インターネット技術特論

F:Ruby CGI

山口 実靖

http://www.ns.kogakuin.ac.jp/~ct13140/inet

課題(は)

- 本学Webサーバに自作RubyCGIプログラムを公開し，そのURLとプログラムをメールにて提出.
  - 宛先"ct13140@ns.kogakuin.ac.jp"
  - メールの件名(Subject:)は，必ず"インターネット技術特論課題は"とすること.
    - 途中に空白無し. ダブルクォーテーション不要.
  - メール本文に，「学籍番号，研究科(学部)，専攻(学科，コース)，研究室，氏名，URL」を記述.
  - 作成したプログラムをメールに添付.
- 提出期限 指定日の23時59分

質問は早めにMLに！

課題(に)

- 本学WebサーバにRuby作成した掲示板CGIプログラムを公開し，そのURLとプログラムをメールにて提出.
  - 宛先"ct13140@ns.kogakuin.ac.jp"
  - メールの件名(Subject:)は，必ず"インターネット技術特論課題に"とすること.
    - 途中に空白無し. ダブルクォーテーション不要.
  - メール本文に，「学籍番号，研究科(学部)，専攻(学科，コース)，研究室，氏名，URL」を記述.
  - 作成したプログラムをメールに添付.
- 提出期限 指定日の23時59分

質問は早めにMLに！

掲示板とは

- 必須の機能
  - ブラウザからユーザ書き込める.
    - 入力は最低でも，"ユーザ名"と"書き込み内容"を含む.
  - 過去の書き込みを読める.
- 好ましい機能
  - 1回の表示記事数が有限.
    - 「過去の全ての書き込み内容を表示」だと大変なことに？

000.cgi

```
#!/usr/local/bin/ruby
print "Content-type: text/html¥n¥n"
print "hello,<br>world!<br>¥n"
```

実行結果

```
hello,<br>world!<br>
```


001.html

```
<html>
  <body>
    <form action="001.cgi" method="GET">
      <input type="text" name="abc">
      <input type="text" name="def">
      <input type="text" name="ghi">
      <input type="submit" value="送信">
    </form>
    <hr>
    <form action="001.cgi" method="POST">
      <input type="text" name="abc">
      <input type="text" name="def">
      <input type="text" name="ghi">
      <input type="submit" value="送信">
    </form>
  </body>
</html>
```


001.cgi

```
#!/usr/local/bin/ruby
print "Content-type: text/html¥n¥n"

print ENV["REQUEST_METHOD"], "<br>¥n"
if ENV["REQUEST_METHOD"] == "GET" then
        print ENV["QUERY_STRING"], "<br>¥n"
else
        print $stdin.gets
end
```


002.html

```
<html>
  <head>
    <title></title>
  </head>
  <body>
    <form action="002.cgi" method="POST">
      <input type="text" name="abc">
      <input type="text" name="def">
      <input type="text" name="ghi">
      <input type="submit" value="送信">
    </form>
  </body>
</html>
```

002.cgi

```
1  #!/usr/local/bin/ruby
2  print "Content-type: text/html¥n¥n"
3  indata = nil
4  if ENV["REQUEST_METHOD"] == "GET" then
5          indata = ENV["QUERY_STRING"]
6  else
7          indata = $stdin.gets
8  end
9  print indata, "<br>¥n<hr>¥n"
10 pairs = indata.split(/&/)
11 pairs.each do | p |
12         print p, "<br>¥n"
13 end
14 print "<hr>¥n"
15 print "<table border=1>¥n"
16 pairs.each do | a_pair |
17         tokens = a_pair.split(/=/)
18         print "<tr>"
19         print "<td>", tokens[0], "</td>"
20         print "<td>", tokens[1], "</td>"
21         print "</tr>¥n"
22 end
23 print "</table>¥n"
```


003.html

```
<html>
  <head>
    <title></title>
  </head>
  <body>
    <form action="003.cgi" method="POST">
      name : <input type="text" name="name"> <br>
      txt : <input type="text" name="txt"> <br>
      <input type="submit" value="送信">
    </form>
  </body>
</html>
```


