大多数算法都定义在头文件algorithm中。但标准库在头文件numeric中定义了一组数值泛型算法，用来处理不同的容器

迭代器使得算法不依赖于容器，但算法依赖于元素类型的操作。算法不执行容器的操作，它们只运行在迭代器之上，执行迭代器操作。

这些算法不直接操作容器，而是遍历两个迭代器指定的一个元素范围来进行操作。算法不会改变容器的大小。

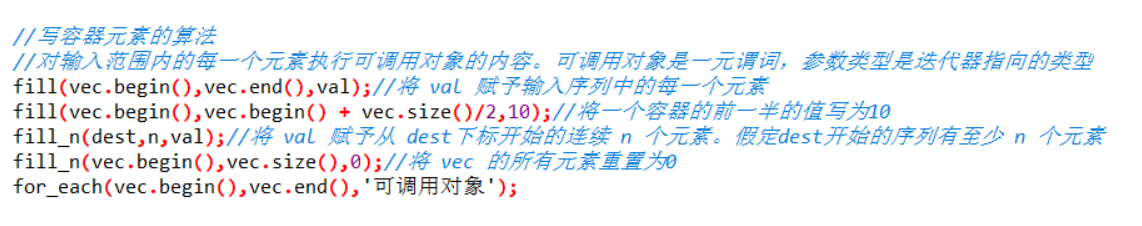
只读算法：只读取其输入范围内的元素，而从不改变元素。比如find方法包含的算法。对于只读算法，最好使用 cbegin() 和 cend()。

1.第一个算法是求和。指定顺序容器中的两个迭代器，前两个迭代器代表求和的范围（同样左开右闭区间），这个和再加上val求和，返回最后的结果。这个算法隐含了条件“容器内元素相加是合法的且结果能和val相加”。

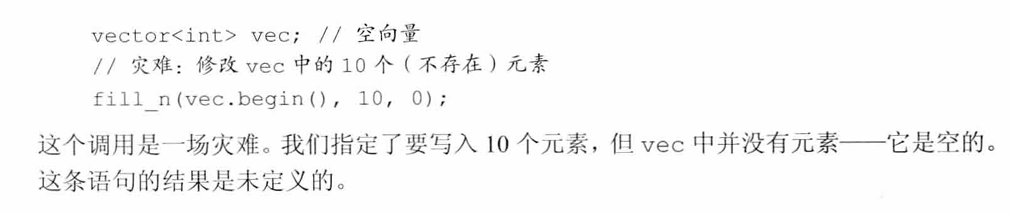
2.第二个算法是比较两个序列中的元素是否全部相等。前两个迭代器代表第一个序列中元素范围，第三个迭代器代表第二个序列的首元素。这个方法接收的两个容器的类型可以不相同，例如第一个容器是vector<string>第二个容器list<const char\*>。隐含条件是第二个容器



写容器元素的算法：算法会将新值赋给序列中的元素。使用这类算法时要注意保证原序列大小大于等于新写入的元素的数目



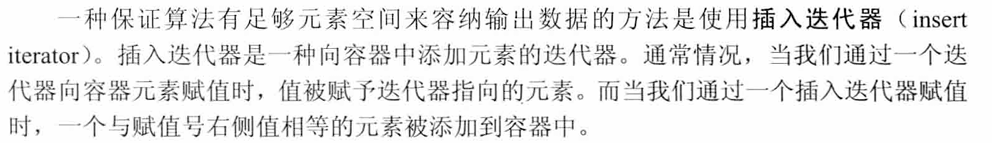
ps：算法不直接操作容器。但算法的形参为容器类型，方法在调用时需要接收容器类型的实例，但是不会检查容器对象是否为空，对于方法来说，只要接收到了正确类型的参数即可，哪怕这个参数为空容器（这样很容易发生未定义行为），也不会在编译时检查出错误。就像下面书上的例子一样：



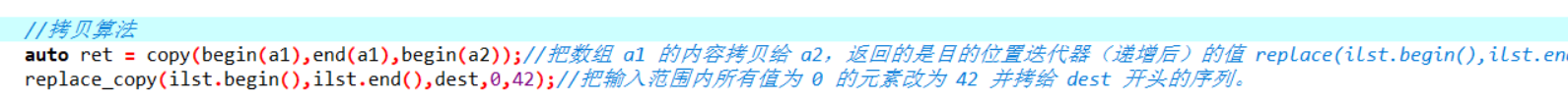
为了减少甚至消除这些未定义行为，要确保传入的容器有足够元素空间来容纳输出数据，为了达成这一目的可以使用插入迭代器。

