Java naplózás megvalósítása Jakarta Log4J-vel

Viczián István (viczus@freemail.hu)

E cikk olyan Java programozóknak nyújt segítséget, kik tisztában vannak a Java nyelv alapjaival, és egy megbízható, elterjedt, jól dokumentált és támogatott naplózó rendszerre van szükségük. A Log4J projekt mindezen jellemzőkkel rendelkezik, és szoros kapcsolatban áll a Jakarta projekt többi tagjával, de persze önállóan is használható.

A naplózásra szinte minden szoftver fejlesztésekor szükség van, a legkisebbtől kezdve a komplex, nagyvállalati környezetben használatos, elosztott alkalmazásokig. Nem csupán hibakeresési lehetőség, hanem használható monitorozásra, teljes naplózásra, különböző tesztek sikerességének ellenőrzésére. A naplózás és debug nem keverendő össze, inkább kiegészítik egymást. Van olyan eset, mikor a debugger-ek nem alkalmazhatók, vagy használatuk körülményes lenne, ilyenkor jól jön a naplózás. A naplózás jóval több, mint `System.out.println` utasítások, elvárásunk lehet, hogy az eseményekhez pontos dátum kapcsolódjon, a különböző szoftverkomponensek üzenetei megkülönböztethetőek legyenek egymástól, szűkíteni vagy bővíteni lehessen a naplózást különböző szintek bevezetésével, a formátum és cél tetszőlegesen konfigurálható legyen (pl. fájl, adatbázis, más hálózati erőforrás).

A Log4J egy minden igényt kielégítő naplózó rendszer, mely a felsorolt kívánalmaknak maximálisan megfelel, és egyéb hasznos tulajdonságai is vannak. Bevezeti a loggerek fa hierarchiáját (hasonlóan a csomagokhoz), így naplózáskor kiválaszthatunk bármilyen részfát, és külön konfigurálhatjuk ezek tulajdonságait. A beállításokat a szülő loggerre kell megadni, és ezt minden gyermeke örökli.

A Log4J a többi Apache projecthez hasonlóan szabad forráskódú, és az Apache Software Licence hatálya alá tartozik, ami tulajdonképpen azt jelenti, hogy bárhol felhasználható, csupán a dokumentációba meg kell említeni a Log4J nevét, és csatolnunk kell a licence fájlt.

A Log4J fejlesztésekor különösen figyeltek arra, hogy a naplózás kikapcsolásakor ne kelljen a forráskódot megváltoztatni, és mégse legyen tapasztalható észrevehető sebességkülönbség egy log nélküli kódhoz képest. Mivel a Java nyelv alapvetően nem rendelkezik előfordítóval (, persze léteznek alternatív megoldások), a log üzeneteknek mindenképp szerepelniük kell a forráskódban, elrontva kicsit annak olvashatóságát. Ezt sajnos számításba kell vennünk, és ésszerűen megszabni a log üzenetek és a tényleges kód mennyiségének az arányát.

A Log4J egy szöveges állományban konfigurálható, nincs szükség a forrás módosítására.

A Log4J fő fejlesztője Ceki Gülcü, és a honlapon rövid dokumentációt, cikkeket, prezentációkat, API dokumentációt is találhatunk, illetve a különböző verziók binárisát illetve forrását. Sajnos a teljes dokumentáció csak kereskedelmi forgalomban kapható nyomtatott könyv. A cikk írásakor az aktuális verzió az 1.2.8-as, de nem szándékom olyan mélyen belemenni, hogy bármilyen verzió specifikus információval szolgáljak.

A Log4J három fő komponensei a loggerek, appenderek és a layoutok. A loggerek használandóak a forrásban és fa hierarchiát építenek fel. Az appenderek biztosítják a logok perzisztenciáját, tehát írnak fájlba, konzolra, socket-re. A layout-ok felelősek a log formázásáért, megjelenéséért.

A Log4J lehetővé teszi kategóriák kialakítását, és azok kiválasztását, hogy csak a minket érdeklő debug üzenetek jelenjenek meg. A csomagokhoz hasonlóan egyedi névvel rendelkeznek, és kis és nagybetű érzékenyek. Minősíteni is hasonlóan kell, tehát a szülő után

pontot téve kell írnunk a gyermek nevét. Pl. ha a szülő `szulo` nevű, akkor annak egyik közvetlen leszármazottja lehet a `szulo.gyermek` nevű.

