日志介绍

有很多的日志框架，以为二大类

- 日志抽象层（日志门面）
- 日志实现层

之所以会分成这样二大类是因为每次如果都对实现编程的话，那么就会改动很多，再此基础上提取一个抽象层，然后针对接口编程以达到目的

日志分类

日志抽象层

- JCL： jakarta commons logging
- SLF4j： simple logging facade for java
- jboss-logging

日志实现层

- log4j
- log4j2
- logback
- JUL：java.util.logging

SLF4j，log4j，logback是同一个人写的
JUL：java.util.logging 是jdk1.4自带的，为了防止被log4j占有市场，所以临时出的，功能不咋地
log4j2是apache的

发展历史

最早Apache 提供的Log4j
SUN觉得我也要搞个，然后就在jdk1.4中模仿创建了JUL
又因为有的人使用log4j，有的人又使用JUL，造成了混乱
所以Apache又创建了Apache Commons Logging，这是一个日志门面接口，支持动态加载实现，底层可以是log4j也可以是JUL
后来log4j的作者觉得这个不行，他又创建了slf4j
slf4j和Apache Commons Logging类似，也是一个日志门面接口，同时他还写了一个日志的实现logback，替代了log4j

最后Apache也写了个log4j2

Logger

配置文件详解

较为常用的配置项设置

```
# handlers 配置要输出的位置
# java.util.logging.ConsoleHandler = 控制台
# java.util.logging.FileHandler = 文件
handlers= java.util.logging.ConsoleHandler,java.util.logging.FileHandler

# 全局配置，其凌驾于所有的等级配置之上
.level= WARNING


# 文件具体配置项
# 文件路径和名称的格式
# "/" 本地路径名分隔符
# "%t" 系统临时目录
# "%h" "user.home" 系统属性的值
# "%g" 区分循环日志的生成号
# "%u" 解决冲突的惟一号码
# "%%" 转换为单个百分数符号"%"
java.util.logging.FileHandler.pattern = z:/work/test/log/log%u.log
# 每个文件最大的存放的字节数
java.util.logging.FileHandler.limit = 50000
# 指代能存放多少个文件，例如上方配置50000字节，当达到时就会因为count=1导致需要删除满的，然后创建新的存放
java.util.logging.FileHandler.count = 1
# 文件输出格式，默认是xml格式 = java.util.logging.XMLFormatter
# java.util.logging.SimpleFormatter 会以字符串的形式输出
java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
# 设置每次输出是否追加的已有文件的后面，默认是false
java.util.logging.FileHandler.append = true

# 设置默认控制台输出的Level（受全局Level影响）
java.util.logging.ConsoleHandler.level = WARNING
# 设置控制台输出格式
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

# 设置SimpleFormatter的输出格式
java.util.logging.SimpleFormatter.format=%4$s: %5$s [%1$tc]%n

# 设置具体的类的输出Level
com.chenx.test = WARNING
```

配置文件位置

java.util.logging 是java 1.4 自带出现的，因此其配置文件自然在其Java安装目录下

配置文件位置： C:\Program Files\Java\jdk1.8.0_191\jre\lib\logging.properties

如何使用自己的配置文件

Logger 有一个管理类 LogManager，通过此类加载具体配置文件位置

```
public static void main(String[] args) {
      // 创建LogManager
   LogManager manager = LogManager.getLogManager();
   // 创建Logger
   Logger log = Logger.getLogger("test");
   try {
      // 获取配置文件的输入流,配置文件放在src下
      InputStream input = test.class
            .getClassLoader()
            .getResourceAsStream("log.properties");
      // 读取配置输入流
      manager.readConfiguration(input);
      // 将logger加入manager，以获取到配置文件中的配置
      manager.addLogger(log);

      log.info("dd");
   } catch(Exception e) {
      System.out.print(e);
   }
}
```

