Ruby初級者向けレッスン第 18 回 演習問題解答例

okkez @ Ruby 関西

2008 年 03 月 15 日

演習問題解答例

今回の解答例の方針は、見やすい出力を出すのではなく知りたい情報を素早く得ることにしました。

テキスト解析

ある英文を解析してみましょう。

- 文字数
- 単語数
- 行数
- 文字別の出現頻度
- 単語別出現頻度

解答例

仕様について補足ですが、文字数、単語数、行数は wc コマンドの出力と一致すれば OK です。

```
01: #!/usr/bin/ruby
02:
03: str = File.read('text.txt')
04:
05: p str.scan(/\n/).size  # lines
06: p str.scan(/\s+/).size # words
07: p str.size             # bytes
08: p str.split(//).inject(Hash.new{|hash,key| hash[key] = 0 }){|ret, v| ret[v] += 1; ret}
09: p str.split(/\s+/).inject(Hash.new{|hash,key| hash[key] = 0 }){|ret, v| ret[v] += 1; ret}
10:
```

あまりテキストのサイズが大きくない場合は、一度にファイルを読み込んで文字列化してから処理するのがシンプルで良いと思います。サイズが大きい場合は、行ごとに処理するなどの工夫を行うと消費メモリを抑えることができることがあります。

細かい解説は、各メソッドのマニュアルを参照してください。

ブロックなしの Hash.new とブロック付きの Hash.new には以下のような違いがあります。

```
h1 = Hash.new([])
h1[:a] << :a << :b   #=> [:a, :b]
h1[:b] << :a << :b   #=> [:a, :b, :a, :b]
h2 = Hash.new{|h, k| h[k] = [] }
h2[:a] << :a << :b   #=> [:a, :b]
h2[:b] << :a << :b   #=> [:a, :b]
```

この様に、初期値のオブジェクトを破壊的に変更した場合に違いがあります。しかし、今回の解答例では初期値は Fixnum で immutable なので、どちらを使用しても問題ありません。

ログ解析

Apache のログを解析してみましょう。

- '/index.php' へのアクセスは何回？
- 最初の Mac ユーザのアクセスはいつ？
- Google 経由のアクセスは何回？
- 何曜日のアクセスが一番多い？
- ブラウザごとのアクセス数ランキングは？

解答例

```
01: #!/usr/bin/ruby
02:
03: require 'date'
04:
05: class Log
06:   attr_reader :host, :identity, :user, :timestamp,
07:   :request, :response, :size, :referer, :user_agent
08:
09:   def initialize(line)
10:     @host, @identity, @user, @timestamp, @request, @response, @size, @referer, @user_agent =
11:       *line.scan(/^([0-9:.]+) (.+?) (.+?) \[(.+?)\] "(.+?)" ([0-9]+?) ([0-9]+) "(.+?)" "(.+?)
12:   end
13: end
14:
```

```
15: logs = File.readlines('access.log').map{|line| Log.new(line.chomp) }
16:
17: p logs.select{|log| /\/index\.php/ =~ log.request }.size
18: p logs.detect{|log| /Macintosh/ =~ log.user_agent }.timestamp
19: p logs.select{|log| /google/ =~ log.referer }.size
20: p logs.reject{|log|
21:   log.timestamp.nil? or log.timestamp.empty?
22: }.inject(Hash.new{|h, k| h[k] = 0 }){|ret, log|
23:   ret[Date.strptime(log.timestamp, '%d/%b/%Y:%H:%M:%S %z').strftime('%a')] += 1; ret
24: }.to_a.max{|a, b| a[1] <=> b[1] }
25: p logs.inject(Hash.new{|h, k| h[k] = 0 }){|ret, log|
26:   ret[log.user_agent] += 1; ret
27: }.sort_by{|a| -a[1] }
```

ログの様な特定のフォーマットに従った文字列を解析する場合はクラスを作成すると、拡張性と可読性が上がるのでおすすめです。

この解答例は最初に全部のデータを読み込んで、Log クラスのインスタンスにしています。こうすると、データ量が多くなると処理速度が遅くなりますが、あまりややこしいことを考えずに最初に作成したコレクションを処理していけばよいだけなので楽です。

処理速度が求められる場合は以下の様にするとよいでしょう。(まだまだ工夫する余地はありますが、この程度でも少しは速くなります。)

```
01: #!/usr/bin/ruby
02:
03: require 'date'
04:
05: access_count_on_index = 0
06: first_access_time_by_mac = nil
07: access_count_from_google = 0
08: access_count_by_day = Hash.new(0)
09: access_count_by_user_agent = Hash.new(0)
10:
11: File.foreach('access.log') {|line|
12:   host, identity, user, time, request, response, size, referer, user_agent =
13:     *line.scan(/^([0-9:.]+) (.+?) (.+?) \[(.+?)\] "(.+?)" ([0-9]+?) ([0-9]+) "(.+?)" "(.+?)"/
14:   access_count_on_index += 1 if /\/index\.php/ =~ request
15:   first_access_time_by_mac ||= time if /Macintosh/ =~ user_agent
16:   time = Date.strptime(time, '%d/%b/%Y:%H:%M:%S %z') if time
17:   access_count_from_google += 1 if /google/ =~ referer
18:   access_count_by_day[time.strftime('%a')] += 1 if time
19:   access_count_by_user_agent[user_agent] += 1
20: }
21:
22: p access_count_on_index
```

```
23: p first_access_time_by_mac
24: p access_count_from_google
25: p access_count_by_day.max_by{|a| a[1] }
26: p access_count_by_user_agent.sort_by{|a| -a[1] }
```

どちらの解答例でも使用しているので String#scan のマニュアルはよく読んでおいてください。

簡単なベンチマーク。

```
-            user   system total
log.rb (1.8) 9.28s  0.27s  9.900
log2.rb(1.8) 9.07s  0.26s  9.627
log.rb (1.9) 6.98s  0.04s  7.849
log2.rb(1.9) 5.28s  0.04s  6.720
```