提供的爬取软件来源于：i.ijrou.com 免费下载使用

你用过Mybatis的动态SQL后，就知道写SQL有多爽了！

点击关注 👉 [Java面试那些事儿](javascript:void(0);)

**Java面试那些事儿**

微信号 javatiku

功能介绍 一个能带你进入BATM大厂的技术号，这里不仅有面试题、内推渠道，还有Java技术分享，包括Intellij IDEA、多线程、JVM、Spring Cloud、Dubbo、Zookeeper、Redis、架构设计、微服务、消息队列等。

2020-07-23[原文](https://mp.weixin.qq.com/s?__biz=MzIzMzgxOTQ5NA==&mid=2247498718&idx=2&sn=6f1591c926abc5b03fae12bc2a977978&chksm=e8fd67d7df8aeec1fb7fd988fd3549d1ad28e910587d9565393873f9519210a8bd602fc13e8e&mpshare=1&scene=24&srcid=0802oLG9dXksmGtXsNBMzyiv&sharer_sharetime=1596380691900&sharer_shareid=cee05ae4c377edf328aa206f82e8dd39&key=d3c7cb8eeb23bc834fdd3d4dbe5193fc46e8d47c7ea4227f6ab88b989f517bf0979cc6b9fce3e4bbdcabe35a1efb0c12f20b76540e356e952e33504657307171c3aa613e84224c15ef5a67af7ec85052&ascene=14&uin=MTc4NTE2NTcyMA==&devicetype=Windows)



**# 一起来学习 mybatis**

MyBatis 令人喜欢的一大特性就是动态 SQL。 在使用 JDBC 的过程中， 根据条件进行 SQL 的拼接是很麻烦且很容易出错的。 MyBatis 动态 SQL 的出现， 解决了这个麻烦。

MyBatis通过 OGNL 来进行动态 SQL 的使用的。

目前， 动态 SQL 支持以下几种标签

| **元素** | **作用** | **备注** |
| --- | --- | --- |
| if | 判断语句 | 单条件分支 |
| choose(when、otherwise) | 相当于 Java 中的 if else | 多条件分支 |
| trim(where、set) | 辅助元素 | 用于处理 SQL 拼接问题 |
| foreach | 循环语句 | 批量插入， 更新， 查询时经常用到 |
| bind | 创建一个变量， 并绑定到上下文中 | 用于兼容不同的数据库， 防止 SQL 注入等 |

**1 数据准备**

为了后面的演示， 创建了一个 Maven 项目 mybatis-dynamic, 创建了对应的数据库和表

DROP TABLE IF EXISTS `student`;

CREATE TABLE `student` (

`student\_id` int(10) unsigned NOT NULL AUTO\_INCREMENT COMMENT '编号',

`name` varchar(20) DEFAULT NULL COMMENT '姓名',

`phone` varchar(20) DEFAULT NULL COMMENT '电话',

`email` varchar(50) DEFAULT NULL COMMENT '邮箱',

`sex` tinyint(4) DEFAULT NULL COMMENT '性别',

`locked` tinyint(4) DEFAULT NULL COMMENT '状态(0:正常,1:锁定)',

`gmt\_created` datetime DEFAULT CURRENT\_TIMESTAMP COMMENT '存入数据库的时间',

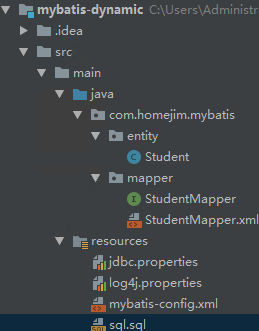
`gmt\_modified` datetime DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP COMMENT '修改的时间',

`delete` int(11) DEFAULT NULL,

PRIMARY KEY (`student\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci COMMENT='学生表';

