知识回顾：

1. Mybatis结构图



#{}和${}区别？

#{}它可以防止sql注入，它使用？占位符方式（推荐）

${}直接原样输出，容易产生sql注入，直接拼接字符串（特殊 orderby）

参数parameterType

支持简单类型：integer,string,long,double…

支持对象类型：array/list/map/pojo

今天任务：

1. 动态SQL语句（主干在变化）一系列标签
2. 小小的优化，减少代码量
3. 对象关联（一对一、一对多、多对多、多对一）resultMap
4. 动态SQL语句

SELECT \* FROM USER

WHERE id=5 AND NAME='张居正'

两个查询条件，这个条件决定于我的参数

如果id存在条件就包含：id=5；如果id没有值去掉这个条件

如果name存在条件就包含：NAME='张居正'；如果name不存在，去掉

步骤：

1. 传递参数，传递两个（支持多个）

Map万能传参，多个kv

<select id="find" parameterType="map" resultType="cn.tedu.mybatis.pojo.User">

select id,name,birthday,address from user

where id=#{id} and name=#{name}

</select>

//定义参数，参数复杂对象，可以容纳多个值

Map<String,Object> map1 = new HashMap<String,Object>();

map1.put("id", 5);

map1.put("name", "张居正");

List<User> userList = session.selectList(ns, map1);

支持比较方便的方式，而且更加简洁pojo

<!-- 查询所有，resultType返回值，标识返回类型List<> -->

<select id="find" parameterType="cn.tedu.mybatis.pojo.User" resultType="cn.tedu.mybatis.pojo.User">

select id,name,birthday,address from user

where

id=#{id}

and name=#{name}

and address like #{address}

</select>

@Test

public void findPojo(){

//SqlSession非安全

SqlSession session = factory.openSession();

String ns = "cn.tedu.mybatis.pojo.UserMapper.find";

//定义参数，参数复杂对象，可以容纳多个值

User user = new User();

user.setId(5);

user.setName("张居正");

user.setAddress("%村%");

List<User> userList = session.selectList(ns, user);

for(User u:userList){

System.out.println(u);

}

}

1. 多余的and如何处理

3）集合变量（典型：批量删除）

动态标签：

1）<if test=””></if> 判断字段是否出现，出现就把标签中的内容拼接SQL语句中，如果判断条件不成立时，标签中的内容就不出现。

2）<where></where> 替代WHERE 1=1，增加where关键字，会去掉and或者or

3）<set> 显示set关键字，自动把最后一个,去掉。没有,时，就不处理

4）<foreach collection=”array/list/ids” open=“(” close=”) item=”id” separator=”,”>#{id}

5）array new Integer[]{6,7,8};

6）list list.add(6);list.add(7);…

7）map parameterType=”map” foreach collection=”ids” item=id #{id}

8）<sql id="cols"> <include refid=”cols”> 声明字符串变量，后面 SQL可以直接引用，修改一个地方，其他引用地方自动修改，一般就用于字段名称

9）<typeAlias type="cn.tedu.mybatis.pojo.User" alias="User"/> 简化UserMapper.xml内容

UserMapper.xml

<?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">

<!-- namespace命名空间，唯一特性 -->

<mapper namespace="cn.tedu.mybatis.pojo.UserMapper">

<!-- 集中声明字符串 -->

<sql id="cols">id,name,birthday,address</sql>

<!-- 查询所有，resultType返回值，标识返回类型List<> -->

<select id="find" parameterType="cn.tedu.mybatis.pojo.User" resultType="cn.tedu.mybatis.pojo.User">

select <include refid="cols"/> from user

<where>

<if test="id != null">id=#{id}</if>

<if test="name != null">and name=#{name}</if>

<if test="address != null">

and address like #{address}

</if>

</where>

</select>

<!-- 只返回一个值，返回一个整数值 -->

<select id="count" resultType="integer">

SELECT COUNT(\*) FROM user

</select>

<!-- 获取一个用户的信息（一条记录）#{id}参数，解析获取id值 -->

<select id="get" parameterType="integer" resultType="User">

SELECT <include refid="cols"/> FROM USER WHERE id = #{id}

</select>

<!-- 新增 -->

<insert id="insert">

INSERT INTO USER

(id,NAME,birthday,address)

VALUES(6,'tony','2018-06-19','北京')

