dao层接口的作用与分析

--在项目a-maven-mybatis01中我们发现UserDao接口除了将里面的方法与mapper文件中的每条sql语句的id绑定了之外就没有使用了，好像程序执行并没有用到接口中的方法。

1、下面展示mybatis中dao层接口的传统使用方式

在idea-maven-mybatis过程中添加新的项目模块b-maven-mybatis02，内容直接复制a-maven-mybatis01中的内容。
b-maven-mybatis02项目中在dao包下新建一个impl包，然后在其中新建一个UserDao接口的实现类UserDaoImpl.java，实现方法的内容就是执行某条sql语句所写的代码，就是MyApp类中的内容，如下

```
package com.studymyself.dao.impl;

import com.studymyself.dao.UserDao;
import com.studymyself.entity.User;
import com.studymyself.utils.MyBatisUtil;
import org.apache.ibatis.session.SqlSession;

import java.util.List;

public class UserDaoImpl implements UserDao {
    @Override
    public List<User> getAll() {

        //获取sqlSession对象
        SqlSession sqlSession = MyBatisUtil.getSqlSession();
        //mapper文件中某条sql语句的唯一标识
        String selectId = "com.studymyself.dao.UserDao.getAll";
        //执行sql语句
        List<User> users = sqlSession.selectList(selectId);
        //关闭资源
        MyBatisUtil.close(sqlSession);
        //将存储查询结果集映射到User对象的User对象的集合返回
        return users;

    }

    @Override
    public int addUser(User user) {
        //获取sqlSession对象
        SqlSession sqlSession = MyBatisUtil.getSqlSession();

        //mapper文件中某条sql语句的唯一标识
        String selectId = "com.studymyself.dao.UserDao.addUser";
        //执行sql语句
        int count = sqlSession.insert(selectId,user);
        //提交事务
        sqlSession.commit();
        //关闭资源
        MyBatisUtil.close(sqlSession);
        return count;
```

```
    }
}
```

这样我们MyApp类中main方法就直接创建一个UserDao类型的UserDaoImpl实现类对象，然后用该对象调用对应的增删改查等方法就可以了。

在这里我们在测试类中测试接口的实现方法TestUserDaoImpl.java如下

```
package com.studymyself;

import com.studymyself.dao.UserDao;
import com.studymyself.dao.impl.UserDaoImpl;
import com.studymyself.entity.User;
import org.junit.Test;

import java.util.List;

public class TestUserDaoImpl {

    /**
     * 测试接口中的getAll方法
     */
    @Test
    public void testGetAll(){

        //创建实现类对象，使用多态
        UserDao userDao = new UserDaoImpl();
        //调用查询方法，返回list集合
        List<User> users = userDao.getAll();
        //遍历集合
        for (User user:
             users) {
            System.out.println(user);
        }
    }

    /**
     * 测试接口中添加数据的方法
     */
    @Test
    public void testAddUser(){

        //创建实现类对象，使用多态
        UserDao userDao = new UserDaoImpl();
        //创建User对象，赋值
        User user = new User();
        user.setLoginName("10086");
        user.setLoginPwd("10086zzz");
        user.setRealName("中国移动");
        //调用添加方法，返回新增的记录条数
        int count = userDao.addUser(user);
        System.out.println("新增记录条数"+count);
```

```
    }

}
```

分析：

代码经过测试，可以执行。
--与之前的代码比较
1、因为在工具类中在获取SqlSessionFactory对象的时候已经在类加载阶段把IO异常捕捉了，如果出现异常，意味着没法获取到SqlSession对象，sql语句就没法执行，也就不需要在修改数据库的方法中添加回滚操作。至于什么时候添加回滚操作，以后项目中遇到异常时可以在catch语句中添加回滚方法。

