情報システム技術論'08 — ＃3

久野　靖*

2008.4.24

1　World Wide Web

1.1　インターネットの歴史

□ インターネットの起源→ 1969 年 (ARPANET)

- 最初は非常に限られた世界

□ インターネットの研究機関への普及→ 1980 年代

- CFNET(1981)、NSFNET(1987) →研究機関が多くネットに接続
- 日本でも 1988 年 WIDE Internet により TCP/IP 接続が開始

□ インターネットの社会への普及→ 1990 年代

- IIJ →日本で最初のプロバイダ→ WIDE 関係者が中心となり 1992 年末設立

1.2　初期のインターネットの利用方法…

□ 電子メール→現在でもメインのサービス

□ ネットニュース→ Web 掲示板の利用が増えたためマイナー化

- メール、ニュースは IP 接続以前から利用できていた

□ telnet →他のマシンに接続して利用するのは重要な利用形態だった

□ FTP → telnet と対にして自分のアカウント間でファイルを転送するのに利用

- 匿名 FTP →フリーソフトの配付手段として

□ 情報公開/共有の仕組みとして…

- 匿名 FTP ではファイルの置き場所が分からないと役に立たない
- ネットニュースや「匿名 telnet」の利用
- goher →メニューベースで情報を次々に見ることができる
- WAIS →キーワード検索で問い合わせに適合性の高い文書をリストアップ

□ これらはいずれも

- 別個のサービスであり、
- 個別の特性を知っていて使い分ける必要があり、
- それぞれを活用するためのコマンドも全く別のものだった
- →インターネットから情報を取り出すのは熟知した人にしかできなかった

1.3　World Wide Web のはじまり

□ 「World Wide Web とは何か」定義できます?

□ CERN(欧州粒子物理研究所) の研究者 Tim Berners-Lee が 1991 年に開発

- 研究者どうしの情報流通をスムーズにするための手段として作った
- 主要なアイデア→ハイパーテキスト、URI によるポインタ
- WWW サーバ→ CERN httpd
- 当初は文字のみのブラウザ、NeXT Station (マイナーなシステム) で稼働

□ Mosaic --- 最初のグラフィカルブラウザ

- イリノイ大 NCSA の学生 Marc Andressen が 1992 年末に開発
- 1993〜1994 に爆発的に広まる (最初は Unix/X →後に Mac、Windows 版も)
- 当然、最初は英語のみ→高田敏弘@NTT　が Mosaic-L10N を開発→日本語も可能に
- 計算機研究者の間で大ブームに (1994 ころ)
- NCSA は NCSA httpd も公開

*筑波大学大学院経営システム科学専攻

□ イリノイ大の学生グループはその後も Mosaic の開発を続けていたが…

- 大学は Mosaic を自分のところのソフトとして金儲けしようとした
- 自由にやりたい学生グループとの対立

□ Sillicon Graphics Inc. (SGI) の創立者 Jim Clark は WWW が商売になると考え、Netscape Communications 社を設立して Marc Andressen ほか学生グループの主要メンバーを引き抜く (1994)

- Netscape ブラウザ→圧倒的な支持を得てブラウザのシェア No. 1 に
- 最初から Unix/X、Windows、Mac で稼働
- Netscape 1.1 からは日本語も OK →日本でもインターネットブームに

□ それで結局、WWW とは?

- ネットワークにまたがるハイパーテキストシステム
- 文書は世界中のサーバによって公開されている (容量ば無尽蔵、オリジナルソースの情報は常に新しい)
- 文書が相互に (サーバをまたがって) ハイパーリンクされている
- URI で記述できるもの (ネット上の資源) は何でもリンク可能

1.4 ブラウザ戦争

□ Microsoft 社は当初は MSN(自前のパソコン通信) に顧客を囲いこもうとしてアンチインターネット路線だった

□ インターネットがもはや主流だと分かると路線転換→自前のブラウザ Microsoft Internet Explorer(MSIE) を普及させて Netscape 支配を止めようとした

- MSIE を Windows に標準付属 (Win 95〜)、しかも OS 組み込みにして取り外せないように (Win 98〜)
- PC を出荷しているメーカに圧力を掛けて Netscape ブラウザを付属させないようにした (→訴訟の対象となっている行為の 1 つ)
- さまざまな「新技術」を取り入れて差別化を計る (どれくらいそれが「有難い」かどうかは…?)

