MyBatis

课程目标

- 理解讲关系的目的。
- 掌握 MyBatis 中关系处理步骤。
- 掌握关系中单向多对一的保存和查询。
- 理解什么是 N+1 问题，并掌握怎么解决该问题。
- 掌握关系中单向多对多的保存，删除，查询。

一、关系概述（了解）

1、关系应用（理解）

生活中数据很多是存在关系的，就是把生活中有关系的数据通过 MyBatis 持久化到数据库，且存储的数据也能表示出来这种关系，再由数据库中把有关系的数据查询出来在页面展示。

- 保存：页面的数据 ---> 使用 Java 对象封装 ---> 通过 MyBatis ---> 数据库表的数据
- 查询：数据库表的数据 ---> 通过 MyBatis ---> 封装成 Java 对象 ---> 页面展示数据

那么这里需要解决问题：

- 怎么使用数据库表设计来表示数据之间关系；
- 怎么使用 Java 类设计来表示对象之间关系；
- 怎么通过 MyBatis 配置来映射上面两者（翻译）。

2、对象关系分类

- 泛化关系
- 实现关系
- 依赖关系
- 关联关系
- 聚合关系
- 组合关系

3、关联关系

A 对象依赖 B 对象，并且把 B 作为 A 的一个成员变量，则 A 和 B 存在关联关系。在 UML 中依赖通常使用实线箭头表示。

3.1、关联关系分类

3.1.1、按照导航性分

若通过 A 对象中的某一个属性可以访问到 B 对象，则说 A 可以导航到 B。

- 单向：只能从 A 通过属性导航到 B，B 不能导航到 A。
- 双向：A 可以通过属性导航到 B，B 也可以通过属性导航到 A。

3.1.2、按照多重性分

- 一对一
- 一对多
- 多对一
- 多对多。

4、判断对象的关系（理解）

- 判断都是从对象的实例上面来看的；
- 判断关系需要根据对象的属性；
- **判断关系必须确定具体需求。**

二、单向多对一之保存（掌握）

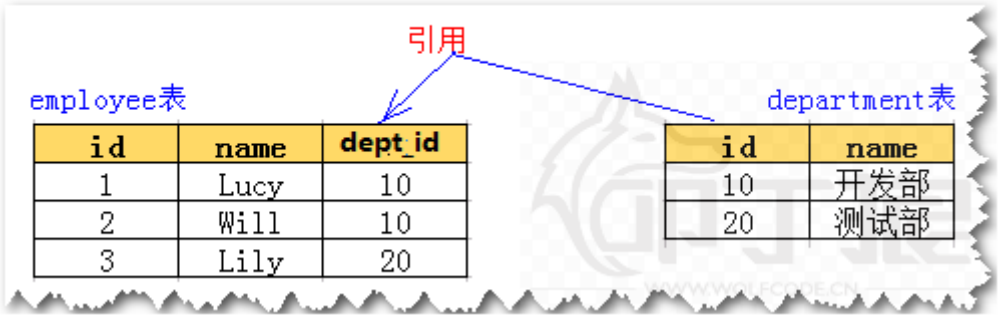
拷贝之前的项目，改项目名为 many2one，再导入。

1、需求

保存一个部门和两个员工，且这两个员工都是这个部门的。

2、表设计

表设计：外键在many方



3、类设计

```
package cn.wolfcode.domain;

@Setter
@Getter
@ToString
public class Department {
    private Long id;
    private String name;
}
```

```java
package cn.wolfcode.domain;

@Setter
@Getter
@ToString
public class Employee {
    private Long id;
    private String name;
    // 关联属性
    private Department dept;
}
```

