Servicios Web y RMI

Pedro J. Muñoz Merino
http://www.it.uc3m.es/pedmume/

Servicios Web: Concepto

- Uso más habitual de un servidor es dar una respuesta al usuario de una determinada página (PHP, ASP, J2EE…). Las páginas están disponibles para los usuarios
- Otro uso es el de proporcionar disponibles una serie de métodos que son accesibles para ser llamados desde otras máquinas de manera remota. Es el concepto de Servicios Web. Los métodos remotos están disponibles para las máquinas
  - Computación distribuida. En lugar de tener todos los métodos locales, se pueden invocar métodos remotos en servidores que ofrecen esos servicios
- Los Servicios Web ofrecen unos métodos con una serie de parámetros y devuelven una salida

Servicios Web: Ventajas

- El tener los métodos de manera remota, puede parecer inicialmente un problema, pero presenta una serie de ventajas:
  - Los cambios en los métodos de los Servicios Web son transparentes para el programador. El programador sólo se debe ocupar de conocer la interfaz al método del Servicio Web
  - Se pueden implementar funcionalidades nuevas que no es posible mediante la programación local, aprovechando la potencia de la Web. Por ejemplo un buscador de páginas web
  - El almacenamiento y ejecución de los programas no recae en una sola máquina sino que está repartida por todas las máquinas de Internet de las que se hace uso. Computación distribuida
  - Permiten interconexión de clientes y máquinas remotas con independencia de los lenguajes de programación de ambos

Servicios Web: Visión General

- Servicios Web: Visión General
  - Descripción de cuales son los parámetros de entrada y de salida, los tipos de mensajes intercambiados, etc. de forma que el servicio Web quede descrito. Por ejemplo WSDL. Tecnología en XML para dicha descripción
  - Mensajes intercambiados en los protocolos para transferir la información y hacer las peticiones y respuestas. Por ejemplo SOAP. Tecnología en XML para dicha descripción
  - APIs y artilugios de los clientes y los servidores para dejar disponibles esos métodos y poder llamarlos. Dependerá del lenguaje de programación. Por ejemplo AXIS como librería en JAVA

Servicios Web: WSDL

- Fichero XML con estas etiquetas:
  - definitions: Elemento raíz
  - types: Los tipos de datos que se van a transmitir
  - message: Los mensajes que se van a transmitir. Pueden ir diferentes elementos part, que indican parámetros de entrada y salida, diciendo su tipo de datos
  - portType: Las operaciones o funciones que serán soportados. Pueden ir elementos input y output que referencian a los mensajes definidos, para ver cada secuencia de operaciones
  - binding: Cómo se van a transmitir los mensajes y detalles SOAP
  - service: Dónde está localizado el servicio

Ejemplo WSDL (I)

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:Datos" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="urn:Datos" xmlns:intf="urn:Datos"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:message name="setResponse">
    <wsdl:part name="setReturn" type="xsd:string"/>
  </wsdl:message>

  <wsdl:message name="getRequest">
    <wsdl:part name="in0" type="xsd:string"/>
    <wsdl:part name="in1" type="xsd:string"/>
  </wsdl:message>

  <wsdl:message name="getResponse">
    <wsdl:part name="getReturn" type="xsd:string"/>
  </wsdl:message>

  <wsdl:message name="setRequest">
    <wsdl:part name="in0" type="xsd:string"/>
    <wsdl:part name="in1" type="xsd:string"/>
    <wsdl:part name="in2" type="xsd:string"/>
  </wsdl:message>
```

Ejemplo WSDL (II)

```
<wsdl:portType name="Datos">
    <wsdl:operation name="get" parameterOrder="in0 in1">
      <wsdl:input message="impl:getRequest" name="getRequest"/>
      <wsdl:output message="impl:getResponse" name="getResponse"/>
    </wsdl:operation>

    <wsdl:operation name="set" parameterOrder="in0 in1 in2">
      <wsdl:input message="impl:setRequest" name="setRequest"/>
      <wsdl:output message="impl:setResponse" name="setResponse"/>
    </wsdl:operation>
  </wsdl:portType>
```

Ejemplo WSDL (III)

```
  <wsdl:binding name="DatosSoapBinding" type="impl:Datos">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="get">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:Datos"
use="encoded"/>
      </wsdl:input>
      <wsdl:output name="getResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:Datos"
use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="set">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="setRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:Datos"
use="encoded"/>
      </wsdl:input>
      <wsdl:output name="setResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:Datos"
use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="DatosService">
    <wsdl:port binding="impl:DatosSoapBinding" name="Datos">
      <wsdlsoap:address location="http://localhost:8080/axis/services/Datos"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

SOAP: Introducción

- SOAP: Simple Object Access Protocol
- Es un formato de transmisión de mensajes entre Servicios Web
- En los Servicios Web, SOAP suele ir sobre HTTP, para también puede ir con otros protocolos

