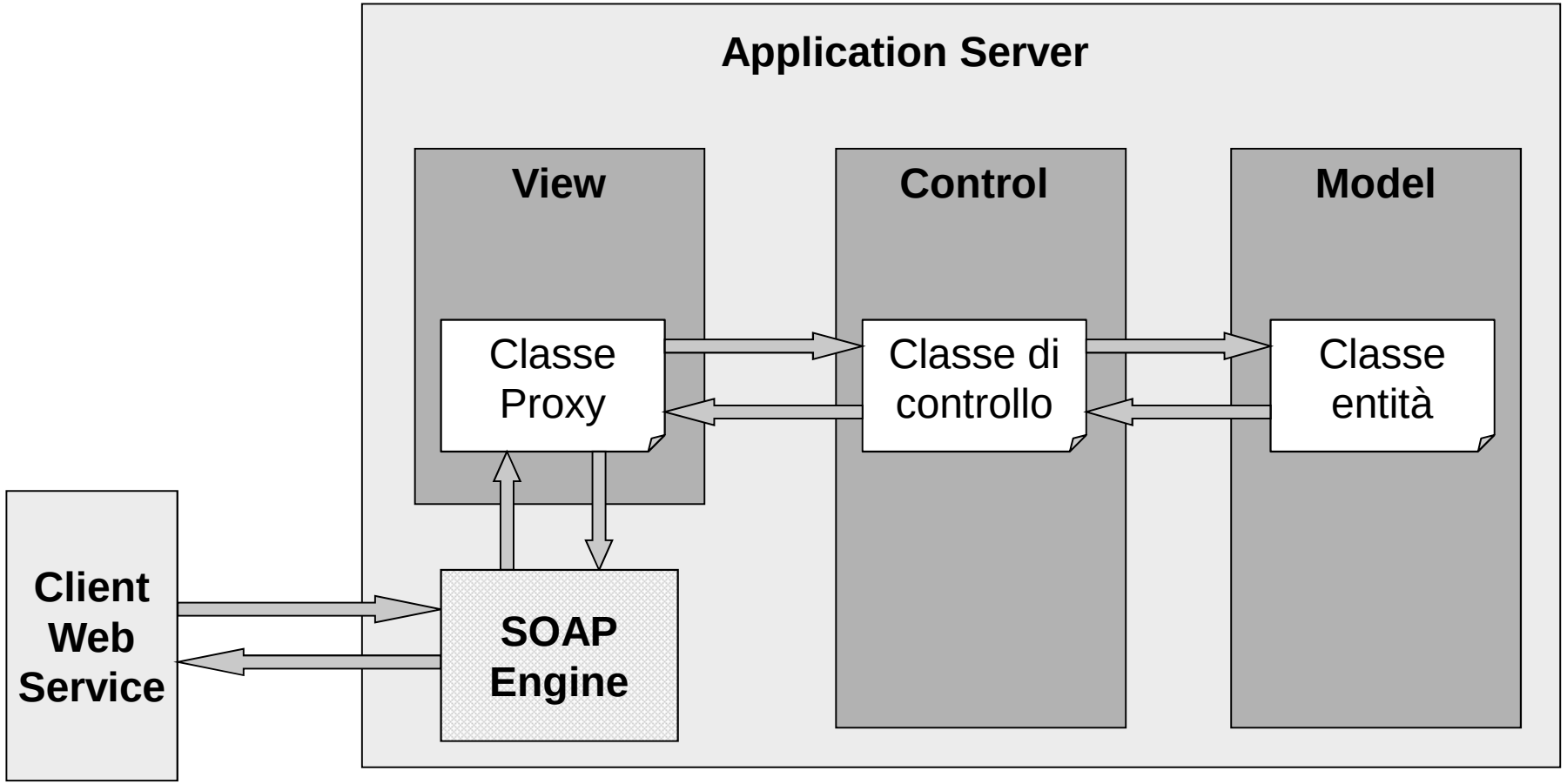
MEMO


Application Server
View
Control
Model
Classe Proxy
Classe di controllo
Classe entità
Client Web Service
SOAP Engine

Programmazione dei Web Service intorno al 2007: Esempi

Stefano Scarrone – Gianfranco Pesce – Giovanni Cantone

Sommario:

