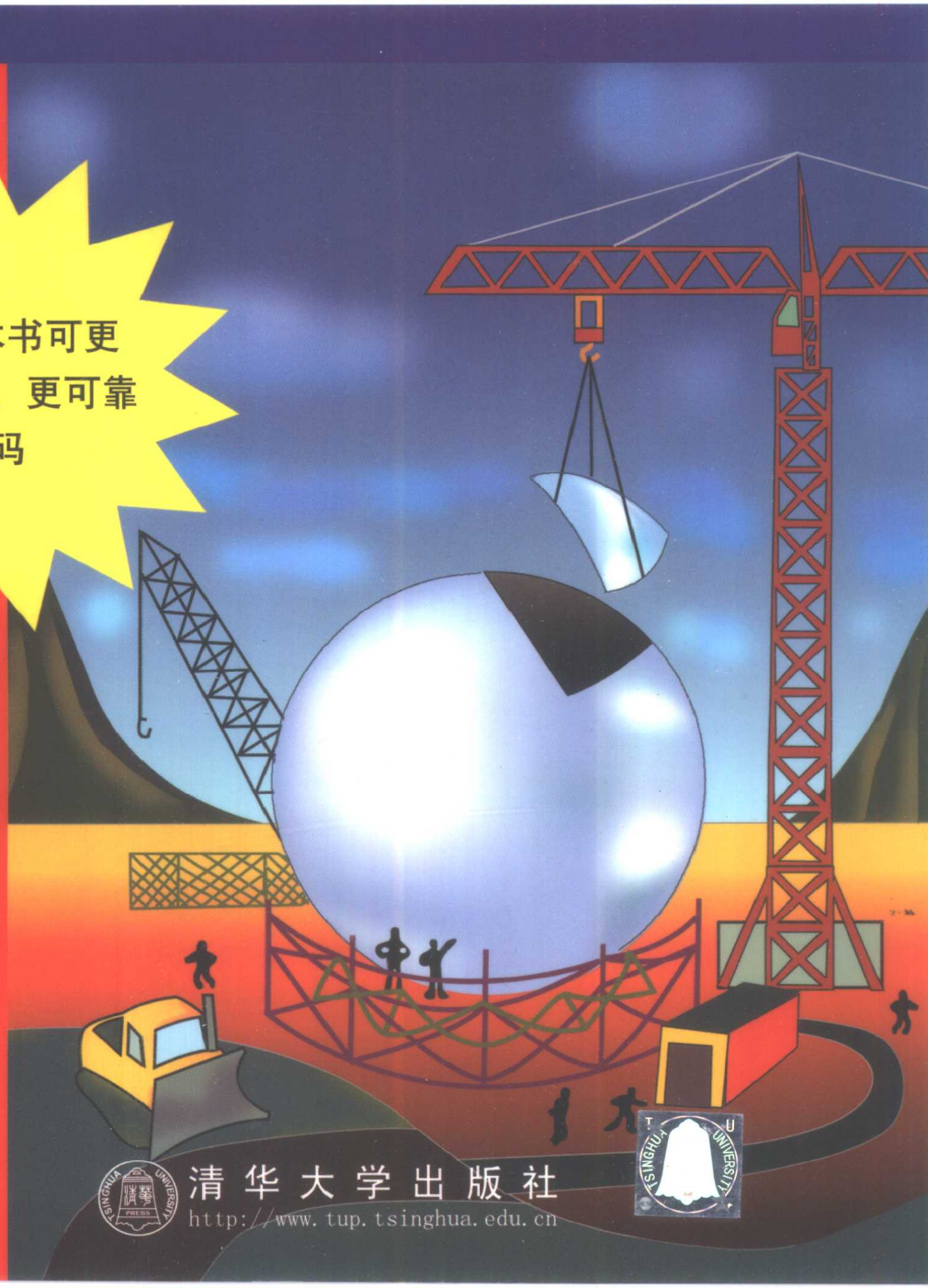
Perl调试技术

阅读本书可更
快、更好、更可靠
地编写代码



(美)Peter Scott
Ed Wright 著
曾奇 译

清华大学出版社
http://www.tup.tsinghua.edu.cn

Perl 调试技术

(美) Peter Scott Ed Wright 著
曾奇 译

清华大学出版社

(京)新登字 158 号
北京市版权局著作合同登记号:01-2001-3174

内 容 简 介

本书深入浅出地介绍了 Perl 程序的调试、测试、错误处理与代码优化等高级编程议题。全书共分 14 章,通过大量常见的错误代码实例分析了其出错原因和调试方法。

本书适合于中高级 Perl 程序员阅读,也可供相关的开发小组人员参考。

Peter Scott Ed Wright: Perl Debugged
EISBN:0-201-70054-9

书　　名:Perl 调试技术
作　　者:(美) Peter Scott Ed Wright 著 曾奇 译
出 版 者:清华大学出版社(北京清华大学学研大厦,邮编 100084)
http://www.tup.tsinghua.edu.cn
责任编辑:陈萍
印 刷 者:北京市清华园胶印厂
发 行 者:新华书店总店北京发行所
开　　本:787×1092 1/16 **印张**:10.25 **字数**:243 千字
版　　次:2001 年 10 月第 1 版 2001 年 10 月第 1 次印刷
书　　号:ISBN 7-302-04875-4/TP·2894
印　　数:0001～5000
定　　价:16.00 元

前　言

在所有编程语言中，Perl 就象一个巨大的熔炉，融入了所有文化、宗教和信仰。缺乏支持的程序员们，Perl 可以把你们从疲惫中解放出来，你们的杂乱的程序也渴望从人为约束中解放出来。那些使用 Perl 的人会发现，他们不会再受到那些编程语言设计者为了完美的语义和纯正的语法而使用的古怪概念的困扰。

Perl 的通用性和易用性使它成为了最普及的编程语言。与其他的编程语言不同，Perl 的初学者可以用 Perl 编出实用的程序，而使用其他编程语言的程序员通常需要花费大量的时间学习语法、操作和函数。Perl 程序员既可能是专家级，又可以通过修改一些样例脚本来执行新功能的新手。

