計算機科学'11 ＃3 —
スクリプトとテキスト処理/Webアプリケーション

久野　靖*

2011.6.7

1　スクリプトとスクリプト言語

1.1　スクリプトとは…

スクリプト (**script**) とはもともと「台本」のことで、計算機の世界では「実行しなければならないことを順番に書き連ねたもの」といった意味で使われてきました。たとえば「計算機科学基礎」のシェルのところで取り上げた、シェルコマンドをファイルに書いたもの (**シェルスクリプト**、Windows/MS-DOS でいうと**バッチファイル**) は、スクリプトの元祖だと言えます。

動作を順に書いたもの、というのは要するに「プログラム」なわけで、つまりプログラムを書くことで1つひとつ人間が指示しなくてもいいようにしましょう、という非常にアタリマエのことを意味しているわけです。そんなアタリマエがどうして重要なのでしょうか。

たとえば GUI である操作をやるのに、範囲を選択してマウス操作でメニューを出して選んで、それで完了、簡単でいいね、と思っていたとしましょう。しかしその操作を 100 個のファイルについてやらなければならないとしたら? テキストのあちこちにある 100 箇所についてやらなければならないとしたら? それはまさに「苦行」ですね? 100 ならまだ我慢するにしても、1,000 だったら? 10,000 だったら? あなたはまだその「苦行」を続ける気がありますか?

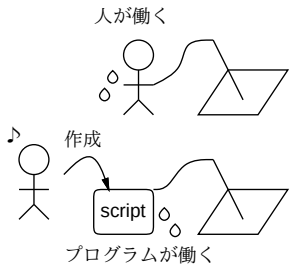


図 1: スクリプトの考え方

もちろん、そういう「苦行」から人間を解放することこそ計算機の本来の存在意義なわけで、計算機のためにそんな苦行をするというのは本末転倒以外の何者でもありません。ではどうするか? 同じことを繰り返すのはプログラムを書いてプログラムにやらせればよいわけです (図 1)。[1]そしてそれを「さっと」簡単にできる、というのがスクリプトの思想です。今日では、スクリプトを書くための言語、つまり**スクリプト言語**が多く作られ使われるようになっています。さらに、これまでは「傍流」だったスクリプト言語を使って多くのアプリケーションが書かれるようになってきています。

*経営システム科学専攻

[1]ここでもしも、データが GUI でしか操作できないものだと厄介なわけですが、Unix の文化ではデータはテキストファイルであり、スクリプトで操作できるわけです。

今回はこのような状況を背景に、スクリプト言語とその応用を取り上げます。以下次節でスクリプト言語の歴史と現状について概観した後、具体的なスクリプト言語として Perl(汎用/ファイル操作)、PHP(Web のサーバ側プログラミング)、JavaScript(Web のクライアント側プログラミング) の 3 つを取り上げて順次見て行くことにします。

1.2 スクリプト言語の歴史と現状

先に述べたように、スクリプト言語の原点は「コマンドを並べてファイルに格納したもの」にあります。Unix であればこれはシェルスクリプトですが、Unix 以外の (そして Unix 以前の) 多くのオペレーティングシステムでもコマンドの並びをファイルに入れて実行させることが可能でした。また、その一部は「枝分かれ」「ジャンプ」のような制御構造も持っていました。そういう意味では、スクリプトは最初からプログラムだったと言えます。

ただし、シェルスクリプトが重要なのは、Unix の豊富なツール (フィルタ類) を組み合わせて柔軟なデータ処理が行えたことと、if、while など構造化言語と同様の制御構造を取り入れて言語としての体裁を整えたことにあります。たとえば bash で次のようなことができます。

```
% for i in *
> do
> echo $i $i
> done
temp temp           ←各ファイル名を
test.txt test.txt   ← 2 回打ち出す
%
```

このようにループを使って多数のファイルをまとめて処理することなどもできるわけです。

しかし、シェルはもともと「その場で人間がコマンドを打ち込む」ことが基本になっているため、データを保持し扱う機能などはあまり充実したものとは言えませんでした。フィルタを組み合わせることで様々な処理はできますが、「ファイル単位」のまとまったものであるので、1 行ずつ順に処理するようなものには向きません。このため、フィルタの 1 つとして簡単なプログラム言語を処理するプログラムを用意することも行われましたが (**AWK** という言語が有名 — 図 2)、上記の問題を十分解消するのには至りませんでした。

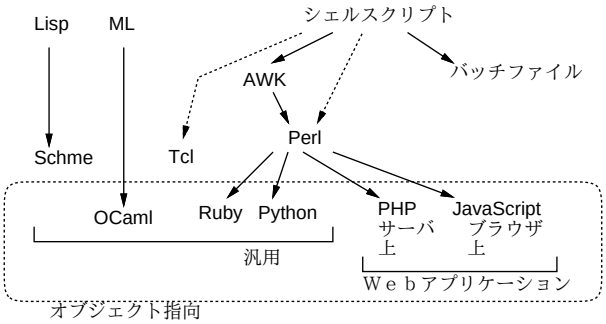


図 2: スクリプト言語の系譜

この流れを変えたのが、1988 年に Larry Wall によって開発された **Perl** 言語です。Perl はそれ以前の文字列処理に適したプログラミング言語の機能も参考にしつつ、1 つの言語で Unix のフィルタ+シェルスクリプトと同等以上のことが自在に書ける言語として設計されました。またその処理系は**インタプリタ**方式により、コンパイルなどの処理を経ずすぐに実行可能であり、シェルスクリプト同様「さっと」プログラムを作って使えるようになっていました。この特性により、Perl は広く普及し、

「スクリプト言語」という新しいプログラミング言語のカテゴリを築くこととなりました。この時期のスクリプト言語としては Perl のほかに **Tcl** などもあります。

1990 年代後半になると、一般のプログラミング言語においても C のような**構造化言語**から C++や Java のような**オブジェクト指向言語**への移行がすすみ、その利点も広く知られて来ました。これに伴い、Perl もオブジェクト指向機能を追加した Perl5 に進化しましたが、その追加の方法はそれほどスマートとは言えませんでした。

そこで Perl に代わるものとして、最初からオブジェクト指向機能を前提にデザインされたスクリプト言語が探求されるようになりました。現在ではオブジェクト指向スクリプト言語としては **Python** と **Ruby**(日本発です) が代表的です。

また、WWW の普及期に、ブラウザにスクリプト言語を組み込むというアイデアが出され、そのための言語として **JavaScript** が考案されました。JavaScript もオブジェクト指向スクリプト言語であり、ブラウザに組み込まれるという性質上、最も処理系が普及したスクリプト言語だと言えます (ブラウザを載せていない PC はまずないでしょうから)。

このほか、人工知能研究に使われて来た **Lisp** 言語の流れを汲む **Scheme**、関数型言語 **ML** の流れを汲む **Ocaml** なども処理系が小さく「さっと使える」ことからスクリプト言語として使われることもあります。

