

C++ 好用的日志库--spdlog

原创

litanyuan

于 2023-08-03 13:59:28 发布

阅读量2.2k

★ 收藏 20

👍 点赞数 5

分类专栏:

C++ 开发库

文章标签:

c++

开发语言



C++ 开发库 专栏收录该内容

1 订阅 3 篇文章

背景

spdlog 是一个快速、异步的、header-only 的 C++ 日志库。它提供了简单易用的 API 并具有高性能和可扩展性。

下载和使用

下载

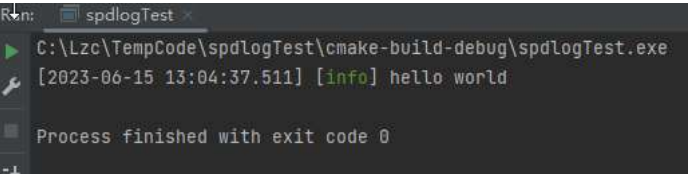
spdlog 库下载地址: [github 链接](#)

hello world

在使用时只需要 include 整个 /include/spdlog 文件夹即可。

```
1 #include "spdlog/spdlog.h"
2 int main() {
3     spdlog::info("hello world");
4     return 0;
5 }
```

运行结果如下:



如上图所示, spdlog 库上手非常简单。

基本概念

核心组件

spdlog 有以下基本组成部分:

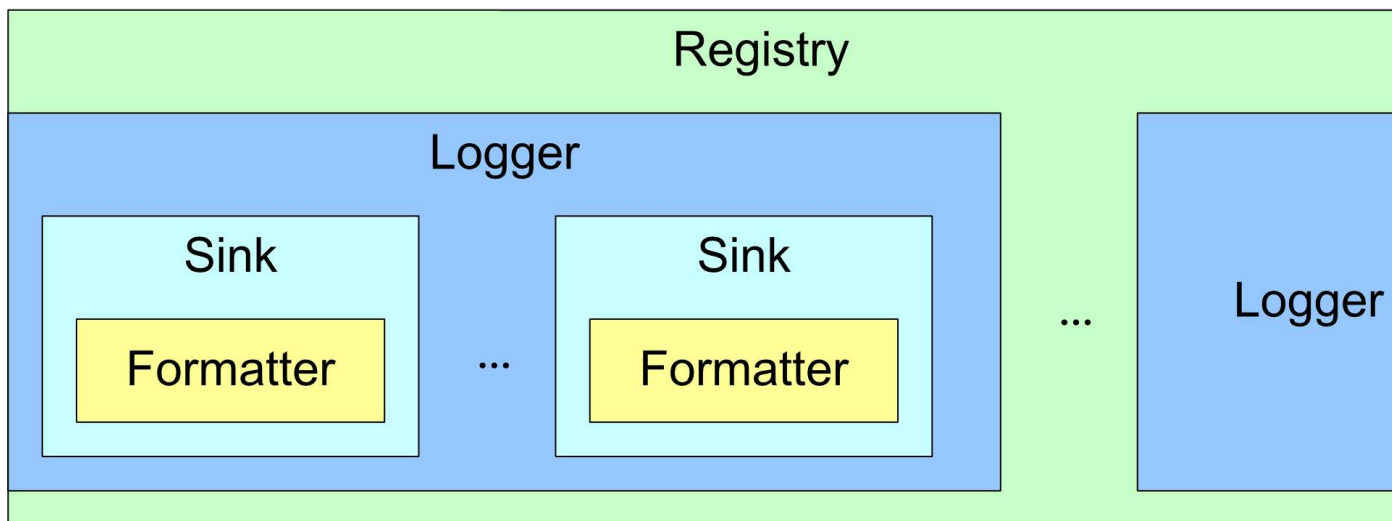
- Registry (日志记录器注册表): Registry 用于管理所有已创建的 Logger 对象。
- Logger (日志记录器): Logger 是打印日志的核心对象, 负责记录日志消息。可以根据需要创建多个 Logger 对象。
- Sink (日志输出): Sink 是 Logger 的目标输出位置, 它指定了日志消息的最终存储位置。每个 Logger 内包含一个 Sink 组成的 vector。
- Formatter (日志格式化): Formatter 用于格式化日志消息的输出格式。



litanyuan

关注





通过以上组件，可以灵活地配置和使用 spdlog，以满足不同的日志需求。例如，可以创建多个 Logger 对象，并将它们的日志消息输出到不同的文件。自定义日志消息的格式，添加时间戳和其他元数据。

日志格式

spdlog 自带了默认的 formatter，其格式为：[日期时间] [logger名] [log级别] log内容。

```
Run: spdlogTest x
C:\Lzc\TempCode\spdlogTest\cmake-build-debug\spdlogTest.exe
[2023-06-15 13:04:37.511] [info] hello world
Process finished with exit code 0
```

