# 面向对象程序设计实验报告

| | |
|---|---|
| 课程名称： | 面向对象程序设计 |
| 作业名称： | STL allocator + memory pool |
| 姓　　名： | 周楠 |
| 学　　号： | 3220102535 |
| 电子邮箱： | 3220102535@zju.edu.cn |
| 联系电话： | 19858621101 |
| 指导教师： | 许威威 |

2024 年 6 月 26 日

浙江大学实验报告

# 一. 实验目的和要求

1. 实现自己的内存分配器
2. 分配器应使用内存池优化内存分配速度
3. 分配器应支持任意内存大小分配请求

# 二. 实验环境

windows11

# 三. 实验流程

## 3.1 类的说明

- memory_pool_table: 静态成员变量，存储所有分配的内存池的指针。初始化为一个空的 vector<void *>
- current_pool: 静态成员变量，指向当前正在使用的内存池。初始化为 nullptr。
- pool_size: 静态成员变量，表示每个内存池的大小，初始化为 4096 * 1024 字节（即 4MB）。
- free_lists: 静态成员变量，表示多个自由列表，用于存储不同大小的已释放内存块。初始化为一个包含 11 个空向量的 vector<vector<void *»。

成员函数：

- allocate(size_t n): 内存分配函数，根据需求分配大小为 n * sizeof(T) 的内存块。如果请求的大小超过 max_size，则使用::operator new 直接分配。否则，从自由列表中获取或从内存池中分配。
- deallocate(T *p, size_t n): 内存释放函数，根据分配时的大小将内存块归还到自由列表或使用::operator delete 直接释放。

- alloc_from_memory_pool(size_t size): 内部函数，从当前内存池中分配大小为 size 字节的内存块。如果当前内存池空间不足，则调用 refill_memory_pool 重新填充内存池。
- refill_memory_pool(): 内部函数，重新填充内存池，分配一个新的 pool_size 大小的内存块作为新的内存池。

## 3.2 Allocate

allocate(size_t n): 内存分配函数，根据需求分配大小为 n * sizeof(T) 的内存块。如果请求的大小超过 max_size，则使用::operator new 直接分配。否则，根据大小找到合适的自由列表，如果列表不为空则从列表中取出，否则从内存池中分配。

1. 计算所需内存大小：根据类型 T 和数量 n 计算总大小 total_size。
2. 检查总大小是否超过最大值：如果超过 max_size，则使用默认的::operator new 分配内存。
3. 查找合适的自由列表：通过二倍增长的方式找到适合的自由列表 list_id。
4. 从自由列表中分配内存：如果找到合适的自由列表且不为空，则从自由列表中取出一个内存块。
5. 从内存池中分配内存：如果自由列表为空，则调用 alloc_from_memory_pool 从内存池中分配内存。

```
1
2    template <class T>
3    T *MyAllocator<T>::allocate(size_t n)
4    {
5        // 计算所需的总大小
6        size_t total_size = sizeof(T) * n;
7        // 如果所需的总大小超过最大值，则使用默认的 new
8        if (total_size > MyAllocator<T>::max_size)
9        {
10           return static_cast<T *>(::operator new(total_size));
11       }
12       // 找到合适的空闲列表，二倍增长的方式
13       size_t list_id = 0;
```

```
        size_t size_ = 4;
        while (size_ < total_size)
        {
            size_ <<= 1;
            list_id++;
        }
        // 如果空闲列表不为空，则从空闲列表中取出一个块，并返回
        if (!free_lists[list_id].empty())
        {
            auto result = free_lists[list_id].back();
            free_lists[list_id].pop_back();
            return static_cast<T *>(result);
        }
        // 如果空闲列表为空，则从内存池中取出一个块，并返回
        return MyAllocator<T>::alloc_from_memory_pool(size_);
    }
```

## 3.3 Deallocate

deallocate(T *p, size_t n): 内存释放函数，根据分配时的大小将内存块归还到自由列表或使用::operator delete 直接释放。

1. 计算总大小：根据类型 T 和数量 n 计算总大小 total_size。
2. 检查总大小是否超过最大值：如果超过 max_size，则使用默认的::operator delete 释放内存。
3. 查找合适的自由列表：通过二倍增长的方式找到适合的自由列表 list_id。
4. 将内存块添加到自由列表：将内存块添加到对应的自由列表中，以便将来重用。

```

    template <class T>
    void MyAllocator<T>::deallocate(T *p, size_t n)
    {
        size_t total_size = n * sizeof(T);
        // 检查总大小是否超过最大值：如果超过 max_size，则使用默认的 ::
            operator delete 释放内存。
```

```
        if (total_size > max_size)
        {
            ::operator delete(p);
        }
        else if (total_size <= 2048)
        {
            size_t list_id = 0;
            size_t size_ = 4;
            // 通过二倍增长的方式找到适合的自由列表 list_id
            while (size_ < total_size)
            {
                size_ <<= 1;
                list_id++;
            }
            free_lists[list_id].push_back(p); // Add to the free list
        }
    }
```

## 3.4 Alloc_from_memory_pool

alloc_from_memory_pool(size_t size): 内部函数，从当前内存池中分配大小为 size 字节的内存块。如果当前内存池空间不足，则调用 refill_memory_pool 重新填充内存池。

1. 检查当前内存池是否有足够空间：如果有足够空间，则从当前内存池分配 size 大小的内存。
2. 更新指针位置：更新 pool_ptr 位置以反映分配情况。
3. 如果内存池空间不足：调用 refill_memory_pool 重新填充内存池，并再次尝试分配。

```

    template <class T>
    T *MyAllocator<T>::alloc_from_memory_pool(size_t size)
    {
        // 如果有足够空间，则从当前内存池分配 size 大小的内存。
```

```
6         if (pool_ptr + size <= pool_size)
7         {
8             char *ptr = static_cast<char *>(current_pool) + pool_ptr;
9             // 更新内存池指针
10            pool_ptr += size;
11            return reinterpret_cast<T *>(ptr);
12        }
13        else
14        {
15            // 如果内存池空间不足：调用 refill_memory_pool 重新填充内存
                  池
16            refill_memory_pool();
17            return alloc_from_memory_pool(size);
18        }
19    }
```

## 3.5 Refill_memory_pool

refill_memory_pool(): 内部函数，重新填充内存池，分配一个新的 pool_size 大小的内存块作为新的内存池。

1. 分配新的内存池：使用::operator new 分配一个新的 pool_size 大小的内存池。
2. 检查分配是否成功：如果分配失败，抛出 std::bad_alloc 异常。
3. 重置指针位置：将 pool_ptr 重置为 0。
4. 将新内存池添加到表中：将新的内存池指针添加到 memory_pool_table。

```
1
2     template <class T>
3     bool MyAllocator<T>::refill_memory_pool()
4     {
5         // 分配新的内存池：使用 ::operator new 分配一个新的 pool_size
              大小的内存池。
6         current_pool = ::operator new(pool_size);
7         if (current_pool == nullptr)
```

```
        {
            // 如果分配失败，则抛出异常
            throw bad_alloc();
        }
        pool_ptr = 0;
        // 将新的内存池添加到内存池表中
        memory_pool_table.push_back(current_pool);
        return true;
    }
```

# 四. 测试结果

1. 在终端中输入 make，执行 makefile 文件，编译生成可执行文件 test.exe。
2. 在终端中输入./test.exe，运行测试程序。


图 1: 测试结果