容器

首先，必须理解一下什么是容器，在 C++ 中容器被定义为：在数据存储上，有一种对象类型，它可以持有其它对象或指向其它对像的指针，这种对象类型就叫做容器。很简单，容器就是保存其它对象的对象，当然这是一个朴素的理解，这种“对象”还包含了一系列处理“其它对象”的方法，因为这些方法在程序的设计上会经常被用到，所以容器也体现了一个好处，就是“容器类是一种对特定代码重用问题的良好的解决方案”。

容器还有另一个特点是容器可以自行扩展。在解决问题时常常不知道需要存储多少个对象，也就是说不知道应该创建多大的内存空间来保存对象。显然，数组在这一方面也力不从心。容器的优势就在这里，它不需要预先告诉它要存储多少对象，只要创建一个容器对象，并合理的调用它所提供的方法，所有的处理细节将由容器来自身完成。它可以为你申请内存或释放内存，并且用最优的算法来执行命令。

容器是随着面向对象语言的诞生而提出的，容器类在面向对象语言中特别重要，甚至它被认为是早期面向对象语言的基础。在现在几乎所有的面向对象的语言中也都伴随着一个容器集，在 C++ 中，就是标准模板库（STL ）。

和其它语言不一样，C++ 中处理容器是采用基于模板的方式。标准 C++ 库中的容器提供了多种数据结构，这些数据结构可以与标准算法一起很好的工作，这为软件开发提供了良好的支持！

通用容器的分类

STL 对定义的通用容器分三类：顺序性容器、关联式容器和容器适配器。

一、顺序性容器:是一种各元素之间有顺序关系的线性表，是一种线性结构的可序群集。顺序性容器中的每个元素均有固定的位置，除非用删除或插入的操作改变这个位置。这个位置和元素本身无关，而和操作的时间和地点有关，顺序性容器不会根据元素的特点排序而是直接保存了元素操作时的逻辑顺序。比如一次性对一个顺序性容器追加三个元素，这三个元素在容器中的相对位置和追加时的逻辑次序是一致的。

二、关联式容器---和顺序性容器不一样，关联式容器是非线性的树结构，更准确的说是二叉树结构。各元素之间没有严格的物理上的顺序关系，也就是说元素在容器中并没有保存元素置入容器时的逻辑顺序。但是关联式容器提供了另一种根据元素特点排序的功能，这样迭代器就能根据元素的特点“顺序地”获取元素。

关联式容器另一个显著的特点是它是以键值的方式来保存数据，就是说它能把关键字和值关联起来保存，而顺序性容器只能保存一种（可以认为它只保存关键字，也可以认为它只保存值）。这在下面具体的容器类中可以说明这一点。

三、容器适配器---一个比较抽象的概念，C++的解释是：适配器是使一事物的行为类似于另一事物的行为的一种机制。容器适配器是让一种已存在的容器类型采用另一种不同的抽象类型的工作方式来实

现的一种机制。其实仅是发生了接口转换。可以把它理解为容器的容器，它实质还是一个容器，只是它不依赖于具体的标准容器类型，可以理解是容器的模版。或者把它理解为容器的接口，而适配器具体采用哪种容器类型去实现，可以在定义适配器的时候决定。

下表列出 STL 定义的三类容器所包含的具体容器类：

顺序性容器:

1.vector

特点：从后面快速的插入与删除，直接访问任何元素

2.deque

特点：从前面或后面快速的插入与删除，直接访问任何元素

3.list

特点：双链表，从任何地方快速插入与删除

关联容器

1.set

特点：快速查找，不允许重复值

2.multiset

特点：快速查找，允许重复值

3.map

特点：一对多映射，基于关键字快速查找，不允许重复值

4.multimap

特点：一对多映射，基于关键字快速查找，允许重复值

容器适配器

1.stack

特点：后进先出

2.queue

特点：先进先出

3.priority_queue

特点：最高优先级元素总是第一个出列

vector ,deque 和 list

顺序性容器：

向量 vector ：

是一个线性顺序结构。相当于数组，但其大小可以不预先指定，并且自动扩展。它可以像数组一样被操作，由于它的特性我们完全可以将 vector 看作动态数组。

在创建一个 vector 后，它会自动在内存中分配一块连续的内存空间进行数据存储，初始的空间大小可以预先指定也可以由 vector 默认指定，这个大小即 capacity （）函数的返回值。当存储的数据超过分配的空间时 vector 会重新分配一块内存块，但这样的分配是很耗时的，在重新分配空间时它会做这样的动作：

