コンピュータ言語の発展と応用分野
～筑波 EDP-entry: ビジネス戦略を支える IT 進展～

久野　靖*

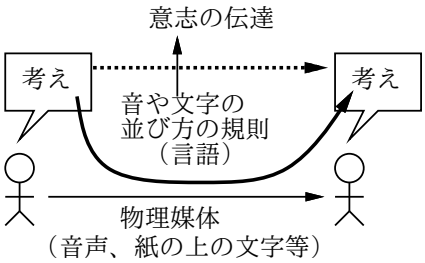
2007.3.5

1　コンピュータ言語

□ コンピュータ言語:　コンピュータで扱うために設計された人工言語

- 反対語:　自然言語 (人間が日常使う言語)…日本語、英語
- 注:　エスペラント (人間が日常使う言語だけど人工言語…)

□ なぜ「言語」が大切か?　→ 意思の疎通



- 並び方の「規則」(=言語) →その規則を利用することで、情報の流通や意思の疎通が可能になっている
- 言語以外の意思疎通手段もあるが、言語の効率、正確さにはかなわない
- (人類の歴史において「言語の発明」は重要な節目)

□ 「コンピュータ言語」の用途

- コンピュータと人間のやりとりも「文字情報」が中心 (効率、正確さ)
- 文字の並びにおける「規則」(=言語) により情報を伝達→コンピュータ言語
- ただし、主に「人間→コンピュータ」の方向で用いる。逆向きは自然言語とか、画像、動画、音などが使われる (←人間の柔軟性)
- 質問:　「コンピュータに、自然言語 (日本語) で指示を出せたらいいと思いますか?」

*筑波大学大学院経営システム科学専攻

□ 「人工言語」がいいのか?

- 曖昧さを排した簡潔な規則→コンピュータで曖昧さなく容易に処理、
- 自然言語処理は実用化されつつあるが、完全でない
- 自然言語は本質的に曖昧 (人間の性質) →「曖昧でないこと」が求められる場面には向かない
- 質問:「GUI(グラフィカルユーザインタフェース) の方が、言語 (文字の並び) よりも優れていると思いますか?」

□ 「言語」であることによる利点…

- 少数の語彙と規則だけで、ほとんど無限の意味を伝達可能 (小学校で習う漢字は 800 字とか)
- 「予め決めておかなかった意味」を組み合わせて作り出すことができる

□ もちろん「言語」であることの弱点もある

- 「文字」「語彙」「文法」「文の作り方、読解方法」を「学習」しなければならない。きちんと学習しないと「間違った文」ができてしまい意味が伝わらない
- これに対し、GUI であればメニューなどにより「語彙」が見えるので覚えなくてよい。いつ何が押せる/選べるかもコンピュータが規制してくれるので「間違った文」にはならない。その替わり、表現できる範囲は限られている
- 結局「何が優れている」とは一律に言えない「どういう目的で」「どういう状況で」を決めなければ評価できない

□ 1 節のまとめ

- 「文字の並び」の規則→言語
- コンピュータ言語→人間からコンピュータへの情報伝達に有用
- GUI など別の手段も普及しているが、言語には「多様性」「柔軟性」などの利点がある

2 さまざまな言語

□ コンピュータの歴史の上ではさまざまなコンピュータ言語が出現し、使われて来た

- 実際には「○○語」という名前はつけていないかも知れない。書き方の規則→言語だから
- 質問:「どんな用途のコンピュータ言語があると思いますか?」

□ プログラミング言語 --- コンピュータの動作内容 (=プログラム) を記述するための言語

- 初期のコンピュータ→「0」「1」の列を直接メモリに書き込み実行: 機械語
- CPU が実行する命令 (足すとか転送とか) に名前をつけて記述: アセンブリ言語
- より「人間に分かりやすい」記法、特定の CPU に依存しない記法:高水準言語
- 例: C、C++、Pascal、Basic、Java、…
- 質問:「なぜこんなに色々なプログラミング言語があるのだと思いますか?」

□ プログラミング言語がたくさんある理由…

- プログラミング言語に関する研究の進展→より新しい (利点を持った) 機構や記述方法の発明→それを採り入れた言語
- コンピュータの高度化→ソフトウェアも爆発的に複雑なものに→従来の言語では書くのがつらい→より大きなソフトが書ける言語への進化
- コンピュータの用途がさまざま→それぞれの用途に応じた (その使い方が記述しやすい) 言語の方が有利→用途ごとの言語
- プログラミング言語を作るのが趣味な人が沢山いるから?
- これらの理由は他の言語が色々ある理由としてもあてはまるが、プログラミング言語が圧倒的に色々あることは事実

□ マークアップ言語 --- 文書を記述するための言語

- 文書は「見出し」「段落」「図」など多くの要素から成っている→どの範囲が何であるかを「印」をつけて示す

```
<p>このように<em>強調</em>の範囲も印で表す。</p>
```

- 文書作成には TeX、Web ページ作成には HTML/XHTML が使われる
- これらの基本的考え:論理的マークアップ (「段落」「見出し」「強調」など文書の各部分の「意味」を指定
- 質問:「このような方法より Word みたいにメニューで指示する方がよい?」

□ マークアップ言語を使うことの利点…

- さまざまな環境に応じて自動的に表現を選択し整形できる。たとえば HTML は「ブラウザが整形して表示」するため、PC、携帯などさまざまな環境でそれなりに見えるようにできる
- 多様な処理の可能性。たとえば 200p くらいある文書について「『強調』という文字を全部赤字にしろ」と言われたとき、Word だと大変 (ですよね?)、HTML なら簡単
- ワープロソフトの場合は「A4 版」とかサイズが決まっているので直接文字サイズ等を指定も可能。しかし流用は大変。結局「何が一概によい」とは言えない

□ データ表現も「言語」と考える…

- 例: CSV…Excel 等のデータを「,」で区切って出力する機能による (セルの中に「,」が入っていると CSV ではどうなるか知ってますか?)
- 例: MML…MIDI による音楽のデータを記述
- その他、さまざまな場面で「決まった形で」データを保存する→言語