但是新手还有另外一个问题：缺乏调试技能。多年积累下来的经验教训促使那些聪明人去开发一种用于调试的技巧。由于深受其害，人们希望这种惨痛教训减到最小。Perl 的易用性使程序员无需了解太多知识就可以开发出可用的（也是脆弱的）程序。然而，不同的人调试 Perl 程序所花费的时间却各不相同。我们的目标就是帮助用户将开发、调试和维护所花费的时间降至最低。

不要一看到本书的标题就认为本书讲述的是调试本书中的 Perl 程序。书中的 Perl 程序几乎没有错误。Perl 解释程序中有几个错误也没有奇怪的，因为它们是由几位优秀的支持 Perl 的志愿工作人员在短时间内写出的。本书的更为贴切的标题应该是“调试 Perl 程序（Debugging Your Perl Program）”，但是那样显得太呆板，落入俗套，并且丢失了 unplugged（丢掉障碍）的双关语意思。

编写本书的目的是让您了解程序的开发过程。大多数有关编程的书都包涵了作者精心制作的例子，这些例子运行良好，无意中显示了作者优美的编程风格。这些作者没有向读者展示把例子变得象模象样的那个令人恼火的艰难过程。事实上，这些例子与创建之初已经完全不一样了。因为您开发程序时会经历同样的过程，所以本书将引导读者经历这种过程，并提供各种方式来避免编写出优秀程序可能会遇到的麻烦、困苦和令人惊讶的缺陷。

本书描述了新 Perl 程序员可能会犯的最普遍、最恼人的错误，以及辨别和修正这些错误的详细过程。读者应该了解一些有关 Perl 的知识。因为有几个有关 Perl 的很好的指南，在本书中就不对标量、数组、散列这类概念作出解释。

本书不会试图去定义和描述一种确定的编程“风格”。风格对于个体来说是唯一的——但是许多一般规则产生了一个共同的基准，因此我们能够轻松地阅读别人的程序。

本书也不是介绍“如何编程”的书。虽然书中时常会介绍一些深入到编程一般原理的结构和基础中去的知识，但是，我们的目的并不是引导初学者把精力放在编程的原理上。

本书的读者对象

本书适用于那些运用 Perl 编程(时间在 1 周到 1 年)并希望加快开发进度的程序员。我们还解决了团队开发中编程人员面临一些问题。对于那些刚开始学习 Perl 的读者,本书通过对开发实践提供实用的建议提供帮助。

本书所覆盖的内容

本书分为如下章节:

第 1 章:Perl 文档的介绍与浏览

第 2 章:培养正确的编程思想和开发思想

第 3 章:Perl 中的"猎物":兢兢业业的利用一些窍门,正确理解 Perl 编程中的概念

第 4 章:逆向调试:如何防御性地编码

第 5 章:如何组织代码

第 6 章:如何测试 Perl 程序

第 7 章:了解 Perl 的调试器:使用这种内置工具的指南

第 8 章:了解语法错误,寻求错误原因

第 9 章:运行时错误

第 10 章:语义错误:当程序可以运行,但所执行的任务出错时产生

第 11 章:如何提高资源缺乏(内存、CPU 等)程序的性能

第 12 章:针对从其他语言转到 Perl 的程序员的提示

第 13 章:CGI 编程:调试此类 Perl 程序的特殊提示

第 14 章:结束语

附录 A:Perl 调试器命令参考

附录 B:Perl 格言

本书将花费大量时间介绍出错的例子以及调试它们的方法。

学习 Perl

尽管本书并不致力于介绍如何安装和创建 Perl,[1] 但仍然会为读者提供一些获得 Perl 开发环境的指导。

• 对于 Windows 系统，可获得免费的 ActivePerl。

http://www.activeState.com/ActivePerl/download.htm

• 对于 Macintosh 系统：

http://www.cpan.org/ports/index.html # mac

• 对于其他系统的二进制分布：

http://www.cpan.org/ports/

• Perl 的源程序：

http://www.cpan.org/src/

一旦下载了 Perl 源程序并正确解压缩后，在所支持的 Unix 系统下创建 Perl 仅需要如下命令：

```
./Configure
make
make test
make install  #  if the make test succeeds
```

在 Configure 过程中，会向用户提出许多问题，大多数用户不明白这些问题是什么意思，这时通常选择缺省设置。

为了学习的目的，读者可能希望创建一个具有调试功能的 perl 环境(这里建议采用特殊的 D 标志，这样 Perl 的输出信息被激活，并告诉读者 Perl 环境是如何操作程序的。这种情况与 Perl 内置的交互式调试器无关——将在第 7 章介绍——所有 Perl 环境均如此)。如果读者确实希望这样做，先从源程序创建 perl，当 Configure 显示如下问题时："Any additional cc flags?"将它已经显示的内容粘贴至括号内作为缺省显示的信息，并加上" - DDEBUGGING"。详情请参见 perlrun POD(本书后面将会解释)。

这里偶尔会提及一些本不是核心 Perl 分布模块，它们可以在 Comprehensive Perl Archive Network (CPAN)中找到，有关如何从 CPAN 中查找、下载以及安装模块的信息，参见 http://www.cpan.org/misc/cpan - faq.html。

印刷约定

有时，在一行写不下的代码中会有源程序中所没有的短线，当这些短线会带来问题或不明显时，在代码行的末尾会加上一个反斜线(\ )，用于表示该行还没有结束，与下一行相连。

有时，我们希望读者了解是哪一位作者在与你们对话，因此，这里将采用不同的人像标志来告诉你们。(顺便提一下，我是 Ed。)

这是 Peter。

其 他 参 考

本书的 Web 站点：http://www.perldebugged.com。

参考书目如下(大体按照有用性来排序)：

- Learning Perl, 2nd ed., by Randal Schwartz and Tom Christiansen(O'Reilly & Associates, 1997)
- Programming Perl, 3rd ed., by Larry Wall, Tom Christiansen, and Jon Orwant(O'Reilly & Associates, 2000)
- Perl, the Programmer's Companion, by Nigel Chapman(John Wily & Sons, 1998)
- Elements of Programming with Perl, by Andrew Johnson(Manning Publications, 1999)
- Effective Perl Programming, by Joseph Hall with Randal Schwartz(Addison - Wesley, 1998)

Perl 的版本

本书针对写作时最新的 Perl 版本编写，版本号为 5.6.0。对于 Perl 5.0 来说，Perl 的大多数功能是没有变化的，但是对于 Perl 4.0 就不同了。如果读者用的版本比 5.004-04 还要老，就必须升级了。版本 5.003 存在着安全性和内存泄漏的问题。读者可以用 - v 标志获得 Perl 的版本号。

```
% perl - v
This is perl, v5.6.0 built for i586 - linuxCopyright 1987 - 200, Larry Wall
[…]
```

