日志(Logging)

如果只是要作一些简单的文件日志，可以考虑包含在JDK中的Logging API 。 功能或许简单了一些，但好处是它从J2SE1.4之后就成为了Java SE 标准API的一员，这样不必额外配置logging组件就可以运行在标准的Java平台上。无论是在命令行模式简单地显示信息，或是在文件中输出日志，java.util.logging下都提供了一些默认的工具类。

common-logging和Log4j的配合使用，在项目中也很常用。

# 简介Logging

程序中不免会出现错误，如果想将错误信息以某种方式存储下来，以供用户或是程序人员调试(Debug)时使用，Java SE中的java.util.logging包提供了一系列的logging工具类。

要使用JavaSE的日志功能，首先要取得java.util.logging.Logger实例，这可以通过Logger类的getLogger()方法来取得。

看范例18.7，假设程序在启动时必须提供参数给程序，如果用户没有提供参数则必须显示警示信息。

**public** **class** LoggingDemo {

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

Logger logger = Logger.*getLogger*("LoggingDemo");

**try** {

System.*out*.println(args[0]);

} **catch** (ArrayIndexOutOfBoundsException e) {

logger.warning("没有提供执行时的参数！");

}

}

}

执行结果：

2014-3-30 19:41:41 cn.ifinance.LoggingDemo main

警告: 没有提供执行时的参数！

简单地通过Logger的静态方法getLogger()取得Logger对象，给getLogger()的字符串参数被用来作为Logger对象的名称，之后就可以运用Logger实例上的方法进行信息输出，默认是从命令行模式输出信息。除了提供的文字信息之外，Logger还自动收集了执行时相关的信息，像信息输出所在的类与包名称，执行该段程序时的线程名称以及信息产生的时间等。

对于大中型系统来说，java.util.logging包下所提供的工具在功能上可能不足，但作为一个简单的日志工具，java.util.logging包下所提供的工具是可以考虑的方案。

# Logging的等级

在进行信息的记录时，依信息程度的不同，会设置不同等级的信息输出。可以通过操作Logger上的几个方法来得到不同等级的信息输出。

如范例18.8：

**public** **class** LoggingLevelDemo {

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

Logger logger = Logger.*getLogger*("loggingLevelDemo");

logger.severe("严重信息");

logger.warning("警示信息");

logger.info("一般信息");

logger.config("设置方面的信息");

logger.fine("细微的信息");

logger.finer("更细微的信息");

logger.finest("最细微的信息");

}

}

执行结果：

2014-3-30 19:58:30 cn.ifinance.LoggingLevelDemo main

严重: 严重信息

2014-3-30 19:58:30 cn.ifinance.LoggingLevelDemo main

警告: 警示信息

2014-3-30 19:58:30 cn.ifinance.LoggingLevelDemo main

信息: 一般信息

依照不同的等级，可以采用不同的信息等级方法，程序中依程度高低编写。

注意到config()方法及以下的信息并没有显示出来，这是因为Logger的默认等级是INFO,比这个等级更低的信息,Logger并不会将信息输出。

Logger的默认等级是定义在执行环境的属性文件logging.properties中，这个文件位于JRE的安装目录的lib目录下。部分内容如下：

handlers= java.util.logging.ConsoleHandler

.level= INFO

java.util.logging.ConsoleHandler.level = INFO

Logger默认的处理者（Handler）是java.util.logging.ConsolerHandler。也就是将信息输出至命令行模式。一个Logger可以拥有多个处理者，每个处理者可以有自己的信息等级，在通过Logger的等级限制后，实际上还要再经过处理者的等级限制。所以在范例18.8中如果想要看到所有的信息，则必须同时设置 Logger 与 ConsoleHandler的等级。

范例18.9示范了如何设置：

**public** **class** LoggingLevelDemo2 {

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

Logger logger = Logger.*getLogger*("loggingLevelDemo2");

//显示所有的等级信息

logger.setLevel(Level.*ALL*);

ConsoleHandler consoleHandler = **new** ConsoleHandler();

//显示所有等级的信息

consoleHandler.setLevel(Level.*ALL*);

//设置处理者为consoleHandler

logger.addHandler(consoleHandler);

logger.severe("严重信息");

logger.warning("警示信息");

logger.info("一般信息");

logger.config("设置方面的信息");

logger.fine("细微的信息");

logger.finer("更细微的信息");

logger.finest("最细微的信息");

}

}

如果要关闭所有的信息，可以设置为Level.OFF .

Logger的server()、warning()、info()等方法，实际上是个便捷的方法。也可以直接使用log()方法并指定等级来执行相同的作用。

例如18.10，其执行结果与范例18.9是一样的。

**public** **class** LoggingLevelDemo3 {

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

Logger logger = Logger.*getLogger*("loggingLevelDemo3");

//显示所有的等级信息

logger.setLevel(Level.*ALL*);

ConsoleHandler consoleHandler = **new** ConsoleHandler();

//显示所有等级的信息

consoleHandler.setLevel(Level.*ALL*);

//设置处理者为consoleHandler

logger.addHandler(consoleHandler);

logger.log(Level.*SEVERE*,"严重信息");

logger.log(Level.*WARNING*,"警示信息");

logger.log(Level.*INFO*,"一般信息");

logger.log(Level.*CONFIG*,"设置方面的信息");

logger.log(Level.*FINE*,"细微的信息");

logger.log(Level.*FINER*,"更细微的信息");

logger.log(Level.*FINEST*,"最细微的信息");

}

}

# Handler和Formatter

Logger默认的输出处理者是ConsoleHandler 。 ConsoleHandler的输出是使用System.err对象，而信息的默认等级是 INFO ， 这可以在JRE安装目录下lib目录的logging.properties中看到：

handlers= java.util.logging.ConsoleHandler

java.util.logging.ConsoleHandler.level = INFO

JavaSE提供了5个默认的Handler :

