Десятая лекция
java for web
Log4j

Что такое логгирование.

Логирование помогает нам следить за выполнением логики нашей программы.

Обычно начинающие используют System.out.println для вывода логов но это неправильный подход и не всегда удобно.

Принципиально логи нужны не только для отслеживания программых ошибок – точно так же можно отслеживать и целенаправленные действия пользователей.

Допустим мы администрирует магазин - начнет разбираться в чем же проблема., но на не все логи сервера нужны и как найти то что нужно.

В этом случае решение следующее, выводить нужные нам логи в отдельный файл. Но как понять, какие из всех логов, которые сыпятся в общий лог сервера нужны нам?

Для этого нужно реализовать свою систему логирования, где мы сможете указать какие логи куда выводить, или же настроить уровни логирования.

Логгирование в java

System.out.println()

Log4J

java.util.logging

Apache Commons Logging

Simple Logging Facade for Java

Logback

Краткое резюме

Apache Log4J – хороший фреймворк для логирования, практически лишенный недостатков. Широко используется. Разработка находится, фактически, в замороженном5 состоянии, производится только исправление ошибок.

java.util.logging – фреймворк, являющийся частью JavaSE. По возможностям уступает Log4J. Тем не менее используется хотя бы потому, что всегда под рукой и не требует дополнительных библиотек.

Apache Commons Logging – фреймворк, предназначенный для абстрагирования конкретного фреймворка ("под" ним может работать как Log4J, так и java.util.logging, а также несколько других). Имеет определенные проблемы с загрузчиком классов, что в определенных ситуациях затрудняет его использование.

SLF4J – Simple Logging Facade for Java еще один абстрагирующий фреймворк, существенно более удачный, чем Commons Logging. Может работать в двух ипостасях – как общий интерфейс к лежащим ниже фреймворкам и как приемник соответствующего типа для фреймворков, расположенных "над" ним.

Logback – молодой, но весьма интересный фреймворк, выросший из Log4J. Взял от родителя все преимущества, плюс еще добавил своих. Возможно, в будущем станет даже более привлекательным, чем Log4J.

Подключение

Для использования этого фреймверка нам необходимо
– подключить библиотеку фреймворка;
– создать конфигурационный файл с параметрами логирования;
– создать объект лога в своем приложении;
– воспользоваться методами для записи в лог.

Зависимости, которые надо было подключить.
Добавим код в теги <dependencies></dependencies>

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.16</version>
</dependency>
```

В основе библиотеки Log4J лежит три понятия

– логгер (logger),
– аппендер (appender)
– компоновка (layout).

Использование Log4J

Логгер представляет собой объект класса org.apache.log4j.Logger, который используется для вывода данных и управления уровнем (детализацией) вывода. В текущей версии – 1.2.16 – Log4J поддерживает следующие уровни вывода, в порядке возрастания:

1. TRACE - сообщения самого низкого уровня, наиболее подробные
2. DEBUG - отладочные сообщения, менее подробные, чем TRACE
3. INFO - стандартные информационные сообщения
4. WARN - некритичные ошибки, не препятствующие работе приложения
5. ERROR - ошибки, которые могут привести к неверному результату
6. FATAL - ошибки, препятствующие дальнейшей работе приложения

Аппендер

Для добавления записи в лог используется так называемый аппендер. В библиотеке определено множество различных типов аппендеров

- Консоль
- Файлы (несколько различных типов)
- JDBC
- Темы (topics) JMS
- NT Event Log
- SMTP
- Сокет
- Syslog
- Telnet
- Любой java.io.Writer или java.io.OutputStream

Файловые аппендеры

org.apache.log4j.FileAppender -этот аппендер добавляет данные в файл до бесконечности. И в этом его существенный недостаток, этот аппендер сам по себе практически не используется. Он является базой для остальных, предоставляя общие средства работы с файлами.

org.apache.log4j.RollingFileAppender позволяет ротировать файл по достижении определенного размера. "Ротировать" означает, что текущему файлу приписывается расширение ".0" и открывается следующий. По достижении им максимального размера – первому вместо расширения ".0" выставляется ".1", текущему – ".0" (свойствами maximumFileSize и maxBackupIndex соответственно.)

org.apache.log4j.DailyRollingFileAppender В отличии от org.apache.log4j.RollingFileAppender-а, ротирующего файл по достижении определенного размера, org.apache.log4j.DailyRollingFileAppender ротирует файл с определенной частотой. Она зависит от шаблона, указанного в конфигурации

Компоновка

Для конфигурирования формата вывода используются наследники класса org.apache.log4j.Layout:

- org.apache.log4j.SimpleLayout
- org.apache.log4j.HTMLLayout
- org.apache.log4j.xml.XMLLayout
- org.apache.log4j.TTCCLayout
- org.apache.log4j.PatternLayout / org.apache.log4j.EnhancedPatternLayout

org.apache.log4j.SimpleLayout

Наиболее простой вариант. На выходе дает уровень вывода и, собственно, сообщение. Т.е. следующий код –

