**C++:类的构造函数和析构函数**

标签（空格分隔）： c++   
作者：陈小默

* [C类的构造函数和析构函数](https://blog.csdn.net/qq_32583189/article/details/52415478#c%E7%B1%BB%E7%9A%84%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0%E5%92%8C%E6%9E%90%E6%9E%84%E5%87%BD%E6%95%B0)
* [构造函数](https://blog.csdn.net/qq_32583189/article/details/52415478#%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0)
  + [声明和定义](https://blog.csdn.net/qq_32583189/article/details/52415478#%E5%A3%B0%E6%98%8E%E5%92%8C%E5%AE%9A%E4%B9%89)
  + [引用参数与构造函数](https://blog.csdn.net/qq_32583189/article/details/52415478#%E5%BC%95%E7%94%A8%E5%8F%82%E6%95%B0%E4%B8%8E%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0)
  + [使用构造函数定义对象](https://blog.csdn.net/qq_32583189/article/details/52415478#%E4%BD%BF%E7%94%A8%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0%E5%AE%9A%E4%B9%89%E5%AF%B9%E8%B1%A1)
    - [默认构造函数](https://blog.csdn.net/qq_32583189/article/details/52415478#%E9%BB%98%E8%AE%A4%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0)
* [析构函数](https://blog.csdn.net/qq_32583189/article/details/52415478#%E6%9E%90%E6%9E%84%E5%87%BD%E6%95%B0)
  + [声明和定义](https://blog.csdn.net/qq_32583189/article/details/52415478#%E5%A3%B0%E6%98%8E%E5%92%8C%E5%AE%9A%E4%B9%89-1)
  + [析构函数的执行时机](https://blog.csdn.net/qq_32583189/article/details/52415478#%E6%9E%90%E6%9E%84%E5%87%BD%E6%95%B0%E7%9A%84%E6%89%A7%E8%A1%8C%E6%97%B6%E6%9C%BA)
  + [const成员函数](https://blog.csdn.net/qq_32583189/article/details/52415478#const%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0)
* [实例演示](https://blog.csdn.net/qq_32583189/article/details/52415478#%E5%AE%9E%E4%BE%8B%E6%BC%94%E7%A4%BA)

开发环境：Visual Studio 2010

* 1
* 2

**构造函数**

当我们需要在对象创建时初始化一些数据的时候，我们不可能提供一个普通的成员方法供程序猿在对象创建后调用。因为如果程序猿故意或者无意间忘记了调用该方法，就可能导致程序出现偏离预期的结果。为了防止这种情况的发生，C++中提供了一种特殊的成员函数–构造函数。

**声明和定义**

构造函数与普通函数在声明与定义上别无二致，除了不需要声明返回值。   
在声明中添加如下代码

Student(std::string name,int chScore,int enScore,int mathScore);

* 1

在实现中的一般实现方式为

Student::Student(std::string name,int chScore=0,int enScore=0,int mathScore=0){

cout<<"您好呀! "<<name<<",很高兴见到你!"<<endl;

setName(name);

setChScore(chScore);

setEnScore(enScore);

setMathScore(mathScore);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7

**引用参数与构造函数**

当我们在类声明时声明了一个引用参数，而引用参数又要求必须初始化，且只能初始化一次。应该怎么办呢？这是我们需要一种新的成员初始化方式

std::string & name；

* 1

比如我们的name属性是引用类型，那么我们需要在每一个构造方法实现的参数列表后面加上这样的初始化器

Student::Student(std::string &n):name(n){

......

}

Student::Student(std::string &n,int chScore,int enScore,int mathScore):name(n){

......

}

* 1
* 2
* 3
* 4
* 5
* 6

如果我们有多个引用需要初始化，这些构造器只需要使用，逗号分割,并且除了引用类型，任何类型属性都可以使用这种方式初始化

Class::Constructor(type \*t1,type &t2,type t3...):t1(t1),t2(t2),t3(t3)...{}

* 1

**使用构造函数定义对象**

当我们声明过一个构造函数之后，我们可以有三种方式去调用它：   
1，显式的调用

Student jack = Student("jack",99,98,97);

* 1

2，隐式的调用

Student jack("jack",99,98,97);

* 1

3，使用new运算符

Student \*jack = new Student("jack",99,98,97);

* 1

上述3中方法的关键区别在于内存分配：前两种方式的对象将会被在栈中创建，第3中方式对象将会在堆中被创建

**默认构造函数**

当我们没有显式的在声明中添加构造方法时，编译器会默认提供一个空参列表且没有任何行为的默认构造方法，就像下面这样

Student(){}

* 1

但是当我们声明了自己的构造方法之后，编译器便不会再提供默认的构造方法了。   
当我们想定义一个默认的构造器有两种方式：   
1，给已有的构造函数的所有参数提供默认值

Student(std::string name,int chScore=0,int enScore=0,int mathScore=0)；

* 1

2，定义一个没有参数的函数重载

Student()；

* 1

由于每种类型的构造器只能声明一个，所以同时使用上述两种方式会报错。

注意：声明隐式方式调用空参构造方法不能使用括号，因为这是编译器会将这行代码解释为函数声明，比如：

Student jack;//available

Student jack();//invalidate

* 1
* 2

**析构函数**

析构函数是当对象被销毁时被系统调用的方法，一般用来清空内存。

**声明和定义**

析构函数的声明只用一种形式，就是~加上类名

~Student();

* 1

析构函数的实现就可以根据需求去自定义

Student::~Student(){

cout<<"再见! "<<name<<"."<<endl;

