

C++ 好用的格式化库--fmt

原创

litanyuan

于 2023-08-23 12:57:15 发布

阅读量4.7k

收藏 25

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

fmt 库是一个开源的 C++ 格式化库，它提供了一种简洁、安全和高效的方式来进行字符串格式化。
该库的设计目标是提供与 Python 的字符串格式化语法类似的功能，同时保持 C++ 的类型安全性和性能。

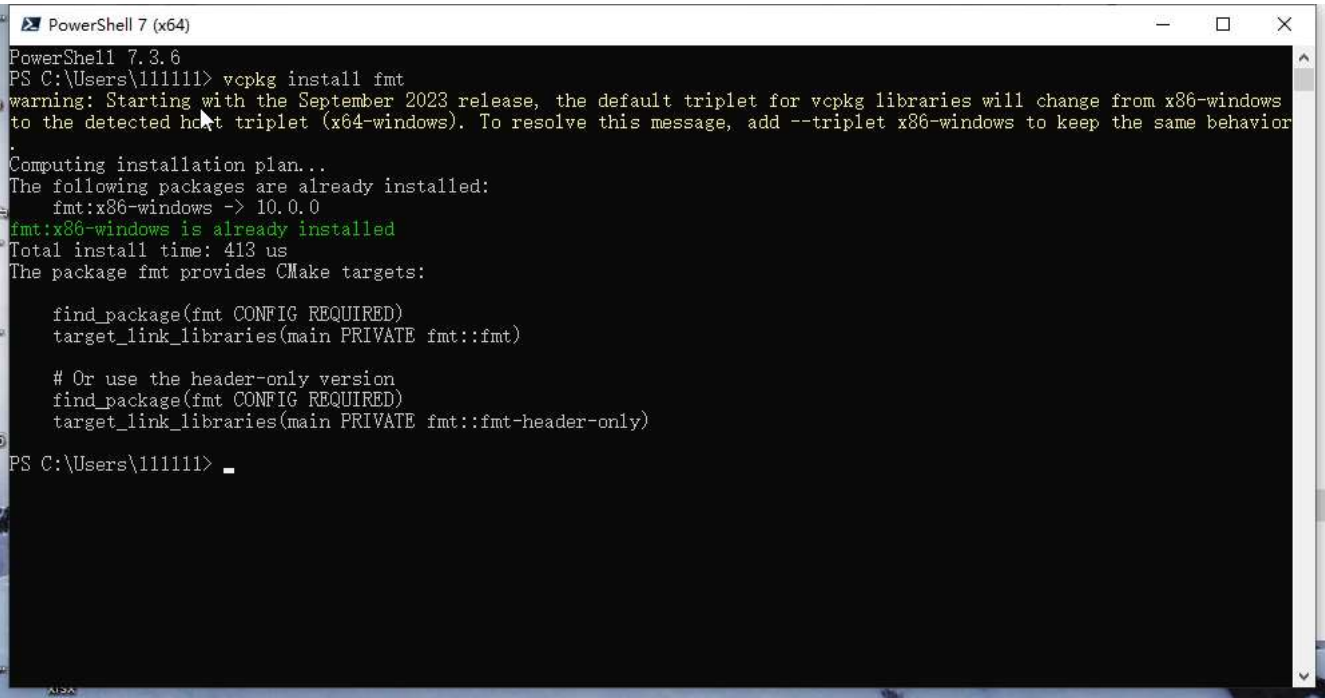
下载与安装

官网下载

fmt 官网地址: <https://fmt.dev/latest/index.html>。
可以从官网或者 GitHub 存储库 (<https://github.com/fmtlib/fmt>) 下载源代码并手动构建。

使用 vcpkg 安装

fmt 也可以通过 vcpkg 包管工具进行下载安装:



header-only


fmt 库也支持 header-only 方式使用，需要设置 FMT_HEADER_ONLY 宏。

```
1 | #define FMT_HEADER_ONLY
2 | #include "fmt/core.h"
3 |
```

hello world

下面是一个使用C++fmt库的简单示例:

```
1 | #include "fmt/core.h"
2 |
3 | int main()
4 | {
5 |     fmt::print("hello {}", "world");
6 | }
```



litanyuan

关注



```
7 |     return 0;  
   | }
```

运行上述代码，将输出以下结果：

```
#include "fmt/core.h"  
  
int main()  
{  
    fmt::print(fmt:: "hello {}", "world");  
}  
C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe  
hello world
```

