# 容器（containers）

## array

array 是固定大小的顺序容器，它们保存了一个以严格的线性顺序排列的特定数量的元素。

|  |  |
| --- | --- |
| 方法 | 含义 |
| begin | 返回指向数组容器中第一个元素的迭代器 |
| end | 返回指向数组容器中最后一个元素之后的理论元素的迭代器 |
| rbegin | 返回指向数组容器中最后一个元素的反向迭代器 |
| rend | 返回一个反向迭代器，指向数组中第一个元素之前的理论元素 |
| cbegin | 返回指向数组容器中第一个元素的常量迭代器（const\_iterator） |
| cend | 返回指向数组容器中最后一个元素之后的理论元素的常量迭代器（const\_iterator） |
| crbegin | 返回指向数组容器中最后一个元素的常量反向迭代器（const*reverse*iterator） |
| crend | 返回指向数组中第一个元素之前的理论元素的常量反向迭代器（const*reverse*iterator） |
| size | 返回数组容器中元素的数量 |
| max\_size | 返回数组容器可容纳的最大元素数 |
| empty | 返回一个布尔值，指示数组容器是否为空 |
| operator[] | 返回容器中第 n（参数）个位置的元素的引用 |
| at | 返回容器中第 n（参数）个位置的元素的引用 |
| front | 返回对容器中第一个元素的引用 |
| back | 返回对容器中最后一个元素的引用 |
| data | 返回指向容器中第一个元素的指针 |
| fill | 用 val（参数）填充数组所有元素 |
| swap | 通过 x（参数）的内容交换数组的内容 |
| get（array） | 形如 std::get<0>(myarray)；传入一个数组容器，返回指定位置元素的引用 |
| relational operators (array) | 形如 arrayA > arrayB；依此比较数组每个元素的大小关系 |

## vector

vector 是表示可以改变大小的数组的序列容器。

|  |  |
| --- | --- |
| 方法 | 含义 |
| vector | 构造函数 |
| ~vector | 析构函数，销毁容器对象 |
| operator= | 将新内容分配给容器，替换其当前内容，并相应地修改其大小 |
| begin | 返回指向容器中第一个元素的迭代器 |
| end | 返回指向容器中最后一个元素之后的理论元素的迭代器 |
| rbegin | 返回指向容器中最后一个元素的反向迭代器 |
| rend | 返回一个反向迭代器，指向中第一个元素之前的理论元素 |
| cbegin | 返回指向容器中第一个元素的常量迭代器（const\_iterator） |
| cend | 返回指向容器中最后一个元素之后的理论元素的常量迭代器（const\_iterator） |
| crbegin | 返回指向容器中最后一个元素的常量反向迭代器（const*reverse*iterator） |
| crend | 返回指向容器中第一个元素之前的理论元素的常量反向迭代器（const*reverse*iterator） |
| size | 返回容器中元素的数量 |
| max\_size | 返回容器可容纳的最大元素数 |
| resize | 调整容器的大小，使其包含 n（参数）个元素 |
| capacity | 返回当前为 vector 分配的存储空间（容量）的大小 |
| empty | 返回 vector 是否为空 |
| reserve | 请求 vector 容量至少足以包含 n（参数）个元素 |
| shrink*to*fit | 要求容器减小其 capacity（容量）以适应其 size（元素数量） |
| operator[] | 返回容器中第 n（参数）个位置的元素的引用 |
| at | 返回容器中第 n（参数）个位置的元素的引用 |
| front | 返回对容器中第一个元素的引用 |
| back | 返回对容器中最后一个元素的引用 |
| data | 返回指向容器中第一个元素的指针 |
| assign | 将新内容分配给 vector，替换其当前内容，并相应地修改其 size |
| push\_back | 在容器的最后一个元素之后添加一个新元素 |
| pop\_back | 删除容器中的最后一个元素，有效地将容器 size 减少一个 |
| insert | 通过在指定位置的元素之前插入新元素来扩展该容器，通过插入元素的数量有效地增加容器大小 |
| erase | 从 vector 中删除单个元素（position）或一系列元素（[first，last)），这有效地减少了被去除的元素的数量，从而破坏了容器的大小 |
| swap | 通过 x（参数）的内容交换容器的内容，x 是另一个类型相同、size 可能不同的 vector 对象 |
| clear | 从 vector 中删除所有的元素（被销毁），留下 size 为 0 的容器 |
| emplace | 通过在 position（参数）位置处插入新元素 args（参数）来扩展容器 |
| emplace\_back | 在 vector 的末尾插入一个新的元素，紧跟在当前的最后一个元素之后 |
| get\_allocator | 返回与vector关联的构造器对象的副本 |
| swap(vector) | 容器 x（参数）的内容与容器 y（参数）的内容交换。两个容器对象都必须是相同的类型（相同的模板参数），尽管大小可能不同 |
| relational operators (vector) | 形如 vectorA > vectorB；依此比较每个元素的大小关系 |

