# C++的日志框架 log4cplus 简介

徐少辉

# 目录

# 一 为什么要引入日志框架？

在日常的开发工作中，对于调试代码，我们常用的方法有：

**1. 下断点单步调试**

嵌入式开发或多线程问题存在局限性。

**2. printf 或者 cout 输出**

格式不统一，控制台字符编码问题。

**3. 写文件**

文件管理，线程同步等问题。

**4. MessageBox 弹窗**

控制台程序不方便使用，UI 框架的问题，比如 MFC，Qt 的弹框机制和方法的差异。

上面每一种调试方法都存在一定的局限性，很大程度上，我们开发项目的时候，因人而异，每个人选择自己喜欢的方式来跟踪问题。问题解决了，怎么清理调试代码？这方面做得也是各有千秋。临时性的，实验性的代码一不小心就提交到了仓库。看各种风格迥异的调试代码就像吃一顿大餐的时候突然掉进去一只苍蝇一样。

所以要引入日志框架，所有项目成员采用接口一致，风格相同的形式来跟踪调试代码，并且要能解决单一技术遇到的局限性问题，真正让调试日志做到可管理可维护起来。

# 二 什么是日志框架？

既然日志框架这么重要，那么什么是日志框架呢，从根本上来讲，就是一个库程序，它要解决以下几个核心问题：

- 日志内容除了能打印到控制台，还可以输出到文件，输出到网络服务器上等。
- 日志内容应该可以做各种格式化，例如变成纯文本，xml，html 等格式。
- 对于不同的类或动态库，应该可以灵活的输出到不同文件中。
- 能对日志进行分级，比如 info，debug，warn，error 等。
- 可以根据分级灵活调整输出哪些级别的日志。
- 要能方便在多线程程序中的日志区分，NDC 功能。
- 所有的开关选项最好都能通过配置文件搞定，做到开箱即用的灵活性。

# 三 log4cplus 是什么？

## 3.1 官方简介

log4cplus 是 C++编写的开源的日志系统，前身是 java 编写的 log4j 系统，受 Apache Software License 保护，作者是 Tad E.Smith。

log4cplus 具有线程安全、灵活、以及多粒度控制的特点，通过将日志划分优先级使

其可以面向程序调试、运行、测试、和维护等全生命周期。你可以选择将日志输出到控制台、文件、NT event log、甚至是远程服务器。通过指定策略对日志进行定期备份等。

