Ruby 道チュートリアル

かずひこ@株式会社 ネットワーク応用通信研究所

2006 年 6 月 29 日

前回のあらすじ

『アジャイルと言語〜Is Ruby Agile?』

2005 クリスマスイベント・まつもとゆきひろさん講演

- Ruby はアジャイルか？
  - 答えはもちろん YES!
- Ruby のどこがアジャイルなのか？
  - 動的型 − 型チェックは動的に行われる
  - 高い柔軟性
  - 型チェックがなくて不安では？ − 結局はテストが必要
- Ruby のどこがアジャイルなのか？（その 2）
  - メタプログラミング
  - リフレクション − プログラム自身を操作するプログラム
  - アダプティブプログラミング − プログラム自身が変化する
- Ruby のどこがアジャイルなのか？（その 3）
  - 簡潔な表記
  - 多様なリテラル
  - 宣言が不要
  - 変数の種別は名前で分かる
  - 無駄な記述はなるたけ避ける − Don't Repeat Yourself
- Ruby のどこがアジャイルなのか？（その 4）
  - 高い生産性
  - コンパクトなコード
  - ダイレクトなプログラミング
  - 楽しいプログラミング − プログラミングに喜びを、再び

今日のメニュー

- Rubyist の基礎知識
- Ruby で書こう Iterator パターン
- Ruby で書こう Delegate パターン

Rubyist の基礎知識

- 変数の種類
- リファレンスマニュアル
- 対話型 ruby（irb）

変数の種類

- ローカル変数（例：foo）
  - ローカル変数は小文字または_で始まる。初期化されていないローカル変数の参照は引数のないメソッド呼び出しとみなす。
- インスタンス変数（例：@foo）
  - インスタンス変数は@で始まる。初期化されていないインスタンス変数の値は nil。
- クラス変数（例：@@foo）
  - クラス変数は@@で始まる。初期化されていないクラス変数の参照はエラー。
- グローバル変数（例：$foo）
  - グローバル変数は$で始まる。初期化されていないグローバル変数の値は nil。
- 定数（例：Foo）
  - 定数は大文字で始まる。初期化されていない定数の参照はエラー。定義済みの定数への再代入は警告。
- 擬似変数
  - self 現在のメソッドの実行主体。
  - true 真の代表値。
  - false 偽の代表値。
  - nil 未定義を示す値。偽とみなす。
  - __FILE__ 現在のソースファイル名
  - __LINE__ 現在のソースファイル中の行番号

リファレンスマニュアル

- ウェブのリファレンスマニュアル（日本語）
  - http://www.ruby-lang.org/ja/man/
- ri（英語）
  - Ruby に同梱されている
- refe（日本語）
  - http://i.loveruby.net/ja/prog/refe.html
- rbbr（日本語・英語）
  - http://ruby-gnome2.sourceforge.jp/hiki.cgi?rbbr

メソッドの表記

- リファレンスマニュアルでは、以下のような表記でメソッドを記述する

Klass#method
: Klass というクラスの method というインスタンスメソッド

Klass::method または Klass.method
: Klass というクラスの method というクラスメソッド

ri の使い方

- クラスの説明
  - （例）ri Array
- メソッドの説明
  - （例）ri Array.new
  - （例）ri Array#each
- あいまいに探す
  - （例）ri A
  - （例）ri A.e
- ヘルプの表示
  - ri –help

- クラス・モジュールの一覧
  - ri -c
- クラス・モジュール・メソッドの一覧
  - ri -l

対話型 ruby（irb）

- 式を入力すると、すぐに実行して式の値を表示する

```
$ irb
irb(main):001:0> 1 + 2 ←式を入力
=> 3 ←式の値を出力
irb(main):002:0> puts "Hello" ←式を入力
Hello ← puts の実行結果
=> nil ← 式の値を出力（puts の返り値は nil）
```

Ruby で書こう Iterator パターン

- 配列のような複数のオブジェクトを格納したコンテナオブジェクトに対して、各要素に順にアクセスする方法を提供する

外部イテレータ

- コレクション内の各要素にアクセスするためのオブジェクトを別に用意する
- my_iterator.rb

```
class Array
  def iterator
    return ArrayIterator.new(self)
  end
end

class ArrayIterator
  def initialize(array)
    @array = array
    @current = 0
  end

  def first
```

```
    @current = 0
  end

  def next
    @current += 1
  end

  def is_done
    return @current >= @array.length
  end

  def current_item
    return @array[@current]
  end
end
```

