第一章：STL概论

STL六大组件：容器，算法，迭代器，仿函数，配接器，配置器

第二章：空间配置器

空间配置器：为STL存放数值的各种容器配置空间的，空间包括但不限于内存

allocator有许多接口

自行实现一个简单的空间配置器：主要实现了分配内存并返回指针给对象的功能，用到了new，delete等运算符，还有对应对象的构造函数析构函数等。

次配置力的SGI空间配置器：SGI STL每一个容器都已经指定缺省的空间配置器为alloc，SGI也有一个符合标准的名为allocator的空间配置器，但是SGI不建议我们使用，因为效率不佳，只是把C++的new和delete做了一层包装。

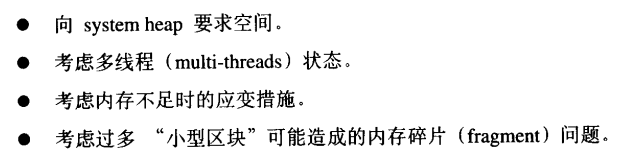
一般的内存分配：new和delete new先分配内存，再构造对象内容，delete先调用析构再释放内存

在STL空间配置器中：两阶段被分开，alloc::allocate()配置内存，alloc::deallocate()释放内存 对象构造由::construct()负责，析构由::destroy()负责

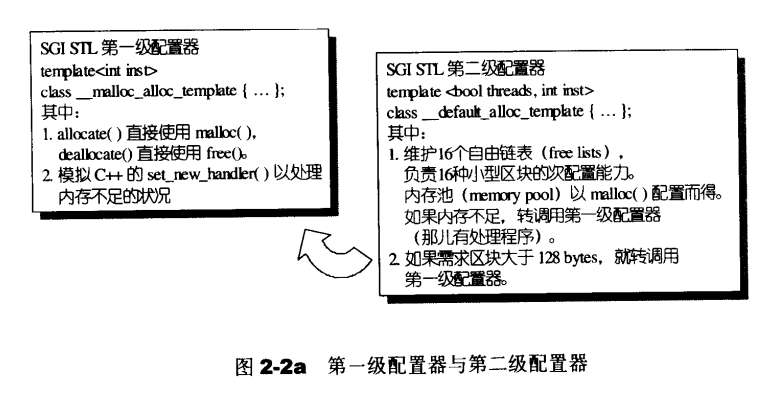
配置器定义在<memory>中：而SGI<memory>包含stl\_alloc.h和stl\_construct.h两个文件，stl\_alloc.h定义了一二级配置器，stl\_construct.h定义了全局函数construct() destroy()，负责对象的构造和析构。

stl\_construct.h中使用construct()接收一个指针p和一个初值value，这个函数用途是将初值设定到指针所指的空间上，new来构造对象，返回指针，destory()有直接析构的版本，接收指针，将指针所指之物析构掉，有接收迭代器的版本，将first和last范围的析构掉

stl\_alloc.h控制内存配置和释放，设计哲学

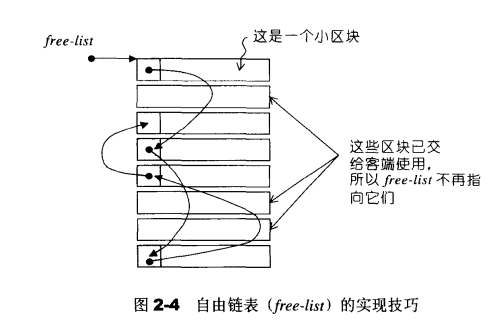


为了避免小型区块造成内存破碎，SGI使用了双层级配置器，第一层直接使用malloc和free，第二层则根据配置区间来决定，超过128字节使用第一级配置器，小于则使用内存池整理方式



C++ new handler机制是：你可以要求系统在内存配置需求无法满足时，调用一个你指定的函数。可以重复使用处理函数直到内存足够。

SGI STL二级配置器是每次配置一大块内存，并维护对应之自由链表。下次若再有相同的内存需求，就直接从free-lists中播出，如果客端释放归还小额区块，就由配置器会受到free-list中，并且SGI第二级配置器会主动将任何小额区块内存需求量上调至8的倍数并维护16个free-lists，为了节省内存，链表节点还用的是union，一处两用



空间配置函数allocate()，身为配置器的标准接口函数。首先判断区块大小，大于128byte调用第一级配置器，小于128byte检查对应的free list。如果freelist之内有可用的区块，就直接拿来用，如果没有可用区块，就将区块大小上调8倍数边界，然后调用refill()，准备为free list重新填充空间

