# new和delete

## 引言

C++中的动态内存分配是在程序运行时根据需要进行内存空间的分配和释放，与静态内存分配相比，具有更大的灵活性和适应性。动态内存分配可以通过new和delete或者malloc()和free()等函数实现。

动态内存分配通常用于以下情况：

1. 在程序运行时无法确定所需的内存大小。
2. 需要在多个函数之间共享数据，并且数据可能被频繁地创建和销毁。
3. 静态数组无法满足需求，例如需要支持可变长度数组。

如果在动态分配内存后忘记了释放它，则会导致内存泄漏。为避免内存泄漏，可以使用智能指针或手动管理内存的方式来释放动态分配的内存。

## new和delete的使用

### new 运算符

new 运算符用于在堆内存（动态存储区）上动态分配单个对象的内存空间，new开辟的空间在堆上，而一般声明的变量存放在栈上。

通常来说，当在局部函数中new出一段新的空间，该段空间在局部函数调用结束后仍然能够使用，可以用来向主函数传递参数。另外需要注意的是，new的使用格式，new出来的是一段空间的首地址。所以一般需要用指针来存放这段地址。

new 运算符会执行以下操作：

1. 分配足够的内存来容纳指定类型的对象。
2. 调用适当的构造函数来初始化对象。
3. 返回指向新分配对象的指针。

使用语法：

T\* pointer = new T;

其中 T 是要分配内存的类型，pointer 是指向新分配的对象的指针。

使用示例

分配单个对象：

int\* p = new int; // 分配一个int类型的内存空间

\*p = 10; // 使用分配的内存

分配数组：

int\* arr = new int[10]; // 分配一个包含10个int类型元素的数组

arr = 1; // 使用数组

初始化对象（对于类）：

class MyClass {

public:

MyClass(int val) : value(val) {}

private:

int value;

};

MyClass\* obj = new MyClass(10); // 分配并初始化MyClass对象

### delete 运算符

delete 运算符用于释放之前使用 new 运算符分配的内存空间。

delete 运算符会执行以下操作：

1. 调用适当的析构函数来销毁对象。
2. 释放由 new 运算符分配的内存空间。

注意：只能使用delete运算符释放使用new运算符分配的内存，并且确保 pointer 指向动态分配的对象。

使用语法：

delete pointer;

其中 pointer 是要释放的对象的指针。

使用示例

释放单个对象：

delete p; // 释放p指向的内存

p = nullptr; // 将指针置为空，避免悬挂指针

释放数组：

delete[] arr; // 释放arr指向的数组内存

arr = nullptr; // 将指针置为空

释放对象（对于类）：

delete obj; // 释放obj指向的对象内存，并调用析构函数

obj = nullptr; // 将指针置为空

## 使用实例

### 实例1

using namespace std;

int main ()

{

float\* pvalue = NULL;

// 初始化为 null 的指针

pvalue = new float;

// 为变量请求内存

\*pvalue = 29.0;

// 在分配的地址存储值

cout << "变量值: " << \* pvalue << endl;

delete pvalue;

// 释放内存

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

变量值: 29

### 实例2

对象与简单的数据类型的操作没有什么不同，参考以下代码。

#include <iostream>

using namespace std;

class Box

{

public:

Box()

{

cout << "调用构造函数！" << endl;

}

~Box()

{

cout << "调用析构函数！" << endl;

}

};

int main()

{

Box\* myBoxArray = new Box; // 为变量请求内存

cout << "中间操作" << endl;

delete myBoxArray; // 释放内存

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

调用构造函数！

中间操作

调用析构函数！

### 数组的动态内存分配实例

一维数组

// 动态分配，数组长度为 m

int \*array=new int[m];

//释放内存

delete [] array;

二维数组

int \*\*array;  
// 假定数组第一维长度为 m，第二维长度为 n

// 动态分配空间

array = new int \*[m];

for(int i=0; i<m; i++)

{

array[i] = new int [n];  
}

//释放

for(int i=0; i<m; i++)

{

delete [] array[i];

}

delete [] array;

## malloc/free和new/delete的区别

### 基本属性：

new、delete是C++中的关键字操作符需要编译器支持，而malloc和free是标准库函数，需要加入头文件stdlib.h。

### 使用上的区别

1. malloc申请内存空间时需要显式填入申请内存的大小。new会根据new的类型来分配内存，所以无需显式填入申请的内存大小。
2. malloc申请的空间不会初始化，new可以初始化。
3. malloc内存分配成功时返回void \* ，需要通过强制类型转换，将void\*指针转换成我们需要的类型。new操作符内存分配成功时，返回的是对象类型的指针，类型严格与对象匹配，无须进行类型转换，所以new是符合类型安全性的操作符，在C++程序中使用new会比malloc安全可靠。
4. malloc分配内存失败时返回NULL，我们可以通过返回值判断内存是否分配成功。new内存分配失败时，不会返回NULL，而是抛出异常，如果不捕捉异常，那么程序就会异常退出。
5. 申请自定义类型对象时，malloc/free只会开辟空间，不会调用构造函数与析构函数，而new在申请空间后会调用构造函数完成对象的初始化，delete在释放空间前会调用析构函数完成空间中资源的清理。
6. new/delete比malloc和free的效率稍微低点，因为new/delete的底层封装了malloc/free

### 语言特性

1. 是否可以重载：在C++中new和delete符号是可以重载的，所以可以重新实现new的实现代码，让其分配的内存位置在静态存储区等。而malloc和free是C里的库函数，无法对其进行重载。
2. 是否可以动态扩充内存：使用malloc分配内存后，如果发现内存不够用，可以通过realloc函数来扩充内存大小。而new没有扩充内存的机制。

## 总结

以上就是今天要讲的内容，后续会有更多内容。

## 参考资料

版权声明：本文参考了其他资料和CSDN博主的文章，遵循CC 4.0 BY-SA版权协议，现附上原文出处链接及本声明。

1. https://blog.csdn.net/m0\_53591689/article/details/125739218
2. https://blog.csdn.net/Yinrtyu\_/article/details/145259581