基本格式化语法

参数替换

类似于 printf 的 % 占位符输出，fmt 使用大括号替代参数，且和参数类型无关：

```
1 | #include "fmt/core.h"  
2 |  
3 | int main()  
4 | {  
5 |     fmt::print("hello {}", 123);  
6 |     return 0;  
7 | }
```

运行上述代码，将输出以下结果：

```
int main()  
{  
    fmt::print(fmt:: "hello {}", 123);  
}  
C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe  
hello 123
```

按位置替换参数

参数位置默认从 0 开始：

```
1 | #include "fmt/core.h"  
2 |  
3 | int main()  
4 | {  
5 |     fmt::print("{0}+{1}={2}", 1,2,3);  
6 |     return 0;  
7 | }
```

运行上述代码，将输出以下结果：

```
int main()  
{  
    fmt::print(fmt:: "{0}+{1}={2}", 1, 2, 3);  
}  
C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe  
1+2=3
```

如果没有指定位置，默认从 0 往后依次替换：

```
1 | fmt::print("{}+{}={}", 1,2,3);
```

运行上述代码，将输出以下结果：



litanyuan

关注



```
int main()
{
    fmt::print(fmt::format("{}+{}={}", 1, 2, 3));
}
```



参数格式化

可以按指定格式替换指定位置的参数，用 {位置:格式} 的形式表示替换内容。

指定填充字符

可以指定字符串长度，若长度不够则使用指定的字符进行填充。

右侧填充


使用 < 表示右填充，即文本居左：

```
1 | fmt::print("{0:*<10}", "hello");
```

运行结果如下：

```
#include "fmt/core.h"

int main()
{
    fmt::print(fmt::format("{0:*<10}", "hello"));
}
```



左侧填充


使用 > 表示左填充，即文本居右：

```
1 | fmt::print("{0:*>10}", "hello");
```

运行结果如下：

```
#include "fmt/core.h"

int main()
{
    fmt::print(fmt::format("{0:*>10}", "hello"));
}
```




两边填充

使用 ^ 表示在两层填充，即文本居中：

```
1 | fmt::print("{0:*^10}", "hello");
```

运行结果如下：

```
int main()
{
    fmt::print(fmt::format("{0:*^10}", "hello"));
}
```



litanyuan

关注



默认填充字符


可以指定填充字符时，默认使用空格进行填充：

```
1 | fmt::print("{0:>10}\n{1:>10}", "hello", "world");
```

运行结果如下：

```
#include "fmt/core.h"

int main()
{
    fmt::print(fmt::format(" {0:>10}\n{1:>10}", "hello", "world"));
}
```




动态设置字符串长度

字符串的长度也可以用参数指定：

```
1 | fmt::print("{0:>{1}}\n{2:>10}", "hello", 20, "world");
```

运行结果如下：

```
int main()
{
    fmt::print(fmt::format(" {0:>{1}}\n{2:>10}", "hello", 20, "world"));
}
```



按精度格式化

使用 . 表示精度格式化。

格式化字符串长度


使用 .n 表示把字符串的长度格式为 n：

```
1 | fmt::print("{0:.4}", "abcdefg");
```

运行结果如下：

```
#include "fmt/core.h"

int main()
{
    fmt::print(fmt::format(" {0:.4}", "abcdefg"));
}
```



格式化浮点数长度


使用 .n 也可以用来表示把浮点数长度为 n：

```
1 | fmt::print("{0:.3}", 1.23456);
```

运行结果如下：

```
#include "fmt/core.h"

int main()
{
    fmt::print(fmt::format("{}", 1.23456));
}
```




格式化小数位数

使用 .nf 表示把浮点数小数位数格式化为 n:

```
#include "fmt/core.h"

int main()
{
    fmt::print(fmt::format("{}", 3.14));
}
```



动态设置精度


精度值也可以通过参数动态设置:

```
1 | fmt::print("{}", 3.141596, 2);
```

运行结果如下:

```
#include "fmt/core.h"

int main()
{
    fmt::print(fmt::format("{}", 3.141596, 2));
}
```