插入迭代器的具体作用：允许在容器中插入元素而不是覆盖元素。它们是标准库算法与容器之间的桥梁，特别适用于插入操作而非替换操作。

插入迭代器实现的思路：



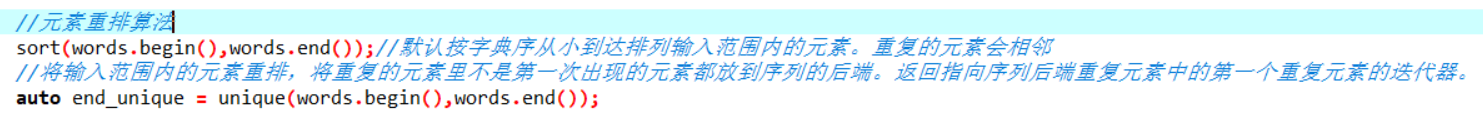
拷贝算法：将一个范围内的元素复制到另一个范围的算法。



上面例子中的copy方法中，前两个迭代器表示输入范围，第三个迭代器表示拷贝的序列的起始地址。返回值是复制后迭代器被递增后的位置，即a2的尾后元素位置。

replace\_copy与replace的不同点在于前者在修改后不会改变ilst序列，仅仅dest序列中的0被改为42，而replace直接在原来的顺序容器中修改。

重排容器元素的算法：



1.sort方法：利用元素类型的 < 运算符实现排序。即按照字典序从小到大排序。

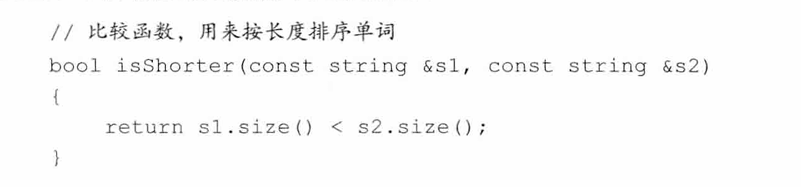
2.unique方法：用于删除容器中相邻的重复元素，使得每个元素仅出现一次。unique方法不会改变容器的大小，只是将重复的元素移动到容器的末尾，并返回一个指向新范围末尾的迭代器。要真正删除重复的元素，需要结合容器的 erase 方法。

定制操作：可以使用自定义操作符来替代默认运算符。

什么是谓词：谓词是一个可调用的表达式，返回结果是一个可以作为条件的值，如返回一个 bool 值。

谓词的参数类型必须是元素类型可以转换到的类型。谓词的实参是输入序列的元素（是元素本身不是迭代器）。

谓词分为一元谓词和二元谓词，分别接受一个参数和两个参数。不同的标准库算法会接受不同的谓词作为参数。 一元谓词代表只接受一个参数，二元谓词代表接受两个参数。下面是书上的二元谓词例子：



接收二元谓词的sort函数

image.png

stable\_sort排序算法：稳定排序算法，维持了相等元素的原有顺序。

Lambda表达式：

Lambda 表达式是一种简洁的方式来定义匿名函数（也称为闭包），它允许你在需要一个函数对象的地方嵌入一个小型函数。尤其是在需要简洁地定义和传递函数的情况下十分有用，例如在算法和回调函数中。



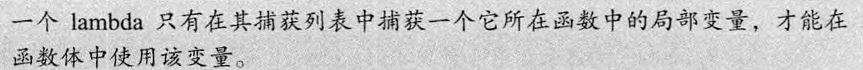
lambda的捕获与返回

定义一个 lambda 时，编译器生成一个与 lambda 对应的未命名的类类型。当向一个函数传递一个 lambda 时，同时定义了一个新类型和该类型的一个对象，传递的参数实际上就是此编译器生成的类类型的未命名对象。

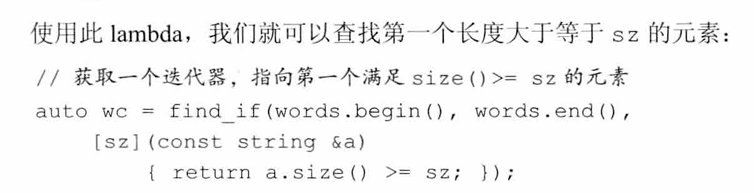
Lambda 表达式的捕获机制允许在 Lambda 表达式内部访问外部作用域中的变量。

