インターフェイスの街角– Web で WEB

増井 俊之

Web ページ上では、文章や写真、イラスト、動画、音声、MIDI など、さまざまなマルチメディア・データが公開されています。しかし、プログラムのソースコードは、相変わらず*.tar.gz や*.lzh などの〝地味な〟形式でまとめて置かれていることが多く、プログラムリストがそのまま掲載されている Web ページはほとんどありません。文章や絵、写真などは人に見せたくても、ソースコードについてはそう思わない人が多いのかもしれません。

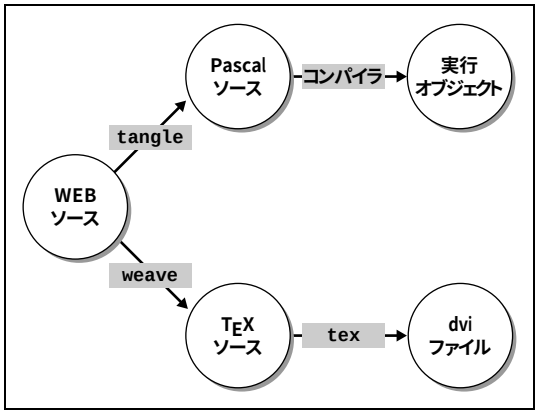
しかし、本来はプログラムも一般の文章と同様に扱われるべきであり、それ自体が読むに堪えるものだと思います。そこで、今回は Web 上でのプログラムの公開を前提とした「WEB on Web システム」を紹介します。

文芸的プログラミングと WEB システム

スタンフォード大学の Donald E. Knuth 教授は、文章のようにプログラムを書くことによってプログラミング上のいくつかの問題点を解決する〝文芸的プログラミング (Literate Programming)〟[1]という手法を提唱しています。

文芸的プログラミングとは、プログラムの作成と文書化を同時におこなう手法です。プログラムとその解説を 1 つのファイルに混在させながら追加や修正をおこない、解説文書と完全に整合性のとれたプログラムを開発するというものです。一般的な手法では、まずプログラムを開発してから解説文書を作成します（その逆の場合もあります）。しかし、あとで追加や修正を加えると、両者の整合性がとれなくなる場合があります。これに対し、文芸的プログラミングの手法では両者が 1 つのファイルに統合されているため、そのようなことが起こりにくくなります。また、プログラム開発と文書作成の相乗作用によって、より適切なプログラムと文書ができるという効果もあります。

図 1　WEB の処理



Knuth 先生は文書整形システム TEX の開発者として有名ですが、その開発の際に文芸的プログラミングの実例として **WEB** というシステムを作成しました[1]。WEB は Pascal と TEX を対象としていますが、C 言語を扱えるようにした CWEB システムなども開発されています。

WEB の処理は、図 1 のようにおこなわれます。

WEB 文書は複数のプログラム・モジュールの集合で、各モジュールはその解説部とプログラム部から構成されます。WEB 文書を tangle というプログラムにとおすとプログラム部だけが抽出されて 1 つの Pascal プログラムになり、weave というプログラムで処理すると解説部とプログラム部の両方がまとめられた TEX 文書になります。プログラム部はきれいに整形され、モジュールごとに章が立

1　〝Web〟というと、いまでは誰もが World Wide Web のことだと思うかもしれませんが、ひと昔前に計算機業界で WEB といえば Knuth 先生のシステムを指すのが普通でした。

図 2　weave により得られた文書出力

§1　　CWEB のサンプルテキスト　1

1. CWEB のサンプルテキスト。これは CWEB の使い方を簡単に説明するためのテキストです。

2. この関数を呼ぶとメッセージを表示した後終了します。

```
fatal(s)
char *s;
{
  fprintf(stderr,"%s\n",s); exit(1);
}
```

3. ⟨ 初期化処理 3 ⟩ ≡

```
initialized ← NO; maxlin ← 1000;
```

This code is used in section 4.

4. これがメインプログラムです。

```
int initialized;
int maxlin; main() { ⟨ 初期化処理 3 ⟩... }
```

exit: 2.
fatal: 2.
fprintf: 2.
initialized: 3, 4.
main: 4.
maxlin: 3, 4.
NO: 3.
s: 2.
stderr: 2.

