# Mybatis

## Mybatis是什么

MyBatis 是一个优秀持久层框架，由apache的开源项目iBatis演化而来，主要用于解决数据持久化问题，底层实现了对JDBC操作的封装。

## MyBatis 应用场景

* 1. 简化JDBC编写步骤的复杂度
  2. 更好的实现ORM(对象关系映射)

## 映射器

### 查询语句

id 配合Mapper 的全限定名，联合成为一个唯一的标识，用于标识这条sql语句

parameterType 标识这条sql语句接受的参数类型，可以是mybatis系统自定义或者自定义的别名，如int,string ,float,也可以是类的全限定名如：com.jt.pojo.User.

resultTpye 表示这条sql语句返回的结果类型，可以是mybatis系统自定义或者自定义的别名，如int,string ,float,也可以是类的全限定名如：com.jt.pojo.User.

resultMap

## 自动映射和驼峰映射

在setting文件中配aotoMapperingBehavior和mapUnderscoreCamelCase,他们是控制自动映射和驼峰映射的开关。

AotoMapperingBehavior

NONE,不进行自动映射

Partial，默认值对没有嵌套的结果集进行自动映射

Full，对所有结果集进行自动映射，包括嵌套的结果集

自动映射：

<select id="queryUser" parameterType="long" resultType="com.jt.pojo.User">  
 SELECT id user\_name as username from t\_user where id = #{id}  
</select>

驼峰映射

<select id="queryUser" parameterType="long" resultType="com.jt.pojo.User">  
 SELECT id user\_name from t\_user where id = #{id}  
</select>

## 传递多个参数

1. 使用map接口传递多个参数（可读性差）

public List<User> queryUser(Map<String,Object> parameterMap);

详细：

Map <String ,Object> parameterMap = new HashMap<String,Object>();

ParameterMap.put(“userName”,”小明”);

ParamteerMap.put(“note”,”that is note”);

List<user> list = userMapper.queryUser(parameterMap);

1. 使用注解传递多个参数（推荐使用）

public List<User> queryUser(@Param(“userName”)String username,@Param(“note”)String note);

1. JavaBean的形式

首先建立需要传递参数pojo类，写出set和get的方法

public class UserParams{

private String username;

private String note;

}

定义接口（Dao）

public List<User> findUserByJavaBean(UserParams userParams);

引入javabean的属性,在业务层处理数据

UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

UserParams userParams = new UserParams();

userParams.setUserName(“xiaoming”);

userParams.setNote(“that is note”);

List <User> list = userMapper.findUserByJavaBean(userParams)

<**select id="queryJavaBean" parameterType="com.jt.mapper.UserParams" resultType="user"**>  
 SELECT user\_name as userName note from t\_user where user\_name like concat (‘%’,#{userName},’%’)and note like concat (‘%’,#{note},’%’)  
 </**select**>

混合使用

案例分页

1.分页pojo

public class PageParams{

private int start;

private int limit;

}

持久层

public List<User>findByPage(@Params(“params”) UserParams userParams, @Param(“page”) PageParams pageParams)；

Mapper.xml

<**select id="findByPage" resultType="user"**>  
 SELECT id ,user\_name as userName ,note from t\_user WHERE user\_name LIKE concat ('%',#{params.userName},'%') and note LIKE concat ('%',#{params.note},'%') limit start =#{page.start} ,#{page.limit}  
</**select**>

总结，传递参数的4种方式

1.使用Map的传递的方式，最好不使用，业务的可读性差，最好不使用，

[2.使用@Params](mailto:2.使用@Params)注解的形式，推荐使用，当传递参数小于5个时建议使用

1. 使用javaBean的形式，传递参数时建议使用
2. 对于混合参数，要明确参数的合理性

## resultMap映射的

</select>  
 <resultMap id="userMap" type="user">  
 <id property="id" column="id"/>  
 <result property="userName" column="user\_name"/>  
 <result property="note" column="note"/>  
 </resultMap>  
 <select id="getUserByResultMap" resultMap="userMap" resultType="user">  
 select id ,user\_name ,note from t\_user where id=#{id} and user\_name like concat ('%',#{userName},'%') and note LIKE concat ('%',#{note},'%')  
 </select>

1. resultMap中的id代表它的标识，type代表它的映射的类，可以写别名。
2. 子类中的id代表它的主健，property代表其pojo的书香名，cloun代表其sql的名称

