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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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>org.example</groupId>
    <artifactId>2022-6-29</artifactId>
    <version>1.0-SNAPSHOT</version>

<dependencies>

    <!--lombok  插件 -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok-maven-plugin</artifactId>
        <version>1.18.12.0</version>
        <scope>provided</scope>
    </dependency>


        <!-- spring 核心依赖包 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.3.18</version>
        </dependency>

<!--     测试依赖-->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>

        <!-- mysql8.0 驱动包 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.16</version>
```

```
        </dependency>

    <!-- aop 织入包-->
    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>1.9.9.1</version>
    </dependency>


    <!-- mybatis依赖 -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.5.6</version>
    </dependency>

    <!-- spring-mybatis 依赖 -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis-spring</artifactId>
        <version>2.0.7</version>
    </dependency>

    <!-- srping-jdbc 提供 spring 的数据源dataSource -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>5.3.21</version>
    </dependency>

    <!-- 事务相关的依赖包 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>5.3.21</version>
    </dependency>

       <!-- spring-mvc 的底层有servlet实现，必须导入  -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.5</version>
    </dependency>


    <!--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 -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.9.3</version>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.9.3</version>
    </dependency>
```

```
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-annotations</artifactId>
        <version>2.9.3</version>
    </dependency>


</dependencies>


      <!-- 在运行的时候使得 xml文件和 properties文件得以留下 -->

    <build>
        <resources>
            <resource>
                <directory>src/main/resources</directory>
                <includes>
                    <include>**/*.xml</include>
                    <include>**/*.properties</include>
                </includes>
                <filtering>true</filtering>
            </resource>

            <resource>
                <directory>src/main/java</directory>
                <includes>
                    <include>**/*.xml</include>
                    <include>**/*.properties</include>
                </includes>
                <filtering>true</filtering>
            </resource>
        </resources>
    </build>

</project>
```

总体配置文件架构

建表语句,测试 mybatis 是否成功连接

```sql
create database if not exists ssm;

use ssm;

drop table if exists users;

create table users(
      id int primary key auto_increment,
      username varchar(20),
      password varchar(20)
);

```

总的applicationContext.xml 文件

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

<!--导入spring-dao.xml配置文件-->
<import resource="classpath:spring-dao.xml"/>
<!--    导入spring-mvc 配置文件-->
<!--    导入spring-service 配置文件-->


<!--    自动扫描包路径下的注解-->
    <context:component-scan base-package="com.*"/>

<!--    开启aop支持-->

    <aop:aspectj-autoproxy/>

</beans>
```

spring整合 mybatis

db.properties

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

```
<?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>


     <!--把包路径下的类 的全限定名 都换成别名 类名首字母小写  -->

    <typeAliases>
        <package name="com.bit.pojo"/>
    </typeAliases>

</configuration>
```

spring-dao.xml

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


<!--    引入外部的 properties 文件-->
```

```
    <context:property-placeholder location="classpath:db.properties" />

    <!--    拿到spring的数据源，以后可以是c3p0、德鲁伊等数据源-->
    <bean class="org.springframework.jdbc.datasource.DriverManagerDataSource" id="dataSource">
        <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>

    <!--    拿到sqlSessionFactory-->
    <bean class="org.mybatis.spring.SqlSessionFactoryBean" id="sqlSessionFactory">
        <property name="dataSource" ref="dataSource"/>
        <property name="configLocation" value="classpath:mybatis-config.xml"/>
        <property name="mapperLocations" value="classpath:com/bit/mapper/UserMapper.xml"/>
    </bean>

<!--     在配置dao接口扫描包，动态实现了dao接口注入到 spring容器中-->
   <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
<!--       注入sqlSessionFactory-->
       <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
       <property name="basePackage" value="com.bit.mapper"/>

<!--       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后getMapper再去使用mybatis-->
   </bean>


</beans>
```

建一个pojo包，创建一个实体类

```
package com.bit.pojo;

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

@Data
@AllArgsConstructor
@NoArgsConstructor
public class User {
    private int id;
    private String username;
    private String password;
}

```

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

UserMapper 接口

```java
package com.bit.mapper;

import com.bit.pojo.User;
import org.apache.ibatis.annotations.Param;
import java.util.List;

public interface UserMapper {

    int insert(User user);

    int delete(@Param("id") int id);

    int update(@Param("id") int id,User user);

    List<User>  selectAll();

    User selectById(int id);

}
```

UserMapper.xml 文件

```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.bit.mapper.UserMapper">
     <insert id="insert" parameterType="user">
         insert into users values(#{id},#{username},#{password})
     </insert>

    <delete id="delete">
        delete from users where id=#{id}
    </delete>

    <update id="update" parameterType="user">
        update  users set username=#{username},password=#{password} where id=#{id}
    </update>

    <select id="selectAll" resultType="user" >
        select * from users;
```

```
    </select>

    <select id="selectById" resultType="user" >
        select * from users where id=#{id};
    </select>

