

# Mybatis

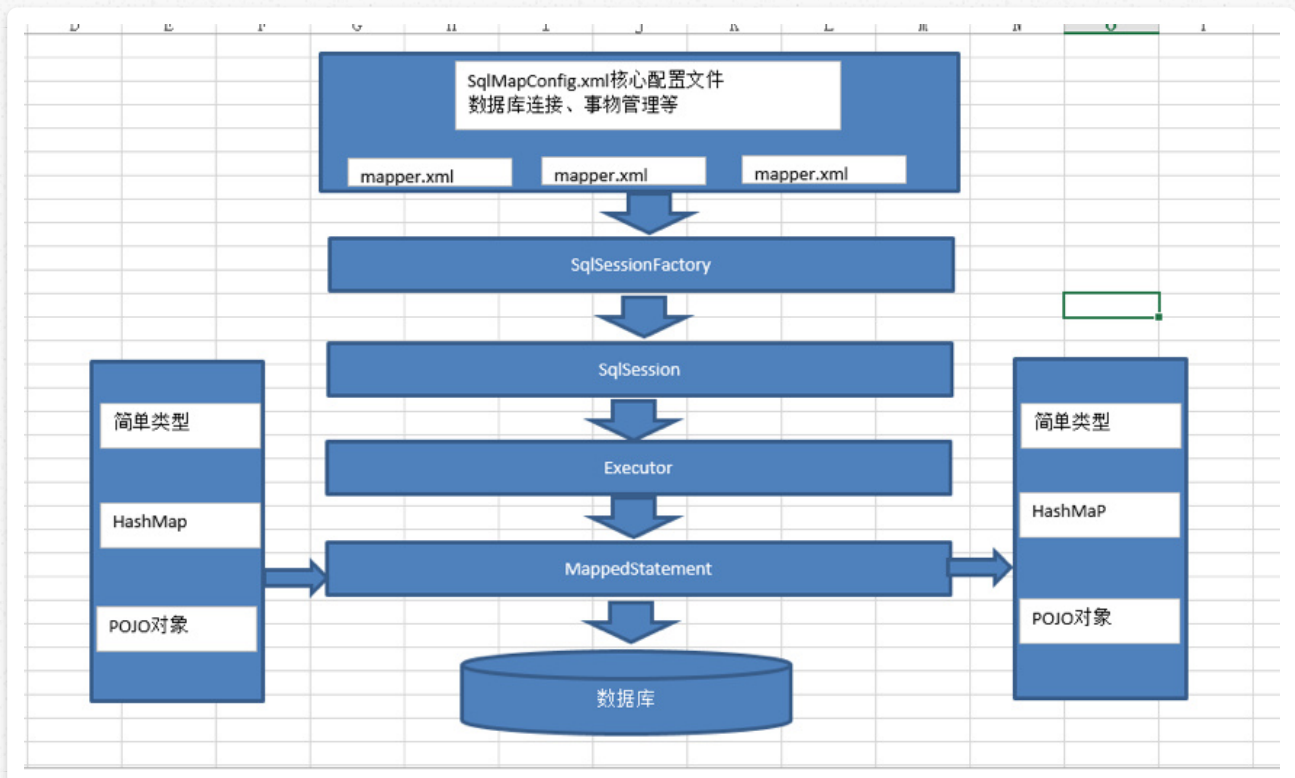
## 1.Mybatis是什么？

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, 实质上Mybatis对ibatis进行一些改进。

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、 CallableStatement) 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。

## 2.Mybatis框架原理（核心）



### 分析结论

- 1、mybatis配置文件，包括Mybatis全局配置文件和Mybatis映射文件，其中全局配置文件配置了数据源、事务等信息；映射文件配置了SQL执行相关的信息。
- 2、mybatis通过读取配置文件信息（全局配置文件和映射文件），构造出SqlSessionFactory，即会话工厂。
- 3、通过SqlSessionFactory，可以创建SqlSession即会话。Mybatis是通过SqlSession来操作数据库的。
- 4、SqlSession本身不能直接操作数据库，它是通过底层的Executor执行器接口来操作数据库的。Executor接口有两个实现类，一个是普通执行器，一个是缓存执行器（默认）。
- 5、Executor执行器要处理的SQL信息是封装到一个底层对象MappedStatement中。该对象包括：SQL语句、输入参数映射信息、输出结果集映射信息。其中输入参数和输出结果的映射类型包括java的简单类型、HashMap集合对象、POJO对象类型。

