JavaScript とプロトタイプ方式のオブジェクト指向言語

久野　靖*

2000.11.6

はじめに

「プロトタイプ方式のオブジェクト指向言語」としては Self が元祖かつ代表格だが、実は JavaScript もそうである。ここでは JavaScript を材料にそれがどんな感じなのか紹介する。

1　JavaScript 言語

□ …って、知ってます?

□ WWW ブラウザにインタプリタが組み込まれている言語

- クライアント側スクリプト用に作られた

□ スクリプト言語…って何?

□ Java と構文は「少し」似てるが本質は全然違う言語

- もとは LiveScript という名前だったけど Java ブームにあやかって改名

□ オブジェクト指向言語

- 「プロトタイプ方式の」オブジェクト指向言語

1.1　スクリプト言語とは?(という ML 座談会があって…)

□ ささっと簡単に書ける言語?　とりあえずこなす言語?

□ 書いてすぐ実行できる?←「インタプリタ」「弱い型」

□ Lisp は (SNOBOL は) スクリプト言語か?

□ 外界にあるものを利用したり組み合わせて目的を達成?

□ やっつけっぽくいい加減な言語?← Ruby や Python は…

1.2　JavaScript では…

□ ページ内容を「その場で」生成できる。

```
<script type="text/javascript">
document.write('<table border="2"><tr>');
for(int i = 0; i < 10; ++i) {
  document.write('<th>', i*i, '<\/th>');
}
document.write('<\/tr><\/table>');
</script>
```

□ ボタンの押し等のユーザ入力に対して起動

```
<script type="text/javascript">
function push() { window.alert('Push'); }
</script>
<form action="">
<input type="button" value="Push Me" onclick="push()">
</form>
```

□ しかしこの「documment.write()」「window.alert()」とは…?

- 「外界のモノ」はそれぞれ「オブジェクト」
- 「オブジェクト」の「メソッド」を呼ぶことで外界を色々に利用

□ つまり「オブジェクト指向言語」

- 「外界を呼ぶ」という意味でのスクリプト言語にはオブジェクト指向が適している (と思う)。

1.3　オブジェクト指向言語の 2 大流派

□ 「クラス方式」← Simula、Smalltalk-80、C++、Java、(CLU)

- 「オブジェクトの種類」に対応する「構文単位 (クラス)」がある
- オブジェクトはクラスを雛型として生成される
- オブジェクトの内部表現や操作群はクラスに対応させて記述

□ 利点→強い型による静的検査とかには向いている

*筑波大学大学院経営システム科学専攻

□ 弱点→記述量が多く繁雑、継承関係とか動的変更とか言いだすと複雑

□ 「プロトタイプ方式」→ Self、JavaScript

- オブジェクトは個々にデータやメソッドを持つ
- 自分で持ってないもの→「親」(プロトタイプ) が持ってれば使える
- 全部「オブジェクト」で統一、「クラス」はない

□ 利点→言語概念が少く簡潔、動的な変更になじむ

□ 弱点→実行させてみないと分からない (動的だから)

□ これまで「Self…」とか言っても誰も知らなかったが、JavaScript だったらそこら中に処理系があってデモもできる→よいことだ (久野のひとりごと)

1.4 JavaScrpipt のオブジェクト指向機能

□ オブジェクトを作る→プロパティを設定→メソッドとして呼べる

```
<pre><script type="text/javascript">
var o1 = new Object();
o1.value = 0;
o1.add = function(x) { this.value += x; }
o1.get = function() { return this.value; }
o1.add(3); document.writeln('value = ', o1.get());
o1.add(4); document.writeln('value = ', o1.get());
</script></pre>
```

□ ちょっと面白い機能…すべてのオブジェクトは連想配列であり、「o. 名前」と「o["名前"]」が等価

```
<pre><script type="text/javascript">
var o1 = new Object();
o1.value = 0;
o1.add = function(x) { this.value += x; }
o1.get = function() { return this.value; }
for(var i in o) {
  document.writeln(i, ' : ', o1[i]);
}
</script></pre>
```

