Mybatis

1. 防止sql注入 #与$的区别

#的用法

使用#{}格式的语法，在mybatis中会使用Preparement语句来安全地设置值，跟踪断点会看到即将执行的sql中用“?”做占位符，执行sql类似于

PreparedStatement ps = conn.prepareStatement(sql);

ps.setInt(1,id);

#{}写法会将传入的数据都当成一个字符串，会对传入的数据加一个双引号，因此可以在很大程度上防止sql注入，

如：select \* from user where user\_id =  #{userId}，如果传入的值是111,那么解析成的sql时为... where user\_id =  "111";

$的用法

${}时，mybatis不会将传入的数据当成一个字符串，执行sql类似于

Statement st = conn.createStatement();

ResultSet rs = st.executeQuery(sql);

 这种写法会将传入的数据会直接显示在执行的sql中。

如：select \* from user where user\_id =  ${userId}，如果传入的值是100,那么解析成的sql为... where user\_id =  100;

可能会发生sql注入，例如传入的参数是单引号'，将会执行... where user\_id =’ 这样的语句，会报sql语法错误。

2、传入一个不改变的字符串或者传入数据库字段(列名)，例如要传入order by 后边的参数

这种情况下必须使用${}。

#{}方式一般用于传入字段值，并将该值作为字符串加到执行sql中，一定程度防止sql注入；

${}方式一般用于传入数据库对象，例如传入表名，不能防止sql注入，存在风险。

模糊查询中，如果使用如下方式：select \* from reason\_detail where reason\_en like '%${reason}%'，此处只能使用$,使用#的话，反而会被解析为列，报错java.sql.SQLException: Column 'reason' not found

Mysql： 使用like % 查询 LIKE CONCAT('%',#{name},'%' )  
Oracle：使用like % 查询 LIKE '%'||#{name}||'%' 或者CONCAT函数

#与$的总结：

#{}：以预编译的形式，将参数设置到sql语句中，用？表示参数,去参数值用这种形式

${}:

String sql = “select \* from tb\_name where name= ‘”+varname+”’ and passwd=’”+varpasswd+”’”;

把[‘;drop table tb\_name;]作为varpasswd传入进来,则:

select \* from tb\_name = ‘随意’ and passwd = ”;drop table tb\_name;有些数据库是不会让你成功的,但也有很多数据库就可以使这些语句得到执行. --🡪会对数据库进行删除，因此#{}采用预编译的方法更加安全，而且预编码数据库有缓存sql的功能，减少数据库压力

预编译会对传过来的值进行处理，保证sql非法注入

1. 语法

<trim></trim>标签的用法：

prefix:在trim标签内sql语句加上前缀。

suffix:在trim标签内sql语句加上后缀。

suffixOverrides:指定去除多余的后缀内容，如：suffixOverrides=","，去除trim标签内sql语句多余的后缀","prefixOverrides:指定去除多余的前缀内容

比如

<trim prefix="(" suffix=")" prefixOverrides="and" >  
 <foreach collection="criteria.criteria" item="criterion" >  
 <choose >  
 <when test="criterion.noValue" >  
 and ${criterion.condition}  
 </when>  
 <when test="criterion.singleValue" >  
 and ${criterion.condition} #{criterion.value}  
 </when>

在查询字段前后加上“（”后面加上“）”,为了防止字段为空需要把 and ${criterion.condition}

前面的and去除掉，需要用prefixOverrides

<**choose></choose>,<when></when>,<otherwise></otherwise>标签的用法**

**choose**元素的作用就相当于JAVA中的switch语句，基本上跟JSTL中的choose的作用和用法是一样的，通常都是与when和otherwise搭配的。看如下一个例子：

***<select id="dynamicChooseTest" parameterType="Blog" resultType="Blog">***

***select \* from t\_blog where 11 = 1***

***<choose>***

***<when test="title != null">***

***and title = #{title}***

***</when>***

***<when test="content != null">***

***and content = #{content}***

***</when>***

***<otherwise>***

***and owner = "owner1"***

***</otherwise>***

***</choose>***

***</select>***

when元素表示当when中的条件满足的时候就输出其中的内容，跟JAVA中的switch效果差不多的是按照条件的顺序，当when中有条件满足的时候，就会跳出choose，即所有的when和otherwise条件中，只有一个会输出，当所有的我很条件都不满足的时候就输出otherwise中的内容。所以上述语句的意思非常简单， 当title!=null的时候就输出and titlte = #{title}，不再往下判断条件，当title为空且content!=null的时候就输出and content = #{content}，当所有条件都不满足的时候就输出otherwise中的内容。

