Perl语言高级编程专题
Lesson 10

周晓方
courses@xfzhou.homeftp.org

《Perl调试技巧》清华大学出版社2001年

(Peter Scott, Ed Wright, "Perl Debugged")

对Perl程序员的忠告

- 区分$, @, %; 函数后的括号可以省略

- 和C语言的一些区别：
  - 类型的多态性
  - 没有malloc/free的困扰
  - $a || $b不返回布尔值，返回$a或$b之一的值，
    $a or $b，$a && $b，$a and $b等也是这样
  - ? : 可以是左值: ($flag ? $a : $b) = 6;
  - 没有struct，union，switch等
  - C的i[a]和a[i]等价，perl不是

- 逻辑或运算，||还是or

```
$x >=0 or $x = -$x;          # or的优先级比>=、=低
$x = $title || 'N/A';        # ||的优先级比=高
return $PULL or 'hi';       # return($PULL) or 'hi'!!
```

资源的优化

- 首先确保可读性和可维护性 (据Moore's Law)
- 利用任务管理器查看内存占用情况，用Benchmark，Devel::Profile，Devel::NYTProf确定运行的时间
- 及早结束循环：循环体开头合理增加next/last
- 尽量避免调用外部程序，避免`` `或system( )
- 节约内存：优先用数组，再hash，最后标量
- 空间换速度：Memoize模块，记住已有调用
- 速度换空间：大hash用tie函数关联到数据库

语义（理解上的差别）

- 语义错误：open (A, 'f') | die "Error";
- 语义错误：printf "Number of \@a = %d", @a;
- 用B::Deparse查看perl对代码的理解
  Deparse还很简陋，但遇到了奇怪现象，可以单独拿出来试一试

```
perl -MO=Deparse -e "for ($i=1;$i<10;$i++) {$j+=$i;}"
for ($i = 1; $i < 10; ++$i) {
    $j += $i;
}
-e syntax OK

perl -MO=Deparse -e "use strict my $a = 1; print $a; exit;"
print $a;
exit;
-e syntax OK  (This sample fails on perl 5.10)
```

perl的调试器

- perl -d myscripts.pl
  - s　　Step　运行一行，跟踪进入子程序
  - nNext　运行一行
  - r　　Return 运行到当前子程序结束
  - p 变量名　　查看变量
  - x 变量名　　查看变量,友好格式(hash用引用)
  - l/-/w　列出前后的代码
  - c 行号　　执行到"行号"
  - c　　执行，直到遇到断点
  - b 行号　　设置断点（b 子程序名）
  - b 行号 条件　设置条件断点
  - d 行号　　去除"行号"处的断点

– L　列出所有的断点

– t　跟踪执行

```
1    #!/usr/bin/perl -wd
2    use strict;
3    my $j;
4    foreach my $i (1..10) {
5        $j += $i;
6        print "\$i is $i, \$j is $j\n";
7    }
8    1;
```

```
C:\>bug.pl
Default die handler restored.

Loading DB routines from perl5db.pl
version 1.07
Editor support available.

Enter h or `h h' for help, or `perldoc
perldebug' for more help.

main::(C:\bug.pl:3):    my $j;
  DB<1> b 6 $i == 8
  DB<2> c
$i is 1, $j is 1
$i is 2, $j is 3
$i is 3, $j is 6
$i is 4, $j is 10
```

```
$i is 5, $j is 15
$i is 6, $j is 21
$i is 7, $j is 28
main::(C:\bug.pl:6):            print
"\$i is $i, \$j is $j\n";
  DB<2> t
Trace = on
  DB<2> c
$i is 8, $j is 36
main::(C:\bug.pl:4):    foreach my $i
(1..10) {
main::(C:\bug.pl:5):            $j += $i;
main::(C:\bug.pl:6):            print
"\$i is $i, \$j is $j\n";
$i is 9, $j is 45
main::(C:\bug.pl:4):    foreach my $i
(1..10) {
main::(C:\bug.pl:5):            $j += $i;
main::(C:\bug.pl:6):            print
"\$i is $i, \$j is $j\n";
$i is 10, $j is 55
main::(C:\bug.pl:4):    foreach my $i
(1..10) {
main::(C:\bug.pl:8):    1;
Debugged program terminated.  Use q to
quit or R to restart,
  use O inhibit_exit to avoid stopping
after program termination,
  h q, h R or h O to get additional info.
  DB<2>
```

