Spago
Utilizzo dei Web Service

Redatto da Angelo Bernabei
Gianfranco Boccalon
Andrea Zoppello

INDICE DEI CONTENUTI

Scopo del Documento

In questo documento vengono presentate alcune linee guida su come impostare i progetti sviluppati con Spago utilizzando Eclipse, JBoss e Lomboz. In particolare viene approfondito come pubblicare i servizi sviluppati con Spago tramite il protocollo SOAP.

Informazioni sulla versione

Versione/Release n° :	1.1	**Data Versione/Release :**	29/10/2004
Descrizione modifiche:	Correzioni formali		
Versione/Release n° :	1.2	**Data Versione/Release :**	10/01/2006
Descrizione modifiche:	Aggiunta l'implementazione Axis		

1 Implementazioni disponibili

1.1 Apache SOAP

L'implementazione Apache del protocollo SOAP è la prima integrata in Spago. Tale implementazione è relativamente semplice ma è datata: infatti non riesce a gestire aspetti delle specifiche SOAP più recenti, come ad esempio gli attachment.

1.1.1 Configurazione del server

In questa sezione per server si intende un'applicazione (Spago) che espone i propri servizi applicativi attraverso il canale SOAP. Per semplicità in questo paragrafo si farà riferimento a webservices su endpoint http, quindi il "server" è un'applicazione web opportunamente configurata per servire le richieste SOAP. In seguito <CONTEXT-ROOT> indica il contesto dell'applicazione che funge da server SOAP.

L'applicazione di esempio SpagoStartup è già configurata in modo da esporre i servizi come Web Services, per cui potete far riferimento a tale applicazione per quanto riguarda le librerie e i file di configurazione.

Per la corretta di configurazione del web service è necessario:

1. Assicurarsi che siano presenti le seguenti librerie nella cartella <CONTEXT-ROOT>/WEB-INF/lib, se non presenti copiare i file dalla distribuzione di Spago.
    - soap-2.3.jar
    - activation.jar
    - mail.jar
    - spago-core-X.X.jar
    - spago-web-X.X.jar
2. Aggiungere la configurazione delle servlet e dei servlet-mapping di Apache SOAP aggiungendo la sezione seguente al file di configurazione <CONTEXT-ROOT>/WEB-INF/web.xml.

```
<context-param>
  <param-name>ConfigFile</param-name>
  <param-value>/WEB-INF/conf/soap/soap.xml</param-value>
</context-param>
<servlet>
  <servlet-name>rpcrouter</servlet-name>
  <display-name>Apache-SOAP RPC Router</display-name>
  <description>no description</description>
  <servlet-class>org.apache.soap.server.http.RPCRouterServlet</servlet-class>
  <init-param>
```

```
            <param-name>faultListener</param-name>
            <param-value>org.apache.soap.server.DOMFaultListener</param-value>
        </init-param>
    </servlet>
    <servlet>
        <servlet-name>messagerouter</servlet-name>
        <display-name>Apache-SOAP Message Router</display-name>
        <description>no description</description>
        <servlet-class>org.apache.soap.server.http.MessageRouterServlet</servlet-
class>
        <init-param>
            <param-name>faultListener</param-name>
            <param-value>org.apache.soap.server.DOMFaultListener</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>rpcrouter</servlet-name>
        <url-pattern>/servlet/rpcrouter</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>messagerouter</servlet-name>
        <url-pattern>/servlet/messagerouter</url-pattern>
    </servlet-mapping>
```

3. Copiare la cartella /WEB-INF/conf/soap fornita con Spago, nella cartella <CONTEXT-ROOT>/WEB-INF/conf (all'interno di "conf" dovrebbero trovarsi le cartelle "spago" e "soap"). Tale cartella contiene i file soap.xml e soap.ds che è un file binario contenente la descrizione dei servizi SOAP esposti ("ds" sta per "deployed services"). Tale file si modifica attraverso la pagina di amministrazione dei servizi SOAP, accessibile tramite l'URL <CONTEXT-ROOT>/admin.

4. Copiare la cartella WEB-INF/wsdl, fornita con Spago, nella cartella <CONTEXT-ROOT>/WEB-INF/wsdl. L'assenza di tale cartella non pregiudica il funzionamento dell'Adapter.

