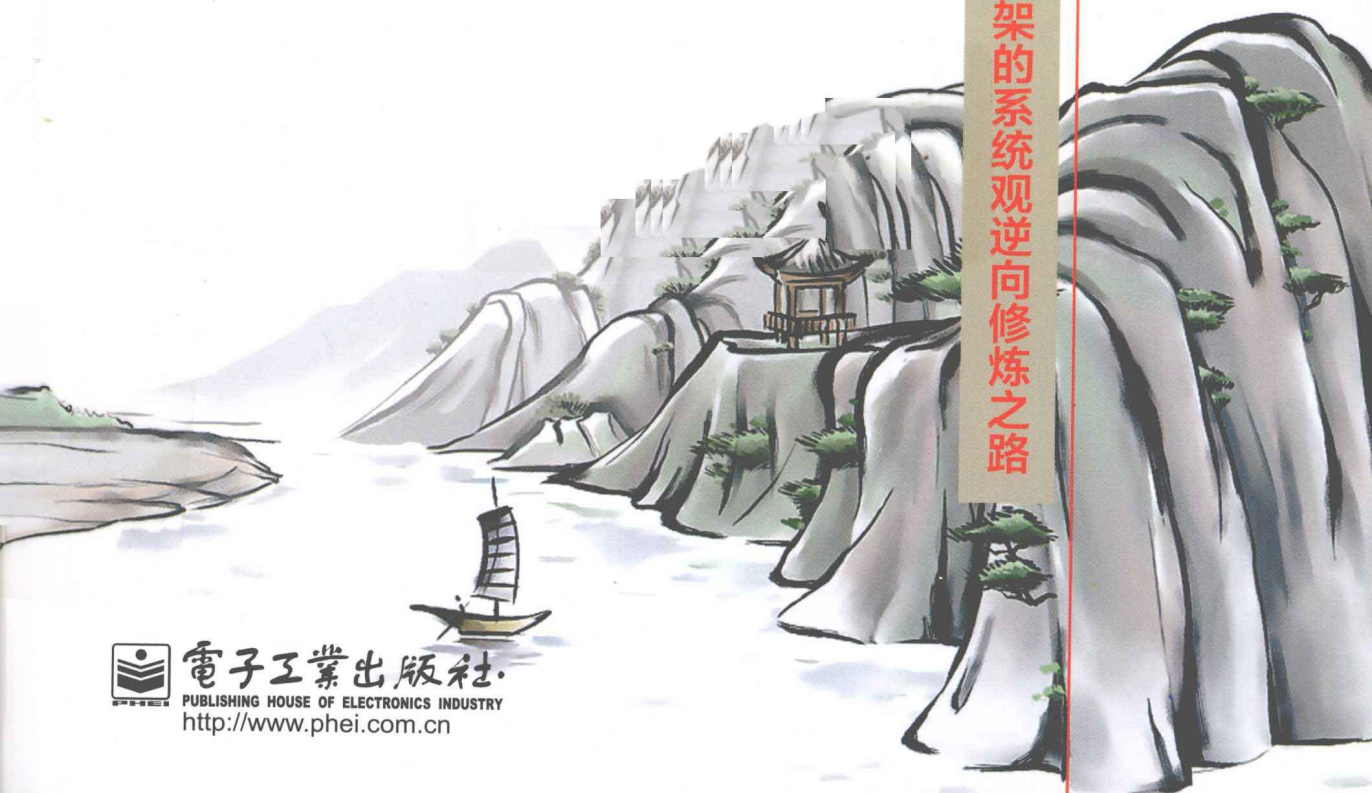
老『码』识途

韩宏 李林 ◎ 著

从机器码到框架的系统观逆向修炼之路

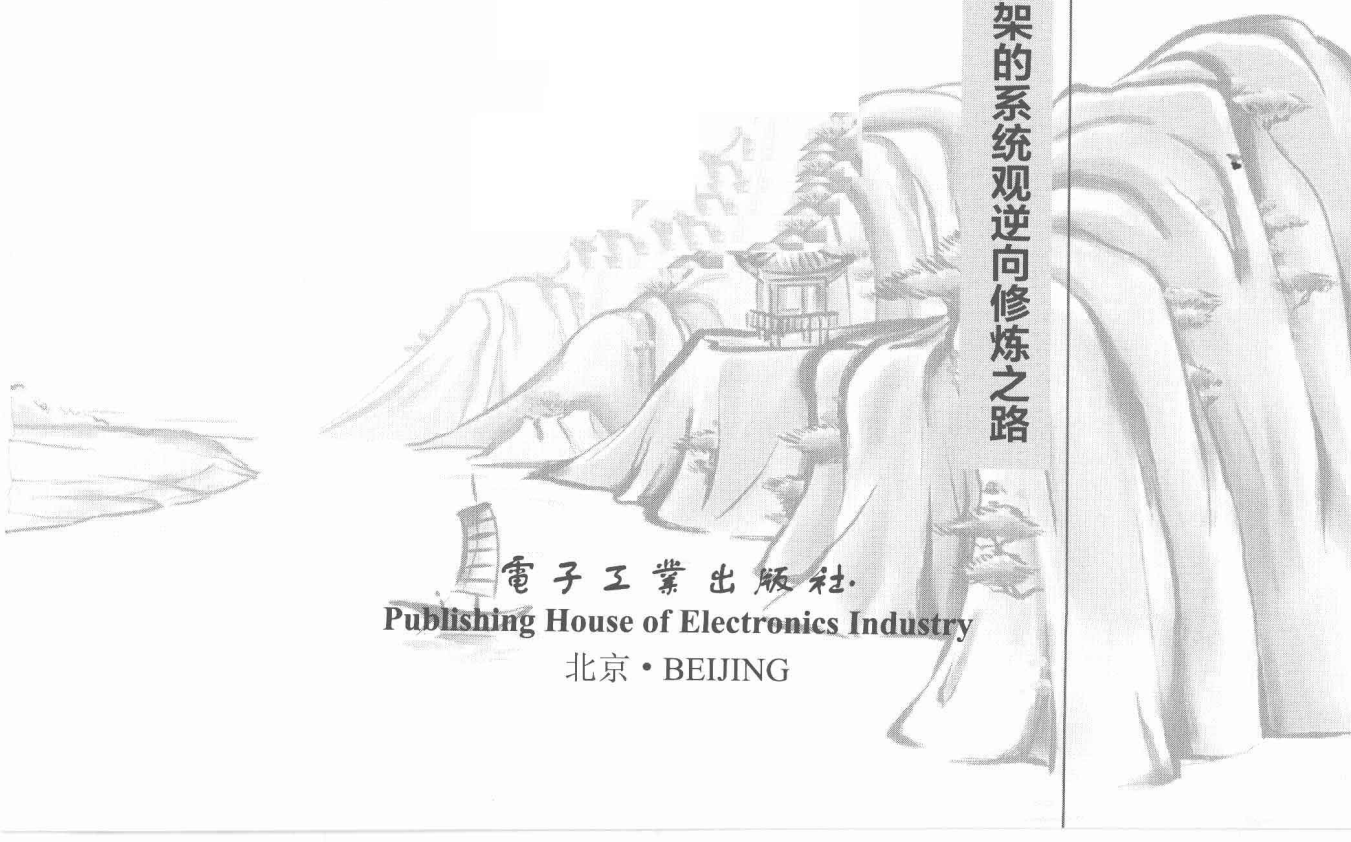


電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

老『码』识途

韩宏 李林 ◎ 著

从机器码到框架的系统观逆向修炼之路



電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书以逆向反汇编为线索，自底向上，从探索者的角度，原生态地刻画了对系统机制的学习，以及相关问题的猜测、追踪和解决过程，展现了系统级思维方式的淬炼方法。该思维方式是架构师应具备的一种重要素质。本书内容涉及反汇编、底层调试、链接、加载、钩子、异常处理、测试驱动开发、对象模型和机制、线程类封装、跨平台技术、插件框架、设计模式、GUI框架设计等。书中包含不少工业级或非公开案例。读者不仅能以底层观和调试技巧解决各种实际问题；还可掌握一套学习方法，如“猜测—实证—构建”，调构学习法。

