[HOME]

Perl で CSV 形式ファイルを扱う

黒崎 浩行
<hkuro@kokugakuin.ac.jp>
Since Nov 2, 1999.
Last updated on: Nov 12, 1999.

目次

はじめに

このメモは、調査データなどの処理で CSV 形式ファイルを扱うときに、[スリニバサン 1998 (1997)] を参考に、 perl のオブジェクト指向プログラミングを勉強しながら作業したことをまとめたものです。

CPAN を探してみると、 以下に述べるのと同様の処理をカバーする、 汎用性の高いモジュールがすでに登録されていることがわかります。 それらは各節の末尾に紹介しておきました。

今のところ、モジュールの設計方法など、まったく我流でやっていますので、 おかしな点のご指摘やアドバイスなどありましたら、ご教示いただければ幸いです。

[目次へ]

CSV 形式の文字列を配列に分割する

むかし日経MIXの awk 会議室で、CSV 形式の文字列を配列に分割するスクリプトが議論されていた。 それを perl で書くと、次のようなものだった。

```
sub splitcsv {
    local($csvstr) = @_;

    $csvstr .= ',';
    $csvstr =~ s/("([^"]|"")*"|[^,]*),/$1$;/g;
    $csvstr =~ s/"([^$;]*)"$;/$1$;/g;
    $csvstr =~ s/""/"/g;
    return split(/$;/, $csvstr);
}
```

ちょっと複雑でわかりにくいが、 $; という perl の特殊変数を使っていることに注目するとよい。 これは、「多次元配列のエミュレーションに用いる添え字セパレータ」[ウォールほか

1997 (1996): 150] である。 フィールドの区切りに相当するカンマをこの文字に置換し、 split 関数を使って配列に分割するというのが、基本的な考え方である。

CSV には次のような決まりがある。

- フィールド内にカンマが登場するときは必ず、フィールドの両端をダブルクォートで囲む。
- フィールド内のダブルクォートは、ダブルクォート2つで表す。

この決まりによって、5行めのパターンマッチが書かれている。 あとは、フィールド両端のダブルクォートを取り除いたり、 ダブルクォート2つをダブルクォート1つに置換しているだけである。

同じルーチンを、今話題のオブジェクト指向スクリプト言語 Ruby で書くと、次のようになる (全然オブジェクト指向になっていないが)。

```
def splitcsv(str)
  sep = "\x1c"
  str << ','
  str = str.gsub('("([^"]|"")*"|[^,]*),', "\\1#{sep}")
  str = str.gsub("\"([^#{sep}]*)\"#{sep}", "\\1#{sep}")
  str = str.gsub('""', '"')
  return str.split(sep)
end
```

ちなみに、CPAN には、Text::CSV と Text::CSV_XS という、同様の機能をもつモジュールが登録されている。

[目次へ]

複数行レコードへの対応

上記の splitcsv サブルーチンを使って CSV 形式のファイルを読み込んで処理するには次のように書けばよい。

```
while (<>) {
    chomp;
    @a = splitcsv($_);
    # 処理 ...
}
```

しかし、アプリケーションによっては、 CSV ファイルを次のような形で出力するものがある。

- フィールド内に改行文字が入ってもよい。 その代わり、フィールドの両端をダブルクォートで囲む。

これに対応するには、ファイルを読み込むループを、 次のように書き換えて、続きの行を連

結する必要がある。

```
my $REC_LIMIT = 65535;
my $lc = 1;
my $lastargv = '';
my $sx = '';

while (<>) {
    # 標準入力かコマンドライン引数で指定したファイルから読み込む
    $lastargv ne $ARGV ? $lc = 1 : $lc++;
    while (($_ =~ s/"/"/g) % 2) {
        die "Error: " . $ARGV . "(" . $lc . "): record too large\n"
            if length($_) > $REC_LIMIT;
        die "Error: " . $ARGV ."(" . $lc . "): invalid data\n"
            unless ($sx = <>); # 続きを読み込む
        $lastargv ne $ARGV ? $lc = 1 : $lc++;
        $_ .= $sx;
        $lastargv = $ARGV;
    }
    $lastargv = $ARGV;
    chomp;
    @a = splitcsv($_);
    # 処理 ...
}
```

これは、1つのレコードを表す文字列は、 必ず偶数またはゼロのダブルクォートを含んでいることにもとづいている。 ある行が奇数のダブルクォートを含んでいる場合は、 ダブルクォートは開いた状態にある、ということである。 そこで、次の行を続きとして読み込んでいく。

壊れた CSV ファイルをメモリの限界に達するまで読み込んでしまうのを防ぐために、 1行の文字数制限を `$REC_LIMIT` に指定し、 それを超えたら処理を中断するようにしている。

[目次へ]

