SSM 集成案例

叩丁狼教育：任小龙

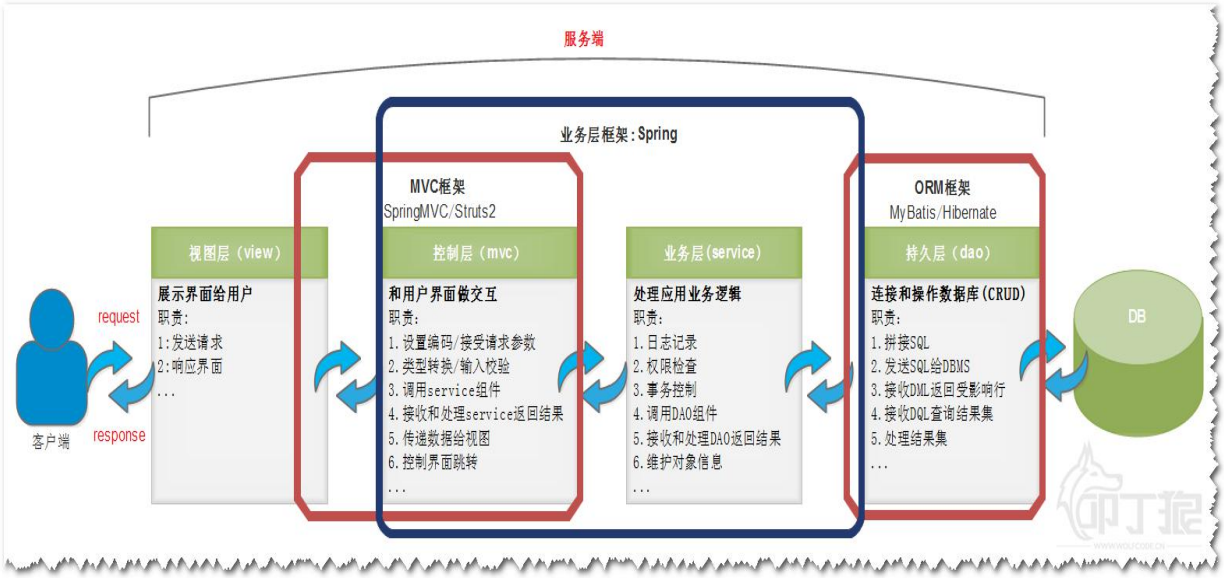
春水初生,春林初盛,春风十里,不如你——冯唐

1. Java EE 三层架构

1.1. 三层架构

在生活中，饭店中员工根据工作岗位的不同会划分为很多个职位，比如服务员、厨师、采购员、收营员等等。我们可以想象一下如果在工作中岗位不清晰，一个员工一会扮演厨师、一会扮演服务员，此时工作不仅混乱且效率比较低。所以我们能发现职责分离非常重要。

类比饭店中的职位划分，在 Web 开发中的最佳实践就是根据每一层的功能职责的不同，划分为控制层、业务层、持久层。



- 控制层 ：web/mvc: 负责处理与界面交互的相关操作（Struts2/Spring MVC）
- 业务层 ：service: 负责复杂的业务逻辑计算和判断（Spring）
- 持久层 ：dao/mapper: 负责将业务逻辑数据存储到数据库（Hibernate/MyBatis）

在这里，大家耳熟能详的 SSH（Struts1/2+Spring+Hibernate），SSM（SpringMVC+Spring+MyBatis）框架组合就出现了。

注意：无论是 SSH 还是 SSM，第二个 S 绝对是 Spring，详细为什么。SSH 的组合已经不再流行了，现在企业喜欢采用 SSM 组合。

2. SSM 环境准备

本案例是采用 MyBatis 3.45 版本和 Spring 5.02 版本做整合集成操作，在整合中不会再具体到每一个知识点的原理和具体讲解，忘记的童鞋需要回头去看看之前讲解的内容。当前我们的目的就是把三个框架整合起来，并保证整合之后能成功运行。

本案例也不会考虑页面的美丑问题，除了 CRUD 操作并没有包含其他业务功能。

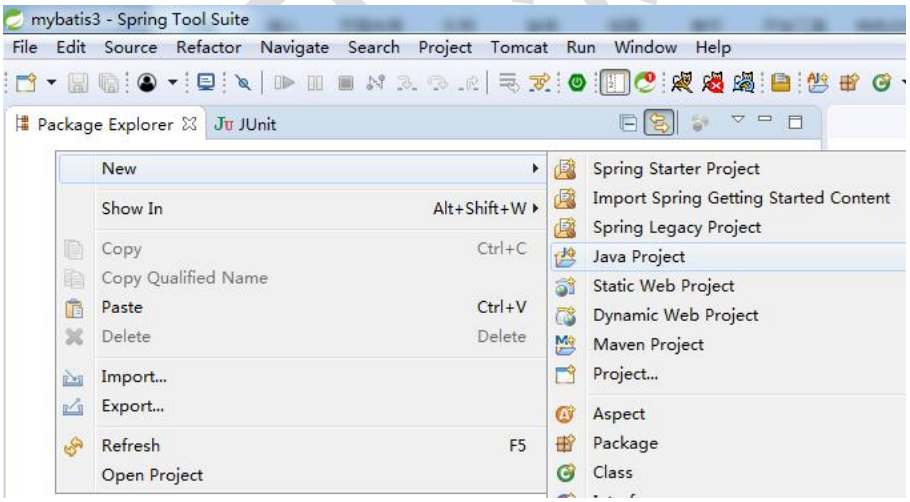
如果整合不成功的童鞋，或者在整合过程中有奇葩的问题，可以加我微信号 **iwiller** 询问，QQ 用的比较少了，希望大家能理解，我会尽我所能去帮助大家。

