# 尚马教育 JAVA 课程

# Mybatis高级

文档编号：C02

创建日期： 2017-07-07

最后修改日期：2021-01-18

版 本 号：V3.5

电子版文件名：尚马教育-第三阶段-2.mybatis高级.docx

**文档修改记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| 更新日期 | 更新作者 | 更新说明 | 版本号 |
| 2017-07-30 | 张元林 | 初始版本 | V1.0 |
| 2018-08-01 | 王绍成 | Mybatis版本更新 | V2.0 |
| 2019-08-09 | 冯勇涛 | 课件格式以及课程深度加深 | V3.0 |
| 2021-01-18 | 冯勇涛 | 深化内容，突出重点 | V3.5 |

目录

[尚马教育 JAVA 课程 1](#_Toc20531)

[Mybatis高级 1](#_Toc5816)

[1. Sql标签 3](#_Toc30604)

[2. resultMap标签进行属性名映射 5](#_Toc22421)

[2.1. 使用场景 5](#_Toc4319)

[2.2. resultMap示例 5](#_Toc30208)

[2.3. 应用resultMap 5](#_Toc31759)

[3. resultMap标签进行对象关系映射 7](#_Toc20540)

[3.1. 多对一持有关系 7](#_Toc19606)

[3.1.1. 创建实体类与数据库表 7](#_Toc10518)

[3.1.2. 实现思路1:手工进行多次查询 8](#_Toc10849)

[3.1.3. 实现思路2:association标签进行多次查询 9](#_Toc32716)

[3.1.4. 实现思路3:association标签对级联sql查询结果映射 10](#_Toc23190)

[3.2. 一对多聚合关系 11](#_Toc17484)

[3.2.1. 修改实体类 12](#_Toc14172)

[3.2.2. 实现思路1:手工进行多次查询 12](#_Toc12870)

[3.2.3. 实现思路2:collection标签进行多次查询 13](#_Toc22308)

[3.2.4. 实现思路3:collection标签对级联sql查询结果映射 14](#_Toc12772)

[4. 动态SQL语句 15](#_Toc26376)

[4.1. If与where标签 16](#_Toc21283)

[4.2. if与Set标签 16](#_Toc6447)

[4.3. Foreach标签 17](#_Toc24870)

[4.3.1. 批量删除 18](#_Toc20321)

[4.3.2. 批量插入 19](#_Toc13565)

[4.3.3. 批量查询 19](#_Toc11056)

[4.4. Choose-when-otherwise 19](#_Toc27568)

[5. 数据延迟加载 20](#_Toc14426)

[5.1. Mybatis延迟加载使用 20](#_Toc397)

[6. Mybatis缓存 21](#_Toc19284)

[6.1. 一级缓存 21](#_Toc9486)

[6.2. 二级缓存 22](#_Toc15649)

[① Xml核心配置中开启二级缓存 22](#_Toc26069)

[② 映射文件配置 22](#_Toc14045)

[7. 代理模式 23](#_Toc18570)

[7.1. 静态代理 23](#_Toc4773)

[7.2. 动态代理 25](#_Toc20126)

[8. 总结 28](#_Toc17605)

[9. mybatis总体任务 28](#_Toc14661)

|  |
| --- |
| 本章节学习内容：  深入学习mybatis映射文件中的各个标签用法；  理解静态代理与动态代理思想；  了解mybatis数据延迟加载思想；  了解mybatis数据查询缓存思想； |

## Sql标签

Sql标签作用：可被其他语句引用的可重用语句块。

实际使用中通常把查询字段包装为sql语句块。

新建映射文件sysuserMapper.xml

|  |
| --- |
| <**sql id="allFields"**>  uid,uname,uphone,upwd,uemail,create\_time,update\_time </**sql**> <**select id="selectUserByKey" parameterType="int" resultType="Sysuser"**>  select  <**include refid="allFields"**></**include**>  from sysuser where uid=#{uid} </**select**> |

