

# Spring、SpringMVC、Mybatis 框架整合

## 准备工作导入依赖

导入 spring mybatis mvc 等依赖，同时过滤 xml文件和 properties 文件

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>org.example</groupId>
8     <artifactId>2022-6-29</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11 <dependencies>
12
13     <!-- lombok 插件 -->
14     <dependency>
15         <groupId>org.projectlombok</groupId>
16         <artifactId>lombok-maven-plugin</artifactId>
17         <version>1.18.12.0</version>
18         <scope>provided</scope>
19     </dependency>
20
21
22     <!-- spring 核心依赖包 -->
23     <dependency>
24         <groupId>org.springframework</groupId>
25         <artifactId>spring-webmvc</artifactId>
26         <version>5.3.18</version>
27     </dependency>
28
29     <!-- 测试依赖-->
30     <dependency>
31         <groupId>junit</groupId>
32         <artifactId>junit</artifactId>
33         <version>4.12</version>
34         <scope>test</scope>
35     </dependency>
36
37     <!-- mysql8.0 驱动包 -->
38     <dependency>
39         <groupId>mysql</groupId>
40         <artifactId>mysql-connector-java</artifactId>
41         <version>8.0.16</version>
```

```
42         </dependency>
43
44     <!-- aop 织入包 -->
45     <dependency>
46         <groupId>org.aspectj</groupId>
47         <artifactId>aspectjweaver</artifactId>
48         <version>1.9.9.1</version>
49     </dependency>
50
51
52     <!-- mybatis依赖 -->
53     <dependency>
54         <groupId>org.mybatis</groupId>
55         <artifactId>mybatis</artifactId>
56         <version>3.5.6</version>
57     </dependency>
58
59     <!-- spring-mybatis 依赖 -->
60     <dependency>
61         <groupId>org.mybatis</groupId>
62         <artifactId>mybatis-spring</artifactId>
63         <version>2.0.7</version>
64     </dependency>
65
66     <!-- srping-jdbc 提供 spring 的数据源dataSource -->
67     <dependency>
68         <groupId>org.springframework</groupId>
69         <artifactId>spring-jdbc</artifactId>
70         <version>5.3.21</version>
71     </dependency>
72
73     <!-- 事务相关的依赖包 -->
74     <dependency>
75         <groupId>org.springframework</groupId>
76         <artifactId>spring-tx</artifactId>
77         <version>5.3.21</version>
78     </dependency>
79
80     <!-- spring-mvc 的底层有servlet实现，必须导入 -->
81     <dependency>
82         <groupId>javax.servlet</groupId>
83         <artifactId>servlet-api</artifactId>
84         <version>2.5</version>
85     </dependency>
86
87
88     <!--下面都是 mvc 有返回json数据的要求的话，必须导入的json相关依赖包 -->
89     <dependency>
90         <groupId>com.fasterxml.jackson.core</groupId>
91         <artifactId>jackson-databind</artifactId>
92         <version>2.9.3</version>
93     </dependency>
94
95     <dependency>
96         <groupId>com.fasterxml.jackson.core</groupId>
97         <artifactId>jackson-core</artifactId>
98         <version>2.9.3</version>
99     </dependency>
```

```

100
101     <dependency>
102         <groupId>com.fasterxml.jackson.core</groupId>
103         <artifactId>jackson-annotations</artifactId>
104         <version>2.9.3</version>
105     </dependency>
106
107
108 </dependencies>
109
110
111     <!-- 在运行的时候使得 xml文件和 properties文件得以留下 -->
112
113     <build>
114         <resources>
115             <resource>
116                 <directory>src/main/resources</directory>
117                 <includes>
118                     <include>**/*.xml</include>
119                     <include>**/*.properties</include>
120                 </includes>
121                 <filtering>true</filtering>
122             </resource>
123
124             <resource>
125                 <directory>src/main/java</directory>
126                 <includes>
127                     <include>**/*.xml</include>
128                     <include>**/*.properties</include>
129                 </includes>
130                 <filtering>true</filtering>
131             </resource>
132         </resources>
133     </build>
134
135 </project>

```

## 总体配置文件架构



建表语句,测试 mybatis 是否成功连接

