C:\Users\tdqua_000\Dropbox\SS-Slides\DeCuong-CDIO\Template CDIO v4.2\Templates\Hinh anh\LogoTruong.pngĐại học Khoa Học Tự Nhiên TP HCM

Khoa Công nghệ thông tin

Bộ môn Công nghệ phần mềm

**LẬP TRÌNH ỨNG DỤNG JAVA**

**Giảng viên phụ trách:**

**Nguyễn Văn Khiết – Hồ Tuấn Thanh**

**SEMINAR:**

**LOG IN JAVA**

**Nhóm sinh viên:**

1212499 – Trần Trọng Thanh Tùng.

1212505 – Nguyễn Long Uy

1212513 – Lê Quốc Vin

*Thành phố Hồ Chí Minh tháng 03 năm 2015*

MỤC LỤC

[1. Log là gì? 2](#_Toc414872613)

[2. Ý nghĩa của log 2](#_Toc414872614)

[3. Log trong Java 3](#_Toc414872615)

[**A.** **Logger** 3](#_Toc414872616)

[ **Các phương thức log():** 4](#_Toc414872617)

[ **Các phương thức logp():** 4](#_Toc414872618)

[ **Các phương thức logrb()** 5](#_Toc414872619)

[ **Những phương thức log khác** 5](#_Toc414872620)

[**B.** **Logger Hierarchy** 5](#_Toc414872621)

[**C.** **Log levels của các Logger trong Hệ thống** 6](#_Toc414872622)

[**D.** **FORMATTER** 8](#_Toc414872623)

[**E.** **HANDLER** 9](#_Toc414872624)

[ **HANDLER VÀ FORMATER** 9](#_Toc414872625)

[ **PHÂN LOẠI HANDLER** 10](#_Toc414872626)

[***CONSOLEHANDLER:*** 10](#_Toc414872627)

[***FILEHANDLER:*** 10](#_Toc414872628)

[***STREAMHANDLER:*** 12](#_Toc414872629)

[***SOCKETHANDLER:*** 12](#_Toc414872630)

[***MEMORYHANDLER:*** 12](#_Toc414872631)

[**F.** **FILTERS** 13](#_Toc414872632)

[**G.** **LogRecord** 14](#_Toc414872633)

[**H.** **Configuration** 15](#_Toc414872634)

[Cấu hình lớp 15](#_Toc414872635)

[Configuration File 15](#_Toc414872636)

[**I.** **LogManager** 16](#_Toc414872637)

[TÀI LIỆU THAM KHẢO 18](#_Toc414872638)

# Log là gì?

* Log là hành động ghi lại tất cả các sự kiện xảy ra trong hệ điều hành, trong một ứng dụng nào đó hoặc trong trình duyệt vào các tập tin log. Các tập tin log có thể đơn giản chỉ là một tập tin văn bản thông thường. Tập tin log cung cấp các minh chứng về những gì bạn đã làm, bạn đã mở những trang web nào, gửi mail cho ai, down những tập tin nào.
* Những tập tin log này có thể được tìm thấy trong:
* Hệ điều hành
* Trình duyệt web (dưới dạng các tập tin cache)
* Các ứng dụng (dưới dạng các tập tin backup)
* Email

Ví dụ về một tập tin log:

