

# 刘沛霞

18600949004

## 永和大王门店管理系统-SSM版

# 2018

### 目录

[永和大王门店管理系统-SSM版](#)

[1 第一天：持久层框架MyBatis](#)

[1.1 前言](#)

[1.1.1 JDBC回顾](#)

[1.1.2 Mybatis概述](#)

[1.1.3 MyBatis原理图](#)

[1.2 开发环境](#)

[1.2.1 引入DTD文件提示支持](#)

[1.2.2 创建mybatisdb数据库，创建user表](#)

[1.3 Mybatis入门案例](#)

[1.3.1 Maven工程结构](#)

[1.3.2 pom.xml](#)

[1.3.3 sqlMapConfig.xml](#)

[1.3.4 User](#)

[1.3.5 UserMapper.xml](#)

[1.3.6 sqlMapConfig.xml中引入 UserMapper.xml](#)

[1.3.7 测试类](#)

[1.3.8 引入log4j打印SQL语句](#)

[1.4 总结](#)

[1.4.1 MyBatis中的重要对象](#)

[1.4.2 配置文件](#)

[1.5 常见操作](#)

[1.5.1 【根据id查询记录】](#)

[1.5.2 【查询总记录数】](#)

[1.5.3 【新增一条记录】](#)

[1.5.4 【修改指定记录】](#)

[1.5.5 【删除记录】](#)

[1.6 拓展：](#)

[1.6.1 #获取参数和\\$的区别](#)

[1.6.2 配置别名](#)

[1.6.3 SQL中有特殊字符](#)

[1.6.4 resultMap映射不规范字段](#)

[1.6.5 自动匹配规范驼峰规则](#)

[1.6.6 自增主键](#)

# 1 第一天：持久层框架MyBatis



## 1.1 前言

### 1.1.1 JDBC回顾

Java database connectivity, Java数据库连接。专门用来通过一段Java代码连接数据库操作数据库的一门技术。

开发步骤：

1, 注册驱动 2, 获取数据库连接

**3, 获取传输器(Statement、PreparedStatement)**

4, 执行SQL 5, 遍历结果集 6, 释放资源

其实, Mybatis也是用来操作数据库的技术, jdbc就可以了为啥又今天又要学框架?

### 1.1.2 Mybatis概述

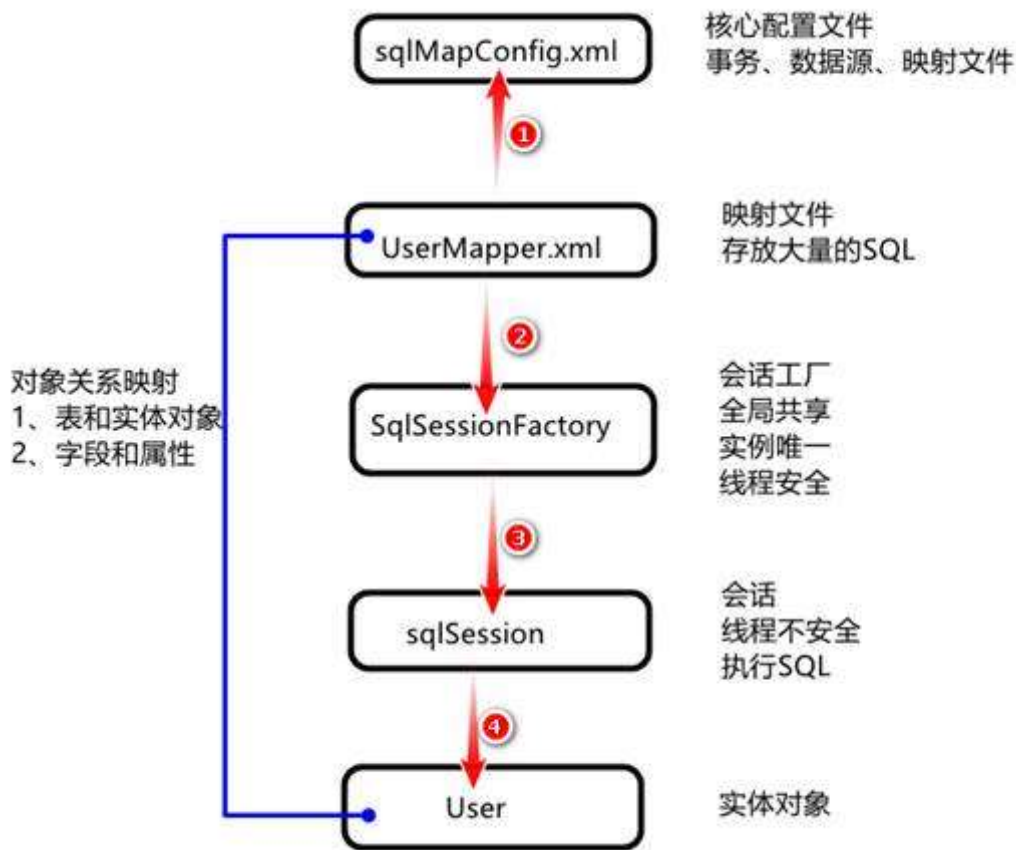
MyBatis的前身就是iBatis,iBatis本是apache的一个开源项目, 2010年5月这个项目由apahce sofeware foundation 迁移到了google code, 并且改名为MyBatis。

