

JavaScript 逆引きハンドブック

古簾 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

古籾 一浩 ◆ 著



■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・販売することは強く禁止します。

●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

III 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。

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 : Windows 7/Mac OS X 10.6/Mac OS X 10.7
- Chrome for Android : Android 4.0
- Firefox 14 : Windows 7/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 : Android 2.3/3.2/4.0
- 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。



III 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。

- ① <http://www.c-r.com/>にアクセスします。
- ② 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。
- ③ 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。
- ④ 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。
- ⑤ 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。
- ⑥ 「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。

III 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)。

CHAPTER 01 JavaScriptの基礎知識と基本文法

001	JavaScriptの概要	38
002	JavaScriptの基本事項	40
003	演算子について	42
	COLUMN ■ 予約語について	
004	リテラルについて	45
005	strictモードとバージョンの指定について	46
006	コメントについて	49
007	変数と定数について	51
	COLUMN ■ constでの不具合	
008	繰り返し処理(ループ)について	54
	COLUMN ■ DOMを繰り返し操作する場合	
009	条件分岐について	58
	COLUMN ■ 配列/ハッシュを利用する	
010	関数の定義	62
011	関数のパラメータの読み出しについて	64
012	「this」キーワードの固定について	66
013	例外処理について	68
014	イベント処理について	71
	COLUMN ■ IE6 ~ 8でのイベント設定	
015	ドキュメントオブジェクトモデルについて	74

CHAPTER 02 オブジェクト

016	文字列やオブジェクトを評価する	78
	ONEPOINT ■ 式や文の評価を行うには「eval()」メソッドを使う	
	COLUMN ■ ビルトインオブジェクト	
017	数値かどうか調べる	80
	ONEPOINT ■ 数値かどうか調べるには「isNaN()」メソッドを使う	
	COLUMN ■ 整数値かどうか調べる	
018	文字列をエスケープ化/エンコード/デコードする	82
	ONEPOINT ■ 文字をエスケープ化するには「escape()」「encodeURIComponent()」	
	「encodeURIComponent()」関数を使う	
019	数値に変換する/数値の単位を削除する	84
	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	
	COLUMN ■ 「eval()」メソッドで文字を数値に変換する	

□ 2 0	JavaScriptのデータをJSON形式に変換する	86
	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	
□ 2 1	JSON形式からオブジェクトに変換する	88
	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	
	COLUMN ■ Local StorageとJSON	
□ 2 2	オブジェクトを生成する	90
	ONEPOINT ■ オブジェクトを生成するには「new Object()」とするか「{}」を使う	
	COLUMN ■ オブジェクトの生成速度	
□ 2 3	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	92
	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	
□ 2 4	オブジェクトのインスタンスを調べる	94
	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	
□ 2 5	オブジェクトの内容を読み出す	96
	ONEPOINT ■ オブジェクトの内容を読み出すには「for...in」文を使う	
	COLUMN ■ 「in」演算子を使ってプロパティがあるか調べる	
□ 2 6	プロトタイプを利用する	98
	ONEPOINT ■ プロトタイプを利用するには「prototype」を使う	
□ 2 7	プロパティの属性値を設定・取得する	100
	ONEPOINT ■ プロパティの属性値を設定するには「オブジェクト名.プロパティ名 = 内容」、取得するには「オブジェクト名.プロパティ名」とする	
□ 2 8	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	102
	ONEPOINT ■ 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には「Object.create()」「Object.defineProperty()」メソッドなどを使う	
□ 2 9	オブジェクトの変更を禁止する	104
	ONEPOINT ■ オブジェクトを保護するには「freeze()」メソッドを使う	
□ 3 0	プロパティの追加を禁止する	106
	ONEPOINT ■ プロパティの追加を禁止するには「preventExtensions()」メソッドを使う	
□ 3 1	オブジェクトの変更・拡張などが可能か調べる	108
	ONEPOINT ■ オブジェクトが拡張可能か確認するには「isExtensible」プロパティを使う	
□ 3 2	オブジェクトのプロパティ一覧を取得する	110
	ONEPOINT ■ オブジェクトのプロパティ一覧を取得するには「keys()」メソッドと「getOwnPropertyNames()」メソッドを使う	
□ 3 3	オブジェクトの種類を調べる	112
	ONEPOINT ■ オブジェクトの種類を調べるには「typeof」演算子を使う	
□ 3 4	プロパティのオーナーかどうか調べる	114
	ONEPOINT ■ プロパティのオーナーかどうか調べるには「hasOwnProperty()」メソッドを使う	

□ 35 「Boolean」オブジェクトを生成する	116
ONEPOINT ■「Boolean」オブジェクトを生成するには「new Boolean()」とする	
□ 36 プロパティやオブジェクトを削除する	117
ONEPOINT ■オブジェクトを削除するには「null」を代入、 プロパティを削除するには「delete」演算子を使う	

CHAPTER 03 配列

□ 37 配列オブジェクトを生成する	120
ONEPOINT ■配列を生成するには「new Array()」や「[]」を使う	
□ 38 指定したビット長の配列を生成する	122
ONEPOINT ■ビット長を指定して配列要素を生成するには「Typed Array」を使う	
□ 39 配列同士や配列要素を連結する	124
ONEPOINT ■配列同士を連結するには「concat()」メソッド、 配列要素を連結するには「join()」メソッドを使う	
COLUMN ■IE6で高速に文字列を連結する	
□ 40 配列要素を抜き出す	126
ONEPOINT ■配列要素を抜き出すには「slice()」メソッドを使う	
□ 41 配列要素を抽出・挿入する	128
ONEPOINT ■配列要素を抽出・挿入するには「splice()」メソッドを使う	
□ 42 条件を満たす配列要素を調べる	130
ONEPOINT ■配列要素が条件を満たしているか調べるには 「every()」メソッドや「some()」メソッドを使う	
□ 43 配列要素を順番に処理する	132
ONEPOINT ■配列要素を順番に処理するには「forEach()」メソッドを使う	
COLUMN ■重複している要素を排除するには	
□ 44 配列要素を検索する	134
ONEPOINT ■配列要素を検索するには 「indexOf()」メソッドや「lastIndexOf()」メソッドを使う	
□ 45 配列かどうか調べる	136
ONEPOINT ■配列かどうか調べるには 「instanceof」演算子や「Array.isArray()」メソッドを使う	
□ 46 配列要素の総数を求める	138
ONEPOINT ■配列の要素数を調べるには「length」プロパティを使う	
COLUMN ■連想配列も含めた要素数を求めるには	
□ 47 配列要素を加工する	140
ONEPOINT ■配列要素を加工するには「map()」メソッドを使う	
□ 48 配列要素を追加・削除する	142
ONEPOINT ■配列の末尾に要素を追加するには「push()」メソッド、 末尾の要素の削除を行うには「pop()」メソッドを使う	
COLUMN ■任意の位置に要素を追加するには	

049	配列要素を前方または後方から処理する	144
	ONEPOINT ■ 要素を先頭から処理するには「reduce()」メソッド、 末尾から処理するには「reduceRight()」メソッドを使う	
050	配列要素をソートする	146
	ONEPOINT ■ 要素をソートするには「sort()」メソッドを使う	
051	配列要素を逆順にする	148
	ONEPOINT ■ 配列要素の順序を逆順にするには「reverse()」メソッドを使う	
052	配列要素内容を前方/後方にずらす	149
	ONEPOINT ■ 配列内容を前方に移動させるには「shift()」メソッド、 後方に移動させるには「unshift()」メソッドを使う	
	COLUMN ■ 配列内容をローテーションさせる	
053	配列要素を文字列に変換する	151
	ONEPOINT ■ 配列要素を文字列に変換するには 「toString()」メソッドや「JSON.stringify()」メソッドを使う	
	COLUMN ■ 配列オブジェクトの生成時間	
054	「WeakMap」オブジェクトを生成する	153
	ONEPOINT ■ 「WeakMap」オブジェクトを生成するには「new WeakMap()」とする	
055	WeakMapで該当するキーがあるか調べる	155
	ONEPOINT ■ WeakMapでキーの有無を調べるには「has()」メソッドを使う	
056	WeakMapのキーを削除する	157
	ONEPOINT ■ キーの削除を行うには「delete()」メソッドを使う	

CHAPTER 04 数値・数学

057	「Number」オブジェクトを生成する	160
	ONEPOINT ■ 「Number」オブジェクトを生成するには「new Number()」とする	
	COLUMN ■ 虚数について	
058	数値の桁数を指定する	162
	ONEPOINT ■ 桁数を指定するには「toFixed()」メソッドを使う	
059	数値を小数点以下n桁にする	163
	ONEPOINT ■ 小数点以下の桁数を指定するには「toFixed()」メソッドを使う	
060	3桁ごとに区切る	164
	ONEPOINT ■ 3桁ごとに区切るには数値を文字列に変換して「,」を連結する	
061	数値を指定桁数にして不足分を先頭から「0」で埋める	166
	ONEPOINT ■ 不足分の桁を「0」で埋めるには「slice()」メソッドを使う	
062	小数部分と整数部分を分けて取り出す	168
	ONEPOINT ■ 小数部分と整数部分を分けるには文字型に変換した後で 「split()」メソッドで分割する	
063	使用可能な最大値/最小値を取得する	170
	ONEPOINT ■ 扱える値の最大値を取得するには「Number.MAX_VALUE」、 最小値を取得するには「Number.MIN_VALUE」を使う	

□ 64	演算誤差を減らす	171
	ONEPOINT ■ 演算誤差を減らすにはいったん整数にして演算する	
□ 65	大きな数値同士の演算を行う	173
	ONEPOINT ■ 正しい演算結果を得るにはBigDecimal.jsライブラリを利用する	
	COLUMN ■ BigDecila.jsの演算メソッド	
□ 66	ビット演算を行う	177
	ONEPOINT ■ ビット演算を行うにはビット演算子を使う	
□ 67	絶対値を求める	179
	ONEPOINT ■ 絶対値を求めるには「Math.abs()」メソッドを使う	
□ 68	値を比較する	180
	ONEPOINT ■ 値の比較を行うには「Math.min()」メソッドや「Math.max()」メソッドを使う	
□ 69	三角関数で計算する	181
	ONEPOINT ■ 三角関数を取得するには「Math.sin()」メソッドや「Math.cos()」メソッドなどを使う	
□ 70	円周率を求める	183
	ONEPOINT ■ 円周率を扱うには「Math.PI」を使う	
	COLUMN ■ 度(degree)とラジアン(radian)を変換するには	
□ 71	四捨五入/切り捨て/切り上げを行う	184
	ONEPOINT ■ 四捨五入を行うには「Math.round()」メソッド、切り捨てを行うには「Math.floor()」メソッド、切り上げを行うには「Math.ceil()」メソッドを使う	
	COLUMN ■ その他の切り捨て方法	
□ 72	自然対数の値を求める	186
	ONEPOINT ■ 自然対数の値を求めるには「Math.log()」メソッドや「Math.E」「Math.LN2」「Math.LN10」を使う	
	COLUMN ■ 底が10の場合の対数と底が2の場合の対数を求めるには	
□ 73	eの累乗を求める	188
	ONEPOINT ■ eの累乗を求めるには「Math.exp()」メソッドを使う	
□ 74	べき乗を求める	189
	ONEPOINT ■ べき乗計算を行うには「Math.pow()」メソッドを使う	
□ 75	平方根を求める	190
	ONEPOINT ■ 平方根を求めるには「Math.sqrt()」メソッドを使う	
□ 76	乱数を求める	191
	ONEPOINT ■ 乱数を求めるには「Math.random()」メソッドを使う	
	COLUMN ■ 整数値の乱数を求めるには	