主键的处理：

自增处理

数据库支持自增：MySQL

1. useGenetatedkeys=”true” 设置为true keyProperty=“id” 指定pojo中主键的属性

<insert useGenetatedkeys=”true” keyProperty=” id“

Oracle不支持自增：采用序列模拟自增

<selectKey keyProperty="id" resultType="Long" order="BEFORE">  
 select  
 SEQ\_CARD\_TYPE.nextval from dual  
</selectKey>

查看用户序列： select\* from user\_sequences 查看所有序列 利用selectKey来获取自增主键，与insert中的id相对应（红色字体相对应） order=”BEFORE” 查询自增主键的sql在insert语句执行之前执行 resultTyoe=“Integer”表示查询出来的结果

<insert>

<selectKey keyProperties=’id’ order=”BEFORE” resultTyoe=“Integer”> select Employed\_SEQ.nextval from dual </>

Insert into employ (id,name) values (#{id},#{name})

</insert>

传入多个参数处理：

Mybatis遇到多个参数会做特殊处理封装为一个map

1. 返回值类型可以为resultType

在接口中利用@param（"id”） Integer id, @param（“name”） string name 来指定参数的名字

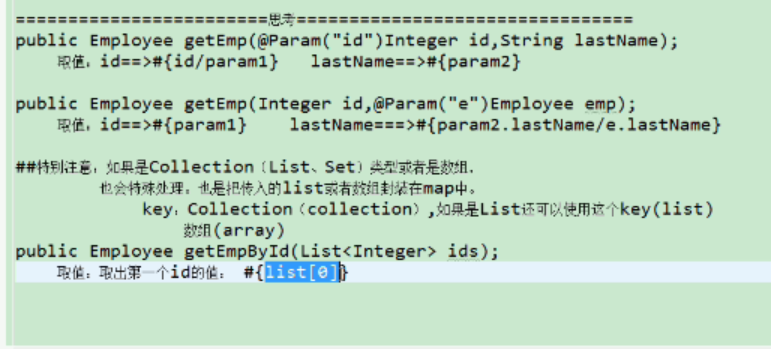
在xml中可以明确传入参数的名字 <insert> #{id} #{name} </>

1. 利用map集合指定 Map.put(“id”,1) Map.put(“name”,zhang)

参数中放map集合 （map<String,Object> map）

<insert> #{id} #{name} </>

参数取值处理



源码部分：

Mybatis会为mapper接口创建一个代理对象

org.apache.ibatis.binding.MapperProxy----🡪invoke方法

根据传入的参数开始执行

return mapperMethod.execute(sqlSession, args);

根据判断选择不同增删改查的方法进行执行

Object result;  
switch (command.getType()) {  
 case *INSERT*: {  
 Object param = method.convertArgsToSqlCommandParam(args);  
 result = rowCountResult(sqlSession.insert(command.getName(), param));  
 break;  
 }  
 case *UPDATE*: {  
 Object param = method.convertArgsToSqlCommandParam(args);  
 result = rowCountResult(sqlSession.update(command.getName(), param));  
 break;  
 }  
 case *DELETE*: {  
 Object param = method.convertArgsToSqlCommandParam(args);  
 result = rowCountResult(sqlSession.delete(command.getName(), param));  
 break;  
 }  
 case *SELECT*:

将传入的参数转换为sql能用的参数：

Object param = method.convertArgsToSqlCommandParam(args);

继续进入

public Object convertArgsToSqlCommandParam(Object[] args) {  
 return paramNameResolver.getNamedParams(args);  
}

继续进入--🡪可以看到对参数进行处理的代码

public Object getNamedParams(Object[] args) {  
 final int paramCount = names.size();  
 if (args == null || paramCount == 0) {  
 return null;  
 } else if (!hasParamAnnotation && paramCount == 1) {  
 return args[names.firstKey()];  
 } else {  
 final Map<String, Object> param = new ParamMap<Object>();  
 int i = 0;  
 for (Map.Entry<Integer, String> entry : names.entrySet()) {  
 param.put(entry.getValue(), args[entry.getKey()]);  
 // add generic param names (param1, param2, ...)  
 final String genericParamName = *GENERIC\_NAME\_PREFIX* + String.*valueOf*(i + 1);  
 // ensure not to overwrite parameter named with @Param  
 if (!names.containsValue(genericParamName)) {  
 param.put(genericParamName, args[entry.getKey()]);  
 }  
 i++;  
 }  
 return param;  
 }  
}

final int paramCount = names.size();--🡪name根据下面的方法来得到

org.apache.ibatis.reflection.ParamNameResolver#ParamNameResolver

这个方法里面，如果有param注解，就获取注解的值，等获获取参数值的相应的操作

if (annotation instanceof Param) {  
 hasParamAnnotation = true;  
 name = ((Param) annotation).value();  
 break;  
}

