Better CSV processing with Ruby 2.6

Kouhei Sutou/Kazuma Furuhashi

ClearCode Inc./Speee, Inc.

RubyKaigi 2019
2019-04-19

Ad: Silver sponsor

Silver Sponsors

ClearCode Inc.

https://www.clear-code.com/

Free software is important in ClearCode. We develop/support software with our free software development experiences. We feed back our business experiences to free software.

Ad: Cafe sponsor

Cafe Sponsor

Speee, Inc.

https://speee.jp/

Our mission is to " Think through every problem with depth and clarity. To bring the future a little forward.". We solve various issues in our society with technology. We have our core competence in Digital Consulting field. We have more businesses in various fields such as Online Media business, Medical business, Global business, and etc. We invited Mr. Yukihiro Matsumoto and Mr. Goro Fuji as engineering advisers. And, Mr. Kenta Murata, who is Ruby Committer, has joined. We aim to bring the future a little forward by developing businesses using technology.

Kouhei Sutou

- ✓ The president of ClearCode Inc.
  クリアコードの社長
- ✓ A new maintainer of the csv library
  csvライブラリーの新メンテナー
- ✓ The founder of Red Data Tools project
  Red Data Toolsプロジェクトの立ち上げ人
  - ✓ Provides data processing tools for Ruby
    Ruby用のデータ処理ツールを提供するプロジェクト

Kazuma Furuhashi

- ✓ A member of Asakusa.rb / Red Data Tools
  Asakusa.rb/Red Data Toolsメンバー
- ✓ Worikng at Speee Inc.
  Speeeで働いている

csv in Ruby 2.6 (1)

Ruby 2.6のcsv (1)

Faster CSV parsing

CSVパースの高速化

Unquoted CSV

クォートなしのCSV

```
AAAAA,AAAAA,AAAAA
...
```

2.5	2.6	Faster?
432.0i/s	764.9i/s	1.77x

Quoted CSV

クォートありのCSV

```
"AAAAA","AAAAA","AAAAA"
...
```

2.5	2.6	Faster?
274.1i/s	534.5i/s	1.95x

Quoted separator CSV (1)

区切り文字をクォートしているCSV (1)

```
",AAAAA",",AAAAA",",AAAAA"
...
```

2.5	2.6	Faster?
211.0i/s	330.0/s	1.56x

Quoted separator CSV (2)

区切り文字をクォートしているCSV (2)

```
"AAAAA\r\n","AAAAA\r\n","AAAAA\r\n"
...
```

2.5	2.6	Faster?
118.7i/s	325.6/s	2.74x

Quoted CSVs

クォートありのCSV

	2.5	2.6
Just quoted	274.1i/s	554.5i/s
Include sep1	211.0i/s	330.0i/s
Include sep2	118.0i/s	325.6i/s
(Note)	(Slow down)	(Still fast)

Note: "Just quoted" on 2.6 is optimized

Multibyte CSV

マルチバイトのCSV

```
あああああ,あああああ,あああああ
...
```

2.5	2.6	Faster?
371.2i/s	626.6i/s	1.69x

csv in Ruby 2.6 (2)

Ruby 2.6のcsv (2)

Faster CSV writing

CSV書き出しの高速化

CSV.generate_line

```
n_rows.times do
  CSV.generate_line(fields)
end
```

2.5	2.6	Faster?
284.4i/s	684.2i/s	2.41x

CSV#<<

```
output = StringIO.new
csv = CSV.new(output)
n_rows.times {csv << fields}
```

2.5	2.6	Faster?
2891.4i/s	4824.1i/s	1.67x

CSV.generate_line vs. CSV#<<

	2.5	2.6
generate_line	284.4i/s	684.2i/s
<<	2891.4i/s	4824.1i/s

Use << for multiple writes

複数行書き出すときは<<を使うこと

csv in Ruby 2.6 (3)

Ruby 2.6のcsv (3)

New CSV parser

for

further improvements

さらなる改良のための新しいCSVパーサー

Benchmark with KEN_ALL.CSV

KEN_ALL.CSVでのベンチマーク

```
01101,"060  ","0600000","ﾎｯｶｲﾄﾞｳ","ｻｯﾎﾟﾛｼﾁｭｳｵｳｸ",...
...(124257 lines)...
47382,"90718","9071801","ｵｷﾅﾜｹﾝ","ﾔｴﾔﾏｸﾞﾝﾖﾅｸﾞﾆﾁｮｳ",...
```

