MCMD Ruby Package Documentation

対応 NYSOL パッケージ: Ver. 1.0〜

revise history:
March 10, 2014 : nysol パッケージへの統合に伴いインストール方法変更
November 2, 2013 : first release

2014 年 3 月 10 日

目次

第 1 章

はじめに

1.1　概要

本ライブラリは、Ruby 上で CSV データを効率的に扱うための関数を提供する。主な関数として、CSV データのシーケンシャルな読み込み関数 (Mcsvin)、および書き込み関数 (Mcsvout)、そして CSV データを表とみなしたセル単位でのランダムアクセスを可能とする関数 (Mtable) がある。いずれの関数も CSV の標準仕様である RFC4180 に概ね準拠しておりカンマや改行を含む文字列を扱うことができる。データへのアクセスは、項目名をキーとする Hash でも可能であるし、項目番号による Array でも可能である。さらに同等の機能を有する他のライブラリよりも効率的である。図 1.1 に、CSV データをコピーするサンプルスクリプトを示す。

```
#!/usr/bin/env ruby
require "mcmd"

# input.csv
# customer,date
# A,20081201
# A,20081202
MCMD::Mcsvin.new("i=input.csv -array"){|iCSV|
  MCMD::Mcsvout.new("o=output.csv f=customer,date"){|oCSV|
    iCSV.each{|flds|
      oCsv.write(flds)
    }
  }
}
```

図 1.1　mcmd ライブラリを用い CSV ファイルをコピーする Ruby スクリプト

CSV データの操作以外にも、NYSOL で提供されるコマンドのいくつかは、本ライブラリを利用した Ruby スクリプトにより実装されている。そこに mcmd と同等のインターフェースや基本機能を持たせるための関数をいくつか用意している。インターフェースとしてはコマンドライン引数を処理するための Margs、そして完了メッセージやエラーメッセージの出力を行う endMsg、errorMsg などの関数がある。また基本機能としては、作業ファイル名の自動生成/削除を行う Mtemp、そして mcmd と同じ環境変数の設定も可能である。

なお、Ruby のバージョンは 1.9.2 以降のみ動作確認している。

1.2 インストール

本パッケージは全て nysol パッケージに含まれている。nysol パッケージをインストールすれば必要なソフトウェアは全てインストールされる。詳しくは nysol パッケージのインストールの説明 (http://www.nysol.jp/install) を参照のこと。

第 2 章

クラス

2.1 Mcsvin CSV の読み込みクラス

CSV データファイルを行単位で処理するためのクラス。以下のような特徴を持つ。

- C++ で実装されており高速に動作する。
- 一行目が項目名行であれば、項目名を key とする Hash にデータを格納できる。
- データの格納は Hash/Array を選択可能 (Array の方が 2 倍高速)。
- キーブレイク処理を容易に扱える。
- RFC4180 にほぼ準拠
- 一行の項目数は固定であることを前提とする。

2.1.1 メソッド

* MCMD::Mcsvin::new(arguments){block}

Mcsvin オブジェクトを生成する。`arguments` に、以下の引数をスペースで区切った文字列として指定する。

`i=`	入力ファイル名 (String) 【必須】
`k=`	キーブレイクを検知する項目リスト。複数項目はカンマで区切る。 キー指定の有無によって each メソッドの yield 引数が異なることに注意する。
`-nfn`	1 行目を項目名と見なさない。
`-array`	each メソッドで各データ項目を Array に格納する。 指定がなければ Hash に格納する。 Array は Hash に比べ、約 2 倍効率的である (詳細は「ベンチマーク」を参照)。
`block`	ブロックがが指定されていれば実行 (yield) する。

* MCMD::Mcsvin::each{|val| block}

* MCMD::Mcsvin::each{|val,top,bot| block}

CSV ファイルを一行ずつ処理する。1) の書式はキー (k=) を指定しなかった場合で、val に値が設定される。2) の書式はキーを指定した場合で、val 以外にもキーブレイク情報 top,bot の変数も設定される。

val	項目名をキーとして値を格納した Hash(もしくは Array)。値は全て String 型で格納。
top	k=で指定したキーの先頭であれば true、さもなければ false がセットされる。詳細は備考を参照。
bot	k=で指定したキーの終端であれば true、さもなければ false がセットされる。詳細は備考を参照。

* MCMD::Mcsvin::names()

項目名配列を返す。`-nfn` が指定されていれば nil を返す。

2.1.2 備考

- `-nfn` が指定された場合、データは Array に格納される。Hash では格納できない。
- `k=`を指定する場合は、そこで指定した項目で並べ替えておかなければならない。
- キーブレイクのロジック:

```
MCMD::Mcsvin.new("i=input.dat k=key"){|csv|
```

```
csv.each{|val,top,bot|
    :
  }
}
```

