<https://blog.csdn.net/yas12345678/article/details/52601578>

**1.关于list容器**

list是一种序列式容器。list容器完成的功能实际上和数据结构中的双向链表是极其相似的，list中的数据元素是通过链表指针串连成逻辑意义上的线性表，也就是list也具有链表的主要优点，即：在链表的任一位置进行元素的插入、删除操作都是快速的。list的实现大概是这样的：list的每个节点有三个域：前驱元素指针域、数据域和后继元素指针域。前驱元素指针域保存了前驱元素的首地址；数据域则是本节点的数据；后继元素指针域则保存了后继元素的首地址。其实，list和循环链表也有相似的地方，即：头节点的前驱元素指针域保存的是链表中尾元素的首地址，list的尾节点的后继元素指针域则保存了头节点的首地址，这样，list实际上就构成了一个双向循环链。由于list元素节点并不要求在一段连续的内存中，显然在list中是不支持快速随机存取的，因此对于迭代器，只能通过“++”或“--”操作将迭代器移动到后继/前驱节点元素处。而不能对迭代器进行+n或-n的操作，这点，是与vector等不同的地方。

**我想把三个常用的序列式放在一起对比一下是有必要的：**

**vector ：**vector和built-in数组类似，拥有一段连续的内存空间，能非常好的支持随即存取，即[]操作符，但由于它的内存空间是连续的，所以在中间进行插入和删除会造成内存块的拷贝，另外，当插入较多的元素后，预留内存空间可能不够，需要重新申请一块足够大的内存并把原来的数据拷贝到新的内存空间。这些影响了vector的效率，但是实际上用的最多的还是vector容器，建议大多数时候使用vector效率一般是不错的。vector的用法解析可以参考本人的另一篇随笔：<http://www.cnblogs.com/BeyondAnyTime/archive/2012/08/08/2627666.html>

**list：**list就是数据结构中的双向链表(根据sgi stl源代码)，因此它的内存空间是不连续的，通过指针来进行数据的访问，这个特点使得它的随即存取变的非常没有效率，因此它没有提供[]操作符的重载。但由于链表的特点，它可以以很好的效率支持任意地方的删除和插入。

**deque：**deque是一个double-ended queue，它的具体实现不太清楚，但知道它具有以下两个特点：它支持[]操作符，也就是支持随即存取，并且和vector的效率相差无几，它支持在两端的操作：push\_back,push\_front,pop\_back,pop\_front等，并且在两端操作上与list的效率也差不多。

因此在实际使用时，如何选择这三个容器中哪一个，应根据你的需要而定，具体可以遵循下面的**原则**：  
1. 如果你需要高效的随即存取，而不在乎插入和删除的效率，使用vector  
2. 如果你需要大量的插入和删除，而不关心随即存取，则应使用list  
3. 如果你需要随即存取，而且关心两端数据的插入和删除，则应使用deque。

**2.list中常用的函数**

**2.1list中的构造函数：**

list() 声明一个空列表；

list(n) 声明一个有n个元素的列表，每个元素都是由其默认构造函数T()构造出来的

list(n,val) 声明一个由n个元素的列表，每个元素都是由其复制构造函数T(val)得来的

list(n,val) 声明一个和上面一样的列表

list(first,last) 声明一个列表，其元素的初始值来源于由区间所指定的序列中的元素

**2.2 begin()和end()：**通过调用list容器的成员函数begin()得到一个指向容器起始位置的iterator，可以调用list容器的 end() 函数来得到list末端下一位置，相当于：int a[n]中的第n+1个位置a[n]，实际上是不存在的，不能访问，经常作为循环结束判断结束条件使用。

**2.3 push\_back() 和push\_front()：**使用list的成员函数push\_back和push\_front插入一个元素到list中。其中push\_back()从list的末端插入，而 push\_front()实现的从list的头部插入。

**2.4 empty()：**利用empty() 判断list是否为空。

**2.5 resize()：** 如果调用resize(n)将list的长度改为只容纳n个元素，超出的元素将被删除，如果需要扩展那么调用默认构造函数T()将元素加到list末端。如果调用resize(n,val)，则扩展元素要调用构造函数T(val)函数进行元素构造，其余部分相同。

**2.6 clear()：** 清空list中的所有元素。

**2.7 front()和back()：**通过front()可以获得list容器中的头部元素，通过back()可以获得list容器的最后一个元素。但是有一点要注意，就是list中元素是空的时候，这时候调用front()和back()会发生什么呢？实际上会发生不能正常读取数据的情况，但是这并不报错，那我们编程序时就要注意了，个人觉得在使用之前最好先调用empty()函数判断list是否为空。

**2.8 pop\_back和pop\_front()：**通过删除最后一个元素，通过pop\_front()删除第一个元素；序列必须不为空，如果当list为空的时候调用pop\_back()和pop\_front()会使程序崩掉。

**2.9 assign()：**具体和vector中的操作类似，也是有两种情况，第一种是：l1.assign(n,val)将 l1中元素变为n个T(val）。第二种情况是：l1.assign(l2.begin(),l2.end())将l2中的从l2.begin()到l2.end()之间的数值赋值给l1。

**2.10 swap()：**交换两个链表(两个重载)，一个是l1.swap(l2); 另外一个是swap(l1,l2)，都可能完成连个链表的交换。

**2.11 reverse()：**通过reverse()完成list的逆置。

**2.12 merge()：**合并两个链表并使之默认升序(也可改)，l1.merge(l2，greater<int>()); 调用结束后l2变为空，l1中元素包含原来l1 和 l2中的元素，并且排好序，升序。其实默认是升序，greater<int>()可以省略，另外greater<int>()是可以变的，也可以不按升序排列。

看一下下面的程序：

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

复制代码

1 #include <iostream>

2 #include <list>

3

4 using namespace std;

5

6 int main()

7 {

8 list<int> l1;

9 list<int> l2(2,0);

10 list<int>::iterator iter;

11 l1.push\_back(1);

12 l1.push\_back(2);

13 l2.push\_back(3);

14 l1.merge(l2,greater<int>());//合并后升序排列，实际上默认就是升序

15 for(iter = l1.begin() ; iter != l1.end() ; iter++)

16 {

17 cout<<\*iter<<" ";

18 }

19 cout<<endl<<endl;

20 if(l2.empty())

21 {

22 cout<<"l2 变为空 ！！";

23 }

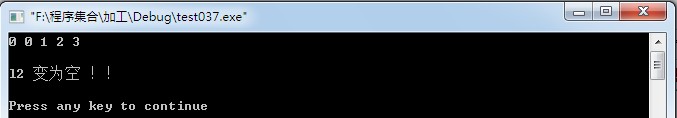
24 cout<<endl<<endl;

25 return 0;

26 }

复制代码

运行结果：



**2.13 insert()：**在指定位置插入一个或多个元素(三个重载)：

l1.insert(l1.begin(),100); 在l1的开始位置插入100。

l1.insert(l1.begin(),2,200); 在l1的开始位置插入2个100。

l1.insert(l1.begin(),l2.begin(),l2.end());在l1的开始位置插入l2的从开始到结束的所有位置的元素。

**2.14 erase()：**删除一个元素或一个区域的元素(两个重载)

l1.erase(l1.begin()); 将l1的第一个元素删除。

l1.erase(l1.begin(),l1.end()); 将l1的从begin()到end()之间的元素删除。