

白魔術師への道

Road to white mages




Tama Ruby会議01

2019年7月6日

五十嵐邦明 / igaiga

twitter: @igaiga555


自己紹介

- フリーランスRails/Rubyエンジニア
- 著書
 -  Ruby超入門
 -  Railsの教科書
 -  RubyとRailsの学習ガイド

今回の方針

- 依頼 「基調講演ですがテクい話でお願いします」
- 本会議のテーマは「成長」
- 興味あることを調べて発表したいと思った
- 私も3ヶ月前には知らなかったことばかり話します
- 📖 Ruby超入門など著書の話はあまりしません
 - 代わりに書籍で説明した内容に 📖 マーク付けました

前置き

- 「後世の歴史家」と「最前線の勇者」の語る話は違う
-  Ruby超入門は「後世の歴史家」
 - 道が定まり、経路を新たに設計して説明
- 今日の話は「最前線の勇者」
 - どの道が良いかを模索する
 - 調べ方や悩んだタイミングもお伝えします

プロローグ

魔術とは何か？

matz 「魔術には2種類ある。良いことをする白魔術と、嫌なことをする黒魔術だ。」

- 書籍「メタプログラミングRuby」
- 魔術とは
 - メタプログラミングの技法
- メタプログラミングとは
 - コードを記述するコードを記述すること
 - 言語要素を実行時に操作するコードを記述すること

昨今のRuby界での魔術の定義

「（一見無理な）やりたいことを、手段を問わず実現すること」

※私の意見です

Ruby界の偉大な黒魔術師2人



joker1007

@joker1007



Repro inc. CTO Ruby/Rails Programmer



Satoshi "moris" Tagomori

@tagomoris



OSS developer/maintainer: Fluentd, Norikra, MessagePack-Ruby, Woothee and many others mainly about Web services, data collecting and distributed/streaming data processing. Living in Tokyo, and day job is for Treasure Data.

JA

Hijacking Ruby Syntax in Ruby

This talk shows how to introduce new syntax-ish stuffs using meta programming techniques and some more Ruby features not known well by many Rubyists. Have fun with magical code!

- Show Ruby features to hack Ruby syntax (including Binding, TracePoint, Refinements, etc)
- Describe stuffs introduced by these techniques
 - method modifiers (final, abstract, override)
 - Table-like syntax for testing DSL
 - Safe resource allocation/collection (with, defer)
- Propose new traceable events, hooks, etc

Presentation Material

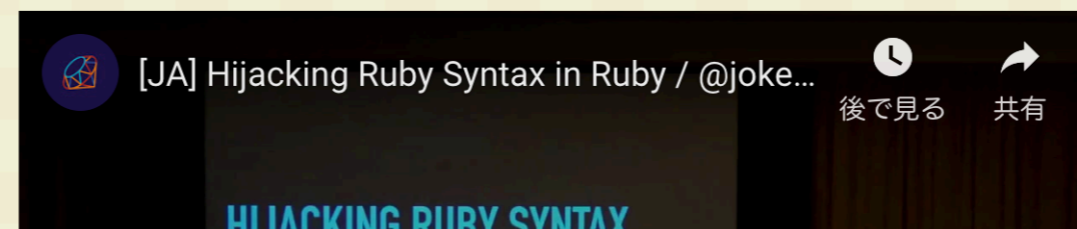
[Hijacking Ruby Syntax in Ruby](#)

joker1007, Satoshi "moris" Tagomori tagomoris

Recorded video

<https://youtu.be/04HGOEw3A6Y>

RubyKaigi2018



偉大になると
コミッターに目を付けられる



Shugo Maeda
@shugomaeda

|>が嬉しいかは別にして、ジョーカーさんがこう言っているのでもし入ってもとりあえず害はなさそう



ジョーカー 参戦!! @joker1007 · 22h

Rubyのパイプラインオペレーター、イマイチ嬉しさが無い。メソッドの部分適用が欲しい。

魔術で使われがちな道具

たとえば TracePoint (組み込みライブラリ)

何らかのコード実行をトリガーに処理を挟む機能

TracePointを使おうとするとみんなに心配される

 ジョーカー 参戦!!
@joker1007

自分達で発表しておきながら何だけど、あの発表を必要に迫られて見る状況は、ちょっと心配になるなw

 igaiga @igaiga555 · 4月25日

RubyKaigi2018の @joker1007 & @tagomoris "Hijacking Ruby Syntax in Ruby" を必要に迫られて見返したのだけど、やっぱりこれめっちゃ楽しいなあ。偉大な黒魔道士は仲間の体を傷つけないための魔法 (Refinement) も知っているって感じでかっこいい。勉強になる。
youtube.com/watch?v=04HGQE...

 tagomoris
@tagomoris

“必要に迫られて”

 igaiga @igaiga555 · 4月25日

RubyKaigi2018の @joker1007 & @tagomoris "Hijacking Ruby Syntax in Ruby" を必要に迫られて見返したのだけど、やっぱりこれめっちゃ楽しいなあ。偉大な黒魔道士は仲間の体を傷つけないための魔法 (Refinement) も知っているって感じでかっこいい。勉強になる。
youtube.com/watch?v=04HGQE...

 みょうが
@mrkn

返信先: @igaiga555 @joker1007 さん、 @tagomorisさん
大丈夫ですか？

 Takafumi ONAKA
@onk

返信先: @igaiga555さん
必要に迫られて.....?(´Д`;)ゴク

 ぺこちゃん
@ryopeko

TracePoint 使う人、もう治安荒らしに来てるか真面目に使ってるか極端な例しか見ていない

なぜTracePointを使うとあぶないのか？

初歩から説明される機会はあまりない

なぜRubyKaigi LTで「TracePoint Bomb!」というタイトルが出ただけでみんな爆笑するのか

黒魔術は強力だが意図しない動作も引き起こす可能性がある

RubyKaigi 2019
Fukuoka International Congress Center
2019.04.19 (Fri)

The TracePoint Bomb!

Bug of the year 2018

Koichi ITO / ESM, Inc.

Koichi ITO April 19, 2019 Programming ☆ 0 👁 660 ↓

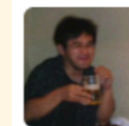
- TracePointはローカル変数を上書きできる
- この上書きを戻し忘れるルートが9ヶ月後判明
- 想定しない広い範囲でローカル変数が上書き
- そのバグが発見されたときの限界の様子



