# MyBatis第二天课堂笔记

# 输入映射和输出映射

## parameterType(输入类型)

### 传递简单类型

参考第一天内容

### 传递pojo对象

参考第一天内容

Mybatis使用ognl表达式解析对象字段的值，#{}或者${}括号中的值为pojo属性名称。

### 传递pojo包装对象

1. **新建包装pojo对象QueryVo**

/\*\*

\* 包装pojo

\* **@author** Steven

\*/

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

//用户对象

**private** User user;

**public** User getUser() {

**return** user;

}

**public** **void** setUser(User user) {

**this**.user = user;

}

}

1. **映射文件与sql**

<!-- 1、resultType:如果要返回数据集合，只需设定为每一个元素的数据类型

2、 包装的pojo取值通过 "."来获取

-->

<select id=*"getUserByQueryVo"* parameterType=*"queryvo"* resultType=*"com.itheima.mybatis.pojo.User"*>

<!-- SELECT \* FROM USER WHERE username LIKE #{name} -->

SELECT \* FROM USER WHERE username LIKE '%${**user.username**}%'

</select>

1. **新增接口方法**
2. **增加测试方法，完成测试**

## resultType(输出类型)

### 输出简单类型

<!-- 查询用户总记录数，演示返回简单类型 -->

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

SELECT COUNT(1) FROM USER

</select>

*其它步骤跟前面类似，添加接口方法与测试方法，完成测试。*

### 输出pojo对象

参考第一天内容

### 输出pojo列表

参考第一天内容。

## 输出resultMap

*演示基于完成订单列表的查询，由user\_id字段与pojo属性不一致时引出的resultMap。*

<!-- resultMap入门

type:映射成的pojo类型

id：resultMap唯一标识

-->

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

<!-- id标签用于绑定主键 -->

<!-- <id property="id" column="id"/> -->

<!-- 使用result绑定普通字段 -->

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

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

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

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

</resultMap>

<!-- 使用resultMap -->

<select id=*"getOrderListResultMap"* resultMap=*"orderMap"*>

SELECT \* FROM `order`

</select>

*其它步骤跟前面类似，添加接口方法与测试方法，完成测试。*

# 动态sql

## If

*演示基于完成用户列表查询功能，由多查询条件拼装引出if标签。*

<!-- 演示动态sql-if标签的使用情景 -->

<select id=*"getUserByWhere"* parameterType=*"user"* resultType=*"com.itheima.mybatis.pojo.User"*>

<!-- SELECT \* FROM USER WHERE username LIKE '%${username}%' and id = #{id} -->

SELECT \* FROM USER where 1 = 1

<!-- if标签的使用 -->

<if test=*"id != null"*>

and id = #{id}

</if>

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

and username LIKE '%${username}%'

</if>

</select>

*其它步骤跟前面类似，添加接口方法与测试方法，完成测试。*

## Where

*复制getUserByWhere修改一下，改名为getUserByWhere2。*

<!-- 演示动态sql-where标签的使用情景 -->

<select id=*"getUserByWhere2"* parameterType=*"user"*

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

<!-- include:引入sql片段,refid引入片段id -->

SELECT

\*

FROM USER

<!-- where会自动加上where同处理多余的and -->

<where>

<!-- if标签的使用 -->

<if test=*"id != null"*>

and id = #{id}

</if>

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

and username LIKE '%${username}%'