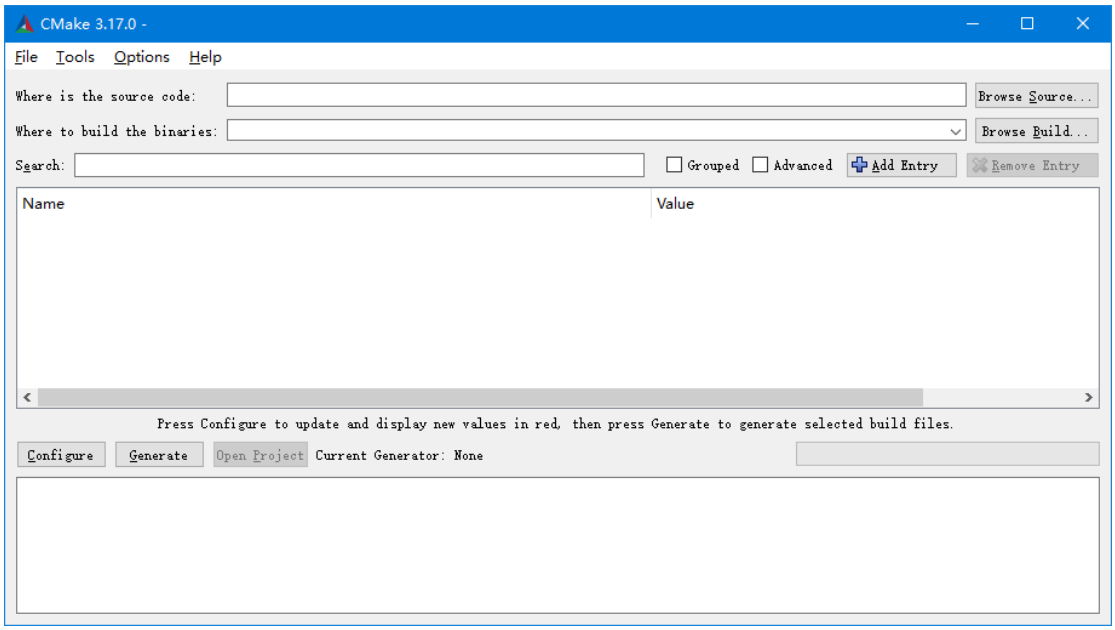
## 3.2 官网地址

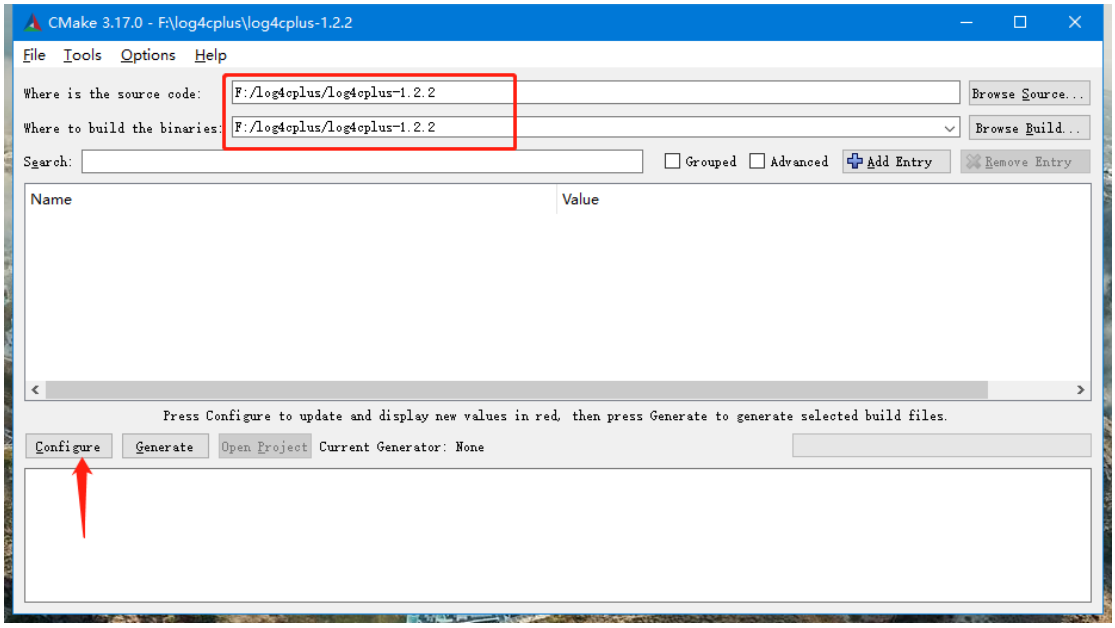
主页：https://sourceforge.net/projects/log4cplus/
文档：https://log4cplus.sourceforge.io/docs/html/config_8hxx.html

