Log4j API设计为分层结构，每一层提供了不同的对象，其中包括核心对象、支持对象

核心对象：

Logger 对象：负责捕捉日志信息及存储在一个空间的层次结构。

Layout对象：格式化输出不同风格的的日志信息

Appender 对象：（输出源）负责发布日志信息，控制输出到对应的位置

支持对象；

Level对象：level定义任何记录信息的粒度和优先级，off debug info error warn fatal all

Filter对象：分析日志信息过滤日志的记录 一个appender对象可以关联多个filter对象

日志管理：系统级的配置文件或配置初始化参数

Log4j.properties语法

# Define the root logger with appender X 定义记录器的级别为bebug 并且命名为x

log4j.rootLogger = DEBUG, X

# Set the appender named X to be a File appender 为x指定一个有效输出源以及输出文件位置

log4j.appender.X=org.apache.log4j.FileAppender

log4j.appender.FILE.File=${log}/log.out

# Define the layout for X appender 定义x 布局模式 以及转换模式

log4j.appender.X.layout=org.apache.log4j.PatternLayout

log4j.appender.X.layout.conversionPattern=%m%n

appender与之关联的不同的属性，并且这些属性表名对象的行为

layout 指定布局对象以及相关的格式化的日志记录信息转换模式。

Target 目标可以是控制台，一个文件或者另一个项目

Level 控制日志消息的过滤级别

Filter 可分析出超出级别的匹配记录信息

log4j.appender.X可以输出到不同目的地：

ConsoleAppender

FileAppender

JDBCAppender

DailyRollingFileAppender 每天生成日志文件

ExternallyRolledFileAppender 指定大小的日志文件，超出即新建

Fileappender 日志文件输出路径：

相对路径法：

log4j.appender.logfile.File=../logs/app.log，将日志记录到tomcat下的logs文件夹；

log4j.appender.logfile.File=logs/app.log，将日志记录到tomcat的bin目录下的logs文件夹；

log4j.appender.logfile.File=${catalina.home}/app.log，将日志记录到tomcat的bin目录下的logs文件夹；

环境变量相对路径：

1、如果是在eclipse中执行这些日志输出的类的方法，那么将在workspace所在的磁盘，例如E盘中创建目录/usr/local/var/log，并输出full.log和solr.log.

2、如果是在tomcat中应用执行日志的输出，那边就是在tomcat所在的磁盘中，例如D盘中创建目录/usr/local/var/log，并输出full.log和solr.log,即便是在eclipse中启动tomcat，这时tomcat的wtpapps目录是在eclipse所在的目录，如E盘中，但是真正使用的tomcat是在D盘中的，那么同样是在/usr/local/var/logs中输出如上的日志文件。

log4j.home由容器启动时指定，jvm中加上 -Dlog4j.home=D:/log

这样在Log4j.xml中的${log4j.home}就知道了实际的输出目录了，

同样也可以把这个log4j.home放到分离的properties中，这时候可以

在容器启动时在监听器来解析properties，获取到log4j.home变量后

把值设置到system.env中

JDBCAppender输出源到数据库：

例子：

建表 CREATE TABLE LOGS

(USER\_ID VARCHAR(20) NOT NULL,

DATED DATE NOT NULL,

LOGGER VARCHAR(50) NOT NULL,

LEVEL VARCHAR(10) NOT NULL,

MESSAGE VARCHAR(1000) NOT NULL

);

配置文件：

# Define the root logger with appender file

log4j.rootLogger = DEBUG, DB

# Define the DB appender

log4j.appender.DB=org.apache.log4j.jdbc.JDBCAppender

# Set JDBC URL

log4j.appender.DB.URL=jdbc:mysql://localhost/DBNAME

# Set Database Driver

log4j.appender.DB.driver=com.mysql.jdbc.Driver

# Set database user name and password

log4j.appender.DB.user=user\_name

log4j.appender.DB.password=password

# Set the SQL statement to be executed.

log4j.appender.DB.sql=INSERT INTO LOGS VALUES('%x','%d','%C','%p','%m')

# Define the layout for file appender

log4j.appender.DB.layout=org.apache.log4j.PatternLayout

Layout （基类）

模式选项：（子类）

DateLayout

HTMLLayout

PatternLayout

SimpleLayout

XMLLayout

HTMLLayout 生成

例子：

log4j.rootLogger = DEBUG, FILE

# Define the file appender

log4j.appender.FILE=org.apache.log4j.FileAppender

log4j.appender.FILE.File=htmlLayout.html

# Define the layout for file appender

log4j.appender.FILE.layout=org.apache.log4j.HTMLLayout

log4j.appender.FILE.layout.Title=HTML Layout Example

log4j.appender.FILE.layout.LocationInfo=true

patternlayout生成模式的conversionPattern

c 用于输出的记录事件的类别。例如，对于类别名称"a.b.c" 模式 %c{2} 会输出 "b.c"

C 用于输出呼叫者发出日志请求的完全限定类名。例如，对于类名 "org.apache.xyz.SomeClass", 模式 %C{1} 会输出 "SomeClass".

d 用于输出的记录事件的日期。例如， %d{HH:mm:ss,SSS} 或 %d{dd MMM yyyy HH:mm:ss,SSS}.

