Perl 5.6/5.8系を使おう

早川 真也 <hayakawa@edge.co.jp>

Shibuya.pm Technical Talks

2002/1/31

株式会社オン・ザ・エッヂ

概要

- Perl 5.6.0/5.8.0 新機能の紹介
  - perldelta
  - ラクダ本
- 【特集】コンパイラの存在
- ごちゃまぜ Tips 集

新しい機能

- **`sort()`**の第1引数にコードリファレンスが使用可能に [560]

  - **`sort $coderef @foo;`**

新しい機能

- **`exists()`**の引数にサブルーチンが指定可能に [560]

  - **`print "foo¥n" if exists &foo;`**
  - **`sub foo;`**

新しい機能

- リファレンスへのリファレンスが少し特殊に[580]

```
– $ perl -e 'print ¥¥$foo,"¥n"'
– REF(0x816c15c)
– $ perl -e 'print ref(¥¥$foo)'
– REF
```

新しい機能

- `-c` で実行された時は **`END`** ブロックを実行しないよう改善 [560?]

- **`BEGIN`** で **`die`** されても **`END`** ブロックを間違いなく実行するよう改善 [580]

新しい機能

- **`import`** にならい、 **`unimport()`**が定義されていない場合でも **`no Module;`** がエラーを出さないように [561]

パフォーマンスの向上

- **sort()** の高速化 [560]

```
$ perl -MBenchmark -e ¥
'timethese(-3,{sort => ¥
q/sort {$a <=> $b} reverse 0..99/})‘

[5.005_03]
@ 13805.67/s (n=41417)
[5.6.1]
@ 177970.57/s (n=563110)
[5.8.0]
@ 187452.41/s (n=574073)
```

パフォーマンスの向上

- 代入の高速化 [560]

```
$ perl -MBenchmark -e ¥
'timethese(-3,{try => ¥
q/my $foo=qw(foo)/})`

[5.005_03]
@ 71227.20/s (n=215351)
[5.6.1]
@ 422473.82/s (n=1333433)
[5.8.0]
@ 421550.34/s (n=1300878)
```

パフォーマンスの向上

- サブルーチン呼び出しの高速化 [560]

```
$ perl -MBenchmark -e ¥
'timethese(-3,{call => ¥
q/sub{}->()/})‘

[5.005_03]
@ 99404.00/s (n=298212)
[5.6.1]
@ 154403.32/s (n=476479)
[5.8.0]
@ 141780.46/s (n=433095)
```

パフォーマンスの向上

- ハッシュ操作の高速化 [560]

```
use Benchmark;
use strict;

my %hash = map {$_ => 1} 'a'..'z';

timethese(-3,
    { delete => sub { my %h = %hash;
        delete $h{$_} for 'a'..'z' }
    , each   => sub { while(each %hash) {} }
    , values => sub { values %hash }
    , list   => sub { my @list = %hash }
    } );
```

ハッシュ操作の高速化

```
[5.005_03]
 delete: @ 1934.94/s (n=5956)
   each: @ 4089.23/s (n=12683)
   list: @ 2500.47/s (n=7560)
 values: @ 144276.00/s (n=432828)
[5.6.1]
 delete: @ 2501.61/s (n=7798)
   each: @ 5465.85/s (n=17337)
   list: @ 4695.55/s (n=14857)
 values: @ 610335.12/s (n=1902529)
[5.8.0]
 delete: @ 2735.01/s (n=8611)
   each: @ 5976.49/s (n=18910)
   list: @ 3786.07/s (n=11654)
 values: @ 488228.79/s (n=1884258)
```

パフォーマンスの向上

- **map()**の高速化 [560]

```
$ perl -MBenchmark -e ¥
'timethese(-3,{map => ¥
q/map {$_ => 1} 0..50/})‘

[5.005_03]
@ 2341.33/s (n=7024)
[5.6.1]
@ 3820.33/s (n=12237)
[5.8.0]
@ 4237.47/s (n=13209)
```