*1/1/2009 - New year*

*1/2/2009 - Date formatted log.*

*2/8/2009 - More log information*

*5/12/2009 - Done with log*

# Ý nghĩa của log

* Log sẽ giúp các nhà phát triển biết và tìm ra cách sửa chữa khi một ứng dụng của họ bị lỗi hoặc bị crash trên máy của người dùng.
* Log giúp người dùng có thể khôi phục lại trạng thái của chương trình nếu họ lỡ tay xóa một cái gì đó hoặc chương trình bị lỗi.
* Log còn giúp người dùng đồng bộ thông tin trên một chiếc điện thoại thông qua phương thức “Backup and restore”.
* Kết hợp với phân tích thống kê, log sẽ giúp các nhà phát triển ứng dụng biết được nhu cầu của người dùng từ đó nâng cao chất lượng sản phẩm tốt hơn, điển hình nhất là tập đoàn Google.
* Hoặc đơn giản log sẽ cho người dùng biết được lịch sử duyệt web của họ, những trang họ đã mở, những tin nhắn họ đã gửi.
* Tuy nhiên, log có thể sử dụng với mục đích xấu là đánh cắp thông tin người dùng bằng cách ghi lại các sự kiện trên máy tính của họ và gửi đến cho các tin tặc.
* Một ví dụ cụ thể : Bạn không thể ngồi hàng giờ, hàng ngày, hàng tháng để đợi ứng dụng chạy và phát sinh ra Exceptions rồi xử lý nó hay bạn không thể bị động khi ứng dụng bán cho khách hàng bị lỗi và gây ra treo hệ thống. Chúng ta cần 1 giải pháp để cập nhật, theo dõi, xử lý sự cố 1 cách nhanh nhất. Đó là sử dụng Log vaf gửi Email khi có lỗi xảy ra. Tôi có Smartphone, laptop, tôi có thể biết được ứng dụng nào đang bị lỗi, hệ thống có vấn đề gì không và tôi có thể khắc phục nó dễ dàng và nhanh nhất mà không bị thụ động về thời gian.

# Log trong Java

* Log trong Java đơn giản là một quá trình ghi lại các log messages trong suốt quá trình thực thi của chương trình về một nơi cố định. Việc ghi log này sẽ cho phép bạn báo cáo lỗi, các warning messages cũng như info messages mà sau này có thể được lấy ra và phân tích.
* Đối tượng thực hiện việc ghi log trong ứng dụng thì được gọi là Logger

## **Logger**

* Lớp java.util.Logger là một lớp chính để truy cập vào Java logging API. Cách tạo:

Logger logger = Logger.getLogger("myLogger");

* Chuỗi truyền vào phương thức getLogger() sẽ là tên của Logger được tạo. Bạn có thể đặt tên thoải mái nhưng nó sẽ quyết định Logger nằm ở đâu trong Hệ thống Logger (Logger Hierarchy). Mỗi dấu chấm (.) trong tên bạn đặt sẽ là một nhánh trong Hệ thống Logger. Hãy nhìn ví dụ dưới đây:

myApp

myApp.user

myApp.admin

* Tất cả các tên trên đều sử dụng được. Chúng tạo thành một hệ thống. Tên “myApp” nằm ở đỉnh của hệ thống. Hai tên “myApp.user” và “myApp.admin” là con của tên “myApp”
* Nếu bạn muốn biết tên của Logger bạn có thể gọi phương thức getName():

String name = logger.getName();

* Người ta thường dùng tên lớp của lớp tạo ra Logger, bao gồm cả package name, để đặt tên cho Logger. Ví dụ:

Logger logger = Logger.getLogger( MyClass.class.getName());

### **Các phương thức log():**

* Nhóm phương thức log() sẽ ghi lại message tại một mức log (log Level) cố định. Mức log sẽ được truyền vào dưới dạng tham số. Các mức log này có các hằng số cố định.
* Một số phương thức log() có thể dùng đối tượng làm tham số. Những đối tượng này sẽ được đưa vào log message trước khi được ghi xuống. Việc đưa đối tượng này vào log message chỉ diễn ra nếu như message không được lọc. Một số ví dụ:

Logger logger = Logger.getLogger("myLogger");

logger.log(Level.SEVERE, "Hello logging");

* Đây là những gì được ghi vào console (nơi ghi log mặc định):

08-01-2012 14:10:43 logging.LoggingExamples main

SEVERE: Hello logging

* Còn đây là ví dụ khi truyền một tham số vào message: [1]

logger.log(Level.SEVERE, "Hello logging: {0}, {1}",

new Object[] {"P1", "P2"});

