iBATIS SQL Maps

Tutorial

Per SQL Maps Versione 2.0

1 Maggio, 2005

Traduzione italiana a cura di Fabrizio Gianneschi (fabrizio@gianneschi.it)

Introduzione

Questo breve tutorial illustra un utilizzo tipico di SQL Maps. I dettagli di ciascuno degli argomenti che seguiranno possono essere trovati sulla Guida per lo sviluppatore di SQL Maps (*SQL Maps Developer Guide*), disponibile su http://www.ibatis.com

---

Prepararsi all'uso di SQL Maps

Il framework SQL Maps è molto tollerante rispetto a cattivi modelli di database e addirittura a cattivi modelli ad oggetti. Ciò nonostante, si raccomanda di progettare il proprio database (attraverso una corretta normalizzazione) ed il proprio modello secondo lo stato dell'arte. Facendo ciò, si avranno buone performance ed un progetto migliore.

Il modo più semplice per iniziare è analizzare su cosa si sta lavorando. Quali sono gli oggetti di business? Quali sono le tabelle del database? Come si relazionano tra loro? Come primo esempio, si consideri la seguente semplice classe Person, conforme alla tipica convenzione sui JavaBean.

Person.java

```
package examples.domain;
//import omessi….

public class Person {
    private int id;
    private String firstName;
    private String lastName;
    private Date birthDate;
    private double weightInKilograms;
    private double heightInMeters;

    public int getId () {
        return id;
    }
    public void setId (int id) {
        this.id = id;
    }

    //…assumiamo che ci siano gli altri get e set…
}
```

In che modo questa classe Person si mappa sul nostro database? SQL Maps non vi costringe ad avere relazioni del tipo *una-tabella-per-classe* oppure *più-tabelle-per-classe* oppure *più-classi-per-tabella*. Dato che avete disponibile tutta la potenza dell'SQL, esistono ben poche restrizioni. Nel nostro esempio si consideri la seguente semplice tabella, che potrebbe essere adatta in una situazione del tipo *una-tabella-per-classe*.

Person.sql

```
CREATE TABLE PERSON(
        PER_ID                  NUMBER          (5, 0)    NOT NULL,
        PER_FIRST_NAME          VARCHAR         (40)      NOT NULL,
        PER_LAST_NAME           VARCHAR         (40)      NOT NULL,
        PER_BIRTH_DATE          DATETIME ,
        PER_WEIGHT_KG           NUMBER          (4, 2)    NOT NULL,
        PER_HEIGHT_M            NUMBER          (4, 2)    NOT NULL,
        PRIMARY KEY (PER_ID)
```

)

Il file di configurazione di SQL Maps

Dopo aver preso confidenza con le classi e le tabelle da utilizzare, il luogo migliore per iniziare è il file di configurazione di SQL Maps. Tale file agisce da *radice* per quanto riguarda la configurazione della nostra particolare implementazione di SQL Maps.

Il file di configurazione è un file XML. Al suo interno configureremo proprietà, fonti dati JDBC e *mappature* SQL. Il file rappresenta un punto unico di configurazione per i vostri DataSource, che possono essere diversi e differenti. Il framework può gestire varie implementazioni dell'interfaccia DataSource, tra cui l'iBATIS SimpleDataSource, Jakarta DBCP (Commons) e qualunque altro DataSource che possa essere reperito tramite un contesto JNDI (ad esempio dall'interno di un application server). Tutto ciò è descritto con maggiore dettaglio nella Developer's Guide.

La struttura è molto semplice e, per l'esempio di cui sopra, può essere la seguente:

L'esempio continua nella pagina seguente...

SqlMapConfigExample.xml

```
<?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-config-2.dtd">

  <!-- Assicuratevi sempre di utilizzare una corretta intestazione XML come sopra! -->

  <sqlMapConfig>

    <!-- Le proprietà (nome=valore) nel file di seguito specificato possono essere usate come segnaposto
    in questo file (es. "${driver}". Il file è di solito relativo al classpath ed è opzionale. -->

    <properties resource="examples/sqlmap/maps/SqlMapConfigExample.properties" />

    <!-- Questi settaggi controllano i dettagli di configurazione di SQLMaps, principalmente per quanto riguarda
    la gestione delle transazioni. Sono tutti opzionali (consultare la Developer Guide per dettagli) -->

    <settings
      cacheModelsEnabled="true"
      enhancementEnabled="true"
      lazyLoadingEnabled="true"
      maxRequests="32"
      maxSessions="10"
      maxTransactions="5"
      useStatementNamespaces="false"
    />

    <!-- I Type alias vi consentono di utilizzare nomi più corti al posto dei nomi completi delle classi -->

    <typeAlias alias="order" type="testdomain.Order"/>

    <!-- Configura un DataSource da utilizzare in questa SQL Map, utilizzando un SimpleDataSource.
    Notare l'uso di proprietà caricate dal file di properties definito in precedenza -->

    <transactionManager type="JDBC" >
      <dataSource type="SIMPLE">
            <property name="JDBC.Driver" value="${driver}"/>
            <property name="JDBC.ConnectionURL" value="${url}"/>
            <property name="JDBC.Username" value="${username}"/>
            <property name="JDBC.Password" value="${password}"/>
      </dataSource>
    </transactionManager>

    <!-- Identifica tutti i file XML di SQL Map da caricare per la mappatura corrente. Notare che i percorsi sono
    relativi al classpath. Per ora, ne abbiamo solo uno … -->

    <sqlMap resource="examples/sqlmap/maps/Person.xml" />
  </sqlMapConfig>
```

