# MyBatis框架

## 一、MyBatis简介

### 1、MyBatis概述

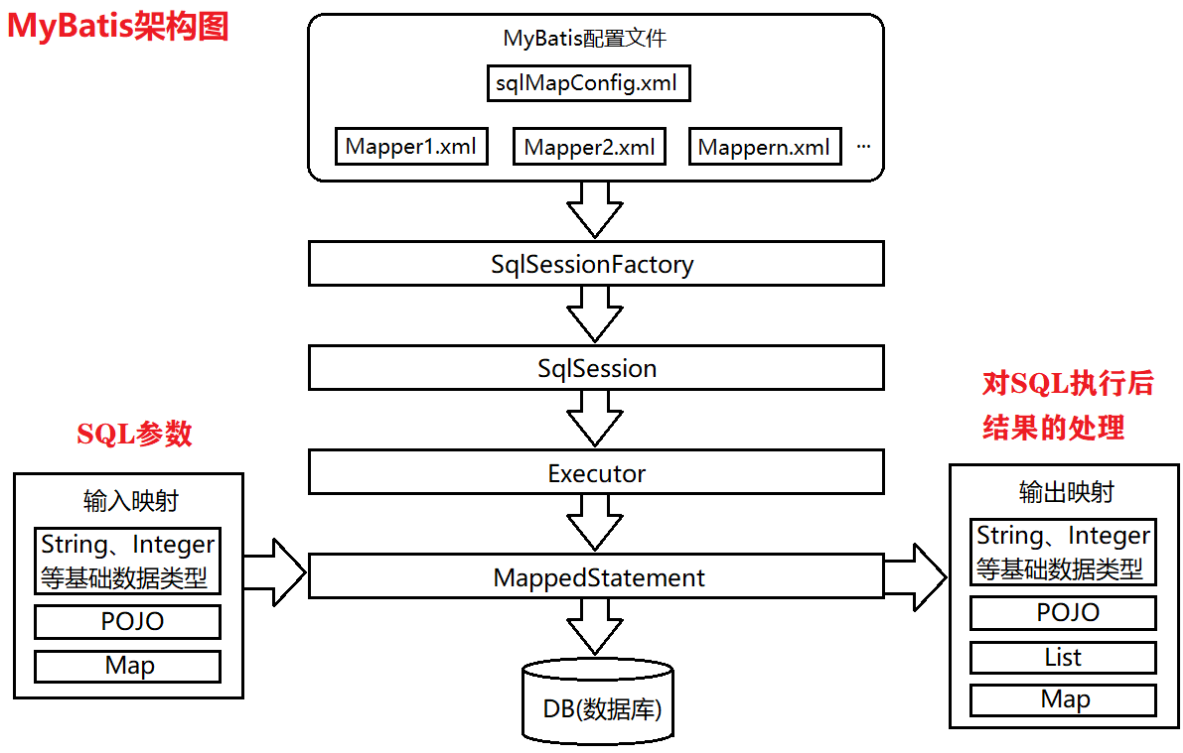
MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis 。2013年11月迁移到Github。

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、preparedStatemnt）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




- mybati-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SqlSesiion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

```
public class JdbcTest {
    // 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
    public List<User> findAll() {
        // 声明变量
        List<User> userList = new ArrayList<>();

        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;

        try {
            // 1、注册驱动
            Class.forName("com.mysql.cj.jdbc.Driver");

            // 2、获取连接
            String url = "jdbc:mysql:///hbnu?
serverTimezon=GMT&useSSL=false&characterEncoding=utf-8";
            String username = "root";
            String password = "chendikai";
            conn = DriverManager.getConnection(url, username, password);

            // 3、获取数据库操作对象
            String sql = "select * from tb_user";
            ps = conn.prepareStatement(sql);

            // 4、执行sql语句
            rs = ps.executeQuery();

            // 5、处理查询结果集
            while (rs.next()) {
                User user = new User();

                String username = rs.getString("username");
                String password = rs.getString("password");
                String email = rs.getString("email");

                user.setUsername(username);
                user.SetPassword(password);
                user.SetEmail(email);

                userList.add(user);
            }

            return userList;
        } catch(SQLException e) {
            e.printStackTrace();
        } finally {
            // 6、释放数据库资源
            rs.close();
            ps.close();
            conn.close();

        }
```

```
53      }
54  }
```

