

























步骤

1. 定义自定义的annotation类型的log

|  |
| --- |
| **package** cn.huangwei.annotation;  **import** java.lang.annotation.Documented;  **import** java.lang.annotation.ElementType;  **import** java.lang.annotation.Retention;  **import** java.lang.annotation.RetentionPolicy;  **import** java.lang.annotation.Target;  @Target(ElementType.***METHOD***)  @Documented  @Retention(RetentionPolicy.***RUNTIME***)  **public** **@interface** Token {  **boolean** save() **default** **false**;  **boolean** remove() **default** **false**;  } |

1. 在指定的方法上加上定义的annotation的日志类型

|  |
| --- |
| **package** cn.huangwei.aop;  **import** cn.huangwei.annotation.Token;  **public** **class** Book {  @Token(remove=**true**)  **public** **void** add(){  System.***out***.println("add......");  }  } |

1. 定义切片

|  |
| --- |
| **package** cn.huangwei.aspect;  **import** java.lang.reflect.Method;  **import** org.aspectj.lang.JoinPoint;  **import** org.aspectj.lang.annotation.Aspect;  **import** org.aspectj.lang.annotation.Before;  **import** org.aspectj.lang.annotation.Pointcut;  **import** org.springframework.stereotype.Component;  **import** cn.huangwei.annotation.Token;  @Component  @Aspect  **public** **class** TokenAspect {  @Pointcut("@annotation(cn.huangwei.annotation.Token)")//直接定位到annotation的类的类的路径名  **public** **void** aspect(){    }    @Before("aspect()")  **public** **void** before(JoinPoint joinPoint) **throws** ClassNotFoundException{  String targetName = joinPoint.getTarget().getClass().getName();//得到目标方法的类名  String methodName = joinPoint.getSignature().getName();//得到方法名  Object[] argus = joinPoint.getArgs();//获取方法的参数  Class targetClass = Class.*forName*(targetName);//取到目标类  Method[] methods = targetClass.getMethods();//取到目标类的所有方法  Token annotation = **null**;  **for** (Method method : methods) {  **if**(method.getName().equals(methodName)){//如果等于调用的方法名  Class<?>[] classes = method.getParameterTypes();  **if**(classes.length==argus.length){//并且参数也相等  annotation = method.getAnnotation(Token.**class**);//获取设置的annotation对象@Token(remove=true)  System.***out***.println(annotation.remove());//获取方法的属性值  }    }  }  }  } |

1. 开启aop---springmvc.xml

|  |
| --- |
| <!-- 开启aop操作 -->  <aop:aspectj-autoproxy proxy-target-class=*"true"* />  <context:component-scan base-package=*"cn.huangwei.aspect"*></context:component-scan> |
|  |

