[**std::vector介绍**](http://www.cnblogs.com/yxnchinahlj/archive/2011/03/06/1972435.html)

vector是C++标准模板库中的部分内容，它是一个多功能的，能够操作多种数据结构和算法的模板类和函数库。vector之所以被认为是一个容器，是因为它能够像容器一样存放各种类型的对象，简单地说，vector是一个能够存放任意类型的动态数组，能够增加和压缩数据。  
为了可以使用vector，必须在你的头文件中包含下面的代码：  
#include <vector>  
vector属于std命名域的，因此需要通过命名限定，如下完成你的代码：  
using std::vector;  
vector<int> vInts;  
或者连在一起，使用全名：  
std::vector<int> vInts;  
  
建议使用全局的命名域方式：usingnamespace std;   
函数   
表述   
c.assign(beg,end)  
c.assign(n,elem) 将[beg; end)区间中的数据赋值给c。  
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，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.begin() 传回迭代器中的第一个数据地址。   
c.capacity() 返回容器中数据个数。   
c.clear() 移除容器中所有数据。   
c.empty() 判断容器是否为空。   
c.end() 指向迭代器中的最后一个数据地址。   
c.erase(pos)  
c.erase(beg,end) 删除pos位置的数据，传回下一个数据的位置。  
删除[beg,end)区间的数据，传回下一个数据的位置。   
c.front() 传回第一个数据。   
get\_allocator 使用构造函数返回一个拷贝。   
c.insert(pos,elem)   
c.insert(pos,n,elem)  
c.insert(pos,beg,end) 在pos位置插入一个elem拷贝，传回新数据位置。  
在pos位置插入n个elem数据。无返回值。  
在pos位置插入在[beg,end)区间的数据。无返回值。   
c.max\_size() 返回容器中最大数据的数量。   
c.pop\_back() 删除最后一个数据。   
c.push\_back(elem) 在尾部加入一个数据。   
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.resize(num) 重新指定队列的长度。   
c.reserve() 保留适当的容量。   
c.size() 返回容器中实际数据的个数。   
c1.swap(c2)   
swap(c1,c2) 将c1和c2元素互换。  
同上操作。   
vector<Elem> c  
vector<Elem> c1(c2)  
vector <Elem> c(n)  
vector <Elem> c(n, elem)  
vector <Elem> c(beg,end)  
c.~ vector <Elem>() 创建一个空的vector。  
复制一个vector。  
创建一个vector，含有n个数据，数据均已缺省构造产生。  
创建一个含有n个elem拷贝的vector。  
创建一个以[beg;end)区间的vector。  
销毁所有数据，释放内存。   
operator[] 返回容器中指定位置的一个引用。

创建一个vector  
vector容器提供了多种创建方法，下面介绍几种常用的。  
创建一个Widget类型的空的vector对象：  
vector<Widget> vWidgets;

创建一个包含500个Widget类型数据的vector：  
vector<Widget> vWidgets(500);

创建一个包含500个Widget类型数据的vector，并且都初始化为0：  
vector<Widget> vWidgets(500, Widget(0));

创建一个Widget的拷贝：  
vector<Widget> vWidgetsFromAnother(vWidgets);

向vector添加一个数据  
vector添加数据的缺省方法是push\_back()。push\_back()函数表示将数据添加到vector的尾部，并按需要来分配内存。例如：向vector<Widget>中添加10个数据，需要如下编写代码：  
for(int i= 0;i<10; i++)  
vWidgets.push\_back(Widget(i));

获取vector中制定位置的数据  
vector里面的数据是动态分配的，使用push\_back()的一系列分配空间常常决定于文件或一些数据源。如果想知道vector存放了多少数据，可以使用empty()。获取vector的大小，可以使用size()。例如，如果想获取一个vector v的大小，但不知道它是否为空，或者已经包含了数据，如果为空想设置为-1，你可以使用下面的代码实现：  
int nSize = v.empty() ? -1 : static\_cast<int>(v.size());