なお、人によっては、スクリプトという言葉が「お手軽」というイメージを持つことを嫌って、これらの言語のことを**軽量言語** (**light language**) と呼ぶこともあります。

2 Perl とテキスト処理プログラミング

2.1 文字の処理の歴史

計算機が最初に作られたときはその目的は文字通り「計算する」ことだったので、扱うのは数値が中心でした。文字も符合化すればビットの列で表せ、計算機に取り込めることは既にやった通りですが、最初のころは入出力装置が CPU と文字をやりとりするのに符合化コードを使うから必要、という程度のものでした。このため、最初のころは文字を扱う機能を中心に据えたプログラム言語は実験的なものに限られていました。

しかし、事務計算の場合などでは帳票を出力する際には計算した数値といっしょに顧客名や商品名なども打ち出したいわけです。そのため、事務処理言語 (**COBOL** が代表的) を中心に、文字を扱う機能がいろいろつけ加えられ、計算機で文字を扱うことは当たり前になってきました。それでも、77 年規格になる前の Fortran では変数に文字型がなかったくらいです。

しかし結局、人間が扱う情報のうち数値だけ、というのは比較的少なく、一方文字は文章 (テキスト) の形をとればどんな情報でも扱えるわけですから、その処理の比重が高まるのは当然のことでした。というわけで現在では、汎用の手続き言語では文字が自由に扱えることは当たり前になっています。

ただし、汎用の手続き型言語で文字列を扱うのは結構面倒な面がありました。現在ではオブジェクト指向機能の採用により、比較的容易に文字列が扱えるようになっていますが、そのような動きと並行して「もっと柔軟に/簡単に文字列を扱いたい」という要望やそれに応える動きもあります。

その結果、「ささっと書く」スクリプト言語の中からテキスト処理に適したものが生まれるようになりました。Perl はその代表格といえます。以下でも Perl の文字列処理を中心に扱います。

2.2 テキスト処理のための言語機能 OPTION

数値に対しては四則演算や三角関数など各種の「演算」があるのと同様、テキスト (文字列) に対しても各種の「演算」があって、特に文字列処理言語の場合はそれが効率よく扱えるような言語機構が備わっています。具体的な「演算」としては次のようなものが挙げられます:

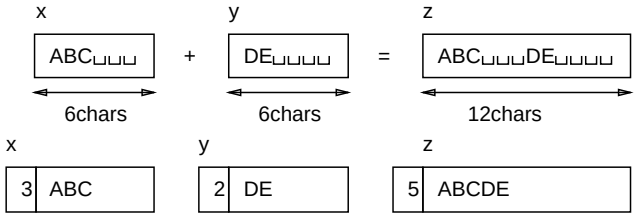


図 3: 固定長文字列と可変長文字列

- 文字列と文字列を連結する。
- 文字列の中から部分文字列を取り出す。
- 文字列の中に、指定した部分文字列があるかどうか探索する。
- パターンマッチング
- 各文字を規則に従って別の文字に置き換える (mapping)。
- 探索やパターンマッチで見つかった部分文字列を他の文字列に置き換える。

これらは一見 (パターンマッチを除いては) 別に難しくないように思えるかも知れませんが、実際にやると色々問題が起きます。たとえば文字列の連結ですが、これが意味を持つためにはまず文字列の長さが可変長である必要があります (なぜか?)。ちなみに、可変長というのは、ある変数に入る文字列の長さが可変である、という程度の意味である。例えば標準 Pascal や Fortran や COBOL の文字列は可変長ではなく固定長です (図 3)。

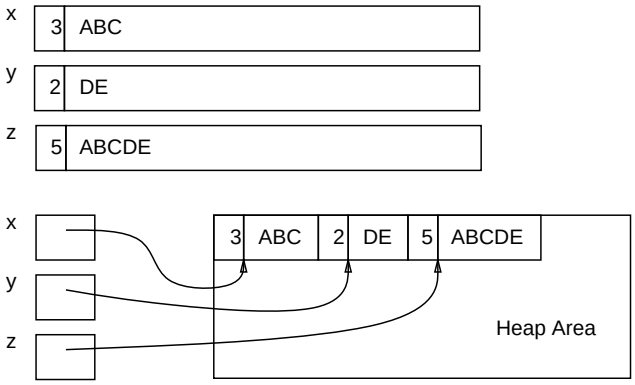


図 4: 変数領域方式とヒープ方式

次に可変長だと、その長さは連結していくとどんどん長くなる、という問題もあります。適当な上限を設けるとしても、それぞれの変数にその上限まで入れられるだけ領域を用意する、というのはあまりよい方法ではありません (なぜか?)。で、結局柔軟に文字列を扱うためには文字列をヒープ (ある程度の大きさのメモリ領域を確保しておき、実行時にそこから必要なだけ切り取って来て利用すること) に入れてごみ集め (不要になった切り取り部分を回収して来てあとで再利用できるようにすること) をするようなシステムが必要になってきます (図 4)。

このような方法が性能の面でも有利な場合が多くなります。例えば部分文字列の取り出しにしても、変数領域に直接入れる方法だと毎回コピーが必要ですが、ヒープを使う場合には「どの場所」ということだけ覚えておけば済むように工夫することもできます。

次に、部分文字列の探索ですが、これは素直に考えれば図 5 上のように探したい文字列を 1 番目、2 番目、... の位置に置いてみてそれであてはまるかどうかを調べればよいわけです。しかし、この素朴な方法だと最悪の場合「探される文字列の長さ×探す文字列の長さ」回の比較が必要になり、ひど

く効率が悪くなることがあります (図 5 下)。これを避けるため様々な「高速文字列探索アルゴリズム」が研究されています。パターンマッチの場合も同様です。

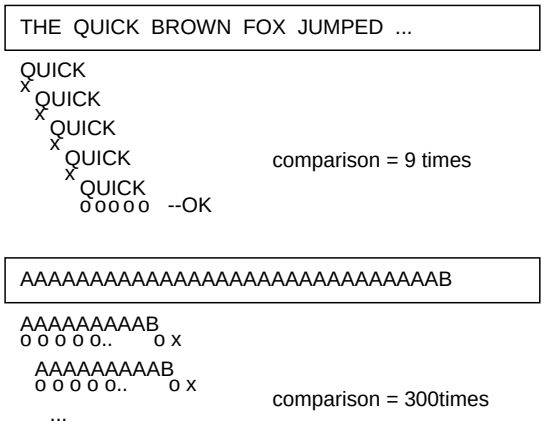


図 5: 2 次マッチングによる探索

とはいえ、これらの機能は文字列処理言語の場合その処理系内部に実現済みなわけで、利用者はその言語でプログラムを書けば良いだけだから話は簡単です。現在のスクリプト言語の繁栄にはこのような背景があると言えます。

なお、文字列処理は常に文字列処理言語で書かなければならない、というものではなく、例えば C など普通の言語でももちろん可能です。ただしその場合には上で述べたような問題を自分で何とかする必要が生じて結構大変です。[2]一方、よく行なうパターンの文字列処理については `sed`, `grep`, `tr`, `sort`, `uniq` などのフィルタを駆使すればわざわざプログラムを書かなくても済む、という面もあります。

