

Standard Template Library

项润冶

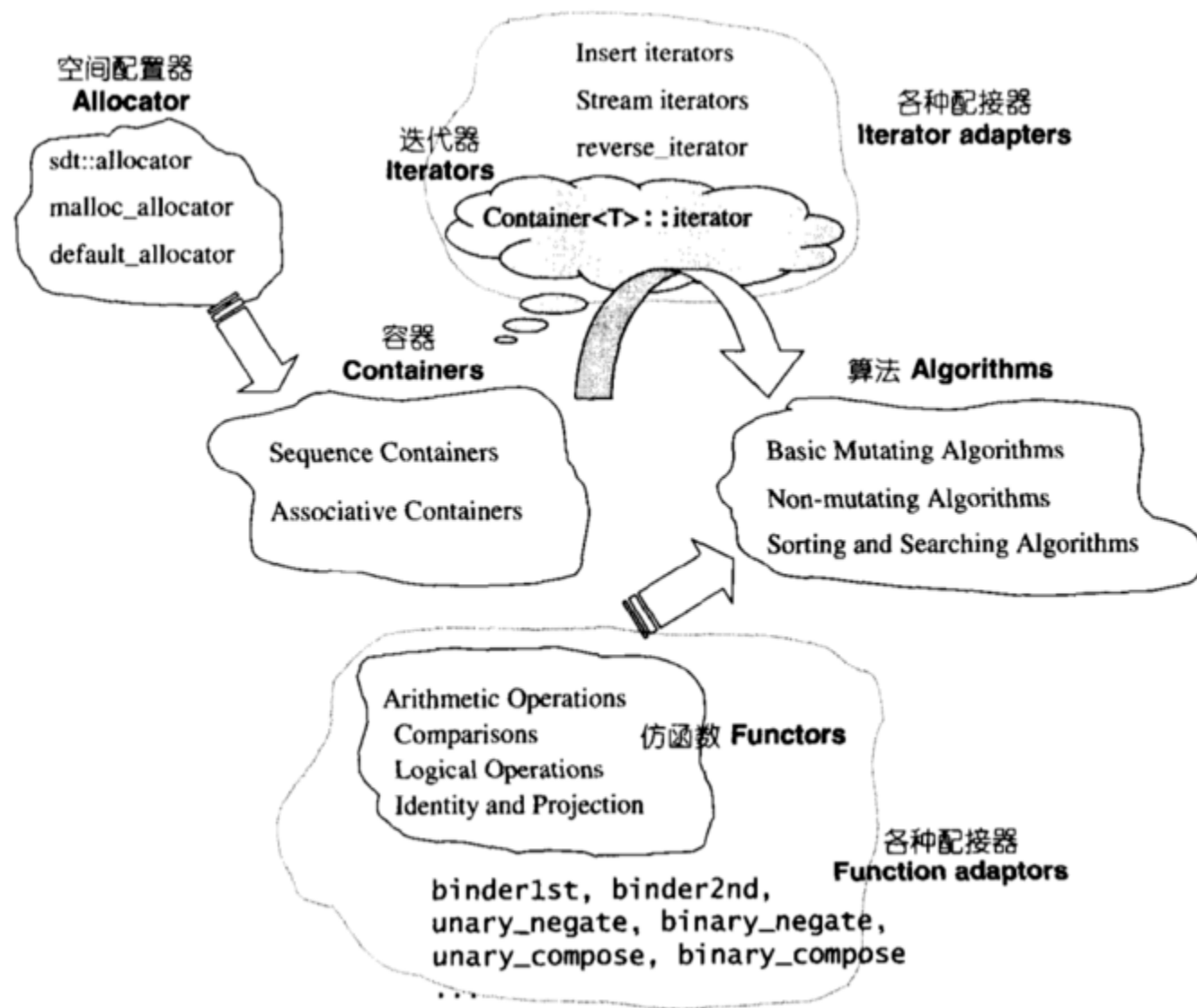
2018-05-23

介绍

- 标准模板库，C++标准中一个重要的组成部分。
 - 只规定接口形式，不同编译器实现细节的不同。
 - 做题有巨大的帮助。
-
- 体系复杂，成员繁多，需要抓住重点。
 - 一般讲解关注语法，而忽视内部实现。
 - 完美的数据结构设计、工程入门典范。
 - C++ 准标准库：Boost

