C++内置的数组支持容器的机制，但是它不支持容器抽象的语义。要解决此问题我们自己实现这样的类。在标准C++中，用容器向量（vector）实现。容器向量也是一个类模板。  
标准库vector类型使用需要的头文件：#include <vector>。vector 是一个类模板。不是一种数据类型，vector<int>是一种数据类型。Vector的存储空间是连续的,list不是连续存储的。

**一、 定义和初始化**  
vector< typeName > v1;       //默认v1为空，故下面的赋值是错误的v1[0]=5;  
vector<typeName>v2(v1); 或v2=v1;或vector<typeName> v2(v1.begin(), v1.end());//v2是v1的一个副本，若v1.size（）>v2.size()则赋值后v2.size()被扩充为v1.size()。  
vector< typeName > v3(n,i);//v3包含n个值为i的typeName类型元素  
vector< typeName > v4(n); //v4含有n个值为0的元素  
int a[4]={0,1,2,3,3}; vector<int> v5(a,a+5);//v5的size为5，v5被初始化为a的5个值。后一个指针要指向将被拷贝的末元素的下一位置。  
vector<int> v6(v5);//v6是v5的拷贝  
vector< 类型 > [标识符](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=%B1%EA%CA%B6%B7%FB&k0=%B1%EA%CA%B6%B7%FB&kdi0=0&luki=5&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)（最大容量，初始所有值）；

**二、 值初始化**  
1>     如果没有指定元素初始化式，标准库自行提供一个初始化值进行值初始化。  
2>     如果保存的式含有构造函数的类类型的元素，标准库使用该类型的构造函数初始化。  
3>     如果保存的式没有构造函数的类类型的元素，标准库产生一个带初始值的对象，使用这个对象进行值初始化。

**三、vector对象最重要的几种操作**  
1. v.push\_back(t)    　　　　在容器的最后添加一个值为t的数据，容器的size变大。  
                             　　　　另外list有push\_front()[函数](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=%BA%AF%CA%FD&k0=%BA%AF%CA%FD&kdi0=0&luki=8&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)，在前端插入，后面的元素下标依次增大。  
2. v.size()              　　　　返回容器中数据的个数，size返回相应vector类定义的size\_type的值。v.resize(2\*v.size)或

    v.resize(2\*v.size, 99)　　 将v的容量翻倍(并把新元素的值初始化为99)  
3. v.empty()     　　　　　　判断vector是否为空  
4. v[n]  或  v.at(n)         　　　　　　　返回v中位置为n的元素，后者更加安全  
5. v.insert(pointer,number, content)    向v中pointer指向的位置插入number个content的内容。  
    还有v. insert(pointer, content)，v.insert(pointer,a[2],a[4])将a[2]到a[4]三个元素插入。  
6. v.pop\_back()    删除容器的末元素，并不返回该元素。  
7. v.erase(pointer1,pointer2) 删除pointer1到pointer2中间（包括pointer1所指）的元素。  
    vector中删除一个元素后，此位置以后的元素都需要往前移动一个位置，虽然当前迭代器位置没有自动加1，  
    但是由于后续元素的顺次前移，也就相当于迭代器的自动指向下一个位置一样。  
8. v1==v2          判断v1与v2是否相等。  
9. ！=、<、<=、>、>=      保持这些操作符惯有含义。  
10. vector<typeName>::iterator p=v1.begin( ); p初始值指向v1的第一个元素。\*p取所指向元素的值。  
      对于const vector<typeName>只能用vector<typeName>::const\_iterator类型的指针访问。  
11. p=v1.end( ); p指向v1的最后一个元素的下一位置。  
12. v.clear()      删除容器中的所有元素。

#include<algorithm>中的泛函算法  
搜索算法：find() 、search() 、count() 、find\_if() 、search\_if() 、count\_if()   
分类排序：sort() 、merge()   
删除算法：unique() 、remove()   
生成和变异：generate() 、fill() 、transformation() 、copy()   
关系[算法](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=%CB%E3%B7%A8&k0=%CB%E3%B7%A8&kdi0=0&luki=2&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)：equal() 、min() 、max()   
sort(v1.begin(),vi.begin()+v1.size/2）; 对v1的前半段元素排序  
list<char>::iterator pMiddle =find(cList.begin(),cList.end(),'A');找到则返回被查内容第一次出现处指针，否则返回end()。  
vector< typeName >::size\_type x ; vector< typeName >类型的计数，可用于循环如同for(int i)

