情報システム技術論'00 — ＃3

久野　靖*

2000.6.8

はじめに

どうも、資料＃1を2週間やっても終らなかったので…とりあえず、＃1の残りはなるべく簡単にしておいて、Javaのお話を追加することにしました。時間があればまた戻ってちゃんと説明しましょう。

1　CGIと動的コンテンツ

1.1　CGIとは

□ CGI(Common Gateway Interface)とは→Webサーバがクライアント(ブラウザ)の要求に応じてサーバ上にあるプログラムを呼び出して実行させ、その出力をクライアントに返送するような枠組み

□ なぜ必要か…「計算機内部での処理に応じて変化するコンテンツ」(動的コンテンツ)のためにはこれが必要

- 現在ではアプレットとか色々あるけど最初はこれしかなかった

□ CGIプログラム(以下単に「CGI」)の必須の条件 --- HTTP応答ヘッダの一部(通常はContent-type:ヘッダ)を返すこと

- ヘッダの終りは空行で表す(メール等と同じ)

1.2　ごく簡単なCGI

```
#!/bin/sh
echo 'Content-type: text/plain'
echo ''
date
```

□ これをdate.cgiという名前で用意して「chmod a+rx date.cgi」で実行可能に→サーバからアクセスしてみる

□ text/plainじゃなくtext/htmlという嘘を書いたらどうなるか?

□ ちゃんとHTMLを返すようにしてみよう

□ HTMLファイルからこのCGIへのリンクを貼って動かしてみよう

```
#!/bin/sh
PATH=$PATH:/usr/local/X11R6.3/bin; export PATH
echo 'Content-type: image/gif'
echo ''
ppmpat -anticamo 200 200 | ppmtogif
```

□ こっちだとどうか?　man ppmpatでマニュアルを見てppmpatのオプションを変更してみたらどうなるか?　再読み込みするとどうなるか?

□ たとえば上記がimage.cgiという名前だとして、スタイル指定「body　background-image: url(image.cgi)」とするとどうなるか?

2　フォームとCGI

2.1　フォームとは

□ HTMLの機能で、「記入欄」等の集まりを生成する機能

□ 記入した内容はCGIに送られ、CGIで好きなように処理

□ form要素→開始タグのaction属性でCGIを指定する

```
<form method="post" action="…">…</form>
```

2.2　フォームの入力部品

□ <input type="text" name="名前" size="文字数"> --- 入力欄

□ <input type="checkbox" name="名前" [checked]> --- チェックボックス

□ <inptu type="radio" name="名　前" value="値" [checked]> --- ラジオボタン

□ <input type="submit" value="ラベル"> --- 提出ボタン

*筑波大学大学院経営システム科学専攻

□ <input type="reset" value="ラベル"> --- リセットボタン

□ <select name="名 前">…</select> --- 選 択 メニュー (中に option 要素)

□ <option [value="値"] [selected]>…</option> --- 選択メニューの項目

□ <textarea name="名 前" rows="行 数" cols="文 字数">…</textarea> --- テキスト入力領域

2.3 簡単なフォームの例

□ たとえば「2 つの数値を入力して計算する」

```
<!DOCTYPE HTML PUBLIC
 "-//W3C//DTD HTML 4.0//EN">
<html lang="ja">
<head>
<title>A Sample</title>
</head>
<body>
<h1>Sample Form</h1>
<form method="post" action="sample8.cgi">
<p>
 <input type="text" name="n1" size="5">
 <select name="op">
  <option>+</option><option>-</option>
 </select>
 <input type="text" name="n2" size="5"><br>
 <input type="submit" value="計算">
 <input type="reset" value="リセット"></p>
</form>
</body>
</html>
```

2.4 Perl — CGI「にも」適したスクリプト言語

□ Perl --- Larry Wall が開発したスクリプト言語

- 「スクリプト言語」とは「ちょっとした処理をさっと書く」ための言語→Perl の場合、できが良いので結構大きなものまでこれで書かれている
- 言語の構文や概念はちょっと変わっているので慣れが必要
- version 5 でオブジェクト指向機能が加わった

