JavaScript 逆引きハンドブック

古籏 一浩◆著

基本的な処理や便利なTipsとともに、
HTML5のAPIを数多く掲載!

まさにWeb制作のバイブル!

HTML5
CSS3
対応!

「やりたいこと」からJavaScriptの機能を探せる!

24時間無料でサンプルデータをダウンロードできます。

C&R研究所

JavaScript
逆引きハンドブック

古籏 一浩◆著

C&R研究所

■権利について

- 本書に記述されている社名・製品名などは、一般に各社の商標または登録商標です。
- 本書では™、©、®は割愛しています。

■本書の内容について

- 本書は著者・編集者が実際に操作した結果を慎重に検討し、著述・編集しています。ただし、本書の記述内容に関わる運用結果にまつわるあらゆる損害・障害につきましては、責任を負いませんのであらかじめご了承ください。
- 本書で紹介しているコードの実行結果の画面などは、環境によって異なる場合がございますので、あらかじめ、ご了承ください。
- 本書の内容は、2012年7月現在の情報を基に記述しています。仕様の変更やブラウザのバージョンアップなどにより、サンプルの動作が変わったり、プログラムの書き換えが必要になったり、動作しなくなったりする場合があります。また、紹介しているURLなども変更になる場合があります。あらからじめ、ご了承ください。

■サンプルについて

- 本書で紹介しているサンプルは、C&R研究所のホームページ(http://www.c-r.com)からダウンロードすることができます。ダウンロード方法については、5ページを参照してください。
- サンプルデータの動作などについては、著者・編集者が慎重に確認しております。ただし、サンプルデータの運用結果にまつわるあらゆる損害・障害につきましては、責任を負いませんのであらかじめご了承ください。
- サンプルデータの著作権は、著者及びC&R研究所が所有します。許可なく配布・販売することは堅く禁止します。

●本書の内容についてのお問い合わせについて

この度はC&R研究所の書籍をお買いあげいただきましてありがとうございます。本書の内容に関するお問い合わせは、FAXまたは郵送で「書名」「該当するページ番号」「返信先」を必ず明記の上、次の宛先までお送りください。お電話や電子メール、または本書の内容とは直接的に関係のない事柄に関するご質問にはお答えできませんので、あらかじめご了承ください。

〒950-3122 新潟県新潟市北区西名目所4083-6　株式会社 C&R研究所　編集部
FAX 025-258-2801
「JavaScript逆引きハンドブック」サポート係

PROLOGUE

今日、JavaScriptは、Web制作において欠かせないものとなりました。今やJavaScriptが使われていないWebサイトを探す方が大変でしょう。

当初はフォームの入力チェック程度にしか使われていなかったJavaScriptが、現在ではWebアプリケーションまで担うようになったのはすごいことです。一時期、JavaScriptを動作しないようにブラウザを設定するような時代もありました。また、ブラウザ間の動作の違いによる問題に悩まされた時代もありました。今では、jQueryなど各種ライブラリの利用によってかなり解消されています。

そして、JavaScriptが誕生してから16年、さまざまな困難を乗り越えて現在に至っています。

特にその原動力となったのはAjax（非同期通信）、そしてHTML5によるところが大きいでしょう。HTML5のAPIを利用することにより、高度で複雑なことが簡単にできるようになりました。数年前は不可能だったことがJavaScriptで簡単にできるようになっているのです。たとえば、次のような機能をJavaScriptから使うことができます。

- 位置情報
- 2D画像処理、3D描画
- ビデオ再生/オーディオ再生
- 音声/波形処理
- カメラ撮影/ビデオ撮影/録音
- カメラからのリアルタイムキャプチャー
- 並列処理
- ファイル処理
- バッテリーやセンサーなどデバイス関係
- ソケット通信（双方向通信）
- データベース（Key Value Storeタイプ）

HTML5のAPIではさまざまな機能や仕様が追加・定義され、多くのブラウザで利用できるようになっています。特にスマートフォンで利用できるというのは、大きなポイントです。

本書は、最新のHTML5のAPIを利用したサンプルを数多く掲載しています。また、JavaScriptの基本的な処理や便利なTIPSなども多数、掲載しています。

HTML5は進歩が早い上に仕様が確定していないこともあり、将来的に動作しない可能性もあります。それでも、現時点で動作するという点において役立つ部分もあるはずだと考え、掲載しています。

本書が読者の皆様のWeb制作の一助となれば幸いです。

2012年7月

古籏一浩

本書について

動作確認の環境について

本書では、次のような環境で動作を確認しています。

- IE6～8：Windows XP
- IE9：Windows 7/Windows Phone 7.5
- IE10：Windows 8(Release Preview)
- Google Chrome 21：Windows7/Mac OS X 10.6/Mac OS X 10.7
- Chrome for Android：Android 4.0
- Firefox 14：Windows7/Mac OS X 10.6/Mac OS X 10.7
- Firefox(Android版)：Android 4.0
- Safari 5.x：Mac OS X 10.6/Mac OS X 10.7
- Safari 6：Mac OS X 10.8
- Safari(iOS版)：iOS 4/iOS 5.1
- Android 標準ブラウザ：Android2.3/3.2/4.0
- Opera 11～12：Windows 7

スマートフォン・タブレットについては、次の機種で確認しています。

- iOS 4/5：iPhone 4/4S、iPad 2/New iPad
- Android 2.3：Galaxy S/S2、AQUOS PHONE(SH12C)、XPERIA(SO-01C)
- Android 3.2：SONY Tablet(SGPT111JP/S)、Arrows Tab(F-01D)
- Android 4.0：Galaxy Nexus
- Windows Phone 7.5：Windows Phone IS12T

本書の表記方法

本書の表記についての注意点は、次のようになります。

▶サンプルコードの中の「\」(バックスラッシュ)と「¥」(円記号)について

環境によっては、「\」(バックスラッシュ)は「¥」(円記号)で表示されます。本書の誌面に掲載しているサンプルコードでは「\」で表記していますので、必要に応じて読み替えてください。なお、Unicodeでは「¥」と「\」は異なる文字コードになっています。このため、一部の書体を除き、「\」を「¥」で入力すると動作しません。

▶サンプルコードの中の▼について

本書に記載したサンプルコードは、誌面の都合上、1つのサンプルコードがページをまたがって記載されていることがあります。その場合は▼の記号で、1つのコードであることを表しています。

▶ブラウザの対応バージョンについて

本書では項目のタイトル部分に、ブラウザの対応バージョンのマークを記載しています。「WP」はWindows Phoneを表しています。基本的には記載しているバージョン以降でその項目（メソッドなど）が対応となりますが、一部、例外があります。その場合は、HINTやCOLUMNに記載しています。なお、薄く表示されている場合は、非対応となります。

サンプルファイルのダウンロードについて

本書のサンプルデータは、C&R研究所のホームページからダウンロードすることができます。本書のサンプルを入手するには、次のように操作します。

❶「http://www.c-r.com/」にアクセスします。
❷トップページ左上の「商品検索」欄に「108-5」と入力し、[検索]ボタンをクリックします。
❸検索結果が表示されるので、本書の書名のリンクをクリックします。
❹書籍詳細ページが表示されるので、[サンプルデータダウンロード]ボタンをクリックします。
❺下記の「ユーザー名」と「パスワード」を入力し、ダウンロードページにアクセスします。
❻「サンプルデータ」のリンク先のファイルをダウンロードし、保存します。

サンプルのダウンロードに必要な
ユーザー名とパスワード

ユーザー名 **jsgh**
パスワード **f6h3t**

※ユーザー名・パスワードは、半角英数字で入力してください。また、「J」と「j」や「K」と「k」などの大文字と小文字の違いもありますので、よく確認して入力してください。

サンプルコードの利用方法

サンプルファイルは、CHAPTERごとのフォルダの中に、項目番号のフォルダに分かれています。サンプルはZIP形式で圧縮してありますので、解凍してお使いください。

それぞれのフォルダ内には、HTMLファイルやJavaScriptファイル、CSSファイルなどが保存されています。

なお、サンプルによっては、ローカルディスク上（HTMLファイルを直接、ブラウザで開く方法）では動作しない場合があります。その場合は、動作の確認を行うにはネットワーク環境（Webサーバー上）で確認してください。Webサーバーについては、ローカルサーバーでも構いません。

また、本書のサンプルでは、主に「addEventListener()」メソッドを使っています。IE6～8では「addEventListener()」メソッドがないため、そのままでは動作しません。IE6～8では、「attachEvent()」メソッドを使うように書き換える必要があります（400ページ参照）。

CONTENTS

CHAPTER 01 JavaScriptの基礎知識と基本文法

CHAPTER 02 オブジェクト

CHAPTER 05 文字列・正規表現

CHAPTER 06 日付・時刻

CHAPTER 07 ブラウザオブジェクト

CHAPTER 09 イベント

CHAPTER 13 HTML5のその他のAPI

CHAPTER 01

JavaScriptの基礎知識と基本文法

SECTION-001

JavaScriptの概要

JavaScriptとは

JavaScriptは数あるスクリプト言語の1つです。JavaScriptは多くのブラウザに搭載されており、Webアプリケーションの実現に欠かせない言語となっています。特にHTML5の登場で3D処理や並列処理、カメラからのリアルタイム取り込みなど、これまでよりも大幅に実現できる処理が増えており、従来のJavaScriptとは役割も作成されるプログラムの規模や複雑さも大きく変わってきています。このため、作成したJavaScriptプログラムが正しく動作するかをテストするJSUnit（http://www.jsunit.net/）やQUnit（http://qunitjs.com/）などのツールも登場しています。もちろん画像を入れ替えるだけ、入力フォームのチェックを行うだけといった従来通りの簡単な処理もできます。

▶さまざまな場面で使われている

JavaScriptはInternet ExplorerやFirefox、Safari、Google Chromeなどのブラウザに搭載されているため、ブラウザ専用のスクリプト言語と思われがちです。しかし、JavaScriptはOSであるWindows、Adobe Photoshopなどのアプリケーション、Adobe FlashやPDFなどにも搭載されており、さまざまな場面で使われています。また、HTML5（http://www.w3.org/TR/html5/）の普及に伴ってスマートフォン上で動作するアプリケーションを駆動する言語としても利用され始めています。たとえば、Titanium（http://www.appcelerator.com/）やPhoneGap（http://phonegap.com/）などのスマートフォン向けアプリ開発のフレームワークでJavaScriptが使われています。

▶JavaScriptの基本仕様

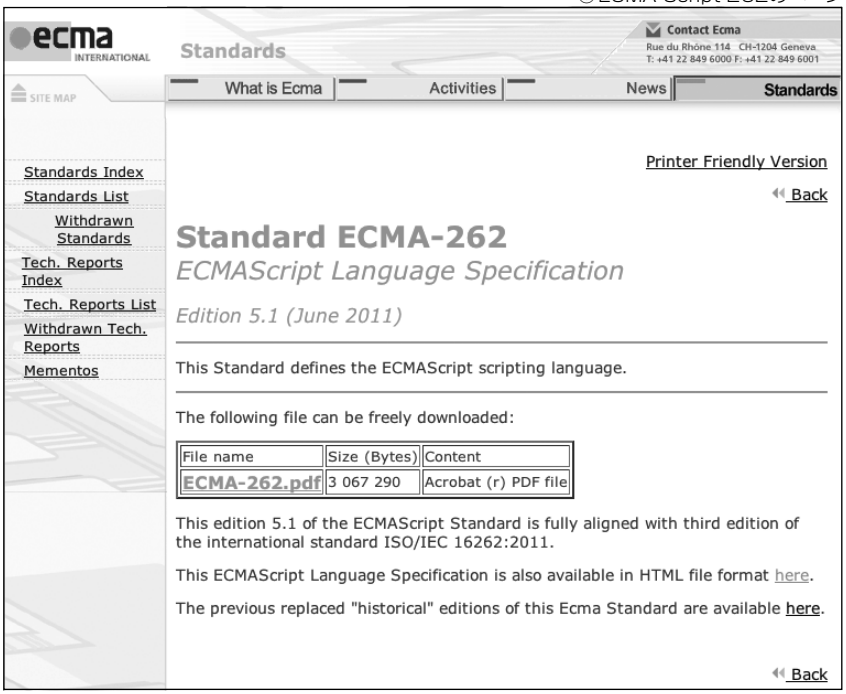
JavaScriptの基本となる部分の仕様はECMAにより公開されており、ECMA Scriptと呼ばれています。この基本部分にブラウザやアプリケーションが独自のオブジェクトを追加したものが、一般的にいわれるJavaScriptです。

