Advanced C++

Memory Management

Йордан Зайков

Димитър Трендафилов

Memory Management

Effective C++ 2nd ed: 3, 5, 7, 8, 9, 10

More Effective C++: 1, 3, 8

Exceptional C++: 35, 36

Какво всъщност прави new?

Грубо казано:

- Заделя памет

  *void * operator new(size_t size);*

- Вика конструктор върху нея (не важи за вградените типове)

Пример

*string *ps = new string("Memory Mngmnt");*

=>

// get raw memory for a string object

*void *memory = operator new(sizeof(string));*

// initialize the object in the memory

*call string::string("Memory Mngmnt") on *memory;*

// make *ps* point to the new object

*string *ps = static_cast<string*>(memory);*

Предпочитайте *new / delete* пред *malloc / free*

Пример:

- *string s1 = malloc(10 * sizeof(string));*
- *string s2 = new string[10];*

Как ще инициализирате обектите, сочени от *s1*? А после как ще ги деструкторирате?

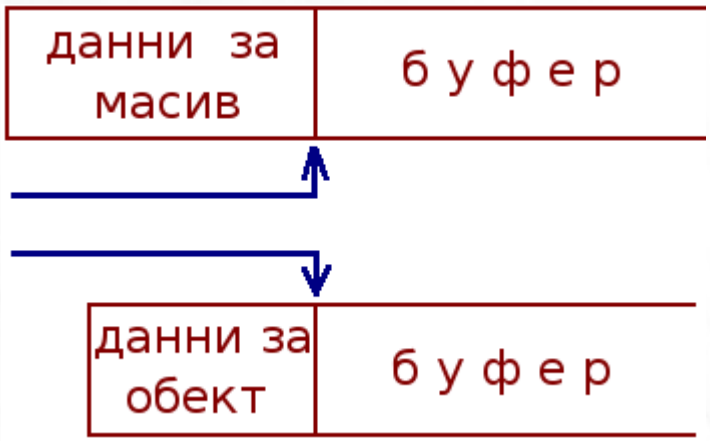
- Никога не смесвайте *new* с *free* или *malloc* с *delete*!

Не смесвайте *new* с *delete [ ]* и *new [ ]* с *delete*

delete / *delete [ ]* приемат за аргументи само един указател. Как да решат колко памет да освободят (и колко деструктора да викнат)?

*T * p = new T[n];*

*T * p = new T;*



Ами ако няма достатъчно памет?

- *operator new* хвърля *std::bad_alloc*,
  не връща *NULL*
- Но преди това вика Ваша функция за обработка на грешки:

  *typedef void (****new_handler****)();*

  new_handler ***set_new_handler****(new_handler p) throw();*

  - Дефинирани са в *<new>*
- Така не се налага да пишете *try* / *catch* всеки път

Глупав пример

```
void noMoreMemory()
{
    cerr << "Unable to satisfy request for memory\n"; //
    abort();
}

int main()
{
    std::set_new_handler(noMoreMemory);
    int *pBigDataArray = new int[1000000000L];
    ...
```

Цялата истина за *::operator new*

```
void * operator new (size_t size) {
    if (size == 0) size = 1;

    while (1) {
        attempt to allocate <size> bytes;

        if (the allocation was successful)
            return (a pointer to the memory);

        new_handler globalHandler = set_new_handler(0);
        set_new_handler(globalHandler);

        if (globalHandler) (*globalHandler)();
        else throw std::bad_alloc();
    }
}
```

new_nandler трябва да прави едно от следните:

- Да увеличава количеството налична памет
- Да задава различен манипулатор на *new*
- Да отменя манипулатора на *new*
- Да хвърля *std::bad_alloc* или негов наследник
- Да прекъсва изпълнението на програмата

operator new се наследява

```
class Base {
public:
    static void * operator new (size_t size);
    ...
};

// Derived не предефинира operator new
class Derived: public Base
{ ... };

// Извиква се Base::operator new !
Derived *p = new Derived;
```

Как да постъпим?

- Правете проверка за коректност на големината във Вашите версии на operator new:

```
void * Base::operator new(size_t size)
{
    if (size != sizeof(Base))
        return ::operator new(size);

    … // Вашият operator new идва тук
}
```

placement new

- Не заделя памет, а само конструира обект върху вече заделена памет.
- Синтаксис:
  - *new (location) class_name;*
  - *new (location) class_name [count];*
- *operator new* на *placement new*:

```
void * operator new (size_t, void * location)
{
        return location;
}
```

Избягвайте да скривате нормалната форма на *operator new*

- Как пък така да я “скрия”?!
- Пример:

```
class C {
public:
    static void * operator new(size_t size, new_handler p);
    ...
};

C * px1 = new (specialErrorHandler) C;    // добре
C * px2 = new C;                          // грешка
```

Как да постъпим?

- Използвайте делегация

```
static void * operator new(size_t size)
{
    return ::operator new(size);
}
```

- Или подразбиращи се параметри

```
static void *
operator new(size_t size, new_handler p = 0);
```

operator delete()

```
void Base::operator delete(void *rawMemory, size_t size)
{
    if (rawMemory == 0) return;

    if (size != sizeof(Base)) {
        ::operator delete(rawMemory);
        return;
    }

    deallocate the memory pointed to by rawMemory;

    return;
}
```

```
class C {
public:
        C(int _x) : x(_x) { std::cout << x << std::endl; }
        ~C() { std::cout << "~" << x << std::endl; }
        int x;
};

C * p = static_cast<C*>(operator new(2 * sizeof(C)));
new (p) C(3);                                          //   3
new (p+1) C(5);                                        //   5

p[1].~C();                                             //  ~5
p[0].~C();                                             //  ~3
operator delete(p);
```

Даден е следният код:

```
class B {
public:
    virtual ~B();
    void operator delete  ( void*, size_t ) throw();
    void operator delete[ ] ( void*, size_t ) throw();
    void f( void*, size_t ) throw();
};

class D : public B {
public:
    void operator delete  ( void* ) throw();
    void operator delete[ ] ( void* ) throw();
};
```

Кой *operator delete* ще се извика?

```
D* pd1 = new D;
delete pd1;

B* pb1 = new D;
delete pb1;

D* pd2 = new D[10];
delete[ ] pd2;

B* pb2 = new D[10];
delete[ ] pb2;
```

Още няколко съвета

- Предоставите ли една от *new* (*new [ ]*) или *delete* (*delete [ ]*), предоставяйте и другата!
- Винаги декларирайте *operator new()* и *operator delete()* като статични функции (те винаги са такива)!
- Никога не третирайте масивите полиморфично!

Custom Allocators

- https://molecularmusings.wordpress.com/2012/08/14/memory-allocation-strategies-a-linear-allocator/
- http://www.codingwisdom.com/codingwisdom/2012/09/you-need-a-frame-allocator.html
- https://github.com/emeryberger/Heap-Layers
- https://github.com/emeryberger