CHAPTER 05 文字列・正規表現

- 077 「String」オブジェクトを生成する/文字列を連結する 194
 - ONEPOINT ■ 「String」オブジェクトを生成するには「new String()」や「」を使う
 - COLUMN ■ サロゲートペアの文字
 - COLUMN ■ Latin-1コード表
- 078 文字列の長さを求める 197
 - ONEPOINT ■ 文字列の長さを求めるには「length」プロパティを使う
- 079 文字と文字コードを変換する 198
 - ONEPOINT ■ 文字を文字コードに変換するには「charCodeAt()」メソッド、文字コードを文字に変換するには「String.fromCharCode()」メソッドを使う
 - COLUMN ■ 文字コードについて
- 080 文字列を検索する 200
 - ONEPOINT ■ 文字列の検索を行うには「indexOf()」メソッドや「lastIndexOf()」メソッドを使う
- 081 文字を取り出す 202
 - ONEPOINT ■ 文字を取り出すには「charAt()」メソッドを使う
 - COLUMN ■ 文字を文字コードで取得するには
- 082 文字列を指定文字で分割する 204
 - ONEPOINT ■ 文字列を指定文字で分割するには「split()」メソッドを使う
- 083 指定範囲の文字を抜き出す 206
 - ONEPOINT ■ 文字列を抜き出すには「substring()」メソッドや「slice()」メソッド、「substr()」メソッドを使う
- 084 英文字の大文字/小文字に変換する 208
 - ONEPOINT ■ 英文字を小文字に変換するには「toLowerCase()」メソッド、大文字に変換するには「toUpperCase()」メソッドを使う
 - COLUMN ■ 全角数字を半角数字に変換する
- 085 文字列の前後にある空白を削除する 210
 - ONEPOINT ■ 文字列の前後の空白を削除するには「trim()」メソッドを使う
- 086 特定の文字列を削除する 211
 - ONEPOINT ■ 文字列を削除するには「split()」メソッドのパラメータに削除したい文字を指定する
 - COLUMN ■ 文字列の数をカウントする
- 087 「RegExp」オブジェクトを生成する 213
 - ONEPOINT ■ 「RegExp」オブジェクトを生成するには「new RegExp()」とする
 - COLUMN ■ 正規表現文字の一覧
- 088 正規表現を使って文字を検索する 216
 - ONEPOINT ■ 正規表現を使って検索するには「match()」メソッドを使う
- 089 正規表現を使って文字を置換する 218
 - ONEPOINT ■ 正規表現を使って文字を置換するには「replace()」メソッドを使う

CHAPTER 06 日付・時刻

090	「Date」オブジェクトを生成する	222
	ONEPOINT ■ 日付を扱うには「Date」オブジェクトを使う	
091	日付を求める	224
	ONEPOINT ■ 日付を読み出すには「getYear()」メソッドや「getDate()」メソッドなどを使う	
092	時刻を求める	226
	ONEPOINT ■ 時刻を読み出すには「getHours()」メソッドや「getMinutes()」メソッドを使う	
	COLUMN ■ JavaScriptの「Date」オブジェクトで扱える日付の限界	
093	協定世界時の日付を求める	228
	ONEPOINT ■ 協定世界時を読み出すには「getUTCFullYear()」メソッドや「getUTCDate()」メソッドなどを使う	
	COLUMN ■ グリニッジ標準時	
094	協定世界時の時刻を求める	230
	ONEPOINT ■ 協定世界時の時刻を読み出すには「getUTCHours()」メソッドや「getUTCMinutes()」メソッドなどを使う	
095	1970年1月1日午前0時0分0秒から指定時までの 経過ミリ秒を求める	232
	ONEPOINT ■ 経過ミリ秒を取得するには「Date.now()」や「getTime()」メソッドを使う	
096	日付を設定する	234
	ONEPOINT ■ 日付を設定するには「setFullYear()」「setMonth()」「setDate()」メソッドを使う	
097	時刻を設定する	236
	ONEPOINT ■ 時刻を設定するには「setHours()」メソッドや「setMinutes()」メソッドなどを使う	
098	協定世界時の日付を設定する	238
	ONEPOINT ■ 協定世界時の日付を設定するには「setUTCFullYear()」「setUTCMonth()」「setUTCDate()」メソッドを使う	
099	協定世界時の時刻を設定する/時差を求める	240
	ONEPOINT ■ 協定世界時の時刻を設定するには「setUTCHours()」「setUTCMinutes()」メソッドなどを使う	
100	さまざまな日付や時刻の形式に変換する	242
	ONEPOINT ■ 地域に合わせた日付と時刻を取得するには「toLocaleDateString()」メソッドや「toLocaleTimeString()」メソッドを使う	
101	グリニッジ標準時を求める	244
	ONEPOINT ■ グリニッジ標準時を求めるには「toGMTString()」メソッドを使う	
	COLUMN ■ GMTなのにUTCと表示される	
102	日付情報をJSON形式に変換する	246
	ONEPOINT ■ 日付をJSON形式に変換するには「toJSON()」メソッドを使う	
	COLUMN ■ JSONについて	

CHAPTER 07 ブラウザオブジェクト

1 0 3	ブラウザ名やバージョンを取得する	250
	ONEPOINT ■ ブラウザに関する情報を取得するには「navigator」オブジェクトを使う	
	COLUMN ■ ブラウザを判別するには	
1 0 4	JavaやGeolocationなどの機能が使えるかどうか調べる	252
	ONEPOINT ■ 利用できる機能のチェックを行うにはオブジェクトの存在を確認する	
1 0 5	接続回線の種類を調べる	253
	ONEPOINT ■ 接続回線の種類を調べるには 「navigator.connection.type」プロパティを使う	
1 0 6	履歴数を求める	255
	ONEPOINT ■ 履歴数を取得するには「history」オブジェクトの 「length」プロパティを使う	
1 0 7	ページを進める/戻す	256
	ONEPOINT ■ 履歴間でページを移動するには 「back()」「forward()」「go()」メソッドを使う	
1 0 8	履歴を追加・置換する	258
	ONEPOINT ■ 履歴を追加するには「pushState()」メソッド、 置換するには「replaceState()」メソッドを使う	
1 0 9	画面のサイズを求める	261
	ONEPOINT ■ モニタ画面のサイズを調べるには「screen」オブジェクトの 「width」プロパティや「height」プロパティを使う	
	COLUMN ■ マルチモニタの場合	
1 1 0	画面の色深度を求める	263
	ONEPOINT ■ 画面の色深度を調べるには「screen」オブジェクトの 「colorDepth」プロパティを使う	
1 1 1	画面の解像度を調べる	265
	ONEPOINT ■ 画面の解像度を調べるには「deviceXDPI」プロパティや 「deviceYDPI」プロパティ、「devicePixelRatio」プロパティを使う	
1 1 2	ウィンドウの幅を求める	267
	ONEPOINT ■ ウィンドウ幅を求めるには「outerWidth」「outerHeight」 「innerWidth」「innerHeight」プロパティを使う	
1 1 3	指定位置までスクロールさせる	269
	ONEPOINT ■ ページ内容をスクロールさせるには「scrollTo()」メソッドや 「scrollBy()」メソッドを使う	
	COLUMN ■ スマートフォンでアドレスバーを隠す	
1 1 4	Base64エンコード/デコードする	272
	ONEPOINT ■ Base64エンコードするには「btoa()」メソッド、 Base64デコードするには「atob()」メソッドを使う	
1 1 5	印刷する	274
	ONEPOINT ■ 印刷するには「print()」メソッドを使う	

1 1 6	選択された文字を取得する	276
	ONEPOINT ■ 選択された文字を取得するには「getSelection()」メソッドと「getRangeAt()」メソッドを使う	
1 1 7	アラートダイアログ/確認ダイアログ/入力ダイアログを表示する	278
	ONEPOINT ■ 各種ダイアログを表示するには「alert()」「confirm()」「prompt()」メソッドを使う	
	COLUMN ■ 表示できない文字	
	COLUMN ■ Windows Phoneでは「prompt()」メソッドは使えない	
1 1 8	モーダルダイアログを表示する	281
	ONEPOINT ■ モーダルダイアログを表示するには「showModalDialog()」メソッドを使う	
1 1 9	タイマーを設定・解除する	283
	ONEPOINT ■ タイマーを設定するには「setTimeout()」メソッド、タイマーを解除するには「clearTimeout()」メソッドを使う	
	COLUMN ■ 「setTimeout()」メソッドの3番目以降のパラメータ	
1 2 0	インターバルタイマーを設定・解除する	286
	ONEPOINT ■ インターバルタイマーを設定するには「setInterval()」メソッド、解除するには「clearInterval()」メソッドを使う	
1 2 1	アニメーション同期タイマーを設定・解除する	289
	ONEPOINT ■ アニメーション同期タイマーを設定するには「window.requestAnimationFrame()」メソッド、解除するには「window.cancelAnimationFrame()」メソッドを使う	
	COLUMN ■ ベンダープレフィックスについて	
1 2 2	非同期処理を行う	292
	ONEPOINT ■ 非同期処理を行うには「setImmediate()」メソッドを使う	
	COLUMN ■ 「setTimeout()」メソッドを使った非同期処理	
1 2 3	ドキュメントのタイトルを取得・設定する	294
	ONEPOINT ■ ドキュメントのタイトルを取得・設定するには「title」プロパティを使う	
1 2 4	ドキュメント内に出力する	296
	ONEPOINT ■ ページへの出力を行うには「document.write()」メソッドを使う	
1 2 5	クッキーに書き込みが可能かどうか調べる	297
	ONEPOINT ■ クッキーが利用できるか調べるには「cookieEnabled」プロパティを使う	
1 2 6	クッキーの読み書きを行う	299
	ONEPOINT ■ クッキーの読み書きを行うには「document.cookie」プロパティを使う	
1 2 7	ドメイン名・ホスト名・パス名・プロトコルを求める	302
	ONEPOINT ■ URL情報を取得するには「location」オブジェクトと「document」オブジェクトのプロパティを使う	
1 2 8	直前に見ていたページのURLを求める	304
	ONEPOINT ■ 前に見ていたページURLを取得するには「document.referrer」プロパティを使う	
1 2 9	ページをリロードする	305
	ONEPOINT ■ ページをリロードするには「reload()」メソッドを使う	

