Maven + Spring + Hibernate + MySql Example
Written on March 23, 2010 at 9:39 am by mkyong

This example will use Maven to create a simple Java project structure, and demonstrate how to use Hibernate in Spring framework to do the data manipulation works(insert, select, update and delete) in MySQL database.

Prerequisite requirement

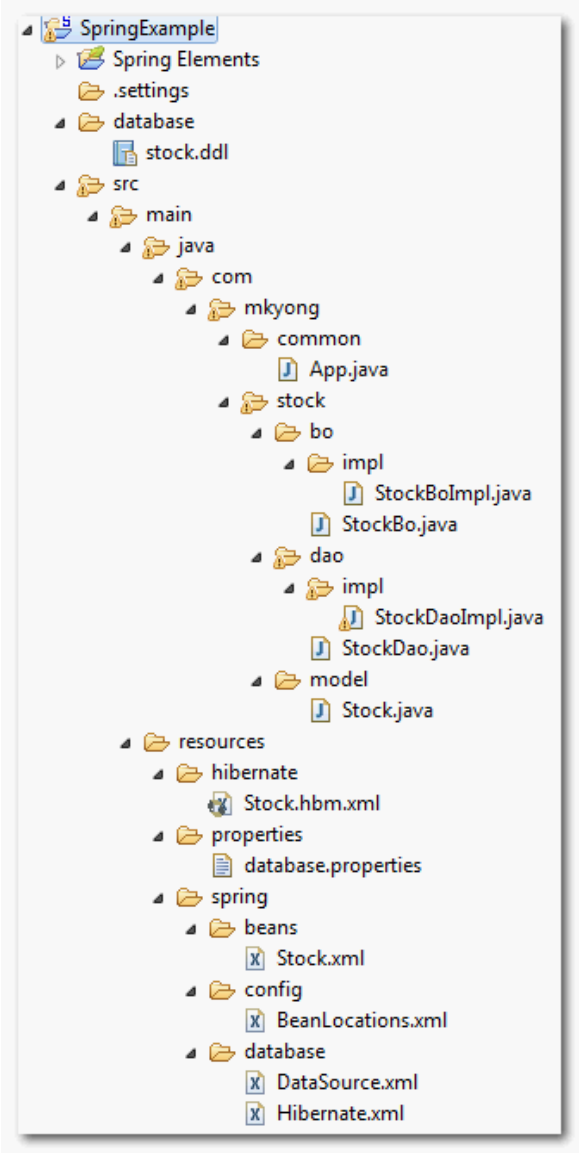
- Installed and configured Maven, MySQL, Eclipse IDE.

Tutorial…

Download this Maven + Spring + Hibernate + MySql Example here – Spring-Hibernate-Example.zip

Final project structure

Your final project file structure should look exactly like following, if you get lost in the folder structure creation, please review this folder structure here.



1. Table creation

Create a 'stock' table in MySQL database. SQL statement as follow :

```
CREATE TABLE `mkyong`.`stock` (
  `STOCK_ID` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
  `STOCK_CODE` varchar(10) NOT NULL,
  `STOCK_NAME` varchar(20) NOT NULL,
  PRIMARY KEY (`STOCK_ID`) USING BTREE,
  UNIQUE KEY `UNI_STOCK_NAME` (`STOCK_NAME`),
  UNIQUE KEY `UNI_STOCK_ID` (`STOCK_CODE`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8;
```

2. Project File Structure

Create a quick project file structure with Maven command '**mvn archetype:generate**', see example here. Convert it to Eclipse project (**mvn eclipse:eclipse**) and import it into Eclipse IDE.

```
E:\workspace>mvn archetype:generate
[INFO] Scanning for projects...
...
```

```
Choose a number:
(1/2/3....) 15: : 15
...
Define value for groupId: : com.mkyong.common
Define value for artifactId: : HibernateExample
Define value for version:  1.0-SNAPSHOT: :
Define value for package:  com.mkyong.common: : com.mkyong.common
[INFO] OldArchetype created in dir: E:\workspace\HibernateExample
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
```

3. Pom.xml file configuration

Add the Spring, Hibernate , MySQL and their dependency in the Maven's pom.xml file.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mkyong.common</groupId>
  <artifactId>SpringExample</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>SpringExample</name>
  <url>http://maven.apache.org</url>

