# 面向切面编程(AOP)

## AOP 统一记录 HTTP 请求日志

* 实现思路同样适用于非 HTTP 请求类型日志记录.
* 本文需求是: 通过日志记录 Controller 中的请求.
* 本文不对日志相关的配置作说明.
* 完整示例可以直接看参考.

### 环境

* apache-tomcat-8.5.11
* jdk1.8.0\_121 (1.7 也可以)

### 配置

maven pom.xml 配置:

<dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjrt</artifactId>  
 <version>1.8.4</version>  
</dependency>  
<dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjweaver</artifactId>  
 <version>1.8.4</version>  
</dependency>  
<dependency>  
 <groupId>cglib</groupId>  
 <artifactId>cglib</artifactId>  
 <version>2.2</version>  
</dependency>

dispatcher-servlet.xml 配置:

<context:component-scan base-package="com.xx.xxxx" />  
<aop:aspectj-autoproxy />

### AOP 日志记录实现

import org.apache.log4j.Logger;  
import org.aspectj.lang.JoinPoint;  
import org.aspectj.lang.annotation.Aspect;  
import org.aspectj.lang.annotation.Pointcut;  
import org.aspectj.lang.annotation.AfterReturning;  
import org.aspectj.lang.annotation.Before;  
import org.springframework.core.annotation.Order;  
import org.springframework.stereotype.Component;  
import org.springframework.web.context.request.RequestContextHolder;  
import org.springframework.web.context.request.ServletRequestAttributes;  
  
import javax.servlet.http.HttpServletRequest;  
import java.util.Arrays;  
  
/\*\*  
\* Order(3) 制定 Aspect 处理顺序, 数值越小, 优先级越高  
\*/  
@Aspect()  
@Order(3)  
@Component()  
public class HttpLogAspect {  
  
 private Logger logger = Logger.getLogger(getClass());  
 private ThreadLocal<Long> startTime = new ThreadLocal<Long>(); // 记录请求与响应花费的时间  
  
 @Pointcut("within(@org.springframework.stereotype.Controller \*)")  
 public void controller() {}  
  
 @Pointcut("execution(\* \*.\*(..))")  
 protected void allMethod() {}  
  
 /\*\*  
 \* 执行前  
 \* 记录 HTTP 请求详细  
 \* @param joinPoint joinPoint  
 \*/  
 @Before("controller() && allMethod()")  
 public void logBefore(JoinPoint joinPoint) {  
 // 开始计时  
 startTime.set(System.currentTimeMillis());  
  
 logger.info("\*\* START HTTP REQUEST \*\*");  
  
 ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();  
 HttpServletRequest request = attributes.getRequest();  
  
 // 记录类名及方法名  
 logger.info("HTTP\_CLASS\_METHOD : " + joinPoint.getSignature().getDeclaringTypeName() + "."  
 + joinPoint.getSignature().getName());  
 // 记录请求参数  
 logger.info("HTTP\_ARGUMENTS : " + Arrays.toString(joinPoint.getArgs()));  
  
 if (null != request) {  
 // 记录请求地址  
 logger.info("HTTP\_REQUEST\_URL : " + request.getRequestURL().toString());  
 // 记录请求方法  
 logger.info("HTTP\_METHOD : " + request.getMethod());  
 // 记录请求 IP  
 logger.info("HTTP\_REQUEST\_IP : " + request.getRemoteAddr());  
 }  
 }  
  
 /\*\*  
 \* 执行后  
 \* 请求结束, 记录返回内容  
 \* @param result 响应内容  
 \*/  
 @AfterReturning(pointcut = "controller() && allMethod()", returning = "result")  
 public void logAfterReturning(Object result) {  
 logger.info("HTTP\_RESPONSE : " + result);  
 // 结束计时  
 logger.info("HTTP\_SPEND\_TIME : " + (System.currentTimeMillis() - startTime.get()) + " ms");  
 logger.info("\*\* END HTTP REQUEST \*\*");  
 }  
  
}

# maven 使用相关

## 不同环境(开发,上线)配置切换

在编译时使用 maven 命令参数打包不同环境下的配置文件, 比如 src/main/resources/prod 和 src/main/resources/dev 文件夹下分别是线上环境和开发环境的配置文件. maven pom.xml 配置文件部分配置如下.

<build>  
 <resources>  
 <resource>  
 <directory>src/main/resources</directory>  
 <!-- 资源根目录排除各环境的配置，使用单独的资源目录来指定 -->  
 <excludes>  
 <exclude>prod/\*</exclude>  
 <exclude>dev/\*</exclude>  
 </excludes>  
 </resource>  
 <resource>  
 <directory>src/main/resources/${profiles.active}</directory>  
 </resource>  
 </resources>  
</build>  
<profiles>  
 <profile>  
 <!-- 开发环境 -->  
 <id>dev</id>  
 <properties>  
 <profiles.active>dev</profiles.active>  
 </properties>  
 <activation>  
 <activeByDefault>true</activeByDefault>  
 </activation>  
 </profile>  
 <profile>  
 <!-- 生产环境 -->  
 <id>prod</id>  
 <properties>  
 <profiles.active>prod</profiles.active>  
 </properties>  
 </profile>  
</profiles>

打包时通过 mvn clean package -P prod 实现线上环境配置打包, mvn clean package -P dev 实现开发环境配置打包.

## 本地 JAR 文件引入

**本地 JAR 文件加入到本地 maven 库**

<!--build>plugins 下添加-->  
<!--安装本地 jar 包-->  
<!-- https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-install-plugin -->  
<plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-install-plugin</artifactId>  
 <version>2.5.2</version>  
 <configuration>  
 <groupId>com.mycompany</groupId>  
 <artifactId>myproject</artifactId>  
 <version>1.0</version>  
 <packaging>jar</packaging>  
 <generatePom>true</generatePom>  
 <file>${basedir}/src/main/webapp/WEB-INF/lib/myjar.jar</file>  
 </configuration>  
 <executions>  
 <execution>  
 <id>install-jar-lib</id>  
 <goals>  
 <goal>install-file</goal>  
 </goals>  
 <phase>validate</phase>  
 </execution>  
 </executions>  
</plugin>

**依赖添加**

<!--dependencies 下添加-->  
<dependency>  
 <groupId>com.mycompany</groupId>  
 <artifactId>myproject</artifactId>  
 <version>1.0</version>  
</dependency>

**打包前先执行** mvn install:install-file 或者 mvn validate. 或者一行命令执行 mvn validate & mvn clean package -P dev.

* [Maven安装jar文件到本地仓库](https://www.cnblogs.com/xguo/archive/2013/06/04/3117894.html)
* [Install local jar dependency as part of the lifecycle, before Maven attempts to resolve it](https://stackoverflow.com/questions/26618192/install-local-jar-dependency-as-part-of-the-lifecycle-before-maven-attempts-to)