std::list

目录

**[1. 简介](#_Toc1116_WPSOffice_Level1)** **[1](#_Toc1116_WPSOffice_Level1)**

**[2. 容器属性](#_Toc31181_WPSOffice_Level1)** **[1](#_Toc31181_WPSOffice_Level1)**

[2.1. 序列化](#_Toc31181_WPSOffice_Level2) [1](#_Toc31181_WPSOffice_Level2)

[2.2. 双向链表](#_Toc17633_WPSOffice_Level2) [2](#_Toc17633_WPSOffice_Level2)

[2.3. Allocator-aware](#_Toc12137_WPSOffice_Level2) [2](#_Toc12137_WPSOffice_Level2)

**[3. 模板参数](#_Toc17633_WPSOffice_Level1)** **[2](#_Toc17633_WPSOffice_Level1)**

[3.1. T](#_Toc7988_WPSOffice_Level2) [2](#_Toc7988_WPSOffice_Level2)

[3.2. Alloc](#_Toc18848_WPSOffice_Level2) [2](#_Toc18848_WPSOffice_Level2)

**[4. 成员变量](#_Toc12137_WPSOffice_Level1)** **[2](#_Toc12137_WPSOffice_Level1)**

**[5. 成员函数](#_Toc7988_WPSOffice_Level1)** **[3](#_Toc7988_WPSOffice_Level1)**

[5.1. 构造与析构相关](#_Toc1317_WPSOffice_Level2) [3](#_Toc1317_WPSOffice_Level2)

**[6. 重载的非成员函数](#_Toc18848_WPSOffice_Level1)** **[4](#_Toc18848_WPSOffice_Level1)**

# 简介

模板：template<class T, class Alloc = allocator<T> > class list;

list容器属于序列容器，可在序列的任意位置以常量时间进行插入与删除操作，并可进行双向迭代。

list容器实现为双向链表，双向链表能够将他们所包含的每一个元素存储在不同且不相关的存储区域。其内部的排序，通过每一个元素的前元素链接与后元素链接进行关联。

list与forward\_list很相似：主要的不同是，forward\_list是单向链表，因此他们只能单向迭代，以此换取更小的体积和更高的效率。

与其他基本的标准容器进行相比（array, vector和deque），lists在已获得迭代器的容器内部任意位置进行插入、删除和移动操作通常表现得更好，同样地在大量使用这些操作的算法中也一样，例如排序算法。

与其他序列容器相比，lists和forward\_list的主要缺点是：他们不能使用坐标对元素进行直接访问。例如，我们要访问一个list中的第6个元素，必须从一个已知的位置（像开始或结束的位置）一直迭代到那个位置。这个操作是线性复杂度的，消耗的时间取决于迭代的距离。他们也需要额外地消耗一些内存用来记录每个元素之间的指针信息，对于非常小的对象，这个消耗就会非常明显。

# 容器属性

## 序列化

序列化容器中的元素按照严格的线性顺序进行排序，内部的元素通过其在序列中的位置进行访问。

## 双向链表

每个元素记录着如何定位其前一个或后一个元素的信息，允许以常量时间在特定的元素（甚至整个范围）前后进行插入或删除操作，但是不能够直接随机访问。

## Allocator-aware

容器使用模板中的allocator对象动态地处理其内存分配。

# 模板参数

## T

list元素的类型。

作为成员变量list::value\_type的别名。

## Alloc

分配器的类型，用于定义内存分配模型。默认情况，分配器类模板用于定义最简单的内存分配模型，并且值独立。

作为成员变量list::allocator\_type的别名。

# 成员类型

|  |  |  |
| --- | --- | --- |
| **成员类型** | **定义** | **说明** |
| value\_type | 第一个模板参数 (T) |  |
| allocator\_type | 第二个模板参数 (Alloc) | 默认为: [allocator](http://www.cplusplus.com/allocator)<value\_type> |
| reference | value\_type& |  |
| const\_reference | const value\_type& |  |
| pointer | [allocator\_traits](http://www.cplusplus.com/allocator_traits)<allocator\_type>::pointer | 作为默认的分配器： value\_type\* |
| const\_pointer | [allocator\_traits](http://www.cplusplus.com/allocator_traits)<allocator\_type>::const\_pointer | 作为默认的分配器: const value\_type\* |
| iterator | a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) to value\_type | convertible to const\_iterator  双向迭代器 |
| const\_iterator | a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) to const value\_type | const类型的双向迭代器 |
| reverse\_iterator | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<iterator> | 反向迭代器 |
| const\_reverse\_iterator | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<const\_iterator> | const类型的反向迭代器 |
| difference\_type | 有符号整型，等同于: iterator\_traits<iterator>::difference\_type | 通常等同于[ptrdiff\_t](http://www.cplusplus.com/ptrdiff_t) |
| size\_type | 无符号整型，代表difference\_type的非负值 | 通常等同于[size\_t](http://www.cplusplus.com/size_t) |

