Linux系统编程之C 游戏程序优化（2） PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/144/2021_2022_Linux_E7_B3_BB_E7_BB_c103_144829.htm 1.2 要前自增不要后自增（即要I 不要I ） 当写x=y 时产生的问题是自增功能将需要制造一个保持y的原值的拷贝，然后y自增，并把原始的值返回。后自增包括了一个临时对象的构造，而前自增则不要。对于整数，这没有额外的负担，但对于用户自定义类型，这就是浪费，你应该在有可能的情况下运用前自增，在循环变量中，你会常遇到这种情形。 不使用有返回值的操作符 在C 中经常看到这样写顶点的加法： Vector operator (const Vector amp.v2)这个操作将引起返回一个新的Vector对象，它还必须被以值的形式返回。虽然这样可以写v=v1 v2这样的表达式，但象构造临时对象和对象的拷贝这样的负担，对于象顶点加法这样常被调用的事情来说太大了一点。有时候是可以好好规划代码以使编译器可以把临时对象优化掉（这一点就是所谓的返回值优化）。但是更普遍的情形下，你最好放下架子，写一点难看但更快速的代码： void Vector::Add(const Vector amp.v2)注意 =操作符并没有同样的问题，它只是修改第一个参数，并不需要返回一个临时对象，所以，可能的情况下，你也可以用 =代替 。 1.3 使用轻量级的构造函数 在上一个例子中Vector的构造函数是否需要初始化它的元素为0？这个问题可能在你的代码中会有好几处出现。如果是的话，它使得无论是否必要，所有的调用都要付初始化的代价。典型的来说，临时顶点以及成员变量就会要无辜的承受这些额外的开销。 一个好的编译器可以很好的移除一些这种多余的代码，但是为什么

要冒这个险呢？作为一般的规则，你希望构造函数初始化所有的成员变量，因为未初始化的数据将产生错误。但是，在频繁实例化的小类中，特别是一些临时对象，你应该准备向效率规则妥协。首选的情况就是在许多游戏中有的vector和Matrix类，这些类显然应当提供一些方法置0和识别，但它的缺省构造函数却应当是空的。 这个概念的推论就是你应当为这种类提供另一个构造函数。如果我们的第二个例子中的Vebicle类是这样写的话： class Vehicle { public: vehicle() { } void SetName(const std::string &amp.name) { mName=name. } private: std::string mName }.我们省去了构造mName的开销，而在稍后用SetName方法设置了其值。相似的，使用拷贝构造函数将比构造一个对象然后用=操作符要好一些。宁愿这样来构造：Vebicle V1(V2)也不要这样来构造： Vehicle v1.v1=v2.如果你需要阻止编译器帮你拷贝对象，把拷贝构造函数和操作符=声明为私有的，但不要实现其中任何一个。这样，任何企图对该对象的拷贝都将产生一个编译时错误。最好也养成定义单参数构造函数的习惯，除非你是要做类型转换。这样可以防止编译器在做类型转换时产生的隐藏的临时对象。 