目标:

- [ ] 理解框架的作用
- [ ] 理解 MyBatis 框架的作用,能为咱们提供哪些方便
- [ ] 理解 MyBatis 框架的执行流程
- [ ] 掌握 MyBatis 框架的配置文件 mybatis-config.xml 和 XxxMapper.xml
- [ ] 了解如何使用 MyBatis 框架获取插入数据生成的主键
- [ ] 掌握如何使用 MyBatis 框架完成 CRUD 操作
- [ ] 了解细节处理中的类型别名,日志管理
- [ ] 掌握 MyBatisUtil 工具类抽取
- [ ] 掌握 db.properties 抽取和配置

1_框架概述

框架是一个半成品，已经对基础的代码进行了封装并提供相应的API，开发者在使用框架时直接调用封装好的api可以省去很多代码编写，从而提高工作效率和开发速度,框架是一种经过校验、具有一定功能的半成品软件.

经过校验：指框架本身经过测试，且框架自身所具有的功能已经实现

具有一定功能：指框架可以完成特定的功能，不同的框架功能不同

半成品软件：指框架自身是一个软件，但是该软件无法直接运行，需要配合其他的程序才可以完成指定的工作.

> 框架的工作模式：
> 开发工程师建立在框架的基础之上完成开发部分功能 加 框架自身完成部分功能组成一个完整的产品

例子: 小店卖早餐

1 自给自足

2 进货(框架) 卖

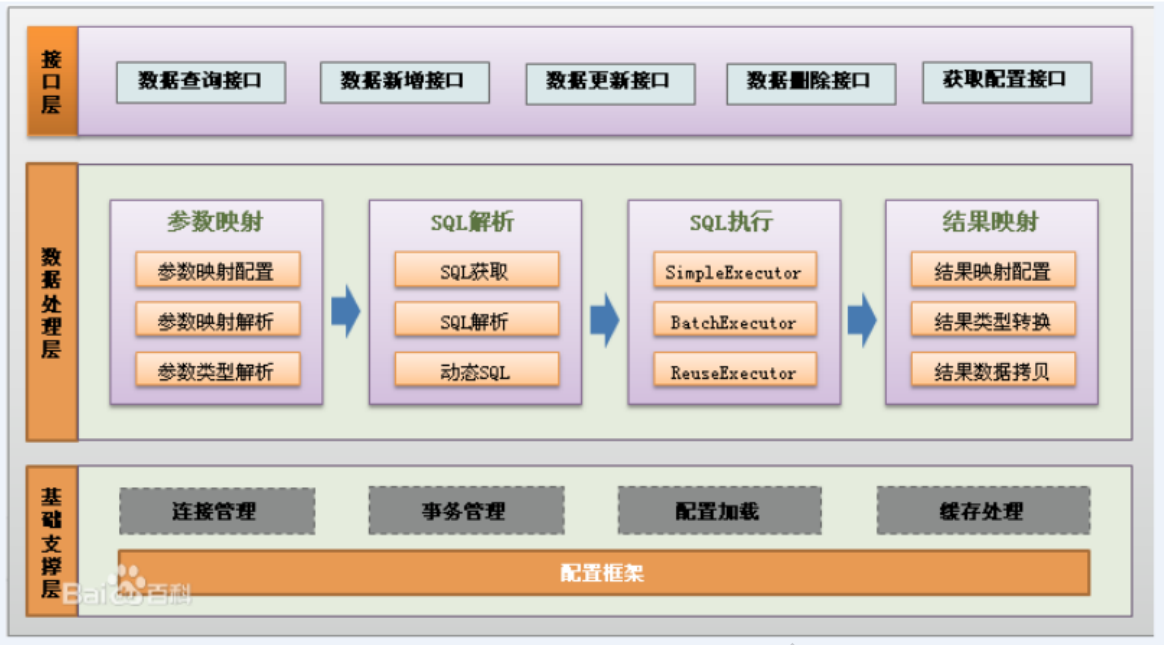
2_MyBatis 基础

2.1_MyBatis的概述

MyBatis 是一款优秀的**持久层框架**，它支持定制化 SQL、存储过程以及高级映射。**MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。**MyBatis 可以使用简单的 XML 或注解来配置和映射原生类型、接口和 Java 的 POJO（Plain Old Java Objects，普通老式 Java 对象）为数据库中的记录。

