Mybatis框架技术

版本：3.6.0

第3章 Mybatis逆向工程和动态SQL

本章教学目标

- 了解什么是逆向工程；
- 掌握逆向工程的使用；
- 掌握if、where标签的使用；
- 掌握trim、set标签的使用；
- 掌握foreach、choose标签的使用；
- 掌握SQL片段的使用；

本章教学内容

Neuedu

节	知识点	掌握程度	难易程度	教学形式	对应在线微课
Mybatis逆向工程	什么是逆向工程	了解		线下	什么是逆向工程
	使用逆向工程	掌握		线下	使用逆向工程
动态SQL	if+where标签	掌握		线下	if+where标签
	trim标签	掌握	难	线下	trim标签
	set标签	掌握		线下	set标签
	foreach标签	掌握	难	线下	foreach标签
	choose标签	掌握	难	线下	choose标签
	SQL片段	掌握		线下	SQL片段
本章实战项目任务实现	实战项目任务实现	掌握		线下	实战项目任务实现

本章实战项目任务

Neuedu

- 通过本章内容的学习，使用动态SQL，完成《跨境电商系统》用户注册功能。
  - 如果用户名已经存在，注册失败，输出提示信息
  - 如果用户名不存在，注册成功

- 运行效果在控制台输出：

```
DEBUG [main] - ==>  Preparing: select * from SYS_USER WHERE USER_NAME = ? and STATUS = 'A'
DEBUG [main] - ==> Parameters: lisi5(String)
DEBUG [main] - <==      Total: 0
DEBUG [main] - ==>  Preparing: insert into sys_user ( USER_ID, USER_NAME, PASSWORD, NICK_NAME,
DEBUG [main] - ==> Parameters: lisi5(String), 123456(String), 李四(String), 2(String), 2(String)
DEBUG [main] - <==    Updates: 1
DEBUG [main] - ==>  Preparing: select seq_sysuser.currval from dual
DEBUG [main] - ==> Parameters:
DEBUG [main] - <==      Total: 1
DEBUG [main] - Committing JDBC Connection [oracle.jdbc.driver.T4CConnection@382db087]
 INFO [main] - 新增用户成功
```

```
DEBUG [main] - ==>  Preparing: select * from SYS_USER WHERE USER_NAME = ? and STATUS = 'A'
DEBUG [main] - ==> Parameters: lisi(String)
DEBUG [main] - <==      Total: 11
ERROR [main] - 用户名“lisi”已被使用，重新换一个试试
```

CONTENTS 目录

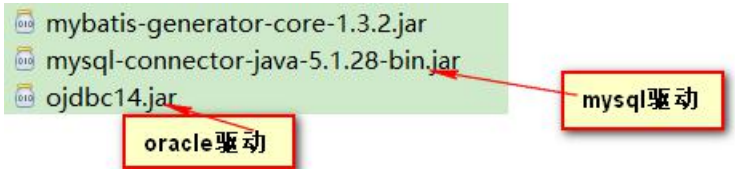
什么是逆向工程

Neuedu

- MyBatis的一个主要的特点就是需要程序员自己编写sql，那么如果表太多的话，难免会很麻烦，所以Mybatis官方提供了一个逆向工程，可以针对单表自动生成Mybatis执行所需要的代码（包括mapper.xml、mapper.java、po..）。
- 一般在开发中，常用的逆向工程方式是通过数据库的表生成代码。

使用逆向工程

- 1、新建一个java工程
- 2、引入使用的jar包



- 3、添加配置文件generatorConfig.xml
  - 设置数据库驱动、配置、包名、文件保存位置、表名等

- 4、定义GeneratorSqlmap类，调用Mybatis自动创建接口，在main方法执行自动创建

配置文件generatorConfig.xml

Neuedu

- 设置数据库驱动、url、用户名和密码

```
<jdbcConnection driverClass="oracle.jdbc.OracleDriver"
    connectionURL="jdbc:oracle:thin:@10.10.21.9:1521:orcl"
    userId="scott"
    password="tiger">
</jdbcConnection>
```