上記のコードにおいて、bool 型のブロック変数 top および bot の設定ロジックは以下のとおり。
データ行を $i = 1, 2, \cdots, n$、i 行目のキー項目 (“key”) の値を k_i とし、便宜上 $k_0 = k_{n+1} = \phi$ とする。ただし、$k_i (1 \leq i \leq n) \neq \phi$ である。

$$\texttt{top} = \begin{cases} true, & \text{if } k_i \neq k_{i-1} \\ false, & \text{otherwise} \end{cases} \tag{2.1}$$

$$\texttt{bot} = \begin{cases} true, & \text{if } k_i \neq k_{i+1} \\ false, & \text{otherwise} \end{cases} \tag{2.2}$$

2.1.3 利用例

例 1 項目名の出力と行番号・値の出力

```
# dat1.csv
顧客,日付,金額
A,20081201,10
B,20081002,40

MCMD::Mcsvin.new("i=dat1.csv"){|csv|
  puts csv.names.join(",")
  csv.each{|val|
    p val
  }
}
# 出力結果
顧客,日付,金額
["顧客"=>"A", "日付"=>"20081201", "金額"=>"10"]
["顧客"=>"B", "日付"=>"20081002", "金額"=>"40"]
```

例 2 キーブレイク処理

```
# dat1.csv
顧客,日付
A,20081201
A,20081202
B,20081003
C,20081004
C,20081005
C,20081006

csv=MCMD::Mcsvin.new("i=dat1.csv k=顧客")
csv.each{|val,top,bot|
  puts "#{val['顧客']},#{val['日付']} top=#{top} bot=#{bot}"
}
csv.close

# 出力結果
A,20081201 top=true bot=false
A,20081202 top=false bot=true
B,20081003 top=true bot=true
C,20081004 top=true bot=false
```

```
C,20081005 top=false bot=false
C,20081006 top=false bot=true
```

例 3 項目名行のないデータの処理

-nfn を指定すると Array に格納される。

```
# dat1.csv
A,20081201
A,20081202

MCMD::Mcsvin.new("i=dat1.csv k=1 -nfn"){|csv|
  puts csv.names # -> nil
  csv.each{|val|
    p val
  }
}

# 出力結果
nil
["A","20081201"]
["A","20081202"]
```

例 4 Array に格納する例

```
# dat1.csv
顧客,日付,金額
A,20081201,10
B,20081002,40

# -array オプションで Array 格納
MCMD::Mcsvin.new("i=dat1.csv -array"){|csv|
  puts csv.names.join(",")
  csv.each{|val|
    p val
  }
}

# 出力結果
顧客,日付,金額
["A", "20081201", "10"]
["B", "20081002", "40"]
```

関連コマンド

Mcsvout：CSV データの書き込み
Mtable：セル単位読み込み操作

2.1.4 ベンチマークテスト

CSV データの読み込み処理について、Ruby の拡張ライブラリとして提供されている各種ライブラリをベンチマークとして処理速度の比較を行う。ベンチマーク対象は以下の 4 つのライブラリと 1 つのコマンドである。

CSVScan　http://raa.ruby-lang.org/project/csvscan/
LightCsv　http://tmtm.org/ruby/lightcsv/
FasterCSV　http://www.gesource.jp/programming/ruby/database/fastercsv.html

CSV http://www.ruby-lang.org/ja/old-man/html/CSV.html
mcut 項目を切り出す M コマンド (全て C++ で実装)。参考までに掲載。

表 2.1 にベンチマークテストの結果を示す。1 万行〜500 万行のデータについて読み込み実験を行った。図 2.1 には、ベンチマークテストで利用したスクリプトの抜粋が示されている。Mcsvin は CSVScan(C による実装) とほぼ同等性能である。その他はいずれも Ruby ネイティブコードによる実装なのでその差があらわれていることがわかる。ただし、mcut との比較においては、Mcsvin も到底及ばない。mcut と Mcsvin で採用している CSV の parsing ロジックおよびその実装は全く同じであるので、データを Array に格納するなどの Ruby とのインターフェースにまつわるコストによって、ここまでの差が出ていることになる。図 2.1 に、ベンチマークテストで利用したスクリプトの抜粋を示す。

表 2.1 各種 CSV ライブラリの実行速度比較 (単位:秒)

行数	10K	100K	1000K	2000K	3000K	4000K	5000K
Mcsvin	0.020	0.196	1.76	3.51	5.26	7.02	8.79
CSVScan	0.021	0.187	1.83	3.67	5.50	7.33	9.17
LightCsv	0.155	1.62	15.99	–	–	–	–
FasterCSV	0.196	1.96	19.50	–	–	–	–
CSV	1.44	14.3	–	–	–	–	–
mcut	–	–	0.095	0.177	0.260	0.342	0.423