考虑到有同学会使用 Maven 工具，有的不会，所以我采用两种方式来搭建项目，可自己选择。

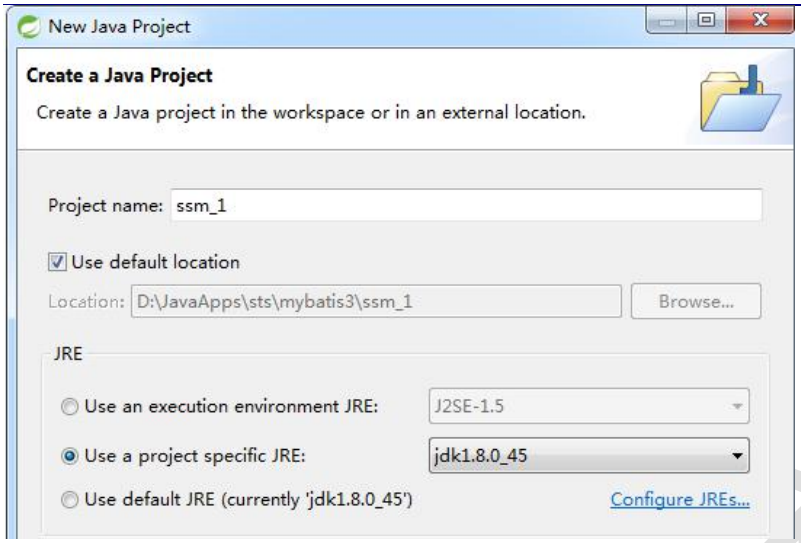
2.1. 传统方式搭建项目

2.1.1. 创建项目

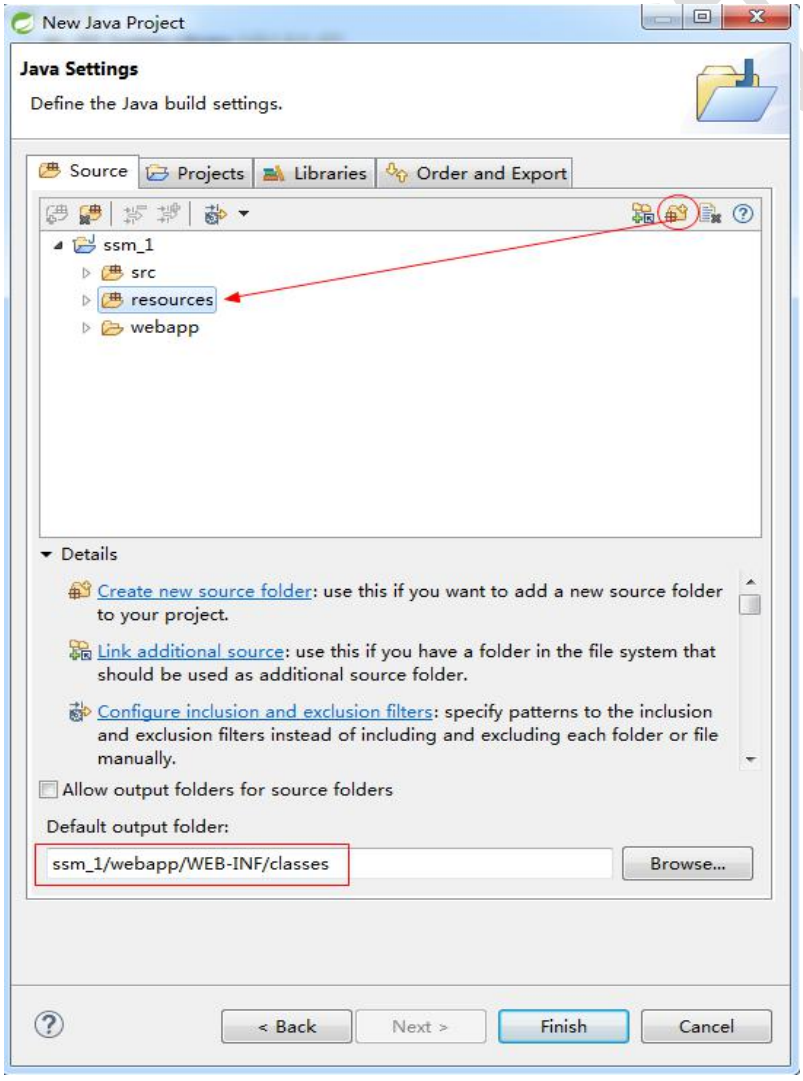
步骤 1：创建一个 Java 项目，如下图



步骤 2：设置项目名称 ssm_1（名字可随意），使用 JDK1.8



步骤 3：先添加一个 source folder 目录——resources，专门用来存放配置文件；再把项目的字节码输出目录设置到项目目录下的 **webapp/WEB-INF/classes** 目录中，如下图。



步骤 4：在 WEB-INF 目录下创建 lib 目录和 web.xml 文件，如下图。

该结构是 Web 项目的标准，不能改动其中：

- src: 存放 Java 文件
- resources: 存放配置文件
- webapp： 存放 Web 开发的资源
- lib： 存放依赖的 jar
- web.xml：存放 Web 项目基础配置

web.xml 文件的内容：

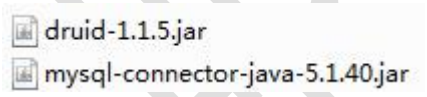
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
   http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">

