

Services Web – SOAP en Java

Fabrice Rossi

<http://apiacoa.org/contact.html>.

Université Paris-IX Dauphine

Services Web – SOAP en Java – p.1/71

Outils utilisés

- API Java : JAX-RPC (XML RPC), <http://java.sun.com/xml/jaxrpc/>, sera intégrée à J2EE 1.4
- implémentations :
 - le Java Web Services Developer Pack 1.1 <http://java.sun.com/webservices/>
 - AXIS
 - <http://ws.apache.org/axis/>
 - implémentation open source du groupe apache
 - descendant de Apache SOAP 2.x et de IBM SOAP4J

Dans ce cours : AXIS!

Services Web – SOAP en Java – p.3/71

Plan du cours SOAP en Java

1. outils
2. programmation d'un client :
 - (a) principe de JAX-RPC
 - (b) exemples (trois types de clients)
 - (c) types Java et types SOAP
 - (d) traduction d'un service complet
 - (e) gestion des erreurs
3. programmation d'un service (avec Axis) :
 - (a) déploiement instantané
 - (b) déploiement adapté
 - (c) EJB

Services Web – SOAP en Java – p.2/71

Rôle de JAX-RPC

Rendre les RPC basés sur XML presque aussi simples que RMI :

- un service web devient une **interface** héritant de Remote
- une opération devient une **méthode**
- une erreur SOAP (Fault) devient une **exception** héritant de RemoteException
- JAX-RPC définit un ensemble de traduction :
 - faire le lien entre les types Java et les types des schémas du W3
 - représenter informatiquement les concepts de WSDL (service, port, etc.)
 - etc.

Services Web – SOAP en Java – p.4/71

Exemple de base

Service web minimaliste : Echo proposant une opération echo

Echo.wsdl

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <wsdl:definitions
3   targetNamespace="http://localhost:8080/axis/services/Echo"
4   xmlns="http://schemas.xmlsoap.org/wsdl/"
5   xmlns:apache="http://xml.apache.org/xml-soap"
6   xmlns:impl="http://localhost:8080/axis/services/Echo"
7   xmlns:intf="http://localhost:8080/axis/services/Echo"
8   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
9   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
10  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
11  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
12  <wsdl:message name="echoResponse">
13    <wsdl:part name="echoReturn" type="xsd:string"/>
14  </wsdl:message>
15  <wsdl:message name="echoRequest">
16    <wsdl:part name="in0" type="xsd:string"/>
17  </wsdl:message>
18  <wsdl:portType name="Echo">
19    <wsdl:operation name="echo" parameterOrder="in0">
20      <wsdl:input message="impl:echoRequest" name="echoRequest"/>
21      <wsdl:output message="impl:echoResponse" name="echoResponse"/>
22    </wsdl:operation>
23  </wsdl:portType>
```

Services Web – SOAP en Java – p.5/71

Exemple de base (2)

Echo.wsdl

```
24 <wsdl:binding name="EchoSoapBinding" type="impl:Echo">
25   <wsdlsoap:binding style="rpc"
26     transport="http://schemas.xmlsoap.org/soap/http"/>
27   <wsdl:operation name="echo">
28     <wsdlsoap:operation soapAction=""/>
29     <wsdl:input name="echoRequest">
30       <wsdlsoap:body
31         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
32         namespace="http://localhost:8080/axis/services/Echo"
33         use="encoded"/>
34     </wsdl:input>
35     <wsdl:output name="echoResponse">
36       <wsdlsoap:body
37         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
38         namespace="http://localhost:8080/axis/services/Echo"
39         use="encoded"/>
40     </wsdl:output>
41   </wsdl:operation>
42 </wsdl:binding>
43 <wsdl:service name="EchoService">
44   <wsdl:port binding="impl:EchoSoapBinding" name="Echo">
45     <wsdlsoap:address location="http://localhost:8080/axis/services/Echo"/>
46   </wsdl:port>
47 </wsdl:service>
48 </wsdl:definitions>
```

Services Web – SOAP en Java – p.6/71

Client magique (AXIS)

- on récupère Echo.wsdl
- on lance `java org.apache.axis.wsdl.WSDL2Java Echo.wsdl` et on obtient 4 classes java (package `localhost.axis.services.Echo`):
 - Echo : interface (au sens java) du port
 - EchoService : interface de localisation du port (utilise abstraite au sens du *design pattern Abstract Factory*)
 - EchoServiceLocator : implémentation concrète de EchoService
 - EchoSoapBindingStub : *stub* pour traduire les appels locaux en messages SOAP et vice-versa (implémente l'interface Echo)
- il ne reste plus qu'à coder le client

Services Web – SOAP en Java – p.7/71

Interfaces

Echo.java

```
1 package localhost.axis.services.Echo;
2
3 public interface Echo extends java.rmi.Remote {
4   public java.lang.String echo(java.lang.String in0)
5     throws java.rmi.RemoteException;
6 }
```

EchoService.java

```
1 package localhost.axis.services.Echo;
2
3 public interface EchoService extends javax.xml.rpc.Service {
4   public java.lang.String getEchoAddress();
5
6   public localhost.axis.services.Echo.Echo getEcho()
7     throws javax.xml.rpc.ServiceException;
8
9   public localhost.axis.services.Echo.Echo getEcho(java.net.URL portAddress)
10     throws javax.xml.rpc.ServiceException;
11 }
```

Services Web – SOAP en Java – p.8/71

Le client

```
Client.java
1 import localhost.axis.services.Echo.*;
2 public class Client {
3     public static void main(String[] args) throws Exception {
4         EchoService service=new EchoServiceLocator();
5         Echo port=service.getEcho();
6         System.out.println(port.echo(args[0]));
7     }
8 }
```

Ce client est (en partie) spécifique AXIS (ligne 4)

- la norme impose d'engendrer les interfaces EchoService et Echo, mais pas la classe EchoServiceLocator
- solution différente dans l'implémentation de Sun (le nom de la classe n'est pas le même)
- dans un contexte J2EE, on obtient une implémentation de EchoService grâce à l'annuaire J2EE

Message envoyé

```
echoRequest
1 POST /axis/services/Echo HTTP/1.0
2 Content-Type: text/xml; charset=utf-8
3 Accept: application/soap+xml, application/dime, multipart/related, text/*
4 User-Agent: Axis/1.1RC2
5 Host: 127.0.0.1
6 Cache-Control: no-cache
7 Pragma: no-cache
8 SOAPAction: ""
9 Content-Length: 455
10
11 <?xml version="1.0" encoding="UTF-8"?>
12 <soapenv:Envelope
13     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
14     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
15     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
16 <soapenv:Body>
17 <ns1:echo
18     soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
19     xmlns:ns1="http://localhost:8080/axis/services/Echo">
20 <in0 xsi:type="xsd:string">Essai</in0>
21 </ns1:echo>
22 </soapenv:Body>
23 </soapenv:Envelope>
```

Message reçu

