JavaScript マニュアル
入 門 編
JavaScript 構文概論 と 記述 まで

1.JavaScript 概論

➢ JavaScript の特徴

JavaScript は、HTML に組み込むだけで Web ページをインタラクティブに動作させることができるインタープリタ型スクリプト言語のひとつです。

スクリプト言語とは、大まかにいってしまえばソフトウェアを制作するときなどに使用するプログラム言語をよりシンプルにしたもので、コンパイル（＝プログラムをコンピュータが実行可能な形に変換する）することなくアプリケーション上で動作します。
その為、Web ページに組み込む際にも特別なツールを必要とせず、HTML と同様にテキストエディタと確認用の Web ブラウザさえあれば手軽に実装可能です。

また JavaScript 対応のブラウザであれば、実行にプラグインなども使用せずにユーザーのブラウザ上で処理を行えます。
CGI のようにサーバーサイドが処理を行ってその値を返すものと比べて、通信の時間も必要なく、多少複雑な処理を行っても動作は軽快でストレスを感じさせません。

但し、仕様で SSL を扱えないなどさまざまな制約があるため、処理の内容によってほかのスクリプト言語と使い分ける必要があります。

➢ ブラウザごとに異なる対応

一般に Web ページをダイナミックに変化させるためのスクリプト言語を一口に「JavaScript」と呼んでいますが、そもそも JavaScript は Netscape 社が Sun Microsystems 社と共同開発を行ったもので、厳密には Netscape に搭載されているスクリプト言語のことを指します。

Internet Explorer に搭載されているスクリプト言語は JScript といい、これらは共通する部分も数多くありますが、HTML と同じく、そのブラウザでしか実行できないものも少なくありません。
そのため一方では正しく動作するのに、もう一方では動作しないという問題が持ち上がりました。

現在では Netscape 社や Microsoft 社、Sun Microsystems 社などが参加した ECMA（European Computer Manufactures Association）という団体が標準化を行い、JavaScript1.1 をもとにした ECMAScript(ECMA-262)が仕様としてまとめられています（2004 年７月段階では第３版）。

Web ブラウザ開発時のスクリプト言語実装の規格となっており、各ブラウザが準拠していますが、JavaScript1.2 以降で追加された Script も多く、結局はすべてのブラウザで同じように動作させることはまだまだ難しいのが現状です。

2.JavaScript の組み込み方

➢ JavaScript の組み込み方

JavaScript を HTML 文書に組み込むためには、いくつかの方法があります。

まず、<script>タグで「以下は Script の記述である」ということを宣言して組み込む方法です。
この場合、language 属性で「JavaScript」または type 属性で「text/JavaScript」として JavaScript であることを明確にします。Explorer や Netscape では、これらの属性を省略するとデフォルトで JavaScript と判断して動作しますが、他のブラウザやスクリプト言語を併用する可能性などを考慮して、明示しておいた方がいいでしょう。
ただし、language 属性は W3C より将来サポートされなくなることが勧告されています。

type 属性が推奨されていますが、古いブラウザや一部のブラウザでは対応していません。
また、src 属性を使用して、外部 Script ファイル（拡張子.js）を読み込むことも可能です。

```
<script type="text/javascript" language="JavaScript">
<!--
   alert("Hello World!");
<//-->
</script>

<script type="text/javascript" language="JavaScript" src="myScript.Js"></script>
```

➢ HTML のタグに組み込む

<script>タグを使わずに組み込むこともできます。
これは、HTML のタグの属性としてイベントハンドラを設定し、その中に Script を記述する方法です。
これは主に関数を呼び出す場合に使用しますが、メソッドなどを記述しても構いません。

この方法は手軽に組み込めることができますが、どこに Script を記述したのかわかりにくくなる上、同じ処理を別の場所で行いたい場合は同じ記述を何度もしなければなりません。それぞれの特性を考えて、状況によって使い分けをする必要があります。

```
<input type="button" value="Click!" onClick="alert('Hello World!')">
```

3.JavaScript の記述のルール

➢ JavaScript の記述位置

JavaScript は基本的に以下の部分に記述できます。

- ✓ <head> </head>タグ間
- ✓ <body> </body>タグ間
- ✓ HTML タグ

HTML タグは前述したとおり要素の属性としてイベントハンドラを設定し、その値として Script を記述する方法です。<body> </body>タグ間は、<script>タグで組み込む記述方法になります。これらのうちのどこに記述するかは自由ですが、JavaScript は「組み込まれた順に実行する」インタープリタ型のスクリプト言語のため、それぞれ注意が必要です。

<head> </head>タグ間に記述する方法は、Web ページの読み込み時に実行されます。Web ページ内の要素が表示されるよりも先に実行されるため、ページ内の要素に対して操作を行いたい場合は関数を定義しておく必要があります。スクリプトを１箇所にまとめて記述でき、管理もしやすいため、ここに記述されるケースがほとんどです。

<body> </body>タグ間の場合、実行させたい場所に記述をすれば、タイミングは自由に設定できるという利点があります。しかし、たとえば以降に現れる関数などを呼び出すことはできないため、ほかの処理との連携などは複雑になりがちです。また、処理があちこちに散ってしまうと管理もしづらく、混乱のもとになります。明確な理由がない限り、<head> </head>タグ間に記述しましょう。

➢ スクリプトの表記

JavaScript では記述する際のルールがいくつか存在します。これらを守らないと正しく動作しません。

✓ 大文字小文字を認識する

JavaScript では、大文字と小文字を明確に区別します。
たとえば、「window.moveTo()」というメソッドを「window.Moveto()」と記述すると動作せずにエラーが出ます。さらに、変数「a」と「A」は別のものであると判断されます。

✓ 使用できる文字は半角英数字のみ

JavaScript で使用できる文字は半角英数字のみです。
ただし、” ”（ダブルクォーテーション）または、’ ’（シングルクォーテーション）で括られた文字は文字列オプションとして扱われるため、2バイト文字も使用可能になります。ブラウザによっては全角のスペースが入っているだけでエラーになるものもあるため、注意が必要です。