SOAP: Introducción

- Partes de un mensaje SOAP
  - envelope: Elemento raíz
  - header: Es opcional, no tiene porque ir información de la aplicación. Se pueden poner atributos que deciden quien debe procesar dicha información: actor, mustUnderstand, relay
  - body: Tiene contenido de la aplicación que puede ser referido a la invocación, respuesta, notificación o error
- RPC: Envío de parámetros y respuesta de resultados

Ejemplo SOAP (I)

```
<?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>
      <get
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
            <permiso
xsi:type="xsd:string">escritura</permiso>
            <parametro xsi:type="xsd:string">IP</parametro>
      </get>
 </soapenv:Body>
</soapenv:Envelope>
```

Ejemplo Servidor con AXIS y JWS (I)

```
import java.util.HashMap;
import java.util.Map;

public class Datos {
  private static HashMap datos = inicializar();
  private String dato=null;
  public static HashMap inicializar() {
    HashMap datosini= new HashMap();
    datosini.put("OS", "Windows 98");
    datosini.put("IP", "152.27.3.1");
    datosini.put("Name", "teclado");
    datosini.put("Phone", "916545949");
    datosini.put("M1", " ");
    datosini.put("M2", " ");
    datosini.put("M3", " ");
    datosini.put("M4", " ");
    return(datosini);
  }

  public String get(String clave, String parametro) {
   if (!clave.equals("lectura")&&!clave.equals("escritura")){
         dato="La clave suministrada no es correcta.";
         return(dato);
  }
  else{
     if (parametro == null || parametro.length() == 0
      || (dato = (String) datos.get(parametro)) == null) {
      dato = "No existe dicho parámetro en la base de datos.";
      return (dato);
     }
  }
  return (parametro+": "+dato);
  }
```

Ejemplo Servidor con AXIS y JWS (II)

```
public String set(String clave, String parametro, String valor) {
  if (!clave.equals("escritura")){
        dato="La clave suministrada no es correcta o sólo proporciona permiso de lectura.";
        return(dato);
 }
  else{
     if (parametro == null || parametro.length() == 0
      || (dato = (String) datos.get(parametro)) == null) {
      dato = "No existe dicho parámetro en la base de datos.";
      return(dato);
     }
     else {
        if (parametro.equals("M1")||parametro.equals("M2")||
parametro.equals("M3")||parametro.equals("M4")){
        datos.put(parametro, valor);
        return(parametro+": "+valor);
        }
        else{
       return("Este parámetro es de sólo lectura");
        }
     }
  }
 }
}
```

Ejemplo Cliente con AXIS y JWS (I)

```
package ejemplo1;

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.utils.Options;

import javax.xml.rpc.ParameterMode;

public class DatosClient
{
public static void main(String [] args) throws Exception {
String host = "http://localhost:";
String servicepath = "/axis/Datos.jws";
Options options = new Options(args);
int port = options.getPort();
String endpoint = host + port + servicepath;
String method = null;
String op1=null;
String op2=null;
String op3=null;

args = options.getRemainingArgs();

if (args == null || (!((method = args[0])).equals("get") && !method.equals("set"))) {
System.err.println("Usage:");
System.err.println(" DatosClient get parameter");
System.err.println(" DatosClient set parameter value");
return;
}
```

Ejemplo Cliente con AXIS y JWS (II)

```
if ((method.equals("get"))&&(args.length!=3))
{
 System.err.println("Número de parámetros introducido es incorrecto");
 return;
}

  if ((method.equals("set"))&&(args.length!=4))
{
 System.err.println("Número de parámetros introducido es incorrecto");
 return;
}

if (method.equals("get")) {
op1 = args[1];
op2 = args[2];
}

if (method.equals("set")) {
op1 = args[1];
op2 = args[2];
op3 = args[3];
}
```

Ejemplo Cliente con AXIS y JWS (III)

```
String ret = null;
Service service = new Service();
Call call = (Call) service.createCall();

call.setTargetEndpointAddress(new java.net.URL (endpoint));
call.setOperationName(method);

if (method.equals("set")){
call.addParameter("op1", XMLType.XSD_STRING, ParameterMode.IN);
call.addParameter("op2", XMLType.XSD_STRING, ParameterMode.IN);
call.addParameter("op3", XMLType.XSD_STRING, ParameterMode.IN);
call.setReturnType(XMLType.XSD_STRING);
ret = (String) call.invoke(new Object [] {op1, op2, op3});
}
else{
call.addParameter("op1", XMLType.XSD_STRING, ParameterMode.IN);
call.addParameter("op2", XMLType.XSD_STRING, ParameterMode.IN);
call.setReturnType(XMLType.XSD_STRING);
ret = (String) call.invoke(new Object [] {op1,op2});
}
System.out.println("Resultado : " + ret);
}
}
```