</web-app>
```

注意：确保使用的是 servlet3.0 规范的约束。

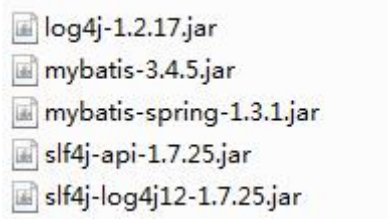
2.1.2.拷贝依赖 jar

数据库相关：



mysql-connector-java-5.1.40.jar	：MySQL 数据库驱动
druid-1.1.5.jar	：阿里巴巴德鲁伊数据库连接池

MyBatis 相关：



log4j-1.2.17.jar	：log4j 日志实现

slf4j-api-1.7.25.jar ：slf4j 日志标准

slf4j-log4j12-1.7.25.jar ：slf4j 和 log4j 两种日志规范的整合桥梁和适配包

mybatis-3.4.5.jar ：MyBatis 核心包

mybatis-spring-1.3.1.jar ：MyBatis 和 Spring 整合的桥梁包（插件包）

Spring 相关：

com.springsource.org.aopalliance-1.0.0.jar ：AOP 联盟规范

com.springsource.org.apache.commons.logging-1.1.1.jar：commons 日志规范

com.springsource.org.aspectj.weaver-1.6.8.RELEASE.jar：AOP 思想 AspectJ 织入组件

spring-aop-5.0.2.RELEASE.jar ：Spring 对 AOP 的集成支持

spring-aspects-5.0.2.RELEASE.jar ：Spring 对 AspectJ 的集成支持

spring-beans-5.0.2.RELEASE.jar ：Spring 管理 bean 核心组件

spring-context-5.0.2.RELEASE.jar ：Spring 容器核心组件

spring-core-5.0.2.RELEASE.jar ：Spring 核心工具组件

spring-expression-5.0.2.RELEASE.jar ：Spring 表达式组件

spring-jdbc-5.0.2.RELEASE.jar ：Spring 对 JDBC 的支持

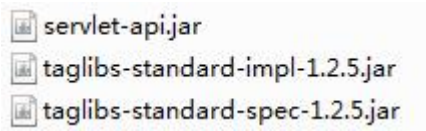
spring-tx-5.0.2.RELEASE.jar ：Spring 对事务的集成支持

spring-test-5.0.2.RELEASE.jar ：Spring 对 JUnit 的集成支持

