2016 年 08 月 05 日

計算機実習 III (2016 年度)
第 15 回: バッチファイル、AWK の応用 その 3

(http://takeno.iee.niit.ac.jp/%7Eshige/
math/lecture/comp4/comp4.html)

目 次

1 system() 関数

今回は、バッチファイルや AWK の応用例として、AWK 内からコマンドプロンプトコマンドやバッチファイルを実行する system() 関数、バッチファイルから AWK に子バッチファイルを作らせてそれを実行する方法について説明する。

コマンドプロンプト (バッチファイル) では、環境変数の利用や、for, goto によるループ、if での条件分岐等が利用できるが、制限が色々ありプログラミングには十分とは言えない。例えば配列変数もないし、if 文の条件の AND や OR などもない。

一方、AWK の文法はしっかりしていて、C とほぼ同じような構文、配列などが利用できるから、コマンドプロンプト (バッチファイル) でもその機能を利用すればよい。この節では、その方法の一つとして、system() 関数を利用する方法を紹介する。

AWK は、外部コマンド、すなわちコマンドプロンプトで実行できるコマンドを実行するための system() という関数を持っている [1]。これを使えば、バッチファイルで行えることは、ほぼ AWK スクリプト内部からもすべて行えることになる。

[1] system() は、AWK だけでなく、C 言語にもあるし多くのプログラミング言語が標準的に持っている。

関数	意味
system(s)	文字列 s をコマンド文字列として AWK 外部で実行する

表 1: 関数 system()

system() の引数の s には、リダイレクションやパイプを含んだ文字列を指定することもできる。system() は、AWK のコマンドを一旦中断して外部コマンドを実行し、終了後また AWK の処理に戻る。また、system() に返り値はない。

例えば、以下のバッチファイル (test1.bat とする)

```
@echo off
dir
time /t
color 64
```

の作業は、以下の AWK スクリプト (test1.awk) でも同じことができる。

```
BEGIN { system("dir"); system("time /t");
    system("color 64") }
```

このスクリプトをコマンドプロンプトで「gawk -f test1.awk」と実行すれば、test1.bat を実行したのと同じことが行われ、同じように表示される。

以下の AWK スクリプトは、今日の日付が 5 の倍数ならブラウザで工科大のホームページを開く:

```
BEGIN { d = strftime("%d")
    if (d % 5 == 0) system("start http://www.niit.ac.jp") }
```

同じことをバッチファイルでやると以下のようになる。

```
@echo off
set /a d=(1%date:~-2% - 100) %% 5
if %d%==0 start htt://www.niit.ac.jp
```

実質的には AWK と同じく 2 行で書けているが、バッチファイルの方の d の設定はやや面倒で、少し不自然なことをしている[2]。それに、バッチファイルの if の条件部分には AWK の例のような「d % 5」のような数式を書くことはできない。

[2] このようなことをしなければいけない理由については、第 3 回の資料の p7 を参照。

10 日前の日付けのファイル名のログファイル (20160726.log 等) を削除し、今日の日付のファイル名のログファイルに現在の日付と時刻を書く AWK スクリプト:

```
BEGIN { tcur = systime() ; told = tcur - 24*60*60*10
    fold = strftime("%Y%m%d.log", told)
    fcur = strftime("%Y%m%d.log", tcur)
    scur = strftime("%Y-%m-%d %H:%M:%S", tcur)
    system(sprintf("if exist %s del %s", fold, fold))
    system(sprintf("echo %s > %s", scur, fcur)) }
```

tcur は現在の時刻 (epoch 秒)、told は 10 日前の時刻で、fold、fcur にそれぞれ 10 日前と現在のログファイルの名前を strftime() で作成し、scur には現在の日付と時刻の文字列を作成した上で、古いログファイルの削除と新しいログファイルの作成をそれぞれ system() で行っている。実際には、最後の 2 つの system() により、

```
if exist 20160726.log del 20160726.log
echo 2016-08-05 13:12:00 > 20160805.log
```

の 2 つのコマンドがコマンドプロンプト上で実行されることになる。

現在の日付や時刻はバッチファイルでも取得できるが、それらを自由な書式で書くのは難しいし、バッチファイルだけで 10 日前の日付を文字列として作りだすことも難しいが、gawk ならば容易にそれらが行える。

2 子バッチファイルと call コマンド

バッチファイルの中から別のバッチファイルを実行するために、call というコマンドが用意されている。

コマンド	意味
call [バッチファイル]	[バッチファイル] を実行して、元のバッチファイルの次の行に戻る

表 2: 子バッチファイルの実行

この命令の後ろに、実行するバッチファイルにオプションを指定することも可能である。

例えば test2.bat が以下のようであるとき、

```
@echo off
rem こちらが子バッチファイル (test2.bat)
set x=8
echo [%0] 1. [%x%] [%y%]
set /a x=x + 1
set /a y=y + %1
```