图书在版编目（CIP）数据

老“码”识途：从机器码到框架的系统观逆向修炼之路 / 韩宏，李林著 . —北京：电子工业出版社，2012.8

ISBN 978-7-121-17382-0

Ⅰ . ① 老… Ⅱ . ① 韩… ② 李… Ⅲ . ①程序设计 Ⅳ . ① TP311.1

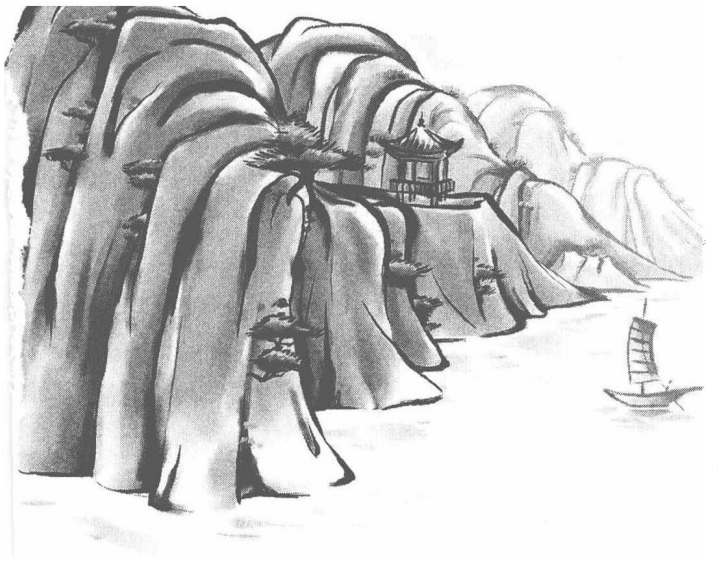
中国版本图书馆 CIP 数据核字（2012）第 136517 号

策划编辑：章海涛
责任编辑：章海涛　　特约编辑：何　雄
印　　刷：三河市鑫金马印装有限公司
装　　订：
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱　邮编　100036
开　　本：787×1092　1/16　　印张：21.5　　字数：560 千字　　插页：4
印　　次：2012 年 8 月第 1 次印刷
定　　价：56.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。



