

関数の名前付けのルールは、変数の名前のつけ方と同じものです(p.27参照)。具体的には、アルファベット、数字、アンダースコア(_)を使うことができ、大文字と小文字を区別します。また、名前の最初の文字は、必ずアルファベットまたはアンダースコアから始めます。

関数に渡す引数は、関数名の後の括弧内に「,」(カンマ)で区切って並べます。定義された関数が実行されるのは、関数が呼び出されたときのみです。

■引数の渡し方

引数は、実際にはどのように使えばよいのでしょうか？ 関数の記述例を見ながら解説しましょう。

```
function putDate(date) {
    document.write(date);
}
```

上記のputDate関数は、dateという引数を定義しています。これで、その関数内で引数dateを自由に使うことができます。また、関数内で引数を使うとき、その中に入っている値は、関数を呼び出す際に渡したもののが参照されることになります（関数の呼び出しについては後述します）。

ここで注意しなくてはならないのは、ある関数に引き渡した変数に関数内で変更を加えたとしても、元の変数の値は変わらないということです。

▼関数に引き渡した変数に関数内で変更を加えても、元の変数は変わらない

関数定義部

```
function printName(name) {
    name="John";
    document.write(name);
}

name="Mike";
printName(name);

document.write(name);
```

(1) 関数の読み込み
 4 nameを「Mike」から「John」に変更。
 そのため、(3)の実行結果は「John」と表示される

2 変数の定義
 3 関数の実行

5 関数内ではnameが変更されたが、
 ここでは「Mike」と表示される

※丸数字は実行順序

前ページの図では、printName関数内で、引数で与えられたnameの内容をMikeからJohnに変更していますが、関数を呼び出した元の値の内容までは変更されないことがあります。

■値を返す関数

関数は、処理した結果を呼び出し元に返すことができます。このとき、返すことのできる値は1つです。返される値を「戻り値」と呼びます。値を返すには、return文を使います。return文では、戻り値をreturnの後ろに1つ書きます。

return 戻り値;

これを実際の関数内で使うと、以下のような記述になります。

```
function Cube(num) {  
    cube=num*num*num;  
    return cube;  
}
```

なお、returnを記述すると、関数の処理はそこで終了します。

■関数を実行する

関数の定義は行いました。では、実際に関数を実行するには、どのようにしたらよいのでしょうか？

関数を実行することを「関数の呼び出し」といいます。関数は、値が返ってこない場合は、関数名と引数を書くことで実行できます。書式としては、以下のようになります。

関数名(引数);

たとえば、以下のような関数が定義されているとします。

```
function printName(name) {  
    document.write(name);  
}
```

このように定義されたprintName関数を実行する場合は、次のように記述します。

```
printName("John");
```

関数の呼び出しで注意しなくてはならないのは、関数を実行する前までに、関数の定義が行われていなければならぬということです。

関数を使ったサンプルは、以下のようにになります。

```
<html>
<title>execute function</title>
<script type="text/javascript">
<!--
function printName(name) {
    document.write("名前は"+name+"です。");
}
//-->
</script>
<body bgcolor="#fff8dc">
<h3>関数の実行</h3>
<hr />
<script type="text/javascript">
<!--
    var name="Mike";      //変数の定義
    printName(name);     //関数の呼び出し
//-->
</script>
</body>
</html>
```

このスクリプトを実行させると、次のような結果が得られます。

▼関数を呼び出すサンプル



また、値が返ってくる場合は、それを代入するための変数を用意します。したがって、関数呼び出しの書式は、以下のようにになります。

変数=関数名(引数);

たとえば、次のように定義された関数があるとします。

```
function Cube(num) {  
    cube=num*num*num;  
    return cube;  
}
```

この場合には、関数の呼び出しは、次のようになります。

```
answer=Cube(num);
```

この「answer」は、返ってくる値を受け取る変数です。実際は、好きな名前をつけてかまいません。

■変数の範囲

関数の引数についての注意は、変数がどの範囲で使えるかという問題と同じものです。ここで、変数はどの範囲内で使えるかについて解説します。これを、「変数のスコープ」と呼びます。

変数は、宣言された場所、初めて使われた場所によって、どの範囲で使えるかが決まります。その場所を大きく分けると、次の2つになります。

- ・関数内
- ・それ以外

```
<script type="text/javascript">
<!--
var Today="2001/9/1"; ● 関数外での宣言
function PrintBirthday() {
    var myBirthday="1973/9/1"; ● 関数内の宣言
    document.write(Today);
}
document.write(myBirthday);
//-->
</script>
```

関数の内部で宣言された変数は、その関数でしか使うことができません。そのため、外部から関数内部で宣言された変数を呼び出しても、定義されていない、または予期せぬ値を参照することになります。

それ以外の場所(<script>～</script>内)で宣言された変数は、どこからでも参照することができます。

```
<script type="text/javascript">
<!--
var Today="2001/9/1";
function PrintBirthday() {
    var myBirthday="1973/9/1";
    document.write(Today); ● 関数外で定義された変数
}
document.write(myBirthday); ● 関数内で定義された変数
//-->
</script>
```

このため、さまざまな関数内で共通に使いたい変数(たとえば日付などを格納するもの)は、関数の外で宣言しておきましょう。また逆に、関数内でしか使わない変数は、関数の中で宣言しておくのがよいでしょう。

オブジェクトモデル

JavaScriptのスクリプトを書くうえで、重要な役割を果たすのが「オブジェクト」です。いきなりオブジェクトといわれると、とても難しいもののように聞こえるかもしれません。しかし、一度その考え方を理解してしまえば、後はとても簡単に使いこなせるようになります。また、JavaScriptのオブジェクトの定義は、いわゆる「オブジェクト指向言語」を一部だけ抜き出したようなものなので、Javaを代表とするオブジェクト指向言語を少しでも知っていれば、より容易に理解できるでしょう。もちろん、まだ何も知らない人にもよくわかるように、これから詳しく解説していきます。

■オブジェクトとは

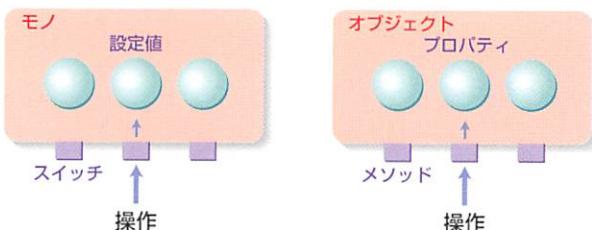
オブジェクトとは、いったいどのようなものなのでしょうか？ それは、「モノ」にたとえると、わかりやすくなります。日常生活をしていると、いわゆるモノをよく使います。たとえばテレビ、電話、カメラ、炊飯器、電子レンジ、……。これらはどれもモノです。これらのモノの仕組みは複雑ですが、実際にユーザーが用いるときには、簡単に使うことができます。なぜなら、それらを使うのに必要な操作はかぎられているからです。

たとえば、炊飯器を考えたとき、お米を炊くためにわざわざ自分で温度や時間などを設定し、さらにはその構造や電気の流れ方まで指定する必要はありません(古い炊飯器なら、もしかしたら温度や時間を設定しないといけないのかな？)。普段、そんなことを考えなくても炊飯器を使っているのは、炊飯器を使うための操作がかぎられているからです。「炊く」「保温する」……、といった操作は、その操作に対応するスイッチを押すだけで、ほとんど自動で目的の動作(お米を炊くなど)を得られます。このようなモノの概念をプログラミングの世界に取り入れたのが、オブジェクト指向なのです。

モノでは、自身が持っている各種の値を設定し、操作することができます。具体的にどのような値が設定できるかを挙げてみると、時間、温度、強さ……、といったものになります。また、それぞれの値を設定するには、スイッチが必要となります。ここで、オブジェクトをモノと対比して考えると、次のような対応関係になります。

- ・オブジェクト ←→ モノ
- ・メソッド ←→ スイッチ
- ・プロパティ ←→ 設定値

▼オブジェクトはモノと対比できる

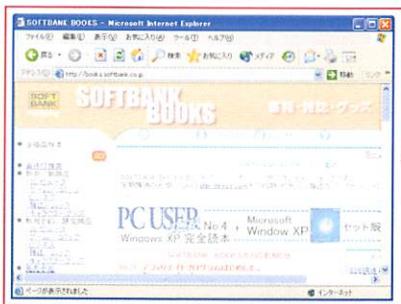


このように考えると、オブジェクトがどんなものか、なんとなくイメージが湧くのではないかでしょうか。要するに、プログラミングの世界でのモノに相当するオブジェクトを動作させるには、スイッチに相当するメソッドを使って操作することになります。

ここで、JavaScriptで扱うオブジェクトの中から、わかりやすい例を挙げましょう。JavaScriptでは、「Webブラウザ本体」「Webページが表示される部分」は、どちらもオブジェクトとして操作できます。

- ・Webブラウザの本体 ←→ navigatorオブジェクト
- ・Webページが表示される部分 ←→ documentオブジェクト

▼Webブラウザの本体や、ページ表示部分は、オブジェクトとして操作できる



navigatorオブジェクト



documentオブジェクト

ただし、JavaScriptは、JavaやC++などの純粋なオブジェクト指向から見ると、継承にあたる機能がないなど、オブジェクト指向とはいきれない部分もあります。

