Mybatis 任务一：基本应用

课程任务主要内容：

```
* 框架介绍
* Mybatis：ORM
* 快速入门
* 映射文件简单概述
* 实现CRUD
* 核心配置文件介绍
* api介绍
* mybatis的dao层开发使用（接口代理方式）
```

SSM = springmvc + spring + mybatis

一 框架简介

1.1 三层架构

软件开发常用的架构是三层架构，之所以流行是因为有着清晰的任务划分。一般包括以下三层：

- 持久层：主要完成与数据库相关的操作，即对数据库的增删改查。

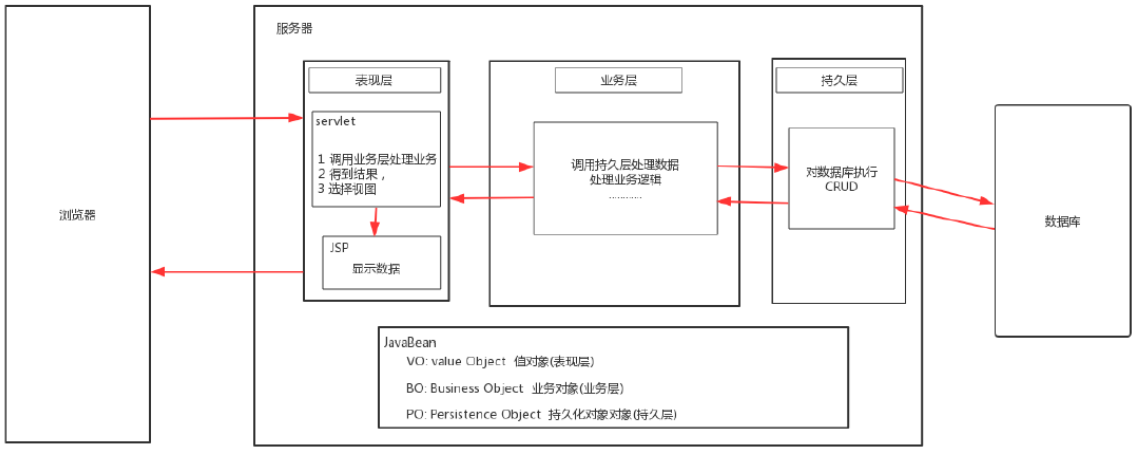
  因为数据库访问的对象一般称为Data Access Object（简称DAO），所以有人把持久层叫做DAO层。
- 业务层：主要根据功能需求完成业务逻辑的定义和实现。

  因为它主要是为上层提供服务的，所以有人把业务层叫做Service层或Business层。
- 表现层：主要完成与最终软件使用用户的交互，需要有交互界面（UI）。

  因此，有人把表现层称之为web层或View层。

三层架构之间调用关系为:表现层调用业务层，业务层调用持久层。

各层之间必然要进行数据交互，我们一般使用java实体对象来传递数据。



1.2 框架

1.2.1 什么是框架?

框架就是一套规范，既然是规范，你使用这个框架就要遵守这个框架所规定的约束。

框架可以理解为半成品软件，框架做好以后，接下来在它基础上进行开发。

1.2.2 为什么使用框架?

框架为我们封装好了一些冗余，且重用率低的代码。并且使用反射与动态代理机制，将代码实现了通用性，让 开发人员把精力专注在核心的业务代码实现上。

比如在使用servlet进行开发时，需要在servlet获取表单的参数，每次都要获取很麻烦，而框架底层就使用反射机制和拦截器机制帮助我们获取表单的值，使用jdbc每次做专一些简单的crud的时候都必须写sql，但使用框架就不需要这么麻烦了，直接调用方法就可以。当然，既然是使用框架，那么还是要遵循其一些规范进行配置

1.2.3 常见的框架

Java世界中的框架非常的多，每一个框架都是为了解决某一部分或某些问题而存在的。下面列出在目前企业中
流行的几种框架（一定要注意他们是用来解决哪一层问题的）：

- 持久层框架：专注于解决数据持久化的框架。常用的有mybatis、hibernate、spring jdbc等等。
- 表现层框架：专注于解决与用户交互的框架。常见的有struts2、spring mvc等等。
- 全栈框架: 能在各层都给出解决方案的框架。比较著名的就是spring。

这么多框架，我们怎么选择呢?

