**Mybatis**

Mybatis框架开发步骤（基于xml配置）：

1. **创建maven工程**

创建 mybatis01 的工程，工程信息如下：

Groupid:com.itheima

ArtifactId:mybatis01

Packing:jar

1. **添加 Mybatis3.4.5 的坐标**

在pom.xml中添加坐标：

<dependencies>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.4.5</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.10</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.6</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.12</version>

</dependency>

</dependencies>

1. **编写实体类User**

**public class** User **implements** Serializable {

**private** Integer id;

**private** String username;

**private** Date birthday;

**private** String sex;

**private** String address;

**public** Integer getId() {

**return** id;

}

**public void** setId(Integer id) {

**this**.id = id;

}

**public** String getUsername() {

**return** username;

}

**public void** setUsername(String username) {

**this**.username = username;

}

**public** Date getBirthday() {

**return** birthday;

}

**public void** setBirthday(Date birthday) {

**this**.birthday = birthday;

}

**public** String getSex() {

**return** sex;

}

**public void** setSex(String sex) {

**this**.sex = sex;

}

**public** String getAddress() {

**return** address;

}

**public void** setAddress(String address) {

**this**.address = address;

}

@Override

**public** String toString() {

**return** "User [id=" + id + ", username=" + username + ", birthday=" + birthday

+ ", sex=" + sex + ", address="

+ address + "]";

}

}

1. **编写持久层接口 UserDao**

**public interface** IUserDao {

/\*\*\* 查询所有用户\* **@return**\*/

List<User> findAll();

}

1. **编写持久层接口的映射文件 UserDao.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=*"com.itheima.dao.IUserDao"*>

<!-- 配置查询所有操作 -->

<select id=*"findAll"* resultType=*"com.itheima.domain.User"*>

select \* from user

</select>

</mapper>

1. **编写 SqlMapConfig.xml 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">

<configuration>

<!-- 配置 mybatis 的环境 -->

<environments default=*"mysql"*>

<!-- 配置 mysql 的环境 -->

<environment id=*"mysql"*>

<!-- 配置事务的类型 -->

<transactionManager type=*"JDBC"*></transactionManager>

<!-- 配置连接数据库的信息：用的是数据源(连接池) -->

<dataSource type=*"POOLED"*>

<property name=*"driver"* value=*"com.mysql.jdbc.Driver"*/>

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/ee50"*/>

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

<property name=*"password"* value=*"1234"*/>

</dataSource>

</environment>

</environments>

<!-- 告知 mybatis 映射配置的位置 -->

<mappers>

<mapper resource=*"com/itheima/dao/IUserDao.xml"*/>

</mappers>

</configuration>

1. **编写测试类**

**public class** MybatisTest {

**public static void** main(String[] args)**throws** Exception {

//1.读取配置文件

InputStream in = Resources.*getResourceAsStream*("SqlMapConfig.xml");

//2.创建 SqlSessionFactory 的构建者对象

SqlSessionFactoryBuilder builder = **new** SqlSessionFactoryBuilder();

//3.使用构建者创建工厂对象 SqlSessionFactory

SqlSessionFactory factory = builder.build(in);

//4.使用 SqlSessionFactory 生产 SqlSession 对象

SqlSession session = factory.openSession();

//5.使用 SqlSession 创建 dao 接口的代理对象

IUserDao userDao = session.getMapper(IUserDao.**class**);

//6.使用代理对象执行查询所有方法

List<User> users = userDao.findAll();

**for**(User user : users) {

System.***out***.println(user);

}

//7.释放资源

session.close();

