一、认识框架

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。

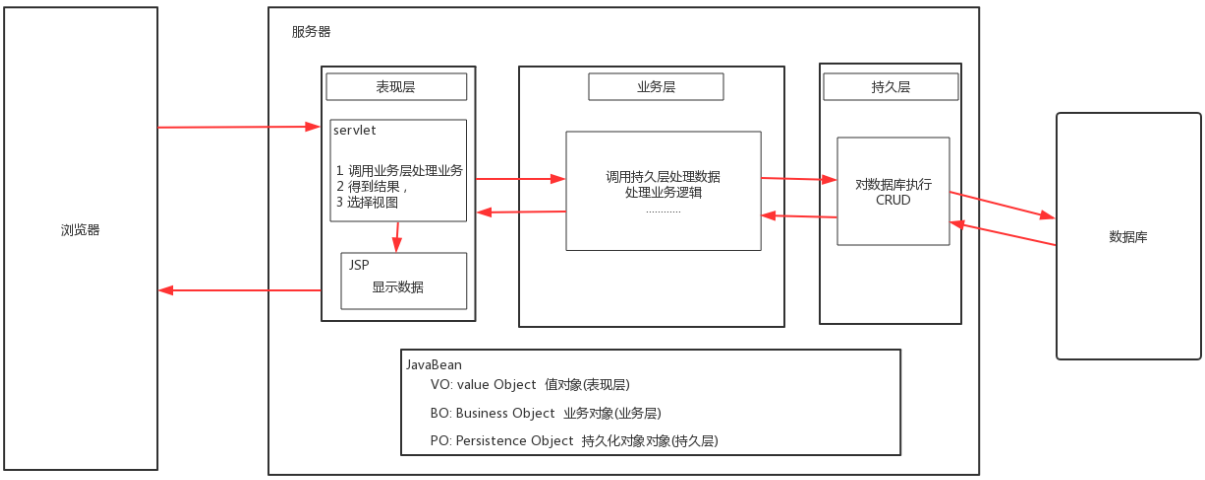
  因为它主要是为上层提供服务的，所以有人把业务层叫做Service层或Business层。

- 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 框架说明

1.2.1 什么是框架?

把重复的代码工作抽取出来，让程序员把精力专注在核心的业务代码实现上。

框架可以理解为一种套路，框架做好以后，接下来就可以按照套路做事情了。

1.2.2 为什么要使用框架?

因为学了它之后，我们的开发确实变得简单。

企业开发中都在用，不会它，你就无法正常进入企业进行开发工作。

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、认识Mybatis

2.1 ORM概述

ORM（object Relational Mapping）对象关系映射,是一个针对持久层的理论思想。

O----对象----类

R----关系----数据表

M----映射----在类和数据表之间建立的一一对应的关系（类名-->表名 属性名-->字段名）

ORM用来解决什么问题呢?

一句话说，就是ORM可以让我们以面向对象的形式操作数据库

总结：ORM就是建立实体类和数据库表之间的关系，从而达到操作实体类就相当于操作数据库表的目的。

```
常见的ORM框架有哪些?
    Hibernate
    JPA（SUN公司的规范，只有接口名，没有实现）
    Mybatis(半ORM框架，让我们既可以使用ORM的思想，又不受ORM的严格约束，表现为可以手动书写SQL)
```

2.2 Mybatis介绍

历史

- MyBatis本是apache的一个开源项目，名为iBatis。
- 2010年这个项目由apache迁移到了google，并且改名为MyBatis。
- 2013年迁移到Github。

简介

- MyBatis是一款优秀的持久层框架，它不需要像JDBC那样去写复杂代码、手动设置参数、繁琐的处理结果集
- 它采用简单的XML配置 + 接口方法的形式实现对数据库的增删改查，使得让程序员只关注sql本身

