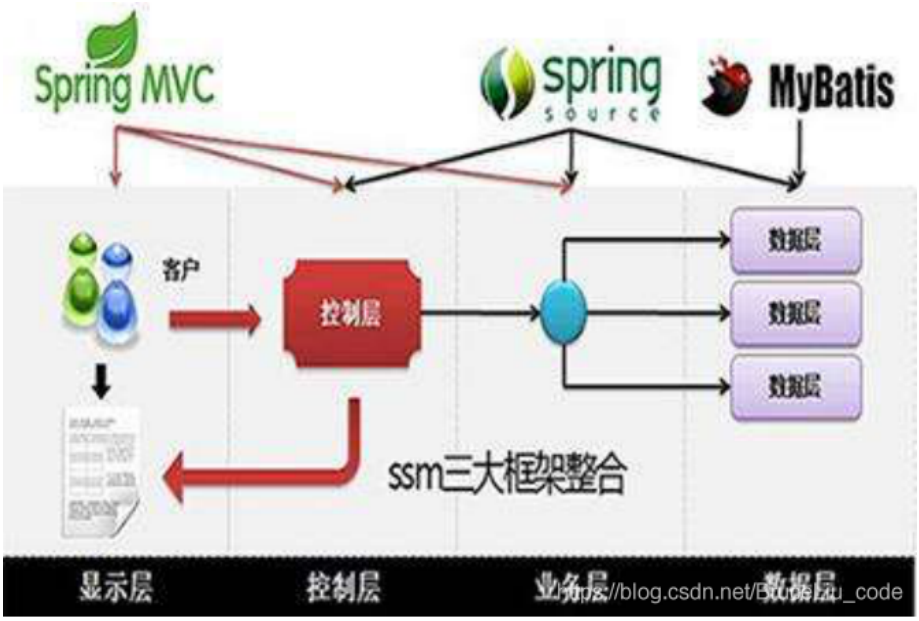
1.基本介绍



Spring

Spring是一个开源框架，Spring是于2003 年兴起的一个轻量级的Java 开发框架，由Rod Johnson 在其著作Expert One-On-One J2EE Development and Design中阐述的部分理念和原型衍生而来。它是为了解决企业应用开发的复杂性而创建的。Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益。 简单来说，Spring是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架。

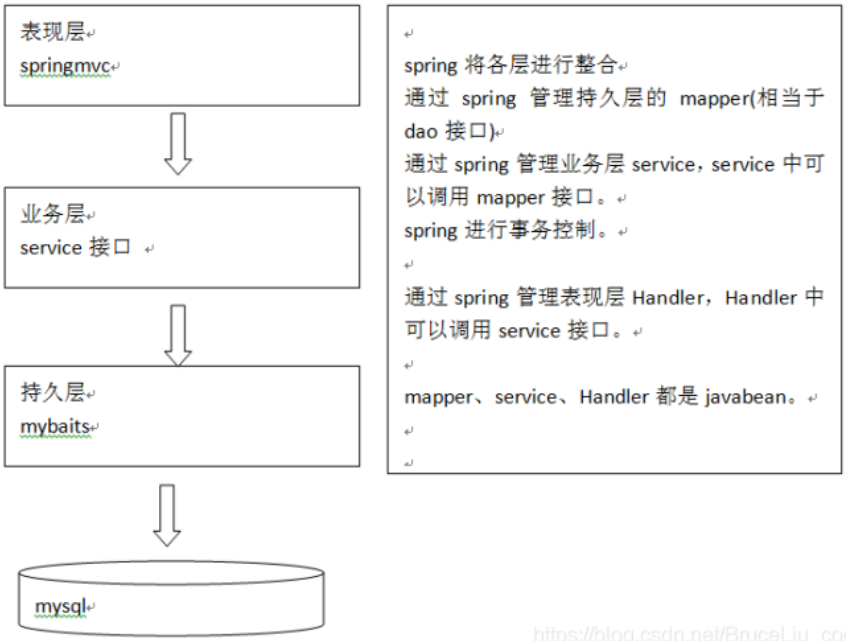
SpringMVC

Spring MVC属于SpringFrameWork的后续产品，已经融合在Spring Web Flow里面。Spring MVC 分离了控制器、模型对象、分派器以及处理程序对象的角色，这种分离让它们更容易进行定制。

