関ケ原Ruby会議01

PicoRubyに於ける Refinementsの再解釈

橘羽角前下総守情操指南匠均之助

東軍大将斯く申され候

_ko1 @_ko1 · 2020年9月19日
Refinements ツライ #RubyKaigi_SmartHR

_ko1 @_ko1 · 2022年12月15日
権力で**refinements** が強化されてしまう

_ko1 @_ko1 · 2018年11月16日
例えば **refinements** を捨てる

_ko1 @_ko1 · 2020年12月19日
確実に再現するバグをデバッグするのは気持ちがいいなぁ。**refinements** でも許せてしまいそう。でも許せない

_ko1 @_ko1 · 2023年5月11日
Refinements だ！　捕まえろ！
565

_ko1 @_ko1 · 2020年10月22日
返信先: @shugomaedaさん
クラス変数も **refinements** も消しちゃうのがいいんじゃないですかね...

_ko1 @_ko1 · 2019年12月14日
evalや**refinements**はしすべきt

_ko1 @_ko1 · 2020年10月22日
返信先: @shugomaedaさん
じゃあ **refinements** は消しときますね (•'◡'•)

_ko1 @_ko1 · 2021年4月5日
いっけね、**refinements** について完全に無視していた

_ko1 @_ko1 · 2020年1月9日
メソッド定義周りの魔窟を触り始めて、また **refinements** 死すべしという思いを新たにしている 難しすぎる

_ko1 @_ko1 · 2016年6月1日
block->proc 使っている **refinements** というやつが居た消したい

_ko1 @_ko1 · 2021年3月25日
返信先: @shugomaedaさん、@joker1007さん、他2人
refinement と一緒に for も消し去りましょう

_ko1 @_ko1 · 2016年9月23日
返信先: @shugomaedaさん
refinement だ、殺せ！

_ko1 @_ko1 · 2017年9月25日
返信先: @shugomaedaさん
その悲しみも **refinement** とともに洗い流してしまいましょう。

_ko1 @_ko1 · 2020年2月13日
また **refinement** が邪魔をしているぞ

_ko1 @_ko1 · 2016年6月12日
refinement はぶっ壊していると思うので消したい

_ko1 @_ko1 · 2017年9月25日
refinement を消せば解決だよ！

_ko1 @_ko1 · 2015年11月18日
返信先: @shugomaedaさん
@shugomaeda **refinement** を迫害したいので、こっそりキャッシュしないようにしておけばみんな幸せ

Refinementsこそ
戦乱の火種🔥

そもそもRefinementsとは？

こんなモンキーパッチは影響範囲が広すぎる

```
module Kernel
  def puts(*args)
    $stdout.puts(*args.map{"#{it}でござる!"})
  end
end
puts "Hey!", "You!"
#=>
# Hey!でござる!
# You!でござる!
```

モンキーパッチを局所化する

```ruby
module Gozaru
  refine Kernel do
    def puts(*args)
      super(*args.map{"#{it}でござる!"})
    end
  end
end
using Gozaru
puts "Hey!", "You!"
#=>
# Hey!でござる!
# You!でござる!
```

Refinementsの兵具

- `Module#refine(klass) { } -> Module`
  - ブロックが表現する機能を現在のモジュールに追加（拡張）
- `Module#using(module) -> self`
  - 拡張モジュールを現在のクラス、モジュールで有効にする
- `class Refinement`
  - refine ブロックの中の `self` のクラス。つまり黒魔術

Refineされる範囲

using以降に有効 そして 別のファイルには無効

```
puts "Hey!", "You!"
using Gozaru
puts "Hey!", "You!"
#=>
# Hey!
# You!
# Hey!でござる
# You!でござる
```

別のファイルには無効🤔

RubyとはBasicObjectから乾坤開闢するもの

```ruby
class Object < BasicObject
  include Kernel
end

Object.new #=> オブジェクト！
```