✓ 処理の区切りにはセミコロン

ここで処理が一区切りするという場合には改行が入っていてもセミコロンがなければ一文とみなされます。セミコロンがなくともブラウザが自動的に区切りを判断しますが、if 文や for 文などの複雑な処理の場合、判別できずにエラーになることがあります。

✓ 行頭や演算子との間にはスペース挿入可能

行頭や演算子との間には半角のスペースや改行を自由に入れることができます。
これはスクリプトの実行になんの影響も与えません。関数や if 文などで処理を入れ子にしておくと、記述やデバッグがしやすくなります。

✓ 命令法則

JavaScript には変数や関数など、任意に名前をつけられるものが数多く存在します。
その際につけられる名前は基本的には自由ですが、半角英数字とアンダーバー以外を使用したもの、1文字目が数字、予約語はつけられません。

4.オブジェクト

➢ オブジェクトとは

JavaScript の記述の基本は「何の・何は・どの状態」「何が（何を）・どうする」ということを指定する事です。

この「何の」や「何が（何を）」に相当するのがオブジェクトです。
オブジェクトというと難しそうに聞こえますが、要はモノを構成するパーツだと考えればよいでしょう。

たとえば車で考えてみると、「車体」「ライト」「ハンドル」「タイヤ」「ドア」「窓」など、さまざまなパーツ（＝オブジェクト）が集まって「車」を構成しています。「エンジンを動かしなさい」「減速させなさい」というような指示を常に与えることで車は動いています。

Web ページも同様で、「ブラウザ」「文字」「数字」「画像」「フォーム」などのパーツが集まって構成されています。これら1つ1つがオブジェクトで、これらに指示を与えることで動的な動きを実現させているのです。

このため、JavaScript では、常に対象となるオブジェクトを指定しなければなりません。
たとえば、ウインドウの位置を移動させたりという場合は、window.moveTo(100, 100); といったふうに指示をブラウザに与えています。

このような「構成する要素をオブジェクトとして分解・整理し、それぞれのオブジェクトごとに動作させる」という考え方を一般的に「オブジェクト指向」と呼びます。JavaScript はオブジェクト指向の要素を持ったスクリプト言語です。

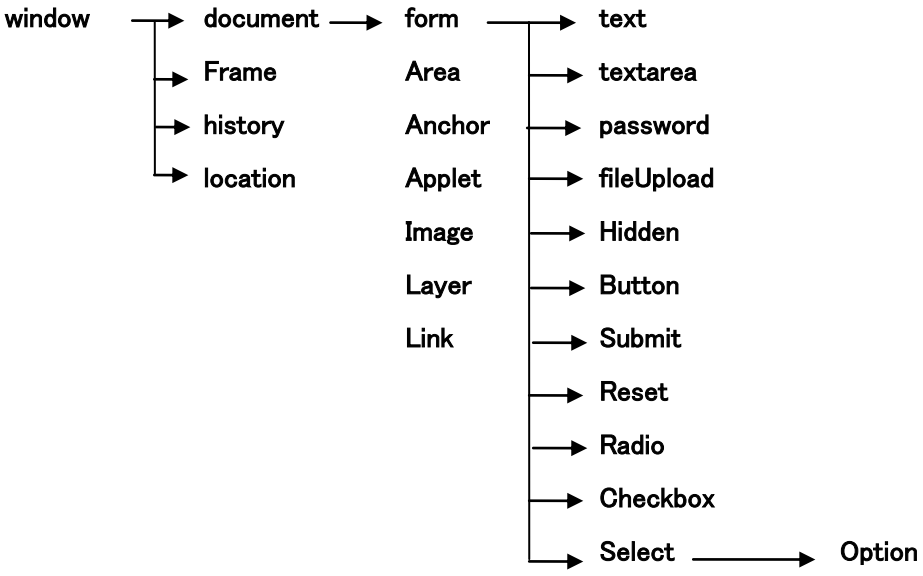
➢ オブジェクトの種類

「常にオブジェクトを指定する」ということは、逆に言えば、JavaScript で扱えるものは全てオブジェクトであるということです。
JavaScript では「ブラウザ」や「フォーム」のほかに、「時間」や「履歴」など目に見えないものもオブジェクトとして扱えます。

オブジェクトはそれぞれ特性があり、異なる役割を持っています。
たとえば、車で例えるとエンジンは動力を発生させるためのものですし、ハンドルは方向転換させるためのもので、それ以外の使われ方はしません。
オブジェクトも同様に、それぞれの持つ役割についてよく知っておく必要があります。
また、JavaScript のオブジェクトには大きく分けてナビゲーターオブジェクトと、ビルトインオブジェクトの2種類があります。

5.ナビゲーターオブジェクト

ナビゲーターオブジェクトはブラウザの持つ本来の機能や情報をオブジェクト化したもので、ウインドウや画像、フォームなどを扱うことができます。ナビゲーターオブジェクトは階層状になっており、上位オブジェクトは下位オブジェクトを自身のプロパティ(データ)として持っています。下位オブジェクトに対して指示を与えたい場合は、上位オブジェクトから順にドット(.)をつないで記述していきます。



