log4cplus

1、日志的重要性

随着计算机技术的进步和发展，计算的重心开始由客户端向服务端转移，出现了诸如面向服务、并行计算、云计算等应用模式，后台服务端开始承担越来越来的工作，运行的应用程序也越来越复杂。而作为后台服务端应用，我们必须通过某种方式对其运行状况进行监控，以便在出现问题时能更好的跟踪和定位。这就需要一个给力的日志系统。可以说日志系统在当前的服务端应用中已成为一个必不可少的重要模块。

2、大名鼎鼎的Log4j

说起日志系统，不得不提大名鼎鼎的Log4j，特别是使用Java的人们，可以说是无人不知无人不晓无人不用。Log4j以其简单的使用方式（引入一个jar包，一行代码即可调用），灵活（可通过配置文件随意配置），功能强大（多个级别，可配置多个输出目的地，Console，File，系统日志，远端的LogServer等等，可订制日志格式，自动产生，删除日志文件）等等等等特性，一直是Java日志系统的首选。

3、Log4j的C++版本Log4cplus

上面说到，日志在现在的系统里必不可少，Java有功能强大的Log4j可以使用，作为最重要变成语言之一的C++有什么选择呢？幸运的是，有一些大牛很早之前就发现了这个问题，他们仿照Log4J，使用C++语言开发了一套日志系统Log4cplus，Log4cplus的目的很明确，打造C++版的Log4j。而且最重要

的是Log4cplus是开源的。下载: http://sourceforge.net/projects/log4cplus/, Log4j是一个C++库，编译以后即可使用。同时log4cplus支持windows和linux, windows下：打开根目录下的msvc10下面的vs工程，编译即可, Linux下：configure, make。

4、使用

C++ 复制代码

```cpp
#include <QCoreApplication>

#include <log4cplus/logger.h>
#include <log4cplus/fileappender.h>
#include <log4cplus/layout.h>
#include <log4cplus/ndc.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/loggingmacros.h>

#include <iostream>
using namespace std;
#pragma comment(lib, "advapi32.lib")

void test001() {
    log4cplus::initialize();

    log4cplus::helpers::LogLog::getLogLog()->setInternalDebugging(true);
    log4cplus::SharedAppenderPtr share_appender_ptr(new
log4cplus::RollingFileAppender(LOG4CPLUS_TEXT("Test.log"), 5 * 1024, 5));
    share_appender_ptr->setName(LOG4CPLUS_TEXT("First"));
    share_appender_ptr->setLayout(std::auto_ptr<log4cplus::Layout>(new
log4cplus::TTCCLayout()) );
    log4cplus::Logger::getRoot().addAppender(share_appender_ptr);

    log4cplus::Logger root = log4cplus::Logger::getRoot();
    log4cplus::Logger test =
log4cplus::Logger::getInstance(LOG4CPLUS_TEXT("test"));
    log4cplus::Logger subTest =
log4cplus::Logger::getInstance(LOG4CPLUS_TEXT("test.subtest"));

    for(int i = 0; i < 20000; ++i) {
        log4cplus::NDCContextCreator _context(LOG4CPLUS_TEXT("loop"));
        LOG4CPLUS_DEBUG(subTest, "Entering loop #" << i);
    }
}

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    test001();

    return a.exec();
}
```

5、主要类说明

类名	说明
Filter	过滤器，过滤输出消息。过滤器，解决哪些信息需要输出的问题，比如DEBUG，WARN，INFO等的输出控制
Layout	布局器，控制输出消息的格式。格式化输出信息,解决了如何输出的问题。
Appender	挂接器，与布局器和过滤器紧密配合，将特定格式的消息过滤后输出到所挂接的设备终端如屏幕，文件等等)。接收日志的各个设备,如控制台、文件、网络等。解决了输出到哪里去的问题
Logger	记录器，保存并跟踪对象日志信息变更的实体，当你需要对一个对象进行记录时，就需要生成一个logger。日志模块,程序中唯一一个必须得使用的模块，解决了在哪里使用日志的问题。
Hierarchy	分类器，层次化的树型结构，用于对被记录信息的分类，层次中每一个节点维护一个logger的所有信息。
LogLevel	优先权，包括TRACE，DEBUG, INFO, WARNING, ERROR, FATAL。

6、基本使用步骤

使用log4cplus有六个基本步骤：

-- 实例化一个封装了输出介质的appender对象；

-- 实例化一个封装了输出格式的layout对象；

-- 将layout对象绑定(attach)到appender对象；如省略此步骤，简单布局器SimpleLayout(参见5.1小节)对象会绑定到logger。

-- 实例化一个封装了日志输出logger对象,并调用其静态函数getInstance()获得实例，log4cplus::Logger::getInstance("logger_name")；

-- 将appender对象绑定(attach)到logger对象；

-- 设置logger的优先级，如省略此步骤，各种有限级的日志都将被输出。

例子1 – 标准和简单使用

C++ 复制代码

```cpp
#include <QCoreApplication>

#include <log4cplus/logger.h>
#include <log4cplus/fileappender.h>
#include <log4cplus/layout.h>
#include <log4cplus/ndc.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/loggingmacros.h>
#include <log4cplus/consoleappender.h>

#include <iostream>
#include <memory>
using namespace std;
#pragma comment(lib, "advapi32.lib")

//标准使用，严格实现步骤1-6
void test001() {
    /* step 1: Instantiate an appender object */
    log4cplus::helpers::SharedObjectPtr<log4cplus::Appender> _append(new log4cplus::ConsoleAppender());
    _append->setName(LOG4CPLUS_TEXT("append for test001"));

    /* step 2: Instantiate a layout object */
    std::auto_ptr<log4cplus::Layout> _layout(new log4cplus::PatternLayout(LOG4CPLUS_TEXT("%d{%m/%d/%y %H:%M:%S} -LHW- %m [%l]%n")));

    /* step 3: Attach the layout object to the appender */
    //_append->setLayout(_layout);
    _append->setLayout(std::auto_ptr<log4cplus::Layout>(new log4cplus::PatternLayout(LOG4CPLUS_TEXT("%d{%Y/%m/%d %H:%M:%S} -LHW- %p [%l] %m%n"))));
    //_append->setLayout(std::unique_ptr<log4cplus::Layout>(new log4cplus::PatternLayout(LOG4CPLUS_TEXT ("%-5p %c <%x> - %m%n"))));

    /* step 4:  Instantiate a logger object */
    log4cplus::Logger _logger = log4cplus::Logger::getInstance(LOG4CPLUS_TEXT("test001"));

    /* step 5: Attach the appender object to the  logger  */
    _logger.addAppender(_append);

    /* step 6: Set a priority for the logger  */
    _logger.setLogLevel(log4cplus::ALL_LOG_LEVEL);

```

```
39      /* log activity */
40      LOG4CPLUS_DEBUG(_logger, "This is the FIRST log message...");
41      LOG4CPLUS_WARN(_logger, "This is the SECOND log message...");
42  }
43
44  //例2-简洁使用 -- 简洁使用，仅实现步骤1、4、5。
45  void test002() {
46      /* step 1: Instantiate an appender object */
47      log4cplus::SharedAppenderPtr _append(new
    log4cplus::ConsoleAppender());
48      _append->setName(LOG4CPLUS_TEXT("append test002"));
49
50      /* step 4: Instantiate a logger object */
51      log4cplus::Logger _logger =
    log4cplus::Logger::getInstance(LOG4CPLUS_TEXT("test002"));
52
53      /* step 5: Attach the appender object to the  logger  */
54      _logger.addAppender(_append);
55
56      /* log activity */
57      LOG4CPLUS_DEBUG(_logger, "This is the FIRST log message...");
58      LOG4CPLUS_WARN(_logger, "This is the SECOND log message...");
59  }
60
61  int main(int argc, char *argv[])
62  {
63      QCoreApplication a(argc, argv);
64
65      test001();
66      test002();
67
68      return a.exec();
69  }
70  /*
71  输出:
72  2022/01/18 08:21:16 -LHW- DEBUG [main.cpp:41] This is the FIRST log
    message...
73  2022/01/18 08:21:16 -LHW- WARN [main.cpp:42] This is the SECOND log
    message...
74  DEBUG - This is the FIRST log message...
75  WARN - This is the SECOND log message...
76  */
```

例子2 – 输出到控制台和文件

C++ 复制代码

```cpp
#include <QCoreApplication>

#include <log4cplus/logger.h>
#include <log4cplus/fileappender.h>
#include <log4cplus/layout.h>
#include <log4cplus/ndc.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/loggingmacros.h>
#include <log4cplus/consoleappender.h>

#include <iostream>
#include <memory>
#include<iomanip>
using namespace std;
#pragma comment(lib, "advapi32.lib")

//输出日志到控制台 -- iostream模式，appender输出到控制台。
void test001() {
    /*step1:Instantiate an appender object*/
    log4cplus::SharedAppenderPtr _append(new log4cplus::ConsoleAppender());
    _append->setName(LOG4CPLUS_TEXT("append test001"));

    /*step4:Instantiate a logger object*/
    log4cplus::Logger _logger = log4cplus::Logger::getInstance(LOG4CPLUS_TEXT("test001"));

    /*step5:Attach the appender object to the logger*/
    _logger.addAppender(_append);

    /*log activity*/
    LOG4CPLUS_TRACE(_logger, "This i s" << "just a t" << "est." << std::endl);
    LOG4CPLUS_DEBUG(_logger, "This is a bool:" << true);
    LOG4CPLUS_INFO(_logger, "This is a char:" << 'x');
    LOG4CPLUS_WARN(_logger, "This is a int:" << 1000);
    LOG4CPLUS_ERROR(_logger, "This is a long(hex):" << std::hex  << 100000000);
    LOG4CPLUS_FATAL(_logger, "This is a double:" <<  std::setprecision(5) << 1.2345234234);
}

//输出日志到文件 -- 文件模式，appender输出到文件
void test002() {
    log4cplus::initialize();                //必须要有
```

```
41
42      /*step1:Instantiate an appender object*/
43      log4cplus::SharedAppenderPtr _append(new
    log4cplus::FileAppender(LOG4CPLUS_TEXT("Test.log")));
44      _append->setName(LOG4CPLUS_TEXT("file log test002"));
45
46      /*step4:Instantiate a logger object*/
47      log4cplus::Logger _logger =
    log4cplus::Logger::getInstance(LOG4CPLUS_TEXT("test.subtestof_filelog"));
48
49      /*step5:Attach the appender object to the logger*/
50      _logger.addAppender(_append);
51
52      /*log activity*/
53      for (int i = 0; i < 5; ++i) {
54          LOG4CPLUS_DEBUG(_logger, "Entering loop#" << i  << " End line#");
55      }
56  }
57
58  int main(int argc, char *argv[])
59  {
60      QCoreApplication a(argc, argv);
61
62      test001();
63      test002();
64
65      return a.exec();
66  }
```