## resultMap标签进行属性名映射

### 使用场景

ResultMap标签使用起来偏复杂，用来对实体对象的属性与数据库字段名做映射，同时提供子标签做对象之间的关系映射,进行级联查询操作。

ResultMap 的设计思想是，对于简单的语句不需要配置显式的结果映射(属性名与字段名一致)，而对于复杂一点的语句只需要描述它们的关系就行了。

### resultMap示例

|  |
| --- |
| <resultMap type=*"com.javasm.sys.entity.Sysuser"* id=*"sysuserResultMap"*>  <id column=*"uid"* property=*"uid"*/><!-- 对主键列做映射 -->  <result column=*"uname"* property=*"uname"*/>  <result column=*"uphone"* property=*"uphone"*/>  **<result column="user\_address" property="uaddress"/>**  <**result column="create\_time" property="createTime"**></**result**>  <**result column="update\_time" property="updateTime"**></**result**>  </resultMap> |

### 应用resultMap

|  |
| --- |
| <!--select标签的resultMap属性的值对应resultMap标签的id-->  <select id=*"selectUserById"* parameterType=*"int"* resultMap=*"sysuserResultMap"*>  select \* from sysuser where uid=#{uid}  </select> |

注意点：

注意点1：resultMap标签的id子标签用来进行主键列映射；result子标签进行非主键列映射；

注意点2：数据库字段与实体类属性同名的字段可以不映射,不过一般都配置。

注意点3：select标签的resultMap属性与resultType属性不能同时出现，resultType属性的值一般是实体类，map或简单类型；resultMap属性的值必须是resultMap标签的id值。

## resultMap标签进行对象关系映射

在实际开发中，经常出现多表联接查询的情况，使用sql交叉，左右联接，子查询等复杂的sql语句，mybatis对类似的复杂sql语句查询结果进行封装。

### 多对一持有关系

用户与角色对象之间维护成用户持有角色关系，查询用户时，把用户关联的角色信息同步查询。

#### 创建实体类与数据库表

|  |
| --- |
| **public class** Sysrole {  **private** Integer rid;  **private** String rname;  **private** String createTime;  **private** String updateTime;  // TODO setter,getter  }  **public class** Sysuser {  **private** Integer uid;  **private** String uname;  **private** String upwd;  **private** String uphone;  **private** String uemail;  **private** String createTime;  **private** String updateTime;  **private** Integer rid;  **private** Sysrole srole;*//对象持有关系*  //TODO setter,gettter  } |

#### 实现思路1:手工进行多次查询

手工进行两次sql查询，第一次查询用户，第二次查询角色，把查询结果组装。

1. 创建SysuserMapper.java与sysuserMapper.xml

|  |
| --- |
| **public interface** SysuserMapper {  **public** Sysuser selectUserByKey(Integer uid);  } |
| <**select id="selectUserByKey" parameterType="int" resultType="Sysuser"**>  select *\** from sysuser where uid=#{uid} </**select**> |

1. 创建SysroleMapper.java与sysroleMapper.xml

|  |
| --- |
| **public interface** SysroleMapper {  **public** Sysrole selectRoleByRid(Integer rid);  } |
| <**select id="selectRoleByRid" parameterType="int" resultType="sysrole"**>  select *\** from sysrole where rid=#{rid} </**select**> |

1. 测试类

|  |
| --- |
| @Test **public void** test1\_selectUserAndRole2(){  Integer uid=1;  SysuserMapper mapper = session.getMapper(SysuserMapper.**class**);  SysroleMapper roleMapper = session.getMapper(SysroleMapper.**class**);  //第一次查询用户  Sysuser suser = mapper.selectUserByKey(uid);  //第二次查询角色  Sysrole sysrole = roleMapper.selectRoleByRid(suser.getRid());  //组装对象  suser.setSrole(sysrole);  System.out.println(suser); } |

