なんとなく日記

<[Emacs/Meadow]C-aで「行頭」と... | [ニコニコ動画][Greasemonkey]ni...>

2010年01月31日

JavaScript基礎文法最速マスター 16

Javascript

続々と増え続ける基礎文法最速マスターシリーズ(あまりにも増えてきたので他の言語記事へのリンクは別の記事に移しました)。

JavaScript 版は誰も書いていなかったようなので書いてみます。こういう解説記事的なものを書くのは初めてなので変なところがあったら指摘して頂けるとありがたいです。

JavaScriptの基礎概念についての解説はこちら:JavaScript基本概念最速マスター - TechTalkManiacs

1/31 23:58追記
コメント欄のos0xさんのご指摘を基に一部追記・修正を行いました。
2/2 2:52追記
switch文・正規表現・例外処理について追加しました。
2/2 6:44追記
コメントでfavrilさんにご指摘頂いた点(typo & コメント・サンプル追加)を修正・加筆しました。
2/2 7:15追記
トラックバックでLiosKさんにご指摘頂いた点(call, apply, プリミティブ値, strict モードのthis) を修正・加筆しました。
2/2 20:02追記
コメント欄のos0xさんのご指摘を基に, call, applyのサンプルコードを修正しました。

まえがき

本記事は主に JavaScript の文法面について解説します。基本的に ECMAScript 第3版の範囲内の文法について取り扱いますが, そこからはみ出す部分については適宜注釈をつけておきます。

JavaScript の標準仕様

JavaScript の仕様は標準化団体 Ecma International によって ECMAScript として定められています。この ECMAScript を各社が実装したのが Mozilla 等の JavaScript や Microsoft の JScript, Adobe の ActionScript などです(詳細はECMAScript - Wikipedia参照)。ただ単に JavaScript と言った場合はこれらの実装をひっくるめて呼んでいることが多いようです。

JavaScript の実行 (ブラウザ編)

JavaScript で書かれたを実行するためにはブラウザを用いるのが一番手っ取り早いです。1行で済むスクリプトを実行する場合, ブラウザのロケーションバーに以下のように打ち込みます。

```
javascript: 実行したいプログラム
```

例えば, メッセージを表示させるなら以下のようにします。

```
javascript: alert("aaa");
```

ブックマークレットはこのようにして JavaScript プログラムを実行させています。

上記の仕組みで実行できるのは~~1行に収まる~~(改行を含まない) JavaScript だけです。改行ではなくセミコロンで文を区切ることで複数の文を含むプログラムも実行できますが, そのようなプログラムは読み書きしづらく大変です。複数行の JavaScript を実行させるためには, JavaScript のソースコードだけではなく以下のような HTML を書き, それをブラウザに読み込ませる必要があります。

日記の検索
検索
詳細 一覧

カレンダー
<< 2010/01 >>

日	月	火	水	木	金	土
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

カテゴリー
ニコニコ動画
Javascript
D言語
OCaml
TeX
MATLAB
数学
コピペ
Firefox
Greasemonkey
Stylish
ブックマークレット
その他
Xpath
あとでやる
PowerShell
HTML/XHTML
Emacs/Meadow
Ubuntu
Perl
Linux
sh
Java
プログラミング

最新タイトル
基礎文法最速マスターシリーズのまとめ
nicovideo Thumbinfo Popup を更新
JavaScript基礎文法最速マスター
C-aで「行頭」と「インデントを飛ばした行頭」を行き来する

```
<html>
<head>
<title>JavaScript テスト</title>


<script type="text/javascript">

</script>


<script type="text/javascript" src="ファイル名"></script>

</head>
<body>

<script type="text/javascript"></script>
</body>
</html>
```

JavaScript の実行（ブラウザ以外編）

ブラウザ上の JavaScript と比較するとマイナーですが，他の言語のようにシェル上で JavaScript を実行できる実装もあります。ここでは詳細には触れません。

基礎

print文

JavaScript の標準ライブラリには入出力に関する関数が一切定義されていません！文字列を出力する場合は，ブラウザで実装されている alert 関数を用いる事が多いです。

```
alert("Hello, world");
```

また，最近は多くのブラウザで console.log がサポートされているので，コンソールウインドウにメッセージを出すこともできます (Firefoxの場合，Firebugがインストールされていて起動していなければなりません)。console.log を使うと文字列だけでなく様々なオブジェクトを出力することができます。

```
console.log("Hello, world");
console.log([1, 2, 3]);
```