lambda不允许有默认参数。

关于捕获列表的使用方法：



使用lambda作为参数传入find\_if方法的谓词参数：



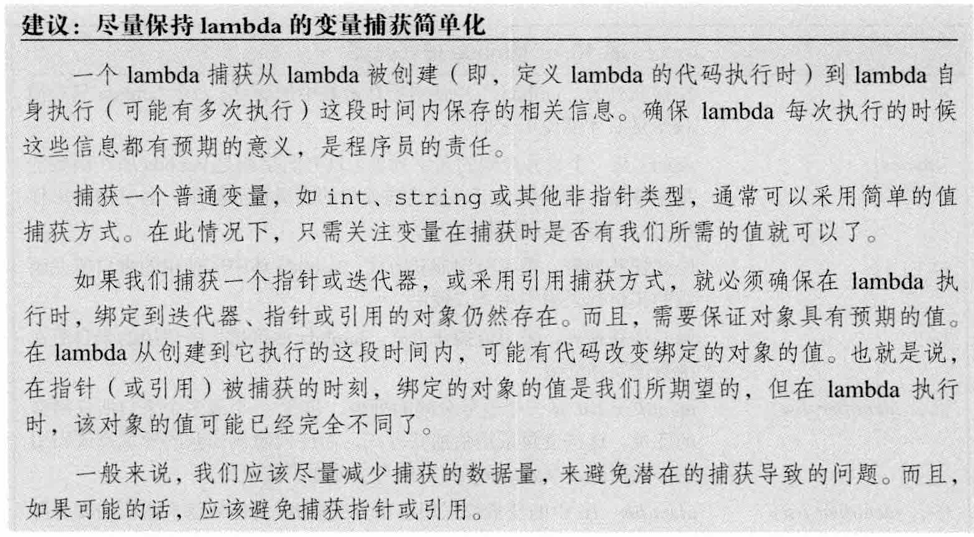
lambda 捕获变量的方式分为值捕获和引用捕获，类似参数传递。

值捕获（捕获变量的副本）：

被捕获的变量的值被拷贝，且捕获的变量的值在lambda创建时才拷贝，而非调用时拷贝。捕获后即使外界的原变量发生改变，但是被捕获的值不会发生改变（因为是拷贝，而且使用值捕获的前提是变量可以被拷贝）。

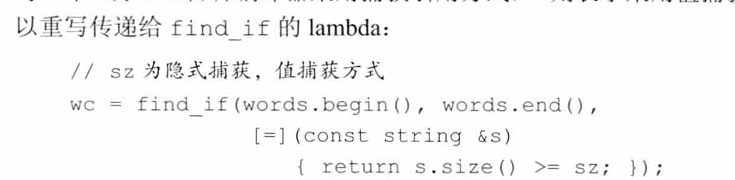
引用捕获：

与值类型不同的是引用捕获的变量，在lambda内部使用时，实际上仍然作用于外界引用绑定的对象。因此在调用时确定外界的该变量是否存在就很有必要，如果此时外界的引用绑定的变量不存在了，会发生未定义行为。

