

iBatis

Gian Lorenzo Meocci

2 gennaio 2006

Sommario

Questo documento vuole essere una semplice introduzione al framework iBatis per ogni ulteriore chiarimento rimandiamo al sito ufficiale (<http://ibatis.apache.org>)

1 Introduzione

Partiamo da un esempio pratico. Stiamo sviluppando un' applicazione Java e vogliamo salvare lo stato degli oggetti in modo persistente, per poter successivamente compiere operazioni con essi. La soluzione più efficiente consiste nell'utilizzare un **RDBMS** per contenere i dati e un'interfaccia Java per gestirli.

Il problema maggiore consiste nello scrivere una serie di istruzioni SQL per interagire con la base dati per recuperare o scrivere le informazioni richieste. Quello che fa iBatis è rendere trasparente tutto ciò allo sviluppatore creando un mapping 1 a 1 tra gli oggetti Java e le relazioni nella base dati.

Questo permette di scrivere codice molto pulito eliminando quella parte del codice che si interfaccia con il database.

Il codice SQL viene inserito all'interno di file XML gestiti da iBatis, come anche la configurazione per l'accesso alla base dati. Il compito dello sviluppatore è quello di scrivere i file XML (con le relative query SQL) e il codice SQL per creare il DB, le tabelle etc., che ospiteranno gli oggetti.

Avremo quindi la possibilità di scrivere esattamente il codice SQL che desideriamo sul database da noi prescelto (iBatis gestisce il DB con JDBC), spostando ogni riferimento a tale codice al di fuori dell' applicazione reale. Successivamente potremo ragionare attraverso i concetti dell' OOP senza dover implementare nessuna classe apposita.

2 Un esempio

2.1 L'implementazione degli oggetti in Java

Per capire meglio cos'è e a cosa serve immaginiamoci questo semplice esempio. Dobbiamo scrivere un' applicazione in grado di gestire gli studenti e gli esami di una particolare facoltà, utilizzando come linguaggio di sviluppo Java.

Abbiamo dunque a disposizione un linguaggio OOP puro, e vogliamo sfruttare tutte le sue potenzialità. Iniziamo dunque a scrivere il codice per gestire i due oggetti chiave: **Studente** e **Esame**.

Implementiamo adesso Studente come semplice *JavaBean*.

```
1 package personal;
2
3 public class Studente{
4     private int Id=0;
5     private String Nome;
6     private String Cognome;
7     private String Matricola;
8
9     public void setId(int Id){this.Id=Id;}
10    public void setName(String Nome){
11        this.Nome=Nome;
12    }
13    public void setCognome(String Cognome){
14        this.Cognome=Cognome;
15    }
16    public void setMatricola(String Matricola){
17        this.Matricola=Matricola;
18    }
19    public int getId(){return Id;}
20    public String getNome(){return Nome;}
21    public String getCognome(){
22        return Cognome;
23    }
24    public String getMatricola(){
25        return Matricola;
26    }
27 }
```

In questa implementazione si notano i metodi **setXXX()** e **getXXX()**, utilizzati da iBatis per scrivere e leggere i valori dell' oggetto.

Alla stessa maniera si implementa l'oggetto **Esame**.

```
1 package personal;
2
3 public class Esame {
4     private int CodEsame;
5     private int CodStudente;
6     private String NomeEsame;
7     private int Voto;
8
9     public int getCodEsame(){
10         return CodEsame;
11     }
12     public int getCodStudente(){
13         return CodStudente;
14     }
15     public String getNomeEsame(){
16         return NomeEsame;
17     }
18     public int getVoto(){
19         return Voto;
20     }
21
22     public void setCodEsame(int CodEsame){
23         this.CodEsame =CodEsame;
24     }
25     public void setCodStudente(int CodStudente){
26         this.CodStudente =CodStudente;
27     }
28     public void setEsame(String Esame){
29         this.NomeEsame =Esame;
30     }
31     public void setVoto(int Voto){
32         this.Voto =Voto;
33     }
34 }
```

Adesso non resta che creare il database che ospiterà tali dati. Essendo questo un semplice esempio non importa che la struttura del DB sia

normalizzata ed efficiente ma dovrà rispecchiare la struttura logica che è stata seguita per implementare le classi Java (Studente ed Esame).

2.2 L'implementazione del DB

Come DataBase utilizziamo PostgreSQL, sia per la sua completa aderenza agli standard SQL99, sia per la robustezza che offre.