MyBatis 是支持普通 SQL 查询,存储过程和**高级映射**的优秀持久层框架。MyBatis 消除了几乎所有的 JDBC 代码和参数的手工设置以及结果集的检索。

1、简化JDBC的开发

2、能够更好的完成ORM(**对象关系映射**)

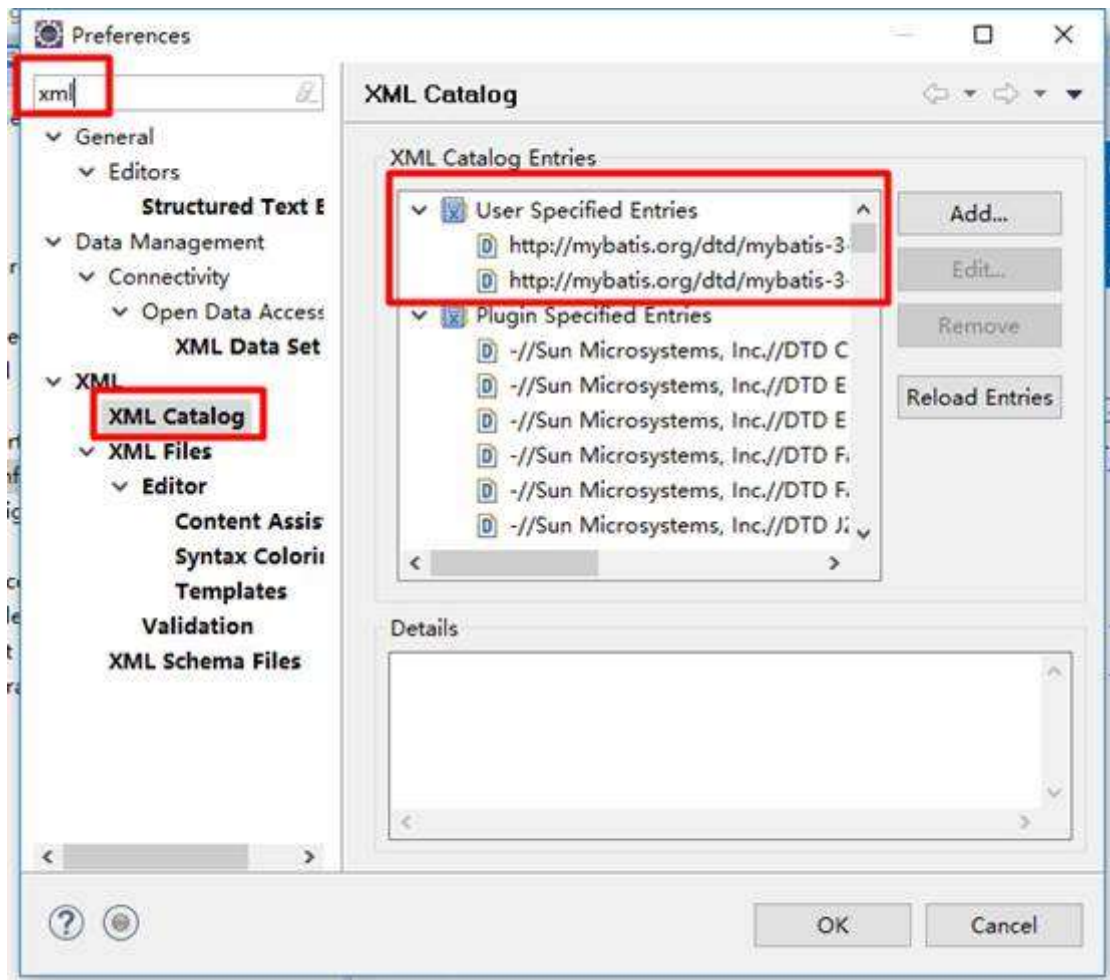
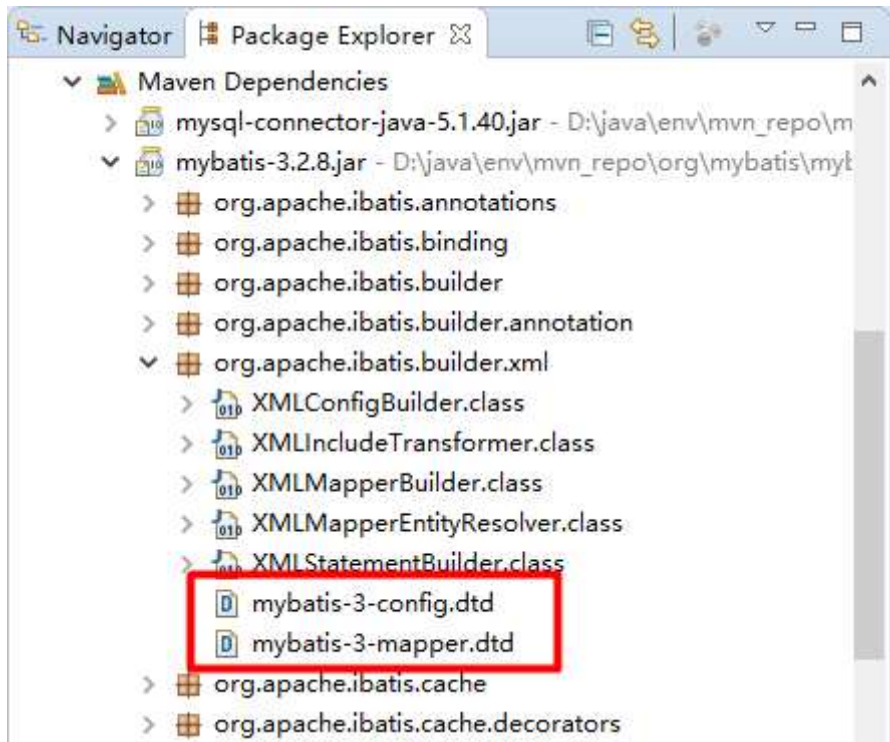
### 1.1.3 MyBatis原理图