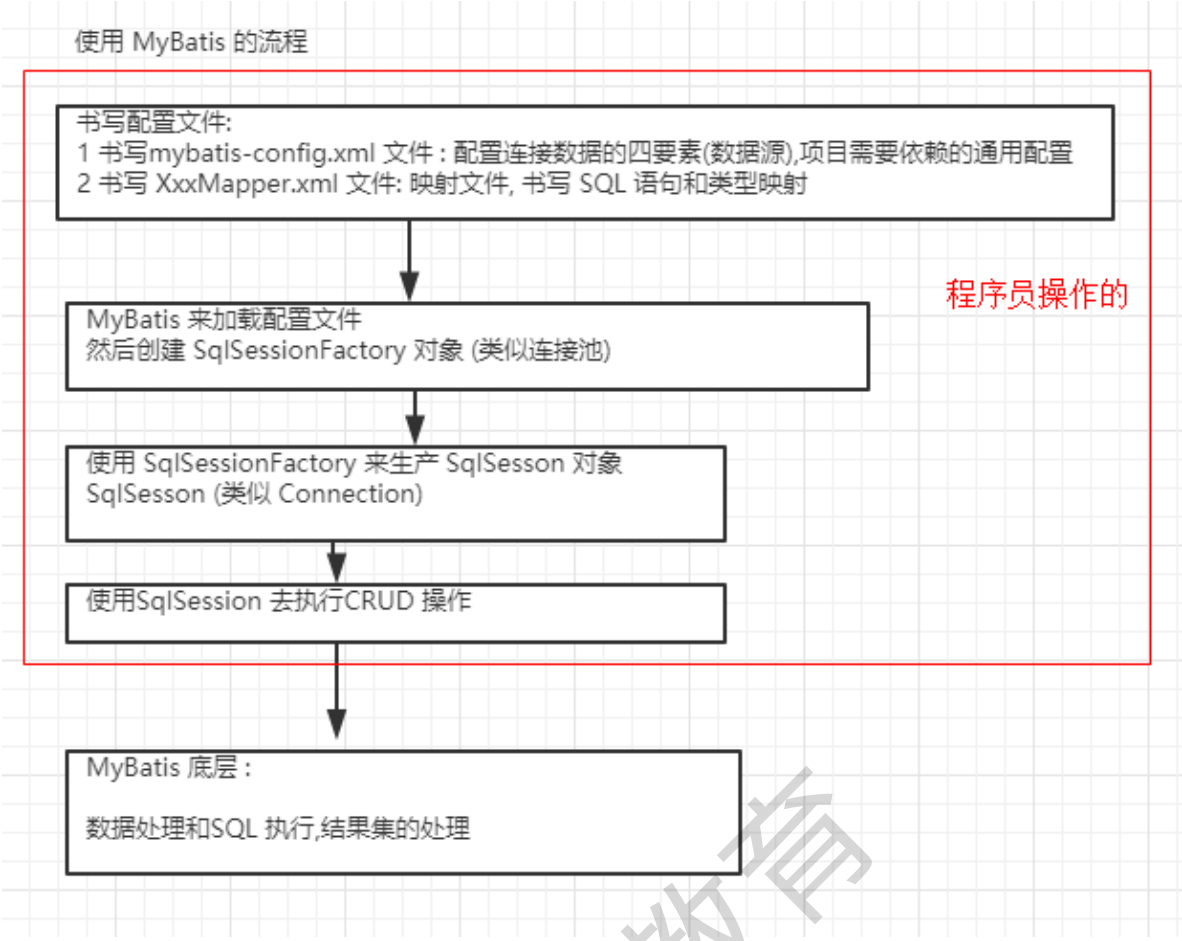
mybatis框架架构图:

mybatis框架架构图:



我们把Mybatis的功能架构分为三层：

1. API接口层：提供给外部使用的接口API，**开发人员通过这些本地API来操纵数据库**。接口层一接收到调用请求就会调用数据处理层来完成具体的数据处理。
2. 数据处理层：负责具体的SQL查找、SQL解析、SQL执行和执行结果映射处理等。它主要的目的是根据调用的请求完成一次数据库操作。
3. 基础支撑层：负责最基础的功能支撑，包括连接管理、事务管理、配置加载和缓存处理，这些都是共用的东西，将他们抽取出来作为最基础的组件。为上层的数据处理层提供最基础的支撑。