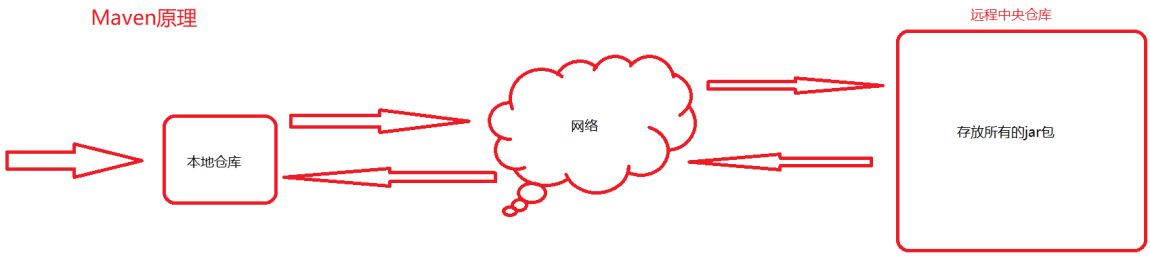
- JDBC访问数据库存在大量重复代码（比如注册驱动、获取连接、释放资源等等）
- JDBC自身不支持数据库连接池，会频繁创建连接、断开连接，这个操作比较耗资源，效率低
- sql语句是写在程序里面的，一旦sql语句发送改变，需要重新编译类
- JDBC对于结果集的处理，需要手动进行处理，有时候比较麻烦

**使用MyBatis框架访问数据库**

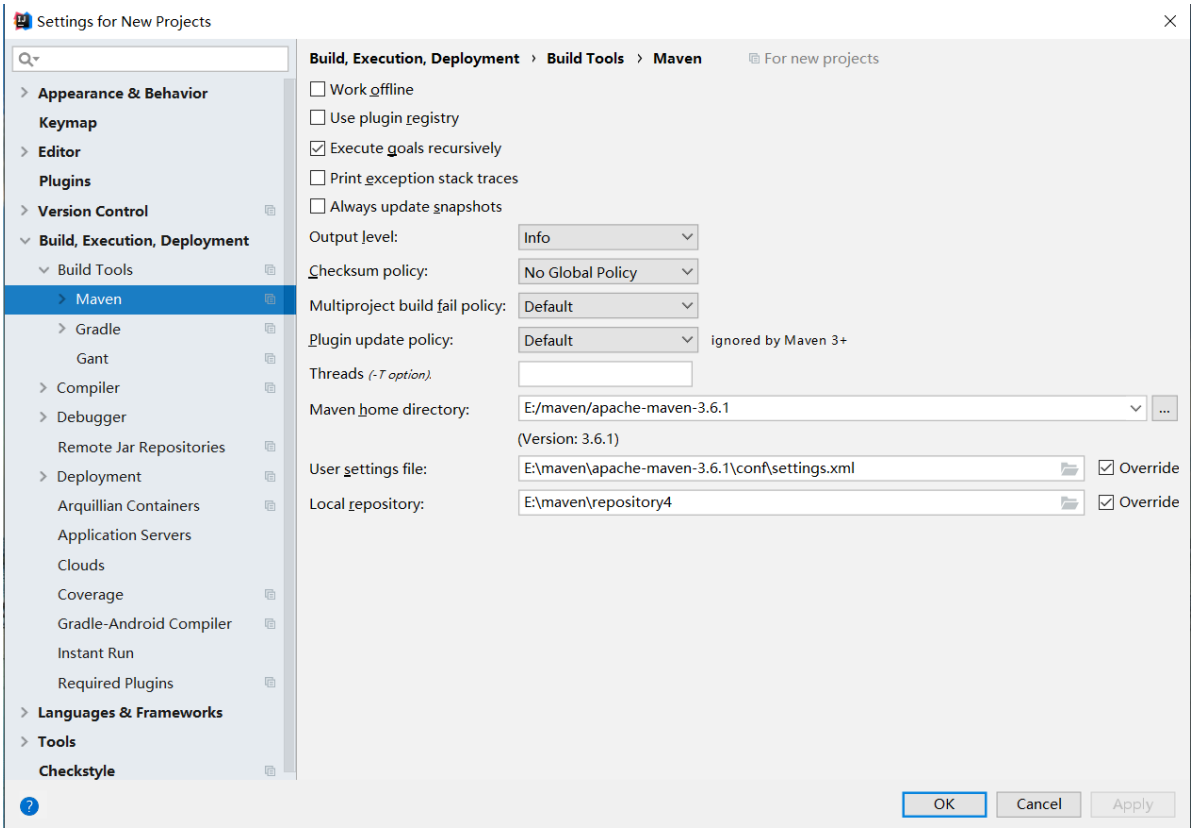
- mybatis框架对JDBC访问数据库的过程进行了封装，简化了JDBC操作
- mybatis自身支持数据连接池（还可以配置其他的数据库连接池），提高效率
- mybatis中的sql语句是在mapper配置文件中，修改sql语句只需要修改配置文件就可以了，不需要重新编译类
- mybatis对查询结果集进行了处理，可以自动将查询结果映射为相应的结果