■ビルトイン(組み込み)オブジェクト

オブジェクトを実際に使うときは、最初からJavaScriptに組み込まれている「ビルトイン(組み込み)オブジェクト」を利用することになります。たとえば、先ほどのnavigatorオブジェクトやdocumentオブジェクト以外にも、Formオブジェクト、Dateオブジェクト(日付オブジェクト)、Mathオブジェクト(算術オブジェクト)などのビルトインオブジェクトがあります。このうち、navigatorオブジェクト、documentオブジェクト、Formオブジェクトは目に見えるオブジェクト、Dateオブジェクト、Mathオブジェクトは目に見えないオブジェクトです。

このようなビルトインオブジェクトがあるのは、Webページ制作者がよく利用するであろう基本的なモノを、それがプログラムを書かなくても使えるようにするためにです。もしも、基本的なモノがビルトインオブジェクトとして組み込まれていないと、Webページ上のマウスカーソルのx座標とy座標を調べたいときや、ログの値や平方根の値などを求めたいときに、自分でその機能をプログラミングしなくてはなりません。これでは、Webページの制作はとても苦痛なものとなってしまいます。こうした問題が起こらないようにするために、Webページ制作上でよく使うモノをオブジェクトとして実装し、メソッドを用いて利用できるようにしているのです。

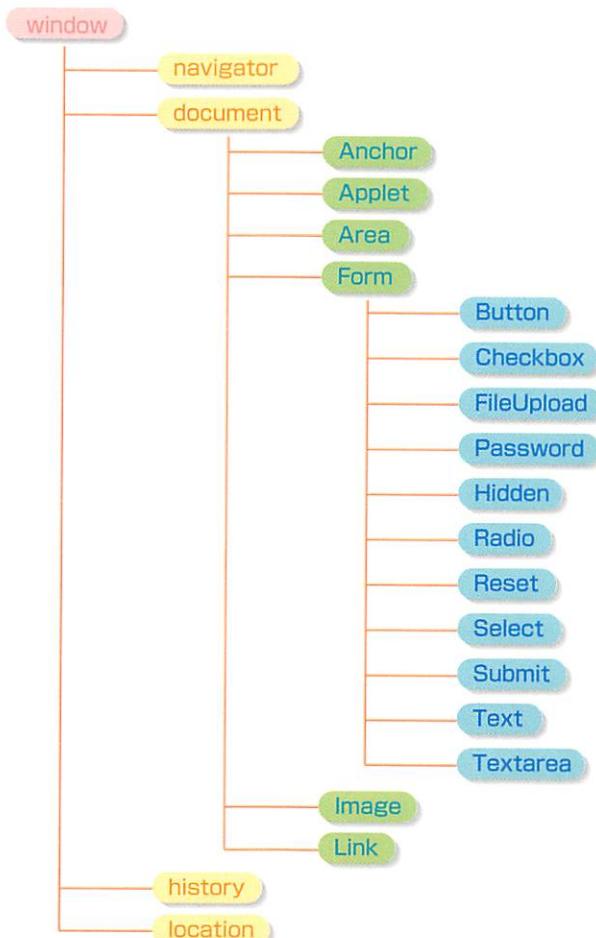
▼ビルトインオブジェクト一覧

| | | | |
|---------------|-----------------|-----------|----------------|
| Anchor | anchors array | Applet | applets array |
| Area | arguments array | Array | Button |
| Checkbox | Date | document | elements array |
| embeds array | FileUpload | Form | forms array |
| Frame | frames array | Function | Hidden |
| History | history array | Image | images array |
| Link | links array | location | Math |
| MimeType | mimeTypes array | navigator | Number |
| Option | options array | Password | Plugin |
| plugins array | Radio | RegExp | Reset |
| Select | String | Submit | Text |
| Textarea | window | | |

●オブジェクトの配置

JavaScriptのビルトインオブジェクトの大半は、windowオブジェクトを頂点にした階層構造に組み込まれています。この階層を「オブジェクト階層」と呼びます。また、この階層構造自体をオブジェクトモデルと呼ぶこともあります。

▼オブジェクトモデル

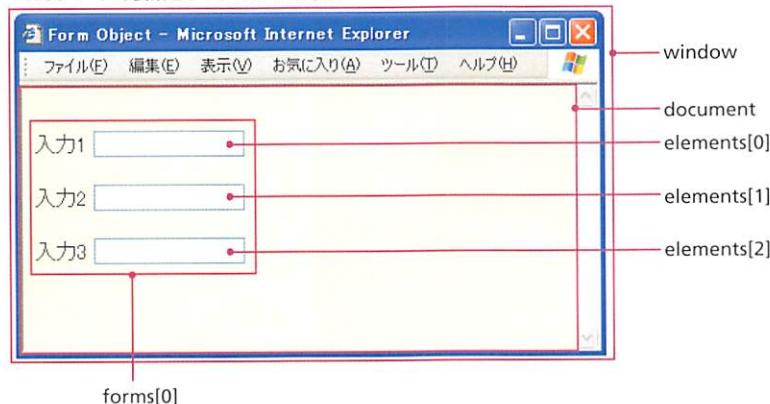


このような階層構造になっているのは、オブジェクトへのアクセスを容易にするためです。たとえば、インプットフォームが配置されているWebページを考えてみましょう。次ページの図は、そのときのオブジェクトの範囲と名前を対応させたものです。

図のように、Webブラウザのそれぞれのオブジェクトには、対応する名前がついてい

ます。なお、同じ要素が並んだときは配列として扱われ、[]内の数字で区別します。[]内の数字は、0から始まります。

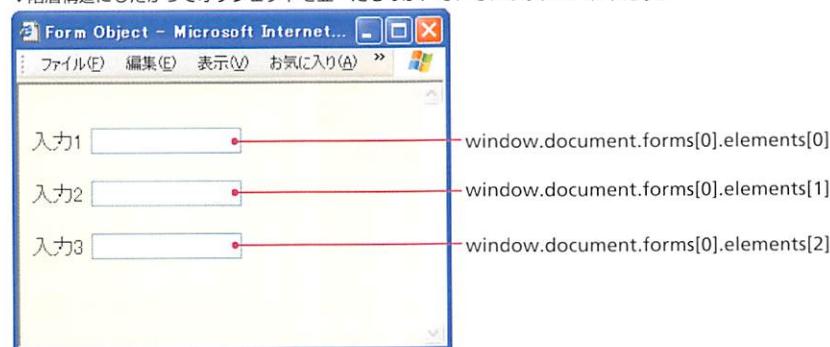
▼Webページの要素とオブジェクトの対応



なぜ、このような構造になっているのでしょうか？それは、Webページ内が構造を持たなかった場合、それぞれのフォームなどのオブジェクトを管理するのに、各オブジェクトを識別するための名前をつけなければならないからです。これは一見簡単なようですが、要素をプログラムでコントロールしたいときなどに面倒なことになります。そのため、Webブラウザは、Webページを階層構造により管理しているのです。とはいえ、それぞれのオブジェクトに自分で名前をつけることもできます。

オブジェクトの階層構造にしたがって、それぞれのオブジェクトへアクセスするための名前を考えてみると、次の図のようになります。

▼階層構造にしたがってオブジェクトを並べたものが、それぞれのオブジェクトを示す



ここからは、主要なオブジェクトを取り上げて、それぞれ解説していきましょう。

●windowオブジェクト

windowオブジェクトは、すべてのオブジェクトの元となるものです。このオブジェクトが扱うものは、ウィンドウの操作や、メニューバー、ツールバー、アラートといった、Webブラウザ全体の動作です。

windowオブジェクトは、すべてのオブジェクトの親であり、必ず存在するものです。そのため、何かオブジェクトを指定する際に、windowオブジェクトを参照する記述を省略してもかまいません。たとえば、アラート(警告のダイアログボックス)を呼び出す場合、本来なら、次のように記述する必要があります。

```
window.alert();
```

これを、windowオブジェクトを省略して、次のように記述することもできます。

```
alert();
```

▼windowオブジェクトの主なプロパティ

| | | | | | |
|--------|---------------|--------|--------|--------|--------|
| closed | defaultStatus | frames | length | name | opener |
| parent | self | status | top | window | |

▼windowオブジェクトの主なメソッド

| | | | | |
|-------|------|--------------|--------|------------|
| alert | blur | clearTimeout | close | confirm |
| focus | open | prompt | scroll | setTimeout |

●navigatorオブジェクト

navigatorオブジェクトは、Webブラウザについての情報を持っているオブジェクトです。このオブジェクトを使うことにより、ユーザーが使っているWebブラウザやOSなどの環境について調べることができます。

▼navigatorオブジェクトの主なプロパティ

| | | |
|-------------|---------|------------|
| appCodeName | appName | appVersion |
| mimeTypes | plugins | userAgent |

▼navigatorオブジェクトの主なメソッド

| |
|-------------|
| javaEnabled |
|-------------|

●documentオブジェクト

documentオブジェクトは、「Webページを表示する部分」を表すオブジェクトです。

documentオブジェクトには、アンカー、フォーム、リンク、タイトル、色、場所など、Webページの要素に関するさまざまな情報が含まれており、多くの要素オブジェクトを操作するメソッドやプロパティを提供しています。documentオブジェクトは、実質的には<body>～</body>で囲まれた部分に対応する、と考えることができます。

