1、什么是标准模板库（STL）？

(1)C++标准模板库与C++标准库的关系

C++标准模板库其实属于C++标准库的一部分，C++标准模板库主要是定义了标准模板的定义与声明，而这些模板主要都是类模板，我们可以调用这些模板来定义一个具体的类；与之前的自己手动创建一个函数模版或者是类模板不一样，我们使用了STL就不用自己来创建模板了，这些模板都定义在标准模板库中，我们只需要学会怎么使用这些类模板来定义一个具体的类，然后能够使用类提供的各种方法来处理数据。

(2)STL六大组件：容器（containers）、算法（algorithms）、迭代器（iterators）、函数对象（functors）、适配器（adapters）、分配器（allocators）

2、迭代器

迭代器是一种对象，它能够用来遍历STL容器中的部分或全部元素，每个迭代器对象代表容器中的确定的地址，所以可以认为迭代器其实就是用来指向容器中数据的指针，我们可以通过改变这个指针来遍历容器中的所有元素。

3、容器

首先，我们必须理解一下什么是容器，对比我们生活当中的容器，例如水杯、桶、水瓶等等这些东西，其实他们都是容器，他们的一个共同点就是：都是用来存放液体的，能够用来存放一些东西；其实在我们的C++中说的这个容器其实作用也是用来存放"东西"，但是存放的是数据，在C++中容器就是一种用来存放数据的对象。

(1)C++中的容器其实是容器类实例化之后的一个具体的对象，那么可以将这个对象看成就是一个容器。

(2)因为C++中容器类是基于类模板定义的，也就是我们这里说的STL（标准模板类）。为什么需要做成模板的形式呢？因为我们的容器中存放的数据类型其实是相同的，如果就因为数据类型不同而要定义多个具体的类，这样就不合适，而模板恰好又能够解决这种问题，所以C++中的容器类是通过类模板的方式定义的，也就是STL。

(3)容器还有另一个特点是容器可以自行扩展。在解决问题时我们常常不知道我们需要存储多少个对象，也就是说我们不知道应该创建多大的内存空间来存放我们的数据。显然，数组在这一方面也力不从心。容器的优势就在这里，它不需要你预先告诉它你要存储多少对象，只要你创建一个容器对象，并合理的调用它所提供的方法，所有的处理细节将由容器来自身完成。它可以为你申请内存或释放内存，并且用最优的算法来执行您的命令。

(4)容器是随着面向对象语言的诞生而提出的，容器类在面向对象语言中特别重要，甚至它被认为是早期面向对象语言的基础。

4、容器的分类

STL对定义的通用容器分三类：**顺序性容器**、**关联式容器**和**容器适配器**。

我想说的是对于上面的每种类型的容器到底是是什么意思，其实没必要去搞懂，没什么价值，只要你能够大概理解知道即可，知道每种容器类型下有哪些具体的容器即可。

顺序性容器：vector、deque、list

关联性容器：set、multiset、map、multimap

容器适配器：stack、queue、

本文主要介绍vector、list和map 这3种容器。

5、vector向量

vector向量是一种顺序性容器。相当于数组，但其大小可以不预先指定，并且自动扩展。它可以像数组一样被操作，由于它的特性我们完全可以将vector 看作动态数组。在创建一个vector 后，它会自动在内存中分配一块连续的内存空间进行数据存储，初始的空间大小可以预先指定也可以由vector 默认指定。当存储的数据超过分配的空间时vector 会重新分配一块内存块，但这样的分配是很耗时的，在重新分配空间时它会做这样的动作：

首先，vector 会申请一块更大的内存块；

然后，将原来的数据拷贝到新的内存块中；

其次，销毁掉原内存块中的对象（调用对象的析构函数）；

最后，将原来的内存空间释放掉。

当vector保存的数据量很大时，如果此时进行插入数据导致需要更大的空间来存放这些数据量，那么将会大大的影响程序运行的效率，所以我们应该合理的使用vector。

 (1)初始化vector对象的方式：

vector<T> v1;     // 默认的初始化方式，内容为空

vector<T> v2(v1);   // v2是v1的一个副本

vector<T> v3(n, i)   // v3中包含了n个数值为i的元素

vector<T> v4(n);   // v4中包含了n个元素，每个元素的值都是0

(2)vector常用函数

empty()：判断向量是否为空，为空返回真，否则为假

begin()：返回向量（数组）的首元素地址

end()： 返回向量（数组）的末元素的下一个元素的地址

clear()：清空向量

front()：返回得到向量的第一个元素的数据

back()：返回得到向量的最后一个元素的数据

size()：返回得到向量中元素的个数

push\_back(数据)：将数据插入到向量的尾部

pop\_back()：删除向量尾部的数据

.....