- 生成的POJO类的位置

```
<!-- targetProject:生成PO类的位置 -->
<javaModelGenerator targetPackage="com.neuedu.pojo"
    targetProject=".\src">
    <!-- enableSubPackages:是否让schema作为包的后缀 -->
    <property name="enableSubPackages" value="false" />
    <!-- 从数据库返回的值被清理前后的空格 -->
    <property name="trimStrings" value="true" />
</javaModelGenerator>
```

- 全部代码参见：ch03-generatorSqlmap工程

配置文件generatorConfig.xml

Neuedu

- Mapper映射文件生成位置

```
<!-- targetProject:mapper映射文件生成的位置 -->
<sqlMapGenerator targetPackage="com.neuedu.mapper"
    targetProject=".\src">
    <!-- enableSubPackages:是否让schema作为包的后缀 -->
    <property name="enableSubPackages" value="false" />
</sqlMapGenerator>
```

- Mapper接口生成位置

```
<!-- targetPackage: mapper接口生成的位置 -->
<javaClientGenerator type="XMLMAPPER"
    targetPackage="com.neuedu.mapper"
    targetProject=".\src">
    <!-- enableSubPackages:是否让schema作为包的后缀 -->
    <property name="enableSubPackages" value="false" />
</javaClientGenerator>
```

- 全部代码参见：ch03-generatorSqlmap工程

配置文件generatorConfig.xml

Neuedu

- 根据DB表生成POJO类

```
<table tableName="T_USER" domainObjectName="User"
 enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false"
       enableSelectByExample="false" selectByExampleQueryId="false" >
       <columnOverride column="id" javaType="Integer" />
       </table>
```

  - tableName:要生成的表名
  - domainObjectName:生成后的实例名
  - enableCountByExample:Count语句中加入where条件查询，默认为true开启
  - enableUpdateByExample:Update语句中加入where条件查询，默认为true开启
  - enableDeleteByExample:Delete语句中加入where条件查询，默认为true开启
  - enableSelectByExample:Select多条语句中加入where条件查询，默认为true开启
  - selectByExampleQueryId:Select单个对象语句中加入where条件查询，默认为true开启
  - 全部代码参见：ch03-generatorSqlmap工程

使用逆向工程

Neuedu

- 调用Mybatis自动创建接口

```
public class GeneratorSqlmap {

    public void generator() throws Exception{
        List<String> warnings = new ArrayList<String>();
        boolean overwrite = true;
        //指定 逆向工程配置文件
        File configFile = new File("generatorConfig.xml");
        ConfigurationParser cp = new ConfigurationParser(warnings);
        Configuration config = cp.parseConfiguration(configFile);
        DefaultShellCallback callback = new DefaultShellCallback(overwrite);
        MyBatisGenerator myBatisGenerator = new MyBatisGenerator(config,
                callback, warnings);
        myBatisGenerator.generate(null);

    }
```

- 全部代码参见：ch03-generatorSqlmap工程

使用逆向工程

Neuedu

- GeneratorSqlmap的main方法执行后会生成3个文件：
  - User.java

```
public class User {
    private Integer id;

    private String username;

    private Date birthday;

    private String sex;

    private String address;
```

  - UserMapper.java

```
public interface UserMapper {
    int deleteByPrimaryKey(Integer id);

    int insert(User record);

    int insertSelective(User record);

    User selectByPrimaryKey(Integer id);

    int updateByPrimaryKeySelective(User record);

    int updateByPrimaryKey(User record);
}
```