1 3 0	ページのURLを取得・設定する	307
	ONEPOINT ■ URLの取得と移動を行うには「href」プロパティを使う	
1 3 1	「Image」オブジェクトを生成する	309
	ONEPOINT ■ 「Image」オブジェクトを生成するには「new Image()」とする	
1 3 2	画像の幅を設定・取得する	311
	ONEPOINT ■ 画像のサイズを取得・設定するには「width」プロパティと「height」プロパティを使う	
1 3 3	画像のオリジナルの幅を取得する	313
	ONEPOINT ■ 画像の元のサイズを取得するには「naturalWidth」プロパティと「naturalHeight」プロパティを使う	
1 3 4	画像データが読み込まれた場合に処理する	315
	ONEPOINT ■ 画像の読み込み完了のチェックを行うには「onload」プロパティを使う	
1 3 5	画像データを入れ替える	317
	ONEPOINT ■ 画像の入れ替えを行うには「src」プロパティを使う	
	COLUMN ■ CSSでローラーオーバーを行うには	
1 3 6	フォーム要素にアクセスする	319
	ONEPOINT ■ フォームを制御するには要素にアクセスする	
1 3 7	ラジオボタン/チェックボックスの状態を設定・取得する	321
	ONEPOINT ■ ラジオボタンとチェックボックスの選択状態を調べるには「checked」プロパティを使う	
1 3 8	フォーム要素のフォーカスを設定する	323
	ONEPOINT ■ フォーカスを設定するには「focus()」メソッドや「blur()」メソッド、「select()」メソッドを使う	
1 3 9	フォーム内容を読み出す	325
	ONEPOINT ■ 入力された内容を読み出すには「value」プロパティを使う	
1 4 0	内容をDate型/数値型として読み出す	327
	ONEPOINT ■ 数値型として読み出すには「valueAsNumber」プロパティ、日付型として読み出すには「valueAsDate」プロパティを使う	
1 4 1	フォームへの入力を禁止・許可する	329
	ONEPOINT ■ テキストフィールドへの入力を禁止するには「disabled」プロパティを使う	
1 4 2	セレクトメニューで選択された項目を取得する	331
	ONEPOINT ■ 選択された項目の情報を取得するには「selectedIndex」プロパティや「value」プロパティ、「text」プロパティを使う	
1 4 3	フォーム内容を初期化する	333
	ONEPOINT ■ フォーム内容を初初期化するには「reset()」メソッドを使う	
1 4 4	数値専用のフィールドで値を増減する	335
	ONEPOINT ■ ボタンで値を増減させるには「stepUp()」メソッドと「stepDown()」メソッドを使う	
1 4 5	非同期通信オブジェクトを生成する	337
	ONEPOINT ■ 非同期通信を行うには「XMLHttpRequest」オブジェクトを使う	

1 4 6	サーバーからテキストデータやXMLデータを受信する	339
	ONEPOINT ■サーバーからテキストデータを受信するには「responseText」プロパティ、XMLデータを受信するには「responseXML」プロパティを使う	
1 4 7	リクエストヘッダーを設定する	342
	ONEPOINT ■リクエストヘッダーを設定するには「setRequestHeader()」メソッドを使う	
	COLUMN ■IE6 ～ 9でキャッシュデータが利用されるのを回避する方法	
1 4 8	MIMEタイプを設定する	344
	ONEPOINT ■MIMEタイプを設定するには「overrideMimeType()」メソッドを使う	
1 4 9	デバッグコンソールに文字を出力する	346
	ONEPOINT ■デバッガのコンソールに出力するには「console.log()」メソッドを使う	
	COLUMN ■「console.log()」メソッドの出力結果について	
	COLUMN ■IEの場合	
	COLUMN ■その他のコンソール出力用メソッド	
	COLUMN ■デバッガを表示するには	
1 5 0	デバッグコンソールにオブジェクトの内容を出力する	349
	ONEPOINT ■オブジェクトの中身を確認するには「console.dir()」メソッドを使う	
	COLUMN ■「console.dir()」メソッドの出力結果について	
1 5 1	処理にかかった時間を詳細に把握する	351
	ONEPOINT ■処理時間を詳しく計測するにはプロファイルを利用する	
	COLUMN ■「console.profile()」メソッドの出力結果について	
1 5 2	呼び出し元の関数を把握する	353
	ONEPOINT ■呼び出し元の関数を調べるには「console.trace()」メソッドを使う	
	COLUMN ■「console.trace()」メソッドの出力結果について	

CHAPTER 08 DOM

1 5 3	特定のID名を持つ要素にアクセスする	356
	ONEPOINT ■特定のID名を持つ要素を操作するには「getElementById()」メソッドを使う	
	COLUMN ■DOMの読み込みタイミング	
1 5 4	特定のスタイルシートクラスを持つ要素にアクセスする	358
	ONEPOINT ■特定のスタイルシートクラスを持つ要素にアクセスするには「getElementsByClassName()」メソッドを使う	
1 5 5	要素名を指定してアクセスする	360
	ONEPOINT ■要素名を指定してアクセスするには「getElementsByTagName()」メソッドを使う	
1 5 6	特定の名前を持つ要素にアクセスする	362
	ONEPOINT ■名前を指定してアクセスするには「getElementsByName()」メソッドを使う	
1 5 7	セレクタを使って要素にアクセスする	364
	ONEPOINT ■セレクタを使って要素にアクセスするには「querySelector()」「querySelectorAll()」メソッドを使う	

1 5 8	要素を生成する	366
	ONEPOINT ■ 要素を生成するには「document.createElement()」メソッドを使う	
1 5 9	テキストノードを生成する	368
	ONEPOINT ■ テキストノードを生成するには 「document.createTextNode()」メソッドを使う	
1 6 0	子ノードにアクセスする	370
	ONEPOINT ■ 子ノードにアクセスするには 「firstChild」「childNodes」「lastChild」プロパティを使う	
1 6 1	前後のノードや親ノードにアクセスする	372
	ONEPOINT ■ 前後のノードにアクセスするには「nextSibling」「previousSibling」プロパティ、親ノードにアクセスするには「parentNode」プロパティを使う	
1 6 2	ノードを追加・挿入する	374
	ONEPOINT ■ ノードを追加するには「appendChild()」メソッド、 挿入するには「insertBefore()」メソッドを使う	
1 6 3	子ノードを削除・置換する	376
	ONEPOINT ■ 子ノードを削除するには「removeChild()」メソッド、 置換するには「replaceChild()」メソッドを使う	
1 6 4	ノードの内容にアクセスする	378
	ONEPOINT ■ ノードの内容にアクセスするには「nodeValue」プロパティを参照する	
1 6 5	要素の内容を書き換える	380
	ONEPOINT ■ 要素の内容を書き換えるには「innerHTML」プロパティや 「innerText」プロパティを使う	
1 6 6	属性値を取得・設定する	382
	ONEPOINT ■ 属性値を取得するには「getAttribute()」メソッド、 設定するには「setAttribute()」メソッドを使う	
	COLUMN ■ IE6/IE7での「class」属性	
1 6 7	属性を新規作成・削除する	384
	ONEPOINT ■ 新規に属性を作成するには「document.createAttribute()」メソッド、 削除するには「removeAttribute()」メソッドを使う	
1 6 8	ノードを一括して追加する	386
	ONEPOINT ■ ノードを一括して追加するには「document.createDocument Fragment()」メソッドを使って他のDOMツリーを用意する	
	COLUMN ■ ノードを追加する際の速度について	
1 6 9	Shadow DOMとして要素を追加する	388
	ONEPOINT ■ Shadow DOMとして追加するには 「new WebKitShadowRoot()」とする	
	COLUMN ■ Chromeでの設定	
1 7 0	通常のDOMツリーの要素をShadow DOMにする	390
	ONEPOINT ■ 要素をShadow DOMにするには 「new WebKitShadowRoot()」とする	
	COLUMN ■ Chromeでの設定	

- 171 Shadow DOMにある要素にアクセスする 392
 ONEPOINT ■ Shadow DOMにある要素にアクセスするには
 DOM関連のメソッドを使う
 COLUMN ■ Chromeでの設定

CHAPTER 09 イベント

- 172 イベントハンドラを設定する 396
 ONEPOINT ■ イベントハンドラを設定するには要素に割り当てられたプロパティを使う
- 173 イベントリスナーを設定する 398
 ONEPOINT ■ イベントリスナーを設定するには
 「addEventListener()」メソッドを使う
 COLUMN ■ IE6 ~ 8でイベントリスナーを設定するには
- 174 押されたキーを調べる 401
 ONEPOINT ■ 押されたキーを調べるには「keyCode」プロパティや
 修飾キー専用のプロパティを使う
 COLUMN ■ スマートフォンでのキー入力
- 175 イベントが発生した座標を求める 404
 ONEPOINT ■ イベントが発生した座標を求めるには「clientX」プロパティや
 「clientY」プロパティなどを使う
- 176 発生したイベントの種類を求める 406
 ONEPOINT ■ 発生したイベントの種類を求めるには「type」プロパティを使う
- 177 マウスホイールの値を取得する 408
 ONEPOINT ■ マウスホイールの値を取得するには「mousewheel」イベントや
 「DOMMouseScroll」イベントを検知する
- 178 トラックパッドの移動量を求める 410
 ONEPOINT ■ トラックパッドの移動量を求めるには
 「MozMousePixelScroll」イベントを検知する
 COLUMN ■ マウスホイールとの関係
- 179 フォーカスした際に処理を行う 412
 ONEPOINT ■ フォーカスに関する処理を行うには「focus」イベントや
 「blur」イベントを使う
- 180 クリック時/ダブルクリック時の処理を行う 414
 ONEPOINT ■ クリック時に処理を行うには「click」イベント、
 ダブルクリック時に処理を行うには「dblclick」イベントを捕捉する
- 181 マウスオーバー時/マウスアウト時の処理を行う 416
 ONEPOINT ■ マウスオーバー時の処理を行うには「mouseover」イベント、
 マウスアウト時の処理を行うには「mouseout」イベントを捕捉する
- 182 マウスダウン時/マウスアップ時の処理を行う 418
 ONEPOINT ■ マウスダウン時の処理を行うには「mousedown」イベント、
 マウスアップ時の処理を行うには「mouseup」イベントを捕捉する

1 8 3	マウスの右ボタンが押されたときに処理する	420
	ONEPOINT ■ マウスの右ボタンが押されたときに処理するには「contextmenu」イベントを捕捉する	
	COLUMN ■ Windows Phoneの場合	
1 8 4	カット/コピー /ペーストされた際に処理を行う	422
	ONEPOINT ■ カット・コピー・ペーストされた際に処理を行うには「copy」イベントや「paste」イベントなどを補足する	
1 8 5	ドラッグ時に処理を行う	424
	ONEPOINT ■ ドラッグ時に処理を行うには「dragstart」「drag」「dragend」イベントを補足する	
1 8 6	ページの読み込みやDOMの構築が完了した際に処理を行う	427
	ONEPOINT ■ DOMの構築が完了した際に処理を行うには「DOMContentLoaded」イベントを検知する	
1 8 7	リセット時・送信時に処理する	429
	ONEPOINT ■ リセット時に処理するには「reset」イベント、送信時に処理するには「submit」イベントを補足する	
1 8 8	スクロール時に処理を行う	431
	ONEPOINT ■ スクロール時に処理するには「scroll」イベントを捕捉する	
1 8 9	テキストフィールド内のテキスト選択時に処理を行う	433
	ONEPOINT ■ テキスト選択時に処理を行うには「select」イベントを捕捉する	
	COLUMN ■ IE6 ~ 8で動作させるには	
1 9 0	指定した座標にある複数の要素を求める	436
	ONEPOINT ■ 指定した座標にある要素を求めるには「elementFromPoint()」「elementsFromPoint()」「elementsFromRect()」メソッドを使う	
1 9 1	CSSのメディアクエリが変化したら処理する	438
	ONEPOINT ■ メディアクエリが変化したら処理するには「window.matchMedia()」メソッドと「addListener()」メソッドを使う	

