

JavaScript 入門

— HTML5 アプリケーション開発のためのプログラミング —

第 1.1 版

Copyright © 2014-2021, Katsunori Nakamura

中村勝則

2021 年 1 月 16 日

免責事項

本書に掲載したプログラムリストは全て試作品です。また本書の使用に伴って発生した損害、不利益の一切の責任を筆者は負わないことをご了承ください。

目次

1	JavaScript とは	1
2	学習に必要なもの	1
2.1	プログラミングに適したテキストエディタ	2
2.1.1	クロスプラットフォームに対応したテキストエディタ	2
3	JavaScript を学ぶ前に	3
3.1	HTML の基本	3
3.1.1	基本構造	3
3.1.1.1	改行要素	4
3.1.1.2	段落要素	4
3.1.2	GUI コンポーネント	4
3.1.2.1	ボタン	4
3.1.2.2	チェックボックス	4
3.1.2.3	ラジオボタン	4
3.1.2.4	テキストフィールド	5
3.1.2.5	パスワードフィールド	5
3.1.2.6	テキストエリア	5
3.1.2.7	選択リスト	6
3.1.2.8	スライダ	6
3.1.2.9	メータ, 進捗バー	7
3.1.3	HTML のコメント記述	8
3.1.4	HTML タグの誤りを検査する方法	8
3.2	CSS の基本	9
3.2.1	基礎事項	9
3.2.1.1	セレクタの書き方に関して	9
3.2.2	基本的なスタイル要素	10
3.2.2.1	フォントの設定 (スタイル, サイズなど)	10
3.2.2.2	行の高さと文字間の設定	11
3.2.2.3	段落開始のインデント	11
3.2.2.4	色の設定	11
3.2.3	ハイパーリンクのスタイル	12
3.2.4	HTML 要素の不透明度 (opacity)	12
3.2.5	コンテンツの配置に関する基礎	12
3.2.5.1	div 要素	13
3.2.5.2	位置とサイズの設定	14
3.2.5.3	枠線と余白	14
3.2.5.4	枠線の太さ, 余白の大きさ, 表示の位置, サイズの関係	15
3.2.5.5	GUI レイアウトの例	16
3.2.6	div 要素内でのテキストの配置の設定	17
3.2.6.1	div 要素内の下に子要素を配置する工夫	18
3.2.7	HTML の span タグについて	18
3.2.8	ブロックレベル要素とインライン要素	18
3.2.9	CSS のコメント記述	19
3.2.10	CSS を別のファイルに分離する方法	19

3.2.11	CSS の誤りを検査する方法	19
3.3	表と箇条書き	20
3.3.1	表: table 要素	20
3.3.1.1	表に関するスタイル	20
3.3.2	箇条書き: ul 要素, ol 要素	21
3.3.2.1	箇条書きの見出しの設定	21
3.3.2.2	箇条書き項目を横並びにする方法	21
3.4	画像データの配置	22
3.4.1	画像の位置, サイズについて	22
4	JavaScript を使う前に	23
4.1	まずは体験	23
4.1.1	JavaScript プログラムを別のファイルに分離する方法	24
4.1.2	プログラム開発やデバッグのための機能	24
4.2	演習の進め方	26
5	JavaScript の基礎	27
5.1	変数とデータ型	27
5.1.1	基本的なデータ型	27
5.1.2	変数の初期状態	27
5.1.3	データ型の変換	27
5.1.3.1	文字列→整数: parseInt メソッド	27
5.1.3.2	文字列→浮動小数点数: parseFloat メソッド	28
5.1.3.3	数値→文字列: toString メソッド	28
5.1.4	数値計算	29
5.1.4.1	数学関数と定数	29
5.1.4.2	桁の大きな整数	30
5.1.5	値の比較	30
5.1.6	型の検査	30
5.1.7	文字列の表記	30
5.1.7.1	テンプレート文字列	31
5.1.8	文字列に対する操作	31
5.1.8.1	連結 (1): +	31
5.1.8.2	比較	31
5.1.8.3	長さ	31
5.1.8.4	部分の取り出し	31
5.1.8.5	分割	31
5.1.8.6	連結 (2): join メソッド	32
5.2	JavaScript のコメント記述	32
5.3	制御	33
5.3.1	条件分岐 (1): if 文	33
5.3.1.1	条件判定のための 3 項演算子	33
5.3.2	条件分岐 (2): switch 文	33
5.3.3	繰り返し (1): while 文	34
5.3.4	繰り返し (2): for 文	34
5.3.4.1	繰り返し処理の中断	35
5.3.4.2	繰り返し処理のスキップ	35

5.4	関数	35
5.4.1	関数内でのローカル変数	37
5.4.2	関数の高度な応用 (1): 無名関数	38
5.4.3	アロー関数	38
5.4.4	引数の扱い	39
5.4.4.1	可変長の引数を受け取る方法	39
5.4.4.2	デフォルト引数	40
5.4.5	関数の高度な応用 (2): apply, call	40
5.5	変数のスコープについて	40
5.5.1	変数宣言の巻き上げ (hoisting)	41
5.5.2	let 宣言, const 宣言	42
5.6	データ構造	44
5.6.1	配列	44
5.6.1.1	複数の配列の連結	45
5.6.1.2	配列要素の連結	45
5.6.1.3	配列による FILO (スタック) の実現	45
5.6.1.4	配列要素の順序の逆転	45
5.6.1.5	配列かどうかの判定	46
5.6.1.6	配列の途中の要素の削除	46
5.6.1.7	配列の部分列の抽出	46
5.6.1.8	条件による要素の抽出	46
5.6.1.9	条件による要素の探索	46
5.6.1.10	全要素に対する一斉処理	47
5.6.1.11	全要素に対する順次処理	47
5.6.1.12	要素に対する累積的处理	47
5.6.1.13	要素の並べ替え (ソート)	48
5.6.1.14	全ての要素に対する繰り返し処理の実現	49
5.6.1.15	全ての要素に対する判定: (全ての~, 1 つでも~)	49
5.6.2	オブジェクト (連想配列 1)	50
5.6.2.1	プロパティの表記について	50
5.6.2.2	プロパティの削除	50
5.6.2.3	オブジェクトの明示的宣言	50
5.6.2.4	全てのキーの取得	50
5.6.2.5	全てのキーに対する繰り返し処理の実現	51
5.6.2.6	オブジェクトを配列に変換する方法	52
5.6.3	Set オブジェクト	53
5.6.3.1	Set の作成	53
5.6.3.2	要素の追加	53
5.6.3.3	要素の削除	53
5.6.3.4	要素の存在検査	53
5.6.3.5	要素数の取得	53
5.6.3.6	全要素の消去	53
5.6.3.7	Set を配列に変換する方法	53
5.6.4	Map オブジェクト (連想配列 2)	54
5.6.4.1	Map オブジェクトの作成	54
5.6.4.2	エントリの参照と登録	54

5.6.4.3	エントリの確認	54
5.6.4.4	エントリの個数の調査	54
5.6.4.5	エントリの削除	55
5.6.4.6	全エントリの消去	55
5.6.4.7	キーの列, 値の列の取出し	55
5.6.4.8	全要素に対する順次処理	55
5.6.4.9	Map を配列に変換する方法	57
5.7	スプレッド構文	57
5.7.1	スプレッド構文の関数の仮引数への応用	57
5.7.2	スプレッド構文の分割代入への応用	58
5.7.2.1	オブジェクトの分割代入	58
5.8	日付・時刻・時間	59
5.8.1	年, 月, 日, 曜日, 時, 分, 秒を取得するメソッド	59
5.8.2	時間の扱い (再設定, 加算, 差分)	59
5.8.3	サンプルプログラム	60
5.8.4	経過時間の算出について	62
5.9	GUI の扱い方	62
5.9.1	GUI のコンポーネントの名前と値	63
5.9.1.1	要素の属性と値について	63
5.9.2	ボタンのクリック	64
5.9.3	テキストフィールド/パスワードフィールド/テキストエリアの内容	64
5.9.4	チェックボックスの状態	65
5.9.5	ラジオボタンの状態	65
5.9.6	選択リストの状態	65
5.9.7	スライダの値	65
5.9.8	更に簡単な方法	65
5.10	Image オブジェクト	67
5.11	DOM のオブジェクト階層	68
5.11.1	window オブジェクト	68
5.11.2	document オブジェクト	68
5.11.3	サンプルを用いた解説	68
5.11.3.1	子要素の取得: children プロパティ	69
5.11.3.2	HTML 要素の属性とテキスト	69
6	canvas グラフィックス	71
6.1	canvas とコンテキスト	71
6.1.1	コンテキスト	71
6.2	図形の描画	71
6.2.1	線と塗り	71
6.2.1.1	線の太さ, 色, ダッシュの設定	72
6.2.1.2	塗りつぶしの色の設定	72
6.2.2	円, 円弧, 楕円	72
6.2.3	四角形	74
6.2.4	文字列描画	74
6.2.4.1	フォント, スタイル, サイズの指定	74
6.3	画像ファイルの読み込みと表示	76

6.4	拡大縮小, 平行移動, 回転など	76
6.4.1	拡大縮小	76
6.4.2	平行移動	76
6.4.3	回転	76
6.4.4	コンテキストの保存と復元	77
6.5	ピクセル (画素) の操作	79
6.5.1	ImageData オブジェクトについて	79
6.5.2	ピクセル描画の手順	80
6.5.3	サンプルプログラム	81
7	Web アプリケーション開発に必要なこと	84
7.1	イベント駆動型プログラミング	84
7.1.1	イベントの種類	84
7.1.2	プログラムの記述と実行の流れ	85
7.1.2.1	イベントリスナーの設定	87
7.1.2.2	イベントオブジェクト	87
7.1.2.3	HTML 要素にイベントハンドリングを記述する方法	88
7.1.2.4	マウスの位置の取得	89
7.1.2.5	要素のサイズの取得	89
7.1.2.6	サンプル: イベントハンドリング登録のための各種の方法	91
7.2	メディアデータ (音声, 動画) の再生	94
7.2.1	audio/video タグを用いたメディアデータの読み込み	94
7.2.2	再生と一時停止	94
7.2.2.1	audio/video タグへのコントロールの追加	95
7.2.3	audio タグを使用せずに音声を再生する方法	95
7.2.3.1	音声ファイルの読み込み	95
7.2.4	再生の時刻に関すること	96
7.2.4.1	メディアデータ再生時に発生するイベント	96
7.2.5	再生の音量に関すること	96
7.2.6	再生の速度に関すること	97
7.3	タイミングイベント	97
7.3.1	タイミングイベントの繰り返し	98
7.3.1.1	setInterval	98
7.4	データの保存	99
7.4.1	値の保存	99
7.4.2	値の参照	99
7.4.3	値の消去	99
7.4.4	その他 (データの管理に関する機能など)	99
7.4.4.1	登録されているデータの個数を調べる方法	99
7.4.4.2	登録されているデータを順番に取り出す方法	100
7.4.5	localStorage の応用プログラム	100
7.5	表示環境の取得と設定	104
7.5.1	表示領域のサイズの取得	104
7.5.2	サーバとブラウザに関する情報の取得	105
7.6	JavaScript によるスタイルの設定	107
7.6.1	id 属性による HTML 要素の参照	107

7.6.2	スタイルの設定	107
7.6.2.1	位置属性の設定	107
7.6.2.2	可視属性の設定	108
7.7	要素の位置とサイズの取得	109
7.8	ファイルの読み込み	111
7.8.1	ファイル選択ダイアログ	112
7.8.2	FileReader	112
7.8.3	サンプルプログラム	113
7.9	ドラッグアンドドロップ	115
7.9.1	ドラッグアンドドロップに伴う処理	116
7.9.1.1	dragstart イベントを受けて行う処理	116
7.9.1.2	dragover イベントを受けて行う処理	117
7.9.1.3	drop イベントを受けて行う処理	117
7.9.1.4	イベントオブジェクトの target 属性	117
7.9.2	Web ブラウザ外部からドラッグアンドドロップを受け付ける処理	118
7.10	インラインフレーム (iframe)	120
8	DOM に基づくプログラミング	122
8.1	HTML の編集	122
8.1.1	基本的な編集機能	122
8.1.2	HTML の階層構造	123
8.1.2.1	要素の取得	123
8.2	SVG の描画	125
8.2.1	SVG の基本構造と HTML 文書内での配置	125
8.2.2	SVG の座標系と長さ, 角度の単位	125
8.2.2.1	ビューポートとビューボックス	126
8.2.3	JavaScript による SVG の描画	128
8.2.3.1	XML の名前空間	128
8.2.4	SVG の代表的な図形要素	130
8.2.4.1	多角形・折れ線	130
8.2.4.2	塗りとストローク, 色の指定, 曲がり角	131
8.2.4.3	パス	133
8.2.4.4	矩形, 円, 楕円, 直線, 文字	136
8.2.5	SVG 図形要素のイベントハンドリング	137
8.2.6	SVG 要素のグループ化	138
8.2.7	参考: SVG 関連のライブラリ	139
9	オブジェクト指向プログラミング	140
9.1	クラスの定義	140
9.1.1	コンストラクタ	140
9.1.2	メソッド	140
9.2	応用例: SVG 描画クラスの実装	140
9.2.1	実装したクラスライブラリの解説	142
9.2.2	実装したクラスライブラリの使用例	142

10 正規表現	145
10.1 文字列探索（検索）： search メソッド	145
10.2 パターンマッチ： match メソッド	145
10.3 置換処理： replace メソッド	146
10.4 正規表現の記述方法	146
11 ライブラリ	147
11.1 jQuery	147
11.1.1 jQuery を使用するための準備	147
11.1.2 jQuery の基本的な使い方	147
11.1.2.1 CSS の要素へのアクセス	148
11.1.2.2 \$で関数を実行する方法	148
11.1.2.3 イベントハンドリングの登録	149
11.1.3 DOM の操作	151
11.1.3.1 子要素の追加	151
11.1.3.2 テキストの取得と設定	152
11.1.3.3 HTML の取得と設定	153
11.1.4 セレクタの扱い	154
11.1.4.1 n 番目の要素の選択 (:nth-child)	154
11.1.4.2 n 番目の要素の選択 (eq)	156
11.1.4.3 属性値で要素を選択	157
11.1.5 視覚効果と高度な GUI	158
11.1.5.1 要素を表示する／隠す	158
11.1.5.2 jQuery UI	159
11.1.6 その他の便利な機能	163
11.1.6.1 要素のインデックスの取得	163
11.1.6.2 文字列のトリミング	164
11.2 MathJax	166
A 付録	168
A.1 フォントの活用	168
A.1.1 フォントの入手について	168
A.1.2 利用法 1：フォントファイルの組込み	168
A.1.3 利用法 2：Web で公開されているフォントの利用	170
A.2 ダウンロード機能の実装例	174
A.3 ブラウザ付属の開発機能	176
A.3.1 ソースプログラムの自動整形	176
A.3.1.1 Firefox の場合	177
A.3.1.2 Google Chrome の場合	178
A.3.2 プログラムのステップ実行	179
A.3.2.1 Firefox の場合	179
A.3.2.2 Google Chrome の場合	179
A.3.3 プログラムのデバッグ	181
A.3.3.1 Google Chrome によるデバッグ作業の例	181
A.3.4 JavaScript の対話的な実行	184

1 JavaScript とは

JavaScript は Mozilla Foundation が仕様を策定し、ECMAScript (ECMA-262) として標準化されているプログラミング言語です。この言語によるプログラミングによって、Web ブラウザ上のコンテンツの表示や動作を制御することができます。JavaScript という名称から Oracle 社の Java 言語を連想することがありますが、それら 2 つは異なる別の言語です。ただし、それらの言語は互いに親和性が良く、連携することによって、高機能な Web アプリケーションを構築することが可能になります。実際に JavaScript と Java 言語は互いに文法上の共通点も多いです。

Web ブラウザ上のコンテンツは、基本的には HTML で記述された静的な（動きの無い）ものですが、JavaScript を用いることで動きを与えることができ、高機能な Web アプリケーションを構築することができます。

多くの場合、HTML コンテンツは「Web コンテンツ」として理解されることが多く、HTML や CSS も「文書構造」「ページの体裁」を記述する言語として認識されています。しかし JavaScript はプログラミング言語として十分な機能と性能を備えており、これを利用することで、かなり高機能なアプリケーションプログラムを構築することができます。従って、HTML や CSS に比べて JavaScript は奥が深く、学ぶための心構えを少しばかり強く持つことが要求されます。

本書は JavaScript の基本的な機能を紹介し、事例を示しながら初歩的な取り扱いの方法について解説するものです。

2 学習に必要なもの

JavaScript によるプログラム構築と実行のために特別なソフトウェア製品は必要ありません。JavaScript を実行できる Web ブラウザと、プログラムの編集のためのテキストエディタがあれば JavaScript の学習とプログラム開発が可能になります。JavaScript を実行することができる Web ブラウザとして代表的なものに、Google Chrome と Mozilla Firefox (図.2,3) があります。また、Microsoft Windows には標準的にメモ帳 (notepad.exe : 図.1) が付属しており、これをテキストエディタとして利用することができます。



図 1: メモ帳

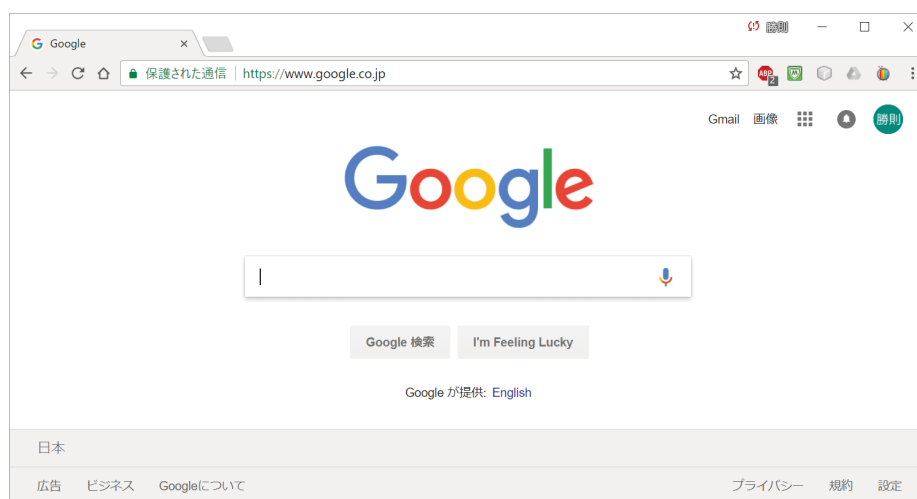


図 2: Google Chrome ブラウザ

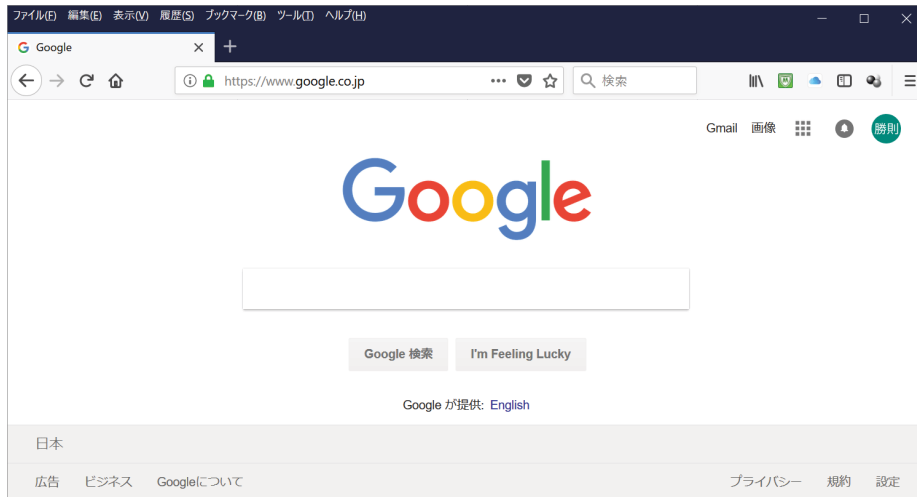


図 3: Mozilla Firefox ブラウザ

2.1 プログラミングに適したテキストエディタ

MS-Windows 用のテキストエディタとしてはメモ帳以外にも、「TeraPad」¹ や「サクラエディタ」², 「Notepad++」³ などがあります。また Apple の macOS 用のテキストエディタとして「CotEditor」⁴ や「mi」⁵ といったものがあります。

2.1.1 クロスプラットフォームに対応したテキストエディタ

Electron⁶ ベースのエディタとして「Atom」⁷ や「Visual Studio Code」⁸ (図 4) などがあります。



図 4: Electron ベースのエディタ

この他にも「Brackets」⁹ (図 5) のようなものもあります。



図 5: Brackets

ここに挙げたクロスプラットフォーム対応のエディタは、各種言語の語をハイライトするなど、便利な機能が備わっています。

¹<http://www5f.biglobe.ne.jp/~t-susumu/>

²<http://sakura-editor.sourceforge.net/>

³<http://notepad-plus-plus.org/>

⁴<http://coteditor.github.io/>

⁵<http://www.mimikaki.net/>

⁶Chromium と Node.js をベースにしたオープンソースのソフトウェアフレームワーク。

⁷<https://atom.io/>

⁸<https://code.visualstudio.com/>

⁹<http://brackets.io/>

3 JavaScript を学ぶ前に

JavaScript は Web ブラウザの動作を制御して、Web コンテンツに動きを与えます。このため、JavaScript で記述されたプログラムは、基本的には対象となる HTML コンテンツに埋め込んで実行します。したがって JavaScript の学習は HTML の基礎知識が前提となります。学習の便宜のために、ここでは HTML に関する基礎事項について説明しますが、HTML に関する解説が不要な読者はこの章を読み飛ばしてください。もちろん、本書は HTML の全てについて言及するものではありません。

3.1 HTML の基本

3.1.1 基本構造

HTML のコンテンツは **html タグ** `<html>~</html>` で囲む形で記述します。更にこれらのタグの内部には **head タグ** `<head>~</head>` で囲む部分と、**body タグ** `<body>~</body>` で囲む部分があり、前者はコンテンツに付随する情報を、後者にはコンテンツ本体を記述します。

HTML コンテンツの記述に先立って文書型宣言をしておくことが正しい形式であり、`<html>` タグを書く前に、文書の先頭行に

```
<!DOCTYPE html>
```

と記述しておきます。ただし、文書型宣言がなくても、ほとんどの Web ブラウザでは HTML コンテンツを正常に表示することができます。本書でも文書型宣言をせずにサンプルを提示することがしばしばあります。

HTML コンテンツの記述に、日本語をはじめとする多バイト文字コードを使用する場合は、`<head>` の配下に使用する文字コードの体系を指定しておく必要があります。指定方法は

```
<meta charset="文字コード名">
```

です。(表 1 参照)

表 1: 文字コード

文字コード体系	指定する文字コード名
シフト JIS (Shift-JIS)	"x-sjis", "Shift_JIS", "shift_jis"
JIS	iso-2022-jp
EUC	"x-euc-jp", "EUC-JP", "euc-jp"
UTF-8	"UTF-8", "utf-8"

日本語のコンテンツを作成する場合、文字コード体系は utf-8 が標準です。次に、標準的な日本語 HTML コンテンツの例を示します。

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="utf-8">
    <title>タイトル</title>
  </head>
  <body>
    (コンテンツ本体の記述)
  </body>
</html>
```

UTF-8 コードで HTML コンテンツを記述する場合の例

html タグの中の `lang="ja"` という記述 (**lang 属性**) は、コンテンツが日本語であることを意味しますが、この記述をしなくても誤りではありません。ただし、**title タグ** は必須のもので、`<title>~</title>` の内容も空ではいけません。

3.1.1.1 改行要素

記述した HTML 要素は Web ブラウザ上では基本的に「上から下」「左から右」の方向に配置されます。このとき、HTML 文書中の改行は表示の際の改行とはならないことに注意してください。表示の際の改行は**改行要素**（‘
’ タグ）で行うことができます。

HTML 要素が行内に埋め込まれて表示されるか、強制的に改行されるかの特性については「3.2.8 ブロックレベル要素とインライン要素」（p.18）で説明します。また、要素の表示の位置を座標を指定して自由に決定する方法については「3.2.5.2 位置とサイズの設定」（p.14）で説明します。

3.1.1.2 段落要素

1 続きの文字列（テキスト）から成る段落は**段落要素** <p>～</p> で括ります。

3.1.2 GUI コンポーネント

HTML コンテンツは Web サーバ上の処理プログラム（CGI ¹⁰ など）と連携して、Web アプリケーションのユーザインターフェースとして使用されることがあります。HTML コンテンツ内に **form タグ**<form>～</form> で囲んだ記述を行うことで GUI ¹¹ を構築し、送信ボタンのクリック操作によって、このタグで囲んだ範囲のコンテンツの状態をサーバに送信することができます。内容を受け取ったサーバ側のプログラムは処理を実行します。

本書では、サーバ上のプログラムとの連携処理については言及せず、ローカル（手元のコンピュータ）の環境で JavaScript のプログラムを実行することに焦点を絞って解説します。したがって、GUI コンポーネントを設置するタグも、<form>～</form>で囲むことなく記述する形で解説することが多くなります。

3.1.2.1 ボタン

HTML コンテンツ内にボタンを配置するには <button>～</button> もしくは <input type="button" …> を記述します。例えば「送信」という表示のボタンを配置するには、

「<button>送信</button>」 もしくは 「<input type="button" value="送信">」

と記述します。これにより図 6 のようなボタンがコンテンツ内に配置されます。

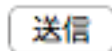


図 6: ボタン

<button> タグで括ると、画像など文字以外の対象もボタンとして扱うことができます。

3.1.2.2 チェックボックス

HTML コンテンツ内にチェックボックスを配置するには<input type="checkbox">を記述します。例えば

「<input type="checkbox">メールマガジンを希望する」

と記述すると図 7 のようなチェックボックスがコンテンツ内に配置されます。

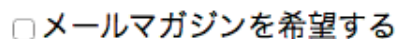


図 7: チェックボックス

3.1.2.3 ラジオボタン

HTML コンテンツ内にラジオボタンを配置するには<input type="radio"> を記述します。ラジオボタンは複数の選択肢の中の 1 つをチェック（選択）する仕組みです。これを実現するためには、複数のラジオボタンをある**グルー**

¹⁰CGI (Common Gateway Interface) : Web サーバ上でプログラムを動作させる仕組み。直接ユーザが操作する部分（インターフェース）をクライアントシステム側で実装し、具体的な処理を Web サーバ側で実現する際によく用いられる。

¹¹GUI (Graphical User Interface) : ディスプレイに表示される可視的な対象を、マウスなどのポインティングデバイスやキーボードを用いて操作するインターフェース

ブでまとめ、その中の1つだけがチェックされるという形で記述します。

例えば☐と記述します。次の記述例を見てください。

「性別:<input type="radio" name="r1">男性,<input type="radio" name="r1" checked>女性」

グループの名前としては好きなものを指定することができ、この例では"r1"としています。これをHTMLコンテンツ内に記述すると図8のようなラジオボタンがコンテンツ内に配置されます。

性別： ☐ 男性, ☒ 女性

図 8: ラジオボタン

この例は、「女性」のボタンが予めチェックされているもので、当該<input>タグ内に 'checked' が記述されています。

3.1.2.4 テキストフィールド

HTMLコンテンツ内にテキストフィールドを配置するには

<input type="text" size="文字数" value="内容の文字列">

を記述します。例えば

「ユーザ名： <input type="text" size="20" value="値の設定">」

と記述すると図9のようなフィールドがコンテンツ内に配置されます。

ユーザ名：

図 9: テキストフィールド

3.1.2.5 パスワードフィールド

HTMLコンテンツ内にパスワードフィールドを配置するには<input type="password" size="文字数"> を記述します。例えば

「パスワード:<input type="password" size="20">」

と記述すると図10のようなフィールドがコンテンツ内に配置されます。

パスワード：

図 10: パスワードフィールド

パスワードフィールドの内容は秘匿され、ここに文字を入力しても「●」のようなダミー文字が表示されます

3.1.2.6 テキストエリア

HTMLコンテンツ内にテキストエリアを配置するには<textarea cols="桁数" rows="行数" >を記述します。例えば

<textarea cols="38" rows="5" ></textarea>

と記述すると図11のように38桁×5行の入力エリアがコンテンツ内に配置されます。



図 11: テキストエリア

<textarea>～</textarea> で文字列を括ると、その文字列がテキストエリアの初期値として予め表示されます。

3.1.2.7 選択リスト

HTML コンテンツ内に選択リストを配置するには`<select><option>…</option>…</select>` を記述します。例えば

```
年齢：
<select>
  <option value="20">20代</option>
  <option value="30">30代</option>
  <option value="40">40代</option>
  <option value="50">50代</option>
</select>
```

と記述すると図 12 のような選択リストがコンテンツ内に配置されます。

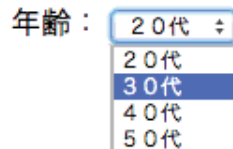


図 12: 選択リスト

選択リストの初期値を設定するには、次の例の様に option タグの中に「selected」を記述します。

例. `<option value="40" selected>40代</option>`

3.1.2.8 スライダー

HTML コンテンツ内にスライダーを配置するには`<input type="range">` を記述します。これにより図 13 のようなスライダーがコンテンツ内に配置されます。



図 13: スライダー

スライダーの値（value 属性）は標準的には 0～100 の範囲ですが、input タグの属性として min, max を設定することで下限値と上限値を与えることができます。また value 属性でスライダーの初期位置（初期値）を設定できます。step 属性を与えると value 属性の値の刻み幅（つまみの最小移動量）が設定できます。

【サンプル】

次のような HTML コンテンツ "test1.html" を Web ブラウザで表示すると図 14 のようになります。

サンプル：test1.html

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>GUIのテスト </title>
6    </head>
7    <body>
8      <input type="checkbox" checked>ポイントを受け取る<br>
9      <input type="checkbox">メールマガジンを希望する<br>
10     性別：<input type="radio" name="r1">男性，
11     <input type="radio" name="r1" checked>女性<br>
12     年齢：
13     <select>
14       <option value="20">20代</option>
15       <option value="30">30代</option>
16       <option value="40">40代</option>
17       <option value="50">50代</option>
18     </select>
19     <br>
20     ユーザ名： <input type="text" size="20" value="値の設定"><br>
```



```

21      パスワード:<input type="password" size="20"><br>
22      よろしければメッセージを入力してください:<br>
23      <textarea cols="38" rows="5" ></textarea>
24      <br>
25      <button>送信</button>
26  </body>
27 </html>

```

図 14: Web ブラウザによる test1.html の表示

このようにして HTML コンテンツとして GUI を構築します。GUI の操作に応じて具体的に動作を起こすプログラムを JavaScript で記述することになります。

3.1.2.9 メータ, 進捗バー

HTML コンテンツ内にメータを配置するには `<meter>~</meter>` を記述します。これにより図 15 のようなメータがコンテンツ内に配置されます。



図 15: メータ

meter 要素は値の大きさを棒グラフのように表示するもので、値は value 属性に与えます。値の最小値は 0、最大値は 1.0 ですが、それらを min 属性、max 属性で設定することができます。

HTML コンテンツ内に進捗バーを配置するには `<progress>~</progress>` を記述します。これにより図 16 のような進捗バーがコンテンツ内に配置されます。



図 16: 進捗バー

進捗バーはメータと似たものですが、その名の通り、進捗の様子が左から右にかけて走るような視覚表現が加えられています。進捗バーの値は value 属性に与えます。値の最小値は 0、最大値は 1.0 ですが、最大値のみ max 属性で設定することができます。

3.1.3 HTML のコメント記述

HTML 要素として認識しない内容（コメント）を HTML 中に記述するには

```
<!-- コメント -->
```

のように“<!--”と“-->”で括ります。コメントは複数行にまたがっても構いません。

3.1.4 HTML タグの誤りを検査する方法

W3C¹² が提供する Markup Validation Service (<https://validator.w3.org/>) を利用すると，作成した HTML コンテンツの検査ができます。

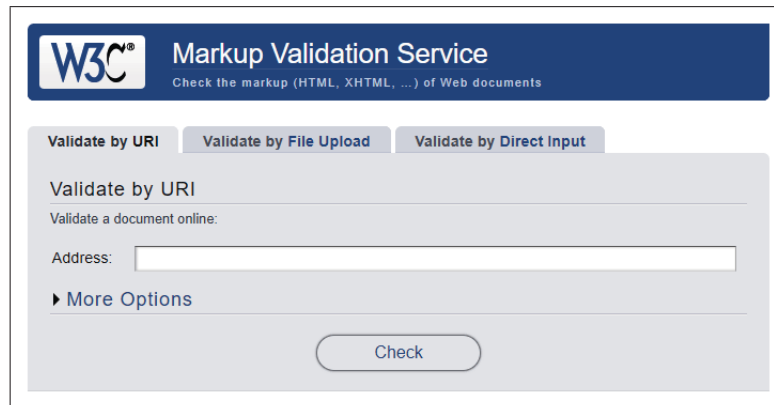
The image shows the W3C Markup Validation Service web interface. At the top, there is a dark blue header with the W3C logo and the text "Markup Validation Service" and "Check the markup (HTML, XHTML, ...) of Web documents". Below the header, there are three tabs: "Validate by URI" (selected), "Validate by File Upload", and "Validate by Direct Input". Under the "Validate by URI" tab, there is a section titled "Validate by URI" with the text "Validate a document online:". Below this, there is a label "Address:" followed by a text input field. Under the input field, there is a link "More Options". At the bottom of the form, there is a "Check" button.

図 17: W3C Markup Validation Service

¹²World Wide Web Consortium：WWW の標準化団体

3.2 CSSの基本

CSS (Cascading Style Sheets) は HTML コンテンツの体裁 (スタイル) を制御するためのものであり、コンテンツの装飾のための基本です。Web ブラウザは CSS の記述にしたがってコンテンツの体裁を調整します。CSS による体裁の記述は HTML の文書内容とは別のも であり、HTML コンテンツの `<head>~</head>` 内に記述¹³ します。

本書では CSS についての最低限の事項について解説します。また CSS に関する解説が不要な読者はこの節を読み飛ばしてください。

3.2.1 基礎事項

スタイルの記述は HTML コンテンツの `<head>…</head>` 内に `<style>~</style>` を記述し、この「~」の部分に具体的なスタイルを記述します。

スタイルの記述は「HTML コンテンツのどのタグにどんな設定を施すか」という考え方を基本にしています。例えば

```
<style>
  p{~}
</style>
```

という記述をすると、HTML コンテンツ内の `<p>` タグの内容に対して「~」で記述した設定を施します。ここ場合、HTML コンテンツ内の全ての `<p>` タグの内容に対して同じ設定が施されるということが重要です。もちろん特定のタグに対してスタイルを設定することもできます。その場合、HTML コンテンツ内のスタイルを設定したい対象のタグに識別子 (**id 属性**) を与え、スタイルの記述をその id 属性を持つタグに対して施すものとします。すなわち、

「`<p id="abc">この部分にスタイルを設定したい</p>`」

と記述すると、HTML コンテンツ内のこの部分に `"abc"` という id 属性が与えられ、

```
<style>
  p#abc{~}          (← p を省略して #abc としても良い)
</style>
```

とスタイルを記述することで、`"abc"` の id 属性を持つ部分にのみスタイル設定が施されます。

上の例では `p#abc` を `#abc` と記述することもでき、id 属性をスタイル設定の対象とする場合は接頭辞 `#` を付けます。また、CSS の設定対象として `{ ~ }` の前に記述するものを**セレクトア**と呼びます。

複数のタグをまとめて、それらタグの集合に共通のスタイルを設定する場合は、まとめたい複数のタグの **class 属性** に共通の値を与え、その class に対してスタイルを記述します。例えば、

```
<p class="c">対象箇所 1</p>
<p class="c">対象箇所 2</p>
<p class="c">対象箇所 3</p>
```

と記述すると、上記 3 つの行全ての class 属性に共通の値 `"c"` が与えられます。これにスタイルを設定するには次のように CSS を記述します。

```
<style>
  .c{~}
</style>
```

この例のように、対象 class の名前 (class の属性値) をセレクトアとする場合は、接頭辞としてドット `.` を付けます。

3.2.1.1 セレクトアの書き方に関して

複数の要素に対して同じスタイルを設定するには、対象のセレクトア群をコンマ `,` で区切って列挙します。また、ある要素の子要素に対して CSS を設定する際は、要素の上下関係 (親子関係) をスペースで区切って書き並べることができます。

¹³HTML タグの `style` 属性 (`style="スタイル記述"`) としてスタイル設定を施すことも可能です。

例. 内の に対するスタイル設定

```
ul li { スタイル設定の記述 }
```

, に関しては「3.3 表と箇条書き」(p.20) で説明します.

次に, '{ ~ }' の部分にどのような形でスタイルの設定を記述するかについて説明します.

