链滴

myBatis【常规使用】

作者：haxLook

原文链接：https://ld246.com/article/1617158199256

来源网站：链滴

myBatis

MyBatis 是一款优秀的持久层框架，它支持自定义 SQL、存储过程以及高级映射。MyBatis 免除了几所有的 JDBC 代码以及设置参数和获取结果集的工作。MyBatis 可以通过简单的 XML 或注解来配置映射原始类型、接口和 Java POJO（Plain Old Java Objects，普通老式 Java 对象）为数据库中的录。

mybatis和spring的整合，以及关于的mybatis配置数据源就不在这个给介绍了，本文主要是介绍mybtis在工作日常中一些使用。

mybatis的接口写法

- 使用注解，在接口的方法上面添加@Select@Update等注解，里面写上对应的SQL语句进行SQL语的绑定。
- 通过映射文件xml方式进行绑定，指定xml映射文件中的namespace对应的接口的全路径名

但是现在一般的项目都是整合了mybatis_plus，所以使用使用第一种方式很少，因为如果需要写的sq过于复杂，一般都是使用mapper.xml进行sql写入。

动态SQL

if

日常sql都要使使用where语句的判断，在使用if的过程中会需要注意的是对于where语句写法

第一种写法

```
<select id="findmybatis" resultType="user">
  SELECT * FROM user
 <where>
  <if test="user_name!= null">
    AND user_name = #{user_name}
  </if>
  <if test="user != null and user.userId!= null">
   AND user_id = #{user.userId}
  </if>
 </where>
</select>
```

第二种写法

```
<select id="findmybatis" resultType="user">
  SELECT * FROM user
<trim prefix="WHERE" prefixOverrides="AND |OR ">
  <if test="user_name!= null">
    AND user_name = #{user_name}
  </if>
  <if test="user != null and user.userId!= null">
   AND user_id = #{user.userId}
  </if>
</trim>
</select>
```

set,trim,where

第一种使用where语句，若子句的开头为 “AND” 或 “OR”，where 元素也会将它们去除。

第二种使用属于自定义类型:

- prefix=添加前缀
- suffix=添加后缀
- prefixOverrides=去掉前缀
- suffixOverrides=去掉后缀

suffixOverrides可以用于在使用动态的更新数据，与他功能类似的是set语法,set会去除最后不为空值，’

suffixOverrides去除最后一个 ‘，’ ，也可以用于在其他的操作上。

第一种写法

```
<update id="updateuser">
 update user
   <set>
     <if test="user_name != null">user_name=#{username},</if>
     <if test="pass_word != null">pass_word=#{password},</if>
     <if test="user_email != null">user_email=#{email}</if>
   </set>
 where user_id=#{id}
</update>
```

第二种写法

```
<update id="updateuser">
 update user
   <trim prefix="set" suffixoverride="," suffix="where user_id=#{id}">
     <if test="user_name != null">user_name=#{username},</if>
     <if test="pass_word != null">pass_word=#{password},</if>
     <if test="user_email != null">user_email=#{email}</if>
   </trim>

</update>
```

foreach

使用sql中的in语句，使用list集合遍历查询，在使用过程中如果传递的参数是一个map，就直接将ma进行参数的传递进来，栗子：第二种方式写法

```
List<Integer> ids = new ArrayList<Integer>();
       ids.add(1);
       ids.add(2);
       ids.add(3);
       ids.add(6);
```

```
        ids.add(7);
        ids.add(9);
map.put('userids',ids)
//==========
IPage<user> queryByCondition(Page page, @Param("parameter") Parameter parameter,@Pa
am('map') Map><String,Object> map);

第一种
<select id="finduser" resultType="user">
  SELECT *
  FROM user
  WHERE id in
  <foreach item="item" index="index" collection="list"
     open="(" separator="," close=")">
       #{item}
  </foreach>
</select>

第二种
<select id="finduser" resultType="user">
  SELECT *
  FROM user
  WHERE id in
  <foreach item="userids" index="index" collection="list"
     open="(" separator="," close=")">
       #{item}
  </foreach>
</select>
```