MyBatis

MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis 。MyBatis是一个基于Java的持久层框架。iBATIS提供的持久层框架包括SQL Maps和Data Access Objects（DAO）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。MyBatis 使用简单的 XML或注解用于配置和原始映射，将接口和 Java 的POJOs（Plain Old Java Objects，普通的 Java对象）映射成数据库中的记录。

2.整合思路

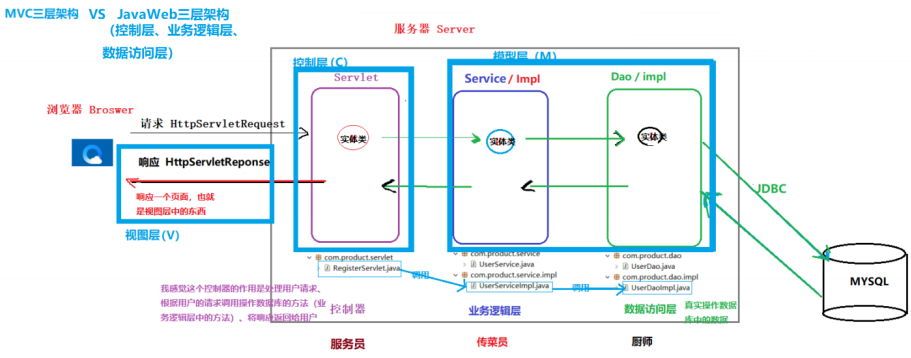


①表现层，也就是 Controller,由 SpringMVC 来控制，而SpringMVC 是Spring 的一个模块，故不需要整合。

②业务层，也就是 service，通常由 Spring 来管理 service 接口，我们会使用 xml 配置的方式来将 service 接口配置到 spring 配置文件中。而且事务控制一般也是在 service 层进行配置。

③持久层，也就是 dao 层，而且包括实体类，由 MyBatis 来管理，通过 spring 来管理 mapper 接口，使用mapper的扫描器自动扫描mapper接口在spring中进行注册。

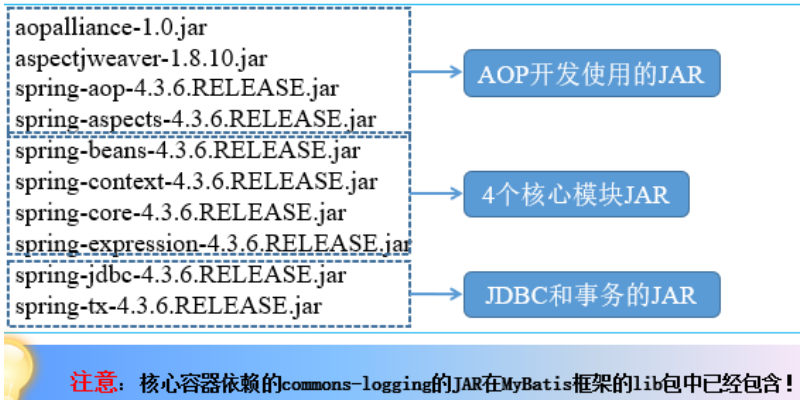
很明显，spring 在三大框架的整合中占据至关重要的地位，类似于一个大管家，将 MyBatis 和 SpringMVC 揉合在一起。



