**Spring+SpringMVC+Mybatis整合**

**转载**<https://blog.csdn.net/lp20171401131/article/details/106579908>

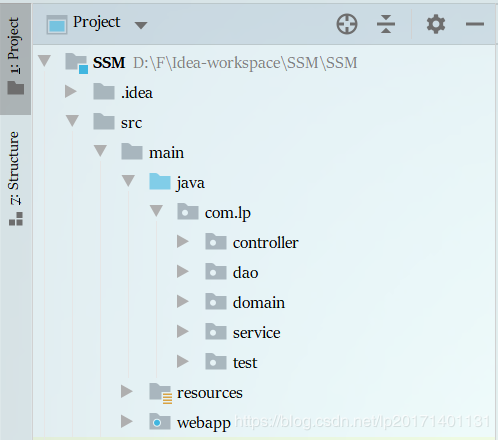
首先，我们要明确一点，是通过Spring来整合SpringMVC，Spring来整合MyBatis

# **1、我们先来做一下前期准备**

所有需要的依赖我们先导入

|  |
| --- |
| <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>1.8</maven.compiler.source>  <maven.compiler.target>1.8</maven.compiler.target>  <spring.version>5.2.6.RELEASE</spring.version>  <mysql.version>5.1.49</mysql.version>  </properties>  <dependencies>  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>1.9.5</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  <version>${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>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-tx</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.11</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>${mysql.version}</version>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5</version>  </dependency>  <dependency>  <groupId>javax.servlet.jsp</groupId>  <artifactId>jsp-api</artifactId>  <version>2.1</version>  </dependency>  <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  <version>1.2</version>  </dependency>  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  <version>1.7.30</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>1.7.12</version>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.5.4</version>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>2.0.2</version>  </dependency>  <dependency>  <groupId>com.mchange</groupId>  <artifactId>c3p0</artifactId>  <version>0.9.5.4</version>  </dependency>  </dependencies>、 |

然后我们下一步将所需要的目录结构创建好



然后在数据库中创建一个简单的Account，拥有id主键自增，name，money，OK！  
此时在domain里面创建一个实习类Account.java

|  |
| --- |
| package com.lp.domain;  import java.io.Serializable;  /\*\*  \* @Date 2020/6/5 22:11  \* @Author luopeng  \*/  public class Account implements Serializable {  private Integer id;  private String name;  private Double money;  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public Double getMoney() {  return money;  }  public void setMoney(Double money) {  this.money = money;  }  @Override  public String toString() {  return "Account{" +  "id=" + id +  ", name='" + name + '\'' +  ", money=" + money +  '}';  }  } |

然后是dao接口AccountDao.java

|  |
| --- |
| package com.lp.dao;  import com.lp.domain.Account;  import org.springframework.stereotype.Repository;  import java.util.List;  /\*\*  \* @Date 2020/6/5 22:13  \* @Author luopeng  \*/  @Repository  public interface AccountDao {  List<Account> findAll();  void saveAccount(Account account);  } |

再继续创建service层接口的AccountService.java

|  |
| --- |
| package com.lp.service;  import com.lp.domain.Account;  import java.util.List;  /\*\*  \* @Date 2020/6/5 22:15  \* @Author luopeng  \*/  public interface AccountService {  List<Account> findAll();  void saveAccount(Account account);  } |

service层实现类

|  |
| --- |
| package com.lp.service.impl;  import com.lp.domain.Account;  import com.lp.service.AccountService;  import org.springframework.stereotype.Service;  import java.util.List;  /\*\*  \* @Date 2020/6/5 22:16  \* @Author luopeng  \*/  @Service("accountService")  public class AccountServiceImpl implements AccountService {  @Override  public List<Account> findAll() {  System.out.println("业务层查询所有...");  return null;  }  @Override  public void saveAccount(Account account) {  System.out.println("业务层保存账户...");  }  } |