□ XML →汎用に使えるデータ表現言語 (形式)

- もともとは Web で HTML が非常に普及したことに発端
- HTML では特定のタグ群が決まっていて増やしたりできない (文書の記述のみに特化)
- データの種別に応じて自分でタグを定義できるようしたい→ XML (eXtensible Markup Language)
- たとえば書籍のデータとか…

```
<book><author>久野__靖</author>
      <title>入門 JavaScript</title>
      <isbn>4756138713</isbn></book>
```

- 質問:「なぜわざわざこんな長々しい形をわざわざ使うの?」

□ XML を使うことの利点/欠点

- 汎用性が高く、入れ子構造 (ある部分がさらに細かい構造を持つ) がそのまま扱える
- 処理するためのツール (ソフト、ライブラリ) が既にある→いちいち作らなくてもいい、生産性が高まる、安心

- このため今日では多くのデータ表現が XML に基づくものになってきている。HTML の XML 化→ XHTML
- 一方で、人間が直接読み書きするには長くて煩わしいという弱点もある→専用の言語やデータ表現が選択される場合も
- たとえばプログラミング言語の表現を XML にするというのはあまりない

□ 2 節のまとめ

- さまざまなコンピュータ言語がある
- プログラミング言語→プログラムを書く。例： C、Java
- マークアップ言語→文書にマークアップ。例： TeX、HTML
- データ表現言語→さまざまなデータの表現。例： XML

3 プログラミング言語の発展

□ プログラミング言語は多くの変遷を経て来ている

- 1960 頃→科学技術計算言語 (FORTRAN)、事務処理言語 (COBOL)
- 1975 頃→汎用言語 (Pascal、C)
- 1980 頃→オブジェクト指向言語のはじまり
- 1995 頃→実用的オブジェクト指向言語 (C++、Java)
- 同時期→ WWW の急速な普及、スクリプト言語 (Perl、JavaScript) の活用
- 現在→ Light Language(軽量言語)…スクリプト言語の発展概念 Perl、JavaScript に加えて、PHP、Python、Ruby、OCaml など

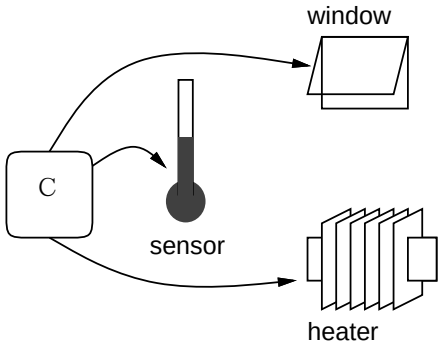
3.1 オブジェクト指向言語

□ オブジェクト指向とは? プログラム (ないし様々なソフトウェア) が扱う対象を自立した「もの」(オブジェクト) と見る「考え方」

□ オブジェクト指向が取り入れられている分野はいろいろある

- オブジェクト指向プログラミング
- オブジェクト指向ソフトウェア工学 (分析、設計、開発、UML…)
- オブジェクト指向ソフトウェア技術 (コンポーネント、分散オブジェクト、…)
- オブジェクト指向データベース
- ここでは「言語」を中心に扱う

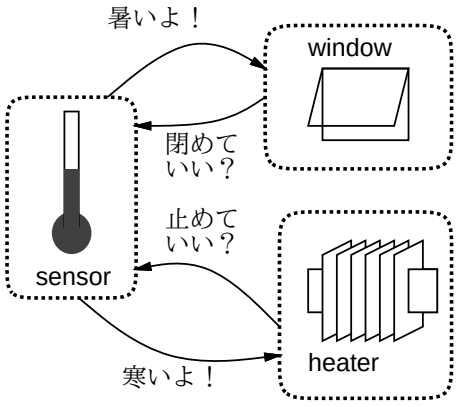
□ たとえば、「温室の温度調節システム」を考える



□ 旧来の考え方→手続き+受動的なデータ

- 「センサーを見て、気温が下がってきたらヒーターを通電するが、温度が上がりすぎたらヒーターを切る」「気温が上がってきたら窓を開くが、下がってきたら閉める」など機能中心に考える
- 制御する要素や条件が複雑になるとごちゃごちゃになりやすい

□ オブジェクト指向だと…



- オブジェクト指向→「気温センサ」「ヒーター」「窓開閉装置」などの「もの」を考える→「気温センサ」は温度が低いと「ヒーター」、高いと「窓」に注意を喚起→「ヒーター」は注意を喚起されると、定期的に「センサ」に温度を尋ね、一定以下だと通電、十分暖かいならヒーターを止めて仕事を終る→人間にとって考えやすく、適度な大きさに分けて考えられる

□ …で

- 「結局、従来と同じことをやってる」と思いますか?

- 「それは大変よさそうだ」と思いますか?

□ オブジェクト指向言語が持つ機能 (クラス方式の場合)

- クラス定義…「データ (インスタンス変数)」と「処理 (メソッド)」を一体としてパッケージ…抽象データ型の機能
- 継承機能…「あるクラスを土台として別のクラスを定義」これにより新しいクラスが簡単に作れる (差分プログラミング) が、クラス同士の関係が込み入るため弊害も指摘
- 動的分配…ある変数 x に対して処理 (メソッド) 呼び出しを行った場合、x に現在入っているオブジェクトに対応したメソッドが呼び出され実行される←現実世界とよく対応し有用な機能

□ 「クラス方式」以外に「プロトタイプ方式」もある (JavaScript など) ←スクリプト言語などに適した方式

□ 「オブジェクト指向」がよいかどうか…

- プログラムの動作そのものは同じことをさまざまに書ける。当然
- 人間にとって考えやすくさせてくれるならそれは「よいこと」
- 問題は「その分だけ余計な概念が増えて負担になる」ことを差し引いてトータルで儲かっているかどうか