□ Perl は CGI で多く必要な文字列処理を得意とする

□ CGI のためのモジュールが提供されている

2.5 Perl の CGI モジュール

□ フォームデータの受け取り等各種機能を提供

□ 使い方は次の通り簡単

- 冒頭部分での初期設定:

```
use CGI; $in = new CGI;
```

- フォームデータの参照 (「名前」はフォーム側で指定したもの):

```
$var = $in->param(’名前’);
```

2.6 簡単な Perl による CGI

□ 先のフォームを処理

```
#!/usr/local/bin/perl
use CGI; $in = new CGI;
print "Content-type: text/html\n\n";
$n1 = $in->param(’n1’);
$n2 = $in->param(’n2’);
$op = $in->param(’op’);
if($op eq ’+’) { $result = $n1 + $n2; }
else           { $result = $n1 - $n2; }
print ’<!DOCTYPE HTML PUBLIC’, "\n";
print ’  "-//W3C//DTD HTML 4.0//EN">’, "\n";
print "<head><title>sample</title></head>\n";
print "<body><h1>sample</h1>\n";
print "<p>$n1 $op $n2 = $result</p>\n";
print "</body></html>\n";
```

2.7 ファイルへの書き込み

□ 単純に計算して返すだけなら簡単

□ 実際には入力データを保存したい

□ ファイルの所有者は誰に? いくつかの選択肢

- Web サーバプロセス (nobody) が所有
- 自分が所有して「誰にでも書ける」に

□ 1 つの CGI の複数のインスタンスが同時実行される→排他制御が必要→ファイルにロックを掛ける機構を利用する

□ 簡単な BBS プログラム

```
#!/usr/local/bin/perl
require ’jcode.pl’;
use CGI;
$in = new CGI;
umask 022;
print "Content-type: text/html\n\n";
print ’<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN>’;
print "\n", ’<html lang="ja">, "\n";
print "<head><title>BBS</title></head><body>’, "\n";
print ’<h1>かんたん BBS</h1><hr>’, "\n";
```

```
if($in->request_method() eq 'POST') {
  if(!open(DATA, '>>bbs.data')) {
    print "<p>作業ファイルが書けない。</p>\n";
    print "</body></html>\n"; exit(); }
  while(!flock(DATA, 2+4)) { # LOCK_EX+LOCK_NB
    if(++$count > 5) {
      print "<p>ロックが獲得できない。\n";
      print "</p></body></html>\n"; exit(); }
    sleep(3); }
  if(open(SEQ, 'bbs.seq')) {
    $seqno = <SEQ>; close(SEQ);
  }
  ++$seqno;
  if(open(SEQ, '>bbs.seq')) {
    print SEQ "$seqno"; close(SEQ);
  }
  seek(DATA, 0, 2); select(DATA);
  $name = $in->param('name');
  jcode::convert(*name, 'euc');
  $face = $in->param('face');
  $text = $in->param('text');
  jcode::convert(*text, 'euc');
  $text =~ s/&/&amp;/g;
  $text =~ s/</&lt;/g;
  $text =~ s/>/&gt;/g;
  $time = gmtime();
  print "<h2 class=\"name\">No.$seqno\n";
  print "$face [ $name ] $time</h2>\n";
  print "<pre class=\"text\">\n$text</pre><hr>\n";
  flock(DATA, 8); close(DATA); select(STDOUT);
}
if(open(DATA, 'bbs.data')) {
  while(<DATA>) { print; }
}
print '<h2 class="write">書き込み</h2>', "\n";
print "<form method='post' action='sample9.cgi'>\n";
print '<p><select name="face">', "\n";
print "<option selected>^_^</option>\n";
print "<option>-_-</option>\n";
print "<option>;_;</option>\n";
print "<option>?_?</option></select>\n";
print "名前:\n";
print "<input type='text' name='name' size='12'>\n";
print "<input type='submit' value=' 書き込み'><br>\n";
print '<textarea name="text" rows="10" cols="30">', "\n";
print ' めっせえじ</textarea></p>', "\n";
print '</form></body></html>', "\n";
```

2.8 状態の保存

□ CGI では「ある実行」と「次の実行」は全く別のもの→何かを「記憶」しておきたければかなり工夫が必要

- たとえば「顧客番号」を割り当てて買物をするたびに何を買ったかという情報を蓄積し、最後にまとめて支払処理をしたい→それらの情報はどうする? CGI は個別に「これを買う」というごとにばらばらに起動される
- 情報をファイルに書けばよい? →複数人が同時に買物をしても全く不思議ではない→「誰が」という情報は別に用意しないと混乱する