## 三、MyBatis快速入门案例

### 1、Maven简单介绍

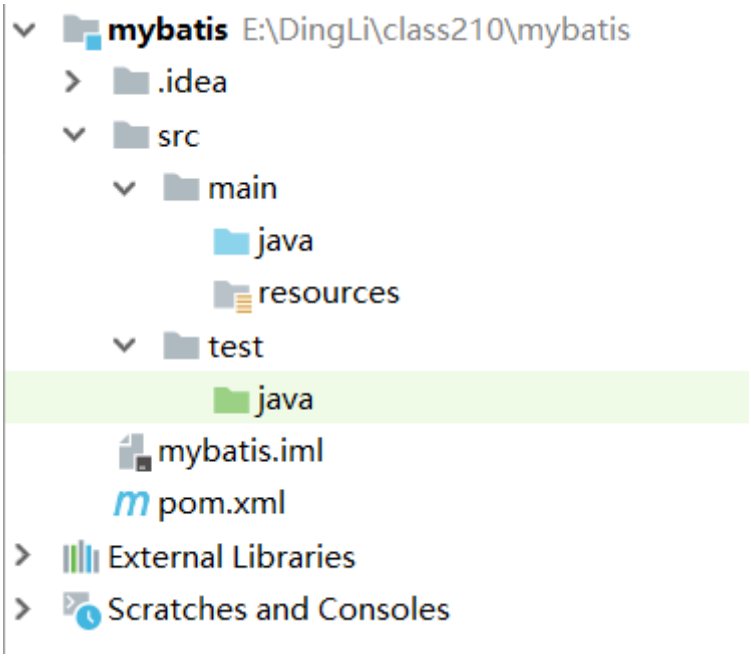



### 2、Maven在IDEA中的配置



### 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

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

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

    <dependencies>
        <!-- junit单元测试 -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.9</version>
        </dependency>
        <!-- mysql驱动 -->
        <dependency>
```

```
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.12</version>
        </dependency>
        <!-- mybatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.2.8</version>
        </dependency>
        <!-- 整合log4j -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.6.4</version>
        </dependency>

    </dependencies>
</project>
```

6、创建实体类User

```
package com.hbnu.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-03 9:12
 */
public class User {
    private String username;
    private String password;
    private String email;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

```

```
    @Override
    public String toString() {
        return "User{" +
                "username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", email='" + email + '\'' +
                '}';
    }
}
```

### 7、编写映射配置文件UserMapper.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="UserMapper">
    <select id="findAll" resultType="com.hbnu.pojo.User">
        select * from tb_user
    </select>

</mapper>
```

### 8、编写核心配置文件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>
    <environments default="develop">
        <environment id="develop">
            <transactionManager type="JDBC"></transactionManager>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql:///hbnu?
serverTimezone=GMT&amp;useSSL=false&amp;characterEncoding=utf-8"/>
                <property name="username" value="root"/>
                <property name="password" value="chendikai"/>
            </dataSource>
        </environment>

    </environments>

    <mappers>
        <mapper resource="UserMapper.xml"/>
    </mappers>
</configuration>
```

---

### 9、测试

```
package com.hbnu.pojo;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
```

```java
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
import org.junit.Test;

import java.io.IOException;
import java.io.InputStream;
import java.util.List;

/**
 * @author 陈迪凯
 * @date 2021-03-10 8:14
 */
public class MyBatisTest {

    @Test
    public void findAll() throws IOException {
        // 1、通过mybatis-config.xml核心配置文件构建SqlSessionFactory
        InputStream in = Resources.getResourceAsStream("mybatis-config.xml");

        // 2、构建工厂SqlSessionFactory
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);

        // 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
        SqlSession sqlSession = sqlSessionFactory.openSession();