## 分页源码

1. 首先通过反射方法来创建工具类

**package** com.jt.util;  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** org.apache.ibatis.io.Resources;  
**import** org.apache.ibatis.session.SqlSession;  
**import** org.apache.ibatis.session.SqlSessionFactory;  
**import** org.apache.ibatis.session.SqlSessionFactoryBuilder;  
*//辅助类。创建SqlSession对象,通过反射来创建***public class** SqlSessionFactoryUtil {  
 *//反射来创建* **private final static** Class<SqlSessionFactoryUtil> ***LOCK*** = SqlSessionFactoryUtil.**class**;  
 *//创建SqlSessionFactory对象* **private static** SqlSessionFactory *sessionFactory* = **null**;  
 *//构造方法* **public** SqlSessionFactoryUtil() {  
 }  
 *//get方法  
 //如何通过反射获取对象* **public static** SqlSessionFactory getSessionFactory() {  
 *//防止多线程的实例化* **synchronized** (***LOCK***) {  
 **if** (*sessionFactory* != **null**) {  
 **return** *sessionFactory*;  
 }  
 *//创建SqlSessionFactory* String resources = **"mybaitis-config.xml"**;  
 InputStream inputStream;  
 **try** {  
 inputStream = Resources.*getResourceAsStream*(resources);  
 *sessionFactory* = **new** SqlSessionFactoryBuilder().build(inputStream);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 **return null**;  
 }  
 **return** *sessionFactory*;  
 }  
 }  
 **public static** SqlSession openSqlSession(){  
 **if**(*sessionFactory* == **null**){  
 *getSessionFactory*();  
 }  
 **return** *sessionFactory*.openSession();  
 }  
}

1. 分页工具类

package com.jt.util;  
  
public class RowBounds {  
 private static final int *NO\_ROW\_OFFSET* = 0;  
 private static final int *NO\_ROW\_LIMIT* = Integer.*MAX\_VALUE*;  
 private static final RowBounds *DEFAULT* = new RowBounds();  
 private int offset;  
 private int limit;  
  
 public RowBounds() {  
 this.offset = *NO\_ROW\_OFFSET*;  
 this.offset = *NO\_ROW\_LIMIT*;  
 }  
  
 public RowBounds(int offset, int limit) {  
 this.offset = offset;  
 this.limit = limit;  
 }  
  
 public int getOffset() {  
 return offset;  
 }  
  
 public void setOffset(int offset) {  
 this.offset = offset;  
 }  
  
 public int getLimit() {  
 return limit;  
 }  
  
 public void setLimit(int limit) {  
 this.limit = limit;  
 }  
}

1. 测试类

**package** mybatis.test;  
**import** com.jt.mapper.UserMapper;  
**import** com.jt.pojo.User;  
**import** com.jt.util.RowBounds;  
**import** com.jt.util.SqlSessionFactoryUtil;  
**import** org.apache.ibatis.session.SqlSession;  
**import** java.util.List;  
**public class** TestRowBounds {  
 **public static void** main(String[] args) {  
 SqlSession sqlSession = **null**;  
 **try** {  
 sqlSession = SqlSessionFactoryUtil.*openSqlSession*();  
 UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);  
 RowBounds rowBounds = **new** RowBounds();  
 List<User> list = userMapper.findUserByRowBounds(**"user\_name"**,**"note"**,rowBounds);  
 }**catch** (Exception e){  
 e.printStackTrace();;  
 }**finally** {  
 **if**(sqlSession != **null**){  
 sqlSession.close();  
 }  
 }  
 }  
}

1. 映射user.xml的接口

**package** com.jt.mapper;  
**import** com.jt.pojo.User;  
**import** com.jt.util.RowBounds;  
**import** org.apache.ibatis.annotations.Param;  
**import** java.util.List;  
**public interface** UserMapper {  
 List<User> findUserByRowBounds(@Param(**"userName"**) String username,@Param(**"note"**) String note, RowBounds rowBounds);  
}

1. user.xml

<**select id="findUserByRowBounds" resultType="user"**>  
 SELECT user\_name as userName ,note where user\_name LIKE user\_name like concat ('%',#{userName},'%') and note LIKE concat ('%',#{note},'%')  
</**select**>

## insert元素

主键回填

在插入user 和role表的时候，通过主键id去关联，所以必须拿到用户的主键

<insert id="insertRole " parameterType="role" useGeneratedKeys="true" keyColumn="id">  
 insert into t\_role(role\_name ,note) values (#{roleName},#{note});  
</insert>

解释：

userGenneratedKeys表示JDBC的statement对象调用getGenneratedKey的方法返回主键

keyProtperty表示匹配的pojo属性的这个主键，这里指id

### 自定义主键

<**insert id="saveRole" parameterType="role"**>  
 <**selectKey keyProperty="id" resultType="long" order="BEFORE"**>  
 SELECT if(*max*(id)=null,1,*max*(id)+3) from t\_role  
 </**selectKey**>  
 INSERT into t\_role (id,user\_name,note`` )VALUES (#{id},#{userName},#{note})  
</**insert**>

通过selecttKey元素定义

keyProperty作为pojo的属性

order = “before” 表示在sql语句之前处理

## update和delete元素

与insert语句用法相同

## sql元素

当使用大量的重复的sql语句

<sql id="userCols" >id user\_name ,note</sql>  
<select id="getUser" resultType="user">  
 SELECT <include refid="userCols" /> from t\_user where id = #{id}  
</select>

