MyBatis

1、它是一个持久层框架

2、持久层：实现的就是对数据库的持久化操作，就是 CRUD 操作。指的就是增删改查操作。

3、Java 中原生的持久化方法：JDBC. (Java Database Connectivity)Java 数据库连接技术。

JDBC 中提供了操作数据库的方法。

回顾一下 JDBC 操作数据库的过程：

JDBC 执行的过程如下：

①加载驱动 Class.forName("com.mysql.jdbc.Driver"); //驱动字符串在代码中写死 ？能否写到一个文件中

② 获得连接 Connection conn = DriverManager.getConnection("jdbc:mysql://192.168.1.123:3306/mybatis","root","root");

连接字符串、用户名和密码都在代码中写死 ？ 能否写到一个文件中

③ 获得状态集 Statement/PreparedStatement 发送并执行 SQL

Statement stmt = conn.createStatement();

④ 获得结果(集)

int num = stmt.executeUpdate(sql); //增删改 sql 程序员自己写

ResultSet rs = stmt.executeQuery(sql); //查询 sql 程序员自己写

此处的 SQL 也是在代码中编写，不便于修改 ？ 能否也写在文件中

⑤ 处理结果(集)

```
while(rs.next()){
    //封装成对象 产生查询的结果与对象的映射 ？能否不需要程序员自己写
}
```

⑥释放资源

rs.close(); stmt.close(); conn.close();

从原生的 JDBC 看出的问题：很多可能需要被修改的代码都以硬编码的方式写在了程序中，不便于代码的维护和扩展，解决的方法是使用文件来维护。

这时，存在一个框架：MyBatis，某种程度上 MyBatis 就是对 JDBC 的封装，且通用和需要修改的数据以**配置文件(是一系列的 XML 文件)**的形式存在。

4、 我的第一个 MyBatis

首先创建一个工程，然后导入框架的 jar 包(使用框架就是在使用别人的东西)

引入 jar 包：①将 jar 包复制粘贴到工程的 lib 中；②如果使用 maven，通过 maven 的依赖

再其次，我们知道 MyBatis 使用的是配置文件进行数据源管理的，那么该配置文件该如何编写？

（1）MyBatis 的核心配置文件 Config，通常将其存放在类路径下。

主要会配置数据源、事务等。

```xml
mybatis-config.xml
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <!-- 配置数据源  -->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://localhost:3309/mystudy"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="com/ww/mapper/UserMapper.xml"/>
    </mappers>
</configuration>
```

（2）ORM （Object Relationship Mapping）对象关系映射

持久层做的事情：实现了**程序中的数据**与**数据库表中数据**之间的映射。

实现程序中**对象**与表中的**记录**之间的映射。

以前我们对于持久层/数据访问层，DAO(Data Access Object)：

就是要能够 将程序中对象以数据的形式保存到数据库表中

同时，也要能够将数据库表中的数据查询出来封装成对象供程序使用。

Java 程序	数据库
类	表
对象	记录/一条数据
属性	字段/列

强调：通常我们在写代码时，一张表就会对应一个实体类。

表创建完成后 --> 对应的实体类 也创建完成。

（3）编写映射文件 XxxMapper.xml

编写 SQL 语句，实现表中的记录 与 类的对象之间的映射

注：此处的 XxxMapper 就好比以前我们写的 XxxDao

UserMapper.xml

```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">
<!--
强调：①namespace要和XxxMapper接口的全路径名保持一致
②insert、delete、update、select中的id必须要和XxxMapper中的某个方法名相同
-->
<mapper namespace="com.ww.mapper.UserMapper">
    <!-- 编写我们的SQL，实现映射 -->
    <!--
     parameterType: 传入参数的类型，就是方法的参数类型
     #{}占位符
     -->
    <insert id="save" parameterType="com.ww.pojo.User">
        insert into t_user(name,sex) values(#{userName},#{sex})
    </insert>
</mapper>
```

