mybatis-day02

第一章 数据输出

数据输出是针对查询数据的方法返回查询结果

第一节 返回单个简单类型数据

Mapper接口中的抽象方法：方法的返回值是简单数据类型

```
/**
     * 统计员工数量
     * @return
     */
Long selectEmployeeCount();
```

映射配置文件: 此时标签的resultType的类型对应抽象方法的返回值类型

```
<!--
        返回简单类型:
        resultType表示结果类型:结果集返回的类型,要和Mapper接口中对应的方法的返回值类型保持一
致
    -->
<select id="selectEmployeeCount" resultType="long">
    select count(emp_id) from t_emp
</select>
```

第二节 返回一条数据

1. 返回实体类对象

Mapper接口中的抽象方法：方法的返回值是POJO类型

```
Employee selectEmployee(Integer empId);
```

映射配置文件: 此时标签的resultType的类型对应抽象方法的返回值类型的全限定名

```
<!-- 编写具体的SQL语句，使用id属性唯一的标记一条SQL语句 -->
<!-- resultType属性：指定封装查询结果的Java实体类的全类名 -->
<select id="selectEmployee" resultType="com.atguigu.mybatis.entity.Employee">
    <!-- Mybatis负责把SQL语句中的#{}部分替换成“?”占位符 -->
    <!-- 给每一个字段设置一个别名，让别名和Java实体类中属性名一致 -->
    select emp_id empId,emp_name empName,emp_salary empSalary from t_emp where
emp_id=#{maomi}
</select>
```

通过给数据库表字段加别名，让查询结果的每一列都和Java实体类中属性对应起来。

增加全局配置自动映射驼峰命名规则

在Mybatis的核心配置文件中做下面的配置，select语句中可以不给字段设置别名

```
<!-- 在全局范围内对Mybatis进行配置 -->
<settings>
    <!-- 具体配置 -->
    <!-- 从org.apache.ibatis.session.Configuration类中可以查看能使用的配置项 -->
    <!-- 将mapUnderscoreToCamelCase属性配置为true，表示开启自动映射驼峰式命名规则 -->
    <!-- 规则要求数据库表字段命名方式：单词_单词 -->
    <!-- 规则要求Java实体类属性名命名方式：首字母小写的驼峰式命名 -->
    <setting name="mapUnderscoreToCamelCase" value="true"/>
</settings>
```

2. 返回Map类型

适用于SQL查询返回的各个字段综合起来并不和任何一个现有的实体类对应，没法封装到实体类对象中。**能够封装成实体类类型的，就不使用Map类型。**

Mapper接口中的抽象方法：方法的返回值是Map类型

```
/**
* 根据empId查询员工信息，并且将结果集封装到Map中
* @param empId
* @return
*/
Map selectEmployeeMapByEmpId(Integer empId);
```

映射配置文件: 此时标签的resultType的类型为`map`

```
<!--
        返回Map类型:
        resultType表示结果类型: 就是Map的全限定名或者别名
    -->
<select id="selectEmployeeMapByEmpId" resultType="java.util.Map">
    select * from t_emp where emp_id=#{empId}
</select>
```

第三节 返回多行数据

1. 返回List<POJO>

查询结果返回多个实体类对象，希望把多个实体类对象放在List集合中返回。此时不需要任何特殊处理，在resultType属性中还是设置实体类类型即可。

Mapper接口中的抽象方法：方法的返回值是List<POJO>

```
List<Employee> selectAll();
```

映射配置文件: 此时标签的resultType的类型为POJO类的全限定名

```
<!-- List<Employee> selectAll(); -->
<select id="selectAll" resultType="com.atguigu.mybatis.entity.Employee">
    select emp_id empId,emp_name empName,emp_salary empSalary
    from t_emp
</select>
```