对应的项目结构



**2 if 标签**

if 标签是我们最常使用的。在查询、删除、更新的时候很可能会使用到。必须结合 test 属性联合使用。

2.1 在 WHERE 条件中使用 if 标签

这是常见的一种现象， 我们在进行按条件查询的时候， 可能会有多种情况。

2.1.1 查询条件

根据输入的学生信息进行条件检索

1. 当只输入用户名时， 使用用户名进行模糊检索；
2. 当只输入性别时， 使用性别进行完全匹配
3. 当用户名和性别都存在时， 用这两个条件进行查询匹配查询

2.1.2 动态 SQL

接口函数

*/\*\**

*\* 根据输入的学生信息进行条件检索*

*\* 1. 当只输入用户名时， 使用用户名进行模糊检索；*

*\* 2. 当只输入邮箱时， 使用性别进行完全匹配*

*\* 3. 当用户名和性别都存在时， 用这两个条件进行查询匹配的用*

*\* @param student*

*\* @return*

*\*/*

List selectByStudentSelective(Student student);

对应的动态 SQL

<select id="selectByStudentSelective" resultMap="BaseResultMap" parameterType="com.homejim.mybatis.entity.Student">

select

"Base\_Column\_List" />

from student

where 1=1

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

and name like concat('%', #{name}, '%')

if>

<if test="sex != null">

and sex=#{sex}

if>

select>

在此 SQL 语句中， where 1=1 是多条件拼接时的小技巧， 后面的条件查询就可以都用 and 了。

同时， 我们添加了 if 标签来处理动态 SQL

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

and name like concat('%', *#{name}, '%')*

if>

<if test="sex != null">

and sex=*#{sex}*

if>

此 if 标签的 test 属性值是一个符合 OGNL 的表达式， 表达式可以是 true 或 false。如果表达式返回的是数值， 则0为 false, 非 0 为 true;

2.1.3 测试

@Test

public void selectByStudent() {

SqlSession sqlSession = null;

sqlSession = sqlSessionFactory.openSession();

StudentMapper studentMapper = sqlSession.getMapper(StudentMapper.class);

Student search = new Student();

search.setName("明");

System.out.println("只有名字时的查询");

List studentsByName = studentMapper.selectByStudentSelective(search);

for (int i = 0; i < studentsByName.size(); i++) {

System.out.println(ToStringBuilder.reflectionToString(studentsByName.get(i), ToStringStyle.MULTI\_LINE\_STYLE));

}

search.setName(null);

search.setSex((byte) 1);

System.out.println("只有性别时的查询");

List studentsBySex = studentMapper.selectByStudentSelective(search);

for (int i = 0; i < studentsBySex.size(); i++) {

System.out.println(ToStringBuilder.reflectionToString(studentsBySex.get(i), ToStringStyle.MULTI\_LINE\_STYLE));

}

System.out.println("姓名和性别同时存在的查询");

search.setName("明");

List studentsByNameAndSex = studentMapper.selectByStudentSelective(search);

for (int i = 0; i < studentsByNameAndSex.size(); i++) {

System.out.println(ToStringBuilder.reflectionToString(studentsByNameAndSex.get(i), ToStringStyle.MULTI\_LINE\_STYLE));

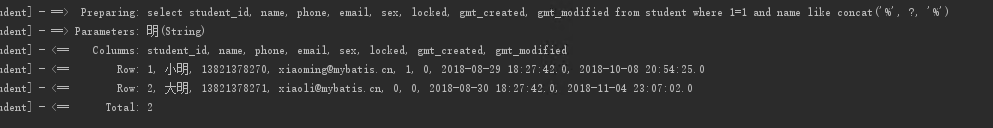
}

sqlSession.commit();

sqlSession.close();

}

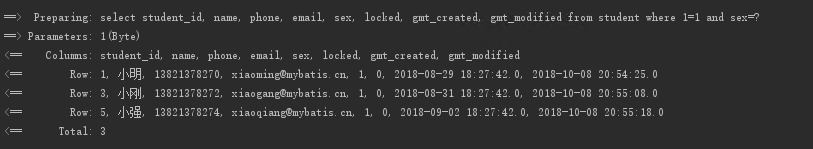
只有名字时的查询， 发送的语句和结果



查询的条件只发送了

where 1=1 and name like concat('%', ?, '%')

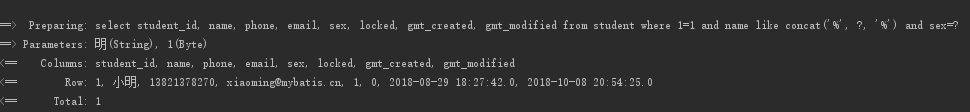
**只有性别时的查询**， 发送的语句和结果



查询的条件只发送了

where 1=1 and sex=?