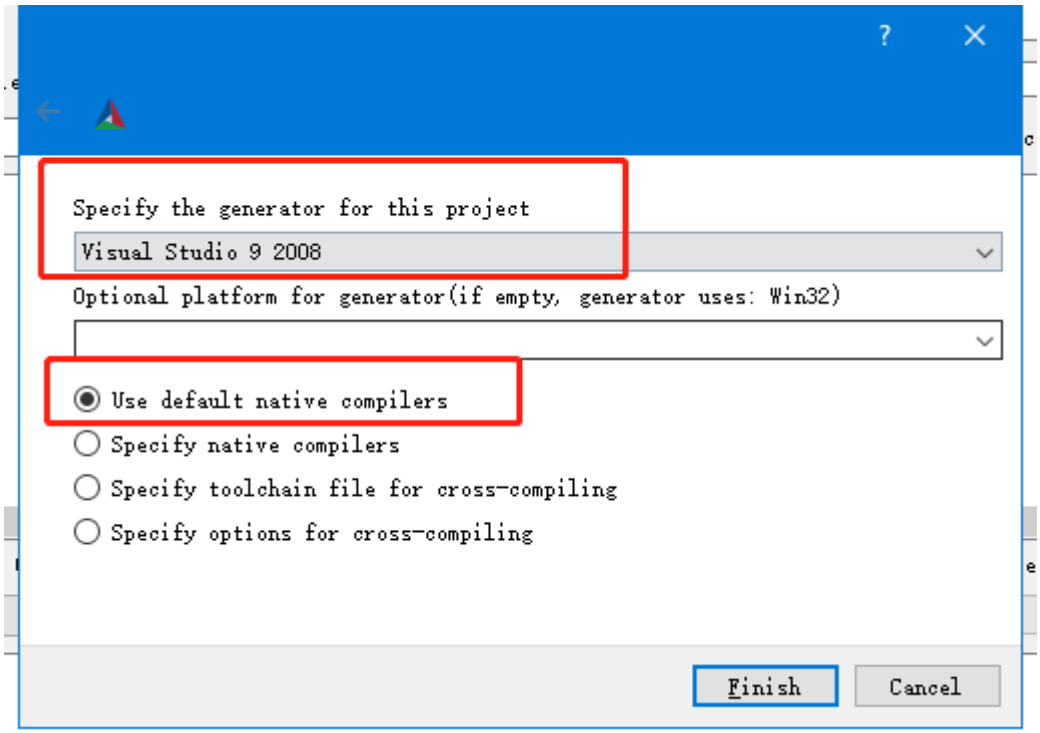
## 3.3 编译方法

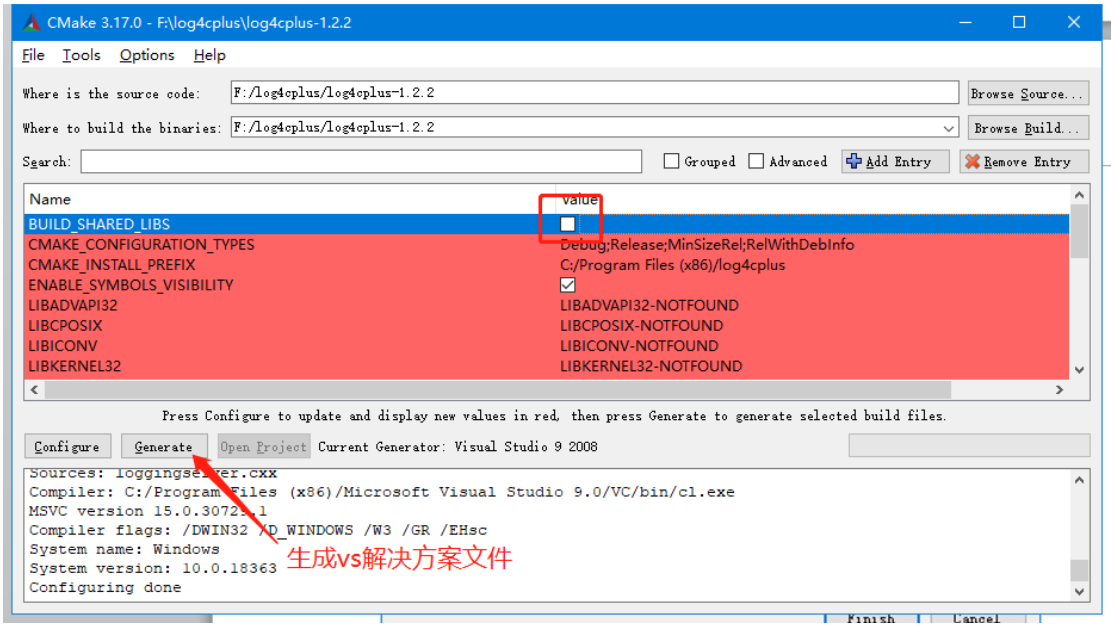
我这边演示一下怎么在 Windows 下把 log4cplus 编译成静态库使用。最新的稳定版是 2.0.5，这个版本需要 C++11 的特性。因为要在 VS2008 中使用，2008 不能全部支持 C++11，所以我下载的是 1.2.2 版本。