1.1.2 Configurazione del client

Per accedere ai servizi precedentemente esposti si può utilizzare una classe fornita da Spago che si chiama AdapterSOAPProxy.

Consultare il paragrafo [3.Pubblicazione del Servizio tramite WebService] per le informazioni di dettaglio relative ad un client SOAP.

1.2 Apache Axis

Come gia visto nel paragrafo precedente, Spago può esporre i servizi applicativi (ACTION o PAGE) attraverso un webservice sul canale SOAP.

Esistono tuttavia delle problematiche che non è possibile gestire in maniera standard attraverso il protocollo SOAP, tra cui la più comune riguarda la gestione di una sessione applicativa nelle chiamate verso webservice.

Per questo motivo in Spago è stato introdotto il web service AdapterAxis, che espone i servizi applicativi di spago sul canale SOAP basandosi sullo stack SOAP apache axis (http://ws.apache.org/axis/), che fornisce dei meccanismi attraverso il quale è stato possibile implementare le funzionalità del SessionContainer di spago.

Quando si utilizza questo webservice è quindi possibile scrivere dei servizi applicativi che facciano uso del session container in modo totalmente trasparente.

1.2.1 Configurazione del server

In questa sezione per server si intende un'applicazione (Spago) che espone i propri servizi applicativi attraverso il canale SOAP . Per semplicità in questo paragrafo si farà riferimento a webservices su endpoint http, quindi il "server" è un'applicazione web opportunamente configurata per servire le richieste SOAP. In seguito <CONTEXT-ROOT> indica il contesto dell'applicazione che funge da server SOAP.

Per la corretta di configurazione del web service è necessario:

5. Assicurarsi che siano presenti le seguenti librerie nella cartella <CONTEXT-ROOT>/WEB-INF/lib, se non presenti copiare i file dalla distribuzione di axis versione 1.2.1 e dalla distribuzione di spago.
    - axis.jar
    - axis-ant.jar
    - commons-discovery-0.2.jar
    - commons-logging-1.0.4.jar
    - jaxrpc.jar
    - log4j-1.2.8.jar
    - saaj.jar
    - wsdl4j-1.5.1.jar
    - spago-axis-X.X.jar
    - spago-core-X.X.jar
    - spago-web-X.X.jar

6. Aggiungere la configurazione delle servlet e dei servlet-mapping di axis aggiungendo la sezione seguente al file di configurazione <CONTEXT-ROOT>/WEB-INF/web.xml.

```
<servlet>
      <display-name>
      Apache-Axis Servlet</display-name>
      <servlet-name>AxisServlet</servlet-name>
      <servlet-class>
      org.apache.axis.transport.http.AxisServlet</servlet-class>
  </servlet>
  <servlet>
      <display-name>
      Axis Admin Servlet</display-name>
      <servlet-name>AdminServlet</servlet-name>
      <servlet-class>
      org.apache.axis.transport.http.AdminServlet</servlet-class>
      <load-on-startup>100</load-on-startup>
  </servlet>
  <servlet-mapping>
      <servlet-name>AxisServlet</servlet-name>
      <url-pattern>/servlet/AxisServlet</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
      <servlet-name>AxisServlet</servlet-name>
      <url-pattern>*.jws</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
      <servlet-name>AxisServlet</servlet-name>
      <url-pattern>/services/*</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
      <servlet-name>AdminServlet</servlet-name>
      <url-pattern>/servlet/AdminServlet</url-pattern>

  </servlet-mapping>
```

7. Copiare il file server-config.wsdd fornito con spago-axis nella cartella <CONTEXT-ROOT>/WEB-INF/ (allo stesso livello del file web.xml).

8. Copiare il file AdapterAxis.wsdl nella cartella <CONTEXT-ROOT>/WEB-INF/wsdl.

1.2.2 Configurazione del client

Un client di servizi SOAP può essere costituito da un'applicazione standalone o da un'applicazione web. Per semplicità in questo paragrafo e nel successivo si farà riferimento a client basati comunque sulla tecnologia Java.

In entrambi i casi la configurazione del client è molto semplice:

1. Assicurarsi che siano presenti le seguenti librerie nel class-path dell'applicazione (<CONTEXT-ROOT>/WEB-INF/lib nel caso di applicazioni web), se non presenti copiare i file dalla distribuzione di axis versione 1.2.1 e dalla distribuzione di spago.