我们以企业中最常用的组合为准来学习Spring + Spring MVC + mybatis（SSM）

二 Mybatis简介

1.1 原始jdbc操作（查询数据）

```java
@Test
public void testJDBC() throws ClassNotFoundException, SQLException {

    // 注册驱动
    Class.forName("com.mysql.jdbc.Driver");
    // 获取连接
    Connection connection = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/lagou_test", user: "root", password: "root");
    // 获得statement
    String sql = "select id,username from user";
    PreparedStatement statement = connection.prepareStatement(sql);
    // 执行查询
    ResultSet resultSet = statement.executeQuery();
    // 遍历结果集
    while (resultSet.next()){
        // 封装实体
        User user = new User();
        user.setId(resultSet.getInt( columnLabel: "id"));
        user.setUsername(resultSet.getString( columnLabel: "username"));
        // 实体封装完毕
        System.out.println(user);
    }
    // 释放资源
    resultSet.close();
    statement.close();
    connection.close();
}
```

1.2 原始jdbc操作的分析

原始jdbc开发存在的问题如下：

① 数据库连接创建、释放频繁造成系统资源浪费从而影响系统性能

② sql 语句在代码中硬编码，造成代码不易维护，实际应用 sql 变化的可能较大，sql 变动需要改变java代码。

③ 查询操作时，需要手动将结果集中的数据手动封装到实体中。

应对上述问题给出的解决方案：

① 使用数据库连接池初始化连接资源

② 将sql语句抽取到xml配置文件中

③ 使用反射、内省等底层技术，自动将实体与表进行属性与字段的自动映射

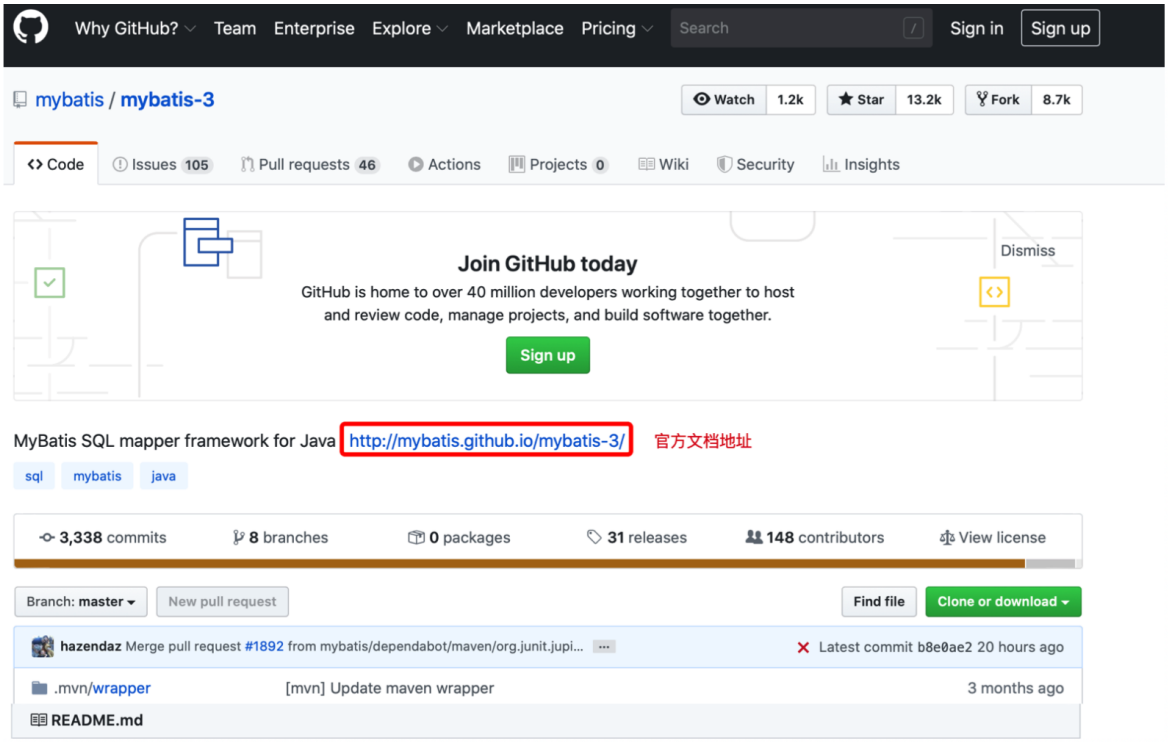
1.3 Mybatis简介

MyBatis是一个优秀的基于**ORM**的**半自动轻量级**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 历史

MyBatis 本是apache的一个开源项目iBatis, 2010年6月这个项目由apache software foundation 迁移到了google code，随着开发团队转投到Google Code旗下，iBatis正式改名为MyBatis ，代码于2013年11月迁移到Github