单条记录

如果是接口中返回值是list<Emp>集合 xml中的resultType=“emp”这个对象的全类名”

如果接口中返回值是Map<String,Object> xml中resultType=”map”

多条记录

ResultMap的使用：

方式一

如果employee这个实体类中有emp实体类

<resultMap type=”全类名” id=”emp”>

<id column property

<result …

<result column=”did” property=”dept.did”

<resultMap>

<select id=” ” resultMap=”emp”>

查询语句

</>

方式二：注意<association 中 column的名字不能与上面一样，否则会被覆盖



延迟加载：LazyLoadingEnable 设置为true开始，默认为false

有二条sql是一起执行的，分别查询员工和部门信息，但是现在只需要员工信息，开启延迟加载就只会执行一条sql，需要部门信息的时候才会执行另一条sql

缓存（一级缓存类似局部，二级缓存是全局）

一级缓存（本地缓存，一直开启无法关闭，为sqlsession级别，不同的sqlsession对象有不同的一级缓存）

失效的四种情况：

1. Sqlsesion不同

SqlSession openSession = sqlSessionFactory.openSession();  
SqlSession openSession2 = sqlSessionFactory.openSession();

2.查询条件不同

3.增删改操作

4.清除缓存

openSession.clearCache();

二级缓存

二级缓存是基于namespace（xml文件）的，不同的namespace查询的数据放在对应的namespace下面

1. 开启全局二级缓存配置（在对应的xml中）

<cache eviction=” 回收策略” flushInterval=” 缓存刷新间隔” readOnly size ></>

eviction:缓存的回收策略：

• LRU – 最近最少使用的：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：移除基于垃圾回收器状态和软引用规则的对象。

• WEAK – 弱引用：更积极地移除基于垃圾收集器状态和弱引用规则的对象。

• 默认的是 LRU。

flushInterval：缓存刷新间隔

缓存多长时间清空一次，默认不清空，设置一个毫秒值

readOnly:是否只读：

true：只读；mybatis认为所有从缓存中获取数据的操作都是只读操作，不会修改数据。

mybatis为了加快获取速度，直接就会将数据在缓存中的引用交给用户。不安全，速度快

false：非只读：mybatis觉得获取的数据可能会被修改。

mybatis会利用序列化&反序列的技术克隆一份新的数据给你。安全，速度慢

size：缓存存放多少元素；

2.引用其他xml的缓存

<cache-ref namespace="com.ibm.mybatis.dao.EmployeeMapper"/>

3. 自定义缓存

3-1. 在xml开启

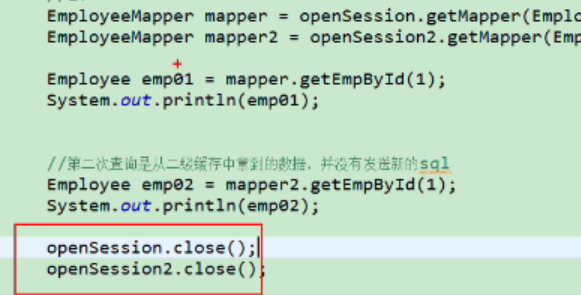
<cache type="org.mybatis.caches.ehcache.EhcacheCache"></cache>

3.2 java文件实现cache接口

public final class EhcacheCache implements Cache

查询的数据会默认放在一级缓存只有当一级缓存关掉才会放置进入二级缓存，否则会进行二次查询

发送二次sql



发送一次sql



缓存有关的配置：

1）、cacheEnabled=true：false：关闭缓存（二级缓存关闭）(一级缓存一直可用的)

2）、每个select标签都有useCache="true"：false：不使用缓存（一级缓存依然使用，二级缓存不使用）

3）、【每个增删改标签的：flushCache="true"：（一级二级都会清除）】增删改执行完成后就会清楚缓存；

测试：flushCache="true"：一级缓存就清空了；二级也会被清除；查询标签：flushCache="false"：

如果flushCache=true;每次查询之后都会清空缓存；缓存是没有被使用的；

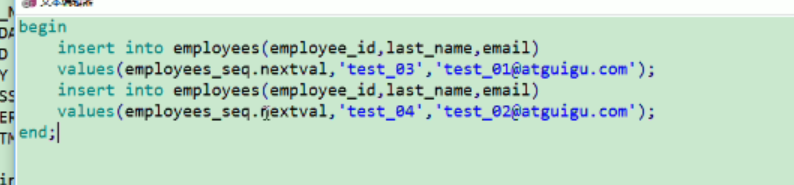
4）、sqlSession.clearCache();只是清楚当前session的一级缓存；

