

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。

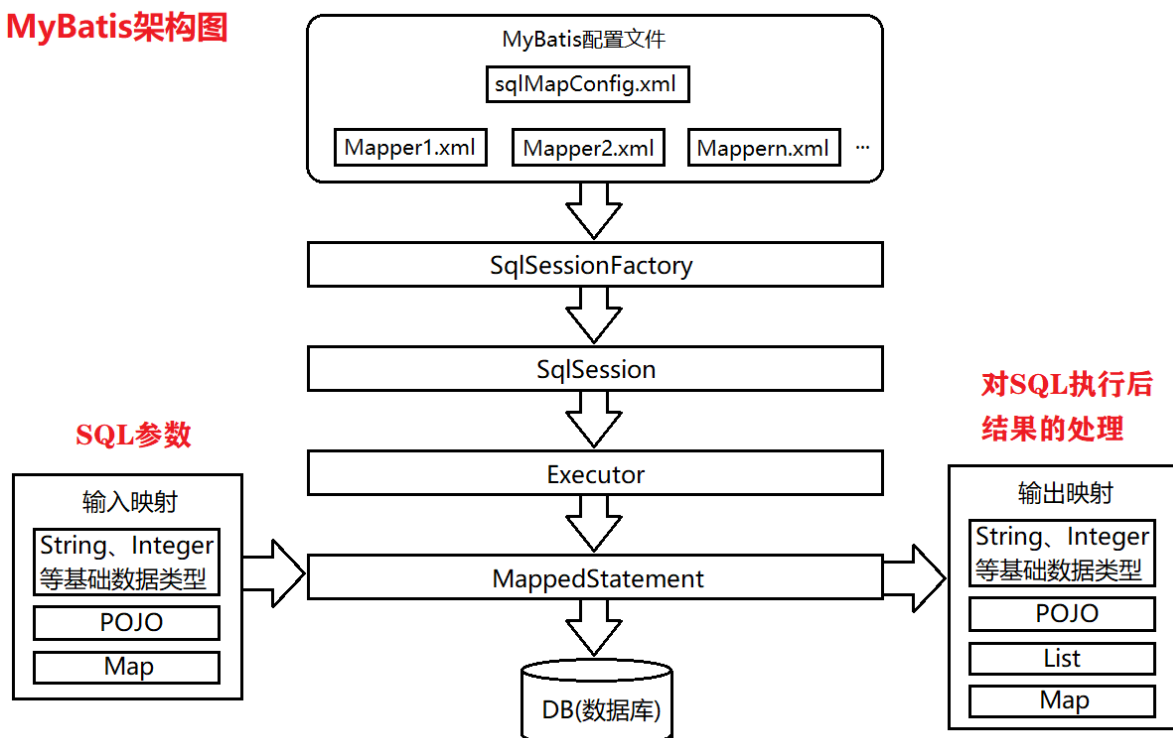
MyBatis是一个优秀的持久层框架, 它对jdbc的操作数据库的过程进行封装, 使开发者只需要关注SQL 本身, 而不需要花费精力去处理例如注册驱动、创建connection、创建statement、手动设置参数、结果集检索等jdbc繁杂的过程代码。

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。

总之, mybatis框架对jdbc访问数据库的过程进行了封装, 简化了jdbc操作, 对JDBC查询的结果进行了自动处理

mybatis架构图:

MyBatis架构图



- mybatis-config.xml核心配置文件, 通过核心配置文件可以生成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, 接收返回结果, 类似于jdbc中的connection, SqlSession很重要的类
- Executor用于执行sql语句, 是sqlsession底层实现
- MappedStatement也是SqlSession底层实现, 用于接收输入映射 (也就是sql语句中的参数), 返回相对于的结果

2、为什么要用mybatis框架

思考：通过JDBC查询数据库表tb_user的所有记录，封装到List<User>集合返回

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

- 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不支持数据库连接池，会频繁获取链接、关闭连接，效率低
- sql语句是写在程序里面的，一旦修改sql语句，需要重新编译程序
- jdbc中查询结果集的处理，需要手动进行处理，有时候会比较麻烦

