Ruby 初級者向けレッスン第 22 回

okkez @ Ruby 関西, モリ@小波ゼミ

2008 年 07 月 26 日

今回の内容

- irb の基礎知識
- irb の便利な使い方

今回のゴール

- irb の使い方を知る

irb について

- Intaractive Ruby の略
- Ruby をインストールするともれなく付いてきます。
- Ruby の式を標準入力から簡単に入力・実行することができます。
- Rubyist 必携の一品

irb でよく使うパターン

- ほとんど overflow しない電卓として
  - メモリが続く限り計算できるハズ
- クラスやメソッドの動作確認
  - 正規表現のマッチング確認
  - フォーマット文字列の表示確認
- メソッドを探す
  - あのクラスにあるはずのメソッドを探したい
  - クラスの継承関係を調べたい
  - あのモジュールは include されているのか

簡単な使用例

```
>> 10 + 100         # 普通の計算
=> 110              # 結果が表示される
>> def add(x, y)    # メソッド定義もできる
>>   x + y
>> end
=> nil
>> add(10 + 100)    # 引数を間違えた
ArgumentError: wrong number of arguments (1 for 2)
        from (irb):5:in `add'
        from (irb):5
>> add(10,100)      # 先ほど定義したメソッドの呼び出し
=> 110              # 結果が表示される
```

もっと使用例

紙面の都合上、適宜改行しています。

```
>> 2 ** 1000
=> 10715086071862673209484250490600018105614048117055336O \
   74437503883703510511249361224931983788156958581275946 7 \
   29175531468251871452856923140435984577574698574803934 5 \
   67774824230985421074605062371141877954182153046474983 5 \
   81941267398767559165543946077062914571196477686542167 6 \
   60429831652624386837205668069376
>> /\ARuby/ =~ 'Perl'
=> nil
>> /\ARuby/ =~ 'Ruby'
=> 0
>> /Ruby/ =~ 'Perl Ruby'
=> 5
>> Time.new.strftime('%Y-%m-%d')
=> "2008-07-20"
>> [].methods.sort.select{|m| /ect\z/ =~ m }
=> ["collect", "detect", "inject", "inspect", \
    "pretty_inspect", "pretty_print_inspect", "reject", "select"]
>> Array.ancestors
=> [Array, Enumerable, Object, PP::ObjectMixin, Kernel]
```

irb のカスタマイズ方法

コマンドライン引数で行う

```
$ irb --help
Usage:  irb.rb [options] [programfile] [arguments]
  -f                ~/.irbrc を読み込まない.
  -m                bc モード (分数, 行列の計算ができる)
  -d                $DEBUG を true にする (ruby -d と同じ)
  -r load-module    ruby -r と同じ.
  -I path           $LOAD_PATH に path を追加する.
  --inspect         結果出力に inspect を用いる (bc モード以外はデフォルト).
  --noinspect       結果出力に inspect を用いない.
  --readline        readline ライブラリを利用する.
  --noreadline      readline ライブラリを利用しない.
  --prompt prompt-mode/--prompt-mode prompt-mode
                    プロンプトモードを切替えます. 現在定義されているプ
                    ロンプトモードは, default, simple, xmp, inf-ruby が
                    用意されています.
  --inf-ruby-mode   emacs の inf-ruby-mode 用のプロンプト表示を行なう. 特
                    に指定がない限り, readline ライブラリは使わなくなる.
  --simple-prompt   非常にシンプルなプロンプトを用いるモードです.
  --noprompt        プロンプト表示を行なわない.
  --tracer          コマンド実行時にトレースを行なう.
  --back-trace-limit n
                    バックトレース表示をバックトレースの頭から n, 後ろ
                    から n だけ行なう. デフォルトは 16
  --irb_debug n     irb のデバッグデバッグレベルを n に設定する (利用しな
                    い方が無難でしょう).
  -v, --version     irb のバージョンを表示する
```

設定ファイルに記述しておく

- ホームディレクトリの .irbrc というファイルを読み込みます。
- .irbrc は Ruby スクリプト。
- ホームディレクトリに .irbrc が存在しない場合は、カレントディレクトリの .irbrc, irb.rc, _irbrc, $irbrc を順番にロードしようと試みます。
- 環境変数の確認方法
  - Unix(cygwin) : $ echo $HOME
  - Windows : $ echo %HOME%

