# 业务系统日志管理

|  |  |  |  |
| --- | --- | --- | --- |
| 文档版本号： | V1.0 | 文档编号： |  |
| 文档密级： |  | 归属部门/项目： |  |
| 产品名： |  | 子系统名： |  |
| 编写人： | 雷少杰 | 编写日期： | 2016-11-15 |

目录

[业务系统日志管理 1](#_Toc467050815)

[1 组件及规范 3](#_Toc467050816)

[1.1 日志文件规范 3](#_Toc467050817)

[1.2 日志输出组件 3](#_Toc467050818)

[1.3 代码编写规范 3](#_Toc467050819)

[1.4 日志内容规范 3](#_Toc467050820)

[1.4.1 日志格式 3](#_Toc467050821)

[1.4.2 说明 3](#_Toc467050822)

[2 统一采集 4](#_Toc467050823)

[2.1 实现机制 4](#_Toc467050824)

[2.2 客户端配置 4](#_Toc467050825)

[2.2.1 使用属性文件配置 4](#_Toc467050826)

[2.2.2 使用XML文件配置 4](#_Toc467050827)

[2.3 服务器启动 5](#_Toc467050828)

[2.3.1 建议 5](#_Toc467050829)

[2.3.2 配置 5](#_Toc467050830)

[2.4 启动脚本 5](#_Toc467050831)

[2.5 源代码 6](#_Toc467050832)

[2.5.1 Log4jSocketServer 6](#_Toc467050833)

[2.5.2 AspireSocketNode 7](#_Toc467050834)

[2.6 日志输出效果 8](#_Toc467050835)

[3 其他 8](#_Toc467050836)

[3.1 配置文件自动检测更新 8](#_Toc467050837)

[3.1.1 代码 8](#_Toc467050838)

[3.1.2 配置 10](#_Toc467050839)

[3.2 Kafka日志输出 10](#_Toc467050840)

# 组件及规范

## 日志文件规范

|  |  |  |
| --- | --- | --- |
| 文件名 | 类型 | 示例 |
| 业务名/服务名.log | 正在写入的日志文件名 | auditService.log |
| 业务名/服务名.log\_yyyy-MM-dd-HH.log | 历史日志文件名 | auditService.log\_2016-11-14-17.log |

## 日志输出组件

统一采用SLF4J+LOG4j输出业务文本日志。

## 日志级别定义

error - 其他错误运行期错误；

warn - 警告信息，如程序调用了一个即将作废的接口，接口的不当使用，运行状态不是期望的但仍可继续处理等；

info - 有意义的事件信息，如程序启动，关闭事件，收到请求事件等；

debug - 调试信息，可记录详细的业务处理到哪一步了，以及当前的变量状态；

trace - 更详细的跟踪信息；

## 代码编写规范

采用slf4j来输出日志，并且避免采用拼接字符串的形式。

|  |
| --- |
| **import** org.slf4j.Logger;  **import** org.slf4j.LoggerFactory;  #静态final类型  **private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(AuditEngineApp.**class**); |

日志输出

* 正确方式

LOGGER.info("用户[{}]登录失败,没有找到对应的用户。",name);

* 不正确的方式

LOGGER.info("用户[" + name + "]登录失败,没有找到对应的用户。");

## 日志内容规范

### 日志格式

Ip|日期时间|线程名|级别|类名:代码行|业务信息%n

### 说明

Ip：服务器IP地址

日期时间：yyyy-MM-dd HH:mm:ss,SSS

线程名：当前执行的线程号

级别：日志级别：DEBUG，INFO，WARN，ERROR

类名：打印日志的类名

代码行：打印日志的代码行

业务信息：输出的内容

%n：换行。

# 统一采集

## 实现机制

采用Log4j自带的SocketAppender输出到统一的远程服务器。

## 客户端配置

客户端即程序执行位置，包括后台服务、WEB程序等业务日志发生的地方，可以有多个。

建议一种类型的业务日志保存在一个位置中。

日志服务器出现故障或者重新启动时不会对正常业务造成影响。

### 使用属性文件配置

|  |
| --- |
| log4j.rootLogger=DEBUG, server  log4j.appender.server=org.apache.log4j.net.SocketAppender  log4j.appender.server.Port=4360  log4j.appender.server.locationInfo=true  log4j.appender.server.RemoteHost=10.12.3.26  #Define any connection delay before attempting to reconnect  log4j.appender.server.ReconnectionDelay=10000 |