Az 1.2-es verziótól kezdve `Logger` osztály használatos a `Category` osztály helyett.

Létezik egy gyökér kategória is, mely nem rendelkezik névvel, és mindig létezik. Így lekérdezni név alapján nem is lehet, csupán kódból a `Logger.getRootLogger` statikus metódussal. Kódból a loggereket lekérdezni a `Logger.getLogger` metódussal lehet.

Egy loggerhez szinteket lehet rendelni, név szerint a `DEBUG, INFO, WARN, ERROR` és `FATAL`, melyek az `org.apache.log4j.Level` osztályban vannak deklarálva. A felsorolás sorrendje mutatja a szintek sorrendjét is. A `DEBUG` az összes szintet tartalmazza, míg a `FATAL` nem tartalmaz egy más szintet sem. A gyökér loggernek mindig meg kell adni szintet, míg a többi logger örökölheti azt a szülőjétől.

Amikor naplózni szeretnénk a programban, a `debug, info, warn, error, fatal` és `log` metódusok valamelyikét használhatjuk. Egy ilyen kiíró metódusnak akkor lesz csak hatása, ha a loggernek megadott szint ugyanaz, vagy magában foglalja a hozzá tartozó szintet. Azaz ha `debug` metódust használunk, akkor csak akkor jelenik meg, ha a logger szintje is `DEBUG`. A `fatal` mindig megjelenik. Egy `warn`-nal hívott kiírás csak akkor jelenik meg, ha a logger szintje `WARN, ERROR` vagy `FATAL`.

A `Logger.getLogger` metódus adja vissza a kódban a megadott loggert, név alapján. Ha a név ugyanaz, akkor a metódus mindig ugyanaz a példányt adja vissza.

A legelterjedtebb stratégia a loggerek elnevezésére az osztályhierarchia pontos követése, és a logger deklarálása statikus változóként.

```
package hu.log4jtest;

import org.apache.log4j.Logger;

public class Test {
      volatile private static Logger ourLogger =
Logger.getLogger(Test.class.getName());

      public static void main(String[] args) {
            ourLogger.debug("Hello Log4J!");
      }
}
```

Ebben az esetben a logger neve `hu.log4jtest.Test` lesz. Fordításhoz nem kell mást tennünk, mint a Log4j JAR fájlját (jelenleg `log4j-1.2.8.jar`) elhelyezni a `CLASSPATH`-ban.

Ebben az esetben még nem vagyunk kész az alkalmazásunkkal, hiszen egy konfigurációs fájlt is meg kell adni, melyben a szinteket, appendereket és layoutokat is konfigurálni kell. Futtatáskor a következő hibaüzenetet kapjuk:

```
log4j:WARN No appenders could be found for logger (hu.log4jtest.Test).
log4j:WARN Please initialize the log4j system properly.
```

Appenderek a következő helyekre írhatnak: konzolra, fájlba, GUI komponensre, socket szerverre, JMS-t (Java Messaging Service) igénybe véve, NT event logba, és UNIX Syslog demonnak. Lehetséges asszinkron appender használata is, ahol egy külön szál gyűjti az üzeneteket, és puffereli őket. Egy loggerhez több appender is kapcsolható, pl. az `addAppender` metódussal. Alapesetben az összes appendernek elmegy az üzenet, mely egy adott loggerhez, illetve annak szüleihez van kapcsolva. Ez a működés átdefiniálható.

Az üzenet formáját a layout határozza meg. A layoutot egy minta segítségével konfigurálhatjuk, ami egy szöveges sor, vezérlő karakterekkel. A pontos leírását a PatternLayout osztály API dokumentációja tartalmazza. Egy lehetséges beállítás:

```
%4d{dd MMM yyyy HH:mm:ss,SSS} %-5p - %c: %m\n
```

Kiírja a dátumot, időt ezredmásodperc pontossággal, az üzenet osztályát és kategóriáját, valamint az üzenetet és egy sortörés karaktert:

```
31 máj. 2003 16:20:13,875 DEBUG - hu.log4jtest.Test: Hello Log4J!
```

