freesql语法

sql语句

## 使用select实现聚集

 用法和联合很类似，区别在于，这是一对多，所以一般映射过来的都是列表。所以这里需要定义javaType为ArrayList，还需要定义列表中对象的类型ofType，以及必须设置的select的语句名称（需要注意的是，这里的查询student的select语句条件必须是外键classID）。

ClassMapper.xml文件部分内容：

<resultMap type="ClassEntity" id="classResultMap">

<id property="classID" column="CLASS\_ID" />

<result property="className" column="CLASS\_NAME" />

<result property="classYear" column="CLASS\_YEAR" />

<association property="teacherEntity" column="TEACHER\_ID" select="getTeacher"/>

<collection property="studentList" column="CLASS\_ID" javaType="ArrayList" ofType="StudentEntity" select="getStudentByClassID"/>

</resultMap>

<select id="getClassByID" parameterType="String" resultMap="classResultMap">

SELECT \* FROM CLASS\_TBL CT

WHERE CT.CLASS\_ID = #{classID};

</select>

<!-- 查询学生list，根据班级id -->

**<select** id="getStudentByClassID" parameterType="String" resultMap="studentResultMap"**>**

**<include** refid="selectStudentAll" **/>**

    WHERE ST.CLASS\_ID = #{classID}

**</select>**

## 使用resultMap实现聚集

 使用resultMap，就需要重写一个sql，left join学生表。

<resultMap type="ClassEntity" id="classResultMap">

<id property="classID" column="CLASS\_ID" />

<result property="className" column="CLASS\_NAME" />

<result property="classYear" column="CLASS\_YEAR" />

<association property="teacherEntity" column="TEACHER\_ID" resultMap="teacherResultMap"/>

<collection property="studentList" column="CLASS\_ID" javaType="ArrayList" ofType="StudentEntity" resultMap="studentResultMap"/>

</resultMap>

<select id="getClassAndTeacherStudent" parameterType="String" resultMap="classResultMap">

SELECT \*

FROM CLASS\_TBL CT

LEFT JOIN STUDENT\_TBL ST

ON CT.CLASS\_ID = ST.CLASS\_ID

LEFT JOIN TEACHER\_TBL TT

ON CT.TEACHER\_ID = TT.TEACHER\_ID

WHERE CT.CLASS\_ID = #{classID};

</select>

## discriminator鉴别器

 有时一个单独的数据库查询也许返回很多不同（但是希望有些关联）数据类型的结果集。鉴别器元素就是被设计来处理这个情况的，还有包括类的继承层次结构。鉴别器非常容易理解，因为它的表现很像Java语言中的switch语句。

定义鉴别器指定了column和javaType属性。列是MyBatis查找比较值的地方。JavaType是需要被用来保证等价测试的合适类型（尽管字符串在很多情形下都会有用）。

下面这个例子为，当classId为20000001时，才映射classId属性。

<resultMap type="liming.student.manager.data.model.StudentEntity" id="resultMap\_studentEntity\_discriminator">

<id property="studentId" column="STUDENT\_ID" javaType="String" jdbcType="VARCHAR"/>

<result property="studentName" column="STUDENT\_NAME" javaType="String" jdbcType="VARCHAR"/>

<result property="studentSex" column="STUDENT\_SEX" javaType="int" jdbcType="INTEGER"/>

<result property="studentBirthday" column="STUDENT\_BIRTHDAY" javaType="Date" jdbcType="DATE"/>

<result property="studentPhoto" column="STUDENT\_PHOTO" javaType="byte[]" jdbcType="BLOB" typeHandler="org.apache.ibatis.type.BlobTypeHandler" />

<result property="placeId" column="PLACE\_ID" javaType="String" jdbcType="VARCHAR"/>

<discriminator column="CLASS\_ID" javaType="String" jdbcType="VARCHAR">

<case value="20000001" resultType="liming.student.manager.data.model.StudentEntity" >

<result property="classId" column="CLASS\_ID" javaType="String" jdbcType="VARCHAR"/>