姓名和性别同时存在的查询， 发送的语句和结果



查询条件

where 1=1 and name like concat('%', ?, '%') and sex=?

2.2 在 UPDATE 更新列中使用 if 标签

有时候我们不希望更新所有的字段， 只更新有变化的字段。

2.2.1 更新条件

只更新有变化的字段， 空值不更新。

2.2.1 动态 SQL

接口方法

*/\*\**

*\* 更新非空属性*

*\*/*

int updateByPrimaryKeySelective(Student record);

对应的 SQL

"updateByPrimaryKeySelective" parameterType="com.homejim.mybatis.entity.Student">

update student

<set>

<if test="name != null">

`name` = *#{name,jdbcType=VARCHAR},*

if>

<if test="phone != null">

phone = *#{phone,jdbcType=VARCHAR},*

if>

<if test="email != null">

email = *#{email,jdbcType=VARCHAR},*

if>

<if test="sex != null">

sex = *#{sex,jdbcType=TINYINT},*

if>

<if test="locked != null">

locked = *#{locked,jdbcType=TINYINT},*

if>

<if test="gmtCreated != null">

gmt\_created = *#{gmtCreated,jdbcType=TIMESTAMP},*

if>

<if test="gmtModified != null">

gmt\_modified = *#{gmtModified,jdbcType=TIMESTAMP},*

if>

set>

where student\_id = *#{studentId,jdbcType=INTEGER}*

2.2.3 测试

@Test

public void updateByStudentSelective() {

SqlSession sqlSession = null;

sqlSession = sqlSessionFactory.openSession();

StudentMapper studentMapper = sqlSession.getMapper(StudentMapper.class);

Student student = new Student();

student.setStudentId(1);

student.setName("明明");

student.setPhone("13838438888");

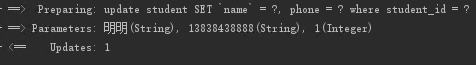
System.out.println(studentMapper.updateByPrimaryKeySelective(student));

sqlSession.commit();

sqlSession.close();

}

结果如下



2.3 在 INSERT 动态插入中使用 if 标签

我们插入数据库中的一条记录， 不是每一个字段都有值的， 而是动态变化的。在这时候使用 if 标签， 可帮我们解决这个问题。

2.3.1 插入条件

只有非空属性才插入。

2.3.2 动态SQL

接口方法

*/\*\**

*\* 非空字段才进行插入*

*\*/*

int insertSelective(Student record);

对应的SQL

"insertSelective" parameterType="com.homejim.mybatis.entity.Student">

insert into student

"(" suffix=")" suffixOverrides=",">

<if test="studentId != null">

student\_id,

if>

<if test="name != null">

`name`,

if>

<if test="phone != null">

phone,

if>

<if test="email != null">

email,

if>

<if test="sex != null">

sex,

if>

<if test="locked != null">

locked,

if>

<if test="gmtCreated != null">

gmt\_created,

if>

<if test="gmtModified != null">

gmt\_modified,

if>

"values (" suffix=")" suffixOverrides=",">

<if test="studentId != null">

*#{studentId,jdbcType=INTEGER},*

if>

<if test="name != null">

*#{name,jdbcType=VARCHAR},*

if>

<if test="phone != null">

*#{phone,jdbcType=VARCHAR},*

if>

<if test="email != null">

*#{email,jdbcType=VARCHAR},*

if>

<if test="sex != null">

*#{sex,jdbcType=TINYINT},*

if>

<if test="locked != null">

*#{locked,jdbcType=TINYINT},*

if>

<if test="gmtCreated != null">

*#{gmtCreated,jdbcType=TIMESTAMP},*

if>

<if test="gmtModified != null">

*#{gmtModified,jdbcType=TIMESTAMP},*

if>

这个 SQL 大家应该很熟悉， 毕竟是自动生成的。

2.3.3 测试

@Test

public void insertByStudentSelective() {

SqlSession sqlSession = null;

sqlSession = sqlSessionFactory.openSession();