三、Mybatis简单入门

3.1 需求说明

将一个User对象持久化到数据库中去

3.2 数据库环境准备

3.2.1 创建表

```
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;
```

3.2.2 添加测试数据

```
insert  into `user`(`id`,`username`,`birthday`,`sex`,`address`) values (41,'老王','2018-02-27 17:47:08','男','北京'),(42,'小二王','2018-03-02 15:09:37','女','北京金燕龙'),(43,'小二王','2018-03-04 11:34:34','女','北京金燕龙'),(45,'传智播客','2018-03-04 12:04:06','男','北京金燕龙'),(46,'老王','2018-03-07 17:37:26','男','北京'),(48,'小马宝莉','2018-03-08 11:44:00','女','北京修正');
```

3.3 创建工程并引入坐标

```
<dependencies>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.46</version>
    </dependency>
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.5.0</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
```

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

3.4 配置mybatis主配置文件

```
<?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>
    <!--指定默认的环境-->
    <environments default="mysql">
        <!--环境配置，可以存在多个-->
        <environment id="mysql">
            <!--使用了JDBC的事务管理-->
            <transactionManager type="JDBC"></transactionManager>
            <!--先配置为POOLED,代表以池的形式管理连接-->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver" />
                <property name="url" value="jdbc:mysql://127.0.0.1:3306/mybatisdb" />
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>
</configuration>
```

3.5 编写xml映射文件

```
文件位置:\resources\com\itheima\mapper\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">
        <insert id="save" parameterType="com.itheima.domain.User">
            insert into user (username, birthday, sex, address) values
            (#{username}, #{birthday}, #{sex}, #{address})
        </insert>
</mapper>
```

3.5 将Mapper文件加入到主配置文件中

```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>
    <environments default="mysql">
        <!--省略数据库配置-->
    </environments>
    <!--引入映射文件-->
    <mappers>
        <mapper resource="mapper/UserMapper.xml"/>
    </mappers>
</configuration>
```

3.6 测试

```java
    @Test
    public void save() throws Exception {
        //把主配置文件变成流
        InputStream is = Resources.getResourceAsStream("sqlMapConfig.xml");
        //解析器配置文件并构建一个SqlSessionFactroy工厂对象
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);
        //生产SqlSession对象【这个对象就是mybatis真正操作数据库的对象】
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //创建用户对象
        User paramUser = new User();
        paramUser.setUsername("传智播客");
        paramUser.setBirthday(new Date());
        paramUser.setSex("男");
        paramUser.setAddress("北京");
        //执行数据库操作【使用mybatis自己封装的增删改查操作】，里面需要传一个sql语句的id
        sqlSession.insert("userMapper.save", paramUser);
        //提交事务
        sqlSession.commit();
        //关闭资源
        sqlSession.close();
    }
```

四、Mybatis实现Dao层的两种操作方式

4.1 传统方式实现dao

4.1.1 添加Mapper接口【注：mybatis中dao也叫mapper】

```
public interface UserMapper {
    public void save(User user) throws IOException;
}
```

4.1.2 添加Mapper实现类

```
public class UserMapperImpl implements UserMapper {
    @Override
    public void save(User user) throws IOException {
        //把主配置文件变成流
        InputStream is = Resources.getResourceAsStream("sqlMapConfig.xml");
        //解析器配置文件并生成一个SqlSessionFactory工厂对象
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);
        //生成SqlSession对象【这个对象就是mybatis真正操作数据库的对象】
        SqlSession sqlSession = sqlSessionFactory.openSession();
        //执行数据库操作【使用mybatis自己封装的增删改查操作】，里面需要传一个sql语句的id
        sqlSession.insert("userMapper.save", user);
        //提交事务
        sqlSession.commit();
        //关闭资源
        sqlSession.close();
    }
}
```

4.1.3 编写XML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">
        <insert id="save" parameterType="com.itheima.domain.User">
            insert into user (username, birthday, sex, address) values
            (#{username}, #{birthday}, #{sex}, #{address})
        </insert>
</mapper>
```

4.1.4 测试

