第10章　付録: World Wide Web

久野の担当は今回でおしまいなので、ちょっと順序が前後するけれど今回は「お楽しみ」として World Wide Web の話をして、HTML で自分の Web ページを作ったり CGI で対話的コンテンツを作ったりしてみよう。これらの体験から計算機とネットワークの「総合的な可能性」についていくらかでも知って頂ければ嬉しく思う。

10.1　WWW とその概念

10.1.1　ハイパーテキストとは

といっても、皆様は既に WWW を体験されているのでハイパーテキストについては知っていることになる。改めてまとめると、ハイパーテキストとは

- 計算機の画面上にテキストが表示されていて、
- そのテキスト上のさまざまな点を選択すると、その場所につながれている (リンクされている) 別の画面に移ることができ、
- そのようなリンクされた画面の集合体としてひとまとまりの情報 (ドキュメントないし文書) を構成している

ようなものだと考えればよい (図 10.1)。文字通り、Web (くもの巣) のようなからみ合った構造が任意に作れるわけである。なお、WWW では 1 つの画面のことを「ページ」と呼ぶ。また、ページやそれ以外のもの (サウンド、ムービー、絵など) を統合して「コンテンツ」(つまり「内容」) と呼ぶ。

ではハイパーテキストには普通の (紙の) 文書とくらべてどんな利点があるだろう?

- 最初から順番に読まなくても、必要なところだけたどっていくのが容易である。
- 計算機の機能を活用して、さまざまな支援ができる。
- 紙ではないので、画像や動画や音といったマルチメディアが取り入れられる。

最後の点であるが、単なる文字 (テキスト) でないことを強調する場合には「ハイパーメディア」と呼ぶ場合もある。

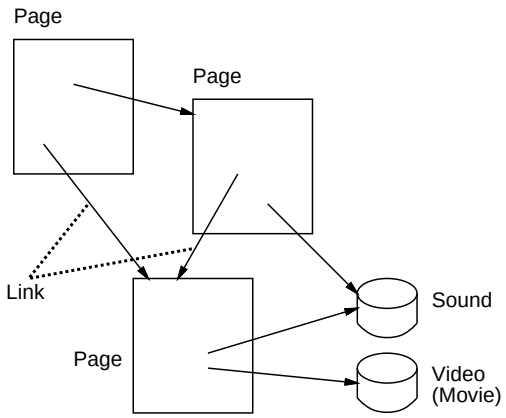

図 10.1: ハイパーテキストとハイパーメディア

10.1.2　World Wide Web とは?

実は、ハイパーテキストの例としては Windows のヘルプのようなものが代表的で、WWW はそれとはちょっと毛色が違っている。どこが違うか分かるだろうか? そもそも、なんで WWW が「インターネット」の代名詞になってしまったのだろうか?

- WWW では、リンクは「ネットワーク上の任意のページ」を指すことができ、従って世界中の情報源を行き先とすることができる (World Wide の意味)。
- さらに言えば、リンクはページだけでなく、「ネットワーク上のさまざまなモノ (資源)」を指すことができる。
- それらの資源の中には、単なる受動的なデータではなく、自らが能動的に動くようなものも含まれている。

WWW を最初に提案したのはスイスの CERN(粒子物理研究所) の研究者であり、現在はここが中心となって「WWW コンソーシアム」を発足させて、プロトコル、記法、記述言語などの標準化のとりまとめを行っている。[1]

難しい話は少し置いておき、上で挙げた特徴が実際にどのような形で現われているか、もう少し見て行くことにしよう。

10.1.3　リンクと URL

WWW では「さまざまなモノを指すポインタ」として URL(Uniform Resource Locator) と呼ばれる形式を使用している。URL の最も一般的な形式は

[1] 詳しくは `http://www.w3.org/` を参照されたい。

プロトコル:アドレス

の形をしていて「プロトコル」部で資源にアクセスする方式 (ネットワークプロトコル) を指定し、「アドレス」部でその資源のありかを一意的に指定する。たとえば資源にアクセスするのにメール転送プロトコルを使用する場合は

```
mailto:kuno@gssm.otsuka.tsukuba.ac.jp
```

