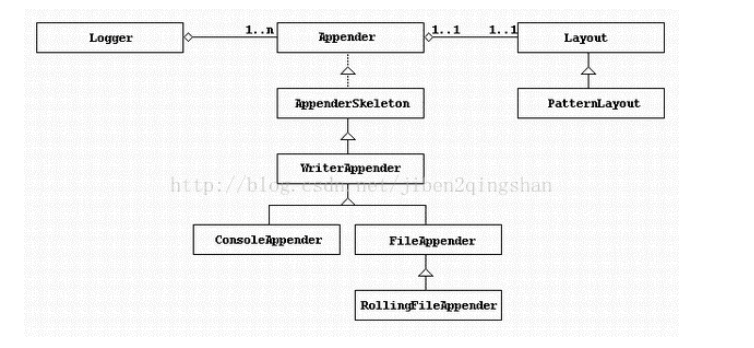
**Log4j**

## 一、Log4j介绍

**1 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。

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。



**2 Logger类**

logger类的作用就是供客户端代码调用。在配置文件(配置文件或者xml文件)进行相应的配置，配置的Logger分为两类，一类是rootLogger，一类是自定义。rootLogger为根logger，配置文件中可以配置多个Logger，且默认情况下，其它的Logger都是继承rootLogger的配置，我们也可以手动取消该继承。

**3 Appender类**

日志的输出(存储的位置)，日志的输出方式(输出到数据库，输出到Console，等)。一个Logger可以对应多个Appender类，即：一条日志可以被输出到多个地方。

**4 Layout类**

日志信息的格式化(日志格式化器)，将程序员的日志格式成字符串。Appender指定输出的位置，Layout指定输出哪些内容和格式。当Layout分为两种类型，分为两种两类型，一种是格式和内容定义好的，一种是自定义格式和内容的，当使用自定义格式时，需要ConversionPattern的配合。

## 二、Log4j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properties格式的文件。**下面我们介绍使用properties格式做为配置文件的方法：

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

**1. 配置根Logger，其语法为：**

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

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

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）

**(1).ConsoleAppender选项**

Threshold=WARN:指定日志消息的输出最低层次。

ImmediateFlush=true:默认值是true,意谓着所有的消息都会被立即输出。

Target=System.err：默认情况下是：System.out,指定输出控制台

**(2).FileAppender 选项**

Threshold=WARN:指定日志消息的输出最低层次。

ImmediateFlush=true:默认值是true,意谓着所有的消息都会被立即输出。

File=mylog.txt:指定消息输出到mylog.txt文件。

Append=false:默认值是true,即将消息增加到指定文件中，false指将消息覆盖指定的文件内容。

**(3).DailyRollingFileAppender 选项**

Threshold=WARN:指定日志消息的输出最低层次。

ImmediateFlush=true:默认值是true,意谓着所有的消息都会被立即输出。

File=mylog.txt:指定消息输出到mylog.txt文件。

Append=false:默认值是true,即将消息增加到指定文件中，false指将消息覆盖指定的文件内容。

DatePattern=’.'yyyy-ww:每周滚动一次文件，即每周产生一个新的文件。当然也可以指定按月、周、天、时和分。即对应的格式如下：

1)’.'yyyy-MM: 每月

2)’.'yyyy-ww: 每周

3)’.'yyyy-MM-dd: 每天

4)’.'yyyy-MM-dd-a: 每天两次

5)’.'yyyy-MM-dd-HH: 每小时

6)’.'yyyy-MM-dd-HH-mm: 每分钟

**(4).RollingFileAppender 选项**

Threshold=WARN:指定日志消息的输出最低层次。

ImmediateFlush=true:默认值是true,意谓着所有的消息都会被立即输出。

File=mylog.txt:指定消息输出到mylog.txt文件。

Append=false:默认值是true,即将消息增加到指定文件中，false指将消息覆盖指定的文件内容。

MaxFileSize=100KB: 后缀可以是KB, MB 或者是 GB. 在日志文件到达该大小时，将会自动滚动，即将原来的内容移到mylog.log.1文件。

MaxBackupIndex=2:指定可以产生的滚动文件的最大数。

**3. 配置日志信息的布局，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

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）

**4、输出格式设置**

在配置文件中可以通过log4j.appender.A1.layout.ConversionPattern设置日志输出格式。

%p: 输出日志信息优先级，即DEBUG，INFO，WARN，ERROR，FATAL,

%d: 输出日志时间点的日期或时间，默认格式为ISO8601，也可以在其后指定格式，比如：%d{yyy MMM dd HH:mm:ss,SSS}，输出类似：2002年10月18日 22：10：28，921

%r: 输出自应用启动到输出该log信息耗费的毫秒数

%c: 输出日志信息所属的类目，通常就是所在类的全名

%t: 输出产生该日志事件的线程名

%l: 输出日志事件的发生位置，相当于%C.%M(%F:%L)的组合,包括类目名、发生的线程，以及在代码中的行数。举例：Testlog4.main(TestLog4.java:10)