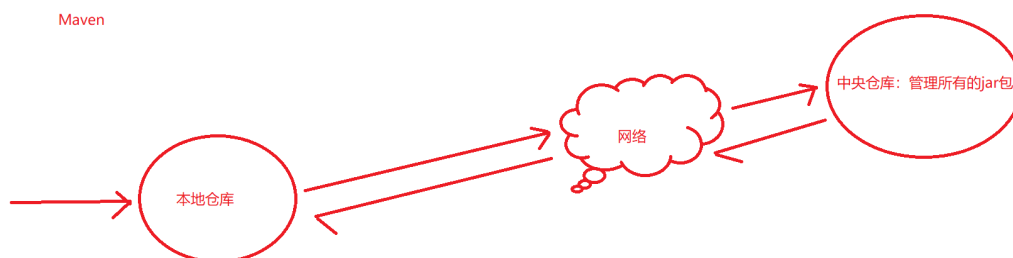
mybatis框架优点：

- mybatis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的（可以配置其他的连接池），提高效率
- mybatis中的sql语句是写在mapper配置文件里，如果需要修改sql语句，只需要修改配置文件就行，不需要重新编译程序
- mybatis对查询结果集能做到自动处理

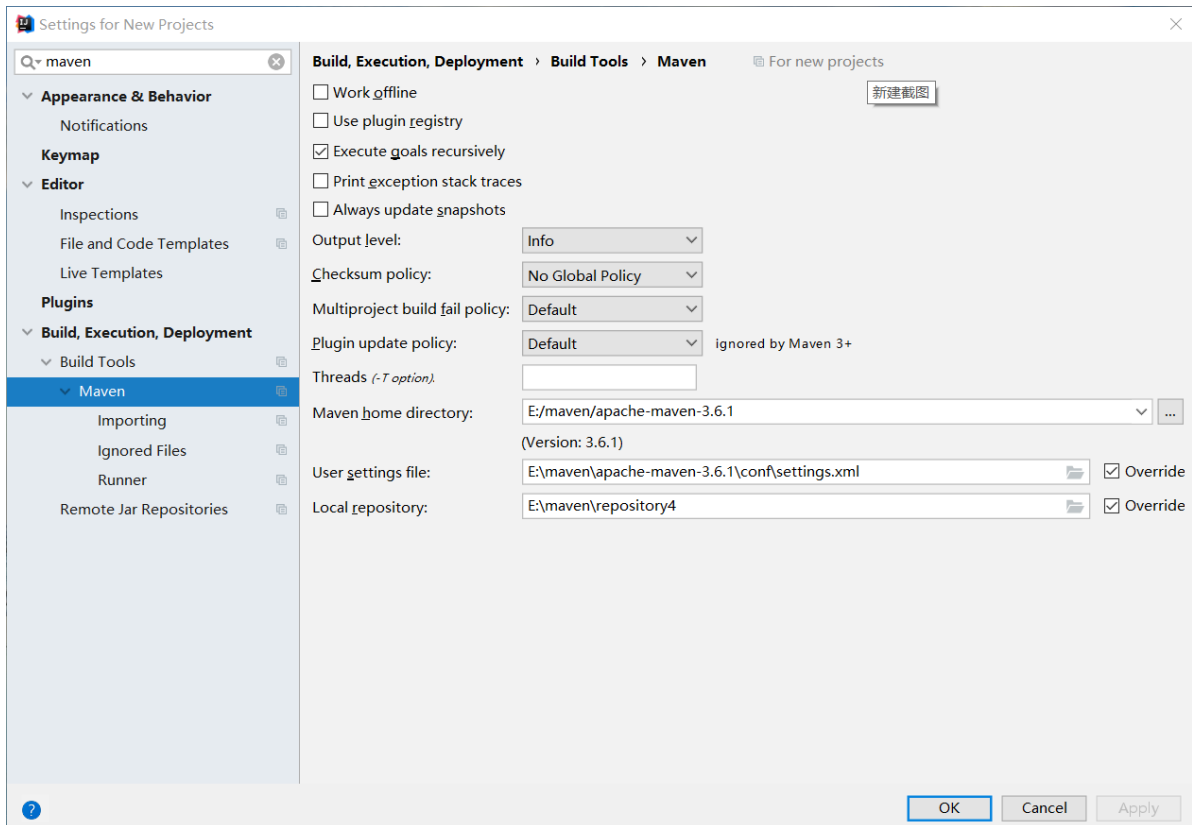
总之，mybatis几乎可以解决jdbc中存在的所有问题

二、Mybatis快速入门

1、Maven原理

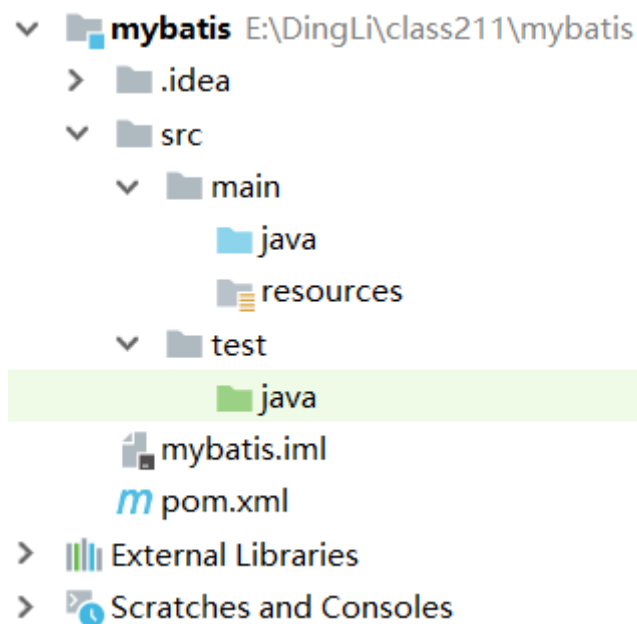


2、IDEA中配置maven



3、创建maven的普通Java项目

项目结构如下：



导入jar包，通过pom文件配置

```

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
8     <groupId>com.hbnu</groupId>
9     <artifactId>mybatis</artifactId>
10    <version>1.0-SNAPSHOT</version>

```

```

10     <dependencies>
11         <!-- junit单元测试 -->
12         <dependency>
13             <groupId>junit</groupId>
14             <artifactId>junit</artifactId>
15             <version>4.9</version>
16         </dependency>
17         <!-- mysql驱动 -->
18         <dependency>
19             <groupId>mysql</groupId>
20             <artifactId>mysql-connector-java</artifactId>
21             <version>8.0.12</version>
22         </dependency>
23         <!-- mybatis -->
24         <dependency>
25             <groupId>org.mybatis</groupId>
26             <artifactId>mybatis</artifactId>
27             <version>3.2.8</version>
28         </dependency>
29         <!-- 整合log4j -->
30         <dependency>
31             <groupId>org.slf4j</groupId>
32             <artifactId>slf4j-log4j12</artifactId>
33             <version>1.6.4</version>
34         </dependency>
35     </dependencies>
36 </project>

```

4、准备数据，建数据库和数据库表

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、创建实体类User.java

```

1  package com.hbun.pojo;
2
3  /**
4   * @author 陈迪凯
5   * @date 2021-03-02 9:16
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 }

```

6、创建映射配置文件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 </mapper>

```

