STL更易型算法：算法会变动、改动区间内的元素内容。

注意：目标区间不可以是个associative或unordered容器，因为它们的元素被视为常量。例如要修改map容器的某个key对应的value元素，需要先删除这个key-value，然后再重新添加新的key-value。

1. 复制元素：copy，线性复杂度

Iterator2 copy(Iterator1 sourceBeg, Iterator1 sourceEnd, Iterator2 destBeg)

将源区间[sourceBeg, sourceEnd)复制到destBeg开始的目标区间内。

注意：destBeg不能在[sourceBeg,sourceEnd)内。但是目标区间有可能与源区间重叠。

Iterator2 copy\_if(Iterator1 sourceBeg, Iterator1 sourceEnd, Iterator2 destBeg, UnaryPredicate op)

将源区间[sourceBeg, sourceEnd)满足op(elem)为true的元素复制到destBeg开始的目标区间内。

注意：源区间与目标区间不能重叠。

Iterator2 copy\_n(Iterator sourceBeg, Size num, Iterator2 destBeg)

将以sourceBeg开始的num个元素复制到destBeg开始的目标区间内

BidirectionIterator2

copy\_backward(BidirectionalIterator1 sourceBeg,

BidirectionalIterator1sourceEnd,

BidirectionalIterator2 destEnd)

将源区间[sourceBeg, sourceEnd)反向复制到以destEnd结尾的模板区间内

注意：destEnd不能在(sourceBeg, sourceEnd]内。

以上4个函数都返回目标区间内最后一个被复制元素的下一位置，也就第一个未被覆盖的元素位置。

例：程序stl\_test71

// copy

*vector*<*string*> coll1 = {"Hello", "this", "is", "an", "example"};

*list*<*string*> coll2;

// use back\_inserter into insert instead of overwrite

*copy*(coll1.*cbegin*(), coll1.*cend*(), *back\_inserter*(coll2));

// copy elements to cout using an ostream iterator

*copy*(coll2.*cbegin*(), coll2.*cend*(), *ostream\_iterator*<*string*>(*cout*, " "));

*cout* << *endl*;

*cout* << "reverse copy: " << *endl*;

// overwriting

*copy*(coll1.*crbegin*(), coll1.*crend*(), coll2.*begin*());

*copy*(coll2.*cbegin*(), coll2.*cend*(), *ostream\_iterator*<*string*>(*cout*, " "));

输出为：

Hello this is an example

reverse copy:

example an is this Hello

例：程序stl\_test71

*vector*<char> source(10, '.');

for (int c = 'a'; c <= 'f'; ++c)

{

source.*push\_back*(c);

}

source.*insert*(source.*end*(), 10, '.');

PrintElements(source, "source: ");

*vector*<char> c1(source.*cbegin*(), source.*cend*());

// 源区间与目标区间发生了重叠

// copy all letters three elements in front of the 'a'

*copy*(c1.*cbegin*() + 10, c1.*cbegin*() + 16, // source range

c1.*begin*() + 7); // destination range

PrintElements(c1, "c1: ");

*vector*<char> c2(source.*cbegin*(), source.*cend*());

// 源区间与目标区间发生重叠

// copy all letters three elements behind the 'f'

*copy\_backward*(c2.*cbegin*() + 10, c2.*cbegin*() + 16, c2.*begin*() + 19);

PrintElements(c2, "c2: ");

输出为：

source: . . . . . . . . . . a b c d e f . . . . . . . . . .

c1: . . . . . . . a b c d e f d e f . . . . . . . . . .

c2: . . . . . . . . . . a b c a b c d e f . . . . . . .

1. 搬移元素，move，线性复杂度

Iterator2 move(Iterator1 sourceBeg, Iterator1 sourceEnd, Iterator2 destBeg)

BidirectionalInterator2

move\_backward(BidirectionalIterator1 sourceBeg,

BidirectionalIterator1 sourceEnd,

BirdirectionalIterator2 destEnd)

将源区间[sourceBeg, sourEnd)中所有元素搬移至以destBeg开始或以destEnd结束的目标区间内。

针对每一个元素调用：

\*destElem = std::move(\*sourceElem)

注意：如果元素类型提供了move语义，源端元素从此不再明确，就不应该再被使用，除非重新赋值或初始化。如果元素类型未提供move语义，元素会被copy，类似copy()或copy\_backward().

