接着来看一下id 和result 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。可以映射简单的如“ username ”这样的属性，也

可以映射一些复杂对象中的属性， 例如“ address.street.number ”，这会通过“ ．”

方式的属性嵌套赋值。

javaType ： 一个Jav a 类的完全限定名，或一个类型别名（通过typeAlias 配置或者

默认的类型）。如果映射到一个JavaB ean, M yB at is 通常可以自动判断属性的类型。如

果映射到HashMap ，则需要明确地指定j av a Type 属性。

jdbcType ： 列对应的数据库类型。JDB C 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JDB C j dbcType 的需要，而不是MyB atis 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。

接口中定义的返回值类型必须和泊位中配置的resultType 类型一致。返回值类型是由XM L 中的resul tType （或resultMap 中的type ）决

定的，不是由接口中写的返回值类型决定的

如果使用result Type 来设置返回结果的类型，需要在SQL 中为所有列

名和属性名不一致的列设置别名，通过设置别名使最终的查询结果列和result Type 指定对象

的属性名保持一致，进而实现自动映射。

property 属性或别名委和对象中属性的名字相同，但是实际匹配时， MyBatis 会先将两

者都转换为大写形式，然后再判断是否相同，即property ＝” userName ”和property=

” username ”都可以匹配到对象的userName 属性上。判断是否相同的时候要使用USERNAME,

因此在设直property 属性或别名的时候，不需要考虑大小写是否一致，但是为了便于阅读，

要尽可能按照统一的规则来设直。

由于大多数数据库设置不区分大小写，因此下画线方式的命名很常见，如user\_name 、user\_email 。在Java 中， 一般都使用驼峰式命名，如userName 、userEmail 。

因为数据库和Java 中的这两种命名方式很常见，因此MyBatis 还提供了一个全局属性

mapUnderscoreToCamelCase ，通过配置这个属性为true 可以自动将以下画线方式命名的

数据库列映射到Java 对象的驼峰式命名属性中。

<settings>

〈！－－其他面己直一〉

<setting name= ” mapUnderscoreToCamelCase ” value=” true ” />

</ settings >

多表联合查询

1，虽然这个多表关联的查询中涉及了3 个表，但是返回的结果只有sys role 表中的信息，  
 所以直接使用SysRole 作为返回值类型即可

2.查询的结果不仅要包含sys\_role 中的信息，还要包含当前用户的部分信息（不考虑嵌套的情

况），例如增加查询列u .user\_name as userName 。这时resultType 该如何设置呢？

第一种方法就是在SysRole 对象中直接添加userName 属性，

或者也可以创建一个如下所示的对象。将resultType 设置为扩展属性后的SysRoleExtend 对象

public class SysRoleExtend extends SysRole {

private String userName ;

public String getUserName() {

return userName;

public void setUserName(String userName) {

this . userName = userName ;

}

}

第二种方法，直接在SysRole 中增加SysUser 对象， 字段名为user ，增加这个字段后，修改XML

中的selectRolesByUserid 方法。。

/\*\*

角色表

\*/

public class SysRole {

//其他原有字段． ．．

private SysUser user ;

}

<select id="selectRolesByUserId" resultType="tk.mybatis.simple.model.SysRole">  
 select   
 r.id,   
 r.role\_name roleName,   
 r.enabled,  
 r.create\_by createBy,  
 r.create\_time createTime,  
 u.user\_name as "user.userName",  
 u.user\_email as "user.userEmail"  
 from sys\_user u  
 inner join sys\_user\_role ur on u.id = ur.user\_id  
 inner join sys\_role r on ur.role\_id = r.id  
 where u.id = #{userId}  
</select>

查询列增加的两行，如下所示。

u.user\_name as "user.userName",  
u.user\_email as "user.userEmail"  
这里在设置别名的时候，使用的是“ user . 属性名”， user 是SysRole 中刚刚增加的SysUser字段

