サーバー構築/運用術

第3回 Perl(前編)

WWW(World Wide Web)が普及して以来,CGI(Common Gateway Interface)用の言語と見なされてしまうこともあるperlですが,CGIだけでなくあらゆる用途に活用できる万能スクリプト言語であり,システム管理に無くてはならないツールと言えます。perlを使いこなせることは,システム管理者の条件と言っても過言ではないでしょう。

(ケイ・ラボラトリー　仙石 浩明)

私が初めてコンピュータと出会ったのは今から20年以上前のことです。最初のパソコンは,コモドールPET 2001(**写真1**)*1でした。8KバイトのRAMと,BASICインタプリタを収めたROMが搭載されていて,当然OSのようなものは無く,プログラムの保存にはカセット・テープを使っていました。

現在のPCと比べると信じられないくらい遅く*2,またメモリーも少ないため,用途*3と言えばゲームが大半で,しかも当時のアーケイド・ゲームに比べても貧弱で,一体何がそんなに魅力で夢中になった*4のかと思います。

なぜこんな昔話*5をするかといいますと,現在の携帯電話は当時のパソコンによく似ているからです。Javaが搭載されて無限の可能性を期待されつつ,実行速度が遅く,プログラム・サイズも小さいため,用途はゲームが大半です。しかも,そうしたゲームは,PCやゲーム専用機のゲームに比べれば,かなり見劣りします。

プログラマブル携帯電話の将来性を感じつつも,現在のところは実用にならない,というのが多くの人の感想ではないでしょうか。ちょうど20年前のパソコンのように。今年はPC-DOSとMS-DOSの登場からちょうど20年目の年に当たります。DOSというパソコンにおける統一プラットフォームの登場がその後のパソコン業界の勢力図を決定したように,携帯電話コンテンツ業界の将来は,プラットフォームを制した者が握ることになるでしょう。

それは何年も先の話ではなく,間近に迫っていて,しかも今のところ日本が一歩先行しています。チャンスがつかめる*6としたら,それは今をおいてほかにあるでしょうか?

写真1　コモドール・ジャパンが1978年2月に発売した「コモドール PET2001」
電脳博物館(http://www.tk.airnet.ne.jp/mit/midland/Musium/musium.html)の許可を得て掲載。

＊1　1978年発売。定価が29万8000円(当時の物価を考えるとかなり高額ですね)と,当時中学生だった私に買えるわけはなく,学校のマイコン部にあった3台のPET2001を20人以上の部員で使っていたのでした。
＊2　CPUはMOS6502(動作周波数は1.0MHz)でした。初めてハンド・アセンブルして作ったプログラム(詳細は忘れましたが,画面全体を反転させるプログラムだったと思います)を実行したとき,BASICに比べて何て速いんだ,と感動したのを覚えています。
＊3　当時の広告いわく「本格的なコンピュータの機能をあわせ持つパーソナルコンピュータです。役に立つ,使いものになるということが充分に考えられています。家庭・個人用から産業用までの広い多様な応用が可能です。」
＊4　結局,コンピュータと縁が切れることはその後一度もなく,現在もプログラマをやっていて,今後もかかわり続けることになるのでしょう。あのときパソコンと出会わなかったら,全く違う人生になっていたのかも知れません。
＊5　昔話をするのは年寄りの証拠,なのであまりしたくはないのですが…。
＊6　我々と一緒に挑戦しようという方,連絡下さい。

```
asao:/usr/lib/perl5 $ perl -MCPAN -e shell;↵
cpan shell -- CPAN exploration and modules installation (v1.59)
ReadLine support enabled

cpan> install Mail::POP3Client↵
    ...
Running install for module Mail::POP3Client
Running make for S/SD/SDOWD/POP3Client-2.7.tar.gz
    ...
Running make test
    ...
All tests successful.
    ...
Running make install
    ...
cpan>                                          ↵このマークで改行
```

図1　CPANモジュール

perl

perlは，その名前「実用データ取得レポート作成言語（Practical Extraction and Report Language）」が示す通り，大量のテキストを効率よく処理するための言語として作られました。従来，テキスト処理用のツールとしては，sed（Stream EDitor）＊7やawkがあったのですが，どちらも複数ファイルの入出力という点で難がありました。

「UNIX的」考え方＊8からすると，足りない機能を補う別のツールを用意してsedやawkと組み合わせるのが定石なのですが，perlの作者であるLarry Wall氏は，1つのツールにどんどん機能を追加する道を選びました。

perlの初期のバージョンでは，sedとawkの機能に若干の機能追加をしただけのツールだったのですが，sedやawkより高速だったので次第に多くの用途に使われるようになりました。用途が増えれば必要とされる機能も増え，perlの進化が始まりました。