访问vector中的数据  
使用两种方法来访问vector。  
1、 vector::at()  
2、 vector::operator[]  
operator[]主要是为了与C语言进行兼容。它可以像C语言数组一样操作。但at()是我们的首选，因为at()进行了边界检查，如果访问超过了vector的范围，将抛出一个例外。由于operator[]容易造成一些错误，所有我们很少用它，下面进行验证一下：  
分析下面的代码：  
vector<int> v;  
v.reserve(10);  
  
for(int i=0; i<7; i++)  
v.push\_back(i);  
  
try  
{  
int iVal1 = v[7]; // not bounds checked - will not throw  
int iVal2 = v.at(7); // bounds checked - will throw if out of range  
}  
catch(const exception& e)  
{  
cout << e.what();  
}

删除vector中的数据  
vector能够非常容易地添加数据，也能很方便地取出数据，同样vector提供了erase()，pop\_back()，clear()来删除数据，当删除数据时，应该知道要删除尾部的数据，或者是删除所有数据，还是个别的数据。  
Remove\_if()算法 如果要使用remove\_if()，需要在头文件中包含如下代码：：  
#include <algorithm>

Remove\_if()有三个参数：  
1、 iterator \_First：指向第一个数据的迭代指针。  
2、 iterator \_Last：指向最后一个数据的迭代指针。  
3、 predicate \_Pred：一个可以对迭代操作的条件函数。  
条件函数 条件函数是一个按照用户定义的条件返回是或否的结果，是最基本的函数指针，或是一个函数对象。这个函数对象需要支持所有的函数调用操作，重载operator()()操作。remove\_if()是通过unary\_function继承下来的，允许传递数据作为条件。  
例如，假如想从一个vector<CString>中删除匹配的数据，如果字串中包含了一个值，从这个值开始，从这个值结束。首先应该建立一个数据结构来包含这些数据，类似代码如下：  
#include <functional>  
enum findmodes   
{  
FM\_INVALID = 0,  
FM\_IS,  
FM\_STARTSWITH,  
FM\_ENDSWITH,  
FM\_CONTAINS  
};  
typedefstruct tagFindStr  
{  
UINT iMode;  
CString szMatchStr;  
} FindStr;  
typedef FindStr\* LPFINDSTR;

然后处理条件判断：  
class FindMatchingString   
: public std::unary\_function<CString, bool>   
{  
  
public:   
FindMatchingString(const LPFINDSTR lpFS) : m\_lpFS(lpFS) {}   
  
bool operator()(CString& szStringToCompare) const  
{   
bool retVal = false;  
  
switch(m\_lpFS->iMode)   
{  
case FM\_IS:   
{   
retVal = (szStringToCompare == m\_lpFDD->szMatchStr);   
break;  
}  
case FM\_STARTSWITH:   
{   
retVal = (szStringToCompare.Left(m\_lpFDD->szMatchStr.GetLength())  
== m\_lpFDD->szWindowTitle);  
break;  
}   
case FM\_ENDSWITH:   
{  
retVal = (szStringToCompare.Right(m\_lpFDD->szMatchStr.GetLength())   
== m\_lpFDD->szMatchStr);  
break;   
}   
case FM\_CONTAINS:   
{  
retVal = (szStringToCompare.Find(m\_lpFDD->szMatchStr) != -1);  
break;   
}  
}  
  
return retVal;  
}  
  
private:   
LPFINDSTR m\_lpFS;  
};

通过这个操作你可以从vector中有效地删除数据：  
FindStr fs;  
fs.iMode = FM\_CONTAINS;  
fs.szMatchStr = szRemove;  
  
vs.erase(std::remove\_if(vs.begin(), vs.end(), FindMatchingString(&fs)), vs.end());

Remove(),remove\_if()等所有的移出操作都是建立在一个迭代范围上的，不能操作容器中的数据。所以在使用remove\_if()，实际上操作的时容器里数据的上面的。  
  
看到remove\_if()实际上是根据条件对迭代地址进行了修改，在数据的后面存在一些残余的数据，那些需要删除的数据。剩下的数据的位置可能不是原来的数据，但他们是不知道的。  
调用erase()来删除那些残余的数据。注意上面例子中通过erase()删除remove\_if()的结果和vs.enc()范围的数据。

本文来自CSDN博客，转载请标明出处：<http://blog.csdn.net/willoj/archive/2008/04/05/2252543.aspx>