反向适配器：

例：程序stl\_test56

*vector*<int> coll = {1, 2, 3, 4, 5, 6, 7, 8, 9};

// 元素5所在的位置

*vector*<int>::*const\_iterator* pos = *find*(coll.*cbegin*(), coll.*cend*(), 5);

*cout* << "pos: " << \*pos << *endl*;

// 反向迭代器

*vector*<int>::*const\_reverse\_iterator* rpos(pos);

*cout* << "rpos: " << \*rpos << *endl*;

*cout* << \*coll.*rbegin*() << *endl*; // 9

// 内存错误

//cout << \*coll.rend() << endl;

输出为：

pos: 5

rpos: 4

9

从输出可以看出：同一个位置的正向迭代器和反向迭代器的值不同。

正向迭代器的物理位置和逻辑位置是对应的，逻辑位置即begin()和end()。逻辑位置begin()指向物理位置第1个元素，end()指向物理位置最末元素的下一个位置。

[rbegin, rend)和[begin,end)所包含的元素的物理地址正好完全颠倒。

反向迭代器[rbegin(), rend())，通过base()转换为正向迭代器，需要[rend().base(), rbegin().base())，代表正向迭代整个容器区间，才符合容器左闭右开的原则。所以反向迭代器的base()获取的正向迭代器指向的物理元素不能和反向的一致，否则rend().base()就会指向一个无效的地址。

例：程序effective\_stl\_test10

*vector*<int> v{ 1, 2, 3, 4, 5, 6, 7, 8, 9 };

*cout* << "正向迭代器指向元素的地址" << *endl*;

for (auto iter = v.*begin*(); iter != v.*end*(); ++iter)

{

*cout* << *hex* << &(\*(iter)) <<*endl*;

}

*cout* << *endl*;

*cout* << "反向迭代器指向元素的地址" << *endl*;

for (auto r\_iter = v.*rbegin*(); r\_iter != v.*rend*(); ++r\_iter)

{

*cout* << *hex* << &(\*(r\_iter)) << *endl*;

}

*cout* << *endl*;

*cout* << "通过反向迭代器的base()获取的迭代器指向元素的地址" << *endl*;

for (auto iter = v.*rend*().*base*(); iter != v.*rbegin*().*base*(); ++iter)

{

*cout* << *hex* << &(\*(iter)) << *endl*;

}

输出为：

正向迭代器指向元素的地址

012DE0D0

012DE0D4

012DE0D8

012DE0DC

012DE0E0

012DE0E4

012DE0E8

012DE0EC

012DE0F0

反向迭代器指向元素的地址

012DE0F0

012DE0EC

012DE0E8

012DE0E4

012DE0E0

012DE0DC

012DE0D8

012DE0D4

012DE0D0

通过反向迭代器的base()获取的迭代器指向元素的地址

012DE0D0

012DE0D4

012DE0D8

012DE0DC

012DE0E0

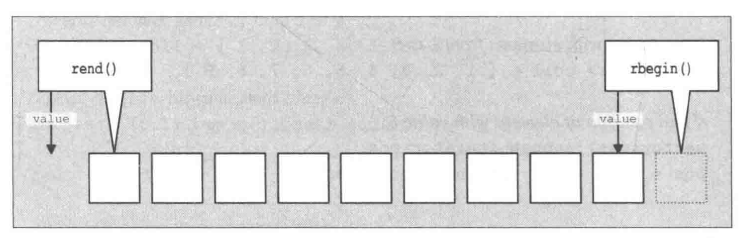
012DE0E4

012DE0E8

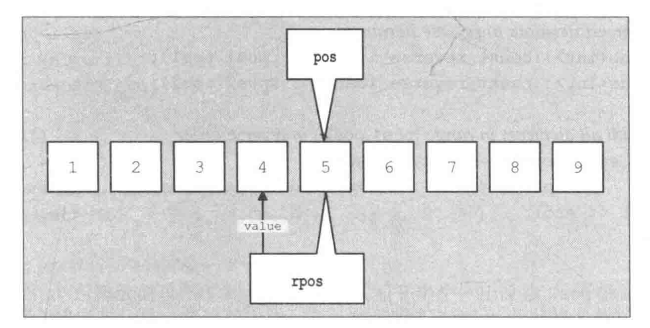
012DE0EC

012DE0F0

Reverse迭代器所定义的逻辑区间为[rbegin, rend)，即rbegin()和rend()。但实际的物理区间不包括起点，却包含了终点(begin, end]，其逻辑上指向的元素位置就和物理上值的元素位置不同。



所以\*rbegin有值，而\*rend会报内存错误



正向迭代器其物理位置与逻辑位置一致，反向迭代器其物理位置与逻辑位置不一致。