# 基本查询标签

## Select标签

## resultMap

resultMap是一种很重要的配置结果映射的方法，我们必须熟练掌握resultMap的配置方法，resultMap包含的所有属性如下：

id：必填，并且唯一，在select标签中，resultMap指定的值即为此处id所设置的值

type：必填，用于配置查询列所映射到的java对象类型。

extends：选填，可以配置当前的resultMap继承其他的resultMap，属性值为继承resultMap的id。

autoMapping：选填，可选值为true或false，用于配置是否启用非映射字段的自动映射功能，该配置可以覆盖全局的autoMappingBehavior配

constructor：配置使用构造方法注入结果，包含以下两个字标签：

idArg:id参数，标记结果作为id（谓一致），可以帮助提高整体性能。

arg：注入到构造方法的一个普通结果。

id：一个id结果，记结果作为id（谓一致），可以帮助提高整体性能

result：注入到Java对象属性的普通结果。

association：一个复杂的类型关联，许多结果将包成这种类型。

collection：复杂类型的集合

discriminator：根据结果值来决定使用哪个结果映射。

case：基于某些指定结果映射。

### result子标签

rusult标签包含的属性：

column：从数据库中得到的列名，或者是列的别名。

property：映射到列结果的属性，可以映射简单类型，如“username”，也可以映射复杂类型，如“address.schoolAddress”，这会通 过“.”方式的属性嵌套赋值。

javaType：一个java类的完全限定名，或一个类型别名（通过typeAlias配置或默认类型）。如果映射到一个JavaBean，MyBatis通常可以 自动判断属性类型。

jdbcType：列对应的数据库类型。JDBC类型仅仅需要对插入，更新，删除操作可能为空的列进行处理。这是JDBC javaType的需要，而不 是Mybatis的需要。

typeHandler：使用这个属性可以覆盖默认的类型处理器。这个属性值是类的完全限定名或类型别名。

### association标签

association标签常用于一对一关系中。

property：对应实体类中的属性名，必填。

javaType：属性对应的Java类型。

resultMap：可以直接使用现有的resultMap，而不需要再重复配置。

columnPrefix：查询列的前缀，配置前缀后，在子标签配置result的column时可以省略前缀。

嵌套查询常用属性：

select：另一个映射查询的id，MyBatis会额外执行这个查询获取嵌套对象的结果。

column：列名（或别名），将主查询中列的结果作为嵌套查询的参数，配置方式如：column={prope1=col2,prop2=col2}，prop1he prop2将作为嵌套查询的参数。

fetchType：数据加载方式，可选值为lazy和eager，分别为延迟加载和积极加载，这个配置会覆盖全局的lazyLoadingEnabled配置。

案例：

有用户表实体和角色表实体（get和set方法省略了。）一个用户对用于一个角色。

|  |
| --- |
| *\*\*  \* 用户表  \*/* public class SysUser implements Serializable {  private static final long serialVersionUID = -328602757171077630L;  */\*\*  \* 用户ID  \*/* private Long id;  */\*\*  \* 用户名  \*/* private String userName;  */\*\*  \* 密码  \*/* private String userPassword;  */\*\*  \* 邮箱  \*/* private String userEmail;  */\*\*  \* 简介  \*/* private String userInfo;  */\*\*  \* 头像  \*/* private byte[] headImg;  */\*\*  \* 创建时间  \*/* private Date createTime;    */\*\*  \* 用户角色  \*/* private SysRole role;    */\*\*  \* 用户的角色集合  \*/* private List<SysRole> roleList;  @Override  public String toString(){  return "SysUser{"+"id="+id+", userName='"+userName+'\''+", userPassword='"+userPassword+'\''+", userEmail='"+userEmail+'\''+", userInfo='"+userInfo+'\''+", headImg="+Arrays.toString(headImg)+", createTime="+createTime+", role="+role+", roleList="+roleList+'}';  } } |

