链滴

C++ 中头文件（.h）和源文件（.cpp）都应该写些什么

作者：woyehua

原文链接：https://ld246.com/article/1541603507161

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20180428.jpg?imageView2/1/w/960/h/520/interlace/1/q/100">
C++ 中头文件（.h）和源文件（.cpp）都应该写些什么
头文件(.h)：
<pre><code class="highlight-chroma"><span class="highlight
cl">写类的声明（包括类里面的成员和方法的声明）、函数原型、#define常数等，但一般来说不写
具体的实现。

在写头文件时需要
意，在开头和结尾处必须按照如下样式加上预编译语句（如下）：
</code></pre>
#ifndef CIRCLE_H
#define CIRCLE_H
//你的代码写在这里
#endif
<pre><code class="highlight-chroma"><span class="highlight
cl">这样做是为了防止重复编译，不这样做就有可能出错。

至于CIRCLE_H这
名字实际上是无所谓的，你叫什么都行，只要符合规范都行。原则上来说，非常建议把它写成这种形
，因为比较容易和头文件的名字对应。
</code></pre>
源文件（.cpp）：
<pre><code class="highlight-chroma"><span class="highlight
cl">源文件主要写实现头文件中已经声明的那些函数的具体代码。需要注意的是，开头必须#include
下实现的头文件，以及要用到的头文件。那么当你需要用到自己写的头文件中的类时，只需要#includ
进来就行了。

下面举个最简单的
子来描述一下，咱就求个圆面积。

 第1步，建立一个
工程（以在VS2003环境下为例）。

 第2步，在头文件
文件夹里新建一个名为Circle.h的头文件，它的内容如下：
</code></pre>
#ifndef CIRCLE_H
#define CIRCLE_H
class Circle
{
private:
double r;//半径
public:
Circle();//构造函数
Circle(double R);//构造函数
double Area();//求面积函数
};
#endif
注意到开头结尾的预编译语句。在头文件里，并不写出函数的具体实现。
<pre><code class="highlight-chroma"><span class="highlight
cl">第3步，要给出Circle类的具体实现，因此，在源文件夹里新建一个Circle.cpp的文件，它的内容
下：
</code></pre>

#include "Circle.h"
Circle::Circle()
{
this->r=5.0;
}
Circle::Circle(double R)
{
this->r=R;
}
double Circle:: Area()
{
return 3.14rr;
}
需要注意的是：开头处包含了 Circle.h，事实上，只要此 cpp 文件用到的文件，都要包含进来！这个件的名字其实不一定要叫 Circle.cpp，但非常建议 cpp 文件与头文件相对应。
<pre><code class="highlight-chroma">最后，我们建一个main.cpp来测试我们写的Circle类，它的内容如下：
</code></pre>
#include
#include "Circle.h"
using namespace std;
int main()
{
Circle c(3);
cout<<"Area="<<c.Area()<<endl;
return 1;
}
注意到开头时有#include "Circle.h"的声明，证明我们使用到了我们刚才写的 Circle 类。
至此，我们工程的结构为：
<pre><code class="highlight-chroma">运行一下，输出结果为：
</code></pre>
说明我们写的 Circle 类确实可以用了。
1..h 叫做头文件，它是不能被编译的。“#include”叫做编译预处理指令，可以简单理解成，在 1 cpp 中的#include"1.h"指令把 1.h 中的代码在编译前添加到了 1.cpp 的头部。每个.cpp 文件会被编，生成一个.obj 文件，然后所有的.obj 文件链接起来你的可执行程序就算生成了。
发现了没有，你要在.h 文件中严格区分声明语句和定义语句。好的习惯是，头文件中应只处理常、变量、函数以及类等等等等的声明，变量的定义和函数的实现等等等等都应该在源文件.cpp 中进行
至于.h 和.cpp 具有同样的主文件名的情况呢，对编译器来讲是没有什么意义的，编译器不会去配二者的主文件名，相反它很傻，只认#include 等语句。但是这样写是一种约定俗成的编程风格，个类的名字作为其头文件和源文件的主文件名比如 Class1.h 和 Class1.cpp，这个类的声明在 Class1.h 中，实现在 Class1.cpp 中，我们人类看起来比较整齐，读起来方便，也很有利于模块化和源代码的用。
为什么这个风格会约定俗成？有一句著名的话，叫“程序是为程序员写的”。
2.h 文件和 cpp 文件也就是说，在 h 文件中声明 Declare，而在 cpp 文件中定义 Define。 “明”向计算机介绍名字，它说， “这个名字是什么意思”。而“定义”为这个名字分配存储空间。无涉及到变量时还是函数时含义都一样。无论在哪种情况下，编译器都在“定义”处分配存储空间。对变量，编译器确定这个变量占多少存储单元，并在内存中产生存放它们的空间。对于函数，编译器产代码，并为之分配存储空间。函数的存储空间中有一个由使用不带参数表或带地址操作符的函数名产的指针。定义也可以是声明。如果该编译器还没有看到过名字 A，程序员定义 int A，则编译器马上这个名字分配存储地址。声明常常使用于 extern 关键字。如果我们只是声明变量而不是定义它，则求使用 extern。对于函数声明， extern 是可选的，不带函数体的函数名连同参数表或返回值，自动作为一个声明。

