C++容器库和泛型算法

补充：

1、新旧版本标准库比较

（1）旧版本容器类型

A.顺序容器：向量(vector)、双端队列(deque)、双向链表(list)、字符串(string)

B.适配器容器：堆栈(stack)、队列(queue)、优先队列(priority\_queue)

C.关联容器：映射(map)、多重映射(multimap)、集合(set)、多重集合(multiset)

（2）新标准库新增容器

A.顺序容器：单向链表(forward\_list)、数组(array)、初始化列表(initializer\_list)

2、其他概念

（1）容器：利用类模板构建的一组泛型化的数据结构。

（2）泛型算法：利用函数模板构建的一组与元素和容器的类型无关的通用算法。

（3）迭代器：利用操作符重载为不同的容器提供接口一致的元素访问方式，协助泛型算法做到容器无关。

---》迭代器又叫游标卡尺，模仿连续内存的访问模式(地址++)。

（4）随机访问与顺序访问

---》随机访问：下标访问，如数组

---》顺序访问：访问元素，需要遍历整个容器

---》只有vecto和deque这两种连续内存的容器提供随机迭代器，其它内存不连续的容器都只提供顺序迭代器。

=====================================================

二、容器库概览

补充：虽然我们可以在容器中保存几乎任何类型，但某些操作对元素有其自己的特殊要求

---》我们可以为不支持特定操作的类型定义容器，但这种情况下就只能使用那些没有特殊要求的容器操作了

---》例如：可以定义一个包含没有默认构造的类型的容器，但实例化时，不能只给元素数目

1、容器操作（所有容器通用的操作）（p295）

（1）类型名和别名

---》iterator、const\_iterator

---》size\_type： 无符号整数，足够保存容器大小

---》difference\_type： 有符号整数，足够保存两个迭代器之间的距离

---》value\_type： 元素类型

---》reference/const\_reference：元素的引用/元素的const引用

（2）构造函数

---》缺省构造、拷贝构造

---》vector<A> a(iter1, iter2); // 迭代器指定范围拷贝，array不支持

---》vector<A> a{1,2,3,4...}; // 列表初始化

（3）赋值与swap

---》拷贝赋值： c1 = c2；

---》另一种拷贝赋值： c1 = { 1,2,3,4... };

---》swap(a, b) <=> a.swap(b)// 全局函数与成员函数

（4）大小

---》a.size()/a.length() // 返回元素数（forward\_list不支持）

---》a.max\_size() // 返回可保存的最大元素数

---》a.empty() // 判空

（5）添加/删除元素（不适用于array）

注意：这些函数在不同容器中，可能只是函数名一样，形参表不同，并且有多个重载版本

---》a.insert()/a.emplace() // 指定位置插入，可以插入多个（重载）

---》<C++11>：emplace类似于insert，可以用参数直接构造想插入的元素，避免拷贝

---》a.erase()/a.clear() // 删除指定(范围)元素/清空所有元素

（6）获取迭代器

---》正向：a.begin()/a.end()/a.cbegin()/a.cend();

---》反向：a.rbegin()/a.crbegin()...

---》forward\_list不支持反向迭代

---》rbegin()返回尾元素的迭代器，以此类推

2、迭代器

（1）对于正向迭代器而言，起始位置在首元素，终止位置在尾元素之后；对于反向迭代器而言，起始位置在尾元素，终止位置在首元素之前。

（2）任何可能导致容器内存布局发生变化的结构性改变，如增删元素等，都可能使原先获得的迭代器失效，因此稳妥起见，在这样的改变发生以后应该重新获得反应新内存布局的迭代器，保证元素访问正确。

3、容器拷贝

---》两个容器的容器类型和元素类型都必须匹配，才能拷贝

---》但传递迭代器参数来拷贝一个范围为时，就不要求容器类型匹配

4、赋值和swap

（1）赋值

---》赋值运算要求左右具有相同的类型，但assign操作可以让不同但相容的类型赋值

---》赋值相关运算：会导致左边容器内部的迭代器、引用和指针失效

---》所有的STL都有支持深拷贝的拷贝构造和拷贝赋值。

（2）swap

---》不会导致左边容器内部的迭代器、引用和指针失效，除了string

---》swap操作后，指向虽不变，但这些元素已经属于不同容器

---》除了array外，swap不对进行任何拷贝，只交换内部数据结构地址，常数时间内完成

---》swap两个array会真正交换它们，因此所需时间与元素数成正比

---》建议：统一使用新标准提供的非成员版本的swap，且快得多

5、关系运算符

---》每个容器都支持：==、！=

---》除了无序关联容器都支持：>、<、<=、>=

---》比较时，左右两边容器类型、元素类型都必须相同

---》比较方式：与比较字符串类似

---》比较的条件：容器中元素重载了关系运算符

=====================================================

三、顺序容器（Sequence Container）

1、顺序容器独有的操作

（1）容器定义和初始化（array不支持）

---》vector<int> a(n); // 含有n个元素的向量，string不支持

---》vector<int> a(n, t); // 含有n个初始化为t值得元素，t省略元素默认

（2）赋值：assign （p302）

---》assign操作不适用于关联容器和array

---》3个版本（传参）：两个迭代器范围/初始化列表/替换为n个t值，t省略元素默认

（3）添加元素（见p305）

---》push\_back()/emplace\_back()/push\_front()/emplace\_front()/insert()

