MyBatis框架

一、MyBatis简介

1、MyBatis概述

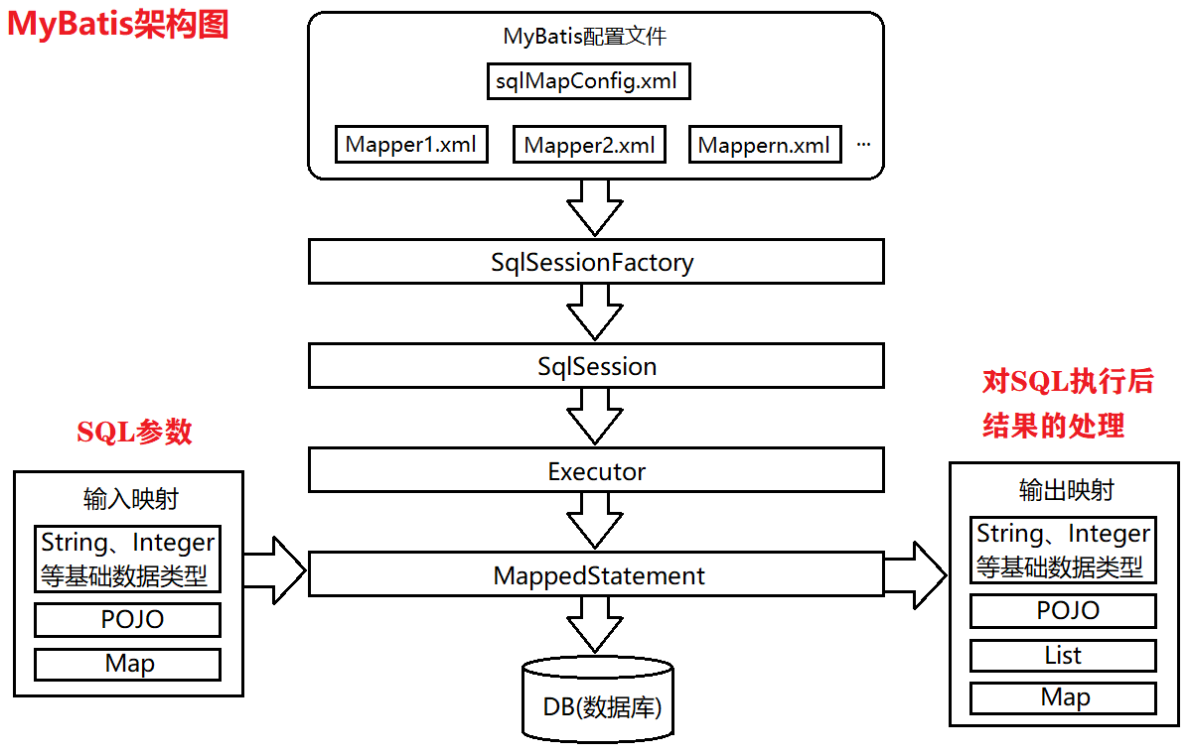
MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis 。2013年11月迁移到Github。

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

mybati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
- Excutor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

```java
public class JdbcTest {
    // 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
    public List<User> findAll() {
        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?serverTimezone=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、释放资源
            rs.close();
            ps.close();
            conn.close();
        }
    }
}
```