- **標準クラス** Array **に、**iterator **メソッドを追加している**
- **リファレンスを読もう**
  - Array#length, Array#[]
- my_iterator_sample.rb

```
require 'my_iterator'

array = Array.new
array << 1
array << 2
array << 3
it = array.iterator
it.first
until it.is_done
  p it.current_item
  it.next
end
```

- **リファレンスを読もう**
  - Array#<<, Kernel#p

内部イテレータ

- **コレクションそのものが、その中の各要素へのアクセスを提供する**

- my_iterator_sample2.rb

```
require 'my_iterator'

array = Array.new
array << 1
array << 2
array << 3
array.each do |elem|
  p elem
end
```

- **リファレンスを読もう**
  - Array#each
- each **の後ろの「**do〜end**」は、**each **メソッドに渡されるブロックで、「**{〜}**」と書くこともできる**
- **ブロック内の「**|elem|**」のように、ブロックパラメータを指定する**

外部イテレータと内部イテレータの比較

- **内部イテレータは、二つのコレクションの要素を順に比較するような処理を書けない**
- **外部イテレータは、イテレータオブジェクトがコレクションオブジェクトの内部情報を参照する必要がある**

Ruby で書こう Delegate パターン

- **オブジェクトの機能を再利用する手法として、再利用したい機能を自分に取り込むのではなく、その機能を持つオブジェクトに処理を委譲する**

キューを作ろう

- **先に入ったものから先に出る（**FIFO**）リスト。待ち行列。**

initialize
: **空のキューを作る**

enq(x)
: **キューの最後に** x **を追加する**

deq
: **キューの最初の要素を取り除く**

peek
: **キューの最初の要素を返す（キューは変更しない）**

length
: **キューの要素の数を返す**

empty?
: **キューが空なら真、空でなければ偽**

- **実行例**

```
q = MyQueue.new
q.enq(1)     # 後ろに 1 を追加
q.enq(4)     # 後ろに 4 を追加
q.enq(2)     # 後ろに 2 を追加
q.peek #=> 1 # 先頭を問い合わせる
q.deq        # 先頭を取り除く
q.peek #=> 4 # 先頭を問い合わせる
```

キューのテストコード

- test_queue.rb

```
require 'test/unit'

class TestMyQueue < Test::Unit::TestCase
  def setup
    @queue = MyQueue.new
  end

  def test_empty?
    assert(@queue.empty?, 'a new queue is empty.')
  end

  def test_enq_and_peek
    @queue.enq(3)
    assert_equal(3, @queue.peek, 'peek returns the first value.')
  end

  def test_enq_and_length
    @queue.enq(3)
    assert_equal(1, @queue.length, 'enq increments the length.')
    @queue.enq(5)
    assert_equal(2, @queue.length, 'enq increments the length.')
```



```
  end

  def test_enq_and_empty?
    @queue.enq(3)
    assert_equal(false, @queue.empty?, 'a queue with data is not empty.')
  end

  def test_enq_enq_deq_and_length
    @queue.enq(3)
    @queue.enq(5)
    @queue.deq
    assert_equal(1, @queue.length, 'deq decrements the length.')
  end

  def test_enq_enq_deq_and_peek
    @queue.enq(3)
    @queue.enq(5)
    assert_equal(3, @queue.peek, 'peek returns the first value.')
    @queue.deq
    assert_equal(5, @queue.peek, 'peek returns the first value.')
  end
end
```

- **リファレンスを読もう**
  - Test::Unit::TestCase#setup, Test::Unit::Assertions#assert, Test::Unit::Assertions#assert_equal
- –help **を付けるとテストのヘルプが表示される**

```
$ ruby test_queue.rb --help
Test::Unit automatic runner.
Usage: test_queue.rb [options] [-- untouched arguments]

    -r, --runner=RUNNER              Use the given RUNNER.
                                     (c[onsole], f[ox], g[tk], g[tk]2, t[k])
    -n, --name=NAME                  Runs tests matching NAME.
                                     (patterns may be used).
    -t, --testcase=TESTCASE          Runs tests in TestCases matching TESTCASE.
                                     (patterns may be used).
    -v, --verbose=[LEVEL]            Set the output level (default is verbose).
                                     (s[ilent], p[rogress], n[ormal], v[erbose])
        --                           Stop processing options so that the
                                     remaining options will be passed to the
                                     test.
    -h, --help                       Display this help.
```

