PB173 Perl

04 Operácie nad zoznamami 2

Roman Lacko <xlacko1@fi.muni.cz>

Obsah

Operácie s dátovými štruktúrami (pokračovanie)

Autovivifikácia

Dereferencia v kontexte, ktorý predpokladá existenciu referovanej hodnoty, ju automaticky vytvorí, ak ešte neexistuje.

ex01t-autovivification.pl

```
my $books;

$books->[42]->{title} = "A Hitchiker's Guide to the Galaxy";
$books->[666]->{chapters}->[0]->{demons}++;
```

Takto je možné ušetriť si vytváranie medziľahlých štruktúr:

```
# $books //= [];                                  # Unnecessary.
# $books->[73] //= {};
$books->[37]->{title} = "A Game of Thrones";      # This is enough.
```

Autovivifikácia

Niekedy však so sebou nesie nečakané problémy. Napríklad,

```
my $books;

delete $books->[42]->{title};

say scalar $books->@*;  # 43!
```

Podobné správanie spôsobí aj `exists` alebo `defined`.

Autovivifikácia

Prekvapenie môže priniesť aj oživenie parametra vo funkcii:

```
sub autovivify($ref) {
    exists $ref->[0]->[0];
}

my $a;
autovivify($a);
say defined $a->[0];        # No.

my $b = [];
autovivify($b);
say defined $b->[0];        # Sí.
```

Perl priradí do $a vytvorenú hodnotu.
Stále je to však obyčajná lokálna premenná, ktorá rozsah funkcie neprežije.
Pri $b sa modifikuje už existujúce pole, samotná referencia sa nezmení.

Odbočka: `undef` v číselnom kontexte

Niektoré iné aspekty Perlu sa chovajú podobne ako autovivifikácia, aj keď ňou striktne nie sú.

Napríklad operátory `+=`, `-=`, `++` a `--`.

```
my ($a, $b, ...);

say $a += 1;        # 1, no warning
say $b -= 5;        # -5, no warning
say ++$c, --$d;     # 1 -1, no warning

say $e *= 1;        # 0, but with a warning
```

Odbočka: `undef` v číselnom kontexte

Operátor `++` je navyše *magický*:

```
say $x++;                 # 0, no warning
say $y--;                 # Prints nothing and complains

my $a = "Head22";         # Works only for ‹[a-zA-Z]*[0-9]*›
my $b = "A9";
say ++$text, ++$b;        # Head23 B0
```

To môže byť niekedy okrajovo užitočné:

```
EXPR while $i++ < LIMIT; # ‹0 .. LIMIT› instead of ‹undef, 1 .. limit›.

my $passwd = 'aaaa';
++$passwd while !login("GLaDOS", $passwd);
```

Ďalšie operátory nad zoznamom

```
reverse LIST
```

- V zoznamovom kontexte vráti zoznam s opačným poradím prvkov.
- V skalárnom kontexte spojí hodnoty do reťazca a obráti ho.

Ďalšie operátory nad zoznamom

```
sort BLOCK LIST          # Note the lack of comma!
```

- Zoradí prvky zoznamu podľa výsledku porovnania v prvom argumente. Očakáva sa `< 0`, `0` alebo `> 0` (ako `strcmp()` alebo `qsort()` v C).
- `BLOCK` dostane prvky v lexikálnych premenných `$a` a `$b`. *

```
sort { $a <=> $b } (11, 10, 1, 5);
```

- Ak `BLOCK` neuvedieme, `sort` zoradí prvky ako reťazce, lexikograticky (`cmp`).

Aký algoritmus `sort` používa? Dá sa zmeniť?
`perldoc -D sort`

Ďalšie operátory nad zoznamom

```
sort FUNCTION LIST      # Again, no comma!
```

- `FUNCTION` dostane prvky v *globálnych* premenných `$a`, `$b`.
  Preferujte radšej `BLOCK`.

