log4j 入门、详解

作者	日期	版本	备注
雪飘寒	2010-05-18	1.0	创建
雪飘寒	2012-06-06	1.1	第二次修改，添加章节 6.1

目录

1． Log4j 简介

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。

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**。

官方站点：http://logging.apache.org/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。

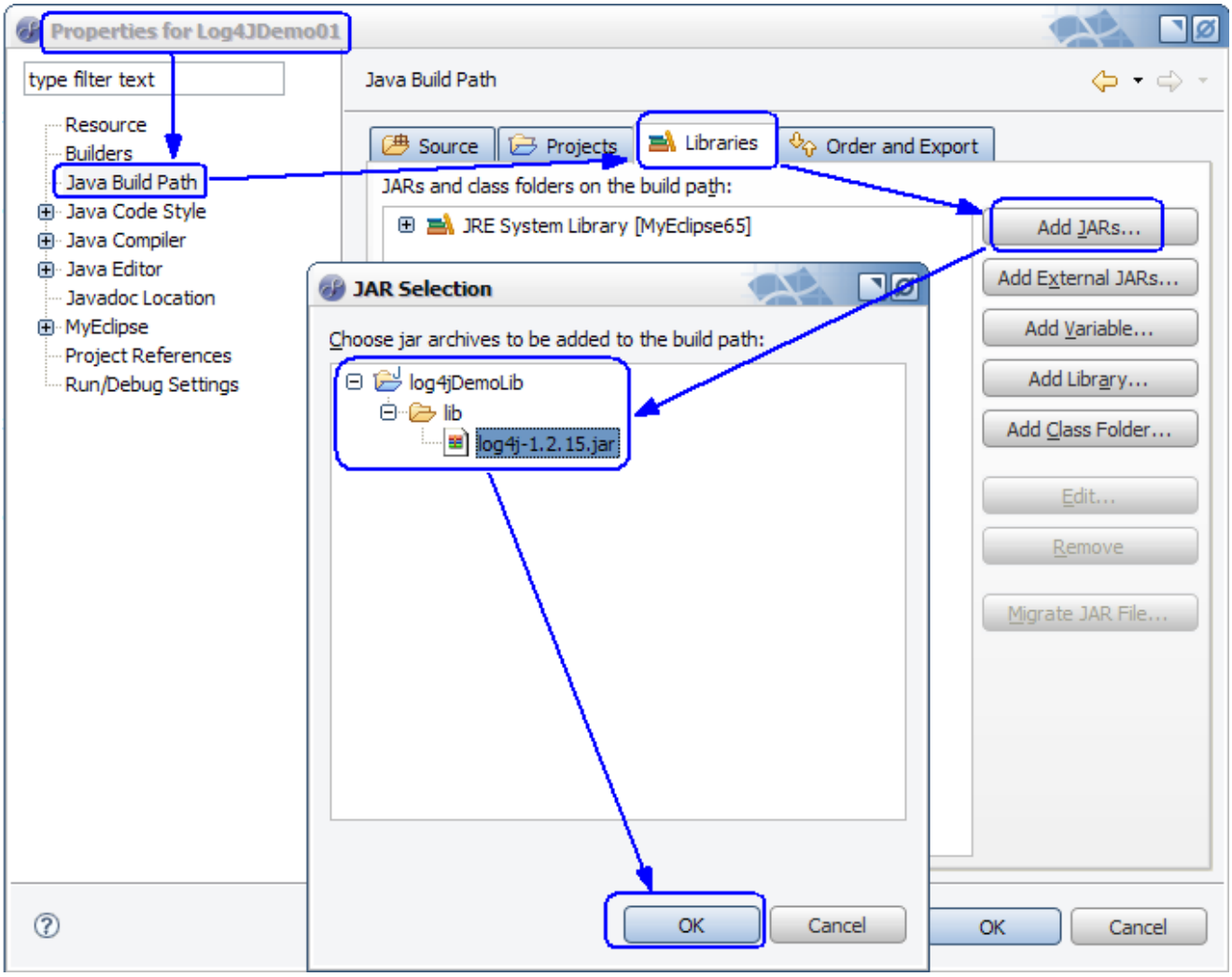
此外，通过 Log4j 其他语言接口，您可以在 C、C+ +、.Net、PL/SQL 程序中使用 Log4j，其语法和用法与在 Java 程序中一样，使得多语言分布式系统得到一个统一一致的日志组件模块。而且，通过使用各种第三方扩展，您可以很方便地将 Log4j 集成到 J2EE、JINI 甚至是 SNMP 应用中。

