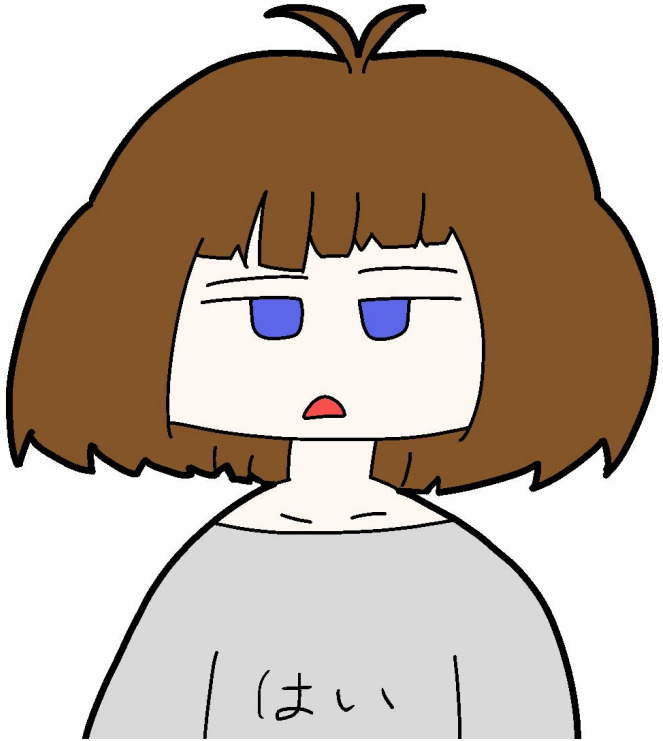
prototypeとjust epic. と私

YAPC::Japan::Online 2022 @utgwkk

自己紹介

- @utgwkk (うたがわきき)
- 株式会社はてな

  Webアプリケーションエンジニア
- KMC (京大マイコンクラブ)
- 最近はTypeScriptを書いています



好きなPerlの言語機能

- wantarray関数
  - 3値を返す関数なので
- DESTROYメソッド
  - オブジェクトの解放に処理を差し込めて便利
  - 今後はdeferでやる方向になる?
- prototype
  - 今日は主にこれの話をします
- 他にいい言語機能があれば教えてください!!

prototypeとは

- サブルーチンの引数の解釈方法を制御できる
  - パーサの挙動が変わる!!
- 書き方
  - sub name (ここに記号をいろいろ書く)
  - sub name : prototype(ここに記号をいろいろ書く)
  - サブルーチンシグネチャのことを考えると後者の書き方がよさそう

prototypeの例

```
sub foo ($) { }

sub bar ($;$@) { }

sub mymap (&@) { }

sub zip (\@\@) { }
zip @xs, @ys;

sub foo () { 1 }
```

prototypeの読み方 (残りはperldoc perlsubで)

sub foo ($) { } # 括弧の中に記号を書く

sub bar ($;$@) { } # ; 以降は省略可能

sub mymap (&@) { } # &はサブルーチン、@はリスト

sub zip (\@\@) { } # リストをリファレンスとして受ける

zip @xs, @ys;

sub foo () { 1 } # 定数をreturnなしで返すとインライン展開される

身近に潜むprototype採用事例 (Try::Tiny)

try { } catch { };

sub try (\&;@)

try-catch構文をprototypeで再現 (末尾のセミコロンが必要)

最新のPerlではtry-catch構文が組み込まれている (こっちもセミコロン不要)

身近に潜むprototype採用事例 (Test2::V0)

```
is $obj, object {
  prop isa => 'Foo';
  call meth => 'blah';
};
```

$obj の性質を宣言的にテストできる

身近に潜むprototype採用事例 (List::MoreUtils)

zip6 @xs, @ys, @zs;

sub zip6

(\@\@;\@\
@\@\@\@\@) # \@ が32個

一番好きなprototypeです

リストを33個渡すと……??

```
use List::MoreUtils qw(zip6);
my @x = (1);
zip6 @x, @x, @x, @x, @x, @x, @x, @x, @x, @x, @x, @x, @x, @x, @x, @x,
@x, @x, @x, @x, @x, @x, @x, @x, @x, @x, @x, @x, @x, @x, @x, @x;

# @x が33個
```

```
Too many arguments for List::MoreUtils::XS::zip6 at zip6.pl line 6, near "@x;
```

prototype (+α) 活用事例

just epic.

“just epic”と書くだけでサーバを起動する方法 - Hatena Developer Blog

just epic.

2011-2022;

とりあえずDeparse

```
% perl -MO=Deparse -e 'just epic. 2011-2022;'
'epic'->just . '2011' - 2022;
-e syntax OK
```

間接オブジェクト記法

- b A と書いたら A->b と解釈される
  - Foo->new() を new Foo() って書ける
- just epicでepicパッケージのjustメソッドを呼び出している
- 新しめのPerl (>= 5.32) では no feature 'indirect'; で無効化できる
- print STDERR "YO" のような記法はno feature 'indirect' 下でも書ける

just epic?

no feature 'indirect';

just epic.