- 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

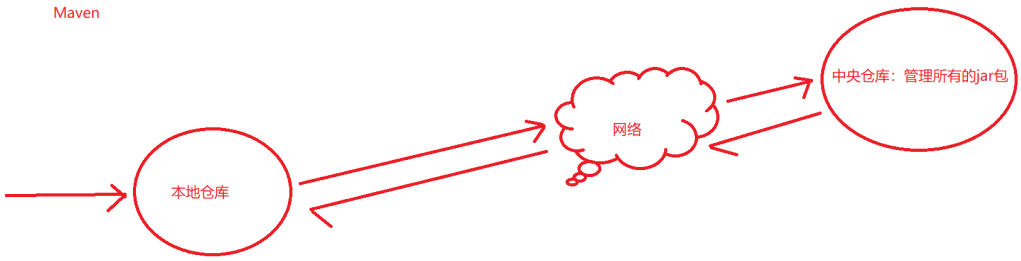
mybatis框架优点：

- 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

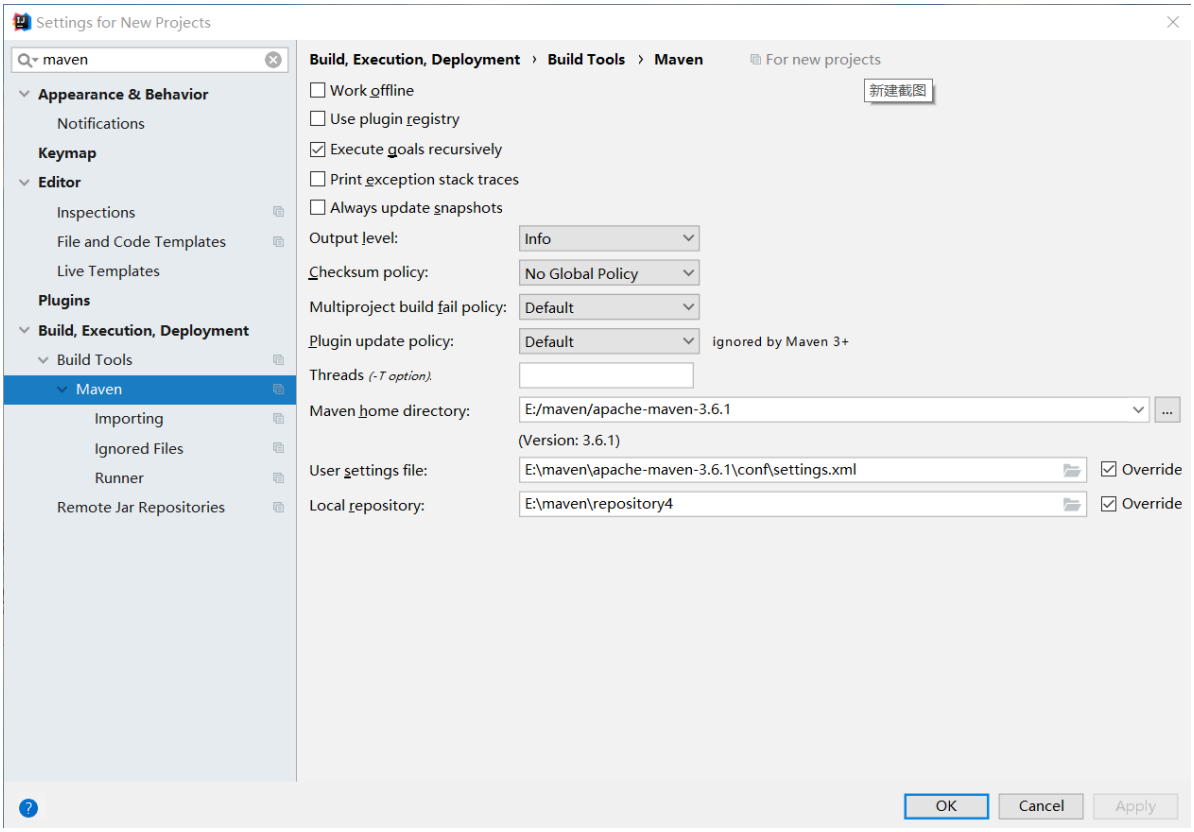
总之，mybatis几乎可以解决jdbc中存在的所有问题

二、Mybatis快速入门

1、Maven原理

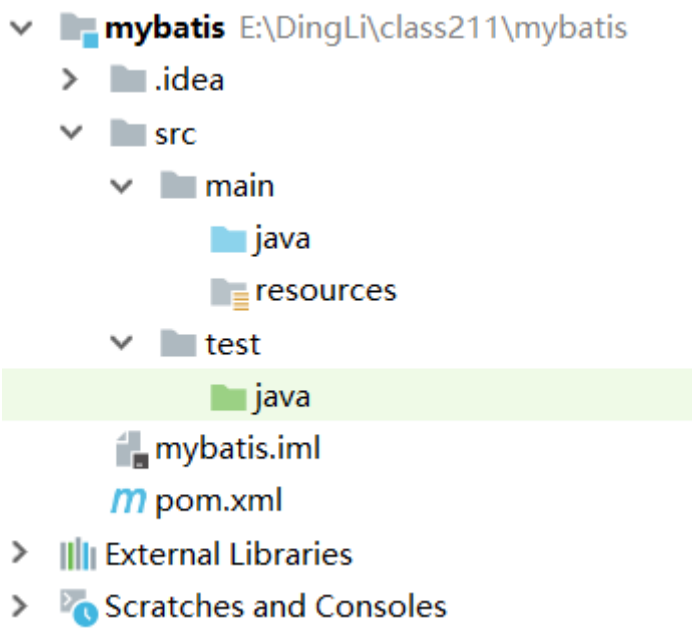


2、IDEA中配置maven



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

```
<?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>
```

```xml
    <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>
```

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

```java
package com.hbun.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-02 9:16
 */
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 + '\'' +
                '}';
    }
}
```

6、创建映射配置文件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>
```

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

8、测试

```
package com.hbnu.pojo;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
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 2021-03-09 8:08
 */
public class MyBatisTest {

    @Test
    public void findAll() throws IOException {
        // 1、读取核心配置文件mybatis-config.xml
        InputStream resource = Resources.getResourceAsStream("mybatis-config.xml");

        // 2、通过配置信息创建SqlSessionFactory
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(resource);

        // 3、通过sqlSessionFactory构建sqlSession,sql可以发送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);

        // 打印结果
        for (User user : userList) {
            System.out.println(user);
        }
    }
}
```

扩展知识：链式调用

- 创建Person类

```java
package com.hbnu.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-09 8:29
 */
public class Person {
    private String username;
    private String password;
    private String gender;
    private String email;

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

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

    public Person setGender(String gender) {
        this.gender = gender;
        return this;
    }

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