□ 「オブジェクト指向」が主流となっている理由…

- ハードウェア性能の向上 (CPU 能力増大、メモリ容量の大規模化) →ソフトがいくらでも載せられる→今日のソフトウェアは大規模化
- このような複雑なコード→「オブジェクト」とその組み合わせという形でなければ実質作れない状況に

□ 単純に「開発する言語が変わった」以上の多様な手法

- クラスライブラリ→汎用性のある機能を提供する大量のクラスを用意→そこから機能を選んで呼び出すことで「書かずに済ます」
- コンポーネント→オブジェクトを「ソフトウェア部品」としてパッケージ→他のソフトウェアに「そのまま」組み込んで使える
- フレームワーク→自分が作成するオブジェクトをほぼできあがっているソフトウェアの「穴」に「はめる」ことでソフトが完成
- いずれも「できているものを活用」(書かない) ことで生産性を高めるが、その「できているもの」を熟知しないと使えないという敷居の高さもある

□ 3.1 節のまとめ

- オブジェクト指向…ソフトウェアが扱う対象を「もの」としてとらえる「考え方」
- オブジェクト指向言語…オブジェクト指向の考え方を素直に表現できるような設計のプログラミング言語
- ソフトウェアの大規模/複雑化→オブジェクト指向により「複雑な部分」をパッケージしてまとめるやり方が不可欠となった

3.2 スクリプト言語/軽量言語

□ 伝統的なプログラミング言語: 「ソース作成→コンパイル→組み立て (リンク) →実行」の手間が必要…重量級 (?)

- これに対し、「ソース作成→直ちに実行」のような「軽い」言語が便利と考えられるようになってきた
- 実際はそのような言語は古くからあった (BASIC、LISP 等) が、傍流とされてきた←実行速度が遅い等
- CPU が非常に高速になったため、それでも性能的に問題がなくなったことが普及の理由
- 質問:「どのような言語があると思いますか?」

□ Perl

- スクリプト言語の最初としては「シェルスクリプト」「バッチファイル」など「コマンドを並べておき順次実行」するものがあったが、言語としての機能は不十分だった
- これを受けて Rally Wall がスクリプトでありながらさまざまなプログラムを書くのに十分な機能を持った言語として開発→手軽で便利なため広く普及した (特に CGI…Web ページのデータ処理…のための言語として)
- 変数の冒頭に $、@などさまざまな記号を置くことを始めた
- 「1 つのことでもさまざまに便利な書き方ができる」特徴

□ JavaScript

- Web ページ記述 (HTML) の中に埋め込んで動作させることで、ページ内容を自在に操作できることをめざして作られた言語
- Netscape 社が開発→普及して事実上の標準→MS Internet Explorer などもに「互換な言語 (JScript)」を搭載している

- 今日では Web ページ内容をより動的にすること、またページを表示したままでサーバと通信して多様な機能を提供すること (Ajax) などで活用
- 詳しくはまた後で…

□ PHP

- Perl が CGI むけ言語として普及したが、Perl でページ内容を全部「print」で打つのは面倒
- このため「ほとんどは HTML で一部分だけプログラムを埋め込む」方法が考えられる→その先駆けが PHP 言語

```
<table border="2"><tbody>
<?php
  for($i = 0; $i < 10; ++$i) {
    echo "<tr><th>$i</th></tr>"
  }
?>
</tbody></table>
```

- さらに、多くの機能をライブラリとして提供
- 記述の便利さとあいまって、現在、Web ページの内容を (サーバ側で) 処理する言語として最も広く使われている

□ Python --- Perl より「きれいな」構文 (→書きやすさ) で普及してきているスクリプト言語。制御構造の範囲を「字下げ」で示すところが特徴。

□ Ruby --- 日本生まれのオブジェクト指向スクリプト言語で国際的にもファンは多い。Ruby で実装されている Rails という Web 向けフレームワークが 2006 年に人気を呼び、その利用のために Ruby を学ぶ人も増えた

□ OCaml (Objective Caml) --- 関数型言語 ML のオブジェクト指向拡張から来ている。型推論による強い型検査と関数型パラダイムにより実行時のエラーが少なくできるとされる。

□ スクリプト/軽量言語とオブジェクト指向…深い結び付き

- オブジェクト指向機能を活用することにより、スクリプト言語部分で大量に書かなくてもそれなりの機能を実現できる
- 既存のオブジェクトを活用するだけであれば、重量級の言語よりも手軽な言語の方が向いている
- 質問:「これらの普及により、ソフトウェア開発のあり方がどのように変化していると思いますか?」

□ 現在多く行われているソフトウェア開発→ Web ベースでのさまざまなシステムの構築

- 個々のページやそれが使う機能がある程度独立して動作→小さい案件を短時間で開発する必要に迫られることが多い
- このような場合に、開発言語としてもスクリプト言語を使い、短時間で設計から実装までをこなさざるを得なくなっている
- 時間が短いこと、言語が「簡便な」タイプであることは従来のソフトウェア開発と大きく違う→とまどいもある。開発方式も試行錯誤がある段階 (例: XP、アジャイル開発)
- ユーザ (発注) 側も、「まっとうな仕事」と「やっつけ」の違いが分からないという問題

□ 3.2 節のまとめ

- スクリプト言語/軽量言語の普及← CPU の能力向上、オブジェクト指向機能の普及などによりこれらの言語が実用になる
- 誰でも小規模な、しかしそれなりに役立つプログラムが書けるようになっている
- その一方で、そのような「アマチュアっぽい」環境/言語で「お仕事の」ソフトを開発しなければならないという変化にとまどいもある

4 Web 言語

□ Web 言語…という言葉があるわけではないが (久野の造語)、ここでは「WWW に関連するコンピュータ言語」程度の意味で使う

□ なぜ「WWW に関連…」か? ← WWW の影響力の大きさ

- ユーザ数の多さ: PC を使っている人ならほぼ確実に WWW のユーザ
- どんな新しいことでも」ブラウザ上で動く」ならそれで使える (インストールレス)

□ なぜ「…に関連するコンピュータ言語」か?

