# Spago
# Utilizzo dei Web Service

Redatto da Angelo Bernabei

# 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 sia possibile pubblicare i servizi sviluppati con il framework tramite il protocollo SOAP.

## *Informazioni sulla versione*

| | | | |
|---|---|---|---|
| **Versione/Release n°** : | 1.1 | **Data Versione/Release** : | 29/10/2004 |
| **Descrizione modifiche:** | Correzioni formali | | |

# 1 Caso di test

Come primo passo realizzo una semplice pagina JSP e una Action che eseguono 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:

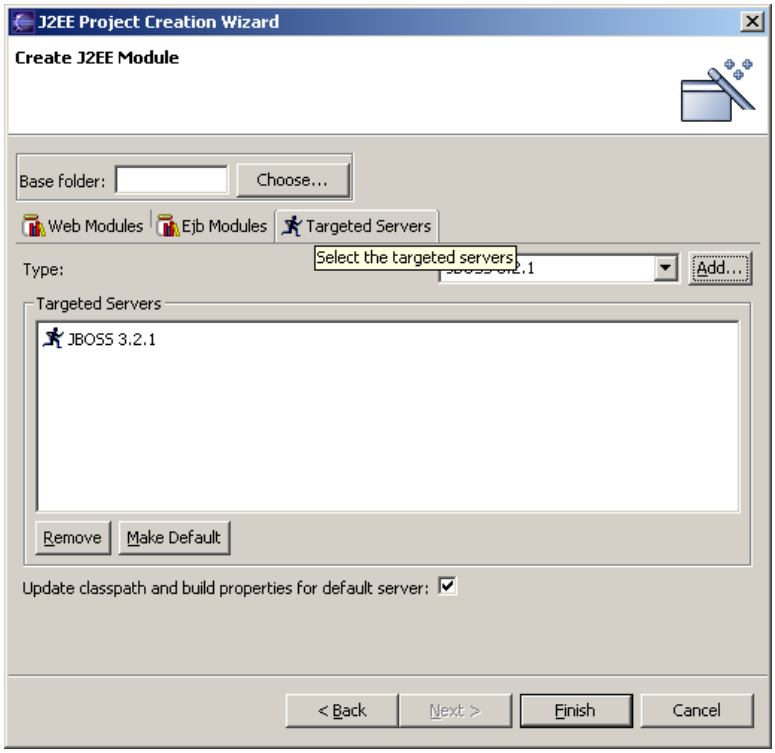
**Passo 1,** creare un progetto Lomboz. Dal menu di Eclipse *File.new* selezionare la voce *Project*, apparirà una finestra dove sarà possibile selezionare la tipologia del progetto,

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

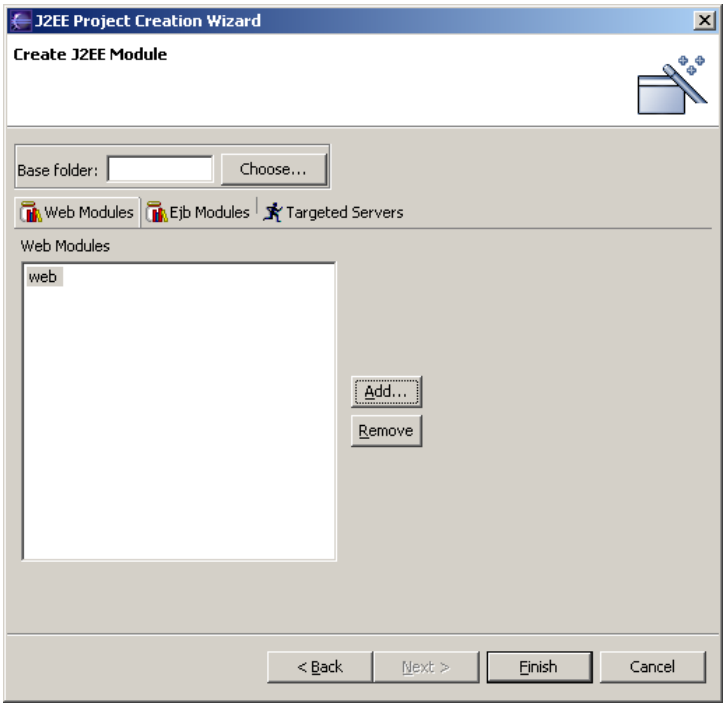
**Passo 3,** a questo punto viene richiesto il nome del progetto, inseriamo il nome che desideriamo e premiamo il bottone next.

