Linux系统编程之C 游戏程序优化（1） 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_144833.htm 一般而言，比起C程序来说，C游戏程序是可重用和可维护的。可这真的有价值吗？复杂的C 可以在速度上与传统的C程序相提并论吗？ 如果有一个好的编译器，再加上对语言的了解，真的有可能用C 写出一些有效率的游戏程序来。本文描述了典型的几种你可以用来加速游戏的技术。它假设你已经非常肯定使用C 的好处，并且你也对优化的基本概念相当熟悉。 第一个经常让人获益的基本概念显然是剖析（profiling）的重要性。缺乏剖析的话，程序员将犯两种错误，其一是优化了错误的代码：如果一个程序的主要指标不是效率，那么一切花在使其更高效上的时间都是浪费。靠直觉来判断哪段代码的主要指标是效率是不可信的，只有直接去测量。第二个概念是程序员经常“ 优化 ”到降低了代码的速度。这在C 是一个典型问题，一个简单的指令行可能会产生巨大数量的机器代码，你应当经常检查你的编译器的输出，并且剖析之。 1、对象的构造与析构 对象的构造与析构是C 的核心概念之一，也是编译器背着你产生代码的一个主要地方。未经认真设计的程序经常花费不少时间在调用构造函数，拷贝对象以及初始化临时对象等等。幸运的是，一般的感觉和几条简单的规则可以让沉重的对象代码跑得和C只有毫厘之差。 除非需要否则不构造。最快的代码是根本不运行的代码。为什么要创建一个你根本不去使用的对象呢？在后面的代码中： voide Function(int arg) { Object boj. If(arg==0) Return. ... }即便arg为0，我们也付出了调

用Object的构造函数的代价。特别是如果arg经常是0，并且Object本身还分配内存，这种浪费会更加严重。显然的，解决方案就是把obj的定义移到判断之后。 小心在循环中定义复杂变量，如果在循环中按照除非需要否则不构造的原则构造了复杂的对象，那么你在每一次循环的时候都要付出一次构造的代价。最好在循环外构造之以只构造一次。如果一个函数在内循环中被调用，而该函数在栈内构造了一个对象，你可以在外部构造并传递一个应用给它。 1.1 采用初始化列表 考虑下面的类： class Vehicle { public Vehicle(const std::string amp.name):mName(name) { } private: std::string mName. }