4、Mapper 接口和 Mapper XML 文件编写

注意 Mapper XML 放置的位置。

```java
package cn.wolfcode.mapper;

public interface DepartmentMapper {
    void save(Department dept);
}
```

```xml
<!--
    useGeneratedKeys=true 获取数据库保存数据的的主键值
    keyProperty="id" 主键设置设置对象的 id 属性
-->
<insert id="save" useGeneratedKeys="true" keyProperty="id">
    INSERT INTO department(name) VALUES(#{name})
</insert>
```

```java
package cn.wolfcode.mapper;

public interface EmployeeMapper {
    void save(Employee employee);
}
```

```xml
<insert id="save" useGeneratedKeys="true" keyProperty="id">
    INSERT INTO employee(name, dept_id) VALUES(#{name}, #{dept.id})
</insert>
```

5、编写单元测试类

```java
public class Many2oneTest {
    // 保存一个部门和两个员工，且这两个员工都是这个部门的
    @Test
    public void testSave() throws Exception {
        Department dept = new Department();
        dept.setName("开发部");

        Employee e1 = new Employee();
        e1.setName("张三");
        e1.setDept(dept); // 设置关系
```

```
        Employee e2 = new Employee();
        e2.setName("李四");
        e2.setDept(dept); // 设置关系

        SqlSession session = MyBatisUtil.getSession();
        DepartmentMapper departmentMapper =
session.getMapper(DepartmentMapper.class);
        EmployeeMapper employeeMapper = session.getMapper(EmployeeMapper.class);

        // 先保存部门再保存员工
        departmentMapper.save(dept);
        employeeMapper.save(e1);
        employeeMapper.save(e2);

        session.commit();
        session.close();
    }
}
```

三、单向多对一之额外 SQL 查询（掌握）

1、需求

根据员工 id 查询员工，并知道该员工的所在的部门。

2、修改员工的 Mapper 接口及 Mapper XML

```
Employee get(Long id);
```

```
<select id="get" resultType="Employee">
    SELECT id, name, dept_id FROM employee WHERE id = #{id}
</select>
```

3、编写单元测试方法

```
public class Many2oneTest {
    @Test
    public void testGet() throws Exception {
        SqlSession session = MyBatisUtil.getSession();
        EmployeeMapper employeeMapper = session.getMapper(EmployeeMapper.class);
        Employee employee = employeeMapper.get(1L);
        System.out.println(employee);
        session.close();
    }
}
```

4、存在的问题

发现查询出来的员工部门为 null，原因是当结果集的列名与对象的属性名不一致。

```
+----+------+---------+
| id | name | dept_id |
+----+------+---------+
|  1 | 张三 |       1 |
+----+------+---------+
```

```
@Setter
@Getter
@ToString
public class Employee {
    private Long id;
    private String name;
    // 关联属性
    private Department dept;
}
```

那么我们可以在 EmployeeMapper.xml，通过 resultMap 元素来解决。

```
<select id="get" resultMap="baseResultMap">
    SELECT id, name, dept_id FROM employee WHERE id = #{id}
</select>

<resultMap type="Employee" id="baseResultMap">
    <!-- 什么列名对应值封装到对象的什么属性上 -->
    <id column="id" property="id"/>
    <result column="name" property="name"/>
    <result column="dept_id" property="dept.id"/>
</resultMap>
```

但问题只查询出部门的 id，但要查询出部门的名称怎么办呢?

5、手动发送额外 SQL

既然已获得部门的 id，那么再在 DepartmentMapper 中提供 get 方法根据 id 查询部门对象，再把该部门对象设置到员工对象的 dept 属性上。

5.1、修改部门的 Mapper 接口及 Mapper XML

```
Department get(Long id);
```

```
<select id="get" resultType="Department">
    SELECT id, name FROM department WHERE id = #{id}
</select>
```

5.2、修改单元测试方法

```
public class Many2oneTest {
    @Test
    public void testGet() throws Exception {
        SqlSession session = MyBatisUtil.getSession();
        EmployeeMapper employeeMapper = session.getMapper(EmployeeMapper.class);
        Employee employee = employeeMapper.get(1L); // 此时员工对象 dept 的 id 属性上已存在对应的部门 id 值

        // 再调用 DepartmentMapper 接口中根据 id 查询的方法，把对应部门查询出来
```

```
        DepartmentMapper departmentMapper =
session.getMapper(DepartmentMapper.class);
        Department dept = departmentMapper.get(employee.getDept().getId());
        // 把查询出来部门对象设置 employee 对象的 dept 属性上
        employee.setDept(dept);

        System.out.println(employee);
        session.close();
    }
}
```

通过控制台发现会额外发送 SQL 去查询，但问题这种操作（发送额外的 SQL 数据和设置 dept 属性操作）都不想自己做怎么办呢?

