



C++期末不挂科

CH6 - 类动态内存分配

本章内容

- 回顾 malloc / free
- 对象的动态内存分配: new / delete
- 两者异同
- 虚析构函数

理解以下名词：

- 程序内存模型
- 堆区与栈区
- 内存泄露
- 内存溢出
- 虚析构函数

思考并回答以下问题：

- malloc/free和new/delete的区别与联系
- 内存泄露和内存溢出的关系
- new/delete一个对象具体发生哪些事情
- 为什么父类的析构函数要声明为虚函数

熟悉以下代码过程：

- new/delete动态创建/销毁对象
- new[]/delete[]动态创建/销毁对象数组
- 使用虚析构函数避免对象资源泄露

回顾malloc / free

面向对象动态内存分配: new / delete

两者异同

虚析构函数

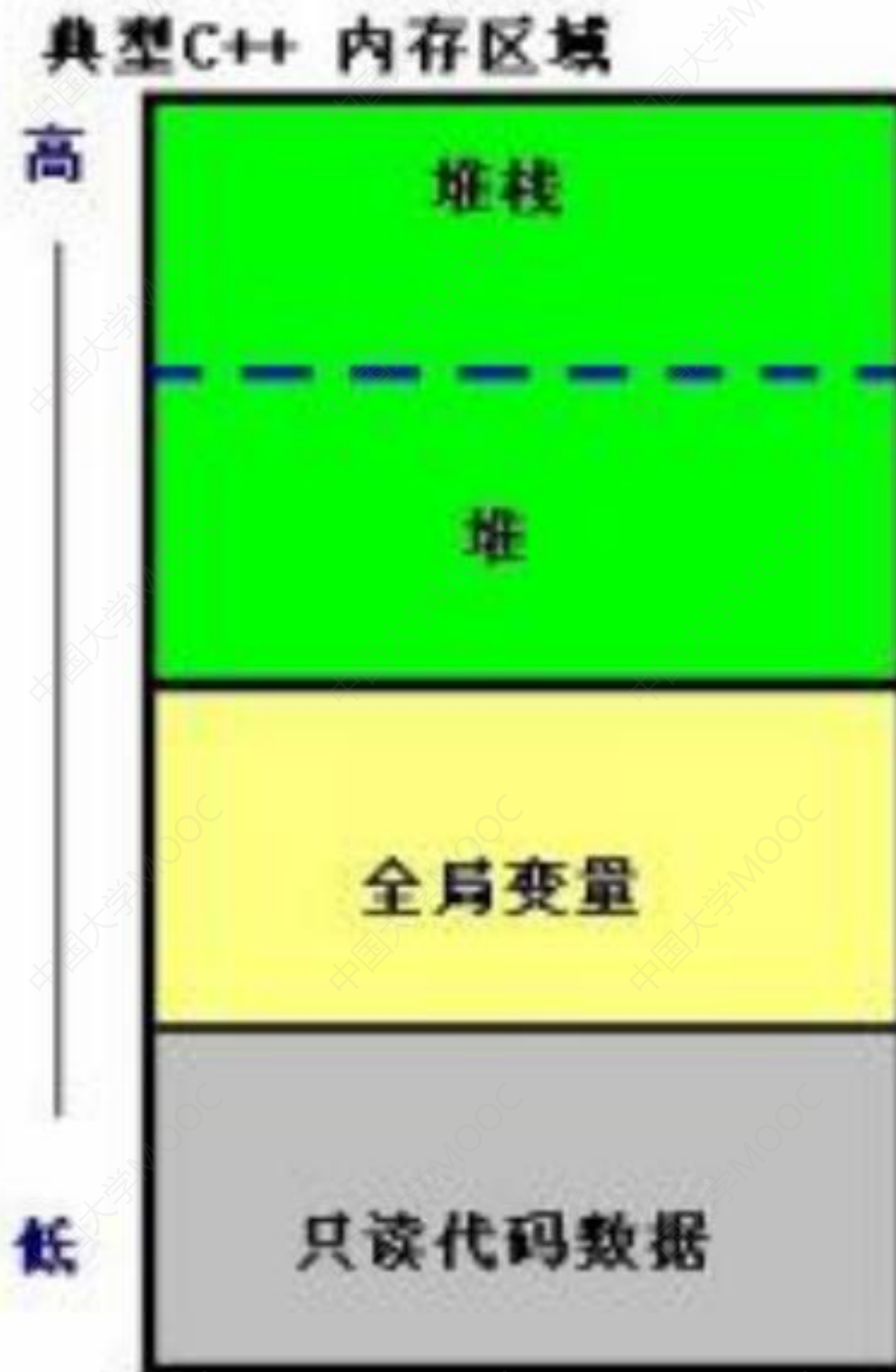
内存 Memory



地址	十六进制数据																ASCII
0275FFC8	C3	E2	00	00	1C	E8	00	00	5F	ED	00	00	4E	F1	00	00	Ãâççèççíççñçç
0275FFD8	AB	F6	00	00	1C	FC	00	00	B2	01	01	00	47	06	01	00	«öççüçç††çç†ç
0275FFE8	C1	0B	01	00	75	0F	01	00	C4	14	01	00	38	1A	01	00	À†çç†ç†ç†ç†ç
0275FFF8	A0	1F	01	00	F6	24	01	00	82	2A	01	00	00	30	01	00	■†çç\$†ç,*†çç†ç
02760008	99	35	01	00	EC	3A	01	00	31	40	01	00	99	45	01	00	™5†ç†ç†ç1†ç™E†ç
02760018	6A	49	01	00	79	4D	01	00	A7	51	01	00	C9	55	01	00	jI†çyM†ççççE†ç
02760028	EA	59	01	00	09	5E	01	00	29	62	01	00	65	66	01	00	êY†ç^†ç)b†çef†ç
02760038	C8	6A	01	00	7F	6F	01	00	A2	73	01	00	32	77	01	00	Èj†çço†ççs†ç2w†ç
02760048	36	78	01	00	38	79	01	00	95	79	01	00	85	7A	01	00	6x†ç8y†ççy†ççz†ç