2.3 Perl 入門

前述のように、Perl は Larry Wall によって開発された、近代的スクリプト言語の元祖であり、非常に豊富な機能を持っています。しかしその一方で、全部の機能をマスターしなくても、とりあえず「こう使う」というところだけわかればそれで役に立つという実用的な側面も持っています。Perl の基本的な特徴は次の通りです。

- 変数はシェルのように「`$`」をつけて表す。変数は宣言も型指定も不要で、使った時に作られ、数値でも文字列でも自由に入れられる。
- 文字列は「`'...'`」でも「`"..."`」でも表せるが、これもシェルと同じく、前者は完全に「そのまま」の文字列だが、後者はその中に変数があると変数の値が埋め込まれる。
- if 文や for 文などの本体部分はすべてブロックである必要がある。つまり「`{ ... }`」で囲む必要がある。

たとえば `test1.pl` に次のような内容が入っているとします。

```
for($i = 0; $i < 10; ++$i) {
  print " ", $i;
}
print "\n";
```

これを動かすには、perl コマンドを使って次のようにします。

[2]実際にはわざわざ書かなくてもライブラリルーチンとして用意されている機能で大体用が足りますが。

```
% perl test1.pl
 0 1 2 3 4 5 6 7 8 9
%
```

Perl の print 命令は出力するものをいくつでも指定でき、数値は適当に文字列に変換されて出力されます。また、"\n"(改行文字) を出力しない限りはどんどん続けて同じ行に出力がなされていきます。

2.4 Perl のファイル入出力

次に、Perl でファイルを読み込む方法を見てみましょう。Perl では Unix のファイルディスクリプタに相当するものを「ファイルハンドル」といい、そこから読み込む時には「<名前>」という形を使います。とくに名前を空っぽにした「<>」だと、「プログラム実行にファイル名を指定したときはそのファイルから順に、指定していないときは標準入力から読む」という Unix のフィルタで一般的な動作と同じになります。ですから、次のプログラムで入力をそのまま出力するプログラム、つまり cat と同じプログラムになります。

```
while($line = <>) {
  print $line;
}
```

動かしてみましょう (上のプログラムは test2.pl に入っているものとします)。

```
% cat t1
This is a pen.  ← t1 の内容
% perl test2.pl t1 t1  ← t1 を 2 回指定したので
This is a pen.
This is a pen.  ← 2 回連結されている
%
```

なお、ファイルから読み込んだ時は行末の「"\n"」までくっついた文字列が読み込まれています。

出力については、ファイルハンドルと既に学んだ print を組み合わせて使います。まず open でファイルを開くと同時にファイルハンドルを作ります。

```
open(F, ">test.txt");  ←「>」は「出力」をあらわす
```

そのあと、ファイルハンドルを指定した print を何回でも実行できます (ファイルハンドルの後には「,」は不要なのに注意)。

```
print F "this is a pen.\n";
```

出力が全部終わったら、ファイルを閉じます。これでファイルが完成します。

```
close(F);
```

2.5 フィールドに分かれたデータの処理

先の例ではただ単に入力した行をそのまま出力していましたが、今度は各行の中のデータを処理してみます。いちばんよく使う方法は、split を使ってデータを複数のフィールドに分けて扱うことです。例として、次のようなデータファイルを扱うものとしましょう:

```
久野  20 180  60
大木  10  60 170
吉田 190 100 100
```

何のデータかは分かりませんが、3 つの数値はそのデータの「1970 年、1980 年、1990 年」の値だということにして、それぞれの人ごとに平均を計算することにします。

ここで、行が変数$line に入っているとして、まず chop($line); というのを実行して末尾の改行文字を削除します。次にフィールドに分けるのには、次のように split という関数を使います:

```
($name, $d1, $d2, $d3) = split(/ +/, $line);
```

「/ +/」は「空白 1 個以上」を表すパターン (正規表現) で、split は空白が 1 個以上あるところで行を分割します。そして分割したそれぞれの部分を左辺の変数$name、$d1、$d2、$d3 に入れるわけです。なお、データの形式が CSV の場合はパターンに「/,/」を指定すればよいでしょう。

では、平均を計算するプログラムを示します:

```
while($line = <>) {
  chop($line);
  ($name, $d1, $d2, $d3) = split(/ +/, $line);
  $avg = ($d1 + $d2 + $d3) / 3.0;
  print "$name   $d1   $d2   $d3  $avg\n";
}
```

フィールドに分けて、平均を計算して、print するだけです。"..."で囲んだ文字列の中に変数を書くと、出力時にその場所に変数の値が埋め込まれます:

```
perl test3.pl t.data
久野   20   180   60  86.6666666666667
大木   10   60   170  80
吉田   190   100   100  130
```

2.6 他形式のファイルの生成

上のような処理だけだったら、もちろん表計算ソフトの方が簡単ですね。せっかく Perl なので、出力をただのテキストにする代わりに、latex にしてみましょう。それには latex の制御命令を一緒に埋め込んで出力すればいいだけです:

```
print "\\documentclass[12pt,a4j]{jarticle}\n";
print "\\usepackage{graphicx}\n";
print "\\begin{document}\n";
print "\\title{データ表}\n";
print "\\maketitle\n";
print "\\begin{center}\n";
print "\\begin{tabular}{|c|r|r|r|c|}\n";
print "\\hline\n";
print "名前 & 1970 & 1980 & 1990 & 平均\\\\\n";
print "\\hline\n";
$num = 0;
while($line = <>) {
  chop($line);
  ($name, $d1, $d2, $d3) = split(/ +/, $line);
  $avg = ($d1 + $d2 + $d3) / 3.0;
  print "$name & $d1 & $d2 & $d3 & $avg\\\\\n";
  print "\\hline\n";
```

```
}
print "\\end{tabular}\n";
print "\\end{center}\n";
print "\\end{document}\n";
```

プログラムの大部分は latex のソースを生成するための `print` 命令で、表の各行を処理するループだけが先と同様の計算をおこない、ただし出力するときは各フィールドを`&`で区切って出力しています。なお、「\」はエスケープ文字 (特別な処理を行う文字) なので、1 つ出力するためには文字列の中では 2 つ続けて書く必要があります。では実際に動かしてみましょう。

```
perl test4.pl t.data >test.tex
platex test.tex
…以下 latex の回と同様…
```

生成されるソースの中の、表の部分だけを示しておきます:

名前	1970	1980	1990	平均
久野	20	180	60	86.6666666666667
大木	10	60	170	80
吉田	190	100	100	130

2.7 複数のファイルを生成する

もっと頑張って、平均の数値ではなく、グラフを生成してみましょう。それには PostScript 形式でグラフを出力することにします。PostScript については#5 でやるので、今回は次の 5 つの命令だけ覚えておいてください。

