目录

[BFP整体依靠SSM框架构建 1](#_Toc522712200)

[Spring 1](#_Toc522712201)

[SpringMVC 3](#_Toc522712202)

[Mybatis 4](#_Toc522712203)

[Druid 5](#_Toc522712204)

[Shiro 5](#_Toc522712205)

[Redis 7](#_Toc522712206)

[Quartz 7](#_Toc522712207)

# BFP整体依靠SSM框架构建

Spring作为容器类框架，负责对象的实例化和管理各个对象之间的关系，同时提供声明式事务和面向切面编程，方便集成其他优秀框架。

SpringMVC在项目中拦截用户请求，通过DispatcherServlet将用户请求通过HandlerMapping去匹配Conntroller，再通过Controller去对应具体的执行操作。相当于ssh框架中的Struts。

Mybatis完成了对jdbc的封装。其所有的操作都是围绕一个sqlSessionFactory实例展开。mybatis通过配置文件关联到各实体类的Mapper文件，Mapper文件中配置了每个dao类对数据库所需进行的sql语句映射。

## Spring

配置扫描注解

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

配置自动扫描的包名

<**context:component-scan base-package="com.hikvision.vfp, com.hikvision.bfp"**/>

<context:annotation-config>处理@autowired之类（共有四类），前提是注解作用的类已经被注册到spring容器里

<context:component-scan>除了包含annotation-config的作用外，还能自动扫描和注册base-package下有@component之类注解的类，将其作为bean注册到spring容器里

所以配置文件如果有component-scan就不需要annotation-config了。

在操作数据库时开启基于数据库的事务管理

<tx:annotation-driven transaction-manager="transactionManager"/>

<**bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  
 <**property name="dataSource" ref="dynamicDataSource"**/>  
 <**property name="globalRollbackOnParticipationFailure" value="false"**/> *<!--指定此参数为false-->*</**bean**>

设置各个方法的事务传播行为和隔离级别

<**tx:advice id="txAdvice" transaction-manager="transactionManager"**>  
 <**tx:attributes**>  
 <**tx:method name="get\*" propagation="NOT\_SUPPORTED" read-only="true"**/>  
 <**tx:method name="find\*" propagation="NOT\_SUPPORTED" read-only="true"**/>  
 <**tx:method name="update\*" propagation="REQUIRED" isolation="DEFAULT" rollback-for="Exception"**/>  
 ………  
 <**tx:method name="delete\*" propagation="REQUIRED" isolation="DEFAULT" rollback-for="Exception"**/>  
 </**tx:attributes**>  
</**tx:advice**>

配合切面的通知器实现事务管理

<**aop:config**>  
 <**aop:pointcut id="servicePointcut" expression="execution(\* com.hikvision.vfp.\*.service..\*.\*(..)) or execution(\* com.hikvision.vfp.\*.\*.service..\*.\*(..)) or  
 execution(\* com.hikvision.bfp.techaccess.\*.service..\*.\*(..)) or execution(\* com.hikvision.bfp.techweb.\*.service..\*.\*(..))** />  
 <**aop:advisor advice-ref="txAdvice" pointcut-ref="servicePointcut"**/>  
</**aop:config**>

在web.xml文件中配置spring，扫描到spring-context.xml

<**context-param**>  
 <**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>classpath:spring/application-\*.xml</**param-value**>  
</**context-param**>

Spring监听器

<**listener**>  
 <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**>  
</**listener**>  
<**listener**> <**listener-class**>org.springframework.web.context.request.RequestContextListener</**listener-class**>  
</**listener**>

Spring字符集过滤器

<**filter**>  
 <**filter-name**>characterEncodingFilter</**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**>forceEncoding</**param-name**>  
 <**param-value**>true</**param-value**>  
 </**init-param**>  
</**filter**>  
<**filter-mapping**>  
 <**filter-name**>characterEncodingFilter</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
</**filter-mapping**>

## SpringMVC

<mvc:default-servlet-handler/>

会在Spring MVC上下文中定义一个org.springframework.web.servlet.resource.DefaultServletHttpRequestHandler，它会像一个检查员，对进入DispatcherServlet的URL进行筛查，如果发现是静态资源的请求，就将该请求转由Web应用服务器默认的Servlet处理，如果不是静态资源的请求，才由DispatcherServlet继续处理。