`sort` predá funkcii parametre civilizovane, ak má *prototyp* `($$)`:

```
sub compare :prototype($$) ($a, $b);
sort compare @numbers;
```

O prototypoch bude reč (možno) niekedy neskôr.

Ďalšie operátory nad zoznamom

```
grep BLOCK LIST         # No comma here either!
grep EXPRESSION, LIST
```

- V zoznamovom kontexte vráti hodnoty, pre ktoré je `BLOCK` pravdivé. Pre `EXPRESSION` vráti hodnoty, ktoré vyhovujú regulárnemu výrazu.
- V skalárnom kontexte vráti počet vyhovujúcich hodnôt.
- Hodnotu testovanej premennej dostaneme v `$_`.

```
grep { defined $_ } @nubmers;
grep { defined } @numbers;      # Same thing.
grep /^A/, @strings;            # Strings beginning with ‹A›.
grep { !/^B/ } @strings;        # This works too.
```

Regulárne výrazy budú podrobne neskôr.

Ďalšie operátory nad zoznamom

```
map BLOCK LIST          # No comma here either!
map EXPRESSION, LIST
```

- Podobné vlastnosti ako `grep`, ale aplikuje na prvky transformáciu popísanú v `BLOCK` alebo `EXPRESSION`, a vráti nový zoznam.

```
map { $_ + 1 } @numbers;
map { $_ % 2 == 0 ? ($_, $_) : () } @numbers;
map s/(\w)\1/#/gr, @strings;
```

Vyskúšajte `ex04p-filters.pl`

Symbolické referencie

Type glob

V Perli existuje ešte jeden *sigil*, ktorý sme dosiaľ nepoužívali, `*`.
Volá sa *type glob* a umožňuje pristupovať k tabuľke symbolov.

Symboly sú v tomto kontexte *globálne* premenné (v Perli *package variables*).

Package variables

Deklarujú sa **bez** `my` a s plne kvalifikovaným menom.
Východzí menný priestor sa volá `main`, stačí však použiť prázdne meno.

```
# $pi = 3.14;       # This would complain.
$main::pi = 3.14;   # This creates a package (global) variable.
$::pi = 3.14;       # Same thing.
```

Package variables

Ku *package variables* môžeme pristupovať odkiaľkoľvek.
Majú teda charakter *globálnych* premenných.

Nepoužívajte ich pre skaláry, polia a hashe, možno jedine ako konštanty.

Tabuľku symbolov získame ako (pseudo)hash syntaxou `%NAMESPACE::`:

```
say foreach keys %::;   # Same as ‹%main::›.
```

K jednotlivým typom v tabuľke pre symbol `pi` potom môžeme pristúpiť takto:

```
say *::pi{SCALAR};      # SCALAR(…)
say *::pi{SCALAR}->$*;  # Value of the scalar.
```

V týchto menných priestoroch žijú aj *funkcie*.

Symbolické referencie

Bežné referencie sú skaláry, ktoré obsahujú „adresu“ inej premennej.
Symbolické referencie obsahujú *meno* inej premennej.

```
{                               # Scope where symbolic refs will be allowed
    no strict 'refs';           # ‹strict› is turned on by ‹use v5.32›

    $::pi = 3.14159;
    my $symref = "pi";
    say $symref->$*;            # 3.14159
}                               # Disallow symbolic refs again
```

Ak musíte, používajte `no strict 'refs';` v čo najmenšom rozsahu!

Symbolické referencie sú silný, ale nebezpečný nástroj.
Nevypínajte `strict` globálne.

Symbolické referencie

Môžeme ich použiť na vytváranie funkcií za behu:

```
sub spawn($name) {
    no strict 'refs';           # Limited only to the scope of ‹spawn›.
    *$name = sub ($arg) {       # Note the type glob
        say "$name says $arg!";
    };
}

spawn("foo");           # ‹foo()› is a valid function from now on
foo("bar");             # foo says bar!
```

Toto je užitočné v prípade, že chceme vyrobiť niekoľko funkcií
na základe nejakého spoločného predpisu.

