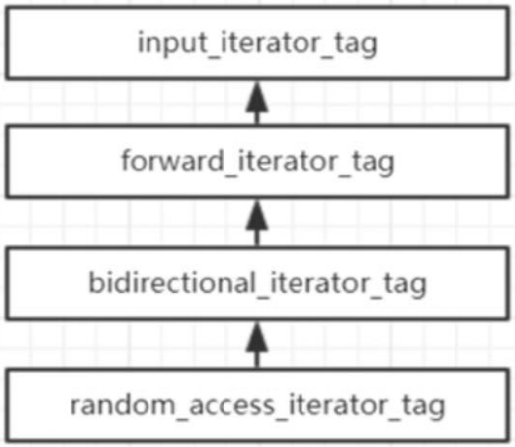

output_iterator_tag

迭代器种类（Category）	能力	提供者
Output 迭代器	向前写入	Ostream、inserter
Input 迭代器	向前读取一次	Istream
Forward 迭代器	向前读取	Forward list、unordered containers
Bidirectional 迭代器	向前和向后读取	List、set、multiset、map、multimap
Random-access 迭代器	以随机访问方式读取	Array、vector、deque、string、C-style array

```
//二：迭代器的分类：迭代器是分种类的；
//分类的依据：依据是 迭代器的移动特性以及在这个迭代器上能够做的操作；

//迭代器，行为如指针，到处跳，表示一个位置，我们一般分类是依据他的跳跃能力，每个分类是一个对应的struct定义：
//a)输出型迭代器：(Output iterator)
//struct output_iterator_tag;
//b)输入型迭代器：（Input iterator）
//struct input_iterator_tag;
//c)前向迭代器（Forward iterator)
//struct forward_iterator_tag;
//d)双向迭代器（Bidirectional iterator）
//struct bidirectional_iterator_tag
//e)随机访问迭代器（Random-access iterator)
//struct random_access_iterator_tag

//这些分类（结构）都有继承关系的
```

//大多数容器里边都有一个::iterator迭代器类型；并不是所有容器里都有迭代器；比如stack,queue这种容器就不提供迭代器；

输出型迭代器（Output iterator）

表达式	效果
*iter = val	将val写至迭代器所指位置
++iter	向前步进（step forward），返回新位置
iter++	向前步进（step forward），返回旧位置
TYPE(iter)	复制迭代器（copy 构造函数）

输入型迭代器（Input iterator）

表达式	效果
*iter	读取实际元素
iter ->member	读取实际元素的成员（如果有的话）
++iter	向前步进（返回新位置）
iter++	向前步进（返回旧位置）
iter1 == iter2	判断两个迭代器是否相等
iter1 != iter2	判断两个迭代器是否不等
TYPE(iter)	复制迭代器（copy 构造函数）

前向迭代器（Forward iterator）

表达式	效果
*iter	访问实际元素
iter->member	访问实际元素的成员
++iter	向前步进（返回新位置）
iter++	向前步进（返回旧位置）
iter1 == iter2	判断两个迭代器是否相等
iter1 != iter2	判断两个迭代器是否不等
TYPE()	创建迭代器（default 构造函数）
TYPE(iter)	复制迭代器（copy 构造函数）
iter1 = iter2	对迭代器赋值（assign）

双向迭代器（Bidirectional iterator）

表达式	效果
--iter	步退（返回新位置）
iter--	步退（返回旧位置）

随机访问迭代器（Random-access iterator）

表达式	效果
iter[n]	访问索引位置为n的元素
iter+=n	前进n个元素（如果n是负数，则改为回退）
iter-=n	回退n个元素（如果n是负数，则改为前进）
iter+n	返回iter之后的第n个元素
n+iter	返回iter之后的第n个元素
iter-n	返回iter之前的第n个元素
iter1-iter2	返回iter1和iter2之间的距离
iter1<iter2	判断iter1是否在iter2之前
iter1>iter2	判断iter1是否在iter2之后
iter1<=iter2	判断iter1是否不在iter2之后
iter1>=iter2	判断iter1是否不在iter2之前

```
//完善迭代器能力：
//a)输出型迭代器 struct output_iterator_tag; 一步一步能往前走，并且能够通过这个种类的迭代器来改写容器中的数据；
//b)输入型迭代器：struct input_iterator_tag; 一次一个以向前的方向来读取元素，按照这个顺序一个一个返回元素值；
//c)前向迭代器：struct forward_iterator_tag;因为继承自从Input迭代器，因此它能以向前的方向来读取元素，并且读取时提供额外保证；
//d)双向迭代器：struct bidirectional_iterator_tag: 在前向迭代器基础之上增加了往回（反向）迭代，也就是迭代位置可以回退，新增加
//e)随机访问迭代器：struct random_access_iterator_tag: 在双向迭代器基础上又增加了所谓的随机访问能力：也就是增减某个偏移里，能
```