

1. Mybatis基础操作

学习完mybatis入门后，我们继续学习mybatis基础操作。

1.1 需求

需求说明：

- 根据资料中提供的《tlias智能学习辅助系统》页面原型及需求，完成员工管理的需求开发。

员工管理

姓名	请输入员工姓名	性别	请选择	入职时间	从	开始时间	至	结束时间	查询
<input type="checkbox"/>	赵敏		女	班主任	2008-12-18	2022-07-22 12:05:20	编辑	删除	
<input type="checkbox"/>	风清扬		男	讲师	2015-07-22	2022-07-21 15:00:00	编辑	删除	
<input type="checkbox"/>	风清扬		男	讲师	2015-07-22	2022-07-21 15:00:00	编辑	删除	
<input type="checkbox"/>	风清扬		男	讲师	2015-07-22	2022-07-21 15:00:00	编辑	删除	
<input type="checkbox"/>	风清扬		男	讲师	2015-07-22	2022-07-21 15:00:00	编辑	删除	
<input type="checkbox"/>	风清扬		男	讲师	2015-07-22	2022-07-21 15:00:00	编辑	删除	
<input type="checkbox"/>	风清扬		男	讲师	2015-07-22	2022-07-21 15:00:00	编辑	删除	
<input type="checkbox"/>	风清扬		男	讲师	2015-07-22	2022-07-21 15:00:00	编辑	删除	
<input type="checkbox"/>	风清扬		男	讲师	2015-07-22	2022-07-21 15:00:00	编辑	删除	
<input type="checkbox"/>	风清扬		男	讲师	2015-07-22	2022-07-21 15:00:00	编辑	删除	

+ 新增员工 - 批量删除

每页展示记录数 10 共500条数据 < 1 2 3 4 5 ... 50 > 跳至 1 页

新增员工

*用户名 请输入用户名, 2-20字符, 不可重复

*员工姓名 请输入员工姓名, 2-10个字

*性 别 请选择

图 像

职 位 请选择

入职日期 2022-07-22

归属部门 请选择

保存 取消

修改员工

*用户名 zhaomin

*员工姓名 赵敏

*性 别 女

图 像

职 位 班主任

入职日期 2008-12-18

归属部门 学工部

保存 取消



通过分析以上的页面原型和需求，我们确定了功能列表：

1. 查询
 - 根据主键ID查询
 - 条件查询
2. 新增
3. 更新
4. 删除
 - 根据主键ID删除
 - 根据主键ID批量删除

1.2 准备

实施前的准备工作：

1. 准备数据库表
2. 创建一个新的springboot工程，选择引入对应的起步依赖（mybatis、mysql驱动、lombok）
3. application.properties中引入数据库连接信息
4. 创建对应的实体类 Emp（实体类属性采用驼峰命名）
5. 准备Mapper接口 EmpMapper

准备数据库表

```
1  -- 部门管理
2  create table dept
3  (
4      id          int unsigned primary key auto_increment comment '主键
5          ID',
5      name        varchar(10) not null unique comment '部门名称',
6      create_time datetime    not null comment '创建时间',
```

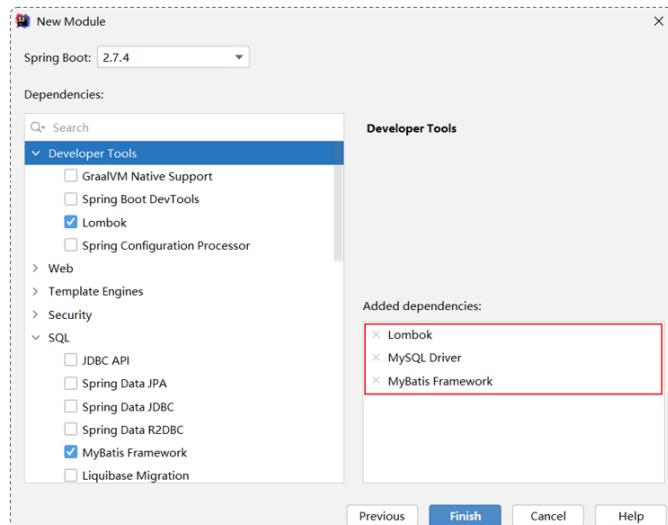
```
7      update_time datetime      not null comment '修改时间',
8  ) comment '部门表';
9  -- 部门表测试数据
10 insert into dept (id, name, create_time, update_time)
11 values (1, '学工部', now(), now()),
12       (2, '教研部', now(), now()),
13       (3, '咨询部', now(), now()),
14       (4, '就业部', now(), now()),
15       (5, '人事部', now(), now());
16
17
18 -- 员工管理
19 create table emp
20 (
21     id          int unsigned primary key auto_increment comment
22     'ID',
23     username    varchar(20)      not null unique comment '用户名',
24     password    varchar(32) default '123456' comment '密码',
25     name        varchar(10)     not null comment '姓名',
26     gender      tinyint unsigned not null comment '性别, 说明: 1 男,
27     2 女',
28     image       varchar(300) comment '图像',
29     job         tinyint unsigned comment '职位, 说明: 1 班主任,2 讲师,
30     3 学工主管, 4 教研主管, 5 咨询师',
31     entrydate   date comment '入职时间',
32     dept_id     int unsigned comment '部门ID',
33     create_time datetime        not null comment '创建时间',
34     update_time datetime        not null comment '修改时间'
35 ) comment '员工表';
36 -- 员工表测试数据
37 INSERT INTO emp (id, username, password, name, gender, image, job,
38 entrydate, dept_id, create_time, update_time)
39 VALUES
40 (1, 'jinyong', '123456', '金庸', 1, '1.jpg', 4, '2000-01-01', 2,
41 now(), now()),
42 (2, 'zhangwujie', '123456', '张无忌', 1, '2.jpg', 2, '2015-01-01', 2,
43 now(), now()),
44 (3, 'yangxiao', '123456', '杨逍', 1, '3.jpg', 2, '2008-05-01', 2,
45 now(), now()),
46 (4, 'weiyixiao', '123456', '韦一笑', 1, '4.jpg', 2, '2007-01-01', 2,
47 now(), now()),
48 (5, 'changyuchun', '123456', '常遇春', 1, '5.jpg', 2, '2012-12-05',
49 2, now(), now()),
```

```

41     (6, 'xiaozhao', '123456', '小昭', 2, '6.jpg', 3, '2013-09-05', 1,
        now(), now()),
42     (7, 'jixiaofu', '123456', '纪晓芙', 2, '7.jpg', 1, '2005-08-01', 1,
        now(), now()),
43     (8, 'zhouzhiruo', '123456', '周芷若', 2, '8.jpg', 1, '2014-11-09', 1,
        now(), now()),
44     (9, 'dingminjun', '123456', '丁敏君', 2, '9.jpg', 1, '2011-03-11', 1,
        now(), now()),
45     (10, 'zhaomin', '123456', '赵敏', 2, '10.jpg', 1, '2013-09-05', 1,
        now(), now()),
46     (11, 'luzhangke', '123456', '鹿杖客', 1, '11.jpg', 5, '2007-02-01',
        3, now(), now()),
47     (12, 'hebiweng', '123456', '鹤笔翁', 1, '12.jpg', 5, '2008-08-18', 3,
        now(), now()),
48     (13, 'fangdongbai', '123456', '方东白', 1, '13.jpg', 5, '2012-11-01',
        3, now(), now()),
49     (14, 'zhangsanfeng', '123456', '张三丰', 1, '14.jpg', 2, '2002-08-
        01', 2, now(), now()),
50     (15, 'yulianzhou', '123456', '俞莲舟', 1, '15.jpg', 2, '2011-05-01',
        2, now(), now()),
51     (16, 'songyuanqiao', '123456', '宋远桥', 1, '16.jpg', 2, '2010-01-
        01', 2, now(), now()),
52     (17, 'chenyouliang', '123456', '陈友谅', 1, '17.jpg', NULL, '2015-03-
        21', NULL, now(), now());

```

创建一个新的springboot工程，选择引入对应的起步依赖（mybatis、mysql驱动、lombok）



application.properties中引入数据库连接信息

提示：可以把之前项目中已有的配置信息复制过来即可

```
1 #驱动类名称
2 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
3 #数据库连接的url
4 spring.datasource.url=jdbc:mysql://localhost:3306/mybatis
5 #连接数据库的用户名
6 spring.datasource.username=root
7 #连接数据库的密码
8 spring.datasource.password=1234
```

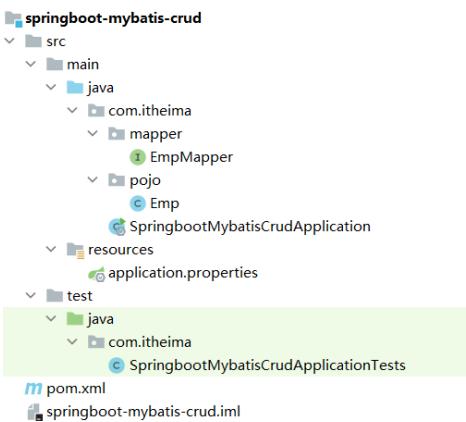
创建对应的实体类Emp (实体类属性采用驼峰命名)

```
1 @Data
2 @NoArgsConstructor
3 @AllArgsConstructor
4 public class Emp {
5     private Integer id;
6     private String username;
7     private String password;
8     private String name;
9     private Short gender;
10    private String image;
11    private Short job;
12    private LocalDate entrydate;      //LocalDate类型对应数据表中的date
13    类型
14    private Integer deptId;
15    private LocalDateTime createTime; //LocalDateTime类型对应数据表中的
16    datetime类型
17    private LocalDateTime updateTime;
18}
```

准备Mapper接口：EmpMapper

```
1 /*@Mapper注解：表示当前接口为mybatis中的Mapper接口
2 程序运行时会自动创建接口的实现类对象(代理对象)，并交给Spring的IOC容器管理
3 */
4 @Mapper
5 public interface EmpMapper {  
6
7 }
```

完成以上操作后，项目工程结构目录如下：



1.3 删除

1.3.1 功能实现

页面原型：

	姓名	用户名	性别	职位	入职日期	最后操作时间	操作
<input type="checkbox"/>	赵敏	zhaomin	女	班主任	2008-12-18	2022-07-22 12:05:20	编辑 删除
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	编辑 删除
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	编辑 删除