Alternatívou je jedna funkcia, ktorá tento predpis očakáva v parametri.
V závislosti na okolnostiach to však môže byť menej pohodlný variant.

Moduly

Jednou z veľkých výhod Perlu je veľké množstvo rozširujúcich modulov.

MetaCPAN je vyhľadávací nástroj pre moduly:
https://metacpan.org/

Niektoré moduly sú zabudované priamo v Perli:
https://perldoc.perl.org/modules

Prítomnosť modulu v jadre Perlu je možné zistiť programom `corelist`:

```
$ corelist List::Util
```

Ako pridať modul do Perlu

Najprv sa pokúste zistiť, či požadovanú funkcionalitu nemá Perl už v jadre.

1. Preferujte inštaláciu modulov zo systémového správcu (`apt`, `yum`, ...).
2. Ak modul v systéme neexistuje a máte **Gentoo**, skúste `g-cpan`.
3. Inak ho musíte inštalovať sami:

```
$ perl -MCPAN -e 'CPAN::shell'
cpan[1]> install Some::Module
```

```
$ cpan -I Some::Module
$ cpanm Some::Module
```

4. Ako najhoršiu možnosť si môžete preložiť zdrojový kód z MetaCPAN.
   Good luck and have fun.

Ako moduly používať

```
require FILENAME
require MODULE
```

- Sprístupní modul z uvedeného súboru.
- Neimportuje z neho žiadne symboly, musíme použiť `$NAMESPACE::SYMBOL`.
- Ak je parametrom slovo bez úvodzoviek, predpokladá sa názov modulu a cestu k súboru si odvodí sám.

```
require 'List/Util.pm';          # Perl Modules have ‹pm› suffix usually.
require List::Util;              # Basically the same thing.
```

Ako moduly používať

```
use MODULE
use MODULE LIST
use MODULE ()
```

- Ako `require MODULE`, ale navyše umožní importovať symboly modulu.
- Zoznam symbolov je u niektorých modulov možné zmeniť parametrom.

```
use List::Util;              # Imports no symbols by default.
use List::Util qw(shuffle); # Now we can say ‹shuffle› without namespace.

use Data::Dumper;            # Imports ‹Dumper()› by default.
use Data::Dumper ();         # Imports nothing.

# Some modules have more complex import semantics:
use Log::Any '$log';         # Creates ‹$log› variable in our namespace.
```

Výber štandardných modulov

CORE

V mennom priestore CORE žijú všetky zabudované Perl funkcie.

Používa sa napríklad v prípade, že nejaký modul omylom prekryje zabudovanú funkciu.

```
use Some::Bad::Module;  # ‹reverse› gets shadowed.

reverse @list;          # ‹Some::Bad::Module::reverse› gets called
CORE::reverse @list;    # Built-in ‹reverse› gets called.
```

Výber štandardných modulov

`Data::Dumper`

Vráti reťazcovú reprezentáciu zložitej štruktúry. Veľmi užitočné na ladenie.

```
use Data::Dumper;       # Imports ‹Dumper› by default.

say Dumper($something);
```

`FindBin`

Definuje *package variables*, ktoré obsahujú cestu k skriptu alebo adresára, v ktorom sa skript nachádza.

```
use FindBin qw($Bin $RealBin $Script $RealScript);
say "Hello, this is $RealScript in $RealBin.";
```

`$Real*` premenné rezolvujú symbolické odkazy.