序 Preface

——“码”途有道何为径

你主要能得到什么

如果你想成为高级程序员或软件架构师，什么才是技术上的核心竞争力？仅仅是知识？在这个随时可求助于谷歌和百度的年代，知识似乎已变得非常廉价了。而青春的流失并不能给我们留下技术财富，似乎只是将我们变成自嘲的“码奴”。

核心竞争力究竟在哪里？本书认为，一个关键要素就是“系统观”，这是高级软件人才必备的素质。系统观是美妙的，它能自我生长，自我完善，有了它，你就拥有了一颗能成长的原核。系统观是核心知识架构和对它们不断运用所形成的思维方式的复合体。本书的首要任务就是帮助大家建立系统观。

◎ 本书采用独特的自底向上贯通的方式帮助读者完成系统观的构建，从反汇编、机器码入手，以逆向分析贯穿，运用一点点增加的知识不断探索出新知识并最终上升到框架。

◎ 本书采用生长式的学习方法，每个后续章节都是运用前面内容探索而来的，既让你制造“砖头”，又让你用它建造“大厦”。读完本书后，你不仅能掌握一个小而全的可自我发展的核心知识架构，更能掌握一种被笔者称为“猜测—实证—构建”的系统观的思维与学习方法。你将经历一次畅快的知识探索的发现之旅。

你还能得到什么

学完本书后，除了系统观这一主要成果，你还能获得一些有意思的能力，比如：

◎ 掌握被称为“调构学习法”的第三方源代码学习方法（成为架构师的重要能力）。

◎ 在C语言中模拟面向对象的继承、封装、虚函数覆盖等。

◎ 用病毒常用的自定位代码技术解决框架级软件的优雅构造。

◎ 利用钩子技术，在只有执行程序的条件下改变第三方软件的行为。

◎ 运用链接原理自己动手链接OBJ文件，使其可执行。

◎ ……

对笔者而言，这是一本关于“探索知识、让知识融入生命”的书；是一本关于能力与学习的书；是一本展示如何将看似琐碎的知识碎片不断打磨成珠玉，慢慢串成美丽珠宝的书；

是一本展示编程技艺如何自我锻造的书。也可以看成是与大家分享码途人生的飞絮呓语。

学什么？少即多的系统观

学什么？记得在Internet还没有成为主流的时代，有个学通信的硕士被一个IP路由器问题困扰了一个多星期。笔者最好的朋友，学空气动力学的，他玩了10多年DOS系统，从没实际接触过网络，只是“看”过一本资料，还不是IP网络，仅花两天就解决了这个问题。为什么有人学了很多技术和语言却不如未学习该技术的人呢？该问题另一个问法就是“什么才是我们最需要掌握的？”

笔者的看法是：“少就是多，将这个少变成多”。下足工夫，将“少”做深、做厚、变多，就是将来应对千变万化的舵。这个“少”就是计算机的系统观。看看上面的例子，那个朋友对DOS的熟悉程度已经到了对任何一段地址空间作用都了如指掌的地步，自己还实现过一个汉化的DOS。他无疑是建立起了坚实的系统观。这个系统观能指导他按照计算机的“逻辑”方式思考，而路由器不过是这种思维方式的一个例子罢了。有了系统观，学习新东西时，用它去猜测、构想学习对象的可能做法并实证分析，是一种迅捷的办法。笔者在处理SaaS软件中的hibernate将同一持久化对象存入不同库的问题时就是这样，虽然没用过hibernate，但通过猜测和调试，用了约半天时间就解决了问题（搜索并非银弹，该问题当时在网络的中英文查询中均未查到）。

到底什么是系统观？有些东西只可意会难以言传，且见仁见智。这时，纠结含义不如能够运用。诗词意境颇似系统观，诗人自身可能都说不清其内涵，可他能运用这种意境，“运用为王”。其实，我们只要能建立系统观就好了。

如何建立？一种方法是从无到有自己撰写一个系统级软件（哪怕是实验性的），如嵌入式数据库、小型操作系统或编译器。这非常有效，那位朋友得益于撰写汉化DOS的经历，在没有学过嵌入式的情况下，一个月时间就解密了一个单片机太阳能系统，并汉化。但这种方法需要较长一段时间完全投入，且难度不小。针对这些问题，本书给出了另一种方法。

