

std::ref和std::cref使用

原创

幻想之渔

🕒 于 2018-08-09 20:42:44 发布

👁 25819

🌟 收藏 95

版权

分类专栏:

C++11

 文章标签:


std::ref

std::cref

bind ref cref

bind 引用

bind参数修改



C++11 专栏收录该内容

1 订阅

17 篇文章

订阅专栏

std::ref和std::cref

解释

std::ref 用于包装按引用传递的值。

std::cref 用于包装按const引用传递的值。

为什么需要std::ref和std::cref

bind()是一个函数模板，它的原理是根据已有的模板，生成一个函数，但是由于bind()不知道生成的函数执行的时候，传递进来的参数是否还有效。所以它选择参数值传递而不是引用传递。如果想引用传递，std::ref和std::cref就派上用场了。

```
1  #include <functional>
2  #include <iostream>
3
4  void f(int& n1, int& n2, const int& n3)
5  {
6      std::cout << "In function: n1[" << n1 << "]"    n2[" << n2 << "]"    n3[" << n3 << "]" << std::endl;
7      ++n1; // 增加存储于函数对象的 n1 副本
8      ++n2; // 增加 main() 的 n2
9      //++n3; // 编译错误
10     std::cout << "In function end: n1[" << n1 << "]"    n2[" << n2 << "]"    n3[" << n3 << "]" << std::endl;
11 }
12
13 int main()
14 {
15     int n1 = 1, n2 = 1, n3 = 1;
16     std::cout << "Before function: n1[" << n1 << "]"    n2[" << n2 << "]"    n3[" << n3 << "]" << std::endl;
17     std::function<void()> bound_f = std::bind(f, n1, std::ref(n2), std::cref(n3));
18     bound_f();
19     std::cout << "After function: n1[" << n1 << "]"    n2[" << n2 << "]"    n3[" << n3 << "]" << std::endl;
20 }
```

运行结果：

```
1  Before function: n1[1]    n2[1]    n3[1]
2  In function: n1[1]    n2[1]    n3[1]
3  In function end: n1[2]    n2[2]    n3[1]
4  After function: n1[1]    n2[2]    n3[1]
```

分析

n1是值传递，函数内部的修改对外面没有影响。

n2是引用传递，函数内部的修改影响外面。

n3是const引用传递，函数内部不能修改。