1. 解压 log4cplus-1.2.2.zip
2. 安装 cmake
3. 打开 cmake-gui




CMake 3.17.0 - F:\log4cplus\log4cplus-1.2.2
File Tools Options Help
Where is the source code: F:/log4cplus/log4cplus-1.2.2
Where to build the binaries: F:/log4cplus/log4cplus-1.2.2
Browse Source...
Browse Build...
Search:
Grouped
Advanced
Add Entry
Remove Entry
Name
Value
Press Configure to update and display new values in red, then press Generate to generate selected build files.
Configure
Generate
Open Project
Current Generator: None


Specify the generator for this project
Visual Studio 9 2008
Optional platform for generator(if empty, generator uses: Win32)
Use default native compilers
Specify native compilers
Specify toolchain file for cross-compiling
Specify options for cross-compiling
Finish
Cancel




然后就可以使用 vs 来编译静态库了。

## 3.4 基本概念

| 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，WARR,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。 |
| LogLevel | 优先权，包括 TRACE， DEBUG，INFO，WARNING，ERROR，FATAL。 |

## 3.5 基本步骤

使用 log4cplus 有 6 个基本步骤

1. 实例化一个封装了输出介质的 appender 对象。
2. 实例化一个封装了输出格式的 layout 对象（可省略，默认使用 SimpleLayout）。
3. 将 layout 对象绑定到 appender 对象。
4. 实例化一个封装了日志输出 logger 对象。
5. 将 appender 对象绑定到 logger 对象。
6. 设置 logger 的优先级（可省略，默认输出所有级别）

# 四 实战举例

## 4.1 输出到控制台

```
//基本用法
int main(int argc, char *argv[])
{
    //1. 创建一个控制台appender
    SharedObjectPtr<Appender> append(new ConsoleAppender());

    //2. 创建一个pattern的layout
    tstring pattern = L"%d{%Y-%m-%d %H:%M:%S}---%b[%L]  - %m %n";
    std::auto_ptr<Layout> layout(new PatternLayout(pattern));

    //3. 把layout挂接到appender
    append->setLayout(layout);

    //4. 获取一个logger对象
    Logger logger = Logger::getInstance(L"test");

    //5. 把appender挂接到logger对象
    logger.addAppender(append);

    //6. 设置logger的日志级别，大于该级别的都会输出
    logger.setLogLevel(ALL_LOG_LEVEL);

    //LOG宏的使用，输出字符串 puts
    LOG4CPLUS_DEBUG(logger, L"This is the debug message");
    LOG4CPLUS_WARN(logger, L"This is the warn message");
    LOG4CPLUS_ERROR(logger, L"This is the error message");

    //LOG宏的使用，c++流语法
    LOG4CPLUS_DEBUG(logger, L"this is a bool: " << true);
    LOG4CPLUS_INFO(logger, L"this is a char: " << L'x');
    LOG4CPLUS_WARN(logger, L"this is a int: " << 1000);
    LOG4CPLUS_FATAL(logger, L"this is a double: " << 123.456);

    //LOG宏的使用，printf语法
    LOG4CPLUS_DEBUG_FMT(logger, L"%d", 2000);
    LOG4CPLUS_WARN_FMT(logger, L"%.3f", 34.56789);
    LOG4CPLUS_WARN_FMT(logger, L"%c", L'a');
    LOG4CPLUS_DEBUG_FMT(logger, L"%s", L"hello");

    return 0;
}
```

