Mini e-book

Menggunakan Logging Framework

log4j

last updated 20080710

Daftar Isi

I. Tentang buku ini

A. Lisensi

Buku ini berlisensi Creative Commons Attribution-NonCommercial-ShareAlike. Artinya:

Anda bebas untuk:

- mengkopi, membagikan, dan mengirimkan dokumen ini
- mengadaptasi dokumen ini

Dengan persyaratan berikut:

- Attribution : Nama pengarang yang asli harus dicantumkan
- NonCommercial : Tidak boleh digunakan untuk keperluan komersil
- ShareAlike : Bila Anda membuat perubahan, mentransformasi, atau membuat dokumen lain dari dokumen ini, hasilnya harus didistribusikan dengan lisensi yang sejenis dengan ini.

Lebih lanjut tentang lisensi ini bisa dilihat di:

http://creativecommons.org/licenses/by-nc-sa/3.0/

B. Segmen Pembaca

Berikut adalah asumsi tentang pembaca buku ini:

- menguasai Java Fundamental versi 1.4 ke atas

II. Tentang logging

Logging adalah salah satu cara untuk memonitor aplikasi yang sedang berjalan. Dengan menggunakan logging, kita dapat melihat isi variabel, mengetahui event penting dalam aplikasi, dan melihat jalannya aplikasi. Kemampuan melihat ke dalam aplikasi ini memudahkan kita untuk melakukan tuning performance, mendeteksi sumber masalah, dan lain sebagainya.

Untuk melakukan logging dengan benar, ada beberapa hal yang harus kita pertimbangkan, antara lain:

- Kemudahan menyalakan/mematikan log.
- Fleksibilitas display log message.
- Fleksibilitas pengaturan format message.
- Kemudahan filtering bagian aplikasi yang ingin dilog

Perangkat logging harus mudah dinyalakan dan dimatikan sesuai kebutuhan. Log message yang ingin kita lihat pada fase development tentu berbeda dengan pada fase production. Di fase development kita ingin melihat isi dari variabel, parameter pemanggilan method, kondisi suatu object, dan berbagai informasi detail lainnya. Informasi detail ini tidak dibutuhkan setelah aplikasi gol-live. Pada fase production, yang lebih dibutuhkan adalah informasi tentang event dan error yang terjadi.

Lokasi display juga menjadi pertimbangan penting. Pada waktu aplikasi baru dinyalakan misalnya, kita ingin melihat log message di konsol/command prompt. Setelah aplikasi berjalan beberapa bulan, tentunya kita ingin media penyimpanan yang lebih permanen agar log message bisa difilter dan dicari dengan lebih efektif. Media yang digunakan antara lain database atau file. Kadangkala kita juga ingin mengintegrasikan log aplikasi dengan log sistem operasi, agar system administrator mudah mendeteksi masalah pada aplikasi. Di lingkungan Unix kita mengenal syslog, sedangkan di Windows kita mengenal Event Viewer.

Selanjutnya, format log message itu sendiri. Untuk kebutuhan audit misalnya, kita hanya ingin melihat nilai transaksi yang terjadi. Tapi untuk kebutuhan troubleshooting, kita mungkin butuh melihat waktu terjadinya error, client mana yang menyebabkan error, dan informasi kontekstual lainnya.

Bila aplikasi yang ingin dimonitor terdiri dari banyak modul, kita ingin memiliki fleksibilitas untuk memilah-milah log message. Misalnya, untuk modul A kita ingin melihat tanggal dan pesan, diarahkan ke file, hanya menampilkan pesan error. Untuk modul B, kita ingin melihat informasi lengkap dan disimpan ke database.

Logging framework yang baik harus bisa mengakomodasi keempat hal di atas. Beruntunglah kita para programmer Java, karena hal ini sudah disediakan oleh framework Log4J.

III. Pengenalan Log4J

A. Fitur Log4J

- Berbagai level logging.
- Berbagai appender
- Format log message fleksibel
- Mendukung multithreading
- Mendukung aplikasi multiclient

B. Framework Komplementer

- commons-logging
- slf4j

C. Framework alternatif

- java.util.logging

IV. Menggunakan log4j

A. Instalasi

Sebagaimana library Java lainnya, log4j tidak membutuhkan langkah instalasi khusus. Cukup masukkan *.jar dan file konfigurasinya (`log4j.properties`) ke dalam classpath.