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

- 测试

```java
package com.hbnu.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-09 8:37
 */
public class Main {
    public static void main(String[] args) {
        Person person = new Person();

 person.setUsername("chendikai").setGender("男").setPassword("123456").setEmail("chendikai1314@163.com");
        System.out.println(person);
    }
}
```

三、Mybatis入门细节

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">
<!-- 配置mybatis全局信息 -->
<configuration>
    <!--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-->
    <environments default="develop">
        <environment id="develop">
            <!-- 事务管理器配置
            JDBC:自动管理事务，推荐使用
            MANAGED:手动管理事务
            -->
            <transactionManager type="JDBC"></transactionManager>
            <!-- 数据源配置
            JNDI:已过时，不推荐
            POOLED：使用数据库连接池
            UNPOOLED：不适用数据库连接池
            -->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
                <property name="url"
                          value="jdbc:mysql://localhost:3306/hbnu?serverTimezone=GMT&amp;useSSL=false&amp;characterEncoding=utf-8"/>
                <property name="username" value="root"/>
                <property name="password" value="chendikai"/>
            </dataSource>
        </environment>

    </environments>

    <!-- 配置映射文件，可以配置多个 -->
    <mappers>
        <mapper resource="UserMapper.xml"></mapper>
    </mappers>
</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

```
<?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 -->
<mapper namespace="UserMapper">
    <!-- select、update、delete、insert这些标签可用于数据库的crud操作
     resultType:结果集类型，List<User>
     resultMap:复杂的结果类型（多表联合查询）
     reultType和resultMap不能同时存在
     -->
    <select id="findAll" resultType="com.hbnu.pojo.User">
        select * from tb_user
    </select>
</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、添加用户信息 -->
<insert id="insert">
    insert into tb_user(username, password, email) values ('张无忌', '123456', 'zhangwuji@163.com')
</insert>
```