## deque

deque（['dek]）（双端队列）是double-ended queue 的一个不规则缩写。deque是具有动态大小的序列容器，可以在两端（前端或后端）扩展或收缩。

|  |  |
| --- | --- |
| 方法 | 含义 |
| deque | 构造函数 |
| push\_back | 在当前的最后一个元素之后 ，在 deque 容器的末尾添加一个新元素 |
| push\_front | 在 deque 容器的开始位置插入一个新的元素，位于当前的第一个元素之前 |
| pop\_back | 删除 deque 容器中的最后一个元素，有效地将容器大小减少一个 |
| pop\_front | 删除 deque 容器中的第一个元素，有效地减小其大小 |
| emplace\_front | 在 deque 的开头插入一个新的元素，就在其当前的第一个元素之前 |
| emplace\_back | 在 deque 的末尾插入一个新的元素，紧跟在当前的最后一个元素之后 |

## forward\_list

forward\_list（单向链表）是序列容器，允许在序列中的任何地方进行恒定的时间插入和擦除操作。

|  |  |
| --- | --- |
| 方法 | 含义 |
| forward\_list | 返回指向容器中第一个元素之前的位置的迭代器 |
| cbefore\_begin | 返回指向容器中第一个元素之前的位置的 const\_iterator |

## list

list，双向链表，是序列容器，允许在序列中的任何地方进行常数时间插入和擦除操作，并在两个方向上进行迭代。

## stack

stack 是一种容器适配器，用于LIFO（后进先出）的操作，其中元素仅从容器的一端插入和提取。

## queue

queue 是一种容器适配器，用于在FIFO（先入先出）的操作，其中元素插入到容器的一端并从另一端提取。

## priority\_queue

## set

set 是按照特定顺序存储唯一元素的容器。

## multiset

## map

map 是关联容器，按照特定顺序存储由 key value (键值) mapped value (映射值) 组合形成的元素。

|  |  |
| --- | --- |
| 方法 | 含义 |
| map | 构造函数 |
| begin | 返回引用容器中第一个元素的迭代器 |
| key\_comp | 返回容器用于比较键的比较对象的副本 |
| value\_comp | 返回可用于比较两个元素的比较对象，以获取第一个元素的键是否在第二个元素之前 |
| find | 在容器中搜索具有等于 k（参数）的键的元素，如果找到则返回迭代器，否则返回 map::end 的迭代器 |
| count | 在容器中搜索具有等于 k（参数）的键的元素，并返回匹配的数量 |
| lower\_bound | 返回一个非递减序列 [first, last)（参数）中的第一个大于等于值 val（参数）的位置的迭代器 |
| upper\_bound | 返回一个非递减序列 [first, last)（参数）中第一个大于 val（参数）的位置的迭代器 |
| equal\_range | 获取相同元素的范围，返回包含容器中所有具有与 k（参数）等价的键的元素的范围边界（pair< map<char,int>::iterator, map<char,int>::iterator >） |