例子3 – 使用loglog输出日志

LogLog类实现了debug, warn, error 函数用于log4cplus运行时显示log4cplus自身的调试、警告或错误信息，是对标准输出的简单封装，它也可以用来进行简单的日志输出。LogLog 同时提供了两个方法来进一步控制所输出的信息，其中setInternalDebugging()方法用来控制是否屏蔽输出信息中的调试信息，当输入参数为false则屏蔽，缺省设置为false。 setQuietMode()方法用来控制是否屏蔽所有输出信息，当输入参数为true则屏蔽，缺省设置为false。

C++ 复制代码

```cpp
#include <QCoreApplication>

#include <log4cplus/logger.h>
#include <log4cplus/fileappender.h>
#include <log4cplus/layout.h>
#include <log4cplus/ndc.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/loggingmacros.h>
#include <log4cplus/consoleappender.h>

#include <iostream>
#include <memory>
#include <iomanip>
using namespace std;
#pragma comment(lib, "advapi32.lib")

// 使用loglog输出日志
void printMsgs() {
    std::cout << "Entering  printMsgs()..." << std::endl;
    log4cplus::helpers::LogLog::getLogLog()->debug(LOG4CPLUS_TEXT("This is a  Debug statement..."));
    log4cplus::helpers::LogLog::getLogLog()->warn(LOG4CPLUS_TEXT("This is a  Warning..."));
    log4cplus::helpers::LogLog::getLogLog()->error(LOG4CPLUS_TEXT("This is a  Error..."));
    std::cout << "Exiting  printMsgs()..." << std::endl << std::endl;
}

void test001() {
    printMsgs();

    std::cout << "Turning on  debug..." << std::endl;
    log4cplus::helpers::LogLog::getLogLog()->setInternalDebugging(true);
    printMsgs();

    std::cout << "Turning on  quiet mode..." << std::endl;
    log4cplus::helpers::LogLog::getLogLog()->setQuietMode(true);
    printMsgs();
}

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    test001();
```

```
43        //test002();
44
45        return a.exec();
46    }
```

7、转换标识符

PatterLayout支持的转换标识符主要包括：

（1）"%%"，转义为%, 即，std::string pattern = "%%" 时输出"%"。

（2）"%c"，输出logger名称，比如std::string pattern ="%c" 时输出: "test_logger.subtest"，　也可以控制logger名称的显示层次，比如"%c{1}"时输出"test_logger"，其中数字表示层次。

（3）"%D"，显示本地时间，当std::string pattern ="%D" 时输出:"2004–10–16 18:55:45"，%d显示标准时间，所以当std::string pattern ="%d" 时输出"2004–10–16 10:55:45" （因为北京时间位于东8区，差8个小时）。

可以通过%d{...}定义更详细的显示格式，比如%d{%H:%M:%s}表示要显示小时:分钟：秒。大括号中可显示的预定义标识符如下：

%a –– 表示礼拜几，英文缩写形式，比如"Fri"

%A –– 表示礼拜几，比如"Friday"

%b –– 表示几月份，英文缩写形式，比如"Oct"

%B –– 表示几月份，"October"

%c –– 标准的日期＋时间格式，如 "Sat Oct 16 18:56:19 2004"

%d –– 表示今天是这个月的几号(1–31)"16"

%H –– 表示当前时刻是几时(0–23)，如 "18"

%I –– 表示当前时刻是几时(1–12)，如 "6"

%j –– 表示今天是哪一天(1–366)，如 "290"

%m –– 表示本月是哪一月(1–12)，如 "10"

%M –– 表示当前时刻是哪一分钟(0–59)，如 "59"

%p –– 表示现在是上午还是下午， AM or PM

%q –– 表示当前时刻中毫秒部分(0–999)，如 "237"

%Q –– 表示当前时刻中带小数的毫秒部分(0–999.999)，如 "430.732"

%S –– 表示当前时刻的多少秒(0–59)，如 "32"

%U –– 表示本周是今年的第几个礼拜，以周日为第一天开始计算(0–53)，如 "41"

%w –– 表示礼拜几，(0–6, 礼拜天为0)，如 "6"

%W –– 表示本周是今年的第几个礼拜，以周一为第一天开始计算(0–53)，如 "41"

%x –– 标准的日期格式，如 "10/16/04"

%X –– 标准的时间格式，如 "19:02:34"

%y –– 两位数的年份(0–99)，如 "04"

%Y -- 四位数的年份，如 "2004"
%Z -- 时区名，比如 "GMT"
（4）"%F"，输出当前记录器所在的文件名称，比如std::string pattern ="%F" 时输出: "main.cpp"。
（5）"%L"，输出当前记录器所在的文件行号，比如std::string pattern ="%L" 时输出: "51"
（6）"%l"，输出当前记录器所在的文件名称和行号，比如std::string pattern ="%l" 时输出"main.cpp:51"。
（7）"%m"，输出原始信息，比如std::string pattern ="%m" 时输出: "teststr"，即上述代码中LOG4CPLUS_DEBUG的第二个参数，这种实现机制可以确保原始信息被嵌入到带格式的信息中。
（8）"%n"，换行符，没什么好解释的。
（9）"%p"，输出LogLevel，比如std::string pattern ="%p" 时输出: "DEBUG"。
（10）"%t"，输出记录器所在的线程ID，比如std::string pattern ="%t" 时输出: "1075298944"。
（11）"%x"，嵌套诊断上下文NDC (nested diagnostic context) 输出，从堆栈中弹出上下文信息，NDC可以用对不同源的log信息（同时地）交叉输出进行区分，关于NDC方面的详细介绍会在下文中提到。
（12）格式对齐，比如std::string pattern ="%-10m"时表示左对齐，宽度是10，此时会输出"teststr "，当然其它的控制字符也可以相同的方式来使用，比如"%-12d"，"%-5p"等等。

8、日志输出宏

log4cplus在头文件loggingmacros.h中提供了以下的日志输出宏，其中logger 为Logger实例名称，logEvent为日志内容。由于log4cplus选用C++的流机制进行日志输出，因此为了区分包含<<运算符和不包含<<运算符的日志内容，分别提供了LOG4CPLUS_XXXX和LOG4CPLUS_XXXX_STR两种日志输出宏。 另外，日志输出宏LOG4CPLUS_TRACE_METHOD主要用来跟踪方法的调用轨迹。

```cpp
LOG4CPLUS_TRACE_METHOD(logger,logEvent)

LOG4CPLUS_TRACE(logger,logEvent)
LOG4CPLUS_TRACE_STR(logger,logEvent)

LOG4CPLUS_DEBUG(logger,logEvent)
LOG4CPLUS_DEBUG_STR(logger,logEvent)

LOG4CPLUS_INFO(logger,logEvent)
LOG4CPLUS_INFO_STR(logger,logEvent)

LOG4CPLUS_WARN(logger,logEvent)
LOG4CPLUS_WARN_STR(logger,logEvent)

LOG4CPLUS_ERROR(logger,logEvent)
LOG4CPLUS_ERROR_STR(logger,logEvent)

LOG4CPLUS_FATAL(logger,logEvent)
LOG4CPLUS_FATAL_STR(logger,logEvent)
```

9、输出格式控制(layout)

log4cplus通过布局器（Layouts）来控制输出的格式，log4cplus提供了三种类型的Layouts，分别是SimpleLayout、PatternLayout、和TTCCLayout。

SimpleLayout

一种简单格式的布局器，在输出的原始信息之前加上LogLevel和一个"-"，如果初始化时没有将布局器附加到挂接器，则默认使用SimpleLayout，以下代码演示了如何使用SimpleLayout。

C++ 复制代码

```cpp
#include <QCoreApplication>

#include <log4cplus/logger.h>
#include <log4cplus/fileappender.h>
#include <log4cplus/layout.h>
#include <log4cplus/ndc.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/loggingmacros.h>
#include <log4cplus/consoleappender.h>

#include <iostream>
#include <memory>
#include <iomanip>
using namespace std;
#pragma comment(lib, "advapi32.lib")

void test001() {
    /* step 1:  Instantiate an appender object */
    log4cplus::helpers::SharedObjectPtr<log4cplus::Appender> _append(new log4cplus::ConsoleAppender());
    _append->setName(LOG4CPLUS_TEXT("append for  test"));

    /* step 2:  Instantiate a layout object */
    //std::auto_ptr<log4cplus::Layout> _layout(new log4cplus::SimpleLayout());

    /* step 3: Attach  the layout object to the appender */
    _append->setLayout(std::auto_ptr<log4cplus::Layout>(new log4cplus::SimpleLayout()));

    /* step 4:  Instantiate a logger object */
    log4cplus::Logger _logger = log4cplus::Logger::getInstance(LOG4CPLUS_TEXT("test"));

    /* step 5: Attach  the appender object to the logger  */
    _logger.addAppender(_append);

    /* log activity */
    LOG4CPLUS_DEBUG(_logger,  "This  is the simple formatted log message...");
}

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
```

```
41
42          test001();
43          //test002();
44
45          return a.exec();
46      }
```

PatternLayout

一种有词法分析功能的模式布局器，类似于C语言的printf()函数，能够对预定义的转换标识符（conversion specifiers）进行解析，转换成特定格式输出。例子《标准和简单使用例子》演示了如何使用PatternLayout

TTCCLayout

是在PatternLayout基础上发展的一种缺省的带格式输出的布局器，其格式由时间，线程ID，Logger和NDC 组成（consists of time, thread, Logger and nested diagnostic context information, hence the name），因而得名, 关于NDC请参见6.4小节。
以下代码片段演示了如何使用TTCCLayout。TTCCLayout在构造时，有机会选择显示本地时间或GMT时间，缺省是按照本地时间显示：TTCCLayout::TTCCLayout(bool use_gmtime = false)。如果需要构造TTCCLayout对象时选择GMT时间格式，则使用方式如下代码片断所示。

C++ 复制代码

```cpp
#include <QCoreApplication>

#include <log4cplus/logger.h>
#include <log4cplus/fileappender.h>
#include <log4cplus/layout.h>
#include <log4cplus/ndc.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/loggingmacros.h>
#include <log4cplus/consoleappender.h>

#include <iostream>
#include <memory>
#include <iomanip>
using namespace std;
#pragma comment(lib, "advapi32.lib")

void test001() {
    /* step 1:  Instantiate an appender object */
    log4cplus::helpers::SharedObjectPtr<log4cplus::Appender> _append(new log4cplus::ConsoleAppender());
    _append->setName(LOG4CPLUS_TEXT("append for  test"));

    /*step2:Instantiatealayoutobject*/
    //std::auto_ptr<log4cplus::Layout> _layout(new log4cplus::TTCCLayout());

    /* step 3: Attach  the layout object to the appender */
    //_append->setLayout(_layout);
    _append->setLayout(std::auto_ptr<log4cplus::Layout>(new log4cplus::TTCCLayout()));

    /* step 4:  Instantiate a logger object */
    log4cplus::Logger _logger = log4cplus::Logger::getInstance(LOG4CPLUS_TEXT("test"));

    /* step 5: Attach  the appender object to the logger  */
    _logger.addAppender(_append);

    /* log activity */
    LOG4CPLUS_DEBUG(_logger,  "This  is the simple formatted log message...");
}

int main(int argc, char *argv[])
{
```

```
41      QCoreApplication a(argc, argv);
42
43      test001();
44      //test002();
45
46      return a.exec();
47  }
48  /*
49  输出:
50  1642507305455 [372] DEBUG test <> - This  is the simple formatted log
    message...
51  */
```