また、マイクロソフト社の場合は、JavaScriptと互換性を持った独自拡張のスクリプト言語であるJScriptとしてブラウザなどに実装されています。JScriptではWindows固有のActive Xを利用した処理（ファイル処理などローカル環境の制御など）も可能になっています。ただし、HTML5の普及に伴って直接、ファイル処理やデバイスへのアクセスが可能になってきているため、Active Xを利用した処理の必要性は低下しています。

- ECMA Script-262

  URL http://www.ecma-international.org/publications/standards/Ecma-262.htm

◉ECMA Script-262のページ



▶プロトタイプ指向のオブジェクト指向言語

JavaScriptはJavaなどの一般的なクラスを利用したものと異なり、基本となるオブジェクトを拡張・派生させるプロトタイプ指向のオブジェクト指向言語です。ただし、当初、実装されたブラウザであるNetscape 2と現時点のものでは異なっている部分が多くあります。当初はプロトタイプによる継承もなく、単純なオブジェクト指向言語で、変数名や文字数制限、利用できるオブジェクトなど、多くの制限がありました。これが、次第に改良されて現在に至っています。

▶ライブラリ

ブラウザの普及とともにJavaScriptも改良されたとはいえ、ブラウザによって動作が異なることも多く、制作者泣かせの言語であることも確かです。これを解消するのがJavaScriptライブラリです。2012年現在で最も多く利用されているのがjQueryライブラリです。jQueryは短いコードで多くの処理を実現できるのが魅力の1つです。また、UI処理を行うためのライブラリやスマートフォン向けのjQuery Mobileも用意されています。その他のJavaScriptライブラリとしては中規模・イントラネット向けのExtJS(SenchaTouch)、大規模向けのDojo ToolKitなどがあります。

JavaScriptの基本事項

JavaScriptの記述場所

JavaScriptプログラムのコードは、HTML文書内や外部ファイルとして記述することができます。JavaScriptのプログラムは、読み込まれるとすぐに実行が開始されます。

実際に記述する場所については、次のようになります。

▶HTML/XHTMLファイル内に記述する場合

HTML/XHTMLファイル内に記述する場合は、文書内にプログラムコードを書きます。基本的には次のように、<script>～</script>タグ内に記述します。HTML4の場合は<script>タグに対応していないブラウザでプログラムが表示されないようにHTMLのコメント「<!--」と「// -->」で囲みます。HTML5の場合は<script>タグだけでよく、「type」属性の指定やコメントは不要です。

XHTMLの場合は、本来は別ファイルにするのがよいのですが、どうしてもXHTMLファイル内に記述する必要がある場合は「//<![CDATA[」と「//]]>」で囲みます。<script>タグは、特に<head>タグ内でなくてもよく、<body>タグで囲まれた範囲でも問題なく記述できます。

◉HTML4の場合

```
<script type="text/javascript">
<!--
  ここにスクリプトを書く
//-->
</script>
```

◉HTML5の場合

```
<script>
  ここにスクリプトを書く
</script>
```

◉XHTMLの場合

```
<script type="text/javascript">
//<![CDATA[
  ここにスクリプトを書く
//]]>
</script>
```

また、HTML4でデフォルトのスクリプト言語としてJavaScriptを指定する場合は、次の<meta>タグを指定しておきます。

```
<meta http-equiv="Content-Script-Type" content="text/javascript">
```

▶外部ファイルとして記述する場合

外部ファイルとして使用する場合は、純粋にスクリプトだけを記述したテキストファイルを作成し、<script>タグの「src」属性でファイル名を指定します。これはHTML4でもHTML5でも同様です。

```
<script src="ファイル名.js"></script>
```

ファイル名の末尾（拡張子）は「.js」にする必要があります。「src」属性には拡張子が「.js」のファイルだけでなく、CGIのパスなどを記述して、結果をJavaScriptコードで返すこともできます。また、JSON形式など、データだけを取得したい場合にも利用することができます。一度、読み込まれたスクリプトはメモリ内に残るため、<script>タグを削除しても実行することが可能です。

また、外部ファイルの場合は「同期」で読み込まれるため、プログラムサイズが巨大であったり、解析に時間がかかったりすると、ブラウザの反応速度が低下し、ページ表示までに時間がかかる場合があります。そこで、<script>タグには、非同期でスクリプトを読み込ませて実行する属性が用意されています。ファイルを非同期で読み込ませる場合には、<script>タグに「async」属性（非同期で読み込まれ実行）や「defer」属性（非同期で読み込み、実行はページ構築後）を指定します。ただし、「async」属性を指定すると、実行順序が保証されないので注意が必要です。

HTML5のWeb Workersを利用している場合は、「importScripts()」メソッドにより、スクリプトファイルを読み込ませることができます。この場合、Web Workersが非同期なので、メイン処理とは別に読み込まれます。ただし、「importScripts()」メソッドで読み込むスクリプトの読み込み順序と実行順序は保証されます。

▶タグ内に記述する場合

タグ内に記述する場合には、イベント名を記述し、「=」の後ろに実行するプログラムを指定します。イベント名は「on」を付加した名称になります。たとえば、「click」イベントをタグ内に記述する場合は、「onclick」となります。多くの場合、関数を呼び出すようになっていますが、通常のプログラムコードをそのまま記述することができます。

```
<a href="#" onclick="alert('ok?')">〜</a>
```

「onclick」や「onmouseover」などのイベント名は、大文字で書いても小文字で書いても動作します。大文字と小文字が混在していても問題ありません。ただし、プログラムでイベントハンドラとして定義する場合にはすべて小文字にする必要があります。

SECTION-003

演算子について

JavaScriptで使える演算子

JavaScriptには基本的な演算子のみ用意されており、高度な演算を行う場合は「Math」オブジェクトを利用するようになっています。JavaScriptで使用できる演算子は、次の表のようになります。ビットシフトなどのビット演算は32ビット長で処理されます。ビット演算以外の計算に関してはIEEE 754の規格に沿って処理が行われます。IEEE 754の規格については、次のURLを参照してください。

- IEEE 754 - Wikipedia

  URL http://ja.wikipedia.org/wiki/IEEE_754

演算子	意味	記述例	演算の結果 (x=12、y=4とした場合)
+	加算	z = x + y	16
−	減算	z = x − y	8
*	乗算	z = x * y	48
/	除算	z = x / y	3
%	剰余	z = x % y	0
++	プリインクリメント	y = ++x	y=13、x=13 (代入前に加算)
++	ポストインクリメント	y = x++	y=12、x=13 (代入後に加算)
−−	プリデクリメント	y = −−x	y=11、x=11 (代入前に減算)
−−	ポストデクリメント	y = x−−	y=12、x=11 (代入後に減算)
−	符号反転	y = −x	−12
&	論理積	15 & 9	9 (1111 & 1001 = 1001)
&&	論理積(論理式)	(x == 12) && (y == 4)	true
|	論理和	15 | 9	15 (1111 | 1001 = 1111)
||	論理和(論理式)	(x == 12) || (x == 9)	true
^	排他的論理和	15 ^ 9	6 (1111 ^ 1001 = 0110)
!	否定	!(x == 12)	false
<<	符号付き左シフト	9<<2	36(1001<< = 100100)
>>	符号付き右シフト	9>>2	2(1001>> = 10)
>>>	符号なし右シフト	19>>>2	4(10011>>> = 100)

代入演算子

JavaScriptには、代入演算子も用意されています。使用できる代入演算子は、次の表のようになります。

代入演算子	記述例	「=」を使った記述
=	x = y	なし
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y

比較演算子

比較演算子は、値の比較を行います。結果は「true」か「false」となります。

比較演算子	意味
==	式が等しい
===	式が等しく、型も等しい
!=	式が等しくない
>	左が大きい
>=	左が等しいか大きい
<	右が大きい
<=	右が等しいか大きい

三項演算子

JavaScriptには、条件によってどちらか1つの値を取る三項演算子も用意されています。三項演算子は、「?」と「:」(コロン)で区切って指定します。たとえば、変数「b」の値に応じて代入する文字列を変えるには、次のように記述します。

```
a = (b > 2) ? "2以上" : "2未満"
```

条件を満たした場合には「?」直後の値が代入され、条件を満たしていない場合には「:」より後ろの値が代入されます。なお、三項演算子については、59ページも参照してください。

演算子の優先順位

演算子には優先順位が決められています。優先順位は、次の表のようになります。なお、表の上が優先順位が高く、下になるほど優先順位が低くなります。

優先順位	演算子/特殊演算子	記号/キーワード
1	メンバー、オブジェクト生成	. [ ] new
2	カッコ/関数呼び出し	( )
3	インクリメント、デクリメント	++ --
4	否定(NOT)、正数記号(+)、符号反転(−)	! ~ + - typeof void delete
5	乗算、除算、剰余	* / %
6	加算、減算	+ -
7	ビットシフト	<< >> >>>
8	大小、プロパティチェック	< <= > >= in instanceof
9	比較	== != === !==
10	ビット演算/論理積(AND)	&
11	ビット演算/排他的論理和(XOR/EOR)	^
12	ビット演算/論理和(OR)	|
13	論理的AND	&&
14	論理的OR	||
15	条件(三項演算子)	?:
16	代入	= += -= *= /= %= <<= >>= >>>= &= ^= |=
17	カンマ	,

COLUMN 予約語について

JavaScript(ECMA Script ver 5)では、次のキーワードは予約語となっているため、変数名などで使用しないようにしてください。なお、constやletなど、一部はすでにFirefoxなどのブラウザで使用されています。

- class
- const
- enum
- export
- extends
- implements
- import
- interface
- let
- package
- private
- protected
- public
- static
- super
- yield

リテラルについて

JavaScriptのリテラル

リテラルは定数という意味で、その名通り、決まっている値です。プログラム中に直接、記述可能な文字列や数値などです。JavaScriptのリテラルは、次のようになります。

▶数値

数値は8進数、10進数、16進数や、eを使った指数表記などを使用することができます。ECMA-262 5.1thでは、10進数と16進数の数値となります。Strictモードの場合も、10進数と16進数です(160ページ参照)。

▶文字列

文字列は、「'」(シングルクォーテーション)、または、「"」(ダブルクォーテーション)で囲んで表現します(194ページ参照)。

▶真偽値

「真」か「偽」を表します。値は「true」(真)、または、「false」(偽)となります(116ページ参照)。

▶正規表現

正規表現は、「/」(スラッシュ)で囲んで表現します(213ページ参照)。

▶配列

配列は、「[」と「]」の間に、値を「,」(カンマ)で区切って記述します(120ページ参照)。

▶オブジェクトリテラル

オブジェクトは、「{」「}」の間にプロパティと値をセットにして「key : value」の形式で、「,」(カンマ)で区切って記述します(90ページ参照)。

▶関数リテラル(無名関数/匿名関数)

