MyBatis

主讲:石小俊

1.介绍MyBatis

持久层框架

ORM框架

前身叫ibatis

所有的持久层的框架

底层代码一定是JDBC

JDBC是java访问数据库的唯一方式

1-1 持久层

dao层--DataAccessObject--数据访问对象层

单一的持久化操作

1-2 框架

由其他程序员所封装的一系列的资源

主要是对于某些技术的封装

将其封装成一个jar、js、css、less...

简化了我们的开发

1-3 ORM

Object Relational Mapping

对象关系映射

是一种程序技术，用于实现面向对象编程语言中不同类型的系统之间的数据的转换

类 映射 表

属性 映射 字段

1-4 优点

- 提高代码复用性

- 是一个支持普通sql、高级映射以及存储过程的一个ORM框架
- 几乎消除了所有的jdbc代码，不在需要手动设置参数
- 对于简单操作，mybatis会自动帮我们进行映射
- 支持大量优秀的插件

1-5 开发依赖

- 自己的
  - mybais-xxx.jar
- 数据库的
  - mysql-xxx.jar
  - ojdbcxxx.jar

2.配置文件

- configuration配置
  - 主配置文件
  - 一个mybatis工程只能存在一个configuration配置
  - 主要配置如下信息
    - 连接信息
    - 环境信息
    - 事务信息
    - 别名配置
    - 插件配置
    - 注册mapping
    - 访问properties
    - ……
  - 配置的是与整个工程相关的信息
- mapping配置
  - 可以存在无数个
  - 其需要在configuration中进行注册后才会生效
  - 该配置类似于我们以前所使用的dao的实现类
  - 针对每一个功能进行配置
    - sql操作
    - 对象映射
    - 。。。

3.configuration配置

```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>

    <!--
        environments:为当前的工程配置其所使用的所有的环境
        可以配置无数个环境
        default属性:指定当前正在使用的环境是哪一个
            其值所对应的数据为某一个环境的id属性所对应的值
    -->
    <environments default="mybatis">
        <!--
            environment:配置其中一个环境
            id:当前环境的唯一性标识符
        -->
        <environment id="mybatis">
            <!--
                transactionManager:配置事务管理器
                type:指定当前的事务管理器的类型，其值有两种
                    jdbc:使用简单的jdbc事务，开启、提交、回滚
                        相当做初始做了如下操作
                        conn.setAutoCommint(false)
                    managed:该配置表示我啥也不做
                        我从不关心事务
                        将事务交给其他框架来处理，例如:spring
            -->
            <transactionManager type="jdbc"></transactionManager>
            <!--
                dataSource:配置数据源
                type:当前数据源的类型，其值有三种
                    UNPOOLED:简单配置,相当于DriverManager.getConnection(url,username,password)
                    POOLED:使用数据源连接池
                    JNDI:向其他容器要连接
            -->
            <dataSource type="UNPOOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"></property>
                <property name="url" value="jdbc:mysql://127.0.0.1:3306/mybatis?
useUnicode=true&amp;characterEncoding=utf8"></property>
                <property name="username" value="root"></property>
                <property name="password" value=""></property>
            </dataSource>
        </environment>
    </environments>

</configuration>
```