_ko1
@_ko1

```
21 def self.tmpdir_test
22   tmp = 1
23   tmp
24 end
```

このメソッドが nil 返すとか Ruby 面白すぎる



nobu 🍷 11:31 PM

バグ・オブ・ザ・イヤー

<https://speakerdeck.com/koic/the-tracepoint-bumb>

偉大な黒魔術師が偉大な理由

偉大な理由は仲間を守る魔法も使えるから

黒魔術は時として強大すぎる力を制御できなくなる
熟練の黒魔術師たちは魔術を制御する力に長けている

†Ruby黒魔術経典†

<https://speakerdeck.com/joker1007/rubyhei-mo-shu-jing-dian>

黒魔術において守るべきルール

- 組み込みクラスを壊さない
- スタックを追えるようにする
- パフォーマンスを意識する
- TracePointの利用は明示的に

黒魔術は適切につかわなければならない
実は制御する方が難しいことも多い
しかし強大な力は魅力的である

白魔術と黒魔術

デバッグに使う魔術を本発表では白魔術と呼ぶ

まずは魔術をproductionコードに入れな
い形で使ってみるのはどうか

productionコードに入れなければ危ないコードも気軽に書ける
たとえばデバッグ

デバッグに使う魔術を本発表では白魔術と呼ぶことにします

黒魔術（黒魔法）：プログラムで価値を獲得するための魔術

白魔術（白魔法）：プログラムの傷を癒やす魔術

今日の構成

- 白魔術基礎講座 「オブジェクトと会話する」
- 白魔術入門 「盗まれたインスタンス変数」 編
- 白魔術概論 その他の白魔術

白魔術基礎講座


「オブジェクトと会話する」

- Rubyのオブジェクトは色々なことを教えてくれる
- Ruby世界のオブジェクトたちの気持ちを知る
- オブジェクトたちを説得して状態を変えてもらおう

デバツガの使い方

プログラムを実行途中で一時停止してプログラムを実行する

- **binding.irb**  Ruby超入門 P.62

- **pry gem**  Ruby超入門 P.241 **ifでON/OFFすると楽**

- **binding.pry**
- **step 実行 pry-byebug gem**
- **ソース表示 pry-doc gem**

```
if 条件
  require "pry"; binding.pry
end
```

環境変数でON/OFFすると楽

```
if ENV["prying"]
  require "pry"; binding.pry
end
```

- **pry on**
 - **\$ prying=true rspec spec/a_spec.rb**
- **pry off**
 - **\$ rspec spec/a_spec.rb**

あなたは誰ですか？

オブジェクトを知る

- **class** メソッド  **Ruby超入門 P.178**
 - オブジェクトのクラスを知る
- **object_id** メソッド  **Ruby超入門 P.128, 269**
 - オブジェクトの識別番号を知る



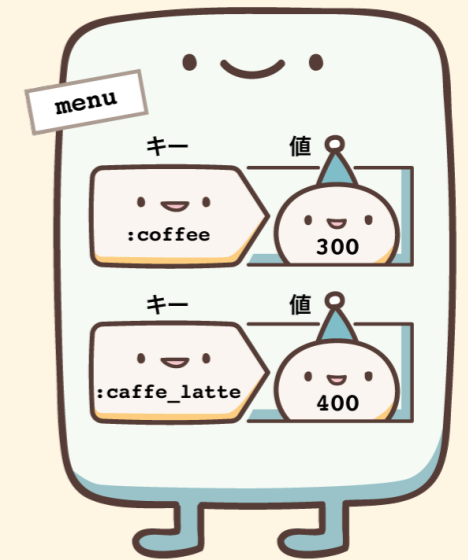
Integer



Float



String



**{coffee: 300,
caffe_latte: 400}**

Hash

どんなオブジェクトがありますか？

オブジェクト界を扱うObjectSpace

- **ObjectSpace**
 - オブジェクト界を扱う組み込みライブラリのモジュール
- **ObjectSpace.each_object(klass)**
 - 引数に指定したクラスのインスタンスオブジェクトを得る
 - クラス一覧を取得: **ObjectSpace.each_object(Class)**
- **ObjectSpace.count_objects**
 - オブジェクトを種類ごとにカウント

どんなメソッドが使えますか？

呼び出せるメソッドを知る

- **methods** メソッド  **Ruby超入門 P.189**
 - オブジェクトが呼び出せるメソッドを知る
- **Array.methods**
 - クラスメソッドを調べる
- **{}.methods**
 - インスタンスメソッドを調べる

あなたはどこから来ましたか？

ソースコード編

- **Object#method, Module#instance_method**
 - Methodオブジェクト取得
- Methodオブジェクトで使えるメソッド
 - **source_location**: 定義されているファイル
 - **owner**: 定義されているクラス
 - **parameters**: 引数情報

method(:pp).source_location

```
#=> ["/Users/igaiga/.rbenv/versions/2.6.2/lib/ruby/2.6.0/pp.rb", 582]
```

method(:pp).owner

```
#=> Kernel
```

```
require "time"
```

Time.instance_method(:httpdate).source_location

```
#=> ["/Users/igaiga/.rbenv/versions/2.6.2/lib/ruby/2.6.0/time.rb", 691]
```

Time.instance_method(:succ).source_location

```
#=> nil # Rubyで書かれていないネイティブなメソッドはnil
```

あなたはどこから来ましたか？

gem編

- gemソースコードが置いてある場所
 - **gem which**
 - **bundle show**
 - **bundle open**
 - 指定されたエディタで開く
 - 環境変数 **EDITOR** or **BUNDLER_EDITOR** で指定


あなたはどこから来ましたか？

呼び出し元編

- **Kernel#caller**
 - メソッド呼び出し履歴を表示
- **Kernel#caller_locations**
 - callerの結果を Thread::Backtrace::Location で取得
- **Exception#backtrace**
 - そのExceptionがraiseされたときのメソッド呼び出し履歴を表示

オブジェクトを変更する

だいたいのこと是可以る

- **instance_variables**  Ruby超入門 P.198
 - インスタンス変数一覧を取得
- **instance_variable_get, instance_variable_set**
 - インスタンス変数を取得、改ざん
- **instance_eval, class_eval**
 - そのオブジェクトになりきって実行する
 - インスタンス変数をいじるときにも便利
 - privateメソッドを呼び出すときにも便利
 - sendメソッドを使う手もある
- **binding**を使うとローカル変数を変更できる
 - 前述の **joker&moris** 講演参照

ソースコードの世界、オブジェクトの世界

デバッグが難しいのは2つの世界に隔たりがあるから

- 演劇で例えると

- ソースコードは**台本**

- 私たちが書けるのは台本

- オブジェクトの世界は**舞台**

- 実際にプログラムが動いているのはここ

- 私たちはここで自分が演じることも、観ることもできない

オブジェクトの世界 →

ソースコードの世界 →



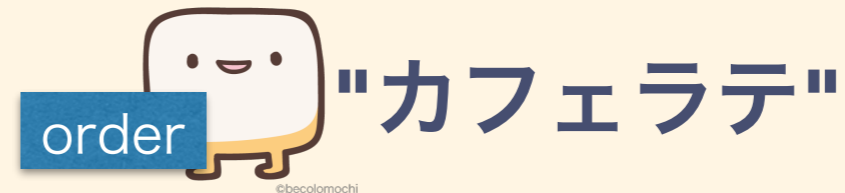
p 1+2

ソースコードの世界、オブジェクトの世界

変数はオブジェクトの名札🔥、名前重要 📖 Ruby超入門 P.53

「名前」はソースコードの世界からオブジェクトの世界にいる「そのオブジェクト」を呼ぶ（ほぼ）唯一の仕組み

オブジェクトの世界 ➡



ソースコードの世界 ➡

```
order.rb  
order = "カフェラテ"  
puts order
```

変数orderに "カフェラテ" オブジェクトを代入
= "カフェラテ" オブジェクトに
変数orderという名前を付ける

様々な手段でオブジェクトの世界（舞台）の様子を知る方法をこのあと説明していきます。

白魔術基礎講座 「オブジェクトと会話する」 まとめ

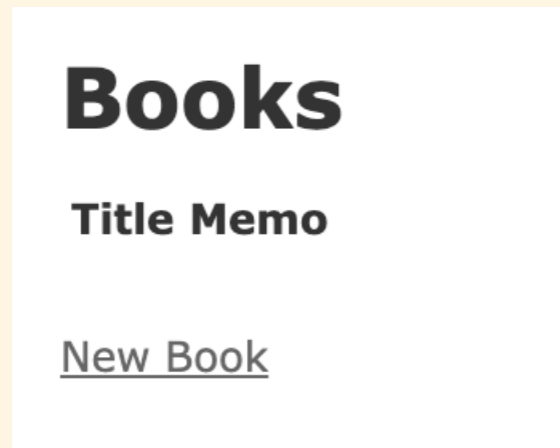
- オブジェクトとの会話によってさまざまな情報が得られる
- オブジェクトの状態を変更することもできる
- オブジェクトの世界、ソースコードの世界には隔たりがある