%x: 输出和当前线程相关联的NDC(嵌套诊断环境),尤其用到像java servlets这样的多客户多线程的应用中。

%%: 输出一个”%”字符

%F: 输出日志消息产生时所在的文件名称

%L: 输出代码中的行号

%m: 输出代码中指定的消息,产生的日志具体信息

%n: 输出一个回车换行符，Windows平台为”\r\n”，Unix平台为”\n”输出日志信息换行

可以在%与模式字符之间加上修饰符来控制其最小宽度、最大宽度、和文本的对齐方式。如：

1)%20c：指定输出category的名称，最小的宽度是20，如果category的名称小于20的话，默认的情况下右对齐。

2)%-20c:指定输出category的名称，最小的宽度是20，如果category的名称小于20的话，”-”号指定左对齐。

3)%.30c:指定输出category的名称，最大的宽度是30，如果category的名称大于30的话，就会将左边多出的字符截掉，但小于30的话也不会有空格。

4)%20.30c:如果category的名称小于20就补空格，并且右对齐，如果其名称长于30字符，就从左边交远销出的字符截掉。

**5、log4j的组成、配置及使用**

1）主要由3个重要的组件组成部分：Logger组件、Appender组件、Layout组件

a、 Logger组件：日志器，里面提供了一些方法，用于往外输出日志

log4j.additivity.org.apache=false：表示Logger不会在父Logger的appender里输出，默认为true。

b、Appender组件：用于指定信息输出方式，例如控制台输出、文件输出、流输出等

**(1)asyncAppender：异步记日志**

**(2)fileAppender：文件**

选项：

1) Threshold=WARN 指定日志消息的输出最低层次。

2) ImmediateFlush=true 默认值是true,意谓着所有的消息都会被立即输出。

3) File=mylog.txt 指定消息输出到mylog.txt文件。

4) Append=false 默认值是true,即将消息追加到指定文件中，false指将消息覆盖指定的文件内容。

例如：

log4j.rootLogger = debug,stdout,D,E

log4j.appender.D = org.apache.log4j.FileAppender

log4j.appender.D.File = E://logs/log.log

log4j.appender.D.Append = true

log4j.appender.D.Threshold = DEBUG

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

log4j.appender.D.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss} [ %t:%r ] - [ %p ]

**(3)consoleAppender：控制台记录日志**

选项：

(1) Threshold=WARN 指定日志消息的输出最低层次。

(2) ImmediateFlush=true 默认值是true,意谓着所有的消息都会被立即输出。

(3)Target=System.out 默认情况下是：System.out,指定输出控制台

例如：

log4j.rootLogger = debug,stdout

log4j.appender.stdout = org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target = System.out

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

log4j.appender.stdout.layout.ConversionPattern = [%-5p] %d{yyyy-MM-dd HH:mm:ss,SSS} meth

**(4)DailyRollingFileAppender：每天产生一个日志文件**

选项：

(1)Threshold=WARN 指定日志消息的输出最低层次。

(2)ImmediateFlush=true 默认值是true,意谓着所有的消息都会被立即输出。

(3)File=mylog.txt 指定消息输出到mylog.txt文件。

(4)Append=false 默认值是true,即以追加的方式将消息增加到指定文件中，false指将消息覆盖指定的文件内容。

(5)DatePattern=''.''yyyy-ww 每周滚动一次文件，即每周产生一个新的文件。当然也可以指定按月、周、天、时和分

1)''.''yyyy-MM: 每月

2)''.''yyyy-ww: 每周

3)''.''yyyy-MM-dd: 每天

4)''.''yyyy-MM-dd-a: 每天两次

5)''.''yyyy-MM-dd-HH: 每小时

6)''.''yyyy-MM-dd-HH-mm: 每分钟

例如：

log4j.appender.file=org.apache.log4j.DailyRollingFileAppender

log4j.appender.file.DatePattern='.'yyyy-MM-dd

log4j.appender.file.File=D：\\run.log

当前月的日志文件名为run.log，前一个月的日志文件名为run.log.yyyy-MM-dd

**(5)RollingFileAppender：文件大小到达指定尺寸的时候产生一个新的文件**

选项：

(1)Threshold=WARN 指定日志消息的输出最低层次。

(2)ImmediateFlush=true 默认值是true,意谓着所有的消息都会被立即输出。

(3)File=mylog.txt 指定消息输出到mylog.txt文件。

(4)Append=false 默认值是true,即将消息增加到指定文件中，false指将消息覆盖指定的文件内容。

(5)MaxFileSize=100KB 后缀可以是KB, MB 或者是 GB. 在日志文件到达该大小时，将会自动滚动 。

(6)MaxBackupIndex=2 指定可以产生的滚动文件的最大数。

log4j.appender.R.MaxFileSize=10MB

log4j.appender.R.MaxBackupIndex=10

一个日志文件最大10M，到了10M后自动创建一个新的文件，但最多创建10个，超过10个后自动删除最旧的那个。

**(6)WriterAppender：将日志信息以流格式发送到任意指定的地方**

log4j.rootCategory= DEBUG,WriterAppender

log4j.appender.WriterAppender=org.apache.log4j.WriterAppender

log4j.appender.WriterAppender.Threshold=debug

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

log4j.appender.WriterAppender.layout.ConversionPattern=%d{ISO8601} - %p - %m%n

c、Layout组件：用于指定信息输出格式

(1)HTMLLayout：以HTML表格形式布局

选项：

LocationInfo=true:默认值是false,输出java文件名称和行号

Title=my app file: 默认值是 Log4J Log Messages.