10 回実行した結果の平均値 (real time) を示している。
– は値が大き (小さ) 過ぎるために計測していないことを意味する。
行数 10K は 10000 行の意味。データのサイズは行数が 1000K で約 25M バイト。項目数は 5 つ。
バージョン: CSVScan 0.0.20070920, FasterCSV 1.5.1, LightCsv 0.2.2 CSV(Ruby 1.8.7)
テスト環境: Mac Book Pro, 2.66GHz Intel Core i7, 8GB メモリ, Mac OS X 10.6.8

```
require 'rubygems'
require 'csv'
require 'fastercsv'
require 'lightcsv'
require 'csvscan'
require 'mcmd'

require 'benchmark'

puts Benchmark.measure{
  (0...10).each{|i|
    # Mcsvin の場合
    csv=MCMD::Mcsvin.new("i=data.csv -array"){|csv| csv.each{|val|}}

    # CSVScan の場合
    File.open("data.csv","r"){|fp| CSVScan.scan(fp){|row|}}

    # LightCsv の場合
    LightCsv.foreach("data.csv"){|row|}

    # FasterCSV の場合
    FasterCSV.foreach("data.csv"){|row|}

    # CSV の場合
    CSV.open("data.csv", 'r'){|row|}
  }
}
```

図 2.1 ベンチマークテストのスクリプト (抜粋)

次に、Mcsvin において、キー指定の有無およびデータを格納する型による実行時間の差について見てみる (表 2.2)。キー指定の有無による速度には、さほど大きな違いはないが、データの格納型については、Array が Hash より 2 倍ほど効率的である。

表 2.2　キーの有無とデータ格納型による実行速度比較 (単位:秒)

キー	型	1000K	2000K	3000K	4000K	5000K
なし	Array	1.76	3.51	5.26	7.02	8.79
なし	Hash	3.52	6.99	10.50	14.00	17.52
あり	Array	1.97	3.92	5.88	7.84	9.83
あり	Hash	3.68	7.34	11.01	14.73	18.34

キーのサイズは平均 10 行程度。

2.2 Mcsvout CSV の書き込みクラス

CSV データファイルに出力するためのクラス。以下のような特徴を持つ。

- C++ で実装されており高速に動作する。
- 項目名行あり/なし、いずれの形式も扱うことができる。
- RFC4180 にほぼ準拠
- 一行の項目数は固定であることを前提とする。

2.2.1 メソッド

* MCMD::Mcsvout::new(arguments){block}

Mcsvout オブジェクトを生成する。arguments に、以下の引数をスペースで区切った文字列として指定する。

o=	出力ファイル名 (String)
f=	出力する CSV データのヘッダー (一行目) の項目名文字列を配列で指定する (String Array)。 f=を省略して size=を指定すれば、項目名なしの CSV を出力する。
size=	項目名を出力しない時には、CSV 項目の数 (Fixnum) を指定する。
precision=	Float 型変数の有効桁数を指定する。デフォルトは 10 桁。 C 言語の出力書式"%.ng"の n の値。 100/3 に対して有効桁数 5 桁であれば 33.333、2 桁であれば 33 となる。
bool=	true と false の出力値をカンマで区切って指定する。デフォルトは”1,0”

* MCMD::Mcsvout::write(values)

values 配列 (Array) に格納された値を CSV データとして出力する。配列に格納できるデータクラスは String, Fixnum, Bignum, Float, nil, true, false である。それ以外のクラスは全て nil として扱われる。配列のサイズが項目名の数より少い場合は null 値が追加出力される。配列のサイズが項目名の数より多い場合は、超過分は出力されない。

2.2.2 備考

- 文字列にカンマが含まれていれば、その値は自動的にダブルクオーテーションで囲われ出力される。文字列にダブルクオーテーションが含まれていれば、連続する二つのダブルクオーテーションに変換される。

2.2.3 利用例

例 1 項目名行ありの CSV データ出力例

```
csv=MCMD::Mcsvout.new("i=rsl.csv f=a,b,c"){|csv|
  csv.write(["1",2,3.4])
  csv.write([1,2,3,4,5])
  csv.write([1,2])
}

# 出力結果 (rsl.csv)
a,b,c
1,2,3.4
```

```
1,2,3
1,2,
```

例 2 項目名行なしの CSV データ出力例

```
csv=MCMD::Mcsvout.new("i=rsl.csv size=3"){|csv|
  csv.write(["1",2,3.4])
  csv.write([true,nil,false])
  csv.write(["4\"5","","6,7"])
}

# 出力結果 (rsl.csv)
1,2,3.4
1,,0
"4""5",,"6,7"
```

例 3 オプション指定 (有効桁数,bool 値)

```
MCMD::Mcsvout.new("i=rsl.csv size=3 precision=3 bools=T,F"){|csv|
  csv.write([0.123456,123456.0]) # 有効桁数の指定は小数点以下の桁数でないことに注意する
  csv.write([123456,0])  # 有効桁数の指定は Fixnum には影響しない
  csv.write([true,false])
}

# 出力結果 (rsl.csv)
0.123,1.23e+05
123456,0
T,F
```

例 4 データコピー

```
# dat1.csv
顧客,日付
A,20081201
B,20081002

MCMD::Mcsvin.new("i=dat1.csv -array"){|csvIn|
  MCMD::Mcsvout.new("i=rsl.csv f=#{csvIn.names.join(",")}){|csvOut|
    csvIn.each{|val|
      csvOut.write(val)
    }
  }
}

# rsl.csv
顧客,日付
A,20081201
B,20081002
```