## multimap

## unordered\_set

## unordered\_multiset

## unordered\_map

## unordered\_multimap

## tuple

元组是一个能够容纳元素集合的对象。每个元素可以是不同的类型。

## pair

这个类把一对值（values）结合在一起，这些值可能是不同的类型（T1 和 T2）。每个值可以被公有的成员变量first、second访问。

算 法

// 简单查找算法，要求输入迭代器（input iterator）  
find(beg, end, val); //返回一个迭代器指向输入序列中第一个等于 val 的元素，未找到返回 end  
find\_if(beg, end, unaryPred); //返回迭代器，指向第一个满足 unaryPred 的元素，未找到返回 end  
find\_if\_not(beg, end, unaryPred); // 返回迭代器，指向第一个令 unaryPred 为 false 的元素，未找到返回 end  
count(beg, end, val); // 返回一个计数器，指出 val 出现了多少次  
count\_if(beg, end, unaryPred); // 统计有多少个元素满足 unaryPred  
all\_of(beg, end, unaryPred); // 返回一个 bool 值，判断是否所有元素都满足 unaryPred  
any\_of(beg, end, unaryPred); // 返回一个 bool 值，判断是否任意（存在）一个元素满足 unaryPred  
none\_of(beg, end, unaryPred); // 返回一个 bool 值，判断是否所有元素都不满足 unaryPred

// 查找重复值的算法，传入向前迭代器（forward iterator）

adjacent\_find(beg, end); // 返回指向第一对相邻重复元素的迭代器，无相邻元素则返回 end  
adjacent\_find(beg, end, binaryPred); // 返回指向第一对相邻重复元素的迭代器，无相邻元素则返回 end  
search\_n(beg, end, count, val); // 返回迭代器，从此位置开始有 count 个相等元素，不存在则返回 end  
search\_n(beg, end, count, val, binaryPred); // 返回迭代器，从此位置开始有 count 个相等元素，不存在则返回 end

// 查找子序列算法，除 find\_first\_of（前两个输入迭代器，后两个前向迭代器） 外，都要求两个前向迭代器  
search(beg1, end1, beg2, end2); // 返回第二个输入范围（子序列）在爹一个输入范围中第一次出现的位置，未找到则返回 end1  
search(beg1, end1, beg2, end2, binaryPred); // 返回第二个输入范围（子序列）在爹一个输入范围中第一次出现的位置，未找到则返回 end1  
find\_first\_of(beg1, end1, beg2, end2); // 返回迭代器，指向第二个输入范围中任意元素在第一个范围中首次出现的位置，未找到则返回end1  
find\_first\_of(beg1, end1, beg2, end2, binaryPred); // 返回迭代器，指向第二个输入范围中任意元素在第一个范围中首次出现的位置，未找到则返回end1  
find\_end(beg1, end1, beg2, end2); // 类似 search，但返回的最后一次出现的位置。如果第二个输入范围为空，或者在第一个输入范围为空，或者在第一个输入范围中未找到它，则返回 end1  
find\_end(beg1, end1, beg2, end2, binaryPred); // 类似 search，但返回的最后一次出现的位置。如果第二个输入范围为空，或者在第一个输入范围为空，或者在第一个输入范围中未找到它，则返回 end1

// 其他只读算法，传入输入迭代器  
for\_each(beg, end, unaryOp); // 对输入序列中的每个元素应用可调用对象 unaryOp，unaryOp 的返回值被忽略  
mismatch(beg1, end1, beg2); // 比较两个序列中的元素。返回一个迭代器的 pair，表示两个序列中第一个不匹配的元素  
mismatch(beg1, end1, beg2, binaryPred); // 比较两个序列中的元素。返回一个迭代器的 pair，表示两个序列中第一个不匹配的元素  
equal(beg1, end1, beg2); // 比较每个元素，确定两个序列是否相等。  
equal(beg1, end1, beg2, binaryPred); // 比较每个元素，确定两个序列是否相等。

// 二分搜索算法，传入前向迭代器或随机访问迭代器（random-access iterator），要求序列中的元素已经是有序的。通过小于运算符（<）或 comp 比较操作实现比较。  
lower\_bound(beg, end, val); // 返回一个非递减序列 [beg, end) 中的第一个大于等于值 val 的位置的迭代器，不存在则返回 end  
lower\_bound(beg, end, val, comp); // 返回一个非递减序列 [beg, end) 中的第一个大于等于值 val 的位置的迭代器，不存在则返回 end  
upper\_bound(beg, end, val); // 返回一个非递减序列 [beg, end) 中第一个大于 val 的位置的迭代器，不存在则返回 end  
upper\_bound(beg, end, val, comp); // 返回一个非递减序列 [beg, end) 中第一个大于 val 的位置的迭代器，不存在则返回 end  
equal\_range(beg, end, val); // 返回一个 pair，其 first 成员是 lower\_bound 返回的迭代器，其 second 成员是 upper\_bound 返回的迭代器  
binary\_search(beg, end, val); // 返回一个 bool 值，指出序列中是否包含等于 val 的元素。对于两个值 x 和 y，当 x 不小于 y 且 y 也不小于 x 时，认为它们相等。

