Mybatis学习

Mybatis：一款优秀的持久层框架，用于简化JDBC开发

JavaEE：表现层、业务层、持久层

官方参考文档：MyBatis中文网

快速入门

安装

如果使用SpringBoot，无需配置，添加模块即可

maven构建依赖

```
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>x.x.x</version>
</dependency>
```

核心配置文件

Mybatis核心配置文件*mybatis-config.xml*：修改数据库连接信息

```
<?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 default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="${driver}"/>
        <property name="url" value="${url}"/>
        <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="org/mybatis/example/BlogMapper.xml"/>
  </mappers>
</configuration>
```

配置SQL语句

一般按*XXXMapper.xml*命名，如*UserMapper.xml*，并加载到上方核心配置文件中

```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="org.mybatis.example.BlogMapper"> <!-- 命名空间 -->
  <select id="selectBlog" resultType="Blog">  <!-- SQL语句、SQL语句的标识、返回结果的包装类型 -->
    select * from Blog where id = #{id}  <!-- SQL语句 -->
  </select>
</mapper>
```

Main中使用

假设*MyBatisDemo.java*，main方法中：

```java
// 加载mybatis配置
String resource = "org/mybatis/example/mybatis-config.xml";
InputStream inputStream = Resources.getResourceAsStream(resource);
SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);

// 获取SqlSession对象，执行sql
SqlSession sqlSession = sqlSessionFactory.openSession();

// 执行sql
List<Blog> blogs = sqlSession.selectList("命名空间.id");

// 释放资源
sqlSession.close();
```

Mapper代理开发

接口映射的四个一致：

- **接口**名要和对应的**映射**文件的名称相同（只是后缀名不同），如：`UserMapper.java`与`UserMapper.xml`
- 接口的全限定名要和mapper**映射**文件的namespace **一致**，如`<mapper namespace="com.example.learn.UserMapper`
- 接口中的方法名要和mapper映射文件中的唯一标识的id相同，如`<select id="selectAll" resultType="User">`与`List<User> users = selectAll();`
- **接口**的方法返回类型和mapper **映射**文件返回的类型**一致**

1. 定义与SQL映射文件同名的Mapper接口，并将Mapper接口和SQL映射文件放置在同一目录下

```
src/main/java
├── com.example.project
|   └── mapper
|       └── UserMapper.java
└── resources
    └── mapper
        └── UserMapper.xml
```

mapper/UserMapper.java：

```
package xxx;

public interface UserMapper {
    // ...
}
```

核心配置文件配置包扫描：

```
<mapper>
    <package name="包.mapper" />
</mapper>
```

2. 设置SQL映射文件的namespace属性为Mapper接口全限定名

```
<mapper namespace="com.example.learn.UserMapper"> <!-- 命名空间 -->
  <select id="selectBlog" resultType="Blog">  <!-- SQL语句、SQL语句的标识、返回结果的包装类型 -->
    select * from Blog where id = #{id}  <!-- SQL语句 -->
  </select>
</mapper>
```

3. Mapper接口中定义方法，方法名就是SQL映射文件中sql语句的id，并保持参数类型和返回值类型一致。

   UserMapper.java：

```
List<User> selectAll();
```

4. 编码
   - 通过SqlSession的getMapper方法获取Mapper接口的代理对象
   - 调用对应方法

```
UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
List<User> users = userMapper.selectAll();
```

Mybatis核心配置文件

核心配置文件：一般命名为***mybatis-config.xml***

内容框架：

```
<?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>
    <!-- ... -->
</configuration>
```