screen

event

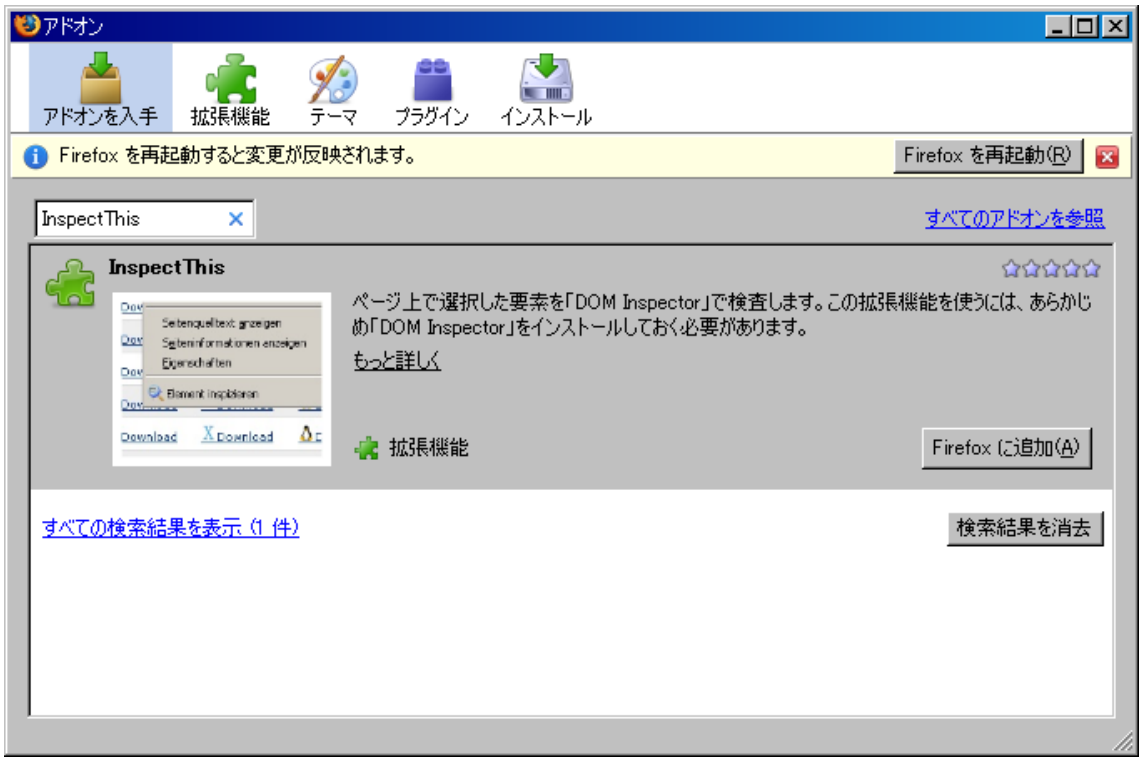
navigator → plugin

Mimetype

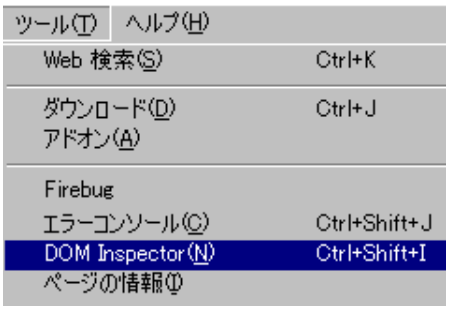
たとえば、最下位オブジェクトである text オブジェクトに指示を与えたい場合、
window.document.form オブジェクト名.text オブジェクト名.プロパティまたはメソッドのように記述します。

これは、「window オブジェクト内の document オブジェクト内にある form オブジェクトのさらに内部にある text オブジェクト」という意味になります。かなり面倒ですが、これを正しく指定していかないとブラウザはなにに対して指示を行ったのか認識せず、動作しません。ただし、window オブジェクトは最上位オブジェクトのため、省略可能です。

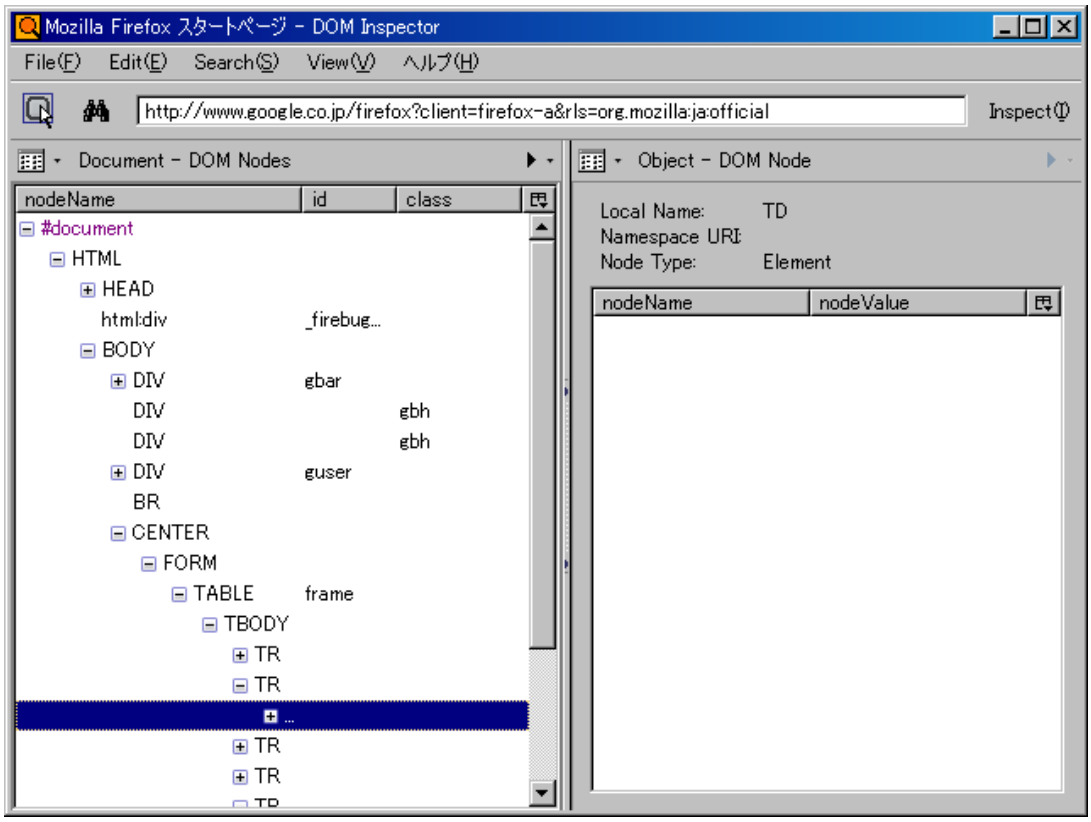
Firefox の DOM Inspector アドオン　を使うと階層がわかりやすくなります。
Firefox のインストールなどは HTML マニュアル基礎と応用を参照してください。