</case>

</discriminator>

</resultMap>

## selectKey 标签

       在insert语句中，在Oracle经常使用序列、在MySQL中使用函数来自动生成插入表的主键，而且需要方法能返回这个生成主键。使用myBatis的selectKey标签可以实现这个效果。

       下面例子，使用mysql数据库自定义函数nextval('student')，用来生成一个key，并把他设置到传入的实体类中的studentId属性上。所以在执行完此方法后，边可以通过这个实体类获取生成的key。

<!-- 插入学生 自动主键-->

<insert id="createStudentAutoKey" parameterType="liming.student.manager.data.model.StudentEntity" keyProperty="studentId">

<selectKey keyProperty="studentId" resultType="String" order="BEFORE">

select nextval('student')

</selectKey>

INSERT INTO STUDENT\_TBL(STUDENT\_ID,

STUDENT\_NAME,

STUDENT\_SEX,

STUDENT\_BIRTHDAY,

STUDENT\_PHOTO,

CLASS\_ID,

PLACE\_ID)

VALUES (#{studentId},

#{studentName},

#{studentSex},

#{studentBirthday},

#{studentPhoto, javaType=byte[], jdbcType=BLOB, typeHandler=org.apache.ibatis.type.BlobTypeHandler},

#{classId},

#{placeId})

</insert>

调用接口方法，和获取自动生成key

StudentEntity entity = new StudentEntity();

entity.setStudentName("黎明你好");

entity.setStudentSex(1);

entity.setStudentBirthday(DateUtil.parse("1985-05-28"));

entity.setClassId("20000001");

entity.setPlaceId("70000001");

this.dynamicSqlMapper.createStudentAutoKey(entity);

System.out.println("新增学生ID: " + entity.getStudentId());

selectKey语句属性配置细节：

|  |  |  |
| --- | --- | --- |
| 属性 | 描述 | 取值 |
| keyProperty | selectKey 语句生成结果需要设置的属性。 |  |
| resultType | 生成结果类型，MyBatis 允许使用基本的数据类型，包括String 、int类型。 |  |
| order | 1：BEFORE，会先选择主键，然后设置keyProperty，再执行insert语句；  2：AFTER，就先运行insert 语句再运行selectKey 语句。 | BEFORE  AFTER |
| statementType | MyBatis 支持STATEMENT，PREPARED和CALLABLE 的语句形式， 对应Statement ，PreparedStatement 和CallableStatement 响应 | STATEMENT  PREPARED  CALLABLE |
|  | |  |  | |

### if标签

 if标签可用在许多类型的sql语句中，我们以查询为例。首先看一个很普通的查询：

<!-- 查询学生list，like姓名 -->

<select id="getStudentListLikeName" parameterType="StudentEntity" resultMap="studentResultMap">

SELECT \* from STUDENT\_TBL ST

WHERE ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName}),'%')

</select>

但是此时如果studentName或studentSex为null，此语句很可能报错或查询结果为空。此时我们使用if动态sql语句先进行判断，如果值为null或等于空字符串，我们就不进行此条件的判断，增加灵活性。

参数为实体类StudentEntity。将实体类中所有的属性均进行判断，如果不为空则执行判断条件。

<!-- 2 if(判断参数) - 将实体类不为空的属性作为where条件 -->

<select id="getStudentList\_if" resultMap="resultMap\_studentEntity" parameterType="liming.student.manager.data.model.StudentEntity">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.STUDENT\_PHOTO,

ST.CLASS\_ID,

ST.PLACE\_ID

FROM STUDENT\_TBL ST

WHERE

<if test="studentName !=null ">

ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName, jdbcType=VARCHAR}),'%')

</if>

<if test="studentSex != null and studentSex != '' ">

AND ST.STUDENT\_SEX = #{studentSex, jdbcType=INTEGER}

</if>

<if test="studentBirthday != null ">

AND ST.STUDENT\_BIRTHDAY = #{studentBirthday, jdbcType=DATE}

</if>

<if test="classId != null and classId!= '' ">

AND ST.CLASS\_ID = #{classId, jdbcType=VARCHAR}

</if>

<if test="classEntity != null and classEntity.classId !=null and classEntity.classId !=' ' ">