</insert>

<!-- 修改 -->

<update id="update" parameterType="cn.tedu.mybatis.pojo.User">

UPDATE USER

<set>

<if test="name!=null">NAME = #{name},</if>

<if test="address!=null">ADDRESS = #{address},</if>

</set>

WHERE id = #{id}

</update>

<!-- 删除 -->

<delete id="delete" parameterType="integer">

DELETE FROM USER WHERE id=#{pid}

</delete>

<!-- 删除多条 -->

<delete id="deleteArray">

DELETE FROM USER WHERE id

in <foreach collection="array"

open="(" close=")"

item="id" separator=",">

#{id}

</foreach>

</delete>

<!-- 删除多条 -->

<delete id="deleteList">

DELETE FROM USER WHERE id

in <foreach collection="list"

open="(" close=")"

item="id" separator=",">

#{id}

</foreach>

</delete>

<!-- 删除多条 -->

<delete id="deleteMap" parameterType="map">

DELETE FROM USER WHERE id

in <foreach collection="ids"

open="(" close=")"

item="id" separator=",">

#{id}

</foreach>

</delete>

<!-- 排序 -->

<select id="orderby" parameterType="map"

resultType="cn.tedu.mybatis.pojo.User">

SELECT \* FROM USER ${ob}

</select>

</mapper>

测试类

package mybatis1803;

import java.io.IOException;

import java.io.InputStream;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.apache.ibatis.io.Resources;

import org.apache.ibatis.session.SqlSession;

import org.apache.ibatis.session.SqlSessionFactory;

import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import org.junit.Before;

import org.junit.Test;

import cn.tedu.mybatis.pojo.User;

public class TestDynamicSQL {

private SqlSessionFactory factory; // 线程安全

// 初始化工厂对象

@Before

public void init() throws IOException {

String resource = "sqlMapConfig.xml";

InputStream is = Resources.getResourceAsStream(resource);

factory = new SqlSessionFactoryBuilder().build(is);

}

@Test

public void get() {

// SqlSession非安全

SqlSession session = factory.openSession();

String ns = "cn.tedu.mybatis.pojo.UserMapper.get";

User u = session.selectOne(ns, 5);

System.out.println(u);

}

@Test

public void findmap() {

// SqlSession非安全

SqlSession session = factory.openSession();

String ns = "cn.tedu.mybatis.pojo.UserMapper.find";

// 定义参数，参数复杂对象，可以容纳多个值

Map<String, Object> map1 = new HashMap<String, Object>();

map1.put("id", 5);

map1.put("name", "张居正");

map1.put("address", "%村%"); // 注意like语法

List<User> userList = session.selectList(ns, map1);

for (User u : userList) {

System.out.println(u);

}

}

@Test

public void findPojo() {

// SqlSession非安全

SqlSession session = factory.openSession();

String ns = "cn.tedu.mybatis.pojo.UserMapper.find";

// 定义参数，参数复杂对象，可以容纳多个值

User user = new User();

// user.setId(5);

user.setName("张居正");

// user.setAddress("%村%");

// 没有设置address，pojo有get，mybatis拿到get方法，参数未被设置过，默认null

List<User> userList = session.selectList(ns, user);

for (User u : userList) {

System.out.println(u);

}

}

@Test

public void updatePojo() {

// SqlSession非安全

SqlSession session = factory.openSession();

String ns = "cn.tedu.mybatis.pojo.UserMapper.update";

// 定义参数，参数复杂对象，可以容纳多个值

User user = new User();

user.setId(5);

user.setAddress("村4");

user.setName("张居正4");

session.update(ns, user);

session.commit();

}

@Test

public void deleteArray() {

// SqlSession非安全

SqlSession session = factory.openSession();

String ns = "cn.tedu.mybatis.pojo.UserMapper.deleteArray";

//创建一个整形数组，直接初始化

Integer[] arr = new Integer[]{6,7};

session.delete(ns, arr);

session.commit();

}

@Test

public void deleteList() {

// SqlSession非安全

SqlSession session = factory.openSession();

String ns = "cn.tedu.mybatis.pojo.UserMapper.deleteList";

List<Integer> list = new ArrayList<Integer>();

list.add(6);

list.add(7);

list.add(8);

session.delete(ns, list);

session.commit();

}

@Test

public void deleteMap() {

// SqlSession非安全

SqlSession session = factory.openSession();

String ns = "cn.tedu.mybatis.pojo.UserMapper.deleteMap";

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

map.put("ids", new Integer[]{6,7,8});

session.delete(ns, map);

session.commit();

}

}

使用动态SQL，它mybatis核心

利用标签：

1）<if>判断字段或者部分内容是否在sql语句中出现。

2）<where> 多个条件时，会可能造成某个条件没有出现 and 或者 or开头情况，它实现把这个头去掉

3）<set>只能用在update SQL中，多个字段更新时，前面的字段不出现，最后面多了一个逗号，它把这个逗号去掉。

4）<foreach> 只能用在in子查询串使用。 in(6,7,8) 通过配置遍历传入集合，拼接成字符串。

传参：

1. integer
2. string

上面的只能一次传递一个参数get（id）

1. array
2. list
3. map

上面的集合的变量，一次可以传递多个值，只能传递一个类型的变量String[]

