CGI における Perl のデータベースインターフェースの利用

片 山 敏 之

目 次

1．はじめに

WWW 上での情報ファイルは，HTML というテキスト記述言語に従って，文章や画像を組合わせていけば比較的簡単に作ることができる。CGI は，WWW の情報ファイルに情報の送信側と受信側との間で情報をやり取りする相互作用的な情報処理を行なうための仕組みである。

CGI のプログラムを，それ自体は WWW サーバから見ると外部プログラムであるが，HTML の FORM 機能と組合わせることにより，アンケートのデータ収集と結果表示，WWW 上のチャットや会議室などの実時間処理を行なったりすることができる。また，CGI のプログラムでは，データベースサーバ等の外部プログラムとの連携[1,2]や Java プログラムとのインターフェースを提供[3-5]し，それらの利用を媒介支援するなど，幅広い目的に利用分野が広がってきている。

UNIX のシステム管理で広く使われている簡易データベースシステムである DBM には，従来からの dbm データベースの他にいくつかの新しい実装がある。CGI のプログラム記述言語としては従来から Perl がよく使われている。Perl は，テキストデータを柔軟に処理できる，パターンマッチング技術により大量データを高速探索できる，インタプリタのような開発環境を提供している，モジュールと呼ばれる機能拡張ライブラリが豊富にある[6,7]，そして最新情報がインターネットで公開されている[8]，などの優れた特徴をもっている。

しかし，これらの最新情報はいづれも開発者向けのものであり，一般のプログラミング学習者は，ネットワーク世界の専門用語の多用に敷居の高さを感じてしまう等の理由で，市販の書籍や雑誌に頼ることになる。これらの書籍や雑誌は一般向けの内容に限られていることが多く[9-11]，CGI のプログラムにおけるデータベースサーバ等の外部プログラムとの連携などの知識は各種の参考文献をあたって自分で獲得するしか方法がないようである。

Perl（Perl 5）では標準ライブラリモジュールとして，AnyDBM_File，ODBM_File，NDBM_File，SDBM_File，GDBM_File，および DB_File のモジュール（パッケージ）を利用することができる[6]。これらを利用するには tie，untie，tied 関数（Perl 5 で導入された）または 従来からある dbmopen，dbmclose 関数を使う。tie 関数では，データベースのデータを Perl のハッシュまたは連想配列と呼ばれる特別なデータ構造の形式で扱う。

本稿では，CGI における Perl のデータベー

表1：HTMLにおける色名と16進コードの対応表

色名(キー)	16進コード(値)	和名(値)	別名(値)
aqua	#00FFFF	水色	cyan
magenta	#FF00FF	赤紫	fuchsia
yellow	#FFFF00	黄	red
#FF0000	赤	green	#00FF00
緑	lime	blue	#0000FF
青			

スインターフェースの利用に必要な事項を，これらの関連する知識も最小限に含めて，理解しやすく参照に便利なようにまとめて記述する。これらの説明では，HTML 言語による HTML ファイルの記述や Perl 言語による CGI プログラミングの基礎知識を前提にしている。実際のプログラミングにおいては，当然ながら，それらに関する適当な手引き書[6,7,9-12]を参照できることを前提に記述している。

2．ハッシュ変数の操作

2．1 ハッシュ変数

ハッシュ（hash）変数は，キー（key）と呼ばれる文字列を添字（index）とする配列（array）変数と考えることができる。通常の配列は0からはじまる整数の番号を添字としている。

ハッシュおよびキーという名前は，Perl 処理系でこれを実現するために使われているハッシュ表と呼ばれる特有なデータ構造に由来している[13]。ハッシュ変数ではキーと値のリストの順番は Perl 処理系で自動制御される。

例えば，表1の対応表を扱おうとするとき，ハッシュ変数に格納すれば色名と16進コードなどを関連付けてデータベースとして利用することが可能となる。このような意味で，ハッシュ変数は連想配列（associative array）とも呼ばれる[14]。

ハッシュ変数の名前は「%」で始まる英数字列で定義する。%の直後は英文字でなければならない。文字列に下線「_」を含んでいてもよい。キーは文字列または文字列を値に持つスカラ変数である。

2．2 ハッシュ変数に値を代入する

ハッシュ変数に値を代入するには，値のリストを使って

```
%HASH = (key1,val1,key2,val2, ... );
```

または，キーと値を特別な記号「=>」で区切って

```
%HASH = (key1=>val1, key2=> val2, ... );
```

のように記述する。

ハッシュ変数を初期化するには，空のリストを代入する：

```
%HASH = ();
```

正確には，空のリストの代入のみでは，ハッシュ変数へのメモリ領域は割り当てられたままである。ハッシュ変数を実現する内部表現を削除する（メモリの解放）するには，次のように undef 関数を利用する：

```
undef %HASH;
```

