Perf4J

# 概要

# 2.计时代码

Org.perf4j.LoggingStopWatch类日志框架的一个用来记录执行时间特殊的子类.

//创建LoggingStopWatch开始计时,构造器中包含tag,标记了被计时的代码块

//通常需要实例化LoggingStopWatch的一个子类例如Log4JStopWatch 或者 CommonsLogStopWatch

StopWatch stopWatch = **new** LoggingStopWatch("codeblock1");

//执行代码块 延迟0-1秒

Thread.*sleep*((**long**) (Math.*random*() \* 1000L));

//停表并记录,调用stop()或lap()方法LoggingStopWatch自动的记录执行时间

stopWatch.stop();

上述输出会进入标准错误流中,如果采用了LoggingStopWatch的一个子类的话会输出到在日志框架中设置的file中.

此外,在stopWatch的tag中可以指定一些可选信息,这些tag或者message可以在StopWatch的构造器或者在stop方法中写明,在程序执行时tag可变.

StopWatch stopWatch = new LoggingStopWatch();

try {

// the code block being timed - this is just a dummy example

long sleepTime = (long)(Math.random() \* 1000L);

Thread.sleep(sleepTime);

if (sleepTime > 500L) {

throw new Exception("Throwing exception");

}

stopWatch.stop("codeBlock2.success", "Sleep time was < 500 ms");

} catch (Exception e) {

stopWatch.stop("codeBlock2.failure", "Exception was: " + e);

}

日志结果

INFO: start[1230493236109] time[447] tag[codeBlock2.success] message[Sleep time was < 500 ms]

INFO: start[1230493236719] time[567] tag[codeBlock2.failure] message[Exception was: java.lang.Exception: Throwing exception]

INFO: start[1230493237286] time[986] tag[codeBlock2.failure] message[Exception was: java.lang.Exception: Throwing exception]

INFO: start[1230493238273] time[194] tag[codeBlock2.success] message[Sleep time was < 500 ms]

INFO: start[1230493238467] time[463] tag[codeBlock2.success] message[Sleep time was < 500 ms]

INFO: start[1230493238930] time[310] tag[codeBlock2.success] message[Sleep time was < 500 ms]

[除了显式的创建StopWatch可以使用org.perf4j.aop.Profiled标注需要计时的方法.@Profield](mailto:除了显式的创建StopWatch可以使用org.perf4j.aop.Profiled标注需要计时的方法.@Profield)应配合一个面向方面的框架一起使用例如AspectJ,Spring AOP

# 3.解析日志文件生成性能统计

一旦在程序中使用日志记录时间,可以通过解析来生成性能统计例如平均,最小或者最大次数.可以将org.perf4j.LogParser配置为main class类运行perf4j的jar包:

Java –jar perf4j-0.9.16.jar times.log

输出如下：

Performance Statistics 20:32:00 - 20:32:30

Tag Avg(ms) Min Max Std Dev Count

codeBlock1 249.4 2 487 151.3 37

codeBlock2.failure 782.9 612 975 130.8 17

codeBlock2.success 260.7 6 500 159.5 20

Performance Statistics 20:32:30 - 20:33:00

Tag Avg(ms) Min Max Std Dev Count

codeBlock1 244.0 7 494 150.6 41

codeBlock2.failure 747.9 531 943 125.3 21

codeBlock2.success 224.1 26 398 106.8 21

Performance Statistics 20:33:00 - 20:33:30

Tag Avg(ms) Min Max Std Dev Count

codeBlock1 289.3 10 464 141.1 22

codeBlock2.failure 781.1 599 947 135.1 8

codeBlock2.success 316.2 115 490 112.6 13

如果没有为perf4j配置输出文件默认会从标准输入流中读取,因此一个获取实时性能统计的方法是使用unix命令tail将logparser接到管道上.

Tail –f times.log |java –jar perf4j.jar

# 4.使用Log4J Appenders生成实时性能统计

Perf4j的一个最大优点是可以实时监控正在运行的程序通过扩展通用的类似log4j或者logback日志框架,自定义的log4j appenders可以基于标准配置机制并且不需要修改现有代码,下面的例子都是以log4j已经加入项目为前提的。

另外,在perf4j0.9.16后加入了对logback框架的支持,下面提到的log4j appenders都有对应的logback版本.