CHAPTER 10 Canvas/Canvas 3D/SVG

1 9 2	コンテキストを取得する	442
	ONEPOINT ■ コンテキストを取得するには「getContext()」メソッドを使う	
1 9 3	パスを作成する	444
	ONEPOINT ■ 新規にパスを作成するには「beginPath()」メソッド、クローズパスに設定するには「closePath()」メソッドを使う	
	COLUMN ■ Androidでの不具合	
1 9 4	直線を描画する	446
	ONEPOINT ■ 直線を描画を行うには「lineTo()」メソッドを使う	
	COLUMN ■ 連続して直線を描くには	
1 9 5	四角形を描画する	448
	ONEPOINT ■ 四角形を描画を行うには「fillRect()」「strokeRect()」「clearRect()」「rect()」メソッドを使う	

196	円や円弧を描く	450
	ONEPOINT ■ 円や円弧の描画を行うには「arc()」メソッドや「arcTo()」メソッドを使う	
	COLUMN ■ Androidでの不具合	
	COLUMN ■ Operaでの不具合	
197	曲線を描く	453
	ONEPOINT ■ 曲線の描画を行うには「bezierCurveTo()」メソッドや「quadraticCurveTo()」メソッドを使う	
198	多角形を描く	455
	ONEPOINT ■ 多角形を描画するには角度からX座標とY座標を求める	
199	角丸四角形を描く	457
	ONEPOINT ■ 角丸四角形を描画するには「lineTo()」メソッドと「arc()」メソッドを組み合わせる	
200	ドーナツなどの中をくり抜いた図形を描く	459
	ONEPOINT ■ 中をくり抜いた図形を描くにはパスのベクトルを考慮する	
	COLUMN ■ Androidでの不具合	
201	色と線のスタイルを設定する	461
	ONEPOINT ■ スタイルの指定を行うには「strokeStyle」プロパティを使う	
202	不透明度を設定する	463
	ONEPOINT ■ 不透明度の指定を行うには「globalAlpha」プロパティを使う	
203	描画モードを設定する	465
	ONEPOINT ■ 描画モードの指定を行うには「globalCompositeOperation」プロパティを使う	
204	クリッピングする	468
	ONEPOINT ■ クリッピング領域の指定を行うには「clip()」メソッドを使う	
205	文字を描画する	470
	ONEPOINT ■ 文字を描画するには「fillText()」メソッドや「strokeText()」メソッドを使う	
	COLUMN ■ Safari 5とOperaでの不具合	
206	文字の幅を求める	472
	ONEPOINT ■ 文字幅を求めるにはテキストメトリクスオブジェクトの「width」プロパティを使う	
	COLUMN ■ 検討中のテキストメトリクスのプロパティ	
207	文字の位置揃えを指定する	474
	ONEPOINT ■ 文字の位置揃えを指定するには「textAlign」プロパティを使う	
208	グラデーションを設定する	476
	ONEPOINT ■ グラデーションを描画するには「createLinearGradient()」メソッドや「createRadialGradient()」メソッドを使う	
209	画像を描画する	478
	ONEPOINT ■ 画像を描画するには「drawImage()」メソッドを使う	

2 1 0	グレースケール画像にする	480
	ONEPOINT ■ グレースケール画像にするにはピクセルデータを読み出して 計算式に従って変換する	
	COLUMN ■ ローカルディスク上での動作について	
	COLUMN ■ 処理する画像が大きい場合	
	COLUMN ■ Safari 6とRetina Displayの場合	
2 1 1	エンボス加工する	484
	ONEPOINT ■ エンボス加工するには2つのピクセルの輝度の差分を取る	
	COLUMN ■ ローカルディスク上での動作について	
	COLUMN ■ 処理する画像が大きい場合	
2 1 2	画像をぼかす	488
	ONEPOINT ■ 画像をぼかすにはピクセルの平均を取る	
	COLUMN ■ ローカルディスク上での動作について	
	COLUMN ■ 処理する画像が大きい場合	
	COLUMN ■ ガウスぼかしを適用するには	
2 1 3	画像のガンマ補正を行う	492
	ONEPOINT ■ 画像のガンマ補正を行うには「 $255 \times (\text{輝度} \div 255)^{1/\gamma}$ 」の計算式を使う	
	COLUMN ■ ローカルディスク上での動作について	
	COLUMN ■ 処理する画像が大きい場合	
2 1 4	画像を白黒化(2値化)する	495
	ONEPOINT ■ 画像を白黒化(2値化)するにはしきい値によって 黒と白にピクセルを置き換える	
	COLUMN ■ ローカルディスク上での動作について	
	COLUMN ■ 処理する画像が大きい場合	
2 1 5	画像を特定の色相(トーン)にする	499
	ONEPOINT ■ 画像を特定の色相(トーン)にするにはピクセルの色情報をRGBから HSVやHSLなどに変換する	
	COLUMN ■ ローカルディスク上での動作について	
	COLUMN ■ 処理する画像が大きい場合	
2 1 6	パターンを生成する	503
	ONEPOINT ■ パターンを生成するには「createPattern()」メソッドを使う	
2 1 7	影を描画する	505
	ONEPOINT ■ 影を描画するには「shadowColor」「shadowBlur」 「shadowOffsetX」「shadowOffsetY」プロパティを使う	
	COLUMN ■ Androidの「shadowOffsetY」プロパティの不具合	
2 1 8	映像をCanvasに描画する	507
	ONEPOINT ■ 映像を描画するには「drawImage()」メソッドを使う	
2 1 9	Canvasの内容を他のCanvasに描画する	509
	ONEPOINT ■ 他のCanvasの内容を描画するには「drawImage()」メソッドを使う	
2 2 0	図形を回転させる	512
	ONEPOINT ■ 回転を行うには「rotate()」メソッドを使う	
2 2 1	図形を移動させる	514
	ONEPOINT ■ 移動処理を行うには「translate()」メソッドを使う	

2.2.2	図形を拡大・縮小させる	516
	ONEPOINT ■ 拡大・縮小を行うには「scale()」メソッドを使う	
2.2.3	図形や画像を変形させる	518
	ONEPOINT ■ 図形や画像を変形させるには「transform()」メソッドや「setTransform()」メソッドを使う	
2.2.4	パス内に座標点があるか調べる	520
	ONEPOINT ■ パス内に座標点があるか調べるには「isPointInPath()」メソッドを使う	
2.2.5	Canvasの状態を保存・復元する	523
	ONEPOINT ■ Canvasの状態を保存するには「save()」メソッド、復元するには「restore()」メソッドを使う	
2.2.6	Canvasに描画済みの内容をスクロールする	525
	ONEPOINT ■ Canvasの内容をスクロールするには「getImageData()」メソッドと「putImageData()」メソッドを使う	
2.2.7	Canvasの画像データをdataURL形式に変換する	527
	ONEPOINT ■ ピクセルデータをdataURL形式に変換するには「toDataURL()」メソッドを使う	
	COLUMN ■ ローカルディスク上での動作	
2.2.8	画像をローカルストレージに保存する	529
	ONEPOINT ■ 画像を保存するには「toDataURL()」メソッドを使う	
	COLUMN ■ ローカルディスク上での動作	
2.2.9	WebGLが利用できるか調べる	532
	ONEPOINT ■ 3D描画が可能か調べるには「getContext()」メソッドを使う	
2.3.0	Three.jsを使って3D処理を行う	534
	ONEPOINT ■ 3D処理を簡単に行うにはThree.jsを使う	
2.3.1	ワイヤーフレーム表示の幅と不透明度を設定する	537
	ONEPOINT ■ ワイヤーフレームの幅を指定するには「wireframeLinewidth」プロパティ、不透明度を指定するには「opacity」プロパティを使う	
2.3.2	カメラを設定する	539
	ONEPOINT ■ 設定できるカメラには透視投影のカメラと正投影のカメラ、複合カメラがある	
2.3.3	光源を追加する	542
	ONEPOINT ■ 平行光源を追加するには「new THREE.DirectionalLight()」とする	
2.3.4	立方体を生成する	544
	ONEPOINT ■ 立方体を生成するには「new THREE.CubeGeometry()」とする	
2.3.5	球体を生成する	546
	ONEPOINT ■ 球体を生成するには「new THREE.SphereGeometry()」とする	
2.3.6	トーラスを生成する	549
	ONEPOINT ■ トーラスを生成するには「new THREE.TorusGeometry()」とする	
2.3.7	トーラス結び目を生成する	551
	ONEPOINT ■ トーラス結び目を生成するには「new THREE.TorusKnotGeometry()」とする	

238	平面を生成する	553
	ONEPOINT ■ 平面を生成するには「new THREE.PlaneGeometry()」とする	
239	円筒を生成する	555
	ONEPOINT ■ 円筒を生成するには「new THREE.CylinderGeometry()」とする	
240	正二十面体を生成する	557
	ONEPOINT ■ 正二十面体を生成するには 「new THREE.IcosahedronGeometry()」とする	
241	回転体を生成する	560
	ONEPOINT ■ 回転体を生成するには「new THREE.LatheGeometry()」とする	
242	線を生成する	563
	ONEPOINT ■ 線を生成するには「new THREE.Geometry()」とする	
243	テクスチャマッピングを行う	566
	ONEPOINT ■ 画像を貼り付けるには 「new THREE.ImageUtils.loadTexture()」を使う	
244	周りの景色を反射させて表示する	569
	ONEPOINT ■ 環境マップを設定するには 「THREE.ImageUtils.loadTextureCube()」を使う	
245	立方体などの物体を回転させる	572
	ONEPOINT ■ 回転させるには「rotation.x」「rotation.y」「rotation.z」プロパティを使う	
246	立方体などの物体を移動させる	574
	ONEPOINT ■ 移動させるには「position.x」「position.y」「position.z」プロパティを使う	
247	立方体などの物体のスケールを変更する	577
	ONEPOINT ■ スケールを変更するには「scale.x」「scale.y」「scale.z」プロパティを使う	
248	文字を3Dで描画する	580
	ONEPOINT ■ Three.jsで文字を描画するにはtypeface.jsを使う COLUMN ■ ベベルを指定するには	
249	霧(フォグ)を設定する	583
	ONEPOINT ■ 霧(フォグ)を設定するには「fog」プロパティを使う	
250	立体の影を表示する	586
	ONEPOINT ■ 立体の影を表示するには「shadowMapEnabled」プロパティなどを使う	
251	WebGLが使えないデバイスでも3D描画を可能にする	589
	ONEPOINT ■ iPhoneやAndroidで3D描画を可能にするには 「new THREE.CanvasRenderer()」とする COLUMN ■ IE9で表示されない場合について	
252	SVGの図形のサイズを変更する	592
	ONEPOINT ■ SVGの図形のサイズを変更するには 図形を示す要素の「width」属性と「height」属性の値を変更する	
253	SVGの図形の位置を変更する	594
	ONEPOINT ■ SVGの図形の位置を変更するには 図形を示す要素の「x」属性と「y」属性の値を変更する	