□ Netscape 社はブラウザでは食べて行けなくなり、ブラウザは無償化 (これまで儲かっていたビジネスが突然なくなるとは…)

- Netscape 4.x の次のブラウザはオープンソース開発モデルにより開発するとして、1998 年にソースコードを公開→ Mozilla プロジェクト。Mozilla ブラウザを公開
- そうしている間にも経営は苦しくなり AOL により買収
- AOL は Mozilla プロジェクトの成果をもとに Netscape 6 を公開 (2000 年末)。Netscape 9 までバージョンは進んだが 2008.2 で開発終了。→ FireFox 2 まで発展。15%くらいのシェアを持つようになった。

□ MSIE →圧倒的シェア、標準への準拠はそこそこ、よく分からない独自機能、よくセキュリティ上の問題が発見される

□ Mozilla 陣営→よりよく標準へ準拠。Mozilla の後継→ FireFox。FireFox 2 は 16%くらいのシェアを持つ (IE が嫌いな人、Linux など非 Windows システム利用者の支持)

- そのほか、Opera、Safari(Apple)、…
- あと、i-mode とか組み込みブラウザも増えている…

□ ブラウザは「WWW への単なる接続口」「WWW を見る道具」という位置付けに…

- しかし Ajax などの技術→ Web アプリの高度化もあり、一定の標準化や機能搭載は必要に→今後も進歩

2 WWW のプロトコルと Web サーバ

2.1 HTTP

□ WWW の基本的なアーキテクチャ→これまでに学んで来たクライアントサーバ型

- クライアント (主にブラウザ、最近は Web アプリも) とサーバのやりとり→ HTTP(HyperText Transfer Protocol)。もともとは Web のためのプロトコルなのだが…
- 現在のネットは組織の出入口にパケットフィルタが設置されていて、任意のプロトコルによる情報流通は制約
- しかし HTTP だけは Web のために中継されるのが普通→さまざまな用途の通信が HTTP を経由することでフィルタ越しに行われるように

□ HTTP のメッセージ形式→ヘッダとボディに分かれていて、両者の境界は「空白行」(電子メール、ネットニュースと同じ)

- 上記はクライント→サーバ (リクエスト)、サーバ→クライアント (レスポンス) とも同一
- 先頭の要求行でメソッド (コマンド)、操作対象、プロトコルを指定

```
要求行
フィールド: 値
フィールド: 値
...
        ←空行
データ ←データはない場合も
データ (HTTP 要求ではない方が多い)
...
```

- 先頭応答行は状態コードを返す

```
応答行
フィールド: 値
フィールド: 値
...
        ←空行
データ ←データはない場合も
データ (HTTP リダイレクト等)
...
```

□ 要求行

```
メソッド URI HTTP/1.1 (or 1.0)
```

- HTTP のメソッドは GET、HEAD、POST、PUT、DELETE、OPTIONS、TRACE、あと拡張メソッドがあることがある。
- GET --- リソースを取得する (一番多い)
- HEAD --- GET と同じだがヘッダ情報だけ受け取り本体は省略
- POST --- 情報を送付し応答を受け取る
- これ以外はあまり使わない (少なくともブラウザは使わない)

□ 要求ヘッダ

- Accept、Accept-Charset、Accept-Encoding 等 --- どの種別のデータ、文字セット、符号化、等を受け取れるかを通知
- Authorization --- 認証時に使用
- Cache-control --- キャッシュ制御に使用
- Connection --- 接続を維持するかどうか
- Host --- どのホスト名でアクセスするか
- Referer --- どの URI にあったリンクからこのアクセスが発生しているかを通知 (どのページから来たかが分かる)。英語のスペルミス (正しくは referrer のはず) のまま規格になってしまった
- If-Modified-Since、If-Match 等 --- 条件つき GET 用
- User-agent --- ブラウザ種別等の送付
- Cookie --- クッキーの送付 (規格外)

