**MyBatis01基础+使用Dao**

**前置：**

**Jar准备 , 日志配置文件，数据库连接配置文件，**

**Jar: （1-3 ：sql连接 4-7：日志记录 8：mybatis）**

1. **mysql-connector-java-5.1.7-bin.jar**
2. **c3p0-0.9.5.2.jar**
3. **mchange-commons-java-0.2.12.jar**
4. **commons-logging-1.1.1.jar**
5. **log4j-1.2.17.jar**
6. **log4j-api-2.0-rc1.jar**
7. **log4j-core-2.0-rc1.jar**
8. **mybatis-3.5.4.jar**

**log4j.properties:**

1. log4j.rootLogger=CONSOLE,stdout,logfile
2. log4j.appender.stdout=org.apache.log4j.ConsoleAppender
3. log4j.appender.stdout.Target=System.out
4. log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
5. log4j.appender.stdoCut.layout.ConversionPattern=%d %p [%c]:%L - %m%n
6. log4j.appender.logfile=org.apache.log4j.RollingFileAppender
7. log4j.appender.logfile.File=D:/JavaLogFilekey/service.log
8. log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
9. log4j.appender.logfile.layout.ConversionPattern = %d{yyyy-MM-dd HH:mm:ss} [ %p ] - [ %l ] %m%n

**C3p0.propeties:**

1. c3p0.driverClass = com.mysql.jdbc.Driver
2. c3p0.jdbcUrl = jdbc:mysql://localhost:3306/db1?useSSL=false&serverTimezone=UTC
3. c3p0.user = root
4. c3p0.password = 123456
5. c3p0.initialPoolSize = 5
6. c3p0.maxPoolSize = 10
7. c3p0.checkoutTimeout = 3000

**构建model类**

**Model.xml：**

1. <!DOCTYPE mapper    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
2. "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

5. <mapper namespace="xxx">
6. <select id="selectById" parameterType="int" resultType="cn.cxd.models.User">
7. select \* from user where id = #{password};
8. </select>
10. <select id="selectByName" parameterType="String" resultType="cn.cxd.models.User">
11. select \* from user where username like '%${avdc}%';
12. </select>
14. <select id="selectByUserId" parameterType="cn.cxd.models.User" resultType="cn.cxd.models.User">
15. select \* from user where id = #{id};
16. </select>
18. <select id="selectByUserVo" parameterType="cn.cxd.models.UserQueryVo" resultType="cn.cxd.models.User">
19. select \* from user where id = #{user.id};
20. </select>
22. <insert id="insert" parameterType="cn.cxd.models.User">
23. insert into user (id,username,password) values (#{id},#{username},#{password})
24. </insert>

27. <delete id="delete" parameterType="int">
28. delete from user where id = #{xxx};
29. </delete>
31. <update id="update" parameterType="cn.cxd.models.User">
32. update user set username = #{username} , password = #{password} where id = #{id};
33. </update>
35. <update id="update2" parameterType="HashMap">
36. update user set username = #{username} , password = #{password} where id = #{id};
37. </update>
39. </mapper>

**Dao: 依赖modelMap.xml的增删改method标注的id调用**

1. **package** cn.cxd.dao;
3. **import** cn.cxd.impls.IUserDao;
4. **import** cn.cxd.models.User;
5. **import** cn.cxd.models.UserQueryVo;
6. **import** org.apache.ibatis.io.Resources;
7. **import** org.apache.ibatis.session.SqlSession;
8. **import** org.apache.ibatis.session.SqlSessionFactory;
9. **import** org.apache.ibatis.session.SqlSessionFactoryBuilder;
11. **import** java.io.InputStream;
12. **import** java.util.HashMap;
13. **import** java.util.List;
15. **public** **class** UserDaoImpl **implements** IUserDao {
16. **private** SqlSessionFactory sqlSessionFactory;
17. **private** **static** UserDaoImpl instance;

20. **private** UserDaoImpl() {
21. SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();
22. **try**{
23. sqlSessionFactory = builder.build(Resources.getResourceAsStream("sqlMapConfig.xml"));
24. }**catch** (Exception e){
25. e.printStackTrace();
26. }
27. }
29. **public** **static** UserDaoImpl getInstance(){
30. **if**(instance == **null**){
31. **synchronized** (UserDaoImpl.**class**){
32. **if**(instance == **null**){
33. instance = **new** UserDaoImpl();
34. }
35. }
36. }
37. **return** instance;
38. }


