平成 18 年 8 月 14 日

AWK による HTML ファイルの整形

新潟工科大学 情報電子工学科 竹野茂治

1 はじめに

今回は、データの加工例として、HTML データ、特に Yahoo! のニュース記事を手元で加工する方法について紹介します。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。

多くの方がご存じのように、Yahoo! (`http://www.yahoo.co.jp/`) には各新聞社などが提供するニュース[1]を紹介しているコーナーがあります。

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

ここには、毎日のニュース記事がカテゴリ別 (社会、国際、スポーツなど) に掲示されていて、私もよく利用しています。

しかし、私が普段利用している古い (がしかし軽快な) ブラウザ Netscape 3.04 で見るときに不便に感じる点がいくつかあります。

1. 一覧と各記事の行ったりきたりが不便
   ニュース記事の一覧からリンクを開くと、ブラウザの同じウィンドウにリンク先の記事が表示されるので、見たい記事を探すためにまた同じウィンドウ内の一つ前の一覧のページに戻らないといけません。

   ニュース記事のページには前後 2 つずつ位の記事へのリンクも貼られているのですが、やはり一覧全体を見て記事を探したいので、そちらに戻ることになります。

   最近のタブブラウザならば、過去にアクセスしたページとの切り替えは軽快に行えるのですが、Netscape 3.04 はそうではないので、一覧に戻ろうとするとまたそこでネットワークにアクセスして一覧のデータを再び取得しようとしてしまいます。

2. 記事の並び方
   ニュース記事は、時間順に並んでいるので、ある記事の続報がすぐその次に並ぶとは限りません。特に社会ニュースのように記事数の大変多いカテゴリでは、その続報や関連する分野のニュースを見たい場合にそれを探すのが大変です。

[1] ちなみに、インターネットには「ネットニュース」というものもありますが、これはこの Yahoo! などが紹介している「ニュース記事」とは全く意味が違う別のものです。

3. 一覧自体が複数ページ
   ニュース記事の一覧には、その一覧だけではなくて、広告、最新のニュース、天気予報など、色々不要な情報が含まれていて、それで一覧のページに書ける情報量が少なくなっているためか、1 日のニュースの一覧自体が複数のページに分割されてしまっています。

   例えば社会ニュースのように 1 日 100 件以上のニュースが流れているところは、一覧自体が 6 ページ位に分かれていて、一覧の全体を見るのにもいちいちページの切り替えとネットワークアクセスが必要となります。

4. 記事のページの情報量
   一覧だけではなく、記事自体のページにも色々なリンクや広告などが含まれていて、ニュースの内容以外のものがページ内のある割合を占めています。これは、そのニュース記事を印刷するときにも無用にページ数が増えてしまう原因になります。

私はこれらを解消するために、一覧の記事全体を手元に一旦取得してから、csh のスクリプトや AWK のスクリプトを利用してそれらを加工して新たな一覧を作って、上のような問題を解消するような方法でニュースの閲覧を行っています。

今回はこの 4. の、記事のページから不要なものを削除する、という方法について紹介します。

なお、この 4. に関しては、ブラウザで記事の内容を見るときにこれを行っているわけではありません。つまり見るときは通常の広告等の入った記事を見ていますが、それを印刷するときにそのページを一旦手元に保存して、今回で紹介するような AWK スクリプトを使って加工して、記事の内容だけを取りだしたものを印刷する、という手法を使っています。

もし、記事を閲覧するときにも広告等を削除したものを見るようにするには、すべての記事を一旦手元に持ってきて、それらをすべて加工してから閲覧する、ということをしなければいけませんが、一覧はすべて目を通しますがニュース記事は必ずしもすべての記事を読むわけではありませんから、それは不要な記事まで持ってくるための不要なネットワークアクセスを増やすことになりますので、そういう風にはしていません。

また、今回は 1., 2., 3. の手法、すなわち一覧を取得してそれを加工する、という方法については説明しませんが、これは別の機会に紹介したいと思います。

2 HTML のおおまかな仕組み

HTML とは、HyperText Markup Language の略で、WWW ページ文書を書くための書式を決めたものです。我々が普段見ている WWW ページ (何とか `.html` の形式の

ページ) はほぼすべてがこの規格に従って書かれています。

HTML は、文書の見た目の構造や論理的な構造をタグと呼ばれるマークアップ (印付け) によって示すやり方[2]を取っていて、文書の中にこのタグを含めて書いていきます。