```
1 create database if not exists ssm;
2
3 use ssm;
4
5 drop table if exists users;
6
7 create table users(
8     id int primary key auto_increment,
9     username varchar(20),
10    password varchar(20)
11 );
12
```

总的applicationContext.xml 文件

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:context="http://www.springframework.org/schema/context"
4       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5       xmlns:aop="http://www.springframework.org/schema/aop"
6       xmlns:cache="http://www.springframework.org/schema/cache"
7       xsi:schemaLocation="http://www.springframework.org/schema/beans
8         http://www.springframework.org/schema/beans/spring-beans.xsd
9         http://www.springframework.org/schema/context
10        http://www.springframework.org/schema/context/spring-context.xsd
11        http://www.springframework.org/schema/aop
12        http://www.springframework.org/schema/aop/spring-aop.xsd
13        http://www.springframework.org/schema/cache
14        http://www.springframework.org/schema/cache/spring-cache.xsd">
15
16    <!-- 导入spring-dao.xml配置文件-->
17    <import resource="classpath:spring-dao.xml"/>
18
19    <!-- 导入spring-mvc 配置文件-->
20    <!-- 导入spring-service 配置文件-->
21
22    <!-- 自动扫描包路径下的注解-->
23    <context:component-scan base-package="com.*"/>
24
25    <!-- 开启aop支持-->
26    <aop:aspectj-autoproxy/>
27
28 </beans>
```

# spring整合 mybatis

db.properties

```
1 jdbc.driverClassName=com.mysql.cj.jdbc.Driver
2 jdbc.url=jdbc:mysql://127.0.0.1:3306/ssm?
  useUnicode=true&characterEncoding=utf-8&serverTimezone=GMT
3 jdbc.username=root
4 jdbc.password=123456
```

mybatis-config.xml, 这个文件可以完全省略, 但是还是留下方便做一些其他的配置

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3     PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6
7
8     <!--把包路径下的类 的全限定名 都换成别名 类名首字母小写 -->
9
10    <typeAliases>
11        <package name="com.bit.pojo"/>
12    </typeAliases>
13
14 </configuration>
```

spring-dao.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:context="http://www.springframework.org/schema/context"
4       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5       xmlns:aop="http://www.springframework.org/schema/aop"
6       xsi:schemaLocation="http://www.springframework.org/schema/beans
7           http://www.springframework.org/schema/beans/spring-beans.xsd
8           http://www.springframework.org/schema/context
9           https://www.springframework.org/schema/context/spring-context.xsd
10          http://www.springframework.org/schema/aop
11          https://www.springframework.org/schema/aop/spring-aop.xsd">
12
13
14 <!-- 引入外部的 properties 文件-->
```

```

15     <context:property-placeholder location="classpath:db.properties" />
16
17     <!-- 拿到spring的数据源，以后可以是c3p0、德鲁伊等数据源-->
18     <bean
19 class="org.springframework.jdbc.datasource.DriverManagerDataSource"
20 id="dataSource">
21     <property name="driverClassName" value="${jdbc.driverClassName}"/>
22     <property name="username" value="${jdbc.username}"/>
23     <property name="url" value="${jdbc.url}"/>
24     <property name="password" value="${jdbc.password}"/>
25 </bean>
26
27     <!-- 拿到sqlSessionFactory-->
28     <bean class="org.mybatis.spring.SqlSessionFactoryBean"
29 id="sqlSessionFactory">
30     <property name="dataSource" ref="dataSource"/>
31     <property name="configLocation" value="classpath:mybatis-
32 config.xml"/>
33     <property name="mapperLocations"
34 value="classpath:com/bit/mapper/UserMapper.xml"/>
35 </bean>
36
37     <!-- 在配置dao接口扫描包，动态实现了dao接口注入到 spring容器中-->
38     <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
39     <!-- 注入sqlSessionFactory-->
40     <property name="sqlSessionFactoryBeanName"
41 value="sqlSessionFactory"/>
42     <property name="basePackage" value="com.bit.mapper"/>
43
44     <!-- 经过上面两步，相当于给接口加入了SqlSession，也相当于个接口创建一个实现类然
45 后getMapper再去使用mybatis-->
46 </bean>
47
48 </beans>