(3)遍历方式

vector向量支持两种方式遍历，因为可以认为vector是一种动态数组，所以可以使用数组下标的方式，也可以使用迭代器

[复制代码](javascript:void(0);)

1 #include <iostream>

2 #include <vector>

3 #include <list>

4 #include <map>

5

6 using namespace std;

7

8 int main(void)

9 {

10 vector<int> vec;

11

12 vec.push\_back(1);

13 vec.push\_back(2);

14 vec.push\_back(3);

15 vec.push\_back(4);

16 vec.push\_back(5);

17

18 cout << "向量的大小：" << vec.size() << endl;

19

20 // 数组下标方式遍历vector

21 for (int i = 0; i < vec.size(); i++)

22 cout << vec[i] << " ";

23 cout << endl;

24

25 // 迭代器方式遍历vector

26 vector<int>::iterator itor = vec.begin();

27 for (; itor != vec.end(); itor++)

28 cout << \*itor << " ";

29 cout << endl;

30

31 return 0;

32 }

[复制代码](javascript:void(0);)

6、双向链表list

对于链表我不想多说了，我之前已经学过链表，对于一个双向链表来说主要包括3个：指向前一个链表节点的前向指针、有效数据、指向后一个链表节点的后向指针。链表相对于vector向量来说的优点在于：(a)动态的分配内存，当需要添加数据的时候不会像vector那样，先将现有的内存空间释放，在次分配更大的空间，这样的话效率就比较低了。(b)支持内部插入、头部插入和尾部插入。缺点：不能随机访问，不支持[]方式和vector.at()、占用的内存会多于vector（非有效数据占用的内存空间）。

(1)初始化list对象的方式

list<int> L0;    //空链表

list<int> L1(3);   //建一个含三个默认值是0的元素的链表

list<int> L2(5,2); //建一个含五个元素的链表，值都是2

list<int> L3(L2); //L3是L2的副本

list<int> L4(L1.begin(),L1.end());    //c5含c1一个区域的元素[begin, end]。

(2)list常用函数

begin()：返回list容器的第一个元素的地址

end()：返回list容器的最后一个元素之后的地址

rbegin()：返回逆向链表的第一个元素的地址（也就是最后一个元素的地址）

rend()：返回逆向链表的最后一个元素之后的地址（也就是第一个元素再往前的位置）

front()：返回链表中第一个数据值

back()：返回链表中最后一个数据值

empty()：判断链表是否为空

size()：返回链表容器的元素个数

clear()：清除容器中所有元素

insert(pos,num)：将数据num插入到pos位置处（pos是一个地址）

insert(pos,n,num)：在pos位置处插入n个元素num

erase(pos)：删除pos位置处的元素

push\_back(num)：在链表尾部插入数据num

pop\_back()：删除链表尾部的元素

push\_front(num)：在链表头部插入数据num

pop\_front()：删除链表头部的元素

sort()：将链表排序，默认升序

......

(3)遍历方式

双向链表list支持使用迭代器正向的遍历，也支持迭代器逆向的遍历，但是不能使用 [] 索引的方式进行遍历。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 #include <iostream>

2 #include <vector>

3 #include <list>

4 #include <map>

5

6 using namespace std;

7

8 int main(void)

9 {

10 list<int> l1;

11

12 // 插入元素方式演示

13 l1.push\_front(1); // 头部插入

14 l1.push\_back(2); // 尾部插入

15 l1.insert(l1.begin(), 3); // 开始位置插入

16 l1.insert(l1.end(), 4); // 结束位置插入

17

18 cout << "链表是否为空：" << l1.empty() << endl;

19 cout << "list链表中元素个数：" << l1.size() << endl;

20 cout << "list链表第一个元素：" << l1.front() << endl;

21 cout << "list链表最后一个元素：" << l1.back() << endl;

22

23 // 遍历链表正向

24 list<int>::iterator itor = l1.begin();

