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](http://www.cnblogs.com/ider/archive/2011/06/30/what_is_in_cpp_header_and_implementation_file.html" \l "cpp_director" \t "http://blog.csdn.net/tcolin/article/details/_blank)**学习C++对这方面还是感到很模糊。虽然我可以以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#中定义和实现是都在一个文件中(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，所写的程序也只要一个文件就够了。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](http://www.cnblogs.com/ider/archive/2011/06/30/what_is_in_cpp_header_and_implementation_file.html" \l "cpp_director" \t "http://blog.csdn.net/tcolin/article/details/_blank)**有详细的**[PPT](http://files.cnblogs.com/ider/CPP_Packages.ppt" \t "http://blog.csdn.net/tcolin/article/details/_blank)**让我了解，一次对于Package的认识就明白多了。简单讲，一个Package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可以是接口或模板(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](http://files.cnblogs.com/ider/CPP_Packages.ppt" \t "http://blog.csdn.net/tcolin/article/details/_blank)**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们的程序。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，定义实现写在.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](http://www.cnblogs.com/ider/archive/2011/06/30/what_is_in_cpp_header_and_implementation_file.html" \l "cpp_director" \t "http://blog.csdn.net/tcolin/article/details/_blank)**，他很耐心地给我详详细细地表述了如何在C++中进行代码分离。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象。

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](http://www.cnblogs.com/ider/archive/2011/06/30/what_is_in_cpp_header_and_implementation_file.html" \l "cpp_director" \t "http://blog.csdn.net/tcolin/article/details/_blank)**，他又一次耐心地给我讲解了一遍（我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)](http://www.cnblogs.com/ider/archive/2011/06/30/what_is_in_cpp_header_and_implementation_file.html" \l "none_template_type" \t "http://blog.csdn.net/tcolin/article/details/_blank)** | **[模板类型(template)](http://www.cnblogs.com/ider/archive/2011/06/30/what_is_in_cpp_header_and_implementation_file.html" \l "none_template_type" \t "http://blog.csdn.net/tcolin/article/details/_blank)** |
| **[头文件(.h)](http://www.cnblogs.com/ider/archive/2011/06/30/what_is_in_cpp_header_and_implementation_file.html" \l "header_file" \t "http://blog.csdn.net/tcolin/article/details/_blank)** | 全局变量申明（带extern限定符）  全局函数的申明  带**[inline限定符](http://www.cnblogs.com/ider/archive/2011/06/30/what_is_in_cpp_header_and_implementation_file.html" \l "inline_qualifier" \t "http://blog.csdn.net/tcolin/article/details/_blank)**的全局函数的定义 | 带**[inline限定符](http://www.cnblogs.com/ider/archive/2011/06/30/what_is_in_cpp_header_and_implementation_file.html" \l "inline_qualifier" \t "http://blog.csdn.net/tcolin/article/details/_blank)**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](http://www.cnblogs.com/ider/archive/2011/06/30/what_is_in_cpp_header_and_implementation_file.html" \l "static_const_qualifier" \t "http://blog.csdn.net/tcolin/article/details/_blank)**的数据成员在**类内部**的初始化  带**[inline限定符](http://www.cnblogs.com/ider/archive/2011/06/30/what_is_in_cpp_header_and_implementation_file.html" \l "inline_qualifier" \t "http://blog.csdn.net/tcolin/article/details/_blank)**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定义之处。

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inline。

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做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）。因此在下边，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的函数成员”

## 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类型）来说，其申明就需要用extern来修饰。

然后在.cpp文件里定义、实现或初始化这些全局函数和全局变量。

不过导师一直反复强调：不许使用全局函数和全局变量。用了之后造成的后果，目前就是交上去的作业项目会扣分。当然不能用自有不能用的理由以及解决方案，不过不在目前的讨论范围内。

### 自定义类型

对于自定义类型，包括类（class）和结构体（struct），它们的定义都是放在.h文件中。其成员的申明和定义就比较复杂了，不过看上边的表格，还是比较清晰的。

#### 函数成员

函数成员无论是否带有static限定符，其申明都放在.h文件的类定义内部。

对于要inline的函数成员其定义放在.h文件；其他函数的实现都放在.cpp文件中。

#### 数据成员

数据成员的申明与定义都是放在.h文件的类定义内部。对于数据类型，关键问题是其初始化要放在什么地方进行。

对于只含有static限定符的数据成员，它的初始化要放在.cpp文件中。因为它是所有类对象共有的，因此必须对它做合适的初始化。

对于只含有const限定符的数据成员，它的初始化只能在构造函数的初始化列表中完成。因为它是一经初始化就不能重新赋值，因此它也必须进行合适的初始化。

对于既含有static限定符，又含有const限定符的数据成员，它的初始化和定义同时进行。它也是必须进行合适的初始化

对于既没有static限定符，又没有const限定符的数据成员，它的值只针对本对象可以随意修改，因此我们并不在意它的初始化什么时候进行。

## 模板类型

C++中，模板是一把开发利器，它与C#，**[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "http://blog.csdn.net/tcolin/article/details/_blank)**的泛型很相似，却又不尽相同。以前，我一直只觉得像泛型，模板这种东西我可能一辈子也不可能需要使用到。但是在导师的强制逼迫使用下，我才真正体会到模板的强大，也真正知道要如何去使用模板，更进一步是如何去设计模板。不过这不是三言两语可以讲完的，就不多说了。

对于模板，最重要的一点，就是在定义它的时候，编译器并不会对它进行编译，因为它没有一个实体可用。

只有模板被具体化（specialization）之后（用在特定的类型上），编译器才会根据具体的类型对模板进行编译。

所以才定义模板的时候，会发现编译器基本不会报错（我当时还很开心的：我写代码尽然会没有错误，一气呵成），也做不出智能提示。但是当它被具体用在一个类上之后，错误就会大片大片的出现，却往往无法准确定位。

因此设计模板就有设计模板的一套思路和方式，但是这跟本文的主题也有偏。

因为模板的这种特殊性，它并没有自己的准确定义，因此我们不能把它放在.cpp文件中，而要把他们全部放在.h文件中进行书写。这也是为了在模板具体化的时候，能够让编译器可以找到模板的所有定义在哪里，以便真正的定义方法。

至于模板类函数成员的定义放在哪里，导师的意见是放在类定义之外，因为这样当你看类的时候，一目了然地知道有那些方法和数据；我在用Visual Studio的时候查看到其标准库的实现，都是放在类内部的。

可能是我习惯了C#的风格，我比较喜欢把它们都写在类内部，也因为在开发过程中，所使用的编辑器都有一个强大的功能：代码折叠。

当然还有其他原因就是写在类外部，对于每一个函数成员的实现都需要把模板类型作为限定符写一遍，把类名限定符也要写一遍。