http://www.jugsardegna.org

Java Web Services con Axis

III Parte

di

Nicola Mura

nikhol@tiscali.it

Introduzione

Nei due precedenti articoli (1),(2) sono stati affrontati alcuni aspetti dell'impiego del tool Apache Axis (3) per la creazione/gestione dei Web services. Con questi concetti è quindi possibile scrivere ex novo applicazioni raggiungibili da ogni punto del Web e con qualsiasi tecnologia: .Net, php, etc. Ora, però, è necessario voltarsi un attimo e soffermarsi sull'utilità dei Web services e sul dilagante successo di SOAP. Infatti bisogna tenere presente l'attitudine all'integrazione dei Web services per cui, ipoteticamente, nessuna applicazione già scritta deve essere buttata via per ricrearla con i nuovi standard. Il tema dell'integrazione rappresenta, in sintesi, il vero stimolo che ha portato all'affermazione delle tecnologie XML, SOAP, WSDL (4) come substrato della filosofia Web services.
Su queste considerazioni si basa questo articolo che ha l'obiettivo di mostrare il lato di integrazione di Axis verso l'architettura J2EE (5).

L'esempio (Esporre un EJB come Web service)

Questo semplice esempio tende a sottolineare l'aspetto di integrazione che i Web services in generale portano con se ed è indirizzato a coloro che hanno investito risorse nel realizzare applicazioni basate sull'architettura J2EE e che ora hanno la necessità di estendere lo strato EJB dotandolo di accesso alla rete.
In questo esempio utilizzeremo le potenzialità di deploy di Axis con cui risulta praticamente indifferente attivare un servizio implementando direttamente le classi java o sfruttando uno strato EJB preesistente.

Supponiamo che si voglia esporre come WS l'EJB *MiaClasseBean* la quale potrebbe rappresentare, per esempio, il Session Bean (6) di facciata di uno strato sottostante anche estremamente complesso di logica business. In **figura 1** viene mostrato uno schema ad alto livello del possibile scenario preso in considerazione:

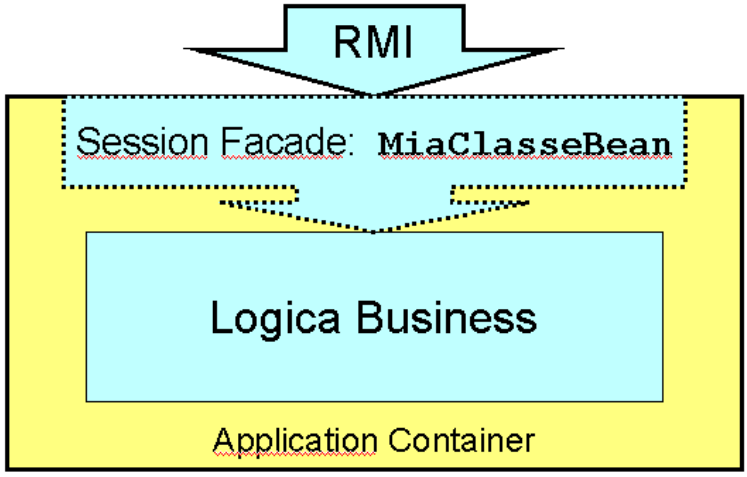

Figura 1 –Schema di Applicazione J2EE con Session Facade

L'esempio descrive il caso in cui sia stato impiegato il pattern Session Facade (7) nell'architettura dell'applicazione J2EE il cui ciclo di vita è gestito all'interno di un Application Container (Jboss,JOnAS,etc.). Come si nota in **figura 1** il grado di accessibilità del session bean stateless *MiaClasseBean*, è fornito dalle caratteristiche d'impiego del protocollo RMI (8) ma tipicamente non va oltre l'ambito di rete interna.