<**mvc:annotation-driven/**>

标签会为我们注册三个bean，分别是RequestMappingHandlerMapping,RequestMappingHandlerAdapter,ExceptionHandler,使得我们的@RequestMapping,@RequestParam,@ExceptionHandler这些注解生效，在之前这些都得手动配置才行。

<**mvc:interceptors**>  
 *<!-- 使用bean定义一个Interceptor，直接定义在mvc:interceptors根下面的Interceptor将拦截所有的请求 -->* <**bean class="com.hikvision.vfp.web.interceptor.SessionInterceptor"**/></**mvc:interceptors**>

配置一个拦截器，通过重写prehandle、posthandle或afterCompetion方法，在controller层方法执行前、后或请求完成后可以做session或权限等相关操作。

在web.xml添加springMVC配置

<**servlet**>  
 <**servlet-name**>spring</**servlet-name**>  
 <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  
 <**init-param**>  
 <**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>classpath:spring/application-context.xml</**param-value**>  
 </**init-param**>  
 <**load-on-startup**>0</**load-on-startup**>  
</**servlet**>  
<**servlet-mapping**>  
 <**servlet-name**>spring</**servlet-name**>  
 <**url-pattern**>/</**url-pattern**>  
</**servlet-mapping**>

相关注解：

在controller层：@RestController、@RequestMapping("/")、@PostMapping(value = {"/ "})、@ResponseBody、@GetMapping(value = {"/ "})、@Autowired等

在service层：@Service("")等

## Mybatis

采用阿里的druid配置数据源

<**bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close"**>  
 <**property name="url" value="${jdbc.url}"**/>  
 <**property name="username" value="${jdbc.username}"**/>  
 <**property name="password" value="${jdbc.password}"**/>  
 ……  
</**bean**>

另，通过以下配置去扫描属性文件，添加配置项<**bean id="propertyConfigurer" class="com.hikvision.vfp.utils.AddressPopeModel"**>  
 <**property name="ignoreResourceNotFound" value="true"**/>  
 <**property name="locations"**>  
 <**list**>  
 <**value**>classpath:config.properties</**value**>  
 </**list**>  
 </**property**>  
</**bean**

将数据源交给sessionFactory管理，并配置需要扫描的mapper.xml文件位置

<**bean id="lazySqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  
 <**property name="dataSource" ref="dynamicDataSource"**/>  
 *<!-- 当mybatis的xml文件和mapper接口不在相同包下时，需要用mapperLocations属性指定xml文件的路径。  
 \*是个通配符，代表所有的文件，\*\*代表所有目录下 -->* <**property name="mapperLocations"**>  
 <**array**>  
 <**value**>classpath:mapper/\*.xml</**value**>  
 <**value**>classpath:mapper/\*\*/\*.xml</**value**>  
 </**array**>  
 </**property**>  
</**bean**>

扫描mybatis映射接口类

<**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  
 <**property name="basePackage" value="com.hikvision.vfp.\*.dao, com.hikvision.vfp.\*.\*.dao, com.hikvision.bfp.\*.dao,com.hikvision.bfp.\*.\*.dao"**/>  
 <**property name="sqlSessionFactoryBeanName" value="lazySqlSessionFactory"**/>  
</**bean**>

然后就可以编写xxDao.xml文件，通过<mapper>的namespace属性将xxDao.xml与xxDao.java文件关联起来了。

## Druid

除了使用druid的数据连接池以外，还可以使用它的web监控统计功能。在web.xml文件中添加下列代码：

<**filter**>  
 <**filter-name**>DruidWebStatFilter</**filter-name**>  
 <**filter-class**>com.alibaba.druid.support.http.WebStatFilter</**filter-class**>  
 <**init-param**>  
 <**param-name**>exclusions</**param-name**>  
 <**param-value**>\*. js ,\*. gif ,\*. jpg ,\*. png ,\*. css ,\*. ico ,/druid/\*</**param-value**>  
 </**init-param**>  
</**filter**>  
<**filter-mapping**>  
 <**filter-name**>DruidWebStatFilter</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
</**filter-mapping**>  
<**servlet**>  
 <**servlet-name**>DruidStatView</**servlet-name**>  
 <**servlet-class**>com.alibaba.druid.support.http.StatViewServlet</**servlet-class**>  
</**servlet**>  
<**servlet-mapping**>  
 <**servlet-name**>DruidStatView</**servlet-name**>  
 <**url-pattern**>/druid/\*</**url-pattern**>  
</**servlet-mapping**>