10、重定向到文件

log4cplus提供了三个类用于文件操作，它们是FileAppender类、RollingFileAppender类、DailyRollingFileAppender类。

FileAppender

实现了基本的文件操作功能，构造函数如下：

C++ 复制代码

```
1  FileAppender ::FileAppender(const log4cplus::tstring& filename,
2                              LOG4CPLUS_OPEN_MODE_TYPE mode =
3                              LOG4CPLUS_FSTREAM_NAMESPACE::ios::trunc,
4                              bool immediateFlush = true);
```

filename -- 文件名
mode -- 文件类型，可选择的文件类型包括app、ate、binary、in、out、trunc，因为实际上只是对stl的一个简单包装，这里就不多讲了。缺省是trunc，表示将先前文件删除。
immediateFlush -- 缓冲刷新标志，如果为true表示每向文件写一条记录就刷新一次缓存，否则直到FileAppender被关闭或文件缓存已满才更新文件，一般是要设置true的，比如你往文件写的过程中出现了错误（如程序非正常退出），即使文件没有正常关闭也可以保证程序终止时刻之前的所有记录都会被正常保存。

RollingFileAppender

实现可以滚动转储的文件操作功能，构造函数如下：

```cpp
RollingFileAppender::RollingFileAppender(const log4cplus::tstring&
filename,
                                         long maxFileSize,
                                         int maxBackupIndex,
                                         bool immediateFlush);
```

filename -- 文件名
maxFileSize -- 文件的最大尺寸
maxBackupIndex -- 最大记录文件数
immediateFlush -- 缓冲刷新标志
RollingFileAppender类可以根据你预先设定的大小来决定是否转储，当超过该大小，后续log信息会另存到新文件中，除了定义每个记录文件的大小之外，你还要确定在RollingFileAppender类对象构造时最多需要多少个这样的记录文件(maxBackupIndex+1)，当存储的文件数目超过maxBackupIndex+1时，会删除最早生成的文件，保证整个文件数目等于maxBackupIndex+1。需要指出的是，这里除了Test.log之外，每个文件的大小都是200K，而不是我们想像中的5K，这是因为log4cplus中隐含定义了文件的最小尺寸是200K，只有大于200K的设置才生效，<= 200k的设置都会被认为是200K。

DailyRollingFileAppender

实现根据频度来决定是否转储的文件转储功能，构造函数如下：

```cpp
DailyRollingFileAppender::DailyRollingFileAppender(const
log4cplus::tstring& filename,
                                                   DailyRollingFileSchedule
schedule,
                                                   bool immediateFlush,
                                                   int maxBackupIndex);
```

filename -- 文件名
schedule -- 存储频度
immediateFlush -- 缓冲刷新标志
maxBackupIndex -- 最大记录文件数
DailyRollingFileAppender类可以根据你预先设定的频度来决定是否转储，当超过该频度，后续log信息会另存到新文件中，这里的频度包括：MONTHLY（每月）、WEEKLY（每周）、DAILY（每日）、

TWICE_DAILY（每两天）、HOURLY（每时）、MINUTELY（每分）。maxBackupIndex的含义同上所述，比如下面例子，运行后会以分钟为单位，分别生成名为Test.log.2004-10-17-03-03、Test.log.2004-10-17-03-04和Test.log.2004-10-17-03-05这样的文件。需要指出的是这里的"频度"并不是你写入文件的速度，其实是否转储的标准并不依赖你写入文件的速度，而是依赖于写入的那一时刻是否满足了频度条件，即是否超过了以分钟、小时、周、月为单位的时间刻度，如果超过了就另存。

C++ 复制代码

```cpp
#include <QCoreApplication>

#include <log4cplus/logger.h>
#include <log4cplus/fileappender.h>
#include <log4cplus/layout.h>
#include <log4cplus/ndc.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/loggingmacros.h>
#include <log4cplus/consoleappender.h>

#include <iostream>
#include <memory>
#include <iomanip>
using namespace std;
#pragma comment(lib, "advapi32.lib")

void test001() {
    log4cplus::initialize();

    log4cplus::SharedAppenderPtr _append(new log4cplus::DailyRollingFileAppender(LOG4CPLUS_TEXT("Test.log"), log4cplus::MINUTELY, true, 5));
    _append->setName(LOG4CPLUS_TEXT("First"));

    _append->setLayout(std::auto_ptr<log4cplus::Layout>(new log4cplus::TTCCLayout()));
    log4cplus::Logger::getRoot().addAppender(_append);

    log4cplus::Logger root = log4cplus::Logger::getRoot();
    log4cplus::Logger test = log4cplus::Logger::getInstance(LOG4CPLUS_TEXT("test"));
    log4cplus::Logger subTest = log4cplus::Logger::getInstance(LOG4CPLUS_TEXT("test.subtest"));

    for(int i=0; i < 20000; ++i) {
        log4cplus::NDCContextCreator _context(LOG4CPLUS_TEXT("loop"));
        LOG4CPLUS_DEBUG(subTest, "Entering  loop #"  << i);
    }
}

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    test001();
```

```
41        //test002();
42
43        return a.exec();
44    }
```

11、重定向到远程服务器

log4cplus提供了SocketAppender，实现了C/S方式的日志记录，用于支持重定向到远程服务器。

客户端程序需要做的工作

(1) 定义一个SocketAppender类型的挂接器

SharedAppenderPtr _append(new SocketAppender(host, 8888, "ServerName"));

(2) 把该挂接器加入到logger中

Logger::getRoot().addAppender(_append);

(3) SocketAppender类型不需要Layout, 直接调用宏就可以将信息发往loggerServer了

LOG4CPLUS_INFO(Logger::getRoot(), "This is a test: ")

注意这里对宏的调用其实是调用了SocketAppender::append()，里面有一个数据传输约定，即先发送一个后续数据的总长度，然后再发送实际的数据：

C++ 复制代码

```
1    SocketBuffer  buffer = convertToBuffer(event, serverName);
2    SocketBuffer  msgBuffer(LOG4CPLUS_MAX_MESSAGE_SIZE);
3    msgBuffer.appendSize_t(buffer.getSize());
4    msgBuffer.appendBuffer(buffer);
```

服务器端程序需要做的工作

(1) 定义一个ServerSocket

ServerSocket serverSocket(port);

(2) 调用accept函数创建一个新的socket与客户端连接

Socket sock = serverSocket.accept();

(3) 此后即可用该sock进行数据read/write了,形如(完整代码见6.3.3小节)：

```cpp
SocketBuffer msgSizeBuffer(sizeof(unsigned int));
if(!clientsock.read(msgSizeBuffer)){
    return;
}
unsigned int msgSize = msgSizeBuffer.readInt();
SocketBuffer buffer(msgSize);
if(!clientsock.read(buffer)){
    return;
}
```

(4) 为了将读到的数据正常显示出来，需要将SocketBuffer存放的内容转换成InternalLoggingEvent格式：

log4cplus::spi::InternalLoggingEvent event = readFromBuffer(buffer);

然后输出：

Logger logger = Logger::getInstance(event.getLoggerName());

logger.callAppenders(event);

注意read/write是按照阻塞方式实现的，意味着对其调用直到满足了所接收或发送的个数才返回。

例1 – 重定向到远程服务器

服务器端代码：

C++ 复制代码

```cpp
#include <log4cplus/config.h>
#include <log4cplus/configurator.h>
#include <log4cplus/consoleappender.h>
#include <log4cplus/socketappender.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/helpers/socket.h>
#include <log4cplus/helpers/threads.h>
#include <log4cplus/spi/loggerimpl.h>
#include <log4cplus/spi/loggingevent.h>

#include <iostream>
using namespace std;
using namespace log4cplus;
using namespace log4cplus::helpers;
using namespace log4cplus::thread;

namespace loggingserver {
    class ClientThread : public AbstractThread {
        public:
            ClientThread(Socket clientsock) : clientsock(clientsock) {
                cout << "Received a  client connection!!!!" << endl;
            }

            ~ClientThread() {
                cout << "Client  connection closed." << endl;
            }

            virtual void run();

        private:
            Socket clientsock;
    };
}



int main(int argc, char** argv) {

    if (argc < 3) {
        cout << "Usage: port  config_file" << endl;
        return 1;
    }

    int port = atoi(argv[1]);
    tstring configFile =  LOG4CPLUS_C_STR_TO_TSTRING(argv[2]);
```

```
46      PropertyConfigurator config(configFile);
47      config.configure();
48      ServerSocket serverSocket(port);
49      while (1) {
50          loggingserver::ClientThread* thr = new
    loggingserver::ClientThread(serverSocket.accept());
51          thr->start();
52      }
53
54      return 0;
55  }
56
57  ////////////////////////////////////////////////////////////////////////
    /////
58  //  loggingserver::ClientThread implementation
59  ////////////////////////////////////////////////////////////////////////
    /////
60  void loggingserver::ClientThread::run() {
61      while (1) {
62          if (!clientsock.isOpen()) {
63              return;
64          }
65
66          SocketBuffer msgSizeBuffer(sizeof(unsigned int));
67          if (!clientsock.read(msgSizeBuffer))  {
68              return;
69          }
70
71          unsigned int msgSize = msgSizeBuffer.readInt();
72          SocketBuffer buffer(msgSize);
73          if (!clientsock.read(buffer)) {
74              return;
75          }
76
77          spi::InternalLoggingEvent event = readFromBuffer(buffer);
78          Logger logger = Logger::getInstance(event.getLoggerName());
79          logger.callAppenders(event);
80      }
81  }
```