        // 4、执行sql语句
        String sqlId = "UserMapper.findAll";
        List<User> userList = sqlSession.selectList(sqlId);

        for (User user : userList) {
            System.out.println(user);
        }

    }
}
```

## 四、Mybatis配置文件细节

### 1、核心配置文件

```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">
<!-- mybatis全局配置 -->
<configuration>
    <!-- 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-->
    <environments default="develop">
        <!-- 指定的环境 -->
        <environment id="develop">
            <!-- 事务管理配置
             JDBC:推荐使用，可以进行事务的自动管理
             MANAGED:不推荐使用，需要手动进行管理事务
```

```
             -->
            <transactionManager type="JDBC"></transactionManager>
            <!-- 数据源配置
            JNDI:已过时，不推荐使用
            POOLED:使用数据库连接池
            UNPOOLED:不使用数据库连接池
            -->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql:///hbnu?
serverTimezone=GMT&amp;useSSL=false&amp;characterEncoding=utf-8"/>
                <property name="username" value="root"/>
                <property name="password" value="chendikai"/>
            </dataSource>
        </environment>

    </environments>

    <!-- 引入mapper配置文件，可以引入多个pmapper配置文件 -->
    <mappers>
        <mapper resource="UserMapper.xml"/>
    </mappers>
</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**

```
<?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 命名空间，代码中通过namespace + id 来确定执行哪个sql语句 -->
<mapper namespace="UserMapper">
    <!--insert、update、delete、select对应数据库的crud操作
    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
    -->
    <select id="findAll" resultType="com.hbnu.pojo.User">
        select * from tb_user
    </select>

</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

```java
package com.hbnu.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-17 9:01
 */
public class Person {

    private String username;

    private String password;

    public String getUsername() {
        return username;
    }

    public Person setUsername(String username) {
        this.username = username;
        return this;
    }

    public String getPassword() {
        return password;
    }

    public Person setPassword(String password) {
        this.password = password;

        return this;
    }
}
```

- 测试

```java
package com.hbnu.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-17 9:03
 */
public class TestPerson {
    public static void main(String[] args) {
        Person person = new Person();
        person.setUsername("chendikai").setPassword("123456");
    }
}
```

## 五、Mybatis执行增删改查

### 1、添加用户信息

- 修改映射配置文件

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

- 测试

```java
@Test
public void testInsert() throws IOException {
    InputStream in = Resources.getResourceAsStream("mybatis-config.xml");

    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);

    SqlSession sqlSession = sqlSessionFactory.openSession();

    int rows = sqlSession.insert("UserMapper.insert");

    sqlSession.commit();

    System.out.println("影响的数据行数：" + rows);
}
```

### 2、修改用户信息

- 修改映射配置文件

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

- 测试

```java
@Test
public void testUpdate() throws IOException {
    InputStream in = Resources.getResourceAsStream("mybatis-config.xml");

    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);

    SqlSession sqlSession = sqlSessionFactory.openSession();

    int rows = sqlSession.update("UserMapper.update");

    sqlSession.commit();

    System.out.println("影响的数据行数：" + rows);
}
```

### 3、删除用户信息

- 修改映射配置文件

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

- 测试

```java
@Test
public void testDelete() throws IOException {
    InputStream in = Resources.getResourceAsStream("mybatis-config.xml");

    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);

    SqlSession sqlSession = sqlSessionFactory.openSession();

    int rows = sqlSession.delete("UserMapper.delete");

    sqlSession.commit();

    System.out.println("影响的数据行数：" + rows);
}
```

### 4、查询用户信息

- 修改映射配置文件

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

- 测试

```java
@Test
public void testSelectByUsername() throws IOException {
    InputStream in = Resources.getResourceAsStream("mybatis-config.xml");

    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);

    SqlSession sqlSession = sqlSessionFactory.openSession();

    User user = sqlSession.selectOne("UserMapper.selectByUsername");

    System.out.println(user);
}
```

## 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

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

- 测试

```java
@Test
public void testInsert2() throws IOException {
    InputStream in = Resources.getResourceAsStream("mybatis-config.xml");

    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);

    SqlSession sqlSession = sqlSessionFactory.openSession();

    User user = new User();
    user.setUsername("铠");
    user.setPassword("123456");
    user.setEmail("kai@qq.com");
    user.setSalary(80000.00);

    int rows = sqlSession.insert("UserMapper.insert2", user);

    sqlSession.commit();

    System.out.println("影响的数据行数：" + rows);
}
```

### 2、修改用户信息

- 修改映射配置文件

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

- 测试

```
@Test
public void testUpdate2() throws IOException {
    InputStream in = Resources.getResourceAsStream("mybatis-config.xml");

