aop的使用

笔记本：	java		
创建时间：	2018\10\12 星期五 17:16	**更新时间：**	2018\10\27 星期六 16:01
作者：	804790605@qq.com		
URL：	about:blank		

由于最近需要做一些特定操作日志记录的需求，所以使用了一下aop。
下面借由这个机会回顾下spring中aop的使用，以及aop的一些知识。

一：aop的一些概念

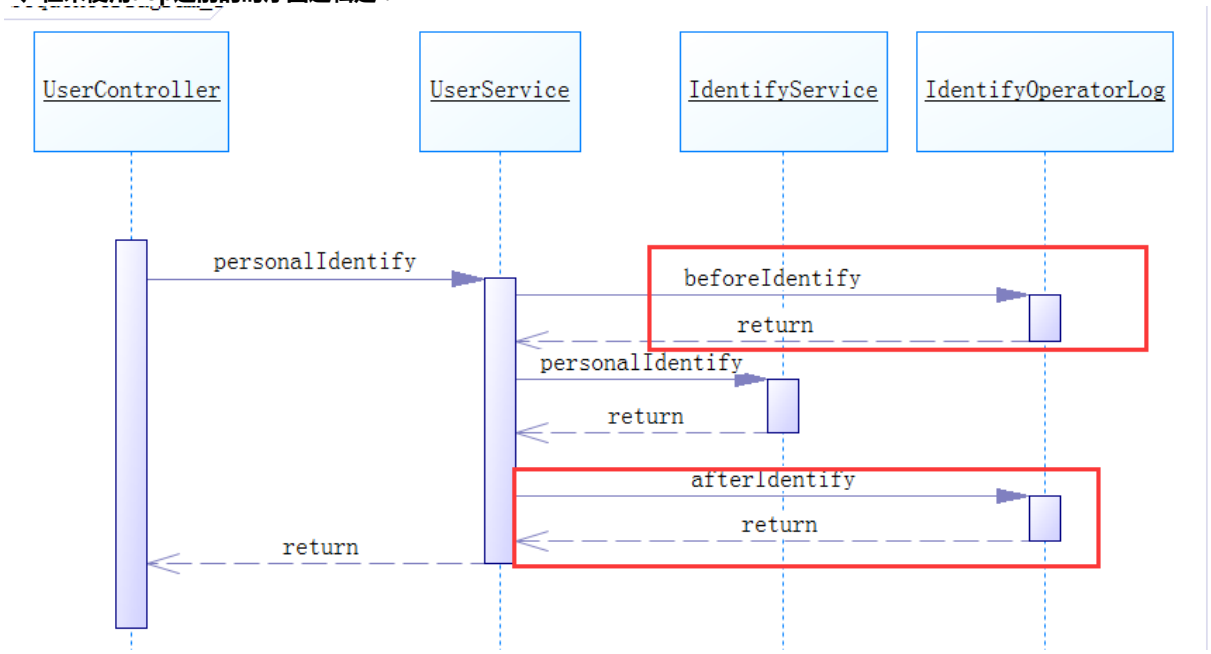
1、横切关注点：对哪些方法进行拦截，拦截后怎么处理，这些关注点称之为横切关注点
2、切面（aspect）：类是对物体特征的抽象，切面就是对横切关注点的抽象，比如上文所说的特定操作（不管访不访问，都已经客观存在那里了）
3、连接点（joinpoint）：被拦截到的点，因为Spring只支持方法类型的连接点，所以在Spring中连接点指的就是被拦截到的方法，实际上连接点还可以是字段或者构造器（比如代码中的通知的参数）
4、切入点（pointcut）：对连接点进行拦截的定义（具体的，比如下文代码中的注解）
5、通知（advice）：所谓通知指的就是指拦截到连接点之后要执行的代码，通知分为前置、后置、异常、最终、环绕通知五类（可以是方法，也可以是对应的通知类）
6、目标对象：代理的目标对象
7、织入（weave）：将切面应用到目标对象并导致代理对象创建的过程
8、引入（introduction）：在不修改代码的前提下，引入可以在运行期为类动态地添加一些方法或字段

