

MyBatis框架

一、MyBatis简介

1、MyBatis概述

MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code, 并且改名为MyBatis。2013年11月迁移到Github。

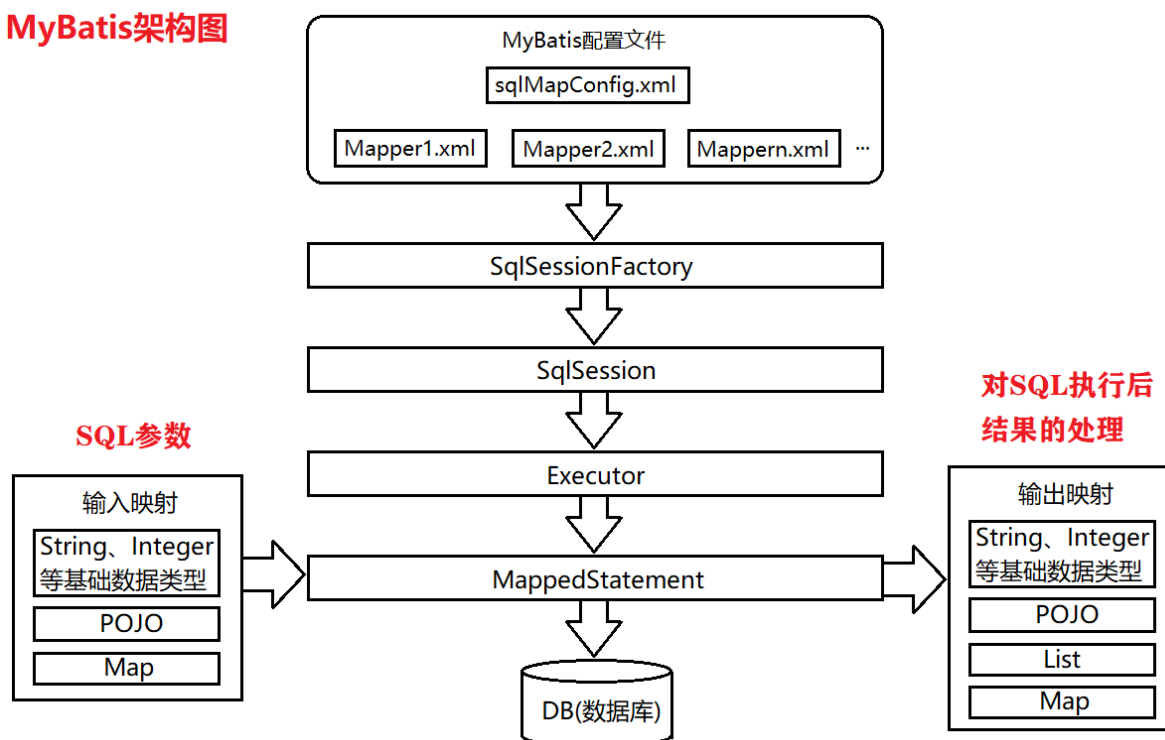
MyBatis是一个优秀的**持久层框架**, 它对jdbc的操作数据库的过程进行封装, 使开发者只需要关注SQL 本身, 而不需要花费精力去处理例如注册驱动、创建connection、创建statement、手动设置参数、结果集检索等jdbc繁杂的过程代码。

Mybatis通过xml或注解的方式将要执行的各种statement (statement、preparedStatement) 配置起来, 并通过java对象和statement中的sql进行映射生成最终执行的sql语句, 最后由mybatis框架执行sql并将结果映射成java对象并返回