- `newpath` — 新しい線引きを開始する。
- X Y `moveto` — ペンを指定した座標 (X, Y) に移動。
- X Y `lineto` — 座標 (X, Y) までの線を登録しながら移動。
- `stroke` — `lineto` で指定した線引き一式を実行する。
- W `setlinewidth` — 線の太さを Wpt にする。

以下のプログラムでは、個人ごとに 300 × 200pt (1pt = $\frac{1}{72}$inch) の描画面を用意し、そこに上の操作を使ってグラフを描きます。実際にはこれは 1 人に 1 つずつ、`fig1.ps` のような名前の PostScript ファイルを開いて、そこに命令を書き込んでいます。latex の中では次のような命令を使ってそのファイルを埋め込むことができます。

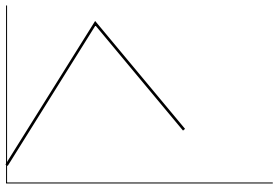
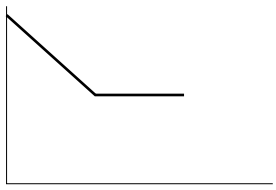
```
\includegraphics[scale=倍率]{ファイル名}
```

プログラムを見てみましょう (「.」は文字列連結の操作を表します):

```
print "\\documentclass[12pt,a4j]{jarticle}\n";
print "\\usepackage{graphicx}\n";
print "\\begin{document}\n";
print "\\title{データ表+グラフ}\n";
print "\\maketitle\n";
print "\\begin{center}\n";
print "\\begin{tabular}{|c|r|r|r|c|}\n";
print "\\hline\n";
```

```
print "名前 & 1970 & 1980 & 1990 & グラフ\\\\\n";
print "\\hline\n";
$num = 0;
while($line = <>) {
  chop($line);
  ($name, $d1, $d2, $d3) = split(/ +/, $line);
  $file = "fig" . ++$num . ".ps";
  open(F, ">".$file);
  print F "%!PS-Adobe-2.0\n";
  print F "%%BoundingBox: -5 -5 310 210\n";
  print F "newpath 0 200 moveto 0 0 lineto 300 0 lineto stroke\n";
  print F "3 setlinewidth\n";
  print F "newpath 0 $d1 moveto 100 $d2 lineto 200 $d3 lineto stroke\n";
  close(F);
  print "$name & $d1 & $d2 & $d3 &";
  print "\\includegraphics[scale=0.333]{$file}\\\\\n";
  print "\\hline\n";
}
print "\\end{tabular}\n";
print "\\end{center}\n";
print "\\end{document}\n";
```

変数`$file`に PostScript のファイル名を生成し、それを出力指定で `open` し、そこに PostScript を書き出して `close` します。latex の側では平均は計算せず、代わりに `includegraphics` で生成した PostScript ファイルを埋め込みます。今度は表の部分は次のようになります:

名前	1970	1980	1990	グラフ
久野	20	180	60	
大木	10	60	170	
吉田	190	100	100	

3 PHP と Web サーバ側プログラミング

3.1 PHP 言語とその由来

前回も取り上げたように、Web アプリケーションは基本的には HTML で記述されたフォームによるユーザインタフェースと、そこから提出されてきたデータを処理する、サーバ側の CGI プログラ

ムとの組み合わせでできています (今日では Ajax などもっと込み入った形態のものもありますが)。

CGI プログラムとは、CGI(Common Gateway Interface) と呼ばれる規約に従って動作するプログラムのことで、この規約が Web サーバとプログラム、Web ページとプログラムの間のやりとり方法を定めています。CGI プログラムは、WWW の初期には C 言語など普通のプログラミング言語で書かれたり、シェルスクリプトを活用して作られたりしてきましたが、その性質上、文字列の処理を多く必要とし、Perl などのスクリプト言語との相性がよいため、現在では Perl をはじめとするスクリプト言語で作られることが多くなっています。

そのような言語のひとつ、**PHP**(Hypertext Processor — どこに最初の P があるんだろうと思うでしょうが、その初期において Personal Home Page tools と呼ばれていたことに由来するらしい) は、Web サーバに埋め込んで実行するスクリプトを書くためのプログラミング言語として、1995 年に Rasmus Lerdorf によって開発されました。その後開発者も実装も変遷して、現在は 2004 年に公開された PHP5 が最新版となっており、以下の説明もこれを対象とします。

PHP の最大のアイデアは、HTML を記述する中に PHP 言語のプログラムが埋め込まれてサーバ上で動くという方式を確立したことで、現在では MS ASP(Active Server Page)、JSP(Java Server Page) など多くのものがそのマネをしています。もう 1 つの特徴は豊富なモジュールによってインターネット上のさまざまな標準をサポートしていることで、そのために PHP のマニュアルは (そして解説本も) ぶ厚いものとなっています。

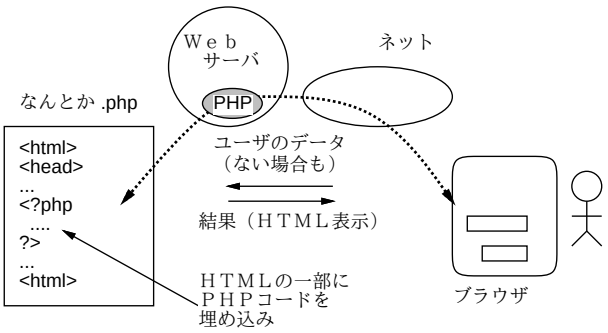


図 6: PHP によるデータの受け取りと処理

さらに、現在標準の Web サーバとして広く使われている Apache ではモジュールとして PHP を組み込むことができ、「`.php`」で終わるファイルに対しては自動的に PHP 処理系が解釈実行するように設定できます。この場合、「`.php`」で終わるファイルは自動的に PHP 処理系が実行し、これによって (PHP 言語で記述された) ユーザデータの処理やページ内容の処理による生成が可能になります (図 6)。我々の内部サーバもこのように設定してあります。

一般に、ユーザ側から PHP プログラムを参照する形は次のどちらかの方法になります。

- 「なんとか`.php`」の URI を自分で開いたりリンクを通じて開く — データは送信しないので、PHP 側では「最初に開かれた」ものとして処理を行う。
- フォーム要素の `action` 属性として「なんとか`.php`」の URI が指定されていてそこに送信する — データが送られるので、PHP 側では「送られたデータを処理し、新たな (結果の) ページを返送する」ものとして処理を行う。

以下では、PHP 言語を使うとどのように簡単に CGI プログラム、つまり Web アプリケーションのサーバ側プログラムが構成できるか、ということを見物して行きたいと思います。

3.2 PHP 言語の基本的な構成

最初に知っておくべきことは、Web で使用する場合の PHP プログラムは HTML ファイルの中に次のような形 (**処理命令**と呼ばれるものの一種) として埋め込む、ということです。

```
<?php …PHP プログラム… ?>
```

なお、PHP プログラムの部分は何行に渡っても構いません。PHP プログラムの中では C や C++や Java と同じコメント (/* ... */」で囲むコメント、または「//」(「#」でもよい) から行末までのコメントが使えます。