当我们点击后面的"删除"按钮时，前端页面会给服务端传递一个参数，也就是该行数据的ID。

我们接收到ID后，根据ID删除数据即可。

功能：根据主键删除数据

- SQL语句

```
1 -- 删除 id=17 的数据
2 delete from emp where id = 17;
```

Mybatis框架让程序员更关注于SQL语句

- 接口方法

```
1 @Mapper
2 public interface EmpMapper {
3
4     // @Delete("delete from emp where id = 17")
5     // public void delete();
```

```
6      //以上delete操作的SQL语句中的id值写成固定的17，就表示只能删除id=17的用
7      //SQL语句中的id值不能写成固定数值，需要变为动态的数值
8      //解决方案：在delete方法中添加一个参数(用户id)，将方法中的参数，传给
9      //SQL语句
10     /**
11      * 根据id删除数据
12      * @param id    用户id
13      */
14     @Delete("delete from emp where id = #{id}") //使用#{key}方式获取方
15     public void delete(Integer id);
16
17 }
```

@Delete注解：用于编写delete操作的SQL语句

如果mapper接口方法形参只有一个普通类型的参数，#{...}里面的属性名可以随便写，如：#{id}、#{value}。但是建议保持名字一致。

- 测试

- 在单元测试类中通过@Autowired注解注入EmpMapper类型对象

```
1  @SpringBootTest
2  class SpringbootMybatisCrudApplicationTests {
3      @Autowired //从Spring的IOC容器中，获取类型是EmpMapper的对象并注入
4      private EmpMapper empMapper;
5
6      @Test
7      public void testDel() {
8          //调用删除方法
9          empMapper.delete(16);
10     }
11
12 }
```

1.3.2 日志输入

在Mybatis当中我们可以借助日志，查看到sql语句的执行、执行传递的参数以及执行结果。具体操作如下：

1. 打开application.properties文件
2. 开启mybatis的日志，并指定输出到控制台

```
1 #指定mybatis输出日志的位置，输出控制台
2 mybatis.configuration.log-
  impl=org.apache.ibatis.logging.stdout.StdOutImpl
```

开启日志之后，我们再次运行单元测试，可以看到在控制台中，输出了以下的SQL语句信息：

```
11:19:10.249 INFO 44716 --- [           main] com.zaxxer.hikari.HikariDataSource      : HikariPool-1 - Starting...
11:19:11.733 INFO 44716 --- [           main] com.zaxxer.hikari.HikariDataSource      : HikariPool-1 - Start completed.
JDBC Connection [HikariProxyConnection@1673604690 wrapping com.mysql.cj.jdbc.ConnectionImpl@21bd20ee] will not be managed by Spring
=> Preparing: delete from emp where id = ?
=> Parameters: 16(Integer)
<==   Updates: 1
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@14ac77b9]
```

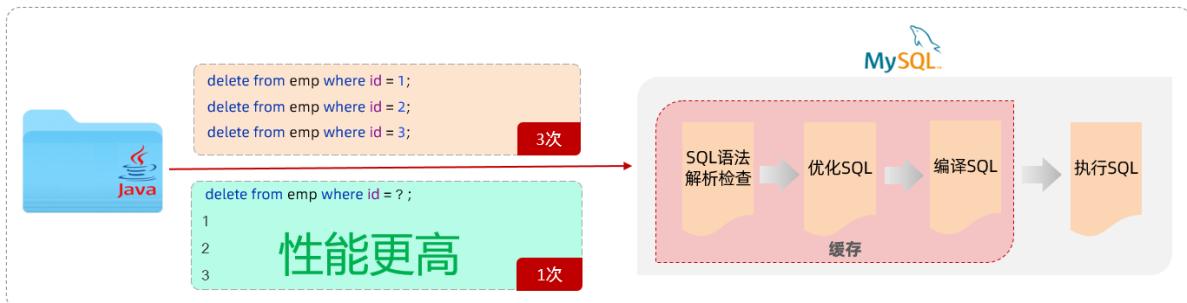
但是我们发现输出的SQL语句：delete from emp where id = ?，我们输入的参数16并没有在后面拼接，id的值是使用?进行占位。那这种SQL语句我们称为预编译SQL。

1.3.3 预编译SQL

1.3.3.1 介绍

预编译SQL有两个优势：

1. 性能更高
2. 更安全(防止SQL注入)



性能更高：预编译SQL，编译一次之后会将编译后的SQL语句缓存起来，后面再次执行这条语句时，不会再次编译。（只是输入的参数不同）

更安全(防止SQL注入)：将敏感字进行转义，保障SQL的安全性。

1.3.3.2 SQL注入

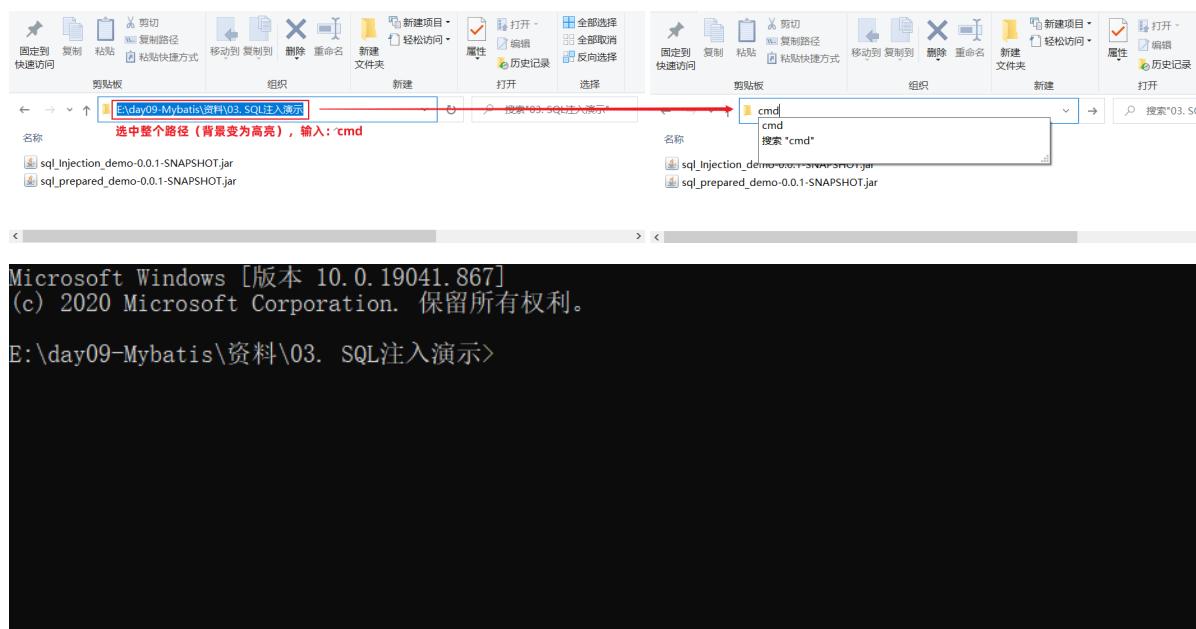
SQL注入：是通过操作输入的数据来修改事先定义好的SQL语句，以达到执行代码对服务器进行攻击的方法。

由于没有对用户输入进行充分检查，而SQL又是拼接而成，在用户输入参数时，在参数中添加一些SQL关键字，达到改变SQL运行结果的目的，也可以完成恶意攻击。

测试1：使用资料中提供的程序，来验证SQL注入问题

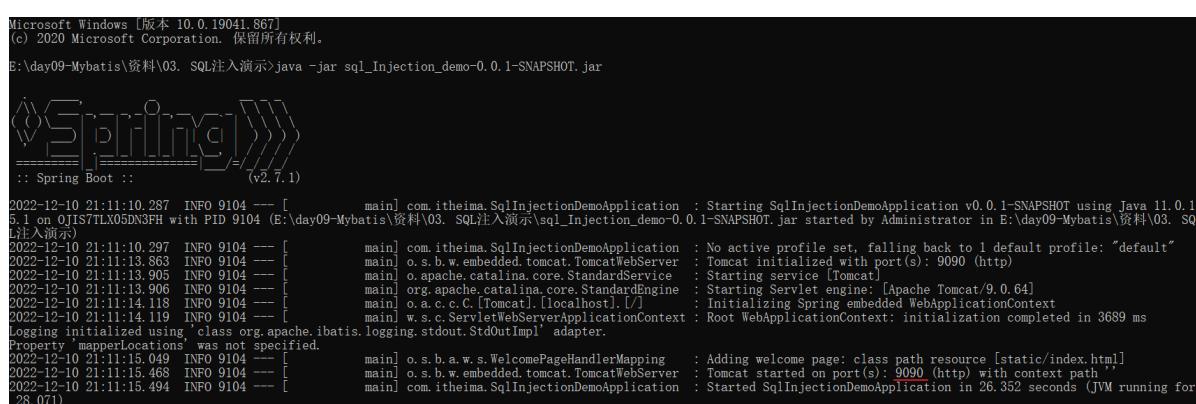
sql_Injection_demo-0.0.1-SNAPSHOT.jar → 存在SQL注入
sql_Prepared_demo-0.0.1-SNAPSHOT.jar → 已解决SQL注入

第1步：进入到DOS



第2步：执行以下命令，启动程序

```
1 #启动存在SQL注入的程序
2 java -jar sql_Injection_demo-0.0.1-SNAPSHOT.jar
```



第3步：打开浏览器输入 <http://localhost:9090/login.html>

① localhost:9090/login.html

帐号 testtest **任意帐号名**

密码 **' or '1='1** 实施SQL注入

登录

发现竟然能够登录成功：



以上操作为什么能够登录成功呢？

- 由于没有对用户输入内容进行充分检查，而SQL又是字符串拼接方式而成，在用户输入参数时，在参数中添加一些SQL关键字，达到改变SQL运行结果的目的，从而完成恶意攻击。