spring-web-5.0.2.RELEASE.jar ：Spring 对 Web 的集成支持

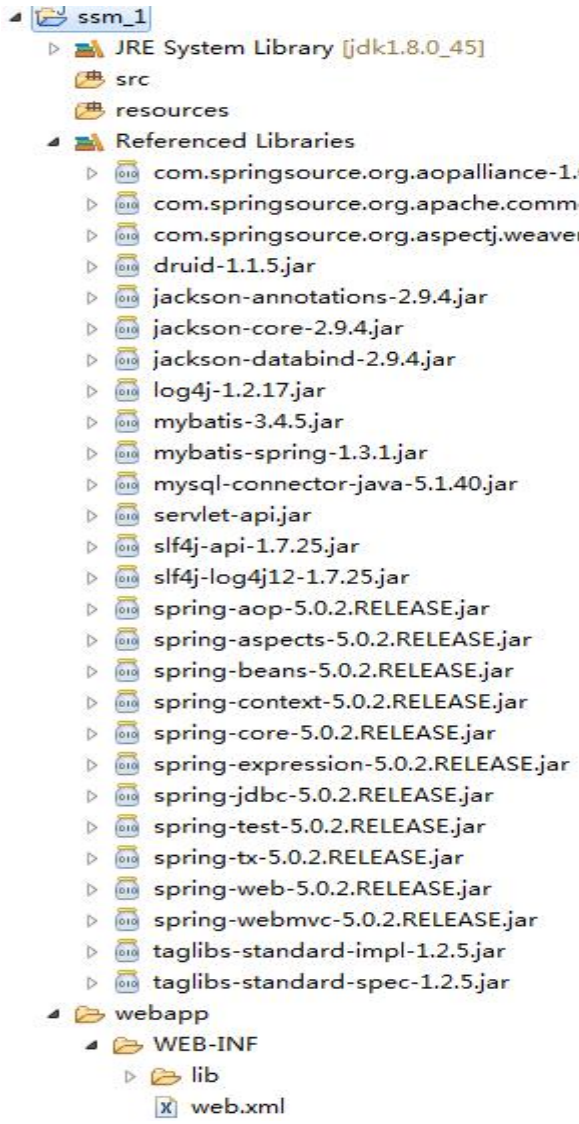
spring-webmvc-5.0.2.RELEASE.jar	：Spring 的 MVC 核心组件

Servlet 相关：

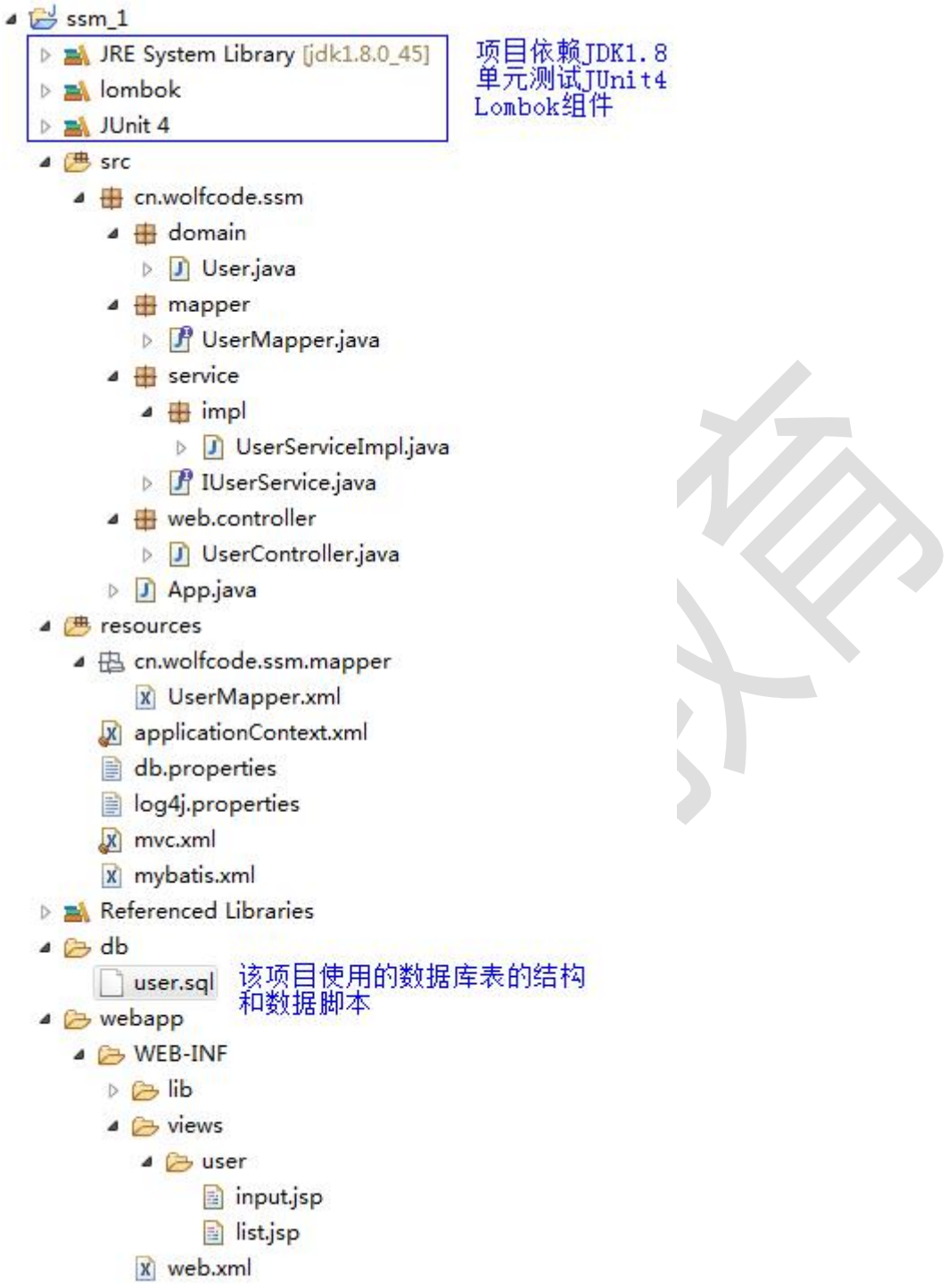


servlet-api.jar	：Web 开发对 Servlet API 的支持
taglibs-standard-spec-1.2.5.jar	：Web 开发对 JSTL 的支持规范接口
taglibs-standard-impl-1.2.5.jar	：Web 开发对 JSTL 的支持规范实现

选中 lib 目录所有的 jar，鼠标右键【Build Path】 再点击【Add to Build Path】，操作后效果如下图。



最终项目的结构和组成如下图：

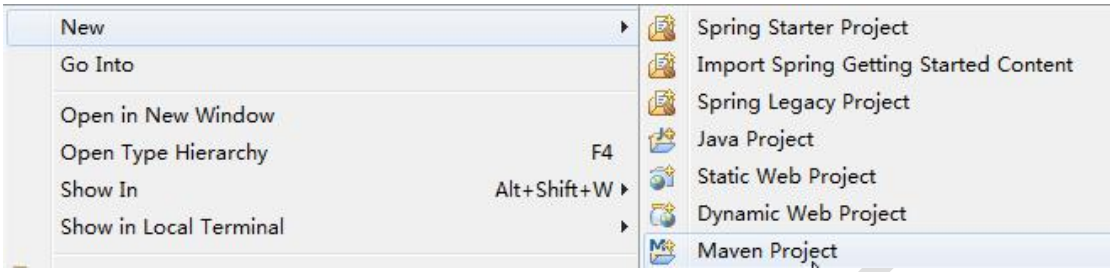


2.2. Maven 方式搭建项目

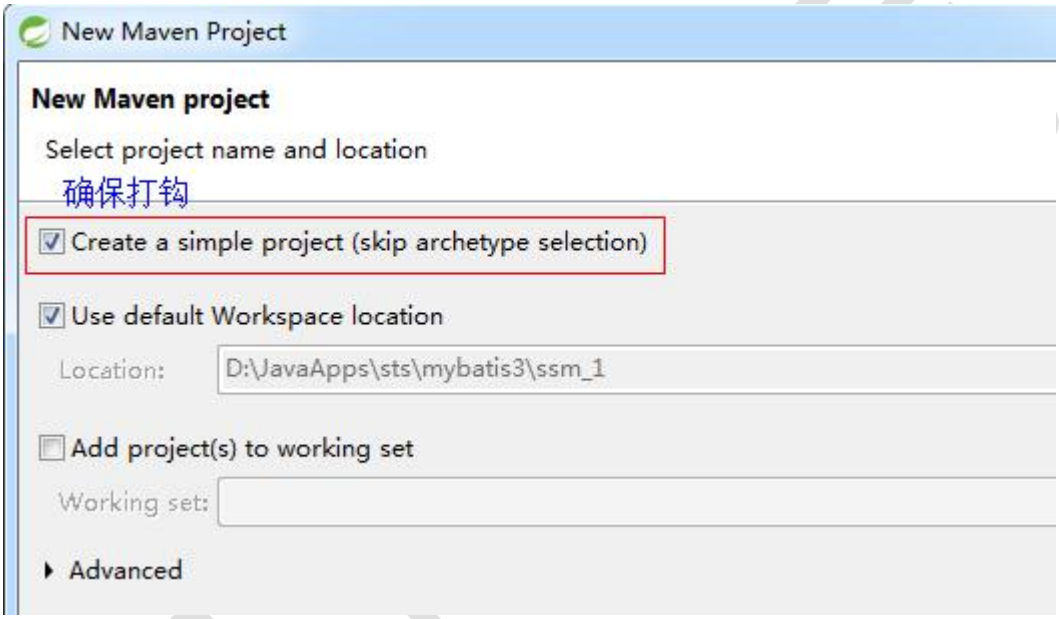
Maven 是企业开发使用非常频繁的项目构建工具，至少有一个好处——不用再去忧虑要拷贝哪些 jar。这里不对 Maven 做过多解释，直接上手使用。

2.2.1. 创建项目

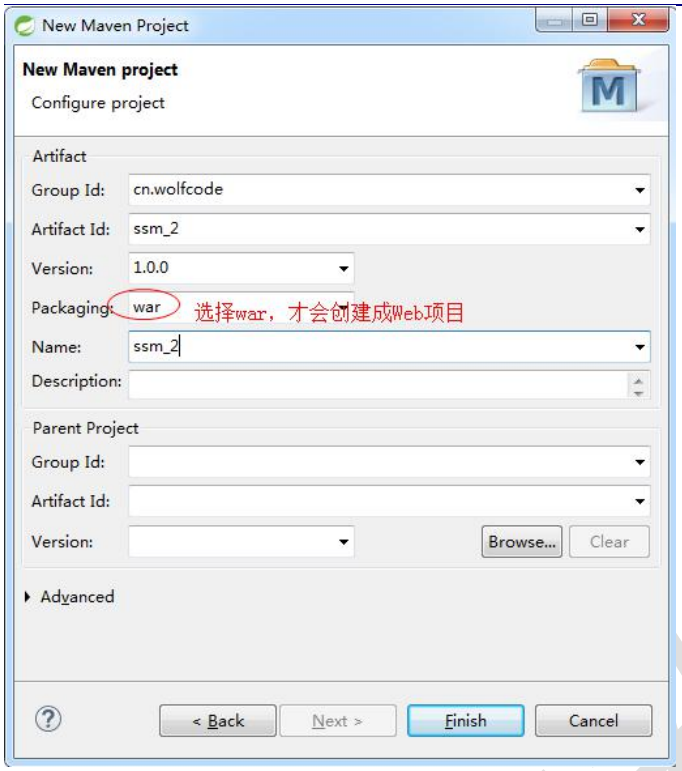
步骤 1：创建一个 Maven 项目，如下图



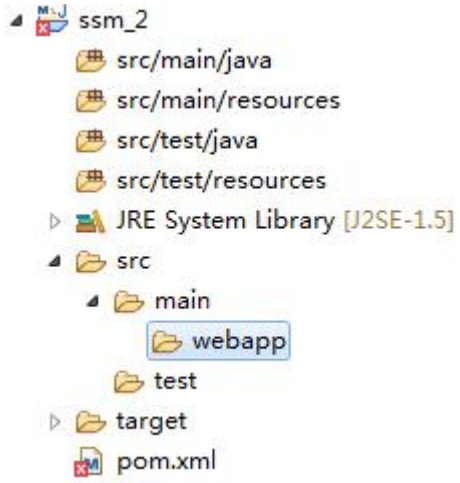
步骤 2：使用跳过骨架选择的方式创建 Maven 项目，如下图



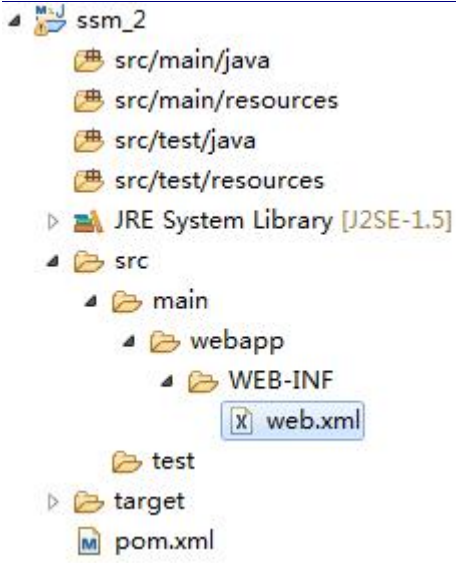
步骤 3：创建 Web 项目，如下图



步骤 4：创建出 Web 项目如下图，此时有保存，也需要我们去做一些调整



此时报错的原因是 Web 项目没有 web.xml 文件，所以我们需要在 webapp 目录下创建 WEB-INF 目录，再拷贝一个 web.xml 文件。



如此，使用 Maven 创建 Web 项目结构就成功了，此时项目还没有依赖的 jar，我们就得去操作 pom 文件了。

2.2.2. pom 文件（重点）

在 Maven 中所有的依赖和插件都是通过配置 pom.xml 文件来完成的，pom 文件的配置如下。

pom.xml 文件配置如下（可以直接拷贝到各自项目中）