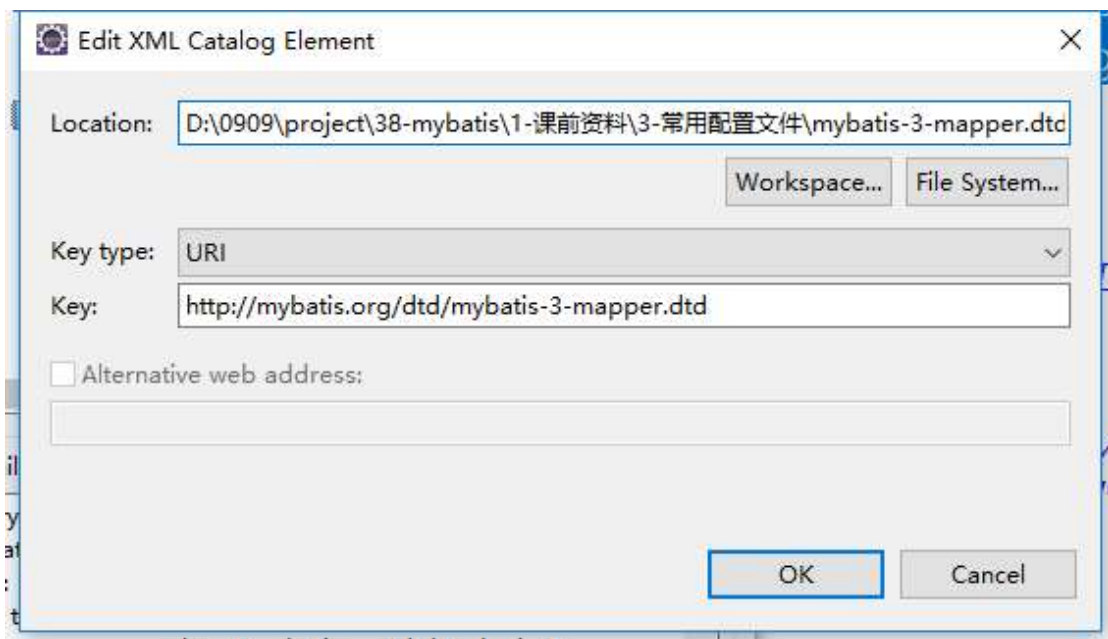
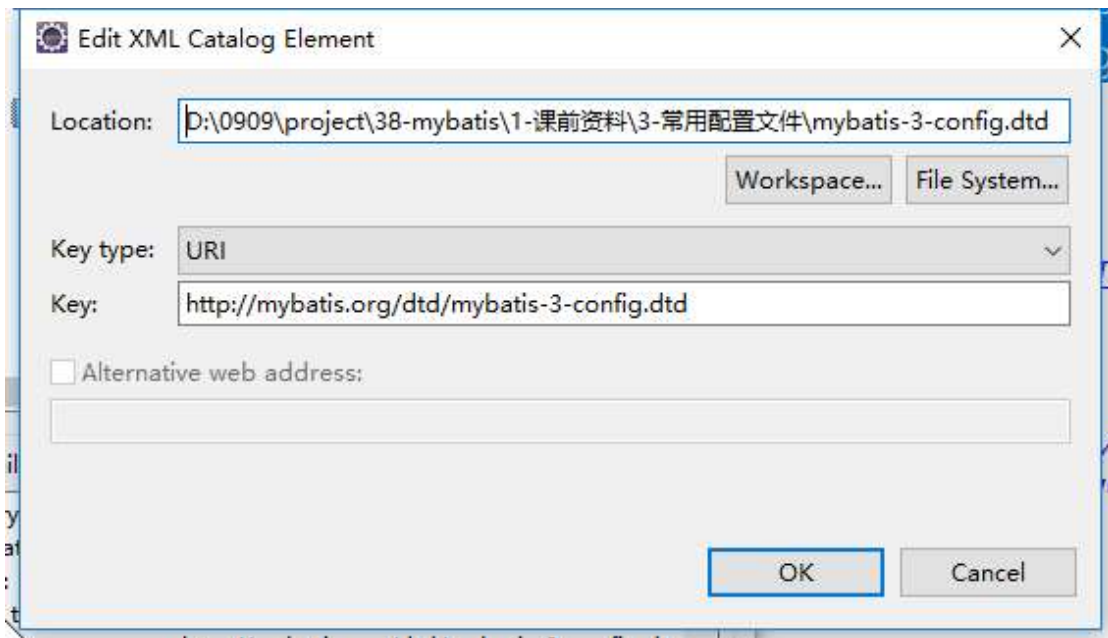


