About Rahul Mondal

MyBatis 3 – Spring integration tutorial

by Rahul Mondal on February 15th, 2012 | **Filed in:** Enterprise Java **Tags:** MyBatis, Spring

As a first step of this tutorial, Spring MVC 3 CRUD example with MyBatis 3, we will define a MyBatis service that will help us to perform CRUD operation on database.

We have a domain class for User and a database table to store the User information on database. We will use xml configuration model for our example to define SQL commands that will perform CRUD operation.

Our Domain class

```
01 package com.raistudies.domain;
02
03 import java.io.Serializable;
04
05 public class User implements Serializable{
06
07     private static final long serialVersionUID = 3647233284813657927L;
08
09     private String id;
10     private String name = null;
11     private String standard = null;
12     private String age;
13     private String sex = null;
14
15     //setter and getter have been omitted to make the code short
16
17     @Override
18     public String toString() {
19         return "User [name=" + name + ", standard=" + standard + ", age=" + age
20         + ", sex=" + sex + "]";
21     }
22 }
```

We have five properties in our domain class called User for which have to provide database services.

Our Database Table

Following is our database table:

```
1  CREATE TABLE `user` (
2      `id` varchar(36) NOT NULL,
3      `name` varchar(45) DEFAULT NULL,
4      `standard` varchar(45) DEFAULT NULL,
5      `age` varchar(45) DEFAULT NULL,
6      `sex` varchar(45) DEFAULT NULL,
7      PRIMARY KEY (`id`)
8  ) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Creating interface for CRUD operations

For defining the CRUD database operation using MyBatis 3, we have to specify the methods that will be used to perform CRUD operation. Following is the interface for our example:

```
01 package com.raistudies.persistence;
02
03 import java.util.List;
04
05 import com.raistudies.domain.User;
06
07 public interface UserService {
08
09     public void saveUser(User user);
10     public void updateUser(User user);
11     public void deleteUser(String id);
12     public List<User> getAllUser();
13 }
```

We have four methods here to perform operations create,update , delete and get from database.

XML Mapping file for UserService interface

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


Newsletter

20,709 insiders are already enjoying weekly updates and complimentary whitepapers! **Join them now** to gain exclusive access to the latest news in the Java world, as well as insights about Android, Scala, Groovy and other related technologies.
As an **extra bonus**, by joining you will get our **brand new e-books**, published by Java Code Geeks and their JCG partners for your reading pleasure!

Join Us

With **748,895** Jun unique visitors and over **500** authors we are placed among the top Java related sites around. Constantly being on the lookout for partners; we encourage you to join us. So if you have a blog with unique and interesting content then you should check out our **JCG** partners program. You can also be a **guest writer** for Java Code Geeks and hone your writing skills in addition to utilizing our **revenue sharing model** to **monetize** your technical writing!

Carrer Opportunities

Technical Solutions Consultant, Social and Knowledge (FULL-TIME) July 20th, 2014

Technical Lead/Manager/Director Engineer- Core Systems (FULL-TIME) July 19th, 2014

```
<mapper namespace="com.raistudies.persistence.UserService">

    <resultMap id="result" type="user">
        <result property="id" column="id"/>
        <result property="name" column="name"/>
        <result property="standard" column="standard"/>
        <result property="age" column="age"/>
        <result property="sex" column="sex"/>
    </resultMap>

    <select id="getAllUser" parameterType="int" resultMap="result">
        SELECT id,name,standard,age,sex
        FROM user;
    </select>

    <insert id="saveUser" parameterType="user">
        INSERT INTO user (id,name,standard,age,sex)
        VALUE (#{id},#{name},#{standard},#{age},#{sex})
    </insert>

    <update id="updateUser" parameterType="user">
        UPDATE user
        SET
        name = #{name},
        standard = #{standard},
        age = #{age},
        sex = #{sex}
        where id = #{id}
    </update>