□ 応答行

```
HTTP/1.1 コード 説明
```

- 2xx --- 成功「200 OK」「201 Created」
- 3xx --- リダイレクト「301 Moved Permanently」「303 See Other」
- 4xx --- クライアントエラー「403 Forbidden」「404 Not Found」
- 5xx --- サーバエラー「500 Internal Server Error」「503 Service Unavailable」

□ 応答ヘッダ

- Content-type --- 応答情報の種別 (MIME タイプ)
- Location --- 移った先 (リダイレクト用)
- Date --- 日付
- Server --- サーバ種別等の送付
- Refresh --- 何秒たったらどこへ移動してね
- Set-Cookie --- クッキーの送付 (規格外)

2.2 WWW サーバ

□ Web サーバとは結局、上記の HTTP をやりとりするサーバ→「ふつうに」Web ページを返さなくてもいい。簡単な例を見てみる。

```
import java.net.*;
import java.io.*;

public class Sample31 {
  public static void main(String[] args)
      throws Exception {
    ServerSocket ss = new ServerSocket(4192);
    String mesg = "Hello.";
    while(true) {
      Socket cs = ss.accept();
      PrintWriter out = new PrintWriter(
        cs.getOutputStream(), true);
      BufferedReader in = new BufferedReader(
        new InputStreamReader(cs.getInputStream()));
      String path = "";
      while(true) {
        String line = in.readLine();
        System.out.println(line);
        if(line.equals("")) break;
        String[] a = line.split(" +");
        if(a[0].equals("GET")) path = a[1];
      }
```

```
      out.println("HTTP/1.0 200 OK");
      out.println("Content-type: text/plain");
//    out.println("Refresh: 5; URL=/");
      out.println("");
      out.println(mesg);
      out.println(path);
      if(!path.equals("/")) mesg = path;
      cs.close();
      if(path.equals("/bye")) break;
    }
    ss.close();
  }
}
```

- サポートするのは GET リクエストだけ
- 接続を待ち受け、来たらば各行を読み込み空白で区切る
- GET リクエストのときは URI(パス) を記録
- 空行が来たらリクエスト読み込みは終わり
- OK 行と Content-type: text/plain を出力
- 記憶しておいた文字列と受け取ったパスを出力
- 受け取ったパスを新たに記憶
- 受け取ったのが「/bye」ならおしまい

□ 演習

- このサーバをこのまま動かしなさい。ブラウザで「http://ホスト:4192/適当な文字列」を開いて動作を観察する。
- クライアントからどのようなヘッダが来ているかも観察する。
- 他人のサーバに接続して/他人に接続してもらって同様に観察しなさい
- コメントアウトしたる Refresh ヘッダを出力して「定期的に再読み込み」させてみる
- 応答を「HTTP/1.0 301 Moved Permanently」と「Location: http://www.yahoo.co.jp」だけにして同様に観察しなさい

□ このように、HTTP サーバが渡されて来た URI をどのように扱うかは完全にサーバに任されている

□ ファイルを読み込んで返すサーバ

```
import java.net.*;
import java.io.*;

public class Sample32 {
  public static void main(String[] args)
      throws Exception {
    ServerSocket ss = new ServerSocket(4192);
    String mesg = "Hello.";
    while(true) {
      Socket cs = ss.accept();
      PrintWriter out = new PrintWriter(
        cs.getOutputStream(), true);
      BufferedReader in = new BufferedReader(
        new InputStreamReader(cs.getInputStream()));
      String path = "";
      while(true) {
        String line = in.readLine();
        System.out.println(line);
        if(line.equals("")) break;
        String[] a = line.split(" +");
        if(a[0].equals("GET")) path = a[1];
      }
      out.println("HTTP/1.0 200 OK");
      out.println("Content-type: text/plain");
      out.println("");
      try {
        BufferedReader rd = new BufferedReader(
          new FileReader(path));
        String line = rd.readLine();
        while(line != null) {
          out.println(line); line = rd.readLine();
        }
        rd.close();
      } catch(Exception ex) {
        out.println("file " + path + " not found.");
      }
      cs.close();
      if(path.equals("/bye")) break;
    }
    ss.close();
  }
}
```