AND ST.CLASS\_ID = #{classEntity.classId, jdbcType=VARCHAR}

</if>

<if test="placeId != null and placeId != '' ">

AND ST.PLACE\_ID = #{placeId, jdbcType=VARCHAR}

</if>

<if test="placeEntity != null and placeEntity.placeId != null and placeEntity.placeId != '' ">

AND ST.PLACE\_ID = #{placeEntity.placeId, jdbcType=VARCHAR}

</if>

<if test="studentId != null and studentId != '' ">

AND ST.STUDENT\_ID = #{studentId, jdbcType=VARCHAR}

</if>

</select>

使用时比较灵活， new一个这样的实体类，我们需要限制那个条件，只需要附上相应的值就会where这个条件，相反不去赋值就可以不在where中判断。

public void select\_test\_2\_1() {

StudentEntity entity = new StudentEntity();

entity.setStudentName("");

entity.setStudentSex(1);

entity.setStudentBirthday(DateUtil.parse("1985-05-28"));

entity.setClassId("20000001");

//entity.setPlaceId("70000001");

List<StudentEntity> list = this.dynamicSqlMapper.getStudentList\_if(entity);

for (StudentEntity e : list) {

System.out.println(e.toString());

}

}

### if + where 的条件判断

当where中的条件使用的if标签较多时，这样的组合可能会导致错误。我们以在3.1中的查询语句为例子，当java代码按如下方法调用时：

@Test

public void select\_test\_2\_1() {

StudentEntity entity = new StudentEntity();

entity.setStudentName(null);

entity.setStudentSex(1);

List<StudentEntity> list = this.dynamicSqlMapper.getStudentList\_if(entity);

for (StudentEntity e : list) {

System.out.println(e.toString());

}

}

如果上面例子，参数studentName为null，将不会进行STUDENT\_NAME列的判断，则会直接导“WHERE AND”关键字多余的错误SQL。

这时我们可以使用where动态语句来解决。这个“where”标签会知道如果它包含的标签中有返回值的话，它就插入一个‘where’。此外，如果标签返回的内容是以AND 或OR 开头的，则它会剔除掉。

上面例子修改为：

<!-- 3 select - where/if(判断参数) - 将实体类不为空的属性作为where条件 -->

<select id="getStudentList\_whereIf" resultMap="resultMap\_studentEntity" parameterType="liming.student.manager.data.model.StudentEntity">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.STUDENT\_PHOTO,

ST.CLASS\_ID,

ST.PLACE\_ID

FROM STUDENT\_TBL ST

<where>

<if test="studentName !=null ">

ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName, jdbcType=VARCHAR}),'%')

</if>

<if test="studentSex != null and studentSex != '' ">

AND ST.STUDENT\_SEX = #{studentSex, jdbcType=INTEGER}

</if>

<if test="studentBirthday != null ">

AND ST.STUDENT\_BIRTHDAY = #{studentBirthday, jdbcType=DATE}

</if>

<if test="classId != null and classId!= '' ">

AND ST.CLASS\_ID = #{classId, jdbcType=VARCHAR}

</if>

<if test="classEntity != null and classEntity.classId !=null and classEntity.classId !=' ' ">

AND ST.CLASS\_ID = #{classEntity.classId, jdbcType=VARCHAR}

</if>

<if test="placeId != null and placeId != '' ">

AND ST.PLACE\_ID = #{placeId, jdbcType=VARCHAR}

</if>

<if test="placeEntity != null and placeEntity.placeId != null and placeEntity.placeId != '' ">

AND ST.PLACE\_ID = #{placeEntity.placeId, jdbcType=VARCHAR}

</if>

<if test="studentId != null and studentId != '' ">

AND ST.STUDENT\_ID = #{studentId, jdbcType=VARCHAR}

</if>

</where>

</select>

### if + set 的更新语句

当update语句中没有使用if标签时，如果有一个参数为null，都会导致错误。

当在update语句中使用if标签时，如果前面的if没有执行，则或导致逗号多余错误。使用set标签可以将动态的配置SET 关键字，和剔除追加到条件末尾的任何不相关的逗号。