のようになる (メールアドレスは宛先となる人を一意的に指定できますから)。

そして、WWW でもっとも一般的に使われるプロトコルは HTTP(HyperText Transfer Protocol) であり、その場合はアドレスはさらに「ホスト」と「ディレクトリ/ファイル」に分けられる:

```
http://ホスト指定/ディレクトリ/.../ディレクトリ/ファイル
```

この場合には、ディレクトリ以降は Unix のパス名とほとんど同様に見えるが、ただしディレクトリ階層の起点はルートディレクトリではなく、ドキュメントルートと呼ばれる特に設定されたディレクトリになっている。また、ユーザごとにそのユーザのページを個別に持てるような設定のシステムでは

```
http://ホスト指定/~ユーザ名/...
```

のように、最上位のディレクトリの代りに「~ユーザ名」を指定する。

10.1.4 クライアントサーバシステムとしての WWW

ネットワークアプリケーションの多くは「クライアントサーバシステム」の形を取る。これはシステム全体を次の 2 種類の構成要素に分けて構成するものである (図 10.2)。

- クライアント — 利用者の手元で動作し、利用者からの入力、利用者への出力をおもに扱う
- サーバ — 利用者から離れたところで動作し、ネットワーク経由で複数のクライアントからのサービス要求に応える

この方法だと、クライアント側は 1 人の利用者のことだけ考えればよく、一方サーバは利用者との細かなやり取りは扱わずに、複数のクライアントにサービスを提供することだけを考えればよいので、機能が切り分けやすく開発が容易だという特徴を持つ。また、サーバが管理する資源はクライアントからは直接アクセスされない (必ずサーバ経由となる) ので、資源の管理が容易で不整合も起きにくいという利点がある。

WWW の場合、クライアントはページを画面に表示するためのプログラムであり、「ブラウザ」と呼ばれる。ブラウザには非常に多くの種類があるが、代表的なものとしては次のものがある。

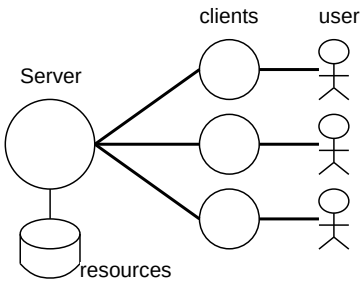

図 10.2: クライアントサーバシステム

- Mosaic — NCSA(イリノイ大のスーパコンピュータセンタ) で開発された。グラフィクスを取り入れた最初のブラウザ。1 年前くらいまでは圧倒的に主流だった。
- Netscape — Mosaic を作った人がスピンアウトして作った企業 Netscape Communications 社の商品。現在圧倒的に主流。
- Microsoft Internet Explorer(MSIE) — Windows95 に添付されているので、これから増えるか?
- Lynx — 文字ブラウザ。モデム経由での login などでも使えるので重宝する。
- w3.el — Mule にロードすると Mule がブラウザとして動作する。

我々のサイトでは上記のものを (MSIE を除いて) サポートしている。

サーバは HTTP プロトコルをサポートすることから「HTTP サーバ」「httpd」などと呼ばれる。[2]Unix では CERN が開発したサーバと NCSA が開発したサーバが多く使われている。Mac や Windows、WindowsNT などで動くサーバもある。我々のサイトではおもに NCSA のサーバ。

では、サーバとブラウザはどのように通信をするのだろう? HTTP による通信は極めて簡単で、ブラウザがサーバに言うことは「このパス名のファイルをください」だけ、サーバの応答はそのファイルの内容を返すだけ、である。たとえばブラウザの代りに telnet コマンドで httpd に接続して見よう。

```
% telnet smb 80  ← 80 は httpd の標準のポート番号
Trying ...
Connected ...
Escape character is ...
GET /           ←一番根本のページをおくれ
...             ←そのページの内容が送られてくる
```