最常用的perf4j appender 是[AsyncCoalescingStatisticsAppender](http://perf4j.codehaus.org/apidocs/org/perf4j/log4j/AsyncCoalescingStatisticsAppender.html)，这个appender会每间隔一段时间记录所有的日志信息，将他们聚合到一个GroupedTimingStatistics的log message中,然后将这些信息输出到想FileAppender这样的downStream appender中.

下面是一个log4j.xml配置[AsyncCoalescingStatisticsAppender](http://perf4j.codehaus.org/apidocs/org/perf4j/log4j/AsyncCoalescingStatisticsAppender.html)的例子,重点是必须将他连到downstream appender上，在log4j.properties中不能配置只能在log4j.xml中配置文档格式:

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration debug="false" xmlns:log4j="http://jakarta.apache.org/log4j/">

<!--

This default ConsoleAppender is used to log all NON perf4j messages

to System.out

-->

<appender name="console" class="org.apache.log4j.ConsoleAppender">

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%-5p %c{1} - %m%n"/>

</layout>

</appender>

<!-- Perf4J appenders -->

<!--

This AsyncCoalescingStatisticsAppender groups StopWatch log messages

into GroupedTimingStatistics messages which it sends on the

file appender defined below

-->

<appender name="CoalescingStatistics"

class="org.perf4j.log4j.AsyncCoalescingStatisticsAppender">

<!--

The TimeSlice option is used to determine the time window for which

all received StopWatch logs are aggregated to create a single

GroupedTimingStatistics log. Here we set it to 10 seconds, overriding

the default of 30000 ms

-->

<param name="TimeSlice" value="10000"/>

<appender-ref ref="fileAppender"/>

</appender>

<!-- This file appender is used to output aggregated performance statistics -->

<appender name="fileAppender" class="org.apache.log4j.FileAppender">

<param name="File" value="perfStats.log"/>

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%m%n"/>

</layout>

</appender>

<!-- Loggers -->

<!--

The Perf4J logger. Note that org.perf4j.TimingLogger is the value of the

org.perf4j.StopWatch.DEFAULT\_LOGGER\_NAME constant. Also, note that

additivity is set to false, which is usually what is desired - this means

that timing statements will only be sent to this logger and NOT to

upstream loggers.

-->

<logger name="org.perf4j.TimingLogger" additivity="false">

<level value="INFO"/>

<appender-ref ref="CoalescingStatistics"/>

</logger>

<!--

The root logger sends all log statements EXCEPT those sent to the perf4j

logger to System.out.

-->

<root>

<level value="INFO"/>

<appender-ref ref="console"/>

</root>

</log4j:configuration>

Logger rootLogger = Logger.getRootLogger();

for (int i = 0; i < 50; i++) {

// By default the Log4JStopWatch uses the Logger named org.perf4j.TimingLogger

StopWatch stopWatch = new Log4JStopWatch();

// for demo purposes just sleep

Thread.sleep((long) (Math.random() \* 1000L));

rootLogger.info("Normal logging messages only go to the console");

// Calling lap() stops timing for the previous block, sends the

// message to the log4j Logger, and starts timing the next block.

stopWatch.lap("firstBlock");

Thread.sleep((long) (Math.random() \* 2000L));

stopWatch.stop("secondBlock");

}

只有rootLogger记录的信息会显示在控制台,perfStats.log的文本如下

Performance Statistics 17:05:40 - 17:05:50

Tag Avg(ms) Min Max Std Dev Count

firstBlock 434.2 128 869 309.7 6

secondBlock 968.4 196 1996 675.5 5

和log parser相似,log4j appender也支持csv格式。

# 5.通过GraphingStaticsAppender画图

使用AsyncCoalescingStatisticsAppender的好处是其他自定义的perf4j appenders可以处理GourpedtimingStatics信息, AsyncCoalescingStatisticsAppender使用AsyncCoalescingStatisticsAppender生成性能图像,这些图像是通过Google Chart api来生成的图像可以在浏览器中查看.

下面的log4j.xml继承了先前的版本扩展了GraphingStatisticsAppenders

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration debug="false" xmlns:log4j="http://jakarta.apache.org/log4j/">

<!--

This default ConsoleAppender is used to log all NON perf4j messages

to System.out

-->

<appender name="console" class="org.apache.log4j.ConsoleAppender">

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%-5p %c{1} - %m%n"/>

</layout>

</appender>

<!-- Perf4J appenders -->

<!--

This AsyncCoalescingStatisticsAppender groups StopWatch log messages

into GroupedTimingStatistics messages which it sends on the

file appender defined below

-->

<appender name="CoalescingStatistics"

class="org.perf4j.log4j.AsyncCoalescingStatisticsAppender">

<!--

The TimeSlice option is used to determine the time window for which

all received StopWatch logs are aggregated to create a single

GroupedTimingStatistics log. Here we set it to 10 seconds, overriding

the default of 30000 ms

-->

<param name="TimeSlice" value="10000"/>

<appender-ref ref="fileAppender"/>

<!--

Note how the GraphingStatisticsAppenders have been attached to the

CoalescingStatistics here.