3.2.2 基本的なスタイル要素

3.2.2.1 フォントの設定 (スタイル, サイズなど)

記述: `font-family`: フォント名

この「フォント名」の部分に具体的なフォントの名前¹⁴を与えます. 特定のフォント名を与えない場合は, 次のようなフォント名を与えてください.

<code>sans-serif</code>	(ゴシック体)
<code>serif</code>	(明朝体)
<code>monospace</code>	(等幅ゴシック体)

記述: `font-size`: フォントサイズの値

この「フォントサイズの値」の部分に大きさを指定します. 大きさの指定には数値とその後ろに次のような単位を記述します.

<code>em</code>	標準の文字サイズに対する倍率
<code>%</code>	標準の文字サイズに対する倍率 (百分率)
<code>pt</code>	ポイント (1/72 インチ)
<code>px</code>	当該デバイスにおけるピクセル数
<code>mm, cm</code>	ミリメートル, センチメートル
<code>pc</code>	パイカ (1pc は 12 ポイント)

注) これら**大きさの単位**は色々な所で応用できます.

記述: `font-style`: フォントスタイル

この「フォントスタイル」の部分に次のようなスタイルを指定します.

<code>normal</code>	通常スタイル (デフォルト値)
<code>italic</code>	イタリック
<code>oblique</code>	斜体 (厳密にはイタリックとは違う)

記述: `font-weight`: フォントの強さ

この「フォントの強さ」の部分には 100~900 までの整数値を 100 刻みで指定¹⁵します.

CSS で文字の体裁を設定する例 `CSSTest1.html` を次に示します.

サンプル: `CSSTest1.html`

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>CSSのテスト </title>
6     <style>
7       p#s1{
8         font-family:sans-serif;
9       }
10      p#s2{
11        font-family:serif;
12      }
13      p#s3{
14        font-size:24pt;
```

¹⁴フォントの扱いについて詳しくは「A.1 フォントの活用」(p.168) で説明します.

¹⁵ただし, 扱うフォントの種類によっては指定した値が有効にならない場合もあるので, フォントの種類毎に試してください.

```

15         font-weight:900;
16         font-style:italic;
17     }
18     </style>
19 </head>
20 <body>
21     <p>文字列の一部分の体裁をCSSで設定する例. </p>
22     <p id="s1">ゴシック体：あいうアイウabcABC</p>
23     <p id="s2">明朝体：あいうアイウabcABC</p>
24     <p id="s3">24ポイント（強さ900でイタリック）</p>
25 </body>
26 </html>

```

この例では1種類のタグ(<p>)に複数のスタイルを設定しています。それぞれのスタイル設定の最後にセミicolon";"を付けます。

CSStest1.html を Web ブラウザで表示した例を図 18 に示します。



図 18: CSStest1.html を Web ブラウザで表示したところ

より進んだフォントの活用方法については p.168 「A.1 フォントの活用」を参照してください。

3.2.2.2 行の高さと文字間の設定

記述: line-height:行の高さ

例. 1 行の高さを 54px にする

```
line-height: 54px;
```

記述: letter-spacing:文字間

例. 文字間を 9px にする

```
letter-spacing: 9px;
```

3.2.2.3 段落開始のインデント

記述: text-indent:インデント幅

例. 段落開始のインデントを 24px にする

```
text-indent: 24px;
```

3.2.2.4 色の設定

記述: color:#16進RGB値 もしくは color:色名

例. 緑の指定

```
color: #00ff00;                   もしくは   color: green;
```

RGB 値は光の三原色（赤，緑，青）を合成する値で，各色 0～255 の範囲の整数で表されます．それらの値を 16 進数 2 桁で表現したものを連結して color に設定します．参考として，表 2 に 0～15 までの 10 進数と 16 進数との対応を挙げます．

表 2: 0～15 までの 10 進数と 16 進数との対応

10 進数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16 進数	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

これら 0～F までの 16 個の記号を使って各色の強度の数値を表現します．具体的には 0～255 の値を次のように 16 進数に対応させますが，基数の表現に関する解説は本書では割愛します．

10 進数	0	1	2	...	9	10	11	...	14	15	16	...	240	241	...	255
16 進数	00	01	02	...	09	0A	0B	...	0E	0F	10	...	F0	F1	...	FF

3.2.3 ハイパーリンクのスタイル

通常，<a>～ によるコンテンツのリンク（ハイパーリンク）には下線が付きます．また，リンクには未参照，参照済み，ポイント中，選択中といった状態があり，それぞれ異なったスタイル（色など）で表示されます．このそれぞれの状態（下記参照）に対して個別にスタイルを設定することができます．

記述	状態	記述	状態
a:link	未参照.	a:visited	参照済み.
a:hover	ポイント中の状態.	a:active	選択中の状態.

また，リンクには通常下線が付きますが，下線が表示されない設定も可能（下記参照）です．

要素	解説
text-decoration	none を指定するとリンクの下線が表示されない．

3.2.4 HTML 要素の不透明度（opacity）

HTML 要素に不透明度を設定するスタイル要素 opacity があります．opacity には 0～1.0 の数値を与え，0 が透明，1.0 が不透明になります．例えば，

```
<p>前面の文字</p>
```

という <p> 要素（color は red）に様々な不透明度を与えた様子を図 19 に示します．



図 19: 様々な opacity（不透明度）の値の設定の例

これは <p>背景の文字</p> を背景にして不透明度を設定した例です．

3.2.5 コンテンツの配置に関する基礎

JavaScript で Web アプリケーションを作る場合，GUI は HTML コンテンツとして作ることになります．この場合，div タグを一種のコンテナとして用い，それを階層的に構築することで GUI を構成するのが一般的です．例えば図 20 のように，ヘッダー部分，目次リンク，内容，フッター部分をラッパー（全体の容れ物）で包み込んで GUI を構成する例が多く見られます．

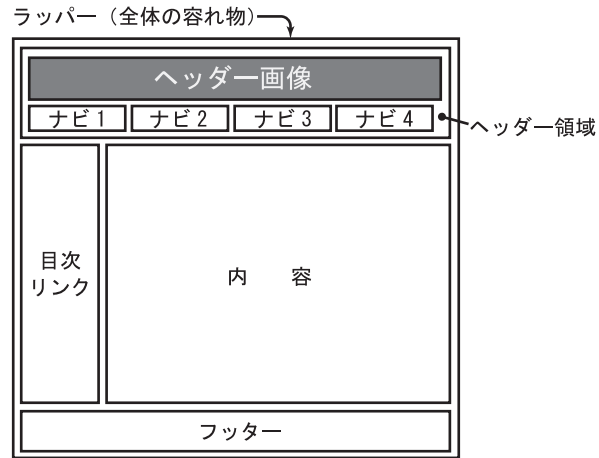


図 20: コンテンツフレームの例

ここでは、div タグを用いて階層的に GUI を構築するための初歩的な方法について説明します。

3.2.5.1 div 要素

とりあえず div タグを記述してみます。次のサンプル GUIframe01.html には 2 つの div タグが記述されています。

サンプル：GUIframe01.html

```

1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>div要素のテスト </title>
5 </head>
6 <body>
7 <div id="div1">内容1</div>
8 <div id="div2">内容2</div>
9 </body>
10 </html>

```

これを Web ブラウザで表示すると図 21 のような表示になります。

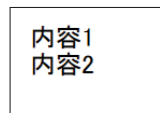


図 21: Web ブラウザによる表示

このように、div タグを並べて記述すると、画面上では上から下に向けて配置されます。次に、CSS を用いてこれら div タグに枠 (border) を付けてみます。(GUIframe02.html)

サンプル：GUIframe02.html

```

1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>div要素のテスト </title>
5 <style>
6 div {
7     border: solid 1pt;
8 }
9 </style>
10 </head>
11 <body>
12 <div id="div1">内容1</div>
13 <div id="div2">内容2</div>
14 </body>
15 </html>

```

これを Web ブラウザで表示すると図 22 のような表示になります。

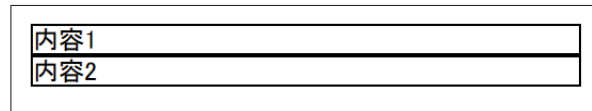


図 22: Web ブラウザによる表示

このように、div タグによるオブジェクトはウィンドウ幅いっぱいに広がっていることがわかります。div 要素の位置やサイズなどの設定に関するスタイル要素を次に説明します。

3.2.5.2 位置とサイズの設定

記述： position: 位置の指定方法

「位置の指定方法」の部分には次のようなものが指定できます。

- | | |
|----------|------------------------------------|
| absolute | 上位要素の左上を原点とする。最上位の要素はブラウザの描画領域である。 |
| fixed | ウィンドウの左上を原点とする。（スクロールに関係なく不動） |
| relative | 直前の要素からの相対的な位置を指定する。 |

記述： top: 原点からの縦の距離

記述： left: 原点からの横の距離

記述： width: 横のサイズ

記述： height: 縦のサイズ

記述： float: left / right (横方向の回り込み指定)

複数の div 要素を並べて記述すると上下方向に配置されますが、float を指定することで、左右に回りこむ配置にすることができます。

記述： overflow: ウィンドウからはみ出したコンテンツの表示方法

コンテンツが Web ブラウザのウィンドウよりも大きい場合の表示方法を設定します。hidden を指定すると、Web ブラウザのウィンドウが小さくても表示は崩れません。

距離とサイズの指定に関しては、前述の「フォントサイズの値」を参考にしてください。

3.2.5.3 枠線と余白

対象のオブジェクトに対して次のような属性を設定することができます。

- | | |
|------------------|--|
| border-style | 枠線のスタイル |
| | 次のようなものが指定できます。(代表的な値) |
| | solid → 実線, double → 二重線, dashed → 破線, dotted → 点線 |
| border-width | 枠線の太さ |
| border-color | 枠線の色 |
| margin | 枠外マージン |
| padding | 枠内マージン |
| background-color | 背景色 |

参考)

枠線の属性(スタイル, 太さ, 色)は、まとめて border に設定することができます。次のサンプル GUIframe03.html を見てください。

サンプル：GUIframe03.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>div要素のテスト </title>
5 <style>
```

```

6  div {
7    border: solid 4pt #ff0000;
8    width: 80pt;
9    height: 14pt;
10   float: left;
11   margin: 4pt;
12   padding: 8pt;
13   background-color: yellow;
14   text-align: center;
15 }
16 </style>
17 </head>
18 <body>
19 <div id="div1">内容1</div>
20 <div id="div2">内容2</div>
21 </body>
22 </html>

```

これを Web ブラウザで表示すると図 23 のような表示になります。

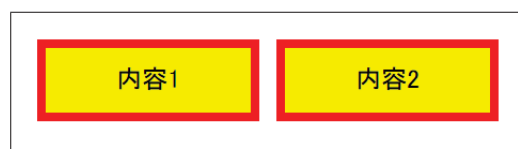


図 23: Web ブラウザによる表示

参考)

スタイル要素 `border`, `margin`, `padding` には接尾辞 `-top`, `-bottom`, `-left`, `-right` を付けて値を設定することもでき、上下左右の部分に指定したスタイル設定ができます。これにより、例えば「上の線のみ」や「左の余白のみ」といった特定の部分にスタイル設定することができます。

3.2.5.4 枠線の太さ、余白の大きさ、表示の位置、サイズの関係

HTML 要素は文書を構成するためのものであり、その意味では、枠の描画に関しても、他の多くのグラフィックス作成のためのアプリケーションとは前提が異なります。例えば `<div>` 要素による枠の表示においても、位置やサイズの指定方法が独特のものになっています。

`<div>` の枠線の太さ、表示位置、サイズとスタイル要素との関係を図 24 に示します。

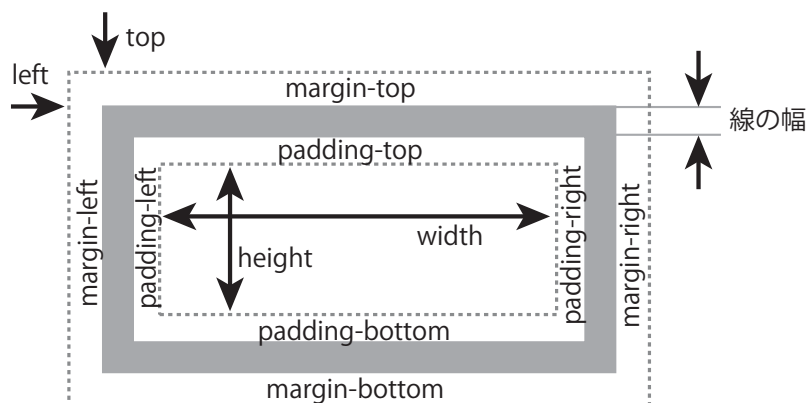


図 24: 要素の位置とサイズ

このように、`width` と `height` は要素内のコンテンツを収納する領域のサイズを意味します。また、`width` と `height` の解釈を枠線の外側に変更するには、当該 `<div>` 要素にスタイル `box-sizing: border-box` を設定します。(暗黙値は `box-sizing: content-box`)

このように、スタイルの各種要素の値を反映した形で枠の表示位置やサイズが決定されます。この考えに慣れておかなければ `<div>` で枠を表示する際に、位置やサイズが意図しない形になる印象を受けることがあるので注意が必要です。

3.2.5.5 GUI レイアウトの例

図 20 のような GUI のフレームを構築するサンプル (GUIframe04.html) について考えます。

サンプル：GUIframe04.html

```
1 <html>
2 <head>
3
4 <!-- 文字コードの設定とタイトル -->
5 <meta charset="utf-8">
6 <title>div要素のテスト </title>
7
8 <!-- CSSの記述 -->
9 <style>
10 #wrapper {
11     border: solid 1pt #999999;
12     width: 400pt;
13     height: 300pt;
14     margin: 2pt;
15     background-color: white;
16 }
17 #header {
18     margin: 2pt;
19     height: 80pt;
20     border: solid 1pt black;
21 }
22 #hdimage {
23     margin: 2pt;
24     height: 45pt;
25     border: solid 1pt black;
26     font-size: 44pt;
27     text-align: center;
28 }
29 .navi {
30     margin: 2pt;
31     float: left;
32     width: 90pt;
33     height: 20pt;
34     border: solid 1pt black;
35     font-size: 18pt;
36     text-align: center;
37 }
38 #toc {
39     float: left;
40     width: 80pt;
41     height: 150pt;
42     margin: 2pt;
43     border: solid 1pt black;
44     text-align: center;
45 }
46 #container {
47     float: left;
48     width: 308pt;
49     height: 150pt;
50     margin: 2pt;
51     border: solid 1pt black;
52     text-align: center;
53 }
54 #footer {
55     float: left;
56     width: 394pt;
57     height: 52pt;
58     margin: 2pt;
59     border: solid 1pt black;
60     text-align: right;
61 }
62 </style>
63 </head>
64
```

```

65 <!-- 表示内容 -->
66 <body>
67 <div id="wrapper">
68   <div id="header">
69     <div id="hdimage">ヘッダー画像</div>
70     <div class="navi" id="navi1">ナビ1</div>
71     <div class="navi" id="navi2">ナビ2</div>
72     <div class="navi" id="navi3">ナビ3</div>
73     <div class="navi" id="navi4">ナビ4</div>
74   </div>
75   <div id="toc">目次領域</div>
76   <div id="container">内容</div>
77   <div id="footer">フッター</div>
78 </div>
79 </body>
80 </html>

```

これを Web ブラウザで表示すると図 25 のような表示になります。

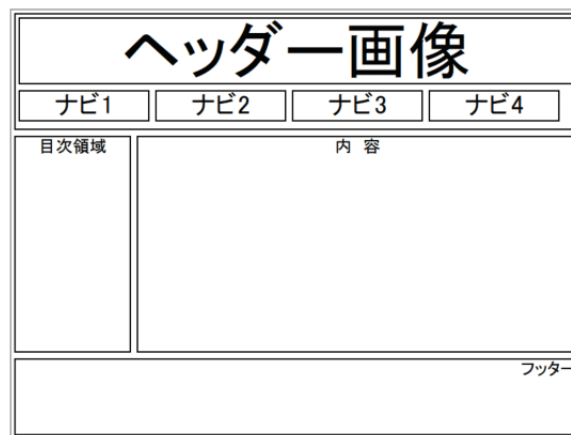


図 25: Web ブラウザによる表示

3.2.6 div 要素内でのテキストの配置の設定

テキストの左右の位置の設定はスタイル要素 `text-align` に設定します。この要素には値として `left`, `center`, `right` のいずれかを設定し、それぞれ左寄せ、中央揃え、右寄せの配置となります。また `div` 要素の高さと同じ値をスタイル要素 `line-height` に与えると上下方向の中央に配置されます。

これらスタイル要素を応用してテキストの配置を制御するサンプルを `GUIframe05.html` に示します。

サンプル：GUIframe05.html

```

1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>div要素の配置 </title>
5 <style>
6 .d {
7   /* 位置とサイズ, 枠 */
8   float:      left;
9   width:      110px;
10  height:     50px;
11  margin-left: 5px;
12  border:     1px solid black;
13  /* 文字の設定 */
14  font-size:  12px;
15 }
16 #d1 { text-align: left; }
17 #d2 { text-align: center; }
18 #d3 { text-align: right; }
19 #d5 { line-height: 50px; }
20

```

```

21 #d6 {    line-height: 50px;
22         text-align: center;    }
23 /* divの下に配置する */
24 #d7 {    line-height: 50px;    }
25 #p1 {    bottom: 0px          }
26 </style>
27 </head>
28 <body>
29 <div id="d1" class="d">左寄せテキスト</div>
30 <div id="d2" class="d">中央揃え</div>
31 <div id="d3" class="d">右寄せテキスト</div>
32 <div id="d4" class="d">設定なし</div>
33 <div id="d5" class="d">上下の中央</div>
34 <div id="d6" class="d">divの中央</div>
35 <div id="d7" class="d"><p id="p1">divの下</p></div>
36 </body>
37 </html>

```

これを Web ブラウザで表示すると図のような表示になります。

左寄せテキスト	中央揃え	右寄せテキスト	設定なし	上下の中央	divの中央	divの下
---------	------	---------	------	-------	--------	-------

図 26: Web ブラウザによる表示

3.2.6.1 div 要素内の下に子要素を配置する工夫

先のサンプルプログラム GUIframe05.html にあるように、div 要素の高さと同じ値をその div 要素のスタイル要素 `line-height` に与え、かつ、その子要素（例の中の<p>要素）のスタイル要素 `bottom` に `0px` を設定すると、div 要素の下に子要素を配置することができます。

3.2.7 HTML の span タグについて

HTML コンテンツ内の文字列など（行内の要素）に～ タグを施すと、その部分に対してスタイル設定ができます。ただしこのタグで指定した部分の位置の変更はできません。コンテンツ内で位置の変更をしたい場合は、対象部分を<div>～</div> で囲んで、<div>タグに対してスタイルを設定してください。

3.2.8 ブロックレベル要素とインライン要素

HTML 要素は**ブロックレベル要素**と**インライン要素**に分類することができます。ブロックレベル要素はその前後に改行が施されます。例えば<div>や<form>、<p>などは、内部に複数の要素を収めることができる構造物です。それに対してインライン要素は「行を構成するもの」であり、例えばや<input>などがあります。インライン要素は行の中に埋め込まれるものなので前後に改行は入りません。

HTML 要素がブロックレベルであるかインラインであるかはスタイル `display` に設定（下記）されており、意図的に変更することも可能です。

スタイル設定	解説
<code>display:inline</code>	対象の HTML 要素をインライン要素とする。
<code>display:block</code>	対象の HTML 要素をブロックレベル要素とする。

インライン要素のスタイル `vertical-align` の設定（下記）によって、当該要素の行内での上下の配置を設定することができます。

vertical-align の値	解説
top	行内の上の位置への配置
middle	行内の中央への配置
bottom	行内の下の位置への配置

3.2.9 CSS のコメント記述

CSS の要素として認識しない内容（コメント）を CSS 中に記述するには

```
/* コメント */
```

のように “/*” と “*/” で括ります。コメントは複数行にまたがっても構いません。

3.2.10 CSS を別のファイルに分離する方法

CSS は HTML コンテンツの<head>〜</head>内に記述しますが、CSS のみを別のファイルとして作成し、HTML コンテンツ内に読み込んで用いるという方法も一般的です。この場合は HTML コンテンツの<head>〜</head>内に

```
<link href="ファイル名.css" rel="stylesheet" type="text/css">
```

と記述して、“ファイル名.css” のファイルから CSS の内容を読み込むことができます。

3.2.11 CSS の誤りを検査する方法

W3C が提供する CSS Validation Service（図 27 <https://jigsaw.w3.org/css-validator/>）を利用すると、作成した HTML コンテンツの CSS の検査ができます。

図 27: W3C CSS Validation Service

3.3 表と箇条書き

表や箇条書きのための HTML 要素は Web アプリケーションの構築（GUI の構築）において重要な役割を果たします。それらについての基本的な事柄について説明します。

3.3.1 表： table 要素

縦横の格子状の表を作成するには `<table>...</table>` タグ（下記参照）を使います。

```
<table>
  <caption>表の見出し</caption>
  <tr> <th>列見出し 1</th> <th>列見出し 2</th> ... </tr>
  <tr> <td>列項目 1</td>   <td>列項目 2</td> ... </tr>
                                     :
</table>
```

表は表見出し（`<caption>...</caption>`タグ）、行（`<tr>...</tr>`タグ）、列見出し（`<th>...</th>`タグ）、列項目（`<td>...</td>`タグ）から構成されます。

3.3.1.1 表に関するスタイル

● 枠線： border

`<table>`、`<th>`、`<td>` の各要素が枠線（border）を持ちます。これら 3 種類のタグに枠線を設定した例を図 28 に示します。

キャプション

見出し 1	見出し 2
列項目 11	列項目 12
列項目 21	列項目 22

図 28: `<table>`で作成した表に枠線を施した例

`<th>`、`<td>` は 1 つのセルであり、個々のセルに枠線を施すことになります。また `<table>` の枠線は表の外周となります。暗黙では各セルの間には隙間（多くのブラウザでは 2 ピクセル程度）があります。

● セル間の隙間： border-collapse

セル間の隙間の有無は `<table>` のスタイル `border-collapse` で設定します。

`border-collapse: separate;` →隙間あり
`border-collapse: collapse;` →隙間なし

セル間の隙間が有るスタイル設定のもとで、`<table>` の `cellspacing` 属性に値を設定すると、セル間の隙間の大きさを変えることができます。

例. `<table cellspacing="5">` →隙間の大きさが 5 に設定される。

3.3.2 箇条書き： ul 要素, ol 要素

見出し記号や通し番号が付いた箇条書きを作成するには ... タグ や ... タグを使います。

見出し記号付きの箇条書き：

```
<ul>
  <li>箇条書き項目 1 </li>
  <li>箇条書き項目 2 </li>
  .
</ul>
```

通し番号付きの箇条書き：

```
<ol>
  <li>箇条書き項目 1 </li>
  <li>箇条書き項目 2 </li>
  .
</ol>
```

これらのタグを用いて箇条書きを実現した例を図 29 に示します。

による箇条書き

- 項目a
- 項目b
- 項目c

による箇条書き

1. 項目a
2. 項目b
3. 項目c

図 29: 箇条書きの例

3.3.2.1 箇条書きの見出しの設定

表 3 に示すようなスタイルの設定によって、箇条書きの見出しを変えることができます。

表 3: 箇条書きの見出し設定			
に対するスタイル	見出し	に対するスタイル	見出し
list-style:disc	●	list-style-type:decimal	1, 2, 3 ...
list-style-type:circle	○	list-style-type:lower-alpha	a, b, c ...
list-style-type:square	■	list-style-type:upper-alpha	A, B, C ...
list-style-type:disk	(数字)	list-style-type:lower-roman	i, ii, iii ...
list-style:none	(なし)	list-style-type:upper-roman	I, II, III ...
list-style-image:url(画像の URL)	(画像)		

3.3.2.2 箇条書き項目を横並びにする方法

 はブロックレベル要素なので、スタイル設定 display:inline を施すとインライン要素となり、箇条書き項目が横並びになります。

参考) Web アプリケーションを構築する際、箇条書きはメニューの作成に頻繁に使用されます。

3.4 画像データの配置

画像の配置には `` タグを使用します。

書き方: ``

‘alt=’ を省略しても多くの Web ブラウザは画像を正常に表示します。ただし、この属性に画像に関する説明を与えることで、対象の画像に文書としての付随情報を与えることができ、SEO の観点からも重要な意味を持ちます。また、何らかの事情で対象の画像を読み込むことができない場合は alt に設定された内容が表示されます。

画像データは Image オブジェクトとして扱うこともできます。これに関することは後の「5.10 Image オブジェクト」(p.67) で解説します。

3.4.1 画像の位置, サイズについて

`` はインライン要素なので上下位置はスタイル `vertical-align` で設定します。また縦横のサイズはスタイル `width`, `height` に設定します。

画像のスタイル `width`, `height` に設定した値の比率がその画像の元のアスペクト比（縦横の比率）と異なる場合は画像が変形して表示されます。(図 30)

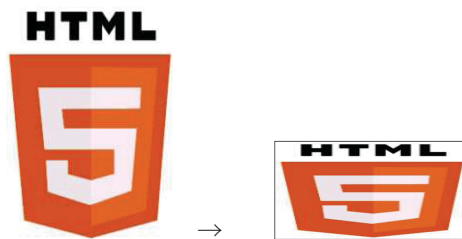


図 30: サイズ変更による画像の変形

`width`, `height` スタイルの設定によって画像が変形しないようにするには、当該画像のスタイル `object-fit` に `cover` や `contain` といった値を設定します。

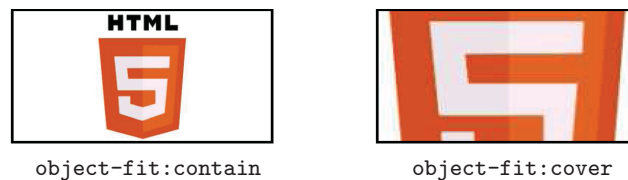


図 31: 画像のアスペクト比の維持

`object-fit:contain` と設定すると、指定した `width`, `height` の範囲内に画像全体が収まるように表示されます。また、`object-fit:cover` と設定すると、`width`, `height` の範囲を全て満たす形で画像が表示されます。すなわち、`object-fit:cover` はトリミングの処理です。

4 JavaScript を使う前に

4.1 まずは体験

HTML コンテンツとして設置されたボタンをクリックするとメッセージ ”ボタンが押されました” が表示される仕組みを JavaScript でつくる例を見てみましょう。まず次のようにボタンを設置します。

```
<button>押してください</button>
```

ボタンをクリックすると起動する JavaScript のプログラムを HTML コンテンツの<head>～</head>の中に記述¹⁶ します。JavaScript のプログラムは<script>～</script> で囲んで記述します。例えば次のように記述します。

```
<head>
  <meta charset="utf-8">
  <title>タイトル</title>
  <script>
    function f01( ) {
      window.alert("ボタンが押されました！");
    }
  </script>
</head>
```

このサンプルの中で "function f01()" という記述がありますが、これは "f01" というプログラム（関数）を定義するものです。定義内容はその後の {～} に記述します。プログラム中の

```
「window.alert("ボタンが押されました！")」
```

はウィンドウ上にメッセージボックスを表示させ、その中に ”ボタンが押されました” と表示させる文です。

これでボタンとプログラムが用意できました。次にこのボタンが押された（クリックされた）ときに JavaScript のプログラム f01 が起動するようにします。そのためには<button> を次のように変更します。

```
<button onClick="f01( )">押してください</button>
```

これでボタンとプログラムが関連付けられました。

この"onClick"が「ボタンが押された（クリックされた）」ことを意味するイベントです。イベントに関しては p.84 「7.1 イベント駆動型プログラミング」で詳しく説明します。

では出来上がったコンテンツ全体をファイル名 "test2.html" として保存して Web ブラウザで表示してみましょう。図 32 のような表示になります。

サンプル：test2.html

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>JSテスト </title>
5     <meta charset="utf-8">
6     <script>
7       function f01( ) {
8         window.alert("ボタンが押されました！");
9       }
10    </script>
11  </head>
12  <body>
13    <button onClick="f01( )">押してください</button>
14  </body>
15 </html>
```

¹⁶<body>...</body> 内に <script>～</script> を記述することもできます。

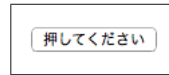


図 32: Web ブラウザによる test2.html の表示

Web ブラウザ上の「押してください」のボタンをクリックすると図.33 のような表示になります。



図 33: ボタンをクリックしたときの表示

この例のように、コンテンツの何らかの動作（今回はボタンのクリック）を起点として JavaScript で記述したプログラムが起動するというのが基本的なスタイルで、これを**イベント駆動方式**といいます。

4.1.1 JavaScript プログラムを別のファイルに分離する方法

JavaScript のプログラムを HTML コンテンツ本体のファイルとは別のファイルとして作成し、HTML コンテンツ内に読み込んで用いるという方法も一般的です。この場合は<script> タグを

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

と記述（src 属性）して、"ファイル名.js" のファイルから JavaScript のプログラムを読み込むことができます。

4.1.2 プログラム開発やデバッグのための機能

Mozilla Firefox や Google Chrome には、HTML,CSS,JavaScript に関するエラー検出などのための機能が備わっており、コンテンツ開発の利便性を提供しています。

【Mozilla Firefox の開発機能】

図 34 のように「ツール」→「ウェブ開発」→「開発ツールを表示」とメニュー選択をして開発機能の利用を開始します。

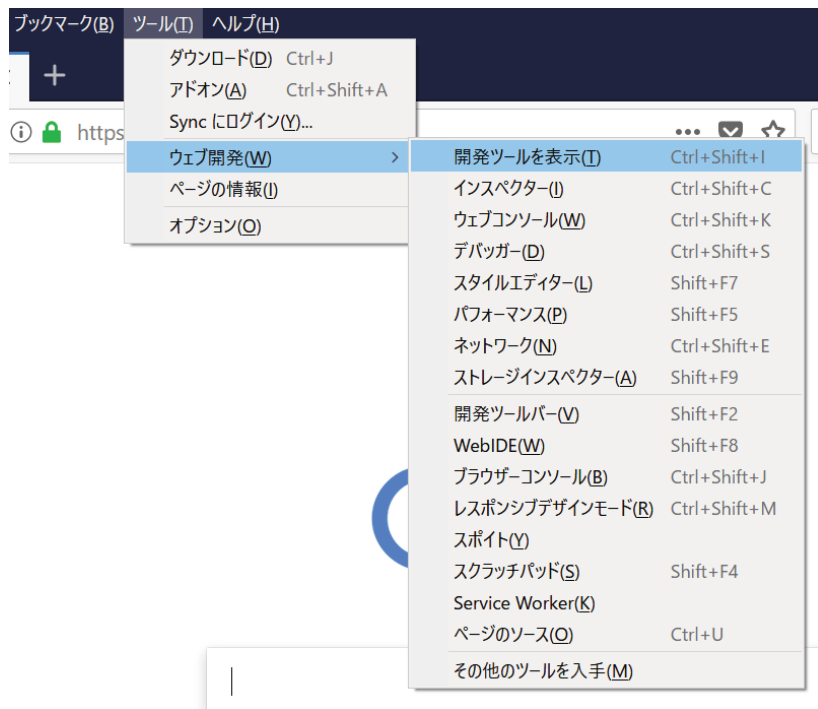
この操作により図 35 のようなウィンドウ表示になり、デバッグ機能などが利用できます。

【Google Chrome の開発機能】

図 36 のように、設定ボタン「⋮」から「その他のツール」→「デベロッパーツール」を選択して開発機能の利用を開始します。

この操作により図 37 のようなウィンドウ表示になり、デバッグ機能などが利用できます。

詳しくは p.176 「A.3 ブラウザ付属の開発機能」の所で説明します。



「ツール」→「ウェブ開発」→「開発ツールを表示」

図 34: 開発ツールの開始 (Mozilla Firefox の場合)

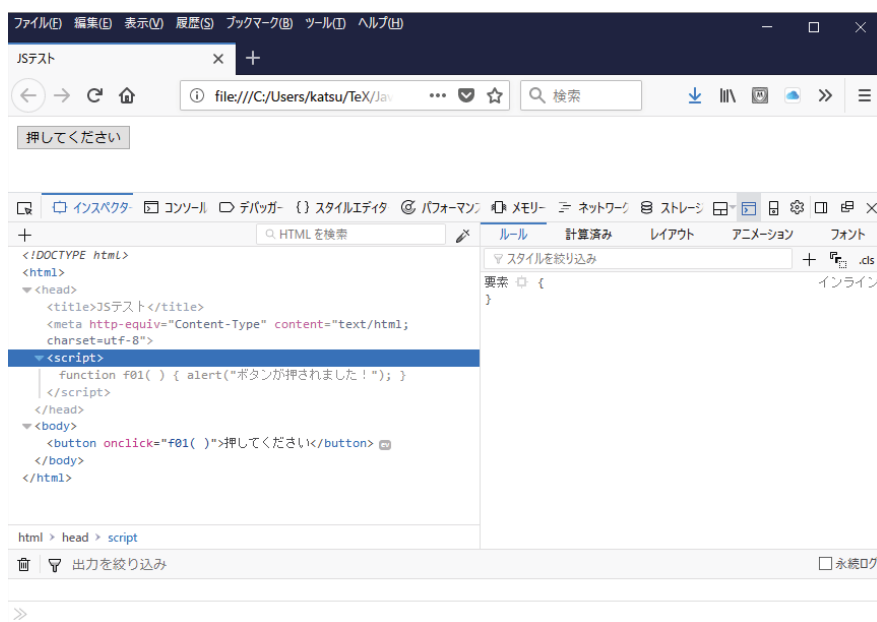


図 35: 開発ツールを表示したところ (Mozilla Firefox の場合)



設定ボタン「⚙」から「その他のツール」→「デベロッパーツール」を選択

図 36: 開発ツールの開始 (Google Chrome の場合)

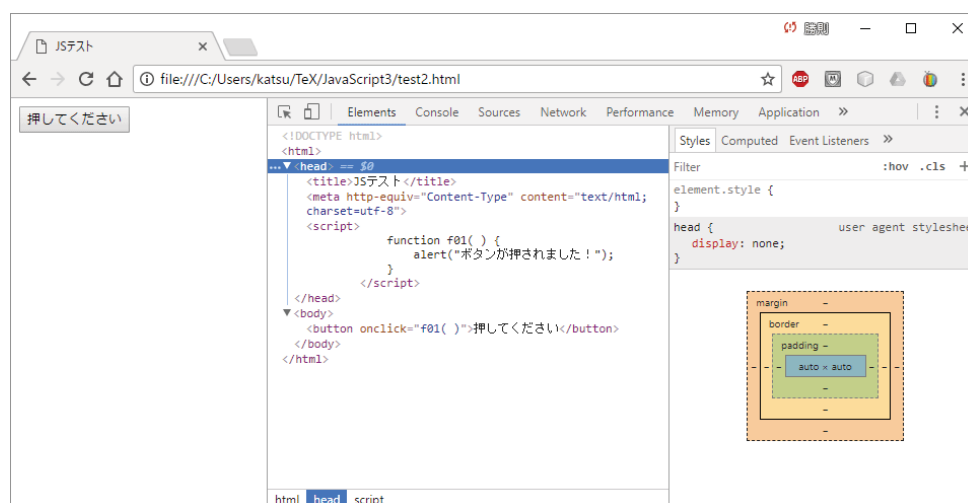


図 37: 開発ツールを表示したところ (Google Chrome の場合)

4.2 演習の進め方

JavaScript の有効な学習には演習が欠かせません。また、演習の形態は 2 つのスタイルを適宜使い分けることが重要です。1 つは、HTML5 のコンテンツをテキストエディタで作成して、その動作を実際に Web ブラウザで確認・検証するスタイルです。もう 1 つは、JavaScript の式や文を Web ブラウザで 1 つ 1 つステップ実行して動作を確認するスタイルです。この 2 つ目の作業には p.184 「A.3.4 JavaScript の対話的な実行」に示す方法¹⁷があります。

¹⁷他にも “Node.js” (<https://nodejs.org/>) を用いる方法もありますが、これに関しては本書では触れません。

5 JavaScript の基礎

この章では、言語としての JavaScript の基本的な事柄や、JavaScript から HTML 要素にアクセスする方法について説明します。

5.1 変数とデータ型

他のプログラミング言語と同様に、JavaScript でも変数を使用してそこに値を格納します。JavaScript では変数の使用を宣言するためには

```
var 変数名 [, 変数名 [, …]];
```

のように記述¹⁸ します。変数に格納する値には様々な型¹⁹ のものがあります。(変数の使用を宣言する段階では型を指定する必要はありません)

5.1.1 基本的なデータ型

最も基本的なデータ型は次の 3 種類のものです。

- **文字列型** (string) - 二重引用符 "…", もしくは単引用符 '…' で括ったもの。
- **数値型** (number) - 特に整数型, 浮動小数点型の区別はしません。
- **真理値型** (boolean) - true (真) か false (偽) の値をとります。

これら基本的なデータ型を**プリミティブ型**¹⁹ と呼びます。

変数に値を設定するにはイコール「=」を使用します。

例. a = 12 ←変数 a に数値の 12 を設定

5.1.2 変数の初期状態

var によって使用が宣言され、かつ値が未設定の変数を参照すると「undefined」となります。

参考) undefined と似たものに null があり、「値を持たない」ことを明に示すものとして用いられることがあります。

使われていない記号 (var 宣言すらされていない変数) にアクセスしようとするエラー (ReferenceError) が発生します。また不要になった変数は delete 演算子で開放することができます。例えば値を持つ変数 a があるとき、

```
delete a
```

とすると変数 a が開放されて未使用の状態 (var 宣言していない状態) になります。

5.1.3 データ型の変換

数値型のデータと文字列型のデータは異なるものですが、プログラミングに当たって、これらの間でデータの変換が必要になることがあります。例えば、GUI コンポーネントのテキストフィールドから得られるデータは文字列型ですが、これを数値型の値に変換したい場合などがあります。

5.1.3.1 文字列→整数: parseInt メソッド

例. 文字列 st の内容を整数に変換して n に代入します。

```
var n = parseInt(st);
```

parseInt(st, 基数) とすると、指定した基数で st を解釈します。

¹⁸ var による変数宣言の他にも重要なものとして let, const 宣言があります。詳しくは後の「5.5 変数のスコープについて」(p.40) で解説します。

¹⁹ 厳密には、null, undefined, symbol などを含みます。

5.1.3.2 文字列→浮動小数点数: `parseFloat` メソッド

例. 文字列 `st` の内容を浮動小数点数に変換して `n` に代入します.