Iniziamo a creare il DB di appoggio:

```
CREATE DATABASE prova;
```

ed in successione tutte le tabelle necessarie.

```
1 create table Studente (  
2 id serial primary key,  
3 nome varchar(25),  
4 cognome varchar(25),  
5 matricola varchar(25)  
6 );  
7  
8 create table Esame (  
9 codesame serial primary key,  
10 cod_studente int references Studente(id),  
11 nome_esame varchar(25),  
12 voto int check(voto >= 18)  
13 );
```

2.3 Configurazione di iBatis

Come precedentemente detto, iBatis poggia la propria struttura su file XML. Sia il file di configurazione che i file per descrivere le query sono implementati attraverso file XML.

Iniziamo a vedere com'è strutturato il primo di questi file.

```
<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE sqlMapConfig PUBLIC  
" -//iBatis.com//DTD_SQL_Map_Config_2.0//EN"  
" http://www.ibatis.com/dtd/sql-map-2.dtd">  
  
<sqlMapConfig>
```

```

<transactionManager type="JDBC">
  <dataSource type="SIMPLE">
    <property name="JDBC.Driver" value="org.postgresql.Driver" />
    <property name="JDBC.ConnectionURL"
      value="jdbc:postgresql://localhost/prova" />
    <property name="JDBC.Username" value="postgres" />
    <property name="JDBC.Password" value="password" />
  </dataSource>
</transactionManager>
<sqlMap resource="insert.xml" />
<sqlMap resource="select.xml" />
</sqlMapConfig>

```

Innanzitutto dobbiamo definire quale sarà il driver che verrà utilizzato da iBatis per dialogare con il DB. Nel nostro caso utilizzeremo il **JDBC** in modalità **SIMPLE** (cioè senza utilizzare architetture proprietarie).

Successivamente si passa ad impostare il percorso del driver (necessario al compilatore Java, e dunque dobbiamo ricordarci di includere il driver JDBC di PostgreSQL nel CLASSPATH) e i vari parametri necessari alla comunicazione col DB.

Un' importante parametro è rappresentato dal tag **sqlMap** che specifica in quali file sono descritte le query SQL. Questi file devono essere depositati all'interno della cartella del progetto altrimenti iBatis non li troverà. Cosa molto importante è la definizione del DOCTYPE sia per il file di configurazione che per quello delle risorse.

Non ci resta che scrivere il codice che si occuperà delle query. Iniziamo a presentare un prima bozza alla quale andremo ad aggiungere le funzionalità via via richieste.

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMap PUBLIC "-//iBatis.com//DTD_SQL_Map_2.0//EN"
"http://www.ibatis.com/dtd/sql-map-2.dtd">
<sqlMap namespace="Studente">
  <insert id="insertStud" parameterClass="personal.Studente">
    INSERT INTO studente(nome,cognome,matricola)
    VALUES (#nome#,#cognome#,#matricola#)
  </insert>
  <selectKey resultClass="int" keyProperty="id">
    SELECT MAX(id) FROM studente
  </selectKey>

```

```

</insert>

<insert id="insertEsame" parameterClass="personal.Esame">
INSERT INTO esame(cod_studente , nome_esame , voto)
VALUES (#cod_studente#,#nome_esame#,#voto#)
</insert>
</sqlMap>

```

Con questo codice siamo già pronti per sperimentare alcune delle funzionalità messe a disposizione di iBatis.

2.4 L'Applicazione

2.4.1 Inserimento Dati

Iniziamo ad aprire un nuovo progetto Java con un qualsiasi ambiente di sviluppo. Ricordiamoci di includere nel CLASSPATH i percorsi per le librerie di iBatis e per il driver JDBC di PostgreSQL.