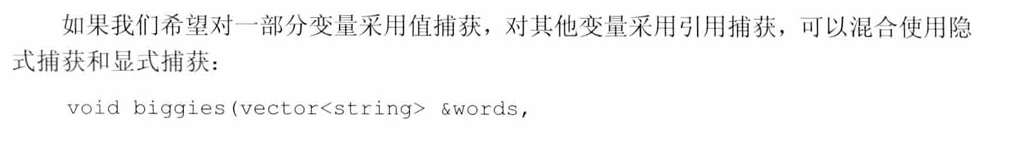


隐式捕获

捕获列表中填&告诉编译器采取引用捕获，填=告诉编译器采用值捕获



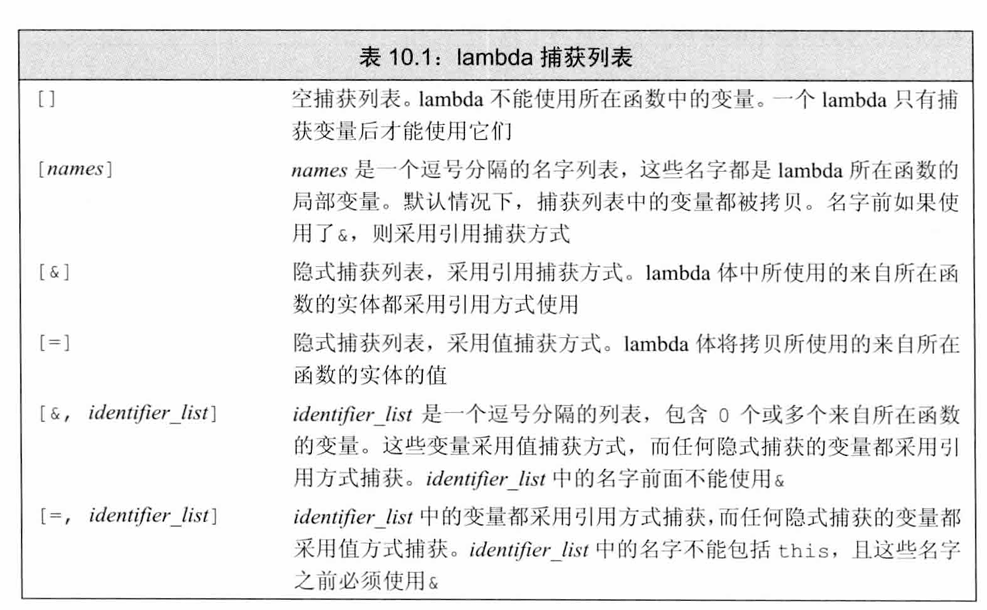
混合捕获，一部分采取隐式捕获，另一部分使用显式捕获





默认的隐式捕获写在显式捕获之前，且显式捕获必然和隐式捕获采取不同方式。

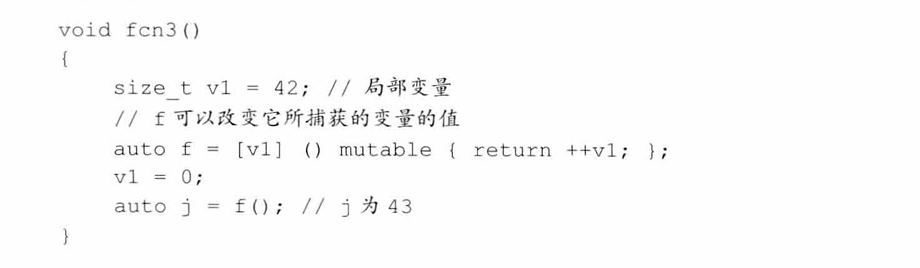
捕获列表：



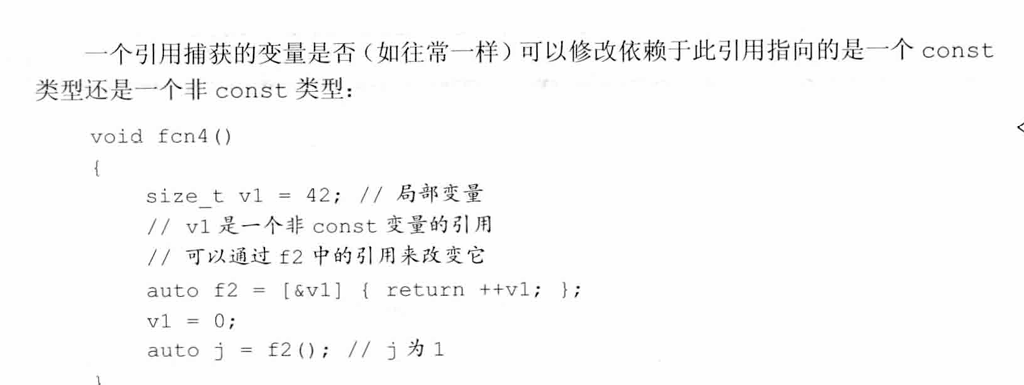
可变lambda

默认情况下，通过值捕获拷贝的变量，lambda 不会改变其值。如果希望改变，必须在参数列表后加上关键字 mutable。

ps：我理解的可变lambda可以更改lambda外部变量过程，拷贝外部变量，在内部改变这个拷贝，最后赋值给外部的变量。

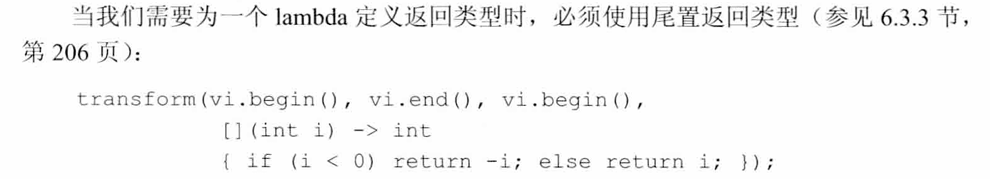


而引用捕获的变量因为修改的一直是同一片空间，因此呈现的结果不同



引用捕获的变量是否可以修改依赖于引用指向的是 const 还是非 const 类型。

lambda指定返回类型需要用尾置返回类型



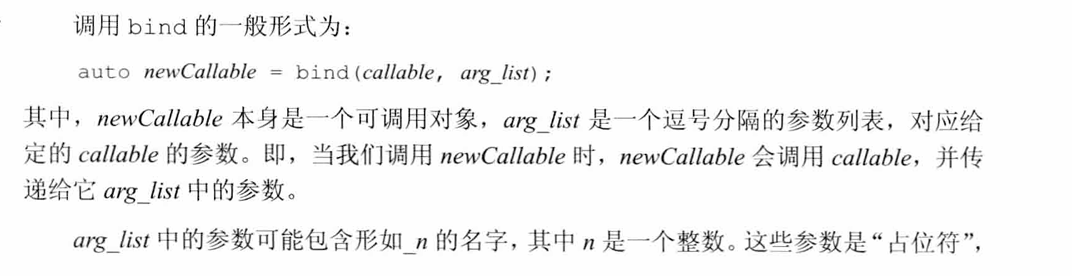