如果 - v 标志存在，Perl 不会执行命令行中命名的脚本。有关 Perl 配置的更详细说明可以用 - v 来得到。如果读者发布了一个 bug 报告，完成这项工作的方式会自动将这些信息包含到报告中。[2]

Perl 有一些中间版本，这些中间版本的发布号包含一个后面接数字 50 或更大的数的下划线，或包含一个存在于两点之间的奇数，所以可以通过版本号来确定你的 Perl 是否是中间版本。这些中间版本并不能保证正确，一般是用于测试。如果读者的 Perl 恰好是中间版本，而你本不想要它，那么你很可能是通过错误的 FTP 链接下载的。

第四届 Perl 年会(Monterey, California, July 2000)宣布，Perl 6 正在认真的开发中，并且它注重性能甚至于向后兼容性。本书付印时，有关新的语言功能的讨论仍然没有结束。

致　谢

感谢 David Noble 编写的最初 proxylog 工具版本。感谢 llya Zakharevich 给我讲授的 Perl 调试器。尤其感谢我的妻子 Grace，在编书过程中，她给予了我极大的支持。

与以往一样，感谢我的祖母，Charles D. Wright Sr，她让我使用计算机和阅读 Doc Savage 的书。感谢 Mary O'Brien，他让我接下了编著这本书的工作。

此外，感谢我们的编辑 Mike Hendrickson，以及他的同事 Julie Debaggis 和 Marcy Barnes，感谢他们不知疲倦的支持、耐心和理解，他们冒着风险将两位新人引入这一领域，并且逐渐地将我们引入作者群中。在 Addison - Wesley 出版社的众多普通人员和其他不知疲倦的帮助者中，同时还感谢 Cathy Comer、Mary Cotillo、Jacquelyn Doucette、John Fuller、Karin Hansen、Katie Noyes、Heather Olszyk 和 Heather Peterson。

感谢 Elaine Ashton 所作的评论，感谢 Brad Appleton、Sean M. Burke、Joseph Hall、Jarkko Hietaniemi、Michael G. Schwern 和其他评论家的深刻的、有益的评注，这些评注使本书得到改进。

注释：

1. 这并不是打印错误。根据惯例，大写的 P 指抽象的 Perl 语言，而小写的 p 指运行 Perl 程序的 Perl 环境。

2. 使用 perlbug 程序包含 Perl 分布。

目　录

第1章 概 述

1.1 软件开发的现状

编程业(以及程序员)是不幸的,项目宣告完成以后(一个越来越任意的、常常是不准确的标志),软件产品的开发还没完没了。产品的维护和调试耗费的时间可能比编写代码还要多,特别是当负责程序维护的人员可能不是原始的开发人员时,这表现得更为明显。这种情况意味着我们要花费大量时间去推导代码中的运算法则的细节:它是如何完成的?为什么这么做?为什么它不再工作?[1]

创建一个没有 bug 的程序是相当漫长的。任何一个比常见的 Hello World 程序大的程序都可能存在 bug。没有任何方法能够保证完全消除 bug,但已形成了一些实用的法则:

1. 减少可能出现 bug 的地方,即增加标准设计、封装和进行单位测试。

2. 鉴别 bug 产生的原因。运用监测功能:什么不应该发生而发生了;什么应该发生而没有发生。

3. 将 bug 的位置分离开来——在哪里出现的 bug?(这就是创建调试器的原因。)

4. 确定生成 bug 的条件。是什么激发的 bug?为什么以前没有发现它?

5. 纠正 bug——重新测试修改过的代码。

6. 第一次 bug 是如何产生的?如果您输入错误,5 分钟后在单元测试时发现了该 bug,那是一种情况。如果由于逻辑错误造成的 bug,就是另一种问题。寻找改善开发过程的方法,类似的 bug 就可以很快地发现了。

1.2 为什么选择 Perl

如果您使用 Perl,就会知道为什么选择它,并且不会改变选择。但是附加的功能不会对编程不利,如果您不得不向管理层说明为什么选择 Perl,可以这么说:Perl 不像大的跨国集团一样在市场竞争中投入大量资金,[2] 因此用户会像我们一样赞扬 Perl 的实用性。