1. SqlMapConfig.xml: 此文件作为mybatis的全局配置文件，配置了mybatis的运行环境等信息。
2. UserMapper.xml: sql映射文件，文件中配置了操作数据库的sql语句。此文件需要在SqlMapConfig.xml中加载。
3. SqlSessionFactory: 通过mybatis环境等配置信息构造会话工厂对象
4. SqlSession: 由会话工厂创建会话，操作数据库需要通过SqlSession进行。
5. User: Executor把执行sql后的内容输出映射到java对象中，输出结果映射过程相当于jdbc编程中对结果的解析处理过程。

## 1.2 开发环境

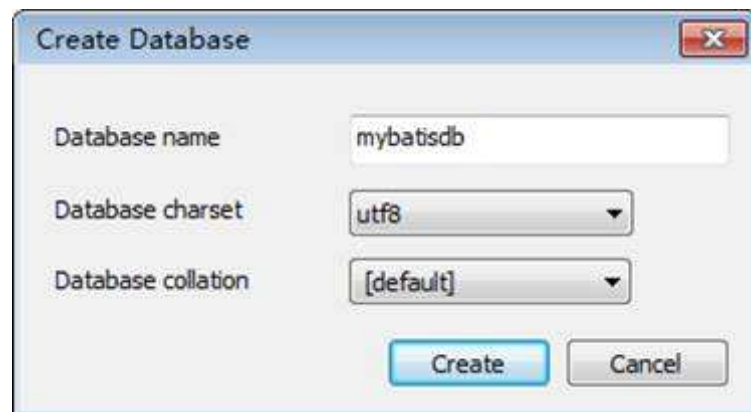
### 1.2.1 引入DTD文件提示支持





### 1.2.2 创建mybatisdb数据库, 创建user表

创建数据库



创建mybatisdb数据库, 设置utf8字符集。

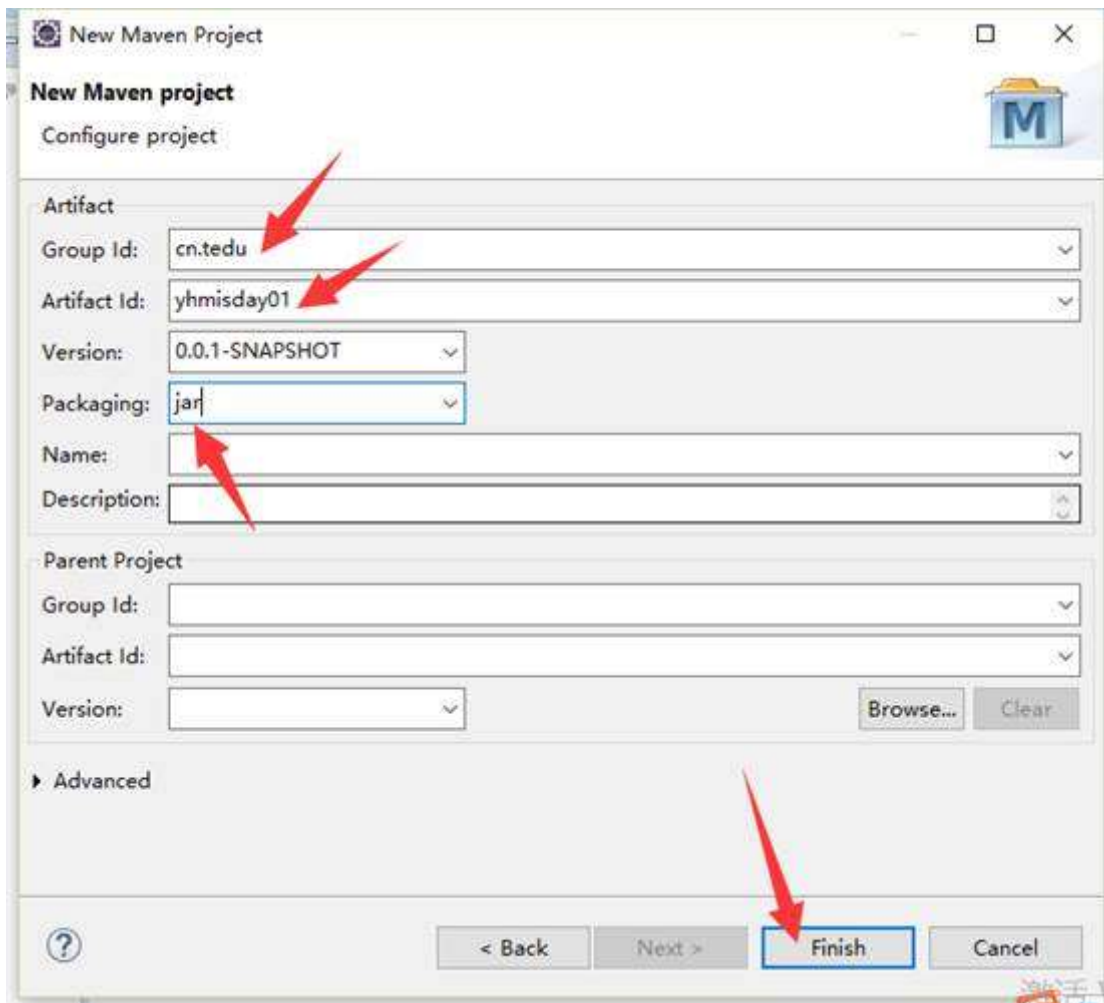
创建user表:

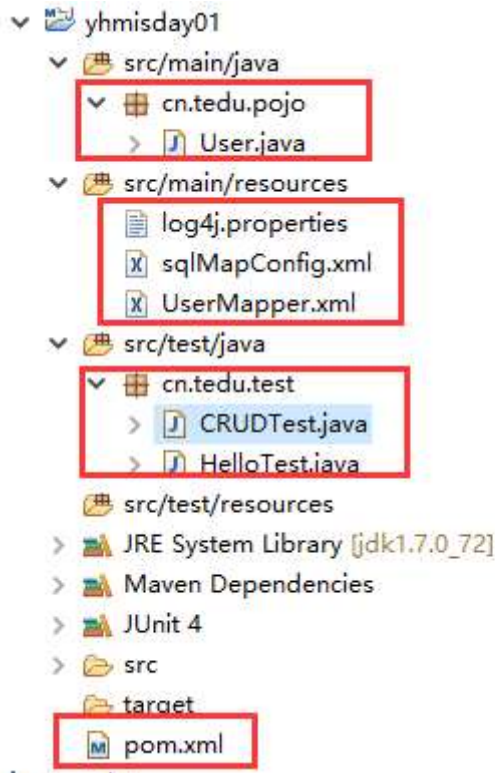
```
create database mybatisdb default character set utf8;  
use mybatisdb;  
create table user(
```

```
id int primary key auto_increment,  
name varchar(100),  
addr varchar(100),  
age int);  
insert into user values(null,' hanmeimei,' 北京' ,28);  
insert into user values(null,' xiongda' , ' 上海' ,20);  
insert into user values(null,' xiaonger' , ' 上海' ,19);
```

## 1.3 Mybatis入门案例

### 1.3.1 Maven工程结构





### 1.3.2 pom.xml

```

<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>cn.tedu</groupId>
    <artifactId>mybatis</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>mybatis</name>
    <url>http://maven.apache.org</url>

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

    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.40</version>
        </dependency>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.2.8</version>
        </dependency>
        <dependency>
            <groupId>cglib</groupId>

```

```

        <artifactId>cglib</artifactId>
        <version>2.2.2</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
</dependencies>
</project>

```

### 1.3.3 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">

<!-- Mybatis核心配置文件 -->
<configuration>
    <!-- 事务 、 数据源、 映射文件 -->
    <!-- 配置开发环境
        default: 默认的环境
    -->
    <environments default="dev">
        <!--id值就是默认的环境 -->
        <environment id="dev">
            <!-- 配置事务管理 -->
            <transactionManager type="JDBC"></transactionManager>

            <!-- 配置数据源 type: 池化的-->
            <dataSource type="pooled">
                <property name="driver"
                    value="com.mysql.jdbc.Driver"/>

                <property name="url"
                    value="jdbc:mysql://localhost:3306/mybatisdb?characterEncoding=utf-
8"/>

                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>

</configuration>

```

### 1.3.4 User

```

package cn.tedu.pojo;
//将表和字段映射给实体类和属性
public class User {

```



```
//id,name,addr,age
//映射id字段
private Integer id;

//映射name字段
private String name;

//映射addr字段
private String addr;

//映射age字段
private Integer age;

//getters and setters

public Integer getId() {
    return id;
}

public void setId(Integer id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getAddr() {
    return addr;
}

public void setAddr(String addr) {
    this.addr = addr;
}

public Integer getAge() {
    return age;
}

public void setAge(Integer age) {
    this.age = age;
}

//toString()
@Override
public String toString() {
    return "User [id=" + id + ", name=" + name + ", addr=" + addr + ", age=" + age + "];"
}
```