， userName 和userEmail 是SysUser 对象中的属性， 通过这种方式可以直接将值赋给

user 字段中的属性。

先看＜insert>元素，这个标签包含如下属性。

• id：命名空间中的唯一标识符，可用来代表这条语句。

parameterType ：即将传入的语句参数的完全限定类名或别名。这个属性是可边的，

因为MyBatis 可以推断出传入语句的具体参数，因此不建议配置该属性。

flushCache ：默认值为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。

statemetType ：对于STATEMENT 、PREPARED 、CALLABLE, MyBatis 会分别使用

对应的Statement 、PreparedStatement 、Callable S tatement ，默认值为

PREPARED 。

useGeneratedKeys ：默认值为false 。如果设置为true, MyBatis 会使用JDB C

的getGeneratedKeys 方法来取出由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，属性值也可以是以逗号分隔的属性名称列表。

keyColumn ：仅对INSERT 和UPDATE 有用。通过生成的键值设置表中的列名， 这个

设置仅在某些数据库（如PostgreSQL ）中是必须的， 当主键列不是表中的第一列时需

要设置。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表。

databaseId ：如果配置了databaseidProvider (4.6 节有详细配置方法） ， MyBatis

会加载所有的不带databaseid 的或匹配当前databaseid 的语句。如果同时存在带

databaseid 和不带databaseid 的语句，后者会被忽略。

由于数据库区分date 、time 、datetime 类型，但是Java 中一般都使用j ava.ut 工l. Date

类型。因此为了保证数据类型的正确，需要手动指定日期类型， date 、time 、datetime 对

应的JDBC 类型分别为DATE 、TIME 、TIMESTAMP 。

<insert id="insert">  
 insert into sys\_user(  
 user\_name, user\_password, user\_email,  
 user\_info, head\_img, create\_time)  
 values(  
 #{userName}, #{userPassword}, #{userEmail},  
 #{userInfo}, #{headImg, jdbcType=BLOB}, #{createTime, jdbcType=TIMESTAMP})  
</insert>

#{createTime, jdbcType=TIMESTAMP} 插入的create\_time 的格式是 2019-05-08 23:41:51.602(Timestamp)

#{createTime, jdbcType=DATE} 插入的create\_time 的格式是 2019-05-08(Date)

<insert id="insert2" useGeneratedKeys="true" keyProperty="id">

当需要设置多个属性时，使用逗号隔开，这种情况下通常还需要设置keyCo lumn 属性， 按顺

序指定数据库的列，这里列的值会和keyProperty 配置的属性一一对应

有些数据库（如Oracle）不提供主键自增的功能，而是使用序列得到一个值可以采用另外一种方式,使用＜selectKey>标签来获取主键的值

<selectKey keyColumn= ” id” resultType= ” long ” keyProperty= ” id” order= ” AFTER” >

SELECT LAST INSERT ID ()

< /selectKey>

selectKey 标签的ke yColumn 、keyProperty 和上面useGeneratedKeys 的用法含

义相同，这里的resultType 用于设置返回值类型。order 属性的设置和使用的数据库有关。在MySQL 数据库中， order 属性设置的值是AFTER ，因为当前记录的主键值在insert 语句

执行成功后才能获取到。而在Oracle 数据库中， order 的值要设置为BEFORE ，这是因为Oracle中需要先从序列获取值，然后将值作为主键插入到数据库中。

Mapper接口方法的参数

现目前所列举的接口中方法的参数只有一个，参数的类型可以分为两种：

一种是基本类型，另一种是JavaBean 。当参数是一个基本类型的时候，它在XML 文件中对应的SQL 语句只会使用一个参数，例

如delete 方法。当参数是一个JavaBean 类型的时候，它在XML 文件中对应的SQL 语句会有多个参数，例如insert 、update 方法。

JavaBean 类型参数相当于把多个参数合并到一个调用参数中，但是不能只为了两三个参数去创建新的JavaB ean 类，因此对于参数比较

少的情况，还有两种方式可以采用：使用Map 类型作为参数或使用＠ Par am 注解。

在Map 中通过key 来映射XML 中SQL 使用的参数

值名字， value 用来存放参数值，由于这种方式还需要自己手动创建Map 以及对参数进行赋值，其实并不简洁，

List<SysRole> selectRolesByUserIdAndRoleEnabled(@Param("userId") Long userId, @Param("enabled") Integer enabled);  
 接口方法参数如果大于1个，要使用@Param注解参数。否则mybatis会推断 xml中的参数名，第一个参数对应  
 #{0}或#{param1} ，第二个参数对应 #{1}或#{param2}  
 给参数配置@Param注解后， MyBatis 就会自动将参数封装成Map 类型，@Param注解值  
会作为Map 中的key

多个参数，且参数不是简单类型  
 List<SysRole> selectRolesByUserAndRole(@Param("user") SysUser user, @Param("role") SysRole role);  
xml中的参数写法  
 u.id = #{user.id} and r.enabled = #{role.enabled}