- さすがに 1 行とか 2 行ではつまらないので、パスで指定されたファイルを返送するように直す
- パスは Unix の絶対パス名として扱う。本物のサーバではこのようにしては絶対にいけない (なぜか?)
- 内容としてファイルを読み込み返送するだけ
- ファイルが無かった場合は代わりにメッセージを返す

□ やってみると…

- テキストファイルはちゃんと取得できる
- HTML ファイルもテキストファイルとして表示 (Content-type: text/plain としているから)

□ とりあえず名前が「.html」で終わるファイルだけ Content-type: text/html を返すように直す

```
import java.net.*;
import java.io.*;

public class Sample33 {
  public static void main(String[] args)
      throws Exception {
    ServerSocket ss = new ServerSocket(4192);
    String mesg = "Hello.";
    while(true) {
```

```
      Socket cs = ss.accept();
      PrintWriter out = new PrintWriter(
        cs.getOutputStream(), true);
      BufferedReader in = new BufferedReader(
        new InputStreamReader(cs.getInputStream()));
      String path = "";
      while(true) {
        String line = in.readLine();
        System.out.println(line);
        if(line.equals("")) break;
        String[] a = line.split(" +");
        if(a[0].equals("GET")) path = a[1];
      }
      System.out.println(path);
      out.println("HTTP/1.0 200 OK");
      if(path.matches(".*\\.html$"))
        out.println("Content-type: text/html");
      else
        out.println("Content-type: text/plain");
      out.println("");
      try {
        BufferedReader rd = new BufferedReader(
          new FileReader(path));
        String line = rd.readLine();
        while(line != null) {
          out.println(line); line = rd.readLine();
        }
        rd.close();
      } catch(Exception ex) {
        out.println("file " + path + " not found.");
      }
      cs.close();
      if(path.equals("/bye")) break;
    }
    ss.close();
  }
}
```

□ 演習

- このサーバをコピーしてきて動かせ。久野の HTML ファイルは/u1/kuno/WWW/index.html にあります。
- リンクをたどったりしてどのような不都合があるか観察せよ。またその不都合はなぜ生じているのか考える。

3 WWW の記述言語

3.1 HTML (HyperText Markup Language)

□ WWW 上では WYSIWYG(見たまま) はあり得ない (なぜ?)

□ 理由: ユーザによって持っている画面もブラウザも違う→同一の「見た目」は不可能→「見たまま」も不可能

- c.f. Adobe PDF →印刷イメージそのまま。画面で読めるか? 読めるとすれば「十分な大きさの画面を持つ人」だけのはず。
- 「見たまま」が無理とすればどうすればいい?

□ 回答: 「意味マークアップ」→ブラウザがそれぞれの環境に合わせて整形。

- マークアップ…ファイル中に「印」をつけることで付加情報を表現
- 意味マークアップ…「ここは見出し」「ここは段落」など、文章の「意味」に対応してマークアップする。
- 「見出し」「段落」などをどう表示するのがよいかはそれぞれの環境に応じて決まる→ブラウザにおまかせする。
- c.f. 命令マークアップ「ここは 12pt」「ここはクーリエフォント」これは環境に応じて変化させられないからダメ。

3.2 HTML のバージョン

□ HTML1.0 → HTML2.0 → HTML3.2 → HTML4.0 → HTML4.01(最終版)

