STL中的线性数据结构

主要内容：

1. Vector
2. List
3. Stack
4. Queue
5. Deque

Vector:

STL中的vector，是我们常用的线性表，底层是用数组实现的，我们可以把vector当作普通的数组来用，但vector是动态数组，支持动态扩容，简单点说，就是初始化是可以指定与不指定数组大小。由于vector是用数组实现的，支持快速访问。在Vector支持的基本操作中（添加、插入、删除），内部实现都与我们做的基于数组实现的线性表类似，其时间复杂度的分析也大同小异，我们要根据实际情况选择不同的线性表。

基本操作：
assign() 对Vector中的元素赋值
at() 返回指定位置的元素
back() 返回最末一个元素
begin() 返回第一个元素的迭代器
capacity() 返回vector所能容纳的元素数量(在不重新分配内存的情况下)
clear() 清空所有元素
empty() 判断Vector是否为空（返回true时为空）
end() 返回最末元素的迭代器(译注:实指向最末元素的下一个位置)
erase() 删除指定元素
front() 返回第一个元素
get_allocator() 返回vector的内存分配器
insert() 插入元素到Vector中
max_size() 返回Vector所能容纳元素的最大数量（上限）
pop_back() 移除最后一个元素
push_back() 在Vector最后添加一个元素
rbegin() 返回Vector尾部的逆迭代器
rend() 返回Vector起始的逆迭代器
reserve() 设置Vector最小的元素容纳数量
resize() 改变Vector元素数量的大小
size() 返回Vector元素数量的大小
swap() 交换两个Vector

List:

List的底层实现是双向链表，至于为什么使用双向链表而不使用单向链表呢？个人认为，双向链表虽然增加了一定的时间代价，但能支持的功能也多了一点。在单向链表中，如果我们要找到某个节点的前驱节点，那就必须要从头遍历，有一定是时间代价，但如果是双向链表，这就很容易了。List也是一种线性表，基于链表实现，内部功能的实现与我们自己用链表实现线性表时是类似的。List不支持快速访问，只能从根节点遍历，但由于是双向链表，有一个加速的操作，有点类似与折半查找，详情是这样的：若index < 双向链表长度的1/2，则从前向后查找;否则，从后向前查找，这也是使用双向链表的优势。内部功能的时间复杂度分析与我们自己实现是的复杂度基本一致。

基本函数：

```
c.insert(pos,num);//在pos位置插入元素num
c.insert(pos,n,num);//在pos位置插入n个元素num
c.erase(pow);//删除pos位置的元素

c.push_back(num);//在末尾增加一个元素
c.pop_back();//删除末尾的元素
c.push_front(num);//在开始位置增加一个元素
c.pop_front();//删除第一个元素

c.reverse();//翻转链表
c.sort();//将链表排序，默认升序
c.sort(cmp);//自定义小于函数
```

queue

queue单向队列与栈有点类似，一个是在同一端存取数据，另一个是在一端存入数据，另一端取出数据。单向队列中的数据是先进先出（First In First Out,FIFO）。在STL中，单向队列也是以别的容器作为底部结构，再将接口改变，使之符合单向队列的特性就可以了。因此实现也是非常方便的。下面就给出单向队列的函数列表和VS2008中单向队列的源代码。单向队列一共6个常用函数（front()、back()、push()、pop()、empty()、size()），与栈的常用函数较为相似。

queue 的基本操作有：

入队，如例：q.push(x); 将x 接到队列的末端。
出队，如例：q.pop(); 弹出队列的第一个元素，注意，并不会返回被弹出元素的值。
访问队首元素，如例：q.front()，即最早被压入队列的元素。
访问队尾元素，如例：q.back()，即最后被压入队列的元素。
判断队列空，如例：q.empty()，当队列空时，返回true。
访问队列中的元素个数，如例：q.size()

deque

deque双向队列是一种双向开口的连续线性空间，可以高效的在头尾两端插入和删除元素，deque在接口上和vector非常相似. deque的实现比较复杂，内部会维护一个map（注意！不是STL中的map容器）即一小块连续的空间，该空间中每个元素都是指针，指向另一段（较大的）区域，这个区域称为缓冲区，缓冲区用来保存deque中的数据。因此deque在随机访问和遍历数据会比vector慢。

基本用法：

deque.begin();//返回指向第一个元素的迭代器
deque.end();//返回指向最后一个元素下一个位置的迭代器
deque.rbegin();
deque.rend();//反向迭代器

deque.empty();//判断deque是否空
deque.front();//返回第一个元素
deque.back();//返回最后一个元素
deque.size();//返回容器大小
deque.clear();//清除deque

deque.erase(pos);//删除pos位置的元素
deque.push_back(num);//在末尾插入元素
deque.pop_back();//弹出末尾的元素
deque.pop_front();//删除开头位置的元素

statck

栈（statck）这种数据结构在计算机中是相当出名的。栈中的数据是先进后出的（First In Last Out, FILO）。栈只有一个出口，允许新增元素（只能在栈顶上增加）、移出元素（只能移出栈顶元素）、取得栈顶元素等操作。在STL中，栈是以别的容器作为底部结构，再将接口改变，使之符合栈的特性就可以了。因此实现非常的方便。下面就给出栈的函数列表和VS2008中栈的源代码，在STL中栈一共就5个常用操作函数（top()、push()、pop()、 size()、empty() ），很好记的。

基本用法：

empty();//判断是否为空
push(class T);//栈顶压入一元素
pop();//弹出栈顶元素
top();//返回栈顶元素
size();//返回栈中元素个数

最后的干货：

一个小疑问：

既然 STL 中已经为我们封装好了这些常用的数据结构？为什么还要我们自己实现呢？这是我们经常会有的疑问。

如果自己实现的这些常用的数据结构，我们就能更好的理解这些不同的数据结构的优缺点，就能更好地根据实际情况选择合适的数据结构，使我们的程序显得更加专业。同时，自己实现过，也能懂得 STL 中封装的方法的实现，帮助我们更好地实用。

一个小建议：

如果有机会有能力有兴趣，我们可以看一下 STL 的源码，从中学习别人是如何把这些数据结构封装起来的，是如何实现各种方法的，我相信我们能从中学习到不少，无论是在写代码还是思想上。