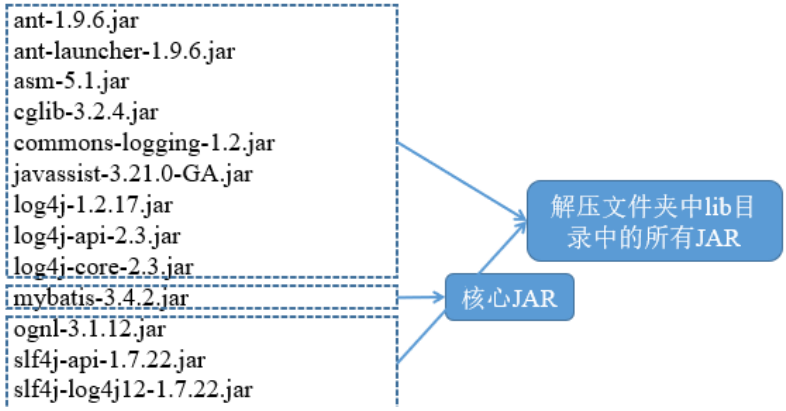
2.1 依赖的jar包

要实现MyBatis与Spring的整合，很明显需要这两个框架的JAR包，但是只使用这两个框架中所提供的JAR包是不够的，还需要其他的JAR包来配合使用，整合时所需准备的JAR包具体如下。

1. Spring框架所需要的JAR包



2. Mybatis框架所需要的JAR包



3. Mybatis与Spring整合的中间件JAR
   - mybatis-spring-1.3.1.jar
4. 数据库驱动JAR（MySQL）
   - mybatis-spring-1.3.1.jar
5. 数据源所需JAR（DBCP）
   - commons-dbcp2-2.1.1.jar
   - commons-pool2-2.4.2.jar

3.整合步骤

3.1.准备测试数据

```
/*
Navicat MySQL Data Transfer

Source Server         : mysql
Source Server Version : 50521
Source Host           : localhost:3306
Source Database       : ssm

Target Server Type    : MYSQL
Target Server Version : 50521
File Encoding         : 65001

Date: 2020-08-04 13:53:35
*/
```

```
SET FOREIGN_KEY_CHECKS=0;

-- ----------------------------
-- Table structure for t_user
-- ----------------------------
DROP TABLE IF EXISTS `t_user`;
CREATE TABLE `t_user` (
  `userId` int(11) NOT NULL AUTO_INCREMENT,
  `userLoginName` varchar(10) DEFAULT NULL,
  `userPhone` varchar(255) DEFAULT NULL,
  `userAge` int(6) DEFAULT NULL,
  `userPwd` varchar(10) DEFAULT NULL,
  `userName` varchar(15) DEFAULT NULL,
  `state` int(11) DEFAULT NULL COMMENT 'state 1启用 0冻结',
  `createTime` date DEFAULT NULL,
  `delState` int(11) DEFAULT NULL COMMENT 'delState 删除状
态1 删除 0未删除',
  PRIMARY KEY (`userId`)
) ENGINE=InnoDB AUTO_INCREMENT=18 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of t_user
-- ----------------------------
INSERT INTO `t_user` VALUES ('1', 'luoxian', null, null,
'123456', '杨晓林', '1', '2018-07-09', '0');
INSERT INTO `t_user` VALUES ('2', '小王', null, null,
'123456', '王二小', '1', '2018-07-14', '0');
INSERT INTO `t_user` VALUES ('3', 'qazwsx', null, null,
'qazwsx', '吕杰', '0', '2018-09-05', '1');
INSERT INTO `t_user` VALUES ('4', 'luoxianxin', null,
null, '123456', '吕杰', '1', '2018-09-06', '1');
INSERT INTO `t_user` VALUES ('8', 'zhangsan', null, null,
'324234', '张三', '1', '2018-09-04', '1');
INSERT INTO `t_user` VALUES ('9', 'lisi', null, null,
'1234567', '小张', '0', '2018-09-26', '0');
INSERT INTO `t_user` VALUES ('10', 'lisi1', null, null,
'12345671', '小张1', '1', '2018-09-26', '1');
INSERT INTO `t_user` VALUES ('11', 'lisi2', null, null,
'12345672', '小张2', '1', '2018-09-26', '1');
INSERT INTO `t_user` VALUES ('12', 'lisi3', null, null,
'12345673', '小张3', '1', '2018-09-26', '0');
INSERT INTO `t_user` VALUES ('13', 'lisi4', null, null,
'12345674', '小张4', '1', '2018-09-26', '1');
INSERT INTO `t_user` VALUES ('14', 'lisi5', null, null,
'12345675', '小张5', '0', '2018-09-26', '1');
INSERT INTO `t_user` VALUES ('15', 'lisi6', null, null,
'12345676', '小张6', '1', '2018-09-26', '0');
INSERT INTO `t_user` VALUES ('16', 'lisi7', null, null,
'12345677', '小张7', '1', '2018-09-26', '0');
INSERT INTO `t_user` VALUES ('17', 'lisi8', null, null,
'12345678', '小张8', '0', '2018-09-26', '0');
```