* Đây là những gì được ghi xuống:

08-01-2012 14:45:12 logging.LoggingExamples main

SEVERE: Hello logging: P1, P2

* Còn đây là một ví dụ về một đối tượng Throwable:

logger.log(Level.SEVERE, "Hello logging",

new RuntimeException("Error"));

Và được ghi xuống như sau:

08-01-2012 14:54:29 logging.LoggingExamples main

SEVERE: Hello logging

java.lang.RuntimeException: Error

at logging.LoggingExamples.main(LoggingExamples.java:18)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39) at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

at java.lang.reflect.Method.invoke(Method.java:597)

at com.intellij.rt.execution.application.AppMain.main(AppMain.java:120)

### **Các phương thức logp():**

* Các phương thức logp() hoạt động giống với các phương thức log() nhưng mỗi phương thức có thêm hai tham số: sourceClass và sourceMethod.
* Hai tham số này sẽ cho biết lớp nào và phương thức nào mà log message đã được tạo ra.

### **Các phương thức logrb()**

* Các phương thức logrb() cũng giống các phương thức log(), nhưng ngoài ra chúng có thể lấy log message từ một resource bundle. Một resource bundle là một tập hợp chứa các cặp khóa và giá trị. Các khóa và giá trị này không đổi bất chấp ngôn ngữ nào.

Ví dụ về logrb():

logger.logrb(Level.SEVERE, "logging.LoggingExamples", "main",

"resources.myresources", "key1");

Trong ví dụ này chúng ta tìm trong resource bundle có tên là resources.myresources với khóa là “key 1”. Nếu trong resource bundle không có chứa khóa nào tên là “key 1” thì giá trị ghi xuống log sẽ là “key 1”, còn nếu tìm thấy nó sẽ lấy giá trị tương ứng với khóa “key 1” trong resource bundle và ghi xuống. Đây là những gì trong resource bundle:

key1 : This is message 1

key2 : this is message 2

Và kết quả ghi xuống console là:

08-01-2012 17:14:39 logging.LoggingExamples main

SEVERE: This is message 1

### **Những phương thức log khác**

Logger còn có các phương thức log sau đây:

entering(String sourceClass, String sourceMethod);

entering(String sourceClass, String sourceMethod, Object param1);

entering(String sourceClass, String sourceMethod, Object[] params);

exiting (String sourceClass, String sourceMethod);

exiting (String sourceClass, String sourceMethod, Object result);

fine (String message);

finer (String message);

finest (String message);

config (String message);

info (String message);

warning (String message);

severe (String message);

Mỗi phương thức tương ứng với một mức log khác nhau cùng tên với nó. Sử dụng những phương thức nà để ghi log tương ứng với sử dụng phương thức log().

## **Logger Hierarchy**

* Phần này chúng ta sẽ nói kĩ hơn về việc tổ chức hệ thống Logger/Cây Logger

Xem xét ví dụ sau:

Logger logger = Logger.getLogger("com.jenkov.web");

Trong ví dụ này tên của Logger được tạo ra là com.jenkov.web

* Tên của Logger sẽ chỉ ra một hệ thống Loggers. Mỗi dấu chấm (.) sẽ đánh dấu một mức trong hệ thống. Với tên com.jenkov.web thì Logger có 3 Logger cha là:

""

"com"

"com.jenkov"

* Nếu bạn gọi getParent() của Logger vừa được tạo trên, kết quả sẽ trả về com.jenkov. Và bạn gọi tiếp getParent() ở Logger đó sẽ nhận được kết quả là com. Đỉnh của hệ thống sẽ là Logger với tên rỗng (“”).

