> 通过剖析SGI STL二级空间配置器内存池源码深入理解其实现原理

SGI STL包含了一级空间配置器和二级空间配置器，其中一级空间配置器allocator采用malloc和free来管理内存，和C++标准库中提供的allocator是一样的，但其二级空间配置器allocator采用了基于freelist自由链表原理的内存池机制实现内存管理。

空间配置器的相关定义

```
template <class _Tp, class _Alloc = __STL_DEFAULT_ALLOCATOR(_Tp) >
class vector : protected _Vector_base<_Tp, _Alloc>
```

可以看到，容器的默认空间配置器是__STL_DEFAULT_ALLOCATOR(_Tp)，它是一个宏定义，如下：

```
# ifndef __STL_DEFAULT_ALLOCATOR
#   ifdef __STL_USE_STD_ALLOCATORS
#     define __STL_DEFAULT_ALLOCATOR(T) allocator< T >
#   else
#     define __STL_DEFAULT_ALLOCATOR(T) alloc
#   endif
# endif
```

从上面可以看到__STL_DEFAULT_ALLOCATOR通过宏控制有两种实现，一种是allocator< T >，另一种是alloc，这两种分别就是SGI STL的一级空间配置器和二级空间配置器的实现。

```
template <int __inst>
class __malloc_alloc_template    // 一级空间配置器内存管理类 -- 通过malloc和free管理内存
```

```
template <bool threads, int inst>
class __default_alloc_template {  // 二级空间配置器内存管理类 -- 通过自定义内存池实现内存管理
```

重要类型和变量定义

```
// 内存池的粒度信息
enum {_ALIGN = 8};
enum {_MAX_BYTES = 128};
enum {_NFREELISTS = 16};
```

```
// 每一个内存chunk块的头信息
union _Obj {
        union _Obj* _M_free_list_link;
        char _M_client_data[1];    /* The client sees this. */
};
```

```
// 组织所有自由链表的数组，数组的每一个元素的类型是_Obj*，全部初始化为0
static _Obj* __STL_VOLATILE _S_free_list[_NFREELISTS];
```

```
// Chunk allocation state. 记录内存chunk块的分配情况
static char* _S_start_free;
static char* _S_end_free;
static size_t _S_heap_size;

template <bool __threads, int __inst>
char* __default_alloc_template<__threads, __inst>::_S_start_free = 0;

template <bool __threads, int __inst>
char* __default_alloc_template<__threads, __inst>::_S_end_free = 0;

template <bool __threads, int __inst>
size_t __default_alloc_template<__threads, __inst>::_S_heap_size = 0;
```

重要的辅助接口函数

```
/*将 __bytes 上调至最邻近的 8 的倍数*/
static size_t  _S_round_up(size_t __bytes)
    { return (((__bytes) + (size_t) _ALIGN-1) & ~((size_t) _ALIGN - 1)); }
```

```
/*返回 __bytes 大小的chunk块位于 free-list 中的编号*/
static  size_t _S_freelist_index(size_t __bytes) {
        return (((__bytes) + (size_t)_ALIGN-1)/(size_t)_ALIGN - 1); }
```

内存池管理函数

```
// 分配内存的入口函数
static void* allocate(size_t __n)

// 负责把分配好的chunk块进行连接，添加到自由链表当中
static void* _S_refill(size_t __n);

// 分配相应内存字节大小的chunk块，并且给下面三个成员变量初始化
static char* _S_chunk_alloc(size_t __size, int& __nobjs);

// 把chunk块归还到内存池
static void deallocate(void* __p, size_t __n);

// 内存池扩容函数
template <bool threads, int inst>
void*
__default_alloc_template<threads, inst>::reallocate(void* __p,
                                                    size_t __old_sz,
                                                    size_t __new_sz);
```