## 4.2 日志级别的使用

```
logger.setLevel(ERROR_LOG_LEVEL); //只输出 ERROR 以上的日志
logger.setLevel(OFF_LOG_LEVEL);   //关闭日志输出
```

## 4.3 输出到文件

```
int main(int argc, char *argv[])
{
    //1. 创建一个文件appender
    SharedObjectPtr<Appender> append(new FileAppender(L"./test.log"));
    //SharedObjectPtr<Appender> append2(new ConsoleAppender());

    //2. 创建一个pattern的layout
    tstring pattern = L"%D{%Y-%m-%d %H:%M:%S}---%b[%L]  - %m %n";
    std::auto_ptr<Layout> layout(new PatternLayout(pattern));
    //std::auto_ptr<Layout> layout2(new PatternLayout(pattern));

    //3. 把layout挂接到appender
    append->setLayout(layout);
    //append2->setLayout(layout2);

    //4. 获取一个logger对象
    Logger logger = Logger::getInstance(L"test");

    //5. 把appender挂接到logger对象
    logger.addAppender(append);
    //logger.addAppender(append2);

    //6. 设置logger的日志级别，大于该级别的都会输出
    logger.setLogLevel(ALL_LOG_LEVEL);

    //LOG宏的使用，输出字符串 puts
    LOG4CPLUS_DEBUG(logger, L"This is the debug message");
    LOG4CPLUS_WARN(logger, L"This is the warn message");
    LOG4CPLUS_ERROR(logger, L"This is the error message");

    //LOG宏的使用，c++流语法
    LOG4CPLUS_DEBUG(logger, L"this is a bool: " << true);
    LOG4CPLUS_INFO(logger, L"this is a char: " << L'x');
    LOG4CPLUS_WARN(logger, L"this is a int: " << 1000);
    LOG4CPLUS_FATAL(logger, L"this is a double: " << 123.456);

    //LOG宏的使用，printf语法
    LOG4CPLUS_DEBUG_FMT(logger, L"%d", 2000);
    LOG4CPLUS_WARN_FMT(logger, L"%.3f", 34.56789);
    LOG4CPLUS_WARN_FMT(logger, L"%c", L'a');
    LOG4CPLUS_DEBUG_FMT(logger, L"%s", L"hello");

    return 0;
```

## 4.4 输出到滚动文件

```
int main(int argc, char *argv[])
{
    //1. 创建一个滚动文件appender
    //默认最小的文件大小是200K，第3个参数为3，最多会有4个log文件，循环滚动
    SharedObjectPtr<Appender> append(new RollingFileAppender(L"./test.log", 200*1024, 3));

    //2. 创建一个pattern的layout
    tstring pattern = L"%D{%Y-%m-%d %H:%M:%S}---%b[%L]  - %m %n";
    std::auto_ptr<Layout> layout(new PatternLayout(pattern));

    //3. 把layout挂接到appender
    append->setLayout(layout);

    //4. 获取一个logger对象
    Logger logger = Logger::getInstance(L"test");

    //5. 把appender挂接到logger对象
    logger.addAppender(append);

    for (int i = 0; i < 3000; i++) {
        LOG4CPLUS_INFO(logger, L"the loop time: " << i);

        //LOG宏的使用，输出字符串 puts
        LOG4CPLUS_DEBUG(logger, L"This is the debug message");
        LOG4CPLUS_WARN(logger, L"This is the warn message");
        LOG4CPLUS_ERROR(logger, L"This is the error message");

        //LOG宏的使用，c++流语法
        LOG4CPLUS_DEBUG(logger, L"this is a bool: " << true);
        LOG4CPLUS_INFO(logger, L"this is a char: " << L'x');
        LOG4CPLUS_WARN(logger, L"this is a int: " << 1000);
        LOG4CPLUS_FATAL(logger, L"this is a double: " << 123.456);

        //LOG宏的使用，printf语法
        LOG4CPLUS_DEBUG_FMT(logger, L"%d", 2000);
        LOG4CPLUS_WARN_FMT(logger, L"%.3f", 34.56789);
        LOG4CPLUS_WARN_FMT(logger, L"%c", L'a');
        LOG4CPLUS_DEBUG_FMT(logger, L"%s", L"hello\n");
    }

    return 0;
}
```