関数についても、関数リテラルを使用することができます。これは、無名関数や匿名関数と呼ばれています(62ページ参照)。

▶特殊記号やUnicode文字

改行コードやタブなどの特殊記号やUnicode文字は、「\値」の形式で記述します。

▶null

「null」(ヌル)は、何もないことを示します。

strictモードとバージョンの指定について

JavaScriptの実行バージョンの指定について

JavaScriptには、いくつかのバージョンがあります。Firefoxでは、「script」要素の「type」属性で明示的にバージョンを指定することができます。たとえば、次のようにするとJavaScriptのバージョン1.7を使用することを指定できます。

```
<script type="text/javascript;version=1.7">
```

「1.7」の部分がバージョンになるので、他のバージョンを指定する場合は、この値を変えればよいことになります。JavaScriptのバージョン指定はFirefoxのみ対応しているため、他のブラウザでは利用できません。

strictモードについて

IE6～9を除くブラウザでは、JavaScriptの実行モードを指定することができます。これはstrictモードと呼ばれ、JavaScriptの曖昧さを回避できるため、プログラム作成上のミスを減らすことができます。strictモードを使用するには、「"use strict";」とだけ記述します。関数内だけ、strictモードを有効にすることもできます。「"use strict";」を記述すると、次のような制限や違いが発生します。

- varなしでグローバル変数に代入するとエラー
- 8進数表記の禁止
- 「with」の使用不可
- 「arguments.callee」の使用不可
- 「eval()」スコープ範囲の違い
- 「NaN」などに代入するとエラー
- 「eval()」、「arguments」の名前に対してバインドや代入不可
- 関数での「arguments」はエイリアスにならずに当初の引数を保持
- 関数内での「this」の扱いの変更（指定された「this」を変更しない）
- 書き込み不可のプロパティに代入するとエラー
- 拡張不可のオブジェクトへの新規プロパティ作成でエラー
- 削除できないプロパティを削除するとエラー
- 重複するオブジェクトリテラルがあるとエラー
- 関数/メソッドのパラメーター名が重複するとエラー
- 関数内で「関数名.caller」「関数名.arguments」指定はエラー
- 「implements」「interface」「let」「package」「private」「protected」「public」「static」「yield」の単語は予約語のため使用不可

なお、iOS 5、Androidでもstrictモードに対応しています。ただし、エラーとなった場合、「try...catch」文で捕捉できないことがあります（次のサンプルも動作結果が異なる）。

サンプルでの動作の確認

実行バージョンの指定とstrictモードの指定についてのサンプルは、次のようになります。各ブラウザでindex.htmlを表示してください。

SAMPLE CODE HTMLのコード（index.html）

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
  </head>
  <body>
    <output></output>
    <script type="text/javascript;version=1.7">
      // letはFirefoxでのみ動作
      let n = 123.45;
      // nの内容を出力
      var ele = document.getElementsByTagName("output")[0];
      ele.innerHTML = "nの値 :"+n+"<br>";
    </script>
    <script type="text/javascript">
      // "use strict";
      var ele = document.getElementsByTagName("output")[0];
      // ここはstrictモードではなく通常動作。グローバル変数に代入可能
      gexpress = 999;
      ele.innerHTML += "gexpressの値 :"+gexpress+"<br>";
      // 以下の関数はstrictモード
      function execTest(){
        // strictモードに設定
        "use strict";
        try{
          // strictモードではvarなしグローバル変数代入はエラー
          arcadia = 5;
          ele.innerHTML += "arcadiaの値 :"+arcadia+"<br>";
        }catch(e){
          ele.innerHTML += "arcadiaに代入できません<br>";
        }
      }
      execTest();
    </script>
  </body>
</html>
```

FirefoxではJavaScriptのバージョンを指定して処理することができるため、変数「n」の値が表示されています。

Firefox、Safari、Opera、IE10、Google Chromeではstrictモードに対応しているため、varなしでグローバル変数に代入するとエラーになります。

IE9ではstrictモードに対応していないため、グローバル変数に代入した結果が表示されます。

コメントについて

JavaScriptのコメント

JavaScriptのコメント(注釈)は、2種類が用意されています。1つは1行コメントである「//」です。これは、「//」より後ろに続く文字から行末(改行コード)までをコメントとするものです。もう1つが複数行をコメントとする「/*～*/」です。「/*」と「*/」で囲まれた部分はすべてコメントとみなされ、処理されません。

```
var a; // これ以降は行末(改行コード)までをコメントとする

var b; /* この部分はコメントとなる */

/*
  このコメントでは、複数行のコメントを
  記述することができる

*/
```

たとえば、次のコードでは最初のアラートダイアログはコメントになっていないので表示されますが、その次の行にある「alert()」メソッドはコメントになっているので表示されません。また、ページ内にOKの文字が表示されますが、以後に続く処理はコメントになっているので処理されません。

SAMPLE CODE HTMLのコード(index.html)

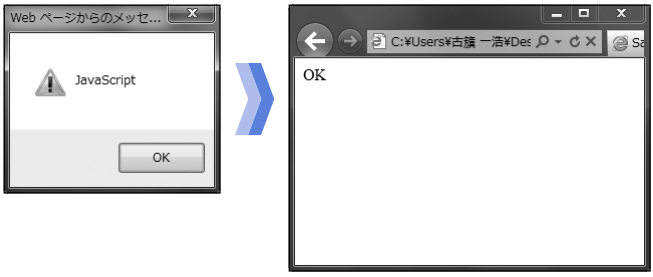
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <output></output>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  alert("JavaScript");
  // alert("OK");
  document.getElementsByTagName("output")[0].innerHTML = "OK";
  /*
  document.getElementsByTagName("output")[0].innerHTML = "NG";
```

```
  alert("NG!");
  */
}, false);
```

ブラウザでindex.htmlを表示すると、次のように表示されます。



変数と定数について

JavaScriptでの変数の定義

JavaScriptでの変数宣言には、いくつかの方法があります。JavaScriptでは、「var」や「const」を使わずに、いきなり変数を使用することができます。「var」宣言しない場合、変数はグローバル変数として定義されます。また、関数外では「var」宣言してもしなくても、グローバル変数として定義されます。グローバル変数ではなく、ローカル変数として宣言する場合には、関数ブロック内で「var」を使って宣言します。ブロック内でのみ有効な変数にしたい場合には「let」がありますが、これはFirefoxでしか使用することができません。

変数をまとめて宣言する場合は、「,」(カンマ)で区切って1行で記述します。また、変数の宣言と同時に初期値を代入することもできます。これを変数の初期化といいます。

```
// 変数「apple」を宣言する
var apple;

// 変数「banana」「cherry」をまとめて宣言する
var banana, cherry;

// 宣言と同時に初期値を指定する
var apple = 123;

// 複数の変数にそれぞれ初期値を指定してまとめて宣言する
var banana = 456, cherry = 789;

// var宣言なしで変数「orange」を定義する(グローバル変数として定義される)
orange = "OK";

// ブロック内でのみ有効な変数を定義する(Firefoxのみ)
let n ="NG";
```

▶変数名の制限

変数名については、次の制限があります。

- 最初が英文字、または、「_」(アンダーバー)で始まる
- 2文字目以降は、英数字と「_」(アンダーバー)を使用することができる
- 大文字と小文字は区別される(例:「abc」と「Abc」は別の変数とみなされる)
- 予約語を変数名にすることはできない

```
// 次の変数は有効
var abc // 英字で始まる
var _abc //「_」から始まる
var abc_123;
```

```
// 変数「abc」と変数「Abc」は区別される
var abc;
var Abc;

// 次の変数は無効
var 123; // 数字で始まる
var var; // 予約語
```


▶代入

変数への値の代入は、「=」演算子を使います。また、「a = b = c = 123;」のように一括して同じ内容を代入することもできます。

```
a = 123;            // 変数「a」に「123」を代入する
a = b;              // 変数「a」に変数「b」の値を代入する
a = b = c = 123;    // 変数「a」「b」「c」に「123」を代入する
```

▶変数で扱えるデータ

JavaScriptの変数は非常に柔軟な仕様になっており、どんなデータ（数値や文字列、関数、オブジェクト、メソッド、プロパティ）でも入れることができます。つまり、変数の型を問いません。たとえば、「a = 12.3」と変数「a」に数値の「12.3」を入れた後で、「a = "OK"」としても問題ありません。ただし、変数の型を問わないのは非常に便利な反面、型の不一致によるエラーや思わぬ結果を引き起こすことがあります。これを防ぐには、変数の型を調べてから処理するのが安全です。変数の型を調べる方法については、112ページを参照してください。

▶注意点

変数に数値や文字列を代入した場合はそのものが変数になりますが、オブジェクトなどの場合は注意が必要です。変数にオブジェクトを代入すると、オブジェクトそのものがコピーされるのではなく、オブジェクトへの参照が代入されます。

たとえば、次のコードでは、変数「a」にページ全体のスタイルシートオブジェクトへの参照を代入しています。その後、変数「b」に変数「a」を代入していますが、ページ全体のスタイルシートオブジェクトが複製されて変数「b」に代入されているわけではありません。単純にオブジェクトへの参照だけが変数「b」に代入されます。このため、コードを実行すると、ページ全体の背景色が赤になります。「b = a」は「b = document.body.style」と等価になるということです。

```
a = document.body.style;
b = a;
b.backgroundColor = "red";
```

また、ブラウザに実装されているJavaScriptでは、単純に変数を作成すると、「window」オブジェクトのプロパティとして追加されます。つまり、グローバル変数という名目になっていますが、実際には「window」オブジェクトのプロパティとして割り当てられています。これは、「for...in」文で「window」オブジェクトの中身を確認するとわかります。このため、「window」オブジェクトのプロパティと変数名が重複してしまうと、正常に動作しなくなるので注意してください。

次のコードを実行すると、変数「a」の内容も「window」オブジェクトのプロパティ「a」の内容も同じ文字が表示されます。

```
a = "OK";
// 変数「a」の内容を表示する
alert(a);
// 「window」オブジェクトのプロパティ「a」の内容を表示する
alert(window.a);
```

▶変数の削除

変数を削除するには、「delete」演算子を使います（117ページ参照）。ただし、「var」宣言した変数を削除することはできません。

JavaScriptでの定数の定義

「var」を使って宣言した変数の内容は、自由に変更できます。内容を変更してほしくない場合には「var」でなく、「const」を使います。「const」を使って宣言する場合には、定数名（変数名）と同時に値も設定する必要があります。定数名の制限などは、変数と同様です。なお、「const」はIE9（Windows Phone含む）では動作しません。

```
// constで定数として宣言
const android = 1;
// constで宣言した変数は変更ができないのでエラー
android = android + 1;
```

COLUMN **constでの不具合**

Operaでは「const」宣言はできますが、変数の内容を変更することができてしまいます。

同名の変数を「try...catch」文内で「const」宣言し、「catch」ブロック内で「var」宣言した場合、Firefox/Google Chromeではエラーになりますが、Safari/Operaではエラーになりません。

```
try{
  const mac = "SE/30";
  alert("constを使用しました");
}catch(e){
  alert("constは使えないのでvarで代用します");
  var mac = "SE/30";
}
```

繰り返し処理(ループ)について

JavaScriptでの繰り返し処理の種類

JavaScriptには繰り返し処理を行う命令として、「for」文、「while」文、「do...while」文が用意されています。

「for」文を使った繰り返し処理

「for」文は、一定の回数、処理を繰り返します。「for」文の構文は、次のようになります。

```
for (初期値; 繰り返し条件; 増減値){
  繰り返す処理
}
```

たとえば、「0」から「9」までの合計を求めるには、次のように記述します。

```
n = 0;
for (i=0; i<10; i++){
  n = n +i
}
```

初期値・繰り返し条件・増減値は「,」(カンマ)で区切って複数、指定することができます。たとえば、次のように記述することができます。

```
n = 0;
m = 0;
for (i=0, j=1; i<10, j<5; i++, j++){
  n = n + i;
  m = m + j;
}
```

