C++ 编译单元和命名空间

编译单元

编译单元，指的是代码的物理组织形式。根据C++标准，每一个cpp 文件就是一个编译单元。

- 编译器不会去编译 `.h` 或者 `.hpp` 文件；
- 编译器只会编译 `.c` 或 `.cpp` 文件；

简单来说，当一个c或cpp文件在编译时， 预处理器首先 递归包含头文件，这也就是为什么常会有：#ifndef……#define……#endif。之后，形成一个含有所有必要信息的单个源文件，这个源文件就是一个编译单元。这个编译单元会被编译成为一个与cpp文件名同名的目标文件 。
编译器不能检查跨越目标文件或编译单元之间的名称冲突，这是链接器的工作。链接器把不同编译单元中产生的符号联系起来，构成一个可执行程序。如：

```
//文件first.cpp
int integerValue = 0;
int main(){
    int integerValue = 0;
    return 0;
};
//文件second.cpp
int integerValue = 0;

/* 错误: error LNK2005: "int integerValue" (?integerValue@@3HA) 已经在 second.obj 中定义 first.obj   */
```

GCC将C++代码转为机器码，理论上需要四个步骤：预处理（preprocessing）、编译（compilation）、汇编（assembly）以及链接（linking）3；四个步骤对应四个主体：预处理器（preprocessor）、编译器（compiler）、汇编器（assembler）以及链接器（linker）。实际预处理与编译其实是一个步骤，共需要三个步骤：预处理&编译、汇编以及链接。参见：GCC的C++入门。

宏include的作用

C/C++中的宏本质是文本处理器，`#include` 从机制上来说，只是一种内容的拷贝。参见： 深入理解include预编译原理。

作用域

作用域指的是对象(变量、类or函数)的可见性。作用域有：类内部，代码块内部，函数内部等等。函数外部声明和定义的变量的作用域为整个文件，从定义处到本文件末尾。如果函数内部局部变量和外部的全局变量重名，此时在函数内操作外部全局变量时就要使用作用域操作符 `::` 符号。 如：

```
int integerValue = 0; //全局变量
int main(){
int integerValue = 0;
::integerValue=10;  //切换到全局作用域
return 0;
};
```

链接性

名称链接性分为内部(internal)和外部(external)，指的是：名称在一个还是多个编译单元中可用。 const变量默认链接性为内部，如果要修改需要显示的声明进行覆盖。

命名空间

命名空间基本思想是，将相关的项目组合到一个特定的(已命名区域)。关键字namespace，用法类似结构体和类：namespace 名称{} 。对于已命名的命名空间，可以有多个实例。这些实例可以在一个文件中，也可在不同的编译单元，编译器会把它们合并成一个命名空间。

命名空间可以嵌套，因为命名空间的定义也是声明，如：

```
namespace Window{
    namespace Pane{
        int a;
    }
}
// 在命名空间外部访问就是:
Window::Pane::a;
```

命名空间中可以声明和定义函数，但是优秀的设计一般将接口(声明)和实现(定义)分开，所以命名空间定义中一般只放声明。详见 Google C++ Style Guide 。添加新成员，只能在命名空间体体内进行。
命名空间使用， using关键字导 入后，作用域 从声明处到当前作用域结束：

- using编译：将命名空间中声明的所有名称导入当前作用域。 如：using namespace std;
- using声明：将命名空间中声明的指定名称导入当前作用域。 如 ： using std::cout;

命名空间可以起别名，尤其对于命名空间名字很长时适用，注意不要和已有的命名空间冲突。未命名命名空间，常用于防止目标文件和其他编译单元中的全局数据发生名称冲突。每个编译单元都有一个独一无二的未命名名称空间。