2nd Edition

高级 Perl 编程（影印版）

Advanced Perl Programming

O'REILLY®

東南大學出版社

Simon Cozens 著

第二版

高级 Perl 编程（影印版）
Advanced Perl Programming

Simon Cozens

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc. 授权东南大学出版社出版

东南大学出版社

图书在版编目（CIP）数据

高级 Perl 编程：第 2 版 /（美）科曾斯（Cozens, S.）著
—影印本 .— 南京：东南大学出版社，2006.4
书名原文：Advanced Perl Programming, Second Edition
ISBN 7-5641-0315-9

Ⅰ. 高… Ⅱ. 科… Ⅲ. PERL 语言－程序设计－英文
Ⅳ. 312

中国版本图书馆 CIP 数据核字（2006）第 021828 号

江苏省版权局著作权合同登记
图字：10-2006-39 号

书　　名 /	高级 Perl 编程　第二版（影印版）
书　　号 /	ISBN 7-5641-0315-9
责任编辑 /	张烨
封面设计 /	Edie Freedman，马冬燕
出版发行 /	东南大学出版社（press.seu.edu.cn）
地　　址 /	南京四牌楼 2 号（邮政编码 210096）
印　　刷 /	扬中市印刷有限公司
开　　本 /	787 毫米 × 980 毫米　16 开本　18.75 印张
版　　次 /	2006 年 4 月第 1 版　2006 年 4 月第 1 次印刷
印　　数 /	0001-2000 册
定　　价 /	40.00 元（册）

出版说明

随着计算机技术的成熟和广泛应用，人类正在步入一个技术迅猛发展的新时期。计算机技术的发展给人们的工业生产、商业活动和日常生活都带来了巨大的影响。然而，计算机领域的技术更新速度之快也是众所周知的，为了帮助国内技术人员在第一时间了解国外最新的技术，东南大学出版社和美国 O'Reilly Meida, Inc.达成协议，将陆续引进该公司的代表前沿技术或者在某专项领域享有盛名的著作，以影印版或者简体中文版的形式呈献给读者。其中，影印版书籍力求与国外图书“同步”出版，并且“原汁原味”展现给读者。

我们真诚地希望，所引进的书籍能对国内相关行业的技术人员、科研机构的研究人员和高校师生的学习和工作有所帮助，对国内计算机技术的发展有所促进。也衷心期望读者提出宝贵的意见和建议。

最新出版的一批影印版图书，包括：

- 《深入理解 Linux 内核　第三版》（影印版）
- 《Perl 最佳实践》（影印版）
- 《高级 Perl 编程　第二版》（影印版）
- 《Perl 语言入门　第四版》（影印版）
- 《深入浅出 HTML 与 CSS、XHTML》（影印版）
- 《UML 2.0 技术手册》（影印版）
- 《802.11 无线网络权威指南　第二版》（影印版）
- 《项目管理艺术》（影印版）
- 《.NET 组件开发　第二版》（影印版）
- 《ASP.NET 编程　第三版》（影印版）

Preface

It was all Nathan Torkington's fault. Our Antipodean programmer, editor, and O'Reilly conference supremo friend asked me to update the original *Advanced Perl Programming* way back in 2002.

The Perl world had changed drastically in the five years since the publication of the first edition, and it continues to change. Particularly, we've seen a shift away from techniques and toward resources—from doing things yourself with Perl to using what other people have done with Perl. In essence, advanced Perl programming has become more a matter of knowing where to find what you need on the CPAN,* rather than a matter of knowing what to do.

Perl changed in other ways, too: the announcement of Perl 6 in 2000 ironically caused a renewed interest in Perl 5, with people stretching Perl in new and interesting directions to implement some of the ideas and blue-skies thinking about Perl 6. Contrary to what we all thought back then, far from killing off Perl 5, Perl 6's development has made it stronger and ensured it will be around longer.

So it was in this context that it made sense to update *Advanced Perl Programming* to reflect the changes in Perl and in the CPAN. We also wanted the new edition to be more in the spirit of Perl—to focus on how to achieve practical tasks with a minimum of fuss. This is why we put together chapters on parsing techniques, on dealing with natural language documents, on testing your code, and so on.

But this book is just a beginning; however tempting it was to try to get down everything I ever wanted to say about Perl, it just wasn't possible. First, because Perl usage covers such a wide spread—on the CPAN, there are ready-made modules for folding DNA sequences, paying bills online, checking the weather, and playing poker. And more are being added every day, faster than any author can keep up. Second, as we've mentioned, because Perl is changing. I don't know what the next big advance

* The *Comprehensive Perl Archive Network* (*http://www.cpan.org*) is the primary resource for user-contributed Perl code.

in Perl will be; I can only take you through some of the more important techniques and resources available at the moment.

