2006 年 9 月 5 日

AWK による HTML ファイルの整形 その 2

新潟工科大学 情報電子工学科 竹野茂治

1 はじめに

以前、[4] で AWK で

- Yahoo! ニュース: `http://headlines.yahoo.co.jp/hl`

のニュース記事を加工する例を紹介しました。そこでは、ニュース記事のページを HTML ファイルとして手元に保存した後、その必要な部分のみ取り出した HTML ファイルを作成する、という方法を紹介しました。

今回は、複数のデータファイルを一度に処理する例として、[4] でも触れた、ニュース記事の一覧を加工することによるニュース記事へのアクセスの向上をはかる方法について紹介します。今回は以下のことを目標とします。

- 複数に分かれている一覧の HTML ファイルから必要な情報のみを取り出し一つの HTML ファイルにつなげる (リンクの貼り方も少し加工する)

私は、普段こうやって作った HTML ファイルをさらに加工しているのですが、それについてはまた次の機会に紹介したいと思います。

なお、加工したデータを個人的に楽しむのは違法ではありませんが、それを公開したり、第 3 者に渡したりするのは問題がありますので注意してください。

2 Yahoo! ニュース一覧の構造

Yahoo! ニュースの記事の一覧は、基本的に `<ul>`〜`</ul>` タグ (箇条書き) を使って、

```
<ul>
<li><a href="[記事本体の URL]">どこそこで交通事故</a><small> (XXX 新聞)
- 15 日 (火)15 時 35 分</small><br>
<li><a href="[記事本体の URL]">どこそこで火事</a><small> (XXX 新聞)"
- 15 日 (火)15 時 30 分</small><br>
```

```
....
</ul>
```

のように書かれていて、それによってブラウザでは、

- どこそこで交通事故 (XXX 新聞) – 15 日 (火)15 時 35 分
- どこそこで火事 (XXX 新聞) – 15 日 (火)15 時 30 分
- ...

のように表示され、記事本体へのリンクが貼られています。

しかし、[4] で紹介したニュース記事本体の HTML ファイルの構造と同様に、この一覧の HTML ファイルにも一覧以外に不要な情報が数多く含まれていますし、一覧が多い場合は一覧自体が複数の HTML ファイルに渡ることもあります。

よって、まずはその複数の一覧ファイルから必要な情報をとり出し、それをつなげて一つの HTML ファイルにすることを考えます。

必要な情報とは、以下のものとします。

1. <!--- CONTENTS_TITLE_TABLE ---> の下にある

   ```
   <b><font size=+1>XXX ニュース</font></b>
   <small> - 8 月 15 日 (火)15 時 40 分</small></td>
   ```

   のニュースタイトルと日付の部分
2. <!--- OUTLINE_TABLE ---> の下にある記事の一覧部分 (
〜 の部分)

他にも使えそうな情報がなくはないのですが、今回はこれらのみを取り出すことにします。

3 複数のデータファイルの処理

今回は、複数のデータファイルを処理することになるわけですが、AWK でそれを 1 ファイルずつ処理してもいいわけですが、一度に複数のデータファイルを処理することも可能です。

AWK に複数のデータファイルを処理させる場合は、単にそれを並べて指定するだけです。

```
awk -f [スクリプト] [データ 1] [データ 2] ...
```

この場合、まず [データ 1] が先頭から最後まで読み込まれて処理され、次に [データ 2] が読み込まれて処理され、という風に処理が進みますから、結局データファイルをすべてつなげて awk に処理をさせている、つまり

```
cat [データ 1] [データ 2] ... | awk -f [スクリプト]
```

と同等の処理をしていると見ることもできますが、暗黙の大域変数の中には各ファイル毎の情報を持つものもあって、この 2 つの処理方法では実際に異なる点もあります。

- FILENAME : 現在処理中のデータファイルのファイル名
- FNR : 現在処理中のデータファイルの先頭からの行番号
- NR : 全データファイル通しての先頭からの行番号
- ARGIND : 現在処理中のファイルのインデックス