2、这样传统的dao接口使用方式，与之前比较也并没有简化多少工作。在dao接口的实现类中每个方法之间相比较还是有许多重复的步骤的，那么为什么还要这样写呢？

--mybatis动态代理
在a-maven-mybatis01中我们分析过，mapper文件中的namespace的值和sql语句标签中的id值确定了这条sql语句在这个项目中的唯一标识，这样在多表多mapper文件的项目中也可以唯一确定要执行哪一条sql语句。还有我们发现，我们执行sql语句时，填入mapper文件中sql语句时mybatis会根据返回类型判断使用sqlSession对象的哪一个方法。

通过上面对应关系，我们就可以使用mybatis的动态代理来让mybatis底层动态生成dao接口的实现类：
1、--mybatis怎么动态生成接口的实现类？
接口的全限定名是：com.studymyself.dao.UserDao，而mapper文件中的namespace的值就是接口限定名，这样就可以让mybatis用反射机制来创建哪个接口的实现类。而且将这个(接口)实现类和这个mapper文件进行绑定

2、--mybatis怎么知道接口中的方法是需要执行绑定的映射文件的哪条SQL语句？
接口中的方法名与映射文件中sql语句的id一样，这样mybatis就可以知道在实现类的实现方法中调用sqlSession的方法的代码阶段通过反射获取方法名的字符串作为SqlSession对象中方法中的第一部分参数如：sqlsession.("方法名", xxx)

3、--mybatis怎么知道接口中的方法是需要执行SqlSession对象中的哪个方法？
上面2知道这个实现方法应该执行的是哪条sql语句后，通过该sql语句的标签是<select>还是<insert>等以及方法的返回值是什么类型来确定调用的是selectList还是select亦或是insert方法 --： public List<User> getAll();

--mybatis动态代理，就是mybatis根据dao接口，创建该接口的实现类，并创建这个类的对象。帮助我们完成SqlSession对象方法的调用，访问数据库。我们就没必要自己手动创建dao层接口的实现类了，直接在需要的类中调用获取实现类对象的方法，然后根据接口中的对应方法来实现增删改查访问数据库的操作。
要求就是：
    1、sql映射文件中namespace属性的值必须是某一接口的全限定名：接口绑定映射文件
    2、接口中的方法根据实现功能要和映射文件中的sql语句的id一致
    --接口中方法的参数类型与映射文件中的parameterType属性的作用一样。
    其他要求视情况而定

--获取mybatis动态代理生成的实现类的对象的方法：
```
UserDao userDao = sqlSession.getMapper(接口的Class对象);
UserDao userDao = sqlSession.getMapper(UserDao.class);
```

2、MyBatis动态代理的测试

将项目b-maven-mybatis02的代码赋值到c-maven-mybatis03中，删掉impl包。
其中在测试类中改为如下代码

```
import java.util.List;

public class TestUserDaoImpl {

    /**
     * 测试接口中的getAll方法
     */
    @Test
    public void testGetAll(){

        //获取SqlSession对象
        SqlSession sqlSession = MyBatisUtil.getSqlSession();
        //通过SqlSession对象中的方法获取实现类对象
        UserDao userDao = sqlSession.getMapper(UserDao.class);
        //调用查询方法，返回list集合
        List<User> users = userDao.getAll();
        //遍历集合
        for (User user:
             users) {
            System.out.println(user);
        }
    }

    /**
     * 测试接口中添加数据的方法
     */
    @Test
    public void testAddUser(){

        //获取SqlSession对象
        SqlSession sqlSession = MyBatisUtil.getSqlSession();
        //通过SqlSession对象中的方法获取实现类对象
        UserDao userDao = sqlSession.getMapper(UserDao.class);
        //创建User对象，赋值
        User user = new User();
        user.setLoginName("10086");
        user.setLoginPwd("10086zzz");
        user.setRealName("中国移动");
        //调用添加方法，返回新增的记录条数
        int count = userDao.addUser(user);
         //提交事务
        sqlSession.commit();
        System.out.println("新增记录条数"+count);

    }

}
```

可以看到，使用mybati动态代理的方式相较于传统的使用dao接口的方式，多了获取SqlSession对象的步骤和提交事务的步骤，省却了很多步骤。