□ 以下では HTML 4.01 を解説

- それ以前と比較して「意味マークアップ」を重視。
- 意味マークアップでない機能 (フォントの変更、色の変更など) をすべて非推奨に。
- 見た目 (表現) の指定は CSS(スタイルシート) で指定。

□ XML(eXtensible Markup Language) の普及

- XML では「用途に応じて自由にタグを作って使える」
- XML の形式で HTML の機能を記述するもの→ XHTML。1.0、1.1
- XHTML もある程度使われるようになっている。構文上の制約が厳しいのでまあ今回はパスで。

3.3 HTML の基本

□ HTML は SGML(Standard Generalized Markup Language) を土台とする言語。

- SGML ではマークアップを「<名前>…</名前>」という形で範囲 (要素) の指定を行う。「<名前>」は開始タグ、「</名前>」は終タグ。

- 単独のタグ (開始タグのみで終了タグがないもの) もある。
- 追加の指定は属性を付加。「<名前 属性 1=値 属性 2=値 …>」

□ 最も基本的な HTML の構造:DOCTYPE+ヘッダ+本体

- DOCTYP 宣言は「どのバージョンの文書か」表明するのに必要

```
<DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>Sample</title>    ←タイトル
</head>
<body>
...        ←ここにページ本体の中身を記述
</body>
</html>
```

□ ここまでに出て来た HTML の要素:

- <html>…</html>: HTML 記述全体を表す
- <head>…</head>: ヘッダ (ページに関する情報) の範囲
- <title>…</title>: 文書のタイトルを表す
- <body>…</body>: ページ本体 (ブラウザ窓の中身)

□ 文字エントリ: 「特別な文字」をあらわす。

```
<  → &lt;    タグ開始文字
>  → &gt;    タグ終了文字
&  → &amp;   文字エントリ開始文字
```

□ ページ本体の中身→見出し、段落など。

□ 例: sam31.html

```
<DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>Sample</title>
</head>
<body>
<h1>自己紹介</h1>
<h2>仕事関係</h2>
<p>私は大学で先生をやって
います。教えるのは好きです。</p>
<h2>趣味関係</h2>
<p>コンピュータは趣味です。趣味を
仕事にするのもよしあしです。</p>
</body>
</html>
```

□ ここまでに出て来た HTML の要素:

- <h1>…</h1>: 第一レベルの見出し (大見出し)
- <h2>…</h2>: 第二レベルの見出し (h6 まである)
- … : 普通の段落 (地の文)

□ あとちょっとおまけ:

-
: 改行
- <hr>: 区切りの横線

□ 実習:上の HTML をコピーしてきて表示せよ。

```
mkdir WWW    ←最初 1 回だけ
chmod a+rx WWW   ←最初 1 回だけ
cp /u1/kuno/work/sam31.html WWW/sam31.html
chmod a+r WWW/sam31.html   ←コピーしたら毎回
ブラウザで「http://w3in/~ユーザ/sam31.html」開く
```

□ 実習:上の HTML の内容を変更してみよ。
と<hr>も追加して効果を確認せよ。

- Emacs で「WWW/sam31.html」を開く。
- 編集して保存。ブラウザの再読み込み。

3.4 CSS(Cascading StyleSheet)

□ HTML 4.01 では「意味マークアップ」

- どのような色/大きさ/配置という指定は HTML では行わない
- これらの (表現の) 指定→スタイルシートの機能
- CSS: スタイルシート記述言語として最も広く使われている

□ 基本的な書き方:「どういう要素はどう表示」

```
セレクタ { 属性:値; 属性:値; … }
```

- セレクタ→タグ名 (とりあえず)。
- 属性→指定するものの種類。
- 値→その種類の値をこれこれに設定せよ。

□ 値の書き方:

- 長さ→「10mm」(ミリ)「5px」(ピクセル)「3ex」(文字幅) など単位指定が必須。
- 割合→「100%」「40%」などパーセントで。
- 色→「rgb(赤, 緑, 青)」赤/緑/青の強さを 0〜255 の整数で指定。