**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 ),

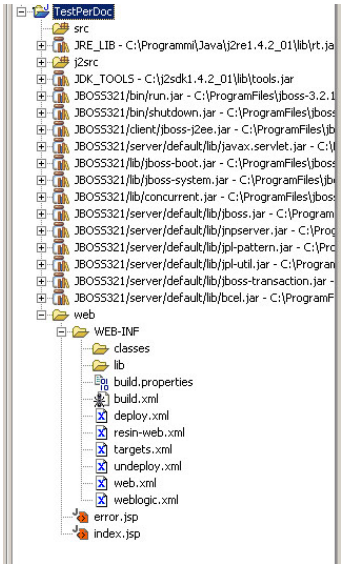



aggiungiamo un modulo web che rappresenta anche il contesto della mia web application.



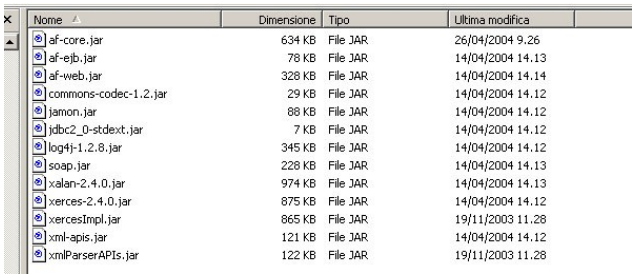

Con il bottone Finish terminiamo la creazione del progetto.

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.

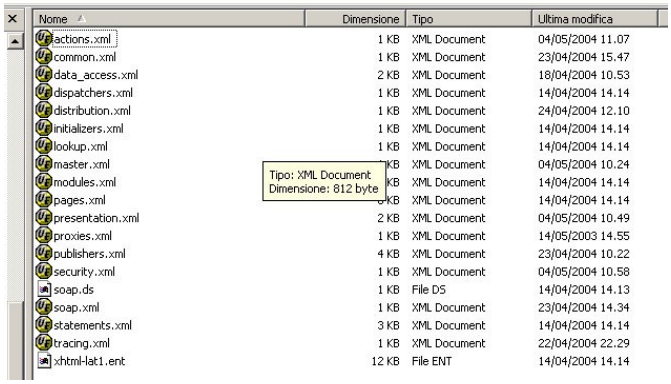
A questo punto è necessario inserire le librerie del framework (nella directory /web/WEB-INF/lib del progetto) e i suoi file di configurazione:



| Nome | Dimensione | Tipo | Ultima modifica |
|---|---|---|---|
| af-core.jar | 634 KB | File JAR | 26/04/2004 9.26 |
| af-ejb.jar | 78 KB | File JAR | 14/04/2004 14.13 |
| af-web.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 avendo cura di verificare la correttezza del parametro AF_ROOT_PATH.

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" />
```

*publischer.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

# 2 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:

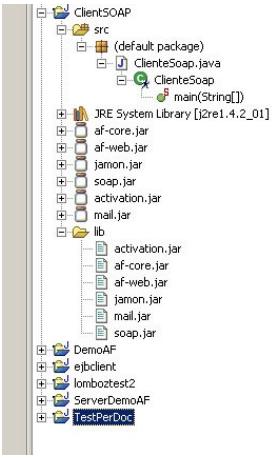
1. all'interno della directory /conf siano presenti i file soap.ds e soap.xml. Il file soap.ds è generato dal componete integrato nel framework ( Apache Soap ).
2. Copiare nella directory /web l'applicativo di amministrazione del server Apache SOAP ( /admin)
3. per verificare se il servizio SOAP è attivo 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>
```