HTML のタグは、

- <タグ名>それが影響を受ける文書</タグ名>

のように、ペアのタグで文書の部分を囲むものと、

- <タグ名>

のように、単独のタグで意味を持つものの 2 種類に大きく分かれます。

HTML 文書の基本的な構造は、以下のようになります:

```
<html>
<head> <title>タイトル</title>
</head>
<body>
本文
</body>
</html>
```

上にも見られますが、いくつか HTML 文書での決まりを上げます。

- 文書中の改行は入れても入れなくても表示とはあまり関係はなく、HTML ファイルの本文中の改行は表示ではスペース一つと同じ意味になる (強制的に改行するには明示的に改行タグを書き入れる)
- タグ名のアルファベットは大文字でも小文字でも構わず、同じ意味になる[4]
- タグで囲んだ文書中にまたタグを使うこともできるが、その場合内側のタグ部分を終了させてから外側のタグ部分を終了させるように書く
- タグにはオプションをつけることができるものもあり、その場合は、<タグ名 オプション名="その値" ...> のように書く。

[2]数学者が論文を書くのによく用いられる LaTeX と呼ばれる文書形式[3]もマークアップ方式で、よって私は HTML のタグ方式での文書の記述に特に不自然さは感じませんが、最近は生の HTML 文書を手書きせずに HTML ファイルを作れるツールがよく使われているかもしれません。

[3]この文書自体も LaTeX で書かれています。

[4]HTML の新しい規格では小文字に統一されるそうです。

次に、上で示したいくつかのタグの意味を紹介します。

- <html>～</html>: HTML 部分を明示するタグ。通常 HTML ページ全体がこれで囲まれる[5]。その中味はヘッダと本文に大きく分かれる。
- <head>～</head>: HTML のヘッダ部分を囲むタグ。ヘッダにはタイトルや著者、使用言語などが書かれる。この部分はブラウザには表示されない。
- <title>～</title>: この HTML ファイルのタイトルを定義するタグ。このタイトルは本文に表示されるわけではなく、通常はブラウザの上部のタイトルバーに表示されるか、ブックマークの登録時にこの WWW ページの名称として利用される。
- <body>～</body>: HTML 本文を囲むタグ。オプションとして本文の背景色などを指定することができる。基本的にこの部分がブラウザに表示される。

この他にも、文字の大きさ、改行、見出し、箇条書きなどに関する多くのタグがありますが、今回利用する HTML ファイルに含まれる主要なタグを以下に紹介します。

-
: 強制改行
- <hr>: 水平線 + 改行
- ～: タグで囲んだ部分を太字にする
- ～: オプションを使って、タグで囲んだ部分の表示に使用するフォント、サイズ、色などを指定する
- <small>～</small>: タグで囲んだ部分を小さな文字にする
- ～ : タグで囲んだ部分を中央揃えする
- <h[n]>～</h[n]>: タグで囲んだ部分を見出しとする ([n] は 1 から 6 まで、1 が最も大きい見出し)
- <a>～: オプションを使って、タグで囲んだ部分にリンクを貼ったり、逆にここをリンク先とするための名前をつけたりする
- ～ : タグで囲んだ部分をブロックとして、オプションの値に従って位置合わせなどを行う
- <table>～</table>: タグで囲んだ部分を表として作成する
- <tr>～</tr>: <table> タグ内の表の行の要素を意味する
- <td>～</td>: <tr> タグ内の列の要素を意味する

[5]HTML の新しい規格では、この前に xml 宣言や DOCTYPE 宣言が置かれることもあるようです。

- <script>～</script>: Javascript 等の定義部分
- <noscript>～</noscript>: Javascript 非対応のブラウザ用の定義部分
- <form>～</form>: 入力フォーム定義部分
- <img>: オプションで指定した画像を貼りつける
- <!-- ～ -->: コメント (表示や機能とは無関係)

HTML で利用可能なタグの一覧や詳細については、HTML のタグ辞典 ([7], [8] など) や、HTML タグを紹介するサイト ([9] など) などを参照してください。

3 Yahoo! ニュース記事の構造

Yahoo! ニュース記事の HTML ファイルでは、特に画面を分割して情報を配置するために<table> タグ (及びその中で使用される <tr>, <td> タグ) が多用されています。これは画面内に複雑なレイアウトで色々な情報を盛り込んでいる WWW ページの特徴です[6]。ニュース記事の内容自体も、ある <table> タグの要素になっています。