choose, when, otherwise

类似于java代码中的switch语句

下面的语句使用中需要注意2个点，在参数传递中，如果涉及到在test中进行字符串的比较的时候，需注意将“”改成‘’ 里面使用“”来标识字符串，否则是失效的，使用大于，小于号的时候，使用<![DATA[标识]]>格式处理，防止出现编译错误！

```
<select id="finduser"
    resultType="user">

  <bind name="pattern" value="_parameter.getUsersex()" />

  SELECT * FROM user WHERE state =   'ACTIVE'
  <choose>
    <when test='pattern == "1"'>
      AND user_age <![CDATA[>]]> #{user_age}
    </when>
    <when test="'pattern == "2"'>
      AND user_age <![CDATA[<]]> #{user_age}
    </when>
    <otherwise>
      AND user_age = #{user_age}
    </otherwise>
  </choose>
```

```
</select>
```

bind

bind 元素可以从 OGNL 表达式中创建一个变量并将其绑定到上下文。比如：

```
<select id="selectBlogsLike" resultType="Blog">
  <bind name="pattern" value="'%' + _parameter.getTitle() + '%'" />
  SELECT * FROM BLOG
  WHERE title LIKE #{pattern}
</select>
```

MyBatis 中#{}和${}区别

#{} 是预编译处理，像传进来的数据会加个" "（#将传入的数据都当成一个字符串，会对自动传入的据加一个双引号）

${}是指就是字符串替换。直接替换掉占位符,一般使用的话，之传入表名。

使用${}会导致sql注入风险，举一个栗子:

select * from user where id = ${value}

value 应该是一个数值吧。然后如果对方传过来的是 '123' and user_name = 'hax'。直接会多个条改动了原有的sql语句。如果是攻击性的语句呢？ ‘123；drop table user，直接威胁了数据库数据全，所有尽量建议不使用

模糊查询的写法

记得上一次写代码的过程，我写了这样的一段代码and username LIKE '%${username}%'，有一个事过和说，你这个代码写的生效吗？虽然说我这个写法不安全，但是至少是正确的。

我但是就直接怼回去了，我说你可能只会写 AND name LIKE CONCAT(CONCAT('%',#{name},'%'))
我觉得在工作中，质疑他人前一定要有正确的结论，都是搞技术的谁都不愿意被质疑。

5种写法，个人推荐使用第三种写法，我一般使用，至于上次为什么使用第一种，em.....，主要是想体一下新鲜感joy

方式1：$ 这种方式，简单，但是无法防止SQL注入，所以不推荐使用

```
LIKE '%${username}%'
```

方式2：

```
LIKE "%"#{username}"%"方式3：字符串拼接
```

```
AND username LIKE CONCAT(CONCAT('%',#{username},'%'))
```

方式4：bind标签

```
<select id="finduser" resultType="com.hax.entity.user"
  parameterType="com.hax.entity.user">
  <bind name="pattern1" value="'%' + _parameter.name + '%'" />
```

```
 <bind name="pattern2" value="'%' + _parameter.address + '%'" />
 SELECT * FROM user
 <where>
  <if test="name != null and name != ''">
   AND user_name LIKE #{pattern1}
  </if>
  <if test="address != null and address != ''">
   AND user_address LIKE #{pattern2}
  </if>
 </where>
</select>
```

方式5：java代码里写

param.setUsername("%username%");

在 java 代码中传参的时候直接写上

```
<if test="username!=null"> AND username LIKE #{username}</if>
```

mybatis如何获取自增的主建id

insert 方法总是返回一个 int 值 ， 这个值代表的是插入的行数。如果采用自增长策略，自动生成的键在 insert 方法执行完后可以被设置到对象对应的属性中

