# 本篇是本人看完所有Mybatis教程视频后总结的内容

# Mybatis框架学习3——连接池、事务与sql

## 主要学习内容：

Mybatis的连接池与事务控制

Mybatis的动态sql语句

Mybatis的多表查询

## Mybatis的连接池技术

在 Mybatis 的 SqlMapConfig.xml 配置文件中，通过<dataSource type=”pooled”>来实现 Mybatis 中连接池的配置。

Mybatis的数据源分为三类：

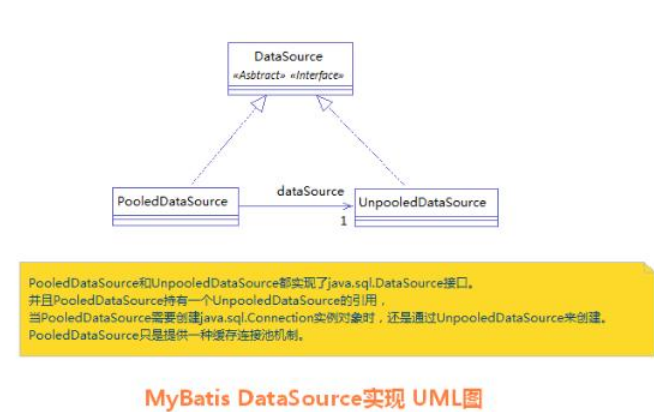
UNPOOLED 不使用连接池的数据源

POOLED 使用连接池的数据源

JNDI 使用 JNDI 实现的数据源

所谓数据源可以理解为使用连接池技术来更好的管理数据库连接

在本次课程学习中，采用的都是POOLED数据源。



* 1. 数据源配置

在SqlMapConfig.xml的<environments>中配置数据源

<dataSource type="POOLED">

<!-- 配置连接数据库的4个基本信息 -->

<property name="driver" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost:3306/eesy\_mybatis"/>

<property name="username" value="root"/>

<property name="password" value="1234"/>

</dataSource>

type=”POOLED”,代表mybatis创建pooledDataSource实例，每次使用的时候，从连接池中获取一个连接使用，然后返回使用。

1.2数据源获取

配置完数据源之后，mybatis根据工厂模式来创建数据源DataSource实例，并将其放入Configuration对象中的Environment对象中。

1.3Mybatis连接的获取过程

当创建SqlSession对象并执行Sql语句时，Mybatis才会调用datasource对象来创建connection。**也就是说，java.sql.Connection对象的创建一直延迟到执行SQL语句 的时候。**

InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml"); SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(in);

SqlSession sqlSession = factory.openSession();

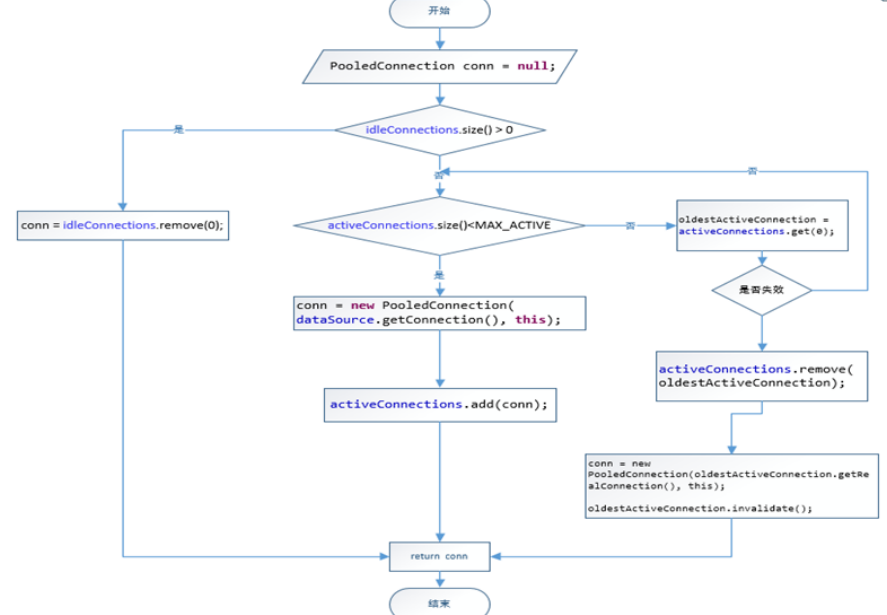
List list = sqlSession.selectList("findUserById",41);

System.out.println(list.size());