日志级别

spdlog 提供了一组日志级别，用于控制记录哪些级别的日志消息：

- trace：最详细的日志级别，提供追踪程序执行流程的信息。
 - debug：调试级别的日志信息，用于调试程序逻辑和查找问题。
 - info：通知级别的日志信息，提供程序运行时的一般信息。
 - warn：警告级别的日志信息，表明可能发生错误或不符合预期的情况。
 - error：错误级别的日志信息，表明发生了某些错误或异常情况。
 - critical：严重错误级别的日志信息，表示一个致命的或不可恢复的错误。
- 通过设置日志记录器的级别，可以控制哪些级别的日志进行输出：

```
1 #include "spdlog/sinks/stdout_color_sinks.h"
2 int main() {
3     auto logger = spdlog::stdout_color_mt("console");
4     logger->set_level(spdlog::level::warn);
5
6     logger->trace("trace message");
7     logger->debug("debug message");
8     logger->info("info message");
9     logger->warn("warn message");
10    logger->error("error message");
11    logger->critical("critical message");
12
13    return 0;
14 }
```

运行结果如下：

```
Run: spdlogTest x
[2023-06-15 15:26:45.961] [console] [warning] warn message
[2023-06-15 15:26:45.963] [console] [error] error message
[2023-06-15 15:26:45.963] [console] [critical] critical message
Process finished with exit code 0
```



litanyuan

关注

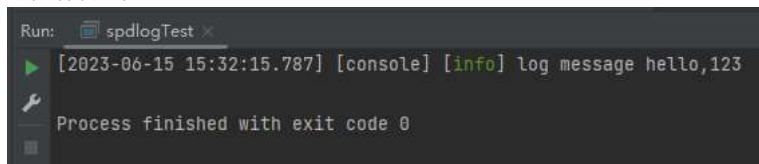


日志参数

spdlog 绑定了 fmt 库，可以用于格式化输出日志内容：

```
1 | #include "spdlog/sinks/stdout_color_sinks.h"
2 | int main() {
3 |     auto logger = spdlog::stdout_color_mt("console");
4 |     logger->info("log message {}", {}, "hello", 123);
5 |     return 0;
6 | }
```

运行结果如下：



spdlog 快速上手

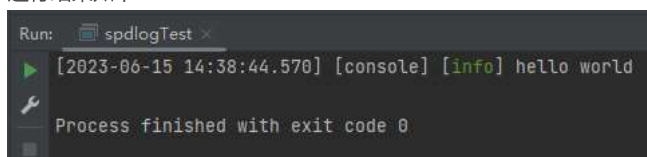
概述

spdlog 提供了一系列工厂函数用于创建 Logger。其中以 _mt 后缀的表示创建多线程的日志记录器、以 -st 后缀的表示创建单线程的日志记录器。

创建控制台 Logger

```
1 | #include "spdlog/spdlog.h"
2 | #include "spdlog/sinks/stdout_color_sinks.h"
3 | int main() {
4 |     auto logger = spdlog::stdout_color_mt("console");
5 |     logger->info("hello world");
6 |     return 0;
7 | }
```

运行结果如下：



创建基本文件 Logger

```
1 | #include "spdlog/spdlog.h"
2 | #include "spdlog/sinks/basic_file_sink.h"
3 | int main() {
4 |     auto logger = spdlog::basic_logger_mt("file", "my_log.log");
5 |     logger->info("hello world");
6 |     return 0;
7 | }
```

运行结果如下：



创建滚动文件 Logger

可以设置日志文件的大小及数量限定：



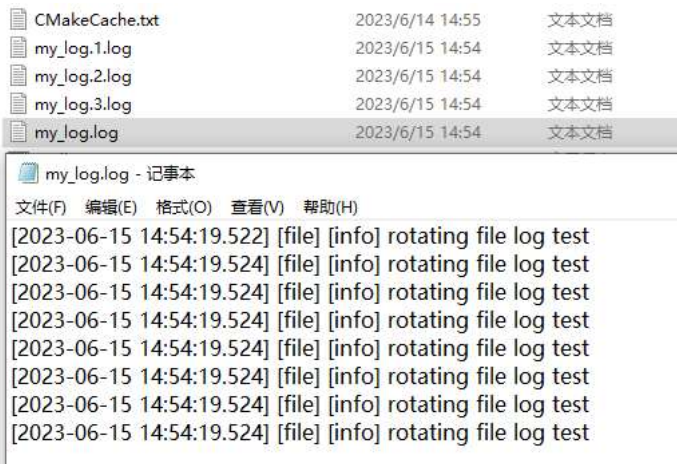
litanyuan

关注



```
1 #include "spdlog/spdlog.h"
2 #include "spdlog/sinks/rotating_file_sink.h"
3 int main() {
4     auto max_size = 1024*2;// 每个文件最大 2 k
5     auto max_files = 3;//最多滚动 3 次
6     auto logger = spdlog::rotating_logger_mt("file","my_log.log", max_size, max_files);
7
8     for( int i = 0 ;i < 100;++i)
9     {
10         logger->info("rotating file log test");
11     }
12     return 0;
13 }
```

运行结果如下:



创建每日 Logger

可以在每天固定时间点创建一个新的日志文件:

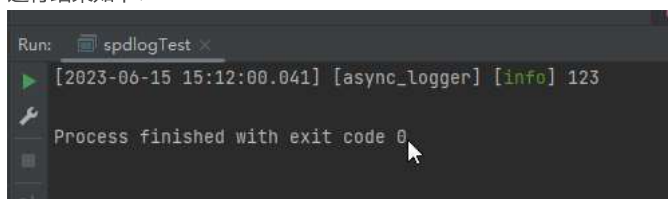
```
1 #include "spdlog/spdlog.h"
2 #include "spdlog/sinks/daily_file_sink.h"
3 int main() {
4     auto logger = spdlog::daily_logger_mt("daily_logger", "my_log.log", 0, 0);
5     return 0;
6 }
```

创建异步日志

可以与 `spdlog::async_factory` 搭配使用, 以实现异步日志。异步记录可以提高程序的性能, 因为日志写入操作不会阻塞主线程。

```
1 #include "spdlog/async.h"
2 #include "spdlog/sinks/stdout_color_sinks.h"
3 int main() {
4     auto logger = spdlog::stdout_color_mt<spdlog::async_factory>("async_logger");
5     logger->info("123");
6     return 0;
7 }
```

运行结果如下:



创建 Logger

spdlog 提供的工厂方法封装了 sink 的创建过程, 也可以根据需要先创建 sink 再创建 Logger

创建 Sink



litanyuan

关注



sink 是将日志实际写入其目标（文件、控制台、数据库等）的对象，且应仅负责单个目标。

创建控制台 sink

```
1 | auto sink = std::make_shared<spdlog::sinks::stdout_color_sink_mt>();
```

创建文件 sink

```
1 | auto sink = std::make_shared<spdlog::sinks::basic_file_sink_mt>("my_log.log");
```

创建每日文件 sink

```
1 | auto sink = std::make_shared<spdlog::sinks::daily_file_sink_mt>("my_log.log", 23,59);
```

创建滚动文件 sink

```
1 | auto sink = std::make_shared<spdlog::sinks::rotating_file_sink_mt>("my_log.log",1048576 * 5, 3,false);
```

创建流输出 sink

```
1 | std::ostringstream oss;
2 | auto ostream_sink = std::make_shared<spdlog::sinks::ostream_sink_mt> (oss);
```

根据 sink 创建 Logger

创建单 sink 记录器

可以直接在构造函数中传入 sink 来创建 Logger:

```
1 | int main() {
2 |     auto sink = std::make_shared<spdlog::sinks::stderr_color_sink_mt>();
3 |     auto logger = std::make_shared<spdlog::logger>("my_logger", sink);
4 |     logger->info("hello world ");
5 |     return 0;
6 | }
```

创建多 sink 记录器

也可以传入一个 sink 列表来创建 Logger，每个 sink 可单独设置日志级别和样式:

```
1 | int main() {
2 |     auto sink1 = std::make_shared<spdlog::sinks::stderr_color_sink_mt>();
3 |     auto sink2 = std::make_shared<spdlog::sinks::basic_file_sink_mt>("my_log.log");
4 |     spdlog::sinks_init_list sinks = {sink1,sink2};
5 |
6 |     auto logger = std::make_shared<spdlog::logger>("my_logger", sinks.begin(),sinks.end());
7 |
8 |     logger->info("hello world ");
9 |     return 0;
10 | }
```

创建共用 sink 记录器

多个 Logger 也可以共用相同的 sink :

```
1 | int main() {
2 |     auto sink = std::make_shared<spdlog::sinks::stderr_color_sink_mt>();
3 |     auto logger1 = std::make_shared<spdlog::logger>("logger1", sink);
4 |     auto logger2 = std::make_shared<spdlog::logger>("logger2", sink);
5 |     auto logger3 = std::make_shared<spdlog::logger>("logger3", sink);
6 |
7 |     logger1->info("hello world ");
8 |     logger2->info("hello world ");
9 |     logger3->info("hello world ");
10 |     return 0;
11 | }
```



litanyuan

关注



Logger 注册与获取

spdlog 提供了一个全局注册和获取 logger 的方法。

Logger 注册

使用 spdlog 工厂方法创建的 logger 无需手动注册即可根据名称获取，手动创建的 logger 需要注册。

```
1 | #include "spdlog/sinks/stdout_color_sinks.h"
2 | #include <spdlog/spdlog.h>
3 |
4 | void register_logger()
5 | {
6 |     auto sink = std::make_shared<spdlog::sinks::stderr_color_sink_mt>();
7 |     auto logger = std::make_shared<spdlog::logger>("my_logger", sink);
8 |     spdlog::register_logger(logger);
9 | }
10 | int main() {
11 |     register_logger();
12 |
13 |     auto logger = spdlog::get("my_logger");
14 |     logger->info("hello world");
15 |     return 0;
16 | }
```

Logger 删除

手动注册的全局 logger 也可以删除：

```
1 | spdlog::drop("my_logger");//全局注册中删除指定 logger
2 | spdlog::drop_all();// 删除所有注册的 logger
```

Logger 的使用与设置

设置默认 Logger

spdlog 提供了最为便捷的默认 logger，注意，该logger在全局公用：

```
1 | spdlog::info("hello world");
```