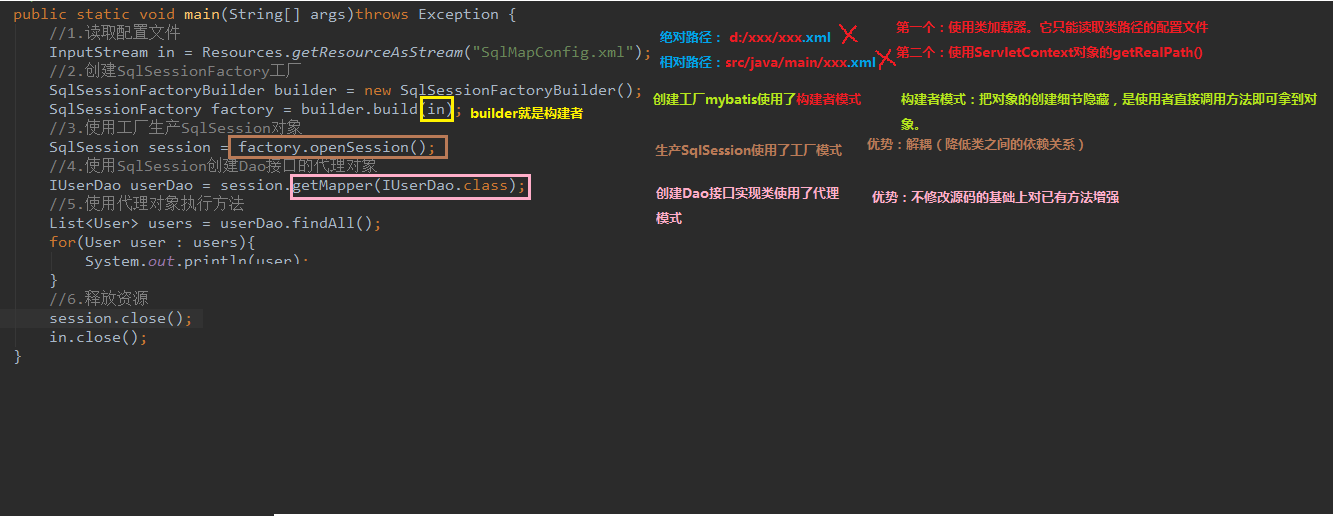
in.close();

}

}

**Mybatis框架分析**

知识点：工厂模式（Factory 工厂模式）、构造者模式（Builder 模式）、代理模式，反射，自定义注解，注解的反射，xml 解析，数据库元数据，元数据的反射等。





分析：

//1.读取配置文件

**public class** Resources {

/\*\*

\* 用于加载 xml 文件，并且得到一个流对象

\* **@param** xmlPath

\* **@return**

\* 在实际开发中读取配置文件:

\* 第一：使用类加载器。但是有要求：a 文件不能过大。 b 文件必须在类路径下(classpath)

\* 第二：使用 ServletContext 的 getRealPath()

\*/

**public static** InputStream getResourceAsStream(String xmlPath) {

**return** Resources.**class**.getClassLoader().getResourceAsStream(xmlPath);

}

}

//2.创建 SqlSessionFactory 的构建者对象

//3.使用构建者创建工厂对象 SqlSessionFactory

**public class** SqlSessionFactoryBuilder {

/\*\*

\* 根据传入的流，实现对 SqlSessionFactory 的创建

\* **@param** in 它就是 SqlMapConfig.xml 的配置以及里面包含的 UserDao.xml 的配置

\* **@return**

\*/

**public** SqlSessionFactory build(InputStream in) {

DefaultSqlSessionFactory factory = **new** DefaultSqlSessionFactory();

//给 factory 中 config 赋值

factory.setConfig(in);

**return** factory;

}

}

//4.使用 SqlSessionFactory 生产 SqlSession 对象

**public interface** SqlSessionFactory {

/\*\*

\* 创建一个新的 SqlSession 对象

\* **@return**

\*/

SqlSession openSession();

}

**public class** DefaultSqlSessionFactory **implements** SqlSessionFactory {

**private** InputStream config = **null**;

**public void** setConfig(InputStream config) {

**this**.config = config;

}

@Override

**public** SqlSession openSession() {

DefaultSqlSession session = **new** DefaultSqlSession();

//调用工具类解析 xml 文件

XMLConfigBuilder.*loadConfiguration*(session, config);

**return** session;

}

}

//5.使用 SqlSession 创建 dao 接口的代理对象

**public class** MapperProxyFactory **implements** InvocationHandler {

**private** Map<String,Mapper> mappers;

**private** Connection conn;

**public** MapperProxyFactory(Map<String, Mapper> mappers,Connection conn) { **this**.mappers = mappers;

**this**.conn = conn;

}

/\*\*

\* 对当前正在执行的方法进行增强

\* 取出当前执行的方法名称

\* 取出当前执行的方法所在类

\* 拼接成 key

\* 去 Map 中获取 Value（Mapper)

\* 使用工具类 Executor 的 selectList 方法

\*/

@Override

**public** Object invoke(Object proxy, Method method, Object[] args) **throws** Throwable

{

//1.取出方法名

String methodName = method.getName();

//2.取出方法所在类名

String className = method.getDeclaringClass().getName();

//3.拼接成 Key

String key = className+"."+methodName;

//4.使用 key 取出 mapper

Mapper mapper = mappers.get(key);

**if**(mapper == **null**) {

**throw new** IllegalArgumentException("传入的参数有误，无法获取执行的必要条件

");

}

//5.创建 Executor 对象

Executor executor = **new** Executor();