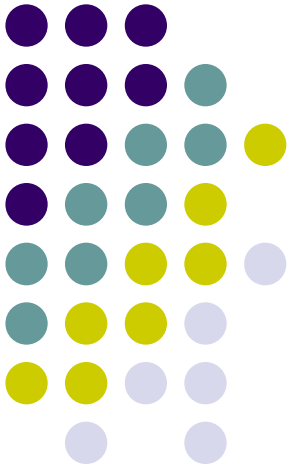
Mybatis技术简介

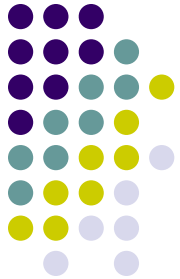
dezhaos@gmail.com



关联映射

- 一对一
- 一对多
- 多对多

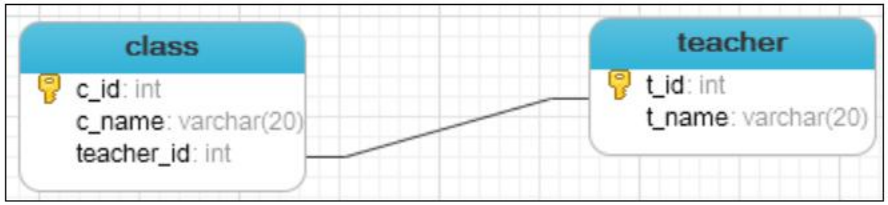
一对一关联



- **需求**
- 根据班级id查询班级信息(带老师的信息)
- 创建一张教师表和班级表，这里我们假设一个老师只负责教一个班，那么老师和班级之间的关系就是一种一对一的关系。

- CREATE TABLE teacher(
- t_id INT PRIMARY KEY AUTO_INCREMENT,
- t_name VARCHAR(20)
-);
- CREATE TABLE class(
- c_id INT PRIMARY KEY AUT(
- c_name VARCHAR(20),
- teacher_id INT
-);
- **ALTER TABLE class ADD CONSTRAINT fk_teacher_id FOREIGN KEY (teacher_id) REFERENCES teacher(t_id);**

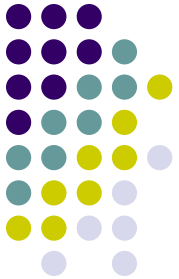
表之间的关系如下：



- INSERT INTO teacher(t_name) VALUES('teacher1');
- INSERT INTO teacher(t_name) VALUES('teacher2');

- INSERT INTO class(c_name, teacher_id) VALUES('class_a', 1);
- INSERT INTO class(c_name, teacher_id) VALUES('class_b', 2);

Teacher

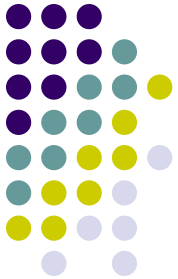


```
package me.gacl.domain;
public class Teacher {
    //定义实体类的属性，与teacher表中的字段对应
    private int id;           //id===>t_id
    private String name;    //name===>t_name
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

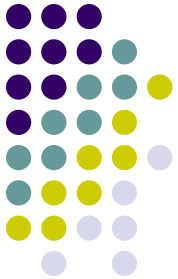
Classes类，Classes类是class表对应的实体类

```
package me.gacl.domain;
public class Classes {
    //定义实体类的属性，与class表中的字段对应
    private int id;          //id===>c_id
    private String name;    //name===>c_name
    /*** class表中有一个teacher_id字段，所以在Classes类中定义一个teacher属性，
     * 用于维护teacher和class之间的一对一关系，通过这个teacher属性就可以知道这个班级是由哪个老师负责的   */
    private Teacher teacher;
  public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Teacher getTeacher() {
        return teacher;
    }
    public void setTeacher(Teacher teacher)
{
        this.teacher = teacher;
    }
}
```

定义sql映射文件classMapper.xml

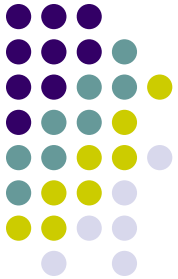