#### 实现思路2:association标签进行多次查询

第二种思路本质仍然是第一种思路，但是二次查询由mybaits底层实现，并进行查询结果的组装。

1. 在sysuserMapper.xml映射文件中添加selectUserAndRoleByUserKey

|  |
| --- |
| <**resultMap id="userAndRoleMap" type="sysuser"**>  <**id column="uid" property="uid"**></**id**><**result column="uname" property="uname"**></**result**>  <**result column="upwd" property="upwd"**></**result**>  <**result column="create\_time" property="createTime"**></**result**><**result column="update\_time" property="updateTime"**></**result**><**association property="srole" column="rid" javaType="Sysrole" select="com.javasm.sys.mapper.SysroleMapper.selectRoleByRid"**></**association**> </**resultMap**>  <**select id="selectUserAndRoleByUserKey" parameterType="int" resultMap="userAndRoleMap"**>  select *\** from sysuser where uid=#{uid} </**select**> |

映射文件注意点：

注意点1：association是resultMap的子标签，用来做持有关系映射。

注意点2：association标签的column属性是二次查询需要的外键列名，property属性是属性名；javaType是属性类型；select是二次查询的namespace.id

1. 测试类

|  |
| --- |
| @Test **public void** test1\_selectUserAndRole(){  Integer uid=1;  SysuserMapper mapper = session.getMapper(SysuserMapper.**class**);  Sysuser user1 = mapper.selectUserAndRoleByUserKey(uid);*//使用mybatis的二次关联查询* System.out.println(user1);  System.out.println(user1.getSrole()); } |

#### 实现思路3:association标签对级联sql查询结果映射

1. 在sysuserMapper.xml和SysuserMapper.java中添加代码

|  |
| --- |
| <**resultMap id="userAndRoleMap2" type="sysuser"**>  <**id column="uid" property="uid"**></**id**>*<!--对主键列做映射-->* <**result column="uname" property="uname"**></**result**>*<!--对非主键列做映射-->* <**result column="upwd" property="upwd"**></**result**>*<!--对非主键列做映射-->* <**result column="create\_time" property="createTime"**></**result**>*<!--对非主键列做映射-->* <**result column="update\_time" property="updateTime"**></**result**>*<!--对非主键列做映射-->* <**result column="rid" property="rid"**></**result**>  <**association property="srole" javaType="Sysrole"**>  <**id column="rid" property="rid"**></**id**>  <**result column="rname" property="rname"**></**result**>  <**result column="create\_time" property="createTime"**></**result**>  <**result column="update\_time" property="updateTime"**></**result**>  </**association**> </**resultMap**>  <**select id="selectUserAndRoleByUserKey2" parameterType="int" resultMap="userAndRoleMap2"**>  select u.*\**,r.rname from sysuser u,sysrole r where u.rid=r.rid and u.uid=#{uid} </**select**> |

注意点1：selectUserAndRoleByUserKey2的sql语句是表交叉查询。

注意点2：association标签不需要写column与select属性进行二次查询。

1. 执行测试

|  |
| --- |
| @Test **public void** test1\_selectUserAndRole(){  Integer uid=1;  SysuserMapper mapper = session.getMapper(SysuserMapper.**class**);  Sysuser user2 = mapper.selectUserAndRoleByUserKey2(uid);*//mysql数据库表的交叉连接查询，把查询结果进行结果集映射。* System.out.println(user2);  System.out.println(user2.getSrole()); } |

### 一对多聚合关系

角色持有用户对象的集合，称为聚合关系，要求查询角色时，对该角色下的用户集合进行级联查询。

#### 修改实体类

|  |
| --- |
| **public class** Sysrole {  **private** Integer rid;  **private** String rname;  **private** String createTime;  **private** String updateTime;  **private** List<Sysuser> userList;*//对象聚合关系*  } |

#### 实现思路1:手工进行多次查询

1. 创建SysuserMapper.java与sysuserMapper.xml

|  |
| --- |
| **public interface** SysuserMapper {  **public** List<Sysuser> selectUsersByRoleId(Integer rid);  } |
| <**select id="selectUsersByRoleId" parameterType="int" resultType="sysuser"**>  select *\** from sysuser where rid=#{rid} </**select**> |