2.2.4 ベンチマークテスト

CSV データの書き込み処理について、Ruby の拡張ライブラリとして提供されている各種ライブラリをベンチマークとして処理速度の比較を行う。ベンチマーク対象は以下の 2 つのライブラリである。

FasterCSV　http://www.gesource.jp/programming/ruby/database/fastercsv.html
CSV　http://www.ruby-lang.org/ja/old-man/html/CSV.html

表 2.3 にベンチマークテストの結果を示す。1 万行,10 万行,100 万行のデータについて書き込み実験を行った。ただし、実ファイルとしては出力せず、null デバイス (/dev/null) への書き込みとした。図 2.2 には、ベンチマークテストで利用したスクリプトの抜粋が示されている。Mcsvout は C++ による実装のため、他の二つのライブラリより高速である。Ruby ネイティブコードによる実装との差があらわれているのであろう。

表 2.3 各種 CSV ライブラリの実行速度比較 (単位:秒)

行数	10K	100K	1000K
Mcsvout	0.0158	0.150	1.50
FasterCSV	0.232	1.90	20.0
CSV	0.279	2.80	27.9

10 回実行した結果の平均値 (real time) を示している。
行数 10K は 10000 行の意味。String, Fixnum, Float, true, false, nil の 6 つの値を出力。
バージョン: FasterCSV 1.5.1 CSV(Ruby 1.8.7)
テスト環境: Mac Book Pro, 2.66GHz Intel Core i7, 8GB メモリ, Mac OS X 10.6.8

```
require 'rubygems'
require 'csv'
require 'fastercsv'
require 'mtools'

require 'benchmark'

$data = ["12345678", 10, 1.1, true, nil, false]

puts Benchmark.measure{
  (0...10).each{|i|
    # Mcsvout の場合
    MCMD::Mcsvout.new("o=/dev/null size=6){|csv|
      (0...10000).each{|j|
        csv.write($data)
      }
    }

    # FasterCSV の場合
    FasterCSV.open("/dev/null", 'w'){|csv|
      (0...10000).each{|j|
        csv << $data
      }
    }

    # CSV の場合
    CSV.open("/dev/null", 'w'){|csv|
      (0...10000).each{|j|
        csv << $data
      }
    }
  }
}
```

図 2.2 ベンチマークテストのスクリプト (抜粋)

2.2.5 関連コマンド

Mcsvin : CSV データの読み込み

2.3 Margs 引数操作クラス

コマンドライン引数を扱うクラス。以下のような特徴を持つ。

- keyword=value および -keyword の二つのフォーマットの引数を扱う。
- -keyword はオプションで、Bool 型 (true/false) に変換される。
- value の型としては、Ruby の原型として、String 配列、Fixnum 配列、Float 配列を扱うことができる。
- その他の特殊な型として、ファイル型や項目名型を提供する。
- デフォルト値や値の範囲を指定することができる。
- 指定が正しくなければエラーメッセージを表示して終了する。
- --help が指定されれば help() を呼び出し終了する。またヘルプ関数名を指定することも可能。--help は、一般のオプションとは違い、マイナス記号が 2 つであることに注意する。

2.3.1 メソッド

* MCMD::Margs.new(ARGV[,allKeyWords][,mandatoryKeyWords][,helpFunction)

Margs オブジェクトを生成する。「keyword=値」もしくは「-keyword」の形式で与えられたコマンドライン引数が、クラス内部の Hash もしくは Array 変数にセットされる。

ARGV Ruby の ARGV 変数。

allKeyWords key=value もしくは-key による引数キーワードリスト。ここで指定した以外の引数が ARGV に指定されていないことをチェックし、指定されていればエラー終了してくれる。allKeyWords を省略した場合はこのチェックをしない。

mandatoryKeyWords key=value による引数キーワードリスト。ここで指定した引数がコマンドラインで指定されていなければエラー終了してくれる。mandatoryKeyWords を省略した場合はこのチェックをしない。

helpFunction --help が指定されたときに呼び出される関数名。

```
# コマンドライン
$ ruby test.rb i=dat.csv -abc

# test.rb の内容
args=Margs.new(ARGV, "i=,v=,-abc") # OK
args=Margs.new(ARGV, "i=,v=") # -abc は指定できないのでエラー終了
args=Margs.new(ARGV, "i=,v=,-abc","i=,v=") # v=は必須だが指定されてないのでエラー終了
```

* MCMD::Margs.file(keyWord,mode): ファイル名の取得

keyWord key=形式のキーワード (String)

mode "r"(readable のチェック)、もしくは"w"(writable のチェック) を指定する。(String)

keyword で指定された値を入力ファイル名とみなし、mode が"r"の場合、そのファイルが読み取り可能であればそのファイル名を返す。読み取りが可能でなければエラー終了する。mode が"w"の場合は、書き込みを行うディレクトリに書き込み可能かどうかチェックし、書き込み可能であればそのファイル名を返す。書き込み可能でなければエラー終了する。

```
# コマンドライン
$ ruby test.rb i=dat.csv

# test.rb の内容
args=Margs.new(ARGV)
```

```
puts args.file("i=","r") # dat.csv が readable であれば"dat.csv"
puts args.file("i=","w") # カレントディレクトリが writable であれば"dat.csv"
```

