10 Loggning

10.1 Loggning
10.2 Att skapa loggmeddelanden
10.3 Fördefinierade loggningsnivåer
10.4 Klassen Logger – metoder
10.5 Konfiguration av loggning
10.7 Fördefinierade loggningshanterare

Loggning

Fr o m Java 1.4 finns stöd i standardbiblioteket för *loggning*:

- Användbart för felrapportering, felsökning, spårning och driftövervakning
- Störst nytta i servermiljö och i program utan grafiska användargränssnitt
- Stödet finns i namnrymden `java.util.logging`
- Loggmeddelanden delas in i sju kategorier via loggningsnivåer
- Det går att konfigurera vilka kategorier som ska loggas
- Loggning kan ske till
  - ➜ skärmen (kommandoskal)
  - ➜ en fil
  - ➜ en socket i nätverk
- Loggmeddelanden kan lämnas i olika format
  - ➜ Ren text – praktiskt för människor som ska läsa loggad information
  - ➜ XML – enklare för program som ska bearbeta loggad information
- Loggning kan konfigureras individuellt per program utan omkompilering

Fr o m version 1.4 av Java innehåller standardbiblioteket ett ramverk för *loggning*, dvs en teknik för att på ett enhetligt sätt samla upp och tillhandahålla loggmeddelanden som Java-program genererar.

Loggning kan ha många användningsområden: den kan bilda grunden i ett system för felrapportering, den kan användas för avlusning och spårning i samband med felsökning i program och den kan ge underlag för driftövervakning av program. Alla program kan använda sig av loggning, men nyttan är störst i serverprogram som ofta körs i obemannad och långvarig drift, samt överhuvudtaget i program som saknar grafiska gränssnitt och därför inte lätt kan presentera felmeddelanden i dialoger.

Biblioteksstödet består av klasser och gränssnitt i namnrymden `java.util.logging`. Varje loggmeddelande knyts till en nivå som anger hur kritiskt meddelandet är. Standardbibliotekets ramverk för loggning arbetar med sju nivåer, ordnade i en skala från minst till mest kritiskt. Loggningen utförs med hjälp av meddelandesändningar i koden, varefter ramverket bearbetar de meddelanden som genererats.

En central egenskap i ramverket är att det finns möjligheter att konfigurera hur loggningen ska gå till:

- Det går att välja vilka nivåer av meddelanden som ska loggas, så att endast meddelanden av minst en viss nivå tas med. Loggmeddelanden av lägre nivå kan därmed stanna kvar i källkoden för framtida behov, utan att de för den skull skapar floder av utskrifter.
- Loggning kan ske till olika media: till skärmen (som regel ett kommandoskal som programmet startas från), till en eller flera filer, eller till en socket över ett nätverk.
- Loggmeddelanden kan presenteras i ren text, vilket passar bra när människor ska läsa dem, eller i XML, vilket lämpar sig bättre om ett program ska bearbeta dem.

Loggningen kan konfigureras individuellt per program, och kan ändras utan omkompilering av koden.

Loggning

Att skapa loggmeddelanden

Parser.java

```
import java.util.logging.*;

public class Parser {

    public static void main(String[] args) {
        Logger appLogger = Logger.getLogger("se.bohel.tools.Parser");
        if (args.length < 1) {
            appLogger.severe("Command-line file name missing");
            System.exit(1);
        }
        else {
            String filename = args[0];
            try {
                processFile(filename);
            }
            catch (IOException e) {
                appLogger.log(Level.SEVERE,
                              "I/O exception while processing file",
                              e);
                System.exit(2);
            }
        }
    }
}
```

All loggning sker via ett *loggningsobjekt*. Det finns ett globalt loggningsobjekt som kan användas, men det är nästan lika enkelt att skapa ett specifikt per applikation, vilket öppnar möjligheter att skräddarsy hur loggningen ska fungera per applikation.