2.2_环境准备

> 需求: 向用户表中添加一条数据

1. 添加项目需要的 jar 包
    1. lombok-1.16.6.jar

       Lombok,自动生成 getter/setter/toString 等方法

       使用的前提是已经在 idea 中安装了 lombok 插件
    2. mysql-connector-java-5.1.26-bin.jar

       MySQL 数据库的 JDBC 驱动包,访问 MySQL 必须导入备 jar 包
    3. mybatis-3.4.5.jar

       MyBatis 框架的核心 jar 包
2. 创建一张用户表:user

```
CREATE TABLE `user` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `age` int(255) DEFAULT NULL,
  `salary` decimal(10,0) DEFAULT NULL
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
```

id	bigint	20	0	☑	🔑1
name	varchar	255	0	☐	
age	int	11	0	☐	
salary	decimal	10	0	☐	
hiredate	date	0	0	☐	

3. 根据表结构创建实体类

```
@Getter@Setter@ToString
@NoArgsConstructor@AllArgsConstructor
public class User {
    private Long id;
    private String name;
    private Integer age;
    private BigDecimal salary;
    private Date hiredate;
}
```

4. mybatis主配置文件: mybatis-config.xml

    1. 在项目的resources(source folder)下创建mybatis-config.xml配置文件
    2. 拷贝xml的约束

```
<?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">
```

    3. 添加环境配置(事务管理器 / 连接池 / 映射文件)

```
<?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">
<configuration>
  <environments default="dev">
    <environment id="dev">
      <!--
        MyBatis 内置的事务管理器
        JDBC:org.apache.ibatis.transaction.jdbc.JdbcTransaction的别名
      -->
      <transactionManager type="JDBC"/>
      <!--
        MyBatis 内置的连接池
        POOLED:org.apache.ibatis.datasource.pooled.PooledDataSource的别
名
      -->
      <dataSource type="POOLED">
        <!--
            driver:这是POOLED连接池对象的驱动类的属性名,
            Druid连接池对象的驱动类属性名是driverClassName
        -->
        <property name="driver" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql:///mybatis"/>
```

```
        <property name="username" value="root"/>
        <property name="password" value="admin"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="cn/wolfcode/mybatis/mapper/UserMapper.xml"/>
  </mappers>
</configuration>
```

4. mapper 映射文件: UserMapper.xml

   mybatis中,访问数据库的SQL语句是编写在mapper配置文件中的,程序员按照这个文件约定的格式进行配置即可

   在包 cn.wolfcode.mybatis.mapper 中创建配置文件: UserMapper.xml

   拷贝下面的约束信息到配置文件中

```
<?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">
```

在配置文件中添加 SQL 语句

```
<!--
    一个项目可以操作多张表
    每张表都需要一个mapper配置文件来编写SQL语句
    每条SQL语句都需要有一个唯一的标识
    这个唯一的标识由 namespace + sqlid 组成
    使用下面的namespace+sqlid就得到了保存用户信息的唯一标识:
    cn.wolfcode.mybatis.mapper.UserMapper.insert
    接下来,我们就可以使用上面的标识找到这条SQL语句了
-->
<mapper namespace="cn.wolfcode.mybatis.mapper.UserMapper">
  <insert id="insert">
    insert into user(name,age,salary,hiredate)
    values(#{name}, #{age}, #{salary}, #{hiredate})
  </insert>
</mapper>
```

注意: 一定记得在 mybatis-config.xml 配置文件中关联映射文件

```
<mappers>
    <!--
        这里是mapper文件的路径,所以使用/分割
    -->
    <mapper resource="cn/wolfcode/mybatis/mapper/UserMapper.xml"/>
</mappers>
```

2.3_DAO层开发

1. dao接口

   在 cn.wolfcode.mybatis.dao 包中创建 dao 接口: IUserDAO.java

```
package cn.wolfcode.mybatis.dao;

import cn.wolfcode.mybatis.domain.User;

public interface IUserDAO {
    /**
     * 向user表中插入一条数据
     * @param u 要插入的数据
     */
    void insert(User u);
}
```

