# 线程与进程

**34、线程与进程的区别和联系? 线程是否具有相同的堆栈? dll是否有独立的堆栈?**

答案：进程是死的，只是一些资源的集合，真正的程序执行都是线程来完成的，程序启动的时候操作系统就帮你创建了一个主线程。每个线程有自己的堆栈。 DLL中有没有独立的堆栈，这个问题不好回答，或者说这个问题本身是否有问题。因为DLL中的代码是被某些线程所执行，只有线程拥有堆栈，如果DLL中的代码是EXE中的线程所调用，那么这个时候是不是说这个DLL没有自己独立的堆栈？如果DLL中的代码是由DLL自己创建的线程所执行，那么是不是说DLL有独立的堆栈？以上讲的是堆栈，如果对于堆来说，每个DLL有自己的堆，所以如果是从DLL中动态分配的内存，最好是从DLL中删除，如果你从DLL中分配内存，然后在EXE中，或者另外一个DLL中删除，很有可能导致程序崩溃

## 内存四区

意义在于：赋予其不同的生命周期，给编程带来更大的灵活性

### 运行前

代码区：存放函数体的二进制代码，由操作系统管理

共享的

只读的：防止程序意外修改其指令

全局区：存放全局变量和静态变量以及常量，结束后由系统释放

又叫静态区

全局区还包括常量区（字符串常量，const修饰的全局常量）

### 运行后

栈区：由编译器自动分配和释放，存放函数体的参数值、局部变量等

不能返回局部变量的地址，当离开作用域后，开辟在栈区的局部变量会被编译器自动回收

堆区：由程序员分配和释放，若不释放，程序结束后由操作系统释放

分全局堆和局部堆

全局堆就是所有没有分配的空间，局部堆就是用户分配的空间

堆在操作系统对进程初始化的时候分配，运行过程中也可以向系统要额外的堆

一个C、C++程序编译时内存分为5大存储区：堆区、栈区、全局区、文字常量区、程序代码区。

## 简述C、C++程序编译的内存分配情况

### 静态内存分配

内存在程序编译时就已经分配好，这块内存在程序的整个运行期间都存在。速度快、不容易出错，因为有系统会善后。例如全局变量，静态变量，常量字符串等。

### 从栈上分配

在执行函数时，函数内局部变量的存储单元都在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集中，效率很高，但是分配的内存容量有限。大小为2M。

### 从堆上分配

即动态内存分配

程序在运行的时候用malloc或new申请任意大小的内存，程序员自己负责在何时用free或delete释放内存。动态内存的生存期由程序员决定，使用非常灵活。如果在堆上分配了空间，就有责任回收它，否则运行的程序会出现内存泄漏，另外频繁地分配和释放不同大小的堆空间将会产生堆内碎块。

## 分配函数与释放函数

C：malloc、calloc、realloc/free

C++：new/delete

大多数实现所分配的存储空间比所要求的要稍大一些，额外的空间用来记录管理信息——分配块的长度，指向下一个分配块的指针等等

### [malloc](https://so.csdn.net/so/search?q=malloc&spm=1001.2101.3001.7020" \t "_blank)/free

malloc函数向内存申请一块连续可用的空间

开辟成功则返回一个指向该空间的void\* 型指针，所以需要对其进行强制类型转换，转换成我们想要的类型

开辟失败则返回NULL，所以一定要对malloc的返回值进行检查

free用来释放动态开辟的内存，而不是释放指针

int\* ptr = NULL;

ptr = (int\*)malloc(1000\*sizeof(int));//开辟一千个int大小的内存，并强制类型转换

if(NULL == ptr){

exit(1);

}

free(ptr);

ptr = NULL;

释放只能一次，如果释放两次及两次以上会出现错误

释放空指针例外，释放空指针其实也等于什么都没做，所以释放空指针释放多少次都没有问题

### new/delete

new分配内存步骤

调用operator new函数

调用相应的构造函数构造对象，并传入初值

对象构造完成后，返回一个指向该对象的指针

delete释放内存步骤

调用对象的析构函数

调用operator delete函数释放内存空间

//开辟变量

int\* a = new int(10);

delete a;

//开辟数组

int\* arr = new int[10];

delete[] arr;

## new/delete与malloc/free区别

#### 开辟位置

严格来说，malloc动态开辟的内存在堆区，new开辟的叫做自用存储区

若不重载new操作符，c++编译器一般默认使用堆来实现自用存储，此时等价于堆区

特别：new可以不为对象分配内存

#### 重载

new、delete是操作符，可以重载，只能在C++中使用

malloc、free是函数，可以覆盖，C、C++中都可以使用

#### 是否调用构造与析构函数

new可以调用对象的构造函数，对应的delete调用相应的析构函数。malloc仅仅分配内存，free仅仅回收内存，并不执行构造和析构函数

#### 是否需要指定内存大小

malloc需要显式指出开辟内存的大小，new无需指定，编译器会自动计算

#### 返回值类型

new返回的是某种数据类型指针，malloc返回的是void 指针，new比malloc更安全

new内存分配失败时，会抛出bac\_alloc异常，不会返回NULL；malloc开辟内存失败会返回NULL指针，所以需要判断

## calloc、[realloc](https://so.csdn.net/so/search?q=realloc&spm=1001.2101.3001.7020" \t "_blank)

calloc(number,size):为number个大小为size的元素开辟一块空间，并把每个字节初始化为0

realloc(内存地址，大小)：用于调整申请的空间大小，扩容

## C++中释放互换

在C++中，使用malloc申请的内存能否通过delete释放？使用new申请的内存能否用free？

不能，malloc/free主要为了兼容C，new和 delete 完全可以取代malloc/free的。malloc/free的操作对象都是必须明确大小的。而且不能用在动态类上。new和delete会自动进行类型检查和大小，malloc/free不能执行构造函数与析构函数，所以动态对象它是不行的。当然从理论上说使用malloc申请的内存是可以通过delete释放的。不过一般不这样写的。而且也不能保证每个C++的运行时都能正常