# Log4j

## Log4j概述

###### 1　什么是Log4j

Log4j 是Apache为Java提供的日志管理工具。他与System.out.println()的作用相似，用来跟踪、调试、维护程序。

为了你快速理解Log4j的作用，我们用下面的代码说明Log4j的作用。

我们为了调试程序，总是需要在程序使用System.out.println()来输出一些信息。例如下面的代码：

|  |
| --- |
| // 获取UserService  UserService userService = UserServiceFactory.getUserService();  **if**(userService == **null**) {  System.*out*.println("错误：userService没有获取到！");  **return**;  }    // 获取要查询的User对象id  String uid = request.getParameter("uid");  **int** id;  **try** {  id = Integer.*parseInt*(uid);  } **catch**(NumberFormatException e) {  System.*out*.println("错误：无法将uid转换成int类型！");  **throw** e;  }  // 查询User  User user = userService.load(id);    System.*out*.println("信息：已查询到User对象");    user.setPassword("zhangSan");  System.*out*.println("信息：修改user的密码");    userService.save(user);  System.*out*.println("信息：保存user信息"); |

上面例子中所有的输出语句都是用来调试程序的，相信你的代码中也经常会出现这样的调试代码吧？这些调试代码必须在项目完成后删除。你想一想，一行一行的删除会不会忘记几行没有删除呢？

使用Log4j来打印这些测试代码，然后通过配置文件就能统一管理这些日志信息了！我们可以在配置文件中关闭所有日志，也可以在配置文件中打开所有日志，也可以打开某个级别的日志。甚至还可以管理日志出现的格式（是否加上日期和时间），以及日志输出的目标（是否为控制台，是否为文件）。

###### 2　Log4j核心概念

Log4j中有三大组件：日志器（Logger）、日志输出目标（Appender）、格式化器（Layout）

* Logger：用来输出日志消息的类，它可以输出不同级别的消息，例如错误消息、警告消息等；
* Appender；通常我们希望日志输出到文件中，以及控制台，也可能希望日志输出数据库，该类就表示一个输出的目标；
* Layout：对输出的消息进行格式化，例如在消息中添加日期，以及级别等。

###### 3　Hello Log4j

|  |
| --- |
| **public** **class** Demo1 {  @Test  **public** **void** fun1() {  Logger logger = Logger.*getLogger*(Demo1.**class**);  logger.addAppender(**new** ConsoleAppender(**new** SimpleLayout()));  logger.setLevel(Level.*WARN*);    logger.debug("调试");  logger.info("信息");  logger.warn("警告");  logger.error("错误");  logger.fatal("大错");  }  } |
| @Test  **public** **void** fun2() {  Logger logger = Logger.*getLogger*(Demo1.**class**);    logger.debug("调试");  logger.info("信息");  logger.warn("警告");  logger.error("错误");  logger.fatal("大错");  } |
| #配置根日志器级别为DEBUG，目的地为A1  log4j.rootLogger=DEBUG,A1  #配置A1目的地  log4j.appender.A1=org.apache.log4j.ConsoleAppender  #配置A1目的地的输出样式  log4j.appender.A1.layout=org.apache.log4j.SimpleLayout |

## Logger

###### 1　日志级别

在Log4j中日志消息分为五个级别，级别由高到低排列如下：

* FATAL：重大错误，例如系统崩溃；
* ERROR：错误，例如某模块瘫痪；
* WARN：警告，程序的隐患，如果不处理，将来可能就是错误；
* INFO：信息，可以用来查看程序执行的流程；
* DEBUG：调试，用来调试程序的bug。

我们可以使用日志器输出这五种不同的日志，然后通过设置日志器的级别来控制输出的结果。来看下面的代码：

|  |
| --- |
| @Test  **public** **void** fun() {  *log*.setLevel(Level.*ERROR*);  *log*.debug("hello log4j!");  *log*.info("hello log4j!");  *log*.warn("hello log4j!");  *log*.error("hello log4j!");  *log*.fatal("hello log4j!");  } |