4.第一个mybatis程序

```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>

    <!--
        environments:为当前的工程配置其所使用的所有的环境
        可以配置无数个环境
        default属性:指定当前正在使用的环境是哪一个
            其值所对应的数据为某一个环境的id属性所对应的值
    -->
    <environments default="mybatis">
        <!--
            environment:配置其中一个环境
            id:当前环境的唯一性标识符
        -->
        <environment id="mybatis">
            <!--
                transactionManager:配置事务管理器
                type:指定当前的事务管理器的类型，其值有两种
                    jdbc:使用简单的jdbc事务，开启、提交、回滚
                        相当做初始做了如下操作
                        conn.setAutoCommint(false)
                    managed:该配置表示我啥也不做
                        我从不关心事务
                        将事务交给其他框架来处理，例如:spring
            -->
            <transactionManager type="jdbc"></transactionManager>
            <!--
                dataSource:配置数据源
                type:当前数据源的类型，其值有三种
                    UNPOOLED:简单配置,相当于DriverManager.getConnection(url,username,password)
                    POOLED:使用数据源连接池
                    JNDI:向其他容器要连接
            -->
            <dataSource type="UNPOOLED">
                <property name="driver" value="com.mysql.jdbc.Driver"></property>
                <property name="url" value="jdbc:mysql://127.0.0.1:3306/mybatis?
useUnicode=true&amp;characterEncoding=utf8"></property>
                <property name="username" value="root"></property>
                <property name="password" value=""></property>
            </dataSource>
        </environment>
    </environments>


    <!-- 注册mapper文件-->
    <mappers>
        <mapper resource="mapper/UserMapper.xml"></mapper>
    </mappers>

</configuration>
```

```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">
<!--
    namespace:当前mapper文件的唯一性标识符
        其值可以随便/必须是固定的某一个值
        其值不可以重复
        当直接使用mapper文件时，其值可以随便写
        当使用三层结构，将其作为dao层某个接口的实现类的时候，不能随便写
        此时其值为当前接口的路径名.类名
-->
<mapper namespace="dao.UserDao">

    <!--
       在mapper操作中，CURD操作的标签名对应的是数据库CURD操作的关键字
       insert/delete/update/select标签:对应数据库相关操作
    -->
    <!--
        insert标签:执行添加操作
        id:表示当前操作的唯一性标识符，可以理解为方法名
        如果当前xml指向了某一个dao层的接口
        则此处id的值必须与dao层接口方法名完全一致
        parameterType:方法的参数类型
        标签体中编写sql语句
    -->
    <insert id="insertUser" parameterType="entity.User">
      insert into
      t_user
        (username,password,phone,address)
      values
        (#{username},#{password},#{phone},#{address})
    </insert>

    <!--
        resultType:表示返回值的类型
    -->
    <select id="selectById" parameterType="int" resultType="entity.User">
        select id,username,password,phone,address
        from t_user
        where id = #{id}
    </select>

</mapper>
```

5.多参数查询

- 方式一
  - 将多个参数封装成对象进行操作
  - 将封装后的对象作为当前方法的参数

```
<select id="selectByUsernameAndPassword" parameterType="entity.UserParameter"
resultType="entity.User">
    select <include refid="userColumn"></include>
    from t_user
    where username = #{username}
    and password = #{password}
</select>
```

- 方式二
  - 通过Param注解来实现
  - @Param("占位符的字符")

```
public List<User> selectByUsernameAndPassword2(@Param("aa") String username,
@Param("password") String password);
```

```
<select id="selectByUsernameAndPassword2" resultType="entity.User">
    select <include refid="userColumn"></include>
    from t_user
    where username = #{aa}
    and password = #{password}
</select>
```

- 方法三
  - 通过参数的索引号来实现
  - 索引号从0开始

```
public List<User> selectByUsernameAndPassword3(String username,String password);
```

```
<select id="selectByUsernameAndPassword3" resultType="entity.User">
        select <include refid="userColumn"></include>
        from t_user
        where username = #{0}
        and password = #{1}
    </select>
```

