



**작성**

대덕 인재 개발원. 말자

**목차**

## 1. Log4J 란?

어플리케이션이 실행되고 있을 때 잘 동작하고 있는지 아니면 에러나 경고가 있었는지에 대한 정보를 알아야 한다. 그래서 대부분의 어플리케이션(웹서버, DBMS, 상용프로그램등)에서는 로그를 남긴다. 로그를 분석하여 해당 어플리케이션의 상태정보나 잘못된 원인을 파악하는데 유용한 정보를 얻을수 있으며 또한 어플리케이션의 개발하는 경우라도 로그는 유용한 정보를 제공해준다. 단점이라면 로깅작업을 해야 하므로 소스의 가독성이 떨어지며, 실행하기 위해서 약간의 비용이 들어간다는 것이다. 하지만 System.out.println() 하는 것보다는 훨씬 이득일 것이다.

이러한 로그를 제공하는 것중 단연 Apache에서 제공하는 log4j(Log For Java)프로젝트이며 자바진영에서 거의 표준으로 자리를 잡을 정도로 막강한 기능과 성능을 제공해 준다. 또한 JDK1.4 부터 log4j를 모델로 하여 Logger 관련 API가 추가되었다.

## 2. Log4J 설치 및 간단한 예제

Apache사이트에서 log4j를 다운로드 받도록 한다.

- **log4j** : http://logging.apache.org/ 에서 1.2 버전을 다운로드

apache-log4j-1.2.15.zip 압축파일을 풀어서 log4j-1.2.15.jar 파일을 클래스 패스로 인식할 수 있도록 한다. 여기서는 오랜만에 컴파일도 해볼 겸 에디터에서 작성하고 콘솔에서 컴파일 및 실행을 해보도록 한다. (log4j-1.2.15.jar 파일도 같은 폴더에)

```
// 예제는 C:\log4j\LogTest.java 로 저장하였음
 1: import org.apache.log4j.BasicConfigurator;
 2: import org.apache.log4j.Logger;
 3:
 4: public class LogTest {
 5:    static Logger logger = Logger.getLogger( LogTest.class ) ;
 6:    public static void main(String[] args) {
 7:          BasicConfigurator.configure();
 8:          logger.debug("DEBUG입니다.");
 9:          logger.info("INFO입니다.");
10:          logger.warn("WARN입니다.");
11:          logger.error("ERROR입니다.");
12:          logger.fatal("FATAL입니다.");
13:    } // main
14:} // class
```


파일을 작성하고나서 컴파일 및 실행, 결과를 확인한다.

결과메시지를 간단히 분석해보면

- 첫번째 숫자 0은 Logger 가 동작한다음 걸린시간(밀리세컨드)
- 두번째 [main]은 호출한 스레드
- 세번째 DEBUG, INFO, ERROR 등은 로그 레벨(Level)
- 네번째 LogTest 는 해당 클래스
- 다섯번째 NDC는 출력 안 되었음
- 마지막은 사용자가 입력한 메시지

너무 단순하게 출력되어 그리 유용해 보이지는 않지만 System.out.println() 보다는 훨씬 낫다

## 3. Log4J 주요 컴포넌트

Log4J는 3가지 중요한 컴포넌트가 있는데 다음과 같다.

- **Logger :** Log4j 패키지의 핵심클래스이다. 로그 메세지를 Appender에 전달하는 역할을 하며 debug(), info(), error()등의 로그 메서드를 사용한다. (이전에는 Category클래스를 사용하였는데 현재는 Category의 서브클래스인 Logger를 사용한다. Category 클래스와 관련된 메서드는 deprecated 되었으므로 Logger 클래스의 메소드 사용하는 것이 바람직하다.)