|  |
| --- |
| */\*\*  \* 角色表  \*/* public class SysRole implements Serializable {  private static final long serialVersionUID = 6320941908222932112L;  */\*\*  \* 角色ID  \*/* private Long id;  */\*\*  \* 角色名  \*/* private String roleName;  */\*\*  \* 有效标志  \*/* private Enabled enabled;  */\*\*  \* 创建人  \*/* private String createBy;  */\*\*  \* 创建时间  \*/* private Date createTime;  */\*\*  \* 用户信息  \*/* private SysUser user;  */\*\*  \* 创建信息  \*/* private CreateInfo createInfo;   */\*\*  \* 角色包含的权限列表  \*/* List<SysPrivilege> privilegeList;  @Override  public String toString(){  return "SysRole{"+"id="+id+", roleName='"+roleName+'\''+", enabled="+enabled+", createBy='"+createBy+'\''+", createTime="+createTime+", user="+user+", createInfo="+createInfo+", privilegeList="+privilegeList+'}';  } } |

需求：根据用户ID查询用户信息和用户角色信息。

UserMapper.java

|  |
| --- |
| */\*\*  \* 根据用户 id 获取用户信息和用户的角色信息  \*   \** ***@param*** *id  \** ***@return*** *\*/* SysUser selectUserAndRoleById(Long id); |

**方法一：使用自动映射处理一对一的关系**

映射文件：UserMapper.xml

|  |
| --- |
| <select id="selectUserAndRoleById" resultType="code.lsh.model.SysUser">  select u.id,  u.user\_name userName,  u.user\_password userPassword,  u.user\_email userEmail,  u.user\_info userInfo,  u.head\_img headImg,  r.id "role.id",  r.role\_name "role.roleName",  r.create\_by "role.createBy",  r.create\_time "role.createTime"  from sys\_user u  inner join sys\_user\_role ur on u.id=ur.usr\_id  inner join sys\_role r on ur.role\_id=r.id  where u.id=#{id}; </select> |

**方法二：使用resultMap配置一对一映射**



|  |
| --- |
| <select id="selectUserAndRoleById" resultMap="userRoleMap">  select u.id,  u.user\_name,  u.user\_password,  u.user\_email,  u.user\_info ,  u.head\_img,  r.id role\_id,  r.role\_name role\_roleName,  r.create\_by role\_createBy,  r.enabled role\_enabled,  r.create\_time role\_createTime  from sys\_user u  inner join sys\_user\_role ur on u.id=ur.usr\_id  inner join sys\_role r on ur.role\_id=r.id  where u.id=#{id}; </select> |

**使用resultMap的extend属性：**

<resultMap id="userMap" type="code.lsh.model.SysUser">  
 <result property="userName" column="user\_name"/>  
 <result property="userPassword" column="user\_password"/>  
 <result property="userEmail" column="user\_email"/>  
 <result property="userInfo" column="user\_info"/>  
 <result property="headImg" column="head\_img" jdbcType="BLOB"/>  
 <result property="createTime" column="create\_time" jdbcType="TIMESTAMP"/>  
</resultMap>

<!--使用继承-->  
<resultMap id="userRoleMap1" type="code.lsh.model.SysUser" extends="userMap">  
 <result property="role.id" column="id"></result>  
 <result property="role.roleName" column="role\_name"></result>  
 <result property="role.enabled" column="enabled"></result>  
 <result property="role.createBy" column="create\_by"></result>  
 <result property="role.createTime" column="role\_create\_time" jdbcType="TIMESTAMP"></result>  
</resultMap>

