

C++对象模型研究实验

计算机 002	刘沁宇	2203613019
---------	-----	------------

为研究 C++ 的对象模型, 我们先定义一个 *student* 类, 使其尽可能涵盖已学的基本概念, 具体如下:

```
1. class student
2. {
3. private:
4.     static int age;    //假设均为同龄学生
5.     char* name;       //姓名
6.     int scores;       //分数
7.     double gpa;       //总绩点
8. public:
9.     static void setAge(int num);    //静态成员函数
10.    student(const char* str, int s, double g); //含参构造函数
11.    ~student();    //析构函数
12.    void print();    //输出类内数据成员的信息以及地址
13. };
14. int student::age = 0;    //初始化 age
```

接着在全局区定义一个长度为2的 *student* 类数组 *stu_global[2]* 并将其初始化, 在外部函数 *func* 中定义一个长度为2的 *student* 类数组 *stu_local[2]* 并将其初始化, 在外部函数 *main* 中定义一个长度为2的 *student* 类数组 *stu_main[2]* 并将其初始化, 然后利用 *student::print()* 函数在 *main* 函数中依次输出 *stu_global[2]*、*stu_local[2]*、*stu_main[2]* 的相关信息, 关键代码如下:

```
1. int main()
2. {
3.     student::setAge(18);    //调用静态函数 setAge
4.     student* stu_main = new student[2]{ student("xxx",94,4.3),student("yyy",90,4.0) };    //main 中定义局部对象 stu_main
5.     for (int i = 0; i < 2; i++)    //输出 stu_global 成员信息
6.     {
7.         Line();    //分界线
8.         cout << "\nstu_global[" << i << " ]    ";
9.         stu_global[i].print();
```

```

10.     }
11.     func();                                //输出 stu_local 成员信息
12.     for (int i = 0; i < 2; i++)            //输出 stu_main 成员信息
13.     {
14.         Line();
15.         cout << "\nstu_main[" << i << "] ";
16.         stu_main[i].print();
17.     }
18.     delete[]stu_main;
19.     return 0;
20. }

```

得到以下实验结果，下面进行分析：

```

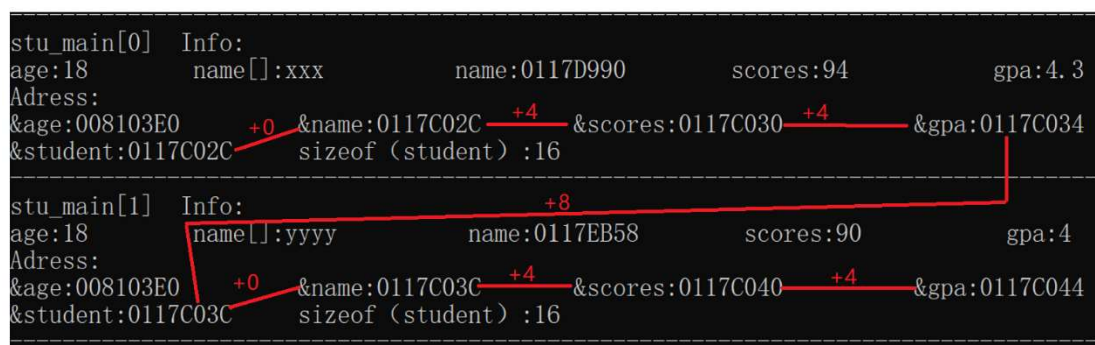
stu_global[0] Info:
age:18      name[]:xxx      name:0117D960      scores:94      gpa:4.3
Address:
&age:008103E0      &name:008103E8      &scores:008103EC      &gpa:008103F0
&student:008103E8      sizeof (student) :16
-----
stu_global[1] Info:
age:18      name[]:yyyyy      name:0117EDF8      scores:90      gpa:4
Address:
&age:008103E0      &name:008103F8      &scores:008103FC      &gpa:00810400
&student:008103F8      sizeof (student) :16
-----
stu_local[0] Info:
age:18      name[]:xxx      name:01175A10      scores:94      gpa:4.3
Address:
&age:008103E0      &name:00D7F690      &scores:00D7F694      &gpa:00D7F698
&student:00D7F690      sizeof (student) :16
-----
stu_local[1] Info:
age:18      name[]:yyyy      name:0117EC70      scores:90      gpa:4
Address:
&age:008103E0      &name:00D7F6A0      &scores:00D7F6A4      &gpa:00D7F6A8
&student:00D7F6A0      sizeof (student) :16
-----
stu_main[0] Info:
age:18      name[]:xxx      name:0117D990      scores:94      gpa:4.3
Address:
&age:008103E0      &name:0117C02C      &scores:0117C030      &gpa:0117C034
&student:0117C02C      sizeof (student) :16
-----
stu_main[1] Info:
age:18      name[]:yyyy      name:0117EB58      scores:90      gpa:4
Address:
&age:008103E0      &name:0117C03C      &scores:0117C040      &gpa:0117C044
&student:0117C03C      sizeof (student) :16
-----

