log4cpp 相关知识讲解

格式化器

Layout:布局，负责设定日志的格式

涉及头文件：

#include<log4cpp/BasicLayout.hh>
#include<log4cpp/SimpleLayout.hh>
#include<log4cpp/PatternLayout.hh>

->BasicLayout: "以时间戳 + 优先级 + 内容"

构造函数：
log4cpp::BasicLayout::BasicLayout()
输出例子格式：
1248337987 ERROR : Hello log4cppin a Error Message!
1248337987 WARN : Hello log4cppin a Warning Message!

->SimpleLayout： "优先级 + 日志信息"

构造函数：
log4cpp::SimpleLayout::SimpleLayout()

->PatterLayout: "使用类似 printf 的格式化模式"（推荐使用这种）

构造函数：
log4cpp::PatternLayout::PatternLayout()
PatterLayout 设置日志格式函数
void log4cpp::PatternLayout::setConversionPattern(const std::string &conversionPattern) throw (ConfigureFailure)
使用 PatterLayout 时的格式化参数：
u %c category；
u %d 日期；日期可以进一步的设置格式，用花括号包围，例如%d{%H:%M:%S,%l} 或者 %d{%d %m %Y%H:%M:%S,%l}。如果不设置具体日期格式，则如下默认格式被使用“Wed Jan 02 02:03:55 1980”。日期的格式符号与 ANSI C 函数 strftime 中的一致。但增加了一个格式符号%l，表示毫秒，占三个十进制位。
u %m 消息；
u %n 换行符，会根据平台的不同而不同，但对于用户透明；
u %p 优先级；
u %r 自从 layout 被创建后的毫秒数；
u %R 从 1970 年 1 月 1 日 0 时开始到目前为止的秒数；
u %u 进程开始到目前为止的时钟周期数；
u %x NDC。
举例：
log4cpp::PatternLayout* pLayout = new log4cpp::PatternLayout();
pLayout->setConversionPattern("%d: [%p] %c %x: %m%n");

Appender:附加目的地，负责指定日志的目的地

输出器

涉及头文件举例：

#include <log4cpp/Appender.hh>
#include <log4cpp/OstreamAppender.hh>
#include <log4cpp/StringQueueAppender.hh>
#include <log4cpp/FileAppender.hh>
#include <log4cpp/RollingFileAppender.hh>

OstreamAppender//输出到一个 ostream 类

构造函数：
log4cpp::OstreamAppender::OstreamAppender(const std::string &name,
std::ostream *stream)*
参数：
name the name of the Appender. ===>名称
stream the name of the ostream.===>流名字
例子：
log4cpp::OstreamAppender osAppender = new log4cpp::OstreamAppender("osAppender",
&cout);

StringQueueAppender//内存队列

构造函数：
log4cpp::StringQueueAppender::StringQueueAppender(const std::string &name)
参数：类似上述
举例：
log4cpp::StringQueueAppender* strQAppender =
newlog4cpp::StringQueueAppender("strQAppender");
最常用的两个 FileAppender，其功能是将日志写入文件中
FileAppender // 输出到文件

FileAppender

构造函数

log4cpp::FileAppender::FileAppender (const std::string &name,
const std::string &fileName,
bool append = true,
mode_t mode = 00644)
参数：
name the name of the Appender. ===>名称
fileName the name of the file to which the Appender has to log.===>日志文件名
append whether the Appender has to truncate the file or just append to it if it
already exists. Defaults to 'true'.(===>是否在日志文件后继续记入日志，还是清空原
日志文件再记录)
mode file mode to open the logfile with. Defaults to 00644.
===>文件的打开方式
举例：
log4cpp::FileAppender* fileAppender =
newlog4cpp::FileAppender("fileAppender","wxb.log");
fileAppender->setLayout(pLayout1);
构造函数：

log4cpp::FileAppender::FileAppender (const std::string &name,
int fd)
参数：
name the name of the Appender.
fd the file descriptor to which the Appender has to log.

RollingFileAppender

构造函数
log4cpp::RollingFileAppender::RollingFileAppender(const std::string &name,
const std::string &fileName,
size_t maxFileSize = 10 *1024 *1024,
unsigned int maxBackupIndex = 1,
bool append = true,
mode_t mode = 00644)
参数：
前两个参数与普通文件的类似
第三个参数 maxFileSize：指出了回滚文件的最大值
第四个参数 maxBackupIndex：指出了回滚文件所用的备份文件的最大个数。所谓备份文件，是用来保存回滚文件中因为空间不足未能记录的日志，备份文件的大小仅比回滚文件的最大值大 1kb。所以如果 maxBackupIndex 取值为 3，则回滚文件(假设其名称是 rollwxb.log，大小为 100kb)会有三个备份文件，其名称分别是 rollwxb.log.1，rollwxb.log.2 和 rollwxb.log.3，大小为 101kb。另外要注意：如果 maxBackupIndex 取值为 0 或者小于 0，则回滚文件功能会失效，其表现如同 FileAppender 一样，不会有大小的限制。这也许是一个 bug。
第五第六参数同上。

举例：
log4cpp::RollingFileAppender* rollfileAppender =
newlog4cpp::RollingFileAppender("rollfileAppender","rollwxb.log",5*1024,1);
rollfileAppender->setLayout(pLayout2)
Appender 有个成员函数（方法）：
virtual void log4cpp::Appender::setLayout(Layout *layout)
函数用作：设置 appender 格式
参数：layout The layout to use.