Github地址：https://github.com/mybatis/mybatis-3/



1.4 ORM思想

ORM（Object Relational Mapping）对象关系映射

O（对象模型）：

实体对象，即我们在程序中根据数据库表结构建立的一个个实体javaBean

R（关系型数据库的数据结构）：

关系数据库领域的Relational（建立的数据库表）

M（映射）：

从R（数据库）到O（对象模型）的映射，可通过XML文件映射

实现：

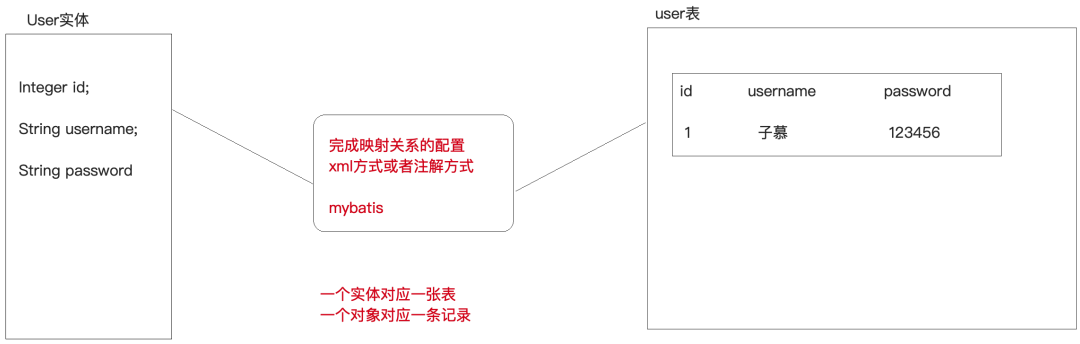
（1）让实体类和数据库表进行一一对应关系

先让实体类和数据库表对应

再让实体类属性和表里面字段对应

（2）不需要直接操作数据库表，直接操作表对应的实体类对象

ORM（Object Relational Mapping）对象关系映射
O：User实体
R：user表
M：建立User实体和user表的映射关系



ORM作为是一种思想

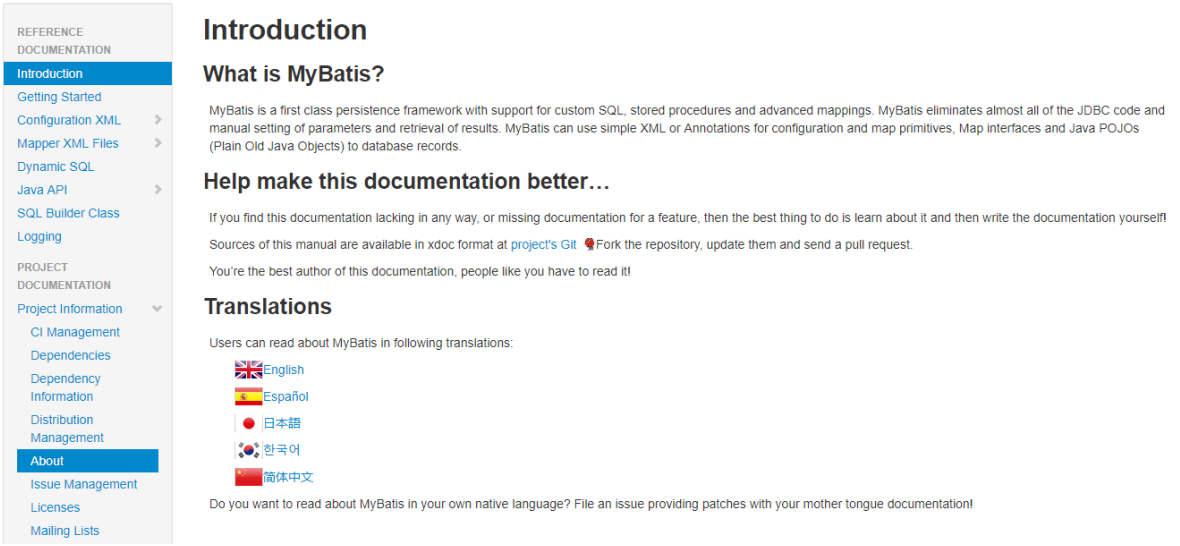
帮助我们跟踪实体的变化,并将实体的变化翻译成sql脚本,执行到数据库中去,也就是将实体的变化映射到了表的变化。

mybatis采用**ORM**思想解决了实体和数据库映射的问题，对jdbc 进行了封装，屏蔽了jdbc api 底层访问细节，使我们不用与jdbc api 打交道，就可以完成对数据库的持久化操作

三 Mybatis快速入门

3.1 MyBatis开发步骤

MyBatis官网地址：http://www.mybatis.org/mybatis-3/



案例需求：通过mybatis查询数据库user表的所有记录，封装到User对象中，打印到控制台上

步骤分析：

```
1. 创建数据库及user表

2. 创建maven工程，导入依赖（MySQL驱动、mybatis、junit）

3. 编写User实体类

4. 编写UserMapper.xml映射配置文件（ORM思想）

5. 编写SqlMapConfig.xml核心配置文件
        数据库环境配置
        映射关系配置的引入(引入映射配置文件的路径)

6. 编写测试代码
    // 1.加载核心配置文件
    // 2.获取sqlSessionFactory工厂对象
    // 3.获取sqlSession会话对象
    // 4.执行sql
    // 5.打印结果
    // 6.释放资源
```