1. 创建sysroleMapper.java与sysroleMapper.xml

|  |
| --- |
| **public interface** SysroleMapper {  **public** Sysrole selectRoleByRid(Integer rid);  } |
| <**select id="selectRoleByRid" parameterType="int" resultType="sysrole"**>  select *\** from sysrole where rid=#{rid} </**select**> |

1. 测试

|  |
| --- |
| @Test **public void** test1\_selectRoleAndUsers(){  Integer rid=1;  SysroleMapper mapper = session.getMapper(SysroleMapper.**class**);  SysuserMapper userMapper= session.getMapper(SysuserMapper.**class**);  *//一次查询，查询角色对象* Sysrole sysrole = mapper.selectRoleByRid(rid);  *//二次查询，查询用户集合* List<Sysuser> userList = userMapper.selectUsersByRoleId(rid);  *//组装对象* sysrole.setUserList(userList);  System.out.println(sysrole); } |

#### 实现思路2:collection标签进行多次查询

使用collection标签进行多次查询，本质仍然是两次单表查询，与思路1代码类似，代码交由mybatis底层完成。

1. 在sysroleMapper.xml与sysroleMapper.java中添加代码

|  |
| --- |
| **public** Sysrole selectRoleAndUsersByRid(Integer rid); |
| <**resultMap id="roleAndUserMap" type="sysrole"**>  <**id column="rid" property="rid"**></**id**>  <**result column="rname" property="rname"**></**result**>  <**result column="create\_time" property="createTime"**></**result**>  <**result column="update\_time" property="updateTime"**></**result**>  <**collection property="userList" column="rid" ofType="sysuser" select="com.javasm.sys.mapper.SysuserMapper.selectUsersByRoleId"**></**collection**> </**resultMap**>  <**select id="selectRoleAndUsersByRid" parameterType="int" resultMap="roleAndUserMap"**>  select *\** from sysrole where rid=#{rid} </**select**> |

注意点1：collection 标签专用来做聚合关系查询，ofType表示集合的泛型类型；select表示二次查询的位置。

1. 测试

|  |
| --- |
| @Test **public void** test2\_selectRoleAndUsers2(){  Integer rid=1;  SysroleMapper mapper = session.getMapper(SysroleMapper.**class**);  Sysrole sysrole = mapper.selectRoleAndUsersByRid(rid);  System.out.println(sysrole); } |

#### 实现思路3:collection标签对级联sql查询结果映射

思路3是使用sql进行表交叉查询，一次查询出需要的所有数据，然后由mybaits进行结果映射。

1. 映射文件与映射接口分别添加如下。

|  |
| --- |
| <**resultMap id="roleAndUserMap2" type="sysrole"**>  <**id column="rid" property="rid"**></**id**>  <**result column="rname" property="rname"**></**result**>  <**result column="create\_time" property="createTime"**></**result**>  <**result column="update\_time" property="updateTime"**></**result**>  <**collection property="userList" ofType="sysuser"** >  <**id column="uid" property="uid"**></**id**>  <**result column="uname" property="uname"**></**result**>  <**result column="upwd" property="upwd"**></**result**>  <**result column="create\_time" property="createTime"**></**result**>  <**result column="update\_time" property="updateTime"**></**result**>  </**collection**> </**resultMap**>  <**select id="selectRoleAndUsersByRid2" parameterType="int" resultMap="roleAndUserMap2"**>  select u.*\**,r.rname from sysrole r,sysuser u where r.rid=u.rid and r.rid=#{rid} </**select**> |
| **public** Sysrole selectRoleAndUsersByRid2(Integer rid); |

注意：collection子标签不需要select进行二次查询。

1. 测试

|  |
| --- |
| @Test **public void** test2\_selectRoleAndUsers3(){  Integer rid=1;  SysroleMapper mapper = session.getMapper(SysroleMapper.**class**);  Sysrole sysrole = mapper.selectRoleAndUsersByRid2(rid);  System.out.println(sysrole); } |

## 动态SQL语句

所谓动态sql语句，即在xml映射文件中进行sql语句拼接。

### If与where标签

If标签通常不单独使用，与where或set一起使用。

If与where联合实现条件查询.