参数绑定

使用参数绑定的意义：lambda 表达式适合那种只在一两个地方使用的简单操作。如果操作需要很多语句或要在很多地方使用，通常应该定义一个函数。

对于捕获列表为空的 lambda，通常可以用函数来代替。对于捕获列表不为空的 lambda，不容易使用函数替换。（例如上面的使用可变参数把修改外界参数）

标准库bind函数

定义在头文件functional头文件



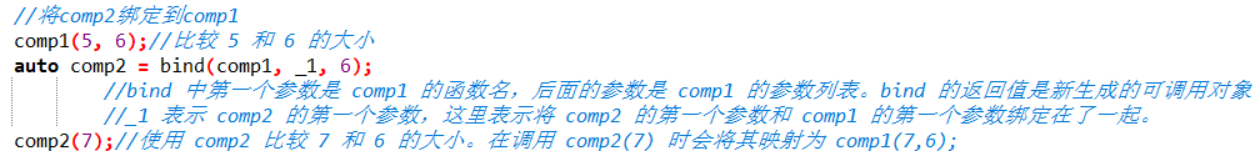


bind绑定的主要功能有两个：

1.可以减少参数数目。（减少掉的参数被设为一个固定值）

2.可以改变参数顺序。

bind方法使用的例子：

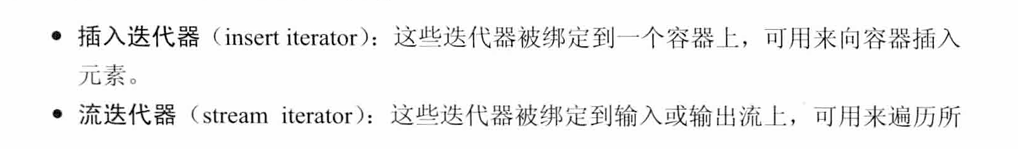


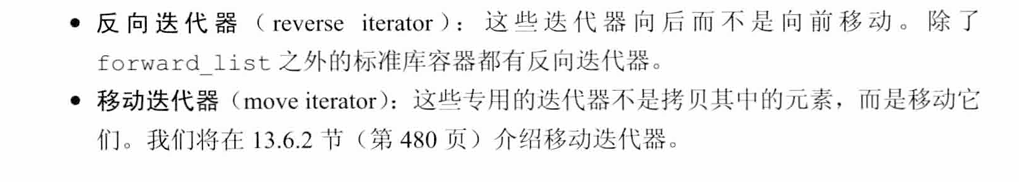
上面下划线接数字的就是bind的占位符：

\_1,\_2,\_n 等占位符分别表示新可调用对象的第 1,2,n 个参数，将 \_n 放在 bind 中不同的参数位置，表示将新可调用对象的第 n 个参数和旧可调用对象在该位置的参数绑定在了一起。

\_1,\_2 等定义在命名空间 placeholders 中，placeholders 这个名字定义在 std 中。placeholders 的实现定义在 functional 头文件中。