6、使用 association 发送额外 SQL

6.1、修改员工 Mapper XML 文件

```
<resultMap type="Employee" id="baseResultMap">
    <!-- 什么列名对应值封装到对象的什么属性上 -->
    <id column="id" property="id"/>
    <result column="name" property="name"/>
    <!-- 使用额外 SQL
        association 针对的关联属性配置，非集合类型
        select      发送什么额外 SQL
        column      发送额外 SQL 参数取上一条 SQL 哪个列的值
        property    封装员工对象的什么属性
    -->
    <association select="cn.wolfcode.mapper.DepartmentMapper.get"
        column="dept_id" property="dept" javaType="Department"/>
</resultMap>
```

额外 SQL 方式可以不配置 javaType 属性（select 元素那已配置了），后面讲的多表查询方式一定要配置。

6.2、修改单元测试方法

```
public class Many2oneTest {
    @Test
    public void testGet() throws Exception {
        SqlSession session = MyBatisUtil.getSession();
        EmployeeMapper employeeMapper = session.getMapper(EmployeeMapper.class);
        Employee employee = employeeMapper.get(1L);
        System.out.println(employee);
        session.close();
    }
}
```

四、单向多对一之多表查询（掌握）

1、额外 SQL 查询 N+1 问题

需求：查询所有员工及其对应部门。假设在 employee 表中有 N 条数据，每一个员工都关联着一个不同的部门 id。当在查询所有员工时，就会发送 N+1 条语句。

- 1 条：SELECT id, name, dept_id FROM employee
- N 条：SELECT id, name FROM department WHERE id = ?

2、代码演示问题

2.1、修改员工 Mapper 接口及 Mapper XML

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

```
<select id="listAll" resultMap="baseResultMap">
    SELECT id, name, dept_id FROM employee
</select>
```

2.2、编写单元测试方法

```
public class Many2oneTest {
    @Test
    public void testListAll() throws Exception {
        SqlSession session = MyBatisUtil.getSession();
        EmployeeMapper employeeMapper = session.getMapper(EmployeeMapper.class);

        List<Employee> employees = employeeMapper.listAll();
        System.out.println(employees);
        session.close();
    }
}
```