□ コンストラクタ…オブジェクトを初期設定するための関数 (といっても普通の関数で単に this にいろいろ設定するだけ)

```
<pre><script type="text/javascript">
function Counter(x) {
  this.value = x;
  this.add = function(x) { this.value += x; }
  this.get = function() { return this.value; }
}
var c1 = new Counter(5);
document.writeln('value = ', c1.get());
c1.add(3); document.writeln('value = ', c1.get());
</script></pre>
```

□ 先の例ではオブジェクトごとに全部のメソッドを連想配列に格納→無駄だし共有がなされていない

□ 「コンストラクタ名.prototype」という変数にプロトタイプオブジェクトが格納されているので、そのプロパティを設定すれば共有される

```
<pre><script type="text/javascript">
function Counter(x) { this.value = x; }
Counter.prototype.add = function(x) { this.value += x; }
Counter.prototype.get = function() { return this.value; }
var c1 = new Counter(5);
document.writeln('value = ', c1.get());
c1.add(3); document.writeln('value = ', c1.get());
</script></pre>
```

□ 継承のようなことをしたければ、プロトタイプをすげかえる

```
<pre><script type="text/javascript">
function Counter(x) { this.value = x; }
Counter.prototype.add = function(x) { this.value += x; }
Counter.prototype.get = function() { return this.value; }
function ExCounter(y) { this.value = y; }
ExCounter.prototype = new Counter(0);
ExCounter.prototype.sub = function(y) { this.value -= y; }
var c1 = new ExCounter(5);
document.writeln('value = ', c1.get());
c1.sub(3); document.writeln('value = ', c1.get());
</script></pre>
```

□ …というわけで、JavaScript のこのあたりの言語機構は簡潔でなかなか面白いと思う

- オブジェクトのプロパティが連想配列と統合
- プロパティに関数を代入するとメソッドになる
- 関数を new 演算子経由で呼び出すとコンストラクタになる
- コンストラクタの prototype プロパティでプロトタイプを指定

1.5 JavaScript のその他の特性

□ スコープ→静的スコープ規則、トップレベルオブジェクトのプロパティが広域変数や言語定義のクラス群を格納していて、入れ子スコープがチェインになっているる (静的なのか動的なのかよく分からない気もする)

```
<pre><script type="text/javascript">
function f() {
  function g() { return x; }
  var x = 10;
  return g;
}
var x = 1000;
var h = f(); document.writeln(h());
</script></pre>
```

□ 値とオブジェクトが別々(Smalltalk-80 流ではなく C++/Java 流) →すごく嫌。

- 数値→ Number オブジェクト、論理値→ Boolean オブジェクト、文字列→ String オブジェクト、null 値 → Null オブジェクト、と対応。

□ しかも値をオブジェクトとして使おうとするとオブジェクトに、オブジェクトを値として使おうとすると値に自動変換→ますます嫌。だったらなんで全部オブジェクトにしておかない?

```
<pre><script type="text/javascript">
document.writeln("abcd".toUpperCase());
document.writeln(3.1416.toExponential(3));
document.writeln((new Number(3)) + 5);
</script></pre>
```

□ その他の標準オブジェクト種別

- Math →数学関数の置き場所、Array →配列、RegExp →正規表現、Date →日付の操作
- Function →関数オブジェクト (名前つき、無名、…)

□ 例外機構 (Java と同じ)

2 Document Object Model (DOM)

□ WWW コンソーシアム (W3C、HTML などの標準を取りまとめている) が提唱している

□ Web ページの中身 (のデータ構造) を「オブジェクトの集まり」として規定したもの

□ 各オブジェクトのメソッドを呼ぶことでページの中身が操作できる

□ 「呼ぶ」のには当然、ブラウザ上で動いている言語処理系で行なう→ JavaScript がもっとも多く使われる

□ DOM ではすべてのオブジェクトの型を規定しているが、いちいち「このオブジェクトはどんな型」と書きつつ操作するのは繁雑→強い型を持たない JavaScript が向いている

2.1 フォーム部品のスクリプトによる操作

□ フォーム部品はもともと「変化する」ようにできているので、わりと古いブラウザでも動く。

```
<pre><script type="text/javascript">
function calc() {
  var f1 = document.forms[0].elements[0];
  var i = parseInt(f1.value);
  f1.value = String(i + 1);
}
</script></pre>
<form action=""><p>
<input type="text" name="a" size="5">
<input type="button" value="Push" onclick="calc()">
</p></form>
```