2． 下载与使用

2.1 下载 log4j 的 jar 文件

下载地址：http://logging.apache.org/log4j/1.2/download.html

将 jar 文件导入到工程中，N 多方法，以 eclipse 为例



2.2 编写一个测试类

```
package test;

import org.apache.log4j.Logger;

public class HelloLog4j {
    private static Logger logger = Logger.getLogger(HelloLog4j.class);
    public static void main(String[] args) {
        // 记录debug级别的信息
        logger.debug("This is debug message.");
        // 记录info级别的信息
        logger.info("This is info message.");
        // 记录error级别的信息
        logger.error("This is error message.");
    }
}
```

2.3 编写配置文件

在类路径下（src 文件夹下）创建 log4j.properties 文件，配置文件 log4j.properties 内容如下：

```
log4j.rootLogger=debug,appender1
log4j.appender.appender1=org.apache.log4j.ConsoleAppender
log4j.appender.appender1.layout=org.apache.log4j.TTCCLayout
```

2.4 输出结果

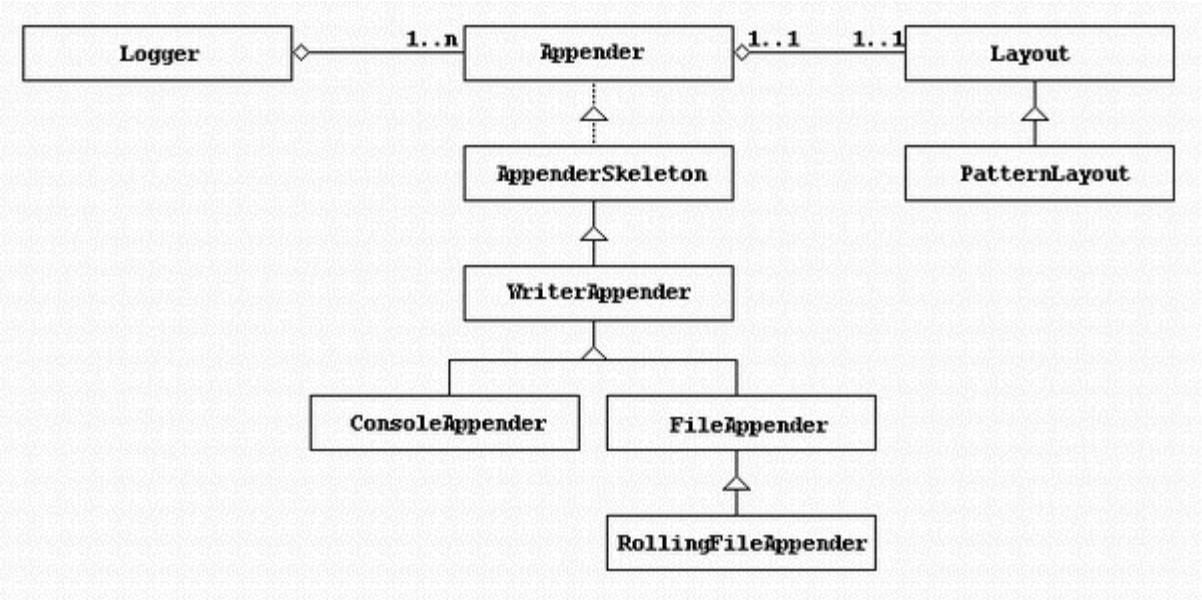
```
[main] DEBUG com.coderdream.log4j.HelloLog4j - This is debug message.
[main] INFO com.coderdream.log4j.HelloLog4j - This is info message.
[main] ERROR com.coderdream.log4j.HelloLog4j - This is error message.
```

