C 箴言：为类型信息使用特征类 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/133/2021_2022_C___E7_AE_B4_E8_A8_80_EF_c97_133667.htm STL 主要是由 containers（容器），iterators（迭代器）和 algorithms（算法）的 templates（模板）构成的，但是也有几个 utility templates（实用模板）。其中一个被称为 advance。advance 将一个指定的 iterator（迭代器）移动一个指定的距离： template // move iter d units void advance(IterT&amp. iter, DistT d). // forward. if d // move iter backward 在概念上，advance 仅仅是在做 iter = d，但是 advance 不能这样实现，因为只有 random access iterators（随机访问迭代器）支持 = operation。不够强力的 iterator（迭代器）类型不得不通过反复利用 或 -- d 次来实现 advance。 你不记得 STL iterator categories（迭代器种类）了吗？没问题，我们这就做一个简单回顾。对应于它们所支持的操作，共有五种 iterators（迭代器）。input iterators（输入迭代器）只能向前移动，每次只能移动一步，只能读它们指向的东西，而且只能读一次。它们以一个输入文件中的 read pointer（读指针）为原型；C 库中的 istream_iterators 就是这一种类的典型代表。output iterators（输出迭代器）与此类似，只不过用于输出：它们只能向前移动，每次只能移动一步，只能写它们指向的东西，而且只能写一次。它们以一个输出文件中的 write pointer（写指针）为原型；ostream_iterators 是这一种类的典型代表。这是两个最不强力的 iterator categories（迭代器种类）。因为 input（输入）和 output iterators（输出迭代器）只能向前移动而且只能读或者写它们指向的地方最多一次，

它们只适合 one-pass 运算。 一个更强力一些的 iterator category（迭代器种类）是 forward iterators（前向迭代器）。这种 iterators（迭代器）能做 input（输入）和 output iterators（输出迭代器）可以做到的每一件事情，再加上它们可以读或者写它们指向的东西一次以上。这就使得它们可用于 multi-pass 运算。STL 没有提供 singly linked list（单向链表），但某些库提供了（通常被称为 slist），而这种 containers（容器）的 iterators（迭代器）就是 forward iterators（前向迭代器）。TR1 的 hashed containers（哈希容器）的 iterators（迭代器）也可以属于 forward category（前向迭代器）。 bidirectional iterators（双向迭代器）为 forward iterators（前向迭代器）加上了和向前一样的向后移动的能力。STL 的 list 的 iterators（迭代器）属于这一种类，set，multiset，map 和 multimap 的 iterators（迭代器）也一样。 最强力的 iterator category（迭代器种类）是 random access iterators（随机访问迭代器）。这种 iterators（迭代器）为 bidirectional iterators（双向迭代器）加上了 "iterator arithmetic"（“ 迭代器运算 ”）的能力，也就是说，在常量时间里向前或者向后跳转一个任意的距离。这样的运算类似于指针运算，这并不会让人感到惊讶，因为 random access iterators（随机访问迭代器）就是以 built-in pointers（内建指针）为原型的，而 built-in pointers（内建指针）可以和 random access iterators（随机访问迭代器）有同样的行为。vector，deque 和 string 的 iterators（迭代器）是 random access iterators（随机访问迭代器）。 对于五种 iterator categories（迭代器种类）中的每一种，C 都有一个用于识别它的 "tag struct"（“ 标签结构体 ”）在标准库中： struct

input_iterator_tag {}. struct output_iterator_tag {}. struct forward_iterator_tag: public input_iterator_tag {}. struct bidirectional_iterator_tag: public forward_iterator_tag {}. struct random_access_iterator_tag: public bidirectional_iterator_tag {}. 这些结构体之间的 inheritance relationships（继承关系）是正当的 is-a 关系：所有的 forward iterators（前向迭代器）也是 input iterators（输入迭代器），等等，这都是成立的。我们不久就会看到这个 inheritance（继承）的功用。 