数字正负号格式化

用于格式化数字正负符号显示。

仅负数显示符号

使用 - 表示仅负数显示符号:

```
1 | fmt::print("正数: {}, 负数: {}", 30, -20);
```

运行结果如下:

```
#include "fmt/core.h"

int main()
{
    fmt::print("正数: {}, 负数: {}", 30, -20);
}
```



正负数都显示符号:

使用 + 表示正负数字都显示符号:

```
1 | fmt::print("正数: {}, 负数: {}", 30, -20);
```



litanyuan

关注



运行结果如下:

```
#include "fmt/core.h"

int main()
{
    fmt::print(fmt::format("正数:{0:~}\n负数:{1:~}", 30, -20));
}
```



数字进制格式化

格式化为10进制

使用 d 表示格式化数字为10进制进行显示:

```
1 | fmt::print("10进制:{0:d}", 10);
```

运行结果如下:

```
#include "fmt/core.h"

int main()
{
    fmt::print(fmt::format("10进制:{0:d}", 10));
}
```



格式化为2进制

使用 b 表示格式化数字为2进制进行显示:

```
1 | fmt::print("2进制:{0:b}", 10);
```

运行结果如下:

```
#include "fmt/core.h"

int main()
{
    fmt::print(fmt::format("2进制:{0:b}", 10));
}
```



格式化为16进制

使用 x 表示格式化数字为16进制进行显示:

```
1 | fmt::print("16进制:{0:x}", 10);
```

运行结果如下:

```
#include "fmt/core.h"

int main()
{
    fmt::print(fmt::format("16进制:{0:x}", 10));
}
```



显示进制符号

在进制符号前加 # 可以在进制格式化时增加符号标记:



litanyuan

关注



```
1 | fmt::print("16进制:{0:#x}", 10);
```

运行结果如下:

```
#include "fmt/core.h"

int main()
{
    fmt::print(fmt: "16进制:{0:#x}", 10);
}

C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe
16进制:0xa
```

格式化数字长度

在进制符号前可以增加长度表示, 长度不够补0:

```
1 | fmt::print("{0:08d}", 10);
```

运行结果如下:

```
#include "fmt/core.h"

int main()
{
    fmt::print(fmt: "{0:08d}", 10);
}

C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe
00000010
```

fmt 库使用

格式化内容到字符串

fmt::format 方法会把格式化的结果转为字符串:

```
1 | #include <iostream>
2 | #include "fmt/core.h"
3 |
4 | int main()
5 | {
6 |     auto res = fmt::format("hello {}", "world");
7 |     std::cout << res << std::endl;
8 |     return 0;
9 | }
```

运行结果如下:

```
#include "fmt/core.h"

int main()
{
    auto res:string = fmt::format(fmt: "hello {}", "world");
    std::cout << res << std::endl;
}

选择 C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe
hello world
```

格式化内容到迭代器

fmt::format_to 方法可以把内容格式化到指定的迭代器:

```
1 | #include "fmt/core.h"
2 |
3 | int main()
4 | {
5 |
```



litanyuan

关注




```
5 |     std::string s;  
6 |     fmt::format_to(std::back_inserter(s), "hello {}", "world");  
7 |     std::cout << s << std::endl;  
8 |     return 0;  
9 | }
```

运行结果如下:

```
#include "fmt/core.h"  
  
int main()  
{  
    std::string s; s = "hello world"  
    fmt::format_to(out: std::back_inserter([&] s), fmt: "hello {}", "world");  
    std::cout << s << std::endl;  
}  
C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe  
hello world
```

格式化内容到控制台

fmt::print 方法把格式化结果输出到控制台显示:

```
1 | fmt::print("hello {}", "world");
```

fmt 库使用进阶

使用参数格式化

命名参数

使用 fmt::arg 可以构建一个命名参数:

```
1 | fmt::print("{a:*<10}{b:#x}", fmt::arg("a", "hello"), fmt::arg("b", 100));
```

运行结果如下:

```
#include "fmt/core.h"  
  
int main()  
{  
    fmt::print(fmt: "{a:*<10}{b:#x}", fmt::arg(name: "a", arg: "hello"), fmt::arg(name: "b", arg: 100));  
}  
C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe  
hello*****0x64
```