* MCMD::Margs.field(keyWord,fileName)

keyWord で指定した項目名について、fileName で指定されたファイルの項目と関連付けて各種情報を返す。

keyWord "key="形式のキーワード (String)。
fileName ファイル名。

コマンドラインでの項目名の指定は以下のフォーマットに従わなければならない。

$$\texttt{key=}name_1\texttt{[:}newName_1\texttt{\%}option_1\texttt{]},name_2\texttt{[:}newName_2\texttt{\%}option_2\texttt{]},\ldots \tag{2.3}$$

複数の項目名はカンマで区切って指定する。$name_i$ は、fileName で指定された CSV ファイルの項目名でなければならない。さもなければ"field name not found"のエラーにて終了する。

項目名 $name_i$ には二つの属性 $newName_i$ と $option_i$ を指定できる (省略可)。それぞれ用途は自由であるが、:と % で区切らなければならない。

一般的な用途としては、ある項目 $name_i$ に対する演算結果を新しい項目名 $newName_i$ として追加出力することを想定している。そして $option_i$ は処理内容を制御するオプションとして利用する。

このメソッドは、以下に示す各種情報を Hash で返す。太字は Hash キー。

names 項目名 names の配列 (String Array)。
newNames 新項目名 newNames の配列 (String Array)。指定がなければ nil。
options オプション options の配列 (String Array)。指定がなければ nil。
fld2csv "key="で指定された項目に対応する CSV ファイル (fileName) の項目番号 (0 から始まる)(String Array)。
csv2fld CSV ファイルの項目番号を要素番号とした"key="で指定された項目の指定順序番号 (0 から始まる)(String Array)。指定のない項目は nil。

```
# test.csv の内容
fld1,fld2,fld3
1,2,3
4,5,6

# コマンドライン
$ ruby test.rb f=fld1,fld3

# test.rb の内容
args=Margs.new(ARGV)
fld=args.field("f=","test.csv")
p fld["names"]   # -> ["fld1","fld3"]
p fld["fld2csv"] # -> [0,2] fld1,fld3 は test.csv の 0 番目と 2 番目の項目に対応
p fld["csv2fld"] # -> [0,nil,1] test.csv の 0 番目の項目は f=の 0 番目に指定された

# コマンドライン
$ ruby test.rb f=fld3:newFld3%n,fld2%nr

# test.rb の内容
args=Margs.new(ARGV)
fld=args.field("f=","test.csv")
p fld["names"]    # -> ["fld3", "fld2"]
p fld["newNames"] # -> ["newFld3", nil]
p fld["options"]  # -> ["n", "nr"]
p fld["fld2csv"]  # -> [2, 1]
p fld["csv2fld"]  # -> [nil, 1, 0]
```

* MCMD::Margs.str(keyWord[,default][,token1][,token2])

文字列引数の取得

keyWord "key="形式のキーワード (String)
default 指定がなかったときのデフォルト値。省略時は nil。
token1 複数の文字列を指定する場合の区切り文字。省略時は区切りは無いものと見なす。
token2 token1 で切り出された文字列をさらに token2 を区切りとする文字列で区切る。省略時は区切りは無いものと見なす。

コマンドライン上で指定された引数のうち、keyWord とマッチする値を文字列として返す。コマンドラインで指定されていなければ default の文字列を返す。この時 default が nil であれば nil を返す。

token1 が指定されていれば区切られた文字列を要素とする配列を返す。さらに token2 が指定されていれば、配列の配列を返す。

```
# コマンドライン
$ ruby test.rb v=abc

# test.rb の内容
args=Margs.new(ARGV)
puts args.str("v=") # ->"abc"
puts args.str("w=") # -> nil
puts args.str("w=","xyz") # -> "xyz"
```

```
# コマンドライン
$ ruby test.rb v=abc,efg:xyz,hij

# test.rb の内容
args=Margs.new(ARGV)
puts args.str("v=") # ->"abc,efg:xyz,hij"
puts args.str("v=",nil,",") # ->["abc", "efg:xyz", "hij"]
puts args.str("v=",nil,",",":") # ->[["abc"], ["efg","xyz"], ["hij"]]
```

* MCMD::Margs.float(keyWord[,default][,from][,to]): Float 型数値引数の取得

keyWord "key=" のキーワード (String)
default 指定がなかったときのデフォルト値 (Float)。省略時は nil。
from 範囲チェックの下限値 (Float)。省略時は下限値チェックをしない。
to 範囲チェックの上限値 (Float)。省略時は上限値チェックをしない。

コマンドライン上で指定された引数のうち、keyWord とマッチする値を Float に変換して返す。コマンドラインで指定されていなければ default の値を返す。範囲チェックにパスしなければエラー終了する。

```
# コマンドライン
$ ruby test.rb v=0.12

# test.rb の内容
args=Margs.new(ARGV)
puts args.float("v=") # -> 0.12
puts args.float("v=",nil,0.2,0.3) # -> 範囲エラー
puts args.float("w=") # -> nil
puts args.float("w=",0.1) # -> 0.1
```