2011-2022;

sub epic::just { }

Unquoted string "just" may clash with future reserved word at - line 2.

(中略)

作戦

- just, epic というサブルーチンをそれぞれ用意する
- 警告ゼロを目指す
  - use strict; use warnings; は義務
  - no warnings; も使わずに済ませたい

just epic. (間接オブジェクト記法なし)

```
no feature 'indirect';
sub just ($) {} # ここでサーバーを起動する
sub epic () { return 1 }

just epic.
2011-2022;
```

prototypeなしでもOK

no feature 'indirect';

sub just {}

sub epic { 1 }

just epic.

2011-2022;

Deparseしたときの美しさはこっちの方が高いかも?

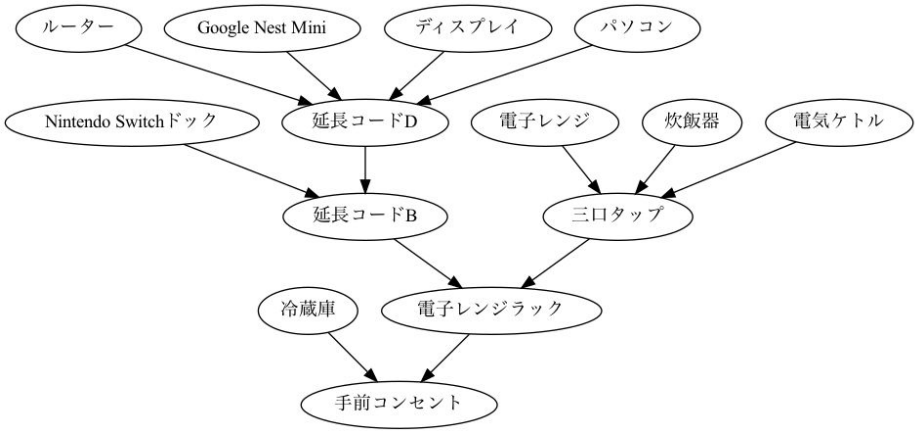
Webサーバーを起動する

- https://gist.github.com/utgwkk/febc9e199ecc3c44eb987ea2a913a461
- ここでデモ

dot言語 (graphviz)

https://blog.utgw.net/entry/2021/10/19/221139

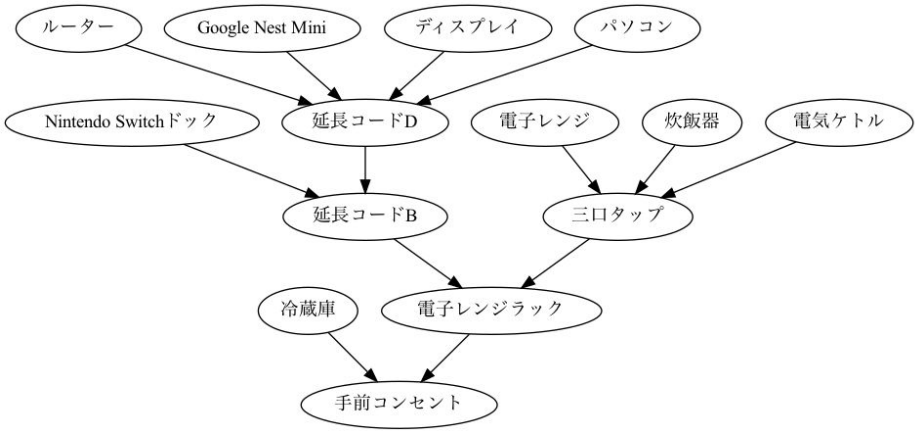
```
digraph haisen {
  電子レンジラック -> 手前コンセント;
  冷蔵庫 -> 手前コンセント;
  …;
}
```



もしかしてこれはPerlでは???

https://blog.utgw.net/entry/2021/10/19/221139

```
digraph haisen {
  電子レンジラック -> 手前コンセント;
  冷蔵庫 -> 手前コンセント;
  …;
}
```



方針

- Perlプログラムの中にdot言語のコードを埋め込む
- Perlとして実行したらdot言語のコードが出力される

適切なprototypeを設定する

sub digraph ($) { }

sub haisen (&) { }

これで digraph haisen { … } と書ける

メソッド呼び出しをハンドリングする

sub UNIVERSAL::AUTOLOAD { … }

あらゆるモジュールの (UNIVERSAL)

存在しないメソッド呼び出しをハンドリング (AUTOLOAD)

(モジュール名)::(メソッド名) を (モジュール名) -> (メソッド名) に変える

できたもの

- https://gist.github.com/utgwkk/47140c80b7a61fbdb6656c7d4ea29ae0
- ここでデモ
- perl dot.pl | dot -Tpng -oout.png

まとめ

- prototypeを使うといろんなことが実現できる
- just epic. を間接オブジェクト記法なしで書ける
- dot言語をPerlプログラムとして解釈できる
    - 今後の課題: グラフ名を自由に指定できるようにする