***Chú ý:***

* Bạn phải tạo hệ thống cây logger theo thứ tự từ trên xuống. Nếu bạn tạo như sau:

Logger logger = Logger.getLogger("com.jenkov.web");

khi gọi phương thức getParent() bạn sẽ nhận được Logger với tên rỗng (“”). Lý do là tại những Logger trung gian chưa được tạo ra. Bạn phải tạo hệ thống cây Logger như sau:

Logger logger = Logger.getLogger("");

Logger logger1 = Logger.getLogger("com");

Logger logger2 = Logger.getLogger("com.jenkov");

Logger logger3 = Logger.getLogger("com.jenkov.web");

## **Log levels của các Logger trong Hệ thống**

* Nếu một Logger được đặt một mức log xác định thì những messages có mức log ít quan trọng hơn mức log đó sẽ được bỏ qua. Thêm vào đó, tất cả những mức dưới mức được đặt sẽ không được truyền lên trong hệ thống Logger. Đây là điều khác với lọc (filter) Logger.
* Dưới đây là một ví dụ về một hệ thống Logger với 3 Loggers và Logger ở giữa (tên 1) được đặt mức log là WARNING.

Logger logger = Logger.getLogger("");

Logger logger1 = Logger.getLogger("1");

Logger logger1\_2 = Logger.getLogger("1.2");

**logger1 .setLevel(Level.WARNING);**

logger .info("msg:");

logger1 .info("msg: 1");

logger1\_2 .info("msg: 1.2");

* Kết quả của ví dụ trên là không có message nào được log trừ root Logger. Mức INFO ít quan trọng hơn mức WARNING nên INFO message được log bằng Logger ở giữa bị bỏ qua, và không được truyền lên Logger đỉnh.
* Mặc khác, INFO message log bằng Logger dưới cùng (tên 1.2) cũng bị bỏ qua và không được truyền lên. Lý do là Logger này không được đặt mức log nên nó sẽ kế thừa mức log từ Logger cha trong hệ thống. Nói cách khác thì Logger dưới cùng có mức log giống với mức log của Logger ở giữa.
* Đây là kết quả được log ra từ đoạn code trên:

14-01-2012 13:25:32 java.util.logging.LogManager$RootLogger log

INFO: msg:

* Để có thể log tất cả message mức INFO ra Logger cuối cùng (tên 1.2), ngay cả khi Logger cha của nó là Logger ở giữa có mức WARNING, chúng ta thêm đoạn code sau (được in đậm)

Logger logger = Logger.getLogger("");

Logger logger1 = Logger.getLogger("1");

Logger logger1\_2 = Logger.getLogger("1.2");

logger1 .setLevel(Level.WARNING);

**logger1\_2.setLevel(Level.INFO);**

logger .info("msg:");

logger1 .info("msg: 1");

logger1\_2 .info("msg: 1.2");

* Kết quả của đoạn code trên là các INFO message log từ Logger dưới cùng (tên 1.2) bây giờ đã được log, nhưng vẫn không được truyền lên trong hệ thống Logger. Thật ra là có nhưng Logger ở giữa đã lọc nó ra vì nó có mức log là WARNING. Vì thế message sẽ không được Logger ở giữa log xuống cũng như không được truyền lên hệ thống Logger.
* Đây là kết quả được log xuống:

14-01-2012 13:30:27 java.util.logging.LogManager$RootLogger log

INFO: msg:

14-01-2012 13:30:27 logging.LoggingExamples main

INFO: msg: 1.2

* Những mức log có sẵn, nằm trong java.util.logging.Level
* SEVERE
* WARNING
* INFO
* CONFIG
* FINE
* FINER
* FINEST

## **FORMATTER**

* Handler trong Java sử dụng Formatter để định dạng LogRecord trước khi ghi các LogRecord ra bên ngoài.
* Java hỗ trợ 2 loại Formatter: SimpleFormatter và XMLFormater.
* Mỗi loại Handler khác nhau trong Java sẽ sử dụng hoặc là SimpleFormatter hoặc XMLFormatter làm mặc định. Tuy nhiên ta có thể tạo ra Formatter của riêng mình :

public class MyFormatter extends Formatter {

@Override

public String format(LogRecord record) {

return record.getLevel() + ":" + record.getMessage();