Mapper 接口动态代理实现原理

为什么Mapper 接口没有实现类却能被正常调用呢？这是因为MyBaits 在Mapper 接口上使用了动态代理的一种非常规的用法，熟悉这种动态代理的用法有利于理解MyBatis 接口和XML 的关系

MyBatis 的动态SQL 在XML 中支持的几种标签。

• if

• choose (when 、oterwise)

• trim (where 、set)

• foreach

• bind

if 标签有一个必填的属性test, test 的属性值是一个符合OGNL 要求的判断表达式，表达式的结果可以是true 或false ，除此之外所有的非0值都为true ，对字符串的判断和Java 中的判断类似，首先需要判断字段是否为null ，然后再去判断是否为空（在OGNL 表达式中，这两个判断的顺序不会影响判断的结果，也不会有空指针异常

if 标签提供了基本的条件判断，但是它无法实现if. . . else 、if ... else ...的逻辑，要想实现这样的逻辑，就需要用到choose when otherwise 标签。一个choose 中至少有一个when ，有0个或者1个otherwise

当参数id 有值的时候优先使用id 查询，当id 没有值时就去判断用户名是否有值，如果有值就用用户名查询，如果用户名

也没有值，就使SQ L 查询无结果。

<choose>  
 <when test="id != null">  
 and id = #{id}  
 </when>  
 <when test="userName != null and userName != ''">  
 and user\_name = #{userName}  
 </when>  
 <otherwise>  
 limit 0  
 </otherwise>  
</choose>

where 、set 、trim 用法

这3 个标签解决了类似的问题，并且where 和set 都属于trim 的一种具体用法

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后面的字符串是以AND 和OR 开头的，就将它们剔除。

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 后面的字符串是以逗号结尾的（开头的逗号不去），就将这个逗号剔除，用于update 语句

where 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。

where 标签对应trim 的实现如下。

<trim prefix=”WHERE ” prefixOverrides=” AND |OR ” >

</ trim>

set 标签对应的trim 实现如下。

<trim prefix=” SET ” suffixOverrides=”, ” >

</ trim>

trim 标签有如下属性。

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 指定的后缀。

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 用法

SQL 语句中有时会使用IN 关键字，例如id in ( 1 , 2 , 3 ）。可以使用${ids｝方式直接获

取值，但这种写法不能防止SQL 注入，想避免SQL 注入就需要用＃｛｝的方式，这时就要配合使

用foreach 标签来满足需求。

foreach 可以对数组、Map 或实现了工terable 接口（如List 、Set ）的对象进行遍

历。数组在处理时会转换为List 对象，因此foreach 遍历的对象可以分为两大类： Itera ble

类型和Map 类型。这两种类型在遍历循环时情况不一样，这一节会通过3 个例子来讲解

foreach 的用法。

foreach 可以对数组、Map 或实现了工terable 接口（如List 、Set ）的对象进行遍

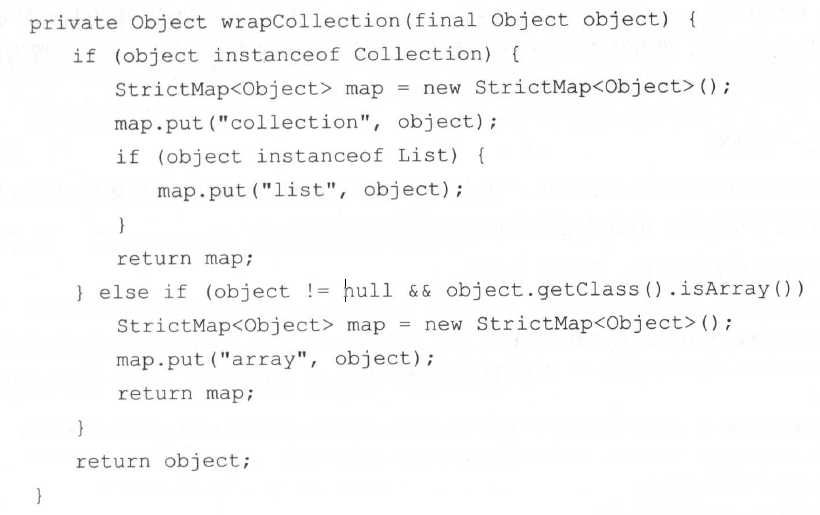
历。数组在处理时会转换为List 对象，因此foreach 遍历的对象可以分为两大类： Itera ble

