第二天:mybatis(下)

备课内容

1. MyBatis 的多表联合查询 (了解一对一,细讲一对多(做延迟加载的铺垫),模拟多对多)
2. MyBatis 的动态 SQL
3. MyBatis 中 #{} ${} 区别
4. MyBatis 的延迟加载 (什么是延迟加载,好处,场景)
5. MyBatis 的缓存机制 (缓存机制策略,一级缓存和二级缓存的不同,二级缓存的实现)
6. MyBatis 逆向工程 (了解逆向工程,说明不方便之处)

---

1. MyBatis 的多表联合查询

> 1对1的说明:在级联关系中oneToone是比较不太频繁出现的一种情况(在数据库设计上面可以考虑共用主键关系，或者利用1对多进行变种实现,然后利用外键可设置唯一约束**UNIQUE**约束),回顾mybatis本身的本质(专注sql,追求高度自由化),所以虽然mybatis提供一对一的关联，但是我们还是利用1对多的方式进行变种实现

** 引用:级联不是必须的，级联的好处是获取关联数据十分便捷，但是级联过多会增加系统的复杂度，同时降低系统的性能，此增彼减，所以当级联的层级超过3层时，就不要考虑使用级联了，因为这样会造成多个对象的关联，导致系统的耦合、复杂和难以维护。在现实的使用过程中，要根据实际情况判断是否需要使用级联。 **

> 1对多的实现方式 与 多对1 实现方式: (场景:1个用户有多个贴子,多个帖子的著作人)

- 表和实体关系说明: user表和 post 表 为1:M的关系

- 实体代码如下: 自行get和set

```
public class User {

    private int uid;
    private String uname;
    private List<Post> posts;

```

```
}
public class Post {
  private  int pid;
  private  int pname;
  }

```

- 接口和mapper映射内容操作如下

```
/**
 * 进行1对多的操作
 * @author Yun
 *
 */
public interface UserDao {

    /**
     * 获取用户的信息和所发的帖子
     * @return
     */
    List<User> getUsers();
}
//------------
<mapper namespace="com.wwj.dao.UserDao">

    <!-- 定义一对多的resultmap -->

    <resultMap type="com.wwj.model.User" id="users">
        <id property="uid" column="uid" />
        <result column="uname" property="uname" />
        <collection property="posts" ofType="com.wwj.model.Post">
            <id property="pid" column="pid" />
            <result column="pname" property="pname" />
        </collection>
    </resultMap>

    <select id="getUsers" resultMap="users">
        select u.*,p.*
        from user u,post p
        where u.uid = p.uid
    </select>
```

> 接下来是多对一的实现 也就是在站在帖子这一边,我在拿出帖子的时候看到著作人是谁（注意:实体会发生一点小的变动,在post属性中添加user属性

```
public interface PostDao {

    /**
     * 获取所有帖子信息
     * @return
     */
```

```
    List<Post>  getPosts();
}
//-------
<mapper namespace="com.wwj.dao.PostDao">

    <!-- 定义一对多的resultmap -->

    <resultMap type="com.wwj.model.Post" id="posts">
        <id property="pid" column="pid" />
        <result column="pname" property="pname" />
        <association property="user" javaType="com.wwj.model.User">
            <id property="uid" column="uid" />
            <result column="uname" property="uname" />
        </association>
    </resultMap>

    <select id="getPosts" resultMap="posts">
        select u.*,p.*
        from user u,post p
        where u.uid = p.uid
    </select>


</mapper>

```

> 下面是多对多上面的操作 场景 (一个用户可以有多个兴趣，一个兴趣可能有多个人选择) (从这句话中就可以看出多对多，其实就是双向1对多和多对1的变种实现)