Yahoo! ニュースの HTML ファイルの構造全体の詳細を述べることはできませんが、少なくとも今回必要な情報が含まれている部分に関しては、おおよそ以下のようになっているようです。

```
<html>
<head>
<!--  -->
<title>Yahoo!ニュース
 - XXXX 新聞 - 隣りの空き地に囲いができた
</title>
........
</head>
<body marginheight=0 topmargin=0>
<center>
........
<!--- CONTENTS_TITLE_TABLE --->
<table border=0 cellpadding=2 cellspacing=0 width=100%>
<tr bgcolor="#9999cc">
<td nowrap>
```

[6]最近は table タグによらずに CSS などの仕組みを利用してこのような配置を行うことも多いようです。

```
<b><font size=+1>社会ニュース</font></b>
  <small> - 8月10日(木)12時00分</small>
</td>
........
<!--- /CONTENTS_TITLE_TABLE --->
<!--br-->
<!--- OUTLINE_TABLE --->
<font size=5 class="s130"><b>隣りの空き地に囲いができた</b></font>
  <br><br>
(記事の本文)
... <div align=right>
（XXXX 新聞） - 8月10日12時00分更新</div><br>
</td></tr>
........
... <!--- /YBB module --->
<hr width=100% size=0>
<small>
<a href="http://help.yahoo.co.jp/help/jp/news/">ヘルプ・お問い合わせ
  </a><br>
Copyright (C) 2006 XXXX 新聞社
 記事の無断転用を禁じます。<br>
Copyright (C) 2006 Yahoo Japan Corporation. All Rights Reserved. <br>
  </small>
</center>
</body>
</html>
........
```

今回は、この中から不要な部分を削除して、必要な情報のみを取り出すことにします。必要な情報とは、上の中の以下の部分であるとします。

1. <title>～</title> の内容
2. <!--- CONTENTS_TITLE_TABLE ---> の後ろにある<b><font size=+1>...</small> の部分
3. <!--- OUTLINE_TABLE ---> の後ろにある記事の本文
4. <!--- /YBB module ---> の後ろにある「Copyright (C) 2006 ...」の部分

出力は、必要な部分が見つかったら随時その時点で出力する、という方法もありますが、今回は、必要な部分を一旦変数に保存しておいて、HTML ファイルを全部読み終

わった後で出力することにします。このようにすることで、元の記事内の配置によらない出力を行うこともできますし、出力の前に必要な部分があるかどうかのチェックなどを行うこともできるようになります。全部読み終わると AWK は END ブロックに処理を移しますから、出力はこの中で行うことにします。

4 目標となる行の取り出し

この節では、3 節で紹介した必要な部分のうち、例えば 1. の <title>〜</title> の行を取得する方法について考えてみます。

3 節で見たように、<title>〜</title> 部分は複数行でできているようです。このサンプルでは 3 行ですが、実際にはもっと長かったり短かったりするかもしれません。<title>, </title> タグ自体もサンプルではそれらが単独の行になっていますが、実際には <title> の次に改行なしですぐに内容の文が続いたり、</title> の後ろに何らかの文字が続いているかもしれません。

一方で AWK は基本的に行単位のフィルタで、入力を 1 行ずつ読み込んで処理する形になっています。よって、このように処理の単位が複数行である場合は、以下のような 2 種類の処理の方法が考えられます。

1. 対象となる部分の先頭の行を読みこんだら、getline を使ってその対象部分の終りまでを一気に読み込んでしまって処理を行う
2. 対象となる部分の先頭の行を読みこんだら、現在その対象の部分内であるという目印 (フラグ) をつけ、終りの行に達したらその目印をはずしてその処理を行う

getline は新たな行を取得するときに使います。

- `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。

これらを AWK の (疑似) コードで書くとそれぞれ以下のようになります。

- getline を使う方法:

```
($0 ~ /<title>/){
     N=0
     h[++N]=$0
     while($0 !~ /<\/title>/){
```

```
            getline
            h[++N]=$0
        }
        # h[1] ~ h[N] までに保存されているのでその処理をする
    }
```