この範囲を操作するには、ほとんどの場合、documentオブジェクトの子にあたるオブジェクトやメソッド、プロパティを扱えばよいことになります。

▼documentオブジェクトの主なプロパティ

| | | | | |
|--------------|------------|---------|---------|--------|
| alinkColor | anchors | applets | bgColor | cookie |
| domain | embeds | fgColor | forms | images |
| lastModified | linkColor | links | referer | title |
| URL | vlinkColor | | | |

▼documentオブジェクトの主なメソッド

| | | | |
|-------|------|-------|---------|
| close | open | write | writeln |
|-------|------|-------|---------|

documentオブジェクトは、formオブジェクトやlinkオブジェクトなどを子として持っています。

●locationオブジェクト

locationオブジェクトは、windowオブジェクトの子にあたるオブジェクトです。このオブジェクトは、場所(URL)に関しての処理を行うときに使います。URLを変更したり、現在表示されているページのアドレスについての情報を取り出したりすることができます。

▼locationオブジェクトの主なプロパティ

| | | | |
|----------|------|----------|--------|
| hash | host | hostname | href |
| pathname | port | protocol | search |

▼locationオブジェクトの主なメソッド

| | |
|--------|---------|
| reload | replace |
|--------|---------|

●historyオブジェクト

historyオブジェクトは、Webブラウザの持っている履歴に関する処理を行うときに使います。このオブジェクトを用いることにより、1つ前のページへ戻るといった動作をさせることができます。

▼historyオブジェクトの主なプロパティ

| current | length | next | previous |
|---------|--------|------|----------|
|---------|--------|------|----------|

▼historyオブジェクトの主なメソッド

| back | forward | go |
|------|---------|----|
|------|---------|----|

■オブジェクトの生成

オブジェクトを使うには、オブジェクトを明示的に「生成」する必要があります。これは、オブジェクトによっては、1つのWebページ内で複数使われる可能性があるからです。Webブラウザは、いくつのオブジェクトが必要になるかわかりませんので、事前にオブジェクトを生成しておくわけにはいきません。そのため、必要な数だけ、自分でオブジェクトを生成することになります。このとき生成されたオブジェクトを、「オブジェクトの実体」と呼びこともあります。

オブジェクトを生成するときには、名前が必要です。この名前は、Webページ内でオブジェクトを一意に区別するために用いられるので、1つのWebページ内で同じ名前をつけることはできません。

オブジェクトの生成には、「new」キーワードを使います。書式は、次のようになります。

ページ内で認識するオブジェクト名=new オブジェクト名();

これで、「オブジェクト名」と同じ機能を持ったオブジェクトが生成されます。

これを具体的な例で見てみましょう。

```
myDate=new Date();
```

この記述で、Dateオブジェクトを基にした「myDate」オブジェクトが生成されます。この生成は、Webページ内で複数使用される可能性のあるオブジェクトにだけ行います。逆にいえば、Webページ内に必ず1つしかないnavigator、historyなどは、オブジェクトの生成を行わなくとも使用することができます。

●with文

JavaScriptでは、スクリプトの中でオブジェクトの階層を記述します。これは、ページ内で論理的にオブジェクトを扱うのに便利なのですが、同じような階層構造を記述するときには、そのつど記述しなくてはならず、入力が面倒です。そんなとき、with文を使

うことで、共通するオブジェクトの記述を省略できます。書式は、次のようにになります。

```
 with(オブジェクト名) {  
    処理;  
}
```

たとえば、文字列を画面に表示するための、次のような記述があります。

```
document.write("第1行目");  
document.write("第2行目");  
document.write("第3行目");
```

with文を用いることで、繰り返し出てくるdocumentオブジェクトをひとまとめにして記述することができます。

```
with(document) {  
    write("第1行目");  
    write("第2行目");  
    write("第3行目");  
}
```

with文は、入れ子に指定(with文の中でwith文を指定)することも可能です。

■メソッド

メソッドは、オブジェクトを操作するためのスイッチです。オブジェクトは、基本的にメソッドにより操作されます。オブジェクトには、それぞれを操作するためのメソッドが用意されています。JavaScriptでは、メソッドを使い、オブジェクトを操作することでプログラミングを行います。

JavaScriptでのメソッドの記述は、ちょうど関数の呼び出しと同じような形をしています。あるオブジェクトのメソッドを操作するには、「どのオブジェクト」に対して「どのような操作」をするかを指定します。この関係を記述するのに、JavaScriptでは「.」(ピリオド)を用います。

メソッドを記述するための書式は、次のようになります。

オブジェクト名.メソッド名();

具体的には次のように利用します。

```
document.write("Hello! JavaScript");
```

このように、Webページが表示される部分に該当するdocumentオブジェクトに対して、writeメソッドを使うことで、Webページに文字列を表示することができます。

■プロパティ

オブジェクトは、その内部にあらかじめさまざま値を持っています。それを保持しているのがプロパティです。プロパティは、オブジェクト内の固有の変数ということもできます。

JavaScriptでは、プロパティはビルトインオブジェクトがもともと持つものとして実装されています。プロパティに値を設定することにより、オブジェクトの設定値を変更することができます。プロパティの値として設定されているのは、背景色、テキストの色、Webブラウザの各種情報(ブラウザ名、バージョン情報など)などです。

プロパティの使い方は、メソッドと同様に、オブジェクトを指定してプロパティを指定します。つまり、「どのオブジェクト」の「どの値」といった指定になります。メソッドと異なるのは、単なる値なので、プロパティ名の後ろに「()」をつけないことです。

プロパティの書式は、次のようにになります。

オブジェクト名.プロパティ名

プロパティへアクセスするための記述例をいくつか挙げておきます。

```
document.bgColor;  
navigator.appName;  
this.value;
```

プロパティには、値を設定したり参照したりすることができます(ただし、プロパティの中には値が設定できないものもあります)。プロパティに値を設定するには、プロパティに値を代入します。書式は、次のようにになります。

オブジェクト名.プロパティ名=値;

この記述からわかるように、代入演算子を用いて、プロパティに値をセットします。記述例としては、次のようにになります。

```
document.bgColor="#ff0000";
document.forms[0].elements[0].value=123;
```

また、プロパティの参照(読み出し)は、プロパティを指定することで行います。代入演算子で行うこともできますし、メソッドの引数の部分でも行うことができます。

変数=オブジェクト名.プロパティ名;
 メソッド(オブジェクト名.プロパティ名)

記述例としては、次のようなものになります。

```
a=document.forms[0].elements[0].value;
document.write(navigator.appName);
```

COLUMN

JavaScriptの応用範囲

JavaScriptを使えば、Webページ上のさまざまな要素を操作することができます。しかし、それだけではなく、プラグインやJavaなども操作することができます。

現在、広く使われているMacromedia社のFlashも、ムービーの再生、停止、フレームの指定などをJavaScriptから行うことができます。これにより、Flashのムービーと連携できるとともに、Flashだけでは難しかった動作も実現できることになります。これは、同社のShockwaveについても同様です。これらの機能は、「Live Connect」と呼ばれています。

また、Javaアプレットで公開されているメソッドを、JavaScriptから操作することもできます。Javaアプレットの作り方次第ですが、HTMLで書かれたフォームから、Javaアプレットをコントロールすることが可能となります。

以上のように、さまざまなプラグインをJavaScriptによりコントロールできます。

DOM

JavaScriptにはオブジェクト階層があり、それをスクリプトで操作できることは、すでに解説したとおりです。しかし、最近のWebブラウザは、p.57で紹介したものとは別に、もう1つのオブジェクト階層を実装しています。それが、DOM (Document Object Model)と呼ばれるものです。

DOMは、Webページにかぎらず、さまざまなXML (HTMLも含む)を階層構造により表現し、管理しようと考案されたものです。DOMを実装しているWebブラウザでは、Webページ内のすべての要素に対してDOMを使ってアクセスし、操作することができます。

DOMは、JavaScriptのオブジェクト階層とは別に存在しています。しかし、その構造はよく似ており、一部には同様の構造をしている部分もあります。この同じ部分については、JavaScriptのオブジェクト階層のアクセス方法と、DOMのアクセス方法が同じになることがあります。

DOMの規格は、W3Cで検討され、制定されています。現在、DOM Level1とDOM Level2が勧告案になっています。なお、DOM Level2は、5つの仕様から構成されます。

▼Document Object Model (DOM) Level 1 Specification

URL

<http://www.w3.org/TR/REC-DOM-Level-1/>

▼Document Object Model (DOM) Level 2 Core Specification

URL

<http://www.w3.org/TR/DOM-Level-2-Core/>

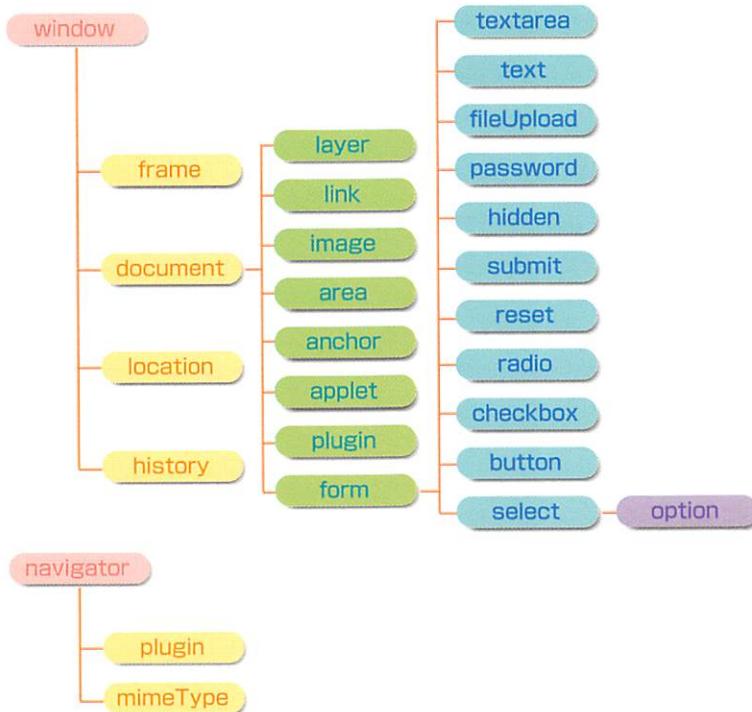
(DOM Level2に関してはコアのみを紹介)