测试类：

```java
public static void main(String[] args) throws IOException {
    //加载mybatis的主配置文件
    InputStream is = Resources.getResourceAsStream("mybatis-config.xml");
    //封装获取SqlSessionFactory对象
    SqlSessionFactory sqlSessionFactory =  new SqlSessionFactoryBuilder().build(is);
    //参数true ：事务自动提交    false和不写:事务不会自动提交
    SqlSession session = sqlSessionFactory.openSession( b: false);
    //获得UserMapper对象
    UserMapper userMapper = session.getMapper(UserMapper.class);
    User user = new User();
    user.setUserName("HanMeimei");
    user.setSex('f');
    userMapper.save(user);
    session.commit();//手动提交事务
    session.close();
}
```

5、示例

（1）核心配置文件

该配置文件在与 spring 进行整合的时候，会将该配置交给 spring 容器管理。

```xml
<configuration>
    <typeAliases>
        <!--
        <typeAlias type="com.ww.po.Student" alias="Student"/>
        -->
        <package name="com.ww.po"/>
        <package name="com.ww.vo"/>
    </typeAliases>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://localhost:3309/mybatis?useUnicode=true&amp;characterEncoding=utf8"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <!--
        <mapper resource="com/ww/mapper/StudentMapper.xml"/>
        -->
        <package name="com.ww.mapper"/>
    </mappers>
</configuration>
```

（2）表

```
mysql> desc t_student;
+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| id      | int(11)     | NO   | PRI | NULL    | auto_increment |
| name    | varchar(30) | YES  |     | NULL    |                |
| age     | int(11)     | YES  |     | NULL    |                |
| address | varchar(30) | YES  |     | NULL    |                |
+---------+-------------+------+-----+---------+----------------+
```

（3）实体类

Student.java

```java
package com.ww.po;

/**
 * 实体类
 *  与表一一对应
 *  类中的属性 对应  表中的字段
 */
public class Student {
    private int id;
    private String stuName;
    private int age;
    private String address;
```

（3）编写 XxxMapper 接口

接口就是方法的定义，表示了一个能力：就是能做什么，但自己不做。

```java
package com.ww.mapper;

import ...

public interface StudentMapper {
    /** 保存学生信息到数据库中 ...*/
    public int save(Student student);

    /** 查询所有的学生 ...*/
    public List<Student> findAll();

    public List<Student> findAll2();

    /** 根据ID查询一个学生信息 ...*/
    public Student findStudentById(int id);

    /** 修改学生信息 ...*/
    public void update(Student student);

    /** 删除学生信息 ...*/
    public void delete(int id);
```

（4）编写映射文件 XxxMapper.xml

该映射文件就好比 XxxMapper 接口的实现类。

```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">
<!-- 要求namespace的值与mapper接口的全路径名相同 -->
<!-- mybatis框架底层在解析该文件的时候
    获取mapper中namespace的动态代理对象
-->
<mapper namespace="com.ww.mapper.StudentMapper">

</mapper>
```

重点讲解 XxxMapper.xml 中的元素/标签。

①插入操作

```java
/**
 * 保存学生信息到数据库中
 * @param student 学生信息
 */
public int save(Student student);
```

```xml
<insert id="save" parameterType="com.ww.po.Student">
    <!-- #就是占位符 -->
    insert into t_student(name,age,address) values(#{stuName},#{age},#{address})
</insert>
```

parameterType：传入参数，写的就是方法的参数的类型

resultType: 传出参数，设置方法的返回类型

#{}是占位符，如果传入参数是对象，那么{}中直接写对象的属性来获取对象属性的值。

②查询操作

（1）查询的列名和属性名不同，使用别名的方式解决

```java
/**
 * 查询所有的学生
 * @return 所有学生的集合
 */
public List<Student> findAll();
```

```xml
<!--
    返回数据是对象或者对象集合，resultType中填写对象的类型
    resultType: 传出参数，设置方法的返回类型
 -->
<select id="findAll" resultType="com.ww.po.Student">
    <!-- select * from t_student -->
    <!-- select id,name,age,address from t_student -->
    <!-- 查询的列和类中的属性不同时，框架底层无法实现自动映射。
        注意：映射本质是调用pojo中bean的set方法
        方法1：使用别名的方式
    -->
    select id,name stuName,age,address from t_student
</select>
```