可以设置自定义的 logger 为全局默认：

```
1 | auto logger = spdlog::stdout_color_mt("my_log");
2 | spdlog::set_default_logger(logger);
```

设置日志级别

logger 和 sink 都可以单独指定日志级别。

设置指定 Logger 级别

```
1 | auto logger = spdlog::stdout_color_mt("my_log");
2 | logger->set_level(spdlog::level::debug);
```

设置指定 sink 级别

```
1 | auto sink = std::make_shared<spdlog::sinks::stdout_color_sink_mt>();
2 | sink->set_level(spdlog::level::debug);
```

设置默认 Logger 级别

```
1 | spdlog::set_level(spdlog::level::debug);
```

设置缓存刷新策略

创建好 Logger 后建议设置 flush 方式，否则可能无法立刻在文件中看到 logger 的内容：

定时刷新



litanyuan

关注



```
1 | spdlog::flush_every(std::chrono::seconds(5)); // 定期为所有注册的logger隔5秒刷新
```

基于级别刷新

```
1 | auto logger = spdlog::stdout_color_mt("my_log");  
2 | logger->flush_on(spdlog::level::warn); //遇到 warn 就立即刷新
```

手动刷新

```
1 | auto logger = spdlog::stdout_color_mt("my_log");  
2 | logger->flush() // logger 将依次在每个 sink 上调用 flush
```

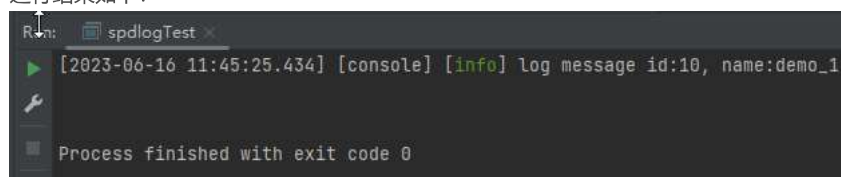
spdlog 使用进阶

记录自定义类型

用户自定义类型对象作为日志参数进行记录，需要重置输出运算符：

```
1 | #include "spdlog/sinks/stdout_color_sinks.h"  
2 | #include "spdlog/fmt/ostr.h"  
3 |  
4 | class demo{  
5 | public:  
6 |     demo(int id_,const std::string &name_):id(id_),name(name_){}  
7 | private:  
8 |     int id ;  
9 |     std::string name;  
10 | public:  
11 |     friend std::ostream& operator<<(std::ostream& os, const demo& d);  
12 | };  
13 |  
14 | std::ostream &operator<<(std::ostream &os, const demo &d) {  
15 |     os << "id:"<<d.id<<"", name:"<<d.name<<std::endl;  
16 |     return os;  
17 | }  
18 |  
19 | int main() {  
20 |     demo d(10,"demo_1");  
21 |     auto logger = spdlog::stdout_color_mt("console");  
22 |     logger->info("log message {}",d);  
23 |     return 0;  
24 | }
```

运行结果如下：



记录 vector 中数据

vector 对象可以直接作为参数进行输出：

```
1 | #include "spdlog/sinks/stdout_color_sinks.h"  
2 | #include "spdlog/fmt/ranges.h"  
3 | int main() {  
4 |     auto logger = spdlog::stdout_color_mt("console");  
5 |     std::vector<int> vec{1,2,3,4,5};  
6 |  
7 |     logger->info("vector data :{}",vec);  
8 |     return 0;  
9 | }
```

运行结果如下：



litanyuan

关注

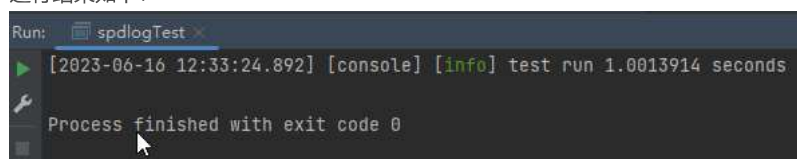


记录运行时间

使用 `spdlog::stopwatch` 对象可以记录代码运行时间:

```
1 | #include "spdlog/sinks/stdout_color_sinks.h"
2 | #include "spdlog/stopwatch.h"
3 | #include <thread>
4 |
5 | void test()
6 | {
7 |     std::this_thread::sleep_for(std::chrono::milliseconds(1000));
8 | }
9 |
10 | int main() {
11 |     auto logger = spdlog::stdout_color_mt("console");
12 |     spdlog::stopwatch sw;
13 |     test();
14 |     logger->info("test run {} seconds",sw);
15 |     return 0;
16 | }
```

运行结果如下:

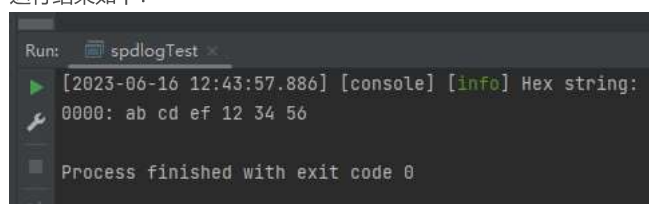


记录十六进制数据

使用 `to_hex` 可以把二进制数据转十六进制进行记录:

```
1 | #include "spdlog/sinks/stdout_color_sinks.h"
2 | #include "spdlog/fmt/bin_to_hex.h"
3 | int main() {
4 |     auto logger = spdlog::stdout_color_mt("console");
5 |
6 |     unsigned char data[] = {0xAB, 0xCD, 0xEF, 0x12, 0x34, 0x56};
7 |     logger->info("Hex string: {}", spdlog::to_hex(std::begin(data), std::begin(data)+sizeof(data)));
8 |
9 |     return 0;
10 | }
```

运行结果如下:



还可以显示对应 ASCII 值:

```
1 | int main() {
2 |     auto logger = spdlog::stdout_color_mt("console");
3 |
4 |     unsigned char data[] = {0x61, 0x62, 0x63, 0x64, 0x65, 0x66};
5 |
6 |     logger->info("Hex string: {:a} ", spdlog::to_hex(std::begin(data), std::begin(data)+sizeof(data),4));
7 |     return 0;
8 | }
```

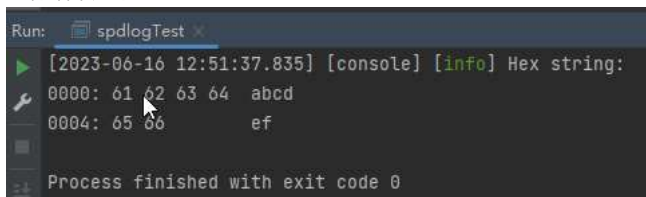


litanyuan

关注



运行结果如下:



```
Run: spdlogTest x
[2023-06-16 12:51:37.835] [console] [info] Hex string:
0000: 61 62 63 64  abcd
0004: 65 66          ef
Process finished with exit code 0
```

记录文件名及行号

在使用 spdlog 记录日志时, 可以通过格式化字符串来包含方法名、行号和文件名的信息:

```
1 | #define SPDLOG_ACTIVE_LEVEL SPDLOG_LEVEL_TRACE
2 | #include <spdlog/spdlog.h>
3 | #include "spdlog/sinks/stdout_color_sinks.h"
4 |
5 | int main() {
6 |     spdlog::set_pattern("[%H:%M:%S] [%n] [%^---%L---%$] [%s:%#] [%!] %v");
7 |     auto logger = spdlog::stdout_color_mt("my_log");
8 |     SPDLOG_LOGGER_INFO(logger, "This is a log message");
9 |     return 0;
10 | }
```

运行结果如下:



```
Run: spdlogTest x
[15:42:28] [my_log] [---I---] [main.cpp:13] [main] This is a log message
Process finished with exit code 0
```

一定确保 SPDLOG_ACTIVE_LEVEL 定义的日志级别低于或等于你期望的日志级别, 并且在包含 spdlog.h 之前定义了它。

其他特殊 Logger

qt sink

qt_sink 可以向 QTextBrowser、QTextEdit 等控件输出日志消息:

```
1 | #include "spdlog/sinks/qt_sinks.h"
2 | auto logger = spdlog::qt_logger_mt("QLogger", ui->textBrowser);
3 | logger->info("hello QTextBrowser");
4 | logger->warn("this msg from spdlog");
```