本书内容：逆向入道，自底向上建立系统观

在建立系统观方面，本书没有选择正规的站马步的方式——构建一个操作系统或编译器，而是提供了一种剑走偏锋的方法——自底向上，以逆向反汇编入道，贯穿从机器码至框架的学习。因为，逆向是一把匕首，小巧，却能劈开黑箱直见根本，实乃实证利器。

本书涉及范围既广也窄。从“广”来说，覆盖了汇编、反汇编、逆向、调试、链接、线程、插件风格编程、设计模式、对象模型和机制、框架源代码学习等内容。从“窄”来讲，本书紧紧围绕运用基础知识这把“瑞士军刀”，披荆斩棘，最终建立系统观这一主旨。同时，本书不仅让我们多了一种能力（书中各章节均给出一些调试技巧），更将OS、Compiler等相关知识从压箱底处翻到台面上，并通过逆向思维有效整合在一起。你会发现，这些原来似乎抽象的知识，完全渗透到我们破疑解惑的过程中了。

本书由7章构成，其中第6章（除6.5节）为李林著，其余为韩宏著。

第1章是最重要的基础，以逆向反汇编角度建立了语言的物理模型，充分利用GUI调试环境，将所有知识点做到可“把玩”。例如，在分析mov指令机器码时，通过断点，并利用内存窗体修改机器码，从而改变C语言赋值语句的效果，也展现了一种学习汇编的方法——基于分

析RTL（runtime library）的学习方式。本章还充分展示了“猜测—实证—构建”的思想。

第2章展现基础知识的底层力量，给出多个运用前章逆向知识解决问题的例子。其中有的例子应该是首次公开的。本章揭示出逆向分析的力量，不仅可运用于加密、解密，而且真的能指导我们解决平常的开发难题。部分内容覆盖了钩子、异常、注入等技术。

第3章呈现以构建为驱动的软件开发学习方式，知识覆盖链接器、测试驱动开发、数据结构、软件重用、文件访问等。内容有其特点，如以构建的角度学习了链接器相关知识，并创建了一个最小“Linker”演示。在后续动态链接库运用中，以此为基础解决了棘手问题。

第4章以创造面向对象语言方式，展示了对象模型、机制及其重要特点和使用模式，涉及构造、析构、基于栈构造技巧、虚函数机制等。这种创造的方式集中体现了猜测、实证的思想，在理解的基础上深入探讨了在C语言中如何运用面向对象思想，给出了相关宏；并分析了一些重要设计原则及其在C++中的利弊和不同语种的权衡。以逆向思维贯穿所有问题的解决是本章的特点。

第5章构建了一个跨平台线程类，充分利用了第4章的知识，最终创建了跨平台线程库，并展示不同语言下虚函数机制导致的软件设计差异以及bug的解决过程。从设计到debug的过程体现了系统观和底层调试的重要性。

第6章在第4章基础上展示了一个实用插件框架的演化，在不修改已有代码的前提下，运行时为系统添加新功能。其中，DLL内存释放bug的解决方案充分展示了调试和系统观的力量。

第7章讨论了阅读优秀源代码的方法与技巧。这不仅是程序员成长的重要手段，也是开源时代的重要能力（对一个框架，在资料匮乏之下，你必须快速扩展框架，加入自己需要的功能）。以分析VCL框架为例，并对比MFC，结合框架阅读和创建，展现了被笔者称为“调构学习法”的源代码学习方法。调试、追踪、分析了自定位机器码解决this指针获取的问题，并模仿这一机制创建了可重用的窗体控件框架。本章将猜测—实证—构建思想贯穿到了框架源代码学习和构建中。

我的码道：三步斩“码”刀

修炼系统观不是简单知识的积累，需要一种“道”。笔者的“道”就是：猜测—实证—构建。

“猜测”让我们主动思考，发现问题。这样学习不再是按部就班，而是“寻找式”，对症下药，有选择地吸收，迅速且准确，直指症结。如1.3.1节学习call指令时，不直接看其解释，而是先分析机器码，猜测哪部分可能包含转跳地址，最后经多次猜测实证才搞清楚。这样，将一般被动接受式学习变成了主动研究式的破疑。简单地说，学习任何不懂的技术或框架前先不要急于看它怎么做，而想想自己怎么做，要想到可实证的细节，否则无意义。

“实证”体现为每个知识点都可触摸，或调试、或编程、或使用，能验证。许多人猜测后认为一些东西“很显然”，而不验证，此时“真知”就从指间溜走了。如2.9.3节猜测状态保存时，细分了局部变量、参数，并考虑优化。实证与猜测相悖处即产生前进动力：我们常感叹自己不善提问，而实证就是最好的设问及解答的过程。实证时往往一个知识点牵扯了很多复杂因素，看来几乎无法实证。此时关键就是“领悟精髓，删繁就简”。本书第3章展示了如何通过摸索理清链接的关键逻辑，构建一个2~3天能完成的Linker核心小程序。