3． Log4j 构成

通过配置文件可知，我们需要配置 3 个方面的内容：

1、根目录（级别和目的地）；
2、目的地（控制台、文件等等）；
3、输出样式。

下面我们来看看 Log4J 的类图：



Logger - 日志写出器，供程序员输出日志信息
Appender - 日志目的地，把格式化好的日志信息输出到指定的地方去
ConsoleAppender - 目的地为控制台的 Appender
FileAppender - 目的地为文件的 Appender
RollingFileAppender - 目的地为大小受限的文件的 Appender
Layout - 日志格式化器，用来把程序员的 logging request 格式化成字符串
PatternLayout - 用指定的 pattern 格式化 logging request 的 Layout

4． Log4j 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。

4.1 properties 配置文件详解

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

（1）配置根 Logger，其语法为：

```
log4j.rootLogger = level,appenderName1,appenderName2, …
```

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 B 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。优先级：**ALL < DEBUG < INFO <WARN < ERROR < FATAL < OFF**

（2）配置日志信息输出目的地 Appender，其语法为：

```
log4j.appender.appenderName = Log4j 提供的 appender 类
log4j.appender.appenderName.属性名 = 属性值
…
log4j.appender.appenderName.属性名 = 属性值
```

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）配置日志信息的格式（布局），其语法为：

```
log4j.appender.appenderName.layout = Log4j 提供的 layout 类
log4j.appender.appenderName.layout.属性 = 值
…
log4j.appender.appenderName.layout.属性 = 值
```

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）

1） HTMLLayout 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 选项

ConversionPattern=%m%n :指定怎样格式化指定的消息。

3） XMLLayout 选项

LocationInfo=true:默认值是 false,输出 java 文件和行号

Log4J 采用类似 C 语言中的 printf 函数的打印格式格式化日志信息，打印参数如下：

```
og4j.appender.A1.layout.ConversionPattern=%-4r %-5p %d{yyyy-MM-dd HH:mm:ssS} %c %m%n
```

这里需要说明的就是日志信息格式中几个符号所代表的含义：

-X 号：X 信息输出时左对齐；

%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，配置文件中的名字，通常就是所在类的全名（若使用 rootLogger）

%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

%M：输出日志信息所属的方法

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

4.2 XML 配置文件详解

xml 格式的 log4j 配置文件需要使用 org.apache.log4j.html.DOMConfigurator.configure()方法来读入。对 xml 文件的语法定义可以在 log4j 的发布包中找到：org/apache/log4j/xml/log4j.dtd。

log4j 的 xml 配置文件的树状结构如下所示，注意下图只显示了常用的部分。

```
xml declaration and dtd
  |
log4j:configuration
  |
  +-- appender (name, class)
  |    |
  |    +-- param (name, value)
  |    +-- layout (class)
  |         |
  |         +-- param (name, value)
  +-- logger (name, additivity)
  |    |
  |    +-- level (class, value)
  |    |    |
  |    |    +-- param (name, value)
  |    +-- appender-ref (ref)
  +-- root
       |
       +-- param (name, class)
       +-- level
       |    |
       |    +-- param (name, value)
       +-- appender-ref (ref)
```

xml 配置文件的头部包括两个部分：xml 声明和 dtd 声明，不多介绍。头部的格式如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
```

log4j:configuration (root element)

xmlns:log4j [#FIXED attribute]

定义 log4j 的名字空间，取定值"http://jakarta.apache.org/log4j/"

appender [* child] : 一个 appender 子元素定义一个日志输出目的地

logger [* child] : 一个 logger 子元素定义一个日志写出器

root [? child] : root 子元素定义了 root logger

appender 元素定义一个日志输出目的地。

name [#REQUIRED attribute] : 定义 appender 的名字，以便被后文引用

class [#REQUIRED attribute] : 定义 appender 对象所属的类的全名

param [* child] : 创建 appender 对象时传递给类构造方法的参数

layout [? child] : 该 appender 使用的 layout 对象

layout 元素定义与某一个 appender 相联系的日志格式化器。
class [#REQUIRED attribute] : 定义 layout 对象所属的类的全名
param [* child] : 创建 layout 对象时传递给类构造方法的参数

logger 元素定义一个日志输出器。
name [#REQUIRED attribute] : 定义 logger 的名字，以便被后文引用
additivity [#ENUM attribute] : 取值为"true"（默认）或者"false"，是否继承父 logger 的属性
level [? child] : 定义该 logger 的日志级别
appender-ref [* child] : 定义该 logger 的输出目的地

root 元素定义根日志输出器 root logger。
param [* child] : 创建 root logger 对象时传递给类构造方法的参数
level [? child] : 定义 root logger 的日志级别
appender-ref [* child] : 定义 root logger 的输出目的地

level 元素定义 logger 对象的日志级别。
class [#IMPLIED attribute] : 定义 level 对象所属的类，默认情况下是"org.apache.log4j.Level 类
value [#REQUIRED attribute] : 为 level 对象赋值。可能的取值从小到大依次为"all"、"debug"、"info"、"warn"、"error"、"fatal"和"off"。当值为"off"时表示没有任何日志信息被输出
param [* child] : 创建 level 对象时传递给类构造方法的参数

appender-ref 元素引用一个 appender 元素的名字，为 logger 对象增加一个 appender。
ref [#REQUIRED attribute] : 一个 appender 元素的名字的引用
appender-ref 元素没有子元素

param 元素在创建对象时为类的构造方法提供参数。
它可以成为 appender、layout、filter、errorHandler、level、categoryFactory 和 root 等元素的子元素。
name and value [#REQUIRED attributes] : 提供参数的一组名值对
param 元素没有子元素

在 xml 文件中配置 appender 和 layout
创建不同的 Appender 对象或者不同的 Layout 对象要调用不同的构造方法。可以使用 param 子元素来设定不同的参数值。

创建 ConsoleAppender 对象
ConsoleAppender 的构造方法不接受其它的参数。

