**<http://logging.apache.org/log4j/2.x/manual/migration.html>**

<http://logging.apache.org/log4j/1.2/manual.html>

## 概述

### 1.1. 背景

在应用程序中添加日志记录总的来说基于三个目的：监视代码中变量的变化情况，周期性的记录到文件中供其他应用进行统计分析工作；跟踪代码运行时轨迹，作为日后审计的依据；担当集成开发环境中的调试器的作用，向文件或控制台打印代码的调试信息。

最普通的做法就是在代码中嵌入许多的打印语句，这些打印语句可以输出到控制台或文件中，比较好的做法就是构造一个日志操作类来封装此类操作，而不是让一系列的打印语句充斥了代码的主体。

### 1.2. Log4j简介

在强调可重用组件开发的今天，除了自己从头到尾开发一个可重用的日志操作类外，Apache为我们提供了一个强有力的日志操作包-Log4j。

Log4j是Apache的一个开放源代码项目，通过使用Log4j，我们可以控制日志信息输送的目的地是控制台、文件、GUI组件、甚至是套接口服务器、NT的事件记录器、UNIX Syslog守护进程等；我们也可以控制每一条日志的输出格式；通过定义每一条日志信息的级别，我们能够更加细致地控制日志的生成过程。最令人感兴趣的就是，这些可以通过一个配置文件来灵活地进行配置，而不需要修改应用的代码。

此外，通过Log4j其他语言接口，您可以在C、C++、.Net、PL/SQL程序中使用Log4j，其语法和用法与在Java程序中一样，使得多语言分布式系统得到一个统一一致的日志组件模块。而且，通过使用各种第三方扩展，您可以很方便地将Log4j集成到J2EE、JINI甚至是SNMP应用中。

本文介绍的Log4j版本是1.2.3。作者试图通过一个简单的客户/服务器Java程序例子对比使用与不使用Log4j 1.2.3的差别，并详细讲解了在实践中最常使用Log4j的方法和步骤。在强调可重用组件开发的今天，相信Log4j将会给广大的设计开发人员带来方便。加入到Log4j的队伍来吧！

**[回页首](http://www.ibm.com/developerworks/cn/java/l-log4j/index.html" \l "ibm-pcon)**

## 一个简单的例子

我们先来看一个简单的例子，它是一个用Java实现的客户/服务器网络程序。刚开始我们不使用Log4j，而是使用了一系列的打印语句，然后我们将使用Log4j来实现它的日志功能。这样，大家就可以清楚地比较出前后两个代码的差别。

### 2.1. 不使用Log4j

**2.1.1. 客户程序**

package log4j ;

import java.io.\* ;

import java.net.\* ;

/\*\*

\*

\* <p> Client Without Log4j </p>

\* <p> Description: a sample with log4j</p>

\* @version 1.0

\*/

public class ClientWithoutLog4j {

/\*\*

\*

\* @param args

\*/

public static void main ( String args [] ) {

String welcome = null;

String response = null;

BufferedReader reader = null;

PrintWriter writer = null;

InputStream in = null;

OutputStream out = null;

Socket client = null;

try {

client = new Socket ( "localhost", 8001 ) ;

System.out.println ( "info: Client socket: " + client ) ;

in = client.getInputStream () ;

out = client.getOutputStream () ;

} catch ( IOException e ) {

System.out.println ( "error: IOException : " + e ) ;

System.exit ( 0 ) ;

}

try{

reader = new BufferedReader( new InputStreamReader ( in ) ) ;

writer = new PrintWriter ( new OutputStreamWriter ( out ), true ) ;

welcome = reader.readLine () ;

System.out.println ( "debug: Server says: '" + welcome + "'" ) ;

System.out.println ( "debug: HELLO" ) ;

writer.println ( "HELLO" ) ;

response = reader.readLine () ;

System.out.println ( "debug: Server responds: '" + response + "'") ;

System.out.println ( "debug: HELP" ) ;

writer.println ( "HELP" ) ;

response = reader.readLine () ;

System.out.println ( "debug: Server responds: '" + response + "'" ) ;

System.out.println ( "debug: QUIT" ) ;

writer.println ( "QUIT" ) ;

} catch ( IOException e ) {

System.out.println ( "warn: IOException in client.in.readln()" ) ;

System.out.println ( e ) ;

}

try{

Thread.sleep ( 2000 ) ;

} catch ( Exception ignored ) {}

}

}

**2.1.2. 服务器程序**

package log4j ;

import java.util.\* ;

import java.io.\* ;

import java.net.\* ;