配置信息

- `<environments>`与`<environment>`，可修改default值切换环境信息
- `<transactionManager type="JDBC"/>`，事物管理方式，将被Spring接管
- `<dataSource type="POOLED">`，数据库连接池，将被Spring接管
- `<property>`，数据库连接信息
- `<typeAliases>`与`<package>`，配置包扫描，可扫描Mapper类、实体类等

```
<environments default="development">
    <typeAliases>
        <package name="com.example.pojo" />
    </typeAliases>
    <environment id="development">
        <transactionManager type="JDBC" />
        <dataSource type="POOLED">
            <!-- 数据库连接信息 -->
            <property name="driver" value="com.mysql.jdbc.Driver" />
            <property name="url" value="jdbc:mysql:///mybatis?useSSL=false" />
            <property name="username" value="root" />
            <property name="password" value="123456" />
        </dataSource>
        <!-- ... -->
    </environment>
    <mapper>
        <package name="包.mapper" />
    </mapper>

    <environment id="test">
        <!-- ... -->
    </environment>
</environments>
```

配置文件完成增删改查

MybatisX是一款基于IDEA的快速开发插件，为效率而生

列名与参数的映射关系

1. 列名与实体类属性名的映射关系

默认情况下，实体类属性名与列名**全部一致**可直接映射

若列名存在下划线等情况，实体类属性名使用**驼峰命名法**，解决方案：

1. **配置文件中启用MyBatis驼峰映射功能**

```
mybatis:
  configuration:
    map-underscore-to-camel-case: true
```

2. **手动指定映射**

```xml
<!--
    id: 唯一标识，用在resultMap
    type: 映射的类型，支持别名
-->
<resultMap id="userResultMap" type="User">
    <!-- 取别名、映射类型 -->
    <!--
        column: 列名
        property: 属性类名
    -->
    <id></id>  <!-- 主键用id -->
    <result column="user_name" property="userName"></result> <!-- 其他用result
-->
    <result column="phone_number" property="phoneNumber"></result>
</resultMap>

<!-- resultMap属性替代resultType属性 -->
<select id="selectAllUsers" resultMap="userResultMap">
    select * from users;
</select>
```

3. **可对不一样的名字使用`as`取别名，别名必须与属性列名一致**，如：

```xml
<select id="selectAllUsers" resultType="user">
    select id, user_name as userName, phone_number as phoneNumber
    from users;
</select>
```

4. **注解SQL中显示映射列名**

```java
@Mapper
public interface UserMapper {
    @Select("SELECT id, user_name, created_at FROM user WHERE id = #
{userId}")
    @Results({
        @Result(column = "id", property = "id"),
        @Result(column = "user_name", property = "userName"),
        @Result(column = "created_at", property = "createdAt")
    })
    User findUserById(@Param("userId") Long userId);
}
```

5. 截取片段（不推荐）

```xml
<sql id="user_column">
    id, user_name as userName, phone_number as phoneNumber
</sql>

<select id="selectAllUsers" resultType="Users">
    select
        <include refid="user_column"/>
    from users;
</select>
```

2. mapper接口方法的形参与列名、属性名的映射关系

mapper接口方法形参与实体类属性名互不影响。

mapper接口与属性列名的映射关系：

1. 若只有一个参数，形参名与列名不一致不重要
2. 使用**param**
   - 若有多个参数，MyBatis 默认会在**SQL语句**中使用 `param1`、`param2` 等顺序编号来引用参数，也可手动设置，如

     ```
     @Select("<script>" +
             "SELECT * FROM user WHERE id = #{param1} AND status = #{param2}"
     +
             "</script>")
     User findUserByIdAndStatus(Long id, String status);
     ```

   - 也可在接口方法中明确指定参数名称（例如通过 `@Param` 注解）

     ```
     @Select("<script>" +
             "SELECT * FROM user WHERE id = #{id} AND status = #{status}" +
             "</script>")
     User findUserByIdAndStatus(@Param("id") Long id, @Param("status") String
     status);
     ```

3. 多个参数也可使用对象接收

3. MyBatis动态SQL语句中的参数绑定

MyBatis动态SQL语句中的参数，如`<if test="参数名 == 条件">`中的参数名，需与`#{ 参数名 }`中的参数名一致才可绑定，而`#{参数名}`可自己指定，只需与接口方法形参中`@Param("")`注解指定的一致。

