2006 年 9 月 8 日

AWK による HTML ファイルの整形 その 3

新潟工科大学 情報電子工学科 竹野茂治

1 はじめに

これまで、[4], [5] 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てきました。[4] 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方法について、[5] では、複数の HTML ファイルに分割されているニュースの記事一覧から必要な部分を取り出して一つの HTML ファイルとして作成する方法について紹介しました。そして AWK に関する知識としては、[4] では主にデータから必要な部分を取り出す方法としてsub(),gsub(),match(),substr() の利用法や正規表現、getline を使った読み飛ばしなどを、また [5] では同様の必要な部分の抜き出しと、複数のデータファイルを一度に処理する方法について取り上げてきました。

今回はその続きとして、前回結合したニュース記事の一覧のファイルをさらに加工して、記事をある程度分類する方法を紹介します。もちろん Yahoo! ニュースの方でも社会ニュース、スポーツニュース、政治ニュースのように一応は分類されていて、前回のニュース記事の一覧もその分野のニュースに関するものなのですが、例えば社会ニュースなどの場合 1 日に 200 件近くの流量があり、前回作成した 1 つのファイルでの記事の一覧も単に時系列順にニュースを並べたものなので、例えばある特定のニュースの続報を見る、といった場合にそれを探すのが大変です。

よって、今回は社会ニュースなら社会ニュースを、キーワードによるマッチングの手法を使って、さらにある程度ジャンル毎に分類する方法について紹介します。今回は AWK の 2 次元配列やスクリプトの分割 (ライブラリ化) についても紹介する予定です。

今回紹介する方法は、ほぼ私が普段ニュース記事を参照するのに利用している方法ですが、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。

また、今回は [5] で作成したニュース記事の一覧ファイルを対象としますので、それで作ったファイルが必要となります。詳しくは [5] を参照してください。

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 前回作成した一覧ファイル

前回 [5] で作成した一覧ファイルは、基本的に Yahoo! ニュースの記事の一覧から ∼ タグで囲まれている部分のみを抜き出したもので、たくさんの記事を持つニュース分野ではそれが複数の HTML ファイルに分かれているのですが、それらを結合して一つの HTML ファイルとしたものです。

その作成した一覧ファイルは以下のような構造になっています。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-JP">
<title>Yahoo News (社会ニュース)</title></head>
<body>
<h2>Yahoo News (社会ニュース: XX 月 XX 日 (金)XX 時 XX 分 : 181 件)</h2>
<hr>
<ul>
<li><a href="http://headlines.yahoo...." target="yahoonews">
  どこそこで交通事故</a><small> ( XXXX 新聞 )
  - 9 時 0 分</small>
<li><a href="http://headlines.yahoo...." target="yahoonews">
  どこそこで火事</a><small> ( XXXX 新聞 )
  - 8 時 50 分</small>
<li><a href="http://headlines.yahoo....." target="yahoonews">
  だれそれを指名手配</a><small> ( XXXX 通信 )
  - 8 時 40 分</small>
<li><a href="http://headlines.yahoo....." target="yahoonews">
  target="yahoonews">某氏、いくら脱税</a><small> ( XXXX 通信 )
  - 8 時 30 分</small>
<li><a href="http://headlines.yahoo....." target="yahoonews">
  target="yahoonews">だれそれに無罪判決</a><small> ( XXXX 通信 )
  - 8 時 10 分</small>
<br>(ここまで 5 件)<br><br>
<li><a href="http://headlines.yahoo....." target="yahoonews">
  target="yahoonews">だれそれに無罪判決</a><small> ( XXXX 通信 )
  - 8 時 0 分</small>
....
```