SqlMapConfigExample.properties

```
# Questa è un semplice file di proprietà che semplifica la configurazione automatica
# del file di configurazione di SQL Maps (es. da parte di Ant oppure di strumenti di
# integrazione continua su ambienti differenti… etc.)
# I valori seguenti possono essere utilizzati in ogni valore di proprietà del file precedente (es. "${driver}")
# Utilizzare un file di proprietà come questo, comunque, è assolutamente opzionale.

driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:oracle1
username=jsmith
password=test
```

I file SQL Map (mappature SQL)

Ora che abbiamo un DataSource configurato ed il nostro file di configurazione è pronto, dobbiamo produrre un file SQL Map che contenga il nostro codice SQL e le mappature per gli oggetti che faranno da parametri di input e di output.

Proseguendo con l'esempio di cui sopra, costruiamo un SQL Map file per la classe Person e per la tabella PERSON. Inizieremo con la struttura generale di un documento SQL ed una semplice query:

Person.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="Person">

    <select id="getPerson" resultClass="examples.domain.Person">
        SELECT PER_ID as id,
        PER_FIRST_NAME as firstName,
        PER_LAST_NAME as lastName,
        PER_BIRTH_DATE as birthDate,
        PER_WEIGHT_KG as weightInKilograms,
        PER_HEIGHT_M as heightInMeters
        FROM PERSON
        WHERE PER_ID = #value#
    </select>

</sqlMap>
```

L'esempio sopra citato mostra la forma più semplice di mappatura SQL. Utilizza una funzionalità del framework SQL Maps che mappa automaticamente le colonne di un ResultSet in proprietà JavaBean (o chiavi di una java.util.Map), basandosi sull'uguaglianza del nome. Il token *#value#* è un parametro di input. Più nel dettaglio, l'uso del "value" implica che si sta usando un wrapper di tipi primitivi (es. Integer; ma non è l'unico possibile).

Sebbene molto semplice, ci sono alcune limitazioni nell'utilizzare l'approccio a mappature automatiche. Non c'è modo di specificare il tipo delle colonne di output (se necessario) o caricare automaticamente dei dati correlati (proprietà complesse); c'è inoltre un lieve decadimento di prestazioni, in quanto quest'approccio richiede l'accesso all'interfaccia ResultSetMetaData. Utilizzando un **resultMap**, possiamo oltrepassare queste limitazioni. Tuttavia, in questa sede, il nostro scopo è la semplicità, e potremo sempre cambiare in seguito (senza essere costretti a

modificare il codice Java).

La maggior parte delle applicazioni agenti sui database non devono semplicemente accedere in lettura, ma anche in scrittura. Abbiamo già visto un esempio di come viene mappata una semplice SELECT, ma che dire a proposito delle INSERT, UPDATE e DELETE? La buona notizia è che non c'è alcuna differenza. Di seguito completeremo la nostra SQL Map relativa alla classe Person per fornire un set completo di statement per leggere e modificare i dati.

Person.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="Person">

    <!-- Utilizza una classe Wrapper (es. Integer) come parametro di input e mappa
    automaticamente i risultati sulle proprietà (JavaBean) di un oggetto Person  -->

    <select id="getPerson" parameterClass=”int” resultClass="examples.domain.Person">
        SELECT PER_ID as id,
        PER_FIRST_NAME as firstName,
        PER_LAST_NAME as lastName,
        PER_BIRTH_DATE as birthDate,
        PER_WEIGHT_KG as weightInKilograms,
        PER_HEIGHT_M as heightInMeters
        FROM PERSON
        WHERE PER_ID = #value#
    </select>

    <!-- Utilizza le proprietà (JavaBean) di un oggetto Person come parameteri per l'inserimento. Ciascuno dei
    parametri mappati tra i simboli #...# è una proprietà JavaBean. -->

    <insert id="insertPerson" parameterClass="examples.domain.Person">
        INSERT INTO
        PERSON (PER_ID, PER_FIRST_NAME, PER_LAST_NAME,
        PER_BIRTH_DATE, PER_WEIGHT_KG, PER_HEIGHT_M)
        VALUES (#id#, #firstName#, #lastName#,
        #birthDate#, #weightInKilograms#, #heightInMeters#)
    </insert>

    <!-- Utilizza le proprietà (JavaBean) di un oggetto Person come parameteri per la modifica. Ciascuno dei
    parametri mappati tra i simboli #...# è una proprietà JavaBean. -->

    <update id="updatePerson" parameterClass="examples.domain.Person">
        UPDATE PERSON
        SET PER_FIRST_NAME = #firstName#,
        PER_LAST_NAME = #lastName#, PER_BIRTH_DATE = #birthDate#,
        PER_WEIGHT_KG = #weightInKilograms#,
        PER_HEIGHT_M = #heightInMeters#
        WHERE PER_ID = #id#
    </update>

    <!-- Utilizza le proprietà (JavaBean) di un oggetto Person come parameteri per la cancellazione. Ciascuno dei
    parametri mappati tra i simboli #...# è una proprietà JavaBean. -->

    <delete id="deletePerson" parameterClass="examples.domain.Person">
        DELETE PERSON
        WHERE PER_ID = #id#
    </delete>
</sqlMap>
```

