一.动态sql

动态 SQL 元素和 JSTL 或基于类似 XML 的文本处理器相似,例如拼接时要确保不能忘记添加必要的空格，还要注意去掉列表最后一个列名的逗号。利用动态 SQL 这一特性可以彻底摆脱这种痛苦。

- if
- choose (when, otherwise)
- trim (where, set)
- foreach

foreach

接口:

```
//一次添加多个Girl
int insert1(List<Girl> girlList);

int deleteByIds(int[] ids);
```

实现:

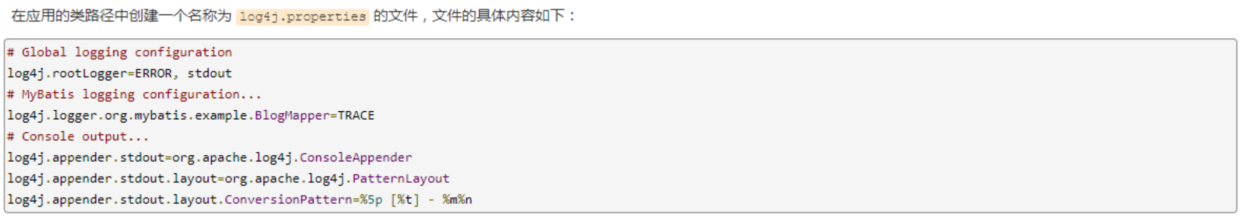
```
实现单个对象(girl)添加
<insert id="insert">
 insert into sms_girl(g_name, g_age) VALUES (#{gName},#{gAge})
 <selectKey keyProperty="gId" order="AFTER" resultType="long">
 select LAST_INSERT_ID();
 </selectKey>
</insert>
实现多个对象(girl)添加
<insert id="insert1">
 insert into sms_girl(g_name, g_age) values
 <foreach collection="list" item="girl" separator=",">
 (#{girl.gName},#{girl.gAge})
 </foreach>
</insert>
实现删除多个的操作
<delete id="deleteByIds">
 delete from sms_girl where g_id in
 <foreach collection="array" separator="," open="(" close=")">
 #{i}
 </foreach>
</delete>
```

测试:

```
@Test
void test2() {
 SqlSession sqlSession = sqlSessionFactory.openSession();
 Girl girl = new Girl();
 girl.setGName("刘亦菲");
 girl.setGAge(36);
 GirlDao girlDao = sqlSession.getMapper(GirlDao.class);
 int row = girlDao.insert(girl);
 System.out.println(row);

  System.out.println(girl);
 }

 @Test
 void test3() {
  SqlSession sqlSession = sqlSessionFactory.openSession();
  List<Girl> girlList = new ArrayList<Girl>();
  for (int i = 0; i < 3; i++) {
  Girl girl = new Girl();
  girl.setGAge(34 + i);
  girl.setGName("柳岩" + i);
  girlList.add(girl);
  }
  GirlDao girlDao = sqlSession.getMapper(GirlDao.class);
  girlDao.insert1(girlList);
 }
```

二.日志配置:

步骤 1：添加 Log4J 的 jar 包
maven中导架包
步骤 2：配置 Log4J



在应用的类路径中创建一个名称为 `log4j.properties` 的文件，文件的具体内容如下：

```
# Global logging configuration
log4j.rootLogger=ERROR, stdout
# MyBatis logging configuration...
log4j.logger.org.mybatis.example.BlogMapper=TRACE
# Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

```
# Global logging configuration
log4j.rootLogger=ERROR, stdout
# MyBatis logging configuration...
log4j.logger.com.lanou.dao=TRACE
# Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

三.表与表的关系:(通过log4j我们可以清楚的看到表表的对应关系)

1.单对单

2.一对多

3,多对多

举例说明,单对单的关系:

(一个男生对一个女生)

```
<mapper namespace="com.lanou.dao.AManDao">
全部加别名:
 <!--<select id="selectAll" resultType="com.lanou.pojo.AMan">-->
 <!--select-->
 <!--m.id id,m.name name,m.age age,
 w.id 'woman.id',w.name 'woman.name',w.age 'woman.age' from a_man m-->
 <!--join a_woman w on m.id = w.id;-->
 <!--</select>-->

  <resultMap id="AMan" type="com.lanou.pojo.AMan">
  <id property="id" column="id"/>
  <result property="name" column="name"/>
  <result property="age" column="age"/>
  全部加前缀:
  <association property="woman" javaType="com.lanou.pojo.AWoman" columnPrefix="woman_">
  <id property="id" column="id"/>
  <result property="name" column="name"/>
  <result property="age" column="age"/>
  </association>
  </resultMap>

  <select id="selectAll" resultMap="AMan">
  select m.id id,m.name name,m.age age,w.id woman_id,w.name 
 woman_name,w.age woman_age
  from a_man m
  join a_woman w on m.id = w.id
  </select>
 </mapper>
```