てられて、目次や索引も備えた完全なドキュメントが生成されます。このように、1 つの WEB ファイルから、プログラムとその解説文書の両方が生成できるのです。

Pascal の代わりに C を用いる CWEB の文書の例を以下に示します。

```
@* CWEBのサンプルテキスト
これはCWEBの使い方を簡単に説明するためのテキストです。
@
この関数を呼ぶとメッセージを表示した後終了します。
@c
fatal(s) char *s;
{
      fprintf(stderr,"%s\n",s);
      exit(1);
}
@ @<初期化処理@>=
initialized = NO;
maxlin = 1000;
@
これがメインプログラムです。
@c
int initialized;
int maxlin;
main()
{
      @<初期化...@>
      ……
}
```

`@␣´（␣は空白）からモジュールの解説部が、`@c´からプログラム部が始まります。`@<モジュール名@>=´はモジュールのマクロ定義で、`@<モジュール名@>´で呼び出せます。モジュール名が長い場合は、呼び出すときに `...´として後ろを省略することができます。これらのほかにも、整形処理のためのコマンドがいくつか用意されています。

プログラムの開発は、

WEB 文書の編集 → tangle によるプログラム抽出 → コンパイル → テスト

の繰返しによっておこなわれます。

前掲の WEB 文書を weave で処理すると、図 2 の出力が得られます。モジュールごとに節をあらため、キーワードを太字にするといった整形がおこなわれ、さらに相互参照や索引も作成されています。

WEB の問題点

WEB はプログラミングに効果的なシステムとして有名ですが、そのわりには使っているという人をあまりみかけません。これは、次のような短所もあるためではないかと思います[2]。

- ソースが読みにくくなる。

- tangle、コンパイル、実行テスト、フィードバックという繰返しが煩わしい。
- weave、TEX、dviware（xdvi や dvi2ps など）といった手順を踏まなければ出力できない。
- 解説部やプログラム部に整形のための指令を入れなければならない。
- マクロ定義機能が分かりにくい。
- プログラムを分割した場合、それがすべてモジュールとみなされて新たな段落になってしまう。
- 1 つのプログラミング言語にしか使えない。
- 出力には TEX しか使えない。
- 分割コンパイルができない。

WEB on Web

そこで、文芸的プログラミングの考え方を活かしつつ、WEB システムの短所をできるかぎり取り除いた **WEB on Web システム**を作ってみました。これは、特殊な WEB 形式の文書ファイルを使うのではなく、プログラムとその説明を HTML で記述するものです。プログラムを HTML ファイルのなかに記述すれば weave 関連プログラムは不要になりますし、編集したファイルをすぐにブラウザで眺めることもできます。

このシステムでは、以下のような規則に従って HTML ファイルを記述します。

- HTML ファイルのなかにプログラムとその説明を両方記述する。
- プログラムは、<pre>と</pre>で囲まれた部分に記述する。
- プログラム・ファイルの名前は、<pre>タグのなかで〝<pre file=foo.c>〟のように指定する。複数のファイルをカンマで区切って列挙してもよい。

プログラムの部分を切り出すプログラムは必要ですが、<pre>と</pre>のあいだを対象とすればいいので、ごく簡単なものですみます。

WEB on Web の例

HTML ファイルからプログラム部を切り出すための、Perl プログラム **wtangle**（WEB システムの tangle に相当します）を WEB on Web で記述した例を示します。

HTML ソース

WEB on Web を用いて wtangle について説明した文書 wtangle.html を末尾のリスト 1 に示します。<pre file=...>と</pre>に囲まれた部分がプログラムです。この HTML ファイルを wtangle で処理すると、wtangle プログラムが抽出されます。

ブラウザでの表示

リスト 1 の HTML ファイルは、Web ブラウザでは図 3〜5 のように表示されます。 や<table>のなかなど、任意の場所にプログラムを書けるので、枠を付けたり表にしたり背景に着色したりと、さまざまな表示方法が選択できます。

おわりに

率直にいって、プログラムを WEB システムの形式で書いたり、あるいは HTML で記述するのはかなり面倒です。その意味では、普通のプログラム開発に文芸的プログラミングの手法を適用するのはちょっと無理があるかもしれません。しかし、オープンソースとしてプログラムを公開する場合や、プログラミングの解説をしたり解説書を執筆するようなときは役に立つのではないでしょうか。きちんとした図や解説文書を作るのは手間のかかる作業ですが、プログラムに手描きの GIF 画像を添付するだけでも、あとでプログラムを読み返すときに役立ちますし、説明や関連情報などへのリンクも手軽に付けることができます。