3、多表查询

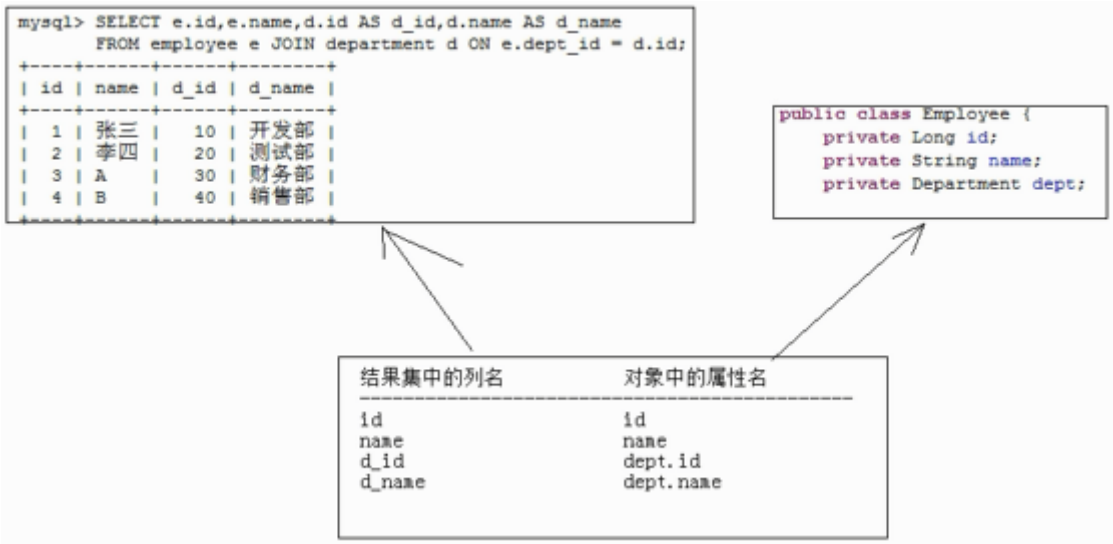
3.1、修改 EmployeeMapper.xml 查询的 SQL

使用多表查询，此时一条 SQL 语句搞定，实现查询所有员工及其对应部门。

```
<select id="listAll" resultMap="baseResultMap">
    SELECT e.id, e.name, d.id AS d_id, d.name AS d_name
    FROM employee e JOIN department d ON e.deptId = d.id
</select>
```

N+1 问题没有，但如何解决结果集映射问题呢?

3.2、使用 resultMap 处理结果集映射



3.2.1、结果集映射方式 1

```
<resultMap type="Employee" id="baseResultMap">
    <id column="id" property="id"/>
    <result column="name" property="name"/>
    <result column="d_id" property="dept.id"/>
    <result column="d_name" property="dept.name"/>
</resultMap>
```

问题：属性过多，dept. 则会重复写好多。

3.2.2、结果集映射方式 2

```
<resultMap type="Employee" id="baseResultMap">
    <id column="id" property="id"/>
    <result column="name" property="name"/>
    <association property="dept" javaType="Department">
        <result column="d_id" property="id"/>
        <result column="d_name" property="name"/>
    </association>
</resultMap>
```

问题：当关联表查询的列太多了，那么前缀 d_ 就要写很多。

3.2.3、结果集映射方式 3

```
<resultMap type="Employee" id="baseResultMap">
    <id column="id" property="id"/>
    <result column="name" property="name"/>
    <association columnPrefix="d_" property="dept" javaType="Department">
        <result column="id" property="id"/>
        <result column="name" property="name"/>
    </association>
</resultMap>
```

五、单向一对多之保存（了解）

拷贝之前的项目，改项目名为 one2many，再导入。

1、需求

保存一个部门和两个员工，且这两个员工都是这个部门的。

2、表设计

和多对一表设计是一样的。

3、类设计

```
package cn.wolfcode.domain;

@Setter
@Getter
@ToString
public class Employee {
    private Long id;
    private String name;
}
```

```
package cn.wolfcode.domain;

@Setter
@Getter
@ToString
public class Department {
    private Long id;
    private String name;
    // 关联属性，建议集合对象直接 new 出来，避免后面写测试类的时候造成空指针。
    private List<Employee> employees = new ArrayList<>();
}
```

4、Mapper 接口和 Mapper XML 文件编写

```
package cn.wolfcode.mapper;

public interface DepartmentMapper {
    void save(Department dept);
}
```

```
<insert id="save" useGeneratedKeys="true" keyProperty="id">
    INSERT INTO department(name) VALUES(#{name})
</insert>
```

```
package cn.wolfcode.mapper;

public interface EmployeeMapper {
    void save(Employee employee);
}
```

```
<insert id="save" useGeneratedKeys="true" keyProperty="id">
    INSERT INTO employee(name) VALUES(#{name})
</insert>
```