キューと Array の違い

- length と empty?は Array の同名メソッドと同じ
- enq は Array#push と同じ
- deq は Array#shift と同じ
- peek は Array#first と同じ
- リファレンスを読もう
  - Array#empty?, Array#push, Array#shift, Array#first

まずは継承

- スーパークラスにできることを継承する

```
class A
  def hello
    puts 'hello'
  end
end

class B < A # B クラスは A クラスを継承
  def bye
    puts 'bye'
  end
end

B.new.hello # → "hello"を出力
B.new.bye # → "bye"を出力
```

継承で作ろう（1）

- Array を継承して足りないメソッドを定義する（my_queue1.rb）

```
class MyQueue < Array
  def enq(x)
    push(x)
  end

  def deq
    shift
  end
```



```
  def peek
    return first
  end
end
```

- テストを実行（「-r my_queue1」で my_queue1.rb をロードする）

```
$ ruby -r my_queue1 test_queue.rb
Loaded suite test_queue
Started
......
Finished in 0.004184 seconds.

6 tests, 8 assertions, 0 failures, 0 errors
```

継承で作ろう（2）

- わざわざ定義しなくても別名で済む（my_queue2.rb）

```
class MyQueue < Array
  alias_method(:enq, :push)
  alias_method(:deq, :shift)
  alias_method(:peek, :first)
end
```

- リファレンスを読もう
  - Module#alias_method
- テストを実行（「-r my_queue2」で my_queue2.rb をロードする）

```
$ ruby -r my_queue2 test_queue.rb
Loaded suite test_queue
Started
......
Finished in 0.004184 seconds.

6 tests, 8 assertions, 0 failures, 0 errors
```

継承の問題

- 必要のないこともできてしまう

```
q = MyQueue.new
q.enq(1)
q.enq(2)
q.enq(3)
p q.peek #=> 1
p q[1] #=> 2    # キューの仕様外
p q.last #=> 3  # キューの仕様外
```

- スーパークラスとの結び付きが密接になる
- スーパークラスの内部構造の変化に追従する必要がある
- スーパークラスとメソッド名やインスタンス変数名が重複するとややこしいことになる
- 「継承は最後の武器だ……それじゃ忍者部隊月光か」by まつもとゆきひろさん

継承の問題を確認するテスト

- test_queue.rb に以下のメソッドを追加して、NoMethodError クラスの例外が発生することを確認する

```
class TestMyQueue < Test::Unit::TestCase
（略）
  def test_undefined_methods
    assert_raise(NoMethodError) do
      @queue[1]
    end
    assert_raise(NoMethodError) do
      @queue.last
    end
  end
end
```

- リファレンスを読もう
  - Test::Unit::Assertions#assert_raise
- 実行結果

```
$ ruby -r my_queue1 test_queue.rb
Loaded suite test_queue
Started
......F
Finished in 0.054905 seconds.
```



```
  1) Failure:
test_undefined_methods(TestMyQueue) [test_queue.rb:45]:
<NoMethodError> exception expected but none was thrown.

7 tests, 9 assertions, 1 failures, 0 errors
```

委譲とは？

- あるクラスの全てを「持ってくる（継承する）」のではなく、必要なものだけを「相手にお願い（委譲）」する
- 委譲先はスーパークラスでなくてよいので、関係の弱いクラス同士で使える

キューの処理の委譲

- length と empty?は Array の同名メソッドに委譲
- enq は Array#push に委譲
- deq は Array#shift に委譲
- peek は Array#first に委譲

まずは書いてみよう

- my_queue3.rb

```
class MyQueue
  def initialize
    @q = [] # 委譲するオブジェクトの準備
  end

  # 同名メソッドへの委譲
  def length
    @q.length
  end

  def empty?
    @q.empty?
  end

  # 別名メソッドへの委譲
  def enq(x)
    @q.push(x)
```

```
  end

  def deq
    @q.shift
  end

  def peek
    @q.first
  end
end
```

- **テストを実行**

```
$ ruby -r my_queue3 test_queue.rb
Loaded suite test_queue
Started
.......
Finished in 0.005964 seconds.

7 tests, 10 assertions, 0 failures, 0 errors
```

- **同じようなコードを何度も書いている** − Don't Repeat Yourself.