}

}

Ví dụ tạo formatter cho HTML :

**class** MyHtmlFormatter **extends** Formatter {

*// this method is called for every log records*

**public** String format(LogRecord rec) {

StringBuffer buf = **new** StringBuffer(1000);

buf.append("<tr>\n");

*// colorize any levels >= WARNING in red*

**if** (rec.getLevel().intValue() >= Level.WARNING.intValue()) {

buf.append("\t<td style=\"color:red\">");

buf.append("<b>");

buf.append(rec.getLevel());

buf.append("</b>");

} **else** {

buf.append("\t<td>");

buf.append(rec.getLevel());

}

buf.append("</td>\n");

buf.append("\t<td>");

buf.append(calcDate(rec.getMillis()));

buf.append("</td>\n");

buf.append("\t<td>");

buf.append(formatMessage(rec));

buf.append("</td>\n");

buf.append("</tr>\n");

**return** buf.toString();

}

**private** String calcDate(**long** millisecs) {

SimpleDateFormat date\_format = **new** SimpleDateFormat("MMM dd,yyyy HH:mm");

Date resultdate = **new** Date(millisecs);

**return** date\_format.format(resultdate);

}

*// this method is called just after the handler using this*

*// formatter is created*

**public** String getHead(Handler h) {

**return** "<!DOCTYPE html>\n<head>\n<style"

+ "type=\"text/css\">\n"

+ "table { width: 100% }\n"

+ "th { font:bold 10pt Tahoma; }\n"

+ "td { font:normal 10pt Tahoma; }\n"

+ "h1 {font:normal 11pt Tahoma;}\n"

+ "</style>\n"

+ "</head>\n"

+ "<body>\n"

+ "<h1>" + (**new** Date()) + "</h1>\n"

+ "<table border=\"0\" cellpadding=\"5\" cellspacing=\"3\">\n"

+ "<tr align=\"left\">\n"

+ "\t<th style=\"width:10%\">Loglevel</th>\n"

+ "\t<th style=\"width:15%\">Time</th>\n"

+ "\t<th style=\"width:75%\">Log Message</th>\n"

+ "</tr>\n";

}

*// this method is called just after the handler using this*

*// formatter is closed*

**public** String getTail(Handler h) {

**return** "</table>\n</body>\n</html>";

}

}

## **HANDLER**

* Một Handler trong Java là một thành phần cho phép đưa các message ra thế giới bên ngoài.
* Ta có thể thêm một hoặc nhiều Handler cho một Logger.
* Cách thêm một Handler vào Logger:

logger.addHandler(new ConsoleHandler());

### **HANDLER VÀ FORMATER**

* Một Handler thường sử dụng một Formater để định dạng messages trước khi ghi log.
* Chúng ta có thể tạo ra Formater riêng tự tạo ra hoặc sử dụng một trong hai loại Formater (Simple Formater hay XMLFormater).
* Cách thêm một Formater vào Handler:

ConsoleHandler handler = new ConsoleHandler();

handler.setFormatter(new SimpleFormatter());

Formatter formatter = handler.getFormatter();

### **PHÂN LOẠI HANDLER**

* Java 4 hỗ trợ 5 loại Handler chính:
* ConsoleHandler.
* FileHandler.
* StreamHandler.
* SocketHandler.
* MemoryHandler.

Trong đó ta thường sử dụng nhất là FileHandler.