二、aop的简述

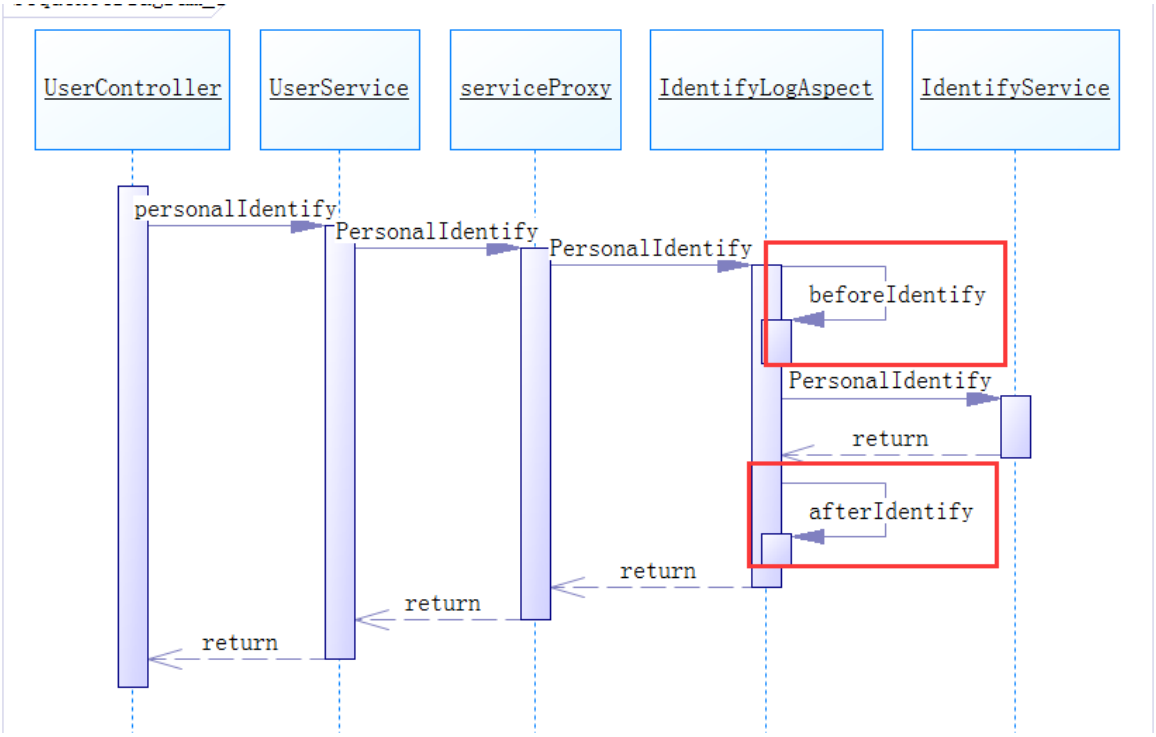
1、默认使用Java动态代理来创建AOP代理，这样就可以为任何接口实例创建代理了
2、当需要代理的类不是代理接口的时候，Spring会切换为使用CGLIB代理，也可强制使用CGLIB
AOP编程其实是很简单的事情，纵观AOP编程，程序员只需要参与三个部分：
1、定义普通业务组件
2、定义切入点，一个切入点可能横切多个业务组件
3、定义增强处理，增强处理就是在AOP框架为普通业务组件织入的处理动作
所以进行AOP编程的关键就是定义切入点和定义增强处理，一旦定义了合适的切入点和增强处理，AOP框架将自动生成AOP代理，即：代理对象的方法=增强处理+被代理对象的方法。

三、实现原理

场景是：在调用认证接方法的时候，系统需要进行记录操作次数，以及依据操作返回结果判断该操作是否属于收费，或者有效的操作。
1、在未使用aop之前的时序图逻辑是：



2、使用aop后的时序逻辑是：这样就可以针对任何调用这个认证方法进行记录了。



（注意：要让aspect生效，这些类的实例必须让spring容器管理，不然生效不了）

四、例子

废话不多说，直接上代码

由于当前模块已经集成了aop，所以我没有添加相关的依赖，不过在这里可以查看一下：

1、在**pom.xml**文件中添加：

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aspects</artifactId>
    <version>5.0.2.RELEASE</version>
</dependency>
```

2、创建一个注解，用来描述切入点，如文件：PartnerIdentifyLogAnnotation.java