```
    public class UserTest {
    @Test
    public void save() throws Exception {
        UserMapper userMapper = new UserMapperImpl();
        User paramUser = new User();
        paramUser.setUsername("传智播客");
        paramUser.setBirthday(new Date());
        paramUser.setSex("男");
        paramUser.setAddress("北京");
        userMapper.save(paramUser);
    }
}
```

4.1.5 小结

SqlSession用于CRUD的API主要有下面几个：

- int insert(String statement, Object parameter) 新增
- int update(String statement, Object parameter) 更新
- int delete(String statement, Object parameter):删除
- T selectOne(String statement, Object parameter):查询返回一条结果
- List selectList(String statement, Object parameter):查询返回多条结果

4.2 接口代理方式实现dao（重点）

基于接口代理方式开发是我们的一个主流方式，必须掌握。

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接口的类路径与Mapper.xml 文件中的namespace相同
- Mapper接口方法名和Mapper.xml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中定义的每个sql的resultType的类型相同

4.2.1 编写Mapper接口

```
public interface UserMapper {
    public void save(User user) throws IOException;
}
```

4.2.2 编写Mapper.xml

注意此时namespace必须是dao接口的全限定名，sql语句的id必须与对应接口方法名一致。

```
<?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.mapper.UserMapper">
        <insert id="save" parameterType="com.itheima.domain.User">
            insert into user (username, birthday, sex, address) values
            (#{username}, #{birthday}, #{sex}, #{address})
        </insert>
</mapper>
```

4.2.3 测试

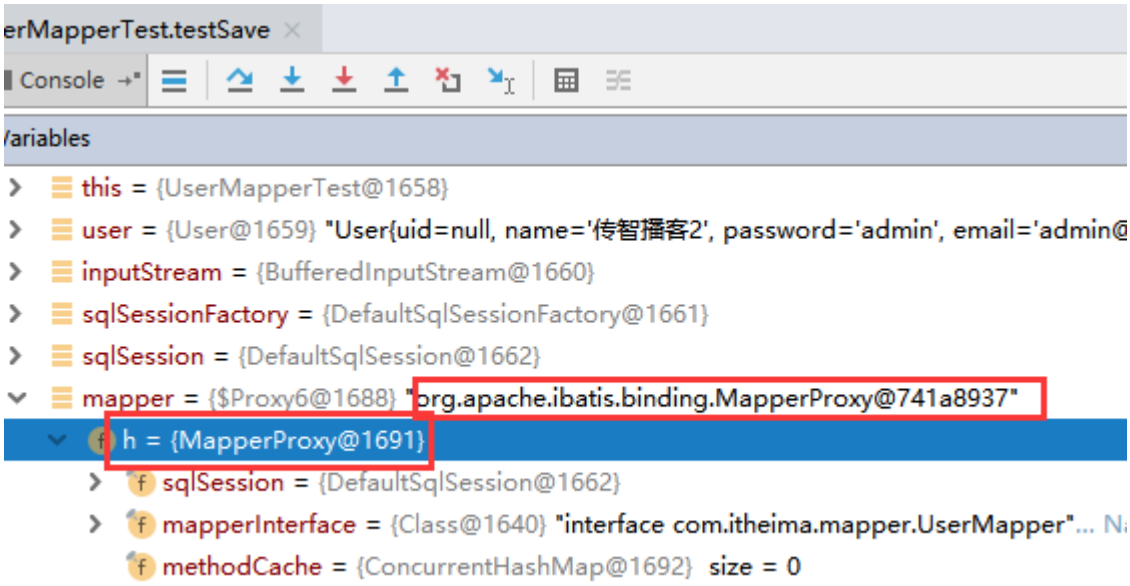
```
@Test
public void testSave() throws IOException {
        InputStream inputStream = Resources.getResourceAsStream("mybatis-config.xml");
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
        SqlSession sqlSession = sqlSessionFactory.openSession();

        User paramUser = new User();
        paramUser.setUsername("传智播客");
        paramUser.setBirthday(new Date());
        paramUser.setSex("男");
        paramUser.setAddress("北京");