    - axis.jar

- axis-ant.jar
- commons-discovery-0.2.jar
- commons-logging-1.0.4.jar
- jaxrpc.jar
- log4j-1.2.8.jar
- saaj.jar
- wsdl4j-1.5.1.jar
- spago-axis-X.X.jar
- spago-core-X.X.jar
- spago-web-X.X.jar

2. Copiare il file client-config.wsdd fornito con spago-axis nel class-path dell'applicazione.

1.2.3 Utilizzo della classe AdapterAxisProxy e gestione della sessione

Con spago è possibile invocare direttamente il webservice AdapterAxis senza entrare nei dettagli del protocollo SOAP. A questo scopo viene fornita la classe di utilità AdapterAxisProxy.

Questa classe mette a disposizione il metodo *public String service(String request)* che prende in input l'XML per la richiesta dei servizi spago e il metodo *public Long getSessionId()* per ottenere l'id di sessione generato dalla richiesta.

Il codice seguente dimostra un caso d'uso in cui la stessa istanza di proxy viene utilizzata per richiamare un servizio di business che usa la sessione, in questo caso essendo il proxy lo stesso non è necessario gestire manualmente la sessione.

```
public static void main(String[] args) {


        //
        // Example1 : Use the same proxy instance for mutiple calls.
        //                 In the case you'use the same proxy you doesnt't
        //                 need to handle manually the session ID
```

```
        AdapterAxisProxy proxy1 = null;


        StringBuffer aStringBuffer = new StringBuffer();
        aStringBuffer.append("<SERVICE_REQUEST ACTION_NAME=\"INCREMENTA_NUMERO\" />");
        try{
              proxy1 = new AdapterAxisProxy();
              proxy1.setEndpoint("http://localhost:8080/ValidationTest/services/AdapterAxis");
              String response = proxy1.service(aStringBuffer.toString());
              System.out.println(" Response = ["+response+"]");
              Thread.sleep(1000);


              response = proxy1.service(aStringBuffer.toString());
              System.out.println(" Response = ["+response+"]");
        }catch (Exception e) {
              e.printStackTrace();
        }
  }
```

Nel secondo esempio invece si dimostra come è possibile utilizzare l'identificativo di sessione ritornato dal primo proxy e utilizzare questo id per continuare l'esecuzione del servizio con la seconda istanza del proxy.

```
//
// Example2 : Using two instance of proxy you need to pass the session id,
//                     returned from proxy1 when constructing proxy2 instance
public static void main(String[] args) {


        AdapterAxisProxy proxy1 = null;
        AdapterAxisProxy proxy2 = null;
        StringBuffer aStringBuffer = new StringBuffer();
        aStringBuffer.append("<SERVICE_REQUEST ACTION_NAME=\"INCREMENTA_NUMERO\" />");
        try{
              proxy1 = new AdapterAxisProxy();
              proxy1.setEndpoint("http://localhost:8080/ValidationTest/services/AdapterAxis");
              String response = proxy1.service(aStringBuffer.toString());
              System.out.println(" Response = ["+ response +"]");
              Long idSession = proxy1.getSessionId();
              Thread.sleep(1000);
              proxy2 = new AdapterAxisProxy(idSession);
              proxy2.setEndpoint("http://localhost:8080/ValidationTest/services/AdapterAxis");
              response = proxy2.service(aStringBuffer.toString());
              System.out.println(" Response = ["+response+"]");

        }catch (Exception e) {
              e.printStackTrace();
        }
   }
```

2 Un esempio

Come primo passo realizzo una semplice pagina JSP e una Action che implementano un moltiplicatore per due di un numero per avere un semplice caso di test su cui operare.