(2) PatternLayout：可以灵活地指定布局模式

选项：

ConversionPattern=%m%n :指定怎样格式化指定的消息

(3)SimpleLayout：包含日志信息的级别和信息字符串

(4)TTCCLayout：包含日志产生的时间、线程、类别等等信息

(5)XMLLayout: 以XML格式进行输出

选项：LocationInfo=true:默认值是false,输出java文件名称和行号

三个组件之间的关系：一个logger可以有多个Appender。然后每个Appender只有一个layout

2) 配置日志信息的格式

#src/log4j.properties

log4j.rootLogger = DEBUG,console,file

#appender

log4j.appender.console = org.apache.log4j.ConsoleAppender

log4j.appender.file = org.apache.log4j.FileAppender

log4j.appender.file.File = D\:\\log4j.html

#layout

log4j.appender.console.layout = org.apache.log4j.SimpleLayout

log4j.appender.file.layout =org.apache.log4j.HTMLLayout

第一个参数定义消息级别，将大于等于这个消息级别的信息进行输出

第二个参数定义输出方式，只是一个标识符，通过这个标识符来指定输出方式

第三个组件指定信息输出格式

## 三、在程序中的使用

在程序中使用Log4j之前，首先要将commons-logging.jar和logging-log4j-1.2.9.jar导入到classpath中，并将log4j.properties放于src根目录中。接下来就可以使用了。

**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 () ) ;

注：推荐使用commons-logging结合log4j进行日志记录

private static Log logger = LogFactory.getLog(Yourclass.class);

**2.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 ) ;

## 四、Log4j比较全面的配置

LOG4J的配置之简单使它遍及于越来越多的应用中了：Log4J配置文件实现了输出到控制台、文件、回滚文件、发送日志邮件、输出到数据库日志表、自定义标签等全套功能。择其一二使用就够用了。

1.log4j.rootLogger=DEBUG,CONSOLE,A1,im

2.log4j.addivity.org.apache=true

3.# 应用于控制台

4.log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender

5.log4j.appender.Threshold=DEBUG

6.log4j.appender.CONSOLE.Target=System.out

7.log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout

8.log4j.appender.CONSOLE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p %c %x - %m%n

9.#log4j.appender.CONSOLE.layout.ConversionPattern=[start]%d{DATE}[DATE]%n%p[PRIORITY]%n%x[NDC]%n%t[thread] n%c[CATEGORY]%n%m[MESSAGE]%n%n

10.#应用于文件

11.log4j.appender.FILE=org.apache.log4j.FileAppender

12.log4j.appender.FILE.File=file.log

13.log4j.appender.FILE.Append=false

14.log4j.appender.FILE.layout=org.apache.log4j.PatternLayout

15.log4j.appender.FILE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p %c %x - %m%n

16.# Use this layout for LogFactor 5 analysis

17.# 应用于文件回滚

18.log4j.appender.ROLLING\_FILE=org.apache.log4j.RollingFileAppender

19.log4j.appender.ROLLING\_FILE.Threshold=ERROR

20.log4j.appender.ROLLING\_FILE.File=rolling.log //文件位置,也可以用变量${java.home}、rolling.log

21.log4j.appender.ROLLING\_FILE.Append=true //true:添加 false:覆盖

22.log4j.appender.ROLLING\_FILE.MaxFileSize=10KB //文件最大尺寸

23.log4j.appender.ROLLING\_FILE.MaxBackupIndex=1 //备份数

24.log4j.appender.ROLLING\_FILE.layout=org.apache.log4j.PatternLayout

25.log4j.appender.ROLLING\_FILE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p %c %x - %m%n

26.

27.#应用于socket

28.log4j.appender.SOCKET=org.apache.log4j.RollingFileAppender

29.log4j.appender.SOCKET.RemoteHost=localhost

30.log4j.appender.SOCKET.Port=5001

31.log4j.appender.SOCKET.LocationInfo=true

32.# Set up for Log Facter 5

33.log4j.appender.SOCKET.layout=org.apache.log4j.PatternLayout

34.log4j.appender.SOCET.layout.ConversionPattern=[start]%d{DATE}[DATE]%n%p[PRIORITY]%n%x[NDC]%n%t[thread]%n%c[CATEGORY]%n%m[MESSAGE]%n%n

35.

36.# Log Factor 5 Appender

37.log4j.appender.LF5\_APPENDER=org.apache.log4j.lf5.LF5Appender

38.log4j.appender.LF5\_APPENDER.MaxNumberOfRecords=2000

39.# 发送日志给邮件

40.log4j.appender.MAIL=org.apache.log4j.net.SMTPAppender

41.log4j.appender.MAIL.Threshold=FATAL

42.log4j.appender.MAIL.BufferSize=10

43.log4j.appender.MAIL.From=web@www.wuset.com

44.log4j.appender.MAIL.SMTPHost=www.wusetu.com

45.log4j.appender.MAIL.Subject=Log4J Message

46.log4j.appender.MAIL.To=web@www.wusetu.com

47.log4j.appender.MAIL.layout=org.apache.log4j.PatternLayout

48.log4j.appender.MAIL.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p %c %x - %m%n

49.# 用于数据库

50.log4j.appender.DATABASE=org.apache.log4j.jdbc.JDBCAppender

51.log4j.appender.DATABASE.URL=jdbc:mysql://localhost:3306/test

52.log4j.appender.DATABASE.driver=com.mysql.jdbc.Driver

53.log4j.appender.DATABASE.user=root

54.log4j.appender.DATABASE.password=

55.log4j.appender.DATABASE.sql=INSERT INTO LOG4J (Message) VALUES (’[framework] %d - %c -%-4r [%t] %-5p %c %x - %m%n’)

56.log4j.appender.DATABASE.layout=org.apache.log4j.PatternLayout

57.log4j.appender.DATABASE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p %c %x - %m%n

58.