2. 返回List<Map>

查询结果返回多个Map对象，希望把多个Map对象放在List集合中返回。此时不需要任何特殊处理，在resultType属性中还是设置`map`即可。

Mapper接口中的抽象方法：方法的返回值是List<Map>类型

```
List<Map> selectAllMap();
```

映射配置文件: 此时标签的resultType的类型为`map`

```
<select id="selectAllMap" resultType="map">
    select emp_id empId,emp_name empName,emp_salary empSalary
    from t_emp
</select>
```

第四节 返回自增主键

1. 使用场景

例如：保存订单信息。需要保存Order对象和List<OrderItem>。其中，OrderItem对应的数据库表，包含一个外键，指向Order对应表的主键。

在保存List<OrderItem>的时候，需要使用下面的SQL：

```
insert into t_order_item(item_name,item_price,item_count,order_id) values(...)
```

这里需要用到的order_id，是在保存Order对象时，数据库表以自增方式产生的，需要特殊办法拿到这个自增的主键值。

2. 实现方案

Mapper接口中的抽象方法：

```
int insertEmployee(Employee employee);
```

映射配置文件:

```
<!-- int insertEmployee(Employee employee); -->
<!-- useGeneratedKeys属性字面意思就是“使用生成的主键” -->
<!-- keyProperty属性可以指定主键在实体类对象中对应的属性名，Mybatis会将拿到的主键值存入这个属性 -->
<insert id="insertEmployee" useGeneratedKeys="true" keyProperty="empId">
    insert into t_emp(emp_name,emp_salary)
    values(#{empName},#{empSalary})
</insert>
```

junit测试代码:

```
@Test
public void testSaveEmp() {

    EmployeeMapper employeeMapper = session.getMapper(EmployeeMapper.class);

    Employee employee = new Employee();

    employee.setEmpName("john");
    employee.setEmpSalary(666.66);

    employeeMapper.insertEmployee(employee);

    System.out.println("employee.getEmpId() = " + employee.getEmpId());

}
```

注意:

Mybatis是将自增主键的值设置到实体类对象中，而**不是以Mapper接口方法返回值**的形式返回。

另一种做法

```
<insert id="insertEmployee">
    insert into t_emp (emp_name,emp_salary) values (#{empName},#{empSalary})
    <!--
            keyColumn="emp_id"表示要查询的主键的列名
            keyProperty="empId"表示将查询到的主键值赋给JavaBean的哪个属性
            resultType="int"表示查询的结果类型
            order="AFTER" 表示这个查询是执行在insert之前还是之后呢?如果为AFTER表示之后，
BEFORE表示之前
        -->
    <selectKey keyColumn="emp_id" keyProperty="empId" resultType="int"
order="AFTER">
        select last_insert_id()
    </selectKey>
</insert>
```

3. 不支持自增主键的数据库怎么获取主键值(以下代码仅供参考)

而对于不支持自增型主键的数据库（例如 Oracle），则可以使用 selectKey 子元素：selectKey元素将会首先运行，id 会被设置，然后插入语句会被调用

```
<insert id="insertEmployee"
        parameterType="com.atguigu.mybatis.beans.Employee"
            databaseId="oracle">
        <selectKey order="BEFORE" keyProperty="id"
                                        resultType="integer">
            select employee_seq.nextval from dual
        </selectKey>
        insert into orcl_employee(id,last_name,email,gender) values(#{id},#
{lastName},#{email},#{gender})
</insert>
```

或者:

```
<insert id="insertEmployee"
        parameterType="com.atguigu.mybatis.beans.Employee"
            databaseId="oracle">
        <selectKey order="AFTER" keyProperty="id"
                                        resultType="integer">
            select employee_seq.currval from dual
        </selectKey>
    insert into orcl_employee(id,last_name,email,gender)
values(employee_seq.nextval,#{lastName},#{email},#{gender})
</insert>
```

第五节 结果集的字段和实体类属性对应关系

1. 自动映射

Mybatis在做结果集与POJO类的映射关系的时候，会自动将结果集的字段名与POJO的属性名(其实是和getXXX方法)进行对应映射，结果集的数据会自动映射给POJO对象中同名的属性; 所以当我们遇到表的字段名和POJO属性名不一致的情况，我们可以在编写查询语句的时候，给结果集的字段取别名，让别名与POJO的属性名一致以保证结果集的正确映射

