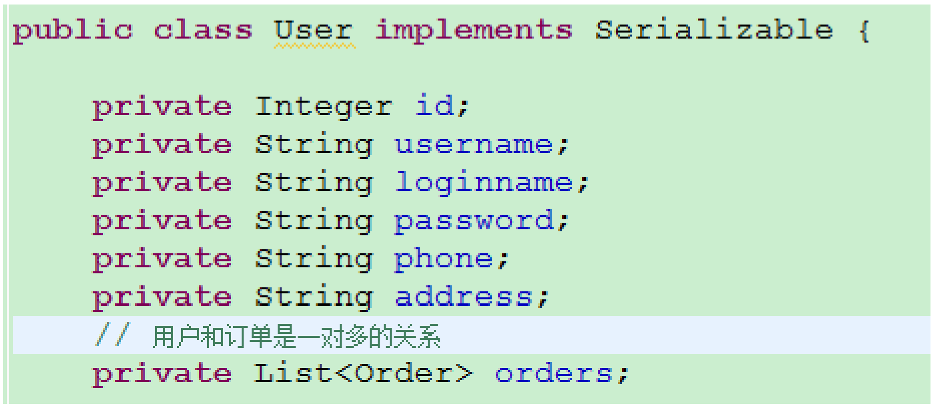
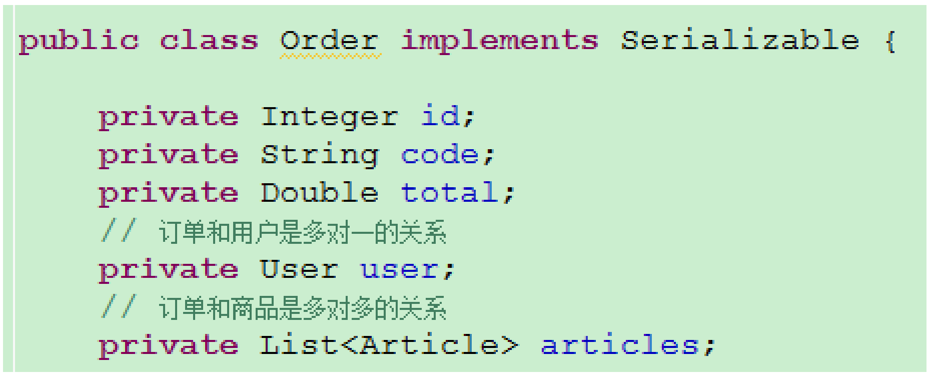
多对多关系中，association和collection就要更会灵活运用才行

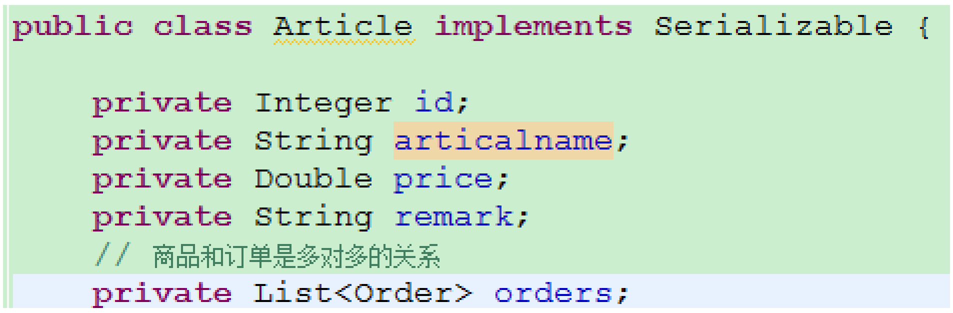
三个持久化类



```
public class User implements Serializable {

    private Integer id;
    private String username;
    private String loginname;
    private String password;
    private String phone;
    private String address;
    // 用户和订单是一对多的关系
    private List<Order> orders;
```