然后项目启动后，访问localhost:port/druid/index.html即可

## Shiro

Shiro是安全管理框架，主要用来完成身份验证、授权、密码学和会话管理。

安全管理器securityManager是shiro框架的核心，shiro通过它来管理内部组件实例和提供安全管理的各种服务。

<**bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager"**>  
 <**property name="realm" ref="customRealm"**/>  
 <**property name="cacheManager" ref="redisCacheManager"**/>  
 <**property name="sessionManager" ref="sessionManager"**/>  
</**bean**>

其中的realm充当了shiro与应用安全数据间的桥梁。当用户执行登录和授权验证时，shiro会从配置的realm中查找用户及其权限信息。

<**bean id="customRealm" class="com.hikvision.vfp.web.shiro.CustomRealm"**>  
 <**property name="credentialsMatcher" ref="credentialsMatcher"**/>  
 <**property name="cachingEnabled" value="true"**/>  
 <**property name="cacheManager" ref="redisCacheManager"**/>  
</**bean**>

缓存管理器交给redis实现

<**bean id="redisCacheManager" class="com.hikvision.vfp.cache.RedisCacheManager"**>  
 <**property name="redisTemplate" ref="redisTemplate"**/><**property name="defaultExpiration" value="${redis.sessionTimeout}"**/>  
</**bean**>

会话管理器，设定会话超时及保存

<**bean id="sessionManager" class="org.apache.shiro.web.session.mgt.DefaultWebSessionManager"**>  
 *<!-- 全局会话超时时间（单位毫秒），默认30分钟，86400000为24小时 -->* <**property name="globalSessionTimeout" value="${session.timeout}"**/>  
 <**property name="sessionValidationSchedulerEnabled" value="true"**/>

<!-- sessionDAO保存认证信息 -->  
 <**property name="sessionDAO" ref="redisSessionDAO"**/>  
 <**property name="sessionIdCookie.name" value="${shiro.sessionIdCookie.name}"**/>  
</**bean**>

Shiro的web过滤器：

<**bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean"**>

……….

</**bean**>

相关概念：subject、token

一般需要在继承AuthorizingRealm的类中重写doGetAuthorizationInfo方法(提供用户信息返回权限信息)和doGetAuthenticationInfo(提供账户信息返回认证信息)方法

在继承SimpleCredentialsMatcher的类中重写doCredentialsMatch方法。

## Redis

Redis作为key-value型数据库，在项目中的应用是对postgresql关系型数据库的补充，可用于高速缓存。

## Quartz

Quartz作为作业调度框架，可以用于处理定时任务。

Quartz定时任务配置:使用MethodInvokingJobDedailFactoryBean,任务类可以不实现Job接口,通过targetMethod指定调用方法 cron表达式:\*\*\*\*\*\*(共6位,使用空格隔开,具体如下 )

cron表达式:\*(秒0-59)\*(分钟0-59) \*(小时0-23)\*(日期1-31)\*(月份1-12或JAN-DEC)\*(星期1-7或SUN-SAT)

*<!-- 请求数据统计定时器-->*<**bean id="statisticDataTimer" class="com.hikvision.bfp.timer.StatisticDataTimer"**></**bean**>  
<**bean id="statisticDataJobMethod" class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean"**>  
 <**property name="targetObject" ref="statisticDataTimer"**/>  
 <**property name="targetMethod" value="flushDataToDB"**/>  
 <**property name="concurrent" value="false"**/>  
</**bean**>  
*<!-- =======调度触发器 ========== -->*<**bean id="statisticDataCronTriggerBean" class="org.springframework.scheduling.quartz.CronTriggerFactoryBean"**>  
 <**property name="jobDetail" ref="statisticDataJobMethod"**/>  
 <**property name="startDelay" value="1000"**/> *<!--调度器启动与第一次执行任务之间的间隔 (单位：毫秒)-->* <**property name="cronExpression" value="0 0 \*/1 \* \* ?"**/>  
</**bean**>