C++构造函数初始化列表

构造函数的一项重要功能是对成员变量进行初始化，为了达到这个目的，可以在构造函数的函数体中对成员变量一一赋值，还可以采用初始化列表。

C++构造函数的初始化列表使得代码更加简洁，请看下面的例子：

```
01. #include <iostream>
02. using namespace std;
03. 
04. class Student{
05. private:
06.     char *m_name;
07.     int m_age;
08.     float m_score;
09. public:
10.     Student(char *name, int age, float score);
11.     void show();
12. };
13. 
14. //采用初始化列表
15. Student::Student(char *name, int age, float score): m_name(name), m_age(age),
    m_score(score){
16.     //TODO:
17. }
18. void Student::show(){
19.     cout<<m_name<<"的年龄是"<<m_age<<", 成绩是"<<m_score<<endl;
20. }
21. 
22. int main(){
23.     Student stu("小明", 15, 92.5f);
24.     stu.show();
25.     Student *pstu = new Student("李华", 16, 96);
26.     pstu -> show();
27. 
28.     return 0;
29. }
```

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

如本例所示，定义构造函数时并没有在函数体中对成员变量一一赋值，其函数体为空（当然也可以有其他语句），而是在函数首部与函数体之间添加了一个冒号`:`，后面紧跟`m_name(name), m_age(age), m_score(score)`语句，这个语句的意思相当于函数体内部的`m_name = name; m_age = age; m_score = score;`语句，也是赋值的意思。

使用构造函数初始化列表并没有效率上的优势，仅仅是书写方便，尤其是成员变量较多时，这种写法非常简单明了。

初始化列表可以用于全部成员变量，也可以只用于部分成员变量。下面的示例只对 m_name 使用初始化列表，其他成员变量还是一一赋值：

```
Student::Student(char *name, int age, float score): m_name(name){
    m_age = age;
    m_score = score;
}
```

注意，成员变量的初始化顺序与初始化列表中列出的变量的顺序无关，它只与成员变量在类中声明的顺序有关。请看代码：

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

class Demo{
private:
    int m_a;
    int m_b;
public:
    Demo(int b);
    void show();
};

Demo::Demo(int b): m_b(b), m_a(m_b){ }
void Demo::show(){ cout<<m_a<<", "<<m_b<<endl; }

int main(){
    Demo obj(100);
    obj.show();
    return 0;
}
```

运行结果：
2130567168, 100

在初始化列表中，我们将 m_b 放在了 m_a 的前面，看起来是先给 m_b 赋值，再给 m_a 赋值，其实不然！成员变量的赋值顺序由它们在类中的声明顺序决定，在 Demo 类中，我们先声明的 m_a，再声明的 m_b，所以构造函数和下面的代码等价：

```
Demo::Demo(int b): m_b(b), m_a(m_b){
    m_a = m_b;
    m_b = b;
}
```

给 m_a 赋值时，m_b 还未被初始化，它的值是不确定的，所以输出的 m_a 的值是一个奇怪的数字；给 m_a 赋值完成后才给 m_b 赋值，此时 m_b 的值才是 100。

> obj 在栈上分配内存，成员变量的初始值是不确定的。

初始化 const 成员变量

构造函数初始化列表还有一个很重要的作用，那就是初始化 const 成员变量。初始化 const 成员变量的唯一方法就是使用初始化列表。例如 VS/VC 不支持变长数组（数组长度不能是变量），我们自己定义了一个 VLA 类，用于模拟变长数组，请看下面的代码：

```
class VLA{
private:
    const int m_len;
    int *m_arr;
public:
    VLA(int len);
};

//必须使用初始化列表来初始化 m_len
VLA::VLA(int len): m_len(len){
    m_arr = new int[len];
}
```

VLA 类包含了两个成员变量，m_len 和 m_arr 指针，需要注意的是 m_len 加了 const 修饰，只能使用初始化列表的方式赋值，如果写作下面的形式是错误的：

```
class VLA{
private:
    const int m_len;
    int *m_arr;
public:
```

```
06.     VLA(int len);
07. };
08.
09. VLA::VLA(int len){
10.     m_len = len;
11.     m_arr = new int[len];
12. }
```