30_代码实战：基于 Spring AOP 机制，拦截 Controller 打印日志

1、开篇

上节课介绍了如何配置 logback.properties 文件对日志文件进行配置，同时还介绍了 Logback 的基本用法，并且提出了 logger 代码被重复调用的问题，导致编写大量的 logger 代码。本节课看看如何使用 Spring AOP 机制拦截 Controller 打印日志，内容包括：

- 复习 Spring AOP 概念
- 如何实现 Controller 拦截日志

2、复习 Spring AOP 概念

为了解决不重复编写日志代码，我们需要借用 Spring AOP 的帮助。这里先复习一下 Spring AOP，它是通过面向切面的方式把哪些和业务无关的功能抽象出来，很显然这里抽象出来的功能就是日志的记录。然后将这些抽象出来的日志功能根据我们的要求织入到原来的代码中。因此会用到 Join points、Pointcut 以及 Advice 的概念，如图 1 所示，Join points 就是我们需要进行切增强的方法，也就是 Controller。Pointcut 是需要切入点，比如是在 Controller 执行之前或者是执行之后或者两者都有。最后 Advice 就是我们需要增强的具体内容，对应的就是具体记录日志的代码。

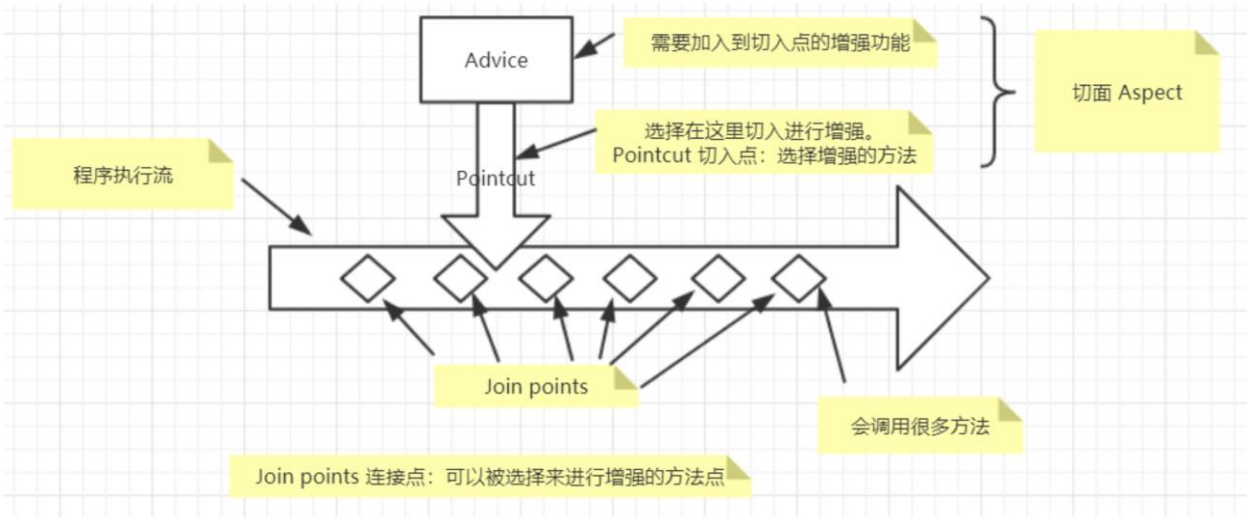

图 1 Spring AOP 概念图

3、如何实现 Controller 拦截日志

把 Spring AOP 概念过了一遍以后，我们回头看看在代码中是如何实现的。如图 2 所示，我们在已经创建好的项目中建立 ControllerLogAspect 类文件，它位于 spring/aspect 目录下面。在文件中通过 LoggerFactory 的工厂类创建 LOGGER 实体用来记录日志。需要注意的是这里有一个@Pointcut 的 annotation 标签，它是用来定义切入点信息的，在 execution 中指明了“ com.ruyuan.little.project.spring.controller.*.*(..)”这个 namesapce 下所有的类，意思是只要是在这个命名空间以内的类文件都需要进行日志拦截。