2. **dao 接口实现类**

   在 cn.wolfcode.mybatis.dao.impl 包中创建 dao 的实现类: UserDAOImpl.java

   按以下 API 实现数据的保存操作:

   1. SqlSessionFactory : 对连接池(DataSource)的封装

      SqlSession openSession() : 获取SqlSession对象

   2. SqlSession,对连接(Connection)的封装

      int insert(String statementid, Object param);

      statementid: 有mapper配置文件中的namespace+sqlid组成

      void commit(): 提交事务

      void close() : 释放资源

   3. 调用 SqlSession 中的 insert 方法,执行指定的 SQL

```
public class UserDAOImpl implements IUserDAO {
    public void insert(User u) {
        try {
            InputStream in = Resources.getResourceAsStream("mybatis-
config.xml");
            SqlSessionFactory fac = new
SqlSessionFactoryBuilder().build(in);
            SqlSession session = fac.openSession();
            session.insert("cn.wolfcode.mybatis.mapper.UserMapper.insert",
u);
            session.commit();
            session.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4. 在测试类中测试 insert 方法

```
public class UserDAOTest {

    private IUserDAO dao = new UserDAOImpl();
    @Test
    public void testInsert() {
        User u = new User(null,"张三",20,new BigDecimal(20000), new Date());
        dao.insert(u);
    }
}
```

2.4_MyBatis 执行流程(理解)

1. 加载主配置文件(mybatis-config.xml)到内存中,将数据封装成对象
   Configuration/Environment/TransactionManager/DataSource等
2. 通过操作拿到访问数据库的基本信息,根据这些数据创建 SqlSessionFactory 对象
3. 从 SqlSessionFactory 对象中获取到 SqlSession 对象,然后执行SQL
4. INSERT INTO user(name,age,salary,hiredate)
   VALUES(#{name},#{age},#{salary},#{hiredate})

   被翻译成

   INSERT INTO user(name,age,salary,hiredate)
   VALUES(?, ?, ?, ?)
5. 使用 PreparedStatement 来执行指定的SQL

   从传递进来的 User 对象中依次获取到 name/age/salary/hiredate这些属性的值

   这里需要使用到内省机制来访问对象中的属性

注:

在加载完当前映射文件之后,会将SQL中的 #{} **OGNL表达式**翻译成对应的占位符?,

执行上面的SQL,需要将数据设置给当前的语句对象
ps.setObect(1, 从参数中获取到name属性的值);
ps.setObect(2, 从参数中获取到age属性的值);
ps.setObect(3, 从参数中获取到salary属性的值);
ps.setObect(4, 从参数中获取到hiredate属性的值);

操作步骤回顾:

1 创建项目,导入jar包

2 创建表和模型

3 创建 mybatis-config.xml 配置环境

4 创建 UserMapper.xml 文件 存在 mapper包中,配置sql语句

5 创建 dao 接口和实现类

6 启动 MyBatis 去执行插入操作

3_获取插入数据生成的主键(了解)

在开发中,如果需要获取到数据库中自动生成的主键,那么使用 MyBatis 应该如何实现呢?

往下看:

```
<insert id="insert" useGeneratedKeys="true" keyProperty="id" keyColumn="id">
    insert into user(name,age,salary,hiredate)
    values(#{name}, #{age}, #{salary}, #{hiredate})
</insert>
```

useGeneratedKeys: 是否要获取自动生成的主键

keyColumn: 表中的主键列

keyProperties: 主键列对应的属性

表示从获取哪个列的值封装到哪个属性中

通过上面的配置,在执行了保存操作后,mybatis 会自动将主键值封装到传递进来的 User 对象的 id 属性中

所以,此时的 User 对象的 id 属性就有值了(在保存之前是没有的)

```
parameterType : 参数类型,可以不用写,由MyBatis自己通过用户传入的对象去推导.
```

4_更新和删除操作

和保存操作的开发流程一致,在完成了保存操作之后,更新和删除操作一样可以实现

dao 中更新的实现

```
@Override
public void update(User u) {
    try {
        InputStream in = Resources.getResourceAsStream("mybatis-config.xml");
        SqlSessionFactory fac = new SqlSessionFactoryBuilder().build(in);
        SqlSession session = fac.openSession();
        session.update("cn.wolfcode.mybatis.mapper.UserMapper.update", u);
        session.commit();
        session.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