コメント

コメントは C 系の言語と同じです。

```
// 一行のコメント
/*
複数行の
コメント
*/
```

文法チェック

次世代バージョンの JavaScript (ECMAScript 5) では Strict モードという，より厳密に文法チェックが行われるモードの導入が予定されています。メジャーな処理系はまだ Strict モードをサポートしていないので，Strict モードがどのようなものになるかは

未だに不透明な面があります (仕様は定まったのですが，仕様通りに実装が行われるとは限らない為)。しかし今後は Perl のように Strict モードでスクリプトを書くのが常識，となる可能性は高そうなので，今から JavaScript を覚えるのなら厳密な書き方を身につけておいた方が良いでしょう。

Strict モードにするためには，スクリプトの先頭に以下を書き込みます。このコード自体はただの文字列なので，Strict モードに対応していないブラウザには何の影響も与えません。

```
"use strict";
```

文

文の最後にはセミコロン (;) を付けます。付けない場合自動で補われますがたまに変な挙動を起こすのでできるだけ付けるようにしましょう。

変数の宣言

var で宣言します。変数に型はありません。

```
var hoge = 1;
hoge = "a"; // 数字も文字も代入できる
```

宣言していない変数に値を代入することもできますが，その場合はグローバル変数が作られそこに代入されます (厳密にはちょっと違いますが)。未定義変数への代入操作は Strict モードではエラーになるのでできるだけ使わないようにしましょう。

Emacs23対応版
nicovideo Tag Edit Helper を更新
nicovideo Thumbinfo Popup を更新
nicovideo Tag Edit Helper を更新
Firefox 3.6でFlashの上に position: fixed; な要素を表示できなくなった件
Cygwin 1.7 で作成したファイルにWindowsの共有属性が付かないようにする
Orchis経由で起動した minttyで日本語入力ができない不具合

注目エントリー
JavaScript基礎文法最速マスター - なんとなく日記 1242users
nicovideo Thumbinfo popup の新版を公開 - なんとなく日記 14users
ニコニコ動画情報挿入系スクリプトを更新 - なんとなく日... 6users
nicovideo Tag Edit Helper Helper を更新。 - なんとなく日記 5users
ニコニコ動画関連スクリプトの(βββ)対応 - なんとなく... 5users

最近言及したキーワード
.emacs ActionScript CSS Firebug Greasemonkey HTML border インストール オプション コンパイル スクリプト デフォルト ニコニコ動画 パラメータ ポップアップ ユーザー リンク 言語 仕様 不具合

最近のコメント
2010/01/31 gifnksm
2010/01/31 os0x
2010/01/31 gifnksm
2010/01/31 favril
2010/01/31 gifnksm
2010/01/31 通りすがり
2010/01/31 gifnksm
2010/01/31 t_takata
2009/04/03 gifnksm
2010/01/31 gifnksm

最近のトラックバック
2010/02/02 燈明日記 - 基

```
a = 1; // エラーにならない
```

数値

JavaScript の数値は全てが実数型です。整数型という概念はありません。

```
var d = 123456;  // 10進数の整数
var h = 0xffff;  // 16進数
var o = 0123;    // 8進数（Strictモードでは10進数となります）
var f = 12.345;  // 実数
```

数値演算

JavaScript には実数型しかないので，演算の結果も実数型になります。ただし， ビット演算の場合は小数点以下を切り捨てて整数に変換してから行われます。

```
var a = 1 + 2;    // => 3
a = 3 - 2;        // => 1
a = 1 * 5;        // => 5
a = 3 / 2;        // => 1.5（整数同士の割り算でも結果は実数）
a = 3 % 2;        // => 1   （余り）
a = 255.1 & 2.1; // == 255 & 2 => 2   （ビット演算は整数に変換(小数点以下切り捨て)してから行われる）
a = 12.3 >> 1;    // => "6"（同上）
```

代入演算子とインクリメント・デクリメント

C 系言語と同じように使えます。

```
var a = 0;
a += 3;          // => 3
a -= 2;          // => 1
a *= 3;          // => 3
a /= 3;          // => 1
a = 0;
var b = a++;     // => a == 1, b == 0
var c = ++a;     // => a == 2, c == 2
var d = a--;     // => a == 1, d == 2
var e = --a;     // => a == 0, e == 0
```

