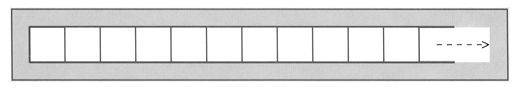
容器的 vector 和 deque，在操作方面比较相似，所以放在一起总结。除了共通操作外，它们还有自己的一些特点，具体如下：

**一. vector**

vector 模塑出来一个动态数组，在末端添加和删除元素时，性能相当好，在前端或中部插入或删除元素时，性能不怎么样。



**1. vector 的容量**

vector 性能优异的原因是，就是分配了比其所容纳元素所需的，更多的内存空间。有两个函数要特别注意一下：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7305932)

1. v.capacity; //返回 vector 实际能够容纳的元素数量

v.capacity; //返回 vector 实际能够容纳的元素数量

这个容量值是当前所能容纳的最大值，这个值是会变的，当超过当前最大容量时 vector 就会重新配置内部存储空间。

容量一但分配，就不会缩减。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7305932)

1. //仅分配空间，size 是不变的
2. v.reserve();
3. //下面两种方式是一样的。不过第一种方法只分配空间，而第二种方法还要调用构造函数
4. vector<**int**> v;
5. v.reserve(80);
6. vector<**int**> v(80); //如果构造函数很耗时，那么推荐使用上面的方法

//仅分配空间，size 是不变的

v.reserve();

//下面两种方式是一样的。不过第一种方法只分配空间，而第二种方法还要调用构造函数

vector<int> v;

v.reserve(80);

vector<int> v(80); //如果构造函数很耗时，那么推荐使用上面的方法

如果容量不足，则扩大；如果调用所给的参数比当前 vector 的容量小，不会引发任何反应；

**2. vector 的操作**

**元素存取**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7305932)

1. //返回索引 idx 所标示的元素。如果idx越界，抛出out\_of\_range。只有这个是进行范围检查的。
2. v.at(idx);
3. //返回索引 idx 所标示的元素。不进行范围检查
4. v[idx];
5. //返回第一个元素。不检查第一个元素是否存在
6. v.front();
7. //返回最后一个元素。不检查最后一个元素是否存在
8. v.back();

//返回索引 idx 所标示的元素。如果idx越界，抛出out\_of\_range。只有这个是进行范围检查的。

v.at(idx);

//返回索引 idx 所标示的元素。不进行范围检查

v[idx];

//返回第一个元素。不检查第一个元素是否存在

v.front();

//返回最后一个元素。不检查最后一个元素是否存在

v.back();

**插入与删除**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7305932)

1. //在尾部添加和删除一个elem，无返回
2. v.push\_back(elem);
3. v.pop\_back();
4. //移除pos位置上的元素，返回下一元素的位置
5. v.erase(pos);
6. //将元素数量改为num。
7. //如果 num < 当前 size，则后面的元素被置为空。
8. //如果 num > 当前 size，多出来的新元素都以default/elem构造完成
9. v.resize(num);
10. v.resize(num,elem)

//在尾部添加和删除一个elem，无返回

v.push\_back(elem);

v.pop\_back();

//移除pos位置上的元素，返回下一元素的位置

v.erase(pos);

//将元素数量改为num。

//如果 num < 当前 size，则后面的元素被置为空。

//如果 num > 当前 size，多出来的新元素都以default/elem构造完成

v.resize(num);

v.resize(num,elem)

**3. vector 当成数组**

大多数情况下，可以把 vector 当成数组来用。

但是，千万不要把迭代器当做第一元素的地址来传递，vector 迭代器是由实现版本定义的，也许并不是一个指针。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7305932)

1. printf("%s\n", v.begin()); //Error, might work, but not portable
2. printf("%s\n", &v[0]); //OK

printf("%s\n", v.begin()); //Error, might work, but not portable

printf("%s\n", &v[0]); //OK

**二. deque**

deque也采用动态数组来管理元素，提供随机存取，与vector非常相似。不同的是deque的动态数组头尾都开放，因此能在头尾两端进行快速安插和删除。

http://hi.csdn.net/attachment/201202/29/0_1330497519xIis.gif

操作上与 vector 基本一样，但是多了几个操作：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7305932)

1. //在头部添加和删除一个elem，无返回
2. v.push\_front(elem);
3. v.pop\_front();

//在头部添加和删除一个elem，无返回

v.push\_front(elem);

v.pop\_front();