测试，使用Test模块

```
#!/usr/bin/perl -w
use strict;
use Test;
plan tests => 10;
sub s1 {
    my $j;
    $j += $_
        foreach (1..$_[0]);
    $j;
}
sub s2 {
    $_[0] * ($_[0] + 1) / 2;
}
ok(0, s1 0);
ok(0, s2 0);
ok(1, s1 1);
ok(1, s2 1);
ok(3, s1 2);
ok(3, s2 2);
ok(55, s1 10);
ok(55, s2 10);
ok(5050, s1 100);
ok(5050, s2 100);
1;
```

```
C:\>test.pl
1..10
not ok 1
# Failed test 1 in C:\test.pl at
line 13
ok 2
ok 3
ok 4
ok 5
ok 6
ok 7
ok 8
ok 9
ok 10

C:\>
```

测试，使用Devel::Coverage

- 需要到CPAN上下载、安装
- `perl -d:Coverage coverage.pl`
- 生成同名的*coverage.pl.cvg*
- 运行`coverperl coverage.pl.cvg`显示覆盖率

```
perl -d:Coverage coverage.pl
```

代码

```
#!/usr/bin/perl -w
use strict;
sub ss {
    if ($_[0] <= 0) {
        print "Ignored\n";
        return;
    }
   print $_[0] * ($_[0] + 1) / 2;
    print "\n";
}
foreach (1..10) {
    ss($_);
}
1;
```

运行

```
1
3
6
10
15
21
28
36
45
55
```

显示覆盖率

```
C:\>coverperl coverage.pl.cvp
Total of 5 instrumentation runs.

\C:\coverage.pl
        10  main::ss
                10    line 4
                0     line 5
                0     line 6
                10    line 8
                10    line 9
        3     line 2
        11    line 11
        10    line 12
        1     line 14


C:\>
```

覆盖率，使用Devel::Cover

- https://metacpan.org/release/Devel-Cover
  - 点击<Devel::Cover::Tutorial>了解覆盖率的概念
  - 点击<Devel::Cover>了解基本用法
  - 点击<cover>如何用cover命令生成各种报告
- perl -MDevel::Cover mycode.pl args
- 运行后生成cover_db子目录
- *cover (cover.bat)*产生报告
- 浏览器打开cover_db/coverage.html

Report Date:	2021-04-19 20:48:44
Perl Version:	v5.30.0
OS:	MSWin32
Thresholds:	< 75%

file	stmt	bran	cond	sub	po
coverage.pl	80.0	50.0	n/a	100.0	n/
Total	80.0	50.0	n/a	100.0	n/

Branch Coverage

File:	coverage.pl
Coverage:	50.0%

line	%	coverage		branch
4	50	T	F	`if ($_[0] <= 0)`

了解速度瓶颈：Devel::DProf

```
#!/usr/bin/perl -w
use strict;
foreach (90000..100000) {s1($_);}
foreach (90000..100000) {s2($_);}
sub s1 {
    my $j;
    $j += $_ foreach (1..$_[0]);
}
sub s2 {
    $_[0] * ($_[0] + 1) / 2;
}
1;
```

- 运行时增加**-d:DProf**选项，会自动生成一个***tmon.out***文件
- 查看profile结果，运行**dprofpp**

tmon.out
该文件会很大

```
C:\>perl -d:DProf profile.pl

C:\>dprofpp
Total Elapsed Time = -0.04001 Seconds
  User+System Time = 296.9769 Seconds
Exclusive Times
%Time ExclSec CumulS #Calls sec/call Csec/c  Name
 99.9   296.9 296.91  10001   0.0297 0.0297  main::s1
 0.01   0.040  0.030  10001   0.0000 0.0000  main::s2
 0.00   0.010  0.010      1   0.0100 0.0100  main::BEGIN
 0.00   0.000 -0.000      1   0.0000      -  strict::import
 0.00   0.000 -0.000      1   0.0000      -  strict::bits

C:\>
```

基于行的--Devel::SmallProf

- 用法 perl –d:SmallProf 脚本，结果在smallprof.out中
- 显示每行、而不是每个子程序的执行时间
- 需要预装Time::HiRes，要有配套的C编译器
- 一些选项开关：
  - 暂时关闭和打开profile：
    - `$DB::profile = 0;`
    - 这之间的代码不会被SmallProf跟踪
    - `$DB::profile = 1;`
  - 指定要跟踪的模块名称，主程序用'main'表示，例如
    - use Data::Dumper; use TK;
    - %DB::package = ('main' => 1, 'Data::Dumper' => 1);
    - 只跟踪主程序和Data::Dumper的代码执行情况
  - 不列出执行计数为0的行
    - $DB::drop_zeros = 1;