文字列

文字列はシングルクオート(')かダブルクオート (")で囲みます。両者は全く等価です。シングル・ダブルクオートのどちらの場合でも \t（タブ），\n（改行）などの特殊文字を利用することができます。変数展開などの便利な機能はありません。

```
var a = "abc¥tdef"; // "abc[tab]def" ([tab]はタブ文字)
var b = 'abc¥tdef'; // "abc[tab]def"
```

文字列操作

結合

```
var join1 = 'aaa' + 'bbb';
var join2 = ['aaa', 'bbb', 'ccc'].join(','); // => 'aaa,bbb,ccc'
```

分割

```
var record = 'aaa,bbb,ccc'.split(/,/); // => ['aaa', 'bbb', 'ccc']
```

長さ

```
var length = 'abcdef'.length;     // => 6
var jplen  = 'あいうえお'.length; // => 5
```

切り出し (substrは多くの環境でサポートされていますが非標準なメソッドなので一応注意)

```
var substr    = 'abcd'.substr(1, 2);    // => bc
var substring = 'abcd'.substring(1, 2); // => b
```

検索

```
// 見つかった場合はその位置，見つからなかった場合は-1が返る
var result1 = 'abcd'.indexOf('cd');  // => 2
var result2 = 'abcd'.indexOf('ef');  // => -1
```

配列

配列の生成方法いろいろ。

礎文法最速マスターぞくぞくキターーー！

2010/01/31 jQuery基礎文法最速マスター

2010/01/31 Life like a clown - はてな的プログラミング言語人気ランキング

2010/01/31 TechTalkManiacs - JavaScript基本概念最速マスター

2010/01/31 [C#]C#基礎文法最速マスター

2010/01/31 どうでもいい情報置き場 - Whitespace基礎文法最速マスター

2010/01/31 (rubikitch loves (Emacs Ruby CUI)) - Emacs Lisp基礎文法最速マス...

2010/01/31 [JavaScript] prototypeに直接代入しちゃうのってダメじゃなかった...

2010/01/31 CHOLOG - くだらないつぶやきのLOG | twitter本日分post

2010/01/31 [Flash] ActionScript 3.0 基礎文法最速マスター

プロフィール

gifnksm

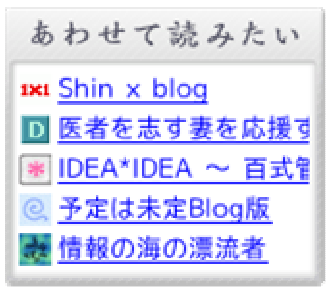
神奈川在住の岐阜県民。大学生。

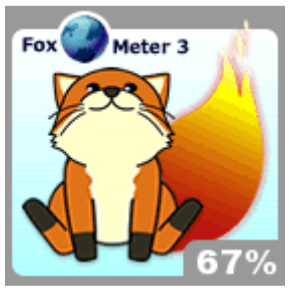
いろいろ

Greasemonkey等サポート掲示板

User Script (Greasemonkey)

User Style (Stylish)





ページビュー

175108

```
var ary1 = [1, 2, 3];          // => [1, 2, 3]
var ary2 = new Array(3);       // => [undefined, undefined, undefined]
var ary3 = new Array(3, 4, 5); // => [3, 4, 5]
```

配列の参照と代入

```
var ary = [1, 2, 3];
ary[2];               // => 3  (配列のインデックスは0オリジン)
ary[0] = 3;           // => ary == [3, 2, 3]
```

要素の個数

```
ary.length
```

配列の操作

```
var ary = [1, 2, 3];
// 先頭を取り出す
var a = ary.shift(); // => a == 1, ary == [2, 3]
// 先頭に追加
ary.unshift(5);      // => ary == [5, 2, 3]
// 末尾を取り出す
var b = ary.pop();   // => b == 3, ary == [5, 2]
// 末尾に追加
ary.push(9);         // => ary == [5, 2, 9]
// 部分コピーを得る
var c = ary.slice(1, 2);
                     // => c == [2], ary == [5, 2, 9]
// 一部を置き換える
var d= ary.splice(1, 2, "a", "b", "c");
                     // => d == [2, 9], ary == [5, "a", "b", "c"]
```