白魔術入門

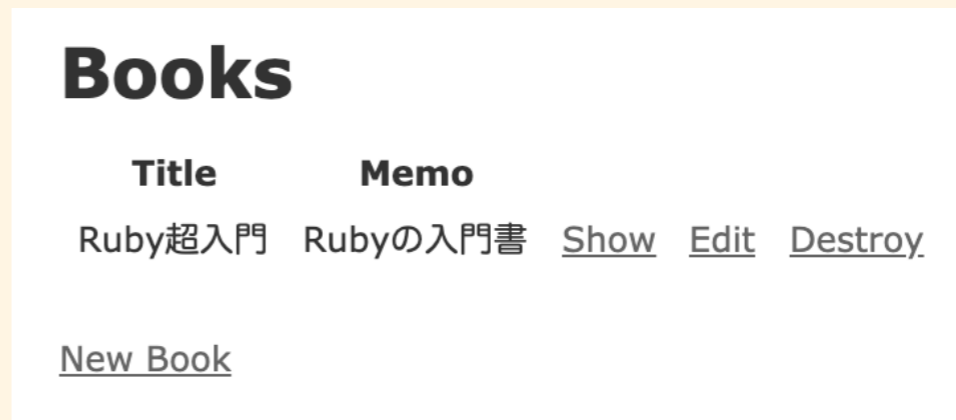
「盗まれたインスタンス変数」 編

「やつは大変なものを盗んでいきました。
あなたのインスタンス変数です。」

現状の動作



正しい動作



題材：「RailsのCRUD indexアクションで、@books インスタンス変数が [] になっていて表示内容が意図通りではない」

「インスタンス変数への代入を検知する」方法を考える

- ちなみにグローバル変数は Kernel#trace_var がある
- インスタンス変数はオブジェクトではない
- コード: <https://github.com/igaiga/tmrk01/>

TracePoint

何かをきっかけに処理を挟む

実行するブロックを登録しておくと特定トリガーでブロックを実行してくれる

主なトリガー

- **:line** 式の評価
- **:call** Rubyメソッドの実行
- **:c_call** Cメソッドの実行
- **:b_call** ブロック実行
- **:raise** 例外発生

tp.rb

```
class Foo
  def foo
    p "foo!"
  end
end
```

```
TracePoint.trace(:call) do |tp|
  # メソッドが呼び出されるごとにこのブロックが呼び出される
  p tp
end
```

```
Foo.new.foo
#=> #<TracePoint:call `foo'@tp.rb:2>
#=> "foo!"
```

↑ p tpの結果

<https://docs.ruby-lang.org/ja/latest/class/TracePoint.html>

インスタンス変数への代入を検知する

準備

最初に小さなコードから考えます。

ターゲットになるコード 

@hi への代入を検知するのが目標です。

※ただしgrepは考慮しないものとする。

 rb

```
class User
  def hi
    @hi = "hi(。・ω・。)ノ"
  end  @hiへの代入を検知したい
end

p User.new.hi #=> "hi(。・ω・。)ノ"
```

TracePoint

TracePointをしかけて全ての式評価に処理を挟む


TracePoint.trace に :line を渡すと、式評価時にブロック実行
ブロックの変数tpで受け取るTracePointオブジェクトから情報取得
tp.methods やるりまで持っているメソッドを調べる  P.123


- event
 - トリガー
- path
 - ソースパス
- lineno
 - 行番号
- method_id
 - メソッド
- defined_class
 - クラス


```
rb
class User
  def hi
    @hi = "hi(｡・ω・｡)ﾉ" # 3行目
  end
end


def tracer
  TracePoint.trace(:line) do |tp|
    puts "[TP:#{tp.event}] #{tp.path}:#{tp.lineno} #{tp.method_id} #{tp.defined_class}"
  end
end

tracer
p User.new.hi # 14行目
#=> [TP:line] test.rb:14
#=> [TP:line] test.rb:3 hi User
#=> "hi(｡・ω・｡)ﾉ"
```

 @hi へ代入 3行目 User#hi

 |tp| TracePointオブジェクト

 TracePointオブジェクトの情報表示

 [TP:line] test.rb:3 hi User

TracePoint

全ての式評価の中から目的の行を探し出す

式評価時にブロック実行はできた

今のままだと全ての式でブロックが実行される

その中からインスタンス変数代入を検知できないか？

ここで
1週間悩んだ

あとで笹田さん
にも聞いた

```
rb

class User
  def hi
    @hi = "hi(｡・ω｡)ノ" # 3行目
  end
end

def tracer
  TracePoint.trace(:line) do |tp|
    puts "[TP:#{tp.event}] #{tp.path}:#{tp.lineno} #{tp.method_id} #{tp.defined_class}"
  end
end

tracer
p User.new.hi # 14行目
#=> [TP:line] test.rb:14
#=> [TP:line] test.rb:3 hi User
#=> "hi(｡・ω｡)ノ"
```

次のコードからUserクラスは省略



TracePoint

式の元になったソースコードを得る

TracePointオブジェクトにはソースコード情報がない

評価された式の `path`, `lineno` は分かる

➡ `path`, `lineno` からソースコードを取れそう

```
def tracer
  TracePoint.trace(:line) do |tp|
    puts "[TP:#{tp.event}] #{tp.path}:#{tp.lineno} #{tp.method_id} #{tp.defined_class}"
    p line = File.open(tp.path, "r"){|f| f.readlines[tp.lineno - 1]}
  end
end
```

↑ ソースファイルからソースコード取得

```
tracer
p User.new.hi
#=> [TP:line] test.rb:15
#=> "p User.new.hi\n"
#=> [TP:line] test.rb:3 hi User
#=> "    @hi = \"hi(｡・ω・｡)ノ\"\n"
#=> "hi(｡・ω・｡)ノ"
```

" @hi = \"hi(｡・ω・｡)ノ\"\n"
↑ ソースコードトレタ

取得コード: " @hi = \"hi(｡・ω・｡)ノ\"\\n"

次はこのコードが何をしているのか知りたい

```
def tracer
  TracePoint.trace(:line) do |tp|
    puts "[TP:#{tp.event}] #{tp.path}:#{tp.lineno} #{tp.method_id} #{tp.defined_class}"
    p line = File.open(tp.path, "r"){|f| f.readlines[tp.lineno - 1] }
  end
end
```

```
tracer
p User.new.hi
#=> [TP:line] test.rb:15
#=> "p User.new.hi\\n"
#=> [TP:line] test.rb:3 hi User
#=> " @hi = \"hi(｡・ω・｡)ノ\"\\n"
#=> "hi(｡・ω・｡)ノ"
```