254	SVGの図形を変形する	596
	ONEPOINT ■ SVGの図形を変形するには「transform」属性を使う	
255	SVGの図形の塗りや線の色を変更する	599
	ONEPOINT ■ SVGの図形の塗りや線の色を変更するには 図形を示す要素の「fill」属性や「stroke」属性の値を変更する	
256	SVGのテキストの内容を変更する	601
	ONEPOINT ■ SVGのテキストの内容を変更するには 「textContent」プロパティを使う	
257	SVGの図形のグラデーションの種類を変更する	603
	ONEPOINT ■ SVGの図形のグラデーションの種類を変更するには 図形を示す要素の「fill」属性の値を変更する	

CHAPTER 11 Audio/Video/HTML Media

258	オーディオオブジェクトを生成する	608
	ONEPOINT ■ オーディオオブジェクトを生成するには「new Audio()」を使う	
	COLUMN ■ 音を生成する他の方法	
	COLUMN ■ FirefoxやOperaでの音声の再生について	
259	オーディオを再生・停止する	610
	ONEPOINT ■ オーディオを再生するには「play()」メソッド、 停止するには「pause()」メソッドを使う	
260	再生時間を取得・設定する	612
	ONEPOINT ■ 演奏開始時間と再生時間を取得・設定するには 「currentTime」プロパティを使う	
261	オーディオの長さを調べる	614
	ONEPOINT ■ オーディオの長さを取得するには「duration」プロパティを使う	
262	指定されたオーディオ形式が再生可能かどうか調べる	616
	ONEPOINT ■ 再生可能なオーディオファイルの形式を調べるには 「canPlayType()」メソッドを使う	
263	オーディオファイルの読み込みに応じて変化する 通信状態を調べる	618
	ONEPOINT ■ 通信状態を調べるには「networkState」プロパティを使う	
	COLUMN ■ Windows Phoneの場合の「networkState」プロパティと 「readyState」プロパティ	
264	オーディオのボリュームを設定・ミュートにする	620
	ONEPOINT ■ ボリュームを設定するには「volume」プロパティ、 ミュートするには「muted」プロパティを使う	
265	オーディオのコントローラーの表示/非表示を設定する	623
	ONEPOINT ■ オーディオのコントローラーの表示制御を行うには 「controls」プロパティを使う	
266	オーディオデータを読み込む	625
	ONEPOINT ■ オーディオデータを読み込ませるには「src」プロパティを使う	

267	オーディオの再生状態を調べる	627
	ONEPOINT ■ オーディオの状態を取得するには「currentTime」プロパティや「paused」プロパティなどを使う	
268	オーディオの再生速度を変更する	629
	ONEPOINT ■ 再生速度を変更するには「playbackRate」プロパティを使う	
269	オーディオの再生時のイベントを取得する	631
	ONEPOINT ■ オーディオが再生された際に処理を設定するには「timeupdate」イベントを使う	
270	オーディオの再生が終了した際のイベントを取得する	633
	ONEPOINT ■ オーディオの再生が終了した際の処理を設定するには「ended」イベントを使う	
271	オーディオのボリュームが変更された際のイベントを取得する	635
	ONEPOINT ■ ボリューム変更時の処理を行うには「volumechange」イベントを使う	
272	「video」要素にアクセスする	637
	ONEPOINT ■ 映像にアクセスするには「video」要素を使う COLUMN ■ FirefoxやOperaでの映像の再生について	
273	映像を再生・停止する	639
	ONEPOINT ■ 再生を行うには「play()」メソッド、 停止を行うには「pause()」メソッドを使う	
274	映像の再生時間を取得・設定する	641
	ONEPOINT ■ 再生開始時間の設定と再生時間の取得には「currentTime」プロパティを使う COLUMN ■ スマートフォンの場合の「currentTime」プロパティ	
275	映像の長さを調べる	643
	ONEPOINT ■ 映像の長さを取得するには「duration」プロパティを使う	
276	指定した形式の映像が再生可能かどうか調べる	645
	ONEPOINT ■ 映像ファイルの形式が再生可能かどうか調べるには「canPlayType()」メソッドを使う	
277	映像ファイルの読み込みに応じて変化する通信状態を調べる	647
	ONEPOINT ■ 通信状態を調べるには「networkState」プロパティを使う	
278	映像のボリュームを設定・ミュートする	649
	ONEPOINT ■ ボリュームを調整するには「volume」プロパティ、 ミュートを設定するには「muted」プロパティを使う	
279	映像のコントローラーの表示/非表示を設定する	652
	ONEPOINT ■ コントローラーの処理を行うには「controls」プロパティを使う	
280	映像データを読み込む	654
	ONEPOINT ■ 映像データを読み込ませるには「src」プロパティを使う	
281	映像の再生状態を調べる	656
	ONEPOINT ■ 映像の状態を取得するには「currentTime」プロパティや「paused」プロパティなどを使う	

282	映像の再生速度を変更する	659
	ONEPOINT ■再生速度を変更するには「playbackRate」プロパティを使う	
283	映像の再生時のイベントを取得する	661
	ONEPOINT ■映像の再生時の処理を設定するには「timeupdate」イベントを使う	
284	映像の再生が終了した際のイベントを取得する	663
	ONEPOINT ■映像の再生が終了したら処理するには「ended」イベントを使う	
285	映像のボリュームが変更された際のイベントを取得する	665
	ONEPOINT ■ボリューム変更時の処理を行うには「volumechange」イベントを使う	
286	カメラからの映像を表示する	667
	ONEPOINT ■カメラ映像を表示するにはWebRTC APIを使う	
	COLUMN ■Google Chromeでの設定	
	COLUMN ■Opera MobileでのWebRTC	
287	カメラからの映像をCanvasに描画する	670
	ONEPOINT ■カメラ映像をCanvasに描画するには「drawImage()」メソッドの最初のパラメータに「video」要素を指定する	
	COLUMN ■Google Chromeでの設定	
288	カメラで撮影した画像を表示する	673
	ONEPOINT ■カメラで撮影した画像を表示するにはHTML Media Captureを使う	
	COLUMN ■ビデオカメラとマイクからの取り込み	
	COLUMN ■「capture」属性に指定できる値	
289	カメラで撮影した画像をCanvasに描画する	676
	ONEPOINT ■カメラで撮影した画像をCanvasに描画するには「Image」オブジェクトの「onload」プロパティにイベントハンドラを設定する	

CHAPTER 12 File API

290	ファイル名や種類を取得する	680
	ONEPOINT ■ファイル情報を取得するには「name」プロパティや「size」プロパティ、「type」プロパティを使う	
291	テキストファイルを読み込む	683
	ONEPOINT ■テキストファイルを読み込むには「readAsText()」メソッドを使う	
292	バイナリファイルを読み込む	685
	ONEPOINT ■バイナリファイルを読み込むには「readAsBinaryString()」メソッドを使う	
293	画像や映像ファイルを読み込む	687
	ONEPOINT ■画像ファイルや映像ファイルを読み込むには「readAsDataURL()」メソッドを使う	
294	ファイルの読み込み状況を表示する	689
	ONEPOINT ■ファイルの読み込み状況を表示するには「progress」イベントを使う	
295	指定したディスク容量が使用可能か調べる	691
	ONEPOINT ■使用可能な割り当て容量を調べるには「window.storageInfo.requestQuota()」メソッドを使う	

296	ディスクの使用状況を調べる	694
	ONEPOINT ■ ディスクの使用状況を調べるには 「window.storageInfo.queryUsageAndQuota()」メソッドを使う	
297	ファイルを作成する	696
	ONEPOINT ■ ファイルを作成するには「window.requestFileSystem()」メソッドと 「fs.root.getFile()」メソッドを使う	
298	ファイルにテキストを書き込む	699
	ONEPOINT ■ ファイルにテキストを書き込むには「createWriter()」メソッドと 「BlobBuilder」オブジェクトを使う	
	COLUMN ■ 「getBlob()」メソッドの動作の違い	
299	保存されているファイルの内容を読み出す	702
	ONEPOINT ■ 保存されているファイルの内容を読み出すには 「fs.root.getFile()」メソッドと「FileReader」オブジェクトを使う	
300	保存されているファイルを削除する	705
	ONEPOINT ■ ファイルを削除するには「fs.root.getFile()」メソッドと 「remove()」メソッドを使う	
301	ディレクトリを作成する	707
	ONEPOINT ■ ディレクトリを作成するには「window.requestFileSystem()」メソッド と「fs.root.getDirectory()」メソッドを使う	
302	サブディレクトリを作成する	710
	ONEPOINT ■ サブディレクトリを作成するには「getDirectory()」メソッドを使う	
303	ディレクトリを削除する	713
	ONEPOINT ■ ディレクトリを削除するには「fs.root.getDirectory()」メソッドと 「remove()」メソッドや「removeRecursively()」メソッドを使う	
304	ディレクトリー一覧を取得する	716
	ONEPOINT ■ ディレクトリー一覧を取得するには「createReader()」メソッドを使う	
305	ファイルをコピーする	719
	ONEPOINT ■ ファイルをコピーするには「copyTo()」メソッドと 「getParent()」メソッドを使う	
306	ディレクトリをコピーする	722
	ONEPOINT ■ ディレクトリをコピーするには「copyTo()」メソッドと 「getParent()」メソッドや「getDirectory()」メソッドを使う	
307	ファイルの移動とファイル名の変更を行う	725
	ONEPOINT ■ ファイルの移動やファイル名の変更には「moveTo()」メソッドを使う	
308	URL形式でファイルにアクセスする	728
	ONEPOINT ■ ファイルやディレクトリはURL形式でアクセスすることができる	