1. pojo

自定义，里面可以非常复杂，不限制个数，不限制类型

Mybatis操作方式

1. xml操作方式（session.selectList()）
2. 接口操作方式，面向对象方式思考

接口方式：

1. 创建接口文件，文件名称和映射文件一致UserMapper.java
2. 这个文件必须放在和UserMapper.xml映射文件同样路径中cn.tedu.mybaits.pojo
3. 创建接口的方法，把映射文件中的某个标签<select id=”find”>。照着它写

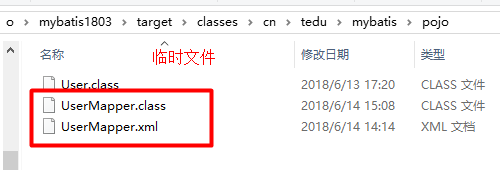
public List<User> find(User user);

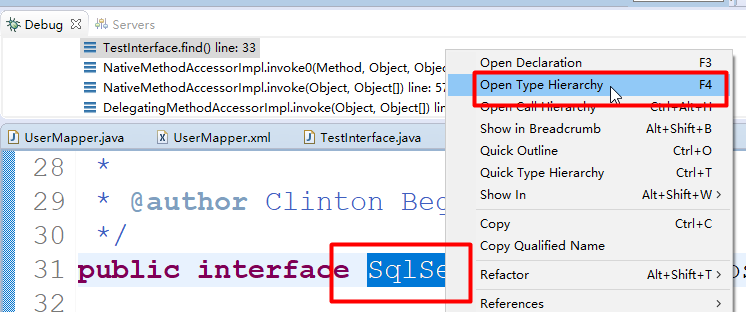
1. 测试UserMapper mapper = session.getMapper(UserMapper.class);
2. List<User> userList = mapper.find(user)

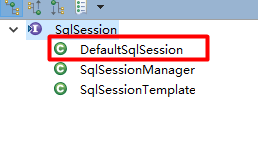
接口能调用吗？执行方法吗？

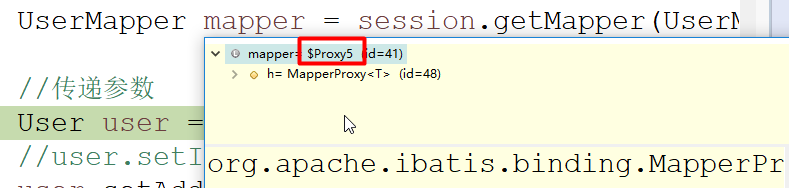
接口，规范类的行为。团队开发。

不能执行！只是方法的声明就没有执行者！实现类。









动态代理，它能实现动态创建类。$Proxy jdk动态代理，指定一个接口

实现类动态代理动态创建的类，它又怎么实现find方法呢？

结论：产生动态类其实是把之前mybatis方式包装一下，还是调用的它的原始的方法。

接口方式按面向对象方式来调用，内部还是走的原有流程。

原始方法找到namespace，还要找到id方法

cn.tedu.mybatis.pojo.UserMapper

全路径=包名.接口名称

等价于

包名=映射文件所在路径，找到映射文件目录，定位唯一文件

接口的名称=映射文件名称，找到映射文件了

接口实现proxy对象，调用find方法，mapper.find(user);根据执行方法名find唯一定位某个标签

1. 复杂ResultMap,Mybatis核心，ORM

需求：

数据库表中USER\_NAME，全大写（oracle）user\_name（mysql）

把它内容映射到pojo对象中Person，规则：驼峰规则

resultType

最终mybatis封装的数据是表中的数据还是查询的结果集？

小结：

1. 动态SQL语句

Mybatis提供很多动态标签

常用标签：<if>判断；<where>组织where语句，删除and或者or头；<set>删除最后一个逗号，<foreach>用在in子查询

<sql><include>定义变量，这个变量可以xml文件中多次引用

用的最多查询sql语句，where条件

1. 接口方式

传统方式调用不是面向对象方式，session.selectList

接口方式

1. 创建接口
2. 接口的方法，完全根据UserMapper.xml映射文件中标签来决定

<select id=”find” parameterType=”User” resultType=”User”>

public List<User> find(User user);

1. 接口我们没有自己去实现这个实现类，但是sqlSession.getMapper（UserMapper.class）方法它给我们实现了一个代理类$ProxyN（$Proxy5）这个利用jdk动态代理，作为接口的实现类。
2. 内部是去找对应UserMapepr.xml的id的方法，把参数传递给它，把结果封装返回对象。
3. 执行还是传统方式

接口方式更贴近面向对象思考方式，

Mybatis半ORM映射框架（半后面还是SQL语句，结构化查询语言，面向过程思考）

接口之后映射完成都把数据放在pojo对象中。

1. ResultMap

ResultType和ResultMap有什么差异？

ResulType针对简单变量string/integer/double…/pojo

ResultMap中间多了一层映射，特殊数据库字段就和pojo驼峰规则属性之间有了一个映射规则。

对象关联：在mybatis世界简单：对一和对多