参数列表

fmt::vformat 支持使用参数列表进行格式化:

```
1 | #include <iostream>  
2 | #include "fmt/core.h"  
3 |  
4 | int main()  
5 | {  
6 |     const fmt::format_args args = fmt::make_format_args(  
7 |         fmt::arg("a", "hello"),  
8 |         fmt::arg("b", 100)  
9 |     );  
10 |    const auto s = fmt::vformat("{a:*<10}{b:#x}", args);  
11 |    std::cout << s << std::endl;  
12 |    return 0;  
13 | }
```

运行结果如下:



litanyuan

关注



```
#include "fmt/core.h"

int main()
{
    const fmt::format_args args = fmt::make_format_args(
        fmt::arg(name: "a", arg: "hello"),
        fmt::arg(name: "b", arg: 100)
    );
    const auto s:string = fmt::vformat(fmt: "{a:*<10} {b}"
    std::cout << s << std::endl;
}

C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe
hello*****0x64
```

同样 `fmt::vprint` 也支持传入参数列表进行格式化。

自定义类型格式化

特例化 `formatter< T >` 方式

特例化 `fmt::formatter< T >` 并且实现其 `parse` 和 `format` 方法，可以实现对自定义类型的格式化。

自定义数据类型

使用以下自定义类型作为示例：

```
1 struct my_struct
2 {
3     int id;
4     std::string name;
5 };
```

特例化 `fmt::formatter< T >`

特例化 `fmt::formatter< T >` 并实现其 `parse` 和 `format` 方法：

```
1 template <>
2 struct fmt::formatter<my_struct>
3 {
4     char presentation = 'f';
5     auto parse(fmt::format_parse_context &ctx) -> decltype(ctx.begin())
6     {
7         auto it = ctx.begin(), end = ctx.end();
8         if (it != end && (*it == 'f' || *it == 'i')) presentation = *it++;
9         if (it != end && *it != '}') throw "invalid format";
10        return it;
11    }
```

使用示例

```
1 int main()
2 {
3     my_struct ms{ 0,"hello" };
4     fmt::print("my_struct f 格式化:{0:f}\nmy_struct i 格式化:{0:i}", ms);
5     return 0;
6 }
```

运行结果如下：



litanyuan

关注



```
int main()
{
    my_struct ms{ .id: 0, .name: "hello" }; ms = {id=0 name="hello"}
    fmt::print("my_struct f 格式化: {0:f}\nmy_struct i 格式化: {0:i}", [&] ms);
}

C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe
my_struct f 格式化:[my_struct]id=0,name= hello
my_struct i 格式化:[my_struct]id=0_
```

继承现有 formatter 类

也可以通过继承现有的 formatter 实现自定义类的格式化:

```
1 #include "fmt/core.h"
2 #include "fmt/format.h"
3
4 struct my_struct
5 {
6     int id;
7     std::string name;
8 };
9 template <>
10 struct fmt::formatter<my_struct> : formatter<string_view>
11 {
```

运行结果如下:

```
int main()
{
    my_struct ms{ .id: 0, .name: "hello" };
    fmt::print("{}", [&] ms);
}

C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe
[my_struct]id=0
```

枚举类型格式化

对于枚举类型 fmt 提供了 format_as 接口用于类型转换。

转底层数据类型

使用 fmt::underlying 可以把枚举值转为底层数据类型:

```
1 #include "fmt/core.h"
2 #include "fmt/format.h"
3
4 enum class my_enum
5 {
6     red = 0,
7     green,
8     blue
9 };
10 auto format_as(my_enum e)
11 {
```

运行结果如下:



litanyuan

关注



```
int main()
{
    fmt::print(fmt:: "{}", my_enum::green);
}

C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe
1
```

转其他类型

也可以在 `format_as` 把枚举值转为其他类型:

```
1 | #include "fmt/core.h"
2 | #include "fmt/format.h"
3 |
4 | enum class my_enum
5 | {
6 |     red = 0,
7 |     green,
8 |     blue
9 | };
10 | auto format_as(my_enum e)
11 | {
```

运行结果如下:

```
int main()
{
    fmt::print(fmt:: "{}", my_enum::green);
}

C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe
green
```