Ecco i passi che è necessario eseguire in sequenza:

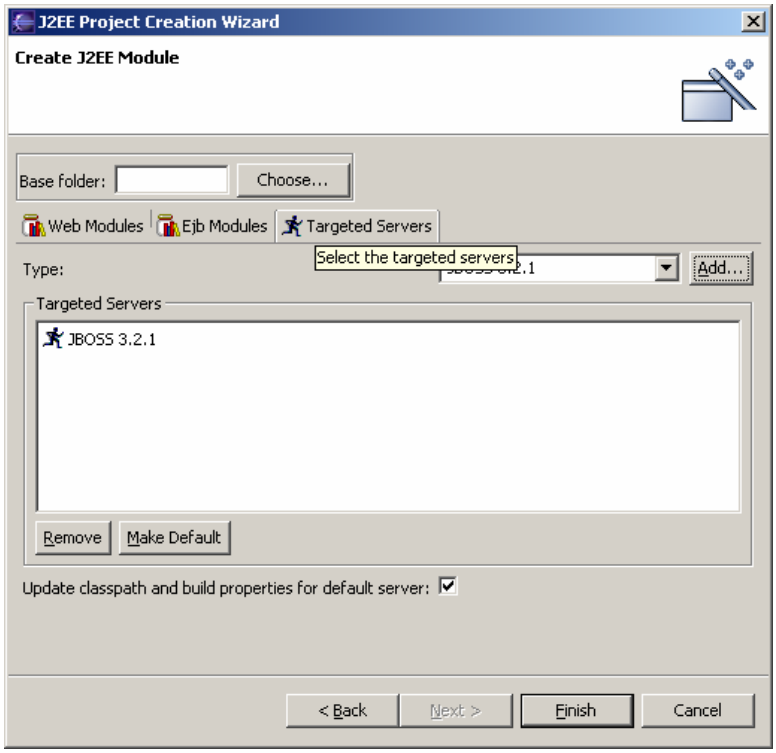
Passo 1, creare un progetto Lomboz. Dal menu di Eclipse *File.new* selezionare la voce *Project*, apparirà una finestra dove sarà possibile selezionare la tipologia del progetto,

Passo 2, selezionando la tipologia *Lomboz J2EE Wizards* troviamo sulla destra della finestra *Lomboz J2EE Project*, selezioniamo e premiamo il bottone next.

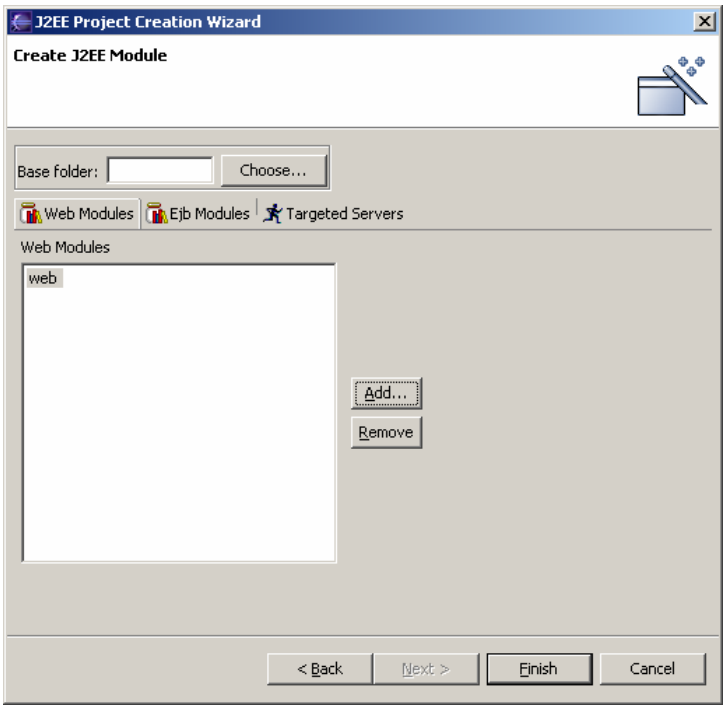
Passo 3, a questo punto viene richiesto il nome del progetto, inseriamo il nome che desideriamo e premiamo il bottone next.

Passo 4, vengono richiesti i java settings che lasciamo inalterati premendo il bottone next,

Passo 5, questo passo è dedicato alle impostazioni di Lomboz, selezioniamo il server (già configurato nelle operazioni di installazione di Lomboz),

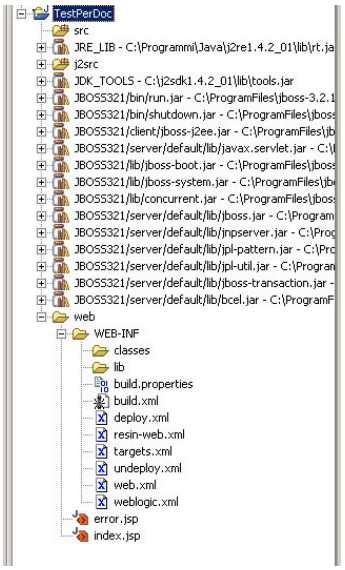


aggiungiamo un modulo web che rappresenta anche il contesto della mia web application.