STL 组成

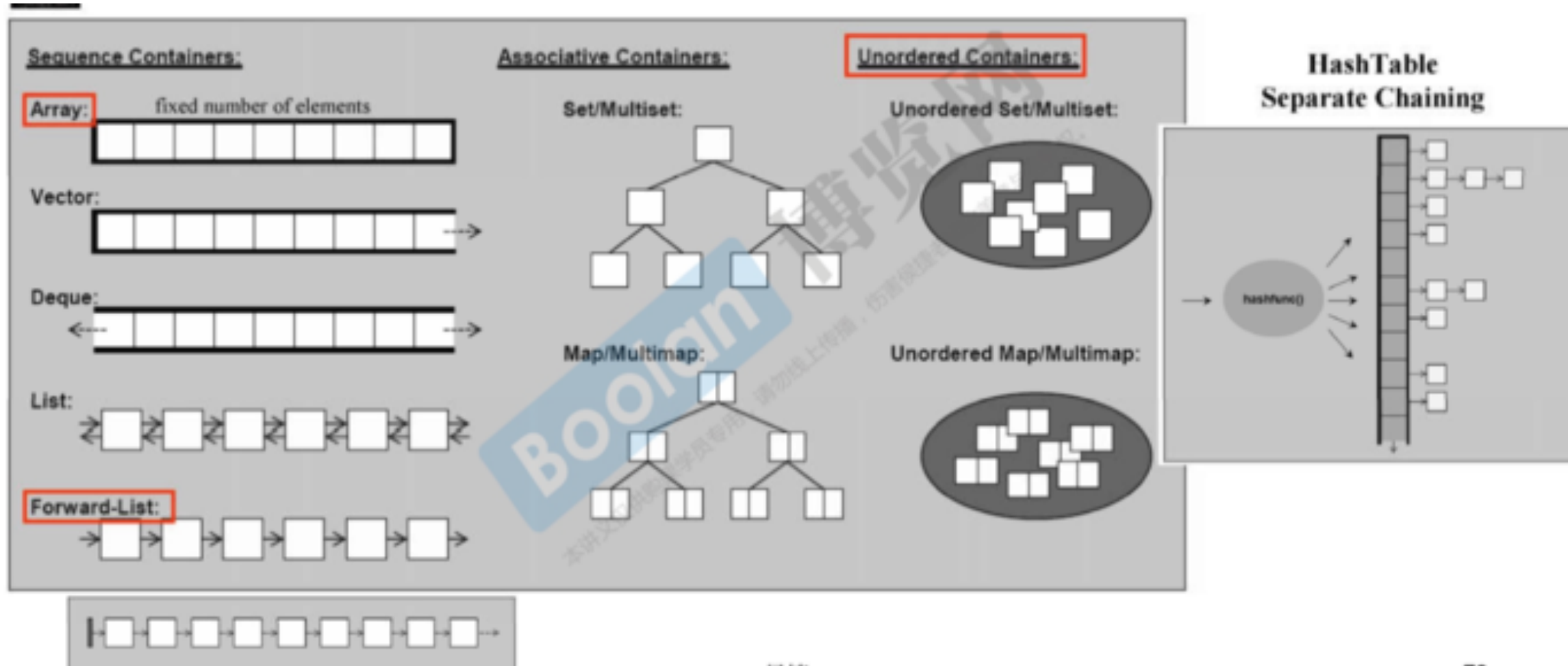
- 空间配置器(allocator)
 - 迭代器(iterators)
 - 容器(containers)
 - 适配器(adapters)
 - 算法(algorithms)
 - 仿函数(functors)
-
- using namespace std;
 - std::map



空间配置器(allocator)

- 与使用STL基本上没有太大关系
- STL中实现容器功能时不可缺少的一部分，与内存空间分配有关
- 建议感兴趣的去了解一下。

容器(containers)



Sequence containers

- **Vector**
- 不定长数组
- 头文件 : `<vector>`
- 定义 : `vector<int>v;`
- `vector<pair<int,int>>v;`
- `vector<vector<int> >v;`
- 方法 : `v.push_back()` `v.pop_back()`