この場合、「j」の値が「5」以上になったら終了するので、変数「n」は「6」、変数「m」は「10」となります。

増減値を指定する場合、ポストインクリメント(i++)よりもプリインクリメント(++i)の方が処理が速い場合もあります。

また、繰り返しを途中で抜ける場合には、「break」文を使います。繰り返しの先頭に戻る場合には、「continue」文を使います。

▶ 初期値・繰り返し条件・増減値の省略

初期値・繰り返し条件・増減値のすべてを省略することが可能です。すべて省略する場合は、次のように記述します。

```
for (;;)
```

ただし、これは無限に繰り返すことになり、Web Workers上で動作させる場合以外では、ブラウザが停止状態になってしまうことがあります。このような場合には、次のように「break」文を使って繰り返し処理から抜け出すことができます(直近の繰り返し処理から抜ける)。

```
n = 0;
for (;;){
  n++;
  if (n > 10) break;
}
```

▶サンプルでの確認

次のコードでは、「0」から「9」までの文字と、「10」から「1」までの文字が表示されます。

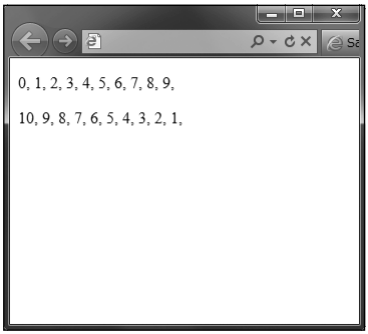
SAMPLE CODE HTMLのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <p><output></output></p>
    <p><output></output></p>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  // 文字入れる変数を用意
  var total = "";
  // 10回繰り返す
  for(var i=0; i<10; i++){
    total = total + (new String(i)) + ", ";
  }
  // 結果を出力
  document.getElementsByTagName("output")[0].innerHTML = total;
  // 文字入れる変数を初期化
  total = "";
  // 10回繰り返す
  for(var i=10; i>0; i--){
    total = total + (new String(i)) + ", ";
  }
  // 結果を出力
  document.getElementsByTagName("output")[1].innerHTML = total;
}, false);
```

ブラウザでindex.htmlを表示すると、次のように表示されます。



「while」文、「do...while」文を使った繰り返し処理

条件を満たしている間、繰り返し処理を行うには、「while」文、または、「do...while」文を使います。構文はそれぞれ、次のようになります。

```
while (条件式){
  繰り返す処理
}
```

```
do{
  繰り返す処理
} while (条件式);
```

「while」文は繰り返し前に条件を調べるため、条件によっては一度も処理が行われない場合があります。「do...while」文の場合は最後に条件を調べるため、最低1回は繰り返し処理が行われます。また、「while」文、「do...while」文とも、「break」文を使って繰り返しから抜けることができます。

▶サンプルでの動作確認

次のコードでは、最初のループでは10回繰り返されます。2回目のループは条件を満たしていない場合でも最低1回は実行されるため、「100」の数値が表示されます。

SAMPLE CODE HTMLのコード(index.html)

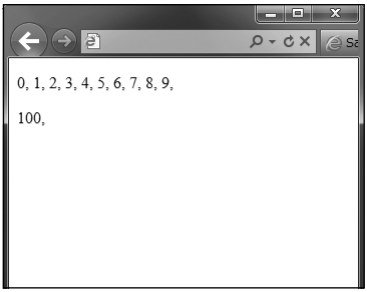
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <p><output></output></p>
    <p><output></output></p>
```

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

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  // 文字を入れる変数を用意
  var total = "";
  // 10回繰り返す
  var count = 0;
  while(count <10){
    total = total + (new String(count)) + ", ";
    count++;
  }
  // 結果を出力
  document.getElementsByTagName("output")[0].innerHTML = total;
  // 文字を入れる変数を用意
  total = "";
  // 繰り返す。do...whileは条件判断が最後なので最低1回は実行される
  count = 100;
  do{
    total = total + (new String(count)) + ", ";
    count++;
  }while(count <10)
  // 結果を出力
  document.getElementsByTagName("output")[1].innerHTML = total;
}, false);
```

ブラウザでindex.htmlを表示すると、次のように表示されます。



COLUMN DOMを繰り返し操作する場合

DOMのノードを繰り返し操作する場合は注意が必要です。DOMのノードを削除するような場合には、動的にノードが変化します。このとき、DOMノードの先頭から処理するとうまくいかない場合があります。このような場合にはDOMノードの末尾から処理するようにします。

条件分岐について

条件に応じて処理を分けるには

JavaScriptで条件に応じて処理を分けるには、「if」文、「switch」文、三項演算子を使います。

「if」文

「if」文は、条件を満たした場合に、以後に続く処理を行います。条件を満たしていない場合の処理も同時に記述するには、「else」節を使います。複数の条件で分岐させたり、ブロック内でさらに「if」を記述することもできますが、あまり条件判断が複雑になるようであれば、関数やメソッドなどでまとめるなどの工夫が必要になります。「if」文の構文は、次のようになります。

```
if（条件）{
  条件を満たした場合の処理
}
```

```
// 条件を満たしていない場合の処理も記述する
if（条件）{
  条件を満たした場合の処理
} else {
  条件を満たさなかった場合の処理
}
```

```
// 複数の条件で分岐させる
if（条件1）{
  条件1を満たした場合の処理
} else if（条件2）{
  条件2を満たした場合の処理
} else {
  条件1、条件2のどちらも満たさなかった場合の処理
}
```

なお、「if」に続く処理が1行のときは、「{}」を省略して記述することができます。

```
if（条件）条件を満たした場合の処理
```

「switch」文

「if」文はYes/Noの判断には向いていますが、複数の条件をまとめて処理する場合には、「switch」文を使います。「case」節に一致する値や文字列を指定し、「:」（コロン）で区切って直後に処理する命令を記述します。ここで処理を終わらせる場合には、「break」文を使って「switch」から抜けます。「break」文を記述しないと、次の「case」節の処理も実行されるの

で注意してください。

「case」節で指定した以外の値でも処理を行いたい場合には、「default」節を使います。「default」節は、省略することができます。

```
switch(調べる対象){
  case 値1 : 調べる対象と値1が一致した場合の処理
      break;
  case 値2 : 調べる対象と値2が一致した場合の処理
      break;
  default : caseで指定した以外の値の場合の処理
}
```

三項演算子

条件があまり複雑ではなく、Yes/Noだけの判別ができる場合には、三項演算子を使う方法もあります。三項演算子については、43ページも参照してください。

```
条件式 ? 条件を満たした場合の処理 : 条件を満たさなかった場合の処理
```

サンプルでの動作確認

次のコードでは、ダイアログで数値を入力し、入力値された値によって処理を分岐しています。

SAMPLE CODE HTMLのコード(index.html)

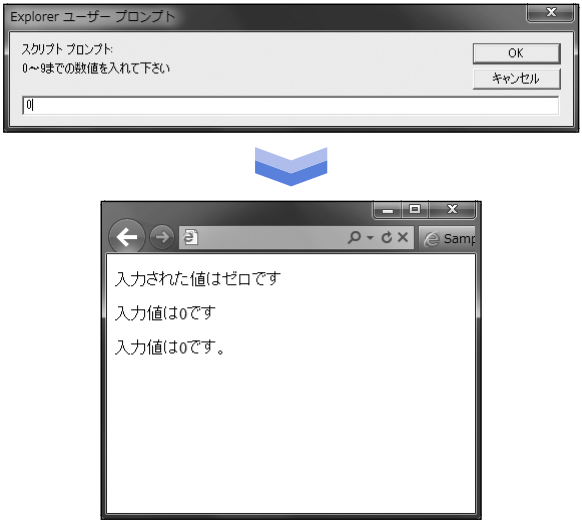
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <p><output></output></p>
    <p><output></output></p>
    <p><output></output></p>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

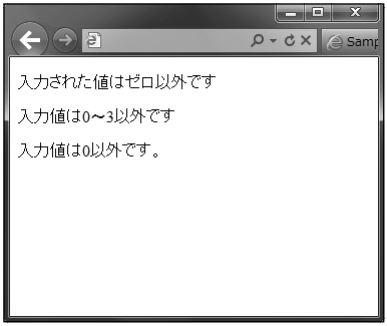
```
window.addEventListener("load", function(){
  var ele = document.getElementsByTagName("output");
  // ダイアログで数値を入力
  var num = prompt("0〜9までの数値を入れて下さい", 1) | 0;
  // 0かどうか調べる
  if (num === 0){
    ele[0].innerHTML = "入力された値はゼロです";
  }else{
```

```
    ele[0].innerHTML = "入力された値はゼロ以外です";
  }
  // 入力された数値に応じて処理を分ける
  switch(num){
    case 0 : ele[1].innerHTML = "入力値は0です";
      break;
    case 1 : ele[1].innerHTML = "入力値は1です";
      break;
    case 2 : ele[1].innerHTML = "入力値は2です";
      break;
    case 3 : ele[1].innerHTML = "入力値は3です";
      break;
    default : ele[1].innerHTML = "入力値は0～3以外です";
  }
  // 三項演算子を使って処理を分ける
  ele[2].innerHTML = (num === 0) ? "入力値は0です。" : "入力値は0以外です。"
}, false);
```

ブラウザでindex.htmlを表示すると、ダイアログが表示されます。ダイアログに「0」を入力して「OK」ボタンをクリックすると、次のように表示されます。



「0」以外の数値を入力すると、次のように表示されます。



COLUMN **配列/ハッシュを利用する**

サンプルのように条件に応じて文字を表示したり、特定の関数やメソッドを呼び出したりするのであれば、配列/ハッシュを利用することもできます。次の例では入力された数値に応じて配列にメッセージを入れておき、入力値を配列の添え字にすることでメッセージを表示しています。なお、配列にメッセージが入っていない範囲の場合は、次のように「||」を使って、別途、メッセージを表示することができます。

```
var ele = document.getElementsByTagName("output");
// ダイアログで数値を入力
var num = prompt("0〜9までの数値を入れてください", 1) | 0;
// 配列に表示文字を代入
var msg = ["0です", "1です"];
// 0かどうか調べる
ele[0].innerHTML = msg[num] || "0と1以外です";
```

関数の定義

関数を定義する

JavaScriptで関数を定義するには、いくつかの方法があります。通常の関数定義は、次のようになります。

```
function 名前(パラメータ){
  処理
}
```

パラメータは「,」(カンマ)で区切っていくつでも記述することができます。渡すパラメータの種類に制限はありません。

JavaScriptでは変数に関数を代入することができるため、次のように定義することもできます。ただし、通常の関数定義では処理が実行される前に関数定義が行われるので、プログラム中のどこでも関数を呼び出すことができますが、変数に代入する場合は代入後でないと関数を呼び出して使用することはできません。

```
var 名前 = function(パラメータ){ 処理 }
```

イベントハンドラやメソッド、および、一時的に処理を呼び出して使用する場合には、次のように名前のない関数(無名関数/匿名関数)を使うこともできます。

```
function(パラメータ){
処理
}
```

また、次のようにして定義した関数を即時実行させることもできます(即時関数)。

```
(function(パラメータ){
処理
})(パラメータ)
```

「function」文で定義する以外にも、「new Function()」を使って行うこともできます。「new Function()」では、パラメータも処理内容も文字列で指定します。複数のパラメータを指定することができますが、パラメータの最後が実際に処理する内容になります。「new Function()」は処理速度が遅いため、あまり使われません。