□ 隠し入力欄を利用する方法

- <input type="hidden" name="名 前" value="値"> --- フォームの画面には何も現れないが値はデータとして送信されてくる→データを記憶しておくことに相当
- 画面がフォーム提出の連続でできていないといけない→不自由

□ クッキー機能 --- Netscape 独自拡張だったが現在では広く普及

- CGI からブラウザに「クッキー」と呼ばれる情報を送出→その CGI(ないし近辺) が呼ばれる時、同じクッキー情報が提出されてくる→それによってデータを記憶

2.9 クッキーを使う例

□ クッキー→「ちょっとした情報」という意味。WWW でのクッキーと言うのは、Netscape 社が独自拡張の 1 つとして始めたもので、便利なので多くのブラウザで実装されている。

- サーバはブラウザへの応答中に Set-cookie:応答ヘッダを含めることで、ブラウザにクッキー情報を記憶させることができる
- ブラウザはクッキーをそれが「どのドメインのどのパスから来たか」の情報を保存しておく
- ブラウザは URI にアクセスするとき、その URI にマッチするクッキーの情報を Cookie:ヘッダに入れてサーバに送る

□ これらの機構を使えば CGI が使う情報を「ブラウザに覚えておいてもらう」ようにできるので前述の問題が簡単になる

2.10 クッキーの制御

□ クッキーの形:

Set-cookie: 名 前=値; domain=ド メ イ ン; path=パス; expires=日時

- 「名前=値」がクッキーの本体

- ドメインはそのクッキーがどの範囲内のサーバで有効かを示す
- パスはクッキーがそのサーバ内のどの範囲で有効かを示す
- 日時はそのクッキーがいつまで「生きている」かを示す。指定しないとそのブラウザを終了させたときになくなる。

□ クッキーの制御方法:

- 「名前」と「パス」が違うクッキーは別のものとして扱われる
- expires が過去の日時だとクッキーは削除される
- ドメインには「.」が 3 個以上 (.edu 等では 2 個以上) 含まれる必要←クッキーを使って「顧客の嗜好調査」(悪くいえばプライバシーをあばく行為) を行う会社がある

2.11 クッキーを使った例題

□ よくあるパターン: 買物サイト

- 最初に顧客登録→クッキーを発行
- 商品を選ぶ→クッキーに基づいて商品情報を追加していく
- 最後に支払い→クッキーに基づいて支払、発送

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html lang="ja">
<head>
<title>Form Examples</title>
</head>
<body>
<h1>おかいもの</h1>
<form method="post" action="sample10a.cgi">
 <p>名前: <input type="text" name="name" size="20">
    <input type="submit" value="登録"></p>
</form>
<hr>
<form method="post" action="sample10b.cgi">
 <p>
 <input type="submit" name="goods" value="鉛筆"></p>
</form>
<form method="post" action="sample10b.cgi">
 <p>
 <input type="submit" name="goods" value="定規"></p>
</form>
<form method="post" action="sample10b.cgi">
 <p>
 <input type="submit" name="goods" value="手帳"></p>
</form>
<hr>
<form method="post" action="sample10c.cgi">
 <p><input type="submit" value="会計"></p>
</form>
</body>
</html>
```

```
#!/usr/local/bin/perl
use CGI;
$in = new CGI;
$name = $in->param('name');
if(!$name) {
  print "Content-type: text/plain\n\n";
  print "名前を入れてください\n"; exit;
}
$cookie = $in->cookie(-name=>"goods",
            -value=>"$name", -expires=>"+1h");
print "Content-type: text/html\n";
print "Set-cookie: $cookie\n\n";
print '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">';
print "\n", '<html lang="ja">', "\n";
print "<head><title>Registerd</title></head><body>\n";
print "<h1>登録完了</h1>\n";
print "<p>名前「$name」で登録しました。</p>\n";
print "<p><a href='sample10.html'>戻る</a></p>\n";
print "</body></html>\n";
```