### ***CONSOLEHANDLER:***

* Console Handler ghi các messages ra System.err.
* Mặc định Console Handler sử dụng SimpleFormatter để định dạng các messages trước khi ghi ra System.err.
* Cách tạo một Console Handler:

ConsoleHandler handler = new ConsoleHandler();

### ***FILEHANDLER:***

* FileHandler dùng để ghi tất cả các messages ra file, đó có thể là một file hoặc là một tập hợp các file luân phiên nhau. Nếu sử dụng tập hợp các file, mỗi file được chỉ định một kích thước nhất định, sau đó tạo ra tập tin lưu trữ message; mỗi tên file bao gồm một tên file và số thứ tự, ví dụ mylog.0.txt, mylog.1.txt, …
* Mặc định FileHandler mặc định sử dụng XMLFormatter đề định dạng các message trước khi ghi chúng ra file.
* Sau đây là các cách khác nhau để tạo ra FileHandler:

FileHandler handler = new FileHandler();

FileHandler handler = new FileHandler(String pattern);

FileHandler handler = new FileHandler(String pattern, boolean append);

FileHandler handler = new FileHandler(String pattern, int limit, int count);

FileHandler handler = new FileHandler(String pattern, int limit, int count,

boolean append);

Ví dụ :

FileHandler handler = new FileHandler("myapp-log.%u.%g.txt");

FileHandler handler = new FileHandler("myapp-log.%u.%g.txt", true);

FileHandler handler = new FileHandler("myapp-log.%u.%g.txt", 1024 \* 1024, 10);

FileHandler handler = new FileHandler("myapp-log.%u.%g.txt",1024 \* 1024,10, true);

Chú ý: File Name Pattern.

File Name Pattern là một chuỗi có chứa tên tập tin cộng với một hoặc nhiều ký tự đặc biệt, báo cho FileHandler biết làm cách nào để tạo ra các tên file. Các mã đặc biệt có thể sử dụng là:

|  |  |
| --- | --- |
| Mã | Ý nghĩa |
| / | Phân cách tệp tin của hệ thống, thông thường là / hoặc \ |
| %t | Thư mục temp của hệ thống. |
| %h | Thư mục user home của hệ thống. |
| %g | Tạo ra các số phân biệt giữa các tập tin. |
| %u | Một số duy nhất để tránh tạo ra các tập tin cùng tên. |
| %% | Dấu %, trong trường hợp muốn sử dụng trong tên tập tin của mình. |

Ví dụ

|  |  |
| --- | --- |
| File Name Pattern | Ý nghĩa |
| Logfile.txt | Các file có tên là Logfile và nằm tại thư mục hiện tại của ứng dụng. |
| Logfile%g.txt | Các file có tên là Logfile, nằm trong thư mục hiện tại của ứng dụng. Một số tự tạo ra và được thêm vào đằng sau tên của tệp tin, ví dụ Logfile0.txt, Logfile1.txt … |
| Logfile%u%g.txt | Các file có tên là Logfile, nằm trong thư mục hiện tại của ứng dụng. Một số tự tạo ra và được thêm vào đằng sau tên của tệp tin, tạo ra một số duy nhất và thêm vào sau dấu “.”. Ví dụ: Logfile0.0.txt, Logfile0.1.txt … |

### ***STREAMHANDLER:***

* StreamHandler ghi các message vào OutputStream.
* Cách tạo StreamHandler:

StreamHandler handler = new StreamHandler();

StreamHandler handler = new StreamHandler(outputStream, formatter);

Constructer thứ nhất tạo ra một StreamHandler mà không có outputStream và sử dụng mặc định SimpleFormatter. Nếu dung Constructer này khi sử dụng ta phải setOutputStream() cho handler.

Constructer thứ hai tạo ra một StreamHandler với outpurStream và formatter. Forrmatter ở đây có thể 1 trong 2 loại Formatter hoặc tự tạo ra Formatter cho riêng mình, outputStream ở đây có thể là FileOutputStream…

### ***SOCKETHANDLER:***

* SocketHandler ghi các message vào một vài địa chỉ kết nối mạng thông qua socket.
* Cách tạo SocketHandler:

SocketHandler socketHandler = new SocketHandler(host, port);

Trong đó host là tên domain name, port là TCP port kết nối tới.