[2]「d」は「デーモン」の略。悪魔、という意味ではなく、Unix ではずーっとバックグラウンドで動いていて、システムの下支えをしてくれるプログラムのことを「デーモン」と呼ぶ。

```
%                    ←これですべて完了
```

このように簡単にしておくことで、ブラウザはあちこちのサーバのページをロードしまくれるし、サーバは現在どこのブラウザが自分につながっているかをいちいち管理しなくても済むため高速なサービスが行え、大量のクライアントを扱える。

10.1.5 防火壁、proxy サーバ、キャッシュ

多くのサイトでは、それが管理するマシン群が直接インターネットからの攻撃に晒されることがないように、防火壁 (firewall) と呼ばれる機構を導入している。たとえば、我々のサイトでは utogw というマシンが防火壁ホストであり、このマシンと外部のホストの間でのみ直接通信が行える。

すると、各ユーザが自分のブラウザから外部のサーバに接続しようとしても、防火壁の機能により接続が行えない。これでは困るので、防火壁上に代理 (proxy) サーバと呼ばれるプログラムを実行させておく。代理サーバはクライアントから外部の URL を受け付けて、自分が外部のサーバに対してクライアントとなってページを取り寄せ、元のクライアントにそのページ内容を返すようになっている (図 10.3)。

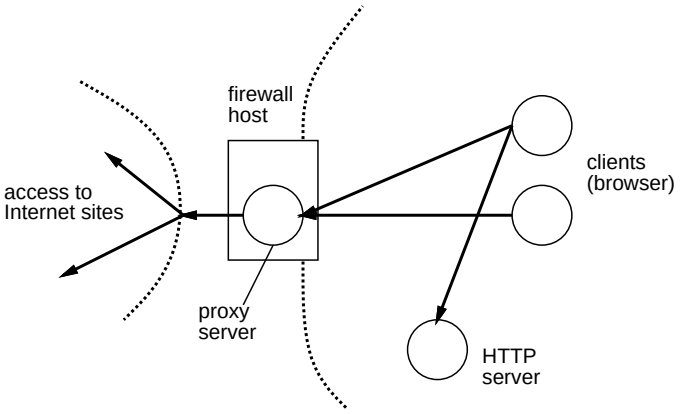

図 10.3: proxy サーバの概念

さらに、proxy サーバにキャッシュ機能を組み込むことがある。キャッシュ機能とは、一度取り寄せたページを proxy サーバが管理するディスク領域に保管しておくことで、これにより複数のクライアントが同じ外部のページを参照した場合に一度取り寄せたものを使い回すことでネットワーク帯域を効率良く使用し、またクライアントの応答も改善できる。

ただし、外部のサーバが提供するページが更新された場合、キャッシュの内容は元のままなので、何らかの対策を取らなければクライアントがいつま

でも古いページを見せられることになる。そこで、クライアントからページの要求が来たら外部のサーバにそのページの更新日付を問い合わせ、それがキャッシュにある内容より新しい場合には再取り寄せを行うといった処理が必要になる。

なお、ブラウザにも一度取り寄せたページをキャッシュを保持する機能を持つものが多い。しかし proxy サーバにキャッシュが搭載されている場合には手元のディスクに各自のキャッシュを持つのは無駄だから我々のサイトではキャッシュ機能を off にして使っていただいている。直接インターネットに接続されたホストでブラウザのみを使う場合には当然、ブラウザのキャッシュ機能を利用すべきである。

10.1.6　WWW のクロスプラットフォーム性

WWW のもう 1 つの長所として、クロスプラットフォーム性が挙げられる。つまり、Unix でも Windows でも Mac でもブラウザさえあれば同じようにページを参照することができる。これは WWW 以前にはほとんど考えられなかったことである。

また、代表的なブラウザは複数のプラットフォームで (ほぼ) 同一のものが稼働している一方、1 つのプラットフォームでも非常に多数のブラウザから使用するものを選ぶことができる。

