1. MyBatis多表查询
2. 一对一查询
3. 数据表的关系

用户表和订单表的关系为，一个用户有多个订单，但是一个订单只从属于一个用户。因此从订单的角度来看，订单与用户的关系是一对一的关系。

一对一查询的需求：查询一个订单，与此同时查询出该订单所属的用户。

数据库中表和表的关系是通过主外键进行关联的，如一对一的关系中，要么是两个表拥有相同的主键；要么是一个表用外键，这个外键是另一个表的主键。

1. 用户表和订单表的实例
2. 用户表和订单表的设计

用户表：

id：用户的id

username：用户名

password:密码

订单表：

id：订单的id，主键

ordertime:订单时间

total:金额

uid:外键，对应用户表的主键id

1. Java中实现两表的连接

数据库中订单表直接使用uid作为外键，对应用户表的主键id，以此来连接两个表。但是在Java中，订单对象Order中若只是存储uid这些数字是没法连接用户对象User的。我们需要在Order对象中引用User对象，即把User对象作为Order对象的属性，这样才能连接两个对象。

1. 配置映射文件

数据库中可以直接用SQL语句同时查询两个表，而Java中可是不能直接查询的，一切都是与对象联系起来的，若查询出来的数据对应的对象进行封装，是无法使用的。因此，我们需要配置对象的关系，将两个对象联系起来。

由于SQL语句写在映射文件中，因此我们需要在映射文件中进行配置。如两表同时查询出来的数据是无法用Order对象进行封装的，需要对Order对象进行配置。

这里使用的是一个新的标签resultMap，用于手动指定数据表的字段和实体对象的属性的映射关系。id属性是名称，type属性是实体对象的全限定名。

<resultMap id=”orderMap” type=”order”>

在其中可以添加子标签id，用于映射数据库的主键和实体对象属性的关系，

如：Order表的主键oid字段与Order对象的id属性对应

<id column=”oid” property=”id”></id>

如：其他的数据表普通字段使用result标签

<result column=”ordertime” property=”ordertime”></result>

而到了Order对象中的User对象时，我们要怎么映射呢？

同样是将数据表的字段映射到实体对象的属性中，只不过这次要映射到User对象的属性中了。

如，将数据表中的username字段映射到User对象的username属性中，注意property属性的写法，user.username代表着Order对象中的user属性对象中的username属性：

<result column=”username” property=”user.username”></result>

整体的resultMap如下所示：

<resultMap id=”orderMap” type=”order”>

<id column=”oid” property=”id”></id>

<result column=”ordertime” property=”ordertime”></result>

<result column=”total” property=”total”></result>

<result column=”uid” property=”user,id”></result>

<result column=”username” property=”user.username”></result>

<result column=”password” property=”user.password”></result>

</resultMap>

最后将配置好的resultMap放到sql语句的标签属性resultMap中：

<select id=”findAll” resultMap=”orderMap”>

SELECT \*,o.id oid FROM order o, user u, where o.uid=u.id

</select>

1. 配置User对象的优化

我们可以优化User对象的配置，用association标签，property属性表示要配置的User对象在Order实体中的属性名称，javaType属性表示该user属性的全限定名。

如：

<association property=”user” javaType=”user”>

其中可以添加子标签id,result，方法与上面差不多，唯一的差别是property属性中不需要再添加user了，直接用User对象中的属性名即可：

<association property=”user” javaType=”user”>

<id column=”uid” property=”id”></id>

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

<result column=”password” property=”password”></result>

</association>

1. 一对多查询的模型
2. 一对多关系

从用户的角度来看，一个用户可以有多个订单。因此一对多查询的需求：查询一个用户，于此同时查询出该用户具有的订单。

1. 数据表的设计

用户表：

id:用户id，主键

username:用户名

password:密码

订单表：

id:订单id，主键

ordertime:订单时间

total:金额

uid:订单所属用户id，外键

1. Java中实现一对多

一个用户可以有多个订单，数据表中可以直接用订单表的uid外键与用户表相关联，但是Java中使用对象来链接的，因此需要在User对象中引用Order对象，将Order对象的集合作为User对象的一个属性使用，即可表示一个User对象可以有多个Order对象：

private List<Order> orderList;

1. 配置映射文件

同理，我们通过查询User用户表以及其中引用的Order订单表，是无法直接封装到任何一个对象上的，需要手动配置映射文件，将数据表中的每一项和User对象，Order对象的属性分别对应起来。