Ví dụ :

SocketHandler socketHandler = new SocketHandler("hcmus.edu.vn", 80);

### ***MEMORYHANDLER:***

* MemoryHandler giữ LogRecords bên trong các bộ đệm (Buffer). Khi bộ đệm đầy, LogRecords sẽ ghi đè lên các dữ liệu cũ trong bộ đệm.
* Khi một sự kiện được kích hoạt, LogRecords bên trong bộ đệm sẽ được đẩy đến một Handler cụ thể, Handler này sẽ ghi LogRecords.
* Hai cách khởi tạo một MemoryHandler:

MemoryHandler handler = MemoryHandler();

MemoryHandler handler = MemoryHandler( targetHandler, bufferSize, pushLevel);

## **FILTERS**

* Ta có thể set Filters cho Logger. Một Filter dùng để lọc các log message, nghĩa là quyết định xem message có được ghi hay không.
* Cách tạo một Filters:

Filter filter = new MyFilter();

logger1.setFilter(filter);

public class MyFilter implements Filter {

public boolean isLoggable(LogRecord record) {

return false;

}

}

Filters trên từ chối tất cả các message. Đây chỉ là một ví dụ về cách implements Filter trong Java. Ta có thể kiểm tra LogRecord và đưa ra cài đặt chi tiết hơn cho hàm isLoggable.

## **LogRecord**

* Các lớp LogRecord bọc lấy các thông điệp ghi vào một Logger . Có thể ghi một LogRecord trực tiếp vào một Logger , hoặc để Logger bọc những gì được ghi lại trong một LogRecord.
* Khi một LogRecord được truyền đến, hoặc được tạo ra trong nội bộ của Logger , nó có thể được truyền đến một Filter (nếu một Filter được thiết lập).
* Các LogRecord chứa các phương thức getter sau:

getLevel ()

getLoggerName ()

getMessage ()

getMillis ()

getParameters ()

getResourceBundle ()

getResourceBundleName ()

getSequenceNumber ()

getSourceClassName ()

getSourceMethodName ()

getThreadID ()

getThrown ()

* Mỗi phương thức getter có một phương thức setter tương ứng quá, nhưng trừ khi tự tạo LogRecord cho riêng mình, ta sẽ không cần các phương thức setter. Dưới đây là giải thích ngắn gọn về các phương thức getter:
* getLevel(): trả về Level của các thông điệp.
* getLoggerName(): trả về tên của Logger.
* getMessage(): trả về thông điệp.
* getMillis() : trả về thời gian mili-giây mà thông điệp được ghi lại.
* getParameters() : trả về danh sách cách tham số được chèn vào LogRecord.
* getResourceBundle() : trả về ResourceBundle (nếu có) được sử dụng để khoanh vùng thông điệp.
* getResourceBundleName(): trả về  tên của ResourceBundle (nếu có).
* getSequenceNumber(): trả về một số chuỗi tạo ra trong LogRecord constructor khi LogRecord được tạo ra.
* getSourceClassName(): trả về tên lớp của class mà LogRecord đại diện.
* getSourceMethodName() trả về tên lớp của phương thức mà LogRecord đại diện.
* getThreadID(): trả về ID của thread ghi thông điệp .
* getThrown() phương thức trả về Throwable rằng đã đánh dấu ném khi ghi các thông điệp được đại diện bởi LogRecord .

## **Configuration**

* Java Logging API có thể được cấu hình theo hai cách:
* Qua một lớp cấu hình.
* Thông qua một tập tin cấu hình.
* Việc khởi tạo cấu hình được lớp java.util.logging.LogManager đảm nhiệm.

### Cấu hình lớp

* Ta có thể sử dụng một lớp Java để cấu hình Logging API Java. Bạn làm như vậy bằng cách xác định tên của lớp trong tham số JVM java.util.logging.config.class .