StudentMapper studentMapper = sqlSession.getMapper(StudentMapper.class);

Student student = new Student();

student.setName("小飞机");

student.setPhone("13838438899");

student.setEmail("xiaofeiji@qq.com");

student.setLocked((byte) 0);

System.out.println(studentMapper.insertSelective(student));

sqlSession.commit();

sqlSession.close();

}

对应的结果



SQL 中， 只有非空的字段才进行了插入。

**3 choose 标签**

choose when otherwise 标签可以帮我们实现 if else 的逻辑。

一个 choose 标签至少有一个 when, 最多一个otherwise

下面是一个查询的例子。

3.1 查询条件

假设 name 具有唯一性， 查询一个学生

* 当 studen\_id 有值时， 使用 studen\_id 进行查询；
* 当 studen\_id 没有值时， 使用 name 进行查询；
* 否则返回空

3.2 动态SQL

接口方法

*/\*\**

*\* - 当 studen\_id 有值时， 使用 studen\_id 进行查询；*

*\* - 当 studen\_id 没有值时， 使用 name 进行查询；*

*\* - 否则返回空*

*\*/*

Student selectByIdOrName(Student record);

对应的SQL

<select id="selectByIdOrName" resultMap="BaseResultMap" parameterType="com.homejim.mybatis.entity.Student">

select

<include refid="Base\_Column\_List" />

from student

where 1=1

<choose>

<when test="studentId != null">

and student\_id=#{studentId}

when>

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

and name=#{name}

when>

<otherwise>

and 1=2

otherwise>

choose>

select>

3.3 测试

@Test

public void selectByIdOrName() {

SqlSession sqlSession = null;

sqlSession = sqlSessionFactory.openSession();

StudentMapper studentMapper = sqlSession.getMapper(StudentMapper.class);

Student student = new Student();

student.setName("小飞机");

student.setStudentId(1);

Student studentById = studentMapper.selectByIdOrName(student);

System.out.println("有 ID 则根据 ID 获取");

System.out.println(ToStringBuilder.reflectionToString(studentById, ToStringStyle.MULTI\_LINE\_STYLE));

student.setStudentId(null);

Student studentByName = studentMapper.selectByIdOrName(student);

System.out.println("没有 ID 则根据 name 获取");

System.out.println(ToStringBuilder.reflectionToString(studentByName, ToStringStyle.MULTI\_LINE\_STYLE));

student.setName(null);

Student studentNull = studentMapper.selectByIdOrName(student);

System.out.println("没有 ID 和 name, 返回 null");

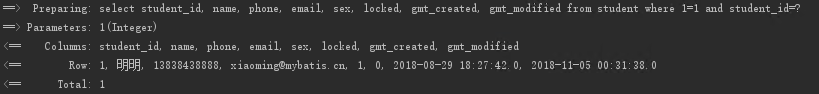
Assert.assertNull(studentNull);

sqlSession.commit();

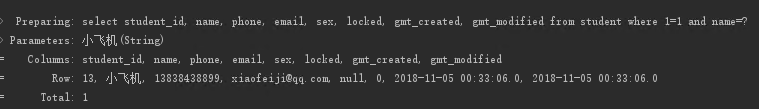
sqlSession.close();

}

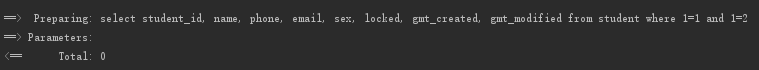
有 ID 则根据 ID 获取， 结果



没有 ID 则根据 name 获取



没有 ID 和 name, 返回 null



**4 trim(set、where)**

这三个其实解决的是类似的问题。如我们在写前面的[在 WHERE 条件中使用 if 标签] SQL 的时候， where 1=1 这个条件我们是不希望存在的。

4.1 where

4.1.1 查询条件

根据输入的学生信息进行条件检索。

1. 当只输入用户名时， 使用用户名进行模糊检索；
2. 当只输入性别时， 使用性别进行完全匹配
3. 当用户名和性别都存在时， 用这两个条件进行查询匹配查询

不使用 where 1=1。

4.1.2 动态 SQL

很显然， 我们要解决这几个问题

* 当条件都不满足时：此时 SQL 中应该要不能有 where ， 否则导致出错
* 当 if 有条件满足时：SQL 中需要有 where， 且第一个成立的 if 标签下的 and | or 等要去掉