- 大量のモジュールが追加

```
B::Concise, Devel::PPPort,
Filter::Simple, Memoize,
if, sort, open,
Scalar::Util, List::Util,
Hash::Util, ... [580]
```

- 【特集】コンパイラの存在
- ごちゃまぜ Tips 集

- コンパイラとインタープリタ
  - フェーズの違い
  - コンパイルフェーズ
  - 実行フェーズ

```
sub baz;
my $foo = 1;
bar();
baz();
sub foo { … }
BEGIN { … }
INIT  { … }
CHECK { … }
END   { … }
sub bar { … }
foo();
sub baz { … }
```

- コンパイラに影響を与えるもの

  my, our, sub, BEGIN,

  use, require, プロトタイプ, …

- コンパイル時と実行時

- CHECKはコンパイルフェーズの最後
- INITは実行フェーズの最初

  (CHECK は LIFO順、INITはFIFO順)

```
$ perl -e 'eval "INIT{warn q/init/}"`
$ perl -e 'BEGIN{eval "INIT{warn q/init/}"}`
init at (eval 1) line 1.
```

- our, local, my の違い

`our` 名前をスコープに限定

`local` 値をスコープに限定

`my` 名前と値の両方をスコープに限定

(ラクダ本 4.8.1 スコープ付き変数宣言)

- パッケージクォートされたクラス

```
1. $obj = new SuperSaiyan;
2. $obj = SuperSaiyan->new;
3. $obj = new SuperSaiyan::;
4. $obj = SuperSaiyan::->new;
```

(ラクダ本 12.3.4 パッケージクォートされたクラス)

- 多重継承をする場合のAUTOLOAD

foo() の呼び出しに失敗した時
sub foo; 宣言のある package を優先
してAUTOLOADが呼び出される。

(ラクダ本 12.5.4 メソッドをオートロードする)

- perly.y を活用する（おまけ）

例：矢印を使ったメソッド呼び出し

```
term ARROW method '(' listexprcom ')'
term ARROW method
```

```
method   :        METHOD
         |        scalar
scalar   :        '$' indirob
indirob  :        WORD
         |        scalar %prec PREC_LOW
         |        block
         |        PRIVATEREF
```

```
$foo->"method";     # NG
$foo->{method};     # NG
$foo->("method");   # NG
$foo->method;       # OK
$foo->$method;      # OK
$foo->$$method;     # OK
$foo->$$$method;    # OK
$foo->${cond ? ¥"foo" : ¥"bar"};#OK
```

- デフォルト変数の利用1

```
for ( @list ) {
    /^foo$/ && do { ... }
    /^bar$/ && do { ... }
    /^baz$/ && do { ... }
}
```

- デフォルト変数の利用2

```
wantarray ?
    @foo : join '', @foo;
```

より

```
sub {wantarray ?
    @_ : join '', @_} -> (@foo);
```

- **local サブルーチン**

```
sub method {
  my $self = shift;
  local *foo =
    sub :lvalue {$self->{foo}};
  ……
  foo() = ‘foo’;
```

- **UNIVERSAL Teleportation**

```
sub method {
    my $self = shift;
        ……
    local *UNIVERSAL::foo =
        sub { $self->{foo} };
```

- ファイル内で部分的に実行する (-x)

```
#!perl
warn 'foo';
__END__
#!perl
warn 'bar';
__END__
#!perl
warn 'bar';
```

- マクロ機能を利用する
  - Filter::Simple
  - -P オプション

- assert 文の除去 (Carp::Assert等)

assert(…) if DEBUG; の代わり

```
use Filter::Simple sub {
    s/assert¥s*¥([^;]+;//g
        unless DEBUG;
};
```

Avoid producing write-only code.
書き込み専用コードを避けよう

(Effective STL 47章)

ソースコードは人に書かれる時間よりも
読まれる時間の方が圧倒的に長い

- **Perl 5.005 と Perl 5.8.0 は別物**
- **ラクダ本はお勧め**