```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="me.gacl.mapping.classMapper">
<!--
    方式一：嵌套结果：使用嵌套结果映射来处理重复的联合结果的子集
        封装联表查询的数据(去除重复的数据)
    select * from class c, teacher t where c.teacher_id=t.t_id and c.c_id=1
-->
<select id="getClass" parameterType="int" resultMap="ClassResultMap">
    select * from class c, teacher t where c.teacher_id=t.t_id and c.c_id=#{id}
</select>
<!-- 使用resultMap映射实体类和字段之间的一一对应关系 -->
<resultMap type="me.gacl.domain.Classes" id="ClassResultMap">
    <id property="id" column="c_id"/>
    <result property="name" column="c_name"/>
    <association property="teacher" javaType="me.gacl.domain.Teacher">
        <id property="id" column="t_id"/>
        <result property="name" column="t_name"/>
    </association>
</resultMap>
```

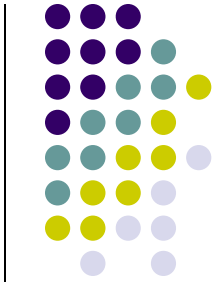


```
<!--
  方式二：嵌套查询：通过执行另外一个SQL映射语句来返回预期的复杂类型
    SELECT * FROM class WHERE c_id=1;
    SELECT * FROM teacher WHERE t_id=1   //1 是上一个查询得到的teacher_id的值
  -->
  <select id="getClass2" parameterType="int" resultMap="ClassResultMap2">
    select * from class where c_id=#{id}
  </select>
  <!-- 使用resultMap映射实体类和字段之间的一一对应关系 -->
  <resultMap type="me.gacl.domain.Classes" id="ClassResultMap2">
    <id property="id" column="c_id"/>
    <result property="name" column="c_name"/>
    <association property="teacher" column="teacher_id" select="getTeacher"/>
  </resultMap>

  <select id="getTeacher" parameterType="int" resultType="me.gacl.domain.Teacher">
    SELECT t_id id, t_name name FROM teacher WHERE t_id=#{id}
  </select>

</mapper>
```

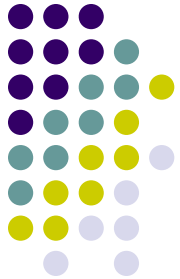
在conf.xml文件中注册classMapper.xml



- <mappers>
- <mapper resource="me/gacl/mapping/classMapper.xml"/>
- </mappers>

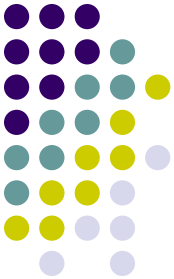


```
public class Test3 {
@Test
    public void testGetClass(){
        SqlSession sqlSession = MyBatisUtil.getSqlSession();
        /**
         * 映射sql的标识字符串，
         * me.gacl.mapping.classMapper是classMapper.xml文件中mapper标签的namespace属性的值， getClass是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL
         */
        String statement = "me.gacl.mapping.classMapper.getClass";//映射sql的标识字符串
        //执行查询操作，将查询结果自动封装成Classes对象返回
        Classes clazz = sqlSession.selectOne(statement,1);//查询class表中id为1的记录
        //使用SqlSession执行完SQL之后需要关闭SqlSession
        sqlSession.close();
        System.out.println(clazz);
    }
}
```