F 用于输出被发出日志记录请求，其中的文件名

l 用于将产生的日志事件调用者输出位置信息

L 用于输出从被发出日志记录请求的行号

m 用于输出使用日志事件相关联的应用程序提供的消息

M 用于输出发出日志请求所在的方法名称

n 输出平台相关的行分隔符或文字

p 用于输出的记录事件的优先级

r 用于输出毫秒从布局的结构经过直到创建日志记录事件的数目

t 用于输出生成的日志记录事件的线程的名称

x 用于与产生该日志事件的线程相关联输出的NDC（嵌套诊断上下文）

X 在X转换字符后面是键为的MDC。例如 X{clientIP} 将打印存储在MDC对键clientIP的信息

% 文字百分号 %%将打印％标志

例子：

日期 - 类名 - 最大长度为4的r毫秒值 - [线程名] - 最大长度为5的优先级 %n 事件相关的消息 换行

log4j.appender.FILE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p %x - %m%n

Java 中获取logger对象

Logger logger = Logger.getLogger(test.class);//默认在classpath路径下加载配置文件。

Logger的方法：

1 public void debug(Object message)

这种方法打印使用 Level.DEBUG 消息级别

2 public void error(Object message)

这种方法打印使用 Level.ERROR 消息级别

3 public void fatal(Object message);

这种方法打印使用 Level.FATAL 消息级别

4 public void info(Object message);

这种方法打印使用 Level.INFO 消息级别

5 public void warn(Object message);

这种方法打印使用 Level.WARN 消息级别

6 public void trace(Object message);

这种方法打印使用Level.TRACE消息级别

Level级别

All

Debug 指定细粒度信息事件是最有用的应用程序调试（后期调试的提示日志）；

Info 突出粗粒度级别的应用程序运行情况的信息的消息（比如某些初始化信息）

Warn 指定具有潜在危害的情况

Error 错误事件可能仍然允许应用程序的继续执行，而打印的错误信息

Fatal 表示严重错误，会导致程序终止

Off关闭日志记录

级别从上到下逐次增加（严重程度在增加），在实际工作中我们需要将严重的错误收集起来，所以指定level后，级别高于指定（越严重的）的日志会被输出

 %c    输出所属类的全名，可在修改为 %d{Num} ,Num类名输出的维（如："org.apache.elathen.ClassName",%C{2}将输出elathen.ClassName）

     %d    输出日志时间其格式为 %d{yyyy-MM-dd HH:mm:ss,SSS}，可指定格式 如 %d{HH:mm:ss}

     %l    输出日志事件发生位置，包括类目名、发生线程，在代码中的行数

     %n    换行符

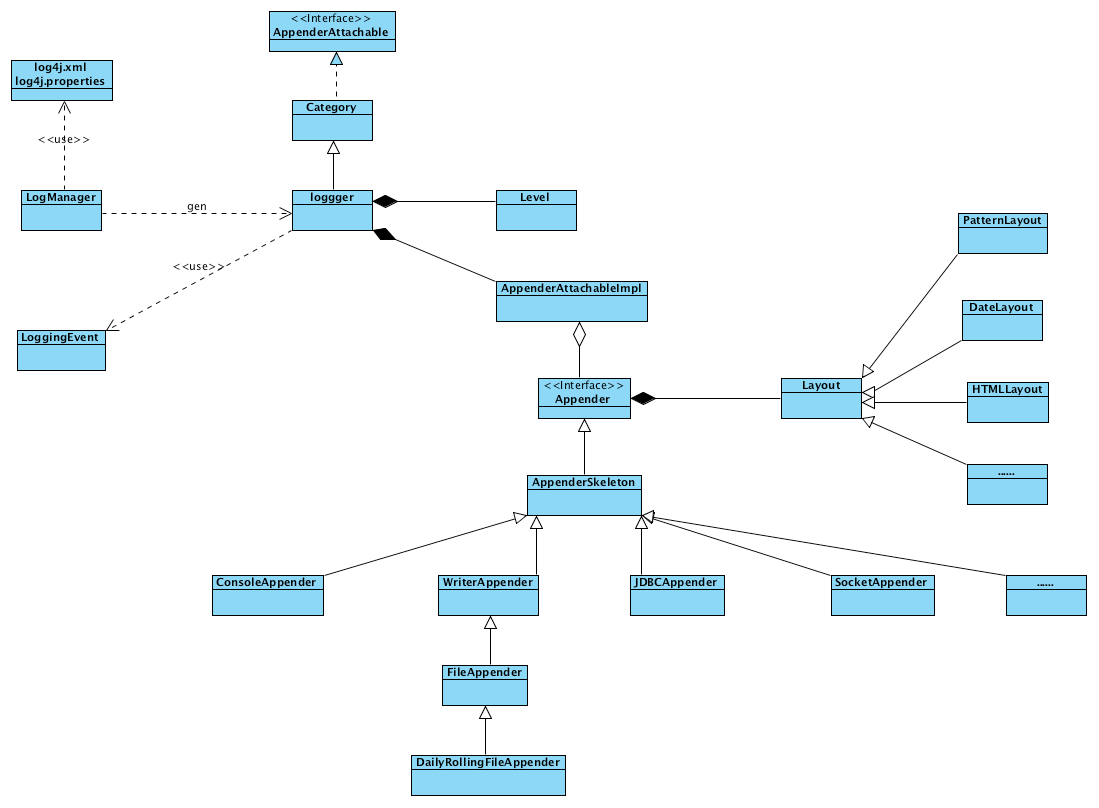
     %m    输出代码指定信息，如info(“message”),输出message

