About Siva Reddy

Katamreddy Siva Prasad is a Senior Software Engineer working in E-Commerce domain. His areas of interest include Object Oriented Design, SOLID Design principles, RESTful WebServices and OpenSource softwares including Spring, MyBatis and Jenkins.

MyBatis Tutorial – CRUD Operations and Mapping Relationships – Part 1

by Siva Reddy on November 16th, 2012 | **Filed in:** Enterprise Java **Tags:** MyBatis

CRUD Operations

MyBatis is an SQL Mapper tool which greatly simplifies the database programing when compared to using JDBC directly.

Step1: Create a Maven project and configure MyBatis dependencies.

```
<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/xsd/maven-4.0.0.xsd'>

 <modelVersion>4.0.0</modelVersion>

 <groupId>com.sivalabs</groupId>
 <artifactId>mybatis-demo</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>jar</packaging>

 <name>mybatis-demo</name>
 <url>http://maven.apache.org</url>

 <properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 </properties>

 <build>
  <plugins>
   <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>2.3.2</version>
    <configuration>
     <source>1.6</source>
     <target>1.6</target>
     <encoding>${project.build.sourceEncoding}</encoding>
    </configuration>
   </plugin>
  </plugins>
 </build>

 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.10</version>
   <scope>test</scope>
  </dependency>

  <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis</artifactId>
      <version>3.1.1</version>
  </dependency>
  <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.21</version>
            <scope>runtime</scope>
```

```
        </dependency>
 </dependencies>
</project>
```

Step#2: Create the table USER and a Java domain Object User as follows:

```
CREATE TABLE  user (
  user_id int(10) unsigned NOT NULL auto_increment,
  email_id varchar(45) NOT NULL,
  password varchar(45) NOT NULL,
  first_name varchar(45) NOT NULL,
  last_name varchar(45) default NULL,
  PRIMARY KEY  (user_id),
  UNIQUE KEY Index_2_email_uniq (email_id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
package com.sivalabs.mybatisdemo.domain;
public class User
{
 private Integer userId;
 private String emailId;
 private String password;
 private String firstName;
 private String lastName;

 @Override
 public String toString() {
  return 'User [userId=' + userId + ', emailId=' + emailId
    + ', password=' + password + ', firstName=' + firstName
    + ', lastName=' + lastName + ']';
 }
 //setters and getters
}
```

Step#3: Create MyBatis configuration files.

a) Create jdbc.properties file in src/main/resources folder

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/mybatis-demo
jdbc.username=root
jdbc.password=admin
```

b) Create mybatis-config.xml file in src/main/resources folder

```
<?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>
 <properties resource='jdbc.properties'/>
 <typeAliases>
  <typeAlias type='com.sivalabs.mybatisdemo.domain.User' alias='User'></typeAlias>
 </typeAliases>
 <environments default='development'>
  <environment id='development'>
    <transactionManager type='JDBC'/>
    <dataSource type='POOLED'>
   <property name='driver' value='${jdbc.driverClassName}'/>
   <property name='url' value='${jdbc.url}'/>
   <property name='username' value='${jdbc.username}'/>
   <property name='password' value='${jdbc.password}'/>
    </dataSource>
  </environment>
  </environments>
  <mappers>
 <mapper resource='com/sivalabs/mybatisdemo/mappers/UserMapper.xml'/>
  </mappers>
</configuration>
```

Step#4: Create an interface UserMapper.java in src/main/java folder in com.sivalabs.mybatisdemo.mappers package.

```
package com.sivalabs.mybatisdemo.mappers;

import java.util.List;
import com.sivalabs.mybatisdemo.domain.User;

public interface UserMapper
{

 public void insertUser(User user);

 public User getUserById(Integer userId);

 public List<User> getAllUsers();

 public void updateUser(User user);

 public void deleteUser(Integer userId);

}
```

Step#5: Create UserMapper.xml file in src/main/resources folder in com.sivalabs.mybatisdemo.mappers package.