示例：mysql中加入 usegeneratedkeys="true" keyproperty="id"

```
<insert id="insertname" usegeneratedkeys="true" keyproperty="
id">
insert into names (name) values (#{name})
</insert>

java代码
user user = new user();
user.setname("username");
int rows = mapper.insertname(user);
system.out.println("user-id"+ rows.getId());
```

orcal在这里不介绍，自行百度

MyBatis 传递多个参数

- 方法一:使用map接口传递参数

```
public List<user> finduserBymap(Map<String, Object> parameterMap);

<select id="findRolesByMap" parameterType="map" resultType="role">
  select *from user where user_id =#{id}
</select>
```

在mapper文件中直接使用map在put的值 栗子：map.put('id',id); 使用就是#{id}

- 方法二:使用注解传递多个参数

MyBatis为开发者提供了一个注解@Param（org.apache.ibatis.annotations.Param），可以通过它定义映射器的参数名称，使用它可以得到更好的可读性

```
public List<user> finduserByAnnotation(@Param("username") String username, @Param("sex
) String sex);

<select id="finduserByAnnotation" resultType="role">
 select *from user where user_name like CONCAT('%'+#{username}+'%') and user_sex = #{sex

</select
```

- 方法三:通过Java Bean传递多个参数,其实就是自己定义个包装类进行封装即可。

总体的来说，方法一，使用起来简洁，但是可读性不强，你不能很直观看到自己传入的参数类型，方二的就可读性最强，但是当需要传入参数过多的时候，在写代码的时候，也是不是很直观，会显得繁建议参数小于5个，方法三，一般使用包装类的话，只能说明参数传入比较特殊，必然复用性不好。

参数多推荐 一或者三

参数少推荐 二

mybatis的缓存机制

官方网站写的太好了。我怕我的语言组织误导大家，所以直接copy过来了
ob 地址

缓存机制减轻数据库压力，提高数据库性能

mybatis的缓存分为两级：一级缓存、二级缓存

- 一级缓存:

一级缓存为 sqlsesson 缓存，缓存的数据只在 SqlSession 内有效。在操作数据库的时候需要先创建 SlSession 会话对象，在对象中有一个 HashMap 用于存储缓存数据，此 HashMap 是当前会话对象有的，别的 SqlSession 会话对象无法访问。

具体流程:

第一次执行 select 完毕会将查到的数据写入 SqlSession 内的 HashMap 中缓存起来

第二次执行 select 会从缓存中查数据，如果 select 同传参数一样，那么就能从缓存中返回数据，不去数据库了，从而提高了效率

注意:

1. 如果 SqlSession 执行了 DML 操作（insert、update、delete），并 commit 了，那么 mybatis 会清空当前 SqlSession 缓存中的所有缓存数据，这样可以保证缓存中的存的数据永远和数据库中一，避免出现差异
2. 当一个 SqlSession 结束后那么他里面的一级缓存也就不存在了， mybatis 默认是开启一级缓存，

需要配置

3. mybatis 的缓存是基于 [namespace:sql语句:参数] 来进行缓存的，意思就是， SqlSession 的 HashMap 存储缓存数据时，是使用 [namespace:sql:参数] 作为 key ，查询返回的语句作为 value 保存的

- 二级缓存:

二级缓存是 mapper 级别的缓存，也就是同一个 namespace 的 mapper.xml ，多个 SqlSession 使用同一个 Mapper 操作数据库的时候，得到的数据会缓存在同一个二级缓存区域

二级缓存默认是没有开启的。需要在 setting 全局参数中配置开启二级缓存

开启二级缓存步骤:

1. conf.xml 配置全局变量开启二级缓存

```
<settings>
   <setting name="cacheEnabled" value="true"/>默认是false：关闭二级缓存
<settings>
```

2. 在 userMapper.xml 中配置

```
<cache eviction="LRU" flushInterval="60000" size="512" readOnly="true"/>当前mapper下所语句开启二级缓存
```

基本上就是这样。这个简单语句的效果如下:

- 映射语句文件中的所有 select 语句的结果将会被缓存。
- 映射语句文件中的所有 insert、update 和 delete 语句会刷新缓存。
- 缓存会使用最近最少使用算法（LRU, Least Recently Used）算法来清除不需要的缓存。
- 缓存不会定时进行刷新（也就是说，没有刷新间隔）。
- 缓存会保存列表或对象（无论查询方法返回哪种）的 1024 个引用。
- 缓存会被视为读/写缓存，这意味着获取到的对象并不是共享的，可以安全地被调用者修改，而不扰其他调用者或线程所做的潜在修改。