3.2 代码实现

1) 创建user数据表

```
CREATE DATABASE `mybatis_db`;
USE `mybatis_db`;

CREATE TABLE `user` (
  `id` int(11) NOT NULL auto_increment,
  `username` varchar(32) NOT NULL COMMENT '用户名称',
  `birthday` datetime default NULL COMMENT '生日',
  `sex` char(1) default NULL COMMENT '性别',
  `address` varchar(256) default NULL COMMENT '地址',
```

```
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- insert....
insert  into `user`(`id`,`username`,`birthday`,`sex`,`address`) values (1,'子慕','2020-11-11 00:00:00','男','北京海淀'),(2,'应颜','2020-12-12 00:00:00','男','北京海淀');
```

2) 导入MyBatis的坐标和其他相关坐标

```
<!--指定编码和版本-->
  <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.encoding>UTF-8</maven.compiler.encoding>
        <java.version>1.11</java.version>
        <maven.compiler.source>1.11</maven.compiler.source>
        <maven.compiler.target>1.11</maven.compiler.target>
 </properties>

<!--mybatis坐标-->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.5.4</version>
</dependency>
<!--mysql驱动坐标-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.6</version>
    <scope>runtime</scope>
</dependency>
<!--单元测试坐标-->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

3) 编写User实体

```
public class User {
    private Integer id;
    private String username;
    private Date birthday;
    private String sex;
    private String address;
    // getter/setter 略
}
```

4) 编写UserMapper映射文件

```
<?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">
<mapper namespace="UserMapper">
    <!--查询所有-->
    <select id="findAll" resultType="com.lagou.domain.User">
        select * from user
    </select>
</mapper>
```

5) 编写MyBatis核心文件

```
<?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>
    <!--环境配置-->
    <environments default="mysql">
        <!--使用MySQL环境-->
        <environment id="mysql">
            <!--使用JDBC类型事务管理器-->
            <transactionManager type="JDBC"></transactionManager>
            <!--使用连接池-->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver">
</property>
                <property name="url" value="jdbc:mysql:///mybatis_db">
</property>
                <property name="username" value="root"></property>
                <property name="password" value="root"></property>
            </dataSource>
        </environment>
    </environments>

    <!--加载映射配置-->
    <mappers>
        <mapper resource="com/lagou/mapper/UserMapper.xml"></mapper>
    </mappers>
</configuration>
```

6) 编写测试类

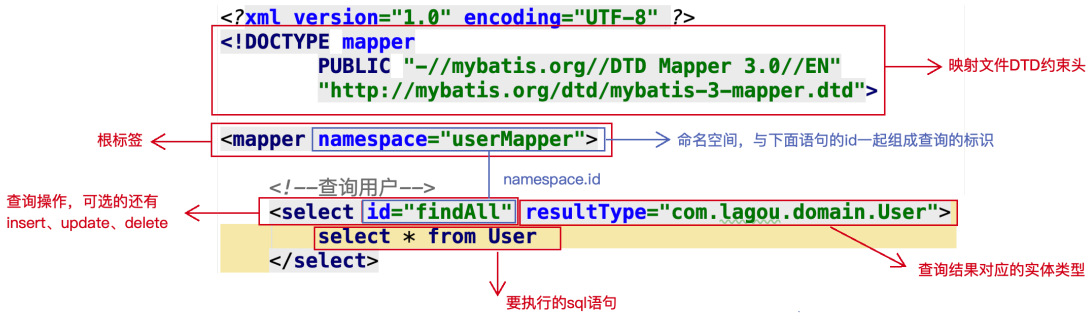
```
@Test
public void testFindAll() throws Exception {
    // 加载核心配置文件
    InputStream is = Resources.getResourceAsStream("SqlMapConfig.xml");
    // 获取SqlSessionFactory工厂对象
```

```
    SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(is);
    // 获取SqlSession会话对象
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 执行sql
    List<User> list = sqlSession.selectList("UserMapper.findAll");
    for (User user : list) {
        System.out.println(user);
    }
    // 释放资源
    sqlSession.close();
}
```

3.3 知识小结

1. 创建mybatis_db数据库和user表
2. 创建项目，导入依赖
3. 创建User实体类
4. 编写映射文件UserMapper.xml
5. 编写核心文件SqlMapConfig.xml
6. 编写测试类

四 Mybatis映射文件概述



五 Mybatis增删改查

5.1 新增

1）编写映射文件UserMapper.xml

```
<!--新增-->
<insert id="save" parameterType="com.lagou.domain.User">
    insert into user(username,birthday,sex,address)
        values(#{username},#{birthday},#{sex},#{address})
</insert>
```

2）编写测试类

```
@Test
public void testSave() throws Exception {
    // 加载核心配置文件
    InputStream is = Resources.getResourceAsStream("SqlMapConfig.xml");
    // 获取SqlSessionFactory工厂对象
    SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(is);
    // 获取SqlSession会话对象
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 执行sql
    User user = new User();
    user.setUsername("jack");
    user.setBirthday(new Date());
    user.setSex("男");
    user.setAddress("北京海淀");
    sqlSession.insert("UserMapper.save", user);

    // DML语句，手动提交事务
    sqlSession.commit();
    // 释放资源
    sqlSession.close();
}
```

3）新增注意事项