これを利用すると、データファイルが切り替わったことを知って処理の内容をそれに合わせて変更する、などということも可能になりますが、今回はそこまでは必要ありません。

ファイルのインデックスというのは、引数の何番目か、ということを意味します。AWK の引数は、AWK 自身へのオプション指定とデータファイルの指定部分に分けられますが、AWK 自身へのオプション以外の部分、つまり (主に) データファイルの指定部分の引数は、C 言語のように ARGV という配列、ARGC という変数に保存されています。

- ARGC : コマンド名自体と AWK 自身へのオプション以外の引数の個数
- ARGV : それらを文字列として保存する配列 (ARGV[0]∼ARGV[ARGC−1])

例えば、

```
awk -f test.awk -v s=3 file1 file2
```

の場合は `-f test.awk` と `-v s=3` は AWK 自身へのオプションなので $\mathrm{ARGC} = 3$ であり、

```
ARGV[0]="awk", ARGV[1]="file1", ARGV[2]="file2"
```

となります。以下のようなスクリプトを test.awk として、file1, file2 を 2,3 行のファイルとして実際に上のように実行してみればこれらがなんとなくわかると思います。

```
# 暗黙の大域変数テスト
BEGIN{ for(j=1;j<=ARGC;j++) printf "ARGV[%d]=%s\n",j-1,ARGV[j-1] }
{
     printf "(FILENAME,FNR,NR,ARGIND)"
     printf "=(%s,%d,%d,%d)\n",FILENAME,FNR,NR,ARGIND
}
```

多分、結果は以下のようになります。

```
ARGV[0]=awk
ARGV[1]=file1
ARGV[2]=file2
(FILENAME,FNR,NR,ARGIND)=(file1,1,1,1)
(FILENAME,FNR,NR,ARGIND)=(file1,2,2,1)
(FILENAME,FNR,NR,ARGIND)=(file1,3,3,1)
(FILENAME,FNR,NR,ARGIND)=(file2,1,4,2)
(FILENAME,FNR,NR,ARGIND)=(file2,2,5,2)
(FILENAME,FNR,NR,ARGIND)=(file2,3,6,2)
```

今回取得するタイトルや日付データは、先頭のファイルからのみ取得すればいいので、それは ARGIND が 1 のときにだけ取得すればいいことになります。

4 タイトルと日付の取得

この節では、タイトルと日付の取得を考えます。これは、`<b><font size=+1>` という行がここにしかないようなので、簡単のためにこれを利用してこの行を取得し、その次の行も getline で取得することにします。不要なタグ部分は、sub() で削除します。

- `getline`: 現在の入力行の次の行を読みこんでそれを新たな現在行とし $0 等に代入する。読み込みに成功すると 1, ファイルの最後に達した場合は 0, 読み込みに失敗したら -1 を返す。
- `sub(r,s,c)`: 文字列 c (省略した場合は現在の入力行 $0) の、正規表現 r に最初にマッチした部分を文字列 s で置きかえる

タイトルと日付は最初のファイルから取得すればいいので、ARGIND が 1 のときにだけ取得します。

```
##### タイトルや日付の取得 #####
(ARGIND == 1 && $0 ~ /<b><font size=\+1>/){
    sub(/.*<b><font size=\+1>/,"")
    sub(/<\/font><\/b>.*/,"")
    title=$0
    getline
    sub(/.*<small> - /,"")
    sub(/<\/small>.*/,"")
    date=$0
}
```

正規表現は / / で囲んで指定します。+ は正規表現では特別な意味 (1 回以上の繰り返し) を持ちますので、\ でエスケープする必要があります。また、.* という正規表現は、

. = 任意の 1 文字
.* = 任意の 1 文字の 0 文字以上の繰り返し

ということで、すなわち「空文字列も含む任意の文字列」ということを意味しています。よって、例えば sub(/<\/small>.*/,"") は、

</small> とその後につながる任意の文字列を、空文字列 "" に変換

することを意味します。

5 一覧部分の取得

次は、一覧部分の取得ですが、実はこちらは少し厄介です。

2 節で一覧の構造を少し紹介しましたが、実際には少し違っていて、実は、各 <li> の行は別々の行にはなっておらず

```
<ul>
<li>...<br><li>...<br><li>...<br><li>...<br><li>...<br><br>
<li>...<br><li>...<br><li>...<br><li>...<br><li>...<br><br>
<li>...<br><li>...<br><li>...<br><li>...<br><li>...<br><br>
.....
</ul>
```

