1.5 堆区内存管理 new 和 delete 运算符_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 1.5 堆区内存管理 new 和 delete 运算符涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

5. 堆区内存管理 new 和 delete 运算符

```
void *malloc(size_t size);
void free(void *ptr);
```

1.只分配内存，不初始化

```
格式:type  * p_var = new type;
例如:int * a = new int;
```

2.分配内存，并且进行初始化

```
格式:type  * p_var = new type(value);
例如:int * a = new int(8);
```

3.释放内存

```
格式:delete val_ptr;
例如:delete a;
```

3.new[] /delete[] 多个内存的分配与释放

1.只分配内存，不初始化

```
格式:type * p_var = new type [size];
例如:int  * pa = new int[3];
```

2.分配内存，并且进行初始化

C++98 标准规定，new 创建的对象数组不能被显式初始化， 数组所有元素被缺省初始化。

如果数组元素类型没有缺省初始化（默认构造函数），则编译报错。

但 C++11 已经允许显式初始化:

```
格式:type  * p_var = new type[size]{value1,value2,...};
```

例如:int * pa = new int[3] {1,2,3};

3.释放内存

格式:delete [] val_ptr;

例如:delete [] a;

1. 有了 malloc/free 为什么还要 new/delete?

- malloc 与 free 是 C++/C 语言的标准库函数，new/delete 是 C++ 的运算符。它们都可用于申请动态内存和释放内存。
- 对于非内部数据类型的对象而言，光用 malloc/free 无法满足动态对象的要求。对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数 **。**
- 由于 malloc/free 是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于 malloc/free。因此 C++ 语言需要一个能完成动态内存分配和初始化工作的运算符 new，以及一个能完成清理与释放内存工作的运算符 delete

2. 既然 new/delete 的功能完全覆盖了 malloc/free，为什么 C++ 不把 malloc/free 淘汰出局呢?

这是因为 C++ 程序经常要调用 C 函数，而 C 程序只能用 malloc/free 管理动态内存。

3.malloc/free 与 new/delete 之间的区别?

- 他们都是动态管理内存的入口
- malloc/free 是 C/C++ 标准库的函数，new/delete 是 C++ 运算符
- malloc/free 只是动态分配内存空间 / 释放内存空间，而 new/delete 除了分配空间还会调用构造函数进行初始化，析构函数进行资源释放
- malloc/free 需要手动计算类型大小且返回值为 void *, new/delete 可自己计算类型的大小, 返回对应类型得指针
- new/delete 的底层调用了 malloc/free
- malloc/free 申请后得判空，new/delete 则不需要
- new 直接跟类型，malloc 跟字节个数

在堆区分配 10 个字节的内存空间，将这 10 个字节的内存空间数据写成: 0x00 ,0x11,0x22, …0X99

并在屏幕上输出。要求如下:

- 设计一个函数分配内存，通过参数带回分配内存的首地址
- 设计一个函数完成数据写入
- 设计一个函数完成数据输出