これでハッシュ変数全体を削除（未定義に[13]）する。

ハッシュ変数の要素をまるごと配列変数に代入することもできる：

```
@ARRATY = %HASH;
```

ただし，代入された配列 @ARRAY の要素の順番は，Perl 処理系がハッシュ変数 %HASH を自動制御した後の順番に変更されているので，見かけ上ランダムなはずである。

ハッシュ変数の1つのキーと値の組みを追

加したり，変更したりすることもできる。ハッシュ変数のキーは添字 $key を「{」と「}」で括弧して指定する。値はスカラなので「%」の代わりにスカラ変数を表す「$」を付けて $HASH{$key} のように表現する。表1の色名とコードの対応表について例を示す：

```
%TBL_color = ();
%TBL_color=('red'=>'#FF0000',
'green'=>'#00FF00',
'blue'=>'#0000FF');
$TBL_color{'aqua'}='#00FF00';
$TBL_color{'aqua'}=
'#00FFFF';
```

この例では，最初，'red'，'green'，'blue' というキーと対応する値をリストで代入し，次に 'aqua' というキーと値を追加した。ハッシュ変数に新しいキーと値を追加するには未だ定義されていないキーを指定するだけでよい。最後の行では 'aqua' というキーの値を変更している。

上の例からも分かるように，ハッシュ変数の値を参照するには，やはり $HASH{$key} の表現を使えばよい。次の例では，ハッシュ変数の値を参照して2つのスカラ変数にその値を代入している。$val1 の値は '#0000FF' となるが，$val2 の値には，'magenta' というキーは %TBL_color にはまだ無いので，未定義値が代入される。

```
$val1 = $TBL_color{'blue'};
$val2=$TBL_color{'magenta'};
```

ハッシュ変数の特定のキーの値を未定義にするには，undef 関数を使う：

```
$TBL_color{'magenta'}=
'#FF00FF';
undef $TBL_color{'magenta'};
```

ハッシュ変数の特定のキーの値を実際に削除するには，delete 関数を使う：

```
delete $TBL_color{'magenta'};
```

2．3　ハッシュ変数の操作関数

ハッシュ変数のキーまたは値をリストとして取り出すために，次の関数が標準で提供されている[12, 13]。

keys 関数はハッシュ変数のすべてのキーをリストとして返す。次の例は，ハッシュ変数 $TBL_color の要素をすべて取り出して，簡単な表として印刷している。

```
例：  print "key :¥t value ¥n";
      #項目の印刷，"¥t"はタブの挿入
      foreach $i (keys %TBL_color) {
                print"$i:¥t",$TBL_
color{$i},"¥n";
      }
```

ハッシュ変数はそのデータ構造の特性として，要素（キーと値の組）が代入のときの順序とは無関係に格納されている。従って，上の例では要素が見かけ上ランダムに印刷される。keys，values，each 関数の戻り値について，リストの並びの順序は同じである。

keys 関数の戻り値はリスト（配列）なので，keys の返す値を sort 関数で並べ替えて利用することができる。次の例では，ハッシュ変数 $TBL_color の要素をすべて取り出して，キーの昇順からなる表として印刷してい

表2：ハッシュ変数の操作関数

関数名と書式	機能
keys %HASH	全てのキーのリスト(配列)を返す
values %HASH	全ての値のリスト(配列)を返す
each %HASH	ハッシュから次の要素(キーと値の組)を取り出して，リストとして返す。
scalor(keys %HASH)	ハッシュ変数のキーの個数を返す
undef %HASH;	ハッシュ変数全体を削除する
delete $HASH{$key}	指定したキーと値を削除する

る。

例： print "key :¥t value ¥n";
foreach $i (sort keys %TBL_color) {
$key = $i;
print "$key :¥t", $TBL_color{$key},"¥n";
}

スカラ・コンテキストでは，keys 関数はハッシュ変数の要素の個数を返す。次の例では，スカラ変数 $num1，$num2 共にハッシュ変数 %TBL_color の要素（キーと値の組で 1 つ）の個数が代入される。scalar 関数は，リスト・コンテキストの返り値をもつ式 keys %TBL_color を，強制的にスカラ・コンテキストで評価したい場合に利用する。

