# Architecture of Enterprise Applications 16 Web Services and SOAP

**Haopeng Chen**

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*

Shanghai Jiao Tong University

Shanghai, China

http://reins.se.sjtu.edu.cn/~chenhp

e-mail: chen-hp@sjtu.edu.cn

# Contents



- Web Service Standards
  - SOAP
  - WSDL
  - UDDI
- Web Services Implementation
  - Java EE Web Services
- Web Services Discovery

# Web Services Overview

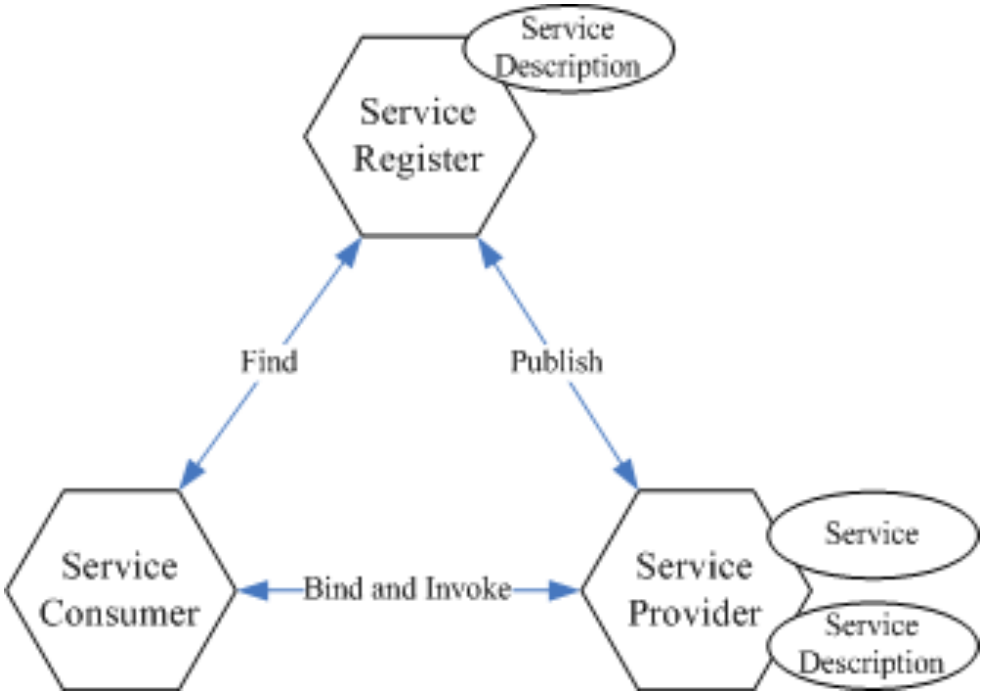


- Web Services
  - present the opportunity for real interoperability across hardware, operating systems, programming languages, and applications.
- Web
  - Access with web protocols
- Services
  - Independent of the implementation

# Web Services Overview



- A web service is a remote application
  - described using the Web Service Description Language (WSDL)
  - and accessed using the Simple Object Access Protocol (SOAP)
  - according to the rules defined by the WS-I Basic Profile 1.1.



# SOAP



- SOAP is defined by its own XML Schema and relies heavily on the use of XML Namespaces.
  - Here's a SOAP request message that might be sent from a client to a server:

```
<?xml version='1.0' encoding='UTF-8' ?>
<env:Envelope
          xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
    <env:Header />
    <env:Body>
        <reservation xmlns="http://www.titan.com/Reservation">
            <customer>
                <!-- customer info goes here -->
            </customer>
        </reservation>
    </env:Body>
</env:Envelope>
```

# WSDL



- Imagine that you want to develop a web services component that implements the following interface:

```
public interface TravelAgent {
    public String makeReservation(int cruiseID,
            int cabinID, int customerId, double price);
}
```

# WSDL



- A WSDL document that describes the `makeReservation( )` method might look like this:

```
<?xml version="1.0"?>
 <definitions name="TravelAgent"
       xmlns="http://schemas.xmlsoap.org/wsdl/"
       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:titan="http://www.titan.com/TravelAgent"
       targetNamespace="http://www.titan.com/TravelAgent">
 <!-- message elements describe the parameters and return values -->
 <message name="RequestMessage">
        <part name="cruiseId" type="xsd:int" />
        <part name="cabinId" type="xsd:int" />
        <part name="customerId" type="xsd:int" />
        <part name="price" type="xsd:double" />
 </message>
 <message name="ResponseMessage">
        <part name="reservationId" type="xsd:string" />
 </message>
```