       使用if+set标签修改后，如果某项为null则不进行更新，而是保持数据库原值。如下示例：

<!-- 4 if/set(判断参数) - 将实体类不为空的属性更新 -->

<update id="updateStudent\_if\_set" parameterType="liming.student.manager.data.model.StudentEntity">

UPDATE STUDENT\_TBL

<set>

<if test="studentName != null and studentName != '' ">

STUDENT\_TBL.STUDENT\_NAME = #{studentName},

</if>

<if test="studentSex != null and studentSex != '' ">

STUDENT\_TBL.STUDENT\_SEX = #{studentSex},

</if>

<if test="studentBirthday != null ">

STUDENT\_TBL.STUDENT\_BIRTHDAY = #{studentBirthday},

</if>

<if test="studentPhoto != null ">

STUDENT\_TBL.STUDENT\_PHOTO = #{studentPhoto, javaType=byte[], jdbcType=BLOB, typeHandler=org.apache.ibatis.type.BlobTypeHandler},

</if>

<if test="classId != '' ">

STUDENT\_TBL.CLASS\_ID = #{classId}

</if>

<if test="placeId != '' ">

STUDENT\_TBL.PLACE\_ID = #{placeId}

</if>

</set>

WHERE STUDENT\_TBL.STUDENT\_ID = #{studentId};

</update>

### if + trim代替where/set标签

   trim是更灵活的去处多余关键字的标签，他可以实践where和set的效果。

#### trim代替where

if/trim代替where(判断参数) - 将实体类不为空的属性作为where条件

<select id="getStudentList\_if\_trim" resultMap="resultMap\_studentEntity">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.STUDENT\_PHOTO,

ST.CLASS\_ID,

ST.PLACE\_ID

FROM STUDENT\_TBL ST

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

<if test="studentName !=null ">

ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%',#{studentName, jdbcType=VARCHAR}),'%')

</if>

<if test="studentSex != null and studentSex != '' ">

AND ST.STUDENT\_SEX = #{studentSex, jdbcType=INTEGER}

</if>

<if test="studentBirthday != null ">

AND ST.STUDENT\_BIRTHDAY = #{studentBirthday, jdbcType=DATE}

</if>

<if test="classId != null and classId!= '' ">

AND ST.CLASS\_ID = #{classId, jdbcType=VARCHAR}

</if>

<if test="classEntity != null and classEntity.classId !=null and classEntity.classId !=' ' ">

AND ST.CLASS\_ID = #{classEntity.classId, jdbcType=VARCHAR}

</if>

<if test="placeId != null and placeId != '' ">

AND ST.PLACE\_ID = #{placeId, jdbcType=VARCHAR}

</if>

<if test="placeEntity != null and placeEntity.placeId != null and placeEntity.placeId != '' ">

AND ST.PLACE\_ID = #{placeEntity.placeId, jdbcType=VARCHAR}

</if>

<if test="studentId != null and studentId != '' ">

AND ST.STUDENT\_ID = #{studentId, jdbcType=VARCHAR}

</if>

</trim>

</select>

#### trim代替set

if/trim代替set(判断参数) - 将实体类不为空的属性更新

<update id="updateStudent\_if\_trim" parameterType="liming.student.manager.data.model.StudentEntity">

UPDATE STUDENT\_TBL

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

<if test="studentName != null and studentName != '' ">

STUDENT\_TBL.STUDENT\_NAME = #{studentName},

</if>

<if test="studentSex != null and studentSex != '' ">

STUDENT\_TBL.STUDENT\_SEX = #{studentSex},

</if>

<if test="studentBirthday != null ">

STUDENT\_TBL.STUDENT\_BIRTHDAY = #{studentBirthday},

</if>

<if test="studentPhoto != null ">

STUDENT\_TBL.STUDENT\_PHOTO = #{studentPhoto, javaType=byte[], jdbcType=BLOB, typeHandler=org.apache.ibatis.type.BlobTypeHandler},