ただし、この長所を享受するにはページを作る側でのそれなりの配慮が必要である。というのは、WWW のページは「どんなブラウザで見るか」によって全然見え方が違うし、同じブラウザでもマシンの画面やカラー能力などによる違いの影響も受ける。従って、どんなブラウザで見てもそれなりに役にたち、しかも高機能のブラウザではその機能を活用できるようなページを作成するのは易しくはない。

…と、お説教 (?) はこれくらいにして、以下ではとりあえず実習により自分のページを作ってみて頂こう。

10.1.7　Netscape の環境設定について

全然関係ないですが、これまで皆様に標準ブラウザとして Mosaic を推奨してきましたが、Java が使えず、表のタグがまともに動かないことと、印刷出力がイモなため、Netscape「も」併用しましょう。そのためには、次の設定を行ってください。

1. `smb` の窓を開くか、または `smb` に `rlogin` する。
2. コマンド「`gssmsetup netscape`」と「`gssmsetup x11`」を実行する。
3. `login` し直し、背景の中ボタンメニューで「`Netscape`」を選択。

そして、ページをプリンタに出力するには、次の設定を行います。

1．Print アイコンをクリックする。

2．ダイアログが出たら「Print Command:」は「/usr/ucb/lpr -Plw」(または-Plg、-Pln、-Pnz) とし、
「Grayscale」と「A4」を選択してから「Print」ボタンを押す。

3．2 回目からは前回の設定が保存されるので設定し直さなくてすむ。ただし Netscape を終了するとその設定は忘れられてしまう。

10.2 WWW ページの作成

では、皆様が見ている WWW のそれぞれのページはどのようにして用意されているのだろう? それは、HTTP サーバのあるマシンのしかるべきディレクトリに、そのページ内容を記述したファイルを置いてく、というのが答えである。ではそのファイルはどんな風に書けばいいのか? 具体的にどこに置くのか? という事柄を以下で見て行こう。

10.2.1 WWW ページの記述言語 HTML

WWW のページは、大きい文字の見出しやさまざまな書式指定で満ちているように見えるが、実は MS Word や一太郎を使って書式や字の大きさを指定するのとは全然違う方法で作られる。というのは、ブラウザによって使える色や文字の大きさはざまざま (1 色と 1 種類の大きさのことも!) なので、具体的な大きさや書式を指定するのはほとんど無意味だから。

ではどうするか? それには、テキストの中に「ここは第 1 レベルの見出し」「ここは箇条書」といった印 (タグ) を埋め込んでおき、実際に見出しや箇条書などをどのように表示するかはブラウザに任せる、というのが WWW で採用した方法である。つまり、ファイルの中にはページに現われるテキスト以外に特別な印 (タグ) が加わったものになっている。

テキストに様々なタグを埋め込んでそのテキストの論理的構造や付随情報を表す規格として SGML(Standerd Generalized Markup Language) と呼ばれるものがある。WWW では、これに準拠した比較的コンパクトな記述言語 HTML(HyperText Markup Language) によってページを記述する。では HTML とは具体的にどんな言語? それは実際に自分のホームページを作ってみながら体験するのが速そうである。

10.2.2 WWW の個人ホームページの作成

いよいよ自分の WWW ページを作っていただこう。我々のサイトでは、個人の情報ページは各自の「WWW」という名前のサブディレクトリに置くこ

とになっている。そして、そのディレクトリや内部のファイルは外部から読めるように保護モードを設定する必要がある。そこで次のようにする。

```
% cd                ←ホームディレクトリに行く
% chmod go+x .      ←ホームディレクトリの保護モードを変更
% mkdir WWW         ←サブディレクトリ WWW を作る
% chmod go+x WWW   ←保護モードを「WWW 中のファイルが触れる」に
% touch WWW/index.html  ←ホームページファイルを用意
% chmod go+r index.html ←保護モードを「誰でも見られる」に
```

この後、Mule を動かして「^X^F WWWindex.html/」で編集を開始する。

ではいよいよ、HTML による記述を行なう。HTML では上述の特別な印ないし「タグ」は

```
<タグ名 各種指定…>
<タグ名>....</タグ名>
```