```
echoAnswer
1 HTTP/1.1 200 OK
2 Content-Type: text/xml; charset=utf-8
3 Date: Thu, 03 Apr 2003 12:01:01 GMT
4 Server: Apache Coyote/1.0
5 Connection: close
6
7 <?xml version="1.0" encoding="UTF-8"?>
8 <soapenv:Envelope
9     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
10     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
11     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
12 <soapenv:Body>
13 <ns1:echoResponse
14     soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
15     xmlns:ns1="http://localhost:8080/axis/services/Echo">
16 <echoReturn xsi:type="xsd:string">Essai</echoReturn>
17 </ns1:echoResponse>
18 </soapenv:Body>
19 </soapenv:Envelope>
```

Version plus lourde

- la solution précédente est statique : il faut engendrer les classes Java puis faire l'appel
- aspect dynamique : appeler un service web sans le connaître avant le lancement du programme
- très lourd :
 - pas de gestion des types
 - pas d'interface
 - pas de gestion des espaces de noms
 - etc.
- on peut faire plus simple que la solution du programme qui suit, mais on devient spécifique AXIS

Code

```
ClientDynamique.java
1 import javax.xml.rpc.Call;
2 import javax.xml.rpc.Service;
3 import javax.xml.rpc.ServiceFactory;
4 import javax.xml.rpc.ParameterMode;
5 import javax.xml.namespace.QName;
6
7 public class ClientDynamique {
8     public static void main(String [] args) throws Exception {
9         String ns="http://localhost:8080/axis/services/Echo";
10        String endpoint="http://localhost:8080/axis/services/Echo";
11        QName serviceName=new QName(ns,"EchoService");
12        QName portName=new QName(ns,"Echo");
13        QName operationName=new QName(ns,"echo");
14        ServiceFactory factory=ServiceFactory.newInstance();
15        Service service=factory.createService(serviceName);
16        Call call=service.createCall(portName,operationName);
17        call.setTargetEndpointAddress(endpoint);
18        QName XSDString=new QName("http://www.w3.org/2001/XMLSchema","string");
19        call.addParameter("in0",XSDString,ParameterMode.IN);
20        call.setReturnType(XSDString);
21        String ret = (String) call.invoke( new String[] { "Essai" } );
22        System.out.println(ret);
23    }
24 }
```

Totalement portable et dynamique

Services Web – SOAP en Java – p.13/71

Solution intermédiaire

Mécanisme de proxy dynamique, basé sur le système de *dynamic proxy* introduit par le jdk 1.3 :

- un *dynamic proxy* est une classe qui implémente une ou plusieurs interfaces spécifiées **au moment de l'exécution**
- elle implémente l'interface *InvocationHandler* (package `java.lang.reflect`) :

```
1 public interface InvocationHandler {
2     public Object invoke(Object proxy,
3                           Method method,
4                           Object[] args);
5 }
```

- un appel à une méthode d'une interface implémentée par un *dynamic proxy* est transformé en un appel à la méthode `invoke`
- JAX-RPC propose d'engendrer dynamiquement un *stub* à partir du WSDL

Services Web – SOAP en Java – p.14/71

Code

```
ClientDynamiqueProxy.java
1 import localhost.axis.services.Echo.Echo;
2 import java.net.URL;
3 import javax.xml.rpc.Service;
4 import javax.xml.rpc.ServiceFactory;
5 import javax.xml.namespace.QName;
6
7 public class ClientDynamiqueProxy {
8     public static void main(String [] args) throws Exception {
9         String ns="http://localhost:8080/axis/services/Echo";
10        String wsdl="http://localhost:8080/axis/services/Echo?wsdl";
11        URL wsdlURL=new URL(wsdl);
12        QName serviceName=new QName(ns,"EchoService");
13        QName portName=new QName(ns,"Echo");
14        ServiceFactory factory=ServiceFactory.newInstance();
15        Service service=factory.createService(wsdlURL,serviceName);
16        Echo port=(Echo)service.getPort(portName,Echo.class);
17        System.out.println(port.echo("Essai"));
18    }
19 }
```

Services Web – SOAP en Java – p.15/71

Mapping types Java vers types W3C

- Difficulté principale par rapport à RMI
- Types supportés :
 - les types fondamentaux (par exemple `boolean` devient `xsd:boolean`)
 - les chaînes de caractères (`String` devient `xsd:string`)
 - les entiers et réels arbitraires (par exemple `BigDecimal` devient `xsd:decimal`)
 - les dates (`Calendar` devient `xsd:dateTime`)
 - le binaire sous la forme d'un tableau de `byte` (`byte[]`) qui devient `xsd:base64Binary` ou `xsd:hexBinary`
 - les types tableaux et les *JavaBeans* construits à partir des types fondamentaux
- certaines implémentations proposent des traductions d'autres types comme par exemple les *Collections*, mais on a alors de sérieux problèmes d'interopérabilité

Services Web – SOAP en Java – p.16/71

Exemple d'une struct

- une *struct* SOAP correspond à un *JavaBean*
- le nom du *bean* correspond à celui de la *struct*
- chaque champ de la *struct* correspond à une propriété du *bean*, c'est-à-dire à une paire :
 - setXxx pour fixer la valeur de propriété
 - getXxx pour obtenir la valeur de la propriété
- exemple :

- PhoneBookWS et son opération getPhoneNumber
- représentation d'une personne : prénom et nom
- WSDL :

```
1 <complexType name="Person">
2   <sequence>
3     <element name="first" nillable="true" type="xsd:string"/>
4     <element name="last" nillable="true" type="xsd:string"/>
5   </sequence>
6 </complexType>
```

Services Web – SOAP en Java – p.17/71

Exemple (2) : code engendré par Axis

```
1  /**
2   * Person.java
3   *
4   * This file was auto-generated from WSDL
5   * by the Apache Axis WSDL2Java emitter.
6   */
7
8  package PhoneBookWS;
9
10 public class Person implements java.io.Serializable {
11     private java.lang.String first;
12     private java.lang.String last;
13
14     public Person() {
15     }
16
17     public java.lang.String getFirst() {
18         return first;
19     }
20
21     public void setFirst(java.lang.String first) {
22         this.first = first;
23     }
24
25     public java.lang.String getLast() {
26         return last;
27     }
28 }
```

Services Web – SOAP en Java – p.18/71

Exemple (3) : code engendré par Axis

28
29
30
31
32

```
1  public void setLast(java.lang.String last) {
2      this.last = last;
3  }
```

- le reste de la classe est très complexe
- il est spécifique Axis
- le programmeur n'a pas besoin de le connaître

Message SOAP engendré (partie) :