/\*\*

\*

\* <p> Server Without Log4j </p>

\* <p> Description: a sample with log4j</p>

\* @version 1.0

\*/

public class ServerWithoutLog4j {

final static int SERVER\_PORT = 8001 ; // this server's port

/\*\*

\*

\* @param args

\*/

public static void main ( String args [] ) {

String clientRequest = null;

BufferedReader reader = null;

PrintWriter writer = null;

ServerSocket server = null;

Socket socket = null;

InputStream in = null;

OutputStream out = null;

try {

server = new ServerSocket ( SERVER\_PORT ) ;

System.out.println ( "info: ServerSocket before accept: " + server ) ;

System.out.println ( "info: Java server without log4j, on-line!" ) ;

// wait for client's connection

socket = server.accept () ;

System.out.println ( "info: ServerSocket after accept: " + server ) ;

in = socket.getInputStream () ;

out = socket.getOutputStream () ;

} catch ( IOException e ) {

System.out.println( "error: Server constructor IOException: " + e ) ;

System.exit ( 0 ) ;

}

reader = new BufferedReader ( new InputStreamReader ( in ) ) ;

writer = new PrintWriter ( new OutputStreamWriter ( out ) , true ) ;

// send welcome string to client

writer.println ( "Java server without log4j, " + new Date () ) ;

while ( true ) {

try {

// read from client

clientRequest = reader.readLine () ;

System.out.println ( "debug: Client says: " + clientRequest ) ;

if ( clientRequest.startsWith ( "HELP" ) ) {

System.out.println ( "debug: OK!" ) ;

writer.println ( "Vocabulary: HELP QUIT" ) ;

}

else {

if ( clientRequest.startsWith ( "QUIT" ) ) {

System.out.println ( "debug: OK!" ) ;

System.exit ( 0 ) ;

}

else{

System.out.println ( "warn: Command '" +

clientRequest + "' not understood." ) ;

writer.println ( "Command '" + clientRequest

+ "' not understood." ) ;

}

}

} catch ( IOException e ) {

System.out.println ( "error: IOException in Server " + e ) ;

System.exit ( 0 ) ;

}

}

}

}

### 2.2. 迁移到Log4j

**2.2.1. 客户程序**

package log4j ;

import java.io.\* ;

import java.net.\* ;

// add for log4j: import some package

import org.apache.log4j.PropertyConfigurator ;

import org.apache.log4j.Logger ;

import org.apache.log4j.Level ;

/\*\*

\*

\* <p> Client With Log4j </p>

\* <p> Description: a sample with log4j</p>

\* @version 1.0

\*/

public class ClientWithLog4j {

/\*

add for log4j: class Logger is the central class in the log4j package.

we can do most logging operations by Logger except configuration.

getLogger(...): retrieve a logger by name, if not then create for it.

\*/

static Logger logger = Logger.getLogger

( ClientWithLog4j.class.getName () ) ;

/\*\*

\*

\* @param args : configuration file name

\*/

public static void main ( String args [] ) {

String welcome = null ;

String response = null ;

BufferedReader reader = null ;

PrintWriter writer = null ;

InputStream in = null ;

OutputStream out = null ;

Socket client = null ;

/\*

add for log4j: class BasicConfigurator can quickly configure the package.

print the information to console.

\*/

PropertyConfigurator.configure ( "ClientWithLog4j.properties" ) ;

// add for log4j: set the level

// logger.setLevel ( ( Level ) Level.DEBUG ) ;

try{

client = new Socket( "localhost" , 8001 ) ;

// add for log4j: log a message with the info level

logger.info ( "Client socket: " + client ) ;

in = client.getInputStream () ;

out = client.getOutputStream () ;

} catch ( IOException e ) {

// add for log4j: log a message with the error level

logger.error ( "IOException : " + e ) ;

System.exit ( 0 ) ;

}

try{

reader = new BufferedReader ( new InputStreamReader ( in ) ) ;

writer = new PrintWriter ( new OutputStreamWriter ( out ), true ) ;

welcome = reader.readLine () ;

// add for log4j: log a message with the debug level

logger.debug ( "Server says: '" + welcome + "'" ) ;

// add for log4j: log a message with the debug level

logger.debug ( "HELLO" ) ;

writer.println ( "HELLO" ) ;

response = reader.readLine () ;

// add for log4j: log a message with the debug level

logger.debug ( "Server responds: '" + response + "'" ) ;

// add for log4j: log a message with the debug level

logger.debug ( "HELP" ) ;

writer.println ( "HELP" ) ;

response = reader.readLine () ;