```
var n = parseFloat(st);
```

参考) `parseInt`, `parseFloat` はそれぞれ `Number.parseInt`, `Number.parseFloat` と同じです.

5.1.3.3 数値→文字列: `toString` メソッド

例. 数値 `n` を文字列に変換して `st` に代入します.

```
var st = n.toString(10);
```

この例のように `toString(基数)` とすると, 指定した基数表現の文字列に変換することができます.

`parseInt` と `toString` を使用すると, 任意の**基数の変換**が可能になります. 次のサンプル `convRadix.html` にそれを示します.

サンプル: `convRadix.html`

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>基数変換</title>
5 <script>
6 function f1() {
7     var r1 = document.getElementById("r1").value;
8     var r2 = document.getElementById("r2").value;
9     var v1 = document.getElementById("v1").value;
10    var v = parseInt(v1,r1);
11    document.getElementById("v2").value = v.toString(r2);
12 }
13 </script>
14 </head>
15 <body>
16 基数1:<input type="text" id="r1" value="10">
17 基数2:<input type="text" id="r2" value="16"><br>
18 値1:<input type="text" id="v1" value="255">
19 <input type="button" value="→変換→" onClick="f1()">
20 値2:<input type="text" id="v2">
21 </body>
22 </html>
```

これを Web ブラウザで実行した例を図 38 を図に示します.

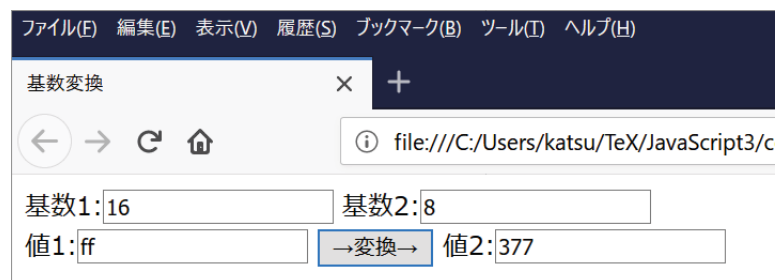


図 38: `convRadix.html` を実行して ff_{16} を 377_8 に変換した例

このサンプルは `id="v1"` のテキストフィールドから読み取った値の基数を変換して `id="v2"` のテキストフィールドに表示するものです. 変換前の基数は `id="r1"` のテキストフィールドに, 変換後の基数は `id="r2"` のテキストフィールドに入力します. サンプルの 7~9 行目でテキストフィールドのタグ `id="r1"`, `id="r2"`, `id="v1"` を JavaScript の変数 `r1`, `r2`, `v1` に取得 (`getElementById` メソッド) して, それぞれの `value` 属性からテキストフィールド内の値を取り出しています. 更にそれに続いて基数の変換処理と表示の処理 (11 行目) をしています.

これら一連の処理はボタンのクリックを起点にして開始しますが、それは 19 行目の「onClick=f1()」の記述によります。この記述は「ボタンがクリックされたら JavaScript の関数 f1() を実行する」ということを意味すると理解しておいてください。JavaScript プログラムの起動にはいくつかの方法があり、テキストの内容を進める中で順次説明します。

HTML タグと JavaScript の間での値のやり取りについては p.62 「5.9 GUI の扱い方」で詳しく解説します。

5.1.4 数値計算

算術演算をはじめとする基本的な計算の表現を表 4 に示します。

表 4: 値 v1, v2 の計算

式	解説	式	解説
v1 + v2	v1 と v2 の和	v1 - v2	v1 と v2 の差
v1 * v2	v1 と v2 の積	v1 / v2	v1 と v2 の商
v1 ** v2	べき乗 $v1^{v2}$	v1 % v2	剰余 (v1 ÷ v2 の余り)

JavaScript には特殊な数値表現のための記号など (表 5) があります。

表 5: 特殊な数値

式	解説	式	解説
Infinity	形式的な無限大	NaN	非数
Number.MAX_VALUE	扱える数の最大値 (正)	Number.MIN_VALUE	扱える数の最小値 (正)
Number.MAX_SAFE_INTEGER	扱える <u>整数</u> の最大値 (正)	Number.MIN_SAFE_INTEGER	扱える <u>整数</u> の最小値 (負)

Infinity はあらゆる数値よりも大きいことを表す 形式的な記号 で、Number.MAX_VALUE、Number.MIN_VALUE はそれぞれ、システムが扱うことのできる最大の数 ($1.7976931348623157e+308$) と最小の数 ($5e-324$) です。また、 $\text{Number.MAX_VALUE} < \text{Infinity}$ です。

数値計算は、絶対値が Number.MIN_VALUE ~ Number.MAX_VALUE の範囲に収まる形で実行するべきです。また、Infinity は数値ではないので計算に使用することはあまり良くありません。(次の例参照)

例. 無限大を使用した演算の試み

```
Infinity + Infinity    →    Infinity
Infinity - Infinity    →    NaN
```

この例からもわかるように、Infinity 同士の差は値として解釈できず、**非数** NaN となります。

5.1.4.1 数学関数と定数

表 6: 数学関数と定数 (一部)

式	解説	式	解説
Math.pow(a,b)	べき乗 a^b	Math.sqrt(x)	平方根 \sqrt{x}
Math.E	自然対数の底 e	Math.exp(x)	指数関数 $\exp[x]$
Math.log(x)	対数関数 $\ln(x)$	Math.log10(x)	対数関数 $\log_{10}(x)$
Math.log2(x)	対数関数 $\log_2(x)$	Math.PI	円周率 π
Math.sin(x)	正弦関数 $\sin(x)$	Math.cos(x)	余弦関数 $\cos(x)$
Math.tan(x)	正接関数 $\tan(x)$	Math.asin(x)	逆正弦関数 $\sin^{-1}(x)$
Math.acos(x)	逆余弦関数 $\cos^{-1}(x)$	Math.atan(x)	逆正接関数 $\tan^{-1}(x)$
Math.round(x)	小数第一位で四捨五入	Math.random()	乱数生成 (0 以上 1 未満)
Math.abs(x)	絶対値 $ x $	Math.sign(x)	正負の符号 $x < 0 \rightarrow (-1)$, $x = 0 \rightarrow 0$, $0 < x \rightarrow 1$

5.1.4.2 桁の大きな整数

通常の number 型の制限を超える**桁の大きな整数** (bigint 型) を扱うには、数値の記述の末尾に **n** をつけます。

例. 123456789^5 の計算

```
123456789**5      →  2.8679718602997177e+40
123456789n**5n    →  28679718602997181072337614380936720482949n
```

このような桁の大きな整数は bigint 型の値です。bigint 型の値を生成する場合に BigInt 関数を使用することもできます。

例. `a = BigInt("193856718396768489276893456529")`

5.1.5 値の比較

値の比較は表 7 のように記述します。

表 7: 2 つの値 v1,v2 の比較

記述	解説
<code>v1 < v2</code>	v2 が v1 より大きい場合に true (真), それ以外の場合は false (偽)。
<code>v1 <= v2</code>	v2 が v1 以上の場合に true (真), それ以外の場合は false (偽)。
<code>v1 > v2</code>	v2 が v1 より小さい場合に true (真), それ以外の場合は false (偽)。
<code>v1 >= v2</code>	v2 が v1 以下の場合に true (真), それ以外の場合は false (偽)。
<code>v1 == v2</code>	v1 と v2 が等しい場合に true (真), それ以外の場合は false (偽)。 v1,v2 の型が互いに違っても「同じ値」と見なせる場合は true (真)。
<code>v1 === v2</code>	v1 と v2 が等しい場合に true (真), それ以外の場合は false (偽)。 v1,v2 の型が互いに異なる場合は false (偽)。

このような比較の記述は、条件分岐や繰り返し制御 (p.33 「5.3 制御」) などで使用します。

5.1.6 型の検査

データのタイプを調べるには `typeof` 関数を使います。例えば変数 `a` に 1 が格納されている場合、`typeof(a)` の戻り値は "number" となります。typeof 関数の戻り値は、調べたデータのタイプを意味する **文字列** です。

number 型の値が整数かどうかを調べるには

```
Number.isInteger( n )
```

と記述します。もしも `n` が整数なら true, それ以外なら false を返します。

使用されていない変数といった**未定義の記号**を typeof 関数に与えると "undefined" が得られます。これを応用すると、記号が使用されているか未使用かを検査することができます。

5.1.7 文字列の表記

文字列は**ダブルクォート** `"` `"` もしくは**シングルクォート** `'` `'` で括って記述²⁰ します。

例. `"apple"`, `'文字列'`, `"He said 'yes'"`, `'Letter "A" is a capital'`

また、**バッククォート** ``` ``` で括ると複数の行に渡る文字列を表現することができます。

例. `s = `一行目
二行目
三行目``

²⁰標準的なブラウザでは文字列は**ダブルクォート** `"` `"` が基本的な表現です。

5.1.7.1 テンプレート文字列

バッククォート「```」で括った文字列は**テンプレート文字列**と呼ばれます。テンプレート文字列の中には**プレースホルダ**を記述することができます。これは JavaScript の式を文字列として展開するものです。例えば

```
a = 12;
b = 34;
```

と値が割り当てられた変数があるとき、

```
s = `a + b = ${a+b}`;
```

とすると、`s` には `"a + b = 46"` という文字列が与えられます。このように、プレースホルダは「`${ 式 }`」のように記述します。

5.1.8 文字列に対する操作

5.1.8.1 連結 (1)： `+`

`+` で連結することで複数の文字列を連結することができます。

例. 文字列 `s1` と `s2` を連結して `s3` に格納する記述

```
s3 = s1 + s2;
```

5.1.8.2 比較

比較演算子 (`"<"`, `"=="`, `">"` など) を用いて文字列の順序を判定することができます。これは、条件分岐や繰り返し制御 (p.33 「5.3 制御」で解説します) などで文字列の比較判定をするときに使用します。比較の順序は文字コードの大小に基いており、おおよそ

数字 < 英大文字 < 英小文字 < ひらがな < カタカナ < 漢字

という順序で判定します。

5.1.8.3 長さ

`length` 属性を用いて文字列の長さを得ることができます。

例. 文字列 `s1` の長さを求める記述

```
s1.length
```

5.1.8.4 部分の取り出し

`substring`, `substr` といったメソッドを用いて文字列の部分を取り出すことができます。

例. 文字列 `s1` の `n` 番目から `m-1` 番目までを取り出す記述

```
s1.substring(n,m)
```

例. 文字列 `s1` の `n` 番目から長さ `1` 個の文字列を取り出す記述

```
s1.substr(n,1)
```

文字列の先頭の位置 (左端の位置) は「`0` 番目」です。

5.1.8.5 分割

`split` メソッドを用いると、指定した区切り文字 (列) を境にして文字列を分割することができます。分割された文字列は配列 (p.44 「5.6 データ構造」で解説) の形で与えられます。

例. コンマ `,` を区切りにして文字列 `s` を分割したものを配列 `sa` に格納する記述。

```
var sa = s.split(",");
```

この結果、分割された部分文字列が `sa[0]`, `sa[1]`, ... に格納されます。

`split` メソッドに与える区切り文字には**正規表現**が使用できます。正規表現に関しては後の「10 正規表現」(p.145)で解説します。

5.1.8.6 連結 (2): join メソッド

文字列を要素として持つ配列があるとき、要素の全てを連結するには join メソッドを使用します。

例. 配列の要素の連結

```
var ar = ["a","b","c"]    ←文字列を要素として持つ配列 ar の生成
s = ar.join(":")          ← ar の要素を区切り文字 ":" で連結
```

この結果、s には "a:b:c" が得られます。join の引数（括弧の中）に空文字 "" を与えると区切り文字なしで連結されます。

5.2 JavaScript のコメント記述

JavaScript の文や式として認識しない内容（コメント）を JavaScript 中に記述するには

```
/*   コメント   */
```

のように “/*” と “*/” で括ります。コメントは複数行にまたがっても構いません。また、

```
// 右端まで1行分のコメント
```

のように、行の先頭に “//” と記述すると、行末（右端）までの1行分がコメントとなります。

5.3 制御

指定した条件を判定して処理の内容を選択して実行したり、繰り返し実行する方法について説明します。

5.3.1 条件分岐 (1) : if 文

次のように if 文を記述して条件判定による処理の選択的実行を実現します。

■ パターン 1

```
if ( 条件 ) {  
    (条件が真になった場合 (成立した場合) の実行部)  
}
```

■ パターン 2

```
if ( 条件 ) {  
    (条件が真になった場合 (成立した場合) の実行部)  
} else {  
    (条件が偽になった場合 (成立しなかった場合) の実行部)  
}
```

■ パターン 3

```
if ( 条件 ) {  
    (条件が真になった場合 (成立した場合) の実行部)  
} else if ( 条件 2 ) {  
    (条件 2 が真になった場合 (成立した場合) の実行部)  
} else if ( 条件 3 ) {  
    (条件 3 が真になった場合 (成立した場合) の実行部)  
    :  
} else {  
    (全ての条件が偽になった場合 (成立しなかった場合) の実行部)  
}
```

5.3.1.1 条件判定のための 3 項演算子

条件判定の結果によって返す値を選択する **3 項演算子** (下記) があります。

条件式 ? 条件式が真の場合に返す値 : 偽の場合に返す値

例. 偶数／奇数を判定するプログラム

```
n = 5;  
s = (n%2==0)? "even" : "odd"
```

この結果, s に "odd" が格納されます。

5.3.2 条件分岐 (2) : switch 文

次のように switch 文を記述して 値の判定 による処理の選択的実行を実現します。

```

switch ( 値 ) {
    case 値 1 :
        (値が値 1 と同じ場合の実行部)
        break;
    case 値 2 :
        (値が値 2 と同じ場合の実行部)
        break;
    :
    default :
        (値がどれにも該当しなかった場合の実行部)
}

```

各 case 句の終わりの break の記述を忘れないように注意してください。

5.3.3 繰り返し (1) : while 文

指定した条件が成立する（真になる）間、処理の実行を繰り返すには次のように記述します。

```

while ( 条件 ) {
    (条件が真である（成立する）場合の実行部)
}

```

条件の判定を、実行部の毎回の終了時に行うには次のように記述（do...while 形式）します。

```

do {
    (実行部)
while ( 条件 );

```

この形式では、実行部の実行が終わったときに条件を判定するので、実行部は最低でも 1 回は必ず実行されます。

5.3.4 繰り返し (2) : for 文

制御用の変数を用いた繰り返し処理を実現するには次のように記述します。

```

for ( 初期化処理; 条件; 各回の後処理 ) {
    (条件が真である（成立する）場合の実行部)
}

```

上記の「初期化処理」「条件」「後処理」には任意のプログラムを書くことができますが、次のような例の使い方が一般的です。

```

for ( c = 0; c < 100; c++ ) {
    (実行部)
}

```

この例では、繰り返しに先立って変数 c を 0 を代入して初期化し、c の値が 100 未満であれば「実行分」を実行します。そして毎回の実行後に c の値を 1 増やします。すなわち、変数 c をカウンタ変数に用いて、この値が 0～99 の間「実行部」を実行します。

5.3.4.1 繰り返し処理の中断

while や for による繰り返し処理の中で break 文を実行すると、その繰り返し処理を強制的に終了します。

5.3.4.2 繰り返し処理のスキップ

while や for による繰り返し処理の中で continue 文を実行すると、その繰り返し処理をスキップして次の回に進みます。

5.4 関数

‘function’ で始まる定義文は関数を定義します。関数とは、計算の対象となる**引数**²¹ に対して計算処理を行って、結果を**戻り値**²² として返すものです。例えば整数 n の階乗 $n!$ は、

$$n! = \begin{cases} n = 0 & \rightarrow 1 \\ n > 0 & \rightarrow n \times (n-1)! \end{cases}$$

と定義されますが、これを JavaScript の関数として実装することを考えます。

実装したサンプルプログラムを testFunc1.html に示します。

サンプル：testFunc1.html

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5  <title>関数テスト</title>
6  <meta charset="utf-8">
7  <script>
8  function fctn( n ) {
9      if ( n == 0 ) {
10         return( 1 );
11     } else {
12         return( n * fctn( n-1 ) );
13     }
14 }
15
16 function fct() {
17     var a = fctn( tx1.value );
18     tx2.value = a;
19 }
20 </script>
21 </head>
22
23 <body>
24 n=<input type="text" id="tx1">
25 <input type="button" value="n! = " onClick="fct()">
26 <input type="text" id="tx2"><br>
27 </body>
28
29 </html>
```

与えた数 n の階乗を計算する関数が fctn(n) として定義されています。このように括弧 ‘()’ の中に記述された n が**引数**であり、関数はこのようにして値を受け取ります。計算結果（戻り値、値域）は return 文で返します。

このサンプルでは、テキストフィールドの値にアクセスするのに直接 id 名を指定（17,18 行目）しています。以前に説明したように、getElementById メソッドで HTML タグを JavaScript の変数に取得するのが基本的な方法ですが、多くの Web ブラウザでは直接に id 名を JavaScript のプログラム中で使用することができ、このサンプルのように「tx1.value」などとして値にアクセスすることができます。

²¹数学の関数でいうところの**定義域**。

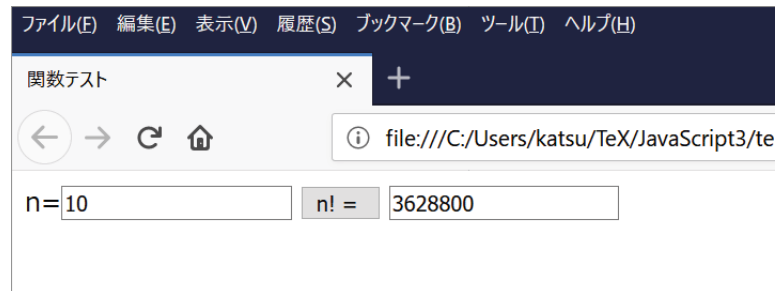
²²数学の関数でいうところの**値域**。

《関数定義の記述》

```
function 関数名 ( 仮引数列 ) {  
    (処理内容)  
    return( 戻り値 );  
}
```

「仮引数列」には複数の引数（仮引数）をコンマ「,」で区切って記述することができます。
「仮引数列」は省略可能で、引数を取らない関数も記述できます。

サンプル testFunc1.html を Web ブラウザで実行した結果の例を図 39 に示します。



「n=」のフィールドに値を入力して「n!=」をクリックすると結果が表示される。

図 39: testFunc1.html を実行した例

JavaScript では、値を返さない（return 文を記述しない）関数も定義することができます。

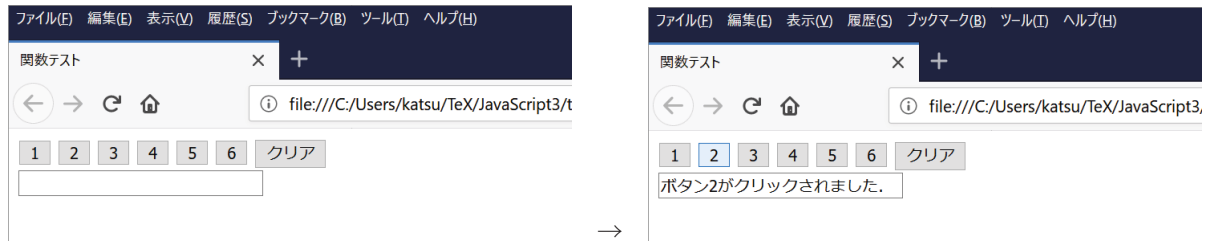
サンプル testFunc1.html の 25 行目にあるように、HTML5 コンテンツでは、html のタグから onClick（クリックの検知）などのイベントハンドリングで JavaScript の関数を呼び出すこと²³が多いですが、この際にも関数に引数を渡すことができます。次に示すサンプルプログラム testFunc0.html でそれを示します。

サンプル：testFunc0.html

```
1  <!DOCTYPE html>  
2  <html>  
3  
4  <head>  
5  <title>関数テスト</title>  
6  <meta charset="utf-8">  
7  <script>  
8  function f( n ) {  
9      if ( n > 0 ) {  
10         tx1.value = "ボタン" + n + "がクリックされました。";  
11     } else {  
12         tx1.value = "";  
13     }  
14 }  
15 </script>  
16 </head>  
17  
18 <body>  
19 <input type="button" value="1" onClick="f(1)">  
20 <input type="button" value="2" onClick="f(2)">  
21 <input type="button" value="3" onClick="f(3)">  
22 <input type="button" value="4" onClick="f(4)">  
23 <input type="button" value="5" onClick="f(5)">  
24 <input type="button" value="6" onClick="f(6)">  
25 <input type="button" value="クリア" onClick="f(0)"><br>  
26 <input type="text" size="30" id="tx1">  
27 </body>  
28  
29 </html>
```

²³p.84「7.1 イベント駆動型プログラミング」で詳しく解説します。

これは、クリックしたボタンの番号が引数として関数 `f` に渡され、それがメッセージとして表示されるプログラムです。各ボタンのクリックに対して同じ関数 `f` が呼び出されますが、呼び出す際に渡す引数がそれぞれ違います。このプログラムを Web ブラウザで実行した例を図 40 に示します。



クリックしたボタンの番号が表示される。

図 40: testFunc0.html を実行した例

5.4.1 関数内でのローカル変数

関数定義の記述内で `var` によって宣言した変数は、その関数内でのみ有効です。そのような変数を**ローカル変数**²⁴といいます。ローカル変数は、その関数の実行が終了したときに失われます。これに対して、関数定義の外部で定義された変数は**グローバル変数**²⁵といい、関数の内外を問わずアクセスできます。ローカル変数の名前にはグローバル変数として使用されている変数と同じものを宣言することができます。その場合、関数内ではローカル変数の方が有効になります。グローバル変数とローカル変数の違いをサンプル testFunc2.html で確かめることができます。

サンプル：testFunc2.html

```
1 <html>
2 <head>
3 <title>変数のスコープ</title>
4 <meta charset="utf-8">
5 <script>
6 var v = 0;
7
8 function f1( ) {
9     tx1.value = v;      // グローバル変数
10    v++;
11    f2();
12 }
13
14 function f2( ) {
15     var v;              // 関数 f2 のローカル変数
16     v = Math.random();
17     tx2.value = v;
18 }
19 </script>
20 </head>
21
22 <body>
23 グローバル変数 v: <input type="text" id="tx1">
24 ローカル変数 v: <input type="text" id="tx2">
25 <input type="button" value="check" onClick="f1()">
26 </body>
27
28 </html>
```

このサンプルでは、`v` という名前の変数が関数定義の外部と関数 `f2` 内で宣言されています。ボタンがクリックされると、それぞれの `v` に別の値を設定します。

サンプル testFunc2.html を Web ブラウザで実行した例を図 41 に示します。

²⁴局所変数と呼ぶこともあります。

²⁵大域変数と呼ぶこともあります。

グローバル変数v :	<input type="text"/>	ローカル変数v :	<input type="text"/>	<input type="button" value="check"/>
------------	----------------------	-----------	----------------------	--------------------------------------

↓

グローバル変数v :	<input type="text" value="0"/>	ローカル変数v :	<input type="text" value="0.5828514996949108"/>	<input type="button" value="check"/>
------------	--------------------------------	-----------	---	--------------------------------------

↓

グローバル変数v :	<input type="text" value="1"/>	ローカル変数v :	<input type="text" value="0.9133386891688609"/>	<input type="button" value="check"/>
------------	--------------------------------	-----------	---	--------------------------------------

↓

グローバル変数v :	<input type="text" value="2"/>	ローカル変数v :	<input type="text" value="0.5777645490501897"/>	<input type="button" value="check"/>
------------	--------------------------------	-----------	---	--------------------------------------

同名のグローバル／ローカル変数が別々の値を保持

図 41: testFunc2.html を実行した例

グローバル／ローカルの v の値が別々に扱えることがわかります。

重要) 関数内でのみ有効な変数を宣言する方法として、後で説明する `let` を用いるものがあります。

5.4.2 関数の高度な応用 (1)： 無名関数

関数定義の書き方は、

```
function 関数名 (引数の列) {…}
```

という形をしています。これは「定義内容…」を「関数名」に与える記述であると見做します。例えば、2つの数の和を計算する関数「wa」は次のように定義することができます。

```
function wa(x,y) {
  return(x+y);
}
```

もちろんこの関数を用いて `wa(2,3)` を評価すると、戻り値として 5 が得られますが、関数名「wa」自体には関数定義の中身が保持されており、その定義内容を別の記号（関数名）に与える²⁶ こともできます。次の例を見てください。

```
tasu = wa;
a = tasu(3,4);
```

これは「wa」の定義内容そのものを「tasu」という記号に与えている例で、結果として a に計算結果である 7 が代入されます。また、関数 wa も次のような記述で定義することができます。

```
wa = function(x,y) {
  return(x+y);
}
```

これは、「function」の直後に関数名を書かない記述で、**無名関数**²⁷ と言います。この例では、無名関数の形で記述された定義内容が記号「wa」に割り当てられています。この記述により `wa(4,5)` は正しく計算（評価）され、9 という結果が得られます。

無名関数と似た働きをするものに、次に説明する**アロー関数**があります。

5.4.3 アロー関数

次のような書き方でも加算する関数「wa」を定義することができます。

```
wa = (x,y) => {
  return(x+y);
}
```

²⁶変数に代入できる形の実体を「第一級オブジェクト」と呼びます。JavaScript では関数定義の内容自体も「オブジェクトとして受け渡しが可能」な第一級オブジェクトです。

²⁷他の言語（Lisp, Python など）で `lambda`（ラムダ）と呼ばれるものに相当します。

これは**アロー関数**²⁸ と呼ばれるもので、与えられた**引数並び**を用いて計算結果を得るための表記法（下記）です。

(引数並び) => { 定義内容 }

上に例示した形の `wa` を通常の関数のように用いて `wa(3,4)` として計算結果を得ることができます。

5.4.4 引数の扱い

関数が受け取った引数は `arguments` からアクセスできます。例えば次のような関数について考えます。

```
function f1(a,b,c) {  
  let n = arguments.length;  
  for ( let i = 0; i < n; i++ ) {  
    console.log( arguments[i] );  
  }  
  return( arguments );  
}
```

この関数は3つの引数を取り、それらをブラウザのコンソールに順番に出力します。ただしその際、仮引数である `a`, `b`, `c` から値を取り出すのではなく、`arguments[0]`, `arguments[1]`, `arguments[2]` から引数の値を取得しています。また関数を呼び出す際に、関数定義の冒頭に記述した仮引数の個数より多い引数を与えた場合、それら引数の全てが `arguments` に保持されます。(次の例参照)

例. 上の関数を `f(1,2,3,4,5)` として呼び出す

`r = f1(1,2,3,4,5)` ←定義した仮引数より多い引数を与える

```
1  
2  
3  
4  
5      ←与えた引数が全て保持されている
```

実行の結果、`r` に `Arguments{ 0:1, 1:2, 2:3, 3:4, 4:5, ... }` が得られます。この得られた値は**連想配列**（オブジェクト／object）で、詳しくは後の「5.6 データ構造」（p.44）で解説します。

5.4.4.1 可変長の引数を受け取る方法

関数を定義する際、仮引数としてドット3つ `...` を記述することで、任意の個数の引数（可変長の引数）を受け取ることができます。先に例示した関数 `f1` と同等の機能を実現する `f2` について考えます。

```
function f2( ...args ) {  
  let n = args.length;  
  for ( let i = 0; i < n; i++ ) {  
    console.log( args[i] );  
  }  
  return( args );  
}
```

この関数の内部では受け取った引数を `args[0]`, `args[1]`, `args[2]`, ...として扱うことができます。また今回は関数の仮引数に `args` と記述していますが、実際には任意の名前を使うことができます。

この `f2` の引数に記述した `args` は**配列**（Array）というもので、詳しくは後の「5.6 データ構造」（p.44）で解説します。また、今回の例のようなドット3つ `...` による記述は**スプレッド構文**と呼ばれるもので、詳しくは後の「5.7 スプレッド構文」（p.57）で解説します。

²⁸アロー関数もまた**無名関数**の1つであり、他の言語の**ラムダ式**に対応するものです。

5.4.4.2 デフォルト引数

関数の定義を記述する際、**デフォルト引数**を設定することができます。これは引数として値が与えられない場合の**暗黙値**を設定するものです。デフォルト引数に関する次の例について考えます。

```
function makeRandom( n = 10, scale = 10 ) {  
  let r = [];  
  for ( let i=0; i<n; i++ ) {  
    r.push(Math.round(scale*Math.random()));  
  }  
  return( r );  
}
```

ここに定義した関数 `makeRandom` は乱数列を生成するもので、2つの引数を取ります。引数 `n` には生成する乱数の個数を、引数 `scale` には各乱数の範囲 (0～`scale`) を与えます。この関数を `makeRandom(3, 100)` として呼び出すと「0～10 の乱数を 3 個」得ることができます。(次の例)

`makeRandom(3, 100)` → [15,65,75] (乱数：この結果は 1 つの例です)

この関数はデフォルト引数が記述されているので、例えば `scale` を省略すると自動的に 10 が与えられたものとして戻り値は次のように 0～10 の乱数となります。

`makeRandom(3)` → [10,2,5] (乱数：この結果は 1 つの例です)

更に `n`, `scale` 両方の引数を省略すると仮引数に記述した設定で乱数を生成します。(次の例)

`makeRandom()` → [3,5,9,1,8,5,3,3,0,2] (結果は 1 つの例です)

5.4.5 関数の高度な応用 (2)： `apply`, `call`

関数の値を評価する方法に `apply`, `call` を用いるものがあります。

書き方： 関数名.`apply`(`this` の指定, [引数 1, 引数 2, …, 引数 `n`])

書き方： 関数名.`call`(`this` の指定, 引数 1, 引数 2, …, 引数 `n`)

これは「関数名 (引数 1, 引数 2, …, 引数 `n`)」の評価と同じです。`apply`, `call` について、次のような関数 `sumall` を例に挙げて説明します。

```
function sumall(...args) {  
  var r = 0;  
  for ( var arg of args ) {  
    r += arg;  
  }  
  return( r );  
}
```

これは引数に与えられた数の合計を求める関数で、通常は `sumall(1,2,3,4)` などと記述して値を評価します。`apply` を使って関数を評価するには次のように記述します。

`sumall.apply(this,[1,2,3,4])`

同様に、`call` を使って関数を評価するには次のように記述します。

`sumall.call(this,1,2,3,4)`

`this` に関しては「9 オブジェクト指向プログラミング」(p.140) のところで説明します。

5.5 変数のスコープについて

先の「5.4.1 関数内でのローカル変数」(p.37) で解説したように、変数には有効な範囲 (グローバル／ローカルなど) があります。ここでは更に重要な事柄に関して解説します。

5.5.1 変数宣言の巻き上げ (hoisting)

関数定義の記述の中で var 宣言をせずに変数を参照すると、その変数はグローバル変数とみなされます。ただし、注意しなければならないことがあります。次のサンプル testFunc3.html を見てください。

サンプル：testFunc3.html

```
1 <html>
2 <head>
3 <title>変数宣言の巻き上げ</title>
4 <meta charset="utf-8">
5 <script>
6 var v = "global";
7
8 function f1( ) {
9     tx1.value = v;          // グローバル変数
10    f2();
11 }
12 function f2( ) {
13     tx2.value = v;          // 1番目
14     // ここで local の var 宣言
15     var v = "local";
16     tx3.value = v;          // 2番目
17 }
18 </script>
19 </head>
20
21 <body>
22 グローバル変数v: <input type="text" id="tx1"><br>
23 関数内でのv (1番目) : <input type="text" id="tx2"><br>
24 関数内でのv (2番目) : <input type="text" id="tx3"><br>
25 <input type="button" value="check" onClick="f1()">
26 </body>
27
28 </html>
```

この例では「check」ボタンをクリックすると各 <input type="text"> のフィールドにプログラム中の各所にある変数 v の値を表示します。

まず6行目でグローバル変数 v が宣言され、文字列 "global" が設定されており、ボタンをクリックすると関数 f1 が呼び出され、id="tx1" のフィールドにそれが表示されます。更に関数 f2 が呼び出され、関数内の変数 v の値が id="tx2", id="tx3" のフィールドに表示されます。記述の順番から考えると、13行目で参照している変数 v はグローバル変数のように考えられますが、実際に Web ブラウザで実行すると意外な結果となります。実行例を図 42 に示します。

グローバル変数v:	<input type="text" value="global"/>
関数内でのv (1番目):	<input type="text" value="undefined"/>
関数内でのv (2番目):	<input type="text" value="local"/>
<input type="button" value="check"/>	

図 42: testFunc3.html の実行結果

この実行結果から、13行目の v の値が "undefined" となっていることがわかります。この行では、関数内で var 宣言をしていない段階なので v はグローバル変数と思われるかもしれませんが、関数内での var 宣言は定義の先頭に巻き上げられ (hoisting)、13行目の v も関数内のローカル変数とみなされます。そのような事情で、13行目では変数 v は「var 宣言された未設定の変数」であることになります。

変数宣言の巻き上げは見つけにくいバグの原因となる可能性があるので注意が必要です。安全のため、関数定義の記述においては、ローカル変数の var 宣言は関数定義の冒頭で行うと良いです。

5.5.2 let 宣言, const 宣言

変数を宣言する際、var ではなく let, const として宣言すると、その変数のスコープ（有効範囲）を { ... } のブロックの範囲内にすることができます。これに関することを確認するためのサンプル testScope1.html を示します。

サンプル：testScope1.html

```
1 <html>
2 <head>
3 <title>let, const</title>
4 <meta charset="utf-8">
5 <script>
6 let v = "global";
7
8 function f1( ) {
9     f2();
10    tx1.value = v;
11 }
12 function f2( ) {
13     let v = "ブロック1";
14     if ( true ) {
15         let v = "ブロック2";
16         if ( true ) {
17             let v = "ブロック3";
18             console.log("block3:"+v);
19         }
20         console.log("block2:"+v);
21     }
22     tx2.value = v;
23 }
24 </script>
25 </head>
26
27 <body>
28 グローバル変数v: <input type="text" id="tx1"><br>
29 関数内でのv: <input type="text" id="tx2"><br>
30 <input type="button" value="check" onClick="f1()">
31 </body>
32
33 </html>
```

このサンプルでは、関数 f2 の中に 2 段の if 文があり、複数のブロックが入れ子になっています。「check」ボタンをクリックすることで各 <input type="text"> のフィールドにグローバルスコープの v と、関数 f2 のスコープの v の値が表示されます。（図 43）

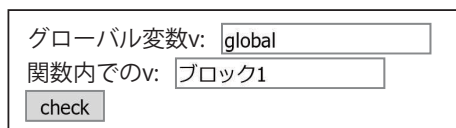


図 43: testScope1.html の実行結果

ブラウザのコンソールメッセージを確認すると、if 文のブロック毎の v の値が確認（下記）できます。

block3:ブロック 3

block2:ブロック 2

以上のことから、ブロック各所の v がそれぞれ別のものとして扱われていることがわかります。

■ const 宣言について

let 宣言と似たものに const 宣言があります。これも変数のスコープをブロックに限定するものですが、値の再設定ができない変数を宣言します。また、const 宣言は宣言時に値の初期設定をする必要があります。すなわち、

```
const a;
```

と記述することはできず（エラーになります）,

```
const a = 123;
```

などと、宣言時に値を設定します。

参考)

JavaScript で規模の大きなプログラムを作成する場合、var による変数宣言に替わって let, const で変数を宣言することが多いです。変数のスコープをブロック単位に限定することで、同名の変数の衝突を防ぐことができ、プログラムの可読性を高めるだけでなく、ソースコードの再利用性を向上させることができます。

let, const 宣言の変数は delete 演算子による削除ができない という点に注意してください。

5.6 データ構造

プログラム中で使用する値を格納するための最も基本的なものは変数ですが、多くのデータを効率良く扱うために配列 (Array) やオブジェクト (Object) を利用することができます。また、これらのデータ構造は JSON (JavaScript Object Notation) として広く普及²⁹ しており、JavaScript での利用にとどまらず、各種アプリケーション間でのデータ交換に利用されています。

配列、オブジェクト以外にも、集合論に近い扱いができる Set オブジェクトや、Map オブジェクト (高度な連想配列) もあり、後ほど説明します。

5.6.1 配列

配列の使用を宣言するには、変数の宣言と同じく `var` を記述しますが、値の読み書きに添字 (整数インデックス: 要素を指定する番号) を付けます。添字は 0 から始まります。

例. `var theBeatles =`

```
["ジョン・レノン","ポール・マッカートニー","ジョージ・ハリスン","リンゴ・スター"];
```

として配列を生成すると、`theBeatles[0]` の値は"ジョン・レノン"を、`theBeatles[1]` の値は"ポール・マッカートニー"になります。

配列の要素の個数は `length` 属性で得られます。すなわち、配列 `theBeatles` の要素数は、`theBeatles.length` を参照して取得します。

配列の要素を順番にテキストエリアに表示するサンプル `"test4.html"` と `"test4.js"` を次に示します。
(今回は HTML と JavaScript を別々のファイルに分離した形で示します)

サンプル: `test4.html`

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>test4</title>
6     <script src="test4.js"></script>
7   </head>
8   <body>
9     <form name="fm">
10      <input type="button" value="ビートルズ" onClick="f01( )">
11      <input type="button" value="クラフトワーク" onClick="f02( )"><br>
12      <textarea name="a1" cols="34" rows="5"></textarea>
13    </form>
14  </body>
15 </html>
```

サンプル: `test4.js`

```
1 var theBeatles =
2   ["ジョン・レノン","ポール・マッカートニー",
3     "ジョージ・ハリスン","リンゴ・スター"];
4
5 var Kraftwerk =
6   ["ラルフ・ヒュッター","フリッツ・ヒルパート",
7     "ヘニング・シュミッツ","ファルク・グリーフェンハーゲン"];
8
9 function f01( ) {
10   var c, txt="";
11   for ( c = 0; c < theBeatles.length; c++ ) {
12     txt = txt + c + " " + theBeatles[c]+"\\n";
13   }
14   document.fm.a1.value = txt;
15 }
```

²⁹RFC 8259

```

16
17 function f02( ) {
18     var c, txt="";
19     for ( c = 0; c < Kraftwerk.length; c++ ) {
20         txt = txt + c + ")" + Kraftwerk[c]+"\\n";
21     }
22     document.fm.a1.value = txt;
23 }

```

"test4.html"を Web ブラウザ内で表示してボタンをクリックしたときの表示例を図 44 に示します。



図 44: test4.html の表示

5.6.1.1 複数の配列の連結

複数の配列を連結するには `concat` メソッドを使用します。例えば複数の配列

配列 1, 配列 2, 配列 3,...