他人に見せる機会のあるプログラムや、将来再利用する可能性があったり、複雑なアルゴリズムを用いたプログラムなどについては WEB on Web は有効なシステムではないかと思います。

（ますい・としゆき　ソニー CSL）

[参考文献]

[1] Donald E. Knuth, *Literate Programming*, Computer Journal, May 1984
邦訳：黒川利明訳「文芸的プログラミング」、bit 1985 年 4 月号（『文芸的プログラミング』（アスキー、1994 年）にも所収）

[2] 増井俊之『Perl 書法』、アスキー、1993 年 7 月

リスト 1　wtangle.html

```
<html>
<head>
<title>Web上での「文芸的プログラミング」</title>
<style type="text/css">
div.gray {
   background: #c0c0c0;
   padding: 0 0 0 0;
   border-width: 0;
   border: none;
   margin: 0 0 0 0;
   font-size: small;
}
div.border {
   background: white;
   padding: 0 0 0 0;
   border-width: 0;
   border: thin solid black;
   margin: 0 0 0 0;
   font-size: small;
}
h3 {
   color: black;
   background: white;
   padding: 7 7 7 7;
   border-width: 0;
   border: thick solid blue;
   margin: 0 0 0 0;
   font-size: small;
}
</style>
</head>

<body bgcolor=white>

<h1>Web上での「文芸的プログラミング」</h1>

<div align=right>
<a href="http://www.csl.sony.co.jp/~masui/">増井俊之</a><br>
<a href="http://www.csl.sony.co.jp">ソニーコンピュータサイエンス研究所</a>
</div>
<hr>

<h2>文芸的プログラミングとは</h2>

「<b>文芸的プログラミング</b>」とは、
スタンフォード大学の<b>Donald Knuth</b>が提唱している、
プログラムの作成とその文書化を同時におこなう手法で、
プログラムとその解説文書をひとつのファイルに混在させながら
追加したり修正したりして同時に開発していくことにより、
完全に整合性のとれた文書とプログラムを開発しようというものである。

プログラムをまず作ってからその解説文書を作成する(またはその逆)という
一般的な手法では、後で修正などを加える場合などは注意しないと
両者の整合性がとれなくなってしまうことがよくあるが、
文芸的プログラミングの手法では
両者がひとつのファイルになっているので
そのようなことが起こりにくくなる。
また正しい文書を書くという作業と正しいプログラムを書くという作業の
```

相互作用によってより適切なプログラムと文書ができあがるという
効果がある。
<p>

WEBシステムは
文芸的プログラミングを支援するために
Knuthが開発したシステムである。
WEBシステムはもともと
KnuthのTeXシステムを記述するために
開発されたものなのでPascalとTeXが対象になっているが、
Cが使えるようにしたCWEBシステム
Cとtroffとの組合せが使えるcwebシステム、
任意のプログラミング言語に対応させることのできる
Spiderといったシステムも開発されている。

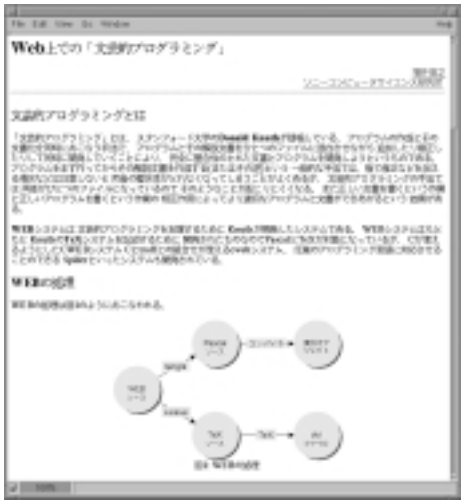
<h2>WEBの処理</h2>

WEBの処理は図1のようにおこなわれる。
<p>

<img src=figures/WEB.gif>
図1: WEBの処理

<p>

図 3　Web ブラウザでの表示（先頭部分）



WEB文書は複数のプログラムモジュールの集合という形になっており、
各モジュールはその解説部分とプログラム部分とで構成される。
WEB文書を<code>tangle</code>というプログラムに通すと
プログラム部だけが抽出されてひとつのPascalプログラムになり、
<code>weave</code>というプログラムに通すと解説部とプログラム部の両方が
融合したTeXの文書になる。
ここではプログラム部は美しくフォーマットされ、モジュールごとに章立てがおこなわれ、
目次や索引も完備した完全なドキュメントが生成される。
このように、ひとつのWEBファイルからプログラムと
そのドキュメントの両方を生成することができるのが特長である。