B. Istilah penting dalam dunia log4j

Logger

Logger adalah object yang bertugas mengeluarkan pesan log. Inilah object utama yang kita gunakan dalam kode program.

Category

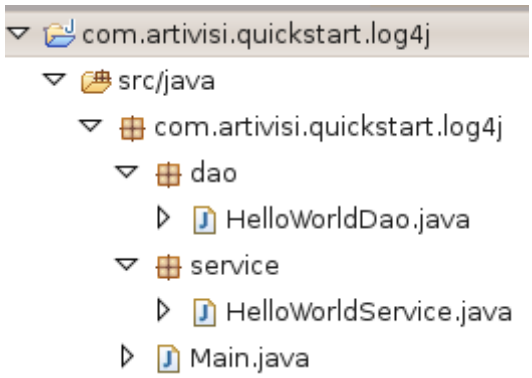
Logger dikelompokkan ke dalam kategori. Ini memudahkan kita untuk melakukan pemilahan log sesuai kebutuhan modul tertentu.

Konfigurasi kategori dilakukan pada file `log4j.properties`. Masing-masing kategori dapat ditentukan level dan appendernya masing-masing, independen terhadap kategori lainnya.

Kategori memiliki fitur inheritance. Dengan demikian, kategori yang

menjadi sub dari kategori induk akan memiliki konfigurasi yang sama dengan induknya.

Misalnya kode program kita berstruktur sebagai berikut:



Maka package `com.artivisi.quickstart.log4j` akan menjadi induk dari

`com.artivisi.quickstart.log4j.dao`

dan

`com.artivisi.quickstart.log4j.service`

Level

Secara default, log4j menyediakan beberapa level logging, diurutkan dari yang paling tidak penting.

- TRACE
- DEBUG
- INFO
- WARN
- ERROR
- FATAL

Kalau enam level ini masih kurang, kita dapat mendefinisikan level sendiri. Tapi biasanya enam sudah lebih dari cukup.

Pada saat aplikasi dijalankan, kita bisa memilih salah satu level sebagai ambang batas logging. Log message yang memiliki level sama atau dibawah ambang batas akan ditampilkan.

Sebagai contoh, bila ambang batasnya adalah INFO, maka log message yang berlevel INFO, WARN, ERROR, FATAL akan ditampilkan. Bila ambang batasnya adalah WARN, maka level TRACE,

DEBUG, dan INFO tidak akan tampil.

Appender

Appender adalah object yang bertugas menampilkan log message. Ada beberapa appender yang biasa digunakan, misalnya:

- ConsoleAppender : untuk menampilkan log ke konsol atau command prompt
- RollingFileAppender : untuk menyimpan log ke dalam file. File ini bisa dikonfigurasi ukuran maksimumnya, nama filenya, dan jumlah backup yang akan dibuat.

Selain kedua appender tersebut, ada juga appender untuk menyimpan ke database, mengirim email, mengakses log sistem operasi, dan berbagai tujuan lainnya.

C. Menggunakan log4j dalam kode program

Inisialisasi logger

Sebagai contoh, kita akan membuat satu class yang bernama HelloWorldDao. Untuk class ini, kita akan mendeklarasikan object logger. Object logger ini diberi nama sesuai dengan nama classnya. Berikut kode programnya.

```
public class HelloWorldDao {

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

}
```

Menampilkan log message

Selanjutnya, kita bisa menggunakan object logger ini dalam method save sebagai berikut:

```
    public void save() {

        logger.info("Menjalankan method save");

    }
```

Kadangkala, kita perlu menampilkan isi variabel untuk kebutuhan debugging. Mari kita lakukan hal ini di method delete.

```
    public void delete(Integer id) {
        logger.info("Menjalankan method delete");
        if (logger.isDebugEnabled()) {
            logger.debug("ID : " + id);
        }
    }
```

Mendeteksi Level yang sedang aktif

Pada contoh kode di atas, kita menambahkan pemeriksaan dengan blok if untuk mendeteksi level yang sedang aktif. Ini dilakukan untuk alasan performance. Kita bisa saja menjalankan debug log seperti ini:

```
    public void delete(Integer id) {
        logger.info("Menjalankan method delete");
        logger.debug("ID : " + id);
    }
```

Pada saat kode program ini dijalankan, Java VM akan terlebih dulu menggabungkan string `"ID : "` dengan isi variabel `id`, hasilnya kemudian akan diumpankan ke dalam method debug. Walaupun pada akhirnya log message tersebut tidak jadi ditampilkan (karena ambang batasnya tidak mencakup DEBUG), tapi penggabungan string tetap sudah terjadi. Ini akan menimbulkan overhead yang tidak perlu.