(. . . の部分がリンクのためのタグや見出し、日時などの部分)

のように、いくつかの行がつながって長い行を作っていて、いくつかのまとまり (5 個程度) 毎に最後の
 が 2 つつながっていてそこでやっと改行が入っています。よって 1 行を取得しても、そこに複数の <li> 行が含まれるので、それを分割する必要があります。

 や は、上では単独行で書きましたが必ずしもそうであるとは限らず、それも他の行につながっている場合もあるようです (実際最終ページの は直前の行の最後につながっている場合があります)。

このブロック部分の判別は、HTML ファイル内に 〜 ブロックがここにしか現れないようなので、簡単のため <ul> にマッチするかどうかで判別することにし、 が現われるまで getline で取得することにします。

なお、明示的な getline を使わず、フラグを使うことで現在 〜 内であるかどうかを判別する、という方法もありますが、それについては [4] を参照してください。

AWK の疑似コードは以下のようになります。

```
##### 各行の取得 #####
($0 ~ /<ul>/){
    sub(/<ul>/,"")
    if($0 !~ /<li>/) getline
    do{
        # (1) 1 行を複数の <li> 行に分割して配列に保存
        # (2) それを整形して出力
        getline
    }while($0 !~ /<\/ul>/)
    if($0 ~ /<li>/){
        sub(/<\/ul>/,"")
        # (1) 1 行を複数の <li> 行に分割して配列に保存
        # (2) それを整形して出力
    }
}
```

ところで、これだと同じ処理 (上の (1),(2)) を 2 箇所に書くことになりますが、それを 1 箇所にまとめるには、ループの中の getline を最初に持ってきて、

```
($0 ~ /<ul>){
    sub(/<ul>/,"")
    do{
        getline
```

```
        # (1) 1 行を複数の <li> 行に分割して配列に保存
        # (2) それを整形して出力
    }while($0 !~ /<\/ul>/)
}
```

とでもすれば済みそうですが、これだと最初の行が <ul> のすぐ後ろから が始まっている場合はそれを捨てることになってしまいますし、最後が 単独の行である場合にも対応しておらずその行に対して (1),(2) を実行してしまいます。

(1),(2) の部分をループの中に 1 回だけしか書かないようにするには、フラグなどを用いて次のようにする方法があります:

```
##### 各行の取得 その 2 #####
($0 ~ /<ul>){
    sub(/<ul>/,"")
    flag=0
    do{
        if(flag==1 || $0 !~ /<li>/) getline
        flag=1
        if($0 ~ /<\/ul>/ && $0 !~ /<li>/) break
        # (1) 1 行を複数の <li> 行に分割して配列に保存
        # (2) それを整形して出力
    }while($0 !~ /<\/ul>/)
}
```

しかしこれだと、最初の if 文が意味があるのは
～ ブロックの最初の 1 行だけで、その他の行に関しては if 文の判断は無駄になりますし、2 つ目の if 文も最後の行にしか意味がないので、その他の行に関しては if 文の判断は無駄です。

最初のコードと最後のコードは、(1),(2) のような部分を何度も書いていない、という点では後者の方が綺麗に見えるかもしれませんが、ループの中で無駄な処理を何度も繰り返すことになる、という点では前者の方が無駄はなくてよいコード、ということになります。

どちらを採用するかは、ある程度は趣味の問題ですが、とりあえずは最初のコードの方を採用することにします。

なお (2) の部分は、この時点で出力を行なうかわりに全体を一つの配列に保存しておいて、END ブロックでまとめて出力する、という方法もあります。このようにすると、全体の項目数を知ることもできますし、全体を逆順に出力することも可能になりますので、今回もそのような方向で設計することにします。