" @hi = \"hi(｡・ω・｡)ノ\"\\n"
ソースコードトレタ




RubyVM::AbstractSyntaxTree


コードからASTを得る

RubyVM::AbstractSyntaxTree.parseを使うとコードが何をするものなのかの情報を得られる

```
def tracer
  TracePoint.trace(:line) do |tp|
    puts "[TP:#{tp.event}] #{tp.path}:#{tp.lineno} #{tp.method_id} #{tp.defined_class}"
    line = File.open(tp.path, "r"){|f| f.readlines[tp.lineno - 1] }
    p node = RubyVM::AbstractSyntaxTree.parse(line)
    pp node
  end
end
```

 RubyVM::AbstractSyntaxTree.parse(" @hi = \"hi(｡•ω｡)ノ\"\\n")

```
tracer
User.new.hi
```



```
#=> [TP:line] 04_a #<RubyVM::AbstractSyntaxTree::Node:SCOPE@1:0-1:33>
#=> #<RubyVM::AbstractSyntaxTree::Node:SCOPE@1:0-1:33>
#=> (SCOPE@1:0-1:33
#=>  tbl: []
#=>  args: nil
#=>  body: (IASGN@1:4-1:33 :@hi (STR@1:10-1:33 "hi(｡•ω｡)ノ")))
```

RubyVM::AbstractSyntaxTree

AbstractSyntaxTree(AST)って何？

Rubyのコードが実行されるまでの流れ

Ruby のコードが実行されるまで

RubyVM::AbstractSyntaxTree.parse

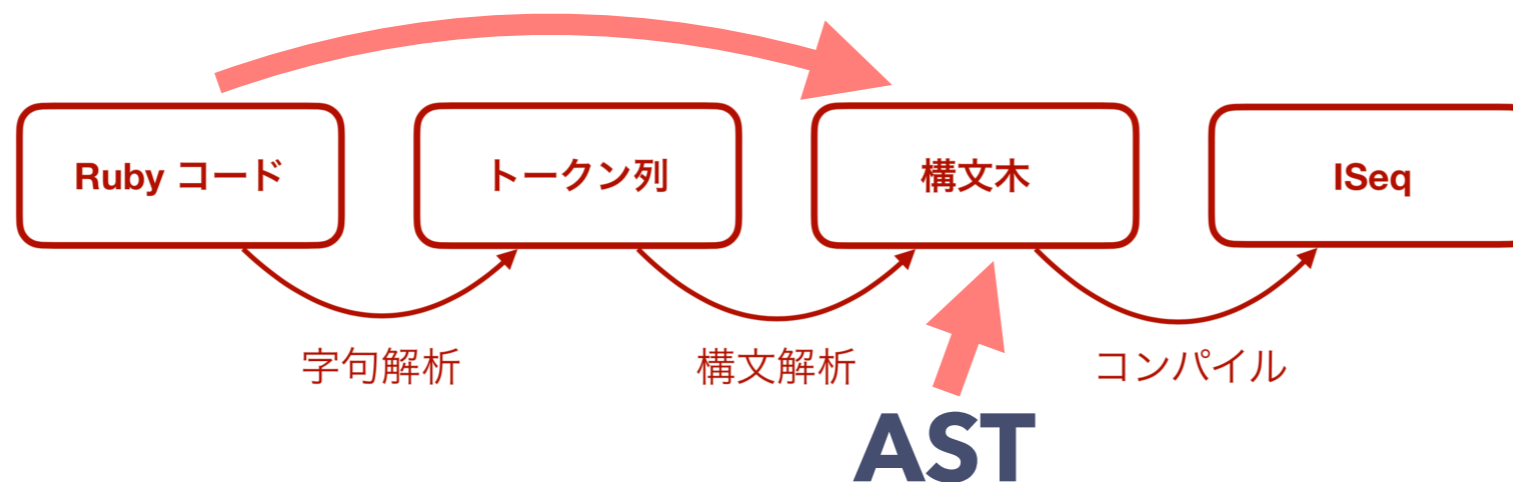


図1. Ruby が実行されるまでにどのような変換が行われるのか



©becolomochi

コードの世界

オブジェクトの世界

Rubyのコード解析の「静」と「動」 玉寄 修一

<https://speakerdeck.com/siman/railsdm2019?slide=6>

Rubyのしくみ <https://tatsu-zine.com/books/ruby-under-a-microscope-ja>

iSeq探訪 <https://booth.pm/ja/items/834594>

RubyVM::AbstractSyntaxTree

AST::Nodeオブジェクトの構造

RubyVM::AbstractSyntaxTree.parse(" @hi = \"hi(｡・ω・｡)ノ\"\\n")

```
def tracer
```

```
  TracePoint.trace(:line) do |tp|
```

```
    puts "[TP:#{tp.event}] #{tp.path}:#{tp.lineno} #{tp.method_id} #{tp.defined_class}"
```

```
    line = File.open(tp.path, "r"){|f| f.readlines[tp.lineno - 1]}
```

```
    p node = RubyVM::AbstractSyntaxTree.parse(line)
```

```
    pp node
```

```
  end
```

```
end
```

node.methodsで持っているメソッドを探して調べる  P.189

node #=> (SCOPE@1:0-1:33 tbl: [] args: nil

body: (IASGN@1:4-1:33 :@hi (STR@1:10-1:33 "hi(｡・ω・｡)ノ")))

node.class #=> RubyVM::AbstractSyntaxTree::Node

node.type #=> :SCOPE

node.children #=> [[], nil, (IASGN@1:4-1:33 :@hi (STR@1:10-1:33 "hi(｡・ω・｡)ノ"))]

node.children.last.class #=> RubyVM::AbstractSyntaxTree::Node

node.children.last.type #=> :IASGN

RubyVM::AbstractSyntaxTree::Nodeオブジェクトが積層

RubyVM::AbstractSyntaxTree

AST::Nodeオブジェクトの構造

RubyVM::AbstractSyntaxTree.parse(" @hi = \"hi(｡•ω•｡)ノ\"\\n")

```
def tracer
  TracePoint.trace(:line) do |tp|
    puts "[TP:#{tp.event}] #{tp.path}:#{tp.lineno} #{tp.method_id} #{tp.defined_class}"
    line = File.open(tp.path, "r"){|f| f.readlines[tp.lineno - 1] }
    p node = RubyVM::AbstractSyntaxTree.parse(line)
    pp node
  end
end
```

node.children.last #=> (IASGN@1:4-1:33 :@hi (STR@1:10-1:33 "hi(｡•ω•｡)ノ"))

node.children.last.type #=> :IASGN

IASGNをRubyのソースコードなどで調べてみる

IASGN #=> インスタンス変数への代入

RubyVM::AbstractSyntaxTree::Node 詳細 - siman

<https://qiita.com/siman/items/3017ed399ded43229189>

RubyVM::AbstractSyntaxTree

AST::Node情報からインスタンス変数代入を検知

```
def tracer
  TracePoint.trace(:line) do |tp|
    # ソース取得
    line = File.open(tp.path, "r"){|f| f.readlines[tp.lineno - 1] }
    # AST取得
    node = RubyVM::AbstractSyntaxTree.parse(line).children.last
    # インスタンス変数への代入かを調べる
    next unless node.type == :IASGN  next unless node.type == :IASGN
    puts "[TP:#{tp.event}] #{tp.path}:#{tp.lineno} #{tp.method_id} #{tp.defined_class}"
  end
end

tracer
User.new.hi  検知できた(´・ω・`)b
#=> [TP:line] 05 ast2.rb:3 hi User
```