再次查看2**100和$two**100

```
#!/usr/bin/perl -w
$DB::profile = 0;
$DB::drop_zeros = 1;
%DB::packages = ('main' => 1);
use strict;

use Math::BigInt;

my $two = Math::BigInt->new('2');
my $p;

$DB::profile = 1;
foreach (1..100) {
    scalar $two ** 100;
}
foreach (1..46000) {
    scalar 2 ** 100;
}

1;
```

- 速度大约相差460倍

```
perl -d:SmallProf bigpower.pl
type smallprof.out
```

```
================ SmallProf version 2.00_03 ================
                    Profile of bigpower.pl                    Page 1
=============================================================
    count wall tm  cpu time line
        1 0.000165 0.000000     2:$DB::profile = 0;
      101 0.000281 0.000000    13:foreach (1..100) {
      100 0.376569 0.370000    14:    scalar $two ** 100;
    46001 0.084747 0.330000    16:foreach (1..46000) {
    46000 0.080029 0.360000    17:    scalar 2 ** 100;
        1 0.000017 0.000000    20:1;
```

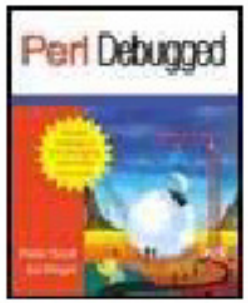
Profiler速度瓶颈分析器

- CPAN查找perlperf
  - Perl Performance and Optimization Techniques

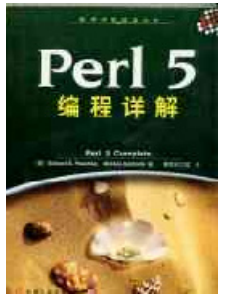
- 小型模块：基于块/模块的Devel::Profile
  - 用法 perl -d:Profile mycode.pl args，查看prof.out

- 大型模块：Devel::NYTProf
  - 可能用到POSIX clock_gettime()、Time::HiRes等
  - perl –d:NYTProf mycode.pl args 可能跑得很慢
  - nytprofhtml --open 查看报告
  - 运行时控制：
    - DB::disable_profile()/enable_profile()/finish_profile()

一些建议和进一步学习



- 留意typoerror
- 每次遇到错误后要仔细分析
- 学Peter Scott, Ed Wright "Perl Debugged", Addison Wesly 2001。其中译本打印错误较多，尚可阅读
- 学习E.S. Peschko, etc "Perl 5 Complete"，Chapter 22-23，机械工业出版社有中译本"Perl 5 编程祥解"
- 学习Perl的调试器, 看perldebtut, perldebug
- 自己编写Perl调试器，看perldebguts帮助主页
- 编写Regression Test代码，多测试边界条件
- 使用覆盖率和速度分析，了解代码执行情况
- 学习Devel::类的模块



McGraw-Hill 1998

其他需要知道的内容和主页

- perldiag　解释了perl的出错信息
- perlsytle　一些可供参考的perl编程风格
- use diagnostics; 比perl –w更详细的警告信息
- use Carp qw(cluck); carp/cluck/croak/confess
  - confess打印全部调用堆栈并终止，可以替代die
- %SIG的用法，截获系统信号量和特殊的__DIE__, __WARN__
  - `$SIG{INT} = sub { die "Ooops\n"; };`
  - `use Carp;`
  - `$SIG{'__DIE__'} = sub {confess "@_"; };`
- 记得使用Data::Dumper()和Tie::Watch()
  - Tie::Watch(): 跟踪变量赋值、访问等情况，查找数据错误时十分有效!

Tie::Watch实例

```
#!/usr/bin/perl -w
use strict;
use Tie::Watch;

my @arr;
my $watch = new Tie::Watch(
     -variable => \@arr,
     -store => \&store,
     -fetch => \&fetch,
     -destroy => sub {print "Destroy.\n"},
     );
@arr = (3, 1, 4, 16);
print "@arr\n";

sub store {
     my ($tie, $k, $v) = @_;
     $tie->Store($k, $v);
     print "Store $v to index $k.\n";
}

sub fetch {
     my ($tie, $k) = @_;
     my ($v) = $tie->Fetch($k);
    print "Fetch index $k, value is $v.\n";
     $v;
}

1;
```

- 截获变量的各种操作
- for all: -fetch, -store, -destroy
- arr: -clear, -extend, -fetchsize, -pop, -push, -shift, -splice, -storesize, -unshift.
- hash: -clear, -delete, -exists, -firstkey, -nextkey
- 详见Perl手册

运行结果

```
C:\>tie.pl
Store 3 to index 0.
Store 1 to index 1.
Store 4 to index 2.
Store 16 to index 3.
Fetch index 0, value is 3.
Fetch index 1, value is 1.
Fetch index 2, value is 4.
Fetch index 3, value is 16.
3 1 4 16
Destroy.

C:\>
```