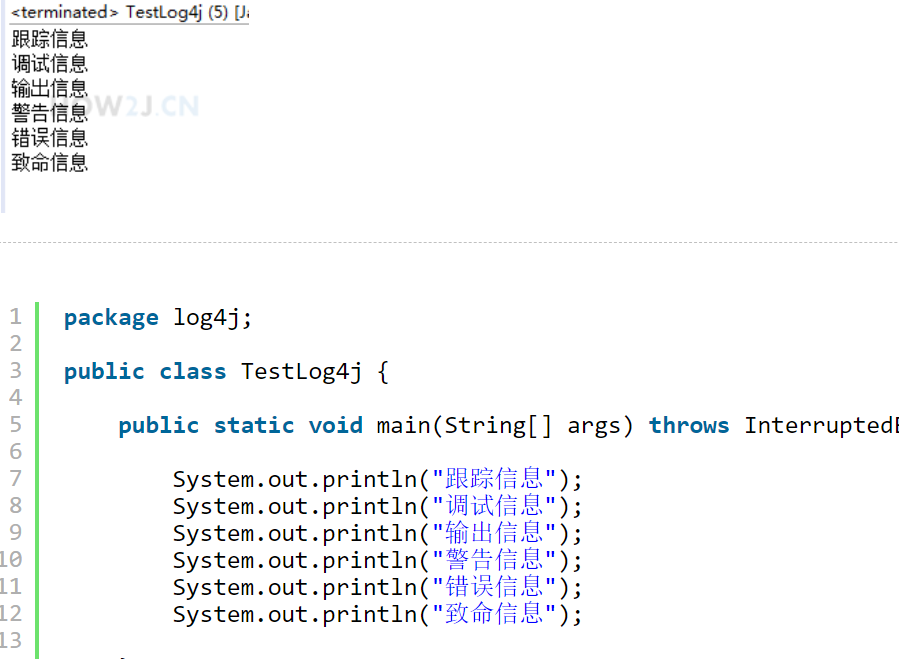
**一、入门**

1. **System.out.println**

通常，在写代码的过程中，免不了要输出各种调试信息。在没有使用任何日志工具之前，都会使用System.out.println来做到。这么做直观有效，但存在一系列缺点：

1. 不知道这句话是在哪个类，哪个线程里出来的。
2. 不知道什么时候前后两句输出间隔了多少时间。
3. 无法关闭调试信息，一旦System.out.println多了之后，到处都是，增加定位自己需要信息的难度等。

如图：



1. **导入jar包**

在使用之前导入jar包。

1. **使用Log4j**

在代码中使用Log4j的步骤：

1. 得到记录器（负责控制日志信息）
2. 读取配置文件
3. 插入记录信息

**public** **class** Log4jTest {

// 1.获取日志记录器Logger，名字为本类类名

**static** Logger *logger* =

Logger.*getLogger*(Log4jTest.**class**);

**public** **static** **void** main(String[] args)

**throws** InterruptedException {

// 2.配置Log4j环境

//自动快速地使用缺省Log4j环境

BasicConfigurator.*configure*();

//改变Log4j日志输出级别

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

// 3.插入记录信息（格式化日志信息）

*logger*.trace("跟踪信息");

*logger*.debug("调试信息");

*logger*.info("输出信息");

Thread.*sleep*(1000);

*logger*.warn("警告信息");

*logger*.error("错误信息");

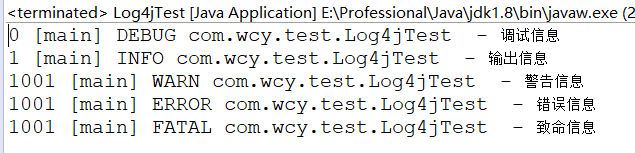
*logger*.fatal("致命信息");

}

}

为了应对使用System.out.println带来的缺点，使用Log4j来进行日志输出。采用如下代码，执行雷同的输出。可以看到输出结果有几个改观：

1. 知道是com.wcy.test.Log4jTest这个类里的日志；
2. 是在[main]线程里的日志；
3. 日志级别可观察，一共有6个级别**TRACE** **DEBUG** **INFO** **WARN** **ERROR** **FATAL**
4. 日志输出级别范围可控制，如代码所示，只输出高于DEBUG级别的，那么TRACE级别的日志自动不输出；
5. 每句日志消耗的毫秒数（最前面的数字），可观察，这样就可以进行性能计算。



1. **代码讲解**
2. 基于类的名称获取日志对象

static Logger logger = Logger.getLogger(TestLog4j.class);

1. 进行默认配置

BasicConfigurator.configure();

1. 设置日志输出级别

logger.setLevel(Level.DEBUG);

1. 进行不同级别的日志输出

logger.trace("跟踪信息");

logger.debug("调试信息");

logger.info("输出信息");

Thread.sleep(1000);

logger.warn("警告信息");

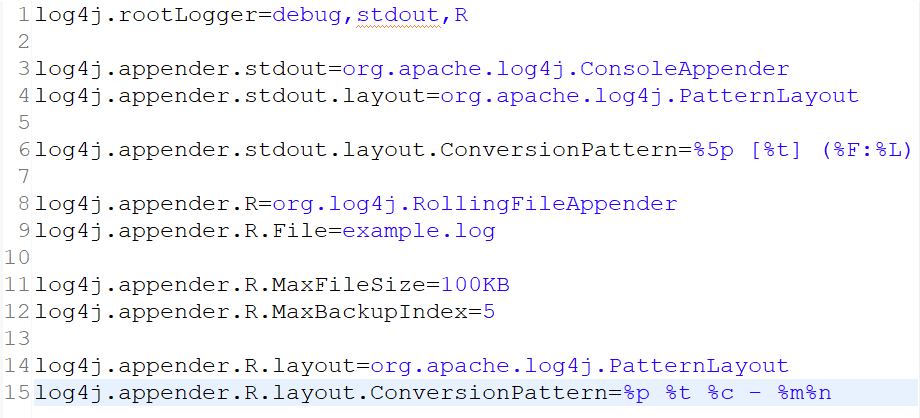
logger.error("错误信息");

logger.fatal("致命信息");