2. 全局配置自动识别驼峰式命名规则

因为我们表中字段的命名规则采用"_",而POJO的属性名命名规则采用驼峰命名法，所以导致我们在执行查询语句的时候总是要对查询的字段取别名，以确保正确地进行结果集映射

Mybatis框架当然也注意到了这个问题，所以它提供了一种自动识别驼峰命名规则的配置，我们只要做了该配置，那么全局的所有查询语句的执行都会自动识别驼峰命名规则

在Mybatis全局配置文件加入如下配置:

```
<!-- 使用settings对Mybatis全局进行设置 -->
<settings>
    <!-- 将xxx_xxx这样的列名自动映射到xxXxx这样驼峰式命名的属性名 -->
    <setting name="mapUnderscoreToCamelCase" value="true"/>
</settings>
```

SQL语句中可以不使用别名:

```
<!-- Employee selectEmployee(Integer empId); -->
<select id="selectEmployee" resultType="com.atguigu.mybatis.entity.Employee">
    select emp_id,emp_name,emp_salary from t_emp where emp_id=#{empId}
</select>
```

3. 手动映射

声明一个封装结果集的POJO

```
package com.atguigu.pojo;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * 包名:com.atguigu.pojo
```

```
 *
 * @author Leevi
 * 日期2021-08-25  09:18
 */
@Data
@AllArgsConstructor
@NoArgsConstructor
public class EmployeeInfo {
    private Integer id;
    private String name;
    private Double salary;
}
```

使用resultMap标签手动指定结果集字段与POJO属性的映射关系,可以非常灵活地进行结果集的映射

```
<!--
        手动映射:通过resultMap标签配置映射规则
            1. id属性:表示这个手动映射规则的唯一表示
            2. type属性: 表示这个手动映射规则是将结果集映射给哪个类的对象，就是JavaBean类的全限定名
        resultMap标签中的子标签就是一一指定映射规则:
            1. id标签:指定主键的映射规则
            2. result标签:指定非主键的映射规则
        id标签和result标签的属性:
            1. column:要进行映射的结果集的字段名
            2. property:要进行映射的JavaBean的属性名
    -->
<resultMap id="EmployeeInfoMap" type="com.atguigu.pojo.EmployeeInfo">
    <id column="emp_id" property="id"/>
    <result column="emp_name" property="name"/>
    <result column="emp_salary" property="salary"/>
</resultMap>


<!--
        在select标签中通过resultMap属性来指定使用哪个手动映射规则
    -->
<select id="selectEmployeeInfoByEmpId" resultMap="EmployeeInfoMap">
    select * from t_emp where emp_id=#{empId}
</select>
```

第二章 多表关联查询

第一节 物理建模

1. 建表和插入数据

```
CREATE TABLE `t_customer` (
     `customer_id` INT NOT NULL AUTO_INCREMENT,
     `customer_name` varchar(100),
     PRIMARY KEY (`customer_id`)
);
CREATE TABLE `t_order` (
    `order_id` INT NOT NULL AUTO_INCREMENT,
    `order_name` varchar(100),
    `customer_id` INT,
```

```
    PRIMARY KEY (`order_id`)
);
INSERT INTO `t_customer` (`customer_name`) VALUES ('c01');
INSERT INTO `t_customer` (`customer_name`) VALUES ('c02');
INSERT INTO `t_order` (`order_name`, `customer_id`) VALUES ('o1', '1');
INSERT INTO `t_order` (`order_name`, `customer_id`) VALUES ('o2', '1');
INSERT INTO `t_order` (`order_name`, `customer_id`) VALUES ('o3', '1');
INSERT INTO `t_order` (`order_name`, `customer_id`) VALUES ('o4', '2');
INSERT INTO `t_order` (`order_name`, `customer_id`) VALUES ('o5', '2');
```

