一、Mybatis概述

> MyBatis属于半自动ORM（Object Relation Mapping） 对象关系映射框架；
>
> 特点：
>
> 对开发人员而言，核心sql还是需要自己优化
>
> sql（配置文件）和java编码分开，功能边界清晰，一个专注业务、一个专注数据
>
> 框架组合：
>
> SSH：Spring + Stuart2 + Hibernate
>
> SSI：Spring + Stuart2 + iBatis
>
> SSM：Spring + SpringMVC + MyBatis

通过Mybatis执行一条SELECT SQL步骤

- 第一步：创建Mybatis全局配置mybatis-config.xml，添加jdbc连接配置+mappers引入SQL映射文件；
  - mybatis-config.xml
  - ```
    <?xml version="1.0" encoding="UTF-8" ?>
    <!DOCTYPE configuration
            PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
            "http://mybatis.org/dtd/mybatis-3-config.dtd">
    <configuration>
        <!-- 引入jdbc配置文件 -->
        <properties resource="jdbc.properties"></properties>

        <settings>
            <!--    映射下划线到驼峰命名（解决别名转换问题）     -->
            <setting name="mapUnderscoreToCamelCase" value="true"/>
        </settings>

        <!-- 将bean下所有类添加别名，默认别名=类名    -->
        <typeAliases>
            <package name="com.josen.getting_start.bean"/>
        </typeAliases>

        <!--  引入外部jdbc配置文件  -->
        <environments default="development">
            <environment id="development">
                <transactionManager type="JDBC"/>
                <!-- 数据库配置信息 -->
                <dataSource type="POOLED">
                    <!-- 使用配置文件 -->
                    <property name="driver" value="${jdbc.driver}"/>
                    <property name="url" value="${jdbc.url}"/>
    ```
    ```

```
                <property name="username" value="${jdbc.username}"/>
                <property name="password" value="${jdbc.password}"/>
            </dataSource>
        </environment>
    </environments>
    <!-- 引入SQL映射文件 -->
    <mappers>
        <!-- 引入单个文件 -->
<!--        <mapper resource="EmployeeMapper.xml"/>-->
        <!--   批量引入指定包下所有SQL映射文件      -->
        <package name="com.josen.getting_start.mapper"/>
    </mappers>

</configuration>
```

- 第二步：创建SQL映射文件，编写对应Mapper Interface的SQL语句；
  - EmpMapper.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="EmpMapper">
        <!--  查询一条记录  -->
        <select id="queryOne" resultType="Employee">
             select * from jobs where job_id = #{id}
         </select>
    </mapper>
    ```
  - `<mapper>`标签里对应的增删改查标签：`select/insert/update/delete`；
- 第三步：创建Mapper Interface接口文件，定义增删改查抽象方法；
  - EmpMapper.java
  - ```
    /**
     * @ClassName EmpMapper
     * @Description
     * 映射接口全类名=EmpMapper
     * 要执行的映射接口方法名=queryOne
     * @Author Josen
     */
    public interface EmpMapper {
        // 根据id查询一条记录
        Job queryOne(Integer id);
    }
    ```
- 第四步：创建一个Java Bean（也叫：POJO）类，类中的属性对应数据库表的字段名；
- 第五步：加载Mybatis全局配置，获取`SqlSession`调用`getMapper(映射接口.class)`方法返回实现映射接口的代理类，调用`queryOne`执行指定的SQL映射文件里的SQL语句；
  - testing.java
  - ```
    @Test
       public void testMyBatisMapper() throws IOException {
    ```
    ```

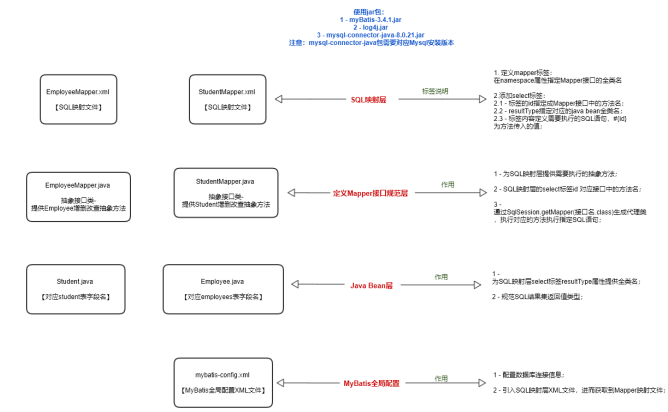
```
        // 1. 加载MyBatis全局配置文件
        String resource = "mybatis-config.xml";
        InputStream inputStream =
Resources.getResourceAsStream(resource);
        // 2. 获取SqlSessionFactory
        SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
        // 3. SqlSessionFactory生成一个SQL会话
        SqlSession sqlSession = sqlSessionFactory.openSession();
        // 4. 获取EmployeeMapper接口代理类
        EmpMapper mapper = sqlSession.getMapper(EmpMapper.class);
        // 5. 调用EmpMapper接口queryOne方法，传入id并执行SQL语句
        // --------------select---------------------
        Employee employee = mapper.queryOne(1);