# 4.5 多线程使用 NDC

```
//线程执行函数
static DWORD WINAPI threadProc1(LPVOID lpParam)
{
    //线程标识戳
    NDCContextCreator ndc(L"thread1");

    for (int i = 0; i < 20; i++) {
        LOG4CPLUS_DEBUG_FMT(logger, L"%s", L"hello world 1");
    }
    return 0;
}

//线程执行函数
static DWORD WINAPI threadProc2(LPVOID lpParam)
{
    //线程标识戳
    NDCContextCreator ndc(L"thread2");

    for (int i = 0; i < 20; i++) {
        LOG4CPLUS_DEBUG_FMT(logger, L"%s", L"hello world 2");
    }
    return 0;
}

int main(int argc, char *argv[])
{
    HANDLE hThread1, hThread2;

    //1. 创建一个控制台appender
    SharedObjectPtr<Appender> append(new ConsoleAppender());

    //2. 创建一个pattern的layout
    //%x就是输出线程标识戳用的，-7表示左对齐，7个字符，不足补空格
    tstring pattern = L"%D{%Y-%m-%d %H:%M:%S}---%b[%L] - [%-7x] - %m %n";
    std::auto_ptr<Layout> layout(new PatternLayout(pattern));

    //3. 把layout挂接到appender
    append->setLayout(layout);

    //4. 获取一个logger对象
    logger = Logger::getInstance(L"test");

    //5. 把appender挂接到logger对象
    logger.addAppender(append);

    //6. 设置logger的日志级别，大于该级别的都会输出
    logger.setLogLevel(ALL_LOG_LEVEL);

    //主线程标识戳
    NDCContextCreator ndc(L"main");

    LOG4CPLUS_INFO(logger, L"in main function");

    hThread1 = CreateThread(NULL, 0, threadProc1, NULL, 0, NULL);
    if (hThread1 == NULL) {
        LOG4CPLUS_ERROR(logger, L"create thread1 error\n");
    }
    CloseHandle(hThread1);

    hThread2 = CreateThread(NULL, 0, threadProc2, NULL, 0, NULL);
    if (hThread2 == NULL) {
        LOG4CPLUS_ERROR(logger, L"create thread2 error\n");
    }
    CloseHandle(hThread2);

    for (int i = 0; i < 20; i++) {
        LOG4CPLUS_DEBUG_FMT(logger, L"%s", L"hello world 0");
    }

    return 0;
}
```

## 4.6 配置文件的使用

```
int main(int argc, char *argv[])
{
    //加载配置文件
    PropertyConfigurator::doConfigure(L"./test.cfg");
    Logger logger = Logger::getRoot();

    for (int i = 0; i < 3000; i++) {
        LOG4CPLUS_INFO(logger, L"the loop time: " << i);

        //LOG宏的使用，输出字符串 puts
        LOG4CPLUS_DEBUG(logger, L"This is the debug message");
        LOG4CPLUS_WARN(logger, L"This is the warn message");
        LOG4CPLUS_ERROR(logger, L"This is the error message");

        //LOG宏的使用，c++流语法
        LOG4CPLUS_DEBUG(logger, L"this is a bool: " << true);
        LOG4CPLUS_INFO(logger, L"this is a char: " << L'x');
        LOG4CPLUS_WARN(logger, L"this is a int: " << 1000);
        LOG4CPLUS_FATAL(logger, L"this is a double: " << 123.456);

        //LOG宏的使用，printf语法
        LOG4CPLUS_DEBUG_FMT(logger, L"%d", 2000);
        LOG4CPLUS_WARN_FMT(logger, L"%.3f", 34.56789);
        LOG4CPLUS_WARN_FMT(logger, L"%c", L'a');
        LOG4CPLUS_DEBUG_FMT(logger, L"%s", L"hello\n");
    }

    return 0;
}
```