上面代码中，只有error()和fatal() 两个方法的输出会完成，其他级别的输出都不会完成。因为设置日志器的级别为Level.ERROR后，只有高于ERROR级别的日志才能输出。

###### 2　日志器名称

　　在创建日志器时，需要给日志器指定一个名称：Logger log = Logger.getLogger(“hello”);

日志器的名称不只是一个名称而已，日志器的名称说明了日志器之间的父子关系。子日志器会继承父日志器的Appender和Level。

日志器的父子关系是通过日志器的名称来决定的，例如名称为cn.itcast的日志器是cn.itcast.logger的日志器的爸爸。cn.itcast.logger会继承cn.itcast的Appender以及Level。

|  |
| --- |
| @Test  **public** **void** fun() {  Logger log = Logger.*getLogger*("cn.itcast");  Appender appender = **new** ConsoleAppender(**new** SimpleLayout());  log.addAppender(appender);  log.setLevel(Level.*ERROR*);    Logger log1 = Logger.*getLogger*("cn.itcast.logger");  log1.debug("看不见！");  } |

通常我们创建日志器都是使用当前类的名称来创建：

Logger log = Logger.getLogger(Demo.class.getName());

这样我们可以通过配置父日志器来改变日志器的Appender和Level。

## 配置文件

###### 1　log4j配置概述

log4j有两种配置文件，log4j.properties和log4j.xml。因为log4j是比较简单的组件，所以建议使用log4j.properties。

如果你把log4j放到项目的src下（即放到classes下），并且命名为log4j.properties，那么log4j会自动加载这个配置文件，这也是我们最为常用的方式。

###### 2　根日志器

在配置文件中，需要配置名为log4j.rootLogger的日志器，它就是根日志器。其他的日志器都是根日志器的子日志器，会继承根日志器的目的地，以及级别等。

log4j.properties

|  |
| --- |
| log4j.rootLogger=DEBUG,A1  log4j.appender.A1=org.apache.log4j.ConsoleAppender  log4j.appender.A1.layout=org.apache.log4j.SimpleLayout |

* log4j.rootLogger表示根日志器，DEBUG是指定根日志器的输出级别，只要大于等于DEBUG级别的日志就可以输出。A1是说明目的地（Appender）的名称；
* log4j.appender.A1表示目的地，org.apache.log4j.ConsoleAppender表示目的地为控制台目的地，即输出结果在控制台上显示；
* log4j.appender.A1.layout表示A1这个目的地的格式化器，org.apache.log4j.SimpleLayout表示简化格式化器。

|  |
| --- |
| **public** **void** fun1() {  Logger log = Logger.*getLogger*(Demo1.**class**);  log.debug("Hello");  } |

###### 3　其他日志器

当然，也可以去配置其他的日志器，语法如下：

|  |
| --- |
| log4j.rootLogger=DEBUG,A1  log4j.logger.cn.itcast=WARN,A1  log4j.appender.A1=org.apache.log4j.ConsoleAppender  log4j.appender.A1.layout=org.apache.log4j.SimpleLayout |

　　我们不只配置了一个根日志器，还配置了一个名称为cn.itcast的日志器。继承关系你是懂得对吧。所有日志器如果名称是以cn.itcast前缀的，那么都会继承上面的配置，如果不是，那么继承的就是root日志器了。

## Appender

###### 1　Appender概述

Appender是用来指定输出目标的类，你可以叫它目的地。上面我们使用的ConsoleAppender就是用来向控制台输出的目的地。常用的目的地有：

* ConsoleAppender：向控制台输出日志；
* FileAppender：向文件输出日志
* DailyRollingFileAppender：向文件输出日志，每天一个日志文件；
* RollingFileAppender：向文件输出日志，当文件大小达到指定大小后，生成新文件；

###### 2　ConsoleAppender

ConsoleAppender的目标是控制台！这也不多做解释

|  |
| --- |
| log4j.rootLogger=DEBUG,A1  log4j.appender.A1=org.apache.log4j.ConsoleAppender  log4j.appender.A1.layout=org.apache.log4j.SimpleLayout |