** 构建二个实体 分别是animal interest 和第三方表**

1. 创建一个第三方表的映射接口

```
    /**
 * 根据兴趣id查看有多少用户选择
 * @param iid
 * @return
 */
List<Animal> getAnimalByIid(int iid);

/**
 * 根据用户id查看有当前用户有哪些兴趣
 * @param aid
 * @return
 */
List<Interest> getInterestByAid(int aid);
//------
<mapper namespace="com.wwj.dao.Animal_InterestDao">
<resultMap type="com.wwj.model.Animal" id="animals">
    <id property="aid" column="aid" />
    <result column="aname" property="aname" />
</resultMap>
```

```
20
21 <resultMap type="com.wwj.model.Interest" id="interests">
22     <id property="iid" column="iid" />
23     <result column="iname" property="iname" />
24 </resultMap>
25
26 <select id="getAnimalByIid" parameterType="int" resultMap="animals">
27     select
28     a.*,ai.iid
29     from animal a,animal_interest ai
30     where a.aid = ai.aid
31     and ai.iid = #{iid}
32 </select>
33
34 <select id="getInterestByAid" parameterType="int" resultMap="interests">
35     select
36     i.*,ai.aid
37     from interest i,animal_interest ai
38     where i.iid = ai.iid
39     and ai.aid = #{aid}
40 </select>
```

```

2. 分别构建AnimalDao 和 InterestDao 以及映射文件

```
1   /**
2   * 获取用户信息
3   * @return
4   */
5   List<Animal> getAnimals();
6   //-------
7   <mapper namespace="com.wwj.dao.AnimalDao">
8
9   <!-- 定义一对多的resultmap -->
10
11   <resultMap type="com.wwj.model.Animal" id="animals1">
12   <id property="aid" column="aid" />
13   <result column="uname" property="uname" />
14   <collection property="interests" column="aid"
  select="com.wwj.dao.Animal_InterestDao.getInterestByAid">
15   </collection>
16   </resultMap>
17
18   <select id="getAnimals" resultMap="animals1" >
19   select a.*
20   from animal a
21   </select>
22
23
24 </mapper>
```

```
1 public interface InterestDao {
```
```

```
    /**
     * 获取兴趣信息
     * @return
     */
    List<Interest>  getInterests();
}
//------

    <!-- 定义一对多的resultmap -->
    <resultMap type="com.wwj.model.Interest" id="interests1">
        <id property="iid" column="iid" />
        <result column="uname" property="uname" />
        <collection property="animals"  column="iid"
select="com.wwj.dao.Animal_InterestDao.getAnimalByIid">
        </collection>
    </resultMap>

    <select id="getInterests" resultMap="interests1" >
        select i.*
        from interest i
    </select>
</mapper>
```

2.MyBatis 的动态 SQL (就是加入条件判断和循环,当然一旦加入这些就可以相互嵌套)

** 模拟一张用户表进行说明 以及代码说明**

```
public interface TestUserDao {

    /**
     *  依次为if /whereif  / set /(whenchoose)/foreach
     * @param msg
     * @return
     */
    TestUser getUser(Map msg);
    TestUser getUserUseWhere(Map msg);
    TestUser updateUserById(Map msg);
    TestUser selectUserByChoose(Map msg);
    List<TestUser> selectUserByListId(List ids);

}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
```

```
<mapper namespace="com.wwj.dao.UserDao">

    <!-- 定义一对多的resultmap -->

    <resultMap type="com.wwj.model.TestUser" id="tuser">
        <id property="tid" column="tid" />
        <result column="tname" property="tname" />
        <result column="tage" property="tage" />
    </resultMap>

    <select id="getUser" resultMap="tuser">
        select  * from testuser
        where
       <if test="tname != null">
          tname=#{tname}
        </if>
        <if test="tage != null">
          and tage=#{tage}
        </if>
    </select>

    <!-- where”标签会知道如果它包含的标签中有返回值的话，它就插入一个‘where’。此外，如果标签返回的内容是以AND 或OR 开头的，则它会剔除掉。 -->
    <select id="getUserUseWhere" resultMap="tuser">
        select  * from testuser
        <where>
       <if test="tname != null">
          tname=#{tname}
        </if>
        <if test="tage != null">
          and tage=#{tage}
        </if>
        </where>

        <!--   trim标记是一个格式化的标记，可以完成set或者是where标记的功能
        prefix：前缀
        prefixoverride：去掉第一个
        suffix：后缀
        suffixoverride：去掉最后一个
        <trim prefix="where" prefixOverrides="and | or">
            <if test="username != null">
              and username=#{username}
            </if>
            <if test="sex != null">
              and sex=#{sex}
            </if>
        </trim> -->
    </select>


    <update id="updateUserById"  parameterType="java.util.Map">
    update testuser
        <set>
            <if test="tname != null and tname != ''">
                tname = #{tname},
            </if>
```

```
            <if test="tage != null and tage != ''">
                tage = #{tage}
            </if>
        </set>

    where tid=#{tid}
</update>


<select id="selectUserByChoose"  resultMap="tuser" parameterType="java.util.Map">
      select * from testuser
      <where>
          <choose>
              <when test="tid !='' and tid != null">
                  tid=#{tid}
              </when>
              <when test="tname !='' and tname != null">
                  and tname=#{tname}
              </when>
              <otherwise>
                  and tage=#{tage}
              </otherwise>
          </choose>
      </where>
  </select>

  <select id="selectUserByListId" resultMap="tuser" parameterType="java.util.List">
    select * from testuser
    <where>
        <!--
            collection:指定输入对象中的集合属性
            item:每次遍历生成的对象
            open:开始遍历时的拼接字符串
            close:结束时拼接的字符串
            separator:遍历对象之间需要拼接的字符串
            select * from user where 1=1 and (id=1 or id=2 or id=3)
         -->
        <foreach collection="list" item="tid" open="and (" close=")" separator="or">
            tid=#{tid}
        </foreach>
    </where>
</select>


</mapper>
```