Esempio 1

- **Descrizione applicazione service oriented ProvaWS_1**
- **Dettagli implementativi di ProvaWS_1**
- **Applicazione ProvaWS_1client (web application di test)**

Esempio 2

- **Descrizione applicazione service oriented ProvaWS_2**
- **Dettagli implementativi di ProvaWS_2**
- **Applicazione ProvaWS_2client (web application di test)**

Esempio 1 – ProvaWS_1

Scopo: interfacciare una applicazione java con il SOAP engine Axis di Apache, esporre un servizio web, utilizzare le Remote Procedure Call

Struttura: una classe di controllo e una classe proxy

Descrizione: servizio web che prende in input una Stringa e restituisce una Stringa. Se la Stringa di input è "`italiano`" il servizio restituisce la Stringa "`Ciao Mondo!`" altrimenti restituisce "`Hello World!`"

Esempio 1 – ProvaWS_1

Funzionamento (il controller)

Il nucleo dell'applicazione risiede ovviamente all'interno della classe di controllo '`ControllProvaWS`' ed è rappresentato dal metodo:

```
public String saluto(String lingua)
```

Se la Stringa di input è "`italiano`" il metodo restituisce la Stringa "`Ciao Mondo!`" altrimenti restituisce "`Hello World!`"

Esempio 1 – ProvaWS_1

Funzionamento (il proxy)

Per poter trasformare il metodo del controllore in un servizio è stato necessario introdurre una classe **`Proxy`** che funga da proxy tra il controllore e il SOAP engine, Axis. Tale classe rappresenta lo strato View.

All'interno di `Proxy` vi è l'operazione:

```
public String saluto(String lingua);
```

che sarà il punto di accesso al servizio; difatti quando Axis riceverà una richiesta per il servizio invocherà proprio questo metodo di **`Proxy`**.

Una volta invocato questo metodo, viene istanziato il controllore (specifico) e ne viene chiamato il metodo '**`saluto`**'.

Esempio 1 – ProvaWS_1

Deploy di un servizio sotto Axis

Avvio

Spostare tutti i file compilati (.**`class`**) dell'applicativo sotto la cartella '**`classes`**' di Axis, mantenendo ovviamente la struttura dei package (albero di cartelle). In questo modo Axis avrà libero accesso all'interfaccia della classe **`Proxy`**.

NB: Tale operazione viene effettuata in automatico da Eclipse utilizzando, in input a un tool di costruzione (e.g., Ant), un file di direttive (**`build.xml`**) (oggi anche Maven? **`pom.xml`**), che può essere messo nel progetto, nel ns. caso **`ProvaWS_1`**.

Esempio 1 – ProvaWS_1

Completamento del deploy di un servizio sotto Axis

A questo punto, bisogna informare Axis che, a una tal richiesta di servizio, deve associare la tal classe **`Proxy`** (esposizione di un servizio) e invocarne il tal metodo.

In altre parole, è necessario completare il deployment del servizio.

A tale scopo si istruisce l'engine di Axis affinché, in corrispondenza di una determinata richiesta (sottoforma di messaggio SOAP), esso istanzi la classe e ne richiami il relativo metodo. MEMO:

WA: HTTP(link.jsp) → ServletContainer.Dispatcher → Servlet (HTTPMessage) →Ret→ HTML

WS: HTTP(SOAPMessage) → SOAPEngine.Dispatcher → Proxy (SOAPMessage)→Ret→
→ SOAPMessage

Esempio 1 – ProvaWS_1

Deploy di un servizio sotto Axis

In generale “deployare” un servizio richiede la stesura di un file xml di deployment che nel nostro caso è ProvaWS_1.wsdd e contiene :

```
<deployment xmlns= “http://xml.apache.org/axis/wsdd/”
xmlns:java=“http://xml.apache.org/axis/wsdd/providers/java">
 <service name=“urn:ProvaWS_1" provider="java:RPC" >
  <parameter name="className" value="isp_0708.ProvaWS_1.proxy.ProvaWS1"/>
  <parameter name="allowedMethods" value="saluto"/>
  <parameter name="scope" value="Request"/>
 </service>
</deployment>
```

Esempio 1 – ProvaWS_1

```
<service name=“urn:ProvaWS_1" provider="java:RPC" >
```

L’attributo **`name`** rappresenta lo uniform resource name, cioè il nome che noi vogliamo assegnare al servizio e che servirà ad identificarlo univocamente.
L’attributo **`provider`** è stato valorizzato con “**`java:RPC`**” che indica ad Axis di pubblicare il servizio secondo un meccanismo di comunicazione RPC.