59.log4j.appender.A1=org.apache.log4j.DailyRollingFileAppender

60.log4j.appender.A1.File=SampleMessages.log4j

61.log4j.appender.A1.DatePattern=yyyyMMdd-HH’.log4j’

62.log4j.appender.A1.layout=org.apache.log4j.xml.XMLLayout

63.#自定义Appender

64.log4j.appender.im = net.cybercorlin.util.logger.appender.IMAppender

65.log4j.appender.im.host = mail.cybercorlin.net

66.log4j.appender.im.username = username

67.log4j.appender.im.password = password

68.log4j.appender.im.recipient = corlin@cybercorlin.net

69.log4j.appender.im.layout=org.apache.log4j.PatternLayout

70.log4j.appender.im.layout.ConversionPattern =[framework] %d - %c -%-4r [%t] %-5p %c %x - %m%n

## 五、commons-logging + Log4j的使用(推荐使用)

**1、为什么要使用commons-logging + Log4j**

commons-logging是Apache下的开源项目，目的是为“所有的Java日志实现”提供一个统一的接口，使项目与日志实现工具解耦。但它自身的日志功能比较弱（只有一个简单的 SimpleLog），所以一般不会单独使用它。

Log4j的功能非常强大，是目前最主流的java日志工具。结合使用两者可以避免使项目与log4j紧密耦合，用户既可以轻松切换到其他日志工具，同时又可以使用log4j的强大功能。强调一点，“同时使用commons-logging和Log4j”，与“单独使用Log4j”相比，并不会带来更大的学习、配置和维护成本，反而更加简化了我们的工作。这也是为什么“所有用到Log4j的项目一般也同时会用到commons-loggin”的原因之一吧。

**2、如何使用commons-logging + Log4j**

(1)引入commons-logging-1.2.jar、(log4j-1.2.17.jar)

(2)在src根目录下放进commons-logging.properties文件，进行配置哪个第三方日志包。

#定义了使用的具体第三方的日志包

#org.apache.commons.logging.Log=org.apache.commons.logging.impl.SimpleLog

org.apache.commons.logging.Log=org.apache.commons.logging.impl.Log4JLogger

#org.apache.commons.logging.Log=org.apache.commons.logging.impl.NoOpLog

#org.apache.commons.logging.Log=org.apache.commons.logging.impl.LogKitLogge

#org.apache.commons.logging.Log=org.apache.commons.logging.impl.Jdk14Logger

#org.apache.commons.logging.Log=org.apache.commons.logging.impl.AvalonLogge

(3)程序中调用

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

**public static Log log = LogFactory.getLog(test.class);**

**log.error("成功关闭链接");**

LogFactory.getLog(test.class)的调用过程:调用该函数会启动一个发现过程，即找出必需的底层日志记录功能的实现，具体的发现过程在下面列出

1) 首先在classpath下寻找自己的配置文件commons-logging.properties，如果找到，则使用其中定义的Log实现类;

2) 如果找不到commons-logging.properties文件，则在查找是否已定义系统环境变量org.apache.commons.logging.Log，找到则使用其定义的Log实现类;

3) 否则，查看classpath中是否有Log4j的包，如果发现，则自动使用Log4j作为日志实现类;

4) 否则，使用JDK自身的日志实现类(JDK1.4以后才有日志实现类);

5) 否则，使用commons-logging自己提供的一个简单的日志实现类SimpleLog;

**3、通常会见到两种使用方法：**

（1）使用org.apache.commons.logging.Log类：

例如private final Log log = LogFactory.getLog(getClass());

（2）直接使用log4j的Logger类：

获得rootLogger：Logger rootLogger=Logger.getRootLogger();­

获得自定义Logger：Logger myLogger =Logger.getLogger("log4j.logger.myLogger");­

（3）区别

LogFactory来自common-logging包。如果用LogFactory.getLog:

优点：你可以用任何实现了通用日志接口的日志记录器替换log4j，而程序不受影响。

缺点：需要多一个common-logging包

Logger来自log4j自己的包。如果用Logger.getLogger：

优点：你只需要一个log4j的jar包。

缺点：你只能用log4j作为日志记录器，因为log4j已经硬编码进你的系统了。

apache的common-logging包是通用日志接口，通过这个中间层，你可以通过其配置文件commons-logging.properties，来指定到底用哪个日志系统，以此增加系统的灵活性。