“构建”要求尽量将学到的知识通过编写程序模仿出来。这非常有助于以系统观思考问题。而构建中会出现意想不到的问题，它们又是进步的阶梯。比如，多年前学习《Windows

核心编程》中内存访问API时，笔者抛开书上的例子，用这些API编写了游戏修改器。之后就发现了新问题：遍历整个线性地址空间寻找某个整数耗时太长，速度完全无法与真正的修改器比拟。最后，基于最基本的对齐原理解决这一问题，就不用挨个字节搜索了。基础知识在一个个构建、意外及调试解决中被贯穿起来，终成系统观。本书每章均有展现构建的威力。

其实，“码”途之道最重要的就是“快乐”。有了它，你就有了自己的“道”。在本书中，你能看到面对问题的兴奋和努力探索的快乐。如第2章最后两个例子，就是笔者大年初一连续几天疯狂逆向的结果。一个个失望和惊喜后，明媚的阳光穿过0/1 bit的迷雾晒到皮肤上，这就是生命的味道。

如何使用本书

读者有一定的C和C++语言的基础知识就可上路了，你不需要很懂汇编，我们会一点点探索。中途遇到什么概念不懂，可上网查阅学习。既然强调体验式，你就将本书当成游戏攻略吧，你一定要在计算机上“玩”成这个“游戏”，而不是在床头“看”它，要通关哦。随书附带了源代码，但你不要直接用，请自己实现一次，将源代码当做答案吧。（所有源代码并没有采用附盘的形式提供给读者，而是采取网站免费下载的方式，读者可以登录http://www.phei.com.cn或者http://www.hxedu.com.cn进行下载。）

本书以VS 2008为主要调试环境，还用到其他工具：Delphi、Ultraedit、PELord、PEView和IDA pro。

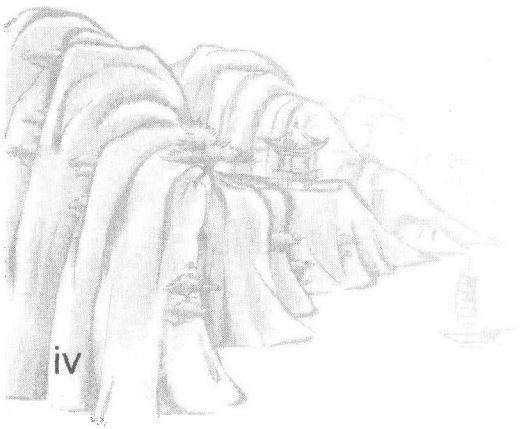
书中解决问题的体验式过程只从文字看（特别纠缠了反汇编后）容易迷失，所以每章都挑选了一个较难的案例编写了思维导图（除第7章外），读者可配合使用。通过导图的符号能索引正文；通过页侧的标记又可快速定位导图中位置，以便来回参阅，理顺思路。

结束本文前，用一首打油诗以抒心意：

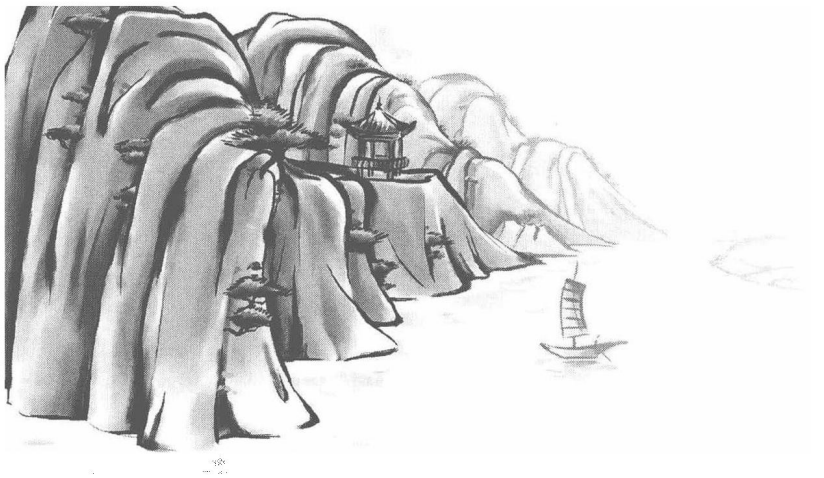
人被“码”驯为码奴，
何日驯“码”骋康途。
老“码”识途途何在，
三步斩“码”刀自出。