## 3.Mybatis开始

### 3.1 需求





**对用户信息的增删改查操作。**

- 1、 根据用户ID来查询用户信息；
- 2、 根据用户名称来模糊查询用户信息列表；
- 3、 添加用户
- 4、 删除用户（练习）
- 5、 修改用户（练习）

### 3.2 下载mybatis

mybaits的代码由github.com管理，下载地址：<https://github.com/mybatis/mybatis-3/releases>

## 名称

▶	lib
	LICENSE
	mybatis-3.4.5.jar
	mybatis-3.4.5.pdf
	NOTICE

Lib: mybatis的依赖包

Mybatis-3.4.5.jar: mybatis的核心包

Mybatis-3.4.5.pdf: mybatis的使用指南

### 3.3 maven创建mybatis项目

pom.xml文件

```
<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.lanou</groupId>
  <artifactId>Mybatis-01</artifactId>
  <version>1.0-SNAPSHOT</version>

  <name>Mybatis-01</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
```

```

        <scope>test</scope>
    </dependency>

    <!-- Mybatis需要的jar包:
    cglib,commons-logging,log4j,ognl,
    log4j-core,javassist,slf4j-api,slf4j-log4j12, -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.4.5</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/cglib/cglib -->
    <dependency>
        <groupId>cglib</groupId>
        <artifactId>cglib</artifactId>
        <version>3.2.5</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/commons-logging/commons
    -logging -->
    <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>1.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/log4j/log4j -->
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/ognl/ognl -->
    <dependency>
        <groupId>ognl</groupId>
        <artifactId>ognl</artifactId>
        <version>3.1.15</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.apache.logging.log4
    j/log4j-core -->
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.3</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.javassist/javassist
    -->
    <dependency>

```



```

        <groupId>org.javassist</groupId>
        <artifactId>javassist</artifactId>
        <version>3.22.0-CR2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-api -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.25</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-api -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.25</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-log4j12 -->
-->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.25</version>
        <scope>test</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-j
ava -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.18</version>
    </dependency>

</dependencies>
</project>

```

### 3.4 mybatis中的jar包说明

- MyBatis核心包（必须）



**mybatis-3.4.5.jar**

- MyBatis依赖包（必须）



asm-5.2.jar



cglib-3.2.5.jar



commons-logging-1.2.jar



javassist-3.22.0-CR2.jar



log4j-1.2.17.jar



log4j-api-2.3.jar



log4j-core-2.3.jar



ognl-3.1.15.jar



slf4j-api-1.7.25.jar



slf4j-log4j12-1.7.25.jar

- MySQL驱动包



mysql-connector-java-5.1.18-bin.jar

- Junit单元测试包（可选）



junit-4.12.jar

### 3.5 编程步骤

- 1、创建PO类，根据需求创建；
- 2、创建全局配置文件SqlMapConfig.xml；
- 3、编写映射文件；
- 4、加载映射文件，在SqlMapConfig.xml中进行加载；
- 5、编写测试程序，即编写Java代码，连接并操作数据库。思路：
  - a) 读取配置文件；
  - b) 通过SqlSessionFactoryBuilder创建SqlSessionFactory会话工厂。
  - c) 通过SqlSessionFactory创建SqlSession。
  - d) 调用SqlSession的操作数据库方法。
  - e) 关闭SqlSession。

### 3.5.1 创建PO类

#### User.java类

```
public class User {  
    private int id;  
    private String username;// 用户姓名  
    private String sex;// 性别  
    private Date birthday;// 生日  
    private String address;// 地址  
    //省略get/set  
}
```

### 3.5.2 创建SqlMapConfig.xml配置文件

#### SqlMapConfig.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>  
    <!--配置mybatis的环境信息-->  
    <environments default="development">  
        <environment id="development">  
            <!-- 配置JDBC事务控制，由mybatis进行管理 -->  
            <transactionManager type="JDBC"/>  
            <!-- 配置数据源，采用dbcp连接池 -->  
            <dataSource type="POOLED">  
                <property name="driver" value="com.mysql.jdbc.Driver"/>  
                <property name="url"  
                    value="jdbc:mysql://localhost:3306/mybatis?useUnicode=true&characterEncoding=utf8"/>  
                <property name="username" value="root"/>  
                <property name="password" value="1"/>  
            </dataSource>  
        </environment>  
    </environments>  
</configuration>
```

### 3.5.3 创建映射文件

#### User.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，它的作用就是对SQL进行分类化管理，可以理解为SQL隔离
    注意：使用mapper代理开发时，namespace有特殊且重要的作用
-->
<mapper namespace="test">

    <!-- 根据用户ID，查询用户信息 -->
    <!--
        [id]: statement的id，要求在命名空间内唯一
        [parameterType]: 入参的java类型
        [resultType]: 查询出的单条结果集对应的java类型
        [#{ }]: 表示一个占位符？
        [#{id}]: 表示该占位符待接收参数的名称为id。注意：如果参数为简单类型时，#{ }
        里面的参数名称可以是任意定义
    -->
    <select id="findUserById" parameterType="int"
            resultType="com.lanou.domain.User">
        SELECT * FROM USER WHERE id = #{id}
    </select>
</mapper>