- ファイルスコープ？なにそれ？
  - ローカル変数とかいう田舎侍の所領を安堵するための不承不承の和睦では？（暴言）

「見える範囲」みたいな足軽ぽいやつじゃなくて

もっと忍者っぽいのがいい

ブロックの中で暗躍するみたいな？

Rubyの命はブロック！
（唐突ですが真理）

橘羽角前下総守情操指南匠均之助
@hasumikin

ブーストする

やっぱりブロック内のRefinementsが要るよなあ

午前11:24 · 2025年11月17日 · **474** 件の表示

返信をポスト

返信

Akira Matsuda @a_matsuda · 2025年11月17日

要りますねー

256


橘羽角前下総守情操指南匠均之助
@hasumikin
ブーストする
ブロックの中だけRefineしたいじゃんねー
午前9:39 · 2022年11月28日
返信をポスト
返信
Akira Matsuda @a_matsuda · 2022年11月28日
したいっすねー
Koichi ITO @koic · 2022年11月28日
ちょうどそれ思っていたところでした！

道半ばにて潰えし夢

- るびま「Refinementsとは何だったのか」
  - https://magazine.rubyist.net/articles/0041/0041-200Special-refinement.html
- amatsuda/activerecord-refinements

```
User.where { :name == 'matz' }
```

覇業なお一縷の望みを留む

Feature: `Proc#using`

https://bugs.ruby-lang.org/issues/16461 👍押してくれ


```ruby
module IntegerDivExt
  refine Integer do
    def /(other)
      quo(other)
    end
  end
end

def instance_eval_with_integer_div_ext(obj, &block)
  block.using(IntegerDivExt)
  obj.instance_eval(&block)
end

using Proc::Refinements

p 1 / 2 #=> 0
instance_eval_with_integer_div_ext(1) do
  p self / 2 #=> (1/2)
end
p 1 / 2 #=> 0
```

人間五十年
大望を遂ぐるに足らず

我PicoRubyのTask#usingを所望す

- PicoRubyのTaskクラスとは
  - “プリエンプティブ”なタスクスケジューラ
    - 外形上はCRubyのThreadクラスに似ています（後述）
    - スケジューラが強制的にタスク制御を移す（yieldする）
    - Fiberが「自分もういいんで他の方どうぞ」と内発的にyieldするのと対照的
- これをRefineしたい！したいしたい！やるぞ！
  - 自分で好き勝手できるRuby処理系があるのって最高！
    おすすめ

畢竟斯くの如し

```
task = Task.new do
  using Gozaru
  puts "Refineしたタスク"
end
puts "別のタスク"
task.join
#=>
# 別のタスク
# Refineしたタスクでござる!
```

そも、何ゆえ欲する

- パイプラインってのがありますね

```
$> cat slide.md | head -1
# PicoRubyに於ける
```

- パイプとは
  - OSが提供するプロセス間通信
  - pipe(2)システムコールが無名パイプをつくる
  - 標準ストリームをつなぎかえる

そも、何ゆえ欲する

- R2P2でもパイプやりたい！でもOSがない！
- RubyはUNIXのラッパー（極論）だけど、PicoRubyはベアメタル

…ここでちょっと脱  します…

追憶…二〇一四年夏（たぶん）

- 我、出雲国へ国替えして間もなき頃
  - まだRuby初心者でした
- 「高度Ruby人材養成講座」たしかそんな名前
- テーマは「Thread」
- 講師は世界のshugomaeda!!!（とnari3）
  - この人に一生ついて行こうと思った

CRubyのThreadなる勇み足（個人の感想）

- Ruby 1.8までのThreadはグリーンスレッド
  - PicoRubyのTaskと似たやつ。じつはよく知らんけど
- Ruby 1.9あたりでpthread（OSネイティブスレッド）のラッパーになった
  - 名馬なれど御し難し
- JavaScriptがシングルスレッド上のイベントループであれだけの性能を叩き出しているのに……

CRubyのThreadなる勇み足（個人の感想）

- この禍根を晴らすのがFiber::Schedulerなのかな？
  - てきとうに言ってます
- て言うかCRubyチャン、Threadはなかったことにして PicoRubyからTaskを逆輸入しませんか？
  - ……いや冗談です我々にはRactorがある

我が流儀にて志を継ぐ

- PicoRuby.WASMもTaskスケジューラで動きます
- コールバックの通信に `Task::Queue` を活用
- …この続きはKaigi on  にて（ほんとに？）