Oleh karena itu, kita lakukan dulu pemeriksaan dengan `logger.isDebugEnabled()` sebelum menggabungkan string. Beban sistem dalam menjalankan `logger.isDebugEnabled()` jauh lebih ringan daripada penggabungan string.

Log message untuk Exception

Berikutnya, ini adalah contoh kode yang benar dalam membuat log message untuk exception. Bila terjadi exception, kita ingin mengambil semua informasi yang tersedia untuk memudahkan troubleshooting. Pertama, kita edit method delete agar menimbulkan exception.

```
    public void delete(Integer id) {
        logger.info("Menjalankan method delete");

        if (logger.isDebugEnabled()) {
            logger.debug("ID : " + id);
        }

        if (id < 0) {
           throw new IllegalArgumentException("ID tidak boleh kurang dari nol");
        }
    }
```

Selanjutnya, mari kita tangkap exception ini di dalam class HelloWorldService.

```
public class HelloWorldService {
 private static final Logger logger = Logger.getLogger(HelloWorldService.class);
 private HelloWorldDao helloWorldDao;

    public void setHelloWorldDao(HelloWorldDao helloWorldDao) {
        this.helloWorldDao = helloWorldDao;
    }

    public void deleteTheWorld(Integer id){
        logger.log(Level.WARN, "Method deleteTheWorld dijalankan");
        try {
            helloWorldDao.delete(id);
        } catch (IllegalArgumentException e) {
            logger.error(e.getMessage(), e);
        }
    }
}
```

Perhatikan blok try-catch. Kita memasukkan dua parameter ke method `logger.error()`. Parameter pertama adalah pesan errornya, sedangkan parameter kedua adalah object Exception itu sendiri. Dengan cara ini, semua informasi yang ada dalam exception tidak akan hilang.

Seperti kita lihat pada contoh kode di atas, kita menggunakan berbagai level logging agar kita bisa mengamati perubahan isi log message bila terjadi perubahan konfigurasi.

Menjalankan aplikasi

Terakhir, mari kita buat class Main untuk menjalankan kedua class di atas.

```
public class Main {
    private static Logger logger = Logger.getLogger(Main.class);
    public static void main(String[] args) {
        logger.info("Menginstankan dao");
        HelloWorldDao dao = new HelloWorldDao();

        logger.info("Menginstankan service");
        HelloWorldService service = new HelloWorldService();

        logger.debug("Merangkai object");
        service.setHelloWorldDao(dao);

        // jalankan method save
        logger.info("Save the world");
        service.saveTheWorld();

        // jalankan method delete, skenario normal
        logger.trace("Akan menghapus World 10");
        service.deleteTheWorld(10);

        // jalankan method delete, skenario error
        logger.trace("Akan menghapus World -7");
        service.deleteTheWorld(-7);
    }
}
```

D. Konfigurasi log4j

Konfigurasi sederhana

Untuk konfigurasi awal, kita akan menampilkan log message ke konsol, dengan level INFO, untuk seluruh aplikasi. Berikut konfigurasinya.

```
# Konfigurasi kategori
log4j.rootLogger=INFO,Konsole

# Konfigurasi appender Konsole
log4j.appender.Konsole=org.apache.log4j.ConsoleAppender
log4j.appender.Konsole.layout=org.apache.log4j.PatternLayout
# Format log : tanggal - thread - level -
log4j.appender.Konsole.layout.ConversionPattern=%d [%t] %-5p %c - %m%n
```