</if>

</where>

</select>

*其它步骤跟前面类似，添加接口方法与测试方法，完成测试。*

## Foreach

*复制getUserByWhere2修改一下，改名为getUserByIds。*

<!-- 演示动态sql-foreach标签的使用情景 -->

<select id=*"getUserByIds"* parameterType=*"queryvo"*

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

SELECT

\*

FROM USER

<!-- where会自动加上where同处理多余的and -->

<where>

<!-- id IN(1,10,25,30,34) -->

<!-- foreach循环标签

collection:要遍历的集合，来源入参

open:循环开始前的sql

separator:分隔符

close:循环结束拼接的sql

-->

<foreach item=*"uid"* collection=*"ids"* open=*"id IN("* separator=*","*

close=*")"*>

#{uid}

</foreach> </where>

</select>

*其它步骤跟前面类似，添加接口方法与测试方法，完成测试。*

## Sql片段

*演示通过select \* 不好引出查询字段名，抽取共用sql片段。*

1. 定义

<!-- sql片段 定义，id:片段唯一标识 -->

<sql id=*"user\_column"*>

`id`,

`username`,

`birthday`,

`sex`,

`address`,

`uuid2`

</sql>

1. 使用

SELECT

<!-- sql片段的使用：include:引入sql片段,refid引入片段id -->

<include refid=*"user\_column"* />

FROM USER

*其它步骤跟前面类似，添加接口方法与测试方法，完成测试。*

# 关联查询

## 一对一关联

### 方法一，使用resultType

1. 新建OrderUser的pojo，继承自Order。

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

**private** String username;

**private** String address;

…….get,set

}

1. 修改order的映射文件，新增查询方法getOrderUser。

<!-- 一对一关联查询，使用resultType -->

<select id=*"getOrderUser"* resultType=*"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 u.id = o.`user\_id`

</select>

*其它步骤跟前面类似，添加接口方法与测试方法，完成测试。*

### 方法二，使用resultMap

1. 改造order的pojo



1. 修改order的映射文件

<!-- 一对一关联查询-resultMap -->

<resultMap type=*"order"* id=*"order\_user\_map"*>

<!-- id标签用于绑定主键 -->

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

<!-- 使用result绑定普通字段 -->

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

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

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

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

<!-- association:配置一对一关联

property:绑定的用户属性

javaType:属性数据类型，支持别名

-->

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

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

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

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

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

</association>

</resultMap>

<!-- 一对一关联查询-使用resultMap -->

<select id=*"getOrderUser2"* resultMap=*"order\_user\_map"*>

SELECT

o.`id`,

o.`user\_id`,

o.`number`,

o.`createtime`,

o.`note`,

u.`username`,

u.`address`,

u.`sex`

FROM `order` o

LEFT JOIN `user` u

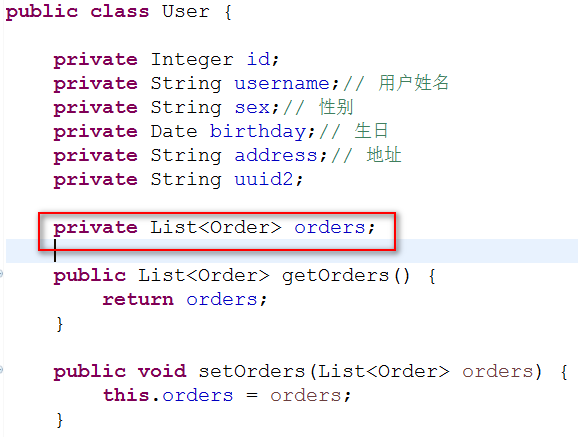
ON u.id = o.`user\_id`

</select>

*其它步骤跟前面类似，添加接口方法与测试方法，完成测试。*

## 一对多关联

1. 改造user的pojo



1. 修改user的映射文件

<!-- 一对多关联查询 -->

<resultMap type=*"user"* id=*"user\_order\_map"*>

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

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

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

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

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

<result property=*"uuid2"* column=*"uuid2"* />

<!-- collection:配置一对多关系

property:用户下的order属性

ofType:property的数据类型，支持别名

-->

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

<!-- id标签用于绑定主键 -->

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

<!-- 使用result绑定普通字段 -->

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

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

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

</collection>

</resultMap>

<!-- 一对多关联查询 -->

<select id=*"getUserOrder"* resultMap=*"user\_order\_map"*>

SELECT

u.`id`,

u.`username`,

u.`birthday`,

u.`sex`,

u.`address`,

u.`uuid2`,

o.`id` oid,

o.`number`,

o.`createtime`

FROM `user` u

LEFT JOIN `order` o

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

</select>

*其它步骤跟前面类似，添加接口方法与测试方法，完成测试。*

# Mybatis整合spring

## 整合思路

1. SqlSessionFactory对象应该放到spring容器中作为单例存在。
2. 传统dao的开发方式中，应该从spring容器中获得sqlsession对象。
3. Mapper代理形式中，应该从spring容器中直接获得mapper的代理对象。
4. 数据库的连接以及数据库连接池事务管理都交给spring容器来完成。