RMI: Introducción

- RMI: Remote Method Invocation
  - Es una tecnología basada en JAVA que permite a un determinado programa cliente instanciar objetos que han sido dados de alta y que existen en una máquina servidor. Una vez instanciados dichos objetos, el cliente puede ejecutar los métodos del objeto remoto que reside en otra máquina.
  - Concepto muy parecido a los servicios Web, pero aquí sólo en Java
  - Computación distribuida. Código no se ejecuta todo en la misma máquina sino en diferentes puntos de Internet
  - Métodos de objetos remotos quedan disponibles para que los ejecuten programas que tengan una instancia del objeto remoto

RMI: Diferencias con AXIS respecto al ejemplo visto

- Diferencias de RMI con AXIS en el ejemplo anterior
  - En el ejemplo de servicios web no podíamos pasar objetos como parámetros. En RMI sí se pueden pasar objetos como parámetros de la invocación remota. El servidor en principio no debe porqué conocerlo, y en ese caso RMI proporciona un mecanismo que permite descargar los bytecodes de dicha clase
  - En RMI los objetos son instanciados por parte del cliente, que coge una referencia de instancia del objeto remoto, se trabaja sobre los mismos atributos, son datos persistentes entre llamadas. En el servicio Web, un nuevo objeto se crea de la clase por cada nueva llamada a un método. Los datos no son persistentes. Se necesitaba una variable y un método estático
  - En los servicios web se accedía vía web, pero en RMI se puede acceder vía web y también acceder al sistema de ficheros
  - En los servicios web con AXIS el protocolo usado de intercambio era SOAP, mientras que en RMI se usa una tecnología diferente

RMI: Procedimiento

- En las aplicaciones RMI podemos distinguir entre programas clientes y servidores.
  - El programa servidor es el encargado de crear objetos remotos, poner accesibles mediante RMI referencias a ellos para que los programas clientes puedan acceder a sus métodos
  - El programa cliente obtiene referencias de objetos remotos que el servidor ha puesto disponibles y puede invocar sus métodos
  - Además el cliente puede pasar los bytecodes de clases que necesite el servidor porque se le hayan pasado como parámetro. RMI es la herramienta que soporta todo esto
  - Para todo ello se usa JAVA

RMI: Tutorial de motor potente de cálculo

- Tutorial de Sun de motor de cálculo
  - Motor permitirá que un determinado cliente invoque un método del servidor al que le pasará un objeto que tiene un método de una tarea que queremos que se ejecute en el servidor
  - Necesario compilar el cliente y el servidor
  - Pasos:
    - Crear la estructura de directorios y colocar los ficheros de acuerdo a los paquetes en el cliente y el servidor
    - Crear la interfaz remota
    - Compilar interfaces del servidor y dejarlos disponibles
    - Crear parte servidora
    - Compilar la clase servidora y generar los stubs y skeletons
    - Configurar fichero de seguridad java.policy
    - Compilar y ejecutar el cliente

RMI: Tutorial de motor potente de cálculo

- Tutorial de Sun de motor de cálculo: http://java.sun.com/docs/books/tutorial/rmi/
- Preguntas:
  - Si en vez de comunicar nuestro cliente con el servidor RMI que está en nuestra misma máquina, queremos comunicarnos con los servidores de nuestros vecinos. ¿Qué debemos hacer para tal fin?
  - Si queremos realizar una tarea distinta al cálculo del número pi, ¿Qué modificaciones se deberán realizar en el sistema para tal fin?
  - ¿Qué diferencias existen entre el compilador javac y el compilador rmic?
  - ¿Porqué es necesario poner el fichero java.policy? ¿Qué sucede si se ejecuta la aplicación cliente o servidora desde un directorio donde no se encuentra el fichero java.policy?
  - ¿De qué forma consigue un cliente los stubs necesarios que se encuentran en el servidor?
  - ¿Por qué el interfaz Task es serializable?
  - Hacer la aplicación anterior de Servicios Web, pero utilizando RMI

Trabajo propio del alumno asociado a la sesión

- Tutorial WSDL: http://www.w3schools.com/wsdl/default.asp
- Tutorial SOAP: http://www.w3schools.com/soap/default.asp
- Manual de AXIS: http://ws.apache.org/axis/java/user-guide.html
- Tutorial de RMI: http://download.oracle.com/javase/tutorial/rmi/

Referencias Extra

- L/D 004.738.52 WEB, Web services : concepts, architectures and applications. Alonso, Gustavo
- L/D 004.738.52 CER, Web services essentials. Cerami, Ethan
- L/S 004.738.5.057.4 SNE, Programming Web services with SOAP. Snell, James
- L/D 004.438 JAVA BUI, Building Web services with Java : making sense of XML, Soap, WSDL and UDDI. Graham, Steve
- Apache AXIS http://xml.apache.org/axis
- Documentación de Sun sobre RMI: http://java.sun.com/j2se/1.4.2/docs/guide/rmi/