

C++类的定义和对象的创建详解

类和对象是 C++ 的重要特性，它们使得 C++ 成为面向对象的编程语言，可以用来开发中大型项目，本节重点讲解类和对象的语法，如果你对它们的概念还不了解，请先阅读《[C++类和对象到底是什么意思](#)》。

类是创建对象的模板，一个类可以创建多个对象，每个对象都是类类型的一个变量；创建对象的过程也叫类的实例化。每个对象都是类的一个具体实例 (Instance)，拥有类的成员变量和成员函数。

有些教程将类的成员变量称为类的属性 (Property)，将类的成员函数称为类的方法 (Method)。在面向对象的编程语言中，经常把函数 (Function) 称为方法 (Method)。

与结构体一样，类只是一种复杂数据类型的声明，不占用内存空间。而对象是类这种数据类型的一个变量，或者说是通过类这种数据类型创建出来的一份实实在在的数据，所以占用内存空间。

类的定义

类是用户自定义的类型，如果程序中要用到类，必须提前说明，或者使用已存在的类（别人写好的类、标准库中的类等），C++语法本身并不提供现成的类的名称、结构和内容。

一个简单的类的定义：

```
01.  class Student{
02.  public:
03.      //成员变量
04.      char *name;
05.      int age;
06.      float score;
07.
08.      //成员函数
09.      void say() {
10.          cout<<name<<"的年龄是"<<age<<"，成绩是"<<score<<endl;
11.      }
12.  };
```

`class` 是 C++ 中新增的关键字，专门用来定义类。`Student` 是类的名称；类名的首字母一般大写，以和其他的标识符区分开。`{ }` 内部是类所包含的成员变量和成员函数，它们统称为类的成员 (Member)；由 `{ }` 包围起来的部分有时也称为类体，和函数体的概念类似。`public` 也是 C++ 的新增关键字，它只能用在类的定义中，表示类的成员变量或成员函数具有“公开”的访问权限，初学者请先忽略该关键字，我们将在《[C++类成员的访问权限以及类的封装](#)》中讲解。

注意在类定义的最后有一个分号 `;`，它是类定义的一部分，表示类定义结束了，不能省略。

整体上讲，上面的代码创建了一个 `Student` 类，它包含了 3 个成员变量和 1 个成员函数。

类只是一个模板 (Template)，编译后不占用内存空间，所以在定义类时不能对成员变量进行初始化，因为没有地方存储数据。只有在创建对象以后才会给成员变量分配内存，这个时候就可以赋值了。

类可以理解为一种新的数据类型，该数据类型的名称是 `Student`。与 `char`、`int`、`float` 等基本数据类型不同的是，`Student` 是一种复杂数据类型，可以包含基本类型，而且还有很多基本类型中没有的特性，以后大家会见到。

创建对象

有了 `Student` 类后，就可以通过它来创建对象了，例如：

```
01. Student liLei; //创建对象
```

`Student` 是类名，`liLei` 是对象名。这和使用基本类型定义变量的形式类似：

```
01. int a; //定义整型变量
```

从这个角度考虑，我们可以把 `Student` 看做一种新的数据类型，把 `liLei` 看做一个变量。

在创建对象时，`class` 关键字可要可不要，但是出于习惯我们通常会省略掉 `class` 关键字，例如：

```
01. class Student LiLei; //正确
02. Student LiLei; //同样正确
```

除了创建单个对象，还可以创建对象数组：

```
01. Student allStu[100];
```

该语句创建了一个 `allStu` 数组，它拥有 100 个元素，每个元素都是 `Student` 类型的对象。

访问类的成员

创建对象以后，可以使用点号 `.` 来访问成员变量和成员函数，这和通过结构体变量来访问它的成员类似，如下所示：

```
01. #include <iostream>
02. using namespace std;
```

```
03.
04. //类通常定义在函数外面
05. class Student{
06. public:
07.     //类包含的变量
08.     char *name;
09.     int age;
10.     float score;
11.     //类包含的函数
12.     void say() {
13.         cout<<name<<"的年龄是"<<age<<"，成绩是"<<score<<endl;
14.     }
15. };
16.
17. int main() {
18.     //创建对象
19.     Student stu;
20.     stu.name = "小明";
21.     stu.age = 15;
22.     stu.score = 92.5f;
23.     stu.say();
24.
25.     return 0;
26. }
```

运行结果：

小明的年龄是15，成绩是92.5

stu 是一个对象，占用内存空间，可以对它的成员变量赋值，也可以读取它的成员变量。

类通常定义在函数外面，当然也可以定义在函数内部，不过很少这样使用。

使用对象指针

C语言中经典的指针在 C++ 中仍然广泛使用，尤其是指向对象的指针，没有它就不能实现某些功能。

上面代码中创建的对象 stu 在栈上分配内存，需要使用 `&` 获取它的地址，例如：

```
01. Student stu;
02. Student *pStu = &stu;
```

pStu 是一个指针，它指向 Student 类型的数据，也就是通过 Student 创建出来的对象。

当然，你也可以在堆上创建对象，这个时候就需要使用前面讲到的 `new` 关键字（[C++ new和delete 运算符简介](#)），例如：

```
01. Student *pStu = new Student;
```

在栈上创建出来的对象都有一个名字，比如 stu，使用指针指向它不是必须的。但是通过 new 创建出来的对象就不一样了，它在堆上分配内存，没有名字，只能得到一个指向它的指针，所以必须使用一个指针变量来接收这个指针，否则以后再也无法找到这个对象了，更没有办法使用它。也就是说，使用 new 在堆上创建出来的对象是匿名的，没法直接使用，必须要用一个指针指向它，再借助指针来访问它的成员变量或成员函数。

栈内存是程序自动管理的，不能使用 delete 删除在栈上创建的对象；堆内存由程序员管理，对象使用完毕后可以通过 delete 删除。在实际开发中，new 和 delete 往往成对出现，以保证及时删除不再使用的对象，防止无用内存堆积。

栈（Stack）和堆（Heap）是 C/C++ 程序员必须要了解的两个概念，我们已在《[C语言内存精讲](#)》专题中进行了深入讲解，相信你必将有所顿悟。

有了对象指针后，可以通过箭头 `->` 来访问对象的成员变量和成员函数，这和通过[结构体指针](#)来访问它的成员类似，请看下面的示例：

```
01. pStu -> name = "小明";
02. pStu -> age = 15;
03. pStu -> score = 92.5f;
04. pStu -> say();
```

下面是一个完整的例子：

```
01. #include <iostream>
02. using namespace std;
03.
04. class Student{
05. public:
06.     char *name;
07.     int age;
08.     float score;
09.
10.     void say() {
11.         cout<<name<<"的年龄是"<<age<<"，成绩是"<<score<<endl;
12.     }
```

```
13. };
14.
15. int main() {
16.     Student *pStu = new Student;
17.     pStu -> name = "小明";
18.     pStu -> age = 15;
19.     pStu -> score = 92.5f;
20.     pStu -> say();
21.     delete pStu; //删除对象
22.
23.     return 0;
24. }
```

运行结果：

小明的年龄是15，成绩是92.5

虽然在一般的程序中无视垃圾内存影响不大，但记得 delete 掉不再使用的对象依然是一种良好的编程习惯。

总结

本节重点讲解了两种创建对象的方式：一种是在栈上创建，形式和定义普通变量类似；另外一种是在堆上使用 new 关键字创建，必须要用一个指针指向它，读者要记得 delete 掉不再使用的对象。

通过对象名字访问成员使用点号 `.`，通过对象指针访问成员使用箭头 `->`，这和结构体非常类似。