# 成员函数

## 构造与析构相关

### 构造函数（C++11）

#### 构造函数列表

|  |  |
| --- | --- |
|  |  |
| default | explicit list(const allocator\_type& alloc = allocate\_type());  构造一个空容器，没有元素。 |
| fill | explicit list(size\_type n);  list(size\_type n, const value\_type& val, const allocator\_type& = allocator\_type());  构造一个有n个元素的容器，每一个元素是val的拷贝(如果有val的话) |
| range | template<class InputIterator>  list(InputIterator first, InputIterator last, const allocator\_type& alloc = allocator\_type());  使用和[first, last)一样多的元素构造一个容器，并且范围与顺序与范围内的元素保持一致。 |
| copy | list(const list& x);  list(const list& x, const allocator\_type& alloc);  构造一个容器，使用x中的每一个元素同样的排序作为拷贝 |
| move | list(list&& x);  如果指定了alloc，并且和x的allocator不一样，所有的element将被移走，否则没有元素被创建（他们的所有权被直接转移）。  x将变成不确定但有效的状态（即x可以使用，但是其值不确定）。 |
| initializer list | list(initializer\_list<value\_type> il, const allocator\_type& alloc = allocator\_type());  通过拷贝初始化列表中的内容，构造一个list容器。 |

容器保留一个alloc的内部拷贝，用于分配和释放元素所占用的内存，并用于构造和销毁元素。如果构造函数没有传递alloc参数，将会使用默认构造好的allocator，以下情况除外：

1. copy构造函数（copy中的第一个构造函数），创建一个容器，保留并使用通过调用selected\_on\_container\_copy\_construction（对x的allocator）所返回的拷贝。
2. move构造函数（move中的第一个构造函数），获取x的allocator。

通过适当的参数调用allocator\_traints::construct对所有的元素进行复制、移动或者创建。

#### 参数

1. alloc

Allocator对象。容器保留与使用这个分配器的内部拷贝。成员类型allocator\_type是被容器所使用的内部分配器的类型，在list中定义为他的第二个模板参数的别名。

1. n

初始化容器的大小（例如：创建容器的时候元素的个数）。

成员类型size\_type是一无符号整型。

1. val

填充容器所使用的值。容器n个元素中的每一个都被初始化为这个值的拷贝。成员类型type\_value是容器中的元素类型，在list中定义为他的第一个模板参数的别名。

1. first, last

输入迭代器范围的起始和终止位置。范围使用半开半闭区间[first, last)，包含的元素位于first和last之间，包括first位置的元素，但不包括last位置的元素。函数模板的参数InputIterator应该是input iterator类型，并且是value\_type对象能够通过其进行创建的元素类型。

1. x

相同类型的另一个list（具有相同的类模板参数），他的内容用来拷贝或获取。

1. il

一个initializer\_list对象。

这些对象将通过initializer list声明符号进行自动创建。

成员类型value\_type是容器中的元素类型，在list定义作为他的第一个模板参数（T）的别名。

#### 例子

// constructing lists#include <iostream>#include <list>

int main ()

{

// constructors used in the same order as described above:

std::list<int> first; // empty list of ints

std::list<int> second (4,100); // four ints with value 100

std::list<int> third (second.begin(),second.end()); // iterating through second

std::list<int> fourth (third); // a copy of third

// the iterator constructor can also be used to construct from arrays:

int myints[] = {16,2,77,29};

std::list<int> fifth (myints, myints + sizeof(myints) / sizeof(int) );

std::cout << "The contents of fifth are: ";

for (std::list<int>::iterator it = fifth.begin(); it != fifth.end(); it++)

std::cout << \*it << ' ';

std::cout << '\n';

return 0;

}

#### 复杂度

固定复杂度：对于默认构造函数和move构造函数（除非alloc与x的alloc不同）

线性复杂度：其他情况属于线性复杂度，与容器内元素个数对应。

#### 迭代器的有效性

对于move构造函数，如果x中的元素被移除，则与这些元素相关的迭代器、 指针和引用都是无效的。

#### 数据竞争

1. 所有拷贝过来的元素都是可访问的；
2. move构造函数会修改x;

#### 异常与安全

### 析构函数

### 赋值运算符重载

## 迭代器

### begin

#### 声明

1. iterator begin() noexcept;
2. const\_iterator begin() noexcept;

返回指向容器第一个元素的迭代器。

注意：不像list::front()返回的是容器第一个元素的引用，begin()返回的是一个指向元素的双向迭代器。

如果容器为空，则返回的迭代器的值不能够被间接引用。

#### 参数

无

#### 返回值

指向序列容器开始位置的迭代器。

如果list对象受const限制，则函数返回一个const\_iterator，否则返回iterator。

成员类型iterator和const\_iterator是***双向***迭代器类型。

#### 例子

### end

// list::begin#include <iostream>#include <list>

int main ()

{

int myints[] = {75,23,65,42,13};

std::list<int> mylist (myints,myints+5);

std::cout << "mylist contains:";

for (std::list<int>::iterator it=mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

return 0;

}

#### 声明

iterator end() noexcetp;

const\_iterator end() noexcept;

返回list容器past-the-end元素的参考迭代器。Past-the-end元素是一个理论上的元素，紧接着容器内的最后一个元素，但他不指向任何元素，所以不能被间接引用。

如果容器为空，则end()返回与begin()返回相同。通常end()用来与begin()结合一起使用，用于判断是否已经超出了容器的界限。

### rbegin

#### 声明

reverse\_iterator rbegin() noexcept;

const\_reverse\_iterator rbegin() noexcept;

返回一个反向迭代器，指向容器内的最后一个元素。

### rend

### cbegin(C++11)

### cend(C++11)

### crbegin(C++11)

### crend(C++11)

# 重载的非成员函数