### Configuration File

* Nếu không có lớp cấu hình được chỉ định, thay vào đó ta có thể chỉ định một tập tin cấu hình (nhưng không có lớp cấu hình có thể được xác định sau đó!).
* Các Logging API Java có một tập tin cấu hình đăng nhập mặc định nằm ở "lib/logging.properties" , bên trong thư mục JRE. Nếu chỉnh sửa tập tin này, ta sẽ chỉnh sửa các thiết lập mặc định cho toàn bộ JRE.
* Ta có thể thiết lập một tập tin cấu hình riêng biệt cho ứng dụng của chúng ta. Bằng cách thiết lập thuộc tính JVM java.util.logging.config.file để trỏ đến tập tin này.
* Ví dụ về file cấu hình:

handlers = java.util.logging.FileHandler

config =

"logger".handlers =

"logger".useParentHandlers =

"logger".level =

java.util.logging.FileHandler.level = WARNING

java.util.logging.FileHandler.filter =

java.util.logging.FileHandler.formatter =

java.util.logging.FileHandler.encoding =

java.util.logging.FileHandler.limit =

java.util.logging.FileHandler.count =

java.util.logging.FileHandler.append = false

java.util.logging.FileHandler.pattern = log.%u.%g.txt

java.util.logging.ConsoleHandler.level = WARNING

java.util.logging.ConsoleHandler.filter =

java.util.logging.ConsoleHandler.formatter =

java.util.logging.ConsoleHandler.encoding =

java.util.logging.StreamHandler.level = WARNING

java.util.logging.StreamHandler.filter =

java.util.logging.StreamHandler.formatter =

java.util.logging.StreamHandler.encoding =

java.util.logging.SocketHandler.level = WARNING

java.util.logging.SocketHandler.filter =

java.util.logging.SocketHandler.formatter =

java.util.logging.SocketHandler.encoding =

java.util.logging.SocketHandler.host =

java.util.logging.SocketHandler.port =

java.util.logging.MemoryHandler.level = WARNING

java.util.logging.MemoryHandler.filter =

java.util.logging.MemoryHandler.size =

java.util.logging.MemoryHandler.push =

java.util.logging.MemoryHandler.target =

## **LogManager**

* Lớp LogManager quản lý bên trong hệ thống phân cấp Logger, và khởi tạo các thiết lập của Logger, thông qua configuration class hoặc configuration file.
* Chỉ có duy nhất một đối tượng LogManager được tạo ra trong JVM. Nói cách khác nó là một singleton. Đây là cách lấy đối tượng LogManager:

LogManager manager = LogManager.getLogManager();

* Chúng ta sẽ không cần phải tương tác nhiều với LogManager, bởi vì các cấu hình đều được thiết lập ở constructor.
* Nếu muốn nạp lại configuration file, ta có thể sử dụng các phương thức sau:

readConfiguration();

readConfiguration(inputStream);

* Phương thức đầu tiên trên LogManager chỉ đơn giản là đọc lại cấu hình từ file (hoặc lớp), trong trường hợp này đã thay đổi.
* Phương thức thứ hai đọc các cấu hình từ InputStream.
* Ta cũng có thể truy cập một MXBean (Java Management Extensions) từ LogManager bằng cách sử dụng phương thức getLoggingMXBean(). Ví dụ:

LoggingMXBean mxBean = logManager.getLoggingMXBean();

* LogManager còn rất nhiều phương thức khác nhưng hầu như chúng ta sẽ không sử dụng chúng.

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | C. Hope, "ComputerHope.com," [Online]. Available: http://www.computerhope.com/jargon/l/log.htm. [Accessed 21 03 2015]. |
| **[2]** | **Jenkov, "Java-logging," [Online]. Available: http://tutorials.jenkov.com/java-logging/index.html. [Accessed 21 03 2015].** |
| [3] | K. Bonsor, "How stuff works," HowStuffWorks, [Online]. Available: http://computer.howstuffworks.com/workplace-surveillance4.htm. [Accessed 21 03 2015]. |
| **[4]** | **Vogella, "Vogella," Vogella, [Online]. Available: http://www.vogella.com/tutorials/Logging/article.html. [Accessed 21 03 2015].** |