iBATIS SQL Maps

Tutorial

para SQL Maps Versión 2.0

Octubre, 2006

Traducción al Español de Kike Mota (kikemota@gmail.com)

Introduction

Este breve tutorial te dará un paseo por el uso típico de SQL Maps. Los detalles de cada uno de los temas tratados aquí los puedes encontrar en la guía de desarrollo con SQL Maps disponible en http://ibatis.apache.org.

Preparado para usar SQL Maps.

El marco de trabajo SQL Maps es muy tolerante tanto con las malas implementaciones de los modelos de datos, como con las malas implementaciones de los modelos de objetos. A pesar de ello, es muy recomendable que uses las *mejores practicas* tanto cuando diseñes tu base de datos (normalización apropiada, etc.), como cuando diseñes tu modelo de objetos. Haciendo esto tendrás garantizado un mejor rendimiento y un diseño más claro.

La forma mas fácil de empezar es analizando con qué estas trabajando. ¿Cuales son tus objetos de negocio?. ¿Cuales son tus tablas?. ¿Como se relacionan?. Para el primer ejemplo, considera el siguiente clase Person que sigue las convenciones típicas de los JavaBeans.

Person.java

```
package examples.domain;

//imports omitidos….

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;
   }
   //…asumimos que aquí tendríamos otros métodos get y set….
}
```

¿Como se mapea esta clase Person a nuestra base de datos?. SQL Maps no te limita a la hora de establecer relaciones del tipo tabla-por-clase o multiples-tablas-por-clase o múltiples-clases-por-tabla. Existen muy pocas restricciones dado que dispones de toda la potencia que ofrece el SQL. Por ejemplo, usemos la siguiente tabla, que podría ser apropiada para una relación tabla-por-clase:

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))
```

El fichero de configuración de SQL Map

Una vez hemos visto las clases y las tablas con las que vamos a trabajar, la mejor manera de empezar a trabajar con SQL Maps es viendo el fichero de configuración. Este fichero actuará como el fichero maestro para la configuración de nuestra implementación basada en SQL Map.

El fichero de configuración es un fichero XML. Dentro de el configuraremos ciertas propiedades, el DataSource JDBC y los mapeos SQL que utilizaremos en nuestra aplicación. Este fichero te ofrece un lugar central donde configurar tu DataSource. El marco de trabajo puede manejar varias implementaciones de DataSources , dentro de los que se incluyen iBATIS están SimpleDataSource, Jakarta DBCP (Commons), y cualquier DataSource que se pueda obtener a través de un contexto JNDI (p.e. Un DataSource configurado dentro de un servidor de aplicaciones). Todo esto se describe con mas detalle en la Guía del Desarrollador. La estructura es simple y para el ejemplo expuesto arriba, debe ser algo como esto:

El ejemplo continua en la siguiente página…

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">

<!-- Asegúrate siempre de usar las cabeceras XML correcta como la de arriba! -->

<sqlMapConfig>

<!--
El fichero de properties especificado aquí es el lugar donde poner  las propiedades (name=value)
que se usarán después en este fichero de configuración donde veamos elementos de configuración
como por ejemplo "${driver}". El fichero suele ser relativo al classpath y  es opcional.-->


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

  <!-- Estas propiedades controlan los detalles de configuración de SqlMap, principalmente las
relacionadas con la gestión de transacciones. Todas son opcionales (ver Guía del Desarrollador
para más información).-->



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

  <!-- typeAlias permite usar un nombre corto en referencia a un nombre cualificado de una clase-->


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

  <!-- Configura un DataSource que podrá ser usado con SQL Map usando SimpleDataSource. Date
cuenta del uso de las propiedades del fichero de recursos de arriba. -->

  <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 todos los ficheros de mapeos XML usados en SQL Map para cargar los mapeos SQL.
Date cuenta de los paths son relativos al classpath. Por ahora, solo tenemos uno... -->

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

</sqlMapConfig>
```

SqlMapConfigExample.properties

```
# Este es un ejemplo de fichero de properties que simplifica la automatización de la configuración
#del fichero de configuración de SQL Maps (ej. A través de Ant o de herramientas para la
#integracióncontinua para diferentes entornos... etc.).
#Estos valores pueden ser usados en cualquier valor de propiedad en el fichero de arriba (ej.
#"${driver}".
#El uso del fichero de propiedades es completamente opcional.


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

Los ficheros SQL Map

Ahora que tenemos configurado un DataSource y listo el fichero de configuración central, necesitaremos proporcionar al fichero de SQL Map nuestro código SQL y los mapeos para cada uno de los objetos parámetro y de los objetos resultado (entradas y salidas respectivamente).

Continuando con nuestro ejemplo de arriba, vamos a construir un fichero SQL Map para la clase Person y la tabla PERSON. Empezaremos con la estructura general de un documento SQL, y una sencilla sentencia SQL:

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>
```

El ejemplo de arriba muestra la forma más sencilla de un mapeo SQL. Usa una característica del marco de trabajo SQL Maps que permiten mapear de forma automática columnas de un ResultSet a propiedades de un JavaBean (o *keys* de un objeto java.util.Map, etc.) basándose en el encaje de los nombres, es decir que encajen los nombres de los campos del ResultSet con los de los atributos de JavaBean. El símbolo #value# es un parámetro de entrada. Más específicamente, el uso de "value" implica que estamos usando un recubrimiento de un tipo primitivo (ej. java.lang.Integer; pero no estamos limitados solo a este).