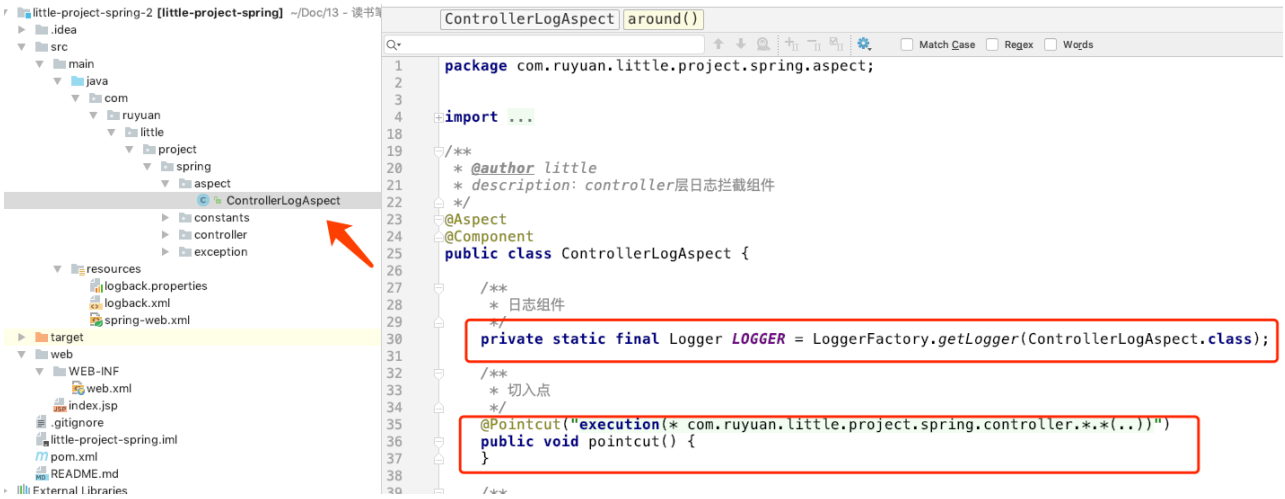

图 2 创建 ControllerLogAspect 类

如图 3 所示，在创建的类中有一个 around 的方法用来记录日志。该方法使用 @Around(”pointcut()”)的 annotation 表明使用了环绕的方式切入方法，也就是在 Controller 方法执行之前和之后都插入日志。around 方法的传入参数是 ProcessdingJoinPoint 也就是 Spring AOP 中的 Join Point，这里指需要增强的方法后文中将其称作 point，也就是 Controller 中需要记录日志的方法。

之后获取 point 的签名并且强转为方法信息，同时获取方法的参数名称。通过 point 中的 getTarget 方法获取需要执行的方法对象。通过 LOGGER.info 方法“记录处理方法”和“请求处理参数”。通过 point.processd()语句执行对应的方法，同时也在该方法的前后都记录了 startTime 和 endTime，也就是方法执行的开始和结束时间。这些信息可以记录方法执行的时长。最后，在 catch 中通过 LOGGER.error 记录方法出错的信息。

```
     * 环绕通知，在方法执行前后
     *
     * @param point 切入点
     * @return 结果
     * @throws BusinessException
     */
    @Around("pointcut()")
    public Object around(ProceedingJoinPoint point) throws BusinessException {
        // 签名信息
        Signature signature = point.getSignature();
        // 强转为方法信息
        MethodSignature methodSignature = (MethodSignature) signature;
        // 参数名称
        String[] parameterNames = methodSignature.getParameterNames();
        //执行的对象
        Object target = point.getTarget();
        LOGGER.info("请求处理方法:{}", target.getClass().getName() + StringPoolConstant.DOT + methodSignature.getMethod().getName());
        Object[] parameterValues = point.getArgs();
        //查看入参
        LOGGER.info("请求参数名:{}，请求参数值:{}", JSONObject.toJSONString(parameterNames), JSONObject.toJSONString(parameterValues));
        try {
            // 开始时间
            long startTime = System.currentTimeMillis();
            // 执行方法
            Object response = point.proceed();
            // 结束时间
            long endTime = System.currentTimeMillis();
            LOGGER.info("请求处理时间差:{}，响应结果:{}", endTime - startTime, JSON.toJSONString(response));
            return response;
        } catch (Throwable throwable) {
            LOGGER.error("执行方法:{}失败，异常信息:{}", methodSignature.getMethod().getName(), throwable);
            if (throwable instanceof BusinessException) {
                // 业务异常
                return CommonResponse.fail();
            }
            // 非业务异常，封装一层异常
            throw new BusinessException("方法 " + methodSignature.getMethod().getName() + " 执行失败");
        }
    }
}
```

图 3 日志执行方法

通过上述日志方式，只需要定义自己的日志处理类，在类中定义处理日志类的范围（Joint points）、切入点（Pointcuts）以及增强的内容（around 方法的内容），就可以定制化自己的日志类。这样在 Controller 类中执行的所有方法都遵循这个日

志类的记录方式，例如在方法执行前后记录日志，记录方法执行的时长，在方法出错时记录错误日志等等。

4、总结

本节课首先复习了 Spring AOP 的 Join point、Pointcuts、Advice 的概念，然后介绍了如何在项目中定义日志类 ControllerLogAspect，针对所有 Controller 中的方法进行日志记录。下节课我们会测试上述日志代码，看看结果如何。这里将代码给大家，下期见，拜拜。

友情提示：本章讲述代码只是部分核心代码，完整代码请查阅文末链接中代码，谢谢。