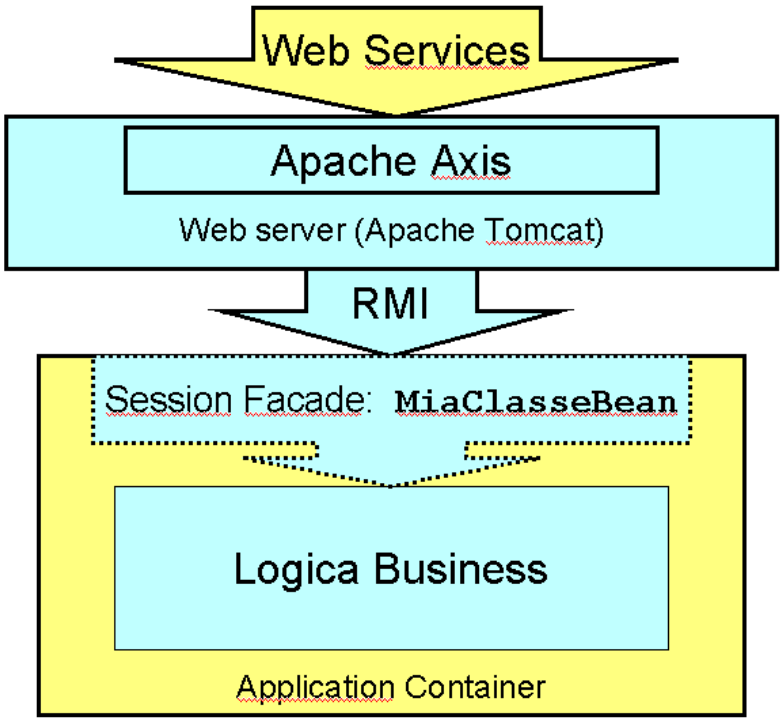

Figura 2 –Schema di sovrapposizione dello strato Web services Axis al Session Facade

In **figura 2** viene invece mostrato come l'aggiunta dell'ulteriore strato logico fornito dal Web server Tomcat (9) e dal Web service engine Axis (tramite l'operazione di deployment), renda universalmente raggiungibile la session facade *MiaClasseBean* a generici client Web Services senza vincoli tecnologici e di linguaggio.

Per semplicità ipotizziamo che le interfacce "remote" e "home" dell'EJB *MiaClasseBean* siano quelle mostrate nei **listati 1, 2**:

Listato 1 – Interfaccia MiaClasse (remote interface)

```
package miopackage;

import java.rmi.*;
import javax.ejb.*;
import miopackage.vo.* ;

public interface MiaClasse extends EJBObject{

  public RisultatoVO getRisultato(MioVO  vo);

}
```

Listato 2 – Interfaccia MiaClasseHome (home interface)

```
package miopackage;

import java.rmi.*;
import javax.ejb.*;

public interface MiaClasseHome extends EJBHome {

  public MiaClasse create() throws RemoteException, CreateException;

}
```

Praticamente si vuole consentire la chiamata del metodo *getRisultato()* da qualsiasi client sparso nella rete. Si tratta quindi di "realizzare" un servizio richiamabile tramite RPC ovvero attraverso un meccanismo sincrono request/response. Ciò è reso possibile da Axis grazie ad una semplice operazione di deployment!

Ora costruiamo il file di deployment che chiamemeremo, per esempio, deploy3.wsdd (wsdd sta per Web Service Deploy Description), in base alle informazioni possedute circa la nostra EJB.

Listato 3 – deploy3.wsdd

```
<deployment
  xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <service name="urn:nomeServizio" provider="java:EJB">
    <parameter name="jndiURL" value="jnp://nomehost:1099"/>
    <parameter name="jndiContextClass" value="org.jnp.interfaces.NamingContextFactory"/>
    <parameter name="beanJndiName" value="nomeJNDImiaEjb"/>
    <parameter name="homeInterfaceName" value="miopackage.MiaClasseHome"/>
    <parameter name="remoteInterfaceName" value="miopackage.MiaClasse"/>
    <parameter name="allowedMethods" value="getRisultato"/>
    <parameter name="scope" value="Session"/>
    <beanMapping qname="myNS1:MioVO" xmlns:myNS1=" urn:nomeServizio"
                 languageSpecificType="java:miopackage.vo.MioVO"/>
    <beanMapping qname="myNS2: RisultatoVO " xmlns:myNS2=" urn:nomeServizio"
                 languageSpecificType="java:miopackage.vo. RisultatoVO "/>

  </service>
</deployment>
```