        System.out.println(employee);

        // 默认为手动提交，insert/delete/update操作需要调用commit()方法提交SQL
        //sqlSession.commit();
        sqlSession.close();
    }
```

二、Mapper接口开发

> 两个绑定：
>
> - Mapper接口与SQL映射文件的绑定，映射文件的`namespace`的值必须指定成接口的全类名；
> - Mapper接口的方法与SQL映射文件的具体SQL语句的绑定，SQL语句的id值必须指定成接口的方法名；
>
> 接口开发的好处：
>
> - 有更明确的类型约束；
> - 利用接口定制规范；



三、全局配置文件（mybatis-config.xml）

- properties：引入外部的属性文件
  - resource: 从类路径下引入属性文件 【类路径：当前工程下src目录，或在idea标记为Sources Root的目录】；
  - url: 引入网络路径或者是磁盘路径下的属性文件；
  - `<property>`：一个具体的属性配置；
- settings：包含了很多重要的配置项
  - ```
    <settings>
        <!-- 1.映射下划线到驼峰命名 -->
        <setting name="mapUnderscoreToCamelCase" value="true"></setting>
    </settings>
    ```
- typeAliases：别名处理
  - ```
    <typeAliases>
        <!--
            1. typeAlias:给某个java类型取别名
            type：指定java的全类名
            alias：指定别名，默认的别名就是类名【不区分大小写】
        -->
        <typeAlias
                    type="com.josen.beans.Student"
                    alias="student"
                    >
        </typeAlias>
        <!-- 2. package：为指定的包以及包下的类批量取别名 -->
    ```

```
    <package name="com.josen.beans"></package>
</typeAliases>
```

- 如果有别名冲突的情况，可以到指定的java类文件中，加上`@Alias("other")`指定别名；

- environments：Mybatis支持配置多个环境，通过default属性来指定具体使用的环境
  - transactionManager：事务管理，Mybatis属于DAO层，通常会交给Spring 业务层去处理事务。可选值【JDBC | MANAGED】；
  - dataSource：数据源（数据源交给Spring处理）
    - UNPOOLED：不使用连接池
    - POOLED：使用连接池
    - JNDI ： 从Web服务器获取数据源【如：Tomcat】

```
<environments default="development">
        <!-- 开发环境 -->
        <environment id="development">
            <transactionManager type="JDBC"/>
            <!-- 数据库配置信息 -->
            <dataSource type="POOLED">
                <property name="xxx" value="xxx"/>
                ...
            </dataSource>
        </environment>