（2）查询的列名和属性名不同，使用 resultMap 来手动进行映射关系的维护

```java
public List<Student> findAll2();
```

```xml
<!-- 针对属性名和列名不同的情况，其实还可以使用resultMap标签实现 -->
<resultMap id="studentMapper" type="Student">
    <!-- 如果列名 和 属性名相同，可以不写 -->
    <!--
    <id property="id" column="id"></id>
    -->
    <result column="name" property="stuName"></result>
    <!--
    <result column="age" property="age"></result>
    <result column="address" property="address"/>
    -->
</resultMap>

<select id="findAll2" resultMap="studentMapper">
    select * from t_student
</select>
```

此处的类型没有写全名，是应为在主配置文件中进行了别名配置。

方法的返回类型

一致

mybatis-config.xml

```xml
<typeAliases>
    <!--
        为类取别名的，alias就是别名，可以随便写，但通常就是类名
        <package>标签是以扫描的方式进行别名，都默认是类名
     -->
    <!--
    <typeAlias type="com.ww.po.Student" alias="Student"/>
    -->
    <package name="com.ww.po"/>
    <package name="com.ww.vo"/>
</typeAliases>
```

（3）既有传入参数也有传出参数

```java
/**
 * 根据ID查询一个学生信息
 * @param id 学生编号
 * @return 学生对象
 */
public Student findStudentById(int id);
```

```xml
<!-- 根据编号查询一个学生
    parameterType: 输入参数，实现输入值的映射
    resultType: 输出参数，实现将查询结果映射到值的映射关系
    parameterType类型为普通类型  占位符中的值可以随便写
-->
<select id="findStudentById" parameterType="int" resultMap="studentMapper">
    select id,name,age,address
    from t_student
    where id = #{id}
</select>
```

③修改操作

```java
/**
 * 修改学生信息
 * @param student 修改后的学生信息
 */
public void update(Student student);
```

```xml
<!-- 修改学生信息  -->
<update id="update" parameterType="Student">
    update t_student
    set name = #{stuName},age = #{age},address = #{address}
    where id = #{id}
</update>
```

④删除操作

```java
/**
 * 删除学生信息
 * @param id 被删除的学生的编号
 */
public void delete(int id);
```

```xml
<!-- 删除学生 -->
<delete id="delete" parameterType="int">
    delete from t_student where id = #{xxx}
</delete>
```

⑤ 特殊操作值 #{} 和 ${} ，它们是在 Mybatis 中获取值使用的两种方式

```java
/**
 * 根据学生的姓名查询所有所学
 * @param name 学生姓名
 * @return 查询出来的所有学生
 */
public List<Student> findStudentsByName(String name);
```

```
<!-- 根据学生姓名查找所有的学生
    #{} 占位符
    ${} 用于字符串拼接  必须要加''
-->
<select id="findStudentsByName" parameterType="java.lang.String" resultMap="studentMapper">
    select id,name,age,address
    from t_student
    <!-- where name = #{stuName} -->
    where name = '${stuName}'
</select>
```

⑥ 方法的传入参数是多个值的情况

```
/**
 * 根据名字和地址查询学生
 * @param name 名字中的关键字
 * @param address 地址中的关键字
 * @return 查询出来的学生
 */
public List<Student> findStudentsByNameAndAddress(String name,String address);
```

```
<!-- 根据名字和地址中的关键字查询学生
    多个参数时，无法输入输入参数类型
    获取参数的值时，使用arg0,arg1,...   或者param1,param2,...
-->
<select id="findStudentsByNameAndAddress" resultMap="studentMapper">
    <!--
    select *
    from t_student
    where name = #{param1}
    and address = #{param2}
    -->
    <!--
    select *
    from t_student
    where name = #{arg0}
    and address = #{arg1}
    -->
    select *
    from t_student
    where name = #{param1}
    and address = #{arg1}
</select>
```