```
/**
* 认证日志操作注解
*
* @author zhengfeng
* @since 2018年10月08日 14:33
*/
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface PartnerIdentifyLogAnnotation {

    String value() default "";
}
```

3、创建通知类，用来定义通知处理方法，如文件：IdentifyLogAspect.java

```
@Aspect
@Component
public class IdentifyLogAspect {

    private static final Logger LOGGER = LoggerFactory.getLogger(IdentifyLogAspect.class);
    private static final ObjectMapper OBJECT_MAPPER = new ObjectMapper();
    @Resource
    private PartnerIdentifyLogRepository partnerIdentifyLogRepository;
    @Resource
    private CommonService commonService;
```

```
@Pointcut("@annotation(com.ancun.netsign.manage.system.annotation.PartnerIdentifyLogAnnotation)")
public void operateLogPointCut() {
}
/**
 * 环绕通知：
 */
@Around(value="operateLogPointCut()")
public Object doAroundAdvice(ProceedingJoinPoint joinPoint){
    LOGGER.info("开始保存日志信息设置，环绕通知的目标方法名："+joinPoint.getSignature().getName());
    PartnerIdentifyLog partnerIdentifyLog = new PartnerIdentifyLog();
    Partner partner = null;
    try {
        LOGGER.info("-----1--------" + partnerIdentifyLog.toString());
        partnerIdentifyLog.setCreateTime(new Date());
        partnerIdentifyLog.setOperatorDefineId(PartnerIdentifyLogConstants.OPERATOR_DEFINE_ID_PERSONAL_IDENTIFY);
        partnerIdentifyLog.setOperatorDefineName(PartnerIdentifyLogConstants.OPERATOR_DEFINE_NAME_PERSONAL_IDENTIFY);
        String params = getParams(joinPoint);
        LOGGER.info("------2-------" + partnerIdentifyLog.toString());
        LOGGER.info("-------------" + params);
        partnerIdentifyLog.setStartTime(new Date());
        partnerIdentifyLog.setIdentifyApplyInfo(params);
        partnerIdentifyLog.setPartnerOperatorIp(getIpFromHttpServletRequest());
        partnerIdentifyLog.setPartnerIdentifyDesc(getContent(joinPoint));
        partnerIdentifyLog.setSystemCode(String.valueOf(PartnerIdentifyLogConstants.SYSTEM_CODE_WEB));

        ServletRequestAttributes requestAttributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();
        HttpServletRequest request;
        LOGGER.info("--------3-----" + partnerIdentifyLog.toString());
        if (requestAttributes != null) {
            request = requestAttributes.getRequest();
        } else {
            throw new NetSignException(ResponseInfo._100005);
        }
        String requestUri = request.getRequestURI();
        partnerIdentifyLog.setRequestUri(requestUri);
        partner = (Partner) request.getAttribute("partner");
        LOGGER.info("------4-------" + partnerIdentifyLog.toString());
        if (partner == null) {
            LOGGER.warn("开始认证日志到数据库中,获取接入者失败!获取当前用户的接入者");
            partner = commonService.getAncunPartner();//使用安存接入者分表
            partnerIdentifyLog.setPartnerId(partner.getId());
            User user = UserUtil.getCurrentUser();
            // user.getPartnerSet();
            String operatorAccount = user.getId() + ","+user.getUsername() + "," + user.getUniqueIdentifier();
            partnerIdentifyLog.setPartnerOperatorAccount(operatorAccount);
        }else{
            partnerIdentifyLog.setPartnerId(partner.getId());
            partnerIdentifyLog.setPartnerOperatorAccount(partner.getPartnerName());
        }