003.cgi

```
#!/usr/local/bin/ruby
require "cgi"

print "Content-type: text/html¥n¥n"
cgi = CGI.new

in_name = cgi["name"]
in_txt = cgi["txt"]
print "<html><body>¥n"
print "name -> ", in_name, "<br>¥n"
print "txt -> ", in_txt, "<br>¥n"
print "</body></html>¥n"
```

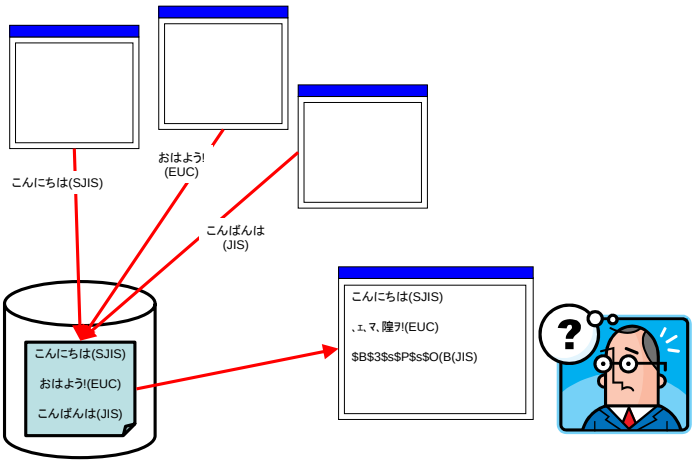

日本語処理

- 日本語の文字コード体系では以下の3個が有名
  - Shift_JIS (SJIS)：主にWindows, Macで使用
  - ISO-2022-JP (JISコード)：E-mailで使用される
  - EUC-JP：主にUNIXで使用される
    - 最近はUTF-8も有名.
- 文字コード体系とは，「文字コード(数値)と文字」の対応関係の規格.
  - よって，別の規格を使用していると「ある数値(文字コード)がどの文字を意味するか」が異なる.
  - 文字化けの原因となる.

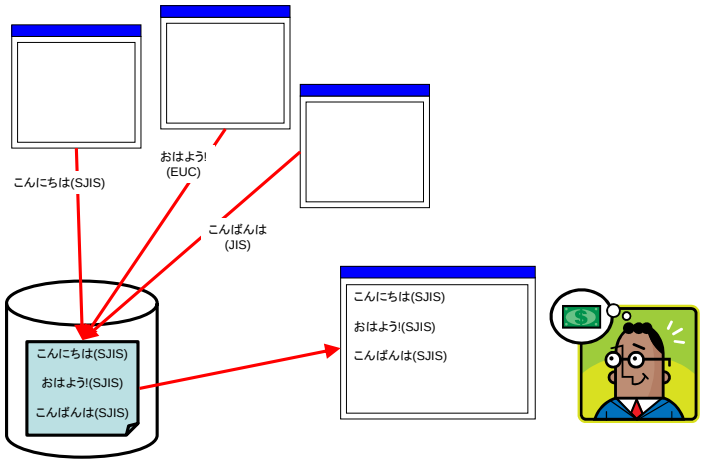
日本語処理

- 1個のファイルに複数の文字コードを混在させ，それを正しく表示させることは(多くの場合)不可能.
  - 通常，「このファイルは文字コード〇〇で記述されている」と，ファイル単位で扱う.
- ブラウザから送信される日本語文字列が使用している文字コードは複数種類あり，不統一.
  - それらを，そのまま単一ファイルに格納し，そのまま(複数の文字コードが混在したまま)ブラウザに渡すと，ブラウザには正しく表示されない．文字化け.