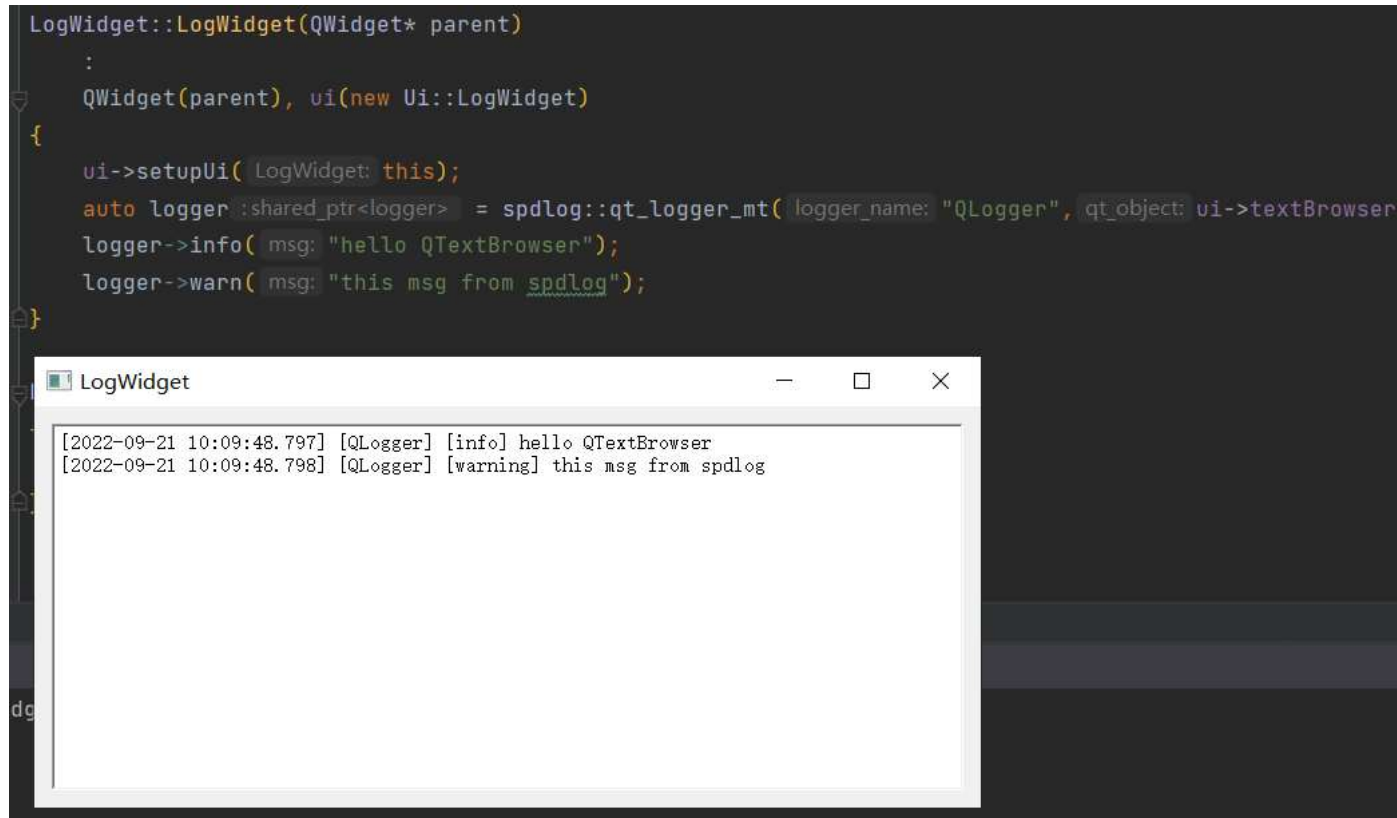


litanyuan

关注



运行结果如下:

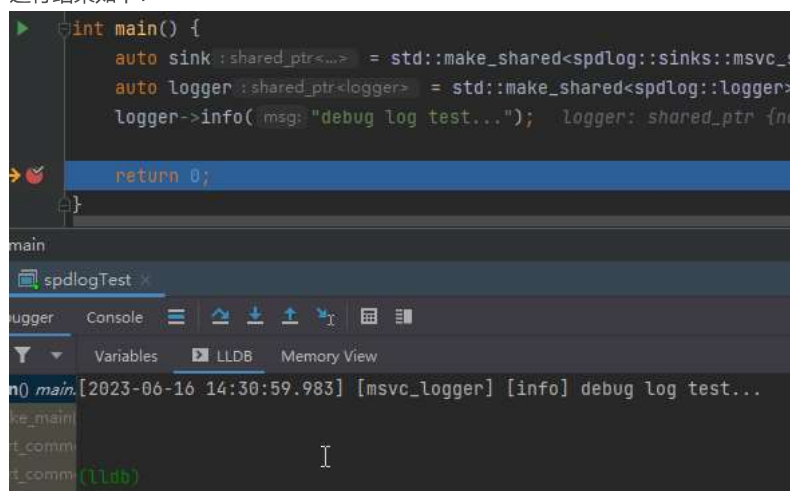


msvc sink

msvc_sink 使用 OutputDebugStringA 向 Windows 调试接收器发生日志记录:

```
1 #include "spdlog/sinks/stdout_color_sinks.h"
2 #include "spdlog/sinks/msvc_sink.h"
3 int main() {
4     auto sink = std::make_shared<spdlog::sinks::msvc_sink_mt>();
5     auto logger = std::make_shared<spdlog::logger>("msvc_logger", sink);
6     logger->info("debug log test...");
7     return 0;
8 }
```

运行结果如下:



消息过滤 sink

dup_filter_sink 可以实现重复消息删除:

```
#include "spdlog/sinks/stdout_color_sinks.h"
#include "spdlog/sinks/dup_filter_sink.h"
1 int main() {
2     auto sink = std::make_shared<spdlog
```



litanyuan

关注

```

4
5
6     auto dup_filter_sink= std::make_shared<spdlog::sinks::dup_filter_sink_mt>(std::chrono::seconds(5));
7     dup_filter_sink->add_sink(sink);
8
9     auto logger = std::make_shared<spdlog::logger>("msvc_logger", dup_filter_sink);
10    logger->info("hello world");
11    logger->info("hello world");
12    logger->info("hello world");
13    logger->info("hello world");
14    logger->info("log msg");
15
16    return 0;
17 }

```

运行结果如下:

```

Run: spdlogTest x
[2023-06-16 14:43:42.333] [msvc_logger] [info] hello world
[2023-06-16 14:43:42.334] [msvc_logger] [info] Skipped 3 duplicate messages..
[2023-06-16 14:43:42.334] [msvc_logger] [info] log msg