```
<appender name="console.log" class="org.apache.log4j.ConsoleAppender">
  <layout ... >
    ... ...
  </layout>
</appender>
```

创建 FileAppender 对象
可以为 FileAppender 类的构造方法传递两个参数：File 表示日志文件名；Append 表示如文件已存在，是否把日志追加到文件尾部，可能取值为"true"和"false"（默认）。

```
        <appender name="file.log" class="org.apache.log4j.FileAppender">
          <param name="File" value="/tmp/log.txt" />
          <param name="Append" value="false" />
          <layout ... >
            ... ...
          </layout>
        </appender>
```

创建 RollingFileAppender 对象

除了 File 和 Append 以外，还可以为 RollingFileAppender 类的构造方法传递两个参数：MaxBackupIndex 备份日志文件的个数（默认是 1 个）；MaxFileSize 表示日志文件允许的最大字节数（默认是 10M）。 :.

```
        <appender
             name="rollingFile.log" class="org.apache.log4j.RollingFileAppender">
          <param name="File" value="/tmp/rollingLog.txt" />
          <param name="Append" value="false" />
          <param name="MaxBackupIndex" value="2" />
          <param name="MaxFileSize" value="1024" />
          <layout ... >
            ... ...
          </layout>
        </appender>
```

创建 PatternLayout 对象

可以为 PatternLayout 类的构造方法传递参数 ConversionPattern。 :.

```
        <layout class="org.apache.log4j.PatternLayout>
          <param name="Conversion" value="%d [%t] %p - %m%n" />
        </layout>
```

4.3 properties 比较详细的例子

```
log4j.rootLogger=INFO,consoleAppender,logfile,MAIL
log4j.addivity.org.apache=true

#ConsoleAppender，控制台输出
#FileAppender，文件日志输出
#SMTPAppender，发邮件输出日志
#SocketAppender，Socket 日志
#NTEventLogAppender，Window NT 日志
#SyslogAppender，
#JMSAppender，
#AsyncAppender，
#NullAppender

#文件输出：RollingFileAppender
#log4j.rootLogger = INFO,logfile
log4j.appender.logfile = org.apache.log4j.RollingFileAppender
log4j.appender.logfile.Threshold = INFO
# 输出以上的 INFO 信息
log4j.appender.logfile.File = INFO_log.html
```