現状は全てのインスタンス変数代入を検知している
次は指定したインスタンス変数代入だけを検知したい

TracePoint

指定クラスのインスタンス変数代入を検知

tp.self : イベントを発生させたオブジェクト

tp.self.is_a?(クラス) で指定クラスかどうかチェック

```
def tracer(target_class_name)
  TracePoint.trace(:line) do |tp|
    # ソース取得
    line = File.open(tp.path, "r"){|f| f.readlines[tp.lineno - 1] }
    # AST取得
    node = RubyVM::AbstractSyntaxTree.parse(line).children.last
    # インスタンス変数への代入かを調べる
    next unless node.type == :IASGN
    # クラス名を調べる
    target_class = Kernel.const_get(target_class_name)
    next unless tp.self.is_a?(target_class)
    puts "[TP:#{tp.event}] #{tp.path:#{tp.lineno} #{tp.method_id} #{tp.defined_class}"
  end
end
```

```
tracer("User")
User.new.hi
```

```
#=> [TP:line] 06_ast3.rb:3 hi User
```

Kernel.**const_get**(target_class_name)

"User" を Userクラスへ

is_a? メソッドでそのクラスか判定

※**tp.self**は「どのクラスで実行されている処理か」なので、includeしたModuleで代入しているケースも問題なく動く。クラス外でインスタンス変数代入（可能？）するとダメかも。

TracePoint

指定クラス、指定名のインスタンス変数代入を検知

インスタンス変数名を指定して検知

```
def tracer(target_class_name, target_instance_variable_name)
  TracePoint.trace(:line) do |tp|
    # ソース取得
    line = File.open(tp.path, "r"){|f| f.readlines[tp.lineno - 1] }
    # AST取得
    node = RubyVM::AbstractSyntaxTree.parse(line).children.last
    # インスタンス変数への代入かを調べる
    next unless node.type == :IASGN
    # クラス名を調べる
    target_class = Kernel.const_get(target_class_name)
    next unless tp.self.is_a?(target_class)
    # インスタンス変数名を調べる
    instance_variable_name = node.children.first
    next unless instance_variable_name == target_instance_variable_name.to_sym
    puts "[TP:#{tp.event}] #{tp.path}:#{tp.lineno} #{tp.method_id} #{tp.defined_class}"
  end
end
```

↓ AST nodeの中にインスタンス変数名がある

```
tracer("User", "@hi")
```

```
User.new.hi
```

```
#=> [TP:line] 07_target_name.rb:3 hi User
```

インスタンス変数代入検知

完成版

できた！(´・ω・`)b

trace_instance_variables.rb

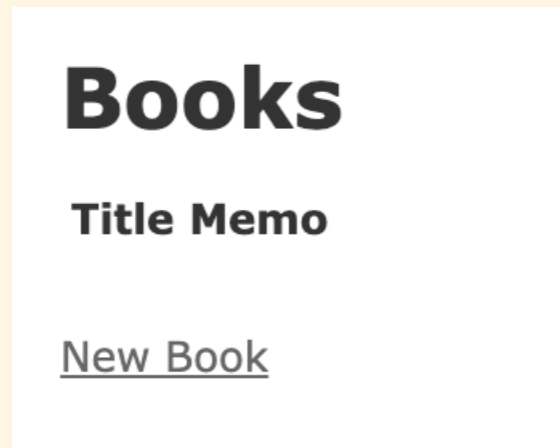
```
class User
  def hi
    @hi = "hi(｡・ω｡)ﾉ"
  end
end

def tracer(target_class_name, target_instance_variable_name)
  TracePoint.trace(:line) do |tp|
    # ソース取得
    begin
      line = File.open(tp.path, "r"){|f| f.readlines[tp.lineno - 1] }
    rescue Errno::ENOENT => e
    end
    next unless line
    # AST取得
    begin
      node = RubyVM::AbstractSyntaxTree.parse(line).children.last
    rescue Exception => e # 乱暴
    next
    end
    # インスタンス変数への代入かを調べる
    next unless node.type == :IASGN
    # クラス名を調べる
    target_class = Kernel.const_get(target_class_name)
    next unless tp.self.is_a?(target_class)
    # インスタンス変数名を調べる
    instance_variable_name = node.children.first
    next unless instance_variable_name == target_instance_variable_name.to_sym
    puts "#{target_class_name} #{@target_instance_variable_name} is assigned in #{tp.path}"
  end
end

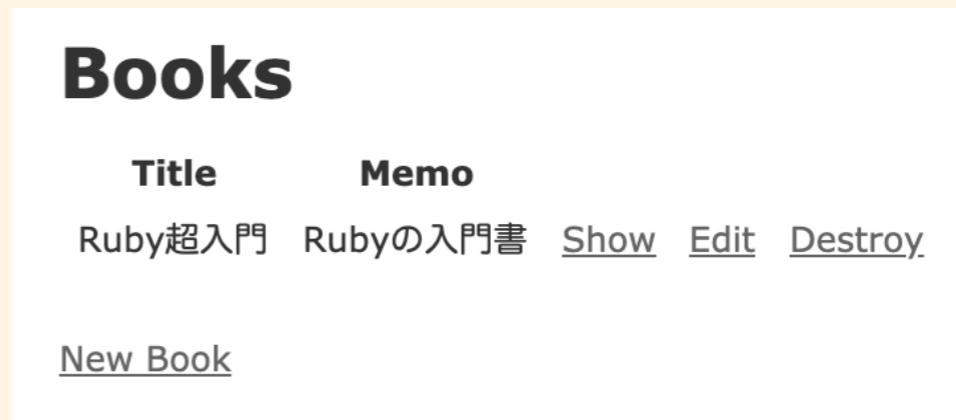
tracer("User", "@hi")
User.new.hi
#=> User @hi is assigned in trace_instance_variables.rb:4 hi User
```


「やつは大変なものを盗んでいきました。
あなたのインスタンス変数です。」

現状の動作






正しい動作



題材：「RailsのCRUD indexアクションで、@books インスタンス変数が [] になっていて表示内容が意図通りではない」

このアプリに対してインスタンス代入検知をしかけてみる

Railsアプリの準備

- 題材:  「Railsの教科書」で作っているアプリ
 - <https://tatsu-zine.com/books/rails-textbook>
 - 書籍管理アプリ
 - Bookモデル(title, memo)のCRUD
 - rails g scaffold book title:string memo:text
 - https://github.com/igaiga/tmrk01/missing_instance_variables_sample_rails_app
 - config/initializers 以下に検知コード配置 (次ページ説明)
 - /books へアクセス(indexアクション)
 - app/views/books/index.html.erb
 - <% @books.each do |book| %>  代入検知対象 @books
 - app/controllers/books_controller.rb:7 で代入されている
 - @books = Book.all  @books 代入コード

