Author：Shine

Version：9.0.2

一、引言

1.1 日志介绍

> 用于记录系统中发生的各种事件。记录的位置常见的有： 控制台、磁盘文件等

1.2 日志级别

> 日志级别从低到高：
>
> TRACE、DEBUG、INFO、WARN、ERROR、FATAL

1.3 日志作用

> - 通过日志观察、分析项目的运行情况 (项目维护)
> - 通过日志分析用户的使用情况 (大数据分析)
> - ...

二、解决方案1

2.1 Log4j+Commons-Logging

2.1.1 导入依赖

> 项目中添加 Log4j和Commons-Logging的依赖

```
<!-- https://mvnrepository.com/artifact/log4j/log4j -->
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
<!-- https://mvnrepository.com/artifact/commons-logging/commons-logging -->
<dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.2</version>
</dependency>
```

2.1.2 基本使用

> 基本API

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.junit.Test;
```

```java
public class HelloLog {
  // 需要输出日志的类，可以创建一个log属性
    Log log = LogFactory.getLog(HelloLog.class);
    @Test
    public void test1(){
        log.trace("hello trace");
        log.debug("hello debug");
        log.info("hello info");
        log.warn("hello warn");
        log.error("hello error");
        log.fatal("hello fatal");
    }
}
```

2.1.3 配置信息

> 定义配置文件 **log4j.xml**

占位符	描述
%p	输出优先级，即DEBUG，INFO，WARN，ERROR，FATAL
%r	输出自应用启动到输出该log信息耗费的毫秒数
%c	输出所在类的全名
%t	输出产生该日志事件的线程名
%n	输出一个回车换行符
%d	输出日志时间点的日期或时间，默认格式为ISO8601，也可以在其后指定格式，比如：%d{yyyy-MM-dd HH:mm:ss,SSS}，输出类似：2002-10-18 22:10:28,921
%l	输出日志事件的发生位置，包括类名、发生的线程，以及在代码中的行数。举例：Testlo4.main(TestLog4.java:10)

```xml
<?xml version="1.0"  encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration PUBLIC "-//LOGGER"
        "http://org/apache/log4j/xml/log4j.dtd">
<log4j:configuration>

    <!-- org.apache.log4j.ConsoleAppender 输出到控制台 -->
    <appender name="myConsole" class="org.apache.log4j.ConsoleAppender">
        <!--输出格式-->
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern"
                   value="%-d{yyyy-MM-dd HH:mm:ss,SSS} [%c]-[%p] %m%n"/>
        </layout>
    </appender>
<!-- 输出到文件 -->
<appender name="myFile1" class="org.apache.log4j.RollingFileAppender">
        <param name="File" value="d:/log/hello.log"/><!--文件位置-->
        <param name="Append" value="true"/><!--是否选择追加-->
        <param name="MaxFileSize" value="1kb"/><!--文件最大字节数-->
        <param name="MaxBackupIndex" value="2" /><!--新文件数量-->
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%-d{yyyy-MM-dd HH:mm:ss,SSS} [%c]-[%p] %m%n" />
        </layout>
    </appender>
    <!-- 输出到文件 -->
    <appender name="myFile2" class="org.apache.log4j.DailyRollingFileAppender">
        <param name="File" value="d:/log/hello2.log"/><!--文件位置-->
        <param name="Append" value="true"/><!--是否选择追加-->
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern"
                   value="%-d{yyyy-MM-dd HH:mm:ss,SSS} [%c]-[%p] %m%n"/>
        </layout>
    </appender>
    <!-- 根logger的设置-->
    <root>
        <!--优先级设置，all < trace < debug < info < warn < error < fatal < off -->
        <priority value="all"/>
        <appender-ref ref="myConsole"/>
        <appender-ref ref="myFile"/>
        <appender-ref ref="myFile2"/>
    </root>
</log4j:configuration>
```

三、解决方案2

3.1 Logback+SLF4j

3.1.1 导入依赖

项目中导入依赖