**2、以上基本的类都创建好了，接下来我们先搭建一下Spring的环境**

对于Spring，是专注于业务层的所以我们只需要他注解扫描业务层和dao层的类

配置文件application.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:aop="http://www.springframework.org/schema/aop"  xmlns:context="http://www.springframework.org/schema/context"  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/aop  http://www.springframework.org/schema/aop/spring-aop.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx.xsd">  <!--开启自动扫描 -->  <context:component-scan base-package="com.lp">  <!-- 配置哪些不扫描-->  <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller"/>  </context:component-scan>  </beans> |

resources目录下创建一下log4j.properties日志配置文件

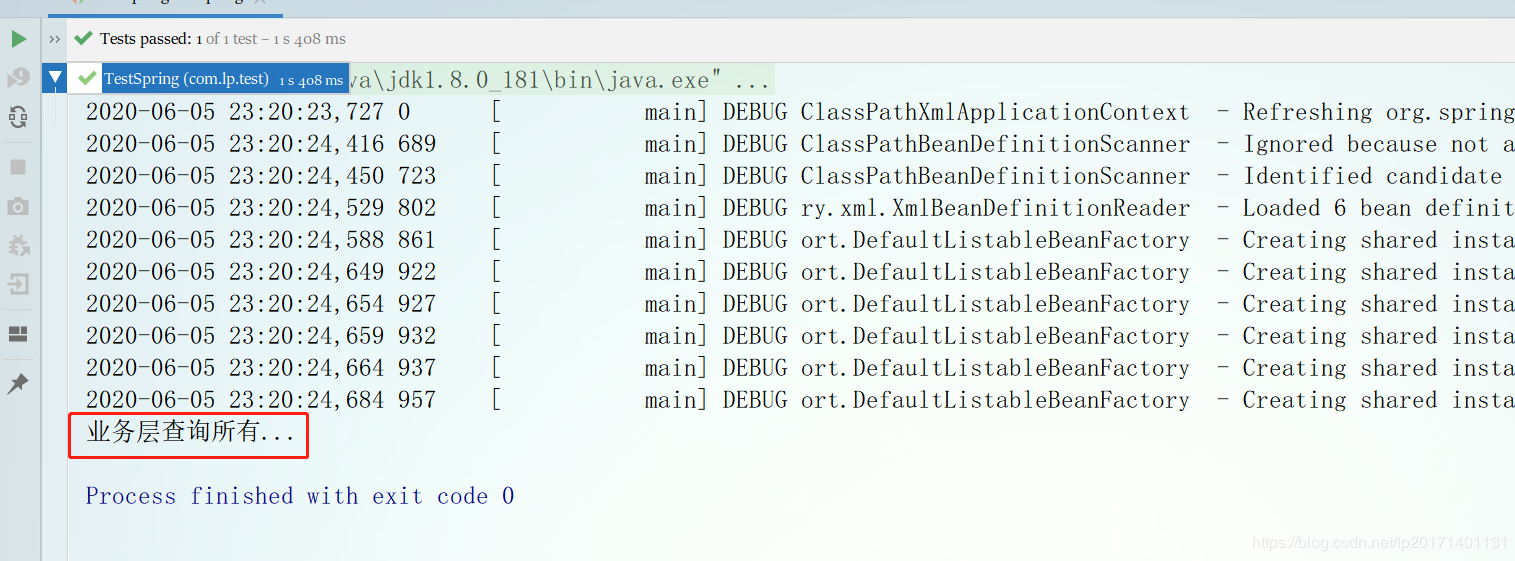
|  |
| --- |
| # Set root category priority to INFO and its only appender to CONSOLE.  #log4j.rootCategory=INFO, CONSOLE debug info warn error fatal  log4j.rootCategory=debug, CONSOLE, LOGFILE  # Set the enterprise logger category to FATAL and its only appender to CONSOLE.  log4j.logger.org.apache.axis.enterprise=FATAL, CONSOLE  # CONSOLE is set to be a ConsoleAppender using a PatternLayout.  log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender  log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout  log4j.appender.CONSOLE.layout.ConversionPattern=%d{ISO8601} %-6r [%15.15t] %-5p %30.30c %x - %m\n  # LOGFILE is set to be a File appender using a PatternLayout.  log4j.appender.LOGFILE=org.apache.log4j.FileAppender  log4j.appender.LOGFILE.File=d:\axis.log  log4j.appender.LOGFILE.Append=true  log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout  log4j.appender.LOGFILE.layout.ConversionPattern=%d{ISO8601} %-6r [%15.15t] %-5p %30.30c %x - %m\n |