只要遇见需要用编程解决的问题,我们就会先在脑海里以图形和字词构思出答案,这些答案形成解决问题的众多开发方法,在将这些方法转换成代码之前,必须用英文(或其他母语)将它们描述成一定的算法。如果算法的描述已经完成并且非常明确,那么若计算机能够明白我们的描述,[3] 就可以在计算机上运行该算法。但是计算机中还没有用英文算法表示的分析器,所以我们必须将它们翻译成计算机能够理解的形式。

这样就存在着困难:在执行语句的过程中,理想的计算机语言在编程者实现算法时不会增添额外的开销。但是这样的语言并不存在,所以我们希望有一种语言能将这种额外的开

销降至最低。语言的选择随任务种类的不同有很大变化——如果是编写设备驱动程序,最好的选择很可能是汇编器;如果是编写图形用户界面,至少有部分问题最好通过所见即所得编辑器来解决,因为它创建了一个布局定义。

编写程序时,有些代码常常是为了满足语言的需要,而不是问题本身的需要,这样就增加了额外的工作量。例如,在编写 C 和 C++ 程序时,经常需要考虑为动态数据结构分配内存,以及何时、何地释放内存的问题。

Perl 为用户提供了一种比任何其他编程语言更为轻松的编程环境,在这种环境下,用户可以以最小的开销解决绝大多数问题。用户只需少量的学习,就可以在极短的时间内创建出短小精悍的用于解决复杂问题的程序。因为 Perl 语言本身所要求的开销很少,更因为把英语描述更加自然地映象到 Perl 是 Perl 语言的一个特征,所以需求的改变不会引起大段无关代码的改变。同时,由于 Perl 的表达清晰、简练,所以阅读程序的人员一眼就能够理解它的意思。

事实上,Perl 在解决产生歧义的能力上,在它的直观和直角的界面上,甚至大声阅读时听起来的效果上都和英语类似。[4]Perl 的创造者 Larry Wall 常说,他在语言学方面的背景知识使得他在创造 Perl 的时候下意识地实现了这种一致性。[5]Perl 把这种集合现象称为 Perl 的 DWIM(Do What I Mean,按我的意思做)特征。

若您要了解有关说服管理者支持 Perl 的评述,参见 http://perl.oreilly.com/news/success-stories.html 上的 Perl Success Stories 页。若读者要了解有关科学界人士的评述,参见 http://wwwipd.ira.uka.de/~prechelt/Biblio/jccpprtTR.pdf 上的 Lutz Prechelt 撰写的有关语言的比较一文。

1.3 Perl 的环境

要了解一门语言,必须了解不同环境下出错信息的实际含义。如果一段 Unix 程序返回出错信息"Segmentation fault"(在其他的操作系统下可能会是"ABEND"、"% SYSTEM-F-ACCVIO",或计算机被冻结),有经验的程序员首先就会怀疑是内存问题,这种情况常常是由于读、写数组超出数组边界引起的。另一个 Unix 出错信息"bus error"常常是在子程序或函数调用了错误参数表时产生的。快速、准确地识别出错原因,需要丰富的编程经验。本书尽可能地列出了 Perl 编程时可能遇到的出错信息及解决方法。

1.4 Perl 的语言

正如每个国家都有自己的风俗一样,不同的编程语言也有其特殊的习惯用语。当程序员能够正确运用这些常用语,而不是试图使新语言适应旧的编程方式时,就能够自如地运用新语言了。

几年以前，当我在我的第一台大型机(运行 George III 的一台 ICL 1903T)上工作了一段时间后，我引进了第二台机器(从技术上来说是一台小型机：一台运行 VAX/VMS 的 VAX11/780)。为了能够运用与旧计算机相同的命令执行诸如列出文件等操作，我花费了一些时间在新的操作系统中定义符号。我附近的一位聪明的程序员问我，“为什么您要让 VMS 看起来像 George III?”任何一个了解它们各自命令行界面具有不同输入方式的人，都会欣赏这句具有讽刺意义的问话。

您应该找到 Perl 通用的惯用语并练习它们的使用方法。有很多原因可以解释为什么需要具有通用的惯用语。因为采用通用的语言编程，读懂程序的人就越多，程序代码就越好维护。

这些惯用语可以从简单的两个单词“open 或 die”：

```
open FH, $ file or die "Cann't open $ file: $ ! \ n";
```

到复杂的(第一眼看上去)Schwartzian Transform：

```
@ sorted = map $ _- > [0]
        sort{ $ a- > [1] <=> $ b = - > [1]}
        map[ $ _, sortable _ func( $ _)]
        @ unsorted
```