□ CSS の主要な属性

- 「color: 色」文字の色
- 「background-color: 色」背景の色

- 「margin: 長さ 長さ 長さ 長さ」上下左右マージン。長さのかわりに「auto」も指定できる。
- 「padding: 長さ 長さ 長さ 長さ」上下左右の詰め物
- 「text-indent: 長さ」段落 1 行目の字下げ
- 「text-decoration: 種別」文字飾り。種別は「underline」「overline」「blink」など。点滅は絶対使わない (ださい)
- 「text-align: 種別」そろえ。「left」「center」「right」
- 「border: 種別 色 長さ」枠を指定。長さは枠の幅。種別は「solid」「double」「dashed」「dotted」「ridge」「groove」「inset」「outset」「none」

□ CSS の HTML への入れ方 (複数の方法があるがとりあえず 1 つ):

```
<DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>Sample</title>
<style type="text/css">
body { background-color: rgb(254,200,189) }
h1 { text-align: center;
     text-decoration: underline }
</style>
</head>
<body>
<h1>自己紹介</h1>
<h2>仕事関係</h2>
<p>私は大学で先生をやって
います。教えるのは好きです。</p>
<h2>趣味関係</h2>
<p>コンピュータは趣味です。趣味を
仕事にするのもよしあしです。</p>
</body>
</html>
```

□ 演習: 自分の HTML ファイルに CSS のスタイル指定をつけてみよ。具体的には次のようなことを試せ。

- 見出しを適当な色の枠で囲む (左右 margin は auto にするとよい)。
- 段落の字下げを 1 文字にする (日本語ではそれが標準的)。
- 段落の背景色を適当な色に設定する。

□ スタイル設定に使う汎用のマークアップ:

- … : ブロック単位のグループ
- …: 段落の中の部分文字列
- CSS で「. 名前」をセレクタとして指定→ class の名前に対応

□ 例: 一部をカコミにして下線つける

```
<DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>Sample</title>
<style type="text/css">
body { background-color: rgb(254,200,189) }
h1 { text-align: center;
     text-decoration: underline }
.box { border: ridge blue 6px;
       padding: auto 3mm }
.key { font: red; text-decoration: underline }
</style>
</head>
<body>
<h1>自己紹介</h1>

<div class="box">
<h2>仕事関係</h2>
<p>私は大学で先生をやって
います。教えるのは好きです。</p>
</div>

<h2>趣味関係</h2>
<p><span class="key">コンピュータ</span>
は趣味です。趣味を
仕事にするのもよしあしです。</p>
</body>
</html>
```

□ 演習: 自分の例題に div と span も追加してみよ。

3.5 フォーム機能

□ ここまでの範囲では、Web ページは「静的な、変化しないもの」

- 情報を見るだけならこれでも役に立つが…
- やはり「ユーザから入力→サーバの応答」というやりとりがしたい

□ 「動的なページ」の定義は?

- (a) ユーザからの入力に応じた内容が返されるもの (それ自体は変化しない)
- (b) ページの内容が表示後も変化するもの
- 一応、(a) だけでも役に立つ Web アプリケーションは開発可能

□ フォーム: 決まった書式のこと

- HTML では: ページの中に「穴」があってそこに入力を入れられる

- 埋めた入力はサーバに送信 (submit、提出) できる
- 送信の形式: 2 種類

□ HTTP GET: URI の後ろに「?名前=値&名前=値&…」という形で付加

- △ URI を見ると送信データが分かってしまう
- ○自分の手でも簡単に作れる、ブックマークに入れられる

□ HTTP POST: データを別途送信

- ○データが見られにくい (通信路を監視すれば見られるけど)
- △フォームからしか送れない、ブックマークできない

□ フォームの構造

```
<form ...>
  ここに普通の要素と一緒に部品を入れる
</form>
```

□ form タグの属性:

- name="名前" → フォーム名。参照時に必要になることがある
- method="post" または method="get" → 送信形式
- action="URI" → フォームのデータを受け取る対象

□ おもな部品:

- <input name="名前" type="text" [size="文字数"]> ←入力欄
- <input name="名前" value="値" type="hidden"> ←「見えない部品」
- <input name="名前" type="checbox"> ←チェックボックス
- <input name="名前" type="radio" value="値" [checked]> ←ラジオボタン
- <textarea rows="行数" cols="文字数"></textarea> ←複数行入力欄
- <select name="名前">項目…</select> ←選択メニュー
- <option value="値" [selected]>表示文字列</option> ←選択メニューの項目
- <button name="名前" value="値">表示文字列</button> ←提出ボタン

□ 例: sam34.html

```
<DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>Sample</title>
<style type="text/css">
.sheet { background-color: rgb(200,220,240);
         padding: 5px 10px }
</style>
</head>
<body>
<h1>フォームのテスト</h1>
<form name="f0" method="get" action="#">
<div class="sheet">
名前: <input name="ident" type="text" size="12"><br>
性別:
男性<input name="sex" type="radio" value="male" checked>
女性<input name="sex" type="radio" value="female"><br>
学年: <select name="grade">
<option value="1" selected>1 年生</option>
<option value="2">2 年生</option>
<option value="3">3 年生</option></select><br>
<button name="cmd" value="x">提
出</button></div></form>
</body>
</html>
```

□ 演習: 上の例をコピーしてきて表示せよ。実際にデータを入れて送信し、method="get"だとどのような URI になるのか検討せよ。また、編集して部品を増やしてみよ。

4 PHP による Web アプリケーション

4.1 CGI

□ CGI (Common Gateway Interface): Web サーバからサーバ上のプログラムを呼び出す時の「インタフェース」

- ブラウザ→サーバ→プログラム: 「環境変数」によって必要な情報を渡す (加えて method="post"では標準入力からデータが読み込める)。
- プログラム→サーバ→ブラウザ: 標準出力による: 先頭に「ヘッダ情報」、続いて「空行」(ヘッダの終わりを表す)、その後に任意のデータ。

□ 例: sam35.cgi --- 環境変数をそのまま表示 (プレーンテキスト出力)

```
#!/bin/sh
echo 'Content-type: text/plain'  ←最低限のヘッダ
echo ''      ←ヘッダ終わり
env          ←環境変数表示
```

□ 例: sam36.cgi --- ヘッダで別ページへ転送

```
#!/bin/sh
echo 'Location: http://www.yahoo.com'
echo ''
```

□ 例: sam34.cgi --- さっきのフォームの結果受け取り (環境変数 QUERY_STRING で GET のパラメタが受け取れる)

```
#!/bin/sh
echo 'Content-type: text/html; charset=euc-jp'
echo ''
echo '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">'
echo '<html><head><title>sample</title></head><body>'
echo '<h1>見本</h1><p>' $QUERY_STRING '</p></body></html>'
```

□ しかしちゃんとパラメタを受け取って内部の処理も行い、結果の HTML を出力するのはかなり大変そうですね?

4.2 PHP

□ PHP (Hypertext Prosessor) --- 初期は「Personal HomePage tools」。ともかくプログラミング言語の 1 つ。

- 最大のアイデア: 「ほとんどは HTML なんだから HTML を書いて、ちょっとプログラムを使いたいとこだけプログラムを埋め込めばいい」
- 「<?php ……PHP プログラム…… ?>」の中だけプログラムコード
- 残りの (外側の) 部分→そのまま出力 (実は PHP のそういう命令に翻訳されて実行されるけど一見 HTML がそのままあるように見える)
- 重要: ヘッダの出力動作は「先頭で」やらないといけない (HTML 本文に入った時はヘッダは終わっていることに注意)

□ 基本的なカタチ

```
<?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>
```

□ 例: sam37.php --- 数を出力する

```
<?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.test { margin: 2px 20px; background: rgb(200,215,255) }
</style></head><body>
<?php
  for($i = 0; $i < 10; ++$i) {
    echo "<div class='test'>番号".$i."です。</div>";
  }
?>
</body></html>
```