类型和Map 类型。

foreach 包含以下属性。

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

只有一个数组或集合参数时，Mybatis对传给foreach参数的处理入下



这个属性的值反应了集合的类型。

如果传入的参数类型是(object instanceof Collection)，默认会转换为Map 类型，井添加一个key 为collection 的值。map.put(“collection”,obj)，如果传入的参数类型是List 集合，那么就继续添加一个key 为list的值，当参数类型为数组的时候，也会转换成Map 类型，默认的key 为array。

当接口方法使用map参数时，将collection 指定为对应Map中的key即可。这个key是索引map类型参数中，用于索引要迭代的集合

当接口方法有多个参数，要使用＠Param注解给每个参数指定一个名字，因此将collection 设置为＠Param注解指定的名字即可

当接口方法参数是一个对象，这种情况下指定为对象的属性名即可。当使用对象内多层嵌套的对象时，使用属性．属性（集

合和数组可以使用下标取值）的方式可以指定深层的属性值。

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， 当迭代循环的对象是Map类型时，这个值为Map 的key （键值）。

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 实现批量插入

如果数据库支持批量插入，就可以通过foreach 来实现

。批量插入的语法如下。

INSERT INTO tablename (column- a , [ column-b, ... ] )

VALUES (’ value-la ’, [ ’ value -lb ’,...]) ,

( ’ value-2a ’, [ ’ value- 2b ’,... ]) ,

Foreach 实现动态UPDATE

<update id="updateByMap">  
 update sys\_user   
 set   
 <foreach collection="\_parameter" item="val" index="key" separator=",">  
 ${key} = #{val}  
 </foreach>  
 where id = #{id}  
</update>

这里没有通过@Param 注解指定参数名，因而MyBatis 在内部的上下文中使用了默认值 \_parameter 作为该参数的key

bind 用法

bind 标签可以使用OGNL 表达式创建一个变量井将其绑定到上下文中。

多数据库支持

这需要用到if 标签以及由MyBatis 提供的databaseidProvider 数据库厂商标识配置。

MyBatis 会加载不带database Id 属性和带有匹配当前数据库databaseld 属性的所有语句。如果同时找到带有databaseId 和不带databaseId 的相同id语句，则后者会被舍弃。

为支持多厂商特性，只要像下面这样在mybatis-config.xml 文件中加入databaseidProvider 配置即可。

<databaseidProvider type = ” DB\_VENDOR ” />

一对一映射

关联查询两张表，user实体代表sys\_user, role实体代表sys\_role。定义a实体类时，将role实体的引用作为user实体的一个字段

使用自动映射处理一对一关系，查询的列别名要和实体的字段名一样。Role上的字段，对应的别名要使用 “role.字段名“ 的方式

r.enabled "role.enabled",  
r.create\_by "role.createBy",  
r.create\_time "role.createTime"

使用resultMap 配置一对一映射

resultMap配置可以使用extends属性继承其他resultMap的配置

<**resultMap id="userRoleMap2" extends="userMapBase" type="com.lile.mybatisExer.model.SysUser"**>  
 <**result property="role.id" column="role\_id"** />

</**resultMap**>

使用resultMap 的*association*标签配置一对一映射

在resultMap 中，*association* 标签用于和一个复杂的类型进行关联，即用于一对一的关联配置。

association 标签包含以下属性。

property ：对应实体类中的属性名，必填项。实体类中的实体类属性名

javaType ： 属性对应的Java 类型。

resultMap ： 可以直接使用现有的resultMap ，而不需要在这里配置。

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。

<**resultMap id="userMapBase3" extends="userMapBase" type="com.lile.mybatisExer.model.SysUser"**>  
 <**association property="role" columnPrefix="role\_"** >  
 <**result property="id" column="id"** />   
 </**association**>  
</**resultMap**> result标签内省略了**role实体名，省略了列别名前缀role\_**

一对一查询，嵌套查询，

association 标签的嵌套查询常用的属性如下。

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={propl=coll , prop2=col2}, propl 和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。

**collection**集合的嵌套结果映射

**collection** 支持的属性以及属性的作用和association 完全相同，这里的**javaType是集合的类型。ofType是集合元素的来行，**在一般情况下，MyBatis 可以推断 javaType 属性，因此并不需要填写

