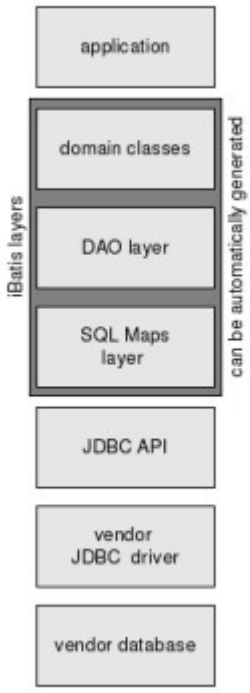
Introduction To iBatis

Abstract

The iBatis data mapping framework for Java, .NET and Ruby increases developer productivity by facilitating database storage at a higher level than say, JDBC, without the complexity of larger object-relational mapping frameworks. This introduction is intended to show Java developers how to quickly integrate iBatis with a legacy MySQL database.

Introduction

The iBatis framework is a lightweight data mapping framework and persistence API that can be used to quickly leverage a legacy database schema to generate a database persistence layer for your Java application. A set of XML encoded SQL Map files–one for each database table–holds SQL templates that are executed as prepared statements and map the expected results to Java domain classes. From application code, a layer of iBatis Data Access Objects (DAO) acts as the API that executes the SQL Map templates and assigns the results to the corresponding Java domain classes. Therefore, the architectural stack looks like this:



This tutorial will show how to automatically generate the code for these three layers from an existing database. To enable communication among these layers, an XML configuration file describes each DAO interface and implementation class, as well as the location of a second XML configuration file that in turn points to each SQL Map file and contains the database connection information.

This tutorial will focus on using iBatis in a Java application and a legacy MySQL database. Abator, a code generation tool for creating the files mentioned above, will also be introduced, but otherwise, only basic

familiarity with Java, XML, and SQL are assumed along with a few common Java tools and libraries including Eclipse, Ant and Log4J.

prerequisites

This introduction assumes a fairly standard project directory structure, including these directories and files directly under the project root:

- src/
- lib/
- dist/
- doc/
- etc/
- build.xml

Other XML files and Java *.properties files not automatically generated by Abator are assumed to be directly under src/ . This introduction shows iBatis configured for a database running on localhost, so obviously a locally running MySQL server is needed.

The MySQL Connector/J JDBC driver should be in your lib/ directory:

> http://www.mysql.com/products/connector/j/

Also download the iBatis JAR files to the lib/ directory:

> http://ibatis.apache.org/javadownloads.html

To add the Abator plugin for iBatis to an Eclipse installation, use this update site:

> http://ibatis.apache.org/tools/abator

However, the author recommends downloading the Abator JAR file to the project's lib/ directory to more flexibly call Abator's code generation functions from Ant, and to use Abator more portably:

> http://ibatis.apache.org/abator.html

Make sure Eclipse sees the Abator, iBatis and Connector/J jars by refreshing the project and adding the jars to the project build path.

Minimal Implementation

This introduction will be based on the following database schema for a simplistic bookmark database:

```
use mysql;
drop database if exists bookmarks;
create database bookmarks;
use bookmarks;
```

```
drop table if exists users;
create table users (
  id                        integer   (10)    auto_increment,
  first_name                varchar   (50),
  last_name                 varchar   (50),
  registered_on             datetime,
  last_logon                datetime,

  primary key (id)
) type = innodb;

insert into users (first_name, last_name) values ('test', 'user');

drop table if exists urls;
create table urls (
  id                        integer   (10)    auto_increment,
  users_id                  integer   (10),
  url                       varchar   (255),
  title                     varchar   (255),
  description               text,
  added_on                  datetime,

  primary key (id),
  index(users_id),
  foreign key (users_id)
    references users (id)
    on delete cascade
) type = innodb;

USE mysql;

GRANT SELECT, INSERT, UPDATE, DELETE
ON bookmarks.*
TO bk_user@'localhost'
IDENTIFIED BY 'bk_password';

use bookmarks;
```

Once this sample database is installed in your local MySQL server, begin creating an Ant build file for the project with the following properties that correspond to the directory structure presented above:

```
        <property name="src.dir"                value="src"/>
        <property name="build.dir"              value="build"/>
        <property name="lib.dir"                value="lib" />
        <property name="dist.dir"               value="dist"/>

        <property name="javadoc.dir"            value="doc/javadoc"/>
        <property name="etc.dir"                value="etc"/>

        <property file="${src.dir}/configuration.properties"/>
```

