sub decode

目次

はじめに

ここでは SOSII が CGI として動作するための要となる sub decode について解説します。

sub decode では主に

・URL のデコード
・文字コードのコンバート
・HTML タグの無効化
・スプリッターとして使用する文字列のエスケープ

を行っています。

ソースコードの解説

```
# Sub Decode #
sub decode {

  if ($ENV{'REQUEST_METHOD'} eq "POST") {
```

まず環境変数からファイルメソッド POST または GET を評価します。

```
    $post = 1;
```

POST メソッドであった場合、$post という変数に 1 を代入しフラグを立てます。
Ver.1.10 の場合、$post 変数が真でなければ多くのサブルーチンが実行できません。
これは不正行為への対策と思われます。

```
    read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
```

STDIN で入力されたフォームデータをファイルハンドルにより CONTENT_LENGTH の長さ（ようするに全て）読み込み、$buffer に代入します。

```
    @pairs = split(/&/, $buffer);
```

$buffer に代入されたフォームデータを split 関数で､& をスプリッターとして分解し @pairs という配列に格納します。

例としてフォームデータが「id=0000&ps=PASS&nm=NAME」であった場合

・@pairs = (id=0000,ps=PASS,nm=NAME);

となります。

```
  } else { @pairs = split(/&/, $ENV{'QUERY_STRING'}); }
```

GET であった場合、上と同じく QUERY_STRING を @pairs という配列に格納します。
QUERY_STRING はフォームデータ自体が格納されていますので、ファイルハンドルによる操作は不要になります。

```
foreach $pair (@pairs) {
```

foreach 構文によるループ構文で @pairs 配列を展開します。
$pair には @pairs の要素が順に代入されてゆきます。
ループは @pairs の要素の数だけ繰り返して行われます。

```
($name, $value) = split(/=/, $pair);
```

$pair に代入された変数を split 関数で、今度は = をスプリッターとして分解します。
分解されたものは、それぞれ $name、$value という変数に代入されます。

例として $pair が「id=0000」であった場合

- $name = id;
- $value = 0000;

となります。

```
$value =~ tr/+/ /;
```

tr/// 演算子を使いパターンマッチによる置き換えを行っています。
+ という文字を （半角スペース）に変換しています。
URL エンコードのさい､送られてくるデータの半角スペースが + に置き換えられて CGI に渡されるためこのような変換が必要になります。

```
$value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
```

s/// 演算子を使いパターンマッチによる置き換えを行っています。

日本語の文字列は URL エンコードによって %xx という形でフォームに送られます。
これを pack 関数でもとの文字列に置き換えています。

$1 には左側の一番目の括弧（[a-fA-F0-9][a-fA-F0-9]）にマッチした内容が入ります。

オプション e は右側の評価を行います。
上の式の場合、置き換えを行う前に pack("C", hex($1)) を実行し、その評価結果を置き換えの対象としています。

オプション g は繰り返し置き換えを行います。
上の式の場合、%xx という形式の文字列をすべて置き換えるまで実行しています。

```
&jcode'convert(*value,'sjis');
```

$value の変数を日本語コード（Shift-JIS）に変換しています。

・&jcode'convert(* スカラ ," 文字コード ");

というのは jcode.pl というライブラリに入っている命令文です。
スカラを文字コードのタイプへと変換します。

```
$value =~ s/</&lt;/g;
$value =~ s/>/&gt;/g;
$value =~ s/"/&quot;/g;
$value =~ s/\,/, /g;
$value =~ s/△/▲/g;
$value =~ s/\r\n/<br>/g;
$value =~ s/\r/<br>/g;
$value =~ s/\n/<br>/g;
```

$value に代入された一意の文字を置き換えています。
HTML タグや改行コード、システム上スプリッターとして使用している文字等が該当しています。

```
$Fm{$name} = $value;
```

フォームで送信された値を最終的な形（$Fm{'hoge'}）として代入しています。

・$Fm{'id'};

という形は %Fm という連想配列（ハッシュ）の参照です。

・id=0000（<input type=hidden name=id value=0000>）

であるならば

・$Fm{'id'} = 0000;

となります。これは %Fm というハッシュのキー値 id に対してペア値 0000 を代入している式になります。

この部分が SOSII の汎用性の高さを物語っていると、勝手に思っています。

```
  }
}
```

実際の動き

```
<input type=hidden name=nm value=なまえ>
<input type=hidden name=ps value=ぱす>
```

上記のようなフォームであった場合、これをスクリプトへ送信すると、クエリーは「nm= なまえ

&ps= ぱす」となります。
この時、URL エンコードが行われ、実際の文字列は「nm=%82%C8%82%DC%82%A6&ps=%82%CF%82%B7」と変換されます。
この文字列がファイルハンドルにより $buffer に代入されます。

```
$buffer = 'nm=%82%C8%82%DC%82%A6&ps=%82%CF%82%B7';
```