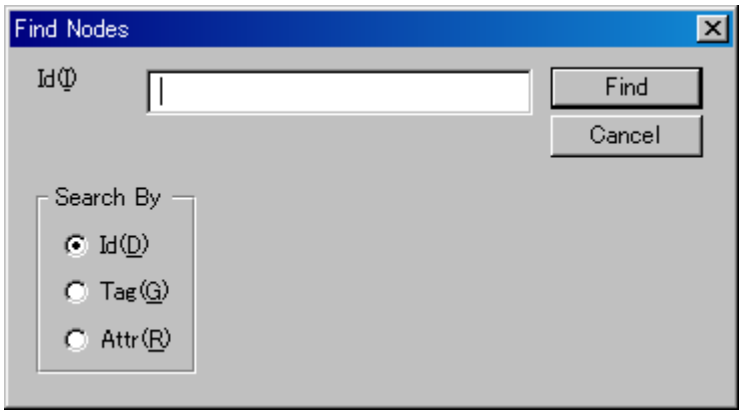
アドオンとして InspectorThis を追加して下さい。



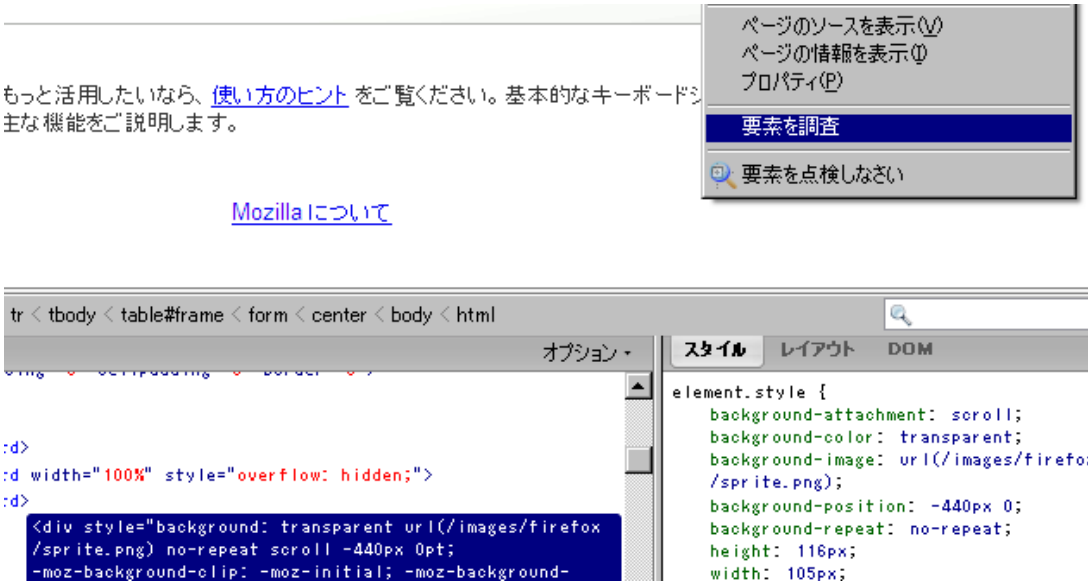
ツールから DOM Inspector を起動してみましょう。



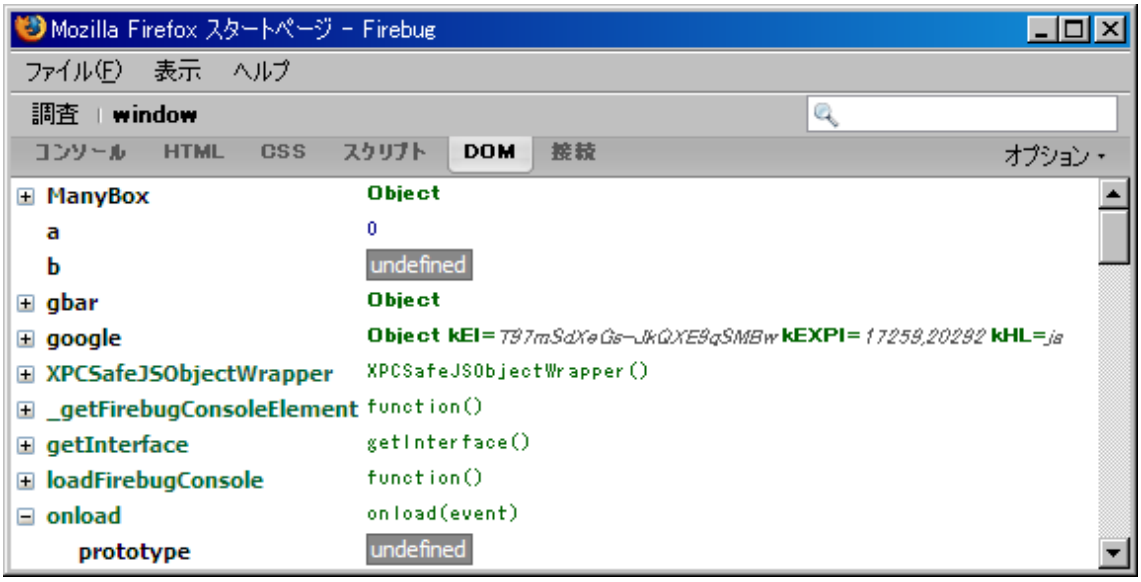
操作には慣れが必要ですが、上記の様に階層が見れます。また Id, Tag, Attr から検索もできます。



InspectorThis アドオンは要素を右クリックして DOM 表示ができます。



Firebug から DOM タグでもプロパティの参照などができます。



こういったツールを使いこなすようになると開発スピードが格段にアップします。
使いこなせるようになるまでが大変と言えますが、早めに慣れるようにしましょう。

最初のうちは、alert(オブジェクト.プロパティ); や、document.write(オブジェクト.プロパティ); でひとつひとつ値を確認してどこにどの値がはいっているのかを確認するといいでしょう。