```
#!/usr/local/bin/perl
use CGI;
$in = new CGI;
$goods = $in->cookie('goods');
$added = $in->param('goods');
$cookie = $ENV{'HTTP_COOKIE'};
if(!$goods) {
    print "Content-type: text/plain\n\n";
    print "最初に登録してください\n"; exit;
}
$goods = "$goods:$added";
$cookie = $in->cookie(-name=>"goods",
            -value=>"$goods", -expires=>"+1h");
@list = split(/:/, $goods);
$name = shift(@list);
$list = join(' ', @list);
print "Content-type: text/html\n";
print "Set-cookie: $cookie\n\n";
print '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"';
print "\n", '<html lang="ja">', "\n";
print "<head><title>Registerd</title></head><body>\n";
print "<h1>買物かごに追加</h1>\n";
print "<p>$name さんの選定品: $list</p>\n";
print "<p><a href='sample10.html'>戻る</a></p>\n";
print "<p>(生クッキー: $cookie)</p>\n";
print "</body></html>\n";
```

3 JavaScript

3.1 JavaScript とは

□ HTML の中に埋め込むことができるスクリプト言語

- もともとは Netscape 社が開発し LiveScript という名前だったが Java ブームのとき Sun から名前を買って JavaScript になった

□ できること→ HTML の一部をスクリプトで生成、フォームなどの内容をスクリプトで読み書きして処理、ステータスバーにメッセージを表示する (定期的に実行も可能) など

□ ブラウザ内で実行するので、サーバ内の情報は参照できないが、その代わり「ユーザと短い周期でのやりとり」が可能

3.2 JavaScript の特徴

□ プロトタイプ方式のオブジェクト指向言語

□ データは「値」か「配列」かその他の「オブジェクト」

□ オブジェクトに対してはメソッドが呼べる

- 「このページにあるすべてのフォームオブジェクトの配列」「このフォームにあるすべての部品の配列」などが利用可能→これらのメソッドを呼びながら動作できる

□ どうやって起動? →ロード時、提出時、…、「ボタン」部品が押された時

3.3 JavaScript による簡単な例題

□ 例: 最小公倍数の計算

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html lang="ja">
<head>
<title>JavaScript Examples</title>
<script type="text/javascript">
function keisan() {
  fieldx = document.forms[0].elements[0];
  fieldy = document.forms[0].elements[1];
  fieldz = document.forms[0].elements[2];
  x = parseInt(fieldx.value);
  y = parseInt(fieldy.value);
  while(x != y) {
    if(x > y) {
      x = x - y;
    } else {
      y = y - x;
    }
    fieldz.value = x;
  }
}
</script>
</head>
<body>
<h1>最大公約数</h1>
<form>
 <p>
  X: <input type="text" value="0" size="8"><br>
  Y: <input type="text" value="0" size="8"><br>
  最大公約数: <input type="text" value="0" size="8"><br>
  <input type="button"
         value="計算!" onclick="keisan()">
 </p>
</form>
</body>
</html>
```

4 XML

4.1 XML とは…

□ HTML ではどのようなタグがあるかはすべて HTML の枠内で決まっていた

- ユーザが自分独自のタグを定めることはできない
- SGML を直接使えばよいが、SGML 宣言を書くのは大変

□ そこで、「ユーザが自分でタグを定義できる」「SGML より簡単な」枠組み→ XML(eXtensive Markup Language)

4.2 XML 入門

□ 基本的な書き方:

```
<?xml version="1.0" encoding="iso-8859-1"
  standalone="yes"?>
<名前>
 …
</名前>
```

4.3 XML の特徴

□ HTML と違って「閉じタグは省略できない」「単独のタグは <img/>」のように末尾に「/」が来る

□ DTD はあってもよい (自動的にチェックできる) し、ないものも許される (必要に応じて柔軟に…)

□ ネームスペース機能→複数の違ったタグセットを混ぜられる

□ リンク機能→ HTML より柔軟な形態のリンク

4.4 XML は何のために使う?

□ 計算機によるデータ処理…さまざまな種類のデータを扱う。例: データベース、トランザクション、取り引き…

- これまではそれらのデータ形式は「適当に」決めていた

- XML ではタグを使ってこれらの形式を「共通に」しかし意味的には「各用途固有に」決められる

  <書籍><著者>久野</著者><題名>Java 言語入門</題名>
  <ISBN>...</ISBN><価格>…</価格></書籍>

□ それを見たり編集したり処理したりするのに、「XML 用ツール」が共通して使える

- つまりデータ処理業務の構築が「ネットスピード」に (???)

4.5 XML と HTML

□ 「HTML の次は XML」と言われているが… HTML と XML は用途が違う

□ ただし、XML の構文で HTML を定義したもの…XHTML

- XML の枠内で HTML と同じことが書ける
- ネームスペース機能を使えば「別のタグセット」(例: MathML による数式など) を混ぜられる

4.6 XML のこれから…

□ XML を処理するツールが多く出回るようになっている

- 一太郎 Ark for Java --- XML を出力
- その他のワープロソフト等でも?
- XML の作成や編集が容易に
- それを XML ツールをつかってデータ処理の対象に…
- まだどうなるかよく分からないところも…

5 WWW と Java

5.1 Java の由来

□ Java とは…「プログラミング言語」の名前。C とか Fortran とか COBOL とか…

□ どういう言語か?