</if>

<if test="classId != '' ">

STUDENT\_TBL.CLASS\_ID = #{classId},

</if>

<if test="placeId != '' ">

STUDENT\_TBL.PLACE\_ID = #{placeId}

</if>

</trim>

WHERE STUDENT\_TBL.STUDENT\_ID = #{studentId}

</update>

## choose (when, otherwise)

有时候我们并不想应用所有的条件，而只是想从多个选项中选择一个。而使用if标签时，只要test中的表达式为true，就会执行if标签中的条件。MyBatis提供了choose 元素。if标签是与(and)的关系，而choose比傲天是或（or）的关系。

    choose标签是按顺序判断其内部when标签中的test条件出否成立，如果有一个成立，则choose结束。当choose中所有when的条件都不满则时，则执行otherwise中的sql。类似于Java 的switch 语句，choose为switch，when为case，otherwise则为default。

    例如下面例子，同样把所有可以限制的条件都写上，方面使用。choose会从上到下选择一个when标签的test为true的sql执行。安全考虑，我们使用where将choose包起来，放置关键字多于错误。

<!-- 6 choose(判断参数) - 按顺序将实体类第一个不为空的属性作为where条件 -->

<select id="getStudentList\_choose" resultMap="resultMap\_studentEntity" parameterType="liming.student.manager.data.model.StudentEntity">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.STUDENT\_PHOTO,

ST.CLASS\_ID,

ST.PLACE\_ID

FROM STUDENT\_TBL ST

<where>

<choose>

<when test="studentName !=null ">

ST.STUDENT\_NAME LIKE CONCAT(CONCAT('%', #{studentName, jdbcType=VARCHAR}),'%')

</when >

<when test="studentSex != null and studentSex != '' ">

AND ST.STUDENT\_SEX = #{studentSex, jdbcType=INTEGER}

</when >

<when test="studentBirthday != null ">

AND ST.STUDENT\_BIRTHDAY = #{studentBirthday, jdbcType=DATE}

</when >

<when test="classId != null and classId!= '' ">

AND ST.CLASS\_ID = #{classId, jdbcType=VARCHAR}

</when >

<when test="classEntity != null and classEntity.classId !=null and classEntity.classId !=' ' ">

AND ST.CLASS\_ID = #{classEntity.classId, jdbcType=VARCHAR}

</when >

<when test="placeId != null and placeId != '' ">

AND ST.PLACE\_ID = #{placeId, jdbcType=VARCHAR}

</when >

<when test="placeEntity != null and placeEntity.placeId != null and placeEntity.placeId != '' ">

AND ST.PLACE\_ID = #{placeEntity.placeId, jdbcType=VARCHAR}

</when >

<when test="studentId != null and studentId != '' ">

AND ST.STUDENT\_ID = #{studentId, jdbcType=VARCHAR}

</when >

<otherwise>

</otherwise>

</choose>

</where>

</select>

## foreach

对于动态SQL 非常必须的，主是要迭代一个集合，通常是用于IN 条件。List 实例将使用“list”做为键，数组实例以“array” 做为键。

foreach元素是非常强大的，它允许你指定一个集合，声明集合项和索引变量，它们可以用在元素体内。它也允许你指定开放和关闭的字符串，在迭代之间放置分隔符。这个元素是很智能的，它不会偶然地附加多余的分隔符。

注意：你可以传递一个List实例或者数组作为参数对象传给MyBatis。当你这么做的时候，MyBatis会自动将它包装在一个Map中，用名称在作为键。List实例将会以“list”作为键，而数组实例将会以“array”作为键。

这个部分是对关于XML配置文件和XML映射文件的而讨论的。下一部分将详细讨论Java API，所以你可以得到你已经创建的最有效的映射。

### 3.7.1参数为array示例的写法

接口的方法声明：

public List<StudentEntity> getStudentListByClassIds\_foreach\_array(String[] classIds);

public List<StudentEntity> getStudentListByClassIds\_foreach\_array(String[] classIds);