        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
        userMapper.save(paramUser);

        sqlSession.commit();
        sqlSession.close();
}
```

4.2.4 Mybatis基于接口代理方式的内部执行原理

此时我们的Dao层只有一个接口，而接口仅是定义了一个规范，并未真正与数据库交互，那么是谁在做save的实际工作呢?

下面通过追踪源码看一下:

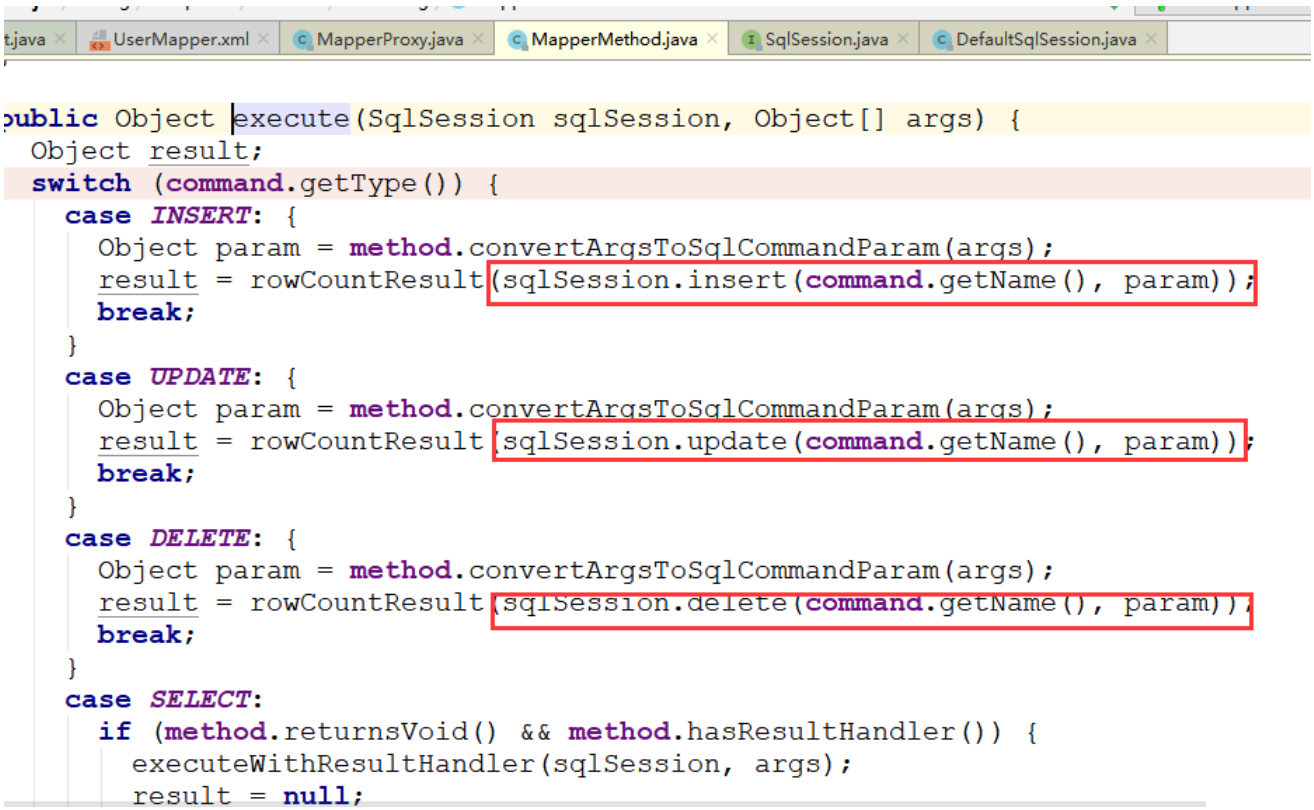