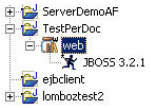


Con il bottone Finish terminiamo la creazione del progetto.

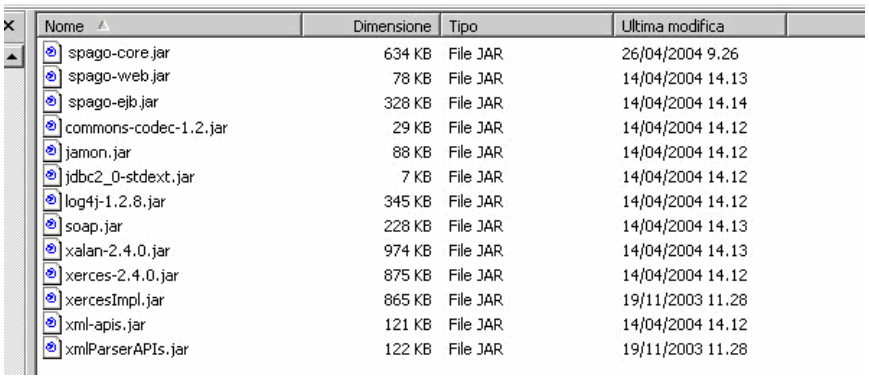
Su Eclipse sarà presente un nuovo progetto con la seguente struttura:



Come possiamo notare Lomboz utilizza Ant per tutte le operazioni di deploy su JBoss. Una volta creato il progetto, nella Lomboz J2EE View possiamo effettuare le operazioni di deploy/undeploy e di start/stop di JBoss.



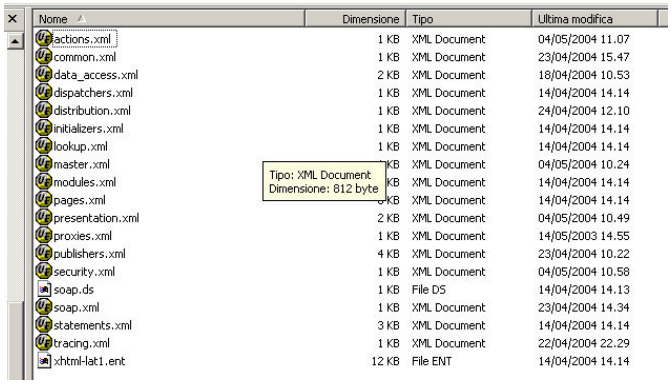
A questo punto è necessario inserire le librerie del framework (nella directory /web/WEB-INF/lib del progetto) e i suoi file di configurazione:



Nome	Dimensione	Tipo	Ultima modifica
spago-core.jar	634 KB	File JAR	26/04/2004 9.26
spago-web.jar	78 KB	File JAR	14/04/2004 14.13
spago-ejb.jar	328 KB	File JAR	14/04/2004 14.14
commons-codec-1.2.jar	29 KB	File JAR	14/04/2004 14.12
jamon.jar	88 KB	File JAR	14/04/2004 14.12
jdbc2_0-stdext.jar	7 KB	File JAR	14/04/2004 14.12
log4j-1.2.8.jar	345 KB	File JAR	14/04/2004 14.12
soap.jar	228 KB	File JAR	14/04/2004 14.13
xalan-2.4.0.jar	974 KB	File JAR	14/04/2004 14.13
xerces-2.4.0.jar	875 KB	File JAR	14/04/2004 14.12
xercesImpl.jar	865 KB	File JAR	19/11/2003 11.28
xml-apis.jar	121 KB	File JAR	14/04/2004 14.12
xmlParserAPIs.jar	122 KB	File JAR	19/11/2003 11.28

aggiungendo le librerie al classpath del progetto di Eclipse.

