1、日志级别

org.apache.log4j.Level类提供以下级别，但也可以通过Level类的子类自定义级别。

Level	描述
ALL	各级包括自定义级别
DEBUG	指定细粒度信息事件是最有用的应用程序调试
ERROR	错误事件可能仍然允许应用程序继续运行
FATAL	指定非常严重的错误事件，这可能导致应用程序中止
INFO	指定能够突出在粗粒度级别的应用程序运行情况的信息的消息
OFF	这是最高等级，为了关闭日志记录
TRACE	指定细粒度比DEBUG更低的信息事件
WARN	指定具有潜在危害的情况

日志级别是如何工作？

级别p的级别使用q，在记录日志请求时，如果p>=q启用。这条规则是log4j的核心。它假设级别是有序的。对于标准级别它们关系如下：ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF。

下面的例子明确指出如何可以过滤所有的DEBUG和INFO消息。这个程序使用记录并执行setLevel（Level.X）方法来设置所需的日志记录级别：

这个例子将打印，除了调试和信息的所有消息：

```
import org.apache.log4j.*;

public class LogClass {
   private static org.apache.log4j.Logger log = Logger
                                    .getLogger(LogClass.class);
   public static void main(String[] args) {
      log.setLevel(Level.WARN);

      log.trace("Trace Message!");
      log.debug("Debug Message!");
      log.info("Info Message!");
      log.warn("Warn Message!");
      log.error("Error Message!");
      log.fatal("Fatal Message!");
   }
}
```

当编译并运行LogClass程序会产生以下结果：

```
Warn Message!
Error Message!
Fatal Message!
```

使用配置文件设置级别：

Log4j提供这些可以让程序员自由更改源代码，改变调试级别的配置级别是基于文件设置。以下是上面的例子使用 log.setLevel（Level.WARN）方法的配置文件与上面的例子例子功能一样。

```
# Define the root logger with appender file
log = /usr/home/log4j
log4j.rootLogger = WARN, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=${log}/log.out

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

现在，使用下面的程序：

```
import org.apache.log4j.*;

public class LogClass {
   private static org.apache.log4j.Logger log = Logger
                                    .getLogger(LogClass.class);
   public static void main(String[] args) {
      log.trace("Trace Message!");
      log.debug("Debug Message!");
      log.info("Info Message!");
      log.warn("Warn Message!");
      log.error("Error Message!");
      log.fatal("Fatal Message!");
   }
}
```

现在，编译和运行上面的程序，得到以下结果在 /usr/home/log4j/log.out 文件：

```
Warn Message!
Error Message!
Fatal Message!
```

2、日志格式化

Apache log4j 提供了各种布局对象，每一个对象都可以根据各种布局格式记录数据。另外，也可以创建一个布局对象格式化测井数据中的特定应用的方法。

所有的布局对象 - Appender对象收到 LoggingEvent 对象。布局对象检索来自 LoggingEvent 的消息参数，并应用适当的 ObjectRenderer 获得消息的字符串表示。

布局类型：

在层次结构中的顶级类是抽象类是org.apache.log4j.Layout。这是 log4j 的 API 中的所有其他布局类的基类。

布局类定义为抽象在应用程序中，不要直接使用这个类; 相反，使用它的子类来工作，如下：

- DateLayout
- HTMLLayout (在本教程解释)
- PatternLayout (在本教程解释)
- SimpleLayout
- XMLLayout

布局方法：

这个类提供了一个框架实现在所有其它布局对象的所有常见的操作，并声明了两个抽象方法。

S.N.	方法 & 描述
1	**public abstract boolean ignoresThrowable()** 这种方法表示日志信息是否处理传递给它的日志记录事件的一部分，任何 java.lang.Throwable 对象。如果布局对象处理 Throwable 对象，那么布局对象不忽视它，并返回false。
2	**public abstract String format(LoggingEvent event)** 独特的布局子类将实施这一方法的布局特定的格式

除了这些抽象方法，布局类提供具体的实现下列方法：

S.N.	方法 & 描述
1	**public String getContentType()** 返回使用的布局的对象的内容类型。基类将返回 text/plain 作为默认的内容类型
2	**public String getFooter()** 指定日志消息的页脚信息
3	**public String getHeader()** 指定日志消息的标头信息

每个子类可以通过重写的具体实现这些方法返回类特定的信息。