// 只写不读算法，要求输出迭代器（output iterator）  
fill(beg, end, val); // 将 val 赋予每个元素，返回 void  
fill\_n(beg, cnt, val); // 将 val 赋予 cnt 个元素，返回指向写入到输出序列最有一个元素之后位置的迭代器  
genetate(beg, end, Gen); // 每次调用 Gen() 生成不同的值赋予每个序列，返回 void  
genetate\_n(beg, cnt, Gen); // 每次调用 Gen() 生成不同的值赋予 cnt 个序列，返回指向写入到输出序列最有一个元素之后位置的迭代器  
// 使用输入迭代器的写算法，读取一个输入序列，将值写入到一个输出序列（dest）中  
copy(beg, end, dest); // 从输入范围将元素拷贝所有元素到 dest 指定定的目的序列  
copy\_if(beg, end, dest, unaryPred); // 从输入范围将元素拷贝满足 unaryPred 的元素到 dest 指定定的目的序列  
copy\_n(beg, n, dest); // 从输入范围将元素拷贝前 n 个元素到 dest 指定定的目的序列  
move(beg, end, dest); // 对输入序列中的每个元素调用 std::move，将其移动到迭代器 dest 开始始的序列中  
transform(beg, end, dest, unaryOp); // 调用给定操作（一元操作），并将结果写到dest中  
transform(beg, end, beg2, dest, binaryOp); // 调用给定操作（二元操作），并将结果写到dest中  
replace\_copy(beg, end, dest, old\_val, new\_val); // 将每个元素拷贝到 dest，将等于 old\_val 的的元素替换为 new\_val  
replace\_copy\_if(beg, end, dest, unaryPred, new\_val); // 将每个元素拷贝到 dest，将满足 unaryPred 的的元素替换为 new\_val  
merge(beg1, end1, beg2, end2, dest); // 两个输入序列必须都是有序的，用 < 运算符将合并后的序列写入到 dest 中  
merge(beg1, end1, beg2, end2, dest, comp); // 两个输入序列必须都是有序的，使用给定的比较操作（comp）将合并后的序列写入到 dest 中

// 使用前向迭代器的写算法，要求前向迭代器  
iter\_swap(iter1, iter2); // 交换 iter1 和 iter2 所表示的元素，返回 void  
swap\_ranges(beg1, end1, beg2); // 将输入范围中所有元素与 beg2 开始的第二个序列中所有元素进行交换。返回递增后的的 beg2，指向最后一个交换元素之后的位置。  
replace(beg, end, old\_val, new\_val); // 用 new\_val 替换等于 old\_val 的每个匹配元素  
replace\_if(beg, end, unaryPred, new\_val); // 用 new\_val 替换满足 unaryPred 的每个匹配元素  
// 使用双向迭代器的写算法，要求双向选代器（bidirectional iterator）  
copy\_backward(beg, end, dest); // 从输入范围中拷贝元素到指定目的位置。如果范围为空,则返回值为 dest；否则，返回值表示从 \*beg 中拷贝或移动的元素。  
move\_backward(beg, end, dest); //从输入范围中移动元素到指定目的位置。如果范围为空,则返回值为 dest；否则,返回值表示从 \*beg 中拷贝或移动的元素。  
inplace\_merge(beg, mid, end); // 将同一个序列中的两个有序子序列合并为单一的有序序列。beg 到 mid 间的子序列和 mid 到 end 间的子序列被合并，并被写入到原序列中。使用 < 比较元素。  
inplace\_merge(beg, mid, end, comp); // 将同一个序列中的两个有序子序列合并为单一的有序序列。beg 到 mid 间的子序列和 mid 到 end 间的子序列被合并，并被写入到原序列中。使用给定的 comp 操作。