        String identifyType = request.getParameter("identifyType");
        if(org.apache.commons.lang3.StringUtils.isBlank(identifyType) || "null".equalsIgnoreCase(identifyType)){
            identifyType = String.valueOf(getOperateParameterKeyValue(joinPoint,"identifyType"));
            if(org.apache.commons.lang3.StringUtils.isBlank(identifyType) || "null".equalsIgnoreCase(identifyType)){
                String jsonObj = request.getParameter("jsonObj");
                if(org.apache.commons.lang3.StringUtils.isBlank(jsonObj)){
                    if (requestUri.contains("updateNotifyMobile")){
                        identifyType = "2";
                    }else{
                        identifyType = "0";//未知
                    }
                }else{
                    Map jsonParams = JSON.parseObject(jsonObj);
                    identifyType = String.valueOf(jsonParams.get("identifyType"));
                }
            }
        }
```

```
if(org.apache.commons.lang3.StringUtils.isBlank(identifyType) || "null".equalsIgnoreCase(identifyType)){
    String jsonObj = request.getParameter("jsonObj");
    if(org.apache.commons.lang3.StringUtils.isBlank(jsonObj)){
        identifyType = "0";//未知
    }else{
        Map paramsJson = JSON.parseObject(jsonObj);
        identifyType = String.valueOf(paramsJson.get("identifyType"));
    }
}
if (requestUri.contains("personalIdentify")||requestUri.contains("identityAuthenticationPersonal")
        || requestUri.contains("saveProxyManager")) {
    partnerIdentifyLog.setOperatorDefineId(PartnerIdentifyLogConstants.OPERATOR_DEFINE_ID_PERSONAL_IDENTIFY);
    partnerIdentifyLog.setOperatorDefineName(PartnerIdentifyLogConstants.OPERATOR_DEFINE_NAME_PERSONAL_IDENTIFY);
}else if (requestUri.contains("identityAuthenticationCompany")||requestUri.contains("identityCompanyLegal")){
    if (org.apache.commons.lang3.StringUtils.isBlank(identifyType) || "null".equalsIgnoreCase(identifyType)){
        if (requestUri.contains("identityAuthenticationCompany")){
            identifyType = "1";//企业信息+四要素
        }else{
            identifyType = "2";//企业信息+法人二要素
        }
    }
    partnerIdentifyLog.setOperatorDefineId(PartnerIdentifyLogConstants.OPERATOR_DEFINE_ID_ENTERPRISES_IDENTIFY);
    partnerIdentifyLog.setOperatorDefineName(PartnerIdentifyLogConstants.OPERATOR_DEFINE_NAME_ENTERPRISES_IDENTIFY);
}else if (requestUri.contains("peopleOCRIdentify")){
    partnerIdentifyLog.setOperatorDefineId(PartnerIdentifyLogConstants.OPERATOR_DEFINE_ID_OCR_IDENTIFY);
    partnerIdentifyLog.setOperatorDefineName(PartnerIdentifyLogConstants.OPERATOR_DEFINE_NAME_OCR_IDENTIFY);
    identifyType = "1";
}else{
    identifyType = "0";//未知
}
partnerIdentifyLog.setOperatorType(Integer.valueOf(identifyType));
Object obj = null;
LOGGER.info("------5-------" + partnerIdentifyLog.toString());
try {
    //调用原来的方法
    obj = joinPoint.proceed();
}catch (Throwable throwable){
    //异常保存
    partnerIdentifyLog.setStatus(PartnerIdentifyLogConstants.OPERATOR_IDENTIFY_THROW);
    partnerIdentifyLog.setEndTime(new Date());
    partnerIdentifyLog.setModifyTime(new Date());
    partnerIdentifyLog.setIdentifyAproveInfo(throwable.getMessage());
    partnerIdentifyLog.setJuheChargeStatus(PartnerIdentifyLogConstants.JUHE_CHARGE_STATUS_NO);
    // throw new NetSignException(NetSignResponseCode._100200);
    LOGGER.warn("认证过程中：" + throwable.getMessage());
    return NetSignResponse.error(ResponseInfo._100149);
}