2.2 Node オブジェクト

□ ドキュメントの内容は Node オブジェクトの木構造で表される。これを再帰的にたどることで全ノードを表示させられる。

```
<script type="text/javascript">
var w1 = window.open("", "Test");
var d1 = w1.document; d1.open();
d1.writeln("<pre>\nTest...");
function doNodes(node, ind, num) {
  d1.write(ind, "[", num, "]TYPE=",
      node.nodeType,";NAME=", node.nodeName);
  if(node.id) d1.write(";ID=", node.id);
  if(node.className)
    d1.write(";CLASS=", node.className);
  if(node.nodeValue)
    d1.write(";VAL=", node.nodeValue.replace(/\n\<\>/g,"."));
  d1.writeln("");
  if(node.attributes) {
    for(var i = 0; i < node.attributes.length; ++i) {
      doNodes(node.attributes.item(i), ind + " |", i);
    }
  }
  if(node.childNodes) {
    for(var i = 0; i < node.childNodes.length; ++i) {
      doNodes(node.childNodes.item(i), ind + "  ", i);
    }
  }
}
</script>
<form action=""><p><input type="button" value="exec"
onclick="doNodes(document, '', 1)"></p></form>
```

2.3 ノード内のテキストの変更

□ テキスト型ノードではメソッド insertData() 等を使って中身のテキストを変更できる。

```
<p id="x1" onmouseover="doTest()">たとえばこれですが。
</p>
<script type="text/javascript">
function doTest() {
  var x1 = document.getElementById("x1");
  x1.firstChild.insertData(0, "あの一、");
}
</script>
```

2.4 スタイル情報の変更

□ ノードに付随しているスタイルシートの指定値を変更することで、ノードの表現 (位置や色やフォント等) も変更できる。

```
<p id="x1" style="position: absolute; top: 50px; left: 50px"
onmouseover="doTest()">たとえばこれですが。</p>
<script type="text/javascript">
var pos = 50;
function doTest() {
  var x1 = document.getElementById("x1");
  x1.style.removeProperty("top");
  x1.style.removeProperty("left");
  x1.style.top = x1.style.left = (pos += 10) + "px";
}
</script>
```

2.5 DOM を利用してみると…

□ これまでは「オブジェクトを自分で用意してそれを利用する」感覚だったが、それが「オブジェクトはもう山のようにあって、そこから使いたいものを取り出して来て操作する」感じに近付いている。

□ こういう使い方はいかにもオブジェクト指向言語っぽいのかも…

□ しかし NIH 症候群っぽい抵抗感もありますね…

□ オブジェクト群の構造を設計するのが面白いのにそれはもう与えられてしまっているというか…

3 教育用オブジェクト指向言語ドリトル

□ (久野なりの紹介です)

3.1 動機

□ 高校新教科「情報」などでプログラミングをちゃんとやって欲しい

- 「情報 B」に一応入っているが表計算のマクロなどでもいいという…
- 現在使える言語がよくないということもあるかも?
- 「普通の」言語 (工業→ C、商業→ COBOL、その他…) →敷居が高い

□ 小/中/高での「教育用」言語といえば…

- Basic ←ただでついて来たから、JIS だから (教科書に載っている)
- Logo ←タートルグラフィクスで根強い人気
- いずれも「古くさい」「現在のプログラミング言語の水準から遅れている」と思う

□ どちらにも共通する問題…苦労してプログラムを作っても「行単位入力/出力」だとちっとも「見栄え」がしない→やる気が出ない

3.2 言語の設計方針

□ オブジェクト指向言語

- 部品を組み合わせて「それらしい」ものが作れることをさっさと体験させたい
- そのようにしても「プログラミングを学ぶ」という本質は保たれると思う…(どうでしょ?)