3.2.导入依赖

```
<dependencies>

        <!--导入Spring框架和MyBatis框架的所有依赖-->
        <!-- Junit测试 -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>

        <!--导入Spring的依赖-->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.2</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>5.3.2</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>5.3.2</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>5.3.2</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context-
support</artifactId>
            <version>5.3.2</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-expression</artifactId>
            <version>5.3.2</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
```

```
            <artifactId>spring-test</artifactId>
            <version>5.3.2</version>
        </dependency>

        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.14</version>
        </dependency>

        <!--Lombok -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.16.18</version>
            <scope>provided</scope>
        </dependency>

        <!-- myBatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.4.5</version>
        </dependency>

        <!-- mysql驱动包 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8</version>
        </dependency>

        <!--Spring和MyBatis整合包专用包-->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
            <version>1.3.0</version>
        </dependency>

        <!--数据库连接池依赖-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.1.10</version>
        </dependency>

    </dependencies>
```

3.3.新建jdbc.properties

```
jdbc.driver=com.mysql.cj.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/sm?
useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=U
TC
jdbc.username=root
jdbc.password=271441
```

3.4. 新建日志文件log4j.properties

```
# log4J日志框架的配置文件 文件名字不能改
# DEBUG 表示日志的级别  调试
# Console 日志打印在控制台
log4j.rootLogger=DEBUG, Console
log4j.appender.Console=org.apache.log4j.ConsoleAppender
log4j.appender.Console.layout=org.apache.log4j.PatternLayo
ut
log4j.appender.Console.layout.ConversionPattern=%d [%t]
%-5p [%c] - %m%n

# 哪些日志需要打印
log4j.logger.java.sql.ResultSet=INFO
log4j.logger.org.apache=INFO
log4j.logger.java.sql.Connection=DEBUG
log4j.logger.java.sql.Statement=DEBUG
log4j.logger.java.sql.PreparedStatement=DEBUG
```

3.5. 新建applicationContext.xml

这个配置文件中的内容 === （Spring中applicationContext.xml中的内容） + （Mybatis中mybatis-config.xml配置文件中的内容）

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xmlns:context="http://www.springframework.org/schema/conte
xt"

xsi:schemaLocation="http://www.springframework.org/schema/
beans http://www.springframework.org/schema/beans/spring-
beans.xsd http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-
context.xsd">

    <!--
        Spring整合MyBatis，一共五步：
            第1步：扫描指定包下的所有注解用于生成Bean实例或者注入
属性
            第2~4步：配置SqlSession相关，最后配置好的
SqlSession会在第5步中使用
```

```
            第5步：采用自动扫描的形式来配置MyBatis中的映射器（具体内容看3.7.3节第二种方式），本质也是给Mapper层的接口生成实现类的实例对象；可以在Mapper层的接口上使用注解Mapper、MapperScan来代替这段bean标签

    -->


    <!--1.扫描包-->
    <context:component-scan base-package="com.sm.*"/>

    <!--2.加载数据库配置文件-->
    <context:property-placeholder
location="classpath:jdbc.properties" file-encoding="UTF-
8"/>

    <!--3.配置数据源 如何连接数据库-->
    <bean id="ds"
class="com.alibaba.druid.pool.DruidDataSource">
        <property name="driverClassName"
value="${jdbc.driver}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username"
value="${jdbc.username}"/>
        <property name="password"
value="${jdbc.password}"/>
    </bean>