# WSDL



```
<!-- portType element describes the abstract interface of a web service -->
<portType name="TravelAgent">
   <operation name="makeReservation">
      <input message="titan:RequestMessage"/>
      <output message="titan:ResponseMessage"/>
   </operation>
</portType>
<!--binding element tells us which protocols and encoding styles are used -->
<binding name="TravelAgentBinding" type="titan:TravelAgent">
 <soap:binding style="rpc“ transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="makeReservation">
     <soap:operation soapAction="" />
     <input>
      <soap:body use="literal" namespace="http://www.titan.com/TravelAgent"/>
     </input>
     <output>
      <soap:body use="literal" namespace="http://www.titan.com/TravelAgent"/>
     </output>
 </operation>
</binding>
```

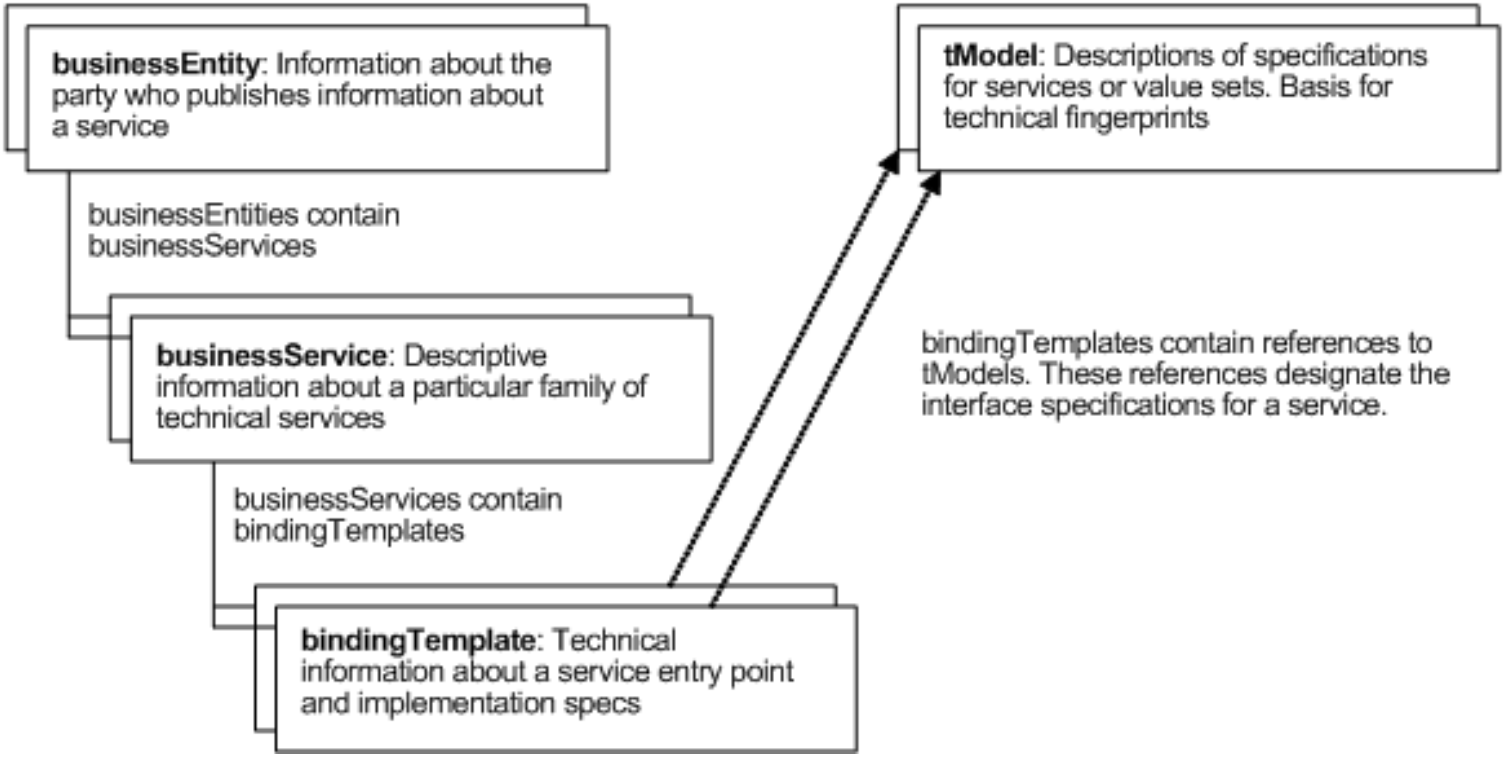
# WSDL

```
  <!-- service element tells us the Internet address of a web service -->
  <service name="TravelAgentService">
    <port name="TravelAgentPort" binding="titan:TravelAgentBinding">
      <soap:address location="http://www.titan.com/webservices/TravelAgent" />
    </port>
  </service>
</definitions>
```

# UDDI



- One or more UDDI nodes may be combined to form a UDDI Registry. The nodes in a UDDI registry collectively manage a particular set of UDDI data. This data is distinguished by the visible behavior associated with the entities contained in it.