CHAPTER 13 HTML5のその他のAPI

309	一度だけ位置情報を求める	732
	ONEPOINT ■ 位置情報を一度だけ取得するにはGeolocation APIの「getCurrentPosition()」メソッドを使う	
310	定期的に位置情報を取得・停止する	735
	ONEPOINT ■ 定期的に位置情報を取得するには「watchPosition()」メソッド、停止するには「clearWatch()」メソッドを使う	
	COLUMN ■ 機能が利用できるか調べるには	
311	Web Storageにデータを設定する	738
	ONEPOINT ■ ストレージを利用するには「sessionStorage」や「localStorage」を使う	
312	Web Storageからデータを読み出す	740
	ONEPOINT ■ ストレージから読み出すには「getItem()」メソッドを使う	
313	Web Storage内のすべてのキーと値を取得する	742
	ONEPOINT ■ キーを取得するには「key()」メソッドを使う	
	COLUMN ■ Web Storageに保存されている内容を調べる	
314	Web Storageの特定のデータを削除する	744
	ONEPOINT ■ 特定のデータを消去するには「removeItem()」メソッドを使う	
315	Web Storageのすべてのデータを削除する	746
	ONEPOINT ■ 全データを消去するには「clear()」メソッドを使う	
316	ストレージ内容が変更された際のイベントを設定する	748
	ONEPOINT ■ データ保存時の処理を設定するには「storage」イベントを使う	
317	ワーカーオブジェクトを生成する	750
	ONEPOINT ■ ワーカーオブジェクトを作成するには「new Worker()」を使う	
318	ワーカーにデータを渡す	752
	ONEPOINT ■ ワーカーとやり取りするには「postMessage()」メソッドを使う	
319	ワーカー側でJavaScriptファイルを読み込む	754
	ONEPOINT ■ ワーカー側でJavaScriptファイルを読み込むには「importScripts()」メソッドを使う	
320	ワーカーを終了する	756
	ONEPOINT ■ ワーカーを終了するには「terminate()」メソッドや「close()」メソッドを使う	
321	オフライン/オンライン状態を調べる	759
	ONEPOINT ■ オンラインかどうか調べるには「onLine」プロパティを使う	
322	アプリケーションキャッシュを更新する	760
	ONEPOINT ■ アプリケーションキャッシュを更新するには「swapCache()」メソッドを使う	
323	タッチ数を求める	764
	ONEPOINT ■ 同時タッチ処理を行うにはタッチ関係のイベントを使う	
	COLUMN ■ 同時タッチ数の制限	

324	タッチされた座標を求める	766
	ONEPOINT ■ タッチされた座標を調べるには「clientX」「clientY」「pageX」「pageY」「screenX」「screenY」を使う	
325	加速度センサーの値を求める	768
	ONEPOINT ■ 加速度センサーを扱うには「devicemotion」イベントを使う	
326	ジャイロセンサーの値を求める	770
	ONEPOINT ■ ジャイロセンサーを扱うには「deviceorientation」イベントを使う	
327	ジェスチャーされたら拡大や回転処理を行う	772
	ONEPOINT ■ ジェスチャーによる回転と拡大・縮小を行うには「gesturechange」イベントなどを使う	
328	デバイスが回転したら処理を行う	774
	ONEPOINT ■ デバイスの回転時に処理を行うには「orientationchange」イベントを使う	
329	日付の入力を行うようにする	776
	ONEPOINT ■ 日付を選択するには「input」要素の「type」属性に「date」を指定する	
330	日時の入力を行うようにする	778
	ONEPOINT ■ 日付と時間をまとめて選択するには「input」要素の「type」属性に「datetime」を指定する	
331	メールやURL、電話番号を入力する専用フィールドにする	780
	ONEPOINT ■ メールアドレスやURLの入力専用フィールドにするには「input」要素の「type」属性に「email」や「url」を指定する	
332	数値入力・スライダー入力の専用フィールドにする	782
	ONEPOINT ■ 数値入力専用にするには「input」要素の「type」属性で「number」を指定する	
333	カラーピッカーを表示する	784
	ONEPOINT ■ カラーピッカーを表示するには「input」要素の「type」属性に「color」を指定する	
334	データリストを定義する	786
	ONEPOINT ■ データリストを表示するには「datalist」要素を使う	
335	音声入力を行う	788
	ONEPOINT ■ 音声入力を行うには「input」要素に「x-webkit-speech」属性を指定する	
336	ブラウザのページ読み込みタイミングを取得する	790
	ONEPOINT ■ 読み込みタイミングを取得するには「performance.navigation」オブジェクトの各種プロパティを使う	
	COLUMN ■ ミリ秒以下の計測	
337	異なるオリジンのインラインフレームにメッセージを送信する	793
	ONEPOINT ■ インラインフレームにメッセージを送信するには「postMessage()」メソッドを使う	

338	映像をフルスクリーン表示にする	796
	ONEPOINT ■映像をフルスクリーンにするには「video」要素の「requestFullscreen()」メソッドを使う	
	COLUMN ■Fullscreen APIのベンダープレフィックス	
339	画像をフルスクリーン表示にする	799
	ONEPOINT ■画像をフルスクリーンにするには「img」要素の「requestFullscreen()」メソッドを使う	
	COLUMN ■Fullscreen APIのベンダープレフィックス	
340	テキストをフルスクリーン表示にする	802
	ONEPOINT ■テキストをフルスクリーンにするにはテキストを囲んでいる要素の「requestFullscreen()」メソッドを使う	
	COLUMN ■Fullscreen APIのベンダープレフィックス	
341	インラインフレームをフルスクリーン表示にする	805
	ONEPOINT ■インラインフレームをフルスクリーンにするには「iframe」要素の「requestFullscreen()」メソッドを使う	
	COLUMN ■Fullscreen APIのベンダープレフィックス	
342	フルスクリーンと通常表示が切り替わったら処理を行う	808
	ONEPOINT ■画面切り替え時に処理を行うには「fullscreenchange」イベントを捕捉する	
343	バイブレーション機能を使う	811
	ONEPOINT ■バイブレーション機能を使うにはVibration APIを使う	
344	バッテリーの充電レベルを取得する	813
	ONEPOINT ■バッテリーの充電レベルを取得するには「battery」オブジェクトの「level」プロパティを使う	
345	バッテリーの充電状態を取得する	815
	ONEPOINT ■バッテリーの充電状態を取得するには「battery」オブジェクトの「charging」プロパティを使う	
346	バッテリーの充電状態が変化したら処理を行う	817
	ONEPOINT ■充電状態が変化した場合に処理を行うには「chargingchange」イベントを使う	
347	バッテリーレベルが変化した場合に処理する	819
	ONEPOINT ■バッテリーレベルが変化した場合に処理するには「levelchange」イベントを使う	
348	ゲームパッドが使用可能か調べる	821
	ONEPOINT ■ゲームパッドを使用可能か調べるには「gamepads」オブジェクトを使う	
	COLUMN ■ゲームパッドを有効にするには	
349	要素内の文字を編集可能にする	823
	ONEPOINT ■要素内の文字や画像を編集するには「contentEditable」プロパティを使う	
350	WebSocketが使用可能か調べる	825
	ONEPOINT ■WebSocketが利用できるかどうかは「WebSocket」オブジェクトの有無を調べる	

3 5 1	WebSocketでサーバーに接続する	827
	ONEPOINT ■ サーバーに接続するには「new WebSocket()」とする	
3 5 2	WebSocketを使ってサーバーとデータをやり取りする	829
	ONEPOINT ■ サーバーとやり取りするにはメッセージ送信を使う	
	COLUMN ■ Node.jsとは	
	COLUMN ■ サンプルの動作について	
3 5 3	Indexed Databaseが使用可能か調べる	833
	ONEPOINT ■ Indexed Databaseが使用できるかどうかは「IndexedDB」オブジェクトの有無を調べる	
3 5 4	Indexed Databaseでデータベースを開く	835
	ONEPOINT ■ Indexed Databaseでデータベースを開くには「open()」メソッドを使う	
3 5 5	Indexed Databaseでオブジェクトストアを作成する	837
	ONEPOINT ■ Indexed Databaseでオブジェクトストアを作成するには「createObjectStore()」メソッドを使う	
3 5 6	Indexed Databaseにデータを書き込む	841
	ONEPOINT ■ オブジェクトストアにデータを書き込むにはトランザクションを利用する	
	COLUMN ■ Indexed Databaseの仕様変更とサンプルの動作について	
3 5 7	Indexed Databaseからデータを読み出す	845
	ONEPOINT ■ オブジェクトストアのデータを読み出すにはトランザクションを利用する	
	COLUMN ■ Indexed Databaseの仕様変更とサンプルの動作について	
3 5 8	Web Audioが使用可能か調べる	848
	ONEPOINT ■ Web Audioが利用できるかどうかは「audioContext」オブジェクトの有無を調べる	
3 5 9	Web Audioで音を鳴らす	850
	ONEPOINT ■ Web Audio APIで音を鳴らすにはオーディオコンテキストを生成する	
3 6 0	Web Audioでファイルを読み込んで演奏する	852
	ONEPOINT ■ Web Audio APIでオーディオファイルを演奏するには非同期通信を使ってファイルを読み込ませておく	
3 6 1	Web Audioでボリュームを指定する	855
	ONEPOINT ■ Web Audio APIでボリュームを指定するには「createGainNode()」メソッドを使ってゲインノードを生成する	
3 6 2	Pointer Lock APIが使用可能か調べる	857
	ONEPOINT ■ マウスポインタがロック可能かどうか調べるには「requestPointerLock()」メソッドの有無を調べる	
3 6 3	タブを切り換えた場合に処理を行う	859
	ONEPOINT ■ タブが切り替わったときに処理を行うには「visibilitychange」イベントを使う	
3 6 4	要素をドラッグできるようにする	862
	ONEPOINT ■ 要素をドラッグできるようにするには「draggable」プロパティを使う	

365	要素をドラッグ&ドロップできるようにする	864
	ONEPOINT ■ ドロップ側の処理は「dragover」「dragleave」「drop」イベントを使う	
366	ドラッグ要素からドロップ要素にデータを渡す	866
	ONEPOINT ■ データを受け渡すには「setData()」メソッドを使う	
367	通知可能かどうか調べる	869
	ONEPOINT ■ 通知が利用できるかどうかは 「webkitNotifications」オブジェクトの有無で調べる	
	COLUMN ■ Safari 6での動作について	
	COLUMN ■ Google Chromeでの動作について	
368	通知する	872
	ONEPOINT ■ 通知するには「createNotification()」メソッドを使う	
369	通知を消す	874
	ONEPOINT ■ 通知を消すには「cancel()」メソッドを使う	
370	通知に関するイベントを処理する	876
	ONEPOINT ■ 通知に関するイベントを処理するには 「Notification」オブジェクトのイベントを利用する	

CHAPTER 14 スタイルシート

371	スタイルシートを生成する/読み込む	880
	ONEPOINT ■ スタイルシートを生成したり読み込んだりするには 「document.createElement()」メソッドを使う	
372	スタイルシートプロパティの値を設定する	882
	ONEPOINT ■ スタイルシートプロパティの値を設定するには 要素の「style」オブジェクトの各プロパティに値を書き込む	
373	スタイルシートの値を読み出す	884
	ONEPOINT ■ スタイルシートの値を読み出すには 要素の「style」オブジェクトの各プロパティを参照する	
374	文字の色を変更する	886
	ONEPOINT ■ 文字の色を変更するにはスタイルシートの「color」プロパティを使う	
375	文字のサイズを変更する	888
	ONEPOINT ■ 文字のサイズを変更するには「fontSize」プロパティを使う	
376	文字の書体を変更する	889
	ONEPOINT ■ 文字の書体を変更するには スタイルシートの「fontFamily」プロパティを使う	
	COLUMN ■ iOS 5でのフォント	
377	文字の太さを変更する	891
	ONEPOINT ■ 文字の太さを変更するには スタイルシートの「fontWeight」プロパティを使う	
378	文字のスタイルを変更する	893
	ONEPOINT ■ 文字のスタイルを変更するには スタイルシートの「fontStyle」プロパティを使う	