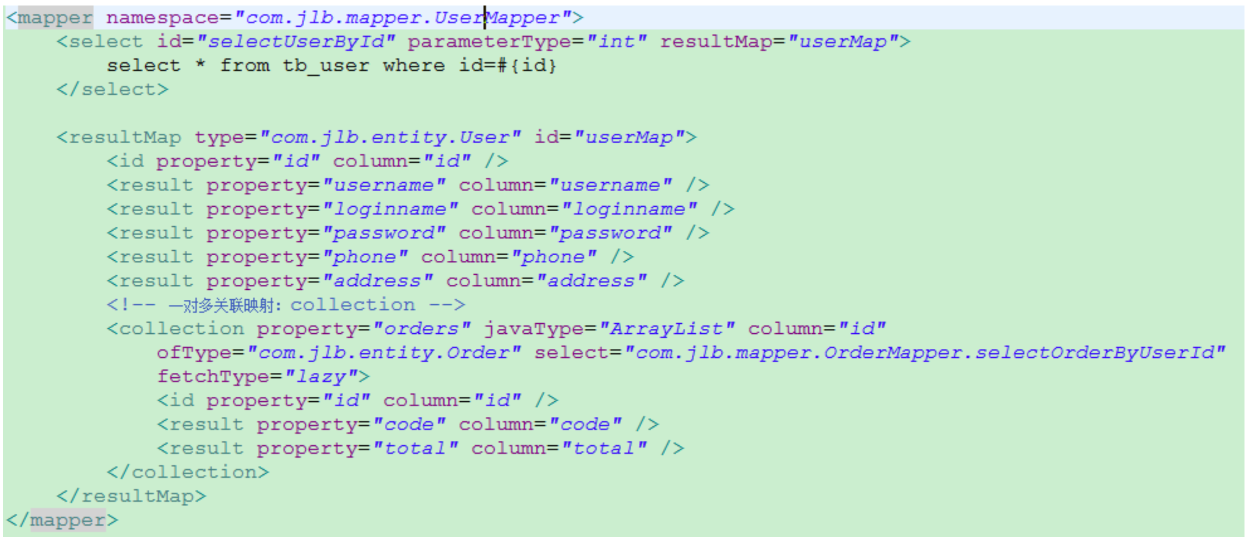
```
public class Order implements Serializable {

    private Integer id;
    private String code;
    private Double total;
    // 订单和用户是多对一的关系
    private User user;
    // 订单和商品是多对多的关系
    private List<Article> articles;
```



```
public class Article implements Serializable {

    private Integer id;
    private String articalname;
    private Double price;
    private String remark;
    // 商品和订单是多对多的关系
    private List<Order> orders;
```

