Item01：视C++为一个语言联邦

记住：

* C++可以分为4种次语言，包括（C、Object-Oriented C++、Template C++、STL），C++高效编程守则视情况而改变，取决于你使用C++的哪一部分。

Item02：尽量以const，enum，inline替换 #define

记住：

* 对于单纯常量，最好以const对象或enums替换#defines。
* 对于形似函数的宏（macros），最好改用inline函数替换#defines。

Item03：尽可能使用const

记住：

* 将某些东西声明为const可帮助编译器侦测出错误用法。Const可被施加于任何作用域内的对象、函数参数、函数返回类型、成员函数本体。
* 编译器强制实施bitwise constness，但你编写程序时应该使用“概念上的常量性”。
* 当const和non-const成员函数有着实质等价的实现时，令non-const版本调用const版本可避免代码重复。

Item04：确定对象被使用前已先被初始化

记住：

* 为内置型对象进行手工初始化，因为C++不保证初始化它们。
* 构造函数最好是很好成员初值列（member initialization list），而不要在构造函数本体内使用赋值操作（assignment），初始列列出的成员变量，其排列次序应该和它们在class中的声明次序相同。
* 为免除“跨编译单元之初始化次序”问题，请以local static对象替换non-local static对象。

Item05：了解C++默默编写并调用哪些函数

记住：

* 编译器可以暗自为class创建default构造函数、copy构造函数、copy assignment操作符，以及析构函数。

Item06：若不想使用编译器自动生成的函数，要明确拒绝。

记住：

* 为驳回编译器自动（暗自）提供的机能，可将相应的成员函数声明为private并且不予实现。使用像Uncopyable这样的base class也是一种做法。

Item07：为多态基类声明virtual析构函数

记住：

* Polymorphic（多态性质的）base classes应该声明一个virtual析构函数。如果class带有任何virtual函数，它就应该拥有一个virtual析构函数。
* Classes的设计目的如果不是作为base classes使用，或不是为了具备多态性质（polymorphically），就不应该声明virtual析构函数。

Item08：别让异常逃离析构函数

记住：

* 析构函数绝对不要吐出异常。如果一个被析构函数调用的函数可能抛出异常，析构函数应该捕捉任何异常，然后吞下它们（不传播）或结束程序。
* 如果客户需要对某个 操作函数运行期间抛出的异常做出反应，那么class应该提供一个普通函数（而非在析构函数中）执行该操作。