DOM をつかいこなせるようになると Google Map などといった高度な非同期通信（Ajax）や jQuery, Prototype 等といった JavaScript ライブラリの理解も早くなります。

それまでは少しずつトライ＆エラーを繰り返していきます。ひとつひとつやったほうが確実であり、近道です。

6.ビルトインオブジェクト

ブラウザ内の要素をオブジェクト化したナビゲーターオブジェクトに対して、JavaScript が使用できるようにあらかじめ独自にブラウザに組み込むオブジェクトをビルトインオブジェクトといいます。ビルトインオブジェクトのなかには、日付や時間などの情報を持つ Date オブジェクトや数学的な計算を行える Math オブジェクトなどがあります。

ナビゲーターオブジェクトはページ内に存在する要素をそのまま使用できますが、ビルトインオブジェクトを使用するためには新たにそのオブジェクトを生成しなければなりません。オブジェクトを生成するには new 演算子を使用して、

new オブジェクト名 ( ) ;

と明示的に生成します。
なかにはオブジェクトの値を持つものを使用しただけで自動的に生成されるものもあります。

ビルトインオブジェクト一覧

オブジェクト	説明
Array	配列オブジェクト
Boolean	真偽オブジェクト
Date	日付オブジェクト
Function	関数オブジェクト
Math	算術オブジェクト
Number	数値オブジェクト
Object	Object オブジェクト
RegExp	正規表現オブジェクト
String	文字列オブジェクト

オブジェクトの生成例

```
myDate = new Date();    //オブジェクト指定
myVal = Math.random();   //Math と記述すれば自動生成
myNum = 1               //数値を使用すれば数値オブジェクトが自動生成
```

7.プロパティ・メソッド・変数

➢ オブジェクトの状態を表す「プロパティ」

オブジェクトはそれぞれ「数量」や「大きさ」など、そのオブジェクト独自の状態を持っています。
例えば image オブジェクトなら「画像の URL」や「幅」、「高さ」などです。
これがプロパティで、この値を参照してほかの処理に代入したり、新たに設定して状態を変化させる事が出来ます。

各オブジェクトは異なる特性を持っている為、使用できるプロパティもそれぞれ異なります。また、プロパティによっては参照だけが可能で、設定できないものもあります。

オブジェクトの説明の際に挙げた事の例でいえば、ハンドルには「色」や「大きさ」というプロパティを持っています。これに対してガソリンには「形」というプロパティはありませんが、「品質」や「残量」などといったプロパティを持っています。
さらに、ハンドルの「色」は塗り替える事も可能ですが、「大きさ」は測ることはできても変更する事は出来ません。

実際にプロパティを参照するには、

オブジェクト名. プロパティ名;

とします。

逆に設定する場合には、

オブジェクト名. プロパティ名 = 値;

のように記述します。

このときに設定できる値も各プロパティによって異なります。

プロパティの参照例

```
mybgCol = document.bgColor;      //背景色を参照設定するプロパティ
```

プロパティの設定例

```
document.bgColor = "#FFFFFF";
```

➢ オブジェクトに命令を出す「メソッド」

　オブジェクトはそれぞれ特性に合わせた「できること」を持っています。
オブジェクトに対して、「できること」を指定して命令を行うのがメソッドです。
例えば、スピードを出したいときは「エンジン」オブジェクトに「回転数を上げなさい」というメソッドを使用すればスピードは上がります。

メソッドを記述する場合には、

オブジェクト名．メソッド名（）；

と指定します。

　メソッドには必ずメソッド名に続いて（）が入ります。
　ここには、メソッドを実行する際に必要な情報を記述します。車の例でいえば、「どれだけ回転数を上げるのか」という情報がここに入ることになります。これを引数（パラメータ）といい、引数の内容はメソッドによって異なります。
複数の情報が必要な場合は、カンマ記号で区切って指定を行います。メソッドによっては引数を省略できたり、引数が必要ないものもありますが、その場合でも（）は必要です。

　メソッドによってはそれを行ったことに対して起きた変化をブラウザに返すものがあります。
その返ってきた値を「戻り値」と呼び、これをほかのオブジェクトに受け渡して複雑な処理を行うことが可能です。

メソッドのパラメータや戻り値、プロパティの参照・設定値で使用できる値には、いくつか種類があります。

引数の設定例

```
window.open("sub.html", "sub");    //新規ウインドウを開くメソッド
```

戻り値

```
sub = window.open("sub.html");
```

➢ 何でも入れられる容器、「変数」

変数はスクリプトを記述する際に欠かせない概念で、「データを入れておける器」のようなものです。
変数の中には数値や文字列はもちろん、オブジェクトやプロパティなど、どのようなものでも納めておくことが可能です。

変数を定義するためには、

```
var 変数名 = 値;
```

とします。

var は変数であることを明示する命令文ですが、JavaScript は初めて登場する単語を変数として認識する為、省略が可能です。

変数名は JavaScript の記述ルールで説明した命名法則に沿っていれば自由につける事ができます。
長い処理を記述する際は多くの変数が使用される為、何の値を格納している変数かをわかりやすくつけておくといいでしょう。

変数は定義後も異なる値を代入していく事によって、中身の内容が変化していきます。変数に代入を行うには、

```
変数名 = 値;
```

とします。

常に右辺の値が左辺に代入されます。

8.関数

➢ 一連の処理を登録できる「関数」

動的な Web ページを制作していると、同じ処理を何度もくり返し行いたいことがあります。
そのような場合、いちいち同じ処理を何度も記述するのは不効率ですし、ソースコードも非常に長くなってしまいます。関数は、1つまたは複数の Script の処理をまとめておき、任意の名前をつけて管理するものです。一度関数を定義しておけば、その処理を行いたい場合に関数名を記述するだけで何度でも実行させる事ができます。つまり、メソッドを自作できるわけです。