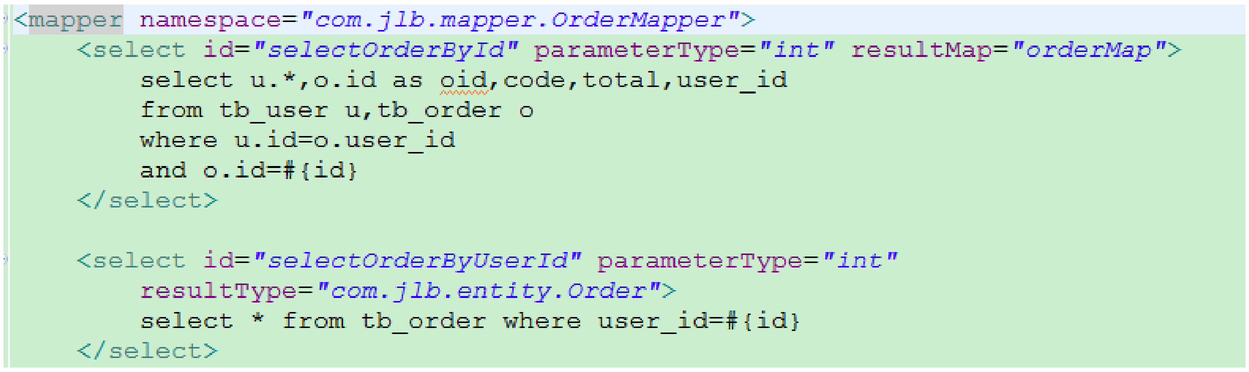
各种映射关系就如图中注释所说

对应的三个mapper

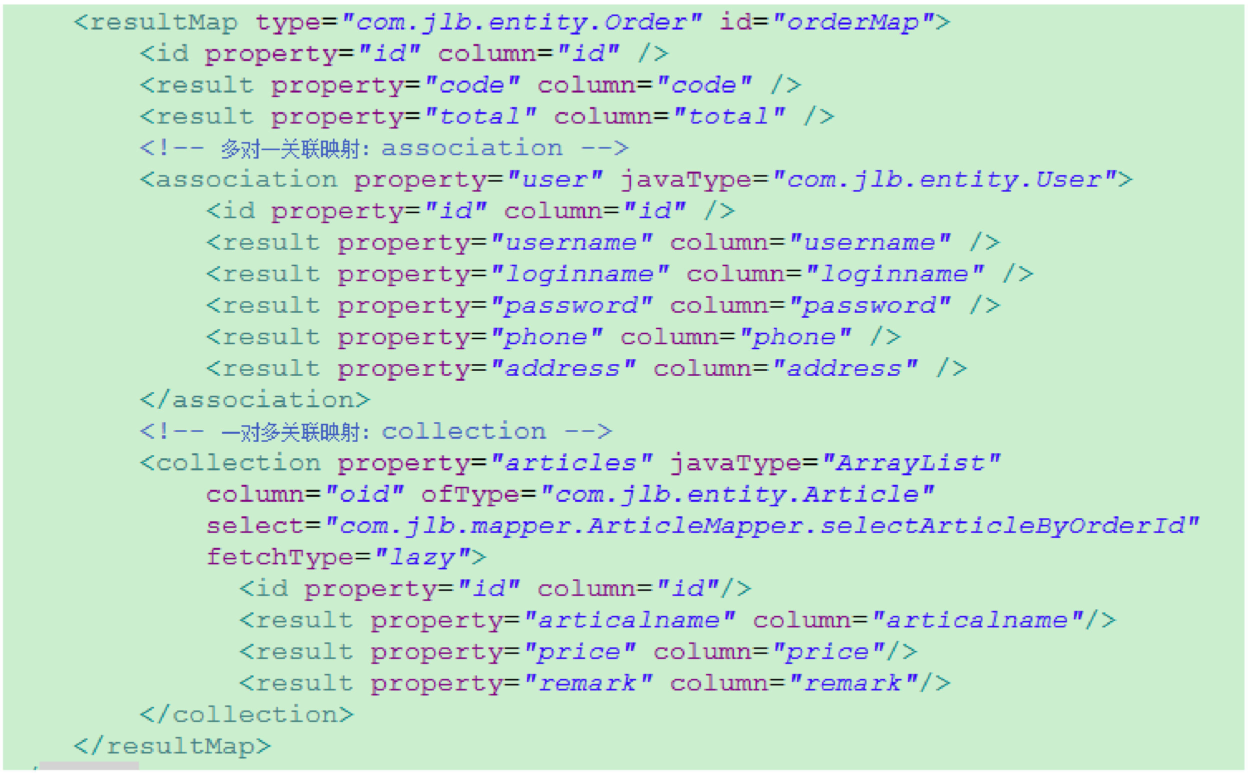


```xml
<mapper namespace="com.jlb.mapper.UserMapper">
    <select id="selectUserById" parameterType="int" resultMap="userMap">
        select * from tb_user where id=#{id}
    </select>

    <resultMap type="com.jlb.entity.User" id="userMap">
        <id property="id" column="id" />
        <result property="username" column="username" />
        <result property="loginname" column="loginname" />
        <result property="password" column="password" />
        <result property="phone" column="phone" />
        <result property="address" column="address" />
        <!-- 一对多关联映射: collection -->
        <collection property="orders" javaType="ArrayList" column="id"
            ofType="com.jlb.entity.Order" select="com.jlb.mapper.OrderMapper.selectOrderByUserId"
            fetchType="lazy">
            <id property="id" column="id" />
            <result property="code" column="code" />
            <result property="total" column="total" />
        </collection>
    </resultMap>
</mapper>
```

慢慢道来，UserMapper有一个操作，根据user的id查询user的信息，当然返回的map级联查询了该用户的订单信息，执行的是根据user的id查询所有的订单，下面会介绍，这边实现的业务就是用户和订单是一对多的关联关系，根据实际需求，一个用户是可以去获取ta的所有订单的，所以会有这个collection关联映射。