</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

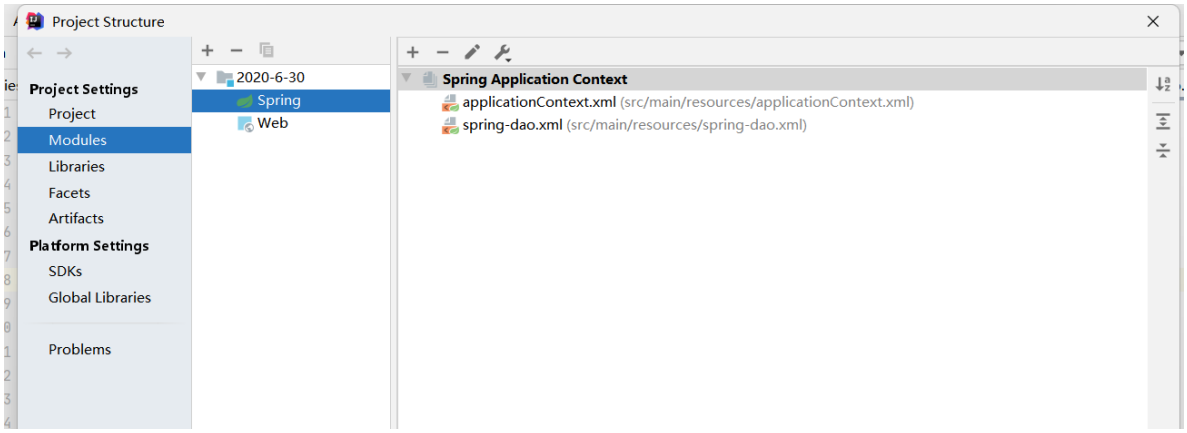
```
package com.bit.service;

import com.bit.pojo.User;

import java.util.List;

public interface UserService {

    int  addUser(User user);

    int  delteteUser(int id);

    int  updateUser(int id,User user);

    List<User> selectAllUser();

    User selectOne(int id);

}

```

UserServiceImpl 实现类

```
package com.bit.service;

import com.bit.mapper.UserMapper;
import com.bit.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
```

```java
public class UserServiceImpl implements UserService{

    @Autowired
    private UserMapper userMapper;

    public int addUser(User user) {
        return userMapper.insert(user);
    }

    public int delteteUser(int id) {
        return userMapper.delete(id);
    }

    public int updateUser(int id, User user) {
        return userMapper.update(id,user);
    }

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

    public User selectOne(int id) {
        return userMapper.selectById(id);
    }

    public static void main(String[] args) {

    }
}

```

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

```java
import com.bit.pojo.User;
```

```java
import com.bit.service.UserService;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

    @Test
    public void test(){
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
        UserService userService = context.getBean("userServiceImpl", UserService.class);
        User user = userService.selectOne(5);
        System.out.println(user);
    }
}
```

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <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:applicationContext.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>
```

spring-mvc.xml

```
<?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/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">

<!--    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resource引入-->

    <!--加载静态资源location表示访问的路径return"/static/login.html",mapping表示映射的静态资源位置-->
<!--    <mvc:resources location="/static/css/" mapping="/static/css/**"/>-->
<!--    <mvc:resources location="/static/js/" mapping="/static/js/**"/>-->
<!--    <mvc:resources location="/static/" mapping="/static/**"/>-->


    <!--    视图解析器:  如果返回数据（JSON、HTML），不需要经过该步骤-->
    <!--    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Dispatcher，在展示给前端-->
<!--    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver" id="internalResourceViewResolver">-->
<!--        <property name="prefix" value="/WEB-INF/jsp/"/>-->
<!--        <property name="suffix" value=".jsp"/>-->
<!--    </bean>-->

    <!--    开启注解扫描，将使用注解的类托管到spring 容器中-->
    <context:component-scan base-package="com.bit.controller"/>

   <!--    每次请求过来，先经过 DefaultServletHttpRequestHandler 判断是否是静态文件，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-->
    <mvc:default-servlet-handler/>

    <!--    开启mvc注解驱动-->
    <mvc:annotation-driven/>

</beans>
```

spring-service

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



    <!--    开启注解扫描-->
       <context:component-scan base-package="com.bit.service"/>

<!--    3、声明式事务-->
    <bean class="org.springframework.jdbc.datasource.DataSourceTransactionManager" id="transactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

<!--    4、aop支持-->
     <aop:aspectj-autoproxy/>


</beans>
```

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

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

```java
package com.bit.controller;

import com.bit.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequestMapping("/user")
public class UserController {

    @Autowired
    private UserService userService;

    @RequestMapping("/select")
    @ResponseBody
    public Object select(){
        return userService.selectAllUser();
    }

}

```

运行项目，访问接口出现JSON格式的数据，整合mvc 成功

[{"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。