1、通过追踪源码我们会发现，我们使用的mapper实际上是一个代理对象,是由MapperProxy代理产生的。



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。



4.2.5 小结

使用了基于接口代理的方式实现后，我们程序员所做的事情就转化成了这样：

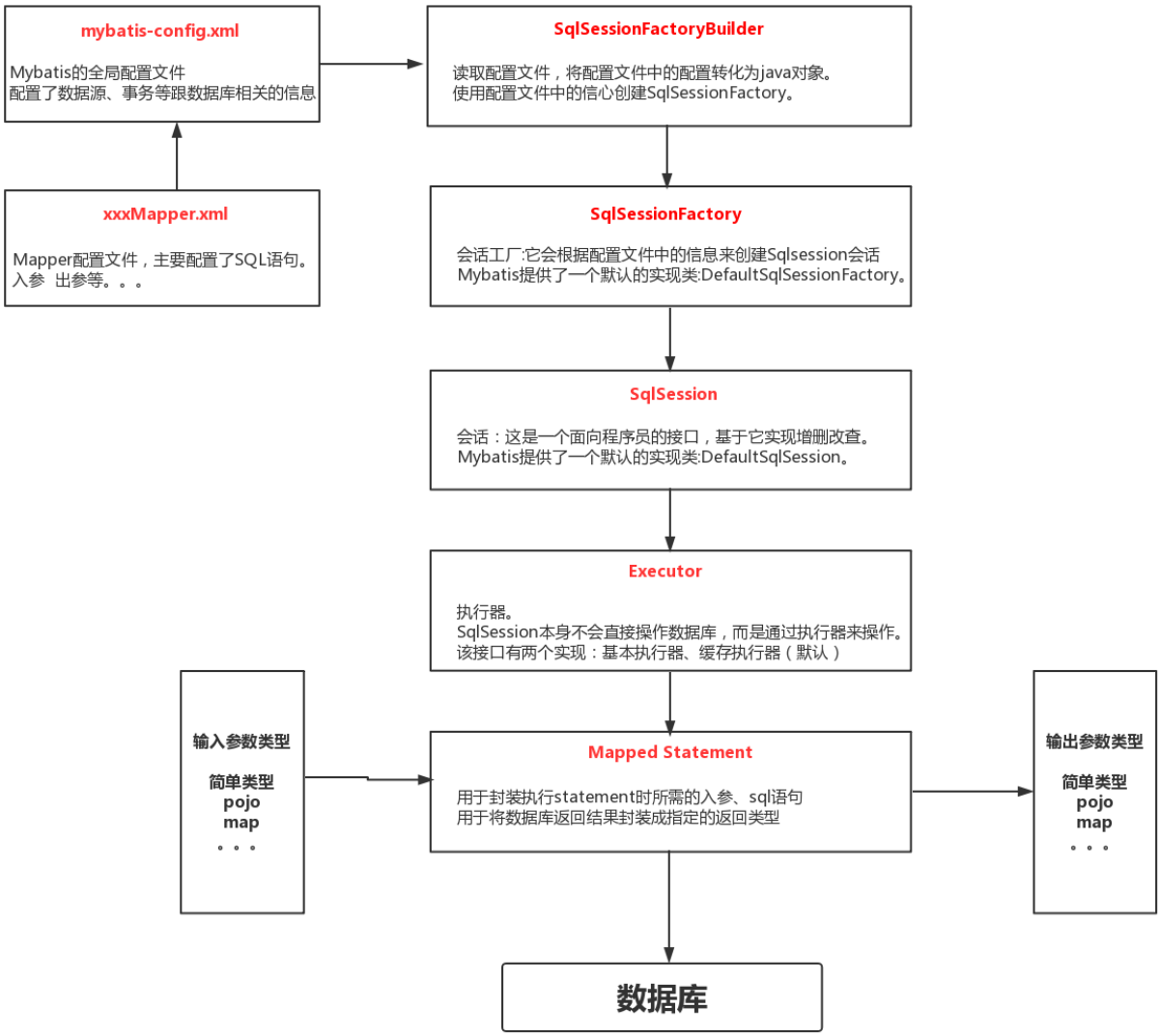
1. 编写接口文件，在接口中定义方法
2. 编写XML映射文件，在XML中书写SQL语句

五、Mybatis的API

5.1 API介绍