- 最初に述べたように「言語であれば色々応用できる」ため、またプログラミング言語であれば「動作を記述できる」ため

4.1 HTML+CSS

□ HTML(HyperText Markup Language) --- Web ページを記述するマークアップ言語

- Tim Berners-Lee (WWW を作った人) が生み出したもの (HTML 0.9?) から徐々に発展、最終版は HTML 4.01
- Tim はマークアップの形式 (「< ... >」というタグの形) を SGML から借りて来て使った。SGML では「どんなタグ」は規定しないので、段落、見出し、箇条書きなど Web ページに有用そうなタグを定義して HTML と名付けた。
- SGML(Standard Generalized Markup Language) は「さまざまな文書のマークアップに使うための標準形式」として制定されたが、ややこしくて使いにくいので低迷していた→ HTML で一躍脚光を浴びた (しかし今では XML が主流に)

□ HTML の変遷… HTML 0.9 → 1.0 → 2.0 → 3.2 → 4.0/4.01

- 最初は img タグ (画像) とかもなかった→必要のつど機能を拡張
- 2.0 くらいで一応の形が整う (Mosaic 時代)
- しかし Netscape ブラウザが「独自拡張」のタグを多く追加→それを「後追い」して HTML 3.2 ができた
- 3.2 は「表機能」「font タグ」等により表現力が大きくなった
- 質問:「ブラウザメーカが独自のタグを追加するのはいい?悪い?」

□ HTML の標準化… WWW コンソーシアム (http://www.w3.org/) 主導

- ブラウザ戦争 (Netscape vs MSIE) の時→「こっちのブラウザはこんなかっこいい機能がありますよ」と独自拡張
- 「ブラウザによって見え方が違う/見えない」などの問題が発生
- 現在では「標準に準拠する」(標準外の機能は使わない) でほとんどの制作者が合意 (個人でも作れるのでそこは野放しだが)
- 質問:「次の HTML は正しい? 正しくない?」

```
<p>たとえば
  <ul><li>Netscape</li><li>IE</li><ul>
  などが代表的なブラウザです。</p>
```

□ 「正しい HTML」の難しさ…

- p の中に ul は入れてはいけない (最初に定義したときにそうした)
- 世の中の人はいい加減に書いている人がかなり多いが、ブラウザは「エラーがあってもそれなりに表示」することになっているため、表示されてしまい間違いに気付かないまま
- チェックするツールがあるがあまり使われていなかったりする
- しかし正しくないと CSS や DOM(後述) の適用で問題→正しさを確認することは大切

□ HTML では (HTML 4.0 以降)「構造」と「表現」を分離

- 構造…「この範囲が段落」「この範囲が見出し」など、文章の意味に対応
- 表現…文字や地の色、大きさ、配置など「見え方」に関する指定すべて
- font タグ (文字のサイズや色を変更)、b タグ (太字) などは「表現」であるため、HTML 4.0 以降では非推奨になっている→ CSS の役割に
- しかし現在でも CSS を覚えないで font タグを使っている人は多数
- 質問:「HTML を書くときに font タグを使いますか?」

□ CSS(Cascading StyleSheet) … HTML において表現を指定する記法

- スタイルシートの基本的な考え方: 「段落はこう」「見出しはこう」のように HTML の要素ごとにその要素をどういう見え方にするか指定できる→ 1 つのスタイルシートで多数のページを統一した見え方にできる (いちいち表現指定のタグを書かなくてよくなる)
- さらに必要なら「この ID を持つ HTML 要素」「この class 指定を持つ HTML 要素」などの指定も可能だし、〜の内側の〜などの指定も可能→表現の指定対象は柔軟に選択できる
- 指定できるものも、色 (文字、背景)、柄、余白 (マージン、詰め物)、枠など非常に多様
- 質問:「HTML のフレーム機能を使いますか?」

□ Web ユーザビリティ…ページの「使いやすさ」という概念

- ヤコブ・ニールセンの「ウェブユーザビリティ」という本あたりから知られるようになった
- 例:フレーム機能 (1 つのページを複数の枠に分ける) →ユーザビリティ上よくないとされる…特定の構成でブックマーク/リンクできない
- 現在では多くのサイトが CSS で表現指定したページを作っている (フレームは 2 ちゃんねるなど特殊な用途に限定されるようになっている)

□ 4.1 節まとめ

- HTML は SGML を元にした言語 (記法) で、変遷してきている。現在の HTML 4.01 が最終版
- 標準からの逸脱がいつも問題に→検証ツールを使った正しい HTML が望まれる
- HTML では構造と表現を分離→表現は CSS で指定する
- Web ページのユーザビリティが重視されるようになっている

4.2 HTML+CSS 以外のコンテンツ

□ Web のコンテンツは HTML+CSS が基本だが、それ以外にも多様なものが存在している

- マルチメディア…画像、音、動画。画像は GIF、JPEG、PNG のみブラウザが直接サポートしページ内に入れられる (IE だと BMP も)
- ブラウザが直接サポートしないものは「ヘルパーアプリケーション」(ダウンロード後自動的にソフトが起動) または「プラグイン」(同様だがブラウザの窓の中で実行される) による。例: 音、動画、PDF
- ヘルパー/プラグインは標準で入っていれば手間はいらないが、変わったものだとユーザが自前でインストール必要。Web 経由でできることが多いが、それでも抵抗が大きい
- 質問:「XHTMLって使っていますか? 使うとしたらなぜ?」

□ XHTML…HTML を XML の構文に合わせて定義し直したもの

- 機能的には HTML と同様 (だったら何で使うの?)
- XML のツールで扱うことができる
- XML には namespace 機能があり、複数の XML ベースの言語を一緒に入れられる→ XHTML のページの中にそのようなものを埋め込めるというのも利点
- Mozilla Filrefox などではいくつかの XML 言語もサポート→それらのコンテンツが埋め込める
- MSIE も追従してくれれば普及するのだけど…

□ MathML (Mathematical Markup Language)