25 for (; itor != l1.end(); itor++)

26 cout << \*itor << " ";

27 cout << endl;

28

29 // 遍历链表逆向

30 list<int>::reverse\_iterator reitor = l1.rbegin();

31 for (; reitor != l1.rend(); reitor++)

32 cout << \*reitor << " ";

33 cout << endl;

34

35 // 将链表排序

36 l1.sort();

37 itor = l1.begin();

38 cout << "重新排序之后正向遍历：";

39 for (; itor != l1.end(); itor++)

40 cout << \*itor << " ";

41 cout << endl;

42

43 // 清除容器中的所有元素

44 l1.clear();

45 cout << "清除容器所有元素之后大小：" << l1.size() << endl;

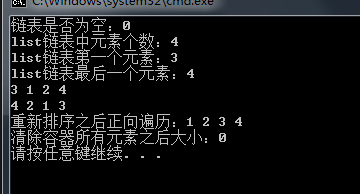
46

47 return 0;

48 }

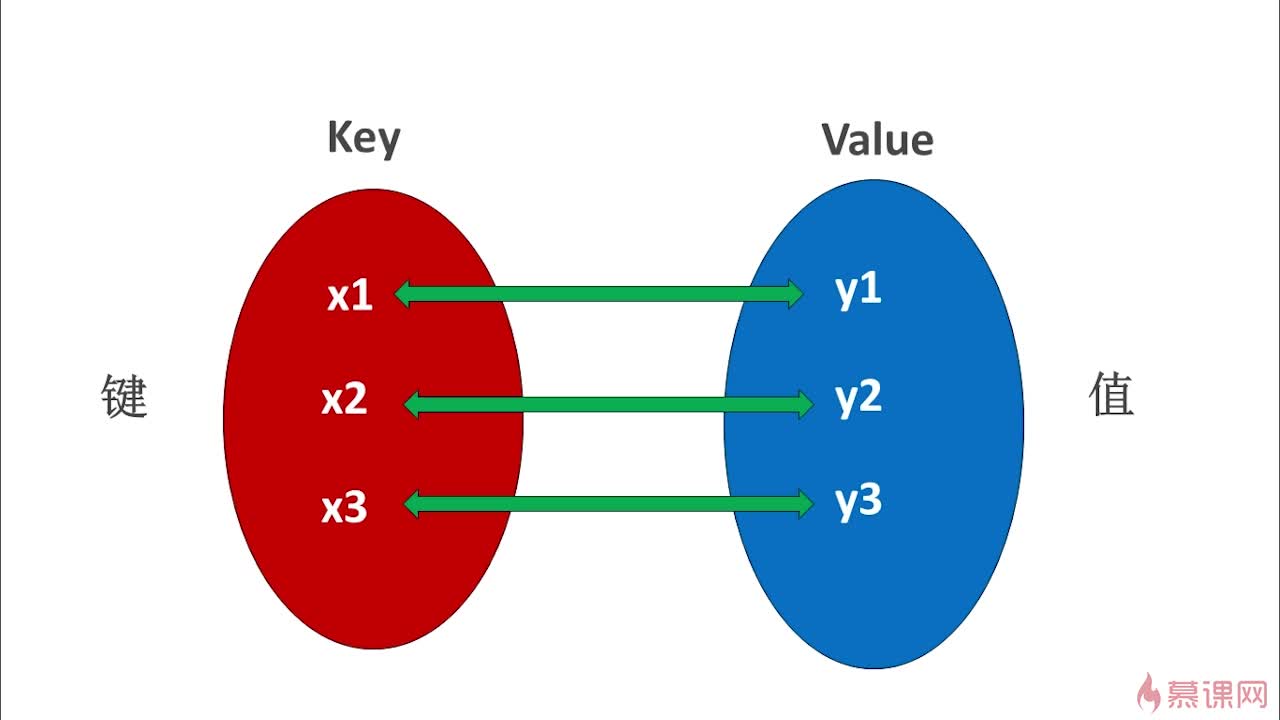
[复制代码](javascript:void(0);)