- Sun Microsystems 社がセットトップボックス開発プロジェクトを 1991 に開始した
- そのボックスは「ネット経由でプログラムをダウンロードして動かす」ことで「何でもできる」ようになるはず
- 言語として C++を採用するはずだったが → C++では可搬性や頑健性や安全性に問題

□ →その結果、新しい言語を開発することに→これが Java 言語

5.2 可搬性 (portability) とは?

□ 多くのプラットフォームで同一のプログラムが動くこと

- ソースコードレベルでは割合実現しやすい (でも色々な問題…)
- コンパイル済みコードレベルではなかなか難しい

□ Java では「Java 仮想マシン」(JVM) を採用

- JVM のマシン語→バイトコードと呼ばれる→コンパイラはバイトコード生成
- JVM とは→バイトコードを実行するソフトウェア
- コンパイルしたバイトコードはどの環境へ持って行ってもそこに JVM があれば動かすことができる→可搬性がある

5.3 頑健性 (robustness) とは?

□ C や C++言語→ポインタデータ型がある→任意の番地をアクセスできる

- 間違ったポインタ捜査→ OS は破壊できない (メモリ保護機構がある) が、自分の実行環境内のすべてのデータは間違えば破壊してしまう可能性がある

□ より安全な言語→ポインタを直接操作できないようにしてある→コンパイラによって検査された「安全な処理」だけを行なえる

- 結構古くからある概念なのに C はそれを捨てている (根本的な問題) → C++もそれを C から引き継いでいる
- Java ではコンパイラによるチェックに加えて、バイトコードをシステムにロードするときにもポインタを操作していないことを検査している→危なくない、頑健である

5.4 安全性 (safety) とは?

□ マシン語のプログラム→どんな「悪い事」でもできてしまう

- 例: OS に頼んで「全部のファイルを消す」とか

- 例: 機密のファイルを盗み出してメールで送信するとか

□ OS にこのような操作を禁止する機能があればいいが… そういう OS は一般的でないしそれに依存すると可搬性がない

□ Java では JVM の機能の一環として「セキュリティサンドボックス」が実装されている→「信頼できないコード」はファイルの読み書きや任意ホストとのネットワーク通信を禁止

- これによって安全性を保証している
- cf. ActiveX → マシン語をダウンロードして動かす → 安全性の保証しようがない (そのコンポーネントのディジタル署名を見てどうするか決める以外に方法がない)

5.5 再度歴史に戻って…

□ こうして Java 言語は作られたが、Sun のプロジェクトは SGI に取られて撤収になった

□ しかしその時にわかに現われたのが WWW ブーム

- しかし初期の WWW ではページは「取り寄せた時の内容がそのまま見えるだけで変化しない」ものだった

□ Sun の一人が Java でブラウザを書いて見せた→そのブラウザでは Java の小さなプログラム (アプレット) を WWW 経由でダウンロードして動かすことができた

- 「動かないページ」に飽きていた人たちに熱狂をもたらした
- ちょうど発足した Netscape 社もブラウザに Java アプレットの機能を取り込むことにした
- にわかに Java ブームに

□ 結局、Java は汎用で普通の言語だったが、WWW ブーム +Java ブームのおかげて幸運にも市民権を得た。プログラミング言語の世界ではこういう幸運はめったにない。

6 Java 言語

6.1 Java 言語はどういう言語?

□ 可搬性、頑健、安全性→おもに実行環境の設計 (ただし頑健さについては言語仕様の部分も大きい)

□ オブジェクト指向言語→現在では大きく複雑なソフトウェアを効率よく開発するにはオブジェクト指向以外の選択肢はほぼ考えにくい

□ 構文は C や C++によく似ている→プログラマの抵抗を減らそうとした。構文はある意味どうでもよいとも言える (しかしどうでもよくはないとも言えるのだが…)。