LOGGER.info("-------6------" + partnerIdentifyLog.toString());
//后置保存：正常成功
LOGGER.info("-------------" + "personalIdentigysuccess" + obj);
if(obj instanceof Map){
    Map result = (Map)obj;
    if(null != result ){
        String resultCode = String.valueOf(result.get("resultCode"));
        if(org.apache.commons.lang3.StringUtils.isNotBlank(resultCode) && "0".equalsIgnoreCase(resultCode)){
            partnerIdentifyLog.setJuheChargeStatus(PartnerIdentifyLogConstants.JUHE_CHARGE_STATUS_YES);
            partnerIdentifyLog.setStatus(PartnerIdentifyLogConstants.OPERATOR_SUCCESS);
        }else{
            String chargeStatus = String.valueOf(result.get("chargeStatus"));
            if ("1".equals(resultCode) && org.apache.commons.lang3.StringUtils.isNotBlank(chargeStatus) &&
                "1".equals(chargeStatus)){
                partnerIdentifyLog.setJuheChargeStatus(PartnerIdentifyLogConstants.JUHE_CHARGE_STATUS_YES);
            }else {
                partnerIdentifyLog.setJuheChargeStatus(PartnerIdentifyLogConstants.JUHE_CHARGE_STATUS_NO);
            }
```

```
                partnerIdentifyLog.setStatus(PartnerIdentifyLogConstants.OPERATOR_IDENTIFY_FAILED);
            }
        }else{
            partnerIdentifyLog.setJuheChargeStatus(PartnerIdentifyLogConstants.JUHE_CHARGE_STATUS_NO);
            partnerIdentifyLog.setStatus(PartnerIdentifyLogConstants.OPERATOR_IDENTIFY_FAILED);
        }
    }else if(obj instanceof  NetSignResponse){
        NetSignResponse result = (NetSignResponse)obj;
        if(null != result && result.isSuccess()){
            partnerIdentifyLog.setJuheChargeStatus(PartnerIdentifyLogConstants.JUHE_CHARGE_STATUS_YES);
            partnerIdentifyLog.setStatus(PartnerIdentifyLogConstants.OPERATOR_SUCCESS);
        }else{
            if (ResponseInfo._100150.getCode()==result.getCode()){
                partnerIdentifyLog.setJuheChargeStatus(PartnerIdentifyLogConstants.JUHE_CHARGE_STATUS_YES);
            }else {
                partnerIdentifyLog.setJuheChargeStatus(PartnerIdentifyLogConstants.JUHE_CHARGE_STATUS_NO);
            }
            partnerIdentifyLog.setStatus(PartnerIdentifyLogConstants.OPERATOR_IDENTIFY_FAILED);
        }
    }else{
        partnerIdentifyLog.setJuheChargeStatus(PartnerIdentifyLogConstants.JUHE_CHARGE_STATUS_NO);
        partnerIdentifyLog.setStatus(PartnerIdentifyLogConstants.OPERATOR_IDENTIFY_FAILED);
    }

    partnerIdentifyLog.setEndTime(new Date());
    partnerIdentifyLog.setModifyTime(new Date());
    partnerIdentifyLog.setIdentifyAproveInfo(obj.toString());

    LOGGER.info("--------7-----" + partnerIdentifyLog.toString());
    return obj;
} catch (Throwable throwable) {
    if(throwable instanceof NetSignException){
        NetSignException netSignException = (NetSignException)throwable;
        if(netSignException.getCode() != ResponseInfo._100149.getCode()){
            //异常保存
            partnerIdentifyLog.setStatus(PartnerIdentifyLogConstants.OPERATOR_FAILED);
            partnerIdentifyLog.setEndTime(new Date());
            partnerIdentifyLog.setModifyTime(new Date());
            partnerIdentifyLog.setIdentifyAproveInfo(throwable.getMessage());
            // throwable.printStackTrace();
        }
    }else {
        //异常保存
        partnerIdentifyLog.setStatus(PartnerIdentifyLogConstants.OPERATOR_FAILED);
        partnerIdentifyLog.setEndTime(new Date());
        partnerIdentifyLog.setModifyTime(new Date());
        partnerIdentifyLog.setIdentifyAproveInfo(throwable.getMessage());
        // throwable.printStackTrace();
    }
    partnerIdentifyLog.setJuheChargeStatus(PartnerIdentifyLogConstants.JUHE_CHARGE_STATUS_NO);
    // throw new NetSignException(NetSignResponseCode._100020);
    LOGGER.warn("认证过程中：" + throwable.getMessage());
    return NetSignResponse.error(ResponseInfo._100142);
}finally {
    if(null == partner){
        TenantContext.setGlobalTenant();
    }else{
        String partnerTag = partner.getPartnerTag();
        if (StringUtils.isEmpty(partnerTag)) {
            TenantContext.setGlobalTenant();
        } else {
            TenantContext.setCurrentTenant(partnerTag);
        }
    }
    partnerIdentifyLogRepository.save(partnerIdentifyLog);
}
}
```