例： $num1 = keys %TBL_color;
$num2 = scalar (keys %TBL_color);

values 関数は，keys 関数とは逆に，ハッシュ変数のすべての値のリストを返す。values 関数の戻り値は見かけ上ランダムな順番になるが，これは同じハッシュ変数に対して keys 関数が返すキーの順番と対応していることが保障されている。values 関数の戻り値を整列したい場合には sort 関数を合わせて用いる必要がある。

サブルーチン名を sort 関数の式として与えることで，任意の順番でリストを整列することができる。

次の例では，ハッシュ変数 $TBL_color の要素をすべて取り出して，値の昇順からなる表として印刷している[15]。

例： print "value :¥t key ¥n";
foreach $key (sort valSort keys %TBL_color) {
print $TBL _ color{$key}, ":¥t", $key,"¥n";
}
sub valSort {
$TBL _ color{$a} <=> $TBL_color{$b};
} #値を数値として比較

each 関数は，それが評価される度にハッシュから次の要素（キーと値の組）を取り出して，リストとして返す。each 関数を連続して呼び出せば，ハッシュのすべての要素を取り出すことができる。each 関数の戻り値もランダムであり，その戻り値の並びは keys，values 関数と同じである。each 関数によって繰り返し処理を行なっている最中には，そのハッシュ変数に対して，別の each 関数をつかったり，要素を追加してはならない。しかし，delete 関数を使っても構わない。

次の例は，ハッシュ変数 $TBL_color の要素を順次取り出していき，すべての要素を簡単な表として印刷している。

例： print "key :¥t value ¥n";
foreach (($key,$value) = each %TBL_color) {
print "$key :¥t $value¥n";
}

または，each 関数ですべての要素を読み終わったら空リストが返されるので，while 構文を使って次のようにも記述できる：

例： print "key :¥t value ¥n";
while (($key,$value) = each %TBL_color) {
print "$key :¥t $value¥n";
}

3．タイ変数の利用

Perl 5 では，tie，untie，tied 関数が新たに導入されている。これらは通常の変数（スカラ，配列，ハッシュ）を用いて外部モジュールなどを利用して管理するための関数となっている。利用する変数の種類によって対応するモジュールの仕様，つまり実装クラスとの

結び付きの仕組みが異なるので，それらの部分のインタフェースの記述を提供してくれている。Perl 5はオブジェクト指向プログラミング対応を取入れているので[16]，ここでは関連する専門用語を適宜に使用して解説する。

3．1　tie関数

tieとuntieは，変数を外部モジュールを利用して管理するための関数である。

tie関数の書式：

tie　VARIABLE,　CLASSNAME, LIST

tie関数は，変数VARIABLEを適切なメソッド（サブルーチン）が実装されているクラス（パッケージ）に結び付ける。VARIABLEは変数の名前，CLASSNAMEはクラスの名前，LISTはクラスのコンストラクタ（メソッド）に引き渡される引数である。

このようにして結び付られた変数をタイ変数（tied variable）という。

コンストラクタの名前は，通常使われる"new"ではなく（new CLASSNAME, LIST），tied変数のデータ型に応じて，TIESCALAR, TIEARRAY, TIEHASH, のいずれかとなる。

例：　TIEHASH CLASSNAME, LIST

このメソッドは，このクラスのコンストラクタである。

tie関数は，コンストラクタ（メソッド）によって返されたオブジェクト（へのリファレンス）を値として返す。リファレンスとは[17]，通常の変数や関数の格納場所そのものを値として取り出したもので，C言語のポインタに相当する。Perlではリファレンスをスカラ変数に記録することができる。

$object = tie VARIABLE, CLASSNAME, LIST;

tie関数は，クラスから変数を扱うためのオブジェクトを内部的（Perl処理系で）に生成することによって，この結びつきの関係を確立する[16]。指定したクラスや適切なオブジェクトが無い場合はエラーとなり，$objectは未定義値となる。

untie関数は，tie関数によって結び付られた関係を解除するものである。

既にtie関数によって返されたオブジェクトへのリファレンスは，tied関数によっても得ることができるので，後からでもアクセスが可能となる。tied関数は変数に結び付られているオブジェクトへのリファレンスを返す。変数VARIABLEがパッケージに結び付られていなければ，tiedは未定義値を返す。

tied関数の書式：

tied VARIABLE

例　：　$object＝VARIABLE, CLASSNAME, LIST;

untie VARIABLE;

$object = tied VARIABLE;

tie関数は，典型的にはハッシュ変数に対して使われる。ハッシュ変数をUNIXシステムのDBMファイルを実装したクラスにtie関数で結び付ると，プログラムでは，そのハッシュ変数をアクセスすることで，物理的なディスク上に記録されたデータベースのファイルを読み書きすることができる。

