**C++中的头文件和源文件：如何写头文件**



[guoyong10721073](https://blog.csdn.net/guoyong10721073) 2014-05-07 21:31:26  38201  收藏 34

分类专栏： [个人笔记](https://blog.csdn.net/guoyong10721073/category_1308511.html)

[C++中的头文件和源文件](http://www.cnblogs.com/lidabo/archive/2012/04/17/2454568.html)

转自：博客园http://www.cnblogs.com/lidabo/archive/2012/04/17/2454568.html

一、C++编译模式

通常，在一个C++程序中，只包含两类文件——.cpp文件和.h文件。其中，.cpp文件被称作C++源文件，里面放的都是C++的源代码；而.h文件则被称作C++头文件，里面放的也是C++的源代码。  
C+ +语言支持“分别编译”（separate compilation）。也就是说，一个程序所有的内容，可以分成不同的部分分别放在不同的.cpp文件里。.cpp文件里的东西都是相对独立的，在编 译（compile）时不需要与其他文件互通，只需要在编译成目标文件后再与其他的目标文件做一次链接（link）就行了。比如，在文件a.cpp中定义 了一个全局函数“void a() {}”，而在文件b.cpp中需要调用这个函数。即使这样，文件a.cpp和文件b.cpp并不需要相互知道对方的存在，而是可以分别地对它们进行编译， 编译成目标文件之后再链接，整个程序就可以运行了。  
这是怎么实现的呢？从写程序的角度来讲，很简单。在文件b.cpp中，在调用 “void a()”函数之前，先声明一下这个函数“void a();”，就可以了。这是因为编译器在编译b.cpp的时候会生成一个符号表（symbol table），像“void a()”这样的看不到定义的符号，就会被存放在这个表中。再进行链接的时候，编译器就会在别的目标文件中去寻找这个符号的定义。一旦找到了，程序也就可以 顺利地生成了。  
注意这里提到了两个概念，一个是“定义”，一个是“声明”。简单地说，“定义”就是把一个符号完完整整地描述出来：它是变 量还是函数，返回什么类型，需要什么参数等等。而“声明”则只是声明这个符号的存在，即告诉编译器，这个符号是在其他文件中定义的，我这里先用着，你链接 的时候再到别的地方去找找看它到底是什么吧。定义的时候要按C++语法完整地定义一个符号（变量或者函数），而声明的时候就只需要写出这个符号的原型了。 需要注意的是，一个符号，在整个程序中可以被声明多次，但却要且仅要被定义一次。试想，如果一个符号出现了两种不同的定义，编译器该听谁的？  
这 种机制给C++程序员们带来了很多好处，同时也引出了一种编写程序的方法。考虑一下，如果有一个很常用的函数“void f() {}”，在整个程序中的许多.cpp文件中都会被调用，那么，我们就只需要在一个文件中定义这个函数，而在其他的文件中声明这个函数就可以了。一个函数还 好对付，声明起来也就一句话。但是，如果函数多了，比如是一大堆的数学函数，有好几百个，那怎么办？能保证每个程序员都可以完完全全地把所有函数的形式都 准确地记下来并写出来吗？  
二、什么是头文件  
很显然，答案是不可能。但是有一个很简单地办法，可以帮助程序员们省去记住那么多函数原型的麻烦：我们可以把那几百个函数的声明语句全都先写好，放在一个文件里，等到程序员需要它们的时候，就把这些东西全部copy进他的源代码中。  
这 个方法固然可行，但还是太麻烦，而且还显得很笨拙。于是，头文件便可以发挥它的作用了。所谓的头文件，其实它的内容跟.cpp文件中的内容是一样的，都是 C++的源代码。但头文件不用被编译。我们把所有的函数声明全部放进一个头文件中，当某一个.cpp源文件需要它们时，它们就可以通过一个宏命令 “#include”包含进这个.cpp文件中，从而把它们的内容合并到.cpp文件中去。当.cpp文件被编译时，这些被包含进去的.h文件的作用便发 挥了。  
举一个例子吧，假设所有的数学函数只有两个：f1和f2，那么我们把它们的定义放在math.cpp里：  
/\* math.cpp \*/  
double f1()  
{  
    //do something here....  
    return;  
}  
double f2(double a)  
{  
    //do something here...  
    return a \* a;  
}  
/\* end of math.cpp \*/  
并把“这些”函数的声明放在一个头文件math.h中：  
/\* math.h \*/  
double f1();  
double f2(double);  
/\* end of math.h \*/  
在另一个文件main.cpp中，我要调用这两个函数，那么就只需要把头文件包含进来：  
/\* main.cpp \*/  
#include "math.h"  
main()  
{  
    int number1 = f1();  
    int number2 = f2(number1);  
}  
/\* end of main.cpp \*/  
这 样，便是一个完整的程序了。需要注意的是，.h文件不用写在编译器的命令之后，但它必须要在编译器找得到的地方（比如跟main.cpp在一个目录下）。 main.cpp和math.cpp都可以分别通过编译，生成main.o和math.o，然后再把这两个目标文件进行链接，程序就可以运行了。  
三、#include  