// add for log4j: log a message with the debug level

logger.debug ( "Server responds: '" + response + "'") ;

// add for log4j: log a message with the debug level

logger.debug ( "QUIT" ) ;

writer.println ( "QUIT" ) ;

} catch ( IOException e ) {

// add for log4j: log a message with the warn level

logger.warn ( "IOException in client.in.readln()" ) ;

System.out.println ( e ) ;

}

try {

Thread.sleep ( 2000 ) ;

} catch ( Exception ignored ) {}

}

}

**2.2.2. 服务器程序**

package log4j;

import java.util.\* ;

import java.io.\* ;

import java.net.\* ;

// add for log4j: import some package

import org.apache.log4j.PropertyConfigurator ;

import org.apache.log4j.Logger ;

import org.apache.log4j.Level ;

/\*\*

\*

\* <p> Server With Log4j </p>

\* <p> Description: a sample with log4j</p>

\* @version 1.0

\*/

public class ServerWithLog4j {

final static int SERVER\_PORT = 8001 ; // this server's port

/\*

add for log4j: class Logger is the central class in the log4j package.

we can do most logging operations by Logger except configuration.

getLogger(...): retrieve a logger by name, if not then create for it.

\*/

static Logger logger = Logger.getLogger

( ServerWithLog4j.class.getName () ) ;

/\*\*

\*

\* @param args

\*/

public static void main ( String args[]) {

String clientRequest = null ;

BufferedReader reader = null ;

PrintWriter writer = null ;

ServerSocket server = null ;

Socket socket = null ;

InputStream in = null ;

OutputStream out = null ;

/\*

add for log4j: class BasicConfigurator can quickly configure the package.

print the information to console.

\*/

PropertyConfigurator.configure ( "ServerWithLog4j.properties" ) ;

// add for log4j: set the level

// logger.setLevel ( ( Level ) Level.DEBUG ) ;

try{

server = new ServerSocket ( SERVER\_PORT ) ;

// add for log4j: log a message with the info level

logger.info ( "ServerSocket before accept: " + server ) ;

// add for log4j: log a message with the info level

logger.info ( "Java server with log4j, on-line!" ) ;

// wait for client's connection

socket = server.accept() ;

// add for log4j: log a message with the info level

logger.info ( "ServerSocket after accept: " + server ) ;

in = socket.getInputStream() ;

out = socket.getOutputStream() ;

} catch ( IOException e ) {

// add for log4j: log a message with the error level

logger.error ( "Server constructor IOException: " + e ) ;

System.exit ( 0 ) ;

}

reader = new BufferedReader ( new InputStreamReader ( in ) ) ;

writer = new PrintWriter ( new OutputStreamWriter ( out ), true ) ;

// send welcome string to client

writer.println ( "Java server with log4j, " + new Date () ) ;

while ( true ) {

try {

// read from client

clientRequest = reader.readLine () ;

// add for log4j: log a message with the debug level

logger.debug ( "Client says: " + clientRequest ) ;

if ( clientRequest.startsWith ( "HELP" ) ) {

// add for log4j: log a message with the debug level

logger.debug ( "OK!" ) ;

writer.println ( "Vocabulary: HELP QUIT" ) ;

}

else {

if ( clientRequest.startsWith ( "QUIT" ) ) {

// add for log4j: log a message with the debug level

logger.debug ( "OK!" ) ;

System.exit ( 0 ) ;

}

else {

// add for log4j: log a message with the warn level

logger.warn ( "Command '"

+ clientRequest + "' not understood." ) ;

writer.println ( "Command '"

+ clientRequest + "' not understood." ) ;

}

}

} catch ( IOException e ) {

// add for log4j: log a message with the error level

logger.error( "IOException in Server " + e ) ;

System.exit ( 0 ) ;

}

}

}

}

**2.2.3. 配置文件**

2.2.3.1. 客户程序配置文件

log4j.rootLogger=INFO, A1

log4j.appender.A1=org.apache.log4j.ConsoleAppender

log4j.appender.A1.layout=org.apache.log4j.PatternLayout

log4j.appender.A1.layout.ConversionPattern=%-4r %-5p [%t] %37c %3x - %m%n

2.2.3.2. 服务器程序配置文件

log4j.rootLogger=INFO, A1

log4j.appender.A1=org.apache.log4j.ConsoleAppender

log4j.appender.A1.layout=org.apache.log4j.PatternLayout

log4j.appender.A1.layout.ConversionPattern=%-4r %-5p [%t] %37c %3x - %m%n

### 2.3. 比较

比较这两个应用可以看出，采用Log4j进行日志操作的整个过程相当简单明了，与直接使用System.out.println语句进行日志信息输出的方式相比，基本上没有增加代码量，同时能够清楚地理解每一条日志信息的重要程度。通过控制配置文件，我们还可以灵活地修改日志信息的格式，输出目的地等等方面，而单纯依靠System.out.println语句，显然需要做更多的工作。