|  |
| --- |
| <select id="selectUsers" parameterType="Sysuser" resultType="Sysuser">  Select \* from sysuser  <where>  <if test="uname != null and uname!=''">  and uname like "%"#{uname}"%"  </if>  <if test="upwd != null and upwd!=''">  and upwd=#{upwd}  </if>  <if test="uphone!= null and uphone!=''">  and uphone=#{uphone}  </if>  </where>  </select> |

注意点1：多条件之间使用and或or运算符。

注意点2：加上uphone!=''是因为前端传递的查询参数可能为空字符串。

注意点3：Where标签自动生成where关键字，在遇到第一个字段是AND或OR时,会做自动去除的处理。如果where标签中间无内容，则不生成where

### if与Set标签

Set标签用在update标签中，来完成修改SQL语句中的逗号去除。

|  |
| --- |
| <update id="updateUser" parameterType="Sysuser">  update sysuser  <set>  <if test="uname != null">  uname=#{uname},  </if>  <if test="upwd != null">  upwd=#{upwd},  </if>  <if test="uphone!= null">  uphone=#{uphone},  </if>  <if test="uemail!= null">  uemail=#{uemail},  </if>  </set>  where uid=#{uid}  </update> |

注意点1：set标签自动生成set关键字，在遇到最后一个逗号时,会做自动去除的处理。

### Foreach标签

批量删除或批量添加或批量查询等操作可以使用foreach标签实现

collection – 传递参数类型list/array/自定义key

index – 循环索引

item – 集合中的元素

open –以什么开始; close – 以什么结束

#### 批量删除

1. 写法1：

|  |
| --- |
| <delete id="delUsersByKeys" >  delete from sysuser where uid in  <foreach collection="array" open="(" close=")" separator="," item="userid">  #{userid}  </foreach>  </delete> |
| public int delUsersByKeys(Integer[] uids); |

注意：由于mapper接口中的方法形参是数组，mybatis底层把数组对象封装到Map中，key为”array”,value为数组对象，因此foreach标签的collection写array

1. 写法2

|  |
| --- |
| public int delUsersByKeys2(List<Integer> uids); |
| <delete id="delUsersByKeys2" >  delete from sysuser where uid in  <foreach collection="list" open="(" close=")" separator="," item="userid">  #{userid}  </foreach>  </delete> |

注意：由于mapper接口中的方法形参是List，mybatis底层把数组对象封装到Map中，key为”list”,value为List集合对象，因此foreach标签的collection写list

1. 写法3

|  |
| --- |
| public int delUsersByKeys3(@Param(“uids”)List<Integer> uids); |
| <delete id="delUsersByKeys3" >  delete from sysuser where uid in  <foreach collection="uids" open="(" close=")" separator="," item="userid">  #{userid}  </foreach>  </delete> |

注意：由于mapper接口中的方法形参通过@Param注解指定名称uids，mybatis底层把参数封装到Map中，key为”uids”,value为List集合对象，因此foreach标签的collection写uids。

#### 批量插入

|  |
| --- |
| <insert id="addUser2">  insert into sysuser(uname, upwd, uphone, uemail) values  <foreach collection="array" separator="," item="suser">  (#{suser.uname},#{suser.upwd},#{suser.uphone},#{suser.uemail})  </foreach>  </insert> |
| public int addUser2(Sysuser[] suser); |

#### 批量查询

|  |
| --- |
| public List<Sysuser> selectusersByIds(Integer[] uids); |
| <select id="selectusersByIds" resultType="sysuser">  select \* from sysuser where rid in  <foreach collection="array" open="(" close=")" separator="," item="userid">  #{userid}  </foreach>  </select> |

### Choose-when-otherwise

Mybatis中的choose标签相当于Java中的switch。仅供了解。

## 数据延迟加载

Mybatis中的延迟加载是在使用association或collection标签进行多次单表查询时对二次查询进行延迟查询的一种实现。实际项目中不经常使用，仅供了解。