7、Mybatis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://localhost:3306/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"></mapper>
20 </mappers>
21 </configuration>

```

8、测试

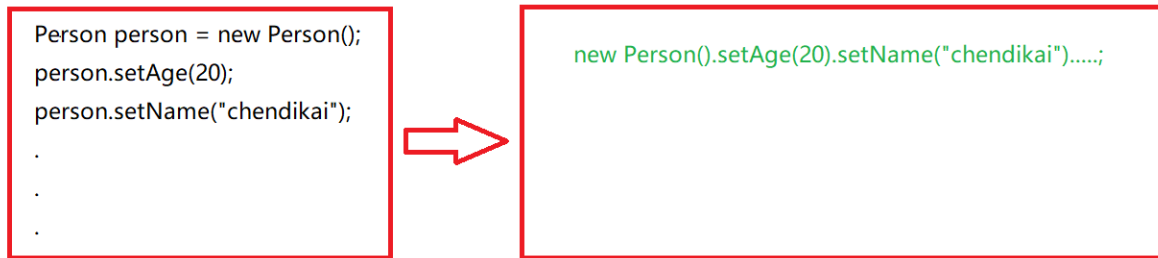
```

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 2021-03-09 8:08
16  */
17 public class MyBatisTest {
18
19     @Test
20     public void findAll() throws IOException {
21         // 1、读取核心配置文件mybatis-config.xml
22         InputStream resource = Resources.getResourceAsStream("mybatis-
config.xml");
23
24         // 2、通过配置信息创建SqlSessionFactory
25         SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(resource);
26
27         // 3、通过sqlSessionFactory构建sqlSession,sql可以发送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         // 打印结果
35         for (User user : userList) {
36             System.out.println(user);
37         }
38     }
39 }

```

扩展知识：链式调用

链式调用



- 创建Person类

```
1 package com.hbnu.pojo;
2
3 /**
4  * @author 陈迪凯
5  * @date 2021-03-09 8:29
6  */
7 public class Person {
8     private String username;
9     private String password;
10    private String gender;
11    private String email;
12
13    public Person setUsername(String username) {
14        this.username = username;
15        return this;
16    }
17
18    public Person setPassword(String password) {
19        this.password = password;
20        return this;
21    }
22
23    public Person setGender(String gender) {
24        this.gender = gender;
25        return this;
26    }
27
28    public Person setEmail(String email) {
29        this.email = email;
30        return this;
31    }
32
33    @Override
34    public String toString() {
35        return "Person{" +
36            "username='" + username + '\'' +
37            ", password='" + password + '\'' +
38            ", gender='" + gender + '\'' +
39            ", email='" + email + '\'' +
40            '}';
41    }
42 }
```