## 整合步骤

1. 创建一个java工程。
2. 导入jar包。（课前资料中mybatis与spring整合所有包）
3. mybatis的配置文件sqlmapConfig.xml
4. 编写Spring的配置文件
5. 数据库连接及连接池
6. sqlsessionFactory对象，配置到spring容器中
7. 编写Spring的配置文件
8. 复制jdbc.properties配置文件到新工程
9. 复制log4j.properties配置文件到新工程

## Dao开发

### 复制user的pojo到新工程

### 传统Dao开发

1. 复制user.xml到新工程，并修改，只留下要测试的三个方法
2. 在SqlMapConfig.xml加载user.xml
3. 复制UserDao接口到新工程，并修改，只留下要测试的三个方法
4. 编写UserDaoImpl实现类，关键是继承SqlSessionDaoSupport

**public** **class** UserDaoImpl **extends** SqlSessionDaoSupport **implements** UserDao {

@Override

**public** User getUserById(Integer id) {

SqlSession sqlSession = **super**.getSqlSession();

//查询用户

User user = sqlSession.selectOne("user.getUserById", id);

//不能关闭SqlSession

//sqlSession.close();

**return** user;

}

@Override

**public** List<User> getUserByUserName(String name) {

SqlSession sqlSession = **super**.getSqlSession();

List<User> list = sqlSession.selectList("user.getUserByName", name);

//不能关闭SqlSession

**return** list;

}

@Override

**public** **void** insertUser(User user) {

SqlSession sqlSession = **super**.getSqlSession();

sqlSession.insert("user.insertUser", user);

//不用手动提交事务，交给spring

}

}

1. 在applicationContext.xml中配置UserDaoImpl实现类

<!-- 传统dao -->

<bean class=*"com.itheima.mybatis.dao.impl.UserDaoImpl"*>

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

</bean>

1. 编写测试类，新建单完测试类

**public** **class** UserDaoTest {

**private** ApplicationContext applicationContext;

@Before

**public** **void** init(){

applicationContext = **new** ClassPathXmlApplicationContext("applicationContext.xml");

}

@Test

**public** **void** testGetUserById() {

UserDao userDao = applicationContext.getBean(UserDao.**class**);

User user = userDao.getUserById(10);

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

}

………省略其它方法

}

### Mapper代理模式开发Dao

1. 复制UserMapper.xml到新工程，并修改，只留下要测试的三个方法
2. 复制UserMapper接口到新工程，并修改，只留下要测试的三个方法
3. 配置Mapper
4. 单个接口配置MapperFactoryBean

<!-- 动态代理Dao开发，第一种方式 -MapperFactoryBean -->

<bean id=*"baseMapper"* class=*"org.mybatis.spring.mapper.MapperFactoryBean"* abstract=*"true"* lazy-init=*"true"*>

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

</bean>

<!-- 用户动态代理扫描 -->

<bean parent=*"baseMapper"*>

<property name=*"mapperInterface"* value=*"com.itheima.mybatis.mapper.UserMapper"* />

</bean>

1. 配置包扫描器

<!-- 动态代理Dao开发，第一种方式，包扫描器(推荐使用) -->

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

<!-- basePackage：配置映射包装扫描，多个包时用","或";"分隔 -->

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

</bean>

1. 测试

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

**private** ApplicationContext applicationContext;

@Before

**public** **void** init(){

applicationContext = **new** ClassPathXmlApplicationContext("applicationContext.xml");

}

@Test

**public** **void** testGetUserById() {

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

User user = userMapper.getUserById(10);

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

}

# Mybatis逆向工程

注意的点：在generatorConfig.xml中配置mapper生成的详细信息，注意改下几点：

1. 添加要生成的数据库表
2. po文件所在包路径
3. mapper文件所在包路径

*其余的可以参考教案。*