* 我希望你没有刻意为追求一个数字而生活，我希望你找到了真正的价值所在
* 你一定要坚持做自己，静下心来做自己喜欢的事，然后把自己交给命运

@[toc]

# STL基本概念

* STL(Standard Template Library,**标准模板库**)
* STL 从广义上分为: **容器(container) 算法(algorithm) 迭代器(iterator)**
* **容器**和**算法**之间通过**迭代器**进行无缝连接。
* STL 几乎所有的代码都采用了模板类或者模板函数

## STL六大组件

STL大体分为六大组件，分别是:**容器、算法、迭代器、仿函数、适配器（配接器）、空间配置器**

1. 容器：各种数据结构，如vector、list、deque、set、map等,用来存放数据。
2. 算法：各种常用的算法，如sort、find、copy、for\_each等
3. 迭代器：扮演了容器与算法之间的胶合剂。
4. 仿函数：行为类似函数，可作为算法的某种策略。
5. 适配器：一种用来修饰容器或者仿函数或迭代器接口的东西。
6. 空间配置器：负责空间的配置与管理。

## STL中容器、算法、迭代器

**容器**: 置物之所也

* STL**容器**就是将运用**最广泛的一些数据结构**实现出来
* 常用的数据结构：数组, 链表,树, 栈, 队列, 集合, 映射表 等

这些容器分为**序列式容器**和**关联式容器**两种:

* **序列式容器**:强调值的排序，序列式容器中的每个元素均有固定的位置。
* **关联式容器**:二叉树结构，各元素之间没有严格的物理上的顺序关系

**算法** ; 问题之解法也

* 有限的步骤，解决逻辑或数学上的问题，这一门学科我们叫做算法(Algorithms)

算法分为:**质变算法**和**非质变算法**。

* 质变算法：是指运算过程中会更改区间内的元素的内容。例如拷贝，替换，删除等等
* 非质变算法：是指运算过程中不会更改区间内的元素内容，例如查找、计数、遍历、寻找极值等等

**迭代器** : 容器和算法之间粘合剂

* 提供一种方法，使之能够依序寻访某个容器所含的各个元素，而又无需暴露该容器的内部表示方式。
* 每个容器都有自己专属的迭代器
* 迭代器使用非常类似于指针，初学阶段我们可以先理解迭代器为指针

迭代器种类：

|  |  |  |
| --- | --- | --- |
| 种类 | 功能 | 支持运算 |
| 输入迭代器 | 对数据的只读访问 | 只读，支持++、==、！= |
| 输出迭代器 | 对数据的只写访问 | 只写，支持++ |
| 前向迭代器 | 读写操作，并能向前推进迭代器 | 读写，支持++、==、！= |
| 双向迭代器 | 读写操作，并能向前和向后操作 | 读写，支持++、--， |
| 随机访问迭代器 | 读写操作，可以以跳跃的方式访问任意数据，功能最强的迭代器 | 读写，支持++、--、[n]、-n、<、<=、>、>= |

常用的容器中迭代器种类为双向迭代器，和随机访问迭代器

# 容器算法迭代器初识

## Vector 容器

1. 容器： vector
2. 算法： for\_each
3. 迭代器： vector<int>::iterator

## Vector 三大遍历算法

第一种方法：

vector<int>::iterator Begin = v.begin();  
 vector<int>::iterator End = v.end();  
  
 while (Begin != End) {  
 cout << \*Begin << endl;  
 Begin++;  
 }

第二种方法 :

for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {  
 cout << \*it << endl;  
 }

第三种方法 :

#include <algorithm>  
  
void myPrintln(int val) {  
 cout << val << endl;  
}  
for\_each(v.begin(), v.end(), myPrintln);