42. @Override
43. **public** **int** addUserReturnRow(User user) {
44. SqlSession sqlSession = sqlSessionFactory.openSession(**true**);
45. **int** row = sqlSession.insert("insert", user);
46. sqlSession.close();
47. **return** row;
48. }
49. @Override
50. **public** **int** selectLikeReturnCount(String name) {
51. **return** 0;
52. }
54. @Override
55. **public** User findUserById(**int** id) {
56. SqlSession sqlSession = sqlSessionFactory.openSession();
57. User user = sqlSession.selectOne("selectById", id);
58. sqlSession.close();
59. **return** user;
60. }

63. @Override
64. **public** List<User> findUserByUsernameTag(String username) {
65. SqlSession sqlSession = sqlSessionFactory.openSession();
66. List<User> users =  sqlSession.selectList("selectByName", username);
67. sqlSession.close();
68. **return** users;
69. }
71. @Override
72. **public** User findUserByUserId(User user) {
73. SqlSession sqlSession = sqlSessionFactory.openSession();
74. User target = sqlSession.selectOne("selectByUserId", user);
75. sqlSession.close();
76. **return** target;
77. }
79. @Override
80. **public** **int** delUserReturnRow(**int** id) {
81. SqlSession sqlSession = sqlSessionFactory.openSession(**true**);
82. **int** row = sqlSession.delete("delete", id);
83. sqlSession.close();
84. **return** row;
85. }

88. @Override
89. **public** **int** updateUserInforByMap(HashMap<String, Object> hashMap) {
90. SqlSession sqlSession = sqlSessionFactory.openSession(**true**);
91. **int** row = sqlSession.update("update2", hashMap);
92. sqlSession.close();
93. **return** row;
94. }
96. @Override
97. **public** **int** updateUserInforByUser(User user) {
98. SqlSession sqlSession = sqlSessionFactory.openSession(**true**);
99. **int** row = sqlSession.update("update", user);
100. sqlSession.close();
101. **return** row;
102. }

105. @Override
106. **public** User findUserByUserVo(UserQueryVo userQueryVo) {
107. SqlSession sqlSession = sqlSessionFactory.openSession();
108. User target = sqlSession.selectOne("selectByUserVo", userQueryVo);
109. sqlSession.close();
110. **return** target;
111. }
112. }

**Sqlconfig:加载配置文件和model.xml**

1. <?xml version="1.0" encoding="UTF-8" ?> <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
2. "http://mybatis.org/dtd/mybatis-3-config.dtd">
3. <configuration>
4. <properties resource="c3p0.properties"></properties>
5. <environments **default**="development">
6. <environment id="development">
7. <transactionManager type="JDBC"/>
8. <dataSource type="POOLED">
9. <property name="driver" value="${c3p0.driverClass}"/>
10. <property name="url" value="${c3p0.jdbcUrl}"/>
11. <property name="username" value="${c3p0.user}"/>
12. <property name="password" value="${c3p0.password}"/>
13. </dataSource>
14. </environment>
15. </environments>
17. <mappers>
18. <mapper resource="cn/cxd/models/user.xml"/>
19. </mappers>
20. </configuration>

**向上提供服务**

**Main 如果没做对应的处理，那么就是需要完成一系列步骤**

1. SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();
2. SqlSessionFactory sqlSessionFactory = builder.build(Resources.getResourceAsStream("sqlMapConfig.xml"));
3. SqlSession sqlSession = sqlSessionFactory.openSession(**true**);
4. **int** row = sqlSession.insert("insert", user);
5. sqlSession.close();

**注意：**

1. **modelMap.xml中mapper标签需要nameSpace属性，但可以随意取名，modelMap的标签可以是<select><insert><delete><update>，他们的属性常见有parameterType，id,** **resultType，resultMap，**