设置resultMap标签，配置User对象，方法都是相同的。

<resultMap id=”userMap” type=”user”>

<id column=”uid” property=”id”></id>

……

只不过User对象中有Order对象的集合属性，需要使用collection标签来配置。property属性为User对象中的Order对象的集合属性名称，ofType属性为集合中的对象类型的全限定名。其中可以添加子标签，表示Order对象的属性。

<collection property=”orderList” ofType=”order”>

<id column=”oid” property=”id”></id>

……

1. 多对多查询
2. 多对多查询的模型

用户表和角色表的关系：一个用户有多个角色，一个角色被多个用户使用，这就是多对多的关系。现在的需求是：查询用户的同时查询出该用户的所有角色。多对多的查询需要使用中间表，存储两个表的主键的映射关系，如要查询用户，则查询用户表的主键ID对应的角色表的ID即可。

1. 数据表的设计

用户表：

id:用户id，主键

username:用户名

password:密码

角色表：

id:角色id，主键

roleName:角色名称

roleDesc:角色描述

中间表：

userId:用户表的主键id

roleId:角色表的主键id

1. Java中实现多对多

一个用户可以有多个角色，我们要通过查询用户来查询用户有多少个角色，因此，在用户User对象中设置Role对象集合属性，表示该用户拥有的角色集合。

private List<Role> roleList;

1. 配置映射文件

在UserMapper接口添加了接口方法findUserAndRoleAll()方法后，需要配置这个方法的映射文件。其实，多对多查询在使用中单个查询，本质上还是一对多查询，只是多了个中间表而已，SQL语句不太一样，SQL语句中通过中间表将两个表映射起来。

<resultMap id=”userRoleMap” type=”user”>

<id column=”userId” property=”id”></id>

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

<result column=”password” property=”password”></reseult>

<collection property=”roleList” ofType=”role”>

<id column=”roleId” property=”id”></id>

<result column=”roleName” property=”roleName”></result>

<result column=”roleDesc” property=”roleDesc”></result>

</collection>

</resultMap>

<select id=”findUserAndRoleAll” resultMap=”userRoleMap”>

select \* from user u, sys\_user\_role ur, sys\_role r where u.id=ur.userId and ur.roleId=r.id

</select>

1. MyBatis的注解开发
2. MyBatis的常用注解

MyBatis可以使用注解开发的方式，这样可以减少编写Mapper映射文件了，直接在接口的方法上进行注解开发。下面是一些常用的注解：

1. @Insert

实现插入数据

1. @Update

实现更新数据

1. @Delete

实现删除数据

1. @Select

实现查询数据

1. @Result

实现结果集的封装

1. @Results

可以和@Result一起使用，封装多个结果集

1. @One

实现一对一的结果集封装

1. @Many

实现一对多的结果集封装

1. 注解完成基本的CRUD
2. 接口方法添加注解

注解是用于替代Mapper映射文件的，原有的Mapper映射文件中的SQL语句需要放到注解中，同时还有手动配置的数据库表的字段和实体对象的属性的映射关系，都要放到注解中。

如：

插入数据：

@Insert(“insert into user values(#{id}, #{username}, #{password})”)

public void save(User user);

可以看到，这里直接把SQL语句原封不动地写过来是可行的，因为接口方法的传入参数即是原来的映射文件中的parameterType，而返回类型即是resultType，因此，可以根据传入参数使用#{id}这种方式。

1. 核心配置文件中加载映射关系

由于没有了Mapper映射文件，核心配置文件中的加载映射文件也要删除，替换成加载映射关系。同样是使用mappers标签，添加子标签package，name属性为借口方法所在的包的全限定名。

<mappers>

<package name=”com.mybatis.mapper”></package>

</mappers>

1. 注解实现复杂映射开发

上面通过注解实现了简单的CRUD操作，那么下面我们要通过注解实现复杂的映射。

1. 常用的注解

（1）@Results

代替标签<resultMap>，该注解中可以使用单个@Result注解，也可以使用@Result集合。使用的格式：@Results({@Result(), @Result()})或者@Results(@Result())

（2）@Result

代替了<id>和<result>标签。@Result中的属性为：

column:数据库的字段名

property:实体对象的属性名

one:需要使用的@One注解，用于一对一查询(@Result(one=@One)())

many:需要使用到@Many注解，用于一对多查询（@Result(many=@many)()）

（3）@One

代替了<association>标签，用于指定子查询返回单一对象，用于一对一查询。

@One中的属性：

select:指定用来查询的接口方法的全限定名

使用格式：@Result(column=””, property=””, one=@One(select=””))