这时候， 我们可以使用 where 标签。

接口方法

*/\*\**

*\* 根据输入的学生信息进行条件检索*

*\* 1. 当只输入用户名时， 使用用户名进行模糊检索；*

*\* 2. 当只输入邮箱时， 使用性别进行完全匹配*

*\* 3. 当用户名和性别都存在时， 用这两个条件进行查询匹配的用*

*\*/*

List selectByStudentSelectiveWhereTag(Student student);

对应的 SQL

<select id="selectByStudentSelectiveWhereTag" resultMap="BaseResultMap" parameterType="com.homejim.mybatis.entity.Student">

select

"Base\_Column\_List" />

from student

<where>

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

and name like concat('%', #{name}, '%')

if>

<if test="sex != null">

and sex=#{sex}

if>

where>

select>

4.1.3 测试

@Test

public void selectByStudentWhereTag() {

SqlSession sqlSession = null;

sqlSession = sqlSessionFactory.openSession();

StudentMapper studentMapper = sqlSession.getMapper(StudentMapper.class);

Student search = new Student();

search.setName("明");

System.out.println("只有名字时的查询");

List studentsByName = studentMapper.selectByStudentSelectiveWhereTag(search);

for (int i = 0; i < studentsByName.size(); i++) {

System.out.println(ToStringBuilder.reflectionToString(studentsByName.get(i), ToStringStyle.MULTI\_LINE\_STYLE));

}

search.setSex((byte) 1);

System.out.println("姓名和性别同时存在的查询");

List studentsBySex = studentMapper.selectByStudentSelectiveWhereTag(search);

for (int i = 0; i < studentsBySex.size(); i++) {

System.out.println(ToStringBuilder.reflectionToString(studentsBySex.get(i), ToStringStyle.MULTI\_LINE\_STYLE));

}

System.out.println("姓名和性别都不存在时查询");

search.setName(null);

search.setSex(null);

List studentsByNameAndSex = studentMapper.selectByStudentSelectiveWhereTag(search);

for (int i = 0; i < studentsByNameAndSex.size(); i++) {

System.out.println(ToStringBuilder.reflectionToString(studentsByNameAndSex.get(i), ToStringStyle.MULTI\_LINE\_STYLE));

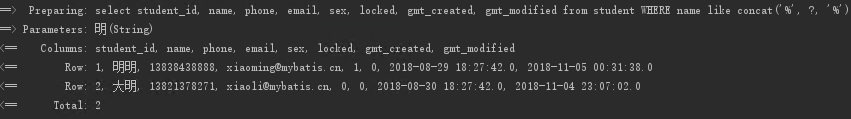
}

sqlSession.commit();

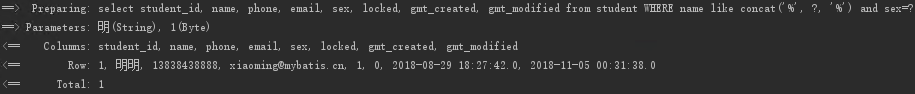
sqlSession.close();

}

只有名字时的查询, 有 where



姓名和性别同时存在的查询， 有 where



姓名和性别都不存在时查询, 此时， where 不会再出现了。



4.2 set

set 标签也类似， 在 [2.2 在 UPDATE 更新列中使用 if 标签] 中， 如果我们的方法 updateByPrimaryKeySelective 没有使用 标签， 那么我们就要想办法处理字段全为空的条件， 字段不为空的条件等。有了这个， 我们只需要写 if 标签即可， 不需要处理类似的问题。

