
回到首页

芋道源码 —— 知识星球

我是一段不羁的公告！
记得给芳芳这 3 个项目加油，添加一个 STAR 噢。
https://github.com/YunaiV/SpringBoot-Labs
https://github.com/YunaiV/onemall
https://github.com/YunaiV/ruoyi-vue-pro
2018-11-14
Dubbo

精尽 Dubbo 源码分析 —— 过滤器（三）之 AccessLogFilter

本文基于 Dubbo 2.6.1 版本，望知悉。

1. 概述

本文分享记录访问日志的过滤器 AccessLogFilter ，需要在 <dubbo:protocol /> 或 <dubbo:provider /> 或 <dubbo:service /> 中，设置 "accesslog" 配置项开启。有两种配置项选择：

【配置方式一】 true ：将向日志组件 logger 中输出访问日志。
【配置方式二】访问日志文件路径：直接把访问日志输出到指定文件。

2. AccessLogFilter

com.alibaba.dubbo.rpc.filter.AccessLogFilter ，实现 Filter 接口，记录服务的访问日志的过滤器实现类。

2.1 构造方法

```
/**
 * 访问日志在 {@link LoggerFactory} 中的日志名
 */
private static final String ACCESS_LOG_KEY = "dubbo.accesslog";

/**
 * 访问日志的文件后缀
 */
private static final String FILE_DATE_FORMAT = "yyyyMMdd";
/**
 * 日历的时间格式化
 */
private static final String MESSAGE_DATE_FORMAT = "yyyy-MM-dd HH:mm:ss";
/**
 * 队列大小，即 {@link #logQueue} 值的大小
```

```
 */
private static final int LOG_MAX_BUFFER = 5000;
/**
 * 日志输出频率，单位：毫秒。仅适用于 {@link #logFuture}
 */
private static final long LOG_OUTPUT_INTERVAL = 5000;

/**
 * 日志队列
 *
 * key：访问日志名
 * value：日志集合
 */
private final ConcurrentMap<String, Set<String>> logQueue = new ConcurrentHashMap<String, Set<String>>();
/**
 * 定时任务线程池
 */
private final ScheduledExecutorService logScheduled = Executors.newScheduledThreadPool(2, new NamedThreadFactory("Dub
/**
 * 记录日志任务
 */
private volatile ScheduledFuture<?> logFuture = null;
```

【配置方式一】

- ACCESS_LOG_KEY

【配置方式二】

- 文件相关：
  - FILE_DATE_FORMAT
  - MESSAGE_DATE_FORMAT
- 队列相关：
  - logQueue
  - LOG_MAX_BUFFER
- 任务相关：
  - logScheduled
  - LOG_OUTPUT_INTERVAL
  - logFuture
- 日志流向为：logMessage => 队列 => 任务 => 文件。

2.2 invoke

```
 1: @Override
 2: @SuppressWarnings("Duplicates")
 3: public Result invoke(Invoker<?> invoker, Invocation inv) throws RpcException {
 4:     try {
 5:         // 记录访问日志的文件名
 6:         String accesslog = invoker.getUrl().getParameter(Constants.ACCESS_LOG_KEY);
 7:         if (ConfigUtils.isNotEmpty(accesslog)) {
 8:             // 服务的名字、版本、分组信息
 9:             RpcContext context = RpcContext.getContext();
10:             String serviceName = invoker.getInterface().getName();
11:             String version = invoker.getUrl().getParameter(Constants.VERSION_KEY);
12:             String group = invoker.getUrl().getParameter(Constants.GROUP_KEY);
13:             // 拼接日志内容
14:             StringBuilder sn = new StringBuilder();
```

```
15:             sn.append("[").append(new SimpleDateFormat(MESSAGE_DATE_FORMAT).format(new Date())).append("] ") // 时
16:                     .append(context.getRemoteHost()).append(":").append(context.getRemotePort()) // 调用方地址
17:                     .append(" -> ").append(context.getLocalHost()).append(":").append(context.getLocalPort()) //
18:                     .append(" - ");
19:             if (null != group && group.length() > 0) { // 分组
20:                 sn.append(group).append("/");
21:             }
22:             sn.append(serviceName); // 服务名
23:             if (null != version && version.length() > 0) { // 版本
24:                 sn.append(":").append(version);
25:             }
26:             sn.append(" ");
27:             sn.append(inv.getMethodName()); // 方法名
28:             sn.append("(");
29:             Class<?>[] types = inv.getParameterTypes(); // 参数类型
30:             if (types != null && types.length > 0) {
31:                 boolean first = true;
32:                 for (Class<?> type : types) {
33:                     if (first) {
34:                         first = false;
35:                     } else {
36:                         sn.append(",");
37:                     }
38:                     sn.append(type.getName());
39:                 }
40:             }
41:             sn.append(") ");
42:             Object[] args = inv.getArguments(); // 参数值
43:             if (args != null && args.length > 0) {
44:                 sn.append(JSON.toJSONString(args));
45:             }
46:             String msg = sn.toString();
47:             // 【方式一】使用日志组件，例如 Log4j 等写
48:             if (ConfigUtils.isDefault(accesslog)) {
49:                 LoggerFactory.getLogger(ACCESS_LOG_KEY + "." + invoker.getInterface().getName()).info(msg);
50:             // 【方式二】异步输出到指定文件
51:             } else {
52:                 log(accesslog, msg);
53:             }
54:         }
55:     } catch (Throwable t) {
56:         logger.warn("Exception in AcessLogFilter of service(" + invoker + " -> " + inv + ")", t);
57:     }
58:     // 服务调用
59:     return invoker.invoke(inv);
60: }
```