迭代器的四种类型：





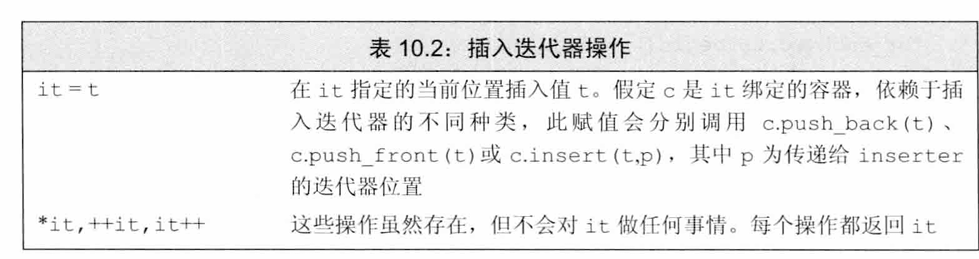
1.插入迭代器

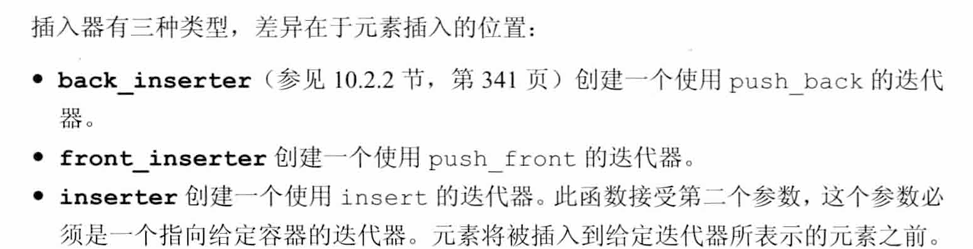
插入器是一种迭代器适配器，它接受一个容器，生成一个迭代器，能实现向给定容器添加元素。

插入器实际上是一个函数。

ps： back\_inserter 是插入器，back\_insert\_iterator> 是插入迭代器类型。back\_inserter(v) 返回绑定到容器 v 的 back\_insert\_iterator，并实现其自增。