The referenced variables follow typical naming conventions you may already use. The referenced configuration.properties file will hold the database connection information as specified above and the location where our automatically generated iBatis files will reside:

```
database.driver=org.gjt.mm.mysql.Driver
database.url=jdbc:mysql:///bookmarks
database.username=bk_user
database.password=bk_password
database.driver.jar.path=lib/mysql-connector-java-3.1.13-bin.jar
```

```
ibatis.generated.source.dir=net/stonemind/models
ibatis.generated.source.pkg=net.stonemind.models
```

Note that in addition to the database connection information, you need to give an exact relative path to the JDBC driver jar file. The values in configuration.properties will not only be used by the build file, but also by several iBatis configuration files that follow. For now, add the relevant Ant target for automatically generating iBatis files:

```
  <target name="generate-ibatis" description="Generate iBatis Code">
        <taskdef name="abator"
              classname="org.apache.ibatis.abator.ant.AbatorAntTask"
                                    classpathref="classpath"/>
        <abator overwrite="true" configfile="${src.dir}/abatorConfig.xml" verbose="true"
>
                <propertyset>
                        <propertyref name="ibatis.generated.source.pkg"/>
                        <propertyref name="database.driver" />

                        <propertyref name="database.url" />
                        <propertyref name="database.username" />
                        <propertyref name="database.password" />
                        <propertyref name="database.driver.jar.path" />
                </propertyset>
        </abator>

        <!-- This isn't done by abator but is something you commonly want.
              Works only if the primary key is always called "id" -->
        <replace dir="${src.dir}/${ibatis.generated.source.dir}">
          <include name="**/*.xml"/>
          <replacetoken><![CDATA[</insert>]]></replacetoken>

          <replacevalue><![CDATA[
              <selectKey resultClass="int" keyProperty="id">
                SELECT LAST_INSERT_ID() AS id
              </selectKey>
            </insert>]]>
          </replacevalue>
        </replace>

  </target>
```

The basic generate-ibatis target shown above is taken from documentation on the iBatis Web site (http://ibatis.apache.org/docs/tools/abator/running.html), and adapted to use the values from configuration.properties. A replace task has also been added so that when new records are inserted into the database, iBatis will automatically query for the primary key value assigned to each record and assign that value in turn to the appropriate field in that table's mapped Java class. Abator doesn't generate this code automatically because the method for getting new primary key values varies among database implementations, but this functionality is usually desirable, and the replace task shown here is written specifically for MySQL. So, after Abator introspects the database and generates the SQL Map files, each of the generated insert blocks in those files will be modified to add this select block:

```
  <selectKey resultClass="int" keyProperty="id">

        SELECT LAST_INSERT_ID() AS id
  </selectKey>
```

In this select block, “id” is the name of the field representing the primary key. If the primary key is always named the same way in each table, and the replace task reflects that naming convention, then the replace task will correctly update each insert block in each SQL Map file accordingly. Note that each statement in the block is executed in order by default, so for example, a full insert block in the users SQL Map file will look like this:

```
  <insert id="abatorgenerated_insert" parameterClass="net.stonemind.models.Urls">
        insert into urls (id, users_id, url, title, added_on, description)
        values (#id:INTEGER#, #usersId:INTEGER#, #url:VARCHAR#,
        #title:VARCHAR#, #addedOn:TIMESTAMP#, #description:LONGVARCHAR#)

        <selectKey resultClass="int" keyProperty="id">

                SELECT LAST_INSERT_ID() AS id
        </selectKey>
  </insert>
```

Every time the generate-ibatis Ant task is executed, Abator regenerates every SQL Map file and automatically inserts this select block. You can also tell Abator not to overwrite existing SQL Map files if you want by changing the overwrite=”true” property in the abator Ant task.

As the generate-ibatis Ant task indicates, an abatorConfig.xml file must be available to inform Abator about how to connect to the database, which tables to introspect for code generation and what types of Java classes to generate and where they should be placed:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE abatorConfiguration PUBLIC "-//Apache Software Foundation//DTD Abator for
iBATIS Configuration 1.0//EN"
  "http://ibatis.apache.org/dtd/abator-config_1_0.dtd">

<abatorConfiguration>
  <abatorContext>
    <jdbcConnection driverClass="${database.driver}"
        connectionURL="${database.url}"
        userId="${database.username}"
        password="${database.password}">
      <classPathEntry location="${database.driver.jar.path}" />
    </jdbcConnection>

    <javaModelGenerator targetPackage="${ibatis.generated.source.pkg}"
        targetProject="src">

        <property name="enableSubPackages" value="true" />
        <property name="trimStrings" value="true" />
    </javaModelGenerator>