0	1	2	3	4	5	more...
---	---	---	---	---	---	---------

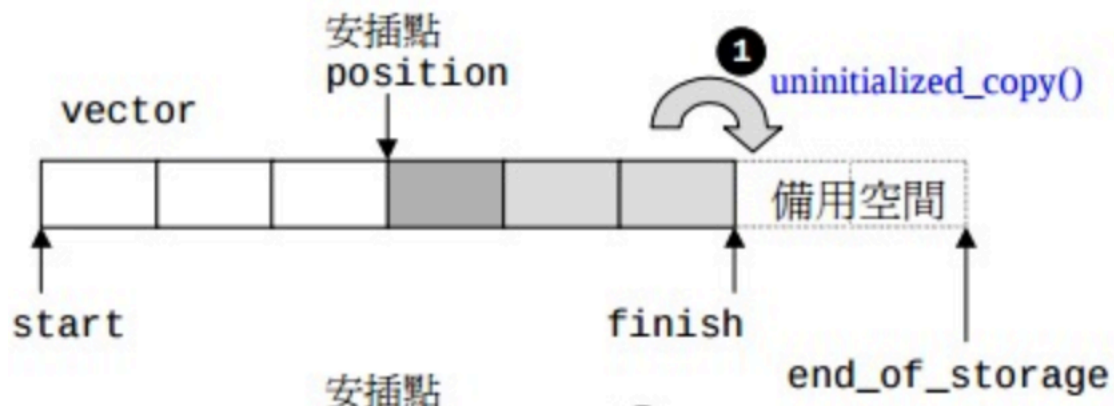
a	b	c	d	e	f	
h	e	l	l	o	o	more...
w	o	r	l	d		

more...

Memory Limit

Vector

- 方法：v.clear() v.size() v.capacity()
- 获取迭代器：v.begin() v.cbegin() v.end() v.cend() v.rbegin() v.rend()



Vector iterator

public:

//一大堆反正都是取容器的首尾迭代器

//反向迭代器可能在某些算法里面有特殊的作用, 比如`sort(v.rbegin(), v.rend())`, 可以实现对v进行降序排列

//在`stl_iterator`中实现方法主要将迭代器`++`动作互换

```
iterator begin() { return start; }
```

//使用二级分配器

```
typedef simple_alloc<value_type, Alloc> data_allocator;
```

```
iterator start; //申请的内存起点
```

```
iterator finish; //实际使用的内存终点
```

```
iterator end_of_storage; //申请的内存终点
```

```
void insert_aux(iterator position, const T& x);
```

//申请大小为n的内存, 并用value来填充

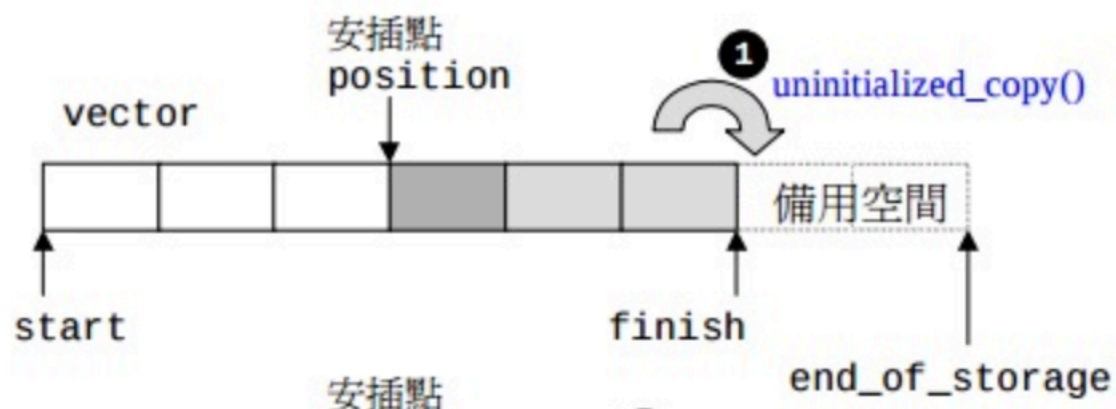
```
void fill_initialize(size_type n, const T& value) {
```

```
    start = allocate_and_fill(n, value);
```

```
    finish = start + n;
```

```
    end_of_storage = finish;
```

```
}
```



迭代器(iterators)