        <!-- 测试环境 -->
        <environment id="testing">
            <transactionManager type="JDBC"/>
            <!-- 数据库配置信息 -->
            <dataSource type="POOLED">
                <property name="xxx" value="xxx"/>
                ...
            </dataSource>
        </environment>
    </environments>
```

- mappers：引入SQL映射文件
  - ```
    <!-- 引入SQL映射文件  -->
    <mappers>
        <!-- 引入指定SQL映射文件 -->
        <mapper resource="EmployeeMapper.xml"/>
        <!-- 批量引入SQL映射文件，要求：必须与Mapper接口同名同位置 -->
        <package name="com.josen.mappers"/>
    </mappers>
    ```

四、生成主键方式

> 1) 支持主键自增，例如MySQL数据库
>
> 2) 不支持主键自增，例如Oracle数据库
    ```

> 若数据库支持自动生成主键的字段（比如 MySQL 和 SQL Server），则可以设置 useGeneratedKeys=”true”，然后再把 keyProperty 设置到目标属性上。

```
<insert id="insertEmployee"
        parameterType="com.atguigu.mybatis.beans.Employee"
        databaseId="mysql"
        useGeneratedKeys="true"
        keyProperty="id">
        insert into tbl_employee(last_name,email,gender) values(#{lastName},#
{email},#{gender})
</insert>
```

五、参数传递

- SQL xml = SQL映射文件
- Mapper interface = 接口映射文件
- Tips：
  - select查询必须声明resultType返回值类型；

> 1) 单个普通(基本/包装+String)参数
>
> 这种情况MyBatis可直接使用这个参数，不需要经过任何处理。
>
> 取值:#{随便写}

SQL xml

```
<select id="queryOne" resultType="job">
    select * from jobs where job_id = #{jobId}
</select>
```

Mapper interface

```
// 根据id查询一条记录
Job queryOne(String jobId);
```

> 2) 多个参数
>
> 任意多个参数，都会被MyBatis重新包装成一个Map传入。Map的key是param1，param2，或者0，1...，值就是参数的值
>
> 取值: #{0 1 2 ...N 或 param1 param2 ..... paramN}

SQL xml

```
    <select id="queryByJobIdAndMaxSalary" resultType="Job">
        select * from jobs where job_id=#{id} and max_salary>#{salary}
    </select>
```

Mapper interface

```
/**
     * 根据id&salary条件查询记录
     * @param id
     * @param salary
     * @return Job
     */
    Job queryByJobIdAndMaxSalary(String id,  Double salary);
```

> 3) 命名参数
>
> 为参数使用@Param起一个名字，MyBatis就会将这些参数封装进map中，key就是我们自己指定的名字
>
> 取值: #{自己指定的名字 或 param1 param2 ... paramN}

SQL xml

```
   <!--  Mapper接口提供两个参数查询（使用@Param指定参数名）  -->
    <select id="queryByJobIdAndMaxSalary" resultType="Job">
        select * from jobs where job_id=#{jobId} and max_salary>#{maxSalary}
    </select>
```

Mapper interface

```
/**
     * 根据job_id&max_salary条件查询指定记录
     * 使用@Param注解指定参数名称，提供SQL映射文件#{name}获取
     * @param id jobId
     * @param salary maxSalary
     * @return Job
     */
    Job queryByJobIdAndMaxSalary(@Param("jobId") String id, @Param("maxSalary")
Double salary);
```

> 4) POJO
>
> 当这些参数属于我们业务POJO时，我们直接传递POJO
>
> 取值: #{POJO的属性名}

SQL xml

```
   <insert id="insertOne">
        insert into jobs values(#{jobId},#{jobTitle},#{minSalary},#{maxSalary})
    </insert>
```

Mapper interface

```
// 新增一条记录
    Boolean insertOne(Job job);
```

> 5) Map
>
> 我们也可以封装多个参数为map，直接传递
>
> 取值: #{使用封装Map时自己指定的key}

SQL xml

```
   <!--  接受Map集合传递过来的数据 - 通过#{key}取值  -->
    <select id="queryJobForMap" resultType="Job">
        select * from jobs where job_title like '${title}%' and min_salary > #
{salary}
    </select>
```