    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);

    SqlSession sqlSession = sqlSessionFactory.openSession();

    User user = new User();
    user.setUsername("chendikai");
    user.setPassword("chendikai");
    int rows = sqlSession.update("UserMapper.update2", user);

    sqlSession.commit();

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

### 3、删除用户信息

- 修改映射配置文件

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

- 测试

```
@Test
public void testDelete2() throws IOException {
    InputStream in = Resources.getResourceAsStream("mybatis-config.xml");

    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);

    SqlSession sqlSession = sqlSessionFactory.openSession();

    int rows = sqlSession.delete("UserMapper.delete2", "孤独患者");

    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-
   config.xml");
4
5      SqlSessionFactory sqlSessionFactory = new
   SqlSessionFactoryBuilder().build(in);
6
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

```java
@Test
public void testSelect3() throws IOException {
    InputStream in = Resources.getResourceAsStream("mybatis-config.xml");

    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(in);

    SqlSession sqlSession = sqlSessionFactory.openSession();

    Map<String, String> map = new HashMap<>();
    map.put("cols", "username, email");
    List<User> userList = sqlSession.selectList("UserMapper.select3", map);

    for (User user : userList) {
        System.out.println(user.getUsername() + ":" + user.getEmail());
    }
}
```

## 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

```xml
<!-- 动态SQL语句 -->
<!-- 1、if、where元素 -->
<select id="select4" resultType="com.hbnu.pojo.User">
    select * from tb_user
    <where>
        <if test="minSal != null">
            salary > #{minSal}
        </if>
        <if test="maxSal != null">
            and salary <![CDATA[<]]> #{maxSal}
        </if>
    </where>
</select>
```

### 1.2、测试

```
@Test
public void testSelect4() throws IOException {
    InputStream in = Resources.getResourceAsStream("mybatis-config.xml");

    SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(in);

    SqlSession sqlSession = sqlSessionFactory.openSession();

    Map<String, Double> map = new HashMap<>();
    map.put("minSal", 2000.00);
    map.put("maxSal", 80000.00);

    List<User> userList = sqlSession.selectList("UserMapper.select4", map);

    for (User user : userList) {
        System.out.println(user);
    }
}
```

---

## 2、set

- set标签用于对包含在set标签里面的sql语句进行筛选，在需要的时候可以剔除连接符（比如逗号），也可以添加set关键字

  需求：修改tb_usr表中指定用户名的用户信息，用户传了password、email、salary这些字段的值，则修改，否则不修改

### 2.1、修改映射配置文件

```
<!-- 2、set元素 -->
<update id="update2">
    update tb_user
    <set>
        <if test="password != null">password = #{password},</if>
        <if test="email != null">email = #{email},</if>
        <if test="salary != null">salary = #{salary}</if>
    </set>
    where username = #{username}
</update>
```

### 2.2、测试

```
@Test
public void testUpdateSet() throws IOException {

    InputStream inputStream = Resources.getResourceAsStream("mybatis-
config.xml");

    SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);

    SqlSession sqlSession = sqlSessionFactory.openSession();

```

```
    User user = new User();
    user.setUsername("chendikai");
    user.setPassword("123456");
    user.setSalary(30000.00);

    int rows = sqlSession.update("UserMapper.updateSet", user);

    sqlSession.commit();

    System.out.println("影响了数据行数：" + rows);
}
```

### 3、foreach

需求：查询指定用户的用户信息

#### 3.1、修改映射配置文件

```
<!-- 3、foreach元素 -->
<select id="selectForeach" resultType="com.hbnu.pojo.User">
    select * from tb_user where username in
    <foreach collection="array" open="(" close=")" item="username"
separator=",">
        #{username}
    </foreach>
</select>
```

#### 3.2、测试

```
@Test
public void testSelectForeach() throws IOException {

    InputStream inputStream = Resources.getResourceAsStream("mybatis-
config.xml");

    SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);

    SqlSession sqlSession = sqlSessionFactory.openSession();

   String[] usernames = {"chendikai", "陌上杨花"};