もう 1 つ重要なことですが、PHP プログラムの中で HTTP ヘッダを出力する機能 (header() 関数) を使うときは、一切の通常出力を行う前に行う必要があることです。たとえば、PHP で日本語が正しく表示できるように文字エンコーディングを指定するときにも header() が必要です。なので、PHP プログラムをサーバ上に置く時は次の形のものを使うとよいでしょう (文字化けの問題がないなら、最初のヘッダ生成などは省略してもよいです):

```
<?php
  header("Content-type: text/html; charset=euc-jp");
  /* その他ヘッダ出力はここで */
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>title...</title>
<style type="text/css">
/* CSS 記述はここに */
</style><head><body>
<?php
  /* ページ本体出力を含む動作 */
?>
</body></html>
```

ファイル中に日本語を含める場合は、ファイルの文字コードは EUC にしてください。Emacs であれば「Ctrl-X [RET] f euc-jp [RET]」を実行することで編集中のファイルの文字コードを EUC にできます。ステータス表示の最後が「…E」となっていれば EUC ですので、そうなっていない時だけ上記を実行すればよいでしょう。

ではまず最初に、PHP で単純に HTML を生成するという簡単なサンプルを示します (図 7)。

```
<php? header("Content-type: text/html; charset=euc-jp"); ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>php sample</title>
<style type="text/css">
div.num { margin: 2px 20px; background: rgb(200,215,255) }
</style></head><body>
<h1>数を並べる</h1>
<?php
  for($i = 0; $i < 10; ++$i) {
    echo "<div class='num'>番号{$i}です。</div>";
  }
?>
</body></html>
```

資料では見やすいように下げてありますが、実際のファイルでは最初の「<?php」より前に 1 文字の空白もあってはいけません (あるとページ内容が出力されてしまい、header() が失敗します)。

この例は、単に「番号 i です。」という div 要素が 10 個出力されるだけです。ソースと対比すればプログラムが動いて出力していることに納得が行くと思います。echo 命令は文字列を出力する機能で、文字列中に変数を埋め込む場合は (Perl とはちょっと違って) 「{$変数名}」という形を使います。

図 7: PHP による簡単なページ生成

また、CSS を使って div 要素のマージンと背景色を指定することで、ページの見ためをそれらしくしています。

3.3 フォームデータの受け取り

PHP から HTML フォームのデータを受け取るのは非常に簡単で、form 要素からの送信メソッドが get のときは「$_GET[’名前’]」、post のときは「$_POST[’名前’]」を参照するだけです (「名前」は form 内の入力部品の name 属性で指定した名前を指定します)。

また、関数 isset() を使ってその名前のデータが送られているかどうかを判定することもできます。これを利用して、「最初に呼び出された場合」と「データが送られて来た場合」を次のように枝分かれして扱うのが普通です (「名前」にはフォームで指定した名前の 1 つを指定すればよいでしょう)。

```
<?php
  if(isset($_GET['名前']) {
    フォームからのデータがある場合の処理
  } else {
    最初にページが表示される場合の処理
  }
?>
...
<form action="#">
...
</form>  ←送信先として自分を指定
```

では、先の例を手直ししてフォームにより等差数列の初項、階差、上限を指定できるようにしてみます。

```
<php? header("Content-type: text/html; charset=euc-jp"); ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>php sample</title>
<style type="text/css">
div.num { margin: 2px 0px; background: rgb(200,215,255) }
form { padding: 1ex; background: rgb(200,185,220) }
</style></head><body>
```

図 8: PHP によるフォームデータの受け取り

```
<h1>数を並べる 2</h1>
<div><form action="#">
<?php
  if(isset($_GET['init'])) {
    $i = $_GET['init']; $s = $_GET['step']; $m = $_GET['max'];
  } else {
    $i = 0; $s = 1; $m = 10;
  }
  echo "  初項:<input type='text' name='init' size='4' value='{$i}'>";
  echo "  階差:<input type='text' name='step' size='4' value='{$s}'>";
  echo "  上限:<input type='text' name='max' size='4' value='{$m}'>";
  echo "  <button>変更</button></form></div>";
  for($k = $i; $k < $m; $k = $k + $s) {
    echo "<div class='num'>番号{$k}です。</div>";
  }
?>
</body></html>
```

この例ではフォームの入力部品として、text 入力欄を 3 つと送信ボタンを用意しています。text 入力欄部分の出力を PHP 側で行っているのは、入力欄の value 属性に現在値を入れた形で出力したいためです。

3.4 共有データとファイル入出力

先の例ではデータを PHP で処理はしていましたが、各ユーザが入力したデータに対して処理をして返すだけでした。実際の Web アプリケーションの本質は、サーバ上のデータを読み書きすることで、複数のユーザがデータを共有できる点にあります。その簡単な例として「掲示板」のようなものを作ってみましょう (図 9)。

具体的には、掲示板に書き込むメッセージを PHP ソースと同じ場所にある msg.txt というファイルに蓄積していきます。このファイルは最初から存在している必要があり、また Web サーバのプロセス (ユーザ ID nobody で動作している) に読み書きできる必要があるので、次のようにして空 (実際には改行 1 バイトだけ) の内容を用意し、保護モードも「誰でも読み書き可能」にしておきます。

図 9: PHP による掲示板

```
% echo ’  ’ >msg.txt
% chmod a+rw mgs.txt
```

ファイルを読み書きする PHP の関数として、以下では次のものを使います (`file()` だけは後の例で使います)。

- `fopen`(ファイル名, モード) — ファイルを開く (読み書きの準備をする)。モードは「’r’」なら読み込み、「’w’」なら書き出し、「’a’」なら追加書き込みを意味する。この関数は**ストリーム** (読み書きチャネル) を返すので、それを指定して以後の読み書きを行う。
- `fclose`(チャネル) — チャネルを閉じる (対応するファイルの読み書きを終了する)。
- `fwrite`(チャネル, 文字列) — チャネルを指定して文字列を書き込む。
- `fread`(チャネル, バイト数) — チャネルを指定して指定バイト数だけ読み込み、内容を文字列として返す。
- `fsize`(ファイル名) — ファイルの大きさ (バイト数) を取得する。
- `file`(ファイル名) — ファイルの内容を 1 行 1 要素の配列として返す。

では掲示板プログラムを見てみましょう。冒頭でフォームの書き込みメッセージがあるかどうか調べ、ある場合はファイルを追記モードで開いて末尾に名前とメッセージを入れ、チャネルを閉じます。その後フォームを書き、最後にファイルの内容を全部読んで出力します。このようにすることで、初回でも書き込み時でも処理がほぼ共通になります。

```
<?php header("content-type: text/html; charset=euc-jp"); ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>php sample</title>
<style type="text/css">
div.msg { margin: 2px 0px; padding: 4px; background: rgb(200,215,255) }
form { padding: 1ex; background: rgb(200,185,220) }
</style></head><body>
<h1>掲示板</h1>
<?php
  if(isset($_POST[’msg’])) {
    $fd = fopen(’msg.txt’, ’a’);
    fwrite($fd, "<div class=’msg’>{$_POST[’namae’]}: {$_POST[’msg’]}</div>\n");
```

```
    fclose($fd);
  }
  $fd = fopen('msg.txt', 'r');
  echo fread($fd, filesize('msg.txt'));
  fclose($fd);
?>
<div><form method="post" action="#">
名前: <input type="text" name="namae"><br>
<textarea name="msg" rows="5" cols="40"></textarea> <button>書き込み</button>
</div></form>
</body></html>
```