1. 查询

1.1 查询所有数据：列名到属性的映射问题

1. 编写接口方法：Mapper接口
   - 参数：无
   - 结果：`List<Brand>`
   - ```
     List <Brand> selectAllUsers();
     ```

2. 编写SQL语句：SQL映射文件

   ```
   <select id="selectAllUsers" resultType="user">
       select * from users;
   </select>
   ```

3. 执行方法，测试
   ```

1.2 查询，查看详细：参数占位符、特殊字符处理

1. 编写接口方法：Mapper接口
   - 参数：id
   - 结果：User
   - ```
     User selectById(int id);
     ```
2. 编写SQL语句，SQL映射文件

```
<!-- parameterType指定参数类型，可省略 -->
<select id="selectById" parameterType="int" resultType="User">
    select * from users where id = #{id};
</select>
```

参数占位符：

- `#{}`：会将其替换为 '?'，为了放置SQL注入
- `${}`：拼SQL，会存在SQL注入问题
- 使用时机：
  - 参数传递的时候：`#{}`
  - 表名或列名不固定的情况下：`${}`，会存在SQL注入问题
- parameterType指定参数类型，可省略

特殊字符处理，如'<':

- 转义，`<` = `<`
- CDATA区，输入CD回车生成CDATA区

```
<![CDATA[
    <
]]>
```

3. 执行方法，测试

1.3 查询-多条件模糊查询：参数接收、三种参数接收方式

1. 边写接口方法：Mapper接口
   - 参数：所有查询条件
   - 结果：`List<Brand>`
   - **参数接收**：
     - 散装参数：如果方法中有多个参数，需要使用`@Param("SQL语句参数占位符名称")`，模糊查询前后要加上`"%"`

       UserMapper.java：

       ```
       List<User> selectByCondition(@Param("username")String username,
       @Param("phoneNumber")String phoneNumber);
       ```

       传参：
       ```

```
String username = "admin";
String phoneNumber = "1";

username = "%" + username + "%";
phoneNumber = "%" + phoneNumber + "%";
```

- 对象参数：对象的属性名称要和参数占位符一致

```
User user = new User();
user.setUsername(username);
user.setPhoneNumber(phoneNumber);
```

- Map集合参数

```
Map map = new HashMap();
map.put("username", username);
map.put("phoneNumber", phoneNumber);
```

2. 编写SQL语句：SQL映射文件，条件查询

```
<select id="selectByCondition" resultMap="userResultMap">
    select *
    from users
    where username like #{userName}
    and phone_number like #{phoneNumber}
</select>
```

3. 执行方法，测试

1.4 查询，多条件动态查询：mybatis动态SQL标签

改写SQL语句：

```
<select id="selectByCondition" resultMap="userResultMap">
    select *
    from users
    <where>
        <if test="userName != null and userName != '' ">
            username like #{userName}
        </if>
        <if test="phone_number != null and phone_number != '' ">
            phone_number like #{phoneNumber}
        </if>
    </where>
</select>
```

2. 添加

2.1 添加

1. 编写接口方法：Mapper接口
    - 参数：除了id之外的所有数据
    - 结果：void

- ```
  void addUser(User user)
  ```

2. 编写SQL语句：SQL映射文件

```
<insert id="addUser">
    insert into users (username, password, role, email, phone_number, sex)
    values (#{username}, #{password}, #{role}, #{email}, #{phoneNumber}, #
{sex});
</insert>
```

3. 执行方法、测试

   MyBatis事务：

   - `openSession()`：默认开启事务，进行增删改查操作后需要使用`sqlSession.commit();`**手动提交事务**
   - `openSession(true)`：可以设置为自动提交事务（关闭事务）

2.2 添加-主键返回

在数据添加成功后，需要获取插入数据库数据的主键的值

如：添加订单和订单项

1. 添加订单
2. 添加订单项，订单项中需要设置所属订单的id

解决办法：

使用`useGeneratedKeys="true" keyProperty="id"`属性，keyProperty的值为指向的列，就能使用**实体类的get方法**获取对应的值了

如：