がある場合、

配列 1.`concat`(配列 2, 配列 3,...);

とすると、全ての配列を連結した新たな配列が得られるので、これを別の変数に代入するなどしてください。

5.6.1.2 配列要素の連結

`join` メソッドを用いると、配列変数の全ての要素を順番に連結した文字列を得ることができます。

例. 配列 `ar` の要素を連結する例.

```
var s = ar.join(":");
```

`join` の引数には、連結の際の区切り文字（要素間に挟み込む文字）を指定します。`join` メソッドを用いると、先の "test4.js" をもっと簡略化することができます。（考えてみてください）

5.6.1.3 配列による FILO（スタック）の実現

配列に対して `push` メソッドを用いることで、要素を末尾に追加することができます。例えば配列 `ar` に対して `ar.push("e1");` とやると、末尾に新たな要素 "e1" が追加されます。反対に `pop` メソッドによって末尾の要素を削除することができます。このとき、削除した要素が戻り値になります。

例. 配列 `ar` を `pop` する。

```
var r = ar.pop();
```

これで配列の末尾の要素が削除され、その削除した要素が `r` に代入されます。

先頭の要素を対象とする FIFO も実現可能で、その場合は `push`, `pop` の代わりに `unshift` メソッド（先頭に追加）, `shift` メソッド（先頭を取り出して削除）を用います。

（考察） `push`, `pop`, `unshift`, `shift` を利用して FIFO を実現する方法を考えてください。

5.6.1.4 配列要素の順序の逆転

配列に対して `reverse` メソッドを用いるとその配列の要素の順序が逆順になります。

5.6.1.5 配列かどうかの判定

配列かどうかを判定するには `Array.isArray` を使用します。

例. 配列かどうかの判定

```
Array.isArray( [1,2,3] )    → true (真)
Array.isArray( 123 )        → false (偽)
```

5.6.1.6 配列の途中の要素の削除

配列の途中の要素を削除するには `splice` メソッドを使用します。

書き方: `配列.splice(消去対象のインデックス位置, 消去する個数)`

例. `splice` メソッドによる要素の削除

```
ar = ["A","B","C","D","E","F"]    ←配列 ar の作成
p = ar.splice(2,3)                ←インデックス位置 2 から 3 つの要素を削除
```

この結果, 削除した部分の配列 `["C","D","E"]` が `p` に得られ, `ar` の内容は `["A","B","F"]` となります。

5.6.1.7 配列の部分列の抽出

配列の部分列を抽出するには `slice` メソッドを使用します。

書き方: `配列.slice(抽出開始のインデックス位置, 抽出終了位置+1)`

例. `slice` メソッドの実行

```
ar = ["A","B","C","D","E","F"]    ←配列 ar の作成
p = ar.slice(2,5)                 ←インデックス位置 2 以上 5 未満の部分列を抽出
```

この結果, `p` に部分配列 `["C","D","E"]` が得られます。 `ar` の内容は変わりません。

5.6.1.8 条件による要素の抽出

指定した条件を満たす要素を抽出するには `filter` メソッドを使用します。

書き方: `配列.filter(条件を判定する関数の定義)`

例. 整数の配列から偶数だけを取り出す (その 1)

```
ar = [8, 7, 9, 1, 6, 2, 5, 4, 3, 10];    ←整数の配列
function f1(n) {return( (n%2)==0 );}      ←偶数であることを判定する関数 f1 の定義
ar2 = ar.filter( f1 );                    ← f1 を用いて偶数のみを抽出
```

抽出結果が配列 `ar2` に得られます。 `filter` メソッドの引数には関数名だけでなく, `function` の記述の本体やアロー関数を与えることもできます。 次の 2 つの例 (その 2, その 3) も上の例と同じ結果になります。

例. 整数の配列から偶数だけを取り出す (その 2)

```
ar = [8, 7, 9, 1, 6, 2, 5, 4, 3, 10];    ←整数の配列
ar2 = ar.filter( function(n){return((n%2)==0);} );    ← function の記述を与えて抽出
```

例. 整数の配列から偶数だけを取り出す (その 3)

```
ar = [8, 7, 9, 1, 6, 2, 5, 4, 3, 10];    ←整数の配列
ar2 = ar.filter( n => (n%2)==0 );          ← アロー関数を与えて抽出
```

5.6.1.9 条件による要素の探索

指定した条件を満たす要素を探索するには `find` メソッドを使用します。

書き方: `配列.find(条件を判定する関数の定義)`

「条件を判定する関数の定義」に関しては `filter` メソッドと同様です。 このメソッドは条件を満たす要素の最初のも

のを返します。

例. 8 より大きい要素を探索（最初に見つけたもの）

```
ar = [8, 7, 9, 1, 6, 2, 5, 4, 3, 10];    ←整数の配列
e = ar.find(n=>(n>8));                  ←「8 より大きい」要素を探索
```

この結果, e には 9 が得られます。

findIndex メソッドを使用すると, 見つけた要素そのものではなく「見つけた位置」のインデックスを返します。

例. 8 より大きい要素を探索（最初に見つけたもの）：位置を取得

```
ar = [8, 7, 9, 1, 6, 2, 5, 4, 3, 10];    ←整数の配列
n = ar.findIndex(n=>(n>8));              ←「8 より大きい」要素を探索
```

この結果, n に 2 が得られます。

5.6.1.10 全要素に対する一斉処理

配列の全要素に一斉に処理を実行して, 得られた値の配列を得るには map メソッドを使用します。

書き方: 配列.map(関数の定義)

1 つの引数を取って 1 つの値を返す「関数の定義」を引数に与えます。

例. 配列の全ての要素を 2 倍する

```
ar = [0,1,2,3,4];                      ←配列の作成
ar2 = ar.map( n => 2*n );                ←全要素を一斉に 2 倍
```

この結果, ar2 には [0,2,4,6,8] が得られます。元の配列は変化しません。

5.6.1.11 全要素に対する順次処理

先の map メソッドよりも更に汎用性のある forEach メソッドがあります。

書き方: 配列.forEach(関数の定義)

1 つの引数を取って 1 つの値を返す「関数の定義」を引数に与えます。

例. 全要素をコンソールに出力

```
ar = [0,1,2];                          ←配列の作成
ar.forEach( e => console.log(e) );      ←全要素を順番に出力
```

この処理の結果, コンソールに次のように出力されます。

```
0
1
2
```

注意) forEach は値を返しません。

5.6.1.12 要素に対する累積的処理

2 つの値 x, y を f(x,y) によって 1 つの値にする処理があると, これを繰り返し適用すると複数の値を最終的に 1 つの値にすることができます。例えば, 複数の数値 v_1, v_2, \dots, v_n の合計を求める処理は 2 つの値の加算 $f(v_1, v_2) = v_1 + v_2$ を繰り返し適用することで実現できます。すなわち,

$$f(f(f(v_1, v_2), v_3), v_4)$$

のような繰り返しです。このような形で, 2 項の計算を用いて複数の要素に対する計算が実現できます。

2 つの引数を取り, 1 つの値を返す関数がある場合, それを配列の全要素に対して繰り返し適用し, 1 つの値にするためのメソッド reduce があります。

書き方： 配列.reduce(関数の定義)

例. 全要素の合計を求める処理

```
ar = [0,1,2,3,4];           ←配列の作成
s = ar.reduce( (x,y) => x+y ); ←繰り返し適用
```

この処理の結果, s に 10 が得られます.

累積的処理を施す際の要素の順序を逆にした reduceRight もあります. これに関して例をあげて説明します. まず次のような関数があるとします.

例. 2つの文字列要素を連結して括弧で括る関数

```
function f(x,y) { return("("+x+y+")"); }
```

この関数は, 与えられた2つの文字列を連結して括弧 () で括るもので,

```
["a","b"].reduce(f);
```

を実行すると

```
"(ab)"
```

が得られます. この関数 f を reduce の引数に渡して

```
["a","b","c","d"].reduce(f);
```

を実行すると,

```
"(((ab)c)d)"
```

となりますが, reduceRight に渡して

```
["a","b","c","d"].reduceRight(f);
```

を実行すると,

```
"(((dc)b)a)"
```

が得られ, 適用順序が reduce の場合の逆になっていることがわかります.

5.6.1.13 要素の並べ替え (ソート)

配列に対して sort メソッドを用いることで要素の順序を整列することができます. (対象の配列自体を変更します)

例えば ar という配列を整列するには,

```
ar.sort()
```

を実行します. 整列順序は「昇順」です.

整列の順序は自由に設定することができます. その場合このメソッドは sort(比較関数) のようにして実行します. この比較関数は, 配列の隣接する2つの要素を引数として与えた場合に負, 0, 正の3種類の数値を返す関数で, プログラマが自由に定義することができます. 例えば,

```
function cmp( x, y ) {
    return( y - x );
}
```

のように関数 cmp を定義して, 数値を要素として持つ配列 ar に対して

```
ar.sort(cmp);
```

として実行すると, 配列の要素が数値の降順として整列されます.

比較関数は, 仮引数の並びによって要素の交換をするかどうかを判定するためのもので, 2つの引数が等しい順序である場合に 0 を, 交換処理を要する場合に正の値を返すように定義します.

上の例の cmp 関数では, 2つの仮引数が昇順になっている場合に正の値を, 降順になっている場合に負の値を, 等し

い場合に 0 を返します。従って、整列前の配列の要素の順序が昇順になっている場合は要素の交換処理が働いて降順に整列されます。

5.6.1.14 全ての要素に対する繰り返し処理の実現

配列の全ての要素に対して網羅的に処理を実現する場合は、for 文の中で "of" を使用します。例えば配列 `ar` がある場合は次のように記述します。

```
for ( m of ar ) {  
    (m に ar の要素が 1 つずつ格納されるので、これを処理に使用する)  
}
```

例. 配列を用いた繰り返し (Web ブラウザのコンソールで実行)

```
q = ["a","b","c"];  
for ( m of q ) {  
    console.log( m );  
}
```

これを実行すると Web ブラウザのコンソールに次のように表示されます。

```
a  
b  
c
```

考察) 上記の試みにおいて、"of" の代わりに "in" を用いるとどうなるか試してみてください。

5.6.1.15 全ての要素に対する判定：(全ての～, 1 つでも～)

配列の全ての要素がある条件を満たすかどうかを検査するには `every` メソッドを使用します。また、配列の要素が 1 つでもある条件を満たすかどうかを検査するには `some` メソッドを使用します。例えば、与えられた数が正 (プラス) の値かどうかを判定する次のような関数 `pos` を定義します。

例. 正の数を判定する関数 `pos`

```
function pos(x) { return(x>0); }
```

この関数は真理値 (true/false) を返します。もちろんこの関数は「1 つの値」を判定するもので

```
r1 = pos(3);  
r2 = pos(-3);
```

の結果は、`r1` が true、`r2` が false となります。

この関数を使って、配列の要素全てが正かどうかを `every` メソッドで判定する例を次に示します。

例. `every` による判定：その 1

```
ar1 = [2,4,1,3,7,5];  
r = ar1.every( pos );
```

判定結果の `r` は true となります。当然ですが、配列の要素に 1 つでも負 (マイナス) の値があると判定結果は false となります。(次の例)

例. `every` による判定：その 2

```
ar2 = [2,4,-1,3,7,5];  
r1 = ar2.every( pos );
```

判定結果の `r1` は false となります。

`some` を用いて、配列が「1 つでも正の要素を含むか」を判定する例を次に示します。

例. some による判定

```
r2 = ar2.some( pos );
```

判定結果の r2 は true となります。

5.6.2 オブジェクト（連想配列 1）

オブジェクト（Object）は配列とよく似た働きをしますが、整数インデックスの添字による要素の選択ではなくキーで要素を選択します。使用法はとても簡単で、既存の obj というオブジェクトに対して

```
obj[ "国籍" ] = "日本";
```

とすると obj["国籍"] に”日本”という値が格納されます。キーと値の組を複数まとめてオブジェクトに格納するには次のようにします。

```
obj = { キー 1:値 1, キー 2:値 2, キー 3:値 3, …};
```

キーと値を”:"で区切って組にします。キーは文字列の形で与えます。文字列以外のデータをキーに使用すると、自動的に文字列型に変換されます。文字列型以外のデータをキーとして使用するには、後で説明する Map オブジェクトを使用します。

注意） オブジェクトのキーはプロパティとも呼ばれます。

5.6.2.1 プロパティの表記について

オブジェクトのプロパティにアクセスするには ‘[]’ を付ける方法以外にもドット ‘.’ を付ける書き方もあります。例えば

```
obj = { "grape": "ぶどう", "orange": "みかん" }
```

としてオブジェクト obj を作成した後、obj["orange"] とすると値 ”みかん” が得られますが、obj.orange としても同様の値が得られます。また、キーと値のペアを新規に登録する際も

```
obj.banana = "バナナ"
```

という書き方ができます。

5.6.2.2 プロパティの削除

変数の開放と同様に delete 演算子によってプロパティを削除（キーと値のペアの削除）ができます。

例. プロパティの削除

```
delete obj.grape
```

これによりオブジェクト obj のプロパティ grape とその値が削除されます。（他のプロパティは影響されません）

5.6.2.3 オブジェクトの明示的宣言

new を使うと、明示的にオブジェクトを生成することができます。

例. var o = new Object();

これで空のオブジェクト o が生成されます。また

```
var o = { }
```

でも同様の結果となります。

この方法は既存のオブジェクトを初期化する際にも応用できます。

5.6.2.4 全てのキーの取得

keys メソッドを使うと、指定したオブジェクトの全てのキーを要素として持つ配列を取得することができます。

例. `var k = Object.keys(o);`

これでオブジェクト `o` のすべてのキーを要素として持つ配列 `k` が得られます。これを応用すると、オブジェクトに登録されているデータの個数を調べる（次の例参照）ことができます。

例. `var k1 = (Object.keys(o)).length;`

5.6.2.5 全てのキーに対する繰り返し処理の実現

オブジェクトの全てのデータに対して網羅的に処理をする場合は、`for` 文の中で `"in"` を使用³⁰ します。例えばオブジェクト `Ob` がある場合は次のように記述します。

```
for ( k in Ob ) {  
    (k に Ob のキーが 1 つずつ格納されるので、これを処理に使用する)  
}
```

【サンプルプログラム】

オブジェクトを利用して、データの登録と検索を行うプログラムを作成した例 `OBJtest.html` を次に示します。

サンプル：OBJtest.html

```
1  <!DOCTYPE html>  
2  <html>  
3  <head>  
4  <meta charset="utf-8">  
5  <title>オブジェクトのテスト</title>  
6  <script>  
7  // オブジェクトの宣言  
8  var Obj = new Object();  
9  
10 // データの検索  
11 function srch() {  
12     var k = k1.value + "." + k2.value;  
13     var v = Obj[k];  
14     if ( v == null ) {  
15         v1.value = "（登録がありません）";  
16     } else {  
17         v1.value = v;  
18     }  
19     dcnt();  
20 }  
21  
22 // データの登録  
23 function rgst() {  
24     var k = k1.value + "." + k2.value;  
25     var v = v1.value;  
26     Obj[k] = v;  
27     dcnt();  
28 }  
29  
30 // データを全て消去  
31 function zap() {  
32     Obj = new Object();  
33     dcnt();  
34 }  
35  
36 // データ個数の表示  
37 function dcnt() {  
38     n1.value = ((Object.keys(Obj)).length).toString(10);  
39 }  
40  
41 // データの一覧
```

³⁰連想配列オブジェクトでは、`for` による繰り返しに `"of"` は使用できません。

```

42 function dlst() {
43     var k, v, s = ""
44     var c = 0;
45     for ( k in Obj ) {
46         v = Obj[k];
47         s = s + c + ") " + k + " : " + v + "\n";
48         c++;
49     }
50     a1.value = s;
51     dcnt();
52 }
53 </script>
54 <body>
55 <form name="fm">
56   【簡易データベース】<br>
57   検索してください<br>
58   キー：<input type="text" id="k1"><br>
59   属性：<input type="text" id="k2"><br>
60   値：<input type="text" id="v1"><br>
61   <input type="button" value=" 検 索 " onClick="srch()">
62   <input type="button" value="新規登録" onClick="rgst()">
63   <input type="button" value="全て消去" onClick="zap()"><br>
64   登録データ数：<input type="text" id="n1" size="5"><br>
65   <input type="button" value="データ一覧表示" onClick="dlst()"><br>
66   <textarea id="a1" cols="30" rows="8"></textarea>
67 </form>
68 </body>
69 </html>

```

このプログラムを実行した様子を図 45 に示します。

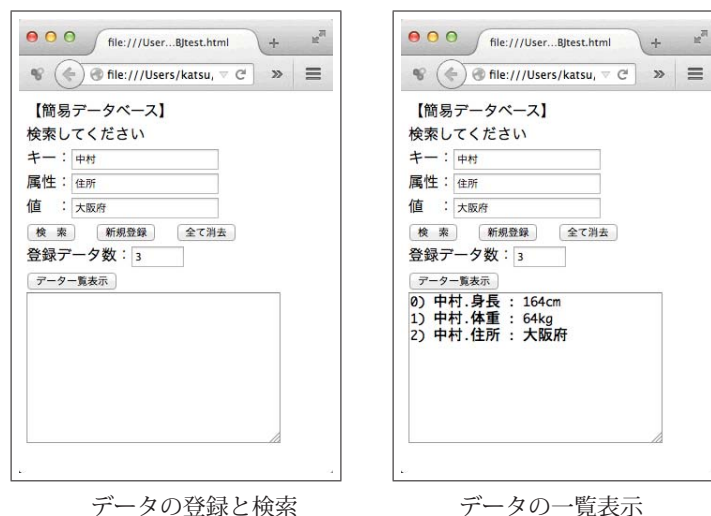


図 45: OBJtest.html の実行の様子

5.6.2.6 オブジェクトを配列に変換する方法

オブジェクトを配列に変換すると、配列用のメソッドが使用できます。Object クラスの entries メソッドを使用するとオブジェクトを配列に変換することができます。

書き方： Object.entries(オブジェクト)

これにより得られた配列は、オブジェクトの各エントリ（キーと値を要素として持つ配列）を要素として持ちます。

例. オブジェクトを配列に変換する

```

obj = {"grape":"ぶどう","orange":"みかん"};    ←オブジェクトの作成
ar = Object.entries(obj)                        ← obj を配列 ar に変換

```

この結果、ar に ['["grape","ぶどう"],["orange","みかん"]] という形の配列が得られます。

5.6.3 Set オブジェクト

重複しない複数の要素を保持するデータ構造に Set があります。Set に対する基本的な操作としては、要素の追加と削除、要素の存在検査があります。配列と違い、Set の要素にはインデックスによるアクセスができません。

5.6.3.1 Set の作成

‘new Set()’ で作成することができます。また作成時に Set() の引数に配列を与えることで、その配列の要素を Set の要素とすることができます。

例. Set の作成

```
s = new Set()           ← 空の Set オブジェクト s を作成
s = new Set(["a","b","c"]) ← 要素を与えて Set オブジェクト s を作成
```

5.6.3.2 要素の追加

add メソッドで Set オブジェクトに要素を追加することができます。

例. 要素の追加

```
s.add("d")           ← Set オブジェクト s に要素 "d" を追加
```

5.6.3.3 要素の削除

delete メソッドで Set オブジェクトの要素を削除することができます。

例. 要素の削除

```
s.delete("a")        ← Set オブジェクト s の要素 "a" を削除
```

5.6.3.4 要素の存在検査

has メソッドで、指定した要素が Set オブジェクトに含まれるかどうかを検査できます。

例. 要素の存在検査

```
p = s.has("c")        ← Set オブジェクト s が要素 "c" を持つかどうかを検査
```

要素が存在すれば true を、存在しなければ false が p に得られる。

5.6.3.5 要素数の取得

Set オブジェクトの size プロパティから要素数が得られます。

例. 要素数の取得

```
n = s.size            ← Set オブジェクト s の要素数を n に取得
```

5.6.3.6 全要素の消去

clear メソッドで、Set オブジェクトが持つ全ての要素を削除することができます。

例. 全要素の消去

```
s.clear()             ← Set オブジェクト s の要素数を削除
```

5.6.3.7 Set を配列に変換する方法

Array クラスの from メソッドで、Set オブジェクトの要素を配列に変換することができます。

例. Set を配列に変換

```
s = new Set(["a","b","c"]) ← Set オブジェクトを変数 s に設定
ar = Array.from(s)         ← Set オブジェクトを配列に変換
```

この例では変数 ar に配列を取得しています。これを応用すると、配列と Set オブジェクト間の相互変換が可能となります。

5.6.4 Map オブジェクト (連想配列 2)

Map オブジェクトは先に説明したオブジェクト型と似た働きをするデータ構造ですが、文字列型以外のデータをキーとして使うことができます。

5.6.4.1 Map オブジェクトの作成

■ 空の Map オブジェクトの作成

書き方の例: `var m = new Map()`

これによって、空の Map オブジェクトが作成されて変数 `m` に与えられます。

■ 初期値を与える形の Map オブジェクトの作成

書き方の例: `var m = new Map([[キー 1, 値 1], [キー 2, 値 2], …, [キー n, 値 n]])`

これによって、キーと値の組 (エントリ) を与える形で Map オブジェクトが作成され、変数 `m` に与えられます。

例. Map オブジェクトの作成

```
var m = new Map([["dog", "犬"], ["cat", "猫"]]);
```

これにより変数 `m` の値が `Map{ dog => "犬", cat => "猫" }` となります。また、キーと値が「=>」で対応づけられます³¹。

5.6.4.2 エントリの参照と登録

キーに対応する値を参照するには `get` メソッドを使います。

例. 値の参照 (先の例の続き)

```
m.get("dog")
```

このように、`get` の引数にキーを与えると、それに対応する値が返されます。また、キーと値の組 (エントリ) を新たに登録するには `set` メソッドを使います。

例. エントリの登録 (先の例の続き)

```
m.set("bird", "鳥")
```

このように、`set` の第 1 引数にキーを、それに対する値を第 2 引数に与えます。

5.6.4.3 エントリの確認

Map オブジェクトに存在しないキーを `get` で参照しようとすると、「未定義」を意味する `undefined` が返されます。Map オブジェクトにキーが存在するかどうかを調べるには `has` メソッドを使用します。

例. キーの存在チェック (先の例の続き)

```
m.has("fish")      ← 存在しないキーの検査
→ false           ← 「存在しない」

m.has("dog")        ← 存在しないキーの検査
→ true            ← 「存在する」
```

このように、存在の有無を真理値で返します。

5.6.4.4 エントリの個数の調査

Map オブジェクトに登録されているエントリの個数は `size` プロパティから参照できます。

³¹Google Chrome ブラウザの場合、Mozilla Firefox では「→」の記号が使われます。

例. エントリの個数を調べる (先の例の続き)

```
m.size      ← Map オブジェクト m が持つエントリの数調べる  
→ 3        ← エントリ数
```

5.6.4.5 エントリの削除

既存のエントリを削除するには delete メソッドを使います.

例. エントリの削除 (先の例の続き)

```
m.delete("dog")    ← キーが "dog" であるエントリを削除
```

delete を実行すると, 真理値が返されます. すなわち, 既存のエントリの削除が成功した場合に true を, 元々存在しなかったキーを指定すると false を返します.

5.6.4.6 全エントリの消去

全てのエントリを削除して中身を空にするには clear メソッドを使います.

例. 全エントリの削除 (先の例の続き)

```
m.clear()
```

このように, 引数なしで実行します.

5.6.4.7 キーの列, 値の列の取出し

全エントリのキーのみの列を取り出すには keys を, 値のみの列を取り出すには values を使います.

例. キーのみ, 値のみを列として取り出す

```
m = new Map([[1,"one"],[3.1415926535,"pi"],[[1,2,3],"list"]])    ← サンプル  
k = m.keys()             ← キーのみの列を k に取得  
v = m.values()           ← 値のみの列を v に取得
```

この例では, キーのみの列を k に, 値のみの列を v に取得しています. 得られた列は MapIterator という型のオブジェクトで, これは for による繰り返し処理に使うことができます. 具体的な方法について次に解説します.

5.6.4.8 全要素に対する順次処理

Map オブジェクトから抽出したキーの列と値の列 (MapIterator 型の列) を表示するサンプルを map01.html に示します.

サンプル: map01.html

```
1  <!DOCTYPE html>  
2  <html lang="ja">  
3  <head>  
4  <meta charset="utf-8">  
5  <title>Map オブジェクトの扱い1 </title>  
6  <script>  
7  function f1 ( x ) {           // 型と値を表示する関数  
8      document.write("型:");    document.write(typeof(x));  
9      document.write(", ");    document.write(x);  
10     document.write("<br>");  
11 }  
12 // サンプル  
13 var m = new Map([[1,"one"],[3.1415926535,"pi"],[[1,2,3],"list"]]);  
14 var k = m.keys();             // キーの列  
15 var v = m.values();           // 値の列  
16 // 出力  
17 document.write("*** キーの列 ***<br>");  
18 for ( e of k ) {  
19     f1(e);  
20 }  
21 document.write("*** 値の列 ***<br>");
```



```

22 | for ( e of v ) {
23 |     f1(e);
24 | }
25 | </script>
26 | </head>
27 | <body></body>
28 | </html>

```

サンプル中にあるように「for (列の要素 of 列)」の形で列の要素にアクセスすることができます。これをブラウザで表示すると図 46 のようになります。

```

*** キーの列 ***
型: number, 1
型: number, 3.1415926535
型: object, 1,2,3
*** 値の列 ***
型: string, one
型: string, pi
型: string, list

```

図 46: map01.html をブラウザで表示した例

もちろん、キーと値を別々の列にせず、Map オブジェクトをそのまま for 文に与えることもできます。その例を map02.html に示します。

サンプル: map02.html

```

1 | <!DOCTYPE html>
2 | <html lang="ja">
3 | <head>
4 | <meta charset="utf-8">
5 | <title>Mapオブジェクトの扱い2 </title>
6 | <script>
7 | function f2 ( k, v ) {           // キーと値を表示する関数
8 |     document.write(k); document.write(" -> ");    document.write(v);
9 |     document.write("<br>");
10 | }
11 | // サンプル
12 | var m = new Map([[1,"one"],[3.1415926535,"pi"],[[1,2,3],"list"]]);
13 | // 出力
14 | document.write("*** エントリ一覧 ***<br>");
15 | for ( e of m ) {
16 |     f2(e[0],e[1]);
17 | }
18 | </script>
19 | </head>
20 | <body></body>
21 | </html>

```

このサンプルでも「for (個々のエントリ of Map オブジェクト)」の形で Map オブジェクトのエントリにアクセスすることができます。これをブラウザで表示すると図 47 のようになります。

```

*** エントリ一覧 ***
1 -> one
3.1415926535 -> pi
1,2,3 -> list

```

図 47: map02.html をブラウザで表示した例

5.6.4.9 Map を配列に変換する方法

Map オブジェクトを配列に変換すると、配列用の各種のメソッドを使用することができます。変換には Array クラスの from メソッドを使います。

例. Map を配列に変換

```
m = new Map([[1,"one"],[3.1415926535,"pi"],[[1,2,3],"list"]])    ← Map オブジェクトを変数 m に設定
ar = Array.from(m)      ← Map オブジェクトを配列に変換
```

この例では変数 ar に配列を取得しています。これを応用すると、配列と Map オブジェクト間の相互変換が可能となります。

参考) Array.from で MapIterator オブジェクトを配列に変換することもできます。

5.7 スプレッド構文

配列をはじめとするデータ構造の要素を展開する記述の形式に**スプレッド構文**があります。例えば次のような配列を考えます。

```
a1 = ["a","b","c"]
```

この配列の要素を展開して別の配列の要素にするには次のように記述します。

```
a = [...a1]
```

このように、配列の前にドットを3つ「...」付けた書き方がスプレッド構文です。この処理の結果、a は ["a","b","c"] となります。この例だけではスプレッド構文の有用性はわかりにくいですが、次の例を見てください。

```
a = [0,1,2, ...a1]
```

この結果、a は [0, 1, 2, "a", "b", "c"] となります。つまりスプレッド構文はデータ構造の要素を展開して「そこにあるかのように書き並べる」記述表現とすることができます。「...」は次の例のように複数記述することができます。

```
a = ["x", ...a1, "y", ...a1]
```

この結果、a は ["x","a","b","c","y","a","b","c"] となります。

配列以外のデータ構造もスプレッド構文で展開することができます。

例. Set オブジェクトを配列の要素内に展開する

```
s = new Set(["a","b","c"]);    ← Set オブジェクト s を作成
ar = [...s, "d","e"];          ← s の要素を配列の要素に展開
```

この結果、配列 ar は ["a","b","c","d","e"] となります。

5.7.1 スプレッド構文の関数の仮引数への応用

関数やメソッドを定義する際の仮引数にスプレッド構文を用いると**可変長の引数**を受け付けることができます。例えば次のような関数 sumAll について考えます。

```
function sumAll(...args) {
  let r = 0;
  for (let i=0; i<args.length; i++ ) {
    r += args[i];
  }
  return( r );
}
```

この関数は引数に与えられた数を全て合計した結果を返すものです。定義を記述する際の仮引数にスプレッド構文を用いており、これにより任意の個数の引数を受け取ることができます。(次の例)

例. 可変長の引数の受け付け

```
sumAll(1)           → 1    (引数 1 個)
sumAll(1,2,3)       → 6    (引数 3 個)
sumAll(1,2,3,4,5,6,7,8,9,10) → 55 (引数 10 個)
```

5.7.2 スプレッド構文の分割代入への応用

分割代入とは、変数への値の割り当てにデータ構造を応用するもので、例えば次のような処理が 1 つの例です。

```
[a,b,c] = [1,2,3]
```

これは次のような記述と同等の結果となります。

```
a = 1; b = 2; c = 3;
```

分割代入はデータ構造の入れ子にも対応します。例えば、

```
[a, [_,c], d] = [2, [3,4], 5]
```

を実行すると、`a = 2; c = 4; d = 5` となります。またこの例のように、左辺にアンダースコアを記述することで、受け取る変数を省略することができます。

分割代入の左辺にはスプレッド構文を用いることができます。(次の例)

```
[a,b, ...c] = [0,1,2,3,4,5,6,7]
```

この結果、`a = 0; b = 1; c = [2,3,4,5,6,7]`; となります。

左辺に記述するスプレッド構文はデータ列の末尾に記述する必要があります。例えば次のような分割代入はできません。(エラーになります)

```
[...a, b] = [0,1,2,3,4,5] ← これはエラー
```

入れ子の分割代入にスプレッド構文を使用することもできます。(次の例)

```
[a,[b, ...c],d] = [0,[1,2,3,4],5]
```

この結果、`a = 0; b = 1; c = [2,3,4]; d = 5`; となります。

5.7.2.1 オブジェクトの分割代入

オブジェクトの分割代入が可能です。(次の例)

```
var obj = {d:4,c:3,b:2,a:1};
var {a,b} = obj;
```

この結果、`a = 1; b = 2`; となります。この例のように、右辺に受け取るための変数の並びを '{~}' で括って記述します。また、次の例のように代入式自体を '()' で括ると、宣言文の形を取らずに分割代入ができます。

```
({a,b} = {d:4,c:3,b:2,a:1})
```

オブジェクトの分割代入においてもスプレッド構文を使用することができます。(次の例)

```
({a,b, ...rest} = {d:4,c:3,b:2,a:1})
```

この結果、`a = 1; b = 2; rest = {d:4, c:3}` となり、「...」で指定した変数に残りのオブジェクトが代入されます。

5.8 日付・時刻・時間

システムの現在の時刻は例えば次のようにして取得します。

```
var datetime = new Date();
```

これにより、この文を実行した瞬間の日付と時刻の情報がオブジェクト `datetime` に格納されます。このオブジェクトに次のようなメソッドを用いることで年、月、日、曜日、時、分、秒の情報を取り出すことができます。

この例において、`datetime` から西暦年を取り出すには例えば次のように `getFullYear` メソッドを使用します。

```
var yyyy = datetime.getFullYear();
```

これで変数 `yyyy` に 4 桁の西暦年が格納されます。同様にして月、日、曜日、時、分、秒を取得します。

5.8.1 年、月、日、曜日、時、分、秒を取得するメソッド

■ 西暦年の取得： `getFullYear()`

■ 月の取得： `getMonth()`

■ 日の取得： `getDate()`

■ 曜日の取得： `getDay()`

曜日は 0～6 の数値で表現され、日曜日を 0 とします。

■ 時の取得： `getHours()`

■ 分の取得： `getMinutes()`

■ 秒の取得： `getSeconds()`

■ ミリ秒の取得： `getMilliseconds()`

JavaScript では日付・時刻の規準を西暦 1970 年 1 月 1 日 0 時 0 分 0 秒 (UTC) とし、ここから何ミリ秒経過したかという数値で管理しています。Date オブジェクトの日付・時刻をミリ秒単位の数値に変換するには `getTime` メソッドを使用します。

例. 日付・時刻をミリ秒単位に変換

```
var d = new Date();
```

```
var m = d.getTime();
```

この例では、ミリ秒単位での現在時刻を `m` に得ています。逆に、ミリ秒単位の時刻を Date オブジェクトに変換するには次のようにします。

```
var d2 = new Date( m );
```