- 测试

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

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

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-
   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

```
@Test
public void testInsert() 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("马云");
    user.setPassword("maoyun");
    user.setEmail("maoyun@aliyun.com");

    // 执行添加操作
    int row = sqlSession.insert("UserMapper.insert2", user);

    sqlSession.commit();

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

2、修改用户信息

- 修改映射配置文件

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

- 测试

```
@Test
public void testUpdate() 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("马云");
    user.setPassword("123456");

    int rows = sqlSession.update("UserMapper.update2", user);

    sqlSession.commit();

    System.out.println("影响的行数：" + rows);
```

```
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-
   config.xml");
4
5      SqlSessionFactory sqlSessionFactory = new
   SqlSessionFactoryBuilder().build(in);
6
7      SqlSession sqlSession = sqlSessionFactory.openSession();
8
9      User user = sqlSession.selectOne("UserMapper.findByUsername", "马
   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-
   config.xml");
4
5      SqlSessionFactory sqlSessionFactory = new
   SqlSessionFactoryBuilder().build(in);
6
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，在执行时会对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

```
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 **${}**

```
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

```
<!-- 使用${}占位符指定要查询的字段 -->
<select id="select2" resultType="com.hbnu.pojo.User">
    select ${cols} from tb_user
</select>
```

- 测试

```
@Test
public void testSelect2() 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, password, email");

    List<User> userList = sqlSession.selectList("UserMapper.select2", map);

    for (User user : userList) {
        System.out.println("用户名:" + user.getUsername() + "；邮箱：" + user.getEmail() + "；密码："
        + user.getPassword());
    }

}
```

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！

七、动态SQL

1、if、where

- mybatis中的if表示根据字段的参数值是否为空来决定是否执行if元素里面的内容
- where元素用于对包含在其中的sql语句进行检索，需要时，可以剔除多余的连接词（比如and,or），还可以添加where关键词

需求：根据用户输入的minSal和maxSal动态执行sql语句

1. 如果minSal不为空，maxSal为空，查询所有大于minSal的用户信息
2. 如果minSal为空，maxSal不为空，查询所有小于maxSal的用户信息
3. 如果minSal和maxSal都不为空，查询介于minSal和maxSal之间的所有用户信息

- 在User类中添加salary属性

```java
package com.hbnu.pojo;

/**
 * @author 陈迪凯
 * @date 2021-03-02 9:16
 */
public class User {
    private String username;
    private String password;
    private String email;
    private Double salary;

    public Double getSalary() {
        return salary;
    }

    public void setSalary(Double salary) {
        this.salary = salary;
    }

    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 + '\'' +
                ", salary=" + salary +
                '}';
    }
}
```

- 配置映射配置文件

```
<select id="findBySalary" 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>
```

- 测试

```
@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, Double> salary = new HashMap<>();
    salary.put("minSal", 3000.00);
    salary.put("maxSal", 8001.00);

    List<User> userList = sqlSession.selectList("UserMapper.findBySalary", salary);

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

2、set

set元素用于对包含在其中的sql语句进行检索，需要时可以剔除连接符（比如逗号），也可以添加关键词set

需求：修改指定用户的信息，如果passwrod、email、salary不为空，则修改这些值

- 修改映射配置文件

```
<update id="updateByInput">
    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>
```

- 测试

```
@Test
public void testUpdateByInput() 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("123456");
    user.setEmail("chendikai1314@163.com");
    user.setSalary(20000.00);

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

    sqlSession.commit();

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

3、foreach

需求1：查询指定用户名的用户信息

- 修改映射配置文件

```
<select id="findByUsername1" resultType="com.hbnu.pojo.User">
    select * from tb_user where username in
    <foreach collection="array" open="(" close=")" item="username" separator=",">
        #{username}
    </foreach>
</select>
```

- 测试

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

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

    SqlSession sqlSession = sqlSessionFactory.openSession();

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

    List<User> users = sqlSession.selectList("UserMapper.findByUsername1", usernames);

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

需求：删除指定用户的用户信息

- 修改配置文件

```
<delete id="deleteByUsername1">
    delete from tb_user where username in
    <foreach collection="array" open="(" close=")" item="username" separator=",">
        #{username}
    </foreach>
</delete>
```

