

Ruby vs Go

RJ Isao Aruga

概要

1. rubyistの求めること
2. rubyとgoの良いところ悪いところ
3. 結論

rubyistクラスタの人が求めて いること(主観です)

- 楽しくプログラムをしたい (matzもこれをを目指している)
 - 簡単に余計なことを書かずに書ける
 - すぐ動かせる
 - (メタプログラミングを書きたい・・・?)

Goだと

- rubyよりも書かなきゃいけないことは多いけど、その分安心して書けるぞ！
- コンパイル言語だけど、コンパイル早いしその場で実行もできるよ
- (メタプログラミングはやめよう)
- rubyが遅いってよく言われてるから、速いGoを使う

Rubyも良いけど
Goも良いよ
比較していくよ！

型

- Ruby
 - 動的型付け
 - 型は書かない！Rubyの信念(matz談)
 - https://twitter.com/yukihiro_matz/status/773871448435720192

```
str = "hello world"  
print str
```

型

- Go
 - 静的型付け言語
 - 型推論はある

```
var str = "hello world"  
fmt.Println(str)
```

コーディングスタイル

- Ruby
 - rubocopが一般的
 - 設定する項目が多すぎる
 - 設定値のデファクトがない？
 - 初期値の設定が厳しすぎてハゲる
 - RubyMineがやってくれたりもする

コーディングスタイル

- Go
 - go fmt さいつよ。
 - goのformatに従うのが標準的
 - どのプロジェクト見ても同じなので見やすい。
 - 頭空っぽにして従えばいい

標準ライブラリ (HTTP Server)

- ruby
こんな感じ？あんまり使ったことないです。

```
require 'webrick'
srv = WEBrick::HTTPServer.new({ :DocumentRoot => './',
                                 :BindAddress => '127.0.0.1',
                                 :Port => 8080})
srv.mount('/view.cgi', WEBrick::HTTPServlet::CGIHandler,
          trap("INT")){
    srv.shutdown
}
srv.start
```

標準ライブラリ (HTTP Server)

- go
これは簡単！！

```
http.HandleFunc("/", func(w http.ResponseWriter, r *http.R
    fmt.Fprintf(w, "Hello, World")
})
log.Fatal(http.ListenAndServe("localhost:8000", nil))
```

標準ライブラリ (Parser)

- Ruby (JSON, XML, YAML 標準)
簡単！

```
require 'json'  
JSON.parse('{"foo":"bar"}')
```

- Go (JSON 標準)

ハッシュとかにならないので、ちょっとめんどくさい

```
type Sample struct {
    Id int
}

func main() {
    data := "{\"id\": 3}"
    v := Sample{}
    err := json.Unmarshal([]byte(data), &v)
    if err != nil {
        return
    }
    fmt.Println(v.Id)
}
```

Test

- ruby
 - rspec
 - Minitest

ちょっと覚えること多くて難しいかなあ
- Go
 - go test

標準のテスト。高機能とはいえないけど、覚えること少ないし、coverageやbenchmarkも簡単に取れる。
assert書かるのは違和感だけど、慣れる。

実行環境

- Ruby
 - 環境ごとにバージョンにあわせたRubyをインストールしなければいけない
 - ビルドしてインストールして・・・rubyを消すときも作ったときのログを手がかりに消さないと・・・
- Go
 - Cross compiler
 - シングルバイナリで動作する

開発環境 (Editor)

- Ruby
 - Ruby Mine (有料: 基本はこれかなあ)
 - Vim (あれこれ設定すると便利。最近使ってないのでわからんが、補完機能が微妙だった気がする。)
 - vimに慣れてない(Emacs派のことを言っているわけではない。Emacsでも開発できるのかもしれないけど知らない。)人からの敷居が高い

開発環境 (Editor)

- Go
 - Atom
 - Vim (何回も書くが、Emacsは知らん！！)
 - IntelliJ?
 - Eclipse?
基本、fmtが標準であるから、何で書いてもいい
んじゃないかなー

開発環境(複数バージョン)

- Ruby
 - rbenv で複数バージョン管理しながら使う
- Go
 - 簡単には、GOROOTを切り替えてあげれば良い。
 - gvm というツールで管理する方法もある
<https://github.com/moovweb/gvm#features>