空间释放函数deallocate()，大于128bytes就调用第一季配置器，小于128bytes就找出对应的freelist，将区块回收

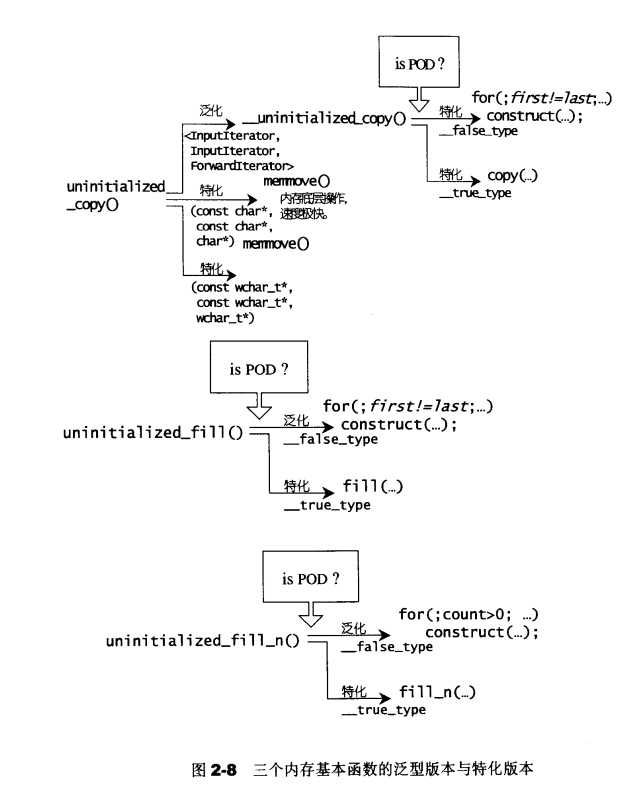
refill()函数：当freelist没有区块了，就调用refill()函数，为freelist重新填充空间。新的空间取自内存池。缺省取得20个新节点。

内存池：从memory pool取得空间给free list使用，是chunk\_alloc()的工作。先判断内存池的容量，够就直接给20个到freelist，不够二十但还有一个往上就给这么多，完全没了就从堆里再申请新的内存给内存池，如果整个系统堆内存都不够了，再寻找没使用完区块的另一个freelist，找到了就分一点给现在这个，找不到就要用第一级配置器了，它有out of memory机制，或许能分配成功，最终失败则发出badalloc异常。

内存基本处理工具：uninitialized\_copy() 函数把result范围内的每一个迭代器都指向未初始化区域，则uninitialized\_copy()会使用copy constructor给输入来源的范围内的每一个对象产生一份复制品，放进输出范围里。如果你需要实现一个容器，uninitialized\_copy()这样的函数可以配置内存区块，足已包含范围内的所有元素，使用uniniitalized\_copy()，在该内存区块上构造元素。

uninitialized\_fill函数会在范围上产生第三个参数x的复制品，范围内要么全部产生，要么全部别产生，如果有任何一个copy constructor出现异常，就必须析构掉其他所有的元素

uninitialized\_fill\_n接受三个参数，迭代器first指向初始化空间的起始位置，n表示欲初始化空间的大小，x表示初值，



第三章：迭代器概念与traits编程技法

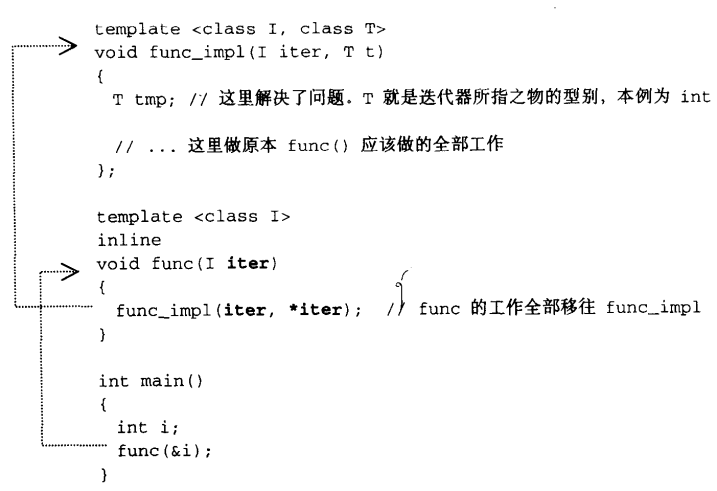
迭代器（iterator）模式：提供一种方法，使之能够依序巡防某个聚合物（容器）所含的各个元素，而又无需报炉该聚合物的内部表述方式