**方法三：使用resultMap和association标签配置一对一映射。**

|  |
| --- |
| <resultMap id="userRoleMap" type="code.lsh.model.SysUser" extends="userMap">  <association property="role" javaType="code.lsh.model.SysRole" columnPrefix="role\_" >  <result property="id" column="id"></result>  <result property="roleName" column="role\_name"></result>  <result property="enabled" column="enabled"></result>  <result property="createBy" column="create\_by"></result>  <result property="createTime" column="create\_time" jdbcType="TIMESTAMP"></result>  </association> </resultMap>  <select id="selectUserAndRoleById" resultMap="userRoleMap">  select u.id,  u.user\_name,  u.user\_password,  u.user\_email,  u.user\_info ,  u.head\_img,  r.id role\_id,  r.role\_name role\_roleName,  r.create\_by role\_createBy,  r.enabled role\_enabled,  r.create\_time role\_createTime  from sys\_user u  inner join sys\_user\_role ur on u.id=ur.usr\_id  inner join sys\_role r on ur.role\_id=r.id  where u.id=#{id}; </select> |

**我们可以将sys\_role相关映射提取出来：**

<resultMap id="roleMap" type="code.lsh.model.SysRole">  
 <result property="id" column="id"></result>  
 <result property="roleName" column="role\_name"></result>  
 <result property="enabled" column="enabled"></result>  
 <result property="createBy" column="create\_by"></result>  
 <result property="createTime" column="create\_time" jdbcType="TIMESTAMP"></result>  
</resultMap>

**使用association中的resultMap属性：**

<resultMap id="userRoleMap" type="code.lsh.model.SysUser" extends="userMap">  
 <association property="role" javaType="code.lsh.model.SysRole" columnPrefix="role\_" resultMap="roleMap" ></association>  
</resultMap>

**方法四：使用association标签的嵌套查询**

1.UserMapper.java新增方法。

*/\*\*  
 \* 根据用户 id 获取用户信息和用户的角色信息，嵌套查询方式  
 \*   
 \** ***@param*** *id  
 \** ***@return*** *\*/*SysUser selectUserAndRoleByIdSelect(Long id);

2.新增resultMap。

<!--嵌套查询-->  
<resultMap id="userRoleMapSelect" extends="userMap" type="code.lsh.model.SysUser">  
 <association property="role" column="{id=role.id}" select="selectRoleById"></association>  
</resultMap>

3.使用userRoleMapSelect创建新的查询。

<select id="selectUserAndRoleByIdSelect" resultMap="userRoleMapSelect">  
 select  
 u.id ,  
 u.user\_name,  
 u.user\_password,  
 u.user\_email,  
 u.head\_img,  
 u.create\_time,  
 ur.role\_id  
 from sys\_user u  
 inner join sys\_user\_role ur on u.id=ur.user\_id  
 where u.id=#{id};  
</select>

注意：该查询并没有sys\_role表，我们并不能通过一个SQL获取全部信息，角色信息

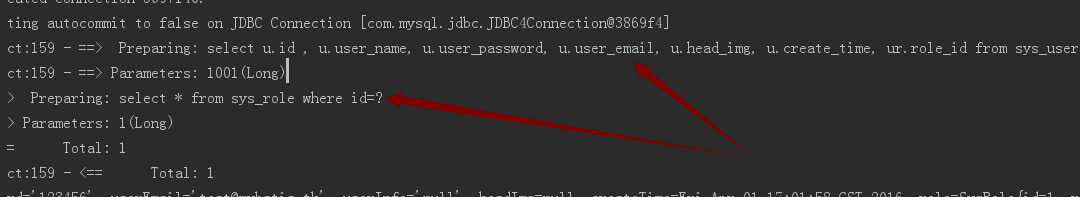
selectRoleById查询的。

<select id="selectRoleById" resultMap="roleMap">  
 select \* from sys\_role where id=#{id}  
</select>

4.运行测试。

@Test  
public void testSelectUserAndRoleByIdSelect()  
{  
 UserMapper mapper = session.getMapper(UserMapper.class) ;  
 SysUser user = mapper.selectUserAndRoleByIdSelect(1001L) ;  
 System.*out*.println(user.toString());  
 Assert.*assertNotNull*(user);  
 Assert.*assertNotNull*(user.getRole());  
}