**3、Spring的环境搭建好了，我们测试一下是否搭建成功！**

测试类TestSpring.java

|  |
| --- |
| package com.lp.test;  import com.lp.service.AccountService;  import org.junit.Test;  import org.springframework.context.ApplicationContext;  import org.springframework.context.support.ClassPathXmlApplicationContext;  /\*\*  \* @Date 2020/6/5 22:23  \* @Author luopeng  \*/  public class TestSpring {  @Test  public void testSpring(){  ApplicationContext ac = new ClassPathXmlApplicationContext("application.xml");  AccountService accountService = ac.getBean("accountService", AccountService.class);  accountService.findAll();  }  } |

结果如下：说明搭建成功！



# **接下来我们开始搭建SpringMVC的环境了**

web.xml

|  |
| --- |
| <!DOCTYPE web-app PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  "http://java.sun.com/dtd/web-app\_2\_3.dtd" >  <web-app>  <display-name>Archetype Created Web Application</display-name>  <!-- 配置编码过滤器-->  <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>  </filter>  <filter-mapping>  <filter-name>encodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <!-- 配置前端控制器-->  <servlet>  <servlet-name>dispatcherServlet</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:springmvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>dispatcherServlet</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |

前端控制器拦截了所有的网页请求，来扫描springmvc.xml中的的配置  
所以resources目录下需要springmvc.xml配置文件，此时的springmvc只需要扫描controller层即可，其他不归他管！

|  |
| --- |
| <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:context="http://www.springframework.org/schema/context"  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/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc.xsd">  <!-- 开启自动扫描 -->  <context:component-scan base-package="com.lp">  <context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>  </context:component-scan>  <!-- 注册SpringMVC注解开发驱动 -->  <mvc:annotation-driven/>  <!-- 视图解析器-->  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <!-- 前缀 -->  <property name="prefix" value="/WEB-INF/pages/"></property>  <!-- 后缀 -->  <property name="suffix" value=".jsp"></property>  </bean>  <!-- 前端控制器，哪些静态资源不拦截-->  <mvc:resources mapping="/js/\*\*" location="/js/"/>  <mvc:resources mapping="/css/\*\*" location="/css/"/>  <mvc:resources mapping="/img/\*\*" location="/img/"/>  </beans> |

创建AccountController.java类

|  |
| --- |
| package com.lp.controller;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  import javax.xml.ws.soap.Addressing;  /\*\*  \* @Date 2020/6/5 22:17  \* @Author luopeng  \*/  @Controller  @RequestMapping("/account")  public class AccountController {  @RequestMapping("/testSpringMVC")  public String testSpringMVC(){  System.out.println("控制层执行。。");  return "success";  }  } |

这里去往success.jsp页面，我们创建这个页面，WEB-INF/pages/success.jsp

|  |
| --- |
| <%@ page contentType="text/html;charset=UTF-8" language="java" %>  <html>  <body>  <h2>Hello World!</h2>  <a href="/account/testSpringMVC">跳转测试</a>  </body>  </html> |