帐号 testtest

密码 **' or '1='1**

登录

```
select count(*) from emp where username='testtest' and password = '' or '1='1';
```

```
[IDBC Connection [HikariProxyConnection@1872194012 wrapping, com.mysql.cj.jdbc.ConnectionImpl@7a78daa5] will not be managed by Spring  
==> Preparing: select count(*) from emp where username = 'testtest' and password = '' or '1'='1'  
==> Parameters:  
<==     Columns: count(*)  
<==      Row: 17  
<==     Total: 1
```

拼接字符串

用户在页面提交数据的时候人为的添加一些特殊字符，使得sql语句的结构发生了变化，最终可以在没有用户名或者密码的情况下进行登录。

测试2：使用资料中提供的程序，来验证SQL注入问题

第1步：进入到DOS

第2步：执行以下命令，启动程序：

```
1 #启动解决了SQL注入的程序  
2 java -jar sql_prepared_demo-0.0.1-SNAPSHOT.jar
```

第3步：打开浏览器输入 <http://localhost:9090/login.html>



发现无法登录：



以上操作SQL语句的执行：

```
==> Preparing: select count(*) from emp where username = ? and password = ?
==> Parameters: testtest(String), ' or '1'='1(String)
<==   Columns: count(*)
<==     Row: 0
<==   Total: 1
```

把整个 ' or '1'='1 作为一个完整的参数，赋值给第2个问号 (' or '1'='1 进行了转义，只当做字符串使用)

1.3.3.3 参数占位符

在Mybatis中提供的参数占位符有两种：\${...}、#\${...}

- # {...}
 - 执行SQL时，会将#\${...}替换为？，生成预编译SQL，会自动设置参数值
 - 使用时机：参数传递，都使用#\${...}
- \${...}
 - 拼接SQL。直接将参数拼接在SQL语句中，存在SQL注入问题
 - 使用时机：如果对表名、列表进行动态设置时使用

注意事项：在项目开发中，建议使用#\${...}，生成预编译SQL，防止SQL注入安全。

1.4 新增

功能：新增员工信息

新增员工

*用户名	请输入用户名，2-20字符，不可重复
*员工姓名	请输入员工姓名，2-10个字
*性 别	请选择
图 像	
职 位	请选择
入职日期	2022-07-22
归属部门	请选择
<input type="button" value="保存"/> <input type="button" value="取消"/>	

1.4.1 基本新增

员工表结构：

Table:	Comment:			
emp	员工表			
Columns (11)	Keys (2)	Indexes (1)	Foreign Keys	Grants
<code>id int unsigned -- part of primary key /*ID*/</code>				
<code>username varchar(20) /*用户名*/</code>				
<code>password varchar(32) default '123456' /*密码*/</code>				
<code>name varchar(10) /*姓名*/</code>				
<code>gender tinyint unsigned /*性别, 说明: 1 男, 2 女*/</code>				
<code>image varchar(300) /*图像*/</code>				
<code>job tinyint unsigned /*职位, 说明: 1 班主任, 2 讲师, 3 学工主管, 4 教研主管, 5 咨询师*/</code>				
<code>entrydate date /*入职时间*/</code>				
<code>dept_id int unsigned /*部门ID*/</code>				
<code>create_time datetime /*创建时间*/</code>				
<code>update_time datetime /*修改时间*/</code>				

SQL语句：

```
1  insert into emp(username, name, gender, image, job, entrydate,
2    dept_id, create_time, update_time) values ('songyuanqiao','宋远
3    桥',1,'1.jpg',2,'2012-10-09',2,'2022-10-01 10:00:00','2022-10-01
4    10:00:00');
```

接口方法：

```
1  @Mapper
2  public interface EmpMapper {
3
4      @Insert("insert into emp(username, name, gender, image, job,
5        entrydate, dept_id, create_time, update_time) values (#{username}, #
6        {name}, #{gender}, #{image}, #{job}, #{entrydate}, #{deptId}, #
7        {createTime}, #{updateTime})")
8      public void insert(Emp emp);
9
10 }
```

说明：#{...} 里面写的名称是对象的属性名

测试类：

```
1  import com.itheima.mapper.EmpMapper;
2  import com.itheima.pojo.Emp;
3  import org.junit.jupiter.api.Test;
4  import org.springframework.beans.factory.annotation.Autowired;
```

```
5 import org.springframework.boot.test.context.SpringBootTest;
6 import java.time.LocalDate;
7 import java.time.LocalDateTime;
8
9 @SpringBootTest
10 class SpringbootMybatisCrudApplicationTests {
11     @Autowired
12     private EmpMapper empMapper;
13
14     @Test
15     public void testInsert(){
16         //创建员工对象
17         Emp emp = new Emp();
18         emp.setUsername("tom");
19         emp.setName("汤姆");
20         emp.setImage("1.jpg");
21         emp.setGender((short)1);
22         emp.setJob((short)1);
23         emp.setEntrydate(LocalDate.of(2000,1,1));
24         emp.setCreateTime(LocalDateTime.now());
25         emp.setUpdateTime(LocalDateTime.now());
26         emp.setDeptId(1);
27         //调用添加方法
28         empMapper.insert(emp);
29     }
30 }
31
```

日志输出：

```
==> Preparing: insert into emp(username, name, gender, image, job, entrydate, dept_id, create_time, update_time) values (?, ?, ?, ?, ?, ?, ?, ?, ?)
==> Parameters: tom(String), 汤姆(String), 1(Short), 1.jpg(String), 1(Short), 2000-01-01(LocalDate), 1(Integer), 2022-12-11T14:01:39.976077700(LocalDateTime)
<==    Updates: 1
```

1.4.2 主键返回

概念：在数据添加成功后，需要获取插入数据库数据的主键。

如：添加套餐数据时，还需要维护套餐菜品关系表数据。

新建套餐

*套餐名称:

*套餐分类:

*套餐价格: 元

*套餐菜品: **1 添加菜品** 该套餐关联的菜品信息 -> 套餐菜品关系表

菜品名称	原价	份数	操作
鱼香肉丝	¥44.00	- 1 +	删除

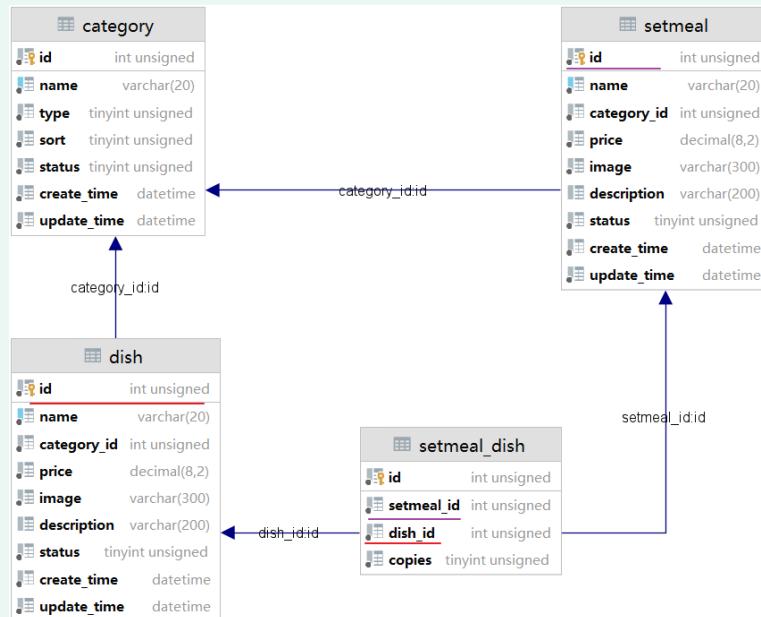
*套餐图片: 图片大小不超过2M 仅能上传 PNG JPEG JPG类型图片

套餐描述: 菜品描述, 最长200字

保存 **保存并继续添加菜品** **返回**

1. 先保存套餐信息, 并获取套餐ID。
2. 然后再保存套餐菜品关联信息(需要记录套餐ID、菜品ID)

业务场景：在前面讲解到的苍穹外卖菜品与套餐模块的表结构，菜品与套餐是多对多的关系，一个套餐对应多个菜品。既然是多对多的关系，是不是有一张套餐菜品中间表来维护它们之间的关系。



在添加套餐的时候，我们需要在界面当中来录入套餐的基本信息，还需要来录入套餐与菜品的关联信息。这些信息录入完毕之后，我们一点保存，就需要将套餐的信息以及套餐与菜品的关联信息都需要保存到数据库当中。其实具体的过程包括两步，首先第一步先需要将套餐的基本信息保存了，接下来第二步再来保存套餐与菜品的关联信息。套餐与菜品的关联信息就是往中间表当中来插入数据，来维护它们之间的关系。而中间表当中有两个外键字段，一个是菜品的ID，就是当前菜品的ID，还有一个就是套餐的ID，而这个套餐的 ID 指的就是此次我所添加的套餐的ID，所以在第一步保存完套餐的基本信息之后，就需要将套餐的主键值返回来供第二步进行使用。这个时候就需要用到主键返回功能。

那要如何实现在插入数据之后返回所插入行的主键值呢？

- 默认情况下，执行插入操作时，是不会主键值返回的。如果我们想要拿到主键值，需要在Mapper接口中的方法上添加一个Options注解，并在注解中指定属性useGeneratedKeys=true和keyProperty="实体类属性名"

主键返回代码实现：

```
1  @Mapper
2  public interface EmpMapper {
3
4      //会自动将生成的主键值，赋值给emp对象的id属性
5      @Options(useGeneratedKeys = true, keyProperty = "id")
6      @Insert("insert into emp(username, name, gender, image, job,
entrydate, dept_id, create_time, update_time) values (#{username}, #
{name}, #{gender}, #{image}, #{job}, #{entrydate}, #{deptId}, #
{createTime}, #{updateTime})")
7      public void insert(Emp emp);
8
9  }
```

测试：

```
1  @SpringBootTest
2  class SpringbootMybatisCrudApplicationTests {
3
4      @Autowired
5      private EmpMapper empMapper;
6
7      @Test
8      public void testInsert(){
9          //创建员工对象
10         Emp emp = new Emp();
11         emp.setUsername("jack");
12         emp.setName("杰克");
13         emp.setImage("1.jpg");
14         emp.setGender((short)1);
15         emp.setJob((short)1);
16         emp.setEntrydate(LocalDate.of(2000,1,1));
17         emp.setCreateTime(LocalDateTime.now());
18         emp.setUpdateTime(LocalDateTime.now());
19         emp.setDeptId(1);
20         //调用添加方法
21         empMapper.insert(emp);
22
23         System.out.println(emp.getDeptId());
24     }
```