运行的结果：



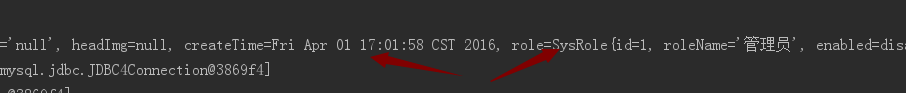
执行两次查询。

问题：如果查询的不是1条数据，而是N条数据，那就会出现N+1问题，主SQL

会出查询一次，查询出N条结果，这N条结果要各自自行一次查询，那就需要进行N次查

询。

解决方法：使用association标签中的fetchType属性,实现延迟加载，解决N+1的问题。



### collection标签

和association类似，集合的嵌套结果映射就是指通过一次SQL查询将所有的结果查询出来，然后通过配置的结果映射，将数据映射到不同的对象中去，在一对多的关系中，主表的一条数据会对应关联表中的多条数据，因此一般查询时会查询出多个结果，按照一对多的数据结构存储数据的时候，最终的结果数会小于等于查询的总记录数。

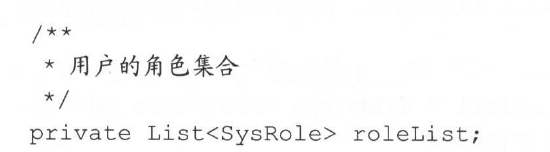
属性：ofType：集合存储的类型。

**collection标签用于一对多的关系。**

**collection支持的属性以及属性的作用和association完全相同。**

**案例：**

**第一步：**修改SysUser表。



**一个用户对应于多个角色。**

**第二步：**创建新的resultMap



至少有一个id标签。id唯一作用就是在嵌套的映射配置时判断数据是否相同，当配置的id标签时，MyBatis只需要逐条比较所有数据中id标签配置的字段值是否相同即可，在配置嵌套结果查询时，配置di标签可以提高处理效率。

**简化：**

<resultMap id="" type="code.lsh.model.SysUser" extends="userMap">  
 <collection property="roleList" resultMap="roleMap" columnPrefix="role\_" javaType="code.lsh.model.SysRole"></collection>  
</resultMap>

**第三步：**创建新的select语句

### discriminator标签

discriminator鉴别器标签用于处理一个单独数据库查询返回很多不同数据类型的结果集。

常用属性：

column：设置要进行鉴别比较值的列。

javaType：指定列的类型，保证使用相同的java类型来比较值。

discriminator标签可以有1个或多个case标签，case标签包含以下三个属性：

value:该值为discriminator指定column用来匹配的值。

resultMap：当column的值和value的值匹配时，用于配置使用resultType指定的映射。

resultType：当column的值和value的值匹配时，用于配置使用resultType指定的映射。

## Insert标签

Insert标签包含属性：

id：命名空间中的唯一标识符。

parameterType：即将传入的语句参数的完全限定类名或别名。这个属相是可选的，因为MyBatis可以推断传入语句的具体参数，因此不建议配置该属性。

flushCache：默认为true，任何时候只要语句被调用，都会清空一级缓存和二级缓存。

timeout：设置在抛出异常之前，驱动程序等待数据库返回请求结果的秒数。

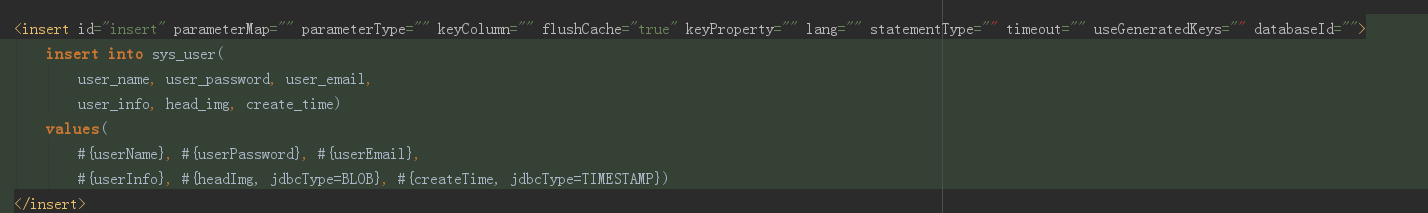
statementType：对于STATEMENT，PREPARED，CALLABLE，MyBatis会分别使用赌赢的Statemnet，PreparedStatment，CallableStatement，默认值为PREPARED。