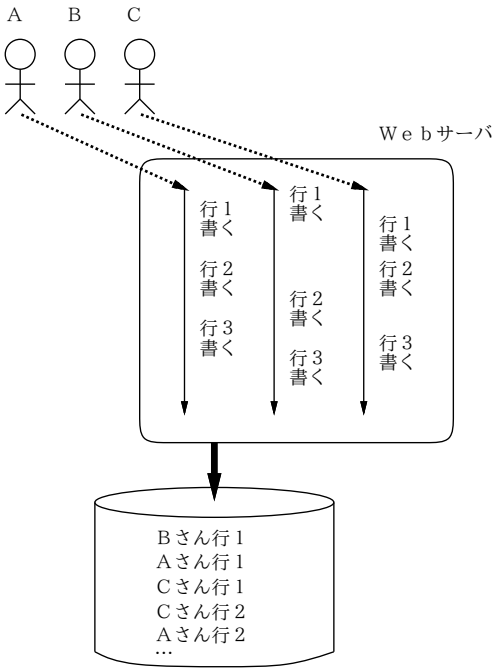


図 10: 並行アクセスの問題

ただし、このようなファイル共有型のアプリケーションには注意すべき点があります (図 10)。それは、Web アプリケーションは多数のユーザが Web サーバにアクセスしてきてて**並行**して実行されるという点です。このため、複数行に渡るファイルアクセスのような処理は、複数ユーザの処理が「重なって」実行されてしまい、正しくないデータが書かれてしまう可能性があるという点です。これを避ける方法としては、次のものがあります。

(a) 干渉しないような形でファイルアクセスを行う。

(b) ファイルにロックを掛ける。

(c) ファイルの代わりにデータベースシステムを使う。

今回の例題では、(a) の方法を用いています。すなわち、各ユーザの処理は「書き込み」と「読み出し」に別れていて、書き込みの処理は数十バイト程度のデータを追加書き込みするだけです。この程度の量なら 1 つのシステムコールで書かれるため、他のプロセスの処理と互い違いになる心配はほとんどありません (あったとしたも 1 個ぶん書き込みが失われる程度ですが、その場合はユーザにもう 1 度書いてもらってもそんなにまずくないでしょう)。

(b) については、さほどアクセスが多くないサーバでの処理ならいいのですが、ファイルをロックしてしまうと他のプロセスがファイルを使えるまで待たされるので、大量にアクセスがあると多くのプロセスが待たされてサーバに負荷が掛かるという弱点があります。そのため、多くのデータを扱い多数のユーザが共有する場合は (c) データベースシステムを利用する方法が一般的です。データベースシステムでは複数ユーザが並行してデータを処理することが基本的な要請なので、そのための機構が充実していて、性能上も問題ないように作られているからです。(このあたりは、次回取り上げます。)

4 JavaScript と Web クライアント側プログラミング

4.1 組み込みスクリプト

ここまでに見て来たスクリプト言語 (処理系) はいずれも、「〜をするのに便利なもの」という位置付けでした。[3]

これとは別のスクリプト言語の用途として、ある程度大きなソフトウェアがあったとして、それの動作を必要に応じて「調整」するのに利用する、というものがあります。これを、他のソフトウェアに「組み込む」ことから「組み込みスクリプト」などと呼びます。Scheme、Tcl、VBA(Visual Basic for Applications) などの言語がこの用途に多く使われています (図 11)。

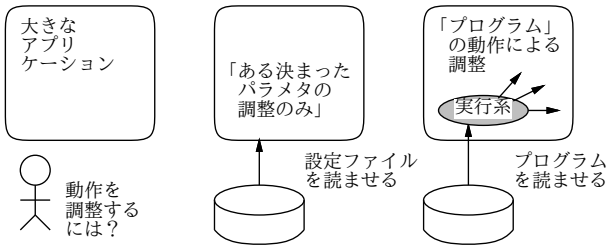


図 11: 組み込みスクリプト

組み込みスクリプトは何が嬉しいのでしょうか? 大きなアプリケーションでは複雑な動きを実現しているため、それをユーザの必要に応じて調整することは使いやすさのために重要です。では、さまざまな設定のためのオプションを用意すればよいでしょか? それだと、予め想定されたある一定の調整だけしかできないという弱点があります。これに対し、組み込みスクリプトであればプログラムが記述できるので、もっと柔軟な調整が可能になるわけです。

具体的には、組み込みスクリプトの処理系には本体のアプリケーションを「操る」ための入口ないし呼び出し点がいくつか用意してあって、これらを経由して本体アプリケーションを操作するわけです。

4.2 JavaScript とブラウザ

JavaScript は HTML の中に埋め込むことによって、そのページを表示している「間だけ」ブラウザの動作を調整することができるようなスクリプト言語とその処理系です (厳密に言えば JavaScript 言語でブラウザ組み込みでないものもあるのですが、現実に存在し使われている大多数の JavaScript 処理系はブラウザ組み込み型のものです)。

なぜ「間だけ」なのでしょう? それは、あるページを見おわって別のページに行っても前のページの JavaScript の効果が残っているようだと、それぞれのページの JavaScript プログラムが干渉して

[3]PHP の場合は Web サーバに処理系を組み込んでいますが、それは起動が速く効率がいいからとうだけで、以下で説明する組み込みスクリプトとしての用途とは違っています。

しまいますし、それに読み手のプライバシーを侵害するような危険なスクリプトが作れてしまう可能性が生じるからです。

そもそも別のページ云々とは独立に、JavaScirpt でブラウザを制御できる範囲はユーザにとってプライバシー侵害やセキュリティ上の危険が生じないように注意深く限定されています。このような安全性の保証は、Web 上にあってブラウザにダウンロードされてきて動くコードすべてに対して共通に課せられる課題だと言えます。(にもかかわらず、この種のコードによるセキュリティホールは今日でもしばしば問題になっています。)

