### SpringMVC工作流程

1. Dispatcher Servlet(前端控制器)：核心组件，协调调用其他组件。
2. handlerMapping(映射处理器)：根据请求查找处理该请求的处理器（Handler）
3. Handler:处理请求的方法
4. HandlerAdapter(处理器适配器):适配参数以及调用执行处理器方法。
5. ModelAndVIew:封装跳转的页面以及向页面传递的参数
6. ViewResolver(视图解析器)：查找跳转的页面并将数据解析到该页面。

### IOC

**（Inversion of Control）**控制反转，应用本身不负责对象的创建，由spring容器（spring配置文件）完成对象的创建。

### DI

**（(Dependency Injection）**依赖注入，即为Spring容器中类中属性赋值的过程称为依赖注入。

依赖注入两种方式（xml）：**所谓依赖注入，就是把底层类作为参数传入上层类，实现上层类对下层类的控制**

## 动态代理

1. 什么是代理：一个类（代理类）或者一个机构帮助另外一个类（目标类）或者一个机构完成一部分额外的业务，从而可以增强目标类业务的作用，同时降低主要业务和辅助业务的耦合度。
2. 静态代理:为每一个目标类手动的创建一个代理类。
3. 动态代理：根据指定的目标类，通过反射动态生成对应的代理类。

JDK代理：针对实现了接口的目标类代理（目标类和代理类实现了相同的接口）

## AOP

在软件业，AOP为Aspect Oriented Programming的缩写，意为：[面向切面编程](https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%88%87%E9%9D%A2%E7%BC%96%E7%A8%8B/6016335)，通过[预编译](https://baike.baidu.com/item/%E9%A2%84%E7%BC%96%E8%AF%91/3191547)方式和运行期动态代理实现程序功能的统一维护的一种技术。AOP是[OOP](https://baike.baidu.com/item/OOP)的延续，是软件开发中的一个热点，也是[Spring](https://baike.baidu.com/item/Spring)框架中的一个重要内容，是[函数式编程](https://baike.baidu.com/item/%E5%87%BD%E6%95%B0%E5%BC%8F%E7%BC%96%E7%A8%8B/4035031)的一种衍生范型。利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的[耦合度](https://baike.baidu.com/item/%E8%80%A6%E5%90%88%E5%BA%A6/2603938)降低，提高程序的可重用性，同时提高了开发的效率。

1. Aspect Oriented Programming,通过动态代理实现对功能的统一维护；
2. 对OOP的补充，降低业务逻辑之间的耦合度；
3. 纵向重复，横向抽取

### SSM整合思想

1. Spring集成Mybatis（数据源，实例化SqlSessionFactory,实例化Mapper代理模式生成dao层实现类,事务控制）
2. Spring集成SpringMVC（去除重复扫描）
3. 在web.xml中加载spring和SpringMVC 的配置文件

## SSM集成步骤

1. 创建项目添加依赖包  
   spring+springmvc+mybaits+mybatis和spring 的中间包
2. 加入三个框架的配置文件

Spring集成Mybatis（数据源，实例化SqlSessionFactory,实例化Mapper代理模式生成dao层实现类,事务控制）

从mybatis的配置文件中去除数据源，然后修改spring配置文件如下：

|  |
| --- |
| <context:component-scan base-package=*"com.offcn"*>  <!-- 去除对控制层的重复扫描 -->  <context:exclude-filter type=*"annotation"* expression=*"org.springframework.stereotype.Controller"*/>  </context:component-scan>    <!-- 引入外部的 properties文件-->  <context:property-placeholder location=*"classpath:db.properties"*/>  <!-- c3p0数据源注入 -->  <bean id=*"ds"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>  <property name=*"driverClass"* value=*"${db\_driver}"*></property>  <property name=*"jdbcUrl"* value=*"${db\_url}"*></property>  <property name=*"user"* value=*"${db\_uname}"*></property>  <property name=*"password"* value=*"${db\_pwd}"*></property>  </bean>    <!-- 实例化SqlSessionFactory -->  <bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <property name=*"configLocation"* value=*"classpath:mybatisconf.xml"*></property>  <property name=*"dataSource"* ref=*"ds"*></property>  </bean>    <!-- 实例化dao层的实现类（由mapper代理模式生成的实现类） -->  <bean class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <!-- 指定dao层接口（实现类）所在的包名 -->  <property name=*"basePackage"* value=*"com.offcn.dao"*></property>  </bean>      <!-- 实例化事务的切面类（代理类） -->  <bean id=*"tx"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <property name=*"dataSource"* ref=*"ds"*></property>  </bean>    <!-- 事务的通知：事务的特性 -->  <tx:advice id=*"myadvice"* transaction-manager=*"tx"*>  <tx:attributes>  <!-- name:指定连接点  isolation：隔离级别，DEFAULT隔离级别和数据库保持一致  propagation：传播行为：设置当前业务所处的事务环境REQUIRED：如果有事务就使用当前的事务环境，否则会开启一个新的事务  read-only：是否只读 -->  <tx:method name=*"\*"* isolation=*"DEFAULT"* propagation=*"REQUIRED"* read-only=*"true"*/>  <tx:method name=*"trans\*"* isolation=*"DEFAULT"* propagation=*"REQUIRED"* read-only=*"false"*/>  </tx:attributes>  </tx:advice>    <aop:config>  <aop:pointcut expression=*"execution(\* com.offcn.service.\*.\*(..))"* id=*"pid"*/>  <aop:advisor advice-ref=*"myadvice"* pointcut-ref=*"pid"*/>  </aop:config> |

Spring集成SpringMVC（去除重复扫描）

从spring的配置文件beans.xml中去除对控制层的扫描

|  |
| --- |
| <context:component-scan base-package=*"com.offcn"*>  <!-- 去除对控制层的重复扫描 -->  <context:exclude-filter type=*"annotation"* expression=*"org.springframework.stereotype.Controller"*/>  </context:component-scan> |

3.在web.xml中加载spring和SpringMVC 的配置文件

|  |
| --- |
| <!--编码过滤器 -->  <filter>  <filter-name>encode</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>encode</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <!-- 前端控制器以及springmvc配置文件配置 -->  <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:springmvc.xml</param-value>  </init-param>  </servlet>  <servlet-mapping>  <servlet-name>springmvc</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>    <!-- 通过监听器加载spring配置文件 -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <!-- 设置监听的配置文件路径 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:beans.xml</param-value>  </context-param> |