□ プロトタイプ方式

- 理解する概念をなるべく少くしたい
- さらに、オブジェクトの作り方は「あるオブジェクトをコピー」→コピー元がプロトタイプになる

□ テキスト表現によるソースコードを持つ

- Programming by Example や図的プログラミングはやりたくない
- 「テキストの記号表現→計算機による解釈→計算機の動作」というモデルを与えたい

□ terse な言語

- Basic や LOGO がよかった点→「ひとこと」言えばそれで動く→そのような利点は受け継ぎたい
- JavaScript の「関数を代入するとそれでメソッド」みたいな…
- 1 行プログラミング (しかし APL みたいなおまじないじゃあなく)

□ 日本語との対応性

- 小学校から使うかも→英語はやめよう。
- 予約語という概念も難しい→予約語はなし。日本語の単語と基本記号
- 「」でも [] でもよい、「。」でも「.」でもよいようにしたい
- わかち書き…悩みの種。でもわかち書きしないのも難しい…

□ 入れ子構造はなるべく避ける

- 入れ子構造の概念はむずかしい
- 言語仕様としては可能でも実際にはあまり入れ子にしないで書けるように
- メソッドをバラバラに分けたり動的分配で枝分かれすれば済むのでは

3.3 実行系の設計方針

□ オブジェクトが画面上にも現われるように

- 最初から「アイコン」
- アイコンをつつく→メッセージが送られる→メソッドが定義されていればそれが動く、という形で「すぐ試せる」ようにしたい
- 日常使っている GUI 部品とのギャップを小さく
- GUI 部品のようなものも最初から用意

□ LAN 上でのオブジェクトの共有

- 一人の成果を多人数で共有したい→ブラウザとサーバを使って共有をサポートした実行系にしたい
- 最初から分散プログラミング…過激でしょうか?
- 共有とコピーの違いを体験…過激でしょうか?
- CSCW のようなことを「ささっと」作れるようにしてみたい

□ 標準の部品オブジェクトを十分用意

- どうやって探させる? Smalltalk-80 のブラウザみたいのは難しそうだから…
- 「本」や「巻物」にオブジェクトが貼り付けてあって→それを読んでそのままコピーして利用できるとか…
- ブラウザベースならブラウザの画面にあればどうにでもなる?

3.4 文法 (現時点の)

```
プログラム ::= (文 ';')…
文   ::= [変数 '='] 式
変数 ::= [項 '.'] 名前
式   ::= 単純式 | 送信
送信 ::= [項] ',' 電文
電文 ::= 単純式… 名前 ('.' 単純式… 名前)…
括弧 ::= '(' 中置式 ')' | '(' 送信 ')'
単純式 ::= 整定数 | 文字列 | 括弧 | ブロック
ブロック ::= '[' ['|' 名前… '|'] 文 (';' 文)… ']'
中置式 ::= 中置式 演算子 中置式 | 項
項   ::= 単純式 | 名前
```

□ 「文」1 つ打ち込むごとに実行するイメージ (「;」はちょっと嬉しくない)

□ メッセージは「オブジェクト、 引数… セレクタ」という語順 (丸かっこで囲んだ引数リスト、というのから逃れたかった)、しかし後置記法ではない。

□ 中置記法かメッセージ送信かの 2 者択一。メッセージ送信は入れ子よりも連続送信を中心にしたい

3.5 コード例

```
カメ太 = タートル、作る
  // 「、」の左がレシーバ、右がメッセージセレクタ
  // 「作る」はレシーバをプロトタイプとするコピーを返す

矩形 = カメ太、50 上へ 100 右へ 50 下へ 閉じる
  // 識別子が来るとそこまでがメッセージとして送られる
  // その右は返値に対するメッセージ送信
  // 最後の値が式全体の値
  // ここではカメ太が描いた図形 (パス) がオブジェクトとして
  // 取り出される

矩形、90 度回転 ("青" 色) 塗る
  // パスオブジェクトはそれ自体自由に移動、変形、回転できる
  // 文字列オブジェクトの「色」で色オブジェクトを生成

  白矩形 = 矩形、作る 50 右移動 ("赤" 色) 塗る
  赤矩形 = 白矩形、作る 50 右移動 ("赤" 色) 塗る
    // コピーするとコピー元がプロトタイプになる

  // メソッドをつける例
カメ太.矩形 = [|x y|、(2*x) 上へ (x) 右へ (2*x) 下へ
    閉じる (y) 塗る];
  // […] はブロック (後から評価されるコード列)
  // 変数を参照したければ丸かっこの中に入れる
  // 丸かっこの中はメッセージ式か中置記法のいずれか

カメ太、50("青"、色) 矩形。50("白"、色) 矩形。
    50("赤"、色) 矩形;
  // 「。」があるとそれ以降は再度「、」の
  // 左のレシーバへのメッセージ
```

