# 精讲C++、实战内存池

C++相比于C语言呢，对程序猿更友好，因为编译器替程序猿做了很多事情。理解这句话是学好C++的关键。

C++在堆中创建的对象（申请的内存）都需要手动释放

## 对比C语言

1、C语言是面向过程的，C++是面向对象的。非要死磕的话，两种语言既可以开发出面向过程的程序，也可以开发出面向对象的程序。C语言可以用struct实现面向对象。

2、不精通C语言是无法精通C++的

3、C语言的头文件带.h，C++的头文件不带.h

// C语言的头文件

#include <stdio.h>

// c++的头文件

#include <iostream>

4、extern "C" 告诉编译器要按照C语言的规范来编译

## OOP

### 1、访问权限

* private
* protected
* public

注意：struct中的成员默认是public的，但class中的成员默认是private的

struct Teacher{}

class Teacher{}

### 2、构造函数与析构函数

（1）这两个函数都是由操作系统在运行时自动调用。

（2）构造函数是在创建对象时调用，析构函数是在释放对象时调用。

（3）如果不定义构造函数、析构函数、拷贝构造函数，编译器会自动生成。

#include <iostream>

using namespace std;

class Teacher

{

public:

Teacher() // 构造函数，创建对象时调用

{

cout << "创建一个老师" << endl;

}

~Teacher() // 析构函数，释放对象时调用

{

cout << "释放一个老师" << endl;

}

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

Teacher \*teacher = new Teacher;

delete teacher;

Teacher t1;

{

Teacher t2; // 程序执行这段代码过后会释放Teacher对象，并调用析构函数

}

// 接收用户的输入

getchar();

return 0;

}

### 3、三种方法创建对象

#include <iostream>

using namespace std;

class Teacher

{

private:

bool isCool;

public:

Teacher()

{

cout << "创建一个老师（无参构造）" << endl;

}

Teacher(bool cool)

{

isCool = cool;

cout << "创建一个老师（有参构造）" << endl;

}

Teacher(const Teacher &teacher) // 拷贝构造函数，传了个引用进去&

{

isCool = teacher.isCool;

cout << "创建一个老师（拷贝构造）" << endl;

}

~Teacher()

{

cout << "释放一个老师" << endl;

}

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

// 1 括号法

// Teacher t1; // 调用默认构造函数生成对象,不能写成Teacher t1()

// Teacher t2(true); // 调用有参构造函数生成对象

// Teacher t3(t2); // 调用拷贝构造函数生成对象

// 2 显示法

// Teacher t4 = Teacher();

// Teacher t5 = Teacher(true);

// Teacher t6 = Teacher(t5);

// 3 隐式转换法

Teacher t7 = true;

Teacher t8 = t7;

// 接收用户的输入

getchar();

return 0;

}

### 4、匿名对象

创建对象后马上释放。因为匿名对象在编译器看来就是没有意义的对象。但是如果定义了变量接收匿名对象，则不会创建后马上释放。

#include <iostream>

using namespace std;

class Teacher

{

private:

bool isCool;

public:

Teacher()

{

cout << "创建一个老师" << endl;

}

Teacher(bool cool)

{

isCool = cool;

cout << "创建一个老师（有参构造）" << endl;

}

~Teacher()

{

cout << "释放一个老师" << endl;

}

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

Teacher();// 匿名对象，对象没有引用，对象创建后会立马释放对象

Teacher t1 = Teacher();

// 接收用户的输入

getchar();

return 0;

}

注意：不要使用拷贝构造函数创建匿名对象,程序运行时会报错

Teacher t5 = Teacher(true);

Teacher(t5);

### 5、拷贝构造函数调用时机

如果不定义一个拷贝构造函数，编译器会默认生成一个。

默认的拷贝构造函数执行的是浅拷贝。

#include <iostream>

using namespace std;

class Teacher

{

private:

bool isCool;

public:

Teacher()

{

cout << "创建一个老师（无参构造）" << endl;

}

Teacher(bool cool)

{

isCool = cool;

cout << "创建一个老师（有参构造）" << endl;

}

Teacher(const Teacher &teacher)// 拷贝构造函数，传了&

{

isCool = teacher.isCool;

cout << "创建一个老师（拷贝构造）" << endl;

}

~Teacher()

{

cout << "释放一个老师" << endl;

}

};

void print(Teacher t)

{

cout << "print" << endl;

}

Teacher getTeacher()

{

Teacher t(true);

return t;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

// 1、通过一个已创建的对象来创建一个新对象，先调用无参再调用拷贝函数

Teacher t1;

Teacher t2(t1);

cout << "================" << endl;

// 2、值传递，先调用拷贝函数、再执行print方法、最后执行析构函数

print(t1);

cout << "================" << endl;

// 3、函数返回对象，有参构造、拷贝构造、析构函数

Teacher t3 = getTeacher();