* MCMD::Margs.int(keyWord[,default][,from][,to]) Fixnum 型数値引数の

取得

keyWord key=のキーワード (String)
default 指定がなかったときのデフォルト値 (Float)。省略時は nil。
from 範囲チェックの下限値 (Float)。省略時は下限値チェックをしない。
to 範囲チェックの上限値 (Float)。省略時は上限値チェックをしない。

コマンドライン上で指定された引数のうち、keyWord とマッチする値を Float に変換して返す。コマンドラインで指定されていなければ default の値を返す。範囲チェックにパスしなければエラー終了する。

```
# コマンドライン
$ ruby test.rb v=10

# test.rb の内容
args=Margs.new(ARGV)
puts args.int("v=") # -> 10
puts args.int("v=",,20,30) # -> 範囲エラー
puts args.int("w=") # -> nil
puts args.int("w=",15) # -> 15
```

* MCMD::Margs.bool(keyWord) Bool 型引数の取得

keyWord "-key"によるキーワード (String)

コマンドライン上で指定された引数のうち、keyWord とマッチする引数があれば true を、なければ false を返す。

```
# コマンドライン
$ ruby test.rb -flag

# test.rb の内容
args=Margs.new(ARGV)
puts args.bool("-flag") # -> true
puts args.bool("-x") # -> false
```

2.3.2 利用例

例 1

```
# コマンドライン
$ ruby test.rb i=dat.csv v=value -abc

# test.rb の内容
args=Margs.new(ARGV, "i=,o=,w=,-flag,-x", "i=,w=")
iFileName = args.file("i=") # -> "dat.csv"
oFileName = args.str("o=","result.csv") # -> "result.csv"
weight    = args.float("w=",0.1,0.0,1.0) # -> 0.1
flag      = args.bool("-abc") # -> true
wFlag     = args.bool("-w") # -> false
```

2.3.3 関連コマンド

2.4 Mtable CSV データのセル単位での読み込み操作クラス

指定した CSV データ全体をメモリに読み込み、セル単位でランダムアクセスを可能としたクラス。以下のような特徴を持つ。

- 行と列の指定によりセルをランダムにアクセス可能。
- 読み込み専用であり、データの更新や追加は一切できない。
- データは全て文字列として読み込むので、その他の方で利用する時は適宜型変換 (ex. to_i) が必要。
- メモリが空いている限りデータを読み込む。領域がなくなればエラー終了する。

2.4.1 メソッド

* MCMD::Mtable::new(arguments)

Mtable オブジェクトを生成する。`arguments` に、以下の引数をスペースで区切った文字列として指定する。

`i=` 入力ファイル名 (String)
`-nfn` 1 行目を項目名と見なさない。

* MCMD::Mtable::cell(col=0, row=0) -> String

row(行),col(列) に対応するセルの値を返す。row,col の与え方は、列番号と行番号による。列/行番号共に 0 から始まる整数 (Mcsvin の列番号は 1 から始まる)。row,col が範囲外の場合は nil を返す。

row 行番号で、0 以上の整数を用いる。デフォルトは 0。
col 列番号で、0 以上の整数を用いる。項目名は指定できない。デフォルトは 0。

col のみ与えると 0 行目の col 番目の項目の値を返す。cell(col,0) を指定したのと同等。また col,row 両方とも与えなければ 0 行目の 0 項目目の値を返す。cell(0,0) を指定したのと同等。

* MCMD::Mtable::names() -> String Array

項目名配列を返す。

* MCMD::Mtable::name2num() -> String=>Fixnum Hash

項目名をキー、対応する項目番号を値とする Hash を返す。

* MCMD::Mtable::size() -> Fixnum

行数を返す。

2.4.2 利用例

例 1

```
# dat1.csv
customer,date,amount
A,20081201,10
B,20081002,40

tbl=MCMD::Mtable.new("i=dat1.csv")
```

```
p tbl.names     # -> ["customer", "date", "amount"]
p tbl.name2num  # -> {"amount"=>2, "date"=>1, "customer"=>0}
p tbl.size      # -> 2
p tbl.cell(0,0) # -> "A"
p tbl.cell(0,1) # -> "B"
p tbl.cell(1,1) # -> "20081202"
p tbl.cell(1)   # -> "20081201"
p tbl.cell      # -> "A"
```

例 2 項目名行なし

```
# dat1.csv
customer,date,amount
A,20081201,10
B,20081002,40

tbl=MCMD::Mtable.new("i=dat1.csv -nfn") # 一行目もデータと見なしてしまう。
p tbl.names     # -> nil
p tbl.name2num  # -> nil
p tbl.size      # -> 3
p tbl.cell(0,0) # -> "customer"
p tbl.cell(0,1) # -> "A"
p tbl.cell(1,1) # -> "20081201"
p tbl.cell(1)   # -> "date"
p tbl.cell      # -> "customer"
```

