参考：<https://zh.cppreference.com/w/%E9%A6%96%E9%A1%B5>

主要基于C++ 98

# 容器

## vector

头文件：<vector>

模板：template<class T, class Allocator = std::allocator<T>> class vector;

std::vector是封装动态数组的顺序容器。

元素相继存储，这意味着不仅可通过迭代器，还能用指向元素的常规指针访问元素。这意味着指向 vector 元素的指针能传递给任何期待指向数组元素的指针的函数。

vector 的存储是自动管理的，按需扩张收缩。 vector 通常占用多于静态数组的空间，因为要分配更多内存以管理将来的增长。vector 所用的方式不在每次插入元素时，而只在额外内存耗尽时重分配。分配的内存总量可用 [capacity()](https://zh.cppreference.com/w/cpp/container/vector/capacity" \o "cpp/container/vector/capacity) 函数查询。

重分配通常是性能上有开销的操作。若元素数量已知，则 [reserve()](https://zh.cppreference.com/w/cpp/container/vector/reserve" \o "cpp/container/vector/reserve) 函数可用于消除重分配。

vector 上的常见操作复杂度（效率）如下：

随机访问——常数 O(1)

在末尾插入或移除元素——均摊常数 O(1)

插入或移除元素——与到 vector 结尾的距离成线性 O(n)

### 元素访问

at, operator[], front, back, data

其中只有at会做边界检查并抛出std::out\_of\_range的异常，其他几个都不会，所以在使用的时候需要对其他几个做边界的检查，否则可能会导致程序崩溃。

其中data返回的是指针，其他返回的都是引用。

对于容器 c ，表达式 c.front() 等价于 \*c.begin() 。

### 迭代器

begin, cbegin, end, cend, rbegin, crbegin, rend, crend

其中begin, cbegin, end,cend都返回的是双向迭代器（iterator或const\_iterator）,其他都返回的是逆向迭代器（reverse\_iterator或const\_reverse\_iterator）。

另外，只有带begin的返回的迭代器才是真正指向了合法元素（空容器除外），带end的返回的迭代器都是最后一个合法元素的后一个元素。

对于空容器，begin的返回与end一样，都指向不合法的元素。

### 容量

empty, size, max\_size, reserve, capacity, shrink\_to\_fit(C++11)

其中max\_size()返回的是系统或库可提供的最大capacity，与capacity不同的是capacity是当前已经为容器分配好的最大可使用容量，capacity可根据内容大小而变动，但是max\_size一般是系统或库决定并不可改变的。

reserve(size\_type new\_cap)：人为决定capacity的值，需要new\_cap大于当前的capacity，否则不生效。

shrink\_to\_fit()：（C++11）请求移除未使用的容量，这个请求只是尝试性，不一定会生效。

### 修改器

clear, insert, emplace(C++11), erase, push\_back, emplace\_back, pop\_back, resize, swap

其中resize(size\_type count, T value = T())主要是改变size()的大小，当count < size()时，capacity()不变，同时移除部分元素。

pop\_back()是移除最后一个有效元素。

需要注意的是迭代器在执行修改操作之后可能会部分或全部失效的问题。比如insert()操作之后，其insert()前插入位置之后的迭代器全部失效，如果capacity变化导致所有内存重分配，则所有迭代器都失效。

另外，需要注意的是很多都有返回值，可以使用。

## list

头文件：<list>

原型：template<class T,  class Allocator = [std::allocator](http://zh.cppreference.com/w/cpp/memory/allocator)<T>> class list;

std::list 是支持常数时间从容器任何位置插入和移除元素的容器。不支持快速随机访问。它通常实现为双向链表。与 [std::forward\_list](https://zh.cppreference.com/w/cpp/container/forward_list" \o "cpp/container/forward list) 相比，此容器提供双向迭带但在空间上效率稍低。

在 list 内或在数个 list 间添加、移除和移动元素不会非法化迭代器或引用。迭代器仅在对应元素被删除时非法化。

### 元素访问

front, back

分别返回容器第一个元素和最后一个元素的引用。

这里需要与vector访问方法进行比较，并能知道为什么vector多at,operator[]等访问方法。

### 迭代器

begin, cbegin, end, cend, rbegin, crbegin, rend, crend

与<vector>一样

### 容量

empty, size, max\_size

list与vector相比，没有reserve, capacity, shrink\_to\_fit这些容量方法，这与他们的存储结构相关。vector的存储是连续空间存储，而list的存储不需要各元素在内存空间上连续，所以也就不存在与reserve,capacity,shrink\_to\_fit相关的方法。