```
1 <multiRef id="id0" soapenc:root="0"
2   soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
3   xsi:type="ns2:Person"
4   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
5   xmlns:ns2="urn:PhoneBookWS">
6   <first xsi:type="xsd:string">John</first>
7   <last xsi:type="xsd:string">Doe</last>
8 </multiRef>
```

Services Web – SOAP en Java – p.19/71

Exemple de client basique

```
1  import PhoneBookWS.Person;
2  import localhost.axis.services.PhoneBookWS.*;
3  public class Test {
4      public static void main(String[] args) throws Exception {
5          PhoneBookService service=new PhoneBookServiceLocator();
6          PhoneBook port=service.getPhoneBookWS();
7          Person p=new Person();
8          p.setFirst("John");
9          p.setLast("Doe");
10         int[] result=port.getPhoneNumber(p);
11         for(int i=0;i<result.length;i++) {
12             System.out.print(result[i]+" ");
13         }
14         System.out.println();
15     }
16 }
```

Services Web – SOAP en Java – p.20/71

Requête

Requête

```
1 POST /axis/services/PhoneBookWS HTTP/1.0
2 Content-Type: text/xml; charset=utf-8
3 Accept: application/soap+xml, application/dime, multipart/related, text/*
4 User-Agent: Axis/1.1RC2
5 Host: localhost
6 Cache-Control: no-cache
7 Pragma: no-cache
8 SOAPAction: ""
9 Content-Length: 776
10
11 <?xml version="1.0" encoding="UTF-8"?>
12 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
13   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
14   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
15   <soapenv:Body>
16     <ns1:getPhoneNumber soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
17       xmlns:ns1="http://localhost:8080/axis/services/PhoneBookWS">
18       <in0 href="#id0"/>
19     </ns1:getPhoneNumber>
20     <multiRef id="id0" soapenc:root="0"
21       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
22       xsi:type="ns2:Person" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
23       xmlns:ns2="urn:PhoneBookWS">
24       <first xsi:type="xsd:string">John</first>
25       <last xsi:type="xsd:string">Doe</last>
26     </multiRef>
27   </soapenv:Body>
28 </soapenv:Envelope>
```

Services Web – SOAP en Java – p.21/71

Réponse

Réponse

```
1 HTTP/1.1 200 OK
2 Content-Type: text/xml; charset=utf-8
3 Date: Wed, 30 Apr 2003 08:13:59 GMT
4 Server: Apache Coyote/1.0
5 Connection: close
6
7 <?xml version="1.0" encoding="UTF-8"?>
8 <soapenv:Envelope
9   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
10   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
11   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
12   <soapenv:Body>
13     <ns1:getPhoneNumberResponse
14       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
15       xmlns:ns1="http://localhost:8080/axis/services/PhoneBookWS">
16       <getPhoneNumberReturn xsi:type="soapenc:Array"
17         soapenc:arrayType="xsd:int [3]"
18         xmlns:ns2="http://www.w3.org/2002/12/soap-encoding"
19         xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
20         <item>1</item>
21         <item>2</item>
22         <item>3</item>
23       </getPhoneNumberReturn>
24     </ns1:getPhoneNumberResponse>
25   </soapenv:Body>
26 </soapenv:Envelope>
```

Services Web – SOAP en Java – p.22/71

Modes de passage des paramètres

- trois modes de passage en SOAP/RPC :
 - in : pas de problème en Java
 - out : résultat, inconnu en Java (sauf si le résultat est unique, i.e., un seul paramètre en mode out)
 - inout : mixte, inconnu en Java
- représentation en Java de out et inout :
 - classes *Holder*
 - déjà définies pour les types standards (par exemple *IntHolder* et *StringHolder*)
 - engendrées automatiquement pour les autres types par exemple pour le type *Truc* :

TrucHolder.java

```
1 public final class TrucHolder implements javax.xml.rpc.holders.Holder {
2   public Truc value;
3   public TrucHolder() {
4   }
5   public TrucHolder(Truc value) {
6     this.value = value;
7   }
8 }
```

Services Web – SOAP en Java – p.23/71

Exemple

- un service de calculs proposant le port *Calc*
- type de port *Arrays*
- opération *meanVar* :
 - prend comme paramètre in un tableau de double
 - prend comme paramètres inout deux doubles, la moyenne et la variance du tableau paramètre
- en Java, il n'y a pas vraiment de différences en out et inout

Services Web – SOAP en Java – p.24/71

Exemple : WSDL

```
1 <!-- Calc.wsdl -->
2 <wsdl:types>
3   <schema targetNamespace="http://localhost:8080/axis/services/Calc"
4     xmlns="http://www.w3.org/2001/XMLSchema"
5     <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
6     <complexType name="ArrayOf_xsd_double">
7       <complexContent>
8         <restriction base="soapenc:Array">
9           <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:double[]" />
10        </restriction>
11      </complexContent>
12    </complexType>
13  </schema>
14 </wsdl:types>
15 <wsdl:message name="meanVarRequest">
16   <wsdl:part name="x" type="impl:ArrayOf_xsd_double" />
17   <wsdl:part name="mean" type="xsd:double" />
18   <wsdl:part name="var" type="xsd:double" />
19 </wsdl:message>
20 <!-- ... -->
21 <wsdl:message name="meanVarResponse">
22   <wsdl:part name="mean" type="xsd:double" />
23   <wsdl:part name="var" type="xsd:double" />
24 </wsdl:message>
25 <wsdl:portType name="Arrays">
26   <!-- ... -->
27   <wsdl:operation name="meanVar" parameterOrder="x mean var">
28     <wsdl:input message="impl:meanVarRequest" name="meanVarRequest" />
29     <wsdl:output message="impl:meanVarResponse" name="meanVarResponse" />
30   </wsdl:operation>
</wsdl:portType>
```

Services Web – SOAP en Java – p.25/71

Exemple : code client

```
1 <!-- ClientStat.java -->
2 import localhost.axis.services.Calc.*;
3 import javax.xml.rpc.holders.DoubleHolder;
4 public class ClientStat {
5   public static void main(String[] args) throws Exception {
6     ArraysService service=new ArraysServiceLocator();
7     Arrays port=service.getCalc();
8     double[] x={1.5,2.5,2};
9     DoubleHolder mean=new DoubleHolder();
10    DoubleHolder var=new DoubleHolder();
11    port.meanVar(x,mean,var);
12    System.out.println(mean.value);
13    System.out.println(var.value);
14  }
}
```

Services Web – SOAP en Java – p.26/71

Requête