- 测试

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

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

    SqlSession sqlSession = sqlSessionFactory.openSession();

    String[] usernames = {"jixin"};

    int rows = sqlSession.delete("UserMapper.deleteByUsername1", usernames);

    sqlSession.commit();

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

八、mapper接口开发

1、mapper接口开发介绍

之前通过sqlSession对象调用里面的方法进行数据的CRUD操作，方法的第一个参数是一个字符串，这个字符串是映射配置文件的namespace + id，执行CURU操作时，通过namespace找到映射文件，再通过id找到对应的sql语句，mybatis框架底层执行sql语句。这种方式存在一点小问题，字符串容易拼写错误，错误的情形下不会进行编译检查。

mapper接口开发是企业开发中常用的方式，即使用mapper接口，使用mapper接口开发有几点需要注意：

1. mapper接口的全路径名要和映射配置文件（mapper配置文件）中的namespace值要保持一致
2. mapper配置文件中每一条要执行的sql语句，在mapper接口里面要添加一个对应的方法，这个方法名称要和sql语句标签的id值保持一致
3. mapper接口中方法接收参数要和mapper配置文件中sql的接收参数类型要一致
4. mapper接口中方法的返回值要和mapper配置文件中sql语句的返回值一致（mapper接口方法的返回值为list集合时，sql语句的返回值是list集合中元素的类型）

2、mapper接口开发案例

需求：根据username查询用户信息，使用mapper接口进行开发

1. 创建UserMapper接口

```java
package com.hbnu.dao;

/**
 * @author 陈迪凯
 * @date 2021-03-30 8:23
 */
public interface UserMapper {
}
```

2. 创建映射配置文件

   namespace值和接口全路径名保持一致

```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 -->
<mapper namespace="com.hbnu.dao.UserMapper">

</mapper>
```

3. 修改UserMapper接口

```java
package com.hbnu.dao;

import com.hbnu.pojo.User;

/**
 * @author 陈迪凯
 * @date 2021-03-30 8:23
```

```
 */
public interface UserMapper {

    /**
     * 根据用户名查询用户信息
     *
     * @param username 用户名
     * @return 用户信息
     */
    public User findUserByUsername(String username);
}
```

4. 修改UserMapper1.xml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 -->
<mapper namespace="com.hbnu.dao.UserMapper">

    <!-- 根据用户名查询用户信息 -->
    <select id="findUserByUsername" resultType="com.hbnu.pojo.User">
        select * from tb_user where username = #{username}
    </select>
</mapper>
```

5. 测试

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

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

    SqlSession sqlSession = sqlSessionFactory.openSession();

    // 通过sqlSession对象中的getMapper方法获取到mapper对象
    UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

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

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

6. 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='chendikai', email='chendikai1314@163.com', salary=20000.0}

Process finished with exit code 0
```

课堂练习：采用mapper接口开发，查询所有用户信息

九、优化

1、加入log4j日志

配置文件为log4j.properties，配置放在项目资源目录下

```
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
```

2、数据库配置文件

1. 数据库配置文件jdbc.properteis，配置文件放到资源目录下

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

2. 在mybatis核心配置文件中引入jdbc.properties配置文件

```
<?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">
<!-- 配置mybatis全局信息 -->
<configuration>

    <!-- 引入jdbc.properties数据库配置信息 -->
    <properties resource="jdbc.properties"></properties>

    <!--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-->
    <environments default="develop">
        <environment id="develop">
            <!-- 事务管理器配置
            JDBC:自动管理事务，推荐使用
            MANAGED:手动管理事务
            -->
            <transactionManager type="JDBC"></transactionManager>
            <!-- 数据源配置
            JNDI:已过时，不推荐
            POOLED：使用数据库连接池
            UNPOOLED：不适用数据库连接池
            -->
            <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>

    <!-- 配置映射文件，可以配置多个 -->
    <mappers>
        <mapper resource="UserMapper1.xml"></mapper>
        <mapper resource="UserMapper.xml"></mapper>
    </mappers>
</configuration>
```