**其中id,是交给dao层使用的方法字符串id，**

**paratermeterType是参数类型，只能是一个对象，但是可以是List,Map或是一个包装类，也可以是简单类型**

**resultType是返回值类型，同样是一个对象，但是有的标签没有这个属性，比如insert,delete,因为按照数据库返回的是受影响的行数，因此返回的是int。**

**resultMap是返回值的对应关系，就是说select的属性可以按照映射关系到一个对象上，而不必要求select的属性和对象的属性存在一致。**

**标签体的内容是一个sql语句：**

1. <select id="selectById" parameterType="int" resultType="cn.cxd.models.User">
2. select \* from user where id = #{password};
3. </select>

**1.在**parameterType **为简单类型时，#{}的内容是无所谓的，跟参数没有关系，仅是占位作用**

1. <select id="selectByUserId" parameterType="cn.cxd.models.User" resultType="cn.cxd.models.User">
2. select \* from user where id = #{id};
3. </select>
5. <select id="selectByUserVo" parameterType="cn.cxd.models.UserQueryVo" resultType="cn.cxd.models.User">
6. select \* from user where id = #{user.id};
7. </select>

**2.在**parameterType **为model类型时，#{}的内容应该是model中的属性名，要对应上，但是model中并不需要提供get/setter方法，因此是算model而不算是bean，至于是怎么注入成一个对象的还有待考究，也许是反射直接getFileds，再注入属性，不大清楚。**

1. <select id="selectByName" parameterType="String" resultType="cn.cxd.models.User">
2. select \* from user where username like '%${avdc}%';
3. </select>
4. **如果是模糊查询，那么需要注意的是 注入式的 ‘${}’ ,使用$符号同时需要单引号**
5. SqlSession sqlSession = sqlSessionFactory.openSession();
6. List<User> users =  sqlSession.selectList("selectByName", username);
7. sqlSession.close();
8. **return** users;

**4.SelectList: 查询由select出来的元组包装成若干个对象，数目不限**

**SelectOne: 同上，但是仅查询0~1个对象，如果由SelctOne去接收多个对象那么就会有TooManyResultException**

1. **Model类需要无参构造器，否则我们自己使用 select \* 时的returnType 想转成model类，而当数据库表中的属性 多余 对应的model类中的成员变量时，select\*返回出来的model对象的对应赋值上会出现这种情况：数据库的属性值被多次往下赋值给model类出现的成员变量值，而如果对应的上的属性可以被完好的赋值上去。如果提供了无参构造器，那么这些model类中不能被对应赋值上去的会为null，或是其他默认值。**
2. **Dao类依赖于modelMap.xml的增删改method标注的id调用,**

**int row = sqlSession.insert("insert", user);**

**sqlSession.close();**

**第一个是modelMap.xml中的方法id，**

**第二个参数是需要被传递进入的参数，也就是parameterType, 参数的传递只有一个，但允许封装，也就是说可以是数组，list，model对象，以此完成多参数的传递功能。**

1. **OpenSession传入的参数如果是true,则代表事务会自动提交，也就是insert,update,delete不需要将session.commit上去，会自动执行，默认为false，需要手动提交事务。**

SqlSession sqlSession = sqlSessionFactory.openSession(**true**);

1. **全局配置文件：**
2. <mappers>
3. <mapper resource="cn/cxd/models/user.xml"/>
4. </mappers>

**Resource 的配置是 / 而不是包名，是路径。**

**在加载局部资源上有4中方式：**

1. <mapper resource="cn/cxd/modelMapper/userMapper.xml"/>
2. <mapper **class**="cn.cxd.modelMapper.UserMapper"></mapper>
3. <**package** name="cn.cxd.modelMapper"/>

**4. <mapper url>**

**需要注意：其中的<package>包标签的配置仅适用于Mapper，不使用于Dao**

**另外：在配置环境时，需要设置properties resource资源路径，用于加载数据库连接配置文件，随后在配置envrioments中配置上dataSource 标签体中的各个propety成员变量的值。**

1. **使用步骤：**
2. **准备Jars**
3. **准备log4j.properties**
4. **准备C3p0.properties**
5. **写model类**
6. **准备sql中的数据**
7. **配置model.xml局部配置xml，准备一些dao方法**
8. **配置sqlConfig.xml全局配置xml，读入C3p0.properties和model.xml局部配置文件**
9. **配置dao层的类，应用上model.xml提供的方法**
10. **在main中实现调用dao层方法。**