总结: mybatis对JDBC访问数据库进行了封装, 简化了JDBC操作, 解决了jdbc将结果映射为Java对象的麻烦

2、MyBatis架构图

MyBatis架构图



- **mybatis-config.xml**是mybatis的核心配置文件, 通过配置文件 (核心配置文件和映射配置文件) 可以生成SqlSessionFactory, 也就是SqlSession工厂
- 基于SqlSessionFactory可以生成SqlSession对象
- SqlSession可以发送sql语句去执行, 得到返回结果, 类似JDBC中的connection, 是mybatis中至关重要的类
- Executor是SqlSession底层实现, 用来执行sql语句
- MappedStatement也是SqlSession底层实现, 用来接收输入映射 (也就是sql语句中的参数), 将查询的结果映射为相应的结果

二、为什么要使用MyBatis框架

思考：通过JDBC查询数据库表tb_user中的所有记录，将查询结果封装到List<User>并返回

```
1 public class JdbcTest {
2     // 通过JDBC查询数据库表tb_user中的所有记录，将查询结果封装到List<User>并返回
3     public List<User> findAll() {
4         // 声明变量
5         List<User> userList = new ArrayList<>();
6
7         Connection conn = null;
8         PreparedStatement ps = null;
9         ResultSet rs = null;
10
11         try {
12             // 1、注册驱动
13             Class.forName("com.mysql.cj.jdbc.Driver");
14
15             // 2、获取连接
16             String url = "jdbc:mysql:///hbnu?
serverTimezon=GMT&useSSL=false&characterEncoding=utf-8";
17             String username = "root";
18             String password = "chendikai";
19             conn = DriverManager.getConnection(url, username, password);
20
21             // 3、获取数据库操作对象
22             String sql = "select * from tb_user";
23             ps = conn.prepareStatement(sql);
24
25             // 4、执行sql语句
26             rs = ps.executeQuery();
27
28             // 5、处理查询结果集
29             while (rs.next()) {
30                 User user = new User();
31
32                 String username = rs.getString("username");
33                 String password = rs.getString("password");
34                 String email = rs.getString("email");
35
36                 user.setUsername(username);
37                 user.setPassword(password);
38                 user.setEmail(email);
39
40                 userList.add(user);
41             }
42
43             return userList;
44         } catch (SQLException e) {
45             e.printStackTrace();
46         } finally {
47             // 6、释放数据库资源
48             rs.close();
49             ps.close();
50             conn.close();
51
52         }
```

```
53     }  
54 }
```

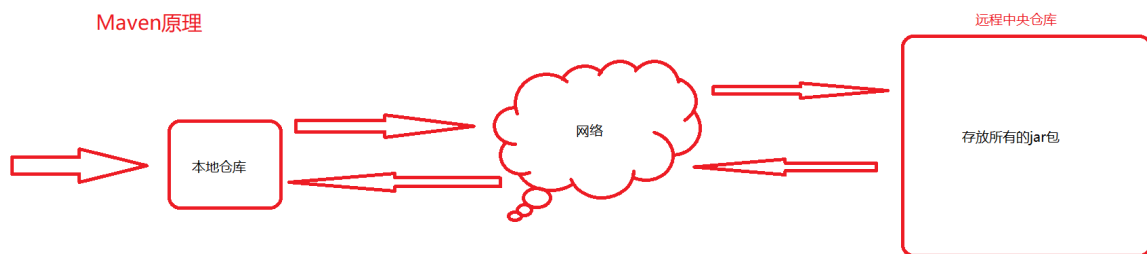
- JDBC访问数据库存在大量重复代码（比如注册驱动、获取连接、释放资源等等）
- JDBC自身不支持数据库连接池，会频繁创建连接、断开连接，这个操作比较耗资源，效率低
- sql语句是写在程序里面的，一旦sql语句发送改变，需要重新编译类
- JDBC对于结果集的处理，需要手动进行处理，有时候比较麻烦

