iBATIS SQL Maps

Einführung

für SQL Maps Version 2.0

Oktober 27, 2007

Übersetzung Axel Leucht (Axel.Leucht@gmx.net)

Einführung

Dieses Handbuch gibt ihnen eine Einführung des typischen Einsatzes von SQL Maps. Details von SQL Maps können sie im Handbuch lesen und hier beziehen: http://ibatis.ibatis.apache.org.org

---

Vorbereitung zur Nutzung von SQL Maps

Das SQL Maps Framework ist sehr tolerant gegenüber schlechter Datenbankmodellierung und/oder einem schlechten Objektmodell. Trotzdem empfehlen wir ihnen Standardpraktiken beim Entwurf der Datenbank (Normalisierung) oder des Objektmodells anzuwenden. Dadurch erhalten sie eine gute Performance und ein makelloses Design.

Am einfachsten starten sie, indem sie sich fragen womit sie arbeiten. Was sind ihre Business Objekte? Welche Datenbanktabellen benötigen sie? Welche Beziehungen existieren zwischen ihnen? Im ersten Beispiel nehmen wir folgende einfache Person-Klasse an, die dem typischen JavaBean-Muster genügt.

Person.java

```
package examples.domain;

//korrekte imports angenommen….

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;
   }

   //…hier die üblichen getters und setters um Platz zu sparen…
}
```

Wie wird diese Person-Klasse auf ihre Datenbank abgebildet? SQL Maps schränkt sie nicht bei Beziehungen von Objekten zur Datenbank ein, sondern sie können Beziehungen verwenden um zum Beispiel eine Tabelle pro Klasse, mehrfache Tabellen pro Klasse oder mehrfache Klassen pro Tabelle. Da sie die volle Kontrolle über das verwendete SQL haben, gibt es praktisch keine Einschränkungen. Nehmen wir für dieses Beispiel eine einfache Tabelle-pro-Klasse Beziehung an:

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)
)
```

Die SQL Map Konfigurationsdatei

Nachdem wir wissen mit welchen Klassen und Tabellen wir arbeiten ist der nächste Schritt die SQL Map Konfigurationsdatei. Diese Datei beschreibt die zentrale Konfiguration der SQL Map Implementierung.

Bei der Konfigurationsdatei handelt es sich um eine XML Datei. In ihr werden Properties, JDBC DataSources und SQL Maps konfiguriert. Dieses gibt ihnen einen Ort um zentral ihre DataSource aus einer Reihe von möglichen Implementierungen zu definieren, unter anderem iBATIS SimpleDataSource, Jakarta DBCP (Commons), oder DataSource aus einem JNDI Kontext (z.B. aus einem Appserver). Diese Möglichkeiten sind detaillierter im Handbuch beschrieben. Die Struktur ist simpel und für obiges Beispiel könnte sie wie folgt aussehen:

Beispiel fortgesetzt auf nächster Seite…

SqlMapConfigExample.xml

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE sqlMapConfig
    PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

<!-- Nehmen sie IMMER den korrekten XML header wie oben angegeben! -->

<sqlMapConfig>

  <!-- Die properties-Datei stellt diverse Name/Wert-Paare (name=value) bereit, die in dieser
Konfigurationsdatei einfach mit Platzhaltern der Form "${driver}" referenziert werden können. Die
properties-Datei ist üblicherweise im Klassenpfad angegeben. Eine properties-Datei Angabe ist
optional.  -->

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

  <!-- Die folgende Einstellungen spezifizieren Details der SqlMap Konfiguration wie zum Beispiel
das Transaktion Management.  Sie sind alle optional (Details im Handbuch).  -->

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

  <!-- Type aliases erlauben ihnen, einen kurzen Namen statt eines langen voll qualifizierten
Klassennamen zu definieren. -->

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

  <!-- Spezifizieren sie eine DataSource für diese SQL Map. Hier nehmen wir eine einfache
SimpleDataSource.
        Beachten sie die Verwendung der properties aus oben angegebener properties-Datei. -->

  <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>

  <!-- Geben sie alle SQL Map XML Dateien an, die in dieser SQL Map geladen werden sollen.
Beachten sie das alle Dateien im Classpath liegen.  Zur Zeit haben wir nur eine Datei… -->

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

</sqlMapConfig>
```

SqlMapConfigExample.properties

```
# Dies ist eine einfache properties-Datei, um zum Beispiel umgebungsspezifische Werte in einer
externen Datei zu halten.
# Somit kann durch Tausch einer einzigen Datei zum Beispiel auf verschiedene Datenbanken
# (Test,Deployment,Produktion,...) zugegriffen werden
# Benutzung einer properties Datei ist komplett optional.

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

SQL Map Datei(en)

Nun haben wir eine DataSource eingerichtet. Wir benötigen zusätzlich eine SQL Map Datei, in der wir unsere SQL-Befehle, die Abbildung von Input-Parametern und Ausgabe-Parametern der SQL-Befehle vornehmen.

Fahren wir mit dem Beispiel fort und erstellen eine SQL Map Datei für die Person Klasse und der PERSON Tabelle. Wir starten mit dem generellen Aufbau der Datei und einem einfachen select Anweisung:

Person.xml

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE sqlMap
   PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
   "http://ibatis.apache.org/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>
```

Obiges Beispiel zeigt die einfachste Form einer SQL Map Datei. Es verwendet ein Feature des SQL Maps Frameworks das automatisch Spalten aus dem ResultSet auf JavaBean Properties (oder Map keys etc.) abbildet. Die #value# Angabe ist ein Input Parameter. Präzise geben wir an, das es sich bei „value“ um einen einfachen Wrapper für Java-Primitive (z.B. Integer) handelt.