```
- 插入语句使用insert标签

- 在映射文件中使用parameterType属性指定要插入的数据类型

- Sql语句中使用#{实体属性名}方式引用实体中的属性值

- 插入操作使用的API是sqlSession.insert("命名空间.id",实体对象);

- 插入操作涉及数据库数据变化，所以要使用sqlSession对象显示的提交事务，即
sqlSession.commit()
```

5.2 修改

1）编写映射文件UserMapper.xml

```
<!--修改-->
<update id="update" parameterType="com.lagou.domain.User">
    update user set username = #{username},birthday = #{birthday},
        sex = #{sex},address = #{address} where id = #{id}
</update>
```

2）编写测试类

```
@Test
public void testUpdate() throws Exception {
    // 加载核心配置文件
    InputStream is = Resources.getResourceAsStream("SqlMapConfig.xml");
    // 获取SqlSessionFactory工厂对象
    SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(is);
    // 获取SqlSession会话对象
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 执行sql
    User user = new User();
    user.setId(4);
    user.setUsername("lucy");
    user.setBirthday(new Date());
    user.setSex("女");
    user.setAddress("北京朝阳");
    sqlSession.update("UserMapper.update", user);

    // DML语句，手动提交事务
    sqlSession.commit();
    // 释放资源
    sqlSession.close();
}
```

3）修改注意事项

```
- 修改语句使用update标签

- 修改操作使用的API是sqlSession.update("命名空间.id",实体对象);
```

5.3 删除

1）编写映射文件UserMapper.xml

```
<!--删除-->
<delete id="delete" parameterType="java.lang.Integer">
    delete from user where id = #{id}
</delete>
```

2）编写测试类

```
@Test
public void testDelete() throws Exception {
    // 加载核心配置文件
    InputStream is = Resources.getResourceAsStream("SqlMapConfig.xml");
    // 获取SqlSessionFactory工厂对象
    SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(is);
    // 获取SqlSession会话对象
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 执行sql
```

```
    sqlSession.delete("UserMapper.delete", 50);
    // DML语句，手动提交事务
    sqlSession.commit();
    // 释放资源
    sqlSession.close();
}
```

3）删除注意事项

```
- 删除语句使用delete标签

- Sql语句中使用#{任意字符串}方式引用传递的单个参数

- 删除操作使用的API是sqlSession.delete("命名空间.id",Object);
```

5.4 知识小结

```
* 查询
    代码：
        List<User> list = sqlSession.selectList("UserMapper.findAll");
    映射文件：
    <select id="findAll" resultType="com.lagou.domain.User">
        select * from user
    </select>

* 新增
    代码：
        sqlSession.insert("UserMapper.save", user);
    映射文件：
    <insert id="save" parameterType="com.lagou.domain.User">
        insert into user(username,birthday,sex,address)
            values(#{username},#{birthday},#{sex},#{address})
    </insert>

* 修改
    代码：
        sqlSession.update("UserMapper.update", user);
    映射文件：
    <update id="update" parameterType="com.lagou.domain.User">
        update user set username = #{username},birthday = #{birthday},
            sex = #{sex},address = #{address} where id = #{id}
    </update>

* 删除
    代码：
        sqlSession.delete("UserMapper.delete", 4);
    映射文件：
    <delete id="delete" parameterType="java.lang.Integer">
        delete from user where id = #{id}
    </delete>
```

六 Mybatis核心文件概述

6.1 MyBatis核心配置文件层级关系

MyBatis 的配置文件包含了会深深影响 MyBatis 行为的设置和属性信息。

配置文档的顶层结构如下：

- configuration（配置）
  - properties（属性）
  - settings（设置）
  - typeAliases（类型别名）
  - typeHandlers（类型处理器）
  - objectFactory（对象工厂）
  - plugins（插件）
  - environments（环境配置）
    - environment（环境变量）
      - transactionManager（事务管理器）
      - dataSource（数据源）
  - databaseIdProvider（数据库厂商标识）
  - mappers（映射器）

6.2 MyBatis常用配置解析

1) environments标签

数据库环境的配置，支持多环境配置

```
<environments default="development">
    <environment id="development">
        <transactionManager type="JDBC"/>
        <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>
```


1. 其中，事务管理器（transactionManager）类型有两种：
   - JDBC：
     这个配置就是直接使用了JDBC 的提交和回滚设置，它依赖于从数据源得到的连接来管理事务作用域。

   - MANAGED：
     这个配置几乎没做什么。它从来不提交或回滚一个连接，而是让容器来管理事务的整个生命周期。
     例如：mybatis与spring整合后，事务交给spring容器管理。

2. 其中，数据源（dataSource）常用类型有三种：
   - UNPOOLED：

这个数据源的实现只是每次被请求时打开和关闭连接。

- POOLED：
  这种数据源的实现利用“池”的概念将 JDBC 连接对象组织起来。

- JNDI ：
  这个数据源实现是为了能在如 EJB 或应用服务器这类容器中使用，容器可以集中或在外部配置数据源，然后放置一个 JNDI 上下文的数据源引用

2）properties标签

实际开发中，习惯将数据源的配置信息单独抽取成一个properties文件，该标签可以加载额外配置的properties：

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql:///mybatis_db
jdbc.username=root
jdbc.password=root
```

3）typeAliases标签

类型别名是为 Java 类型设置一个短的名字。

为了简化映射文件 Java 类型设置，mybatis框架为我们设置好的一些常用的类型的别名：

别名	数据类型
string	String
long	Long
int	Integer
double	Double
boolean	Boolean
... ...	... ...