使用逆向工程

Neuedu

- UserMapper.xml

```
<mapper namespace="com.neuedu.mapper.UserMapper" >
  <resultMap id="BaseResultMap" type="com.neuedu.pojo.User" >
    <id column="ID" property="id" jdbcType="DECIMAL" />
    <result column="USERNAME" property="username" jdbcType="VARCHAR" />
    <result column="BIRTHDAY" property="birthday" jdbcType="DATE" />
    <result column="SEX" property="sex" jdbcType="CHAR" />
    <result column="ADDRESS" property="address" jdbcType="VARCHAR" />
  </resultMap>
  <sql id="Base_Column_List" >
    ID, USERNAME, BIRTHDAY, SEX, ADDRESS
  </sql>
  <select id="selectByPrimaryKey" resultMap="BaseResultMap" parameterType="Integer" >
    select
    <include refid="Base_Column_List" />
    from T_USER
    where ID = #{id,jdbcType=DECIMAL}
  </select>
  <delete id="deleteByPrimaryKey" parameterType="Integer" >
    delete from T_USER
    where ID = #{id,jdbcType=DECIMAL}
  </delete>
  <insert id="insert" parameterType="com.neuedu.pojo.User" >
    insert into T_USER (ID, USERNAME, BIRTHDAY,
      SEX, ADDRESS)
    values (#{id,jdbcType=DECIMAL}, #{username,jdbcType=VARCHAR}, #{birthday,jdbcType=DATE},
      #{sex,jdbcType=CHAR}, #{address,jdbcType=VARCHAR})
  </insert>
```

- 全部代码参见：ch03-generatorSqlmap工程

课堂练习（15分钟）

- 使用逆向工程，生成与商品items表对应的pojo类、mapper接口及mapper.xml文件

CONTENTS 目录

动态SQL

Neuedu

- 动态SQL
  - 主要用于解决查询条件不确定的情况：在程序运行期间，根据提交的查询条件进行查询。
  - 通过MyBatis提供的各种标签对条件作出判断以实现动态拼接SQL语句。
  - MyBatis的动态SQL是基于OGNL表达式的。
- 使用动态SQL的原因
  - 提供的查询条件不同，执行的SQL语句不同。若将每种可能的情况均逐一列出，就将出现大量的SQL语句。

- MyBatis中用于实现动态SQL的元素主要有：
  - if语句
  - where
  - trim
  - set。
  - foreach
  - choose（when，otherwize）

If + where标签

- if标签
  - 就是简单的条件判断，利用if语句我们可以实现某些简单的条件选择。
- where标签
  - 会在写入where元素的地方输出一个where，
  - 不需要考虑where元素里面的条件输出是什么样子的，MyBatis会智能的处理
  - 如果所有的条件都不满足，那么MyBatis就会查出所有的记录
  - 如果输出后是and开头的，MyBatis会把第一个and忽略
  - 如果是or开头的，MyBatis也会把它忽略
  - 在where标签中不需要考虑空格的问题，MyBatis会智能的加上

if + where标签

- 实例：实现用户查询（根据条件查询）

```
<select id="findUserList" parameterType="com.neuedu.pojo.UserQueryVo"
        resultType="com.neuedu.pojo.UserCustom">
SELECT * FROM T_USER
<!--
where可以自动去掉条件中的第一个and
 -->
<where>
    <if test="userCustom!=null">
        <if test="userCustom.sex!=null and userCustom.sex!=''">
            and T_USER.sex = #{userCustom.sex}
        </if>
        <if test="userCustom.username!=null and userCustom.username!=''">
            and T_USER.username LIKE '%${userCustom.username}%'
        </if>
    </if>
</where>
```

- 全部代码参见：ch03-mybatis01工程

课堂练习（15分钟）

- 根据性别及地址（模糊）查询用户信息
  - 使用if+where判断查询条件

trim标签

- trim标签的主要功能是可以在自己包含的内容前加上某些前缀，也可以在其后加上某些后缀，与之对应的属性是prefix和suffix；
- 可以把包含内容的首部某些内容覆盖，即忽略，也可以把尾部的某些内容覆盖，对应的属性是prefixOverrides和suffixOverrides；