同じようなコードの抽出

- **同名メソッドへの委譲（**length, empty?**）**

```
def method(*args, &block)
  @accessor.method(*args, &block)
end
```

- **別名メソッドへの委譲（**enq → push, deq → shift, peek → first**）**

```
def alias_method_name(*args, &block)
  @accessor.method(*args, &block)
end
```

- **「***args**」は可変長引数で、不定個の引数を配列に格納して受け取る**
- **「**&block**」は、手続きオブジェクトをブロックとして渡すための引数で、引数列の一番最後に置く**

動的にメソッドを定義する

- my_queue4.rb

```
class MyQueue
  def initialize
    @q = [] # 委譲するオブジェクトの準備
  end

  # 同名メソッドへの委譲を定義するクラスメソッド
  def self.def_delegators(accessor, *methods)
    for method in methods
      class_eval("
        def #{method}(*args, &block)
          #{accessor}.__send__(:#{method}, *args, &block)
        end
      ")
    end
  end

  # 別名メソッドへの委譲を定義するクラスメソッド
  def self.def_delegator(accessor, method, alias_method)
    class_eval("
      def #{alias_method}(*args, &block)
        #{accessor}.__send__(:#{method}, *args, &block)
      end
    ")
  end

  def_delegators(:@q, :length, :empty?)

  def_delegator(:@q, :push, :enq)
  def_delegator(:@q, :shift, :deq)
  def_delegator(:@q, :first, :peek)
end
```

- **リファレンスを読もう**
  - Module#class_eval, Object#__send__
- **テストを実行**

```
$ ruby -r my_queue4 test_queue.rb
Loaded suite test_queue
```

```
Started
.......
Finished in 0.006048 seconds.

7 tests, 10 assertions, 0 failures, 0 errors
```

似たコードをさらに整理しよう

- my_queue5.rb

```
class MyQueue
  def initialize
    @q = [] # 委譲するオブジェクトの準備
  end

  # 同名メソッドへの委譲を定義するクラスメソッド
  def self.def_delegators(accessor, *methods)
    for method in methods
      def_delegator(accessor, method)
    end
  end

  # 別名メソッドへの委譲を定義するクラスメソッド
  def self.def_delegator(accessor, method, alias_method = method)
    class_eval("
      def #{alias_method}(*args, &block)
        #{accessor}.__send__(:#{method}, *args, &block)
      end
    ")
  end

  def_delegators(:@q, :length, :empty?)

  def_delegator(:@q, :push, :enq)
  def_delegator(:@q, :shift, :deq)
  def_delegator(:@q, :first, :peek)
end
```

- **テストを実行**

```
$ ruby -r my_queue5 test_queue.rb
Loaded suite test_queue
Started
```



```
.......
Finished in 0.005905 seconds.

7 tests, 10 assertions, 0 failures, 0 errors
```

モジュールにしよう

- my_forwardable.rb

```
module MyForwardable
  def def_delegators(accessor, *methods)
    for method in methods
      def_delegator(accessor, method)
    end
  end

  def def_delegator(accessor, method, alias_method = method)
    class_eval("
      def #{alias_method}(*args, &block)
        #{accessor}.__send__(:#{method}, *args, &block)
      end
    ")
  end
end
```

- my_queue6.rb

```
require 'my_forwardable'

class MyQueue
  extend MyForwardable

  def initialize
    @q = [] # 委譲するオブジェクトの準備
  end

  def_delegators(:@q, :length, :empty?)

  def_delegator(:@q, :push, :enq)
  def_delegator(:@q, :shift, :deq)
  def_delegator(:@q, :first, :peek)
end
```

- **リファレンスを読もう**

- Object#extend

- **モジュールは、クラスに類似しているが、スーパークラスを持たず、またインスタンスを作ることができない**
- **テストを実行**

```
$ ruby -r my_queue6 test_queue.rb
Loaded suite test_queue
Started
.......
Finished in 0.005723 seconds.

7 tests, 10 assertions, 0 failures, 0 errors
```