在上述代码中，只有当sqlSession执行selectList时候，connection才创建出来。

可以只写属性名）。

根据分析PooledDataSource的源代码，可得PooledDataSource得工作原理如下：



**1.4 小结**

**由此可知，真正打开数据库连接的时间点是代码执行sql语句的时候，也就是说当我们真正要访问数据库的时候才会获取并打开连接，**用完了就再 立即将数据库连接归还到连接池中，这样很大的节省了连接池资源。

## 2、Mybatis 的事务控制

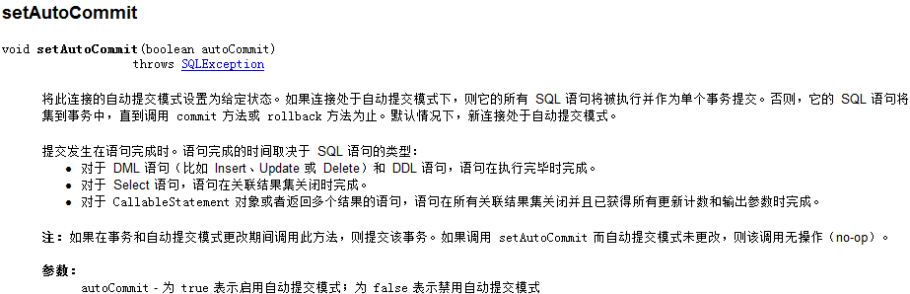
2.1 自动提交事务和手动提交事务

在之前的手动提交事务时，需要在sql的CUD操作执行后调用

sqlsession.commit();

来进行事务的提交，否则对数据库进行的操作无法提交到数据库。

Mybatis 框架的事务控制方式，本身是用 JDBC 的 setAutoCommit()方法来设置事务提交方式的。



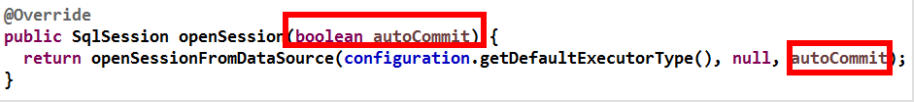
**在自动提交模式下，所有的Sql语句被执行时作为单个事务提交。**

**2.2 设置自动提交事务**

事实上，需要执行sqlsession.commit();来手动提交事务的原因是我们每次从连接池中取出连接，都会将调用 connection.setAutoCommit(false)方法，而设置自动提交事务的方法即在创建Sqlsession时，传入一个true作为参数：

session = factory.openSession(true);

这个参数传入后，会将autoCommit的值设为true并传给连接。



此时，不需要再手动提交事务sqlsession.commit();事务也能自动的提交了。

## 3、Mybatis的动态SQL语句

**3.1 <if>标签**

**满足if条件才执行，比如**在 id 如果不为空时可以根据 id 查询， 如果 username 不同空时还要加入用户名作为条件。这种情况在我们的多条件组合查询中经常会碰到。

<select id="findUserByCondition" resultMap="userMap" parameterType="user">

select \* from user where 1=1

<if test="userName != null">

and username = #{userName}

</if>

<if test="userSex != null">

and sex = #{userSex}

</if>

</select>

**在这里使用where 1=1 是为了满足多个条件的查询。**

**这样，就可以通过<if>标签来查询满足的记录了。**

**3.2<where>标签**

为了简化上面 where 1=1 的条件拼装，也可以采用<where>标签来简化开发。

<select id="findUserByCondition" resultMap="userMap" parameterType="user">

select \* from user

<where>

<if test="userName != null">

and username = #{userName}

</if>

<if test="userSex != null">

and sex = #{userSex}

</if>

</where>

</select>

**这样就不用添加条件 where 1=1 了。**

**3.3<foreach>标签**

<select id="findUserInIds" resultMap="userMap" parameterType="queryvo">

<include refid="defaultUser"></include>

<where>

<if test="ids != null and ids.size()>0">

<foreach collection="ids" open="and id in (" close=")" item="uid" separator=",">

#{uid}

</foreach>

</if>

</where>

</select>

**在where中使用<foreach>标签，用于遍历集合元素。**

collection:代表要遍历的集合元素，注意编写时不要写#{}

open:代表语句的开始部分

close:代表结束部分

item:代表遍历集合的每个元素，生成的变量名

sperator:代表分隔符

这条sql语句为：select \* from user where id in (？);

3.4 简化SQL片段

Mybatis可以将重复的 sql 提取出来，使用时用 include 引用即可，最终达到 sql 重用的目的。

3.4.1 定义SQL片段

<sql id="defaultUser">

select \* from user

</sql>

**抽取重复的sql语句片段，设置一个id**

**3.4.2 引用**SQL**片段**

<select id="findById" parameterType="INT" resultMap="userMap">

<include refid=”defaultSql”> </include>

where id = #{uid}

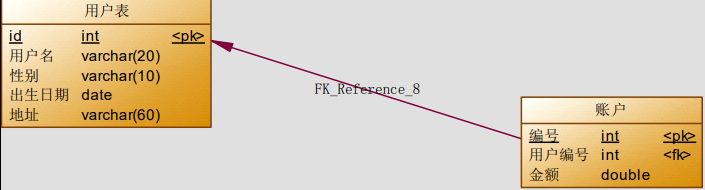
</select>

**使用<include>标签将sql片段引入进来，得到完整的sql语句。**

**上述sql语句为**select \* from user where id = #{uid}

## 4、Mybatis的多表查询

视频中所使用的案例为为简单的用户和账户的模型来分析 Mybatis 多表关系。用户为 User 表，账户为Account 表。一个用户（User）可以有多个账户（Account）。具体关系如下：