     %p    输出优先级，即 FATAL ,ERROR 等

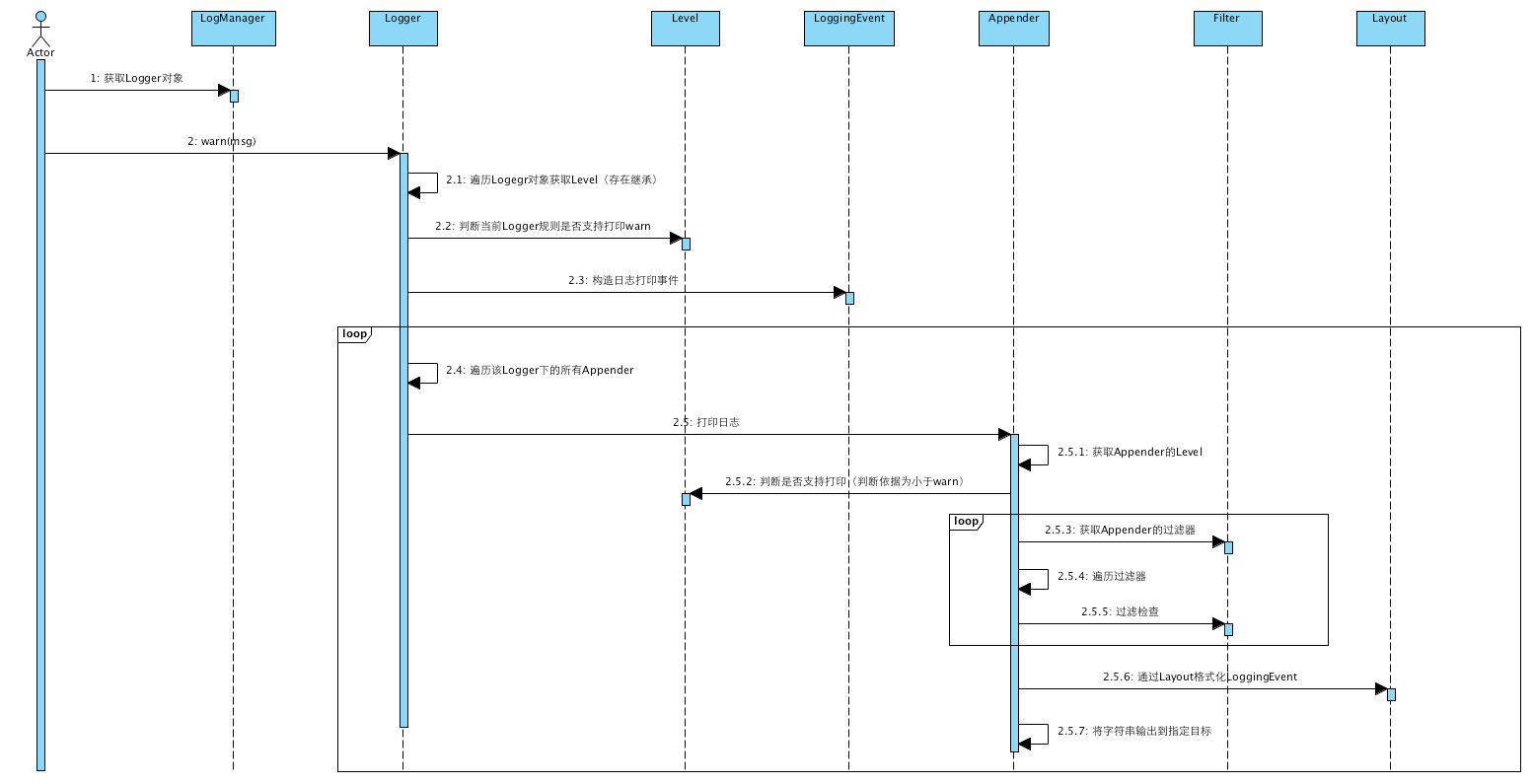
     %r    输出从启动到显示该log信息所耗费的毫秒数

     %t    输出产生该日志事件的线程名

核心类图：



日志打印流程：



Log4jxml格式配置：