1.5 更新

功能：修改员工信息

点击“编辑”按钮后，会查询所在行记录的员工信息，并把员工信息回显在修改员工的窗体上（下个知识点学习）

在修改员工的窗体上，可以修改的员工数据：用户名、员工姓名、性别、图像、职位、入职日期、归属部门

思考：在修改员工数据时，要以什么做为条件呢？

答案：员工id

SQL语句：

```
1 update emp set username = 'linghushaoxia', name = '令狐少侠', gender =
1 , image = '1.jpg' , job = 2, entrydate = '2012-01-01', dept_id = 2,
update_time = '2022-10-01 12:12:12' where id = 18;
```

接口方法：

```
1  @Mapper
2  public interface EmpMapper {
3      /**
4       * 根据id修改员工信息
5       * @param emp
6      */
7      @Update("update emp set username=#{username}, name=#{name},
8 gender=#{gender}, image=#{image}, job=#{job}, entrydate=#
9 {entrydate}, dept_id=#{deptId}, update_time=#{updateTime} where id=#
10 {id}")
11     public void update(Emp emp);
12 }
13 }
```

测试类：

```
1  @SpringBootTest
2  class SpringbootMybatisCrudApplicationTests {
3      @Autowired
4      private EmpMapper empMapper;
5
6      @Test
7      public void testUpdate() {
8          //要修改的员工信息
9          Emp emp = new Emp();
10         emp.setId(23);
11         emp.setUsername("songdaxia");
12         emp.setPassword(null);
13         emp.setName("老宋");
14         emp.setImage("2.jpg");
15         emp.setGender((short)1);
16         emp.setJob((short)2);
17         emp.setEntrydate(LocalDate.of(2012,1,1));
18         emp.setCreateTime(null);
19         emp.setUpdateTime(LocalDateTime.now());
20         emp.setDeptId(2);
21         //调用方法，修改员工数据
22         empMapper.update(emp);
23     }
24 }
```

1.6 查询

1.6.1 根据ID查询

在员工管理的页面中，当我们进行更新数据时，会点击“编辑”按钮，然后此时会发送一个请求到服务端，会根据ID查询该员工信息，并将员工数据回显在页面上。

	姓名	用户名	性别	职位	入职日期	最后操作时间	操作
<input type="checkbox"/>	赵敏	zhaomin	女	班主任	2008-12-18	2022-07-22 12:05:20	编辑 删除
<input type="checkbox"/>	凤清扬	fengqingshang	男	讲师	2015-07-22	2022-07-21 15:00:00	编辑 删除

修改员工

* 用户名 zhaomin
* 员工姓名 赵敏
* 性 别 女
图 像
职 位 班主任
入 职 日 期 2008-12-18
归 属 部 门 学工部
保存 取消

SQL语句：

```
1 select id, username, password, name, gender, image, job, entrydate,  
      dept_id, create_time, update_time from emp;
```

接口方法：

```
1 @Mapper  
2 public interface EmpMapper {  
3     @Select("select id, username, password, name, gender, image, job,  
          entrydate, dept_id, create_time, update_time from emp where id=#{id}")  
4     public Emp getById(Integer id);  
5 }
```

测试类：

```

1  @SpringBootTest
2  class SpringbootMybatisCrudApplicationTests {
3      @Autowired
4      private EmpMapper empMapper;
5
6      @Test
7      public void test GetById() {
8          Emp emp = empMapper.getById(1);
9          System.out.println(emp);
10     }
11 }

```

执行结果：

```

non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@4da1f38a]
1, username=jinyong, password=123456, name=金庸, gender=1, image=1.jpg, job=4, entrydate=2000-01-01, deptId=null, createTime=null, updateTime=null)

```

而在测试的过程中，我们会发现有几个字段(deptId、createTime、updateTime)是没有数据值的

1.6.2 数据封装

我们看到查询返回的结果中大部分字段是有值的，但是deptId、createTime、updateTime这几个字段是没有值的，而数据库中是有对应的字段值的，这是为什么呢？



原因如下：

- 实体类属性名和数据库表查询返回的字段名一致，mybatis会自动封装。
- 如果实体类属性名和数据库表查询返回的字段名不一致，不能自动封装。

解决方案：

1. 起别名
2. 结果映射
3. 开启驼峰命名

起别名: 在SQL语句中，对不一样的列名起别名，别名和实体类属性名一样

```
1  @Select("select id, username, password, name, gender, image, job,
2          entrydate, " +
3          "dept_id AS deptId, create_time AS createTime, update_time AS
4          updateTime " +
5          "from emp " +
6          "where id=#{id}")
7  public Emp getById(Integer id);
```

再次执行测试类:

```
>> Tests passed: 1 of 1 test - 3 sec 69 ms
    ↓
<==  Columns: id, username, password, name, gender, image, job, entrydate, deptId, createTime, updateTime
<==      Row: 1, jinyong, 123456, 金庸, 1, 1.jpg, 4, 2000-01-01, 2, 2022-12-10 21:18:55, 2022-12-10 21:18:55
<==      Total: 1
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@7061622]
Emp(id=1, username=jinyong, password=123456, name=金庸, gender=1, image=1.jpg, job=4, entrydate=2000-01-01, deptId=2, createTime=2022-12-10 21:18:55)
```

手动结果映射: 通过 `@Results` 及 `@Result` 进行手动结果映射

```
1  @Results({@Result(column = "dept_id", property = "deptId"),
2             @Result(column = "create_time", property = "createTime"),
3             @Result(column = "update_time", property = "updateTime")})
4  @Select("select id, username, password, name, gender, image, job,
5          entrydate, dept_id, create_time, update_time from emp where id=#{id}")
6  public Emp getById(Integer id);
```

`@Results` 源代码:

```
1  @Documented
2  @Retention(RetentionPolicy.RUNTIME)
3  @Target({ElementType.METHOD})
4  public @interface Results {
5      String id() default "";
6
7      Result[] value() default {};//Result类型的数组
8  }
```

`@Result` 源代码:

```
1  @Documented
```

```
2  @Retention(RetentionPolicy.RUNTIME)
3  @Target({ElementType.METHOD})
4  @Repeatable(Results.class)
5  public @interface Result {
6      boolean id() default false; //表示当前列是否为主键 (true:是主键)
7
8      String column() default ""; //指定表中字段名
9
10     String property() default ""; //指定类中属性名
11
12     Class<?> javaType() default void.class;
13
14     JdbcType jdbcType() default JdbcType.UNDEFINED;
15
16     Class<? extends TypeHandler> typeHandler() default
17         UnknownTypeHandler.class;
18
19     One one() default @One;
20
21     Many many() default @Many;
22 }
```

开启驼峰命名(推荐): 如果字段名与属性名符合驼峰命名规则, mybatis会自动通过驼峰命名规则映射

驼峰命名规则: abc_xyz => abcXyz

- 表中字段名: abc_xyz
- 类中属性名: abcXyz

```
1  # 在application.properties中添加:
2  mybatis.configuration.map-underscore-to-camel-case=true
```

要使用驼峰命名前提是 实体类的属性 与 数据库表中的字段名严格遵守驼峰命名。

1.6.3 条件查询

在员工管理的列表页面中，我们需要根据条件查询员工信息，查询条件包括：姓名、性别、入职时间。

The screenshot shows a web-based employee management system. At the top, there are search filters: '姓名' (Name) with input '张', '性别' (Gender) with dropdown '男', and '入职时间' (Entry Date) with range '2010-01-01' to '2020-01-01'. Below the filters are two buttons: '+ 新增员工' (Add New Employee) and '- 批量删除' (Batch Delete). The main area displays a table of employee records. One row is highlighted with a pink background, labeled '页面开发规则' (Development Rules). The table columns are: 姓名 (Name), 用户名 (Username), 性别 (Gender), 职位 (Position), 入职日期 (Entry Date), 最后操作时间 (Last Operation Time), and 操作 (Operations). The data in the table includes multiple entries for '风清扬' (fengqingyang) with various gender and position details.

姓名	用户名	性别	职位	入职日期	最后操作时间	操作
赵敏	zhaomin	女	班主任			
风清扬	fengqingyang	男	讲师			
风清扬	fengqingyang	男	讲师			
风清扬	fengqingyang	男	讲师			
风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	编辑 删除
风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	编辑 删除
风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	编辑 删除
风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	编辑 删除
风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	编辑 删除

通过页面原型以及需求描述我们要实现的查询：

- 姓名：要求支持模糊匹配
- 性别：要求精确匹配
- 入职时间：要求进行范围查询
- 根据最后修改时间进行降序排序

SQL语句：

```
1 select id, username, password, name, gender, image, job, entrydate,
2     dept_id, create_time, update_time
3 from emp
4 where name like '%张%'
5     and gender = 1
6     and entrydate between '2010-01-01' and '2020-01-01'
7 order by update_time desc;
```

接口方法：

- 方式一

```

1  @Mapper
2  public interface EmpMapper {
3      @Select("select * from emp " +
4          "where name like '%${name}%' " +
5          "and gender = #{gender} " +
6          "and entrydate between #{begin} and #{end} " +
7          "order by update_time desc")
8      public List<Emp> list(String name, Short gender, LocalDate begin,
9          LocalDate end);

```

`@Select("select * from emp where name like '%${name}%' and gender = #{gender} and entrydate between #{begin} and #{end} order by update_time desc")`
`public List<Emp> list(String name, Short gender, LocalDate begin, LocalDate end);`

性能低、不安全、存在SQL注入问题

以上方式注意事项：

- 方法中的形参名和SQL语句中的参数占位符名保持一致
- 模糊查询使用`{} . . . {}`进行字符串拼接，这种方式呢，由于是字符串拼接，并不是预编译的形式，所以效率不高、且存在sql注入风险。

