iBATIS SQL Maps

チュートリアル

For SQL Maps Version 2.0

February 18, 2006

稲葉 信之(inaba@techmatrix.co.jp) 訳

イントロダクション

この簡潔なチュートリアルは、SQL Maps の標準的な使用方法のウォークスルーを通じて SQL Maps を案内します。トピックごとの詳細は、http://ibatis.apache.org から利用可能な SQL Maps 開発者ガイドで見つけることができます。

SQL Maps を使用する準備

SQL Maps フレームワークは、悪いデータベースモデル、そして悪いモデルにさえも受け入れることができます。しかしながら、データベース（適切な標準化）とオブジェクトモデルをデザインする時にはベストプラクティスを使用することを推奨します。そうすることにより良いパフォーマンスと良い設計を得られるでしょう。

分析を始めるにあたり、一番簡単なのは、あなた自身を分析することです。あなたのビジネスオブジェクトはなんですか？あなたのデータベーステーブルは何ですか？どのようにそれらが関連していますか？初めの実例のために、下記の標準的な JavaBeans パターンに従った Person クラスを考えてみてください。

Person.java

```
package examples.domain;

//imports implied….

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;
    }

    //…let’s assume we have the other getters and setters to save space…
}
```

どのように、この Person クラスをデータベースにマップしますか？SQL Maps は、リレーション（例えば、テーブル対 Person クラス、複数テーブル対 Person クラス、テーブル対複数クラス）を持つことを制限しません。これは、(ほんのわずかな制約がありますが)あなたが持っている SQL の全ての機能を利用するためです。この実例では、テーブルと Person クラスが 1 対 1 のリレーションとなる下記のシンプルなテーブルを使いましょう。

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_MNUMBER  (4, 2)  NOT NULL,
    PRIMARY KEY (PER_ID)
)
```

SQL Map 設定ファイル

一旦、クラスとテーブルが作成できたら SQL Map 設定ファイルを作成しましょう。このファイルは、SQL Map の大本となる設定です。設定ファイルは、XML ファイルです。設定ファイルの中では、JDBC DataSources と SQL Maps のプロパティ設定を行います。設定ファイルは、データソー

スの実装を設定するのに、とても便利です。フレームワークは、iBatis SimpleDataSource、jakarta DBCP (Commons)、JNDI コンテキスト経由で見つけることができる DataSource(例．アプリケーションサーバの中)からデータソースを扱うことができます。開発者ガイドに、より詳細な記述があります。設定ファイルの構造はシンプルで、実例の場合には，このようになります。

実例は、次のページに続く

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">

<!— Always ensure to use the correct XML header as above! —>

<sqlMapConfig>

  <!— The properties (name=value) in the file specified here can be used placeholders in this
config
        file (e.g. “${driver}” .  The file is usually relative to the classpath and is
optional.  —>

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

  <!— These settings control SqlMap configuration details, primarily to do with transaction
        management.  They are all optional (see the Developer Guide for more).  —>

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

  <!— Type aliases allow you to use a shorter name for long fully qualified class names. —>

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

  <!— Configure a datasource to use with this SQL Map using SimpleDataSource.
        Notice the use of the properties from the above resource —>

  <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>

  <!— Identify all SQL Map XML files to be loaded by this SQL map.  Notice the paths
        are relative to the classpath.  For now, we only have one… —>

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

</sqlMapConfig>
```

SqlMapConfigExample.properties

```
# This is just a simple properties file that simplifies automated configuration
# of the SQL Maps configuration file (e.g. by Ant builds or continuous
# integration tools for different environments… etc.)
# These values can be used in any property value in the file above (e.g. "${driver}" )
# Using a properties file such as this is completely optional.

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

SQL Map ファイル

私たちはデータソースを設定しました。そして、設定ファイルの準備ができました。次は、パラメータオブジェクトと(入力と出力ごとの)結果オブジェクトのためのマッピングと SQL コードを含む SQL Map ファイルを提供する必要があります。

実例のための Person クラスと Person テーブルのための SQL Map ファイルを作成しましょう。SQL ドキュメントの構造とシンプルな SQL から始めます。

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>
```

上記の実例は、SQL Map の最もシンプルな形を示しています。ResultSet のカラムと JavaBeans のプロパティ(もしくは、Map キーなど)で名称が一致しているものを自動的にマップする SQL Maps の機能を使っています。#value#トークンは、入力パラメータです。より正確に言うと value の使用は、シンプルプリミティブラッパー(例.Integer;しかし制限していません)を使用するということを意味します。