```

ringbuffer sink

ringbuffer_sink 将最新的日志消息保存在内存中:

```

1 #include <spdlog/spdlog.h>
2 #include "spdlog/sinks/ringbuffer_sink.h"
3 #include "spdlog/fmt/ranges.h"
4
5 int main() {
6     auto ringbuffer_sink = std::make_shared<spdlog::sinks::ringbuffer_sink_mt>(5);
7     auto logger = std::make_shared<spdlog::logger>("ringbuffer_logger", ringbuffer_sink);
8
9     for (int i = 0; i < 20; ++i) {
10        logger->info("Log message {}", i);
11    }
12
13    std::vector<std::string> log_messages = ringbuffer_sink->last_formatted(1);
14    spdlog::info("{} ", log_messages);
15
16    return 0;
17 }

```

运行结果如下:

```

Run: spdlogTest x
[2023-06-16 14:52:21.355] [info] ["[2023-06-16 14:52:21.354] [ringbuffer_logger] [info] Log message 19\r\n"]

```

udp sink

spdlog 提供的一个封装了 UDP 传输的 logger。它可以将日志记录通过 UDP 协议发送到指定的目标地址和端口:

```

1 #include <spdlog/spdlog.h>
2 #include "spdlog/sinks/udp_sink.h"
3 int main() {
4     spdlog::sinks::udp_sink_config cfg(
5     auto my_logger = spdlog::udp_logger
6

```



litanyuan

关注



```
5 |         my_logger->info("hello world");
7 |
8 |
9 |     return 0;
  | }
```

支持回调的 Logger

callback_logger_mt 是一个支持设置调用回调函数的日志记录器:

```
1 | #include <spdlog/spdlog.h>
2 | #include <iostream>
3 | #include "spdlog/sinks/callback_sink.h"
4 | int main() {
5 |     auto logger = spdlog::callback_logger_mt("custom_callback_logger", [] (const spdlog::details::log_msg & msg) {
6 |         std::cout << msg.payload.data() << std::endl;
7 |     });
8 |
9 |     logger->info("123");
10 |    return 0;
11 | }
```

运行结果如下:



微信搜索“编程猿来如此”关注公众号获取更多内容。

鸿蒙DevEco Studio系统及自定义快捷键

所谓工欲善其事，必先利其器，今天给大家带来了可以提高开发效率的系统快捷键的用法，包含了window环境和mac环境，如果系统的不够用，我们还可以自己自定义。

spdlog学习—安装及基本使用_spdlog安装

auto filesink = std::make_shared<spdlog::sinks::simple_file_sink_mt>("./logs/multilogger.txt");// 创建两个日志记录器来进行测试auto one_logger = std::make_shared<spdlog::l

简单好用的C++日志库spdlog使用示例_spdlog 日志格式

C++日志库有很多,glog,log4cpp,easylogging++, easylogger, plog,spdlog等等。每个都大致了解过,看过github代码,看下来还是觉得spdlog用起来最方便最简单,head-only。

spdlog一个非常好用的C++日志库(一): 简介与使用

红军不怕远征难，万水千山只

spdlog是一个开源的C++日志库，它提供了高性能和易用性的日志记录功能。它是为了满足现代C++应用程序的需要而创建的，可以在不同的平台上运行。 spdlog 是一个

[C++]日志记录库SPDLog简介

农家/

文章目录spdlog库日志记录槽sink日志记录器logger输出格式pattern对齐方式截断字符串格式化fmtFormat Specificationsspdlog使用异常处理logger基础用法stdout日志文件

【GitHub项目推荐--一个极快速的 C++ 日志库】【转载】_github c++开源...

5. 支持异步日志记录:通过异步模式,Spdlog 可以将日志消息放入队列,然后由线程池异步处理,这样可以避免在记录日志时阻塞主线程。 6. 丰富的日志格式:Spdlog 使用 fmt

[C++]日志记录库SPDLog_c++ spdlog

[C++]日志记录库SPDLog spdlog是基于C++11实现的一款纯头文件的日志管理库(git地址:https://github.com/gabime/spdlog,API说明:https://spdlog.docsforge.com/v1.x/1.q

探秘：Spdlog - 高速、灵活的 C++日志库

新法的

Spdlog是一个开源的、快速的、仅有头文件的C++11日志库。它提供了向流、标准输出、文件、系统日志、调试器等目标输出日志的能力，支持的平台包括Windows、Lin

[C++] 第三方日志库spdlog介绍和使用

老狼工作室的

本文主要介绍了一个流行的C++日志库 spdlog，及如何把spdlog引入到你的cmake项目，并使用spdlog来写控制台日志和文件日志。

【C++】开源:spdlog跨平台日志库配置使用_spdlog 通过配置文件 使用...