-->

<appender-ref ref="graphExecutionTimes"/>

<appender-ref ref="graphExecutionTPS"/>

</appender>

<!-- This file appender is used to output aggregated performance statistics -->

<appender name="fileAppender" class="org.apache.log4j.FileAppender">

<param name="File" value="perfStats.log"/>

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%m%n"/>

</layout>

</appender>

<!--

This first GraphingStatisticsAppender graphs Mean execution times for the

firstBlock and secondBlock tags

-->

<appender name="graphExecutionTimes"

class="org.perf4j.log4j.GraphingStatisticsAppender">

<!-- Possible GraphTypes are Mean, Min, Max, StdDev, Count and TPS -->

<param name="GraphType" value="Mean"/>

<!-- The tags of the timed execution blocks to graph are specified here -->

<param name="TagNamesToGraph" value="firstBlock,secondBlock"/>

<appender-ref ref="graphsFileAppender"/>

</appender>

<!--

This second GraphingStatisticsAppender graphs transactions per second

for the firstBlock and secondBlock tags

-->

<appender name="graphExecutionTPS"

class="org.perf4j.log4j.GraphingStatisticsAppender">

<param name="GraphType" value="TPS"/>

<param name="TagNamesToGraph" value="firstBlock,secondBlock"/>

<appender-ref ref="graphsFileAppender"/>

</appender>

<!--

This file appender is used to output the graph URLs generated

by the GraphingStatisticsAppenders

-->

<appender name="graphsFileAppender" class="org.apache.log4j.FileAppender">

<param name="File" value="perfGraphs.log"/>

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%m%n"/>

</layout>

</appender>

<!-- Loggers -->

<!--

The Perf4J logger. Note that org.perf4j.TimingLogger is the value of the

org.perf4j.StopWatch.DEFAULT\_LOGGER\_NAME constant. Also, note that

additivity is set to false, which is usually what is desired - this means

that timing statements will only be sent to this logger and NOT to

upstream loggers.

-->

<logger name="org.perf4j.TimingLogger" additivity="false">

<level value="INFO"/>

<appender-ref ref="CoalescingStatistics"/>

</logger>

<!--

The root logger sends all log statements EXCEPT those sent to the perf4j

logger to System.out.

-->

<root>

<level value="INFO"/>

<appender-ref ref="console"/>

</root>

</log4j:configuration>

# 6.通过JMX属性暴露性能统计

另一个比较常用的自定义perf4j appender是JmxAttributeStatisticsAppender,这个appender通过JMX MBean暴露性能指标,并且允许设置临界值.有好多第三方工具可以和jmx交互,暴露性能通过jmx可以拓宽视野.

Log4j.xml：

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration debug="false" xmlns:log4j="http://jakarta.apache.org/log4j/">

<!--

This default ConsoleAppender is used to log all NON perf4j messages

to System.out

-->

<appender name="console" class="org.apache.log4j.ConsoleAppender">

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%-5p %c{1} - %m%n"/>

</layout>

</appender>

<!-- Perf4J appenders -->

<!--

This AsyncCoalescingStatisticsAppender groups StopWatch log messages

into GroupedTimingStatistics messages which it sends on the

file appender defined below

-->

<appender name="CoalescingStatistics"

class="org.perf4j.log4j.AsyncCoalescingStatisticsAppender">

<!--

The TimeSlice option is used to determine the time window for which

all received StopWatch logs are aggregated to create a single

GroupedTimingStatistics log. Here we set it to 10 seconds, overriding

the default of 30000 ms

-->

<param name="TimeSlice" value="10000"/>

<appender-ref ref="fileAppender"/>

<appender-ref ref="graphExecutionTimes"/>

<appender-ref ref="graphExecutionTPS"/>

<!-- We add the JMX Appender reference onto the CoalescingStatistics -->

<appender-ref ref="perf4jJmxAppender"/>

</appender>

<!-- This file appender is used to output aggregated performance statistics -->

<appender name="fileAppender" class="org.apache.log4j.FileAppender">

<param name="File" value="perfStats.log"/>

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%m%n"/>

</layout>

</appender>

<!--

This first GraphingStatisticsAppender graphs Mean execution times for the

firstBlock and secondBlock tags

-->

<appender name="graphExecutionTimes"