3.MyBatis 中 #{} ${} 区别 （记住）

** 简单的说#{}是采用占位符的方式，而$是采用是字符串拼接的方式 **

** 谈下预编译 **

*** 字符串拼接的方式就一定会存在sql注入的问题 比如人为的在后缀上加上条件 or 1=1 满足条件成立 ***

---

4. MyBatis 的延迟加载 （延迟加载===按需加载)(如果提及到后续的服务上面,那么叫做按需调用服务)

1. ** 比如刚才的一个用户有多个兴趣,如果参照第一天的示例,现在需要用户信息的时候,编写sql语句的时候同时也把暂时不需要看用户的兴趣的数据加载出来,在数据量大的情况就肯定有瓶颈 **
2. ** 所以我们可以参照多对多示例中进行设置延迟加载,来观察sql语句的发出**

```
1
```

//代码示例如下: 我仅仅需要用户信息,但是同样的也把其它非相关的信息加载出来了
List as =session.selectList("getAnimals");
System.out.println(as.get(0).getAname());
//---------
DEBUG [main] - ==> Preparing: select a.* from animal a
DEBUG [main] - ==> Parameters:
DEBUG [main] - ====> Preparing: select i.*,ai.aid from interest i,animal_interest ai where i.iid = ai.iid and ai.aid = ?

```
1 3. ** 配置延迟加载 **
```

全局配置文件中配置

//------
DEBUG [main] - ==> Preparing: select a.* from animal a
DEBUG [main] - ==> Parameters:
DEBUG [main] - <== Total: 2
用户1
DEBUG [main] - ==> Preparing: select i.*,ai.aid from interest i,animal_interest ai where i.iid = ai.iid and ai.aid = ?
DEBUG [main] - ==> Parameters: 1(Integer)
DEBUG [main] - <== Total: 3
唱歌

```
1
```

5. MyBatis 的缓存机制

1. 一级缓存 (缓存不相互共享,在同一个事务中,如果存在同样的操作,中间不带增删改操作的话。那么不在进行二次IO读取操作)

```
        List<Animal>  as =session.selectList("getAnimals");
        System.out.println(as.get(0).getAname());
        List<Animal>  ass =session.selectList("getAnimals");
        System.out.println(as.get(0).getAname());
        //------
DEBUG [main] - ==>  Preparing: select a.* from animal a
DEBUG [main] - ==> Parameters:
DEBUG [main] - <==      Total: 2
用户1
用户1
```

2. 二级缓存 (缓存共享)需要映射的接口对应的映射文件加入

```
<mapper namespace="com.wwj.dao.AnimalDao">

 <cache/>
```

```
        System.out.println("----------------------------分割线利于观察");
    //关闭了之后数据会放入缓存中
    //测试二级缓存
    SqlSession session1 = build.openSession();
    List<Animal>  asss =session1.selectList("getAnimals");
    System.out.println(as.get(0).getAname());
    // 提交事务
    session1.commit();
    // 关闭 session
    session1.close();
    //-----
DEBUG [main] - <==      Total: 2
用户1
DEBUG [main] - Cache Hit Ratio [com.wwj.dao.AnimalDao]: 0.0
用户1
DEBUG [main] - Resetting autocommit to true on JDBC Connection
[com.mysql.jdbc.Connection@8c03696]
DEBUG [main] - Closing JDBC Connection [com.mysql.jdbc.Connection@8c03696]
DEBUG [main] - Returned connection 146814614 to pool.
----------------------------分割线利于观察
DEBUG [main] - Cache Hit Ratio [com.wwj.dao.AnimalDao]: 0.3333333333333333
用户1
```