Hopefully, though, at the end of this book you'll have a good idea of how to use what's available, how you can save yourself time and effort by using Perl and the Perl resources available to get your job done, and how you can be ready to use and integrate whatever developments come down the line.

In the words of Larry Wall, may you do good magic with Perl!

Audience

If you've read *Learning Perl* and *Programming Perl* and wonder where to go from there, this book is for you. It'll help you climb to the next level of Perl wisdom. If you've been programming in Perl for years, you'll still find numerous practical tools and techniques to help you solve your everyday problems.

Contents

Chapter 1, *Advanced Techniques*, introduces a few common tricks advanced Perl programmers use with examples from popular Perl modules.

Chapter 2, *Parsing Techniques*, covers parsing irregular or unstructured data with `Parse::RecDescent` and `Parse::Yapp`, plus parsing HTML and XML.

Chapter 3, *Templating Tools*, details some of the most common tools for templating and when to use them, including formats, `Text::Template`, `HTML::Template`, `HTML::Mason`, and the Template Toolkit.

Chapter 4, *Objects, Databases, and Applications*, explains various ways to efficiently store and retrieve complex data using objects—a concept commonly called object-relational mapping.

Chapter 5, *Natural Language Tools*, shows some of the ways Perl can manipulate natural language data: inflections, conversions, parsing, extraction, and Bayesian analysis.

Chapter 6, *Perl and Unicode*, reviews some of the problems and solutions to make the most of Perl's Unicode support.

Chap ter 7, *POE*, looks at the popular Perl event-based environment for task scheduling, multitasking, and non-blocking I/O code.

Chapter 8, *Testing*, covers the essentials of testing your code.

Chapter 9, *Inline Extensions*, talks about how to extend Perl by writing code in other languages, using the `Inline::*` modules.

Chapter 10, *Fun with Perl*, closes on a lighter note with a few recreational (and educational) uses of Perl.

Conventions Used in This Book

The following typographical conventions are used in this book:

Plain text
: Indicates menu titles, menu options, menu buttons, and keyboard accelerators (such as Alt and Ctrl).

Italic
: Indicates new terms, URLs, email addresses, filenames, file extensions, pathnames, directories, and Unix utilities.

`Constant width`
: Indicates commands, options, switches, variables, attributes, keys, functions, classes, namespaces, methods, modules, parameters, values, XML tags, HTML tags, the contents of files, or the output from commands.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values.

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Advanced Perl Programming*, Second Edition by Simon Cozens. Copyright 2005 O'Reilly Media, Inc. 0-596-00456-7."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

We'd Like to Hear from You

Please address comments and questions concerning this book to the publisher:

O'Reilly Media
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/advperl2/

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com

Safari® Enabled

When you see a Safari Enabled icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://safari.oreilly.com*.

Acknowledgments

I've already blamed Nat Torkington for commissioning this book; I should thank him as well. As much as writing a book can be fun, this one has been. It has certainly been helped by my editors, beginning with Nat and Tatiana Apandi, and ending with the hugely talented Allison Randal, who has almost single-handedly corrected code, collated comments, and converted my rambling thoughts into something publishable. The production team at O'Reilly deserves a special mention, if only because of the torture I put them through in having a chapter on Unicode.

Allison also rounded up a great crew of highly knowledgeable reviewers: my thanks to Tony Bowden, Philippe Bruhat, Sean Burke, Piers Cawley, Nicholas Clark, James Duncan, Rafael Garcia-Suarez, Thomas Klausner, Tom McTighe, Curtis Poe, chromatic, and Andy Wardley.

And finally, there are a few people I'd like to thank personally: thanks to Heather Lang, Graeme Everist, and Juliet Humphrey for putting up with me last year, and to Jill Ford and the rest of her group at All Nations Christian College who have to put up with me now. Tony Bowden taught me more about good Perl programming than either of us would probably admit, and Simon Ponsonby taught me more about everything else than he realises. Thanks to Al and Jamie for being there, and to Malcolm and Caroline Macdonald and Noriko and Akio Kawamura for launching me on the current exciting stage of my life.

Table of Contents

CHAPTER 1

Advanced Techniques

Once you have read the Camel Book (*Programming Perl*), or any other good Perl tutorial, you know almost all of the language. There are no secret keywords, no other magic sigils that turn on Perl's advanced mode and reveal hidden features. In one sense, this book is not going to tell you anything new about the Perl language.

What can I tell you, then? I used to be a student of music. Music is very simple. There are 12 possible notes in the scale of Western music, although some of the most wonderful melodies in the world only use, at most, eight of them. There are around four different durations of a note used in common melodies. There isn't a massive musical vocabulary to choose from. And music has been around a good deal longer than Perl. I used to wonder whether or not all the possible decent melodies would soon be figured out. Sometimes I listen to the Top 10 and think I was probably right back then.