上記のURLにある仕様は、かなりのボリュームで、理解するのは非常に面倒と思われるかもしれません。しかし、JavaScriptとHTMLを使ってWebページを作るという観点からすると、使う部分は少なく、実はさほど難しいものではありません。

■DOMの構造

DOMの構造はシンプルで、windowオブジェクトをトップにした階層構造の下に、Webページで使うオブジェクトが配置されているというものです。

▼DOMの構造



このような階層構造になっているのは、Webページ内に散在している各種オブジェクトに容易にアクセスできるようにするためにです。この構造がわからないと、JavaScriptからオブジェクトを指定することができず、操作できることになります。

たとえば、次のようなHTMLがあったとします。

```
<html>
<title>DOM sample</title>
<body>
<h3>Sample</h3>
<hr />

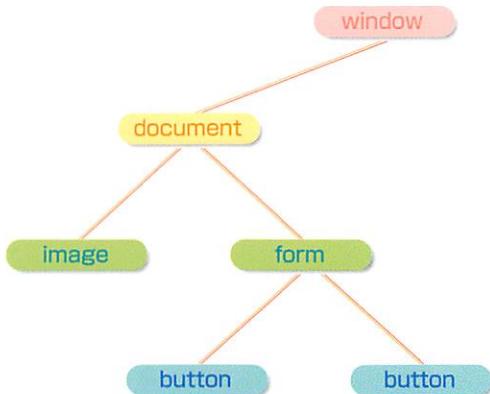
<form id="myForm">
  <input type="button" value="click1" />
  <input type="button" value="click2" />
</form>
```

```
</body>
</html>
```



このときのDOM構造は、次のようにになっています。

▼上記HTML文のDOM構造



このように、HTMLは必ず、DOMで表現できます。JavaScriptなどのスクリプト言語でHTML内のオブジェクトを操作するときには、この構造を必ず意識しなくてはなりません。

逆にいって、DOMを利用するには、HTMLを構造的に記述する必要があります。ここでの「構造的である」とは、HTMLの要素を入れ子にしないということを意味しています。

■オブジェクトの特定

DOMを使って各種オブジェクトを特定するには、どうすればよいのでしょうか？ この特定には、大きく分けて2つの方法があります。

1つは、各要素につけられた名前を基に、階層構造を指定する方法、もう1つは、HTML中の要素につけられたid属性を基に特定する方法です。

●名前を基に階層構造を指定する

HTMLの各要素につけられた名前を基に、階層構造を記述して、要素を指定します。これは、JavaScriptのオブジェクトモデルでの、name属性を使った指定と同じです。

たとえば、次のようなHTMLが存在したとします。

```
<html>
<title>getElementsByName 1</title>
<body>
<h3>getElementsByName01</h3>
<hr />
<br />
</body>
</html>
```

img要素には、id属性を用いてimg1という名前がつけられています。このとき、img要素を特定するには、次のように記述します。

```
document.img1
```

ただし、上記は要素を特定しただけです。実際には、その要素のプロパティを参照・設定する必要があります。たとえば、img要素の幅を設定するときは、次のような記述となります。

```
document.img1.width
```

この記述は、p.69で解説したname属性を使った要素の指定と同じになります。DOMは、IE5以上、NN6以上で実装され始め、Opera Software ASA社のOpera、Apple Computer社のSafariなどでもサポートされています。さらに、W3Cで標準仕様が決められ、それにしたがって実装されているために、互換性の問題もありません。そのため、複数のブラウザに対応したスクリプトを記述するには重要な技術であり、今後はよりいっそう使われるようになるはずです。

●getElementByIdメソッドで特定する

DOMでは、HTML内の要素を特定するためのメソッドとして、getElementByIdメソッドが用意されています。HTML4.01では、id属性により、HTMLの各要素に識別のための名前を定義することができます。このメソッドは、id属性によってHTMLの各要素につけられた名前を基に、要素を特定します。

getElementByIdメソッドは、次のような書式で使います。

document.getElementById("id名")

上記の記述により、id名がつけられたHTMLの要素が特定できます。もしも、複数の要素のid属性に同じ名前がつけられていたときは、1つ目を特定した要素として処理をし、2つ目以降は無視します。

また、実際には要素を特定するだけではなく、要素の値を変更したり、値を取り出したりして使うため、上の記述の後ろにプロパティを指定する必要があります。たとえば、幅を指定するときには、widthプロパティを指定します。コードは、次のようにになります。

```
var imagewidth=document.getElementById("img1").width;
document.getElementById("img1").width=50;
```

getElementByIdメソッドは、IE5以上、NN6以上、Safari、Operaなどで実装されています。実際にgetElementByIdメソッドを利用したスクリプトは、次のようになります。

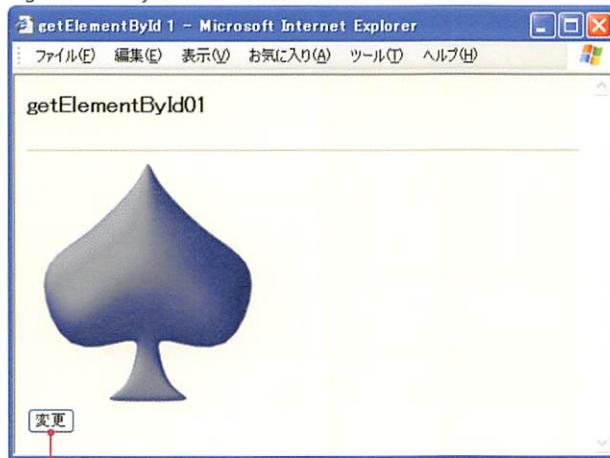
```
<html>
<title>getElementById 1</title>
<body bgcolor="#fff8dc">
<script type="text/javascript">
function changeImage() {
    document.getElementById("img1").width=50;
}
</script>

<h3>getElementById01<h3>
<hr />

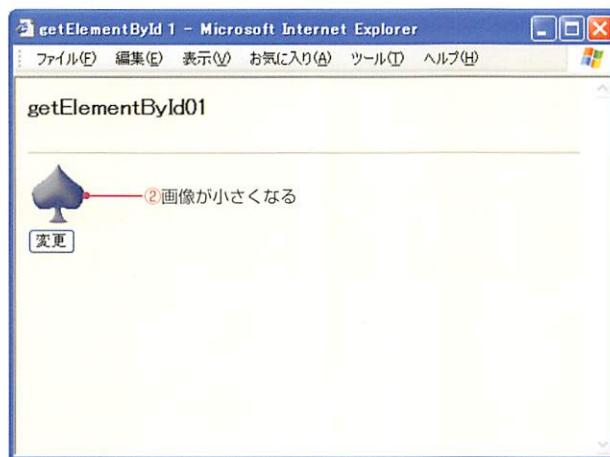
<br />
<input type="button" id="button1" value="変更" onclick=
 "changeImage()" />
</body>
</html>
```

上記のスクリプトの実行結果は、次のようになります。

▼getElementByIdメソッドを利用したサンプル



①ここをクリックすると…



なお、IE4には、当時DOMの仕様がまだドラフト段階にあったため、Microsoft社が独自に定義したallメソッドが実装されています。使い方は、getElementByIdメソッドとほとんど同じです。

document.all("id名").プロパティ

このメソッドは、DOMを正式にサポートするようになったIE5以降も、互換性のために実装されています。したがって、IEだけのWebページを作るのであれば、allメソッド

を使えばIE4までサポートできることになります。

しかし、実際にインターネット上に公開するWebページでは、サポート対象の広いgetElementByIdメソッドを用いたほうがよいでしょう。

●getElementsByNameメソッドで特定する

HTML要素の中から、タグ名を基に特定の要素を指定するには、getElementsByNameメソッドを用います。このメソッドは、引数に与えられたタグ名を取り出すことができます。HTMLタグは通常、HTML文内に複数存在します。そのため、getElementsByNameメソッドでは、指定したタグ名にマッチしたすべての要素を配列にして返します。その要素の中からある特定のものを指定するには、次のようにインデックスを使います。

document.getElementsByName("タグ名")[インデックス]

配列のインデックスは、JavaScriptの配列と同じように、0から始まります。

実際には、要素の値を変更したり、値を取り出したりして使うため、上の記述の後ろにプロパティを指定することになります。コードは、次のようにになります。

```
document.getElementsByName("img")[0].width=50;
```

getElementsByNameメソッドは、IE5以上、NN6以上、Safari、Operaなどで実装されています。実際にgetElementsByNameメソッドを利用したスクリプトは、次のようにになります。