連想配列（のようなもの）

JavaScript には連想配列というものはありませんが，任意のオブジェクトを連想配列のように扱うことができます。

```
// オブジェクトの定義。JSON はこの表記法を基にしている
var a = {a: 123, b: 456};
a['a'];               // => 123（連想配列風アクセス）
a.b;                  // => 456（プロパティ風アクセス）
a['c'] = 789;         // 要素の追加
a.d = 123;
```

以下のような書き方も可能ですが，普通はやりません。

```
"abc"['length'];       // => 3（プロパティの取得）
"a,b,c"['split'](/,/); // => ["a", "b", "c"]（メソッド呼び出し）
```

連想配列風のアクセスとプロパティ風のアクセスは表記が異なるだけで意味上の差はありません。

制御文

if文

```
if (条件) {
  hoge();
  fuga();
}
// if中の文が1つだけの場合は 括弧（ "{", "}" ）を省略可能
if (条件)
  hoge();
```

if-else文

```
if (条件) {
} else {
}
```

else以下に更に条件分岐を重ねる時は以下のようにも書いたりします。

```
if (条件1) {
  // 条件1がtrue
} else if (条件2) {
  // 条件1がfalseで条件2がtrue
} else if (条件3) {
  // 条件1, 2がfalseで条件3がtrue
} else {
  // 条件1, 2, 3 がfalse
}
```

中括弧を省略せずに書くと以下のようになります。

```
if (条件1) {
  // 条件1がtrue
} else {
  if (条件2) {
    // 条件1がfalseで条件2がtrue
  } else {
    if (条件3) {
      // 条件1, 2がfalseで条件3がtrue
    } else {
      // 条件1, 2, 3 がfalse
    }
  }
}
```

while文

```
var i = 0;
while (i < 5) {
  i++;
}
```

for文

```
for (var i = 0; i < 5; i++) {
}
```

for in文

```
var obj = {a: 1, b: 2};
for (var i in obj) {
  alert(obj[i]);  // => 1, 2
}
```

for each文 (Firefoxのみ対応)

```
var obj = {a: 1, b: 2};
for each (var v in obj) {
  alert(v);       // => 1, 2
}
```

Array#forEach (最近のブラウザのみ対応)

```
["a", "b", "c"].forEach(function(v, i) {
  alert(i + ": " + v);  // => "0: a", "1: b", "2: c"
})
```

switch文

構文はCと一緒です。breakが必要。

```
var a = 0;
switch (a) {
case 0: alert('zero'); break;
case 1: alert('one'); break;
case 2: // fall through
case 3: alert('two or three'); break;
default: alert('many!'); break;
}
```

Cと異なり，caseの後には実行時に評価される値を入れることもできます。

```
switch (a) {
case a + '': alert('String'); break;
case a + 0: alert('Number'); break;
case !!a: alert('Boolean'); break;
default: alert('Something'); break;
}
```

関数

関数は以下のように書きます。

```
function sum3a(a, b, c) {
  return a + b + c;
}
```

JavaScriptの関数はファーストクラスのオブジェクトなので，無名の関数を作って変数に代入することもできます。

```
var sum3b = function(a, b, c) {
  return a + b + c;
};
```

sum3aはコンパイル時に定義され，sum3bは実行時に定義されます。それ以外の点で両者に差はありません。

Firefox だと以下のような短縮表記が使えます (式クロージャ記法)。

```
var sum3b = function(a, b, c) a + b + c;
```

ファイル入出力

そんなものはない (ブラウザ上で動作するJSの場合)

サーバとの通信は行えますが煩雑なので省略 (XMLHttpRequestでググってください)

正規表現

JavaScriptで使える正規表現フラグ・特殊文字はRegExp - MDCにまとまっています。

正規表現オブジェクトは以下のようにして作ります。

```
var regex1a = /¥d+/;              // 正規表現リテラルを使用
var regex1b = new RegExp('¥¥d+'); // 文字列から生成 (regex1aと同値)
var regex2a = /¥s+/g;             // g オプションを付与
var regex2b = new RegExp('¥¥s+', 'g');
// スラッシュをたくさん含むときは文字列から生成する方が読みやすい
var urlRegA = /^http:¥/¥/[^¥/]+¥/$/;
var urlRegB = new RegExp('^http://[^/]+/$');
```