```

建一个pojo包，创建一个实体类

```

1 package com.bit.pojo;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 @Data
8 @AllArgsConstructor
9 @NoArgsConstructor
10 public class User {
11     private int id;
12     private String username;
13     private String password;
14 }
15

```

建一个mapper 包, 创建一个mapper接口以及 mapper.xml 文件

## UserMapper 接口

```
1 package com.bit.mapper;
2
3 import com.bit.pojo.User;
4 import org.apache.ibatis.annotations.Param;
5 import java.util.List;
6
7 public interface UserMapper {
8
9     int insert(User user);
10
11     int delete(@Param("id") int id);
12
13     int update(@Param("id") int id, User user);
14
15     List<User> selectAll();
16
17     User selectById(int id);
18
19 }
```

## UserMapper.xml 文件

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6 <mapper namespace="com.bit.mapper.UserMapper">
7     <insert id="insert" parameterType="user">
8         insert into users values(#{id},#{username},#{password})
9     </insert>
10
11     <delete id="delete">
12         delete from users where id=#{id}
13     </delete>
14
15     <update id="update" parameterType="user">
16         update users set username=#{username},password=#{password} where
17         id=#{id}
18     </update>
19
20     <select id="selectAll" resultType="user" >
21         select * from users;
```

```

21     </select>
22
23     <select id="selectById" resultType="user" >
24         select * from users where id=#{id};
25     </select>
26
27 </mapper>

```

在service 层创建接口，对dao层进行封装

创建一个service包，写一个UserService接口 和 UserServiceImpl 实现类

UserService 接口

```

1  package com.bit.service;
2
3  import com.bit.pojo.User;
4
5  import java.util.List;
6
7  public interface UserService {
8
9      int addUser(User user);
10
11     int delteteUser(int id);
12
13     int updateUser(int id,User user);
14
15     List<User> selectAllUser();
16
17     User selectOne(int id);
18
19 }
20

```

ServiceImpl 实现类

```

1  package com.bit.service;
2
3  import com.bit.mapper.UserMapper;
4  import com.bit.pojo.User;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.stereotype.Service;
7
8  import java.util.List;
9
10 @Service

```

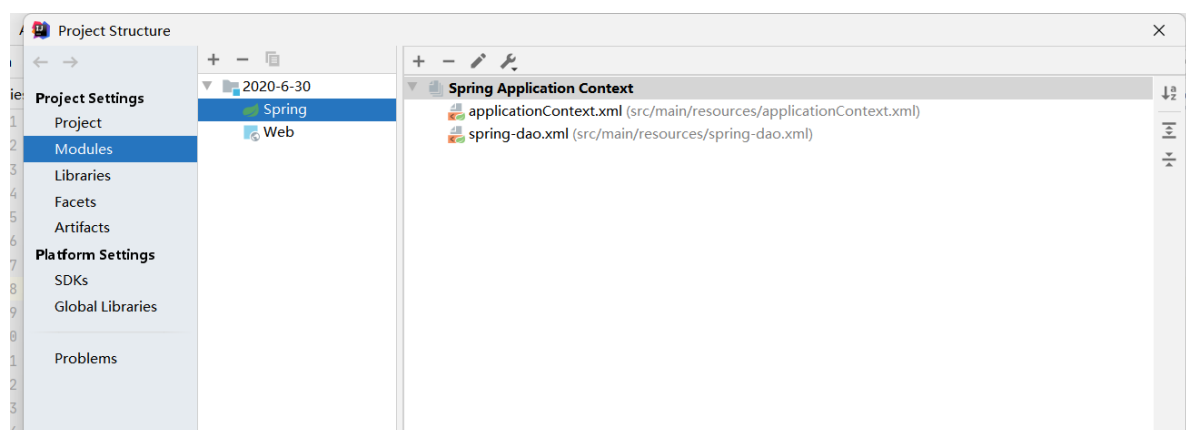