インスタンス変数代入検知

@books 代入検知コード

config/initializers へ検知コード配置

コントローラで代入されてる場合

全てのクラスでの実行を調べるため
クラス名は指定せずに
インスタンス変数名だけ指定

/books へアクセスしたログ

Started GET "/books" for ::1 at 2019-06-08 12:34:50 +0900
Processing by BooksController#index as HTML

@books is assigned in /path_to_app/app/controllers/books_controller.rb:7 index BooksController

Rendering books/index.html.erb within layouts/application

Book Load (6.8ms) SELECT "books".* FROM "books"

↳ app/views/books/index.html.erb:15

Rendered books/index.html.erb within layouts/application (249.2ms)

Completed 200 OK in 2933ms (Views: 2800.7ms | ActiveRecord: 6.8ms)

config/initializers/trace_instance_variables.rb

```
def tracer(target_instance_variable_name)
  TracePoint.trace(:line) do |tp|
    # ソース取得
    begin
      line = File.open(tp.path, "r"){|f| f.readlines[tp.lineno - 1] }
    rescue Errno::ENOENT => e
    end
    next unless line
    # AST取得
    begin
      node = RubyVM::AbstractSyntaxTree.parse(line).children.last
    rescue Exception => e # 乱暴
    next
    end
    # インスタンス変数への代入かを調べる
    next unless node.type == :IASGN
    # インスタンス変数名を調べる
    instance_variable_name = node.children.first
    next unless instance_variable_name == target_instance_variable_name.to_sym
    puts "#{target_instance_variable_name} is assigned in #{tp.path}:#{tp.lineno}"
  end
end

tracer("@books")
```

犯人がない素のコードでテスト

↓ コントローラでの正常代入検知成功

インスタンス変数代入検知

コントローラに犯人がいたケース

/books へアクセスしたログ

Started GET "/books" for ::1 at 2019-06-08 13:02:25 +0900

Processing by BooksController#index as HTML

@books is assigned in /path_to_app/app/controllers/books_controller.rb:7 index BooksController

@books is assigned in /path_to_app/app/controllers/books_controller.rb:8 index BooksController

Rendering books/index.html.erb within layouts/application

Rendered books/index.html.erb within layouts/application (50.2ms)

Completed 200 OK in 3146ms (Views: 2867.5ms | ActiveRecord: 0.0ms)

↑ 検知!!

books_controller.rb:7,8 での代入検知ができた

犯人確保!! 🚓

books_controller.rb

```
class BooksController < ApplicationController
  before_action :set_book, only: [:show, :edit, :update, :destroy]

  # GET /books
  # GET /books.json
  def index
    @books = Book.all
    @books = []
  end
end
```

← books_controller.rb:7

← books_controller.rb:8 🚓

Books


Title Memo

New Book

インスタンス変数代入検知

検知できないケースがある

/books へアクセスしたログ


Started GET "/books" for ::1 at 2019-06-08 12:55:09 +0900  コントローラでの代入は検知
Processing by BooksController#index as HTML
@books is assigned in /path_to_app/app/controllers/books_controller.rb:7 index BooksController
Rendering books/index.html.erb within layouts/application
Rendered books/index.html.erb within layouts/application (83.2ms)
Completed 200 OK in 2881ms (Views: 2742.1ms | ActiveRecord: 0.0ms)

books_controller.rb:7 で（正常な）代入検知はできている

犯人の検知は
できていない 🤔

books_controller.rb

```
class BooksController < ApplicationController
  before_action :set_book, only: [:show, :edit, :update, :destroy]

  # GET /books
  # GET /books.json
  def index
    @books = Book.all  books_controller.rb:7
  end
end
```

Books

Title Memo

New Book

インスタンス変数代入検知

この方法ではViewでの代入を検知できなかった

app/view/books/index.html.erb

<% @books = [] %>  犯人がViewにいるとき、この代入を検知できない

...

<% @books.each do |book| %>

原因

※見当違いしてて解決に1ヶ月かかった

config/initializers/trace_instance_variables.rb

```
def tracer(target_instance_variable_name)
  TracePoint.trace(:line) do |tp|
    # ソース取得
    begin
      line = File.open(tp.path, "r"){|f| f.readlines[tp.lineno - 1] }
    rescue Errno::ENOENT => e
    end
    next unless line
    # AST取得
    begin
      node = RubyVM::AbstractSyntaxTree.parse(line).children.last
    rescue Exception => e # 乱暴
      next
    end
  end
end
```

line #=> "<% @books= [] %>\n"
erbなので前後に <% %> が付いている


RubyVM::AbstractSyntaxTree.parse(line).children.last
#=> **SyntaxError**

インスタンス変数代入検知

erb対応

```
def tracer(target_instance_variable_name)
  TracePoint.trace(:line) do |tp|
    # ソース取得
    begin
      line = File.open(tp.path, "r"){|f| f.readlines[tp.lineno - 1] }
    rescue Errno::ENOENT => _e
    end
    next unless line
    # erb対応
    if match_data = line.match(/A\s*<%=*(.*)%>\s*\z/)
      line = match_data.captures.first
    end
    # AST取得
    begin
      node = RubyVM::AbstractSyntaxTree.parse(line).children.last
    rescue SyntaxError => _e
    next
    end
    # インスタンス変数への代入かを調べる
    next unless node.type == :IASGN
    # インスタンス変数名を調べる
    instance_variable_name = node.children.first
    next unless instance_variable_name == target_instance_variable_name.to_sym
    puts "#{target_instance_variable_name} is assigned in #{tp.path}:#{tp.lineno}"
  end
end

tracer("@books")
```

 Ruby超入門 P. 271 正規表現
前後に <% %> が付いていたら外す

 `line.match(/A\s*<%=*(.*)%>\s*\z/)`

 `line = match_data.captures.first`

正規表現中の(.*)部分取得

<https://docs.ruby-lang.org/ja/latest/class/MatchData.html>

AST取得より後ろは同じ

インスタンス変数代入検知

erb対応

```
def tracer(target_instance_variable_name)
  TracePoint.trace(:line) do |tp|
    # ソース取得
    begin
      line = File.open(tp.path, "r"){|f| f.readlines[tp.lineno - 1] }
    rescue Errno::ENOENT => _e
    end
    next unless line
    # erb対応
    if match_data = line.match(/A\s*<%=*(.*)%>\s*\z/)
      line = match_data.captures.first
    end
    # AST取得
  end
end
```

Started GET "/books" for ::1 at 2019-06-09 09:55:08 +0900

(12.4ms) SELECT "schema_migrations"."version" FROM "schema_migrations" ORDER BY "schema_migrations"."version" ASC

↳ /Users/igaiga/.rbenv/versions/2.6.3/lib/ruby/gems/2.6.0/gems/activerecord-5.2.3/lib/active_record/log_subscriber.rb:98

Processing by BooksController#index as HTML

@books is assigned in /path_to_app/app/controllers/books_controller.rb:7 index BooksController

Rendering books/index.html.erb within layouts/application

@books is assigned in /path_to_app/app/views/books/index.html.erb:1

_app_views_books_index_html_erb__4042701126060758541_70158372829520

ActionView::CompiledTemplates

Rendered books/index.html.erb within layouts/application (76.8ms)