379	文字の形状を変更する	895
	ONEPOINT ■ 文字の形状を変更するには スタイルシートの「fontVariant」プロパティを使う	
380	文字の間隔を変更する	897
	ONEPOINT ■ 文字の間隔を変更するには スタイルシートの「letterSpacing」プロパティを使う	
381	単語の間隔を変更する	899
	ONEPOINT ■ 単語の間隔を変更するには スタイルシートの「wordSpacing」プロパティを使う	
382	行間を変更する	901
	ONEPOINT ■ 行間を変更するにはスタイルシートの「lineHeight」プロパティを使う	
383	行揃えを変更する	902
	ONEPOINT ■ 行揃えを変更するにはスタイルシートの「textAlign」プロパティを使う	
384	文字を装飾する	904
	ONEPOINT ■ 文字を装飾するには スタイルシートの「textDecoration」プロパティを使う	
385	字下げ(インデント)を変更する	906
	ONEPOINT ■ 字下げ(インデント)を変更するには スタイルシートの「textIndent」プロパティを使う	
386	英文字の表示形式を指定する	907
	ONEPOINT ■ 英文字の表示形式を指定するには スタイルシートの「textTransform」プロパティを使う	
387	文字に影を表示する	909
	ONEPOINT ■ 文字に影を表示するには スタイルシートの「textShadow」プロパティを使う	
388	要素の背景色を変更する	910
	ONEPOINT ■ 要素の背景色を変更するには スタイルシートの「color」プロパティを使う	
389	要素の背景画像を変更する	912
	ONEPOINT ■ 要素の背景画像を変更するには スタイルシートの「backgroundImage」プロパティを使う	
390	要素の背景画像の位置を変更する	914
	ONEPOINT ■ 要素の背景画像の位置を変更するには スタイルシートの「backgroundPosition」プロパティを使う	
391	要素の背景画像の繰り返し方法を変更する	916
	ONEPOINT ■ 要素の背景画像の繰り返し方法を変更するには スタイルシートの「backgroundRepeat」プロパティを使う	
392	背景画像の固定方法を変更する	918
	ONEPOINT ■ 要素の背景画像の固定方法を変更するには スタイルシートの「backgroundAttachment」プロパティを使う	

393	要素の位置を変更する	920
	ONEPOINT ■ 要素の位置を変更するには スタイルシートの「position」プロパティを使う	
394	要素の余白を変更する	922
	ONEPOINT ■ 要素の余白を変更するには スタイルシートの「margin」プロパティや「padding」プロパティを使う	
395	要素の枠を変更する	924
	ONEPOINT ■ 要素の枠を変更するにはスタイルシートの「border」プロパティを使う	
396	要素の表示形式を変更する	926
	ONEPOINT ■ 要素の表示形式を変更するには スタイルシートの「display」プロパティを使う	
397	要素の横幅・縦幅を変更する	928
	ONEPOINT ■ 要素の横幅を変更するにはスタイルシートの「width」プロパティ、 縦幅を変更するにはスタイルシートの「height」プロパティを使う COLUMN ■ iPhone/Androidで表示領域のサイズを指定する	
398	要素のはみ出した部分の表示方法を変更する	930
	ONEPOINT ■ 要素のはみ出した部分の表示方法を変更するには スタイルシートの「overflow」プロパティを使う	
399	要素からはみ出した文字の表示方法を変更する	932
	ONEPOINT ■ 要素のはみ出した部分の文字の表示方法を変更するには スタイルシートの「textOverflow」プロパティを使う	
400	要素の表示方法を変更する	934
	ONEPOINT ■ 要素の表示方法を変更するには スタイルシートの「visibility」プロパティを使う	
401	要素のZ座標を変更する	936
	ONEPOINT ■ 要素のZ座標を変更するには スタイルシートの「zIndex」プロパティを使う	
402	要素の表示倍率を変更する	938
	ONEPOINT ■ 要素の表示倍率を変更するには スタイルシートの「zoom」プロパティを使う COLUMN ■ スマートフォンでの表示倍率	
403	要素の角丸を変更する	940
	ONEPOINT ■ 要素の角丸を変更するには スタイルシートの「borderRadius」プロパティを使う	
404	要素に影を表示する	942
	ONEPOINT ■ 要素に影を表示するには スタイルシートの「boxShadow」プロパティを使う	
405	不透明度を指定する	944
	ONEPOINT ■ 不透明度を指定するにはスタイルシートの「opacity」プロパティを使う	
406	リサイズ処理を指定する	946
	ONEPOINT ■ 要素をリサイズするにはスタイルシートの「resize」プロパティを使う	

407	アニメーション時間を変更する	948
	ONEPOINT ■ アニメーション時間を変更するには スタイルシートの「animationDuration」プロパティを使う	
408	3D変形を行う	950
	ONEPOINT ■ 3D変形を行うにはスタイルシートの「transform」プロパティを使う	
409	トランジションを設定する	952
	ONEPOINT ■ トランジションを設定するには スタイルシートの「transition」プロパティを使う	
410	要素にグレースケールを適用する	954
	ONEPOINT ■ グレースケールにするにはスタイルシートの「filter」プロパティに 「grayscale(値)」の文字列を指定する	
	COLUMN ■ 「filter」プロパティのプロパティ名について	
411	要素をセピア調にする	956
	ONEPOINT ■ セピア調にするにはスタイルシートの「filter」プロパティに 「sepia(値)」の文字列を指定する	
	COLUMN ■ 「filter」プロパティのプロパティ名について	
412	要素にトーン(色相変換)フィルタを適用する	958
	ONEPOINT ■ トーン(色相)を変更するにはスタイルシートの「filter」プロパティに 「hue-rotate(値)」の文字列を指定する	
	COLUMN ■ 「filter」プロパティのプロパティ名について	
413	要素に色反転フィルタを適用する	960
	ONEPOINT ■ 色は反転させるにはスタイルシートの「filter」プロパティに 「invert(値)」の文字列を指定する	
	COLUMN ■ 「filter」プロパティのプロパティ名について	
414	要素の輝度(明るさ)を変更する	962
	ONEPOINT ■ 明るさを調整するにはスタイルシートの「filter」プロパティに 「brightness(値)」の文字列を指定する	
	COLUMN ■ 「filter」プロパティのプロパティ名について	
415	要素のコントラストを変更する	965
	ONEPOINT ■ コントラストを調整するにはスタイルシートの「filter」プロパティに 「contrast(値)」の文字列を指定する	
	COLUMN ■ 「filter」プロパティのプロパティ名について	
416	要素の彩度を変更する	968
	ONEPOINT ■ 彩度を調整するにはスタイルシートの「filter」プロパティに 「saturate(値)」の文字列を指定する	
	COLUMN ■ 「filter」プロパティのプロパティ名について	
	COLUMN ■ その他のCSSフィルタ	
417	要素にぼかしフィルタを適用する	971
	ONEPOINT ■ ぼかすにはスタイルシートの「filter」プロパティに 「blur(値)」の文字列を指定する	
	COLUMN ■ 「filter」プロパティのプロパティ名について	

4 1 8 CSSのメディアクエリを調べる973

ONEPOINT ■メディアクエリが変化したら処理するには
「window.matchMedia()」メソッドと「addListener()」メソッドを使う

●索引|.....976



CHAPTER 01

JavaScriptの 基礎知識と基本文法

JavaScriptの概要

III 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JUnit (<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るがちです。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、一般的にいわゆる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 INTERNATIONAL Standards

What is Ecma Activities News Standards

Printer Friendly Version

« Back

Standard ECMA-262

ECMAScript Language Specification

Edition 5.1 (June 2011)

This Standard defines the ECMAScript scripting language.

The following file can be freely downloaded:

File name	Size (Bytes)	Content
ECMA-262.pdf	3 067 290	Acrobat (r) PDF file

This edition 5.1 of the ECMAScript Standard is fully aligned with third edition of the international standard ISO/IEC 16262:2011.

This ECMAScript Language Specification is also available in HTML file format [here](#).

The previous replaced "historical" editions of this Ecma Standard are available [here](#).

« Back

▶ 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

III 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、純粹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

III 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	$15 \wedge 9$	6 ($1111 \wedge 1001 = 0110$)
!	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。

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

リテラルについて

III 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、「`{`」と「`}`」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)。

III 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。



SECTION-006

コメントについて

III 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>
```

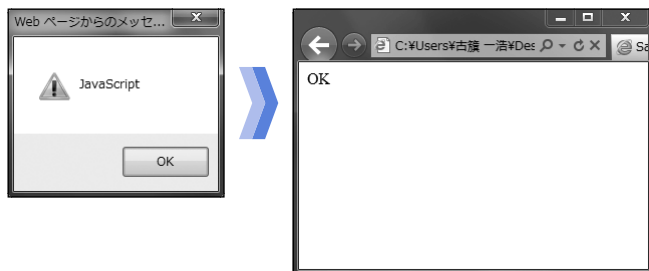
SAMPLE CODE JavaScriptのコード(sample.js)

```
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

III 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。

III 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

III 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。

III 「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。

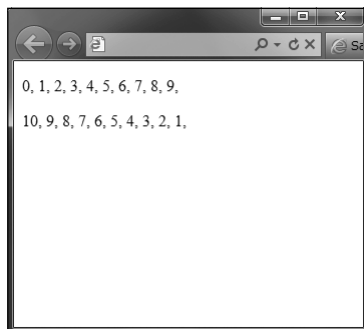
SAMPLE CODE HTMLのコード(index.html)

```
<!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。



III 「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){  
    繰り返し処理  
}
```

```
do{  
    繰り返し処理  
} 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>
```

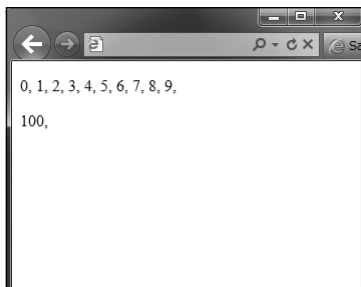


```
</body>  
</html>
```

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回繰り返し  
    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 <100)  
    // 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

III 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。

III 「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
```

III 「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
}
```

III 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

III 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)

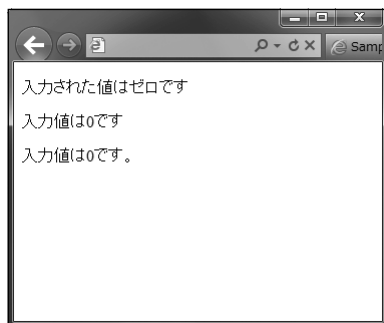
```
window.addEventListener("load", function(){
  var ele = document.getElementsByTagName("output");
  // ダイアログで数値を入力
  var num = prompt("0～9までの数値を入れて下さい", 1) | 0;
  // 0かどうか調べる
  if (num === 0){
    ele[0].innerHTML = "入力された値はゼロです";
  }else{
```

```

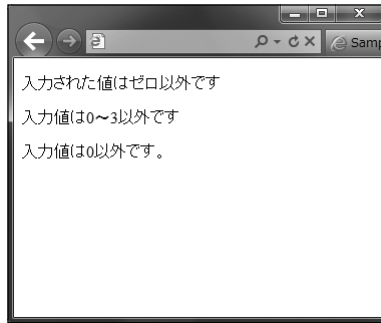
    ele[0].innerHTML = "入力された値はゼロ以外です";
}
// 入力された数値に応じて処理を分ける
switch(num){
    case 0 : ele[1].innerHTML = "入力値は0です";
        break;
    case 1 : ele[1].innerHTML = "入力値は1です";
        break;
    case 2 : ele[1].innerHTML = "入力値は2です";
        break;
    case 3 : ele[1].innerHTML = "入力値は3です";
        break;
    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。

III 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。

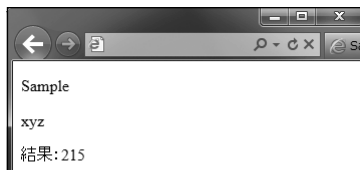
SAMPLE CODE HTMLのコード(index.html)