Zip code data in Japan

日本の郵便番号データ

https://www.post.japanpost.jp/zipcode/download.html

KEN_ALL.CSV statistics

KEN_ALL.CSVの統計情報

Size（サイズ）	11.7MiB
# of columns（列数）	15
# of rows（行数）	124259
Encoding（エンコーディング）	CP932

Parsing KEN_ALL.CSV

KEN_ALL.CSVのパース

```
CSV.foreach("KEN_ALL.CSV",
            "r:cp932") do |row|
end
```

2.5	2.6	Faster?
1.17s	0.79s	1.48x

Fastest parsing in pure Ruby

Ruby実装での最速のパース方法

```
input.each_line(chomp: true) do |line|
  line.split(",", -1) do |column|
  end
end
```

Limitation: No quote

制限：クォートがないこと

KEN_ALL.CSV without quote

クォートなしのKEN_ALL.CSV

```
01101,060  ,0600000,ﾎｯｶｲﾄﾞｳ,ｻｯﾎﾟﾛｼﾁｭｳｵｳｸ,...
...(124257 lines)...
47382,90718,9071801,ｵｷﾅﾜｹﾝ,ﾔｴﾔﾏｸﾞﾝﾖﾅｸﾞﾆﾁｮｳ,...
```

Optimized no quote CSV parsing

最適化したクォートなしCSVのパース方法

```
CSV.foreach("KEN_ALL_NO_QUOTE.CSV",
            "r:cp932",
            quote_char: nil) {|row|}
```

split	2.6	Faster?
0.32s	0.37s	0.86x (almost the same/同等)

Summary: Performance
まとめ：性能

- ✓ Parsing: 1.5x-3x faster
  パース：1.5x-3x高速
  - ✓ Max to the "split" level by using an option
    オプションを指定すると最大で「split」レベルまで高速化可能
- ✓ Writing: 1.5x-2.5x faster
  書き出し：1.5x-2.5x高速
  - ✓ Use CSV#<< than CSV.generate_line
    CSV.generate_lineよりもCSV#<<を使うこと

How to improve performance (1)

速度改善方法（1）

Complex quote

複雑なクォート

Complex quote
複雑なクォート

```
"AA""AAA"
"AA,AAA"
"AA\rAAA"
"AA\nAAA"
```

Use StringScanner

StringScannerを使う

- ✓ String#split is very fast
  String#splitは高速
- ✓ String#split is naive for complex quote
  String#splitは複雑なクォートを処理するには単純過ぎる

2.5 uses String#split

```
in_extended_column = false # "...\n..." case
@input.each_line do |line|
  line.split(",", -1).each do |part|
    if in_extended_column
      # ...
    elsif part.start_with?('"')
      if part.end_with?('"')
        row << pars.gsub('""', '"') # "...""..." case
      else
        in_extended_column = true
      end
    # ...
```

split: Complex quote

Just quoted	274.1i/s
Include sep1	211.0i/s
Include sep2	118.0i/s

Slow down

遅くなる

2.6 uses StringScanner

```
row = []
until @scanner.eos?
  value = parse_column_value
  if @scanner.scan(/,/)
    row << value
  elsif @scanner.scan(/\n/)
    row << value
    yield(row)
    row = []
  end
end
```

parse_column_value

```
def parse_column_value
  parse_unquoted_column_value ||
    parse_quoted_column_value
end
```

Compositable components

部品を組み合わせられる

parse_unquoted_column_value

```ruby
def parse_unquoted_column_value
  @scanner.scan(/[^,"\r\n]+/)
end
```

parse_quoted_column_value

```ruby
def parse_quoted_column_value
  # Not quoted
  return nil unless @scanner.scan(/"/)
  # Quoted case ...
end
```

Parse methods can be composited

パースメソッドを組み合わせられる

```
def parse_column_value
  parse_unquoted_column_value ||
    parse_quoted_column_value
end
```

Easy to maintain

メンテナンスしやすい

Point (1)
ポイント（1）

- ✓ Use StringScanner for complex case
  複雑なケースにはStringScannerを使う
- ✓ StringScanner for complex case:
  複雑なケースにStringScannerを使うと：
  - ✓ Easy to maintain
    メンテナンスしやすい
  - ✓ No performance regression
    性能が劣化しない

StringScanner: Complex quote

Just quoted	554.5i/s
Include sep1	330.0i/s
Include sep2	325.6i/s

No slow down...?

遅くなっていない。。。？

How to improve performance (2)

速度改善方法（2）

Simple case

単純なケース

Simple case
単純なケース

```
AAAAA
"AAAAA"
```

Use String#split

String#splitを使う

StringScanner is slow for simple case

StringScannerは単純なケースでは遅い

Fallback to StringScanner impl.