- 方式二（解决SQL注入风险）

- 使用MySQL提供的字符串拼接函数：`concat('%', '关键字', '%')`

```

1  @Mapper
2  public interface EmpMapper {
3
4      @Select("select * from emp " +
5          "where name like concat('%',#{name},'%') " +
6          "and gender = #{gender} " +
7          "and entrydate between #{begin} and #{end} " +
8          "order by update_time desc")
9      public List<Emp> list(String name, Short gender, LocalDate
10         begin, LocalDate end);
11
12

```

执行结果：生成的SQL都是预编译的SQL语句（性能高、安全）

```

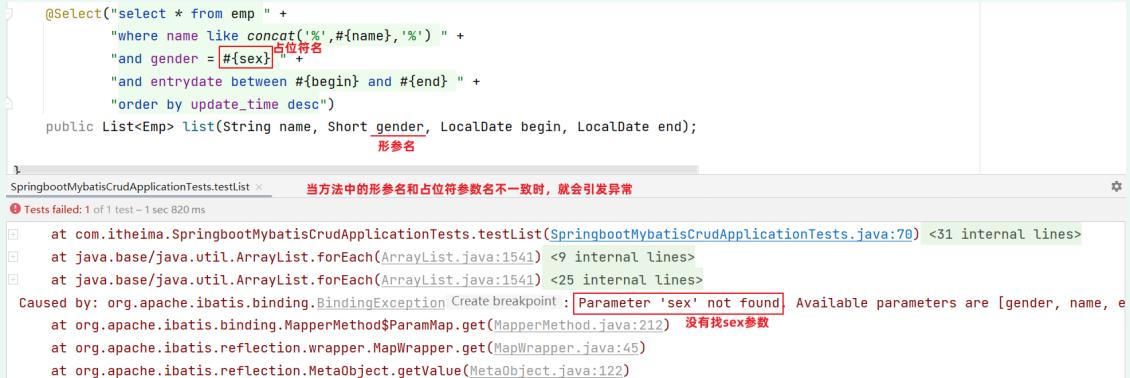
 JDBC Connection [ProxyConnection@47853943 wrapping com.mysql.cj.jdbc.ConnectionImpl@42700142] will not be managed by Spring
 => Preparing: select * from emp where name like concat('%',?, '%') and gender = ? and entrydate between ? and ? order by update_time desc
 => Parameters: 张(String), 1(Short), 2010-01-01(LocalDate), 2020-01-01(LocalDate)
 <==   Columns: id, username, password, name, gender, image, job, entrydate, dept_id, create_time, update_time
 <==     Row: 2, zhangwaji, 123456, 张无忌, 1, 2.jpg, 2, 2015-01-01, 2, 2022-12-10 21:18:55, 2022-12-10 21:18:55
 <==   Total: 1
 Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@5d1b1c2a]
 Emp(id=2, username=zhangwaji, password=123456, name=张无忌, gender=1, image=2.jpg, job=2, entrydate=2015-01-01, deptId=2, createTime=2022-12-10 21:18:55)

```

1.6.4 参数名说明

在上面我们所编写的条件查询功能中，我们需要保证接口中方法的形参名和SQL语句中的参数占位符名相同。

当方法中的形参名和SQL语句中的占位符参数名不相同时，就会出现以下问题：



The screenshot shows a Java code editor with a test class `SpringbootMybatisCrudApplicationTests` containing a method `testList`. The code includes a SQL query with placeholder `#{sex}` and a parameter `gender`. A red box highlights the placeholder `#{sex}` with the label "占位符名" (Placeholder Name) and the parameter name `gender` with the label "形参名" (Parameter Name). Below the code, an error message from the IDE indicates a test failure: "当方法中的形参名和占位符参数名不一致时，就会引发异常". The stack trace shows the exception was caused by a `BindingException` due to the parameter 'sex' not being found, while available parameters are [gender, name, e].

参数名在不同的SpringBoot版本中，处理方案还不同：

- 在springBoot的2.x版本（保证参数名一致）

```
@Select("select * from emp where name like concat('%',#{name},'%') and gender = #{gender} and " +  
        "entrydate between #{begin} and #{end} order by update_time desc")  
public List<Emp> list(String name, Short gender, LocalDate begin, LocalDate end);
```

springBoot的父工程对compiler编译插件进行了默认的参数parameters配置，使得在编译时，会在生成的字节码文件中保留原方法形参的名称，所以#{...}里面可以直接通过形参名获取对应的值

```
@Select({"select * from emp where name like concat('%',#{name},'%') and gender = #{gender}"})  
List<Emp> list(String name, Short gender, LocalDate begin, LocalDate end);
```

- 在springBoot的1.x版本/单独使用mybatis（使用@Param注解来指定SQL语句中的参数名）

```
@Select("select * from emp where name like concat('%',#{name},'%') and gender = #{gender} and " +  
        "entrydate between #{begin} and #{end} order by update_time desc")  
public List<Emp> list(@Param("name")String name, @Param("gender")Short gender,  
                      @Param("begin")LocalDate begin, @Param("end")LocalDate end);
```

在编译时，生成的字节码文件当中，不会保留Mapper接口中方法的形参名称，而是使用var1、var2、...这样的形参名字，此时要获取参数值时，就要通过@Param注解来指定SQL语句中的参数名

```
@Select({"select * from emp where name like concat('%',#{name},'%') and gender = #{gender}")
List<Emp> list(String var1, Short var2, LocalDate var3, LocalDate var4);
```

2. Mybatis的XML配置文件

Mybatis的开发有两种方式：

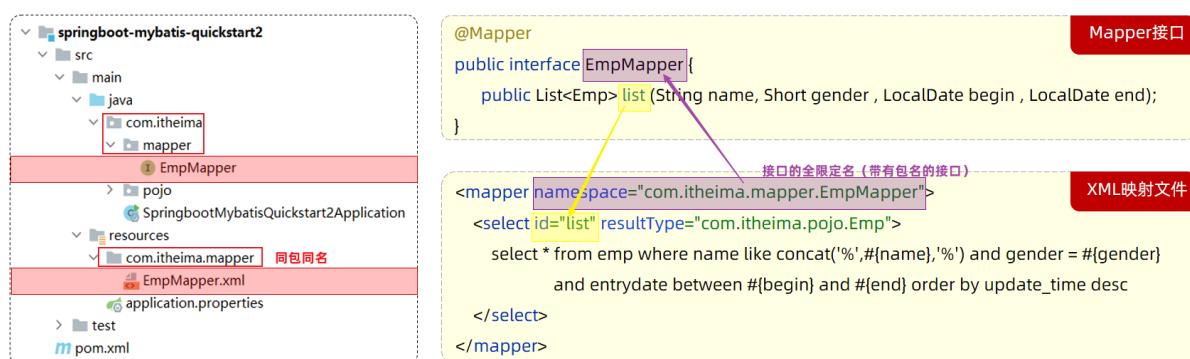
1. 注解
2. XML

2.1 XML配置文件规范

使用Mybatis的注解方式，主要是来完成一些简单的增删改查功能。如果需要实现复杂的SQL功能，建议使用XML来配置映射语句，也就是将SQL语句写在XML配置文件中。

在Mybatis中使用XML映射文件方式开发，需要符合一定的规范：

1. XML映射文件的名称与Mapper接口名称一致，并且将XML映射文件和Mapper接口放置在相同包下（同包同名）
2. XML映射文件的namespace属性为Mapper接口全限定名一致
3. XML映射文件中sql语句的id与Mapper接口中的方法名一致，并保持返回类型一致。

