STL(Standard Template Library)标准模板库

STL六大组件

1. 容器(containers)
2. 算法(algorithms)
3. 迭代器(iterators)
4. 仿函数(functors)
5. 配接器(adaptors)
6. 配置器(allocators)

1.容器

1. 序列式容器

1.vector

1.vector迭代器是Random Access Iterators。

2.所采用数据结构是线性连续空间。初始化，指定配置n个空间。

3.push_back操作请求新空间时：如果原大小是0，则配置1；否则配置原空间大小的两倍。前半段放置原数据，后半段准备放置新数据。

4.insert请求新空间，备用空间小于新增元素个数，则新空间大小是new_size=old_size+max(old_size,n)。

5.erase/clear会清除位置上的元素，但是不会向操作系统归回空间。不会执行紧缩操作，capacity保持原来大小。

2.list

1.list迭代器是Bidirectional Iterator，不能使用sort算法，因为sort只接受Random Access Iterator，使用自己的sort函数。

2.循环双向链表，end指向尾端的空白节点,begin=end->next。

3.插入在指定位置之前插入，采用头插法。

4.迁移操作(transfer),将某连续范围的元素迁移到某个特定位置之前。结合操作(splice)，将某连续范围的元素从一个list移动到一个list的某个特定位置之前(内部调用transfer操作实现)。

3.deque

1.deque迭代器是Random Access Iterators(cur,first,last,node)。cur指向迭代器当前所指向的元素，fisrt指向cur指针指向缓冲区的第一个元素，last指向cur指针指向缓冲区的最后一个元素的下一位置，node指向主控map中指向当前缓冲区的节点。

2.双端队列。双向开口的连续线性空间,动态以分段连续空间组合而成。

3.采用map作为主控。map时一小块连续空间，每个元素都是一个指针，指向实际储存元素的空间，也被称为缓冲区。初始化时，主控map的长度max(8,所需节点+2(前后预留一个，扩充时使用))。

4.若map空间不足，请求配置新空间：map_size+max(map_size,nodes_to_add)+2。

5.clear完成后回复初始状态，保留一个缓冲区。

4.stack

1.先进后出数据结构，默认以deque为stack底部结构。

2.没有迭代器。

3.配接器(adapter)。

5.queue

1.先进先出数据结构，默认以deque为queue底部结构。

2.没有迭代器。

3.配接器(adapter)。

6.heap

1.堆。

2.以vector来实现堆。

3.没有迭代器。

7.priority_queue

1.默认以vector为底部容器。

2.默认以大顶堆来完成优先队列功能。

3.没有迭代器。

8.slist

1.单向链表。

2.slist迭代器是Forward Iterators。

3.只提供push_front，采用尾插法插入。

2. 关联式容器

1.RB-tree

1.红黑树特性

1.每个节点不是黑色就是红色

2.根节点是黑色

3.如果节点是红色，那么子节点一定是黑色。不允许连续的红色节点出现。

4.任一个节点至NULL的任何路径，所含黑节点数必然相同。新插入的节点必然是红色节点。

2.防止插入节点的父子节点皆为红色而持续向RB-tree的上层结构发展，实行一个预处理：新增节点A，沿着A的路径向上递归，如果某节点X的两个子节点都是红色，那么就将X改成红色，并将两个子节点改为黑色。

3.双层迭代器和双层节点设计。_rb_tree_node继承_rb_tree_node_base，*rb_tree_iterator继承自_rb_tree_iterator_base*。

4.迭代器是Bidirectional Iterator，但是没有随机定位功能。

5.实现技巧：**为根节点root再设计一个父节点header,root父节点指向header，header的父节点指向root，left指向最左节点，right指向最右节点。**

6.begin()指向最左节点，end()指向header节点。

2.set,multiset

1.以RB-tree为底层机制。

2.所有元素自动排序，set不允许存在相同的元素，multiset允许存在相同的元素。

3.insert和erase操作，迭代器仍然有效。

4.set插入时调用RB-tree的insert_unique函数(遇到相同元素直接返回，不插入)，multiset插入时调用RB-tree的insert_equal函数(遇到相同元素，继续向右走)。

3.map,multimap

1.以RB-tree为底层机制。

2.所有键值自动排序，map不允许存在相同的键值，multiset允许存在相同的键值。

3.insert和erase操作，迭代器仍然有效。

4.set插入时调用RB-tree的insert_unique函数(遇到相同元素直接返回，不插入)，multiset插入时调用RB-tree的insert_equal函数(遇到相同元素，继续向右走)。

4.hash_table

1.负载系数(loading factor)：元素个数与哈希表长度的比值。

2.主集团，次集团。

3.解决哈希冲突的方法：线性探测，二次探测，拉链法(hash_table使用拉链法)。

5.hash_set,hash_map,hash_multiset,hash_multimap

1.以hash_table为底层机制。

2.无自动排序功能

3.缺省时使用大小为100的表格，并呗hash_table调整为最接近且较大的质数，即193。

2. 算法

1.sort排序算法

1.只接受俩个Random Access Iterator。

2.算法流程。首先使用快速排序(quick sort)，枢轴(pivot)采用三点中值(首，尾和中间值)。防止分割行为恶化，设定阈值(最多分割数为log n)，超过阈值则改用heap sort(堆排序)。如果区间长度小于等于16，则对该区间使用插入排序。