使用MyBatis框架访问数据库

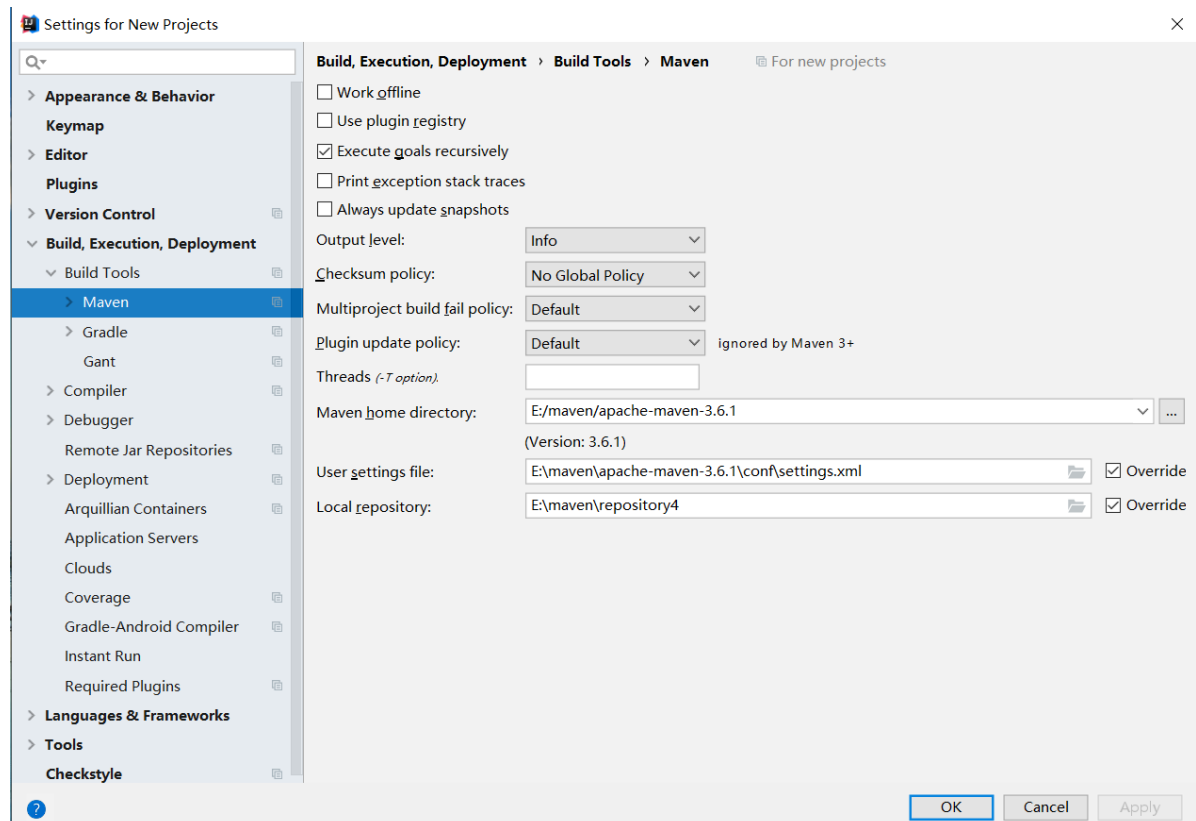
- mybatis框架对JDBC访问数据库的过程进行了封装，简化了JDBC操作
- mybatis自身支持数据库连接池（还可以配置其他的数据库连接池），提高效率
- mybatis中的sql语句是在mapper配置文件中，修改sql语句只需要修改配置文件就可以了，不需要重新编译类
- mybatis对查询结果集进行了处理，可以自动将查询结果映射为相应的结果