需要了解

加载了配置之后，需要将使用的logger加入到管理中，不然使用的还是全局配置文件的配置

Commons—Logging

common-logging 内部使用流程

common-logging 是用于简化Log4j

common-logging 通过查找classpath 是否有 commons-logging.properties，根据里面的信息判断使用什么进行log输出（Logger，SimpleLog，Log4JLogger），如果没有此配置文件则查看是否有Log4j的包，有的话再去 classpath 下查找是否有 Log4j 的配置文件 log4j.properties，如果没有Log4j的包，那么看JDK是否是1.4以上，是则使用JDK中的Logger，反之则使用其自带的SimpleLog

```
# commons-logging 配置文件
# 设置commons-Logging使用何种方式输出
#org.apache.commons.logging.Log=org.apache.commons.logging.impl.Log4JLogger

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


# simplelog.properties 配置文件
# 是否显示logName
org.apache.commons.logging.simplelog.showlogname=true
# 是否以简短的方式显示logName
org.apache.commons.logging.simplelog.showShortLogname=false
# 是否显示日期
org.apache.commons.logging.simplelog.showdatetime=true
# 显示日期的格式
org.apache.commons.logging.simplelog.dateTimeFormat=yyyy-MM-dd HH:mm:ss
# 指定名称的log对象的输出等级
org.apache.commons.logging.simplelog.log.XXXXXX=debug
# 所有的log对象的输出等级，其优先级低于具体配置的对象
org.apache.commons.logging.simplelog.defaultlog=debug
```

Log4j

使用 commons-logging 配合使用 Log4j

需要加入Log4j的包

classpath下加入log4j.properties文件

log4j 配置文件详解

Level： 输出等级

- debug 输出“调试”级别的日志信息
- info 输出“信息”级别的日志信息
- warn 输出“警告”级别的日志信息
- error 输出“错误”级别的日志信息
- fatal 输出“致命错误”级别的日志信息

Appender ： 输出目的地

- 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）

Layout：日志输出格式

- 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）

打印参数

- %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{yyy 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。举例：Testlog4.main(TestLog4.java: 10)

```
# log4j.properties配置文件
# 配置要求：控制台输出，文件输出，指定输出格式
log4j.rootLogger = debug, MyConsole, MyFileDebug, MyFileError

# 配置控制台输出
log4j.appender.MyConsole = org.apache.log4j.ConsoleAppender
# 使用System.out 输出（以信息的形式，黑色字体），还可以使用System.err（以报错的方式，红色字体）
log4j.appender.MyConsole.Target = System.out
log4j.appender.MyConsole.layout = org.apache.log4j.PatternLayout
log4j.appender.MyConsole.layout.ConversionPattern = %d{ yyyy-MM-dd HH:mm:ss} [%p]:%c: %m%n

# 配置输出到文件
# 每天产生一个日志文件
log4j.appender.MyFileDebug = org.apache.log4j.DailyRollingFileAppender
# 输出位置
log4j.appender.MyFileDebug.File = Z:/work/test/src/Test/logs/log.log
# 是否追加到文件
log4j.appender.MyFileDebug.Append = true
# 输出等级
log4j.appender.MyFileDebug.Threshold = DEBUG
log4j.appender.MyFileDebug.layout = org.apache.log4j.PatternLayout
log4j.appender.MyFileDebug.layout.ConversionPattern = %d{ yyyy-MM-dd
```

```
HH:mm:ss} [%p]:%c: %m%n

# 配置输出到文件
# 每天产生一个日志文件
log4j.appender.MyFileError = org.apache.log4j.DailyRollingFileAppender
# 输出位置
log4j.appender.MyFileError.File = Z:/work/test/src/Test/logs/errorlog.log
# 是否追加到文件
log4j.appender.MyFileError.Append = true
# 输出等级
log4j.appender.MyFileError.Threshold = error
log4j.appender.MyFileError.layout = org.apache.log4j.PatternLayout
log4j.appender.MyFileError.layout.ConversionPattern = %d{ yyyy-MM-dd
HH:mm:ss} [%p]:%c: %m%n
```