```

1 import personal.*;
2 import com.ibatis.common.resources.Resources;
3 import com.ibatis.sqlmap.client.*;
4 import java.util.*;
5 import java.io.*;
6
7 public class parse {
8     public static void main(String ... args){
9         try{
10            SqlMapClient map=null;
11            Reader r=Resources.getResourceAsReader("sqlmap.xml");
12            map=SqlMapClientBuilder.buildSqlMapClient(r);
13
14            Studente stud=new Studente();
15            stud.setnome("Gian_Lorenzo");
16            stud.setcognome("Meocci");
17            stud.setmatricola("780200094");
18            map.insert("insertStud",stud);
19
20            Esame esame=new Esame();
21            esame.setcod_studente(stud.getid());

```

```

22         esame.setnome_esame("Complementi di Analisi");
23         esame.setvoto(27);
24         map.insert("insertEsame",esame);
25
26         System.out.println("ok");
27     }catch(Exception ex){ex.printStackTrace();}
28     }
29 }

```

Il codice è molto semplice, si inizia ad includere le librerie necessarie, e ad instanziare l'oggetto principale: **SqlMapClient**.

Tale oggetto è il ponte tra il mondo ad oggetti di Java e il mondo di relazioni SQL. Dopo averlo dichiarato lo si inizializza utilizzando il file *sqlmap.xml* sopra presentato.

Da questo punto in poi possiamo tranquillamente dimenticare la struttura SQL sottostante. Infatti, nell'esempio precedente, non facciamo altro che instanziare l'oggetto *stud* di tipo **Studente** chiamando i relativi metodo per impostare le varie proprietà dell'oggetto.

Una volta che l'oggetto è pronto per essere inserito nella base dati basta chiamare la giusta funzione dell'oggetto *SqlMapClient* precedentemente creato.

Nel nostro caso stiamo facendo una **INSERT** di uno **Studente** e dunque utilizziamo `map.insert("insertStud", stud)` per assolvere a tale compito. Il medesimo discorso vale per l'inserimento dell'esame. In questo caso sfruttiamo utilizziamo la funzione *getid()* per soddisfare l'integrità referenziale tra la tabella **Studente** ed **Esame** sull'attributo **cod_studente**.

2.4.2 Recupero Dati

In questo breve paragrafo vedremo come è estremamente semplice recuperare l'informazione dalla nostra base dati.

Iniziamo a mostrare la struttura del file *select.xml*

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMap PUBLIC "-//iBATIS.com//DTD_SQL_Map_2.0//EN"
"http://www.ibatis.com/dtd/sql-map-2.dtd">
<sqlMap namespace="Studente">
<select id="selectStud" resultClass="personal.Studente">
SELECT * FROM studente
</select>

```

```
<select id="EsamiStud" resultClass="personal.EsamiStud">
SELECT studente.nome AS nome_studente ,
studente.cognome as cognome_studente ,
esame.nome_esame , voto
FROM studente ,esame
WHERE esame.cod_studente=studente.id
</select>
</sqlMap>
```

Attraverso questo file si descrivono le query di tipo **SELECT** richieste dal progetto. Soffermiamoci sulla query "EsamiStud". Questa query ci consente di recuperare tutti gli esami fatti da un determinato studente, riportando anche il voto preso.

Come si osserva dal codice XML la query restituisce i risultati all'interno di un oggetto Java chiamato **EsamiStud** che riportiamo di seguito.

```
1 package personal;
2
3 public class EsamiStud {
4     private String NomeStudente;
5     private String CognomeStudente;
6     private String NomeEsame;
7     private int voto;
8
9     public void setnome_studente(String ns){
10         NomeStudente=ns;
11     }
12     public void setcognome_studente(String cs){
13         CognomeStudente=cs;
14     }
15     public void setnome_esame(String ne){
16         NomeEsame=ne;
17     }
18     public void setvoto(int voto){
19         this.voto=voto;
20     }
21     public String getnome_studente(){
22         return NomeStudente;
23     }
```

```

24     public String getcognome_studente(){
25         return CognomeStudente;
26     }
27     public String getnome_esmae(){
28         return NomeEsame;
29     }
30     public int getvoto(){
31         return voto;
32     }
33 }

```

Dall' interno della nostra applicazione sarà sufficiente scrivere il seguente codice:

```

List<EsamiStud> k=map.queryForList("EsamiStud", null);
ArrayList<EsamiStud> mystud=new ArrayList<EsamiStud>(k);
for(EsamiStud obj:mystud){
    System.out.print(obj.getnome_studente()+ " "
    + obj.getcognome_studente()+"_ha_sostenuto_l'esame_di_");
    System.out.print(obj.getnome_esmae());
    System.out.println("_prendendo_"+obj.getvoto());
}

```

per scorrere tutti i risultati restituiti dal DB.

Per interrogare il DB esistono varie metodi messi a disposizione da SqlMapClient. Principalmente vengono utilizzate due funzioni: *queryForObject()* e *queryForList()*.