**4、Logger.getLogger和LogFactory.getLog的区别**

LogFactory来自common-logging包。如果用LogFactory.getLog，你可以用任何实现了通用日志接口的日志记录器替换log4j，而程序不受影响。apache的common-logging包是通用日志接口，通过这个中间层，你可以随便指定到底用哪个日志系统。增加系统的灵活性。若log4j它不存在, commons-logging 会另行选择其它的日志实现类。 这样保证了程序中不一定要使用log4j这个日志文件了

增强灵活性的理由：

1)首先在 classpath 下寻找自己的配置文件 commons-logging.properties ，如果找到，则使用其中定义的 Log 实现类；

2)如果找不到 commons-logging.properties 文件，则在查找是否已定义系统环境变量 org.apache.commons.logging.Log ，找到则使用其定义的 Log 实现类；

3)否则，查看 classpath 中是否有 Log4j 的包，如果发现，则自动使用 Log4j 作为日志实现类；

4)否则，使用 JDK 自身的日志实现类（ JDK1.4 以后才有日志实现类）；

5)否则，使用 commons-logging 自己提供的一个简单的日志实现类 SimpleLog ；

**为了简化配置 commons-logging ，一般不使用 commons-logging 的配置文件，也不设置与 commons-logging 相关的系统环境变量，而只需将 Log4j 的 Jar 包放置到 classpash 中就可以了。这样就很简单地完成了 commons-logging 与 Log4j 的融合。**

## 六、按日志文件大小、日期切分日志文件

项目中尽管对log4j有基本的配置，例如按天生成日志文件以作区分，但如果系统日志文件过大，则就需要考虑以更小的单位切分或者其他切分方式。下面就总结一下log4j常用的配置参数以及切分日志的不同方式。

**1、基本配置解析**

（1）常用配置1：

#日志级别，还可以log4j.rootLogger=INFO,file2,file3 指定file2、3的输出文件

log4j.rootLogger=info,exception,thread

#info

#每一天产生1个日志文件

log4j.appender.info=org.apache.log4j.DailyRollingFileAppender

#定义日志存放路径

log4j.appender.info.File=/usr/tomcat\_logs/mailSystem\_info.log

#日志文件是否追加

log4j.appender.info.Append=true

#定义的时间格式，如果时间定义到分钟（mm）就是每分钟生成一个日志文件，而这里定义的这个格式就是日志名后缀

#log4j.appender.file3.DatePattern='\_' yyyy-MM-dd-HH-mm 每分钟

#log4j.appender.file3.DatePattern='\_' yyyy-MM-dd-HH 每小时

#log4j.appender.file3.DatePattern='\_' yyyy-MM-dd 每天

#log4j.appender.file3.DatePattern='\_' yyyy-MM-dd-a 每半天

#日志输出级别

log4j.appender.info.Threshold=INFO

#日志编码

#log4j.appender.file3.Encoding=UTF-8

#日志中输出的日志的格式

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

#定义的日志格式

log4j.appender.info.layout.ConversionPattern=%t %d %p [%c] - %m%n

（2）常用配置2：

#debug

log4j.appender.debug=org.apache.log4j.RollingFileAppender

log4j.appender.debug.File=/usr/tomcat\_logs/mailSystem\_debug.log

#设置日志文件的大小

log4j.appender.debug.MaxFileSize=100M

#保存10个备份文件

log4j.appender.debug.MaxBackupIndex=10

log4j.appender.debug.Append=true

log4j.appender.debug.Threshold=DEBUG

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

log4j.appender.debug.layout.ConversionPattern=%t %d %p [%c] - %m%n

配置1采用DailyRollingFileAppender类，可定义日志按照log4j给定的参数值进行配置，配置2使用RollingFileAppender。两者都为log4j配置的核心类。在下进行详细介绍。

**2、配置核心类解析**

**（1）CLASS1：DailyRollingFileAppender**

在配置1中使用到DailyRollingFileAppender，该类继承与FileAppender，so that the underlying file is rolled over at a user chosen frequency.

The rolling schedule is specified by the “DatePattern” option。使用这些rolling schedule 需要注意一个前缀问题。例如if the “File” option is set to “/foo/bar.log” and the “DatePattern” set to '.'yyyy-MM-dd ,则到了midnight，the logging file /foo/bar.log will be copied to "/foo/bar.log.2017-01-01",也就是在bar.log文件后添加 ".yyyy-MM-dd"后缀。It is possible to specify monthly, weekly,half-daily, daily,hourly, or minutely rollover schedules.

总共有：



public DailyRollingFileAppender (Layout layout, String filename,

String datePattern) throws IOException {

super(layout, filename, true);

this.datePattern = datePattern;

activateOptions();

}

这些参数策略会传入到DailyRollingFileAppender的初始化方法中，创建一个DailyRollingFileAppender对象。

**（2）CLASS2：RollingFileAppender**

配置2中使用到另一个类RollingFileAppender，这个类也继承子FileAppender，用于按照一定的size文件大小，back up 日志文件。

这个类中起关键作用的两个参数maxFileSize和maxBackupIndex，用于确定每个日志文件的size大小和备份个数。默认分别为10MB和1.按照设置的index大小，会默认将文件copy to .1,.2等文件后缀进行备份。