插入迭代器的类型和操作：





2.iostream迭代器

流迭代器的作用：

流迭代器可以绑定到 iostream,fstream,stringstream。它将对应的流会被当作一个元素序列来处理。

流迭代器的主要作用是辅助使用泛型算法从流对象读取数据和写入数据。

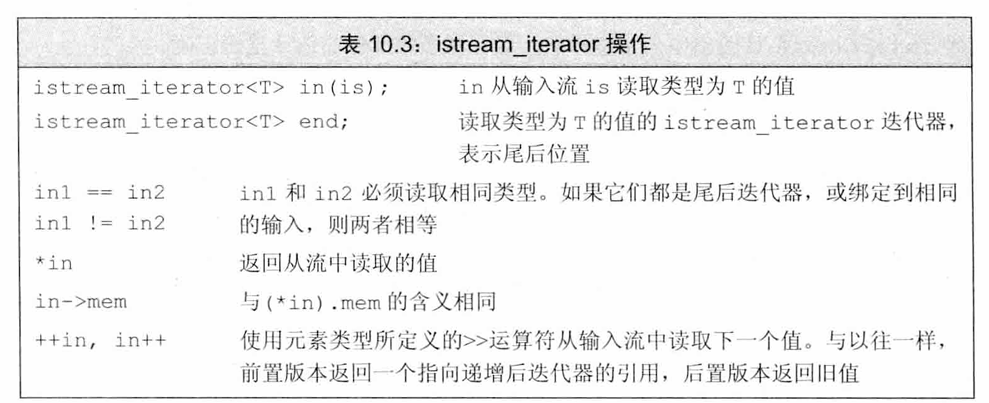
ps：创建流迭代器时，必须指定迭代器将要读写的对象类型。

iostream迭代器分两类：

1.istream\_iterator：读取输入流。

2.ostream\_iterator：向输出流写数据。

流迭代器操作：



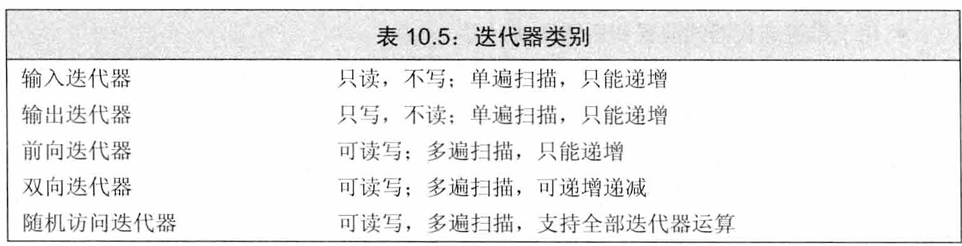
反向迭代器：

除了 forward\_list 外，其他容器都支持反向迭代器。

反向迭代器支持递增和递减操作。注意流迭代器不支持递减操作。

泛型算法结构

根据算法要求的迭代器操作分为五类：



输入迭代器

输入迭代器只用于顺序访问，只能用于单遍扫描算法，如算法 find 和 accumulate。

Istream\_iterator 是一种输入迭代器。

输出迭代器

只能向一个输出迭代器赋值一次，只能用于顺序访问的单遍扫描算法。如 copy 函数的第三个参数。

ostream\_iterator 是一种输出迭代器。

前向迭代器

可以读写元素。只能在序列中沿一个方向移动，可以多次读写同一个元素。

可以保存前向迭代器的状态，使用前向迭代器的算法可以对序列多遍扫描。

算法 replace 要求前向迭代器，forward\_list 的迭代器是前向迭代器。

双向迭代器

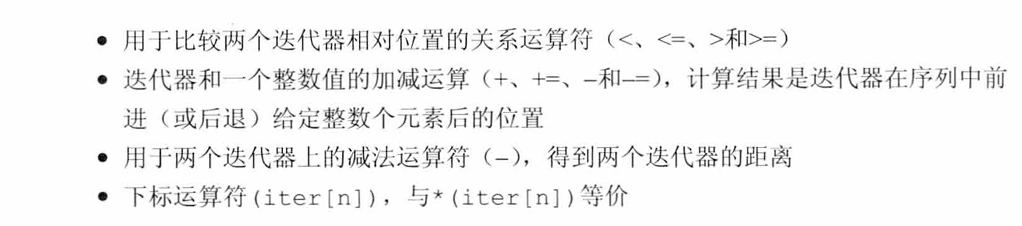
可以正向/反向读写序列中的元素。支持递增递减运算符。

算法 reverse 要求双向迭代器。list 的迭代器是双向迭代器。

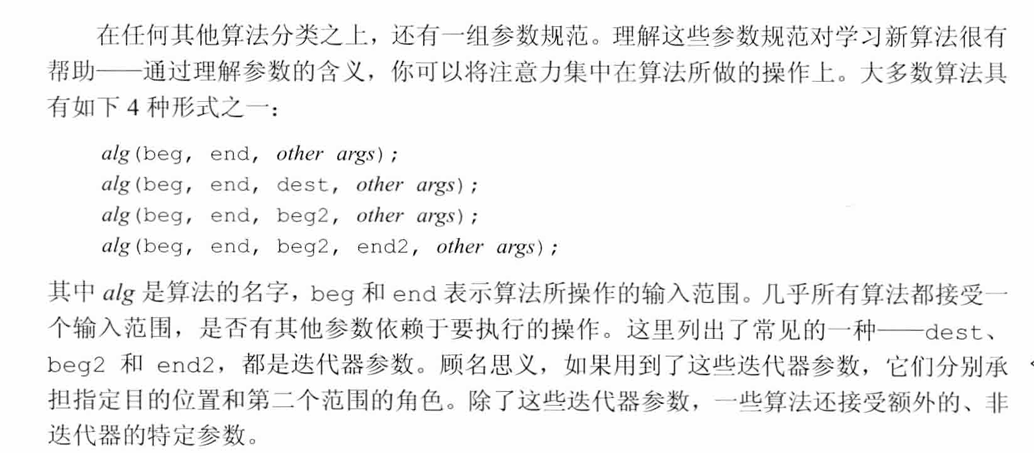
随机访问迭代器

提供在常量时间内访问序列中任意元素的能力。

随机访问迭代器可以当双向迭代器使用，并且支持以下操作：



算法形参模式：



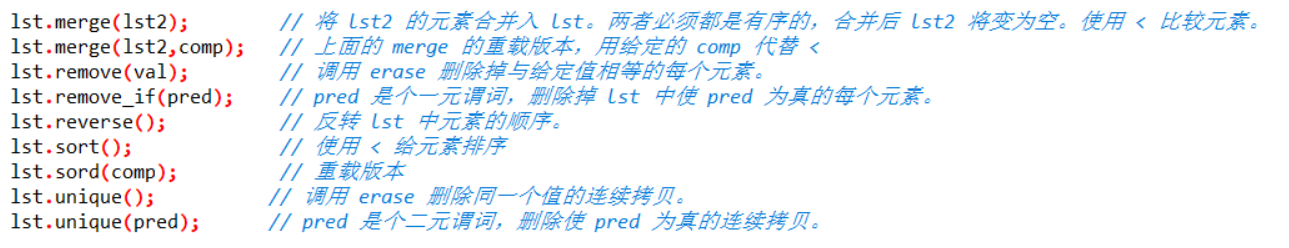
特定容器算法

链表类型 list 和 forward\_list 定义了几个成员函数形式的算法。它们定义了独有的 sort, merge, remove, reverse 和 unique。