```
<br>(ここまで 180 件)<br><br>
<li><a href="http://headlines.yahoo....." target="yahoonews">
  target="yahoonews">＜地震＞ほにゃらかで震度４</a><small>（XXXX 通信）
  - 17 時 20 分</small>
</ul>
<hr>
</body>
</html>
```

上では URL の部分を後半は省略していますし、 の行は 3 行位に折り返して書いていますが、基本的に各 <li> で始まる行は</small> までが 1 行で書かれていて、デフォルトでは 5 件毎に「(ここまで XX 件)」という行が挿入されています。

各項目のニュース記事へのリンク (URL) や target の部分はそのまま利用できますからこれらはそのままにして、この 200 件近い記事を、例えば「自然災害」「交通事故」「訃報」「その他の事件」のようなジャンルに分類することを考えてみます。

見出しは、短い文章で内容を伝えるもので、特に代表的なニュースなどの場合はそれを知っている人には通じる、といったような書き方がされることもありますので、見出しだけで厳密な分類は行えませんが、見出しに含まれるキーワードを利用して、あまり精度は高くはないかもしれませんが AWK でそれを自動的に分類することを考えます。

キーワードとは、例えば「火事」「逮捕」「判決」「死去」などのようなジャンル分けに利用できる文字列のことで、例えば「死去」は、本来は単に人が死んだことを指しますが、これは通常殺人事件や交通事故が主のニュースには用いられず、見出しとして使われる場合は重要人物の訃報の際に用いられることが普通です。よって、見出しが「死去」にマッチした場合はそれを「訃報」というジャンルに分類する、といった手法を取ります。いずれにもマッチしなかった場合は「その他」のジャンルに入れていきます。

このようなキーワードパターンは、あらかじめよく考察した上で設定するというよりも、実際に数日間運用しながらそのパターンを修正、追加、削除していく方がてっとりばやく、また精度もそれなりに向上します。私の場合も、最初は「その他」の記事がかなり多かったですし、誤認識もかなりありましたが、1 週間程度運用しながら修正を行なうことでだいぶ改善され、誤差等がさほど気にならない程度にまでなりました。

後で私が使用している分類キーワードパターンの一部も紹介したいと思いますが、これらはどのようなジャンル分けを行うかにもよりますし個人差もありますので、自分なりのキーワードパターンを設定する必要があるだろうと思います。

3 全体のおおまかな構造

今回は、必要な行の取得は単に <li> で始まる行を拾っていけばいいだけですが、それを分類して出力しなければいけません。しかし、取得した時点でそれがどのジャンルに入るかは分類できたとしても、その時点で出力してしまったのではやはり時系列順に出てしまうことになりますので、各ジャンル毎に、

そのジャンルの 1 番目の記事、そのジャンルの 2 番目の記事、...

のように記事を保存しておく必要があります。そして記事全体を読み込んだ後で、それらを各ジャンル毎に出力していくことになります。

つまり、ジャンルが複数存在して、その各ジャンルに記事が複数存在することになりますが、そのような記事の保存のために 2 次元配列を利用することにします。

2 次元配列とは、配列の添え字を 2 次元的に添え字付けられる配列のことを言います。AWK の配列は連想配列で、元々添え字に文字列も使用できますから、それを工夫すれば、例えば

```
a["1x1"], a["1x2"], a["2x1"], a["2x2"], ...
```

のようにして容易に 2 次元配列を作れるのですが、AWK 自体に 2 次元配列という仕組みが用意されています。

C 言語では、2 次元配列は a[i][j] のように表現しますが、AWK の 2 次元配列は a[i, j] のように添え字を ‘,’ で区切って指定することになっています[1]。

よって、例えば、

```
a[1,1]=1; a[1,2]=2; a[2,1]="x"; a[2,2]="y";
```

のような使い方ができますし、文字列を添え字にして、

```
a["abc",3]=3
```

のように使うこともできます。

今回は「i 番目のジャンルの j 番目の記事」を hs[i,j] という 2 次元配列に保存することにします。

これを利用すれば、全体の AWK の疑似コードは以下のように書けます。

[1]実際には、これは上の文字列引数による 2 次元配列の実現とほとんど同じことをやっています。