useGeneratedKeys：默认为false。如果设置为true，Mybatis会使用JDBC个getGeneratedKeys方法来取出数据库内部生成的主键。

keyProperty：MyBatis通过getGeneratedKeys获取主键值后将要赋值的属性名。如果希望得到多个数据库自动生成的列，属性值也可以用逗号分隔属性名称列表。

keyColumn：仅对INSERT和UPDATE有用。通过生成的主键值设置表中的列名，这个设置仅在某些数据库中时必须的，当主键列不是表中的第一列时必须要设置，如果希望得到多个生成的列，也可以使用逗号分隔的属性名称列表。

databaseId：如果配置了databaseIdProvider，MyBatis会加载多有不带databaseId的或匹配当前databaseId语句。如果同时存在代databaseId和不带databaseId的语句，后者会被忽略。



为了防止插入时类型错误，我们可以在INSERT语句，将全部列都列取出来，并且在VALUES

中使用property ,jdbcType=””}方法。

### selectKey字标签

作用：在insert和update语句中插入查询语句，selectKey操作会将操作查询结果赋值到insert元素的parameterType的入参实例下对应的属性中。并提供给insert语句使用

包含属性：

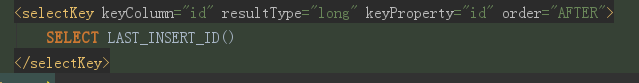
keyProperty：默认值unset，用于设置getGeneratedKeys方法或selectKey子元素返回值将赋值到领域模型的哪个属性中

 resultType ：keyPropety所指向的属性类全限定类名或类型别名

 order ：取值范围BEFORE|AFTER，指定是在insert语句前还是后执行selectKey操作

 statementType ：取值范围STATEMENT,PREPARED（默认值）,CALLABLE

该标签也可以用于返回主键：



在不同的数据库中order值是不同的，在MySQL数据库中order值为AFTER，因为主键值是在insert语句执行之后才能获取，而Oracle数据库中order为BEFORE，因为Oracle表的主键值在数据插入前，需要先从序列中获取值，然后将获取的值作为主键插入到数据库中。

## Update标签

MyBatis在进行更新数据时，首先将要更新的数据先查询出来，后更新。

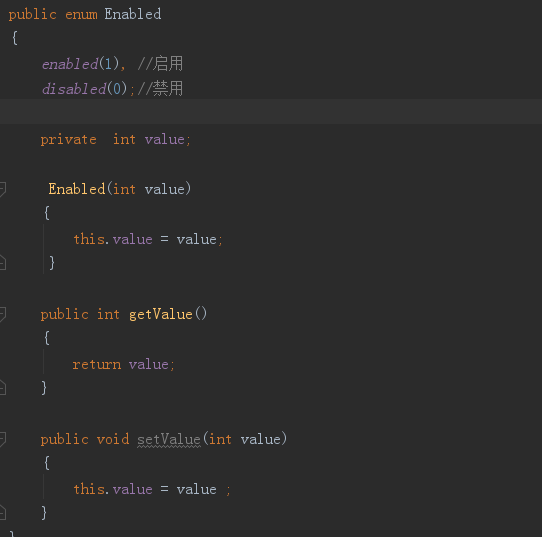
## Delete标签

# 枚举映射

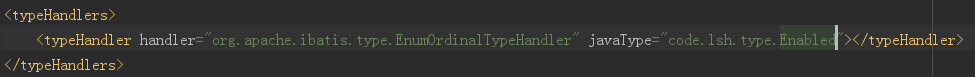
MyBatis在处理Java类型和数据库类型时，使用TypeHandler（类型处理器）对两者进行转换。MyBatis为Java和数据库JDBC中的基本类型和常用类型提供了TypeHandler的实现。