```
    <project                          xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>cn.wolfcode</groupId>
        <artifactId>ssm_2</artifactId>
        <version>1.0.0</version>
        <packaging>war</packaging>
        <name>ssm_2</name>


        <!-- 项目编码和依赖框架版本 -->
        <properties>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
            <spring.version>5.0.2.RELEASE</spring.version>
            <mybatis.version>3.4.5</mybatis.version>
        </properties>
```

```
    <dependencies>

        <!-- Spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
            <scope>test</scope>
        </dependency>
        <!-- mybatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>${mybatis.version}</version>
        </dependency>
        <!-- mybatis-spring -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
            <version>1.3.0</version>
        </dependency>

        <!-- MySQL 驱动 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.38</version>
            <scope>runtime</scope>
```

```
        </dependency>
        <!-- druid 连接池 -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.0.26</version>
        </dependency>

        <!-- aspectJ 织入 -->
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.8.7</version>
        </dependency>

        <!-- JSON 处理 -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.9.5</version>
        </dependency>
        <!-- JUnit4 测试工具 -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>

        <!-- Servlet-API -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.0.1</version>
            <scope>provided</scope>
        </dependency>


        <!-- JSTL 标签库 -->
        <dependency>
            <groupId>jstl</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
```

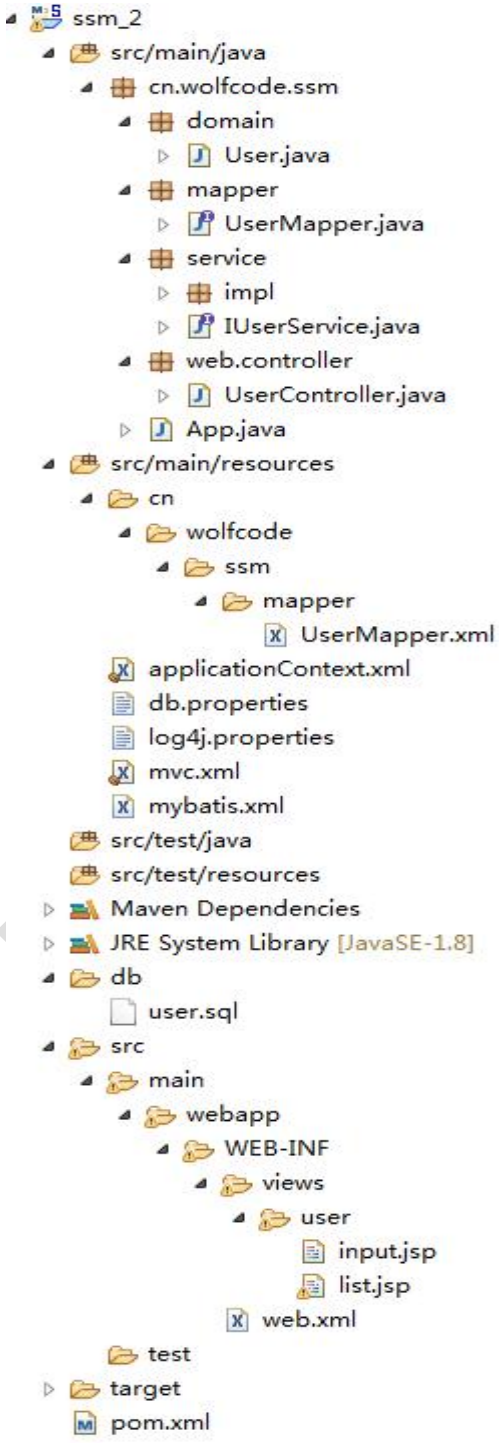
```
        </dependency>
        <!-- 日志 -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.7.21</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.7.21</version>
        </dependency>