<hr>
<h2>WEB on Web</h2>

整形されたWEB文書を見るためには、図1のように
<code>weave</code>によってTeX文書に変換してから
TeXでフォーマットをおこない、<code>dvi2ps</code>などで印刷形式にする必要がある。
この場合、
<code>weave</code>/TeX/<code>dvi2ps</code>/<code>ghostscript</code>
を順に適用する必要ががあり非常にわずらわしいので、
作業を頻繁におこないにくいという問題がある。

<p>
WEB文書のかわりに普通のHTMLファイルを使うようにすれば
このような問題は解決すると考えられる。
プログラムをHTMLファイル中に記述するという形式で
文芸的プログラミングをおこなうことにすれば、
<code>tangle</code>に相当するプログラムは必要であるが、
<code>weave</code>関連プログラムはすべて省略して
直接HTMLファイルをブラウザで眺めることができる。

HTMLファイルに対して
<code>tangle</code>に相当する処理をおこなう
<code>wtangle</code>プログラムを開発した。

<h2>wtangleプログラム</h2>

<code>wtangle</code>プログラムは、
HTMLファイルとして記述された、
簡易文芸的プログラミング文書から
プログラム部分だけを抽出するPerlプログラムである。
本HTML文書に対して<code>wtangle</code>を適用すると
<code>wtangle</code>プログラムが得られるので、
コンパイラのようにブート ストラップ式に
開発していくことができるはずである。

以下に<code>wtangle</code>プログラムの全ソースと
その解説を記述する。<p>

<h3>プログラム</h3>

<code>prologue()</code>で
プログラムの初期化をおこない、
<code>main()</code>で
HTMLファイルからのプログラム抽出処理をおこない、
<code>epilogue()</code>で
ファイル書出しをおこなう。