まず，テキスト処理だけではなく，テキストを格納しているファイルそのものを操作できるようになり，続いてプロセス操作やプロセス間通信がサポートされました。さらにソケット経由で他のホスト上のプロセスとも通信できるようになり，ネットワーク・プログラミングにも使えるようになりました。

その結果，数多くのツールがperlで書かれるようになりました。今やperlは，UNIX互換システムにとって欠くことのできないツールの一つと言えるでしょう。つまりperlはUNIX的規範を大きく逸脱することによって，UNIXの世界を支えるプラットフォームの一つとなったのです＊9。

perl5でモジュールがサポートされたことにより，perlのプラットフォームとしての性格がさらに強まりました。C言語で書かれたモジュールを追加することにより，perl本体に手を加えずにさまざまな機能を追加することが可能になったのです。例えばデータベース・サーバーへのアクセスも，ラッパー・モジュールを追加すれば可能です。

CPAN

CPANとは，Comprehensive Perl Archive Networkの略で，perl関連ソフトウエアと関連文書の膨大な集積です。世界各国にCPANのサーバーが設置されている＊10ので，手近のサーバーにアクセスすると良いでしょう。

CPANには，perl5用の各種モジュールが大量に蓄積されています。目的に応じて適切なモジュールを利用すれば，perlを使ったプログラミングが大幅に省力化できます。CPANサイトからモジュ

＊7　perlを理解するには，まずその祖先とも言えるsedを理解するのが早道かも知れません。sedの理解には，拙著「SED 教室」（http://www.gcd.org/sengoku/sedlec/）をどうぞ。
＊8　2001年2月に出版された書籍「UNIXという考え方 その設計思想と哲学」（オーム社，ISBN：4-274-06406-9）が大いに参考になるでしょう。
＊9　似たような進化の過程を経てプラットフォームになったツールに，Emacsがあります。Emacsもperlと同様，UNIX的基準からすると巨大なシステムですが，数多くのツール群を擁し，UNIXには無くてはならないプラットフォームの一つと言えるでしょう。
＊10　CPANのミラー・サイトの一覧は，http://www.cpan.org/SITES.htmlにあります。

ールをダウンロードしてインストールするには，CPANモジュールを利用すると便利です。

図1に示すようにCPANモジュールを実行し，「install モジュール名↵」と入力すれば，CPANサイトからダウンロード*11してmakeし，テストを行った上でインストールが行われます。CPANモジュールのコマンド一覧は，**図2**のように「help↵」と入力すれば得られます*12。

図1の例でインストールしたMail::POP3Clientモジュールを利用すると，POPサーバーへアクセスするperlスクリプトを簡潔に書くことができます。例えば，新着メールのFrom:とSubject:の一覧を表示するスクリプトは，**図3**のように書けます。

インストール済モジュールのリストは，**図4**のように「cpan>」プロンプトにおいてautobundleコマンドを入力することにより，~/.cpan/Bundle/ディレクトリに保存されます。

このモジュール・リストのファイル（図4の場合は，Snapshot_2001_04_09_00.pm）を，例えばBundle/my_bundleという名前で@INCに含まれるディレクトリにインストールしておく（例えば，/usr/lib/perl5/site_perl/Bundle/とい

```
cpan> help↵

Display Information
 a                                  authors
 b         string           display  bundles
 d         or               info     distributions
 m         /regex/          about    modules
 i         or                        anything of above
 r         none             reinstall recommendations
 u                          uninstalled distributions

Download, Test, Make, Install...
 get                        download
 make                       make (implies get)
 test      modules,         make test (implies make)
 install   dists, bundles   make install (implies test)
 clean                      make clean
 look                       open subshell in these dists' directories
 readme                     display these dists' README files

Other
 h,?           display this menu       ! perl-code   eval a perl command
 o conf [opt]  set and query options   q             quit the cpan shell
 reload cpan   load CPAN.pm again      reload index  load newer indices
 autobundle    Snapshot                force cmd     unconditionally do cmd
cpan>
```

図2　CPANモジュールのコマンド一覧

```
#!/usr/bin/perl
use Mail::POP3Client;
$pop = new Mail::POP3Client (USER     => "sengoku",
                             PASSWORD => "*********",
                             HOST     => "asao.gcd.org");
for ($i=1; $i <= $pop->Count(); $i++) {
    foreach ($pop->Head($i)) {
        /^(From|Subject):\s+/i && print $_, "\n";
    }
}
$pop->Close();
```