□ なお、ここではタートルオブジェクトをコードによって動かし図形を生成していたが、タートルオブジェクトを直接ドラグして図形を生成してもよい。

3.6 制御構造

□ 制御構造は Boolean オブジェクトやブロックオブジェクトへのメッセージとして実現 (Smalltalk-80 と同じ)。ただしセレクタが最後の 1 語だけという制約があるので形を変えている。

□ 枝分かれ…

```
(x > y)、ならば [...] 実行。でなければ [...] 実行。
```

「ならば」や「でなければ」を Boolean に送ると、ブロックを引数として「実行」メッセージを送られたときにそのブロックを評価する/しないオブジェクトが返される。ループも同様。

□ ループ…

```
[(x > y)]、の間 [...] 実行。
```

本当はこの丸かっこを取り除きたい (構文を検討中)。この程度の簡単な構文と、あとは予め用意されたオブジェクト群だけでそれなりに動作するプログラムが書けるはずと考えている。

3.7 配列

□ 配列 (リスト) は特にそのための構文は用意しないが、「配列」オブジェクトに任意個の引数をつけてメッセージを送ることで生成できる。

```
配列 1 = 配列、1 2 3 4 5 作る
数値 1 = 配列 1、3 番目
```

3.8 その他…

□ 言語は以上のような感じだが、このシステム全体としての有用性 (特に教育効果の面で) は、あらかじめどの程度有用な部品 (オブジェクト) を用意しておけるかに掛かっている。現在考えているリストの一部を示す。

- 文字列
- 数値
- 論理値
- ブロック
- 配列
- ペン先 (ないしタートル)
- 図形 (パス)
- 色
- 模様
- GUI 部品 (ボタン、メニュー、スライダ、チェック…)

3.9 とりあえず処理系

□ Java 2+SableCC(コンパイラコンパイラ) でささっと作った

- SableCC は日本語も自由に扱えるのでその辺は楽
- Lex/Yacc とはだいぶ使い勝手が違う (すぐにできることはできる)

□ ちゃんとユーザインタフェースを作らないと実験ができないので…

3.10 デモ 1(フランス国旗)

```
カメ太 = ルート, タートル ペンなし -250 歩く
  -90 右回り 200 歩く 90 右回り ペンあり;

カメ太, 150 歩く 90 右回り;
カメ太, 300 歩く 90 右回り;
カメ太, 150 歩く 90 右回り;
カメ太, 300 歩く 90 右回り;

長方形 1 = カメ太, 図形にする 青 塗る;

長方形 2 = 長方形 1, 複製;
長方形 2, 150 0 移動;
長方形 2, 白 塗る;

長方形 3 = 長方形 2, 複製;
長方形 3, 150 0 移動;
長方形 3, 赤 塗る;
```

3.11 デモ (図形もオブジェクト)

```
カメ太 = ルート, タートル;

カメ太, ペンなし -150 歩く ペンあり;
カメ太, 100 歩く 90 右回り 100 歩く 90 右回り 100 歩く
  90 右回り 100 歩く 90 右回り;
四角形 = カメ太, 図形にする オレンジ 塗る;

カメ太, ペンなし 250 歩く ペンあり;
カメ太, 30 歩く 36 右回り 30 歩く 36 右回り 30 歩く
  36 右回り 30 歩く 36 右回り 30 歩く 36 右回り
  30 歩く 36 右回り 30 歩く 36 右回り 30 歩く 36 右回り
  30 歩く 36 右回り 30 歩く 36 右回り;

十角形 = カメ太, 図形にする 赤 塗る;

カメ太, ペンなし -100 歩く ペンあり;

三角形 = カメ太, 100 歩く 120 右回り 100 歩く 120 右回り
  100 歩く 120 右回り 図形にする 水色 塗る;

三角形, 30 回転;
三角形, 30 回転;
四角形, 緑;
三角形, 30 回転;
三角形, 30 回転;
三角形, 30 回転;
十角形, -20 回転;
四角形, -20 回転;
三角形, 青;
三角形, 2 拡大;
三角形, 30 回転;
十角形, -20 回転;
四角形, -20 回転;
三角形, 30 回転;
```

十角形，-20 回転;
四角形，-20 回転;
三角形，10 回転;