C 箴言：理解typename的两个含义 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/245/2021_2022_C___E7_AE_B4_E8_A8_80_EF_c97_245963.htm 问题：在下面的 template declarations（模板声明）中 class 和 typename 有什么不同？ template class Widget. // uses "class" template class Widget. // uses "typename" 答案：没什么不同。在声明一个 template type parameter（模板类型参数）的时候，class 和 typename 意味着完全相同的东西。一些程序员更喜欢在所有的时间都用 class，因为它更容易输入。其他人（包括我本人）更喜欢 typename，因为它暗示着这个参数不必要是一个 class type（类类型）。少数开发者在任何类型都被允许的时候使用 typename，而把 class 保留给仅接受 user-defined types（用户定义类型）的场合。但是从 C 的观点看，class 和 typename 在声明一个 template parameter（模板参数）时意味着完全相同的东西。 然而，C 并不总是把 class 和 typename 视为等同的东西。有时你必须使用 typename。为了理解这一点，我们不得不讨论你会在一个 template（模板）中涉及到的两种名字。 假设我们有一个函数的模板，它能取得一个 STL-compatible container（STL 兼容容器）中持有的能赋值给 ints 的对象。进一步假设这个函数只是简单地打印它的第二个元素的值。它是一个用糊涂的方法实现的糊涂的函数，而且就像我下面写的，它甚至不能编译，但是请将这些事先放在一边有一种方法能发现我的愚蠢： template // print 2nd element in void print2nd(const Camp. container) { C::const_iterator * x. ... } 这看上去好像是我们将 x 声明为一个指向 C::const_iterator 的 local

variable（局部变量）。但是它看上去如此仅仅是因为我们知道 C::const_iterator 是一个 type（类型）。但是如果 C::const_iterator 不是一个 type（类型）呢？如果 C 有一个 static data member（静态数据成员）碰巧就叫做 const_iterator 呢？再如果 x 碰巧是一个 global variable（全局变量）的名字呢？在这种情况下，上面的代码就不是声明一个 local variable（局部变量），而是成为 C::const_iterator 乘以 x！当然，这听起来有些愚蠢，但它是可能的，而编写 C 解析器的人必须考虑所有可能的输入，甚至是愚蠢的。 直到 C 成为已知之前，没有任何办法知道 C::const_iterator 到底是不是一个 type（类型），而当 template（模板）print2nd 被解析的时候，C 还不是已知的。C 有一条规则解决这个歧义：如果解析器在一个 template（模板）中遇到一个 nested dependent name（嵌套依赖名字），它假定那个名字不是一个 type（类型），除非你用其它方式告诉它。缺省情况下，nested dependent name（嵌套依赖名字）不是 types（类型）。（对于这条规则有一个例外，我待会儿告诉你。） 记住这个，再看看 print2nd 的开头： template void print2nd(const C&amp. container) { if (container.size() >= 2) { C::const_iterator iter(container.begin()). // this name is assumed to ... // not be a type