Web Services via J2SE e J2ME

Vitor F. Pamplona

Nos últimos anos o termo "Web Service" tem chamado a atenção de muitos analistas e arquitetos, principalmente dos mais fánáticos por business-to-business (B2B). O conceito foi criado, implementado e agora está começando a ser utilizado.

As espectativas são grandes, altos investimentos, frameworks poderosos, ganhos em produtividade, portabilidade e em independência. Neste artigo, será apresentado um pouco desta grande tecnologia que prometeu e, já está cumprindo, revolucionar o modo como os sitemas são desenvolvidos.

Introdução a Web Service

Um requisito básico de qualquer empresa é prover serviços, sejam os vendedores de uma empresa, o setor de custos e compras, os prestadores de serviço, etc. Cada empresa oferece serviços para a comunicação entre ela e outras pessoas, sejam pessoas físicas ou jurídicas, internas ou externas a empresa.

Alguns desses serviços podem ser automatizados. Por exemplo, não é necessário existir um representante de vendas se o seu cliente já tem, em mãos, o preço e todos os outros dados relevantes para constituir um pedido de compra. Este pedido pode e, em muitos casos, já é feito, via interfaces computacionais. O cliente entra em seu site, monta o pedido como desejar e confirma a compra. Isto é um serviço web, ou seja, um serviço que está publicado na web para que qualquer pessoa possa fazer uso.

Web Services foram criados para construir aplicações deste tipo, aplicações que são serviços na internet. Porém não faz parte do conceito de Web Service a criação de interfaces gráficas para os usuários, deixando esta parte para outras empresas ou pessoas desenvolverem. É comum encontrar textos afirmando que Web Services disponibilizam serviços somente para desenvolvedores, ou que Web Services nada mais são do que chamada de métodos usando XML. Estas definições estão corretas.

Web Services é a tecnologia ideal para comunicação entre sistemas, sendo muito usado em aplicações B2B. A comunicação entre os serviços é padronizada possibilitando a independencia de plataforma e de linguagem de programação. Por exemplo, um sistema de reserva de passagens aéreas feito em Java e rodando em um servidor Linux pode acessar, com transparência, um serviço de reserva de hotel feito em .Net rodando em um servidor Microsoft.

Para comunicar com o Web Service, é necessário uma implementação do protocolo SOAP (Simple Object Access Protocol) definido no W3C. Este protocolo é o responsável pela independência que o Web Service precisa. Atualmente já encontra-se várias implementações disponíveis em várias linguagens. É só escolher uma e usar.

Na Figura 1 encontra-se um diagrama mostrando as mensagens trocadas entre cliente e servidor em uma comunicação SOAP. Existem duas aplicações se comunicando, um Client Wrapper e um Server Wrapper que estão disponibilizando a transparência para as aplicações. Entre eles só trafega XML, seguindo o protocolo SOAP sobre HTTP.

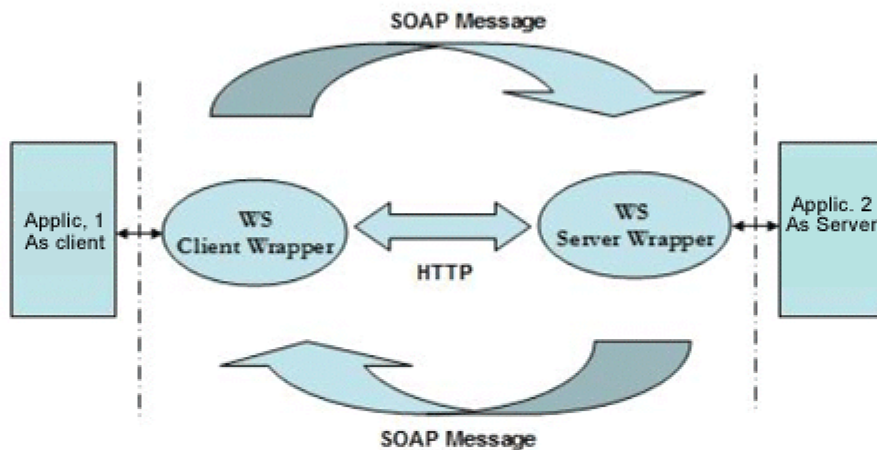

Figura1: Retirada do Artigo Developing JAX-RPC Based Web Services Using Axis and SOAP

Um Web Service será publicado, e para que outras pessoas possam utilizá-lo é necessário definir como ele é, como deve ser acessado, e que valores ele retornará. Estas definições são descritas em um arquivo XML de acordo com a padronização Web Service Description Language (WSDL). Este arquivo deve ser construído para que os usuários do serviço possam entender o funcionamento do Web Service e, logicamente, será de acesso público.

