**每日作业卷**

**MyBatis第1天**

极客营

# 关卡1

## 训练案例1

### 训练描述

使用课堂资料和已经提供的工程MyBatis02-study实现以下功能

1. 创建包装pojo，包含属性类型为user属性，使用包装pojo实现根据用户名模糊查询

2. 实现查询用户表的数据条数

3. 实现查询订单所有数据功能，并使用resultMap解决用户id没有获取的问题

4. 使用用户性别和名字查询用户，并使用if标签、where标签。

5. 使用sql标签提取Maper.xml中的相同的sql片段

6. 使用foreach标签实现使用多个id查询用户信息

7. 使用resultMap配置，实现订单-用户的一对一查询

8. 使用resultMap配置，实现用户-订单的一对多查询

### 操作步骤答案

1. 编写包装类QueryVo

**public** **class** QueryVo {

**private** User user;

**private** List<Integer> ids;

get/set。。。

}

1. 编写UserMapper接口

**public** **interface** UserMapper {

/\*\*

\* 根据包装pojo查询用户

\*

\* **@param** queryVo

\* **@return**

\*/

**public** List<User> queryUserByQueryVo(QueryVo queryVo);

/\*\*

\* 查询数据库数据条数

\*

\* **@return**

\*/

**public** **int** queryUserCount();

/\*\*

\* 根据用户名和性别查询

\*

\* **@param** user

\* **@return**

\*/

**public** List<User> queryUserBySexAndUsername(User user);

/\*\*

\* 测试foreach标签，解析集合

\*

\* **@param** queryVo

\* **@return**

\*/

**public** List<User> queryUserByIds(QueryVo queryVo);

/\*\*

\* 一对多关联查询

\*

\* **@return**

\*/

**public** List<User> queryUserOrder();

}

1. 编写OrderMapper接口

**public** **interface** OrderMapper {

/\*\*

\* 查询订单的所有数据

\*

\* **@return**

\*/

**public** List<Order> queryOrderAll();

/\*\*

\* 查询订单一对一关联用户，使用ResultMap

\*

\* **@return**

\*/

**public** List<Order> queryOrderUser();

}

1. 编写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">

<mapper namespace=*"com.igeek.mybatis.mapper.UserMapper"*>

<!-- 根据包装pojo模糊查询用户 -->

<select id=*"queryUserByQueryVo"* parameterType=*"queryVo"*

resultType=*"user"*>

SELECT id,username,birthday,sex,address FROM `user` WHERE

username LIKE

'%${user.username}%'

</select>

<!-- 查询数据库数据条数，返回的是简单数据类型 -->

<select id=*"queryUserCount"* resultType=*"int"*>

SELECT COUNT(\*) FROM

`user`

</select>

<!-- 声明sql片段 -->

<sql id=*"userField"*>

id,username,birthday,sex,address

</sql>

<!-- 根据用户的性别和名字查询用户 -->

<select id=*"queryUserBySexAndUsername"* parameterType=*"user"*

resultType=*"user"*>

SELECT

<include refid=*"userField"* />

FROM `user`

<where>

<if test=*"sex != null and sex != ''"*>

AND sex=#{sex}

</if>

<if test=*"username != null and username != ''"*>

AND username LIKE '%${username}%'

