Sunday, March 11, 2012

MyBatis: A User Guide Companion

/*---[Rationale]---*/

One of the frustrations I have about reading guides on how to learn a new software technology is that they tend to focus on showing a snippet of code that they want to talk about rather than all the code in context. I have seen others comment that this seems to be a trend in many programming books and online guides these days and was not the case in days when Kernighan and Ritchie wrote the seminal *The C Programming Language*.

The MyBatis 3 User Guide is a case in point. It is easy for someone new to it to quickly get confused because they don't understand the context of the snippets. Simple complete working examples that you can try on the command line or in Eclipse would solve this problem. The authors also sometimes refer to getting the example source code, but finding that is not trivial. I ended up pulling their source code from their SVN repo and that has a couple of different test apps mixed together (the blog and jpetstore are two).

They do provide the JPetStore full code "sample", but it is large and intertwined with Stripes and Spring, so not a place for a newbie to start.

So, since I really like MyBatis and want to promote people learning this, here is my contribution to present the simplest possible MyBatis setup that will allow you to read the MyBatis3 documentation and have working code to allow you to try out its examples.

There are a number of good tutorials for MyBatis (exhibit A and exhibit B) on the web, but they either typically jump right into a full fledged example that can be overwhelming or are a full stack example including servlets and app servers, etc. when you just want to understand the basics and try it out isolation. That's the gap I'm trying to fill in this tutorial companion.

/*---["User Guide Companion" Overview]---*/

This is a companion to, not a substitute for, the MyBatis 3 User Guide, so make sure you download that and the mybatis code bundle from the MyBatis website.

This tutorial companion comes in two parts. Since I'm a big believer in "provide an example of the simplest thing that works" - that's what part 1 is: a bare bones, but fully working, setup of a MyBatis-based system. Part two is the set up you'll need for the Blog example that the MyBatis 3 User Guide largely uses. By having this foundation, you can tweak and test a working system as you read through the User Guide.

Here I assume you know how to put jar files on yourCLASSPATH either from the command line or in an IDE like Eclipse.

Download the latest MyBatis bundle from the MyBatis website here. Put mybatis-3.x.x.jar in your CLASSPATH (Java Build Path > Libraries in Eclipse). You will need to also download the JDBC jar for the database you are using.

I provide .sql files for creating tables and initial data sets for both PostgreSQL and MySQL. You can get all the code I reference here from my GitHub repo.

Tutorial Companion Part One: MyBatis101 - The Simplest Thing That Could Work

I call this first application "MyBatis101" and I've created a directory with that name. In that directory, I have created 7 files, including an Ant build.xml file, so in the end it looks like this:

MyBatis101$ tree

Welcome to the ThornyDev blog.

I like names with multiple meanings. Thorny Dev first refers to my interest in software engineering - tackling nontrivial and "thorny" problems.

But the other meaning is about what is probably the coolest lizard in the world - the Australian Thorny Devil.

Followers

Blog Archive

- ► 2015 (2)
- ► 2014 (8)
- ► 2013 (10)
- ▼ 2012 (25)
  - ► December (4)
  - ► October (1)
  - ► September (3)
  - ► August (2)
  - ► July (1)
  - ► June (3)
  - ► May (1)
  - ► April (1)
  - ▼ March (3)
    - Refactoring to Java Enums
    - MyBatis: A User Guide Companion
    - Technical Podcasts I Listen To
  - ► February (3)
  - ► January (3)
- ► 2011 (3)

About Me

Michael Peterson

Follow 26

View my complete profile

```
.
|-- build.xml
|-- MyBatis101.sql
|-- src
    |--MyBatis101-config.xml
    |-- mybatis101
        |-- Main.java
        |-- Mapper.java
        |-- MyBatis101-mapper.xml
        |-- User.java
```

/*---[First: Set up a Database]---*/

First let's set up a very simple database with one table, having two columns and two rows of data.

As I said above, I will show this in both PostgreSQL and MySQL. I don't describe how to install and set up those databases. If you need to start there, here are links to good documentation:

- PostgreSQL: http://www.postgresql.org/docs/manuals/
- MySQL: http://dev.mysql.com/doc/refman/5.1/en/installing.html

In my setup I am using PostgreSQL 9.1 and MySQL 5.1.