□ 機能的には C++からややこしいものを削除して、より新しく必要度の高いものを入れた

- ヘッダファイルがなく、バイトコードファイルにコンパイル情報も格納
- インタフェース機能→現在のオブジェクト指向プログラミングでは重要

6.2 アプレットプログラミング

□ さっさとアプレットを見てみよう。

- オブジェクト指向言語→「クラス」という単位で機能を作成
- 「クラス」は「もの」を表す
- 「クラス」は他の「クラス」から機能を継承でき、必要な部分だけ差し替えることができる→ここでは paint() を差し替え

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

public class AppSam1 extends Applet {
  Font fn = new Font("Helvetica", Font.BOLD, 24);
  public void paint(Graphics g) {
    g.setFont(fn);
    g.setColor(new Color(100, 0, 255));
    g.drawString("Hello, World", 30, 30);
  }
}
```

□ これを「AppSam1.java」というファイルに打ち込む

□ 「javac AppSam1.java」でコンパイル。何も言われなければ OK

□ 次の内容を「AppSam1.html」に打ち込む (HTML はちょっと適当)

```
<html><head><title>Applet</title></head><body>
<h1>Applet Sample</h1>
<applet code="AppSam1.class" width="300" height="200">
</applet>
</body></html>
```

6.3 アダプタクラスとイベント処理

□ アプレット画面でのマウス等の操作→「イベント」が発生

- イベントを処理するオブジェクト→アダプタオブジェクト
- ここでは単にマウスのありかを保存して画面を再表示

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

public class AppSam2 extends Applet {
  Font fn = new Font("Helvetica", Font.BOLD, 24);
  int x = 30;
  int y = 30;
  public void init() {
    addMouseListener(new MyMouseAdapter());
  }
  public void paint(Graphics g) {
    g.setFont(fn);
    g.setColor(new Color(100, 0, 255));
    g.drawString("Hello, World", 30, 30);
  }
  class MyMouseAdapter extends MouseAdapter {
    public void mouseDragged(MouseEvent evt) {
      x = evt.getX(); y = evt.getY(); repaint();
    }
  }
}
```

6.4 アニメーション

□ アプレットの当初の用途の 1 つ→アニメーション

- Java のスレッド機能を使う (スレッドが標準機能なのも Java の特徴)
- 実際には定期的に画面を更新するだけ
- ローカルにブラウザ上で動くからこそ可能

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

public class AppSam3 extends Applet {
  Font fn = new Font("Helvetica", Font.BOLD, 24);
  int x = 30;
  int y = 30;
  boolean running = false;
  long time = 0l;
  public void init() {
    addMouseMotionListener(new MyMouseAdapter());
  }
  public void start() { (new MyTimerThread()).start(); }
  public void stop() { running = false; }
  public void paint(Graphics g) {
    double rad = 0.001 * (double)time;
    g.setFont(fn);
    g.setColor(new Color(100, 0, 255));
    g.drawString("Hello,", x, y);
    g.drawString("World",
      x+(int)(75*Math.cos(rad)), y+(int)(25*Math.sin(rad)));
  }
  class MyMouseAdapter extends MouseMotionAdapter {
    public void mouseDragged(MouseEvent evt) {
      x = evt.getX(); y = evt.getY(); repaint();
    }
  }
  class MyTimerThread extends Thread {
    public void run() {
      running = true;
      time = System.currentTimeMillis();
      while(running) {
        try { sleep(100); } catch(Exception e) { }
        time = System.currentTimeMillis(); repaint();
      }
    }
  }
}
```

6.5 そのほか…

□ 「言語の話」は時間の都合?聴衆の都合?で扱わない

- オブジェクト指向→クラス、継承、デザインパターン
- マルチスレッドプログラミング、排他制御、モニタ
- インタフェース機能→実装の継承と仕様の継承を分離
- RMI(Remote Method Invocation) →分散プログラミング

□ セキュリティ機能

- アプレットはファイルを読み書きできない
- アプレットは元のサーバ以外とネットワーク接続できない
- 署名+セキュリティポリシー設定で変更可能

6.6 サーブレット — サーバ側における Java 機能

□ WWW サーバ上に常駐

□ HTTP によりブラウザと通信

□ フォームの処理、HTML その他のデータの返送

□ CGI と違って毎回起動不要

6.7 WWW における Java の位置付け

□ プラットフォーム独立なソフトウェア開発

- □ ブラウザ内での対話的処理
- □ サーバ内でのコンポーネント単位の処理
- □ 分散プログラミング
- □ →今後ともさまざまな形で WWW 上での利用 + 汎用言語としての利用