* java.util.logging.ConsoleHandler

以System.err输出日志。

* java.util.logging.FileHandler

将信息输出至文件。

* java.util.logging.StreamHandler

以指定的OutputStream实例输出日志。

* java.util.logging.SocketHandler

将信息通过Socket传送至远程主机。

* java.util.logging.MemoryHandler

将信息暂存在内存中。

范例18.11 示范了如何使用FileHandler将信息输出至文件中。

**public** **class** HandlerDemo {

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

Logger logger = Logger.*getLogger*("handlerDemo");

**try** {

FileHandler fileHandler = **new** FileHandler("%h/myLogger.log");

logger.addHandler(fileHandler);

logger.info("测试信息");

} **catch** (SecurityException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

执行的结果会在命令行模式显示信息，并将结果输出至文件中。在指定输出的文件名称时，可以使用%h来表示用户的根目录(windows系统中的用户根目录是在“\Documents and Setting\用户名称”目录中)。还可以使用%t取得系统的暂存目录，或者是加入%g自动为文件编号。例如可以设置为%h/myLogger%g.log，表示将.log文件存储在用户根目录中，并自动为每一个文件增加编号。

FileHandler默认的输出格式是XML格式，输出格式是由java.util.logging.Formatter来控制。例如FileHandler的默认格式是java.util.logging.XMLFormatter ,而ConsolerHandler默认格式是java.util.logging.SimpleFormatter ，可以使用Handler实例的setFormatter()方法来设置信息的输出格式。例如：

fileHandler.setFormatter(new SimpleFormatter());

如果FileHandler的Formatter设置为SimpleFormatter，则输出的日志文件内容就会是简单的文字信息，打开后会发现与命令行模式下看到的信息内容相同。

# 自定义Formatter

除了XMLFormatter与SimpleFormatter 之外，也可以自定义日志的输出格式，只要继承抽象类Formatter ,并重新定义其format()方法即可。format()方法会传入一个java.util.LogRecord对象作为参数，可以使用它类取得一些与程序执行有关的信息。

范例18.12：

**public** **class** TableFormatter **extends** Formatter {

@Override

**public** String format(LogRecord logRecord) {

**return** "LogRecord info: "+

logRecord.getSourceClassName()+"\n"+

"Level\t|\tLoggerName\t|\tMessage\t|\n"+

logRecord.getLevel()+"\t|\t"+

logRecord.getLoggerName()+"\t|\t"+

logRecord.getMessage()+"\t|\n\n"

;

}

}

范例18.13：

**public** **class** TableFormatterDemo {

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

Logger logger = Logger.*getLogger*("tableFormatter");

**try** {

**for** (Handler h : logger.getParent().getHandlers()) {

**if** (h **instanceof** ConsoleHandler) {

h.setFormatter(**new** TableFormatter());

}

}

logger.info("信息");

logger.warning("信息2");

} **catch** (Exception e) {}

}

}

你取得默认的根(Root)Logger，并判断出ConsoleHandler实例，之后设置ConsoleHandler实例的Formatter为自定义的TableFormatter。执行结果如下：

LogRecord info: cn.ifinance.TableFormatterDemo

Level | LoggerName | Message |

INFO | tableFormatter | 信息 |

LogRecord info: cn.ifinance.TableFormatterDemo

Level | LoggerName | Message |

WARNING | tableFormatter | 信息2 |

# Logger的层次关系

在使用Logger 的静态方法取得 Logger 实例是，给getLogger()方法的名称是有意义的。如果给定onlyfun,实际上将从根（Root）logger继承一些特性，像日志等级（Level）以及根logger的处理者。如果再取得一个Logger实例，并给定名称onlyfun.caterpillar，则这次取得的Logger将继承onlyfun这个Logger的特性。

范例18.14可看出，3个Logger(包括根logger)在名称上的继承关系。

**public** **class** LoggerHierarchyDemo {

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

Logger onlyfunLogger = Logger.*getLogger*("onlyfun");

Logger caterpillarLogger = Logger.*getLogger*("onlyfun.caterpillar");

System.*out*.println("root logger: "+onlyfunLogger.getParent());

System.*out*.println("onlyfun logger: "+caterpillarLogger.getParent().getName());

System.*out*.println("caterpillar logger: "+caterpillarLogger.getName()+"\n");

onlyfunLogger.setLevel(Level.*WARNING*);

caterpillarLogger.info("caterpillar' inof");

caterpillarLogger.setLevel(Level.*INFO*);

caterpillarLogger.info("caterpillar' info");

}

}

执行结果：

root logger: java.util.logging.LogManager$RootLogger@14318bb

onlyfun logger: onlyfun

caterpillar logger: onlyfun.caterpillar

2014-3-30 22:33:58 cn.ifinance.LoggerHierarchyDemo main

信息: caterpillar' info

getParent()方法可以取得Logger的上层父Logger，根logger并没有名称，所以直接调用它的toString()以取得字符串描述。当Logger没有设置等级时，则使用父Logger的等级设置，所以在范例18.14中，onlyfunLogger设置等级为WARNING时，caterpillar调用info()方法时并不会有信息显示（因为WARNING等级比INFO高）。只有在caterpillarLogger设置了自己的等级为INFO之后，才会有显示信息。

在每一个处理者方面，当每一个Logger处理完自己的日志动作之后，它会向父Logger传播，让父Logger的处理者也可以处理日志。例如root logger的处理者是ConsoleHandler , 所以子Logger加入了FileHandler来处理完日志之后，向上传播至父Logger , 会再由ConsolerHandler来处理，故同样会在命令模式上显示信息，之前的范例18.11就是这样的例子。