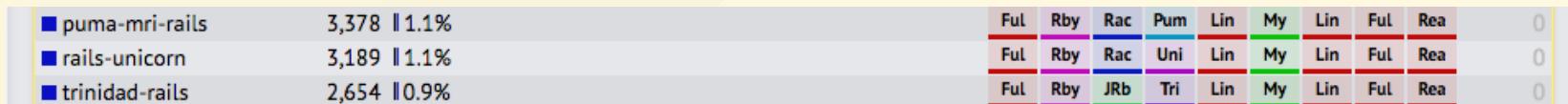
実行速度

frame workごとに実行速度を比較したサイト

<https://www.techempower.com/benchmarks/>

Goの方が数十倍速い

- rails



- go



Rubyの黒魔術

Rubyの黒魔術といえば、メタプログラミング。メタプログラミングはRubyの特徴的な機能であり、うまく使うことで、Ruby on Railsなどが実装されている。

ただし、使い方を間違えれば、たちまち何を可読性が落ち、メンテナンスが不可能なコードが出来上がる可能性を秘めた素晴らしい機能である。

黒魔術(オープンクラス)

既存のクラスを変更できる。

```
foo = nil
if foo.nil?
  p "foo is nil"
else
  p "foo is not nil"
end
#=> foo is nil
```

黒魔術(オープンクラス)

NilClassを変更してやるぞ！！！

```
class NilClass
  def nil?
    false
  end
end
```

NilClassなのに、nil?(nilの場合trueを返すメソッド)で
falseを返すようにしてみた

黒魔術(オープンクラス)

同じことをしてみる

```
foo = nil
if foo.nil?
  p "foo is nil"
else
  p "foo is not nil"
end
#=> foo is not nil
```

すごい・・・！！※ 極端な一例です

黒魔術(method_missing)

メソッドがないからって、メソッドが実行されないなんて誰も言っていない。(ほんまか?)

```
class MissingObj
  def method_missing(method_name, *args)
    p "#{method_name} #{args.join(' ')}"
  end
end
foo = MissingObj.new
foo.hello("world")
#=> hello world
```

メソッドの定義元ジャンプとかも出来ないんで、辛い。ほんまつらい。

黒魔術(特異メソッド)

```
foo = "foo"
def foo.bar
  "bar"
end
p foo.bar
#=> bar
p "bar".bar
#=> Error
```

黒魔術

紹介したのはほんの一部です。(evalとかsendとか・・・メソッドの動的な追加だとか・・・)
rubyは怖い言語ですね。

Goのreflect

黒魔術とまでとはいかないが、型を前もって知ることが出来ない場合に、使うことが出来る。

ただし、次のタイトルのように、使うときはよく考えて使うべき

The Dark Arts Of Reflection

みんなのGo第5章の題名です。

動的に型情報を見て、動作は変えられる。

```
func reflection(v interface{}) {
    vv := reflect.ValueOf(v)
    switch vv.Kind() {
    case reflect.Map:
        ...
    case ...
```

The Dark Arts Of Reflection

値を突っ込むことも出来る ※ ポインタを指定する必要はある。

```
func refl(x interface{}) {
    v := reflect.ValueOf(x).Elem()
    if v.Kind() == reflect.Int && v.CanSet() {
        v.SetInt(100)
    }
    println(v.Int())
}
```

基本的にGoはこの程度出来るくらい
変にメタメタなコードが書かれないので、可読性が落
ちにくい(と思う)

結論

Goのほうが良いよ！と書こうと思っていたが、ruby
もやっぱりいいところ多いね。

ただ、速度とか、メタメタなところとか、スタイルあ
たりに辟易しているのであれば、Golangやってみると
楽しいかもしれない！

Goだと普通のエディタで書けるからスクリプト言語
系の人でもとっつきやすい！

Shall we Go?

おまけ

並行処理

- ruby
 - thread size 2mb
- Go
 - goroutine
 - stack size 10KB 軽量 !
 - CSP(communicating sequential processes)
 - channel
 -

Frame Work

- Ruby
 - Ruby on Rails
- Go
 - Echo, Gin , etc...