Ecco i vari file di configurazione sotto la directory /web/WEB-INF/conf:



Nome	Dimensione	Tipo	Ultima modifica
actions.xml	1 KB	XML Document	04/05/2004 11.07
common.xml	1 KB	XML Document	23/04/2004 15.47
data_access.xml	2 KB	XML Document	18/04/2004 10.53
dispatchers.xml	1 KB	XML Document	14/04/2004 14.14
distribution.xml	1 KB	XML Document	24/04/2004 12.10
initializers.xml	1 KB	XML Document	14/04/2004 14.14
lookup.xml	1 KB	XML Document	14/04/2004 14.14
master.xml	KB	XML Document	04/05/2004 10.24
modules.xml	KB	XML Document	14/04/2004 14.14
pages.xml	KB	XML Document	14/04/2004 14.14
presentation.xml	2 KB	XML Document	04/05/2004 10.49
proxies.xml	1 KB	XML Document	14/05/2003 14.55
publishers.xml	4 KB	XML Document	23/04/2004 10.22
security.xml	1 KB	XML Document	04/05/2004 10.58
soap.ds	1 KB	File DS	14/04/2004 14.13
soap.xml	1 KB	XML Document	23/04/2004 14.34
statements.xml	3 KB	XML Document	14/04/2004 14.14
tracing.xml	1 KB	XML Document	22/04/2004 22.29
xhtml-lat1.ent	12 KB	File ENT	14/04/2004 14.14

Tipo: XML Document
Dimensione: 812 byte

Aggiungere il file web.xml allegato alla distribuzione del framework.

E' inoltre necessario modificare la directory dove Eclipse posiziona il file .class generati durante la fase di compilazione in */web/WEB-INF/classes*, questa operazione può essere fatta dalla finestra delle *property* del progetto.

Si rimanda alla documentazione del framework la descrizione in dettaglio dei singoli file di configurazione xml.

Definiamo il servizio di test configurando i seguenti file xml:

action.xml

```
<ACTION class="it.eng.spago.test.Sommatore" distributed="FALSE" name="Sommatore" scope="REQUEST">
    <CONFIG />
</ACTION>
```

presentation.xml

```
<MAPPING business_name="Sommatore" business_type="ACTION" publisher_name="SommatoreP" />
```

publisher.xml

```
<PUBLISHER name="SommatoreP">
    <RENDERING channel="HTTP" mode="FORWARD" type="JSP">
```

```
                <RESOURCES>
                    <ITEM prog="0" resource="/index.jsp" />
                </RESOURCES>
            </RENDERING>
        </PUBLISHER>
```

La classe seguente implementa l'action:

```
package it.eng.spago.test;

import it.eng.spago.base.SourceBean;
import it.eng.spago.dispatching.action.AbstractAction;

/**
 * @author bernabei
 */
public class Sommatore extends AbstractAction{


    public void service(SourceBean request, SourceBean response) throws Exception {
        String primo=(String)request.getAttribute("primo");
        String secondo=(String)request.getAttribute("secondo");
        int risultato=Integer.parseInt(primo)+ Integer.parseInt(secondo);
        ValueObjectTest vo=new ValueObjectTest();
        vo.setMsg(Integer.toString(risultato));
        response.setAttribute("risultato",vo);
    }
}
```

E' stata creata anche la seguente classe di utilità:

```
package it.eng.spago.test;

/**
 * @author bernabei
 */
public class ValueObjectTest {
private String msg="";
/**
 * @return
 */
```

```
public String getMsg() {
        return msg;
}

/**
 * @param string
 */
public void setMsg(String string) {
        msg = string;
}
public String toString() {
        return msg;
}
}
```

Il risultato sarà pubblicato tramite la seguente pagina JSP indicata nel file di configurazione dei publisher:

```
<%@ page
                        contentType="text/html; charset=ISO-8859-1"
                        pageEncoding="ISO-8859-1"
                        session="true"
                        import="it.eng.spago.base.*,it.eng.spago.test.ValueObjectTest"
%>

<%
ValueObjectTest risultato=null;
ResponseContainer responseContainer=ResponseContainerAccess.getResponseContainer(request);
if (responseContainer!=null){
        SourceBean sb =         responseContainer.getServiceResponse();
        if (sb!=null) risultato = (ValueObjectTest)sb.getAttribute("risultato");
}
if (risultato==null) risultato=new ValueObjectTest();
%>
<html>
        <head>
                <title>Sommatore...</title>
        </head>
        <body>
        <form name="forem" method="post" action="./servlet/AdapterHTTP?ACTION_NAME=Sommatore">
                <center>SOMMATORE...</center>
                Primo Numero : <input type="text" name="primo">
                Secondo Numero : <input type="text" name="secondo">
```

```
                <input type="submit" class="submit" value="Somma">
                </form>
                Risultato= <input type="text"  name=”risultato” value='<%=risultato.getMsg()%>'>
        </body>
</html>
```