    <delete id="deleteUser" parameterType="int">
        DELETE FROM user
        WHERE id = #{id}
    </delete>
</mapper>
```

You will see a lot of things new here:

The mapping file will contain element **<mapper/>** to define the SQL statement for the services. Here the property "**namespace**" defines the interface for which this mapping file has been defined.

<insert/> tag defines that the operation is of type insert. The value of "**id**" property specifies the function name for which the SQL statement is been defined. Here it is "**saveUser**". The property "**parameterType**" defines the parameter of the method is of which type. We have used alias for the class User here. The alias will be configured in MyBatis Configuration file later. Then, we have to define the SQL Statement. **#{id}** defines that the property "**id**" of class User will be passed as a parameter to the SQL query.

<resultMap/> tag is used to specify the mapping between the User class and user table. **id** of <resultMap/> is a unique name to the mapping definition. Under this tag, we define the different properties and which column is bounded to which property.

<select/> tag is used to specify a select SQL statement. The value of "**id**" property specifies the function name for which the SQL statement is been defined.

The attribute **resultMap** is used to define the return type of the SQL statement as a collection.

MyBatis 3 Configuration file

Following is our configuration file for MyBatis:

```
<?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>
    <settings>
        <!-- changes from the defaults -->
       <setting name="lazyLoadingEnabled" value="false" />
    </settings>
    <typeAliases>
        <typeAlias type="com.raistudies.domain.User" alias="user"/>
    </typeAliases>
</configuration>
```

You can see, we have not defined some very important properties here:

1. Database connection properties.
2. Transaction related properties.
3. And also have not defined mappers configuration.

MyBatis 3 is very powerful SQL mapping framework with automatic database access class generation using a proxy implementation of the services defined by users. We get realize it's true power if you integrate MyBatis 3 with Spring framework and use these proxy implementation. It will reduce our database work by 80%. Below, we will see how to integrate MyBatis 3 with Spring 3 framework. Previously, we created the CRUD database service for class User using MyBatis 3. Now, we will integrate the data services implemented using MyBatis with Spring framework.

Tools Used:

- **c3p0-0.9.1.2.jar** – For providing pooled database connection.
- **mybatis-spring-1.0.0.jar** – For integrating MyBatis with Spring (Provided by MyBatis team)
- **Spring JDBC and Core library**

To integrate these two frameworks, we have to follow bellow steps:

Step 1: Defining datasource as a Spring bean
As we will use c3po data source provider, we have to define datasource bean in Spring. Following is the configuration snippet:

```
<!-- Declare a datasource that has pooling capabilities -->
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
destroy-method="close" p:driverClass="${app.jdbc.driverClassName}"
p:jdbcUrl="${app.jdbc.url}" p:user="${app.jdbc.username}" p:password="${app.jdbc.password}"
p:acquireIncrement="10" p:idleConnectionTestPeriod="60" p:maxPoolSize="100"
p:maxStatements="50" p:minPoolSize="10" />
```

Here we have created a spring bean with id **dataSource** of class **com.mchange.v2.c3p0.ComboPooledDataSource** which is provided by c3p0 library for pooled data source.

We have set some properties in the bean. Following is the description of properties defined in bean:

- **driverClass** : Driver class that will be used to connect to database.
- **jdbcUrl** : jdbc url defining the database connection string.
- **user** : username of the database user.
- **password** : password of the database user.
- **acquireIncrement** : how many connections will be created at a time when there will be a shortage of connections.
- **idleConnectionTestPeriod** : after how much delay a connection will be closed if it is no longer in use.
- **maxPoolSize** : Max number of connections that can be created.
- **maxStatements** : Max number of SQL statements to be executed on a connection.
- **minPoolSize** : Minimum number of connections to be created.

We have used Spring EL to define many of the property values, that will be bring from properties file.

Defining Transaction Manager in Spring

We will user Transaction Manager provided by Spring JDBC framework and for defining transaction levels we will use annotations. Following is the configuration for transaction manager:

```
<!-- Declare a transaction manager -->
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
p:dataSource-ref="dataSource" />

<!-- Enable annotation style of managing transactions -->
<tx:annotation-driven transaction-manager="transactionManager" />
```

Defining MyBatis SqlSessionFactory and MapperScanner

```
<!-- define the SqlSessionFactory, notice that configLocation is not needed when you use
MapperFactoryBean -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation" value="WEB-INF/mybatis/sqlmap-config.xml" />
</bean>