```
1 <!-- Requête -->
2 POST /axis/services/Calc HTTP/1.0
3 Content-Type: text/xml; charset=utf-8
4 Accept: application/soap+xml, application/dime, multipart/related, text/*
5 User-Agent: Axis/1.1RC2
6 Host: localhost
7 Cache-Control: no-cache
8 Pragma: no-cache
9 SOAPAction: ""
10 Content-Length: 696
11 <?xml version="1.0" encoding="UTF-8"?>
12 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
13   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
14   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
15 <soapenv:Body>
16 <ns1:meanVar soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
17   xmlns:ns1="http://localhost:8080/axis/services/Calc">
18 <x xsi:type="soapenc:Array" soapenc:arrayType="xsd:double[3]"
19   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
20 <item>1.5</item>
21 <item>2.5</item>
22 <item>2.0</item>
23 </x>
24 <mean xsi:type="xsd:double">0.0</mean>
25 <var xsi:type="xsd:double">0.0</var>
26 </ns1:meanVar>
27 </soapenv:Body>
28 </soapenv:Envelope>
```

Services Web – SOAP en Java – p.27/71

Requête

```
1 <!-- Réponse -->
2 HTTP/1.1 200 OK
3 Content-Type: text/xml; charset=utf-8
4 Date: Thu, 01 May 2003 10:05:14 GMT
5 Server: Apache Coyote/1.0
6 Connection: close
7 <?xml version="1.0" encoding="UTF-8"?>
8 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
9   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
10   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
11 <soapenv:Body>
12 <ns1:meanVarResponse
13   soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
14   xmlns:ns1="http://localhost:8080/axis/services/Calc">
15 <mean xsi:type="xsd:double">2.0</mean>
16 <var xsi:type="xsd:double">0.25</var>
17 </ns1:meanVarResponse>
18 </soapenv:Body>
19 </soapenv:Envelope>
```

Services Web – SOAP en Java – p.28/71

Traduction des types

L'outil de traduction WSDL vers Java doit

- engendrer une classe pour chaque élément type défini dans le WSDL (la classe porte le même nom que le type)
- engendrer une classe `Holder` pour chaque type spécifique utilisé comme paramètre ou `inout` dans une opération
- chaque espace de noms défini dans le document WSDL doit être associé à un package Java : si par exemple le document définit plusieurs espaces de noms, la traduction doit donc produire plusieurs packages

Traduction des portTypes

L'outil de traduction WSDL produit une interface pour chaque `portType` du WSDL :

- portant le nom du `portType`
- dérivée de `Remote`
- proposant une méthode par opération
- qui représente le "serveur" sur le client
- exemple :

```
1 package localhost.axis.services.Calc;
2
3 public interface Arrays extends java.rmi.Remote {
4     public double[] sum(double[] x, double[] y)
5         throws java.rmi.RemoteException;
6     public void meanVar(double[] x,
7                         javax.xml.rpc.holders.DoubleHolder mean,
8                         javax.xml.rpc.holders.DoubleHolder var)
9         throws java.rmi.RemoteException;
10 }
```

Traduction des bindings

L'outil de traduction WSDL produit pour chaque binding une classe concrète :

- de nom `nameStub`
- qui implémente l'interface associée au `portType` qu'incarne le binding
- qui traduit les appels locaux en messages SOAP et les réponses du serveur (messages SOAP) en réponse locale
- exemples :
 - `CalcSoapBindingStub` implémente `Arrays`
 - `EchoSoapBindingStub` implémente `Echo`
 - etc.
- transparent pour le programmeur (ne s'utilise pas directement)

Traduction des services

L'outil de traduction WSDL produit pour chaque service :

- une interface :
 - de nom `nameService`
 - qui propose une méthode `get` pour chacun des ports regroupés dans le service

● exemple :

```
1 package localhost.axis.services.Echo;
2
3 public interface EchoService extends javax.xml.rpc.Service {
4     public java.lang.String getEchoAddress();
5
6     public localhost.axis.services.Echo.Echo getEcho()
7         throws javax.xml.rpc.ServiceException;
8
9     public localhost.axis.services.Echo.Echo getEcho(java.net.URL portAddress)
10         throws javax.xml.rpc.ServiceException;
11 }
```

● une classe :

- de nom `nameServiceLocator`
- spécifique à Axis
- implémentation de l'interface `nameService`

Gestion des erreurs

Exemple :

- un service de calculs proposant le port Calc
- type de port Arrays
- opération sum :
 - prend comme paramètre deux tableaux de double
 - renvoie le tableau somme des paramètres
- client "incorrect" (tableaux de tailles différentes) :

```
Client.java
1 import localhost.axis.services.Calc.*;
2 public class Client {
3     public static void main(String[] args) throws Exception {
4         ArraysService service=new ArraysServiceLocator();
5         Arrays port=service.getCalc();
6         double[] x={1.5,2.5,3};
7         double[] y={1,-2};
8         double[] z=port.sum(x,y);
9         for(int i=0;i<z.length;i++) {
10             System.out.println(z[i]);
11         }
12     }
13 }
```

Services Web – SOAP en Java – p.33/71

Requête

```
Requête
1 POST /axis/services/Calc HTTP/1.0
2 Content-Type: text/xml; charset=utf-8
3 Accept: application/soap+xml, application/dime, multipart/related, text/*
4 User-Agent: Axis/1.1RC2
5 Host: localhost
6 Cache-Control: no-cache
7 Pragma: no-cache
8 SOAPAction: ""
9 Content-Length: 789
10
11 <?xml version="1.0" encoding="UTF-8"?>
12 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
13     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
14     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
15     <soapenv:Body>
16         <ns1:sum soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
17             xmlns:ns1="http://localhost:8080/axis/services/Calc">
18             <in0 xsi:type="soapenc:Array" soapenc:arrayType="xsd:double[3]"
19                 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
20                 <item>1.5</item><item>2.5</item><item>3.0</item>
21             </in0>
22             <in1 xsi:type="soapenc:Array" soapenc:arrayType="xsd:double[2]"
23                 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
24                 <item>1.0</item><item>-2.0</item>
25             </in1>
26         </ns1:sum></soapenv:Body>
27 </soapenv:Envelope>
```

Services Web – SOAP en Java – p.34/71

Réponse (Fault)

```
Requête
1 HTTP/1.1 500 Erreur Interne de Servlet
2 Content-Type: text/xml; charset=utf-8
3 Date: Thu, 01 May 2003 06:37:42 GMT
4 Server: Apache Coyote/1.0
5 Connection: close
6
7 <?xml version="1.0" encoding="UTF-8"?>
8 <soapenv:Envelope
9     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
10     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
11     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
12     <soapenv:Body>
13         <soapenv:Fault>
14             <faultcode>soapenv:Server.userException</faultcode>
15             <faultstring>java.lang.ArrayIndexOutOfBoundsException: 2</faultstring>
16             <detail/>
17         </soapenv:Fault>
18     </soapenv:Body>
19 </soapenv:Envelope>
```

Le client traduit le *Fault* en une RemoteException

Services Web – SOAP en Java – p.35/71

Un exemple complet

Service :

- requêtes DNS
- implémenté en .NET
- modèle document (pas RPC)
- proposé par www.esynaps.com
- WSDL assez complexe :
 - 264 lignes
 - *binding* POST et GET

Services Web – SOAP en Java – p.36/71

Code engendré

Le service :

YourHost.java