</if>

</where>

</select>

<!-- 使用List参数类型，动态sql之foreach标签 -->

<select id=*"queryUserByIds"* parameterType=*"queryVo"* resultType=*"user"*>

SELECT id,username,birthday,sex,address FROM `user` WHERE

<foreach collection=*"ids"* item=*"item"* open=*"id in ("* close=*")"*

separator=*","*>

#{item}

</foreach>

</select>

<!-- 配置resultMap实现一对多关联 -->

<resultMap type=*"user"* id=*"UserOrderResultMap"*>

<id column=*"id"* property=*"id"* />

<result column=*"username"* property=*"username"* />

<result column=*"birthday"* property=*"birthday"* />

<result column=*"sex"* property=*"sex"* />

<result column=*"address"* property=*"address"* />

<collection property=*"orders"* javaType=*"list"* ofType=*"order"*>

<id column=*"oid"* property=*"id"* />

<result column=*"number"* property=*"number"* />

<result column=*"createtime"* property=*"createtime"* />

<result column=*"note"* property=*"note"* />

</collection>

</resultMap>

<!-- 一对多查询 -->

<select id=*"queryUserOrder"* resultMap=*"UserOrderResultMap"*>

SELECT

u.id,

u.username,

u.birthday,

u.sex,

u.address,

o.id oid,

o.number,

o.createtime,

o.note

FROM

`user` u

LEFT JOIN `order` o ON u.id = o.user\_id

</select>

</mapper>

1. 编写OrderMapper.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">

<mapper namespace=*"com.igeek.mybatis.mapper.OrderMapper"*>

<!-- 配置ResultMap -->

<resultMap type=*"order"* id=*"queryOrderAll"*>

<id column=*"id"* property=*"id"* javaType=*"int"* />

<!-- 配置普通属性使用Result标签 -->

<result column=*"user\_id"* property=*"userId"* />

<result column=*"number"* property=*"number"* />

<result column=*"createtime"* property=*"createtime"* />

<result column=*"note"* property=*"note"* />

</resultMap>

<!-- 查询所有的订单数据 -->

<select id=*"queryOrderAll"* resultMap=*"queryOrderAll"*>

SELECT id, user\_id,

number,

createtime, note FROM `order`

</select>

<resultMap type=*"order"* id=*"orderUser"*>

<!-- 配置id -->

<id column=*"id"* property=*"id"* />

<result column=*"user\_id"* property=*"userId"* />

<result column=*"number"* property=*"number"* />

<result column=*"createtime"* property=*"createtime"* />

<result column=*"note"* property=*"note"* />

<association property=*"user"* javaType=*"user"*>

<!-- 配置一对一的对象的主键 -->

<id column=*"user\_id"* property=*"id"* />

<!-- 配置一对一的对象的普通属性 -->

<result column=*"username"* property=*"username"* />

<result column=*"address"* property=*"address"* />

</association>

</resultMap>

<!-- 一对一，查询订单表关联用户数据 -->

<select id=*"queryOrderUser"* resultMap=*"orderUser"*>

SELECT

o.id,

o.user\_id,

o.number,

o.createtime,

o.note,

u.username,

u.address

FROM

`order` o

LEFT

JOIN `user` u ON o.user\_id = u.id

</select>

</mapper>

1. 编写测试类UserMapperTest

**public** **class** UserMapperTest {

**private** SqlSessionFactory sqlSessionFactory;

@Before

**public** **void** setUp() **throws** Exception {

SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();

InputStream inputStream = Resources.*getResourceAsStream*("SqlMapConfig.xml");

**this**.sqlSessionFactory = builder.build(inputStream);

}

@Test

**public** **void** testQueryUserByQueryVo() {

// 获取sqlSession

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

// 获取Mapper动态代理对象

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

// 执行新增

User user = **new** User();

user.setUsername("张");

QueryVo queryVo = **new** QueryVo();

queryVo.setUser(user);

List<User> list = userMapper.queryUserByQueryVo(queryVo);

**for** (User u : list) {

System.***out***.println(u);

}

// 释放资源

sqlSession.close();

}

@Test

**public** **void** testQueryUserCount() {

// 获取sqlSession

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

// 获取Mapper动态代理对象

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

// 执行新增

**int** count = userMapper.queryUserCount();

System.***out***.println(count);

// 释放资源

sqlSession.close();

}

@Test

**public** **void** testQueryUserBySexAndUsername() {

// 获取sqlSession

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

// 获取Mapper动态代理对象

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

// 执行新增

User user = **new** User();

user.setSex("1");

user.setUsername("关");

List<User> list = userMapper.queryUserBySexAndUsername(user);