我的斩“码”刀就是“猜测—实证—构建”，你的呢？

最后感谢家人、朋友与学生（特别是陈松进行了细致的校对），他们在著书的过程中给予了笔者莫大支持并提供了宝贵意见。还要致谢高辉老师，他就strlen算法给出了证明。



韩 宏
二〇一二年八月



如何阅读本书

1. 通读式阅读

如果你是有一定基础但缺乏用底层的视角建立系统观的经验和经历，请从头到尾完全按照书上所展示的实验一步步分析，最好能自己提出新问题并验证。如果确有难以解决的问题，可发到作者邮箱hhan@uestc.edu.cn。对于常见问题，笔者也会在博客中与大家分享、探讨，http://blog.sina.com.cn/u/2103052793。本书更重体验式，所以单从简单阅读来看，多会产生一种知识点散漫之感，但如能静心用程序实证，所有的方法和系统感自会悄然建立。真正的知识探索过程本就是多线索、离散化、过程式的，在一种反复的淬炼中自然会升华，这也是工程性手艺活的特点。最典型的就是中国功夫的训练，或站桩，或一招两式，反复打磨，似乎全无体系，一朝顿悟，自能豁然贯通。这是东方式的体悟和西方式的解析解构的差别。

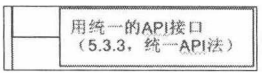
当然，如果你在经历了实证过程后，还需要有人引领你提炼出书中的知识架构，可参考第2种方法。如果你在之后需要快速查找一些知识点，可参考第2种方法。

2. 快速索引式

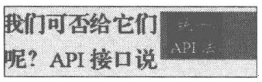
如果你是教师或有很深厚功力的读者，希望从本书中快速查找自己感兴趣的部分阅读，可参阅本节提供的知识导图和索引，快速找到知识点。

总体知识导图是笔者总结的本书所涉及的知识点的架构，可帮助读者回忆和总结，由三部分组成：一是简略总图，能帮助你了解其概况；二是较细的总图，其中主要知识点均有相关索引标注，你能快速跳到感兴趣处；三是分图，针对各主要部分展示最全面的知识索引和架构。

标注分两种：一种是直接给出章节号，另一种是给出章节及其中的关键词索引。后者使用方式如下：例如，知识导图中某知识点如下，先说明该知识点在5.3.3节中，关键词是“统一API法”，然后在5.3.3节正文的外侧查找如下所示的图标，“统一API法”标注的行就是相关内容。



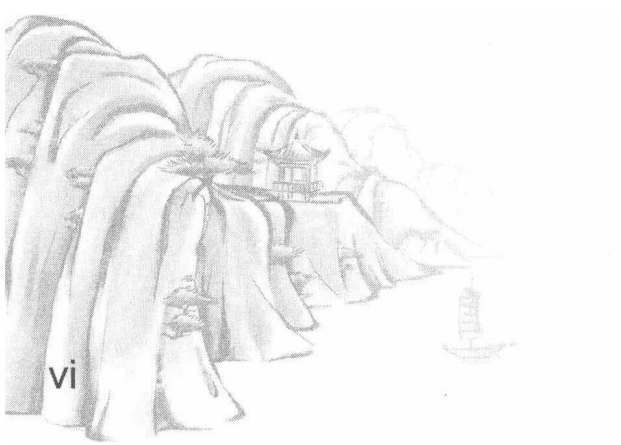
知识导图的标注



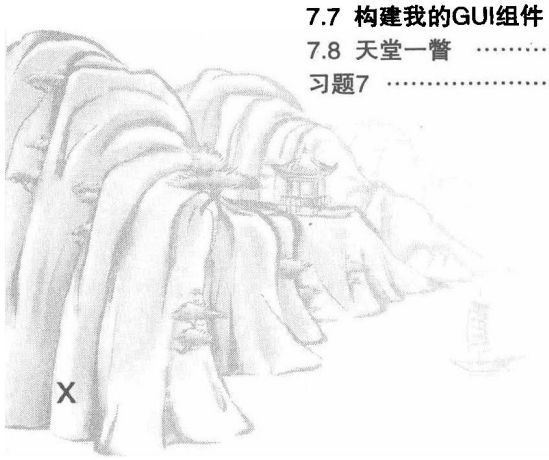
正文中的标注

3. 章首页

每章都有首页，其中展示了本章所需前导知识点及其所在位置，以及本章对后续章节知识的支撑作用。如果你直接切入到某章，“需准备的知识”辅助你有效阅读，“为你提供的能力则”让你明了本章内容有何价值。