好，到这一步，我们该做的事就是先将Spring和SpringMVC整合一下，整合我们需要做什么呢？

首先考虑一下，我们让他们整合是不是就等于是在controller里面能够调用业务层就证明整合成功了，那我们怎么整合，先前我们在springmvc.xml中配置它只扫描@Controller注解，我们的前端控制器也只加载了springmvc.xml，所以现在spring的配置文件根本没有在ioc容器中，我们也无法获取到它，所以我们需要将它在Tomcat服务器启动时让这个spring配置文件生效！

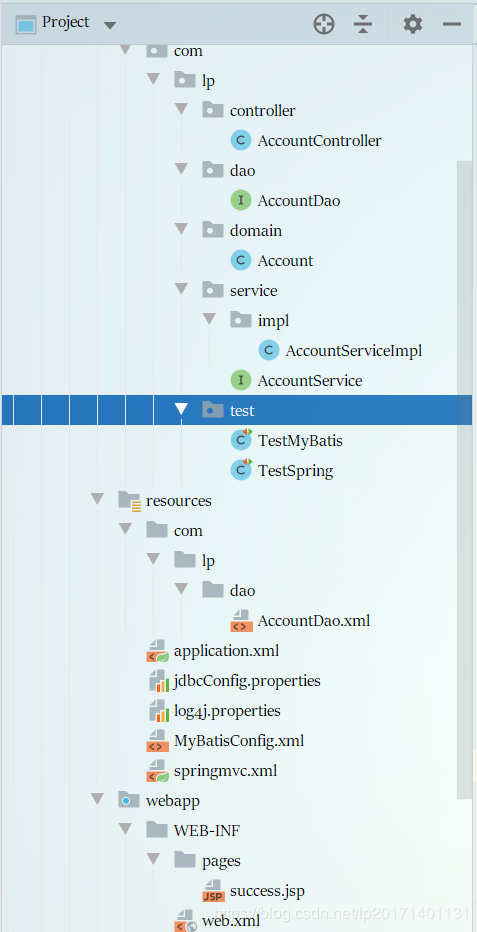
此时SpringMVC提供了一个监听器，可以让它被扫描到，此时的web.xml如下

|  |
| --- |
| <!DOCTYPE web-app PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  "http://java.sun.com/dtd/web-app\_2\_3.dtd" >  <web-app>  <display-name>Archetype Created Web Application</display-name>  <!-- 监听器默认扫描WEB-INF目录下的配置文件，  这里配置让其扫描类路径下的application.xml配置文件-->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:application.xml</param-value>  </context-param>  <!-- 配置编码过滤器-->  <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>  </filter>  <filter-mapping>  <filter-name>encodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <!--配置Spring监听器-->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <!-- 配置前端控制器-->  <servlet>  <servlet-name>dispatcherServlet</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:springmvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>dispatcherServlet</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |

好，此时Spring配置文件通过这一步配置就能生效了，我们在AccountController.java中验证一下能不能取到它

|  |
| --- |
| package com.lp.controller;  import com.lp.service.AccountService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  import javax.xml.ws.soap.Addressing;  /\*\*  \* @Date 2020/6/5 22:17  \* @Author luopeng  \*/  @Controller  @RequestMapping("/account")  public class AccountController {  @Autowired  private AccountService accountService;  @RequestMapping("/testSpringMVC")  public String testSpringMVC(){  System.out.println("控制层执行。。");  accountService.findAll();  return "success";  }  } |

**5、上面讲了Spring和SpringMVC的整合，现在开始下一步，先来搭建一下MyBatis的环境**



首先需要配置文件，名字随意，我这里叫做MyBatisConfig.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>  <!--外部引入jdbc配置文件-->  <properties resource="jdbcConfig.properties"/>  <!-- 配置实体类别名,默认别名为类的字母（不区分大小写）-->  <typeAliases>  <package name="com.lp.domain"/>  </typeAliases>  <!--配置环境-->  <environments default="localhost">  <environment id="localhost">  <transactionManager type="JDBC"/>  <!-- 配置数据库连接信息 -->  <dataSource type="POOLED">  <property name="driver" value="${jdbc.driver}"/>  <property name="url" value="${jdbc.url}"/>  <property name="username" value="${jdbc.username}"/>  <property name="password" value="${jdbc.password}"/>  </dataSource>  </environment>  </environments>  <!--配置dao接口所在位置-->  <mappers>  <package name="com.lp.dao"/>  </mappers>  </configuration> |

