C 的救赎C 开源程序库评话 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/291/2021_2022_C___E7_9A_84_E6_95_91_E8_c67_291934.htm C语言天生就与开放结缘。C最初是作为UNIX的系统编程语言而流行起来的，而UNIX可以被认为是第一个产生重大影响的“ 开源 ”软件。随着UNIX的流行，C语言逐渐被人们认识和喜爱。很快的，在各个平台上C语言都成为了流行的甚至是统治性的程序设计语言。大约到1980年代中期，C已经成为人类历史上第一种工业级程序设计世界语。很多人都知道，正是C这样一种世界语的出现，才使开源运动的出现和最初发展成为可能，从这个意义上讲，说C语言是开源运动之母并不十分过分。但人们不太能够认识到的是，事实上C语言统治地位的获得，却也是早期开放软件运动的直接结果。多数人在回顾这段历史的时候，经常会感染中国文人的不严肃的浪漫主义史观，喜欢把C语言的成功归结为汉高祖斩白蛇般的天赋神格，描述为遥想公瑾当年，谈笑间樯橹灰飞烟灭的轻飘飘。 然而如果我们对历史作一些细致的调查，我们会发现C语言绝非有什么天命，而只不过是幸运地扒上了早期开放运动的快车而已。在C语言“ 小人乍富 ”的那几年，也还有其它不少程序设计语言具有高性能、可移植、系统开发能力强的特点，决不是只有C骨骼特异，貌若天仙。如果Pascal也能借助一个像UNIX那样的开放的幽灵在欧美大学校园里徘徊，那么我们今天很可能要把begin和end直接映射到键盘上。 如果IBM不是在1970年代极端保守地把一种叫做PL/X的语言牢牢地限定在自己的研究所里，也许整个程序员社群的图腾就不是贝尔试验室的那

两个大胡子，而是小沃森实验室里的IBM某院士。事实上，C语言的成功，更须拜开放软件运动之时势所赐，或者更确切地说，C与开放软件是一对共生体，它们相互扶持，相互成就，共同成长兴旺，共同创造历史。 根深自然叶茂。今天C语言体系内所拥有的开放资源，无论是数量和质量，还是丰富性、多样性、创新性、可靠性、重要性，都是其它任何开发技术体系所无法望其项背的。丰富对于开发者是好事，但对于写资源介绍性文章的作者来说，则是绝对的坏事。想要对C语言体系中的开放资源做一个介绍，哪怕只是一次白描，也决不是一个人、一本书所能容纳的，更远远不是杂志中的一篇文章所能及的。因此在本文中，对于C语言开放资源的介绍是以一种蜻蜓点水的姿态进行的。 相比之下，C 语言在开源世界中的分量，与C语言相比就相去甚远了。作为对照，C 语言在工业界的实际地位，如果不是比C更重要的话，至少也是与C在同一个层次上。考虑到这一点，在开源领域中两兄弟的这种差距就令人感到非常震惊。如果说在2000年以前，由于C 在工业界的统治地位，这种差距对C 的影响还不大的话，今天，C 在开源领域里薄弱的基础就非常要命了。现在在开发者社群中，“ C语言万寿无疆，C 无寿无疆 ”的说法得到不少人的支持，其根本原因之一就在于C 在开源运动中的地位远逊于C。究其原因，归根到底是因为编写高质量、可复用而又拳拳服膺的C 程序库实在是一件太困难的事情。一方面，大量的C 开源项目质量不佳，而且经常以一种粗暴的方式要求使用者改变自己程序的风格，另一方面，一旦有人完成了一个可用的C 项目或者程序库，他必须具有极其彪悍的意志才能够咬着牙把这样的项目奉献给开源社群不

仅因为失去了可能的金钱上的回报，更因为可能要面对着暴风雨般的批评和鄙视。总之，诸多的原因使得开源文化未能在C 中深深扎根。 然而，毕竟C 是一种称霸一时的语言，C 社群的规模、强悍和创造力，仍然是很多其它新兴语言社群难以相比的。特别是在标准C 制定之后，C 编程风格有了明确的指导思路，开源项目也就大大繁荣起来。虽然时间还不长，但是已经有一些令人欣慰的成果。这些成果也就构成了写作本文的基本动机和素材。 就重要性而言，开源程序库和工具集对于C 甚至比对C还要重要得多。因为实践证明，没有良好的基础设施支持，C 开发成功的可能性异乎寻常的低。其根本原因是，用C 写优秀的程序库非常非常难，而一旦有了这样的程序库，在其基础上写应用程序就相当容易了。同时，C 的特点又要求基础设施的源代码必须开放，因此，C 程序库对于开发者来说意义非常重大。 我们可以更进一步探究开源C 程序库对于C 开发的重要意义。用C 编写可复用程序库时所需要的思想方法和技术风格，与用C 编写应用程序时所需要的思想方法和技术风格之间存在相当大的差异和差距。前者所需要的高超技术、丰富经验和良好的权衡能力，是很少有人能具备的。在所有程序设计语言中，你恐怕找不出第二种语言像C 那样，对于程序库作者的要求如此之高，以至于远远超过了一般“ 熟练 ”C 开发者的平均水平。在Lisp 中，语言、库和程序根本就是一回事，每一个程序员写的代码都可以看成是语言本身的扩充。 在Java、C、Perl、Python、Ruby中，一个优秀的应用程序开发者在积累一定经验之后，不难写出高质量的可复用代码。而在C 中，这种事情是非常罕见的，即使是天资卓越、经验丰富的大师级人物，也需

要花费多年的打磨，历经几次反复，才能够最终推出受到一致认可的可复用程序库。此道之难，难于上青天，以至于Andrei Alexandrescu感叹道，十几岁的少年天才满目皆是，满鬓斑白的优秀程序库设计者凤毛麟角。而在另一个地方，一本C 可复用技术图书的作者总结道，所谓可复用的C 程序库，不可能是设计出来的，只可能是复用出来的。然而，一旦这样的程序库构造出来并且为人们熟悉，那么就会大大地简化应用程序的开发。这也就是为什么在2000年后，Bjarne Stroustrup无数次地呼吁社群专注程序库的开发。他很清楚，只有程序库能够救C ，只有程序库能够发展C 。 现在我们知道，用来写C 程序库所需要的技术，与用来写C 应用程序所需要的技术存在很大的差别。这已经比较糟糕了。更糟糕的是，一般的C 开发者根本分不清这中间的差别，他们在开发中往往既不是一个称职的程序库开发者，也不是一个单纯的应用开发者。他们一边想着完成手头的工作，一边琢磨如何能够写出高质量的基础库和框架，为万世开太平。如果说C 语言是一把轻快的小匕首，遇谁都是进身猛刺，血溅一尺，那么这种C 的使用方式无异于左手打铁铸兵，右手挥剑刺秦，这种精神分裂的状态直接将很多项目变成了既超期超支又质量低劣的垃圾。 认识到这样的事实之后，C 程序员应当以更理性的态度来看待自己的工作。大部分情况下，你所需要做的是寻找一些可以互相合作的、稳定可靠的开源程序库，然后在其基础之上，面向目标，使用尽可能简朴的技术，专心专意地进行应用开发，把那些复杂精妙的语言技巧和“ 可复用 ”之类的想法扔到Java国去。唯其如此，你才可能更高效地开发出好的应用软件，而且会逐渐积累和重构出真正可复