---》array不支持添加元素，部分容器函数名会有不同

---》insert()有多个版本，可以指定插入别的容器中的迭代器范围

---》insert可以代替某些容器中不支持的push\_front

注意：通过使用insert返回值，可以在容器中的特定位置反复插入元素（p308）

（4）访问元素

---》front()/back() 如果容器是const，则返回const

---》back()不适用于forward\_list

---》c[n]/c.at[n](返回引用)和下标访问只适用于string、vector、deque、array（连续内存）

注意：在使用以上访问方式时，必须用empty()检查容器是否为空，否则会出错

（5）删除元素

---》pop\_back()/pop\_front()：forward\_list不支持pop\_back，string、vector不支持pop\_front

---》erase()：参数迭代器/迭代器范围，forward\_list特殊版本erase\_after()

---》clear()

注意：指定参数的元素，需确保存在

（6）改变容器大小（元素数量）

---》resize(n)/resize(n, t)：默认初始化/初始化成指定值

2、特别的操作注解

（1）insert/emplace：

---》在指定迭代器之前插入一个或多个元素

---》返回第一个新加元素的迭代器（可以重复此操作）

（2）erase：

---》删除迭代器指定范围元素（前迭代器开始，后迭代器之前）

---》返回被删元素之后的迭代器

（3）assign：

---》特色：可以用别的容器的迭代器范围赋值

---》a8 = {2,2,2}; <=> a8.assign(3, 2); <=> a8.assign({2,2,2});