4.3 trim

set 和 where 其实都是 trim 标签的一种类型， 该两种功能都可以使用 trim 标签进行实现。

4.3.1 trim 来表示 where

如以上的 where 标签， 我们也可以写成

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

trim>

表示当 trim 中含有内容时， 添加 where， 且第一个为 and 或 or 时， 会将其去掉。而如果没有内容， 则不添加 where。

4.3.2 trim 来表示 set

相应的， set 标签可以如下表示

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

trim>

表示当 trim 中含有内容时， 添加 set， 且最后的内容为 , 时， 会将其去掉。而没有内容， 不添加 set

4.3.3 trim 的几个属性

* prefix: 当 trim 元素包含有内容时， 增加 prefix 所指定的前缀
* prefixOverrides: 当 trim 元素包含有内容时， 去除 prefixOverrides 指定的 前缀
* suffix: 当 trim 元素包含有内容时， 增加 suffix 所指定的后缀
* suffixOverrides：当 trim 元素包含有内容时， 去除 suffixOverrides 指定的后缀

**5 foreach 标签**

foreach 标签可以对数组， Map 或实现 Iterable 接口。

foreach 中有以下几个属性

* collection: 必填， 集合/数组/Map的名称.
* item: 变量名。即从迭代的对象中取出的每一个值
* index: 索引的属性名。当迭代的对象为 Map 时， 该值为 Map 中的 Key.
* open: 循环开头的字符串
* close: 循环结束的字符串
* separator: 每次循环的分隔符

其他的比较好理解， collection 中的值应该怎么设定呢？

跟接口方法中的参数相关。

1. 只有一个数组参数或集合参数

默认情况：集合collection=list， 数组是collection=array

推荐：使用 @Param 来指定参数的名称， 如我们在参数前@Param("ids")， 则就填写 collection=ids

2. 多参数

多参数请使用 @Param 来指定， 否则SQL中会很不方便

3. 参数是Map

指定为 Map 中的对应的 Key 即可。其实上面的 @Param 最后也是转化为 Map 的。

4. 参数是对象

使用属性.属性即可。

5.1 在 where 中使用 foreach

在 where条件中使用， 如按id集合查询， 按id集合删除等。

5.1.1 查询条件

我们希望查询用户 id 集合中的所有用户信息。

5.1.2 动态 SQL

函数接口

*/\*\**

*\* 获取 id 集合中的用户信息*

*\* @param ids*

*\* @return*

*\*/*

List selectByStudentIdList(List ids);

对应 SQL

<select id="selectByStudentIdList" resultMap="BaseResultMap">

select

"Base\_Column\_List" />

from student

where student\_id in

<foreach collection="list" item="id" open="(" close=")" separator="," index="i">

#{id}

foreach>

select>

5.1.3 测试

@Test

public void selectByStudentIdList() {

SqlSession sqlSession = null;

sqlSession = sqlSessionFactory.openSession();

StudentMapper studentMapper = sqlSession.getMapper(StudentMapper.class);

List ids = new LinkedList<>();

ids.add(1);

ids.add(3);

List students = studentMapper.selectByStudentIdList(ids);

for (int i = 0; i < students.size(); i++) {

System.out.println(ToStringBuilder.reflectionToString(students.get(i), ToStringStyle.MULTI\_LINE\_STYLE));

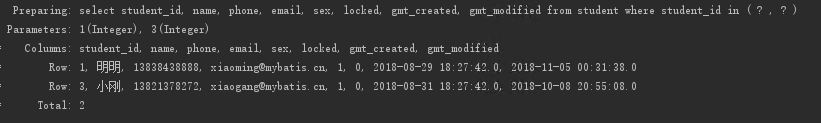
}

sqlSession.commit();

sqlSession.close();

}

结果



5.2 foreach 实现批量插入

可以通过foreach来实现批量插入。

5.2.1 动态SQL

接口方法

*/\*\**

*\* 批量插入学生*

*\*/*

int insertList(List students);

对应的SQL

"insertList">

insert into student(name, phone, email, sex, locked)

values

<foreach collection="list" item="student" separator=",">

(

#{student.name}, #{student.phone},#{student.email},

#{student.sex},#{student.locked}

)

foreach>

5.2.2 测试

@Test

public void insertList() {

SqlSession sqlSession = null;

sqlSession = sqlSessionFactory.openSession();