### 使用XML文件配置

|  |
| --- |
| <appender name="server" class="org.apache.log4j.net.SocketAppender">  <param name="Port" value="8080" />  <param name="locationInfo" value="true" />  <param name="RemoteHost" value="10.12.3.26" />  <param name="ReconnectionDelay" value="10000" />  </appender>  <root>  <priority value ="debug"/>  <appender-ref ref="server "/>  </root> |

## 服务器启动

### 建议

一个服务类型启动一个收集服务端口，避免日志交叉，对分析造成干扰。

### 配置



## 启动脚本

可以启动多个端口，每个端口绑定一个配置文件，输出到指定日志文件。

|  |
| --- |
| export CLASSPATH=$CLASSPATH:./config:./lib/\*:./tlib/\*  export JAVA\_OPTS=" -server -Xmn128m -Xms1024m -Xmx1024m -Xss512K -XX:+DisableExplicitGC -XX:+UseConcMarkSweepGC -XX:+CMSParallelRemarkEnabled -XX:+UseCMSCompactAtFullCollection -XX:LargePageSizeInBytes=128m"  export CMD4560="com.aspire.msp.tools.log4j.server.LogServer 4560 config/log4j.properties ."  nohup java ${JAVA\_OPTS} ${CMD4560} & |

## 源代码

修改扩展的地方已经标红

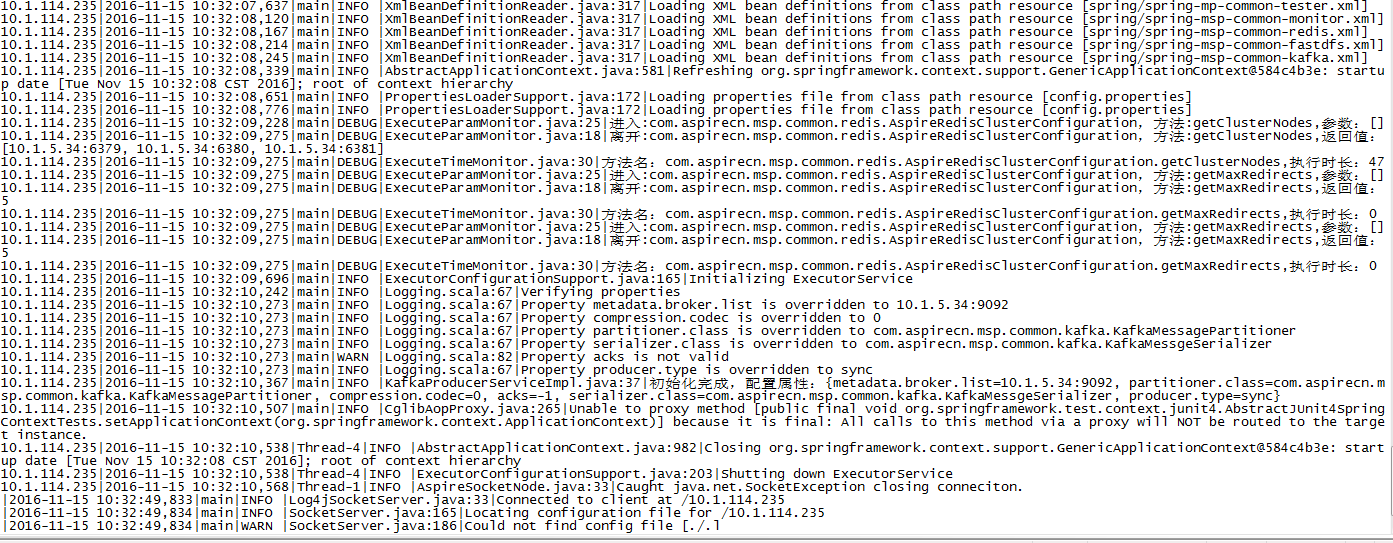
### Log4jSocketServer

|  |
| --- |
| package org.apache.log4j.net;  import java.io.File;  import java.io.IOException;  import java.net.InetAddress;  import java.net.ServerSocket;  import java.net.Socket;  import org.apache.log4j.spi.LoggerRepository;  public class Log4jSocketServer extends SocketServer {  public Log4jSocketServer(File directory) {  super(directory);  }  public static SocketServer getServer() {  return server;  }  public static void main(String argv[]) {  if (argv.length == 3)  init(argv[0], argv[1], argv[2]);  else  usage("Wrong number of arguments.");  ServerSocket serverSocket = null;  try {  cat.info("Listening on port " + port);  serverSocket = new ServerSocket(port);  while (true) {  cat.info("Waiting to accept a new client.");  Socket socket = serverSocket.accept();  InetAddress inetAddress = socket.getInetAddress();  cat.info("Connected to client at " + inetAddress);  LoggerRepository h = (LoggerRepository) server.hierarchyMap  .get(inetAddress);  if (h == null) {  h = server.configureHierarchy(inetAddress);  }  cat.info("Starting new socket node.");  new Thread(new AspireSocketNode(socket, h)).start();  }  } catch (Exception e) {  e.printStackTrace();  } finally {  if (serverSocket != null) {  try {  serverSocket.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  }  } |

### AspireSocketNode

|  |
| --- |
