Ruby 入門

ー Ruby で Web サービスクライアントを作ろうー

板垣正敏

新潟オープンソース協会･日本 Ruby の会

導入

- スクリプト言語としての Ruby 入門です。
- Ruby で簡単なファイル入出力と、Web サービスを使った情報取得を体験します。
- Ruby の予備知識は不要ですが、コマンドラインの扱い方とプログラミング経験は必要です。
- お好きなテキストエディタでプログラミングしていただきます。実行環境はコマンドラインのみです。

予定

- お約束の Hello World!
- 漢字コードについて
- ライブラリの使い方
- コマンドライン引数
- open-uri
- REXML
- イテレータを使う
- クラスとメソッド
- 正規表現
- ファイル入出力
- 例外処理

お約束: Hello World!

```
puts "Hello World!"
```

ご挨拶：Hello ○○ さん！

```
print "お名前を入力してリターンを押してください："
your_name = gets.chomp
puts "#{your_name}さん、こんにちは！"
```

変数のスコープ

- ローカル変数
  - 変数は小文字の英字ではじめます

```
the_local_variable
```

- インスタンス変数
  - インスタンス変数は、@ ではじめます

```
@the_instance_variable
```

- クラス変数
  - クラス変数は、@@ ではじめます

```
@@the_class_variable
```

- グローバル変数
  - グローバル変数は、$ ではじめます

```
$the_global_variable
```

ローカル変数とブロック

- ブロック内で定義されたローカル変数は、ブロック内のみで有効
- ブロックの前に定義されたローカル変数は、ブロック内で参照・書き換え可能
- ただし、Ruby 1.9 ではブロックの前に定義されたローカル変数は、ブロック内からはアクセス不可

Ruby1.8 の漢字コードの扱い①

- Ruby1.8 で扱える日本語文字コード
  - EUC
  - Shift-JIS
  - UTF-8
- プログラム自体のエンコードは一番最初の行に書く
  - `$KCODE='shift-jis'`
  - 実は大文字小文字関係なく最初の一文字で決まる

Ruby1.8 の漢字コードの扱い②

- 漢字変換ライブラリ "Kconv"
- Unix 系のライブラリ nkf (nihongo kanji filter) のラッパー
- String クラスに変換用メソッドを追加
  - `tosjis`
  - `toeuc`
  - `toutf8`
  - `tojis,toutf16`

ライブラリの使い方

- `require 'xxx/xxxxxx'`
- ライブラリの検索パス環境変数 `RUBYLIB`
- プログラム内では `$:`
- サブディレクトリは `/` で表示
- モジュールは `include` で読み込む

コマンドライン引数①

- コマンドライン引数は配列 `ARGV` に格納
- `ARGV` の添え字は `0` から始まる
- プログラム名自体は `$0` で表せる
- ファイル名のみなら `ARGF` が使える
- 便利なライブラリ: `optparse`(OptionParser)

コマンドライン引数②

```
if ARGV.size > 0
  puts "#{ARGV[0]}さん、こんにちは！"
else
  print "お名前を入力し、リターンを押してください："
  puts "#{gets.chomp}さん、こんにちは！"
end
```

OptionParser

```
require 'optparse'
opts = OptionParser.new
opts.on("-k", "--keyword KEYWORD", String)
        {|val| puts "-k #{val}"}
opts.on("-h", "--help") {|val| puts opts.to_s}
rest = opts.parse(ARGV)
```

open-uri

- http/ftp サーバ上のファイルを通常のファイルのように開けるライブラリです。

```
uri = "http://ruby-lang.org/"
response = open(uri).read
puts response
```

価格 .com の Web サービス①

- 商品検索 API
  - http://api.kakaku.com/Ver1/ItemSearch.asp? パラメータ名 = パラメータ値・・・
  - パラメータ例
    - Keyword
    - ResultSet
    - CategoryGroup
    - SortOrder
    - PageNum

価格 .com の Web サービス②

NumOfResult	ヒットしたアイテムの数　0件の場合はエラーコード”ItemNotFound”を返します。	Integer	miniとmedium
ProductID	プロダクトID	String	miniとmedium
ProductName	製品名	String	miniとmedium
MakerName	メーカー名	String	miniとmedium
CategoryName	カテゴリ名 (トップカテゴリ＞サブカテゴリ)	String	mediumのみ
PvRanking	カテゴリ内での人気ランキング。 ランク外の場合は””(空文字)を返します。	Integer	mediumのみ
ImageUrl	イメージのURL	String	mediumのみ
ItemPageUrl	アイテムビューへのURL	String	miniとmedium
BbsPageUrl	クチコミ掲示板へのURL	String	miniとmedium
ReviewPageUrl	レビューページへのURL	String	miniとmedium
LowestPrice	最安価格(税込み)。 価格登録の無い場合は””(空文字)を返します。	Integer	miniとmedium
NumOfBbs	クチコミ掲示板書込み数	Integer	miniとmedium