Esempio 1 – ProvaWS_1

Deploy di un servizio sotto Axis

Di seguito troviamo una sezione composta da tre tag <parameter>.

- **`<parameter name="className" value="isp_0708.ProvaWS_1.proxy.ProvaWS1"/>`** indica il nome (comprensivo di package) della classe **`Proxy`**.
- **`<parameter name="allowedMethods" value="saluto"/>`** descrive i metodi della suddetta classe da esporre (nel nostro caso l'unico metodo implementato 'saluto').
- **`<parameter name="scope" value="Request"/>`** definisce il ciclo di vita dell'oggetto proxy particolare, utilizzato e distrutto ad ogni richiesta, '**`Request`**' (altri valori possibili sono **`Session`** e **`Application`**).

Esempio 1 – ProvaWS_1

Lancio del deploy di un servizio sotto Axis

Per Lanciare il deployment occorre inserire nel classpath delle librerie (*axis.jar ; jaxrpc.jar ; saaj.jar ; commons-logging.jar ; commons-discovery.jar ;log4j-1.2.8.jar*)

e richiedere la esecuzione del seguente comando:

```
java org.apache.axis.client.AdminClient ProvaWS_1.wsdd
```

Esempio 1 – ProvaWS_1client

Funzionamento

Per testare il buon funzionamento del servizio creato è necessario sviluppare un'applicazione client *ad hoc* quale 'ProvaWS_1client'.

ProvaWS_1client potrebbe essere una mini Web application composta solo da una pagina JSP per l'inserimento dei dati e la visualizzazione dei risultati e da una classe Java Bean che, appoggiandosi alle librerie di Axis, effettua una RPC al servizio.

Il lato interessante di questa mini applicazione è incentrato appunto su come si effettua una RPC.

Esempio 1 – ProvaWS_1client

Funzionamento

Per effettuare una RPC si deve istanziare un oggetto di tipo **`Call`** passandogli la URL del Web service:

```
cll = new Call(new URL("http://localhost:8080/axis/services/"));
```

si deve poi inserire il nome del servizio e quello del metodo da chiamare

```
cll.setOperationName(new QName("urn:ProvaWS_1", "saluto"));
```

infine, bisogna invocare il metodo **`invoke`** passandogli un'array di `Object`: questo array rappresenta i parametri del metodo chiamato. Una volta eseguito il servizio, il metodo `invoke` restituirà il valore/oggetto a sua volta restituito dal metodo remoto chiamato.

```
String res = (String)cll.invoke(new Object[]{this.lingua});
```

Esempio 2 – ProvaWS_2

Scopo: interfacciare una applicazione java con Axis, esporre un servizio web, utilizzare le Remote Procedure Call, serializzare tipi di dato complessi, gestire le Eccezioni sollevate da una RPC, interfacciare l'applicazione con il DBMS Postgresql.

Struttura: una classe entità, una classe di controllo, una classe proxy, un'insieme di classi per interfacciarsi con il DBMS, una classe di eccezione.

Descrizione: servizio web che prende in input due Stringhe (username e password) e restituisce un oggetto di tipo 'Titolare' (Utente in MM). Il sistema effettua una connessione al DBMS e controlla se nella tabella 'titolare' esiste un record con i valori di username e password inseriti in input. Se esiste, restituisce un oggetto di tipo Titolare contenente tutti i dati del record, in caso contrario solleva un'eccezione (`null` in MM).