Getopt::Long

Spracovanie prepínačov programu.

```
use Getopt::Long;       # Imports ‹GetOptions› by default.

my $options = {};
GetOptions($options, qw(
    h            help
    output|o=s   set|S=s%
));

say "output: $options->{output}";
say "keys:   ", join ', ', map { ... } sort keys $options->{set}->%*;
say "args:   @ARGV";
# $ perl PROGRAM --output file.xml -S x=1 -S y=2 source1.xml source2.xml
# output: file.xml
# keys:   x=1, y=2
# args:   source1.xml source2.xml
```

List::Util

```
use List::Util qw(any all ...);
```

Veľká sada rozširujúcich funkcií na prácu so zoznamami:
https://metacpan.org/pod/List::Util

Scalar::Util

```
use Scalar::Util qw(looks_like_number);
```

Rozširujúce funkcie na prácu so skalármi:
https://metacpan.org/pod/Scalar::Util

Prehľad štandardných modulov

Honorable mentions

`Archive::Tar`
: Práca s TAR archívmi

`Encode`
: Kódovanie a dekódovanie reťazcov

`Devel::Peek`
: Nástroj na pomoc pri ladení hodnôt premenných

`Fcntl`, **`POSIX`**
: Sprístupní niektoré POSIXové funkcie

`JSON:PP`
: Serializácia a deserializácia JSON

Prehľad štandardných modulov

Honorable mentions

`IO::*`
: Moduly na prácu so vstupom a výstupom

`IPC::*`
: Komunikácia medzi procesmi

`Test::*`
: Testovacie nástroje

`Pod::Usage`
: Generátor nápovedy z dokumentácie skriptu alebo modulu (POD).

Špeciálne premenné

Niekoľkokrát sme narazili na premennú `$_`.
Perl má niekoľko podobných *špeciálnych* premenných.

Celý zoznam viď https://perldoc.perl.org/perlvar.

```
use English;
```

Modul, ktorý pomocou anglické *aliasy* pre špeciálne premenné.

Špeciálne premenné

`$_` (`$ARG`)

- *Implicitná* premenná.
- Vo `for`(`each`), kde premennú nepomenujeme alebo nemôžeme:

```
$sum += $_ foreach @numbers;
```

- Dostupná v bloku `grep`, `map`, `given`.
- Niektoré zabudované funkcie ju použijú, ak im nedáme žiadny parameter:

```
say;            # ≈ say $_;
length;         # ≈ length $_;
```

Používajte ju radšej len tam, kde musíte.

Špeciálne premenné

`@_` (`@ARG`)

- Obsahuje argumenty funkcie.
- Implicitný argument pre `pop` a `shift`.
- Pred `signatures` to bol jediný spôsob, ako pristúpiť k argumentom:

```
sub inspect {
    my ($what, @values) = @_;
}

sub inspect {
    my $what = shift;   # ‹shift› uses ‹@_› by default.
}
```

Signatúry sú dnes pohodlnejšie a bezpečnejšie (kontrolujú aritu).

Špeciálne premenné

`$"` (`$LIST_SEPARATOR`)

- Používa sa ako oddeľovač pri interpolácii zoznamu v reťazci.
- Východzia hodnota je `' '` (medzera).

```
$" = ':';
my @values = qw(a b c);
say "@values";              # a:b:c
```

`$$` (`$PID`)

- Číslo procesu, podobne ako v shelli.

Špeciálne premenné

`$0` (`$PROGRAM_NAME`), `@ARGV`

- `($0, @ARGV)` je ekvivalent poľa `argv` v C.

```
say "Arguments: $0 @ARGV";

# $ perl ex09t-perlvar.pl --test arg
# Arguments: ex09t-perlvar.pl --test arg
```

`%ENV`

- Premenné prostredia

```
say $ENV{HOME};          # /home/pazuzu
```

Špeciálne premenné

`$!` (`$ERRNO`)

Magická premenná: použitá ako číslo vráti kód,
použitá ako reťazec vráti popis chyby (ako `strerror()` v C).

```
$! = 130;
say 0 + $!;      # 130
say "$!";        # Owner died
```

Magic variables je v Perli polo-oficiálny názov pre premenné,
ktoré sa za určitých okolností správajú inak, než bežné premenné.

V príklade `ex09t-perlvar.pl` sú ešte ukážky premenných `$,` a `$\`.