3．2　dbmopen関数

dbmopen関数の書式：

dbmopen　%HASH,　DBMfile, MODE

dbmclose %HASH

dbmopen関数はDBMファイルをハッシュ変数に結合する。DBMファイルについては次の4章で述べる。

dbmopen関数とdbmclose関数はPerl 4で使われていたもので，Perl 5では下位互換性のためにサポートされている。機能的な制約があったり，使用するに際してはいくつかの注意が必要となる[18]。

4．データベースライブラリの操作

UNIX システムの OS には，DBM と総称される比較的小さなデータベースを管理するライブラリが標準で実装されて，システム管理での情報処理に広く使われている。この DBM は，Perl 処理系のハッシュのデータ構造と同じように，キーとキーに対応する値の組を要素として取り扱う。DBM は，その要素の全体をハッシュのように内部メモリ領域に割り当てるのではなく，それらを外部メモリ領域（ディスク）のファイルとして管理し，記録する。

DBM で管理されるファイルは，UNIX のファイルシステムで管理される通常のテキストファイルなどよりも格段に高速に処理される特性をもつ。従って，システム管理のみならず，UNIX システムで稼動する応用プログラムでもよく利用されている。

Perl（Perl 5）では，この DBM データベースシステムにアクセスするためのライブラリモジュールとして，標準で[6]，AnyDBM_File, ODBM_File, NDBM_File, SDBM_File, GDBM_File, および DB_File の6つのモジュール（パッケージ）を利用することができる。

4．1 モジュールの利用宣言

モジュール名 Module を利用するには，あらかじめ use 宣言をする必要がある。

```
use Module LIST
または
use Module
```

use 宣言は，指定したモジュールの内容を，サブルーチンや変数名を現在のパッケージ内にエイリアスすることによって，現在のパッケージに輸入する。つまり，内部的には次のようなサブルーチン BEGIN の実行コードと同等である：

```
BEGIN {
    require Module;
    import Module LIST;
}
```

4．2 DBM モジュールの利用

AnyDBM_File パッケージは，他の様々な DBM パッケージから継承を行なうためにだけの「純粋な仮想ベースクラス」である[19]。Perl 5 以前との下位互換性を保つために，デフォルトでは，NDBM_File, DB_File, GDBM_File, SDBM_File, ODBM_File, の順にどれでも使えるものを継承しようとする。クラス間の継承関係は，Perl 内部で利用される変数 @ISA に定義されている。変数 @ISA は Perl にクラスの親子関係（ISA 関係）を指定する。

例：表1の HTML における色名と16進コードの対応表を，DBM データベースとして管理する例を以下に説明する。

```
use Fcntl;
use AnyDBM_File;
$newfile = "./sample/dbColor";
tie (%TBL_color, AnyDBM_File, $newfile, O_RDWR | O_CREAT,
        0644 ) or die "Error: tie $newfile";
%TBL_color = ();
%TBL_color = ('red'=>'#FF0000', 'green'=>'#00FF00',
              'blue'=>'#0000FF', 'aqua'=>'#00FFFF',
            'magenta'=>'#FF00FF', 'yellow'=>'#FFFF00');
untie %TBL_color;
```

表3：ファイルオープンのフラグの内容

フラグ	内　　容
O_APPEBND	ファイルを追加モードでopen
O_CREAT	ファイルが存在しない場合は新規に作成
O_RDWR	読み/ 書きモードでopen
O_RDONLY	読み専用モードでopen
O_WRONLY	書き専用モードでopen
O_EXCL	O_CREATが指定されていて，ファイルが存在する場合にエラーを返す
O_TRUNC	ファイルを切り詰めてopen

ここで，Fcntlモジュールの利用宣言は，C言語のFcntl.hの定数定義をロードするためのもので，この中にファイルを開くopenシステムコールの引数を提供するO_RDWR, O_CREATなどが含まれる。次にtie関数はハッシュ変数%TBL_colorをAnyDBM Fileモジュールを利用して外部ファイル$newfileに結合している。$newfile以降はこのtie命令のLIST部分である。"O RDWR | O_CREAT"はFcntlモジュールのファイルオープンフラグを指定するもので，既存のファイルを読み書きモードで開くか（O_RDWR）または（|）指定したファイルが存在しなければ新規に作成（O CREAT）する。最後の"0644"は，ファイルの許可モードを指定するもので，"0644"は所有者は読み書き可およびグループと他人は読み出しのみ可を意味し（-rw-r--r--），UNIXのファイルシステムと同じ規則の8進数表現である。