具体的にブラウザ上の JavaScript でできることを整理すると次のようになります。

- ブラウザ自体の動作の制御 — 表示ページの切り替え、1 つ前のページに戻る、窓サイズの変更、新しい窓を開く、ダイアログを出すなど。
- イベントへの応答 — ボタンや任意の要素をクリックしたりその上にマウスポインタを置いた時に動作を起動できる。
- フォーム部品の値の参照と書き換え — フォーム部品 (GUI 部品) の値を読み取ったり、その値を変更できる。
- ページ内容の書き出し (読み込み時) — スクリプト内で `document.write(文字列)` を呼び出すことにより、そのスクリプトのある位置に任意の HTML を書き出すことができる。ただしこの機能が使えるのはページ読み込み途中に限られる。
- ページ内容の変更 (読み込み終了後) — ページを読み終わった後でも、後述する DOM または innerHTML インタフェースを使うことで任意のページ内容を書き換えることができる。
- ページ要素のスタイル変更 — 任意の HTML 要素に対して CSS スタイル指定を設定することで色や位置などを変更できる。
- 一定時間後/一定時間間隔での動作実行 — 一定時間後に動作を起動したり、一定間隔で動作を行いアニメーションのような効果を持たせることができる。

前回はこれらのうち、ボタンへの応答と GUI 部品の値の参照/設定を利用していました。今回は前回やった以外の機能について簡単に見てみることにします。例題は (またしても) 摂氏華氏変換です。

前回やった例は、「変換」ボタンを押した時はじめて計算が行われました。でも、手元で計算しているのですから、ユーザが 1 文字変更するごとにそれに対応する結果を表示してもいいはずです。ついでに、摂氏→華氏、華氏→摂氏のどちらなのかも同じ入力欄で指定させられますね (図 12)。

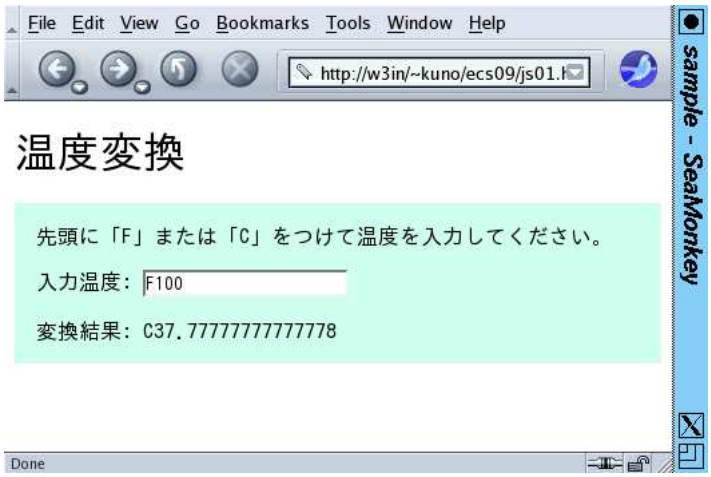


図 12: 摂氏華氏変換 (入力即時変換)

これを行うコードは次のようになっています:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head>
<title>sample</title>
<style type="text/css">
div { background: rgb(200,255,235); padding: 1em }
</style>
<script type="text/javascript">
function chg(elt) {
  var c = elt.value.charAt(0);
  var i = parseFloat(elt.value.substring(1));
  if(c == 'F' || c == 'f') {
    document.getElementById('s0').innerHTML = 'C' + ((5/9)*(i - 32));
  } else {
    document.getElementById('s0').innerHTML = 'F' + ((9/5)*i + 32);
  }
}
</script>
</head><body>
<h1>温度変換</h1>
<div>先頭に「F」または「C」をつけて温度を入力してください。<br><br>
入力温度: <input type="text" name="it" value="F" onkeyup="chg(this)"><br><br>
変換結果: <span id="s0">???</span>
</div></body></html>
```

まず、前回までと違って form 要素を使っていません。そのかわり、入力欄 (input type="text") に onkeyup ハンドラが指定してあり、そこで this をパラメタとして関数 chg() を呼んでいます。実は this はハンドラを持っている HTML 要素 (input) のオブジェクトを参照していて、このため関数 chg() の中で渡された引数の属性.value を参照することで入力文字列が取れます。onkeyup だと打鍵 1 回ごとにハンドラが呼ばれるので、その中で先頭の文字が F か C かに基づいて適切な変換を行い、最後に div 要素の内側にその結果を書き込みます。div 要素を参照するのには、id を指定して document.getElementById() を呼びます。中身を書き換えるのは、.innerHTML プロパティに書き込めばできます。

このように、HTML 上で表されている各要素は JavaScript 側からはオブジェクトとして取り扱え、そのプロパティやメソッドを用いてさまざまな変更を直接施すことができるわけです。これをどのブラウザでも同じように行わせるため、HTML や XML などの文書 (ドキュメント) をどのようなオブジェクトの構造に対応させるかの標準があり、DOM(Document Object Mdel) と呼ばれています。

もう 1 つの例として、今度は「スライダを動かすことでそれに対応する摂氏と華氏の温度を表示させる」ようなインタフェースを作ってみました (図 13):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head>
<title>sample</title>
<style type="text/css">
div { background: rgb(200,255,235); padding: 1em }
#d0 { background: rgb(10,200,190); position: absolute;
      top: 200px; left: 20px; width: 400px; height: 10px }
#s0 { color: red; position: absolute; top: 5x; left: 195px }
</style>
```

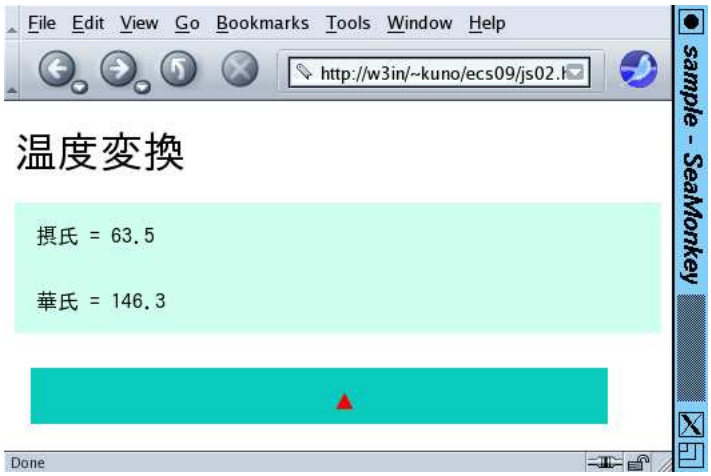


図 13: スライダを使った温度変換

```
<script type="text/javascript">
function drag(ev) {
  var x = ev.clientX || ev.pageX;
  x = x - 25; if(x < 0) x = 0; if(x > 400) x = 400;
  document.getElementById('s0').style.left = x + 'px';
  var c = 0.5*x - 50, f = (9/5)*c + 32;
  document.getElementById('d1').innerHTML = '摂氏 = ' + c;
  document.getElementById('d2').innerHTML = '華氏 = ' + f;
}
</script>
</head><body>
<h1>温度変換</h1>
<div id="d0" onmousemove="drag(event)"><span id="s0">▲</span></div>
<div id="d1">?</div><div id="d2">?</div></body></html>
```