```
@Test
    public void testGetClass2(){
        SqlSession sqlSession = MyBatisUtil.getSqlSession();
        /**
         * 映射sql的标识字符串，
         * me.gacl.mapping.classMapper是classMapper.xml文件中mapper标签的namespace属性的值，
         * getClass2是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL
         */
        String statement = "me.gacl.mapping.classMapper.getClass2";//映射sql的标识字符串
        //执行查询操作，将查询结果自动封装成Classes对象返回
        Classes clazz = sqlSession.selectOne(statement,1);//查询class表中id为1的记录
        //使用SqlSession执行完SQL之后需要关闭SqlSession
        sqlSession.close();
        System.out.println(clazz);
    }
```

MyBatis一对一关联查询总结

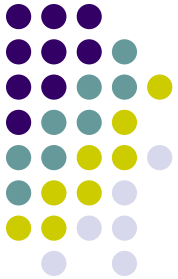


- MyBatis中使用association标签来解决一对一的关联查询，association标签可用的属性如下：
  - property:对象属性的名称
  - javaType:对象属性的类型
  - column:所对应的外键字段名称
  - select:使用另一个查询封装的结果

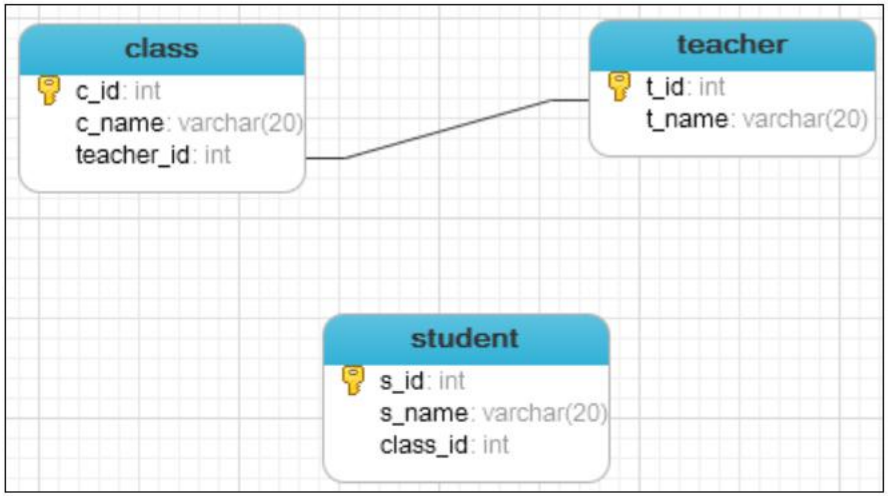
一对多关联

- **需求**
- 根据classId查询对应的班级信息,包括学生,老师。班级与学生之间存在一对多的关系

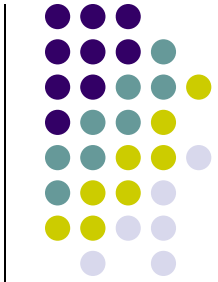
创建表和数据



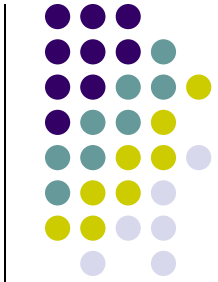
- CREATE TABLE student(
- s_id INT PRIMARY KEY AUTO_INCREMENT,
- s_name VARCHAR(20),
- class_id INT
-);
- INSERT INTO student(s_name, class_id) VALUES('student_A', 1);
- INSERT INTO student(s_name, class_id) VALUES('student_B', 1);
- INSERT INTO student(s_name, class_id) VALUES('student_C', 1);
- INSERT INTO student(s_name, class_id) VALUES('student_D', 2);
- INSERT INTO student(s_name, class_id) VALUES('student_E', 2);
- INSERT INTO student(s_name, class_id) VALUES('student_F', 2);



定义实体类



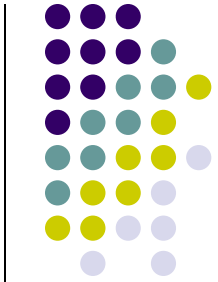
```
public class Student {
    //定义属性，和student表中的字段对应
    private int id;           //id===>s_id
    private String name;    //name===>s_name
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + "]";
    }
}
```



```
public class Classes {
    //定义实体类的属性，与class表中的字段对应
    private int id;          //id===>c_id
    private String name;    //name===>c_name
    /**class表中有一个teacher_id字段，所以在Classes类中定义一个teacher属性，
     * 用于维护teacher和class之间的一对一关系，通过这个teacher属性就可以知道这个班级是由哪个老师负责的
     */
    private Teacher teacher;
    //使用一个List<Student>集合属性表示班级拥有的学生
    private List<Student> students;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
```

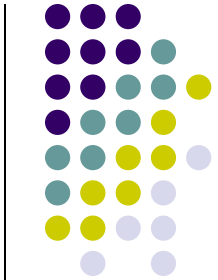
```
public Teacher getTeacher() {
    return teacher;
}
public void setTeacher(Teacher teacher)
{
    this.teacher = teacher;
}

public List<Student> getStudents() {
    return students;
}

public void setStudents(List<Student>
students) {
    this.students = students;
}
```

修改sql映射文件classMapper.xml

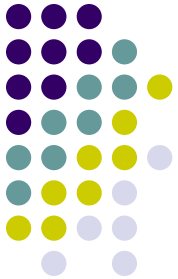


```
<!--
    根据classId查询对应的班级信息,包括学生,老师
 -->
<!--
方式一: 嵌套结果: 使用嵌套结果映射来处理重复的联合结果的子集
SELECT * FROM class c, teacher t,student s WHERE c.teacher_id=t.t_id AND c.C_id=s.class_id AND  c.c_id=1
 -->
<select id="getClass3" parameterType="int" resultMap="ClassResultMap3">
    select * from class c, teacher t,student s where c.teacher_id=t.t_id and c.C_id=s.class_id and  c.c_id=#{id}
</select>
<resultMap type="me.gacl.domain.Classes" id="ClassResultMap3">
    <id property="id" column="c_id"/>
    <result property="name" column="c_name"/>
    <association property="teacher" column="teacher_id" javaType="me.gacl.domain.Teacher">
        <id property="id" column="t_id"/>
        <result property="name" column="t_name"/>
    </association>
    <!-- ofType指定students集合中的对象类型 -->
    <collection property="students" ofType="me.gacl.domain.Student">
        <id property="id" column="s_id"/>
        <result property="name" column="s_name"/>
    </collection>
</resultMap>
```