```
#保存 log 文件路径
log4j.appender.logfile.Append = true
# 默认为 true，添加到末尾，false 在每次启动时进行覆盖
log4j.appender.logfile.MaxFileSize = 1MB
# 一个 log 文件的大小，超过这个大小就又会生成 1 个日志 # KB ，MB，GB
log4j.appender.logfile.MaxBackupIndex = 3
# 最多保存 3 个文件备份
log4j.appender.logfile.layout = org.apache.log4j.HTMLLayout
# 输出文件的格式
log4j.appender.logfile.layout.LocationInfo = true
#是否显示类名和行数
log4j.appender.logfile.layout.Title
=title:\u63d0\u9192\u60a8\uff1a\u7cfb\u7edf\u53d1\u751f\u4e86\u4e25\u91cd\u9519\u8b
ef
#html 页面的 < title >
############################### SampleLayout ####################################
# log4j.appender.logfile.layout = org.apache.log4j.SampleLayout
############################### PatternLayout ###################################
# log4j.appender.logfile.layout = org.apache.log4j.PatternLayout
# log4j.appender.logfile.layout.ConversionPattern =% d % p [ % c] -   % m % n % d
############################### XMLLayout #######################################
# log4j.appender.logfile.layout = org.apache.log4j.XMLLayout
# log4j.appender.logfile.layout.LocationInfo = true #是否显示类名和行数
############################### TTCCLayout ######################################
# log4j.appender.logfile.layout = org.apache.log4j.TTCCLayout
# log4j.appender.logfile.layout.DateFormat = ISO8601
#NULL, RELATIVE, ABSOLUTE, DATE or ISO8601.
# log4j.appender.logfile.layout.TimeZoneID = GMT - 8 : 00
# log4j.appender.logfile.layout.CategoryPrefixing = false ##默认为 true 打印类别名
# log4j.appender.logfile.layout.ContextPrinting = false ##默认为 true 打印上下文信息
# log4j.appender.logfile.layout.ThreadPrinting = false ##默认为 true 打印线程名
# 打印信息如下：
#2007 - 09 - 13   14 : 45 : 39 , 765 [http - 8080 - 1 ] ERROR com.poxool.test.test -
error 成功关闭链接
#################################################################################
#每天文件的输出：DailyRollingFileAppender
#log4j.rootLogger = INFO,errorlogfile
log4j.appender.errorlogfile = org.apache.log4j.DailyRollingFileAppender
log4j.appender.errorlogfile.Threshold = ERROR
log4j.appender.errorlogfile.File = ../logs/ERROR_log
log4j.appender.errorlogfile.Append = true
#默认为 true，添加到末尾，false 在每次启动时进行覆盖
log4j.appender.errorlogfile.ImmediateFlush = true
#直接输出，不进行缓存
# ' . ' yyyy - MM: 每个月更新一个 log 日志
# ' . ' yyyy - ww: 每个星期更新一个 log 日志
# ' . ' yyyy - MM - dd: 每天更新一个 log 日志
# ' . ' yyyy - MM - dd - a: 每天的午夜和正午更新一个 log 日志
# ' . ' yyyy - MM - dd - HH: 每小时更新一个 log 日志
```

```
# ' . ' yyyy - MM - dd - HH - mm: 每分钟更新一个 log 日志
log4j.appender.errorlogfile.DatePattern = ' . ' yyyy - MM - dd ' .log '
#文件名称的格式
log4j.appender.errorlogfile.layout = org.apache.log4j.PatternLayout
log4j.appender.errorlogfile.layout.ConversionPattern =%d %p [ %c] -   %m %n %d

#控制台输出:
#log4j.rootLogger = INFO,consoleAppender
log4j.appender.consoleAppender = org.apache.log4j.ConsoleAppender
log4j.appender.consoleAppender.Threshold = ERROR
log4j.appender.consoleAppender.layout = org.apache.log4j.PatternLayout
log4j.appender.consoleAppender.layout.ConversionPattern =%d %-5p %m %n
log4j.appender.consoleAppender.ImmediateFlush = true

# 直接输出，不进行缓存
log4j.appender.consoleAppender.Target = System.err
# 默认是 System.out 方式输出

#发送邮件：SMTPAppender
#log4j.rootLogger = INFO,MAIL
log4j.appender.MAIL = org.apache.log4j.net.SMTPAppender
log4j.appender.MAIL.Threshold = INFO
log4j.appender.MAIL.BufferSize = 10
log4j.appender.MAIL.From = yourmail@gmail.com
log4j.appender.MAIL.SMTPHost = smtp.gmail.com
log4j.appender.MAIL.Subject = Log4J Message
log4j.appender.MAIL.To = yourmail@gmail.com
log4j.appender.MAIL.layout = org.apache.log4j.PatternLayout
log4j.appender.MAIL.layout.ConversionPattern =%d - %c -%-4r [%t] %-5p %c %x - %m %n