コードの説明ですが、スライダ全体とその中の「つまみ」をそれぞれ div 要素、span 要素として用意し、CSS で位置を絶対配置 (position: absolute) に指定します。こうしておくと、left、top、width、hight などを設定することで要素の位置と大きさを自由に制御できます。そして、HTML 側ではスライダ部分に onmousemove を指定して、そこでイベントオブジェクトをパラメタとして関数 drag() を呼び出します。この中では、イベントオブジェクトのプロパティ.clientX または.pageX(ブラウザの種類で違います) を参照することでマウスの X 座標を取り出し、それに基づいて「つまみ」の位置を変更するとともに、その位置に対応した摂氏と華氏の温度を表示します。位置の変更などは、HTML 要素オブジェクトの.style プロパティの子プロパティを設定することで、CSS で設定するのと同様の設定が自由に行えることを利用しています。

このように、JavaScript ではイベントハンドラと位置の制御などを行うことで、さまざまな柔軟なインタフェースを作れるようになっているわけです。

4.3 Ajax OPTION

Ajax(Asynchronous JavaScript And XML) とは、2006 年くらいから注目を集めるようになった技術で、上に説明したような JavaScirpt を用いたページ変更を、「ページ表示でない」サーバとの通信と組み合わせることで「1 つのページを表示したまま次々に操作をしていける」ような Web アプリケーションを作り出す手法全般を言います。

このためには、既に見て来たフォーム送信などとは違う方法でサーバと通信する必要があります。具体的には、XMLHttpRequest() オブジェクト (IE の 6.0 まででは多少名前が違う) を生成し、このオブジェクトの機能を使ってサーバ側と「こっそり」(ユーザに見えない部分で) 通信を行います。そして、通信の結果を画面に表示するのは DOM あるいは innerHTML プロパティを使用することで、ページ遷移なしに結果を表示できます。

図 14: Ajax による入札アプリケーション

ここでは簡単な例として「入札」を行う Ajax アプリケーションを作って見ましょう (図 14)。入札データはサーバ側で管理するので、PHP でサーバ側プログラムを書きます。最高入札者と入札額は「bid.txt」という名前のファイルに次のような形で入れておきます (このファイルは先の PHP の例題と同様、誰にも読み書きできる保護モードにしておきます)。

```
久野
100
```

次に、このファイルを読み書きして、より高額の入札者があったらデータを更新する PHP プログラムを示します。このプログラムは同じ場所に「bid.php」という名前で入れておくことにします。

```
<?php
  $fdata = file("bid.txt");
  if(isset($_GET['user']) && $fdata[1] < $_GET['val']) {
    $fd = fopen("bid.txt", "w");
    fwrite($fd, $_GET['user']."\n".$_GET['val']."\n");
    fclose($fd);
    $fdata = file("bid.txt");
  }
  echo "user=".$fdata[0]." value=".$fdata[1];
?>
```

このプログラムは get メソッドで user、val という 2 つの値を受け取り、また bid.txt の内容も読み、val の値が現在より大きいなら bid.txt の内容を更新します。データがある場合もない場合もユーザと値を文字列として返します。

では、ユーザが使うページを見てみましょう。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head>
```

```
<title>sample</title>
<style type="text/css">
div { margin: 1ex; padding: 2ex; background: rgb(200,255,240) }
</style>
<script type="text/javascript">
var r = new XMLHttpRequest();
function recv() {
  if(r.readyState == 4) {
    document.getElementById('h0').innerHTML = r.responseText;
  }
}
function update(uri) {
  r.onreadystatechange = recv; r.open('GET', uri, true); r.send(null);
}
function send() {
  var u = document.getElementById('t0').value;
  var v = document.getElementById('t1').value;
  update('bid.php?user=' + u + '&val=' + v);
}
function watch() {
  update('bid.php');
}
</script>
</head><body onload="setInterval(watch, 1000)">
<h1 id="h0">入札:</h1>
<div>
名前: <input type="text" id="t0"><br><br>
値段: <input type="text" id="t1"><br><br>
<button onclick="send()">入札</button></div>
</body></html>
```

まず変数 r に XMLHttpRequest オブジェクトを用意し、以下の通信はこれを使用して行います。定義している関数の機能は次の通りです。

- recv() — XMLHttpRequest から呼び出され、返信が来たときにその文字列を見出し (h1 要素) の中に書き込む。
- update() — 指定した URI を get で取り寄せる。結果表示には recv() を使用するように設定する。
- send() — 入札ボタンが押された時用で、2 つの入力欄から入札者と値段を取り出し、これらのデータを渡す形で update() を呼び出す。
- watch() — 一定時間ごとに表示を更新するために使うもので、データなしで update() を呼び出す。

body 要素の onload で、1 秒間隔で watch() が呼ばれるように設定しています。また、入札ボタンが押されたときは send() が呼ばれます。この HTML ファイルを複数人で表示することで「入札」が行えるわけです。

5　まとめと演習

スクリプト言語とは「簡単にプログラムが書ける言語」という意味あいで、最初はシェルスクリプトなど簡便な言語が使われていましたが、次第に各種用途に使える高機能なものが普及してきました。今回はテキスト処理に多く使われる Perl、Web のサーバ側プログラミングに使われる PHP、Web のクライアント側プログラミングに使われる JavaScript の 3 つを取り上げ見てみました。また、JavaScript とサーバ側プログラムを組み合わせた Ajax と呼ばれる方式も少しだけ見てみました。

3-1. 「latex の表を生成する」Perl プログラムをコピーしてきて動かしなさい。データはコピーしたままでなく適宜変更すること。もしできたら、平均のかわりに別の値を表示するようにしてみなさい (たとえば「1970 年と 1990 年の間の増減」とか)。

3-2. 「latex の表と PostScript の図を生成する」Perl プログラムをコピーしてきて動かしなさい。データはコピーしたままでなく適宜変更すること。もしできたら、各グラフにおいて平均値のところに横線を引くように手直ししてみなさい。

3-3. 「10 未満の数を表示する」PHP プログラムを打ち込んで自分のサイトに設置し、動作を確認しなさい。動いたら次のような手直しのどれかを行ってみなさい。

(a) 10000 未満の数を表示するようにする。
(b) 10000 未満の数を (縦に並べると長いので) ずらっと並べて表示するようにする。「番号」とか「です。」とかは不要。
(c) 掛け算の九九の表を表示するようにする。

3-4. 「等差数列を表示する」PHP プログラムを打ち込んで自分のサイトに設置し、動作を確認しなさい。動いたら次のような手直しのどれかを行ってみなさい。

(a) 初項、公比、上限を指定して等比数列を表示する。
(b) 借り入れ金額、毎年返済額、金利を指定して元利均等返済ローンの毎年の返済状況を表示する。(100 年返済しても終わらなければ打ち切る。)
(c) その他自分で面白いと思う計算の表。

3-5. JavaScript で「ある要素の上にマウスが来るとその要素の色が変わる」Web ページを作成してみなさい。既に作成した自分のページのどこかにそのような仕掛けを入れるのが簡単でよい。

3-6. JavaScript と PHP のいずれか一方、または両方を取り入れた「何らかの動作を行う」Web ページを作りなさい。どのような動作を行うのか、どのような意図でそのように設計したかを説明すること。作ってみて分かったことを報告しなさい。