另篇：
在 C++ 编程过程中，随着项目的越来越大，代码也会越来越多，并且难以管理和分析。于是，在 C++ 中就要分出了头(.h)文件和实现(.cpp)文件，并且也有了 Package 的概念。
对于以 C 起步，C#作为“母语”的我刚开始跟着导师学习 C++ 对这方面还是感到很模糊。虽我可以以 C 的知识面对 C++ 的语法规范，用 C#的思想领悟 C++ 中类的使用。但是 C#中定义和实是都在一个文件中(其实都是在类里面)，而使用 C 的时候也只是编程的刚刚起步，所写的程序也只要个文件就够了。因此对于 C++ 的 Package 理解以及.h 文件和.cpp 文件的总是心存纠结。
幸好导师有详细的 PPT 让我了解，一次对于 Package 的认识就明白多了。简单讲，一个 Packag 就是由同名的.h 和.cpp 文件组成。当然可以少其中任意一个文件：只有.h 文件的 Package 可以是口或模板(template)的定义；只有.cpp 文件的 Package 可以是一个程序的入口。
当然更具体详细的讲解，欢迎下载导师的教学 PPT-Package 来了解更多。
不过我在这里想讲的还是关于.h 文件和.cpp 文件
知道 Package 只是相对比较宏观的理解：我们在项目中以 Package 为编辑对象来扩展和修正我的程序。编写代码时具体到应该把什么放到.h 文件，又该什么放在.cpp 文件中，我又迷惑了。
虽然 Google 给了我很多的链接，但是大部分的解释都太笼统了：申明写在.h 文件，定义实现写.cpp 文件。这个解释没有差错，但是真正下手起来，又会发现不知道该把代码往哪里打。
于是我又把这个问题抛给了导师，他很耐心地给我详详细细地表述了如何在 C++ 中进行代码分。很可惜，第一次我听下了，但是没有听太懂，而且本来对 C++ 就了解不深，所以也没有深刻的印。
经过几个项目的试炼和体验之后，我又拿出这个问题问导师，他又一次耐心地给我讲解了一遍（发誓他绝对不是忘记了我曾经问过同样的问题），这次我把它记录了下来。
为了不再忘记，我将它们总结在这里。
概览
非模板类型(none-template)　模板类型(template)
头文件(.h)　
全局变量申明（带 extern 限定符）
全局函数的申明
带 inline 限定符的全局函数的定义
带 inline 限定符的全局模板函数的申明和定义
类的定义
类函数成员和数据成员的申明（在类内部）
类定义内的函数定义（相当于 inline）
带 static const 限定符的数据成员在类内部的初始化
带 inline 限定符的类定义外的函数定义
模板类的定义
模板类成员的申明和定义（定义可以放在类内或者类外，类外不需要写 inline）
实现文件(.cpp)　
全局变量的定义（及初始化）
全局函数的定义
(无)
类函数成员的定义
类带 static 限定符的数据成员的初始化
*申明：declaration
*定义：definition
头文件
头文件的所有内容，都必须包含在
#ifndef {Filename}
#define {Filename}
//{Content of head file}
#endif
这样才能保证头文件被多个其他文件引用(include)时，内部的数据不会被多次定义而造成错误
inline 限定符
在头文件中，可以对函数用 inline 限定符来告知编译器，这段函数非常的简单，可以直接嵌入到调用义之处。

当然 inline 的函数并不一定会被编译器作为 inline 来实现，如果函数过于复杂，编译器也会拒绝 nline。
因此简单说来，代码最好短到只有 3-5 行的才作为 inline。有循环，分支，递归的函数都不要用 inline。
对于在类定义内定义实现的函数，编译器自动当做有 inline 请求（也是不一定 inline 的）。因在下边，我把带有 inline 限定符的函数成员和写在类定义体内的函数成员统称为“要 inline 的函数员”
非模板类型
全局类型
就像前面笼统的话讲的：申明写在.h 文件。
对于函数来讲，没有实现体的函数，就相当于是申明；而对于数据类型（包括基本类型和自定义型）来说，其申明就需要用 extern 来修饰。
然后在.cpp 文件里定义