- 数式を記述するためのマークアップ言語
- HTML では数式は「画像」で埋め込むしかなかったが、それはあまり嬉しくない (サイズが固定など)
- MathML で数式を記述しておけば、文字サイズ変更につれて大きさが変化する。

```
<?xml version="1.0" encoding="iso-2022-jp"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="jp">
<head><title>test</title></head>
<body>
<p>これはテストです。</p>
<p>たとえば MathML で
<math xmlns="http://www.w3.org/1998/Math/MathML"
      xmlns:html="http://www.w3.org/1999/xhtml" xml:lang="j
  <mfrac>
    <mi><html:a href="../index.html">x</html:a></mi>
    <mrow><msup><mi>x</mi><mn>2</mn></msup><mo>+</mo><mn>1<
  </mfrac>
</math>
なんていうのも書けます。</p>
</body>
</html>
```

- MathML 自体は「形」だけでなく「計算式」の機能も持っている→式の変数に値を入れて計算してみる、グラフにするなどの用途も可能 (ブラウザには計算機能はないが)

□ SVG (Scalable Vector Graphics)

- グラフィクスにはピクセルグラフィクス (画像、細かい点の集まりで表現) とベクターグラフィクス (図形、輪郭などの制御点とそれらを結ぶ曲線/直線の式で表現) がある
- ピクセルグラフィクス→通常の画像
- ベクターグラフィクス→ SVG。図形は「無限に伸びる針金の図形にスクリーントーンが貼られている」ものと思えばよい→個別に回転、移動などが自由にできる。

```
<?xml version="1.0" encoding="iso-2022-jp"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="jp">
<head><title>test</title></head>
<script type="text/javascript">
var s0, e0, x = 100, v = 3; r = 0, d = 3;
function init() {
  s0 = document.getElementById('s0');
  s0.onclick = function() { v += 1; };
  e0 = document.getElementById('e0');
  e0.onclick = function() { d = -1.1*d; };
  setInterval(step, 50);
}
function step() {
  x = (x+v)%450; s0.setAttribute('x', x);
  r += d; e0.setAttribute('transform', 'translate(300,75) r
}
</script>
<body onload="init()">
<p>SVG の例題</p>
<svg width="400" height="150" xmlns="http://www.w3.org/2000
  <g fill-opacity="0.3">
    <rect id="s0" x="100" y="20" width="50" height="50" fil
```

```
    <ellipse id="e0" cx="0" cy="0" rx="70" ry="40" fill="red" />
  </g>
</svg>
</body>
</html>
```

- MSIE は独自拡張の「VML」言語をサポート (標準規格である SVG に合流して欲しいのだけど…)

□ 4.2 節のまとめ

- HTML+CSS 以外のコンテンツとして、マルチメディアと XML 言語とがある
- ブラウザが直接サポートしないものはヘルパーアプリケーションやプラグインで対応
- XML 言語は XHTML の中に埋め込んで活用可能。MathML や SVG など

4.3 サーバ側プログラミング

□ Web における「プログラムの出番」は「サーバ側」「クライアント側」に大別される

- サーバ側 --- Web サーバの上でプログラムが動作するもの
- クライアント側 --- Web クライアント (ブラウザ) 上でプログラムが動作するもの
- 加えて、最近では両者を連携させるもの (Ajax) が流行し普及したがそれは少し後で
- サーバ側とクライアント側でできること (得意とすること)、できないこと (不得手なこと) が明確に決まっている
- 質問:「Web サーバの仕事とは要するに何をすること?」

□ サーバ側プログラミングの原理…

- Web サーバは「階層構造のディレクトリに HTML ファイルを格納していて URL で指定されたファイルを返送する」というイメージを持たれていることが多い
- しかし原理的には、「http://サーバ/パス…」のパス部分をどう解釈するかはサーバの勝手。サーバの仕事は「提示された URL に対して対応するコンテンツを返送する」こと
- 実際には階層ディレクトリと URL のパス構造が対応していることが多いが、たとえば Apache の場合…
- だとっていて「ファイル」があるとそのファイルを返送
- たとえて「プログラム」があるとそのプログラムを実行し、結果を返送する:

  http://www.ex.com/a/b/PGM/c/d

□ CGI(Common Gateway Interface)

- ブラウザが URL を提示しサーバに接続→URL がプログラムの場合、サーバがプログラムを呼び出す→プログラムが結果を返送→結果をサーバが読み取ってクライアントに返送→ブラウザが結果を表示
- このとき、サーバとプログラムの間でデータをやりとりする約束ごとが CGI
- しかし単に CGI といった場合、CGI に従って動作するプログラムのことを指すことが多い

□ ブラウザ/サーバ/CGI プログラム/間の情報のやりとり

- サーバからの情報伝達→環境変数 (プログラム内から getenv() で読み出せる)
- ブラウザからの情報伝達→ HTML のフォーム (入力部品) のデータ→ get または post で送信
- get の場合、URI の末尾に「?」に続けて指定→△見えてしまう、○ブックマークなどにそのまま登録可能
- post の場合、URI とは別に送信し、CGI では標準入力から読み込み→見えない、大量でも平気
- プログラムからの情報→標準出力に「ヘッダ+本体」の順に出力。ヘッダはサーバが受け取り処理、本体はブラウザまで返送

□ サーバ側プログラミングの得失

- 得意とすること…サーバ上のリソースにアクセス。たとえば DB やファイルに情報を格納することはサーバ側でないとできない。
- たとえば複数のユーザ (ブラウザ) 間で情報を共有したいなら、それをサーバ上に格納しておく以外の選択肢はない
- 不得手なこと…高頻度の応答。ブラウザとサーバの間にはインターネットがあり、また CGI などのオーバヘッドもあるので短時間での応答は難しい (データ量の大きさはネットの高速化でだいぶ問題にならなくなってきたが)
- また、従来のパラダイムでは「送信ボタン→サーバへ伝達→ CGI プログラムの処理→結果返送→表示」となるため、沢山情報をセットしてまとめて送信し、結果をゆっくり待つという「古典的」スタイルになってしまう (Ajax はこれを緩和)