これによってDBMデータベースとして管理されるDBMファイルが作成される。DBMファイルは，上の例では，このプログラムの置かれたディレクトリに"sample"というディレクトリがありその中に"dbColor"というファイルが変数$newfileに指定されているが，実際には，dbColor.dirとdbColor.pagという2つのファイルから構成されている。

Fcntlモジュールで定義されているファイルオープンのフラグを表3にまとめておく。

4．3　DBMモジュールの比較

DBMに関連するライブラリモジュールには，ODBM_File, NDBM_File, SDBM_File, GDBM_File, およびDB_Fileがある。ODBM_FileはUNIXで古くから標準的に使われているdbmデータベースにアクセスする。SDBM_Fileはソースプログラムが Perlにバンドルされているsdbmライブラリを利用できる。その他，NDBM_File, GDBM_File, DB_Fileは，いづれも最近になって配布されるようになったもので，それぞれ，UNIX標準のndbm，GNU版のdbmライブラリgdbm，Berkeley版UNIXのBerkeley DB，の各データベースにアクセスする。

どのDBMを利用するか決める場合には，プログラムの大きさ，ディスクの使用量，実行速度などの機能と共にブロックサイズの制限も重要となる。DBMは比較的小さなデータベースを管理するが，データベースの1レコード長の上限は，ODBM_Fileが1Kバイトで一番小さく，SDBM_FileとNDBM Fileの場合は4Kバイトで，GDBM_FileとDB_Fileは無制限となっている。詳しい機能比較については文献6の7．2節を参照されたい。

ハッシュ変数に格納する1レコード長とは，1組みのキーと値のデータのサイズであるから，通常の利用目的には充分の大きさといえる。例えば，UNIXではメールアドレスの別名定義ファイル（/etc/aliases）や，NISサーバのデータベースファイル（/etc/passwd, /

例：

```
use Fcntl;
use ODBM_File;
local %PASSWORD = ();
local ($file, $key, $value, $i);
$file = "/etc/aliases";
tie %PASSWORD, ODBM_File, $file, O_RDONLY, 0644;
while (($key, $value) = each (%PASSWORD)) {
    $i++;
    print "$i: key= $key,  value= $value ¥n;
}
untie %PASSWORD;
```

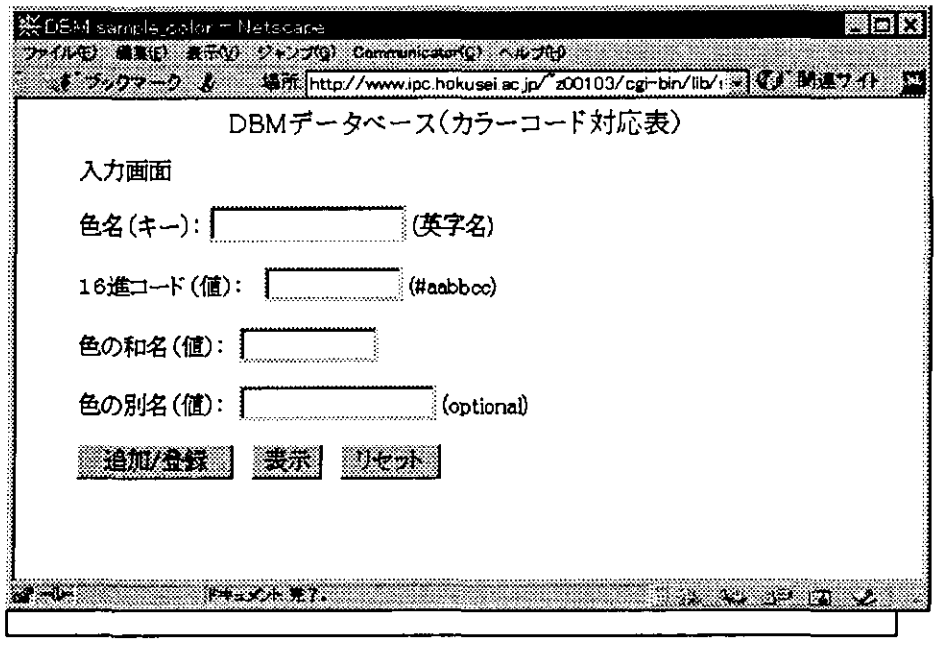
etc/group など）が ODBM 形式で管理されている。

次の例では，/etc/aliases から内容（キーと値の組の集合）をすべて読み出している。

5．CGI におけるデータベースの利用

ここでは，２章の表１の対応表をDBMデータベースに登録して，表示するだけの簡単なデータベース応用プログラムのサンプルを示す。カラーコード対応表のデータ入力画面は下の図１のようになっている。