```
}
```

### 1.3.5 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">
<!-- 映射文件，存放SQL
      namespace是映射文件的唯一标志
-->
<mapper namespace="HelloMapper">

  <!-- select查询用户表的所有数据
        id该SQL的唯一标志
        resultType要把查询到的结果封装给哪个实体对象
  -->
  <select id="SelectAll"
        resultType="cn.tedu.pojo.User">
    select * from user
  </select>

</mapper>
```

### 1.3.6 sqlMapConfig.xml中引入 UserMapper.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核心配置文件 -->
<configuration>

  <!-- 事务 、 数据源、 映射文件 -->
  <!-- 配置开发环境
        default: 默认的环境，引用的是environment的id值
  -->
  <environments default="dev">
    <!--id值就是默认的环境 -->
    <environment id="dev">
      <!-- 配置事务管理 -->
      <transactionManager type="JDBC"></transactionManager>

      <!-- 配置数据源 type: 池化的-->
      <dataSource type="pooled">
        <property name="driver"
          value="com.mysql.jdbc.Driver"/>
      </dataSource>
    </environment>
  </environments>
</configuration>
```

```

        <property name="url"
            value="jdbc:mysql://localhost:3306/mybatisdb?characterEncoding=utf-
8"/>

        <property name="username" value="root"/>
        <property name="password" value="root"/>

        </dataSource>
    </environment>
</environments>

<!-- 引入映射文件 -->
<mappers>
    <!-- resource指定项目中mapper文件的位置 -->
    <mapper resource="UserMapper.xml"/>
</mappers>

</configuration>

```

### 1.3.7 测试类

#### 1.3.7.1 获取SqlSessionFactory对象，加载配置文件

#### 1.3.7.2 获取SqlSession对象，执行SQL

#### 1.3.7.3 解析结果

#### 1.3.7.4 释放资源

```

package cn.tedu.test;

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

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 cn.tedu.pojo.User;

//这个类用来测试MyBatis的入门案例
public class HelloTest {

    //单元测试
    @Test
    public void hello() throws IOException{
//        1.3.7.1 获取SqlSessionFactory对象,
//        加载核心配置文件
        InputStream in =
            Resources.getResourceAsStream
                ("sqlMapConfig.xml");

```

```

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

//    1.3.7.2    获取SqlSession对象, 执行SQL
        SqlSession session =
            ssf.openSession();

//    1.3.7.3    解析结果
//定位SQL的=namespace值+SQL的id值
        List<User> list =
            session.selectList(
                "HelloMapper.SelectAll");
//TODO遍历list
        for (User user : list) {
            System.out.println(user);
        }

//    1.3.7.4    释放资源
        session.close();

    }

}

```

### 1.3.8 引入log4j打印SQL语句

在classpath中建立一个名叫log4j.properties的文件, 内容如下:

```

# Global logging configuration
log4j.rootLogger=DEBUG, stdout
# Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

```

再次运行:

```

DEBUG [main] - ooo Using Connection [com.mysql.jdbc.JDBC4Connection@17671]
DEBUG [main] - ==> Preparing: select * from user
DEBUG [main] - ==> Parameters:

```

可以很清楚的看到执行的sql和执行sql使用的参数。

## 1.4 总结

### 1.4.1 MyBatis中的重要对象

MyBatis中有两个重要的对象, 分别是SqlSessionFactory和SqlSession。

#### 1.4.1.1 SqlSessionFactory

可以理解为会话工厂, 在整个项目中共享, 是线程安全的。通过openSession方法创建SqlSession对象, 该方法存在很多重载方式可以有参数可以无参数。

### 1.4.1.2 SqlSession

可以通过会话工厂产生，线程不安全。用来执行SQL，提供了丰富的方法来完成数据库的操作。

Mybatis不会自动提交事务，需要手动提交。

```
DEBUG [main] - Setting autocommit to false on JDBC Connection [com.mysql.jdbc.JDBC4Connection@56b981c0]
```

**提交方式有两种：openSession(true)或者session.commit()**

#### 1.4.2 配置文件

##### 1.4.2.1 核心配置文件sqlMapConfig.xml

- 文件名可以随意命名，习惯名称sqlMapConfig.xml
- 主要配置3个内容：事务管理器，数据源信息，映射文件
- 后续和spring框架整合后，该文件内容就都没了

##### 1.4.2.2 映射文件UserMapper.xml

- 文件名可以随意命名，习惯的方式是：POJO对象名+Mapper.xml
- 用来描述对应对象的信息，写大量表操作的SQL语句
- 通过#{value}来获取值，value写的是属性的名字。
- 命名空间namespace作为该映射文件的唯一标志
- 映射文件里的配置要和代码中匹配，比如返回值类型，参数类型等等
- 

## 1.5 常见操作

#### 1.5.1 【根据id查询记录】

##### 1.5.1.1 UserMapper.xml

添加SQL