5）、localCacheScope：本地缓存作用域：（一级缓存SESSION）；当前会话的所有数据保存在会话缓存中；

STATEMENT：可以禁用一级缓存；

动态sql

Oracle插入多条数据：



<where>标签只会去掉第一个多出来的and或则or

<foreach>标签 open以什么开始，close以什么结束 open=”(” close=”)” 以”（”开始 以”）”结束

内置参数：

\_parameter:代表传递过来的整个参数（如果只有一个参数就代表哪一个参数）

\_databaseId:与databaseIdProvider对应 代表数据库的别名

Sql优化

1. 避免在where子句中对 字段进行null值判断，否则放弃索引，进行全表扫描

Select \* from table where num is null；

改善：可以在 num 上设置默认值 0,确保表中 num 列没有 null 值，然后这样查询：

select id from t where num=0;

select nvl(card\_no,'空') card\_no from member\_card\_info

可以将card\_no为null的值替换为’空’

1. like函数的替代instr

（7035 36.422秒）

*select \* from member\_info where org\_id like '%070%'*

（7035 37.962秒）

select \* from member\_info where instr(org\_id,'070')>0

instr(org\_id,'070')>0返回在字段org\_id中存在070的数据

1. where字句中使用！=或则<>或则or进行连接或则in与not in以及

select id from t where name like '%c%';都将放弃索引进行全表扫描

改善：对于select \* from table where num in （1,2,3）--->连续的值可以替换为

Select \* from table where num between 1 and 3;

1. 用exist替代in

select num from a where num in(select num from b);

用下面的语句替换：select num from a where exists(select 1 from b where num=a.num);

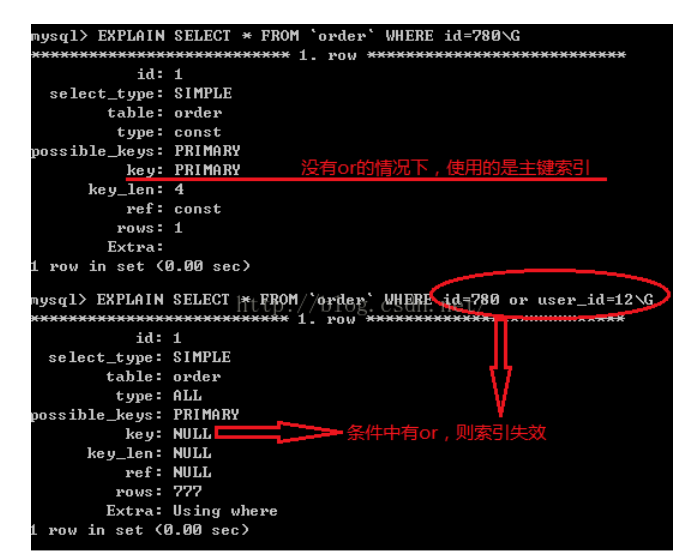
1. MySQL UNION 与UNION All的用法

Union:去除二次查询结果中相同的结果

UNION All：得到全部的结果

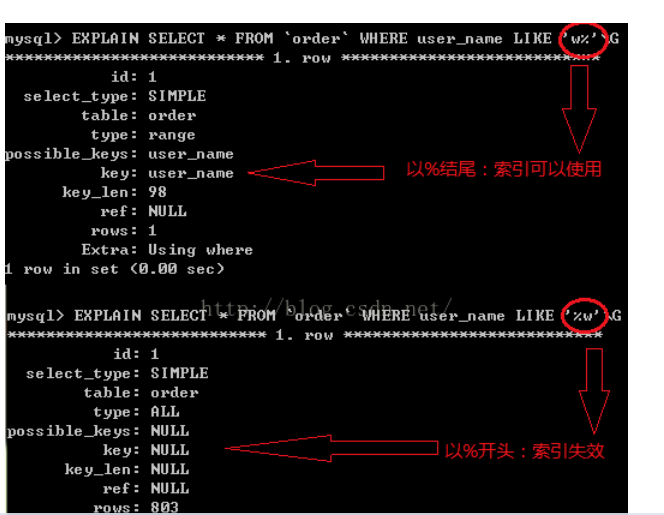
索引失效的情况：

1. 调教用or



要想使用or，又想让索引生效，只能将or条件中的每个[列都](https://www.baidu.com/s?wd=%E5%88%97%E9%83%BD&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)加上索引

1. like查询以%开头



1. 如果列类型是字符串，那一定要在条件中将数据使用引号引用起来,否则不使用索引