客户端代码：

C++ 复制代码

```cpp
#include <log4cplus/logger.h>
#include <log4cplus/socketappender.h>
#include <log4cplus/loglevel.h>
#include <log4cplus/tstring.h>
#include <log4cplus/helpers/threads.h>
using namespace log4cplus;

#include <iomanip>
using namespace std;

int main(int argc, char **argv) {
    log4cplus::helpers::sleep(1);
    tstring serverName = (argc > 1 ?  LOG4CPLUS_C_STR_TO_TSTRING(argv[1]) : tstring());
    //tstring host =  LOG4CPLUS_TEXT("192.168.2.10");
    tstring host = LOG4CPLUS_TEXT("127.0.0.1");
    SharedAppenderPtr append_1(new SocketAppender(host, 9998, serverName));
    append_1->setName( LOG4CPLUS_TEXT("First") );
    Logger::getRoot().addAppender(append_1);

    Logger root = Logger::getRoot();
    Logger test = Logger::getInstance(  LOG4CPLUS_TEXT("socket.test") );

    LOG4CPLUS_DEBUG(root,    "This is"
        << " a reall"
        << "y long message." << endl
        << "Just testing it out" << endl
        << "What do you think?")
        test.setLogLevel(NOT_SET_LOG_LEVEL);

    LOG4CPLUS_DEBUG(test, "This is a  bool: "  << true)
        LOG4CPLUS_INFO(test, "This is a  char: "  << 'x')
        LOG4CPLUS_INFO(test, "This is a  short: "  << (short)-100)
        LOG4CPLUS_INFO(test, "This is a  unsigned short: " << (unsigned short)100)
        log4cplus::helpers::sleep(0, 500000);

    LOG4CPLUS_INFO(test, "This is a  int: "  << (int)1000)
        LOG4CPLUS_INFO(test, "This is a  unsigned int: " << (unsigned int)1000)
        LOG4CPLUS_INFO(test, "This is a  long(hex): " << hex << (long)100000000)
        LOG4CPLUS_INFO(test, "This is a  unsigned long: " << (unsigned long)100000000)
```

```
40          LOG4CPLUS_WARN(test, "This is a  float: "  << (float)1.2345)
41          LOG4CPLUS_ERROR(test, "This is a  double: "
42          << setprecision(15)
43          << (double)1.2345234234)
44
45      LOG4CPLUS_FATAL(test, "This is a  long double: "
46          << setprecision(15)
47          << (long double)123452342342.342)
48
49          return 0;
50  }
```

客户端例子 2

C++ 复制代码

```cpp
#include <QCoreApplication>
#include <QtConcurrent/QtConcurrent>
using namespace QtConcurrent;

#include <log4cplus/logger.h>
#include <log4cplus/fileappender.h>
#include <log4cplus/layout.h>
#include <log4cplus/ndc.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/helpers/stringhelper.h>
#include <log4cplus/configurator.h>
#include <log4cplus/loggingmacros.h>
#include <log4cplus/consoleappender.h>
#include <log4cplus/initializer.h>
#include <log4cplus/log4cplus.h>
using namespace log4cplus;
using namespace log4cplus::helpers;
using namespace log4cplus::thread;

#include <iostream>
#include <memory>
#include <iomanip>
using namespace std;
#pragma comment(lib, "advapi32.lib")

void test001() {
    log4cplus::Initializer initializer;
    log4cplus::helpers::LogLog::getLogLog()->setInternalDebugging(true);
    std::this_thread::sleep_for (std::chrono::seconds (1));

    tstring serverName = LOG4CPLUS_TEXT("serverName LHW");
    tstring host = LOG4CPLUS_TEXT("localhost");
    SharedAppenderPtr append_1(new SocketAppender(host, 9998,
serverName));
    append_1->setName(LOG4CPLUS_TEXT("First"));
    Logger::getRoot().addAppender(append_1);

    Logger root = Logger::getRoot();
    Logger test = Logger::getInstance( LOG4CPLUS_TEXT("socket.test") );

    LOG4CPLUS_DEBUG(root, "This is a really long message. Just testing it
out, What do you think?");
    test.setLogLevel(NOT_SET_LOG_LEVEL);
    LOG4CPLUS_DEBUG(test, "This is a bool: " << true);
    LOG4CPLUS_INFO(test, "This is a char: " << 'x');
```

```
44      LOG4CPLUS_INFO(test, "This is a short: " << static_cast<short>
    (-100));
45      LOG4CPLUS_INFO(test, "This is a unsigned short: " <<
    static_cast<unsigned short>(100));
46      std::this_thread::sleep_for (std::chrono::microseconds (500));
47      LOG4CPLUS_INFO(test, "This is a int: " << 1000);
48      LOG4CPLUS_INFO(test, "This is a unsigned int: " << 1000u);
49      LOG4CPLUS_INFO(test, "This is a long(hex): " << hex << 100000000l);
50      LOG4CPLUS_INFO(test, "This is a unsigned long: " << 100000000ul);
51      LOG4CPLUS_WARN(test, "This is a float: " << 1.2345f);
52      LOG4CPLUS_ERROR(test, "This is a double: " << setprecision(5) <<
    1.2345234234);
53      LOG4CPLUS_FATAL(test, "This is a long double: " << setprecision(8) <<
    123452342342.342L);
54
55      NDC& ndc =  log4cplus::getNDC();
56      ndc.pop();
57      for (int i = 0; i < 999999999; i++) {
58          string ndc_str = std::to_string(i);
59          ndc.push(LOG4CPLUS_STRING_TO_TSTRING(ndc_str));
60          std::this_thread::sleep_for (std::chrono::seconds (2));
61          LOG4CPLUS_INFO(test, "i = " << i);
62          //ndc.pop();
63      }
64  }
65
66  int main(int argc, char *argv[])
67  {
68      QCoreApplication a(argc, argv);
69
70      test001();
71
72      return a.exec();
73  }
74  /*
75  服务端接收到数据为:
76  2022/01/20 11:44:51 DEBUG socket.test NDC:serverName LHW [main.cpp:42]
    This is a bool: 1
77  2022/01/20 11:44:51 INFO socket.test NDC:serverName LHW [main.cpp:43]
    This is a char: x
78  2022/01/20 11:44:51 INFO socket.test NDC:serverName LHW [main.cpp:44]
    This is a short: -100
79  2022/01/20 11:44:51 INFO socket.test NDC:serverName LHW [main.cpp:45]
    This is a unsigned short: 100
80  2022/01/20 11:44:51 INFO socket.test NDC:serverName LHW [main.cpp:47]
    This is a int: 1000
81  2022/01/20 11:44:51 INFO socket.test NDC:serverName LHW [main.cpp:48]
    This is a unsigned int: 1000
```

```
2022/01/20 11:44:51 INFO socket.test NDC:serverName LHW [main.cpp:49] This is a long(hex): 5f5e100
2022/01/20 11:44:51 INFO socket.test NDC:serverName LHW [main.cpp:50] This is a unsigned long: 100000000
2022/01/20 11:44:51 WARN socket.test NDC:serverName LHW [main.cpp:51] This is a float: 1.2345
2022/01/20 11:44:51 ERROR socket.test NDC:serverName LHW [main.cpp:52] This is a double: 1.2345
2022/01/20 11:44:51 FATAL socket.test NDC:serverName LHW [main.cpp:53] This is a long double: 1.2345234e+11
2022/01/20 11:44:53 INFO socket.test NDC:serverName LHW - 0 [main.cpp:61] i = 0
2022/01/20 11:44:55 INFO socket.test NDC:serverName LHW - 1 [main.cpp:61]
```

12、嵌入诊断上下文NDC

log4cplus中的嵌入诊断上下文（Nested Diagnostic Context），即NDC。对log系统而言，当输入源可能不止一个，而只有一个输出时，往往需要分辩所要输出消息的来源，比如服务器处理来自不同客户端的消息时就需要作此判断，NDC可以为交错显示的信息打上一个标记(stamp)，使得辨认工作看起来比较容易些。这个标记是线程特有的，利用了线程局部存储机制，称为线程私有数据（Thread-Specific Data，或TSD）。相关定义如下，包括定义、初始化、获取、设置和清除操作：

C++ | 复制代码

```cpp
linux pthread
#define LOG4CPLUS_THREAD_LOCAL_TYPE pthread_key_t*
#define LOG4CPLUS_THREAD_LOCAL_INIT
::log4cplus::thread::createPthreadKey()
#define LOG4CPLUS_GET_THREAD_LOCAL_VALUE( key )
pthread_getspecific(*key)
#defineLOG4CPLUS_SET_THREAD_LOCAL_VALUE(key,value)  \
       pthread_setspecific(*key, value)
#define LOG4CPLUS_THREAD_LOCAL_CLEANUP( key ) pthread_key_delete(*key)

win32
#define LOG4CPLUS_THREAD_LOCAL_TYPE DWORD
#define LOG4CPLUS_THREAD_LOCAL_INIT TlsAlloc()
#define LOG4CPLUS_GET_THREAD_LOCAL_VALUE( key )  TlsGetValue(key)
#define LOG4CPLUS_SET_THREAD_LOCAL_VALUE( key, value ) \
    TlsSetValue(key, static_cast(value))
#define LOG4CPLUS_THREAD_LOCAL_CLEANUP( key )  TlsFree(key)
```

使用起来比较简单，在某个线程中：

```cpp
NDC& ndc =  log4cplus::getNDC();
ndc.push("ur ndc  string");
LOG4CPLUS_DEBUG(logger,  "this  is a NDC test");
... ...
ndc.pop();
... ...
LOG4CPLUS_DEBUG(logger,  "There  should be no NDC...");
ndc.remove();
```

也可以在自定义的输出格式中使用NDC(用%x)，比如：

```cpp
... ...
std::string  pattern = "NDC:[%x]  - %m %n";
std::auto_ptr  _layout(new  PatternLayout(pattern));
... ...
LOG4CPLUS_DEBUG(_logger,  "This  is the FIRST log message...")
NDC& ndc =  log4cplus::getNDC();
ndc.push("ur ndc  string");
LOG4CPLUS_WARN(_logger,  "This  is the SECOND log message...")
ndc.pop();
ndc.remove();
```

另外一种更简单的使用方法是在线程中直接用NDCContextCreator：

NDCContextCreator _first_ndc("ur ndc string");

G4CPLUS_DEBUG(logger, "this is a NDC test")

不必显式地调用push/pop了，而且当出现异常时，能够确保push与pop的调用是匹配的。

13、输出过滤(filter)

利用日志级别进行输出过滤

日志级别管理，log4cplus将输出的log信息按照LogLevel（从低到高）分为：

级别	说明
NOT_SET_LOG_LEVEL (–1)	接受缺省的LogLevel，如果有父logger则继承它的LogLevel
ALL_LOG_LEVEL (0)	开放所有log信息输出
TRACE_LOG_LEVEL (0)	开放trace信息输出(即ALL_LOG_LEVEL)
DEBUG_LOG_LEVEL(10000)	开放debug信息输出
INFO_LOG_LEVEL (20000)	开放info信息输出
WARN_LOG_LEVEL (30000)	开放warning信息输出
ERROR_LOG_LEVEL(40000)	开放error信息输出
FATAL_LOG_LEVEL (50000)	开放fatal信息输出
OFF_LOG_LEVEL (60000)	关闭所有log信息输出

在log4cplus中，所有logger都通过一个层次化的结构（其实内部是hash表）来组织的，有一个Root级别的logger，可以通过以下方法获取：Logger root = Logger::getRoot();
用户定义的logger都有一个名字与之对应，比如：Logger test = Logger::getInstance("test");
可以定义该logger的子logger: Logger subTest = Logger::getInstance("test.subtest");
注意Root级别的logger只有通过getRoot方法获取，Logger::getInstance("root")获得的是它的子对象而已。有了这些具有父子关系的logger之后可分别设置其LogLevel,比如：
root.setLogLevel(...);
Test.setLogLevel(...);
subTest.setLogLevel(...);
各个logger可以通过setLogLevel设置自己的优先级，当某个logger的LogLevel设置成NOT_SET_LOG_LEVEL时，该logger会继承父logger的优先级，另外，如果定义了重名的多个logger，对其中任何一个的修改都会同时改变其它logger。
log4cplus支持编译时候和运行时刻利用日志级别进行输出过滤。编译时刻通过如下的预定义变量进行过滤：