三、MyBatis快速入门案例

1、Maven简单介绍

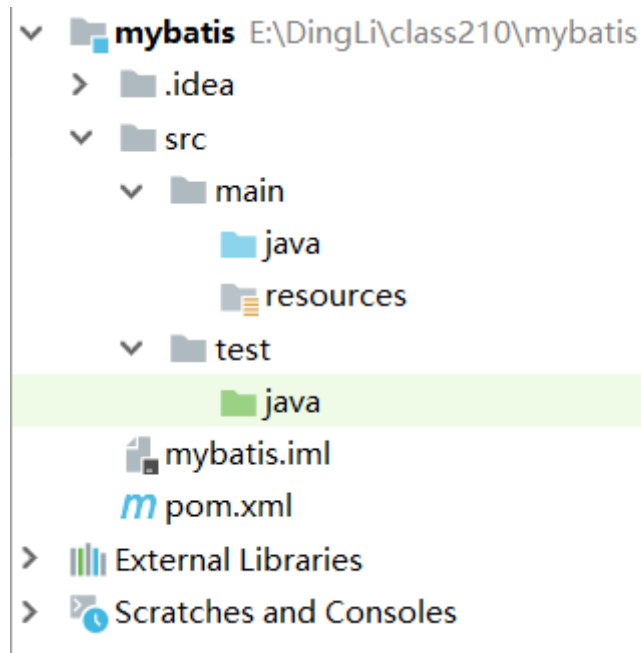


2、Maven在IDEA中的配置



3、创建Maven的简单Java项目

项目结构如下：



4、准备数据

准备数据库和数据库表

username	password	email
chendikai	123456789	chendikai@qq.com
孤独患者	789456	guduhuanzhe@163.com
湖师	hushi	hushi@qq.com
陌上杨花	chendikai	2086@qq.com

5、导入jar包

配置pom文件

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7     <groupId>com.hbnu</groupId>
8     <artifactId>mybatis</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <dependencies>
12         <!-- junit单元测试 -->
13         <dependency>
14             <groupId>junit</groupId>
15             <artifactId>junit</artifactId>
16             <version>4.9</version>
17         </dependency>
18         <!-- mysql驱动 -->
19         <dependency>
```

```

20         <groupId>mysql</groupId>
21         <artifactId>mysql-connector-java</artifactId>
22         <version>8.0.12</version>
23     </dependency>
24     <!-- mybatis -->
25     <dependency>
26         <groupId>org.mybatis</groupId>
27         <artifactId>mybatis</artifactId>
28         <version>3.2.8</version>
29     </dependency>
30     <!-- 整合log4j -->
31     <dependency>
32         <groupId>org.slf4j</groupId>
33         <artifactId>slf4j-log4j12</artifactId>
34         <version>1.6.4</version>
35     </dependency>
36
37 </dependencies>
38 </project>

```

6、创建实体类User

```

1  package com.hbnu.pojo;
2
3  /**
4   * @author 陈迪凯
5   * @date 2021-03-03 9:12
6   */
7  public class User {
8      private String username;
9      private String password;
10     private String email;
11
12     public String getUsername() {
13         return username;
14     }
15
16     public void setUsername(String username) {
17         this.username = username;
18     }
19
20     public String getPassword() {
21         return password;
22     }
23
24     public void setPassword(String password) {
25         this.password = password;
26     }
27
28     public String getEmail() {
29         return email;
30     }
31
32     public void setEmail(String email) {
33         this.email = email;
34     }
35

```

```

36     @Override
37     public String toString() {
38         return "User{" +
39             "username='" + username + '\'' +
40             ", password='" + password + '\'' +
41             ", email='" + email + '\'' +
42             '}';
43     }
44 }

```

7、编写映射配置文件UserMapper.xml

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE mapper
3      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <mapper namespace="UserMapper">
6      <select id="findAll" resultType="com.hbnu.pojo.User">
7          select * from tb_user
8      </select>
9
10 </mapper>

```

8、编写核心配置文件mybatis-config.xml

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE configuration
3      PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-config.dtd">
5  <configuration>
6      <environments default="develop">
7          <environment id="develop">
8              <transactionManager type="JDBC"></transactionManager>
9              <dataSource type="POOLED">
10                 <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
11                 <property name="url" value="jdbc:mysql:///hbnu?
serverTimezone=GMT&useSSL=false&characterEncoding=utf-8"/>
12                 <property name="username" value="root"/>
13                 <property name="password" value="chendikai"/>
14             </dataSource>
15         </environment>
16     </environments>
17
18     <mappers>
19         <mapper resource="UserMapper.xml"/>
20     </mappers>
21 </configuration>
22

```

9、测试