```
<!-- 根据id查询记录
      id: 是这条SQL的唯一标志
      resultType: 查询的结果封装给哪个对象，全路径。。
-->
<select id="SelectOne"
      resultType="cn.tedu.pojo.User">
    select * from user where id=22
</select>
```

##### 1.5.1.2 创建测试类CRUDTest

调用sqlsession的CRUD方法

```
package cn.tedu.test;
```

```
import java.io.IOException;
import java.io.InputStream;
```

```
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 cn.tedu.pojo.User;

public class CRUDTest {

    //根据id查询记录
    @Test
    public void SelectOne() throws IOException{
        //加载核心配置文件
        InputStream in =
            Resources.getResourceAsStream(
                "sqlMapConfig.xml");

        //创建会话工厂
        SqlSessionFactory ssf =
            new SqlSessionFactoryBuilder()
                .build(in);

        //创建会话
        SqlSession session = ssf.openSession();

        //定位SQL并执行
        //namespace值.SQL的id值
        //SQL对应的结果集类型就是测试代码对应的类型
        User user =
            session.selectOne("HelloMapper.SelectOne");

        //处理结果集
        System.out.println(user);
        //释放资源
        session.close();
    }
}
```

### 1.5.2 【查询总记录数】

#### 1.5.2.1 UserMapper.xml

```
<!-- 查询总记录数
    id: 是这条SQL的唯一标志
-->
<select id="Count" resultType="int">
    select count(*) from user
</select>
```

### 1.5.2.2 测试类

```
// 查询总记录数
@Test
public void Count() throws IOException{
    //加载 核心配置文件
    InputStream in =
        Resources.getResourceAsStream(
            "sqlMapConfig.xml");

    //创建ssf会话工厂
    SqlSessionFactory ssf =
        new SqlSessionFactoryBuilder()
            .build(in);

    //创建会话
    SqlSession session = ssf.openSession();

    //执行SQL
    //定位SQL: namespace值.id值
    int rows =
        session.selectOne("HelloMapper.Count");

    //处理结果集
    System.out.println(rows);
    //释放资源
    session.close();
}
```

### 1.5.3 【新增一条记录】

#### 1.5.3.1 UserMapper.xml

```
<!-- 新增一条记录 -->
<insert id="Add">
    insert into user values
        (null,'rose','taiguo',18)
</insert>
```

或者

```
<!-- 新增一条记录
    #{id}获取值 id是实体对象的属性名
-->
<insert id="Add">
    insert into user values
        (#{id},#{name},#{addr},#{age})
</insert>
```

#### 1.5.3.2 改造测试类

```
// 新增一条记录
@Test
public void add() throws IOException{
    //加载核心配置文件
    InputStream in = Resources.getResourceAsStream(
```

```
        "sqlMapConfig.xml");

//创建ssf会话工厂
SqlSessionFactory ssf =
    new SqlSessionFactoryBuilder()
        .build(in);

//创建会话
SqlSession session = ssf.openSession();

//定位SQL并执行
//namespace.id
session.insert("HelloMapper.Add");

//手动提交事务
//openSession(boolean autocommit)
session.commit();

//释放资源
session.close();
}
```

或者

```
// 新增一条记录
@Test
public void add2() throws IOException{
    //加载核心配置文件
    InputStream in = Resources.getResourceAsStream(
        "sqlMapConfig.xml");

    //创建ssf会话工厂
    SqlSessionFactory ssf =
        new SqlSessionFactoryBuilder()
            .build(in);

    //创建会话
    SqlSession session = ssf.openSession();

    //定位SQL并执行
    //namespace.id
    //设置参数，SQL中动态获取参数值
    User user=new User();
    user.setName("jack");
    user.setAddr("shanghai");
    user.setAge(20);

    session.insert("HelloMapper.Add",user);

    //手动提交事务
    //openSession(boolean autocommit)
    session.commit();

    //释放资源
    session.close();
}
```



```
}
```

#### 1.5.4 【修改指定记录】

##### 1.5.4.1 UserMapper.xml

```
<!-- 修改指定记录
      通过#{ }来动态获取参数
      花括号里面，写的是属性名
-->
<update id="update">
    update user set
    age=#{age} where
    id=#{id}
</update>
```

##### 1.5.4.2 测试类

```
@Test
public void update() throws IOException{
    //加载核心配置文件
    InputStream in =
        Resources.getResourceAsStream(
            "sqlMapConfig.xml");

    //创建ssf
    SqlSessionFactory ssf =
        new SqlSessionFactoryBuilder()
            .build(in);

    //创建会话
    SqlSession session =
        ssf.openSession(true);

    //定位SQL并执行
    //第一个参数用来定位SQL
    //第二个参数用来给SQL传递参数
    User user = new User();
    user.setAge(52);
    user.setId(22);

    session.update(
        "HelloMapper.update",user);

    session.close();
}
```

#### 1.5.5 【根据id删除一条记录】

##### 1.5.5.1 UserMapper.xml

```
<!-- 根据id删除一条记录 -->
```

```
<delete id="delete">
    delete from user where id=#{id}
</delete>
```

### 1.5.5.2 测试类