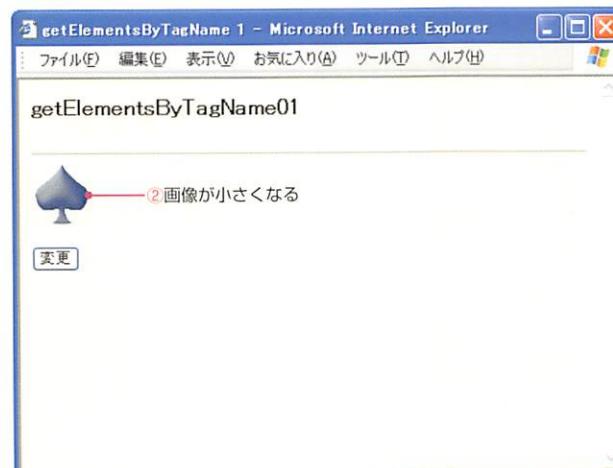
```
<html>
<title>getElementsByName 1</title>
<body bgcolor="#ffff8dc">
<script type="text/javascript">
function changeImage() {
    document.getElementsByName("img")[0].width=50;
}
</script>

<h3>getElementsByName01</h3>
<hr />
<br />
```

```
<br />
<input type="button" id="button1" value="変更" onclick=
  "changeImage()" />
</body>
</html>
```

上記スクリプトの実行結果は、次のようにになります。

▼getElementsByTagNameメソッドを利用したサンプル



● 値の設定

DOMを用いれば、すべてのHTML要素にアクセスできることがわかりました。また、各HTML要素のプロパティの値を参照・設定できることを説明してきました。ここで、もう1つ別の、値の設定方法を紹介します。それは、`setAttribute`メソッドを使う方法です。`setAttribute`メソッドは、特定した要素に対して値を設定するとき用います。書式は、次のようにになります。

`要素.setAttribute("プロパティ", "値")`

要素の特定は、これまでに紹介した方法で行います。実際の記述としては、次のようになります。

`document.getElementById("img").setAttribute("width", "50");`

COLUMN

制御コード

JavaScriptで文字列を出力するときに、改行やタブはどうやって表現すればよいのでしょうか？ そのときには、以下のコードを用います。本書のサンプルでも、アラート内に表示する文字列などで、「¥n」を用いています。Unicodeによる指定は、IE4以降とNN4.06以降、Safariで用いることができます。

| 制御コード | Unicode | 名前 | シンボル |
|-------|---------|--------------|------|
| ¥b | ¥u0008 | バックスペース | <BS> |
| ¥t | ¥u0009 | タブ | <HT> |
| ¥n | ¥u000A | ラインフィード | <LF> |
| ¥f | ¥u000C | フォームフィード | <FF> |
| ¥r | ¥u000D | 復帰 | <CR> |
| ¥" | ¥u0022 | ダブルクオーテーション | " |
| ¥' | ¥u0027 | シングルクオーテーション | ' |
| ¥¥ | ¥u005C | 円(バックスラッシュ) | ¥ |

イベント

「イベント」とは、Webページを見ているユーザーが行った操作のことです。あるイベントが起こった際に、特定のものに対してアクションを起こすことにより、さまざまな機能を実現することができます。

イベントとは、具体的には次のようなものです。

- ・フォームのボタンをクリックした
- ・ハイパーリンクになっている文字列をクリックした
- ・Webページがロードされた

また、JavaScriptで利用可能なイベントを、表にまとめておきます。

▼JavaScriptで利用可能なイベント

| イベント | いつ発生するか | イベントハンドラ |
|-----------|--|-------------|
| abort | ユーザーがイメージの読み込みを中止した場合に生じる。たとえば、イメージのダウンロード中に、ほかのリンクやWebブラウザの中止ボタンをクリックしたときなど | onabort |
| blur | Webブラウザ本体、Webページ内のフレーム、フォームのボタンなどから、ユーザーのフォーカスが離れた場合に生じる | onblur |
| click | ユーザーがリンク上やフォーム要素上でクリックした場合に生じる | onclick |
| change | ユーザーによって要素(フォームフィールド)の値が変更されたときに生じる | onchange |
| error | ドキュメント、イメージの読み込みエラーが起きた場合に生じる | onerror |
| focus | Webブラウザ本体、Webページ内のフレーム、フォームのボタンなどにユーザーのフォーカスが与えられた場合に生じる | onfocus |
| keydown | ユーザーがキーを押したときに生じる | onkeydown |
| load | WebページがWebブラウザにロードされたときに生じる | onload |
| mouseout | ユーザーが領域の外にカーソルを移動させたときに生じる | onmouseout |
| mouseover | ユーザーが領域にカーソルを移動させたときに生じる | onmouseover |
| reset | ユーザーがresetボタンをクリックしたときに生じる | onreset |
| resize | ユーザーがウィンドウのサイズを変更したときに生じる | onresize |
| submit | ユーザーがsubmitボタンをクリックしたときに生じる | onsubmit |
| unload | ユーザーがWebページから出るときに生じる | onunload |

■イベントハンドラ

Webページに対して、ユーザーが何らかの操作を行うことで、さまざまなイベントが発生します。ユーザーがWebページ上の何もないところでクリックしたとしても、イベントは発生しています。つまり、ユーザーのほとんどの操作に対して、イベントが発生しているのです。

しかし、通常は何も起りません。なぜなら、それぞれのイベントに対応した処理が存在しないからです。数多くのイベントの中から、目的の場所で、目的のイベントを取り出すための仕組みを、JavaScriptでは「イベントハンドラ」と呼んでいます。主なイベントハンドラは、前ページの表に示したとおりです。

イベントハンドラを利用する方法には、HTMLタグを用いるものと、オブジェクトを使うものの2つがあります。

●HTMLタグを用いたイベントハンドラの利用

HTMLタグを用いたイベントハンドラの書式は、次のようにになります。

<HTMLタグ イベントハンドラ="JavaScriptプログラム">

この記述により、ユーザーがHTMLタグの領域上でイベントハンドラに対応したイベントを起こしたときに、指定したJavaScriptのプログラムが実行されます。

イベントハンドラに与える「JavaScriptプログラム」の部分は、さまざまな書き方をすることができます。関数を指定することもできますし、直接長いプログラムを記述することもできます。

関数を指定するときは、先に関数を定義しておく必要があります。たとえば、事前にprintstatus関数が定義してある場合、それを呼び出すには次のように記述します。

```
<input type="button" onclick="printstatus()" />
```

この記述では、フォームのボタンをクリックしたとき(onclick)にprintstatus関数が呼び出され、関数内に記述されている処理が行われます。

JavaScriptのプログラムを直接記述する場合は、次のようにになります。

```
<input type="button" onclick="
  if(value>=4) {alert('You are Right!');}
  else {alert("Oh no!");}
  " />
```

また、複数のコマンドを使用したいときは、「;」(セミコロン)で分離することで実装できます。

```
<input type="button" onclick="  
    appName=navigator.appName;  
    alert('あなたのWebブラウザは'+appName+'ですね。');  
" />
```

このように、イベントハンドラに与えるJavaScriptプログラムの定義には、2つの方法があることがわかりました。では、どちらを使うのがよいのでしょうか？ これは、イベントハンドラに対応した処理がどのようなものかで決まります。

もしも、HTML文内で1回しか行われない処理で、処理の記述が短いものならば、HTMLタグ内のイベントハンドラに直接プログラムを記述してしまうのがよいと思います。しかし、1回しか行われない処理であっても、あまりに記述が長いときは、視認性やメンテナンス性のことを考えて、関数を定義してイベントハンドラで呼び出すのがよいでしょう。

たとえば、次のようなコードは視認性がよくありません。

```
<input type="button" onclick="  
    if(value>=4) {  
        document.write(navigator.appName);  
        document.write(navigator.appversion);  
    }  
    else if(value==3) {  
        alert('このページがちゃんと動く？');  
        document.write('<A href="*****">');  
    }  
    else {  
        .....  
    }  
" />
```

この視認性の悪さを改善するには、スクリプトの部分を関数にして、呼び出す形式にします。

```

<script type="text/javascript">
function check() {
    if(value>=4)
        document.write(navigator.appName);
        document.write(navigator.appversion);
    }
    else if(value==3) {
        alert('このページがちゃんと動く?');
        document.write('<A href="****">');
    }
    else {
        .....
    }
</script>
<form>
    <input type="button" onclick="check()" />
</form>

```

● スクリプトの部分を
関数にする

└ イベントハンドラで関数を呼び出す

また、呼び出す処理を同じWebページ内で何度も行うならば、関数として定義して呼び出すようにすると、記述量が少なくなり、視認性やメンテナンス性が向上します。

たとえば、Webページ内に複数のフォームを用意し、ユーザーに数値だけを入力させたいとします。そのとき、入力された内容が本当に数値かどうかをチェックするプログラムをフォームごとに書いていたら、非常に面倒なことになります。そこで、数値かどうかをチェックする関数を1つ記述しておき、フォームからそれを呼び出すことで記述量を減らせます。

なお、イベントハンドラは、どのタグにも属性としてつけられるわけではありません。

●オブジェクトを使ったイベントハンドラの利用

イベントハンドラを利用する方法として、オブジェクトを使うものもあります。HTML要素(タグ)にイベントハンドラを設定する方法は、通常、イベントハンドラの対象を特定できる場合に用いられます。その代表的なものとして、ボタンやチェックボックスなどの各種フォームがあります。