StringScanner実装にフォールバック

```
def parse_by_strip(&block)
  @input.each_line do |line|
    if complex?(line)
      return parse_by_string_scanner(&block)
    else
      yield(line.split(","))
    end
  end
end
```

Quoted CSVs

クォートありのCSV

	StringScanner	split + StringScanner
Just quoted	311.7i/s	**523.4i/s**
Include sep1	312.9i/s	309.8i/s
Include sep2	311.3i/s	312.6i/s

Point (2)
ポイント（2）

- ✓ First try optimized version
  まず最適化バージョンを試す
- ✓ Fallback to robust version when complexity is detected
  複雑だとわかったらちゃんとしたバージョンにフォールバック

How to improve performance (3)

速度改善方法（3）

```
loop do
```

↓

```
while true
```

loop vs. while

How	Throughput
loop	377i/s
while	401i/s

Point (3)
ポイント（3）

- ✓ while doesn't create a new scope
  whileは新しいスコープを作らない
- ✓ Normally, you can use loop
  ふつうはloopでよい
  - ✓ Normally, loop isn't a bottle neck
    ふつうはloopがボトルネックにはならない

How to improve performance (4)

速度改善方法（4）

Lazy

遅延

CSV object is parser and writer

CSVオブジェクトは読み書きできる

- ✓ 2.5: Always initializes everything
  2.5：常にすべてを初期化
- ✓ 2.6: Initializes when it's needed
  2.6：必要になったら初期化

Write performance

	2.5	2.6	Faster?
generate_line	284.4i/s	684.2i/s	2.41x
<<	2891.4i/s	4824.1i/s	1.67x

How to initialize lazily

初期化を遅延する方法

```
def parser
  @parser ||= Parser.new(...)
end

def writer
  @writer ||= Writer.new(...)
end
```

Point (4)
ポイント（4）

- ✓ Do only needed things
  必要なことだけする
- ✓ One class for one feature
  機能ごとにクラスを分ける

New features by new parser
新しいパーサーによる新機能

- ✓ Add support for \" escape
  ¥"でのエスケープをサポート
- ✓ Add strip: option
  strip:オプションを追加

\" escape

¥"でのエスケープ

```
CSV.parse(%Q["a""bc","a\\"bc"],
          liberal_parsing: {backslash_quote: true})
# [["a\\"bc", "a\\"bc"]]
```

strip:

strip:

```
CSV.parse(%Q[ abc  ,  " abc"], strip: true)
# [["abc", " abc"]]
CSV.parse(%Q[abca,abc], strip: "a")
# [["bc", "bc"]]
```

csv in Ruby 2.6 (4)

Ruby 2.6のcsv (4)

Keep backward compatibility

互換性を維持

How to keep backward compat.

互換性を維持する方法

- ✓ Reconstruct test suites
  テストを整理
- ✓ Add benchmark suites
  ベンチマークを追加

Test
テスト

- ✓ Important to detect incompat.
  非互換の検出のために重要
- ✓ Must be easy to maintain
  メンテナンスしやすくしておくべき
  - ✓ To keep developing
    継続的な開発するため

Easy to maintenance
メンテナンスしやすい状態

- ✓ Easy to understand each test
  各テストを理解しやすい
- ✓ Easy to run each test
  各テストを個別に実行しやすい
  - ✓ Focusing a failed case is easy to debug
    失敗したケースに集中できるとデバッグしやすい

Benchmark
ベンチマーク

- ✓ Important to detect performance regression **bugs**
  性能劣化**バグ**を検出するために重要

benchmark_driver gem

Fully-featured benchmark driver for Ruby 3x3

Ruby 3x3のために必要な機能が揃っているベンチマークツール

benchmark_driver gem in csv

- ✓ YAML input is easy to use
  YAMLによる入力が便利
- ✓ Can compare multiple gem versions
  複数のgemのバージョンで比較可能
  - ✓ To detect performance regression
    性能劣化を検出するため