5、编写单元测试

```
public class One2manyTest {
    @Test
    public void testSave() throws Exception {
        Department dept = new Department();
        dept.setName("开发部");

        Employee e1 = new Employee();
        e1.setName("张三");
        Employee e2 = new Employee();
        e2.setName("李四");

        SqlSession session = MyBatisUtil.getSession();
        DepartmentMapper departmentMapper =
session.getMapper(DepartmentMapper.class);
        EmployeeMapper employeeMapper = session.getMapper(EmployeeMapper.class);

        departmentMapper.save(dept);
        employeeMapper.save(e1);
        employeeMapper.save(e2);

        session.commit();
        session.close();
    }
}
```

6、存在的问题及解决办法

问题是保存到员工表中的员工数据没有部门 id。解决办法：

- 发送额外 SQL 修改员工的部门（性能较低不推荐）；
- 改成双向的关联关系；
- 在 many 放添加一个 Long 类型的 deptId，在保存部门之后把部门的 id 值设置到员工对象这个 deptId 属性再保存员工。

6.1、修改员工实体类

```
package cn.wolfcode.domain;

@Setter
@Getter
@ToString
public class Employee {
    private Long id;
    private String name;
    private Long deptId; // 这个属性用来封装这个员工的部门 id 值，不是关联属性
}
```

6.2、修改员工 Mapper XML

```
<insert id="save" useGeneratedKeys="true" keyProperty="id">
    INSERT INTO employee(name, dept_id) VALUES(#{name}, #{deptId})
</insert>
```

6.3、修改单元测试方法

```
public class One2manyTest {
    @Test
    public void testSave() throws Exception {
        Department dept = new Department();
        dept.setName("开发部");

        Employee e1 = new Employee();
        e1.setName("张三");
        Employee e2 = new Employee();
        e2.setName("李四");

        SqlSession session = MyBatisUtil.getSession();
        DepartmentMapper departmentMapper =
session.getMapper(DepartmentMapper.class);
        EmployeeMapper employeeMapper = session.getMapper(EmployeeMapper.class);

        // 先保存部门再保存员工
        departmentMapper.save(dept);

        e1.setDeptId(dept.getId());
        e2.setDeptId(dept.getId());

        employeeMapper.save(e1);
        employeeMapper.save(e2);

        session.commit();
        session.close();
    }
}
```

六、单向一对多之额外 SQL 查询（了解）

1、需求

查询部门，并把其部门的员工信息也查询出来。

2、修改部门 Mapper 接口及 Mapper XML

```
Department get(Long id);
```

```
<select id="get" resultType="Department">
    SELECT id, name FROM department WHERE id = #{id}
</select>
```

3、编写单元测试方法

```
public class One2manyTest {
    @Test
    public void testGet() throws Exception {
        SqlSession session = MyBatisUtil.getSession();
        DepartmentMapper departmentMapper =
session.getMapper(DepartmentMapper.class);

        Department department = departmentMapper.get(1L);
        System.out.println(department);
        session.close();
    }
}
```

4、存在的问题

发现只能查询出部门信息，所属的员工信息没有查询出来，怎么办?

- 手动额外发送 SQL。通过获取部门的 id 作为参数，再到员工表中根据部门查询员工数据，手动封装到部门的的 employees 属性上。
- 通过配置使用 MyBatis 来帮我们发送额外 SQL 来完成。

5、使用 collection 发送额外 SQL

5.1、修改部门 Mapper XML

```
<select id="get" resultMap="baseResultMap">
    SELECT id, name FROM department WHERE id = #{id}
</select>

<resultMap type="Department" id="baseResultMap">
    <id column="id" property="id"/>
    <result column="name" property="name"/>
    <!--
        若关联属性是集合类型，使用 collection 来配置
        select      发送额外的 SQL
        column      额外 SQL 参数取之前 SQL 哪里列的值
        property    查询结果封装部门对象什么属性上
    -->
    <collection select="cn.wolfcode.mapper.EmployeeMapper.queryByDeptId"
        column="id" property="employees"/>
</resultMap>
```