`Task::Queue` is 何？

閑話休題

TaskローカルRefinementsの兵具揃え

- `Task` ... 去年mruby/mrubyにマージ
  - (´-`).。oO（CRubyに存在しないトップレベルなコアクラスがmrubyにマージされるの珍しくない？）
- `Task::Queue` ... つい最近mruby/mrubyにマージ
- task-refinementsの実装 … 我がPCのローカルに

Task::Queue は Thread::Queue の鏡

```
q = Task::Queue.new # 最初は空っぽ
task = Task.new do
  puts q.pop # popできるまでタスクがsuspend
end
puts "沙汰を待て"
# この間いろいろ忙しい
q.push "蟄居申し渡す"
task.join
#=>
# 沙汰を待て
# 蟄居申し渡す
```

例: R2P2のcatコマンド実装（簡略版）

※R2P2はマイコンで動くシェルシステムです。もちろんPure PicoRuby

```ruby
if ARGV.empty?
  while line = gets          # 標準入力から読む
    print line               # 標準出力へ書く
  end
else
  ARGV.each |filename|
    if File.file?(filename)
      File.open(filename, 'r') do |f|
        while line = f.gets # ファイルから読む
          print line         # 標準出力へ書く
        end
      end
    else
      $stderr.puts "#{filename}: No such file"
    end
  end
end
```

此度の策の肝要

- R2P2コマンド実装では、単に `puts` と書きたい
  - パイプされるかどうかはコマンドにはわからない
  - CRubyでもそのまま動くと開発が楽だし
- R2P2コマンドは、すでに個別のTaskなんです
- タスク内で `Kernel#puts` をRefineすればいい
- パイプIO通信にTask::Queueを使うとよさそう

演武

技の仔細：Core構造体

```
/* include/mruby.h */
struct mrb_context {
  ...
  struct mrb_refinement_chain *refinements; /* NULL until first #using */
};

struct mrb_cache_entry {
  ...
  struct mrb_context *ctx;
  ...
}

struct mrb_state {
  ...
  mrb_refinement_lookup_fn refinement_lookup; /* NULL until gem init */
};

/* Returns non-zero if a refined method was found for (c, mid). */
typedef int (*mrb_refinement_lookup_fn)(struct mrb_state *mrb, struct RClass *c,
                                        mrb_sym mid, struct RClass **cp,
                                        mrb_method_t *m);
```

技の仔細：Refinement構造体

```c
/* mrbgems/mruby-task/include/mruby/refinements.h */
struct mrb_refinement {
  struct RClass *target_class; /* class being refined          */
  struct RClass *methods;      /* RClass flagged REFINEMENT    */
  struct RClass *owner;        /* module that called #refine   */
};

/*
 * Per-task singly-linked list of active refinements.
 * Nodes are owned by the gem; the mrb_refinement pointers are NOT owned
 * (they stay alive via the owner module's instance variable).
 */
struct mrb_refinement_chain {
  struct mrb_refinement *ref;
  struct mrb_refinement_chain *next;
};
```

技の仔細：メソッド探索

```
/* src/class.c */
mrb_method_t
mrb_vm_find_method(mrb_state *mrb, struct RClass *c,
                   struct RClass **cp, mrb_sym mid)
{
  // メソッドキャッシュをごにょごにょ

  // Refineされているならmrb_refinement_chainをたどって
  // メソッドを探す
}
```

技の仔細：標準入出力をRefine

```
module PipeOUT # 別途PipeINもあります
  refine Kernel do
    def puts(*args)
      PipelineIO.puts(Task.current, *args)
      #                ^^^^^^^^^^^^ 呼び出し元コンテキスト
    end

    def print(*args)
      PipelineIO.print(Task.current, *args)
      #                 ^^^^^^^^^^^^ 呼び出し元コンテキスト
    end
  end
end
```

技の仔細：パイプライン実装

```
class PipelineIO
  # タスクごとにTask::Queueを開く（outboxとinbox）

  # 例: $> cat slide.md | head -1
  #     catがproducerで、headがconsumerというパタン
  #     catのoutboxとheadのinboxがじつは「同じもの」
  #     catがoutbox.pushすると、headがinbox.popで受け取る

  # Closedキューにpushするとと Task::Error 👇 ここがかっこいい
  # * `head -1`は1行読んだら店仕舞したい。つまり先にconsumerが終了
  # * inbox.closeしちゃう。そこへのpushはTask::Errorになる
  # * これをSIGPIPEとして扱い、producer（catコマンド）も早期終了
  # *  （タスクスイッチのタイミング次第で即時終了できないこともある）
end
```