⑦ 多个参数在传值前封装成 Map

```
//多个值存储到map中通过键来获取值
public List<Student> findStudentsByNameAndAddress2(Map map);
```

```xml
<!--
    将参数封装到一个Map中然后通过键来获取
    下述sql中n和a是Map中的键
-->
<select id="findStudentsByNameAndAddress2" parameterType="java.util.Map" resultMap="studentMapper">
    select *
    from t_student
    where name = #{n}
    and address = #{a}
</select>
```

```java
//测试方法
private static void showStudentsByMap() {
    SqlSession session = MyUtil.getSession();
    StudentMapper studentMapper = session.getMapper(StudentMapper.class);
    Map<String,String> map = new HashMap<~>();
    map.put("n","Lily");
    map.put("a","Nanjing");
    List<Student> studentList = studentMapper.findStudentsByNameAndAddress2(map);
    if(null != studentList && !studentList.isEmpty()){
        for (Student s : studentList) {
            System.out.println(s);
        }
    }else{
        System.out.println("没有学生！");
    }
}
```

⑧多个参数通过注解的方式传值

```java
//通过注解的方式来获取值
public List<Student> findStudentsByNameAndAddress3(@Param("nn") String name, @Param("aa") String address);
```

```xml
<!-- 通过注解的方式来获取值 -->
<select id="findStudentsByNameAndAddress3" resultMap="studentMapper">
    select *
    from t_student
    where name = #{nn}
    and address = #{aa}
</select>
```

⑨多个传入参数也可以封装成一个对象来传递

```java
/**
 * 根据条件进行查询
 * @param studentVo 查询条件封装的vo对象
 * @return 查询出的所有学生
 */
public List<Student> findStudentsBySomeCondition(StudentVo studentVo);
```

```xml
<!-- 也可以将查询的参数封装成一个vo对象 -->
<select id="findStudentsBySomeCondition" parameterType="com.ww.vo.StudentVo" resultMap="studentMapper">
    select id,name,age,address
    from t_student
    where name = #{name}
    and address = #{address}
</select>
```

```java
//测试代码
private static void showStudentsBySomeCondition() {
    SqlSession session = MyUtil.getSession();
    StudentMapper studentMapper = session.getMapper(StudentMapper.class);
    StudentVo stuVo = new StudentVo();
    stuVo.setName("Tony");
    stuVo.setAddress("beijing");
    List<Student> studentList = studentMapper.findStudentsBySomeCondition(stuVo);
    if(null != studentList && !studentList.isEmpty()){
        for (Student s : studentList) {
            System.out.println(s);
        }
    }else{
        System.out.println("没有学生！");
    }
}
```

10、动态 SQL

```java
/**
 * 根据学生姓名关键字 和 年龄区间进行查找
 * @param studentVo 学生查询条件所封装的视图对象
 * @return 查询出来的所有学生
 */
public List<Student> findStudentsByNameAndAge(StudentVo studentVo);
```

```xml
<!-- 多条件模糊查询
    姓名关键字、年龄下限和上限
-->
<select id="findStudentsByNameAndAge" parameterType="StudentVo" resultMap="studentMapper">
    <!-- 如果三个条件都写，可以写以下的SQL -->
    <!--
    select * from t_student where name like '%${name}%' and age &gt;= #{minAge} and age &lt;= #{maxAge}
    -->
```

```xml
    <!--
    select * from t_student where
    <if test="null != name and '' != name">
        name like '%${name}%'
    </if>
    <if test="null != minAge and '' != minAge">
        and age &gt;= #{minAge}
    </if>
    <if test="null != maxAge and '' != maxAge">
        and age &lt;= #{maxAge}
    </if>
    -->
```

```
<!--
select * from t_student where 1 = 1
<if test="null != name and '' != name">
   and name like '%${name}%'
</if>
<if test="null != minAge and '' != minAge">
    and age &gt;= #{minAge}
</if>
<if test="null != maxAge and '' != maxAge">
    and age &lt;= #{maxAge}
</if>
-->
```

```
<!-- <where>会省略第一个and -->
<!--
select * from t_student
<where>
    <if test="null != name and '' != name">
        and name like '%${name}%'
    </if>
    <if test="null != minAge and '' != minAge">
        and age &gt;= #{minAge}
    </if>
    <if test="null != maxAge and '' != maxAge">
        and age &lt;= #{maxAge}
    </if>
</where>
-->
```

```
select * from t_student
<where>
    <include refid="namesql"></include>
    <if test="null != minAge and '' != minAge">
        and age &gt;= #{minAge}
    </if>
    <if test="null != maxAge and '' != maxAge">
        and age &lt;= #{maxAge}
    </if>
</where>
</select>
```

```
<sql id="namesql">
    <if test="null != name and '' != name">
        and name like '%${name}%'
    </if>
</sql>
```