という形をしている。例えばとりあえずおすすめの形として、次のように打って保存してみていただきたい。

```
<HTML>
<HEAD><TITLE>XXXX's HomePage</TITLE></HEAD>
<BODY>
<H1><IMG SRC="/icons/gssm.gif">XXXX's HomePage</H1>

<H2>ごあいさつ</H2>

<P>こんにちは。○○です、よろしく。</P>
</BODY>
</HTML>
```

HTML では 1 つのページ (<HTML>〜</HTML>) はヘッダ (<HEAD>〜</HEAD>) と本体 (<BODY>〜</BODY>) に分かれている。ヘッダの中にはタイトル (<TITLE>〜</TITLE>) が入る。

本体の中身は、基本的には「見出し」と「段落 (パラグラフ)」から成っている。見出しはその字の大きさによって「<H1>〜</H1>」「<H2>〜</H2>」など数種類ある。段落は「<P>〜<P>」の形をしている。段落は基本的に詰め合わせされるが、強制改行したければ「
」というタグを入れておけばよい。最後に、「<IMG SRC="…">」というのは、ここに絵を埋め込むという指定であり、上の例では GSSM のロゴを入れている。

これだけ入れたらファイルを保存し、Mosaic か Netscape を立ち上げて「GSSM のユーザのホームページ一覧」を開けると自分のページへのリンクが現われているはずである。さっそく眺めてみよう。気にいらないところが

あれば、Mule で修正して保存し (いちいち Mule を終わらせる必要はない)、「Reload」ボタンを使ってロードし直す。

10.2.3 よく使う HTML の記法

さて、次によく使うのは箇条書きである。この資料にもよく出てくる、各項目の先頭に黒丸 (●) が置かれた箇条書きは例えば次のように書けばできる。

```
<UL>
<LI>項目 1.....
<LI>項目 2.....
…
</UL>
```

ここで「UL」の代わりに「OL」とすれば番号つきリストになる。あと、通常は本文の内容は (段落も箇条書も) 画面の幅で詰め合わせされるが、プログラム例のように「詰め合わせず元のまま」にしたい場合には次のようにすればよい。

```
<PRE>
そのままになる部分
...
</PRE>
```

最後にハイパーテキストならではの「クリックすると別のページへ飛ぶ」をやろう。それには、「<A HREF="指定">…</A>」を使う。ここで「…」のところに書かれた部分が下線付きで表示され、そこをクリックすると「指定」に書かれたページへ飛ぶことになる。例えば自己紹介のページを作るためにまず

```
% touch WWW/selfintro.html
% chmod go+r WWW/selfintro.html
```

としてから Mule でこのファイルの内容を (やはり HTML で) 書いたとする。自分のホームページからここへ行けるようにするには

…また、<A HREF="selfintro.html">自己紹介のページ</A>もあります。

と書くと、画面上では「自己紹介のページ」が下線つきで表示され、そこをマウスで選択するとそのページへ行けるようになる。なお、この指定は一般に任意の URL でよいが、ホスト名を省略した場合には同じサーバのディレクトリを指し、さらに最初が「/」でない場合には現在見ているページの置かれているディレクトリを起点とした相対パス指定として解釈される。たとえば次の通り。

```
<A HREF="http://www.w3.org/pub/WWW/">...</A> ... 一般の URL
<A HREF="/~kuno/index.html">...</A> ... 同じサーバ内 (絶対パス)
<A HREF="../index.html">...</A>      ... 相対パス
<A HREF="selfintro.html">...</A>     ... 相対パス
```

なお、先にファイル名を「index.html」とつけたのは、NCSA サーバではディレクトリのみ指定してファイル名を指定しないとファイル名として自動的に「index.html」が使われるようになっているから (これを利用するとタイプする量を減らせる。意図としては、そのディレクトリに置くページの総まとめのページにこの名前を使う)。