```
<!--
    方式二：嵌套查询：通过执行另外一个SQL映射语句来返回预期的复杂类型
      SELECT * FROM class WHERE c_id=1;
      SELECT * FROM teacher WHERE t_id=1   //1 是上一个查询得到的teacher_id的值
      SELECT * FROM student WHERE class_id=1  //1是第一个查询得到的c_id字段的值
  -->
  <select id="getClass4" parameterType="int" resultMap="ClassResultMap4">
    select * from class where c_id=#{id}
  </select>
  <resultMap type="me.gacl.domain.Classes" id="ClassResultMap4">
    <id property="id" column="c_id"/>
    <result property="name" column="c_name"/>
    <association property="teacher" column="teacher_id" javaType="me.gacl.domain.Teacher" select="getTeacher2"></association>
    <collection property="students" ofType="me.gacl.domain.Student" column="c_id" select="getStudent"></collection>
  </resultMap>
  <select id="getTeacher2" parameterType="int" resultType="me.gacl.domain.Teacher">
    SELECT t_id id, t_name name FROM teacher WHERE t_id=#{id}
  </select>

  <select id="getStudent" parameterType="int" resultType="me.gacl.domain.Student">
    SELECT s_id id, s_name name FROM student WHERE class_id=#{id}
  </select>
```

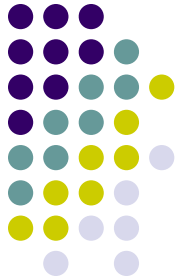


```
public class Test4 {

    @Test
    public void testGetClass3(){
        SqlSession sqlSession = MyBatisUtil.getSqlSession();
        /**
         * 映射sql的标识字符串，
         * me.gacl.mapping.classMapper是classMapper.xml文件中mapper标签的namespace属性的值，
         * getClass3是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL
         */
        String statement = "me.gacl.mapping.classMapper.getClass3";//映射sql的标识字符串
        //执行查询操作，将查询结果自动封装成Classes对象返回
        Classes clazz = sqlSession.selectOne(statement,1);//查询class表中id为1的记录
        //使用SqlSession执行完SQL之后需要关闭SqlSession
        sqlSession.close();
        System.out.println(clazz);
    }

}
```

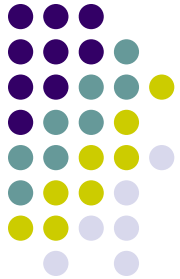


```
@Test
    public void testGetClass4(){
        SqlSession sqlSession = MyBatisUtil.getSqlSession();
        /**
         * 映射sql的标识字符串，
         * me.gacl.mapping.classMapper是classMapper.xml文件中mapper标签的namespace属性的值，
         * getClass4是select标签的id属性值，通过select标签的id属性值就可以找到要执行的SQL
         */
        String statement = "me.gacl.mapping.classMapper.getClass4";//映射sql的标识字符串
        //执行查询操作，将查询结果自动封装成Classes对象返回
        Classes clazz = sqlSession.selectOne(statement,1);//查询class表中id为1的记录
        //使用SqlSession执行完SQL之后需要关闭SqlSession
        sqlSession.close();
        System.out.println(clazz);
    }
```

MyBatis一对多关联查询总结



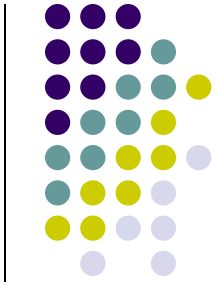
- MyBatis中使用collection标签来解决一对多的关联查询，
- ofType属性指定集合中元素的对象类型。



MyBatis 还有一个强大特性就是它的动态 SQL。

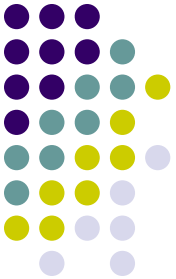
在实际项目开发中，经常需要根据不同条件拼接 SQL 语句，拼接时还要确保不能忘了必要的空格，有时候还要注意省掉列名列表最后的逗号，等等。在使用 JDBC 或其他类似持久层框架操作数据库时，处理这种情况是非常麻烦的，甚至可以用痛苦来形容，而在 MyBatis 中利用动态 SQL 这一特性可以很简单地解决这个问题。

动态SQL



- MyBatis 的一个强大的特性之一通常是它的动态 SQL 能力
  - if
  - choose(when,otherwise)
  - trim(where,set)
  - foreach

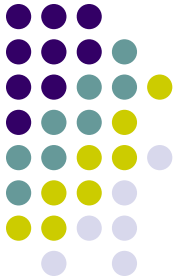
动态SQL：if语句—有条件的包含where子句部分



动态 SQL 通常会做的事情是有条件地包含 where 子句的一部分