```
<?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='com.sivalabs.mybatisdemo.mappers.UserMapper'>

  <select id='getUserById' parameterType='int' resultType='com.sivalabs.mybatisdemo.domain.User'>
     SELECT
      user_id as userId,
      email_id as emailId ,
      password,
      first_name as firstName,
      last_name as lastName
     FROM USER
     WHERE USER_ID = #{userId}
  </select>
  <!-- Instead of referencing Fully Qualified Class Names we can register Aliases in mybatis-
config.xml and use Alias names. -->
   <resultMap type='User' id='UserResult'>
    <id property='userId' column='user_id'/>
    <result property='emailId' column='email_id'/>
    <result property='password' column='password'/>
    <result property='firstName' column='first_name'/>
    <result property='lastName' column='last_name'/>
   </resultMap>

  <select id='getAllUsers' resultMap='UserResult'>
   SELECT * FROM USER
  </select>

  <insert id='insertUser' parameterType='User' useGeneratedKeys='true' keyProperty='userId'>
   INSERT INTO USER(email_id, password, first_name, last_name)
    VALUES(#{emailId}, #{password}, #{firstName}, #{lastName})
  </insert>

  <update id='updateUser' parameterType='User'>
    UPDATE USER
    SET
     PASSWORD= #{password},
     FIRST_NAME = #{firstName},
     LAST_NAME = #{lastName}
    WHERE USER_ID = #{userId}
  </update>

  <delete id='deleteUser' parameterType='int'>
    DELETE FROM USER WHERE USER_ID = #{userId}
  </delete>

</mapper>
```

Step#6: Create MyBatisUtil.java to instantiate SqlSessionFactory.

```
package com.sivalabs.mybatisdemo.service;

import java.io.IOException;
import java.io.Reader;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

public class MyBatisUtil
{
 private static SqlSessionFactory factory;

 private MyBatisUtil() {
 }

 static
 {
  Reader reader = null;
  try {
   reader = Resources.getResourceAsReader('mybatis-config.xml');
  } catch (IOException e) {
   throw new RuntimeException(e.getMessage());
  }
  factory = new SqlSessionFactoryBuilder().build(reader);
 }

 public static SqlSessionFactory getSqlSessionFactory()
 {
  return factory;
 }
}
```

Step#7: Create UserService.java in src/main/java folder.

```
package com.sivalabs.mybatisdemo.service;

import java.util.List;
import org.apache.ibatis.session.SqlSession;
import com.sivalabs.mybatisdemo.domain.User;
import com.sivalabs.mybatisdemo.mappers.UserMapper;

public class UserService
{

 public void insertUser(User user) {
  SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();
  try{
  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
  userMapper.insertUser(user);
```

```
  sqlSession.commit();
  }finally{
   sqlSession.close();
  }
 }

 public User getUserById(Integer userId) {
  SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();
  try{
  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
  return userMapper.getUserById(userId);
  }finally{
   sqlSession.close();
  }
 }

 public List<User> getAllUsers() {
  SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();
  try{
  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
  return userMapper.getAllUsers();
  }finally{
   sqlSession.close();
  }
 }

 public void updateUser(User user) {
  SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();
  try{
  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
  userMapper.updateUser(user);
  sqlSession.commit();
  }finally{
   sqlSession.close();
  }

 }

 public void deleteUser(Integer userId) {
  SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();
  try{
  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
  userMapper.deleteUser(userId);
  sqlSession.commit();
  }finally{
   sqlSession.close();
  }

 }

}
```

Step#8: Create a JUnit Test class to test UserService methods.

```
package com.sivalabs.mybatisdemo;

import java.util.List;

import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.BeforeClass;
import org.junit.Test;

import com.sivalabs.mybatisdemo.domain.User;
import com.sivalabs.mybatisdemo.service.UserService;

public class UserServiceTest
{
 private static UserService userService;

 @BeforeClass
    public static void setup()
 {
  userService = new UserService();
 }

 @AfterClass
    public static void teardown()
 {
  userService = null;
 }

    @Test
 public void testGetUserById()
 {
  User user = userService.getUserById(1);
  Assert.assertNotNull(user);
  System.out.println(user);
 }

    @Test
    public void testGetAllUsers()
    {
  List<User> users = userService.getAllUsers();
  Assert.assertNotNull(users);
  for (User user : users)
  {
   System.out.println(user);
  }