5.2、修改员工 Mapper XML

```
<select id="queryByDeptId" resultType="Employee">
    SELECT id, name, dept_id FROM employee WHERE deptId = #{deptId}
</select>
```

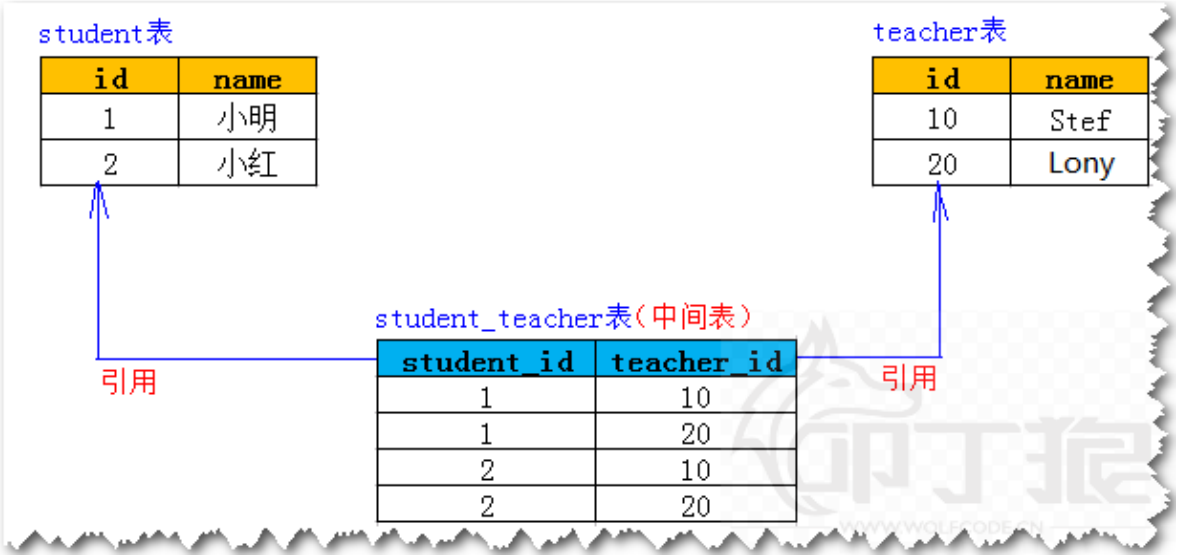
七、单向多对多之保存（掌握）

拷贝之前的项目，改项目名为 many2many，再导入。

1、需求

保存两个学生和两个老师，且这两个老师都教了这个两个学生。

2、表设计



3、类设计

```
package cn.wolfcode.domain;

@Setter
@Getter
@ToString
public class Teacher {
    private Long id;
    private String name;
}
```

```
package cn.wolfcode.domain;

@Setter
@Getter
@ToString
public class Student {
    private Long id;
    private String name;
    // 关联属性
    private List<Teacher> teachers = new ArrayList<>();
}
```

4、Mapper 接口和 Mapper XML 文件编写

```
package cn.wolfcode.mapper;

public interface TeacherMapper {
    void save(Teacher teacher);
}
```

```
<insert id="save" useGeneratedKeys="true" keyProperty="id">
    INSERT INTO teacher(name) VALUES(#{name})
</insert>
```

```
package cn.wolfcode.mapper;

public interface StudentMapper {
    void save(Student student);
    // 往中间表插入关系数据
    void insertRelation(@Param("teacherId")Long teacherId,
@Param("studentId")Long studentId);
}
```

```
<insert id="save" useGeneratedKeys="true" keyProperty="id">
    INSERT INTO student(name) VALUES(#{name})
</insert>
<insert id="insertRelation">
    INSERT INTO teacher_student(teacher_id, student_id) VALUES (#{teacherId}, #
{studentId})
</insert>
```