Mapper interface

```
    /**
     * 使用Map传递参数
     * @param map
     * @return
     */
    List<Job> queryJobForMap(Map<String,Object> map);
```

test.java

```
    // 获取一个SqlSession
    SqlSession sqlSession = CommonUtils.createSqlSession();

    JobsMapper mapper = sqlSession.getMapper(JobsMapper.class);
    // 创建Map集合，定义要传递的参数
    HashMap<String, Object> map = new HashMap<>();
    map.put("title","A");
    map.put("salary",5000);
    List<Job> jobs = mapper.queryJobForMap(map);
    jobs.forEach(System.out::println);
```

> 6) Collection/Array
>
> 会被MyBatis封装成一个map传入, Collection对应的key是collection,Array对应的key是array. 如果确定是List集合，key还可以是list.
>
> 取值:
>
> Array: #{array}
>
> Collection(List/Set): #{collection}
>
> List : #{collection / list}

六、参数的获取方式

> - #{key}：占位符
> - 可取单个普通类型、 POJO类型 、多个参数、 集合类型
> - 使用PreparedStatement获取参数的值，预编译到SQL中。安全

- ${key}：字符拼接
- 可取单个普通类型、POJO类型、多个参数、集合类型.
- 注意: 取单个普通类型的参数，${}中不能随便写，必须使用 _parameter

_parameter 是Mybatis的内置参数.

- 使用Statement获取参数的值，拼接到SQL中。有SQL注入问题。
- 原则: 能用#{}取值就优先使用#{},#{}解决不了的可以使用${}.
- 例如: 原生的JDBC不支持占位符的地方，就可以使用${}
- Select column1 ,column2... from 表 where 条件group by 组标识 having 条件 order by 排序字段 desc/asc limit x, x
- 如：

```
# #{}占位符不支持动态设置表名
SELECT * FROM #{tableName};
# ${}字符拼接可以动态设置表名
SELECT * FROM #{tableName};
```

七、resultType自动映射 & resultMap自定义映射

自动映射

- autoMappingBehavior默认是PARTIAL，开启自动映射的功能。唯一的要求是列名和javaBean属性名一致
- 如果autoMappingBehavior设置为null则会取消自动映射
- 数据库字段命名规范，POJO属性符合驼峰命名法，如A_COLUMN□aColumn，我们可以开启自动驼峰命名规则映射功能，mapUnderscoreToCamelCase=true

自定义映射

- id 和 result：用于完成主键值的映射
  - id：用于完成主键值的映射；
  - result：用于完成普通列的映射；
  - ```
    <resultMap type="com.atguigu.mybatis.beans.Employee" id="myEmp">
            <!-- 指定主键映射名称 -->
            <id column="id"  property="id" />
            <!--
            result标签属性值：
                property：映射到列结果的字段或属性
                column：数据表的列名
                javaType：一个Java类的完全限定名、或一个类型的别名（如果映射一个JavaBean，Mybatis通常可以断定类型）
                jdbcType：JDBC类型是仅仅需要对插入、更新和删除操作可能为空的列进行处理
                typeHandle：类型处理器。这个属性可以覆盖默认的类型处理器，属性值可以是：类的完全限定名、类型处理器的实现、类型别名
            -->
            <result column="last_name" property="lastName"/>
            <result column="email" property="email"/>
            <result column="gender" property="gender"/>
    ```
    ```

```
</resultMap>
```

- association：一个复杂的类型关联，许多结果将包装成这种类型
  - Bean中的属性可能会是一个对象,我们可以使用联合查询，并以级联属性的方式封装对象.使用association标签定义对象的封装规则；
  - 使用级联查询：
    - ①-添加Java Bean

      Department.java

      ```
      public class Department {
          private int department_id;
          private String department_name;
          private int manager_id;
          // 地址id-外键
          private int location_id;
          // 部门地址信息
          private Location location;
          // 省略get/set...
      }
      ```

      Location.java

      ```
      public class Location {
          private int location_id;
          private String street_address;
          private String postal_code;
          private String city;
          private String state_province;
          private String country_id;
          // 省略get/set...
      }
      ```

    - ②-添加Mapper接口映射方法

      DepartmentMapper.java

      ```
      public interface DepartmentMapper {
          // 根据department_id查询指定部门+地址信息
          Department queryDepartmentAndLocation(@Param("departmentId")
      Integer id);
      }
      ```

    - ③-添加SQL映射文件

      DepartmentMapper.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="DepartmentMapper">
          <!--
              使用级联查询:
                          根据department_id查询指定部门+地址信息
      ```