  <dependencies>

        <!-- JUnit testing framework -->
        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>3.8.1</version>
                <scope>test</scope>
        </dependency>

        <!-- Spring framework -->
        <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring</artifactId>
                <version>2.5.6</version>
        </dependency>

        <!-- Spring AOP dependency -->
        <dependency>
                <groupId>cglib</groupId>
                <artifactId>cglib</artifactId>
                <version>2.2</version>
        </dependency>

        <!-- MySQL database driver -->
        <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
                <version>5.1.9</version>
        </dependency>

        <!-- Hibernate framework -->
        <dependency>
                <groupId>hibernate</groupId>
                <artifactId>hibernate3</artifactId>
                <version>3.2.3.GA</version>
        </dependency>
```

```
        <!-- Hibernate library dependecy start -->
        <dependency>
                <groupId>dom4j</groupId>
                <artifactId>dom4j</artifactId>
                <version>1.6.1</version>
        </dependency>

        <dependency>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
                <version>1.1.1</version>
        </dependency>

        <dependency>
                <groupId>commons-collections</groupId>
                <artifactId>commons-collections</artifactId>
                <version>3.2.1</version>
        </dependency>

        <dependency>
                <groupId>antlr</groupId>
                <artifactId>antlr</artifactId>
                <version>2.7.7</version>
        </dependency>
        <!-- Hibernate library dependecy end -->

  </dependencies>
</project>
```

4. Model & BO & DAO

The **Model**, **Business Object** (BO) and **Data Access Object** (DAO) pattern is useful to identify the layer clearly to avoid mess up the project structure.

Stock Model

A Stock model class to store the stock data later.

```
package com.mkyong.stock.model;

import java.io.Serializable;

public class Stock implements Serializable {

        private static final long serialVersionUID = 1L;

        private Long stockId;
        private String stockCode;
        private String stockName;

        //getter and setter methods...
}
```

Stock Business Object (BO))

Stock business object (BO) interface and implementation, it's used to store the project's business function, the real database operations (CRUD) works should not involved in this class, instead it has a DAO (StockDao) class to do it.

```
package com.mkyong.stock.bo;

import com.mkyong.stock.model.Stock;

public interface StockBo {

        void save(Stock stock);
        void update(Stock stock);
        void delete(Stock stock);
        Stock findByStockCode(String stockCode);
}
```

```
package com.mkyong.stock.bo.impl;

import com.mkyong.stock.bo.StockBo;
import com.mkyong.stock.dao.StockDao;
import com.mkyong.stock.model.Stock;

public class StockBoImpl implements StockBo{

        StockDao stockDao;

        public void setStockDao(StockDao stockDao) {
                this.stockDao = stockDao;
        }

        public void save(Stock stock){
                stockDao.save(stock);
        }

        public void update(Stock stock){
                stockDao.update(stock);
        }

        public void delete(Stock stock){
                stockDao.delete(stock);
        }

        public Stock findByStockCode(String stockCode){
                return stockDao.findByStockCode(stockCode);
        }
}
```

Stock Data Access Object

A Stock DAO interface and implementation, the dao implementation class extends the Spring's "**HibernateDaoSupport**" to make Hibernate support in Spring framework. Now, you can execute the Hibernate function via **getHibernateTemplate()**.

```
package com.mkyong.stock.dao;

import com.mkyong.stock.model.Stock;

public interface StockDao {

        void save(Stock stock);
        void update(Stock stock);
        void delete(Stock stock);
        Stock findByStockCode(String stockCode);

}
```

```
package com.mkyong.stock.dao.impl;

import java.util.List;

import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

import com.mkyong.stock.dao.StockDao;
import com.mkyong.stock.model.Stock;

public class StockDaoImpl extends HibernateDaoSupport implements StockDao{

        public void save(Stock stock){
                getHibernateTemplate().save(stock);
        }

        public void update(Stock stock){
                getHibernateTemplate().update(stock);
        }

        public void delete(Stock stock){
                getHibernateTemplate().delete(stock);
        }
```

```
        public Stock findByStockCode(String stockCode){
                List list = getHibernateTemplate().find(
                      "from Stock where stockCode=?",stockCode
                );
                return (Stock)list.get(0);
        }