Konfigurasi di atas akan menampilkan log ke konsol sebagai berikut:

```
2008-07-18 11:44:39,656 [main] INFO com.artivisi.quickstart.log4j.Main -
Menginstankan dao
2008-07-18 11:44:39,659 [main] INFO com.artivisi.quickstart.log4j.Main -
Menginstankan service
2008-07-18 11:44:39,661 [main] DEBUG com.artivisi.quickstart.log4j.Main -
Merangkai object
2008-07-18 11:44:39,661 [main] INFO com.artivisi.quickstart.log4j.Main - Save
the world
2008-07-18 11:44:39,661 [main] INFO
com.artivisi.quickstart.log4j.dao.HelloWorldDao - Menjalankan method save
2008-07-18 11:44:39,662 [main] TRACE com.artivisi.quickstart.log4j.Main - Akan
menghapus World 10
2008-07-18 11:44:39,663 [main] WARN
com.artivisi.quickstart.log4j.service.HelloWorldService - Method deleteTheWorld
dijalankan
2008-07-18 11:44:39,664 [main] INFO
com.artivisi.quickstart.log4j.dao.HelloWorldDao - Menjalankan method delete
2008-07-18 11:44:39,668 [main] TRACE com.artivisi.quickstart.log4j.Main - Akan
menghapus World -7
2008-07-18 11:44:39,669 [main] WARN
com.artivisi.quickstart.log4j.service.HelloWorldService - Method deleteTheWorld
dijalankan
2008-07-18 11:44:39,670 [main] INFO
com.artivisi.quickstart.log4j.dao.HelloWorldDao - Menjalankan method delete
2008-07-18 11:44:39,672 [main] ERROR
com.artivisi.quickstart.log4j.service.HelloWorldService - ID tidak boleh kurang
dari nol
java.lang.IllegalArgumentException: ID tidak boleh kurang dari nol
   at
com.artivisi.quickstart.log4j.dao.HelloWorldDao.delete(HelloWorldDao.java:20)
   at
com.artivisi.quickstart.log4j.service.HelloWorldService.deleteTheWorld(HelloWorld
Service.java:24)
   at com.artivisi.quickstart.log4j.Main.main(Main.java:30)
```

Baris log terakhir menunjukkan terjadinya exception. Keseluruhan stack trace ikut ditampilkan dalam log message. Ini akan

memudahkan programmer dalam melakukan troubleshooting.

Output log ke file

Setelah berhasil menampilkan log message ke konsol, sekarang kita akan menyimpan pesan log ke file. Kita akan gunakan `RollingFileAppender` yang dapat secara otomatis membuat file baru apabila file utama sudah mencapai ukuran tertentu.

Berikut adalah konfigurasi untuk `RollingFileAppender`. Kita memberi nama `Roll` untuk appender ini.

```
# Konfigurasi kategori
log4j.rootLogger=INFO,Konsole,Roll


# Konfigurasi appender Konsole
log4j.appender.Konsole=org.apache.log4j.ConsoleAppender
log4j.appender.Konsole.layout=org.apache.log4j.PatternLayout
# Format tanggal menurut ISO-8601 : %d
log4j.appender.Konsole.layout.ConversionPattern=%d [%t] %-5p %c - %m%n


# Konfigurasi appender Roll
log4j.appender.Roll=org.apache.log4j.RollingFileAppender
log4j.appender.Roll.File=logs/tutorial.log
log4j.appender.Roll.MaxFileSize=10KB
log4j.appender.Roll.MaxBackupIndex=2
log4j.appender.Roll.layout=org.apache.log4j.PatternLayout
# Format tanggal menurut ISO-8601 : %d
log4j.appender.Roll.layout.ConversionPattern=%d [%t] %p (%F:%L) - %m%n
```

Pada konfigurasi di atas, kita menyimpan log message ke file `logs/tutorial.log`. Ukuran maksimal file ini adalah sebesar 10KB. Bila lebih dari 10KB, log4j akan membuat file baru dan mengganti nama file lama menjadi `tutorial.log.1`. Jumlah file backup yang dibuat adalah 2 file, jadi maksimal akan terbentuk file `tutorial.log`, `tutorial.log.1`, dan `tutorial.log.2`.