        <!-- lombok 插件 -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.16.6</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <!--Java 编译器插件 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
            <!--Tomcat7 插件 -->
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <version>2.2</version>
                <configuration>
                    <uriEncoding>UTF-8</uriEncoding>
                    <path>/</path>
                    <port>80</port>
```

```
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

再选择项目按 Alt + F5，此时项目就由原来的 JDK1.5 变成了 JDK1.8。

最终 Maven 项目的结构和组成如下图：



2.3. 项目结构和组成

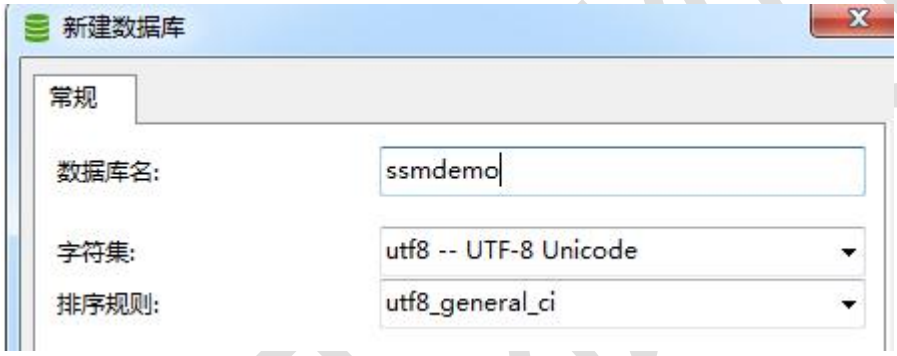
SSM 框架的集成方式有很多种方式，基于 XML 配置的，基于 XML+注解配置的，基于 Java Config 方式。

- 基于全 XML 配置： 古老方式，开发和配置麻烦，配置太多，企业不待见。
- 基于 XML+注解配置：通用配置使用 XML 配置，个别配置使用注解，企业使用比较多。
- 基于 Java Config 方式：没有配置文件，企业新贵 SpringBoot 底层采用的方式。

在这里，我们选用第二种方式来完成。

2.3.1.数据库相关

创建 ssmdemo 数据库，注意使用 utf8 编码。



创建 t_user 表。

```
CREATE TABLE `user` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) DEFAULT NULL,
  `age` tinyint(4) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

插入数据

```
INSERT INTO `user` (`name`, `age`) VALUES ('乔峰', 32)
INSERT INTO `user` (`name`, `age`) VALUES (' 陆小凤', 27)
```

db.properties 文件（连接数据库）

```
#连接数据库的四个基本要素
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/ssmdemo
jdbc.username=root
jdbc.password=admin
#druid 连接池其他配置,省略
```

log4j.properties 文件（打印执行 MyBatis 执行操作时的 SQL 信息）

```
# Global logging configuration
log4j.rootLogger=ERROR, stdout
# MyBatis logging configuration...
log4j.logger.cn.wolfcode.ssm.mapper=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
```

mybatis.xml 文件（MyBatis 配置文件，主要配置延迟加载）

```
<?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>
   <settings>
       <!--配置允许懒加载-->
       <setting name="lazyLoadingEnabled" value="true"/>
       <!--取消关联查询积极性-->
       <setting name="aggressiveLazyLoading" value="false"/>
       <!--哪些方法触发关系查询-->
       <setting name="lazyLoadTriggerMethods" value="clone"/>
   </settings>
</configuration>
```

- applicationContext.xml 文件　　Spring 核心配置文件
- mvc.xml 文件　　SpringMVC 配置文件
- web.xml 文件　　Web 配置文件

三个文件是集成的重点，具体配置见后文

2.3.2.编写 domain 和 Mapper 组件

domain 组件：不会使用 lombok 的参照之前资料。

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Accessors(chain = true)
public class User {
    private Long id;        //唯一标识
    private String name;    //名字
    private Integer age;    //年龄
```

```
    }
```

mapper 组件：如果使用 MBG 生成，则参照之前资料

```
    public interface UserMapper {
        /**
         * 保存操作
         * @param u
         * @return 返回受影响的行数
         */
        int insert(User u);

        /**
         * 根据指定 id,更新操作
         * @param u
         * @return 返回受影响的行数
         */
        int updateById(User u);

        /**
         * 删除操作
         * @param id
         * @return  返回受影响的行数
         */
        int deleteById(Long id);

        /**
         * 根据指定 id,查询单个对象操作
         * @param id
         * @return  如果存在就返回该对象,否则返回 null
         */
        User selectById(Long id);

        /**
         * 查询所有对象操作
         * @return
         */
        List<User> selectAll();
    }
```

UserMapper 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="cn.wolfcode.ssm.mapper.UserMapper">
        <!-- 结果集映射处理-->
```

```
    <resultMap id="BaseResultMap" type="cn.wolfcode.ssm.domain.User">
        <id column="id" property="id" />
        <result column="name" property="name" />
        <result column="age" property="age" />
    </resultMap>
    <!-- 保存操作 -->
    <insert id="insert" useGeneratedKeys="true" keyProperty="id">
        INSERT INTO user (id, name, age) VALUES (null, #{name}, #{age})
    </insert>
    <!-- 更新操作 -->
    <update id="updateById">
        UPDATE user SET name = #{name}, age = #{age} WHERE id = #{id}
    </update>
    <!-- 删除操作 -->
    <delete id="deleteById">
        DELETE FROM user WHERE id = #{id}
    </delete>
    <!-- 查询单个操作 -->
    <select id="selectById" resultMap="BaseResultMap">
        SELECT id, name, age FROM user WHERE id = #{id}
    </select>
    <!-- 查询多个操作 -->
    <select id="selectAll" resultMap="BaseResultMap">
        SELECT id, name, age FROM user
    </select>
</mapper>
```

2.3.3.编写 service 组件

IUserService 接口（因只是 CRUD 操作，比较简单故省略注释）：