补充(缓存机制策略)

1. 默认mybatis映射语句文件中所有的select语句将会被缓存
   映射语句文件中所有的insert update delete 语句会刷新缓存
   缓存会使用(Least Flush Interval,LRU最近最少使用的)算法来收回
   根据时间表（如 no Flush Interval,没有刷新间隔），缓存不会以任何时间顺序来刷新
   缓存会存储集合或对象（无论查询方法返回什么类型的值）的1024个引用
   缓存会被视为read/wriete(可读/可写)的，意味着对象检索不是共享的，而且可以安全的被调用者修改，而不干扰其他调用者或者线程所做的潜在修改
2. eviction(收回策略===更新策略)
   LRU 最近最少使用的，移除最长时间不被使用的对象，这是默认值
   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，更积极的移除基于垃圾收集器状态和弱引用规则的对象
3. 缓存存在数据可能出现脏读的现象 (操作同一数据）缓存数据尽量用在变更不频繁的数据上面
   影响缓存的三个因素（命中率、缓存更新策略、缓存最大数据量）

6. MyBatis 逆向工程 (了解逆向工程,说明不方便之处)

1. 逆向工程我们做简单的展示
2. 先说明一下，逆向工程的含义在于根据数据库的表的结构以面向对象的方式自动的帮助我们生成对应的实体类
3. 逆向为什么不太常用，因为一旦数据库的结构和关联发生变化，那么实际开发过程中就需要自己手动调整对应的实体类,这个可以说是一个非常浩瀚的工程,无论从人力和无力成本看来都得不偿失。
4. 以前我们可以说小的项目,用逆向比较方便，不如我们把逆向看成是不可取的，mybatis本身也是追求语句的自由化。所以逆向作为了解即可

- 引入对应的jar包
- 构建逆向配置xml文件 generatorConfig.xml 位置在项目外

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE generatorConfiguration
  PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
  "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">
<!-- targetRuntime="MyBatis3"可以生成带条件的增删改查，targetRuntime="MyBatis3Simple"可以生成基本的增删改查 -->
<generatorConfiguration>
    <context id="testTables" targetRuntime="MyBatis3">

        <commentGenerator>
            <!-- 是否去除自动生成的注释 true：是 ： false:否 -->
            <property name="suppressAllComments" value="true" />
        </commentGenerator>

        <!--数据库连接的信息：驱动类、连接地址、用户名、密码 -->
        <jdbcConnection driverClass="com.mysql.jdbc.Driver"
            connectionURL="jdbc:mysql://localhost:3306/mybatis" userId="root"
            password="root">
```

```
        </jdbcConnection>

        <!-- 默认false，把JDBC DECIMAL 和 NUMERIC 类型解析为 Integer，为 true时把JDBC DECIMAL 和
            NUMERIC 类型解析为java.math.BigDecimal -->
        <javaTypeResolver>
            <property name="forceBigDecimals" value="false" />
        </javaTypeResolver>


        <!-- targetProject:生成PO类的位置 -->
        <javaModelGenerator targetPackage="com.wwj.model1"
            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>

        <!-- targetProject:mapper映射文件生成的位置 -->
        <sqlMapGenerator targetPackage="com.wwj.dao1"
            targetProject="./src">
            <!-- enableSubPackages:是否让schema作为包的后缀 -->
            <property name="enableSubPackages" value="false" />
        </sqlMapGenerator>


        <!-- targetPackage：mapper接口生成的位置 -->
        <javaClientGenerator type="XMLMAPPER"
            targetPackage="com.wwj.dao1"
            targetProject="./src">
            <!-- enableSubPackages:是否让schema作为包的后缀 -->
            <property name="enableSubPackages" value="false" />
        </javaClientGenerator>


        <!-- 指定数据库表 -->
<table schema="mybatis" tableName="person" domainObjectName="personG"></table>

    </context>
</generatorConfiguration>
```

- 构建生成main函数

```
public class TestG {
  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);
```

```
13      myBatisGenerator.generate(null);
14
15  }
16  public static void main(String[] args) throws Exception {
17      try {
18            TestG generatorSqlmap = new TestG();
19          generatorSqlmap.generator();
20      } catch (Exception e) {
21          e.printStackTrace();
22      }
23
24  }
```