- Windows の場合は設定ファイルを指定した irb を呼び出すバッチファイルを作成しておくのが便利です。

irb のカスタマイズ内容

- プロンプトのカスタマイズ
- よく使うライブラリを読み込んでおく
- よく使う処理をメソッドとして定義しておく

おすすめの設定

```
# pp, yaml を使う
require 'pp'
require 'yaml'

# タブ補完を有効にする
require 'irb/completion'

# simple prompt
IRB.conf[:PROMPT_MODE] = :SIMPLE

# ヒストリーを有効にする
require 'irb/ext/save-history'
IRB.conf[:SAVE_HISTORY] = 1000
IRB.conf[:HISTORY_FILE] = "#{ENV['HOME']}/.irb-save-history"

# そのオブジェクトで呼び出せる全てのメソッドを表示する (via The Ruby Way)
# obj.class.instance_mathods(false) + obj.singleton_methods
def sm(o)
  (o.class.ancestors - [o.class]).inject(o.methods){|r, c| r -= c.methods }.sort
end

# ri を使えるようにする
# 例.
# irb> String.ri
# irb> ri String  # same as above
# irb> String.ri 'reverse'
# irb> ri 'File.new'
def ri(*args)
  puts `ri #{args.join(' ')}`
end
class Module
```

```
  def ri(meth=nil)
    if meth
      if instance_methods(false).include? meth.to_s
        puts `ri #{self}##{meth}`
      else
        super
      end
    else
      puts `ri #{self}`
    end
  end
end
# refe2 も使えるように
# 例.
# r 'Kernel#sprintf'
# r 'Kernel.sprintf'
# r :Array, :sort
module Kernel
  def r(*args)
    puts `refe2 #{args.join(' ')}`
  end
  private :r
end

class Module
  def r(meth = nil)
    if meth
      if instance_methods(false).include? meth.to_s
        puts `refe2 #{self}##{meth}`
      else
        super
      end
    else
      puts `refe2 #{self}`
    end
  end
end
```

exit, quit, abort の違い

- exit, quit は irb_exit に対するエイリアス
  - サブ irb で使用した場合は、その irb だけ終了する。

  - 終了ステータス ($?) は 0

- abort は Kernel#abort を呼び出す。
  - サブ irb で使用した場合は、全ての irb を終了する。
  - Kernel#abort は異常終了させるためのものなので通常は irb_exit を使用すること。
  - 終了ステータス ($?) は 1

load と require

- load は呼び出すたびに対象ファイルを読み直す
  - 自作ライブラリの動作確認時に便利
  - 何回読み込んでも true を返す
- require は一度読み込んだファイルは二度と読み直さない
  - 二回目以降は false を返す

サブ irb

起動時の irb インタプリタとは独立した環境を持つ「サブ irb」を任意の数だけ起動することができます。サブ irb は、irb 実行中に「irb」と入力すると起動します。
例えば以下の実行例を見てください。

```
irb(main):004:0> x = "OK"          # ローカル変数 x を定義
=> "OK"
irb(main):005:0> x                 # x を表示
=> "OK"
irb(main):006:0> irb               # サブ irb を起動
irb#1(main):001:0> x               # x を表示
NameError: undefined local variable or method `x' for main:Object
        from (irb#1):1:in `Kernel#binding'
```

起動時のインタプリタでローカル変数 x を定義しましたが、「irb」でサブ irb を起動すると、ローカル変数 x が見えなくなっています。これが「独立した環境」の意味です。
また .irbrc に以下のように設定しておくとサブ irb の時だけプロンプトを変更できます。

```
# サブ irb の設定（元は :SIMPLE が設定されている）
IRB.conf[:IRB_RC] = lambda{|_|
  IRB.conf[:IRB_RC] = lambda{|conf| conf.prompt_mode = :DEFAULT }
}
```