```

1  package com.hbnu.pojo;
2
3  import org.apache.ibatis.io.Resources;
4  import org.apache.ibatis.session.SqlSession;

```

```

5  import org.apache.ibatis.session.SqlSessionFactory;
6  import org.apache.ibatis.session.SqlSessionFactoryBuilder;
7  import org.junit.Test;
8
9  import java.io.IOException;
10 import java.io.InputStream;
11 import java.util.List;
12
13 /**
14  * @author 陈迪凯
15  * @date 2021-03-10 8:14
16  */
17 public class MyBatisTest {
18
19     @Test
20     public void findAll() throws IOException {
21         // 1、通过mybatis-config.xml核心配置文件构建SqlSessionFactory
22         InputStream in = Resources.getResourceAsStream("mybatis-
config.xml");
23
24         // 2、构建工厂SqlSessionFactory
25         SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(in);
26
27         // 3、通过SqlSessionFactory构建sqlSession对象，用于发送sql语句去执行，获取
返回结果
28         SqlSession sqlSession = sqlSessionFactory.openSession();
29
30         // 4、执行sql语句
31         String sqlId = "UserMapper.findAll";
32         List<User> userList = sqlSession.selectList(sqlId);
33
34         for (User user : userList) {
35             System.out.println(user);
36         }
37
38     }
39 }

```

四、Mybatis配置文件细节

1、核心配置文件

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE configuration
3      PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-config.dtd">
5  <!-- mybatis全局配置 -->
6  <configuration>
7      <!-- 配置环境，可以配置多个环境，比如开发环境、测试环境、生产环境-->
8      <environments default="develop">
9          <!-- 指定的环境 -->
10         <environment id="develop">
11             <!-- 事务管理配置
12             JDBC:推荐使用，可以进行事务的自动管理
13             MANAGED:不推荐使用，需要手动进行管理事务

```

```

14      -->
15      <transactionManager type="JDBC"></transactionManager>
16      <!-- 数据源配置
17      JNDI:已过时，不推荐使用
18      POOLED:使用数据库连接池
19      UNPOOLED:不使用数据库连接池
20      -->
21      <dataSource type="POOLED">
22          <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
23          <property name="url" value="jdbc:mysql:///hbnu?
serverTimezone=GMT&useSSL=false&characterEncoding=utf-8"/>
24          <property name="username" value="root"/>
25          <property name="password" value="chendikai"/>
26      </dataSource>
27  </environment>
28
29  </environments>
30
31  <!-- 引入mapper配置文件，可以引入多个pmapper配置文件 -->
32  <mappers>
33      <mapper resource="UserMapper.xml"/>
34  </mappers>
35  </configuration>

```

- environments:这个标签可以配置多个环境，比如测试、开发、生产环境，每个环境可以配置不同的信息，或者连接不同的数据库，**实际使用中，只能选择一个环境**
- transactionManager:事务管理配置，mybatis有两个参数JDBC/MANAGED
 - JDBC:采用JDBC的自动事务提交和回滚设置，通过数据源的连接来管理事务的范围，推荐使用
 - MANAGED:手动管理事务，不推荐使用
- dataSource:配置数据源，type配置连接池，内建有三个值：JNDI、POOLED、UNPOOLED
 - JNDI:已过时，不推荐
 - POOLED:使用连接池，通过从连接池获取连接访问数据库，执行完后将连接返回给连接池
 - UNPOOLED:不适用连接池