（4）@Many

代替了<collection>标签，用于指定子查询返回对象集合。

使用格式：@Result(property=””, column=””, many=@Many(select=””))

1. 注解实现一对一查询
2. 需求

查询订单对应的唯一用户。

1. 实现

因此，首先要在OrderMapper的接口中添加查询方法findAll()，继而添加注解@Select:

@Select(“select \*,o.id oid from orders o,user u where o.uid=u.id”)

因为是两个表的数据，需要手动映射查询出的数据表字段和实体对象的属性的映射关系。使用@Results注解，在其中添加@Result注解：

@Results({

@Result(column=”oid”, property=”id”),

@Result(column=”ordertime”, property=”ordertime”),

@Result(column=”total”, property=”total”),

@Result(column=”uid”, property=”user.id”),

@Result(column=”username”, property=”user.username”),

@Result(column=”password”, property=”user.password”)

})

1. 优化

我们可以将同时查询两个表分为一次查询一个表，第一次查询Order表，第二次将查询出的uid用来查询User表。

因此，OrderMapper接口中的findAll方法的注解就要修改了。

首先，第一个SQL语句为”select \* from orders”，放在@Select中：

@Select(“select \* from orders”)

Order中的普通属性映射不变，需要变化的是Order属性中的User属性，我们需要用@One注解。

同样是@Result注解，

property属性为要封装的User对象在Order对象中的属性名称”user”，

javaType属性为要封装的实体对象字节码User.class，

column属性为我们要根据第一个SQL语句查询出的哪个字段名来进行第二个SQL语句的查询，这里是Order表中的uid字段。

one属性使用@One注解，select属性表示着第二个SQL语句的接口方法的全限定名。

@Result(

property=”user”,

column=”uid”,

javaType=User.class,

one=@One(select=”com.mybatis.mapper.UserMapper.findById”)

)

第二个SQL语句对应的方法为：

@Select(“select \* from user where id=#{id}”)

public User findById(int id);

id是将第一个SQL语句中查询出的Order表的uid字段作为参数传入的，SQL语句中也可以直接使用该id。

1. 注解实现一对多查询
2. 需求

一个用户有多个订单，一个订单只从属于一个用户。因此查询一个用户，同时查询出该用户的所有订单。

1. 实现

一个用户有多个订单，因此在User对象中添加Order对象集合属性orderList:

private List<Order> orderList;

接着，在UserMapper接口中添加方法findUserAndOrderAll()，并添加注解。这里直接使用上面优化的方法，将SQL语句分为两次执行。第一次查询User表：

@Select(“select \* from user”)

public List<User> findUserAndOrderAll();

那么，查询了User表后，我们需要将uid值传入第二个SQL语句中。

首先，第二个SQL语句是要查询Order表，因此在OrderMapper 接口中添加方法findByUid，传入参数uid：

@Select(“select \* from orders where uid=#{uid}”)

public List<Order> findByUid(int uid)

接着，就可以配置UserMapper接口中的findUserAndOrderAll方法的注解了，大体的方法和一对一查询差不多，只不过将one属性和@One注解换成了many和@Many注解，同时javaType属性为List.class。

@Result(

property=”orderList”,

column=”id”,

javaType=List.class,

many=@Many(select=”com.mybatis.mapper.OrderMapper.findByUid”)

)

1. 注解实现多对多查询
2. 需求

一个用户有多个角色，一个角色被多个用户使用。查询用户的同时查询出该用户的所有角色。

1. 实现

首先，在User对象中添加Role对象集合属性：  
private List<Role> roleList;

接下来其实和一对多查询差不多，只是多了一个中间表。

第一次SQL查询，查询User表，在UserMapper中的方法findUserAndRoleAll上添加注解：

@Select(“select \* from user)

public List<User> findUserAndRoleAll();

接着是第二次SQL查询，查询Role表和中间表，根据第一次SQL查询出的uid进行第二次查询：

@Select(“select \* from sys\_user\_role ur, sys\_role r where ur.roleId=r.id and ur.userId=#{uid}”)

public List<Role> findByUid(int id);

继而，就可以在第一次SQL查询中添加映射注解@Results了，方法同一对多查询：

@Select(“select \* from user”)

@Results({

@Result(column=”id”, property=”id”),

@Result(column=”username”, property=”username”),

@Result(column=”password”, property=”password”),

@Result(

property=”roleList”,

column=”id”,

javaType=List.class,

many=@Many(select=”com.mybatis.mapper.RoleMapper.findByUid”)

)

})

public List<User> findUserAndRoleAll();