以下のバッチファイル test3.bat

```
@echo off
rem こちらが親バッチファイル (test3.bat)
set x=3
set y=5
echo [%0] 1. [%x%] [%y%]
call test2.bat 10
echo [%0] 2. [%x%] [%y%]
```

を実行すると、

```
Z:¥> test3.bat
 [test3.bat] 1. [3] [5]
 [test2.bat] 1. [8] [5]
 [test3.bat] 2. [9] [15]
```

と表示される。すなわち、call で test2.bat を呼び出すと、その中の echo 行の表示が行われ、x と y の値が更新され、そして test3.bat の call 行の次の行に処理が戻り、echo 行の表示が行われる。test2.bat の後ろに指定したオプションである 10 は、test2.bat に渡され、その中で %1 として使用されている。

環境変数の値は、親バッチファイルで設定した値は子バッチファイルの中でもその値を参照できるし、子バッチファイル変更した値は親バッチファイルでもその値を参照できるようになっている。

3 gawk とバッチファイルとの連携

gawk にコマンドプロンプトのコマンドを実行してもらう場合、以下のような 3 種類の形態が考えられる。

- **(A)** AWK スクリプトから直接 system() で必要なコマンドを実行する

- **(B)** AWK スクリプトで内部リダイレクトで子バッチファイルを作成し、それを gawk 側から `system()` で実行する
- **(C)** (親) バッチファイルから gawk を実行してバッチファイル行を出力させ、それをリダイクレクトで子バッチファイルとして保存し、それを親バッチファイルから `call` で実行する

例えば、以下のような作業を、上の 3 通りで行ってみる:

> `IMG_1.JPEG`, `IMG_2.JPEG`, ..., `IMG_150.JPEG` という 150 個のファイルを、`file0008.jpg`, `file0009.jpg`, ..., `file0157.jpg` という名前に変えて `AKB` というディレクトリ内に移動する

実際に実行するには、`move` コマンドを用いて、ひたすら

```
Z:¥> move IMG_1.JPEG AKB¥file0008.jpg
Z:¥> move IMG_2.JPEG AKB¥file0009.jpg
.........
```

のようなことを 150 回やればいいのだが、それを AWK の `for` 文と `printf`、`system()` を利用して簡単に実行することを考える。

なお、バッチファイルの `for` 文では、連続的に 「0008」から「0157」のような文字列を作るのが難しく、この作業をバッチファイルだけでやるのは簡単ではない。

- **(A) gawk で直接 `system()` で実行**

  この場合、以下のような AWK スクリプトを 1 回実行するだけでよい。

```
BEGIN {
    for (j=1; j<=150; j++) {
        s = sprintf("move IMG_%d.JPEG AKB¥¥file%04d.jpg",
        j, j + 7)
        system(s)
    }}
```

  なお、わかりやすくするために `sprintf()` の出力を `s` に代入してから `system()` に渡しているが、直接 `system()` の中に `sprintf()` 部分を書けば `for` 文の中は 1 行で済む。

  この方法の場合、`system()` コマンドが 150 回実行されることになるが、`system()` の呼び出しは、実は毎回新たなコマンドインタープリター (コマンドプロンプトの実行ファイル) を呼び出しているので少し時間がかかり、このやり方は (B), (C) に比べてかなり遅くなる。

- **(B) gawk で子バッチを作ってそれを system() で実行**

  これも、以下の AWK スクリプトを 1 回実行するだけだが、system() コマンドを呼び出す回数は少なくて済む (以下では 2 回)。

```
BEGIN { fname = "tmp1.bat"
    for (j=1; j<=150; j++)
        printf "move IMG_%d.JPEG AKB¥¥file%04d.jpg¥n",
        j, j + 7 > fname
    close(fname); system(fname); system("del " fname) }
```

  tmp1.bat が作成する子バッチファイルで、内部リダイレクトを用いて実行する文をそこに書き出している。(A) とは違い、毎回改行 (¥n) も出力していることに注意せよ。

  それが終わったら、一旦そのファイルをクローズして、system() で実行し、最後の system() では、いらなくなったそのバッチファイルも del コマンドで削除している (それも system() で実行している)。

