C++ STL deque容器（详解版）

< 上一页 下一页 >

deque 是 double-ended queue 的缩写，又称双端队列容器。

前面章节中，我们已经系统学习了 vector 容器，值得一提的是，deque 容器和 vecotr 容器有很多相似之处，比如：

- deque 容器也擅长在序列尾部添加或删除元素（时间复杂度为 O(1) ），而不擅长在序列中间添加或删除元素。
- deque 容器也可以根据需要修改自身的容量和大小。

和 vector 不同的是，deque 还擅长在序列头部添加或删除元素，所耗费的时间复杂度也为常数阶 O(1) 。并且更重要的一点是，deque 容器中存储元素并不能保证所有元素都存储到连续的内存空间中。

> 当需要向序列两端频繁的添加或删除元素时，应首选 deque 容器。

deque 容器以模板类 deque<T>（T 为存储元素的类型）的形式在 <deque> 头文件中，并位于 std 命名空间中。因此，在使用该容器之前，代码中需要包含下面两行代码：

```
01. #include <deque>
02. using namespace std;
```

> 注意，std 命名空间也可以在使用 deque 容器时额外注明，两种方式都可以。

创建deque容器的几种方式

创建 deque 容器，根据不同的实际场景，可选择使用如下几种方式。

1) 创建一个没有任何元素的空 deque 容器：

```
01. std::deque<int> d;
```

和空 array 容器不同，空的 deque 容器在创建之后可以做添加或删除元素的操作，因此这种简单创建 deque 容器的方式比较常见。

2) 创建一个具有 n 个元素的 deque 容器，其中每个元素都采用对应类型的默认值：

```
01. std::deque<int> d(10);
```

此行代码创建一个具有 10 个元素（默认都为 0）的 deque 容器。

3) 创建一个具有 n 个元素的 deque 容器，并为每个元素都指定初始值，例如：

```
std::deque<int> d(10, 5)
```

如此就创建了一个包含 10 个元素（值都为 5）的 deque 容器。

4) 在已有 deque 容器的情况下，可以通过拷贝该容器创建一个新的 deque 容器，例如：

```
std::deque<int> d1(5);
std::deque<int> d2(d1);
```

注意，采用此方式，必须保证新旧容器存储的元素类型一致。

5) 通过拷贝其他类型容器中指定区域内的元素（也可以是普通数组），可以创建一个新容器，例如：

```
//拷贝普通数组，创建deque容器
int a[] = { 1,2,3,4,5 };
std::deque<int>d(a, a + 5);
//适用于所有类型的容器
std::array<int, 5>arr{ 11,12,13,14,15 };
std::deque<int>d(arr.begin()+2, arr.end());//拷贝arr容器中的{13,14,15}
```

deque容器可利用的成员函数

基于 deque 双端队列的特点，该容器包含一些 array、vector 容器都没有的成员函数。

表 1 中罗列了 deque 容器提供的所有成员函数。

表 1 deque 容器的成员函数

函数成员	函数功能
begin()	返回指向容器中第一个元素的迭代器。
end()	返回指向容器最后一个元素所在位置后一个位置的迭代器，通常和 begin() 结合使用。
rbegin()	返回指向最后一个元素的迭代器。
rend()	返回指向第一个元素所在位置前一个位置的迭代器。
cbegin()	和 begin() 功能相同，只不过在其基础上，增加了 const 属性，不能用于修改元素。
cend()	和 end() 功能相同，只不过在其基础上，增加了 const 属性，不能用于修改元素。
crbegin()	和 rbegin() 功能相同，只不过在其基础上，增加了 const 属性，不能用于修改元素。
crend()	和 rend() 功能相同，只不过在其基础上，增加了 const 属性，不能用于修改元素。

size()	返回实际元素个数。
max_size()	返回容器所能容纳元素个数的最大值。这通常是一个很大的值，一般是 2^{32}-1，我们很少会用到这个函数。
resize()	改变实际元素的个数。
empty()	判断容器中是否有元素，若无元素，则返回 true；反之，返回 false。
shrink _to_fit()	将内存减少到等于当前元素实际所使用的大小。
at()	使用经过边界检查的索引访问元素。
front()	返回第一个元素的引用。
back()	返回最后一个元素的引用。
assign()	用新元素替换原有内容。
push_back()	在序列的尾部添加一个元素。
push_front()	在序列的头部添加一个元素。
pop_back()	移除容器尾部的元素。
pop_front()	移除容器头部的元素。
insert()	在指定的位置插入一个或多个元素。
erase()	移除一个元素或一段元素。
clear()	移出所有的元素，容器大小变为 0。
swap()	交换两个容器的所有元素。
emplace()	在指定的位置直接生成一个元素。
emplace_front()	在容器头部生成一个元素。和 push_front() 的区别是，该函数直接在容器头部构造元素，省去了复制移动元素的过程。
emplace_back()	在容器尾部生成一个元素。和 push_back() 的区别是，该函数直接在容器尾部构造元素，省去了复制移动元素的过程。

> 和 vector 相比，额外增加了实现在容器头部添加和删除元素的成员函数，同时删除了 capacity()、reserve() 和 data() 成员函数。

和 array、vector 相同，C++ 11 标准库新增的 begin() 和 end() 这 2 个全局函数也适用于 deque 容器。这 2 个函数的操作对象既可以是容器，也可以是普通数组。当操作对象是容器时，它和容器包含的 begin() 和 end() 成员函数的功能完全相同；如果操作对象是普通数组，则 begin() 函数返回的是指向数组第一个元素的指针，同样 end() 返回指向数组中最后一个元素之后一个位置的指针（注意不是最后一个元素）。

deque 容器还有一个 `std::swap(x , y)` 非成员函数（其中 x 和 y 是存储相同类型元素的 deque 容器），它和 swap() 成员函数的功能完全相同，仅使用语法上有差异。

如下代码演示了表 1 中部分成员函数的用法：

```cpp
#include <iostream>
#include <deque>
using namespace std;
int main()
{
    //初始化一个空deque容量
    deque<int>d;
    //向d容器中的尾部依次添加 1，2,3
    d.push_back(1); //{1}
    d.push_back(2); //{1,2}
    d.push_back(3); //{1,2,3}
    //向d容器的头部添加 0
    d.push_front(0); //{0,1,2,3}

    //调用 size() 成员函数输出该容器存储的字符个数。
    printf("元素个数为：%d\n", d.size());

    //使用迭代器遍历容器
    for (auto i = d.begin(); i < d.end(); i++) {
        cout << *i << " ";
    }
    cout << endl;
    return 0;
}
```

运行结果为：

```
元素个数为：4
0 1 2 3
```

表 1 中这些成员函数的具体用法，后续学习用到时会具体讲解，感兴趣的读者，也可以通过查阅 STL手册做详细了解。

< 上一页　　下一页 >