2. 表关系分析

`t_customer`表和`t_order`表示**一对多**关系，反之`t_order`表和`t_customer`表可以看成**一对一或者多对一**关系

第二节 一对一或者多对一查询

1. 目标

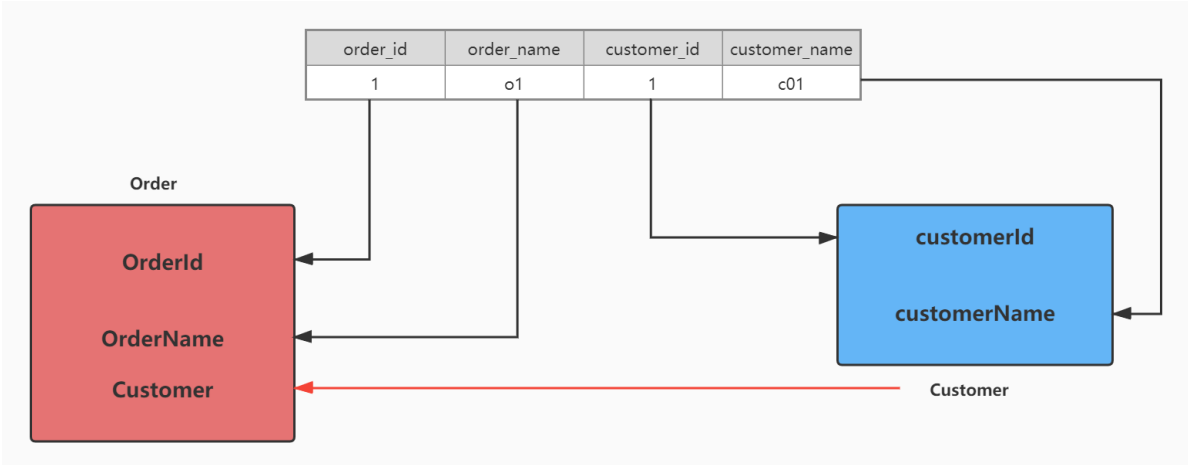
根据订单ID查询出订单信息，并且查询出该订单所属的顾客信息，将查询到的结果集封装到Order对象中

2. SQL语句

```
SELECT * FROM t_order o,t_customer c WHERE o.customer_id=c.customer_id AND
o.order_id=?
```

3. POJO封装结果集

上述SQL语句查询出来的结果集



结果集要封装到Order对象中，而结果集中既包含Order的信息又包含Customer的信息，所以Order类的编写为:

```
public class Order {
    private Integer orderId;
    private String orderName;
    //表示Order和Customer的对一关系
    private Customer customer;
}
```

Customer类的编写为:

```
public class Customer {
    private Integer customerId;
    private String customerName;
}
```

4. Mapper接口中的抽象方法

```
public interface OrderMapper {
    Order selectOrderWithCustomer(Integer orderId);
}
```

5. 映射配置文件OrderMapper.xml

```
<?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.atguigu.mapper.OrderMapper">

    <!--
        创建一个手动映射规则:resultMap标签
        autoMapping如果为true就表示开启自动映射
    -->
    <resultMap id="orderCustomerMap" type="com.atguigu.pojo.Order"
autoMapping="true">
        <!--
            要进行一对一映射: association标签
            也就是说要将customer_id和customer_name属性，映射到一个Customer对象中
            并且要将这个Customer对象赋值给order的customer属性
            一、association标签的属性
               1. property: 表示将一对一映射的结果赋值给JavaBean的哪个属性
               2. javaType: 就是要赋值的JavaBean的属性的类型
        -->
        <association property="customer" javaType="com.atguigu.pojo.Customer"
autoMapping="true">
        </association>
    </resultMap>

    <select id="selectOrderWithCustomer" resultMap="orderCustomerMap">
        SELECT * FROM t_order o,t_customer c WHERE o.customer_id=c.customer_id
AND o.order_id=#{orderId}
    </select>
</mapper>
```