// 接收用户的输入

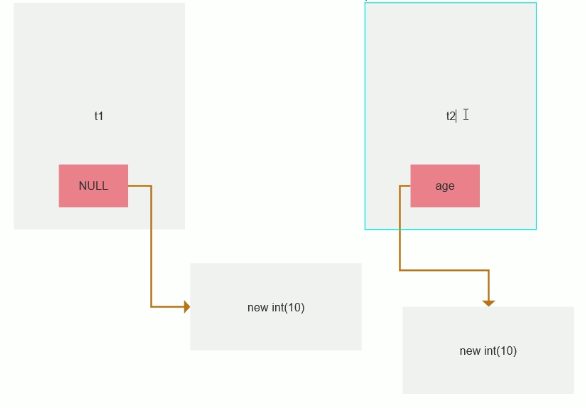
getchar();

return 0;

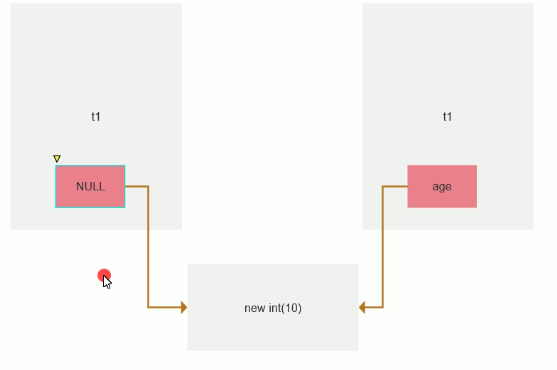
}

### 6、深拷贝与浅拷贝

深拷贝：t1和t2的属性互不关联，内存地址不同，值相同



浅拷贝：t1和t2的属性指针指向同一个对象，内存地址相同，值相同



#include <iostream>

using namespace std;

class Teacher

{

private:

bool isCool;

int \*age;

public:

Teacher()

{

cout << "创建一个老师（无参构造）" << endl;

}

Teacher(bool cool, int age)// 浅拷贝

{

isCool = cool;

this->age = new int(age);

cout << "创建一个老师（有参构造）" << endl;

}

Teacher(const Teacher &teacher) // 深拷贝

{

age = new int(\*teacher.age);

isCool = teacher.isCool;

cout << "析构函数" << endl;

}

~Teacher()

{

if (NULL != age) {

delete age;

age = NULL;

}

cout << "释放一个老师" << endl;

}

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

{

Teacher t1(true, 10);

Teacher t2(t1);

}

// 接收用户的输入

getchar();

return 0;

}

### 7、初始化列表

// 构造函数为属性赋值，相当于java中this.isCool = cool

Teacher(bool cool) : isCool(cool)

{

}

### 8、静态成员

1. 静态成员属性（类内声明，类外初始化）

// 正确

const static int age = 10;

// 错误

static int age = 10;

1. 静态成员方法

只能访问静态成员

可通过类或对象访问（与Java不同）

可通过空指针调用成员函数（成员函数中不得有操作成员属性的代码）

### 9、计算对象大小

1、普通属性32bit，4字节对齐

2、静态属性

3、普通方法

4、静态方法

### 10、this指针的本质

本质是一个指针常量，指向不可变，指向的内容可变

Teacher \* const this;

### 11、const

#### 常函数

// 这就是常函数

int getValue() const

{

this->val = 10;// this代码含义相当于const Integer \* const this引用不可变，值不可变，所以这里会报错

return value;

}

mutable int val;// 加上mutable就不会报错

只能修改mutable修饰的成员属性

#### 常对象

1、只能修改mutable修饰的成员属性

2、只能调用长函数

## 友元

类中被修饰的对象或者函数，在被调用时能访问类的私有属性

class Teacher{

friend class Student;

friend void getTeacherName(Teacher &t);

private:

string a1;

string a2;

}

class Student{ // 友元类

public:

void getTeacherName(Teacher t){

cout << t.a1;

}

}

void getTeacherName(Teacher &t){ // 友元函数

cout << t.a2;

}

### 1、友元函数

friend class Student;

### 2、友元类

friend void getTeacherName(Teacher &t);

### 3、友元类方法（了解即可）

## 运算符重载

### 1、实现方法：成员函数、全局函数

注意：有的操作符重载只能用全局函数

### 2、加号

#include <iostream>

using namespace std;

class Integer

{

public:

Integer()

{

}

Integer(int val)

{

this->val = val;

}

Integer operator+(const Integer &obj)

{

Integer newObj;

newObj.val = this->val + obj.val;

return newObj;

}

int val;

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

Integer t1 = 10;

Integer t2(20);

Integer t3 = t1 + t2;

cout << t3.val << endl;

//===============

Integer t4 = t1.operator+(t2);

cout << t4.val << endl;

// 接收用户的输入

getchar();

return 0;

}

#include <iostream>

using namespace std;

class Integer

{

public:

Integer()

{

}

Integer(int val)

{

this->val = val;

}

int val;

};

Integer operator+(const Integer &obj1, const Integer &obj2)