A konfigurációs fájl lehet XML vagy property (név=érték párok) fájl formátumú. A Log4J alapesetben a `CLASSPATH`-ban keres egy `log4j.properties` nevű fájlt, de megadhatjuk neki explicit is annak helyét.

```
import org.apache.log4j.PropertyConfigurator;
...
PropertyConfigurator.configure(logFilePath);
```

Legyen a konfigurációs fájl a következő:

```
# Megjegyzések.
# A gyökér loggernek beállítja a szintjét FATAL-ra, és megadja neki az A1
# nevű loggert
log4j.rootLogger=FATAL, A1

# Az appender konzolra dolgozik.
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# A appender fájlba ír.
# log4j.appender.A1=org.apache.log4j.FileAppender
# log4j.appender.A1.File=test.log
# log4j.appender.A1.Append=false

# Layout beállítása.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
# Layout minta beállítása.
log4j.appender.A1.layout.ConversionPattern=%4d{dd MMM yyyy HH:mm:ss,SSS} %-
5p - %c: %m\n

# Kategóriák
log4j.category.hu.log4jtest.Test=DEBUG
```

A konfigurációs állományban, ha a `FileAppender`-t alkalmazzuk, az eredmény a `test.log` fájlban fog megjelenni.

Az öt naplózó metódusnak mindegyiknek van olyan párja is, mikor egy `Throwable` interfészt implementáló osztályt adhatunk meg paraméterként, így bármilyen kivétel kiírása is lehetővé válik.

Természetesen a szervletekben is használhatjuk a Log4J-t, ilyenkor ajánlott az `init()` metódusban konfigurálni azt, és a szervletet a konténer indulásakor azonnal betölteni. A konfigurációs fájl helyét ilyenkor célszerűen a `web.xml` fájlban tároljuk.

Lássuk az inicializáló szervlet forrását:

```
package hu.log4jtest;

import org.apache.log4j.PropertyConfigurator;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Log4jInit extends HttpServlet {

  public void init() {
    String prefix = getServletContext().getRealPath("/");
    String file = getInitParameter("log4j-init-file");
    PropertyConfigurator.configure(prefix + file);
  }
}
```

És a hozzá tartozó web.xml bejegyzés:

```
  <servlet>
    <servlet-name>log4j-init</servlet-name>
    <servlet-class> hu.log4jtest.Log4jInit</servlet-class>

    <init-param>
      <param-name>log4j-init-file</param-name>
      <param-value>WEB-INF/classes/log4j.lcf</param-value>
    </init-param>

    <load-on-startup>1</load-on-startup>
  </servlet>
```

A Log4J portolva lett más nyelvekre és platformokra is, mint C, C++, Eiffel, Perl, LotusScript, .NET, PHP, Python, PL/SQL, Qt/C++, Ruby. Létezik JDBC appender, mellyel az adatbázisba naplózás oldható meg, létezik J2ME verziója, mely mobil környezetben használatos, van kiterjesztése webes alkalmazásokhoz és JUnit unit teszthez. Létezik JSP-hez taglib, és alternatív grafikus felületek is.
A Log4J egyszerűen használható, elterjedt, könnyen konfigurálható naplózó keretrendszer, mely a naplózással kapcsolatos minden igényünket kielégíti, és a megszokott Jakarta minőséget garantálja, mind kóddal, áttekinthetőséggel, dokumentációval kapcsolatban.

Budapest, 2003. május 31.

Hivatkozások

The Apache Software Foundation http://www.apache.org
The Jakarta Project http://jakarta.apache.org
Log4J http://jakarta.apache.org/log4j/docs/index.html

Szerzőről

Viczián István a Debreceni Egyetem programtervező matematikus szakán végzett 2001 nyarán, ahol most levelező PhD. hallgató az elosztott rendszerek, middleware-ek témakörében. Jelenleg szoftver fejlesztőként dolgozik Budapesten, melynek keretében middleware szoftverekkel, alkalmazásintegrációval, webes technológiákkal valamint application mining-gal és reverse enginering-gel foglalkozik. Szabadidejében internetes portált fejleszt, és Java technológiákkal ismerkedik, valamint Java blog-ot ír (JTechLog).
E-mail: viczus@freemail.hu
Honlap: http://dragon.unideb.hu/~vicziani