Category:种类，负责向日志中写入信息

涉及头文件

#include<log4cpp/Category.hh>
构造函数：
Log4cpp 中有一个总是可用并实例化好的 Category，即根 Category。使用 log4cpp::Category::getRoot()可以得到根 Category。
在大多数情况下，一个应用程序只需要一个日志种类(Category)，但是有时也会用到多个 Category，此时可以使用根 Category 的 getInstance 方法来得到子 Category。不同的子 Category 用于不同的场合。
log4cpp::Category& root =log4cpp::Category::getRoot();
log4cpp::Category& root =log4cpp::Category::getRoot();
log4cpp::Category& infoCategory =root.getInstance("infoCategory");
infoCategory.addAppender(osAppender);
infoCategory.setPriority(log4cpp::Priority::INFO);

主要的成员函数：

1、设置 category 的级别

void log4cpp::Category::setPriority (Priority::Value priority) throw (std::invalid_argument)
Set the priority of this Category.Parameters:
priority The priority to set. Use Priority::NOTSET to let the category use its parents priority as effective priority.

2、设置或添加 appender

void log4cpp::Category::addAppender (Appender *appender) throw (std::invalid_argument) [virtual]
Adds an Appender to this Category.
This method passes ownership from the caller to the Category.

void log4cpp::Category::setAppender (Appender *appender) [inline]
Adds an Appender to this Category.
This method passes ownership from the caller to the Category.

3、相应日志级别的记录

（emerg/fatal/alter/crit/error/warn/notice/info/debug/notset）
warn
void log4cpp::Category::warn(const std::string &message) throw ()
Log a message with warn priority.
Parameters:
message string to write in the log file
void log4cpp::Category::warn(const char * stringFormat,...) throw ()
Log a message with warn priority.
Parameters:
stringFormat Format specifier for the string to write in the log file.

void log4cpp::Category::error(const std::string &message) throw ()
Log a message with error priority.
Parameters:
message string to write in the log file
void log4cpp::Category::error(const char *stringFormat,...) throw ()
Log a message with error priority.
Parameters:
stringFormat Format specifier for the string to write in the log file.

Priority:优先级，用来指定 Category 优先级和日志优先级

涉及头文件：

#include<log4cpp/Priority.hh>

Log4cpp记录日志的原理

每个Category都有一个优先级，该优先级可以由setPriority
方法设置，或者从其父 Category 中继承而来。每条日志也有一个优先级，当 Category 记录
该条日志时，若日志优先级高于 Category 的优先级时，该日志被记录，否则被忽略。系统
中默认的优先级等级如下：
typedef enum
{EMERG=0,FATAL=0,ALERT=100,CRIT=200,ERROR=300,WARN=400,NOTICE=500,INFO=600,
DEBUG=700,NOTSET =800}PriorityLevel;
注意：取值越小，优先级越高。例如一个 Category 的优先级为 101，则所有 EMERG、FATAL、
ALERT 日志都可以记录下来，而其他则不能。
Log4cpp 的自动内存管理
Log4cpp 的内存对象管理
Log4cpp 中 new 出来的 Category、Appender 和 Layout 都不需要手动释放，因为 Log4cpp 使
用了一个内部类来管理这些对象。此类的名称是 HierarchyMaintainer，它负责管理 Category 的继承
关系，在程序结束时，HierarchyMaintainer 会依次释放所有 Category，而 Category 则会依次释放拥
有的有效 Appender，Appender 则会释放所有附属的 Layout。如果程序员手动释放这些对象，则会造
成内存报错。从下面的代码可以看出这个特征：
appender->setLayout(newlog4cpp::BasicLayout());
这个 new 出来的 BasicLayout 根本就没有保存其指针，所以它只能被 log4cpp 的内存管理类
HierarchyMaintainer 释放。了解到 HierarchyMaintainer 的内存管理方法后，程序员在使用 log4cpp
时应该遵循以下几个使用原则：
1)不要手动释放 Category、Appender 和 Layout;
2)同一个 Appender 不要加入多个 Category，否则它会被释放多次从而导致程序崩溃;
3)同一个 Layout 不要附着到多个 Appender 上，否则也会被释放多次导致程序崩溃;
log4cpp::Category::shutdown()
在不使用 log4cpp 时可调用 log4cpp::Category::shutdown()，其功能如同
HierarchyMaintainer 的内存清理。
但如果不手动调用，在程序结束时 HierarchyMaintainer 会调用 Category 的析构函数来释
放所有 Appender。