}
```

5. Resource Configuration

Create a '**resources**' folder under '**project_name/main/java/**', Maven will treat all files under this folder as resources file. It will used to store the Spring, Hibernate and others configuration file.

Hibernate Configuration

Create a Hibernate mapping file (**Stock.hbm.xml**) for Stock table, put it under "**resources/hibernate/**" folder.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="com.mkyong.stock.model.Stock" table="stock" catalog="mkyong">
        <id name="stockId" type="java.lang.Long">
            <column name="STOCK_ID" />
            <generator class="identity" />
        </id>
        <property name="stockCode" type="string">
            <column name="STOCK_CODE" length="10" not-null="true" unique="true" />
        </property  >
        <property name="stockName" type="string">
            <column name="STOCK_NAME" length="20" not-null="true" unique="true" />
        </property>
    </class>
</hibernate-mapping>
```

Spring Configuration

Database related….

Create a properties file (**database.properties**) for the database details, put it into the "**resources/properties**" folder. It's good practice disparate the database details and Spring bean configuration into different files.

database.properties

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/mkyong
jdbc.username=root
jdbc.password=password
```

Create a "dataSource" bean configuration file (**DataSource.xml**) for your database, and import the properties from database.properties, put it into the **"resources/database**" folder.

DataSource.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location">
                <value>properties/database.properties</value>
        </property>
</bean>
```

```
<bean id="dataSource"
         class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="${jdbc.driverClassName}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
</bean>

</beans>
```

Hibernate related….

Create a session factory bean configuration file **(Hibernate.xml)**, put it into the **"resources/database"** folder. This **LocalSessionFactoryBean** class will set up a shared Hibernate SessionFactory in a Spring application context.

Hibernate.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<!-- Hibernate session factory -->
<bean id="sessionFactory"
     class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

    <property name="dataSource">
      <ref bean="dataSource"/>
    </property>

    <property name="hibernateProperties">
       <props>
         <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
         <prop key="hibernate.show_sql">true</prop>
       </props>
     </property>

     <property name="mappingResources">
        <list>
           <value>/hibernate/Stock.hbm.xml</value>
        </list>
      </property>

    </bean>
</beans>
```

Spring beans related….

Create a bean configuration file (**Stock.xml**) for BO and DAO classes, put it into the "**resources/spring**" folder. Dependency inject the dao (stockDao) bean into the bo (stockBo) bean; sessionFactory bean into the stockDao.

Stock.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <!-- Stock business object -->
   <bean id="stockBo" class="com.mkyong.stock.bo.impl.StockBoImpl" >
                <property name="stockDao" ref="stockDao" />
   </bean>

   <!-- Stock Data Access Object -->
   <bean id="stockDao" class="com.mkyong.stock.dao.impl.StockDaoImpl" >
                <property name="sessionFactory" ref="sessionFactory"></property>
   </bean>

</beans>
```

Import all the Spring's beans configuration files into a single file (BeanLocations.xml), put it into the "**resources/config**" folder.

BeanLocations.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

        <!-- Database Configuration -->
        <import resource="../database/DataSource.xml"/>
        <import resource="../database/Hibernate.xml"/>

        <!-- Beans Declaration -->
        <import resource="../beans/Stock.xml"/>

</beans>
```

6. Run it

You have all the files and configurations , run it.

```
package com.mkyong.common;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.mkyong.stock.bo.StockBo;
import com.mkyong.stock.model.Stock;

public class App
{
    public static void main( String[] args )
    {
        ApplicationContext appContext =
          new ClassPathXmlApplicationContext("spring/config/BeanLocations.xml");

        StockBo stockBo = (StockBo)appContext.getBean("stockBo");

        /** insert **/
        Stock stock = new Stock();
        stock.setStockCode("7668");
        stock.setStockName("HAIO");
        stockBo.save(stock);

        /** select **/
        Stock stock2 = stockBo.findByStockCode("7668");
        System.out.println(stock2);

        /** update **/
        stock2.setStockName("HAIO-1");
        stockBo.update(stock2);

        /** delete **/
        stockBo.delete(stock2);