文字列から生成する場合は, \d などの特殊文字を \\d と書かなければならないことに注意してください。

検索

正規表現を使った検索の方法は4種類あります。

```
var regex = /[a-z](¥d+)/;

// マッチするか否かだけを調べるメソッドが2種類

// RegExp#test
regex.test('!!a123!!');   // => true
regex.test('!!123!!');    // => false

// String#search
'!!a123!!'.search(regex); // => 2
"!!123!!".search(regex);  // => -1

// 詳細なマッチ情報を得るメソッドが2種類

// RegExp#exec
var m1 = regex.exec('aaa123');
m1[0];    // => a123 (マッチした部分全体)
m1[1];    // => 123 (1つめの括弧 ( ) でキャプチャされた部分)
m1.index; // => 2 (マッチした場所)

// String#match
var m2 = 'aaa123'.match(regex);
// この場合得られる結果はRegExp#execと同じ (m2 == m1)
```

正規表現に g フラグを設定すると RegExp#exec, String#match の挙動が変わります。

まずは RegExp#exec について。

```
var s = '123';
var r = /¥d/;   // g フラグなし
var rg = /¥d/g; // g フラグあり

// g フラグが無いと毎回同じ結果
r.exec(s);      // => 1
r.exec(s);      // => 1
r.exec(s);      // => 1

// g フラグがあると前回のマッチの次の部分から検索を開始する
rg.exec(s);     // => 1
rg.exec(s);     // => 2
rg.exec(s);     // => 3
rg.exec(s);     // => null
```

文字列中のマッチした部分すべてについて何か処理を行いたい時, 以下のような書き方をしたりします。

```
var s = 'string';
var m;
while((m = /[a-z]/g.exec(s)) !== null) {
  alert(m[0]);   // => 's', 't', 'r', 'n', 'g'
}
```

次に String#match の場合。g フラグの有る無しで全く異なる挙動を示すので注意が必要です。

```
var r = /¥d(¥d)¥d/;
var rg = /¥d(¥d)¥d/g;

// g フラグ無しの場合 exec とおなじ
var m1 = '123456'.match(r); // => m1[0] == '123', m1[1] == '2', m1.index == 0

// g フラグありの場合, 文字列中のマッチした箇所全てを含んだ配列を返す
// 括弧でキャプチャした部分の取得は行えません
var m2 = '123456'.match(r); // => [123, 456]
```

置換

```
"123".replace(/¥d/, 'a');    // => 'a23'
"123".replace(/¥d/g, 'a');   // => 'aaa'
```

第2引数に渡す文字列中の $n (n > 0) は正規表現中の括弧でキャプチャした部分と置き換えられます。

```
// 括弧でキャプチャした部分を利用する
"___a123___".replace(/[a-z](¥d)(¥d)(¥d)/, '[[$1|$2|$3]]'); // => '___[[1|2|3]]___'
```

第2引数には関数を渡すこともできます。

```
var s = "___a123___".replace(/[a-z](¥d)(¥d)(¥d)/, function(m0, m1, m2, m3) {
  alert(m0); // => 'a123'
  alert(m1); // => '1'
  alert(m2); // => '2'
  return m3;
});
alert(s);    // => ___3___
```

例外処理

```
try {
  throw new Error("エラー！");
} catch(e) {
  alert("エラーが発生しました！！¥n詳細: " + e.message);
} finally {
  alert("おしまい");
}
```

throwで投げられるオブジェクトに制限はありません。

```
try {
  throw "aaa";
} catch(e) {
  alert(e);
}
```

エラーの種類を判別するには以下のようにします。

```
try {
  doHogeHoge();
} catch(e) {
  if (e instanceof EvalError) {
    // do something.
  } else if (e instanceof RangeError) {
    // do something.
  }
}
```

Firefoxだと以下のような書き方もできます。

```
try {
  doHogeHoge();
} catch(e if e instanceof EvalError) {
  // do something.
} catch(e if e instanceof RangeError) {
  // do something.
} catch(e) {
  // other error
}
```

JavaScriptのオブジェクト指向

JavaScript はプロトタイプベースのオブジェクト指向言語です。変数に代入できるものは~~全て~~undefinedなどの特殊な値を除きほとんどがオブジェクト~~です。~~として取り扱うことができます。(数値や文字列, true, falseなどはプリミティブ値と呼ばれオブジェクトではありませんが, ドット演算子でプロパティにアクセスできるなど, オブジェクトとして取り扱うことができます。これは, オブジェクト的な振る舞いが必要とされる場合, プリミティブ値が自動的にオブジェクトに変換されるからです)。