}

* 1
* 2
* 3

**析构函数的执行时机**

析构函数的执行与对象的创建位置相关   
1，当对象被声明为全局变量，则当整个程序退出时，其析构函数才会被调用。   
2，当对象被声明为自动存储类型(局部变量)，当其生命周期结束，即其作用域执行完毕时，其析构函数会被调用。   
3，当对象被声明为指针变量(使用new创建对象)，当执行到delete方法时，其析构函数会被调用。   
4，临时对象，这种对象在临时对象完成任务时自动调用其析构函数。下面我们会专门介绍这种情况。

**const成员函数**

有如下代码：

const Student jack("Jack",99,98,97);

jack.show();//complier error

* 1
* 2

因为我们已经将jack对象声明为const类型，所以当我们在调用show()方法时，编译器并不知道其中有没有更改对象内容的方法。所以拒绝编译。如果我们非要这么做的话，可以采用以下方式对show()函数重新定义

void show() const;

* 1

**实例演示**

1，在头文件student.h中声明类Student

//student.h -- Student class interface

//version 00

#ifndef STUDENT\_H\_

#define STUDENT\_H\_

#include <string>

class Student{ //class declaration

private:

std::string name;

int ch;

int en;

int math;

float average;

void count(){

average = (ch + en + math+0.0F)/3;

}

public:

Student();

Student(std::string name,int chScore,int enScore,int mathScore);

~Student();

void setName(std::string name);

void setChScore(int score);

void setEnScore(int score);

void setMathScore(int score);

void show();

};

#endif

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29

2，在文件student.cpp中实现类方法

//student.cpp -- implementing the Student class

//version 00

#include "stdafx.h"

#include "student.h"

Student::Student(){

cout<<"您好呀! 新同学,你叫什么名字?"<<endl;

}

Student::Student(std::string n,int chScore,int enScore,int mathScore):name(n),ch(chScore),en(enScore),math(mathScore){

cout<<"您好呀! "<<name<<",很高兴见到你!"<<endl;

}

Student::~Student(){

cout<<"再见! "<<name<<"."<<endl;

}

void Student::setChScore(int score){

Student::ch = score;

}

void Student::setName(std::string n){

Student::name = n;

}

void Student::setEnScore(int score){

en = score;

}

void Student::setMathScore(int score){

math = score;

}

void Student::show(){

Student::count();

ios\_base::fmtflags orig = cout.setf(ios\_base::fixed,ios\_base::floatfield);

std::streamsize prec = cout.precision(1);

cout<<name<<" 同学的语文成绩为"<<ch<<"分，数学成绩为"<<math<<"分，英语成绩为"<<en<<"分，平均成绩"<<average<<"分"<<endl;

cout.setf(orig,ios\_base::floatfield);

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33

3，在mian方法中调用

//visual studio 2010 --main program

#include "stdafx.h"

#include "student.h"

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

Student \*s1,s2,s3("Jack",90,80,70);

s1 = new Student("Sam",90,95,100);

s1->show();

s2 = Student("Sue",85,90,95);

s2.show();

s3.show();

s1 = new Student("Joe",91,92,93);

s1->show();

delete s1;

return 0;

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20

根据上面的学习内容你猜猜运行结果是什么？

运行结果：   
您好呀! 新同学,你叫什么名字?   
您好呀! Jack,很高兴见到你!   
您好呀! Sam,很高兴见到你!   
Sam 同学的语文成绩为90分，数学成绩为100分，英语成绩为95分，平均成绩95.0分   
您好呀! Sue,很高兴见到你!   
再见! Sue.   
Sue 同学的语文成绩为85分，数学成绩为95分，英语成绩为90分，平均成绩90.0分   
Jack 同学的语文成绩为90分，数学成绩为70分，英语成绩为80分，平均成绩80.0分   
您好呀! Joe,很高兴见到你!   
Joe 同学的语文成绩为91分，数学成绩为93分，英语成绩为92分，平均成绩92.0分   
再见! Joe.   
再见! Jack.   
再见! Sue.   
请按任意键继续…

我们来一步一步的分析代码执行流程，

Student \*s1,s2,s3("Jack",90,80,70);

* 1

这一句代码声明了三个变量，并且定义了其中的两个!!! 这里声明了一个Student的指针类型s1，声明并定义了Student类型的变量s2，s3，后面的两种定义方式都是隐式的，只不过s2的定义如上述 默认构造函数 所述，容易使人产生这里仅仅是声明而不是定义的错觉。

s2 = Student("Sue",85,90,95);

* 1

现在再来看这一句，这里产生了一个上述临时对象。这个临时对象会将其内部的值复制到s2的空间中，其后自动销毁。那么为什么不让s2的引用指向这个新建立的对象呢？这就要从变量内存上分析，由于我们的s2和s3是在栈中分配的内存，而栈又会随着方法的退出而销毁，就导致了栈中的对象是不稳定的。如果我们采用指向的方式，那么我们就不知道什么时间我们指向的栈帧就会撤销，这是我们再去访问我们的方法就会报错了。

s1 = new Student("Joe",91,92,93);

* 1

由于s1是在堆中分配的内存，所以我们才能在new时直接传递引用，但是这种写法是一个不好的编程习惯。因为我们在给指针重新分配内存的时候并没有释放原空间，如果产生大量这种代码将会造成内存溢出

最后，为什么先释放的是s3，后释放的是32呢？是由于栈的FILO性质导致的。

https://blog.csdn.net/qq\_32583189/article/details/52415478