I also show these using a Linux command line, but it should work the same on Mac and Windows. I list all the files one by one below, but to avoid copy and paste, be sure to pull them from my from my GitHub repo:

```
git clone git@github.com:midpeter444/MyBatis-UserGuide-Companion-Code.git
```

/*---[PostgreSQL]---*/

Create the MyBatis101 database and check that it is there:

```
$ createdb MyBatis101
$ psql --list
                                  List of databases
    Name    |    Owner    | Encoding | Collation  |   Ctype
------------+-------------+----------+------------+-------------
 depot      | midpeter444 | UTF8     | en_US.utf8 | en_US.utf8
 MyBatis101 | midpeter444 | UTF8     | en_US.utf8 | en_US.utf8
 postgres   | postgres    | UTF8     | en_US.utf8 | en_US.utf8
 template0  | postgres    | UTF8     | en_US.utf8 | en_US.utf8

 template1  | postgres    | UTF8     | en_US.utf8 | en_US.utf8

(5 rows)
```

Create the file MyBatis101.sql:

```sql
drop table if exists users;

create table users (
  id integer,
  name varchar(20)
);

insert into users (id, name) values(1, 'User1');
insert into users (id, name) values(2, 'User2');
```

Create the tables and load test data into the MyBatis101 database and check that it is there:

```
$ psql MyBatis101 < MyBatis101.sql
$ psql MyBatis101
psql (9.1.3)
Type "help" for help.

MyBatis101=> \d
          List of relations
 Schema | Name  | Type  |    Owner
--------+-------+-------+-------------
 public | users | table | midpeter444
(1 row)

MyBatis101=> select * from users;
 id | name
----+-------
  1 | User1
  2 | User2
```

(2 rows)

/*---[MySQL]---*/

Create the MyBatis101 database and check that it is there:

```
$ mysql -p
mysql> create database MyBatis101;
Query OK, 1 row affected (0.03 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| MyBatis101         |
| mysql              |
+--------------------+
3 rows in set (0.03 sec)
```

Create the file MyBatis101.sql: (Same file as above)

Create the tables and load test data into the MyBatis101 database and check that it is there:

```
$ mysql -p MyBatis101 < MyBatis101.sql
$ mysql -p MyBatis101

mysql> show tables;
+----------------------+
| Tables_in_MyBatis101 |
+----------------------+
| users                |
+----------------------+
1 row in set (0.00 sec)

mysql> desc users;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | YES  |     | NULL    |       |
| name  | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

Now that the databases is set up with a table and a little sample table, now we can try out MyBatis.

/*---[The MyBatis Set Up Files]---*/

Create the MyBatis101-config.xml in the src directory:

Note: Use the correct driver and url according to which database you are using.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

 <environments default="development">
  <environment id="development">
   <transactionManager type="JDBC"/>
   <dataSource type="UNPOOLED">
    <property name="driver" value="org.postgresql.Driver" />
    <property name="url" value="jdbc:postgresql:MyBatis101" />
    <!--  <property name="driver" value="com.mysql.jdbc.Driver" />   -->
    <!--  <property name="url" value="jdbc:mysql://localhost:3306/MyBatis101" /> -->
    <property name="username" value="" />
    <property name="password" value="" />
   </dataSource>
  </environment>
 </environments>

 <mappers>
  <mapper resource="MyBatis101-mapper.xml" />
 </mappers>

</configuration>
```

Create MyBatis101-mapper.xml in the src/mybatis101 directory:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="mybatis101.Mapper">

    <select id="getUser" parameterType="int" resultType="mybatis101.User">
        select * from users where id = #{id}
    </select>

</mapper>
```

Create src/mybatis101/Mapper.java:

```java
package mybatis101;

public interface Mapper {
  User getUser(Integer id);
}
```

Create src/mybatis101/User.java:

```java
package mybatis101;

public class User {

  private Integer id;
  private String name;

  public Integer getId() {
    return id;
  }

  public void setId(Integer id) {
    this.id = id;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }
}
```

Create src/mybatis101/Main.java:

```java
package mybatis101;

import java.io.Reader;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

public class Main {

  private static SqlSessionFactory sessionFactory;

  public static void initSessionFactory() throws Exception {
    Reader rdr = Resources.getResourceAsReader("MyBatis101-config.xml");
    sessionFactory = new SqlSessionFactoryBuilder().build(rdr);
    rdr.close();
  }

  // uses the old iBATIS style of lookup
  public static void lookUpUserOldWay() throws Exception {
    SqlSession session = sessionFactory.openSession();
    try {
      User user = (User) session.selectOne(
          "mybatis101.Mapper.getUser", Integer.valueOf(1));
      System.out.println(user.getName());  // should print out "User1"

    } finally {
      session.close();
    }
  }

  // uses the new MyBatis style of lookup
  public static void lookUpUser() throws Exception {
    SqlSession session = sessionFactory.openSession();

    try {
      Mapper mapper = session.getMapper(Mapper.class);
      User user = mapper.getUser(2);
      System.out.println(user.getName());  // should print out "User2"

    } finally {
      session.close();
    }
  }

  public static void main(String[] args) throws Exception {
    initSessionFactory();
    lookUpUserOldWay();
    lookUpUser();
  }
}
```

Create build.xml ant script (not required if you are doing this in Eclipse) and run it:

```xml
<project name="MyBatis101" default="run" basedir=".">
  <!-- Template based on: http://sourceforge.net/apps/mediawiki/import-ant/index.php?tit
>
  <description>Build script for simple MyBatis101 app</description>

  <!-- load environment variables as properties -->
  <property environment="env"/>

  <!-- default folder location properties -->
  <property name="src.dir" value="src"/>
  <property name="build.dir" value="bin"/>
  <!-- TODO: EDIT THESE -->
  <property name="lib.dir" value="/home/midpeter444/java/lib"/>
  <property name="jdbc.jar" value="postgresql.jar"/>
  <!-- <property name="jdbc.jar" value="mysql-connector-java.jar"/> -->

  <!-- project classpath -->

  <path id="project.classpath">
    <!-- compiled classes -->
    <pathelement location="${build.dir}" />
    <!-- libraries -->
    <fileset dir="${lib.dir}">
      <include name="${jdbc.jar}" />
      <include name="mybatis.jar" />    <!-- TODO: EDIT THIS -->
    </fileset>
  </path>

  <!-- basic -->

  <target name="init">
    <mkdir dir="${build.dir}"/>
  </target>

  <!-- compile -->

  <target name="prepare-resources" depends="init">
    <copy todir="${build.dir}" overwrite="true">
      <fileset dir="${src.dir}" includes="**/*.xml" />
    </copy>
  </target>

  <target name="compile" depends="init,prepare-resources"
          description="Compile java classes">
    <javac
        srcdir="${src.dir}"
        destdir="${build.dir}"
        includeantruntime="false"> <!-- to overcome misfeature in An t1.8 -->
      <classpath refid="project.classpath" />
    </javac>
  </target>


  <!-- run on console -->

  <property name="run.main-class" value="mybatis101.Main"/>
  <property name="run.args" value=""/>

  <target name="run" depends="compile"
          description="Run MyBatis101 program">
    <java classname="${run.main-class}" fork="true">
      <arg line="${run.args}" />
      <classpath>
        <path refid="project.classpath" />
      </classpath>
    </java>
  </target>
</project>
```