- フラグを使う方法:

```
($0 ~ /<title>/){ flag=1; N=0 }
(flag==1){
    h[++N]=$0
    if($0 ~ /<\/title>/){
        flag=0
        # h[1] ~ h[N] までに保存されているのでその処理をする
    }
}
```

現在の入力行全体は $0 で、文字列が正規表現にマッチするかどうかは ~, !~ で調べます。

- 文字列 ~ /正規表現/ : 文字列が正規表現にマッチすれば真
- 文字列 !~ /正規表現/ : 文字列が正規表現にマッチしなければ真

正規表現は / / で囲んで指定します。正規表現のパターン内に / を書く場合は、区切り記号と区別するために \ をつけて \/ と書きます (エスケープ)。

上のサンプルでは、配列に各行を保存していますが、または <title>～</title> を一つの文字列として保存する手もあります。その場合は、上の “h[++N]=$0” の代わりに、例えば “str = str $0” のようにします。AWK では 2 つの文字列を並べると、それで文字列の連結を意味します。

また、getline を使う場合は、万が一 </title> を含む行が見つからなかった場合を考えて (その場合最後の行まで一気に取得してしまう)、単なる getline の代わりに

```
if(getline<=0){ errorexit=1; exit }
```

のようにしておくといいかもしれません。exit は、入力の読み込みをやめ、END ブロックがあればそこにジャンプするだけなので、その前に errorexit などのような変数 (変数名は何でも構いません) をセットすることで、END ブロックの先頭に

```
END{
  if(errorexit){
       printf "エラー発生 (code = %d)\n",errorexit > "/dev/stderr"
       exit
  }
  ....
```

と書いておけば END ブロックの他の処理を実行せずにエラー終了することができます。なお、この printf の後ろについている「`> "/dev/stderr"`」は本来は Unix に由来する記法で、標準エラー出力への出力を行うための書き方です。こう書けば AWK の出力をファイルなどにリダイレクトしても、この出力はリダイレクトされずに画面に表示されます。

5 必要な部分文字列の取得

4 節では、必要な部分を行単位で保存する方法について説明しましたが、この節では、それを文字列として連結した後で、タグ以外の必要な部分を取得する方法について説明します。1. の `<title>`〜`</title>` の内容の取得などでこれを利用しますので、これについて考えてみます。

AWK ではこのような場合、以下の 2 通りのやり方があります。

1. 必要な部分を含む行の、不要な部分を sub(), gsub() を使って削除する
2. 必要な部分を含む行の、必要な部分文字列を substr() で取り出す (match() や index() も使用)

使用する関数の意味は以下の通りです。

- `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
- `gsub(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 にマッチした全ての部分を文字列 s で置きかえる
- `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
- `index(s,c)`: 文字列 s の中の最初の部分文字列 c 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)

- `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)

`match()` は、さらに暗黙の変数 RSTART と RLENGTH もセットし、それぞれ

- RSTART = r にマッチする部分の先頭位置
- RLENGTH = r にマッチする部分の長さ

となります。例えば、

```
s="54321"
match(s,/[2-4]*/)
```

とすると、RSTART=2, RLENGTH=3 (つまり ”432” がマッチした部分) となります。なお、この正規表現は、

`[2-4]` = 2,3,4 のいずれか、
`[2-4]*` = 「2,3,4 のいずれか」の 0 文字以上の連続

を意味します。正規表現は最長一致 (なるべく長く一致) しようとしますので、上のパターンに一致するのは ”432” ということになります。そして、この直後に

```
c=substr(s,RSTART,RLENGTH)
```

とすれば、c に ”432” が保存されるわけです。

さて、

```
s="<title>タイトル</title>"
```

である場合、この「タイトル」の部分のみを得るには以下のようにすればできます。

1. sub(), gsub() を使う場合
   この場合は、

   ```
   sub(/<title>/,"",s)
   sub(/<\/title>/,"",s)
   ```

とすると s=”タイトル” となります。

この 2 つの sub() は、gsub() を使って以下のように一つで書くこともできます。

```
gsub(/<\/?title>/,"",s)
```

\/? は \/ が一つあるか、またはない状態を意味しますので、これで <title> と </title> の両方にマッチすることになります。この gsub() は、マッチしたものすべてを ”” (空文字列) で置きかえますので、1 回の gsub() で両方が取り除かれることになります。