    <sqlMapGenerator targetPackage="${ibatis.generated.source.pkg}"
        targetProject="src">
        <property name="enableSubPackages" value="true" />
        <property name="trimStrings" value="true" />

    </sqlMapGenerator>

    <daoGenerator targetPackage="${ibatis.generated.source.pkg}"
        targetProject="src" type="IBATIS">
        <property name="enableSubPackages" value="true" />
    </daoGenerator>

    <table schema="bookmarks" tableName="users" />
```

```
    <table schema="bookmarks" tableName="urls" />

  </abatorContext>
</abatorConfiguration>
```

Again, this file is based on an example from the iBatis Web site (http://ibatis.apache.org/docs/tools/abator/configreference/xmlconfig.html); refer to it for a full explanation of each option. If you already have Java classes for your domain objects, simply leave out the javaModelGenerator block. The iBatis DAO layer can use your existing domain classes as is. Either way, with this file, you should now be able to successfully run the generate-ibatis Ant task to have Abator generate Java domain classes, iBatis DAOs and SQL Map files for you. After this is complete, you simply have to finish configuring iBatis by creating an XML file describing the DAOs, and another XML file to describe the SQL Map files. These two configuration files allow fine grained control of the many options iBatis exposes, although for simplicity in this tutorial, I will show only the most basic–but fully functional–implementation.

First, create an new XML file, called dao.xml by convention, directly under src/. This file describes the DAO components generated by Abator:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE daoConfig
   PUBLIC "-//ibatis.apache.org//DTD DAO Configuration 2.0//EN"
   "http://ibatis.apache.org/dtd/dao-2.dtd">

<daoConfig>
    <properties resource="configuration.properties" />
        <context>
                <transactionManager type="SQLMAP">
                        <property name="SqlMapConfigResource" value="SqlMapConfig.xml" />
                </transactionManager>

                <!-- DAO interfaces and implementations should be listed here -->
                <dao interface="${ibatis.generated.source.pkg}.UsersDAO"
                     implementation="${ibatis.generated.source.pkg}.UsersDAOImpl" />
                <dao interface="${ibatis.generated.source.pkg}.UrlsDAO"
                     implementation="${ibatis.generated.source.pkg}.UrlsDAOImpl" />

        </context>

</daoConfig>
```

Then, the SQL Map layer needs the following file that describes how to connect to the database, the type of transaction manager to use (refer to the iBatis documentation for a full explanation of transaction manager choices), and where to find the SQL Map files generated by Abator. This file is also stored directly under src/, called SqlMapConfig.xml by convention, and is referred to in dao.xml file above:

```
<?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">

<sqlMapConfig>
        <properties resource="configuration.properties" />

        <!-- Statement namespaces are required for Abator -->
```

```
        <settings useStatementNamespaces="true" />

        <!-- Setup the transaction manager and data source that are
                appropriate for your environment-->
        <transactionManager type="JDBC">
                <dataSource type="SIMPLE">
                        <property name="JDBC.Driver" value="${database.driver}" />
                        <property name="JDBC.ConnectionURL" value="${database.url}" />

                        <property name="JDBC.Username" value="${database.username}" />
                        <property name="JDBC.Password" value="${database.password}" />
                </dataSource>
        </transactionManager>

        <!-- SQL Map XML files should be listed here -->
        <sqlMap resource="${ibatis.generated.source.dir}/users_SqlMap.xml" />

        <sqlMap resource="${ibatis.generated.source.dir}/urls_SqlMap.xml" />

</sqlMapConfig>
```

You can now begin using iBatis to interact with your database within your application. Here is some sample code to show what this might look like. This is a partially refactored helper class with some basic DAO functions that use the default “auto commit” behavior of iBatis:

```
package net.stonemind;
import java.util.List;

import net.stonemind.models.*;
import com.ibatis.dao.client.DaoManager;

public class ModelHelper {

        private DaoManager daoManager = null;

        public ModelHelper(DaoManager daoManager) {
                super();
                this.daoManager = daoManager;
        }

        public void addUrl(Urls aUrl) {
                UrlsDAO urlsDAO = (UrlsDAO) daoManager.getDao(UrlsDAO.class);
                urlsDAO.insert(aUrl);
        }

        public Urls getUrl(int usersId, String url) {
                UrlsExample example = new UrlsExample();
                example.setUrl(url);
                example.setUrl_Indicator(UrlsExample.EXAMPLE_EQUALS);
                example.setUsersId(usersId);
                example.setUsersId_Indicator(UrlsExample.EXAMPLE_EQUALS);

                UrlsDAO UrlsDAO = (UrlsDAO) daoManager.getDao(UrlsDAO.class);
                List results = UrlsDAO.selectByExampleWithBLOBs(example);
                Urls aUrl = null;
                for (Object result: results) {
                        aUrl = (Urls) result;
                }
                return aUrl;
        }
```

```
        public Urls getOrAddURL(int usersId, String url) {

                Urls aUrl = getUrl(usersId, url);
                if (aUrl == null) {
                        aUrl = new Urls();
                        aUrl.setUrl(url);
                        aUrl.setUsersId(usersId);
                        addUrl(aUrl);
                }

                return aUrl;
        }

        public void deleteUrl(int usersId, String url) {
              deleteUrl(getUrl(usersId, url).getId());
        }

        public void deleteUrl(int id) {
                UrlsDAO urlsDAO = (UrlsDAO) daoManager.getDao(UrlsDAO.class);
                urlsDAO.deleteByPrimaryKey(id);
        }

}
```