- 实例：实现用户添加
- 代码参见：
  ch03-generatorSqlmap工程

```
<insert id="insertSelective" parameterType="com.neuedu.pojo.User" >
  insert into T_USER
  <trim prefix="(" suffix=")" suffixOverrides="," >
    <if test="id != null" >
      ID,
    </if>
    <if test="username != null" >
      USERNAME,
    </if>
    <if test="birthday != null" >
      BIRTHDAY,
    </if>
    <if test="sex != null" >
      SEX,
    </if>
    <if test="address != null" >
      ADDRESS,
    </if>
  </trim>
  <trim prefix="values (" suffix=")" suffixOverrides="," >
    <if test="id != null" >
      #{id,jdbcType=DECIMAL},
    </if>
    <if test="username != null" >
      #{username,jdbcType=VARCHAR},
    </if>
    <if test="birthday != null" >
      #{birthday,jdbcType=DATE},
    </if>
```

set 标签

- set标签主要是用在更新操作的时候，它的功能和where标签差不多。
- 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，则会导致逗号多余错误。
- 使用set标签可以在包含的语句前输出一个set，然后如果包含的语句是以逗号结束的话将会把该逗号忽略。

- 实例：实现用户更新
- 全部代码参见：
  ch03-mybatis02工程

```
<update id="updateUser" parameterType="user">
    update t_user
    <set>
        <if test="username !=null and username !=''">
            username = #{username},
        </if>
        <if test="birthday != null">
            birthday = #{birthday},
        </if>
        <if test="sex != null and sex !=''">
            sex = #{sex},
        </if>
        <if test="address != null and address !=''">
            address = #{address},
        </if>
    </set>
    where id=#{id}
</update>
```

课堂练习（20分钟）

- 实现添加用户的功能
  - 应用trim标签

foreach 标签

- foreach标签主要用在构建in条件中，它可以在SQL语句中进行迭代一个集合。
- foreach标签的属性主要有 item，index，collection，open，separator，close。
  - item表示集合中每一个元素进行迭代时的别名
  - index指定一个名字，用于表示在迭代过程中，每次迭代到的位置
  - open表示该语句以什么开始
  - separator表示在每次进行迭代之间以什么符号作为分隔符
  - close表示以什么结束

foreach 标签

- foreach标签的collection属性是必须指定的，但是在不同情况下，该属性的值是不一样的，主要有以下3种情况：
  - 如果传入的是单参数且参数类型是一个List的时候，collection属性值为list
  - 如果传入的是单参数且参数类型是一个array数组的时候，collection的属性值为array
  - 如果传入的参数是多个的时候，需要把它们封装成一个Map，当然单参数也可以封装成Map
  - 实际上如果在传入参数的时候，在MyBatis里面也是会把它封装成一个Map的，Map的key就是参数名，所以这个时候collection属性值就是传入的List或Array对象在自己封装的Map里面的key

foreach 标签

Neuedu

- 1、单参数List的类型
  - UserMapper.java

```
public List<UserCustom> findUserByForeach1(List<Integer> ids) throws Exception;
```

  - UserMapper.xml中，有如下两种写法

```
<!-- 使用实现下边的sql拼接:
 AND (id=1 OR id=10 OR id=16)
 -->
<foreach collection="list" item="user_id" open="AND (" close=")" separator="or">
   每个遍历需要拼接的串
   id=#{user_id}
</foreach>
```

```
<!-- 实现 " and id IN(1,10,16)"拼接 -->
<foreach collection="list" item="user_id" open="and id IN(" close=")" separator=",">
    #{user_id}
</foreach>
```

  - 测试代码

```
//创建List对象，设置查询条件,传入多个id
List<Integer> ids = new ArrayList<Integer>();
ids.add(1);
ids.add(206);
ids.add(207);
//调用userMapper的方法
List<UserCustom> list = userMapper.findUserByForeach1(ids);
```

- 全部代码参见：
  ch03-mybatis03工程

foreach 标签

Neuedu

- 2、单参数数组的类型
  - UserMapper.java

```
public List<UserCustom> findUserByForeach2(int[] ids) throws Exception;
```

  - UserMapper.xml中，有如下两种写法

```
<!-- 使用实现下边的sql拼接：
 AND (id=1 OR id=10 OR id=16)
 -->
<foreach collection="array" item="user_id" open="AND (" close=")" separator="or">
   <!-- 每个遍历需要拼接的串 -->
   id=#{user_id}
</foreach>
```

```
<!-- 实现 “ and id IN(1,10,16)”拼接 -->
<foreach collection="array" item="user_id" open="and id IN(" close=")" separator=",">
   #{user_id}
</foreach>
```