10.3　対話的コンテンツ

ここまでに学んだところでは、WWW の各ページは (いかにイメージなどで飾ってあっても) あくまでただのファイルであり、従っていつ誰が見てもその内容は固定されている。しかし、少し使い込んでくると「ユーザの選択や応答に対応して、内容も変化するようなページが作りたい」という要求が出るのは自然なことである。このように、ユーザの応答によって内容が変化するようなコンテンツを「対話的コンテンツ」という。以下ではその代表として CGI、フォーム、Java について見てみよう。

10.3.1　CGI と CGI スクリプト

これまでに作成したリンクでは、HREF="..."の中に必ず「.html」という名前の (つまり HTML の) ファイルを指定していたが、ここに「.cgi」という名前のファイルを指定することもできる。CGI というのは Common Gateway Interface の略で、ブラウザからの要求に応じてサーバの中で実行させられるプログラムをいう。

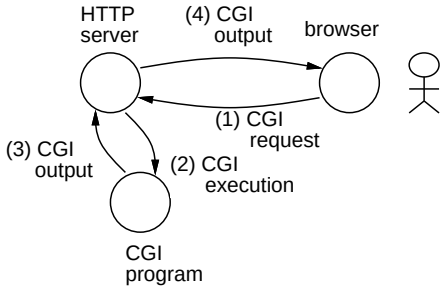

図 10.4: CGI の概念

その概略を示すと図 10.4 のようになる。

(1) まず、ブラウザで CGI を示すリンクを選択する。

(2) サーバは CGI プログラムを起動する。

(3) CGI プログラムは出力を生成する。

(4) CGI の出力は普通のファイルと同じようにブラウザに送られ、新しいページ内容として表示される。

CGI プログラムはどんな言語で書いてもいいのだが、Unix のコマンドを自由に起動できるところからシェルスクリプトを使うことが多い (ほかに Perl などもよく使われる)…おわかりですか? つまり、CGI を使えば任意のプログラムを実行させて、その出力をブラウザの画面に送り込むことができるわけ。

ではさっそく、「smb に誰が login しているか」を表示する CGI スクリプトを作ってみよう。まず、適当なページにリンクを用意することにして、HTML ファイルに次のような内容を追加する。

```
<A HREF="listwho.cgi">ログインしている人のリスト</A>
```

次に、この CGI スクリプト本体を作成する (WWW ディレクトリ、ないし先の HTML ファイルの置かれているディレクトリでやること)。

```
% touch listwho.cgi          ←ファイルを用意
% chmod ugo+rx listwho.cgi   ←誰でも読み出しと実行が可能に
```

次に Mule でこのファイルに次の内容を入れる

```
 1:  #!/bin/sh
 2:  PATH=/usr/local/bin:$PATH
 3:  echo "Content-type: text/html"
 4:  echo ""
 5:
 6:  cat <<EOF
 7:  <HTML>
 8:  <HEAD><TITLE>List Who Result</TITLE></HEAD>
 9:  <BODY>
10:  <H2>Who is logging in?</H2>
11:  <PRE>
12:  EOF
13:
14:  who
15:
16:  cat <<EOF
17:  </PRE></BODY></HTML>
18:  EOF
```

説明は次の通り:

- 1: `/bin/sh` スクリプトであることを示す。
- 2: `PATH` に`/usr/local/bin` を追加しておく (後で必要)。

- 3-4: CGI の出力が HTML であることを示すヘッダ行を生成。ヘッダの終わりの印である空行も生成。
- 6-12: HTML ページの前半を生成。「<<EOF」は、行頭に「EOF」が出て来るまでをコマンドの標準入力として読み込ませるような入力リダイレクション (ヒアドキュメントという) を指定している。だから以下に書かれた内容が cat によってそのまま出力される。
- 14: who コマンドが実行される。
- 16-18: HTML ページの残り部分を生成。

これが完成したら、先に作っておいたリンクを選択するとこの CGI スクリプトが起動され、who コマンドの出力がはさまった HTML ページがブラウザに送り返される。