図3　Mail::POP3Clientモジュールを利用したperlスクリプト

＊11　その他のモジュールに依存しているモジュールをインストールしようとした場合，自動的に必要なモジュールをダウンロードかつインストールした上で，インストールが行われます。また，モジュールのインストールに必要であれば，perlのバージョンアップさえ自動的に行われます。
＊12　CPANモジュールの詳しい説明は，「perldoc CPAN」を実行することによって表示されるマニュアルを参照してください。

```
cpan> autobundle
Going to read /usr/lib/perl5/.cpan/sources/authors/01mailrc.txt.gz
CPAN: Compress::Zlib loaded ok
Going to read /usr/lib/perl5/.cpan/sources/modules/02packages.details.txt.gz
Going to read /usr/lib/perl5/.cpan/sources/modules/03modlist.data.gz

Package namespace         installed    latest  in CPAN file
AnyDBM_File                   undef     undef  G/GS/GSAR/perl-5.6.0.tar.gz
Apache::AuthDBI                0.87      0.88  M/ME/MERGL/ApacheDBI-0.88.tar.gz
Apache::Cookie                 0.01      0.01  D/DO/DOUGM/libapreq-0.31.tar.gz
        ...
utf8                          undef     undef  G/GS/GSAR/perl-5.6.0.tar.gz
vars                          undef     undef  G/GS/GSAR/perl-5.6.0.tar.gz
warnings                      undef     undef  G/GS/GSAR/perl-5.6.0.tar.gz
warnings::register            undef     undef  G/GS/GSAR/perl-5.6.0.tar.gz

Wrote bundle file
    /usr/lib/perl5/.cpan/Bundle/Snapshot_2001_04_09_00.pm


cpan>
```

図4　インストール済みモジュール・リストの作成

```
cpan> install Bundle::my_bundle
```

図5　モジュールを一括してインストール

```
asao:/home/sengoku % perl -p -i.bak \
-e 's@/theme\.ja\.html\"@/theme.ja.html#NIKKEI\"@' \
public_html/docs/NikkeiLinux99-11/*.html
```

図6　perlを使った一括置換

うディレクトリを作成し，その中にmy_bundleというファイル名でコピーする)と，「cpan>」プロンプトで**図5**のように入力することにより，リストに含まれるモジュールを一括してインストールできます。perlをバージョンアップしたときや，他のマシン上のperlに同じモジュール群をインストールしたいときなど便利でしょう。

コマンド・ライン

perlスクリプトは，コマンド・ラインから指定することも可能なので，ちょっとした作業を行う際にも役に立ちます。例えば，HTMLで記述されたテキスト・ファイルにおいて，

```
<a href="../../theme.ja.html">
```

というリンクを

```
<a href="../../theme.ja.html
#NIKKEI">
```

へ変更したい場合を考えます。

変更すべきファイルが1つであれば，テキスト・エディタで置換すれば済むのですが，ファイルがたくさんある場合は面倒です。

perlならば，コマンド・ラインから**図6**のように実行すれば済みます。コマンド・ラインを見やすくするために「\」を入れて複数行に分割していますが，コマンド・ラインを1行で書くなら「\」は不要です。

図6の3行目が，置換すべき入力ファイルの指定，2行目が置換のためのスクリプトです。

「-p」オプションが指定されている(1行目)ため，2行目のスクリプトの前後にwhileループがあるものとして，perlが実行されます。つまり，**図7**に示すスクリプトを実行した場合と同じになります。「-p」オプションを付けると，perlをsedと同じような感じで使うことができる＊13ので，特にsedを知っている人には便利でしょう。

＊13　実際に図6の2行目のスクリプトは，そのままsedで使用できます。

図7の5行目に「print;」がある＊14ので，入力ファイルから読み込まれた行は，図7の3行目の置換命令実行後，標準出力へ出力されます。ところが，図6の1行目に「-i.bak」が指定されているため，標準出力への出力の代わりに，元の入力ファイルとの置き換えが行われます。例えば，

```
public_html/docs/NikkeiLinux
99-11/Welcome.ja.html
```

というファイルがあったとして，図6を実行するとこのファイルは，

```
public_html/docs/NikkeiLinux
99-11/Welcome.ja.html.bak
```