2、映射配置文件

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE mapper
3      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <!-- namespace 命名空间，代码中通过namespace + id 来确定执行哪个sql语句 -->
6  <mapper namespace="UserMapper">
7      <!--insert、update、delete、select对应数据库的crud操作
8      resultType:简单数据类型（比如Integer、String、User），List<User>只需要写集合中元
素的类型
9      -->
10     <select id="findAll" resultType="com.hbnu.pojo.User">
11         select * from tb_user
12     </select>
13
14 </mapper>

```


扩展知识

链式调用

```
public class User {  
    private String username;  
    private String password;  
}  
  
User user = new User();  
user.setUsername("chendikai");  
user.setPassword("123456");
```

链式调用

```
public class User{  
    private String username;  
    private String password;  
}  
  
User user = new User();  
user.setUsername("chendikai").setPassword("123456");
```

- 创建person类

```
1  package com.hbnu.pojo;  
2  
3  /**  
4   * @author 陈迪凯  
5   * @date 2021-03-17 9:01  
6   */  
7  public class Person {  
8  
9      private String username;  
10  
11     private String password;  
12  
13     public String getUsername() {  
14         return username;  
15     }  
16  
17     public Person setUsername(String username) {  
18         this.username = username;  
19         return this;  
20     }  
21  
22     public String getPassword() {  
23         return password;  
24     }  
25  
26     public Person setPassword(String password) {  
27         this.password = password;  
28  
29         return this;  
30     }  
31 }
```

- 测试

```

1 package com.hbnu.pojo;
2
3 /**
4  * @author 陈迪凯
5  * @date 2021-03-17 9:03
6  */
7 public class TestPerson {
8     public static void main(String[] args) {
9         Person person = new Person();
10        person.setUsername("chendikai").setPassword("123456");
11    }
12 }

```

五、Mybatis执行增删改查

1、添加用户信息

- 修改映射配置文件

```

1 <!-- 1、添加用户信息 -->
2 <insert id="insert">
3     insert into tb_user(username, password, email, salary) values ('铠',
4     '123456', 'kai@163.com',10000)
5 </insert>

```

- 测试

```

1 @Test
2 public void testInsert() throws IOException {
3     InputStream in = Resources.getResourceAsStream("mybatis-
4     config.xml");
5     SqlSessionFactory sqlSessionFactory = new
6     SqlSessionFactoryBuilder().build(in);
7     SqlSession sqlSession = sqlSessionFactory.openSession();
8
9     int rows = sqlSession.insert("UserMapper.insert");
10
11     sqlSession.commit();
12
13     System.out.println("影响的数据行数: " + rows);
14 }

```

2、修改用户信息

- 修改映射配置文件

```

1 <!-- 2、修改用户信息 -->
2 <update id="update">
3     update tb_user set password = '987654' where username = '铠'
4 </update>

```

- 测试

```

1  @Test
2  public void testUpdate() throws IOException {
3      InputStream in = Resources.getResourceAsStream("mybatis-
    config.xml");
4
5      SqlSessionFactory sqlSessionFactory = new
    SqlSessionFactoryBuilder().build(in);
6
7      SqlSession sqlSession = sqlSessionFactory.openSession();
8
9      int rows = sqlSession.update("UserMapper.update");
10
11     sqlSession.commit();
12
13     System.out.println("影响的数据行数: " + rows);
14 }

```

3、删除用户信息

- 修改映射配置文件

```

1  <!-- 3、删除用户信息 -->
2  <delete id="delete">
3      delete from tb_user where username = '铠'
4  </delete>

```

- 测试

```

1  @Test
2  public void testDelete() throws IOException {
3      InputStream in = Resources.getResourceAsStream("mybatis-
    config.xml");
4
5      SqlSessionFactory sqlSessionFactory = new
    SqlSessionFactoryBuilder().build(in);
6
7      SqlSession sqlSession = sqlSessionFactory.openSession();
8
9      int rows = sqlSession.delete("UserMapper.delete");
10
11     sqlSession.commit();
12
13     System.out.println("影响的数据行数: " + rows);
14 }

```

4、查询用户信息

- 修改映射配置文件