MyBatis在启动时会加载所有的JDBC对应的类型处理器，在处理枚举类型时默认使用：

org.apache.ibatis.type.EnumTypeHandler处理器，这个处理器会将枚举类型转换成字符串类型的字面值并使用，如果数据库中使用非字符长类型，则转换一定失败。



解决方法：在MyBatis配置文件中配置：



EnumOrdinalTypeHandler使用枚举的索引进行处理。

如果值既不是枚举的字面值，又不是枚举的索引值，我们可以实现自己的类型处理器：

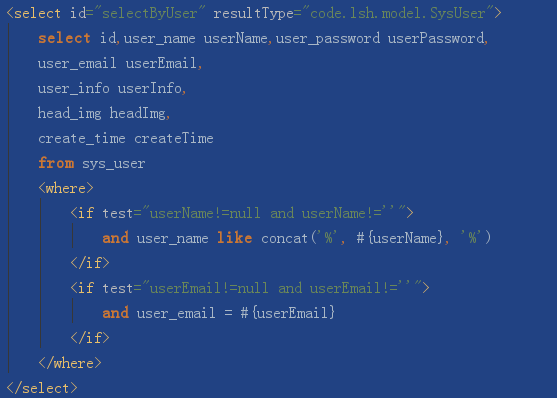
* 实现TypeHandler接口
* 继承BaseTypeHandler类

具体实现可以参考官方已经实现的类型。

# 动态SQL标签

## where标签

Where标签的作用：如果该标签包含的元素中有返回值，就插入一个where，如果where后面的字段串是以AND和OR开头的，就将它们剔除。



and不能少，如果删除and，则在条件都满足时就会报语法错误错：



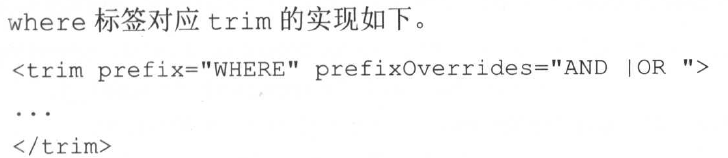
第一个判断语句中的and可以去除。

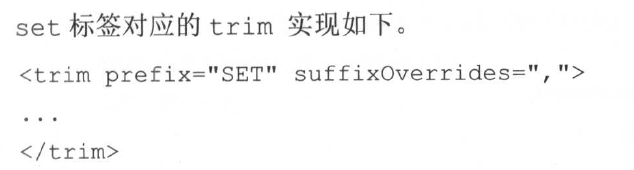
## set标签

set标签的作用：如果该标签包含的元素中有返回值，就插入一个set，如果set后面的字段串是以逗号结尾的，就将它们剔除

## trim标签

where标签和set标签的功能都可以用trim标签来实现，并且在底层就是通过TrimSqlNode实现的。





Trim标签属性：

prefix：当trim元素内包含内容时，会给内容增加prefix指定的前缀。

prefixOverrides：当trim元素内容包含内容时，会把内容中匹配的前缀字符串

去掉。

suffix：当trim元素内包含内容时，会给内容增加suffix指定的前缀

suffixOverrides：当trim元素内包含内容时，会把内容中匹配的后缀字符串去

掉。

## foreach标签

foreach标签可以对数组，Map或实现了Iterable接口的对象进行遍历。数组在处理时会转换成List对象，因此foreach遍历的对象可以分为两大类：Map类型和Iterable类型。