Ett applikationsspecifikt loggningsobjekt får vi via klassmetoden `getLogger` hos klassen `Logger` i namnrymden `java.util.logging` – den namnrymd där alla klasser och gränssnitt i ramverket för loggning finns. Metoden `getLogger` ska ha ett argument, och det ska vara antingen en namnrymd eller ett fullständigt specificerat klassnamn (dvs med namnrymder) som är unikt för applikationen. Den första gången metoden `getLogger` anropas med ett visst namn, skapas ett nytt loggningsobjekt, men om samma argument används på nytt, returneras det redan befintliga objektet. I vårt fall använder vi det fullständigt specificerade klassnamnet:

```
Logger appLogger = Logger.getLogger("se.bohel.tools.Parser");
```

Den generella metoden för loggning av meddelanden hos loggningsobjektet är `log`. Det finns ett antal överlagrade versioner av `log`. I bildens nedre del utnyttjas en version med tre argument. Den passar bra för loggning av språkunderstödda fel. De tre argumenten är loggningsnivån, meddelandets text samt det felbeskrivande objektet.

Loggningsnivån beskriver hur allvarligt felet är. I vårt fall använder vi loggningsnivån `Level.SEVERE` (en klasskonstant från klassen `Level` i namnrymden `java.util.logging`) för att signalera ett allvarligt fel som avbrutit programmets exekvering.

När vi enbart vill logga ett meddelande med en loggningsnivå och en text, kan vi använda oss av bekvämlighetsmetoder som t ex `severe`, som loggar ett meddelande på nivån `Level.SEVERE`.

Loggning

Fördefinierade loggningsnivåer

Loggningsnivå	*Innebörd*
`Level.SEVERE`	Ett allvarligt fel som förhindrar normal exekvering; riktat till systemadministratörer eller användare
`Level.WARNING`	Ett potentiellt problem som inte förhindrar normal exekvering; riktat till systemadministratörer eller användare
`Level.INFO`	Ett informationsmeddelande som inte rör ett fel; riktat till systemadministratörer eller användare
`Level.CONFIG`	Ett meddelande som beskriver systemets konfiguration (operativsystem, fönsterhanterare, databas, drivrutiner etc); riktat till systemadministratörer eller användare
`Level.FINE`	Ett meddelande för grovskalig spårning av exekvering; riktat till utvecklare
`Level.FINER`	Ett meddelande för mer detaljerad spårning av exekvering, t ex när programmet påbörjar resp avslutar en metod; riktat till utvecklare
`Level.FINEST`	Ett meddelande för mycket detaljerad spårning; riktat till utvecklare

Standardbibliotekets ramverk för loggning har sju fördefinierade *loggningsnivåer*. De fyra högsta nivåerna är för meddelanden som riktas till systemadministratörer och användare, och bör därför formuleras så att en icke utvecklingskunnig person kan förstå dem:

- Nivån `Level.SEVERE` är den högsta (dvs mest allvarliga) nivån. Den används för allvarliga fel som förhindrar normal exekvering av programmet. Motsvarande bekvämlighetsmetod är `severe`.
- Nivån `Level.WARNING` används för att ge varningar om potentiella fel som inte förhindrar normal exekvering av programmet. Motsvarande bekvämlighetsmetod är `warning`.
- Nivån `Level.INFO` används för att lämna information om programmets exekvering som inte har att göra med fel. Motsvarande bekvämlighetsmetod är `info`.
- Nivån `Level.CONFIG` används för att lämna information om systemets konfiguration, t ex namn och version av operativsystem, databas, drivrutiner etc. Motsvarande bekvämlighetsmetod är `config`.

De tre lägsta nivåerna är riktade till utvecklare, och används för spårning av exekveringen i ett program:

- Nivån `Level.FINE` används för grovskalig spårning, t ex resultat av större operationer, mindre fel som programmet själv kunde hantera, etc. Motsvarande bekvämlighetsmetod är `fine`.
- Nivån `Level.FINER` används för mer detaljerad spårning av ett programs exekvering, t ex meddelanden om när exekveringen av en metod inleds resp avslutas, eller när språkunderstödda fel genereras. Motsvarande bekvämlighetsmetod är `finer`.
- Nivån `Level.FINEST` är den lägsta nivån. Den används för mycket detaljerad spårning, t ex när en viss punkt inuti en metod nås. Motsvarande bekvämlighetsmetod är `finest`.