Aunque es sencillo, hay algunas limitaciones a la hora de usar el enfoque de mapeo mediante auto-resultado. No hay forma de especificar el tipo de las columnas de salida (si fuera necesario) o cargar de

forma automática datos relacionados (propiedades complejas) y además hay unas leves implicaciones de rendimiento ya que requiere el acceso a ResultSetMetaData. Usando *resultmap*, podemos superar todas estas limitaciones. Pero, por ahora la simplicidad es nuestra meta, y además siempre podemos cambiar a un enfoque diferente más adelante (sin cambiar el código fuente Java).

La mayoría de las aplicaciones de base de datos no solo leen de dicha base de datos, sino que también los tienen que modificar. Ya hemos visto un ejemplo de como se mapea una sencilla sentencia SELECT , pero ¿cómo hacerlo para INSERT, UPDATE y DELETE?. La buena noticia es que no es diferente. Abajo completamos nuestro mapeo SQL para la clase Person con mas mapeos que proporciona un conjunto de sentencias para acceder y modificar los datos en la base de datos.

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">

    <!-- Usa un recubrimiento de un  tipo primitivo  (ej. java.lang.Integer) como parámetro y permite
que los resultados sean auto-mapeados a las propiedades del JavaBean 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>

    <!-- Usa las propiedades del JavaBean Person como parámetro para insertar. Cada uno de los
parámetros entre los símbolos #almuadilla# es una propiedad del 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>

    <!-- Usa las propiedades del JavaBean person como parámetro para actualizar. Cada uno de los
parámetros entre los símbolos #almuadilla# es una propiedad del 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>

    <!-- Usa las propiedades del JavaBean person como parámetro para borrar. Cada uno de los
         parámetros entre los símbolos #almuadilla# es una propiedad del JavaBean. -->

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

</sqlMap>
```

Programando con el marco de trabajo SQL Map

Ahora que tenemos todo configurado y mapeado, todo lo que necesitamos es programar nuestra aplicación Java. El primer paso es configurar SQL Map. Es simplemente una forma de de cargar nuestra configuración del fichero XML de SQL Map que creamos antes. Para cargar el fichero XML, haremos uso de la clase Resources incluida en el marco de trabajo.

.

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

Un objeto de la clase SqlMapClient es *thread safe* y será un servicio de larga duración . Solo es necesario instanciarlo/configurarlo una vez para todo el ciclo de vida de una aplicación. Esto lo convierte en un buen candidato para ser un miembro estático de una clase base (ej. una clase base DAO), o si prefieres mantenerlo configurado de forma mas centralizada y disponible de forma más global, podrías crear una clase wrapper para mayor comodidad. Aquí hay un ejemplo de una clase que podría servir para este propósito:

```
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) {
          // Si hay un error en este punto,  no importa cual sea. Será un error irrecuperable del cual
          // nos interesará solo estar informados.
          // Deberás registrar el error y reenviar la excepción de forma que se te notifique  el
          // problema de forma inmediata.


         e.printStackTrace();
         throw new RuntimeException ("Error initializing MyAppSqlConfig class.  Cause: " + e);
       }
   }

   public static SqlMapClient getSqlMapInstance () {
      return sqlMap;
   }

}
```

Leyendo Objetos de la Base de Datos

Ahora que la instancia de SqlMap está inicializada y es accesible de forma sencilla, podemos hacer uso de ella. Usémosla primero para obtener un objeto Person de la base de datos. (Para este ejemplo, asumiremos que hay 10 registros en la tabla PERSON con rangos de PER_ID del 1 al 10).

Para obtener un objeto Person de la base de datos, simplemente necesitamos la instancia de SqlMap, el nombre de la sentencia a ejecutar y un Person ID. Carguemos por ejemplo el objeto Persona cuyo "id" es el número 5.

```
…
SqlMapClient sqlMap = MyAppSqlMapConfig.getSqlMapInstance(); // Como se codificó arriba
…
Integer personPk = new Integer(5);
Person person = (Person) sqlMap.queryForObject ("getPerson", personPk);
…
```

Escribiendo objetos en la Base de Datos.

Ahora que tenemos un objeto Person de la base de datos. Modifiquemos algunos datos. Cambiaremos por ejemplo la altura y el peso de la persona.

```
…
person.setHeightInMeters(1.83);          // objeto de arriba.
person.setWeightInKilograms(86.36);
…
sqlMap.update("updatePerson", person);
…
```

Si queremos borrar el objeto Person, es igual de fácil.

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

La inserción de un nuevo objeto Person es similar.

```
Person newPerson = new Person();
newPerson.setId(11); // normalmente obtendrás el ID de una secuencia o de una tabla personalizada
newPerson.setFirstName("Clinton");
newPerson.setLastName("Begin");
newPerson.setBirthDate (null);
newPerson.setHeightInMeters(1.83);
newPerson.setWeightInKilograms(86.36);
…
sqlMap.insert ("insertPerson", newPerson);
…
```

¡Esto es todo lo que hay!

Próximos pasos…

Este es el final de este breve tutorial. Por favor visita http://ibatis.apache.org para ver una completa Guía del Desarrollador de SQL Maps 2.0, así como JPetStore 4, que es un ejemplo completo y funcional de una aplicación web basada en Jakarta Struts, iBATIS DAO 2.0 y SQL Maps 2.0.

CLINTON BEGIN NO OFRECE NINGUNA GARANTIA, EXPRESA O IMPLICITA, SOBRE LA INFORMACIÓN CONTENIDA EN ESTE DOCUMENTO.