Kurzanleitung Logging

Bei so gut wie allen größeren Programmen ist es nötig, Log-Ausgaben zu erzeugen. Als Log-Ausgaben werden Ausgaben verstanden, die dazu genutzt werden können, um Fehler im Programm aufzudecken oder den aktuellen Fortschritt darzustellen.

1 Java-Logging

Seit Java-Version 1.4 gibt es im Framework eine Komponente, die zum Loggen verwendet werden kann. Die Konzepte dieser Komponente stammen aus dem log4j-Projekt[1], das sich immer noch großer Beliebtheit erfreut. Da das Java-Logging Teil des Frameworks ist, ist keine separate Installation nötig. Eine Beschreibung und Referenz findet sich unter [2].

Die Idee hinter Java-Logging ist bestechend einfach: Der Programmierer sagt nur, was er mit welcher Priorität loggen will. Alles weitere übernimmt das Framework. Die Konfiguration kann man zur Laufzeit des Programms festlegen: Je nachdem, welche Log-Einträge man benötigt beziehungsweise an welcher Stelle man diese ausgegeben haben möchte.

2 Verwendung

Um Java-Logging in einem Programm zu verwenden, muss zunächst ein „Logger“-Objekt angelegt werden. Dies ist eine Instanz der Klasse `java.util.logging.Logger`, welche statisch in der Klasse vorhanden sein muss. Einen Logger kann man nur mit den statischen Methoden `getLogger(String name)` dieser Klasse erzeugen. Der Parameter `name` gibt dabei einen beliebigen Namen für diese Logger-Instanz an. Ein Beispiel zeigt das folgende Listing.

```
1  package fh.swt;
2
3  import java.util.logging.Logger;
4
5  public class GameLoop {
6
7      /**
8       * Ein Java-Logging-Logger.
9       */
10     private static Logger log = Logger.getLogger(GameLoop.class.getName());
11
12     // Hier kommen Methoden...
13 }
```

Listing 1: Anlegen eines Loggers

Mit einem auf diese Weise erzeugten Logger können nun Log-Ausgaben erzeugt werden. Dazu werden die Methoden des erzeugten Loggers verwendet. Jede Log-Ausgabe hat eine von sechs Prioritäten. Die möglichen Prioritäten zeigt die folgende Tabelle in aufsteigender Reihenfolge.

Priorität	Beschreibung
FINEST	detailliertere Ausgabe als FINER (zum Beispiel Start und Ende jeder Methode)
FINER	detailliertere Ausgabe als FINE
FINE	Ausgabe von wichtigen Schritten im Programmfluss
CONFIG	Ausgabe von Information über eine Konfiguration (zum Beispiel die Art des Betriebssystems oder des CPU-Typs)
INFO	allgemeine Informationen (zum Beispiel wurde eine wichtige Aktion erfolgreich abgeschlossen)
WARNING	es ist ein Fehler aufgetreten (zum Beispiel wurde eine Exception gefangen und nun behandelt)
SEVERE	kritischer Fehler, der dazu führt, dass das Programm nicht ordnungsgemäß fortgesetzt werden kann, eventuell Programmabbruch

Es hat sich gezeigt, dass nicht zwingend alle Prioritäten verwendet werden müssen. Oft reicht es, alle Ausgaben zum Programmfluss in einer Priorität (zum Beispiel FINE) auszugeben. Wichtig ist nur, dass man sich vor dem Beginn der Implementierung Gedanken um die zu verwendenden Stufen macht und dies auch einheitlich durchhält.

Für jede Priorität gibt es eine Methode in dem Logger-Objekt. So wird eine Ausgabe, die mit der Priorität FINE versehen werden soll mit der Methode `fine(String msg)` erzeugt. Die Methode nimmt die Nachricht als Parameter `msg` entgegen. Das Folgende Listing zeigt ein paar Beispiel für die Aufrufe der Methoden.

```
1  public void doLoop() {
2    log.finest("Begin: doLoop");
3    // ...
4    long timeInMills = 0;
5    while (timeInMills < 10) {
6      log.fine("Loop Start: " + timeInMills);
7      try {
8        timeInMills++;
9        // ...
10     } catch (Exception e) {
11       log.severe("Exception in doLoop");
12       break;
13     }
14   }
15   log.finest("Ende : doLoop");
16 }
```

Listing 2: Aufruf der Log-Methoden

3 Laufzeitkonfiguration

Bisher wurde lediglich dargestellt, wie man Log-Ausgaben erzeugt. Diese wurden dabei an das Framwork übergeben und anschließend nicht weiter beachtet. Wird nicht explizit eine Konfiguration angegeben, so werden alle Meldungen ab Loglevel INFO auf die Standardfehlerausgabe ausgegeben.

