36 Member Initializer Lists in C++ (Constructor Initializer List)

构造函数初始化列表，是我们在构造函数中初始化类成员的一种方式。当我们编写类并向这个类添加成员的时候，通常需要用某种方式对这些成员进行初始化，这一般发生在构造函数中，我们有两种方法可以再构造函数中初始化类成员。

1. 构造函数初始化

1. 一般方法

C++

```
#include <iostream>
#include <string>
#define LOG(x) std::cout << x << std::endl;

class Entity
{
private:
    std::string m_Name;
public:
    Entity()    //Entity entity1; 调用默认构造函数，默认使用 "Unknown" 作为名称
    {
        m_Name = "Unknown";
    }

    Entity(const std::string& name)    // Entity entity2("Alice"); 调用带参数构造函数，使用
"Alice" 作为名称
    {
        m_Name = name;
    }

    const std::string& GetName() const { return m_Name; };
};

int main()
{
    Entity e;
    LOG(e.GetName())           // "Unknown"
    Entity e1("Cherno");
    LOG(e1.GetName())          // "Cherno"
    std::cin.get();
}
```

2. 构造函数初始化列表

```C++
class Entity
{
private:
    std::string m_Name;
    int m_Score;
public:
    Entity()
        : m_Name("Unknown"), m_Score(0)   // 要按类成员定义的顺序初始化，否则编译器可能警告
    {
    }

    Entity(const std::string& name)
        : m_Name(name)     // 相当于用括号代替等号，然后移到列表里
    {
       // m_Name = name;
    }


    const std::string& GetName() const { return m_Name; };
};

int main()
{
    Entity e;
    LOG(e.GetName())
    Entity e1("Cherno");
    LOG(e1.GetName())
    std::cin.get();
}
```

不管你怎么写初始化列表，它都会按照类成员的定义顺序进行初始化。如果打破这个顺序，就会导致各种依赖问题。所以要确保写成员初始化列表时，保持和成员变量的声明顺序一样。

为什么要用这个?

代码风格原因，就是如果变量多起来的话，写在list里会让我们的构造函数代码非常干净而且容易阅读。

功能上的区别，这个区别特定的作用在类上。

```C++
Entity()
    :  x(0),y(0),z(0)
{
    m_Name = "Unknown";     // m_Name会被构造两次
    //相当于 std::string("Unknown");
}
```

你创建了两个字符串，其中一个直接被抛弃了，这是性能浪费

```cpp
class Example
{
public:
    Example()
    {
        LOG("Created Entity!");
    }

    Example(int x)
    {
        std::cout << "Created Entity with " << x << "!" << std::endl;
    }
};

class Entity
{
private:
    std::string m_Name;
    Example m_Example;
public:
    Entity()
    {
        m_Name = std::string("Unknown");
        m_Example = Example(8);
    }

    Entity(const std::string& name)
        : m_Name(name)
    {
    }

    const std::string& GetName() const { return m_Name; };
};

int main()
{
    Entity e;                // "Created Entity!"
                             // "Created Entity with 8"

    std::cin.get();
}
```

这里创建了两个Entity，一个是在`private:Example m_Example;`创建的，另一个是`m_Example = Example(8);`，在这里创建了一个新的Example对象，然后赋值给了m_Example(old one)，覆盖造成性能浪费。

但是如果改成移动到初始化列表中：

```C++
Entity()
    : m_Example(8)
    // : m_Example(Example(8))
{
    m_Name = std::string("Unknown");
}
```

这样就只会创建一个对象。所以你应该到处使用成员初始化列表，绝对没有不适用它们的理由