2. substr() を使う場合

   s の先頭が <title> で、最後が </title> であることがわかっていれば

```
n1=length("<title>")
n2=length("</title>")
s1=substr(s,n1+1,length(s)-n1-n2)
```

   で取り出すことができます。length(s) は文字列 s の長さを返す関数です。

なお、実際の文字列には <title> や </title> の前後にスペースなどの不要なものがついている可能性もあります。その場合は、substr() では match() や index() との併用が必要になるでしょう。例えば以下のような具合です。

```
if(match(s,/<title>[ \t]*/))  # [ \t] はスペースかタブ
  s=substr(s,RSTART+RLENGTH)  # マッチしたところ以降
if(match(s,/[ \t]*<\/title>/))
  s=substr(s,1,RSTART-1)      # マッチしたところの手前まで
```

同じことを sub() で行おうとすると以下のようになります。

```
sub(/.*<title>[ \t]*/,"",s)  # .* は任意の文字の 0 文字以上
sub(/[ \t]*<\/title>.*/,"",s)
```

sub(), gsub() を使う方が簡単そうですが、substr() + match() の場合はその部分文字列が確かに含まれているかどうかのチェックが行える、というメリットがあります。

今回はまずそれが含まれる文字列に対して処理をしますのでチェックは不要ですから、sub(), gsub() で行うことにします。

6 各部分の構成

6.1 タイトルの取得部分

これまでの内容をまとめて、全体を構成してみます。

行の取得は、4 節で述べた getline を使う方法を使い、行から必要な部分文字列を取得するには、5 節で述べた sub(), gsub() を使う方法で行うことにします。

まず、3 節の 1. の <title>～</title> の取得は以下のようにします。

```
##### タイトルの取得 #####
($0 ~ /<title>/){
    titlestr=$0
    while(titlestr !~ /<\/title>/){
        if(getline<=0){ errorexit=1; exit }
        titlestr = titlestr $0
    }
    sub(/.*<title>[ \t]*/,"",titlestr)
    sub(/[ \t]*<\/title>.*/,"",titlestr)
    next
}
```

コードの内容はすでに 4, 5 節で説明した通りです。

6.2 見出し部分の取得部分

次に、3 節の 2. の見出しの部分の取得は以下のようにします。

```
##### 見出し部分の取得 #####
($0 ~ / CONTENTS_TITLE_TABLE/){
    # 不要な行の読み飛ばし
    while($0 !~ /<font size/)
      if(getline<=0){ errorexit=2; exit }
    # headline に行を保存
    headline=$0
    while(headline !~ /<\/small>/){
        if(getline<=0){ errorexit=3; exit }
        headline = headline $0
```

```
    }
    # 不要なタグ (font,small) を削除
    gsub(/<\/?font[^>]*>/,"",headline)
    sub(/[ \t]*<small>[ \t]*/,"",headline)
    sub(/[ \t]*<\/small>.*/,"",headline)
    while($0 !~ /\/CONTENTS_TITLE_TABLE/)
      if(getline<=0){ errorexit=4; exit }
    next
}
```

これは、“CONTENTS_TITLE_TABLE” というキーワードがそこにしか使われていないのでできることなのですが、ここでは以下のようなことを行っています。

1. まず最初の while() 文で必要な部分に達するまでの行の読み飛ばし、つまり、“<font size” という文字列に出会うまでは getline を繰り返す
2. 出会ったらそれを headline に保存し、それに “</small>” が含まれていなければ、複数行で構成されている可能性を考えて、再び getline で headline を追加していく
3. headline から不要なタグ (font, small) を削除
4. “/CONTENTS_TITLE_TABLE” というキーワードに出会うまで行の読み飛ばしを行う

今回のタグの削除は、, タグは gsub() で、<small>, </small> タグは sub() で行っていますが、font の方はそのタグのみを削除し、small の方はその前後のゴミも削除するようにそうしています。なお、この文字列には , タグも含まれるのですが、これは出力用に残しておくことにします。

なお、<font> の削除で、「<\/?font[^>]*>」という正規表現を使っていますが、この後半部分は

- [^>] : > 以外の文字
- [^>]* : > 以外の 0 文字以上の連続
- [^>]*> : > 以外の 0 文字以上の連続の後に >

を意味しています。<font> は、実際には <font size=+1> のように使われていてオプションが含まれているので、それも一緒に削除するために

「<font とその後に続く > 以外の文字列とその後の >」

をこれで表現しています。

6.3 記事本文の取得部分

次は、3 節の 3. の部分の取得ですが、これは、上と同じように考えて “OUTLINE_TABLE” というキーワードをまずは目印にすればいいのですが、実はこのキーワードはこの HTML ファイル中で何回か使用されているので、単純に同じようにはできません。