- **Appender :** Logger로부터 전달된 메시지를 기록하는 클래스이다. 콘솔에 출력할 것이지, 파일에 기록할 것인지, email로 보낼것인지등 다양한 방식을 지원한다. Appender중 자주 사용되는 것만 정리해 보면 다음과 같다.
  - ConsoleAppender : 콘솔에 로그 출력
  - FileAppender : 파일에 로그 출력 (보통은 사용안함)
  - RollingFileAppender : 파일 사이즈를 기준으로 로그 출력 ( FileAppender 확장)
  - DailyRollingFileAppender : DatePattern(날짜, 시간)을 기준으로 로그 출력 ( FileAppender 확장)
  - JDBCAppender : DataBase로 로그 출력
  - JMSAppender : JMS(Java Message Service)로 topic 전송
  - NTEventLogAppender : NT 이벤트로그에 로그전송(NT기반 윈도우)
  - SMTPAppender : 메일로 로그 전송
  - SocketAppender : 원격지 서버에 로그 전송 (TCP/IP protocol 기반)

- **Layout :** 로그를 기록할 때 어떤 형식으로 출력할 것이지에 대한 레이아웃에 대한 결정을 하는 클래스이다.
  - DateLayout : 날짜관련 포멧을 사용한다.
  - HTMLLayout : HTML 관련 포멧을 사용한다.
  - PatternLayout : 직접 포멧관련 패턴을 사용한다.
  - SimpleLayout : "DEBUG - Hello world" 같이 간단한 레이아웃
  - XMLLayout : log4j.dtd 를 기반으로 된 xml 레이아웃 사용
  - TTCCLayout : "%r [%t] %p %c %x - %m%n" 형식을 기본으로 하는 레이아웃

## 4. 로깅 Level 및 로깅 메서드

Log4J는 setLevel()메서드로 로그레벨을 설정할 수 있으며 설정된 로그레벨 이하의 로그는 무시된다. (1.2 버전부터 Priority 클래스를 대체한 Level로 구현되었다. (Priority 관련 메서드는 Deprecated)

| Level | Priority int | Use |
|---|---|---|
| ALL | -2147483648 | 가장 낮은 레벨, 모든 레벨의 로그작업이 동작한다. |
| TRACE | 5000 | 버전 1.2.12 에서 추가됨 |

| DEBUG | 10000 | 개발할 때 디버그용으로 사용, 상세출력용 |
|---|---|---|
| INFO | 20000 | 어플리케이션이 실행중 메시지를 남기고자 할 때 |
| WARN | 30000 | 에러는 아니지만 주의나 경고가 필요한 경우 |
| ERROR | 40000 | 일반적인 에러상황인 경우 |
| FATAL | 50000 | 심각한 에러인 경우 |
| OFF | 2147483647 | 가장 높은 레벨, 로그관련 작업을 사용하지 않으려고 할 때 |
| eg : logger.setLevel(org.apache.log4j.Level.DEBUG); | | |

표 1. Logger 의 Level

로거에 메시지를 전달하는 출력 메서드는 다음과 같으며 각각 두가지 형식으로 되어잇다. 이미 사용해봤으므로 간단하게 표만 정리하겠다.

| Method | |
|---|---|
| trace (Object message) | trace (Object message, Throwable t) |
| debug (Object message) | debug (Object message, Throwable t) |
| info (Object message) | info (Object message, Throwable t) |
| warn (Object message) | warn (Object message, Throwable t) |
| error (Object message) | error (Object message, Throwable t) |
| fatal (Object message) | fatal (Object message, Throwable t) |
| log (Level, Object message) | log (Level, Object message, Throwable t) |
| ps : trace()는 1.2.12 버전에서, log()는 불편해서 사용을 잘 안한다. | |

표 2. Logger의 로그출력 Method

- **Log4J 두번째 예제**

이번 예제는 이클립스나 넷빈 IDE에서 작성하도록 한다. ConsoleAppender와 DailyRollingFileAppender 를 사용하여 로그를 두곳에 남기도록 하는데 목적이 있으며 파일로 남기는 경우는 INFO 레벨이상으로 한다.