```
                    Department queryDepartmentAndLocation();
    -->
   <select id="queryDepartmentAndLocation"
resultMap="mapper.getDepartmentAndLocation">
        SELECT

d.department_id,d.department_name,l.city,l.country_id,l.location_id
        FROM
            departments d
        JOIN
            locations l
        ON
            d.`location_id`=l.`location_id`
        WHERE
            d.department_id=#{departmentId};
   </select>
    <resultMap id="getDepartmentAndLocation"
type="beans.Department">
        <id column="department_id" property="did"></id>
        <result column="department_name" property="departmentName">
</result>
        <!--    使用级联    -->
        <association property="location" javaType="beans.Location">
            <id column="location_id" property="locationId"></id>
            <result column="city" property="city"></result>
            <result column="country_id" property="countryId">
</result>
        </association>
    </resultMap>
</mapper>
```

- ④-获取mapper代理类，调用getDepartmentAndLocation方法，执行对应SQL

  testing.java

```
@Test
public void testResultMap(){
    SqlSession sqlSession = null;
    try {
       sqlSession = CommonUtils.createSqlSession();
       DepartmentMapper mapper =
sqlSession.getMapper(DepartmentMapper.class);
       Department department =
mapper.queryDepartmentAndLocation(100);
       System.out.println(department);
    } catch (IOException e) {
       e.printStackTrace();
    }finally {
       if(sqlSession!=null)
         sqlSession.close();
    }
}
```

- 使用分步查询（懒加载）
  - 实际的开发中，对于每个实体类都应该有具体的增删改查方法，也就是DAO层。

- 情况一：只需要查询员工信息即可，并不需要再执行一次查询部门信息的SQL；
- 情况二：需要查询员工信息+部门信息；
- 当只需要获取employee数据时，不会执行第②条SQL：
  - 1）先通过员工的id查询员工信息

```
SELECT employee_id,employee_name,department_id FROM employees
WHERE employee_id = #{eid};
```

  - 2）再通过查询出来的员工信息中的外键`(department_id)`查询对应的部门信息.

```
SELECT * FROM departments WHERE department_id = #{did};
```

- 实现懒加载：
  - ①-配置Mybatis全局文件，开启延迟加载：

```
<!-- 开启延迟加载 -->
<setting name="lazyLoadingEnabled" value="true"/>
<!-- 设置加载的数据是按需(false)还是全部(true) -->
<setting name="aggressiveLazyLoading" value="false"/>
```

  - ②-添加Java Bean

    Employee.java

```
public class Employee {
    private int eid;
    private Double salary;
    private String name;
    // 员工所属部门信息
    private Department dep;
    // 省略get & set...
}
```

    Department.java

```
public class Department {
    // 部门id
    private int did;
    // 部门名称
    private String departmentName;
}
```

  - ③-添加Mapper接口抽象方法

    EmployeeMapper.java

```
public interface EmployeeMapper {
    // 查询指定id员工信息
    Employee queryEmployeeByEidForStep(Integer eid);
}
```

    DepartmentMapper.java

