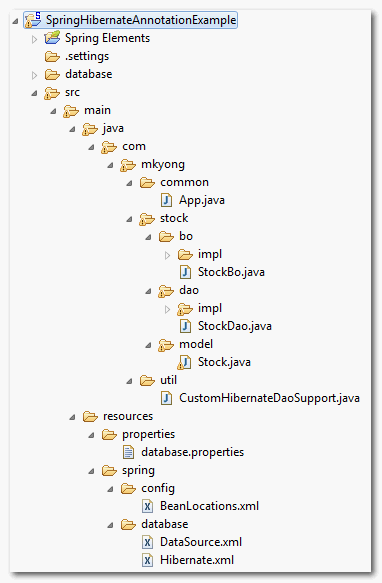
# Maven + (Spring + Hibernate) Annotation + MySql Example

**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](http://www.mkyong.com/maven/how-to-create-a-project-with-maven-template/" \t "_blank). 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, Annotation and 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>

<repositories>

<repository>

<id>JBoss repository</id>

<url>http://repository.jboss.com/maven2/</url>

</repository>

</repositories>

<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 annotation -->

<dependency>

<groupId>hibernate-annotations</groupId>

<artifactId>hibernate-annotations</artifactId>

<version>3.3.0.GA</version>

</dependency>

<dependency>

<groupId>hibernate-commons-annotations</groupId>

<artifactId>hibernate-commons-annotations</artifactId>

<version>3.0.0.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 (Annotation)**

A Stock model annotation class to store the stock data.

package com.mkyong.stock.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import static javax.persistence.GenerationType.IDENTITY;

import javax.persistence.Id;

import javax.persistence.Table;

import javax.persistence.UniqueConstraint;

@Entity

@Table(name = "stock", catalog = "mkyong", uniqueConstraints = {

@UniqueConstraint(columnNames = "STOCK\_NAME"),

@UniqueConstraint(columnNames = "STOCK\_CODE") })

public class Stock implements java.io.Serializable {

private Integer stockId;

private String stockCode;

private String stockName;

public Stock() {

}

public Stock(String stockCode, String stockName) {

this.stockCode = stockCode;

this.stockName = stockName;

}

@Id

@GeneratedValue(strategy = IDENTITY)

@Column(name = "STOCK\_ID", unique = true, nullable = false)

public Integer getStockId() {

return this.stockId;

}

public void setStockId(Integer stockId) {

this.stockId = stockId;

}

@Column(name = "STOCK\_CODE", unique = true, nullable = false, length = 10)

public String getStockCode() {

return this.stockCode;

}

public void setStockCode(String stockCode) {

this.stockCode = stockCode;

}

@Column(name = "STOCK\_NAME", unique = true, nullable = false, length = 20)

public String getStockName() {

return this.stockName;

}

public void setStockName(String stockName) {

this.stockName = stockName;

}

@Override

public String toString() {

return "Stock [stockCode=" + stockCode + ", stockId=" + stockId

+ ", stockName=" + stockName + "]";

}

}

**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);

}

Make this class as a bean “stockBo” in Spring Ioc container, and autowire the stock dao class.

package com.mkyong.stock.bo.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.mkyong.stock.bo.StockBo;

import com.mkyong.stock.dao.StockDao;

import com.mkyong.stock.model.Stock;

@Service("stockBo")

public class StockBoImpl implements StockBo{

@Autowired

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. In last tutorial, you DAO classes are directly extends the “**HibernateDaoSupport**“, but it’s not possible to do it in annotation mode, because you have no way to auto wire the session Factory bean from your DAO class. The workaround is create a custom class (**CustomHibernateDaoSupport**) and extends the “**HibernateDaoSupport**” and auto wire the session factory, and your DAO classes extends this class.

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.util;

import org.hibernate.SessionFactory;

import org.springframework.beans.factory.annotation.Autowired;

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

public abstract class CustomHibernateDaoSupport extends HibernateDaoSupport

{

@Autowired

public void anyMethodName(SessionFactory sessionFactory)

{

setSessionFactory(sessionFactory);

}

}

package com.mkyong.stock.dao.impl;

import java.util.List;

import org.springframework.stereotype.Repository;

import com.mkyong.stock.dao.StockDao;

import com.mkyong.stock.model.Stock;

import com.mkyong.util.CustomHibernateDaoSupport;

@Repository("stockDao")

public class StockDaoImpl extends CustomHibernateDaoSupport 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.

**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. In annotation you have to use the **AnnotationSessionFactoryBean**, instead of **LocalSessionFactoryBean**, and specify your annotated model classes in ‘**annotatedClasses**‘ property instead of ‘**mappingResources**‘ property.

**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.annotation.AnnotationSessionFactoryBean">

<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="annotatedClasses">

<list>

<value>com.mkyong.stock.model.Stock</value>

</list>

</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**  
Import the Spring database configuration and enable the Spring’s auto scan feature.

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd">

<!-- Database Configuration -->

<import resource="../database/DataSource.xml"/>

<import resource="../database/Hibernate.xml"/>

<!-- Auto scan the components -->

<context:component-scan

base-package="com.mkyong.stock" />

</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

**Conclusion**

All Spring, Hibernate related classes and configuration files are annotated, it just left the database details in XML file. Should you know how to annotate the database configuration details, please let me know. Personally, i do not use annotation feature much, because somehow you may need some workaround for certain situation, like ‘CustomHibernateDaoSupport’ extends ‘HibernateDaoSupport’ above. The mature developed XML file in Spring and Hibernate. is more preferably.