### Mybatis延迟加载使用

#### 核心配置

|  |
| --- |
| <setting name=*"logImpl"* value=*"STDOUT\_LOGGING"*/>  <!-- 启用延迟加载 -->  <setting name=*"lazyLoadingEnabled"* value=*"true"*/>  <!-- 积极加载改为消极加载 -->  <setting name=*"aggressiveLazyLoading"* value=*"false"*/>  <!-- 调用toString，equals不触发延迟对象的加载 -->  <setting name=*"lazyLoadTriggerMethods"* value=*""*/> |

#### 使用效果

以多对一查询或一对多查询思路2进行代码测试，查看执行结果

## Mybatis缓存

MyBatis缓存是指对查询操作的结果进行缓存，再下一次进行查询时，先查询缓存中数据，如果缓存中查到则执行返回；如果缓存中查询不到则查询数据库，并把非空查询结果放在缓存中，供下次查询使用。

mybatis缓存设计分为两个级别的缓存，一般认识是session会话级别缓存与sessionFactory全局缓存。

### 一级缓存

一级缓存是默认开启的，仅对同一个会话中的查询生效。Session关闭后，缓存清空，同时在当前Session中使用增删改时,会清空该Session中的缓存

执行测试：

|  |
| --- |
| @Test  **public void** test1\_selectUserByKey(){  SysuserMapper mapper = session.getMapper(SysuserMapper.**class**);Sysuser sysuser = mapper.selectUserByKey(1);  System.out.println(sysuser);  System.out.println(**"------------分割-------------"**); *// session.clearCache(); // session.close();* Sysuser sysuser1 = mapper.selectUserByKey(1);  System.out.println(sysuser1);  } |

### 二级缓存

二级缓存需要在mybatis配置文件中进行相应的配置才可以使用

当使用查询时,查询结果会存入对应的namespace中.

当所属namespace使用增删改时,会清空该namespace中的缓存

二级缓存可能会存入内存,也可能会存入硬盘

由于二级缓存可能会存入硬盘,所以需要将对应需要缓存的实体类进行序列化(implements Serializable)。

1. Xml核心配置中开启二级缓存

|  |
| --- |
| <!-- 启用二级缓存 -->  <setting name=*"cacheEnabled"* value=*"true"*/> |

1. 映射文件配置

* 在Mapper XML文件中设置缓存策略，默认情况下是没有开启缓存的

|  |
| --- |
| <!-- 配置缓存策略 -->  <cache eviction=*"FIFO"* flushInterval=*"60000"* size=*"512"* readOnly=*"true"*></cache> |

* 在Mapper XML文件配置支持cache后，默认当前映射文件中所有查询启用二级缓存，如果需要对个别查询进行调整，可以单独设置useCache=false禁用二级缓存。

|  |
| --- |
| <**select id="selectUserByKey" parameterType="int" resultType="Sysuser"**>  select *\** from sysuser where uid=#{uid} </**select**> |

注意：select标签有useCache=false表示当前查询禁用二级缓存。

* 测试运行

|  |
| --- |
| @Test **public void** test2\_factoryCache(){  SysuserMapper mapper = session.getMapper(SysuserMapper.**class**);  Sysuser sysuser = mapper.selectUserByKey(1);  System.out.println(sysuser);  System.out.println(**"------------分割-------------"**);  session.close();   SqlSession session2 = factory.openSession();  SysuserMapper mapper1 = session2.getMapper(SysuserMapper.**class**);  Sysuser sysuser1 = mapper1.selectUserByKey(1);  System.out.println(sysuser1);  session2.close(); } |

## 代理模式

代理模式的定义：代理模式给某一个对象提供一个代理对象，并由代理对象控制对原对象的引用。通俗的来讲代理模式就是我们生活中常见的中介。

应用目的：一个对象中的某一个或多个方法不满足需求，需要扩展该对象的时候，在不修改源代码的情况下，去扩展对象中的方法。比如原生Connection对象的close方法进行连接关闭，如果引入了连接池对象，放入连接池对象中的Connection对象的close方法不能进行关闭，需要扩展close方法，把对象放回连接池。