Os Web Services também podem ser utilizados para implementar arquiteturas orientadas a serviços, as Service-Oriented Architectures (SOA). Neste modelo de arquitetura os principais requisitos viram serviços e são acessados por outros serviços, modularizando e aumentando a coesão dos componentes da aplicação.

O que será implementado neste artigo é um Web Service simples, para aprendizado da tecnologia e para desmentir a afirmação que Web Services são complexos de construir e complexos para utilizar .

Conhecendo o ambiente servidor

Todo o Web Service precisa ficar ativo e esperando requisições, portanto, necessita estar executando em um servidor. Neste artigo será utilizado o servidor Tomcat para executar um framework de Web Service chamado Axis. Ambos são sub-projetos livres da Apache, sendo que o Tomcat está no projeto Jakarta e o Axis está no projeto Web Services.

O Tomcat é um container para JSP e Servlets muito conhecido e muito utilizado. Encontra-se vários tutoriais e artigos sobre ele espalhados em páginas na internet.

Já o Axis é um conjunto de ferramentas para desenvolver WebServices. Dentre suas principais funcionalidades estão:

- implementação do protocolo SOAP;
- implementação de classes para agilizar a comunicação e a publicação de Web Services;
- utiliza containers JSP para disponibilizar os Web Services na rede.

Instalando o ambiente servidor

Para criarmos o servidor é necessário baixar e instalar o Tomcat. Na época do lançamento deste artigo, a versão do Tomcat em produção era a 5.0 e poderia ser encontrada neste endereço: http://apache.usp.br/jakarta/tomcat-5/v5.0.28/bin/jakarta-tomcat-5.0.28.zip

Após baixar o Tomcat, deve-se descompactar o zip e criar a variável de ambiente CATALINA_HOME que deverá indicar o local onde será descompactado. Depois de concluído esse passo é possível iniciar o servidor executando o arquivo: CATALINA_HOME/bin/startup.bat no Windows ou CATALINA_HOME/bin /startup.sh no Linux.

Para verificar se o Tomcat está rodando, utiliza-se um navegador com o seguinte endereço: http://localhost:8080/ . Se a instalação estiver correta o Tomcat irá apresentar uma mensagem de ok. É claro, esta é a instalação padrão do Tomcat, você poderá modificar e adicionar mais segurança a ela.

O próximo a ser baixado e configurado é o Axis que tem um fonte em Java e um em C++. Será utilizado o fonte em java na versão 1.1 que pode ser encontrado neste endereço: http://apache.usp.br/ws/axis/1_1/axis-1_1.zip

Após baixar o arquivo zip do Axis, e descompactá-lo, deve-se mover a pasta "axis", que encontra-se dentro do diretório webapps do arquivo, para a pasta webapps do Tomcat.

Agora basta reiniciar o servidor Tomcat, abrir um navegador e ir para à página: http://localhost:8080/axis . Uma página do Axis será apresentada concluíndo a instalação como a Figura 2:

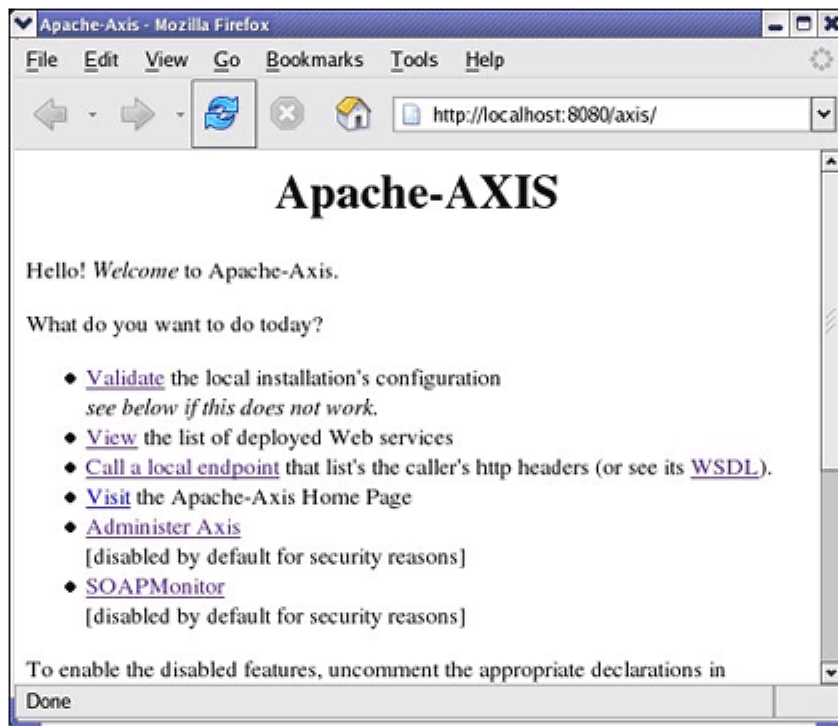

Figura 2

Nesta mesma página de apresentação, existirão dois links importantes:

- um link para validar a instalação: clicando nele será apresentado uma lista de componenetes necessários ("Needed Components"). Caso algum desses não for encontrado ele irá solicitar a instalação. Com as versões trabalhadas neste artigo, o Tomcat e o Axis estarão completos portanto a validação OK;
- um link para visualizar os Web Services já instalados: clicando nele, existirão dois Web Services e clicando no link *(wsdl)* você verá o arquivo de especificação de ambos. Se, ao clicar, não aparecer nenhuma informação, não se preocupe, alguns navegadores não exibem XML, outros exibem como

HTML, sendo necessário abrir o fonte da página para ver o código. Mas ele estará lá e funcionando.

Implementando um Web Service simples

O objetivo é aprender, então será criado um serviço bem simples. O serviço é a soma de duas variáveis inteiras retornando o resultado. Este exemplo poderá servir para qualquer outra implementação. Abaixo está a classe implementada. O nome do arquivo é Servico.java:

```
public class Servico {
   public int soma(int valor1, int valor2) {
      return valor1 + valor2;
   }
}
```

Agora só falta disponibilizá-lo no nosso servidor para o mundo acessar. E, para fazer isso, deve-se alterar o nome do arquivo de Servico.java para Servico.jws, coloca-lo no diretório: CATALINA_HOME/webapps/axis/ e iniciar o servidor, se ele já não estiver iniciado. Se já estiver iniciado, o seu Web Service está publicado.

Os arquivos .jws são lidos pelo Axis e representam Java Web Services. O Axis se baseará nesses arquivos (.jws) para criar os arquivos de definição WSDL. Todos os métodos públicos existentes nessas classes serão automaticamente disponibilizados para terceiros.

Criar documentos XML é demorado e, muitas vezes, chato. Gerar o WSDL é uma característica muito relevante na escolha de uma implementação de SOAP e o Axis é um dos poucos frameworks que conseguem fazer essa façanha de maneira transparente para o desenvolvedor. É por esse motivo que ele é altamente recomendado na construção de Web Services.

Para acessar o Web Service criado basta abrir um navegador e ir ao endereço: http://localhost:8080/axis/Servico.jws . Da mesma forma que os outros dois Web Services foram vistos, este também terá um link para ver a especificação WSDL, e novamente poderá ser visto ou não dependendo do seu navegador.

O arquivo WSDL da classe Servico ficará como abaixo:

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <wsdl:definitions targetNamespace="http://localhost:8080/axis/Servico.jw
03.      xmlns="http://schemas.xmlsoap.org/wsdl/"
04.      xmlns:apachesoap="http://xml.apache.org/xml-soap"
05.      xmlns:impl="http://localhost:8080/axis/Servico.jws"
06.      xmlns:intf="http://localhost:8080/axis/Servico.jws"
07.      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
08.      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
09.      xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
10.      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
11.    <wsdl:message name="somaRequest">
12.     <wsdl:part name="valor1" type="xsd:int"/>
13.     <wsdl:part name="valor2" type="xsd:int"/>
14.   </wsdl:message>
15.   <wsdl:message name="somaResponse">
16.     <wsdl:part name="somaReturn" type="xsd:int"/>
17.   </wsdl:message>
18.   <wsdl:portType name="Servico">
19.     <wsdl:operation name="soma" parameterOrder="valor1 valor2">
20.       <wsdl:input message="impl:somaRequest" name="somaRequest"/>
21.       <wsdl:output message="impl:somaResponse" name="somaResponse"/>
22.     </wsdl:operation>
23.   </wsdl:portType>
24.   <wsdl:binding name="ServicoSoapBinding" type="impl:Servico">
25.     <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/s
26.     <wsdl:operation name="soma">
27.       <wsdlsoap:operation soapAction=""/>
28.       <wsdl:input name="somaRequest">
29.         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/enc
30.             namespace="http://DefaultNamespace" use="encoded"/>
31.       </wsdl:input>
32.       <wsdl:output name="somaResponse">
33.         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/enc
34.             namespace="http://localhost:8080/axis/Servico.jws" use="encod
35.       </wsdl:output>
36.     </wsdl:operation>
37.   </wsdl:binding>
38.   <wsdl:service name="ServicoService">
39.     <wsdl:port binding="impl:ServicoSoapBinding" name="Servico">
40.       <wsdlsoap:address location="http://localhost:8080/axis/Servico.jws"
41.     </wsdl:port>
42.   </wsdl:service>
43.</wsdl:definitions>
```

Analisar este arquivo é essencial para entender a profundidade da implementação. Uma das linhas mais importantes para este arquivo é a linha 19, onde define-se o nome do método e o nome de seus parâmetros. Eles deverão ser de conhecimento público para que as interfaces cliente consigam se comunicar com o Web Service.

Realizando um teste básico

O Axis aceita que um Web Service seja chamado via uma requisição HTTP-GET. Portanto, ao digitar um endereço é possível testar o web service. No exemplo deste artigo o endereço é este: http://localhost:8080/axis/Servico.jws?method=soma&valor1=2&valor2=4.

Como pode-se notar, o endereço é a junção de um namespace, que é o endereço do WebService representado por **http://localhost:8080/axis/Servico.jws**, a variável **method** que, como seu nome diz, contém o nome do

método que se deseja executar, e uma sequência dos parâmetros deste método. Lembrando que o nome dos parâmetros deve ser o mesmo definido na função da classe.

O resultado da execução é um documento XML com a resposta 6. Novamente, dependendo do browser não será visivel as tags XML. O XML que retornou na execução está abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <somaResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encod
   <somaReturn xsi:type="xsd:int">6</somaReturn>

  </somaResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Criando um cliente em Java para acessar o Servidor

O cliente também é uma classe simples, mas exige conhecimento em algumas classes não tão comuns no dia-a-dia.

As classes Service e Call são classes do Axis, portanto, para compilar e executar esta classe é necessário que todo o diretório lib, encontrado dentro do zip do Axis esteja no CLASSPATH da aplicação.

Visto este detalhe, abaixo encontra-se o arquivo fonte do cliente de Web Service. Esta classe fará a conexão ao Web Service para somar 2 com 4 e irá apresentar o resultado 6 na saída padrão.

```
01. import org.apache.axis.client.Service;
02. import org.apache.axis.client.Call;
03.
04. public class Cliente {
05.    public static void main(String[] args) throws Exception {
06.        // Endereço, local onde encontra-se o Web Service
07.        String local = "http://localhost:8080/axis/Servico.jws";
08.
09.        // Criando e configurando o serviço
10.        Call call = (Call) new Service().createCall();
11.        // Configurando o endereço.
12.        call.setTargetEndpointAddress(local);
13.        // Marcando o método a ser chamado.
14.        call.setOperationName("soma");
```