初学C++的[程序员](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=%B3%CC%D0%F2%D4%B1&k0=%B3%CC%D0%F2%D4%B1&kdi0=0&luki=4&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)可能会认为vector的下标操作可以添加元素，其实不然：

vector<int> ivec; // empty [vector](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=vector&k0=vector&kdi0=0&luki=10&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)

for (vector<int>::size\_type ix = 0; ix != 10; ++ix) ivec[ix] = ix; // disaster: ivec has no elements

上述程序试图在ivec中插入10个新元素，元素值依次为0到9的整数。但是，这里ivec是空的vector对象，而且下标只能用于获取已存在的元素。

这个循环的正确写法应该是：

for (vector<int>::size\_type ix = 0; ix != 10; ++ix) ivec.push\_back(ix); // ok: adds new element with value ix

警告：必须是已存在的元素才能用下标操作符进行索引。通过下标操作进行赋值时，不会添加任何元素。仅能对确知已存在的元素进行下标操作。

**四、内存管理与效率**

      1.使用reserve()函数提前设定容量大小，避免多次容量扩充操作导致效率低下。

      关于STL容器，最令人称赞的特性之一就是是只要不超过它们的最大大小，它们就可以自动增长到足以容纳你放进去的数据。（要知道这个最大值，只要调用名叫max\_size的成员函数。）对于vector和string，如果需要更多空间，就以类似realloc的思想来增长大小。vector容器支持随机访问，因此为了提高效率，它内部使用动态数组的方式实现的。在通过 reserve() 来申请特定大小的时候总是按指数边界来增大其内部[缓冲区](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=%BB%BA%B3%E5%C7%F8&k0=%BB%BA%B3%E5%C7%F8&kdi0=0&luki=3&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)。当进行insert或push\_back等增加元素的操作时，如果此时动态数组的内存不够用，就要动态的重新分配当前大小的1.5~2倍的新内存区，再把原数组的内容复制过去。所以，在一般情况下，其访问速度同一般数组，只有在重新分配发生时，其性能才会下降。正如上面的代码告诉你的那样。而进行pop\_back操作时，capacity并不会因为vector容器里的元素减少而有所下降，还会维持操作之前的大小。对于vector容器来说，如果有大量的数据需要进行push\_back，应当使用reserve()[函数](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=%BA%AF%CA%FD&k0=%BA%AF%CA%FD&kdi0=0&luki=8&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)提前设定其容量大小，否则会出现许多次容量扩充操作，导致效率低下。

      reserve成员[函数](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=%BA%AF%CA%FD&k0=%BA%AF%CA%FD&kdi0=0&luki=8&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)允许你最小化必须进行的重新分配的次数，因而可以避免真分配的开销和迭代器/指针/引用失效。但在我解释reserve为什么可以那么做之前，让我简要介绍有时候令人困惑的四个相关成员函数。在标准容器中，只有vector和string提供了所有这些函数。

(1) size()告诉你容器中有多少元素。它没有告诉你容器为它容纳的元素分配了多少内存。   
(2) capacity()告诉你容器在它已经分配的内存中可以容纳多少元素。那是容器在那块内存中总共可以容纳多少元素，而不是还可以容纳多少元素。如果你想知道一个vector或string中有多少没有被占用的内存，你必须从capacity()中减去size()。如果size和capacity返回同样的值，容器中就没有剩余空间了，而下一次插入（通过insert或push\_back等）会引发上面的重新分配步骤。   
(3) resize(Container::size\_type n)强制把容器改为容纳n个元素。调用resize之后，size将会返回n。如果n小于当前大小，容器尾部的元素会被[销毁](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=%CF%FA%BB%D9&k0=%CF%FA%BB%D9&kdi0=0&luki=1&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)。如果n大于当前大小，新默认构造的元素会添加到容器尾部。如果n大于当前容量，在元素加入之前会发生重新分配。   
(4) reserve(Container::size\_type n)强制容器把它的容量改为至少n，提供的n不小于当前大小。这一般强迫进行一次重新分配，因为容量需要增加。（如果n小于当前容量，vector忽略它，这个调用什么都不做，string可能把它的容量减少为size()和n中大的数，但string的大小没有改变。在我的经验中，使用reserve来从一个string中修整多余容量一般不如使用“交换技巧”，那是条款17的主题。）

