>

登龙（DLonng）

选择大于努力

STL 常用容器

STL 简介

STL 是 Standard Template Library 的简称，中文名**标准模板库**，是用 C++ 模板机制来表示泛型的库。STL 现在是 C++ 标准库的一部分（大约 80 %），在工作中也非常常用，非常值得我们学习，其实 STL 就是下面 6 个组件的集合：

- 容器 Container
- 算法 Algorithm
- 迭代器 Iterator
- 仿函数 Function object
- 适配器 Adaptor
- 空间配置器 Allocator

本次我们来学习常用的 STL 容器，主要包括：

- **序列式容器**：vector，queue，deque，priority_queue，list，stack
- **关联式容器**：set，map，multiset，multimap

什么是容器?

普通概念上容器就是用来存放一组元素，在 STL 中也是如此：**STL 中的容器用来管理一组元素。**因为存储的元素的属性不同，使用的场合也不同，于是就开发了多种容器，比如 vector 其实就是一个动态数组，deque 就是一个队列，list 是一个链表，set 是元素的集合，map 是键值对。

可以看出容器的设计跟数据结构有很大的关系，因为我们处理的都是数据，在不同的场合选择最佳的数据结构（STL 容器）可以更方便的解决问题，这其实就是学习数据结构的意义，实际工作中是不需要你自己封装数据结构的，基本都是使用稳定的库（例如 STL），但是我们必须知道在何种场合使用何种数据结构（STL 容器）。

知道并不等于会用，只有真正动手用一个数据结构解决一个问题才能说明你掌握了它。

序列式容器 - Vector

来看看最简单，最常用的序列式容器 - vector，它实际上是一个**动态数组**，它将元素置于一个动态数组中管理，它的优缺点和数组相同：

- 优点：随机访问元素快，直接用索引访问
- 缺点：在中部和头部添加和删除元素比较费时，需要移动大量元素

下面是 vector 的基本用法，这里只介绍常用的 API，全部的 STL 容器相关的用法可以参考 cppreference 网站。

这个例子中我们创建了一个包含了 4 个整型元素的 vector，你也可以换成 double 或其他类型，最常用的函数是 `push_back` 添加元素，其他的 API(size, pop_back) 参考前面的网站（我不可能把所有的函数都列出来，要是这样的话自己看手册不就行了，方法很重要！要学会举一反三）：

```
#include <iostream>

// 需要包含 vector 头文件
#include <vector>

int main(void) {
  std::vector<int> v = { 1, 2, 3, 4 };
  // 在尾部添加元素
  v.push_back(5);
  v.push_back(6);
  for (auto n : v)
    std::cout << n << std::endl;
  return 0;
}
```

既然谈到动态数组，我们又想到静态数组，你可能会想到使用 `int a[10];` 来定义一个静态数组，但是在 C++ 中有更好的 `array` 可以使用：

```cpp
// 使用 array 必须包含该头文件
#include <array>

// 创建一个含有 5 个元素的静态数组
std::array<int, 5> a = { 1, 2, 3, 4, 5 };
```

array 和 vector 的不同点如下：

- 静态数组 array 是在栈上分配内存，容量较小，并且不需要变长
- 动态数组 vector 是在堆上分配的内存，容量较大，需要变长

在合适的场合要选择合适的数据结构来解决问题，没有哪个好哪个不好，只有合适和不合适。

序列式容器 - queue，deque，priority_queue

queue 是先进先出的队列，而 deque 是一个双端队列（队列的两端都可以操作），关于队列概念的介绍不是这里的重点（相信你学习 STL 应该了解些常用的数据结构）。

普通队列 queue

queue 与数据结构中队列的定义相同，队列中的元素先进先出，常用的 API 有 2 个：

```cpp
#include <iostream>
// 队列头文件
#include <queue>

int main(void) {
  // 创建一个队列
  std::queue<int> q;
  // 在尾部添加元素
  q.push(1);
  q.push(2);
  q.push(3);
  // 弹出元素
  while (!q.empty()) {
    std::cout << q.front() << std::endl;
    q.pop();
  }
  return 0;
}
```

要注意：**queue 没有迭代器**，所以输出元素会有点麻烦。实际使用时，当你的数据需要从一端进入，从另一端输出时可以考虑队列容器。

双端队列 deque

双端队列可以说是队列的升级版本，允许在头部和尾部分别进行添加和删除操作，常用的 API 如下：

- push_back：在队列尾部添加一个元素
- pop_back：删除队列尾部的元素
- push_front：在队列头部插入一个元素
- pop_front：删除队列头部的元素