**（3）CLASS3： PatternLayout**

在两个配置中，均使用PatternLayout和ConventionPattern类，PatternLayout主要用于日志输出相关，ConventionPattern用于定义日志文件的输出格式。其中ConventionPattern支持多种类型的日志输出，如线程信息、类信息，均已%打头进行配置，详情如下：

%c 输出日志信息所属的类的全名，例如%c{2}，对于com.max.spring.demo,便会输出spring.demo两级类信息。

%d 输出日志时间点的日期或时间，默认格式为ISO8601，也可以在其后指定格式，比如：%d{yyy-MM-dd HH:mm:ss }，输出类似：2002-10-18- 22：10：28

%f 输出日志信息所属的类的类名

%l 输出日志事件的发生位置，即输出日志信息的语句处于它所在的类的第几行

%m 输出代码中指定的信息，如log(message)中的message

%n 输出一个回车换行符，Windows平台为“rn”，Unix平台为“n”

%p 输出优先级，即DEBUG，INFO，WARN，ERROR，FATAL。如果是调用debug()输出的，则为DEBUG，依此类推

%r 输出自应用启动到输出该日志信息所耗费的毫秒数

%t 输出产生该日志事件的线程名

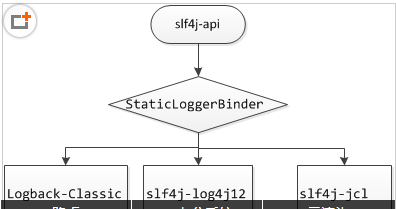


**Slf4j**

## 一、日志系统兼容

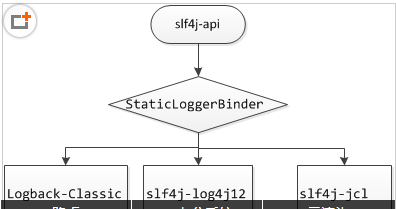
**1 在java的体系里**，主要有**slf4j和common-logging**两种日志体系接口。实现的框架有很多，主流的诸如logback、log4j等。

**2 common-logging挂载的日志实现常为log4j**。在初始化的时候，如果没有特殊指定要挂载谁，会自动按上图顺序去实例化实现，log4jLogger封装了log4j的logger，打印日志的时候调用到log4j过去。



**3 slf4j作为接口定义**，下面有很多种实现。实现原理是获取ILoggerFactory时执行初始化，初始化过程绑定实现对象：load出所有实现StaticLoggerBinder的类，然后获取他的单例，后面执行getLogger的时候都是调用这个单例类的方法获取对应有具体实现日志功能的Logger对象。如果有多个实现，之后绑定其中的一个。这种情况需要排除掉不需要的日志实现类。

SLF4J，即简单日志门面（Simple Logging Facade for Java），不是具体的日志解决方案，它只服务于各种各样的日志系统。按照官方的说法，SLF4J是一个用于日志系统的简单Facade，允许最终用户在部署其应用时使用其所希望的日志系统。他是一套包装Logging 框架的接口程式，**以外观模式实现。可在软件部署的时候决定要使用的 Logging 框架**，目前主要支援的有Java Logging API、log4j及logback等框架。以MIT 授权方式发布。



**4 slf4j优点**

@只有五种日志级别,没有Fatal级别

@格式化输出,不像log4j那样使用+号拼接字符串,它是"{} is larger than {}",a ,b这种形式格式化的

logger.debug("There are now {} user accounts: {}", count, userAccountList);

@没有logger.isDebugable()的判断,代码更加优美

@slf4j只是种接口,他的实现可以是各种log框架,这样就可以方便地进行log框架迁移

**5 logger对象获取**

（1）SLF4J获得logger对象：

private static final Logger logger = LoggerFactory.getLogger(Test.class);

输出日志信息：

logger.debug(“debug”);

（2）LOG4J获得logger对象：

public class A {

private static Logger logger = Logger.getLogger(A.class);

}

**6 slf4j和log4j总结**

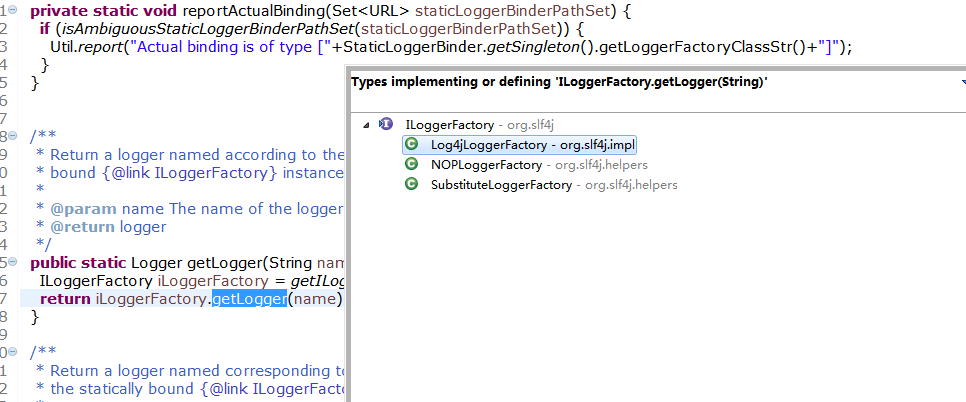
（1）大部分人在程序里面会去写logger.error(exception),其实这个时候log4j会去把这个exception tostring。真正的写法应该是logger(message.exception);而slf4j就不会使得程序员犯这个错误。