□ CGI に使われる言語の変遷… C → shell script → Perl → PHP

- C で書くのはサーバが UNIX ベースだったから。しかし面倒なだけであまりいいことはない
- shell の場合、中から他のプログラムを呼び出してデータを加工するのが便利だが、多数のプログラムの組み合わせでごちゃごちゃになりやすく、さらにパッケージしにくい (よそに持って行くと呼び出すプログラムが無かったり動作が違ったりする)
- Perl は文字列加工を得意とするので HTML を組み立てるのには向いており、それ単独で多様な作業ができるので普及した (今でも多く使われている)。ただし HTML を沢山生成するときそれを文字列リテラルやヒアドキュメントにしないといけないのが不便

```
print <<EOF
<tr><td>$i</td></tr>
EOF
```

- PHP は HTML の中に埋め込む形で書けることと、Web のサーバ側プログラミングで使うような機能のライブラリを充実させていて「便利で取り付きやすい」ことで普及

□ サーバ側プログラミングの注意点…セキュリティ

- サーバ側プログラムは重要な共有リソース (DB 等) にアクセス…もともとそのためのものなので必須
- しかし相手にする「データ」はブラウザ (他人の持ちもの) から送られて来る→その他人がどんな悪意を持っていても安全でなければならない
- 例: SQL インジェクション…フォーム入力値が送信されてきて、それを元に DB 問い合わせ SQL 文を「連結」で組み立てるが、入力値に「問い合わせ以外の悪意ある操作」が連結されていてそれを実行してしまう
- 対策→「;」など制御文字をエスケープ/削除する (△)、ストアドプロシジャ (SQL 文は予め SQL 処理系に格納しておき、パラメタ文字列のみ渡す) の使用 (○)

□ 4.3 節のまとめ

- Web プログラミングはサーバ側とクライアント側に大別
- サーバ側プログラムは CGI インタフェースに従い動作
- サーバ上の共有データにアクセスできるが、細かい応答には向いていない
- 言語はざまざまに変遷してきているが、今日では PHP が多い。Perl も多く使われている
- 共有データを扱うので、セキュリティ対策が重要

4.4 クライアント側プログラミング

□ クライアント側プログラミングの原理…

- Web ではサーバ上にコンテンツが置かれている (アタリマエ)。この一部としてプログラム (ソース、バイナリ) を用意
- このプログラムが他のコンテンツと同様にブラウザ側に転送されてきて、そこで動く
- ということは、ブラウザ側に「そのプログラム言語の」言語処理系 (実行系) が存在している必要がある

□ ブラウザ側に言語処理系がある→大きな制約

- ケース 1: ブラウザに最初から言語処理系が組み込まれている…JavaScript と、MSIE の VBScript だけ
- ケース 2: 「プラグイン」(HTML 画面の四角い一部領域をその言語の表示用に割り当てて使わせる…ブラウザ側で最初から用意しているのは Java(アプレット)、Flash にほぼ限定
- ケース 3: プラグインの自動設置…ユーザに「これを設置してください」というのは結構な障壁
- ケース 4: プラグインではなく別のコードとして独自実行…JSF など…処理系を設置してもらうのは非常に困難

□ クライアント側プログラミング言語…

- JavaScript: 広く普及。Ajax などでも使われる事実上の標準
- VBScript: MS は普及させたかったと思うがジリ貧。たまに JavaScript ではできないが VBScript だとできることがあるのでそういう時に使うかも (ただし IE 専になるが)
- Java Applet: Java ブームの立役者だが今日ではマイナーになっている。ただし Java はさまざまなことができる汎用言語なので「インストールレスで動くアプリ」を提供するとためには使われることがある
- Flash: 多様なグラフィクス表現が容易に実現可能なため、広く普及。「Flash アニメーション」サイト多数
- 質問:「処理系の設置以外にも大きな制約がある。何だか分かりますか? またそれはなぜ?」

□ 最大の制約:「セキュリティ制約」

- ファイルは一切読み書きできない
- ネットワーク接続は「そのコードが置かれていた元のサーバとの間でだけ」行うことができる
- ウィンドウを作る場合は「ブラウザの窓」であるか「アプレットの窓」などと明示し、小さい窓は作れない
- 質問:「なぜこれらの制約があるか分かりますか?」

□ なぜこれらのセキュリティ制約があるか?

- クライアント側プログラミングのコードは「読み手がページを開くと」「自動的に手元のマシンに転送されてきて」「そこで動く」。この動いたプログラムがもしも…
- 「ユーザの機密情報を探って報告する」とか「ファイルを破壊しまくる」ものだったら? →ファイルアクセスの禁止
- 「あちこちにイタズラメールを送る」ものだったら? →元サーバ以外へのアクセス禁止
- 「ログインウィンドウのふりをして ID とパスワードを盗む」「ずっと画面の隅に存在していてユーザの行動を記録する」ものだったら? →特別な窓であることを明示、大きさも 1x1 みたいに小さくはできない
- 質問:「そんなに制約が強いのにあえて使うのはなぜ?」

□ これらの制約にも関わらず使われる理由

- ユーザインタフェースの拡張/改善 … HTML の標準の機能だけでは受動的な入力部品しかない。それを入れ換えたり動作を自動的にサポートしたりできる… JavaScript(Ajax もこの延長)
- 対話的グラフィクス … HTML+マルチメディアコンテンツでは単に絵や動画が見られるだけだが、それを自在に動かしたりユーザ入力に反応させたりできる…Flash
- インストールしなくてもアプリケーションが動かせる…Java Applet(データの保存はサーバ側に通信して行う)

□ 4.4 節のまとめ

- クライアント側プログラミングはコードがサーバからブラウザ側に転送されてきて実行 (モバイルコード)
- 従って強いセキュリティ制約が避けられない
- その一方で静的な Web コンテンツを動的かつ対話的なものとできる可能性→広く使われるようになっている

4.5 JavaScript

□ JavaScript はブラウザに直接組み込まれて実行されるという点でプラグインベースの Flash、Java Applet とは違う

- プラグインベースだと「四角い領域」が設定され、その中だけがその言語の「領分」になる
- JavaScript ではブラウザがやることすべてが JavaScript によって制御可能な領分 (ただしセキュリティ制約があることは前述の通り)
- これにより、さまざまな応用の可能性 (Ajax を含む)

□ JavaScript コードは「いつ」実行されるか?