代理模式分为静态代理与动态代理两种实现。包含两个对象：原对象，代理对象。

### 静态代理

1. 准备代码环境

|  |
| --- |
| **public interface** ISource {  **public** String start(String a,Integer b);  **public** String end(); } |
| **public class** SourceImpl **implements** ISource {  @Override  **public** String start(String a,Integer b) {  System.out.println(**"sourceImpl对象中的start方法:"**+a+**"--"**+b);  **return** a+b;  }  @Override  **public** String end() {  System.out.println(**"sourceImpl对象中的end方法"**);  **return "ok"**;  } } |

测试代码

|  |
| --- |
| ISource source = **new** SourceImpl(); source.start(**"aa"**,123); |

此时我们认为start方法需要扩展，如何能够在不改动源代码的情况下扩展呢？引入代理模式，创建静态代理类：

|  |
| --- |
| **public class** SourceProxy **implements** ISource {  **private** ISource source;*//原对象* **public** SourceProxy(ISource source) {  **this**.source = source;  }   @Override  **public** String start(String a, Integer b) {  System.out.println(**"前置扩展"**);  String result= **this**.source.start(a,b);  System.out.println(**"后置扩展"**);  **return** result;  }   @Override  **public** String end() {  **return this**.source.end();  } } |

注意：静态代理类具有两个特征：

特征1：代理类与原对象同类型；

特征2：代理类持有原对象的引用。

此时测试代码如下：

|  |
| --- |
| *//原对象* ISource source = **new** SourceImpl(); *//代理对象* ISource proxy= **new** SourceProxy(source); proxy.start(**"aa"**,123); |

静态代理缺点：在原对象中的方法非常多时，需要在静态代理类中一个个的调用原对象的方法。代码比较繁琐。

### 动态代理

使用jdk的Proxy类在程序运行期间在虚拟机中创建代理类,类名规则$Proxy0.$Proxy1...。并编译该代理类，返回实例化对象。

|  |
| --- |
| **public static** ISource getProxy(ISource source){  ClassLoader loader = source.getClass().getClassLoader();  Class[] interfaces=**new** Class[]{ISource.**class**};   InvocationHandler handler = **new** InvocationHandler() {@Override  **public** Object invoke(Object proxy, Method method, Object[] args) **throws** Throwable {  String name = method.getName();  **if**(**"start"**.equals(name)){  System.out.println(**"前置扩展"**);  Object result = method.invoke(source,args);  System.out.println(**"后置扩展"**);  **return** result;  }**else**{  Object result = method.invoke(source, args);*//执行某个对象中的方法* **return** result;  }  }  };   ISource proxy = (ISource) Proxy.newProxyInstance(loader,interfaces,handler);  **return** proxy; } |

此时测试代码如下：

|  |
| --- |
| *//原对象* ISource source = **new** SourceImpl(); *//代理对象* ISource proxy= getProxy(source); proxy.start(**"aa"**,123); |

注意：可以修改虚拟机参数，把内存中的代理类保存本地，查看代理类源码

|  |
| --- |
| System.getProperties().put(**"sun.misc.ProxyGenerator.saveGeneratedFiles"**, **"true"**); |

注意点1：

 Proxy.newProxyInstance()方法接受三个参数：

ClassLoader loader:指定当前目标对象使用的类加载器,获取加载器的方法是固定的

Class<?>[] interfaces:指定目标对象实现的接口的类型,使用泛型方式确认类型

InvocationHandler:指定动态处理器，执行目标对象的方法时,会触发事件处理器的invoke方法

注意点2：

Proxy类创建代理类要求原对象必须有接口才适用。因为$Proxy0代理类结构大致如下：

|  |
| --- |
| Public class $Proxy0 extends Proxy implements ISource{  Private InvocationHandler h;  Private static Method method0=null;  Private staitc Method method1=null;  Public $Proxy0(InvocationHandler h){  This.h=h;  }  Public Stirng start(String arg1,Integer arg2){  This.h.invoke(method0,this,new Object[]{arg1,arg2} )  }  Static{  Method0=Class.forname(“ISource”).getMethod(“start”);  Method1=Class.forname(“ISource”).getMethod(end)  }  } |