A questo punto verifichiamo la correttezza del lavoro eseguendo una prova lanciando JBoss ed effettuando il deploy dell'applicazione di test appena creata operando dalla *Lomboz J2EE View*.

Scrivendo la seguente URL nel browser appare la pagina dove è presente il risultato:

http://localhost:8080/web/servlet/AdapterHTTP?ACTION_NAME=Sommatore&primo=2&secondo=2&NEW_SESSION=TRUE

3 Pubblicazione del Servizio tramite WebService

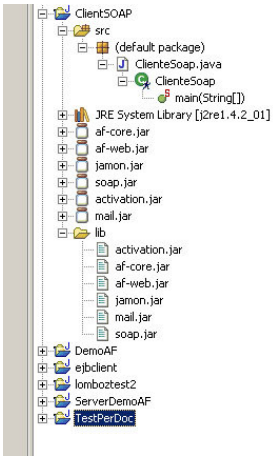
Per accedere al servizio da un client SOAP non dobbiamo modificare la configurazione appena impostata perché il framework automaticamente pubblica tutti i servizi come WebService, dobbiamo solo verificare il corretto funzionamento del server SOAP accertandosi che:

1. all'interno della directory /conf/soap siano presenti i file soap.ds e soap.xml. Il file soap.ds è generato dal componente integrato nel framework (Apache Soap).
2. Copiare nella directory /web l'applicativo di amministrazione del server Apache SOAP (/admin)
3. per verificare se il servizio SOAP è attivo lanciare l'URL http://localhost:8080/web/admin/index.html
4. Nel caso il WebService non sia attivo è possibile installarlo utilizzando l'applicativo citato precedentemente inserendo le seguenti informazioni:

Deployed Service Information

'http://tempuri.org/it.eng.spago.dispatching.soapchannel.AdapterSOAP' Service Deployment Descriptor	
Property	**Details**
ID	http://tempuri.org/it.eng.spago.dispatching.soapchannel.AdapterSOAP
Scope	Session
Provider Type	Java
Provider Class	it.eng.spago.dispatching.soapchannel.AdapterSOAP
Use Static Class	False
Methods	service
Type Mappings	
Default Mapping Registry Class	

Per lanciare il servizio è necessario creare un progetto java con un piccolo client utilizzando le seguenti librerie:



- activation.jar
- spago-core.jar
- spago-web.jar
- mail.jar
- soap.jar

Il servizio si invoca con la seguente classe client:

```
import java.net.URL;


import it.eng.spago.dispatching.soapchannel.AdapterSOAPProxy;


/**
 * @author bernabei
 */
public class ClienteSoap {


	public static void main(String[] args) {
		try {
			AdapterSOAPProxy adapterSOAP = new AdapterSOAPProxy();
			String request =
				"<SERVICE_REQUEST ACTION_NAME=\"Sommatore\" primo=\"2\" secondo=\"2\"
NEW_SESSION=\"TRUE\"/>";
			System.out.println("AdapterSOAPClient::main: request\n" + request);
			URL url=new URL("http://localhost:8080/web/servlet/rpcrouter");
			adapterSOAP.setEndPoint(url);
```

```
                String response = adapterSOAP.service(request);
                System.out.println(
                        "AdapterSOAPClient::main: response\n" + response);
            } // try
            catch (Exception ex) {
                System.out.println(
                        "AdapterSOAPClient::main: errore in adapterSOAPStub.service(request)");
                ex.printStackTrace();
            } // catch (Exception ex) try
        } // public static void main(String[] args)
}
```

Il risultato è il seguente:

```
<RESPONSE>
  <SERVICE_RESPONSE risultato="4"/>
  <ERRORS/>
</RESPONSE>
```