```
1 package org.tempuri;
2
3 public interface YourHost extends javax.xml.rpc.Service {
4     public java.lang.String getYourHostSoapAddress();
5     public org.tempuri>YourHostSoap getYourHostSoap()
6         throws javax.xml.rpc.ServiceException;
7     public org.tempuri>YourHostSoap getYourHostSoap(java.net.URL portAddress)
8         throws javax.xml.rpc.ServiceException;
9 }
```

Le port :

YourHostSoap.java

```
1 package org.tempuri;
2
3 public interface YourHostSoap extends java.rmi.Remote {
4     public org.tempuri.HostInfo whoAmI()
5         throws java.rmi.RemoteException;
6     public org.tempuri.HostInfo getHostInfoByIP(java.lang.String IPAddress)
7         throws java.rmi.RemoteException;
8     public org.tempuri.HostInfo getHostInfoByName(java.lang.String name)
9         throws java.rmi.RemoteException;
10 }
```

Services Web – SOAP en Java – p.37/71

Types engendrés (fragments)

HostInfo.java

```
1 package org.tempuri;
2
3 public class HostInfo implements java.io.Serializable {
4     private java.lang.String name;
5     private org.tempuri.ArrayOfString IPList;
6     private org.tempuri.ArrayOfString1 aliases;
7     private java.lang.String error;
8
9     public HostInfo() {
10    }
11    public java.lang.String getName() {
12        return name;
13    }
14    public void setName(java.lang.String name) {
15        this.name = name;
16    }
17    public org.tempuri.ArrayOfString getIPList() {
18        return IPList;
19    }
20    public void setIPList(org.tempuri.ArrayOfString IPList) {
21        this.IPList = IPList;
22    }
23    public org.tempuri.ArrayOfString1 getAliases() {
24        return aliases;
25    }
}
```

Services Web – SOAP en Java – p.38/71

Types engendrés (fragments) (2)

HostInfo.java

```
26 public void setAliases(org.tempuri.ArrayOfString1 aliases) {
27     this.aliases = aliases;
28 }
29 public java.lang.String getError() {
30     return error;
31 }
32 public void setError(java.lang.String error) {
33     this.error = error;
34 }
```

Services Web – SOAP en Java – p.39/71

Types engendrés (fragments) (3)

ArrayOfString.java

```
1 package org.tempuri;
2
3 public class ArrayOfString implements java.io.Serializable {
4     private java.lang.String[] IPAddress;
5
6     public ArrayOfString() {
7     }
8
9     public java.lang.String[] getIPAddress() {
10        return IPAddress;
11    }
12
13    public void setIPAddress(java.lang.String[] IPAddress) {
14        this.IPAddress = IPAddress;
15    }
16
17    public java.lang.String getIPAddress(int i) {
18        return IPAddress[i];
19    }
20
21    public void setIPAddress(int i, java.lang.String value) {
22        this.IPAddress[i] = value;
23    }
}
```

Services Web – SOAP en Java – p.40/71

Types engendrés (fragments) (4)

```
1 package org.tempuri;
2
3 public class ArrayOfString1 implements java.io.Serializable {
4     private java.lang.String[] alias;
5
6     public ArrayOfString1() {
7     }
8
9     public java.lang.String[] getAlias() {
10        return alias;
11    }
12
13    public void setAlias(java.lang.String[] alias) {
14        this.alias = alias;
15    }
16
17    public java.lang.String getAlias(int i) {
18        return alias[i];
19    }
20
21    public void setAlias(int i, java.lang.String value) {
22        this.alias[i] = value;
23    }
24 }
```

Services Web – SOAP en Java – p.41/71

Client

```
1 import org.tempuri.*;
2 public class Client {
3     public static void print(String[] str) {
4         if(str != null) {
5             for(int i=0;i<str.length;i++) {
6                 System.out.println('\t'+str[i]);
7             }
8         } else {
9             System.out.println("néant");
10        }
11    }
12    public static void main(String[] args) throws Exception {
13        YourHost service=new YourHostLocator();
14        YourHostSoap port=service.getYourHostSoap();
15        HostInfo info=port.getHostInfoByName(args[0]);
16        System.out.println("Nom : "+info.getName());
17        ArrayOfString ips=info.getIPList();
18        System.out.println("Adresse(s) IP :");
19        print(ips.getIPAddress());
20        ArrayOfString1 aliases=info.getAliases();
21        System.out.println("Alias :");
22        print(aliases.getAlias());
23    }
24 }
```

Services Web – SOAP en Java – p.42/71

Exécution

```
1 java Client www.microsoft.com
2
3 Nom : www.microsoft.akadns.net
4 Adresse(s) IP :
5     207.46.249.27
6     207.46.134.155
7     207.46.249.190
8     207.46.134.190
9     207.46.134.222
10    207.46.249.222
11 Alias :
12    www.microsoft.com
```

Services Web – SOAP en Java – p.43/71

Requête

```
1 POST /webservices/YourHostInfo.asmx HTTP/1.0
2 Content-Type: text/xml; charset=utf-8
3 Accept: application/soap+xml, application/dime, multipart/related, text/*
4 User-Agent: Axis/1.1RC2
5 Host: www.esynaps.com
6 Cache-Control: no-cache
7 Pragma: no-cache
8 SOAPAction: "http://tempuri.org/GetHostInfoByName"
9 Content-Length: 374
10
11 <?xml version="1.0" encoding="UTF-8"?>
12 <soapenv:Envelope
13     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
14     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
15     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
16     <soapenv:Body>
17         <GetHostInfoByName xmlns="http://tempuri.org/">
18             <Name>www.microsoft.com</Name>
19         </GetHostInfoByName>
20     </soapenv:Body>
21 </soapenv:Envelope>
```

Services Web – SOAP en Java – p.44/71

Réponse

Réponse