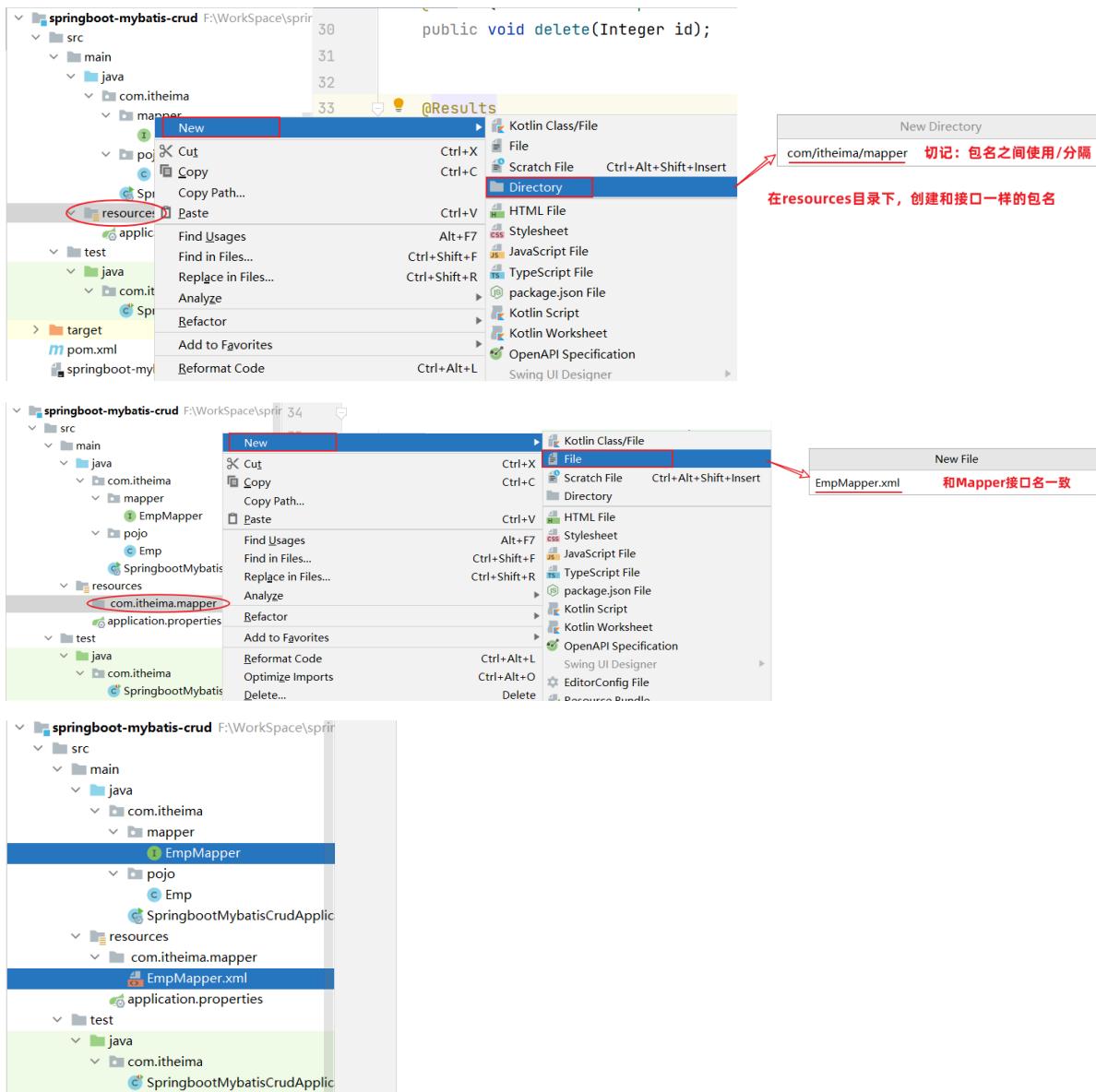


<select>标签：就是用于编写select查询语句的。

- resultType属性，指的是查询返回的单条记录所封装的类型。

2.2 XML配置文件实现

第1步：创建XML映射文件



第2步：编写XML映射文件

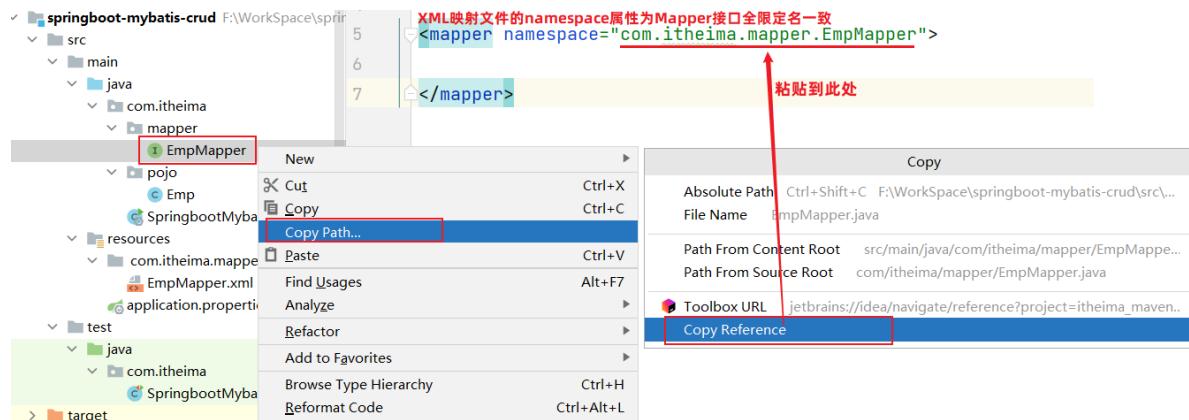
xml映射文件中的dtd约束，直接从mybatis官网复制即可

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3   PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4   "https://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace="">
6
7 </mapper>

```

配置：XML映射文件的namespace属性为Mapper接口全限定名

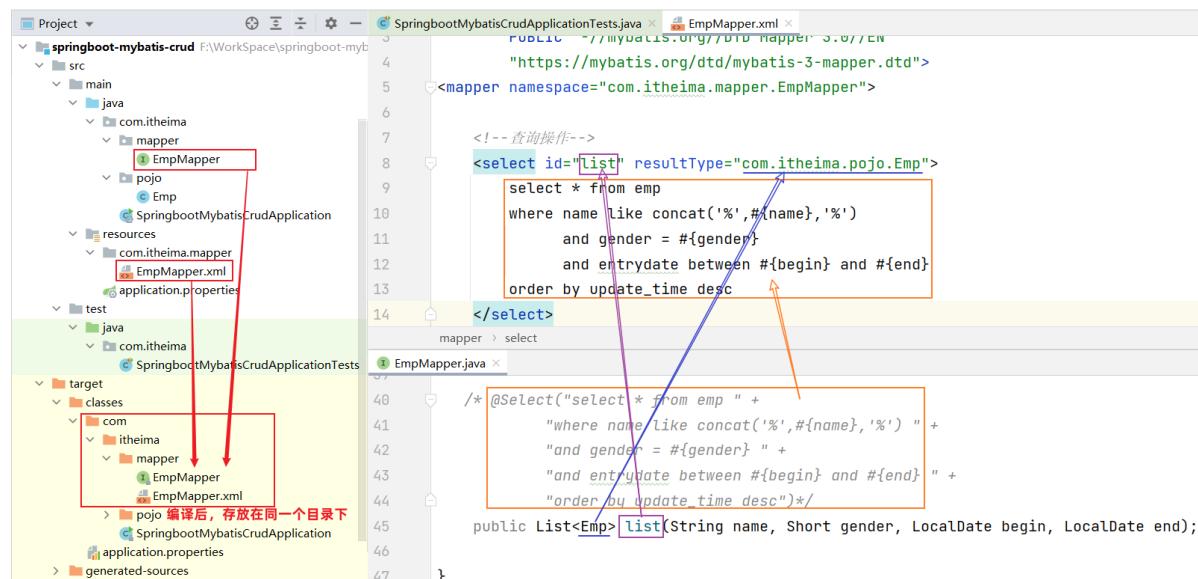


```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3   PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4   "https://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace="com.itheima.mapper.EmpMapper">
6
7 </mapper>

```

配置：XML映射文件中sql语句的id与Mapper接口中的方法名一致，并保持返回类型一致



```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE mapper
3      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4      "https://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <mapper namespace="com.itheima.mapper.EmpMapper">
6
7      <!--查询操作-->
8      <select id="list" resultType="com.itheima.pojo.Emp">
9          select * from emp
10         where name like concat('%',#{name},'%')
11             and gender = #{gender}
12             and entrydate between #{begin} and #{end}
13         order by update_time desc
14     </select>
15 </mapper>

```

运行测试类，执行结果：

```

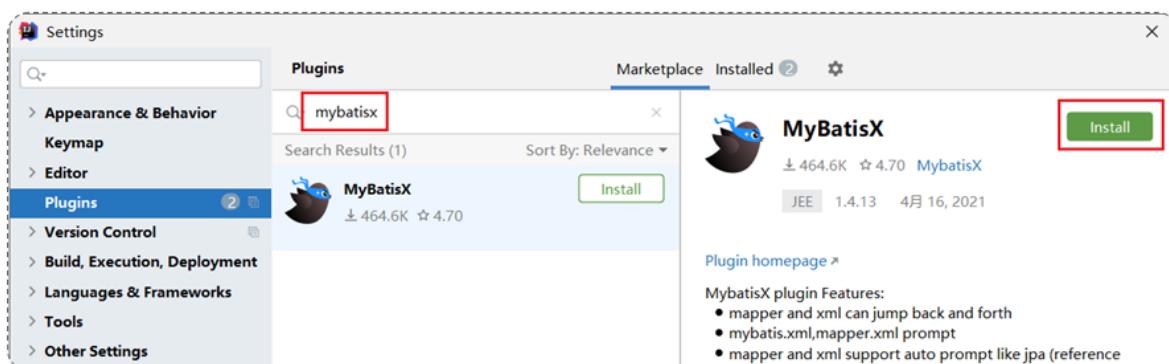
JDBC Connection [HikariProxyConnection@378797968 wrapping com.mysql.cj.jdbc.ConnectionImpl@47058864] will not be managed by Spring
==> Preparing: select * from emp where name like concat('%',?, '%') and gender = ? and entrydate between ? and ? order by update_time desc
==> Parameters: 张(String), 1(Short), 2010-01-01(LocalDate), 2020-01-01(LocalDate)
<==   Columns: id, username, password, name, gender, image, job, entrydate, dept_id, create_time, update_time
<==   Row: 2, zhangwuji, 123456, 张无忌, 1, 2.jpg, 2, 2015-01-01, 2, 2022-12-10 21:18:55, 2022-12-10 21:18:55
<==   Total: 1
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@626d2016]
Emp(id=2, username=zhangwuji, password=123456, name=张无忌, gender=1, image=2.jpg, job=2, entrydate=2015-01-01, deptId=2, createTime=2022-1

```

2.3 MybatisX的使用

MybatisX是一款基于IDEA的快速开发Mybatis的插件，为效率而生。

MybatisX的安装：



可以通过MybatisX快速定位：

```

29     @Delete("delete from emp where id = #{id}")
30     public void delete(Integer id);
31
32     @Select("select id, username, password, name, gender, image, dept_id AS deptId, create_time AS createTime, update_time AS updateTime from emp where id=#{id}")
33     public Emp getById(Integer id);
34
35     /* @Select("select * from emp where name like concat('%',#{name},'%') " +
36     "and gender = #{gender} " +
37     "and entrydate between #{begin} and #{end} " +
38     "order by update_time desc") */
39     public List<Emp> list(String name, Short gender, LocalDate begin, LocalDate end);
40
41     // 点击定位到映射文件中的SQL语句
42 }

```

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3   PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4   "https://mybatis.org/dtd/mybatis-3-mapper.dtd">
5   <mapper namespace="com.itheima.mapper.EmpMapper">
6     <!-- 查询操作-->
7     <select id="list" resultType="com.itheima.pojo.Emp">
8       select * from emp
9       where name like concat('%',#{name},'%')
10      and gender = #{gender}
11      and entrydate between #{begin} and #{end}
12      order by update_time desc
13    </select>
14  </mapper>
15

```

MybatisX的使用在后续学习中会继续分享

学习了Mybatis中XML配置文件的开发方式了，大家可能会存在一个疑问：到底是使用注解方式开发还是使用XML方式开发？

官方说明：<https://mybatis.net.cn/getting-started.html>

```

package org.mybatis.example;
public interface BlogMapper {
    @Select("SELECT * FROM blog WHERE id = #{id}")
    Blog selectBlog(int id);
}

```

使用注解来映射简单语句会使代码显得更加简洁，但对于稍微复杂一点的语句，Java 注解不仅力不从心，还会让你本就复杂的 SQL 语句更加混乱不堪。因此，如果你需要做一些很复杂的操作，最好用 XML 来映射语句。