これで、`m` が日付・時刻表現のオブジェクトに逆変換され、`d2` として得られます。

5.8.2 時間の扱い（再設定、加算、差分）

先の例で得られたオブジェクト `datetime` に対して次のようなメソッドを用いることで日付・時刻の情報を再設定することができます。

■ 西暦年の設定： `setFullYear(年)`

■ 月の設定： `setMonth(月)`

■ 日の取得： `setDate(日)`

■ 曜日の設定： `setDay(曜日)`

曜日は 0～6 の数値で指定し、日曜日を 0 とします。

■ 時の設定： `setHours(時)`

■ 分の設定： `setMinutes(分)`

■ 秒の設定： `setSeconds(秒)`

■ ミリ秒の設定： `setMilliseconds(ミリ秒)`

例えば、`var d = new Date();`として現在時刻を取得した後、この`d`に対して、`d.setFullYear(2001);`とすると、`d`の西暦年の部分だけ2001に変更することができます。

5.8.3 サンプルプログラム

現在時刻を取得して、そこから日付・時刻の各情報を分解して表示し、更に「1年後」、「13ヶ月後」、「90日後」、「72時間後」、「90分後」、「7,200秒後」の時刻を算出するプログラム `testDateTime.html` を示します。

サンプル：testDateTime.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>時間の扱い</title>
6 <script>
7 function f1() {
8     // 現在時刻の取得
9     var dttm = new Date();
10    tx1.value = dttm;
11    // 日付・時刻の分解
12    yyyy.value = dttm.getFullYear();
13    mm.value = dttm.getMonth() + 1;
14    dd.value = dttm.getDate();
15    hh.value = dttm.getHours();
16    min.value = dttm.getMinutes();
17    sec.value = dttm.getSeconds();
18    // 「年」の加算
19    d = new Date( dttm );
20    d.setFullYear( dttm.getFullYear() + 1 );
21    tx2.value = d;
22    // 「月」の加算
23    d = new Date( dttm );
24    d.setMonth( dttm.getMonth() + 13 );
25    tx3.value = d;
26    // 「日」の加算
27    d = new Date( dttm );
28    d.setDate( dttm.getDate() + 90 );
29    tx4.value = d;
30    // 「時」の加算
31    d = new Date( dttm );
32    d.setHours( dttm.getHours() + 72 );
33    tx5.value = d;
34    // 「分」の加算
35    d = new Date( dttm );
36    d.setMinutes( dttm.getMinutes() + 90 );
37    tx6.value = d;
38    // 「秒」の加算
39    d = new Date( dttm );
40    d.setSeconds( dttm.getSeconds() + 7200 );
41    tx7.value = d;
42 }
43 </script>
44 </head>
45 <body>
46 <input type="button" value="今は?" onClick="f1()">
47 <input type="text" size="40" id="tx1"><br>
48 <!-- 日付・時刻の分解 -->
49 日付・時刻情報の分解)
50 年:<input type="text" id="yyyy" size="4">
51 月:<input type="text" id="mm" size="2">
52 日:<input type="text" id="dd" size="2">
53 時:<input type="text" id="hh" size="2">
54 分:<input type="text" id="min" size="2">
55 秒:<input type="text" id="sec" size="2"><br><br>
56 <!-- 半年後の日付・時刻 -->
57 1年後:<input type="text" size="40" id="tx2"><br>
```

```

58 13ヶ月後:<input type="text" size="40" id="tx3"><br>
59 90日後:<input type="text" size="40" id="tx4"><br>
60 72時間後:<input type="text" size="40" id="tx5"><br>
61 90分後:<input type="text" size="40" id="tx6"><br>
62 7200秒後:<input type="text" size="40" id="tx7"><br>
63 </body>
64 </html>

```

このプログラムを Web ブラウザで実行した様子を図 48 に示します。

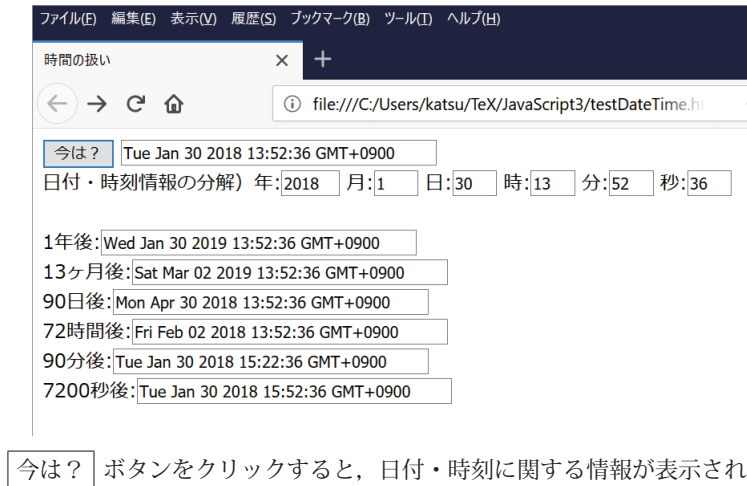


図 48: Web ブラウザによる testDateTime.html の表示

日付・時刻をミリ秒単位に変換し、それを再度日付・時刻の表現に戻す処理を示すサンプル testGetDate.html を次に示します。

サンプル：testGetDate.html

```

1  <html>
2
3  <head>
4  <meta charset="utf-8">
5  <title>経過時間の計算</title>
6  <script>
7  function f1() {
8      var dtm = new Date();
9      var ms = dtm.getTime();
10     tx1.value = ms / 1000.0;
11     var dt2 = new Date( ms );
12     tx2.value = dt2;
13     tx3.value = Math.round( ms / (1000*60*60*24) );
14 }
15 </script>
16 </head>
17
18 <body onLoad="f1()">
19 1970年1月1日0時0分0秒(UTC)から<input type="text" id="tx1">秒経過しています。 <br>
20 日付にすると<input type="text" size="40" id="tx2">で、日数にすると
21 <input type="text" size="10" id="tx3">日です。
22 </body>
23
24 </html>

```

このプログラムを Web ブラウザで実行した様子を図 49 に示します。

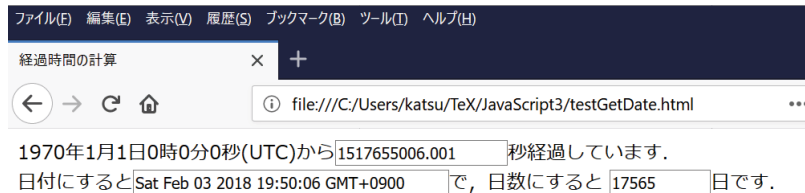


図 49: Web ブラウザによる testGetDate.html の表示

5.8.4 経過時間の算出について

日付・時刻をミリ秒単位で取得すると、2つの異なる時刻の間の差分（経過時間）を単純な引き算で求めることができます。ただし、経過時間の年月日は正確に求めるには少し工夫が必要³²です。

5.9 GUIの扱い方

ここでは、GUIのコンポーネントから値を取り出したり、逆に値を設定したりする方法について説明します。サンプルとして1つのformのGUIから値を取得して、別のformのGUIに取得した値を与えるプログラム test3.html を考えます。

サンプル：test3.html

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>GUIのテスト </title>
6      <script>
7        var txt, pwd, ta, sel;
8        function f01( ) {
9          txt = document.fm.t1.value;
10         pwd = document.fm.p1.value;
11         ta  = document.fm.a1.value;
12         sel = document.fm.s1.value;
13         document.fm2.t2.value = txt;
14         document.fm2.p2.value = pwd;
15         document.fm2.a2.value = ta;
16         if ( document.fm.c1.checked ) {
17           document.fm2.c3.value = "yes";
18         } else {
19           document.fm2.c3.value = "no";
20         }
21         if ( document.fm.c2.checked ) {
22           document.fm2.c4.value = "yes";
23         } else {
24           document.fm2.c4.value = "no";
25         }
26         if ( document.fm.r1[0].checked ) {
27           document.fm2.c5.value = "yes";
28         } else {
29           document.fm2.c5.value = "no";
30         }
31         if ( document.fm.r1[1].checked ) {
32           document.fm2.c6.value = "yes";
33         } else {
34           document.fm2.c6.value = "no";
35         }
36         document.fm2.s2.value = sel;
37       }
38     </script>
39   </head>
40   <body>
41     <form name="fm">

```

³²グレゴリオ暦の性質による。

```

42      <input type="checkbox" name="c1" checked>ポイントを受け取る<br>
43      <input type="checkbox" name="c2">メールマガジンを希望する<br>
44      性別:<input type="radio" name="r1">男性,
45      <input type="radio" name="r1" checked>女性<br>
46      年齢:
47      <select name="s1">
48          <option value="20"> 20代</option>
49          <option value="30"> 30代</option>
50          <option value="40"> 40代</option>
51          <option value="50"> 50代</option>
52      </select>
53      <br>
54      ユーザ名: <input type="text" name="t1" size="20" value="値の設定"><br>
55      パスワード:<input type="password" name="p1" size="20" value="mypswd"><br>
56      よろしければメッセージを入力してください:<br>
57      <textarea name="a1" cols="38" rows="5" ></textarea>
58      <br>
59      <input type="button" value="送信" onClick="f01( )">
60  </form>
61  <hr>
62  <form name="fm2">
63      (内容確認表示)<br>
64      ユーザ名: <input type="text" name="t2" size="20"><br>
65      パスワード:<input type="password" name="p2" size="20"><br>
66      チェックボックス1の状態:<input type="text" name="c3"><br>
67      チェックボックス2の状態:<input type="text" name="c4"><br>
68      ラジオボタン1の状態: <input type="text" name="c5"><br>
69      ラジオボタン2の状態: <input type="text" name="c6"><br>
70      年齢選択の状態: <input type="text" name="s2"><br>
71      メッセージ内容:<br>
72      <textarea name="a2" cols="38" rows="5" ></textarea><br>
73  </form>
74  </body>
75 </html>

```

この例では、fm という名の form の GUI から値を取り出し、別の fm2 という名の form の GUI に値を設定します。以後、このサンプルにしたがって GUI の値の取り扱いについて説明します。

5.9.1 GUI のコンポーネントの名前と値

GUI のコンポーネント（ボタンやテキストフィールドなどの GUI パーツ）には基本的に名前を与え、指定した名前の GUI から値を取り出したり、それに値を設定したりします。

GUI に名前を与えるには、該当する GUI のタグに `name="名前"` という記述を挿入します。例えばテキストフィールドに対して名前 `t1` を与えるには

```
<input type="text" name="t1">
```

と記述します。GUI から値を取得するには `value` 属性を参照します。例えば fm という名前の form に属する n1 という名前の GUI から値を取得するには

```
document.fm.n1.value
```

を参照します。例えば

```
v1 = document.fm.n1.value;
```

とすれば変数 `v1` に取得した値が格納されます。

5.9.1.1 要素の属性と値について

文書中の要素（各種のタグ）の属性の値を取得したり、属性に値を設定するには様々な方法があります。先に説明したように、ドット `'.'` で属性を指定してアクセスする方法とは別に、属性値の取得と設定のための JavaScript のメソッド（表 8 参照）があります。

例えば、要素 `elm` に対して、

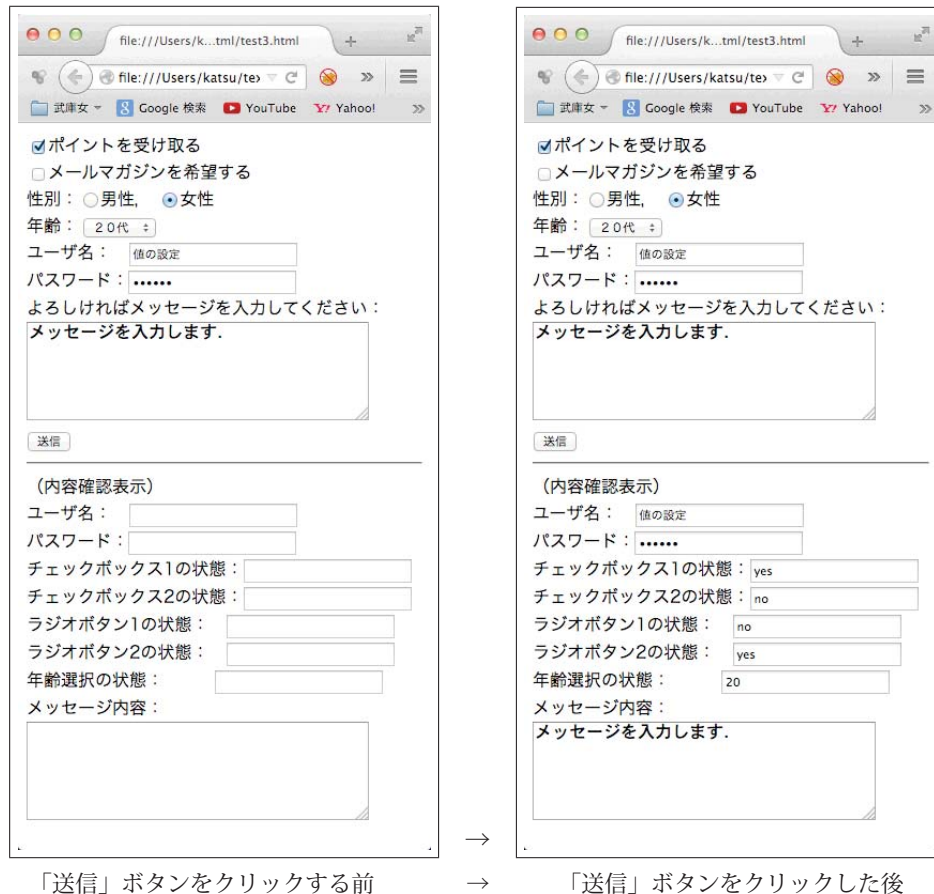


図 50: Web ブラウザによる test3.html の表示

表 8: 要素の属性にアクセスするメソッド

メソッド	説明
<code>getAttribute("属性名")</code>	属性名で指定した属性の値を返す。
<code>setAttribute("属性名", "値")</code>	属性名で指定した属性に値で指定する値を設定する。

```
elm.setAttribute("value","100")
```

とすると、要素 `elm` の属性 `value` に "100" という値が設定されます。

5.9.2 ボタンのクリック

前節 4.1 で見たように、`<button>` タグの中に `onClick="関数呼び出し"` を記述することで、ボタンがクリックされた時に起動するプログラム（関数）を指定することができます。

再度解説) `<button onClick="f01()">押してください</button>`

ただし `<button>` タグは Web サーバとの通信を前提としているので、ボタンをクリックした後の処理は Web サーバからの応答を受信することを想定しています。このため、ローカルのみでの処理を行いたい場合は、ボタンを次のように `<input>` タグとして設置すると良いです。

```
<input type="button" value="送信" onClick="f01( )">
```

5.9.3 テキストフィールド／パスワードフィールド／テキストエリアの内容

例えば、"fm" という名前の form に属するテキストフィールド `t1` から値を取得して変数 `v1` に格納するには、

```
v1 = document.fm.t1.value;
```

とします。逆に、変数 `v1` が保持する値をテキストフィールド `t1` に与えるには

```
document.fm.t1.value = v1;
```

とします。これによりテキストフィールドに内容を与えることができます。また、パスワードフィールドやテキストエリアに対しても同様のことができます。

テキストエリア内の内容を改行するには、値の文字列の行末に"
"を入れます。

注意：

テキストボックス／パスワードフィールドから得られた値は文字列型です。それらを数値に変換したい場合は、先の 5.1.3 を参照してください。

5.9.4 チェックボックスの状態

例えば、"fm" という名前の form に属するチェックボックス c1 がチェックされているかどうかを調べるには

```
document.fm.c1.checked
```

の真偽を調べます。すなわち、

```
if ( document.fm.c1.checked ) {  
    チェックされている場合の処理  
} else {  
    チェックされていない場合の処理  
}
```

checked 属性に true/false の値（真理値）を設定することもできます。

5.9.5 ラジオボタンの状態

ラジオボタンはグループでまとめられており、n 個のラジオボタンは 0～n-1 の番号が付けられています。例えば、"fm" という名前の form に属するラジオボタン r1 の m 番目の要素がチェックされているかどうかを調べるには

```
document.fm.r1[m].checked
```

の真偽を調べます。あとはチェックボックスの場合と同様の扱いができます。

5.9.6 選択リストの状態

例えば、"fm" という名前の form に属する選択リスト s1 から選択されている値を取得して変数 sel に格納するには、

```
sel = document.fm.s1.value;
```

とします。

5.9.7 スライダの値

例えば、"fm" という名前の form に属するスライダ s11 から選択されている値を取得して変数 v に格納するには、

```
v = document.fm.s11.value;
```

とします。

5.9.8 更に簡単な方法

HTML の各要素（タグ）にアクセスするには様々な方法があります。要素の id 属性を引用することでも当該要素にアクセスすることができ、そのような方法を取ると、先のサンプル test3.html をもっと見やすい形にすることができます。（例. test3-2.html）

サンプル：test3-2.html

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>GUIのテスト</title>
6      <script>
7        var txt, pwd, ta, sel;
8        function f01( ) {
9          t2.value = t1.value;
10         p2.value = p1.value;
11         a2.value = a1.value;
12         s2.value = s1.value;
13         if ( c1.checked ) {
14           c3.value = "yes";
15         } else {
16           c3.value = "no";
17         }
18         if ( c2.checked ) {
19           c4.value = "yes";
20         } else {
21           c4.value = "no";
22         }
23         if ( r1_1.checked ) {
24           c5.value = "yes";
25         } else {
26           c5.value = "no";
27         }
28         if ( r1_2.checked ) {
29           c6.value = "yes";
30         } else {
31           c6.value = "no";
32         }
33       }
34     </script>
35   </head>
36   <body>
37     <input type="checkbox" id="c1" checked>ポイントを受け取る<br>
38     <input type="checkbox" id="c2">メールマガジンを希望する<br>
39     性別：<input type="radio" name="r1" id="r1_1">男性,
40     <input type="radio" name="r1" id="r1_2" checked>女性<br>
41     年齢：
42     <select id="s1">
43       <option value="20"> 2 0 代</option>
44       <option value="30"> 3 0 代</option>
45       <option value="40"> 4 0 代</option>
46       <option value="50"> 5 0 代</option>
47     </select>
48     <br>
49     ユーザ名： <input type="text" id="t1" size="20" value="値の設定"><br>
50     パスワード：<input type="password" id="p1" size="20" value="mypswd"><br>
51     よろしければメッセージを入力してください：<br>
52     <textarea id="a1" cols="38" rows="5" ></textarea>
53     <br>
54     <input type="button" value="送信" onClick="f01( )">
55     <hr>
56     (内容確認表示) <br>
57     ユーザ名： <input type="text" id="t2" size="20"><br>
58     パスワード：<input type="password" id="p2" size="20"><br>
59     チェックボックス1の状態：<input type="text" id="c3"><br>
60     チェックボックス2の状態：<input type="text" id="c4"><br>
61     ラジオボタン1の状態： <input type="text" id="c5"><br>
62     ラジオボタン2の状態： <input type="text" id="c6"><br>
63     年齢選択の状態： <input type="text" id="s2"><br>
64     メッセージ内容：<br>
65     <textarea id="a2" cols="38" rows="5" ></textarea><br>
66   </body>
67 </html>

```

5.10 Image オブジェクト

Image オブジェクトは画像を保持するものであり、これに画像ファイルの内容を与えるには次のようにします。

```
var im = new Image();           ← Image オブジェクトの生成
im.src = (画像ファイルのパスもしくは URL);
```

Image オブジェクトの src プロパティに画像ファイルのパスや URL を与えることで画像の内容が読み込まれます。このオブジェクトは画像を扱う際によく用いられます。

Image オブジェクトに読み込んだ画像を タグに与えて表示するサンプル ImageTest1.html を示します。

サンプル：ImageTest1.html

```
1  <!DOCTYPE html>
2  <html lang="ja">
3  <head>
4  <meta charset="utf-8">
5  <title>Imageテスト</title>
6  <style>
7  #im {
8      width: 100px;
9  }
10 </style>
11 <script>
12 // 画像をまとめて読み込む処理
13 let images = new Array();           // Image オブジェクトの配列
14 for ( let i = 1; i < 4; i++ ) {
15     let fname = "ImageTest1_" + i + ".gif"; // ファイル名の設定
16     let image = new Image();
17     image.src = fname;                // 画像の読み込み
18     images.push(image);              // 配列に追加
19 }
20 // 画像表示を切り替える処理
21 let n = 0;
22 function f1() {
23     im.src = images[n].src;          // <img>要素の画像を切り替える
24     n++;
25     if ( n > 2 ) {
26         n = 0;
27     }
28 }
29 </script>
30 </head>
31 <body>
32 <img src="" alt="画像の表示場所です" id="im"><br>
33 <input type="button" value="next" onClick="f1()">
34 </body>
35 </html>
```

このサンプルでは Image オブジェクトの配列 images に複数の画像を読み込みます。「next」ボタンをクリックすると配列の要素の Image オブジェクトを タグに与えて表示します。(切り替え表示)

このサンプルを実行した例を図 51 に示します。

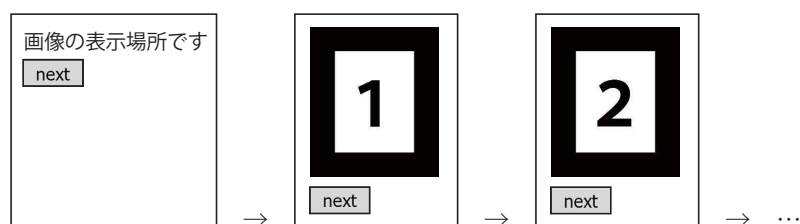


図 51: 「next」ボタンで画像が切り替わる

5.11 DOM のオブジェクト階層

HTML は **DOM**³³ に基づいた文書です。ここでは、HTML 要素（タグなど）や変数などが JavaScript の体系の中でどのように管理されているのかについて説明します。

5.11.1 window オブジェクト

まず、JavaScript の体系の中では全てのものは `window` オブジェクトの配下に属します。window オブジェクト配下のものとして最も重要なものの 1 つが HTML コンテンツで、これは `document` プロパティというもので

```
window.document
```

と、ドット「`.`」表記で記述します。これは通常の日本語表現である「～の…」という形によく似ており理解しやすいです。つまり上記の HTML コンテンツを表すオブジェクトは

「window の `document` プロパティ」

と解釈することができます。window.document には HTML コンテンツの全体が保持されており、ドット表記を更に連結することで、当該オブジェクトの配下のオブジェクトにアクセスすることができます。例えば HTML の `<head>` 要素や `<body>` 要素も window.document の直下に属しており、それぞれ、

```
window.document.head
```

```
window.document.body
```

という記述でアクセスできます。

5.11.2 document オブジェクト

window 直下のオブジェクトにアクセスする場合は基本的に先頭の「window」を省略することができ、HTML の `<head>` 要素や `<body>` 要素はそれぞれ、

```
document.head
```

```
document.body
```

と記述することができます。

JavaScript のグローバル変数も window の直下に属しています。

5.11.3 サンプルを用いた解説

次に示すサンプル JSObj01.html を用いて JavaScript のオブジェクト階層に関する確認をします。

サンプル：JSObj01.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>オブジェクトのテスト</title>
5 <script>
6 var v = 123;
7 </script>
8 </head>
9 <body>
10 <p id="t1">これはサンプルです。 </p>
11 <p id="t2" style="color:red;">2行目です。 </p>
12 </body>
13 </html>
```

³³DOM (Document Object Model)：文書要素（タグなど）の階層的構造のモデル。

このサンプルを Web ブラウザで表示して、Web ブラウザのコンソールを表示して対話的に色々と確認してみましょう。まず、グローバル変数 `v` に「123」という数値が与えられていますが、コンソールから

```
window.v
```

あるいは、

```
v
```

と入力してその値を確認することができます。(やってみてください)

次に<body>の内部にアクセスしてみます。HTML は **DOM** (Document Object Model) に沿った文書構造で、要素の階層構造(上下関係)が基本になっています。

5.11.3.1 子要素の取得：children プロパティ

HTML の要素が持つ**子要素**は、その要素の `children` プロパティから取得できます。これを確認するために Web ブラウザのコンソールで次のように入力してみます。

```
document.body.children
```

これは <body> 要素の子要素を全て取得するための記述であり、結果として2つの<p>要素を持つ `HTMLCollection` 型のオブジェクトが得られます。`HTMLCollection` オブジェクトは配列と同じように扱うことができ、

```
document.body.children[0]
```

と記述することで、その最初の要素

```
<p id="t1">これはサンプルです。</p>
```

にアクセスすることができます。

`children` プロパティが保持する子要素はあくまで HTML 要素(タグ)であり、テキスト(テキストノード)は保持しません。詳しくは「8 DOM に基づくプログラミング」(p.122)で解説します。

5.11.3.2 HTML 要素の属性とテキスト

HTML 要素は様々な**属性**を持ちます。属性とは、HTML タグの内部に記述されたもので、例えば今回のサンプル中の `p` 要素 `<p id="t2" style="color:red;">` では、「`id="t2"`」と「`style="color:red;"`」がそれぞれ属性(それぞれ `id` 属性と `style` 属性)です。

HTML 要素から指定した属性の値を取得するには `getAttribute` メソッドを使用します。

例. 属性値の取得 (Web ブラウザのコンソールでの実行)

```
e = document.body.children[1]  [Enter]  ← body の 2 つ目の子要素の取得
<p id="t2" style="color:red;">...  ← 子要素が得られた旨の応答
e.getAttribute("id")  [Enter]  ← id 属性の値の取得
"t2"  ← 値が取得できた
e.getAttribute("style")  [Enter]  ← style 属性の値の取得
"color:red;"  ← 値が取得できた
```

`getAttribute` メソッドについては p.63 「5.9.1.1 要素の属性と値について」で説明しましたが、次のように記述します。

```
対象要素.getAttribute("属性名")
```

HTML の `p` 要素のようにテキストを持つ要素からテキストを取得するには、その要素の `innerText` プロパティを参照します。

例. テキストの取得（先の例の続き：Web ブラウザのコンソールでの実行）

```
e.innerText  ←テキストの取得  
"2 行目です。" ←テキストが取得できた
```

`innerText` プロパティとよく似たものに `innerHTML` プロパティがあります。これは、要素がその配下に持つ HTML を 文字列 の形で取得するものです。例えば `body` 要素がその配下に持つ HTML の内容は次のようにして取得することができます。

例. `body` 内部の HTML を文字列の形で取得する（Web ブラウザのコンソールでの実行）

```
h = document.body.innerHTML  ←内部に含まれる HTML の取得  
'<p id="t1">これはサンプルです。</p>  
<p id="t2" style="color:red;">2 行目です。</p>'  
←↓得られた HTML（文字列形式）
```

以上に紹介したメソッドやプロパティを連鎖させて（再帰的に）使用することで、HTML の任意の要素の任意の属性やテキストにアクセスすることができます。

参考) `write`, `writeln` メソッド

`document` オブジェクトに対して `write`, `writeln` メソッドを実行することで、`<body>` 要素の内容を直接生成することができます。

サンプル：write01.html

```
1 <!DOCTYPE html>  
2 <html lang="ja">  
3 <head>  
4 <meta charset="utf-8">  
5 <title>writeメソッド</title>  
6  
7 <script>  
8 var txt = '<p>一行目</p>  
9 <p>二行目</p>  
10 <p>三行目</p>';  
11  
12 document.write( txt );  
13 </script>  
14 </head>  
15  
16 </html>
```

この HTML 文書の中には `<body>~</body>` がありませんが、Web ブラウザで実行すると `write` メソッドによりそれが生成され、その内容が変数 `txt` の値（3つの段落要素）になります。`writeln` メソッドは改行処理を行います。

後の「8 DOMに基づくプログラミング」（p.122）では、JavaScript で HTML の文書階層を編集するための機能について説明します。

6 canvas グラフィックス

JavaScript のプログラムでは、HTML コンテンツの中に配置された **canvas** という描画領域に対してグラフィックスを描画することができます。

6.1 canvas とコンテキスト

HTML コンテンツの中に canvas を配置するには<canvas>タグを記述します。例えば、id 名が hcvcs、横幅 600、高さ 400 の canvas を配置するには次のように記述します。

```
<canvas id="hcvcs" width="600" height="400"></canvas>
```

これで HTML 側の準備は完了で、あとは JavaScript のプログラムでグラフィックスを描画します。

まず、HTML コンテンツに設置された canvas を JavaScript のプログラム側で取得しておく必要があります。このためには `getElementById` メソッドで得られた canvas 要素を適当な変数に格納します。例えば、先の例のように HTML コンテンツ内に設置した canvas を取得するには次のように記述します。

```
var cvs = document.getElementById("hcvcs");
```

以後は、この `cvs` という変数に対して描画することになります。

6.1.1 コンテキスト

canvas に対する描画において、色や線種をはじめとする様々な設定は**コンテキスト**と呼ばれるものに与えます。また、コンテキストは各種の描画処理の対象にもなります。従って、canvas に描画するには、当該 canvas のコンテキストを予め取得する必要があります。このためには `getContext` メソッドに "2d" という引数を与えたものを canvas に対して実行します。(次の例)

```
var ctx = cvs.getContext("2d");
```

具体的な描画はこのコンテキスト（上の例では `ctx`）に対して行います。

6.2 図形の描画

コンテキストに対して描画を実行するメソッドを次に説明します。

6.2.1 線と塗り

JavaScript では折れ線を表現する**パス**という概念に沿ってで線を描画します。例えばコンテキスト `ctx` に対してパスを作成するには次のような手順を踏みます。

1) `beginPath` メソッドによるパス作成の開始

例. `ctx.beginPath();`

2) `moveTo` メソッドによる折れ線の開始点の決定

例. `ctx.moveTo(x, y);`

これにより開始点が座標 (x,y) になります。

3) `lineTo` メソッドによる折れ線の中継点の決定

例. `ctx.lineTo(x2, y2);`

これにより折れ線の次の中継点が座標 (x2,y2) になります。このステップを繰り返すことで、徐々に折れ線を作成します。

4) `closePath` メソッドによるパス作成の終了

例. `ctx.closePath();`

これにより、始点と終点が結ばれて、閉じたパスになります。

5) `stroke` メソッドによる折れ線描画の実行

例. `ctx.stroke();`

これにより canvas 上にパスが表示されます。

注) `closePath` を実行せずにパス作成を終了すると、開いたパスになります。

6.2.1.1 線の太さ, 色, ダッシュの設定

■ 太さの設定

パスの作成に先立って `lineWidth` 属性を設定することで線の太さを設定することができます。

例. `ctx.lineWidth = ft;`

こうすることで、描画するパスの線の太さが `ft` の値になります。

■ 色の設定

パスの作成に先立って `strokeStyle` 属性を設定することで線の色を設定することができます。色の設定は#で始まる 16 進数の表記³⁴ と、`"rgb(R,G,B)"` の表記 (R,G,B はそれぞれ赤, 緑, 青の強さ) も使用できます。

例. `ctx.strokeStyle = "rgb(R,G,B)";`

こうすることで、描画するパスの線が R,G,B で指定した色になります。各色は 0~255 の整数か、「0%」~「100%」の表記で与えます。

■ ダッシュの設定

破線や鎖線といったダッシュを描くには、コンテキストに対して `setLineDash` メソッドを実行することで、線の長さと空白の長さを設定します。

例. `ctx.setLineDash([5,5]);`

こうすることで、線の長さが 5、空白の長さが 5 のダッシュを設定できます。

6.2.1.2 塗りつぶしの色の設定

塗りつぶしに先立って `fillStyle` 属性を設定することで塗りの色を設定することができます。色の設定は#で始まる 16 進数の表記と、`"rgb(R,G,B)"` の表記 (R,G,B はそれぞれ赤, 緑, 青の強さ) も使用できます。

例. `ctx.fillStyle = "rgb(R,G,B)";`

こうすることで、R,G,B で指定した色による塗りつぶしになります。

`fill` メソッドの実行により、直前に描いた線 (パス) の内側を設定した色で塗りつぶすことができます。

例. `ctx.fill();`

6.2.2 円, 円弧, 楕円

円弧は、先に説明したパスの一部として描画します。すなわち、パスの作成中に `lineTo` メソッドの代わりに `arc` メソッドを用います。このメソッドは次のような形で記述します。

```
arc(x, y, radius, startAngle, endAngle, anticlockwise)
  x          : 中心の x 座標,
  y          : 中心の y 座標
  radius     : 半径の大きさ
  startAngle : 描画の開始角度 (単位: rad)
  endAngle   : 描画の終了角度 (単位: rad)
  anticlockwise : 描画方向
                (true → 反時計回り, false → 時計回り)
```

円を描く場合は、このメソッドを用いて開始角度を 0、終了角度を 2π に設定します。JavaScript には円周率が `Math.PI` として定義されており、 2π は `2*Math.PI` として得ることができます。canvas 内での角度の考え方については図 52 を参照してください。

楕円は縦横のスケールを変更して円を描画することで実現します。描画のスケール (縮尺) を変更するには、後で説明する `scale` メソッドを用います。

³⁴ 「3.2.2.4 色の設定」(p.11) 参照

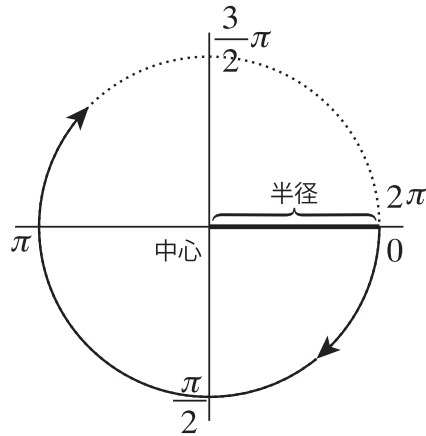


図 52: arc の角度の考え方 (単位: rad)

描画方向 (時計回り / 反時計回り) を問わず, 角度の体系はこれに従う

【サンプルプログラム】

パスと円弧を描画するプログラムの例 CanvasArc.html を次に示します.