该类继承自Proxy类，只能适用implements关键字派生,因此原对象必须有接口。

## 总结

Sql映射XML文件进行增删改查操作。

resultMap

动态sql语句

if

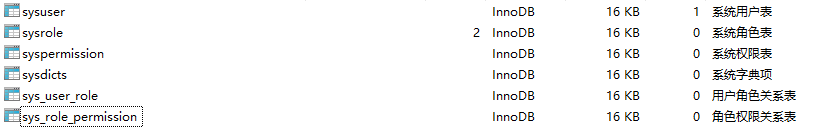
choose、when、otherwise

trim、where、set

foreach

## mybatis总体任务

完成系统管理下用户，角色，权限，字典项四个模块的dao层接口及映射文件，并进行测试。

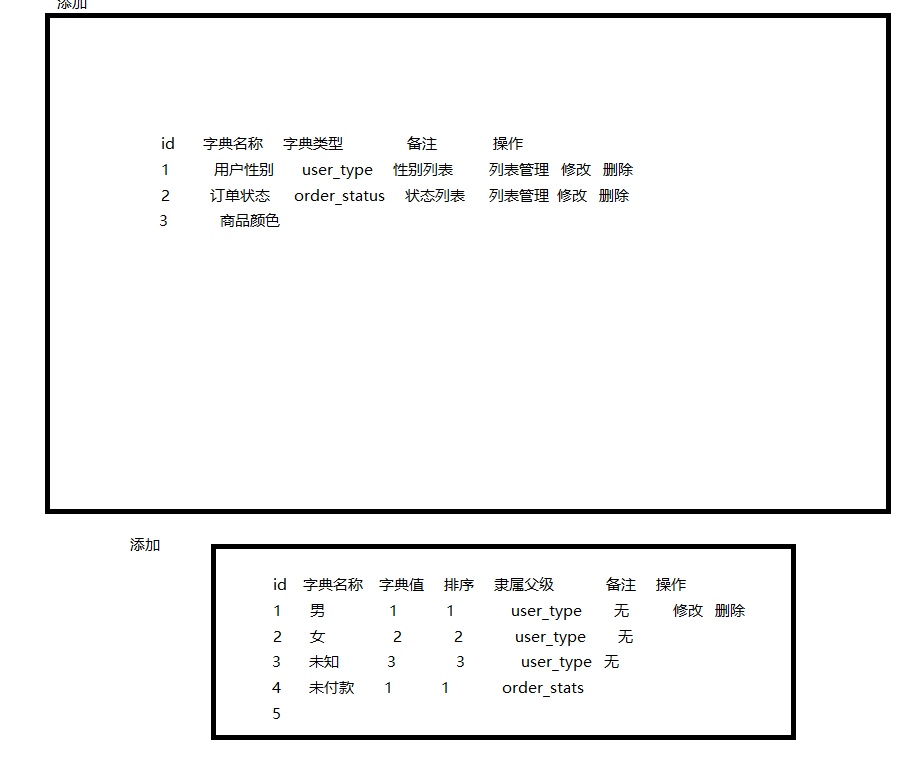


Sysdept部门表(id,部门名称,上级部门,负责人,联系方式)

Sysdicts:字典表

实现内容:(create\_time,update\_time,create\_by,update\_by)

1. 每张表基本的添加方法,根据id批量删除,插入单个用户,批量插入用户,根据id查询,高级条件查询(一定要指定排序方式).添加用户时,维护用户所属部门.
2. 为用户设置角色;
3. 查询用户时,级联查询角色信息;
4. 为角色设置菜单权限;
5. 查询角色下的权限列表;
6. 查询某用户的权限列表;
7. 字典项:项目中固定的列表数据,一般下拉框,复选框,单选框使用.



8.组织机构管理