```
var 名前 = new Function(パラメータ,...,処理)
```

関数内でさらに関数を定義することもできます。この場合、関数外からは関数内に定義された関数を呼び出すことはできません。

サンプルでの動作確認

次のコードでは、定義した2つの関数が呼び出され、処理が行われます。処理が行われると、画面に文字が出力されます。

SAMPLE CODE HTMLのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <p><output></output></p>
    <p><output></output></p>
    <p><output></output></p>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  // 関数xyzを定義
  function xyz(){
    document.getElementsByTagName("output")[1].innerHTML = "xyz";
  }
  outputText("Sample");  // 関数outputTextを呼び出す
  xyz();  // 関数xyzを呼び出す
  var n = add(200, 15);  // 関数addを呼び出す
  document.getElementsByTagName("output")[2].innerHTML = "結果:"+n;
}, false);
// 関数outputTextを定義
function outputText(text){
  document.getElementsByTagName("output")[0].innerHTML = text;
}
// 関数addを定義
var add = new Function("a", "b", "return a+b");
```

ブラウザでindex.htmlを表示すると、次のように表示されます。

関連項目 ▶▶▶

● 関数のパラメータの読み出しについて …… p.64

SECTION-011

関数のパラメータの読み出しについて

関数のパラメータを読み出す方法

JavaScriptの関数に渡すパラメータの数は固定されておらず、自由な数だけ指定することができます。渡すパラメータの種類にも制限がありません。関数側では、渡されたパラメータは「arguments」配列を使って参照することができます。「arguments.length」が渡されたパラメータの総数になります。最初に渡されたパラメータは、「arguments[0]」で参照することができます。

サンプルでの動作の確認

次のコードでは、関数に渡された数値を合計した結果が表示されます。関数を2回、呼び出しているので、2つの計算結果が表示されます。

SAMPLE CODE HTMLのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <p><output></output></p>
    <p><output></output></p>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  // 関数calcを呼び出し
  var n = calc(200, 15);
  document.getElementsByTagName("output")[0].innerHTML = "合計 :"+n;
  // パラメータの数を変更して呼び出し
  n = calc(1,2,3,4,5,6,7,8,9,10);
  document.getElementsByTagName("output")[1].innerHTML = "合計 :"+n;
}, false);
// パラメータの数だけ処理する
function calc(){
  var total = 0;
  for(var i=0; i<arguments.length; i++){
    var n = arguments[i];
    total = total + n;
```

```
  }
  return total;
}
```

ブラウザでindex.htmlを表示すると、次のように表示されます。

関連項目 ▶▶▶

● 関数の定義 …… p.62

SECTION-012

「this」キーワードの固定について

「this」キーワードを固定する方法

JavaScriptでは「this」キーワードは固定されておらず、状況によって何を示すかが変わります。この「this」キーワードを決める（バインド）するには、「bind()」メソッドを使います。「bind()」メソッドのパラメータに、「this」キーワードでアクセスしたいオブジェクトを指定します。

サンプルでの動作確認

次のコードでは、関数内で「this」の内容に応じて加算が行われ、その結果が表示されます。

SAMPLE CODE HTMLのコード（index.html）

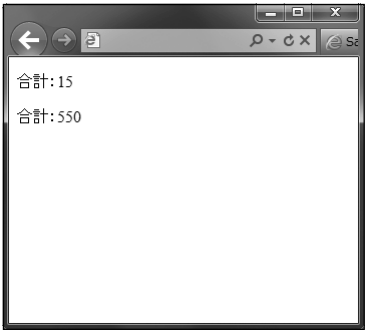
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <p><output></output></p>
    <p><output></output></p>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード（sample.js）

```
window.addEventListener("load", function(){
  // thisにバインド（紐付ける）配列その１
  var ary1 = [1, 2, 3, 4, 5];
  // thisにバインド（紐付ける）配列その２
  var ary2 = [10,20,30,40,50,60,70,80,90,100];
  // 関数calcにary1をバインド
  var myCalc1 = calc.bind(ary1);
  var n = myCalc1();
  document.getElementsByTagName("output")[0].innerHTML = "合計 :"+n;
  // 関数calcにary2をバインド
  var myCalc2 = calc.bind(ary2);
  n = myCalc2();
  document.getElementsByTagName("output")[1].innerHTML = "合計 :"+n;
}, false);
// 数値の合計を計算。bindしない場合はthisはwindowになる
function calc(){
  var total = 0;
  for(var i=0; i<this.length; i++){
```

```
    total = total + this[i];
  }
  return total;
}
```

ブラウザでindex.htmlを表示すると、次のように表示されます。



例外処理について

JavaScriptでの例外処理

JavaScritpで例外処理を行うには、「try...catch」文を使います。構文は、次のようになります。「finally」ブロックについては、省略可能です。

```
try{
  エラーが発生する可能性がある処理
} catch(e) {
  エラーが発生した場合の処理
} finally {
  エラーの有無に関係なく実行される処理
}
```

また、次のように、入れ子にして使うこともできます。

```
try {
  エラーが発生する可能性がある処理1
  try {
    エラーが発生する可能性がある処理2
  }catch(e){
    処理2でエラーが発生した場合の処理
  }
}catch(e){
  処理1でエラーが発生した場合の処理
}
```

エラーが発生すると、エラーの原因を示す文字列が「catch」のパラメータとして渡されます。発生するエラーの種類は、次の表のようになります。

エラーの種類	説明
EvalError	「eval()」実行時のエラー/式や文などが間違っている場合など原因多数
RangeError	値が範囲外/正数または整数のみ指定するところに負数や小数値を指定している場合など
ReferenceError	参照不能/変数名が間違っている、オブジェクトがない場合などに発生
SyntaxError	文法エラー/誤字脱字等、入力ミスの可能性あり
TypeError	型が間違っている/引数の型が違う、渡したパラメータが不足して値が「null」や「undefined」などになっている場合など
URIError	「encodeURI()」「decodeURI()」で不正な引数が渡された場合
DOMException	DOMで例外発生/対象ノードがないなど

また、「throw」文を使って故意にエラーを発生させることができます。この場合、「throw」文で指定した文字列などが「catch」のパラメータとして渡されます。

さらにエラーが発生すると、「catch」にはエラーオブジェクトが渡されます。エラーオブジェクトの「message」プロパティにエラーメッセージが入ります。これは、どのブラウザでも共通で

す。Firefoxでは、これに加えて「lineNumber」プロパティにエラー発生行が入ります。また、「fileName」プロパティには、エラーとなったファイルのURLが入ります。

サンプルでの動作確認

次のコードでは、例外が発生すると、例外の原因が表示されます。2番目のエラーは故意に発生させ、任意のメッセージを表示しています。

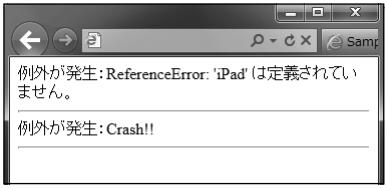
SAMPLE CODE HTMLのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <output></output><hr>
    <output></output><hr>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  var ele = document.getElementsByTagName("output");
  // エラーが発生する可能性がある処理を行う
  try{
    // iPadというオブジェクトは存在しないのでエラー
    var myDevice = new iPad();
  }catch(e){
    ele[0].innerHTML = "例外が発生 :"+e;
  }
  // throwを使ってエラーを投げる
  try{
    throw "Crash!!";
  }catch(e){
    ele[1].innerHTML = "例外が発生 :"+e;
  }
}, false);
```

ブラウザでindex.htmlを表示すると、次のように表示されます。



また、エラーオブジェクトを生成するサンプルは、次のようになります。

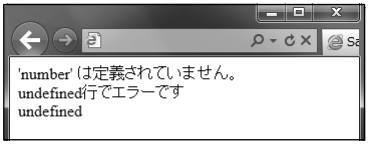
SAMPLE CODE HTMLのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <output></output>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  var ele = document.getElementsByTagName("output")[0];
  // エラーを発生させる
  try{
    // 入力ミスによるエラー
    var n = new number(12.3);
  }catch(e){
    ele.innerHTML = e.message+"<br>";
    ele.innerHTML += e.lineNumber+"行でエラーです<br>";
    ele.innerHTML += e.fileName;
  }
}, false);
```

ブラウザでindex.htmlを表示すると、エラーメッセージが表示されます。Firefox以外ではエラー行とエラーファイル名は「undefined」になります。



Firefoxではエラーメッセージに加えて、エラー発生行とエラーファイル名のURLが表示されます。

イベント処理について

JavaScriptのイベントの設定と解除

JavaScriptでは、「要素がクリックされた」「マウスが重なった」「ページが読み込まれた」など、さまざまなイベントがあります。主なイベントは、次の表のようになります。

イベント	説明
click	クリックした
dblclick	ダブルクリックした
mouseup	マウスの左ボタンが離された
mousedown	マウスの左ボタンが押された
mouseover	マウスが要素に重なった
mouseout	マウスが要素から離れた
keyup	キーが離された
keypress	キーが押されたまま
keydown	キーが押された
load	読み込みが完了した
scroll	スクロールした
touchstart	画面に指がタッチされた
touchmove	画面にタッチしたまま動かした
touchend	画面から指が離された

JavaScriptでイベントを設定するには、要素に用意されているプロパティにイベントハンドラ（イベント発生時に呼び出す関数）を設定します。イベントハンドラを設定できるプロパティ名には規則があり、イベント名の先頭に「on」を付加したものになります。たとえば、「click」イベントであれば、「onclick」プロパティにイベントハンドラを設定します。また、設定したイベントハンドラを解除するには、「onclick」プロパティに「null」を設定します。他のイベントも同様に設定・解除することができます。

イベントは、イベントリスナーを使って設定することもできます。イベントリスナーを使って設定する場合は、すでに設定されているイベントを上書きしません。多人数での作成や、ライブラリを作成・利用するのに便利です。イベントリスナーを設定するには、「addEventListener()」メソッドを使います。「addEventListener()」メソッドには、「イベント名」「イベントハンドラ」「イベントの伝達方向」の3つを指定します。「addEventListener()」メソッドで設定したイベントを解除するには、「removeEventListener()」メソッドを使います。ただし、「addEventListener()」メソッドで無名関数を使用した場合は、イベントを解除できないので注意が必要です。

なお、イベントについては、CHAPTER 09を参照してください。

サンプルでの動作の確認

次にコードでは、リスト項目がクリックされたら、イベントハンドラを呼び出します。最初のリスト項目は「onclick」プロパティにイベントハンドラを設定してあり、クリックするとアラートダイアログが表示されます。2番目のリスト項目は、イベントリスナーを使って2つのイベントハンドラを設定し

ています。このため、クリックすると、2回アラートダイアログが表示されます。

SAMPLE CODE HTMLのコード(index.html)

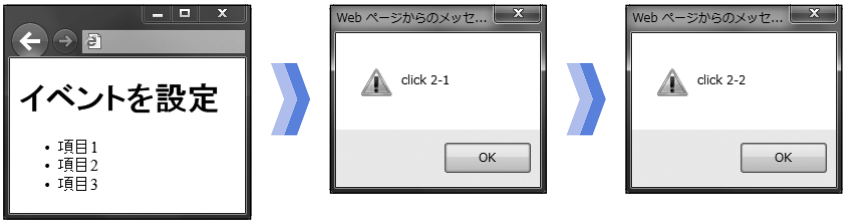
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <h1>イベントを設定</h1>
    <ul>
      <li>項目1</li>
      <li>項目2</li>
      <li>項目3</li>
    </ul>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.onload = function(){
  // イベントを設定する要素を読み出し
  var ele = document.getElementsByTagName("li");
  // 最初のli要素にイベントハンドラを設定
  ele[0].onclick = function(){
    alert("click 1");
  }
  // 2番目のli要素にイベントリスナーを設定
  ele[1].addEventListener("click", function(evt){
    alert("click 2-1");
  }, true);
  // 2番目のli要素に、さらにイベントリスナーを設定
  ele[1].addEventListener("click", function(evt){
    alert("click 2-2");
  }, true);
}
```

