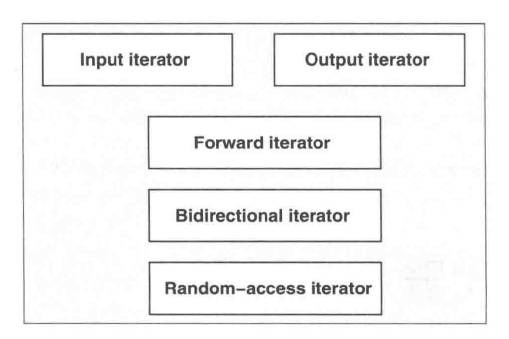
迭代器Iterator：能够迭代某序列（sequence）内所有元素的对象。



|  |  |  |
| --- | --- | --- |
| 种类 | 能力 | 提供者 |
| Output iterator | 向前写入 | ostream, inserter |
| Input iterator | 向前读取一次 | istream |
| Forward iterator | 向前读取 | forward list, unordered container |
| Bidirection iterator | 向前和向后读取 | list, set, multiset, map, multimap |
| Random-access iterator | 以随机访问方式读取 | array, vector, deque, string C-style Array |

例：程序stl\_test55

*vector*<int> coll;

coll.*push\_back*(9);

coll.*push\_back*(3);

coll.*push\_back*(0);

coll.*push\_back*(8);

coll.*push\_back*(5);

// 将容器coll从第2个开始排序

if (coll.*size*() > 1)

{

// 这种写法在visual studio可以编译通过

// 但不具备可移植性，因为有可能在其他平台上

// 编译不通过

// sort(++coll.begin(), coll.end());

}

if (coll.*size*() > 1)

{

// 可移植的写法

*sort*(*next*(coll.*begin*()), coll.*end*());

}

for (const auto& elem : coll)

{

*cout* << elem << " ";

}

*cout* << *endl*;

迭代器的部分辅助函数：提供“原本只有random access迭代器才有”的能力。

1. advance(Iterator& pos, Dist n)：迭代器前进或后退（只对双向迭代器bidirectional和random-access迭代器有效）n个元素。

注意：advance()并不检查是否超过序列的end()，因此有可能会产生不明确的行为。

对于提供random access的容器，advance()调用operator += n，具有常量复杂度；对于其它任何类型的容器，调用++pos或--pos操作n次，具有线性复杂度。

例：程序stl\_test55

// advance

*list*<int> coll\_list;

for (int i = 1; i <= 9; ++i)

{

coll\_list.*push\_back*(i);

}

*list*<int>::*iterator* pos = coll\_list.*begin*();

*cout* << "step three elements forward:" << *endl*;

*advance*(pos, 3);

*cout* << \*pos << *endl*;

*cout* << "step one element backward: " << *endl*;

*advance*(pos, -1);

*cout* << \*pos << *endl*;

// 运行时会报内存错误

// pos越界了

// advance(pos, 9);

输出为：

step three elements forward:

4

step one element backward:

3

1. next()：向前/向后移动迭代器的位置，相对于advance()，它会返回移动后的迭代器的位置。

Iterator next(Iterator pos)

Iterator next(Iteratorpos, Dist n)

对于bidirectional和random-access迭代器，n可为负值，后退移动

注意：next()并不检查是否会跨越序列的end()。

1. prev()：前进/后退Bidirectional迭代器

BidirectionalIterator prev(BidirectionalIterator pos)

BidirectionalIterator prev(BidirectionalIterator pos, Dist n)

n为负值，向前移动

注意：prev()并不检查是否会跨越序列的begin()。

1. distance()：处理两个迭代器之间的距离

Dist distance(Iterator pos1, Iterator pos2)

4.1) 两个迭代器必须指向同一个容器

4.2) 如果不是random-access迭代器，则从pos1开始前进必须能够到达pos2，即pos2的位置必须与pos1相同或在其后。

例：程序stl\_test55

// distance

*list*<int> coll1;

for (int i = -3; i <= 9; ++i)

{

coll1.*push\_back*(i);

}

*list*<int>::*iterator* pos1 = *find*(coll1.*begin*(), coll1.*end*(), 5);

*list*<int>::*iterator* pos2 = *find*(coll1.*begin*(), coll1.*end*(), 8);

// 如果pos2在pos1之前会发生内存错误

//list<int>::iterator pos2 = find(coll1.begin(), coll1.end(), 4);

if (pos1 != coll1.*end*() && pos2 != coll1.*end*())

{

*iterator\_traits*<*list*<int>::*iterator*>::*difference\_type* n = *distance*(pos1, pos2);

*cout* << "difference between 5 and 8 is " << n << *endl*;

}

输出为：

difference between 5 and 8 is 3

1. iter\_swap()：

void iter\_swap(Iterator pos1, Iterator pos2)

注意：迭代器的类型不必相同，但其所指的两个值必须可以相互赋值（assignable）

例：程序stl\_test55

// iter\_swap

PrintElements(coll1);

*cout* << "swap begin and next(begin): " << *endl*;

*iter\_swap*(coll1.*begin*(), *next*(coll1.*begin*()));

PrintElements(coll1);

输出为：

-3 -2 -1 0 1 2 3 4 5 6 7 8 9

swap begin and next(begin):

-2 -3 -1 0 1 2 3 4 5 6 7 8 9