```
Resources
    加载mybatis的配置文件。

SqlSessionFactoryBuilder
    利用Resources指定的资源，将配置信息加载到内存中，还会加载mybatis配置文件中指定的所有映射配置信息，
并用特定的对象实例进行保存，从而创建SqlSessionFactory对象。

SqlSessionFactory
    这是一个工厂对象，对于这种创建和销毁都非常耗费资源的重量级对象，一个项目中只需要存在一个即可。
    也就是说，它的生命周期跟项目的生命周期是一致的(项目不死，我不销毁)
    它的任务是创建SqlSession。

SqlSession
    这是Mybatis的一个核心对象。我们基于这个对象可以实现对数据的CRUD操作。
    对于这个对象应做到每个线程独有，每次用时打开，用完关闭。
```

5.2 Mybatis基本原理介绍



六、简化mybatis开发

6.1 抽取公用代码

6.1.1 封装生产SqlSession对象的工具类

```
public class CreateSqlSessionUtils {

    private static SqlSessionFactory sqlSessionFactory;

    static {
        try {
            InputStream is = Resources.getResourceAsStream("sqlMapConfig.xml");
```

```
            sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static SqlSession openSqlSession(){
        return sqlSessionFactory.openSession();
    }
}
```

6.1.2 抽取junit测试基础类

```
public class MybatisJunitTestUtils {
    protected SqlSession sqlSession;

    @Before
    public void before(){
        sqlSession = CreateSqlSessionUtils.openSqlSession();
    }

    @After
    public void after(){
        sqlSession.commit();
        sqlSession.close();
    }
}
```

6.2 给自己编写的javaBean对象起别名

6.2.1 方式一：指定一个对象，给其起别名

```
<configuration>
    <!--给对象起别名-->
    <typeAliases>
        <!--给指定的对象起别名，使用的使用别名的大小写随意-->
        <typeAlias type="com.itheima.domain.User" alias="user"/>
    </typeAliases>
    <environments default="development">
        ……省略连接数据库信息
    </environments>
    <mappers>
        ……省略映射文件信息
    </mappers>
</configuration>
```

6.2.2 方式二：指定一个包，给其下所有对象起别名，别名默认就是当前对象的类名

```
<configuration>
    <!--给对象起别名-->
    <typeAliases>
        <!--指定一个包，给其下所有对象起别名，别名就是当前类名，大小写随意-->
        <package name="com.itheima.domain"/>
    </typeAliases>
    <environments default="development">
        ……省略连接数据库信息
    </environments>
    <mappers>
        ……省略映射文件信息
    </mappers>
</configuration>
```

注：之后在mybatis的xml配置文件中就无需再写javaBean的全限定名了，直接可以使用别名。

七、基于接口代理实现CRUD

7.1 编写接口

```
//Mapper接口
public interface UserMapper {

    //保存
    void save(User user);

    //根据UID查询
    User findByUid(Integer i);

    //根据UID更新
    void update(User user);

    //根据ID删除
    void deleteById(Integer id);
}
```

7.2 编写xml映射

```
<mapper namespace="com.itheima.dao.UserDao">
    <insert id="save" parameterType="com.itheima.domain.User">
        insert into user (username, birthday, sex, address) values
        (#{username}, #{birthday}, #{sex}, #{address})
    </insert>

    <select id="findByUid" parameterType="int" resultType="com.itheima.domain.User">
        select * from user where id = #{id}
    </select>
```

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

    <update id="deleteById" parameterType="int">
        delete from user where id = #{id}
    </update>
</mapper>
```

7.3 编写测试

```
//测试类
public class CRUDTest extends MybatisJunitTestUtils {

    //保存
    @Test
    public void testSave() throws Exception {
        User paramUser = new User();
        paramUser.setUsername("小明");
        paramUser.setBirthday(new Date());
        paramUser.setSex("女");
        paramUser.setAddress("北京");

        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
        userMapper.save(paramUser);
    }

    //根据ID查询
    @Test
    public void testFindByUid() {
        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
        User user = userMapper.findByUid(1);
        System.out.println(user);
    }


    //根据ID修改
    @Test
    public void testUpdate() {
        User paramUser = new User();
        paramUser.setId(95);
        paramUser.setUsername("小明");
        paramUser.setBirthday(new Date());
        paramUser.setSex("男");
        paramUser.setAddress("北京");
        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
        userMapper.update(paramUser);
```

```
    }

    //根据ID删除
    @Test
    public void testDeleteByUid() {
        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
        userMapper.deleteById(2);
    }
}
```