But of course it's a bit more complicated than that. New music is still being produced. Knowing all the notes does not tell you the best way to put them together. I've said that there are no secret switches to turn on advanced features in Perl, and this means that everyone starts on a level playing field, in just the same way that Johann Sebastian Bach and a little kid playing with a xylophone have precisely the same raw materials to work with. The key to producing advanced Perl—or advanced music—depends on two things: knowledge of techniques and experience of what works and what doesn't.

The aim of this book is to give you some of each of these things. Of course, no book can impart experience. Experience is something that must be, well, *experienced*. However, a book like this can show you some existing solutions from experienced Perl programmers and how to use them to solve the problems you may be facing.

On the other hand, a book can certainly teach techniques, and in this chapter we're going to look at the three major classes of advanced programming techniques in Perl. First, we'll look at introspection: programs looking at programs, figuring out how they work, and changing them. For Perl this involves manipulating the symbol table—especially at runtime, playing with the behavior of built-in functions and using `AUTOLOAD` to introduce new subroutines and control behavior of subroutine dispatch dynamically. We'll also briefly look at bytecode introspection, which is the ability to inspect some of the properties of the Perl bytecode tree to determine properties of the program.

The second idea we'll look at is the class model. Writing object-oriented programs and modules is sometimes regarded as advanced Perl, but I would categorize it as intermediate. As this is an advanced book, we're going to learn how to subvert Perl's object-oriented model to suit our goals.

Finally, there's the technique of what I call *unexpected code*—code that runs in places you might not expect it to. This means running code in place of operators in the case of overloading, some advanced uses of tying, and controlling when code runs using named blocks and `eval`.

These three areas, together with the special case of Perl XS programming—which we'll look at in Chapter 9 on `Inline`—delineate the fundamental techniques from which all advanced uses of Perl are made up.

Introspection

First, though, introspection. These introspection techniques appear time and time again in advanced modules throughout the book. As such, they can be regarded as the most fundamental of the advanced techniques—everything else will build on these ideas.

Preparatory Work: Fun with Globs

Globs are one of the most misunderstood parts of the Perl language, but at the same time, one of the most fundamental. This is a shame, because a glob is a relatively simple concept.

When you access any global variable in Perl—that is, any variable that has not been declared with `my`—the *perl* interpreter looks up the variable name in the *symbol table*. For now, we'll consider the symbol table to be a mapping between a variable's name and some storage for its value, as in Figure 1-1.

Note that we say that the symbol table maps to *storage* for the value. Introductory programming texts should tell you that a variable is essentially a box in which you can get and set a value. Once we've looked up `$a`, we know where the box is, and we can get and set the values directly. In Perl terms, the symbol table maps to a reference to `$a`.

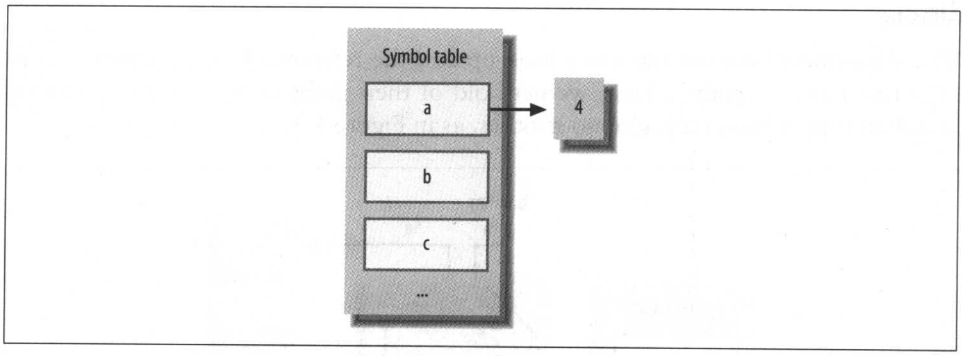

Figure 1-1. Consulting the symbol table, take 1

You may have noticed that a symbol table is something that maps names to storage, which sounds a lot like a Perl hash. In fact, you'd be ahead of the game, since the Perl symbol table is indeed implemented using an ordinary Perl hash. You may also have noticed, however, that there are several things called *a* in Perl, including $a, @a, %a, &a, the filehandle a, and the directory handle a.

This is where the glob comes in. The symbol table maps a name like a to a glob, which is a structure holding references to all the variables called a, as in Figure 1-2.