 }

    @Test
    public void testInsertUser()
    {
```

```
52      User user = new User();
53      user.setEmailId('test_email_'+System.currentTimeMillis()+'@gmail.com');
54      user.setPassword('secret');
55      user.setFirstName('TestFirstName');
56      user.setLastName('TestLastName');
57
58      userService.insertUser(user);
59   Assert.assertTrue(user.getUserId() != 0);
60   User createdUser = userService.getUserById(user.getUserId());
61   Assert.assertNotNull(createdUser);
62   Assert.assertEquals(user.getEmailId(), createdUser.getEmailId());
63   Assert.assertEquals(user.getPassword(), createdUser.getPassword());
64   Assert.assertEquals(user.getFirstName(), createdUser.getFirstName());
65   Assert.assertEquals(user.getLastName(), createdUser.getLastName());
66
67  }
68
69     @Test
70     public void testUpdateUser()
71     {
72      long timestamp = System.currentTimeMillis();
73   User user = userService.getUserById(2);
74   user.setFirstName('TestFirstName'+timestamp);
75      user.setLastName('TestLastName'+timestamp);
76      userService.updateUser(user);
77   User updatedUser = userService.getUserById(2);
78   Assert.assertEquals(user.getFirstName(), updatedUser.getFirstName());
79   Assert.assertEquals(user.getLastName(), updatedUser.getLastName());
80  }
81
82    @Test
83    public void testDeleteUser()
84    {
85      User user = userService.getUserById(4);
86      userService.deleteUser(user.getUserId());
87   User deletedUser = userService.getUserById(4);
88   Assert.assertNull(deletedUser);
89
90  }
91 }
```

Now, I will explain how to perform CRUD operations using MyBatis Annotation support without need of Queries configuration in XML mapper files.

Step#1: Create a table BLOG and a java domain Object Blog.

```
1 CREATE TABLE  blog (
2   blog_id int(10) unsigned NOT NULL auto_increment,
3   blog_name varchar(45) NOT NULL,
4   created_on datetime NOT NULL,
5   PRIMARY KEY  (blog_id)
6 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
01 package com.sivalabs.mybatisdemo.domain;
02
03 import java.util.Date;
04
05 public class Blog {
06
07  private Integer blogId;
08  private String blogName;
09  private Date createdOn;
10
11  @Override
12  public String toString() {
13   return 'Blog [blogId=' + blogId + ', blogName=' + blogName
14     + ', createdOn=' + createdOn + ']';
15  }
16  //Seeters and getters
17 }
```

Step#2: Create UserMapper.java interface with SQL queries in Annotations.

```
01 package com.sivalabs.mybatisdemo.mappers;
02
03 import java.util.List;
04
05 import org.apache.ibatis.annotations.Delete;
06 import org.apache.ibatis.annotations.Insert;
07 import org.apache.ibatis.annotations.Options;
08 import org.apache.ibatis.annotations.Result;
09 import org.apache.ibatis.annotations.Results;
10 import org.apache.ibatis.annotations.Select;
11 import org.apache.ibatis.annotations.Update;
12
13 import com.sivalabs.mybatisdemo.domain.Blog;
14
15 public interface BlogMapper
16 {
17  @Insert('INSERT INTO BLOG(BLOG_NAME, CREATED_ON) VALUES(#{blogName}, #{createdOn})')
18  @Options(useGeneratedKeys=true, keyProperty='blogId')
19  public void insertBlog(Blog blog);
20
21  @Select('SELECT BLOG_ID AS blogId, BLOG_NAME as blogName, CREATED_ON as createdOn FROM BLOG WHERE
BLOG_ID=#{blogId}')
22  public Blog getBlogById(Integer blogId);
23
24  @Select('SELECT * FROM BLOG ')
25  @Results({
26   @Result(id=true, property='blogId', column='BLOG_ID'),
```

```
  @Result(property='blogName', column='BLOG_NAME'),
  @Result(property='createdOn', column='CREATED_ON')
 })
 public List<Blog> getAllBlogs();

 @Update('UPDATE BLOG SET BLOG_NAME=#{blogName}, CREATED_ON=#{createdOn} WHERE BLOG_ID=#{blogId}')
 public void updateBlog(Blog blog);

 @Delete('DELETE FROM BLOG WHERE BLOG_ID=#{blogId}')
 public void deleteBlog(Integer blogId);

}
```

Step#3: Configure BlogMapper in mybatis-config.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>
 <properties resource='jdbc.properties'/>
 <environments default='development'>
  <environment id='development'>
    <transactionManager type='JDBC'/>
    <dataSource type='POOLED'>
   <!-- <property name='driver' value='com.mysql.jdbc.Driver'/>
   <property name='url' value='jdbc:mysql://localhost:3306/mybatis-demo'/>
   <property name='username' value='root'/>
   <property name='password' value='admin'/> -->
   <property name='driver' value='${jdbc.driverClassName}'/>
   <property name='url' value='${jdbc.url}'/>
   <property name='username' value='${jdbc.username}'/>
   <property name='password' value='${jdbc.password}'/>
    </dataSource>
  </environment>
  </environments>
  <mappers>
    <mapper class='com.sivalabs.mybatisdemo.mappers.BlogMapper'/>
  </mappers>
</configuration>
```