```
1 HTTP/1.1 200 OK
2 Server: Microsoft-IIS/5.0
3 Date: Wed, 30 Apr 2003 14:20:06 GMT
4 Cache-Control: private, max-age=0
5 Content-Type: text/xml; charset=utf-8
6 Content-Length: 706
7
8
9 <?xml version="1.0" encoding="utf-8"?>
10 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
11   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
12   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
13   <soap:Body>
14     <GetHostInfoByNameResponse xmlns="http://tempuri.org/">
15       <GetHostInfoByNameResult>
16         <Name>www.microsoft.akadns.net</Name>
17         <IPList>
18           <IPAddress>207.46.249.27</IPAddress>
19           <IPAddress>207.46.134.155</IPAddress>
20           <IPAddress>207.46.249.190</IPAddress>
21           <IPAddress>207.46.134.190</IPAddress>
22           <IPAddress>207.46.134.222</IPAddress>
23           <IPAddress>207.46.249.222</IPAddress>
24         </IPList>
25         <Aliases>
26           <Alias>www.microsoft.com</Alias>
27         </Aliases>
28       </GetHostInfoByNameResult>
29     </GetHostInfoByNameResponse>
30   </soap:Body>
</soap:Envelope>
```

Services Web – SOAP en Java – p.45/71

Solution évoluée

Avenir pour l'implémentation en Java :

- exposer un Enterprise Java Bean en tant que web service
- limitations :
 - Stateless Session Bean
 - prévu dans J2EE 1.4 (qui n'existe pas encore officiellement !)
 - version basique des services web (problème d'intégration avec les mécanismes de sécurité, les transactions, etc.)
- avantages : on récupère toute la puissance des EJB
- possible avec Axis dès maintenant

Services Web – SOAP en Java – p.47/71

Programmation d'un service

Avec Axis :

- Axis est une application web et s'installe donc dans un serveur JSP/Servlet
- *Instant Deployment* : mécanisme (spécifique Axis) permettant de se contenter de copier une classe à un emplacement spécifique pour obtenir le service correspondant !
- *Custom Deployment* : mécanisme (spécifique Axis) permettant de réaliser des services plus complexes (i.e. nécessitant plusieurs classes par exemple)

Problèmes :

- très spécifique Axis
- implémentation basique : pas de persistance, pas de services de haut niveau, etc.

Services Web – SOAP en Java – p.46/71

Exemple d'Instant Deployment

● programme Java :

```
1 import java.util.Calendar;
2 import java.util.Date;
3 public class Calendrier {
4     public Calendar getDate() {
5         Calendar result = Calendar.getInstance();
6         result.setTime(new Date(System.currentTimeMillis()));
7         return result;
8     }
9 }
```

- à recopier sous le nom `Calendrier.jws` dans un dossier particulier (cf la doc Axis)
- le web service est créé :
 - déployé à l'URL `http://localhost:8080/axis/Compteur.jws`
 - WSDL accessible à l'URL `http://localhost:8080/axis/Compteur.jws?wsdl`
- si on modifie le fichier `.jws`, le web service est mis à jour automatiquement (recompilation et production du `.wsdl`)

Services Web – SOAP en Java – p.48/71

WSDL engendré automatiquement

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <wSDL:definitions targetNamespace="http://localhost:8080/axis/Calendrier.jws"
3   xmlns="http://schemas.xmlsoap.org/wSDL/"
4   xmlns:apachesoap="http://xml.apache.org/xml-soap"
5   xmlns:impl="http://localhost:8080/axis/Calendrier.jws"
6   xmlns:intf="http://localhost:8080/axis/Calendrier.jws"
7   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
8   xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
9   xmlns:wSDLsoap="http://schemas.xmlsoap.org/wSDL/soap/"
10  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
11  <wSDL:message name="getDateRequest">
12  </wSDL:message>
13  <wSDL:message name="getDateResponse">
14    <wSDL:part name="getDateReturn" type="xsd:dateTime"/>
15  </wSDL:message>
16  <wSDL:portType name="Calendrier">
17    <wSDL:operation name="getDate">
18      <wSDL:input message="impl:getDateRequest" name="getDateRequest"/>
19      <wSDL:output message="impl:getDateResponse" name="getDateResponse"/>
20    </wSDL:operation>
21  </wSDL:portType>
```

Services Web – SOAP en Java – p.49/71

WSDL engendré automatiquement (2)

```
22 <wSDL:binding name="CalendrierSoapBinding" type="impl:Calendrier">
23   <wSDLsoap:binding style="rpc"
24     transport="http://schemas.xmlsoap.org/soap/http"/>
25   <wSDL:operation name="getDate">
26     <wSDLsoap:operation soapAction=""/>
27     <wSDL:input name="getDateRequest">
28       <wSDLsoap:body
29         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
30         namespace="http://localhost:8080/axis/Calendrier.jws" use="encoded"/>
31     </wSDL:input>
32     <wSDL:output name="getDateResponse">
33       <wSDLsoap:body
34         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
35         namespace="http://localhost:8080/axis/Calendrier.jws" use="encoded"/>
36     </wSDL:output>
37   </wSDL:operation>
38 </wSDL:binding>
39 <wSDL:service name="CalendrierService">
40   <wSDL:port binding="impl:CalendrierSoapBinding" name="Calendrier">
41     <wSDLsoap:address location="http://localhost:8080/axis/Calendrier.jws"/>
42   </wSDL:port>
43 </wSDL:service>
44 </wSDL:definitions>
```

Services Web – SOAP en Java – p.50/71

Exemple de client

```
1 import localhost.axis.Calendrier_jws.*;
2 import java.util.Date;
3 import java.util.Calendar;
4 import java.text.DateFormat;
5 public class Client {
6   public static void main(String[] args) throws Exception {
7     CalendrierService service=new CalendrierServiceLocator();
8     Calendrier port=service.getCalendrier();
9     Calendar now=port.getDate();
10    System.out.println(DateFormat.getDateInstance().format(now.getTime()));
11  }
12 }
```

Services Web – SOAP en Java – p.51/71

Limitations de l'Instant Deployment

- pas de *Java Beans* pour le *mapping* de types évolués
- code source obligatoire
- pas de maîtrise du cycle de vie :
 - création de l'objet
 - destruction
- pas de contrôle fin :
 - sur le fichier WSDL engendré
 - sur les méthodes exportées
 - etc.

Services Web – SOAP en Java – p.52/71

Création

Avec l'*Instant Deployment*, l'objet est recréé à chaque invocation :

● exemple :

```
Compteur.java
1 public class Compteur {
2     private int compte;
3     public Compteur() {
4         compte = 0;
5     }
6     public int next() {
7         return compte++;
8     }
9 }
```

- nouvel objet à chaque appel : le compteur est toujours nul!
- problèmes :
 - pas de persistance simple
 - gaspillage de ressources (reconnexion à une BD par exemple)

Exemple de *Custom Deployment*

- on reprend le même programme
- on copie le fichier compilé (Compteur.class) à un emplacement spécifique (cf la doc Axis)

● on fabrique un fichier WSDD :

- Web Service Deployment Descriptor
- spécifique Axis
- exemple :

```
deploy.wsdd
1 <deployment xmlns="http://xml.apache.org/axis/wsdd/"
2     xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
3     <service name="Compteur" provider="java:RPC">
4         <parameter name="className" value="Compteur"/>
5         <parameter name="allowedMethods" value="*"/>
6     </service>
7 </deployment>
```

- indique la classe principale d'un service, les méthodes à exporter, etc.