Completed 200 OK in 13765ms (Views: 13457.4ms | ActiveRecord: 0.0ms)

犯人確保!! 🚓

📄 index.html.erb:1

<% @books = [] %>

白魔術入門 「盗まれたインスタンス変数」 まとめ

- インスタンス変数代入検知手順の一例
- `TracePoint :line` でフック
- `TracePoint`の情報からソースファイル取得
- `RubyVM::AbstractSyntaxTree.parse`で解析
 - インスタンス変数代入はIASGN
- ここで出てきたソースコード
 - <https://github.com/igaiga/tmrk01>
- `instance_variable_tracer` gem として公開
 - https://github.com/igaiga/instance_variable_tracer

白魔術概論

その他の白魔術

- Tracer
- ObjectSpace#allocation_sourcefile
- g
- signal & trap

Tracer標準添付ライブラリ

ログ出力 <https://docs.ruby-lang.org/ja/latest/class/Tracer.html>

中で `Kernel.#set_trace_func` を利用

全実行行ログ(ソースコード入り)をファイル出力してgrepで調べる

```
require 'tracer'  
fp = File.open('log.txt', "w")  
Tracer.stdout = fp  
Tracer.on
```

log.txt

```
...  
#1:/path_to_app/app/views/books/index.html.erb:1:ActionView::CompiledTemplates:-: <%= @books = [] %>  
...
```

フィルターすることも可能

```
Tracer.add_filter do |event, file, line, id, binding, klass|  
  file == "/path_to_app/app/views/books/index.html.erb"  
end
```

Tracer.on フィルター中ではソースコードを直接取れない (はず)
 = TracePointと同じく必要ならソースファイルから取得

ObjectSpace.allocation_sourcefile

オブジェクトの生産地を知る

ここまで説明してきた作戦: どこでインスタンス変数へ代入されたかを知る

新作戦: どこでオブジェクトが作られたかを知る

ObjectSpaceとobjspaceライブラリ

ObjectSpaceは組み込みライブラリ

objspaceはObjectSpaceに追加でメソッドを生やす標準添付ライブラリ

ObjectSpace.allocation_sourcefile オブジェクトの生成元ファイル

ObjectSpace.allocation_sourceline オブジェクトの生成元ファイル行番号

books_controller.rb

```
def index
  @books = Book.all
  require 'objspace'
  ObjectSpace.trace_object_allocations_start
end
```

index.html.erb

```
<%= p(ObjectSpace.allocation_sourcefile(@books) => ObjectSpace.allocation_sourceline(@books)) %>
```

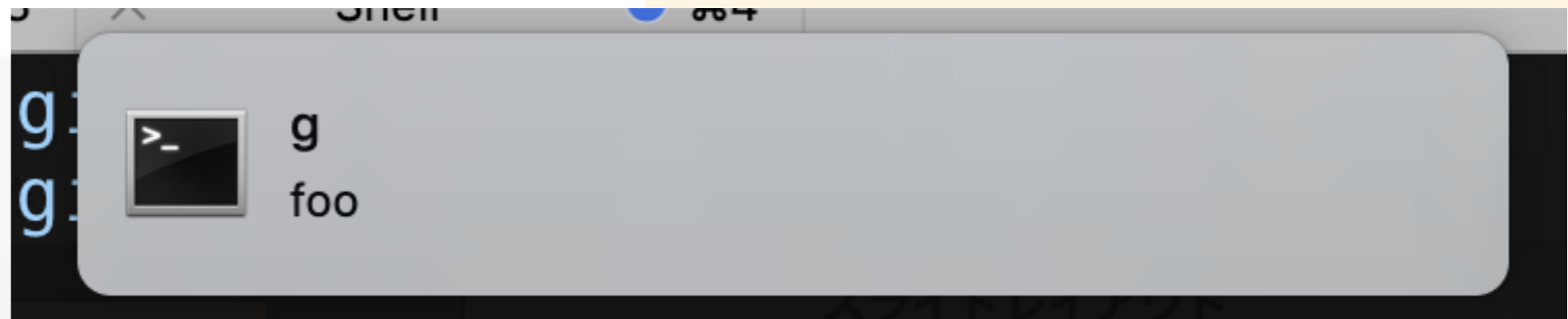
```
#=> {"path_to_app/app/views/books/index.html.erb"=>1}
```

g gem

派手なデバッグ <https://github.com/jugyo/g>

p メソッドのnotification版

 g.rb



```
require "g"
```

```
g "foo"
```

station_signage gem with g

https://github.com/igaiga/station_signage

The screenshot shows a web browser window with the following details:

- System tray: Includes icons for network, battery (100%), and system clock (5月9日(木) 9:12).
- Browser tabs: One tab titled 'Demoapp' is open.
- Address bar: Shows 'localhost:3000/books' with navigation and search icons.
- Page content:
 - Books** (Section Header)
 - Table with columns: **Title**, **Author**, and actions (Show, Edit, Destroy).
 - Row 1: Title 'Ruby超入門', Author 'igaiga', actions Show, Edit, Destroy.
 - Link: New Book

station_signage gem with g

https://github.com/igaiga/station_signage

リクエストしたページのコントローラ名、アクション名を表示
やってること

```
gem "g"
```

```
gem "terminal-notifier"
```

```
↑g gemが内部で利用するgem
```

```
application_controller.rb
```

```
before_action { g "#{self.class.name}##{action_name}" }
```

signal & trap

時間軸でデバッグ

```
Signal.trap(:INFO) {
```

```
  p "trap!!!"
```

← infoシグナルを受け取ると動く

```
}
```

```
while true
```

```
  puts "hi"
```

```
  sleep 1
```

```
end
```

macでは ctrl-T で infoシグナルを発生
または \$ kill -s info [pid]

エピソード

白魔術はオブジェクトたちの舞台を観る魔法

白魔術を使うと
舞台上のオブジェクトたちと対話したり
様子を知ることができる

オブジェクトの世界

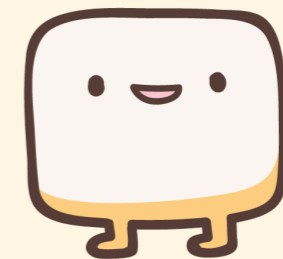


ソースコードの世界



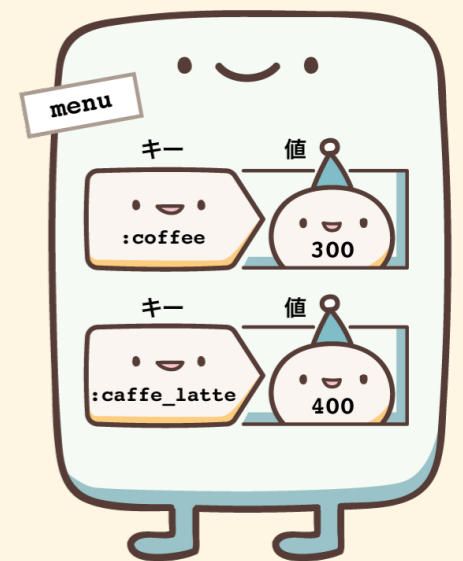
©becolomochi

Integer



©becolomochi

String



©becolomochi

Hash

白魔術は人を癒やす魔法

プログラミングの多くの時間は
デバッグに費やされる

デバッグを楽にできるようにになれば、
それは私たちへ癒やしとなる 🍺 🌴 🍹

もっと良いデバッグの技術はあるはずだし、
もっと体系だったデバッグ本があっても良い

松田さんのstill_lifeとかhocus_pocusとかめっちゃ便利でかっこいい

白魔術と黒魔術は表裏一体

黒魔術を学べば白魔術のスキルも得る

白魔術を学べば黒魔術のスキルも得る

白魔術は黒魔術として使うこともできる

強力な力をどう使うかはあなた次第

メタプログラミングRuby matz序文

「Rubyは君を信頼する。Rubyは君を分別のあるプログラマとして扱う。Rubyはメタプログラミングのような強力な力を与える。ただし、大いなる力には大いなる責任が伴うことを忘れてはいけない。」