返回目标区间内最后一个被复制元素的下一位置，也就是第一个未被覆盖的元素位置。

move()的destBeg不能位于[sourceBeg, sourceEnd)区间内，move\_backward()的destEnd不能处于(sourceBeg, sourceEnd]区间内。

例：程序stl\_test71

*vector*<*string*> coll1 = {"Hello", "this", "is", "an", "example"};

// move

*list*<*string*> coll3(coll1.*size*());

*move*(coll1.*cbegin*(), coll1.*cend*(), coll3.*begin*());

// move并未将coll1的元素搬移走

*cout* << coll1.*size*() << *endl*;

*cout* << \*coll1.*cbegin*() << *endl*;

*list*<*string*> list1 = {"Hello", "World"};

// 使用搬移迭代器会将list1中的元素搬移到list2中

// 但list1仍然有空间

*list*<*string*> list2(*make\_move\_iterator*(list1.*begin*()), *make\_move\_iterator*(list1.*end*()));

*cout* << list1.*size*() << *endl*;

*cout* << \*list1.*begin*() << *endl*; // 没有值

输出为：

5

Hello

2

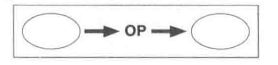
1. 转换和结合元素，transform，线性复杂度

Iterator2 transform(Iterator1 sourceBeg, Iterator1 soureEnd,

Iterator2 destBeg,

UnaryFunc op)

将源区间[sourceBeg, sourceEnd)中的每一个元素调用op(elem)并将结果写到以destBeg起始的目标区间内。



返回目标区间内“最后一个被转换元素”的下一位置，也就是第一个未被覆盖的元素的位置。

sourceBeg可以和destBeg完全相同。

例：程序stl\_test72

*vector*<int> coll1;

*list*<int> coll2;

InsertElements(coll1, 1, 9);

PrintElements(coll1, "coll1: ");

// negate all elements in coll1

*transform*(coll1.*cbegin*(), coll1.*cend*(), // source range

coll1.*begin*(), // destination range

*negate*<int>()); // operation

PrintElements(coll1, "negated: ");

*transform*(coll1.*cbegin*(), coll1.*cend*(), // source range

*back\_inserter*(coll2), // destination range

*bind*(*multiplies*<int>(), *\_1*, 10)); // operation

PrintElements(coll2, "coll2: ");

输出为：

coll1: 1 2 3 4 5 6 7 8 9

negated: -1 -2 -3 -4 -5 -6 -7 -8 -9

coll2: -10 -20 -30 -40 -50 -60 -70 -80 -90

Iterator3

transform(Iterator1 sourceBeg1, Iterator1 sourceEnd1,

Iterator2 sourceBeg2,

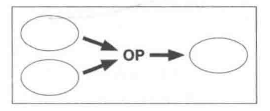
Iterator3 destBeg,

BinaryFunc op)

针对第一源区间[sourceBeg1, sourceEnd1)以及“从sourceBeg2开始的第二源区间”的对应元素，调用：

op(sourceElem1, sourceElem2)

并将结果写入以destBeg起始的目标区间内。



sourceBeg1、sourceBeg2和destBeg可以相同。

例：程序stl\_test72

*vector*<int> coll3;

*list*<int> coll4;

InsertElements(coll3, 1, 9);

PrintElements(coll3, "coll3: ");

// square each element

*transform*(coll3.*cbegin*(), coll3.*cend*(), // first source range

coll3.*cbegin*(), // second source range

coll3.*begin*(), // destination range

*multiplies*<int>()); // operation

PrintElements(coll3, "squared: ");

// add each element traversed forward

//with each element traversed backward

*transform*(coll3.*cbegin*(), coll3.*cend*(),

coll3.*crbegin*(),

*back\_inserter*(coll4),

*plus*<int>());

PrintElements(coll4, "coll4: ");

输出为：

coll3: 1 2 3 4 5 6 7 8 9

squared: 1 4 9 16 25 36 49 64 81

coll4: 82 68 58 52 50 52 58 68 82