<**collection property="roleList" columnPrefix="role\_" javaType="ArrayList" ofType="com.lile.mybatisExer.model.SysRole"**>

MyBatis 在处理一对多查询到的数据时，会判断结果是否相同，如果是相同的结果，则只会保留第一

个结果。M y Batis 判断结果是否

相同时，最简单的情况就是在映射配置中至少有一个id 标签，

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

我们对id （构造方法中为idArg ）的理解一般是，它配置的字段为表的主键（联合主键

时可以配置多个id 标签），id 的唯一作用就是在嵌套的映射配置时判断数据是否相同，当配置工d 标签

时， MyB atis 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id 标签可以提高处理效率。

MyBatis

会对嵌套查询的每一级对象都进行属性比较。MyBatis 会首先比较顶层的对象，如果SysUser

部分相同，就继续比较SysRole 部分， 如果SysRole 不同，就会增加一个SysRole ，两个

SysRole 相同就保留前一个。假设SysRole 还有下一级，仍然按照该规则去比较。

这里要特别注意sys\_privilege 表中列的别名，因为sys\_privilege 嵌套在

rolePrivilegeListMap 中，而rolePrivilegeListMap 的前缀是“ role ”，所以

rolePrivilegeListMap 中**privilegeMap** 的前缀就变成了**privilege\_**

<**resultMap id="userRoleListMap" extends="userMapBase" type="com.lile.mybatisExer.model.SysUser"**>  
 <**collection property="roleList" columnPrefix="role\_"  
 resultMap="com.lile.mybatisExer.mapper.RoleMapper.rolePrivilegeListMap"**/>

</**resultMap**>

<**resultMap id="rolePrivilegeListMap" extends="roleMap" type="com.lile.mybatisExer.model.SysRole"**>  
 <**collection property="privilegeList" columnPrefix="privilege\_"  
 resultMap="com.lile.mybatisExer.mapper.PrivilegeMapper.privilegeMap"**/>  
</**resultMap**>

**collection**集合的嵌套查询

鉴别器映射

根据某列数据的值，选择不同的映射。跟switch 类似

根据enabled列的值，选择映射。按int类型比较。

<**resultMap id="rolePrivilegeListMapChoose" type="com.lile.mybatisExer.model.SysRole"**>  
 <**discriminator column="enabled" javaType="int"**>  
 <**case value="1" resultMap="rolePrivilegeListMapSelect"**/>  
 <**case value="0" resultMap="roleMap"**/>  
 </**discriminator**>  
</**resultMap**>

存储过程

OUT 模式的参数必须指定jdbcType 。这是因为在IN 模式下， MyBatis 提供了默认的jdbcType ，在OUT 模式下没有提供

<**select id="selectUserById" statementType="CALLABLE" useCache="false"**>  
 {call select\_user\_by\_id(  
 #{id, mode=IN},  
 #{userName, mode=OUT, jdbcType=VARCHAR},  
 #{userPassword, mode=OUT, jdbcType=VARCHAR},  
 #{userEmail, mode=OUT, jdbcType=VARCHAR},  
 #{userInfo, mode=OUT, jdbcType=VARCHAR},  
 #{headImg, mode=OUT, jdbcType=BLOB, javaType=\_byte[]},  
 #{createTime, mode=OUT, jdbcType=TIMESTAMP}  
 )}  
</**select**>

使用出参方式时，通常情况下会使用对象中的属性接收出参的值，或者使用Map 类型接收返回值。当使用JavaBean 对象接收出参时，必须保证所有出参在JavaBean 中都有对应的字段存在

*因为调用存储过程返回查询结果集， select 标签需设置resultMap，将结果映射到List<SysUser>*

在数据库中不存在一个和E na b l ed 枚举对应的数据库类型，因此在和数据库交互的时候，

不能直接使用枚举类型，在查询数据时， 需要将数据库int 类型的值转换为Java 中的枚举值。

在保存、更新数据或者作为查询条件时，需要将枚举值转换为数据库中的int 类型。

MyBatis 在处理Java 类型和数据库类型时，使用TypeHandler （类型处理器）对这两者

进行转换

Mybatis 为Java 和数据库JDB C 中的基本类型和常用的类型提供了TypeHandler

接口的实现。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

转换为字符串类型的字面值井使用，对于Enabled 而言便是”disabled “和”enabled “字符

串。在这个例子中，由于数据库使用的是int类型，所以在Java 的String 类型和数据库int 类

型互相转换时，