6.手动映射

自动映射是将数据库查询出来结果将其映射到对应的java对象中

与字段名一致的属性中

当数据库查询结果的字段名与java对象的属性名不一致时

mybatis无法实现自动映射

因此想要实现映射有两种方式

- 方式一
  - 将数据库查询语句的字段名保持与java对象的属性名一致
  - 别名
- 方式二
  - 手动将数据库的字段与java对象的属性进行映射

```
<!-- 取别名实现自动映射 -->
<select id="selectById2" parameterType="integer" resultType="entity.User">
    select user_id 'id',user_username 'username',user_password 'password',
    user_phone 'phone',user_address 'address'
    from t_user2
    where user_id = #{id}
</select>


<!--
    resultMap:配置手动映射
    id:当前resultMap标签的唯一性标识符，用于他人引用
    type:映射的java对象是谁
-->
<resultMap id="userMapper" type="User">
    <!--
            id:配置主键
            result:配置普通字段
        -->
    <id property="id" column="user_id"></id>
    <result property="username" column="user_username"></result>
    <result property="password" column="user_password"></result>
    <result property="phone" column="user_phone"></result>
    <result property="address" column="user_address"></result>
</resultMap>


<!--
    手动映射
   resultMap:手动进行映射，指向某一个resultMap标签
-->
<select id="selectById3" parameterType="integer" resultMap="userMapper">
    select user_id,user_username,user_password,
    user_phone,user_address
    from t_user2
    where user_id = #{id}
</select>
```

7.模糊查询

- 方式一
  - 手动在操作中拼接模糊查询的字符
- 方式二
  - 使用bind标签实现模糊查询的操作

```
<select id="selectByUsername" parameterType="string" resultType="User">
    select <include refid="userColumn"></include>
    from t_user
    where username like #{username} escape '/'
</select>

<select id="selectByUsername2" parameterType="string" resultType="User">
    select <include refid="userColumn"></include>
    from t_user
    <!--
            bind标签:对方法的参数做额外的处理
            name:哪一个参数，如果参数是对象，则name属性指向的是对象中的某一个属性
            value:做什么样的额外处理
                其中_paraneter表示当前传递到方法中的具体的参数值
        -->
    <bind name="username" value="'%'+_parameter+'%'"></bind>
    where username like #{username} escape '/'
</select>
```

8.动态sql

8-1 if标签

类似于jstl的if标签

- test属性
  - 值是一个布尔类型
  - 当值为true的时候，则执行标签体中的内容
  - 当值为false的时候，不执行标签体的内容

```
<!-- 动态sql之if -->
<select id="selectByParams1" parameterType="User" resultType="User">
    select <include refid="userColumn"></include>
    from t_user
    where 1=1
    <if test="id != null and id != ''">
```

```
        and id = #{id}
    </if>
    <if test="username != null and username != ''">
        and username = #{username}
    </if>
    <if test="password != null and password != ''">
        and password = #{password}
    </if>
    <if test="phone != null and phone != ''">
        and phone = #{phone}
    </if>
    <if test="address != null and address != ''">
        and address = #{address}
    </if>
</select>
```

8-2 choose标签

类似于jstl中choose标签

- when标签
  - 通过该标签的test属性判断表达式的布尔值
  - 当值为true，执行标签体的内容，且执行完成之后跳出整个choose标签
  - 当值为false，不执行标签体的内容，进入下一个标签进行判断
- otherwise标签
  - 当同一个choose标签中的所有的when标签的条件均不满足时
  - 执行该标签体的内容
- 在choose标签中，只要有一个标签满足条件，则执行完毕之后跳出整个choose标签

```
<!-- 动态sql之choose -->
<select id="selectByParams2" parameterType="User" resultType="User">
    select <include refid="userColumn"></include>
    from t_user
    where
    <choose>
        <when test="id != null and id != ''">
            id = #{id}
        </when>
        <when test="username != null and username != ''">
            username = #{username}
        </when>
        <when test="password != null and password != ''">
            password = #{password}
        </when>
        <when test="phone != null and phone != ''">
            phone = #{phone}
        </when>
        <otherwise>
            address = #{address}
        </otherwise>
```

```
    </choose>
</select>
```