     这个简介表示了只要有元素需要插入而且容器的容量不足时就会发生重新分配（包括它们维护的原始内存分配和回收，对象的拷贝和析构和迭代器、指针和引用的失效）。所以，避免重新分配的关键是使用reserve尽快把容器的容量设置为足够大，最好在容器被构造之后立刻进行。

例如，假定你想建立一个容纳1-1000值的vector<int>。没有使用reserve，你可以像这样来做：

vector<int> v; for (int i = 1; i <= 1000; ++i) v.push\_back(i);

在大多数STL实现中，这段代码在循环过程中将会导致2到10次重新分配。（10这个数没什么奇怪的。记住[vector](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=vector&k0=vector&kdi0=0&luki=10&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)在重新分配发生时一般把容量翻倍，而1000约等于210。）

把代码改为使用reserve，我们得到这个：

vector<int> v; v.reserve(1000); for (int i = 1; i <= 1000; ++i) v.push\_back(i);

这在循环中不会发生重新分配。

在大小和容量之间的关系让我们可以预言什么时候插入将引起vector或string执行重新分配，而且，可以预言什么时候插入会使指向容器中的迭代器、指针和引用失效。例如，给出这段代码，

string s; ... if (s.size() < s.capacity()) { s.push\_back('x'); }

push\_back的调用不会使指向这个string中的迭代器、指针或引用失效，因为string的容量保证大于它的大小。如果不是执行push\_back，代码在string的任意位置进行一个insert，我们仍然可以保证在插入期间没有发生重新分配，但是，与伴随string插入时迭代器失效的一般规则一致，所有从插入位置到string结尾的迭代器/指针/引用将失效。

回到本条款的主旨，通常有两情况使用reserve来避免不必要的重新分配。第一个可用的情况是当你确切或者大约知道有多少元素将最后出现在容器中。那样的话，就像上面的[vector](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=vector&k0=vector&kdi0=0&luki=10&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)代码，你只是提前reserve适当数量的空间。第二种情况是保留你可能需要的最大的空间，然后，一旦你添加完全部数据，修整掉任何多余的容量。

       2.使用“交换技巧”来修整vector过剩空间/内存

      有一种方法来把它从曾经最大的容量减少到它现在需要的容量。这样减少容量的方法常常被称为“收缩到合适（shrink to fit）”。该方法只需一条语句：vector<int>(ivec).swap(ivec);  
表达式vector<int>(ivec)建立一个临时[vector](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=vector&k0=vector&kdi0=0&luki=10&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)，它是ivec的一份拷贝：vector的拷贝构造函数做了这个工作。但是，vector的拷贝构造函数只分配拷贝的元素需要的内存，所以这个临时vector没有多余的容量。然后我们让临时vector和ivec交换数据，这时我们完成了，ivec只有临时变量的修整过的容量，而这个临时变量则持有了曾经在ivec中的没用到的过剩容量。在这里（这个语句结尾），临时vector被[销毁](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=%CF%FA%BB%D9&k0=%CF%FA%BB%D9&kdi0=0&luki=1&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)，因此释放了以前ivec使用的内存，收缩到合适。

     3.用swap方法强行释放STL Vector所占内存

template < class T> void ClearVector( vector<T>& v ) { vector<T> vtTemp; vtTemp.swap( v ); }

如

vector<int> v ; nums.push\_back(1); nums.push\_back(3); nums.push\_back(2); nums.push\_back(4); vector<int>().swap(v); /\* 或者v.swap(vector<int>()); \*/

/\*或者{ std::[vector](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=vector&k0=vector&kdi0=0&luki=10&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)<int> tmp = v; v.swap(tmp); }; //加大括号{ }是让tmp退出{ }时自动析构\*/

**五、Vector 内存管理成员函数的行为测试**  
C++ STL的vector使用非常广泛，但是对其内存的管理模型一直有多种猜测，下面用实例代码测试来了解其内存管理方式，测试代码如下：

1 #include <iostream>

2 #include <vector>

3 using namespace [std](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=std&k0=std&kdi0=0&luki=7&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0); 4

5 int main() 6 { 7 [vector](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=vector&k0=vector&kdi0=0&luki=10&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)<int> iVec; 8 cout << "容器 大小为: " << iVec.size() << endl; 9 cout << "容器 容量为: " << iVec.capacity() << endl; //1个元素， 容器容量为1

