C++构造函数详解

在C++中，有一种特殊的成员函数，它的名字和类名相同，没有返回值，不需要用户显式调用（用户也不能调用），而是在创建对象时自动执行。这种特殊的成员函数就是构造函数（Constructor）。

在《C++类成员的访问权限以及类的封装》一节中，我们通过成员函数 setname()、setage()、setscore() 分别为成员变量 name、age、score 赋值，这样做虽然有效，但显得有点麻烦。有了构造函数，我们就可以简化这项工作，在创建对象的同时为成员变量赋值，请看下面的代码（示例1）：

纯文本 复制

```
#include <iostream>
using namespace std;

class Student{
private:
    char *m_name;
    int m_age;
    float m_score;
public:
    //声明构造函数
    Student(char *name, int age, float score);
    //声明普通成员函数
    void show();
};

//定义构造函数
Student::Student(char *name, int age, float score){
    m_name = name;
    m_age = age;
    m_score = score;
}
//定义普通成员函数
void Student::show(){
    cout<<m_name<<"的年龄是"<<m_age<<", 成绩是"<<m_score<<endl;
}

int main(){
    //创建对象时向构造函数传参
    Student stu("小明", 15, 92.5f);
    stu.show();
    //创建对象时向构造函数传参
    Student *pstu = new Student("李华", 16, 96);
    pstu -> show();

```

```
35.     return 0;
36. }
```

运行结果：
小明的年龄是15，成绩是92.5
李华的年龄是16，成绩是96

该例在 Student 类中定义了一个构造函数 `Student(char *, int, float)` ，它的作用是给三个 private 属性的成员变量赋值。要想调用该构造函数，就得在创建对象的同时传递实参，并且实参由 `()` 包围，和普通的函数调用非常类似。

在栈上创建对象时，实参位于对象名后面，例如 `Student stu("小明", 15, 92.5f)` ；在堆上创建对象时，实参位于类名后面，例如 `new Student("李华", 16, 96)` 。

构造函数必须是 public 属性的，否则创建对象时无法调用。当然，设置为 private、protected 属性也不会报错，但是没有意义。

构造函数没有返回值，因为没有变量来接收返回值，即使有也毫无用处，这意味着：
- 不管是声明还是定义，函数名前面都不能出现返回值类型，即使是 void 也不允许；
- 函数体中不能有 return 语句。

构造函数的重载

和普通成员函数一样，构造函数是允许重载的。一个类可以有多个重载的构造函数，创建对象时根据传递的实参来判断调用哪一个构造函数。

构造函数的调用是强制性的，一旦在类中定义了构造函数，那么创建对象时就一定要调用，不调用是错误的。如果有多个重载的构造函数，那么创建对象时提供的实参必须和其中的一个构造函数匹配；反过来说，创建对象时只有一个构造函数会被调用。

对示例1中的代码，如果写作 `Student stu` 或者 `new Student` 就是错误的，因为类中包含了构造函数，而创建对象时却没有调用。

更改示例1的代码，再添加一个构造函数（示例2）：

```
01. #include <iostream>
02. using namespace std;
03.
04. class Student{
05. private:
```

```
    char *m_name;
    int m_age;
    float m_score;
public:
    Student();
    Student(char *name, int age, float score);
    void setname(char *name);
    void setage(int age);
    void setscore(float score);
    void show();
};

Student::Student(){
    m_name = NULL;
    m_age = 0;
    m_score = 0.0;
}
Student::Student(char *name, int age, float score){
    m_name = name;
    m_age = age;
    m_score = score;
}
void Student::setname(char *name){
    m_name = name;
}
void Student::setage(int age){
    m_age = age;
}
void Student::setscore(float score){
    m_score = score;
}
void Student::show(){
    if(m_name == NULL || m_age <= 0){
        cout<<"成员变量还未初始化"<<endl;
    }else{
        cout<<m_name<<"的年龄是"<<m_age<<", 成绩是"<<m_score<<endl;
    }
}

int main(){
    //调用构造函数 Student(char *, int, float)
    Student stu("小明", 15, 92.5f);
    stu.show();

```

```
    //调用构造函数 Student()
    Student *pstu = new Student();
    pstu -> show();
    pstu -> setname("李华");
    pstu -> setage(16);
    pstu -> setscore(96);
    pstu -> show();

    return 0;
}
```

运行结果：
小明的年龄是15，成绩是92.5
成员变量还未初始化
李华的年龄是16，成绩是96

构造函数 Student(char *, int, float) 为各个成员变量赋值，构造函数 Student() 将各个成员变量的值设置为空，它们是重载关系。根据 Student() 创建对象时不会赋予成员变量有效值，所以还要调用成员函数 setname()、setage()、setscore() 来给它们重新赋值。

构造函数在实际开发中会大量使用，它往往用来做一些初始化工作，例如对成员变量赋值、预先打开文件等。

默认构造函数

如果用户自己没有定义构造函数，那么编译器会自动生成一个默认的构造函数，只是这个构造函数的函数体是空的，也没有形参，也不执行任何操作。比如上面的 Student 类，默认生成的构造函数如下：

```
Student(){}
```

一个类必须有构造函数，要么用户自己定义，要么编译器自动生成。一旦用户自己定义了构造函数，不管有几个，也不管形参如何，编译器都不再自动生成。在示例1中，Student 类已经有了一个构造函数 Student(char *, int, float)，也就是我们自己定义的，编译器不会再额外添加构造函数 Student()，在示例2中我们才手动添加了该构造函数。

> 实际上编译器只有在必要的时候才会生成默认构造函数，而且它的函数体一般不为空。默认构造函数的目的是帮助编译器做初始化工作，而不是帮助程序员。这是C++的内部实现机制，这里不再深究，初学者可以按照上面说的“一定有一个空函数体的默认构造函数”来理解。

最后需要注意的一点是，调用没有参数的构造函数也可以省略括号。对于示例2的代码，在栈上创建对象可以写作 `Student stu()` 或 `Student stu`，在堆上创建对象可以写作 `Student *pstu = new Student()` 或 `Student *pstu = new Student`，它们都会调用构造函数 Student()。

以前我们就是这样做的，创建对象时都没有写括号，其实是调用了默认的构造函数。