02760058	45	7B	01	00	A5	7B	01	00	AF	7C	01	00	6E	7D	01	00	E{†ç*†ç†ç†ç†ç†ç
02760068	4E	C1	01	00	40	C2	01	00	40	C7	01	00	40	80	01	00	N†çç†çç†çç†ç†ç
02760078	48	80	01	00	42	C1	01	00	40	CB	01	00	43	80	01	00	H†çç†çç†çç†ç†ç
02760088	44	80	01	00	43	80	01	00	40	CC	01	00	C3	01	00	41	D†ççç†çç†çç†çç
02760098	80	01	00	40	80	01	00	40	80	01	00	40	C1	01	00	40	†çç†çç†çç†çç†ç
027600A8	CE	01	00	42	80	01	00	43	C3	01	00	40	80	01	00	41	†ççç†çç†çç†çç
027600B8	CD	01	00	C4	01	00	40	C1	01	00	41	80	01	00	40	D1	†çç†çç†çç†çç†ç
027600C8	01	00	CA	01	00	40	D1	01	00	DD	01	00	DD	01	00	DD	†çç†çç†çç†çç†ç
027600D8	01	00	DC	01	00	80	00	04	DA	01	00	C2	00	04	CC	01	†çç†çç†çç†çç†ç

- 计算机中一切正在运行的程序都运行在内存中
- 程序(代码、数据)都加载在内存中

程序内存模型



栈
堆

变量、对象、调用函数等所需要的空间由编译器管理，
来一个就生成一个压入栈中，结束一个就弹出一个。

动态内存分配

由程序员自己编写代码管理：**申请**，使用，**释放**。
编译器不维护。

动态内存分配

- 需要时申请
- 用完后归还

- 告诉操作系统你要多大的空间（字节数）
- 申请成功给你一个地址（指针）

malloc / free

C语言: malloc / free

- malloc申请堆区内存:

```
int* ptr = (int*)malloc(sizeof(int) * 3); // 获得指定字节数的堆区内存
```

```
*(ptr + 2) = 10; // 使用这块内存
```

- free释放占用的内存:

```
free(ptr);
```

堆区

...

		10

...