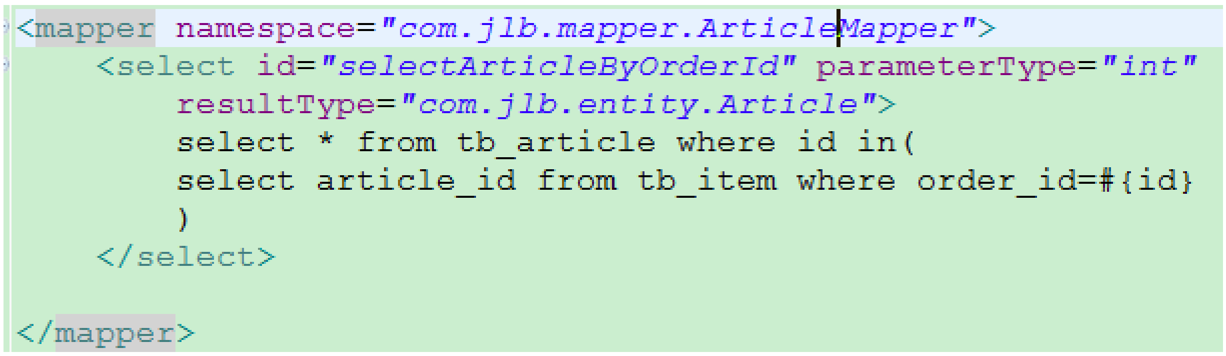


```xml
<mapper namespace="com.jlb.mapper.OrderMapper">
    <select id="selectOrderById" parameterType="int" resultMap="orderMap">
        select u.*,o.id as oid,code,total,user_id
        from tb_user u,tb_order o
        where u.id=o.user_id
        and o.id=#{id}
    </select>

    <select id="selectOrderByUserId" parameterType="int"
        resultType="com.jlb.entity.Order">
        select * from tb_order where user_id=#{id}
    </select>
```



```xml
    <resultMap type="com.jlb.entity.Order" id="orderMap">
        <id property="id" column="id" />
        <result property="code" column="code" />
        <result property="total" column="total" />
        <!-- 多对一关联映射: association -->
        <association property="user" javaType="com.jlb.entity.User">
            <id property="id" column="id" />
            <result property="username" column="username" />
            <result property="loginname" column="loginname" />
            <result property="password" column="password" />
            <result property="phone" column="phone" />
            <result property="address" column="address" />
        </association>
        <!-- 一对多关联映射: collection -->
        <collection property="articles" javaType="ArrayList"
            column="oid" ofType="com.jlb.entity.Article"
            select="com.jlb.mapper.ArticleMapper.selectArticleByOrderId"
            fetchType="lazy">
              <id property="id" column="id"/>
              <result property="articalname" column="articalname"/>
              <result property="price" column="price"/>
              <result property="remark" column="remark"/>
        </collection>
    </resultMap>
```

OrderMapper则实现了两个操作，一个根据user的id查询所有的订单是为了UserMapper服务，还有个是根据一个order的id查询这个order的相信信息，这里用了多表查询，将订单信息个用户信息一起查询了，可以看到我们map的association中没有运用反向select，所以想要user的信息，只能使用多表查询直接查，因为使用了多表查询，所以肯定会有键名的冲突，我们给其中一个id设置了别名，对应的map中collection中的反向select的参数就是oid，这边实现的业务就是订单和用户多对一的关系以及订单和商品多对多的关系。



```xml
<mapper namespace="com.jlb.mapper.ArticleMapper">
    <select id="selectArticleByOrderId" parameterType="int"
        resultType="com.jlb.entity.Article">
        select * from tb_article where id in(
        select article_id from tb_item where order_id=#{id}
        )
    </select>

</mapper>
```

ArticleMapper只实现了一个操作，查询所有符合子查询条件的商品信息，而订单和商品多对多关系的中间表也在这里起作用了，OrderMapper那里运用反向select后相应订单的相应商品信息就传过去了，这里为什么不像OrderMapper一样也做个映射呢，根据实际需求，没有知道一个商品要去查有它的所有订单，所以没必要。

