C/C++内存管理

1. C/C++内存管理



①栈又叫堆栈，非静态局部变量/函数参数/返回值等等，栈是向下增长的。

②内存映射段是高效的I/O映射方式，用于装载一个共享的动态内存库。用户可使用系统接口创建共享共

享内存，做进程间通信。（Linux课程如果没学到这块，现在只需要了解一下）

③堆用于程序运行时动态内存分配，堆是可以上增长的。

④数据段--存储全局数据和静态数据。

⑤代码段--可执行的代码/只读常量。

2.C语言中的内存管理

malloc/calloc/realloc

1. C++内存管理

C语言内存管理方式在C++中可以继续使用，但有些地方就无能为力而且使用起来比较麻烦，因此C++又提出了自己的内存管理方式：通过new和delete操作符进行动态内存管理。

* 1. new/delete管理内置类型

int main()

{

int\* k = new int(8);

cout << \*k << endl;

delete k;

return 0;

}

看看

* 1. new/delete管理自定义类型

注意：在申请自定义类型的空间时，new会调用构造函数，delete会调用析构函数，而malloc与free不会。

1. operator new与operator delete函数

new和delete是用户进行动态内存申请和释放的操作符，operator new 和operator delete是系统提供的全局函数，new在底层调用operator new全局函数来申请空间，delete在底层通过operator delete全局函数来释放空间。

5. new和delete的实现原理

5.1 内置类型

如果申请的是内置类型的空间，new和malloc，delete和free基本类似，不同的地方是：new/delete申请和释放的是单个元素的空间，new[]和delete[]申请的是连续空间，而且new在申请空间失败时会抛异常，malloc会返回NULL。

5.2自定义类型

new的原理：

1. 调用operator new函数申请空间

2. 在申请的空间上执行构造函数，完成对象的构造

delete的原理：

1. 在空间上执行析构函数，完成对象中资源的清理工作

2. 调用operator delete函数释放对象的空间

new T[N]的原理：

1. 调用operator new[]函数，在operator new[]中实际调用operator new函数完成N个对象空间的申请

2. 在申请的空间上执行N次构造函数

delete[]的原理：

1. 在释放的对象空间上执行N次析构函数，完成N个对象中资源的清理

2. 调用operator delete[]释放空间，实际在operator delete[]中调用operator delete来释放空间

1. 定位new表达式(placement-new)

5.1定位new表达式是在已分配的原始内存空间中调用构造函数初始化一个对象。

5.2使用格式：

new (place\_address) type或者new (place\_address) type(initializer-list)

place\_address必须是一个指针，initializer-list是类型的初始化列表

5.3 使用场景：

定位new表达式在实际中一般是配合内存池使用。因为内存池分配出的内存没有初始化，所以如果是自定义类型的对象，需要使用new的定义表达式进行显示调构造函数进行初始化。

1. 设计一个类，该类只能在堆上创造对象

方法一：构造函数私有化

1. 将类的构造函数私有，拷贝构造声明成私有。防止别人调用拷贝在栈上生成对象。

2. 提供一个静态的成员函数，在该静态成员函数中完成堆对象的创建

class A

{

public:

static A\* create(int k)

{

return new A(k);

}

static A\* create()

{

return new A;

}

private:

A(){}

A(int k)

: \_k(k)

{}

A(const A& a);

private:

int \_k;

};

方法二：将析构函数私有化

7.3 请设计一个类，该类只能在栈上创建对象

方法一：构造函数私有化

方法二：只能在栈上创建对象，即不能在堆上创建，因此只要将new的功能屏蔽掉即可，即屏蔽掉operator new和定位new表达式，注意：屏蔽了operator new，实际也将定位new屏蔽掉。

7.4 单例模式

单例模式：

一个类只能创建一个对象，即单例模式，该模式可以保证系统中该类只有一个实例，并提供一个访问它的全局访问点，该实例被所有程序模块共享。比如在某个服务器程序中，该服务器的配置信息存放在一个文件中，这些配置数据由一个单例对象统一读取，然后服务进程中的其他对象再通过这个单例对象获取这些配置信息，这种方式简化了在复杂环境下的配置管理。

7.4.1 饿汉模式

就是说不管你将来用不用，程序启动时就创建一个唯一的实例对象。

优点：简单

缺点：可能会导致进程启动慢，且如果有多个单例类对象实例启动顺序不确定。

7.4.2 懒汉模式

如果单例对象构造十分耗时或者占用很多资源，比如加载插件啊， 初始化网络连接啊，读取文件啊等等，而有可能该对象程序运行时不会用到，那么也要在程序一开始就进行初始化，就会导致程序启动时

非常的缓慢。 所以这种情况使用懒汉模式（延迟加载）更好。

优点：第一次使用实例对象时，创建对象。进程启动无负载。多个单例实例启动顺序自由控制。

缺点：复杂

8．内存泄漏

8.1 什么是内存泄漏

内存泄漏指因为疏忽或错误造成程序未能释放已经不再使用的内存的情况。内存泄漏并不是指内存在物理上的消失，而是应用程序分配某段内存后，因为设计错误，失去了对该段内存的控制，因而造成了内存的浪费。

8.2内存泄漏分类

8.2.1堆内存泄漏(Heap leak)

堆内存指的是程序执行中依据须要分配通过malloc / calloc / realloc / new等从堆中分配的一块内存，用完后必须通过调用相应的 free或者delete 删掉。假设程序的设计错误导致这部分内存没有被释放，那么以后这部分空间将无法再被使用，就会产生Heap Leak。

8.2.2系统资源泄漏

指程序使用系统分配的资源，比方套接字、文件描述符、管道等没有使用对应的函数释放掉，导致系统资源的浪费，严重可导致系统效能减少，系统执行不稳定。