## Vector 存放其他数据类型

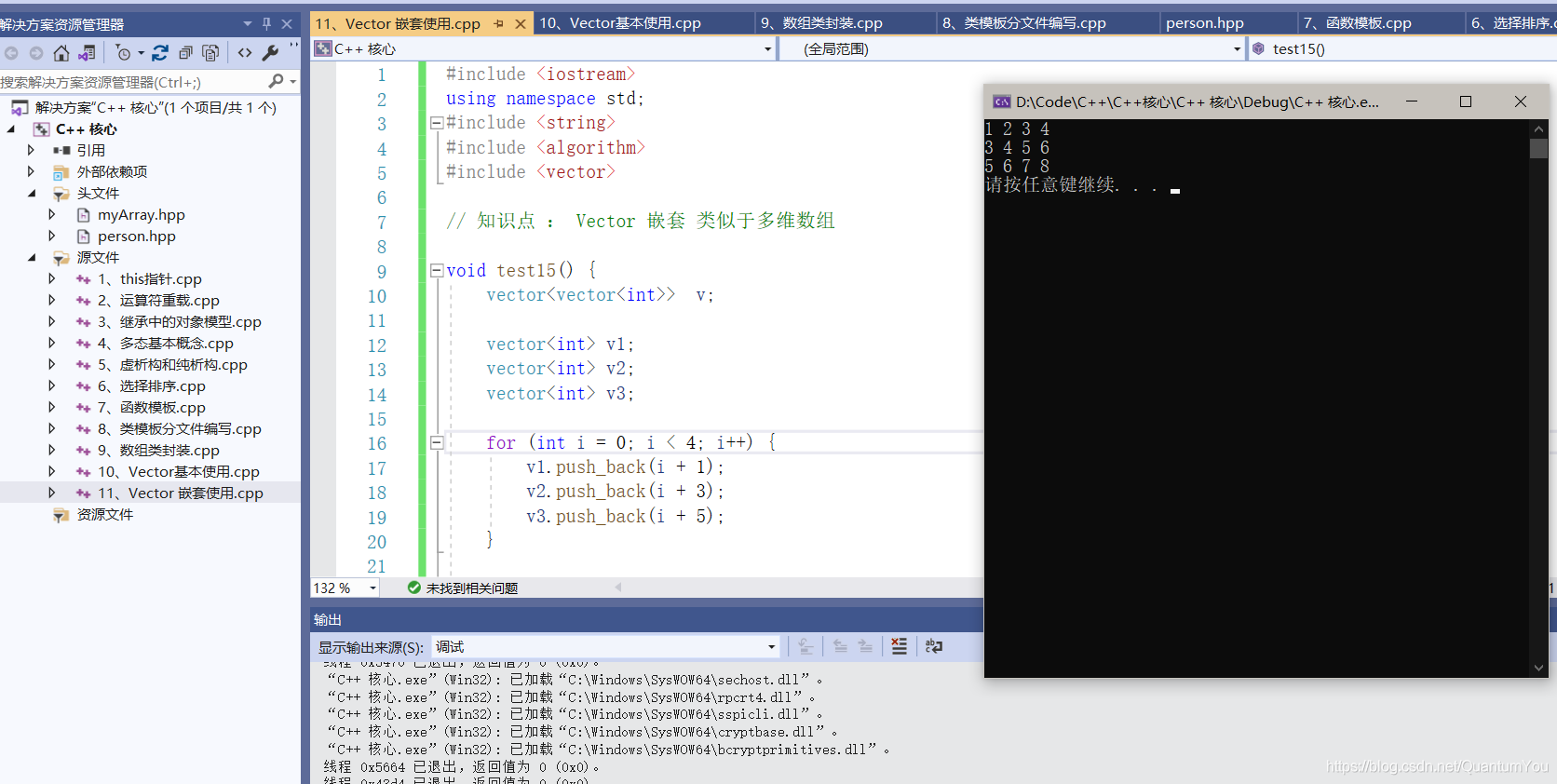
* 存放自定义数据类型 Person

void test13() {  
 vector<Person> v1;  
 Person p1("花丸",16);  
 Person p2("善子", 17);  
 Person p3("露比", 15);  
  
  
 v1.push\_back(p1);  
 v1.push\_back(p2);  
 v1.push\_back(p3);  
  
 for (vector<Person>::iterator it = v1.begin(); it != v1.end(); it++) { // \*it 就是 Vector<> 内的  
 cout << (\*it).m\_Name << " " << (\*it).m\_Age << endl;  
  
 }  
  
}

* 存放指针类型

void test14() {  
 vector<Person\*> v2;  
  
 Person p1("小花丸", 16);  
 Person p2("中善子", 17);  
 Person p3("大露比", 15);  
  
  
 v2.push\_back(&p1);  
 v2.push\_back(&p2);  
 v2.push\_back(&p3);  
  
 for (vector<Person\*>::iterator it = v2.begin(); it != v2.end(); it++) {  
 cout << (\*it)->m\_Name << " " << (\*it)->m\_Age << endl;  
  
 }  
  
}

## Vector 容器嵌套

  
 代码

#include <iostream>  
using namespace std;  
#include <string>  
#include <algorithm>  
#include <vector>  
  
// 知识点 ： Vector 嵌套 类似于多维数组  
  
void test15() {  
 vector<vector<int>> v;  
  
 vector<int> v1;  
 vector<int> v2;  
 vector<int> v3;  
  
 for (int i = 0; i < 4; i++) {  
 v1.push\_back(i + 1);  
 v2.push\_back(i + 3);  
 v3.push\_back(i + 5);  
 }  
  
 v.push\_back(v1);  
 v.push\_back(v2);  
 v.push\_back(v3);  
  
 for (vector<vector<int>>::iterator it = v.begin(); it != v.end(); it++) {  
 // \*it - > vector<int>  
 for (vector<int>::iterator vit = (\*it).begin(); vit != (\*it).end(); vit++) {  
 cout << \*vit << " ";  
 }  
 cout << endl;  
 }  
}  
  
  
int main() {  
  
 test15();  
 system("pause");  
 return 0;  
}