2.4.3 関連コマンド

Mcsvin：CSV データの読み込み

第 3 章

モジュールメソッド

3.1　meach 簡単な並列処理の実行メソッド

排他制御を伴わない簡単な並列処理を実行する Array クラスのメソッド。複数のプロセスを非同期に起動するだけの簡単な実装である。

3.1.1　書式

1) `Array::meach(mpcount){|value| block}`

2) `Array::meach(mpcount){|value,count| block}`

配列の要素をブロックパラメータ `value` として、`block` で与えられたコードを並列に実行する。ブロックパラメータとして `count` を与えれば、実行中の配列要素番号 (0 から始まる整数) がセットされる。

`mpcount`　並列で実行するプロセス数。

3.1.2　利用例

例 1 項目名の出力と行番号・値の出力

```
# 10 から 6 までの 5 つの要素（整数）について、2 つのプロセスで並列処理する。
# 処理内容は、要素の値と配列の要素番号を表示するだけの簡単なものである。
> [10,9,8,7,6].meach(2){|value,count|
>   puts "#{value},#{count}"
> }
10,0
9,1
8,2
7,3
6,4
```

3.1.3　関連コマンド

3.2 mheader CSV データの項目名配列取得メソッド

CSV データファイルの項目名 (先頭行) を配列で返す。先頭行がなくデータ行から始まるファイルについては、1 行目の各項目の値を配列で返す。

3.2.1 書式

`MCMD::mheader(arguments)`

`arguments` に、以下の引数をスペースで区切った文字列として指定する。

`i=` 入力ファイル名 (String)

3.2.2 利用例

例 1

```
# dat1.csv
# 顧客,日付,金額
# A,20081201,10
# B,20081002,40

> p MCMD::mheader("i=dat1.csv")
 => ["顧客", "日付", "金額"]
```

例 2

```
# dat1.csv
# A,20081201,10
# B,20081002,40

> p MCMD::mheader("i=dat1.csv")
 => ["A", "20081201", "10"]
```

3.2.3 関連コマンド

Mcsvin：CSV データの読み込みクラス
Mtable：CSV データのセル単位での読み込みクラス

3.3　mrecount CSV データの行数計算メソッド

CSV データファイルを行単位で処理するためのクラス。以下のような特徴を持つ。

- C++ で実装されており非常に高速に動作する (wc -l より若干高速)。
- 一行目の項目名行を除いたデータの行数のみカウントする。
- 単純に改行 char をカウントしているだけなので、ダブルクォーテーションでエスケープされた改行もカウントしてしまう。この問題を避けたい場合は MCMD::Mtable を利用する。

3.3.1　書式

`MCMD::mrecount(arguments)`

`arguments` に、以下の引数をスペースで区切った文字列として指定する。

`i=`　　入力ファイル名 (String)
`-nfn`　1 行目を項目名と見なさない。

3.3.2　利用例

例 1 項目名の出力と行番号・値の出力

```
# dat1.csv
顧客，日付，金額
A,20081201,10
B,20081002,40

p MCMD::mrecount("i=dat1.csv")       # -> 2
p MCMD::mrecount("i=dat1.csv -nfn") # -> 3
```

3.3.3　関連コマンド

CSV データのセル単位での読み込み操作

3.3.4　ベンチマークテスト

CSV データの行数カウント処理について、UNIX コマンド `wc` と `Mtable` をベンチマークにした速度比較を行う。表 3.1 にベンチマークテストの結果を示す。100 万,200 万,300 万,400 万行,500 万行のデータについて実験を行った。図 3.1 には、ベンチマークテストで利用したスクリプトの抜粋が示されている。`mrecount` は `wc` に比べて若干高速である。また `Mtable` は行数をカウントすることのみを目的としたクラスではないが、比較すると、5〜6 倍高速である。

表 3.1 各種 CSV ライブラリの実行速度比較 (単位:秒)

行数	1000K	2000K	3000K	4000K	5000K
mrecount	0.034	0.066	0.097	0.129	0.161
wc -l	0.038	0.070	0.103	0.133	0.169
Mtable	0.231	0.407	0.503	0.731	0.828

10 回実行した結果の平均値 (real time) を示している。
行数 1000K は 100 万行の意味。データのサイズは行数が 1000K で約 25M バイト。項目数は 5 つ。
テスト環境: Mac Book Pro, 2.66GHz Intel Core i7, 8GB メモリ, Mac OS X 10.6.8

```
require 'rubygems'
require 'mtools'

require 'benchmark'

puts Benchmark.measure{
  (0...10).each{|i|
    # mrecount の場合
    p MCMD::mrecount("i=data.csv")

    # wc の場合
    system "wc -l data.csv"

    # Mtable の場合
    MCMD::Mtable("i=data.csv -array"){|tbl|
                        p tbl.recordSize
    }
  }
}
```

図 3.1 ベンチマークテストのスクリプト (抜粋)