- script タグの内側に実行文を書く→そのタグが読み込まれつつある時点で実行される
- その場合、その中に document.write() があるとそこで出力したものは script タグの位置に挿入される

```
<script type=text/javascript>
document.write(Math.random());
</script>
```

- マウスクリックなどイベントがあった時に実行されるように HTML 側や JavaScript コード側で指定 (イベントハンドラ) →そのイベント生起時に実行される。

```
<span onmouseover="alert(Math.random())">Here</span>
```

- 普通は常時実行される部分で初期設定や関数定義を行い、イベントハンドラで動作を開始させる

□ フォーム部品 (GUI 部品) の読み/書き

- 古くからある機能。今日でも、ユーザが入力したものをまずその場でチェックしてから CGI に送信する形でよく使われる (ただし JavaScript でチェックしても、CGI 側でも再度チェック必要)

```
<script type=text/javascript>
function calc() {
  var x = parseFloat(
    document.getElementById('t0').value);
  var y = parseFloat(
    document.getElementById('t1').value);
  document.getElementById('t2').value = x+y;
}
</script>
<input id="t0" size="3" value="1">+
<input id="t1" size="3" value="1">=
<input id="t2" size="3" readonly>
<button onclick="calc()">Calc</button>
```

□ DOM (Document Object Model) --- HTML や XML のドキュメントをツリー構造に対応させ (モデル化)、そのツリー構造にアクセス (参照、改変) する API(呼び出しインタフェース) を規定

- W3C による標準。DOM level 2(DOM2) が主流。Core、HTML、CSS、Event、Range/Traversal などに分かれている (getElementById() は Core に含まれている)
- たとえば DOM2 CSS を使うとスタイルを JavaScript 側で設定できる

```
<script type=text/javascript>
function f1() {
  document.body.style.backgroundColor =
    'rgb(255,200,'
      + Math.floor(255*Math.random()) + ')';
}
function f2() {
  document.body.margin =
    Math.floor(50*Math.random()) + 'px';
}
</script>
<button onclick="f1()">F1</button>
<button onclick="f2()">F2</button>
```

□ DOM2 HTML --- HTML のあらゆる要素やその属性を JavaScript 側から変更することができる。

```
<ol id="o1"><li>リストです。</li></ol>
<button onclick="f1()">F1</button>
<button onclick="f2()">F2</button>
<script type="text/javascript">
function f1() {
  var e = document.getElementById('o1');
  var s = Math.random() + ' です。';
  var t = document.createTextNode(s);
  var l = document.createElement('li');
  l.appendChild(t); e.appendChild(l);
}
function f2() {
  var b = document.body;
  var e = b.firstChild;
  b.removeChild(e); b.appendChild(e);
}
</script>
```

□ innerHTML --- すべての HTML 要素オブジェクトについて、その内側にある HTML コードを参照/設定 (!!) 可能

- DOM2 HTML でやるより簡単なのでよく使われる。MSIE が始めたものだが便利なので Mozilla など他のブラウザも追従

```
<input id="t0"><input id="t1">
<button onclick="f1()">F1</button>
<button onclick="f2()">F2</button>
<script type="text/javascript">
function f1() { alert(document.body.innerHTML); }
function f2() {
  var x = document.getElementById('t0').value;
  var y = document.getElementById('t1').value;
  document.body.innerHTML =
    document.body.innerHTML.replaceAll(
      new RegExp(x, 'g'), y);
}
</script>
```

□ JavaScript のセキュリティ制約

- 前述の「ファイルアクセス不可」「ネット接続は元サーバだけ」以外にも固有のセキュリティ制約
- history オブジェクト: ブラウザのヒストリが格納されているが、その内容は JavaScript からは読めない。「いくつ前へ戻る」という形でだけ利用できる
- location オブジェクト: 窓やフレームごとに存在し、そこに表示されているコンテンツの URI が格納されているが、JavaScript からは「同じサーバのコンテンツの時」だけ内容が読み出せる。設定する (指定した URI に表示を切替える) ことはいつでも可
- DOM によるページ内容アクセスも別窓/フレームに対して実行できるがこれも「同じサーバのコンテンツの時」だけ
- 質問:「なぜこれらの制約があるのでしょうか?」

□ JavaScript のセキュリティ制約の理由…

- history が全部見えてしまうと、ページが表示された時に「過去の履歴を調べて収集」するようなページが作れてしまう→個人情報の漏出
- location が見えてしまうと、定期的に別の窓の表示 URI を監視することで history と同様のことができてしまう
- ページ内容が操作できてしまうとさらに恐ろしいことに (他サイトのページ内容の書き換え、ショッピングサイトに入力したクレジットカード情報の盗み見、…)
- 実は「サーバが同じ」だけでなく「ディレクトリが同じ」まで必要 (レンタルサーバで複数人が使用することがあるから)。さらにディレクトリを短くできる。サーバも複数サーバで運用することがあるからドメイン名を一定まで短くできる。これらのチェックにバグがあるとセキュリティホールに

□ クロスサイトスクリプティング (XSS)

- これだけ厳重に制約で守ってもサイト設計によっては台無しに…
- 具体的には、掲示板など「読み手が内容が書き込める」サイトで「HTML 可」のときに script タグを書き込むことで自分の JavaScript コードを埋め込む
- その書き込みを見た人は知らないうちに (サイト作成者ではない) 他人が用意した JavaScirpt コードを実行してしまう→そのサイトに対して打ち込んだ情報なのに他人に漏れてしまう