C++ 复制代码

```
#define LOG4CPLUS_DISABLE_FATAL
#define  LOG4CPLUS_DISABLE_WARN
#define  LOG4CPLUS_DISABLE_ERROR
#define  LOG4CPLUS_DISABLE_INFO
#define  LOG4CPLUS_DISABLE_DEBUG
#define  LOG4CPLUS_DISABLE_TRACE
```

运行时刻的过滤则通过使用Logger的setLogLevel设置日志级别进行过滤。

例1 – 日志的优先级

C++ 复制代码

```cpp
#include <QCoreApplication>

#include <log4cplus/logger.h>
#include <log4cplus/fileappender.h>
#include <log4cplus/layout.h>
#include <log4cplus/ndc.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/loggingmacros.h>
#include <log4cplus/consoleappender.h>
using namespace log4cplus;

#include <iostream>
#include <memory>
#include <iomanip>
using namespace std;
#pragma comment(lib, "advapi32.lib")

void test001() {
    SharedAppenderPtr _append(new ConsoleAppender());
    _append->setName(LOG4CPLUS_TEXT("test"));

    Logger::getRoot().addAppender(_append);
    Logger root = Logger::getRoot();

    Logger test = Logger::getInstance(LOG4CPLUS_TEXT("test"));
    Logger subTest = Logger::getInstance(LOG4CPLUS_TEXT("test.subtest"));
    LogLevelManager& llm =  getLogLevelManager();

    cout << endl << "Before  Setting, Default LogLevel" << endl;
    LOG4CPLUS_FATAL(root, "root: " <<
 llm.toString(root.getChainedLogLevel()));
    LOG4CPLUS_FATAL(root, "test: " <<
 llm.toString(test.getChainedLogLevel()));
    LOG4CPLUS_FATAL(root,"test.subtest:" <<
 llm.toString(subTest.getChainedLogLevel()));

    cout << endl << "Setting  test.subtest to WARN" << endl;
    subTest.setLogLevel(WARN_LOG_LEVEL);
    LOG4CPLUS_FATAL(root, "root: " <<
 llm.toString(root.getChainedLogLevel()));
    LOG4CPLUS_FATAL(root, "test: " <<
 llm.toString(test.getChainedLogLevel()));
    LOG4CPLUS_FATAL(root, "test.subtest:  "  <<
llm.toString(subTest.getChainedLogLevel()));

```

```
40      cout << endl << "Setting test  to TRACE" << endl;
41      test.setLogLevel(TRACE_LOG_LEVEL);
42      LOG4CPLUS_FATAL(root, "root: " <<
     llm.toString(root.getChainedLogLevel()));
43      LOG4CPLUS_FATAL(root, "test: " <<
     llm.toString(test.getChainedLogLevel()));
44      LOG4CPLUS_FATAL(root, "test.subtest:  "  <<
    llm.toString(subTest.getChainedLogLevel()));
45
46      cout << endl << "Setting  test.subtest to NO_LEVEL" << endl;
47      subTest.setLogLevel(NOT_SET_LOG_LEVEL);
48      LOG4CPLUS_FATAL(root, "root: " <<
     llm.toString(root.getChainedLogLevel()));
49      LOG4CPLUS_FATAL(root, "test: " <<
    llm.toString(test.getChainedLogLevel()));
50      LOG4CPLUS_FATAL(root, "test.subtest:  "  <<
    llm.toString(subTest.getChainedLogLevel()) << '\n');
51
52      cout << "create a logger test_bak,  named \"test_\", too. " << endl;
53      Logger test_bak = Logger::getInstance(LOG4CPLUS_TEXT("test"));
54      cout << "Setting test to INFO, so  test_bak also be set to INFO" <<
    endl;
55      test.setLogLevel(INFO_LOG_LEVEL);
56      LOG4CPLUS_FATAL(root, "test: " <<
     llm.toString(test.getChainedLogLevel()));
57      LOG4CPLUS_FATAL(root, "test_bak:  "  <<
    llm.toString(test_bak.getChainedLogLevel()));
58  }
59
60  int main(int argc, char *argv[])
61  {
62      QCoreApplication a(argc, argv);
63
64      test001();
65      //test002();
66
67      return a.exec();
68  }
69  /*
70  输出:
71  Before  Setting, Default LogLevel
72  FATAL - root: DEBUG
73  FATAL - test: DEBUG
74  FATAL - test.subtest:DEBUG
75
76  Setting  test.subtest to WARN
77  FATAL - root: DEBUG
78  FATAL - test: DEBUG
```

```
79   FATAL - test.subtest:  WARN
80
81   Setting test  to TRACE
82   FATAL - root: DEBUG
83   FATAL - test: TRACE
84   FATAL - test.subtest:  WARN
85
86   Setting  test.subtest to NO_LEVEL
87   FATAL - root: DEBUG
88   FATAL - test: TRACE
89   FATAL - test.subtest:  TRACE
90
91   create a logger test bak.  named "test ". too.
```

例2 – 运行时利用日志级别进行输出过滤

C++ 复制代码

```cpp
#include <QCoreApplication>

#include <log4cplus/logger.h>
#include <log4cplus/fileappender.h>
#include <log4cplus/layout.h>
#include <log4cplus/ndc.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/loggingmacros.h>
#include <log4cplus/consoleappender.h>
using namespace log4cplus;

#include <iostream>
#include <memory>
#include <iomanip>
using namespace std;
#pragma comment(lib, "advapi32.lib")

void ShowMsg() {
    LOG4CPLUS_TRACE(Logger::getRoot(), "info");
    LOG4CPLUS_DEBUG(Logger::getRoot(), "info");
    LOG4CPLUS_INFO(Logger::getRoot(), "info");
    LOG4CPLUS_WARN(Logger::getRoot(), "info");
    LOG4CPLUS_ERROR(Logger::getRoot(), "info");
    LOG4CPLUS_FATAL(Logger::getRoot(), "info");
}

void test001() {
    SharedAppenderPtr _append(new ConsoleAppender());
    _append->setName(LOG4CPLUS_TEXT("test"));

    _append->setLayout(std::auto_ptr<log4cplus::Layout>(new
log4cplus::TTCCLayout()));
    Logger root = Logger::getRoot();
    root.addAppender(_append);

    cout << endl << "all-log  allowed"  << endl;
    root.setLogLevel(ALL_LOG_LEVEL);
    ShowMsg();

    cout << endl << "trace-log  and above allowed" << endl;
    root.setLogLevel(TRACE_LOG_LEVEL);
    ShowMsg();

    cout << endl << "debug-log  and above allowed" << endl;
    root.setLogLevel(DEBUG_LOG_LEVEL);
```

```
45      ShowMsg();
46
47      cout << endl << "info-log and  above allowed" << endl;
48      root.setLogLevel(INFO_LOG_LEVEL);
49      ShowMsg();
50
51      cout << endl << "warn-log and  above allowed" << endl;
52      root.setLogLevel(WARN_LOG_LEVEL);
53      ShowMsg();
54
55      cout << endl << "error-log  and above allowed" << endl;
56      root.setLogLevel(ERROR_LOG_LEVEL);
57      ShowMsg();
58
59      cout << endl << "fatal-log  and above allowed" << endl;
60      root.setLogLevel(FATAL_LOG_LEVEL);
61      ShowMsg();
62
63      cout << endl << "log  disabled" << endl;
64      root.setLogLevel(OFF_LOG_LEVEL);
65      ShowMsg();
66  }
67
68  int main(int argc, char *argv[])
69  {
70      QCoreApplication a(argc, argv);
71
72      test001();
73      //test002();
74
75      return a.exec();
76  }
77  /*
78  输出:
79  all-log  allowed
80  1642510042733 [15080] TRACE root <> - info
81  1642510042735 [15080] DEBUG root <> - info
82  1642510042735 [15080] INFO root <> - info
83  1642510042736 [15080] WARN root <> - info
84  1642510042737 [15080] ERROR root <> - info
85  1642510042738 [15080] FATAL root <> - info
86
87  trace-log  and above allowed
88  1642510042738 [15080] TRACE root <> - info
89  1642510042739 [15080] DEBUG root <> - info
90  1642510042740 [15080] INFO root <> - info
91  1642510042741 [15080] WARN root <> - info
92  1642510042741 [15080] ERROR root <> - info
```

```
1642510042743 [15080] FATAL root <> - info

debug-log  and above allowed
1642510042743 [15080] DEBUG root <> - info
1642510042744 [15080] INFO root <> - info
1642510042745 [15080] WARN root <> - info
1642510042746 [15080] ERROR root <> - info
1642510042746 [15080] FATAL root <> - info

info-log and  above allowed
1642510042747 [15080] INFO root <> - info
1642510042748 [15080] WARN root <> - info
1642510042748 [15080] ERROR root <> - info
```

利用脚本配置进行输出过滤,由于log4cplus脚本配置中可以设置日志的级别、过滤器Filter，因此它也是进行输出过滤的一种很好的选择。脚本配置的使用具体参见第8节。
LogLog的输出过滤,Loglog可以使用setInternalDebugging()方法用来控制是否屏蔽输出信息中的调试信息，当输入参数为false则屏蔽，缺省设置为false。 另外方法setQuietMode()方法用来控制是否屏蔽所有输出信息，当输入参数为true则屏蔽，缺省设置为false。具体用法请参见4.2.5小节。

14、脚本配置

除了通过程序实现对log环境的配置之外，log4cplus通过PropertyConfigurator类实现了基于脚本配置的功能。通过脚本可以完成对logger、appender和layout的配置，因此可以解决怎样输出，输出到哪里的问题。下面将简单介绍一下脚本的语法规则，包括基本配置语法和高级配置语法。

基本配置

基本配置语法主要针对包括rootLogger和non-root logger。
根Logger的配置语法：log4cplus.rootLogger=[LogLevel], appenderName, appenderName, ...
非根Logger的配置语法：log4cplus.logger.logger_name=[LogLevel|INHERITED], appenderName, appenderName, ...
说明：INHERITED表示继承父Logger的日志级别。
高级配置 – Appender配置
语法：log4cplus.appender.appenderName=fully.qualified.name.of.appender.class
fully.qualified.name.of.appeneder.class可用值：

log4cplus::ConsoleAppender	终端输出
log4cplus::FileAppender	一般文件输出
log4cplus::RollingFileAppender	日志大小输出
log4cplus::DailyRollingFileAppender	日期输出
log4cplus::SocketAppender	网络端口输出