というファイル名に変更され，図7の3行目の置換命令によって置換が行われた後の内容が，元のファイル名である

```
public_html/docs/NikkeiLinux
99-11/Welcome.ja.html
```

へ出力されます。入力ファイルが複数ある場合は，それぞれに対して同様の置換が行われます。

単機能ツール

perlは，いわば万能ツールで，shスクリプトでできることはすべてperlのスクリプトでも記述可能ですから,極端な話，コマンド・ラインで行う作業をすべてperlスクリプトで行うことも可能です。しかし，UNIX互換OSには便利なツールが数多く用意されていて，特定の目的には専用の単機能ツールを用いた方が簡便です。

例えば，あるディレクトリ内の＊.htmlファイルをすべて列挙したい場合を考えます。ファイル列挙のための専用コマンドであるfindを用いれば，**図8**のように簡単にファイルを列挙できます。

ところが，同じことをperlスクリプトだけで行おうとすると，**図9**のようにコマンド・ラインから指定するには少々長すぎるスクリプト＊15になってしまいます。

図9の4行目から18行目が，引数に指定したディレクトリ内の＊.htmlファイルを列挙するためのサブルーチンdirです。8行目で，ディレクトリ内のファイルあるいはディレクトリを1つずつ読み出し，ファイルならば（10行目）ファイル名の末尾が「.html」の場合に限り表示します（11行目）。ディレクトリならば（12行目），これを引数としてサブルーチンdirを再帰的に呼び出します。

従って，"public_html/docs"を引数としてサブルーチンdirを呼び出せば（2行目），public_html/docs内にある＊.htmlファイルの列挙が行われます。図9のスクリプトをファイルtest.plに保存すれば，

```
1  #!/usr/bin/perl
2  while (<>) {
3      s@/theme\.ja\.html\"@/theme.ja.html#NIKKEI\"@;
4  } continue {
5      print;
6  }
```

図7　図6と等価なperlスクリプト

```
asao:/home/sengoku % find public_html/docs -type f -name '*.html'↵
public_html/docs/uucp/index.html
public_html/docs/Welcome.html
public_html/docs/NikkeiLinux99-11/Welcome.ja.html
public_html/docs/NikkeiLinux99-11/2ndFort.ja.html
    ...
```

図8　findコマンドを使った*.htmlファイルの列挙

＊14　「-p」オプションは，このようにデフォルトで標準出力への出力が行われますが，「-p」オプションの代わりに「-n」オプションを使えば，この出力は行われません。
＊15　コマンド・ラインではなく，スクリプト・ファイルを書く場合は，十分短いスクリプトと言えます。その他のプログラミング言語で実現する場合と比べると，perlで書いたスクリプトは極めて簡潔と言えるでしょう。

```
 1  #!/usr/bin/perl
 2  &dir("public_html/docs");
 3
 4  sub dir {
 5      my($dir) = @_;
 6      local(*DIR);
 7      opendir(DIR,$dir);
 8      for (readdir(DIR)) {
 9          my($file) = "$dir/$_";
10          if (-f $file && ! -l $file) {
11              print "$file\n" if /\.html$/;
12          } elsif (-d $file) {
13              next if /^\.{1,2}$/;
14              &dir($file);
15          }
16      }
17      closedir(DIR);
18  }
```

図9　perlスクリプトによる*.htmlファイルの列挙

```
asao:/home/sengoku % ./test.pl↵
public_html/docs/uucp/index.html
public_html/docs/Welcome.html
public_html/docs/NikkeiLinux99-11/Welcome.ja.html
public_html/docs/NikkeiLinux99-11/2ndFort.ja.html
    ...
```

図10　図9のperlスクリプトの実行

```
asao:/home/sengoku % find public_html/docs -type f -name '*.html' \↵
| xargs grep '/theme.ja.html">'↵
public_html/docs/NikkeiLinux99-11/Welcome.ja.html:| <a href="../../theme.ja.html">up</a> |
public_html/docs/NikkeiLinux00-03/Welcome.ja.html:| <a href="../../theme.ja.html">up</a> |
    ...
```

図11　置換対象を確認する

図10のように実行できます*16。図8のコマンドと同じ出力が得られることを確認してください。

perlと単機能ツールの組み合わせ

コマンド・ライン・インタフェース(CLI)において効率よく作業を進めるには，万能ツールであるperlと，単機能ツールとをどう組み合わせるかがかぎとなります。つまり，単機能ツールの守備範囲はそれぞれのツールに任せて，それ以外の部分，あるいはperlが得意とする部分をperlスクリプトで書くわけです。