Benchmark for each gem version

gemのバージョン毎のベンチマーク

```yaml
loop_count: 100
contexts:
  - gems:
      csv: 3.0.1
  - gems:
      csv: 3.0.2
  - name: "master"
    prelude: |
      $LOAD_PATH.unshift(Fil
      require "csv"
benchmark:
  'not convert': CSV.parse(c
  converter: |-
    CSV.parse(csv_text, converters: convert_nil)
  option: |-
    CSV.parse(csv_text, nil_value: "")
```

```
Comparison:
                    not convert
          csv 3.0.2:    26586.3 i/s
              master:    23977.9 i/s - 1.11x  slower
          csv 3.0.1:    18459.6 i/s - 1.44x  slower

                     converter
          csv 3.0.2:    21271.5 i/s
              master:    20130.3 i/s - 1.06x  slower
          csv 3.0.1:    15970.1 i/s - 1.33x  slower

                      option
          csv 3.0.2:    23933.4 i/s
              master:    22724.8 i/s - 1.05x  slower
          csv 3.0.1:    16606.2 i/s - 1.44x  slower
```

csv/benchmark/

- ✓ convert_nil.yaml
- ✓ parse{,_liberal_parsing}.yaml
- ✓ parse_{quote_char_nil,strip}.yaml
- ✓ read.yaml, shift.yaml, write.yaml

Benchmark as a starting point
出発点としてのベンチマーク

- ✓ Join csv developing!
  csvの開発に参加しよう！
- ✓ Adding a new benchmark is a good start
  ベンチマークの追加から始めるのはどう？
  - ✓ We'll focus on improving performance for benchmark cases
    ベンチマークが整備されているケースの性能改善に注力するよ

How to use improved csv?

改良されたcsvを使う方法

```
gem install csv
```

csv in Ruby 2.5

Ruby 2.5のcsv

Default gemified

デフォルトgem化

Default gem
デフォルトgem

- ✓ Can use it just by require
  requireするだけで使える
- ✓ Can use it without entry in Gemfile
  (But you use it bundled in your Ruby)
  Gemfileに書かなくても使えるけど古い
- ✓ Can upgrade it by gem
  gemでアップグレードできる

How to use improved csv?

改良されたcsvを使う方法

```
gem install csv
```

Future
今後の話

Faster

さらに速く

Improve String#split

String#splitを改良

Accept " "
as normal separator

" "をただの区切り文字として扱う

split(" ") works like awk

split(" ")はawkのように動く

```
" a  b c".split(" ", -1)
# => ["a", "b", "c"]
" a  b c".split(/ /, -1)
# => ["", "a", "", "b", "c"]
```

String#split in csv

csvでのString#split

```
if @column_separator == " "
  line.split(/ /, -1)
else
  line.split(@column_separator, -1)
end
```

split(string) vs. split(regexp)

regexp	string	Faster?
344448i/s	3161117i/s	9.18x

See also [Feauture:15771]

Improve StringScanner#scan

StringScanner#scanを改良

Accept String as pattern

Stringもパターンとして使えるようにする

scan(string) vs. scan(regexp)

regexp	string	Faster?
14712660i/s	18421631i/s	1.25x

See also ruby/strscan#4

Faster KEN_ALL.CSV parsing (1)

より速いKEN_ALL.CSVのパース（1）

	Elapsed
csv	0.791s
FastestCSV	0.141s

Faster KEN_ALL.CSV parsing (2)

より速いKEN_ALL.CSVのパース（2）

	Encoding	Elapsed
csv	CP932	0.791s
FastestCSV	CP932	0.141s
csv	UTF-8	1.345s
FastestCSV	UTF-8	0.713s

Faster KEN_ALL.CSV parsing (3)

より速いKEN_ALL.CSVのパース（3）

	Encoding	Elapsed
FastestCSV	UTF-8	0.713s
Python	UTF-8	0.208s
Apache Arrow	UTF-8	0.145s

Further work
今後の改善案

- ✓ Improve transcoding performance of Ruby
  Rubyのエンコーディング変換処理の高速化
- ✓ Improve simple case parse performance by implementing parser in C
  シンプルなケース用のパーサーをCで実装して高速化
- ✓ Improve perf. of REXML as well as csv
  csvのようにREXMLも高速化

Join us!

一緒に開発しようぜ！

- ✓ Red Data Tools: https://red-data-tools.github.io/
- ✓ RubyData Workshop: Today 14:20-15:30
- ✓ Code Party: Today 19:00-21:00
- ✓ After Hack: Sun. 10:30-17:30

Join us!!

一緒に開発しようぜ！！

- OSS Gate: https://oss-gate.github.io/
  - provides a "gate" to join OSS development
    OSSの開発に参加する「入り口」を提供する取り組み
  - Both ClearCode and Speee are one of sponsors
    クリアコードもSpeeeeもスポンサー
- OSS Gate Fukuoka:
  https://oss-gate-fukuoka.connpass.com/