これに対して、Webブラウザ全体のどこで起きてもよいようなイベント(たとえばonkeydownやonloadなど)は、特定のHTML要素に対してイベントハンドラを設定するのが難しいことがあります。そんなときに使うのが、オブジェクトに対してイベントハ

ンドラを設定する方法です。

書式は、次のようにになります。

オブジェクト.イベントハンドラ=命令；

実際の記述例としては、次のようなものになります。

▼ウィンドウがリサイズされたとき、`resize_win`関数を呼び出す

```
window.onresize=resize_win;
```

▼ドキュメント上でキーが押されたら`key_down`関数を呼び出す

```
window.document.onkeydown=key_down;
```

このようにオブジェクトを指定し、それにイベントハンドラを設定することができます。なお、通常の関数の呼び出しは引数を取らない場合でも「()」がつくのですが、この書式を用いたときには「()」をつけませんので、注意してください。

COLUMN

CSS

JavaScriptと並び、WWW関連の技術の中で大きな要素となっているのが、「CSS (Cascading Style Sheets)」です。この技術は、登場してからずいぶんと時間が経っており、Webページのスタイルシートとしてのスタンダードの地位を確保しています。

スタイルシートは、Webページのデータとスタイルの分離のためには非常に重要な技術です。2003年9月現在、CSS Level1に始まり、CSS Level2までがW3Cにより勧告されています。CSS2では、Webページの要素だけではなく、音声などを操作するためのプロパティも定義され、大きな仕様となっています。さらに現在、CSS Level2.1、CSS Level3が検討されています。

URL

<http://www.w3.org/Style/CSS/>

Cookie(クッキー)

■Cookieとは

Webブラウザは、通常、ユーザーが使っているマシンのローカルファイル(クライアント側)に情報を保存することができません。これは、Webページからダウンロードしてきたプログラムなどを勝手に保存できるようになっていると、セキュリティ上問題があるからです。しかし、JavaScriptを用いると「Cookieファイル」と呼ばれるものをクライアント側に保存することができます。

Cookieは、非常に単純なもので、ある決められたフォーマットで、ある決められた場所にだけ、テキストで情報を保存できるというものです。そのため、Webサーバに接続してきたユーザーに発行したIDや、Webページを使ううえでの設定情報を保存するときなどに使います。

▼Cookieの動作



Cookieが使われる以前のWebサーバは、ユーザーが情報を入力してこないかぎり、接続してきたクライアントが誰なのかを特定することができず、特定のユーザーに対して特定の動作を行うといったことは実現困難でした。しかし、それではインターネット上で業務用Webアプリケーションなどを作成することができません。Cookieは、このような問題とセキュリティの両方をクリアしたものです。

Cookieの動作は、上の図のようになります。JavaScriptのプログラムは、あるユー

ユーザーがWebサーバに接続したとき、そのWebサーバがユーザーを特定するために発行したIDが保存されているかどうかを確認します。もしも、IDがなかったときは、Webサーバは次回以降そのユーザーを特定できるように、固有のIDを発行します。それをCookieファイルとして保存しておきます。これで、2回目以降の接続では、固有のIDの確認時に、IDをWebサーバに送ることができます。この固有IDを使って、ユーザーそれぞれにカスタマイズしたWebページを送ったり、以前登録してもらった値を使って、ユーザー固有的の処理を行ったりすることができます。

なお、Cookieは、実際にはWebブラウザごとに保存されるので、ユーザーではなく、あるPCのWebブラウザを特定していることになります。

■Cookieの制限

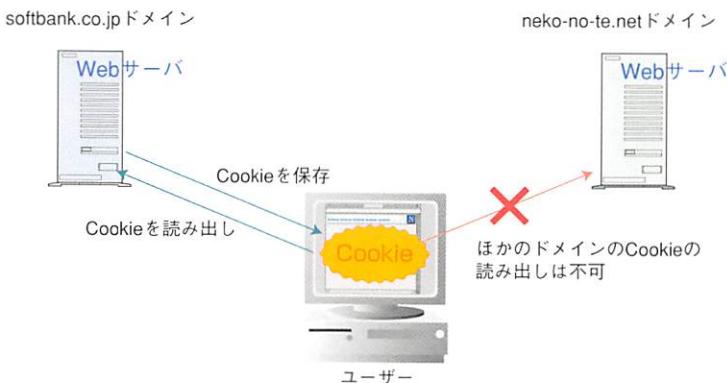
Cookieは、非常に制限された形式でしか、クライアント側に情報を保存することができません。その制限とは、次のようなものです。

- ・保存先が限定されている
- ・一定の書式でしか書き込みができない
- ・保存形式はテキストのみである
- ・ユーザーに関しての情報は送信されない
- ・ほかのドメインのCookieを読み出せない

まず保存先は、ある特定のフォルダに固定されています。そこ以外には、保存することができません。保存できる形式はテキストのみであり、その記述方法も決まっています。また、重要なことなのですが、アクセスしてきた人の電子メールアドレスやハードディスクの内容といった具体的な情報は、いっさい集めることができません。これは、個人の情報を勝手に読み取られないようにするとともに、ウイルスなどのローカルマシンに対して攻撃を加えるものを保存できないようにするためにです。こういった点で、セキュリティやプライバシーの問題も解決されているといえます。

また、ほかのドメインが保存したCookieを読み出すことはできません。ここでいうドメインとは、URLのドメイン部分に相当します。たとえば、あるWebページのドメイン名で登録されたCookieの内容は、そのドメインからしか読み出すことができないのです。この制限によっても、プライバシーが保護されることになります。

▼Cookieは保存したドメインしか読み出せない



■Cookieファイルの保存場所

Cookieファイルは、どこに保存されているのでしょうか？ それは、使っているOSやWebブラウザによって異なります。たとえば、IE、NN、Safariでは、それぞれ次のフォルダに保存されています。

▼IE、NN、SafariのOS別Cookieファイルの保存場所

| ブラウザ | OS | 保存場所 |
|--------|-----------------|---|
| IE6 | Windows 2000/XP | C:\¥Documents and Settings¥<username>\¥Cookies¥ |
| IE5 | Mac OS X | /applications/Internet Explorer.app/Contents/MacOS/Preference Panels/Cookies.cfm/Contents/MacOS/Cookies |
| NN7 | Windows 2000/XP | C:\¥Documents and Settings¥<username>\¥Application data¥Mozilla¥Profiles¥<username>\¥<セットアップ日時.sit>\cookies.txt |
| | Mac OS X | [ユーザーフォルダ]/Library/Mozilla/Profiles/default/[ランダム文字列.sit]/cookies.txt |
| Safari | Mac OS X | [ユーザーフォルダ]/Library/Cookies/Cookies.plist |

この表にあるように、それぞれのOSやWebブラウザごとに、保存される場所やファイル名は異なりますが、そのことをJavaScriptのプログラムで意識する必要はありません。Cookieは、アクセスしたサーバの名前からファイルまたはキーを作成して、情報を保存します。そのため、どこに、どのような名前でファイルを保存するかなどは、いちいち指定しなくてよいのです。

■Cookieへの書き込み

Cookieへの書き込みは、documentオブジェクトのcookieプロパティに値を代入することで行います。

Cookieへの書き込みは、次のような書式になります。

```
 document.cookie="キー名=値; expires=期限(GMT); path=URLパス;  
 domain=サイトドメイン; secure";
```

この中で、"キー名=値"だけは、必ず指定しなくてはなりません。ほかのexpires、path、domain、secureは、指定しなくとも、Webブラウザが適切に設定してくれます。ただし、expiresを設定しておかないと、Webブラウザの終了とともにクッキーの有効期限が切れます。これは、セッションの終了までが有効期限ということです。

それぞれの属性の意味は、次の表のようになります。

▼Cookieに書き込まれる属性とその内容

| 属性 | 説明 |
|----------------|---|
| name=値 | Cookieファイルに保存されるデータの変数名と、その中に保持される値。これは省略することができない |
| expires=期限 | Cookieの有効期限の日時を指定する。この日時以降は、Cookieは保存されず、サーバにも送信されない。この日時の指定は、グリニッジ標準時で行う。書式は原則として、「Wdy, DD-MM-YYYY, HH:MM:SS GMT」となる |
| path=バス | 有効なCookieのURLのうち、バスの部分を指定する。指定がなければ、バスはCookieを設定したドキュメントと同じになる |
| domain=サイトドメイン | 有効なCookieのURLのうち、ドメインの部分を指定する。指定しなければ、ドメインはCookieを設定したドメインと同じになる |
| secure | この属性を記述すると、Cookieを安全な接続だけを使って送信するようになる |

記述の例は、次のようなものです。

```
document.cookie="name="+myName;
```

```
document.cookie="name="+myName+"; expires=Fri, 01-Jan-  
④ 2001 12:00:00 GMT; path='/' ; domain='www.softbank.co  
④ .jp' ; secure";
```

1つ目の記述は、"キー名=値"以外を省略した最も短いもので、2つ目はすべての属性を指定した記述です。

■Cookieの読み出し

Cookieファイルへの保存・参照は、documentオブジェクトの下のcookieプロパティを用います。Cookieは、ドメインごとに保存されます。

document.cookie

Cookieに格納されている値の参照は、次のような記述で行うことができます。
myDataは、任意の変数です。

myData=document.cookie;