Step#4: Create BlogService.java

```
package com.sivalabs.mybatisdemo.service;

import java.util.List;

import org.apache.ibatis.session.SqlSession;

import com.sivalabs.mybatisdemo.domain.Blog;
import com.sivalabs.mybatisdemo.mappers.BlogMapper;

public class BlogService
{

 public void insertBlog(Blog blog) {
  SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();
  try{
  BlogMapper blogMapper = sqlSession.getMapper(BlogMapper.class);
  blogMapper.insertBlog(blog);
  sqlSession.commit();
  }finally{
   sqlSession.close();
  }
 }

 public Blog getBlogById(Integer blogId) {
  SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();
  try{
  BlogMapper blogMapper = sqlSession.getMapper(BlogMapper.class);
  return blogMapper.getBlogById(blogId);
  }finally{
   sqlSession.close();
  }
 }

 public List<Blog> getAllBlogs() {
  SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();
  try{
  BlogMapper blogMapper = sqlSession.getMapper(BlogMapper.class);
  return blogMapper.getAllBlogs();
  }finally{
   sqlSession.close();
  }
 }

 public void updateBlog(Blog blog) {
  SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();
  try{
  BlogMapper blogMapper = sqlSession.getMapper(BlogMapper.class);
  blogMapper.updateBlog(blog);
  sqlSession.commit();
  }finally{
   sqlSession.close();
  }
 }

 public void deleteBlog(Integer blogId) {
  SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();
  try{
  BlogMapper blogMapper = sqlSession.getMapper(BlogMapper.class);
```

```
  blogMapper.deleteBlog(blogId);
  sqlSession.commit();
  }finally{
   sqlSession.close();
  }

 }

}
```

Step#5: Create JUnit Test for BlogService methods

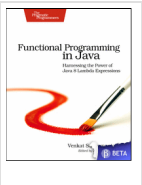
```
package com.sivalabs.mybatisdemo;

import java.util.Date;
import java.util.List;

import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.BeforeClass;
import org.junit.Test;

import com.sivalabs.mybatisdemo.domain.Blog;
import com.sivalabs.mybatisdemo.service.BlogService;

public class BlogServiceTest
{
 private static BlogService blogService;

 @BeforeClass
    public static void setup()
 {
  blogService = new BlogService();
 }

 @AfterClass
    public static void teardown()
 {
  blogService = null;
 }

    @Test
 public void testGetBlogById()
 {
  Blog blog = blogService.getBlogById(1);
  Assert.assertNotNull(blog);
  System.out.println(blog);
 }

    @Test
    public void testGetAllBlogs()
    {
  List<Blog> blogs = blogService.getAllBlogs();
  Assert.assertNotNull(blogs);
  for (Blog blog : blogs)
  {
   System.out.println(blog);
  }

 }

    @Test
    public void testInsertBlog()
    {
     Blog blog = new Blog();
     blog.setBlogName('test_blog_'+System.currentTimeMillis());
     blog.setCreatedOn(new Date());

     blogService.insertBlog(blog);
  Assert.assertTrue(blog.getBlogId() != 0);
  Blog createdBlog = blogService.getBlogById(blog.getBlogId());
  Assert.assertNotNull(createdBlog);
  Assert.assertEquals(blog.getBlogName(), createdBlog.getBlogName());

 }

    @Test
    public void testUpdateBlog()
    {
     long timestamp = System.currentTimeMillis();
  Blog blog = blogService.getBlogById(2);
  blog.setBlogName('TestBlogName'+timestamp);
     blogService.updateBlog(blog);
  Blog updatedBlog = blogService.getBlogById(2);
  Assert.assertEquals(blog.getBlogName(), updatedBlog.getBlogName());
 }

   @Test
   public void testDeleteBlog()
   {
     Blog blog = blogService.getBlogById(4);
     blogService.deleteBlog(blog.getBlogId());
  Blog deletedBlog = blogService.getBlogById(4);
  Assert.assertNull(deletedBlog);
 }
}
```

Reference: MyBatis Tutorial: Part1 – CRUD Operations from our JCG partner, MyBatis Tutorial: Part-2: CRUD operations Using Annotations from our JCG partner Siva Reddy at the My Experiments on Technology blog.

You might also like:

- MyBatis Tutorial – CRUD Operations and Mapping Relationships – Part 2
- Spring MVC 3 Controller for MyBatis CRUD operation
- MyBatis 3 – Spring integration tutorial

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.