  - 测试代码

```
//创建int数组，设置查询条件,传入多个id
int[] ids = {1,206,207};
//调用userMapper的方法
List<UserCustom> list = userMapper.findUserByForeach2(ids);
```

- 全部代码参见：
  ch03-mybatis03工程

foreach 标签

Neuedu

- 3、 Map类型参数
  - UserMapper.java

```
public List<UserCustom> findUserByForeach3(Map<String, Object> params) throws Exceptio
```

  - UserMapper.xml中，有如下两种写法

```
SELECT * FROM T_USER
<where>
    T_USER.username LIKE '%${name}%'
 <!-- 使用实现下边的sql拼接:
  AND (id=1 OR id=10 OR id=16)
  -->
  <foreach collection="ids" item="user_id" open="AND (" close=")" separator="or">
    <!-- 每个遍历需要拼接的串 -->
    id=#{user_id}
</foreach>
```

  - 测试代码

```
//创建Map对象，设置查询条件,传入多个id
Map<String, Object> map = new HashMap<String, Object>();
map.put("name", "小军");
map.put("ids", new int[]{1,206,207});
//调用userMapper的方法
List<UserCustom> list = userMapper.findUserByForeach3(map);
```

- 全部代码参见：
  ch03-mybatis03工程

课堂练习（20分钟）

- 实现用户信息的批量删除
  - 使用foreach标签

choose 标签

- choose标签
  - 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。
- 实例：查询默认性别为男的用户
- 代码参见：ch03-mybatis03工程

```
<!-- 多条件查询用户，默认查询性别为男，即sex=1 -->
<select id="findUserByChoose" parameterType="UserQueryVo" resultType="UserCustom">
    select * from T_USER where 1 = 1
    <choose>
        <when test="userCustom.username !=null and userCustom.username !='' ">
            and username LIKE '%${userCustom.username}%'
        </when>
        <when test="userCustom.sex != null and userCustom.sex !=''">
            and sex = #{userCustom.sex}
        </when>
        <otherwise>
            and sex = '1'
        </otherwise>
    </choose>
</select>
```

SQL片段

- 在实际开发中，存在大量的重复的SQL代码

```
<select id="findUserList" parameterType="com.neuedu.pojo.UserQueryVo"
        resultType="com.neuedu.pojo.UserCustom">
SELECT * FROM T_USER
<!--
where可以自动去掉条件中的第一个and
 -->
<where>
    <if test="userCustom!=null">
        <if test="userCustom.sex!=null and userCustom.sex!=''">
            and T_USER.sex = #{userCustom.sex}
        </if>
        <if test="userCustom.username!=null and userCustom.username!=''">
            and T_USER.username LIKE '%${userCustom.username}%'
        </if>
    </if>
</where>
</select>
```

```
<select id="findUserCount" parameterType="com.neuedu.pojo.UserQueryVo" resultType="i
   SELECT count(*) FROM T_USER
      <!--
    where可以自动去掉条件中的第一个and
     -->
    <where>
        <if test="userCustom!=null">
            <if test="userCustom.sex!=null and userCustom.sex!=''">
                and T_USER.sex = #{userCustom.sex}
            </if>
            <if test="userCustom.username!=null and userCustom.username!=''">
                and T_USER.username LIKE '%${userCustom.username}%'
            </if>
        </if>
    </where>
</select>
```