### 修改器

clear, insert, emplace(C++11), erase, push\_back, emplace\_back(C++11), pop\_back, **push\_front**, **pop\_front**, emplace\_front, resize, swap

与vector不同的是，C++ 98中的list，有push\_front和pop\_front两种方法，但是vector没有，这与与vector连续存储结构相关，不方便这样的操作。

另外，list很多修改操作都没有或很局部地造成迭代器非法，也与list的非连续存储结构有关系。

还有就是要注意不要越界操作，否则有些情况下没有异常抛出，导致程序直接崩溃。

### 操作

***list的操作方法，很多与排序相关，这些是vector所没有的，也与vector连续存储结构有关。***

merge, splice, remove, remove\_if, reverse, unique, sort

merge：归并两个已排序容器为一个。链表应以升序进行排序。不复制元素，操作之后参数中的容器变为空。

splice：将参数中的list按照参数中指定的位置与list合并为一个。不复制或移动元素。

remove：移除满足特定标准的元素。

reverse：逆转容器中的元素顺序。

unique：删除连续的重复元素。

sort：以升序排序元素，保持相等元素的顺序。

## map

头文件：<map>

template<class Key, class T, class Compare = [std::less](http://zh.cppreference.com/w/cpp/utility/functional/less)<Key>, class Allocator = [std::allocator](http://zh.cppreference.com/w/cpp/memory/allocator)<[std::pair](http://zh.cppreference.com/w/cpp/utility/pair)<const Key, T> > > class map;

std::map 是有序键值对容器，它的元素的键是唯一的。用比较函数 Compare 排序键。搜索、移除和插入操作拥有对数复杂度。 map 通常实现为[红黑树](https://en.wikipedia.org/wiki/Red%E2%80%93black_tree" \o "enwiki:Red–black tree)。

在每个标准库使用[比较](https://zh.cppreference.com/w/cpp/named_req/Compare" \o "cpp/named req/Compare) (Compare) 概念的位置，唯一性以等价关系检验。不精确而言，若二个对象 a 与 b 互相比较不小于对方 ： !comp(a, b) && !comp(b, a) ，则认为它们等价（非唯一）。

### 元素访问

at(C++11), operator[]

map的实现是红黑树，所以没有vector和list容器中front,back这样的访问方法；并且容器内元素的键是唯一的，所以可以有at, operator[key]这样的访问方法。

### 迭代器

begin, cbegin, end, cend, rbegin, crbegin, rend, crend

与list和vector容器一样。

### 容量

empty, size, max\_size

与list一样。***说明map中的元素存储空间上也不要求是连续的。***

### 修改器

clear, insert, insert\_or\_assign, emplace(C++11), emplace\_hint(C++11), try\_emplace(C++17), erase, extract(C++17), merge(C++17)

注意，这里map中的key中唯一的。所以Insert操作，只有map中不存在要插入的key的时候才播放成功。而Insert\_or\_assign，则是替换或追加。

因为Map结构的特殊性，map的修改器中并没有push\_back, pop\_back, push\_front, pop\_front这些操作。

map也没有***resize***操作。应该是因为map是关联型容器，不像vector和list是顺序容器，可以方便实现resize操作。

### 查找

***顺序容器没有查找的这些方法。***

count：返回0或1

find：寻找带有特定键的元素，返回iterator

contains(C++20)：检查容器中是否有key键的元素

equal\_range：返回容器中所有拥有给定关键的元素范围。范围以二个迭代器定义，一个指向首个不小于 key 的元素，另一个指向首个大于 key 的元素。首个迭代器可以换用 [lower\_bound()](https://zh.cppreference.com/w/cpp/container/map/lower_bound" \o "cpp/container/map/lower bound) 获得，而第二迭代器可换用 [upper\_bound()](https://zh.cppreference.com/w/cpp/container/map/upper_bound" \o "cpp/container/map/upper bound) 获得。

lower\_bound：返回指向首个不小于给定键的元素的迭代器。

upper\_bound：返回指向首个大于给定键的元素的迭代器。

### 观察器

key\_comp, value\_comp

key\_comp：返回用于比较键的函数对象，它是此容器[构造函数](https://zh.cppreference.com/w/cpp/container/map/map" \o "cpp/container/map/map)参数 comp 的副本。

value\_comp：返回比较 std::map::value\_type （关键-值 pair ）对象的函数对象，它用 [key\_comp](https://zh.cppreference.com/w/cpp/container/map/key_comp" \o "cpp/container/map/key comp) 比较 pair 的第一组分。

## queue

## dequeue

# 算法