図 4　Web ブラウザでの表示 (プログラム部)

```
<p>
<div style="border: thin solid black; margin: 0 0 0 0;">
<pre file=wtangle>
#!/usr/local/bin/perl
#
# wtangle - Web Tangle
#
&prologue;
&main;
&epilogue;
</pre>
</div>

<a name=initialize>
<h3>初期化処理</h3>

<a href="#prologue"><code>prologue()</code></a>では変数の初期化などをおこなう。
<code>%SIG</code>に関数名を指定しておくと、
signalが発生したときに<code>$SIG{シグナル名}</code>の関数が呼ばれる。
ここではSIGINTなどを受け取ったときに
<a href="#finish"><code>finish()</code></a>が呼ばれるようにしている。

<div class=gray>
<a name="prologue"></a>
<pre file=wtangle>
sub prologue {
  require 'cacheout.pl'; # <a name=cacheout>複数ファイル出力ライブラリ</a>
  if($#ARGV < 0){
    print STDERR "wtangle --- Web Tangle\n";
    print STDERR "Usage: wtangle documentfiles\n";
    exit 0;
  }
  $SIG{'INT'} = $SIG{'TERM'} =
  $SIG{'QUIT'} = $SIG{'HUP'} = 'finish';
  $lt = '<';
}
</pre>
</div>
```

```
<h3>メインルーチン</h3>

HTMLファイル中の
<code>&lt;pre&gt;</code>,
<code>&lt;/pre&gt;</code>
で囲まれた部分をファイルに書き出す。
ファイル名は
<code>&lt;pre&gt;</code>のなかで
"<code>file=</code>ファイル名{,ファイル名...}"と記述する。
HTMLファイルが更新された場合でも
プログラム自体は変更されていない場合があるので、
直接プログラムを書き出さずに
<a href="#tmpname"><code>tmpname()</code></a>で計算される
テンポラリファイルに一旦書き出した後で
<a href="#diff"><code>diff()</code></a>でもとのファイルと比較をおこない、
異なっている場合のみファイルを更新するようにする。

<a name="main">
<div class=gray>
<pre file=wtangle>
sub main {
  while($file = shift(@ARGV)){
    unless(open(f,$file)){
      print STDERR "Can't open <$file>\n";
      &finish;
    }
    while(<f>){
      if(m#^${lt}/pre>$#){
        @tmpfiles = ();
      }
      elsif(m#^${lt}pre\s*file=(.*)>$#){
        @files = split(/,/,$1);
        @tmpfiles = ();
        foreach $file (@files){
          $allfiles{$file} = 1;
          push(@tmpfiles,&tmpname($file));
        }
      }
      else {  # cachout()を使って現在行を各ファイルに書き出す
        foreach $tmpfile (@tmpfiles){
          &cacheout($tmpfile);
          print $tmpfile $_;
        }
      }
    }
    close(f);
  }
}
</pre>
</div>

<h3>終了処理</h3>

実際に更新があったプログラムのみ書き出して実行可能ファイルとする。

<a name="epilogue"></a>
<div class=gray>
<pre file=wtangle>
sub epilogue {
  foreach $file (keys %allfiles){
    $tmpfile = &tmpname($file);
```

```
    close($tmpfile);
    if(! -e $file || &diff($tmpfile,$file)){
      push(@newfiles,$file);
      &move($tmpfile,$file);
      chmod 0555,$file; # 書込み禁止/実行可能にする
    }
  }
  sort @newfiles;
  $n = $#newfiles + 1;
  print STDERR
    $n == 0 ? "No file is updated.\n" :
    $n == 1 ? "File '$newfiles[0]' is updated.\n" :
      "Files '".join("', '",@newfiles[0..$n-2]).
      "' and '".$newfiles[$n-1]."' are updated.\n";
  &finish;
}
</pre>
</div>


<h3>サブルーチン</h3>

使用している小さなサブルーチンのリストを示す。
ここでは&lt;<code>table</code>&gt;タグを用いて表形式のリストにしている。
HTMLを用いているのでこのように表示形式を柔軟に選ぶことができる。

<p>
<table border>
<tr>
<td bgcolor=#dddddd>
<a name="tmpname">
<pre file=wtangle>
sub tmpname {
  local($_) = @_;
  s#/#_#g;
  "/tmp/stangle$_$$";
}
</pre>
</td>
<td>
テンポラリに使用されるファイル名を計算する。
</td>
</tr>

<tr>
<td bgcolor=#dddddd>
<a name="finish">
<pre file=wtangle>
sub finish {
  foreach $file (keys %allfiles){
    $tmpfile = &tmpname($file);
    unlink($tmpfile) if -e $tmpfile;
  }
  exit;
}
</pre>
</td>
<td>
テンポラリに使用されたファイルをすべて削除してから
プログラムを終了する。
</td>
```

図5　Webブラウザでの表示（プログラム解説）

```
</tr>

<tr>
<td bgcolor=#dddddd>
<a name="move">
<pre file=wtangle>
sub move {
  local($from,$to) = @_;
  return if $from eq $to;
  system "\\mv -f $from $to";
  if($?){  # mv実行エラー
    print STDERR "Can't move $from to $to\n";
    &finish;
  }
}
</pre>
</td>
<td>
ファイル移動をおこなう。
簡単のため<code>mv</code>コマンドを使用している。
</td>
</tr>

<tr>
<td bgcolor=#dddddd>
<a name="diff">
<pre file=wtangle>
sub diff {
  local($file1,$file2) = @_;
  system "diff $file1 $file2 > /dev/null";
  $? >> 8; # コマンドステータス
}
</pre>
</td>
<td>
ファイル比較をおこなう。
簡単のため<code>diff</code>コマンドを使用している。
</td>
</tr>
</table>

<hr>
<h2>感想など</h2>

実際のところ、
プログラムを書くたびにWEB文書やHTMLを真面目に記述するのはとても面倒なので、
普通のプログラム開発に文芸的プログラミングの手法を適用するのはちょっと無理があるように思う。
しかしオープンソースとしてプログラムを公開したい場合や、
プログラミングの解説をしたり、解説書を執筆するような場合には
この方式は便利なのではないかと思っている。

真面目に図や解説を書くのは面倒かもしれないが、
プログラムに手描きのGIF画像を添付するだけでも後で思い出したりするのに便利だろうし、
説明や関連情報へのリンクを添付するには便利であろう。

人に見せる機会があるプログラムや将来再利用する可能性があるプログラム、
複雑なアルゴリズムを用いたプログラムなどはなるべくこの手法で整備してみようと思っている。

</body>
</html>
```