このとき読み出されるのは、Webサーバと同じ名前で保存されているデータの内容です。読み出されるデータは、次のような、キーと値が「=」により連結された文字列です。

キー=値；

もしも、1つのドメインで複数のCookieを保存している場合には、キーと値が連結された複数の文字列が、「;」(セミコロン)によって区切られて返されます。

キー1=値1;キー2=値2; ……; キーn=値n;

Cookieの値を実際に使うには、このような形で返されるキーと値の組み合わせを、何らかの方法で整形しなくてはなりません。

通常は、Cookieの値が空でないのを確認してからその内容を読み出し、さらにキーと値を分離する作業が必要になります。

実際にCookieから値を読み出す方法については、SECTION12のサンプルを参照してください。

■Cookieを削除する

Cookieは、有効期限がくると自動的に削除されます。しかし、有効期限より前に、Cookieを削除したくなることもあります。このようなときには、明示的にCookieを削

除しなくてはなりません。

Cookieの削除には、特別なメソッドを使うわけではありません。どのように削除するかというと、有効期限を現在の時刻よりも過去にするという作業を行います。まず、Cookieにセットするための、現在よりも過去の時間を指定した有効期限を生成します。有効期限の作成ができたら、それを削除したいCookieに書き込みます。

`document.cookie="name=<削除したいcookie名>"; expires=現在より
過去の時間;`

これで、Cookieは有効期限が切れたことになり、削除されることになります。

具体的なサンプルは、p.376を参照してください。

COLUMN

文字化けへの対処

JavaScriptで、ダイアログボックスやステータスバーに日本語を表示すると、文字化けしてしまうことがあります。これは、日本の漢字コードが2バイトであることに起因します。Webブラウザは、漢字をうまく認識することができないので。さらに、日本語の文字コードには、「JIS」「Shift-JIS (SJIS)」「EUC (Extended UNIX Code)」の3種類があるため、いっそう問題が複雑化しています。

もし、ダイアログボックス内などに表示した文字が化けてしまったときは、化けてしまう文字の前に「¥」を入れてみてください。たとえば、「表示」という文字を表示させたいときに、「示」の字が化けてしまうとします。そのときは、「表¥示」としてみるのです。環境によっては、「¥」(円記号)は「\」(バックスラッシュ)と表示されることもあります。

なお、<meta>タグを用いると、自動的に日本語コードを切り替えてくれます。たとえば、次のように記述すると、JISで日本語エンコードします。

`<meta http-equiv="Content-Type" content="text/html;
charset=iso-2022-jp" />`

これだけPCが発達した現在でも、依然として日本語処理の問題は解決されていません。特に、JavaScriptやCSSなどでは、いくつかの問題が報告されています。しかし、IE4以降では、それらの問題はかなり解決されてきていましたし、NN4.06以降もUnicodeに対応したことでの、徐々に環境は改善に向かっています。

正規表現

■正規表現とは

JavaScriptで文字列を扱うことは、以前はかなり面倒でした。というのは、Perl、AwkなどといったUNIXの世界における標準的なスクリプト言語に実装されている「正規表現(Regular Expression)」が、JavaScriptには実装されていなかったからです。そのため、文字列処理を行うときには、1文字ずつ識別したり、Stringオブジェクトのsubstrメソッドで文字を切り出してから処理を行ったりと、かなり面倒な処理を記述しなくてはなりませんでした。

しかし、JavaScript1.2 (JScript3.0) から正規表現に対応したことで、文字列の検索などが容易に実現できるようになりました。

正規表現とは、文字列がどのようなパターンになっているかを形式的に表すものです。たとえば、「Th」から始まるもの(This, Theなど)は、「Th*」と表現します。このように、正規表現は通常の文字(a~zなど)と、「メタキャラクタ」と呼ばれる特殊文字から構成される文字列のパターンです。

正規表現は、文字列のパターンを表すものなのですが、これだけでは正規表現を生かすことはできません。そこで、JavaScriptには、正規表現を活用するためのオブジェクトと、いくつかのメソッド、プロパティが用意されています。

JavaScriptで正規表現を使うには、最初にどのような検索を行うかを指定するための、正規表現による検索パターンを用意します。その後、matchメソッド、replaceメソッドなどを用いて、実際の処理を行います。

ここでは正規表現と、それらを活用するオブジェクト、メソッド、プロパティについて解説します。なお、正規表現についての知識は、p.274からの文字列をチェックするサンプルを理解するのに必要ですが、それ以外では不要です。必要になってから、以降を読んでいただいても差し支えないでしょう。

■正規表現パターンの作成

正規表現では通常、区切り文字でパターンを囲みます。JavaScriptでは、「/」(スラッシュ)を用いて、次のような書式で表現します。

/パターン/

また、JavaScriptでは、もう1つパターンを指定する書式があります。それは、

RegExpオブジェクトを使って指定する方法です。

var obj=new RegExp("パターン", "フラグ");

フラグには、後述する「g」「i」を指定できます(p.96参照)。

どちらの方法を使っても、できることはまったく同じです。Perlなどで正規表現に慣れている人は、「/」でパターンを囲む指定方法を、VBScriptなどで正規表現を使っていた人はRegExpオブジェクトを用いればよいでしょう。本書では、原則として「/」を用いた書式を使います。

パターンでは、多種多様な指定ができます。以下に、指定の仕方を解説していきましょう。

●1文字にマッチする

マッチする文字が、ある特定の1文字のときには、正規表現として、その文字自体を使います。たとえば、「a」が含まれる文字列をマッチさせたいときには、次のように記述します。

/a/

なお、「/ /」は、「マッチ演算子」と呼ばれるもので、ある文字列が正規表現で表現されたパターンにマッチするかどうかを調べる演算子です。

また、ある特定の1文字と任意の1文字にマッチさせたいときには、「.」(ピリオド)を使います。たとえば、「a」で始まる文字列を探したいときには、次のように記述します。

/a./

これにより「ab」「at」「az」といったものがマッチしたと認識されます。

これ以外に、ある複数の文字の中のどれかにマッチすればよい、という場合があります。このようなときに使うのが「文字クラス」です。文字クラスの書式は、次のようになります。

[文字クラス]

たとえば、「a」「b」「c」のうちのどれか1文字が入っていればよい場合には、次のように記述します。

/[abc]/

また、「-」を使って、その間の記述を省略することもできます。たとえば、何らかの数字1文字にマッチする場合には、次のように記述します。

/[0-9]/

さらに、「含まれていない場合」という否定的な指定をする場合には、「^」を使います。たとえば、数字以外の1文字にマッチしたい場合には、次のように記述します。

/[^0-9]/

これ以外に、「略記法」による指定もできます。主な、略記は以下のとおりです。

▼主な略記法

| 略記法 | 文字クラス | 意味 |
|-----|---------------|---------------------------------|
| ¥w | [a-zA-Z0-9_] | アルファベット、数字、下線 |
| ¥W | [^a-zA-Z0-9_] | アルファベット、数字、下線以外 |
| ¥s | [¥r¥t¥n¥f] | 空白文字(スペース、復帰文字、タブ、改行文字、ラインフィード) |
| ¥S | [^¥r¥t¥n¥f] | 空白文字以外 |
| ¥d | [0-9] | 数字 |
| ¥D | [^0-9] | 数字以外 |

●複数の文字にマッチする

ここまででは、1文字にマッチする場合を紹介してきました。しかし、これだけではなく、複数の文字列にマッチするかどうかを確認したい場合があります。たとえば、「img」で始まり、後に数字がついている文字列をまとめてマッチさせたい場合には、次のように記述します。

/img[0-9]*/

ここで気がつくのはアスタリスク(*)です。これは、「0回以上にマッチする」という意味です。たとえば、上記の例であれば「img」「img8」「img123」などにマッチします。このような記号のことを「量指定子」と呼びます。量指定子には、次のようなものがあります。

▼量指定子

| 記号 | マッチ回数 |
|--------|--------------|
| * | 0回以上にマッチ |
| + | 1回以上にマッチ |
| ? | 0回、1回にマッチ |
| {m} | ちょうどm回にマッチ |
| {m,} | m回以上にマッチ |
| {m, n} | m回以上n回以下にマッチ |

●マッチする位置を指定する

マッチさせたい場所が、行の先頭だけとか行の末尾だけといったことがあります。このようなときには、「位置指定子」を用いてマッチする位置を指定します。これは、単語を調べるときなどに重宝します。

▼位置指定子

| 位置指定子 | 意味 |
|-------|-----------------|
| ^ | 文字列(行)の最初にマッチする |
| \$ | 文字列(行)の最後にマッチする |
| \b | 単語の境界にマッチする |
| \B | 単語の境界以外にマッチする |

●マッチする条件を指定する

マッチするものを1つだけ探すのか、マッチするものをすべて探すのかといったことを指定したい場合があります。また、通常は大文字と小文字は区別されますが、大文字と小文字を区別したくない場合もあります。このときには、以下のようない修飾子で条件を指定できます。

▼条件を指定する修飾子

| 修飾子 | 意味 |
|-----|-----------------|
| g | マッチするものすべてを見つける |
| i | 大文字と小文字を区別しない |

これらの修飾子は、正規表現の後ろにつけて使います。

/a./g

このように指定すると、検索対象文字列から「ab」「az」「ac」などをすべて取り出します。

■正規表現を使う

正規表現のパターンを指定する方法は、すでに解説しました。では、指定したパターンを用いて実際に検索などを行うには、どのようにしたらよいのでしょうか？正規表現を用いる方法には、大きく分けて2つあります。1つは、Stringオブジェクトに実装されているmatchメソッド、replaceメソッド、searchメソッドなどを用いる方法です。もう1つは、RegExpオブジェクトが持っているメソッドなどを用いる方法です。