容器的泛化：类模板 class template

算法的泛化：函数模板 function template

迭代器扮演容器和算法中间的粘胶角色，迭代器是一种智能指针

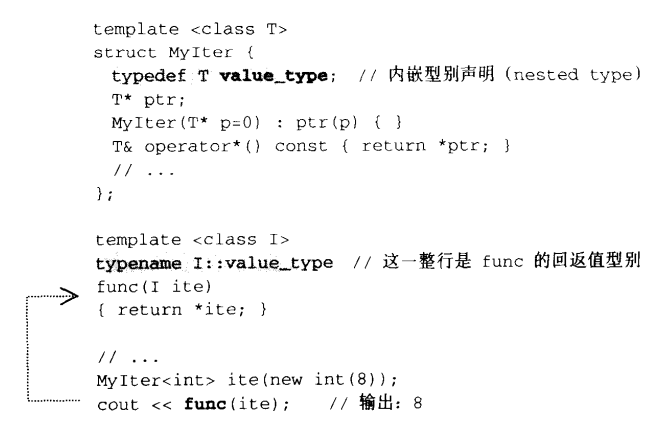
怎么得到迭代器的类别？：利用函数模板的从参数推导机制例如：



而利用此机制可以得到迭代器指向的所属的元素的类别，

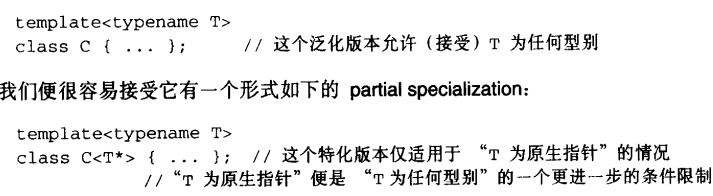
Traits编程技法：

迭代器所指对象的类型：该迭代器的value type。万一value type要作为函数的传回值，可以用声明内嵌类型来解决：



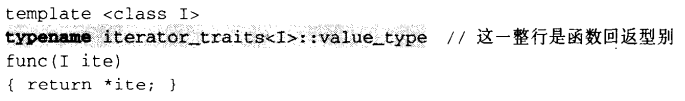
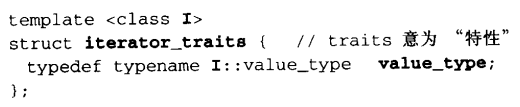
typename不能忘，因为T是一个template参数，未具现化之前，编译器对T一无所知，，通过typename告诉编译器这是一个类型，才能顺利通过编译

特化：任何模板template参数更进一步的条件限制所设计出来的一个特化版本，如下图：



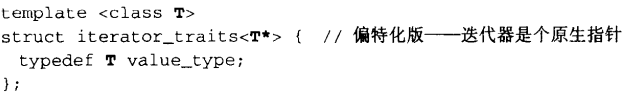
那如果迭代器的类型是一个特化的，要求迭代器为指针，就无法为它们定义内嵌型别。

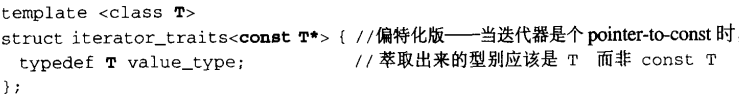
针对迭代器的特化版本，用traits



假设I有value\_type，多了一层间接性，好处是迭代器可以拥有特化的版本了，迭代器是一个原生指针，不是一种class类型，也可以通过在这种方式得到类型了。

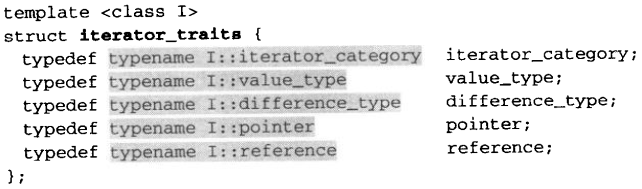
不管迭代器还是原生指针，int\*或者const int \*都可以拥有特化的版本了。





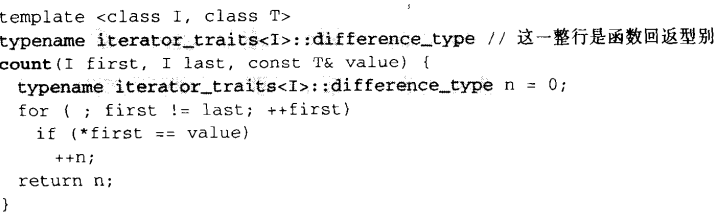
每一个迭代器必须必须遵守约定，自行以内嵌型别定义的方式定义出相应型别，否则就不能兼容于STL这个大家庭