```
<mapper namespace="org.fkit.mapper.EmployeeMapper">
    <select id="selectEmployeeByIdLike"
          resultType="org.fkit.domain.Employee">
          SELECT * FROM tb_employee WHERE state = 'ACTIVE'
        <!-- 可选条件，如果传进来的参数有 id 属性，则加上 id 查询条件 -->
          <if test="id != null ">
              and id = #{id}
          </if>
      </select>
</mapper>
```

以上语句提供了一个可选的根据 id 查找 Employee 的功能。如果没有传入 id，那么所有处于“ACTIVE”状态的 Employee 都会被返回；反之若传入了 id，那么就会把查找 id 内容的 Employee 结果返回。



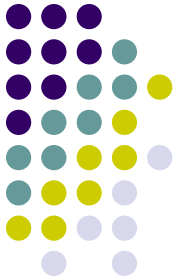
```
public interface EmployeeMapper {
    List<Employee> selectEmployeeByIdLike(HashMap<String, Object> params);
}
```

以上代码提供了一个和 EmployeeMapper.xml 中的 select 元素的 id 同名的方法，需要注意的是，selectEmployeeByIdLike 接收一个 HashMap 作为参数。

在 MyBatis 中，#{id}表达式获取参数有两种方式：一是从 HashMap 中获取集合中的 property 对象；二是从 javabean 中获取 property 对象。

- // 测试<select id="selectEmployeeByIdLike" ...>
- **public void testSelectEmployeeByIdLike(SqlSession session){**
- // 获得EmployeeMapper接口的代理对象
- EmployeeMapper em = session.getMapper(EmployeeMapper.**class**);
- // 创建一个HashMap存储参数
- HashMap<String, Object> params = **new HashMap<String, Object>();**
- // 设置id属性
- params.put("id", 1);
- // 调用EmployeeMapper接口的selectEmployeeByIdLike方法
- List<Employee> list = em.selectEmployeeByIdLike(params);
- // 查看返回结果
- list.forEach(employee -> System.***out.println(employee));***
- }

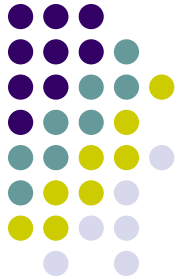
public void testSelectEmployeeByIdLike(SqlSession session){
// 获得EmployeeMapper接口的代理对象
EmployeeMapper em = session.getMapper(EmployeeMapper.**class**);
// 创建一个HashMap存储参数
HashMap<String, Object> params = **new HashMap<String, Object>();**
// 设置id属性
params.put("id", 1);
// 调用EmployeeMapper接口的selectEmployeeByIdLike方法
List<Employee> list = em.selectEmployeeByIdLike(params);
// 查看返回结果
list.forEach(employee -> System.***out.println(employee));***
}



- **public void testSelectEmployeeByIdLike(SqlSession session){**
- // 获得EmployeeMapper接口的代理对象
- EmployeeMapper em = session.getMapper(EmployeeMapper.**class);**
- // 创建一个HashMap存储参数
- HashMap<String, Object> params = **new HashMap<String, Object>();**
- // 调用EmployeeMapper接口的selectEmployeeByIdLike方法
- List<Employee> list = em.selectEmployeeByIdLike(params);
- // 查看返回结果
- list.forEach(employee -> System.***out.println(employee));***
- }

```
DEBUG [main] - ==>  Preparing: SELECT * FROM tb_employee
WHERE state = 'ACTIVE'
DEBUG [main] - ==> Parameters:
DEBUG [main] - <==      Total: 2
Employee [id=1, loginname=jack, password=123456, name=杰克,
sex=男, age=26, phone=13902019999, sal=9800.0, state=ACTIVE]
Employee [id=2, loginname=rose, password=123456, name=露丝,
sex=女, age=21, phone=13902018888, sal=6800.0, state=ACTIVE]
```

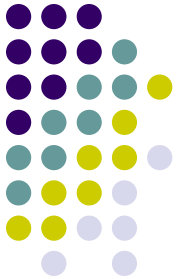
choose



有些时候，我们不想用所有的条件语句，而只想从中择其一二。针对这种情况，MyBatis 提供了 choose 元素，它有点像 Java 中的 switch 语句。

