一．容器的分类

STL容器将运用的最广泛的数据结构实现出来。常用的数据结构：数组(array),链表(list),tree(树)，栈(stack),队列(queue),集合(set),映射表(map),

根据数据在容器中的排列特性，这些数据分为**序列式容器**和**关联式容器**两种。

1. 序列式容器

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，除非用删除或插入的操作改变这个位置。Vector容器、Deque容器、List容器等。

1. 关联式容器

关联式容器是非线性的二叉树结构。各元素之间没有严格的物理上的顺序关系，容器中的元素顺序并不是元素置入容器时的顺序。而是用一个键值key，作为值的索引，以查找存入的值。如Set/multiset容器 Map/multimap容器

二．算法

用有限的步骤解决逻辑上的问题，称为算法。

STL中收录的算法都是常用的算法，如查找，排序，计数，替换，删除等等。

三，迭代器

迭代器是一种抽象的设计概念，并不是一个实物。Iterator模式的定义：提供一种方法，能够依序访问某个容器中的各个元素，并且不会暴露该容器的内部表示形式。

迭代器的种类：

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

我们一般用的是可以读写的双向迭代器和随机访问迭代器。双向迭代器可以向前++，向后--。随机访问迭代器更加自由，功能最强。

四．STL三大件的初步使用

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<vector>

#include<string>

#include<algorithm>

using namespace std;

void myPrint(int v)

{

cout << v << endl;

}

void test01()

{

vector<int> v; //声明容器，存放int类型数据。类模板的写法

v.push\_back(10); //插入数据

v.push\_back(20);

//遍历容器,用迭代器

//vector<int>::iterator itBegin = v.begin(); //指向vector容器的起始位置

//vector<int>::iterator itEnd = v.end(); //指向vector容器的最后数据的后一个位置

//while (itBegin != itEnd)

//{

// cout << \*itBegin << endl;

// itBegin++;

//}

//同样使用迭代器遍历

//for (vector<int>::iterator it = v.begin(); it != v.end(); ++it)

//{

// cout << \*it << endl;

//}

//用算法遍历,自定义函数

for\_each(v.begin(), v.end(), myPrint);

}

//操作自定义数据类型

class Person

{

public:

Person(string name, int age)

{

this->m\_Name = name;

this->m\_Age = age;

}

string m\_Name;

int m\_Age;

};

void test02()

{

vector<Person> v;

Person p1("wxx", 10);

Person p2("mrwxx", 20);

v.push\_back(p1);

v.push\_back(p2);

//遍历

for (vector<Person>::iterator it = v.begin(); it != v.end(); ++it)

{

cout << "姓名：" << (\*it).m\_Name<< "年龄：" << (\*it).m\_Age << endl;

}

}

int main()

{

//test01();

test02();

return 0;

}