代码运行结果：



7、map

Map是STL的一个关联容器，它提供一对一（其中第一个可以称为关键字，每个关键字只能在map中出现一次，第二个可能称为该关键字的值）的数据处理能力，由于这个特性，它完成有可能在我们处理一对一数据的时候，在编程上提供快速通道。map内部自建一颗红黑树(一 种非严格意义上的平衡二叉树)，这颗树具有对数据自动排序的功能，所以在map内部所有的数据都是有序的。至于二叉树这种数据结构，本人暂时没有任何了解。在map这个容器中，提供一种“键- 值”关系的一对一的数据存储能力。其“键”在容器中不可重复，且按一定顺序排列，至于怎么排列，那么红黑树这种数据结构的特性了。



(1)初始化map对象的方式

map<int, string> m1 = { { 1, "guangzhou" }, { 2, "shenzhen" }, { 3, "changsha" } };   // 实例化一个map容器，还有3组数据  
map<char, string> m2;             // 实例化一个空map容器

(2)map常用函数

begin()：返回容器第一个元素的迭代器

end()：返回容器最后一个元素之后的迭代器

rbegin()：

rend()：

clera()：清除容器中所有元素

empty()：判断容器是否为空

insert(p1)：插入元素  p1 是通过pair函数建立的映射关系对

insert(pair<char, string>('S', "shenzhen"))： 插入元素

size()：返回容器中元素的个数

count()：返回指定键对应的数据的出现的次数

get\_allocator()：返回map的配置器

swap()：交换两个map容器的元素

.....

(3)遍历方式

map容器支持迭代器正向方式遍历和迭代器反向方式遍历，同时也支持 [] 方式访问数据，[]中的索引值是键值，这个一定要清楚  
https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 #include <iostream>

2 #include <stdio.h>

3 #include <string>

4 #include <stdlib.h>

5 #include <vector>

6 #include <list>

7 #include <map>

8

9 using namespace std;

10

11 int main(void)

12 {

13 map<int, string> m1 = { { 1, "guangzhou" }, { 2, "shenzhen" }, { 3, "changsha" } };

14 map<char, string> m2;

15

16 // 建立映射关系对

17 pair<char, string> p1('G', "guangzhou");

18 pair<char, string> p2('S', "guangzhou");

19 pair<char, string> p3('C', "changsha");

20

21 // 插入数据

22 m2.insert(p1);

23 m2.insert(p2);

24 m2.insert(p3);

25

26 cout << "map容器m1元素个数：" << m1.size() << endl;

27 cout << "map容器m2元素个数：" << m2.size() << endl;

28

29 // 采用 [] 方式打印数据

30 cout << m1[1] << " " << m1[2] << " " << m1[3] << endl;

31 cout << m2['G'] << " " << m2['S'] << " " << m2['C'] << endl;

32

33 // 迭代器正向方式遍历

34 map<int, string>::iterator itor = m1.begin();

35 for (; itor != m1.end(); itor++)

36 {

37 cout << itor->first << ",";

38 cout << itor->second << endl;

39 }

40

41 // 迭代器反向方式遍历

42 map<char, string>::reverse\_iterator reitor = m2.rbegin();

43 for (; reitor != m2.rend(); reitor++)

44 {

45 cout << reitor->first << ",";

46 cout << reitor->second << endl;

47 }

48

49 // 清空容器

50 m1.clear();

51 m2.clear();

52

53 return 0;

54 }

[复制代码](javascript:void(0);)

8、顺序性容器和关联容器（本段来自其他博客，在此感谢）

(1)关联容器对元素的插入和删除操作比vector要快，因为vector是顺序存储，而关联容器是链式存储；比list 要慢，是因为即使它们同是链式结构，但list 是线性的，而关联容器是二叉树结构，其改变一个元素涉及到其它元素的变动比list 要多，并且它是排序的，每次插入和删除都需要对元素重新排序；

(2)关联容器对元素的检索操作比vector 慢，但是比list 要快很多。vector 是顺序的连续存储，当然是比不上的，但相对链式的list 要快很多是因为list 是逐个搜索，它搜索的时间是跟容器的大小成正比，而关联容器 查找的复杂度基本是Log(N) ，比如如果有1000 个记录，最多查找10 次，1,000,000 个记录，最多查找20 次。容器越大，关联容器相对list 的优越性就越能体现；

**参考博客： http://www.cnblogs.com/xkfz007/articles/2534249.html**

**http://www.cnblogs.com/scandy-yuan/archive/2013/01/08/2851324.html**