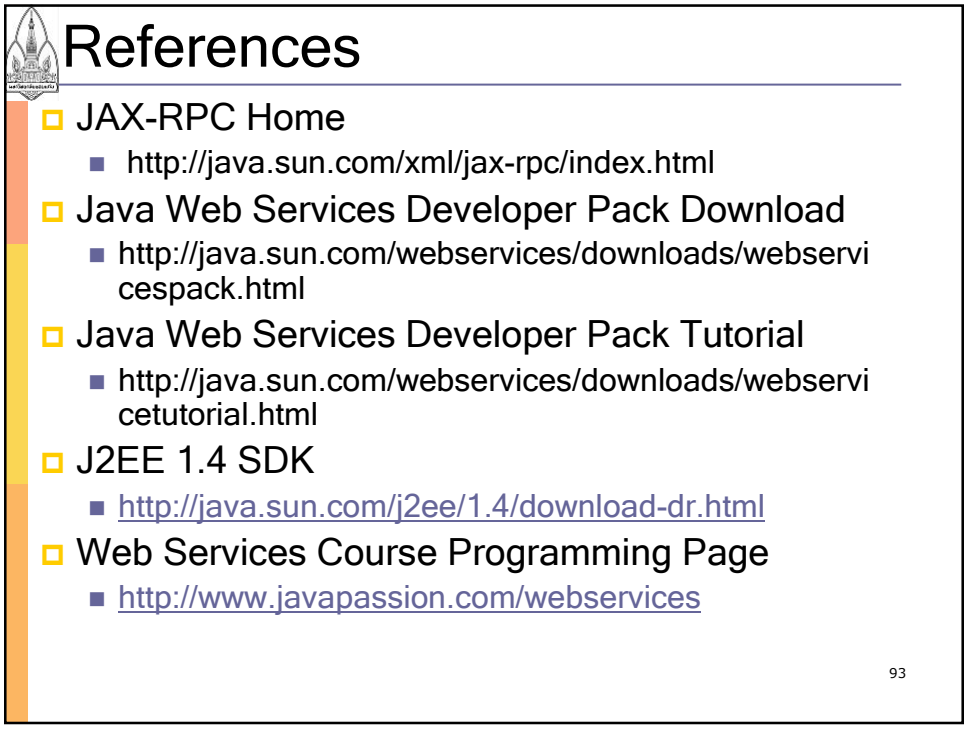
- Generates service definition interface
- Implements service definition interface (service endpoint class)
- Service endpoint class
  - May implement *ServiceLifecycle* interface
  - Can access servlet context via *javax.servlet.ServletContext* interface
- Creates **.war* package

91

## Service Deployer

- Handles
  - Protocol binding
  - Port assignment
- Multiple protocol binds for a single service endpoint definition
- Creates *web.xml*
- Creates complete WSDL document which contains concrete binding information
  - This WSDL document can be published
- Creates and deploys assembled **.war* file

92


References
JAX-RPC Home
http://java.sun.com/xml/jax-rpc/index.html
Java Web Services Developer Pack Download
http://java.sun.com/webservices/downloads/webservicespack.html
Java Web Services Developer Pack Tutorial
http://java.sun.com/webservices/downloads/webservicetutorial.html
J2EE 1.4 SDK
http://java.sun.com/j2ee/1.4/download-dr.html
Web Services Course Programming Page
http://www.javapassion.com/webservices
93