8-3 where标签

该标签单独使用没有任何意义

一般会与if/choose联合使用

会在where标签所包含的语句块的开头部分添加一个where关键字

且会忽略所有连接条件中第一个连接条件的关键字and/or

```
<!-- 动态sql之where -->
<select id="selectByParams3" parameterType="User" resultType="User">
    select <include refid="userColumn"></include>
    from t_user
    <where>
        <if test="id != null and id != ''">
            and id = #{id}
        </if>
        <if test="username != null and username != ''">
            and username = #{username}
        </if>
        <if test="password != null and password != ''">
            and password = #{password}
        </if>
        <if test="phone != null and phone != ''">
            and phone = #{phone}
        </if>
        <if test="address != null and address != ''">
            and address = #{address}
        </if>
    </where>
</select>
```

8-4 set标签

单独使用没有任何意义

一般会与if/choose使用

用于修改操作

会在set标签所包含的语句块的开头部分增加一个set关键

且会忽略包含的语句块末尾的逗号

```
<!-- 动态sql之set -->
```

```
    <update id="updateUser" parameterType="User">
        update t_user
        <set>
            <if test="username != null and username != ''">
                username = #{username},
            </if>
            <if test="password != null and password != ''">
                password = #{password},
            </if>
            <if test="phone != null and phone != ''">
                phone = #{phone},
            </if>
            <if test="address != null and address != ''">
                address = #{address},
            </if>
        </set>
        where id = #{id}
    </update>
```

8-5 trim标签

单独使用没有任何意义

一般会与if/choose联合使用

trim可以实现where与set的功能

可以在包含的语句块中添加某个前缀或者后缀

还能忽略包含的语句块中的首部或者尾部的某些内容

- prefix:添加前缀
- suffix:添加后缀
- prefixOverrides:忽略首部内容
- suffixOverrides:忽略尾部内容
    - 当需要忽略内容不止一个的时候，通过|来表示或者的意思
    - **且|后面不可以跟空格**

```
<!-- 动态sql之trim -->
<select id="selectByParams4" parameterType="User" resultType="User">
    select <include refid="userColumn"></include>
    from t_user
    <trim prefix="where" prefixOverrides="and|or">
        <if test="id != null and id != ''">
            and id = #{id}
        </if>
        <if test="username != null and username != ''">
            or username = #{username}
        </if>
        <if test="password != null and password != ''">
            and password = #{password}
        </if>
```

```
        <if test="phone != null and phone != ''">
            and phone = #{phone}
        </if>
        <if test="address != null and address != ''">
            and address = #{address}
        </if>
    </trim>
    </select>
```

8-6 foreach标签

查询id为1,3,4的数据

```
select * from t_user where id in (1,3,4);
```

主要用于构建in/not in语句

参数的值是多个数据

可以通过foreach进行迭代操作

- collection:当前参数的类型
  - list
  - map
  - array
- item:迭代的每一个元素的别名
- index:当前迭代到的索引号
- open:以什么开始
- close:以什么结束
- separator:多个元素之间以什么分割

```
<select id="selectByIds" resultType="User">
    select <include refid="userColumn"></include>
    from t_user
    where id in
    <foreach collection="list" item="id" open="(" close=")" separator=",">
        #{id}
    </foreach>
</select>
```

9.注解方式

对于简单语句来说，使用注解代码会更加清晰，然而Java注解对于复杂语句来说就会混乱，应该限制使用。因此，如果你不得不做复杂的事情，那么最好使用XML来映射语句。

```
package dao;

import entity.User;
import org.apache.ibatis.annotations.Result;
import org.apache.ibatis.annotations.Results;
import org.apache.ibatis.annotations.Select;

/**
 * Author:shixiaojun@itany.com
 * Date:2018/10/29 16:48
 * Description:
 * version:1.0
 */
public interface UserMapper {

    @Select("select * from t_user where id = #{id}")
    public User selectById(Integer id);

    @Select("select * from t_user2 where user_id = #{id}")
    @Results(value={
            @Result(id=true,property = "id",column = "user_id"),
            @Result(property = "username",column = "user_username"),
            @Result(property = "password",column = "user_password"),
            @Result(property = "phone",column = "user_phone"),
            @Result(property = "address",column = "user_address")
    })
    public User selectById2(Integer id);

}
```

