29_技术挑战：如何解决接口请求和响应日志打印导致代码重复问题？

1、开篇

上节课作为 Logback 框架的第一堂课，从 Logback 框架的基本概念入手，谈到了它的作用和依赖。然后，通过配置文件的节点给大家讲解了其需要配置的节点，包括：configuration、property、appender、logger 以及 root 节点。本节课看看如何使用 Logback 进行日志记录的，内容包括：

- Logback 的 Properties 文件配置
- Logback 在代码中使用

2、Logback 的 Properties 文件配置

上节课主要针对 Logback 的日志记录配置给大家进行讲解，对于日志文件本身在这里需要做一个补充说明。需要对不同级别的日志文件信息进行配置，例如：保存目录、日志级别、单个文件最大容量、总日志文件的容量等等。

如图 1 所示，这个是 logback.properties 文件包含的配置内容，其中通过 LOG_HOME 字段定义日志文件存放的目录，“/home/admin/little-project-spring/logs”这个就是我们服务器上日志文件存放的目录，后面的学习中会经常访问这个目录通过日志文件查看运行结果。FILE_NAME 保存的是日志文件的名字。STDOUT_LEVEL=INFO 这里定义标准输出级别为 INFO，意思是定义信息级别的日志输出。FILE_MAX_SIZE = 512M 是来定义文件的最大容量，FILE_TOTAL_SIZE =40G 这里是定义总日志文件的容量，通常来说每个日志文件到了最大容量以后，会生成新的日志文件，这里定义的是这类日志文件的总容量。FILE_HISTORY=10 意思是日志文件最多保存 10 天的时间。后面是对 ERROR 和 DEBUG 级别对日志文件的定义和 INFO 的定义大同小异，这里不再赘述。

```
#日志文件存储根路径
LOG_HOME=/home/admin/little-project-spring/logs
#日志文件名称前缀
FILE_NAME=little-project-spring

# 控制台日志打印级别
STDOUT_LEVEL=INFO

#INFO级别日志文件配置

#单个文件最大的大小
FILE_MAX_SIZE=512MB
#日志保留天数
FILE_HISTORY=10
#日志总大小
FILE_TOTAL_SIZE=40GB
# info级别日志打印开关，配置为INFO即打印，配置OFF即关闭
FILE_LEVEL=INFO

#DEBUG级别日志文件配置

FILE_DEBUG_MAX_SIZE=512MB
FILE_DEBUG_HISTORY=10
FILE_DEBUG_TOTAL_SIZE=40GB
# debug级别日志打印开关，配置为debug即打印debug级别的日志，配置OFF即关闭
FILE_DEBUG_LEVEL=DEBUG

#ERROR级别日志文件配置

FILE_ERROR_MAX_SIZE=512MB
FILE_ERROR_HISTORY=10
FILE_ERROR_TOTAL_SIZE=10GB
# error级别日志打印开关，配置为error即打印error级别的日志，配置OFF即关闭
FILE_ERROR_LEVEL=OFF
```

图 1 日志文件配置信息

3、Logback 在代码中使用

编写好记录日志信息的配置 logback.xml 和日志文件存放信息的配置 logback.properties 文件以后，就可以记录开始日志代码的编写了。如图 2 所示，直接使用 LoggerFactory 工厂类中的 getLogger 方法传入需要记录日志的类名，此时返回的 Logger 类型的对象 logger 就可以记录日志信息了。在 main 方法中通过 logger.trace、logger.debug、logger.info、logger.warn、logger.error 等方法，传入字符串的方式就可以记录日志信息了。如果说再程序中有异常抛出也可以在 catch 中加入日志代码来捕捉异常信息。

```
public class Main {

    private static final Logger logger = LoggerFactory.getLogger(Main.class);

    public static void main(String[] args) {
        logger.trace("trace log");
        logger.debug("debug log");
        logger.info("info log");
        logger.warn("warn log");
        logger.error("error");

        try{
            throw new IllegalStateException("throw a new exception");
        }catch(Exception e){
            logger.error(e.getMessage(),e);
        }
    }
```

图 2 使用 LoggerFacotry 工厂类记录日志

同样如果我们想在业务方法中写入日志也可以遵照上面的方法进行，如图 3 所示，依旧通过 LoggerFactory 的 getLogger 方法获取 Logger 对象，并且在业务代码的执行前后都可以加入 logger 的记录语句，从而记录日志，同时可以在 catch 中捕捉对应的错误日志。

```
public void Method1(string input1 ,string input2){

    private Logger logger = LoggerFactory.getLogger(Main.class);

    try{
        logger.trace("业务代码开始");
        //业务代码
        logger.trace("业务代码结束");
    }catch(Exception e){
        logger.error(e.getMessage(),e);
    }

}
```

图 3 在业务方法中使用 logger

上面介绍了 Logback 记录日志的方式，总体来说比较简单，配置好以后直接使用工厂类生成日志的实体然后在需要的地方记录日志就好了。但是，在实际的业务场景中，需要记录日志的地方非常的多，这样会造成大量的使用 logger 在程序的各个角落重复地调用 logger 中的 trace、info、debug、error 等方法。

这样会导致产生大量重复的日志代码，即便我们可以将这些日志代码进行封装，让它们更方便地被调用，但是还是免不了重复使用它们。这里就需要用到上周学到的 Spring AOP 的知识，基于 Spring AOP 拦截的方式记录日志。这里先按下不表，我们下节课再娓娓道来。

4、总结

本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。那么有什么办法能让我们从大量的日志代码中解脱呢？这就是下周要介绍的 Spring AOP 拦截日志方式。下期见，拜拜。