```

11 public class UserServiceImpl implements UserService{
12
13     @Autowired
14     private UserMapper userMapper;
15
16     public int addUser(User user) {
17         return userMapper.insert(user);
18     }
19
20     public int delteteUser(int id) {
21         return userMapper.delete(id);
22     }
23
24     public int updateUser(int id, User user) {
25         return userMapper.update(id,user);
26     }
27
28     public List<User> selectAllUser() {
29         return userMapper.selectAll();
30     }
31
32     public User selectOne(int id) {
33         return userMapper.selectById(id);
34     }
35
36     public static void main(String[] args) {
37
38     }
39 }
40

```

测试是否整合mybatis成功的时候一定要看一下，是否spring的配置文件都加载上了



测试是否成功整合代码

```

1 import com.bit.pojo.User;

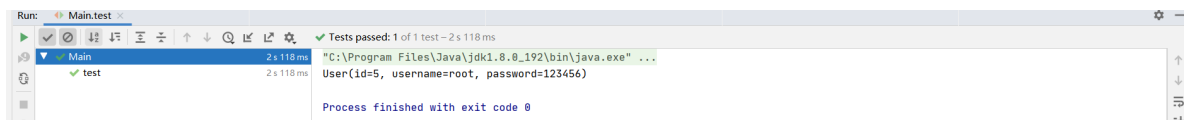
```

```

2  import com.bit.service.UserService;
3  import org.junit.Test;
4  import org.springframework.context.ApplicationContext;
5  import org.springframework.context.support.ClassPathXmlApplicationContext;
6
7  public class Main {
8
9      @Test
10     public void test(){
11         ApplicationContext context = new
12         ClassPathXmlApplicationContext("applicationContext.xml");
13         UserService userService = context.getBean("userServiceImpl",
14         UserService.class);
15         User user = userService.selectOne(5);
16         System.out.println(user);
17     }
18 }

```

测试成功



## Spring 整合 SpringMVC

web.xml, 配置 DispatcherServlet , 同时加载application.xml 总的spring 配置文件

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5                               http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
6           version="4.0">
7
8     <servlet>
9         <servlet-name>springmvc</servlet-name>
10        <servlet-
11        class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
12        <init-param>
13            <param-name>contextConfigLocation</param-name>
14            <param-value>classpath:applicationContext.xml</param-value>
15        </init-param>
16        <load-on-startup>1</load-on-startup>
17
18    </servlet>

```

```

18     <servlet-mapping>
19         <servlet-name>springmvc</servlet-name>
20         <url-pattern>/</url-pattern>
21     </servlet-mapping>
22
23 </web-app>