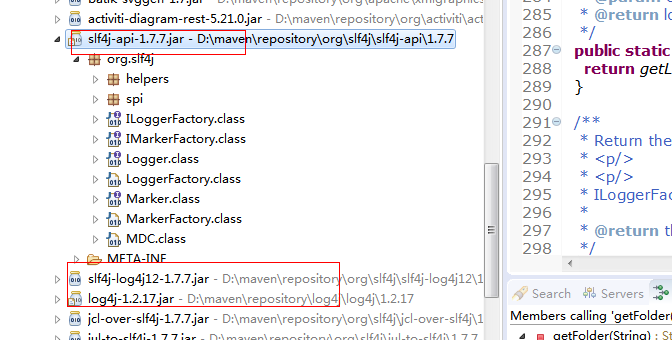
（2）log4j间接的在鼓励程序员使用string相加的写法，而slf4j就不会有这个问题。你可以使用logger.error("{} is+serviceid",serviceid)。

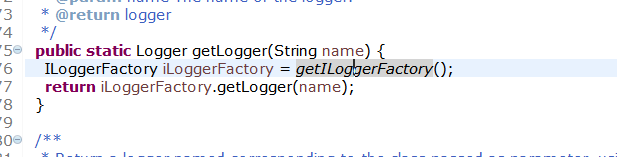
（3）使用slf4j可以方便的使用其提供的各种集体的实现的jar。（类似commons-logger）

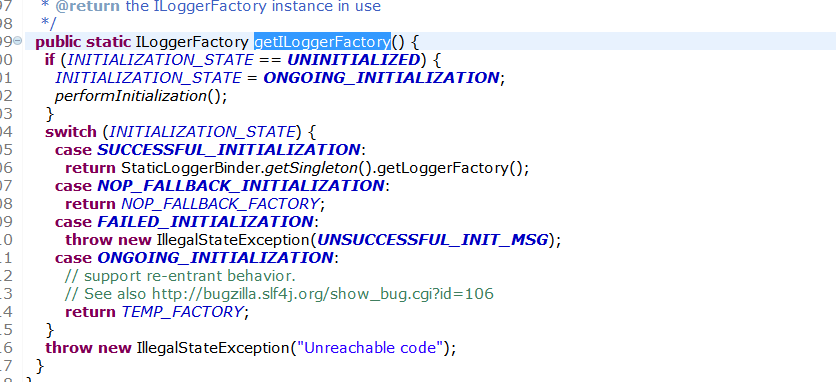
（4）从commons--logger和log4j merge非常方便，slf4j也提供了一个swing的tools来帮助大家完成这个merge。

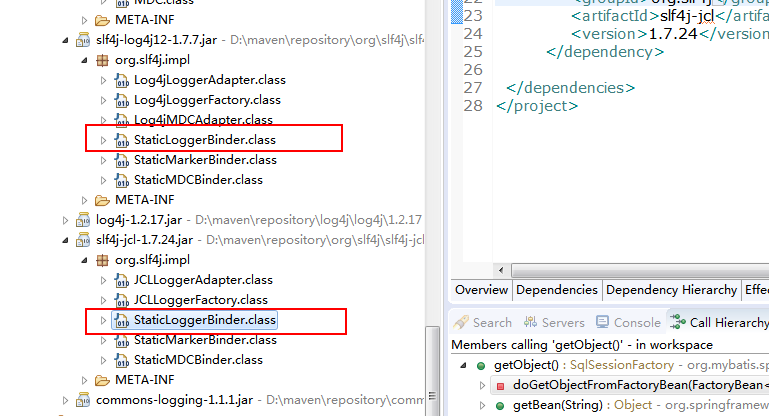
## 代码开发

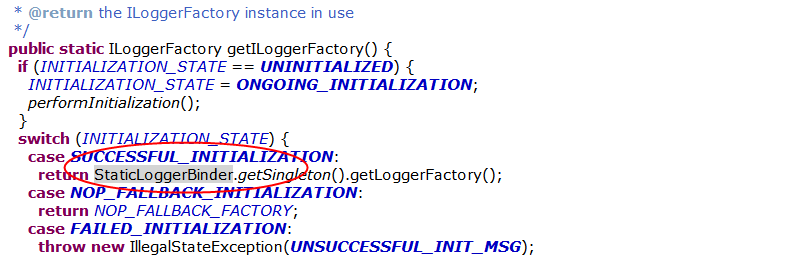












**只有一个实现类可以引进来，初始化对应的工厂类**

1 在系统开发中，统一按照slf4j的API进行开发，**在部署时，选择不同的日志系统包，即可自动转换到不同的日志系统上。**比如：选择JDK自带的日志系统，则只需要将slf4j-api-1.6.1.jar和slf4j-jdk14-1.6.1.jar放置到classpath中即可，如果中途无法忍受JDK自带的日志系统了，想换成log4j的日志系统，仅需要用slf4j-log4j12-1.6.1.jar替换slf4j- jdk14-1.6.1.jar即可（需要log4j的jar及配置文件log4j.properties文件），也可以使用slg4j提供的 simple log，slf4j-simple-1.6.1.jar替换slf4j-jdk14-1.6.1.jar。