ブラウザでindex.htmlを表示し、最初の項目をクリックすると、アラートダイアログが1回、表示されます。

2番目の項目をクリックした場合、2回アラートダイアログが表示されます。



COLUMN IE6～8でのイベント設定

IE6～8では、「addEventListener()」「removeEventListener()」メソッドがありません。代わりに「attachEvent()」「detachEvent()」メソッドを使います。「attachEvent()」メソッド、「detachEvent()」メソッドともパラメータは同じで、最初にイベント名、2番目にイベントハンドラを指定します。イベント名は、「onclick」のように、先頭に「on」を付ける必要があります。具体的には次のようなコードになります。

```
window.onload = function(){
  // イベントを設定する要素を読み出し
  var ele = document.getElementsByTagName("li");
  // 最初のli要素にイベントハンドラを設定
  ele[0].onclick = function(){
    alert("click 1");
  }
  // 2番目のli要素にイベントリスナーを設定
  ele[1].attachEvent("onclick", function(evt){
    alert("click 2-1");
  });
  // 2番目のli要素に、さらにイベントリスナーを設定
  ele[1].attachEvent("onclick", function(evt){
    alert("click 2-2");
  });
}
```

なお、「addEventListener()」メソッドの場合は登録した順番にイベントハンドラが呼び出されますが、「attachEvent()」イベントの場合は最後に登録したイベントハンドラから呼び出されます（「addEventListener()」メソッドとは逆の順番になる）。

SECTION-015

ドキュメントオブジェクトモデルについて

JavaScriptでのHTML文書/XML文書の扱い

JavaScriptでは、ブラウザ上に表示されているHTMLの要素やXMLデータ内にある要素に対してアクセスし、処理する方法が用意されています。Webブラウザでは、HTML/XML文書はドキュメントオブジェクトモデル（DOM:Document Object Model）を採用しており、JavaScriptでもドキュメントオブジェクトモデルに従った文書を操作するためのメソッドやプロパティが用意されています。主なメソッドやプロパティは、次の表のようになります。

ドキュメントオブジェクトモデルでは、HTML/XML文書の階層を保ったまま処理を行います。このため、複雑な構造や深い階層の場合などでは、処理速度が低下することがあります。

メソッド	説明
getElementById()	指定したID名の要素にアクセスする
getElementsByTagName()	指定した要素名を持つHTML/XML要素にアクセスする。結果は配列で返される
getElementsByName()	指定した名前を持つHTML/XML要素にアクセスする。結果は配列で返される
querySelector()	指定されたCSSセレクタに一致するHTML/XML要素にアクセスする
querySelectorAll()	指定されたCSSセレクタに一致するHTML/XML要素にアクセスする。結果は配列で返される
createElement()	指定された要素を生成する
createTextNode()	テキストノードを生成する
appendChild()	子ノードとして末尾に追加する

プロパティ	説明
firstChild	最初の子ノード
lastChild	最後の子ノード
nextSibling	次の兄弟要素
previousSibling	前の兄弟要素
nodeType	ノードの種類
nodeValue	ノードの値

なお、ドキュメントオブジェクトモデル（DOM）については、CHAPTER 08を参照してください。

サンプルでの動作確認

次のコードでは、最初のリスト項目に関する情報が表示されます。また、ノードを生成し、4番目のリスト項目として「li」要素を追加しています。

SAMPLE CODE HTML(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
```

```
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <h1>DOMの処理</h1>
    <ul>
      <li>項目1</li>
      <li>項目2</li>
      <li>項目3</li>
    </ul>
    <hr>
    <output id="result"></output>
  </body>
</html>
```

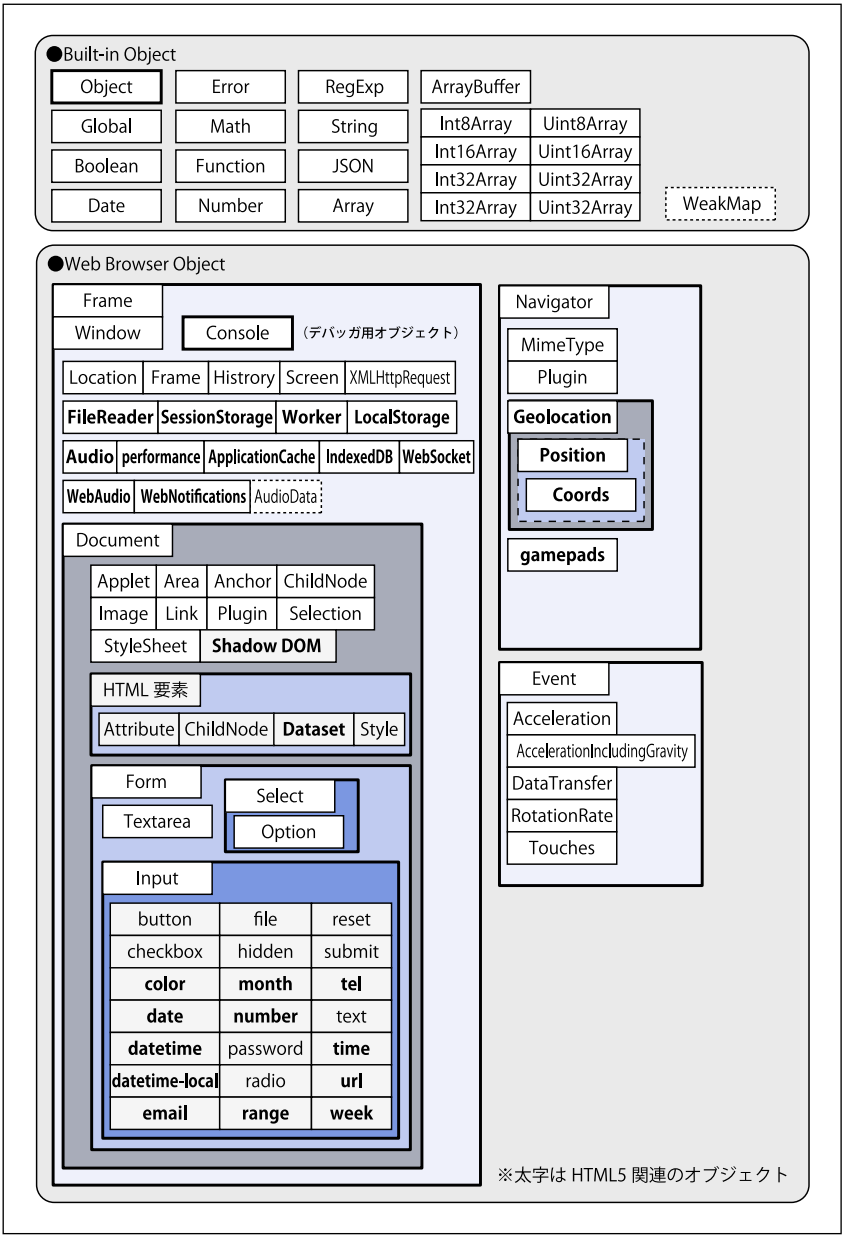
SAMPLE CODE JavaScriptのコード(sample.js)

```
window.onload = function(){
  var ele = document.getElementById("result");
  // 操作する要素を読み出す
  var list = document.getElementsByTagName("li");
  // 最初のli要素のnodeTypeやnodeValueなどを表示
  ele.innerHTML = "nodeType : "+list[0].nodeType+"<br>";
  ele.innerHTML += "nodeValue : "+list[0].firstChild.nodeValue+"<br>";
  ele.innerHTML += "tagName : "+list[0].tagName+"<br>";
  // テキストノードを作成
  var text = document.createTextNode("追加項目");
  // li要素を生成し子ノードとしてテキストを追加
  var li = document.createElement("li");
  li.appendChild(text);
  // リストの最後にli要素を追加
  document.getElementsByTagName("ul")[0].appendChild(li);
}
```

ブラウザでindex.htmlを表示すると、リスト項目に関する情報が表示され、リスト項目が1つ追加されます。

DOMの階層図

DOMの階層図は、次のようになります。



CHAPTER 02

オブジェクト

SECTION-016

文字列やオブジェクトを評価する

ここでは、JavaScriptの文字列やオブジェクトを評価する方法について解説します。

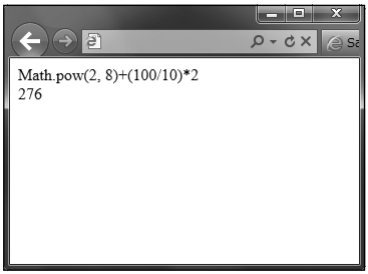
SAMPLE CODE HTMLのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <output></output>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  var ele = document.getElementsByTagName("output")[0];
  // 変数に評価する文字列を代入
  var text = "Math.pow(2, 8)+(100/10)*2";
  // eval()で評価
  var result = eval(text);
  // 結果を出力
  ele.innerHTML = text+"<br>"+result;
}, false);
```

ブラウザでindex.htmlを表示すると、評価する式と評価した結果が表示されます。



ONEPOINT　式や文の評価を行うには「eval()」メソッドを使う

JavaScriptでは、オブジェクトや文字列を「eval()」メソッドを使って評価することができます。これによって、文字列をスクリプトとして解釈し、実行させることができます。「eval()」メソッドに指定できるのは評価可能なものであれば式でも文でも構いません。非同期通信を使って読み込んだJSON形式をJavaScriptのオブジェクトにする際に利用される場合もあります。ただし、この場合、不正なコードが含まれていると思わぬ不具合を発生させることがあります。JSONデータを変換する場合には、「JSON.parse()」メソッドが使えるのであれば、これを使用する方がよいでしょう。

COLUMN　ビルトインオブジェクト

ECMA-262 5th(ECMA Script)では、次の表のビルトインオブジェクトが定義されています。ブラウザではビルトインオブジェクトに加えて、「window」や「document」など、非常に多くのオブジェクトがあります。

ビルトインオブジェクト	説明
Global Object	グローバルオブジェクト
Object	オブジェクト
Function	関数
Array	配列
String	文字列
Boolean	真偽値
Number	数値
Math	各種計算
Date	日付
RegExp	正規表現
Error	エラー
JSON	JSON(ECMA-262 5thより追加)

SECTION-017

数値かどうか調べる

ここでは、数値かどうか調べる方法について解説します。

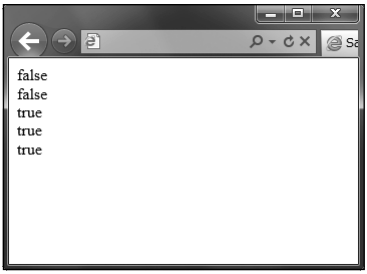
SAMPLE CODE HTMLのコードのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <output></output>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  var ele = document.getElementsByTagName("output")[0];
  // 変数に小数値を文字列として代入
  var data1 = "1969.215";
  // 変数に数値を代入
  var data2 = 2035.92;
  // 変数に文字列を変換
  var data3 = "JavaScript";
  // data1が数値以外かどうか調べる
  var flag1 = isNaN(data1);
  // data2が数値以外かどうか調べる
  var flag2 = isNaN(data2);
  // data3が数値以外かどうか調べる
  var flag3 = isNaN(data3);
  // windowオブジェクトが数値以外かどうか調べる
  var flag4 = isNaN(window);
  // 未定義のプロパティを調べる
  var flag5 = isNaN(window.evol);
  // 結果を出力
  ele.innerHTML = flag1+"<br>"+flag2+"<br>"+flag3+"<br>"+flag4+"<br>"+flag5;
}, false);
```

ブラウザでindex.htmlを表示すると、変数やオブジェクトが数値以外かどうか調べた結果が表示されます。



ONEPOINT　数値かどうかを調べるには「isNaN()」メソッドを使う

JavaScriptでは、数値かどうかを調べるには、「isNaN()」メソッドを使います。パラメータに指定された内容が数値に変換することができなかった場合は「true」を、数値に変換できた場合は「false」を返します。数字で表現される文字列の場合、数値に変換できるのであれば、サンプルのように「false」となります。

COLUMN　整数値かどうか調べる

「0」～「9」までの数値だけで構成されているか、つまり、「+」記号がない整数値かどうか調べるには、次のように正規表現を使います。