10.3.2　フォーム入力と CGI

上の CGI はいつも決まったコマンドを実行するだけで、ユーザの入力に応じて何かをするというものではなかった。ユーザの入力に応じた動作を実現するにはどうしたらいいだろう? まず、ユーザが WWW ページに向かって情報を打ち込めるための仕掛けが必要ですね? 実はそれは「フォーム機能」として用意されている。すなわち、

```
<FORM METHOD="post" ACTION="xxxxxxx.cgi">
...
入力欄を含んだ内容
...
</FORM>
```

のような内容を書いておくと、入力欄の 1 つである「提出ボタン」をユーザが選択したとき、すべての入力欄の内容がまとめて CGI スクリプトに送り込まれるようになっている。入力欄としてはとりあえず次のものだけ挙げておく。

- <INPUT TYPE="text" NAME="フィールド名"> — 任意の文字を入力
- <INPUT TYPE="checkbox" NAME="ボタン名"> — ON/OFF できるボタン
- <INPUT TYPE="radio" NAME="グループ名" VALUE="文字列"> — グループ内で 1 つだけ選択できるボタン
- <INPUT TYPE="subimt"> — 提出ボタン

これらを使った例題ページの HTML を示す。

```
<HTML>
<HEAD><TITLE>Sample Form</TITLE></HEAD>
<BODY>
<H2>簡単な計算をします</H2>
<FORM METHOD="post" ACTION="formsam.cgi">
<P>
値 1: <INPUT TYPE="text" NAME="num1"><BR>
値 2: <INPUT TYPE="text" NAME="num2"><BR>
演算:
<INPUT TYPE="radio" NAME="op" VALUE="+">足し算、
<INPUT TYPE="radio" NAME="op" VALUE="-">引き算、
<INPUT TYPE="radio" NAME="op" VALUE="*">掛け算<BR>
ついでに日付を表示: <INPUT TYPE="checkbox" NAME="showdate"><BR>
<INPUT TYPE="submit"></P>
</BODY></HTML>
```

これをブラウザで表示させている様子を図 10.5 に示す。ここで適当な値、演

図 10.5: 例題フォームの画面表示

算、そして日付表示の有無を選択して「Submit」ボタンを押すと ACTION に指定された CGI が起動される。

ただし、今回は CGI にフォームのデータが送り込まれるので、それを処理してやる必要がある。実はこれは簡単で、

```
  eval ‘cgiparse -form‘
```

という行を実行すると入力欄ごとに「FORM_なんとか」というシェル変数に対応する値が格納されるので、あとはシェル変数を利用すればよい。なお、セキュリティ上の理由から CGI の中でシェル変数を参照するときは「{}」で囲むこと (以下の例のように):

```
#!/bin/sh
PATH=/usr/local/bin:$PATH
echo "Content-type: text/html"
echo ""
eval ‘cgiparse -form‘    ←フォーム入力の処理。「‘」はバッククォート。

VALUE=‘expr "${FORM_num1}" "${FORM_op}" "${FORM_num2}"‘  ←演算

cat <<EOF
<HTML>
<HEAD><TITLE>Form Sample Result</TITLE></HEAD>
<BODY>
<H2>Result of Simple Calculus</H2>

<P>${FORM_num1} ${FORM_op} ${FORM_num2} = ${VALUE}</P>
EOF

if [ x${FORM_showdate} = xon ]  ← sowdate が on かどうか?
then
  echo "<P>Date is: ‘date‘</P>"
fi

cat <<EOF
</PRE></BODY></HTML>
EOF
```

なお、ヒアドキュメント (<<EOF…EOF) の中でも"..."の中と同様、シェル変数の参照が行える。

10.3.3 CGI の限界と Java

このように、CGI を使えばユーザにブラウザ画面でさまざまな情報を記入してもらい、それを受け取って自由に処理することができる。ただし、CGI では常にブラウザ→サーバ→ CGI プログラム→サーバ→ブラウザという情報の流れが必要であり、とくにブラウザとサーバの間がネットワーク接続で結ばれている以上、緊密なフィードバックのようなことは不可能である。