        System.out.println("Done");
    }
}
```

output

```
Hibernate: insert into mkyong.stock (STOCK_CODE, STOCK_NAME) values (?, ?)
Hibernate: select stock0_.STOCK_ID as STOCK1_0_,
stock0_.STOCK_CODE as STOCK2_0_, stock0_.STOCK_NAME as STOCK3_0_
from mkyong.stock stock0_ where stock0_.STOCK_CODE=?
Stock [stockCode=7668, stockId=11, stockName=HAIO]
Hibernate: update mkyong.stock set STOCK_CODE=?, STOCK_NAME=? where STOCK_ID=?
Hibernate: delete from mkyong.stock where STOCK_ID=?
Done
```

Popular Tutorials

- Struts Tutorials
- Spring Tutorials
- Maven Tutorials
- Hibernate Tutorials

4 Comments

1. *mignight_blue* says:
   May 13, 2010 at 6:09 pm

   Very well explained and easy to follow. Thanks.

   Reply
2. *Hibernate Tutorials | Tutorials* says:
   May 2, 2010 at 10:02 pm

   [...] Spring + Hibernate Integration Example to integrate Hibernate with Spring framework. [...]

   Reply
3. *Maven + (Spring + Hibernate) Annotation + MySql Example | Spring* says:
   March 31, 2010 at 3:14 pm

   [...] Annotation + MySql Example Written on March 31, 2010 at 3:12 pm by mkyong In last tutorial, you use Maven to create a simple Java project structure, and demonstrate how to use Hibernate in [...]

   Reply

4. *John Ryan* says:
   March 23, 2010 at 6:24 pm

   Very usefull, many thanks.

   Reply

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Submit Comment

Notify me of followup comments via e-mail

Popular Tutorials

- Hibernate Tutorials

  Hibernate is a object/relational persistence tool for Java developers. Hibernate lets you to use object-oriented way to develop your persistent classes, It's also provide many data manipulation API like …

- Maven Tutorials

  Apache Maven is a software project management tool, not just a build tool like ant. It's provides a new concept of a project object model (POM) file, to manage project's …

- Apache Archiva Tutorials

  Apache Archiva is a Build Artifact Repository Manager, a repository management that helps to create a Maven repository in our end.
  To simple, it helps to create a Maven repository …

- Spring Tutorials

  The Spring framework is a powerful and extensible Inversion of control(IoC) container, provides developers a good habit to program to interface, rather than classes to decouple your components, helps developers …

- JUnit Tutorials

  JUnit is an open source testing framework, and an instance of the xUnit architecture for unit testing frameworks. JUnit is simple and suitable used for pure unit testing, for …

- TestNG Tutorials

  TestNG (Next Generation) is a testing framework inspired from JUnit and NUnit, but introducing some new functionalities like dependency testing, grouping concept to make testing more powerful and easier to …

- 
  Java XML Tutorials

  Java comes with two XML parsers – DOM and SAX to process XML file. The others popular thrid party XML parser is JDOM. Here's few examples to show how to …

- 
  Java Regular Expression Tutorials

  Java has comprehensive support for Regular Expression functionality through the java.util.regex package. The regular expression language is easy to learn but hard to master, the better way to learn it …

- 
  Struts Tutorials

  The Struts 1.x framework is a the most famous, classic and proven success MVC framework. Often times, you will listen something like, meaningless to learn Struts 1.x, it's a dead …

Recent Comments

-  Stephane : For a MySql schema with 100 tables, each having some NOT NULL DEFAULT... More
-  Neeraj : No need to set window.onload function at the end of page… More
-  anonymous : Just go to the following site: warez-bb.org and register. Than... More
-  sahil : thanks , but There is no getServletContext(); in WebApplication.get() More
-  Courtney : Hi my name is courtney and Um srry this must be a really really... More
-  jagadish : I gone through the notes and it is very good for begeers. ~Thanks,jaga More
-  mkyong : Sorry, are you tested in IE6? caused it worked fine in IE 7 & 8 More
-  dattai : that's very cool Thanks alot More

-  mkyong : use jdom to read the xml file and constructs any string format you... More
-  mkyong : Java cache? What to do with XML? More

Authors

Hi, my name is Yong Mook Kim, person behind Mkyong.com.

This website is providing daily Java / J2EE web development tutorials, which involve Spring, Hibernate, Struts, Maven, Java Core, Unit Test...

Friends & Links

- China Wholesale
- Dropship
- Investment Life
- Webmaster Forum
- MySQL Tutorials
- PHP Tutorials

- ChanKelwin
- Internet Blogger
- TanWanMei
- TenthOfMarch
- Find Free Icons
- Excel Consultant
- IT Support Liverpool