- 全部代码参见：ch03-mybatis01工程

SQL片段

- 通过SQL代码片段，实现SQL代码重用

```
<!-- 定义sql片段
id：sql片段的唯一标识

经验：是基于单表来定义sql片段，这样这个sql片段可重用性才高
在sql片段中不要包括 where
 -->
<sql id="query_user_where">
    <if test="userCustom!=null">
        <if test="userCustom.sex!=null and userCustom.sex!=''">
            and T_USER.sex = #{userCustom.sex}
        </if>
        <if test="userCustom.username!=null and userCustom.username!=''">
            and T_USER.username LIKE '%${userCustom.username}%'
        </if>
    </if>
</sql>
```

```
<select id="findUserList" parameterType="com.neuedu.pojo.UserQueryVo"
        resultType="com.neuedu.pojo.UserCustom">
SELECT * FROM T_USER
<!--
where可以自动去掉条件中的第一个and
 -->
<where>
    <!-- 引用sql片段 的id，如果refid指定的id不在本mapper文件中，需要前边加namespace -->
    <include refid="query_user_where"></include>
    <!-- 在这里还要引用其它的sql片段 -->
</where>
```

- 全部代码参见：ch03-mybatis04工程

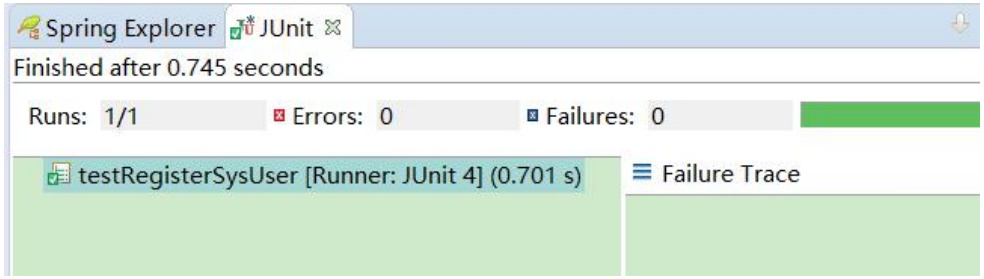
课堂练习（10分钟）

- 理解并完成课堂中的授课案例

CONTENTS 目录

本章实战项目任务实现

- 通过本章内容的学习，使用动态SQL，完成《跨境电商系统》用户注册功能。
  - 如果用户名已经存在，注册失败，输出提示信息
  - 如果用户名不存在，注册成功
- 运行效果在控制台输出：




```
DEBUG [main] - ==>  Preparing: select * from SYS_USER WHERE USER_NAME = ? and STATUS = 'A'
DEBUG [main] - ==> Parameters: lisi5(String)
DEBUG [main] - <==      Total: 0
DEBUG [main] - ==>  Preparing: insert into sys_user ( USER_ID, USER_NAME, PASSWORD, NICK_NAME,
DEBUG [main] - ==> Parameters: lisi5(String), 123456(String), 李四(String), 2(String), 2(String)
DEBUG [main] - <==    Updates: 1
DEBUG [main] - ==>  Preparing: select seq_sysuser.currval from dual
DEBUG [main] - ==> Parameters:
DEBUG [main] - <==      Total: 1
DEBUG [main] - Committing JDBC Connection [oracle.jdbc.driver.T4CConnection@382db087]
 INFO [main] - 新增用户成功
```

```
DEBUG [main] - ==>  Preparing: select * from SYS_USER WHERE USER_NAME = ? and STATUS = 'A'
DEBUG [main] - ==> Parameters: lisi(String)
DEBUG [main] - <==      Total: 11
ERROR [main] - 用户名“lisi”已被使用，重新换一个试试
```

本章重点总结

- 了解什么是逆向工程；
- 掌握逆向工程的使用；
- 掌握if、where标签的使用；
- 掌握trim、set标签的使用；
- 掌握foreach、choose标签的使用；
- 掌握SQL片段的使用；

课后作业【必做任务】

- 1、使用动态SQL，完成《跨境电商系统》用户登录功能。
  - 用户名和密码正确，登录成功
  - 用户名或者密码错误，登录失败

- 2、使用动态SQL，完成《跨境电商系统》用户修改功能。

课后作业【选做任务】

- 使用动态SQL，完成订单商品系统用户信息的综合查询功能。
  - 查询条件使用包装的pojo类型
  - 查询条件包括性别、用户名（模糊）、用户id（多个）

课后作业【线上任务】

- 线上任务
  - 安排学员线上学习任务（安排学员到睿道实训平台进行复习和预习的任务，主要是进行微课的学习）