动态SQL语句：

foreach(循环array参数) - 作为where中in的条**<select** id="getStudentListByClassIds\_foreach\_array" resultMap="resultMap\_studentEntity"**>**

    SELECT ST.STUDENT\_ID,

           ST.STUDENT\_NAME,

           ST.STUDENT\_SEX,

           ST.STUDENT\_BIRTHDAY,

           ST.STUDENT\_PHOTO,

           ST.CLASS\_ID,

           ST.PLACE\_ID

      FROM STUDENT\_TBL ST

      WHERE ST.CLASS\_ID IN

**<foreach** collection="array" item="classIds"  open="(" separator="," close=")"**>**

        #{classIds}

**</foreach>**

**</select>**

<!— 7.1 foreach(循环array参数) - 作为where中in的条件 -->

<select id="getStudentListByClassIds\_foreach\_array" resultMap="resultMap\_studentEntity">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.STUDENT\_PHOTO,

ST.CLASS\_ID,

ST.PLACE\_ID

FROM STUDENT\_TBL ST

WHERE ST.CLASS\_ID IN

<foreach collection="array" item="classIds" open="(" separator="," close=")">

#{classIds}

</foreach>

</select>

测试代码，查询学生中，在20000001、20000002这两个班级的学生：

@Test

public void test7\_foreach() {

    String[] classIds = { "20000001", "20000002" };

    List<StudentEntity> list = this.dynamicSqlMapper.getStudentListByClassIds\_foreach\_array(classIds);

    for (StudentEntity e : list) {

        System.out.println(e.toString());

    }

<P>}<SPAN style="WHITE-SPACE: normal; FONT-SIZE: 14px; FONT-WEIGHT: bold">  </SPAN></P>

@Test

public void test7\_foreach() {

String[] classIds = { "20000001", "20000002" };

List<StudentEntity> list = this.dynamicSqlMapper.getStudentListByClassIds\_foreach\_array(classIds);

for (StudentEntity e : list) {

System.out.println(e.toString());

}

}

### 1.6.2参数为list示例的写法

接口的方法声明：

public List<StudentEntity> getStudentListByClassIds\_foreach\_list(List<String> classIdList);

public List<StudentEntity> getStudentListByClassIds\_foreach\_list(List<String> classIdList);

动态SQL语句：

<!-- 7.2 foreach(循环List<String>参数) - 作为where中in的条件 -->

**<select** id="getStudentListByClassIds\_foreach\_list" resultMap="resultMap\_studentEntity"**>**

    SELECT ST.STUDENT\_ID,

           ST.STUDENT\_NAME,

           ST.STUDENT\_SEX,

           ST.STUDENT\_BIRTHDAY,

           ST.STUDENT\_PHOTO,

           ST.CLASS\_ID,

           ST.PLACE\_ID

      FROM STUDENT\_TBL ST

      WHERE ST.CLASS\_ID IN

**<foreach** collection="list" item="classIdList"  open="(" separator="," close=")"**>**

        #{classIdList}

**</foreach>**

**</select>**

<!-- 7.2 foreach(循环List<String>参数) - 作为where中in的条件 -->

<select id="getStudentListByClassIds\_foreach\_list" resultMap="resultMap\_studentEntity">

SELECT ST.STUDENT\_ID,

ST.STUDENT\_NAME,

ST.STUDENT\_SEX,

ST.STUDENT\_BIRTHDAY,

ST.STUDENT\_PHOTO,

ST.CLASS\_ID,

ST.PLACE\_ID

FROM STUDENT\_TBL ST

WHERE ST.CLASS\_ID IN

<foreach collection="list" item="classIdList" open="(" separator="," close=")">

#{classIdList}

</foreach>

</select>

测试代码，查询学生中，在20000001、20000002这两个班级的学生：

@Test

public void test7\_2\_foreach() {

    ArrayList<String> classIdList = new ArrayList<String>();

    classIdList.add("20000001");

    classIdList.add("20000002");

    List<StudentEntity> list = this.dynamicSqlMapper.getStudentListByClassIds\_foreach\_list(classIdList);

    for (StudentEntity e : list) {

        System.out.println(e.toString());

    }

}