# UDDI



- The entities businessEntity, businessService, bindingTemplate, tModel form the core data structures of UDDI. Within a registry, each instance of the core data structures is uniquely identified by a UDDI key.
  - Find Web services implementations that are based on a common abstract interface definition.
  - Find Web services providers that are classified according to a known classification scheme or identifier system.
  - Determine the security and transport protocols supported by a given Web service.
  - Issue a search for services based on a general keyword.
  - Cache the technical information about a Web service and then update that information at run-time.

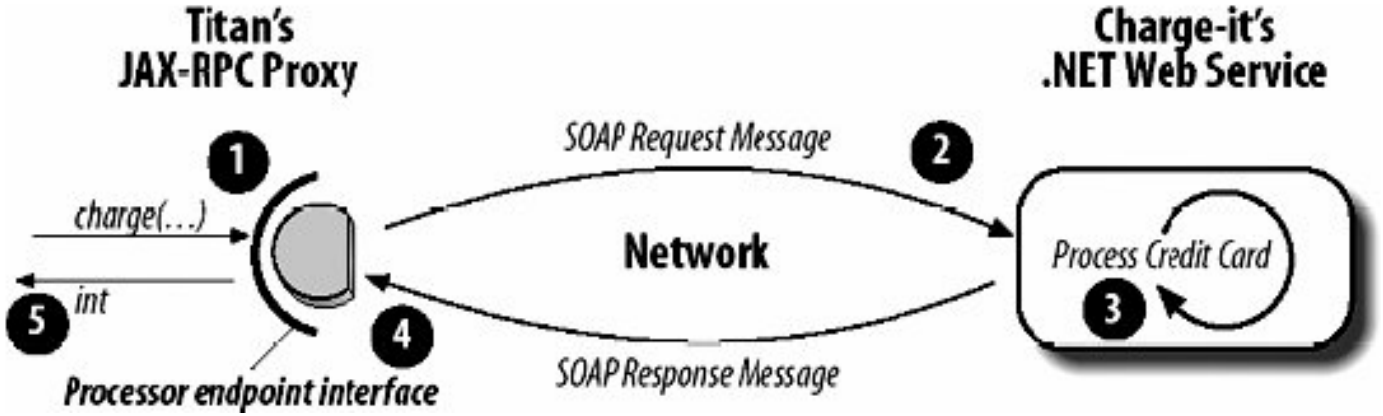
# UDDI



- Node API Sets
  - UDDI Inquiry
  - UDDI Publication
  - UDDI Security
  - UDDI Custody Transfer
  - UDDI Subscription
  - UDDI Replication
- Client API Sets
  - UDDI Subscription Listener
  - UDDI Value Set

