空间配置器——空间不一定是内存，也可以是磁盘或其他辅助存储介质

1. 设计简单的空间配置器

_allocate

·set_new_handler 设置函数，为new或new[]操作失败时调用的处理函数，该函数可尝试使更多的空间变为可分配状态，若失败，返回bad_alloc。

·全局函数operator new类似于malloc()，失败则返回空指针

_deallocate

·全局函数operator delete类似于free()

_construct

·定位new的使用，在开辟的空间调用构造函数

_destroy

·显式调用析构函数

rebind

实现分配器以便使一种类型的对象可以为另一种类型的对象分配存储的结构。

如list<int>，实际存储类型为node<int>，通过allocator<int>::rebind<node>()可以创建出用于分配node类型空间的分配器。

即，以node类型为list类型分配空间。

2. 构造和析构基本工具

trivial destrucotr——默认析构函数

non-trival destrucotr——用户自定义的析构函数

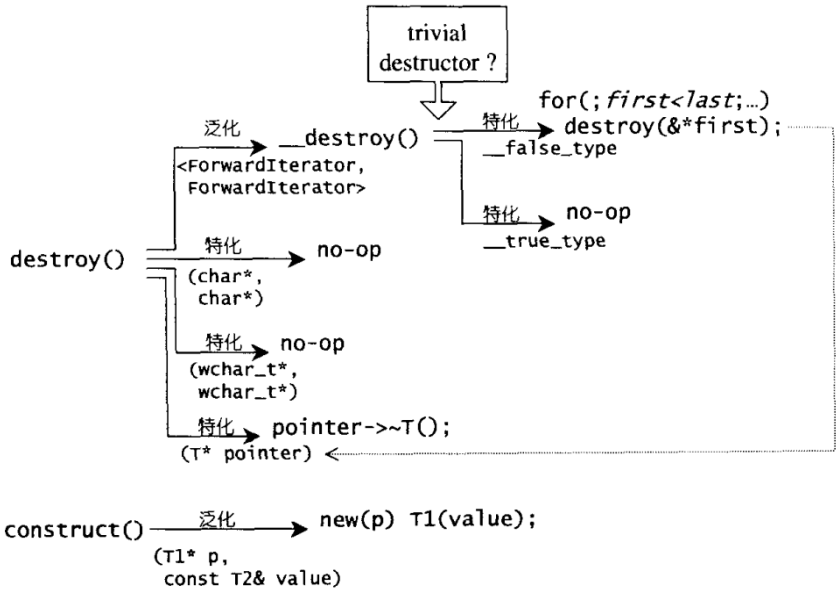

图 **2-1** construct() 和 destroy() 示意

3. 空间的配置与释放

双层级配置器

·第一级(__malloc_alloc_template)：直接使用malloc()和free()；模拟set_new_handler()以处理内存不足的状况。

·第二级(__defalut_alloc_template)：配置区块超过128字节，视为「大」，调用第一级配置器；小于128字节，视为「小」，采用memory pool整理方式。

维护16个自由链表，负责16种小型区块的次配置能力，memory pool 以malloc()配置而得，如果内存不足，转调用第一级配置器。

配置器的开放取决于__USE_MALLOC是否被定义

```
# ifdef __USE_MALLOC
typedef __malloc_alloc_template<0> malloc_alloc;
typedef malloc_alloc alloc;
# else
typedef
__defalut_alloc_template<__NODE_ALLOCATOR_THREADS,0> alloc;
# endif
```

无论alloc被定义为第一级还是第二级，SGI还会再包装一个接口，使配置器接口能符合STL规格。

4. 第一级配置器剖析

分析static void (* set_malloc_handler(void (*f)()))()

由内向外分析，set_malloc_handler有形参列表，形参函数为void(f)()，显然，该参数为函数指针，参数为空，返回类型为void；函数名前有一个*，所以函数返回一个指针，该指针也有参数列表，因此该指针也为函数指针，参数为空，返回类型为void。

综上，set_malloc_handler的参数为void()()，返回类型也为void()()。

·直接使用malloc和realloc

·可以通过static void (*set_malloc_handler(void (*f)()))()设置out-of-memory例程

·失败时，调用设置的例程释放空间，重新配置空间，不断重复。

5. 第二级配置器剖析

配置空间的额外负担——管理内存cookie（索求任何一块内存，都得有一些「税」（记录内存大小的空间）要缴给系统）

次层配置：

每次配置一大块内存，并维护对应之自由链表，下次若有相同大小的内存需求，直接从自由链表中拨出。若客户端释还小额区块，就由配置器会收到自由链表中。

方便管理，小额区块的内存大小上调至8的倍数，维护总共16个自由链表，分别管理8-128字节的小额区块。

free list节点结构：

```
union obj
{
    union obj* free_list_link;
    char client_data[1];                    // 1为举例，可以为8-128中8的倍数
}
```

使用union——一物二用，当不使用该小额区域时，第一个字节指向下一个节点，使用时，直接覆盖即可。

// ROUND_UP() 将bytes上调至8的倍数——(bytes + 7) & ~(7)

allocate流程（逻辑）：

·程序向配置器申请内存

·若需求量大于128字节，交给第一级配置器处理

·配置器将需求量调整至8的倍数，在free lists中寻找可用节点，若找到非空，则返回。

·找到的free list为空，即当前整个free list为空，进入refill函数

·refill函数向内存池默认申请20个节点大小的空间，进入chunk_alloc函数

·若内存池可给的空间大于等于1个，则把第一个给客户端，剩余的重新组建当前大小的free list

·如果内存池空间不够给1个，把当前内存池的空间压榨给某个free list，内存池进行malloc

·如果malloc失败，则把未使用的free list空间释放给内存池，进行重新分配

·直到最后，山穷水尽，调用第一级配置器，使用oom机制，寻找可用内存

·要么成功找到内存，要么抛出异常

6. 内存基本处理工具

STL定义了5个全局函数，除以上用于构造的construct()以及用于析构的destroy()，另外三个函数是uninitialized_copy()，uninitialized_fill()，uninitalized_fill_n()，这三个函数将内存配置和构造函数相结合。全部要借助traits编程技巧以及模板推导机制。

uninitialized_copy接受三个参数

·迭代器first指向输入端的起始位置

·迭代器last指向输入端的结束位置

·迭代器result指向输出端（欲初始化空间）的初始位置

·返回输出端起始位置

uninitalized_fill接受三个参数

·迭代器first指向输出端的起始位置

·迭代器last指向输出端的结束位置
·x表示初值
·返回void

uninitalized_fill_n接受三个参数

·迭代器first指向输出端的起始位置
·n表示欲初始化空间的大小
·x表示初值
·返回输出端起始位置