3. 迭代器

最常用到的迭代器型别：value_type(迭代器所指对象的型别),difference_type(两个迭代器的距离),pointer(指针),reference(引用),iterator_category(迭代器的类别)。

1.Input Iterator,只读

2.Output Iterator，只写

3.Forward Iterator

4.Bidirectional Iterator

5.Random Access Iterator

trait(特性萃取)

trait可以萃取各个迭代器的特性，即是迭代器的相应类别。

1.利用function template的参数推导(argument deducation)。只能推导函数的参数，无法推导函数返回值。

2.声明内嵌类型。但是迭代器如果不是class type，就无法为它定义内嵌型别。原生指针就不是class type。

3.偏特化(partial specialization):针对template的参数更进一步的条件限制所设计出来的一个特化版本。

1.针对原生指针设计一个偏特化版本。

2.针对指向常数对象的指针(point-to-const)设计一个偏特化版本。

4. 仿函数

1.仿函数：重载函数调用运算符(括号operator())的类。

2.调用者可以像函数一样调用仿函数，在仿函数中则是以对象所定义的*函数调用符号(function call operator)*去扮演函数的实际角色。

3.一元仿函数继承 unary_function，二元仿函数继承 binary_function。

4.证同元素：与该元素做op运算，任会得到自身。

5. 配接器

将一个class的接口转换为另一个class的接口，使原本因接口不兼容而不能合作的classes，可以一起运作。

改变仿函数接口：function adapter，改变容器接口：container adapter，改变迭代器接口：iterator adapter。

1.container adapter：stack，queue

2.iterator adapter：

1. insert iterators：将赋值assign转变成插入insert操作。

2.reverse iterators：将++变成--，将--变成++。

3.IOStream iterator：将迭代器绑定到iostream对象上。内部维护一个iostream对象。

3.function adapter：配接操作：bind(连结)，negate(否定)，compose(组合)。

1.所有期望获得配接能力的组件，其本身都必须是可配接的。

1.一元仿函数必须继承 unary_function，二元仿函数必须继承 binary_function。

2.成员函数必须经过mem_fun处理，一般函数必须经过ptr_fun处理。

3.一个未经ptr_fun处理的一般函数，虽然也可以以函数指针的形式传给STL算法使用，但是没有拥有配接能力。

6. 配置器

SGI STL使用alloc而非allocator。

- SGI使用两级配置器。
  - 第一级配置器直接使用malloc()和free()来分配空间，有out-of-memory处理机制。
  - 第二级配置器：当配置区块超过128bit时，调用一级配置器来配置；否则采用内存池(memory pool)技术。内存池维护16个free list，各自管理8至128bytes的区块。当free list中没有可用的区块，就调用refill为free list重新填充空间。默认取相应的20个新节点，若内存池空间不足，新节点不足20但多于1个，则分配。如果内存池一个节点都分配不了，则调用malloc从系统heap中分配内存，大小为需求量的两倍再加上一个附加量。如果系统的heap空间不足，寻找较大区块的free list；如果还是没有，则调用第一级配置器。

POD(Plain Old Data)数据

1.一个类或结构体通过二进制拷贝后还能保持其数据不变，那么它就是一个POD类型。

2.POD数据类型主要用来解决C++与C之间数据类型的兼容性，以实现C++程序与C函数的交互

3.只有同时满足*平凡的（trival）*和标准布局的（standard layout）*两个基本概念才能称为是POD类型

1.平凡的(trival)

1.拥有平凡的默认构造函数（trivial constructor）和析构函数（trivial destructor）。

2.拥有平凡的复制构造函数（trivial copy constructor）和移动构造函数（trivial move constructor

3.拥有平凡的复制赋值运算符（trivial assignment operator）和移动赋值运算符（trivial move operator）。

4.不能包含虚拟函数和虚拟基类。

2.标准布局的(standard layout)

1.所有非静态成员都有相同的访问权限（public, private, protected）。

2.在class或者struct继承时，满足以下两种情况之一的class或者struct也是标准布局的。

1.派生类中有非静态成员，且只有一个仅包含静态成员的基类。

2.基类有非静态成员，而派生类没有非静态成员。

3.类中第一个非静态成员的类型与其基类不同。

4.没有虚拟函数和虚基类。

5.所有非静态数据成员均符合标准布局类型，其基类也符合标准布局。

4.POD类型的好处

1.字节赋值(bytewise copy)。可以使用memset和memcpy对POD类型进行初始化。

2。提供对C内存布局的兼容。C++程序可以与C函数进行交互操作，POD类型保证这种在C与C++之间的操作总是安全的。

3。保证了静态初始化的安全有效，用于提供程序性能。直接放入.bss段，在初始化中直接赋0。