```
}
```

当然由于篇幅的问题，所以这个类省略的部分代码，只把主要的进行标红处理：

@Aspect

@Component

表示：是一个切面，以及使用spring容器管理bean

```
@Pointcut("@annotation(com.ancun.netsign.manage.system.annotation.PartnerIdentifyLogAnnotation)")
  public void operateLogPointCut() {
  }
```

定义切入点，是依据注解进行的，

当然可以进行使用配置规则方法的处理方式，配置的是：（execution(【访问权限修饰符】返回修饰符 路径.类名.方法名（参数列表））），具体百度一下资料一大堆）

@Around(value="operateLogPointCut()")

对切入点进行环绕同志

```
//调用原来的方法
obj = joinPoint.proceed();
```

使用代理调用方法

4、对identifyServiceImp.java 进行配置

```
    }
    @Transactional(propagation= Propagation.REQUIRES_NEW, noRollbackFor={Exception.class})//保证记录日志的时候不受其他事务影响,并且不回滚
    @PartnerIdentifyLogAnnotation("个人实名认证")
    @Override
    public Map<String,String> personalIdentify(String tag, PersonalIdentifyDto personalIdentifyDto) {
        Map<String,String> map = new HashMap<>( initialCapacity: 2);
        Holder<String> msg = new Holder<>();
        Holder<String> resp = new Holder<>();
        msg.setValue("");
        resp.setValue("");
        String personalIdentifyDtoString = JSON.toJSONString(personalIdentifyDto);
        IdentifyServerPrx identifyServerPrx = TarsClientHelper.stringToProxy(IdentifyServerPrx.class, IdentifyConstant.IDENTIFY_TRAS_OBJ);
        int result = identifyServerPrx.personalIdentifyAndSave(tag, personalIdentifyDtoString, resp, msg);
        if (0 == result) {
            logger.info("【{}】个人实名认证成功", personalIdentifyDtoString);
            map.put("resultCode","0");
            map.put("resultMsg",msg.getValue());
        } else {
            if(null != resp.getValue() && StringUtils.isNotBlank(resp.getValue())){
                Map identifyReponse = JSON.parseObject(resp.getValue());
```

屏幕截图 Ctrl + Alt + A
截图时隐藏当前窗口

（注意：在这里配到过一个问题，就是在调用这个方法的地方如果加入了事务Transacational处理，并且由于不是认证保错的其他地方保错，会引起日志记录保存回滚，（花掉了比较长的时间进行找问题，并解决问题），所以在这里加了一个事务处理的注解）

5、进行测试（当然前提是数据库表已经处理好，并且对应的JPA代理的hibernate映射也已经处理好），那么在操作成功或失败的情况下数据库表对应就会产生一条记录，如图：

ancun_partner_identify_... × ancun_operate_log @an... ×

文件 编辑 查看 窗口 帮助

导入向导 导出向导 筛选向导 网格查看 表单查看 备注 十六进制

id	partner_id	operator_define_id	operator_define_name	partner_operator_ip
43	1	3	ocrIdentify	192.168.0.36
42	1	3	ocrIdentify	192.168.0.36
41	1	3	ocrIdentify	192.168.0.36
40	1	3	ocrIdentify	192.168.0.36
39	1	3	ocrIdentify	192.168.0.36
38	1	3	ocrIdentify	192.168.0.36
37	1	3	ocrIdentify	192.168.0.36
36	1	3	ocrIdentify	192.168.0.36
35	1	1	personalIdentify	192.168.0.36
34	1	1	personalIdentify	192.168.0.36
33	1	1	personalIdentify	192.168.0.36
32	1	1	personalIdentify	192.168.0.36
31	1	3	ocrIdentify	192.168.0.36

五、附录：

1、由于系统增加了shiro权限框架，所以在我没有做过特殊处理的情况下，会产生这样一种情况：

controller层可以切面进入，UserService或者其调用的service一直切面不进去，但其他service类可以。
碰到这样的怪问题，我也两眼发黑，脑袋发懵，还以为是我代码那里有问题，所以检查了一遍又一遍，上周用了一天多的时间还是没有解决，后来直接把切面都放在了controller层（不保险，只是暂时使用），这一周一过来，又进行解决。
a、猜测是UserService在哪个地方没有进行spring容器管理
b、service代理有问题(UserRealm autowire 中userservce不是cglib 代理类)
最终试了一下在最开始调用UserService的地方，也就是shiro框架加载的地方加入了懒加载的注解才发现可以成功进入UserService切面
（shiro框架的filter加载机制，有待进入详细梳理）
代码如下：
ShiroConfiguration.java 文件