```
BEGIN{
    # NG = 全ジャンル数
    # hn[j] = j 番目のジャンルの記事数（1<=j<=NG+1）
    #    ((NG+1) 番目のジャンルは「その他」の記事）
    # hs[j,k] = j 番目のジャンルの記事を保存する配列（1<=k<=hn[j]）
    #
    # (1) j 番目のジャンルのキーワードパターン pat[j] を定義（1<=j<=NG）
}
($0 ~ /^<h2>/){ title=$0 }
($0 ~ /^<li>/){
    # (2) <li> 行から見出し部分だけを取り出す（= str とする）
    for(j=1;j<=NG;j++){
        if(str ~ pat[j]){
            hn[j]++
            hs[j, hn[j]]=$0
            break
        }
    }
    if(j>=NG+1){   # いずれにもマッチしなかった場合。この場合 j==NG+1
        hn[j]++
        hs[j, hn[j]]=$0  # 「その他」に保存
    }
}
END{
    # (3) HTML ヘッダ等の出力
    for(j=1;j<=NG+1;j++){
         # (4) hs[j,1] ~ hs[j,hn[j]] を出力
    }
    # (5) HTML フッタ等を出力
}
```

($0 ~ /^<h2>/) は <h2> で始まる行に対する処理を意味しますが、これは /^<h2>/ とだけ書いても構いません。この見出しの行は title という変数に保存していて、最後のヘッダの出力で利用します。

($0 ~ /^<li>/) のブロックが 各 <li> 行に対する処理ですが、この中で各ジャンルにマッチするかをパターンマッチングを利用して調べています。正規表現とのパターンマッチングは、通常は

```
str ~ /正規表現/
```

のように書くのですが、正規表現をあらかじめ変数 (例えば pat) に保存しておいて、

```
str ~ pat
```

のように書いても同じことになります[2]。この場合、変数 pat に代入する文字列には /pat/ は含めませんし、str ~ /pat/ とも書きません。逆に str ~ /pat/ と書いてしまうと "pat" という文字列とのマッチングを意味してしまいます。

パターンにマッチした場合には、そのジャンルに追加するために記事数を一つ増やし (hn[j]++)、記事をそのジャンルの配列の最後に追加しています (hs[j,hn[j]]=str)。その場合はそれ以降のマッチングをやる必要がないので break で for 文を抜けだしています。

いずれにもマッチしなかった場合は、for 文のインデックスの j が $(NG+1)$ になっているはずですから、それを判別してその $(NG+1)$ 番目のジャンル (「その他」) に保存しています。

なお、for 文の break を next に置きかえれば、いずれかにマッチすれば for 文の次の文に進むことはなくなりますので、この for 文の次の if によるチェックは必要ありません。今回も最終的にはそのように書くことにします。

(3) や (5) のヘッダやフッタの出力は、[4] や [5] と同様の関数を使って簡単に出力できます。(2) の部分も、[4] や [5] で述べたように、sub() などを使って容易に取り出すことができます。(4) の部分もほぼそのまま並べるだけなので、よって考えるべき部分は (1) のみ、となります。

4 パターン部分の定義

3 節で紹介したように、最初の BEGIN ブロックでは、各ジャンルに対するキーワードパターンなどを定義します。必要なのは以下のものです。

- 全ジャンル数 $(= NG)$
- 各ジャンルのキーワードパターン $(= pat[])$
- 各ジャンルの見出し名 $(= Gname[])$

最後の見出し名は、HTML ファイルの各ジャンルのところのセクションの見出しとして使用する文字列を意味します。

キーワードパターンは、基本的に str ~ pat[j] のようにパターンマッチングで使用されるので正規表現で書いていきます。ここでは、or のタイプの正規表現パターンを書いていきます。例えば、「自然災害」というジャンルなら、

[2] [6] には、訳注ながら 動的正規表現と書かれています。

地震、台風、豪雪、豪雨、洪水、落雷、雪崩、...

などのキーワードが思い浮かびますが、これらのいずれかにマッチする記事を選べばいいわけですから、or のタイプの正規表現を使用します。or のタイプの正規表現は、以下のように書きます。

(A|B) = A または B
(A|B|C) = A または B または C
....

この A,B,C の部分には文字列が使えますし、入れ子にすることもできます。例えば、

(地震|豪雪) = 「地震」または「豪雪」
豪 (雪|雨) = 「豪雪」または「豪雨」
(地震|豪 (雪|雨)) = 「地震」または「豪雪」または「豪雨」

のような使い方が可能です。

これによって、パターン定義部を書けば例えば以下のようになります。