□ 解説:

- for 文とかは C や Java と同じ。変数は「$」をつける (シェルと同じ)。echo は PHP の命令で文字列を出力する。「.」は文字列連結。

□ 演習: この例題をコピーして動かせ。動いたら出力の内容を変更してみよ。

□ PHP の利点…豊富な組み込み機能 (多数のモジュールが選択可能)

□ HTML からのパラメタの受け取りなども簡単。

- 「$_GET['　名 前']」「$_POST['　名 前']」で method="get"、method="post" で渡された パラメタがすぐ利用可能。

□ 例: sam34.php --- 先のフォームのパラメタを表示。

```
<?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">
p { margin: 2px 20px; background: rgb(200,215,255) }
</style></head><body>
<?php
  echo '<p>こんにちは'.$_GET['ident'].' さん。</p>';
  echo '<p>性別は'.$_GET['sex'].' ですね。</p>';
  echo '<p>学年は'.$_GET['grade'].' ですね。</p>';
?>
</body></html>
```

4.3 PHP による Web アプリケーション

□ 前のようにフォーム (HTML) と処理 (PHP) を分けるといまいち。

- 開発しにくい、分けてあるので分かりにくい
- ある結果を表示しつつ次の入力を入れたい。
- 次々に URI が移っていくのはやりづらい。「ある特定の処理」は 1 つの URI で。

- → 1 つの PHP ページで処理+フォーム (初回は処理しないように)。

□ 例: sam38.php --- 合計の計算

```
<?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 { padding: 10px; background: rgb(200,215,255) }
</style></head><body>
<h1>合計を求める</h1>
<form method="get" action="#"><div>
<?php
  $total = 0; $data = 0;
  if(!isset($_GET['cmd'])) {
    echo "<p>値を入れてください</p>";
  } else if($_GET['cmd'] == 'clear') {
    echo "<p>クリアしました。</p>";
  } else if($_GET['cmd'] == 'calc') {
    if(is_numeric($_GET['total'])) $total = $_GET['total'];
    if(is_numeric($_GET['data'])) $total += $_GET['data'];
    echo "<p>現在までの合計は".$total."です。</p>";
  } else {
   echo "<p>処理内容が不明です。</p>";
  }
  echo '<input type="hidden" name="total" value="'.
         $total.'">';
?>
<input name="data" type="text" size="6">
<button name="cmd" value="calc">計算</button>
<button name="cmd" value="clear">クリア</button>
</div></form></body></html>
```

□ 解説:

- 提出されてくるデータは total、data の 2 つ (プラス cmd…後述)。これらに対応して PHP 側でも変数を用意。初期値は 0。
- isset() はそのデータがあるかどうか。なければ初回のはず→単にメッセージを表示
- cmd は押したボタンに応じた value の値が入って来る。
- それが'clear' だったら total、data とも 0 のままでよい。
- それが'calc' だったら total に data を加える。ただし、数値の形をしているときだけ。(ブラウザから来たデータの形は絶対にそのまま信用してはいけない。) そして total を表示。
- いずれにせよ、現在の total は type="hidden"で次回に受け渡す。
- 最後にフォーム用意する。

□ 演習: この PHP プログラムをまず久野のページで動かせ。

http://w3in/~kuno/ist08/sam38.php

複数人数でバリバリ計算しても干渉しないことを確認すること。

□ 演習: この PHP プログラムを自分のところに持って来て動かせ。動くことを確認したら、足し算の他に「引き算」もできるように改造してみよ。(もっと色々できるようにしてもよい。)

□ 先の例題は「アプリケーションに必要なすべての情報を毎回ブラウザとサーバで受け渡す」ことで成立していた。

□ これが、データを一部サーバで保管しておこうとすると面倒なことが起きる。たとえば total をファイルに格納しておくとどうなるか? (もちろん、これだけのデータならそんな必要はないが、もっと大量のデータがあるとか、ネット上に流せない機密データだと、サーバ上に保管しておく必要がある。)