<dependencies>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.24</version>

</dependency>

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging-api</artifactId>

<version>1.1</version>

</dependency>

<!--<dependency>-->

<!--<groupId>org.slf4j</groupId>-->

<!--<artifactId>jcl-over-slf4j</artifactId>-->

<!--<version>1.7.14</version>-->

<!--</dependency>-->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>log4j-over-slf4j</artifactId>

<version>1.7.24</version>

</dependency>

<!--<dependency>-->

<!--<groupId>org.slf4j</groupId>-->

<!--<artifactId>jul-to-slf4j</artifactId>-->

<!--<version>1.7.7</version>-->

<!--</dependency>-->

<!--<dependency>-->

<!--<groupId>ch.qos.logback</groupId>-->

<!--<artifactId>logback-classic</artifactId>-->

<!--<version>1.2.1</version>-->

<!--</dependency>-->

<!--<dependency>-->

<!--<groupId>org.slf4j</groupId>-->

<!--<artifactId>slf4j-log4j12</artifactId>-->

<!--<version>1.7.5</version>-->

<!--</dependency>-->

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-jcl</artifactId>

<version>1.7.24</version>

</dependency>

</dependencies>

**2 Slf4jTest**

package com.test;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class Slf4jTest {

// 统一按照slf4j的API进行开

Logger logger = LoggerFactory.getLogger(Slf4jTest.class);

public void testLog(){

logger.info("this is a test log");

}

public static void main(String[] args) {

Slf4jTest slf = new Slf4jTest();

slf.testLog();

}

}

**3 使用JDK自带的log输出**

在classpath中加入slf4j-api-1.6.1.jar和slf4j-jdk14-1.6.1.jar两个包，然后运行main函数，输出信息如下：

2010-2-23 11:57:28 com.test.Slf4jTest testLog

信息: this is a test log

**4 使用slg4j提供的simple log**

slf4j-simple-1.6.1.jar替换slf4j-jdk14-1.6.1.jar，选择使用slf4j提供的simple log，输出信息如下：

[main] INFO com.test.Slf4jTest - this is a test log

**5 log4j日志输出**

用slf4j-log4j12-1.6.1.jar替换slf4j-simple-1.6.1.jar（记得classpath也需要增加log4j依赖jar包），同时增加一个log4j.properties文件

我们需要修改下main方法，加载一下log4j.properties，如；

public static void main(String[] args) {

System.setProperty("log4j.configuration", "log4j.properties");

Slf4jTest slf = new Slf4jTest();

slf.testLog();

}

这样就可以以log4j的方式输出了。

**6 格式化日志**

SLF4J还提供了格式化日志的功能，如事例中的语句：

logger.info("Today is {}, Temperature set to {}. Old temperature was {}.", objs);

**7 日志系统绑定原理**

在应用中，通过LoggerFactory类的静态getLogger()获取logger。通过查看该类的代码可以看出，最终是通过StaticLoggerBinder.SINGLETON.getLoggerFactory()方法获取LoggerFactory然后，在通过该具体的LoggerFactory来获取logger的。类org.slf4j.impl.StaticLoggerBinder并不在 slf4j-api-1.6.1.jar包中，仔细查看每个与具体日志系统对应的jar包，就会发现，相应的jar包都有一个 org.slf4j.impl.StaticLoggerBinder的实现，不同的实现返回与该日志系统对应的LoggerFactory，因此就实现了所谓的静态绑定，达到只要选取不同jar包就能简单灵活配置的目的。

**8 综合测试**

public class TestLoggers {

/\*\*

\* slf4j-api -->slf4j-log4j12 -> log4j

\*/

public void testSfl4jUpLog4j(){

org.slf4j.Logger logger = LoggerFactory.getLogger(TestLoggers.class.getName());

logger.info("Slf4j print use log4j");

}

/\*\*

\* slf4j-api --> slf4j-jcl --> common-logging-api (自动向log4j)--> log4j

\*/

public void testSlf4j2CommonLogging(){

org.slf4j.Logger logger = LoggerFactory.getLogger(TestLoggers.class.getName());

logger.warn("Slf4j print to common logging");

}

/\*\*

\* log4j-over-slf4j--> slf4j-api--> logback-classic

\*/

public void testLog4j2Slf4j(){

org.apache.log4j.Logger logger = org.apache.log4j.Logger.getLogger(TestLoggers.class.getName());

logger.info("Log4j print to slf4j");

}

/\*\*

\* jcl-over-slf4j --> sfl4-api-->logback

\*/

public void testCommonLogging2Sl4j(){

Log log = LogFactory.getLog(TestLoggers.class.getName());

log.info("common log to slf4j");

}

/\*\*

\* jul(console default)-->jul-to-slf4j-->slf4j-->logback

\* 要在jre/lib/logging.properties 中.handlers添加org.slf4j.bridge.SLF4JBridgeHandler

\*/

public void testJul2Slf4j(){

java.util.logging.Logger logger = java.util.logging.Logger.getLogger(TestLoggers.class.getName());

logger.info("jul print to slf4j");

}

public static void main(String[] args) {

TestLoggers tester = new TestLoggers();

tester.testSlf4j2CommonLogging();

}

}