5、编写单元测试类

```
public class Many2manyTest {
    @Test
    public void testSave() throws Exception {
        Teacher teacher1 = new Teacher();
        teacher1.setName("波老师");
        Teacher teacher2 = new Teacher();
        teacher2.setName("罗老师");

        Student s1 = new Student();
        s1.setName("小强");
        Student s2 = new Student();
        s2.setName("小红");

        s1.getTeachers().add(teacher1);
        s1.getTeachers().add(teacher2);
        // s1 被两个老师教了

        s2.getTeachers().add(teacher1);
        s2.getTeachers().add(teacher2);
        // s2 被两个老师教了

        SqlSession session = MyBatisUtil.getSession();
        TeacherMapper teacherMapper = session.getMapper(TeacherMapper.class);
        StudentMapper studentMapper = session.getMapper(StudentMapper.class);
```

```
        teacherMapper.save(teacher1);
        teacherMapper.save(teacher2);

        studentMapper.save(s1);
        studentMapper.save(s2);

        // 往中间表存入数据老师教学生的关系数据
        for(Teacher t : s1.getTeachers()) {
            studentMapper.insertRelation(t.getId(), s1.getId());
        }
        for(Teacher t : s2.getTeachers()) {
            studentMapper.insertRelation(t.getId(), s2.getId());
        }

        session.commit();
        session.close();
    }
}
```

八、单向多对多之额外 SQL 查询（掌握）

1、需求

根据 id 查询学生，并查询教过其的老师。

2、Mapper 接口和 Mapper XML 文件编写

2.1、修改学生 Mapper 接口及 Mapper XML

```
Student get(Long id);
```

```
<resultMap type="Student" id="baseResultMap">
    <id column="id" property="id"/>
    <result column="name" property="name"/>
    <!-- 关联属性，让 MyBatis 发额外 SQL -->
    <collection select="cn.wolfcode.mapper.TeacherMapper.queryByStudentId"
        column="id" property="teachers"/>
</resultMap>

<select id="get" resultMap="baseResultMap">
    SELECT id, name FROM student WHERE id = #{id}
</select>
```

2.2、修改老师 Mapper XML

```
<select id="queryByStudentId" resultType="Teacher">
    SELECT t.id, t.name FROM teacher_student ts JOIN teacher t ON ts.teacher_id
= t.id
    WHERE ts.student_id = #{studentId}
</select>
```

3、编写单元测试方法

```
public class Many2manyTest {
    @Test
    public void testGet() throws Exception {
        SqlSession session = MyBatisUtil.getSession();
        StudentMapper studentMapper = session.getMapper(StudentMapper.class);
        Student student = studentMapper.get(2L);
        System.out.println(student);
        session.close();
    }
}
```

九、单向多对多之删除（掌握）

1、需求

根据 id 删除学生。

2、Mapper 接口和 Mapper XML 文件编写

2.1、修改学生 Mapper 接口及 Mapper XML

```
void delete(Long id);
```

```
<delete id="delete">
    DELETE FROM student WHERE id = #{id}
</delete>
```

2.2、编写单元测试方法

```
public class Many2manyTest {
    @Test
    public void testDelete() throws Exception {
        SqlSession session = MyBatisUtil.getSession();
        StudentMapper studentMapper = session.getMapper(StudentMapper.class);
        studentMapper.delete(1L);
        session.commit();
        session.close();
    }
}
```

3、存在的问题

但问题是：中间还有一些无用数据，因为该学生都已删除了，怎么办?

4、解决办法

4.1、修改学生 Mapper 接口及 Mapper XML

```
void deleteRelation(Long studentId);
```

```
<delete id="deleteRelation">
    DELETE FROM teacher_student WHERE student_id = #{studentId}
</delete>
```

4.2、修改单元测试方法

```
public class Many2manyTest {
    @Test
    public void testDelete() throws Exception {
        SqlSession session = MyBatisUtil.getSession();
        StudentMapper studentMapper = session.getMapper(StudentMapper.class);

        studentMapper.deleteRelation(1L);
        studentMapper.delete(1L);

        session.commit();
        session.close();
    }
}
```