In this helper class, instances of DAO classes provide data persistence, transferring values back and forth to domain class instances. This is accomplished without putting SQL or iBatis framework code in the domain classes or application code.

The iBatis documentation gives more complete examples, including how to define transactions consisting of multiple DAO operations. Here is a test case for the helper class showing how the DAO layer is bootstrapped by passing in the dao.xml configuration file:

```
package net.stonemind.test;

import net.stonemind.ModelHelper;
import net.stonemind.models.*;

import com.ibatis.common.resources.Resources;
import com.ibatis.dao.client.DaoManagerBuilder;

import junit.framework.TestCase;

public class ModelHelperTest extends TestCase {

        private ModelHelper modelHelper = null;

        private int testUsersId = 1;
        private String testUrlString = "http://www.stonemind.net";

        public void setUp() throws Exception {
                modelHelper = new ModelHelper(DaoManagerBuilder.buildDaoManager(
                                Resources.getResourceAsReader("dao.xml")));
        }

        public void testAddUrl() throws Exception {
                Urls aUrl = modelHelper.getOrAddURL(testUsersId, testUrlString);
                assertNotNull(aUrl);
                assertTrue(aUrl.getUrl().equals(testUrlString));
                modelHelper.deleteUrl(aUrl.getId());
```

```
        }

        public void testDeleteUrl() throws Exception {
                Urls aUrl = modelHelper.getOrAddURL(testUsersId, testUrlString);
                modelHelper.deleteUrl(aUrl.getId());
                assertNull(modelHelper.getUrl(testUsersId, testUrlString));
        }

}
```

When you move to a production environment, read the iBatis Developers Guides for both SQL Maps and DAO that can be found on the iBatis site. This will tell you how to modify the XML configuration files for iBatis to control such things as connection pooling and results caching.

iBatis uses the Log4J framework, so to see potentially helpful debugging output, add the appropriate directives to the project's log4j.properties file, which should be located directly under src/ :

```
# SQL Map logging configuration...
log4j.logger.com.ibatis=DEBUG
log4j.logger.com.ibatis.common.jdbc.SimpleDataSource=DEBUG
log4j.logger.com.ibatis.common.jdbc.ScriptRunner=DEBUG
log4j.logger.com.ibatis.SQL Map.engine.impl.SQL MapClientDelegate=DEBUG

log4j.logger.java.sql.Connection=DEBUG
log4j.logger.java.sql.Statement=DEBUG
log4j.logger.java.sql.PreparedStatement=DEBUG
log4j.logger.java.sql.ResultSet=DEBUG
```

Conclusion

The iBatis data mapping framework excels at facilitating access to legacy databases, particularly when developing new applications for such databases. By populating a few small configuration files, you can use the Abator code generation tool to effortlessly introspect an existing database and create Java domain classes and a Data Access Object layer without injecting framework code into the database schema or your domain classes.

Although the example I gave was for a Java application and MySQL database, iBatis can also be used in .NET and Ruby applications and for multiple database platforms, such as those with JDBC drivers for use within a Java application. The iBatis framework is extremely flexible, giving you the ability to specify various options for database connection pooling, result caching and transaction management. The iBatis framework can even be used in conjunction with Hibernate. This introduction is intended to allow the reader to make use of iBatis quickly. Complete documentation is provided on the iBatis Web site to further fine tune applications using iBatis.

Resources

Introduction to iBatis Source: The source code used in this tutorial, licensed under the GPL.

iBatis Home Page

iBatis downloads and documentation

Abator Home Page

Abator documentation

MySQL JDBC driver (Connector/J)