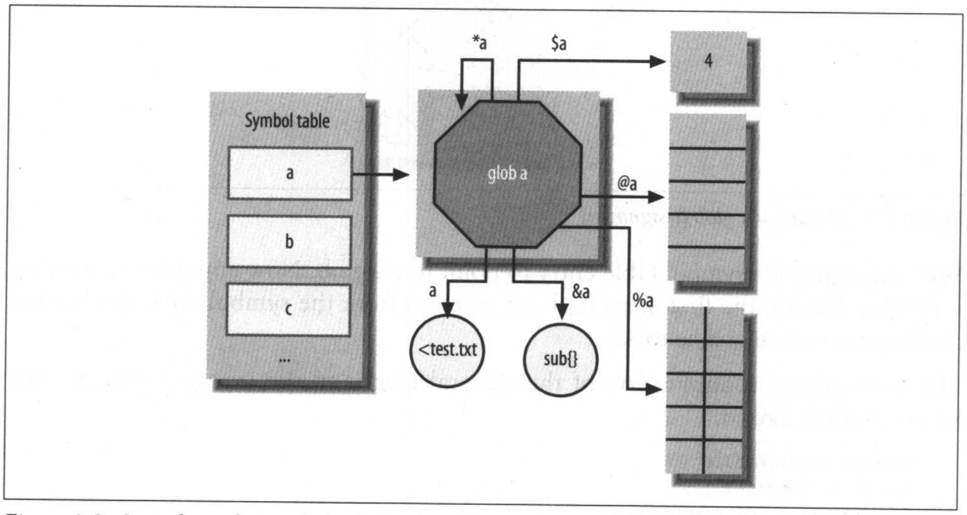

Figure 1-2. Consulting the symbol table, take 2

As you can see, variable look-up is done in two stages: first, finding the appropriate glob in the symbol table; second, finding the appropriate part of the glob. This gives us a reference, and assigning it to a variable or getting its value is done through this reference.

Aliasing

This disconnect between the name look-up and the reference look-up enables us to alias two names together. First, we get hold of their globs using the **name* syntax, and then simply assign one glob to another, as in Figure 1-3.

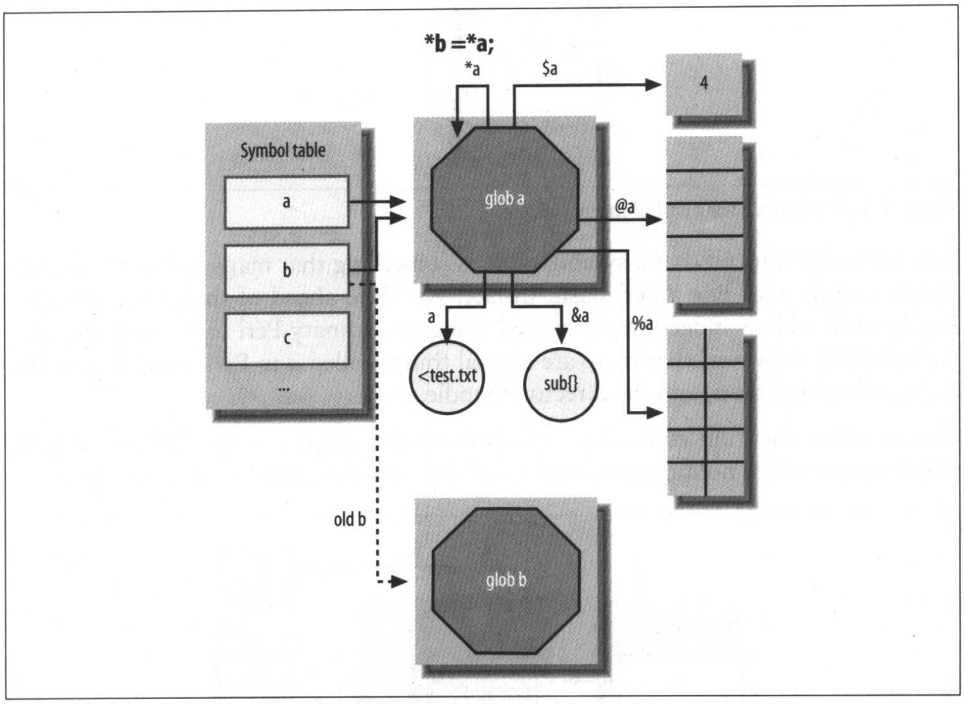

Figure 1-3. Aliasing via glob assignment

We've assigned b's symbol table entry to point to a's glob. Now any time we look up a variable like %b, the first stage look-up takes us from the symbol table to a's glob, and returns us a reference to %a.

The most common application of this general idea is in the Exporter module. If I have a module like so:

```
package Some::Module;
use base 'Exporter';
our @EXPORT = qw( useful );

sub useful { 42 }
```

then Exporter is responsible for getting the useful subroutine from the Some::Module package to the caller's package. We could mock our own exporter using glob assignments, like this:

```
package Some::Module;
sub useful { 42 }
```