```

### 3.5.4 加载映射文件

在SqlMapConfig.xml中，添加以下代码：

```

<!--加载mapper文件-->
<mappers>
    <mapper resource="resources/User.xml"/>
</mappers>

```

### 3.5.5 编写测试代码

```

package com.lanou.test;

import com.lanou.domain.User;
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;

```



```

/**
 * Created by 蓝鸥科技有限公司 www.lanou3g.com.
 */
public class MybatisTest {

    @Test
    public void findUserByIdTest() throws IOException {
        //1、读取配置文件
        InputStream inputStream = Resources.getResourceAsStream("SqlMap
Config.xml");
        //2、根据配置文件创建SqlSessionFactory
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuil
der().build(inputStream);
        //3、SqlSessionFactory创建SqlSession
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //4、SqlSession执行statement，并返回映射结果
        //第一个参数: statement的id, 建议: namespace.statementId (确保唯一)
        //第二个参数: 入参的值, 它的类型要和映射文件中对应的statement的入参类型一
致
        User user = sqlSession.selectOne("findUserById", 1);

        //打印输出结果集
        System.out.println(user);

        //5、关闭SqlSession
        sqlSession.close();
    }
}

```

### 3.5.6 根据用户名称模糊查询用户信息列表

在User.xml中，添加以下内容：

```

<!-- 根据用户名称模糊查询用户信息列表 -->
<!--
    [${}]：表示拼接SQL字符串
    [${value}]：表示要拼接的是简单类型参数。注意：
        1、如果参数为简单类型时，${}里面的参数名称必须为value
        2、${}会引起SQL注入，一般情况下不推荐使用。但是有些场景必须使用${}，比如
order by ${colname}
-->
<select id="findUsersByName" parameterType="String"
        resultType="com.lanou.domain.User">
    SELECT * FROM USER WHERE username LIKE '%${value}%'
</select>

```

测试代码：

```

@Test
    public void findUsersByName() throws IOException {
        //1、读取配置文件
        InputStream inputStream = Resources.getResourceAsStream("SqlMap
Config.xml");
        //2、根据配置文件创建SqlSessionFactory
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuil
der().build(inputStream);
        //3、SqlSessionFactory创建SqlSession
        SqlSession sqlSession = sqlSessionFactory.openSession();
        // 4、SqlSession执行statement，并返回映射结果
        // 第一个参数：statement的id，建议：namespace.statementId（确保唯一）
        // 第二个参数：入参的值，它的类型要和映射文件中对应的statement的入参类型一
致
        List<User> users = sqlSession.selectList("test.findUsersByName"
, "张三");

        //打印输出结果集
        System.out.println(users);

        //5、关闭SqlSession
        sqlSession.close();
    }

```

### 3.5.7 添加用户

在User.xml中，添加以下内容：

```

<!-- 添加用户 -->
<!-- 如果主键的值是通过MySQL自增机制生成的，那么我们此处不需要再显示的给ID赋值
[keyProperty]：指定存放生成主键的属性
[useGeneratedKeys]：true表示给主键设置自增长-->
<insert id="insertUser"
        parameterType="com.lanou.domain.User"
        keyProperty="id"
        useGeneratedKeys="true">
    INSERT INTO USER(username,sex,birthday,address)
    VALUES ({username},{sex},{birthday},{address})
</insert>

```

测试代码：

```

@Test
    public void insertUser() throws IOException {
        //1、读取配置文件
        InputStream inputStream = Resources.getResourceAsStream(

```

```

        "SqlMapConfig.xml");
//2、根据配置文件创建SqlSessionFactory
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder()
    .build(inputStream);
//3、SqlSessionFactory创建SqlSession
SqlSession sqlSession = sqlSessionFactory.openSession();
// 4、SqlSession执行statement, 并返回映射结果

//构建user参数, 没有赋值的属性采取默认值
User user = new User();
user.setUsername("李四");
user.setSex("男");
user.setBirthday(new Date());
user.setAddress("河南");

// 第一个参数: statement的id, 建议: namespace.statementId (确保唯一)
// 第二个参数: 入参的值, 它的类型要和映射文件中对应的statement的入参类型一致
sqlSession.insert("test.insertUser", user);
System.out.println(user);

//切记: 增删改操作时, 要执行commit操作
sqlSession.commit();

//5、关闭SqlSession
sqlSession.close();
}