- 类似指针的一种对象，多种类型
- 提供对容器的一种统一访问方式(++/--)
- 算法摆脱对于容器的依赖
- sum(begin,end)

```
//对于int类的求和函数
int sum(int *a , int n)
{
    int sum = 0 ;
    for (int i = 0 ; i < n ; i++) {
        sum += *a++;
    }
    return sum;
}
```

```
10 //对于ListNode类的求和函数
11 struct ListNode {
12     int val;
13     ListNode * next;
14 };
15 int sum(ListNode * head) {
16     int sum = 0;
17     ListNode *p = head;
18     while (p != NULL) {
19         sum += p->val;
20         p=p->next;
21     }
22     return sum;
23 }
```

Vector

- 遍历
- `for(int i=0;i<v.size();i++) v[i]=i;`
- `for(vector<int>::iterator it = v.begin(); it!= v.end(); it++) *it=0;`
- `for(auto it=v.begin();it!=v.end();it++) *it=0;`
- 区间遍历: `for(auto i :v) cout<<i;`

Simple containers

- pair
- 将两个数据类型合成为一个数据类型，类似结构体
- <utility>
- pair<T1, T2> myPair
- make_pair(T1 t, T2 u)
- .first和.second

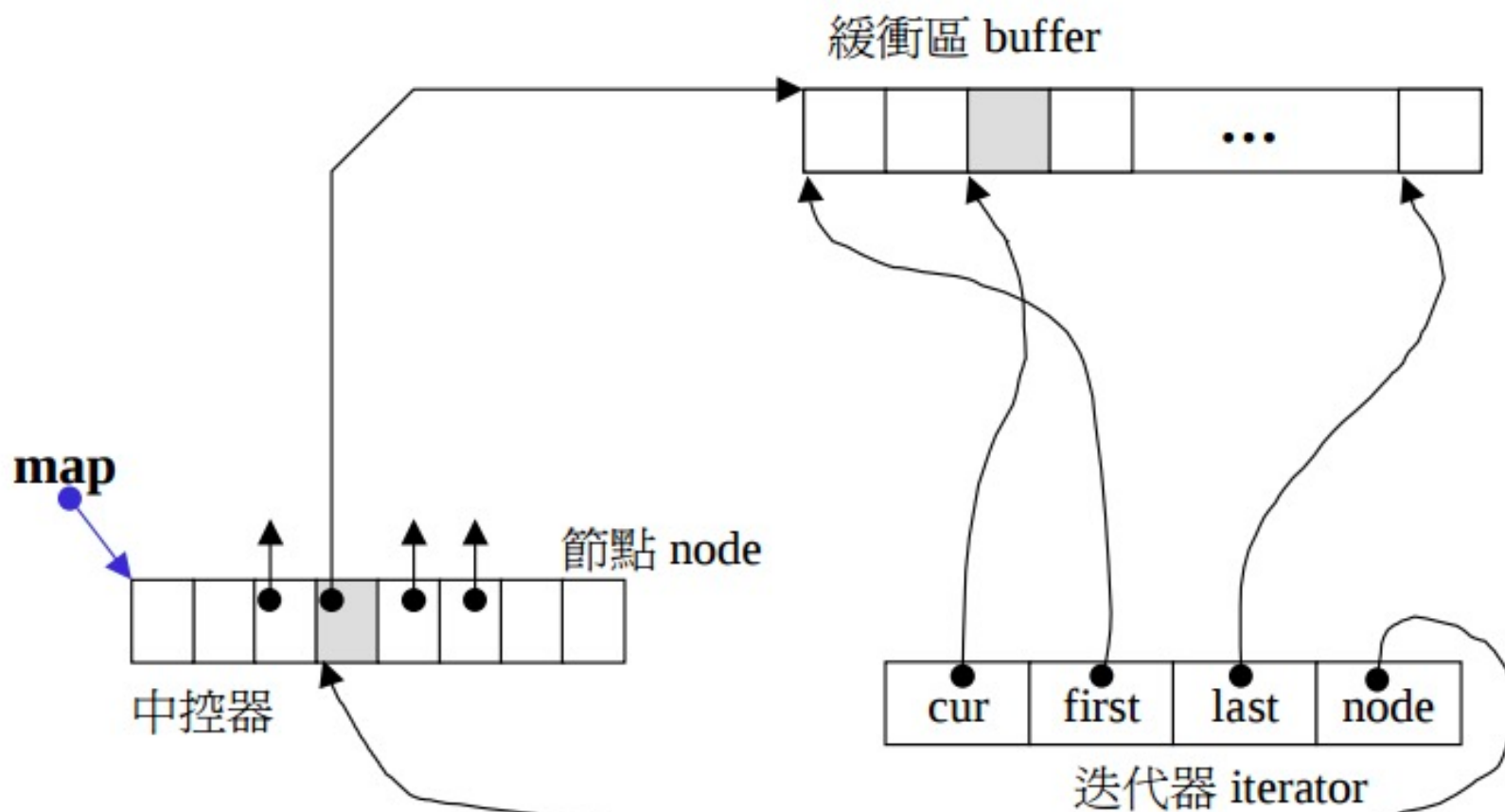
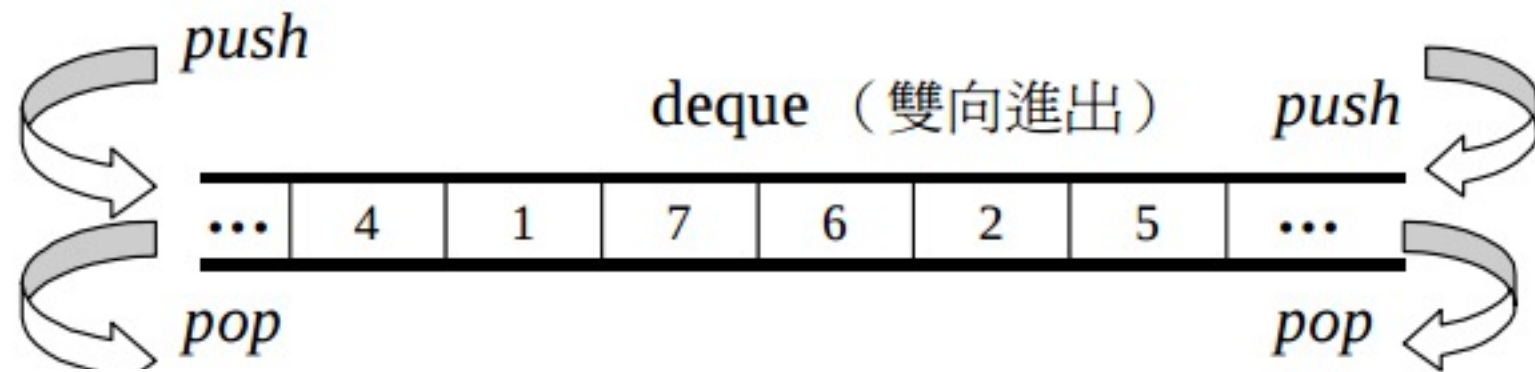
Array