最常用的五种迭代器型别：

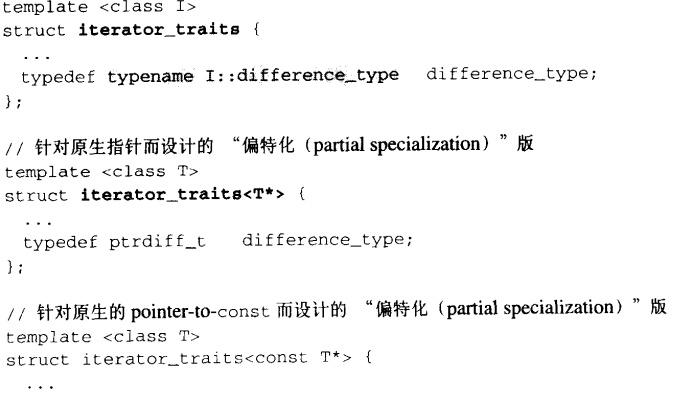


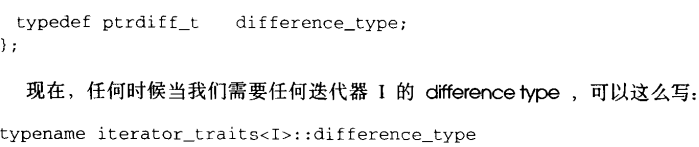
value type：迭代器所指对象的类型，任何一个与STL算法有搭配的类，都应该定义自己的value type内嵌型别