上面所需要的jdbc配置文件写一下

|  |
| --- |
| jdbc.driver=com.mysql.jdbc.Driver  jdbc.url=jdbc:mysql://localhost:3306/spring?useUnicode=true&characterEncoding=utf-8&useSSL=false  jdbc.username=root  jdbc.password=962464 |

现在可以编写一下AccountDao的映射文件了，在这里面编写CRUD代码

|  |
| --- |
| <?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.lp.dao.AccountDao">  <!--如果实体类命名与数据库不一致就需要resultMap-->  <resultMap id="accountMap" type="Account">  <id property="id" column="id"/>  <result property="name" column="name"/>  <result property="password" column="password"/>  </resultMap>  <select id="findAll" resultMap="accountMap">  select \* from account;  </select>  <insert id="saveAccount" parameterType="Account">  insert into account  (name ,money)  values  (#{name},#{money});  </insert>  </mapper> |

到这一步，MyBatis的环境搭建好了，开始测试一下是都搭建成功

|  |
| --- |
| package com.lp.test;  import com.lp.dao.AccountDao;  import com.lp.domain.Account;  import com.lp.service.AccountService;  import org.apache.ibatis.io.Resources;  import org.apache.ibatis.session.SqlSession;  import org.apache.ibatis.session.SqlSessionFactory;  import org.apache.ibatis.session.SqlSessionFactoryBuilder;  import org.junit.Test;  import java.io.IOException;  import java.io.InputStream;  /\*\*  \* @Date 2020/6/6 12:00  \* @Author luopeng  \*/  public class TestMyBatis {  @Test  public void testFindAll() throws IOException {  InputStream in = Resources.getResourceAsStream("MyBatisConfig.xml");  SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(in);  SqlSession sqlSession = factory.openSession();  AccountDao accountDao = sqlSession.getMapper(AccountDao.class);  for (Account account : accountDao.findAll()) {  System.out.println(account);  }  sqlSession.close();  in.close();  }  @Test  public void testSaveAccount() throws IOException {  InputStream in = Resources.getResourceAsStream("MyBatisConfig.xml");  SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(in);  SqlSession sqlSession = factory.openSession();  AccountDao accountDao = sqlSession.getMapper(AccountDao.class);  Account account = new Account();  account.setName("张三");  account.setMoney(5000D);  accountDao.saveAccount(account);  sqlSession.commit();  sqlSession.close();  in.close();  testFindAll();  }  } |

# **注意，如果是从上一篇文章下载的同志，记得最后出错的原因是：我的c3p0连接池版本太低了，我去看了看，05年的，我最后Google，找到原因，换到了下面的版本**

|  |
| --- |
| <dependency>  <groupId>com.mchange</groupId>  <artifactId>c3p0</artifactId>  <version>0.9.5.4</version>  </dependency> |