日本語処理



日本語処理



003e.cgi

```
#!/usr/local/bin/ruby -Ke
require "kconv"
require "cgi"

print "Content-type: text/html¥n¥n"
cgi = CGI.new
name = cgi["name"]
txt = cgi["txt"]
name = Kconv::toeuc(name) if name
txt = Kconv::toeuc(txt) if txt

print "<html><body>¥n"
print "名前 : name -> ", name, "<br>¥n"
print "テキスト : txt -> ", txt, "<br>¥n"
print "</body></html>¥n"
```

注意
このファイルにも
日本語が含まれている.

このファイル自体を
EUC形式で
保存する必要がある.

003e.cgi の解説

```
#!/usr/local/bin/ruby -Ke
```
日本語を含んだソースコードを実行する際に，その文字コードを指定．
SJISなら -Ks， JISなら -Kj， EUCなら -Ke

```
require "kconv"
```
プログラム内で，Kconvを使用する場合に記述する．
C言語の #include に近い．

```
name = Kconv::toeuc(name) if name
```
Kconv::toeuc()で，文字列がEUC形式に変換される．
同様に，Kconv::tosjis，Kconv::tojisがある．

```
print "名前 : name -> ", name, "<br>¥n"
```
このプログラム自体が日本語文字列を含むが，-KeにあわせてEUCにする．

Formへのタグ入力への対策

- 無警戒に以下のプログラムの公開は危険
  - (1) ユーザからのForm入力を受信
  - (2) 受信内容をブラウザに表示
- ユーザが，いたずらでHTMLタブを入力ことがある．
- プログラミング言語によってはシェル展開される
  - `print "`cat /etc/passswd`"`

008.cgi （例0）

CGIプログラム

```
name = cgi["name"]
txt = cgi["txt"]
print "<table border=1>¥n"
print "<tr><td>name</td><td>txt</td></tr>¥n"
print "<tr><td>",name,"</td>"
print "<td>",txt,"</td></tr>¥n"
print "</table>"
```

ユーザが，nameに「sane」
txtに「hello」と入力すると

出力

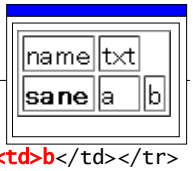
```
<table border=1>
<tr><td>name</td><td>txt</td></tr>
<tr><td>sane</td><td>hello</td></tr>
</table>
```


008.cgi （例1）

CGIプログラム

```
name = cgi["name"]
txt = cgi["txt"]
print "<table border=1>¥n"
print "<tr><td>name</td><td>txt</td></tr>¥n"
print "<tr><td>",name,"</td>"
print "<td>",txt,"</td></tr>¥n"
print "</table>"
```

意地悪なユーザが，
nameに「sane」
txtに「a</td><td>b」と入力すると



出力

```
<table border=1>
<tr><td>name</td><td>txt</td></tr>
<tr><td><b>sane</b></td><td>a</td><td>b</td></tr>
</table>
```


Formへのタグ入力への対策

- 有名な対処策 （0）
  - < や > が含まれている書き込みは拒否する。
- 有名な対処策 （1）
  - < や > はエスケープして記号を表示する．

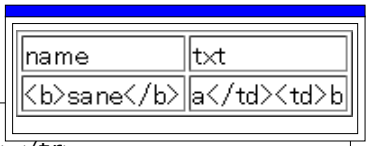
  例 ：「sane」という入力は，
  「sane」に変換して渡す．
  ブラウザに「sane」を渡すと，
  ブラウザに「sane」と表示される．
  - 復習 ： HTMLタグ「<」は，ブラウザに「<」と表示される．

008esc.cgi

CGIプログラム

```
name = cgi["name"] ; txt = cgi["txt"]
name = CGI::escapeHTML(name) if name
txt = CGI::escapeHTML(txt) if txt
(略)
print "<tr><td>",name,"</td>"
print "<td>",txt,"</td></tr>¥n"
```

意地悪なユーザが，
nameに「sane」
txtに「a</td><td>b」と
入力すると




```
<table border=1>
<tr><td>name</td><td>txt</td></tr>
<tr><td>&lt;b&gt;sane&lt;/b&gt;</td><td>a&lt;/td&gt;&lt;td&gt;b</td></tr>
</table>
```

出力