```
public interface DepartmentMapper {
    // 查询指定departmentId部门信息
    Department queryDepartmentByDid(@Param("departmentId")
Integer did);
}
```

- ④-添加SQL映射文件

  EmployeeMapper.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="mapper.EmployeeMapper">
    <select id="queryEmployeeByEidForStep"
resultMap="getEmployeeForStep">
        SELECT
            employee_id,salary,name,department_id
        FROM
            employees
        WHERE
            salary>=#{salary}
    </select>
    <!-- 自定义映射 -->
    <resultMap id="getEmployeeForStep" type="beans.Employee">
        <id column="employee_id" property="eid"></id>
        <result column="salary" property="salary"></result>
        <result column="name" property="name"></result>
        <!--    select：引入指定要分步执行的SQL映射文件（精确到方法名）
   -->
        <association
                property="deps"
                column="department_id"

 select="mapper.DepartmentMapper.queryDepartmentByDid">
        </association>
    </resultMap>
</mapper>
```

  DepartmentMapper.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="mapper.DepartmentMapper">
   <select id="queryDepartmentByDid"
resultMap="getDepartmentByDid">
        SELECT
            department_id,department_name
        FROM
            departments
        WHERE
            d.department_id=#{departmentId};
```

```
    </select>
     <!--  resultMap：自定义映射  -->
     <resultMap id="getDepartmentByDid" type="beans.Department">
        <id column="department_id" property="did"></id>
        <result column="department_name"
property="departmentName"></result>
     </resultMap>
</mapper>
```

- ⑤-测试分步查询懒加载

  testing.java

```
    /**
     * 分步查询-懒加载测试
     */
    @Test
    public void testResultMapForStep(){
        SqlSession sqlSession = null;
        try {
            sqlSession = CommonUtils.createSqlSession();
            EmployeeMapper mapper =
sqlSession.getMapper(EmployeeMapper.class);
            // 查询id=10的员工信息
            Employee emp =
mapper.queryEmployeeByEidForStep(10);

            // 只执行查询员工信息SQL
            System.out.printf("员工：%s \t 月薪：\s
\n",emp.getName(),emp.getSalary().toString());

            // 当需要使用到Dep数据时，执行查询员工+部门信息SQL
            //System.out.println("部门信息"+emp.getDep());
        } catch (IOException e) {
            e.printStackTrace();
        }finally {
            if(sqlSession!=null)
                sqlSession.close();
        }
}
```

- collection：复杂类型的集合
  - Bean中的属性可能会是一个集合对象,我们可以使用联合查询，并以级联属性的方式封装对象.使用collection标签定义对象的封装规则；
  - 用法与`association`类似；
- 总结：
  - `association`标签通常用于处理单个结果集，`collection`则用于处理多个结果集；
  - `association`&`collection`标签的select属性，用于引入外部其他SQL映射文件；
  - column属性用于指定数据库表的字段名，property属性用于指定Java Bean中的属性名；
  - resultMap自定义映射：通常用于使用级联、分步查询；
  - 拓展：

- `association` & `collection` 的fetchType属性：
  - fetchType=”eager”：设置该查询是否需要使用延迟加载；
  - 默认fetchType=”lazy”；
- 分步查询的多列值传递：
  - 如果分步查询时，需要传递给调用的查询中多个参数，则需要将多个参数封装成Map来进行传递，语法如下: {k1=v1, k2=v2....}
  - 在所调用的查询方，取值时就要参考Map的取值方式，需要严格的按照封装map时所用的key来取值.

八、动态SQL

> 作用：
>
> 通过Mybatis自定义的 OGNL（对象图导航语言） 表达式，定义一些动态拼接的SQL语句
>
> 访问对象属性： person.name
>
> 调用方法： person.getName()
>
> 调用静态属性/方法： @java.lang.Math@PI @java.util.UUID@randomUUID()
>
> 调用构造方法： new com.atguigu.bean.Person(‘admin’).name
>
> 运算符： +,-*,/,%
>
> 逻辑运算符： in,not in,>,>=,<,<=,==,!=
>
> 注意：xml中特殊符号如”,>,<等这些都需要使用转义字符 ， 详细查询W3C【如：&符号 = &】

常用标签：

- if & where & set标签
  - If：用于完成简单的判断；
  - where：用于解决SQL语句中where关键字以及条件中第一个and或者or的问题 （通常用于select语句）；
  - set： 主要是用于解决修改操作中SQL语句中可能多出逗号的问题（通常用于update语句）；

  SQL映射文件