3.4　mkDir ディレクトリの作成メソッド

ディレクトリを作成する。パラメータの指定により、指定したディレクトリが存在すれば削除してから作成する。

3.4.1　書式

`MCMD::mkDir(dirName[,rmExistingDir])`

`dirName`　作成するディレクトリ名。

`rmExistingDir`　true を指定したとき、`dirName` が既に存在してれば、削除してから作成する。

3.4.2　利用例

例 1 基本例

```
# ./folder というディレクトリを作成する。
# すでに存在していれば何もしない。
> MCMD::mkDir("./folder")
# 第２引数に true を指定したとき、すでに./folder 存在していれば削除してからディレクトリを作成する。
> MCMD::mkDir("./folder",true)
```

3.4.3　関連コマンド

3.5 endLog 終了ログメッセージの出力

MCMD の正常終了と同様のフォーマットのログメッセージを出力する。フォーマットは以下の通りである。

```
#END# メッセージ; 日時
```

3.5.1 書式

* MCMD::endLog(msg[,fileObject])

msg 表示するメッセージ。
fileObject 出力するファイルオブジェクト。省略すれば標準エラー出力 (STDERR) に出力される。

3.5.2 利用例

例 1 基本例

```
# 標準エラー出力に終了メッセージを表示する。
> MCMD::endLog("mburst.rb 正常に終了しました")
#END# mburst.rb 正常に終了しました。; 2013/11/01 19:09:50
```

3.5.3 関連コマンド

errorLog : エラーログメッセージの出力
messageLog : 一般ログメッセージの出力

3.6　errorLog エラーログメッセージの出力

MCMD のエラー終了と同様のフォーマットのログメッセージを出力する。フォーマットは以下の通りである。

```
#ERROR# メッセージ; 日時
```

3.6.1　書式

```
MCMD::errorLog(msg[,fileObject])
```

`msg`　表示するメッセージ。

`fileObject`　出力するファイルオブジェクト。省略すれば標準エラー出力 (STDERR) に出力される。

3.6.2　利用例

例 1 基本例

```
# 標準エラー出力に終了メッセージを表示する。
> MCMD::errorLog("mburst.rb エラー終了しました。")
#END# mburst.rb  エラー終了しました。; 2013/11/01 19:09:50
```

3.6.3　関連コマンド

endLog：終了ログメッセージの出力

messageLog：一般ログメッセージの出力

3.7 messageLog 終了ログメッセージの出力

MCMD の一般メッセージと同様のフォーマットのログメッセージを出力する。フォーマットは以下の通りである。

```
#MSG# メッセージ; 日時
```

3.7.1 書式

* MCMD::messageLog(msg[,fileObject])

msg 表示するメッセージ。
fileObject 出力するファイルオブジェクト。省略すれば標準エラー出力 (STDERR) に出力される。

3.7.2 利用例

例 1 基本例

```
# 標準エラー出力に一般メッセージを表示する。
> MCMD::messageLog("mburst.rb これはメッセージです。")
#END# mburst.rb これはメッセージです。; 2013/11/01 19:09:50
```

3.7.3 関連コマンド

errorLog : エラーログメッセージの出力
endLog : 終了ログメッセージの出力

第 4 章

その他

4.1　MCMD モジュール環境設定

MCMD モジュールは、KGMOD ライブラリを利用して構築されているので、KGMOD で設定可能なシェル変数は全て利用することができる。以下では、Ruby MCMD モジュールを利用する中での設定方法について解説する。

4.1.1　メッセージ出力

環境変数 KG_VerboseLevel を設定することで Mcsvin などのメソッドが標準エラー出力に出力するメッセージを制御できる。設定値とその内容は以下の通りである。

表 4.1　メッセージを制御する環境変数の設定値とその内容

値	内容
0	メッセージを一切出力しない
1	+ error メッセージ出力
2	+ warning メッセージ出力
3	+ end メッセージ出力
4	+ msg メッセージ出力 (デフォルト)

```
$ irb
> require 'mcmd'

# デフォルトでは KG_Verbose=4 なので、正常終了時のメッセージもエラー終了メッセージも出力される。
> MCMD::Mcsvin.new("i=dat.csv"){|csv| csv.each{|flds|}}
#END# mcsvin i=dat.csv; ; 2013/08/08 15:18:52
> MCMD::Mcsvin.new("x=dat.csv"){|csv| csv.each{|flds|}}
#ERROR# unknown parameter x= (mcsvin); mcsvin x=dat.csv; ; 2013/08/08 15:18:52

# KG_Verbose=1 とすると、エラー終了メッセージは表示されるが、正常終了メッセージは表示されない。
> ENV["KG_VerboseLevel"] = "1"
> MCMD::Mcsvin.new("i=dat.csv"){|csv| csv.each{|flds|}}
> MCMD::Mcsvin.new("x=dat.csv"){|csv| csv.each{|flds|}}
#ERROR# unknown parameter x= (mcsvin); mcsvin x=dat.csv; ; 2013/08/08 15:18:52

# KG_Verbose=0 とすると、いずれのメッセージも表示されない。
> ENV["KG_VerboseLevel"] = "0"
> MCMD::Mcsvin.new("i=dat.csv"){|csv| csv.each{|flds|}}
> MCMD::Mcsvin.new("x=dat.csv"){|csv| csv.each{|flds|}}
```