```
<select id="selectEmployeeChoose"
        parameterType="hashmap"
        resultType="org.fkit.domain.Employee">
        SELECT * FROM tb_employee WHERE state = 'ACTIVE'
        <!-- 如果传入了 id，就根据 id 查询，没有传入 id 就根据 loginname 和 password 查询，否则查
询 sex 等于男的数据 -->
        <choose>
            <when test="id != null">
                and id = #{id}
            </when>
            <when test="loginname != null and password != null">

            and loginname = #{loginname} and password = #{password}
        </when>
        <otherwise>
            and sex = '男'
        </otherwise>
    </choose>
</select>
```

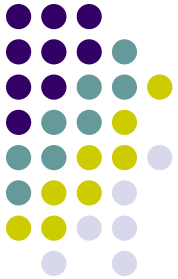
还是上面的例子，但是这次变为提供了 id 就按 id 查找，提供了 loginname 和 password 就按 loginname 和 password 查找，若两者都没有提供，就返回所有 sex 等于男的 Employee。



程序清单：codes/10/ DynamicSQLTest /src/org/fkit/mapper/ EmployeeMapper.java

```
List<Employee> selectEmployeeChoose(HashMap<String, Object> params);
```

```
public void testSelectEmployeeChoose(SqlSession session){
        EmployeeMapper em = session.getMapper(EmployeeMapper.class);
        HashMap<String, Object> params = new HashMap<String, Object>();
        // 设置 id 属性
        params.put("id", 1);
        params.put("loginname", "jack");
        params.put("password", "123456");
        List<Employee> list = em.selectEmployeeChoose(params);
        list.forEach(employee -> System.out.println(employee));
    }
```



```
<select id="selectEmployeeByIdLike"
     resultType="org.fkit.domain.Employee">
     SELECT * FROM tb_employee WHERE
     <if test="state != null ">
        state = #{state}
     </if>
     <if test="id != null ">
        and id = #{id}
     </if>
  </select>
```

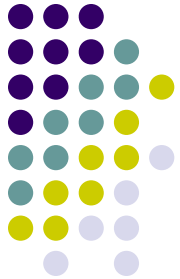
如果传入 state 参数，则执行正常。

如果没有传入参数，则会执行 sql 语句：

```
SELECT * FROM tb_employee WHERE
```

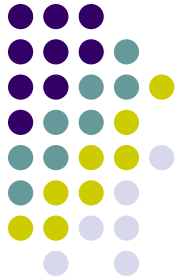
如果只是传入 id，则会执行 sql 语句：

```
SELECT * FROM tb_employee WHERE and id = ?
```



```
<select id="selectEmployeeLike"
     resultType="org.fkit.domain.Employee">
     SELECT * FROM tb_employee
     <where>
        <if test="state != null ">
           state = #{state}
        </if>
        <if test="id != null ">
           and id = #{id}
        </if>
        <if test="loginname != null and password != null">
           and loginname = #{loginname} and password = #{password}
        </if>
     </where>
  </select>
```

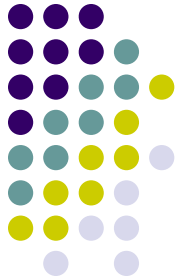
where 元素知道只有在一个以上的 if 条件有值的情况下才去插入 WHERE 子句。而且，若最后的内容是“AND”或“OR”开头，则 where 元素也知道如何将它们去除。



程序清单：codes/10/ DynamicSQLTest /src/org/fkit/mapper/ EmployeeMapper.java

```
List<Employee> selectEmployeeLike (HashMap<String, Object> params);
```

- **public void testSelectEmployeeChoose(SqlSession session){**
- EmployeeMapper em = session.getMapper(EmployeeMapper.**class);**
- HashMap<String, Object> params = **new HashMap<String, Object>();**
- // 设置id属性
- params.put("id", 1);
- params.put("loginname", "jack");
- params.put("password", "123456");
- List<Employee> list = em.selectEmployeeChoose(params);
- list.forEach(employee -> System.***out.println(employee));***
- }

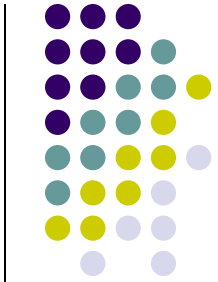