オブジェクトの定義

```
var obj = {a: 123, b: 3};
obj.a;    // => 123
```

メソッド

メソッドはプロパティに関数を代入したものです。

```
var man = {
  hello: function() { alert('hello!'); },
  bye: function() { alert('bye'); }
};
man.hello();  // => hello!
```

クラスのようなもの

関数を作りその prototoype プロパティをいじることでクラスのようなオブジェクトを作ることができます。

```
// クラス（のようなもの）の定義

// コンストラクタとなる関数
var Man = function(name, age) {
  // プロパティの初期化
  this.name = name;
  this.age = age;
};
// メソッド・プロパティの定義
Man.prototype = {
  sayName: function() { alert("My name is " + this.name + "."); }
}

// インスタンスの作成
var bob = new Man('Bob', 35);
bob.sayName();  // => My name is Bob.
```

継承

prototype に親クラスのオブジェクトを代入することで継承が実現できます。

```
var Animal = function() {};
Animal.prototype = {
  sleep: function() { alert('zzz...'); }
};

var Human = function() {};
Human.prototype = new Animal();
Human.prototype.workHarder = function() {
  alert("I'm tired...");
  this.sleep();
};

var me = new Human();
me.workHarder();  // => I'm tired... => zzz...
```

雑多なtips

真偽値

JavaScript では以下の値が false として扱われます。

- false (Bool型)
- 0 (実数型)
- "" (空文字)
- null, NaN, undefined

これ以外は全て true として扱われます。

==と===

==による比較は自動的に型変換が行われてから比較されるため，意図せぬ結果をもたらす場合があります。

```
'0' == 0;     // => true
'' == 0;      // => true
'100' == 100 // => true
```

このような事態を防ぐために，型変換を行わない比較演算子 === を用いるとよいでしょう。

```
'0' === 0;     // => false
'' === 0;      // => false
'100' === 100 // => false
```

~~for in文の落とし穴~~既存のオブジェクト拡張の落とし穴

for in 文には prototype で定義されたプロパティも列挙してしまうので注意が必要です。

```
Object.prototype = {doHoge: function() {}};
var obj = {a: 1, b: 2};
for (var i in obj) {
  alert("i = " + obj[i]); // => "a = 1", "b = 2", "doHoge = function() {}"
}
```

このような挙動を回避するためには， hasOwnProperty を使ってそのオブジェクト自体が持っているプロパティかどうかを確認しましょう。

```
Object.prototype = {doHoge: function() {}};
var obj = {a: 1, b: 2};
for (var i in obj) {
  if (obj.hasOwnProperty(i))
    alert("i = " + obj[i]); // => "a = 1", "b = 2"
}
```

既存のオブジェクト，特に全てのオブジェクトのプロトタイプである Object.prototype を拡張する場合は効果範囲が大きいので慎重に行ってください。

変数のスコープ

変数のスコープは宣言した関数内全体になります。関数内のどの位置で変数宣言しても，関数内全体からその変数を参照できます。

```
var a = 0;
function() {
  alert(a);   // => undefined（関数内で後に定義したaを参照するため）
  var a = 1;  // ここで a に値が代入される
  if (true) {
    var b = 1; // ここで定義した b は関数内ならどこからでも参照出来る
  }
  alert(b);   // => 1（if文の中で宣言した b を参照できる）
}
```

Firefox だと以下のようにしてスコープがブロック内に限定される変数を宣言することができます。

```
if (true) {
  let b = 1;
}
alert(b);    // => undefined（letで宣言するとスコープがif文の中に限定される）
```

thisの指す物

メソッド内で使われている this が指すものは，そのメソッドの呼び出され方によって変わります。具体的には，obj.hoge(); の形で呼び出された場合，thisはobjになり，hoge();の形で呼び出された場合はthisはwindow（グローバルオブジェクト）になります。

```
var smith = {
  name: "Smith",
  sayName: function() { alert(this.name); }
};
smith.sayName(); // => Smith
var john = {name: "John"};
john.sayName = smith.sayName; // 関数を代入
john.sayName();  // => John

var sayName = john.sayName;
sayName();       // => undefined（thisがwindowを指すためthis.name == window.name == undefined）
```

thisの指すものを明示的にしていして関数を呼び出すには， Function#call や Function#apply を使います。

```
// 上のコードの続き
sayName.call(smith);  // => Smith（thisがsmithを指す）
sayName.apply(john);  // => John（thisがjohnを指す）
```