**4.1** 一对一查询

因为一个账户信息只能供某个用户使用，所以从查询账户信息出发关联查询用户信息为一对一查询。有两种方式实现。

4.1.1定义AccountUser类

这个类包含了账户信息和用户信息，这样就可以接受查询结果返回的账户信息和用户信息了。

<select id="findAllAccount" resultType="accountuser">

select a.\*,u.username,u.address from account a , user u where u.id = a.uid;

</select>

在持久层接口的映射文件中将resultType设为accountUser，使用sql的表关联查询，这样就同时接收了账户信息和用户信息。

**4.1.2 定义resultMap**

在Account类中添加User类作为Account类的一个属性。

这样的话，查询结果只用返回Account类，因为 Account 类中包含了一个 User 类的对象，它可以封装账户所对应的用户信息。

<resultMap id="accountUserMap" type="account">

<id property="id" column="aid"></id>

<result property="uid" column="uid"></result>

<result property="money" column="money"></result>

<association property="user" column="uid" javaType="user">

<id property="id" column="id"></id>

<result column="username" property="username"></result>

<result column="address" property="address"></result>

<result column="sex" property="sex"></result>

<result column="birthday" property="birthday"></result>

</association>

</resultMap>

使用<resultMap>标签来建立对应关系，**<association>标签表示一对一的关系映射，用于配置User类中的信息。**

**4.2 一对多查询**

如果从用户信息出发查询用户下的账户信息则为一对多查询，因为一个用户可以有多个账户。

首先，sql语句同样使用关联表查询，在这里使用左外连接查询比较合适。

在一对多的查询中，我们使用resultMap的方式来实现，因此，在User类中类加入 List<Account>来获取所有账户信息。

private List<Account> accounts;

在Dao映射文件中，配置resultMap:

<resultMap id="userAccountMap" type="user">

<id property="id" column="id"></id>

<result property="username" column="username"></result>

<result property="address" column="address"></result>

<result property="sex" column="sex"></result>

<result property="birthday" column="birthday"></result>

<collection property="accounts" ofType="account">

<id column="aid" property="id"></id>

<result column="uid" property="uid"></result>

<result column="money" property="money"></result>

</collection>

</resultMap>

**其中<collection>是用于建立一对多中集合属性的对应关系,<property>指定**关联查询的结果集存储在 User 对象的上哪个属性，**<ofType>用于指定集合元素的数据类型。**

**4.3 多对多查询**

在前两节中，使用 Mybatis 实现一对多关系的维护。**多对多关系其实我们看成是双向的一对多关系。**

查询角色我们需要用到Role表，但角色分配的用户的信息我们并不能直接找到用户信息，而是要通过中 间表(USER\_ROLE 表)才能关联到用户信息，从 User 出发，我们也可以发现一个用户可以具有多个角色，这样用户到角色的关系也还是一对多关系。这样 我们就可以认为 User 与 Role 的多对多关系，可以被拆解成两个一对多关系来实现。因此，我们使用的sql语句同样要通过连接表查询。

因此，只用**分别在Role和UserDao接口的映射文件中配置resultMap并使用一对多的<collection>标签，就可以完成多对多查询**，代码如下：

<!--定义user表的ResultMap-->

<resultMap id="userMap" type="user">

<id property="id" column="id"></id>

<result property="username" column="username"></result>

<result property="address" column="address"></result>

<result property="sex" column="sex"></result>

<result property="birthday" column="birthday"></result>

<collection property="roles" ofType="role">

<id property="roleId" column="rid"></id>

<result property="roleName" column="role\_name"></result>

<result property="roleDesc" column="role\_desc"></result>

</collection>

</resultMap>

<!--定义role表的ResultMap-->

<resultMap id="roleMap" type="role">

<id property="roleId" column="rid"></id>

<result property="roleName" column="role\_name"></result>

<result property="roleDesc" column="role\_desc"></result>

<collection property="users" ofType="user">

<id column="id" property="id"></id>

<result column="username" property="username"></result>

<result column="address" property="address"></result>

<result column="sex" property="sex"></result>

<result column="birthday" property="birthday"></result>

</collection>

</resultMap>

4.4 小结

在这部分中，学习了mybatis关于sql的一些复杂的动态查询，一对一、一对多与多对多等等查询，最基本的方法便是在实体类中把需要关联查询的类作为一个属性，并在配置文件中配置resultMap进行结果映射，最后使用多表查询语句来关联查询，得到需要的结果。

## 5、总结

总的来说，这部分学习内容简单的了解了Mybatis的连接池原理和Mybatis的事务控制机制，同时学了几个mybatis映射文件中的条件标签，这部分内容中最重要的就是mybatis的多表查询，从数据库中查询出与实体类有关联的其他实体类，一对一、一对多、或多对多的关系都可以使用相同的做法，即配置resultMap。