更新在 mapper 中的 SQL

```
<update id="insert">
    update user
    set
        name=#{name}, age = #{age}, salary = #{salary}, hiredate = #{hiredate}
    where
        id = #{id}
</insert>
```

dao 中删除的实现

```
@Override
public void delete(long id) {
    try {
        InputStream in = Resources.getResourceAsStream("mybatis-config.xml");
        SqlSessionFactory fac = new SqlSessionFactoryBuilder().build(in);
        SqlSession session = fac.openSession();
        session.delete("cn.wolfcode.mybatis.mapper.UserMapper.delete", id);
        session.commit();
        session.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

删除在 mapper 中的 SQL

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

注意:

1 当传入的只有一个普通类型(一个值)的时候#{} 中的名称可以随便写,但是建议**见名知义**

2 不管使用的是 insert 方法还是 delete 方法,底层调用的还是update方法,通过sql语句来区别dml操作

5_查询操作

根据 id 查询用户

dao 实现

```
@Override
public User selectOne(long id) {
    User u = null;
    try {
        InputStream in = Resources.getResourceAsStream("mybatis-config.xml");
        SqlSessionFactory fac = new SqlSessionFactoryBuilder().build(in);
        SqlSession session = fac.openSession();
        u = session.selectOne("cn.wolfcode.mybatis.mapper.UserMapper.selectOne",
id);
        session.close();
    } catch (Exception e) {
```

```
            e.printStackTrace();
        }
        return u;
    }
```

mapper 中的 SQL

```
<select id="selectOne" resultType="cn.wolfcode.mybatis.domain.User">
    select * from user where id = #{id}
</select>
```

和 DML 不一样的地方,查询操作需要指定把查询的结果集数据封装成什么类型的对象,resultType 属性就是这个作用

思考: resultType 可以不配置吗? 不可以,需要告诉 MyBatis 把一行数据封装成什么类型的对象

目前: 需要模型的属性和表的列需要一一匹配,如果不匹配,无法获取到数据.

查询所有用户

dao 实现

```
@Override
public List<User> selectList() {
    List<User> users = null;
    try {
        InputStream in = Resources.getResourceAsStream("mybatis-config.xml");
        SqlSessionFactory fac = new SqlSessionFactoryBuilder().build(in);
        SqlSession session = fac.openSession();
        users =
session.selectList("cn.wolfcode.mybatis.mapper.UserMapper.selectAll");
        session.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return users;
}
```

mapper 中的 SQL

```
<select id="selectAll" resultType="cn.wolfcode.mybatis.domain.User">
    select * from user
</select>
```

6_细节处理

跟着笔记能够配置即可

类型别名

在查询操作中, 我们需要使用result属性指定数据封装的类型, 这里的值类型的全限定名, 每次都编写的话比较麻烦,为了简化这里的配置,我们可以在主配置文件(mybatis-config.xml)中对指定的类型做别名的配置

```
<!-- 为指定包中的类来生成别名,默认是类的简单名称 -->
<typeAliases>
    <package name="cn.wolfcode.mybatis.domain"/>
</typeAliases>
```

如此,我们在查询的SQL中, 使用类的全限定名和使用别名是等价的

修改前:

```
<select id="selectAll" resultType="cn.wolfcode.mybatis.domain.User">
    select * from user
</select>
```

修改后:

```
<select id="selectAll" resultType="User">
    select * from user
</select>
```

日志管理

在持久层的开发过程中,我们程序员需要随时观察SQL的执行情况,如果sql有问题,我们能够及时发现所以,如果能够在控制台中将我们执行的SQL全部打印出来,那么就可以方便的观察SQL的相关问题

配置日志文件监控MyBatis的运行.

1. 将日志相关的jar包添加到项目中

2. 在resources(source folder)中创建配置文件, log4j.properties ,添加下面的内容(直接拷贝)

```
# Global logging configuration
log4j.rootLogger=ERROR, stdout
# 配置要打印日志的包
log4j.logger.cn.wolfcode.mybatis=TRACE
# 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
```

3. 将配置中的 包路径修改为项目中对应的路径,我们这里可以使用: cn.wolfcode.mybatis

到此,我们每执行一条 SQL, 都会在控制台中打印出来,这非常有助于我们对 SQL 的分析,特别是 SQL 不正确的时候

7_抽取 MyBatisUtil 工具类

和 JDBC 中的连接池使用一样,在整个项目中,我们只需要一个SqlSessionFactory对象,而不需要每次都创建一个新的,

所以,我们将SqlSessionFactory的创建和SqlSession对象的获取都抽取到工具类中: MyBatisUtil