11、传入参数中有集合，该如何遍历

```
public List<Student> findStudentsByIds(StudentIdsVo studentIdsVo);
```

```
public class StudentIdsVo {
    private String name;
    private List<Integer> ids;
```

```xml
<!--  了解 -->
<!-- 根据ID批量查询 -->
<select id="findStudentsByIds" parameterType="StudentIdsVo" resultMap="studentMapper">
    select *
    from t_student
    <where>
        <include refid="namesql"></include>
        <if test="null != ids ">
            <!-- select * from t_student where name like '%${name}%' and (id = 1 or id = 3 or ...)  -->
            <!--
            <foreach collection="ids" item="sid" open="and (" close=")" separator="or">
                id = #{sid}
            </foreach>
            -->
            <!-- select * from t_student where name like '%${name}%' and id in(1,2,8)  -->
            <foreach collection="ids" item="sid" open="and id in(" close=")" separator=",">
                #{sid}
            </foreach>
        </if>
    </where>
</select>
```

```java
//测试
private static void showStudentsByNameAndIds() {
    SqlSession session = MyUtil.getSession();
    StudentMapper studentMapper = session.getMapper(StudentMapper.class);
    StudentIdsVo studentIdsVo = new StudentIdsVo();
    studentIdsVo.setName("o");
    List<Integer> ids = new ArrayList<~>();
    ids.add(2);
    ids.add(7);
    ids.add(9);
    ids.add(10);
    studentIdsVo.setIds(ids);
    List<Student> studentList = studentMapper.findStudentsByIds(studentIdsVo);
    if(null != studentList && !studentList.isEmpty()){
        for (Student s : studentList) {
            System.out.println(s);
        }
    }else{
        System.out.println("没有学生！");
    }
}
```

6、多表的关联查询

一个部门 有 多个员工　　1:n

一个员工　有 多个房产　　1:n

一个员工属于一个部门　　1:1

一个房产属于一个员工　　1:1

存在一个关系：　1 对 1　 或 1 对多的关系

```
mysql> desc t_dept;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

```java
public class Dept {
    private int id;
    private String name;

    private List<Employee> employees;
```

```
mysql> desc t_emp;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| id       | int(11)     | NO   | PRI | NULL    | auto_increment |
| name     | varchar(30) | YES  |     | NULL    |                |
| birthday | date        | YES  |     | NULL    |                |
| sex      | varchar(2)  | YES  |     | NULL    |                |
| deptid   | int(11)     | YES  | MUL | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
```

```java
public class Employee {
    private int id;
    private String name;
    private Date birthDay;
    private String sex;
    private int deptId;

    private Dept dept;
    private List<House> houseList;
```

```
mysql> desc t_house;
+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| id      | int(11)     | NO   | PRI | NULL    | auto_increment |
| houseno | varchar(20) | YES  |     | NULL    |                |
| address | varchar(50) | YES  |     | NULL    |                |
| empid   | int(11)     | YES  | MUL | NULL    |                |
+---------+-------------+------+-----+---------+----------------+
```