```

1  <!-- 4、查询指定用户信息 -->
2  <select id="selectByUsername" resultType="com.hbnu.pojo.User">
3      select * from tb_user where username = 'chendikai'
4  </select>

```

- 测试

```

1  @Test
2  public void testSelectByUsername() throws IOException {
3      InputStream in = Resources.getResourceAsStream("mybatis-
4      config.xml");
5
6      SqlSessionFactory sqlSessionFactory = new
7      SqlSessionFactoryBuilder().build(in);
8
9      SqlSession sqlSession = sqlSessionFactory.openSession();
10
11     User user = sqlSession.selectOne("UserMapper.selectByUsername");
12
13     System.out.println(user);
14 }

```

六、#{ }占位符

通过以上的测试，sql语句的参数值都是写死在sql语句中的，而实际项目开发中，这些数据是用户输入的，因此我们需要用占位符来替代用户输入的数据，mybatis中的占位符`#{ }`

1、添加用户信息

- 修改映射配置文件

```

1  <!-- 使用占位符 -->
2  <!-- 1、添加用户信息 -->
3  <insert id="insert2">
4      insert into tb_user(username, password, email, salary) values (#{
5      username}, #{password}, #{email}, #{salary})
6  </insert>

```

- 测试

```

1  @Test
2  public void testInsert2() throws IOException {
3      InputStream in = Resources.getResourceAsStream("mybatis-
4      config.xml");
5
6      SqlSessionFactory sqlSessionFactory = new
7      SqlSessionFactoryBuilder().build(in);
8
9      SqlSession sqlSession = sqlSessionFactory.openSession();
10
11     User user = new User();
12     user.setUsername("铠");
13     user.setPassword("123456");
14     user.setEmail("kai@qq.com");
15     user.setSalary(80000.00);
16
17     int rows = sqlSession.insert("UserMapper.insert2", user);
18
19     sqlSession.commit();
20
21     System.out.println("影响的数据行数: " + rows);
22 }

```

2、修改用户信息

- 修改映射配置文件

```
1 <!-- 2、修改用户信息 -->
2 <update id="update">
3     update tb_user set password = #{password} where username = #
4     {username}
5 </update>
```

- 测试

```
1 @Test
2 public void testUpdate2() throws IOException {
3     InputStream in = Resources.getResourceAsStream("mybatis-
4     config.xml");
5     SqlSessionFactory sqlSessionFactory = new
6     SqlSessionFactoryBuilder().build(in);
7     SqlSession sqlSession = sqlSessionFactory.openSession();
8
9     User user = new User();
10    user.setUsername("chendikai");
11    user.setPassword("chendikai");
12    int rows = sqlSession.update("UserMapper.update2", user);
13
14    sqlSession.commit();
15
16    System.out.println("影响的数据库行数: " + rows);
17 }
```

3、删除用户信息

- 修改映射配置文件

```
1 <!-- 3、删除用户信息 -->
2 <delete id="delete2">
3     delete from tb_user where username = #{username}
4 </delete>
```

- 测试

```
1 @Test
2 public void testDelete2() throws IOException {
3     InputStream in = Resources.getResourceAsStream("mybatis-
4     config.xml");
5     SqlSessionFactory sqlSessionFactory = new
6     SqlSessionFactoryBuilder().build(in);
7     SqlSession sqlSession = sqlSessionFactory.openSession();
8
9     int rows = sqlSession.delete("UserMapper.delete2", "孤独患者");
10
11    sqlSession.commit();
```

```

12
13     System.out.println("影响的数据库行数: " + rows);
14 }

```

4、查询指定用户信息

- 修改映射配置文件