Esempio 2 – ProvaWS_2

Funzionamento (il model)

Lo strato Model è composto esclusivamente da una classe, ‘Titolare’; [dovrebbe esserci altresì la ‘TitolareBean’]; questa deve rispettare alcune specifiche dovendo essere serializzata: deve rispettare lo standard dei Bean java. (MEMO Prototype pattern)

- Costruttore senza parametri

- Ogni campo deve avere un metodo per l’operazione di set con segnatura:

```
public void set[nome](String var)
```

  dove [nome] rappresenta il nome del campo con la prima lettera maiuscola

- Ogni campo deve avere un metodo per l’operazione di get con segnatura:

```
public String get[nome]()
```

  dove [nome] rappresenta il nome del campo con la prima lettera maiuscola

Tali metodi sono infatti utilizzati dal SOAP engine per “re-settare” l’oggetto una volta deserializzato (da XML a oggetto java).

Esempio 2 – ProvaWS_2

Funzionamento (classi per il DBMS)

All'interno del package 'isp_0708.ProvaWS_2.db' sono presenti quattro classi che implementano il modello DAO (Database Access Object).

J2EEDAO carica le impostazioni per la connessione al db da un file property; gestisce le transazioni;

SQLDAO interfaccia che permette di disaccoppiare l'applicazione dalla specifica base di dati (facilita la modifica del DBMS e.g. da Postgresql a Oracle)

PostgresqlDAO classe che implementa l' SQLDAO relativa allo specifico DBMS Postgresql

DAOFactory carica dinamicamente la classe atta ad implementare l'SQLDAO (e.g. PostgresqlDAO).

Esempio 2 – ProvaWS_2

Funzionamento (il controllore)

Il nucleo dell'applicazione risiede, come già detto, all'interno delle classi di controllo, nel ns. caso solo 'ControllProvaWS' e la sua sola operazione:

```
public Titolare login(String username, String password);
```

Tale operazione effettua una connessione al DBMS e controlla se nella tabella 'titolare' esiste un record con i valori di username e password. Se esiste, restituisce un oggetto di tipo Titolare contenente il campo username del record, in caso contrario solleva un'eccezione: `LoginIncorrectException`

Esempio 2 – ProvaWS_2

Funzionamento (il proxy)

Per poter trasformare il metodo del controllore in un servizio è stato necessario introdurre una classe che faccia da proxy tra il controllore e il SOAP engine, Axis. Tale classe rappresenta lo strato View.

Al suo interno vi è un metodo:

```
public TitolareBean login(String username, String password);
```

che sarà il punto di accesso al servizio; difatti quando Axis riceverà una richiesta per il servizio invocherà proprio questo metodo.

Una volta invocato questo metodo non fa altro che istanziare il controllore e chiamarne il metodo ‘login’.

Esempio 2 – ProvaWS_2

Deploy di un servizio sotto Axis

Spostare tutti i file compilati (.class) dell'applicativo sotto la cartella 'classes' di Axis, mantenendo ovviamente la struttura dei package (albero di cartelle). In questo modo Axis avrà libero accesso all'interfaccia della classe proxy.

NB: Si ricorda che tale operazione viene effettuata in automatico da Eclipse utilizzando in Ant il file build.xml presente nel progetto di esempio ProvaWS_1.

A questo punto, Axis non è ancora "cosciente" del fatto che deve impiegare tale classe per l'esposizione di un servizio. Perciò è necessario effettuare il deployment del servizio. In questo modo si istruisce l'engine di Axis affinché in corrispondenza di una determinata richiesta (sottoforma di messaggio SOAP), istanzi la classe e ne richiami il relativo metodo.

Esempio 2 – ProvaWS_2

Deploy di un servizio sotto Axis

In generale “deployare” un servizio richiede la stesura di un file xml di deployment che nel nostro caso è ProvaWS_1.wsdd e contiene :