図１：データ入力画面



5．1　カラーコード対応表のデータ入力画面

ここでは図1のHTMLファイルを示す。

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=Shift_JIS">
   <title>DBM sample_color</title>
<SCRIPT Language="JavaScript">
<!--
function btnMenu(btn)
{
   document.Form.Action.value = btn;
   document.Form.submit();
}
//-->
</SCRIPT>
</head>
<body bgcolor="white">
<center><font size=+1>DBMデータベース（カラーコード対応表）</font></center>
<UL>
<p>入力画面
<FORM METHOD=POST ACTION="sample_color.pl" NAME="Form">
<P>色名（キー）:  <INPUT TYPE="text" NAME="Key" SIZE="15">
(英字名)
<p>16進コード（値）:   
<INPUT TYPE="text" NAME="Value16" SIZE="10"> (#aabbcc)
<p>色の和名（値）:  
<INPUT TYPE="text" NAME="ValueJp" SIZE="10">
<p>色の別名（値）:  
<INPUT TYPE="text" NAME="ValueAl" SIZE="15"> (optional)
<p>
<INPUT TYPE="button" NAME="Append" VALUE="追加/登録"
onClick=btnMenu('Append')>
 
<INPUT TYPE="button" NAME="View" VALUE="表示" onClick=btnMenu('View')>
 
<INPUT TYPE="reset" VALUE="リセット">
</FORM>
</UL>
</body></html>
```

5．2 DBM データベースの登録と表示プログラム

ここでは，前節の HTML ファイルにおける指示によって起動する Perl による CGI プログラムの本体を示す。このプログラムはデータベースのレコード登録と表示に係わる処理を記述している。このプログラムの構成については前章までの準備と文献[20]12を参考にしている。処理結果を HTML のデータとして出力するサブルーチンは次節のパッケージで(cgiutil.pl) 記述されている。

```
!/usr/bin/perl
#
# DBM sample_color.pl (C) 2000 T. Katayama
#
require cgiutil.pl;
use Fcntl;
use AnyDBM_File;
#
$Title = 'DBM sample_color';
$Location = './sample';
$Dbfile = $Location . '/dbColor';
$Delim = ',';
$Encoding = 'sjis';
$CharSet = 'shift_JIS';
@Fields = ('Key','Value16','ValueJp','ValueAl');
# Input Data
cgi::ParseInput();
# Which Botton ?
if ($name{'Action'} eq 'Append') {
   appendData();
} else {
   viewData();
}
exit(0);
#
sub viewData
{
   tie (%TBL_color, AnyDBM_File, $Dbfile, O_RDONLY, undef) or
   cgi::exitError("Error view! DBM $Dbfile がopenできません.");

   cgi::HTTPHeader();
   cgi::SimpleTitle($Title, $CharSet);
   print "<TABLE>¥n";
```

```
    print "<TR>¥n";
    for ($i=0, $i < scalar(@Fields); $i++) {
      print "<TH>$Fields{$i}</TH>¥n";
    }
    print "</TR>¥n";
    local @data = ();
    foreach (sort values %TBL_color)
    {
      print "<TR>¥n";
      @data = split(/$Delim/);
      for ($i = 0, $i < scalor(@Fields); $i++) {
        print '<TD BGCOLOR="#FAEBD7">', $data{$i},"</TD>¥n";
      }  # bgcolor=antique white
      print "</TR>¥n";
    }
    print "</TABLE>¥n";
    cgi::SimpleEnd();
    untie %TBL_color;
}
#
sub appendData
{
    local ($key, $record);
    tie (%TBL_color, AnyDBM_File, $Dbfile, O_CREAT | O_RDWR, 0600) or
    cgi::exitError("Error append! DBM $Dbfile が open できません.");
    $key = $name{'Key'};
    if (%TBL_color{$key}) {
      cgi::exitError("Error! A value which has the same $key exists.");
    }
    foreach (@Fields) {
      $record = $record . $name{$_} . $Delim;
    }
    chomp($record);
    %TBL_color{$key} = $record;
    untie (%TBL_color);
    cgi::HTMLHeader();
    cgi::SimpleTitle("追加/登録しました。");
    cgi::SimpleEnd();
}
### End of sample_color.pl
```

5．3 FORMデータの入力解析パッケージ

このパッケージはHTMLのFORMタグから受け渡されたデータを文字列解析する通常のParse部分と，CGIによるHTMLファイルの作成に共通なサブルーチン（9，10，12）をまとめている。