```
BEGIN{
  NG=0
  ### 自然災害 ###
  Gname[++NG]="自然災害関連"
  pat[NG]="("
  pat[NG]=pat[NG] "(地|余) 震|震 (災|度|源)|(災|水) 害|(豪|大)(雨|雪)"
  pat[NG]=pat[NG] "|台風|被災|崩落|落雷|警報|避難|洪水|雷雨|濃霧"
  pat[NG]=pat[NG] "|土砂崩|雪崩|(突|強) 風|竜巻|陥没|倒木"
  pat[NG]=pat[NG] ")"

  ### 火事 ###
  Gname[++NG]="火事関連"
  pat[NG]="("
  pat[NG]=pat[NG] "(放|出|不審) 火|火 (事|災)|(全|半) 焼|ぼや"
  pat[NG]=pat[NG] ")"

  ....

  ### 訃報 ###
  Gname[++NG]="訃報"
  pat[NG]="("
```

```
    pat[NG]=pat[NG] "訃報|通夜|葬儀|告別式|死去"
    pat[NG]=pat[NG] ")"

    ### その他 ###
    Gname[NG+1]="その他"
}
```

上のパターンの中の、(豪|大)(雨|雪) の部分は、「豪雨」「豪雪」「大雨」「大雪」の 4 つの文字列のいずれかを意味しています。

一般にパターンは今後の修正により長くなったり、後で別ジャンルを挿入したり削除したりする可能性がありますから、上にはそれが楽にできる工夫が含まれています。

例えば、上では最初に NG=0 として、$Gname$ の定義の中で Gname[++NG]=... としていますが、これを Gname[1]=... と書いてしまうと、これは最初のジャンルに固定したことになってしまいますので、1 という数字を使わずに何番目のジャンルでも同じように書けるようにしていて、いつでもジャンルの追加、削除、移動等が可能なようにしています。

また、例えば「訃報」のところの $pat[]$ の定義ですが、

```
pat[NG] = "(訃報|通夜|葬儀|告別式|死去)"
```

と 1 行で済むところをあえて

```
pat[NG]="("
pat[NG]=pat[NG] "訃報|通夜|葬儀|告別式|死去"
pat[NG]=pat[NG] ")"
```

のように 3 行に分けて書いているのは、キーワードの追加、削除、修正等をしやすくするためです。

最初の「自然災害」のジャンルを見ればわかりますが、キーワードが長いので数行分けて書いています。最初は少ないキーワードでも、後でどんどん追加して長くなることがあります[3]。その場合、最初のカッコと最後のカッコを除けば '|' とキーワードを追加するだけなので、追加が容易です。

もしそのカッコを除外しておかないと、先頭のカッコや最後のカッコをうっかり削除したり、つけ忘れたり、真中に残してしまったり、ということがある可能性がありますので、最初と最後にそれを別につけてやる、という形にしてあります。

なお AWK では文字列の連結は、文字列を並べるだけでできますので、

[3]実際、「自然災害」のところの項目は、順次運用しながら追加していったものです。

```
pat[NG]="("
pat[NG]=pat[NG] "訃報|通夜|葬儀|告別式|死去"
pat[NG]=pat[NG] ")"
```

は、すなわち