测试:

```
@Test
void test4() {

 SqlSession sqlSession = sqlSessionFactory.openSession();
 AManDao aManDao = sqlSession.getMapper(AManDao.class);
 List<AMan> aManList = aManDao.selectAll();
 for (AMan aMan : aManList) {
 System.out.println(aMan);
 //System.out.println(aMan.getWoman());
  }
 }
```

一对多的关系:(简单的说古代一夫多妻)

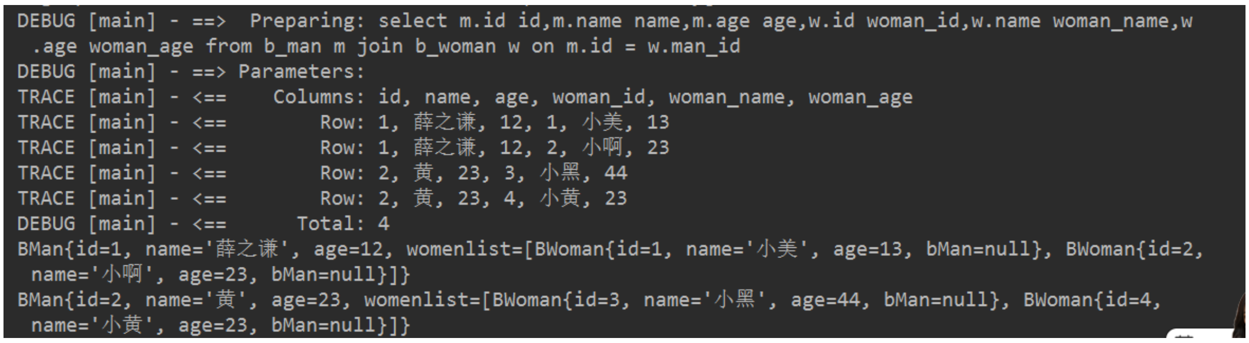
```
<mapper namespace="com.lanou.dao.BManDao">

 <resultMap id="bMan" type="com.lanou.pojo.BMan">
 <id property="id" column="id"/>
 <result property="name" column="name"/>
 <result property="age" column="age"/>
 <!--
 collection:属性的类型是集合时使用
 property:属性名
  ofType:集合中的元素类型
  columnPrefix:字段的前缀
  -->
  <collection property="womenlist" ofType="com.lanou.pojo.BWoman" columnP
refix="woman_">
  <id property="id" column="id"/>
  <result property="name" column="name"/>
  <result property="age" column="age"/>
  </collection>
  </resultMap>

  <select id="selectAll" resultMap="bMan">
  select m.id id,m.name name,m.age age,w.id woman_id,w.name
woman_name,w.age woman_age
  from b_man m
  join b_woman w on m.id = w.man_id
  </select>
 </mapper>
```

```
private long id;
private String name;
private long age;
private List<BWoman> womenlist;//多妻
```

测试:

```
@Test
void test5() {
 SqlSession sqlSession = sqlSessionFactory.openSession();
 BManDao bManDao = sqlSession.getMapper(BManDao.class);
 List<BMan> bManList = bManDao.selectAll();
 for (BMan bMan : bManList) {
 System.out.println(bMan);
 }
}
```

结果:

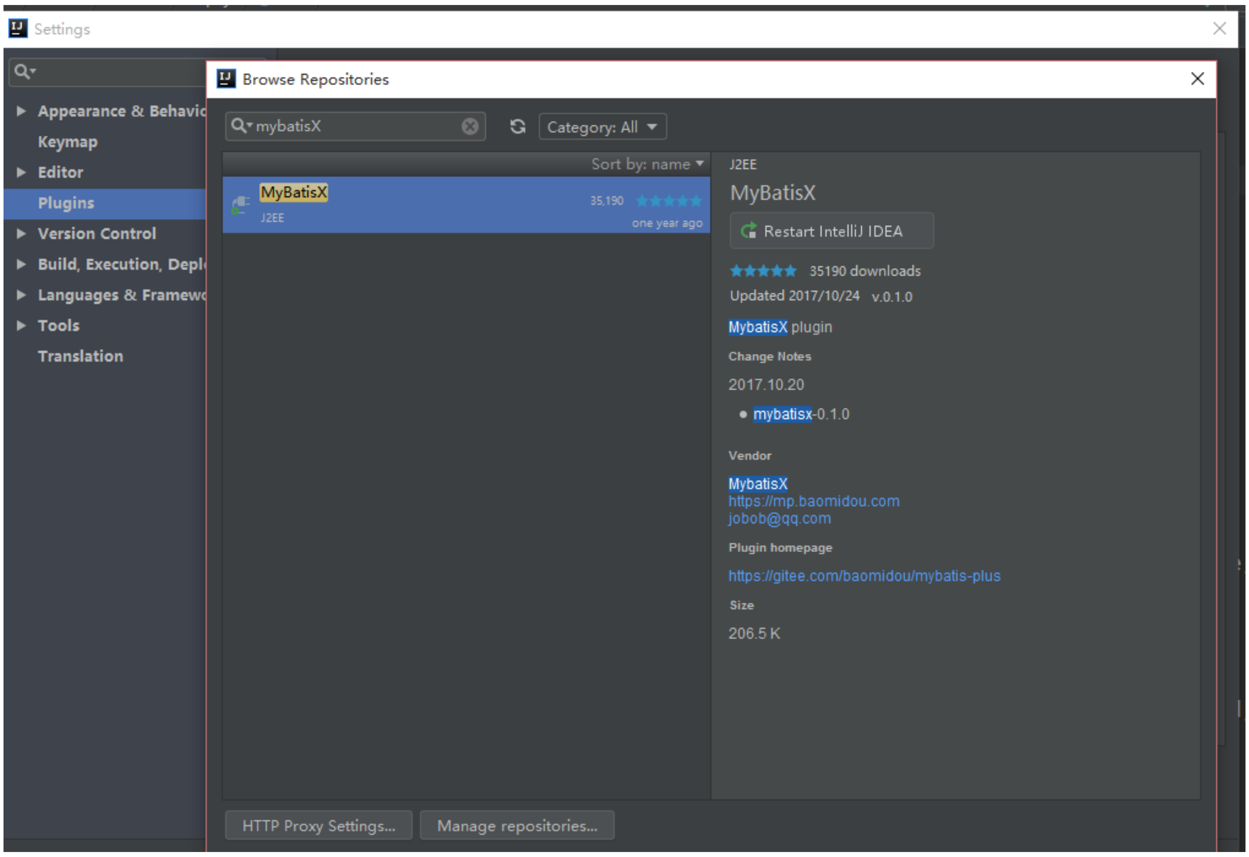


```
DEBUG [main] - ==>  Preparing: select m.id id,m.name name,m.age age,w.id woman_id,w.name woman_name,w
 .age woman_age from b_man m join b_woman w on m.id = w.man_id
DEBUG [main] - ==> Parameters: 
TRACE [main] - <==    Columns: id, name, age, woman_id, woman_name, woman_age
TRACE [main] - <==        Row: 1, 薛之谦, 12, 1, 小美, 13
TRACE [main] - <==        Row: 1, 薛之谦, 12, 2, 小啊, 23
TRACE [main] - <==        Row: 2, 黄, 23, 3, 小黑, 44
TRACE [main] - <==        Row: 2, 黄, 23, 4, 小黄, 23
DEBUG [main] - <==      Total: 4
BMan{id=1, name='薛之谦', age=12, womenlist=[BWoman{id=1, name='小美', age=13, bMan=null}, BWoman{id=2,
 name='小啊', age=23, bMan=null}]}
BMan{id=2, name='黄', age=23, womenlist=[BWoman{id=3, name='小黑', age=44, bMan=null}, BWoman{id=4,
 name='小黄', age=23, bMan=null}]}
```

3.多对多的关系:

必须有中间表,中间表有两个外键,两个外键,一个连接men_id,一个连接wonmen_id.

pojo中加



MyBatisX
J2EE
35,190
one year ago

J2EE
MyBatisX
35190 downloads
Updated 2017/10/24 v.0.1.0
MybatisX plugin
Change Notes
2017.10.20
- mybatisx-0.1.0

Vendor
MybatisX
https://mp.baomidou.com
jobob@qq.com
Plugin homepage
https://gitee.com/baomidou/mybatis-plus
Size
206.5 K

5.1. #{}和${}

#{}：表示 一个占位符，表示使用 prepareStatement 的?来设置参数，能有效防止 SQL 注入

${}表示 sql 拼接，变量直接拼接到 Sql 里