1. **多个参数**

在多个参数时，可以使用@param注解标注方法参数名，collection值为该注解指定的名字。

1. **参数时Map类型**

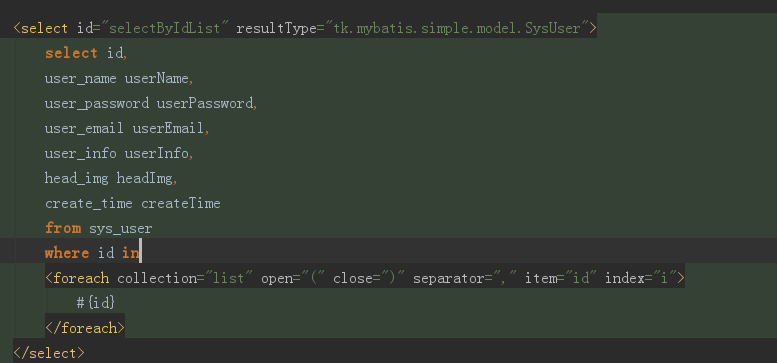
使用Map和使用@param注解方式相似，将collection指定为对应的Map中的key即可。如果要循环所传入的Map，推荐使用@Param注解指定名字，此时可将collection设置为指定的名字，如果不想指定名字，就是用默认值\_Parameter

1. **参数是一个对象。**

指定为对象的属性名即可。当时用对象内多层嵌套的对象时，使用属性.属性

### foreach实现in操作





foreach包含的属性：

collection：必填，值为要迭代循环的属性名，这个属性值的情况有很多。

item：变量名，值为从迭代对象中取出的每一个值。

index：索引的属性名，在集合数组情况下值为当前索引值，当迭代循环的对象时Map类型时，这个值为Map的key。

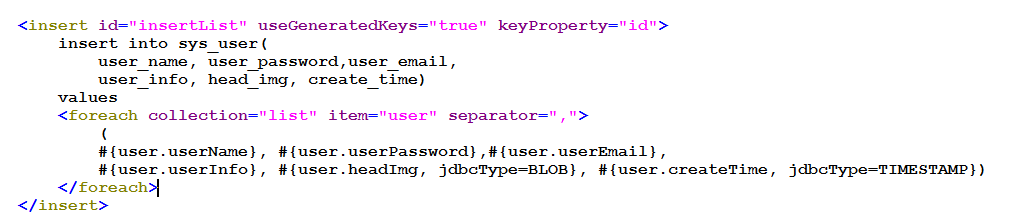
open：整个循环内容开头的字符串。

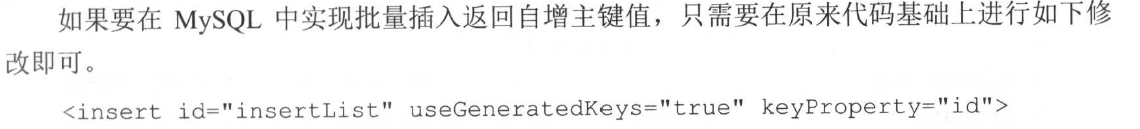
close：整个循环内容结尾的字符串。

separator：每次循环的分隔符。

### foreach批量插入

在MyBatis3.3.1版本开始，MyBatis开始支持批量新增会写主键的功能，这个功能首先要求数据库主键值为自增类型，同时还要求改数据库提供的JDBC驱动可以支持返回批量出入的主键值。





# bind标签

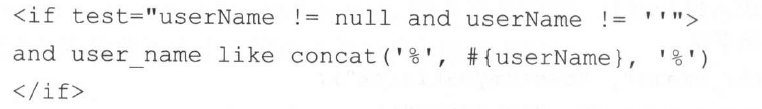
bind标签可以使用OGNL表达式创建一个变量并将其绑定到上下文中。

属性：

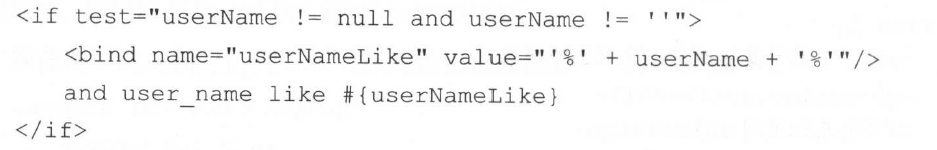
Name：绑定到上下文的变量名。

Value：OGNL表达式。

没使用bind：



使用bind：



。。