各フィールドに名前でアクセスする

splitcsv 関数から返される配列の各要素にアクセスするには、 数字による添え字を使えばよいが、これはプログラムの可読性を損ねてしまう。

そこでいろいろな工夫が考えられるが、CSV ファイルにはひとつ重要な特徴がある。

- 1行めにフィールド名を入れることが多い。

このフィールド名を使って各要素にアクセスできたら、 プログラムが読みやすくなる。そこで、次のような CSV.pm モジュールを作ってみた。

```
package CSV;
# Simple CSV manipulation module
# Written by Hiroyuki KUROSAKI <hkuro@kokugakuin.ac.jp>
```

```
# Date: Nov 2, 1999.

sub splitcsv {
    local($csvstr) = @_;

    $csvstr .= ',';
    $csvstr =~ s/("([^"]|"")*"|[^,]*),/$1$;/g;
    $csvstr =~ s/"([^$;]*)"$;/$1$;/g;
    $csvstr =~ s/""/"/g;
    return split(/$;/, $csvstr);
}

sub new {
    my ($pkg, $line) = @_;
    my (@a) = splitcsv($line);
    my %csv;
    $csv{'_items'} = \@a;
    if (ref($pkg)) {
        for ($i = 0; $i <= $#a; $i++) {
            $csv{'_' . $pkg->item($i)} = \$a[$i];
        }
    }
    bless \%csv;
}

sub item {
    my ($r_csv, $num) = @_;
    return $r_csv->{_items}[$num];
}

sub field {
    my ($r_csv, $name) = @_;
    return ${$r_csv->{'_' . $name}};
}

1;
```

コンストラクタ (new メソッド) は、 クラスメソッドのときとインスタンスメソッドのときとで動作が異なる。 `$csvhead = new CSV($csvstring);` のように、 クラスメソッドとして呼び出した場合は、 メンバ `_items` に、 splitcsv 関数によって分割した配列へのリファレンスが入るだけである。 これは1行めを読み込んだときの処理を想定している。

`$csvbody = $csvhead->new($csvstring);` のように、 インスタンスメソッドとして呼び出した場合は、 オブジェクトに格納されている `_items` をもとに メンバを生成し (**_メンバ名**) 、 それぞれに配列の各要素へのリファレンスを入れている。 こちらは2行目以降の処理にあたる。1行めをフィールド名とみなして、 フィールド名と配列の各要素との対応づけを行っているわけである。

item と field はアクセスメソッド。field を用意したのは、 [スリニバサン 1998 (1997): 134] に挙げられているようなメリットに加えて、 ややこしいデリファレンスを隠すというメリットもある。

この CSV.pm モジュールを使えば、 フィールド名による各要素へのアクセスが可能になる。 以下は、タイトル (フィールド名 title) と著者名 (フィールド名 author) が任意の位置にある CSV 形式ファイルを読み込んで、 整形出力するサンプルである。

```
use CSV;

my ($csvhead, $csvbody);

my $REC_LIMIT = 65535;
my $lc = 1;
my $lastargv = '';
my $sx = '';

while (<>) {
    # 標準入力かコマンドライン引数で指定したファイルから読み込む
    $lastargv ne $ARGV ? $lc = 1 : $lc++;
    while (($_ =~ s/"/"/g) % 2) {
        die "Error: " . $ARGV . "(" . $lc . "): record too large\n"
            if length($_) > $REC_LIMIT;
        die "Error: " . $ARGV ."(" . $lc . "): invalid data\n"
            unless ($sx = <>); # 続きを読み込む
        $lastargv ne $ARGV ? $lc = 1 : $lc++;
        $_ .= $sx;
        $lastargv = $ARGV;
    }
    $lastargv = $ARGV;
    chomp;
    if ($lc == 1) {
        $csvhead = new CSV($_);
    }
    else {
        $csvbody = $csvhead->new($_);
        print $csvbody->field('author')
             . '「' . $csvbody->field('title') . "」\n";
    }
}
```

ちなみに、 CPAN には、DBD::CSV という、 CSV 形式のファイルをデータベースとして扱い、 SQL 言語で操作できるようにする、 かなり汎用性の高いモジュールが登録されている。 これを使うと、上のコードは次のように書ける (入力ファイルは bib.csv とする)。

```
use DBI;

my $csvfile = 'bib.csv';

my $dbh = DBI->connect("DBI:CSV:");
$dbh->{'csv_tables'}->{'bib'} = {
    'file' => $csvfile,
    'eol' => "\n" };
my $sth = $dbh->prepare("SELECT * FROM bib");
$sth->execute();
```

```
while (my $row = $sth->fetchrow_hashref) {
    print $row->{'author'}
        . '「' . $row->{'title'} . "」\n";
}
$sth->finish();
$dbh->disconnect();
```

[目次へ]

参考文献

- [ウォールほか 1997 (1996)] ウォール, ラリー、クリスチャンセン, トム、シュワルツ, ランダル L. 1997 (1996) 『プログラミング Perl 改訂版』近藤嘉雪訳、オライリー・ジャパン。
- [スリニバサン 1998 (1997)] スリニバサン, スリラム 1998 (1997) 『実用Perlプログラミング』須田隆久訳、オライリー・ジャパン。
- 三島俊司「Perl初心者の部屋」 (http://www.harukaze.net/~mishima/perl/)
- 近藤嘉雪「Perlの部屋」 (http://www.context.co.jp/~cond/perl/)

[目次へ]

[HOME]

Hiroyuki KUROSAKI <hkuro@kokugakuin.ac.jp>