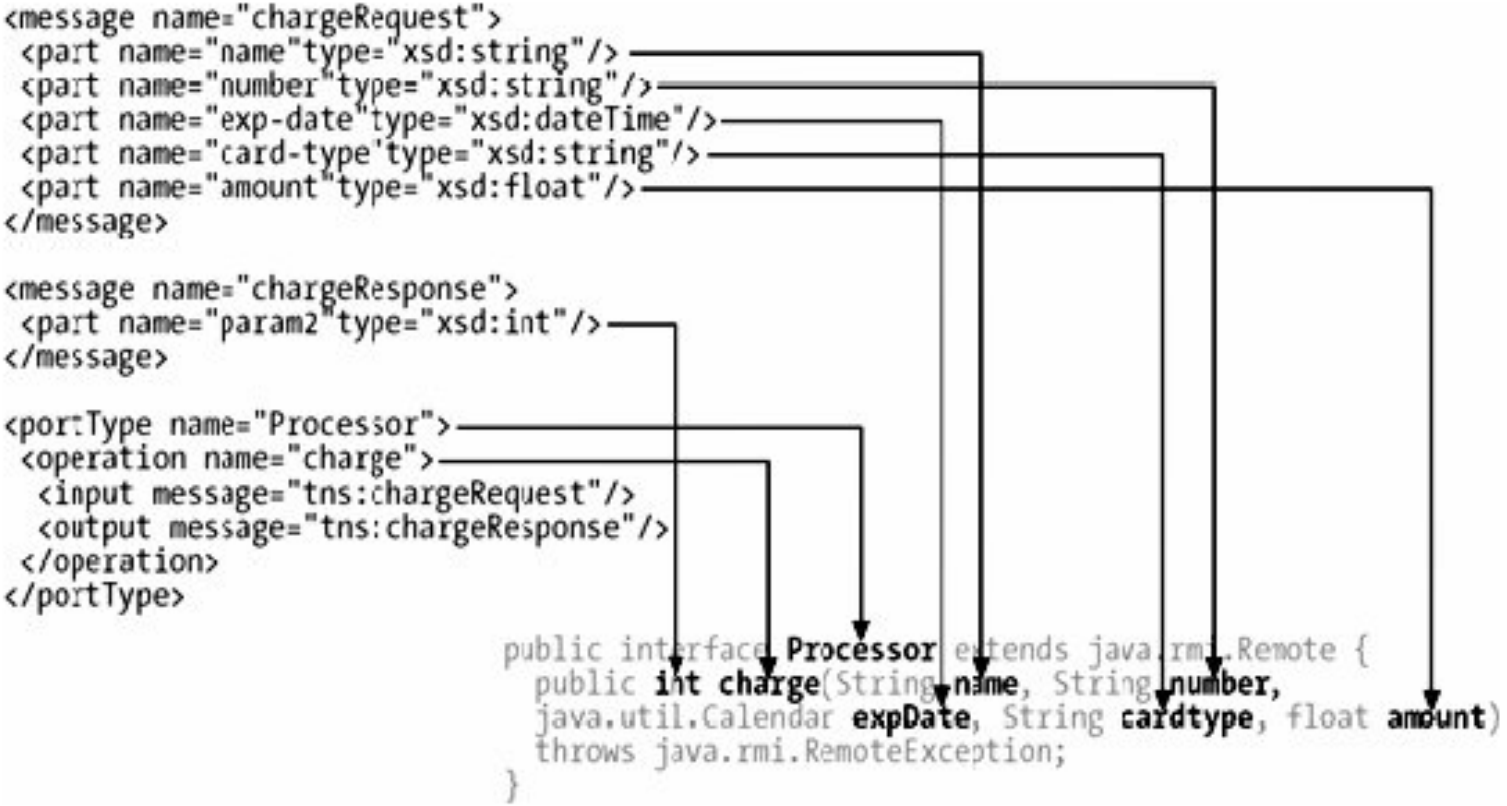
# Accessing Web Service





# Accessing Web Service





# Accessing Web Service – JAX-RPC



```
 package com.charge_it;

public interface ProcessorService extends javax.xml.rpc.Service {
  public com.charge_it.Processor getProcessorPort( ) throws
         javax.xml.rpc.ServiceException;
  public java.lang.String getProcessorPortAddress( );
  public com.charge_it.Processor
         getProcessorPort (java.net.URL portAddress)
         throws  javax.xml.rpc.ServiceException;
}
```

# Accessing Web Service – JAX-RPC



```
package com.titan.travelagent;
import com.charge_it.Processor;
import com.charge_it.ProcessorService;
...
@Stateful
public class TravelAgentBean implements TravelAgentRemote {
  @PersistenceContext(unitName="titanDB")
  private EntityManager em;

  @PersistenceContext
  EntityManager em;

  Customer customer;
  Cruise cruise;
  private Cabin cabin;
  private ProcessorService processorService;
  …
```

# Accessing Web Service – JAX-RPC



```
public TicketDO bookPassage(CreditCardDO card, double price) throws
       IncompleteConversationalState {
        if (customer == null || cruise == null || cabin == null) {
            throw new IncompleteConversationalState( ); }
        try {
          Reservation reservation = new Reservation( customer,
                                          cruise, cabin, price, new Date( ));
          em.persist(reservation);
          String customerName = customer.getFirstName( )+" “
                                                  + customer.getLastName( );
          java.util.Calendar expDate = new Calendar(card.date);
          Processor processor = processorService.getProcessorPort( );
          processor.charge(customerName, card.number, expDate, card.type, price);
            TicketDO ticket = new TicketDO(customer, cruise, cabin, price);
            return ticket;
        } catch(Exception e) { throw new EJBException(e); }
    } ...
  }
```

