**STL源码剖析---迭代器失效小结**

原创 2012年07月10日 21:32:20

* 标签：
* [vector](http://so.csdn.net/so/search/s.do?q=vector&t=blog) /
* [存储](http://so.csdn.net/so/search/s.do?q=存储&t=blog) /
* [iterator](http://so.csdn.net/so/search/s.do?q=iterator&t=blog) /
* [insert](http://so.csdn.net/so/search/s.do?q=insert&t=blog) /
* [list](http://so.csdn.net/so/search/s.do?q=list&t=blog) /
* [测试](http://so.csdn.net/so/search/s.do?q=测试&t=blog)

     迭代器（iterator）是一个可以对其执行类似指针的操作（如：解除引用（operator\*()）和递增（operator++()））的对象，我们可以将它理解成为一个指针。但它又不是我们所谓普通的指针，我们可以称之为广义指针，你可以通过sizeof（vector::iterator）来查看，所占内存并不是4个字节。  
     首先对于vector而言，添加和删除操作可能使容器的部分或者全部迭代器失效。那为什么迭代器会失效呢？vector元素在内存中是顺序存储，试想：如果当前容器中已经存在了10个元素，现在又要添加一个元素到容器中，但是内存中紧跟在这10个元素后面没有一个空闲空间，而vector的元素必须顺序存储一边索引访问，所以我们不能在内存中随便找个地方存储这个元素。于是vector必须重新分配存储空间，用来存放原来的元素以及新添加的元素：存放在旧存储空间的元素被复制到新的存储空间里，接着插入新的元素，最后撤销旧的存储空间。这种情况发生，一定会导致vector容器的所有迭代器都失效。  
     我们看到实现上述所说的分配和撤销内存空间的方式以实现vector的自增长性，效率是极其低下的。为了使vector容器实现快速的内存分配，实际分配的容器会比当前所需的空间多一些，vector容器预留了这些额外的存储区，用来存放新添加的元素，而不需要每次都重新分配新的存储空间。你可以从vector里实现capacity和reserve成员可以看出这种机制。  
    capacity和size的区别：size是容器当前拥有的元素个数，而capacity则指容器在必须分配新存储空间之前可以存储的元素总数。  
    vector迭代器的几种失效的情况：   
1、当插入（push\_back）一个元素后，end操作返回的迭代器肯定失效。  
2、当插入(push\_back)一个元素后，capacity返回值与没有插入元素之前相比有改变，则需要重新加载整个容器，此时first和end操作返回的迭代器都会失效。  
3、当进行删除操作（erase，pop\_back）后，指向删除点的迭代器全部失效；指向删除点后面的元素的迭代器也将全部失效。  
    deque迭代器的失效情况： 在C++Primer一书中是这样限定的：   
1、在deque容器首部或者尾部插入元素不会使得任何迭代器失效。   
2、在其首部或尾部删除元素则只会使指向被删除元素的迭代器失效。  
3、在deque容器的任何其他位置的插入和删除操作将使指向该容器元素的所有迭代器失效。但是：我在vs2005测试发现第一条都不满足，不知为何？等以后深入STL以后慢慢的领会吧！  
     只有list的迭代器好像很少情况下会失效。也许就只是在删除的时候，指向被删除节点的迭代器会失效吧，其他的还没有发现。  
先看两条规制：  
1、对于节点式容器(map, list, set)元素的删除，插入操作会导致指向该元素的迭代器失效，其他元素迭代器不受影响。  
2、对于顺序式容器(vector)元素的删除、插入操作会导致指向该元素以及后面的元素的迭代器失效。  
众所周之当使用一个容器的insert或者erase函数通过迭代器插入或删除元素"可能"会导致迭代器失效，因此很多建议都是让我们获取insert或者erase返回的迭代器，以便用重新获取新的有效的迭代器进行正确的操作：  
代码如下：

**[cpp]** [view plain](http://blog.csdn.net/hackbuteer1/article/details/7734382) [copy](http://blog.csdn.net/hackbuteer1/article/details/7734382)

1. iter = vec.insert(iter);
2. iter = vec.erase(iter);