<!-- scan for mappers and let them be autowired -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="${MapperInterfacePackage}" />
</bean>
```

The **SqlSessionFactory** bean will provide SessionFactory instances of MyBatis. To configure SqlSessionFactory, we need to define two properties. First the data source which will be used by MyBatis to create connection database and MyBatis configuration file name to configure the environment of MyBatis.

MapperScannerConfigurer is used to publish the data service interfaces in defined for MyBatis to configure as Spring Beans. We just have to provide package in which the interfaces and their mapping XML files has been defined. We can specify more than one packages using common separation or semicolon. After that we will be able to get the instances of UserService using @Autowired annotation. We do not have to implement the interface as MyBatis will provide proxy implementation for this.

Spring configuration file as together
Here is our **jdbc-context.xml** as together:

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

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


http://www.springframework.org/schema/beans/spring-beans-3.0.xsd


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


http://www.springframework.org/schema/tx/spring-tx-3.0.xsd


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


```

```
http://www.springframework.org/schema/context/spring-context-3.0.xsd

">

<context:property-placeholder location="/WEB-INF/jdbc.properties,/WEB-INF/mybatis
/mybatis.properties" />

    <!-- Enable annotation style of managing transactions -->
    <tx:annotation-driven transaction-manager="transactionManager" />

    <!-- Declare a datasource that has pooling capabilities -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
    destroy-method="close" p:driverClass="${app.jdbc.driverClassName}"
    p:jdbcUrl="${app.jdbc.url}" p:user="${app.jdbc.username}" p:password="${app.jdbc.password}"
    p:acquireIncrement="10" p:idleConnectionTestPeriod="60" p:maxPoolSize="100"
    p:maxStatements="50" p:minPoolSize="10" />

    <!-- Declare a transaction manager -->
    <bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
    p:dataSource-ref="dataSource" />

    <!-- define the SqlSessionFactory, notice that configLocation is not needed when you use
MapperFactoryBean -->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="configLocation" value="WEB-INF/mybatis/sqlmap-config.xml" />
    </bean>

    <!-- scan for mappers and let them be autowired -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="${MapperInterfacePackage}" />
    </bean>

</beans>
```

jdbc.properties file

```
# database properties
app.jdbc.driverClassName=com.mysql.jdbc.Driver
app.jdbc.url=jdbc:mysql://localhost/mybatis-example
app.jdbc.username=root
app.jdbc.password=password
```

mybatis.properties file

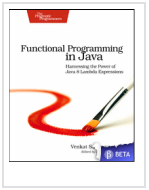
```
MapperInterfacePackage=com.raistudies.persistence
```

Reference: Creating CRUD service using MyBatis 3 Mapping Framew ork – Part 1 & Integrating MyBatis 3 and Spring framew orks – Part 2 from our JCG partner Rahul Mondal at the Rai Studies blog.

You might also like:

- **MyBatis Tutorial – CRUD Operations and Mapping Relationships – Part 2**
- **Spring MVC 3 Controller for MyBatis CRUD operation**
- **Spring Property Placeholder Configurer – A few not so obvious options**

Related Whitepaper:



Functional Programming in Java: Harnessing the Power of Java 8 Lambda Expressions

Get ready to program in a whole new way!

Functional Programming in Java will help you quickly get on top of the new, essential Java 8 language features and the functional style that will change and improve your code. This short, targeted book will help you make the paradigm shift from the old imperative way to a less error-prone, more elegant, and concise coding style that's also a breeze to parallelize. You'll explore the syntax and semantics of lambda expressions, method and constructor references, and functional interfaces. You'll design and write applications better using the new standards in Java 8 and the JDK.

One Response to "MyBatis 3 – Spring integration tutorial"

VietNam

June 11th, 2014 at 1:14 pm

Help me!
You have source code “MyBatis 3 – Spring integration” build success, share public w ith me.
Please!

Reply

Leave a Reply

Name (Required)

Mail (w ill not be published) (Required)

Website

9 − five =

Notify me of follow up comments via e-mail

Sign me up for the new sletter!

Submit Comment