```
// 根据id删除一条记录
@Test
public void delete() throws IOException{
    //加载核心配置文件
    InputStream in =
        Resources.getResourceAsStream(
            "sqlMapConfig.xml");

    //创建ssf
    SqlSessionFactory ssf =
        new SqlSessionFactoryBuilder()
            .build(in);

    //创建会话
    SqlSession session = ssf.openSession(true);

    //定位SQL并执行
    //第一个参数用来定位SQL
    //第二个参数用来给SQL传递参数
    User user = new User();
    user.setId(27);

    session.delete("HelloMapper.delete",user);

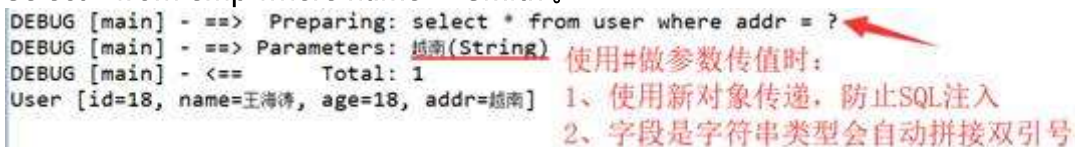
    session.close();
}
```

## 1.6 拓展:

前面讲了CRUD的全部测试，下面是对CRUD示例的优化：

### 1.6.1 #获取参数和\$的区别

(推荐!) #: 使用#{parameterName}引用参数的时候，Mybatis会把这个参数认为是一个字符串，例如传入参数是“Smith”，那么在SQL (Select \* from emp where name = #{employeeName})使用的时候就会转换为 Select \* from emp where name = 'Smith'。



```
DEBUG [main] - ==> Preparing: select * from user where addr = ?
DEBUG [main] - ==> Parameters: 越南(String)
DEBUG [main] - <== Total: 1
User [id=18, name=王海涛, age=18, addr=越南]
```

使用#做参数传值时:

- 1、使用新对象传递，防止SQL注入
- 2、字段是字符串类型会自动拼接双引号

\$: 不做字符串拼接，SQL (Select \* from emp where name = \${employeeName}) 使用的时候就会转换为 Select \* from emp where name = Smith。此时，如果字段是varchar类型直接抛出SQL异常。



从安全性上考虑, 能使用#尽量使用#来传参, 因为这样可以有效防止SQL注入的问题。

### 1.6.2 配置别名

在sqlMapConfig.xml配置, 在映射文件中直接写对象名称即可

```
<typeAliases>
  <typeAlias type="pojo.User" alias="User"/>
</typeAliases>
```

### 1.6.3 SQL中有特殊字符

当SQL中有特殊字符, mybatis不能正常解析时, 用CDATA括起来

```
<![CDATA[ age<=#{age} ]]>
```

### 1.6.4 ResultMap映射不规范字段

```
<?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="cn.tedu.mybatis.pojo.PersonMapper">
  <!-- 最强大对象ResultMap, type结果封装到哪个pojo对象, type就是谁 -->
  <resultMap type="Person" id="personRM">
    <!-- 主键,property对应pojo属性, column表的字段 -->
    <id property="id" column="id"/>
    <!-- 普通字段 -->
    <result property="userName" column="user_name"/>
  </resultMap>

  <!-- 查询所有 -->
  <select id="find" parameterType="Person" resultMap="personRM" >
    SELECT id,user_name FROM person
    WHERE user_name LIKE #{userName}
  </select>
</mapper>
```

### 1.6.5 自动匹配规范驼峰规则

数据库中我们习惯使用全大写, 多个单词用下划线隔开, 而po对象, 习惯使用java驼峰规则。那一个一个手工写resultMap字段, 浪费开发时间。Mybatis可以配置mapUnderscoreToCamelCase, 实现自动映射。这个值默认为true。

1) 全局配置:

```
<?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>
  <settings>
    <setting name="mapUnderscoreToCamelCase" value="true" />
  </settings>
</configuration>
2) resultMap配置autoMapping="true"
  <resultMap type="cn.tedu.jk.pojo.Contract" id="contractRM" autoMapping="true">
    <id property="id" column="CONTRACT_ID"/>
  </resultMap>

```

注意：主键需要单独写，其它字段就可以直接利用驼峰规则自动映射。

#### 1.6.6 自增主键

字段类型必须为int/long，数据还需支持mysql,sqlserver，oracle不支持

```

<?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="cn.mybatis.po.PersonMapper">
  <!-- mybatis自增主键
    useGeneratedKeys 启用mybatis自增
    keyProperty 主键
  -->
  <insert id="insert" parameterType="cn.mybatis.po.Person" keyProperty="id" keyColumn="id"
useGeneratedKeys="true">
    insert into person_c
    (name)
    values
    (#{name})
  </insert>
</mapper>

```