log4j-1.2.15.jar 파일도 프로젝트 성격에 맞게 추가해 주어야 한다.

```
 1: import java.io.IOException;
 2: import org.apache.log4j.ConsoleAppender;
 3: import org.apache.log4j.DailyRollingFileAppender;
 4: import org.apache.log4j.Level;
 5: import org.apache.log4j.Logger;
 6: import org.apache.log4j.PatternLayout;
 7:
 8: public class LogTest2 {
 9:    static Logger logger = Logger.getLogger(LogTest2.class);
10:    public static void main(String[] args) {
11:      logger.setLevel(Level.ALL);
12:      String pattern = "[%-5p %d{yyyy-MM-dd hh:mm:ss} %c{1}.%M:%L] %m%n";
13:
14:      // 2개의 appender 선언(콘솔,파일)) 및 공토으로 사용할 layout 선언
15:      PatternLayout layout = new PatternLayout(pattern);
16:      ConsoleAppender consoleAppender = null;
```

```
17:     DailyRollingFileAppender fileAppender = null;
18:
19:     String fileName = "c:/MyTest.log";
20:     String datePattern = "'.'yyyy-MM-dd";
21:
22:     consoleAppender = new ConsoleAppender(layout);
23:     try {
24:       fileAppender = new DailyRollingFileAppender(layout,fileName,datePattern);
25:     } catch (IOException e) {
26:       e.printStackTrace();
27:     }
28:
29:     // 파일작업용 appender는 레벨을 INFO로 한다.
30:     fileAppender.setThreshold(Level.INFO);
31:
32:     // logger에 각 appender를 추가한다.
33:     logger.addAppender(fileAppender);
34:     logger.addAppender(consoleAppender);
35:
36:     logger.trace("trace 테스트입니다.");
37:     logger.debug("debug 테스트입니다.");
38:     logger.info("info 테스트입니다.");
39:     logger.warn("warn 테스트입니다.");
40:     logger.error("error 테스트입니다.");
41:     logger.fatal("fatal 테스트입니다.");
42:     logger.log(Level.DEBUG, "debug level 테스트입니다");
43:     logger.log(Level.FATAL, "fatal level 테스트입니다");
44:
45:     // 에러가 발생한 경우
46:     int i = 3, j = 1, k;
47:     try {
48:       k = i / (j - 1);
49:     } catch (Exception e) {
50:       // error 또는 fatal 메서드를 사용하며 해당 예외객체까지도 전달해준다.
51:       logger.error("error 발생", e);
52:     }
53:     logger.debug("Portfolio는 언제??");
54:   } // main
55: }// class
```

## 5. 설정파일 log4j.properties

두번째 예제를 하면서 발생한 문제점은? 로그를 남기기 위해서 해당 클래스에서 너무 많은 작업이 이루어진다는 것이다.

그리고 자바 클래스를 만들때마다 동일한 작업을 한다는 것이다. 이러한 작업을 설정파일을 통해서 작업을 하면 편리하다. 기본적으로 log4j에서 설정파일은 log4j.properties 또는 XML형식인 log4j.xml 을 기본으로 한다.

```
# 루트 Logger를 설정한다. Level을 설정하고 그 다음은 appender 를 구분하기 위한 이름이다.
# appender이름은 각각을 구분하기 위한 것일뿐 해당 클래스와는 상관없다.
log4j.rootLogger=ALL, console, file

# console에 대한 설정 appender
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=[%-5p %d{ISO8601} %c{1}.%M:%L] %m%n

# file 이름의 appender 설정
log4j.appender.file=org.apache.log4j.DailyRollingFileAppender
log4j.appender.file.Threshold=INFO
log4j.appender.file.File=C:/MyTest.log
log4j.appender.file.DatePattern='.'yyyy-MM-dd
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=[%-5p %d{ISO8601} %l] %m%n
```

설정파일을 log4j.properties 로 저장하고 좀전에 작성했던 LogTest2 자바파일에서 라이번호 11 부터 35까지를 지운다. 실행을 하면 동일한 결과가 나오는 것을 알수 있다. 설정파일은 여러 방법으로 설정할수 있으므로 API문서나 인터넷을 통해 찾아보기 바람

## 6. Appender

Logger에 Appender를 추가할려면 addAppender(appender) 메서드를 사용하거나 설정파일에 지정해주면 된다.

