C 箴言：将强制转型减到最少 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/133/2021_2022_C___E7_AE_B4_E8_A8_80_EF_c97_133668.htm C 的规则设计为保证不会发生类型错误。在理论上，如果你的程序想顺利地通过编译，你就不应该试图对任何对象做任何不安全的或无意义的操作。这是一个非常有价值的保证，你不应该轻易地放弃它。 不幸的是，强制转型破坏了类型系统。它会引起各种各样的麻烦，其中一些容易被察觉，另一些则格外地微妙。如果你从 C，Java，或 C# 转到 C ，请一定注意，因为强制转型在那些语言中比在 C 中更有必要，危险也更少。但是 C 不是 C，也不是 Java，也不是 C#。在这一语言中，强制转型是一个你必须全神贯注才可以靠近的特性。 我们就从回顾强制转型的语法开始，因为对于同样的强制转型通常有三种不同的写法。C 风格（C-style）强制转型如下：(T) expression // cast expression to be of type T 函数风格（Function-style）强制转型使用这样的语法：T(expression) // cast expression to be of type T 这两种形式之间没有本质上的不同，它纯粹就是一个把括号放在哪的问题。我把这两种形式称为旧风格（old-style）的强制转型。 C 同时提供了四种新的强制转型形式（通常称为新风格的或 C 风格的强制转型）：const_cast(expression) dynamic_cast(expression) reinterpret_cast(expression) static_cast(expression) 每一种适用于特定的目的：const_cast 一般用于强制消除对象的常量性。它是唯一能做到这一点的 C 风格的强制转型。 dynamic_cast 主要用于执行“ 安全的向下转型（safe downcasting） ”，也就是说，要确定一个对象是否

是一个继承体系中的一个特定类型。它是唯一不能用旧风格语法执行的强制转型。也是唯一可能有重大运行时代价的强制转型。（过一会儿我再提供细节。） reinterpret_cast 是特意用于底层的强制转型，导致实现依赖（implementation-dependent）（就是说，不可移植）的结果，例如，将一个指针转型为一个整数。这样的强制转型在底层代码以外应该极为罕见。在本书中我只用了一次，而且还仅仅是在讨论你应该如何为裸内存（raw memory）写一个调谐分配者（debugging allocator）的时候。 static_cast 可以被用于强制隐型转换（例如，non-const 对象转型为 const 对象（就像 Item 3 中的），int 转型为 double，等等）。它还可以用于很多这样的转换的反向转换（例如，void* 指针转型为有类型指针，基类指针转型为派生类指针），但是它不能将一个 const 对象转型为 non-const 对象。（只有 const_cast 能做到。） 旧风格的强制转型依然合法，但是新的形式更可取。首先，在代码中它们更容易识别（无论是人还是像 grep 这样的工具都是如此），这样就简化了在代码中寻找类型系统被破坏的地方的过程。第二，更精确地指定每一个强制转型的目的，使得编译器诊断使用错误成为可能。例如，如果你试图使用一个 const_cast 以外的新风格强制转型来消除常量性，你的代码将无法编译。 当我要调用一个 explicit 构造函数用来传递一个对象给一个函数的时候，大概就是我仅有的使用旧风格的强制转换的时候。例如： class Widget { public: explicit Widget(int size). ... }. void doSomeWork(const Widget w). doSomeWork(Widget(15)). // create Widget from int // with function-style cast doSomeWork(static_cast(15)). // create Widget

from int // with C -style cast 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com