**for** (User u : list) {

System.***out***.println(u);

}

// 释放资源

sqlSession.close();

}

@Test

**public** **void** testQueryUserByIds() {

// 获取sqlSession

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

// 获取Mapper动态代理对象

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

// 执行新增

List<Integer> ids = **new** ArrayList<>();

ids.add(27);

ids.add(28);

ids.add(31);

ids.add(32);

QueryVo queryVo = **new** QueryVo();

queryVo.setIds(ids);

List<User> list = userMapper.queryUserByIds(queryVo);

**for** (User u : list) {

System.***out***.println(u);

}

// 释放资源

sqlSession.close();

}

@Test

**public** **void** testQueryUserOrder() {

// 获取sqlSession

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

// 获取Mapper动态代理对象

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

// 执行新增

List<User> list = userMapper.queryUserOrder();

// 释放资源

sqlSession.close();

}

}

1. 编写测试类OrderMapperTest

**public** **class** OrderMapperTest {

**private** SqlSessionFactory sqlSessionFactory;

@Before

**public** **void** setUp() **throws** Exception {

SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();

InputStream inputStream = Resources.*getResourceAsStream*("SqlMapConfig.xml");

**this**.sqlSessionFactory = builder.build(inputStream);

}

@Test

**public** **void** testQueryOrderAll() {

// 获取OrderMapper

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

OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.**class**);

List<Order> list = orderMapper.queryOrderAll();

**for** (Order order : list) {

System.***out***.println(order);

}

sqlSession.close();

}

}

## 训练案例2

### 训练描述

使用课堂的数据库表创建新工程，实现MyBatis和Spring的整合

并使用接口动态代理的方式实现DAO的开发，

1.1 根据id查询用户

1.2 根据用户名模糊查询用户

1.3 新增用户"

### 操作步骤答案

1. 创建工程mybatis-spring，导入课堂资料提供的整合jar包

2. 配置实现MyBatis和Spring的整合

编写配置文件SqlMapConfig.xml

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

</configuration>

编写配置文件applicationContext.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>

<!-- 加载配置文件 -->

<context:property-placeholder location=*"classpath:db.properties"* />

<!-- 数据库连接池 -->

<bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"*

destroy-method=*"close"*>

<property name=*"driverClassName"* value=*"${jdbc.driver}"* />

<property name=*"url"* value=*"${jdbc.url}"* />

<property name=*"username"* value=*"${jdbc.username}"* />

<property name=*"password"* value=*"${jdbc.password}"* />

<property name=*"maxActive"* value=*"10"* />

<property name=*"maxIdle"* value=*"5"* />

</bean>

<!-- 配置SqlSessionFactory -->

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<!-- 配置mybatis核心配置文件 -->

<property name=*"configLocation"* value=*"classpath:SqlMapConfig.xml"* />

<!-- 配置数据源 -->

<property name=*"dataSource"* ref=*"dataSource"* />

</bean>

</beans>

加入配置文件db.properties

jdbc.driver=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8

jdbc.username=root

jdbc.password=root

加入配置文件log4j.properties

# Global logging configuration

log4j.rootLogger=DEBUG, stdout

# Console output...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

3. 使用接口动态代理的方式实现DAO的开发，

1.1 根据id查询用户

1.2 根据用户名模糊查询用户

1.3 新增用户"

编写User

**public** **class** User {

**private** **int** id;

**private** String username;// 用户姓名

**private** String sex;// 性别

**private** Date birthday;// 生日

**private** String address;// 地址

get/set。。。

}

编写UserMapper接口

**public** **interface** UserMapper {

/\*\*

\* 根据用户id查询

\*

\* **@param** id

\* **@return**

\*/

User queryUserById(**int** id);

/\*\*

\* 根据用户名模糊查询用户

\*

\* **@param** username

\* **@return**

\*/

List<User> queryUserByUsername(String username);