- public void addAppender(Appender appender);
- log4j.logger.loggerName.appender=appenderName

- **WriterAppender**

  WriterAppender를 직접사용하지 않고 해당 WriteAppender를 상속한 ConsoleAppender, FileAppender를 사용하게 된다.

  - Threshold : ALL 부터 OFF 까지의 Level, appender는 지정된 Level이하는 처리하지 않는다. Logger의 Level이 INFO 이고 appender의 Threshold가 WARN이라면 WARN보다 낮은 로그는 하지 않는다.
    지정하지 않을 경우 Logger의 Level을 따른다.
  - Encoding : 로그를 남길 때 encoding을 지정한다. 기본적으로 시스템의 정보를 따른다.
  - ImmediateFlush : 기본값 true. 버퍼를 사용하지 않고 바로 출력, 대부분의 상황에 적당하다.
  - Target : java.io.Writer 또는 java.io.OutputStream

- **ConsoleAppender**

  WriterAppender ← ConsoleAppender

  콘솔에 로그를 남기는 대표적인 객체

  - Threshold, Encoding, ImmediateFlush → WriterAppender

- Target : 기본값은 System.out ( System.err 또는 System.out )

- **FileAppender**

  WriterAppender ← FileAppender

  파일에 로그를 남기는 객체, 이것 보다 확장된 RollingFileAppender, DailyRollingFileAppender를 사용한다.

  - Threshold, Encoding, ImmediateFlush → WriterAppender
  - File : 기록할 파일명.
    윈도우에서 파일명에 경로가 있는 경우 "C:\\log\\test.log" 또는 "C:/log/test.log" 로 지정한다.
  - Append : 기본값 true. 파일 끝에 추가하는 것을 의미한다. false 는 각각의 프로그램이 시작할때 파일에 덮어씌운다.
  - BufferSize : 기본값 8142. 버퍼사이즈
  - BufferedIO : 기본값 false. 만약 true로 설정할 경우 자동으로 ImmediateFlush 가 false가 된다.

- **RollingFileAppender**

  WriterAppender ← FileAppender ← RollingFileAppender

  지정된 사이즈만큼 로그파일에 기록한다. 사이즈가 넘으면 백업을 하며 백업 파일의 갯수도 지정할 수 있다.

  - Threshold, Encoding, ImmediateFlush → WriterAppender
  - File, Append, BufferSize, BufferedIO → FileAppender
  - MaxFileSize : 기본값 10MB. 끝에 KB, MB 또는 GB를 붙인다. 지정한 크기에 도달하면 로그파일을 교체한다(rollover).
  - MaxBackupIndex : 기본값 1. 백업 파일의 유지갯수. 오래된 파일들은 삭제한다. 0 은 백업파일을 만들지 않는다.

- **DailyRollingFileAppender**

  WriterAppender ← FileAppender ← RollingFileAppender

  주기적으로 로그파일을 교체한다. 교체는 DatePattern 에 따르게된다.