- 対策は scipt タグが埋められないように書き込み内容をチェックすること (各サイトの対策であり読み手には危ないサイトを使わないことくらいしかない)

□ 4.5 節のまとめ

- JavaScript はクライアント側プログラミングの中でもブラウザに密着しているという特別な立場
- DOM2、innerHTML などによりページ内容を自由に操作可能
- 複数の窓/フレームにまたがった操作も可能だが、それに関連してセキュリティ制約がある
- セキュリティホール、クロスサイトスクリプティングなどの問題もある (ことがある)
- 質問:「Web 2.0って何ですか?」

4.6 Web 2.0 とプログラミング技術

□ Web 2.0 とは…

- 明確な定義はない。主要な特性として次の 2 点が挙げられる
- 集合知 --- 特定個人/集団がサイトを作って情報発信するのでなく、多くの参加者が持ち寄った情報が全体として価値を形成
- リッチな体験 --- 従来の「単なるサイト」と区別される使いやすいインタフェース
- しかしそう言われても…「掲示板」とかは違うのか? 「2 ちゃんねる」は Web 2.0 ではないのか? 「2 ちゃんねる」にかっこいいインタフェースをつければいいのか? (よく分からないところ)

□ Web 2.0 を体現していると言われるサイト/サービス

- SNS (Social Networking Sites) --- mixi など。足跡により誰が見に来たかが分かりつながりが生まれる
- Wikipedia --- 多くの人が自分の詳しい分野について書くことにより市販の百科辞典に見劣りしない詳しさ
- Google Earth --- 地球全体について広い範囲で倍率を変えて上空からの写真を見ることができる。確かに体験はリッチだが集合知なのかどうか?
- Google (検索) もある意味そう。多くの人が参照するページはランクが上がるため「集合知」を構成

□ CMS (Contents Management System) --- 従来の「HTML+CSS で作成して設置」に変わるモデル

- ページやその内容の追加/更新/削除はすべて Web 経由で (ブラウザ上で) 行える→特定の作業環境が不要
- HTML ではなくより簡単なマークアップ方式で書ける
- 複数の人が共同で作成することも用意
- デザインは CMS が管理 (テンプレート) →すべてのページの見え方が自動的に統一される

□ さまざまな CMS のカテゴリ

- 汎用 CMS --- Zoops など
- Wiki --- 誰もが内容を書き込み修正するというコンセプト。Wiki 名 (大文字小文字混じりの名前と [[xxx]] の形のもの) は自動的にリンクになりそこをたどると「空っぽのページ」ができているので、それを書き換えてページを作成
- Blog --- カレンダーに関連づけて記事を書ける。掲示板が設置でき読み手にコメントをもらえる。トラックバックにより Blog どうしが相互につながりを持てる
- LMS (Lecture Management System) --- 大学の授業などで資料、課題、質問、レポート収集などの機能を科目単位で統合的に提供

□ サーバ側/クライアント側プログラミングとの関連…

- CMS はデータ (ページ内容) を保管するため、基本的にすべてサーバ側プログラミング技術 (設置と管理が必要)
- リッチな体験の部分ではクライアント側技術が使われる→ Ajax

□ Ajax (Asynchronous JavaScript with XML)

- これまでの Web は「ページ」単位でできていて、新しい内容を見る時はそのページに「遷移」していた
- Google Maps などは「1 つのページを表示したまま」次々に地図の上を移動していける→スムーズで使いやすい
- このためには (1) 表示側:JavaScript でページ内容書き換え (DOM)、(2) データ側:フォーム送信ではなく JavaScript 内から「任意の時点でページ遷移せずにデータを送り結果を受け取る」

□ XMLHttpRequest --- Ajax で使われる通信機能

- HTTP プロトコルによる Web サーバとの通信を起動し、結果が返送されてきたら受け取れる。結果を「待つ」こともできるが、それだと処理が「止まって」し

まうので通常は結果が来たらイベントハンドラが呼ばれるように設定 (非同期:Asynchronous)
- 返送される結果は文字列としても受け取れるが、XML データを受け取るのが基本になっている→構造を持ったデータを受け渡しできる
- たとえば地図であれば「マウスをドラグしてスクロール」→「地図画像のないところが表示範囲に入って来そうになる」→「その地図を取り寄せる要求を発行しておく」という形で使用

□ Ajax の適用範囲拡大

- Office スーツなども作れる --- ブラウザ画面でワープロ、表計算などが実行可能 (作成した結果はサーバに保存) → 将来はブラウザだけあればあとは「サービスと契約」するだけで手元の PC にはソフトを買わなくなるかも?
- Ajax で呼び出す機能 (API) を公開するところも --- 手元側で JavaScript を使ってこれらの API を呼び出すことで「自分のページ」に「さまざまな機能」を入れられるようになる
- SOA(Service Oriented Architecture) --- Web 経由で呼び出せる機能/サービスを組み合わせて必要なサービスを作り出し企業の情報システムなどを構成する --- も Ajax 化が進むかも?

□ 4.6 節のまとめ

- Web 2.0 --- 集合知とリッチな体験
- 要素技術 --- CMS(サーバ側)、Ajax(インタフェース)
- 今後、インストールレスの Web アプリがさらに発展かも…

5 さいごに

□ 久野の専門→プログラミング言語、プログラミング技術の応用、情報教育など

- 学生さんの主要な研究テーマ…
- 「自分が扱っているドメインの専用言語」を開発…「ゲームスクリプト記述」「実時間グラフィクス処理」「教育用オブジェクト指向言語」
- 「特定用途に役立つ情報システムの要素技術」…「Web ページ閲覧履歴からの検索効率化」「教育コンテンツの枠組み」「プラント制御システムのインタフェース」
- その他、ものを作って動かすこと、それを使ってもらうこと、教育/技術移転への適用などのテーマ

□ GSSM は情報技術関係の教員が充実していることが大きな特徴の 1 つであり、この方面に興味を持つ方を積極的に受け入れています