第 6 行：获得访问日志的配置项。
第 8 至 12 行：获得服务的名字、版本、分组信息。

第 13 至 46 行：拼接日志的内容。例子如下：

```
[2018-04-14 11:57:58] 192.168.3.17:57207 -> 192.168.3.17:20880 - com.alibaba.dubbo.demo.DemoService say01(java.
```

第 47 至 49 行：调用 `ConfigUtils#isDefault(value)` 方法，判断是否使用日志组件记录日志。例如 Log4J 等等。详细参见 《Dubbo 用户指南 —— 日志适配》 。

第 50 至 53 行：调用 #log(accesslog, logMessage) 方法，添加日志内容到日志队列。代码如下：

```
    /**
     * 初始化任务
     */
    private void init() {
        if (logFuture == null) {
            synchronized (logScheduled) {
                if (logFuture == null) { // 双重锁，避免重复初始化
                    logFuture = logScheduled.scheduleWithFixedDelay(new LogTask(), LOG_OUTPUT_INTERVAL, LOG_OUTPUT_
                }
            }
        }
    }

    /**
     * 添加日志内容到日志队列
     *
     * @param accesslog 日志文件
     * @param logmessage 日志内容
     */
    private void log(String accesslog, String logmessage) {
        // 初始化
        init();
        // 获得队列，以文件名为 Key
        Set<String> logSet = logQueue.get(accesslog);
        if (logSet == null) {
            logQueue.putIfAbsent(accesslog, new ConcurrentHashSet<String>());
            logSet = logQueue.get(accesslog);
        }
        // 若未超过队列大小，添加到队列中
        if (logSet.size() < LOG_MAX_BUFFER) {
            logSet.add(logmessage);
        }
    }
```

2.3 LogTask

LogTask 是 AccessLogFilter 的内部类

```
 1: /**
 2:  * 日志任务
 3:  */
 4: private class LogTask implements Runnable {
 5:
 6:     @Override
 7:     public void run() {
 8:         try {
 9:             if (logQueue.size() > 0) {
10:                 for (Map.Entry<String, Set<String>> entry : logQueue.entrySet()) {
11:                     try {
12:                         String accesslog = entry.getKey();
13:                         Set<String> logSet = entry.getValue();
14:                         // 获得日志文件
15:                         File file = new File(accesslog);
```

```
16:                         File dir = file.getParentFile();
17:                         if (null != dir && !dir.exists()) {
18:                             dir.mkdirs();
19:                         }
20:                         if (logger.isDebugEnabled()) {
21:                             logger.debug("Append log to " + accesslog);
22:                         }
23:                         // 归档历史日志文件，例如： `accesslog` => `access.20181023`
24:                         if (file.exists()) {
25:                             String now = new SimpleDateFormat(FILE_DATE_FORMAT).format(new Date());
26:                             String last = new SimpleDateFormat(FILE_DATE_FORMAT).format(new Date(file.lastModifie
27:                             if (!now.equals(last)) {
28:                                 File archive = new File(file.getAbsolutePath() + "." + last);
29:                                 file.renameTo(archive);
30:                             }
31:                         }
32:                         // 输出日志到指定文件
33:                         FileWriter writer = new FileWriter(file, true);
34:                         try {
35:                             for (Iterator<String> iterator = logSet.iterator(); iterator.hasNext(); iterator.remo
36:                                 writer.write(iterator.next()); // 写入一行日志
37:                                 writer.write("\r\n"); // 换行
38:                             }
39:                             writer.flush(); // 刷盘
40:                         } finally {
41:                             writer.close(); // 关闭
42:                         }
43:                     } catch (Exception e) {
44:                         logger.error(e.getMessage(), e);
45:                     }
46:                 }
47:             }
48:         } catch (Exception e) {
49:             logger.error(e.getMessage(), e);
50:         }
51:     }
52:
53: }
```

从 `#init()` 方法，我们可以看到，LogTask 每 5000 （ `LOG_OUTPUT_INTERVAL` ） 毫秒，执行一次。
第 9 至 10 行：循环日志队列 `logQueue` 。注意，日志集合使用了 ConcurrentHashSet ，所以会有一定的乱序，在最终输出到指定文件后。
第 14 至 22 行：获得日志文件。
第 23 至 31 行：归档历史日志文件，例如：`accesslog => access.20181023` 。
- 注意，因为是按照文件最后修改时间，所以极端情况（写着写着到了第二天），那么就不会归档了。

第 32 至 42 行：输出日志到指定文件。

666. 彩蛋

实际使用时，推荐使用 `accesslog="true"` 配置项。

文章目录

回到首页