とてもシンプルにも関わらず自動結果マッピングアプローチの使用には、いくつかの制限があります。(必要であれば)出力カラムの型を指定する、または関連するデータ(複雑なプロパティ)を自動的にロードする方法がありません。そしてこのアプローチは、ResultMetaData へのアクセスを要求するので若干パフォーマンスに影響があります。
ResultMap を使用することによって、これらの制限を克服することができます。しかし、今はシンプルにすることがゴールなのでアプローチは変更しません。このアプローチは(Java ソースコードの変更無しで)いつでも別のアプローチに変更することができます。
ほとんどのデータベースアプリケーションは、データベースから単に読み込みするだけではなくデータベースのデータを変更しなければなりません。私たちは、シンプルな SELECT 文がどのようにマップされるのかを見ました。しかし、INSERT、UPDATE、DELETE についてはどうでしょうか？もちろん、それらも違いがありません。下記は、データアクセスとデータ変更のためのステートメントの完全なセットを追加して Person SQL マップを完全にしたものです。

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">

      <!-- Use primitive wrapper type (e.g. Integer) as parameter and allow results to
           be auto-mapped results to Person object (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>

      <!-- Use Person object (JavaBean) properties as parameters for insert.  Each of the
           parameters in the #hash# symbols is a JavaBeans property.  -->
      <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>

      <!-- Use Person object (JavaBean) properties as parameters for update. Each of the
           parameters in the #hash# symbols is a JavaBeans property.  -->
      <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>

      <!-- Use Person object (JavaBean)  "id"  properties as parameters for delete. Each of the
           parameters in the #hash# symbols is a JavaBeans property.  -->
      <delete id="deletePerson" parameterClass="examples.domain.Person">
        DELETE PERSON
        WHERE PER_ID = #id#
     </delete>

</sqlMap>
```

SQL Map フレームワークを使ったプログラミング

全ての設定とマップができたので、後は Java アプリケーションのコードが必要です。始めの設定は SQL Map を設定することです。これは、単に以前に作成した SQL Map 設定 XML ファイルをロードすることです。簡単に XML ファイルをロードするために、フレームワークに含まれているリソースクラスを使うことができます。

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

SqlMapClient オブジェクトは、ロングライブでスレッドセーフなオブジェクトです。アプリケーション動作のために一度だけインスタンス化/設定の必要があります。ベースクラス(例.ベース DAO クラス)の static インスタンス変数の良い候補になるでしょう。または、グローバルに参照したいのであれば独自の便利なクラスにラップすることができます。
これは、あなたが書くことができる便利なクラスの実例です。

```
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) {
            // If you get an error at this point, it doesn’t matter what it was.  It is going
to be
            // unrecoverable and we will want the app to blow up hard so we are aware of the
            // problem.  You should always log such errors and re-throw them in such a way that
            // you can be made immediately aware of the problem.
            e.printStackTrace();
            throw new RuntimeException ( “Error initializing MyAppSqlConfig class.  Cause: ” +
e);
        }
    }

    public static SqlMapClient getSqlMapInstance () {
        return sqlMap;
    }

}
```

データベースからオブジェクトの読み込み

SqlMap インスタンスは初期化され容易にアクセスできるようになりました。SqlMap インスタンスを使って始めに、SqlMap インスタンスを使ってデータベースから Person オブジェクトを取得してみましょう。(この実例においては、データベースに PER_ID が 1～10 の PERSON レコードがあると仮定しています。)

データベースから Person オブジェクトを取得するためには、SqlMap インスタンス、マップされたステートメントの名前と Person の番号が必要なだけです。Person 番号 5 のインスタンスを取得してみましょう。

```
…
SqlMapClient sqlMap = MyAppSqlMapConfig.getSqlMapInstance(); // as coded above
…
Integer personPk = new Integer(5);
Person person = (Person) sqlMap.queryForObject ( “getPerson” , personPk);
…
```

データベースへのオブジェクトの書き出し

データベースからPersonオブジェクトを取得しました。いくつかデータを変更してみましょう。personの身長と体重を変更します。

```
…
person.setHeightInMeters(1.83);            // person as read above
person.setWeightInKilograms(86.36);
…
sqlMap.update( “updatePerson” , person);
…
```

もし、このPersonを削除したいのであれば、同じように簡単です。

```
…
sqlMap.delete ( “deletePerson” , person);
…
```

新しいPersonを挿入するのも、同じです。

```
Person newPerson = new Person();
newPerson.setId(11);          // you would normally get the ID from a sequence or custom table
newPerson.setFirstName( “Clinton” );
newPerson.setLastName( “Begin” );
newPerson.setBirthDate (null);
newPerson.setHeightInMeters(1.83);
newPerson.setWeightInKilograms(86.36);
…
sqlMap.insert ( “insertPerson” , newPerson);
…
```

これでチュートリアルは終了です！

次のステップ…

これでチュートリアルは終わりです。http://ibatis.apache.org を訪れてSQL Maps 2.0開発者ガイドとJakarta Struts, iBATIS DAO 2.0とSQL Maps2.0をベースにしたwebアプリケーションのサンプルであるJPetstore 4を見てください。