Exemple de *Custom Deployment* (2)

- on exécute `java org.apache.axis.client.AdminClient deploy.wsdd`
- le service est déployé :
 - à l'URL `http://localhost:8080/axis/services/Compteur`
 - WSDL : `http://localhost:8080/axis/services/Compteur?wsdl`
- pour supprimer le service, on utilise un autre fichier WSDD :

```
undeploy.wsdd
1 <undeployment xmlns="http://xml.apache.org/axis/wsdd/"
2     <service name="Compteur"/>
3 </undeployment>
```

- gestion du cycle de vie :
 - sous-élément de service
 - `<parameter name="scope" value="value"/>`
 - l'attribut `value` peut être :
 - `request` : un objet par requête (par défaut)
 - `application` : un seul objet par serveur
 - `session` : un objet par session (extension Axis)

Java beans

On reprend l'exemple de l'annuaire :

● type Person :

```
Person.java
1 package org.apiacoa.phonebook;
2 public class Person {
3     private String first;
4     private String last;
5     public void setFirst(String f) {
6         first=f;
7     }
8     public String getFirst() {
9         return first;
10    }
11    public void setLast(String f) {
12        last=f;
13    }
14    public String getLast() {
15        return last;
16    }
17 }
```

● service :

```
PhoneBook.java
1 package org.apiacoa.phonebook;
2 public class PhoneBook {
3     public int[] getPhoneNumber(Person p) {
4         return new int[] {1,2,3};
5     }
6 }
```

Déploiement

● fichier WSDD :

```
1 <deployment xmlns="http://xml.apache.org/axis/wsdd/"
2   xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
3   <service name="PhoneBookWS" provider="java:RPC">
4     <parameter name="className" value="org.apiacoa.phonebook.PhoneBook"/>
5     <parameter name="allowedMethods" value="*/>
6     <beanMapping qname="myNS:Person"
7       xmlns:myNS="http://localhost:8080/axis/services/PhoneBookWS"
8       languageSpecificType="java:org.apiacoa.phonebook.Person"/>
9   </service>
10 </deployment>
```

● beanMapping :

- qname : nom du type pour WSDL
- languageSpecificType : objet correspondant (attention au préfixe java)
- mapping standard :
 - une *struct* SOAP
 - un champ par propriété du *Java Bean*

WSDL

- engendré automatiquement au déploiement
- peut être engendré avant :
 - java org.apache.axis.wsdl.Java2WSDL ...
 - permet de préciser certains éléments :
 - traduction *package* vers *namespace*
 - emplacement du WS
 - *encoding* (*literal* ou *encoded*)
 - style (document ou RPC)
 - etc.
- emplacement précisé par l'élément `wSDLFile` (fils de service)

Types engendrés

```
PhoneBookWS.wsdl
1 <wsdl:types>
2   <schema targetNamespace="http://localhost:8080/axis/services/PhoneBookWS"
3     xmlns="http://www.w3.org/2001/XMLSchema">
4     <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
5     <complexType name="Person">
6       <sequence>
7         <element name="first" nillable="true" type="xsd:string"/>
8         <element name="last" nillable="true" type="xsd:string"/>
9       </sequence>
10    </complexType>
11    <complexType name="ArrayOf_xsd_int">
12      <complexContent>
13        <restriction base="soapenc:Array">
14          <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:int []"/>
15        </restriction>
16      </complexContent>
17    </complexType>
18  </schema>
19 </wsdl:types>
```

Fonctions avancées

- mapping XML spécifique :
 - permet de transformer toute classe Java en XML et vice-versa
 - demande la programmation des traductions (!)
- *Handlers* :
 - s'intercale entre le serveur et le client
 - peut modifier les messages SOAP au vol
 - applications : cryptographie, signature, compression, *logging*, etc.
- transport par messages (à la place de HTTP) basé sur JMS
 - signature
 - sessions (HTTP et header SOAP)
 - *attachments*

Publication d'un EJB comme service web

- spécification EJB 2.1, J2EE 1.4
- ne concerne que les *Stateless Session Beans*
- déjà disponible dans divers produits :
 - JOnAS 3.1 (délicat à configurer), basé sur Axis
 - JBoss 3.2 :
 - basé sur Axis
 - à démarrer en mode all
 - attention aux problèmes de sécurité
 - diverses extensions pour la gestion de session, pour la sécurité, etc.
- avec Axis :
 - rien à programmer (en plus des EJB)
 - tout passe par WSDD

Services Web – SOAP en Java – p.61/71

Exemple avec JBoss

Code de l'EJB :

```
EchoBean.java
1 package org.apiacoa.echo;
2 import javax.ejb.CreateException;
3 import javax.ejb.EJBException;
4 import javax.ejb.SessionBean;
5 import javax.ejb.SessionContext;
6
7 /**
8  * Session Bean Template
9  *
10 * @ejb:bean name="Echo"
11 *     display-name="Echo Bean"
12 *     type="Stateless"
13 *     transaction-type="Container"
14 *     jndi-name="ejb/org/apiacoa/echo/Echo"
15 *
16 **/
17 public class EchoBean implements SessionBean {
18     private SessionContext mContext;
19     /**
20      * @ejb:interface-method view-type="remote"
21      **/
22     public String echo(String s) {
23         return "Echo: "+s+"("+hashCode()+)";
24     }
25 }
```

Services Web – SOAP en Java – p.62/71

Code EJB (2)

```
EchoBean.java
25 /**
26  * @ejb:create-method view-type="remote"
27  **/
28 public void ejbCreate() throws CreateException {
29     System.out.println( "EchoBean: create" );
30 }
31 public String toString() {
32     return "EchoBean [ " + hashCode() + " ]";
33 }
34 public void setSessionContext(SessionContext aContext)
35     throws EJBException {
36     mContext = aContext;
37 }
38 public void ejbActivate() throws EJBException {
39 }
40 public void ejbPassivate() throws EJBException {
41 }
42 public void ejbRemove() throws EJBException{
43 }
44 }
45
46 }
```

Services Web – SOAP en Java – p.63/71

XDoclet

Engendre le reste du code :

```
Echo.java
1 /*
2  * Generated file - Do not edit!
3  */
4 package org.apiacoa.echo;
5
6 import java.lang.*;
7 import javax.ejb.CreateException;
8 import javax.ejb.EJBException;
9 import javax.ejb.SessionBean;
10 import javax.ejb.SessionContext;
11
12 /**
13  * Remote interface for Echo.
14  * @xdoclet-generated at 6 mai 2003 14:27:42
15  */
16 public interface Echo
17     extends javax.ejb.EJBObject
18 {
19
20     public java.lang.String echo( java.lang.String s ) throws java.rmi.RemoteException;
21
22 }
```