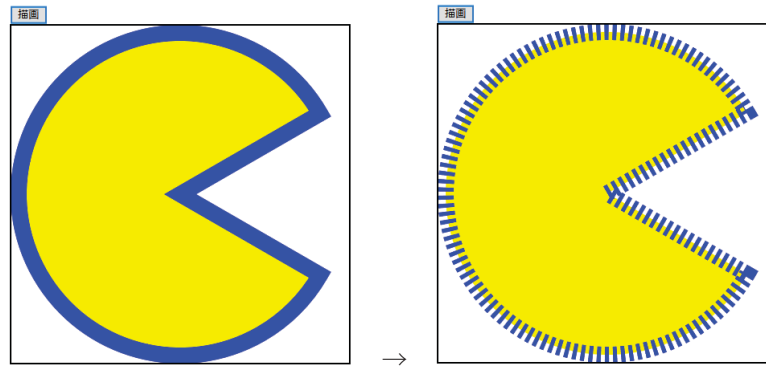
サンプル: CanvasArc.html

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>canvasのテスト </title>
6  <script>
7  // 変数
8  var cvs; // キャンバス
9  var ctx; // コンテキスト
10
11 // 初期化
12 function start() {
13     cvs = document.getElementById("hcvs"); // キャンバス
14     ctx = cvs.getContext("2d"); // コンテキスト
15
16     ctx.lineWidth = 20; // 線の太さの設定
17     ctx.strokeStyle = "#0000FF"; // 線の色の設定
18 // ctx.setLineDash( [5,5] ); // 線のダッシュの設定
19     ctx.fillStyle = "#FFFF00"; // 塗り色の設定
20
21     ctx.beginPath(); // パス作成の開始
22     ctx.moveTo(383.2,310); // 始点
23     ctx.lineTo(210,210); // 線の描画
24     ctx.lineTo(383.2,110); // 線の描画
25     ctx.arc(210,210,201,11*Math.PI/6,Math.PI/6,true); // 円弧
26     ctx.closePath(); // パスを閉じる
27     ctx.fill(); // パスの内部を塗りつぶす
28     ctx.stroke(); // パスの線を描画
29 }
30 </script>
31 </head>
32 <body>
33 <form name="fm">
34 <input type="button" value="描画" onClick="start()">
35 </form>
36 <canvas id="hcvs" width="420" height="420" style="border:2px solid #000000">
37 </canvas>
38 </body>
39 </html>

```

これを Web ブラウザで表示すると図 53 のようになります.



(a) 17 行目をコメントアウト

(b) 18 行目の「//」を外して実行

図 53: CanvasArc.html の実行例

6.2.3 四角形

四角形を描くメソッドには次の 2 つのがあります。

1) strokeRect メソッド：四角形の枠の描画

描く四角形の左上の位置の座標と、四角形のサイズを与えて実行します。

例. `ctx.strokeRect(10, 20, 100, 70);`

これを実行すると、座標 (10,20) の位置に横幅 100、高さ 70 の四角形の枠を描画します。

2) fillRect メソッド：四角形の塗りつぶし

描く四角形の左上の位置の座標と、四角形のサイズを与えて実行します。

例. `ctx.fillRect(10, 20, 100, 70);`

これを実行すると、座標 (10,20) の位置に横幅 100、高さ 70 の塗りつぶした四角形を描画します。

6.2.4 文字列描画

canvas に文字を描くための 2 種類のメソッドがあります。1 つは `fillText` メソッドで「塗りつぶし」の文字描画、もう 1 つは文字の輪郭を表示する `strokeText` メソッドです。

■ 使用例

```
ctx.fillText( txt, x, y );
ctx.strokeText( txt, x, y );
```

txt には表示する文字列を、x,y には表示位置の座標を指定します。

6.2.4.1 フォント、スタイル、サイズの指定

文字の描画に先立って、コンテキストの `font` 属性を設定することができます。これによりフォント、スタイル、サイズを指定した描画ができます。font 属性には

"スタイル サイズ フォント名"

を記述した文字列を与えます。例えば強調斜体の sans-serif フォントを 24 ポイントで設定する場合は次のようにします。

```
ctx.font = "italic bold 24px 'sans-serif'";
```

フォント名は p.10 「3.2.2 設定できるスタイルの種類」で紹介したもの³⁵ が使えます。スタイルは組み合わせたり省略することができます。

【サンプルプログラム】

四角形と文字を描画するプログラムの例 `CanvasRectText.html` を次に示します。

³⁵更に、p.168 「A.1 フォントの活用」で詳しく説明しています。

サンプル：CanvasRectText.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>canvasのテスト </title>
6 <script>
7 // 変数
8 var cvs; // キャンバス
9 var ctx; // コンテキスト
10
11 // 初期化
12 function start() {
13     cvs = document.getElementById("hcvs"); // キャンバス
14     ctx = cvs.getContext("2d"); // コンテキスト
15
16     // 四角形の描画
17     ctx.lineWidth = 20;
18     ctx.strokeStyle = "#000000"; // 黒
19     ctx.fillStyle = "#FFFF00"; // 黄
20     ctx.fillRect(10,10,380,180); // 塗りつぶし
21     ctx.strokeRect(10,10,380,180); // 輪郭
22
23     // 文字の描画
24     ctx.lineWidth = 3;
25     ctx.fillStyle = "#0000FF"; // 青
26     ctx.font = "bold 60px serif"; // 明朝体ボールド
27     ctx.fillText("文字列描画",40,80); // 塗りつぶし文字
28     ctx.strokeStyle = "#FF0000"; // 赤
29     ctx.font = "bold 60px sans-serif"; // ゴシック体ボールド
30     ctx.strokeText("文字列描画",40,160); // 輪郭文字
31 }
32
33 </script>
34 </head>
35 <body>
36 <form name="fm">
37 <input type="button" value="描画" onClick="start()">
38 </form>
39 <canvas id="hcvs" width="400" height="200">
40 </canvas>
41 </body>
42 </html>
```

これを Web ブラウザで表示すると図 54 のようになります。



図 54: CanvasRectText.html の実行例

6.3 画像ファイルの読み込みと表示

画像ファイルを読み込んで canvas に表示することができます。画像ファイルの読み込みに先立って、画像を保持する Image オブジェクトを作成します。Image オブジェクトの src 属性に画像のファイル名を設定すると、画像ファイルの中身がその Image オブジェクトに読み込まれます。

例. Image オブジェクト g を用意して画像データを読み込む例

```
var g = new Image();
g.src = "画像のファイル名";
```

画像を canvas に表示するには drawImage メソッドを使用します。

例. キャンバスの位置 (30,20) に Image オブジェクト g を表示する例

```
ctx.drawImage(g,30,20);
```

6.4 拡大縮小, 平行移動, 回転など

canvas への描画の大きさの比率を変えたり、回転角度を変えることが可能です。

6.4.1 拡大縮小

描画メソッドの実行に先立って、scale メソッドを実行することで、描画の比率を指定することができます。拡大縮小の中心は原点 (0,0) です。

例. 横 2 倍, 縦 1.5 倍に拡大する設定

```
ctx.scale( 2.0, 1.5 );
```

6.4.2 平行移動

translate メソッドを実行することで、canvas の座標の原点の位置を変えることができます。

例. 座標の原点を現在の (10,20) に移す

```
ctx.translate(10,20);
```

6.4.3 回転

rotate メソッドを実行することで、描画の角度を変えることができます。回転の中心は原点 (0,0) で、角度の単位は弧度 (ラジアン) です。また回転方向は時計回りです。

例. 時計回りに 45 度回転する設定

```
ctx.rotate(Math.PI / 4);
```

【サンプルプログラム】

平行移動や回転処理を加えて画像を描画するプログラムの例 CanvasPic.html を次に示します。

サンプル: CanvasPic.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>画像の表示</title>
6  <script>
7  // 変数
8  var cvs; // キャンバス
9  var ctx; // コンテキスト
10 var img = new Image();
11 img.src = "Balloon.jpg";
12
13 // 初期化
14 function start() {
```

```

15     cvs = document.getElementById("hcvs"); // キャンバス
16     ctx = cvs.getContext("2d"); // コンテキスト
17
18     ctx.drawImage(img,25,25);    // 画像表示
19
20     ctx.translate(290,100);    // 平行移動
21     ctx.rotate(Math.PI/8);    // 回転
22     ctx.drawImage(img,-75,-75); // 画像表示
23 }
24 </script>
25 </head>
26 <body>
27 <form name="fm">
28 <input type="button" value="描画" onClick="start()">
29 </form>
30 <canvas id="hcvs" width="400" height="200" style="border:2px solid #000000">
31 </canvas>
32 </body>
33 </html>

```

これを Web ブラウザで表示すると図 55 のようになります。

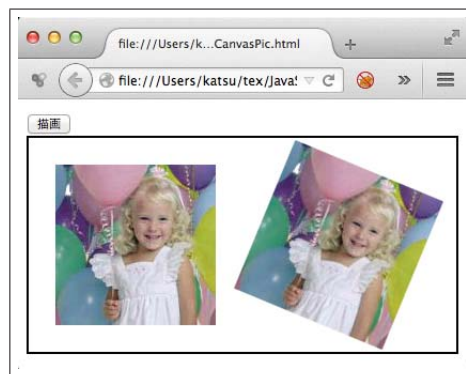


図 55: CanvasPic.html の実行例

6.4.4 コンテキストの保存と復元

`scale`, `transrate`, `rotate` などのメソッドによってキャンバスの描画の比率や原点、回転角を変更すると、現時点の状態に対して変更が加えられます。すなわち、比率や回転角、原点位置の設定は累積的におこなわれるので、場合によっては変更前の状態を保存しておき、描画終了後は元（初期の状態に）戻したいこともあります。

コンテキストの状態を変更する前に保存する場合は `save` メソッドを実行します。また保存した状態を現在のコンテキストに復元するには `restore` メソッドを実行します。

現在の時刻をアナログ時計の形で表示するサンプル `AnalogClock.html` を次に示します。この例では、時計の針の回転の中心と回転の角度を毎回再設定する方法を取っています。

サンプル：AnalogClock.html

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>時計</title>
6 <script>
7 // 時計の部品の画像
8 var clock1 = new Image();
9 var clock2 = new Image();
10 var clock3 = new Image();
11 var clock4 = new Image();
12 clock1.src = "Clock1.gif"; // 文字盤
13 clock2.src = "Clock2.gif"; // 長針
14 clock3.src = "Clock3.gif"; // 短針
15 clock4.src = "Clock4.gif"; // 秒針

```

```

16
17 // 時間
18 var datetime;
19 var h, m, s;
20 var timer;
21
22 // キャンバス用変数
23 var cvs, ctx;
24
25 var p2 = Math.PI * 2.0;
26
27 // 開始
28 function f01( ) {
29     cvs = document.getElementById("cnvs");
30     ctx = cvs.getContext("2d");
31
32     f02( );
33 }
34
35 // 時計表示 (全体)
36 function f02( ) {
37     datetime = new Date();
38     h = datetime.getHours() % 12;
39     m = datetime.getMinutes();
40     s = datetime.getSeconds() + datetime.getMilliseconds() / 1000.0;
41     clockDisp1( );
42     clockDisp2( );
43     clockDisp3( );
44     clockDisp4( );
45     document.fm.tf1.value = h + ":" + m + ":" + s;
46     timer = setTimeout("f02( )",10);
47 }
48
49 // 文字盤の表示
50 function clockDisp1( ) {
51     ctx.drawImage(clock1,0,0);
52 }
53
54 // 長針の表示
55 function clockDisp2( ) {
56     ctx.save();
57     ctx.translate(272,272);
58     ctx.rotate(p2/60.0*(m + s/60.0));
59     ctx.drawImage(clock2,-30,-270);
60     ctx.restore();
61 }
62
63 // 短針の表示
64 function clockDisp3( ) {
65     ctx.save();
66     ctx.translate(272,272);
67     ctx.rotate(p2/12.0*(h + m/60.0));
68     ctx.drawImage(clock3,-30,-208);
69     ctx.restore();
70 }
71
72 // 秒針の表示
73 function clockDisp4( ) {
74     ctx.save();
75     ctx.translate(272,272);
76     ctx.rotate(p2/60.0*s);
77     ctx.drawImage(clock4,-20,-270);
78     ctx.restore();
79 }
80
81 // 停止
82 function f03( ) {
83     clearTimeout(timer);
84 }

```



```

85 </script>
86 <body onLoad="f01( )">
87 <canvas id="cnvs" width="546" height="546"></canvas>
88 <form name="fm">
89 <input type="button" onClick="f03( )" value=" 停止">
90 <input type="text" name="tf1">
91 </form>
92 </body>
93 </html>

```

これを Web ブラウザで表示すると図 56 のようになります。

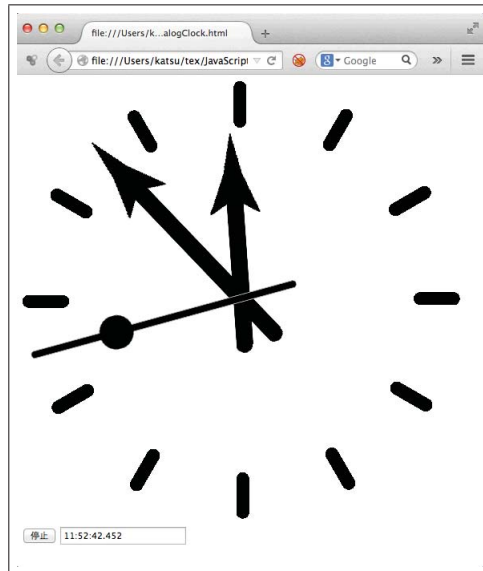


図 56: AnalogClock.html の実行例

このサンプル中で使用している `setTimeout` に関しては、第 7 章「Web アプリケーション開発に必要なこと」で説明します。

6.5 ピクセル（画素）の操作

canvas 上の画像は画素（ピクセル）の集まりです。ここでは canvas の画素を直接操作する方法について説明します。canvas のピクセル操作は概ね次の 3 種類の操作です。

■ 画素データ（ImageData）の生成／取得

画素データの集まりは `ImageData` オブジェクト として扱われます。既存の canvas から `ImageData` オブジェクトを取得したり、新規に `ImageData` オブジェクトを生成したりします。

■ 個々の画素に対する操作

`ImageData` オブジェクトの個々の画素の RGB 値を取得したり、個々の画素に RGB 値を設定したりします。

■ 画素データ（ImageData）の canvas への描画

`ImageData` オブジェクトの内容（画像）を実際に canvas に描画します。

6.5.1 ImageData オブジェクトについて

`ImageData` オブジェクトは画素データの集まりを格納するデータ構造です。具体的には `ImageData` オブジェクトの `data` 属性に個々の画素が収められています。data 属性は個々の画素の RGB 値と不透明度（アルファ値）を格納する配列です。例えば `img` という名の `ImageData` オブジェクトがある場合、その構造は表 9 のようなものです。

画素の順番は描画領域の左上からはじまり、右下に終わります。`ImageData` オブジェクトの `width` 属性には描画領域の横幅の画素数の値が格納されており、これを参照することで描画領域の座標を指定して画素にアクセスすること

表 9: 例. ImageData オブジェクト img の構造

	対象画素 (n 番目)	意味
img.data[0]	第 0 番目の画素	R 値
img.data[1]	第 0 番目の画素	G 値
img.data[2]	第 0 番目の画素	B 値
img.data[3]	第 0 番目の画素	アルファ値
img.data[4]	第 1 番目の画素	R 値
img.data[5]	第 1 番目の画素	G 値
img.data[6]	第 1 番目の画素	B 値
img.data[7]	第 1 番目の画素	アルファ値
img.data[8]	第 2 番目の画素	R 値
img.data[9]	第 2 番目の画素	G 値
img.data[10]	第 2 番目の画素	B 値
img.data[11]	第 2 番目の画素	アルファ値
:	:	:

ができます。例えば次に示すような関数 `setPixel` を定義しておくと、座標と RGB 値、アルファ値を指定して画素を描画することができます。

例. ImageData オブジェクト img の座標 (x,y) に RGB 値 (r,g,b) とアルファ値 (a) を設定する関数

```
function setPixel(img,x,y,r,g,b,a) {
  img.data[(img.width * y + x)*4] = r;
  img.data[(img.width * y + x)*4 + 1] = g;
  img.data[(img.width * y + x)*4 + 2] = b;
  img.data[(img.width * y + x)*4 + 3] = a;
}
```

アルファ値は不透明度を表す 0～255 の数値で、値が大きくなるほど不透明度が大きくなります。通常は 255 にしておきます。

課題) ImageData オブジェクトの指定した座標のピクセル値 (RGB 値とアルファ値) を取得する関数 `getPixel` の実装について考えてみてください。

6.5.2 ピクセル描画の手順

1. ImageData の新規生成

`createImageData` メソッドを使って ImageData オブジェクトを新規に生成することができます。例えば、`canvas` のコンテキスト `ctx` がある場合、次のようにします。

例. `var img = ctx.createImageData(400,300);`

これにより、横幅 400 高さ 300 のサイズの ImageData オブジェクト `img` が生成されます。

2. ImageData オブジェクトに対する操作

`data` 属性から画素の値を参照したり、`data` 属性に値を設定したりします。画素の値の設定は、先に示した関数例 `setPixel`などを参考にしてください。

3. ImageData オブジェクトを canvas に描画

`putImageData` メソッドを使用して `canvas` に ImageData オブジェクトの内容を描画します。例えば `canvas` のコンテキスト `ctx` に対して ImageData オブジェクトを描画するには次のようにします。

例. ctx.putImageData(img,0,0);

これにより, ctx の座標 (0,0) の位置に img の内容を描画します.

6.5.3 サンプルプログラム

【サンプル (1)】

画素を直接操作して描画するプログラムのサンプル PixGraphics01.html を示します.

サンプル: PixGraphics01.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>画素のテスト</title>
6  <script>
7  function setPixel(img,x,y,r,g,b,a) {
8      img.data[(img.width * y + x)*4] = r;
9      img.data[(img.width * y + x)*4 + 1] = g;
10     img.data[(img.width * y + x)*4 + 2] = b;
11     img.data[(img.width * y + x)*4 + 3] = a;
12 }
13
14 function f01( ) {
15     var cvs = document.getElementById("hcvs");
16     var ctx = cvs.getContext("2d");
17     var img = ctx.createImageData(400,300);
18     var x, y, r, g, b;
19     var xs = 0;
20     for ( r = 0; r < 256; r+=255 ) {
21         for ( g = 0; g < 256; g+=255 ) {
22             for ( b = 0; b < 256; b+=255 ) {
23                 for ( x = xs; x < xs+50; x++ ) {
24                     for ( y = 0; y < 300; y++ ) {
25                         setPixel(img,x,y,r,g,b,255);
26                     }
27                 }
28                 xs += 50;
29             }
30         }
31     }
32     ctx.putImageData(img,0,0);
33 }
34 </script>
35 </head>
36 <body>
37 <form name="fm">
38 <input type="button" value="描画" onClick="f01( )">
39 </form>
40 <canvas id="hcvs" style="border:1px solid #000000" width="400" height="300">
41 </canvas>
42 </body>
43 </html>
```

これを Web ブラウザで表示すると図 57 のようになります.

【サンプル (2)】

スライダから値を取得して 3 原色の合成を実演するサンプルプログラム ColorTest.html を示します.

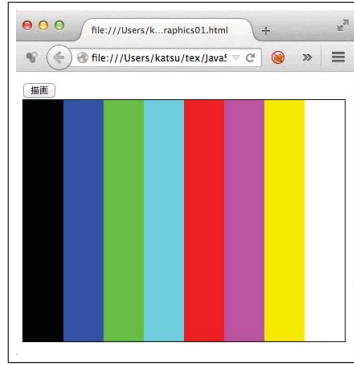


図 57: PixGraphics01.html の実行例

サンプル：ColorTest.html

```

1  <html>
2  <head>
3  <meta charset="utf-8">
4  <title>Color Test</title>
5  <script>
6  var cvs, ctx;
7  var cR=0, cG=0, cB=0, cA; // 色の値 (16進)
8  var aR=0, aG=0, aB=0;    // 色の値 (10進)
9
10 function init() {
11     cvs = document.getElementById('c');
12     ctx = cvs.getContext('2d');
13     document.fm.ar.value = 0;
14     document.fm.ag.value = 0;
15     document.fm.ab.value = 0;
16     cchange1();
17 }
18
19 function cchange1() {
20     cR = parseInt(document.fm.cr.value / 100 * 255);
21     cG = parseInt(document.fm.cg.value / 100 * 255);
22     cB = parseInt(document.fm.cb.value / 100 * 255);
23     aR = cR;
24     aG = cG;
25     aB = cB;
26     document.fm.ar.value = aR;
27     document.fm.ag.value = aG;
28     document.fm.ab.value = aB;
29     cchange0();
30 }
31
32 function cchange2() {
33     cR = parseInt(document.fm.ar.value);
34     cG = parseInt(document.fm.ag.value);
35     cB = parseInt(document.fm.ab.value);
36     document.fm.cr.value = cR / 255 * 100;
37     document.fm.cg.value = cG / 255 * 100;
38     document.fm.cb.value = cB / 255 * 100;
39     cchange0();
40 }
41
42 function cchange0() {
43     if ( cR > 15 ) {
44         cR = cR.toString(16);
45     } else {
46         cR = "0" + cR.toString(16);
47     }
48     document.fm.tr.value = cR;
49     if ( cG > 15 ) {
50         cG = cG.toString(16);
51     } else {
52         cG = "0" + cG.toString(16);
53     }

```

```

54     document.fm.tg.value = cG;
55     if ( cB > 15) {
56         cB = cB.toString(16);
57     } else {
58         cB = "0" + cB.toString(16);
59     }
60     document.fm.tb.value = cB;
61     cA = "#" + cR + cG + cB;
62     ctx.fillStyle = cA;    // タイルの色
63     ctx.fillRect(0,0,199,199);
64 }
65 </script>
66 </head>
67 <body onLoad="init( )">
68 <h2>Color Test</h2>
69 <form name="fm">
70 <table>
71 <tr><td>色</td><td>レベル</td>
72 <td>16進</td><td>10進(0~255)</td></tr>
73 <tr><td>R</td><td>
74 <input type="range" name="cr" value="0" onInput="cchange1()">
75 </td><td>
76 <input type="text" name="tr" size="3" readonly="readonly">
77 </td><td>
78 <input type="number" name="ar" size="3" min="0" max="255" onChange="cchange2()">
79 </td></tr><tr><td>G</td><td>
80 <input type="range" name="cg" value="0" onInput="cchange1()">
81 </td><td>
82 <input type="text" name="tg" size="3" readonly="readonly">
83 </td><td>
84 <input type="number" name="ag" size="3" min="0" max="255" onChange="cchange2()">
85 </td></tr>
86 <tr><td>B</td><td>
87 <input type="range" name="cb" value="0" onInput="cchange1()">
88 </td><td>
89 <input type="text" name="tb" size="3" readonly="readonly">
90 </td><td>
91 <input type="number" name="ab" size="3" min="0" max="255" onChange="cchange2()">
92 </td></tr>
93 </table>
94 </form>
95 <canvas id="c" style="position:relative; border:3px solid #999" width="200" height
    ="200">
96 </canvas>
97 </body>
98 </html>

```

これを Web ブラウザで表示すると図 58 のようになります。

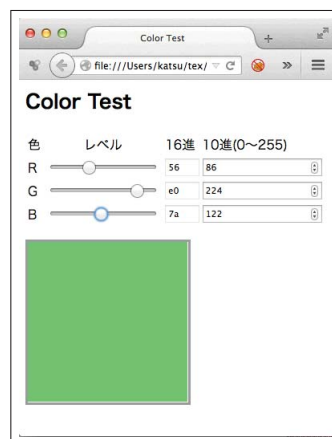


図 58: PColorTest.html の実行例

7 Webアプリケーション開発に必要なこと

ここでは、実用的な Web アプリケーションを作成するために必要となる事柄について説明します。基本的には、コンテンツの構成要素を HTML で作成して JavaScript でコンテンツの動作を記述します。このときに重要なのが**イベント駆動**の考え方です。先の章でもこれについて簡単に触れましたが、ここではより具体的な内容について解説します。

7.1 イベント駆動型プログラミング

GUI に基づくアプリケーションプログラムは、ユーザの GUI に対する操作を起点にして動作を始めます。この「処理の起点」が JavaScript プログラムにおける**イベント**で、様々な種類があります。代表的なイベントとして

- マウスイベント

マウスが操作されたことを意味するイベント。

- キーボードイベント

キーボードが操作されたことを意味するイベント。

- ウィンドウイベント

ウィンドウの状態が変化したことを意味するイベント。

などがあります。

HTML コンテンツの GUI コンポーネントを操作することでイベントは発生します。ボタンなどはそのわかりやすい例で、クリックすることで `click` イベントが発生 (`onClick`) し、それに対して起動する JavaScript プログラムを関連付ける例を先に見ました。

HTML コンテンツ内の要素には色々なものがあり、そのままではイベントに反応しないものも多いです。例えば Web コンテンツを表示するウィンドウなどがその例で、ウィンドウ上でマウスが動いたことを検知するためには、対象となるウィンドウに対して「イベントを検知するように」設定しなければなりません。例えば、マウスのドラッグ操作によって `canvas` に絵を描くような動作を実現するには、

「マウスが動いた」→「`canvas` に描画する」

という形でプログラムを作成します。このためには、ウィンドウが常にマウスの動きを感知する必要があります。

GUI コンポーネントやウィンドウがイベントを検知する仕組みを**イベントリスナ**といい、ユーザからの働きかけを検知する、一種のセンサーと見ることができます。

JavaScript で扱う主なイベントを次の「7.1.1 イベントの種類」に挙げます。

7.1.1 イベントの種類

JavaScript で扱うことができる主なイベントを表 10～14 に挙げます。

表 10: ページの読み込み、離脱など

イベント／検知	説 明
<code>load</code> / <code>onLoad</code>	ページや画像の読み込みが完了した時に発生
<code>unload</code> / <code>onUnload</code>	ウィンドウを閉じた時や他のページに切り替えた時、あるいはページをリロード（更新）した時に発生

表 11: マウスの操作

イベント／検知	説 明
click / onClick	要素やリンクをクリックした時に発生
dblclick / onDoubleClick	要素をダブルクリックした時に発生
mouseout / onMouseOut	マウスポインタが離れた時に発生
mouseover / onMouseOver	マウスポインタが乗った時に発生
mouseup / onMouseUp	クリックしたマウスボタンを放した時に発生
mousedown / onMouseDown	マウスでクリックした時に発生
mousemove / onMouseMove	マウスを動かしている時に発生
contextmenu / onContextMenu	マウスの右ボタンがクリックされた時に発生

表 12: キーボードの操作

イベント／検知	説 明
keydown / onKeyDown	キーを押した時に発生
keyup / onKeyUp	押していたキーを放した時に発生
keypress / onKeyPress	キーを押している時に発生

表 13: 変更・選択

イベント／検知	説 明
resize / onResize	ウィンドウのサイズが変更された時に発生
scroll / onScroll	スクロールが起こった時に発生
select / onSelect	テキストが選択された時に発生
blur / onBlur	ページやフォーム要素からフォーカスが外れた時に発生
focus / onFocus	ページやフォーム要素にフォーカスが当たった時に発生
change / onChange input / onInput	GUI コンポーネントの値が変化した時に発生 change/onChange は値の変化が完全に終わった段階で発生するが、 onInput は「変わりつつある」段階で発生する。 (例えば、スライダをマウスで変更している最中など)

表 14: その他

イベント／検知	説 明
submit / onSubmit	フォームを送信しようとした時に発生
reset / onReset	フォームがリセットされた時に発生
abort / onAbort	画像の読み込みを中断した時に発生
error / onError	画像の読み込み中にエラーが発生した時に発生

7.1.2 プログラムの記述と実行の流れ

JavaScript のプログラムの作成と実行は基本的には、

- 1) function 文の記述によるプログラムの作成
- 2) イベントを起点に 1) で用意したプログラムを起動

という流れになります。しかしイベントリスナの設定といった初期設定の作業はどのタイミングでするのでしょうか。

ここでは、マウスのドラッグで canvas に絵を描く簡単なプログラム GUItest1.html をとりあげ、イベントリスナ

の設定とマウスイベントの取得の方法を例示し、JavaScript プログラムの記述と実行の流れについて説明します。

サンプル：GUltest1.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>マウスによる描画</title>
6  <script>
7  // canvasとコンテキスト用の変数の用意
8  var cvs;
9  var ctx;
10 // 「ドラッグ中」を意味するフラグの用意
11 var drawing = false;
12 // マウス位置を保持する変数の用意
13 var x; // マウス座標X
14 var y; // マウス座標Y
15
16 // ウィンドウの表示が終わった直後に実行する処理
17 // 「windowのloadが終わったときにinitを実行する」
18 window.addEventListener("load", init, false);
19
20 // initの本体
21 function init(ev) {
22     // canvasとコンテキストの取得
23     cvs = document.getElementById('hcvs');
24     ctx = cvs.getContext('2d');
25
26     // コンテキストの設定
27     ctx.lineJoin = "round"; // 角を丸く
28     ctx.lineCap = "round"; // 線の終端を丸く
29     ctx.lineWidth = 10; // 線の幅 (10ポイント)
30     ctx.strokeStyle = "#FF0000"; // 線の色 (赤に設定)
31
32     // イベントリスナの設定
33     // cvsにmousedownイベントが起こったときstart()を呼び出す
34     cvs.addEventListener("mousedown", start, false);
35     // cvsにmousemoveイベントが起こったときmove()を呼び出す
36     cvs.addEventListener("mousemove", move, false);
37     // cvsにmouseupイベントが起こったときstop()を呼び出す
38     cvs.addEventListener("mouseup", stop, false);
39 }
40
41 // マウスボタンが押されたときに実行する処理
42 // (cvsにmousedownイベントが起こったときの処理)
43 function start(ev) {
44     x = ev.layerX; // マウス位置のX座標の取得
45     y = ev.layerY; // マウス位置のY座標の取得
46
47     ctx.beginPath(); // パス描画の開始
48     ctx.moveTo(x,y); // マウスボタンを押した位置を開始点とする
49
50     drawing = true; // ドラッグ中フラグを立てる
51 }
52
53 // マウスが動いた (位置が変わった) ときに実行する処理
54 // (cvsにmousemoveイベントが起こったときの処理)
55 function move(ev) {
56     if ( drawing ) {
57         x = ev.layerX; // マウス位置のX座標の取得
58         y = ev.layerY; // マウス位置のY座標の取得
59         ctx.lineTo(x,y); // 直前のマウス位置から現在のマウス位置まで直線で繋ぐ
60         ctx.stroke(); // 描画の実行
61     }
62 }
63
64 // マウスボタンが上がったときに実行する処理
65 // (cvsにmouseupイベントが起こったときの処理)
66 function stop(ev) {
```



```

67     ctx.closePath(); // パスを閉じる
68     drawing = false; // ドラッグ中フラグを落とす
69 }
70 </script>
71 </head>
72 <body>
73 <canvas id="hcvs" style="border:3px solid #000000" width="640" height="480">
74 </canvas>
75 </body>
76 </html>

```

これを Web ブラウザで表示すると図 59 のようになります。

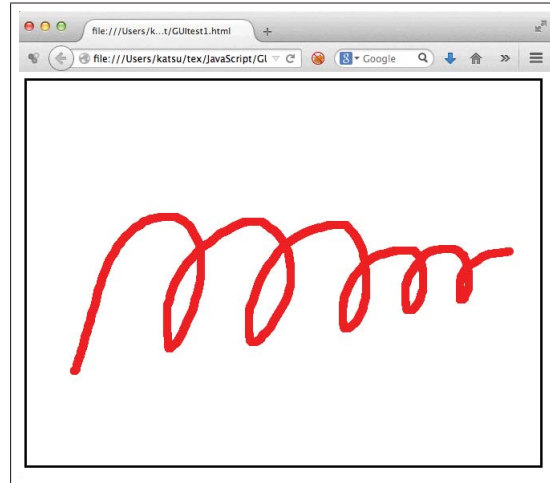


図 59: GUItest1.html の実行例
マウスのドラッグにしたがって絵を描く

7.1.2.1 イベントリスナの設定

Web ブラウザが HTML コンテンツを読み込んで表示した瞬間に発生するイベントが"load"で、これが発生したことを"onLoad" で表します。JavaScript のプログラミングにおいて、イベントリスナの設定といった初期設定はこの load イベントを機に行うのが良いです。

Web ブラウザのウィンドウは JavaScript では window として扱います。プログラムの 18 行目に

```
window.addEventListener("load", init, false);
```

という記述があります。この行では window のイベントリスナに対して load イベントが起こった場合に init というプログラムを起動するように設定しています。すなわち、Web ブラウザがこのコンテンツを読み込んだ直後に init が起動することになります。

addEventListener の第三引数は、DOM 階層間でのイベント伝搬に関するもの³⁶で、通常は false にしておきます。

init のプログラムでは、canvas とコンテキストの取得、それに canvas に対するイベントリスナの設定のための処理を実行します。34,36,38 行の部分が canvas にイベントリスナを設定するもので、この設定により、canvas はマウスの動きやクリックに対して反応するようになります。

7.1.2.2 イベントオブジェクト

イベントに対して起動されるプログラムの引数にはイベントオブジェクトというものが渡されます。今回のプログラムでは、init, start, stop, move のプログラムの記述の際に、仮引数として ev というものが記述されています。これは、当該関数がイベント発生によって起動される際に、そのイベントに関する各種の情報（マウスの現在位置など）を受け取るものです。例えば start, stop のプログラムの記述の中にありますが、ev.layerX や ev.layerY としてマウスの位置情報を取得すること（後述）ができます。イベントオブジェクトのプロパティ（属性）として特に基

³⁶これに関しては「入門」の範囲を超えるので本書では解説を割愛します。

本的なものとして表 15 のようなものがあります。

表 15: イベントオブジェクト `e` の基本的なプロパティ

参照方法	説明
<code>e.target</code>	当該イベントが発生した要素
<code>e.type</code>	当該イベントのタイプ
<code>e.timeStamp</code>	当該イベントが発生した時刻

【もっと簡便な記述】

コンテンツが Web ブラウザに読み込まれたときに発生する `load` イベントを受けてスクリプトを発動する方法を上
に示しましたが、次のような、より簡便な記述も可能です。

```
onload = function() {  
    (処理内容)  
}
```

このように記述することで、コンテンツが Web ブラウザに読み込まれたときに“処理内容”の部分のスクリプトが
実行されます。

イベントリスナを登録する方法に関しても、より簡便な記述方法があります。例えばドキュメント要素 `e1` に、
`onMouseMove` イベントに対応するイベントリスナを登録するには次のように³⁷ 記述します。

```
e1.onmousemove = function(e) {  
    (処理内容)  
}
```

このように記述することで、要素 `e1` 上でマウスが動いた時に“処理内容”の部分のスクリプトが実行されます。こ
の例では、イベントに関する情報は**イベントオブジェクト `e`** が保持しています。

更に、上と同じ処理をアロー関数を使って次のように記述することもできます。

```
e1.onmousemove = (e) => {  
    (処理内容)  
}
```

このように、イベントハンドリングの登録方法は色々あります。後の「7.1.2.6 サンプル：イベントハンドリング登
録のための各種の方法」(p.91) でこれら各種の方法を実装したサンプルを示します。

7.1.2.3 HTML 要素にイベントハンドリングを記述する方法

HTML のタグの中にイベントハンドリングを記述する方法もあり、より簡易な形でプログラムを記述することがで
きます。具体的には HTML 要素（タグ）の属性として

イベント＝”JavaScript の関数呼び出し”

と記述することで、当該イベントが発生したときに JavaScript の関数を呼び出します。例えば、HTML の `<body>` タ
グを

```
<body onLoad="関数呼び出し">
```

のように記述することで、`onLoad` のハンドリングができます。また、呼び出す関数が引数として**イベントオブジェク
ト**を受け取る場合は、呼び出す際の引数に `arguments[0]` を渡すと、それがイベントオブジェクトとして扱われます。

³⁷ イベントリスナを登録する対象の要素（タグ）によっては、この方法が使えない場合があります。その場合は、`addEventListener` を使用
してください。これに関しては「7.1.2.1 イベントリスナの設定」(p.87) で解説しています。

例. input 要素の値の変化によるイベントハンドリング
<input type="text" onChange="f1(arguments[0])">

この input 要素の値が変化した際に関数 f1 が呼び出されますが、その際の引数にイベントオブジェクトを与えています。

7.1.2.4 マウスの位置の取得

先のサンプル GUItest1.html では、イベント ev からマウスポインタの座標を取り出すのに layerX, layerY プロパティを参照していました。それらのプロパティは、対象となる要素の上の座標（相対的な座標）を保持していますが、その他にもマウスポインタの位置情報を保持するプロパティがあります。表 16 にマウスポインタの座標を保持するプロパティを挙げます。

表 16: イベント e のマウスポインタの座標を保持するプロパティ

メソッド	意味
e.pageX, e.pageY	HTML コンテンツの左上を原点とする横位置と縦位置
e.clientX, e.clientY	Web ブラウザの描画領域の左上を原点とする横位置と縦位置
e.screenX, e.screenY	ディスプレイ自体の左上を原点とする横位置と縦位置
e.layerX, e.layerY	HTML 要素の左上を原点とする横位置と縦位置 (ただし、対象要素のスタイル position に static 以外のものが設定されている場合)

取得される座標値の単位は px です。(当該ディスプレイデバイスのピクセル数)

7.1.2.5 要素のサイズの取得

コンテンツ要素の横幅と高さをピクセル単位 (px) で保持するプロパティ (表 17) があります。HTML コンテンツの配置はポイント単位 (pt) でデザインの方が汎用性が高く、異なるデバイスや Web ブラウザで表示する場合の統一感が保てます。ただし、マウスポインタの座標位置といった位置情報を JavaScript で取得する際、ピクセル単位の値として得られることが多く、コンテンツ要素のサイズをピクセル単位で取得する必要がある場合があります。

表 17: 要素 e1 のサイズをピクセル単位で保持するプロパティ

メソッド	意味
e1.offsetWidth, e1.offsetHeight	要素の横幅と高さ (padding と border 含む)
e1.clientWidth, e1.clientHeight	要素の横幅と高さ (padding 含む)
e1.scrollWidth, e1.scrollHeight	内容の横幅と高さ (padding 含む) (これはスクロール領域を持つ要素が対象の場合、スクロール表示される内容の全体のサイズを取得する)

取得される値の単位は px です。(当該ディスプレイデバイスのピクセル数)

マウスポインタの座標位置や要素のピクセルサイズを取得するサンプル GUItest2.html を次に示します。

サンプル: GUItest2.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>値の取得</title>
5 <style>
6 div {
7     position: absolute;
8     width: 100pt;
9     height: 50pt;
10    border: solid 1pt black;
11 }
12 #d1 {
13     top: 0pt;
14     left: 0pt;
```