Programmazione con il framework SQL Maps

Una volta configurato e mappato tutto, ciò che dobbiamo fare è scrivere un po' di codice Java nella nostra applicazione. Il primo passo è configurare la SQL Map. Si tratta solamente di caricare il file XML di configurazione della SQL Map che abbiamo creato in precedenza. Per caricarlo, possiamo utilizzare la classe Resources inclusa nel framework.

```
String resource = “com/ibatis/example/sqlMap-config.xml”;
Reader reader = Resources.getResourceAsReader (resource);
SqlMapClient sqlMap = SqlMapClientBuilder.buildSqlMapClient(reader);
```

L'oggetto di classe SQLMapClient è un oggetto di servizio, *thread safe* e dalla lunga vita. In genere avrete bisogno di istanziarlo e configurarlo una sola volta per ciclo di vita dell'applicazione. Ciò lo rende un ottimo candidato per essere un membro statico di una classe di base (es. un DAO) oppure, se preferite mantenerlo in un punto più centrale e disponibile, potete wrapparlo in una vostra classe creata per l'occasione. Di seguito viene mostrato un esempio di una classe che potrebbe servire a tale scopo:

```
public MyAppSqlConfig {
    private static final SqlMapClient sqlMap;
    static {
        try {
          String resource = “com/ibatis/example/sqlMap-config.xml”;
          Reader reader = Resources.getResourceAsReader (resource);
          sqlMap = SqlMapClientBuilder.buildSqlMapClient(reader);
        } catch (Exception e) {
          // Se si verifica un 'eccezione in questo punto, non preoccupatevi di cos'è successo. Sarà senz'altro
          // un errore non recuperabile del quale vorremo solamente essere informati.
          // Dovreste sempre loggare questi tipi di errori e rilanciarli in una maniera che possano essere
          // immediatamente riconosciuti.
          e.printStackTrace();
          throw new RuntimeException (“Error initializing MyAppSqlConfig class. Cause: ” + e);
        }
    }
    public static getSqlMapInstance () {
        return sqlMap;
    }
}
```

Leggere oggetti dal Database

Ora che l'istanza di SqlMapClient (sqlMap) è inizializzata e facilmente accessibile, possiamo utilizzarla. Iniziamo col leggere un oggetto Person dal database. (In questo esempio, assumiamo che ci siano 10 record nella tabella PERSON, aventi PER_ID da 1 a 10).

Per ottenere un oggetto Person dal database, abbiamo semplicemente bisogno dell'istanza di SqlMapClient, del nome dello statement mappato da invocare e dell'ID del record da caricare. Proviamo a caricare l'oggetto Person #5.

```
SqlMapClient sqlMap = MyAppSqlMapConfig.getSqlMapInstance(); // come definito sopra
…
Integer personPk = new Integer(5);
Person person = (Person) sqlMap.queryForObject (“getPerson”, personPk);
…
```

Salvare oggetti sul Database

Abbiamo appena letto un oggetto Person dal database. Proviamo a modificare alcuni dati. Cambieremo l'altezza ed il peso della persona che l'oggetto rappresenta.

```
...
person.setHeightInMeters(1.83); // person come dall'esempio precedente
person.setWeightInKilograms(86.36);
...
sqlMap.update("updatePerson", person);
...
```

Se vogliamo cancellare l'oggetto Person, è tutto molto semplice.

```
...
sqlMap.delete ("deletePerson", person);
...
```

Creare un nuovo oggetto Person è analogo:

```
Person newPerson = new Person();
newPerson.setId(11); // In genere si ottiene l'ID da una sequence, oppure da un'apposita tabella
newPerson.setFirstName("Clinton");
newPerson.setLastName("Begin");
newPerson.setBirthDate (null);
newPerson.setHeightInMeters(1.83);
newPerson.setWeightInKilograms(86.36);
...
sqlMap.insert ("insertPerson", newPerson);
...
```

Tutto qui. E' tutto ciò che c'è da fare!

Prossimi passi...

Siamo giunti alla fine di questo breve tutorial. Per la Guida Completa a SQL Maps 2.0, così come di *JPetstore 4*, una completa e funzionante applicazione web dimostrativa basata su Jakarta Struts, iBATIS DAO 2.0 e SQL Maps 2.0, visitate http://www.ibatis.com

CLINTON BEGIN NON FORNISCE ALCUNA GARANZIA, ESPLICITA O IMPLICITA, PER QUANTO RIGUARDA LE INFORMAZIONI CONTENUTE NEL PRESENTE DOCUMENTO.