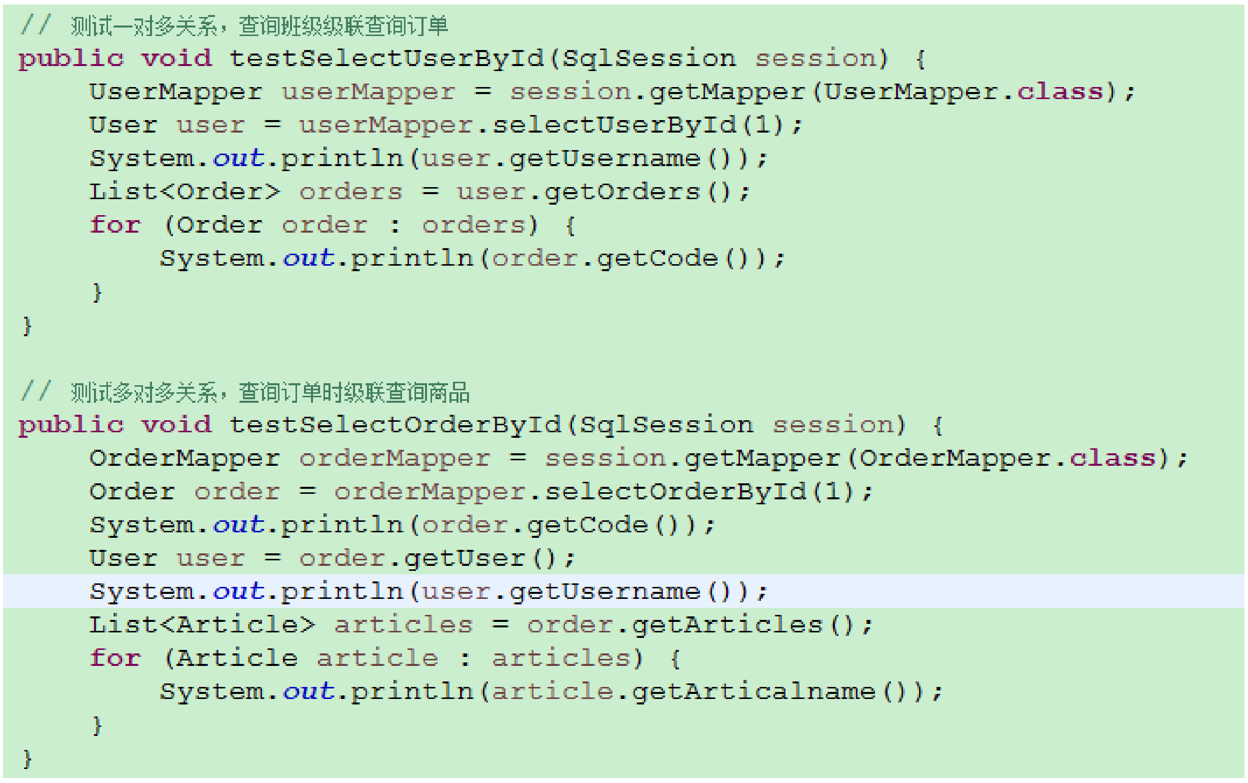
因为只有两个mapper有实际的操作需求，所以只定义相关的两个接口

```java
public interface UserMapper {

    User selectUserById(Integer id);
}
```

```java
public interface OrderMapper {

    Order selectOrderById(Integer id);
}
```