StudentMapper studentMapper = sqlSession.getMapper(StudentMapper.class);

List students = new LinkedList<>();

Student stu1 = new Student();

stu1.setName("批量01");

stu1.setPhone("13888888881");

stu1.setLocked((byte) 0);

stu1.setEmail("13888888881@138.com");

stu1.setSex((byte) 1);

students.add(stu1);

Student stu2 = new Student();

stu2.setName("批量02");

stu2.setPhone("13888888882");

stu2.setLocked((byte) 0);

stu2.setEmail("13888888882@138.com");

stu2.setSex((byte) 0);

students.add(stu2);

System.out.println(studentMapper.insertList(students));

sqlSession.commit();

sqlSession.close();

}

结果



**6 bind 标签**

bind 标签是通过 OGNL 表达式去定义一个上下文的变量， 这样方便我们使用。

如在 selectByStudentSelective 方法中， 有如下

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

and name like concat('%', *#{name}, '%')*

if>

在 MySQL 中， 该函数支持多参数， 但在 Oracle 中只支持两个参数。那么我们可以使用 bind 来让该 SQL 达到支持两个数据库的作用

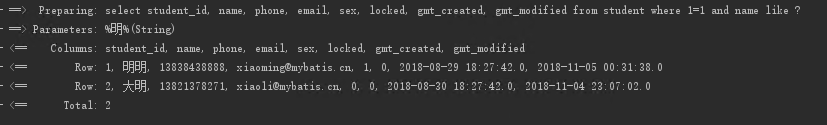
<if test="name != null and name !=''">

<bind name="nameLike" value="'%'+name+'%'"/>

and name like *#{nameLike}*

if>

更改后的查询结果如下



作者: 阿进的写字台

来源：https://www.cnblogs.com/homejim/p/9909657.html

 往期推荐

🔗

* [我的天！将10w行级别的数据导入到Excel，踩了不少坑...](http://mp.weixin.qq.com/s?__biz=MzIzMzgxOTQ5NA==&mid=2247498510&idx=2&sn=34bbe2bde284d97b4a88235ff53432a6&chksm=e8fd6707df8aee111096588b6c19c449e894dd086d332d2633a8bb428de80a6000f683135746&scene=21#wechat_redirect)
* [牛逼！一个逻辑漏洞，黑客潜伏6个月，搞走4000万...](http://mp.weixin.qq.com/s?__biz=MzIzMzgxOTQ5NA==&mid=2247498510&idx=3&sn=6749ef54fa335ae3e9b732f45d494243&chksm=e8fd6707df8aee11fc8c2e06a7d19ce8f0fc0a8db4d36d177f095ea6717f64f149f2448b023a&scene=21#wechat_redirect)
* [什么是程序员？](http://mp.weixin.qq.com/s?__biz=MzIzMzgxOTQ5NA==&mid=2247498510&idx=4&sn=b5359229b536ebf51623f9b2edd2451a&chksm=e8fd6707df8aee1191b8c8480c6049079e49ea53c778240f045bb86ae7786a56ebd22162f1e5&scene=21#wechat_redirect)





点击阅读原文，获得**免费编程资料**

### 精选留言



Crazy、赞：5

where 1=1 会影响性能 会影响索引等查询优化策略



醉酒的戈多赞：3

where后面加上1=1不查询索引了，数据量大的时候贼慢



淼ERRoR赞：2

看到xml就烦😰



小猴纸mini赞：2

这个1＝1也太low了吧，〈where〉标签走起啊。



hello world赞：1

一群说1=1会影响索引性能的，脑子有毛病吧，甚至都怀疑你们有没有用explain去看执行计划，人云亦云，一群云玩家。 where 1=1 在优化器代数优化阶段就会被优化掉，根本就没影响 ，自己多写几条SQL 多研究一下，不要张口就来



赖仕雄赞：0

之前用hibernate，我觉得qbc用起来才爽

[阅读全文](https://mp.weixin.qq.com/s/Nxw3jz11kle_YJmEItvJiA#rd)