```
<!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

関数のパラメータの読み出しについて

III 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。

III 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)

```

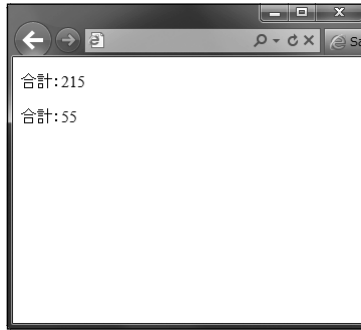
window.addEventListener("load", function(){
  // 関数calcを呼び出し
  var n = calc(200, 15);
  document.getElementsByTagName("output")[0].innerHTML = "合計:" + n;
  // パラメータの数を変更して呼び出し
  n = calc(1,2,3,4,5,6,7,8,9,10);
  document.getElementsByTagName("output")[1].innerHTML = "合計:" + n;
}, false);
// パラメータの数だ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

III 「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。

III 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。



SECTION-013

例外処理について

III JavaScriptでの例外処理

JavaScript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	「encodeURIComponent()」「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。

III 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。

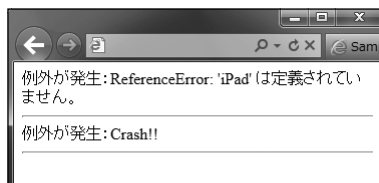
SAMPLE CODE HTMLのコード(index.html)

```
<!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。

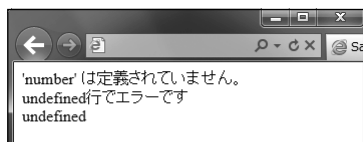
SAMPLE CODE HTMLのコード(index.html)

```
<!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。



SECTION-014

イベント処理について

III 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。

III 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おり、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>
```

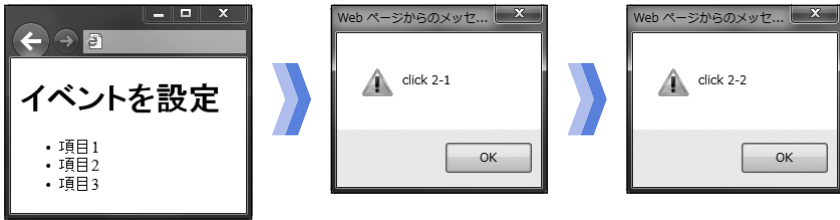
SAMPLE CODE JavaScriptのコード(sample.js)

```
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）。

ドキュメントオブジェクトモデルについて

III 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进行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
<code>getElementById()</code>	指定したID名の要素にアクセスする
<code>getElementsByTagName()</code>	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
<code>getElementsByName()</code>	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
<code>querySelector()</code>	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
<code>querySelectorAll()</code>	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
<code>createElement()</code>	指定された要素を生成する
<code>createTextNode()</code>	テキストノードを生成する
<code>appendChild()</code>	子ノードとして末尾に追加する

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

III 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>

```

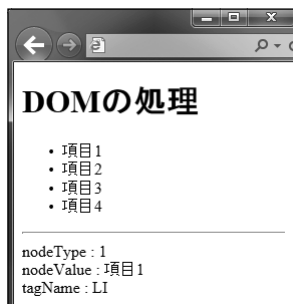
SAMPLE CODE JavaScriptのコード(sample.js)

```

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);
}

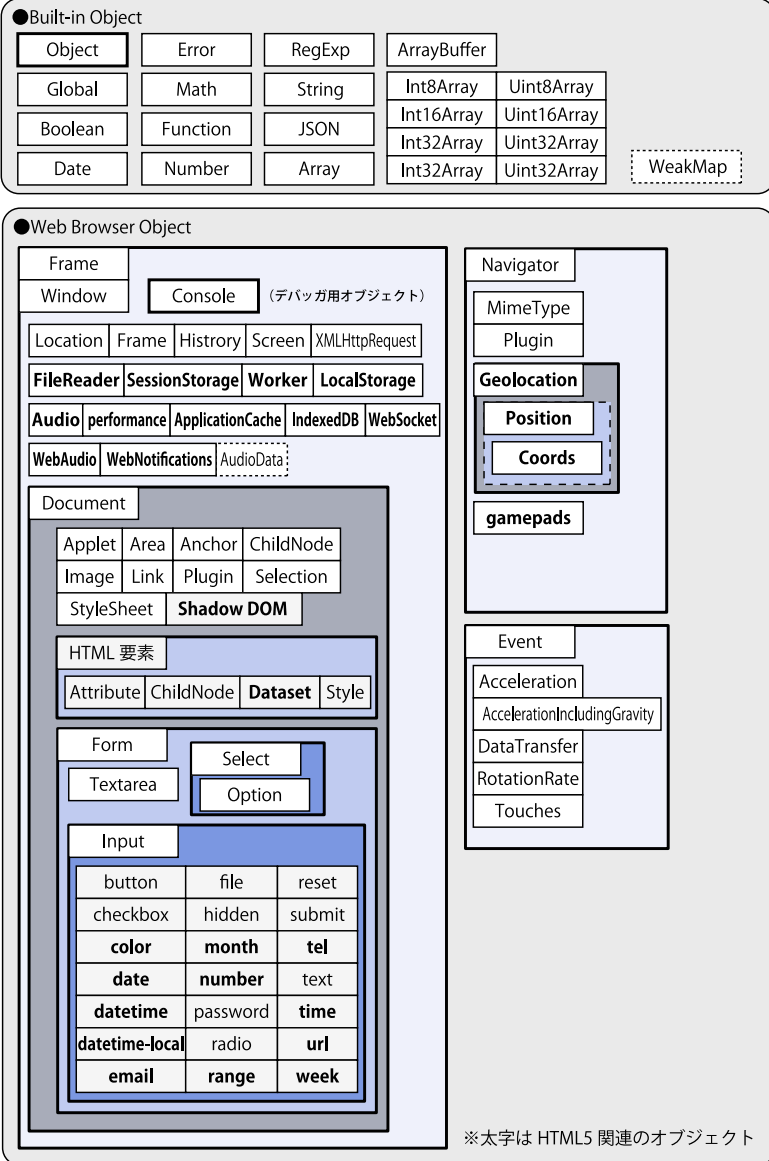
```

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。

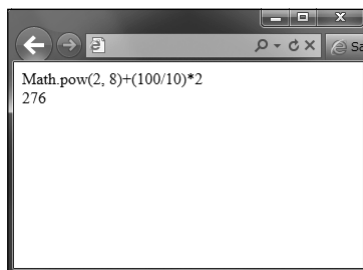
SAMPLE CODE HTMLのコード(index.html)

```
<!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、これを使用の方がよいでしょう。

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。

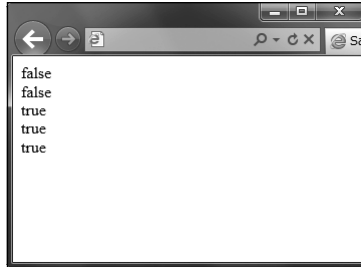
SAMPLE CODE HTMLのコードのコード(index.html)

```
<!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。

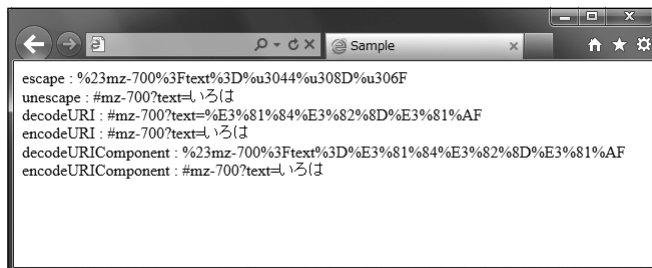
SAMPLE CODE HTMLのコード(index.html)

```
<!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 = encodeURIComponent(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>encodeURIComponent : "+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()」「encodeURIComponent()」「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()」「encodeURIComponent()」「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	{		}	~	

●「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	{		}	~	

●「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

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。

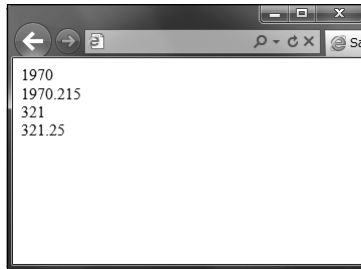
SAMPLE CODE HTMLのコード(index.html)

```
<!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){
```

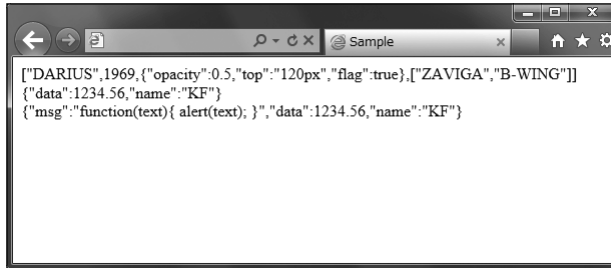


```

    if (new String(value).indexOf("function") > -1) {
        return new String(value);
    }
    return value;
});
// 結果を出力
ele.innerHTML += jsontdata3+"<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を行うことができます。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。

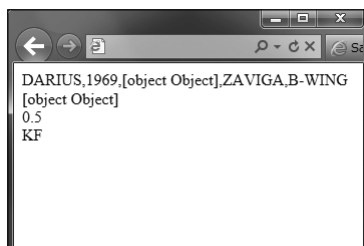
SAMPLE CODE HTMLのコード(index.html)

```
<!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。

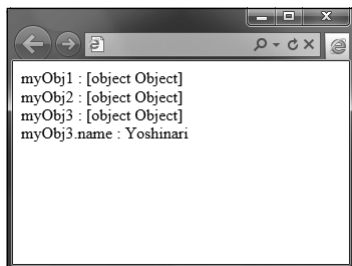
SAMPLE CODE HTMLのコード(index.html)

```
<!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;
```

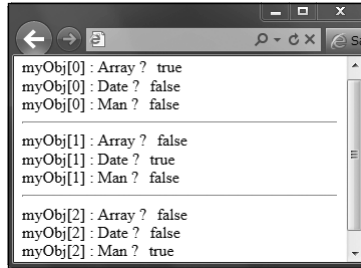


```

    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。

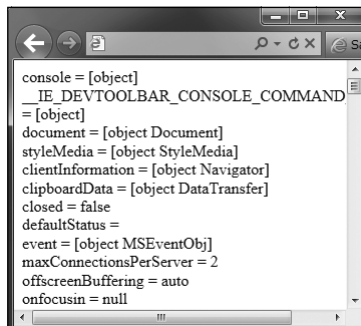
SAMPLE CODE HTMLのコード(index.html)

```
<!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);
```