Function#call と Function#apply の違いは，呼び出したい関数に渡す引数の指定の仕方です。applyは配列として引数を渡してやります。

```
function hoge(a, b, c) {
  return a + b + c;
}
// 以下は同じ意味
hoge(1, 2, 3);                  // => 6（this は window を指す）
hoge.call(null, 1, 2, 3);       // => 6（this は window）
hoge.apply(null, [1, 2, 3]);    // => 6（this は window）
```

既存のオブジェクトの拡張

既存のオブジェクトの prototype をいじることで既存のオブジェクトを拡張できます。

```
[1, 2, 3].sum(); // => Error (sum is not a function)
Array.prototype.sum = function() {
  var sum = 0;
  for (var i = 0; i < this.length; i++) {
    sum += this[i];
  }
  return sum;
};

[1, 2, 3].sum(); // => 6
```

JavaScript参考資料

言語リファレンス

MDCのドキュメントがよくまとまっていて使いやすいです。

- JavaScript - MDC

JavaScriptのオブジェクト指向について

自分は以下のページでオブジェクト指向を学びました

- オブジェクト指向プログラム言語としてのJavaScript
- Latest > Flakes of Ideas > JavaScriptでDOMを使う――オブジェクト指向入門の入門 - outsider reflex

Permalink | コメント(10) | トラックバック(21) | 19:49 1249 users

コメントを書く

os0x 2010/01/31 22:56
ブックマークレットは;で区切れば複数行相当のJavaScriptも実行できますし、どちらかというと文字数の制限のほうが厳しい(特にIE6は500文字ちょっと)かなと思います。
print文はないですが、最近はconsole.logをサポートしているブラウザが増えているのでalertを使うことはかなり減っていると思います。IE6でデバッグしないといけないときくらいでしょうか。
substrはECMAScriptの仕様には含まれていない、非標準なメソッドなので一応注意。まあ普通に使えますし、使えなくなる気配はないので気にするほどのことではないですが。
> 変数に代入できるものは全てオブジェクト
オブジェクトではないもの(undefinedなど)も代入はできるので、代入できる=>オブジェクトだと誤解があるように思います。
真偽値の"0" (文字列型)は偽として扱われませんよ。数値に変換したときは0になるので、0は偽になりますが、
javascript:alert(("0")?true:false);はtrueです。
あと、個人的には「for in文の落とし穴」は「既存のオブジェクトの拡張」の落とし穴であって、prototypeは汚さないルールの方が実用的かなと思います。

gifnksm 2010/02/01 00:13
>>os0xさん
たくさんのご指摘ありがとうございます。
さっそく本文の方に反映させて頂きました。

いつも Greasemonkey でやりたい放題スタイルな JS ばっかり書いているので，そこから離れた部分のことを書くとどうしても怪しい部分が出てきてしまいますね （標準関数とかprototype汚染とか真偽値とか）

t_takata 2010/02/01 15:38
些細なことなのですが文字列の長さの所、jplenの方に.lengthが無いように見えます。

gifnksm 2010/02/01 17:15
>>t_tanakaさん
おお。たしかに抜けてますね。直しておきました。
ご指摘ありがとうございます。

通りすがり 2010/02/01 22:27
基礎文法という意味では、正規表現リテラルを解説しておいた方がよい気がします。
頻繁に使われる上に、他の言語を知っている人ほど混乱する文法だと思いますので。

gifnksm 2010/02/02 03:56
>>通りすがりさん
たしかにJavaScriptの正規表現まわりはちょっと注意が必要かもしれませんね。
ちょっと詳しく書いてみました。

favril 2010/02/02 06:04
細かいところですが、タイポ部分をいくつか。

[代入演算子とインクリメント・デクリメント]
var c,d,eのコメント部分

[文字列]のシングルクオート
var b = 'abc\tdef"; // "abc[tab]def"