6 行の分割

ここでは、5 節で説明した (1) の行の分割の部分を考えます。

文字列をある文字列を切れ目として分割するには、split() が使えます。

> `split(s, h, r)`: 文字列 s を、正規表現 r を区切りとして区切って配列 h に保存し、その個数を返す

これで、例えば

```
<li>(あ)<br><li>(い)<br><li>(う)<br><br>
```

という文字列 str を

```
N=split(str,h,/<br>/)
```

とすると、N と配列 h の内容は以下のようになります:

```
N=5, h[1]="(あ)", h[2]="(い)", h[3]="(う)", h[4]="", h[5]=""
```

最後の 2 つにゴミが残っていますが、それは文字列の最後に `<br>` がついていることによります。よって、それを削除してから split() にかけるといいでしょう。同様に、

```
<li>(あ)<br><li>(い)<br><li>(う)<br></ul>
```

という文字列を同じように split() にかけると

```
N=4, h[1]="(あ)", h[2]="(い)", h[3]="(う)", h[4]="</ul>"
```

のようになりますので、この場合も最後の `<br>` と `</ul>` を削除してから split() にかけます。よって、

```
sub(/<br>(<br>|<\/ul>)? *$/,"",str)
N=split(str,h,/<br>/)
```

のようにすればいいでしょう。この最初の sub() に与えている正規表現は、

(
|<\/ul>) =
 または
(
|<\/ul>)? = (
 または) の 0 回かまたは 1 回

(
|<\/ul>)? =
 か

 か

(
|<\/ul>)? *$ = そのいずれかにスペースが 0 個以上ついて行末

を意味しています。

7 行の整形出力

ここでは、5 節で説明した (2) の整形して出力する部分を考えます。

Yahoo! の一覧の各 <li> は、例えば以下のようになっています。

```
<li><a href="http://headlines.yahoo.co.jp/...">見出し文字列</a>
  <small> (XXX 新聞) - 15 日 (火)15 時 35 分</small>
```

(実際には 1 行)

このリンクは相対的なリンクではなくて絶対的なリンク (http:// からスタートしている) ですから、これを手元の HTML ファイルとしてブラウザで参照しても、そのリンクをたどって向こうにあるリンク先の記事のファイルにアクセスできますので、この URL の部分は変更する必要はありません。

なお、もしこの URL が、相対的なリンクで以下のように書かれている場合:

```
<li><a href="data/20060815/00003.html">見出し文字列</a>
  <small> (XXX 新聞) - 15 日 (火)15 時 35 分</small>
```

この URL を修正しないと手元に HTML ファイルを保存してもこのリンクをたどって向こうの記事にはアクセスできませんから、http://... の部分をこちらで補う必要があります。

ここでは「整形」として、以下のことを行うこととします。

- <a> タグにオプションを追加する
- いくつかの <li> 列を表示したら、適当に区切りを入れる

<a> タグは、以下のようなオプションをつけた使い方ができます。

- <a href="URL">: URL へのリンク
- <a name="名前">: ページ内リンクのリンク先として名前をつける
- <a href="URL" target="名前">: URL のリンク先を、指定した名前のフレーム上で表示する

この最後の target は、最近はあまり推奨されない機能なのですが、これを利用するとブラウザのウィンドウを 2 つ開いて、一方のウィンドウではリンク元の WWW ページを開いたままもう一方のウィンドウ上でリンク先の WWW ページを参照できます。これは、今回のように沢山のリンクのリストとそのリスト先のページを参照する場合に便利な機能で、リンクの一覧とリンク先をひとつのウィンドウ内で行ったりきたりせずに、多くのリンクの一覧を次々と参照していくことができます。今回は、この target オプションを追加することでその機能を利用することにします。

target オプションの追加は、<a href="URL"> の a と href の間か、"URL" と > の間のどちらかに入れればいいのですが、今回は後者の方に入れてみることにします。これは、その "URL" と > の部分を match() を利用して見きわめて、substr() を利用してそれより前の部分の文字列と後の部分の文字列に分けることでできます。