```
    /**
          * 安全管理配置
          *
          * @return
          */
        @Bean
        public DefaultWebSecurityManager securityManager() {
                DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
                securityManager.setRealm(jwtTokenRealm());
                securityManager.setCacheManager(shiroCacheManager());
                DefaultSubjectDAO subjectDAO = new DefaultSubjectDAO();
                DefaultSessionStorageEvaluator defaultSessionStorageEvaluator = new DefaultSessionStorageEvaluator();
                defaultSessionStorageEvaluator.setSessionStorageEnabled(false);
                subjectDAO.setSessionStorageEvaluator(defaultSessionStorageEvaluator);
                securityManager.setSubjectDAO(subjectDAO);
                SecurityUtils.setSecurityManager(securityManager);
                return securityManager;
        }


        @Bean
        public JWTTokenRealm jwtTokenRealm() {
                JWTTokenRealm jwtTokenRealm = new JWTTokenRealm();
                jwtTokenRealm.setCacheManager(shiroCacheManager());
                return jwtTokenRealm;
        }
```

JWTTokenRealm.java 文件

```
@Autowired
@Lazy
private UserService userService;
```

2、aspect中各个通知方法处理

a、配置前置通知

```
/**
* 前置通知，方法调用前被调用
* @param joinPoint
*/
@Before(value = POINT_CUT)
public void before(JoinPoint joinPoint){
    logger.info("前置通知");
    //获取目标方法的参数信息
    Object[] obj = joinPoint.getArgs();
    //AOP代理类的信息
    joinPoint.getThis();
    //代理的目标对象
    joinPoint.getTarget();
    //用的最多 通知的签名
    Signature signature = joinPoint.getSignature();
    //代理的是哪一个方法
    logger.info("代理的是哪一个方法"+signature.getName());
    //AOP代理类的名字
    logger.info("AOP代理类的名字"+signature.getDeclaringTypeName());
    //AOP代理类的类（class）信息
    signature.getDeclaringType();
    //获取RequestAttributes
    RequestAttributes requestAttributes = RequestContextHolder.getRequestAttributes();
```

