PB173 Perl

03 Hashe, referencie

Roman Lacko <xlacko1@fi.muni.cz>

Obsah

Asociatívne polia

Poslednou základnou dátovou štruktúrou sú *asociatívne polia*, alebo *hashe*.

```
my $scalar;
my @array;
my %hash;
```

Kým hodnoty v poli sú indexované číslami, hodnoty v hashi sú indexované kľúčmi.
Kľúče sa **vždy** interpretujú ako reťazce, hodnoty môžu mať ľubovoľný typ.

Inicializácia

Hash sa inicializuje zoznamom:

ex01t-hash.pl

```
my %languages = (
    'sk', 'slovak',
    'cz', 'czech',
    'uk', 'english',
);
```

Operátor =>

Pri inicializácii hashu môžeme použiť trochu pohodlnejšiu a jasnejšiu syntax:

ex01t-hash.pl

```
my %languages = (
    sk => 'slovak',
    cz => 'czech',
    uk => 'english',
);
```

Čo ak chceme ako kľúč hodnotu z funkcie?
Vyskúšajte si `ex02p-fat-comma.pl`.

Prístup k prvkom

K prvkom hashu pristúpime operátorom `{KEY}`. Podobne ako pri poli, aj tu *sigil* znamená „typ“ hodnoty, ktorú očakávame:

```
my %languages;

say $languages{'en'};
```

Prístup k prvkoom

`{VALUE}` vynúti reťazec, takže môžeme písať aj `{en}`. Pozor však na hodnoty, ktoré nie sú reťazce:

ex01t-hash.pl

```
my %numbers = (
    0 => 'integer zero',
    0.0 => 'float zero',
);
```

Čo vypíše `$numbers{0}`?

Kontext

Hash v zoznamovom kontexte vráti zoznam kľúčov a hodnôt.

Poradie týchto hodnôt nie je garantované. Nespoliehajte sa naňho.

Pri vytváraní hashu zo zoznamu sa v prípade konfliktu ponechá vždy posledná hodnota kľúča:

```
my %defaults = (
    user => 'root',
    path => '/etc',
);

my %config = (
    %defaults,
    user => 'daemon',
);
```

Výrezy

Pri hashi máme k dispozícii dva druhy výrezu:

```
my %hash;

my @values = @hash{KEYS...};
```

- Vráti hodnoty kľúčov uvedených v zozname KEYS.

```
my %subhash = %hash{KEYS...};
```

- Vráti podčasť hashu pre uvedené kľúče KEYS.

Výrezy

Pre zhrnutie:

- `[]` alebo `{}` používame podľa toho, z **čoho** vyberáme hodnoty
- `@` alebo `%` používame podľa toho, čo má byť **výsledok**

```
@x[INDICES];        # [] → ‹x› is array   @ → we want array as a result
%x[INDICES];        # [] → ‹x› is array   % → we want hash of indices and
values

@x{KEYS};           # {} → ‹x› is hash    @ → array of values as a result
%x{KEYS};           # {} → ‹x› is hash    % → (sub)hash as a result
```

Výrezy

Koľko rôznych premenných používa tento výraz?

```
$x{$x[$x]}
```

$x
: skalárna premenná $x

$x[…]
: pole @x

$x{…}
: hash %x

Ďalšie operácie

```
keys HASH
values HASH

keys ARRAY
values ARRAY
```

- Vráti zoznam kľúčov, resp. hodnôt.
- Od Perl 5.12 funguje aj na poli, vráti zoznam indexov resp. hodnôt.

Kľúče ani hodnoty hashu nie sú v žiadnom vopred určenom poradí.

Ak chcete kľúče zoradiť, použite `sort keys HASH`.
`sort` uvidíme podrobnejšie neskôr.

Ďalšie operácie

```
exists EXPRESSION
defined EXPRESSION
```

Podobne ako pre pole overí existenciu kľúča, alebo či má definovanú hodnotu.

```
$hash->{u} = undef;

say exists $hash->{u};      # This exists …
say defined $hash->{u};     # … but is not defined.
```

Ďalšie operácie

```
delete EXPRESSION
```

Zmaže hodnotu a kľúč z hashu. Môžeme uviesť aj výrez.

```
delete $languages->{cs};
delete $languages->@{qw(cs sk)};
```

Tento operátor je možné použiť aj na prvky poľa.
Nie je to odporúčané, chová sa to divne.