6. 在Mybatis全局配置文件中注册Mapper配置文件

```
<!--加载映射配置文件:指定映射配置文件的路径-->
<mappers>
    <mapper resource="mappers/OrderMapper.xml"/>
</mappers>
```

7. junit测试程序

```
@Test
public void test1v1(){
    Order order = orderMapper.selectOrderWithCustomer(1);
    System.out.println(order);
}
```

第三节 一对多查询

1. 目标

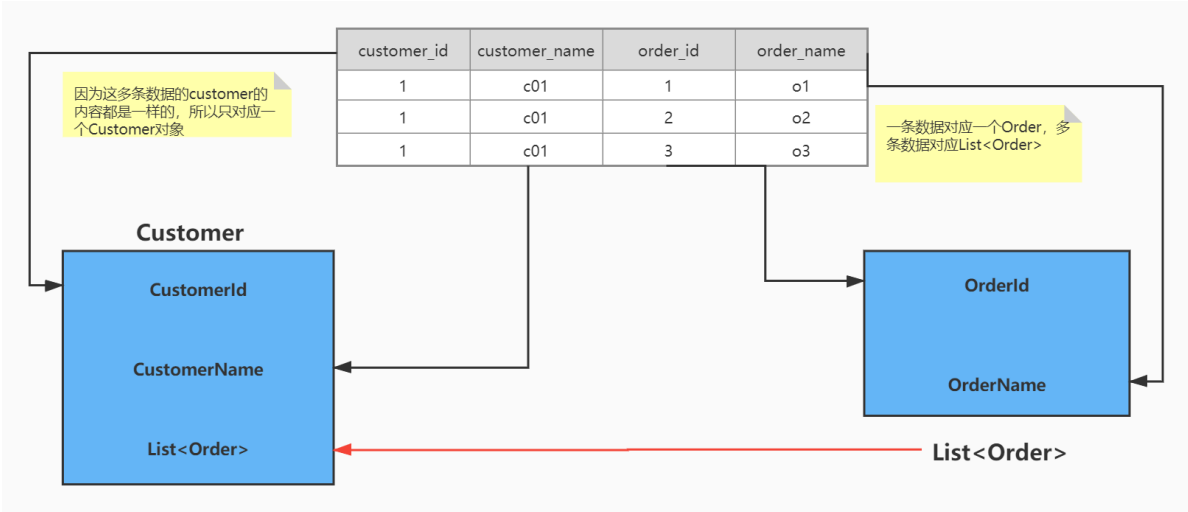
根据客户的ID查询客户信息，并且查询出该客户的所有订单信息,将查询的结果集封装到Customer对象中

2. SQL语句

```
SELECT * FROM t_customer c,t_order o WHERE o.customer_id=c.customer_id AND
c.customer_id=?
```

3. POJO封装结果集

上述SQL语句查询出来的结果集



结果集封装到Customer中，所以Customer中既要包含当前客户的信息，又要包含当前客户的多条订单信息，所以Customer类的编写为:

```
public class Customer {
    private Integer customerId;
    private String customerName;
    private List<Order> orderList;
}
```

4. Mapper接口中的抽象方法

```
public interface CustomerMapper {

    Customer selectCustomerWithOrderList(Integer customerId);

}
```

5. 映射配置文件CustomerMapper .xml

```
<?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.atguigu.mapper.CustomerMapper">
    <!--
        定义手动映射规则
    -->
    <resultMap id="customerOrderListMap" type="com.atguigu.pojo.Customer">
        <id column="customer_id" property="customerId"/>
        <result column="customer_name" property="customerName"/>
        <!--
            进行一对多映射,使用collection标签
                ofType属性指的是orderList的泛型
        -->
        <collection property="orderList" ofType="com.atguigu.pojo.Order"
autoMapping="true">
        </collection>
    </resultMap>
    <select id="selectCustomerWithOrderList" resultMap="customerOrderListMap">
        SELECT * FROM t_customer c,t_order o WHERE o.customer_id=c.customer_id
AND c.customer_id=#{customerId}
    </select>
</mapper>
```