#include 是一个来自C语言的宏命令，它在编译器进行编译之前，即在预编译的时候就会起作用。#include的作用是把它后面所写的那个文件的内容，完完整整地、 一字不改地包含到当前的文件中来。值得一提的是，它本身是没有其它任何作用与副功能的，它的作用就是把每一个它出现的地方，替换成它后面所写的那个文件的 内容。简单的文本替换，别无其他。因此，main.cpp文件中的第一句（#include "math.h"），在编译之前就会被替换成math.h文件的内容。即在编译过程将要开始的时候，main.cpp的内容已经发生了改变：  
/\* ~main.cpp \*/  
double f1();  
double f2(double);  
main()  
{  
    int number1 = f1();  
    int number2 = f2(number1);  
}  
/\* end of ~main.cpp \*/  
不多不少，刚刚好。同理可知，如果我们除了main.cpp以外，还有其他的很多.cpp文件也用到了f1和f2函数的话，那么它们也通通只需要在使用这两个函数前写上一句#include "math.h"就行了。  
四、头文件中应该写什么  
通 过上面的讨论，我们可以了解到，头文件的作用就是被其他的.cpp包含进去的。它们本身并不参与编译，但实际上，它们的内容却在多个.cpp文件中得到了 编译。通过“定义只能有一次”的规则，我们很容易可以得出，头文件中应该只放变量和函数的声明，而不能放它们的定义。因为一个头文件的内容实际上是会被引 入到多个不同的.cpp文件中的，并且它们都会被编译。放声明当然没事，如果放了定义，那么也就相当于在多个文件中出现了对于一个符号（变量或函数）的定 义，纵然这些定义都是相同的，但对于编译器来说，这样做不合法。  
所以，应该记住的一点就是，.h头文件中，只能存在变量或者函数的声明， 而不要放定义。即，只能在头文件中写形如：extern int a;和void f();的句子。这些才是声明。如果写上int a;或者void f() {}这样的句子，那么一旦这个头文件被两个或两个以上的.cpp文件包含的话，编译器会立马报错。（关于extern，前面有讨论过，这里不再讨论定义跟 声明的区别了。）  
但是，这个规则是有三个例外的。  
一，头文件中可以写const对象的定义。因为全局的const对象默 认是没有extern的声明的，所以它只在当前文件中有效。把这样的对象写进头文件中，即使它被包含到其他多个.cpp文件中，这个对象也都只在包含它的 那个文件中有效，对其他文件来说是不可见的，所以便不会导致多重定义。同时，因为这些.cpp文件中的该对象都是从一个头文件中包含进去的，这样也就保证 了这些.cpp文件中的这个const对象的值是相同的，可谓一举两得。同理，static对象的定义也可以放进头文件。  
二，头文件中可 以写内联函数（inline）的定义。因为inline函数是需要编译器在遇到它的地方根据它的定义把它内联展开的，而并非是普通函数那样可以先声明再链 接的（内联函数不会链接），所以编译器就需要在编译时看到内联函数的完整定义才行。如果内联函数像普通函数一样只能定义一次的话，这事儿就难办了。因为在 一个文件中还好，我可以把内联函数的定义写在最开始，这样可以保证后面使用的时候都可以见到定义；但是，如果我在其他的文件中还使用到了这个函数那怎么办 呢？这几乎没什么太好的解决办法，因此C++规定，内联函数可以在程序中定义多次，只要内联函数在一个.cpp文件中只出现一次，并且在所有的.cpp文 件中，这个内联函数的定义是一样的，就能通过编译。那么显然，把内联函数的定义放进一个头文件中是非常明智的做法。  
三，头文件中可以写类 （class）的定义。因为在程序中创建一个类的对象时，编译器只有在这个类的定义完全可见的情况下，才能知道这个类的对象应该如何布局，所以，关于类的 定义的要求，跟内联函数是基本一样的。所以把类的定义放进头文件，在使用到这个类的.cpp文件中去包含这个头文件，是一个很好的做法。在这里，值得一提 的是，类的定义中包含着数据成员和函数成员。数据成员是要等到具体的对象被创建时才会被定义（分配空间），但函数成员却是需要在一开始就被定义的，这也就 是我们通常所说的类的实现。一般，我们的做法是，把类的定义放在头文件中，而把函数成员的实现代码放在一个.cpp文件中。这是可以的，也是很好的办法。 不过，还有另一种办法。那就是直接把函数成员的实现代码也写进类定义里面。在C++的类中，如果函数成员在类的定义体中被定义，那么编译器会视这个函数为 内联的。因此，把函数成员的定义写进类定义体，一起放进头文件中，是合法的。注意一下，如果把函数成员的定义写在类定义的头文件中，而没有写进类定义中， 这是不合法的，因为这个函数成员此时就不是内联的了。一旦头文件被两个或两个以上的.cpp文件包含，这个函数成员就被重定义了。  
五、头文件中的保护措施  
考 虑一下，如果头文件中只包含声明语句的话，它被同一个.cpp文件包含再多次都没问题——因为声明语句的出现是不受限制的。然而，上面讨论到的头文件中的 三个例外也是头文件很常用的一个用处。那么，一旦一个头文件中出现了上面三个例外中的任何一个，它再被一个.cpp包含多次的话，问题就大了。因为这三个 例外中的语法元素虽然“可以定义在多个源文件中”，但是“在一个源文件中只能出现一次”。设想一下，如果a.h中含有类A的定义，b.h中含有类B的定 义，由于类B的定义依赖了类A，所以b.h中也#include了a.h。现在有一个源文件，它同时用到了类A和类B，于是程序员在这个源文件中既把 a.h包含进来了，也把b.h包含进来了。这时，问题就来了：类A的定义在这个源文件中出现了两次！于是整个程序就不能通过编译了。你也许会认为这是程序 员的失误——他应该知道b.h包含了a.h——但事实上他不应该知道。  
使用"#define"配合条件编译可以很好地解决这个问题。在一 个头文件中，通过#define定义一个名字，并且通过条件编译#ifndef...#endif使得编译器可以根据这个名字是否被定义，再决定要不要继 续编译该头文中后续的内容。这个方法虽然简单，但是写头文件时一定记得写进去。