###### 3　FileAppender

FileAppender的目标是磁盘文件

|  |
| --- |
| log4j.rootLogger=DEBUG,A1  log4j.appender.A1=org.apache.log4j.FileAppender  log4j.appender.A1.File=log.txt  log4j.appender.A1.Append=true  log4j.appender.A1.layout=org.apache.log4j.SimpleLayout |

###### 4　RollingFileAppender

RollingFileAppender是FileAppender的子类，它的作用是当日志文件大小超出文件大小大限时，会把日志文件转换成备份文件，然后再生成一个新的日志文件。

例如日志文件名为log.txt，设置文件大小上限为1KB，当log.txt文件的大小超出了1KB后，把log.txt的名称转换成log.txt.1，然后再生成一个log.txt，新的日志会写入到新的log.txt文件中。当log.txt的大小再次达到1KB时，把log.txt.1名称修改成log.txt.2，把log.txt修改成log.txt.1，然后再生成一个新的log.txt文件。

还可以设置文件的个数，当设置备份文件的个数为3时，表示最多可以有3个文件。当文件达到3后，再次达到1KB时，那么会删除最后一个文件。例如当前已经存在log.txt、log.txt.1、log.txt.2，这时如果log.txt又达到了1KB时，那么删除log.txt.2，然后把log.txt.1修改成log.txt.2，再把log.txt修改成log.txt.1，然后再创建log.txt文件。

|  |
| --- |
| log4j.rootLogger=DEBUG,A1  log4j.appender.A1=org.apache.log4j.RollingFileAppender  log4j.appender.A1FILE.File=log.txt  log4j.appender.A1.MaxFileSize=1KB  log4j.appender.A1.MaxBackupIndex=3  log4j.appender.A1.layout=org.apache.log4j.SimpleLayout |

###### 4　DailyRollingFileAppender

DailyRollingFileAppender会根据设定的时间频率生成备份文件。

* 当时间频率为yyyy-MM：按月生成备份文件；
* 当时间频率为yyyy-ww：按周生成备份文件；
* 当时间频率为yyyy-MM-dd：按天生成备份文件；
* 当时间频率为yyyy-MM-dd-a：每天生成两次备份；
* 当时间频率为yyyy-MM-dd-HH：按小时生成备份文件；
* 当时间频率为yyyy-MM-dd-HH-mm：按分钟生成备份文件。

|  |
| --- |
| log4j.rootLogger=DEBUG,A1  log4j.appender.A1=org.apache.log4j.DailyRollingFileAppender  log4j.appender.A1.File=log.txt  log4j.appender.A1.DatePattern='.'yyyy-MM-dd  log4j.appender.A1.layout=org.apache.log4j.SimpleLayout |

## Layout

###### 1　Layout种类

　　我们需要给目的地指定格式化器，这样目的地就可以按照格式化器指定的格式来输出日志信息了。我们刚刚一直都在使用SimpleLayout格式化器，而我们常用的格式化器是PatternLayout！

###### 2　PatternLayout

可以为PatternLayout指定一个模板，目的地会按模板来输出日志信息。例如模板为：hello: %m，其中%m是一个模式字符，下面介绍一下模式字符：

* %m：信息本身；
* %c：日志器的名称；
* %d：日期，还可以指定日期的格式，例如：%d{yyyy-MM-dd HH:mm:ss}；
* %p：日志级别；
* %n：换行；
* %t：当前线程；
* %l：输出日志的Java类相关信息。

在web项目中的配置

|  |
| --- |
| log4j.rootLogger=INFO,c1, f1  log4j.appender.c1=org.apache.log4j.ConsoleAppender  log4j.appender.c1.layout=org.apache.log4j.PatternLayout  log4j.appender.c1.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} [%p] %m [%t] %c [%l]%n  log4j.appender.f1.File=${catalina.home}/logs/myproject.txt  log4j.appender.f1=org.apache.log4j.DailyRollingFileAppender  log4j.appender.f1.layout=org.apache.log4j.PatternLayout  log4j.appender.f1.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} [%p] %m [%t] %c [%l]%n |