下面我们将以前面使用Log4j的应用作为例子，详细讲解使用Log4j的主要步骤。

**[回页首](http://www.ibm.com/developerworks/cn/java/l-log4j/index.html" \l "ibm-pcon)**

## Log4j基本使用方法

Log4j由三个重要的组件构成：日志信息的优先级，日志信息的输出目的地，日志信息的输出格式。日志信息的优先级从高到低有ERROR、WARN、INFO、DEBUG，分别用来指定这条日志信息的重要程度；日志信息的输出目的地指定了日志将打印到控制台还是文件中；而输出格式则控制了日志信息的显示内容。

### 3.1.定义配置文件

其实您也可以完全不使用配置文件，而是在代码中配置Log4j环境。但是，使用配置文件将使您的应用程序更加灵活。

Log4j支持两种配置文件格式，一种是XML格式的文件，一种是Java特性文件（键=值）。下面我们介绍使用Java特性文件做为配置文件的方法：

1. 配置根Logger，其语法为：

log4j.rootLogger = [ level ] , appenderName, appenderName, …

1. 其中，level 是日志记录的优先级，分为OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL或者您定义的级别。Log4j建议只使用四个级别，优先级从高到低分别是ERROR、WARN、INFO、DEBUG。通过在这里定义的级别，您可以控制到应用程序中相应级别的日志信息的开关。比如在这里定义了INFO级别，则应用程序中所有DEBUG级别的日志信息将不被打印出来。   
   appenderName就是指定日志信息输出到哪个地方。您可以同时指定多个输出目的地。
2. 配置日志信息输出目的地Appender，其语法为

log4j.appender.appenderName = fully.qualified.name.of.appender.class

log4j.appender.appenderName.option1 = value1

…

log4j.appender.appenderName.option = valueN

1. 其中，Log4j提供的appender有以下几种：   
   org.apache.log4j.ConsoleAppender（控制台），   
   org.apache.log4j.FileAppender（文件），   
   org.apache.log4j.DailyRollingFileAppender（每天产生一个日志文件），org.apache.log4j.RollingFileAppender（文件大小到达指定尺寸的时候产生一个新的文件），   
   org.apache.log4j.WriterAppender（将日志信息以流格式发送到任意指定的地方）
2. 配置日志信息的格式（布局），其语法为：

log4j.appender.appenderName.layout = fully.qualified.name.of.layout.class

log4j.appender.appenderName.layout.option1 = value1

…

log4j.appender.appenderName.layout.option = valueN

1. 其中，Log4j提供的layout有以下几种：   
   org.apache.log4j.HTMLLayout（以HTML表格形式布局），   
   org.apache.log4j.PatternLayout（可以灵活地指定布局模式），   
   org.apache.log4j.SimpleLayout（包含日志信息的级别和信息字符串），   
   org.apache.log4j.TTCCLayout（包含日志产生的时间、线程、类别等等信息）

### 3.2.在代码中使用Log4j

下面将讲述在程序代码中怎样使用Log4j。

**3.2.1.得到记录器**

使用Log4j，第一步就是获取日志记录器，这个记录器将负责控制日志信息。其语法为：

public static Logger getLogger( String name)，

通过指定的名字获得记录器，如果必要的话，则为这个名字创建一个新的记录器。Name一般取本类的名字，比如：

static Logger logger = Logger.getLogger ( ServerWithLog4j.class.getName () ) ;

**3.2.2.读取配置文件**

当获得了日志记录器之后，第二步将配置Log4j环境，其语法为：   
BasicConfigurator.configure ()： 自动快速地使用缺省Log4j环境。   
PropertyConfigurator.configure ( String configFilename) ：读取使用Java的特性文件编写的配置文件。   
DOMConfigurator.configure ( String filename ) ：读取XML形式的配置文件。

**3.2.3.插入记录信息（格式化日志信息）**

当上两个必要步骤执行完毕，您就可以轻松地使用不同优先级别的日志记录语句插入到您想记录日志的任何地方，其语法如下：

Logger.debug ( Object message ) ;   
Logger.info ( Object message ) ;   
Logger.warn ( Object message ) ;   
Logger.error ( Object message ) ;