一、出发点

Mybatis面向接口编程主要解决以下几种问题

1.配置文件中namespce命名空间问题

2.sql关联id问题

3.返回值问题

4.传入参数问题

二、Message.xml配置文件

namespce为接口的全类名

id是接口方法名

parameterType是对应方法的参数类型

返回值有resultMap或resultType决定

<?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.imooc.dao.IMessageDao">

<resultMap type="com.imooc.bean.Message" id="MessageResult">

<id column="id" jdbcType="INTEGER" property="id"/>

<result column="command" jdbcType="VARCHAR" property="command"/>

<result column="description" jdbcType="VARCHAR" property="description"/>

<result column="content" jdbcType="VARCHAR" property="content"/>

</resultMap>

<!-- select标签用于书写查询语句-->

<select id="queryMessageList" parameterType="com.imooc.bean.Message" resultMap="MessageResult">

select id,command,description,content from message

<where><!--

test中书写布尔表达式，如果成功则拼接下边的条件, command是parameterType中的成员变量

拼接sql的时候会将‘#{command}’替换成?,取值类似于ognl

-->

<if test="command != null and !"".equals(command.trim())">

and command=#{command}

</if>

<!-- like 查询一般会拼接concat()拼接两个字符串 -->

<if test="description != null and ''!=description.trim()">

and description like concat(concat('%',#{description}),'%')

</if>

</where>

</select>

</mapper>

三、接口IMssageDao.java

package com.imooc.dao;

import java.util.List;

import org.apache.ibatis.annotations.Param;

import com.imooc.bean.Message;

/\*\*

\* 于Message配置文件对应的接口

\* \*/

public interface IMessageDao {

/\*\*

\* @description: 如果想要执行配置文件中的某天sql语句,就可以定义于配置文件中id相同的方法名

\* @param: Message message 是配置文件中对应sql中需要的parameterType

\* @return: 返回类型即接口返回类型对应配置文件sql申明的返回类型决定

\* \*/

public List<Message> queryMessageList(Message message);

}

四、使用

调用的时候通过获取到sqlSession中的方法，就可以调用

用法如下边的queryMesageList方法

注：这里感觉使用接口方法还需要多写一个接口类文件，并没有觉得有有什么优点，但是当mybatis整合spring之后，下边的MessageDao.java是不用写的，我们只要写接口文件和对应的配置文件xxx.xml，其他的就交给mybatis管理

package com.imooc.dao;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.apache.ibatis.session.SqlSession;

import com.imooc.bean.Message;

import com.imooc.common.Page;

import com.imooc.db.DBAccess;

public class MessageDao {

DBAccess dbAccess = new DBAccess();

//不是用接口方法

public List<Message> queryMessageList2(String command, String description) throws Exception {

List<Message> messagesList = null;

SqlSession session = null;

try {

//获取SqlSession

session = dbAccess.getSqlSession();

Message message = new Message();

message.setCommand(command);

message.setDescription(description);

SqlSession中提供了很多数据库操作的方法，如selectXXX实现对数据库的查询

\*的方法Message.class.getName().queryMessageList获取配置文件中id=queryMessageList

\*对应的sql,message为传入的数据封装对象，一般只能传入一个对象，还可以常用map封装参数

\*

messagesList = session.selectList(Message.class.getName()+".queryMessageList", message);

} catch (Exception e) {

e.printStackTrace();

throw new Exception(e.getMessage());

} finally {

if (session != null) {

session.close();

}

}

return messagesList;

}

//使用接口方法

public List<Message> queryMessageList(String command, String description) throws Exception {

List<Message> messagesList = null;

SqlSession session = null;

try {

//获取SqlSession

session = dbAccess.getSqlSession();

Message message = new Message();

message.setCommand(command);

message.setDescription(description);

/\*

\* 使用动态代理原理

\* MapperProxy implements InvocationHandler

\* 类里边有个MapperProxy.invoke()

\* Proxy.newPproxyInstance(类加载器，接口，MapperProxy对象) 返回一个IMssageDao的代理对象

\*

\*因为：session.getMapper() == Proxy.newProxyInstance()

\*所以：IMssageDao imssageDao = Proxy.newProxyInstance()

\*imssageDao.queryMessageList() == MapperProxy.invoke()

\*

\* \*/

IMessageDao iMessageDao = session.getMapper(IMessageDao.class);

messagesList = iMessageDao.queryMessageList(message);

} catch (Exception e) {

e.printStackTrace();

throw new Exception(e.getMessage());

} finally {

if (session != null) {

session.close();

}

}

return messagesList;

}

}

注：本案例思想由慕课网提供指导