**注意：**

deque 不提供容量操作 capacity() 和 reserve()。

**三. 迭代器无效**

**1. vector 插入或删除操作会使“作用点”之后的各元素的pointers, references和iterators失效，如果插入操作引发内存重新分配，那么容器上所有的pointers, references和iterators都将失效**。

**2. deque 插入或删除都可能引起内存重新分配，所以任何插入或删除动作都会使所有指向 deque元素的 pointers, references 和 iterators 失效。**

所以在插入或删除操作之后，要重新对 pointers, references和iterators 进行赋值的值。

**四. 代码举例**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7305932)

1. #include <iostream>
2. #include <string>
3. #include <vector>
4. using namespace std;
5. **int** main(**int** argc, **char**\* argv[])
6. {
7. //create empty vector for strings
8. vector<string> sentence;
9. //开辟一个能容纳5个元素的空间，size 可没变
10. sentence.reserve(5);
11. //append some elements
12. sentence.push\_back("Hello,");
13. sentence.push\_back("how");
14. sentence.push\_back("are");
15. sentence.push\_back("you");
16. sentence.push\_back("?");
17. //打印
18. copy (sentence.begin(), sentence.end(),
19. ostream\_iterator<string>(cout," "));
20. cout << endl;
21. //print ''technical data''
22. cout << " max\_size(): " << sentence.max\_size() << endl;
23. cout << " size(): " << sentence.size() << endl;
24. cout << " capacity(): " << sentence.capacity() << endl;
25. //swap second and fourth element
26. swap (sentence[1], sentence [3]);
27. //使用迭代器
28. vector<string>::iterator it1 = sentence.begin();
29. vector<string>::iterator it2 = sentence.end();
30. cout<<\*it1<<endl;
31. cout<<\*(it2-1)<<endl;
32. //在 "?" 前插入元素 "always"
33. sentence.insert (find(sentence.begin(),sentence.end(),"?"),
34. "always");
35. //引发迭代器无效，要崩溃！！！！！！！！！！！！！！！！！！！！！！！！！！！
36. cout<<\*it1<<endl;
37. cout<<\*(it2-1)<<endl;
38. //assign "!" to the last element
39. sentence.back() = "!";
40. //print elements separated with spaces
41. copy (sentence.begin(), sentence.end(),
42. ostream\_iterator<string>(cout," "));
43. cout << endl;
44. //print "technical data" again
45. cout << " max\_size(): " << sentence.max\_size() << endl;
46. cout << " size(): " << sentence.size() << endl;
47. cout << " capacity(): " << sentence.capacity() << endl; //自动变了！！
48. return 0;
49. }

#include <iostream>

#include <string>

#include <vector>

using namespace std;

int main(int argc, char\* argv[])

{

//create empty vector for strings

vector<string> sentence;

//开辟一个能容纳5个元素的空间，size 可没变

sentence.reserve(5);

//append some elements

sentence.push\_back("Hello,");

sentence.push\_back("how");

sentence.push\_back("are");

sentence.push\_back("you");

sentence.push\_back("?");

//打印

copy (sentence.begin(), sentence.end(),

ostream\_iterator<string>(cout," "));

cout << endl;

//print ''technical data''

cout << " max\_size(): " << sentence.max\_size() << endl;

cout << " size(): " << sentence.size() << endl;

cout << " capacity(): " << sentence.capacity() << endl;

//swap second and fourth element

swap (sentence[1], sentence [3]);

//使用迭代器

vector<string>::iterator it1 = sentence.begin();

vector<string>::iterator it2 = sentence.end();

cout<<\*it1<<endl;

cout<<\*(it2-1)<<endl;

//在 "?" 前插入元素 "always"

sentence.insert (find(sentence.begin(),sentence.end(),"?"),

"always");

//引发迭代器无效，要崩溃！！！！！！！！！！！！！！！！！！！！！！！！！！！

cout<<\*it1<<endl;

cout<<\*(it2-1)<<endl;

//assign "!" to the last element

sentence.back() = "!";

//print elements separated with spaces

copy (sentence.begin(), sentence.end(),

ostream\_iterator<string>(cout," "));

cout << endl;

//print "technical data" again

cout << " max\_size(): " << sentence.max\_size() << endl;

cout << " size(): " << sentence.size() << endl;

cout << " capacity(): " << sentence.capacity() << endl; //自动变了！！

return 0;

}