目 录 Contents



第1章

欲向码途问大道，锵锵bit是吾刀

需准备的知识

- 了解一些计算机的基本结构，如CPU、寄存器、内存……
- 基本掌握C语言。
- 了解一点汇编更好。

为你提供的能力

- 本章的调试技巧 → 支撑整本书的底层调试和探索。
- 1.1.3节和1.8.2节的机器码构造能力 → 2.1节解密之hello world和7.4节的构造自定位代码。
- 1.2节指针机制 → 2.3节指针的指针bug、5.2.2节线程函数的this指针传递。
- 1.3节的函数调用机制 → 2.1节、2.2节、2.3节、2.4节、2.6节、2.7节、2.8节和2.9节的错误调试和逆向分析，3.3.5节的bug分析，4.1.6节和4.7.6节的this指针传递机制和虚函数机制分析，5.1.4节的bug分析，6.5节内存释放错误分析和7.4节分析自定位代码。
- 1.4.1节数组模型 → 2.2节的bug分析。
- 1.4.2节结构体模型 → 4.1节的this指针机制，4.3节对比C中“对象”和C++对象，4.6.3节继承机制和4.6.4节在C中玩面向对象。
- 1.5.1节对齐 → 1.5.2节的bug，1.9节的分析strlen中的对齐读，3.3.3节的结构体对齐bug。
- 1.6节的switch和1.9节的分析 → 学会一种分析方法和汇编学习方法。
- 1.8节的加载期重定位 → 3.3节的链接期重定位的理解和程序构造。
- 充分理解猜测、实证和构建方法。

如果你是一个穿越小说的主角来到计算机世界，面对着像尘沙般枯燥简单的0/1 bit、五光十色的软件高楼大厦和像烟火一样绽放又凋谢的各种时髦技术，你的路在哪里？你怎样把握根本大道，创建一个属于自己的王国？是尘沙，那看似枯燥的底层程序观就是你真正纵横捭阖的根基。你想在面对一个没有源代码的黑盒似的执行程序时，为它点石成金般地添加新功能吗？那是上帝说要有光，即有光的神奇。你想在众人困惑于某个诡异bug时，以逆向之刀，游刃于乱花迷人眼的反汇编迷网，一记小李飞刀直指那乱网后的真相吗？那是红尘游侠寂寞的潇洒。你想看透各种辉煌技术的本质，举重若轻吗？那是羽扇纶巾的将帅风华。这些能力都来自本章的底层程序观，它是你的马步，你的正法眼。

来看看本章的功效。如果你从1.1节和1.8节中“玩”会了构造机器码，就能理解并完成第7章的构造自定位代码，解决this指针的传递问题。如果你领悟了1.3节中函数调用的所有细节，就能自己解决2.4节语言互通的奇怪bug。如果你对1.3节和1.4节有了把握，就奠定了整个第4章对象模型的基础。第4章从底层剖析了对象，又为第5章巧妙的线程封装和第6章中从DLL导出对象提供了根基。这条通天大路，环环相扣，均根植于本章的点点滴滴的淬炼。

好了，我们的主角，开始枯燥、琐碎又起伏跌宕的修炼之旅吧。

1.1 全局变量引发的故事

1.1.1 剖析赋值语句机器码

我们从下面这段简单地为全局变量赋值语句的反汇编开始：

```
int gi;
void main(int argc, char* argv[])
{
    gi = 12;
}
```

使用VS 2008作为调试环境。将光标放在“gi=12;”语句行上，然后按F9键，在该行上设置断点（如果继续按F9键，将消除该断点），出现如图1.1所示的标志●。

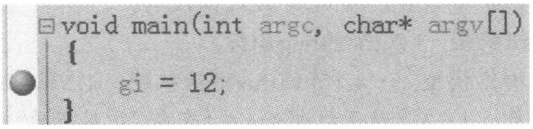


图1.1

> **断点（break point）**：当程序运行时，经过该点所在语句，程序将暂停运行。可在此时观察程序的各种状态，如变量值、内存值、寄存器，甚至可以修改这些相关状态。可以在程序运行前设置断点，也可以在运行中设置断点。

然后保证程序为Debug模式（默认模式），如图1.2中椭圆圈定部分。因为在Debug模式下可进行代码的调试跟踪。VS 2008的Release版本可调试，因为生成了调试信息，而VC 6.0必须手动配置，否则无法调试跟踪。