```
# file name = cgiutil.pl
# copyright: T.Katayama 1996-2000
# ref: ohm p.37 cgiparse.pl by T.Nishida 1997
#      gihyo p.48 by K.Fujita & S.Mishima 1999

package cgi;
  local(@name,%name);

# ohm p.254, default variable $_ and array @_ of subroutine arguments
# cgi::ParseInput(encoding) => <IN>  encoding: JP code (jis | sjis | euc)
#                              <OUT> pair of Name=>Val into %hash (*glob)
# usage?  $name1 = $cgi::name{'FieldName1'}

sub ParseInput {
   my($encoding) = @_;
   my($method) = $ENV{'REQUEST_METHOD'};
   local($data, @data, $key, $value);

   require 'jcode.pl' if $encoding;     # 日本語コードの変換が必要なら要求

   # CGI input method
   if ($method eq "GET") {
       $data = $ENV{'QUERY_STRING'};
   } elsif ($method eq "POST") {
       read(STDIN, $data, $ENV{'CONTENT_LENGTH'});
   }

   # URL decode rule ( see p.229, p.235 for '$[-1)' )
   $data =~ s/¥r¥n/¥n/g;              # replace newline code
   @data = split(/&/, $data);         # split input data 'na=va&nb=vb&nc=vc'

   foreach $_ (@data) {
       if (index($_, "=") == $[-1) { # if '=' is not found
           push(@name, '', $_);
       } else {                      # split a data 'name=value'
```

```
            ($key, $value) = split(/=/, $_, 2));
            push(@name, $key, $value);
        }
    }
    foreach (@name) {
        s/¥+/ /g;                       # replace spaces in values
#       s/%(..)/pack("c", hex($1))/ge;  # decode: char in %ff (or 16) to 10 rep
        s/%([A-Fa-f0-9][A-Fa-f0-9])/pack("c",hex($1))/ge;

        $data = $_;
#       convert of JP code into $encoding
        if ($encoding) {
            jcode'convert(*data, $encoding);
        }
    }

    %name = @name;                      # return as a hash table
    return *name;
}

# cgi::MultiValue(name) for case of a name is assigned to multiple value
# usage?  @value1 = $cgi::MultiValue{'FieldNmae1'}
# $[ は配列の最初の添字を返す変数，$#は配列の最後の添字を返す演算子(p.234-235)
# $_[0] は引数の最初の要素，複数の値が割り当てられる名前
sub MultiValue {
    local ($i, @value);

    &ParseInput unless defined @name;

    for ($i = $[; $i < $#name; $i += 2) {
        push(@value, $name[$i+1]) if $name[$i] eq $_[0];
    }
    return @value;
}
#
sub HTTPHeader {
    return "Content-type: text/html¥n¥n";
}
#
sub SimpleTitle {
```

```
    my(@a) = @_;
    my($title,$CharSet,@color) = ('','','antiquewhite','indigo');
    $Title = $a[0];
    $CharSet = $a[1]
    $title .= "<HTML>¥n<HEAD>¥n";
    $title  .=  qq / <META  HTTP-EQUIV="Content-Type"  Content="text / html;
Charset=$CharSet">¥n/;
    $title .= qq/<META name="GENERATOR" content="cgiutil.pl">¥n/;
    $title .= "<TITLE>" . &Quote($_) . "</TITLE></HEAD>¥n";
    $title .= "<BODY BGCOLOR=$color[0]>¥n";
    $title .= "<P><HR WIDTH=350 ALIGN=center COLOR=$color[1]><P>¥n";
    $title .= "<UL> ¥n<CENTER> ¥n<TABLE BORDER=0 WIDTH=768> ¥n";
    $title .= "<TR ALIGN=left><TD>¥n¥n";
    return $title;
}
#
sub SimpleEnd {
    my($end,$color,@color) = ('','','antiquewhite','indigo');
    $end .= "¥n</TD></TR>¥n</TABLE>¥n";
    $end .= "</CENTER>¥n</UL>¥n";
    $end .= "<P><HR WIDTH=350 ALIGN=center COLOR=$color[1]><P>¥n";
    $end .= "</BODY>¥n</HTML>¥n";
    return $end;
}
#
sub Quote {
    my $string = "";
    foreach (@_) {
        s/&/&amp;/g;
        s/</&lt;/g;
        s/>/&gt;/g;
        s/"/&quot;/g;
        $string .= $_;
    }
    return $string;
}
#
sub exitError {
    local (@msg) = @_;
    foreach $_ (@msg) {
```