ここでは，具体的な作業*17を例に，コマンド・ラインの組み立て方を解説します。

作業例:
ディレクトリ~/public_html/docs内のHTMLファイルにおいて，
<a href="../../theme.ja.html">というリンクがあれば，これを
<a href="../../theme.ja.html#NIKKEI">に書き換える。
ただし，このリンクが無いファイルは変更しない。

まず，public_html/docs内のHTMLファイルは，図8に示したfindコマンドで列挙できます。そこで，このfindコマンドの出力をgrepコマンドに与えることによって，リンクを書き換えるファイルの内容を確認*18します(**図11**)。

ここではfindコマンドの出力をgrepコマンドの引数に与えるために，xargsコマンドを用いています。xargsコマンドは，引数に与えたコマンド・ラインに，標準入力から受け取った引数リストを付加した上で実行するためのコマンドです。

＊16　実行するには，「chmod +x test.pl」などと実行して，test.plの実行許可フラグを立てる必要があります。
＊17　実は，この作業は必要に迫られて実際に行った作業です。
＊18　意図しないファイルを書き換えてしまわないよう，まず確認すべきでしょう。

bashをはじめとするほとんどすべてのシェル(コマンド・ライン・インタプリタ)には, findコマンドなどを「`」(逆引用符)でくくることにより, その出力を他のコマンドの引数として与えることができます(**図12**)が, 引数リストが長すぎるとエラーになるコマンドもあるので注意が必要です。

引数リストが長くなる可能性がある場合は, 図11のように, xargsコマンドを使った方が無難でしょう。

次に, 図11の実行結果から, ファイルの内容を表示している部分を取り除きます。図11のgrepコマンドの引数に「-l」オプションを加えれば, OKです(**図13**)。あるいは, **図14**のように, findコマンドが直接grepコマンドを呼び出す方法もあります。

つまりgrepコマンドの終了ステータスが0(検索対象が見つかった)であるファイル名のみを, findコマンドが列挙します。

さて, いよいよperlを使った置換です。図6のperlコマンドに, 図13で得られたファイル名リストを与えればOKなのですが, その前にまず置換のためのperlスクリプトが正しく動くか確認すべきでしょう*19。図15のように, ファイル名リストを与えずにperlを実行してみて, 標準入力から置換対象となるファイルの内容を入力して, 置換結果を確認します。

最後に, 図14で作ったファイル名リストをxargsコマンドを使って, **図15**のperlコマンドへ与えます(**図16**)。こうして作ったコマンド・ラインは結構な長さになりますが, Emacs上のshellモード*20ならば, コピー・アンド・ペーストするだけでコマンド・ラインを組み立てることができます。

```
asao:/home/sengoku % grep '/theme.ja.html">' \
`find public_html/docs -type f -name '*.html'`
```

図12 xargsコマンドの代わりにシェルの機能を利用する

```
asao:/home/sengoku % find public_html/docs -type f -name '*.html' \
| xargs grep -l '/theme.ja.html">'
public_html/docs/NikkeiLinux99-11/Welcome.ja.html
public_html/docs/NikkeiLinux00-03/Welcome.ja.html
    ...
```

図13 置換対象のファイル名の列挙(その1)

```
asao:/home/sengoku % find public_html/docs -type f -name '*.html' \
-exec grep -q '/theme.ja.html">' {} \; -print
public_html/docs/NikkeiLinux99-11/Welcome.ja.html
public_html/docs/NikkeiLinux00-03/Welcome.ja.html
    ...
```

図14 置換対象のファイルの列挙(その2)

```
asao:/home/sengoku % perl -pi.bak \
-e 's@/theme\.ja\.html\"@/theme.ja.html#NIKKEI\"@' \
| <a href="../../theme.ja.html">up</a> |
| <a href="../../theme.ja.html#NIKKEI">up</a> |
(C-d)
```

図15 標準入力を用いた置換のテスト

```
asao:/home/sengoku % find public_html/docs -type f -name '*.html' \
-exec grep -q '/theme.ja.html">' {} \; -print \
| xargs perl -pi.bak \
-e 's@/theme\.ja\.html\"@/theme.ja.html#NIKKEI\"@'
```

図16 図14と図15のコマンドを組み合わせる

*19 どんな短いスクリプトでも, バグの入り込む余地があります。動作を確認しつつ, 少しずつ進めるべきです。
*20 本連載第1回「キャラクタ・ユーザー・インタフェース」を参照。