文件通用选项：

选项	作用
ImmediateFlush	是否立即刷新（默认为true）
log4cplus.appender.ALL_MSGS.ImmediateFlush=true	
File	使用的文件名
log4cplus.appender.ALL_MSGS.File=all_msgs.log	
Append	是否追加到之前的文件
log4cplus.appender.ALL_MSGS.Append=true	
ReopenDelay	先将日志缓存起来，等指定时间之后再往文件中插入 减少文件的保存次数
log4cplus.appender.ALL_MSGS.ReopenDelay=10【单位为秒】	
UseLockFile	是否使用加锁的方式去写文件，默认是false
log4cplus.appender.ALL_MSGS.UseLockFile=true	
LockFile	使用的加锁文件名
log4cplus.appender.ALL_MSGS.LockFile=fuck_are_you.lock[文件名没有具体要求]	
Locale	使用的字符集
log4cplus.appender.ALL_MSGS.Locale=chs【en，其他参数具体见imbue参数】	
Threshold	指定日志消息的输出最低层次
log4cplus.appender.ALL_MSGS.Threshold=DEBUG	

DailyRollingFileAppender相关配置：

选项	作用
Schedule	文件保存频率 可选值：MONTHLY，WEEKLY, DAILY, TWICE_DAILY, HOURLY, MINUTELY
log4cplus.appender.ALL_MSGS.Schedule=MINUTELY	
MaxBackupIndex	最多文件个数
log4cplus.appender.ALL_MSGS.MaxBackupIndex=10	
DatePattern	指定文件名的日期格式 1)'.'yyyy–MM: 每月 2)'.'yyyy–ww: 每周 3)'.'yyyy–MM–dd: 每天 4)'.'yyyy–MM–dd–a: 每天两次 5)'.'yyyy–MM–dd–HH: 每小时 6)'.'yyyy–MM–dd–HH–mm: 每分钟
log4cplus.appender.ALL_MSGS.DatePattern='.'yyyy–ww	

RollingFileAppender相关配置：

选项	作用
MaxFileSize	最大文件大小，当小于200kb的时候，默认为200kb，单位有（MB、KB）
log4cplus.appender.ALL_MSGS.MaxFileSize=10	
MaxBackupIndex	最多文件个数
log4cplus.appender.ALL_MSGS.MaxBackupIndex=10	

Filter配置

Appender可以附加Filter组成的链表，如果Filter链中存在过滤器Filter， log4cplus在输出日志之前将调用链表中Filter的过滤方法decide(),根据该方法的返回值决定是否过滤该输出日志。
语法：
log4cplus.appender.appenderName.Filters.FilterNumber=fully.qualified.name.of.Filter.class
log4cplus.appender.appenderName.Filters.FilterNumber.FilterCondition=value.of.FilterCondition
log4cplus.appender.appenderName.Filters.AcceptOnMatch=true|false
举例：
log4cplus.appender.append_1.filters.1=log4cplus::spi::LogLevelMatchFilter
log4cplus.appender.append_1.filters.1.LogLevelToMatch=TRACE
log4cplus.appender.append_1.filters.1.AcceptOnMatch=true

目前log4plus提供的过滤器包括DenyAllFilter 、LogLevelMatchFilter、LogLevelRangeFilter、和StringMatchFilter。
LogLevelMatchFilter根据特定的日志级别进行过滤。
过滤条件包括LogLevelToMatch和AcceptOnMatch（true|false）， 只有当日志的LogLevel值与LogLevelToMatch相同，且AcceptOnMatch为true时才会匹配。
LogLevelRangeFilter根据根据日志级别的范围进行过滤。
过滤条件包括LogLevelMin、LogLevelMax和AcceptOnMatch，只有当日志的LogLevel在LogLevelMin、LogLevelMax之间同时AcceptOnMatch为true时才会匹配。
StringMatchFilter根据日志内容是否包含特定字符串进行过滤。
过滤条件包括StringToMatch和AcceptOnMatch，只有当日志包含StringToMatch字符串 且AcceptOnMatch为true时会匹配。
DenyAllFilter则过滤掉所有消息。
过滤条件处理机制类似于Linux中IPTABLE的Responsibility chain机制，（即先deny、再allow）不过执行顺序刚好相反，后写的条件会被先执行，比如：
log4cplus.appender.append_1.filters.1=log4cplus::spi::LogLevelMatchFilter
log4cplus.appender.append_1.filters.1.LogLevelToMatch=TRACE
log4cplus.appender.append_1.filters.1.AcceptOnMatch=true
#log4cplus.appender.append_1.filters.2=log4cplus::spi::DenyAllFilter
会首先执行filters.2的过滤条件，关闭所有过滤器，然后执行filters.1，仅匹配TRACE信息。

Layout配置

可以选择不设置、TTCCLayout、或PatternLayout，如果不设置，会输出SimpleLayout格式的日志。
设置TTCCLayout的语法：log4cplus.appender.ALL_MSGS.layout=log4cplus::TTCCLayout
设置PatternLayout的语法：log4cplus.appender.append_1.layout=log4cplus::PatternLayout
举例：log4cplus.appender.append_1.layout.ConversionPattern=%d{%m/%d/%y %H:%M:%S,%Q} [%t] %-5p – %m%n

例1 – 脚本配置

脚本方式使用起来非常简单，只要首先加载配置即可（urconfig.properties是自行定义的配置文件）：

PropertyConfigurator::doConfigure("urconfig.properties");

下面通过例子体会一下log4cplus强大的基于脚本过滤log信息的功能。以下是urconfig.properties示例脚本配置内容:

C++ 复制代码

```
log4cplus.rootLogger=TRACE,  ALL_MSGS, TRACE_MSGS, DEBUG_INFO_MSGS, FATAL_MSGS
log4cplus.appender.ALL_MSGS=log4cplus::RollingFileAppender
log4cplus.appender.ALL_MSGS.File=all_msgs.log
log4cplus.appender.ALL_MSGS.layout=log4cplus::TTCCLayout

log4cplus.appender.TRACE_MSGS=log4cplus::RollingFileAppender
log4cplus.appender.TRACE_MSGS.File=trace_msgs.log
log4cplus.appender.TRACE_MSGS.layout=log4cplus::TTCCLayout
log4cplus.appender.TRACE_MSGS.filters.1=log4cplus::spi::LogLevelMatchFilter
log4cplus.appender.TRACE_MSGS.filters.1.LogLevelToMatch=TRACE
log4cplus.appender.TRACE_MSGS.filters.1.AcceptOnMatch=true
log4cplus.appender.TRACE_MSGS.filters.2=log4cplus::spi::DenyAllFilter

log4cplus.appender.DEBUG_INFO_MSGS=log4cplus::RollingFileAppender
log4cplus.appender.DEBUG_INFO_MSGS.File=debug_info_msgs.log
log4cplus.appender.DEBUG_INFO_MSGS.layout=log4cplus::TTCCLayout
log4cplus.appender.DEBUG_INFO_MSGS.filters.1=log4cplus::spi::LogLevelRangeFilter
log4cplus.appender.DEBUG_INFO_MSGS.filters.1.LogLevelMin=DEBUG
log4cplus.appender.DEBUG_INFO_MSGS.filters.1.LogLevelMax=INFO
log4cplus.appender.DEBUG_INFO_MSGS.filters.1.AcceptOnMatch=true
log4cplus.appender.DEBUG_INFO_MSGS.filters.2=log4cplus::spi::DenyAllFilter

log4cplus.appender.FATAL_MSGS=log4cplus::RollingFileAppender
log4cplus.appender.FATAL_MSGS.File=fatal_msgs.log
log4cplus.appender.FATAL_MSGS.layout=log4cplus::TTCCLayout
log4cplus.appender.FATAL_MSGS.filters.1=log4cplus::spi::StringMatchFilter
log4cplus.appender.FATAL_MSGS.filters.1.StringToMatch=FATAL
log4cplus.appender.FATAL_MSGS.filters.1.AcceptOnMatch=true
log4cplus.appender.FATAL_MSGS.filters.2=log4cplus::spi::DenyAllFilter
```

C++ 复制代码

```cpp
#include <QCoreApplication>

#include <log4cplus/logger.h>
#include <log4cplus/fileappender.h>
#include <log4cplus/layout.h>
#include <log4cplus/ndc.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/helpers/stringhelper.h>
#include <log4cplus/configurator.h>
#include <log4cplus/loggingmacros.h>
#include <log4cplus/consoleappender.h>
using namespace log4cplus;
using namespace log4cplus::helpers;
using namespace log4cplus::thread;

#include <iostream>
#include <memory>
#include <iomanip>
using namespace std;
#pragma comment(lib, "advapi32.lib")

static Logger logger = Logger::getInstance(LOG4CPLUS_TEXT("log"));

void printDebug() {
    LOG4CPLUS_TRACE_METHOD(logger, LOG4CPLUS_TEXT("::printDebug()"));
    LOG4CPLUS_DEBUG(logger, "This is a DEBUG message");
    LOG4CPLUS_INFO(logger, "This is a INFO message");
    LOG4CPLUS_WARN(logger, "This is a WARN message");
    LOG4CPLUS_ERROR(logger, "This is a ERROR message");
    LOG4CPLUS_FATAL(logger, "This is a FATAL message");
}

void test001() {
    Logger root = Logger::getRoot();

 PropertyConfigurator::doConfigure(LOG4CPLUS_TEXT("urconfig.properties"))
;
    printDebug();
}

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    test001();
```

```
    //test002();

    return a.exec();
}
/*
输出:
1. all_msgs.log
10-17-04  14:55:25,858 [1075298944] TRACE log <> - ENTER: ::printDebug()
10-17-04  14:55:25,871 [1075298944] DEBUG log <> - This is a DEBUG
message
10-17-04  14:55:25,873 [1075298944] INFO log <> - This is a INFO message
10-17-04  14:55:25,873 [1075298944] WARN log <> - This is a WARN message
10-17-04  14:55:25,874  [1075298944] ERROR log <> - This is a ERROR
message
10-17-04  14:55:25,874 [1075298944] FATAL log <> - This is a FATAL
message
10-17-04  14:55:25,875 [1075298944] TRACE log <> - EXIT:::printDebug()

2. trace_msgs.log
10-17-04  14:55:25,858 [1075298944] TRACE log <> - ENTER: ::printDebug()
10-17-04  14:55:25,875 [1075298944] TRACE log <> - EXIT:::printDebug()

3. debug_info_msgs.log
10-17-04  14:55:25,871 [1075298944] DEBUG log <> - This is a DEBUG
message
10-17-04  14:55:25,873 [1075298944] INFO log <> - This is a INFO message

4. fatal_msgs.log
10-17-04  14:55:25,874 [1075298944] FATAL log <> - This is a FATAL
message
*/
```

15、脚本配置的动态加载

多线程版本的log4cplus提供了实用类ConfigureAndWatchThread，该类启动线程对配置脚本进行监控，一旦发现配置脚本被更新则立刻重新加载配置。

类ConfigureAndWatchThread的构造函数定义为：

ConfigureAndWatchThread(const log4cplus::tstring& propertyFile, unsigned int millis = 60 * 1000);

第一个参数propertyFile为配置脚本的路径名，第二个参数为监控时两次更新检查相隔的时间，单位为耗秒ms。

例1 – 使用线程监控脚本的更新

配置文件:

C++ 复制代码

```
log4cplus.rootLogger=INFO, STDOUT, R
log4cplus.logger.test=WARN
log4cplus.logger.test.log_1=FATAL
log4cplus.logger.test.log_2=FATAL
log4cplus.logger.test.log_3=WARN

log4cplus.appender.STDOUT=log4cplus::ConsoleAppender
log4cplus.appender.STDOUT.layout=log4cplus::PatternLayout
log4cplus.appender.STDOUT.layout.ConversionPattern=%d{%m/%d/%y  %H:%M:%S} [%t] %-5p %c{2} %%%x%% - %m [%l]%n

log4cplus.appender.R=log4cplus::RollingFileAppender
log4cplus.appender.R.File=output.log
#log4cplus.appender.R.MaxFileSize=5MB
log4cplus.appender.R.MaxFileSize=500KB
log4cplus.appender.R.MaxBackupIndex=5
log4cplus.appender.R.layout=log4cplus::TTCCLayout
```

C++ 复制代码

```cpp
#include <QCoreApplication>

#include <log4cplus/logger.h>
#include <log4cplus/fileappender.h>
#include <log4cplus/layout.h>
#include <log4cplus/ndc.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/helpers/stringhelper.h>
#include <log4cplus/configurator.h>
#include <log4cplus/loggingmacros.h>
#include <log4cplus/consoleappender.h>
using namespace log4cplus;
using namespace log4cplus::helpers;
using namespace log4cplus::thread;

#include <iostream>
#include <memory>
#include <iomanip>
using namespace std;
#pragma comment(lib, "advapi32.lib")

Logger log_1  =  Logger::getInstance(LOG4CPLUS_TEXT("test.log_1"));
Logger log_2  =  Logger::getInstance(LOG4CPLUS_TEXT("test.log_2"));
Logger log_3  =  Logger::getInstance(LOG4CPLUS_TEXT("test.log_3"));

void printMsgs(Logger& logger) {
    //LOG4CPLUS_TRACE_METHOD(logger, "printMsgs()");
    LOG4CPLUS_TRACE(logger, "printMsgs()");
    LOG4CPLUS_DEBUG(logger, "printMsgs()");
    LOG4CPLUS_INFO(logger, "printMsgs()");
    LOG4CPLUS_WARN(logger, "printMsgs()");
    LOG4CPLUS_ERROR(logger, "printMsgs()");
}

void test001() {
    cout << "Entering main()..." << endl;

    LogLog::getLogLog()->setInternalDebugging(true);
    Logger root = Logger::getRoot();
    try {
        ConfigureAndWatchThread
configureThread(LOG4CPLUS_TEXT("log4cplus.properties"), 1 * 1000);
        LOG4CPLUS_WARN(root, "Testing....");
        for(int i = 0; i < 999999999999; i++) {
            printMsgs(log_1);
```

```
            printMsgs(log_2);
            printMsgs(log_3);
        }
    } catch(...) {
        cout << "Exception..." << endl;
        LOG4CPLUS_FATAL(root, "Exception  occured...");
    }

    cout << "Exiting main()..." << endl;
}

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    test001();
    //test002();

    return a.exec();
}
/*
程序运行过程中动态修改配置中：
log4cplus.logger.test.log_1=FATAL
log4cplus.logger.test.log_2=FATAL
级别为DEBUG等，可以看到控制台输出变化
*/
```

16、定制Log4cplus

定制日志级别

log4cplus支持日志级别的定制。如果需要定义自己的优先级，则可以按以下步骤进行定制。

定义新日志级别对应的常量整数和输出宏: customloglevel.h。

C++ 复制代码

```cpp
#include <log4cplus/logger.h>
#include <log4cplus/helpers/loglog.h>

using namespace log4cplus;
using namespace log4cplus::helpers;

const LogLevel CRITICAL_LOG_LEVEL = 45000;

#define LOG4CPLUS_CRITICAL(logger, logEvent) \
    if(logger.isEnabledFor(CRITICAL_LOG_LEVEL)) { \
        log4cplus::tostringstream _log4cplus_buf; \
        _log4cplus_buf << logEvent; \
        logger.forcedLog(CRITICAL_LOG_LEVEL, _log4cplus_buf.str(), __FILE__, __LINE__); \
    }

void initializeCriticalLogLevel ();
```

定义新日志级别对应的字符串、常量整数与字符串之间的转换函数，定义自己的初始化器将转换函数注册到LogLevelManage:customloglevel.cpp。

C++ 复制代码

```cpp
#include "customloglevel.h"

static log4cplus::tstring const CRITICAL_STRING
(LOG4CPLUS_TEXT("CRITICAL"));
static log4cplus::tstring const empty_str;


static
tstring const &
criticalToStringMethod(LogLevel ll)
{
    if(ll == CRITICAL_LOG_LEVEL) {
        return CRITICAL_STRING;
    }
    else {
        return empty_str;
    }
}


static
LogLevel
criticalFromStringMethod(const tstring& s)
{
    if(s == CRITICAL_STRING) return CRITICAL_LOG_LEVEL;

    return NOT_SET_LOG_LEVEL;
}



void initializeCriticalLogLevel ()
{
    getLogLevelManager().pushToStringMethod(criticalToStringMethod);
    getLogLevelManager().pushFromStringMethod(criticalFromStringMethod);
}

void
writeLogMessage()
{
    {
        Logger subTest =
Logger::getInstance(LOG4CPLUS_TEXT("test.subtest"));
        subTest.log(FATAL_LOG_LEVEL, LOG4CPLUS_TEXT("Entering
writeLogMessage()..."));
```

```
43          LOG4CPLUS_CRITICAL(subTest,
44              LOG4CPLUS_TEXT("writeLogMessage()- This is a message from a
    different file"));
45          subTest.log(FATAL_LOG_LEVEL, LOG4CPLUS_TEXT("Exiting
    writeLogMessage()..."));
46      }
47      LogLog::getLogLog()->warn(LOG4CPLUS_TEXT("REALLY exiting
    writeLogMessage()..."));
48  }
```

完整例子

C++ 复制代码

```cpp
#include <QCoreApplication>

#include <log4cplus/logger.h>
#include <log4cplus/fileappender.h>
#include <log4cplus/layout.h>
#include <log4cplus/ndc.h>
#include <log4cplus/helpers/loglog.h>
#include <log4cplus/helpers/stringhelper.h>
#include <log4cplus/configurator.h>
#include <log4cplus/loggingmacros.h>
#include <log4cplus/consoleappender.h>
#include <log4cplus/initializer.h>
using namespace log4cplus;
using namespace log4cplus::helpers;
using namespace log4cplus::thread;

#include <iostream>
#include <memory>
#include <iomanip>
using namespace std;
#pragma comment(lib, "advapi32.lib")

const LogLevel CRITICAL_LOG_LEVEL = 45000;

#define LOG4CPLUS_CRITICAL(logger, logEvent) \
    if(logger.isEnabledFor(CRITICAL_LOG_LEVEL)) { \
        log4cplus::tostringstream _log4cplus_buf; \
        _log4cplus_buf << logEvent; \
        logger.forcedLog(CRITICAL_LOG_LEVEL, _log4cplus_buf.str(), __FILE__, __LINE__); \
    }

static log4cplus::tstring const CRITICAL_STRING (LOG4CPLUS_TEXT("CRITICAL"));
static log4cplus::tstring const empty_str;

static tstring const & criticalToStringMethod(LogLevel ll) {
    if (ll == CRITICAL_LOG_LEVEL) {
        return CRITICAL_STRING;
    }
    else {
        return empty_str;
    }
}

```

```
44  static LogLevel criticalFromStringMethod(const tstring& s) {
45      if (s == CRITICAL_STRING) {
46          return CRITICAL_LOG_LEVEL;
47      }
48      return NOT_SET_LOG_LEVEL;
49  }
50
51  void initializeCriticalLogLevel () {
52      getLogLevelManager().pushToStringMethod(criticalToStringMethod);
53      getLogLevelManager().pushFromStringMethod(criticalFromStringMethod);
54  }
55
56  void writeLogMessage()  {
57      Logger subTest =
    Logger::getInstance(LOG4CPLUS_TEXT("test.subtest"));
58      subTest.log(FATAL_LOG_LEVEL, LOG4CPLUS_TEXT("Entering
    writeLogMessage()..."));
59      LOG4CPLUS_CRITICAL(subTest, LOG4CPLUS_TEXT("writeLogMessage()- This
    is a message from a different file"));
60      subTest.log(FATAL_LOG_LEVEL, LOG4CPLUS_TEXT("Exiting
    writeLogMessage()..."));
61      LogLog::getLogLog()->warn(LOG4CPLUS_TEXT("REALLY exiting
    writeLogMessage()..."));
62  }
63
64  void test001() {
65      cout << "Entering main()..." << endl;
66      log4cplus::Initializer initializer;
67      initializeCriticalLogLevel();
68      {
69          log4cplus::initialize();
70          SharedAppenderPtr append_1(new ConsoleAppender());
71          append_1->setName(LOG4CPLUS_TEXT("First"));
72          // append_1->setLayout( std::unique_ptr<Layout>(new
    TTCCLayout()) );
73          cout << "Getting root logger...DONE" << endl;
74          Logger::getRoot().addAppender(append_1);
75
76          Logger root = Logger::getRoot();
77          Logger test = Logger::getInstance(LOG4CPLUS_TEXT("test"));
78          Logger subTest =
    Logger::getInstance(LOG4CPLUS_TEXT("test.subtest"));
79          LogLevelManager& llm = getLogLevelManager();
80
81          LOG4CPLUS_FATAL(root, "root: " <<
    llm.toString(root.getChainedLogLevel()));
82          LOG4CPLUS_FATAL(root, "test: " <<
    llm.toString(test.getChainedLogLevel()));
```

```
83          LOG4CPLUS_FATAL(root, "test.subtest: " <<
    llm.toString(subTest.getChainedLogLevel()));
84          cout << endl;
85
86          LOG4CPLUS_FATAL(root, "Setting test.subtest to WARN");
87          subTest.setLogLevel(WARN_LOG_LEVEL);
88          LOG4CPLUS_FATAL(root, "root: " <<
    llm.toString(root.getChainedLogLevel()));
89          LOG4CPLUS_FATAL(root, "test: " <<
    llm.toString(test.getChainedLogLevel()));
90          LOG4CPLUS_FATAL(root, "test.subtest: " <<
    llm.toString(subTest.getChainedLogLevel()));
91          cout << endl;
```

17、log4cplus server client模式

在多进程使用log4cplus同时向一个日志文件写的时候，官方的FAQ建议使用SoskcetAppender, 即以server client模式来写日志，保证写日志同步。写了一个小程序，fork出一个server进程，和5个client进程来写日志，当日志大小到达4G的时候做切割。