```
        print "<p>$_ ¥n";
    }
    print "<p>¥n";
    die @msg;
}
#
1;
# use or require するために必要
```

6．おわりに

本稿では，Perlにおけるハッシュ変数の操作，およびtie関数によるタイ変数の扱い方から始めて，CGIにおけるPerlのデータベースインターフェースの利用に必要な事項を，最小限にかつ理解しやすく参照に便利なようにまとめて解説してきた。

また，簡単な色名と16進コードの対応表をDBMデータベースとして作成するサンプルのCGIプログラムを例示した。サンプルのプログラムとしては長すぎると思われるが，その作成に至るまでに個々の事項についてはそれぞれ数行の小さい例題を与えてあるので，サンプルの概略と細部ともに理解できることを期待している。

現実的な応用プログラムにおいては，データベースの登録と表示機能だけではなく，データの検索や修正と削除，さらに，データベースのユーザ向け画面と作成者向けの画面を分けて構築するなど，種々の機能の高度化やインターフェースの考察が必要とされる。本稿はそのような構築への役に立つ解説となることを目標とするものである。このような研究ノートを積み重ねることによって，大学における演習（ゼミナール）の教材としても活用できることをひとつの目標として考察を深めていきたい。

〔謝辞〕本稿は，「1999年度北星学園大学特別研究費による研究」である。（第3号個人学術研究）

〔参考文献〕

（1）田中克己，1995，Proceedings of Advanced Database System Symposium'95, pp.1-8, SIGDBS/IPSJ; 吉川正俊, ibid., pp.49-58

（2）関根徹，1996，Proceedings of Advanced Database System Symposium'96, pp.43-48, SIGDBS / IPSJ and SIGMOD(Japan Chapter) / ACM;

（3）John Gage, 1995, Proceedings of Advanced Database System Symposium'95, pp.171-205, SIGDBS / IPSJ;

（4）井田昌之，高木浩光，松岡聡他，1998，情報処理 Vol.39 No.4, pp.282-313

（5）菊田英明，2000，JAVA PRESS Vol.13, pp.2-10, 技術評論社

（6）Larry Wall, Tom Christiansen, Randal L. Schwartz, 1996, Programming Perl (2nd Ed.), O'Reilly and Associates;
近藤嘉雪訳，プログラミングPerl改訂版，1997，オライリー・ジャパン

（7）Randal L. Schwartz, 1993, Learning Perl, O'Reilly and Associates;
近藤嘉雪訳，初めてのPerl，1995，ソフト

バンク
(8) WWWのURL, Perl community home page, http://www.perl.com/ ; ニュースグループ, comp.lang.perl.misc; fj.comp.lang.perl
(9) 西田知博, 川本芳久, 1997, CGI 入門講座, オーム社
(10) 三島俊司, 1988, CGI のための実践入門 Perl, 技術評論社
(11) 齋藤靖, 小山裕司, 前田薫, 布施有人, 1966, 新 Perl の国へようこそ, サイエンス社, 2.5節
(12) 藤田郁, 三島俊司, 1999, CGI & Perl ポケットリファレンス, 技術評論社
(13) 文献11, 7.3節, および文献 6 , 3.2節
(14) 文献 6 , 2.3.5節
(15) 文献 6 , 3.2節, p.208および文献11, 7.2節
(16) 文献 6 , 5.3節および5.4節
(17) 文献 6 , 4.1節, 文献10, 3.14節および文献 11, 2.7節
(18) 文献11, 7.3節, および文献 6 , 7.2節
(19) 文献 6 , 3.2節, p.176, p.435および文献10, 3.15節などを参照のこと
(20) 文献12, pp.22-78

正　誤　表

北星学園大学経済学部北星論集第39号

頁・行目	誤	正
54頁から73頁　見出し	国際債権市場	国際債券市場
68頁　左段　上から5行目	プォルムサー	プォルムサー
68頁　右段　上から5行目	オーバーシュレージェン	オーバーシュレージェン
136頁	表1の内容	下記

表1：HTMLにおける色名と16進コードの対応表

色名（キー）	16進コード（値）	和名（値）	別名（値）
aqua	#00FFFF	水色	cyan
magenta	#FF00FF	赤紫	fuchsia
yellow	#FFFF00	黄	
red	#FF0000	赤	
green	#00FF00	緑	lime
blue	#0000FF	青	