```
<insert id="add" useGeneratedKeys="true" keyProperty="id">
</insert>
```

3. 修改

3.1 修改全部字段

1. 编写接口方法：Mapper接口
   - 参数：所有数据
   - 结果：void
   - ```
     void update(User user);
     ```

2. 编写SQL语句：SQL映射文件

```
<update id="updateUser">
    update users
    set
        username = #{username},
        password = #{password}
    where id = #{id}
</update>
```

3. 执行方法，测试

3.2 修改动态字段

SQL语句修改：

```
<update id="updateUser">
    update users
    <set>
        <if test="username != null and username != ''">
            username = #{username},
        </if>
        <if test="password != null and password != ''">
            password = #{password},
        </if>
        <if test="email != null and email != ''">
            email = #{email},
        </if>
        <if test="phoneNumber != null and phoneNumber != ''">
            phoneNumber = #{phoneNumber},
        </if>
            <if test="sex != null">
            sex = #{sex}
        </if>
    </set>
    where id = #{id}
</update>
```

4. 删除

4.1 根据id单个删除

1. 编写接口方法：Mapper接口

```
void deleteById(User user);
```

2. 编写SQL语句：SQL映射文件

```
<delete id="deleteById">
    delete from users where id = #{id}
</delete>
```

4.2 批量删除-`<foreach>`标签

MyBatis会将数组参数，封装为一个Map集合

- 默认叫`array`（=数组）
- 也可在接口方法中使用`@Param("")`注解改变名称，如下

注意：

- 由于SQL中的`in`后需要加括号，可使用foreach标签的`open="("`和`close=")"`进行拼接
- 使用`separator=","`属性拼接逗号

1. 编写接口方法：Mapper接口

   ```
   void deleteByIds(@Param("ids")int[] ids);
   ```

   - 参数：id数组
   - 结果：void

2. 编写SQL语句：SQL映射文件

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

MyBatis参数传递

结论：全使用`@Param("")`注解

单个参数：

- POJO类型：直接使用，属性名和参数占位符`#{}`中的名称保持一致
- Map集合：直接使用，键名和参数占位符`#{}`中的名称保持一致
- Collection：
  1. 判断是否为Collection，创建Map集合，put键`collection`，值为传递的参数
  2. 判断是否为List，put键`list`，值为传递的参数
  3. put键`arg0`，值传递的值
  4. 返回map
  5. 否则，put键`array`，值为传递的值，put`arg0`，值为传递的参数

  即：
  - `map.put("arg0", collection集合)`
  - `map.put("collection", collection集合)`
- List：
  - `map.put("arg0", list集合)`
  - `map.put("collection", list集合)`

    - `map.put("list", list集合)`
- Array：
    - `map.put("arg0", 数组)`
    - `map.put("array", 数组)`
- 其他类型：直接使用

多个参数：

- 接口方法中的`@Param`注解

1. 多个参数的封装

多个参数：将多个参数封装为**Map键值对集合**，可直接使用

- `map.put("arg0", 参数值1)`表示第一个参数的键
- `map.put("param1", 参数值1)`表示第一个参数的值
- `map.put("arg1", 参数值2)`表示第二个参数的键
- `map.put("param2", 参数值2)`表示第二个参数的值

如果可在SQL语句中使用`#{param1}`、`#{param2}`代替`#{}`中的参数，不推荐

推荐使用`@Param`注解，`@Param`注解：

- 该注解是arg[]数组顺序进行替换键的名称，比如只使用一个`@Param("username")`会替换arg0为`map.put("username", 参数1)`，其他保持原样

注解完成增删改查

注解完成简单功能，xml配置文件完成复杂功能

查询：`@Select`

添加：`@Insert`

修改：`@Update`

删除：`@Delete`