[文字列操作] - [長さ]
'abcdef'.length は、5ではなく6です。

[配列の操作] - [部分コピー]
var c = ary.slice(1, 2);
の c は [2, 9] ではなく [2] が正しいです。

[switch文]
case 3: alert('two of three'); break;
は、'two or three' ですかね？

あと、タイポではないですが、
文字列操作の結合、分割の結果もコメントである方がいいかなーとか、
if-elseのとこで、「else if」もあるといいかなーとか思いました。

gifnksm 2010/02/02 06:49
>>favrilさん
うはー。我ながらタイポ多すぎですね。修正致しました。
あと，文字列操作のところや else if についても記述あった方がたしかに親切ですね。
追記しておきました。
コメントありがとうございます！

os0x 2010/02/02 19:29
callとapplyはthisを指定できますが、オブジェクトでないものを渡した場合、thisがグローバルオブジェクトになります。
javascript: function a(){ alert(this) } a.call(null);
はwindowオブジェクトがアラートされます。
ただ、ECMAScript5のStrictモードではnullになります(たぶん…)。Strictモードでは勝手にthisがグローバルオブジェクトになることがないことになっています。
Strictモードはまだ実装が存在しないので、あえて取り上げる必要はないかなとも思います。不確かな情報になってしまいがちなので。

gifnksm 2010/02/02 20:02
>>os0xさん
再びのご指摘ありがとうございます。
callとapplyにオブジェクト以外のものを渡した時の挙動については渡した物がそのままthisになうると思い込んでいました。
現状の仕様だとthisは常にオブジェクトになるのですね。

Strictモード関連のお話は興味を持ってくださった方もいるようなので，冒頭部で「まだ実装がない」ということを強調しておこうと思います。

なまえ メール(非公開) URL

画像認証 画像内の文字列を入力して下さい

投稿

トラックバック - http://d.hatena.ne.jp/gifnksm/20100131/1264934942

燈明日記 - 基礎文法最速マスターぞくぞくキターーー！
わだいのたけひこのざっき - 研究室生活 基礎文法最速マスター
医者を志す妻を応援する夫の日記 - Brainf*ck基礎文法最速マスター
何かしらの言語による記述を解析する日記 - Java基礎文法最速マスタ...
何かしらの言語による記述を解析する日記 - Bash基礎文法最速マスタ...
何かしらの言語による記述を解析する日記 - VBA基礎文法最速マスター
CX's VBScript Diary - VBScript 基礎文法最速マスター
surume000の日記 - プログラミング言語基礎文法最速マスターまとめ
きまぐれメモ - 各種言語による基礎文法最速マスターまとめ
src's note - 気になる技術メモ
My Bookmark - 2010/02/01(月)の出来事
shikaku's memo blog - ○○基礎文法最速マスター
永遠に未完成 - Vimスクリプト基礎文法最速マスター
[Flash] ActionScript 3.0 基礎文法最速マスター
CHOLOG - くだらないつぶやきのLOG | twitter本日分post
(rubikitch loves (Emacs Ruby CUI)) - Emacs Lisp基礎文法最速マス...
どうでもいい情報置き場 - Whitespace基礎文法最速マスター

[C#]C#基礎文法最速マスター
TechTalkManiacs - JavaScript基本概念最速マスター
Life like a clown - はてな的プログラミング言語人気ランキング
jQuery基礎文法最速マスター

リンク元

3195 http://b.hatena.ne.jp/
1177 http://b.hatena.ne.jp/hotentry
735 http://reader.livedoor.com/reader/
502 http://b.hatena.ne.jp/hotentry/it
374 http://ig.gmodules.com/gadgets/ifr?view=home&url=http://choichoi.sakura.ne.jp/hatena_bookmark4.xml&nocache=0&up_num_feed=30&lang=ja&country=jp&.lang=ja&.country=jp&synd=ig&mid=61&ifpctok=4030979726874175731&exp_split_js=1&exp_track_js=1&exp_new_js_flags=1
342 http://www.sleipnirstart.com/
317 http://www.google.co.jp/reader/view/
272 http://d.hatena.ne.jp/
260 http://www.google.com/reader/view/
243 http://ig.gmodules.com/gadgets/ifr?view=home&url=http://www.hatena.ne.jp/tools/gadget/bookmark/bookmark_gadget.xml&nocache=0&up_display_item=10&up_display_thumbnail=false&up_display_summary=false&up_category_all=true&up_category_general=false&up_category_

おとなり日記

2010-02-01 てっく煮ブログ 4/33 12%
2010-02-02 おもしろ日記パワー 4/40 10%
2010-02-02 TechTalkManiacs 5/57 8%
2010-02-02 use GFx::WebLog; 6/91 6%
2010-02-02 $koherent->diary 5/74 6%

<[Ema Meadow]C-aで「行頭」と... | [ニコニコ動画][Greasemonkey]ni...>