```
pat[NG] = "(訃報|通夜|葬儀|告別式|死去)"
```

を意味しています。

5 分野依存部分の分離

前回 [5] で作成したスクリプトは、スクリプト自体は、対象となるニュースの分野、すなわち社会ニュースであるかコンピュータニュースであるかには依存していませんでした。

今回のスクリプトは、最初の BEGIN ブロックには分野に特化したパターンを定義する必要がありますので、もちろん分野に依存しているのですが、3 節の全体のソースコードを見ればわかりますが、その BEGIN ブロック以外は分野には依存していません。よって、BEGIN ブロックだけすげかえれば、他の部分は社会ニュース用、コンピュータニュース用などの他の分野にも使いまわしができます。

このような場合てっとりばやいのは、実際にファイルをコピーして、その BEGIN ブロック部分だけ書き直す、という方法ですが、そういう方法だと、共通部分にバグがあって修正するとか、改良するとかという場合に、すべてのスクリプトの同じ箇所を修正する、改変する、ということをしなければなりません。

しかしそれ以外に、「スクリプトを分割」して、共通部分は一つのファイルにして共有する、という手があります。C 言語で言えば「ライブラリ化」ということに相当します。

AWK では、複数のスクリプトを複数の -f オプションで読み込ませることができます。

```
awk -f script1.awk -f script2.awk ... datafile
```

複数のスクリプトファイルは、それらが連結された一つの大きなスクリプトであるとみなされます。

逆に言えば、今回のスクリプトを BEGIN ブロックとそれ以降のスクリプトに分離すれば、それ以降の部分は他のスクリプトでも上のようにすることで共通に利用できます。

また、複数のスクリプトで BEGIN ブロックなどをそれぞれ持つこともできますが、それもそれらをつなげた一つのスクリプト内に複数の BEGIN ブロックがあることと同じ

で、データを読み込む前に、最初に現われた BEGIN ブロックから順に BEGIN ブロックの処理が行われていくだけです。

6 全体のソースコード

以上をまとめて、全体のソースコードを紹介します。

ここでは、5 節で紹介した共通部分のみ紹介します。各分野に依存した部分のソースコードは、4 節で紹介した BEGIN ブロックがそれになります。

3 節で紹介したおおまかなソースコードからは、多少拡張してある部分もあります。

```
##### ジャンル分け共有部分 #####
## おのおの分野に依存する部分では、
## NG, p[j] (1<=j<=NG), Gname[j] (1<=j<=NG+1) を定義しておくこと。
##
BEGIN{
    if(DIV=="") DIV=5  # 区切りを入れる個数
}
/^<title>/{ headtitle=$0; next } # HTML ヘッダの <title> 行を取得
/^<h2>/{ title=$0; next } # HTML 本文のタイトル部分を取得
/^<li>/{
    str=$0
    sub(/^.*yahoonews\">/,"",str)
    sub(/<\/a>.*/,"",str)
    for(j=1;j<=NG;j++){
        if(str ~ pat[j]){
            hn[j]++
            hs[j, hn[j]]=$0
            next
        }
    }
    hn[j]++
    hs[j, hn[j]]=$0
}
END{
    putheader(headtitle,title)
    puttoc(NG,hn,Gname) # 目次の出力
    putgenre(NG,hn,hs,Gname)
    putfooter()
```