简单的使用方法如下：

```
// 必须包含这个头文件
#include <queue>

// 创建一个存储 int 类型的队列
std::deque<int> d;
d.push_back(1);
d.push_front(2);
// d: 2 1
d.pop_front();
d.pop_back();
// d:
```

当你需要同时操作队列头和尾部的元素时考虑 deque。

优先级队列 - priority_queue

向优先级队列（堆 heap）中添加元素，默认会进行从小到大的排序，我们也可以手动指定排序规则：

```
#include <queue>

// 创建一个优先级队列
std::priority_queue<int> q;

// q: 0 1 2 3 4 5 6 7 8 9
for (int n : {1, 4, 3, 2, 6, 5, 8, 7, 9, 0})
  q.push(n);

std::prority_queue<int, std::vector<int>, std::greater<int>> q2;
// q2: 9 8 7 6 5 4 3 2 1 0
for (int n : {1, 4, 3, 2, 6, 5, 8, 7, 9, 0})
  q2.push(n);
```

`q.pop()` 操作每次删除队列中最顶部（可能最大，可能最小，与排序规则有关）的元素。

序列式容器 - list

list 又可以称为链表，其实跟数据结构中的链表的思想相同，也具有链表的优缺点：

- 优点：插入删除速度快
- 缺点：访问元素速度慢，需要依次遍历

简单的使用方法如下：

```
// list 头文件
#include <list>

// 创建一个包含 int 类型的 list
std::list<int> list = { 1, 2, 3, 4 };

// 尾部添加元素
list.push_back(5);

// 头部添加元素
list.push_front(0);

// 在元素 3 前插入元素 33
auto it = std::find(list.begin(), list.end(), 3);
if (it != list.end)
  list.insert(it, 33);

// 错误：list 不可以随机访问，没有 [] 操作，要用迭代器进行遍历
int item = list[1];

// 删除一个元素后，指向该元素的迭代器会失效
list.erase(list.begin());
```

当你的数据需要频繁的增加和删除时，可以考虑链表容器，它具有 O(1) 的插入和删除的时间复杂度。

序列式容器 - stack

栈平常用的不是很多，但是它的**后进先出**思想却用在很多地方，比如函数调用需要栈的帮助，还是有必要了解下的：

```
#include <stack>

std::stack<int> mystack;

// mystack(top -> end): 3 2 1
mystack.push(1);
mystack.push(2);
mystack.push(3);
// pop: 3
mystack.pop();
// pop: 2
mystack.pop();
// pop: 1
mystack.pop();
```

我们入栈的顺序是 1 2 3，出栈的顺序是 3 2 1，符合后进先出的原则。栈的实现比较简单，可以使用 C 语言的数组来实现，其实就是设置栈顶指针来限制数组中元素的个数，并封装 push 和 pop 的逻辑。

关联式容器 - set

std::set 是一个包含有序的唯一对象（key）的容器，意思是 set 容器不包含重复的元素，并且里面的元素是有序的，排序方式可以通过比较器 `Compare` 来指定，set 底层的实现方式是红黑树，有兴趣可以手动实现 set，下面来看看基本的用法。

```
#include <iostream>
#include <set>
#include <cstring>

struct strless {
  bool operator()(const char *str1, const char *str2) {
    return strcmp(str1, str2)  <  0;
  }
};


int main() {
  const char *str[] = { "a", "b", "c", "d","e","f" };
  // 可以自定义 set 的排序函数
  std::set<const char *, strless> myset(str, str + 6, strless());
  // insert 的返回值是 pair 类型，如果元素已经存在，则插入失败
  std::pair<std::set<const char *>::iterator, bool> res = myset.insert("g
  if (res.second)
```

```
    std::cout << "insert " << *(res.first) << " ok" << std::endl;

  for (auto ib = myset.begin(); ib != myset.end(); ib++)
    std::cout << *ib << " ";
  std::cout << std::endl;
  return 0;
}
```

其中 insert 函数需要注意，它的返回值是 `std::pair<std::set<const char*>::iterator, bool>` 类型，第一个参数 first 是指向插入元素的迭代器，第二个元素 second 表示该元素是否插入成功，成功返回 1，失败返回 0，所以可以用 res.second 来判断元素是否插入成功。关于 set 的其他操作，例如删除，清空，查找等都与其他容器差不多，可以自己写 demo 测试下。