- match(s,r): 文字列 s の中の正規表現 r にマッチする部分の開始位置 (s の何文字目か) を返す (含まれていなければ 0 を返す)。さらに暗黙の変数 RSTART (= r にマッチする部分の開始位置)、RLENGTH (= r にマッチする部分の長さ) もセットする。
- substr(s,n,len): 文字列 s の n 番目の文字から始まる、長さ len (省略した場合は s の最後まで) の部分文字列を返す。

以下のような正規表現で match() を使うと、a タグ全体の位置を調べることができます。

```
match(str,/<a href=\"[^\"]+\">/)
```

この正規表現は、

[^\"] = " 以外の文字 1 文字
[^\"]+ = " 以外の文字 1 文字以上
\"[^\"]+\" = " と " でくくられた (" が含まれない) 文字列
<a href=\"[^\"]+\"> = 「<a href="文字列">」という文字列

を意味しています。このとき、RSTART+RLENGTH がこのタグの次の文字の位置を意味しますので、

substr(str,1,RSTART+RLENGTH-2) = a タグの '>' の前までの文字列
substr(str,RSTART+RLENGTH-1) = a タグの '>' 以降の文字列

ということになります。よってこれを利用すれば、str に <li> の行が入っているとして、

```
if(match(str,/<a href=\"[^\"]+\">/)==0) print str
else{
    printf "%s",substr(str,1,RSTART+RLENGTH-2)
    printf " target=\"targetframe\""
    printf "%s\n",substr(str,RSTART+RLENGTH-1)
}
```

のようにすれば a タグに target オプションを入れることができるようになります。

いくつかの <li> 毎に区切りを入れるには、j 回の <li> タグの出力が終わった後で

```
if(j%5==0) printf "<br><br>\n";
```

のようなものでも入れればいいでしょう。「j%5==0」は j が 5,10,15,... と 5 の倍数のときにだけ真となりますから、これで 5 つ毎に区切りが入ることになります。さらに、

```
if(j%5==0) printf "<br>(ここまで %d 件)<br><br>\n",j;
```

のようにしてもいいかもしれません。

8 全体のソースコード

以上をまとめた全体のソースコードを紹介しますが、HTML のヘッダやフッタ部分の出力は関数を使って書いています。これらは [4] とほぼ同様ですので、詳細はそちらを参照してください。

6 節で考察した行の分割も、2 箇所に現われるので関数化してありますし、7 節で考察した行の整形出力も、後でカスタマイズできるように関数化してあります。

```
BEGIN{
    if(TARGET=="") TARGET="yahoonews" # ターゲットフレーム名
    if(DIV=="") DIV=5 # 区切りを入れる個数
    N=0 # 一覧の項目数、h[]: 一覧を保存する配列
}
```

```
##### タイトルや日付の取得 #####
(ARGIND == 1 && $0 ~ /<b><font size=\+1>/){
    sub(/.*<b><font size=\+1>/,"")
    sub(/<\/font><\/b>.*/,"")
    TITLE=$0
    getline
    sub(/.*<small> - /,"")
    sub(/<\/small>.*/,"")
    DATE=$0
}
##### 各行の取得 #####
($0 ~ /<ul>/){
    sub(/<ul>/,"")
    if($0 !~ /<li>/) getline
    do{
        # (1) 1 行を複数の <li> 行に分割して配列に保存
        N=divideline($0,h,N)
        getline
    }while($0 !~ /<\/ul>/)
    if($0 ~ /<li>/){
        sub(/<\/ul>/,"")
        # (1) 1 行を複数の <li> 行に分割して配列に保存
        N=divideline($0,h,N)
    }
}
##### END ブロック #####
END{
    putheader(DATE,TITLE,N)
    print "<ul>"
    for(j=1;j<=N;j++){
        put1list(h[j],TARGET)
        if(j%DIV==0) printf "<br>(ここまで %d 件)<br><br>\n",j
    }
    print "</ul>"
    putfooter()
}

##### ユーザ定義関数 #####
# 一つの <li> を整形して出力
function put1list(str,target)
{
    if(match(str,/<a href=\"[^\"]+\">/)==0) print str
```

```
    else{
        printf "%s",substr(str,1,RSTART+RLENGTH-2)
        printf " target=\"%s\"",target
        printf "%s\n",substr(str,RSTART+RLENGTH-1)
    }
}


# <br> で文字列を分割して配列 h に追加 (現在 h[1]~h[N] まで保存)
function divideline(str,h,N,    tmp,j,M)
{
    sub(/<br>(<br>|<\/ul>)? *$/,"",str)
    M=split(str,tmp,/<br>/)
    for(j=1;j<=M;j++) if(tmp[j] ~ /<li>/) h[++N]=tmp[j]
    return N
}

# ヘッダの出力
function putheader(date,title,N)
{
    printf "<html>\n"
    printf "<head>\n"
    printf "<meta http-equiv=\"Content-Type\""
    printf " content=\"text/html; charset=EUC-JP\">\n"
    printf "<title>Yahoo News (%s)</title>",title
    printf "</head>\n"
    printf "<body>\n"
    printf "<h2>Yahoo News (%s: %s : %d 件)</h2>\n",title,date,N
    #printf "<a href=\"%s\" target=\"%s\">(home)</a>\n",url,target
    printf "<hr>\n"
}

# フッタの出力
function putfooter()
{
    print "<hr>"
    print "</body>"
    print "</html>"
}
```