```
}

##### ヘッダ出力 #####
function putheader(headtitle,title)
{
    printf "<html>\n"
    printf "<head>\n"
    printf "<meta http-equiv=\"Content-Type\""
    printf " content=\"text/html; charset=EUC-JP\">\n"
    printf "%s\n",headtitle
    printf "<body>\n"
    printf "<a name=\"top\"></a>\n" # 先頭へのページ内リンク
    printf "%s\n",title
    printf "<hr>\n"
}
##### 目次出力 #####
function puttoc(NG,hn,Gname,    j)
{
    printf "<h2><a name=\"toc\">目次</a></h2>\n"
    printf "<ul>\n"
    for(j=1;j<=NG+1;j++)
      printf "<li><a href=\"#S%d\">%s</a> (%d 件)\n",j,Gname[j],hn[j]
    # S1, S2, 等を各ジャンルへのページ内リンクとする
    printf "</ul>\n"
    printf "<a href=\"#top\">先頭へ戻る</a>\n"
    printf "<hr>\n"
}
##### 各セクションの一覧出力 #####
function putgenre(NG,hn,hs,Gname,    j,k)
{
    for(j=1;j<=NG+1;j++){
        printf "<h2><a name=\"S%d\">%s</a>",j,Gname[j]
        # ページ内リンクの定義
        printf " (%d 件)</h2>\n",hn[j]
        printf "<ul>\n"
        for(k=1;k<=hn[j];k++){
            printf "%s\n",hs[j,k]
            if(k%DIV==0) printf "<br>(ここまで %d 件)<br><br>\n",k
        }
        printf "</ul>\n"
        printf "<a href=\"#top\">先頭へ戻る</a>\n"
        printf "<hr>\n"
```

```
    }
}
##### フッタ出力 #####
function putfooter()
{
    printf "</body>\n"
    printf "</html>\n"
}
```

3 節に対する拡張、修正は、

- <title> 行の取得 (分野に依存しない出力のために元のタイトルを取得)
- DIV を導入して各分野毎に DIV 記事毎の区切りを入れる
- 先頭に各ジャンルにジャンプできる目次を出力 (puttoc())
- 各ジャンルの出力全体を関数化 (putgenre())

等です。

最後の putgenre() は、3 節のように END ブロック内に for 文を書いて、各ジャンル毎の出力関数を作ってそれを使うという手もあるのですが、いずれにせよ 2 重配列 $hs[]$ をそっくり関数に渡してしまうことになるので、それなら for 文もその関数の中に入れてしまって全部出力させても同じことになりますからそのようにしました。

7 最後に

今回は、実際に私が利用しているジャンル分けの方法を紹介しましたが、実は私は 6 節のソースをさらに多少拡張したものを利用しています。それは、

- あるジャンルの記事が 0 件ならば、それは目次には置くが、本文は作成しない (正確にはそうするかどうか、目次にも置くかどうかを選択できるようにしている)
- マッチするパターンだけではなく、マッチした場合に排除する排除パターンも用意している

のような点です。

前者は $hn[j]$ の値が 0 かそうでないかで条件分岐すればいいだけですが、この後者は、$pat[j]$ とともに $npat[j]$ (多くは空文字列) を用意しておいて、パターンマッチのところを

```
if((pat[j]=="" || str ~ pat[j]) && (npat[j]=="" || str !~ npat[j]))
```

のようにしています。これは、$npat[j]$ が空文字列なら通常と同じですが、そうでなければ

「str が $pat[j]$ にマッチして」かつ「str が $npat[j]$ にマッチしない」場合

を意味します。これによって、「$pat[j]$ にはマッチするんだけどそのジャンルには入れたくないもの」を取り除くことができます。取り除きは、取り除く方のジャンルを先に持ってきて、そちらでそのような記事を先に釣り上げる、という手もあるのですが、セクションの順番をそのような目的で入れかえるのは問題もありますし、順番を入れかえられない事態もあり得ると思いますので、そのために排除型のマッチングも用意しています。

他にも、ジャンル分けの精度を上げる方法や、各ジャンルの記事数をより偏らないようにする方法、一時的なジャンルの追加を認める方法など、色々な拡張が考えられると思います。AWK の利点を生かして、色々な発展を考えてみたらいいのではないかと思います。

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] 竹野茂治、「AWK による HTML ファイルの整形 その 2」(2006)

[6] 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)

[7] 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)

[8] 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)

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

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

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