```
   DEBUG [main] - ==>  Preparing: SELECT * FROM tb_employee WHERE id = ? and loginname
? and password = ?
   DEBUG [main] - ==> Parameters: 1(Integer), jack(String), 123456(String)
   DEBUG [main] - <==      Total: 1
   Employee [id=1, loginname=jack, password=123456, name=杰克, sex=男, age=26,
ıone=13902019999, sal=9800.0, state=ACTIVE]
```

可以看到，当没有传入 state 参数时，MyBatis 自动过滤掉 id 前面的 and 关键字。

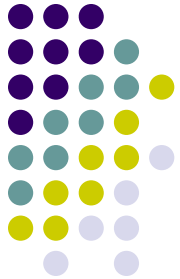
set

关于动态更新语句还可以使用 set 元素。set 元素可以被用于动态包含需要更新的列，而舍去其他的。



```
  <!-- 根据 id 查询员工信息 -->
  <select  id="selectEmployeeWithId"  parameterType="int"  resultType="org.fkit.
main.Employee">
      SELECT * FROM tb_employee where id = #{id}
  </select>
  <!-- 动态更新员工信息 -->
  <update id="updateEmployeeIfNecessary" parameterType="org.fkit.domain.Employee">
    update tb_employee
      <set>
        <if test="loginname != null">loginname=#{loginname},</if>
        <if test="password != null">password=#{password},</if>
        <if test="name != null">name=#{name},</if>
        <if test="sex != null">sex=#{sex},</if>
        <if test="age != null">age=#{age},</if>
        <if test="phone != null">phone=#{phone},</if>
        <if test="sal != null">sal=#{sal},</if>
        <if test="state != null">state=#{state}</if>
      </set>
    where id=#{id}
  </update>
```

set 元素会动态前置 SET 关键字，同时也会消除无关的逗号，因为使用了条件语句之后很可能就会在生成的赋值语句的后面留下这些逗号。



程序清单：codes/10/ DynamicSQLTest /src/org/fkit/mapper/ EmployeeMapper.java

```
// 根据 id 查询员工
Employee selectEmployeeWithId(Integer id);
// 动态更新员工
void updateEmployeeIfNecessary(Employee employee);
```

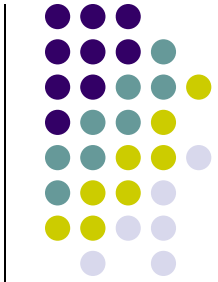
```
public void
testUpdateEmployeeIfNecessary(SqlSession
session){
EmployeeMapper em =
session.getMapper(EmployeeMapper.class);
Employee employee = em.selectEmployeeWithId(2);
// 设置需要修改的属性
employee.setLoginname("mary2");
employee.setPassword("123");
//employee.setSex(null);
employee.setName("玛丽");
em.updateEmployeeIfNecessary(employee);
}
```

运行 DynamicSQLTest 类的 main 方法，测试 updateEmployeeIfNecessary ()方法，控制台显示如下：

```
    DEBUG [main] - ==>  Preparing: SELECT * FROM tb_employee where id = ?
    DEBUG [main] - ==> Parameters: 4(Integer)
    DEBUG [main] - <==      Total: 1
    DEBUG [main] - ==>  Preparing: update tb_employee SET loginname=?, password=?,
name=?, sex=?, age=?, phone=?, sal=?, state=? where id=?
    DEBUG [main] - ==> Parameters: mary(String), 123(String), 玛丽(String), 女(String),
20(Integer), 13902016666(String), 5800.0(Double), ACTIVE(String), 4(Integer)
    DEBUG [main] - <==    Updates: 1
```

可以看到，首先执行了一条查询语句，查询 id 为 4 的员工，之后执行了一条 update 语句，根据传入的 Employee 对象更新员工信息。

切换到数据库，可以看到 id 为 4 的员工信息已经更新。

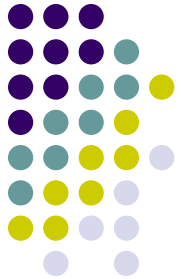


foreach

关于动态 SQL 另外一个常用的操作就是需要对一个集合进行遍历，通常发生在构建 IN 条件语句时。

```
<select id="selectEmployeeIn" resultType="org.fkit.domain.Employee">
  SELECT *
  FROM tb_employee
  WHERE ID in
  <foreach item="item" index="index" collection="list"
      open="(" separator="," close=")">
    #{item}
  </foreach>
</select>
```

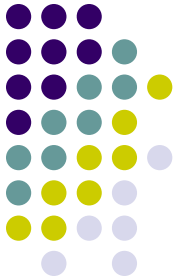
foreach 元素的功能非常强大，它允许指定一个集合，声明可以用在元素体内的集合项和索引变量。它也允许指定开闭匹配的字符串以及在迭代中间放置分隔符。这个元素是很智能的，因此它不会随机地附加多余的分隔符。



程序清单：codes/10/ DynamicSQLTest /src/org/fkit/mapper/ EmployeeMapper.java