原来的类型名称配置如下：

```
<!--查询用户-->
<select id="findAll" resultType="com.lagou.domain.User">
    select *from User
</select>
```

User全限定名称

配置typeAliases，为com.lagou.domain.User定义别名为user：

4）mappers标签

该标签的作用是加载映射的，加载方式有如下几种：

```
1. 使用相对于类路径的资源引用，例如：

    <mapper resource="org/mybatis/builder/userMapper.xml"/>

2. 使用完全限定资源定位符（URL），例如：

    <mapper url="file:///var/mappers/userMapper.xml"/>


《下面两种mapper代理开发中使用:暂时了解》

3. 使用映射器接口实现类的完全限定类名，例如：

    <mapper class="org.mybatis.builder.userMapper"/>

4. 将包内的映射器接口实现全部注册为映射器，例如：

    <package name="org.mybatis.builder"/>
```

6.3 知识小结

核心配置文件常用配置：

properties标签：该标签可以加载外部的properties文件

```
<properties resource="jdbc.properties"></properties>
```

typeAliases标签：设置类型别名

```
<typeAlias type="com.lagou.domain.User" alias="user"></typeAlias>
```

mappers标签：加载映射配置

```
<mapper resource="com/lagou/mapper/UserMapping.xml"></mapper>
```

environments标签：数据源环境配置

```
<environments default="development">
    <environment id="development">
        <transactionManager type="JDBC"/>
        <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>
```

七 Mybatis的API概述

7.1 API介绍

7.1 SqlSession工厂构建器SqlSessionFactoryBuilder

常用API：SqlSessionFactory build(InputStream inputStream)

通过加载mybatis的核心文件的输入流的形式构建一个SqlSessionFactory对象

```
String resource = "org/mybatis/builder/mybatis-config.xml";
InputStream inputStream = Resources.getResourceAsStream(resource);
SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
SqlSessionFactory factory = builder.build(inputStream);
```

其中， Resources 工具类，这个类在 org.apache.ibatis.io 包中。Resources 类帮助你从类路径下、文件系统或一个 web URL 中加载资源文件。

6.2 SqlSession工厂对象SqlSessionFactory

SqlSessionFactory 有多个个方法创建SqlSession 实例。常用的有如下两个：

方法	解释
openSession()	会默认开启一个事务，但事务不会自动提交，也就意味着需要手动提交该事务，更新操作数据才会持久化到数据库中
openSession(boolean autoCommit)	参数为是否自动提交，如果设置为true，那么不需要手动提交事务