**现在开始，我们就要进行Spring和MyBatis的整合了，整合完成，整个项目就完成了**

其实很简单，就是把原本需要MyBatis做的事情交给Spring来做，目的是为了把工厂生成的代理对象交给Spring来管理

首先，application.xml中我们需要以下的配置，注释步骤很明白的

|  |
| --- |
| <?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:aop="http://www.springframework.org/schema/aop"  xmlns:context="http://www.springframework.org/schema/context"  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/aop  http://www.springframework.org/schema/aop/spring-aop.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx.xsd">  <!--开启自动扫描 -->  <context:component-scan base-package="com.lp">  <!--配置哪些不扫描-->  <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller"/>  </context:component-scan>  <!--Spring整合MyBatis框架-->  <!--jdbc配置文件引入Tomcat部署项目，  src/main/resources目录下的配置文件默认位置为：{项目名}/WEB-INF/classes，  而Spring却在项目根目录下寻找，肯定找不到，因此，配置时指定classpath目录下寻找。-->  <context:property-placeholder location="classpath:jdbcConfig.properties"/>  <!--配置连接池-->  <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">  <property name="driverClass" value="${jdbc.driver}"/>  <property name="jdbcUrl" value="${jdbc.url}"/>  <property name="user" value="${jdbc.username}"/>  <property name="password" value="${jdbc.password}"/>  </bean>  <!--配置SqlSessionFactory工厂，  这一步是为了将MyBatis在工厂中生成的代理对象放在Spring容器中-->  <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  <property name="dataSource" ref="dataSource"/>  <!--给实体类取别名-->  <property name="typeAliasesPackage" value="com.lp.domain"/>  </bean>  <!--配置AccountDao接口所在包-->  <bean id="mapperScanner" class="org.mybatis.spring.mapper.MapperScannerConfigurer">  <property name="basePackage" value="com.lp.dao"/>  </bean>  <!-- 配置Spring框架声明式事务管理-->  <!-- spring中基于XML的声明式事务控制配置步骤  1、配置事务管理器  2、配置事务的通知  此时我们需要导入事务的约束 tx名称空间和约束，同时也需要aop的  使用tx:advice标签配置事务通知  属性：  id：给事务通知起一个唯一标识  transaction-manager：给事务通知提供一个事务管理器引用  3、配置AOP中的通用切入点表达式  4、建立事务通知和切入点表达式的对应关系  5、配置事务的属性  是在事务的通知tx:advice标签的内部  -->  <!-- 配置事务管理器 -->  <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  <property name="dataSource" ref="dataSource"/>  </bean>  <!--配置事务通知-->  <tx:advice id="txAdvice" transaction-manager="transactionManager">  <!-- 配置事务的属性  isolation：用于指定事务的隔离级别。默认值是DEFAULT，表示使用数据库的默认隔离级别。  propagation：用于指定事务的传播行为。默认值是REQUIRED，表示一定会有事务，增删改的选择。查询方法可以选择SUPPORTS。  read-only：用于指定事务是否只读。只有查询方法才能设置为true。默认值是false，表示读写。  timeout：用于指定事务的超时时间，默认值是-1，表示永不超时。如果指定了数值，以秒为单位。  rollback-for：用于指定一个异常，当产生该异常时，事务回滚，产生其他异常时，事务不回滚。没有默认值。表示任何异常都回滚。  no-rollback-for：用于指定一个异常，当产生该异常时，事务不回滚，产生其他异常时事务回滚。没有默认值。表示任何异常都回滚。  -->  <tx:attributes>  <tx:method name="\*" propagation="REQUIRED"/>  <tx:method name="find\*" read-only="true" propagation="SUPPORTS"/>  <tx:method name="get\*" read-only="true" propagation="SUPPORTS"/>  <tx:method name="select\*" read-only="true" propagation="SUPPORTS"/>  </tx:attributes>  </tx:advice>  <!--配置AOP-->  <aop:config>  <!--配置切入点表达式-->  <aop:pointcut id="pt1" expression="execution(\* com.lp.service.impl.\*ServiceImpl.\*(..))"/>  <!--配置事务通知与切入点表达式的对应关系 -->  <aop:advisor advice-ref="txAdvice" pointcut-ref="pt1"/>  </aop:config>  </beans> |

如果觉得太多了，可以把他分开配置，比如说，你要配置Spring和MyBatis的整合文件，你可以命名为spring-mybatis.xml，最后你最关键的一点记得在web.xml中让它能加载这个配置文件。当然事务管理的我们也可以单独来一个spring-service啦！

我踩过的坑在注释中有记录，也有提醒！