AOP

AOP是Spring 体系中非常重要的两个概念之一（另一个是IOC），AOP译为面向切面编程，是计算机科学中的一个设计思想，旨在通过切面技术为业务主题增加额外的通知（Advice），从而声明对切点（Pointcut）的代码块进行统一管理和装饰。

AOP术语

- Target：一个个业务组件是需要处理的目标对象。
- Joinpoint: 连接点，Target中运行织入到的位置，属性、构造器、成员方法...
- Aspect: 系统方法不直接写在Target中，而是封装在Aspect中，Aspect叫切面。
- Pointcut: 切点， 最终织入到哪些Target的哪些位置。
- Advice: 通知，实现具体的系统逻辑，具体织入什么，具体织入到哪里。通知的方式有五种：
    - @Before：通知方法会在目标方法调用之前执行
    - @After：通知方法会在目标方法调用后执行
    - @AfterReturning：通知方法会在目标方法返回后执行
    - @AfterThrowing：通知方法会在目标方法抛出异常后执行
    - @Around：把整个目标方法包裹起来，在被调用前和调用之后分别执行通知方法
- Weaving: 将Aspect的代码织入到Target中，是Spring框架提供的功能。有不同的织入方式：1、编译时织入，需要使用特殊的编译器。2、装载时织入，需要使用特殊的类装载器。3、运行时织入，需要为目标生成代理对象。

AOP的实现

AspectJ（全面的解决方案）

- AspectJ是语言级的实现，它扩展了Java语言，定义了AOP语法。
- AspectJ在编译期织入代码，它有一个专门的编译器，用来生成遵守Java字节码规范的class文件。

Spring AOP（高性价比的解决方案）

- Spring AOP使用纯Java实现，它不需要专门的编译过程，也不需要特殊的类装载器。
- Spring AOP在运行时通过代理的方式织入代码，只支持方法类型的连接点。
- Spring支持对AspectJ的集成。

Spring AOP

JDK动态代理

- Java提供的动态代理技术，可以在运行时创建接口的代理实例。
- Spring AOP默认采用此种方式，在接口的代理实例中织入代码。

CGLib动态代理

- 采用底层的字节码技术，在运行时创建子类代理实例。
- 当目标对象不存在接口时，Spring AOP会采用此种方式，在子类实例中织入代码。

实例：

```
```

```java
@Aspect
@Component
public class LogAspect {

    private final Logger logger = Logger.getLogger(LogAspect.class);

    private final ThreadLocal<Long> startTime = new ThreadLocal<>();

    private static final String PRE_TAG = "触发切面-------";

    //将controller包下的所有类和所有方法并携带所有参数都作为切点
    @Pointcut(value = "execution(* com.youxin.oauth2_client.controller.*.*
(..))")
    public void pointcut() {

    }

    @SneakyThrows
    @Before("pointcut()")
    public void doBefore(JoinPoint joinPoint) {
        startTime.set(System.currentTimeMillis());

        //收到请求，并做请求的日志记录
        ServletRequestAttributes attributes = (ServletRequestAttributes)
RequestContextHolder.getRequestAttributes();
        assert attributes != null;
        HttpServletRequest request = attributes.getRequest();

        //记录请求内容
        logger.info(PRE_TAG + "(doBefore) URL : " +
request.getRequestURL().toString());
        logger.info(PRE_TAG + "(doBefore) HTTP_METHOD : " +
request.getMethod());
        logger.info(PRE_TAG + "(doBefore) IP : " + request.getRemoteAddr());
        logger.info(PRE_TAG + "(doBefore) CLASS_METHOD : " +
joinPoint.getSignature().getDeclaringTypeName() + "." +
joinPoint.getSignature().getName());
        logger.info(PRE_TAG + "(doBefore) ARGS : " +
Arrays.toString(joinPoint.getArgs()));
    }

    @SneakyThrows
    @AfterReturning(value = "execution(*
com.youxin.oauth2_client.controller.UserController.findAllUsers(..))", returning
= "result")
    public void doAfterReturning(Object result) {
        // 处理完请求，返回内容
        logger.info(PRE_TAG + "(doAfterReturning) 查询所有用户结果 : " + result);
        logger.info(PRE_TAG + "(doAfterReturning) SPEND TIME : " +
(System.currentTimeMillis() - startTime.get()));
    }

    /*@After("execution(*
com.youxin.oauth2_client.controller.UserController.toUpdUserPage(..))")
    public void doAfterToUpdUserPage(JoinPoint joinPoint) {
        logger.info(PRE_TAG + "(doAfterToUpdUser), id为" + joinPoint.getArgs()[0]
+ "进入修改用户界面");
    }*/
```

```
    /*@After("execution(*
com.youxin.oauth2_client.controller.UserController.deleteUserById(Integer)) &&
args(id))")
    public void doAfterDeleteUserById(Integer id) {
        logger.info(PRE_TAG + "(doAfterDeleteUserById), 删除id为" + id + "用户");
    }*/

    /*@After("execution(*
com.youxin.oauth2_client.controller.UserController.insUser(..))")
    public void doAfterInsUser(JoinPoint joinPoint) {
        //获取参数
        Object[] args = joinPoint.getArgs();
        logger.info(PRE_TAG + "(doAfterInsUser) CLASS_METHOD : " +
joinPoint.getSignature().getDeclaringTypeName() + "." +
joinPoint.getSignature().getName());
        for (int i = 0; i < args.length; i ++) {
            logger.info(PRE_TAG + "args[" + i + "]====" + args[i]);
        }

        logger.info(PRE_TAG + "新增id为:" + args[0].toString() + "用户!");
    }*/

    @After("execution(* com.youxin.oauth2_client.controller.UserController.*
(..))")
    public void doAfter(JoinPoint joinPoint) {
        String method = joinPoint.getSignature().getName();
        //更新用户方法后置通知
        switch (method) {
            case "toUpdUserPage":
                //获取第一个请求参数，获取的参数是Object[] args数组类型
                logger.info(PRE_TAG + "(doAfterToUpdUser), id为" +
joinPoint.getArgs()[0] + "进入修改用户界面");
                break;
            //删除方法添加后置通知
            case "deleteUserById":
                logger.info(PRE_TAG + "(doAfterDeleteUserById), 删除id为" +
joinPoint.getArgs()[0] + "用户");
                break;
            //查询方法添加后置通知
            case "insUser":
                logger.info(PRE_TAG + "(doAfterInsUser)新增id为:" +
joinPoint.getArgs()[0].toString() + "用户!");
                break;
        }
    }
}
```