八、Mybatis中的查询

8.1 查询所有

8.1.1使用resultType

```
<!--使用resultType映射结果
        此时必须保证SQL语句返回的字段跟属性名是一致的，否则映射不上
    -->
<select id="findAll" resultType="com.itheima.domain.User">
   select * from user
</select>
```

8.1.2使用resultMap

```
<!--映射结果集  [将数据表的字段和类的属性做一个映射]-->
<!--
        <resultMap手动指定对象关系映射
        id="userMap" 当前resultMap的唯一标识
        type="user" 当前对象关系映射中的对象
 -->
<resultMap id="userMap" type="user">
    <!--
        <id 表示当前是主键列property指定pojo中的属性，column指定resultSet中的列名
    -->
    <id property="id" column="_id"/>
    <!--普通列用result-->
    <result property="username" column="_username"/>
    <result property="birthday" column="_birthday"/>
    <result property="sex" column="_sex"/>
    <result property="address" column="_address"/>
</resultMap>

<!--使用resultMap映射结果-->
<select id="findAll" resultMap="userMap">
  SELECT id _id, username _username, birthday _birthday, sex _sex, address _address FROM USER
```

```
</select>
```

8.2 条件查询包含多个条件

8.2.1 使用包装类【顾名思义把多个条件包装到一个类中】

8.2.1.1 编写接口

```
public User findByIdAndUserName(User user);
```

8.2.1.1 编写xml

```
<select id="findByIdAndUserName" parameterType="user" resultType="user">
    select * from user where id=#{id} and username=#{username}
</select>
```

8.2.2 使用arg或param来标识参数占位符

8.2.2.1 编写接口

```
public User findByIdAndUserNameTwoParam(Integer id, String username);
```

8.2.2.1 编写xml

```
<select id="findByIdAndUserNameTwoParam" resultType="user">
    select * from user where id=#{arg0} and username=#{arg1}
</select>
```

或者

```
<select id="findByIdAndUserNameTwoParam" resultType="user">
    select * from user where id=#{param1} and username=#{param2}
</select>
```

8.2.3 使用@Param注解来指定参数对应关系

8.2.3.1 编写接口

```
public User findByIdAndUserName(@Param("id") Integer id, @Param("username") String username);
```

8.2.3.1 编写xml

```
<select id="findByIdAndUserName" resultType="user">
    select * from user where id=#{id} and username=#{username}
</select>
```

8.2.3 模糊查询

8.2.3.1 方式一：传入参数的时候，直接传入"%王%"

```
<select id="findByUsername" parameterType="string" resultType="user">
    select * from user where username like #{username}
</select>
```

8.2.3.2 方式二：使用#{}将%直接拼到sql语句中【不推荐】

注：oracle中除了给表或者列起别名是不能使用双引号的

```
<select id="findByUsername" parameterType="string" resultType="user">
    select * from user where username like "%"#{username}"%"
</select>
```

8.2.3.3 方式三：使用${}将%直接拼到sql语句中

```
<select id="findByUsername" parameterType="string" resultType="user">
    select * from user where username like '%${value}%'
</select>
```

8.2.3.4 方式四：使用concat函数拼接字符串【不推荐】

注：oracle中concat函数只能拼接两个字符串

```
<select id="findByUsername" parameterType="string" resultType="user">
    select * from user where username like concat('%',#{username},'%')
</select>
```

面试直达：请说出Mybatis中#{}和${}的区别**

```
#{}表示占位符，${}表示字符串拼接
#{}是生成预编译的sql，执行效率高
${}是每次请求都会重新编译sql，执行效率低
${}可能会引起SQL注入问题，#{}不会
两者都可以接受简单类型的值和pojo类型的属性值，但是${}接收简单类型数据只能使用${value}，#{}可以是随意值
```