#数据库：JDBCAppender
log4j.appender.DATABASE = org.apache.log4j.jdbc.JDBCAppender
log4j.appender.DATABASE.URL = jdbc:oracle:thin:@ 210.51 . 173.94 : 1521 :YDB
log4j.appender.DATABASE.driver = oracle.jdbc.driver.OracleDriver
log4j.appender.DATABASE.user = ydbuser
log4j.appender.DATABASE.password = ydbuser
log4j.appender.DATABASE.sql = INSERT INTO A1 (TITLE3) VALUES ( ' %d - %c %-5p %c %x - %m%n
' )
log4j.appender.DATABASE.layout = org.apache.log4j.PatternLayout
log4j.appender.DATABASE.layout.ConversionPattern =% d -   % c -%- 4r [ % t] %- 5p % c %
x -   % m % n
#数据库的链接会有问题,可以重写 org.apache.log4j.jdbc.JDBCAppender 的 getConnection() 使用数
据库链接池去得链接，可以避免 insert 一条就链接一次数据库
```

4.4 在代码中使用 Log4j

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 () )
```

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.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 ) ;
```

4.5 注意事项

Logger 的命名规则

Logger 由一个 String 类的名字识别，logger 的名字是大小写敏感的，且名字之间具有继承的关系，子名有父名作为前缀，用点号.分隔。如：x.y 是 x.y.z 的父亲。

5. Properties 文件实例说明

文件内容如下：

```
1 log4j.rootCategory=INFO, stdout , R
2
3 log4j.appender.stdout=org.apache.log4j.ConsoleAppender
4  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
5  log4j.appender.stdout.layout.ConversionPattern=[QC] %p [%t] %C.%M(%L) | %m%n
6
7  log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
8  log4j.appender.R.File=D:\\Tomcat 5.5\\logs\\qc.log
9  log4j.appender.R.layout=org.apache.log4j.PatternLayout
10 log4j.appender.R.layout.ConversionPattern=%d-[TS] %p %t %c - %m%n
11
12 log4j.logger.com.neusoft=DEBUG
13 log4j.logger.com.opensymphony.oscache=ERROR
14 log4j.logger.net.sf.navigator=ERROR
15 log4j.logger.org.apache.commons=ERROR
16 log4j.logger.org.apache.struts=WARN
17 log4j.logger.org.displaytag=ERROR
18 log4j.logger.org.springframework=DEBUG
19 log4j.logger.com.ibatis.db=WARN
20 log4j.logger.org.apache.velocity=FATAL
21
22 log4j.logger.com.canoo.webtest=WARN
23
24 log4j.logger.org.hibernate.ps.PreparedStatementCache=WARN
25 log4j.logger.org.hibernate=DEBUG
26 log4j.logger.org.logicalcobwebs=WARN
```

1 log4j.rootLogger=INFO, stdout , R

此句为将等级为 INFO 的日志信息输出到 stdout 和 R 这两个目的地，stdout 和 R 的定义在下面的代码，可以任意起名。等级可分为 OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL，如果配置 OFF 则不打出任何信息，如果配置为 INFO 这样只显示 INFO, WARN, ERROR 的 log 信息，而 DEBUG 信息不会被显示，具体讲解可参照第三部分定义配置文件中的 logger。

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

此句为定义名为 stdout 的输出端是哪种类型，可以是

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）

具体讲解可参照第三部分定义配置文件中的 Appender。

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

此句为定义名为 stdout 的输出端的 layout 是哪种类型，可以是

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）

具体讲解可参照第三部分定义配置文件中的 Layout。

5 log4j.appender.stdout.layout.ConversionPattern= [QC] %p [%t] %C.%M(%L) | %m%n

如果使用 pattern 布局就要指定的打印信息的具体格式 ConversionPattern，打印参数如下：

%m 输出代码中指定的消息

%p 输出优先级，即 DEBUG，INFO，WARN，ERROR，FATAL

%r 输出自应用启动到输出该 log 信息耗费的毫秒数

%c 输出所属的类目，通常就是所在类的全名

%t 输出产生该日志事件的线程名

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

%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{yyyy MMM dd HH:mm:ss,SSS}，输出类似：2002 年 10 月 18 日 22：10：28，921

%l 输出日志事件的发生位置，包括类目名、发生的线程，以及在代码中的行数。

[QC]是 log 信息的开头，可以为任意字符，一般为项目简称。

输出的信息

[TS] DEBUG [main] AbstractBeanFactory.getBean(189) | Returning cached instance of singleton bean 'MyAutoProxy'

具体讲解可参照第三部分定义配置文件中的格式化日志信息。

7 log4j.appender.R=org.apache.log4j.DailyRollingFileAppender

此句与第 3 行一样。定义名为 R 的输出端的类型为每天产生一个日志文件。

8 log4j.appender.R.File=D:\\Tomcat 5.5\\logs\\qc.log

此句为定义名为 R 的输出端的文件名为 D:\\Tomcat 5.5\\logs\\qc.log 可以自行修改。

9 log4j.appender.R.layout=org.apache.log4j.PatternLayout

与第 4 行相同。

10 log4j.appender.R.layout.ConversionPattern=%d-[TS] %p %t %c - %m%n
与第 5 行相同。

12 log4j.logger.com.neusoft=DEBUG
指定 com.neusoft 包下的所有类的等级为 DEBUG。
可以把 com.neusoft 改为自己项目所用的包名。

13 log4j.logger.com.opensymphony.oscache=ERROR
14 log4j.logger.net.sf.navigator=ERROR
这两句是把这两个包下出现的错误的等级设为 ERROR，如果项目中没有配置 EHCache，则不需要这两句。

15 log4j.logger.org.apache.commons=ERROR
16 log4j.logger.org.apache.struts=WARN
这两句是 struts 的包。

17 log4j.logger.org.displaytag=ERROR
这句是 displaytag 的包。（QC 问题列表页面所用）

18 log4j.logger.org.springframework=DEBUG
此句为 Spring 的包。

24 log4j.logger.org.hibernate.ps.PreparedStatementCache=WARN
25 log4j.logger.org.hibernate=DEBUG
此两句是 hibernate 的包。
以上这些包的设置可根据项目的实际情况而自行定制。

6． 注意事项

6.1 为什么使用 logger 之前要判断日志输入级别？

经常看到代码中有这样的写法：