$buffer は分割され @pairs に格納されます。

```
@pairs = (nm=%82%C8%82%DC%82%A6,ps=%82%CF%82%B7);
```

ループにより @pairs の要素を順に評価していきます。
$pair へ代入された要素をさらに分割し、$name と $value に代入されます。

```
$name = 'nm';
$value = '%82%C8%82%DC%82%A6';
```

$value の置き換えを行います。

```
$value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
```

に当てはめた場合、g オプションによって

```
$value =~ s/%82/pack("C", hex(82))/e;
$value =~ s/%C8/pack("C", hex(C8))/e;
$value =~ s/%82/pack("C", hex(82))/e;
$value =~ s/%DC/pack("C", hex(DC))/e;
$value =~ s/%82/pack("C", hex(82))/e;
$value =~ s/%A6/pack("C", hex(A6))/e;
```

このようになるはずです。

しかし、実際には

```
$value =~ s/%82%C8/pack("C2", hex(82), hex(C8))/e; # pack("C2", hex(82), hex(C8)) = な
$value =~ s/%82%DC/pack("C2", hex(82), hex(DC))/e; # pack("C2", hex(82), hex(DC)) = ま
$value =~ s/%82%A6/pack("C2", hex(82), hex(A6))/e; # pack("C2", hex(82), hex(A6)) = え
```

と同じ動作をしているようです。

e オプションで右側式の評価結果に置き換えられますので、最終的には

```
$value = 'なまえ';
```

となります。

ここまでで（URL エンコードに対する）URL デコードが終了します。
これがデコードの必要性です。

さらに $value はコンピューターで扱うための文字、Shift-JIS にコンバートされます。
$value に一意の置き換えを行った後、%Fm というハッシュに格納されます。

ループ構文を繰り返し最終的にすべての $pair をハッシュに格納します。

```
%Fm = ('nm',' なまえ ','ps',' ぱす ');
```

このハッシュを各々のルーチン内でフォームで送信された情報として参照します。

```
%Fm{'nm'} = ' なまえ ';
%Fm{'ps'} = ' ぱす ';
```

キーワード解説

$ENV{'hoge'} は環境変数を表します。
環境変数とはクライアント（プレイヤー）側の情報で、%ENV という連想配列に格納されます。

■ REQUEST_METHOD

スクリプト（CGI）へのデータの受け渡し方法（GET または POST）を格納しています。

■ CONTENT_LENGTH

POST によるフォームデータの長さ（バイト数）を格納しています。

■ QUERY_STRINGS

GET によるフォームデータそのものを格納しています。

■ read 関数（read FILEHANDLE, SCALAR, LENGTH, [OFFSET]）

ファイルハンドルからデータを読み込む関数です。
FILEHANDLE から LENGTH 分の長さのデータを読み取り SCALAR に代入します。
OFFSET を指定することで任意の部分から LENGTH 分の長さの読み出しが可能になります。

■ STDIN

標準入力の意味で、パソコンの場合はキーボード入力が標準入力デバイスとして指定されています。
FILEHANDLE が指定されなかった場合も STDIN になります。

■ split 関数（split /PATTERN/, [EXPR, LIMIT]）

PATTERN をスプリッターとして EXPR を分解します。
LIMIT を指定することで分解する最大数を指定することができます。

■ foreach 構文（foreach SCALAR (LIST) { BLOCK }）

LIST を順に SCALAR へと代入し、BLOCK を実行します。
LIST のからの代入が終了した時点でループは終了します。

■ pack 関数（pack TEMPLATE, LIST）

LIST を TEMPLATE で指定したフォーマット文字によりパック（変換）します。
長いし複雑なので省略します。
ただし日本語のデコード操作には必須です。

■ hex 関数（hex EXPR）

EXPR を 16 進数と解釈し、10 進数へと変換します。

■ tr/// 演算子（tr/SEARCH/REPLACE/cds）

検索文字 SEARCH に含まれる各文字を置き換え文字 REPLACE に一文字ずつ置き換えます。
この場合の一文字ずつ置き換えるとは、例えば

```
$str = 'ABACBC';
$str =~ tr/ABC/DEF/;
```

のとき、tr 演算子は A を D に、B を E に、C を F に置き換えます。結果は

```
print $str;
# DEDFEF
```

となります。
tr/// 演算子の詳しい動作、オプションについては省略します。

※ SEARCH、REPLACE とも、正規表現ではないので注意。

■ s/// 演算子（s/PATTERN/REPLACE/egimosx）

検索文字 PATTERN を使って検索を行い、PATTERN にマッチする文字が見つかれば REPLACE で置き換えます。
s/// 演算子の詳しい動作、オプションについては省略します。

※ SEARCH は正規表現、REPLACE は正規表現ではないので注意。

関連項目

・コラム

このページのコメント

・ページ名は sub decode のほうがいいかもしれない・・・ - Uchimata (2007 年 01 月 15 日 03 時 47 分 03 秒)