Wird eine andere Ausgabe gewünscht kann man Java-Logging auf verschiedene Weisen konfigurieren. So kann dies mit Hilfe einer Konfigurationsdatei, aber auch durch Anweisungen im Programm erfolgen. Hier soll lediglich die Variante mit Hilfe einer Konfigurationsdatei gezeigt werden.

Die Konfigurationsdatei ist eine Datei im Properties-Format. In dieser Datei werden unter anderem die „Handler“ eingestellt. Ein Handler ist für die eigentliche Ausgabe zuständig. So gibt der `ConsoleHandler` die Log-Ausgaben auf die Konsole aus, während ein `FileHandler` sie in eine Datei schreibt. Es gibt noch weitere Handler, wobei jeder einen eigenen Satz an Einstellungsmöglichkeiten besitzt die unter [2] beschrieben sind. Der Schlüssel, der angibt welche Handler zum Einsatz kommen, heißt `handlers`. Hier können auch mehrere Handler angegeben werden, um die Meldungen zum Beispiel sowohl auf die Konsole als auch in eine Datei zu schreiben.

Ein verpflichtender Schlüssel in der Konfigurationsdatei ist die Priorität, ab der Nachrichten ausgegeben werden. Dazu wird die Einstellung `.level` auf die gewünschte Priorität gesetzt. Alle Nachrichten, die diese oder eine höhere Priorität besitzen, werden ausgegeben.

Bisher kann die benötigte Priorität nur mit Hilfe der Einstellung `.level` angegeben werden. Entwickelt man in einem großen Programm ein kleines Modul, so möchte man sicher nicht alle Meldungen zu sehen bekommen, sondern nur diejenigen, die das entsprechende Modul auch betreffen. Dazu kann man zusätzlich zu den bereits genannten Einstellungen auch jeden einzelnen Logger präzise Konfigurieren. So könnte man in diesem Beispiel die Standardpriorität auf `SERVERE` und die Priorität des entsprechenden Loggers auf `FINEST` stellen.

Das Folgende Listing zeigt eine einfache Konfigurationsdatei die alle oben genannten Einstellungen zeigt.

```
1  # Über diese Handler werden die Meldungen ausgegeben
2  handlers = java.util.logging.FileHandler, java.util.logging.ConsoleHandler
3
4  # Der Standard-Loglevel
5  .level = SERVERE
6
7  # Einstellungen für die einzelnen Handler
8  java.util.logging.FileHandler.pattern = game.log
9  java.util.logging.FileHandler.formatter = java.util.logging.XMLFormatter
10
11 java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
12
13 # Einstellungen für die einzelnen Logger
14 fh.swt.Main.level = FINEST
```

Listing 3: Konfigurationsdatei für Java-Logging

Jetzt ist zwar eine Konfigurationsdatei für Java-Logging vorhanden, diese wird aber nicht automatisch verwendet. Um dies zu erreichen, gibt es zwei Möglichkeiten: Die Konfigurationsdatei kann mit Hilfe eines Parameters beim Start des Programms angegeben werden, oder sie wird vom Programm explizit geladen.

Im ersten Fall wird folgender Parameter an die virtuelle Maschine übergeben:

`-Djava.util.logging.config.file=/pfad/zur/datei/logging.properties`

Diese liest dann die Konfigurationsdatei ein und wendet die darin enthaltenen Einstellungen an.

Im zweiten Fall muss im Programm eine Zeile eingefügt werden. Ein `LogManager` ist für die Verwaltung der Logger zuständig und kann eine Konfigurationsdatei einlesen und anwenden. Wird die oben gezeigte Konfigurationsdatei als `logging.properties` im aktuellen Verzeichnis abgespeichert, so funktioniert das im folgenden Listing abgedruckte Programm.

```java
package fh.swt;

import java.io.FileInputStream;
import java.util.logging.LogManager;
import java.util.logging.Logger;

public class Main {

    private static Logger log = Logger.getLogger(Main.class.getName());

    public static void main(String[] args) throws Exception {
        LogManager.getLogManager().readConfiguration(new FileInputStream("logging.properties"));
        log.info("Game gestartet...");

        GameLoop gameLoop = new GameLoop();
        log.finest("GameLoop erzeugt.");
        gameLoop.doLoop();

        log.info("Game ist fertig.");
    }
}
```

Listing 4: Konfigurieren des Java-Logging-Frameworks

Literatur

[1] Apache Foundation. *log4j*, 2008. URL `http://logging.apache.org/`.

[2] SUN. *Java Logging APIs*. URL `http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/`.

Autor: Malte Ried, Institut für SoftwareArchitektur.