容器元素格式化

`fmt::join` 可以定义分隔符对容器中的元素进行格式化:

```
1 | int main()
2 | {
3 |     std::string s = "1234567";
4 |     fmt::print("{} ", fmt::join(s, ", "));
5 |     return 0;
6 | }
```

运行结果如下:

```
int main()
{
    std::string s = "1234567";
    fmt::print(fmt:: "{}", fmt::join([&s], sep: ", "));
}

C:\Lzc\TempCode\qt_test\demo\Debug\demo.exe
1, 2, 3, 4, 5, 6, 7
```

对于 `std::tuple` 可以直接进行格式化操作:

```
1 | #include "fmt/core.h"
2 | #include <fmt/ranges.h>
3 | int main()
4 | {
5 |     std::tuple<int, char> t = { 1, 'a' };
6 |     fmt::print("{} ", t);
7 |     return 0;
8 | }
```




litanyuan

关注



运行结果如下：

```
1 | #include "fmt/core.h"
2 | #include <fmt/ranges.h>
3 | int main()
4 | {
5 |     std::tuple<int, char> t = { _This_arg: 1, 'a' };
6 |     fmt::print("fmt: {}", [&] t);
7 | }
8 |
9 |
~ |
```



微信搜索“编程猿来如此”关注公众号获取更多内容。

鸿蒙NEXT DevEco Studio安装教程

最近一段时间鸿蒙非常火，出现了很多高薪岗位，所以吸引了很多人入行，那么想要入行呢，第一步应该先把环境搞上，直接官网下载DevEco Studio5.0，然后进行安装

2 条评论



fortunely2 热评 fmt::print 这种用法，是fmt外部库形式吧？如果内置的std::format呢？

c++ {fmt}库使用指南—_c++ fmt

此时,我渴望做一些有意义的事情,决定借助这份宁静,静下心来,撰写一份关于fmt库的使用指南。fmt库是一个高效、易用的C++格式化库,可以帮助我们方便地进行字符串格

...fmt库下载,2024年最新C C++面试题2024_ c++ fmt 源码

编译fmt需要事先安装CMake。下载链接 1、打开CMake,并且点击左侧的【Browse Source】和【Browse Build】按钮添加对应的路径。这里我源码路径为F:/C++Project/

C/C++ fmt 格式化字符串实现（轻量）

liulittle的

提高 C/C++ 编译速度，fmt 库模板嵌套过多编译速度非常慢，且编译后程序体积也过大，函数步入的栈帧过多！

C++格式化库fmt使用方法 最新发布

XY_Y_CN的

C++开源格式化库fmt使用方法、格式化语法介绍。

c c++编程 fmt:占位符_c++ 输出占位符

c c++编程 fmt:占位符 #defineLOGW(fmt,arg,...)\do{\printf("INFO:" "[%s:%u] "fmt"\n",FILE,LINE,##arg);}while(0) 1 2 3 4 根据提供的引用内容,给出的代码段是一个宏定义,其

c++20 formatting(fmt)使用方法

Format是一个编程语言非常重要的一环,c++的jioanip里的各种骚操作,在长字符串使用中是一个灾难。因此在c++20中提出了新库formatting,用类似于python的语法实现了

c++20 formatting (fmt) 使用方法

Format是一个编程语言非常重要的一环，c++的jioanip里的各种骚操作，在长字符串使用中是一个灾难。因此在c++20中提出了新库formatting，用类似于python的语法实

fmt文本格式库的源码下载编译（Win10+VS2022）

林夕

fmt是一个先进的文本格式库，具有现代语言的特征，用来代替 C 的 stdio 和 C++ iostreams。实现了 C++20 的 std::format 标准。fmt 基于 CMakeLists.txt 开发，引入到其

【C++】fmt库的用法(一)_c++ 怎么使用 fmt 库

{FMT} 是一个开源格式库,按照作者的说法,是提供快速和安全的 替代 C 的 stdio 和 C++ 的iostreams。项目地址:https://github.com/fmtlib/fmt 下载最新版的 release 文件即

【C++学习笔记】超级好用的fmt库_c++ fmt实现原理

C++ 好用的格式化库—fmt 编程猿来如此 4724 fmt 库的使用 Linux下编译fmt库 xupeng1644的博客 4967 下载fmt的下载地址为https://github.com/fmtlib/fmt/tags,以下载版本