たとえば車を運転する場合、左折するには、

1.減速する
2.ウィンカーを左に出す
3.ハンドルを左に切る
4.加速する

というような流れで左折を行います。
例えば自分が助手席に座っているとき、ドライバーにいちいちそのような動作を指示しなくても、「左折して」と一言いえばドライバーは上の動作を行って左折してくれます。
関数に関しても同じ事と考えればわかりやすいでしょう。

➢ 関数の定義方法

実際に関数を定義するためには、

```
function 関数名(引数) {
    処理の内容;
}
```

とします。
まず、function を使用して「以下の処理は関数である」ということを宣言し、続けて関数名をつけます。

関数名は変数名と同じく命名法則に従っていれば自由につけることができます。
どのような処理を行っているのかを端的に表した、わかりやすい関数名をつけるようにしましょう。

さらに、()で引数名を指定。メソッドと同じく、引数が必要ない場合でも()は必須です。

{ }で囲まれた部分が実際に行われる処理になります。関数の内部にほかの関数を入れ子にする事もできます。

➢ 関数定義時の注意点

関数は、定義しただけでは実行されません。実行する為には、実行位置に関数名を記述して呼び出します。JavaScript は記述された際に実行されるインタープリタ型のスクリプト言語である為、関数を呼び出すにはそれ以前に定義されていなければいけません。

<head> ～</head>タグ間に記述しておけば、Web ページ読み込み時にすべての関数が定義される為、ここに記述しておく方法が一般的です。　　また、関数内で変数を定義すると、その変数は「ローカル変数」となります。ローカル関数はほかの関数　　などで参照することができず、もし参照しても未定義の変数とされてしまいます。これに対して関数外で定義された変数は「グローバル変数」と呼び、どこからでも参照することが可能です。さらに、ローカル変数は関数が実行される度に値が初期状態に戻ることになります。

複数の関数間で使用する様な値や、変化していく値を使用する場合は関数外で定義する様にしましょう。

```
myFunc();        //定義前の関数は呼び出せない
myleft = 200;   //グローバル変数

function myFunc() {
   myTop = 100;                          //ローカル変数
   window.moveTo(myleft, myTop);      //グローバル変数はどこでも参照可能
}
window.alert(myTop);   //ローカル変数は関数外の参照は不可
```

➢ 引数の使用方法

変数を使用するメリットは、一度定義しておけば何度でも同じ処理を繰り返し使用できて記述を簡略化できる点にあります。しかし、一部の値が異なるだけで大部分の処理は同じという場合、いちいち異なる部分だけを記述し直した関数を定義していたのではそのメリットを享受することはできません。

このいう場合は、引数をしようすれば解決します。先ほど挙げた車の運転例でいえば、左折するのも右折するのも方向が異なるだけで、行動自体はほとんど同じなのです。ここで「方向」という引数を設定します。

```
function 転換(方向) {
   減速する;
   ウインカーを方向に出す;
   ハンドルを方向に切る;
   減速する;
}

転換(右);
```

この様にに左右どちらかの値を引数で指定してやればいいわけです。

```
function myFunc(myX, myY) {
   window.moveTo(myX, myY);   //引数を使用する場所に引数名を記述
}

myFunc(100, 100);      //関数呼び出し時に引数の値を入力
```

9.イベント・イベントハンドラ

➢「いつ実行するのか」を設定する

ユーザーが Web ページを閲覧しているときにブラウザに対してなにかしらのアクションをとると、「〇〇をしました」という報告がブラウザに返ります。これが「イベント」です。
イベントには数多くの種類があり、例えば画面をクリックすれば「click」が、マウスが押されたら「mousedown」が常に返っています。

このイベントを取得するのが「イベントハンドラ」で、このイベントハンドラを HTML の要素の属性やオブジェクトに設定しておけば、「このオブジェクトに対して〇〇が行われたら」ということが指定できます。
特に、関数は定義しただけでは処理は行われません。定義した関数はイベントハンドラを使って呼び出す方法が一般的です。

イベントハンドラ内には処理をそのまま記述することも出来ますが、処理が長くなると視認性が悪くなります。一度しか使われないような処理でも、関数にまとめた方が管理しやすくなります。

イベントハンドラ	説　明	イベントハンドラ	説　明
onAbort	読み込み中止	onMouseDown	マウスが押されたら
onBlur	フォーカスが外れたら	onMouseMove	マウス移動時
onChange	内容が変化したら	onMouseOut	マウスが離れたら
onClick	クリックしたら	onMouseOver	マウスが重なったら
onDblClick	ダブルクリックしたら	onMouseUp	マウスが離されたら
onDragDrop	ドラッグ &ドロップ時	onMove	ウインドウ移動時
onError	エラー発生時	onReset	フォームリセット時
onFocus	フォーカスされたら	onResize	ウインドウリサイズ時
onKeyPress	キーが押されたら	onSelect	テキスト選択時
onKeyUp	キーが離されたら	onSubmit	フォーム送信時
onLoad	ページ読み込み時	onUnload	ページ移動時

➢ イベントハンドラ記述の注意点

イベントハンドラはオブジェクトに対して記述します。
オブジェクトによって使用できるイベントハンドラは異なるので注意が必要です。

例えば、「フォームの内容をリセットしたら」という意味の onReset はフォームオブジェクトにしか使用出来ませんし、「マウスが離れたら」という意味の onMouseOut は Image オブジェクトには使用できません。
オブジェクトと使用できるイベントハンドラとの関係を把握しておきましょう。

HTML のタグ内にイベントハンドラを設定するときに気をつけなければならない事があります。
それは処理の途中で文字列を使用する場合です。

通常文字列はダブルクォーテーションで括りますが、タグ内ではイベントハンドラに続いて「 =″処理″ 」とします。その為、文字列にダブルクォーテーションを使用すると、ブラウザは文字列の最初の″で処理の記述が終了したと解釈してしまい、エラーになります。処理や関数の引数として文字列を使用する場合は、シングルクォーテーションで括ります。