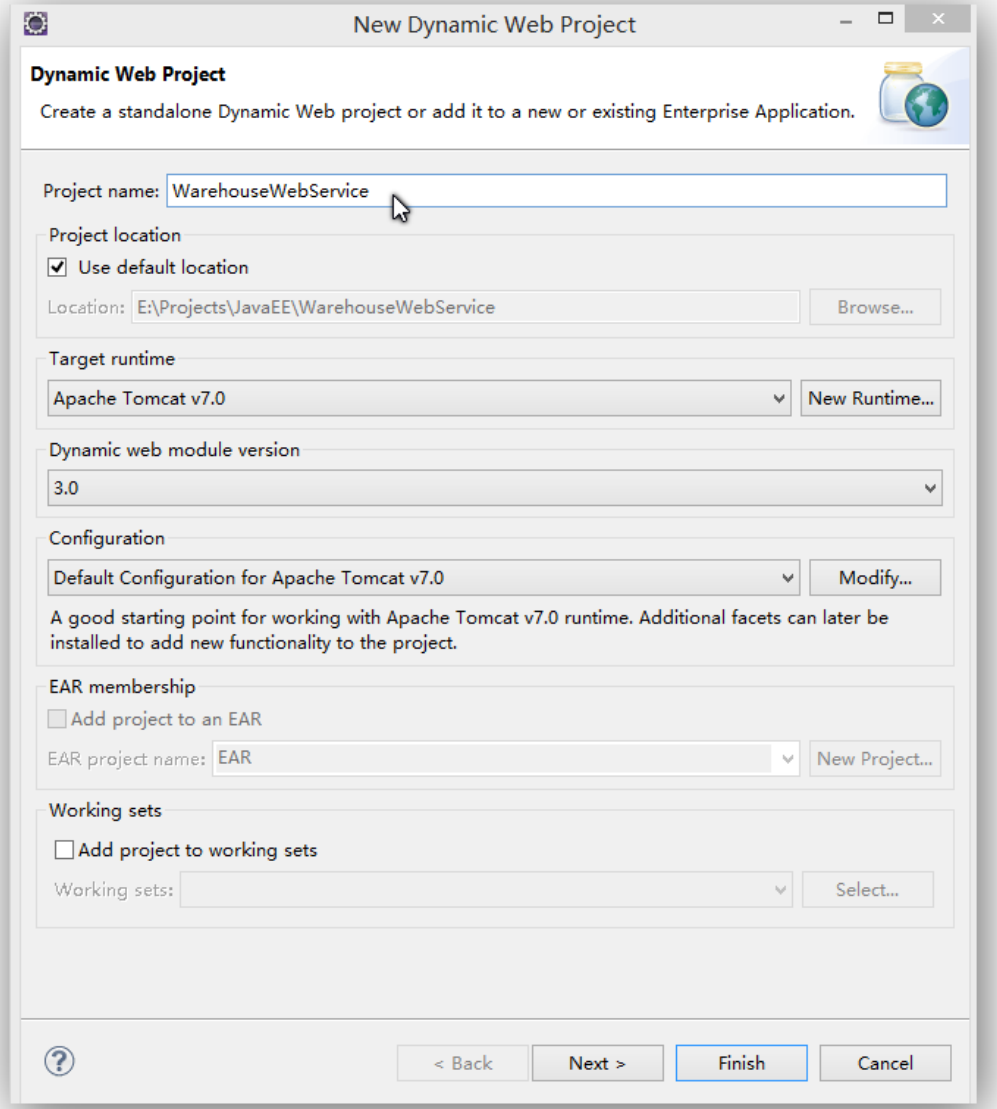
# Accessing Web Service – JAX-WS



```
package com.titan.webservice;
import javax.ejb.Stateless;
import javax.jws.WebService;
import javax.jws.WebMethod;

@Stateless
@WebService
public class TravelAgentBean {
  @WebMethod
  public String makeReservation(int cruiseId, int cabinId, int
                                customerId, double price) {
    ...
  }
}
```