```
<!-- Logback依赖，还会传递 slf4j 和 logback-core -->
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
</dependency>
```

3.1.2 基本使用

Logback+SLF4J 基本API

```
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import sun.rmi.runtime.Log;
public class HelloLog {
    // 需要输出日志的类，可以创建一个log属性
    Logger log = LoggerFactory.getLogger(HelloLog.class);
    @Test
    public void test1(){
        System.out.println(log.getClass());
        log.trace("hello trace");
        log.debug("hello debug");
        log.info("hello info");
        log.warn("hello warn");
        log.error("hello error");
        // 注意，logback中没有fatal日志
    }
}
```

3.1.3 配置信息

定义 logback.xml

占位符	描述
%d{yyyy-MM-dd HH:mm:ss.SSS}	日期
%5p	日志级别，5位字符长度显示，如果内容占不满5位则内容右对齐并在左侧补空格
%-5p	5位字符长度显示日志级别，如果内容占不满5位则内容左对齐并在右侧补空格 -代表左对齐
%logger	日志所在包和类
%M	日志所在方法名
%L	日志所在代码行
%m	日志正文
%n	换行

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- scan:当此属性设置为true时，配置文件如果发生改变，将会被重新加载，默认值为true -->
<!-- scanPeriod:设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒。当scan为true时，此属性生效。
   默认的时间间隔为1分钟。 -->
<!-- debug:当此属性设置为true时，将打印出logback内部日志信息，实时查看logback运行状态。默认值为false。 -->
<configuration  scan="true" scanPeriod="60 seconds" debug="true">

    <!-- 定义变量，可通过 ${log.path}和${CONSOLE_LOG_PATTERN} 得到变量值 -->
    <property name="log.path" value="D:/log" />
    <property name="CONSOLE_LOG_PATTERN"
              value="%d{yyyy-MM-dd HH:mm:ss.SSS} |-[%-5p] in %logger.%M[line-%L] -%m%n"/>

    <!-- 输出到控制台 -->
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <!-- Threshold=即最低日志级别，此appender输出大于等于对应级别的日志
             (当然还要满足root中定义的最低级别)
        -->
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>debug</level>
        </filter>
        <encoder>
            <!-- 日志格式(引用变量) -->
            <Pattern>${CONSOLE_LOG_PATTERN}</Pattern>
            <!-- 设置字符集 -->
            <charset>UTF-8</charset>
        </encoder>
    </appender>
    <!-- 追加到文件中 -->
    <appender name="file" class="ch.qos.logback.core.FileAppender">
        <file>${log.path}/hello2.log</file>
        <encoder>
            <pattern>${CONSOLE_LOG_PATTERN}</pattern>
```

```
        </encoder>
    </appender>
    <!-- 滚动追加到文件中 -->
    <appender name="file2" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- 正在记录的日志文件的路径及文件名 -->
        <file>${log.path}/hello.log</file>
        <!--日志文件输出格式-->
        <encoder>
            <pattern>${CONSOLE_LOG_PATTERN}</pattern>
            <charset>UTF-8</charset> <!-- 设置字符集 -->
        </encoder>
        <!-- 日志记录器的滚动策略，按日期，按大小记录
             文件超过最大尺寸后，会新建文件，然后新的日志文件中继续写入
             如果日期变更，也会新建文件，然后在新的日志文件中写入当天日志
        -->
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- 新建文件后，原日志改名为如下  %i=文件序号，从0开始 -->
            <fileNamePattern>${log.path}/hello-%d{yyyy-MM-dd}.%i.log</fileNamePattern>
            <!-- 每个日志文件的最大体量 -->
            <timeBasedFileNamingAndTriggeringPolicy
                class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>8kb</maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
            <!-- 日志文件保留天数，1=则只保留昨天的归档日志文件 ,不设置则保留所有日志-->
            <maxHistory>1</maxHistory>
        </rollingPolicy>
    </appender>

    <root level="trace">
        <appender-ref ref="CONSOLE"/>
        <appender-ref ref="file"/>
        <appender-ref ref="file2"/>
    </root>

</configuration>
```