- 测试


```

1 package com.hbnu.pojo;
2
3 /**
4  * @author 陈迪凯
5  * @date 2021-03-09 8:37
6  */
7 public class Main {
8     public static void main(String[] args) {
9         Person person = new Person();
10
11         person.setUsername("chendikai").setGender("男").setPassword("123456").setEmail("chendikai1314@163.com");
12         System.out.println(person);
13     }
14 }

```

三、Mybatis入门细节

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 <!-- 配置mybatis全局信息 -->
6 <configuration>
7     <!--配置环境信息，内部可以配置多个环境信息，比如开发环境、测试环境、生产环境-->
8     <environments default="develop">
9         <environment id="develop">
10             <!-- 事务管理器配置
11             JDBC:自动管理事务，推荐使用
12             MANAGED:手动管理事务
13             -->
14             <transactionManager type="JDBC"></transactionManager>
15             <!-- 数据源配置
16             JNDI:已过时，不推荐
17             POOLED: 使用数据库连接池
18             UNPOOLED: 不适用数据库连接池
19             -->
20             <dataSource type="POOLED">
21                 <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
22                 <property name="url"
23                     value="jdbc:mysql://localhost:3306/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     </environments>
29
30     <!-- 配置映射文件，可以配置多个 -->
31     <mappers>
32         <mapper resource="UserMapper.xml"></mapper>
33     </mappers>
34 </configuration>

```

- environments:环境配置标签, 可以配置多个不同环境, 比如开发、测试、生产环境, 每个环境可以有不同的配置和连接不同的数据库, **但是最终使用的只能是一个环境**
- transactionManager:事务管理配置标签, 可以配置mybatis的事务管理方式, 有两种方式JDBC/MANAGED
 - JDBC:使用JDBC的自动事务提交和回滚方式, 依赖数据库连接来管理事务的范围, 推荐使用
 - MANAGED:这种方式不主动提交和回滚事务, 需要手动进行管理
- dataSource:配置数据源信息, 其中type配置的是mybatis中的数据库连接池, 有三种方式JNDI/POOLED/UNPOOLED
 - JNDI:已过时废弃, 不推荐使用
 - POOLED:使用数据库连接池, 操作数据库时, 从连接池中获取数据库连接, 操作完成后, 将连接放回连接池
 - UNPOOLED:不适用数据库连接池
- mappers:配置映射文件, 可以配置多个映射文件

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是当前文件所在的包名+当前文件名, 后面程序中通过namespace + id来执行sql
   语句 -->
6 <mapper namespace="UserMapper">
7     <!-- select、update、delete、insert这些标签可用于数据库的crud操作
8         resultType:结果集类型, List<User>
9         resultMap:复杂的结果类型 (多表联合查询)
10        resultType和resultMap不能同时存在
11    -->
12    <select id="findAll" resultType="com.hbnu.pojo.User">
13        select * from tb_user
14    </select>
15 </mapper>

```

四、MyBatis执行数据库的CRUD操作

1、增加用户信息

- 修改映射配置文件, 新增insert标签

```

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

```

- 测试

```

1 @Test
2 public void insertUser() throws IOException {
3     InputStream resource = Resources.getResourceAsStream("mybatis-
4     config.xml");
5     SqlSessionFactory sqlSessionFactory = new
6     SqlSessionFactoryBuilder().build(resource);

```

```

6
7     sqlSession sqlSession = sqlSessionFactory.openSession();
8
9     String sqlId = "UserMapper.insert";
10    int rows = sqlSession.insert(sqlId);
11
12    sqlSession.commit();
13
14    System.out.println("影响的数据: " + rows);
15
16 }

```

2、修改用户信息

- 修改映射配置文件，新增update标签

```

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

```

- 测试

```

1 @Test
2 public void updateUser() throws IOException {
3     InputStream resource = Resources.getResourceAsStream("mybatis-
4     config.xml");
5
6     sqlSessionFactory sqlSessionFactory = new
7     sqlSessionFactoryBuilder().build(resource);
8
9     sqlSession sqlSession = sqlSessionFactory.openSession();
10
11    String sqlId = "UserMapper.update";
12    int rows = sqlSession.update(sqlId);
13
14    sqlSession.commit();
15
16    System.out.println("影响的数据: " + rows);
17 }