6. 在Mybatis的全局配置文件中注册Mapper配置文件

```
<!--加载映射配置文件:指定映射配置文件的路径-->
<mappers>
    <mapper resource="mappers/CustomerMapper.xml"/>
</mappers>
```

7. junit测试程序

```
@Test
public void test1vn(){
    Customer customer = customerMapper.selectCustomerWithOrderList(1);
    System.out.println(customer);
}
```

第三章 全局配置文件中的高级配置

第一节 配置类型别名

目标

让Mapper配置文件中使用的实体类类型名称更简洁。

具体配置

1. 全局配置文件中配置类型别名

```
<!--
        typeAliases里面就是别名配置
            1. 每一个typeAlias标签就表示配置一个别名
                type属性:要进行别名配置的类
                alias属性:取的别名
            2. 因为所有的POJO类基本上都是放在同一个包中,所以我们可以采用包扫描进行别名配置
               用package标签进行包扫描,别名就是该类的类名(不区分大小写)
        我们一般采用第二种(也就是包扫描的方式进行别名配置)
    -->
<typeAliases>
    <package name="com.atguigu.pojo"/>
</typeAliases>
```

2. Mapper配置文件中使用类型别名

```
<!-- Employee selectEmployeeById(Integer empId); -->
<select id="selectEmployeeById" resultType="Employee">
    select emp_id,emp_name,emp_salary,emp_gender,emp_age from t_emp
    where emp_id=#{empId}
</select>
```

第二节 配置类型处理器(了解)

1. Mybatis内置类型处理器

无论是 MyBatis 在预处理语句（PreparedStatement）中设置一个参数时，还是从结果集中取出一个值时，都会用类型处理器将获取的值以合适的方式转换成 Java 类型。

Mybatis提供的内置类型处理器：

类型处理器	Java 类型	JDBC 类型
BooleanTypeHandler	java.lang.Boolean, boolean	数据库兼容的 BOOLEAN
ByteTypeHandler	java.lang.Byte, byte	数据库兼容的 NUMERIC 或 BYTE
ShortTypeHandler	java.lang.Short, short	数据库兼容的 NUMERIC 或 SHORT INTEGER
IntegerTypeHandler	java.lang.Integer, int	数据库兼容的 NUMERIC 或 INTEGER
LongTypeHandler	java.lang.Long, long	数据库兼容的 NUMERIC 或 LONG INTEGER
FloatTypeHandler	java.lang.Float, float	数据库兼容的 NUMERIC 或 FLOAT
DoubleTypeHandler	java.lang.Double, double	数据库兼容的 NUMERIC 或 DOUBLE
BigDecimalTypeHandler	java.math.BigDecimal	数据库兼容的 NUMERIC 或 DECIMAL
StringTypeHandler	java.lang.String	CHAR, VARCHAR

2. 日期时间处理

日期和时间的处理，JDK1.8以前一直是个头疼的问题。我们通常使用 JSR310 规范领导者 Stephen Colebourne 创建的 Joda-Time 来操作。JDK1.8已经实现全部的JSR310 规范了。

Mybatis在日期时间处理的问题上，提供了基于 JSR310（Date and Time API）编写的各种日期时间类型处理器。

MyBatis3.4以前的版本需要我们手动注册这些处理器，以后的版本都是自动注册的。

如需注册，需要下载mybatistypehandlers-jsr310，并通过如下方式注册

```
<typeHandlers>
  <typeHandler handler="org.apache.ibatis.type.InstantTypeHandler" />
  <typeHandler handler="org.apache.ibatis.type.LocalDateTimeTypeHandler" />
  <typeHandler handler="org.apache.ibatis.type.LocalDateTypeHandler" />
  <typeHandler handler="org.apache.ibatis.type.LocalTimeTypeHandler" />
  <typeHandler handler="org.apache.ibatis.type.OffsetDateTimeTypeHandler" />
  <typeHandler handler="org.apache.ibatis.type.OffsetTimeTypeHandler" />
  <typeHandler handler="org.apache.ibatis.type.ZonedDateTimeTypeHandler" />
  <typeHandler handler="org.apache.ibatis.type.YearTypeHandler" />
  <typeHandler handler="org.apache.ibatis.type.MonthTypeHandler" />
  <typeHandler handler="org.apache.ibatis.type.YearMonthTypeHandler" />
  <typeHandler handler="org.apache.ibatis.type.JapaneseDateTypeHandler" />
</typeHandlers>
```