class="org.perf4j.log4j.GraphingStatisticsAppender">

<!-- Possible GraphTypes are Mean, Min, Max, StdDev, Count and TPS -->

<param name="GraphType" value="Mean"/>

<!-- The tags of the timed execution blocks to graph are specified here -->

<param name="TagNamesToGraph" value="firstBlock,secondBlock"/>

<appender-ref ref="graphsFileAppender"/>

</appender>

<!--

This second GraphingStatisticsAppender graphs transactions per second

for the firstBlock and secondBlock tags

-->

<appender name="graphExecutionTPS"

class="org.perf4j.log4j.GraphingStatisticsAppender">

<param name="GraphType" value="TPS"/>

<param name="TagNamesToGraph" value="firstBlock,secondBlock"/>

<appender-ref ref="graphsFileAppender"/>

</appender>

<!--

This file appender is used to output the graph URLs generated

by the GraphingStatisticsAppenders

-->

<appender name="graphsFileAppender" class="org.apache.log4j.FileAppender">

<param name="File" value="perfGraphs.log"/>

<layout class="org.apache.log4j.PatternLayout">

<param name="ConversionPattern" value="%m%n"/>

</layout>

</appender>

<!--

This JMX appender creates an MBean and publishes it to the platform MBean server by

default.

-->

<appender name="perf4jJmxAppender" class="org.perf4j.log4j.JmxAttributeStatisticsAppender">

<!--

You must specify the tag names whose statistics should be exposed as

MBean attributes.

-->

<param name="TagNamesToExpose" value="firstBlock,secondBlock"/>

<!--

The NotificationThresholds param configures the sending of JMX notifications

when statistic values exceed specified thresholds. This config states that

the firstBlock max value should be between 0 and 800ms, and the secondBlock max

value should be less than 1500 ms. You can also set thresholds on the Min,

Mean, StdDev, Count and TPS statistics - e.g. firstBlockMean(<600).

-->

<param name="NotificationThresholds" value="firstBlockMax(0-800),secondBlockMax(&lt;1500)"/>

<!--

You can also specify an optional MBeanName param, which overrides

the default MBean name of org.perf4j:type=StatisticsExposingMBean,name=Perf4J

-->

</appender>

<!-- Loggers -->

<!--

The Perf4J logger. Note that org.perf4j.TimingLogger is the value of the

org.perf4j.StopWatch.DEFAULT\_LOGGER\_NAME constant. Also, note that

additivity is set to false, which is usually what is desired - this means

that timing statements will only be sent to this logger and NOT to

upstream loggers.

-->

<logger name="org.perf4j.TimingLogger" additivity="false">

<level value="INFO"/>

<appender-ref ref="CoalescingStatistics"/>

</logger>

<!--

The root logger sends all log statements EXCEPT those sent to the perf4j

logger to System.out.

-->

<root>

<level value="INFO"/>

<appender-ref ref="console"/>

</root>

</log4j:configuration>

运行程序时,可以通过传入com.sun.management.jmxremote来开启

java -Dcom.sun.management.jmxremote -cp .:./perf4j-0.9.16.jar:./log4j-1.2.14.jar Perf4JappenderExample

程序启动后可以通过jconsole链接Mbean来观察.

# 7.在web项目中暴露性能监控图像

高性能的web需要良好的性能监控,如果之前配置了GraphingStatisticsAppender,可以通过在web.xml中配置GraphingServlet来暴露性能图像给前端浏览器,例如:

<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.4"

xmlns="http://java.sun.com/xml/ns/j2ee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<servlet>

<servlet-name>perf4j</servlet-name>

<servlet-class>org.perf4j.log4j.servlet.GraphingServlet</servlet-class>

<!--

The graphNames parameter determines which graphs to expose. The

param-value should be a comma-separated list of the

appender NAMES as defined in the log4j.xml file.

-->

<init-param>

<param-name>graphNames</param-name>

<param-value>graphExecutionTimes,graphExecutionTPS</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>perf4j</servlet-name>

<url-pattern>/perf4j</url-pattern>

</servlet-mapping>

</web-app>

GraphingServlet从GraphingStatisticsAppenders获取内存中的图像,因此展示的是实时的数据,