6.3 SqlSession会话对象

SqlSession 实例在 MyBatis 中是非常强大的一个类。在这里你会看到所有执行语句、提交或回滚事务和获取映射器实例的方法。

执行语句的方法主要有：

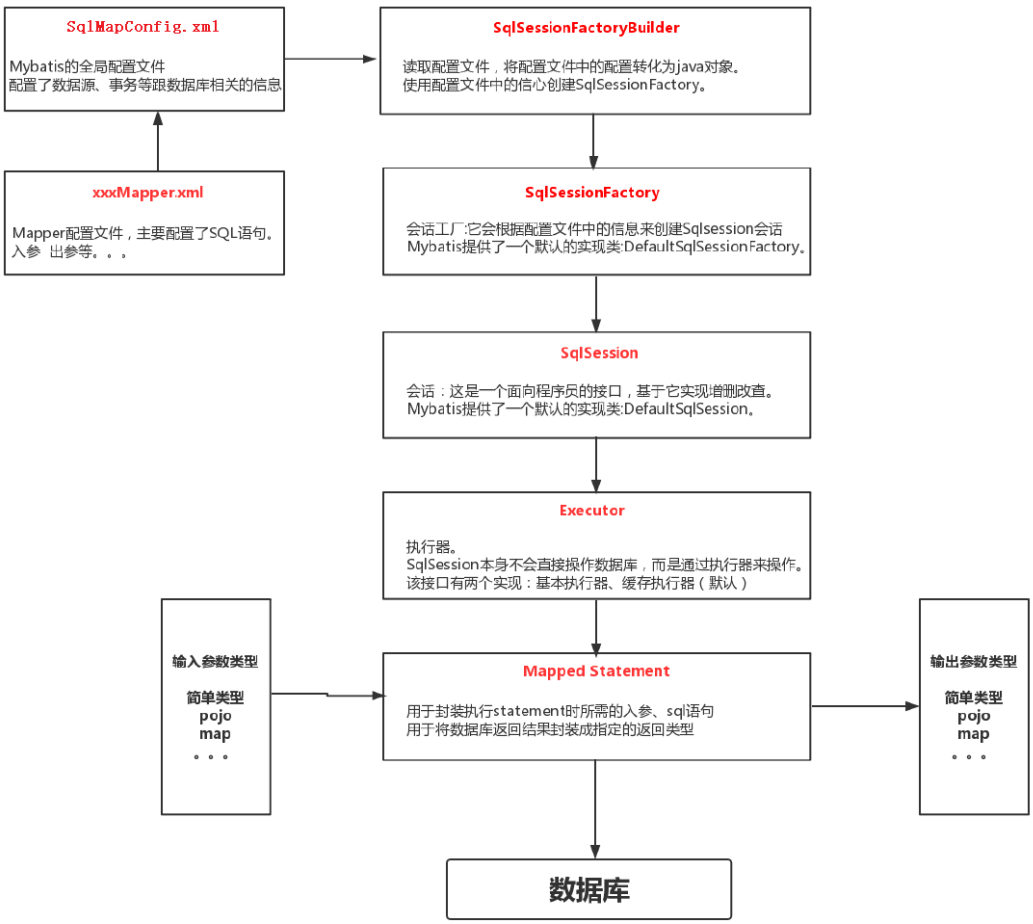
```
<T> T selectOne(String statement, Object parameter)
<E> List<E> selectList(String statement, Object parameter)
int insert(String statement, Object parameter)
int update(String statement, Object parameter)
int delete(String statement, Object parameter)
```

操作事务的方法主要有：

```
void commit()
void rollback()
```

、

7.2 Mybatis基本原理介绍



八 Mybatis的dao层开发使用

8.1 传统开发方式

1）编写UserMapper接口

```
public interface UserMapper {

    public List<User> findAll() throws Exception;

}
```

2）编写UserMapper实现

```
public class UserMapperImpl implements UserMapper {
    @Override
    public List<User> findAll() throws Exception {
        // 加载配置文件
        InputStream is = Resources.getResourceAsStream("SqlMapConfig.xml");
        // 获取SqlSessionFactory工厂对象
        SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(is);
        // 获取SqlSe会话对象
        SqlSession sqlSession = sqlSessionFactory.openSession();
        // 执行sql
```

```
        List<User> list = sqlSession.selectList("UserMapper.findAll");
        // 释放资源
        sqlSession.close();
        return list;
    }
}
```

3）编写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">
    <!--查询所有-->
    <select id="findAll" resultType="user">
        select * from user
    </select>
</mapper>
```

4）测试

```
@Test
public void testFindAll() throws Exception {
    // 创建UserMapper 实现类
    UserMapper userMapper = new UserMapperImpl();
    // 执行查询
    List<User> list = userMapper.findAll();
    for (User user : list) {
        System.out.println(user);
    }
}
```

5）知识小结

传统开发方式

```
1. 编写UserMapper接口

3. 编写UserMapper.xml
```

传统方式问题思考：

1.实现类中，存在mybatis模板代码重复

2.实现类调用方法时，xml中的sql statement 硬编码到java代码中

思考：能否只写接口，不写实现类。只编写接口和Mapper.xml即可？

因为在dao（mapper）的实现类中对sqlsession的使用方式很类似。因此mybatis提供了接口的动态代理。

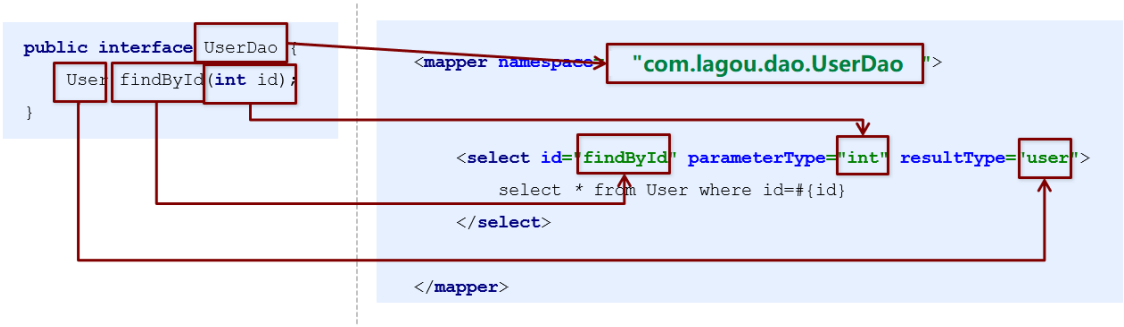
8.2 代理开发方式

1）介绍

采用 Mybatis 的基于接口代理方式实现 持久层 的开发，这种方式是我们后面进入企业的主流。

基于接口代理方式的开发只需要程序员编写 Mapper 接口，Mybatis 框架会为我们动态生成实现类的对象。

这种开发方式要求我们遵循一定的规范：



- Mapper.xml映射文件中的namespace与mapper接口的全限定名相同
- Mapper接口方法名和Mapper.xml映射文件中定义的每个statement的id相同
- Mapper接口方法的输入参数类型和mapper.xml映射文件中定义的每个sql的parameterType的类型相同
- Mapper接口方法的输出参数类型和mapper.xml映射文件中定义的每个sql的resultType的类型相同

Mapper 接口开发方法只需要程序员编写Mapper 接口（相当于Dao 接口），由Mybatis 框架根据接口定义创建接口的动态代理对象，代理对象的方法体同上边Dao接口实现类方法。

2）编写UserMapper接口