第一个惯用语的表达方式很快就成为了一种习惯的表达方式(现在我们可以随意地写出它们)，尽管在第一次学习这个语法的时候通常都表示为如下的形式：

```
if(open FH, $ file)
{
...
}
else
{
die "Unable to open $ file: $ ! \ n"
}
```

上述惯用语流行并不难理解。简而言之，它的工作方式是这样的：如果失败，open 返回假，逻辑运算符“or”短路，因为一旦知道 A 为真，那么“A or B”的计算结果就是固定的，所以当第一个值为真，就不用再检测第二个参数了。

第二个惯用语提供了一个高效率的排序方式，这个排序方式需要在项目能够以“ASCII betical”和数值顺序排序之前对它们进行操作。当您了解了这一点，并意识到只有两个可以变化的记号，[6] 只要可以，您就会自然地认出 map-sort-map 模式，并将它翻译成 Schwartzian Transform。

1.5 联机文档

幸运的是，perl 环境所提供的文档几乎覆盖了您想知道的关于 Perl 的任何信息。但是，

用户想要了解的有关 Perl 的资料可能多得看不过来。尽管如此,在 Perl 环境中,只需要很少的引导,用户就可以非常迅速地知道如何在在线文档中进行搜索查询,与其他语言相比,为用户节约了大量的时间。Perl 的在线文档采用的是 POD(Plain Old Documentation)格式,这是 Perl 的一种十分简单的标记语言。

了解该文档的方法就是一种主要的调试工具。由于这是一个如此要点,所以我们在整本书中都用突出显示引起读者注意。这些提示我们称之为"Perl 格言"。

(1) 熟悉文档,从中能够发现任何事物。

(为了快速参考的目的,本书将所有"Perl 格言"提示都列入了附录 B)。最近几年,由于忽视上述建议而造成不良结果的一个最富有代表性的例子,是 1999 年以后"localtime"函数的行为所引起的关注。许多编程者仅凭看到的 localtime 函数返回的年代值,就错误地认为这个值是由实际年份的最后两位数得来的,因此,他们编写窗口操作代码时只把"19"加在该函数的前面,用来得到实际的年份。但是,如下的 Perl 文档说明了这种看法是错误的,

> ... $ year 指从 1900 年开始的年数,例如 $ year 等于 133,说明该年是 2023 年,而不是简单地指年的最后两位数。如果您这样错误地认为,就会创建非 Y2K 兼容的程序——当然您并不愿意这样,不是吗?

您可以区分出,Perl 文档中的一些文本是由那些曾经由于没有注意语言的文档说明而受挫的文档编写者添加上去的。

1998 年,我被要求修改 10 年前在 VMS 计算机上编写的一段程序,该程序在 2000 年问题测试时出现错误。我非常惊讶地发现,10 年前我编写的这段窗口代码认为,C 函数的时间结构函数所返回的年份只包含两位数,因此,程序打印出的年份是"19100"。程序修改好后,我找到我编写该程序时所用的原始的 VMS C 手册,并在手册中查到了 localtime 函数。令人惊讶的是,手册中说明年份包含在该函数的最后两位数中!(这样的错误在 Perl 的文档中是不会存在如此长的时间的,而这必须感谢开放源代码。)

1.5.1 Windows 中的 Perl 文档

在 Microsoft Windows 下,如果 Perl 的端口是激活的,perldoc 页就可以转换为超链接的 HTML。[7] 用户就可以通过"开始"菜单来访问:

```
programs -> ActivePerl -> Online Documentation
```

或者到安装的 ActivePerl 的根目录下,打开 index.htm 文件。在任何其他平台上,所有在 Core Perl FAQ、Core Perl Docs 以及在 Module Docs 下列出的大多数文档,都包含在标准 perl 发布版中。如果您单击主框架中的 Perl Documentation,将会引导您进入 Perl 本身的核心文档根部,而不是 Active State 所创建的页面。这是浏览文档的好方法,我们相当喜欢这种界面。但是,光靠浏览是不够的。经常需要您在长篇文档中精确定位到某一段特殊部分。例如要查找使用 split 的方法,您首先必须进入庞大的 perlfunc 页,然后滚动到正确的位置。要解决这个问题,就可能采用命令行版本,这样我们就需要了解一下 perldoc。