Loggning

Klassen Logger – metoder

Metod	*Beskrivning*
`log(level, message)`	Logga meddelandet `message` på nivån `level`
`log(level, message, exc)`	Logga meddelandet `message` med felet `exc` på nivån `level`
`severe(message)`	Logga meddelandet `message` på nivån `SEVERE`
`warning(message)`	Logga meddelandet `message` på nivån `WARNING`
`info(message)`	Logga meddelandet `message` på nivån `INFO`
`config(message)`	Logga meddelandet `message` på nivån `CONFIG`
`fine(message)`	Logga meddelandet `message` på nivån `FINE`
`finer(message)`	Logga meddelandet `message` på nivån `FINER`
`finest(message)`	Logga meddelandet `message` på nivån `FINEST`
`entering("class", "method")`	Logga meddelandet att exekveringen av metoden `"method"` i klassen `"class"` påbörjas
`exiting("class", "method")`	Logga meddelandet att exekveringen av metoden `"method"` i klassen `"class"` avslutas
`throwing("class", "method", exc)`	Logga meddelandet att metoden `"method"` i klassen `"class"` avbrutits av felet `exc`

Klassen `Logger` innehåller en stor uppsättning av metoder som kan användas för att logga meddelanden:

- Den grundläggande loggningsmetoden är `log`, som loggar ett givet meddelande på en specificerad loggningsnivå. Det finns ett antal överlagrade varianter av metoden, där några har extra argument som t ex referensen till ett felobjekt som hänger samman med meddelandet. De motsvarande metoderna `logp` resp `logrb` ger möjligheter att manuellt påverka hur meddelandet loggar var i koden det genererats, resp hur det ska anpassas nationellt.
- Det finns ett antal metoder med namn som `severe`, `warning`, `info` etc, som gör det lättare att logga ett enkelt meddelande på en viss loggningsnivå.
- Metoden `entering` gör det lätt att logga ett spårmeddelande på nivån `Level.FINER` om att exekveringen går in i en specifik metod. Metodens argument är klass- och metodnamn i form av textsträngar. Det finns överlagrade varianter som kan ta med metodens argument (ett eller flera) i meddelandet.
- Metoden `exiting` gör det lätt att logga ett spårmeddelande på nivån `Level.FINER` om att exekveringen lämnar en specifik metod. Metodens argument är klass- och metodnamn i form av textsträngar. Det finns en överlagrad variant som kan ta med metodens returvärde i meddelandet.
- Metoden `throwing` gör det lätt att logga ett spårmeddelande på nivån `Level.FINER` om att exekveringen av en metod är på väg att avbrytas genom att ett språkunderstött fel genereras. Metodens argument är klass- och metodnamn i form av textsträngar, samt en referens till felobjektet för det aktuella felet.

Loggning

Konfiguration av loggning

logging.properties

```
# Global properties.

# By default we only configure a ConsoleHandler, which will only
# show messages at the INFO and above levels.
handlers= java.util.logging.ConsoleHandler

# To also add the FileHandler, use the following line instead.
#handlers= java.util.logging.FileHandler, java.util.logging.ConsoleHandler

# Default global logging level.
.level= INFO

# Handler specific properties.

# Default file output is in user's home directory.
java.util.logging.FileHandler.pattern = %h/java%u.log
java.util.logging.FileHandler.limit = 50000
java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.formatter = java.util.logging.XMLFormatter

# Limit the messages that are printed on the console to INFO and above.
java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
```

Hur loggningen ska fungera kan konfigureras genom att ändra filen `logging.properties`, som fr o m **Java SE 9** finns i katalogen `conf` i Javas installationskatalog (t o m **Java SE 8** i underkatalogen `lib` under katalogen `jre`). Standardutseendet på denna fil visas här i något förkortat skick.

Loggmeddelanden kan i ramverket skickas till en eller flera *loggningshanterare*, som var och en kan avgöra om de vill hantera meddelandet och hur det ska ske. I den första sektionen av konfigurationsfilen sätts egenskapen `handlers` till en kommaseparerad lista av aktiva loggningshanterare. I vårt fall är endast en hanterare av klassen `ConsoleHandler` från namnrymden `java.util.logging` aktiv. Detta är den hanterare som skickar loggmeddelanden till streamobjektet `System.err`, vilket normalt innebär att de dyker upp i det kommandoskal som programmet startats från (såvida inte utskrifter till `System.err` dirigerats om).

I en utkommenterad rad (kommentarer inleds med tecknet #) visas hur en loggningshanterare av klassen `FileHandler` (från samma namnrymd) kan läggas till. Med dess hjälp kan loggmeddelanden samlas till en eller flera textfiler.

Därnäst beskrivs (som egenskapen `.level`) vilken som är den minsta loggningsnivå som ramverket bryr sig om att förmedla vidare för bearbetning – i detta fall nivån `Level.INFO`. Loggmeddelanden av lägre nivå sållas bort på ett tidigt stadium och når aldrig loggningshanterarna. Det blir därför inte kostsamt att ha dessa meddelanden kvar i koden.