# Web Services





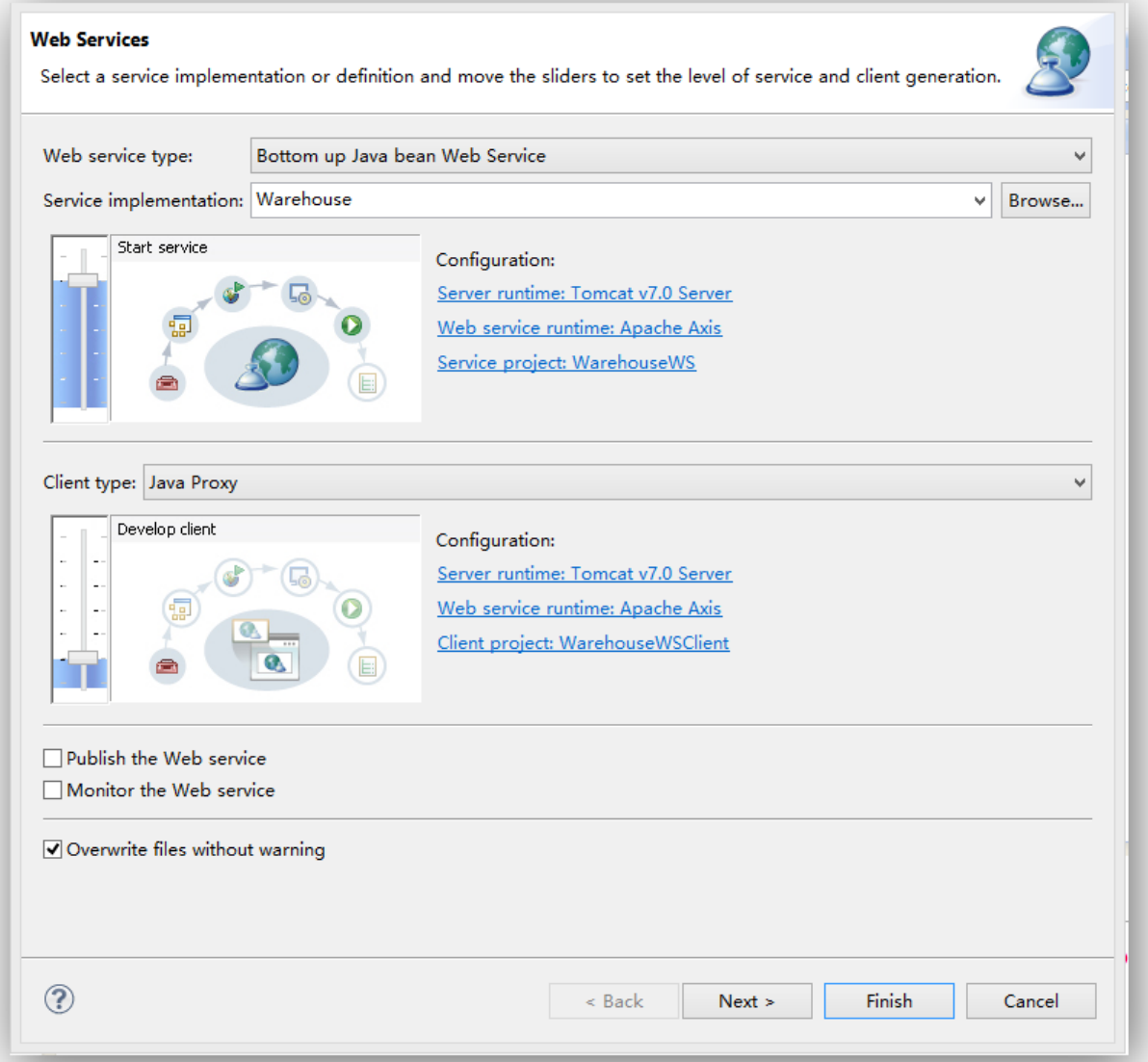
# Web Services



```
import java.util.*;
import javax.jws.*;

@WebService
public class Warehouse {
   public Warehouse() {
      prices = new HashMap<String, Double>();
      prices.put("Blackwell Toaster", 24.95);
      prices.put("ZapXpress Microwave Oven", 49.95);
   }

   public double getPrice(@WebParam(name="description") String description)
   {
      Double price = prices.get(description);
      return price == null ? 0 : price;
   }

   private Map<String, Double> prices;
}
```