首先，vector 会申请一块更大的内存块；

然后，将原来的数据拷贝到新的内存块中；

其次，销毁掉原内存块中的对象（调用对象的析构函数）；

最后，将原来的内存空间释放掉。

如果 vector 保存的数据量很大时，这样的操作一定会导致糟糕的性

能（这也是 vector 被设计成比较容易拷贝的值类型的原因）。所以说 vector 不是在什么情况下性能都好，只有在预先知道它大小的情况下 vector 的性能才是最优的。

vector 的特点：

(1) 指定一块如同数组一样的连续存储，但空间可以动态扩展。即它可以像数组一样操作，并且可以进行动态操作。通常体现在 push_back() pop_back() 。

(2) 随机访问方便，它像数组一样被访问，即支持[ ] 操作符和 vector.at()

(3) 节省空间，因为它是连续存储，在存储数据的区域都是没有被浪费的，但是要明确一点 vector 大多情况下并不是满存的，在未存储的区域实际是浪费的。

(4) 在内部进行插入、删除操作效率非常低，这样的操作基本上是被禁止的。Vector 被设计成只能在后端进行追加和删除操作，其原因是 vector 内部的实现是按照顺序表的原理。

(5) 只能在 vector 的最后进行 push 和 pop ，不能在 vector 的头进行 push 和 pop 。

(6) 当动态添加的数据超过 vector 默认分配的大小时要进行内存的重新分配、拷贝与释放，这个操作非常消耗性能。所以要 vector 达到最优的性能，最好在创建 vector 时就指定其空间大小。

双向链表 list：

是一个线性链表结构，它的数据由若干个节点构成，每一个节点

都包括一个信息块（即实际存储的数据）、一个前驱指针和一个后驱指针。它无需分配指定的内存大小且可以任意伸缩，这是因为它存储在非连续的内存空间中，并且由指针将有序的元素链接起来。

由于其结构的原因，list 随机检索的性能非常的不好，因为它不像 vector 那样直接找到元素的地址，而是要从头一个一个的顺序查找，这样目标元素越靠后，它的检索时间就越长。检索时间与目标元素的位置成正比。

虽然随机检索的速度不够快，但是它可以迅速地在任何节点进行插入和删除操作。因为 list 的每个节点保存着它在链表中的位置，插入或删除一个元素仅对最多三个元素有所影响，不像 vector 会对操作点之后的所有元素的存储地址都有所影响，这一点是 vector 不可比拟的。

list 的特点：

(1) 不使用连续的内存空间这样可以随意地进行动态操作；

(2) 可以在内部任何位置快速地插入或删除，当然也可以在两端进行 push 和 pop 。

(3) 不能进行内部的随机访问，即不支持[ ] 操作符和 vector.at() ；

(4) 相对于 verctor 占用更多的内存。

双端队列 deque ：

是一种优化了的、对序列两端元素进行添加和删除操作的基本序列容器。它允许较为快速地随机访问，但它不像 vector 把所有的对象保存在一块连续的内存块，而是采用多个连续的存储块，并且在一