// 划分算法，要求双向选代器（bidirectional iterator）  
is\_partitioned(beg, end, unaryPred); //如果所有满足谓词unaryPred的元素都在不满足unarypred 的元素之前，则返回 true。若序列为空，也返回 true  
partition\_copy(beg, end, dest1, dest2, unaryPred); // 将满足 unaryPred 的元素拷贝到到 dest1，并将不满足 unaryPred 的元素拷贝到到 dest2。返回一个迭代器 pair，其 first 成员表示拷贝到 dest1 的的元素的末尾，second 表示拷贝到 dest2 的元素的末尾。  
partitioned\_point(beg, end, unaryPred); // 输入序列必须是已经用 unaryPred 划分过的。返回满足 unaryPred 的范围的尾后迭代器。如果返回的迭代器不是 end，则它指向的元素及其后的元素必须都不满足 unaryPred  
stable\_partition(beg, end, unaryPred); // 使用 unaryPred 划分输入序列。满足 unaryPred 的元素放置在序列开始，不满足的元素放在序列尾部。返回迭代器，指向最后一个满足 unaryPred 的元素之后的位置如果所有元素都不满足 unaryPred，则返回 beg  
partition(beg, end, unaryPred); // 使用 unaryPred 划分输入序列。满足 unaryPred 的元素放置在序列开始，不满足的元素放在序列尾部。返回迭代器，指向最后一个满足 unaryPred 的元素之后的位置如果所有元素都不满足 unaryPred，则返回 beg  
// 排序算法，要求随机访问迭代器（random-access iterator）  
sort(beg, end); // 排序整个范围  
stable\_sort(beg, end); // 排序整个范围（稳定排序）  
sort(beg, end, comp); // 排序整个范围  
stable\_sort(beg, end, comp); // 排序整个范围（稳定排序）  
is\_sorted(beg, end); // 返回一个 bool 值，指出整个输入序列是否有序  
is\_sorted(beg, end, comp); // 返回一个 bool 值，指出整个输入序列是否有序  
is\_sorted\_until(beg, end); // 在输入序列中査找最长初始有序子序列，并返回子序列的尾后迭代器  
is\_sorted\_until(beg, end, comp); // 在输入序列中査找最长初始有序子序列，并返回子序列的尾后迭代器  
partial\_sort(beg, mid, end); // 排序 mid-beg 个元素。即，如果 mid-beg 等于 42，则此函数将值最小的 42 个元素有序放在序列前 42 个位置  
partial\_sort(beg, mid, end, comp); // 排序 mid-beg 个元素。即，如果 mid-beg 等于 42，则此函数将值最小的 42 个元素有序放在序列前 42 个位置  
partial\_sort\_copy(beg, end, destBeg, destEnd); // 排序输入范围中的元素，并将足够多的已排序元素放到 destBeg 和 destEnd 所指示的序列中  
partial\_sort\_copy(beg, end, destBeg, destEnd, comp); // 排序输入范围中的元素，并将足够多的已排序元素放到 destBeg 和 destEnd 所指示的序列中  
nth\_element(beg, nth, end); // nth 是一个迭代器，指向输入序列中第 n 大的元素。nth 之前的元素都小于等于它，而之后的元素都大于等于它  
nth\_element(beg, nth, end, comp); // nth 是一个迭代器，指向输入序列中第 n 大的元素。nth 之前的元素都小于等于它，而之后的元素都大于等于它  
// 使用前向迭代器的重排算法。普通版本在输入序列自身内部重拍元素，\_copy 版本完成重拍后写入到指定目的序列中，而不改变输入序列  
remove(beg, end, val); // 通过用保留的元素覆盖要删除的元素实现删除 ==val 的元素，返回一个指向最后一个删除元素的尾后位置的迭代器  
remove\_if(beg, end, unaryPred);//通过用保留的元素覆盖要删除的元素实现删除满足 unaryPred 的元素，返回一个指向最后一个删除元素的尾后位置的迭代器  
remove\_copy(beg, end, dest, val); // 通过用保留的元素覆盖要删除的元素实现删除 ==val 的元素，返回一个指向最后一个删除元素的尾后位置的迭代器  
remove\_copy\_if(beg, end, dest, unaryPred); // 通过用保留的元素覆盖要删除的元素实现删除满足 unaryPred 的元素，返回一个指向最后一个删除元素的尾后位置的迭代器  
unique(beg, end); // 通过对覆盖相邻的重复元素（用 == 确定是否相同）实现重排序列。返回迭代器，指向不重复元素的尾后位置  
unique (beg, end, binaryPred); // 通过对覆盖相邻的重复元素（用 binaryPred 确定是否相同）实现重排序列。返回迭代器，指向不重复元素的尾后位置  
unique\_copy(beg, end, dest); // 通过对覆盖相邻的重复元素（用 == 确定是否相同）实现重排序列。返回迭代器，指向不重复元素的尾后位置  
unique\_copy\_if(beg, end, dest, binaryPred); // 通过对覆盖相邻的重复元素（用 binaryPred 确定是否相同）实现重排序列。返回迭代器，指向不重复元素的尾后位置  
rotate(beg, mid, end); // 围绕 mid 指向的元素进行元素转动。元素 mid 成为为首元素，随后是 mid+1 到到 end 之前的元素，再接着是 beg 到 mid 之前的元素。返回迭代器，指向原来在 beg 位置的元素  
rotate\_copy(beg, mid, end, dest); // 围绕 mid 指向的元素进行元素转动。元素 mid 成为为首元素，随后是 mid+1 到到 end 之前的元素，再接着是 beg 到 mid 之前的元素。返回迭代器，指向原来在 beg 位置的元素