```

## spring-mvc.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:context="http://www.springframework.org/schema/context"
5      xmlns:mvc="http://www.springframework.org/schema/mvc"
6      xsi:schemaLocation="http://www.springframework.org/schema/beans
7          http://www.springframework.org/schema/beans/spring-beans.xsd
8          http://www.springframework.org/schema/context
9          https://www.springframework.org/schema/context/spring-context.xsd
10         http://www.springframework.org/schema/mvc
11         http://www.springframework.org/schema/mvc/spring-mvc.xsd">
12
13  <!-- 所有的css、js、html 文件全部在 webapp下创建static进行保存，同时在下面用
14  resource引入-->
15
16      <!--加载静态资源location表示访问的路径return"/static/login.html",mapping表示
17      映射的静态资源位置-->
18      <!--      <mvc:resources location="/static/css/" mapping="/static/css/**"/>-->
19      >
20      <!--      <mvc:resources location="/static/js/" mapping="/static/js/**"/>-->
21      <!--      <mvc:resources location="/static/" mapping="/static/**"/>-->
22
23      <!--      视图解析器： 如果返回数据（JSON、HTML），不需要经过该步骤-->
24      <!--      如果返回的是视图的话，那么原来的视图经过 之前的数据进行渲染之后 返回给
25      Dispatcher，在展示给前端-->
26      <!--      <bean
27      class="org.springframework.web.servlet.view.InternalResourceViewResolver"
28      id="internalResourceViewResolver">-->
29      <!--          <property name="prefix" value="/WEB-INF/jsp/">-->
30      <!--          <property name="suffix" value=".jsp"/>-->
31      <!--      </bean>-->
32
33      <!--      开启注解扫描，将使用注解的类托管到spring 容器中-->
34      <context:component-scan base-package="com.bit.controller"/>
35
36      <!--      每次请求过来，先经过 DefaultServletHttpRequestHandler 判断是否是静态文
37      件，如果是静态文件，则进行处理，不是则放行交由 DispatcherServlet 控制器处理-->
38      <mvc:default-servlet-handler/>
39
40      <!--      开启mvc注解驱动-->
41      <mvc:annotation-driven/>
42
43  </beans>

```

spring-service

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:context="http://www.springframework.org/schema/context"
4       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5       xmlns:aop="http://www.springframework.org/schema/aop"
6       xmlns:tx="http://www.springframework.org/schema/tx"
7       xsi:schemaLocation="http://www.springframework.org/schema/beans
8                           http://www.springframework.org/schema/beans/spring-beans.xsd
9                           http://www.springframework.org/schema/context
10                          https://www.springframework.org/schema/context/spring-context.xsd
11                          http://www.springframework.org/schema/aop
12                          https://www.springframework.org/schema/aop/spring-aop.xsd
13                          http://www.springframework.org/schema/tx
14                          https://www.springframework.org/schema/tx/spring-tx.xsd">
15
16
17
18     <!-- 开启注解扫描-->
19     <context:component-scan base-package="com.bit.service"/>
20
21     <!-- 3、声明式事务-->
22     <bean
23         class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
24         id="transactionManager">
25         <property name="dataSource" ref="dataSource"/>
26     </bean>
27
28     <!-- 4、aop支持-->
29     <aop:aspectj-autoproxy/>
30
31 </beans>
```

在总的applicationContext 导入 spring-mvc.xml 、 spring-service.xml 文件

```
1 <!-- 导入spring-mvc 配置文件-->
2 <import resource="classpath:spring-mvc.xml"/>
3 <!-- 导入spring-service 配置文件-->
4 <import resource="classpath:spring-service.xml"/>
```

测试mvc框架是否整合完成的代码

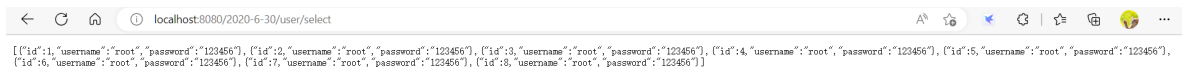
创建controller包, 创建UserController类

```

1  package com.bit.controller;
2
3  import com.bit.service.UserService;
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.stereotype.Controller;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.ResponseBody;
8
9  @Controller
10 @RequestMapping("/user")
11 public class UserController {
12
13     @Autowired
14     private UserService userService;
15
16     @RequestMapping("/select")
17     @ResponseBody
18     public Object select(){
19         return userService.selectAllUser();
20     }
21
22 }
23

```

运行项目，访问接口出现JSON格式的数据，整合mvc 成功



The screenshot shows a web browser window with the address bar displaying "localhost:8080/2020-6-30/user/select". The page content shows a JSON array of user objects, each containing "id", "username", and "password" fields. The data is as follows:

```

[[{"id":1,"username":"root","password":"123456"}, {"id":2,"username":"root","password":"123456"}, {"id":3,"username":"root","password":"123456"}, {"id":4,"username":"root","password":"123456"}, {"id":5,"username":"root","password":"123456"}, {"id":6,"username":"root","password":"123456"}, {"id":7,"username":"root","password":"123456"}, {"id":8,"username":"root","password":"123456"}]]

```

最后所有的整合三个框架的工作完成。

如果有静态资源文件（css、html、js），那么放在web目录下的 创建的static 文件夹下面。