```

### 3.5.8 删除用户

在User.xml中, 添加以下内容:

```

<!-- 根据ID删除用户 -->
<delete id="deleteUser" parameterType="int">
    DELETE FROM USER WHERE id= #{id}
</delete>

```

测试代码:

```

@Test
public void deleteUser() throws IOException {
    //1、读取配置文件
    InputStream inputStream = Resources.getResourceAsStream(
        "SqlMapConfig.xml");
    //2、根据配置文件创建SqlSessionFactory
    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder()

```



```

der()
        .build(inputStream);
//3、SqlSessionFactory创建SqlSession
SqlSession sqlSession = sqlSessionFactory.openSession();
// 4、SqlSession执行statement, 并返回映射结果
// 第一个参数: statement的id, 建议: namespace.statementId (确保唯一)
// 第二个参数: 入参的值, 它的类型要和映射文件中对应的statement的入参类型一致

sqlSession.delete("test.deleteUser", 10);

//切记: 增删改操作时, 要执行commit操作
sqlSession.commit();

//5、关闭SqlSession
sqlSession.close();
}

```

### 3.5.9 修改用户

在User.xml中, 添加以下内容:

```

<!-- 根据传入的用户信息修改用户 -->
<update id="updateUser" parameterType="com.lanou.domain.User">
    UPDATE USER SET username = #{username},sex=#{sex} WHERE id=#{id}
</update>

```

测试代码:

```

@Test
public void updateUser() throws IOException {
    //1、读取配置文件
    InputStream inputStream = Resources.getResourceAsStream(
        "SqlMapConfig.xml");
    //2、根据配置文件创建SqlSessionFactory
    SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder()
der()
        .build(inputStream);
//3、SqlSessionFactory创建SqlSession
SqlSession sqlSession = sqlSessionFactory.openSession();
// 4、SqlSession执行statement, 并返回映射结果
//构建user参数, 没有赋值的属性采取默认值
User user = new User();
user.setId(9);
user.setUsername("李四");
user.setAddress("大连");
}

```



致

```
// 第一个参数: statement的id, 建议: namespace.statementId (确保唯一)
// 第二个参数: 入参的值, 它的类型要和映射文件中对应的statement的入参类型一致

sqlSession.update("test.updateUser", user);

//切记: 增删改操作时, 要执行commit操作
sqlSession.commit();

//5、关闭SqlSession
sqlSession.close();
}
```

## 3.6 小节

- 3.6.1 parameterType和resultType
  - parameterType指定输入参数的java类型, 可以填写别名或Java类的全限定名。
  - resultType指定输出结果的java类型, 可以填写别名或Java类的全限定名。
- 3.6.2 #{}和\${}
  - 1、#{ }: 相当于预处理中的占位符?
    - a) #{ }里面的参数表示接收java输入参数的名称。
    - b) #{ }可以接受HashMap、简单类型、POJO类型的参数。
    - c) 当接受简单类型的参数时, #{ }里面可以是value, 也可以是其他。
    - d) #{ }可以防止SQL注入。
  - 2、\${ }: 相当于拼接SQL串, 对传入的值不做任何解释的原样输出
    - a) \${ }会引起SQL注入, 所以要谨慎使用。
    - b) \${ }可以接受HashMap、简单类型、POJO类型的参数。
    - c) 当接受简单类型的参数时, \${ }里面只能是value。
- 3.6.3 selectOne和selectList
  - selectOne: 只能查询0或1条记录, 大于1条记录的话, 会报错:
  - selectList: 可以查询0或N条记录

**如果遇到如下提示:**

Pages Build

- Warning: java: 源值1.5已过时, 将在未来所有发行版中删除
- Warning: java: 目标值1.5已过时, 将在未来所有发行版中删除
- Warning: java: 要隐藏有关已过时选项的警告, 请使用 -Xlint:-options。

- 解决方式

在POM文件中添加属性值:

```
1    <properties>
2        <maven.compiler.source>1.8</maven.compiler.source>
3        <maven.compiler.target>1.8</maven.compiler.target>
4    </properties>
```

## Maven Projects



### ▼ PreMybatis01 Maven Webapp

- ▶ Lifecycle
- ▶ Plugins
- ▶ Dependencies



Ant Build



Database



Maven Projects

1