これに対して現在売り出し中なのが「アプレット」である。アプレットとは、Java と呼ばれるプログラミング言語によって記述された対話的コンテンツであり、Java 対応ブラウザ (典型的には Netscape) の中にプログラムが読み込まれてブラウザ内で実行される。このため、通常のウィンドウシステムのプログラムと同レベルの応答性が保証できるという長所がある。

詳しくやると全然時間がないので、ここではごく簡単な Java アプレットの例を見てみる。

```
import java.awt.*;
import java.applet.*;

public class SampleLine extends Applet {
  int curx = 0, cury = 0;
  Color c1 = new Color(200, 200, 0);
  Color c2 = new Color(250, 0, 50);
  Color c3 = new Color(0, 100, 200);
  Color c0 = c1;
  public boolean mouseDown(Event e, int x, int y) {
    c0 = c1; curx = x; cury = y; repaint(); return true;
  }
  public boolean mouseDrag(Event e, int x, int y) {
    c0 = c2; curx = x; cury = y; repaint(); return true;
  }
  public boolean mouseUp(Event e, int x, int y) {
    c0 = c3; curx = x; cury = y; repaint(); return true;
  }
  public void paint(Graphics g) {
    g.setColor(c0); g.drawLine(0, 0, curx, cury);
  }
}
```

このアプレットの Java コードは、マウスボタンが押された時、押したままマウスポインタが移動した時、そしてマウスボタンが離された時にそれぞれマウスポインタの現在位置と特定の色を変数に設定して自身を再描画するとい

うだけのものである。このソースコードは必ず SampleLine.java という名前のファイルに入れなければならない。そして、Java コンパイラは特別な場所に置いてあるので (smb でしか動きません!)

```
PATH=$PATH:/usr/local/libproj/java/bin
```

を実行するか、よく使うなら上の行を .bashrc に追加する。その上で

```
% javac SampleLine.java
```

によりコンパイルすると SampleLine.class というファイルができる。さて、これを実行するにはアプレット入りのページを記述した HTML ファイルを用意する。アプレットをページに入れるには<APLET>...</APPLET>というタグを使う。例で示そう。

```
<HTML>
<HEAD><TITLE>Applet Sample</TITLE></HEAD>
<BODY>
<H2>簡単なアプレットです</H2>

<APPLET CODE="SampleLine.class" WIDTH=300 HEIGHT=200>
</APPLET>
</BODY></HTML>
```

これが動いている様子を見ると (図 10.6)、マウスの動きにほぼ遅れなく画面がついて来ることが分かる。

しかし、このようなことをするには実は克服しなければならない問題がものすごく沢山あった (現在でもすべて克服されたとは言いがたい)。

- プラットフォームに関係なく同一のコードを配って実行できなければならない。
- それでいて、それなりの十分な性能がなければ役に立たない。
- 「悪いアプレット」が配られて多くのユーザが被害を被ることを防止しなければならない。
- 「悪いアプレット」には、作成者以外の人が (通信途上で) コードを改ざんした場合も含まれる。

これらの詳しい説明は、もっとちゃんとした Java プログラミングの解説も含めて、いずれ (計算機プログラミングで?) 聞けると思いますのでよろしく。

10.4　演習

10-1. WWW でどんな内容でもいいから自分のホームページを作ってみよ。また、自己紹介のページを分けて作り、ホームページからリンクせよ。

図 10.6: 例題アプレットが動いているようす

10-2. 学外の気に入ったページを 3 つ以上選んで、自分のホームページ (か、またはそこからたどれる「おすすめページ集のページ」) からリンクを張れ。リンクの周辺に、どのような理由で推薦するかの説明を書くこと。

10-3. `listwho.cgi` をそのまま動かせ。次に、これを参考にして、リンクを選択すると何か役に立つ (固定的でない) 情報を表示する CGI を作れ。

10-4. フォームの例題をそのまま動かせ。次に、これを参考にして、何か面白いフォームのページを作れ。

10-5. Java の例題をそのまま動かせ。動いたら何でもいいからちょっと改造してみよ。

今回はすべての問題の中から任意に 2 つ以上選択して報告してください。では短い間 (?) でしたがご静聴 (?) ありがとうございました。