（4）forward\_list独有的操作：insert\_after/emplace\_after/erase\_after：

---》首前元素：before\_begin()，配合上面使用，不可解引用

---》在迭代器之后插入、删除指定后一位元素

---》返回后一位元素迭代器

3、性能比较

（1）所有顺序容器都提供了快速顺序访问的能力。但是，这些容器在访问和增减元素方面都有不同的性能折中

---》vector：支持快速随机访问(下标访问)、在尾部之外的位置插入/删除元素可能很慢

---》deque：双端队列，支持快速随机访问(下标访问)、在头尾位置插入/删除元素很快

---》list：双向链表，只支持双向顺序访问，在任何位置插入/删除元素都很快

---》forward\_list：单向链表，只支持单向顺序访问，在任何位置插入/删除元素都很快

---》array：支持快速随机访问(下标访问)、不能添加/删除元素

---》string：与vector相似，但专门用于保存字符，随机访问快，在尾部位置插入/删除元素很快

4、底层实现原理

（1）string、vector和deque

---》都将元素保存在连续的内存空间：因此可以下标访问，而在中间增减元素慢

---》deque是一个更为复杂的数据结构，但访问和增减的情况与vector相似

（2）list和forward\_list

---》将元素保存在不连续的内存中，占用了额外的内存空间；因此在中间增加元素快

（3）array

---》是一种与内置数据类似，更安全、更容易使用的数据类型

（4）forward\_list

---》设计目标：达到与最好的手写的单向链表数据结构相当的性能

---》因此没有size操作；因为保存或计算大小就会比手写链表多出额外的开销

5、选择使用哪种顺序容器：P293

---》没有特殊需求，一般选用vector

6、vector对象是如何增长的

（1）连续存储元素的容器中，如果没有空间容纳新元素；

---》先将已有元素从旧位置移动到新的更大的空间中

---》然后添加新元素，释放旧空间

---》采用一种可以减少容器重新分配次数的策略（详见p318）

（2）管理容器的成员函数（只适用于vector、deque、string，连续存储容器）

---》c.shrink\_to\_fit()：适用于三者，将capacity()减少至size()，只表示建议

---》c.capacity()：适用于vector、string，不重新分配内存的容器容量

---》c.reserve(n)：适用于vector、string，分配至少能容纳n个元素的内存空间

---》只有当n > capacity()时，reserve(n)才会改变容器容量；成功调用后，capacity() >= n

=====================================================

四、容器适配器（Adaptor）

1、概念：（详见p329）

---》container\_type：实现适配器的底层容器类型

---》适配器：是一种机制，能使某种事物的行为看起来像另一种事物

---》例如：stack适配器接受一个顺序容器，并使其操作起来像一个stack一样

2、给定适配器的容器限制

（1）stack： 默认基于：deque，还可以用：vector、list

（2）queue： 默认基于：deque，还可以用：list

（3）priority\_queue： 默认基于：vector，还可以用：deque

=====================================================

五、泛型算法

---》标准库提供了超过100个算法

---》头文件：<algorithm>、 <numeric>

---》常用非数值算法，如：查找、排序、拆分、合并、复制、交换，等等

1、只读算法：（见p338）

---》有些访问两个序列的操作，可以传递4个迭代器

find/find\_if

count/count\_if 返回第3个参数出现次数

accumulate 返回求和结果，第3个参数决定返回值的类型和和的初始值，<numeric>

equal 返回是否相等，见p339

for\_each 接受一个可调用对象，并对序列每个元素调用此对象

stable\_sort 保证“相等”元素，次序不变

2、写操作算法：（见p340）

（1）泛型算法本身不执行容器操作，不会改变容器大小

---》例如：空容器不可以执行写操作，填充操作不可以越界，删除操作只是将值置为不可用

（2）但可以使用插入迭代器添加元素到容器，如：back\_inserter可以调用push\_back

---》back\_inserter(vec)：参数为容器引用，返回一个插入迭代器

fill/fill\_n 用参数3，填充给定范围

copy/copy\_if 拷贝

replace/replacce\_copy 替换/将替换结果拷贝到另一个序列

transform 对输入序列中每个元素调用可调用对象，并将结果写到目的位置，第三、第四个参数表 示目的位置

（3）排序、排重算法步骤

---》先sort()排序

---》unique将消除相邻的重复项，返回不重复值范围的迭代器

---》使用容器操作erase()，删除尾部的重复值

3、定制操作

---》向某些算法传入可调用对象，表示判断条件或要执行的操作，例如：find\_if，count\_if

---》谓词：可调用的表达式，返回一个能做条件的值，用于给算法定制操作

---》可调用对象，例如：函数指针、lambda表达式

---》一元谓词、二元谓词，意味着可调用对象只接受1或2个参数

---》部分算法接受一元谓词，如：find\_if; 部分接受二元谓词，如：sort

4、lambda表达式（参见p346）

---》一种可调用对象，可以理解为一个未命名的内联函数

（1）与函数相比

---》多了一个捕获列表，可以使用所在函数中定义的局部变量

---》可以省略参数列表和返回类型，如需指明返回类型，必须使用尾置返回

---》lambda不能有默认参数

（2）详细用法（参见p350~p353）

---》引用捕获和值捕获

---》显示捕获、隐式捕获和混合使用

---》可变lambda和引用捕获

---》指定lambda返回类型

（3）代替lambda表达式

---》lambda表达式可以通过捕获列表增加接受的参数

---》可以使用bind函数，函数适配器，参见p354~p357（复杂，了解）

5、再探迭代器

---》额外的迭代器：插入迭代器、流迭代器、反向迭代器、移动迭代器（移动拷贝）

（1）插入迭代器（详见p358）

---》插入器是一种迭代器适配器，模拟迭代器，底层调用push\_back、push\_front或insert

（2）流迭代器（绑定到IO）

---》复杂、懒得了解（参见p359~p362）

（3）反向迭代器

---》++向前；--向后

6、泛型算法结构

（1）迭代器根据算法要求分为5类：（详见p365）

---》输入、输出、单向(forward\_list)、双向、随机访问迭代器

---》注意：给对应的泛型算法，传递符合要求的迭代器类型

---》Warning：向一个算法传递错误的迭代器的问题，很多编译器不会报错

---》举例：通用版本的sort要求随机访问迭代器，而list和forward\_list分别提供双向和单向迭代器

（2）泛型算法的组织规律（了解，p367）

---》算法形参模式

---》算法命名规律（同名重载、if、copy）

7、链表类型容器的成员函数（list、forward\_list）

---》原因：通用版本的sort要求随机访问迭代器，而list和forward\_list分别提供双向和单向迭代器

---》建议：对于链表容器，应该优先使用成员函数版本，而不是通用算法

---》链表成员版本，因为是成员函数，所以可以改变容器大小

---》merge、sort、remove\_if、reverse、unique

---》链表类型独有操作：splice（剪切），参数可以是：单个迭代器、迭代器范围、整个链表

=====================================================

六、关联容器

1、初始化方式

---》可以列表初始化、迭代器范围初始化

2、pair模板

---》make\_pair构造

---》first、second

---》比较大小方式：参见p380

3、类型别名

---》key\_type、mapped\_type、value\_type

4、关键字是const的

---》map和set的key值都是const的

5、无序关联容器

---》map、set会自动排序，以实现快速查找

---》多重set和map会将相同键值连续排列

---》无序容器版本不会排序

6、集合和映射

（1）set

---》没有值只有键的映射。

---》集合会自动排重和排序(平衡有序二叉树)，向集合中插入重复元素会被自动过滤。

---》其迭代器返回常引用。

（2）map

1)逻辑模型：一一对应，键(信息索引)值(信息内容)对，用于信息检索，检索性能可以达到对数级(O(logN))。

2)物理模型：平衡有序二叉树，又名红黑树。

---》平衡有序二叉树：任何一个节点上的左子树和右子树中，元素个数最多相差一个。

3)键必须是唯一的。

4)迭代过程实际上是关于键的中序遍历(L-D-R)，键升序。

5)映射中键都是只读的。

6)构建和修改性能较差。适用于结构稳定但是频繁检索的场合。

7、剩余略，参见p383~396

=====================================================

了解：

1、容器操作可能使用迭代器失效（见p315~p317）

=====================================================

跳过的记录：

1、异常处理的简单介绍

---》p172~p177

2、IO库（见p278~p290）

---》包括：IO类、文件输入输出、string流

1. P320跳过到p328

---》string相关操作