## 4.7 输出到远程服务器

### 4.7.1 客户端

```
int main(int argc, char *argv[])
{
    //SocketAppender不需要配置Layout，输出格式由服务端的Layout决定
    SharedObjectPtr<Appender> append(new SocketAppender(L"127.0.0.1", 8988, L"local server"));
    Logger logger = Logger::getRoot();
    logger.addAppender(append);

    //LOG宏的使用，输出字符串 puts
    LOG4CPLUS_DEBUG(logger, L"This is the debug message");
    LOG4CPLUS_WARN(logger, L"This is the warn message");
    LOG4CPLUS_ERROR(logger, L"This is the error message");

    //LOG宏的使用，c++流语法
    LOG4CPLUS_DEBUG(logger, L"this is a bool: " << true);
    LOG4CPLUS_INFO(logger, L"this is a char: " << L'x');
    LOG4CPLUS_WARN(logger, L"this is a int: " << 1000);
    LOG4CPLUS_FATAL(logger, L"this is a double: " << 123.456);

    //LOG宏的使用，printf语法
    LOG4CPLUS_DEBUG_FMT(logger, L"%d", 2000);
    LOG4CPLUS_WARN_FMT(logger, L"%.3f", 34.56789);
    LOG4CPLUS_WARN_FMT(logger, L"%c", L'a');
    LOG4CPLUS_DEBUG_FMT(logger, L"%s", L"hello");

    //这边很重要，要延时一下，不能提前结束客户端socket
    log4cplus::helpers::sleep(1);

    return 0;
}
```

### 4.7.2 服务器端

```cpp
#include "log4cplus/config.hxx"
#include "log4cplus/configurator.h"
#include "log4cplus/consoleappender.h"
#include "log4cplus/socketappender.h"
#include "log4cplus/helpers/loglog.h"
#include "log4cplus/helpers/socket.h"
#include "log4cplus/thread/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;

//链接WinSock库
#pragma comment(lib, "Ws2_32.lib")

//定义一个客户端类
class ClientThread : public AbstractThread
{
public:
    ClientThread(Socket cs) : clientsock(cs)
    {
        wcout << L"Received a client connection!!!!" << endl;
    }

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

    virtual void run();

private:
    Socket clientsock;
};

void ClientThread::run()
{
    while (1) {
        if (!clientsock.isOpen()) {
            return;
        }

        //分两步读
        //1. 读取一个报文长度
        SocketBuffer msgSizeBuffer(sizeof(unsigned int));
        bool ret = clientsock.read(msgSizeBuffer);
        if (!ret) {
            return;
        }

        //2. 读取实际的报文内容
        unsigned int msgSize = msgSizeBuffer.readInt();
        SocketBuffer buffer(msgSize);
        if (!clientsock.read(buffer)) {
            return;
        }

        spi::InternalLoggingEvent event = readFromBuffer(buffer);
        Logger logger = Logger::getInstance(event.getLoggerName());
        logger.callAppenders(event);
    }
}

int main(int argc, char *argv[])
{
    //加载配置文件
    PropertyConfigurator::doConfigure(L"./test.cfg");

    //启动服务
    ServerSocket server(8988);

    while (1) {
        //监听新的客户端
        ClientThread *cthread = new ClientThread(server.accept());
        cthread->start();
    }

    return 0;
}
```