利用配置文件定制日志
如同 log4j 一样，log4cpp 也可以读取配置文件来定制 Category、Appender 和 Layout 对象。
其配置文件格式基本类似于 log4j，一个简单的配置文件 log4cpp.ini 例子如下:
#log4cpp 配置文件
#定义 Root category 的属性
log4cpp.rootCategory=DEBUG, RootLog
#定义 RootLog 属性
log4cpp.appender.RootLog=ConsoleAppender
log4cpp.appender.RootLog.layout=PatternLayout
log4cpp.appender.RootLog.layout.ConversionPattern=%d [%p] -%m%n
#定义 sample category 的属性
log4cpp.category.sample=DEBUG, sample
#定义 sample 属性
log4cpp.appender.sample=FileAppender
log4cpp.appender.sample.fileName=sample.log
log4cpp.appender.sample.layout=PatternLayout
log4cpp.appender.sample.layout.ConversionPattern=%d [%p] -%m%n
#定义 sample.soncategory 的属性
log4cpp.category.sample.son=DEBUG, son
#定义 son 的属性
log4cpp.appender.son=FileAppender
log4cpp.appender.son.fileName=son.log
log4cpp.appender.son.layout=PatternLayout

log4cpp.appender.son.layout.ConversionPattern=%d[%p] - %m%n
#定义 sample.daughtercategory 的属性
log4cpp.category.sample.daughter=DEBUG,daughter
#定义 daughter 属性
log4cpp.appender.daughter=FileAppender
log4cpp.appender.daughter.fileName=daughter.log
log4cpp.appender.daughter.layout=PatternLayout
log4cpp.appender.daughter.layout.ConversionPattern=%d [%p]- %m%n
对应 category 和 appender 的配置方式，可以发现
category 是"log4cpp.category." + "categoryname"
category 名字可以用"."分隔，以标识包含关系
appender 是"log4cpp.appender." + "appendername"

appender 名字 不能用 "." 分隔，即是说 appender 是没有包含关系的
读取配置文件要依赖 PropertyConfigurator 和 SimpleConfigurator 类。这里仅介绍 PropertyConfigurator，其使用方法代码 ConfigFileExam 所示(该代码来自《便利的开发工具-log4cpp 快速使用指南》一文)：

```
#include
#include<log4cpp/Category.hh>
#include<log4cpp/PropertyConfigurator.hh>
int main(int argc,char* argv[])
{
try
{
log4cpp::PropertyConfigurator::configure("./log4cpp.conf");
}
catch(log4cpp::ConfigureFailure& f)
{
std::cout<< "Configure Problem "<< f.what()<< std::endl;
return -1;
}
log4cpp::Category& root =log4cpp::Category::getRoot();
log4cpp::Category& sub1 =log4cpp::Category::getInstance(std::string("sub1"));
log4cpp::Category& sub3
=log4cpp::Category::getInstance(std::string("sub1.sub2"));
sub1.info("This is someinfo");
sub1.alert("Awarning");
// sub3 only have A2 appender.
sub3.debug("This debug messagewill fail to write");
sub3.alert("All hands abandonship");
sub3.critStream() <<"This will show up<< as "<< 1 <<" critical
message"<<log4cpp::CategoryStream::ENDLINE;
sub3<<log4cpp::Priority::ERROR<<"And this will be
anerror" <<log4cpp::CategoryStream::ENDLINE;
sub3.log(log4cpp::Priority::WARN, "This will be a logged warning");
return0;
}
```

该程序首先读入了配置文件 log4cpp.conf，从中得到了所有 Category、Appender 和 Layout 的优先级和相互附属关系，然后输出了一些日志，其运行结果如下：

```
1248875649 INFO sub1 : This is some info
1248875649 ALERT sub1 : A warning
The message All hands abandon ship at time 2009-07-2921:54:09,515
```

1248875649 ALERT sub1.sub2 : All hands abandonship
The message This will show up<< as 1 critical message at time2009-07-29 21:54:09,531
1248875649 CRIT sub1.sub2 : This will show up<< as 1 critical message
The message And this will be an error at time 2009-07-2921:54:09,531
1248875649 ERROR sub1.sub2 : And this will be anerror

log4cpp 优点：

•可通过配置文件完成所有配置并动态加载；
•性能优秀，内存占用小，经过编译后的 log4cpp.dll 大小仅有 160kb；
•提供了完整的日志动态优先级控制，可随时调整需要记录的日志优先级；
•代码级的平台无关性，Log4cpp 源代码经过编译后，适用于大多数主流的操作系统和开发工具

学习官网:http://log4cpp.sourceforge.net/