/\*\*

\* 添加用户

\*

\* **@param** user

\*/

**void** saveUser(User user);

}

编写UserMapper.xml配置文件

<?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.igeek.mybatis.mapper.UserMapper"*>

<!-- 根据用户id查询 -->

<select id=*"queryUserById"* parameterType=*"int"*

resultType=*"com.igeek.mybatis.pojo.User"*>

select \* from user where id = #{id}

</select>

<!-- 根据用户名模糊查询用户 -->

<select id=*"queryUserByUsername"* parameterType=*"string"*

resultType=*"com.igeek.mybatis.pojo.User"*>

select \* from user where username like '%${value}%'

</select>

<!-- 添加用户 -->

<insert id=*"saveUser"* parameterType=*"com.igeek.mybatis.pojo.User"*>

<selectKey keyProperty=*"id"* keyColumn=*"id"* order=*"AFTER"*

resultType=*"int"*>

select last\_insert\_id()

</selectKey>

insert into user

(username,birthday,sex,address) values

(#{username},#{birthday},#{sex},#{address})

</insert>

</mapper>

修改配置文件applicationContext.xml，添加如下配置

<!-- Mapper扫描包方式配置 -->

<bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>

<!-- 配置Mapper接口 -->

<property name=*"basePackage"* value=*"com.igeek.mybatis.mapper"* />

</bean>

编写测试方法

**public** **class** UserMapperTest {

**private** ApplicationContext context;

@Before

**public** **void** setUp() **throws** Exception {

**this**.context = **new** ClassPathXmlApplicationContext("classpath:applicationContext.xml");

}

@Test

**public** **void** testQueryUserById() {

// 获取Mapper

UserMapper userMapper = **this**.context.getBean(UserMapper.**class**);

User user = userMapper.queryUserById(1);

System.***out***.println(user);

}

@Test

**public** **void** testQueryUserByUsername() {

// 获取Mapper

UserMapper userMapper = **this**.context.getBean(UserMapper.**class**);

List<User> list = userMapper.queryUserByUsername("张");

**for** (User user : list) {

System.***out***.println(user);

}

}

@Test

**public** **void** testSaveUser() {

// 获取Mapper

UserMapper userMapper = **this**.context.getBean(UserMapper.**class**);

User user = **new** User();

user.setUsername("曹操");

user.setSex("1");

user.setBirthday(**new** Date());

user.setAddress("三国");

userMapper.saveUser(user);

System.***out***.println(user);

}

}

# 关卡2

## 训练案例1

### 训练描述

在关卡1的基础上，

实现MyBatis和Spring的整合开发

要求：

1. 实现查询订单表的数据条数

2. 实现查询所有订单功能，并使用resultMap解决用户id没有获取的问题

3. 使用用户性别和名字查询用户，并使用if标签、where标签。

4. 使用sql标签提取Maper.xml中的相同的sql片段

5. 使用foreach标签实现使用多个id查询用户信息

6. 使用自定义pojo的方式，实现订单-用户的一对一查询

7. 使用resultMap配置，实现订单-用户的一对一查询

8. 使用resultMap配置，实现用户-订单的一对多查询"

### 操作步骤答案

1. 编写pojo

**public** **class** User {

**private** **int** id;

**private** String username;// 用户姓名

**private** String sex;// 性别

**private** Date birthday;// 生日

**private** String address;// 地址

**private** List<Order> orders;

set/get。。。

}

**public** **class** Order {

**private** Integer id;

**private** Integer userId;

**private** String number;

**private** Date createtime;

**private** String note;

// 一对一关联用户

**private** User user;

set/get。。。

}

**public** **class** QueryVo {

**private** List<Integer> ids;

set/get。。。

}

**public** **class** OrderUser **extends** Order {

**private** String username;

**private** String address;

set/get。。。

}

1. 编写UserMapper接口

**public** **interface** UserMapper {

/\*\*

\* 根据用户id查询

\*

\* **@param** id

\* **@return**

\*/

User queryUserById(**int** id);