3. 自定义类型处理器

当某个具体类型Mybatis靠内置的类型处理器无法识别时，可以使用Mybatis提供的自定义类型处理器机制。

- 第一步：实现 org.apache.ibatis.type.TypeHandler 接口或者继承 org.apache.ibatis.type.BaseTypeHandler 类。
- 第二步：指定其映射某个JDBC类型（可选操作）。
- 第三步：在Mybatis全局配置文件中注册。

3.1 创建自定义类型转换器类

```
package com.atguigu.handler;

import com.atguigu.pojo.Address;
import org.apache.ibatis.type.BaseTypeHandler;
import org.apache.ibatis.type.JdbcType;
import org.apache.ibatis.type.MappedJdbcTypes;
import org.apache.ibatis.type.MappedTypes;

import java.sql.*;

/**
 * 包名:com.atguigu.handler
 * MappedTypes注解要转成的Java类型
 * MappedJdbcTypes表示要进行转换的JDBC类型
 * @author Leevi
 * 日期2021-08-26  15:25
 */
@MappedTypes(Address.class)
@MappedJdbcTypes(JdbcType.VARCHAR)
```

```
public class AddressTypeHandler extends BaseTypeHandler<Address>{
    @Override
    public void setNonNullParameter(PreparedStatement ps, int i, Address
parameter, JdbcType jdbcType) throws SQLException {

    }

    @Override
    public Address getNullableResult(ResultSet rs, String columnName) throws
SQLException {
        //其实就是要将查询到的结果集中的"address"字段的值 映射到 Address对象中去
        //1. 获取"address"字段的值
        String addressValue = rs.getString(columnName);

        /*做健壮性判断*/
        if (addressValue == null || "".equals(addressValue)) {
            return null;
        }

        //2. 解析addressValue
        String[] addressArr = addressValue.split("-");
        //3. 设置Address对象的属性
        Address address = new Address();
        //"address"字段值中的省份部分，设置给Address对象的province属性
        address.setProvince(addressArr[0]);
        //"address"字段值中的城市部分，设置给Address对象的city属性
        address.setCity(addressArr[1]);
        //"address"字段值中的街道部分，设置给Address对象的street属性
        address.setStreet(addressArr[2]);
        return address;
    }

    @Override
    public Address getNullableResult(ResultSet rs, int columnIndex) throws
SQLException {
        return null;
    }

    @Override
    public Address getNullableResult(CallableStatement cs, int columnIndex)
throws SQLException {
        return null;
    }
}
```

3.2 注册自定义类型转换器

在Mybatis全局配置文件中配置：

```
<!--
        注册自定义类型处理器
        每一个typeHandler标签表示注册一个类型处理器,它的属性如下:
            1. handler属性: 要进行注册的类型处理器的全限定名
    -->
<!--注册自定义的类型处理器-->
<typeHandlers>
    <package name="com.atguigu.handler"/>
    <!--<typeHandler handler="com.atguigu.handler.AddressTypeHandler" />-->
</typeHandlers>
```

第三节 Mapper映射

Mybatis允许在指定Mapper映射文件时，只指定其所在的包：

```
<mappers>
        <package name="com.atguigu.mapper"/>
</mappers>
```

此时这个包下的所有Mapper配置文件将被自动加载、注册，比较方便。

但是，要求是：

- Mapper接口和Mapper配置文件名称一致
- Mapper配置文件放在Mapper接口所在的包内

如果工程是Maven工程，那么Mapper配置文件还是要放在resources目录下：