11 Responses to "MyBatis Tutorial – CRUD Operations and Mapping Relationships – Part 1"

Yannick Majoros
November 18th, 2012 at 3:05 pm

Come on, I hope nobody uses that instead of JPA?

Reply

Siva Prasad Reddy
November 19th, 2012 at 8:58 am

Try building a RESTFul services application which contain very complex object graph structure using JPA. If you try to marshall JPA loaded proxy after it got disconnected from Session it will throw LazyLoadingExcleption, if you try to marshall JPA entity by attaching it to JPA session it will load the entire database, if you are planning to use DTO pattern and populate the necessary properties only All the BEST :-) And then take a look at MyBatis :-)

Reply

Yannick Majoros
December 1st, 2012 at 10:41 pm

How would MyBatis solve that, besides the fact that it's non-standard and as such, the only implementation? If you need to load a complex object graph, you'll have to maintain it somehow. Perhaps your should not make your object graph that complex in the first place (seems it's what you do here anyway, small object graph for which jpa is a much better choice). Perhaps you should better find the bounds of your business operations, and know when to merge and when not to end your transaction or to detach. This thing is a relic, should disappear asap instead of letting people write crap that will eventually have to be refactored to jpa or be forever lost in spaghetti-code land.

Reply

Siva Prasad Reddy
February 4th, 2013 at 11:51 am

Hi,
I am not going to enter into never ending debate on whether STANDARDS are important

or not.
Here w hat I am trying to say is if i load a complex object structure using Mybatis they are really POJOs, not proxies. So I can use any of the marshalling/unmarshalling tools like Jackson/xstream etc to generate XML/JSON. If some property is null it w ill ignore to generate the tag or generates an empty tag.
Where as if i load a complex object by JPA w hich has lazy child collection then if i try to marshall that object it w ill throw LazyLoadingException if the object is detached from EntityManager. If that object is still attached to JPA entityManager it w ill load the w hole database as the marshalling libraries navigate through all the properties triggering loading the data from DB.
Anyw ay it w ould be great if you can show an example on how to marshall a JPA loaded object w hich has a lazy child collection property using Jackson w ithout using manual copy-object (DTO) creation.

Reply

Phongveth

January 23rd, 2013 at 8:00 am

Hi Siva, I still confuse that can w e define method in the interface w hich have return type boolean example like public void insertBlog(Blog blog); into
public boolean insertBlog(Blog blog);

Reply

Siva Prasad Reddy

February 4th, 2013 at 11:31 am

No, for insert/update/delete queries you can return int representing the no of row s affected by the query. You can't return boolean.

Reply

Phongveth Luangsisongkham

February 5th, 2013 at 3:03 am

Thank you very much for your reply

Reply

krish

February 7th, 2013 at 9:35 am

hello friend you have done very good w ork … i used it in my app…. thx
but i have a prob. w hen i open it from differ pc then it not show me onl9 to others …
and w hen i open it in my ow n pc w ith differ brow ser then it w orks good … please help me

Reply

Jed

February 27th, 2014 at 12:47 pm

Hi Siva Reddy,
Thanks for your tutorial, I learned a lot.
I do have some points to address, I hope you dont mind sharing it.
On your Service objects(BlogService.java and UserService.java), i think these object are for actual DB execution. My concern is, how w ould you handle the Exceptions throw n by your JDBC driver?
as far as I can see, try{} – finally{} are the only ones there. can you explain how to determine all the exceptions throw n by the driver? and is it possible to handle those? please provide a sample try{} – catch{} combination.
Please reply thru my mail.
Thank you and Best Regards.

Reply

Siva

February 27th, 2014 at 2:41 pm

Hi Jed,

I haven't covered Exception handling in Service classes in the article. Yes, if you are using plain MyBatis you need to handle JDBC Exceptions using try-cach blocks. If you are using MyBatis w ith Spring then Spring w ill translate JDBC Exceptions into Spring's more appropriate DataAccessExceptions hierarchy and they are all RuntimeExceptions. If you w ant to handle case-by-case then you can catch those Exceptions and take necessary action.
Hope it helps.

Reply

deepak

June 27th, 2014 at 5:33 pm

Hi Siva,
Thanks for this w onderful tutorial. I w as trying a POC w ith ibatis 2 version and got stuck w ith some strange errors. Then w anted to sw itch over to ibatis. With your tutorial, i just completed it w ith half an hour. Thanks :) very useful

Reply

Leave a Reply

Name (Required)

Mail (w ill not be published) (Required)

Website

6 + = ten

Notify me of follow up comments via e-mail

Sign me up for the new sletter!

Submit Comment