/\*\*

\* 根据用户名模糊查询用户

\*

\* **@param** username

\* **@return**

\*/

List<User> queryUserByUsername(String username);

/\*\*

\* 添加用户

\*

\* **@param** user

\*/

**void** saveUser(User user);

// 使用用户性别和名字查询用户，并使用if标签、where标签

**public** List<User> queryUserBySexAndUsername(User user);

// 使用foreach标签实现使用多个id查询用户信息

**public** List<User> queryUserByIds(QueryVo queryVo);

// 使用resultMap配置，实现用户-订单的一对多查询"

**public** List<User> queryUserOrder();

}

1. 编写OrderMapper接口

**public** **interface** OrderMapper {

// 实现查询订单表的数据条数

**public** **int** queryOrderCount();

// 实现查询所有订单功能，并使用resultMap解决用户id没有获取的问题

**public** List<Order> queryOrderAll();

// 使用自定义pojo的方式，实现订单-用户的一对一查询

**public** List<OrderUser> queryOrderUser();

// 使用resultMap配置，实现订单-用户的一对一查询

**public** List<Order> queryOrder();

}

1. 编写UserMapper.xml配置

<?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.igeek.mybatis.mapper.UserMapper"*>

<!-- 根据用户id查询 -->

<select id=*"queryUserById"* parameterType=*"int"*

resultType=*"com.igeek.mybatis.pojo.User"*>

select \* from user where id = #{id}

</select>

<!-- 根据用户名模糊查询用户 -->

<select id=*"queryUserByUsername"* parameterType=*"string"*

resultType=*"com.igeek.mybatis.pojo.User"*>

select \* from user where username like '%${value}%'

</select>

<!-- 添加用户 -->

<insert id=*"saveUser"* parameterType=*"com.igeek.mybatis.pojo.User"*>

<selectKey keyProperty=*"id"* keyColumn=*"id"* order=*"AFTER"*

resultType=*"int"*>

select last\_insert\_id()

</selectKey>

insert into user

(username,birthday,sex,address) values

(#{username},#{birthday},#{sex},#{address})

</insert>

<!-- 使用用户性别和名字查询用户，并使用if标签、where标签 -->

<!-- public List<User> queryUserBySexAndUsername(User user); -->

<select id=*"queryUserBySexAndUsername"* parameterType=*"com.igeek.mybatis.pojo.User"*

resultType=*"com.igeek.mybatis.pojo.User"*>

SELECT

<include refid=*"userField"* />

FROM `user`

<where>

<if test=*"sex != null and sex != ''"*>

AND sex=#{sex}

</if>

<if test=*"username != null and username != ''"*>

AND username LIKE '%${username}%'