標準添付ライブラリ forwardable.rb

- **以下の二つのモジュールを提供する（今回とりあげたのは前者）**

Forwardable
クラスに対してメソッドの委譲機能を定義するモジュール

SingleForwardable
オブジェクトに対してメソッドの委譲機能を定義するモジュール

- forwardable.rb **を読もう**
  - /usr/lib/ruby/1.8/forwardable.rb (UNIX)
  - C:\ruby\lib\ruby\1.8\forwardable.rb (WindowsXP)
- Forwardable **モジュールはわずか** 28 **行**

```
module Forwardable
  @debug = nil
  class << self
    attr_accessor :debug
  end

  def def_instance_delegators(accessor, *methods)
    for method in methods
      def_instance_delegator(accessor, method)
    end
  end

  def def_instance_delegator(accessor, method, ali = method)
```



```
    accessor = accessor.id2name if accessor.kind_of?(Integer)
    method = method.id2name if method.kind_of?(Integer)
    ali = ali.id2name if ali.kind_of?(Integer)

    module_eval(<<-EOS, "(__FORWARDABLE__)", 1)
      def #{ali}(*args, &block)
        begin
          #{accessor}.__send__(:#{method}, *args, &block)
        rescue Exception
          $@.delete_if{|s| /^\\(__FORWARDABLE__\\):/ =~ s} unless Forwardable::debug
          Kernel::raise
        end
      end
    EOS
  end

  alias def_delegators def_instance_delegators
  alias def_delegator def_instance_delegator
end
```

- **リファレンスを読もう**
  - Forwardable, Forwardable#def_instance_delegator, Forwardable#def_instance_delegators, Module#module_eval, Kernel#raise
- **「**class << **オブジェクト〜**end**」は特異クラスの定義**
- **「**begin**〜**rescue**〜**end**」は例外処理の構文**
- **「**alias **別名 メソッド名」は、メソッド名の別名を用意する構文**

使用例を見よう

- Gonzui <http://raa.ruby-lang.org/gonzui/> **で、クラス名（**Forwardable**）やライブラリ名（**forwardable**）で検索してみよう**

使用例（ウェブアプリケーションフレームワーク Arrow）

- arrow/lib/arrow/applet.rb

```
### Add some Hash-ish methods for convenient access to FormValidator#valid.
class FormValidator
  unless method_defined?( :[] )
    extend Forwardable
```

```
    def_delegators :@form, *(Hash::instance_methods(false) - [:[], :[]=])

    def []( key )
      @form[ key.to_s ]
    end

    def []=( key, val )
      @form[ key.to_s ] = val
    end
  end
end
```

- '[]' と'[]=' 以外の全ての Hash のメソッドを委譲して、'[]' と'[]=' はキーを String にしてから参照するようにしている

演習問題：スタックを作ろう

- 後に入ったものから先に出る（LIFO）リスト。皿を積んで上から順に取る感じ。

initialize
: 空のスタックを作る

push(x)
: スタックの最後に x を追加する

pop
: スタックの最後の要素を取り除く

peek
: スタックの最後の要素を返す（スタックは変更しない）

length
: キューの要素の数を返す

empty?
: スタックが空なら真、空でなければ偽

- 実行例

```
q = MyStack.new
q.push(1)    # 後ろに 1 を追加
q.push(4)    # 後ろに 4 を追加
q.push(2)    # 後ろに 2 を追加
q.peek #=> 2 # 末尾を問い合わせる
q.pop        # 末尾を取り除く
q.peek #=> 4 # 末尾を問い合わせる
```


- 委譲を使ってスタックを実装しよう
- テスト・ファーストで実装しよう

まとめ

- 「継承は最後の武器」
- リファレンスを読もう
- ライブラリのソースを読もう
- 使用例のソースを読もう

参考文献

- 『C Magazine 2002 年 8 月号 - なぁRuby を読もうじゃないか 第 7 回 delegate.rb と weakref.rb』
- 『日経 Linux 2005 年 12 月号 - まつもと直伝 プログラミングのオキテ 第 8 回 デザイン・パターン（1）』
- 『Rubyist Magazine 0012 号 - 標準添付ライブラリ紹介 第 6 回 委譲』http://jp.rubyist.net/magazine/?0012-BundledLibraries

宣伝

- 『はじめよう Ruby on Rails』(ISBN:4756147739)
  - TDD で Ruby on Rails アプリケーションの開発をしています

今後の情報源

- 公式 Web サイト http://www.ruby-lang.org/
- リファレンスマニュアル http://www.ruby-lang.org/ja/man/
- 日本 Ruby の会 http://jp.rubyist.net/
- Rubyist Magazine http://jp.rubyist.net/magazine/
- ふぇみにん日記 http://kazuhiko.tdiary.net/