```
public class MyBatisUtil {

    private MyBatisUtil(){}

    private static SqlSessionFactory fac;
    static{
        try {
            InputStream inputStream = Resources.getResourceAsStream("mybatis-config.xml");
            fac = new SqlSessionFactoryBuilder().build(inputStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    //获取SqlSession对象
    public static SqlSession openSession(){
        return fac.openSession();
    }
}
```

注意:

```
Caused by: org.xml.sax.SAXParseException; lineNumber: 1; columnNumber: 1; 前言中不允许有内容。
```

加载配置文件没有填写:

```
// 1 获取 SqlSessionFactory
factory = new SqlSessionFactoryBuilder()
        .build(Resources.getResourceAsStream(""));
```

8_抽取db.properties

在主配置文件中配置的内容较多, 其中包括我们修改频率较高的数据库连接信息(driver/url/username/password)等,我们可以在修改这些信息的过程中误改或者误删到其他的配置,为了解决这个问题,我们仍然是将这些信息配置到db.propertie配置文件中,然后再合并到主配置文件即可,如下:

db.properties

```
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/javaweb
username=root
password=admin
```

mybatis-cofnig.xml

```
<configuration>
    <!-- 关联db.properties:为了把连接数据库的信息单独放到一个配置文件 -->
    <properties resource="db.properties" />

    <!-- 为指定包中的类来生成别名 -->
    <typeAliases>
        <package name="cn.wolfcode.mybatis.domain"/>
    </typeAliases>


    <!-- 连接数据库的环境
        default:指定使用哪一个环境的配置
    -->
    <environments default="dev">
        <environment id="dev">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <!--
                    使用${key}取出db.properties中配置的信息
                    key和db.properties文件中的可以一直即可
                -->
                <property name="driver" value="${driverClassName}"/>
                <property name="url" value="${url}"/>
                <property name="username" value="${username}"/>
                <property name="password" value="${password}"/>
            </dataSource>
        </environment>
    </environments>
    <!-- 关联映射文件 -->
    <mappers>
        <mapper resource="cn\wolfcode\pmis\mapper\ProductMapper.xml"/>
    </mappers>
</configuration>
```


小结

1. 为什么要使用框架
2. 为什么要选择 MyBatis 框架
3. 配置 mybatis-config.xml
    1. **加载 db.properties**
    2. 配置类的别名
    3. **连接池: POOLED**
    4. **事务管理器: JDBC**
    5. **关联mapper映射文件**

4. 配置 UserMapper.xml

   编写 SQL 语句

```
namespace+sqlid = SQL的唯一标识
namespace: 包名+XxxMapper,如果写了不是包名+XxxMapper,日志是无法显示
<mapper namespace="">
    <select id="" resultType=""></select>
    <insert id="" useGeneratedKeys="true" keyProperty=""></insert>
    <delete id=""></delete>
    <update id=""></update>
</mapper>
```

   #{}: OGNL 表达式

   如果参数是 JavaBean 对象,这里使用的是属性名(调的是 getter 方法)

   如果参数是简单类型的,这里可以随便写,但是建议见名知意

5. 在 dao 中使用 MyBatis 提供的 api 完成数据的 crud
   1. Resources: 从classpath 路径下加载指定名称的配置文件 mybatis-config.xml
   2. SqlSessionFactory
   3. SqlSession
6. 细节的处理(根据文档来配置即可)

要求:

第一遍: 按照课堂的知识点的流程完成对 User 表的 CRUD

第二遍: 完成 student 表数据的 CRUD

拷贝上一个项目,将 Java 代码删除(不删包),重新写

domain/dao接口/dao实现类/dao的测试类/mybatis-config.xml/StudentMapper.xml

复习 sql 语句,写一个文件(资料中有sql 的练习题)

写总结: 梳理知识点

画 MyBatis 的执行流程图,书写使用 MyBatis 完成插入操作的书写步骤(文字)