如果不释放内存空间 => 内存泄露 => 导致内存不够 => 内存溢出

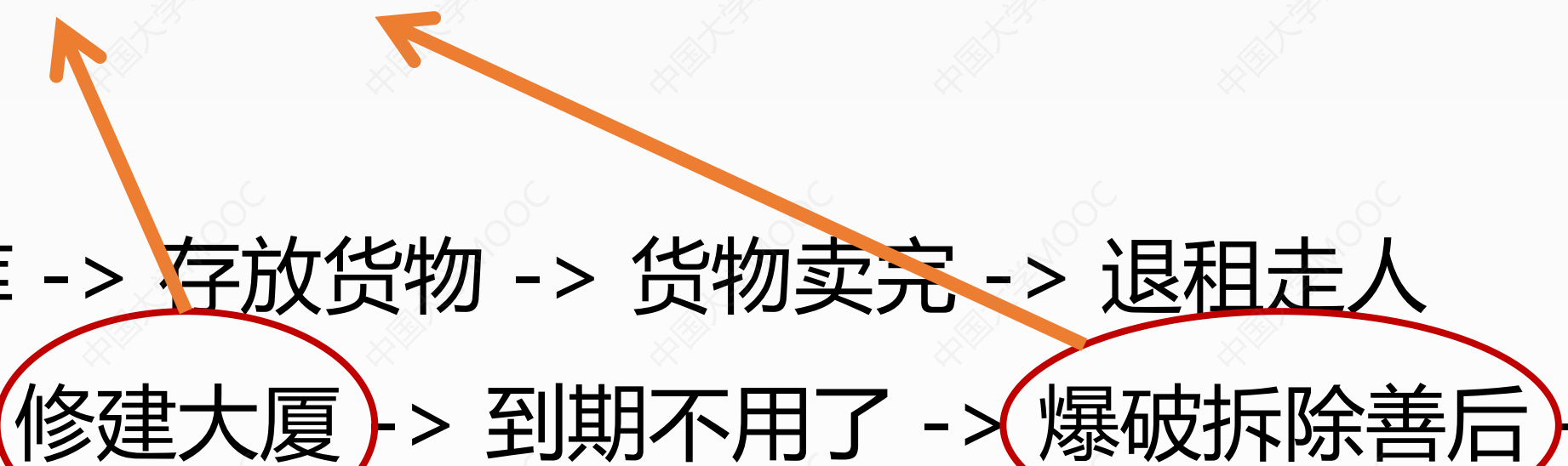
回顾malloc / free

面向对象动态内存分配: new / delete

两者异同

虚析构函数

C++的动态内存分配

- 思考：C++相比C语言主要多了什么？面向对象。
 - 进一步思考：定义一个对象和定义一个普通变量有何区别？
 - 普通变量：分配足够空间即可存放数据
 - 对象：除了需要空间，还要**构造 / 析构**
 - 类比：
 - malloc / free: 租用一个仓库 -> 存放货物 -> 货物卖完 -> 退租走人
 - **new / delete**: 租一块地 -> **修建大厦** -> 到期不用了 -> **爆破拆除善后** -> 地归原主
- 

new 和 delete 运算符

- **new / delete:**

申请所需大小的内存 -> 构造对象 -> 返回指针供使用 -> 析构对象 -> 释放空间

```
class A {  
public:  
    int n;  
    A() { cout << "Ctr1" << endl; }  
    A(int n) : n(n) { cout << "Ctr2" << endl; }  
    void func() { cout << "func called" << endl; }  
    ~A() { cout << "Dtr" << endl; }  
};
```

```
A* p1 = new A;  
p1->func();  
delete p1;  
A* p2 = new A(10);  
p2->func();  
delete p2;
```

```
Ctr1  
func called  
Dtr  
Ctr2  
func called  
Dtr
```

new[] 和 delete[] 运算符

- new[]: 申请空间并批量构造对象数组, 返回**首元素指针**
- delete[]: 批量析构对象数组并释放空间, 只能用于new[]出来的指针

```
class A {  
public:  
    A() { cout << "A constructed\n"; }  
    void func() { cout << "func called\n"; }  
    ~A() { cout << "A destructed\n"; }  
};  
// ...  
  
A* p_a = new A[6];  
(p_a + 1)->func();  
delete[] p_a;
```

```
A constructed  
A constructed  
A constructed  
A constructed  
A constructed  
A constructed  
func called  
A destructed  
A destructed  
A destructed  
A destructed  
A destructed  
A destructed
```


回顾malloc / free

面向对象动态内存分配: new / delete

两者异同

虚析构函数

malloc / free VS new / delete

同

- 都用于动态内存分配（申请 - 使用 - 释放）
- 都发生在堆区内存上
- 申请成功得到的都是指针，通过指针操作得到的内存空间/对象

异

-
- malloc / free 是一对函数，new / delete 是一对**运算符**
 - malloc / free 仅申请空间(不关心空间用途)，而 new / delete 还会构造/析构对象
 - malloc / free 得到 void* 指针，而 new / delete 得到指向指定类型的指针
 - **new / delete 底层有使用 malloc / free**

回顾malloc / free

面向对象动态内存分配: new / delete

两者异同

虚析构函数

损失 ^{增大了可能性} → 不够

内存泄漏 VS 内存溢出

- 内存**溢出**:

一辆车只能坐5个人，来第6个人没法坐了 → 内存空间不够用了

- 内存**泄漏**:

公司一共有10辆车，本来最多能坐50人

小明把其中一辆车**借走了一直不还**，导致别人都**用不了**这辆车

new了不delete掉

本来能坐50人，现在来46个人就溢出了，**增大了内存溢出可能**

虚析构函数

- 思考：下面代码段有什么问题？

```
class A {  
public:  
    char* pA;  
    A() {  
        pA = new char[100];  
    }  
    virtual void func() {  
        cout << "func in A\n";  
    }  
    ~A() {  
        delete[] pA;  
    }  
};
```

```
class B : public A {  
public:  
    char* pB;  
    B() {  
        pB = new char[100];  
    }  
    void func() {  
        cout << "func in B\n";  
    }  
    ~B() {  
        delete[] pB;  
    }  
};
```

```
A* ptr = (A*) new B;  
ptr->func();  
delete ptr;
```

ptr的类型是A*
因此~B()没被调用
导致内存泄露啦！

虚析构函数

- 请务必把父类的析构函数设置为虚函数！

```
class A {  
public:  
    char* pA;  
    A() {  
        pA = new char[100];  
    }  
    virtual void func() {  
        cout << "func in A";  
    }  
    virtual ~A() {  
        delete[] pA;  
    }  
};
```

```
class B : public A {  
public:  
    char* pB;  
    B() {  
        pB = new char[100];  
    }  
    void func() {  
        cout << "func in B";  
    }  
    ~B() {  
        delete[] pB;  
    }  
};
```

```
A* ptr = (A*) new B;  
ptr->func();  
delete ptr;
```

相当于~A()被~B()重写了
因此实际执行B::~~B()
进而A::~~A()也被调用（因析构顺序）
不会内存泄露啦！

★ **Key 1**: new/delete 是一对运算符，动态创建/销毁单个对象（或基础数据）

★ **Key 2**: new[]/delete[] 动态创建/销毁对象数组（或基础类型数组）

★ **Key 3**: new/delete和malloc/free的区别在于前者会构造/析构对象

1. 下面有关构造函数和new运算符关系正确的说法是: **D**

A. new运算符只能用于类，因为基础数据类型没有构造函数

B. 构造函数一定调用new运算符

C. 要创建新类的实例时，需要先使用new运算符，然后调用构造函数进行初始化

D. 使用new运算符动态产生类的对象时，new运算符会自动调用构造函数

2. 关于delete和delete[]运算符的下列描述中，错误的是: **C**

A. 它们不能用于野指针和垂悬指针

B. 它们可以用于空指针

C. 对一个指针可以使用多次该运算符

D. 使用delete[]无需在括号符内写待删数组的维数

野指针：不知道指向什么乱七八糟地址

垂悬指针：指向的地址已经被回收/清理了

空指针：nullptr关键字，表示“没有地址”的含义

★ Key 4: 内存泄露(借了不还): 导致内存溢出(资源不够)的可能性增大

★ Key 5: 虚析构函数是为了避免子类资源泄露(如内存泄露或文件句柄不释放)

1. 小明做项目时定义了一些类: $F \leftarrow S \leftarrow G$ 。项目运行时, 他发现系统存在内存泄露和文件句柄占用的现象, 下面哪个举措有可能解决该问题: **D**

A. 将成员函数全部声明为内联函数

B. 避免使用重载函数

C. 将G的析构函数声明为虚析构函数

D. 将F的析构函数使用virtual关键字声明

2. 填空: 下列代码有可能导致内存泄漏, 增大内存溢出的可能性。

```
class Base {  
private:  
    char* pc;  
public:  
    Base(int n) : pc(new char[n]) {}  
    ~Base() { delete[] pc; }  
};
```

```
class Derived : public Base {  
private:  
    int* pi;  
public:  
    Derived(int n1, int n2) : Base(n1) { pi = new int[n2]; }  
    ~Derived() { delete[] pi; }  
};
```


★ Key 4: 内存泄露(借了不还): 导致内存溢出(资源不够)的可能性增大

★ Key 5: 虚析构函数是为了避免子类的资源泄露(如内存泄露或文件句柄不释放)

3. 请补全下列代码, 避免内存泄漏问题

```
class A {  
private:  
    int n;  
  
public:  
    A(int n) : n(n) {}  
    virtual ~A() {}  
};
```

```
class B : public A {  
private:  
    int* data;  
  
public:  
    B(int n) : A(n) { data = new int[n]; }  
    ~B() {  
        delete[] data;  
    }  
};
```

```
int main () {  
    A* p = new B(100);  
    // 省略对p的操作  
    delete p;  
}
```



中国大学MOOC
搜索: C++不挂科