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。

Item09：绝不在构造和析构过程中调用virtual函数

记住：

* 在构造和析构期间不要调用virtual函数，因为这类调用从不下降至derived class（比起当前执行构造函数和析构函数的那层）。

Item10：令operator=返回一个reference to \*this

记住：

* 令赋值（assignment）操作符返回一个reference to \*this。

Item11：在operator=中处理“自我赋值”

记住：

* 确保当对象自我赋值时operator=有良好行为。其中技术包括比较“来源对象”和“目标对象”的地址、精心周到的语句顺序、以及copy-and-swap。
* 确定任何函数如果操作一个以上的对象，而其中多个对象是同一个对象时，其行为仍然正确。

Item12：复制对象时勿忘记其每一个成分

记住：

* Copying函数应该确保复制“对象内的所有成员变量”及“所有base class成分”。
* 不要尝试以某个copying函数实现另一个copying函数。应该将共同机能放进第三个函数中，并由两个copying函数共同调用。

Item13：以对象管理资源

记住：

* 为了防止资源泄露，请使用RAII对象，它们在构造函数中获得资源并在析构函数中释放资源。
* 两个常被使用的RAII classes分别是tr1：shared\_ptr和atuo\_ptr。前者通常是较佳选择，因为其copy行为比较直观。若选择auto\_ptr，复制动作会使它（被复制物）指向null。

Item14：在资源管理类中小心copying行为

记住：

* 复制RAII对象必须一并复制它所管理的资源，所以资源的copying行为决定RAII对象的copying行为。
* 普通而常见的RAII class copying行为是：抑制copying、施行引用计数法（reference counting）。不过其他行为也都可能被实现。

Item15：在资源管理类中提供对原始资源的访问

记住：

* APIs往往要求访问原始数据（raw resources），所以每一个RAII class应该提供一个“取得其所管理之资源”的办法。
* 对原始资源的访问可能经由显式转换或隐式转换。一般而言显式转换比较安全（个人觉得一般情况下能显式就显式），但隐式转换对客户比较方便。

Item16：成对使用new和delete时要采取相同形式

记住：

* 如果你在new表达式中使用[ ]，必须在相应的delete表达式中也是用[ ]。如果你在new表达式中没有用[ ]，那么一定不要在delete中使用[ ]。

Item17：以独立语句将new**ed（new过了的）**对象置入智能指针

记住：

* 以独立语句将newed对象存储于（置于）智能指针内。如果不这样做，一旦异常被抛出，有可能导致难以察觉的资源泄露。

Item18：让接口容易被正确使用，不易被误用

记住：

* 好的接口很容易被正确使用，不容易被误用。应该在所有接口中努力达成这些性质。
* “促进正确使用”的办法包括接口的一致性，以及与内置类型的行为兼容。
* “阻止误用”的办法包括建立新类型、限制类型上的操作，束缚对象值，以及消除客户的资源管理责任。
* 这tr1::shared\_ptr支持定制型删除器（custom deleter）。这可以防范DLL问题，可被用来自动解除互斥锁（mutexes）等等。

Item19：设计class犹如设计type

记住：

* Class的设计就是type的设计。有很多需要讨论的主题都在本Item。

Item20：宁以pass-by-reference-to-const替换pass-by-value

记住：

* 尽量以pass-by-reference-to-const替换pass-by-value。前者通常比较高效，并可避免切割问题。
* 以上规则并不适用于内置类型，以及STL的迭代器和函数对象。对它们而言pass-by-value比较适当。