10.保存返回主键

- useGeneratedKeys
  - 保存时是否返回主键
  - 默认为false
  - true--返回
  - false--不返回
- keyProperty
  - 主键所映射的对象中的属性是谁
  - 不可以省略
- keyColumn
  - 主键所对应的数据库中的字段是谁

- 可以省略

```
<insert id="insertReturnPrimaryKey" parameterType="User" useGeneratedKeys="true"
keyProperty="id" keyColumn="id">
        insert into
        t_user
          (username,password,phone,address)
        values
          (#{username},#{password},#{phone},#{address})
    </insert>
```

11.多表操作

11-1 有一个的关系

- 一对一
- 多对一

```
部门表t_dept
    id
    name
员工表
    id
    name
    salary
    dept_id
练习要求：构建两张表
使用mybatis实现同时保存一个部门与3个员工的信息
且这3个员工属于同时保存的部门
```

有一个关系的查询

- 直接使用关联属性进行操作

```
<resultMap id="empMapper" type="Emp">
    <id property="id" column="id"></id>
    <result property="name" column="name"></result>
    <result property="salary" column="salary"></result>
    <!--
        association标签:处理有一个关系的连接查询映射
        property属性:表示对当前对象的哪一个属性进行映射
        column属性:当前表中的哪一个字段与连接的表进行关联，可以理解为外键字段
        javaType属性:指定当前属性的对象类型
    -->
```

```
    <association property="dept" column="dept_id" javaType="entity.Dept">
        <id property="id" column="d.id"></id>
        <result property="name" column="d.name"></result>
    </association>
</resultMap>

<select id="selectById" parameterType="integer" resultMap="empMapper">
    select e.id,e.name,e.salary,d.id 'd.id',d.name 'd.name'
    from t_emp e
    join t_dept d
      on e.dept_id = d.id
    where e.id = #{id}
</select>
```

- 关联的嵌套结果

```
<resultMap id="empMapper2" type="Emp">
    <id property="id" column="id"></id>
    <result property="name" column="name"></result>
    <result property="salary" column="salary"></result>
    <!--
        可以引用另一个已经存在的映射
        resultMap属性:指向另一个已经存在的映射，简化开发
     -->
    <association property="dept" column="dept_id" javaType="entity.Dept"
resultMap="dao.DeptDao.deptMapper"></association>
</resultMap>

<select id="selectById2" parameterType="integer" resultMap="empMapper2">
    select e.id,e.name,e.salary,d.id 'd.id',d.name 'd.name'
    from t_emp e
    join t_dept d
      on e.dept_id = d.id
    where e.id = #{id}
</select>
```

- 关联的嵌套查询
    - 该方式会引发N+1问题

```
<resultMap id="empMapper3" type="Emp">
    <id property="id" column="id"></id>
    <result property="name" column="name"></result>
    <result property="salary" column="salary"></result>
    <!--
        可以引用另一个查询方法
        select:指向另一个已经存在的查询方法，简化开发
     -->
    <association property="dept" column="dept_id" javaType="entity.Dept"
select="dao.DeptDao.selectById"></association>
</resultMap>
```

```
<select id="selectById" parameterType="int" resultType="Dept">
    select id,name
    from t_dept
    where id=#{id}
</select>


<!--
    该方式存在一个问题,N+1问题
    所谓的N+1
    表示假设查询员工的时候，查询出来N条记录
    则每一条记录都有一个与之对应的dept信息
    根据每一个部门的id去部门表进行一次查询
    因此一共查询了1次员工，N次部门
    一个方法最终查询次数为:N+1次
    会导致查询效率较低
    因此不建议使用
-->
<select id="selectById3" parameterType="int" resultMap="empMapper3">
    select id,name,salary,dept_id
    from t_emp
    where id = #{id}
</select>

<select id="selectAll" resultMap="empMapper3">
    select id,name,salary,dept_id
    from t_emp
</select>
```