  - Threshold, Encoding, ImmediateFlush → WriterAppender
  - File, Append, BufferSize, BufferedIO → FileAppender
  - DatePattern : SimpleDateFormat 형식을 따르며 지정된 형식은 교체주기를 설정할 뿐만 아니라 백업파일의 붙는 문자열도 정한다. SimpleDateFormat 형식에서 사용되는 콜론(:) 문자는 URL의 포트번호와 충돌하므로 사용하면 안된다. 문자열을 직접 출력하고자 할 때는 작은따옴표(single quote) 안에 문자를 기술하고 쌍이 되게끔 한다.

| pattern | description | eg : my.log |
|---|---|---|
| '.'yyyy-MM | 매달 새로운 로그파일 생성 | my.log.2007-02 |
| '.'yyyy-ww | 매주마다 // | my.log.2007-7 |
| '.'yyyy-MM-dd | 매일 // | my.log.2007-02-12 |
| '.'yyyy-MM-dd-a | 자정과 정오에 // | my.log.2007-02-12-오후 |
| '.'yyyy-MM-dd-HH | 매시간마다 // | my.log.2007-02-12-17 |
| '.'yyyy-MM-dd-HH-mm | 매분마다 // | my.log.2007-02-12-17-15 |
| '_bak'_yy_MM_dd'.log' | 매일 // | my.log_bak_07_02_12.log |
| ps : 1. 기존의 로그파일의 날짜를 검사하여 설정된 날짜패턴과 다르면 기존파일을 백업한다.<br>2. 파일로 저장되므로 파일명으로 부적합한 기호들은 피해야 한다.<br>eg: "\", "/", "<", ":", "?", "*" 등 | | |

표 3. DatePattern의 몇 가지 포맷

이외에도 여러 Appender가 있으며 해당 Appender는 API문서를 참조하기 바람.

## 7. Layout

- **PatternLayout**

  가장 많이 사용되는 Layout이다. 변환문자에 의해 출력형식을 지정할 수 있다.

  - ConversionPattern : 기본값 "%r [%t] %p %c %x - %m%n". 로그메세지의 포멧형식을 지정

  PatternLayout에서 사용하는 패턴은 변환문자를 기반으로 하여 출력 포멧을 지정할 수있다.

| Character | Char Mean | Meaning |
|---|---|---|
| %c | logger (category) | logger name 을 출력, 로거이름은 직접 계층적으로 지정하여 사용할 수 있는데 이때의 이름이다. 중괄호 "{ }" 와 그 안에 첨자를 사용함으로써 오른쪽에서 필요한 만큼만 출력 할 수도 있다.<br>eg) 로거이름이 com.malja.util 인 경우 %c{2}는 malja.util이 출력. |
| %C | class | 클래스명을 출력, %c 와 마찬가지로 중괄호 사용가능<br>eg) 클래스가 com.malja.SomeClass 인 경우 %C{2}는 malja.SomeClass 가 출력. |
| %d | date | 로깅이 발생한 시간을 기록한다. 출력포멧은 %d후에 중괄호 "{}" 에 SimpleDateFormat 의 형식대로 사용하면 된다.<br>eg) %d{yyyy-MM-dd HH:mm:ss} , %d{ISO8601} |
| %F | file | 로깅이 발생한 프로그램 파일명을 출력 |
| %l | location info | 로깅이 발생한 정보를 출력, "%C.%M(%F:%L)" 의 축약형 |
| %L | line | 로깅이 발생한 프로그램의 라인번호 출력 |
| %m | message | 로그 메시지 출력 |
| %M | method | 로깅이 발생한 method 이름을 출력 |
| %n | new line | 플랫폼 종속적인 개행문자가 출력된다. \r\n 또는 \n |
| %p | level(=priority) | Level (priority)을 출력 |
| %r | relative | 프로그램 시작이후 로깅이 발생할 때 까지의 시간(milliseconds) |
| %t | thread | 로그가 발생된 쓰레드의 이름이 출력 |
| %x | ndc | 로깅이 발생한 thread와 관련된 NDC(nested diagnostic context)를 출력 |
| %X | mdc | 로깅이 발생한 thread와 관련된 MDC(mapped diagnostic context)를 출력 |
| %% | | % 문자를 출력 |
| ps : 1. 왼쪽 정렬을 하려면 마이너스 기호(–) 를 사용한다.<br>2. 최소 필드 넓이를 사용하기 위해서는 숫자를 사용한다. (큰경우 자동확장)<br>3. 마침표(.) 다음에 숫자를 사용할 경우는 최대 필드 넓이로 인식한다. (큰경우 왼쪽이 지워짐)<br>eg : "**[%-4.10m]**" 으로 한 경우 왼쪽정렬, 최소 4글자, 최대 10글자의 메시지 출력<br>"Hi" -> "[Hi ]"<br>"Malja" -> "[Malja]"<br>"I Love Malja" -> "[Love Malja]" | | |