| package org.apache.log4j.net;  import java.io.IOException;  import java.net.Socket;  import org.apache.log4j.Logger;  import org.apache.log4j.MDC;  import org.apache.log4j.spi.LoggerRepository;  import org.apache.log4j.spi.LoggingEvent;  public class AspireSocketNode extends SocketNode {  public AspireSocketNode(Socket socket, LoggerRepository hierarchy) {  super(socket, hierarchy);  }  public void run() {  LoggingEvent event;  Logger remoteLogger;  MDC.put("remoteAddr", socket.getInetAddress().getHostAddress());  try {  while (true) {  event = (LoggingEvent) ois.readObject();  remoteLogger = hierarchy.getLogger(event.getLoggerName());  if (event.getLevel().isGreaterOrEqual(  remoteLogger.getEffectiveLevel())) {  remoteLogger.callAppenders(event);  }  }  } catch (java.io.EOFException e) {  logger.info("Caught java.io.EOFException closing conneciton.");  } catch (java.net.SocketException e) {  logger.info("Caught java.net.SocketException closing conneciton.");  } catch (IOException e) {  logger.info("Caught java.io.IOException: " + e);  logger.info("Closing connection.");  } catch (Exception e) {  logger.error("Unexpected exception. Closing conneciton.", e);  }  try {  ois.close();  } catch (Exception e) {  logger.info("Could not close connection.", e);  }  }  } |

## 日志输出效果



# 其他

## 配置文件自动检测更新

### 代码

|  |
| --- |
| package com.aspirecn.msp.common.log4j;  import java.net.URL;  import org.apache.log4j.PropertyConfigurator;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  public class Log4jConfigReload {  private int interval = 60 \* 1000;  private static final Logger logger = LoggerFactory  .getLogger(Log4jConfigReload.class);  public Log4jConfigReload() {  this.loadConfig();  }  /\*\*  \* log4j日志自动加载  \* @param interval  \*自动加载时间(ms)  \*/  public Log4jConfigReload(int interval) {  this.interval = interval;  this.loadConfig();  }  public void loadConfig() {  URL url = Log4jConfigReload.class.getClassLoader().getResource(  "log4j.properties");  if (url == null) {  url = Log4jConfigReload.class.getClassLoader().getResource(  "log4j.xml");  }  if (url != null) {  String log4jPath = url.getPath();  // 间隔特定时间，检测文件是否修改，自动重新读取配置  PropertyConfigurator.configureAndWatch(log4jPath, this.interval);  logger.debug("log4j配置文件定时检测启动成功，文件路径：{}，间隔时间：{} ", log4jPath,  interval);  } else {  logger.debug("log4j配置文件定时检测启动失败，未找到配置文件 ");  }  }  } |

### 配置

|  |
| --- |
| <bean id="log4jConfigReload" class="com.aspirecn.msp.common.log4j.Log4jConfigReload" >  <!-- 可以不设置，默认为60秒，单位为毫秒-->  <constructor-arg index="0" value=”60000”/>  </bean> |

## Kafka日志输出

|  |
| --- |
| log4j.appender.kafka=kafka.producer.KafkaLog4jAppender  log4j.appender.kafka.topic=kafka  # multiple brokers are separated by comma ",".  log4j.appender.kafka.brokerList=localhost:9092, localhost:9093, localhost:9094, localhost:9095  log4j.appender.kafka.compressionType=none  log4j.appender.kafka.syncSend=true  log4j.appender.kafka.layout=org.apache.log4j.PatternLayout  log4j.appender.kafka.layout.ConversionPattern=%d [%-5p] [%t] - [%l] %m%n |