```
var ele = document.getElementsByTagName("output")[0];
// 変数に数値を文字列として代入
var numStr = "1969215";
// 正規表現を使って調べる
var data = numStr.match(/\D/g);
// 結果を出力
if (data === null){
  ele.innerHTML = "0～9の数値です";
}else{
  ele.innerHTML = "0～9の数値ではありません";
}
```

SECTION-018

文字列をエスケープ化/エンコード/デコードする

ここでは、文字列をエスケープ化/エンコード/デコードする方法について解説します。

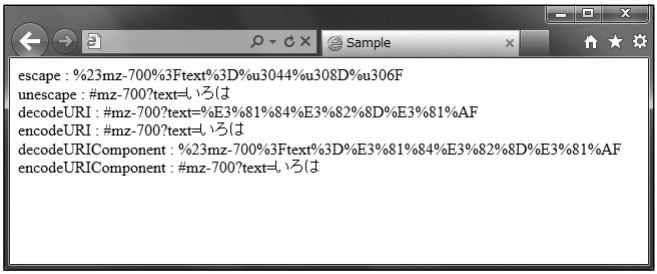
SAMPLE CODE HTMLのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <output></output>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  var ele = document.getElementsByTagName("output")[0];
  // 変数に文字を代入
  var text = "#mz-700?text=いろは";
  // エスケープ文字に変換
  var str1 = escape(text);
  // エスケープされた文字から変換
  var str2 = unescape(str1);
  // 文字をエンコードする
  var str3 = encodeURI(text);
  // エンコードされた文字をデコードする
  var str4 = decodeURI(str3);
  // 文字をエンコードする
  var str5 = encodeURIComponent(text);
  // エンコードされた文字をデコードする
  var str6 = decodeURIComponent(str5);
  // 結果を出力
  ele.innerHTML = "escape : "+str1+"<br>unescape : "+str2+"<br>";
  ele.innerHTML += "decodeURI : "+str3+"<br>encodeURI : "+str4+"<br>";
  ele.innerHTML += "decodeURIComponent : "+str5+"<br>encodeURIComponent : "+str6;
}, false);
```

ブラウザでindex.htmlを表示すると、文字をそれぞれ変換処理した結果が表示されます。



ONEPOINT 文字をエスケープ化するには「escape()」「encodeURI()」「encodeURIComponent()」関数を使う

サーバーにデータを送信する際に記号などは変換する必要があります。JavaScriptでは、「escape()」「encodeURI()」「encodeURIComponent()」の3種類の関数で文字をエンコード(符号化)できます。これらの関数でエンコードした文字列を元に戻す(デコード/復号化)するには、「unescape()」「decodeURI()」「decodeURIComponent()」関数を使います。それぞれの関数でどの文字が変換対象になるかは、次のようになります。なお、左上はスペースを表しています。

◉「escape()」で変換される文字

	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

◉「encodeURI()」で変換される文字

	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

◉「encodeURIComponent()」で変換される文字

	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

%xx形式に変換される文字

変換されない文字

SECTION-019

数値に変換する/数値の単位を削除する

ここでは、文字列で表現されている文字を数値に変換する、数値の単位を削除する方法について解説します。

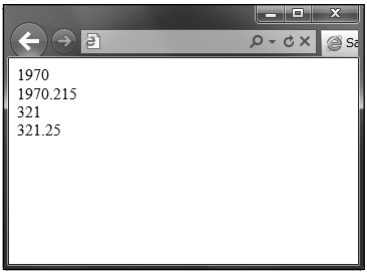
SAMPLE CODE HTMLのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <output></output>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  var ele = document.getElementsByTagName("output")[0];
  // 変数に小数値を文字列として代入
  var num1 = "1969.215";
  // 変数に単位付きの値を文字型として代入
  var num2 = "320.25px";
  // num1を整数値に変換
  var n1 = parseInt(num1);
  n1 = n1 + 1;  // 1を加算
  // num1を小数値に変換
  var n2 = parseFloat(num1);
  n2 = n2 + 1;  // 1を加算
  // num2を整数値に変換
  var n3= parseInt(num2);
  n3 = n3 + 1;
  // num2を小数値に変換
  var n4= parseFloat(num2);
  n4 = n4 + 1;
  // 結果を出力
  ele.innerHTML = n1+"<br>"+n2+"<br>"+n3+"<br>"+n4;
}, false);
```

ブラウザでindex.htmlを表示すると、文字で表現されている値を数値に変換し、演算した結果が表示されます。



ONEPOINT　文字を数値に変換するには「parseInt()」メソッドや「parseFloat()」メソッドを使う

JavaScriptでは変数に代入されている内容が数字を示していても、それが文字型の場合には通常の算術演算が行われません。たとえば、変数「n」に「"12"」という文字型で表現されている数値がある場合、「+」記号を使って「3」を加算すると、「15」ではなく「123」となってしまいます。これは文字列同士の連結になってしまっているためです。このような場合、文字型の数値を数値型に変換する必要があります。文字型になっている数値を数値型に変換するメソッドとして、「parseInt()」メソッドと「parseFloat()」メソッドがあります。「parseInt()」メソッドは整数値に、「parseFloat()」メソッドは小数値までを含めて変換した結果を数値型として返します。

また、CSSに関するプロパティを読み出した際、「"320px"」のように単位が付随している場合があります。座標値を読み出して演算したい場合には、この単位が邪魔になります。そこで「parseInt()」メソッドや「parseFloat()」メソッドを使えば、この単位部分だけを削除することができます。

なお、文字を数値に変換するには、「"1234" - 0;」のように、「0」を減算する方法もあります。

COLUMN　「eval()」メソッドで文字を数値に変換する

文字を数値に変換するには、「parseInt()」メソッドや「parseFloat()」メソッドを利用する以外でも、式評価を行う「eval()」メソッドを利用する方法もあります。「eval("12.3")」のようにパラメータを指定すると式の評価（演算）が行われ、「12.3」という数値になります。また、「eval("12.3+2*Math.PI")」のように計算式を指定することもできます。「eval()」メソッドは、JavaScriptで処理できるものであれば、計算式でもメソッドでもプロパティでも評価・実行が行われます。ただし、「eval()」メソッドは処理速度が速くないので高速に処理する場合は使用を避けた方がよいでしょう。また、パラメータによっては、セキュリティホールの原因になることもあります。

SECTION-020

JavaScriptのデータをJSON形式に変換する

ここでは、JavaScriptのデータをJSON形式に変換する方法について解説します。

SAMPLE CODE HTMLのコード(index.html)

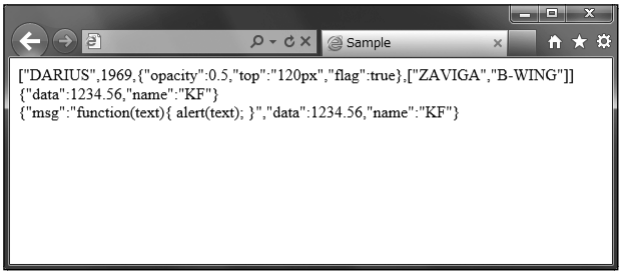
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <output></output>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  var ele = document.getElementsByTagName("output")[0];
  // JSON形式に変換する例として配列を作成
  var myArray = ["DARIUS", 1969,
    {  opacity: 0.5,
      top: "120px",
      flag : true
    },
    ["ZAVIGA", "B-WING"]
  ];
  // JSON形式に変換
  var jsondata = JSON.stringify(myArray);
  // 結果を出力
  ele.innerHTML = jsondata+"<br>";
  // オブジェクト／メソッドを含む
  var myObject = {
    msg : function(text){ alert(text); },
    data : 1234.56,
    name : "KF"
  };
  // JSON形式に変換
  var jsondata2 = JSON.stringify(myObject);
  // 結果を出力
  ele.innerHTML += jsondata2+"<br>";
  // 自前で独自形式に変換
  var jsondata3 = JSON.stringify(myObject, function(key, value){
```

```
    if (new String(value).indexOf("function") > -1) {
      return new String(value);
    }
    return value;
  });
  // 結果を出力
  ele.innerHTML += jsondata3+"<br>";
}, false);
```

ブラウザでindex.htmlを表示すると。JSON形式に変換した結果が表示されます。



ONEPOINT　JavaScriptのデータをJSON形式に変換するには「JSON.stringify()」メソッドを使う

JSONは、JavaScriptで扱える軽量なデータフォーマットです。JavaScriptのデータをJSON形式に変換するには、「JSON.stringify()」メソッドを使います。関数やオブジェクト、メソッドが含まれる場合、期待通りにJSON形式に変換されない場合があります。このような場合、「JSON.stringify()」メソッドの2番目のパラメータに関数を指定することで独自に変換処理することができます。この関数には「key」と「value」が渡されるので、それらに応じて処理することでサンプルのようにメソッドも文字列に変換することができます。ただし、オブジェクトでなく、配列の場合は要素ごとに「key」と「value」が渡されず、一括して関数に渡されます。

関連項目 ▶▶▶
● JSON形式からオブジェクトに変換する …… p.88

SECTION-021

JSON形式からオブジェクトに変換する

ここでは、JSON形式からJavaScriptのオブジェクトに変換する方法について解説します。

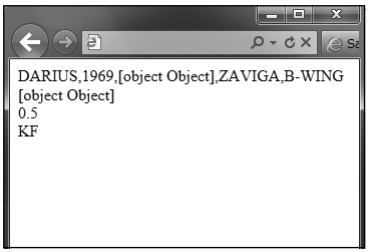
SAMPLE CODE HTMLのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <output></output>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  var ele = document.getElementsByTagName("output")[0];
  // JSONデータを変数に代入(非同期通信で読み込む方式でも可)
  var data1 =
  '["DARIUS",1969,{"opacity":0.5,"top":"120px","flag":true},["ZAVIGA","B-WING"]]';
  var data2 = '{"data":1234.56,"name":"KF"}';
  // JSON形式から変換
  var obj1 = JSON.parse(data1);
  var obj2 = JSON.parse(data2);
  // 結果を出力
  ele.innerHTML = obj1+"<br>"+obj2+"<br>";
  // 内容を読み出して出力
  ele.innerHTML += obj1[2].opacity+"<br>";
  ele.innerHTML += obj2.name;
}, false);
```

ブラウザでindex.htmlを表示すると。JSON形式から変換して読み出した内容などが表示されます。



ONEPOINT　JSON形式からオブジェクトにするには「JSON.parse()」メソッドを使う

非同期通信などを使ってサーバーから読み込まれたJSON形式をJavaScriptのオブジェクト形式に変換するには、「JSON.parse()」メソッドを使います。「eval()」メソッドでも変換することはできますが、「JSON.parse()」メソッドの方が、より安全に変換することができます。

また、jQueryライブラリなどでは、あらかじめJSON形式かどうか調べてから処理を行うようになっているものもあります。

COLUMN　Local StorageとJSON

データをローカルディスク上に保存する場合に多く利用されるのが、Local Storageです。このLocal Storageにデータを保存する場合、文字列で保存する必要があります。配列や各種オブジェクトは、そのままでは保存できません。このため、配列オブジェクトなどは、「JSON.stringify()」メソッドを使って保存する必要があります。「JSON.stringify()」メソッドを使って保存した配列オブジェクトなどは、「JSON.parse()」メソッドを使って元に戻すことができます。なお、「window」オブジェクトなどは、「JSON.stringify()」メソッドを使って変換することはできません。また、「RegExp」オブジェクトも「JSON.stringify()」メソッドを使うと、空のオブジェクトである「{}」に変換されます。

関連項目 ▶▶▶

- JavaScriptのデータをJSON形式に変換する …… p.86
- Web Storageにデータを設定する …… p.738

SECTION-022

オブジェクトを生成する

ここでは、オブジェクトを生成する方法について解説します。

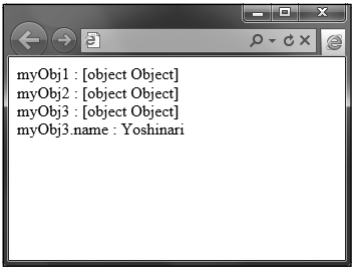
SAMPLE CODE HTMLのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <output></output>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  var ele = document.getElementsByTagName("output")[0];
  // オブジェクトを生成
  var myObj1 = new Object();
  // オブジェクトを作成
  var myObj2 = { };
  // オブジェクトを作成
  var myObj3 = { name : "Yoshinari" };
  // 結果を出力
  ele.innerHTML = "myObj1 : "+myObj1.toString();
  ele.innerHTML += "<br>myObj2 : "+myObj2.toString();
  ele.innerHTML += "<br>myObj3 : "+myObj3.toString();
  ele.innerHTML += "<br>myObj3.name : "+myObj3.name;
}, false);
```