Ďalšie operácie

Ako iterovať cez kľúče a hodnoty?

```
my %hash = (...);

foreach my $key (keys %hash) {
    my $value = $hash{$key};
    ...
}
```

Vyskúšajte si `ex03p-hash.pl`.

Ďalšie operácie

```
each HASH
each ARRAY
```

- Pri každom volaní vráti ďalšiu dvojicu `(key_or_index, value)`.
- Po poslednom prvku vráti prázdny zoznam.
- Iterátor je viazaný na daný kontajner.

```
while (my ($key, $value) = each %hash) {
    ...
}
```

Čo sa stane, ak vnoríme `each`?
Vyskúšajte `ex04t-each-bad.pl`.

Referencie

Hodnotou skalárnej premennej môže byť odkaz (*referencia*) na inú hodnotu.

ex05t-refs.pl

```
my $a = 42;
my $ref_a = \$a;
my $ref_number = \32;
my $ref_string = \"Perl";
my $ref_undef = \undef;
```

Odkazovanou hodnotou môže byť aj pole, hash, funkcia, objekt atď.

Dereferencia

K hodnote pod referenciou môžeme pristúpiť dereferenciou dvoma spôsobmi.

Circumfix notácia

```
# SIGIL { REF }     # Or without { } for simple expressions.
${$ref}, $$ref      # scalar
@{$ref}, @$ref      # array
```

Postfix notácia (od Perl 5.20)

```
# REF -> SIGIL *
$ref->$*            # scalar
$ref->%*            # hash
```

Circumfix alebo Postfix notácia?

Postfix notácia (`$ref->‹S›*`) môže vyzerať na prvý pohľad zložitejšie, než circumfix (`‹S›$ref`). Lepšie sa však číta pri hlbokých štruktúrach.

Z poľa kníh uložených ako hashe chceme autorov prvej knihy

```
# Circumfix
@{${$$books[0]}{authors}}
```

```
# Postfix
$books->[0]->{authors}->@*
```

Používajte ten spôsob, ktorý sa vám páči viac, buďte však **konzistentní**.

Dereferencia

Obyčajné priradenie do premennej referenciu zruší:

```
$a   = 42;
$b   = \$a;
```

```
$b   = 666;
say "$a $b $$b";    # → error, $b is not a reference anymore;
```

Dereferencia vráti *l-value*, do ktorej môžeme priradiť inú hodnotu:

```
$$b = 666;          # The old way
$b->$* = 666;       # The new way (postfix dereference)

say "$a $b $$b";    # → "666 SCALAR(0x…) 666";
```

Viacúrovňové (de)referencie

```
my $x = 4;
my $ref2 = \\$x;

# ${$$ref2} = 8;
# $$$ref2 = 8;

$ref2->$*->$* = 8;
```

Vyskúšajte si: `ex06p-refs2.pl`

Operátor `ref`

```
ref $s;
```

Vráti typ referencie v premennej:

- `""` (nepravdivá hodnota) ak `$s` neobsahuje referenciu
- `"SCALAR"` ak `$s` je referencia na skalár
- `"CODE"` ak `$s` je referencia na funkciu
- `"ARRAY"` ak `$s` je referencia na pole
- `"HASH"` ak `$s` je referencia na hash
- `"REF"` ak `$s` je viacúrovňová referencia
- ... ďalšie hodnoty neskôr

Vyskúšajte si: `ex07p-deref.pl`

Referencie na polia

```
my @array = qw(Hello World);
my $arrayref = \@array;
```

Ak referencia ukazuje na pole, môžeme pristupovať k prvkom takmer priamo:

```
say $array[0];
# say ${$arrayref}[0];  # or $$arrayref[0]
say $arrayref->[0];
```

Prípadne môžeme odkaz na pole rozbaliť na zoznam:

```
# my @new_array = @{$array};    # or @$array
my @new_array = $array->@*;     # Postfix dereference
```

Referencie na polia

Z odkazov na polia môžeme robiť aj výrezy:

```
# my @slice = @{$array}[1..5];  # or @$array[1..5];
my @slice = $array->@[1..5];
```

Môžeme priamo zistiť aj posledný index:

```
# say $#$arrayref;
say $arrayref->$#*;
```

Referencie na polia

Ak chcete používať staršiu (circumfix) notáciu,
dávajte si pozor na zátvorky.