- `array<T, N>a`

0	1	2	3	4	5
---	---	---	---	---	---

a	b	c	d	e	f
h	e	l	l	o	o
w	o	r	l	d	

deque

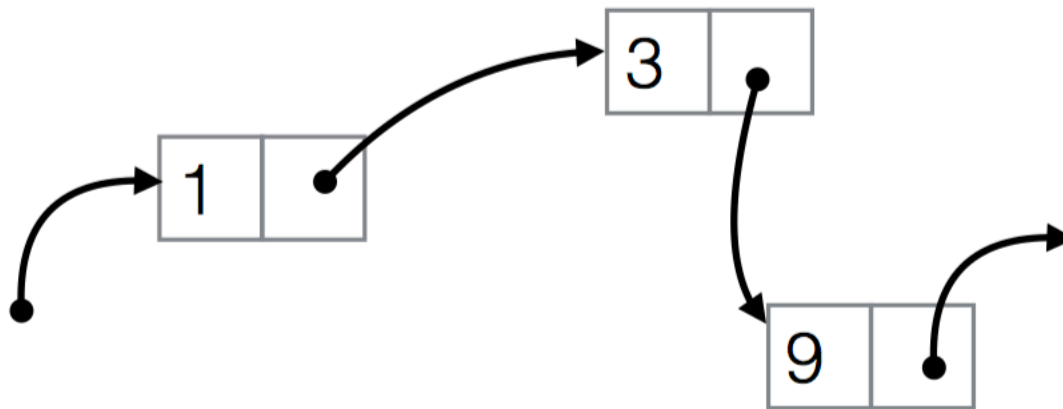


deque

- `deque<T>d`
- `push_back()` `push_front()` `pop_back()` `pop_front()`

list

- 双向链表
 - push_back()
 - push_front()
 - pop_back()
 - pop_front()
 - insert()
 - erase()
-
- 单向链表 forward_list