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

<service name="urn:ProvaWS_2" provider="java:RPC" >
 <parameter name="allowedMethods" value="login"/>
 <beanMapping qname="myNS:LoginExc" xmlns:myNS="urn:ProvaWS_2"
  languageSpecificType= "java:isp_0708.ProvaWS_2.
  myExceptions.LoginIncorrectException"/>
 <beanMapping qname= "myNS:Titolare" xmlns:myNS="urn:ProvaWS_2"
  languageSpecificType="java:isp_0708.ProvaWS_2.entity.Titolare"/>
 <parameter name="scope" value="Request"/>
 <parameter name="className" value="isp_0708.ProvaWS_2.proxy.ProvaWS2"/>
</service>
</deployment>
```

Esempio 2 – ProvaWS_2

Deploy di un servizio sotto Axis

L'attributo `name` rappresenta il nome che noi vogliamo assegnare al servizio e che servirà ad identificarlo univocamente.
L'attributo `provider` è stato valorizzato con "java:RPC" che indica ad Axis di pubblicare il servizio secondo un meccanismo RPC.

Di seguito troviamo una sezione composta da tre tag <parameter>.

- Il primo indica il nome (comprensivo di package) della classe Proxy.
- Il secondo descrive i metodi della suddetta classe che si vuole esporre (nel nostro caso l'unico metodo implementato 'saluto').
- Il terzo definisce il ciclo di vita della classe che in questo caso viene istanziata, utilizzata e distrutta ad ogni richiesta, 'Request' (altri valori possibili sono Application e Session).

Esempio 2 – ProvaWS_2

Deploy di un servizio sotto Axis

I tag <beanMapping> servono ad Axis per gestire il tipo di ritorno ‘Titolare’ e l’eccezione ‘LoginIncorrectException’. Difatti non essendo tipi primitivi è necessario segnalare ad Axis di trattarli come Bean di java e di serializzarli di conseguenza.

Per Lanciare il deployment occorre inserire nel classpath le seguenti librerie:
axis.jar ; jaxrpc.jar ; saaj.jar ; commons-logging.jar ; commons-discovery.jar ;log4j-1.2.8.jar
Infine eseguire il comando:

```
java org.apache.axis.client.AdminClient ProvaWS_1.wsdd
```

Esempio 2 – ProvaWS_2client

Funzionamento

Per testare il buon funzionamento del servizio creato è necessario sviluppare un'applicazione client Ad hoc quale 'ProvaWS_2client'.

ProvaWS_2client è una mini Web application composta solo da due pagine JSP, una per l'inserimento dei dati e l'altra per la visualizzazione dei risultati, da una classe Java Bean che appoggiandosi alle librerie di Axis effettua una RPC al servizio e dalle classi Titolare e LoginIncorrectException.

Il lato interessante di questa mini applicazione è incentrata appunto su come si effettua una RPC, su come si trattano le eccezioni remote e su come si serializzano/deserializzano oggetti tramite Axis.

Esempio 2 – ProvaWS_2client

Funzionamento

Per effettuare una RPC si deve istanziare un oggetto di tipo “Call” passandogli la url del Web service;

```
cll = new Call(new URL("http://localhost:8080/axis/services/"));
[…]
```

Si deve inserire il nome del servizio e quello del metodo da chiamare:

```
this.cll.setOperationName(new QName("urn:ProvaWS_2", "login"));
```

Esempio 2 – ProvaWS_2client

Funzionamento

Bisogna poi invocare il metodo invoke passandogli un'array di Object: questo array rappresenta i paramatri del metodo chiamato. Una volta eseguito il servizio il metodo invoke restituirà il valore/oggetto a sua volta restituito dal metodo remoto chiamato. Nel caso in cui il metodo remoto lanci un'eccezione, 'invoke' lancerà la stessa eccezione a sua volta ottenendo così la massima trasparenza.

```
Titolare result = (Titolare)cll.invoke(new
Object[]{this.username,this.password});
```

Esempio 2 – ProvaWS_2client

Funzionamento

Per predisporre il client del web service a ricevere o inviare oggetti che non sono tipi primitivi è necessario definire un modo per serializzarli e deserializzarli.

Questa operazione porta a correlare quattro fattori:

1. la classe dell'oggetto presente sull'applicazione client (`LoginIncorrectException.class`),
2. la classe dell'oggetto presente sul web service (`QName("urn:ProvaWS_2", "LoginExc")`),
3. →

Esempio 2 – ProvaWS_2client

Funzionamento

3. una classe che permette di serializzare
   `BeanSerializerFactory.class`
4. Una classe ulteriore che permette di deserializzare l'oggetto
   `BeanDeserializerFactory.class`

```
qnameProd = new QName("urn:ProvaWS_2", "LoginExc");
classe = LoginIncorrectException.class;
cll.registerTypeMapping(classe, qnameProd,
BeanSerializerFactory.class,BeanDeserializerFactory.class);
```