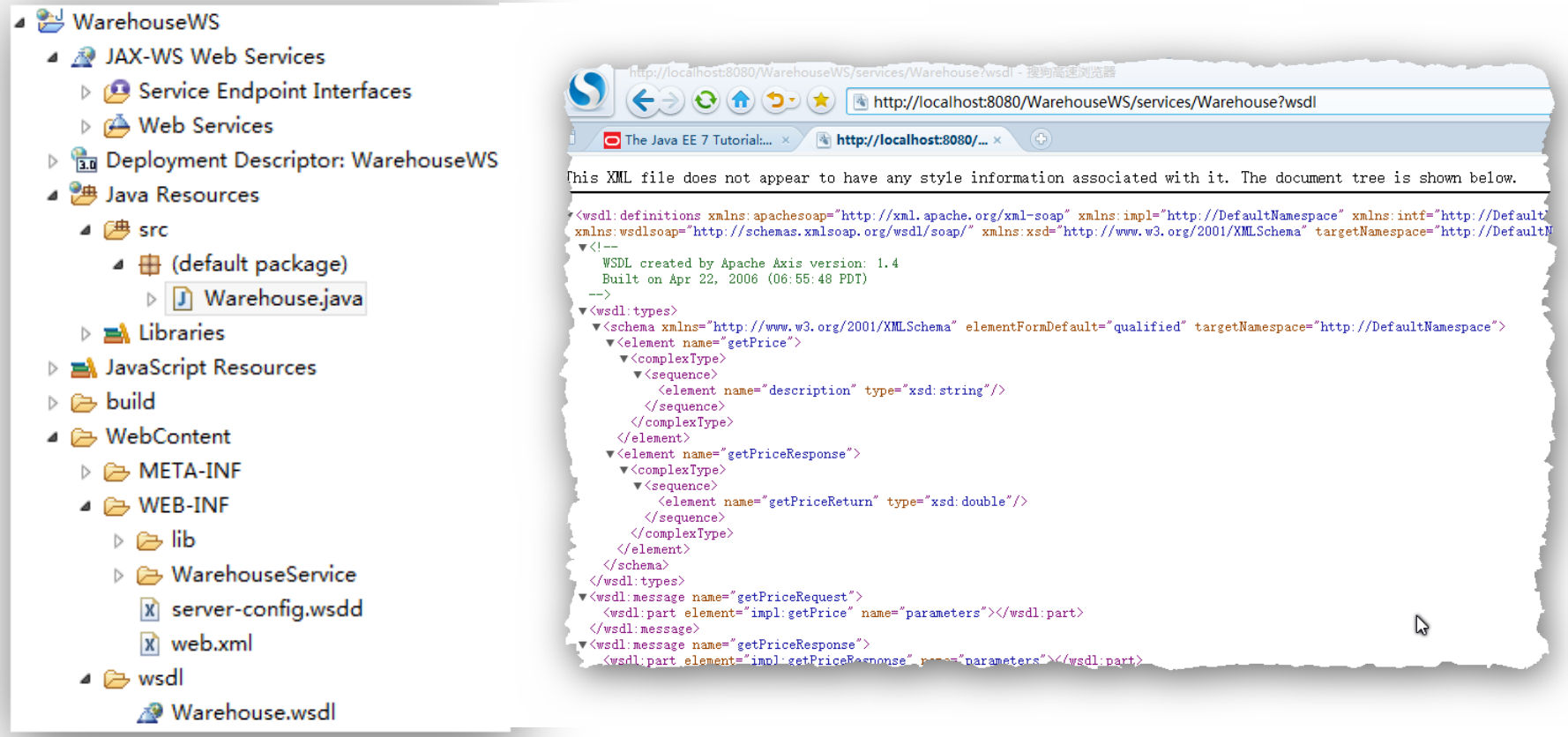
# Web Services



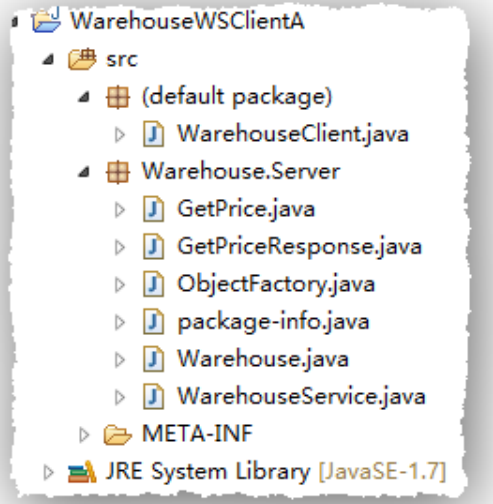


# Web Services





# Web Service Client - A




- Create a plain java project

- Generate the necessary files for client and add them to the project

```
wsimport -keep -p warehouse.server
http://localhost:8080/WebServices/warehouse?wsdl
```

- Write a Web Service Client

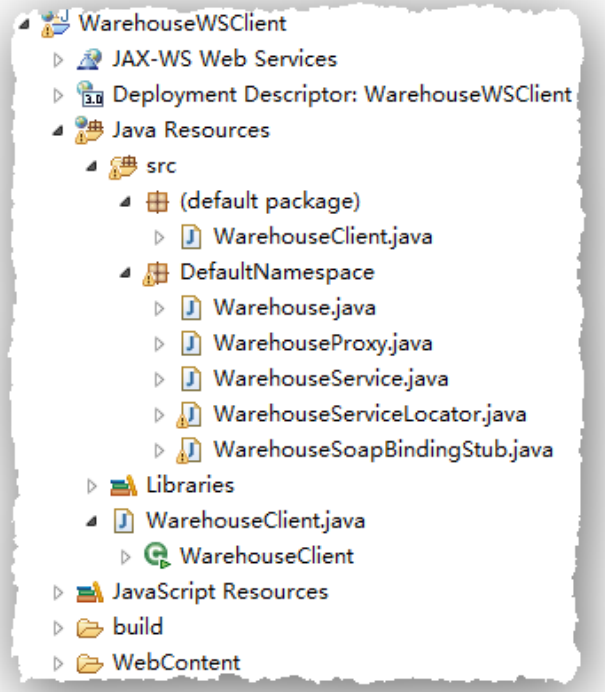
```
public class WarehouseClient
{
   public static void main(String[] args) throws NamingException, RemoteException
   {
      WarehouseService service = new WarehouseService();
      Warehouse port = service.getPort(Warehouse.class);

      String descr = "ZapXpress Microwave Oven";
      double price = port.getPrice(descr);
      System.out.println(descr + ": " + price);
   }
}
```