list

```
#include <algorithm>
#include <iostream>
#include <list>

int main()
{
    // Create a list containing integers
    std::list<int> l = { 7, 5, 16, 8 };

    // Add an integer to the front of the list
    l.push_front(25);
    // Add an integer to the back of the list
    l.push_back(13);

    // Insert an integer before 16 by searching
    auto it = std::find(l.begin(), l.end(), 16);
    if (it != l.end()) {
        l.insert(it, 42);
    }

    // Iterate and print values of the list
    for (int n : l) {
        std::cout << n << '\n';
    }
}
```

Output:

```
25
7
5
42
16
8
13
```


容器适配器

- **stack** `stack<int, vector<T>>s; s.push() s.pop() s.top()`
- **queue** `queue<int>q; q.push() q.pop() q.front() q.back()`
- **priority_queue**
- `priority_queue<T, vector<T>, greator<T> >`
- 指定数据类型， 指定容器， 指定函数对象
- 例题：hdu 5437 Alisha's Party fzu 1182 Argus
- 参考 [【c++】STL里的priority_queue用法总结 - CSDN博客](#)

Associative containers

Red Black Tree(self-balancing binary search tree)

- map/multimap
- set/multiset

map/multimap

- `map<int, string> myMap`
- `.clear() .find() .count()`
- `myMap[1] = "hello world"`
- `myMap[key]=value`
- `pair<Const T1,T> myMap.insert(make_pair(key,value))`
- `for(auto it=myMap.begin();it!=myMap.end();it++)`
- `multimap` 允许key重复
- `无[]`

key	value
1	3
3	3
4	4
5	2
999	-1000

set/multiset

- `set<int> mySet`
- `mySet.clear()`
- `mySet.insert(3)`
- `mySet.find(3) != mySet.end()`

key
1
3
4
5
999

unordered

Hash table

- unordered_map/unordered_multimap
- unordered_set/unordered_multiset
- 不需要内部内容有序时使用，用方法同上
- 插入 $O(1)$
- 内存开销大

bitset

- [ACM位运算&bitset总结 - CSDN博客](<https://blog.csdn.net/chaiwenjun000/article/details/71154235>)
- 题目链接：<http://acm.hdu.edu.cn/showproblem.php?pid=5745>

算法(algorithms)

std::sort

Defined in header `<algorithm>`

<code>template< class RandomIt ></code> <code>void sort(RandomIt first, RandomIt last);</code>	(1)
<code>template< class ExecutionPolicy, class RandomIt ></code> <code>void sort(ExecutionPolicy&& policy, RandomIt first, RandomIt last);</code>	(2) (since C++17)
<code>template< class RandomIt, class Compare ></code> <code>void sort(RandomIt first, RandomIt last, Compare comp);</code>	(3)
<code>template< class ExecutionPolicy, class RandomIt, class Compare ></code> <code>void sort(ExecutionPolicy&& policy, RandomIt first, RandomIt last, Compare comp);</code>	(4) (since C++17)

first, last - the range of elements to sort

policy - the execution policy to use. See [execution policy](#) for details.

comp - comparison function object (i.e. an object that satisfies the requirements of [Compare](#)) which returns `true` if the first argument is *less* than (i.e. is ordered *before*) the second.

The signature of the comparison function should be equivalent to the following:

```
bool cmp(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function object must not modify the objects passed to it.

The types `Type1` and `Type2` must be such that an object of type `RandomIt` can be dereferenced and then implicitly converted to both of them.

sort(first,last,comp)

- sort(v.begin(), v.end(), greater<int>()) sort(v.rbegin(), v.rend())
- [first,last) 需要排序的区间， 地址或者是随机访问迭代器
- comp 函数对象， 用于比较
 1. 函数：bool cmp(const Type1 &a, const Type2 &b)
 2. 仿函数([function.h](#))：greater<T>() less<T>() less_equal<T>() greater_equal<T>()