Services Web – SOAP en Java – p.64/71

XDoclet (2)

EchoHome.java

```
1  /*
2  * Generated file - Do not edit!
3  */
4  package org.apicoa.echo;
5
6  import java.lang.*;
7  import javax.ejb.CreateException;
8  import javax.ejb.EJBException;
9  import javax.ejb.SessionBean;
10 import javax.ejb.SessionContext;
11
12 /**
13  * Home interface for Echo. Lookup using {1}
14  * @xdoclet-generated at 6 mai 2003 12:21:40
15  */
16 public interface EchoHome
17     extends javax.ejb.EJBHome
18 {
19     public static final String COMP_NAME="java:comp/env/ejb/Echo";
20     public static final String JNDI_NAME="ejb/org/apicoa/echo/Echo";
21
22     public org.apicoa.echo.Echo create() throws javax.ejb.CreateException,
23         java.rmi.RemoteException;
24
25 }
```

Services Web – SOAP en Java – p.65/71

Fichier WSDD spécifique

deploy.wsdd

```
1 <deployment xmlns="http://xml.apache.org/axis/wsdd/"
2             xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
3     <service name="EchoService" provider="java:EJB">
4         <parameter name="jndiURL" value="localhost"/>
5         <parameter name="jndiContextClass"
6             value="org.jnp.interfaces.NamingContextFactory"/>
7         <parameter name="beanJndiName" value="ejb/org/apicoa/echo/Echo"/>
8         <parameter name="homeInterfaceName" value="org.apicoa.echo.EchoHome"/>
9         <parameter name="remoteInterfaceName" value="org.apicoa.echo.Echo"/>
10        <parameter name="allowedMethods" value="echo"/>
11    </service>
12 </deployment>
```

- attention, il ne faut pas mettre * dans allowedMethods.
- on déploie (au sens web service) avec la commande :
java org.apache.axis.client.AdminClient
-lhttp://localhost:8080/jboss-net/services/Administration deploy.wsdd
- le service est accessible à l'URL
http://localhost:8080/jboss-net/services/EchoService

Services Web – SOAP en Java – p.66/71

WSDL engendré

EchoService.wsdl

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <wsdl:definitions
3     targetNamespace="http://localhost:8080/jboss-net/services/EchoService"
4     xmlns="http://schemas.xmlsoap.org/wsdl/"
5     xmlns:apachesoap="http://xml.apache.org/xml-soap"
6     xmlns:impl="http://localhost:8080/jboss-net/services/EchoService"
7     xmlns:intf="http://localhost:8080/jboss-net/services/EchoService"
8     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
9     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
10    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
11    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
12 <wsdl:types/>
13 <wsdl:message name="echoRequest">
14     <wsdl:part name="in0" type="xsd:string"/>
15 </wsdl:message>
16 <wsdl:message name="echoResponse">
17     <wsdl:part name="echoReturn" type="xsd:string"/>
18 </wsdl:message>
19 <wsdl:portType name="Echo">
20     <wsdl:operation name="echo" parameterOrder="in0">
21         <wsdl:input message="impl:echoRequest" name="echoRequest"/>
22         <wsdl:output message="impl:echoResponse" name="echoResponse"/>
23     </wsdl:operation>
24 </wsdl:portType>
```

Services Web – SOAP en Java – p.67/71

WSDL engendré (2)

EchoService.wsdl

```
25 <wsdl:binding name="EchoServiceSoapBinding" type="impl:Echo">
26     <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
27     <wsdl:operation name="echo">
28         <wsdlsoap:operation soapAction=""/>
29         <wsdl:input name="echoRequest">
30             <wsdlsoap:body
31                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
32                 namespace="http://localhost:8080/jboss-net/services/EchoService"
33                 use="encoded"/>
34         </wsdl:input>
35         <wsdl:output name="echoResponse">
36             <wsdlsoap:body
37                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
38                 namespace="http://localhost:8080/jboss-net/services/EchoService"
39                 use="encoded"/>
40         </wsdl:output>
41     </wsdl:operation>
42 </wsdl:binding>
43 <wsdl:service name="EchoService">
44     <wsdl:port binding="impl:EchoServiceSoapBinding" name="EchoService">
45         <wsdlsoap:address
46             location="http://localhost:8080/jboss-net/services/EchoService"/>
47     </wsdl:port>
48 </wsdl:service>
49 </wsdl:definitions>
```

Services Web – SOAP en Java – p.68/71

Clients

Web services

```
1 import localhost.jboss_net.services.EchoService.*;
2 public class Client {
3     public static void main(String[] args) throws Exception {
4         EchoService service=new EchoServiceLocator();
5         Echo port=service.getEchoService();
6         System.out.println(port.echo(args[0]));
7     }
8 }
```

EJB

```
1 package test;
2 import javax.naming.InitialContext;
3 import javax.rmi.PortableRemoteObject;
4 import org.apiacoa.echo.*;
5
6 public class Client {
7     public static void main(String[] args) throws Exception {
8         InitialContext context = new InitialContext();
9         EchoHome home = (EchoHome)
10             PortableRemoteObject.narrow(context.lookup("ejb/org/apiacoa/echo/Echo"),
11                                         EchoHome.class);
12
13         Echo service = home.create();
14         System.out.println(service.echo(args[0]));
15     }
16 }
```

Services Web – SOAP en Java – p.69/71

Conseils pratiques

- pas de solution portable pour la gestion de session :
 - utiliser des *stateless session beans*
 - gérer “à la main” la session, par exemple :
 - une clé permanente par l'utilisateur
 - une clé temporaire créée par un mécanisme de login à partir d'une clé permanente
 - transmettre l'information de session à chaque appel
- par encore de solution portable pour la sécurité : utiliser HTTPS (?)
- on peut envisager des *wrappers* web services construits à la main (solution intérimaire)

Services Web – SOAP en Java – p.70/71

Conseils pratiques (2)

- utiliser des *Business Delegates* :
 - “définition” : ensemble d'interfaces et de classes qui permet à la couche de présentation d'utiliser la couche métier
 - *design pattern* standard en J2EE
 - cache totalement au client le fait qu'il utilise un système réparti
 - sous forme d'interfaces, ce qui permet de changer l'implémentation :
 - une implémentation RMI classique (EJB)
 - une implémentation locale (tests)
 - une implémentation web service (firewall)
 - etc.

Services Web – SOAP en Java – p.71/71