    List<User> userList = sqlSession.selectList("UserMapper.selectForeach",
usernames);

    for (User user : userList) {
        System.out.println(user);
    }
}
```

## 九、Mapper接口开发

## 1、mapper接口开发介绍

在上面的mybatis案例中，通过sqlSession对象调用用法进行CRUD操作，方法中的第一个参数是一个字符串，这个字符串的内容：（映射配置文件的）namespace + id，通过这种方式找到映射配置文件的sql语句并执行，这种方式由于传入的是字符串，很容易发生拼写错误，而且编译器不会进行编译检查。

企业开发中常用的方式是通过mapper接口进行操作数据库。使用mapper接口进行开发需要注意以下几点：

1. mapper接口的全路径名和映射配置文件的namespace值要保持一致
2. mapper文件中每条要执行的sql语句，在接口中都要有对应的方法，并且接口中的方法名必须和mapper文件中sql语句标签的id名一致
3. mapper接口中方法的接收参数类型和mapper文件中sql语句的参数类型保持一致
4. mapper接口的返回值要和mapper文件查询结果的返回值对应，如果接口中方法的返回值为list集合，那么mapper中的sql语句的resultType只需要写集合里面元素的类型即可

## 2、mapper接口开发实现

需求：根据用户名查询用户信息

### 2.1、创建UseMapper

```java
package com.hbnu.dao;

import com.hbnu.pojo.User;

/**
 * @author 陈迪凯
 * @date 2021-03-31 9:08
 */
public interface UserMapper {

    public User findUserByUsername(String username);
}
```

### 2.2、创建映射配置文件

```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 命名空间，代码中通过namespace + id 来确定执行哪个sql语句 -->
<mapper namespace="com.hbnu.dao.UserMapper">

    <select id="findUserByUsername" resultType="com.hbnu.pojo.User">
        select * from tb_user where username = #{username}
    </select>
</mapper>
```

### 2.3、测试

```java
@Test
public void testfindUserByUsername() throws IOException {

    InputStream inputStream = Resources.getResourceAsStream("mybatis-
config.xml");
```

```
    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

    SqlSession sqlSession = sqlSessionFactory.openSession();

    UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

    User user = userMapper.findUserByUsername("chendikai");

    System.out.println(user);
}
```

### 2.5、测试结果

```
"D:\Program Files\Java\jdk1.8.0_65\bin\java.exe" ...
log4j:WARN No appenders could be found for logger (org.apache.ibatis.logging.LogFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
User{username='chendikai', password='123456', email='chendikai1314@163.com', salary=30000.0}

Process finished with exit code 0
```

## 十、可以优化的地方

### 1、加入日志框架

实际项目开发中，通过添加日志框架，便于系统开发调试和系统维护，log4j日志配置文件log4j.properties放到resources目录下，日志配置文件内容：

```
log4j.rootLogger=DEBUG, Console

#Console
log4j.appender.Console=org.apache.log4j.ConsoleAppender
log4j.appender.Console.layout=org.apache.log4j.PatternLayout
log4j.appender.Console.layout.ConversionPattern=%d [%t] %-5p [%c] - %m%n

log4j.logger.java.sql.ResultSet=INFO
log4j.logger.org.apache=INFO
log4j.logger.java.sql.Connection=DEBUG
log4j.logger.java.sql.Statement=DEBUG
log4j.logger.java.sql.PreparedStatement=DEBUG
```

mybatis框架默认使用的是log4j这个日志框架，将日志配置文件log4j.properties放到指定目录，mybatis会自动到指定目录加载日志配置文件，读取配置文件里面的配置信息进行记录日志

### 2、加入数据库配置文件

#### 2.1、数据库配置文件内容

```
# 数据库驱动
jdbc.driver=com.mysql.cj.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/hbnu?serverTimezone=GMT&useSSL=false&characterEncoding=utf-8
jdbc.username=root
jdbc.password=chendikai
```

### 2.1、引入数据库配置文件

```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">
<!-- mybatis全局配置 -->
<configuration>

    <!-- 引入数据库配置文件 -->
    <properties resource="jdbc.properties"></properties>

    <!-- 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-->
    <environments default="develop">
        <!-- 指定的环境 -->
        <environment id="develop">
            <!-- 事务管理配置
             JDBC:推荐使用，可以进行事务的自动管理
             MANAGED:不推荐使用，需要手动进行管理事务
             -->
            <transactionManager type="JDBC"></transactionManager>
            <!-- 数据源配置
            JNDI:已过时，不推荐使用
            POOLED:使用数据库连接池
            UNPOOLED:不使用数据库连接池
            -->
            <dataSource type="POOLED">
                <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mapper配置文件，可以引入多个pmapper配置文件 -->
    <mappers>
        <mapper resource="UserMapper.xml"/>
        <mapper resource="UserMapper1.xml"/>
    </mappers>
</configuration>
```