个映射结构中保存对这些块及其顺序的跟踪。向 deque 两端添加或删除元素的开销很小。它不需要重新分配空间，所以向末端增加元素比 vector 更有效。

实际上，deque 是对 vector 和 list 优缺点的结合，它是处于两者之间的一种容器。

deque 的特点：

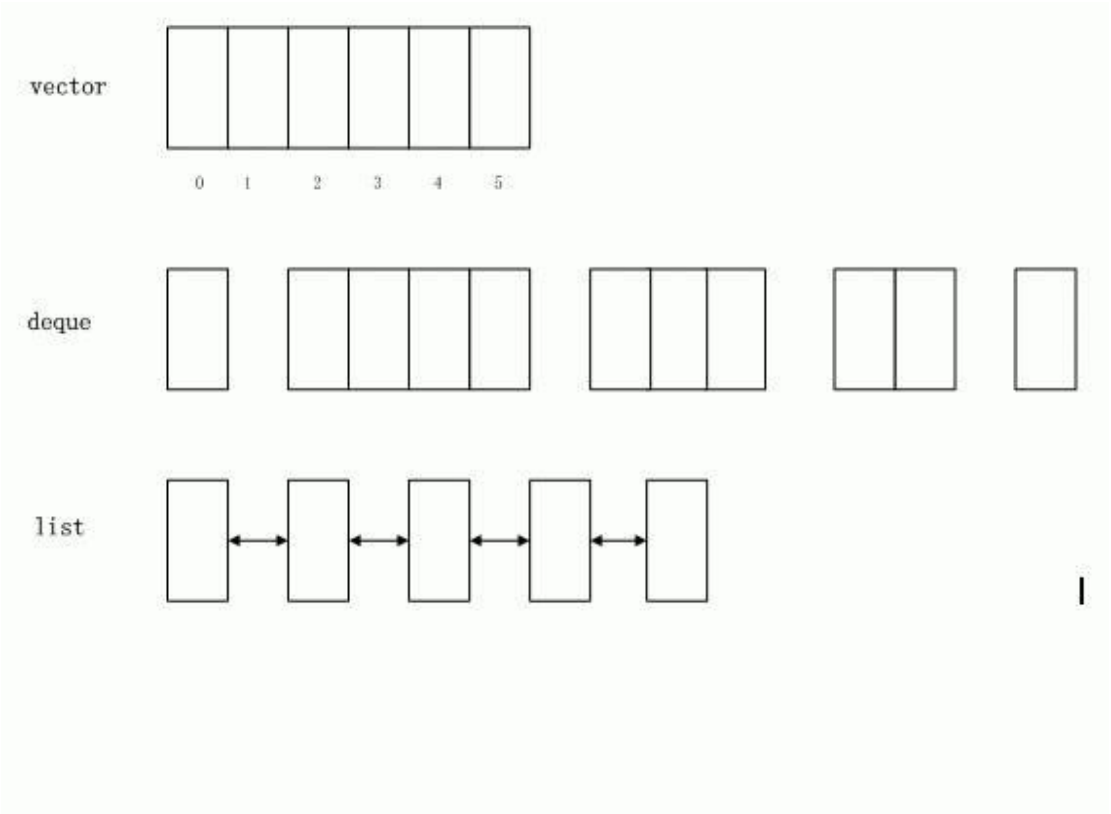
(1) 随机访问方便，即支持[ ] 操作符和 vector.at() ，但性能没有 vector 好；

(2) 可以在内部进行插入和删除操作，但性能不及 list；

(3) 可以在两端进行 push 、pop；

三者的比较

下图描述了 vector 、list 、deque 在内存结构上的特点：



vector 是一段连续的内存块，而 deque 是多个连续的内存块， list 是所有数据元素分开保存，可以是任何两个元素没有连续。

vector 的查询性能最好，并且在末端增加数据也很好，除非它重新申请内存段；适合高效地随机存储。

list 是一个链表，任何一个元素都可以是不连续的，但它都有两个指向上一元素和下一元素的指针。所以它对插入、删除元素性能是最好的，而查询性能非常差；适合大量地插入和删除操作而不关心随机存取的需求。

deque 是介于两者之间，它兼顾了数组和链表的优点，它是分块的链表和多个数组的联合。所以它有被 list 好的查询性能，有被 vector 好的插入、删除性能。如果你需要随即存取又关心两端数据的插入和删除，那么 deque 是最佳之选。

关联容器

set, multiset, map, multimap 是一种非线性的树结构，具体的说采用的是一种比较高效的特殊的平衡检索二叉树—— 红黑树结构。（至于什么是红黑树，我也不太理解，只能理解到它是一种二叉树结构）