</if>

</where>

</select>

<!-- 使用sql标签提取Maper.xml中的相同的sql片段 -->

<sql id=*"userField"*>

id,username,birthday,sex,address

</sql>

<!-- 使用foreach标签实现使用多个id查询用户信息 -->

<!-- public List<User> queryUserByIds(QueryVo queryVo); -->

<select id=*"queryUserByIds"* parameterType=*"com.igeek.mybatis.pojo.QueryVo"*

resultType=*"com.igeek.mybatis.pojo.User"*>

SELECT id,username,birthday,sex,address FROM `user` WHERE

<foreach collection=*"ids"* item=*"item"* open=*"id in ("* close=*")"*

separator=*","*>

#{item}

</foreach>

</select>

<!-- 配置resultMap实现一对多关联 -->

<resultMap type=*"com.igeek.mybatis.pojo.User"* id=*"UserResultMap"*>

<id column=*"id"* property=*"id"* />

<result column=*"username"* property=*"username"* />

<result column=*"birthday"* property=*"birthday"* />

<result column=*"sex"* property=*"sex"* />

<result column=*"address"* property=*"address"* />

<collection property=*"orders"* javaType=*"list"*

ofType=*"com.igeek.mybatis.pojo.Order"*>

<id column=*"oid"* property=*"id"* />

<result column=*"number"* property=*"number"* />

<result column=*"createtime"* property=*"createtime"* />

<result column=*"note"* property=*"note"* />

</collection>

</resultMap>

<!-- 使用resultMap配置，实现用户-订单的一对多查询" -->

<!-- public List<User> queryUserOrder(); -->

<select id=*"queryUserOrder"* resultMap=*"UserResultMap"*>

SELECT

u.id,

u.username,

u.birthday,

u.sex,

u.address,

o.id oid,

o.number,

o.createtime,

o.note

FROM

`user` u

LEFT JOIN `order` o ON u.id = o.user\_id

</select>

</mapper>

1. 编写OrderMapper.xml配置

<?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.igeek.mybatis.mapper.OrderMapper"*>

<!-- 实现查询订单表的数据条数 -->

<!-- public int queryOrderCount(); -->

<select id=*"queryOrderCount"* resultType=*"int"*>

SELECT COUNT(\*) FROM

`order`

</select>

<!-- 配置ResultMap -->

<resultMap type=*"com.igeek.mybatis.pojo.Order"* id=*"queryOrderAll"*>

<id column=*"id"* property=*"id"* javaType=*"int"* />

<!-- 配置普通属性使用Result标签 -->

<result column=*"user\_id"* property=*"userId"* />

<result column=*"number"* property=*"number"* />

<result column=*"createtime"* property=*"createtime"* />

<result column=*"note"* property=*"note"* />

</resultMap>

<!-- 实现查询所有订单功能，并使用resultMap解决用户id没有获取的问题 -->

<!-- public List<Order> queryOrderAll(); -->

<select id=*"queryOrderAll"* resultMap=*"queryOrderAll"*>

SELECT id, user\_id,

number,

createtime, note FROM `order`

</select>

<!-- 使用自定义pojo的方式，实现订单-用户的一对一查询 -->

<!-- public List<OrderUser> queryOrderUser(); -->

<select id=*"queryOrderUser"* resultType=*"com.igeek.mybatis.pojo.OrderUser"*>

SELECT

o.id,

o.user\_id

userId,

o.number,

o.createtime,

o.note,

u.username,

u.address

FROM

`order` o

LEFT JOIN `user` u ON o.user\_id = u.id

</select>

<resultMap type=*"com.igeek.mybatis.pojo.Order"* id=*"orderResultMap"*>

<!-- 配置id -->

<id column=*"id"* property=*"id"* />

<result column=*"user\_id"* property=*"userId"* />

<result column=*"number"* property=*"number"* />

<result column=*"createtime"* property=*"createtime"* />

<result column=*"note"* property=*"note"* />

<association property=*"user"* javaType=*"com.igeek.mybatis.pojo.User"*>

<!-- 配置一对一的对象的主键 -->

<id column=*"user\_id"* property=*"id"* />

<!-- 配置一对一的对象的普通属性 -->

<result column=*"username"* property=*"username"* />

<result column=*"address"* property=*"address"* />

</association>

</resultMap>

<!-- 使用resultMap配置，实现订单-用户的一对一查询 -->

<!-- public List<Order> queryOrder(); -->

<select id=*"queryOrder"* resultMap=*"orderResultMap"*>

SELECT

o.id,

o.user\_id,

o.number,

o.createtime,

o.note,

u.username,

u.address

FROM

`order` o

LEFT

JOIN `user` u ON o.user\_id = u.id

</select>

</mapper>

1. 编写UserMapperTest测试类

**public** **class** UserMapperTest {

**private** ApplicationContext context;

@Before

**public** **void** setUp() **throws** Exception {

**this**.context = **new** ClassPathXmlApplicationContext("classpath:applicationContext.xml");

}

@Test

**public** **void** testQueryUserById() {

// 获取Mapper

UserMapper userMapper = **this**.context.getBean(UserMapper.**class**);

User user = userMapper.queryUserById(1);