    <!--4.Spring框架帮MyBatis生成sqlSession
        MyBatis中的sqlSession交给来了Spring管理；
        Spring容器生成的SqlSession的Bean实例会在
        Mapper层接口的实例（也就是第5步配置生成的代
        理实例）中用到；
        这个标签中class属性的值的作用就是构
        建SsqlSessionFactory
    -->
    <bean
class="org.mybatis.spring.SqlSessionFactoryBean">
        <!--配置数据源-->
        <property name="dataSource" ref="ds"/>
        <!--指定MyBatis的Mapper文件位置-->
        <property name="mapperLocations"
value="classpath*:mappers/*.xml"/>
        <!--指定MyBatis实体类的别名-->
        <property name="typeAliasesPackage"
value="com.sm.bean"/>
    </bean>


    <!--5.spring框架给Mapper层接口生成对象
        这个bean标签也就是无需在数据访问层中使
        用@Repository注解的原因
```

```
    -->
    <bean
class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <!--Spring 框架会自动通过动态代理模式(反射)
            生成对象（这个对象是Mapper层接口的实现类
            的实例）。
            在这个对象中要用到第四步中配置
            在Spring容器中生成的SqlSession的Bean实例
        -->
        <property name="basePackage"
value="com.sm.mapper"/>
    </bean>

</beans>
```

3.6.新建实体类

```
package com.sm.bean;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * @BelongsProject: MyBatis-Spring
 * @BelongsPackage: com.sm.bean
 * @CreateTime: 2020-10-10 16:37
 * @Description: TODO
 */
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {

    private Integer userId;
    private String userLoginName;
    private String userPhone;
    private Integer userAge;
    private String userPwd;
    private String userName;
    private Integer state;
    private String createTime;
    private Integer delState;

    public User(String userLoginName, String userPhone,
Integer userAge, String userPwd, String userName) {
        this.userLoginName = userLoginName;
        this.userPhone = userPhone;
        this.userAge = userAge;
        this.userPwd = userPwd;
        this.userName = userName;
```

```
    }
}
```

3.7.新建Mapper层

3.7.1Mapper层接口

```
package com.sm.mapper;

import com.sm.bean.User;

import java.util.List;

/**
 * @BelongsProject: MyBatis-Spring
 * @BelongsPackage: com.sm.mapper
 * @CreateTime: 2020-10-10 16:39
 * @Description: TODO
 */
public interface UserMapper {

    List<User> findUsers();

    int addUser(User user);
}
```

3.7.2映射文件Mapper.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.sm.mapper.UserMapper">

    <select id="findUsers" resultType="User">
        select * from t_user where delState=0
    </select>

    <insert id="addUser">
        insert  into t_user
        values (null,#{userLoginName},#{userPhone},#
{userAge},#{userPwd},#{userName},1,now(),0)
    </insert>
</mapper>
```

3.7.3 Mapper接口方式的开发整合（Spring+MyBatis）

在MyBatis+Spring项目中，Mapper层接口可以不手动编写实现类，而是在配置文件中配置bean标签，使用动态代理的方式来创建Mapper层接口的实现类：

为了使用Mapper接口方式开发（整合Spring与MyBatis），在配置文件中有两种配置方式：

1. 基于MapperFactoryBean的整合
   MapperFactoryBean是MyBatis-Spring团队提供的一个用于根据Mapper层接口生成Mapper层接口的实现类的实例对象，该类在Spring配置文件中使用时可以配置如下参数：

   a. mapperInterface：用于指定接口
   b. sqlSessionFactory：用于指定配置文件中构建SqlSessionFactory的bean标签的id值
   c. sqlSessionTemplate：用于指定SqlSessionTemplate，如果这个参数和sqlSessionFactory参数一起使用，则只会启用sqlSessionTemplate

   示例：在Spring的配置文件中创建一个Mapper层接口实现类的Bean的代码如下

```
<bean id="customerMapper"
class="org.mybatis.spring.mapper.MapperFactoryBea
n">
        <property name="mapperInterface"
value="com.itheima.mapper.CustomerMapper" />
        <property name="sqlSessionFactory"
ref="sqlSessionFactory" />
</bean>
```