按F5键，进入调试运行方式（按Ctrl+F5组合键，为非调试运行方式，设置的断点无效）。程序在断点处暂停，见图1.3。

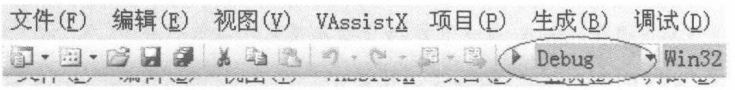


图 1.2

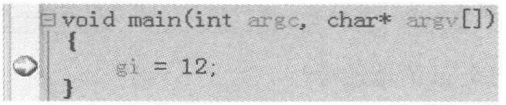


图 1.3

选择菜单的“调试→窗口→反汇编”命令（见图 1.4），进行反汇编（disassemble）（快捷键Ctrl+Alt+D），反汇编的结果见 DM1-1。

图 1.4

第2行是第1行对应的汇编码，左边的数字0041138E是十六进制数表示的赋值指令mov的存放地址，右边是该条指令的汇编表示形式。（**说明：内存地址和其中的值均采用十六进制数表示，在反汇编中一般省略末尾的H或h。全书同。**）学习过汇编的读者会奇怪，全局变量的名字gi怎么会出现在汇编指令中呢？汇编语言没有变量名。因为在VC环境中，为了易理解，开发环境特意将操作对应的符号名也显示在语句中。为了看到纯正的汇编语句，我们选择暂时将符号名从反汇编语句中去除。在代码窗口中单击右键，在弹出的快捷菜单中选择“显示符号名”，取消其选中状态（如果需要显示符号名，则再次单击），见图1.5。此时，反汇编结果见DM1-2，mov指令中没有C语言的变量名gi了，是把0ch（h代表十六进制，0c即十进制数12）赋值给内存。该内存地址为指令方括号中的十六进制数，即00417140h。

DM1-1

```
1  gi = 12;
2  0041138E     mov   dword ptr [gi (417140h)],0ch
3  }
```

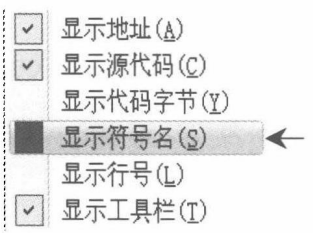


图 1.5

从这里开始，我们就开始探索式学习了。“问题”是探索式学习的关键。我们必须学会存疑发问，然后寻找解决方法求证。让我们来发问吧：

DM1-2

```
  gi = 12;
  0041138E     mov   dword ptr ds:[00417140h],0ch
}
```

计算机的信息存储在内存中，mov 指令将数据存放在指定地址的内存中。C 语言的全局变量本质上是一块内存，所以对它的存取也是通过地址进行的。

“mov [地址值], 存储值”指令将存储值存入括号中的地址指向的内存中。DS 是数据段寄存器，在 x86 寻址中是段寄存器和段内偏移合在一起定位的（这有点像用街道名和门牌号一起定位的方式）。在 Windows 和 Linux 系统中，所有段寄存器都指向一个位置，所以相当于只有一个段，段寄存器可以不用考虑。

① 内存 00417140h 真的是 gi 的地址吗？

② mov 指令真的放在内存 0041138eh 地址中吗？

先来解决第一个问题。打印是我们最熟悉的基本方法。修改源代码如下：

```
gi=12;
printf("gi address=%x\n", &gi);
```

这样太麻烦了，如果每次想看不同值，岂非要不断修改程序？这是不能容忍的！因此我们利用调试环境的一个能力，用监视窗口来观察程序状态，如变量和表达式。选择如图 1.6 所示的菜单命令，激活一个监视（watch）窗口。

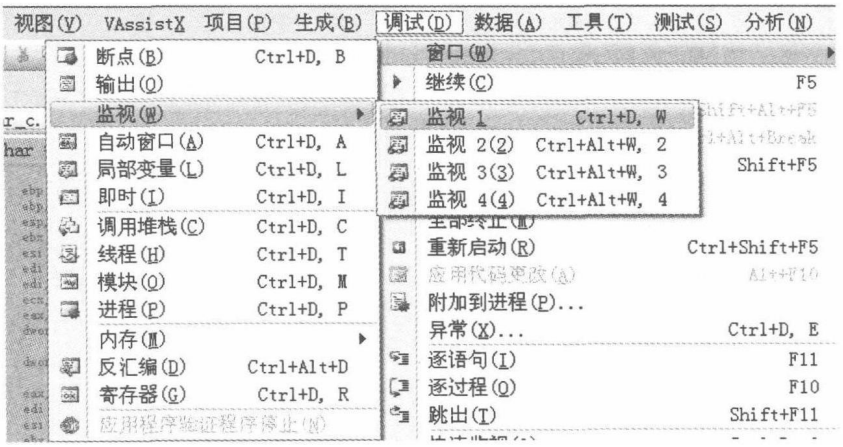


图 1.6

选择一个空行，双击“名称”列，输入“&gi”，回车，则在“值”列中显示其值 0x00417140（**与尾部加 h 一样，头部加 0x 也是十六进制数的一种表示方法**），见图 1.7。证明前面 mov 指令中的地址确实是 gi 的地址。

我们暂时将第二个问题放一放，进一步探索 mov 指令，来看它的机器码。在代码窗体中单击右键，在弹出的快捷菜单中选择“显示代码字节”（见图 1.8），则反汇编的结果如下：

```
gi = 12;
0041138E   c7 05 40 71 41 00 0c 00 00 00      mov   dword ptr ds:[00417140h], 0ch
```

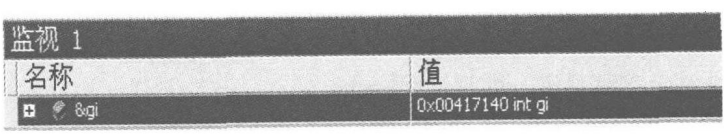


图 1.7

图 1.8