ここでは、簡単に正規表現を扱うことができる、Stringオブジェクトのメソッドを用いる方法を紹介しましょう。Stringオブジェクトの中には、次のようなメソッドがあります。

▼文字列を検索し、マッチした文字を返す

srtObj.match(regExp)

▼マッチした文字列を置換する

srtObj.replace(regExp, repStr)

▼文字列を検索し、マッチした位置を返す

srtObj.search(regExp)

これらはどれも、正規表現パターン(regExp)を引数に取り、マッチ処理、または置換処理を行います。マッチ処理のサンプルは、次のようにになります。

```
<html>
<title>RegExp match</title>
<body bgcolor="#ffff8dc">
<h3>RegExp search</h3>
<hr />
<script type="text/javascript">
    var myReg=/a./;
    var str="Javascript";
    document.write("検索対象: "+str+"<br />");
    document.write("検索文字: "+myReg.source+"<br />");
    strMatch=str.match(myReg);
    strSearch=str.search(myReg);
```

```
document.write("match-返される値: "+strMatch+"<br />");  
document.write("search-返される値: "+strSearch);  
</script>  
</body>  
</html>
```

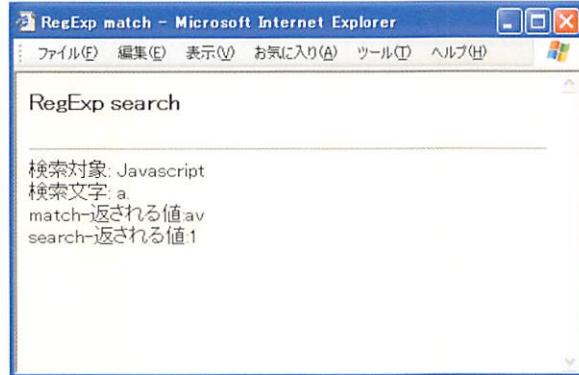
上記のサンプルでは、7行目で正規表現パターンmyRegを作成しています。マッチさせるのは、「a」から始まる、2文字の文字列です。8行目では、文字列オブジェクトstrを作成しています。検索対象となる文字列は「Javascript」です。

10行目のmyReg.sourceですが、RegExpオブジェクトのsourceプロパティを使って、正規表現パターンを参照しています。今回のケースでは、「a.」が返されます。

12行目と13行目で、マッチした文字とマッチした位置がそれぞれ返されます。

このサンプルを実行すると、次のように表示されます。

▼正規表現を使った処理のサンプル



なお、文字列「Javascript」には、「a」から始まる2文字の文字列が2つありますが(avとas)、実行結果ではavしか表示されていません。p.96で説明したとおり、すべてにマッチさせるためには、7行目を次のように書き換えます。

```
var myReg=/a./g;
```

例外処理

プログラムには、バグがつきものです。どんなに気をつけていても、どんなに優秀なプログラマーであっても、これは避けられない問題です。さらに、ユーザーはプログラマーがどんなに想像をめぐらせてても、思いもつかない動作をします。こういったバグや、想定外の動作により引き起こされる問題を、簡単にプログラム上で処理できるような機構がJavaScriptに組み込まれました。それが、「例外処理」です。例外処理は、問題が発生した際に動作します。

JavaScriptで例外処理が扱えるようになったのは、バージョン1.3からであり、ごく最近のことです。現在、例外処理をサポートしているのは、IE5以降、NN6以降、Safariです。なお、困ったことに、IEとNN、Safariでは例外処理の実装が若干異なります。そのため、どちらのWebブラウザかを限定できれば問題なく使うことができますが、両方に対応するように記述するときには、例外処理を分離することぐらいにしか使えません。

しかし、例外処理を使えば、プログラムの生産性を上げることができます。例外処理を用いないスクリプトでは、検出したいエラーごとにその処理を記述し、さらにそれを処理の近くに記述しなくてはなりません。その点、例外処理では、例外を検知したときの処理を「catch文」にまとめて記述できるため、正常時の処理と例外処理をきれいに分離できます。これは、プログラム制作時の効率だけでなく、メンテナンスが発生した際のプログラムの理解にとって有効です。

■例外処理の実装

例外処理を実装するには、以下のような構文を用います。

```
 try {
    处理;
}
catch(例外) {
    例外処理;
}
```

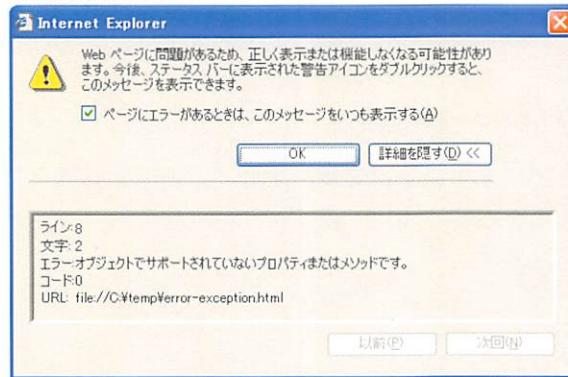
上記のコードでは、try文の後ろの中括弧「{ }」の中に、例外の発生を検知する必要のある処理を記述します。この処理中に問題が発生したときに、例外が投げられます。この例外を受け取るのが、catch文です。catch文では、try文で発生した例外を受け取り、引数の部分で処理します。慣例として、引数には「e」を用いることが多いようです。try文

には、少なくとも1つのcatch文、またはfinally文が必要となります。これらのどちらかがない場合に、スクリプトエラーになります。

では、実際に例外処理のスクリプトを作成してみましょう。まずは、例外処理を行っていないスクリプトを見てください。以下のプログラムでは、documentオブジェクトのwriteメソッドのつづりが間違っています。そのため、スクリプトエラーが発生します。

```
<html>
<title>Exception</title>
<body bgcolor="#ffff8dc">
<h3>例外処理</h3>
<hr />
<script type="text/javascript">
<!--
    document.wrtie("Error line");
//-->      └―――つづりが間違っている
</script>
</body>
</html>
```

▼IE6で表示されるエラー



それでは、このコードに例外処理を組み込んでみましょう。スクリプトは、次のようになります。

```

<html>
<title>Exception</title>
<body bgcolor="#ffff8dc">
<h3>例外処理</h3>
<hr />
<script type="text/javascript">
<!--
try {
    document.wrte("Error line");
}
-->
catch(e) {
    document.write("Error: "+e+"<br />");
}
//--
</script>
</body>
</html>

```

つづりが間違っている

例外処理は、すでに解説したとおり、try文、catch文により実装されます。問題の起こりそうな部分をtry文で囲み、対応した例外処理をcatch文内に記述します。上記プログラムでは、例外の内容を表示しています。実行結果は、次のようになります。

▼例外のサンプル



例外処理は、入れ子にして記述することもできます。そのため、複雑な処理に対して例外処理を組み込むことができます。

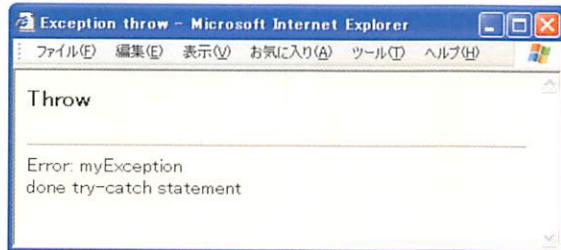
また、例外処理のtry文、catch文の処理を終了した後に行う処理を、finally文により記述できます。finally文は、例外処理が行われる、行われないにかかわらず、try文とcatch文の処理が終了した段階で実行されます。

実際のコードは、次のようにになります。

```
<html>
<title>Exception throw</title>
<body bgcolor="#ffff8dc">
<h3>Throw</h3><hr />
<script type="text/javascript">
<!--
try {
    throw "myException";
}
catch(e) {
    document.write("Error: "+e+"<br />");
}
finally {
    document.write("done try-catch statement");
}
//-->
</script>
</body>
</html>
```

上記のコードでは、例外処理を終了した後に、finally文内を処理します(try文中のthrow文については、次に解説します)。実行結果は、以下のようになります。

▼finally文のサンプル



■例外を発生させる

これまで、例外を処理する方法を解説してきました。しかし、これだけでは、まだ思いどおりに例外処理を使いこなすことはできません。必要となるのが、例外を発生させる方法です。スクリプト内で例外を明示的に発生させることができれば、エラー処理などが楽になります。

例外を発生させるには、`throw`文を用います。`throw`というくらいなので、日本語では「例外を投げる」などともいいます。ここで発生させた例外を受け取るのが、`catch`文なのです。

`throw`文の記述は、以下のようになります。

```
 try {
    处理;
    throw "式";
}
catch(例外) {
    例外処理;
}
```

`throw`文は、引数として式を指定します。`throw`文の引数には、どのような式でも記述することができます。引数に設定された式は、例外を識別するために使われます。

```
<html>
<title>Exception throw</title>
<body>
<h3>Throw</h3>
<hr />
<script type="text/javascript">
<!--
try {
    throw "myException";
}
catch(e) {
    document.write("Error: "+e);
}
-->
```

```
//-->
</script>
</body>
</html>
```

実行結果は、次のようにになります。

▼throw文のサンプル