  なお、tmp1.bat には「@echo off」をつけていないが、system() による実行では、実際には echo off と同じ状態で実行される。

  この方法は、system() の呼び出しが少ないため、(A) よりもかなり早く作業が終わる。

- **(C) gawk に子バッチを吐かせてそれをバッチファイルで実行**

  この場合は、以下のような AWK スクリプト (testc.awk) と、バッチファイル (testc.bat) を使用する。testc.awk:

```
BEGIN { for (j=1; j<=150; j++)
    printf "move IMG_%d.JPEG AKB¥¥file%04d.jpg¥n", j, j + 7 }
```

  testc.bat:

```
@echo off
gawk -f testc.awk > tmp1.bat
call tmp1.bat
del tmp1.bat
```

  このバッチファイル testc.bat を実行すると、gawk で testc.awk を実行して、その出力をリダイレクションを使って子バッチファイル tmp1.bat として保存し、それを実行して、最後にそれを削除する。

  これは、AWK の system() を使わず、AWK にはバッチファイルで実行させる各行のみを出力させているので、(A), (B) に比べても AWK スクリプト自体はシンプルだし、system() も使わないので、(B) と同様に作業は高速に終了する。

ただし、(C) の方法は、AWK スクリプトとバッチファイルの 2 つのファイルを編集、保守しないといけないので、それが (A), (B) と比べると少し面倒である。

4 バッチファイルと AWK の値のやりとり

バッチファイルと AWK の連携作業では、お互いに値をやりとりしたい場合もある。

バッチファイルから gawk へ値を渡すには、第 13 回で説明した -v オプションや、gawk への入力リダイレクションなどを使えばよいが、逆に gawk からバッチファイルへ値を渡すには少し工夫が必要である。

例えば、バッチファイルでは小数計算ができないから、それを gawk におぎなってもらうような次の作業を考える:

環境変数 x にあらかじめ整数が代入されていて、その平方根の値を「文字列として」環境変数 sqrtx に代入する。

x の値は、gawk には -v オプションで渡せばいいが、問題は、gawk から返される値をどうやって sqrtx に保存するか、である。前節の (A) のような AWK スクリプトでは実はうまくいかない:

```
BEGIN { system( sprintf("set sqrtx=%f", sqrt(x)) ) }
```

それは、p5 で述べたように、system() を起動すると、今実行しているコマンドプロンプトとは別なコマンドインタープリンターが実行され、そちらの上で「set sqrtx=...」の行が実行されるためで、その set コマンドの結果は、こちらのコマンドプロンプトには反映されない。(B) の方法も system() を使うので状況は同じである。

よって、この場合は、(C) の方法で gawk に子バッチファイルを作らせてそれをこちらで実行するしかない。AWK スクリプト test4.awk を

```
BEGIN { printf "set sqrtx=%f", sqrt(x) }
```

とし、バッチファイル test4.bat を

```
@echo off
gawk -v x=%x% -f test4.awk > tmp1.bat
call tmp1.bat
del tmp1.bat
echo %sqrtx%
```

として、この test4.bat を実行すれば、子バッチファイルの set コマンドにより、gawk の sqrt(x) の結果が文字列として sqrtx に保存され、表示されることになる。

なお、この test4.awk 程度の短いスクリプトならば1 行スクリプトとして書けるのでバッチファイル一つで済む:

```
@echo off
gawk -v x=%x% "{printf ¥"set sqrtx=%%f¥",sqrt(x)}" > tmp.bat
call tmp.bat
del tmp.bat
echo %sqrtx%
```

ただし、前に説明したように、バッチファイル内では変数展開用の % 以外の % は %% と書き、および 1 行スクリプト内の " は ¥" とすることに注意が必要。

5 最後に

バッチファイルでは、ここまでに上げた、10 日前の日付け文字列を作るとか、「0008〜0157」のような番号をつくるとか、実数計算をする、といったこと以外にも、

- 配列処理
- if 文の条件関連
- ファイルからデータを読み込んでそれに応じてコマンドを実行
- 2 つの時刻の時間を計測
- ( ) ブロック内の制限 (遅延展開や echo コマンド)

などが苦手である。これらはいずれも gawk で解消できる。

卒業研究などでも、例えば動作が終わるまで数時間かかるような C 言語で書いたシミュレーションプログラムを、色々なパラメータを与えて動かさないといけない場面だとか、ログ解析プログラムなどを、一定時間毎に動かしてデータを収集してファイルに追加出力させていくなど、バッチファイルが必要な場面、あるいはバッチファイルを使えば人間がコンピュータにはりついていなくてもいい場面が色々とでてくる。

その場合に、バッチファイルだけでは足りなくても gawk をうまく組み合わせることで解決する問題もたくさんあるだろう。今回学んだことをそういった場面にうまく活かしてもらいたい。