这个更高级的配置创建了一个 FIFO 缓存，每隔 60 秒刷新，最多可以存储结果对象或列表的 512 个用，而且返回的对象被认为是只读的，因此对它们进行修改可能会在不同线程中的调用者产生冲突。用的清除策略有:

- LRU – 最近最少使用：移除最长时间不被使用的对象。
- FIFO – 先进先出：按对象进入缓存的顺序来移除它们。
- SOFT – 软引用：基于垃圾回收器状态和软引用规则移除对象。
- WEAK – 弱引用：更积极地基于垃圾收集器状态和弱引用规则移除对象。

默认的清除策略是 LRU。

flushInterval（刷新间隔）属性可以被设置为任意的正整数，设置的值应该是一个以毫秒为单位的合时间量。 默认情况是不设置，也就是没有刷新间隔，缓存仅仅会在调用语句时刷新。

size（引用数目）属性可以被设置为任意正整数，要注意欲缓存对象的大小和运行环境中可用的内存

源。默认值是 1024。

readOnly（只读）属性可以被设置为 true 或 false。只读的缓存会给所有调用者返回缓存对象的相实例。 因此这些对象不能被修改。这就提供了可观的性能提升。而可读写的缓存会（通过序列化）返缓存对象的拷贝。 速度上会慢一些，但是更安全，因此默认值是 false。

这里配置了一个 LRU 缓存，并每隔60秒刷新，最大存储512个对象，而返回的对象是只读的

若想禁用当前select语句的二级缓存，添加 useCache="false"修改如下：

<select id="getCountByName" parameterType="java.util.Map" resultType="INTEGER" statem
ntType="CALLABLE" useCache="false">

具体流程：

1.当一个 sqlseesion 执行了一次 select 后，关闭此 session 的时候，会将查询结果缓存到二级缓存

2.当另一个 sqlsession 执行 select 时，首先会他自己的一级缓存中找，如果没找到，就回去二级缓存中找，找到了就返回，就不用去数据库了，从减少了数据库压力提高了性能

注意:

1. 如果 SqlSession 执行了 DML 操作（insert、update、delete），并 commit 了，那么 mybatis 就会清空当前 mapper 缓存中的所有缓存数据，这样可以保证缓存中的存的数据永和数据库中一致，避免出现差异

2. mybatis 的缓存是基于 [namespace:sql语句:参数] </pan>来进行缓存的，意思就是，SqlSession 的 HashMap 存储缓存数据时，是使用 [namespace:sql参数] 作为 key ，查询返回的语句作为 value 保存的。

mybatis的缓存机制用的很少，包括于它的一级缓存，使用的人都很少，主要原因是因为对于缓存的制不够好很容易出现脏读的情况，使用一级缓存的时候，因为缓存不能跨会话共享，不同的会话之间于相同的数据可能有不一样的缓存。在有多个会话或者分布式环境下，会存在脏数据的问题。如果要决这个问题，就要用到二级缓存。MyBatis 一级缓存（MyBaits 称其为 Local Cache）无法关闭，但有两种级别可选：

- session 级别的缓存，在同一个 sqlSession 内，对同样的查询将不再查询数据库，直接从缓存中。
- statement 级别的缓存，避坑： 为了避免这个问题，可以将一级缓存的级别设为 statement 级别，这样每次查询结束都会清掉一级缓存。

注意这里的session是sqlsession会话，不是用户的的session会话，MyBatis一级缓存的生命周期和SqSession一致，MyBatis的一级缓存最大范围是SqlSession内部，有多个SqlSession或者分布式的环下，数据库写操作会引起脏数据，建议设定缓存级别为Statement，分布式的开发环境中，直接使用Rdis、Memcached等分布式缓存可能成本更低，安全性也更高。