표 4. 패턴에서 사용하는 문자

- **DateLayout**

  - DateFormat : 기본값 RELATIVE. SimpleDateFormat의 형식이거나, NULL, RELATIVE, ABSOLUTE, DATE, ISO8601 중 하나.

NULL : 날짜와 시간을 출력하지 않는다

RELATIVE : 어플리케이션이 실행되고 나서의 걸린 시간

ABSOLUTE : 패턴 "HH:mm:ss, SSS"

DATE : 패턴 "dd MMM YYYY HH:mm:ss, SSS"

ISO8601 : 패턴 "YYYY-mm-dd HH:mm:ss, SSS"

- TimeZone : java.util.TimeZone.getTimeZone(String) 메서드에 사용할수 있는 문자열 (e.g., GMT-8:00)

- **TTCCLayout**

  DateLayout ← TTCCLayout

  - DateFormat, TimeZone → DateLayout
  - CategoryPrefixing : 기본값 true. category 이름을 출력한다.
  - ContextPrinting : 기본값 true. 현재 스레드에 속하는 NDC 정보를 출력한다.
  - ThreadPrinting : 기본값 true. 스레드명을 출력한다.

- **HTMLLayout**
  - LocatoinInfo : 기본값 false. true로 설정할 경우 호출한 해당 파일명과 행번호를 출력한다.
  - Title : 기본값 "Log4j Log Messages". HTML의 <title>태그에 셑팅된다.

- **XMLLayout**
  - LocatoinInfo : 기본값 false. true로 설정할 경우 호출한 해당 파일명과 행번호를 출력한다.

## 8. Miscellany

### [1] 로깅에 따른 비용(성능)문제

어플리케이션을 개발하는 동안이나 실행중에 의미있는 정보를 얻기 위해 로그를 기록하고 문제를 해결해 나갈수 있다. 하지만 로그작업 자체도 실행을 위해 시스템 자원을 사용한다. 즉, 효율적인 로그관리를 해야한다.

```
logger.debug(message1);
while(condition){
   ...
   logger.debug(message2);
} // while
logger.warn(message3);
```

위와 같은 코드가 있는 경우 운영 Level이 DEBUG 모드이하일 경우는 괜찮지만 INFO 이상인경우 위 코드는 쓸데 없이 자원을 소모한다. (내부적으로 레벨체크 및 기타 메시지를 위한 작업등)

해당 레벨을 체크하여 로그메시지를 기록하는 방식으로 대체하는 것이 좀더 효율적일 것이다.

```
// 클래스에 선언된  변수
private boolean isDebug = logger.isDebugEnabled();

if(isDebug){
     logger.debug(message1);
}
```

```
while(condition){
   ...
   if(isDebug){
      logger.debug(message2);
   }
   // 위 if 문을 다음과 같이 Level 비교를 해도 된다.
   // if(logger.getLevel().isGreaterOrEqual(Level.DEBUG)){
   //    logger.debug(message2);
   // }
} // while
logger.warn(message3);
```

### [2] NDC(Nested Diagnostic Contexts)의 사용

다음과 같은 경우를 생각해 봅시다.

