类对象的构造顺序是这样的：

1.分配内存，调用构造函数时，隐式／显示的初始化各数据成员；

2.进入构造函数后在构造函数中执行一般赋值与计算。

使用初始化列表有两个原因：

原因1.必须这样做：

《C++ Primer》中提到在以下三种情况下需要使用初始化成员列表：

   情况一、需要初始化的数据成员是对象的情况(这里包含了继承情况下，通过显示调用父类的构造函数对父类数据成员进行初始化)；

   情况二、需要初始化const修饰的类成员或初始化引用成员数据；

   情况三、子类初始化父类的私有成员；

 ■情况一的说明：数据成员是对象，并且这个对象只有含参数的构造函数，没有无参数的构造函数；

     如果我们有一个类成员，它本身是一个类或者是一个结构，而且这个成员它只有一个带参数的构造函数，而没有默认构造函数，这时要对这个类成员进行初始化，就必须调用这个类成员的带参数的构造函数，如果没有初始化列表，那么他将无法完成第一步，就会报错。

例子：

#include "iostream"

using namespace std;

class Test

{

public:

Test (int, int, int){

cout <<"Test" << endl;

};

private:

int x;

int y;

int z;

};

class Mytest

{

public:

Mytest():test(1,2,3){ //初始化

cout << "Mytest" << endl;

};

private:

Test test; //声明

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

Mytest test;

return 0;

}

输出结果：

①   如果没有mytest():test(1,2,3){}初始化列表就会报错：

因为Test有了显示的带参数的构造函数，那么他是无法依靠编译器生成无参构造函数的，所以没有三个int型数据，就无法创建Test的对象。Test类对象是MyTest的成员，想要初始化这个对象test，那就只能用成员初始化列表，没有其他办法将参数传递给Test类构造函数。

②初始化列表在构造函数执行前执行(这个可以看上面的结果，对同一个变量在初始化列表和构造函数中分别初始化，首先执行参数列表，后在函数体内赋值，后者会覆盖前者)。

■情况二的说明：对象引用或者cosnt修饰的数据成员

从概念上讲，在进入构造函数的函数体之前，对象已经被创建，所以必须在对象创建之前完成初始化，所以C++发明了初始化列表，这种形式的赋值被认为就是初始化。

     情况二：当类成员中含有一个const对象时，或者是一个引用时，他们也必须要通过成员初始化列表进行初始化，因为这两种对象要在声明后马上初始化，而在构造函数中，做的是对他们的赋值，这样是不被允许的。

例子：

class Test

{

priate:

const int a; //const成员声明

public:

Test():a(10){} //初始化

};

或

class Test

{

private:

int &a; //声明

public:

Test(int a):a(a){} //初始化

}

■情况三的说明：子类初始化父类的私有成员，需要在(并且也只能在)参数初始化列表中显示调用父类的构造函数：如下:

 例子：

class Test{

public:

Test(){};

Test (int x){ int\_x = x;};

void show(){cout<< int\_x << endl;}

private:

int int\_x;

};

class Mytest:public Test{

public:

Mytest() ：Test(110){

//Test(110); // 构造函数只能在初始化列表中被显示调用，不能在构造函数内部被显示调用

};

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

Test \*p = new Mytest();

p->show();

return 0;

}

结果：如果在构造函数内部被显示调用输出结果是：-842150451（原因是这里调用了无参构造函数）；

            如果在初始化列表中被显示调用输出结果是：110

原因2.效率要求这样做：

     类对象的构造顺序显示，进入构造函数体后，进行的是计算，是对成员变量的赋值操作，显然，赋值和初始化是不同的，这样就体现出了效率差异，如果不用成员初始化类表，那么类对自己的类成员分别进行的是一次隐式的默认构造函数的调用，和一次赋值操作符的调用，如果是类对象，这样做效率就得不到保障。

注意：构造函数需要初始化的数据成员，不论是否显示的出现在构造函数的成员初始化列表中，都会在该处完成初始化，并且初始化的顺序和其在类中声明时的顺序是一致的，与列表的先后顺序无关，所以要特别注意，保证两者顺序一致才能真正保证其效率和准确性。

为了说明清楚，假设有这样一个类：

class foo

{

 private:

   int a, b;

};

①、foo(){}和foo(int i = 0){}都被认为是默认构造函数，因为后者是默认参数。两者不能同时出现。

②构造函数列表的初始化方式不是按照列表的的顺序，而是按照变量声明的顺序。比如foo里面，a在b之前，那么会先构造a再构造b。所以无论foo():a(b + 1), b(2){}还是foo():b(2),a(b+1){}都不会让a得到期望的值

③构造函数列表能够对const成员初始化。比如foo里面有一个int const c;则foo(int x) : c(x){}可以让c值赋成x。

  不过需要注意的是，c必须在每个构造函数（如果有多个）都有值。

④在继承里面，只有初始化列表可以构造父类的private成员(通过显示调用父类的构造函数)。比如说：

class child : public foo{};

foo里面的构造函数是这样写的:

foo (int x)

{

  a =x;

}.

而在child里面写child(int x){ foo(x); }是通过不了编译的。

只有把子类构造函数写作child(int x) : foo(x){}才可以。

---------------------

作者：凡旭国

来源：CSDN

原文：https://blog.csdn.net/sinat\_20265495/article/details/53670644

版权声明：本文为博主原创文章，转载请附上博文链接！