difference type：用来表示两个迭代器之间的距离，可以用来表示一个容器的最大容量，例如为一个泛型算法提供计数功能。



特化版本，针对指针和const的指针，以C++内建的ptrdiff\_t作为原生指针的difference type

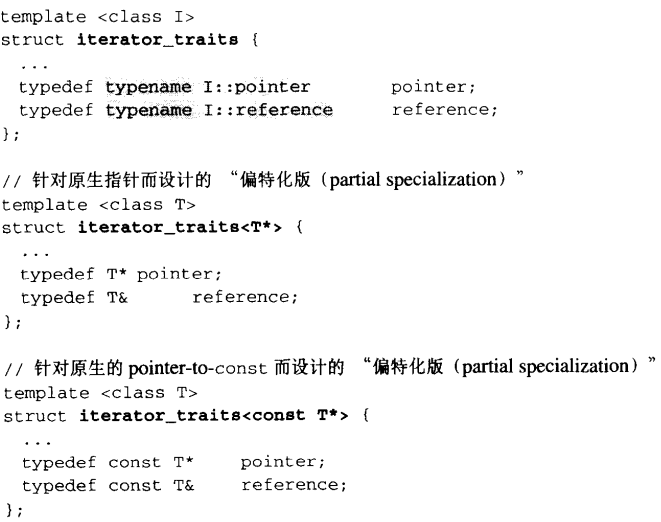




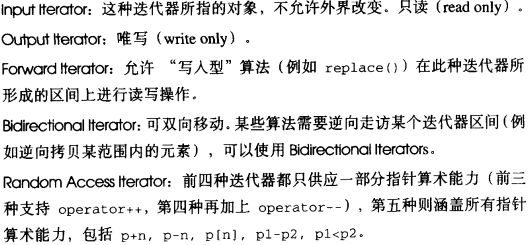
reference type：

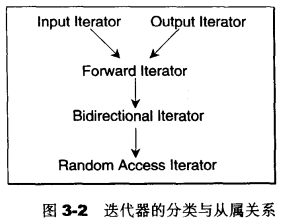
pointer type：返回指针，返回引用



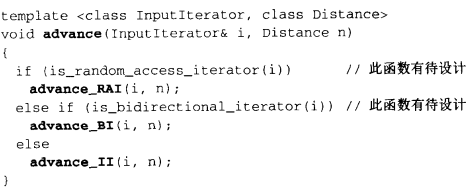


Iterator\_category：首先迭代器分成五类



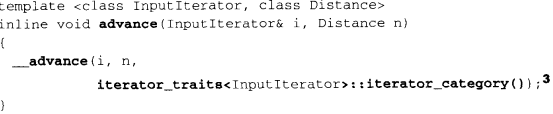


为了效率，应该为合适的算法选择合适的迭代器类型，比如一个算法接受三种迭代器，那可以写成下面这样：



执行期可以确定选择一个最有效率的函数，而函数重载机制可以让程序在编译期就选择正确的版本

例如一个advance函数，实现它的接口：

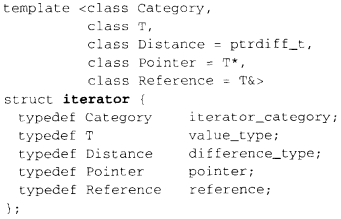


而\_advance就可以用重载来实现，根据最后一个参数，编译器来决定调用哪一个\_advance函数

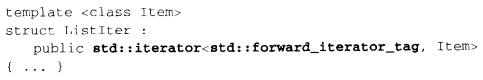
所以迭代器要增加一个类型 iterator\_category



任何一种迭代器都要有这五个内嵌的参数，否则便无法被STL框架承认，STL提供了一个iterators如下：

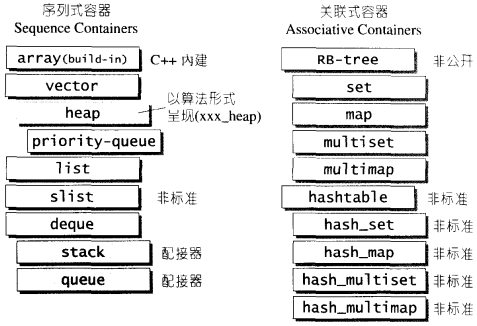


后三个参数都有默认值，自行定义迭代器只需要提供前两个参数即可



第四章：序列式容器

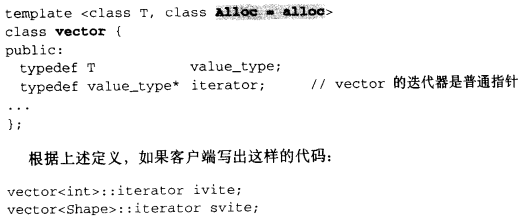
SGI STL的各种容器：



上下包含的关系heap包含vector，priority-queue包含有heap

C++语言本身提供了一个array，STL提供了其他序列式容器，其中stack和queue是把deque改头换面，技术上被归类为adapter

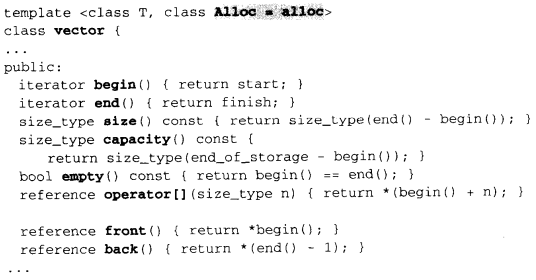
vector：迭代器是指针，因为指针满足所有的vector迭代器的必要条件



指针支持随机存取，所以，vector提供的是随机存取迭代器

vector所采用的数据结构为线性连续空间，两个迭代器start和finish分别指向配置得来的联系空间中已使用的范围，迭代器end\_of\_storage指向整块连续空间的尾端

vector实际配置的大小比客户端需求量更大，以备将来可能的扩充，这就是容量的概念，一个vector的容量永远大于或等于其大小。



缺省使用alloc作为空间配置器，构造函数中使用了空间配置器，为分配的空间赋值，有uninitialized\_fill\_n这样的空间配置器函数的参与。并且会根据第一参数的类型，决定使用算法fill\_n()还是construct()

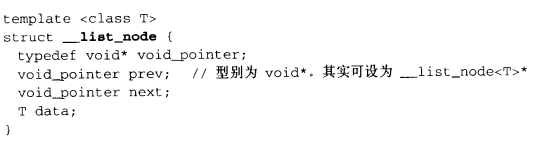
尾端插入：检查是否还有备用空间，有就直接在备用空间上构造，并调整finish，使vector变大，没有就扩充空间

Vector的动态增加大小：并不是在原空间之后接续新空间，而是以原大小的两倍另外配置一块较大的空间，然后将原内容拷贝过来。，然后在原内容之后构造新元素，释放原空间，对于vector的任何操作，一旦引起空间重新配置，原来的迭代器就失效了。分配内存回收内存啥的也都用的是空间配置器。

vector的插入删除等操作也和空间配置器息息相关，在某个位置插入时如果当前的空间不够还要找新的空间

list：每个元素插入或者删除，就配置或者释放一个元素空间，任何元素的插入或者元素移除，list永远是常数时间

list的节点：



list的插入和合并不会使原来的迭代器失效，这在vector是不成立的，因为vector插入可能会导致全部换位置

list数据结构：list是一个环状的双向链表，只要一个指针，就可以表示整个环状双向链表，这个指针指向刻意加载链表尾端的空白节点上

list构造：一个空list含有一个前后指针都指向自己的空节点

list的sort：不能用stl的sort算法，因为其用的是随机读取迭代器，可以用list自身的sort成员函数。用的是快排