{

Integer newObj;

newObj.val = obj1.val + obj2.val;

return newObj;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

Integer t1 = 10;

Integer t2(20);

Integer t3 = t1 + t2;

cout << t3.val << endl;

//===============

Integer t4 = operator+(t1, t2);

cout << t4.val << endl;

// 接收用户的输入

getchar();

return 0;

}

注意：运算符重载也可以函数重载，比如Integer需要支持这两种情况下的运算

Integer t3 = t1 + t2;

Integer t3 = t1 + 10;

### 3、<<

只能全局函数

#include <iostream>

using namespace std;

class Integer

{

public:

Integer()

{

this->val = 0;

}

Integer(int val)

{

this->val = val;

}

void operator<<(ostream &out)

{

out << this->val;

}

private:

int val;

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

Integer t1 = 10;

t1 << cout;

// 接收用户的输入

getchar();

return 0;

}

#include <iostream>

using namespace std;

class Integer

{

public:

Integer()

{

}

Integer(int val)

{

this->val = val;

}

int val;

};

void operator<<(ostream &out, Integer &t)

{

out << t.val << endl;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

Integer t1 = 10;

cout << t1;

// 接收用户的输入

getchar();

return 0;

}

我如果想实现链式操作，怎么做？

### 4、++

1、前置++

#include <iostream>

using namespace std;

class Integer

{

friend ostream & operator<<(ostream &out, Integer &t);

public:

Integer()

{

this->val = 0;

}

Integer(int val)

{

this->val = val;

}

Integer operator++()

{

this->val++;

return \*this;

}

private:

int val;

};

ostream & operator<<(ostream &out, Integer &t)

{

out << t.val;

return out;

}

ostream & operator<<(ostream &out, Integer \*t)

{

out << t;

return out;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

Integer t1 = 10;

// Integer t2 = ++(++t1);

cout << t1 << endl;

cout << ++(++t1) << endl;

cout << t1 << endl;

// Integer operator++ 10 12 11

// ++(++t1) -> Integer t2 = ++t1; ++(t2) 所以输出是12，再次输出t1的时候为什么是11？是因为此时的t1其实是t2，它创建了新的对象

// Integer& operator++ 10 12 12

// 返回引用时，++过后为什么是12？是因为没有生成新的对象，++操作只会在t1上的值进行操作，因为返回引用时，执行不变

// 接收用户的输入

getchar();

return 0;

}

涉及知识点：

1、友元函数

2、返回值与返回引用的区别

2、后置++

注意：必须返回值，不能返回引用（想想为什么）

#include <iostream>

using namespace std;

class Integer

{

friend ostream & operator<<(ostream &out, Integer &t);

public:

Integer()

{

this->val = 0;

}

Integer(int val)

{

this->val = val;

}

Integer operator++()

{

this->val++;

return \*this;

}

Integer operator++(int)

{

Integer tmp = \*this;

this->val++;

return tmp;

}

private:

int val;

};

ostream & operator<<(ostream &out, Integer &t)

{

out << t.val;

return out;

}

ostream & operator<<(ostream &out, Integer \*t)

{

out << t;

return out;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

Integer t1 = 10;

cout << t1++ << endl;

cout << t1 << endl;

// 接收用户的输入

getchar();

return 0;

}

### 5、=

知识点：

1. 为什么需要重载赋值操作符（深拷贝，如果不深拷贝，执行析构函数时会重复释放）
2. 重载赋值操作符的函数中记得释放
3. 返回结果应该是返回引用

#include <iostream>

using namespace std;

class Integer

{

friend ostream & operator<<(ostream &out, Integer &t);

public:

Integer()

{

this->val = new int(0);

}

Integer(int val)

{

this->val = new int(val);

}

~Integer()

{

if (NULL != val)

{

delete val;

val = NULL;

}

}

Integer& operator=(Integer &t)

{

// t2赋值前需要移除t2值

if (NULL != val)

{

delete val;

val = NULL;

}

this->val = new int(\*t.val);

return \*this;

}

private:

int \*val;

};

ostream & operator<<(ostream &out, Integer &t)

{

out << \*t.val;

return out;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

Integer t1(10);

Integer t2(20);

Integer t3(30);

cout << t1 << endl;

//=====

t2 = t1;

cout << t2 << ", " << t1 << endl;

//=====

t3 = t2 = t1;

cout << t3 << ", " << t2 << ", " << t1 << endl;

// 接收用户的输入

getchar();

return 0;

}

### 6、>、<、==、!=

### 7、()

括号重载，又称为仿函数，只能写在类方法

#include <iostream>

using namespace std;

class Add

{

private:

int a;

int b;

public:

Add()

{

}

int operator()(int a,int b)

{

return a + b;

}

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

Add add;

cout << add(1,2) << endl;

cout << Add()(1,2) << endl;// Add()是匿名对象

// 接收用户的输入

getchar();

return 0;

}

Java中看到函数调用，调用的一定是方法。C++中则不一定，它有可能对应的是一个对象