```
// 根据传入的 id 集合查询员工
List<Employee> selectEmployeeIn(List<Integer> ids);
```

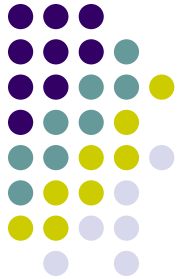
```
    // 创建 List 集合
    List<Integer> ids = new ArrayList<Integer>();
    // 往 List 集合中添加两个测试数据
    ids.add(1);
    ids.add(2);
    List<Employee> list = em.selectEmployeeIn(ids);
    list.forEach(employee -> System.out.println(employee));
```



运行 DynamicSQLTest 类的 main 方法，测试 selectEmployeeIn ()方法，控制台显示如下：

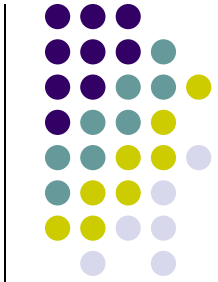
```
DEBUG [main] - ==>  Preparing: SELECT * FROM tb_employee WHERE ID in ( ? , ? )
DEBUG [main] - ==> Parameters: 1(Integer), 2(Integer)
DEBUG [main] - <==      Total: 2
Employee [id=1, loginname=jack, password=123456, name=杰克, sex=男, age=26,
phone=13902019999, sal=9800.0, state=ACTIVE]
Employee [id=2, loginname=rose, password=123456, name=露丝, sex=女, age=21,
phone=13902018888, sal=6800.0, state=ACTIVE]
```

可以看到，执行的 sql 语句是一个 in 条件语句，返回的是 List 集合中的 id 的员工数据。

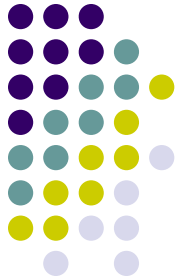


- MyBatis有两种实现方法，分别为基于注解和基于映射 文件。当需要操作的实体类较多时，逐个编写基于注解或基于映射文件的CURD耗时长且容易出错，使用MyBatis Generator可以保证CRUD的正确性，以及节省大量的时间。

MyBatis Generator



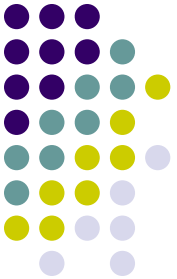
- 根据创建好的数据库表生成MyBatis的表对应的实体类，SQL映射文件和dao



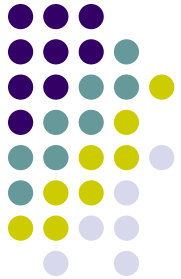
- Create DATABASE spring4_mybatis3;
- USE spring4_mybatis3;

- DROP TABLE IF EXISTS t_user;
- CREATE TABLE t_user (
- user_id char(32) NOT NULL,
- user_name varchar(30) DEFAULT NULL,
- user_birthday date DEFAULT NULL,
- user_salary double DEFAULT NULL,
- PRIMARY KEY (user_id)
-) ENGINE=InnoDB DEFAULT CHARSET=utf8;

下载mybatis-generator-core-1.3.2



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE generatorConfiguration
  PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
  "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">
<generatorConfiguration>
        <classPathEntry location="C:\Users\wyxhhwin\Downloads\mybatis-generator-core-1.3.2\mybatis-generator-core-1.3.2\lib\mysql-connector-5.1.8.jar" />
        <context id="sysGenerator" targetRuntime="MyBatis3">

        <jdbcConnection driverClass="com.mysql.jdbc.Driver"
                        connectionURL="jdbc:mysql://localhost:3306/mybatis"
                        userId="root" password="333333">
                </jdbcConnection>
<javaModelGenerator targetPackage="me.gacl.domain"
                        targetProject="C:\Users\wyxhhwin\Downloads\mybatis-generator-core-1.3.2\mybatis-generator-core-1.3.2\lib">
                        <property name="enableSubPackages" value="true" />
                        <property name="trimStrings" value="true" />
                </javaModelGenerator>

                <sqlMapGenerator targetPackage="me.gacl.mapping"
                        targetProject="C:\Users\wyxhhwin\Downloads\mybatis-generator-core-1.3.2\mybatis-generator-core-1.3.2\lib">
                        <property name="enableSubPackages" value="true" />
```



- java -jar mybatis.jar -configfile generator.xml –overwrite
- 自动生成