```
93  template <class T>
94  struct greater : public binary_function<T, T, bool> {
95      bool operator()(const T& x, const T& y) const { return x > y; }
```


sort

- `stable_sort(first, last, comp)` : `sort` 的稳定版本
- `is_sorted(first, last, comp)` : 判断序列是否有序
- `is_sorted_until(first, last, comp)` : 返回第一个无须的迭代器
- `partial_sort(first, mid, last)` : 部分排序
- `nth_element(first, nth, last, comp)` : 确定第 `n` 大的元素

permutation

- `next_permutation(first, last, comp)` : 全排列的下一个
- `prev_permutation(first, last, comp)` : 全排列的上一个
- `is_permutation(first1, last1, last2)` : 判断是否排列中一种

search

- `lower_bound(first, last, value, comp)` : 下界
- `upper_bound(first, last, value, comp)` : 上界
- {12,15,17,19,20,22,22,23,26,29,35,40,51}
- 20: 4 5 22:5 7
- `binary_search(first, last, value, comp)` : 二分查找 T/F
- `equal_range(first, last, value, comp)` : 第一个可以插入的位置, 以及最后一个可以插入的位置
- `unique(,)` : 删除相邻重复的元素
- 拓展 : `sort(a,a+n) unique(a,a+n)`

文档浏览器

- [Dash](#) : Mac
- [Zeal](#): Win/Linux

The screenshot displays the Zeal documentation browser interface. On the left, a sidebar shows a search bar with 'sort' entered and a list of search results including 'sort', 'sort_stats', 'sortTestMethodsUsing', 'sort_heap', 'sorted', 'Sorting HOW TO', 'lastResort', 'partial_sort', 'qsort', 'stable_sort', 'insert', 'PyList_Sort', and 'qsort'. Below the search results, a 'See also' section lists 'partial_sort' and 'stable_sort'. The main panel shows the documentation for 'std::sort'. It includes the C++ 'Algorithm library' tab, the function signature 'std::sort', and its definition in the <algorithm> header. The definition shows four overloads: (1) a basic sort, (2) a sort with an execution policy, (3) a sort with a custom comparison function, and (4) a sort with an execution policy and a custom comparison function. Below the code, a paragraph explains that 'std::sort' sorts elements in ascending order and that the order of equal elements is not guaranteed. It then lists the parameters: 'first' and 'last' for the range, 'policy' for the execution policy, and 'comp' for the comparison function. The 'Parameters' section further details the requirements for 'comp'. Finally, the 'Type requirements' section lists the requirements for 'RandomIt'.

std::sort

Defined in header `<algorithm>`

```
template< class RandomIt >
void sort( RandomIt first, RandomIt last ); (1)

template< class ExecutionPolicy, class RandomIt >
void sort( ExecutionPolicy&& policy, RandomIt first, RandomIt last ); (2) (since C++11)

template< class RandomIt, class Compare >
void sort( RandomIt first, RandomIt last, Compare comp ); (3)

template< class ExecutionPolicy, class RandomIt, class Compare >
void sort( ExecutionPolicy&& policy, RandomIt first, RandomIt last, Compare comp ); (4) (since C++11)
```

Sorts the elements in the range `[first, last)` in ascending order. The order of equal elements is not guaranteed to be preserved.

- 1) Elements are compared using operator `<`.
- 3) Elements are compared using the given binary comparison function `comp`.
- 2,4) Same as (1,3), but executed according to `policy`. These overloads do not participate in overload resolution unless `std::is_execution_policy_v<std::decay_t<ExecutionPolicy>>` is true

Parameters

first, last - the range of elements to sort

policy - the execution policy to use. See [execution policy](#) for details.

comp - comparison function object (i.e. an object that satisfies the requirements of [Compare](#)) which returns `true` if the first argument is *less* than (i.e. is ordered *before*) the second.

The signature of the comparison function should be equivalent to the following:

```
bool cmp(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function object must not modify the objects passed to it.

The types `Type1` and `Type2` must be such that an object of type `RandomIt` can be dereferenced and then implicitly converted to both of them.

Type requirements

- `RandomIt` must meet the requirements of [ValueSwappable](#) and [RandomAccessIterator](#).
- The type of dereferenced `RandomIt` must meet the requirements of [MoveAssignable](#) and [MoveConstructible](#).

推荐

使用一个东西，
却不明白他的道理，
不高明！

侯捷《STL源码剖析》