記事本文は、そのうち最初の “OUTLINE_TABLE” の後に出てくるので、それが最初の物であるかを調べることにして以下のようにすればいいでしょう。

```
##### 記事本文の取得 #####
# N_ot = 何回目の OUTLINE_TABLE ブロックであるかを保持
($0 ~ / OUTLINE_TABLE/){ N_ot++ }
# 最初の OUTLINE_TABLE ブロックのときだけ以下を実行
(N_ot==1 && $0 ~ / OUTLINE_TABLE/){
    # 不要な部分を読み飛ばし
    while($0 !~ /<font size/)
      if(getline<=0){ errorexit=5; exit }
    # 本文の保存 (body[1] ~ body[N_body])
    N_body=0
    do{
        if($0 ~ /<\/?font/) gsub(/<\/?font[^>]*>/,"")
        body[++N_body]=$0
        if(getline<=0){ errorexit=6; exit }
    }while($0 !~ /<div/)
    # 本文の最後の <div> タグ部分を改行 (<br>) に変換して保存
    sub(/<div[^>]*>/,"<br>")
    body[++N_body]=$0
    # </div> までを保存
    while($0 !~ /<\/div/){
        if(getline<=0){ errorexit=7; exit }
        body[++N_body]=$0
    }
    # </div> タグは単に削除
    sub(/<\/div>/,"",body[N_body])
    next
}
```

このように N_ot という変数に何回目の OUTLINE_TABLE ブロックであるかを保存することで、最初の OUTLINE_TABLE ブロックだけに実行させることができるようになります。

本文の font タグは削除し、div タグは<div ...> は改行 (
) に変換し、 は

削除し、本文全体を body という配列に保存しています。

6.4 Copyright 以下の部分の取得部分

最後は、3 節の 4. の Copyright 以下の部分の取得ですが、これは以下のようにします。

```
##### Copyright 以下の部分の取得 #####
($0 ~ /\/YBB module/){
    while($0 !~ /Copyright/)
      if(getline<=0){ errorexit=8; exit }
    N_tail=0
    do{
        gsub(/<\/?small>/,"")
        gsub(/<\/?center>/,"")
        tail[++N_tail]=$0
    }while($0 !~ /<\/html>/ && getline>0)
    exit
}
```

ここでは、“/YBB module” というキーワード以後の、“Copyright” 以下の文を </html> まで tail という配列に取得しています。その際、small タグと center タグは削除しています。

6.5 サブルーチンと END ブロック部分

そしていよいよ、これら保存した値を用いて最終出力を作成するわけですが、これは関数として作ってみましょう。

```
##### 出力関数 #####
# ヘッダ部分の出力関数
function putheader(titlestr,headline)
{
    print  "<html>"
    print  "<head>"
    printf "<title>%s</title>\n",titlestr
    print  "</head>"
    print  "<body>"
    printf "<h1>%s</h1>\n",titlestr
```

```
    printf "%s<hr>\n",headline
}
# 本文の出力関数
function putbody(body,N,  j)
{
    for(j=1;j<=N;j++) print body[j]
}
# 最後の部分の出力関数
function puttail(tail,N,  j)
{
    print "<hr>"
    for(j=1;j<=N;j++) print tail[j]
}
```

このように関数化しておくことで、後で拡張やカスタマイズなどがしやすいようにしておきます。なお、これらの関数は以下のようなことをしています。

- putheader():
  タイトル文字列 (titlestr) をタイトル (<title>～</title>) と先頭の見出し (<h1>～</h1>) に用い、その次に見出し文字列 (headline) を出力して水平線 (<hr>) を描画
- putbody():
  保存されている記事本文をそのまま出力
- puttail():
  水平線 (<hr>) を描画して、保存されている Copyright 以下の部分をそのまま出力

これらの関数を使えば、END ブロックは以下のようにすればいいことになります。

```
##### END ブロック #####
END{
    if(errorexit){
      printf "エラー発生 (code = %d)\n",errorexit > "/dev/stderr"
      exit
    }
    putheader(titlestr,headline)
    putcontents(body,N_body)
    puttail(tail,N_tail)
}
```