```

1 <!-- 4、查询指定用户信息 -->
2 <select id="select2" resultType="com.hbnu.pojo.User">
3     select * from tb_user where username = #{username}
4 </select>

```

- 测试

```

1 @Test
2 public void testSelect2() throws IOException {
3     InputStream in = Resources.getResourceAsStream("mybatis-
4     config.xml");
5     SqlSessionFactory sqlSessionFactory = new
6     SqlSessionFactoryBuilder().build(in);
7     SqlSession sqlSession = sqlSessionFactory.openSession();
8
9     User user = sqlSession.selectOne("UserMapper.select2", "chendikai");
10
11     System.out.println(user);
12 }

```

七、\${}占位符

上面的增删改查操作，当SQL语句包含传过来的参数值时，使用#{ }进行占位，当真正执行sql语句时，再将传过来的参数值替换占位符，#{ }占位符实际上就是JDBC中的问号（?）。也就是参数值传过来后，会对这个参数值进行转译处理，这个是为了安全考虑。

思考：如果传过来的参数值是一个sql语句片段，这个时候不能用#{ }占位符

```

1 select 列名? ? ? from tb_user

```

这个时候可以使用\${ }占位符进行占位

```

1 select ${cols} from tb_user

```

示例：查询tb_user表中所有用户的用户名和邮箱

- 修改映射配置文件

```

1 <!-- 使用${ }占位符 -->
2 <select id="select3" resultType="com.hbnu.pojo.User">
3     select ${cols} from tb_user
4 </select>

```

- 测试

```

1  @Test
2  public void testSelect3() throws IOException {
3      InputStream in = Resources.getResourceAsStream("mybatis-
      config.xml");
4
5      SqlSessionFactory sqlSessionFactory = new
      SqlSessionFactoryBuilder().build(in);
6
7      SqlSession sqlSession = sqlSessionFactory.openSession();
8
9      Map<String, String> map = new HashMap<>();
10     map.put("cols", "username, email");
11     List<User> userList = sqlSession.selectList("UserMapper.select3",
      map);
12
13     for (User user : userList) {
14         System.out.println(user.getUsername() + ":" + user.getEmail());
15     }
16 }

```

八、动态SQL

1、if、where

- mybatis中的if元素用来判断字段值是否为空，针对判断接口，如果不为空，则sql语句会执行if元素包含的sql片段
- where用于选择执行片段，需要时可以剔除sql中的连接词（and、or），还可以在需要时增加sql关键字where



需求：根据用户输入的minSal（最低价格）和maxSal（最高价格）从tb_user表中查询用户信息

- 如果用户输入了minSal，则查询所有salary大于minSal的用户信息
- 如果用户输入了maxSal，则查询所有salary小于maxSal的用户信息
- 如果用户输入了minSal和maxSal，则查询所有介于minSal和maxSal之间的用户信息

1.1、修改映射配置文件

```

1  <!-- 动态SQL语句 -->
2  <!-- 1、if、where元素 -->
3  <select id="select4" resultType="com.hbnu.pojo.User">
4      select * from tb_user
5      <where>
6          <if test="minSal != null">
7              salary > #{minSal}
8          </if>
9          <if test="maxSal != null">
10             and salary <![CDATA[<]]> #{maxSal}
11          </if>
12      </where>
13  </select>

```

1.2、测试

```
1  @Test
2  public void testSelect4() throws IOException {
3      InputStream in = Resources.getResourceAsStream("mybatis-config.xml");
4
5      SqlSessionFactory sqlSessionFactory = new
6      SqlSessionFactoryBuilder().build(in);
7
8      SqlSession sqlSession = sqlSessionFactory.openSession();
9
10     Map<String, Double> map = new HashMap<>();
11     map.put("minSal", 2000.00);
12     map.put("maxSal", 80000.00);
13
14     List<User> userList = sqlSession.selectList("UserMapper.select4", map);
15
16     for (User user : userList) {
17         System.out.println(user);
18     }
19 }
```