11-2 有多个的关系

有多个关系的查询方法

- 直接使用集合属性进行映射

```
<resultMap id="deptMapper1" type="Dept">
    <id property="id" column="id"></id>
    <result property="name" column="name"></result>
    <!--
        collection标签:处理有多个的关系映射
        property:对哪一个集合属性进行映射
        column属性:当前表中的哪一个字段与关联表进行关联，此处可以理解为主键
        ofType属性:当前集合中对象的类型
    -->
    <collection property="emps" column="id" ofType="Emp">
        <id property="id" column="e.id"></id>
        <result property="name" column="e.name"></result>
        <result property="salary" column="salary"></result>
    </collection>
```

```
</resultMap>

<select id="selectById1" parameterType="integer" resultMap="deptMapper1">
        select d.id ,d.name,e.id 'e.id',e.name 'e.name',e.salary
        from t_dept d
        join t_emp e
          on d.id = e.dept_id
        where d.id = #{id}
    </select>
```

- 集合的嵌套结果
- 集合的嵌套查询

12.代码生成器

mybatis-generator

12-1 介绍代码生成器

是一款mybatis的插件

该插件并不是在项目中使用，而是在为项目开始开发之前做准备工作

主要用于生成在开发过程中所需要使用的持久层以及其依赖的一些代码

在开发之前将这些代码生成出来放入到对应的工程中

12-2 生成的代码有哪些

- dao层接口
- mapper配置文件
- 实体类以及条件对象
    - 数据库中表所映射的对象
    - 数据库操作中可能使用到的一切条件

12-3 缺点

该代码生成器生成的代码仅限单表操作

多表查询时需要我们自己去编写

所有的表之间的关系需要我们手动去维护

12-4 使用

- jar包
  - 插件的
  - 数据库的
- 配置文件
  - generatorConfig.xml

```
<?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>
    <!-- 指定jar包的全名 -->
     <classPathEntry location="mysql-connector-java-5.1.18-bin.jar" />

  <context id="mysql" targetRuntime="MyBatis3">
    <!-- 数据库连接 -->
    <jdbcConnection driverClass="com.mysql.jdbc.Driver"
        connectionURL="jdbc:mysql://127.0.0.1:3306/mybatis"
        userId="root"
        password="">
    </jdbcConnection>

    <javaTypeResolver >
      <property name="forceBigDecimals" value="false" />
    </javaTypeResolver>
    <!--
        配置生成的代码
        javaModelGenerator标签:配置生成的实体类
        sqlMapGenerator标签:配置生成的mapper文件
        javaClientGenerator标签:配置生成的dao接口
        targetPackage属性:生成的代码的路径名
        targetProject属性:生成的代码放在哪
    -->
    <javaModelGenerator targetPackage="entity" targetProject="src">
      <property name="enableSubPackages" value="true" />
      <property name="trimStrings" value="true" />
    </javaModelGenerator>

    <sqlMapGenerator targetPackage="mapper"  targetProject="src">
      <property name="enableSubPackages" value="true" />
    </sqlMapGenerator>

    <javaClientGenerator type="XMLMAPPER" targetPackage="dao"  targetProject="src">
      <property name="enableSubPackages" value="true" />
    </javaClientGenerator>

    <!--
        配置你所需要生成的代码有哪些
```

```
            与数据库相对应
            一张表对应一套
        tableName:表名
        domainObjectName:生成的代码的核心名字
            加入此处核心名字为User
            则实体类叫做:User
            参数对象叫做:UserExample
            dao接口叫做:UserMapper
            mapper文件叫做:UserMapper.xml
    -->
    <table schema="mybatis" tableName="t_user" domainObjectName="User"></table>

  </context>
</generatorConfiguration>
```