ブラウザでindex.htmlを表示すると、3つオブジェクトを生成した結果が表示されます。最終行ではオブジェクトのプロパティの内容が表示されています。



ONEPOINT　オブジェクトを生成するには「new Object()」とするか「{ }」を使う

JavaScriptでオブジェクトを生成するには、「new Object()」とするか、「{ }」を使います。「{ }」内には「キー:値」のペアを「,」(カンマ)で区切って複数、列記することができます。この場合、キー名がプロパティ、または、メソッド名になります。

独自のオブジェクトではなく、既存のオブジェクト(StringやArray)を生成したい場合は、「new」演算子を使います。

COLUMN　オブジェクトの生成速度

JavaScriptでは、「new Object()」としてオブジェクトを生成する以外に「{ }」と記述して生成することもできます。どちらもオブジェクトが生成されることには変わりませんが、処理にかかる時間は大幅に異なります。「new Object()」とした場合と比べて「{ }」を使うと、2倍以上、高速にオブジェクトを生成することができます。

関連項目 ▶▶▶

● 指定されたオブジェクトを継承して新たなオブジェクトを生成する…… p.92

SECTION-023

指定されたオブジェクトを継承して新たなオブジェクトを生成する

ここでは、指定されたオブジェクトを継承し新たなオブジェクトを生成する方法について解説します。

SAMPLE CODE HTMLのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <output></output>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  // オブジェクトを生成
  var company = { name : "C&R研究所", address : "新潟県" };
  // オブジェクトを継承して新たなオブジェクトを生成
  var myCompany1 = Object.create(company);
  // オプションを指定して生成
  var myCompany2 = Object.create(company, {
    owner : {
        value : "Yoshinari",  // プロパティの内容
        writable : false,  // 書き込み可能かどうか
        enumerable : true,  // 列挙可能。for...inで出る
        configurable : false  // 再定義可能かどうか
     },
    stype : {
        value : "男性",  // プロパティの内容
        writable : false,  // 書き込み可能かどうか
        enumerable : false,  // 列挙不可。for...inで出ない
        configurable : true  // 再定義可能かどうか
     }
  });
  // 結果を出力
  outObj(company, "company");
  outObj(myCompany1, "myCompany1");
  outObj(myCompany2, "myCompany2");
```

```
// オブジェクトの内容を出力する関数
function outObj(obj, text){
  var ele = document.getElementsByTagName("output")[0];
  ele.innerHTML += "■"+text+"の内容<br>";
  for(var i in obj){
    ele.innerHTML += i + " = " + obj[i] + "<br>";
  }
  ele.innerHTML += "<hr>";
}
}, false);
```

ブラウザでindex.htmlを表示すると、生成したオブジェクトのプロパティの内容が表示されます。

ONEPOINT　オブジェクトを継承し新たなオブジェクトを生成するには「Object.create()」メソッドを使う

JavaScriptでオブジェクトを継承して、新たなオブジェクトを生成するには、「Object.create()」メソッドを使います。最初のパラメータには継承元となるオブジェクトを指定します。2番目のパラメータはオプションで、プロパティに対して次の表の属性を設定することができます。なお、同時に設定する属性の設定値によってはエラーとなる場合があります。

属性値	説明
value	プロパティの内容
writable	プロパティに書き込みができるかどうか。「true」なら書き込み可能、「false」なら不可
enumerable	列挙可能かどうか。「for...in」文での抽出対象になるかどうか。「true」なら列挙可能、「false」なら列挙不可
configurable	再定義可能かどうか。「true」なら再定義可能、「false」なら再定義不可
set	セッター。プロパティに値を書き込む時に実行する関数
get	ゲッター。プロパティから内容を読み出す時に実行する関数

関連項目 ▶▶▶
●オブジェクトを生成する …… p.90

SECTION-024

オブジェクトのインスタンスを調べる

ここでは、オブジェクトのインスタンスを調べる方法について解説します。

SAMPLE CODE HTMLのコード(index.html)

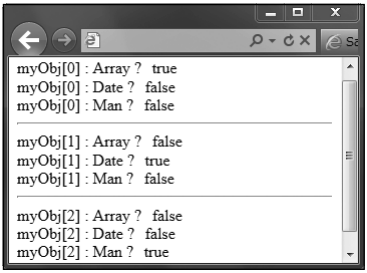
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <output></output>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  var ele = document.getElementsByTagName("output")[0];
  // 各種オブジェクトを格納する配列を用意
  var myObj = [];
  // 配列オブジェクトを作成
  myObj[0] = new Array();
  // Dateオブジェクトを作成
  myObj[1]= new Date();
  // manオブジェクトを作成
  myObj[2] = new Man("KF", 43);
  // 特定のオブジェクトのインスタンスか調べ結果を出力
  for(var i=0; i<myObj.length; i++){
    // 配列オブジェクトのインスタンスかどうか調べる
    var flag1 = myObj[i] instanceof Array;
    // Dateオブジェクトのインスタンスかどうか調べる
    var flag2 = myObj[i] instanceof Date;
    // 自前のオブジェクトのインスタンスかどうか調べる
    var flag3 = myObj[i] instanceof Man;
    // 結果を出力
    ele.innerHTML += "myObj["+i+"] : Array ?  "+flag1+"<br>";
    ele.innerHTML += "myObj["+i+"] : Date ?  "+flag2+"<br>";
    ele.innerHTML += "myObj["+i+"] : Man ?  "+flag3+"<hr>";
  }
  // 自前のオブジェクトを生成するためのクラス
  function Man(name,age){
    this.name = name;
```

```
    this.age = age;
  }
}, false);
```

ブラウザでindex.htmlを表示すると、オブジェクトがどのインスタンスなのかを調べた結果が表示されます。



ONEPOINT　どのオブジェクトから派生したのか調べるには「instanceof」演算子を使う

JavaScriptでは、多くをオブジェクトとして扱います。あらかじめ決められたオブジェクトから生成する以外に、独自に作成したオブジェクトを生成することもできます。生成されたオブジェクトが、どのような種類のオブジェクトなのかを調べるには、「instanceof」演算子を使います。「typeof」演算子では型が返るだけで、正確な情報はわかりません。また、「instanceof」演算子以外には、オブジェクトの「constructor」プロパティを使って生成元のオブジェクト(内容)を確認することもできます。

関連項目 ▶▶▶

- 指定されたオブジェクトを継承して新たなオブジェクトを生成する…… p.92
- プロパティのオーナーかどうか調べる …… p.114

SECTION-025

オブジェクトの内容を読み出す

ここでは、オブジェクトの内容を読み出す方法について解説します。

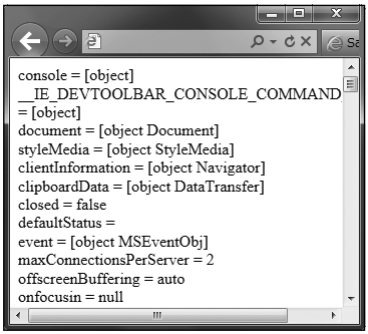
SAMPLE CODE HTMLのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <output></output>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  // 結果を入れる変数を用意
  var data = "";
  // windowオブジェクトの内容を出力
  for(var i in window){
    try{
      data = data + i + " = " + window[i] + "<br>";
    }catch(e){}
  }
  // 結果を出力
  document.getElementsByTagName("output")[0].innerHTML = data;
}, false);
```

ブラウザでindex.htmlを表示すると、「window」オブジェクトの内容が表示されます。



ONEPOINT　オブジェクトの内容を読み出すには「for...in」文を使う

オブジェクトの内容を読み出すには、「for...in」文を使います。「for（変数名 in オブジェクト）」とすると、オブジェクトのプロパティ名が1つずつ読み出されます。「for...in」文では、オブジェクトで「DontEnum」属性（非列挙型）になっているものは読み出すことができません。「DontEnum」属性は、「Object.create()」メソッドによってオブジェクト生成するときや、「defineProperty()」メソッドや「defineProperties()」メソッドを使って後から設定することができます。これらのメソッドなどでは、次の表に示すオブジェクトの属性値を設定することができます。オブジェクトのプロパティが列挙型かどうかは、「enumerable」属性で指定することができます。

また、ブラウザによってはオブジェクト内容を読み出すとエラーになる場合があります。これは、「try...catch」文で回避することができます。

オブジェクトの属性値	説明
writable	プロパティに書き込みができるかどうか。「true」なら書き込み可能、「false」なら不可
enumerable	列挙可能かどうか。「for...in」文での抽出対象になるかどうか。「true」なら列挙可能、「false」なら列挙不可
configurable	再定義可能かどうか。「true」なら再定義可能、「false」なら再定義不可

COLUMN　「in」演算子を使ってプロパティがあるか調べる

「in」演算子を使うと、オブジェクトにプロパティがあるかどうか調べることができます。調べることができるのはプロパティとオブジェクトで、メソッドは対象外になります。次の例では、「navigator」オブジェクトに「geolocation」が含まれる場合は「true」、そうでない場合は「false」が変数「flag」に入ります。

```
var flag = "geolocation" in navigator;
```

なお、「in」演算子は、プロトタイプチェーンをたどってプロパティがあるかどうかを調べます。プロパティがそのオブジェクトだけに存在するものかどうか（オーナーかどうか）を調べるには、「hasOwnProperty()」メソッドを使います。

関連項目 ▶▶▶

- オブジェクトのプロパティの読み出し・書き込み処理を設定する …… p.102
- オブジェクトのプロパティ一覧を取得する …… p.110
- プロパティのオーナーかどうか調べる …… p.114

SECTION-026

プロトタイプを利用する

ここでは、プロトタイプを利用する方法について解説します。

SAMPLE CODE HTMLのコード(index.html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sample</title>
    <script src="js/sample.js"></script>
  </head>
  <body>
    <h1>指定した桁で区切る</h1>
    <output></output>
  </body>
</html>
```

SAMPLE CODE JavaScriptのコード(sample.js)

```
window.addEventListener("load", function(){
  var ele = document.getElementsByTagName("output")[0];
  // Numberオブジェクトにn桁で区切るメソッドを追加
  Number.prototype.numFormat = function(separate){
    // パラメータが省略された場合は3桁ごとにする
    separate = separate || 3;
    // 結果を入れる
    var n = "";
    var count = 0;
    // 数値を文字列に変換
    var str = new String(this);
    for (var i=str.length-1; i>=0; i--){
      n = str.charAt(i) + n;
      count++;
      if (((count % separate) == 0) && (i != 0)) n = ","+n;
    }
    return n;
  }
  // 数値を文字列として3桁で区切る
  var n = (12345678).numFormat();
  ele.innerHTML = n+"<br>";
  // 数値を文字列として4桁で区切る
  var n = (12345678).numFormat(4);
  ele.innerHTML += n;
}, false);
```