```
    //从获取RequestAttributes中获取HttpServletRequest的信息
    HttpServletRequest request = (HttpServletRequest) requestAttributes.resolveReference(RequestAttributes.REFERENCE_REQUEST);
    //如果要获取Session信息的话，可以这样写：
    //HttpSession session = (HttpSession) requestAttributes.resolveReference(RequestAttributes.REFERENCE_SESSION);
    //获取请求参数
    Enumeration<String> enumeration = request.getParameterNames();
    Map<String,String> parameterMap = Maps.newHashMap();
    while (enumeration.hasMoreElements()){
        String parameter = enumeration.nextElement();
        parameterMap.put(parameter,request.getParameter(parameter));
    }
    String str = JSON.toJSONString(parameterMap);
    if(obj.length > 0) {
        logger.info("请求的参数信息为："+str);
    }
}
```

****注意：这里用到了JoinPoint和RequestContextHolder。**
1）、通过JoinPoint可以获得通知的签名信息，如目标方法名、目标方法参数信息等。
2）、通过RequestContextHolder来获取请求信息，Session信息。**

b、 **配置后置返回通知**

```
/**
* 后置返回通知
* 这里需要注意的是:
*            如果参数中的第一个参数为JoinPoint，则第二个参数为返回值的信息
*            如果参数中的第一个参数不为JoinPoint，则第一个参数为returning中对应的参数
* returning：限定了只有目标方法返回值与通知方法相应参数类型时才能执行后置返回通知，否则不执行，
*                        对于returning对应的通知方法参数为Object类型将匹配任何目标返回值
* @param joinPoint
* @param keys
*/
@AfterReturning(value = POINT_CUT,returning = "keys")
public void doAfterReturningAdvice1(JoinPoint joinPoint,Object keys){
        logger.info("第一个后置返回通知的返回值："+keys);
}

@AfterReturning(value = POINT_CUT,returning = "keys",argNames = "keys")
public void doAfterReturningAdvice2(String keys){
        logger.info("第二个后置返回通知的返回值："+keys);
}
```

c、 后置异常通知

```
/**
* 后置异常通知
*    定义一个名字，该名字用于匹配通知实现方法的一个参数名，当目标方法抛出异常返回后，将把目标方法抛出的异常传给通知方法；
*    throwing:限定了只有目标方法抛出的异常与通知方法相应参数异常类型时才能执行后置异常通知，否则不执行，
*                        对于throwing对应的通知方法参数为Throwable类型将匹配任何异常。
* @param joinPoint
* @param exception
*/
@AfterThrowing(value = POINT_CUT,throwing = "exception")
public void doAfterThrowingAdvice(JoinPoint joinPoint,Throwable exception){
        //目标方法名：
        logger.info(joinPoint.getSignature().getName());
        if(exception instanceof NullPointerException){
                logger.info("发生了空指针异常!!!!!");
        }
}
```

d、 后置最终通知

```
/**
* 后置最终通知（目标方法只要执行完了就会执行后置通知方法）
* @param joinPoint
*/
@After(value = POINT_CUT)
public void doAfterAdvice(JoinPoint joinPoint){
        logger.info("后置最终通知执行了!!!!");
}
```

e、 环绕通知

```
/**
* 环绕通知:
*      环绕通知非常强大，可以决定目标方法是否执行，什么时候执行，执行时是否需要替换方法参数，执行完毕是否需要替换返回值。
*      环绕通知第一个参数必须是org.aspectj.lang.ProceedingJoinPoint类型
*/
@Around(value = POINT_CUT)
public Object doAroundAdvice(ProceedingJoinPoint proceedingJoinPoint){
        logger.info("环绕通知的目标方法名："+proceedingJoinPoint.getSignature().getName());
        try {
               before();
               Object obj = proceedingJoinPoint.proceed();
               after();
               return obj;
        } catch (Throwable throwable) {
                throwable.printStackTrace();
        }
        return null;
}
```

参考资料：

SpringBoot @Aspect注解详情 https://blog.csdn.net/DuShiWoDeCuo/article/details/78180803
Spring事务管理与传播机制详解以及使用实例 https://blog.csdn.net/hcmony/article/details/77850183
Spring事务传播机制实例讲解 https://blog.csdn.net/seelye/article/details/40118089
spring boot 整合shiro @autowire service 无法切面的问题 http://niuqiang2008.iteye.com/blog/2424742
详解SpringBoot AOP 拦截器（Aspect注解方式） https://www.jb51.net/article/113683.htm
Spring之AOP由浅入深 https://www.cnblogs.com/zhaozihan/p/5953063.html