- 동시에 여러 클라이언트가 접속하는 경우
- 대부분의 사용자는 정상작동하는데 특정 사용자만 에러가 나는 경우

이때 각각을 구분해 주기위해 사용자 아이디, IP, 세션ID등 클라이언트의 고유한 정보를 NDC에 등록한다. 보통 번역을 "내포 검사 항목" 이라 하는데 의미파악이 쉽지 않아서 그냥 NDC라고 하겠다. 이와 비슷한 역할을 하는 것이 MDC(Mapped Diagnostic Contexts)이다. NDC를 사용하기 위해서는 pop() 과 push() 메서드를 사용해야 한다. 또한 layout 패턴에 "%x" 를 사용해야 한다.

```
NDC.push("SesID:"+ session.getId());
// 로그관련 작업
NDC.pop(); // 해당 로그작업이 필요없어지는 시점에 pop메서드를 사용한다.
```

### [3] 설정파일의 이름(log4j.properties) 및 실행중 Level 변경

log4j는 자바로 작성되는 어플리케이션, 각종 프레임워크등 여러 곳에서 사용된다. 그렇다면 기본 설정파일인 log4j.properties 라는 파일명을 그냥 사용하면 원하지 않는 다른 곳에서 해당 로그가 발생하게 될것이다. 혹시라도 이러한 경우가 발생하는 경우를 대비해서 특별한 이름의 파일이름을 읽어서 Logger를 설정하는 것이 좋을것이다. 또한가지 중요한 문제는 실행중에 Level을 변경할수 있느냐의 문제인데 Logger가 설정파일을 읽어서 생성이 되었다면 그 이후에는 설정파일을 읽지 않는다는 것이다. 이러한 문제를 해결하기 위해서

- PropertyConfigurator.configureAndWatch(String fileName);
- PropertyConfigurator.configureAndWatch(String fileName,Long delayTime);

PropertyConfigurator클래스의 configureAndWatch 메서드를 사용하면 된다. 주의 할 것은 시작할 때 위의 configureAndWatch 메서드가 동작해야 하므로 웹인 경우는 서블릿의 init() 메서드에 구현하고 web.xml파일에서 서버가 기동하면서 해당 서블릿이 동작하게끔 하면 되고 어플리케이션 같은 경우 시작 메서드에서 하는 것이 좋다. 또한 각 클래스파일에서 logger 변수를 선언할 때 static으로 하지말고 그냥 private 으로 하면 된다.

### [4] tail 유틸리티(Windows)

로그가 기록된 파일에 추가되는 정보를 보기위해 Unix계열에서는 "**tail –f 로그파일**" 을 이용하는데 윈도우에서는 tail이라는 명령이 없으므로 해당 유틸리티를 다운받아 실행해야 한다.

- **UnxUtils** : http://unxutils.sourceforge.net/ tail 말고도 다른 Unix명령어 유틸 모음
- **TailMe** : http://www.dschensky.de/ 에서 메뉴중 Software/Staff - kleine Tools
- **Tail for Win32** : http://tailforwin32.sourceforge.net/

부가적인 기능(FTP접속등)등은 각 사이트의 문서를 찾아보면 될것이다.

## 9. 참고문서 및 사이트

- Log4J 소개 : http://logging.apache.org/log4j/1.2/manual.html
- Log4J 1.2 API : http://logging.apache.org/log4j/1.2/apidocs/index.html
- Vipan Singla : Don't Use System.out.println! Use Log4j.
  http://www.vipan.com/htdocs/log4jhelp.html
  한글 번역 : http://www.noct.pe.kr/sum/log4j.html
- Ceki Gülcü : The complete log4j manual (PDF format, not Free)
  http://www.qos.ch/shop/products/eclm/
- Vikran Goyal : Build Flexible Logs With log4j (JDBCAppender 및 Tip 유용)
  http://www.onjava.com/pub/a/onjava/2002/08/07/log4j.html
- Samudra Gupta : Logging in Java with the JDK 1.4 Logging API and Apache log4j
  [Apress출판] , eBook 도 있음
- ANT와 Log4J 활용 : http://network.hanb.co.kr/view.php?bi_id=533
- Log4j와의 만남 : http://network.hanb.co.kr/view.php?bi_id=950