测试方法，对应两个接口两个方法的实现

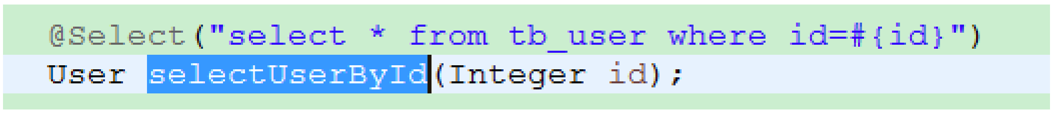
```java
    public static void main(String[] args) {
        // 获得Session实例
        SqlSession session = SelfSqlSessionFactory.getSqlSession();

        ManyToManyTest manyToManyTest = new ManyToManyTest();

//      manyToManyTest.testSelectUserById(session);
        manyToManyTest.testSelectOrderById(session);

        session.commit();
        session.close();

    }
```



```java
//  测试一对多关系，查询班级级联查询订单
public void testSelectUserById(SqlSession session) {
     UserMapper userMapper = session.getMapper(UserMapper.class);
     User user = userMapper.selectUserById(1);
     System.out.println(user.getUsername());
     List<Order> orders = user.getOrders();
     for (Order order : orders) {
          System.out.println(order.getCode());
     }
}

//  测试多对多关系，查询订单时级联查询商品
public void testSelectOrderById(SqlSession session) {
     OrderMapper orderMapper = session.getMapper(OrderMapper.class);
     Order order = orderMapper.selectOrderById(1);
     System.out.println(order.getCode());
     User user = order.getUser();
     System.out.println(user.getUsername());
     List<Article> articles = order.getArticles();
     for (Article article : articles) {
          System.out.println(article.getArticalname());
     }
}
```

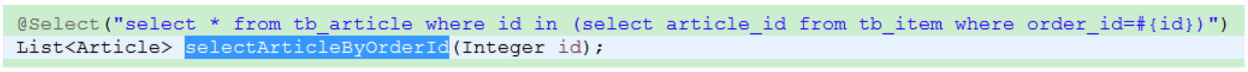
容我慢慢道来注解的使用

UserMapper



```java
@Select("select * from tb_user where id=#{id}")
User selectUserById(Integer id);
```

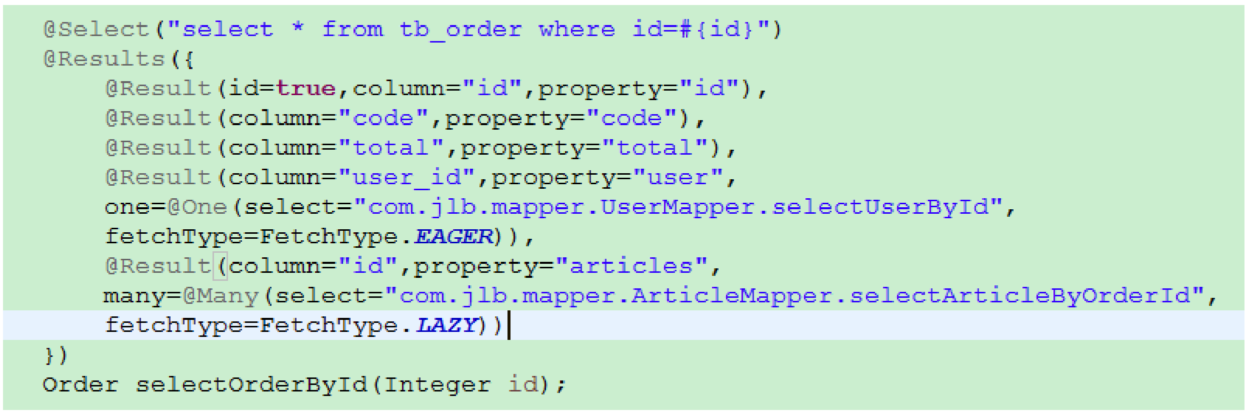
ArticleMapper



```java
@Select("select * from tb_article where id in (select article_id from tb_item where order_id=#{id})")
List<Article> selectArticleByOrderId(Integer id);
```

看着这两个这么孤单的样子，一看就是准备作为反向select使用（手动俏皮）

OrderMapper



```java
@Select("select * from tb_order where id=#{id}")
@Results({
    @Result(id=true,column="id",property="id"),
    @Result(column="code",property="code"),
    @Result(column="total",property="total"),
    @Result(column="user_id",property="user",
    one=@One(select="com.jlb.mapper.UserMapper.selectUserById",
    fetchType=FetchType.EAGER)),
    @Result(column="id",property="articles",
    many=@Many(select="com.jlb.mapper.ArticleMapper.selectArticleByOrderId",
    fetchType=FetchType.LAZY))
})
Order selectOrderById(Integer id);
```

正主来了，规模都能看出来，不仅处理了，不仅处理了和user的一对一（对于单个订单和用户就是一对一）关系，还处理了和article的一对多（对于单个订单和商品就是一对多）关系