C++格式化输出开源库fmt入手教程

性感的小小

格式API在精神上与C print函数家族相似，但比通用标准库实现更安全、更简单且速度快好几倍。格式字符串语法类似于Python中str.form使用的语法。如果不是c++ 20，还

【fmt】fmt简介及例程

WilliamCNTH的

{fmt}是一个开源的文本格式库，用以替代C的stdio和C++的iostreams。

Go——fmt包详解

吴声子夜歌的

Go 语言中的fmt包含有格式化输入输出的函数，类似于C语言的printf和scanf。格式字符串的规则来源于C但更简单更好用。fmt包实现了格式化I/O。主要分为向外输出内容

fmt.formatDate的输出格式详解

jerry191的

fmt.formatDate的输出格式详解 2004-5-31 23:59:59 2004-4-1 23:59:59 2004-5-31 04-5-31 2004-5-31 2004年5月31日 2004年5月31日 星期一 23:59:59 下午11:59 23:59:5

fmtlib 格式化 基本用法 1

laoki的

C++ 数据格式化

Sophus库报错 fatal error:fmt/core.h: No such file or directory 解决

qiqiqi的

Sophus库报错 fatal error:fmt/core.h: No such file or directory 解决 在跑《视觉SLAM十四讲》（第二版）中第5讲5.4.2节 RGB-D视觉中代码时遇到了报错： /usr/local/incl

cpp-cpphttplib一个headeronly的跨平台HTTPHTTPS服务器

cpp-http lib: 一个header-only的跨平台HTTP/HTTPS服务器和



litanyuan

关注



视觉SLAM十四讲-fmt库文件

fmt库是一个现代C++格式化库，它的主要功能是提供类似于C++ I/O流的简洁、高效且安全的文本格式化能力。fmt库的引入可以极大地提高代码的可读性，减少因为字符

现代格式库-C/C++开发

{fmt} fmt是C++的开源格式库。它可以用作printf的安全替代品，也可以用作iostreams的快速替代品。文档功能两种API：基于串联的更快wr {fmt} fmt是C++的开源格式

c++ fmt库，vs2019编译通过

C++的fmt库是一个强大的格式化库，它提供了一种高效、类型安全且易于使用的接口，用于在C++中进行字符串格式化。fmt库最初由Victor Zverovich创建，现在已经成为

c++{fmt} API 详解

本文旨在介绍 fmt 库的常用 API，包括格式化自定义结构体、枚举、标准库和时间等。通过本文，您将了解到如何使用这些 API 来更好地格式化和输出数据。在本文中，手

Sophous：编译时，报错“/usr/local/include/sophous/common.hpp:36:10: fatal error: fmt/core.h: 没有那个文件或目录”

weixin_45137202的排

按照《视觉SLAM十四讲》配置Sophous出现“/usr/local/include/sophous/common.hpp:36:10: fatal error: fmt/core.h: 没有那个文件或目录”其原因显然：缺少fmt/core.h头文

autoware.universe 编译踩坑

nobody258的

autoware.universe 编译踩坑

c++ fmt 库使用方法

C++的fmt库是一个现代的格式化库，它提供了一种简单的方式来格式化输出。以下是fmt库的使用方法： 1.安装fmt库 fmt库可以通过以下命令进行安装： ``shell sudo apt-

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+ 等级
原创 周排名 总排名 访问

2480 78 247 19 1372
积分 粉丝 获赞 评论 收藏



私信

关注

AI圈早知道，每日最新动态
了解全球AI新鲜事！

立即参与

大额流量券送不停
多发多得，流量翻倍！

去查看

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



litanyuan

关注



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++ 好用的包管理工具--vcpkg
- C++ 好用的日志库--spdlog
- 掌握 Effective C++ : 条款01

2023年 6篇

2022年 112篇

2020年 93篇

目录

下载与安装

- 官网下载
- 使用 vcpkg 安装
- header-only
- hello world

基本格式化语法

- 参数替换
- 按位置替换参数
- 参数格式化

指定填充字符

按精度格式化

数字正负号格式化

数字进制格式化

fmt 库使用



litanyuan

关注