Därefter konfigureras varje hanterare för sig. Vi kan bl a ange en egen gräns för vilka loggningsnivåer den bryr sig om. I exemplet visas hur (den icke aktiva) loggningshanteraren `FileHandler` konfigureras så att loggning sker i XML-format till en enda fil (dvs inte till en roterande uppsättning av filer) som inte får växa till mer än 50 000 teckens storlek, och som placeras i användarens hemkatalog och där garanteras få ett unikt namn av formen `java0.log`, `java1.log`, etc.

Avslutningsvis konfigureras den aktiva loggningshanteraren `ConsoleHandler` till att generera utskrifter i enkelt textformat för loggmeddelanden av minst nivån `Level.INFO` (det sistnämnda är onödigt, så länge det överensstämmer med den generella begränsningen).

Om vi inte vill ändra den centrala konfigurationsfilen `logging.properties`, kan vi i stället skapa en egen konfigurationsfil som bara används när vi exekverar det här programmet. En sådan konfigurationsfil utformas på samma sätt som den centrala konfigurationsfilen och pekas ut av systemegenskapen `java.util.logging.config.file` när vi startar programmet:

```
java -Djava.util.logging.config.file=BookLog.properties BookForm
```

Loggning

Fördefinierade loggningshanterare

Loggningshanterare	*Funktion*
`ConsoleHandler`	Skickar loggmeddelanden för utskrift till `System.err`, dvs normalt till det kommandoskal programmet startats från
`FileHandler`	Samlar loggmeddelanden i en eller flera textfiler, med möjlighet att välja antal och maximal storlek på dessa filer, så att äldre logginformation automatiskt raderas; lämplig att använda i obemannad drift
`SocketHandler`	Skickar loggmeddelanden till en socket i ett nätverk
`StreamHandler`	Gemensam överklass till de tre föregående, lämplig att utgå ifrån för den som vill bygga en egen implementation där loggmeddelanden riktas till ett streamobjekt
`MemoryHandler`	Samlar loggmeddelanden i en minnesbuffert som ständigt innehåller de senaste meddelandena (normalt 1000 stycken); skickar dem vidare till en annan hanterare först när ett meddelande av en miniminivå (normalt SEVERE) uppträder

Standardbibliotekets ramverk för loggning innehåller fem fördefinierade loggningshanterare:

- Hanteraren `ConsoleHandler` är den enda som är aktiv i en standardkonfiguration. Loggmeddelanden av en viss miniminivå (antingen den generella, eller en mer begränsande nivå specifikt satt för denna hanterare) skickas till `System.err` för utskrift, normalt i enkelt textformat.
- Hanteraren `FileHandler` är den normala att använda i obemannad drift. Här samlas loggmeddelanden i en grupp av en eller flera textfiler. Via konfigurationsfilen går det att bestämma om en eller flera filer (där den äldsta hela tiden ersätts enligt ett roterande mönster) ska användas, hur stora filerna får bli, var de ska placeras i filsystemet och hur de ska namnsättas. Hanteraren klarar att ge unika namn till loggfiler, t o m i en fleranvändarmiljö med gemensamt filsystem. Meddelanden förmedlas normalt i XML-format.
- Hanteraren `SocketHandler` skickar loggmeddelanden till en socket i ett nätverk, specificerad via servernamn eller IP-nummer samt portnummer. Meddelanden förmedlas normalt i XML-format.
- `StreamHandler` är den gemensamma överklassen till de tre förstnämnda hanterarna. Den kan användas som utgångspunkt för konstruktion av skräddarsydda hanterare där loggmeddelanden riktas till någon typ av streamobjekt, i antingen enkelt textformat eller XML-format.
- Hanteraren `MemoryHandler` bevarar en rullande uppsättning av de senaste (normalt 1000) loggmeddelandena som mottagits, utan efterbearbetning som formatering eller liknande. Endast om ett loggmeddelande av minst en viss triggernivå (normalt nivån `Level.SEVERE`) tas emot, skickas hela uppsättningen av meddelanden vidare till en annan hanterare. Denna hanterare kan alltså användas för att exempelvis bevara de 1000 senaste mer detaljerade loggmeddelandena (t ex spårmeddelanden) som togs emot *innan* ett kritiskt fel inträffade.