```
if(logger.isDebugEnabled()){
    logger.debug(AssessmentParamsUtils.getParams().toString());
}
```

有人就有疑问了，这个和下面直接 debug 的写法有什么区别？

```
logger.debug(AssessmentParamsUtils.getParams().toString());
```

我们来看一下源码：

debug 方法的源码（log4j）

```
public void debug(Object message) {
    if (repository.isDisabled(Level.DEBUG_INT))
        return;
    if (Level.DEBUG.isGreaterOrEqual(this.getEffectiveLevel())) {
        forcedLog(FQCN, Level.DEBUG, message, null);
    }
}
```

isDebugEnabled 方法的源码（log4j）

```
public boolean isDebugEnabled() {
    if (repository.isDisabled(Level.DEBUG_INT))
        return false;
    return Level.DEBUG.isGreaterOrEqual(this.getEffectiveLevel());
}
```

其实判断是一样的，都判断了 debug 级别是否开启，这样看来，判断似乎没有区别，那为什么还要加这个判断呢？

答：为了性能。

其实原因不在 debug 上，而在日志输入的东西上，如果只输入简单的字符串，或者小对象，异常等等，那么这两个没有区别，建议就不用判断了，区别不大。

但是，一旦输出前执行了比较费时的方法，那么就很影响性能了，如上面这段中的

AssessmentParamsUtils.getParams().toString()

如果这句话执行需要 1 分钟，那么加判断的就一下子过去了，不执行这句话，而没有加判断的需要先执行这句话，1 分钟后执行完毕，才去执行 debug，一判断不输出，跳过，虽然结果一样，但是多执行了 1 分钟，严重影响了性能。现实中不会到 1 分钟的，这里的 1 分钟只是示例，但是高并发时，影响还是挺大的。

同理其他方法：

```
logger.isDebugEnabled();
logger.isErrorEnabled();
logger.isFatalEnabled();
logger.isInfoEnabled();
logger.isTraceEnabled();
logger.isWarnEnabled();
```

为了更健壮的代码，像这样的小细节以后也要多注意了。