选择何种方式来配置映射，以及认为是否应该要统一映射语句定义的形式，完全取决于你和你的团队。换句话说，永远不要拘泥于一种方式，你可以很轻松的在基于注解和 XML 的语句映射方式间自由移植和切换。

结论：使用Mybatis的注解，主要是来完成一些简单的增删改查功能。如果需要实现复杂的SQL功能，建议使用XML来配置映射语句。

3. Mybatis动态SQL

3.1 什么是动态SQL

在页面原型中，列表上方的条件是动态的，是可以不传递的，也可以只传递其中的1个或者2个或者全部。

姓名	<input type="text" value="请输入员工姓名"/>	性别	<input type="text" value="请选择"/>	入职时间 从	<input type="text" value="开始时间"/> 至 <input type="text" value="结束时间"/>	<input type="button" value=""/>	<input type="button" value="查询"/>
<input type="button" value="新增员工"/>		<input type="button" value="批量删除"/>					
<input type="checkbox"/>	姓名	用户名	性别	职位	入职日期	最后操作时间	操作
<input type="checkbox"/>	赵敏	zhaomin	女	班主任	2008-12-18	2022-07-22 12:05:20	<input type="button" value="编辑"/> <input type="button" value="删除"/>
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	<input type="button" value="编辑"/> <input type="button" value="删除"/>
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	<input type="button" value="编辑"/> <input type="button" value="删除"/>
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	<input type="button" value="编辑"/> <input type="button" value="删除"/>
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	<input type="button" value="编辑"/> <input type="button" value="删除"/>
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	<input type="button" value="编辑"/> <input type="button" value="删除"/>
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	<input type="button" value="编辑"/> <input type="button" value="删除"/>
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	<input type="button" value="编辑"/> <input type="button" value="删除"/>
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	2022-07-21 15:00:00	<input type="button" value="编辑"/> <input type="button" value="删除"/>

```
@Select("select * from emp where name like concat('%',#{name},'%') and gender = #{gender} and entrydate between #{begin} and #{end} order by update_time desc")
public List<Emp> list(String name, Short gender, LocalDate begin, LocalDate end);
```

而在我们刚才编写的SQL语句中，我们会看到，我们将三个条件直接写死了。如果页面只传递了参数姓名name字段，其他两个字段 性别 和 入职时间没有传递，那么这两个参数的值就是null。

此时，执行的SQL语句为：

```
JDBC Connection [HikariProxyConnection@1296153103 wrapping com.mysql.cj.jdbc.ConnectionImpl@3c87e6b7] will not be managed by Spring
==> Preparing: select * from emp where name like concat('%',?, '%') and gender = ? and entrydate between ? and ? order by update_time desc
==> Parameters: 张(String), null, null, null
<==   Total: 0
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@50a691d3]
```

这个查询结果是不正确的。正确的做法应该是：传递了参数，再组装这个查询条件；如果没有传递参数，就不应该组装这个查询条件。

比如：如果姓名输入了"张"，对应的SQL为：

```
1  select * from emp where name like '%张%' order by update_time desc;
```

如果姓名输入了"张"，性别选择了"男"，则对应的SQL为：

```
1  select * from emp where name like '%张%' and gender = 1 order by
update_time desc;
```

SQL语句会随着用户的输入或外部条件的变化而变化，我们称为：**动态SQL**。

```

<select id="list" resultType="com.itheima.pojo.Emp">
    select id, username, password, name, gender, image, job,
           entrydate, dept_id, create_time, update_time from emp
    where
        <if test="name != null">
            name like concat('%',#{name},'%')
        </if>
        <if test="gender != null">
            and gender = #{gender}
        </if>
        <if test="begin != null and end != null">
            and entrydate between #{begin} and #{end}
        </if>
    order by update_time desc
</select>

```

在Mybatis中提供了很多实现动态SQL的标签，我们学习Mybatis中的动态SQL就是掌握这些动态SQL标签。

3.2 动态SQL-if

<if>：用于判断条件是否成立。使用test属性进行条件判断，如果条件为true，则拼接SQL。

```

1   <if test="条件表达式">
2       要拼接的sql语句
3   </if>

```

接下来，我们就通过**<if>**标签来改造之前条件查询的案例。

3.2.1 条件查询

示例：把SQL语句改造为动态SQL方式

- 原有的SQL语句

```

1   <select id="list" resultType="com.itheima.pojo.Emp">
2       select * from emp
3       where name like concat('%',#{name},'%')
4           and gender = #{gender}
5           and entrydate between #{begin} and #{end}
6       order by update_time desc
7   </select>

```

- 动态SQL语句

```

1   <select id="list" resultType="com.itheima.pojo.Emp">
2       select * from emp
3       where

```

```
4
5          <if test="name != null">
6              name like concat('%',#{name},'%')
7          </if>
8          <if test="gender != null">
9              and gender = #{gender}
10         </if>
11         <if test="begin != null and end != null">
12             and entrydate between #{begin} and #{end}
13         </if>
14
15     order by update_time desc
16 </select>
```

测试方法：

```
1 @Test
2 public void testList(){
3     //性别数据为null、开始时间和结束时间也为null
4     List<Emp> list = empMapper.list("张", null, null, null);
5     for(Emp emp : list){
6         System.out.println(emp);
7     }
8 }
```

执行的SQL语句：

```
JDBC Connection [HikariProxyConnection@384916201 wrapping com.mysql.cj.jdbc.ConnectionImpl@6206b4a7] will not be
==> Preparing: select * from emp where name like concat('%',?, '%') order by update_time desc
==> Parameters: 张(String) 只拼接了name条件，其他为null的则没有拼接SQL
<==   Columns: id, username, password, name, gender, image, job, entrydate, dept_id, create_time, update_time
```

下面呢，我们修改测试方法中的代码，再次进行测试，观察执行情况：

```
1 @Test
2 public void testList(){
3     //姓名为null
4     List<Emp> list = empMapper.list(null, (short)1, null, null);
5     for(Emp emp : list){
6         System.out.println(emp);
7     }
8 }
```

执行结果：

The screenshot shows a JUnit test run with one test failure. The error message is: "Tests failed: 1 of 1 test - 2 sec 163 ms" and "error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'and gender = 1'". Below the error message, there is a stack trace:

```

    at org.apache.ibatis.executor.SimpleExecutor.prepareStatement(SimpleExecutor.java:87)
    at org.apache.ibatis.executor.SimpleExecutor.doQuery(SimpleExecutor.java:61)
    at org.apache.ibatis.session.defaults.DefaultSqlSession.selectList(DefaultSqlSession.java:105)
    at org.apache.ibatis.session.defaults.DefaultSqlSession.selectList(DefaultSqlSession.java:92)
    at com.itheima.mapper.EmpMapper.list(EmpMapper.java:35)
    at com.itheima.controller.EmpController.list(EmpController.java:44)

```

Another screenshot shows the JDBC connection details: "JDBC Connection [HikariProxyConnection@1561651614 wrapping com.mysql.cj.jdbc.ConnectionImpl@38cfecf3] will not be managed by Spring" and the prepared statement: "==> Preparing: select * from emp where and gender = ? order by update_time desc". It also notes "Parameters: 1(short)" and "多出一个and关键字".

再次修改测试方法中的代码，再次进行测试：

```

1  @Test
2  public void testList() {
3      //传递的数据全部为null
4      List<Emp> list = empMapper.list(null, null, null, null);
5      for(Emp emp : list) {
6          System.out.println(emp);
7      }
8  }

```

执行的SQL语句：

The screenshot shows a JUnit test run with one test failure. The error message is: "Tests failed: 1 of 1 test - 3 sec 164 ms" and "error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'where'". Below the error message, there is a stack trace:

```

    at org.apache.ibatis.executor.SimpleExecutor.prepareStatement(SimpleExecutor.java:87)
    at org.apache.ibatis.executor.SimpleExecutor.doQuery(SimpleExecutor.java:61)
    at org.apache.ibatis.session.defaults.DefaultSqlSession.selectList(DefaultSqlSession.java:105)
    at org.apache.ibatis.session.defaults.DefaultSqlSession.selectList(DefaultSqlSession.java:92)
    at com.itheima.mapper.EmpMapper.list(EmpMapper.java:35)
    at com.itheima.controller.EmpController.list(EmpController.java:44)

```

Another screenshot shows the JDBC connection details: "JDBC Connection [HikariProxyConnection@1561651614 wrapping com.mysql.cj.jdbc.ConnectionImpl@38cfecf3] will not be managed by Spring" and the prepared statement: "==> Preparing: select * from emp where order by update_time desc". It also notes "Parameters: 1" and "多出一个where关键字".