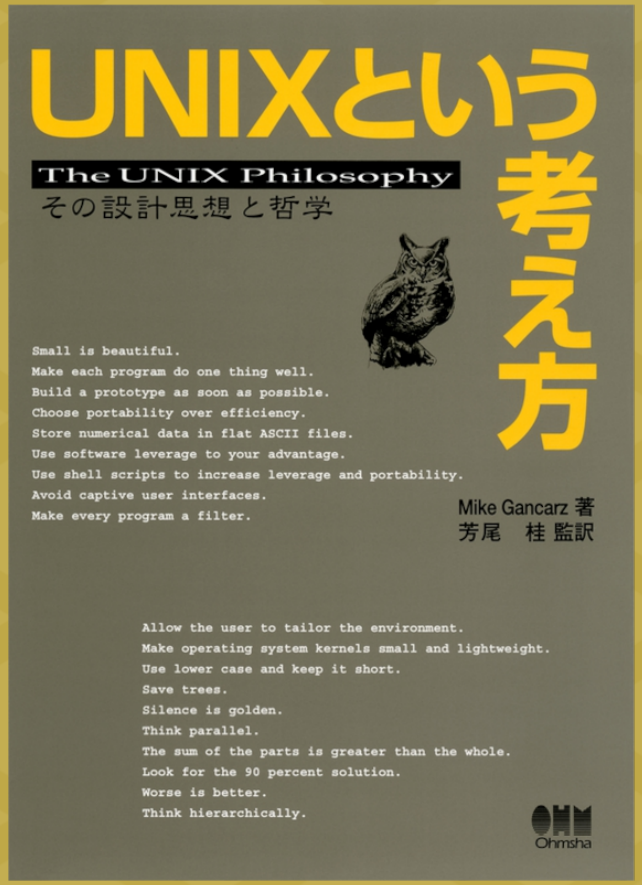
勘所

- `cat`コマンド、`head`コマンドは「標準入出力」に読み書きしているだけ
- R2P2が差配するQueueベースのパイプラインが標準入出力をこっそり差し替えている
- 以上のことがRefinementsによって実現されており、コマンドからはそのことは見えない

総じて左様

- UNIX風儀に外郭被せたるCRuby（個人の感想）
- 然る一方PicoRubyの足元甚だ空虚
- PicoRubyでもパイプしたいじゃん！導管！同感！
- 荒馬たるThreadを馴馬たるTaskへと鞍替えし
- 神代の遺物Refinements（暴言）に松明を掲げ
- 理を転じてPicoRubyにUNIX風情パイプを普請せり

参照兵書


UNIXという考え方
The UNIX Philosophy
その設計思想と哲学
Small is beautiful.
Make each program do one thing well.
Build a prototype as soon as possible.
Choose portability over efficiency.
Store numerical data in flat ASCII files.
Use software leverage to your advantage.
Use shell scripts to increase leverage and portability.
Avoid captive user interfaces.
Make every program a filter.
Mike Gancarz 著
芳尾　桂 監訳
Allow the user to tailor the environment.
Make operating system kernels small and lightweight.
Use lower case and keep it short.
Save trees.
Silence is golden.
Think parallel.
The sum of the parts is greater than the whole.
Look for the 90 percent solution.
Worse is better.
Think hierarchically.
Ohmsha


WORKING WITH UNIX PROCESSES
なるほど
UNIX
プロセス
Rubyで学ぶUnixの基礎
JESSE STORIMER　著
島田浩二・角谷信太郎　訳

宴席にて各々方に問うべき儀あり

- 「Proc#usingを推進する方策」とか
- 「貴殿等の考える最強のRefinements」とか
  - 来週の松江Ruby会議12でジョーカーさんがいい話します
- 「タスクローカルRefinementsをmrubyにマージするためにまつもとさんを調略する計事」とか
- だがその前に……

いざ合戦