```java
public class House {
    private int id;
    private String houseNo;
    private String address;
    private int empId;

    private Employee employee;
```

示例

（1）查询所有的员工

```java
/**
 * 查询所有的员工
 * @return 所有的员工
 */
public List<Employee> findAllEmployees();
```

```xml
<select id="findAllEmployees" resultType="Employee">
    select id,name,birthday,sex,deptid
    from t_emp
</select>
```

（2）查询员工的名字，生日和部门的名字

查询的数据来自多张表，且不是表中全部内容，可以定义一个 vo

```java
public class EmpDeptVo {
    private String empName;
    private Date birthDay;
    private String deptName;
```

```java
/**
 * 查询所有员工的名字、年龄及所属部门的名字
 */
public List<EmpDeptVo> findAllEmpAndDeptName();
```

```xml
<select id="findAllEmpAndDeptName" resultType="EmpDeptVo">
    select
    t_emp.name empName,
    t_emp.birthday,
    t_dept.name deptName
    from t_emp
    inner join t_dept
    on t_emp.deptid = t_dept.id
</select>
```

（3）查询所有员工及其部门信息

```java
/**
 * 查询所有部门信息及其所有的员工信息
 * @return 部门
 */
public List<Dept> findAllDeptAndEmployee();
```

```java
public class Employee {
    private int id;
    private String name;
    private Date birthDay;
    private String sex;
    private int deptId;

    private Dept dept; //1:1
```

```xml
<!--
    在实现级联（关联）查询的时候，请务必分析主从表。
    查询所有员工信息及其所属的部门信息
    分析：查询的主体是什么？ 员工   级联   部门
    主表：t_emp   从表：t_dept
    如果一个pojo:plain old java object中的某一个属性的类型是对象类型，此时resultType就无法解决映射关系
    需要使用resultMap
 -->
<select id="findAllEmployeeAndDept" resultMap="employeeDeptMap">
    select
    t_emp.*,
    t_dept.id deptId,
    t_dept.name deptName
    from t_emp,t_dept
    where t_emp.deptid = t_dept.id
</select>
```

```xml
<!-- 实现查询的列到pojo中属性的映射 -->
<resultMap id="employeeDeptMap" type="Employee">
    <!-- id映射主键 -->
    <id column="id" property="id"></id>
    <!-- 其它简单类型使用result实现映射 -->
    <result column="name" property="name"></result>
    <result column="birthday" property="birthDay"></result>
    <result column="sex" property="sex"></result>
    <result column="deptid" property="deptId"></result>
    <!--
        单个对象类型，某种程度就是在描述1-1的关系
        property: 就是该类中的对象属性
    -->
    <association property="dept">
        <id column="deptId" property="id"></id>
        <result column="deptName" property="name"></result>
    </association>
</resultMap>
```

（4）查询所有的部门信息及其员工信息

```java
/**
 * 查询所有部门信息及其所有的员工信息
 * @return 部门
 */
public List<Dept> findAllDeptAndEmployee();
```

```xml
<!-- 查询所有部门信息及其所有的员工
    分析：主表 t_dept  关联表 t_emp
        最终要封装的对象应该是部门对象      部门中有员工
        一个部门中有多个员工   1-n关系
        注意的是： 部门中也可以没有员工  所以不应该用内连接，而应该使用外连接
    存在一个问题： 部门中可能没有员工，那这个部门的信息也得查出来。
-->
<select id="findAllDeptAndEmployee" resultMap="deptEmployeeMap">
    select
    t_dept.*,
    t_emp.id empId,
    t_emp.name empName,
    t_emp.birthday,
    t_emp.sex,
    t_emp.deptid
    from t_dept
    <!-- inner join t_emp -->
    left join t_emp
    on t_dept.id = t_emp.deptid
</select>
```