オブジェクトに直接記述して関数を指定する場合、関数に引数が必要なければ()を記述する必要はありません。1つのイベントハンドラに複数の処理を設定する場合、処理間にカンマをつければいくつでも指定する事が可能です。

```
<body onLoad="alert('Hit!')">
```

10.JavaScript で使用できる値

メソッドの戻り値やパラメータ、プロパティ参照時に返ってくる値や設定時に使用する値など、JavaScript で使用する値にはいくつかの定期的な値があります。
ここでは一般的によく使用される値について説明します。

➢ オブジェクト

メソッドの戻り値として新たにオブジェクトを生成するものの場合、オブジェクトの参照が可能になります。例えば、新規ウインドウを開くメソッド window.open() などで新たなオブジェクトを変数に代入すればその後は変数名でオブジェクトを操作することができます。
この状態をオブジェクトへの参照と呼びます。

ウインドウオブジェクトの他には Array.concat() のような配列オブジェクト、Date.UTC() のように日付オブジェクトなどを返すものがあります。

厳密に言えば、この後に解説するブール値はブーリアンオブジェクト、文字列は文字列オブジェクト、数値は数値オブジェクトが返っている事になります。

記 述 例

```
subWin = window.open();
subWin.document.open();
subWin.document.write("Hi!");
subWin.document.close();
```

➢ ブール値

ブール値とは、「真（true）」または、「偽（false）」のどちらかの情報を持つ値です。条件の判断や、この値によって処理を振り分ける条件分岐に主に使用されます。オブジェクトの状態や処理を実行した結果が2パターンしかない場合（存在するかしないか、選択されているかいないかなど）メソッドの戻り値として返してきたり、プロパティの参照/設定値として使用します。また、1が true、0 または 1 以外が false として数値をブール値の代わりに使用できます。但し、Mac 版の Explorer では対応していません。

ブール値で参照・設定するものに、checkbox.checked、ブール値が返ってくるものに window.closed や window.confirm() などがあります。

記 述 例

```
document.myForm.mySea01.checked = true;
document.myForm.mySea02.checked = true;
document.myForm.mySea03.checked = false;
```

➢ 文字列

戻り値や参照・設定値として文字列が使用されるケースは非常に多く、内容によってさらにいくつかのパターンに分ける事が出来ます。
ここではその代表的なものを紹介します。文字列を使用する為には、使用したい文字をダブルクォーテーション(″″)または、シングルクォーテーション(')で括ります。

″″内になにも存在しない場合は、空文字として扱われます。

✓ URI

他のページにジャンプしたり、外部のファイルを表示させる場合などに使用します。
URI(Uniform Resource Locators)は、Web 上で参照するソースを識別するための文字列で、直接的にソースを指定する URL(Uniform Resource Locators)と名前を指定する URN(Uniform Resource Name)が含まれます。

指定を行う場合、絶対パス、相対パスいずれの形でも指定可能です。
絶対パスはプロトコル(「http:」「file:」など)から始まるもので、インターネット全体から見た視点でファイルの場所を参照し、相対パスは「現在のファイル」から見たファイルの場所を指定します。

参照値の場合、絶対パスで値が返ってきます。相対パスで設定をしたものを参照する場合でも絶対パスが返ってくるため、注意が必要です。

URI の値を使用するものとして、ハイパーリンクの情報を参照・設定する location.href、画像の URI を参照・設定する image.src などがあります。また、ページ内にアンカーを設定したり、すでにページ内に存在するアンカーの情報を取得することも可能です。

記 述 例

```
<script language="JavaScript"><!--
   function change_image(url) {
      document.myImg.src = url;
   }
//--></script>

～中略～

Photo Change<br>
<img src="" id="myImg">
<input type="radio" name="myRadio" onClick="url('img/sample01.jpg')">sample01
<input type="radio" name="myRadio" onClick="url('img/sample02.jpg')">sample02
<input type="radio" name="myRadio" onClick="url('img/sample03.jpg')">sample03
```

✓ カラー値

文字や背景などの色を指定する場合に使用する値です。
色を指定する場合、HTML と同様に2通りの指定方法があります。

✓ RGB の 16 進数値

16 進数値は、R(赤)・G(緑)・B(青)の値を、通常使用している 10 進数から 16 進数に変換したものです。
色は、RGB3色の値の組み合わせで表現されます。
通常は、それぞれ 0～255 の 256 段階(または 0～100 %)で表現します。
HTML で指定する 16 進数値は、この 0～255 の値を 10 進数から 16 進数(0～9、A～F の 16 段階で桁が上がる)の値に変換して、RGB の順に並べたものです。
指定時には、頭に「#」をつけます。

✓ カラーネーム

カラーの値を属性値としてとるものに、body 要素の bgcolor 属性や、font 要素の color 属性があります。

色見本	カラーネーム	16進数値
	Black	#000000
	Silver	#C0C0C0
	Gray	#808080
	White	#FFFFFF
	Maroon	#800000
	Red	#FF0000
	Purple	#800080
	Fuchsia	#FF00FF
	Green	#008000
	Lime	#00FF00
	Olive	#808000
	Yellow	#FFFF00
	Navy	#000080
	Blue	#0000FF
	Teal	#008080
	Aqua	#00FFFF

記述例

```
document.bgColor = "##FFCC33";
document.fgColor = "Teal";
document.linkColor = "Navy";
```

✓ 任意の文字列

オブジェクト名などあらかじめ設定しておいた文字列や、いわゆる「普通のテキスト」として製作者が自由に設定できる文字列です。対象オブジェクトのプロパティの値を参照・設定する場合や、メッセージを書き出す場合に使用します。

任意の文字列を設定できるものには、アラートを表示する window.alert()、ドキュメントに文字列を書き出す document.write() などがあります。

また、設定した任意の文字列を参照できるものには、各オブジェクト名を参照/設定するプロパティである name, 配列内の要素の値を連結する Array.join() などがあります。

記 述 例

```
Alert("Welcome to the NHK");
myVal = new Array("HTML+CSS", " ActionScript", " JavaScript");
document.write("現在のラインナップ..." + myVal.join() + "<br>");
```