```

3、删除用户信息

- 修改映射配置文件，新增delete标签

```

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

```

- 测试

```

1 @Test
2 public void deleteUser() throws IOException {
3     InputStream resource = Resources.getResourceAsStream("mybatis-
4     config.xml");

```

```

4
5     SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(resource);
6
7     SqlSession sqlSession = sqlSessionFactory.openSession();
8
9     String sqlId = "UserMapper.delete";
10    int rows = sqlSession.delete(sqlId);
11
12    sqlSession.commit();
13
14    System.out.println("影响的数据: " + rows);
15 }

```

4、查询用户信息

- 修改映射配置文件，新增select标签

```

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

```

- 测试

```

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

```

五、#{ }占位符

在上面的练习中，sql语句的参数值都是写死在sql语句中的，但在实际开发中，sql语句参数值往往用户传过来的，因此我们需要将sql语句中的参数值用占位符替代，mybatis中的占位符使用`#{ }`

1、添加用户信息

- 修改映射配置文件

```

1 <!-- 使用占位符#{ }执行sql语句 -->
2 <insert id="insert2">
3     insert into tb_user(username, password, email) values (#{username}, #
{password}, #{email})
4 </insert>

```

- 测试

```
1  @Test
2  public void testInsert() 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("马云");
13     user.setPassword("maoyun");
14     user.setEmail("maoyun@aliyun.com");
15
16     // 执行添加操作
17     int row = sqlSession.insert("UserMapper.insert2", user);
18
19     sqlSession.commit();
20
21     System.out.println("影响的数据: " + row);
22 }
```

2、修改用户信息

- 修改映射配置文件

```
1  <update id="update2">
2      update tb_user set password = #{password} where username = #
3      {username}
4  </update>
```

- 测试

```
1  @Test
2  public void testUpdate() 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("马云");
13     user.setPassword("123456");
14
15     int rows = sqlSession.update("UserMapper.update2", user);
16
17     sqlSession.commit();
18
19     System.out.println("影响的行数: " + rows);
20 }
```

```
19  
20 }
```

3、查询用户信息

- 修改映射配置文件

```
1 <select id="findByUsername" resultType="com.hbnu.pojo.User">  
2     select * from tb_user where username = #{username}  
3 </select>
```

- 测试

```
1 @Test  
2 public void testSelect() 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     User user = sqlSession.selectOne("UserMapper.findByUsername", "马  
9         云");  
10  
11     System.out.println(user);  
12  
13 }
```

4、删除用户信息

- 修改映射配置文件

```
1 <delete id="deleteByUsername">  
2     delete from tb_user where username = #{username}  
3 </delete>
```

- 测试

```
1 @Test  
2 public void testDelete() 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.deleteByUsername", "马云");  
10  
11     sqlSession.commit();  
12  
13     System.out.println("影响的行数: " + rows);  
14 }
```

六、#{}占位符

在上面的增删改查练习中，当SQL语句中包含的参数值是传递过来的，在SQL语句中我们会通过 `#{}` 占位符进行占位，在SQL语句真正执行时，再将传递过来的值替换SQL语句中的占位符。其实，`#{}` 就是 JDBC 中的问号 (?) 占位符，因此为了安全考虑，在执行时会传递过来的值进行转译处理。

思考：那么如果我们在传递的时候不是一个参数值，而是SQL语句本身呢？例如在查询时，我们想动态的传递查询的列

```
1 | select 查询的列?? from tb_user
```

此时传递过来的应该是一个SQL片段，不同于上面的参数值，如果此时还用 `#{}`，也会像上面一样被转译处理，这不是我们希望看到的。如果不想让传过来的值被转译处理，那么这里可以使用 `${}`

```
1 | select ${columns} from tb_user
```

示例：查询tb_user表中所有员工的姓名和邮箱

- 修改映射配置文件

```
1 | <!-- 使用${}占位符指定要查询的字段 -->
2 | <select id="select2" resultType="com.hbnu.pojo.User">
3 |     select ${cols} from tb_user
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 |     Map<String, String> map = new HashMap<>();
10 |    map.put("cols", "username, password, email");
11 |
12 |    List<User> userList = sqlSession.selectList("UserMapper.select2",
13 |        map);
14 |    for (User user : userList) {
15 |        System.out.println("用户名:" + user.getUsername() + "; 邮箱: " +
16 |            user.getEmail() + "; 密码: "
17 |            + user.getPassword());
18 |    }
19 | }
```

需要注意的是，在传递 `${}` 对应的值时，需要将值存入 `map` 集合中！！

总结：在大多数情况下还是使用#{}占位符，而\${}多用于为不带引号的字符串进行占位！