「それでは、
*Ruby*で
たのしいプログラミングを。」

matz

カーテンコール

参考資料

- **Hijacking Ruby Syntax in Ruby**
 - <https://www.youtube.com/watch?v=04HGGQEW3A6Y>
- **RubyVM::AbstractSyntaxTree::Node 詳細**
 - <https://qiita.com/siman/items/3017ed399ded43229189>
- **I am a puts debugger**
 - <http://tenderlovmaking.com/2016/02/05/i-am-a-puts-debuggerer.html>
- **Ruby Debugging Magic Cheat Sheet**
 - <https://www.schneems.com/2016/01/25/ruby-debugging-magic-cheat-sheet.html>
- **rubyでいいかんじにDIするhabuの紹介**
 - https://qiita.com/hanachin_/items/8aa4bd82258bb19b7f91
- **†Ruby黒魔術経典†**
 - <https://speakerdeck.com/joker1007/rubyhei-mo-shu-jing-dian>
- **Railsdm Podcast unasuke.fm #5 Ruby Tendency**
 - <https://soundcloud.com/railsdm>

謝辞 (敬称略)

偉大な黒魔術師のみなさん

joker, moris, koic, hanachin, siman, alitaso
相談に乗ってくださったみなさん

ko1, mame, _tad_
ネタを提供してくれたみなさん

aaron, jugyo
Ruby超入門チームのみなさん

machu, beco
私がRubyを始めるきっかけになったtDiary開発チームのみなさん

tdtds, hsbt, machu

今日の準備をしてくれたスタッフのみなさん

成果

- **bugs.ruby-lang** 起票 "Tracing instance variable assignment"
 - <https://bugs.ruby-lang.org/issues/15854>
- **instance_variable_tracer gem** 作成
 - https://github.com/igaiga/station_signage
- **station_signage gem** 作成
 - https://github.com/igaiga/station_signage
- **RubyConf台湾 CFP** 採択
- **Ruby**の知識

五十嵐邦明/igaiga 自己紹介

新しい仕事は募集していませんが、札幌🇯🇵のご用意がある方はお声かけください

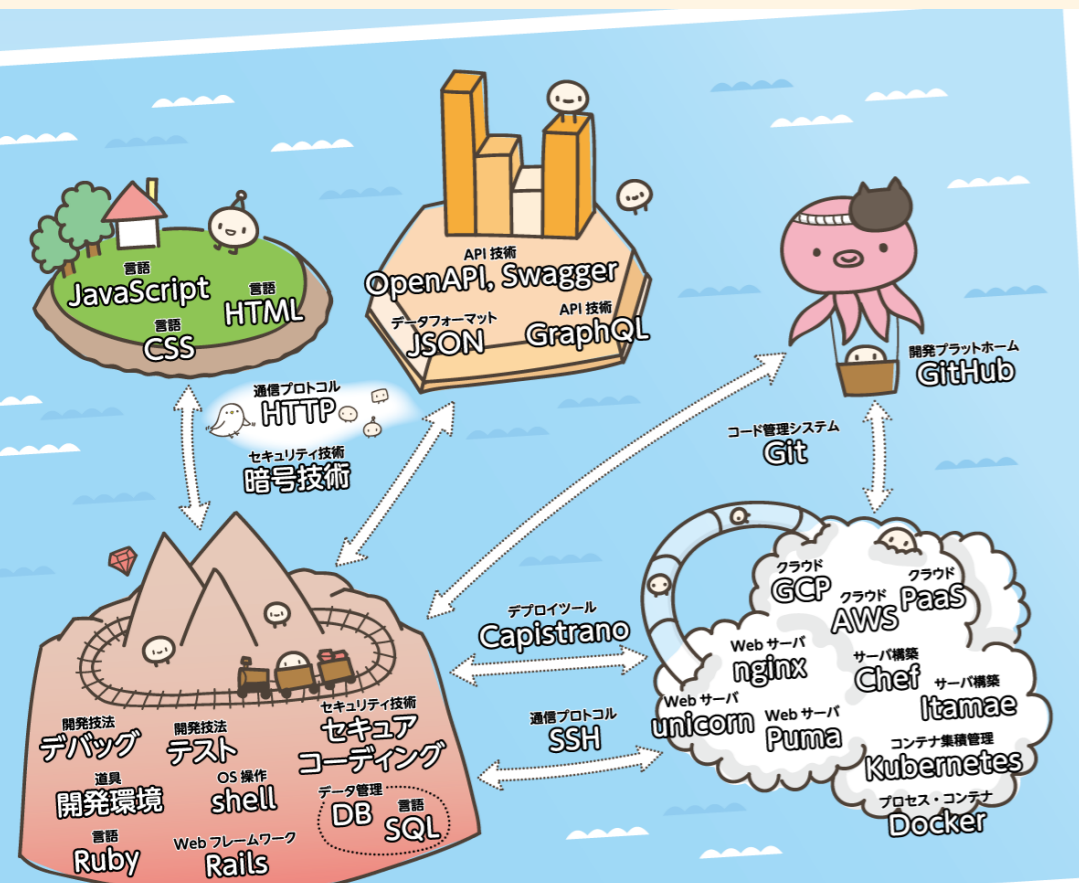
フリーランスRails/Rubyエンジニア

取引先: NaClさん, ピクシブさん, OSCRCさん, IDCフロンティアさん, Classiさん, サイバーセキュリティクラウドさん, GMOペパボさん

著書: 「ゼロからわかるRuby超入門」 「Railsの教科書」

「RubyとRailsの学習ガイド」

<https://twitter.com/igaiga555>

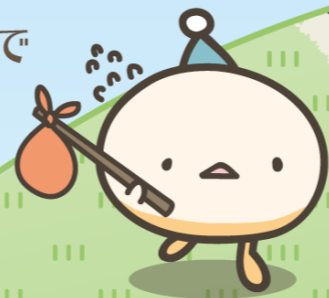


RubyとRailsの 学習ガイド

2019 技術書典6 拡大版

電子書籍版 (PDF)

BOOTHで
発売中!



ゼロからわかる
Ruby 超入門
◇紙版
◇電子版 (PDF, Kindle)
・技術評論社サイト
・書店
・Amazon
で発売中!

DAIMYO

Engineer College

by GMOペパホ"



五十嵐 邦明



近藤 宇智朗



ガーネットテック373株式会社

2019.7.1 設立