// 使用双向迭代器的重排算法  
reverse(beg, end); // 翻转序列中的元素，返回 void  
reverse\_copy(beg, end, dest);; // 翻转序列中的元素，返回迭代器，指向拷贝到目的序列的元素的尾后位置

// 使用随机访问迭代器的重排算法  
random\_shuffle(beg, end); // 混洗输入序列中的元素，返回 void  
random\_shuffle(beg, end, rand); // 混洗输入序列中的元素，rand 接受一个正整数的随机对象，返回 void  
shuffle(beg, end, Uniform\_rand); // 混洗输入序列中的元素，Uniform\_rand 必须满足均匀分布随机数生成器的要求，返回 void

// 最小值和最大值，使用 < 运算符或给定的比较操作 comp 进行比较  
min(val1, va12); // 返回 val1 和 val2 中的最小值，两个实参的类型必须完全一致。参数和返回类型都是 const的引引用，意味着对象不会被拷贝。下略  
min(val1, val2, comp);  
min(init\_list);  
min(init\_list, comp);  
max(val1, val2);  
max(val1, val2, comp);  
max(init\_list);  
max(init\_list, comp);  
minmax(val1, val2); // 返回一个 pair，其 first 成员为提供的值中的较小者，second 成员为较大者。下略  
minmax(vall, val2, comp);  
minmax(init\_list);  
minmax(init\_list, comp);  
min\_element(beg, end); // 返回指向输入序列中最小元素的迭代器  
min\_element(beg, end, comp); // 返回指向输入序列中最小元素的迭代器  
max\_element(beg, end); // 返回指向输入序列中最大元素的迭代器  
max\_element(beg, end, comp); // 返回指向输入序列中最大元素的迭代器  
minmax\_element(beg, end); // 返回一个 pair，其中 first 成员为最小元素，second 成员为最大元素  
minmax\_element(beg, end, comp); // 返回一个 pair，其中 first 成员为最小元素，second 成员为最大元素

// 字典序比较，根据第一对不相等的元素的相对大小来返回结果。如果第一个序列在字典序中小于第二个序列，则返回 true。否则，返回 fa1se。如果个序列比另一个短，且所有元素都与较长序列的对应元素相等，则较短序列在字典序中更小。如果序列长度相等，且对应元素都相等，则在字典序中任何一个都不大于另外一个。  
lexicographical\_compare(beg1, end1, beg2, end2);  
lexicographical\_compare(beg1, end1, beg2, end2, comp);