

# C++ 进阶用法

ZeitHaum

2023 年 2 月 27 日

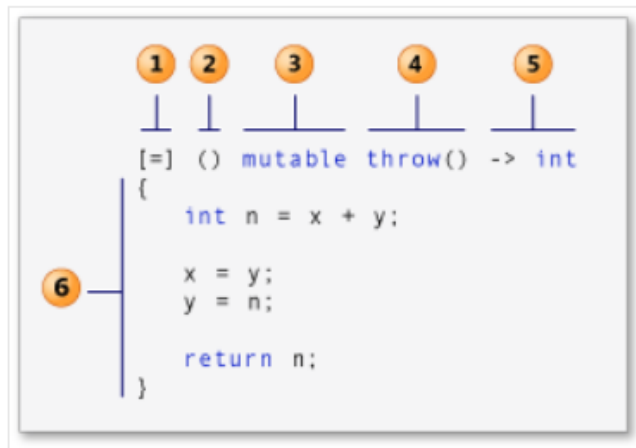
# 目录

<b>1 C++ lambda 表达式</b>	<b>1</b>
1.1 capture 子句	1
1.2 返回类型	2
1.3 mutable	2
<b>2 迭代器</b>	<b>2</b>
2.1 分类	2
2.2 不同容器中的迭代器	3
2.3 辅助函数	4
<b>3 algorithm 库中常用函数</b>	<b>4</b>
3.1 all_of	5
3.2 any_of	5
3.3 none_of	5
3.4 for_each	5
3.5 generate	6
3.6 generate_n	7
3.7 includes	7
3.8 inplace_merge	7
3.9 is_heap	8
3.10 is_heap_until	8
3.11 is_partitioned	8
<b>参考文献</b>	<b>8</b>

## 1 C++ lambda 表达式

C++ 11 以上特性。

下图显示了 lambda 语法的各个部分：



1. capture 子句（在 C++ 规范中也称为 Lambda 表达式捕获子句）。
2. 参数列表（可选）。（也称为 Lambda 声明符）。
3. mutable 规范（可选）。
4. exception-specification（可选）。
5. trailing-return-type（可选）。
6. Lambda 体。

### 1.1 capture 子句

用于访问（捕获）外部变量。`[=]` 用于值捕获，`[&]` 用于引用捕获，`[this]` 捕获外部类指针 `this`（`[&]` 默认包含 `[this]`）。补充：`[args...]` 用于捕获外部可变

参数模板。

## 1.2 返回类型

编译器自动推导。也可以使用关键字 “->” 指定，此时不能省略空参数列表。

## 1.3 mutable

在按值捕获时无法在作用域内修改外部变量的值，使用 mutable 修饰后可以解决这个问题。但是修改仅限于 lambda 表达式内部生效。

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     int m = 1;
6     int n = 0;
7     [=] () mutable {
8         n++;
9         cout<<"The value of n is changed to "<<n<<". "<<endl;
10    }(); //Parentheses at the end indicate the default call.
11    cout<<"The value of n is "<<n<<". "<<endl;
12 }
```

输出结果为

```
The value of n is changed to 1.
The value of n is 0.
```

## 2 迭代器

迭代器用于访问顺序容器 (主要是 vector 和数组)。

### 2.1 分类

分为正向迭代器、常量正向迭代器、反向迭代器、常量反向迭代器。比较常用的是正向迭代器和反向迭代器，反向迭代器的开始和结束分别为 rbegin() 和 rend()。

二者区别在于正向迭代器 ++ 返回顺序容器后一个数，后者返回前一个数。以下是二者使用的一个例子：

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     vector<int> arr {1,2,3,4,5,6,7,8,9};
6     cout<<"The elements of the array are ";
7     for(vector<int>::iterator i = arr.begin();i<arr.end();i++){
8         cout<<*i<<" ";
9     }
10    cout<<"."<<endl;
11    cout<<"The elements of the reversed array are ";
12    for(vector<int>::reverse_iterator i = arr.rbegin();i<arr.
13        rend();i++){
14        cout<<*i<<" ";
15    }
16    cout<<"."<<endl;
17 }
```

输出结果为

```
The elements of the array are 1 2 3 4 5 6 7 8 9 .
The elements of the reversed array are 9 8 7 6 5 4 3 2 1 .
```

数组的迭代器就是指针，使用“数组名 +N”的形式表示。

## 2.2 不同容器中的迭代器

根据容器类型的不同，可将迭代器分为正向迭代器、双向迭代器、随机访问迭代器，限制依次呈递减趋势。其中数组和 vector 的迭代器都是随机访问迭代器。其支持的功能如下：

- $p+=i$ : 使得  $p$  往后移动  $i$  个元素。
- $p-=i$ : 使得  $p$  往前移动  $i$  个元素。
- $p+i$ : 返回  $p$  后面第  $i$  个元素的迭代器。
- $p-i$ : 返回  $p$  前面第  $i$  个元素的迭代器。
- $p[i]$ : 返回  $p$  后面第  $i$  个元素的引用。

另外  $p_2 - p_1$  和  $p_1 < p_2$  均有定义 (和索引类似)。  
不同容器的迭代器类型如下:

容器	迭代器功能
vector	随机访问
deque	随机访问
list	双向
set / multiset	双向
map / multimap	双向
stack	不支持迭代器
queue	不支持迭代器
priority_queue	不支持迭代器

## 2.3 辅助函数

C++ 中关于迭代器的辅助函数为 `advance`、`distance`、`iter_swap`; 其功能如下:

STL 中有用于操作迭代器的三个函数模板, 它们是:

- `advance(p, n)`: 使迭代器 `p` 向前或向后移动 `n` 个元素。
- `distance(p, q)`: 计算两个迭代器之间的距离, 即迭代器 `p` 经过多少次 `++` 操作后和迭代器 `q` 相等。如果调用时 `p` 已经指向 `q` 的后面, 则这个函数会陷入死循环。
- `iter_swap(p, q)`: 用于交换两个迭代器 `p`、`q` 指向的值。

## 3 algorithm 库中常用函数

`algorithm` 是 C++ 标准库之一, 需使用 `using namespace std;` 语句引入名称空间。

`algorithm` 库函数具有丰富的可扩展性, 这些需要使用 `lambda` 表达式和迭代器实现。

### 3.1 all\_of

对列表中的元素执行谓词，如果都为真返回 true.

例子:

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     //Check the elements in the vector whether are all even
6     //numbers.
7     vector<int> arr {1,3,5,7,9,11};
8     vector<int> arr2 {2,4,6,8,10};
9
10    auto even_check = [](vector<int>& arr){
11        if(all_of(arr.begin(),arr.end(), [&](int i){return i
12            %2==1;})) cout<<"Check passed."<<endl;
13        else cout<<"Check unpassed."<<endl;
14    };
15    even_check(arr);
16    even_check(arr2);
17 }
```

输出:

```
Check passed.
Check unpassed.
```

### 3.2 any\_of

和 all\_of 区别是有任一为真即返回 true.

### 3.3 none\_of

无一为真返回 true.

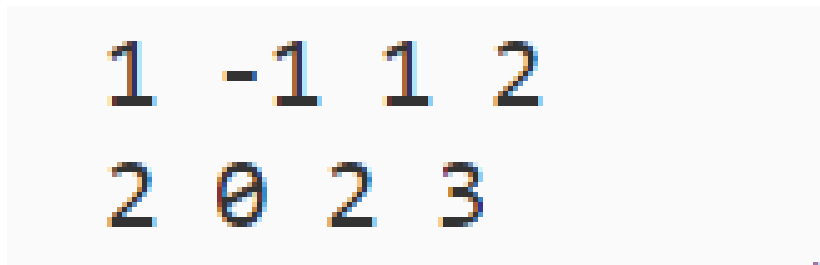
### 3.4 for\_each

为每个函数执行操作，输入可以是函数指针也可以是 lambda 表达式。

例子:

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define int long long
4
5 //function 1
6 void func1(long long &i){
7     i--;
8 }
9
10 signed main(){
11     auto func2 = [&](long long &i){
12         i++;
13     };
14     vector<int> arr{2,0,2,3};
15     auto print_arr = [&]()for_each(arr.begin(),arr.end(),[](
16         int i){cout<<" "<<i;});};//Print the array.
17
18     for_each(arr.begin(),arr.end(),func1);
19     print_arr();
20     cout<<endl;
21     for_each(arr.begin(),arr.end(),func2);
22     print_arr();
23     cout<<endl;
24 }
```

输出结果为



### 3.5 generate

类似于 `for_each`, 只是更新方式由参数指针修改变为返回值赋值。



### 3.6 generate\_n

类似于 generate, 只是结束迭代器被换为了大小 n。(从开始迭代器开始, 包含开始迭代器。)

### 3.7 includes

对于两个**已经排序好的 (增序)** 的范围 [first1,last1) 和 [first2,last2), 检查对于 [first2,last2) 中的元素, 是否**所有的**元素都被包含在 [first1,last1) 中。

若 [first1,last1) 为空, C++98 返回不确定值, 而 C++11 返回真。

例子:

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(){
5     int a[] = {1,2,3,4,5,6,7,8,9};
6     int b[] = {};
7     int c[] = {2,4,5,8};
8     int d[] = {7,8,9,10};
9     auto check = [&a](int* arr,int size){//Size is the size of
10         the array.
11         if (includes(a,a+9,arr,arr+size)) cout<<"All elements are
12             in the array."<<endl;
13         else cout<<"Some elements are not in the array."<<endl;
14     };
15     check(b,0);
16     check(c,4);
17     check(d,4);
18 }
```

输出结果为

```
All elements are in the array.
All elements are in the array.
Some elements are not in the array.
```

### 3.8 inplace\_merge

合并两部分**已经排好序**的数组, 不常用。

### 3.9 `is_heap`

检查一个数组是否是一个堆。

### 3.10 `is_heap_until`

返回一个数组第一个不满足堆性质的非法字符。

### 3.11 `is_partitioned`

若满足谓词性质的元素均在不满足谓词的元素前返回 `true`, 否则返回 `false`.

## 参考文献

- [1]. [Microsoft C++、C 和汇编程序](#)
- [2]. [C++ algorithm 头文件下常用函数整理](#)
- [3]. [C++ 迭代器 \(STL 迭代器\) iterator 详解](#)
- [4]. [cplusplus.com-algorithm](#)