実行例:

```
$ irb
>> Foo = Class.new
=> Foo
>> irb Foo
irb#1(Foo):001:0> def foo
irb#1(Foo):002:1>   puts 'called foo'
irb#1(Foo):003:1> end
=> nil
irb#1(Foo):004:0> exit
=> #<IRB::Irb: @context=#<IRB::Context:0xb7a1428c>,
    @signal_status=:IN_EVAL, @scanner=#<RubyLex:0xb7a113ac>>
>> Foo.new.foo
called foo
=> nil
>>
```

上記のようにサブ irb の実行時のコンテキストを指定することもできます。

演習問題

いろいろな計算をしてみよう

通常モードと bc モードで計算結果の違いを見てみましょう。

1. 1 + 1
2. 10 / 4
3. 100 ** 10000
4. sqrt(16 / 36)
5. BMI を計算してみましょう。BMI = 体重 (kg) / 身長 (m) ** 2
6. 1 年は何時間でしょう？
7. 10 年は何分でしょう？
8. わたしが生まれてから 9 億 3400 万秒経っているとしたら今何歳でしょう？

irb のオプションをいろいろといじってみよう

リファレンスマニュアルを調べてみましょう。

1. 動作中、一時的に結果表示を消してみよう

```
irb(main):007:0> 1+1
=> 2                                # 計算結果が表示される
irb(main):008:0> ***********        # 何かする
irb(main):009:0> 1+1                # もう一度計算する
irb(main):010:0>                    # 何も表示されない
```

1. 動作中にプロンプトの一部を変更してみよう

全体を変更する場合は以下のようにしますが、一部だけ変更するにはどうするでしょう？

```
irb(main):010:0> conf.prompt_mode = :SIMPLE
>> conf.prompt_mode = :DEFAULT
irb(main):012:0>
```

メソッドの動作確認をしてみよう

各メソッドの使い方はリファレンスマニュアルなどで確認してください。

1. Array#zip
2. Kernel#rand, Kernel#srand
3. Array#map, Array#select などにブロックを渡さない場合にどんなオブジェクトが返されるか調べてみましょう。
4. require 'yaml' して、 Kernel#y にいろいろなオブジェクトを渡してみよう。

作ってみよう

irb で動作確認しながら作ってみましょう。

1. 計算 100 もどきを作りましょう。

仕様

- コンピュータが計算問題を次々に出題します
- 回答はキーボードから入力します
- 出題される計算問題は難易度を選択出来ます (最初に選ぶ)
  - かんたん：一桁同士の四則演算 (割り算なし) のみ
  - むずかしい：四則演算 (二桁の計算もあり、割り切れない除算はなし、答えに小数は出ない)
- 合計で 100 問出題します
- 10 問ごとに何問終了したか出力します

- 最後に以下のものを出力します
  - 正解数、不正解数
  - 全問回答するのにかかった時間

実行イメージ

計算 100 開始
難易度を選択してください。 1:かんたん 2:むずかしい >
1:かんたん を開始します。Press Enter >
1 + 1 = [回答入力する]
正解|不正解を表示する
1 + 2 = [回答入力する]
正解|不正解を表示する
....
1 + 2 = [回答入力する]
正解|不正解を表示する
10 問突破 #=> 10 問ごとに何問終わったか出力する
....
1 + 2 = [回答入力する]
正解|不正解を表示する
100 問突破
100 問終了しました
正解　 : xx 問
不正解 : xx 問
タイム : xx 分 xx 秒

参考文献

初めてのプログラミング
http://www.oreilly.co.jp/books/4873112923/

プログラミング Ruby 第 2 版 言語編
http://ssl.ohmsha.co.jp/cgi-bin/menu.cgi?ISBN=4-274-06642-8

たのしい Ruby 第 2 版
http://shop.sbcr.jp/bm_detail.asp?sku=4797336617

Ruby Way 第 2 版
http://seshop.com/detail.asp?pid=9027

今後の情報源

公式 Web サイト

http://www.ruby-lang.org/

Ruby リファレンスマニュアル刷新計画

http://doc.loveruby.net/

日本 Ruby の会

http://jp.rubyist.net/

Rubyist Magazine

http://jp.rubyist.net/magazine/

okkez weblog

http://typo.okkez.net/