C++ 复制代码

```cpp
/*
 *
 ========================================================================
 ==========
 *
 *       Filename:  multiprocesslog.cpp
 *
 *    Description:
 *
 *        Version:  1.0
 *        Created:  11/15/2011 02:48:44 PM
 *       Revision:  none
 *       Compiler:  gcc
 *
 *         Author:  Xu Zhe Ming,
 *        Company:
 *
 *
 ========================================================================
 ==========
 */

#include <log4cplus/fileappender.h>
#include <log4cplus/socketappender.h>
#include <log4cplus/consoleappender.h>
#include <log4cplus/layout.h>
#include <time.h>
#include <log4cplus/configurator.h>
#include <iomanip>
#include <log4cplus/logger.h>
#include <log4cplus/loglevel.h>
#include <log4cplus/helpers/socket.h>
#include <stdio.h>
#include <string>

using namespace log4cplus;
using namespace std;
using namespace log4cplus::helpers;

//server端
int log_server(int port)
{
    //创建一个socket，绑定端口
    helpers::ServerSocket serverSocket(port);
    Logger debug_logger;
```

```
42      Logger info_logger;
43
44      //注意int会溢出
45      long log_size = long(1024) * 1024 * 1024 * 4;
46
47      //初始化一个debug logger，并绑定到一个文件
48      {
49      SharedAppenderPtr pFileAppender_normal(new
    RollingFileAppender(("log_file_debug") , log_size, 10));
50      debug_logger = Logger::getInstance("debug_logger");
51      debug_logger.addAppender(pFileAppender_normal);
52      }
53
54      //初始化一个info logger，并绑定到一个文件
55      {
56      SharedAppenderPtr pFileAppender_normal(new
    RollingFileAppender(("log_file_info") , log_size, 10));
57      info_logger = Logger::getInstance("info_logger");
58      info_logger.addAppender(pFileAppender_normal);
59      }
60
61      //初始化其他的logger，如warn, error, fatal
62      //...
63
64      while(1)
65      {
66      //accept
67      helpers::Socket clientsock = serverSocket.accept();
68      SocketBuffer msgSizeBuffer(sizeof(unsigned int));
69      if(!clientsock.read(msgSizeBuffer))
70      {
71          return 0;
72      }
73      unsigned int msgSize = msgSizeBuffer.readInt();
74      SocketBuffer buffer(msgSize);
75      //读取日志消息到buffer
76      if(!clientsock.read(buffer))
77      {
78          return 0;
79      }
80
81      //转化成event
82      spi::InternalLoggingEvent event = readFromBuffer(buffer);
83      int level = event.getLogLevel();
84      //判断日志的level
85      if(level == DEBUG_LOG_LEVEL)
86      {
87          debug_logger.callAppenders(event);
```

```
88          }
89          else if(level == INFO_LOG_LEVEL)
90          {
91              info_logger.callAppenders(event);
92          }
93          //...
94          }
95
96          return 0;
97      }
98
99      //client端
100     int log_client(string server_ip, int server_port)
```

18、配置文件示例

1

Shell | 复制代码

```
##############################################################
# log4cplus global configuration attributes                  #
##############################################################
#log4cplus.configDebug=false
#log4cplus.quietMode=false
#log4cplus.disableOverride=false


##############################################################
# the root logger configuration                              #
##############################################################
#Tokenize=loglevel,Appenders
#INHERITED、OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE, ALL, CRITICAL
#log4cplus.rootLogger=TRACE
log4cplus.rootLogger=TRACE,ALogger


##############################################################
# non-root loggers configuration                             #
##############################################################
#INHERITED
#log4cplus.logger.non_rootLogger=INHERITED
#log4cplus.logger.non_rootLogger=TRACE
#log4cplus.logger.non_rootLogger=TRACE,ALogger
#log4cplus.additivity.non_rootLogger=true


##############################################################
# Appender configuration                                     #
##############################################################
#prefix log4cplus::
#ConsoleAppender、NullAppender、
#FileAppender、RollingFileAppender、DailyRollingFileAppender(File)、
TimeBasedRollingFileAppender
#SocketAppender
#NTEventLogAppender、Win32ConsoleAppender、Win32DebugAppender
#SysLogAppender、AsyncAppender(SINGLE_THREADED)
#Log4jUdpAppender
log4cplus.appender.ALogger=log4cplus::DailyRollingFileAppender
#
# Appender common configuration                          #
##############################################################
#log4cplus.appender.ALogger.Threshold=ALL
#log4cplus.appender.ALogger.UseLockFile=false
#LockFile的值默认为File的值后加.lock
```

```
#log4cplus.appender.ALogger.LockFile=
#log4cplus.appender.ALogger.AsyncAppend=false
#
# ConsoleAppender special configuration          #
##################################################
#log4cplus.appender.ALogger.logToStdErr=false
#log4cplus.appender.ALogger.ImmediateFlush=false
#
# FileAppender common configuration              #
##################################################
log4cplus.appender.ALogger.File=.\log\All.log
#log4cplus.appender.ALogger.ImmediateFlush=true
log4cplus.appender.ALogger.CreateDirs=true
log4cplus.appender.ALogger.Append=true
#log4cplus.appender.ALogger.ReopenDelay=1
#log4cplus.appender.ALogger.BufferSize=0
#GLOBAL DEFAULT USER CLASSIC
#log4cplus.appender.ALogger.Locale=DEFAULT
#
# RollingFileAppender special configuration      #
##################################################
# MB KB 200K(min) 10MB(def)
#log4cplus.appender.ALogger.MaxFileSize=10MB
#log4cplus.appender.ALogger.MaxBackupIndex=1
#
# DailyRollingFileAppender special configuration #
##################################################
# MONTHLY、WEEKLY、DAILY、TWICE_DAILY、HOURLY、MINUTELY
log4cplus.appender.ALogger.Schedule=DAILY
log4cplus.appender.ALogger.MaxBackupIndex=10
#
# TimeBasedRollingFileAppender special configuration #
##################################################
#log4cplus.appender.ALogger.FilenamePattern=%d.log
#log4cplus.appender.ALogger.MaxHistory=10
#log4cplus.appender.ALogger.CleanHistoryOnStart=false
#log4cplus.appender.ALogger.RollOnClose=false
#
# SocketAppender special configuration           #
##################################################
#log4cplus.appender.ALogger.host=
#log4cplus.appender.ALogger.port=9998
#log4cplus.appender.ALogger.ServerName=
#log4cplus.appender.ALogger.IPv6=false
#
# NTEventLogAppender special configuration       #
##################################################
#log4cplus.appender.ALogger.server=
```

```
#-10l<=x<<==101
#log4cplus.appender.ALogger.log=Application
#log4cplus.appender.ALogger.source=
#
# Win32ConsoleAppender special configuration     #
#################################################
#log4cplus.appender.ALogger.AllocConsole=true
#log4cplus.appender.ALogger.logToStdErr=false
#log4cplus.appender.ALogger.TextColor=0
#
# SysLogAppender special configuration           #
#################################################
#整数
```

包装类

C++ 复制代码

```cpp
#pragma once


#ifdef CRUISE_LOG4CPLUS

#include <log4cplus/logger.h>
#include <log4cplus/helpers/snprintf.h>
#include <log4cplus/configurator.h>
#include <log4cplus/loggingmacros.h>

class cruise_log4cplus
{
private:
    cruise_log4cplus(){}
    cruise_log4cplus(const cruise_log4cplus& obj)
        : m_logger(obj.m_logger){}
    cruise_log4cplus& operator=(const cruise_log4cplus& obj)
        { m_logger = obj.m_logger; }
    //virtual ~cruise_log4cplus(){}

private:
#define DO_LOGGER(log_level/*, file, line, function, format_msg, __VA_ARGS__*/)    \
    int retval = -1;
 \
    try
 \
    {
 \
        if (m_logger.isEnabledFor(log_level))
 \
        {
 \
            const log4cplus::tchar * msg = NULL;
 \
            log4cplus::helpers::snprintf_buf buf;
 \
            std::va_list args;
 \
            do
 \
            {
 \
                va_start(args, format_msg);
 \
```

```
34                  retval = buf.print_va_list(msg, format_msg, args);
     \
35                  va_end(args);
     \
36              } while (retval == -1);
     \
37              m_logger.forcedLog(log_level, msg, file, line, function);
     \
38          }
     \
39          retval = 0;
     \
40        }
     \
41        catch (...) {}

42
43    public:
44        void Fatal(const char* file, const int line, const char* function,
45            const log4cplus::tchar* format_msg, ...)
46        {   DO_LOGGER(log4cplus::FATAL_LOG_LEVEL);   }
47
48        void Error(const char* file, const int line, const char* function,
49            const log4cplus::tchar* format_msg, ...)
50        {   DO_LOGGER(log4cplus::ERROR_LOG_LEVEL);   }
51
52        void Warn(const char* file, const int line, const char* function,
53            const log4cplus::tchar* format_msg, ...)
54        {   DO_LOGGER(log4cplus::WARN_LOG_LEVEL);   }
55
56        void Info(const char* file, const int line, const char* function,
57            const log4cplus::tchar* format_msg, ...)
58        {   DO_LOGGER(log4cplus::INFO_LOG_LEVEL);   }
59
60        void Debug(const char* file, const int line, const char* function,
61            const log4cplus::tchar* format_msg, ...)
62        {   DO_LOGGER(log4cplus::DEBUG_LOG_LEVEL);   }
63
64        void Trace(const char* file, const int line, const char* function,
65            const log4cplus::tchar* format_msg, ...)
66        {   DO_LOGGER(log4cplus::TRACE_LOG_LEVEL);   }
67
68    public:
69        void log_init(log4cplus::tchar* prop_file,
70            log4cplus::tchar* prefix_log_flie = NULL,
71            log4cplus::tchar* sub_logger_name = NULL)
72        {
73            log4cplus::initialize();
```

```
74          m_logger = log4cplus::Logger::getRoot();
75
76          if (NULL == prop_file) return;
77
78          try
79          {
80              if (NULL != prefix_log_flie && '\0' != prefix_log_flie[0])
81                  change_log_file_name(prop_file, prefix_log_flie);
82              else
83                  log4cplus::PropertyConfigurator::doConfigure(prop_file);
84                  //log4cplus::ConfigureAndWatchThread
    cfg_thread(prop_file, 24*60*60*1000);
85
```

测试程序

C++ | 复制代码

```
1   //#define CRUISE_LOG4CPLUS
2   //#include "cruise_log4cplus_init_single.h"
3   #include "cruise_log4cplus.hpp"
4
5
6   int main(int argc, char* argv[])
7   {
8       log_init(_T(".\\log4cplus.properties"), _T("aaaa"));
9       //for (int i = 0; i < 1000; ++i)
10      {
11
12          LOG_METHOD(_TEXT("主函数"));
13          LOG_ASSERT(true);
14          LOG_ASSERT(false);
15
16
17          LOG_TRACE(_TEXT("Log4cplus library!") << 2);
18          LOG_TRACE_FMT(_TEXT("%d, Log4cplus library!"), 3);
19      }
20
21      log_uinit();
22      system("pause");
23      return 0;
24  }
```