ターゲットフレーム名 (= TARGET) や区切りを入れる個数 (= DIV) は変更が容易にできるようにしてあります。例えば、Yahoo! ニュースの一覧の HTML ファイルが

file1.html,file2.html,...で、上のスクリプトファイルを yahoo2.awk とすれば、

```
awk -f yahoo2.awk file?.html > list.html
```

のようにすれば結合された一覧の HTML ファイル list.html ができますが、

```
awk -f yahoo2.awk -v TARGET="another" file?.html > list.html
```

のようにすれば TARGET や DIV の値を AWK のコマンドラインオプションで変更できます。

なお、関数 putheader() の中で、`charset=EUC-JP` としてありますが、これは私の環境が Unix だからではなく、普通に Yahoo! ニュース記事をダウンロードすると EUC–JP という漢字コードになるからです。MS–Windows 上でも EUC–JP のまま保存した場合は上のままでいいようですが、Shift_JIS で保存する場合は適当にその辺を修正するといいでしょう。

また、ブラウザで HTML ファイルを保存する場合、ブラウザによっては (例えば MS–IE) 単純に保存するのではなく色々加工して保存することもあるようで、そのような場合は今回のスクリプトではうまく処理できないかもしれません。

9 最後に

今回は、AWK で複数のデータファイルを処理する例として、Yahoo! ニュースの一覧を結合した HTML ファイルを作成するスクリプトを紹介しました。

これをさらに加工して分類するためのスクリプトを、スクリプトの分割や 2 重配列の例として紹介したかったのですが、それはまた別の機会に紹介したいと思います。

参考文献

[1] 竹野茂治、「AWK に関する基礎知識」 (2006)

[2] 竹野茂治、「AWK による簡単なタイプ練習ソフト」 (2006)

[3] 竹野茂治、「AWK による数合てゲームの作成」 (2006)

[4] 竹野茂治、「AWK による HTML ファイルの整形」 (2006)

[5] A.V. エイホ、B.W. カーニハン、P.J. ワインバーガー (足立高徳訳)、「プログラミング言語 AWK」、新紀元社 (2004) (元版は 1989)

[6] D.Dougherty、A.Robbins (福崎俊博訳)、「sed & awk プログラミング 改訂版」、オライリー・ジャパン (1997)

[7] 志村拓、鷲北賢、西村克信、「AWK を 256 倍使うための本」、アスキー出版 (1993)

[8] 磯野康孝、蔵守伸一、「HTML ハンドブック」、ナツメ社 (1996)

[9] 大藤幹、「詳解 HTML & XHTML & CSS 辞典」、秀和システム (2002)

[10] 「とほほの WWW 入門」、http://www.tohoho-web.com/www.htm