**来自C语言的静态元素：**

在C和C++中，static都有两种基本的含义，并且这两种含义经常是互相冲突的:

（1）在固定的地址上进行存储分配，也就是说对象是在一个特殊的静态数据区(staticdata area) 上创建的，而不是每次函数调用时在堆栈上产生的。这也是静态存储的概念

（2）对一个特定的编译单位来说是局部的(就像在后面将要看到的，这在C++中局限于类的范围)。这样，static控制名字的可见性 (visibility)，所以这个名字在这个单元或类之外是不可见的。这也描述了连接的概念，它决定连接器将看到哪些名字。

关于一般的静态变量的规则同样适用于用户自定义的静态对象，而且它同样也必须有 初始化操作。但是，零赋值只对内建类型有效，用户自定义类型必须用构造函数来初始 化。因此，如果在定义一个静态对象时没有指定构造函数参数，这个类就必须有默认的构 造函数。

一般情况下，在文件作用域（file scope）内的所有名字（即不嵌套在类或函数中的名字）对程序中的所有翻译单元来说都是可见的。这就是所谓的外部连接（external linkage），因为在连接时这个名字对连接器来说是可见的，对单独的翻译单元来说，它是外部的。全局变量和普通函数都有外部连接。

auto是“automatic”的缩写，它指明编译器自动为该变量分配存储空间的方法。实际上编译器总是可以从变量定义时的上下文中判断出这是一个局部变量，所以auto是多余的。 还有一个是register，它说明的也是局部（auto）变量，但它告诉编译器这个特殊的变量要经常用到，所以编译器应该尽可能地让它保存在寄存器中。它用于优化代码。但各种 编译器对这种类型的变量处理方式也不尽相同，它们有时会忽略这种存储类型的指定。一般，如果要用到这个变量的地址，register指定符通常都会被忽略。应该避免用register类型，因为编译器在优化代码方面通常比我们做得更好。

**名字空间：**

可以用C++的名字空间（namespace）特征，把一个全局名字空间分成多个可管理的小空间。关键字namespace，如同class、struct、enum和union一样，把它们的成员的名字放到了不同的空间中去，尽管其他的关键字有其他的目的，但namespace惟一的目的是产生一个新的名字空间。

在一个名字空间中引用一个名字可以采取两种方法：第一种方法是用作用域运算符，第二种方法是用using指令把所有名字引入到名字空间中。

在头文件中，最好使用明确的限定或者被限定在一定范围内的using指令和using声明。

**C++中的静态成员：**

因为类的静态数据成员有着单一的存储空间而不管产生了多少个对象，所以存储空间必须在一个单独的地方定义。编译器不会分配存储空间。如果一个静态数据成员被声明但没有定义时，连接器会报告一个错误。

定义必须出现在类的外部（不允许内联）而且只能定义一次，因此它通常放在一个类的实现文件中。

**静态初始化的相依性：**

有三种方法来处理这一问题：

（1）不用它，避免初始化时的互相依赖。这是最好的解决方法。

（2）如果实在要用，就把那些关键的静态对象的定义放在一个文件中，这样只要让它们在文件中顺序正确就可以保证它们正确的初始化。

（3）如果确信把静态对象放在几个不同的翻译单元中是不可避免的—如在编写一个库时，这时无法控制那些使用该库的程序员—这可以通过两种程序设计技术加以解决。

**替代连接说明：**

C++中提供了一个替代连接说明（alternate linkage specification），它是通过重载 extern关键字来实现的。extern后跟一个字符串来指定想声明的函数的连接类型，后面是函数声明。