✓ あらかじめ設定された文字列

ユーザーの環境など、あらかじめブラウザなどで決められた値を参照/設定するものです。

これらのなかにはブラウザ名や OS などのユーザーの環境、フレームのターゲット、MIME タイプ、言語コード、プロトコル、日付などを表す文字列など、さまざまなパターンがあります。

記 述 例

```
myVal = navigator.userAgent;
document.write(myVal + "<br>");
myPlat = navigator.platform;
document.write(myPlat + "<br>");
```

✓ エンコードされた文字列

メソッドのなかには、文字列を特定のコードに変換（エンコード）し、その値を返すものがあります。
例えば、Cookie などは2バイトの情報は特定の文字コードしか扱えない為、書き込むには文字列を変換した値を使用します。逆に読み込む場合は取得した値を再変換し、その値を使用します。

記 述 例

```
myEnc = escape("ビジュアル");
document.write(myEnc + "<br>");
myDec = unescape(myEnc);
document.write(myDec + "<br>");
```

➢ 数 値

JavaScript で使用できる数値は整数と浮動小数点です。
浮動小数点には整数と負数、さらにべき乗の識別子（E または e）の指数を持った 10 進数が扱えます。
10 進数のほかには、8進数、16 進数も扱うことができます。

✓ 計算結果

演算子を使用して計算した値のほかに、Math オブジェクトを使用してべき乗を求めたりする事が出来ます。また円周率や2の平方根など、決まった値を返すものもあります。
さらに、三角関数など数学的な計算も簡単に行えます計算結果の値は単純な数値のほか、角度で返す場合もあります。

記 述 例

```
myV = Math.Round(5 * Math.PI);
alert("直径 5cm の円周は約" + myV + "cm");
```

✓ 数量

オブジェクトやオブジェクト内の要素の数などを求めることが出来ます。
これを使用して対象オブジェクトの数だけ繰り返しを行う、ということも簡単に行えます。数を取得するには各オブジェクトの length プロパティを使用します。

記 述 例

```
myNum = document.images.length;
Alert("全" + myNum + "個の画像があります");
```

✓ インデックス値

配列の要素など、複数の値が存在するものの「順番」を表すのがインデックス値です。
ドキュメント内にフォームや画像などが複数存在する場合、これを使用すれば「〇番目のフォーム」など、オブジェクト名を使用せずに指定を行うことが可能です。
指定時にインデックス値を使用する場合は [ ] 内に数値を記入します。

インデックス値を使用する場合に気をつけなければならないのは、1 番最初の値のインデックス値は 0 になるという事です。
つまり、たとえば、3 番目の要素を取得したいという場合には、指定は 2 になります。
戻り値がインデックス値の場合、間違えやすいので注意しましょう。

記 述 例

```
myInd = "Visual".indexOf("g");
document.write(myInd + 1 + "文字目");
document.images[0].src = "sample.jpg";
```

✓ サイズ・位置

画像やウインドウなど、オブジェクトにサイズのプロパティがある場合、その値を参照設定することができます（オブジェクトにより異なります）。また、ウインドウはスクリーン上の位置を指定することも可能です。

これらは、いずれも値はピクセル単位の整数となり、小数点を使用することは出来ません。

記 述 例

```
window.resizeTo(350, 300);
window.moveTo(0, 0);
```

✓ 日時を表す数値

Date オブジェクトのメソッドを使用すると、日付や時間を表す数値が戻ります。
戻る値は、年、月、日、曜日を数字で表したものと、時間、分、秒、ミリ秒を表す数字です。
また、1970 年 1 月 1 日午前 0 時 0 分 0.0 秒からの経過時間をミリ秒単位で表す場合もあります。
これらはそれぞれローカル時間と UTC（世界協定時。厳密には異なるものですが、世界標準時間とほぼ同じものと考えて構いません）を戻すものの２通りのメソッドが用意されています。

これらを扱うときに気をつけなければならないのは、月と曜日を取得・設定する場合です。
月を取得した場合に戻ってくる値は実際の月よりも1少ない数が戻ってきます（1月=0、 12月＝11）。また、曜日を取得する場合は日曜日が0、月曜日が1、・・・・・土曜日が6となります。
曜日以外は設定することも可能ですが、その場合の戻り値は 1970 年 1 月 1 日午前 0 時からの経過ミリ秒になります。

また、取得する日時はユーザーのローカル環境から取得しています。その為、ユーザーの環境の時間が正しくなければ制作者の意図通りに動作しない場合もあります。

記 述 例

```
myDays = [″日″,″月″,″火″,″水″,″木″,″金″,″土″];
myDate = new Date(2005, 11, 25);
myY = myDate.getFullYear();
myM = myDate.getMonth() + 1;
myD = myDate.getDate();
myDay = myDays[myDate.getDay()];
document.write(myY + ″年″ + myM + ″月″ + myD + ″日″ + myDay + ″曜日<br>″);

myTime = new Date();
myH = myTime.getHours();
myMin = myTime.getMinutes();
document.write(″現在時刻は″ + myH + ″時″ + myMin + ″分″);
```

➢ そのほかの値

前述した値のほかにも、ある一定の値を返すものがあります。
これらはそれぞれ固有の意味をもっています。

✓ NaN

演算子や Math オブジェクトを使用して値を求める場合、引数の値が数値ではない（Not a Number）場合にこの値が返ります。
また、数値オブジェクトで NaN プロパティを使用すると、その値は数値以外の値になります。
（文字列オブジェクトなどになるわけではありません。）

✓ null

値がなにもない場合に戻ります。
null は値がなにもないという意味を持っているので、空文字や数値の 0 とは意味が異なります。
window.prompt()でユーザーがなにも文字を入力していなかったり、RegExp.exec()でマッチする文字列が存在しない場合などに null が返ります。

✓ undefined

値が見つからない場合や確定していない場合に戻ります。
定義しただけで値を設定していない変数などを参照した場合です。