```
$ ant run
Buildfile: /home/midpeter444/databases/postgresql/mybatis-learn/MyBatis101/build.xml

init:

prepare-resources:
    [copy] Copying 2 files to /home/midpeter444/databases/postgresql/mybatis-learn/MyBatis101/bi
n

compile:

run:
    [java] User1
    [java] User2

BUILD SUCCESSFUL
Total time: 2 seconds
```

I'm not going to explain much about this code. Read it along with the User Guide or theMyBatis Getting Started tutorial and it should start to make sense pretty quickly.

Tutorial Companion Part Two: Blog App in the MyBatis3 User Guide

The MyBatis101 code was intended just to get your feet wet with MyBatis. In this section, we briefly peruse the companion code as part of the "blog" application that is mostly referenced in the MyBatis3 User Guide. *Note:* the references to the blog database structure and codebase are not consistent in the MyBatis3 User Guide, so I've done the best I can at finding something close to most examples.

This code is intended as starter code - it is a fully working example, with a Main.java that exercises a couple of MyBatis query mappings and one insert mapping. I have also provided one JUnit 4 test that only tests one of the query mappings.

This code along with the User Guide could be used as a sort of poor man's koan - a series of exercises to be filled out with more functionality.

The routine will be the same as above, we just have more of it. Again, pull all the files from my GitHub account.

I have created a directory called "blog" and the code base structure I provide looks like this:

```
blog$ tree
.
|-- blogdb-ddl-mysql.sql
|-- blogdb-ddl-postgres.sql
|-- blogdb-dml.sql
|-- build.xml
|-- src
    |-- main
        |-- java
            |-- org
                |-- mybatis
                    |-- example
                        |-- Author.java
                        |-- AuthorMapper.xml
                        |-- Blog.java
                        |-- BlogMapper.java
                        |-- BlogMapper.xml
                        |-- config.properties
                        |-- Configuration.xml
                        |-- Main.java
    |-- test
        |-- java
            |-- org
                |-- mybatis
                    |-- example
                        |-- BlogMapperTests.java
```

We start by creating the database and tables for theBlog and Author classes they discuss in the MyBatis3 User Guide.

/*---[PostgreSQL]---*/

Create the database:

```
$ createdb blogdb
```

Create the DDL for the blog and author tables and save it in a file called "blogdb-ddl-postgres.sql":

```sql
DROP TABLE    IF EXISTS blog;
DROP TABLE    IF EXISTS author;
DROP SEQUENCE IF EXISTS blogdb_blog_seq;
DROP SEQUENCE IF EXISTS blogdb_author_seq;
CREATE SEQUENCE blogdb_blog_seq;
CREATE SEQUENCE blogdb_author_seq;

CREATE TABLE author (
  id              integer PRIMARY KEY DEFAULT nextval('blogdb_author_seq') NOT NULL,
  username        varchar(255) NOT NULL CHECK (username <> ''),
  hashed_password varchar(255) NOT NULL CHECK (hashed_password <> ''),
  email           varchar(100) NOT NULL CHECK (email <> ''),
  bio             text
);

CREATE TABLE blog (
  id        integer PRIMARY KEY DEFAULT nextval('blogdb_blog_seq') NOT NULL,
  title     varchar(255) NOT NULL CHECK (title <> ''),
  author_id integer NOT NULL references author(id)
);
```

Create some fake data to load into the tables and saved it in a file called "blogdb-dml.sql":

```
INSERT into author (username, hashed_password, email, bio)
VALUES('aaron1', 'aaron1', 'aaron@pobox.com', 'Aaron is "The Dude".');

INSERT into author (username, hashed_password, email)
VALUES('barb2', 'barb2', 'barb@pobox.com');

INSERT into author (username, hashed_password, email, bio)
VALUES('carol3', 'carol3', 'carol@pobox.com', 'Carol is an avid atom-smasher and street

INSERT into blog (title, author_id)
VALUES('Why I am "The Dude"', (select id from author where username='aaron1'));

INSERT into blog (title, author_id)
VALUES('A Day in the Life of "The Dude"', (select id from author where username='aaror

INSERT into blog (title, author_id)
VALUES('Sanity is my strong suit', (select id from author where username='barb2'));

INSERT into blog (title, author_id)
VALUES('I are smart?', (select id from author where username='carol3'));

INSERT into blog (title, author_id)
VALUES('The Large-Hadron Collider will not create a black hole that ends the universe',
;
```

Run the DDL and DML scripts against the database:

```
$ psql blogdb < blogdb-ddl-postgres.sql
$ psql blogdb < blogdb-dml.sql
```

/*---[MySQL]---*/

Create the database:

```
$ mysql -p
mysql> create database blogdb;
Query OK, 1 row affected (0.03 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| MyBatis101         |
| blogdb             |
| mysql              |
+--------------------+
4 rows in set (0.03 sec)
```

Create the DDL for the blog and author tables and save it in a file called "blogdb-ddl-mysql.sql":

```
DROP TABLE    IF EXISTS blog;
DROP TABLE    IF EXISTS author;

CREATE TABLE author (
  id              integer NOT NULL AUTO_INCREMENT PRIMARY KEY,
  username        varchar(255) NOT NULL CHECK (username <> ''),
  hashed_password varchar(255) NOT NULL CHECK (hashed_password <> ''),
  email           varchar(100) NOT NULL CHECK (email <> ''),
  bio             text
)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;

CREATE TABLE blog (
  id         integer NOT NULL AUTO_INCREMENT PRIMARY KEY,
  title      varchar(255) NOT NULL CHECK (title <> ''),
  author_id  integer NOT NULL,
  FOREIGN KEY (author_id) REFERENCES author(id)
)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Create some fake data to load into the tables and save it in a file called "blogdb-dml.sql":(Same file as in the PostgreSQL section.)

Run the DDL and DML scripts against the database:

```
$ mysql -p blogdb < blogdb-ddl-mysql.sql
$ mysql -p blogdb < blogdb-dml.sql
```

/*---[Work Through The User Guide]---*/

Next, start by looking at Main.java in src/main/java/org/mybatis/example. Follow each of its steps to see how the MyBatis system works. This example includes a complex mapping - what MyBatis calls a resultMap. On p. 29 of the User Guide they go into some detail about it, as it is one of the most important features of MyBatis.

From this code base you should be able to work through the entire User Guide, trying out new features as you go. Make sure to write tests for everything you do. I've intentionally made the JUnit test very bare bones so that is the place you can learn by doing.

After doing this, I recommend trying out one of the MyBatis tutorials on the web, such as either of those that I already mentioned above:

- http://loianegroner.com/2011/02/introduction-to-ibatis-mybatis-an-alternative-to-hibernate-and-jdbc/
- http://blog.idleworx.com/2011/02/mybatis-3-tomcat-mysql-eclipse.html

Good luck and feedback is welcome.

[Update 28-May-2012]: I've recently published a set of koans to learn MyBatis, so consider trying those out as well. My May 2012 blog entry describes these and how to get started.

Posted by Michael Peterson at 5:01 PM

Labels: mybatis, mysql, postgresql

5 comments:

Eduardo March 14, 2012 at 12:04 AM

Great post Michael. Hope you don't mind I linked it in http://code.google.com/p/mybatis/wiki/FeedbackArticles

Reply

Replies

Michael Peterson March 14, 2012 at 1:02 AM

Hi Eduardo - thanks very much. Glad you liked it despite my "complaining" at the start. Hope it helps the MyBatis community - and thanks for such a great tool.

Reply

Prakash Manjeshwar June 14, 2012 at 11:40 AM

Michael,

You made my day... with this wonderful tutorial. Many thanks.

I was considering Hibernate/JPA/JDO for a very small application consisting of about ~20 domain objects. But all three of them seemed too "heavy" for the small application. Plain JDBC was my final choice (almost). But i(My)Batis was always in my mind, but documentation and examples were my concerns.

After going through this tutorial and the user guide, I am now going to use MyBatis.

Thanks once again.

Cheers
Prakash

Reply

Michael Peterson June 16, 2012 at 10:14 AM

Hi Prakash,

Very glad this companion guide was helpful and you are going with MyBatis. I think you made the right choice :-)

-Michael

Reply

□□□□ September 22, 2017 at 8:51 PM

I just succeeded in following the part one. However, I had to struggle solving a couple of compilation errors. That is in the project build path included these external jar files : log4j-1.2.17.jar, mybatis-3.4.5.jar, mysql-connector-java-5.1.34_1.jar (these versions matter).

Reply

Add comment

Enter your comment...

Comment as: Google Accoun

Publish Preview