# Web Service Client - B



- Use the generated Client

```
public class WarehouseClient
{
    public static void main(String[] args) throws NamingException, RemoteException
    {
      WarehouseServiceLocator locator = new WarehouseServiceLocator();
      Warehouse warehouse = null;
      try{
          warehouse = locator.getWarehouse();
      }catch(Exception e){};

       String descr = "Blackwell Toaster";
       double price = warehouse.getPrice(descr);
       System.out.println(descr + ": " + price);
    }
}
```



# WCF

- Windows Communication Foundation
  - .Net Remoting
  - ASMX
  - WSE
  - MSMQ
- SOAP -based Web Service vs. REST-based Web Service
- WS-Security, WS-ReliableMessaging, and WS-AtomicTransaction

# Develop a Web Service



- Process
  - Create a Class extended from SOAPheader, which receives the messages in SOAP header
  - Add a method into Web service Class on server-side and Proxy on client-side
  - Add attribute SoapHeaderAttribute into Web service Class on server-side and Proxy on client-side
  - Set attributes of SOAPheader. Server-side will check them and decide whether execute the invocation according to the result of validation

# Develop a Web Service



- MySOAPheader

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Web.Services;
using System.Web.Services.Protocols;
```

# Develop a Web Service



```
public class MySoapHeader : SoapHeader
{

   private string _token;
   public MySoapHeader()
   {
     // TODO: Add constructor logic here
   }

   public MySoapHeader(string sToken)
   {
         this._token = sToken;
    }

   public string Token
   {
        get { return this._token; }
        set { this._token = value; }
   }
}
```

# Develop a Web Service



- UserValidation

```
using System;
using System.Collections.Generic;
using System.Text;
namespace WebServiceUserValidation
{
   public class UserValidation
   {
          public UserValidation()
          {
          }
          public static bool IsUserLegal(string sName,string sPsw)
          {
               string psw ="FrankXuLei";
               if (string.Equals(psw, sPsw))
               {
                   return true;
               }
```

# Develop a Web Service



```
            else
            {
                return false;
            }
        }
        public static bool IsUserLegal(string sToken)
        {
            string psw = "FrankXuLei";
            if (string.Equals(psw, sToken))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
```

# Develop a Web Service



- Web Service

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using WebServiceUserValidation;

[WebService(Namespace = "http://www.cnblogs.com/frank_xl/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class FrankXuWebService : System.Web.Services.WebService
{
   public MySoapHeader _authenticationToken;
   private const string _token = "FrankXuLei";
       public FrankXuWebService()
   {
            //InitializeComponent();
    }
```

# Develop a Web Service



```
[SoapHeader("_authenticationToken")]
[WebMethod(EnableSession=false)]
public string HelloFrank()
{
     if (_authenticationToken != null && UserValidation.IsUserLeg
al(_authenticationToken.Token ))
    {
          return "Hello Frank,WebMethod is called sucessfully";
     }
    else
    {
         throw new Exception("Authentication Failed");
     }
  }
}
```

# Develop a Web Service



- Web Service Client

```
namespace ConsoleWebServiceClient
{
   class Program
   {
        static void Main(string[] args)
        {
               localhost.MySoapHeader mySoapHeader =
                              new ConsoleWebServiceClient.
                                  localhost.MySoapHeader();
               mySoapHeader.Token = "FrankXuLei";
               string sResult = string.Empty;
               localhost.FrankXuWebService frankXuWebService = null;
               try
               {
                    frankXuWebService =
                          new ConsoleWebServiceClient.
                              localhost.FrankXuWebService();
                   frankXuWebService.MySoapHeaderValue = mySoapHeader;
                   sResult = frankXuWebService.HelloFrank();
```

# Develop a Web Service



```
            Console.WriteLine(sResult);
        }
        catch (Exception ex)
        {
            Console.WriteLine("Call WebService is failed");
            throw ex;
        }
        finally
        {
            if (frankXuWebService != null)
                frankXuWebService.Dispose();
        }
        Console.WriteLine("Press any key to continue");
        Console.ReadLine();
    }
  }
}
```

# Project



- Requirement
  - To wrap a query API as a Web Service to facilitate the interaction with heterogeneous applications.
  - You also need to develop a web service client to validate your web service.

# References



- Core Java (volume II) 9th edition
  - http://horstmann.com/corejava.html
- The Java EE 7 Tutorial
  - http://docs.oracle.com/javaee/7/tutorial/doc/javaeetutorial7.pdf

# Thank You!