```
public interface IUserService {
    void save(User u);

    void update(User u);

    void delete(Long id);

    User get(Long id);

    List<User> listAll();
}
```

UserServiceImpl 实现类：

```
@Service
public class UserServiceImpl implements IUserService {

    @Autowired
    private UserMapper userMapper;

    public void save(User u) {
        userMapper.insert(u);
    }

    public void update(User u) {
        userMapper.updateById(u);
    }

    public void delete(Long id) {
        userMapper.deleteById(id);
    }

    public User get(Long id) {
        return userMapper.selectById(id);
    }

    public List<User> listAll() {
        return userMapper.selectAll();
    }
}
```

2.3.4. 测试类

编写一个测试类，用于测试 Spring 和 MyBatis 是否整合成功，至于什么时候测试，后面再提及。

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class App {

    @Autowired
    private IUserService userService;

    @Test
    public void testListAll() throws Exception {
        userService.listAll().forEach(System.out::println);
```

```
    }

    @Test
    public void testSave() throws Exception {
        userService.save(new User().setName("will").setAge(17));
    }
}
```

2.4. Controller 类

该类包含了基本的 CRUD 方法。

```
@Controller
@RequestMapping("/user")
public class UserController {
    @Autowired
    private IUserService userService;

    //列表查询
    @RequestMapping("/list")
    public String list(Model model) {
        model.addAttribute("users", userService.listAll());
        return "/user/list";
    }

    //进入编辑界面
    @RequestMapping("/input")
    public String input(Long id, Model model) {
        if (id != null) {
            model.addAttribute("user", userService.get(id));
        }
        return "/user/input";
    }

    //保存或更新操作
    @RequestMapping("/saveOrUpdate")
    public String saveOrUpdate(User user) {
        if (user.getId() == null) {
            userService.save(user);
        } else {
            userService.update(user);
        }
```

```
            return "redirect:/user/list";
        }

        //删除操作
        @RequestMapping("/delete")
        public String delete(Long id) {
            if (id != null) {
                userService.delete(id);
            }
            return "redirect:/user/list";
        }
    }
```

2.5. JSP 页面

2.5.1.列表界面

list.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<head>
<title>用户列表</title>
</head>
<body>

    <a href="/user/input">添加</a>
    <table border="1" width="400">
        <thead>
            <tr>
                <th>ID</th>
                <th>NAME</th>
                <th>AGE</th>
                <th>操作</th>
            </tr>
        </thead>
        <tbody>
            <c:forEach items="${users}" var="u">
                <tr align="center">
                    <td>${u.id}</td>
```

```
                    <td>${u.name}</td>
                    <td>${u.age}</td>
                    <td>
                        <a href="/user/delete?id=${u.id}">删除</a>
                        <a href="/user/input?id=${u.id}">编辑</a>
                    </td>
                </tr>
            </c:forEach>
        </tbody>
    </table>
</body>
</html>
```

2.5.2.编辑界面

input.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java"%>
<html>
<head>
<title>编辑界面</title>
</head>
<body>
    <form action="/user/saveOrUpdate" method="post">
        <input type="hidden" name="id" value="${user.id}">
        名称: <input type="text" name="name" value="${user.name}"><br/>
        年龄: <input type="number" name="age" value="${user.age}"><br/>
         <input type="submit" value="保存">
    </form>
</body>
</html>
```

3. Spring 集成 MyBatis

在 application.xml 文件中，配置如下：

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

    <!-- IoC 注解解析器 -->
    <context:component-scan base-package="cn.wolfcode.ssm"/>

    <!-- DI 注解解析器 -->
    <context:annotation-config/>

    <!-- 加载数据库配置信息 -->
    <context:property-placeholder location="classpath:db.properties"
                system-properties-mode="NEVER"/>
    <!-- 连接池对象 -->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
         init-method="init" destroy-method="close">
        <property name="driverClassName" value="${jdbc.driverClassName}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>
    <!-- 配置 SessionFactory -->
    <bean id="sessionFactory"
          class="org.mybatis.spring.SqlSessionFactoryBean">
        <!-- 1:连接池 -->
        <property name="dataSource" ref="dataSource"/>
        <!-- 2:读取 MyBatis 总配置文件 -->
        <property name="configLocation" value="classpath:mybatis.xml"/>
        <!-- 3:配置别名扫描 -->
        <property name="typeAliasesPackage"
                  value="cn.wolfcode.ssm.domain"/>
        <!-- 4:加载 mapper 文件 -->
        <property name="mapperLocations"
                  value="classpath:cn/wolfcode/ssm/mapper/*Mapper.xml"/>
    </bean>

    <!--Mapper 接口代理扫描器-->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="cn.wolfcode.ssm.mapper"/>
    </bean>
```

```
</beans>
```

接下来对上述配置，做一个简单分析。

3.1. 注解解析器配置

IoC 注解解析器 ，用于去扫描哪些包及其子包中使用 Service，Controller，Component、Repository 注解标注的类，Spring 会自动去管理这些 bean 的生命周期。

```
<context:component-scan base-package="cn.wolfcode.ssm"/>d
```

DI 注解解析器：用于去解析 Autowired 和 Resource 注解，从 Spring3.0 开始可以不配置，建议配置上。

```
<context:annotation-config/>
```

3.2. 连接池相关配置

加载 db.properties 文件，并创建连接池对象

```
<!-- 加载数据库配置信息 -->
  <context:property-placeholder location="classpath:db.properties"
              system-properties-mode="NEVER"/>
  <!-- 连接池对象 -->
  <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
       init-method="init" destroy-method="close">
     <property name="driverClassName" value="${jdbc.driverClassName}"/>
     <property name="username" value="${jdbc.username}"/>
     <property name="url" value="${jdbc.url}"/>
     <property name="password" value="${jdbc.password}"/>
  </bean>
```

3.3. SqlSessionFactory 配置

创建 SqlSessionFactory 对象，这里使用到了之前在 Spring 中讲解的创建 bean 的第四种方式，实现 FactoryBean 接口的方式。

创建 SqlSessionFactory 对象需要注入的信息如下配置。

