用C设计，用C 编码 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/268/2021_2022__E7_94_A8C_E8_AE_BE_E8_AE_A1_c97_268070.htm 昨天晚上看到刘江的blog又补充了好几大段，今天早上又看到云风的人肉trackback，果然还是这种话题引人关注。 云风先是提了一下所谓C 带来的思想包袱(文言文曰“ 心智包袱 ”)问题，然后重重地引用了Linus的话：“ 关键是设计 ”，其实他是在暗示：好的设计C同样能做出来，不劳C 大驾.而C 一旦出面，就要让人背上额外的思想包袱。 我明确地表个态，在系统级程序设计中，事实就是这样的。 别小看这个思想包袱，大部分，甚至绝大部分C 程序员过不了这一关。相反，做系统级开发，C是几乎没有思想包袱的语言，说白了就是刺刀见红，你想要啥你就去写啥，它给你的不多也不少，没什么干不了，也没什么非让你背着不可。 我早在N年前就发现自己写程序速度慢，我当时对STL远比周围人熟悉，照例说长缨在手，应该效率 很高才对。结果发现不是，写程序的时候特别没自信，总在想：“ 这样固然是可以work了，但恐怕有更好的方案吧，会是什么呢?加个模板参数试试?要么抽象 出一个基类?做一个bridge模式?那么Ownership的问题怎么解决?谁 来负责回收内存呢?移植一个boost::shared_ptr过来吧!可多线程情况下会不会拖慢速度呢?应该不会，可是会碰到循环引用的情况。要么在中 间搞一个weak_ptr把循环链断开?哎呀不行不行，太复杂，别人也理解不了。还是先这样吧，能work就行。 ”就这样，兜了一个圈子回来。有的时候，这个圈子不是纯柏拉图式的，我会真的实现不少“ 优化 ”设计来比对，那个时

间啊，花花的就耗在里面了。有的时候确实会获得一些改进，但是多数时候是得不偿失，旁边那些在我看来连C都只是一知半解的家伙采用“ CtrlC-CtrlV-Modify-Debug ”大法，早就冲到我前头去了。这就是“ 心智包袱 ”的威力。 最近几年没怎么用C 写程序，业余时间倒是别的语言用了好几种。大概是体会到这些语言的某些好处之后，对C 就能看得更客观一些了，也琢磨了一下，如 果自己有朝一日重新跑回去写C/C ，我会怎么干?毕竟现在C 程序员全球紧缺，工资越来越高，这个问题还是有其现实意义的。正好昨天跟chensh 聊了一会儿，两个人的看法一致，就是采取“ C Concreate Class STL ”的风格。说白了就是用C来设计，用C 来编码。 这里面的道理是这样的，反正现在C和C 都是来做系统级开发，那些华丽的抽象机制用不上，思考解决方案的时候，就以C的方式。注意，C也是可以做基于对 象甚至面向对象甚至组件级别的设计的，但是在C的层面上思考问题，设计能够更精益(lean，现在这是个时髦词)，更轻便，更直接。当你构思的设计方案出 来以后，如果其中有些部分，恰好是C 现成做好了，而且使用C 又可以提高开发效率，也没什么明显的副作用，那么就用C 来做相应的部分。比如， COM原来设计的时候就是在C基础上做的，设计的时候发现实际上跟C 实现多态的的vptr vtable是吻合的，所以后来就主要用C 来做COM开发。事实上，为了适应COM开发的需要，微软直接改了C 编译器。很显然，微软是首先构思好 的设计，然后让C 去适应这个设计。而后来很多C 程序员，是让设计去适应C 的那些语言机制，在系统开发中，这个叫做本末倒置。当然这样的事情在 应用级别上就不是那么离谱。 实际上回头看看C 早期的

历史，最早C 就是把一些C中常用的patterns内置到语言里而出现的，早期它曾经有效地提高了开发效率。今天应该回头去寻找这种精神。 我支持STL是基于同样的理由。很多时候，你从C出发得到的设计，也无非就是STL已经实现得很好的东西。在这个时候，当然可以用STL。尤其是那些算 法，针对C array也是适用的，用accumulate求和，用transform映射，用adjacent_find寻找相等的毗邻项，用 lower_bound 和equal_range做二分查找，等等，这不是比手写要爽多了吗? 当然，使用STL，还是必须熟悉其背后的机理，没有这个底子，还是规规矩矩用C算了。 