```xml
<resultMap id="deptEmployeeMap" type="Dept">
    <!-- 在使用1-1，1-n的关系模型时，切记使用ID来唯一标识创建的对象 -->
    <id column="id" property="id"></id>
    <result column="name" property="name"></result>
    <!-- 1-n关系:使用collection -->
    <collection property="employees" ofType="Employee">
        <id column="empId" property="id"></id>
        <result column="empName" property="name"></result>
        <result column="birthday" property="birthDay"></result>
        <result column="sex" property="sex"></result>
        <result column="deptid" property="deptId"></result>
        <!-- 列名和属性名相同也需要写上映射关系 -->
    </collection>
</resultMap>
```

（5）查询所有的部门及其员工和员工的房产信息

```java
/**
 * 查询所有的部门信息
 *       及其拥有的所有员工信息和员工的房产信息
 * @return 所有的部门
 */
public List<Dept> findAllDeptAndEmployeeAndHouse();
```

```xml
<select id="findAllDeptAndEmployeeAndHouse" resultMap="deptEmpHouseMapper">
    select
    t_dept.*,
    t_emp.id empid,
    t_emp.name empname,
    t_emp.birthday,
    t_emp.sex,
    t_emp.deptid dept_id,
    t_house.id houseid,
    t_house.houseno,
    t_house.address,
    t_house.empid emp_id
    from t_dept
    <!-- inner  join t_emp -->
    left join t_emp
    on t_dept.id = t_emp.deptid
    <!-- inner join t_house -->
    left join t_house
    on t_emp.id = t_house.empid
</select>
```

```xml
<!--
    查询所有的部门信息
        要求同时查出部门下的所有员工信息和员工的房产信息
    分析：查询的主表应该是 部门表t_dept   关联表（从表）t_emp 及 t_house
    最终要封装的对象应该是部门对象，部门中有员工 ，员工有房产
-->
<resultMap id="deptEmpHouseMapper" type="Dept">
    <id column="id" property="id"></id>
    <result column="name" property="name"></result>
    <!-- 1:n 一个部门有多个员工 -->
    <collection property="employees" ofType="Employee">
        <id column="empid" property="id"></id>
        <result column="empname" property="name"></result>
        <result column="birthday" property="birthDay"></result>
        <result column="sex" property="sex"></result>
        <result column="dept_id" property="deptId"></result>
        <!-- 1:n  一个员工有多个房产 -->
        <collection property="houseList" ofType="House">
            <id column="houseid" property="id"></id>
            <result column="houseno" property="houseNo"></result>
            <result column="address" property="address"></result>
            <result column="emp_id" property="empId"></result>
        </collection>
    </collection>
</resultMap>
```

（6）查询所有的员工及其部门和房产信息

```java
/**
 * 查询所有的员工信息
 *        及其所属的部门信息和所拥有的房产信息
 * @return 所有的员工
 */
public List<Employee> findAllEmployeeAndDeptAndHouse();
```

```xml
<select id="findAllEmployeeAndDeptAndHouse" resultMap="empDeptHouseMapper">
    select
    t_emp.*,
    t_dept.id did,
    t_dept.name deptname,
    t_house.id houseid,
    t_house.houseno,
    t_house.address,
    t_house.empid
    from t_emp
    inner  join t_dept
    on t_emp.deptid = t_dept.id
    left join t_house
    on t_emp.id = t_house.empid
</select>
```

```xml
<!--
    查询所有的员工
        及其所属的部门信息和拥有的房产信息
     分析：要查询的是员工，所有最后封装的对象应该是员工
        主表：t_emp    从表 t_dept、 t_house
 -->
```

```xml
<resultMap id="empDeptHouseMapper" type="Employee">
    <id column="id" property="id"></id>
    <result column="name" property="name"></result>
    <result column="birthday" property="birthDay"></result>
    <result column="sex" property="sex"></result>
    <result column="deptid" property="deptId"></result>
    <!-- 1-1 一个员工属于一个部门  -->
    <association property="dept" javaType="Dept">
        <id column="did" property="id"></id>
        <result column="deptname" property="name"></result>
    </association>
    <!-- 1-n 一个员工可以有多个房产 -->
    <collection property="houseList" ofType="House">
        <id column="houseid" property="id"></id>
        <result column="houseno" property="houseNo"></result>
        <result column="address" property="address"></result>
        <result column="empid" property="empId"></result>
    </collection>
</resultMap>
```

（7）查询所有的房产信息及其所属的员工的信息和部门信息