网上例子

|  |
| --- |
| [**Spring AOP实现复杂的日志记录（自定义注解）**](http://blog.csdn.net/mlc1218559742/article/details/51778224)  标签： [spring](http://www.csdn.net/tag/spring)[aop](http://www.csdn.net/tag/aop)[日志记录](http://www.csdn.net/tag/%e6%97%a5%e5%bf%97%e8%ae%b0%e5%bd%95)[自定义注解](http://www.csdn.net/tag/%e8%87%aa%e5%ae%9a%e4%b9%89%e6%b3%a8%e8%a7%a3)  2016-06-28 19:37 1876人阅读 [评论](http://blog.csdn.net/mlc1218559742/article/details/51778224#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/mlc1218559742/article/details/51778224#report)  http://static.blog.csdn.net/images/category_icon.jpg 分类：  Spring（7） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg Java（18） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg  版权声明：本文为博主原创文章，未经博主允许不得转载。        做项目中，业务逻辑要求只要对[**数据库**](http://lib.csdn.net/base/mysql)数据进行改动的都需要记录日志（增删改），记录的内容有操作者、操作的表名及表名称、具体的操作，以及操作对应的数据。首先想到的就是[**spring**](http://lib.csdn.net/base/javaee) 的AOP功能。可是经过一番了解过后，发现一般的日志记录，只能记录一些简单的操作，例如表名、表名称等记录不到。        于是想到了自定义注解的方法，把想要记录的内容放在注解中，通过切入点来获取注解参数，就能获取自己想要的数据，记录数据库中。顺着这个思路，在网上查找了一些相关资料，最终实现功能。话不多说，以下就是实现的思路及代码：       第一步：在代码中添加自定义注解，并且定义两个属性，一个是日志的描述（description），还有个是操作表类型（tableType），属性参数可按需求改变。代码如下：  **[java]** [view plain](http://blog.csdn.net/mlc1218559742/article/details/51778224) [copy](http://blog.csdn.net/mlc1218559742/article/details/51778224)   1. **import** java.lang.annotation.Documented; 2. **import** java.lang.annotation.ElementType; 3. **import** java.lang.annotation.Retention; 4. **import** java.lang.annotation.RetentionPolicy; 5. **import** java.lang.annotation.Target;  8. /\*\* 9. \* ClassName: SystemServiceLog <br/> 10. \* Function: AOP日志记录，自定义注解 <br/> 11. \* date: 2016年6月7日 上午9:29:01 <br/> 12. \* @author lcma 13. \* @version 14. \* @since JDK 1.7 15. \*/ 16. @Target({ElementType.PARAMETER, ElementType.METHOD}) 17. @Retention(RetentionPolicy.RUNTIME) 18. @Documented 19. **public** **@interface** SystemServiceLog { 20. /\*\* 21. \* 日志描述 22. \*/ 23. String description()  **default** ""; 25. /\*\* 26. \* 操作表类型 27. \*/ 28. **int** tableType() **default** 0; 30. }   第二步：定义切面类，获取切面参数，保存数据库具体代码如下：  **[java]** [view plain](http://blog.csdn.net/mlc1218559742/article/details/51778224) [copy](http://blog.csdn.net/mlc1218559742/article/details/51778224)   1. **import** java.lang.reflect.Method; 2. **import** java.util.Date; 4. **import** javax.annotation.Resource; 5. **import** javax.servlet.http.HttpServletRequest; 7. **import** org.apache.log4j.Logger; 8. **import** org.aspectj.lang.JoinPoint; 9. **import** org.aspectj.lang.annotation.After; 10. **import** org.aspectj.lang.annotation.Aspect; 11. **import** org.aspectj.lang.annotation.Pointcut; 12. **import** org.springframework.stereotype.Component; 13. **import** org.springframework.web.context.request.RequestContextHolder; 14. **import** org.springframework.web.context.request.ServletRequestAttributes;  17. **import** com.iflytek.zhbs.common.annotation.SystemServiceLog; 18. **import** com.iflytek.zhbs.common.util.JacksonUtil; 19. **import** com.iflytek.zhbs.common.util.WebUtils; 20. **import** com.iflytek.zhbs.dao.BaseDaoI; 21. **import** com.iflytek.zhbs.domain.CmsAdmin; 22. **import** com.iflytek.zhbs.domain.CmsOperationLog;  25. @Aspect 26. @Component 27. @SuppressWarnings("rawtypes") 28. **public** **class** SystemLogAspect { 30. @Resource 31. **private** BaseDaoI<CmsOperationLog> logDao; 33. /\*\* 34. \* 日志记录 35. \*/ 36. **private** **static** **final** Logger LOGGER = Logger.getLogger(SystemLogAspect.