因为关联容器的这四种容器类都使用同一原理，所以他们核心的算法是一致的，但是它们在应用上又有一些差别，先描述一下它们之间的差别。

set，又称集合，实际上就是一组元素的集合，但其中所包含的元素的值是唯一的，且是按一定顺序排列的，集合中的每个元素被称作集合中的实例。因为其内部是通过链表的方式来组织，所以在插入的

时候比 vector 快，但在查找和末尾添加上被 vector 慢。

multiset，是多重集合，其实现方式和 set 是相似的，只是它不要求集合中的元素是唯一的，也就是说集合中的同一个元素可以出现多次。

map，提供一种“键- 值”关系的一对一的数据存储能力。其“键”在容器中不可重复，且按一定顺序排列（其实我们可以将 set 也看成是一种键- 值关系的存储，只是它只有键没有值。它是 map 的一种特殊形式）。由于其是按链表的方式存储，它也继承了链表的优缺点。

multimap，和 map 的原理基本相似，它允许“键”在容器中可以不唯一。

关联容器的特点是明显的，相对于顺序容器，有以下几个主要特点：

1，其内部实现是采用非线性的二叉树结构，具体的说是红黑树的结构原理实现的；

2，set 和 map 保证了元素的唯一性，mulset 和 mulmap 扩展了这一属性，可以允许元素不唯一；

3，元素是有序的集合，默认在插入的时候按升序排列。

基于以上特点，

1，关联容器对元素的插入和删除操作比 vector 要快，因为 vector 是顺序存储，而关联容器是链式存储；比 list 要慢，是因为即使它们同是链式结构，但 list 是线性的，而关联容器是二叉树结构，其改变一个元素涉及到其它元素的变动比 list 要多，并且它是排序的，每次插

入和删除都需要对元素重新排序；

2，关联容器对元素的检索操作比 vector 慢，但是比 list 要快很多。vector 是顺序的连续存储，当然是比不上的，但相对链式的 list 要快很多是因为 list 是逐个搜索，它搜索的时间是跟容器的大小成正比，而关联容器查找的复杂度基本是 Log(N)，比如如果有 1000 个记录，最多查找 10 次，1,000,000 个记录，最多查找 20 次。容器越大，关联容器相对 list 的优越性就越能体现；

3，在使用上 set 区别于 vector,deque,list 的最大特点就是 set 是内部排序的，这在查询上虽然逊色于 vector，但是却大大的强于 list。

4，在使用上 map 的功能是不可取代的，它保存了“键- 值”关系的数据，而这种键值关系采用了类数组的方式。数组是用数字类型的下标来索引元素的位置，而 map 是用字符型关键字来索引元素的位置。在使用上 map 也提供了一种类数组操作的方式，即它可以通过下标来检索数据，这是其他容器做不到的，当然也包括 set。（STL 中只有 vector 和 map 可以通过类数组的方式操作元素，即如同 ele[1]方式）

容器适配器

STL 中包含三种适配器：栈 stack、队列 queue 和优先级 priority_queue。

适配器是容器的接口，它本身不能直接保存元素，它保存元素的机制是调用另一种顺序容器去实现，即可以把适配器看作“它保存一个容器，这个容器再保存所有元素”。

STL 中提供的三种适配器可以由某一种顺序容器去实现。默认下 stack 和 queue 基于 deque 容器实现，priority_queue 则基于 vector 容

器实现。当然在创建一个适配器时也可以指定具体的实现容器，创建适配器时在第二个参数上指定具体的顺序容器可以覆盖适配器的默认实现。

由于适配器的特点，一个适配器不是可以由任一个顺序容器都可以实现的。

栈 stack 的特点是后进先出，所以它关联的基本容器可以是任意一种顺序容器，因为这些容器类型结构都可以提供栈的操作有求，它们都提供了 push_back、pop_back 和 back 操作；

队列 queue 的特点是先进先出，适配器要求其关联的基础容器必须提供 pop_front 操作，因此其不能建立在 vector 容器上；

优先级队列 priority_queue 适配器要求提供随机访问功能，因此不能建立在 list 容器上。