```java
/**
 * 查询所有的房产及其所属的员工信息和部门信息
 * @return 所有的房产
 */
public List<House> findAllHouseAndEmployeeAndDept();
```

```xml
<select id="findAllHouseAndEmployeeAndDept" resultMap="houseEmployeeDeptMapper">
    select
    t_house.*,
    t_emp.id eid,
    t_emp.name ename,
    t_emp.birthday ebirthday,
    t_emp.sex esex,
    t_emp.deptid edeptid,
    t_dept.id did,
    t_dept.name dname
    from t_house
    inner join t_emp
    on t_house.empid = t_emp.id
    inner join t_dept
    on t_emp.deptid = t_dept.id
</select>
```

```xml
<!--
    查询房产信息，并查询出该房产属于哪个员工及该员工属于哪个部门的信息
    分析： 查询的主体是房产，所以最后封装的应该是房产对象
    主表： t_house    从表： t_emp,t_dept
-->
```

```xml
<resultMap id="houseEmployeeDeptMapper" type="House">
    <id column="id" property="id"></id>
    <result column="houseno" property="houseNo"></result>
    <result column="addRess" property="address"></result>
    <result column="empId" property="empId"></result>
    <!-- 1:1 一个房产属于一个员工-->
    <association property="employee" javaType="Employee">
        <id column="eid" property="id"></id>
        <result column="ename" property="name"></result>
        <result column="ebirthday" property="birthDay"></result>
        <result column="esex" property="sex"></result>
        <result column="edeptid" property="deptId"></result>
        <!-- 1:1 一个员工属于一个部门  -->
        <association property="dept" javaType="Dept">
            <id column="did" property="id"></id>
            <result column="dname" property="name"></result>
        </association>
    </association>
</resultMap>
```

（8）延迟加载(查询)

```java
/**
 * 查询所有的员工及其部门信息 延迟加载
 * @return 所有的员工
 */
public List<Employee> findEmployeeDeptLazy();
```

```xml
<!-- 延迟加载 -->
<resultMap id="empDeptLazy" type="Employee">
    <id column="id" property="id"></id>
    <result column="name" property="name"></result>
    <result column="birthday" property="birthDay"></result>
    <result column="sex" property="sex"></result>
    <result column="deptid" property="deptId"></result>
    <association property="dept" select="findDeptByIdLazy" column="deptid" fetchType="lazy"></association>
</resultMap>
<select id="findDeptByIdLazy" parameterType="int" resultType="Dept">
    select * from t_dept where id = #{id}
</select>
<select id="findEmployeeDeptLazy" resultMap="empDeptLazy">
    <!-- 开始只查询所有的员工 -->
    select * from t_emp
</select>
```

association: 维护的是 1:1
collection： 维护的是 1：n

日志：log4j.properties 日志文件

```
log4j.properties
1  log4j.appender.stdout=org.apache.log4j.ConsoleAppender
2  log4j.appender.stdout.Target=System.out
3  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
4  log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %m%n
5
6  log4j.appender.file=org.apache.log4j.FileAppender
7  log4j.appender.file.File=abc.txt
8  log4j.appender.file.layout=org.apache.log4j.PatternLayout
9  log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %l  %m%n
10
11 log4j.rootLogger=debug,stdout,file
```

配置向控制台输出的格式
配置向文件输出的格式
向哪个文件输出
debug是日志的级别，日志的级别有
OFF、FATAL、ERROR、WARN、**INFO**、**DEBUG**、TRACE、ALL
以上级别分别由高到底。
配置是否向控制台 及 文件输出，如果不写则不输出。