C 箴言：声明是非成员函数时机 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_133816.htm 我谈到让一个类支持隐式类型转换通常是一个不好的主意。当然，这条规则有一些例外，最普通的一种就是在创建数值类型时。例如，如果你设计一个用来表现有理数的类，允许从整数到有理数的隐式转换看上去并非不合理。这的确不比 C 的内建类型从 int 到 double 的转换更不合理（而且比 C 的内建类型从 double 到 int 的转换合理得多）。在这种情况下，你可以用这种方法开始你的 Rational 类： class Rational { public: Rational(int numerator = 0, // ctor is deliberately not explicit. int denominator = 1). // allows implicit int-to-Rational // conversions int numerator() const. // accessors for numerator and int denominator() const. // denominator - see Item 22 private: ... }. 你知道你应该支持类似加，乘等算术运算，但是你不确定你应该通过成员函数还是非成员函数，或者，非成员的友元函数来实现它们。你的直觉告诉你，当你拿不准的时候，你应该坚持面向对象。你知道这些，于是表示，有理数的乘法与 Rational 类相关，所以在 Rational 类内部为有理数实现 operator* 似乎更加正常。与直觉不符，将函数放置在它们所关联的类的内部的主意有时候与面向对象的原则正好相反，但是让我们将它放到一边，来研究一下将 operator* 作为 Rational 的一个成员函数的主意： class Rational { public: ... const Rational operator*(const Rationalamp. lhs, // now a non-member const Rational&amp. rhs) // function { return Rational(lhs.numerator() * rhs.numerator(),

lhs.denominator() * rhs.denominator()). } Rational oneFourth(1, 4). Rational result. result = oneFourth * 2. // fine result = 2 * oneFourth. // hooray, it works! 这样的确使故事有了一个圆满的结局，但是有一个吹毛求疵的毛病。operator* 应该不应该作为 Rational 类的友元呢？ 在这种情况下，答案是不，因为 operator* 能够根据 Rational 的 public 接口完全实现。上面的代码展示了做这件事的方法之一。这导出了一条重要的结论：与成员函数相对的是非成员函数，而不是友元函数。太多的程序员假设如果一个函数与一个类有关而又不应该作为成员时（例如，因为所有的参数都需要类型转换），它应该作为友元。这个示例证明这样的推理是有缺陷的。无论何时，只有你能避免友元函数，你就避免它，因为，就像在现实生活中，朋友的麻烦通常多于他们的价值。当然，有时友谊是正当的，但是事实表明仅仅因为函数不应该作为成员并不自动意味着它应该作为友元。 本 Item 包含真理，除了真理一无所有，但它还不是完整的真理。当你从 Object-Oriented C 穿过界线进入 Template C 而且将 Rational 做成一个类模板代替一个类，就有新的问题要考虑，也有新的方法来解决它们，以及一些令人惊讶的设计含义。 Things to Remember 如果你需要在一个函数的所有参数（包括被 this 指针所指向的那个）上使用类型转换，这个函数必须是一个非成员。 