価格 .com の商品検索

```
require 'open-uri'
require 'cgi'
require 'kconv'
if ARGV.size > 0
  keyword = ARGV[0]
else
  print "検索キーワードを入力:"
  keyword = gets.chomp
end
uri = "http://api.kakaku.com/Ver1/ItemSearch.
   asp?Keyword=#{CGI.escape(keyword.tosjis)}&R
   esultset=medium"
response = open(uri).read
puts response
```

REXML①

- XML 処理用のライブラリです
- XML のパースと生成が可能です
- DOM を使った操作と、SAX での操作が可能

REXML②

- XML をパースしてエレメントを取得

```
require 'rexml/document'
string = <<-EOF_XML
<root>
  <foo>Foo
    <bar>Bar</bar>
  </foo>
  <foo>Foo2</foo>
</root>
EOF_XML
doc = REXML::Document.new(string)
puts doc.root.elements['foo/bar'].get_text
```

イテレータを使う

- Ruby で繰り返しを扱うなら、イテレータが定石

```
(1..10).each do |i|
  puts i**2
end
```

REXML::XPath でイテレータを使う

```
require 'rexml/document'
...

response = open(uri).read
doc = REXML::Document.new(response)
REXML::XPath.each(doc, "/ProductInfo/Item") do
   |item|
  puts item.elements["ProductName"].get_text
end
```

クラスとメソッド

- Ruby のクラス名は大文字で始まる「定数」
- メソッド定義は def ... end
- コンストラクタ（初期化メソッド）は initialize

```
class Item
  def initialize(name,price)
    @name = name
    @price = price
  end
end
```

レスポンスをオブジェクト化

```
class Item
  attr_reader :name, :price

  def initialize(name,price)
    @name = name
    @price = price
  end
end
...
$items = []
response = open(uri).read
doc = REXML::Document.new(response)
REXML::XPath.each(doc, "/ProductInfo/Item") do |i|
  $items << Item.new(
            i.elements["ProductName"].get_text,
            i.elements["LowestPrice"].get_text.to_s.to_i
            )
end
```

メソッドを追加

```
class Item
  attr_reader :name, :price

  def initialize(name,price)
    @name = name
    @price = price
  end

  def to_csv
    '"' + @name.to_s + '",' + @price.to_s
  end

  def <=>(other)
    self.price <=> other.price
  end
end
```

正規表現①

- 例: IP アドレス

```
reg_ip = /(\d{1,3}).(\d{1,3}).(\d{1,3}).(\d{1,3})/
"172.20.1.254" =~ reg_ip
$& => "172.20.1.254"
$1 = "172"
$2 = "20"
$3 = "1"
$4 = "254"
```

あるいは

```
m = reg_ip.match("172.20.1.254")
m[1] = "172"
...
```

正規表現②

- 便利な機能
  - `String#gsub(pattern, replace)`
    - 文字列を検索し、正規表現 pattern にマッチした部分を replace で置き換え
  - `String#scan(pattern)`
    - 文字列を検索し、正規表現 pattern にマッチした部分を配列にして返す
  - いずれも引数にブロックをとれる
    - `String#gsub(pattern) {|m| ..}`
    - `String#scan(pattern) {|m| ..}`

ファイル入出力

- File クラスを使います
- File クラスは IO クラスのサブクラスです
- File.open メソッドはブロックを引数にとることができます

```
File.open("sample.txt","r") do |f|
  while line = gets
    puts line
  end
end
```

例外処理

```
begin
   例外を捕捉したい処理
    raise  明示的に発生させる例外
rescue  捕捉したいエラータイプ  => e
   e を使ってメッセージ処理等
ensure
   例外の発生の有無にかかわらず実行したい処理
end
```

まとめ

- Ruby を使ったスクリプトプログラミングを学習しました
- よく使われる処理についていくつかを紹介しました
- Ruby はプログラミングが楽しくなる言語です
- 便利に使いましょう