Konfigurasi output message

Kita juga menggunakan pattern yang berbeda untuk menampilkan log message, berikut adalah penjelasan dari conversion pattern:

- %d : tanggal saat log dibuat
- %t : thread yang menulis log
- %p : level log message

- %c : nama class yang menghasilkan log message
- %F : nama file source code
- %L : nomor baris dalam source code

Dengan pattern `%d [%t] %p (%F:%L) - %m%n`, berikut adalah log message yang dihasilkan.

```
 2008-07-18 13:12:05,702 [main] INFO (Main.java:11) - Menginstankan dao

 2008-07-18 13:12:05,706 [main] INFO (Main.java:14) - Menginstankan service

 2008-07-18 13:12:05,707 [main] DEBUG (Main.java:17) - Merangkai object

 2008-07-18 13:12:05,707 [main] INFO (Main.java:21) - Save the world

 2008-07-18 13:12:05,708 [main] INFO (HelloWorldDao.java:9) - Menjalankan method
save

 2008-07-18 13:12:05,708 [main] TRACE (Main.java:25) - Akan menghapus World 10

 2008-07-18 13:12:05,709 [main] WARN (HelloWorldService.java:22) - Method
deleteTheWorld dijalankan

 2008-07-18 13:12:05,712 [main] INFO (HelloWorldDao.java:13) - Menjalankan method
delete

 2008-07-18 13:12:05,713 [main] TRACE (Main.java:29) - Akan menghapus World -7

 2008-07-18 13:12:05,713 [main] WARN (HelloWorldService.java:22) - Method
deleteTheWorld dijalankan

 2008-07-18 13:12:05,713 [main] INFO (HelloWorldDao.java:13) - Menjalankan method
delete

 2008-07-18 13:12:05,715 [main] ERROR (HelloWorldService.java:26) - ID tidak
boleh kurang dari nol

 java.lang.IllegalArgumentException: ID tidak boleh kurang dari nol

        at
com.artivisi.quickstart.log4j.dao.HelloWorldDao.delete(HelloWorldDao.java:20)

        at
com.artivisi.quickstart.log4j.service.HelloWorldService.deleteTheWorld(HelloWorld
Service.java:24)

        at com.artivisi.quickstart.log4j.Main.main(Main.java:30)
```

Filtering berdasarkan Level

Setelah berhasil mengarahkan log message ke berbagai tujuan, dalam hal ini konsol dan file, sekarang kita akan melakukan filtering berdasarkan level. Kita ganti baris pertama konfigurasi dari seperti ini:

```
log4j.rootLogger=INFO,Konsole,Roll
```

menjadi seperti ini:

```
log4j.rootLogger=WARN,Konsole,Roll
```

dengan konfigurasi seperti ini, hanya log message WARN dan ERROR yang akan dicetak. Berikut output yang muncul di konsol.

```
 2008-07-18 13:16:39,425 [main] WARN
com.artivisi.quickstart.log4j.service.HelloWorldService  - Method deleteTheWorld
dijalankan

 2008-07-18 13:16:39,428 [main] WARN
com.artivisi.quickstart.log4j.service.HelloWorldService  - Method deleteTheWorld
dijalankan

 2008-07-18 13:16:39,430 [main] ERROR
com.artivisi.quickstart.log4j.service.HelloWorldService  - ID tidak boleh kurang
dari nol

 java.lang.IllegalArgumentException: ID tidak boleh kurang dari nol

       at
com.artivisi.quickstart.log4j.dao.HelloWorldDao.delete(HelloWorldDao.java:20)

       at
com.artivisi.quickstart.log4j.service.HelloWorldService.deleteTheWorld(HelloWorld
Service.java:24)

       at com.artivisi.quickstart.log4j.Main.main(Main.java:30)
```

Tidak ada log message INFO, DEBUG, dan TRACE.

Filtering berdasarkan kategori/package

Untuk melakukan filtering berdasarkan kategori, kita bisa tambahkan baris konfigurasi berikut.

```
log4j.logger.com.artivisi.quickstart.log4j=TRACE

log4j.logger.com.artivisi.quickstart.log4j.service=DEBUG

log4j.logger.com.artivisi.quickstart.log4j.dao=INFO
```

Ini akan mengatur output log per package dalam aplikasi Java yang kita lihat. Silahkan jalankan Main class untuk melihat efeknya.

V. Penutup

Demikianlah penjelasan singkat tentang konsep dan penggunaan log4j. Log message yang dibuat dengan cermat akan memudahkan development, tuning, monitoring, dan troubleshooting aplikasi. Aplikasi yang berkualitas tinggi terlihat dari log message yang dikeluarkannya.