Perl6 で遊ぼう

青木大祐 / @VienosNotes

情報科学類

Tsukuba.pm #2

概要

- Perl6 とは？
  - Perl5 との違い
  - Perl6 のオモシロ機能
- 今までの経緯とこれからの話

Perl6 とは？

- Perl5 との後方互換性を切り捨てた新しい言語
- Perl5 を置き換えるものではない
  - 開発もコミュニティも別ライン
- モダンなプログラミング技法を取り入れる
  - 真っ当なクラスベースのオブジェクト指向
  - 充実したメタプログラミング
  - その他の先進的 (?) な機能
- 仕様ベースの開発

要するに…

Larry Wallの考えた
最強のプログラミング言語

Perl5 との違い

- コンテキストは無くなり、シジルは分かりやすく
  - 配列の要素にアクセスするときも@を使う
- メソッド呼び出しは－＞ではなく.に
- みんなオブジェクトに
  - リテラル
  - クラスのインスタンス
  - 関数
- 特殊変数も再編
- ある程度の標準ライブラリがある

現状どうなってるか

- 提唱されたのは 2000 年
- 10 年以上経ったが、未だ完成していない
- 完成してないので、当然普及もしてない
  - Perl 歴が長い人は「そんなの言ってた時代もあったね」
  - 最近の人は「何それ？聞いたことも無い」

世間様からは完全に**オワコン**扱い

→しかし、言語的に見ると結構面白い機能が多い

オモシロ機能

- **型制約**
- **関数リテラル**
- Junction
- Grammer

型制約

- 変数や関数の引数にデータ型の制約を書くことができる
- 引数の型によって呼ぶディスパッチを決定できる
  - 実行時オーバーロード
- 制約を満たさない型を検出すると、実行時例外を投げる
  - 静的型付け（コンパイル時検査）ではない

型制約:変数

```
1  my Int $i = 1;
2  # ok
3
4  my Str $s = 1;
5  # Type check failed in assignment to '$s'; expected 'Str' but
       got 'Int'
```

型制約:関数

```
1  sub f (Int $i) returns Int { return $i * 2; }
2
3  say f(1);
4  #> 2
5
6  say f("piyo")
7  #> CHECK FAILED:
8  #> Calling 'f' will never work with argument types (str)
9  #>     Expected: :(Int $i)
10
11 multi sub g (Int $i) { return $i * 2; }
12 multi sub g (Str $s) { return $s x 2; }
13
14 say g(1);
15 #> 2
16 say g("s");
17 #> ss
```

関数リテラル

- map に渡すコールバック関数など、Perl 的記述に置いて関数リテラルを記述する機会は多い
- 簡潔な記述を実現するため、複数の方法が用意されている

Pointy Block

```
1    (1..10).map: -> Int $a { $a ** 2 };
```

- 矢印に続けて引数リスト、その後に本体を書く
- 見た目が格好良いので使う機会が多い

Whatever Star

```
1   say (1..10).map: * ** 2;
```

- *に Perl5 で言うところの$_が暗黙的に代入される
- 記述量は非常に少ないが、見た目が悪魔的で少し戸惑う

Placeholder Variables

```
1  say (1..10).map: { $^a ** 2 };
```

- $^で始まる変数に、引数が順に格納されて行く
- 代入される変数は辞書順で決まる

Junction

変数 i が 0 か 1 であるという処理を、普通はこう書く

```
1  if ($i == 0 | $i == 1) { ... }
```

直感的にはこのように書きたい

```
1  if ($i == 0 | 1) { ... }
```

- これを可能にするのが Junction
- 「0 か 1 である」という状態をオブジェクトとして保持する

Junction の要素について、手続きを平行して適用した後で、(OR ジャンクションなら) 論理和を取る。

```
1  if ($s ~~ /pattern1/ | /pattern2/) { ... }
2  # if $s matches pattern1 or pattern2
```

- バックトラックしないので高速（かもしれない）
- (処理系が対応していれば) 並列で実行される
  - 順番に依存する処理は出来ない

Grammar

- PEG(解析表現文法) をベースとした文字列パース機能
- ふつうの正規表現よりも柔軟な記述が可能

以前書いた Lisp の (不完全な) パーザ

```
1   grammar Lisp {
2     token left { '(' };
3     token right { ')' };
4     token num { \d+ }
5     token str { '"'\w+'"' }
6     token literal { <num>||<str> }
7     token nil { 'nil'|'('  <.ws> ')' }
8     rule ne_sexpr { <atom> || <.left> [<ne_sexpr>+? % <.ws>] [<
          dot> <.ws> <ne_sexpr>]? <.right> }
9     token spform { 'cond' | 'if' | 'define' | 'quote' }
10    rule sp_sexpr { <.left> <spform> [<ne_sexpr>+? % <.ws>]  <.
          right> }
11
12    token symbol { \w+ }
13    token atom { <literal>||<nil>||<spform>||<symbol> }
14
15    token dot { '.' }
16    rule sexpr { <atom> || <sp_sexpr> || <.left> [<sexpr>+? % <
          .ws>] [<dot> <.ws> <sexpr>]? <.right> }
17    rule TOP { ^^ [<sexpr> || <ne_sexpr>] $$ }
18  }
```

Grammar

- 要するに構造を持った名前付き正規表現
- 部分文字列を取り出すのも簡単
- バックトラックの枝刈りも効率が良い (らしい)
- 明確な構造を持った文字列 (JSON, csv, etc...) のパーザが簡潔に書ける

オモシロ機能を見てきた訳ですが

なぜオワコンと呼ばれるに至ったのか？

- 開発が遅い
  - 皆に飽きられてる
- 動作が遅い
  - 製品としては実用にならないレベル
- 仕様が固まっていない
  - いつまで経っても 1.0 がリリースされない

そもそもコミュニティが無い

じゃあコミュニティ作りましょう

- 実は Perl6 勉強会を去年やりました
- 今年もやりたい
  - 参加者募集中です
  - 今年はドキュメント (入門書、仕様書の翻訳) 書きたい

→二学期からは月１くらいでゆるゆるやります