```

15 }
16 #d2 {
17     top: 51pt;
18     left: 101pt;
19 }
20 #d3 {
21     top: 110pt;
22     left: 0pt;
23     width: 400pt;
24     border: solid white;
25 }
26 span {
27     width: 100pt;
28 }
29 </style>
30 <script>
31 var x,y, sx,sy, lx,ly, cx,cy;
32
33 document.onmousemove = function (e) {
34     // コンテンツ上の位置
35     x = e.pageX;
36     y = e.pageY;
37     pX.value = x;
38     pY.value = y;
39     // デバイス（スクリーン）上の位置
40     sx = e.screenX;
41     sy = e.screenY;
42     sX.value = sx;
43     sY.value = sy;
44     // Webブラウザの描画領域上の位置
45     cx = e.clientX;
46     cy = e.clientY;
47     cX.value = cx;
48     cY.value = cy;
49 }
50
51 onload = function() {
52     var d2 = document.getElementById("d2");
53     d2.onmousemove = function(e) {
54         // 対象オブジェクト上の相対位置
55         lx = e.layerX;
56         ly = e.layerY;
57         lX.value = lx;
58         lY.value = ly;
59     }
60     // 対象オブジェクトのサイズを取得（ピクセル数）
61     document.getElementById("sizex").value = d2.offsetWidth;
62     document.getElementById("sizey").value = d2.offsetHeight;
63 }
64 </script>
65 </head>
66 <body>
67 <div id="d1">100x50pt<br>id=d1</div>
68 <div id="d2">100x50pt<br>id=d2</div>
69 <div id="d3">
70     clientX/Y
71     <input type="text" id="cX">
72     <input type="text" id="cY"><br>
73     screenX/Y
74     <input type="text" id="sX">
75     <input type="text" id="sY"><br>
76     layerX/Y
77     <input type="text" id="lX">
78     <input type="text" id="lY"><br>
79     pageX/Y
80     <input type="text" id="pX">
81     <input type="text" id="pY"><br>
82     size of d2 by px:
83     <input type="text" id="sizex">

```

```

84     <input type="text" id="sizey">
85 </div>
86 </body>
87 </html>

```

これを Web ブラウザで表示すると図 60 のようになります。

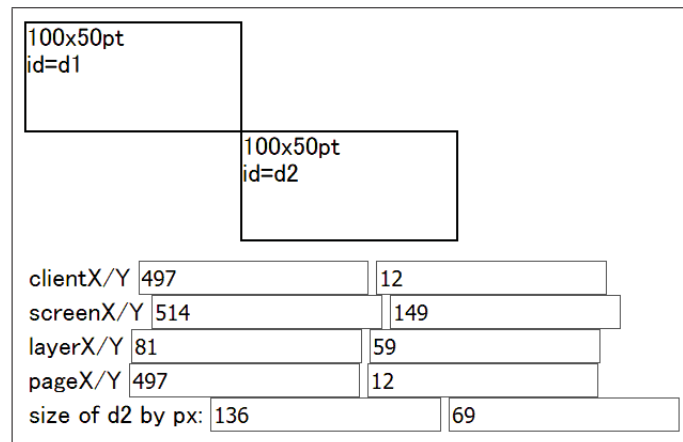


図 60: GUItest2.html の実行例

layerX, layerY で id="d2" の要素の上の相対的な座標位置を取得しています。

7.1.2.6 サンプル：イベントハンドリング登録のための各種の方法

マウス移動のイベントハンドリング（onmousemove のハンドリング）を div 要素に登録する方法に関して、各種のサンプルを例示します。

div 要素で描画された矩形の上でマウスを移動すると下部に配置されたテキストフィールド（input 要素）にマウスの座標を表示するプログラム（図 61）について考えます。

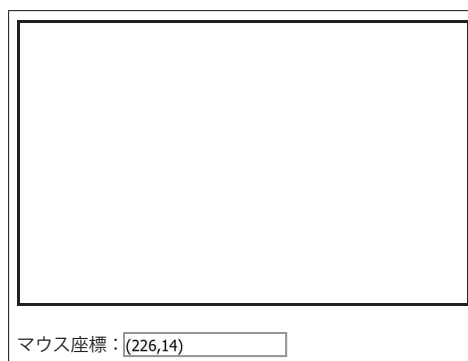


図 61: マウスの移動を検出するプログラム

以下に、イベントハンドリングの登録方法の異なるサンプルを 3 種類示します。（働きは全て同じ）

イベントをハンドリングする関数 msmv を定義して、それを div 要素の onmousemove プロパティに登録する形の実装を GUItest3-3.html に示します。

サンプル：GUItest3-3.html

```

1  <!DOCTYPE html>
2  <html lang="ja">
3  <head>
4  <meta charset="utf-8">
5  <title>マウスイベント</title>
6  <style>
7  #dvl {
8      width: 400px;
9      height: 250px;
10     border: solid 2px black;

```

```

11 }
12 </style>
13 <script>
14 var dv1, tx1;
15 function msmv(e) {          // マウス移動のイベントハンドリング用コールバック
16     var x = e.layerX;
17     var y = e.layerY;
18     tx1.value = "("+x+", "+y+")";
19 }
20 function f1() {              // 初期化関数（イベントハンドリングの登録など）
21     dv1 = document.getElementById("dv1");
22     tx1 = document.getElementById("tx1");
23     dv1.onmousemove = msmv;    // イベントハンドリングの登録
24 }
25 </script>
26 </head>
27 <body onLoad="f1()">
28 <div id="dv1"></div><br>
29 マウス座標：<input type="text" id="tx1">
30 </body>
31 </html>

```

このサンプルでは 23 行目でイベントハンドリングを登録しています。

次に、function の式を div 要素の onmousemove プロパティに登録する形の実装を GUItest3-2.html に示します。

サンプル：GUItest3-2.html

```

1 <!DOCTYPE html>
2 <html lang="ja">
3 <head>
4 <meta charset="utf-8">
5 <title>マウスイベント</title>
6 <style>
7 #dv1 {
8     width: 400px;
9     height: 250px;
10    border: solid 2px black;
11 }
12 </style>
13 <script>
14 var dv1, tx1;
15 function f1() {              // 初期化関数（イベントハンドリングの登録など）
16     dv1 = document.getElementById("dv1");
17     tx1 = document.getElementById("tx1");
18     dv1.onmousemove = function (e) {    // イベントハンドリングの登録
19         var x = e.layerX;
20         var y = e.layerY;
21         tx1.value = "("+x+", "+y+")";
22     }
23 }
24 </script>
25 </head>
26 <body onLoad="f1()">
27 <div id="dv1"></div><br>
28 マウス座標：<input type="text" id="tx1">
29 </body>
30 </html>

```

このサンプルでは 18～22 行目でイベントハンドリングを登録しています。

次に、アロー関数を div 要素の onmousemove プロパティに登録する形の実装を GUItest3-3.html に示します。

サンプル：GUItest3-1.html

```

1 <!DOCTYPE html>
2 <html lang="ja">
3 <head>
4 <meta charset="utf-8">

```

```
5 <title>マウスイベント</title>
6 <style>
7 #dv1 {
8     width: 400px;
9     height: 250px;
10    border: solid 2px black;
11 }
12 </style>
13 <script>
14 var dv1, tx1;
15 function f1() {          // 初期化関数（イベントハンドリングの登録など）
16     dv1 = document.getElementById("dv1");
17     tx1 = document.getElementById("tx1");
18     dv1.onmousemove = (e) => {          // イベントハンドリングの登録
19         var x = e.layerX;
20         var y = e.layerY;
21         tx1.value = "("+x+", "+y+")";
22     }
23 }
24 </script>
25 </head>
26 <body onLoad="f1()">
27 <div id="dv1"></div><br>
28 マウス座標:<input type="text" id="tx1">
29 </body>
30 </html>
```

このサンプルでは 18～22 行目でイベントハンドリングを登録しています。

7.2 メディアデータ（音声、動画）の再生

ここでは、音声ファイルや動画ファイルを読み込んで再生する方法について説明します。

7.2.1 audio/video タグを用いたメディアデータの読み込み

HTML5 には audio タグと video タグがあり、これらを用いてメディアデータをコンテンツ内に配置します。これらのタグの基本的な使い方を次に示します。

```
<audio src="音声のファイル名">～</audio>
```

```
<video src="動画のファイル名">～</video>
```

これらのタグは HTML5 のタグであり、正しく表示するには HTML5 に対応したブラウザを使う必要があります。HTML5 未対応のブラウザでコンテンツを表示した場合は”～”の部分が表示されます。

audio/video タグで配置されたメディアデータを JavaScript で操作するためには、getElementById メソッドを用いるなどしてメディアデータを JavaScript のオブジェクトとして取得する³⁸ 必要があります。

```
var a = getElementById("audio/video タグの id 名");
```

上の例では変数 a にメディアデータが取得されるので、これに対して操作することになります。

7.2.2 再生と一時停止

メディアを再生するには次の例のように play メソッドを使用します。

```
a.play();
```

またメディアの再生を一時停止するには、次の例のように pause メソッドを使用します。

```
a.pause();
```

音声の再生と一時停止を実現したサンプルを SoundTest01.html に示します。

サンプル：SoundTest01.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>サウンドテスト</title>
6  <script>
7  // 一時停止
8  function f01( ) {
9      var a = document.getElementById("a01");
10     a.pause();
11 }
12
13 // 再生
14 function f02( ) {
15     var a = document.getElementById("a01");
16     a.play();
17 }
18 </script>
19 </head>
20 <body>
21 <audio id="a01" src="technomics1.mp3">
22 <p>※ audio タグ対応のブラウザが必要です。 </p>
23 </audio>
24 <form name="fm">
25 <input type="button" value="一時停止" onClick="f01( )">
26 <input type="button" value="再生" onClick="f02( )">
27 </form>
```

³⁸audio, video タグの id 名をそのままメディアのオブジェクトとして扱うこともできます。

```
28 </body>
29 </html>
```

これを Web ブラウザで表示すると図 62 のようになります。



図 62: SoundTest01.html の実行例

ボタンのクリックによって再生／一時停止する

7.2.2.1 audio/video タグへのコントロールの追加

audio/video タグに controls を追加することで、操作のインターフェースをコンテンツ内に表示することができます。

例. `<audio src="音声のファイル名" controls>～</audio>`

controls を追加してコンテンツを表示した例を図 63 に示します。



図 63: controls を追加して実行した例

audio/video タグには controls 以外にも、次のようなものを指定することができます。

autoplay : コンテンツの読み込みと同時に再生を開始する
loop : 再生を繰り返す

7.2.3 audio タグを使用せずに音声を再生する方法

HTML5 の audio タグでコンテンツ内に音声データを配置せずに、JavaScript のプログラム内で直接音声ファイルを読み込んで再生する方法もあります。

7.2.3.1 音声ファイルの読み込み

音声ファイルを指定して Audio オブジェクトを生成することでそのファイルを開きます。

例. `var snd = new Audio("音声ファイル名");`

これで音声ファイルが開きます。このようにして生成した Audio オブジェクトに対して前述の play, pause などの操作を行います。

audio タグを使わずに、Audio オブジェクトを生成することでサウンドを再生するサンプルを SoundTest03.html に示します。

サンプル：SoundTest03.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>Audioオブジェクト</title>
6 <script>
7 var snd = new Audio("one02.mp3"); // サウンドのファイルを開く
8
```



```

9  function f01( ) {
10     snd.load(); // 実際にサウンドデータを読み込む
11     snd.play(); // 再生実行
12 }
13 </script>
14 </head>
15 <body>
16 <form name="fm">
17 <input type="button" value="チャイム" onClick="f01( )">
18 </form>
19 </body>
20 </html>

```

これを Web ブラウザで表示すると図 64 のようになります。

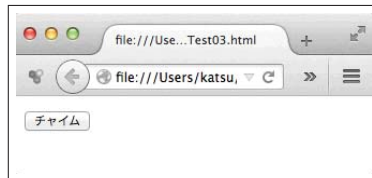


図 64: ボタンをクリックすると音が鳴る

この例では `play` による再生の直前に `load` メソッドで明に音声データを読み込んでいます。こうすることで、再生が完了するまでに再度ボタンをクリックしても音声先頭から再生されます。

7.2.4 再生の時刻に関すること

メディアデータには時刻に関する次のようなプロパティがあります。

表 18: メディアデータ `m` の時刻に関するプロパティ

プロパティ	意味
<code>m.duration</code>	メディア <code>m</code> の長さ (時間) 単位: 秒
<code>m.currentTime</code>	メディア <code>m</code> 再生時の現在時刻 単位: 秒

このうち `duration` は読み取り専用で、値の変更はできません。また `currentTime` は読み書きできます。この値を設定して `play()` を実行すると、指定した秒数の位置から再生を開始することができます。

7.2.4.1 メディアデータ再生時に発生するイベント

メディアデータが再生されている状態では再生位置 (時刻) が変化します。この「再生位置が変化した」ことで発生するイベントに `timeupdate` があります。このイベントに対するイベントリスナは `addEventListener` で登録する必要があります。

メディアデータ `a` に `timeupdate` イベントのリスナを登録する例

```
a.addEventListener("timeupdate", "イベントハンドラ名", false);
```

7.2.5 再生の音量に関すること

■ volume 属性

メディアデータ `a` の再生音量 (ボリューム) は `volume` 属性に設定されています。また、この属性に値を設定することで音量を設定することができます。値の範囲は 0.0~1.0 です。

ボリュームが変化したことを知らせるイベントとして `volumechange` があります。

■ muted 属性

メディアデータの再生音を止めるには、そのメディアデータの `muted` 属性に `true` を設定します。この設定により、

システムの音量設定の状態にかかわらず、当該メディアデータの再生音が抑止されます。muted 属性に false が設定されていると再生音が出ます。

7.2.6 再生の速度に関すること

■ playbackRate 属性

メディアデータの再生速度を変更するには、そのメディアデータの playbackRate 属性に速度の比率を設定します。通常の速度は 1.0 です。また、速度比率の変更があったことを知らせるイベントに ratechange があります。

7.3 タイミングイベント

setTimeout を使用すると、指定した時間が経過した後で指定した処理を実行するように設定することができます。具体的には

```
var timer = setTimeout(実行したい関数, 経過時間 (単位: ミリ秒));
```

と記述します。setTimeout は setTimeoutID と呼ばれる値を返します。この戻り値を使って clearTimeout を実行すると、仕掛けたタイミングイベントをキャンセルすることができます。上の例では戻り値が timer という変数に格納されているので、これをキャンセルするには次のようにします。

```
clearTimeout(timer);
```

タイミングイベントを使用して時計を実現したサンプル Clock.html を示します。

サンプル: Clock.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>時計</title>
6  <script>
7  // 日付・時刻
8  var datetime;
9
10 // タイマー
11 var timer;
12
13 // 曜日の設定
14 function setWeekDay(day) {
15     switch ( day ) {
16         case 0:
17             document.fm.wd.value = "日";
18             break;
19         case 1:
20             document.fm.wd.value = "月";
21             break;
22         case 2:
23             document.fm.wd.value = "火";
24             break;
25         case 3:
26             document.fm.wd.value = "水";
27             break;
28         case 4:
29             document.fm.wd.value = "木";
30             break;
31         case 5:
32             document.fm.wd.value = "金";
33             break;
34         case 6:
35             document.fm.wd.value = "土";
36             break;
37         default:
38             document.fm.wd.value = "?";
```

```

39     }
40 }
41
42 // 時計の開始
43 function f01( ) {
44     datetime = new Date();
45     document.fm.yy.value = datetime.getFullYear();
46     document.fm.mm.value = datetime.getMonth();
47     document.fm.dd.value = datetime.getDate();
48     setWeekDay(datetime.getDay());
49     document.fm.tt.value = datetime.getHours();
50     document.fm.mn.value = datetime.getMinutes();
51     document.fm.s1.value = datetime.getSeconds();
52     document.fm.s2.value = parseInt(datetime.getMilliseconds() / 100);
53     timer = setTimeout("f01( )",100);
54 }
55
56 // 時計の停止
57 function f02( ) {
58     clearTimeout(timer);
59 }
60 </script>
61 <body>
62 <form name="fm">
63 <input type="button" value="時計を開始" onClick="f01( )">
64 <input type="button" value="時計を停止" onClick="f02( )"><br>
65 <input type="text" size="4" name="yy">年
66 <input type="text" size="2" name="mm">月
67 <input type="text" size="2" name="dd">日
68 <input type="text" size="2" name="wd">曜日<br>
69 <input type="text" size="2" name="tt">時
70 <input type="text" size="2" name="mn">分
71 <input type="text" size="2" name="s1">.
72 <input type="text" size="2" name="s2">秒
73 </form>
74 </body>
75 </html>

```

これを Web ブラウザで表示すると図 65 のようになります。



図 65: 時計

7.3.1 タイミングイベントの繰り返し

時計を実現するには、一定時間毎に表示を更新する必要があります。このサンプルでは、現在の日付と時刻を表示する関数 f01 の中で、表示が終わる毎に setTimeout を実行し、100 ミリ秒毎に関数 f01 を再度実行して表示を更新しています。また、setTimeout の実行の度に戻り値が変数 time に格納されており、これに対して clearTimeout を実行することで時計の機能を停止することができます。

7.3.1.1 setInterval

指定した関数を指定した時間間隔で繰り返し実行することを setInterval で設定することができます。(次の例)

関数の繰り返し実行の例： var timer = setInterval(実行したい関数, 経過時間 (単位：ミリ秒))

設定された繰り返し設定をキャンセルするには

```
clearTimeout(timer);
```

とします。

考察) 先の Clock.html を setInterval を用いて作り直してください。

7.4 データの保存

JavaScript のプログラムからは OS のファイルを扱うことは出来ませんが、データ保存のための独自の方法があります。1つは localStorage, もう1つは sessionStorage です。localStorage は Web ブラウザが管理する独自のデータ保存機能で、Web ブラウザを終了した後もデータが保存されます。また sessionStorage は Web ブラウザのウィンドウが開いている間、そのウィンドウに対して有効となるデータ保存機能で、ウィンドウを閉じてしまうと sessionStorage の値は消滅します。

これらの保存機能では、データは**キー**と**値**の組として保存されます。すなわち、データはキーと値の組を指定して保存し、参照する場合はキーのみを指定します。具体的な使用方法を次に示します。

7.4.1 値の保存

localStorage に値を保存するには setItem メソッドを使います。

例. "佐藤の身長"のキーと"171cm"の値を組にして保存する方法

```
localStorage.setItem( "佐藤の身長" , "171cm" );
```

7.4.2 値の参照

localStorage に保存されている値を参照するには getItem メソッドを使います。

例. "佐藤の身長"のキーを指定して値を取り出す方法

```
var v = localStorage.getItem( "佐藤の身長" );
```

これで、変数 v に値が格納されます。

7.4.3 値の消去

localStorage に保存されているデータのうち、特定のキーのデータを消去するには removeItem メソッドを使います。

例. "佐藤の身長"のキーのデータを消去する方法

```
localStorage.removeItem( "佐藤の身長" );
```

これで、"佐藤の身長"のデータが消去されます。

localStorage に保存されているデータの全てを消去するには clear メソッドを使います。

例. 全てのデータを消去する方法

```
localStorage.clear();
```

これで全てのデータが消去されます。(要注意)

sessionStorage も同様のメソッドを用いて使用することができます。

7.4.4 その他 (データの管理に関する機能など)

7.4.4.1 登録されているデータの個数を調べる方法

localStorage の length 属性を参照することで、登録されているデータの組の数がわかります。

7.4.4.2 登録されているデータを順番に取り出す方法

localStorage の key 属性を参照することで、「n 番目」を指定したキーの参照ができます。すなわち n 番目のキーを参照するには次のようにします。

```
var k = localStorage.key(n);
```

これで変数 k に n 番目のキーが得られます。

7.4.5 localStorage の応用プログラム

localStorage を用いて、簡単なデータベースを実現するプログラムを test5.html に示します。先の「5.6.2 オブジェクト（連想配列）」で挙げたサンプルプログラムと似ていますが、今回のプログラムではブラウザを終了しても値が保持されます。

サンプル：test5.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>ストレージのテスト</title>
6  <script>
7  // データの検索
8  function srch() {
9      var k = k1.value + "." + k2.value;
10     var v = localStorage.getItem(k);
11     if ( v == null ) {
12         v1.value = "（登録がありません）";
13     } else {
14         v1.value = v;
15     }
16     dcnt();
17 }
18
19 // データの登録
20 function rgst() {
21     var k = k1.value + "." + k2.value;
22     var v = v1.value;
23     localStorage.setItem(k,v);
24     dcnt();
25 }
26
27 // データを消去（1件だけ消去）
28 function dlt() {
29     var k = k1.value + "." + k2.value;
30     localStorage.removeItem(k);
31     dcnt();
32 }
33 // データを全て消去
34 function zap() {
35     localStorage.clear();
36     dcnt();
37 }
38
39 // データ個数の表示
40 function dcnt() {
41     n1.value = (localStorage.length).toString(10);
42 }
43
44 // データの一覧
45 function dlst() {
46     var k, s = ""
47     var c;
48     for ( c = 0; c < localStorage.length; c++ ) {
49         k = localStorage.key(c);
50         s = s + c + ") " + k + " : " + localStorage.getItem(k) + "\n";
51     }
52     a1.value = s;
```

```

53 }
54 </script>
55 <body>
56 <form name="fm">
57   【簡易データベース】<br>
58   検索してください<br>
59   キー：<input type="text" id="k1"><br>
60   属性：<input type="text" id="k2"><br>
61   値：<input type="text" id="v1"><br>
62   <input type="button" value="検索" onClick="srch()">
63   <input type="button" value="新規登録" onClick="rgst()">
64   <input type="button" value="消去" onClick="dlt()">
65   <input type="button" value="全て消去" onClick="zap()"><br>
66   登録データ数：<input type="text" id="n1" size="5"><br>
67   <input type="button" value="データ一覧表示" onClick="dlst()"><br>
68   <textarea id="a1" cols="30" rows="8"></textarea>
69 </form>
70 </body>
71 </html>

```

このプログラムは、キーと属性を与えて値を検索するもので、データの検索、新規登録、消去などができるものです。(図 66～70)



図 66: Web ブラウザによる test5.html の表示 (1)



「キー」と「属性」を設定して検索キーを押すと

「値」の欄に値が表示される

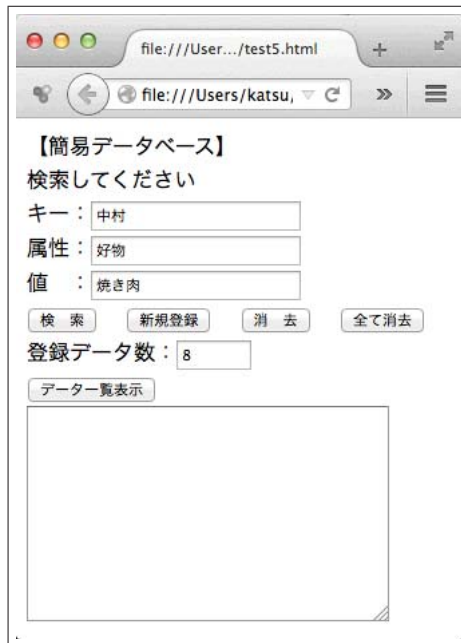
図 67: Web ブラウザによる test5.html の表示 (2)



「キー」と「属性」に値の設定がない場合は

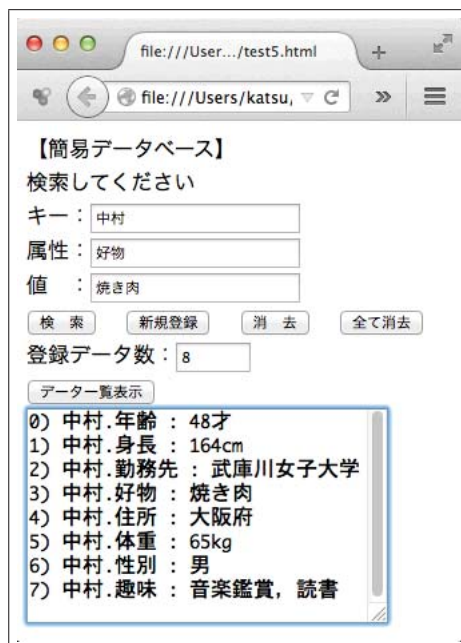
「値」の欄に“(登録がありません)”が表示される

図 68: Web ブラウザによる test5.html の表示 (3)



新規にデータを登録する場合は「キー」と「属性」に値を設定して「新規登録」ボタンをクリック

図 69: Web ブラウザによる test5.html の表示 (4)



「データ一覧表示」をクリックするとテキストエリアに一覧が表示される

図 70: Web ブラウザによる test5.html の表示 (5)

7.5 表示環境の取得と設定

7.5.1 表示領域のサイズの取得

HTML5 コンテンツとしてアプリケーションを作成する場合、スクリーンやウィンドウのサイズは重要な設定項目です。これらに関する値を保持しているオブジェクトや属性、その意味を表 19 に示します。

表 19: スクリーンやウィンドウのサイズに関するオブジェクトと属性

値	意 味
screen.width	ディスプレイの横幅
screen.height	ディスプレイの高さ
screen.availWidth	ディスプレイの有効表示幅
screen.availHeight	ディスプレイの有効表示高さ
window.outerWidth	ウィンドウの横幅（外周）
window.outerHeight	ウィンドウの高さ（外周）
window.innerWidth	ウィンドウの横幅（表示領域）※ 1
window.innerHeight	ウィンドウの高さ（表示領域）※ 1
document.documentElement.clientWidth	ウィンドウの横幅（表示領域）※ 2
document.documentElement.clientHeight	ウィンドウの高さ（表示領域）※ 2

（※ 1 と※ 2 は同様の値となります。 Web ブラウザなどの
実行環境ごとにどちらを使うかを判断したほうが良いです）

これらを利用して、各種の表示領域サイズを調査するサンプルプログラムを Env.html に示します。

サンプル：Env.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>環境の 情報</title>
6  <script>
7  function f01( ) {
8      document.fm.f1.value = screen.width;
9      document.fm.f2.value = screen.height;
10     document.fm.f3.value = screen.availWidth;
11     document.fm.f4.value = screen.availHeight;
12     document.fm.f5.value = window.outerWidth;
13     document.fm.f6.value = window.outerHeight;
14     document.fm.f7.value = window.innerWidth;
15     document.fm.f8.value = window.innerHeight;
16     document.fm.f9.value = document.documentElement.clientWidth;
17     document.fm.fa.value = document.documentElement.clientHeight;
18 }
19 </script>
20 <body onLoad="f01( )" onResize="f01( )">
21 <form name="fm">
22 screen(total):<input type="text" size="4" name="f1">^^e2^^9c^^95
23 <input type="text" size="4" name="f2"><br>
24 screen(available):<input type="text" size="4" name="f3">^^e2^^9c^^95
25 <input type="text" size="4" name="f4"><br>
26 outerWidth:<input type="text" size="4" name="f5">^^e2^^9c^^95
27 outerHeight:<input type="text" size="4" name="f6"><br>
28 innerWidth:<input type="text" size="4" name="f7">^^e2^^9c^^95
29 innerHeight:<input type="text" size="4" name="f8"><br>
30 (document.documentElement)<br>
31 clientWidth:<input type="text" size="4" name="f9">^^e2^^9c^^95
32 clientHeight:<input type="text" size="4" name="fa"><br>
33 </form>
34 </body>
35 </html>
```

これを Web ブラウザで表示すると図 71 のようになります。

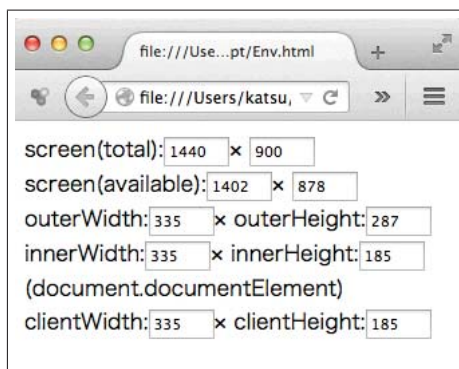


図 71: 表示領域に各種サイズの調査

Env.html の表示においてウィンドウサイズを変更すると、それに合わせて表示が更新されます。

7.5.2 サーバとブラウザに関する情報の取得

当該 HTML コンテンツを配信しているサーバに関する情報や、コンテンツを表示しているクライアントアプリケーションに関する情報を取得することができます。これらに関する値を保持しているオブジェクトや属性、その意味を表 20 に示します。

表 20: サーバとブラウザに関するオブジェクトと属性

値	意 味
location.hostname	サーバのホスト名
location.pathname	サーバ上でのコンテンツのパス
navigator.appName	クライアントアプリケーションの名前
navigator.appVersion	クライアントアプリケーションのバージョン
navigator.language	クライアントアプリケーションが使用する言語
navigator.userAgent	クライアント環境の各種情報

これらを利用して、Web サーバや Web ブラウザに関する情報を調査するサンプルプログラムを EnvTest1.html に示します。

サンプル：EnvTest1.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>システム環境</title>
6 <script>
7 var hname, pname;
8 var aname, aver, lang, uagent;
9
10 function f01( ) {
11     hname = location.hostname;
12     pname = location.pathname;
13     aname = navigator.appName;
14     aver = navigator.appVersion;
15     lang = navigator.language;
16     uagent = navigator.userAgent;
17     document.fm.tf1.value = hname;
18     document.fm.tf2.value = pname;
19     document.fm.tf3.value = aname;
20     document.fm.tf4.value = aver;
21     document.fm.tf5.value = lang;
22     document.fm.tf6.value = uagent;
```

```

23 }
24 </script>
25 </head>
26 <body onLoad="f01( )">
27 <form name="fm">
28 ホスト名:<input type="text" name="tf1" size="60"><br>
29 パス:<input type="text" name="tf2" size="60"><br>
30 ブラウザ名:<input type="text" name="tf3" size="40"><br>
31 ブラウザver.:<br>
32 <textarea name="tf4" rows="4" cols="60"></textarea><br>
33 端末の言語:<input type="text" name="tf5" size="40"><br>
34 ユーザagent:<br>
35 <textarea name="tf6" rows="4" cols="60"></textarea><br>
36 </form>
37 </body>
38 </html>

```

これを Web ブラウザで表示すると図 72, 73 のようになります。

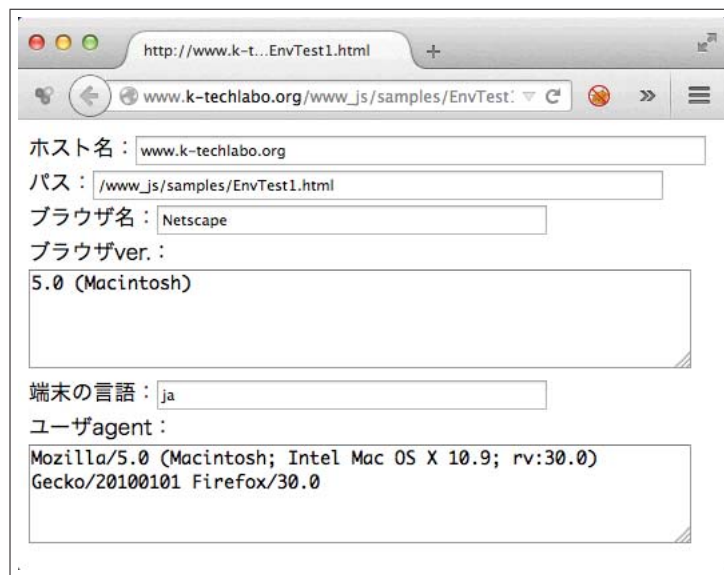


図 72: Mozilla Firefox で実行した例

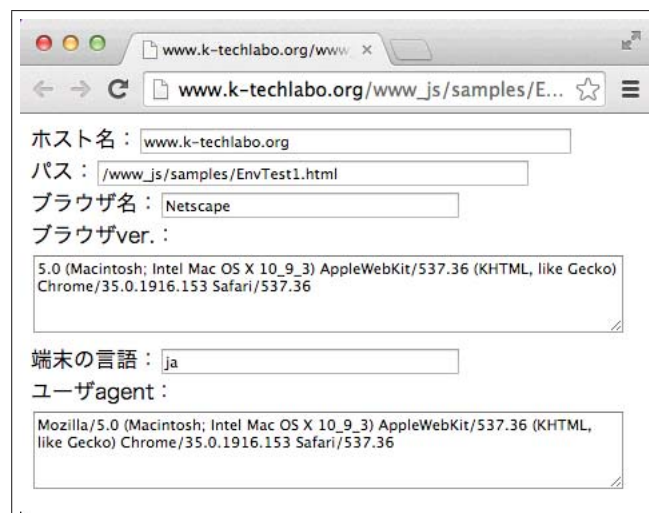


図 73: Google Chrome で実行した例

7.6 JavaScript によるスタイルの設定

CSS による HTML コンテンツの体裁設定と同様のことが JavaScript から可能です。JavaScript のプログラムでは HTML タグの id 名を指定してコンテンツの要素を取得し、それに対してスタイルを設定するという流れになります。

7.6.1 id 属性による HTML 要素の参照

getElementById メソッドを使用すると指定した id 属性の HTML 要素を取り出すこと³⁹が出来ます。次の例は "id 名"を持つ HTML 要素が e1 に得られます。

例. `var e1 = document.getElementById("id 名");`

以後はこれに対してスタイルの設定をします。

7.6.2 スタイルの設定

getElementById メソッドで得られた要素にスタイルを設定するには style 属性を使用⁴⁰します。

7.6.2.1 位置属性の設定

div 要素のスタイルの位置属性 (top, left) を変更することで要素を移動させるサンプルプログラムを Style.html に示します。

サンプル：Style.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>位置属性の設定</title>
6  <script>
7  var e1;
8  var x, y, dx, dy;
9  var timer;
10
11 // 初期化と開始
12 function f01( ) {
13     x = 0;      y = 0;
14     dx = 2;     dy = 2;
15     e1 = document.getElementById("d1");
16     e1.style.position = "relative";
17     e1.style.width = 30;
18     e1.style.height = 20;
19     e1.style.top = 0;
20     e1.style.left = 0;
21     f02( );
22 }
23
24 // 繰り返し処理
25 function f02( ) {
26     document.fm.tf.value = x + "," + y;
27     x += dx;      y += dy;
28     if ( x <= 0 || 297 <= x ) {
29         dx *= -1;
30     }
31     if ( y <= 0 || 203 <= y ) {
32         dy *= -1;
33     }
34     e1.style.top = y + "px";
35     e1.style.left = x + "px";
36     e1.style.visibility="visible";
37     timer = setTimeout("f02( )",5);
```

³⁹id 属性の値をそのまま HTML 要素としてアクセスすることも可能です。

⁴⁰もっと簡単に、「id 名.style」としてアクセスすることもできます。

```

38 }
39
40 // 停止
41 function f03( ) {
42     clearTimeout(timer);
43 }
44 </script>
45 <body onLoad="f01( )">
46 <form name="fm">
47 <input type="text" name="tf">
48 <input type="button" value="停止" onClick="f03( )">
49 </form>
50 <div id="d1">文字</div>
51 </body>
52 </html>

```

これを Web ブラウザで表示すると図 74 のようになります。



図 74: スタイル（位置）を変更することで要素の移動を実現する例

7.6.2.2 可視属性の設定

div 要素のスタイルの可視属性（visibility）を変更することで要素の表示を切り替えるサンプルプログラムを Style2.html に示します。

サンプル：Style2.html

```

1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>可視属性のテスト</title>
5 <style>
6 button {
7     width: 30pt;
8     margin: 0pt;
9 }
10 #b1 {
11     position: absolute;
12     top: 10pt;
13     left: 10pt;
14     width: 108pt;
15     padding: 2pt;
16     border: solid 1px #aaaaaa;
17 }
18 .c1 {
19     position: absolute;
20     top: 36pt;
21     left: 10pt;
22     width: 112pt;
23     border: solid 1px #aaaaaa;
24     font-family: sans-serif;

```

```

25     font-size: 72pt;
26     text-align: center;
27 }
28 </style>
29 <script>
30 function f1(n) {
31     var el, c;
32     for ( c = 1; c <= 3; c++ ) {
33         el = document.getElementById("d"+c);
34         el.style.visibility = "hidden";
35     }
36     el = document.getElementById("d"+n);
37     el.style.visibility = "visible";
38 }
39 </script>
40 </head>
41 <body onLoad="f1('1')">
42 <!-- ボタン -->
43 <div id="b1">
44 <button onClick="f1('1')">1</button>
45 <button onClick="f1('2')">2</button>
46 <button onClick="f1('3')">3</button>
47 </div>
48 <!-- 表示部分 -->
49 <div>
50 <div class="c1" id="d1">1</div>
51 <div class="c1" id="d2">2</div>
52 <div class="c1" id="d3">3</div>
53 </div>
54 </body>
55 </html>

```

これを Web ブラウザで表示すると図 75 のようになります。

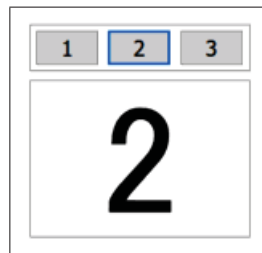


図 75: スタイル（可視属性）変更の例
visibility の設定により，表示／非表示を切り替える

7.7 要素の位置とサイズの取得

DOM の要素に対して `getBoundingClientRect` メソッドを実行すると，その要素の位置とサイズに関する情報が得られます。例えば，DOM 要素 `e` に対して

```
var r = e.getBoundingClientRect();
```