```
logger.info("Some message");
```

– на выходе даст вот так отформатированную строку:

```
INFO - Some message
```

org.apache.log4j.HTMLLayout

Форматирует сообщения в виде HTML-таблицы.

org.apache.log4j.xml.XMLLayout

Этот компоновщик формирует сообщения в виде XML.

org.apache.log4j.TTCCLayout

TTCC – сокращение от Time-Thread-Category-Context. Означает оно, что помимо, собственно, сообщения, в лог выводится информация о времени, потоке, категории (имени логгера) и вложенном диагностическом контексте У компоновщика есть булевские свойства CategoryPrefixing, ContextPrinting и ThreadPrinting, указывающие, выводить или нет категорию, контекст и имя потока, соответственно. По умолчанию все три свойства выставлены в true.

org.apache.log4j.PatternLayout

Наиболее часто используемого компоновщика – PatternLayout. Он использует шаблонную строку для форматирования выводимого сообщения.

org.apache.log4j.PatternLayout

%d{ISO8601} [%-5p][%-16.16t][%32.32c] - %m%n

11:31:32,342 Thread-1 ERROR audit.LoadTest - Check in 344ms: GlobalID=2

11:31:32,358 Thread-17 WARN ServiceLoadTest - Check in 156ms: GlobalID=8

11:31:32,378 Thread-2 INFO trace.ServiceLoadTrace - Check in 328ms: GlobalID=3

11:31:35,358 Thread-44 DEBUG ext.ServiceParallelLoadTest - Check in 250ms: GlobalID=5

11:31:36,637 Thread-503 INFO ServiceLoadTest - Check in 219ms: GlobalID=6

11:31:37,846 Thread-59 INFO extract.Extractor - Check in 94ms: GlobalID=10

11:31:39,072 Thread-86 DEBUG ServiceLoadTest - Check in 188ms: GlobalID=7

11:31:41,309 Thread-10 INFO back.BackLoaderInfo - Check in 47ms: GlobalID=11

Конфигурирование

Конфигурирование Log4J осуществляется двумя способами – через файл свойств и через xml-файл., конфигурационные файлы называются log4j.properties и log4j.xml.

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

     <appender name="console" class="org.apache.log4j.ConsoleAppender">
       <param name="Target" value="System.out"/>
       <layout class="org.apache.log4j.PatternLayout">
         <param name="ConversionPattern" value="%p %c: %m%n"/>
       </layout>
     </appender>

     <root>
       <priority value ="debug" />
       <appender-ref ref="console" />
     </root>
 </log4j:configuration>
```

Конфигурирование

//log4j.properties

log4j.rootCategory=DEBUG, console

log4j.appender.console=org.apache.log4j.ConsoleAppender

log4j.appender.console.layout=org.apache.log4j.PatternLayout

log4j.appender.console.layout.ConversionPattern= %p %c: %m%n

c Категория сообщения.

После символа категории в фигурных скобках может следовать указание – сколько частей имени категории выводить. Еще один вариант сокращения – запись вида %c{1.2.3.}. Означает она, что от первой части остается одна буква, от второй – две, от третьей – три. На оставшиеся части распространяется последнее значение.

d Дата и/или время

Выводит в лог текущие дату и/или время. В фигурных скобках после данной опции указывается формат даты – либо шаблон java.text.SimpleDateFormat, либо один из предустановленных – DATE, ABSOLUTE или ISO8601.

F Имя файла, в котором было сгенерировано сообщение.

l Полная информация о точке генерации сообщения.

L Номер строки, в которой было сгенерировано сообщение.

m Сообщение

То самое сообщение, которое передается в метод логгера. Ради чего, в основном, всё и затевается.

M Имя метода, в котором было сгенерировано сообщение.

n Перевод строки

p Приоритет сообщения.

Выводит уровень логирования для сообщения.

r Количество миллисекунд с момента инициализации системы логирования.

Аналог формата даты RELATIVE компоновщика TTCCLayout. Может использоваться вместо даты, если есть такая необходимость.
t Имя потока.

Использование логгеров

Инициализация логгера статическими методами:

```
private static final Logger logger = Logger.getLogger(MyClass.class);
```

Инициализация по классу. Наиболее распространенный вариант, имя категории принимается равным полному имени класса

```
private static final Logger dbLogger = Logger.getLogger("ru.skipy.DB");
```

Возвращается логгер по указанному имени, этот метод может быть полезен, если есть необходимость в разных классах выводить сообщения в одной категории.

Использование логгера в коде вызывается соответствующий метод:

```
logger.info("Starting mass rate charge calculation...");
```