Es gibt noch andere Einschränkungen für dieses automatische Result Mapping. Es gibt zum Beispiel keine Möglichkeit den Typen der Spalten anzugeben oder komplexe Objektgraphen zu laden. Außerdem bringt dieses Verfahren einen kleinen Performance-Nachteil beim Zugriff auf die ResultSetMetaData mit sich. Durch die Angabe einer Result Map können alle diese Einschränkungen umgangen werden.. Da aber Einfachheit unser Ziel ist, können wir dies zu einem späteren Zeitpunkt nachholen ohne eine Zeile Java Code zu ändern.

Die meisten Datenbank Applikationen lesen nicht nur einfach aus der Datenbank, sondern modifizieren auch die Daten. Wir haben ein Beispiel gesehen, wie ein einfaches SELECT gemapped wird, aber wie geschieht dies mit INSERT, UPDATE and DELETE? Gute Nachricht: Kein Unterschied! Unten geben wir ein vollständigeres Mapping mit mehr SQL Befehlen an, die neben Daten selektieren diese auch einfügen und ändern können.

Person.xml

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE sqlMap
   PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
   "http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="Person">

    <!-- Verwende einen primitiven Wrapper Type (z.B.. Integer) als Parameter und verwende auto-
mapping  vom Resultat-Spalten auf das Person Objekt (JavaBean) Properties -->
    <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>

    <!--Verwende Person Objekt (JavaBean) Properties als Parameter für dasSQL-Insert.  Jeder der
Parameters der Form #hash# Symbole ist eine JavaBean property der Person Klasse  -->
    <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>

    <!-- Verwende Person Objekt (JavaBean) Properties als Parameter für den SQL-update. Jeder der
Parameters der Form #hash# Symbole ist eine JavaBean property der Person Klasse.  -->
    <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>
Verwende Person Objekt (JavaBean) Properties als Parameter für den SQL-delete. Jeder der
Parameters der Form #hash# Symbole ist eine JavaBean property der Person Klasse.  -->
    <delete id="deletePerson" parameterClass="examples.domain.Person">
        DELETE PERSON
        WHERE PER_ID = #id#
   </delete>

</sqlMap>
```

Entwickeln mit dem SQL Map Framework

Wir haben nun alles konfiguriert und können uns der Java Applikation widmen. Zuallererst muss SQL Map initialisiert werden. Dieses besteht einfach darin, die SQL Map Konfigurationsdatei, die wir zuvor erstellt haben, zu laden. Dieses können wir einfach mit der Resources Klassen aus dem Framework bewerkstelligen.

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

Dieses SqlMapClient Objekt ist langlebig und thread safe. Für einen Applikationslauf muss dieses Objekt nur einmal instantiiert werden. Dieses kann geeignet in einem static Member einer Basisklasse (z.B. Basis-DAO Klasse) geschehen. Wenn sie einen anderen Weg bevorzugen kann es zum Beispiel auch in einer Helper-Klasse geschehen:

```
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) {
         // Wenn sie zu dieser Stellen kommen, ist ein Fehler aufgetreten und es eigentlich
         // nicht wichtig um welchen Fehler es sich handelte. Auf jeden Fall ist die Situation nicht
         // behebbar und wir sollten die Applikation „hochgehen" lassen, so das wir vom Problem
        // wissen. Sie sollten mindestens solche Fehler loggen und erneut die Exception werfen
         // damit sie Kenntnis über das Problem bekommen.
         e.printStackTrace();
         throw new RuntimeException ("Error initializing MyAppSqlConfig class.  Cause: " + e);
       }
   }

   public static SqlMapClient getSqlMapInstance () {
      return sqlMap;
   }

}
```

Lesen von Objekten aus der Datenbank

Nun,da die SqlMap Instanz initialisiert ist, können wir sie verwenden. Lesen wir als Erstes ein Person Objekt aus der Datenbank. (Nehmen wir an, das es bereits 10 PERSON Datensätze mit PER_ID von 1 bis 10 gibt).

Um ein Person Objekt aus der Datenbank zu lesen, benötigen wir nur die SqlMap Instanz, der Namen des gemappten Statements und eine Person ID. Laden wir Person #5.

```
…
SqlMapClient sqlMap = MyAppSqlMapConfig.getSqlMapInstance(); // siehe oben
…
Integer personPk = new Integer(5);
Person person = (Person) sqlMap.queryForObject ("getPerson", personPk);
…
```

Schreiben von Objekten in die Datenbank

Nun haben wir ein Person Objekt aus der Datenbank. Ändern wir einige Daten, zum Beispiel Größe und Gewicht der Person.

```
…
person.setHeightInMeters(1.83);          // person wie zuvor gelesen
person.setWeightInKilograms(86.36);
…
sqlMap.update("updatePerson", person);
…
```

Wenn sie die Person löschen möchten, ist dies genauso einfach.

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

Einfügen einer neues Person ist ähnlich.

```
Person newPerson = new Person();
newPerson.setId(11);        // normalerweise würden sie die ID aus einer Sequenz oder einen eigenen
Tabelle holen
newPerson.setFirstName("Clinton");
newPerson.setLastName("Begin");
newPerson.setBirthDate (null);
newPerson.setHeightInMeters(1.83);
newPerson.setWeightInKilograms(86.36);
…
sqlMap.insert ("insertPerson", newPerson);
…
```

Mehr gibt es nicht zu wissen!

Nächste Schritte…

Dies ist das Ende dieser kurzen Einführung Schauen Sie bitte bei http://ibatis.apache.org nach dem komplette SQL Maps 2.0 Handbuch, und dem JPetStore 4, einer kompletten auf Struts, iBATIS DAO 2.0 und SQL Maps 2.0 basierenden Webapplikation.