```
15.
16.         // Parâmetros da função soma.
17.         Object[] param = new Object[]{new Integer(2),new Integer(4)};
18.         // Retorno da Função
19.         Integer ret = (Integer)call.invoke(param);
20.
21.         // Imprime o resultado: ret = 2 + 4.
22.         System.out.println("Resultado da soma : " + ret);
23.     }
24. }
```

Este código está dentro de um arquivo chamado Cliente.java, após compilar e executar esta classe exibirá o resultado "Resultado da soma : 6" como desejado.

O framework do Axis trata a primitiva `int` e a classe wrapper `Integer` como sendo iguais. Portanto, tanto faz usar uma ou outra. Neste exemplo, foi criado o Web Service com dois parâmetros `int` e aqui no cliente estamos usando dois parâmetros `Integer`.

Como pode-se notar, o framework do Axis abstrai qualquer trabalho com XML, evitando que o desenvovedor necessite conhecer a sintaxe do XML do SOAP.

Acessando o Web Service via J2ME

Para este passo, é necessário que o Java Wireless Toolkit esteja instalado e funcionando no ambiente.

A comunicação com Web Services se dá através de XML e do protocolo SOAP. Como o J2ME não possui classes para tratar estas implementações, é necessário utilizar outros dois projetos para atender as transparência. Os projetos são o KSOAP e o KXML da ObjectWeb. Ambos estão sob licença pública.

Como pretende-se criar uma simples aplicação para celular (MIDP2), será utilizado o fonte dos dois projetos junto um outro arquivo fonte que será criado. É o jeito mais fácil de executar uma aplicação J2ME com duas bibliotecas

O fonte do KSOAP pode ser encontrado aqui: http://ksoap.objectweb.org/software/downloads/current/ksoap-source.zip

E o fonte do KXML pode ser encontrado aqui: http://kxml.objectweb.org/software/downloads/current/kxml-source.zip

Descompacte os dois e crie a seguinte estrutura com os arquivos baixados:

- SeuProjetoJ2ME
    - org
        - kxml
          -- Todas as suas pastas e arquivos internos a esta pasta que estão no zip.
        - kobjects
          -- Todas as suas pastas e arquivos internos a esta pasta que estão no zip.
        - ksoap

- transport

-- Necessário excluir o pacote marshal.

Não serão utilizados as pastas referentes a servlets e a j2se do ksoap. Somente referente a J2ME e ao fonte básico.

No diretório SeuProjetoJ2ME, deve ser criado a classe ClienteJ2ME.java conforme abaixo:

```
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.TextBox;

import org.ksoap.SoapObject;
import org.ksoap.transport.HttpTransport;

public class ClienteJ2ME extends javax.microedition.midlet.MIDlet {
  private Display display;
  private String url = "http://localhost:8080/axis/Servico.jws";
  TextBox textbox = null;

  public void startApp() {
    display = Display.getDisplay(this);
    try {
      testWebService();
    } catch (Exception ex) {
      System.out.println(ex);
    }
  }

  public void pauseApp() {}
  public void destroyApp(boolean unconditional) {}

  public void testWebService() throws Exception {
    StringBuffer stringBuffer = new StringBuffer();

    TextBox textBox = null;

    // Chama o WebService
    SoapObject client = new  SoapObject(url,"soma");
    client.addProperty("valor1",new Integer(2));
    client.addProperty("valor2",new Integer(4));
    HttpTransport ht = new HttpTransport(url,"soma");
    stringBuffer.append("
Resultado: " + ht.call(client));

    // mostra o valor do resultado na tela.
    textBox = new TextBox("Teste WebService", stringBuffer.toString(), 1024,
```

```
    display.setCurrent(textBox);
  }
}
```

Pronto! Aplicação construída. Essas poucas linhas de código irão gerar os mesmos documentos XML de envio e recepção que o cliente Desktop produziu. Esse exemplo é um MIDlet para MIDP2 e deve ser executada com o Java Wireless Toolkit. Abaixo segue os passos para compilar e executar a aplicação no Java Wireless Toolkit.

- Abrir o JWT e criar um novo projeto
- Configure as opções do projeto para utilizar MIDP2.0 e CLDC1.1
- Configure o nome do MIDlet para ClienteJ2ME
- Copiar este último arquivo e os fontes que você separou do KSOAP e do KXML para a pasta src do projeto.
- Compilar e Executar (O servidor Tomcat deve estar inicializado).

Em J2ME as aplicações clientes de Web Services são mais difíceis de serem desenvolvidas, pois requerem mais conhecimento que os clientes Desktop. Mas, pela simplicidade dos dispositivos e o valor agregado, essa implementação, em alguns casos, ainda é viável. O que deve ser analizado é o custo da conexão em produção.

Conexões com a internet via celulares normalmente são pagos por kilobyte trafegado nas redes das operadoras. O valor é caro, e como os arquivos XMLs são grandes, o custo de fazer a comunicação com Web Service pode não ser a melhor saída. Analise antes de qualquer escolha.

Bibliotecas e componentes usados

- Apache - Axis
- Apache - Tomcat
- ObjectWeb - KSOAP
- ObjectWeb - KXML

Referências

- Axis: The next generation of Apache SOAP
- Axis com JBoss
- Developing JAX-RPC Based Web Services Using Axis and SOAP
- Interoperability of Web Services: Seeing is believing
- Using Apache Axis version 1 to build Web Services
- Web Services
- Web Services with Axis

Abraço pessoal!