练习

```
CREATE TABLE `department` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

INSERT INTO `department` VALUES ('1', '人事部');
INSERT INTO `department` VALUES ('2', '财务部');

CREATE TABLE `employee` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `dept_id` bigint(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8;

INSERT INTO `employee` VALUES ('1', '荀彧', '1');
INSERT INTO `employee` VALUES ('2', '王修', '2');
INSERT INTO `employee` VALUES ('3', '邓艾', '2');
```

- 练习一，根据以上表结构和数据，完成以下功能（注意只需编写 Mapper 接口、Mapper XML、实体类和单元测试类）：
    - 完成保存一个名叫`行政部`的部门和名叫`程昱`和`孔融`的两个员工，且这两个员工都是这个部门的。
    - 提供根据 id 查询员工并知道该员工所在部门名称的功能。
    - 提供查询所有员工并知道所有员工所在部门名称的功能。

```
CREATE TABLE `student` (
```

```
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

INSERT INTO `student` VALUES ('1', '小明');
INSERT INTO `student` VALUES ('2', '小红');

CREATE TABLE `teacher` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

INSERT INTO `teacher` VALUES ('1', 'Stef');
INSERT INTO `teacher` VALUES ('2', 'Lony');

CREATE TABLE `teacher_student` (
  `teacher_id` bigint(20) DEFAULT NULL,
  `student_id` bigint(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO `teacher_student` VALUES ('1', '1');
INSERT INTO `teacher_student` VALUES ('2', '1');
INSERT INTO `teacher_student` VALUES ('2', '2');
```

- 练习二，根据以上表结构和数据，完成以下功能（注意只需编写 Mapper 接口、Mapper XML、实体类和单元测试类）：
  - 完成保存一个名叫`Willie`的老师和名叫`小强`和`小刚`的两个学生，且这个老师教过这个两个学生。
  - 提供根据 id 查询学生并知道交过其的老师的功能。
  - 提供根据 id 删除学生的功能。

```
CREATE TABLE `brand` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

INSERT INTO `brand` VALUES ('1', '特斯拉');
INSERT INTO `brand` VALUES ('2', '丰田');

CREATE TABLE `product` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `brand_id` bigint(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8;

INSERT INTO `product` VALUES ('1', 'Model 3', '1');
INSERT INTO `product` VALUES ('2', 'Model Y', '1');
INSERT INTO `product` VALUES ('3', '凯美瑞', '2');
```

- 练习三，根据以上表结构和数据，完成以下功能（注意只需编写 Mapper 接口、Mapper XML、实体类和单元测试类）：

    - 完成保存一个名叫`奥迪`的品牌和名叫`奥迪A6L`和`奥迪Q7`的两个产品，且这两个产品都是这个品牌的。
    - 提供根据 id 查询产品并知道该产品所属品牌名称的功能。
    - 提供查询所有产品并知道所有产品所属品牌名称的功能。

```
CREATE TABLE `student` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

INSERT INTO `student` VALUES ('1', '小明');
INSERT INTO `student` VALUES ('2', '小红');

CREATE TABLE `course` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;

INSERT INTO `course` VALUES ('1', '高等数学');
INSERT INTO `course` VALUES ('2', '商务英语');

CREATE TABLE `student_course` (
  `student_id` bigint(20) DEFAULT NULL,
  `course_id` bigint(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO `student_course` VALUES ('1', '2');
INSERT INTO `student_course` VALUES ('1', '1');
INSERT INTO `student_course` VALUES ('2', '2');
```

- 练习四，根据以上表结构和数据，完成以下功能（注意只需编写 Mapper 接口、Mapper XML、实体类和单元测试类）：
    - 完成保存一个名叫`小虎`的学生和名叫`教育心理学`和`法学理论`的两个课程，且这个学生选修这两门课程。
    - 提供根据 id 查询课程并知道有哪些学生选修了这门课程的功能。
    - 提供根据 id 删除课程的功能。