```
<!-- 配置 SessionFactory -->
  <bean id="sessionFactory"
        class="org.mybatis.spring.SqlSessionFactoryBean">
```

```
        <!-- 1:连接池 -->
        <property name="dataSource" ref="dataSource"/>
        <!-- 2:读取 MyBatis 总配置文件 -->
        <property name="configLocation" value="classpath:mybatis.xml"/>
        <!-- 3:配置别名扫描 -->
        <property name="typeAliasesPackage"
                  value="cn.wolfcode.ssm.domain"/>
        <!-- 4:加载 mapper 文件 -->
        <property name="mapperLocations"
                  value="classpath:cn/wolfcode/ssm/mapper/*Mapper.xml"/>
    </bean>
```

注意：在 Spring 中加载任何配置文件，都尽量使用 classpath 前缀，表示从 classpath 路径开始去寻找。

3.4. MapperScannerConfigurer 配置

配置 MapperScannerConfigurer 的目的是告诉 Spring 去哪一个包中寻找接口，并为该接口创建代理对象，这正好验证了之前讲解的 Mapper 接口的实现类对象是通过动态代理创建出来的。

```
<!--Mapper 接口代理扫描器-->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="cn.wolfcode.ssm.mapper"/>
</bean>
```

此时运行测试类中的查询方法 testListAll，能测试通过，说明 Spring 和 MyBatis 初步整合成功。

3.5. Spring 集成事务

接上一节，运行测试类中的保存方法 testSave，也应该能测试通过，user 表中多了一条数据。

此时我们修改 UserServiceImpl 类中的 save 方法的源代码。

修改之前：

```
public void save(User u) {
    userMapper.insert(u);
}
```

修改之后：

```
public void save(User u) {
    userMapper.insert(u);
    //故意抛出 RuntimeException,测试事务回滚
    int a = 1 / 0;
}
```

此时 ,大家会发现 service 类的 save 方法中故意抛出了一个运行异常 ,再运行 testSave 方法，测试也通过，在 user 表中也会多一条数据。这就不合理了，因为如果存在异常则需要回顾该业务方法。出现该问题的原因是，我们还没有配置 MyBatis 的事务。

正如之前在 Spring 中讲解的一样，配置事务需要配置事务管理器，事务增强，切入点语法。

MyBatis 和 JDBC 一样使用相同的事务管理器——DataSourceTransactionManager。

```
<!-- 配置事务 -->
<bean id="txManager"
    class=
"org.springframework.jdbc.datasource.DataSourceTransactionManager">
       <property name="dataSource" ref="dataSource"/>
</bean>

<tx:advice id="txAdvice" transaction-manager="txManager">
   <tx:attributes>
      <tx:method name="get*" read-only="true"/>
      <tx:method name="list*" read-only="true"/>
      <tx:method name="query*" read-only="true"/>
      <tx:method name="*" propagation="REQUIRED"/>
   </tx:attributes>
 </tx:advice>
 <aop:config>
   <aop:pointcut id="txPoint"
expression="execution(* cn.wolfcode.ssm.service.*Service.*(..))"/>
   <aop:advisor advice-ref="txAdvice" pointcut-ref="txPoint"/>
 </aop:config>

 <!--Mapper 接口代理扫描器-->
 <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
   <property name="basePackage" value="cn.wolfcode.ssm.mapper"/>
 </bean>
```

当事务加上之后，再运行 testSave 方法，此时发现 user 表中不会再多出一条数据，如果删除掉 int a = 1/0；这样代码，再运行测试，user 表中多出一条数据。如果是此效果，则说明事务配置成功。

到此，Spring 和 MyBatis 整合也就结束了，同时也表示后台整合成功。

注：对事务不了解的同学，看看 Spring 讲解中关于事务的知识点。

4. Spring 集成 SpringMVC

Spring 和 SpringMVC 本身也就是一家的，他们之前本就是无缝的，严格意义上说不存在集成，只不过是拷贝几个 jar 包，写上几行配置，在这里我们就姑且保留集成或整合一词吧。

4.1. Spring MVC 配置

我们在 mvc.xml 文件中完成 SpringMVC 的配置，包括事务解析器，拦截器等这些。

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

    <!-- 包含 Spring 的核心配置 -->
    <import resource="classpath:applicationContext.xml" />
    <!--MVC 的注解解析器 -->
    <mvc:annotation-driven />
    <!-- 静态资源处理器 -->
    <mvc:default-servlet-handler />
    <!-- JSP 视图解析器 -->
    <bean class="
org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

4.2. web.xml 配置

在 web.xml 中配置前端控制器和编码过滤器。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
   http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">

   <!-- 针对 POST 请求设置编码过滤器 -->
    <filter>
        <filter-name>encodingFilter</filter-name>
        <filter-class>
             org.springframework.web.filter.CharacterEncodingFilter
        </filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <init-param>
            <param-name>forceRequestEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>encodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- SpringMVC 前端控制器 -->
    <servlet>
        <servlet-name>SpringMVC</servlet-name>
        <servlet-class>
              org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:mvc.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
```

```
        <servlet-mapping>
            <servlet-name>SpringMVC</servlet-name>
            <url-pattern>/</url-pattern>
        </servlet-mapping>
    </web-app>
```

4.3. 部署访问

部署该 Web 项目，再访问 http://localhost/user/list，此时会在浏览器看到列表界面，则表明前台整合成功，同时也表示 SSM 整合完毕。

添加

ID	NAME	AGE	操作
1	乔峰	32	删除 编辑
2	陆小凤	27	删除 编辑
4	will	17	删除 编辑

其他的 CRUD 方法和代码，直接看前面的内容即可。

写在最后的话：

到此 SSM 整合就完毕了，其实并不难。

框架集成之后，我们的重心应该是在此基础之上做开发，做一些业务功能。

记住：集成就一次，业务才是王道。

如果在集成 SSM 时遇到解决不了的问题，希望同学们在群里多多交流，或者可以直接加我微信（微信号：iwiller）询问我。