7 全体のソースコード

6 節のソースコードを全部つなげると以下のようになります。

```
##### タイトルの取得 #####
($0 ~ /<title>/){
    titlestr=$0
    while(titlestr !~ /<\/title>/){
        if(getline<=0){ errorexit=1; exit }
        titlestr = titlestr $0
    }
    sub(/.*<title>[ \t]*/,"",titlestr)
    sub(/[ \t]*<\/title>.*/,"",titlestr)
    next
}
##### 見出し部分の取得 #####
($0 ~ / CONTENTS_TITLE_TABLE/){
    # 不要な行の読み飛ばし
    while($0 !~ /<font size/)
      if(getline<=0){ errorexit=2; exit }
    # headline に行を保存
    headline=$0
    while(headline !~ /<\/small>/){
        if(getline<=0){ errorexit=3; exit }
        headline = headline $0
    }
    # 不要なタグ (font,small) を削除
    gsub(/<\/?font[^>]*>/,"",headline)
    sub(/[ \t]*<small>[ \t]*/,"",headline)
    sub(/[ \t]*<\/small>.*/,"",headline)
    while($0 !~ /\/CONTENTS_TITLE_TABLE/)
      if(getline<=0){ errorexit=4; exit }
    next
}
##### 記事本文の取得 #####
# N_ot = 何回目の OUTLINE_TABLE ブロックであるか
($0 ~ / OUTLINE_TABLE/){ N_ot++ }
# 最初の OUTLINE_TABLE ブロックのときだけ実行
(N_ot==1 && $0 ~ / OUTLINE_TABLE/){
    # 不要な部分を読み飛ばし
    while($0 !~ /<font size/)
```

```
      if(getline<=0){ errorexit=5; exit }
    # 本文の保存 (body[1] ~ body[N_body])
    N_body=0
    do{
        if($0 ~ /<\/?font/) gsub(/<\/?font[^>]*>/,"")
        body[++N_body]=$0
        if(getline<=0){ errorexit=6; exit }
    }while($0 !~ /<div/)
    # 本文の最後の <div> タグ部分を改行 (<br>) に変換して保存
    sub(/<div[^>]*>/,"<br>")
    body[++N_body]=$0
    # </div> までを保存
    while($0 !~ /<\/div/){
        if(getline<=0){ errorexit=7; exit }
        body[++N_body]=$0
    }
    # </div> タグは単に削除
    sub(/<\/div>/,"",body[N_body])
    next
}
##### Copyright 以下の部分の取得 #####
($0 ~ /\/YBB module/){
    while($0 !~ /Copyright/)
      if(getline<=0){ errorexit=8; exit }
    N_tail=0
    do{
        gsub(/<\/?small>/,"")
        gsub(/<\/?center>/,"")
        tail[++N_tail]=$0
    }while($0 !~ /<\/html>/ && getline>0)
    exit
}
##### END ブロック #####
END{
    if(errorexit){
      printf "エラー発生 (code = %d)\n",errorexit > "/dev/stderr"
      exit
    }
    putheader(titlestr,headline)
    putbody(body,N_body)
    puttail(tail,N_tail)
}
```

```
##### 出力関数 #####
# ヘッダ部分の出力関数
function putheader(titlestr,headline)
{
    print  "<html>"
    print  "<head>"
    printf "<title>%s</title>\n",titlestr
    print  "</head>"
    print  "<body>"
    printf "<h1>%s</h1>\n",titlestr
    printf "%s<hr>\n",headline
}
# 本文の出力関数
function putbody(body,N,  j)
{
    for(j=1;j<=N;j++) print body[j]
}
# 最後の部分の出力関数
function puttail(tail,N,  j)
{
    print  "<hr>"
    for(j=1;j<=N;j++) print tail[j]
}
```

8 最後に

今回は、HTML の加工という、より AWK らしい処理を紹介しましたが、HTML の簡単な説明や対象となる HTML ファイルの構造の説明などが入ったため少し長くなり、逆に AWK のコード部分の説明がやや短かくなりました。それに関しては以前の AWK の説明 [1], [2], [3] や、[4], [5], [6] などの書籍を参照してください。

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] 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)

[5] 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)

[6] 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)

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

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

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