Rispetto agli articoli precedenti vi sono alcuni punti di differenza riscontrabili sul file di deployment. A parte l'attributo *name* che mantiene il consueto significato di nome del servizio, l'attributo *provider* viene ora impostato con il valore *java:EJB*. Ciò indica ad Axis che si tratta del deployment di una EJB e che deve dunque attendersi ulteriori specifici paramentri:

- l'URL su cui risponde il bean tramite protocollo RMI (attributo *jndiURL*),
- la factory per il contesto JNDI (10) (attributo *jndiContextClass*),
- Il nome JNDI con cui è "deployato" il bean all'interno dell'application container (attributo *beanJndiName*),
- Il nome dell'interfaccia home necessaria ad effettuare il look up del bean (attributo *homeInterfaceName*),
- il nome dell'interfaccia remote (attributo *remoteInterfaceName*).

Il parametro *scope* è comune agli altri esempi, mentre gli elementi *beanMapping* sono necessari ad indicare l'impiego dei tipi complessi *MioVO* e *RisultatoVO* come descritto nell'articolo precedente (2).
Prima di lanciare lo script per l'esecuzione del deployment occorre inserire sotto la directory:

```
WEB-INF\classes
```

i file compilati delle interfacce MiaClasseHome, MiaClasse e del tipo complesso di ritorno RisultatoVO.

Sotto:

`WEB-INF\lib`

Occorre, invece inserire le librerie:

- jboss-client.jar
- jboss-j2ee.jar
- jbossmq-client.jar
- jbosssx-client.jar
- jndi.jar
- jnp-client.jar

con cui Axis è in grado di interrogare l'application container su cui gira l'EJB tramite protocollo RMI.
L'esecuzione del deployment produrrà gli stessi effetti già illustrati in precedenza (1), (2) che sono:

- l'attivazione del servizio di esecuzione RPC,
- la pubblicazione del documento WSDL che descrive il servizio, all'URL:

  `http://localhost:8080/axis/services/urn:nomeServizio?wsdl`

Occorre sottolineare che per entrambe le fasi viene perso il carattere di provenienza EJB. Dunque, le modalità di accesso al servizio (per esempio tramite client DII (11)) e la tipologia della descrizione WSDL, sono analoghe a quelle descritte nei precedenti articoli.

Conclusioni

In questo articolo è stato mostrato un altro aspetto fondamentale delle potenzialità offerte da Apache Axis; quello che riguarda la capacità di integrare applicazioni che implementano l'architettura J2EE, tramite una semplice operazione di deployment. Ciò rispecchia la filosofia Web Services che recita - non riscrivere l'applicazione, integrala! - ove per integrazione si intende anche solamente rendere accessibili al Web applicazioni preesistenti in ambito LAN (si pensi all'azienda che apre una filiale remota).

Bibliografia

(1) David A. Chappell & Tyler Jewell, "Java Web Services", O'Reilly HOPS, 2002
(2) W.Crawford, J.Kaplan, "J2EE Design Patterns", O'Reilly, 2003
(3) Sun, The J2EE 1.4 Tutorial, 2003: http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

Riferimenti

(1) Nicola Mura, "Java Web Services con Axis – I Parte", Programmazione.it, 2003: http://www.programmazione.it/index.php?entity=earticle&idArticle=1405&idArea=18
(2) Nicola Mura, "Java Web Services con Axis – II Parte", Programmazione.it, 2004: http://www.programmazione.it/index.php?entity=earticle&idArticle=1464&idArea=18
(3) Apache Axis: http://ws.apache.org/axis/
(4) Web Services Descritpion Language (WSDL): http://www.w3.org/TR/wsdl
(5) Sun, Java 2 Enterprise Edition (J2EE): http://java.sun.com/j2ee/
(6) Session Bean: http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts3.html
(7) Fabrizio Gianneschi, "Recensione: J2EE Design Patterns", Programmazione.it, 2003: http://www.programmazione.it/index.php?entity=enews&idNews=8448&idArea=18
(8) Remote Method Invocation (RMI): http://java.sun.com/products/jdk/rmi/
(9) Apache Tomcat: http://jakarta.apache.org/tomcat/
(10) Java Naming and Directory Interface (JNDI): http://java.sun.com/products/jndi/
(11) Java API for XML-Based RPC (JAX-RPC) : http://java.sun.com/xml/jaxrpc/