2. 基于MapperScannerConfigurer的整合（本笔记中使用这种方式）

   这种方式是第一种方式的改进版。
   在实际的项目中，Mapper层会包含很多接口，如果每一个接口都在Spring配置文件中配置，不但会增加工作量，还会使得Spring配置文件非常臃肿。为此，可以采用自动扫描的形式来配置MyBatis中的映射器——采用MapperScannerConfigurer类。该类在Spring配置文件中可以配置以下属性：

   a. basePackage：指定Mapper层接口文件所在的包路径，当需要扫描多个包时可以使用分号或逗号作为分隔符。指定包路径后，会扫描该包及其子包中的所有文件。
   b. annotationClass：指定了要扫描的注解名称，只有被注解标识的类才会被配置为映射器。
   c. sqlSessionFactoryBeanName：指定在Spring中定义的SqlSessionFactory的Bean名称。
   d. sqlSessionTemplateBeanName：指定在Spring中定义的SqlSessionTemplate的Bean名称。如果定义此属性，则sqlSessionFactoryBeanName将不起作用。
   e. markerInterface：指定创建映射器的接口。

   示例：

```
<!-- Mapper代理开发（基于MapperScannerConfigurer） -
->
<bean
class="org.mybatis.spring.mapper.MapperScannerCon
figurer">
     <property name="basePackage"
value="com.itheima.mapper" />
</bean>
```

通常情况下，MapperScannerConfigurer在使用时只需通过basePackage属性指定需要扫描的包即可，Spring会自动的通过包中的Mapper层的接口来生成映射器。这使得开发人员可以在编写很少代码的情况下，完成对映射器的配置，从而提高开发效率。

上面说的这两种方式都要符合《MyBatis基础应用.md》中4.2节中的Mapper接口编程方式规范

3.8.新建业务逻辑层

```
package com.sm.service;

import com.sm.bean.User;

import java.util.List;

/**
 * @BelongsProject: Spring-2021
 * @BelongsPackage: com.sm.service
 * @CreateTime: 2021-01-14 14:37
 * @Description: TODO
 */
public interface UserService {

    //测试全查
    List<User> findUsers();

    //测试新增
    int addUser(User user);
}
```

业务逻辑层的实现层

```
package com.sm.service.impl;

import com.sm.bean.User;
import com.sm.mapper.UserMapper;
import com.sm.service.UserService;
```

```
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

/**
 * @BelongsProject: Spring-2021
 * @BelongsPackage: com.sm.service.impl
 * @CreateTime: 2021-01-14 14:38
 * @Description: TODO
 */
@Service
public class UserServiceImpl implements UserService {

    //这个字段的值是来自Spring容器中，在applicationContext.xml
中配置了最后一行内容后，Spring框架会自动通过代理模式（反射）生成
Mapper接口的对象，并将其存放再Spring容器中
    @Autowired
    UserMapper um;

    public List<User> findUsers() {
        return um.findUsers();
    }

    public int addUser(User user) {
        return um.addUser(user);
    }
}
```

3.9.新测试类启动测试

```
package com.sm.test;

import com.sm.bean.User;
import com.sm.service.UserService;
import org.junit.Test;
import org.junit.runner.RunWith;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.test.context.ContextConfiguration;
import
org.springframework.test.context.junit4.SpringJUnit4ClassR
unner;

import java.util.List;

/**
```

```
 * @BelongsProject: MyBatis-Spring
 * @BelongsPackage: com.sm.test
 * @CreateTime: 2020-10-10 17:07
 * @Description: TODO
 */
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class TestSpring {

    @Autowired
    UserService us;

    @Test
    public void testFindUsers(){
        List<User> users = us.findUsers();
        for (User user : users) {
            System.out.println(user);
        }
    }

    @Test
    public void testAddUser(){
        //(String userLoginName, String userPhone, Integer
userAge, String userPwd, String userName)
        User user=new
User("admin","1338989090",18,"admin123","小白");
        int count = us.addUser(user);
        System.out.println(count>0?"新增成功":"新增失败");
    }
}
```