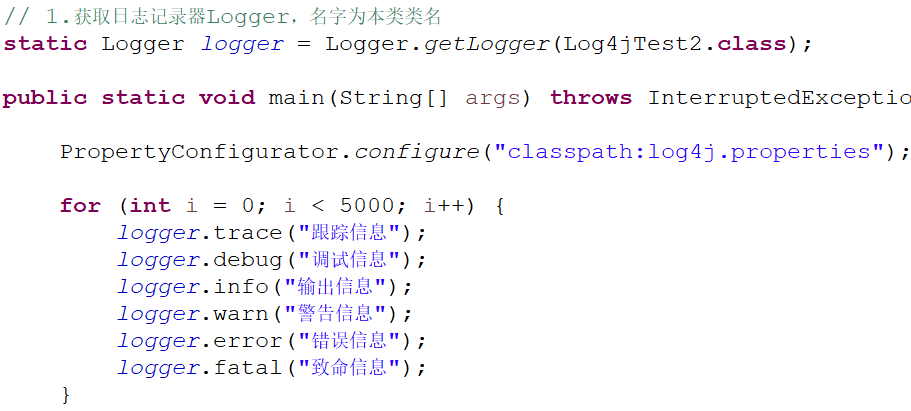
Thread.sleep(1000);是为了便于观察日志输出的时间差。

1. **配置讲解**
2. **log4j.properties**

首先在src目录下添加log4j.properties文件。



1. **Log4jTest2**
2. 输出在控制台，格式有所变化，如图，会显示是哪个类的哪一行输出的信息；
3. 不仅仅在控制台有输出，再把日志输出到



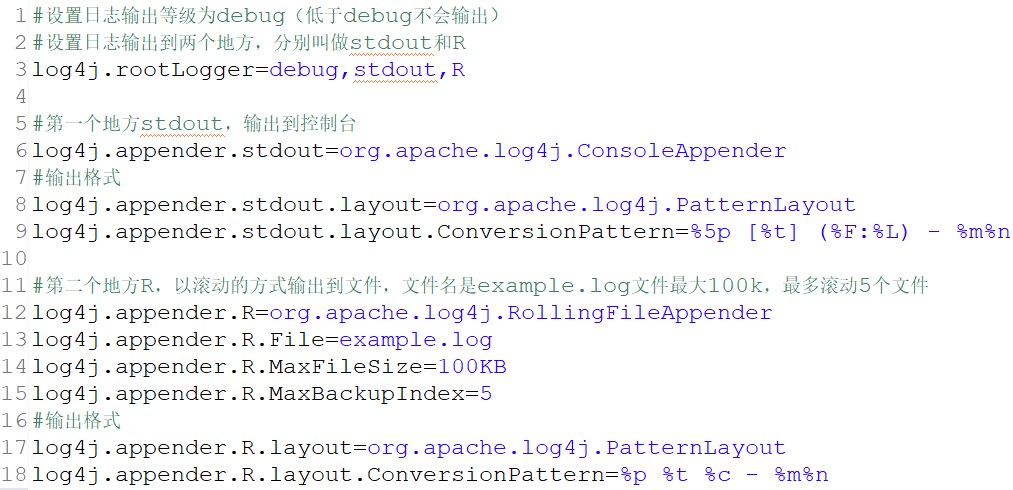
1. **分析代码**

与Log4jTest中的BasicConfigurator.configure();方式不同，采用指定配置文件

PropertyConfigurator.*configure*("classpath:log4j.properties");

Log4j的配置方式按照log4j.properties中的设置进行。

1. **解释log4j.properties**



1. **格式解释**

|  |  |
| --- | --- |
| 格式 | 解释 |
| %c | 输出日志信息所属的类的全名 |
| %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 | 输出日志信息产生时所在的文件名称 |
| %l | 输出日志事件的发生位置，即代码行数 |
| %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 |
| %p | 输出优先级 |
| %r | 输出自应用启动到输出该日志所耗费的毫秒数 |
| %t | 输出产生该日志事件的线程名 |

%5p [%t] (%F:%L) - %m%n 就表示  
宽度是5的优先等级 线程名称 (文件名:行号) - 信息 回车换行

1. **Log4j.xml**
2. **log4j.xml**

log4j.properties，也可以使用xml格式进行配置。在src目录下装备log4j.xml文件。

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE log4j:configuration PUBLIC "-//log4j/log4j Configuration//EN" "log4j.dtd">

<log4j:configuration xmlns:log4j=*"http://jakarta.apache.org/log4j/"*>

<appender name=*"STDOUT"* class=*"org.apache.log4j.ConsoleAppender"*>

<layout class=*"org.apache.log4j.PatternLayout"*>

<param name=*"ConversionPattern"* value=*"%d %-5p %c.%M:%L - %m%n"* />

</layout>

</appender>

<!-- specify the logging level for loggers from other libraries -->

<logger name=*"com.opensymphony"*>

<level value=*"ERROR"* />

</logger>

<logger name=*"org.apache"*>

<level value=*"ERROR"* />

</logger>

<logger name=*"org.hibernate"*>

<level value=*"ERROR"* />

</logger>

<!-- for all other loggers log only debug and above log messages -->

<root>

<priority value=*"ERROR"* />

<appender-ref ref=*"STDOUT"* />

</root>

</log4j:configuration>

1. Log4jTest3