10

11 iVec.push\_back(1); 12 cout << "容器 大小为: " << iVec.size() << endl; 13 cout << "容器 容量为: " << iVec.capacity() << endl; //2个元素， 容器容量为2

14

15 iVec.push\_back(2); 16 cout << "容器 大小为: " << iVec.size() << endl; 17 cout << "容器 容量为: " << iVec.capacity() << endl; //3个元素， 容器容量为4

18

19 iVec.push\_back(3); 20 cout << "容器 大小为: " << iVec.size() << endl; 21 cout << "容器 容量为: " << iVec.capacity() << endl; //4个元素， 容器容量为4

22

23 iVec.push\_back(4); 24 iVec.push\_back(5); 25 cout << "容器 大小为: " << iVec.size() << endl; 26 cout << "容器 容量为: " << iVec.capacity() << endl; //5个元素， 容器容量为8

27

28 iVec.push\_back(6); 29 cout << "容器 大小为: " << iVec.size() << endl; 30 cout << "容器 容量为: " << iVec.capacity() << endl; //6个元素， 容器容量为8

31

32 iVec.push\_back(7); 33 cout << "容器 大小为: " << iVec.size() << endl; 34 cout << "容器 容量为: " << iVec.capacity() << endl; //7个元素， 容器容量为8

35

36 iVec.push\_back(8); 37 cout << "容器 大小为: " << iVec.size() << endl; 38 cout << "容器 容量为: " << iVec.capacity() << endl; //8个元素， 容器容量为8

39

40 iVec.push\_back(9); 41 cout << "容器 大小为: " << iVec.size() << endl; 42 cout << "容器 容量为: " << iVec.capacity() << endl; //9个元素， 容器容量为16

43 /\* vs2005/8 容量增长不是翻倍的，如 44 9个元素 容量9 45 10个元素 容量13 \*/

46

47 /\* 测试effective stl中的特殊的交换 swap() \*/

48 cout << "当前vector 的大小为: " << iVec.size() << endl; 49 cout << "当前vector 的容量为: " << iVec.capacity() << endl; 50 vector<int>(iVec).swap(iVec); 51

52 cout << "临时的vector<int>对象 的大小为: " << (vector<int>(iVec)).size() << endl; 53 cout << "临时的vector<int>对象 的容量为: " << (vector<int>(iVec)).capacity() << endl; 54 cout << "交换后，当前vector 的大小为: " << iVec.size() << endl; 55 cout << "交换后，当前[vector](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=vector&k0=vector&kdi0=0&luki=10&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0) 的容量为: " << iVec.capacity() << endl; 56

57 return 0; 58 }

**六、vector的其他成员函数**

c.assign(beg,end)  
将[beg; end)区间中的数据赋值给c。beg和end是迭代器变量  
c.assign(n,elem)  
将n个elem的拷贝赋值给c。   
c.at(idx)  
传回索引idx所指的数据，如果idx[越界](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=%D4%BD%BD%E7&k0=%D4%BD%BD%E7&kdi0=0&luki=6&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)，抛出out\_of\_range。   
c.back()  
传回最后一个数据，不检查这个数据是否存在。  
c.front()  
传回第一个数据。   
get\_allocator  
使用构造[函数](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=%BA%AF%CA%FD&k0=%BA%AF%CA%FD&kdi0=0&luki=8&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)返回一个拷贝。   
c.rbegin()  
传回一个逆向队列的第一个数据。   
c.rend()  
传回一个逆向队列的最后一个数据的下一个位置。   
c.~ [vector](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=vector&k0=vector&kdi0=0&luki=10&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0) <Elem>()  
[销毁](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=16&is_app=0&jk=9a0b58f5c8784b2f&k=%CF%FA%BB%D9&k0=%CF%FA%BB%D9&kdi0=0&luki=1&n=10&p=baidu&q=xiaoxiaobai_cpr&rb=0&rs=1&seller_id=1&sid=2f4b78c8f5580b9a&ssp2=1&stid=0&t=tpclicked3_hc&td=1972324&tu=u1972324&u=http%3A%2F%2Fwww%2E52ij%2Ecom%2Fjishu%2F5523%2Ehtml&urlid=0)所有数据，释放内存。