```
public interface UserMapper {

    public List<User> findAll() throws Exception;

}
```

3）编写UserMapper.xml

```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.lagou.mapper.UserMapper">
    <!--查询所有-->
    <select id="findAll" resultType="user">
        select * from user
    </select>
</mapper>
```

4）测试

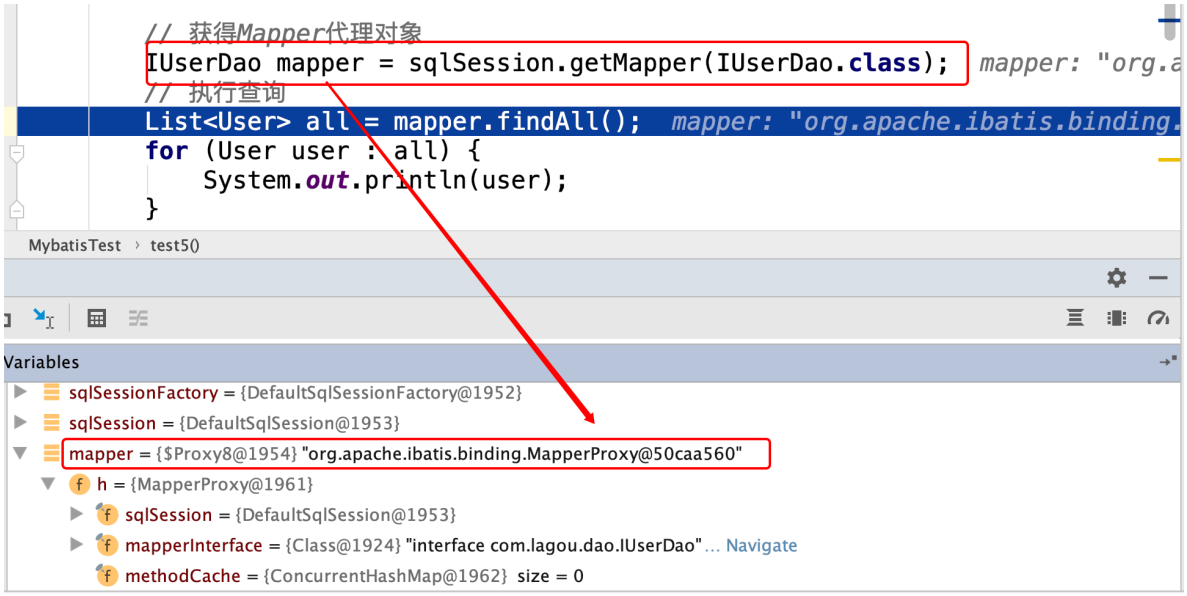
```java
@Test
public void testFindAll() throws Exception {
    // 加载核心配置文件
    InputStream is = Resources.getResourceAsStream("SqlMapConfig.xml");
    // 获得SqlSessionFactory工厂对象
    SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(is);
    // 获得SqlSession会话对象
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 获得Mapper代理对象
    UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
    // 执行查询
    List<User> list = userMapper.findAll();
    for (User user : list) {
        System.out.println(user);
    }
    // 释放资源
    sqlSession.close();
}
```

5）Mybatis基于接口代理方式的内部执行原理

我们的持久层现在只有一个接口，而接口是不实际干活的，那么是谁在做查询的实际工作呢?

下面通过追踪源码看一下:

1、通过追踪源码我们会发现，我们使用的mapper实际上是一个代理对象,是由MapperProxy代理产生的。



2、追踪MapperProxy的invoke方法会发现，其最终调用了mapperMethod.execute(sqlSession, args)

```
@Override
public Object invoke(Object proxy, Method method, Object[
  try {
    if (Object.class.equals(method.getDeclaringClass()))
      return method.invoke( obj: this, args);
    } else if (isDefaultMethod(method)) {
      return invokeDefaultMethod(proxy, method, args);
    }
  } catch (Throwable t) {
    throw ExceptionUtil.unwrapThrowable(t);
  }
  final MapperMethod mapperMethod = cachedMapperMethod(me
  return mapperMethod.execute(sqlSession, args);
}
```

3、进入execute方法会发现，最终工作的还是sqlSession