项目Github地址:https://github.com/gabime/spdlog Spdlog是一个高性能的 C++ 日志库,具有简单易用的 API 和灵活的配置选项。它被设计成易于集成到现有项目中,并提供

异步日志方案——spdlog_spdlog 异步日志

spdlog是一款开源的C++日志库,他有着极高的性能和几乎零成

C++高性能日志库spdlog使用指南



litanyuan

关注



Spdlog 专注于提供极致的性能，在大量日志记录场景下也能保持较低的延迟和较高的吞吐量。

日志---spdlog m0_60274660的
日志

【C++】Spdlog日志库_windows下c++ spdlog日志库
Github: <https://github.com/gabime/spdlog> https://blog.csdn.net/tutou_gou/article/details/121284474 二、 日志封装 #pragmaonce#include<iostream>#include<string>#inc

Windows10中使用VS2022和Cmake编译构建C++开源日志库-spdlog ccf19881030的
Windows10中VS2022中使用spdlog日志库

【C++】CentOS环境搭建-安装C++ spdlog日志库 圈小圈DBA的
spdlog: Speed + Log，是一个高速异步日志库，支持多线程和旋转文件日志，适合用于高负载的系统。glog：Google出品，提供了多种级别的日志输出、多个日志目录的

qt封装的spdlog日志库
简易c++日志库 可以支持基本的日志库使用，同时支持向每日日志和循环日志中记录日志内容 使用spd的多线程模式，线程安全， spdlog直接使用头文件，无cpp，支持后

spdlog:快速 C++ 日志库-开源
spdlog 是一个只有头文件的库。 只需将 include 下的文件复制到您的构建树并使用 C++11 编译器。 它使用捆绑的 fmt 库提供了类似 Python 的格式化 API。 spdlog 采用“

日志输出库spdlog
`spdlog` 是一个高效、现代且功能丰富的C++日志库，它为开发者提供了灵活的日志记录解决方案。这个库的设计目标是提供高性能、轻量级的日志记录，同时保持易于使

cpp-spdlog超级快速C日志库
cpp-spdlog超级快速C++日志库详解 `spdlog` 是一个高效、轻量级且功能丰富的C++日志库，它被...在实际项目中，结合`gabime-spdlog-a7148b7`这样的源码版本，可

C++ 日志库 spdlog C++ 日志库 spdlog C++ 日志库 spdlog C++ 日志库 spdlog
C++ 日志库 spdlog C++ 日志库 spdlog C++ 日志库 spdlog C++ 日志库 spdlog C++ 日志库 spdlog C++ 日志库 spdlog

C语言 之 理解指针 (1) fallzzzzz的
就是先获取该数组的首元素的地址，从前面我们知道，指针的类型的作用是指针+1 所跳过的字节数，int为整形，大小为4个字节，所以p+1就跳过4个字节，p+2就跳过8个

规控面试常见问题 最新发布 每天进步一
(1) A*在节点的扩张时，只会进行相邻节点的直线移动，没有考虑实际底盘的运动学模型（物理模型），而Hybrid A*在每次节点扩张的时候，会对下个位置进行离散状态

C++ primer plus 第16章string 类和标准模板库, 关联容器 zhyjhacker的
C++ primer plus 第16章string 类和标准模板库, 关联容器。

VS2019编译和使用gtest测试 (C++) m0_65635427的
下载下来解压到文件夹，再在文件夹里面新建一个build文件夹，如下：再安装cmake，可以先检查一下是否安装了cmake，打开命令窗口，输入cmake -version：然后双击

C++的spdlog库
C++的spdlog库是一个快速的C++日志库，它支持多线程、异步日志记录和格式化日志输出。它的目标是提供简单易用的API，同时具有高性能和可扩展性。spdlog的使用:

关于我们 招贤纳士 商务合作 寻求报道 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00
公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心
家长监护 网络110报警服务 中国互联网举报中心 Chrome商店下载 账号管理 规范 版权与免责声明 版权申诉 出版物许可证 营业执照
©1999-2024北京创新乐知网络技术有限公司



litanyuan
码龄10年 暂无认证

211	5万+	212万+	37万+	
原创	周排名	总排名	访问	等级
2480	78	247	19	1372
积分	粉丝	获赞	评论	收藏



私信

关注








litanyuan 关注





搜博主文章 

热门文章

- C++ 中生成随机数的方法总结  61393
- C++中定义常量的几种方式  15194
- C++中定义别名的几种方式总结  12837
- Qt 中 deleteLater 使用总结  9096
- std::thread 传递参数  8878

分类专栏

	C++ 开发库	3篇
	FFmpeg	2篇
	Effective C++	1篇
	Qt	40篇
	算法与数据结构	30篇
	protobuf	7篇



最新评论

- C++ 好用的格式化库--fmt
sunhaiyang8: std::print(std::cout, ...)
- C++ 好用的包管理工具--vcpkg
Runesia: 集成到项目不行，失败了
- C++ 好用的格式化库--fmt
fortunely2: fmt::print 这种用法，是fmt外部库形式吧？如果内置的std::format呢？
- C++中定义常量的几种方式
ab84878: #define MAX =5，为什么不能把=去了
- Qt 中 QButtonGroup 使用总结
Sweet hort: 信号为什么要强制类型转换呢？

最新文章

- C++ 好用的格式化库--fmt
- C++ 好用的包管理工具--vcpkg
- 掌握 Effective C++ : 条款01

2023年 6篇 2022年 112篇
2020年 93篇



目录

背景

下载和使用

下载

hello world

基本概念

核心组件

日志格式

日志级别

日志参数

spdlog 快速上手

概述

创建控制台 Logger

创建基本文件 Logger