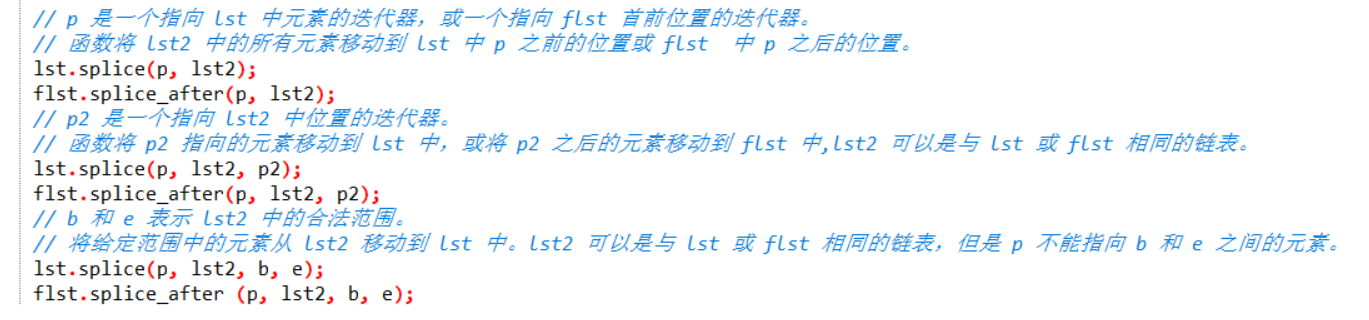
通用版本的 sort 要求随机访问迭代器，而 list 和 forward\_list 分别提供双向迭代器和前向迭代器，因此不能用于 list 和 forward\_list。

其他链表类型定义的算法的通用版本可以用于链表，但是性能差很多，应该优先使用成员函数版本的算法。

成员函数算法：

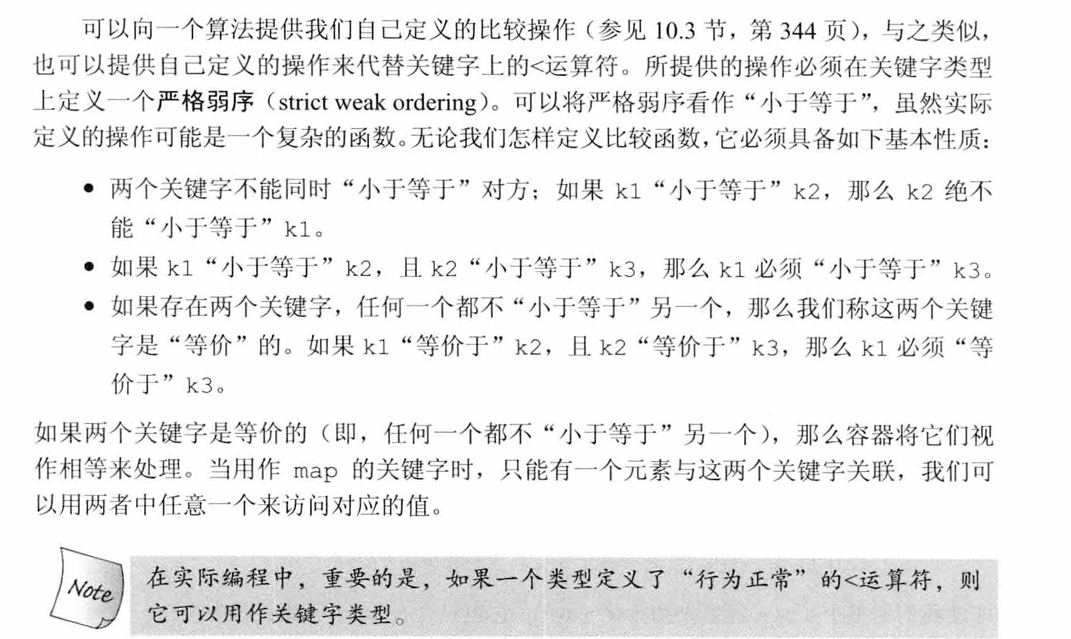


链表独有的splice算法



有序容器关键字类型

有序容器的关键字类型必须定义元素比较的方法，标准库默认使用 < 来比较关键字。可以使用如 vector 等容器的迭代器来作为有序容器的关键字。



使用关键字类型的比较函数