Aký je rozdiel medzi týmito výrazmi?

```
say $$$ref[0];
say ${$$ref}[0];
say ${$$ref[0]};
say $$ref[0][0];
```

Referencie na polia

Referenciu na pole môžeme inicializovať priamo

```
my @array = (1, 2, 3);

my $arrayref = \@array;
my $arrayref = [1, 2, 3];
my $arrayref = [qw(one two three)];
```

Syntax `[ LIST ]` sa volá *Anonymous array*.

Výraz `\( A, B, … )` je len skratka za `( \A, \B, … )`!

Referencie na hashe

Podobne ako pre polia, Perl má špeciálnu syntax aj pre hashe:

```
my $hashref = { key => value, ... };

# say ${$hashref}{key}          # or $$hashref{key}
say $hashref->{key};

# my %hash = %{$hashref};       # or %$hashref
my %hash = $hashref->%*;

foreach my $key (keys $hashref->%*) {
    ...
}

my @slice = $hashref->@{keys...};
my %slice = $hashref->%{keys...};
```

Hlboké štruktúry

Pole aj hash môžu ako hodnoty obsahovať len skaláry. Tieto však môžu byť referenciami:

```
my $books = [
    { authors => ['J. R. R. Tolkien'],
      title => 'The Fellowship of the Ring',
      ... },
];

say $books->[0]->{title};           # The Fellowship of the Ring
say $books->[0]->{authors}->[0];    # J. R. R. Tolkien
```

Pri hlbokých štruktúrach je nutná len prvá šípka:
`$books->[0]{authors}[0]`
typicky sa však píšu aj ostatné.

Referencie na funkcie

ex08t-functions.pl

```
sub foo($arg1, $arg2) {
    ...;
}

my $ref = \&foo;

# &$ref(1, 2);
# &{$ref}(1, 2);

$ref->(1, 2);
```

Tieto referencie môžeme predávať ako parametre iným funkciám.

Odbočka 1: Čo je &?

& je *sigil* podobne ako $, @ a %, akurát sa obvykle písať nemusí.

```
foo(1);
foo 1;
&foo(1);

# &foo 1;       # This does not work
```

Kedy má & zmysel okrem referencií?
`ex09t-sub-sigil.pl`

Odbočka 2: Elipsa

Ak sa `...` nachádza v tele funkcie na mieste výrazu, tak pri vyhodnotení spôsobí chybu `Unimplemented`:

```
sub stub() {
    ...
}

stub;           # → "Unimplemented"
```

Tento operátor sa tiež volá *yada yada*.

Nepliesť si s *operátorom* v skalárnom kontexte `A ... B`.

Anonymné funkcie

ex08t-functions.pl

```
my $inc = sub ($a) { $a + 1 };
say $inc->(1);
```

Ak funkcia nemá `return`, vráti hodnotu posledného výrazu.
Vynechávajte ho však len u veľmi jednoduchých funkcií.

Anonymné funkcie môžeme rovno napísať ako parameter inej funkcie:

ex08t-functions.pl

```
sub apply($v, $f) {
    return $f->($v);
}

apply(42, sub ($n) { $n + 1 });
```

Uzávery

Anonymné funkcie môžu pristupovať k premenným vo vonkajšom rozsahu:

```
sub power_f($exponent = 2) {
    return sub ($n) { $n ** $exponent };
}

say (power_f->(5));     # → 25
say (power_f(3)->(5));  # → 125
```

Vyskúšajte si: `ex10p-callback.pl`

Rekurzívne anonymné funkcie

Lebo prečo nie.

Referenciu na aktuálnu funkciu Perl sprístupní v symbole `__SUB__`.

```
sub $fact = sub ($n) {
    return if $n < 0;
    return 1 if $n <= 1;
    return $n * __SUB__->($n);
};
```