```

首先观察静态数据成员 **age** 的值和地址，对于三个对象数组中所有元素，均有 **age=18**，**&age=008103E0**，与全局对象 **stu_global[0]** 的地址 **&stu_global[0]=008103E8** 相比仅差 8 个字节，而与其他对象数组的地址相差较大，可得 **static** 修饰的静态数据成员是存储

在全局区（静态区）的。

再考察每个 `student` 对象占用的内存大小，`student` 类中包含四个数据成员，类型分别为 `int`，`char*`，`int`，`double`，分别占用 4,4,4,8 个字节，理论上 `sizeof(student)=20`，但实际上其值为 16，因为 `static` 修饰的 `age` 只分配一次内存且分配在全局区，以供所有对象使用，也就是说静态数据成员在程序中也只有一份拷贝，由该类型的所有对象共享访问。当我们在 `main` 函数中为对象数组动态分配内存时也只是为每个对象分配 16 个字节



```
stu_main[0] Info:
age:18      name[]:xxx      name:0117D990      scores:94      gpa:4.3
Address:
&age:008103E0 +0 &name:0117C02C +4 &scores:0117C030 +4 &gpa:0117C034
&student:0117C02C sizeof (student) :16

stu_main[1] Info:
age:18      name[]:yyy      name:0117EB58      scores:90      gpa:4
Address:
&age:008103E0 +0 &name:0117C03C +4 &scores:0117C040 +4 &gpa:0117C044
&student:0117C03C sizeof (student) :16
```

观察 `main` 中的内存分配情况，可看出每个类对象的地址就是对象中第一个非静态数据成员的地址，对象中数据成员的内存是连续分配的，且一个对象数组中的每个对象内存也是连续分配的。

由实验结果看出一个类对象占用的内存空间由类中所由非静态数据成员占用的内存之和决定，定义一个类对象本身不会占用其内存，而且也不包括类中成员函数占用的内存。

值得注意的是当我们在为 `char *name` 动态开辟内存时，这块内存并不在对象中，而是在堆上，对象中仅包含 `name` 这个字符指针占用的 4 个字节。

接着我们重写一个程序，观察函数的地址：


```
1. #include<iostream>
2. #include<string>
3. using namespace std;
4. void text1(){}
5. void text2(){}
6. void text3(){}
7. class student
8. {
9. public:
10.     static void text4() {};
11.     static void text5() {};
```

```

12.     static void text6() {};
13.     void text7() {};
14.     void text8() {};
15.     void text9() {};
16.     void text10() {};
17.
18. };
19. int main()
20. {
21.     cout << "&text1: " << (void*)text1 << endl;
22.     cout << "&text2: " << (void*)text2 << endl;
23.     cout << "&text3: " << (void*)text3 << endl;
24.     printf("&student::text4(): %p\n", &student::text4);
25.     printf("&student::text5(): %p\n", &student::text5);
26.     printf("&student::text6(): %p\n", &student::text6);
27.     printf("&student::text7(): %p\n", &student::text7);
28.     printf("&student::text8(): %p\n", &student::text8);
29.     printf("&student::text9(): %p\n", &student::text9);
30.     printf("&student::text10(): %p\n", &student::text10);
31.     cout << "&main: " << (void*)main << endl;
32.     return 0;
33. }

```

得到以下实验结果：

 选择Microsoft Visual Studio 调试控制台

```

&text1: 0038114A
&text2: 0038104B
&text3: 003810A0
&student::text4(): 00381465
&student::text5(): 00381460
&student::text6(): 0038145B
&student::text7(): 0038146F
&student::text8(): 00381474
&student::text9(): 0038146A
&student::text10(): 00381479
&main: 00381357