另外，STL 也提供了 `std::unordered_set` 来存储默认不排序（根据 key 的 hash 值来确定 key 的位置）的元素，用法与 set 基本相同。

关联式容器 - multiset

multiset 与 set 的唯一区别是：**multiset 允许出现多个相同的元素（key）**。在底层实现中，set 的每个节点就是单一的节点，而 multiset 的每个节点是一个链表，所以一个键可以对应多个不同的值。

```
#include <iostream>
#include <set>
#include <string>
#include <cstring>

struct student {
        int score;
        char name[30];
};

// 根据学生分数来排序
struct stuless {
        bool operator()(const student &s1, const student &s2) {
                return s1.score < s2.score;
        }
};

int main() {
        student sarray[3] = { { 81, "Jack" }, { 90, "Mary" }, { 95, "Tom"
        std::multiset<student, stuless> myset(sarray, sarray + 3, stuless
```

```
        student stu;
        // 分数为 89 对应了 2 个名字相同的学生
        stu.score = 89;
        strcpy(stu.name, "Joa");
        myset.insert(stu);

        strcpy(stu.name, "Joa");
        myset.insert(stu);

        for (auto ib = myset.begin(); ib != myset.end(); ib++)
                std::cout << (*ib).score << " " << (*ib).name << std::endl
        return 0;
}
```

一定要注意 multiset 允许存在相同的 key 元素！

关联式容器 - map

map 是一个包含唯一键值对的关联容器，key 通过比较器来排序（默认按照 key 从小到大排序），map 中不能出现相同的键值对。通常用 std::pair 来遍历 map，其中 pair.first 代表键，pair.second 代表值，例如下面这个例子遍历一个 map：

```
#include <map>

std::map<int, char> counts { {5, 'a'}, {3, 'b'}, {4, 'c'} };
// 从小到大排序输出
for (const auto &pair : counts)
  std::cout << pair.first << ": " << pair.second << std::endl;
// 3: b
// 4: c
// 5: a
```

另外在使用 map 时，要注意 `[]` 的一个使用技巧，来看一个例子：

```
#include <map>

// 单词 -> 出现次数的 map
std::map<std::string, size_t> words_map;

// 统计每个单词出现的次数
for (const auto &w : { "this", "is", "not", "a", "this", "is", "a"})
```

```
    ++words_map[w];

  for (const auto &pair : words_map)
    std::cout << pair.first << ": " << pair.second << "times." << std::endl
```

当一个单词不在 words_map 中时， `++words_map[w]` 就会将该单词插入，并将次数设置为 1，如果单词已经存在则不会再次插入，而仅仅将出现的次数（值）加 1，实现了统计单词出现次数的功能。

同样，STL 也提供了默认不排序（根据 key 的 hash 值来确定 key 的位置）的 `std::unordered_map` ，用法与 map 基本相同。

关联式容器 - multimap

multimap 允许出现多个重复的键值对（包括完全相同的键值对），例如一个分数可能对应多个学生：

```
#include <iostream>
#include <map>

int main() {
        std::multimap<int, const char *> m;
        m.insert(std::pair<int, const char *>(80, "Tom"));
        m.insert(std::pair<int, const char *>(80, "Jack"));
        m.insert(std::pair<int, const char *>(85, "Tark"));
        m.insert(std::pair<int, const char *>(90, "Mary"));

        for (auto ib = m.begin(); ib != m.end(); ib++)
                std::cout << (*ib).first << "    " << (*ib).second << std::
  return 0;
}
```

例子中 Tom 和 Jack 都是 80 分。

结语

STL 的使用非常广泛，本次介绍的都是非常常用的容器，例子中大多都是介绍 API 为主，但是在实际项目中，容器中存储的时候通常都是复杂的数据结构，例如 `std::vector<std::vector>` 这中嵌套的类型。但是不管类型多么复杂，只要理解基本的用法，我们自己也可以设计出复杂的数据结构，一切都在基础之上，勿以浮沙筑高台！

下篇文章介绍 STL 的常用算法，有兴趣可以关注，谢谢你的阅读 :)

本文原创首发于微信公号「登龙」，分享机器学习、算法编程、Python、机器人技术等原创文章，扫码即可关注！

DLonng at 09/17/17

《 》