```
    <!--  if & where标签  -->
    <select id="queryEmployeeForDynamicSql" resultType="beans.Employee">
        SELECT
            employee_id eid,salary,first_name,last_name,phone_number
        FROM
            employees
        <where>
            <if test="eid!=0">
               and employee_id = #{eid}
            </if>
```

```
            <if test="salary!=null">
               and salary = #{salary}
            </if>
            <if test="firstName!=null">
               and first_name = #{firstName}
            </if>
            <if test="lastName!=null">
               and last_name = #{lastName}
            </if>
            <if test="phoneNumber!=null">
               and phone_number = #{phoneNumber}
            </if>
        </where>
    </select>
```

EmployeeMapper

```
public interface EmployeeMapper {
    /**
     * 动态SQL查询-01
     * @param condition 对应的属性值为where查询条件
     * @return
     */
    Employee queryEmployeeForDynamicSql(Employee condition);
}
```

- choose标签 (when, otherwise)
  - 作用：用于分支判断，类似于java中的switch case,只会满足所有分支中的一个；
  - choose-when-otherwise ===》switch-case-default；

  举例：

```
<select id="getEmpsByConditionChoose" resultType="beans.Employee">
        select id ,last_name, email,gender from employees
        <where>
            <choose>
                <when test="id!=null">
                    id = #{id}
                </when>
                <when test="lastName!=null">
                    last_name = #{lastName}
                </when>
                <when test="email!=null">
                    email = #{email}
                </when>
                <otherwise>
                     gender = 'm'
                </otherwise>
            </choose>
        </where>
</select>
```

- trim 标签 - (可以替代where, set)

- trim 标签可以在条件判断完的SQL语句前后 添加或者去掉指定的字符（trim标签可以嵌套）；
- trim标签属性：
  - prefix: 添加前缀；
  - prefixOverrides: 去掉前缀；
  - suffix: 添加后缀；
  - suffixOverrides: 去掉后缀；

举例

```
<select id="getEmpsByConditionTrim" resultType="beans.Employee">
        SELECT id , last_name ,email  , gender FROM employees
        <trim prefix="where"  suffixOverrides="and">
            <if test="id!=null">
                 id = #{id} and
            </if>
            <if test="lastName!=null &amp;&amp; lastName!=&quot;&quot;">
                 last_name = #{lastName} and
            </if>
            <!-- trim()去除左右空格 -->
            <if test="email!=null and email.trim()!=''">
                 email = #{email} and
            </if>
            <if test="&quot;m&quot;.equals(gender) or
&quot;f&quot;.equals(gender)">
                gender = #{gender}
            </if>
        </trim>
</select>
```

- foreach标签
  - 主要用户循环迭代
  - foreach标签属性：
    - collection: 要迭代的集合
    - item: 当前从集合中迭代出的元素
    - open: 开始字符
    - close:结束字符
    - separator: 元素与元素之间的分隔符
    - index:
      - 迭代的是List集合: index表示的当前元素的下标
      - 迭代的Map集合: index表示的当前元素的key

举例：

```
<select id="getEmpsByConditionForeach" resultType="beans.Employee">
         select id , last_name, email ,gender from employees where id in
         <foreach collection="ids" item="curr_id" open="(" close=")"
separator="," >
                #{curr_id}
         </foreach>
</select>
```

- sql标签
  - sql 标签是用于抽取可重用的sql片段，将相同的，使用频繁的SQL片段抽取出来，单独定义，方便多次引用.
  - ```
    <!-- 要频繁复用的SQL -->
    <sql id="selectSQL">
            select id , last_name, email ,gender from tbl_employee
    </sql>
    <!-- 引用SQL -->
    <include refid="selectSQL"></include>
    ```
    ```