以上问题的解决方案：使用 `<where>` 标签代替SQL语句中的where关键字

- `<where>` 只会在子元素有内容的情况下才插入where子句，而且会自动去除子句的开头的AND或OR

```

1  <select id="list" resultType="com.itheima.pojo.Emp">
2      select * from emp
3      <where>
4          <!-- if做为where标签的子元素 -->
5          <if test="name != null">
6              and name like concat('%',#{name},'%')
7          </if>
8          <if test="gender != null">
9              and gender = #{gender}
10         </if>
11         <if test="begin != null and end != null">
12             and entrydate between #{begin} and #{end}

```

```
13             </if>
14         </where>
15         order by update_time desc
16     </select>
```

测试方法：

```
1  @Test
2  public void testList(){
3      //只有性别
4      List<Emp> list = empMapper.list(null, (short)1, null, null);
5      for(Emp emp : list){
6          System.out.println(emp);
7      }
8  }
```

执行的SQL语句：

```
JDBC Connection [HikariProxyConnection@732918485 wrapping com.mysql.cj.jdbc.ConnectionImpl@66756662] will not be managed by Spring
==> Preparing: select * from emp WHERE gender = ? order by update_time desc
==> Parameters: 1(Short)
<==    Columns: id, username, password, name, gender, image, job, entrydate, dept_id, create_time, update_time
```

3.2.2 更新员工

案例：完善更新员工功能，修改为动态更新员工数据信息

- 动态更新员工信息，如果更新时传递有值，则更新；如果更新时没有传递值，则不更新
- 解决方案：动态SQL

修改Mapper接口：

```
1  @Mapper
2  public interface EmpMapper {
3      //删除@Update注解编写的SQL语句
4      //update操作的SQL语句编写在Mapper映射文件中
5      public void update(Emp emp);
6  }
```

修改Mapper映射文件：

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE mapper
3      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4      "https://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <mapper namespace="com.itheima.mapper.EmpMapper">
```

```
6
7      <!--更新操作-->
8      <update id="update">
9          update emp
10         set
11             <if test="username != null">
12                 username=#{username},
13             </if>
14             <if test="name != null">
15                 name=#{name},
16             </if>
17             <if test="gender != null">
18                 gender=#{gender},
19             </if>
20             <if test="image != null">
21                 image=#{image},
22             </if>
23             <if test="job != null">
24                 job=#{job},
25             </if>
26             <if test="entrydate != null">
27                 entrydate=#{entrydate},
28             </if>
29             <if test="deptId != null">
30                 dept_id=#{deptId},
31             </if>
32             <if test="updateTime != null">
33                 update_time=#{updateTime}
34             </if>
35         where id=#{id}
36     </update>
37
38 </mapper>
```

测试方法：

```

1  @Test
2  public void testUpdate2() {
3      //要修改的员工信息
4      Emp emp = new Emp();
5      emp.setId(20);
6      emp.setUsername("Tom111");
7      emp.setName("汤姆111");
8
9      emp.setUpdateTime(LocalDateTime.now());
10
11     //调用方法，修改员工数据
12     empMapper.update(emp);
13 }

```

执行的SQL语句：

```

JDBC Connection [HikariProxyConnection@384916201 wrapping com.mysql.cj.jdbc.ConnectionImpl@6206b4a7] will
=> Preparing: update emp set username=?, name=?, update_time=? where id=?
=> Parameters: Tom111(String), 汤姆111(String), 2022-12-11T15:23:26.272124500(LocalDateTime), 20(Integer)
<==    Updates: 1

```

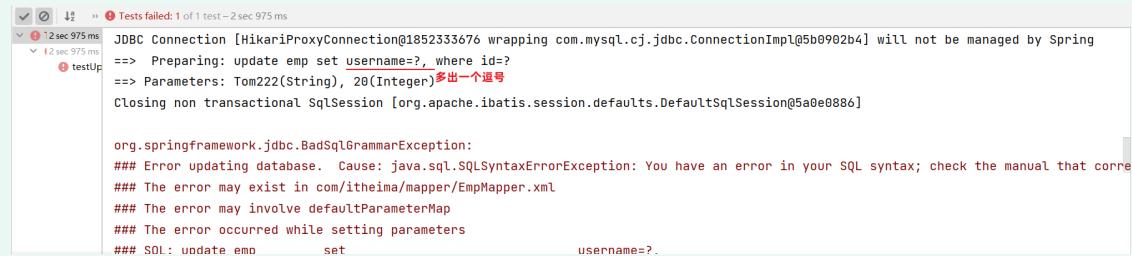
再次修改测试方法，观察SQL语句执行情况：

```

1  @Test
2  public void testUpdate2() {
3      //要修改的员工信息
4      Emp emp = new Emp();
5      emp.setId(20);
6      emp.setUsername("Tom222");
7
8      //调用方法，修改员工数据
9      empMapper.update(emp);
10 }

```

执行的SQL语句：



The screenshot shows the JUnit test run interface. A single test case named "testUpdate2" has failed. The failure message indicates a "BadSqlGrammarException" due to an error in the SQL syntax. The error message states: "Cause: java.sql.SQLSyntaxErrorException: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'set username=?' at line 1". This error occurs because the original code used the standard Java `set` method, which is not valid in SQL. The test log also shows the JDBC connection details and the prepared update statement.

以上问题的解决方案：使用 `<set>` 标签代替SQL语句中的`set`关键字

- `<set>`：动态的在SQL语句中插入`set`关键字，并会删掉额外的逗号。（用于update语句中）

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE mapper
3      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4      "https://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <mapper namespace="com.itheima.mapper.EmpMapper">
6
7      <!--更新操作-->
8      <update id="update">
9          update emp
10         <!-- 使用set标签，代替update语句中的set关键字 -->
11         <set>
12             <if test="username != null">
13                 username=#{username},
14             </if>
15             <if test="name != null">
16                 name=#{name},
17             </if>
18             <if test="gender != null">
19                 gender=#{gender},
20             </if>
21             <if test="image != null">
22                 image=#{image},
23             </if>
24             <if test="job != null">
25                 job=#{job},
26             </if>
27             <if test="entrydate != null">
28                 entrydate=#{entrydate},
29             </if>
30             <if test="deptId != null">
31                 dept_id=#{deptId},
32             </if>
33             <if test="updateTime != null">
34                 update_time=#{updateTime}
35             </if>
36         </set>
37         where id=#{id}
38     </update>
39 </mapper>
```

再次执行测试方法，执行的SQL语句：

```
JDBC Connection [HikariProxyConnection@899476243 wrapping com.mysql.cj.jdbc.ConnectionImpl@1068176]
==> Preparing: update emp SET username=? where id=?
==> Parameters: Tom222(String), 20(Integer)
<==    Updates: 1
```

小结

- <if>
 - 用于判断条件是否成立，如果条件为true，则拼接SQL
 - 形式：

```
1   <if test="name != null"> ... </if>
```

- <where>
 - where元素只会在子元素有内容的情况下才插入where子句，而且会自动去除子句的开头的AND或OR
- <set>
 - 动态地在行首插入 SET 关键字，并会删掉额外的逗号。（用在update语句中）

3.3 动态SQL-foreach

案例：员工删除功能（既支持删除单条记录，又支持批量删除）

	姓名	用户名	性别	职位	入职日期	操作
<input type="checkbox"/>	赵敏	zhaomin	女	班主任	2008-12-18	编辑 删除
<input checked="" type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	编辑 删除
<input checked="" type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	编辑 删除
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	编辑 删除
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	编辑 删除
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	编辑 删除
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	编辑 删除
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	编辑 删除
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	编辑 删除
<input type="checkbox"/>	风清扬	fengqingyang	男	讲师	2015-07-22	编辑 删除

每页展示记录数 每页共500条数据 < 2 3 4 5 ... 50 > 跳至 页

SQL语句：

```
1 delete from emp where id in (1,2,3);
```

Mapper接口：

```
1 @Mapper
2 public interface EmpMapper {
3     //批量删除
4     public void deleteByIds(List<Integer> ids);
5 }
```

XML映射文件：

- 使用 `<foreach>` 遍历 `deleteByIds` 方法中传递的参数 `ids` 集合

```
1 <foreach collection="集合名称" item="集合遍历出来的元素/项" separator="每一次遍历使用的分隔符"
2         open="遍历开始前拼接的片段" close="遍历结束后拼接的片段">
3     </foreach>
```

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4     "https://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace="com.itheima.mapper.EmpMapper">
6     <!--删除操作-->
7     <delete id="deleteByIds">
8         delete from emp where id in
9             <foreach collection="ids" item="id" separator="," open="("
10                close="")">
11                 #{id}
12             </foreach>
13     </delete>
14 </mapper>
```

```
<delete id="deleteByIds">
    delete from emp where id in
    <foreach collection="ids" item="id" separator="," open="(" close="")">
        #{id}
    </foreach>
</delete>
```

执行的SQL语句：

```
JDBC Connection [HikariProxyConnection@1997059690 wrapping com.mysql.cj.jdbc.ConnectionImpl@433ef204]
==> Preparing: delete from emp where id in ( ?, ?, ? )
==> Parameters: 21(Integer), 22(Integer), 23(Integer)
<==    Updates: 3
```

3.4 动态SQL-sql&include

问题分析：

- 在xml映射文件中配置的SQL，有时可能会存在很多重复的片段，此时就会存在很多冗余的代码

```
<select id="list" resultType="com.itheima.pojo.Emp">
    select id, username, password, name, gender, image, job, entrydate, dept_id, create_time, update_time from emp
    where
        <if test="name != null">
            name like concat('%',#{name},'%')
        </if>
        <if test="gender != null">
            and gender = #{gender}
        </if>
        <if test="begin != null and end != null">
            and entrydate between #{begin} and #{end}
        </if>
    order by update_time desc
</select>
```

```
<select id="getById" resultType="com.itheima.pojo.Emp">
    select id, username, password, name, gender, image, job, entrydate, dept_id, create_time, update_time from emp
    where id = #{id}
</select>
```

我们可以对重复的代码片段进行抽取，将其通过`<sql>`标签封装到一个SQL片段，然后再通过

`<include>`标签进行引用。

- `<sql>`：定义可重用的SQL片段
- `<include>`：通过属性`refid`，指定包含的SQL片段

```
<select id="list" resultType="com.itheima.pojo.Emp">
    <include refid="commonSelect"/>
    where
        <if test="name != null">
            name like concat('%',#{name},'%')
        </if>
        <if test="gender != null">
            and gender = #{gender}
        </if>
        <if test="begin != null and end != null">
            and entrydate between #{begin} and #{end}
        </if>
    order by update_time desc
</select>

<select id="getById" resultType="com.itheima.pojo.Emp">
    <include refid="commonSelect"/>
    where id = #{id}
</select>
```

```
<sql id="commonSelect">
    select id, username, password, name, gender, image, job, entrydate,
    dept_id, create_time, update_time from emp
    </sql>
```

SQL片段： 抽取重复的代码

```
1 <sql id="commonSelect">
2   select id, username, password, name, gender, image, job,
3   entrydate, dept_id, create_time, update_time from emp
4 </sql>
```

然后通过 `<include>` 标签在原来抽取的地方进行引用。操作如下：

```
1 <select id="list" resultType="com.itheima.pojo.Emp">
2   <include refid="commonSelect"/>
3   <where>
4     <if test="name != null">
5       name like concat('%',#{name},'%')
6     </if>
7     <if test="gender != null">
8       and gender = #{gender}
9     </if>
10    <if test="begin != null and end != null">
11      and entrydate between #{begin} and #{end}
12    </if>
13  </where>
14  order by update_time desc
15 </select>
```