**C++学习笔记——new和delete**

如无特殊说明，本文中所述的new/delete原则，同样也适用于new[]/delete[]。

**1. new关键字**

new关键字是我们最常用的。

对于如下代码：

T\* pt = new T; // T是任意类型

实际上它依次做了如下三件事：

(1) 调用operator new（后文会详细提到），申请一段堆内存空间，长度等于sizeof(T)。

(2) 内存空间申请成功后，调用T的构造函数，初始化对象。

(3) 返回申请的内存空间的首地址。

如果(1)成功而(2)出现异常，程序会自动调用对应版本的delete关键字（基本等同于operator delete），释放(1)申请的内存空间，然后才会抛出异常。

**2. operator new**

使用operator new，需要包含<new>头文件。

operator new有多个重载版本：

**(1) 默认版本。**功能与1(1)相同，即：申请一段特定长度的堆内存空间，并返回其首地址。这个接口规则与C语言的malloc函数完全相同——事实上其底层正是通过调用malloc来实现的。

默认operator new声明格式：

void\* operator new(std::size\_t size) noexcept(false);

**(2) placement new。**有狭义和广义之分。通常意义上是指狭义，即如下函数声明格式：

void\* operator new(std::size\_t size, void\* at) noexcept;

这个函数声明已经包含在C++的<new>头文件中，可以直接使用。

顾名思义，狭义placement new的含义是，在at这一地址处，申请（或者说划定）一段长度为size的内存空间，并返回at（这段内存空间的首地址）。

这段内存空间可以是（而且通常也是）先前已经被其他new申请过的。placement new的意义在于可以反复利用已有的堆空间资源，从而减少内存资源占用。

示例：使用placement new进行求和运算。

文本

描述已自动生成

广义placement new是指operator new的所有参数个数大于1的重载版本——这些operator new除了必需的size之外，还有其他参数。当然，(3)中即将提到的no-throw new除外。狭义placement new是广义placement new的特例，也是最常用的一种。

**(3) no-throw new**

在C++98中就已不建议使用，只是出于兼容性考虑仍然保留。函数声明：

void\* operator new(std::size\_t size, std::no\_throw\_t& nt) noexcept;

如果没有足够内存空间可分配，且new\_handler无法处理，则会返回空指针（nullptr），而不是按照C++规范，抛出std::bad\_alloc异常。第3条会详细介绍new的异常处理和new\_handler。

**3. new的异常处理**

如果程序可用空间不足，那么new就会出现异常。

这里先介绍new\_handler的概念。new\_handler本身是一个函数指针，其声明格式如下：

typedef void (\*new\_handler) ();

可见，new\_handler函数无参数，且返回值类型为void。new handler是专门用于new异常处理的函数的统称，通常由用户自行定义。

按照C++规范，如果new时可用空间不足，那么程序会调用new\_handler，试图处理该异常；如果一个new\_handler仍无法处理该异常，它可以用新的new\_handler替换自己；如果所有new\_handler都无法处理，或者new\_handler被设置为nullptr，那么程序会抛出std::bad\_alloc异常。注意，此时内存空间并没有被成功申请，所以抛出异常前，程序并不会调用operator delete。

默认的全局new\_handler是nullptr。所以，如果用户没有自定义任何new\_handler，那么当new遇到可用空间不足的情形时，就会立即抛出std::bad\_alloc异常。

示例：使用new\_handler。



程序运行结果：

文本

描述已自动生成