```

`text1`, `2`, `3` 为外部函数, `text4`, `5`, `6`, 为静态成员函数, `text7`, `8`, `9`, `10` 为非静态成员函数, 可看出函数 `text1`, `2`, `3` 虽然是连续定义的, 但其地址并非是连续的; 而静态成员函数 `text3`, `4`, `5` 的地址是连续的, 相邻地址之间只差 5 个字节, 且先定义的函数其地址反而较大; 对于函数 `text7`, `8`, `9`, `10`, 其地址呈现相隔一个函数差 5 个字节, 即 `&text7-&text9=5`, `&text8-&text10=5`; 而 `main` 函数的地址与其他函数地址相差较大。而且这些函数的地址相差值不大, 所以这些函数的内存应该都是分配在全局区。

对于上述现象目前我还没有办法解释，但这应该和 C++ 的实现机制有关，可能在我们学过汇编原理等相关知识之后才可能进一步解释。

代码附录

```
1. #include <iostream>
2. #include <cstring>
3. #include <iomanip>
4. #pragma warning(disable : 4996)
5. using namespace std;
6. class student
7. {
8. private:
9.     static int age;    //假设均为同龄学生
10.    char* name;        //姓名
11.    int scores;        //分数
12.    double gpa;        //总绩点
13. public:
14.     static void setAge(int num);    //静态成员函数
15.     student(const char* str, int s, double g); //含参构造函数
16.     ~student();    //析构函数
17.     void print();    //输出类内数据成员的信息以及地址
18. };
19. int student::age = 0;    //初始化 age
20. void student::setAge(int num)    //静态成员函数
21. {
22.     age = num;
23. }
24. student::student(const char* str, int s, double g) //含参构造函数
25. {
26.     name = new char[strlen(str) + 1]{ '\0' };
27.     strcpy(name, str);
28.     scores = s;
29.     gpa = g;
30. }
31. student::~~student()
32. {
33.     if (name)
34.         delete[] name;
35. }
36. void student::print()
37. {
38.     cout << "Info:" << endl;    //输出对象的值
```

```

39.     cout << "age:" << age << setw(15) << "name[]:" << name << setw(15) <<
        "name:" << (void*)name << setw(15) <<
40.         setw(15) << "scores:" << scores << setw(15) << "gpa:" << gpa << endl;
41.     cout << "Adress:" << endl;           //输出数据成员的地址 (&name 指 name 指针的地址,并非 name 的地址)
42.     cout << "&age:" << &age << setw(15) << "&name:" << &name << setw(15) <<
        "&scores:" << &scores <<
43.         setw(15) << "&gpa:" << &gpa << endl;
44.     cout << "&student:" << this << setw(23) << "sizeof
        (student) :" << sizeof(*this) << endl;           //输出占用的字节
45. }
46.
47.
48. void Line()
49. {
50.     for (int n = 1; n <= 100; n++)
51.     {
52.         cout << "-";
53.     }
54. }
55. student stu_global[2]{ student("xxx",94,4.3),student("yyyy",90,4.0)}; //
    定义全局对象 stu_global
56. void func()
57. {
58.     student stu_local[2]{ student("xxx",94,4.3),student("yyyy",90,4.0) };
        //func 中定义局部对象 stu_loacal
59.     for (int i = 0; i < 2; i++)
60.     {
61.         Line();
62.         cout << endl;
63.         cout << "stu_local[" << i << "]" << " ";
64.         stu_local[i].print();
65.     }
66. }
67. void text(){}
68.
69. int main()
70. {
71.     student::setAge(18);                       //调用静态函数 setAge
72.     student* stu_main = new student[2]{ student("xxx",94,4.3),student("yyy
        y",90,4.0) }; //main 中定义局部对象 stu_main
73.     for (int i = 0; i < 2; i++)                 //输出 stu_global 成员信息
74.     {

```

```
75.     Line();
76.     cout << "\nstu_global[" << i << "]"  ";
77.     stu_global[i].print();
78. }
79. func();                                //输出 stu_local 成员信息
80. for (int i = 0; i < 2; i++)            //输出 stu_local 成员信息
81. {
82.     Line();
83.     cout << "\nstu_main[" << i << "]"  ";
84.     stu_main[i].print();
85. }
86. Line();
87. cout << "\n&func: " << (void*)func << endl; //输出外部函数 func 的地址
88. cout << "&main: " << (void*)main << endl;   //输出 main 函数的地址
89. cout << "&text: " << (void*)text << endl;   //输出 main 函数的地址
90. printf("&student::setAge(): %p\n", &student::setAge); //输出静态成员函数 setAge 的地址
91. printf("&student::print(): %p\n", &student::print); //输出非静态成员函数 print 的地址
92. delete[]stu_main;
93. return 0;
94. }
```