System.***out***.println(user);

}

@Test

**public** **void** testQueryUserByUsername() {

// 获取Mapper

UserMapper userMapper = **this**.context.getBean(UserMapper.**class**);

List<User> list = userMapper.queryUserByUsername("张");

**for** (User user : list) {

System.***out***.println(user);

}

}

@Test

**public** **void** testSaveUser() {

// 获取Mapper

UserMapper userMapper = **this**.context.getBean(UserMapper.**class**);

User user = **new** User();

user.setUsername("曹操");

user.setSex("1");

user.setBirthday(**new** Date());

user.setAddress("三国");

userMapper.saveUser(user);

System.***out***.println(user);

}

@Test

**public** **void** testQueryUserBySexAndUsername() {

// 获取Mapper

UserMapper userMapper = context.getBean(UserMapper.**class**);

User user = **new** User();

user.setSex("1");

user.setUsername("关");

List<User> list = userMapper.queryUserBySexAndUsername(user);

**for** (User u : list) {

System.***out***.println(u);

}

}

@Test

**public** **void** testQueryUserByIds() {

// 获取Mapper

UserMapper userMapper = context.getBean(UserMapper.**class**);

List<Integer> ids = **new** ArrayList<>();

ids.add(27);

ids.add(28);

ids.add(31);

ids.add(32);

QueryVo queryVo = **new** QueryVo();

queryVo.setIds(ids);

List<User> list = userMapper.queryUserByIds(queryVo);

**for** (User u : list) {

System.***out***.println(u);

}

}

@Test

**public** **void** testQueryUserOrder() {

// 获取Mapper

UserMapper userMapper = context.getBean(UserMapper.**class**);

List<User> list = userMapper.queryUserOrder();

}

}

1. 编写OrderMapperTest测试类

**public** **class** OrderMapperTest {

**private** ApplicationContext context;

@Before

**public** **void** setUp() **throws** Exception {

**this**.context = **new** ClassPathXmlApplicationContext("classpath:applicationContext.xml");

}

@Test

**public** **void** testQueryOrderCount() {

// 获取OrderMapper

OrderMapper orderMapper = context.getBean(OrderMapper.**class**);

**int** count = orderMapper.queryOrderCount();

System.***out***.println(count);

}

@Test

**public** **void** testQueryOrderAll() {

// 获取OrderMapper

OrderMapper orderMapper = context.getBean(OrderMapper.**class**);

List<Order> list = orderMapper.queryOrderAll();

**for** (Order order : list) {

System.***out***.println(order);

}

}

@Test

**public** **void** testQueryOrderUser() {

// 获取OrderMapper

OrderMapper orderMapper = context.getBean(OrderMapper.**class**);

List<OrderUser> list = orderMapper.queryOrderUser();

}

@Test

**public** **void** testQueryOrder() {

// 获取OrderMapper

OrderMapper orderMapper = context.getBean(OrderMapper.**class**);

List<Order> list = orderMapper.queryOrder();

}

}

## 训练案例2

### 训练描述

使用课堂提供的逆向工程生成代码

### 操作步骤答案

1. 导入逆向工程到工作空间

2. 修改逆向工程中的配置文件

3. 执行main函数生成代码

4. 把生成的代码复制到到MyBatis整合Spring的工程中。

# 关卡3

## 训练案例1

### 训练描述

使用前一天作业的tb\_order和user表，在mybatis-spring整合的工程中实现DAO的开发。

要求：

1. 实现查询tb\_order的数据条数

2. 使用包装对象，根据用户id查询该用户的所有已付款的订单

3. 根据订单金额payment查询订单，需要查询订单金额为某个范围的订单，需要一对一关联用户数据，需要使用自定义pojo和resultMap两种方式实现。

4. 根据id查询user，需要一对多关联tb\_order

5. 和spring整合使用直接配置Mapper代理对象的方式实现

6. 和spring整合使用Mapper扫描两种方式实现

### 答案

参考附件工程