参数

储存參数的类型

IN

OUT

INIOUT

级联

延迟加载

重点：动态sql

if元素

不填写时就不要它作为查询语句

例如

<if test=”roleName !=null and roleName=’’”/>

例子：

* + - 1. 当角色号不为空，则按照角色号就行查询
      2. 当角色号为空，而用户名不为空，则按照角色名来查询
      3. 当角色号和角色名都为空，则按照角色备注来查询

<resultMap id="roleResultMap" type="role">  
 <id property="id" column="id"/>  
 <result property="roleNo" column="role\_no"/>  
 <result property="roleName" column="role\_name"/>  
 <result property="note" column="note"/>  
</resultMap>  
<select id="queryRoleById" resultMap="roleResultMap">  
 SELECT role\_no ,role\_name,note from t\_role where 1=1  
 <choose>  
 <when test="roleNo !=null and roleNo !=''">  
 AND role\_no = #{roleNo}  
 </when>  
 <when test="roleName !=null and roleName !=null">  
 AND role\_name= #{roleName}  
 </when>  
 <otherwise>  
 AND note is not NULL   
 </otherwise>  
 </choose>  
</select>

1. choose
2. when
3. otherwise

1=1的作用：

如果去掉的话where直接和and直接链接在一起，会报错

4.可以通过下面的例子去修改，不要1=1的sql语句

<**select id="queryRoleById" resultMap="roleResultMap"**>  
 SELECT role\_no ,role\_name,note from t\_role  
 <**where**>  
 <**if test="role\_name != null and role\_name != ''"** >  
 and role\_name like concat ('%', #{roleName},'%')  
 </**if**>  
 <**if test="note != null and note != ''"** >  
 and note like concat ('%',#{note},'%')  
 </**if**>  
 </**where**>  
</**select**>

## trim、where、set的元素

### trim 的用法

<select id="queryRoleById" resultMap="roleResultMap">  
 SELECT role\_no,role\_name,note from t\_role  
 <trim prefix="where" prefixOverrides="and">  
 <if test="roleName != null and roleName != ''">  
 and role\_name like concat ('%',#{roleName},'%')  
 </if>  
 </trim>  
</select>

prefix表示语句的前缀是where

prefixOverrides表示需要去掉的字符

### set的用法

hibernate中如果更新的话，需要把所有的字段发送给持久层，而mybatis中的set只需要把需要修改的字段

<**update id="updateRole" parameterType="role"**>  
 UPDATE t\_role  
 <**set**>  
 <**if test="role\_name != null and role\_name =''"**>  
 role\_name = #{roleName}  
 </**if**>  
 WHERE role\_no = #{roleNo}  
 </**set**>  
</**update**>

## foreach元素

**作用：遍历集合，通常用于in的sql关键字中**

<select id="findUserBySex" parameterType="user">  
 SELECT *\** from t\_role where role\_no in   
 <foreach collection="roleNoList" item="roleNo" index="index" open="(" separator="," close=")">  
 #{roleNo}  
 </foreach>  
</select>

collection 配置roleNoList 传进来的参数的名称，是数组，集合，set

item 循还中当前的元素

index配置当前元素的下标

open和close以什么元素包装起来

separate 以什么符号把元素之间隔开

## bing元素

绑定语句，就可以直接调用，可以在mysql和db2和oracal中使用

<**select id="findRole" resultType="role"** >  
 <**bind name="roleNameb" value="'%'+roleName+'%'"**/>  
 <**bind name="roleNob" value="'%'+roleNo+'%'"**/>  
 SELECT role\_no as roleNo ,role\_name as roleName ,note from t\_role  
 where role\_no like #{roleNob} and role\_name like #{roleNameb}  
</**select**>

# mybatis的运行和解析原理

## 构建SqlSessionFactory过程

1. **通过Xml.ConfigBuilder解析xml文件，读取文件的参数，并将内容存入Configuration，并且为单例**
2. Configuration对象去创建SqlSessionFactory接口，它的实现类为DefaultSqlSessionFactory，大部分情况不用创建实现类

## Configuration的作用

读取配置文件，基础配置文件xml和映射器xml的配置文件

初始化基础配置：

properties 全局参数

typeAliases别名包

Plugins插件

ObjectFactory工厂

objectWrapperFactory 对象包装工厂

reflectionFractory 放射工厂

settings 环境设置

environments数据库坏境

databaseIdProvider类数据库标识

typeHandlers 类型转换器

Mapper映射器

1. 提供单例，为后续创建SqlSessionFactory提供参数

## 构建映射器的内部组成

1. 一条sql由MapperedStatement、SqlSource、BoundSql组成
2. MapperedStatement保存映射节点，是一个类
3. SqlSource接口，生成BoudSql对象的方法
4. BoundSql有parameterMappings，prameterObject和sql
5. prameterObject参数本身，传递对象，传递单个对象“int ,string ,long ”.传递多个对象（@param “key1”，p1, @param”key2”,p2）或者：Map<String,Object>它的键包含：{“key1”:p1,”key2”:p2}

## 构建sqlSessionFactory

sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

## mybateis通过流文件生成configuration对象，进而构建sqlSessionFactory对象