**class**); 38. /\*\* 39. \* Service层切点 40. \*/ 41. @Pointcut("@annotation(com.iflytek.zhbs.common.annotation.SystemServiceLog)") 42. **public** **void** serviceAspect() { 44. } 46. /\*\* 47. \* doServiceLog:获取注解参数，记录日志. <br/> 48. \* @author lcma 49. \* @param joinPoint 切入点参数 50. \* @since JDK 1.7 51. \*/ 52. @After("serviceAspect()") 53. **public**  **void** doServiceLog(JoinPoint joinPoint) { 54. LOGGER.info("日志记录"); 55. HttpServletRequest request = ((ServletRequestAttributes) RequestContextHolder.getRequestAttributes()).getRequest(); 56. //获取管理员用户信息 57. CmsAdmin admin = WebUtils.getAdminInfo(request); 58. **try** { 59. //数据库日志 60. CmsOperationLog log = **new** CmsOperationLog(); 61. log.setOperationType(getServiceMthodTableType(joinPoint)); 62. //获取日志描述信息 63. String content = getServiceMthodDescription(joinPoint); 64. log.setContent(admin.getRealName() + content); 65. log.setRemarks(getServiceMthodParams(joinPoint)); 66. log.setAdmin(admin); 67. log.setCreateTime(**new** Date()); 68. logDao.save(log); 69. }  **catch** (Exception e) { 70. LOGGER.error("异常信息:{}", e); 71. } 72. }  75. /\*\* 76. \* getServiceMthodDescription:获取注解中对方法的描述信息 用于service层注解  . <br/> 77. \* @author lcma 78. \* @param joinPoint 切点 79. \* @return 方法描述 80. \* @throws Exception 81. \* @since JDK 1.7 82. \*/ 83. **private** String getServiceMthodDescription(JoinPoint joinPoint) 84. **throws** Exception { 85. String targetName = joinPoint.getTarget().getClass().getName(); 86. String methodName = joinPoint.getSignature().getName(); 87. Object[] arguments = joinPoint.getArgs(); 88. Class targetClass = Class.forName(targetName); 89. Method[] methods = targetClass.getMethods(); 90. String description = ""; 91. **for**(Method method : methods) { 92. **if**(method.getName().equals(methodName)) { 93. Class[] clazzs = method.getParameterTypes(); 94. **if**(clazzs.length == arguments.length) { 95. description = method.getAnnotation(SystemServiceLog.**class**).description(); 96. **break**; 97. } 98. } 99. } 100. **return** description; 101. } 103. /\*\* 104. \* getServiceMthodTableType:获取注解中对方法的数据表类型 用于service层注解 . <br/> 105. \* @author lcma 106. \* @param joinPoint 107. \* @return 108. \* @throws Exception 109. \* @since JDK 1.7 110. \*/ 111. **private** nt getServiceMthodTableType(JoinPoint joinPoint) 112. **throws** Exception { 113. String targetName = joinPoint.getTarget().getClass().getName(); 114. String methodName = joinPoint.getSignature().getName(); 115. Object[] arguments = joinPoint.getArgs(); 116. Class targetClass = Class.forName(targetName); 117. Method[] methods = targetClass.getMethods(); 118. **int** tableType = 0; 119. **for** (Method method : methods) { 120. **if** (method.getName().equals(methodName)) { 121. Class[] clazzs = method.getParameterTypes(); 122. **if** (clazzs.length == arguments.length) { 123. tableType = method.getAnnotation(SystemServiceLog.**class**).tableType(); 124. **break**; 125. } 126. } 127. } 128. **return** tableType; 129. } 131. /\*\* 132. \* getServiceMthodParams:获取json格式的参数. <br/> 133. \* @author lcma 134. \* @param joinPoint 135. \* @return 136. \* @throws Exception 137. \* @since JDK 1.7 138. \*/ 139. **private** String getServiceMthodParams(JoinPoint joinPoint) 140. **throws** Exception { 141. Object[] arguments = joinPoint.getArgs(); 142. String params = JacksonUtil.toJSon(arguments); 143. **return** params; 144. } 146. }   需要注意的是，定义切点的时候，@Pointcut里面是自定义注解的路径  每个切面传递的数据的都不一样，最终决定，获取切面的所有参数，转成json字符串，保存到数据库中。  第三步：在service需要记录日志的地方进行注解，代码如下：  **[java]** [view plain](http://blog.csdn.net/mlc1218559742/article/details/51778224) [copy](http://blog.csdn.net/mlc1218559742/article/details/51778224)   1. @SystemServiceLog(description=Constants.ADMIN\_SAVE\_OPTIONS,tableType=Constants.ADMIM\_TABLE\_TYPE)   代码图片：  http://img.blog.csdn.net/20160628211647234  在常量类里面配置自定义注解的参数内容:  http://img.blog.csdn.net/20160628211529246http://img.blog.csdn.net/20160628211543451  第四步：把切面类所在的包路径添加到Spring注解自动扫描路径下，并且启动对@AspectJ注解的支持，代码如下：  **[html]** [view plain](http://blog.csdn.net/mlc1218559742/article/details/51778224) [copy](http://blog.csdn.net/mlc1218559742/article/details/51778224)   1. <!-- 启动对@AspectJ注解的支持  --> 2. **<aop:aspectj-autoproxy** proxy-target-class="true" **/>** 3. <!-- 自动扫描包路径  --> 4. **<context:component-scan** base-package="com.iflytek.zhbs.common.aoplog" **/>** 5. **<context:component-scan** base-package="com.iflytek.zhbs.service" **/>**   最后数据库记录数据的效果如图：  http://img.blog.csdn.net/20160628211223260?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center  OK，功能已经实现，初次写博客，写的不好的地方请谅解。  通过记录工作中遇到的问题及解决思路，希望能帮助到各位网友，如内容需要改善或不理解的，欢迎及时沟通，共同进步！ |