とすると，位置とサイズの情報を持ったオブジェクトが `r` に得られます。このオブジェクトのプロパティの内で重要なものは以下のとおりです。

プロパティ	値	プロパティ	値
<code>top</code>	要素の上の位置	<code>bottom</code>	要素の下の位置
<code>left</code>	要素の左の位置	<code>right</code>	要素の右の位置
<code>width</code>	要素の横幅	<code>height</code>	要素の高さ

`getBoundingClientRect` を応用したサンプルを `Style3.html` に示します。

サンプル：Style3.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>要素の位置の取得</title>
6  <style>
7  #d1 {
8      /* 位置, サイズ, 枠 */
9      position:      absolute;
10     top:           10px;
11     left:          20px;
12     width:         160px;
13     height:        90px;
14     border:        1px solid red;
15     /* フォントの設定 */
16     font-size:     24px;
17     text-align:    center; /* 左右中央揃え */
18     line-height:   90px
19 }
20 #t1 {
21     position:      absolute;
22     top:           120px;
23     left:          20px;
24     width:         160px;
25     height:        170px;
26 }
27 </style>
28 <script>
29 function f1( ) {
30     var rct = d1.getBoundingClientRect();
31     var msg =
32         "top: "      + rct.top +      "\n" +      "left: "      + rct.left +      "\n" +
33         "bottom: "   + rct.bottom + "\n" +      "right: "     + rct.right +      "\n" +
34         "width: "    + rct.width +   "\n" +      "height: "    + rct.height +   "\n" +
35         "x: "        + rct.x +       "\n" +      "y: "         + rct.y
36     t1.value = msg;
37 }
38 </script>
39 </head>
40 <body onLoad="f1( )">
41 <div id="d1">DOM要素</div>
42 <textarea id="t1"></textarea>
43 </body>
44 </html>
```

これを Web ブラウザで表示すると図 76 のようになります。

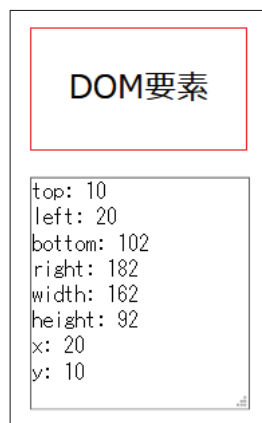


図 76: DOM 要素の位置とサイズの取得

id 名が "d1" の要素（テキスト「DOM 要素」を持つ枠）の位置とサイズを取得し、それらをテキストエリア "t1" に表示しています。

7.8 ファイルの読み込み

主要なブラウザでは、ローカルファイルの読み込みが可能です。ここでは、テキストファイルを読み込みその内容をテキストエリアに表示するプログラム（図 77）を例に挙げて JavaScript でローカルファイルを読み込むための基本的な方法について説明します。

図 77 のウィンドウ上部に「参照」ボタンがあります。（図 78）これをクリックすると、ファイル選択のためのダイアログが開き（図 79）ます。ファイルを選んで「開く (O)」ボタンをクリックすると、図 80 のようにテキストエリア内にファイルの内容を表示します。



図 77: 例. ファイルの内容を表示する Web アプリ

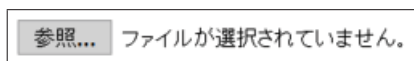


図 78: ローカルファイルを選択するための「参照」ボタン

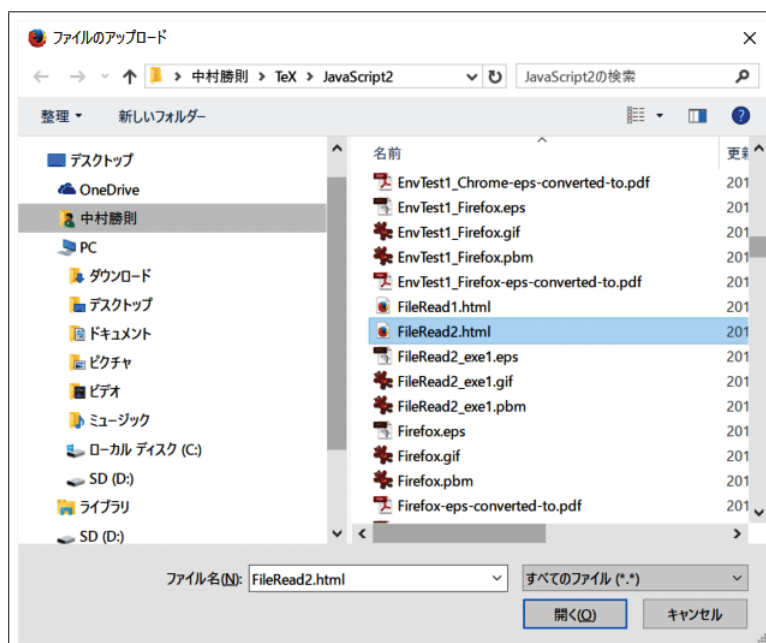


図 79: ファイル選択ダイアログ

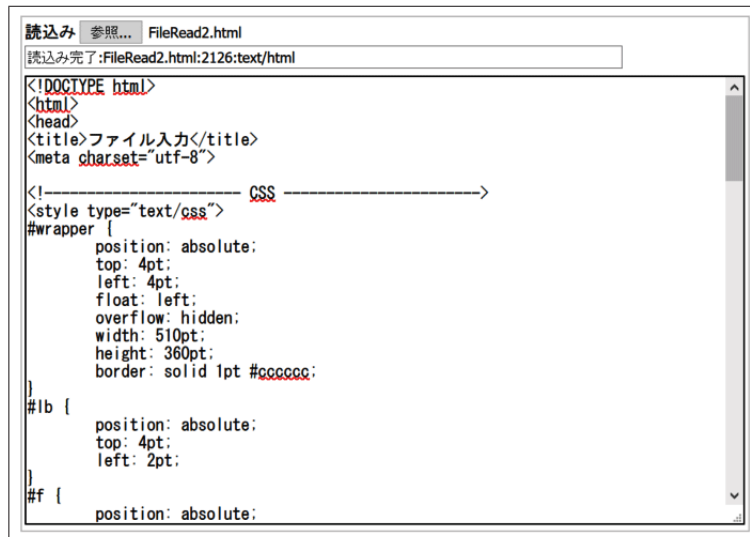


図 80: ファイル内容の表示

7.8.1 ファイル選択ダイアログ

ファイルを選択するための「参照」ボタンは、HTML の input 要素に、type="file" を与えて実現します。

例. `<input type="file">`

このようにしてタグを配置すると図 78 のようなボタンがコンテンツ内に表示されます。ファイルダイアログでファイルを選択して「開く (O)」ボタンをクリックすると、その input 要素に change イベントが発生するので、これを受けてファイル読み込みと表示の処理を起動します。

発生した change イベントの target 属性に選択されたファイルの情報が保持されているので、それを用いて実際にファイルの内容を取得します。例えば、発生した change イベントのオブジェクト e がある場合。

```
var fl = e.target.files;
```

を実行すると、選択されたファイルの情報が file オブジェクトの配列 fl[] として得られます。ファイル選択ダイアログでは複数のファイルを同時に選択するオプションがあり、配列の各々の要素 fl[0], fl[1], ... に選択されたファイルの情報を全て取得することができます。(今回の説明では 1 つだけファイルを選択する処理の例について説明します)

得られた file オブジェクトを用いて実際にファイルの内容を読み込むために、次に説明する FileReader オブジェクトを使います。

7.8.2 FileReader

FileReader オブジェクトはファイルの読み込みに関する処理に必要で、

```
var rd = new FileReader();
```

として生成することができます。この例ではインスタンス rd として FileReader オブジェクトを生成しています。生成した FileReader に対して readAsText メソッドを実行すると、テキストファイルの内容を読み込むことができます。

例. file オブジェクト fl[0] に格納されているファイル情報を元に内容を読み込む

```
rd.readAsText(fl[0]);
```

このメソッドの第 1 引数には file オブジェクトを与えます。

ファイルの内容の読み込みが終了すると、FileReader オブジェクトに onload イベントが発生するので、このイベントを受けて、ファイルの内容を取り出す処理をします。

ファイルの内容は FileReader オブジェクトの result 属性に保持されています。

7.8.3 サンプルプログラム

先に説明したサンプルプログラムを FileRead2.html に示します.

サンプル：FileRead2.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>ファイル入力</title>
6  <style>
7  #wrapper {
8      position: absolute;
9      top: 4pt;
10     left: 4pt;
11     float: left;
12     overflow: hidden;
13     width: 510pt;
14     height: 360pt;
15     border: solid 1pt #cccccc;
16 }
17 #lb {
18     position: absolute;
19     top: 4pt;
20     left: 2pt;
21 }
22 #f {
23     position: absolute;
24     top: 2pt;
25     left: 40pt;
26 }
27 #msg {
28     position: absolute;
29     top: 20pt;
30     left: 2pt;
31 }
32 #txt {
33     position: absolute;
34     top: 40pt;
35     left: 2pt;
36     border: solid 1pt black;
37 }
38 </style>
39 <script>
40 var f, msg, txt;
41 var fl, rd;
42
43 // 読み込み開始
44 function fr(e) {
45     fl = e.target.files;
46     rd.readAsText(fl[0]);
47     msg.value = fl[0].name;
48 }
49
50 // 読み込み終了後の処理
51 function getcnt(e) {
52     if ( rd.error == null ) {
53         msg.value = "読み込み完了:" + fl[0].name + ":" +
54                     fl[0].size + ":" + fl[0].type;
55         txt.value = rd.result;
56     } else {
57         msg.value = fl[0]+" : 読み込みエラー : "+rd.error;
58     }
59 }
60
61 // 初期設定
62 function ini() {
63     // テキストフィールド
64     msg = document.getElementById("msg");
```

```

65 // ファイル選択用オブジェクト
66 f = document.getElementById("f");
67 // ファイルが選択時の処理の登録
68 f.addEventListener("change", fr, false);
69 f.value = "";
70
71 // テキストエリア
72 txt = document.getElementById("txt");
73 txt.value = "";
74
75 // ファイルリーダー
76 if ( window.FileReader ) {
77     msg.value = "（ファイル読み込みが可能です）";
78     rd = new FileReader();
79     // ファイル読み込み終了時の処理の登録
80     rd.onload = getcnt;
81 } else {
82     msg.value = "（ファイル読み込みに対応していません）";
83 }
84
85 }
86 </script>
87 </head>
88 <body onLoad="ini()">
89 <div id="wrapper">
90     <div id="lb">読み込み</div>
91     <input type="file" id="f">
92     <input type="text" size="90" id="msg">
93     <textarea cols="80" rows="24" id="txt"></textarea>
94 </div>
95 </body>
96 </html>

```

解説：

このサンプルでは、ファイルダイアログを表示するためのボタン（id は'f'）を 91 行目に、ファイルの内容を表示するためのテキストエリア（id は'txt'）を 93 行目に記述している。92 行目に記述しているテキストフィールド（id は'msg'）は各処理実行後のメッセージ（エラーなど）を表示するためのものです。

6～38 行目： GUIを構成するためのスタイルの記述。

62～85 行目： 初期化处理，このコンテンツが起動した直後（body 要素が読み込まれた直後）body 要素に起こる onload イベントを受けて起動する処理で，HTML の要素を JavaScript の変数に対応させたり，イベントハンドラの登録などを行います。

69 行目では，ファイルオブジェクト f にファイル選択のイベント change が起こった際にイベントハンドラ fr を起動する設定にしています。

81 行目では，FileReader オブジェクト rd に onload イベントが起こった際に，イベントハンドラ getcnt が起動する設定にしています。

77～84 行目： 使用するブラウザがファイルの読み込みに対応しているかどうかを判定。

window.fileReader が真（true）の場合はファイルの読み込みに対応していることを示します。その場合に FileReader オブジェクトを生成して，onload イベントの処理 getcnt を登録します。

■ file オブジェクトの属性

47,53,54 行目にあるように，file オブジェクトには次のような各種の属性があります。

- name** 選択されたファイルの名前
- size** 選択されたファイルのサイズ（バイト）
- type** 選択されたファイルのタイプに関する情報

■ ファイル読み込み時に起こるエラーの処理

52 行目にあるように、ファイルの内容を読み込んでいる際にエラーが発生したかどうかを `FileReader` オブジェクトの `error` 属性で調べることができます。この属性の値が `null` の場合は、読み込みが正常に終了したことを意味しています。

「A.2 ダウンロード機能の実装例」(p.174) では、簡単なテキストエディタの実装例を紹介しています。ここで紹介したファイルの読み込み処理を更に簡素化した形で応用しています。

7.9 ドラッグアンドドロップ

多くの Web ブラウザはドラッグアンドドロップに対応しており、コンテンツ内の要素をマウスでドラッグ (drag) することができます。マウスで HTML 要素を他の要素の上までドラッグし、そこでマウスボタンを放すことでドロップが起こります。ここでは、ドラッグアンドドロップが可能な HTML コンテンツの作成方法について解説します。

多くの HTML 要素 (タグ) はドラッグアンドドロップに対応しておらず⁴¹、マウスによる移動ができません。そのため、ドラッグ操作の対象とする HTML 要素に対しては、ドラッグ操作を可能とするための設定が必要となります。このことを次のサンプル `DnD01_0.html` を例に上げて説明します。

サンプル：DnD01_0.html

```
1  <!DOCTYPE html>
2  <html lang="ja">
3  <head>
4  <meta charset="utf-8">
5  <title>DnDテスト1</title>
6
7  <style>
8  .dv {
9      position: absolute;
10     width: 200px; height: 80px;
11     border: solid 3px;
12     font-size: 24px;
13     text-align: center;
14     line-height: 80px;
15 }
16 #dv1 {
17     top: 5px; left: 5px;
18     border-color: blue;
19 }
20 #dv2 {
21     top: 5px; left: 240px;
22     border-color: red;
23 }
24 </style>
25
26 <script>
27 </script>
28 </head>
29
30 <body>
31 <div id="dv1" class="dv">これをドラッグ</div>
32 <div id="dv2" class="dv">ここへドラッグ</div>
33 </body>
34 </html>
```

これは2つの<div>要素で実現した四角形 (青と赤で色分け) を表示するものです。<div>要素は最初はドラッグ処理に対応しておらず、青の枠を赤の枠の上にドラッグしようとする、<div>内のテキストをドラッグすることになります。(図 81 参照)

⁴¹一部の要素 (や <a> など) はデフォルトでドラッグに対応しています。

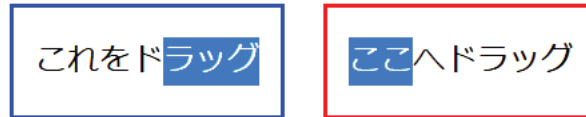


図 81: 敢えてドラッグすると…
枠は動かず、文字をドラッグ（選択）することになる

HTML 要素をドラッグ可能にするためには、その要素の `draggable` 属性に `"true"` を設定します。つまり、青い枠の `<div>` 要素を

```
<div id="dv1" class="dv" draggable="true">これをドラッグ</div>
```

に修正するとドラッグが可能になります。（図 82 参照）

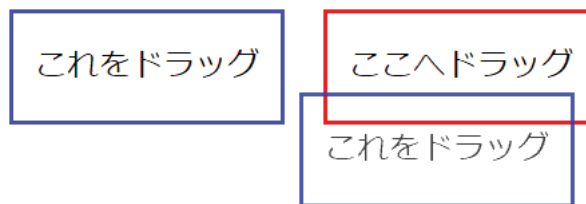


図 82: `draggable="true"` に設定すると…
枠がドラッグに反応して移動する
（Google Chrome ブラウザの場合）

7.9.1 ドラッグアンドドロップに伴う処理

ドラッグアンドドロップにおいては ドラッグされる対象 と ドロップされる対象 があり、前者にドラッグが起これると `dragstart` イベントが、後者にドロップが起これると `drop` イベントが発生します。また、前者をドラッグして後者の上に重ねている間中 `dragover` イベントが連続して発生します。

7.9.1.1 `dragstart` イベントを受けて行う処理

`dragstart` はマウスによってドラッグされた要素に対して発生するものです。この処理を受けて行うべき処理は、当該イベントオブジェクトの `dataTransfer` 属性の設定です。

マウスによるドラッグの操作を行うと、対象の要素が視覚的に移動します。そしてそれをドロップした際に、ドロップ先の要素に対して 事前に設定しておいた値 を渡します。すなわち、その渡すべき値を `dragstart` のイベントオブジェクトの `dataTransfer` 属性に設定しておきます。

先のサンプル `DnD01_0.html` において、ドラッグする青い枠 (`id="dv1"`) がドラッグされ始める際に `dataTransfer` の値を設定する例を示します。

```
dv1 = document.getElementById("dv1");
dv1.addEventListener("dragstart", function(e) {
    e.dataTransfer.setData("text/plain", (文字データ));
}, false);
```

この例では、`id="dv1"` の要素がドラッグされ始めたときに、イベントオブジェクトに「文字データ」を設定しています。この処理は、当該イベントオブジェクトの `dataTransfer` 属性に対して `setData` メソッドで実行します。`setData` メソッドの第 1 引数にはデータの型（**MIME タイプ**）を、第 2 引数にはデータを与えます。上の例はテキストデータ（文字データ）を与えるためのもので、データの型として `"text/plain"` ⁴² を指定しています。

⁴² 単純なテキストデータを意味する MIME タイプ。

7.9.1.2 dragover イベントを受けて行う処理

dragover は、ドロップを受け付ける側の要素の上に何かがドラッグ処理で重ねられた（さしかかった）ときに発生するものです。このときの重要な処理として、ドラッグによってもたらされるデータが元のデータの複製（コピー）であるか、あるいは元のものを持移動してきたものかを設定することがあります。このために、イベントオブジェクトの `dataTransfer.dropEffect` 属性に "copy" もしくは "move" を設定します。

ドラッグアンドドロップのためのイベント処理は、他のイベント処理と混乱が起こることが多く、dragover のイベントを扱う場合は `preventDefault` メソッドで他のイベント処理を抑止しておく安全です。

7.9.1.3 drop イベントを受けて行う処理

drop は、ドロップを受け付ける側の要素の上に何かがドロップされたときに発生するものです。この場合も `preventDefault` メソッドで他のイベント処理を抑止しておく安全です。

7.9.1.4 イベントオブジェクトの target 属性

イベントオブジェクトの `target` 属性には、そのイベントが発生した HTML 要素が保持されています。イベント処理の際、この部分から対象の HTML 要素に関する様々な値（子要素、属性など）を取得することができます。

先に挙げたサンプル `DnD01.0.html` をドラッグアンドドロップに対応する形に改変した `DnD01.2.html` を示します。

サンプル：DnD01.2.html

```
1 <!DOCTYPE html>
2 <html lang="ja">
3 <head>
4 <meta charset="utf-8">
5 <title>DnDテスト1</title>
6
7 <style>
8 .dv {
9     position: absolute;
10    width: 200px; height: 80px;
11    border: solid 3px;
12    font-size: 24px;
13    text-align: center;
14    line-height: 80px;
15 }
16 #dv1 {
17     top: 5px; left: 5px;
18     border-color: blue;
19 }
20 #dv2 {
21     top: 5px; left: 240px;
22     border-color: red;
23 }
24 </style>
25
26 <script>
27 var dv1, dv2;
28
29 function f0() {
30     //--- dv1 をドラッグできるように設定 ---
31     dv1 = document.getElementById("dv1");
32     dv1.addEventListener("dragstart", function(e) {
33         e.dataTransfer.setData("text/plain", e.target.textContent);
34         console.log("ドラッグ開始: "+e.target.id);
35         console.log("    内容="+e.target.textContent);
36     }, false);
37     //--- dv2 がドラッグを受け付けるように設定 ---
38     dv2 = document.getElementById("dv2");
39     // ドラッグがさしかかった際の処理
40     dv2.addEventListener("dragover", function(e) {
41         e.dataTransfer.dropEffect = "copy";
42         e.preventDefault();
43         console.log("ドラッグ受け付け: "+e.target.id);
```

```

44     console.log("    データ："+e.dataTransfer.getData("text/plain"));
45 },false);
46 // ドロップされた際の処理
47 dv2.addEventListener("drop", function(e) {
48     e.preventDefault();
49     console.log("ドロップ受け付け："+e.target.id);
50     console.log("    データ："+e.dataTransfer.getData("text/plain"));
51 },false);
52 }
53 </script>
54 </head>
55
56 <body onLoad="f0()">
57 <div id="dv1" class="dv" draggable="true">これをドラッグ</div>
58 <div id="dv2" class="dv">ここへドラッグ</div>
59 </body>
60 </html>

```

このサンプルを Web ブラウザで実行するとドラッグアンドドロップの処理を行います。表示された青い枠をドラッグするとドラッグ開始時に

```

ドラッグ開始：dv1
内容=これをドラッグ

```

と Web ブラウザのコンソールに表示されます。このとき表示される「内容=」の後の文字列は、当該イベントオブジェクトの `target` 属性からテキスト要素 (`textContent` 属性の値) を取り出したものです。

青い枠を赤い枠の上に重ねると

```

ドラッグ受け付け：dv2
データ：これをドラッグ

```

が複数連続して Web ブラウザのコンソールに表示されます。

青い枠を赤い枠の上にドロップすると

```

ドロップ受け付け：dv2
データ：これをドラッグ

```

と Web ブラウザのコンソールに表示されます。

7.9.2 Web ブラウザ外部からドラッグアンドドロップを受け付ける処理

OS のデスクトップやフォルダのウィンドウからアイコンを Web ブラウザに対してドラッグアンドドロップすることができます。イベント処理の基本的な方法は先に取り上げたものとほぼ同じです。

ブラウザ外部にある画像ファイルのアイコンを HTML 文書中の `<div>` 要素にドラッグアンドドロップし、その画像を `<div>` 要素内に表示するサンプル `DnD02.html` を示して説明します。

サンプル：DnD02.html

```

1  <!DOCTYPE html>
2  <html lang="ja">
3  <head>
4  <meta charset="utf-8">
5  <title>DnDテスト2</title>
6
7  <style>
8  #tgt1 {
9      width: 200px;   height: 200px;
10     border: solid 8px gray;
11     padding: 5px;
12     text-align: center;
13 }
14 #img1 {
15     width: 120px;
16 }

```



```

17 #txt1 {
18     width: 220px;
19     border: solid 2px black;
20 }
21 </style>
22
23 <script>
24 var tgt1, img1, txt1;
25 // ファイルリーダー
26 var fr = new FileReader();
27 fr.onload = function(e) {
28     img1.src = e.target.result;
29     txt1.value = e.target.result;
30 }
31
32 function f0() {
33     tgt1 = document.getElementById("tgt1");
34     txt1 = document.getElementById("txt1");
35     img1 = document.getElementById("img1");
36     //--- tgt1 がドラッグを受け付けるように設定 ---
37     // ドラッグがさしかかった際の処理
38     tgt1.addEventListener("dragover", function(e) {
39         e.dataTransfer.dropEffect = "copy";
40         e.preventDefault();
41     }, false);
42     // ドロップされた際の処理
43     tgt1.addEventListener("drop", function(e) {
44         e.preventDefault();
45         fr.readAsDataURL(e.dataTransfer.files[0]);
46         console.log(e.dataTransfer.files[0].name);
47     }, false);
48 }
49 </script>
50 </head>
51
52 <body onLoad="f0()">
53 <div id="tgt1" class="dv">（ここへドラッグ）<br>
54 <img src="" alt="画像を表示します" id="img1"></div><br>
55 <input type="text" id="txt1">
56 </body>
57 </html>

```

このサンプルでは id 属性が tgt1 の <div> 要素がドロップを受け付けます。この要素の dragover イベントのハンドリングは先に解説した方法と同じです。ブラウザ外からドロップを受けると、そのイベントオブジェクトの dataTransfer 属性からドロップしたファイルのファイル名を取得することができます。具体的にはサンプルの 45 行目にあるように、

```
e.dataTransfer.files[0]
```

として、dataTransfer の files 属性からドロップしたファイルを参照しています。'[0]' とインデックスを指定していますが、これは複数のアイコンをまとめてドロップした場合のインデックスです。取得したファイルを、FileReader の readAsDataURL メソッドの引数に与えて内容を取得しています。このメソッドによるファイルの読み込みが終了すると、FileReader に onload イベントが発生するので、それを受けて 27～30 行目で 要素の src 属性にファイルの内容を与えています。同時に id 属性が txt1 のテキストフィールドにファイルの内容を表示します。このサンプルの動作を図 83 に示します。

画像をドロップすると Web ブラウザのコンソールにその画像のファイル名を表示します。ファイル名は dataTransferfile[インデックス] の name 属性から取得できます。

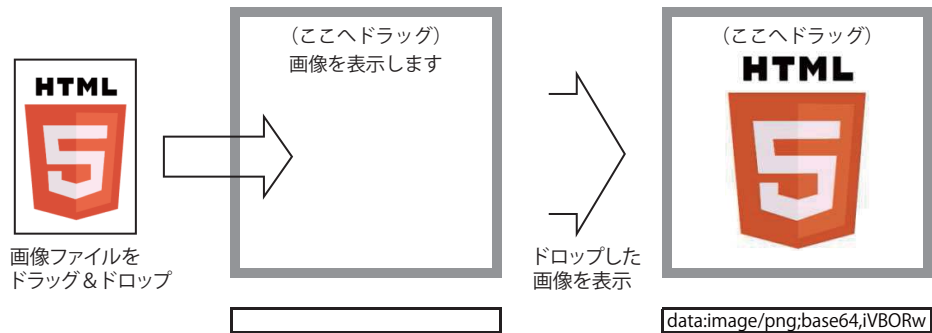


図 83: ブラウザ外のアイコンをドラッグ&ドロップ

7.10 インラインフレーム (iframe)

HTML ドキュメントの中に、別のドキュメントを配置する場合にはインラインフレーム `iframe` を使用します。インラインフレームを設置するには、次のようなタグを記述します。

```
<iframe src="読み込む文書の URL"> (iframe 非対応ブラウザで実行した際のメッセージ) </iframe>
```

【サンプルプログラム】

次に示すプログラム `test_iframe.html` は、入力した URL で示される HTML 文書を `iframe` で表示するものです。

サンプル：test_iframe.html

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>インラインフレーム</title>
6  <style>
7  #ifrm {
8      width: 480pt;
9      height: 360pt;
10 }
11 </style>
12 <script>
13 function f1() {
14     ifrm.src = loc.value;
15 }
16 </script>
17 </head>
18 <body>
19 別のHTMLコンテンツを表示します。 <br>
20 <input type="button" value="表示" onClick="f1()">
21 URL:<input type="text" id="loc" size="80"><br>
22 <iframe id="ifrm"></iframe>
23 </body>
24 </html>

```

このプログラムを Web ブラウザで実行すると図 84 のように表示されます。

次のような同一ディレクトリに存在する HTML 文書 `test_iframe_sub.html` を読み込んでフレーム内に表示した例を図 85 に示します。

サンプル：test_iframe_sub.html

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>別の文書</title>

```



「URL」フィールドに表示したい文書の URL を入力して **表示** をクリックするとフレーム内にそのコンテンツが表示される。

図 84: Web ブラウザによる testDateTime.html の表示

```

6 <style>
7 p {
8     font-size: 70pt;
9     font-weight: 900;
10 }
11 </style>
12 </head>
13 <body>
14 <p>これは別の HTML 文書です. </p>
15 </body>
16 </html>

```

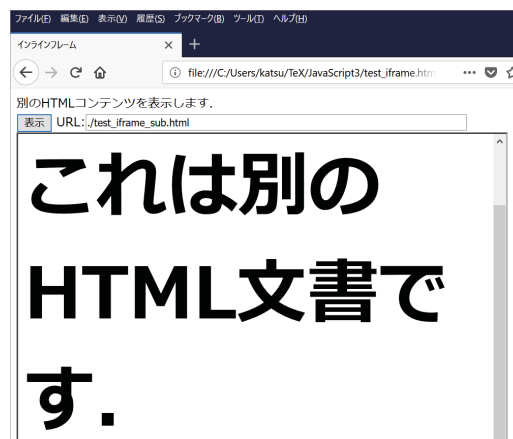


図 85: test_iframe_sub.html を表示した例

注意) どんな URL の文書でも iframe で配置できるとは限りません。iframe による配置を拒否する設定を施したサイトの HTML 文書は読み込むことができません。

8 DOMに基づくプログラミング

HTML 文書はタグの階層構造で構成されます。つまり、最上位の `<html> ~ </html>` 構造の中に、`<head> ~ </head>` と `<body> ~ </body>` があり、`<body> ~ </body>` の中に更に別のタグがあるという階層構造（木構造）です。このような階層構造の各部分を取り出したり、階層構造に新たな要素を追加するといったことが可能です。またこのような文書構造の扱いを可能にする枠組みのことを Document Object Model (DOM) といいます。

「5.11 JavaScript のオブジェクト階層」(p.68) で解説した内容に引き続き、ここでは、JavaScript によって HTML 文書を編集するための基本について説明します。

8.1 HTML の編集

HTML 文書はタグなどの要素の中に別のタグや、テキストノード（文字列要素）を含む構造を基本とします。例えば文章の段落は `<p> ~ </p>` の中にテキストノードを含む構造になっています。

例. `<p>` これは日本語の文章です。`</p>`

JavaScript には HTML の要素（タグ）やテキストノードを生成したり、それを既存の要素に追加するといった機能があり、それらの機能を用いることで HTML 文章を編集することが可能です。

8.1.1 基本的な編集機能

ここでは、最も基本的な編集機能として、要素の生成、テキストノードの生成、既存の要素への追加の方法について実例を挙げながら説明します。testDom1.html に示すプログラムは、ボタンをクリックする度に文章の段落を追加するものです。

サンプル：testDom1.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>DOMテスト1</title>
6  <script>
7  var n = 0;
8  function f1() {
9      var e, t, msg;
10     n++;
11     msg = "日本語のテキスト：" + n + "番目"; // テキスト内容
12     e = document.createElement("p");      // pタグを生成
13     t = document.createTextNode( msg );    // テキストノードを生成
14     e.appendChild(t);                     // pタグにテキストノードを追加
15     document.body.appendChild(e);         // bodyに追加
16 }
17 </script>
18 </head>
19 <body>
20 <input type="button" value="start" onClick="f1()"><br>
21 </body>
22 </html>
```

このプログラムを Web ブラウザで実行した様子を図 86 に示します。

このプログラムの中の

```
e = document.createElement("p");
```

は「p タグ」を生成して、それを e に得るものです。また、

```
t = document.createTextNode( msg );
```

は msg に格納された文字列からテキストノードを生成して t として得るものです。更に、

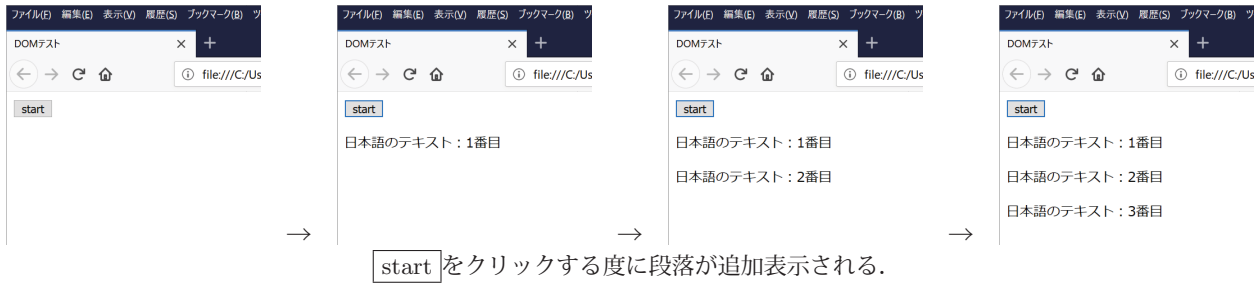


図 86: testDom1.html を実行した例

```
e.appendChild(t);
```

で、テキストノード `t` を `p` タグの要素である `e` に追加し、最終的に

```
document.body.appendChild(e);
```

で HTML 文書の `body` 要素に、生成した段落の要素である `e` を追加しています。

「5.9.1.1 要素の属性と値について」(p.63)の所で説明したように、HTML の要素に対する属性値の設定には `setAttribute` メソッドを使います。例えば要素 `e` に属性値を与えるには

```
e.setAttribute("属性名", "値")
```

とします。またあるいは「`e.属性名 = "値"`」という記述ができる属性⁴³も多いです。

本書で紹介した機能以外にも HTML 編集に関する多くの機能が JavaScript では利用できます。詳しくは他の資料(書籍やインターネットの情報提供サイトなど)を参照してください。

8.1.2 HTML の階層構造

JavaScript では、HTML 文書そのものは `document` というオブジェクトとして扱われます。従って、全ての要素はこの `document` の配下にあります。

8.1.2.1 要素の取得

`document` オブジェクト配下には `head` や `body` といったオブジェクトがあり、それらは

```
document.head
document.body
```

として参照されます。この様子を「`head` は `document` の子要素である」「`body` は `document` の子要素である」と表現します。このようにオブジェクトの階層関係をドット「`.`」で区切って表現します。ドットの表記は**要素の属性**を取り出すのにも使うことができます。例えば

```
<input type="text" value="123">
```

のような要素 `e` がある場合、その要素の `value` 属性の値を

```
e.value
```

として参照することができます。

`document` 要素に表 21 のようなメソッドを適用することで配下のオブジェクトを取得することもできます。

ある要素の直下に属する子要素は、その要素の `childNodes` プロパティから配列として取得すること⁴⁴ができます。サンプル `testDom2.html` は、配列として取得した子要素の `value` 属性を全て表示するものです。

⁴³`value` 属性などが代表的です。

⁴⁴`children` プロパティと違い、`childNodes` プロパティは テキストノードを含む全ての子要素を保持 します。

表 21: 要素を取得するメソッド

メソッド	説明
<code>getElementById("id 名")</code>	"id 名" (id 属性) で示される要素 (単体) を返す.
<code>getElementsByTagName("タグ名")</code>	"タグ名" で示されるタグ要素の配列を返す.
<code>getElementsByClassName("class 名")</code>	"class 名" (class 属性) で示される要素の配列を返す.
<code>getElementsByName("名前")</code>	"名前" (name 属性) で示される要素の配列を返す.

サンプル: testDom2.html

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>DOMテスト2</title>
6  <script>
7  function f1() {
8      var c, msg="";
9      var ar = document.getElementsByClassName("txt");
10     for ( c = 0; c < ar.length; c++ ) {
11         msg = msg + ar[c].id + " : " + ar[c].value + "\n";
12     }
13     ta.value = msg;
14 }
15 </script>
16 </head>
17 <body onload="f1()">
18 <input type="text" value="a" id="t1" class="txt"><br>
19 <input type="text" value="b" id="t2" class="txt"><br>
20 <input type="text" value="c" id="t3" class="txt"><br>
21 <textarea cols="20" rows="5" id="ta"></textarea>
22 </body>
23 </html>

```

このプログラムを Web ブラウザで実行した様子を図 87 に示します。

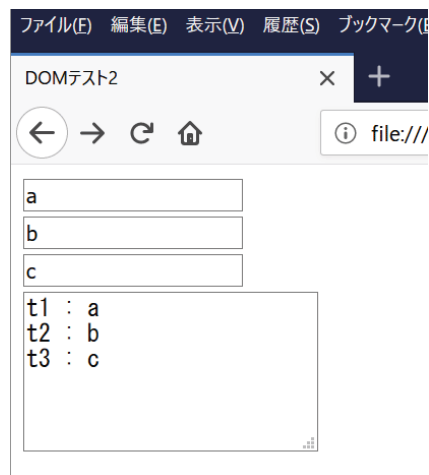


図 87: testDom2.html を実行した例

8.2 SVG の描画

Scalable Vector Graphics (SVG) は XML の階層構造で実現する**ドロー系グラフィックス**（線画図形：ベクターグラフィックス）です。これは**ペイント系グラフィックス**（ビットマップ）の図形とは異なり、表示の際の拡大縮小によって崩れることのないもので、グラフィックデザインの分野で用いられることが多いです。

SVG は DOM に基づく構造を基本とするので、JavaScript によって容易に生成することができます。本書では、SVG の生成と表示に関する基本的な事柄について説明します。SVG の描画要素は本書で説明するもの以外にも多くのものがあります。詳しいことは SVG に関する W3C の公式情報サイト⁴⁵を始めとする各種の情報を参照してください。

8.2.1 SVG の基本構造と HTML 文書内での配置

SVG は次のように **svg 要素**（svg タグ）を用いて記述します。

■ SVG の記述

```
<svg xmlns="http://www.w3.org/2000/svg">
  ⋮
  (図形描画の記述)
  ⋮
</svg>
```

描画する各種の図形は svg 要素の配下に（子要素として）記述します。また、SVG は単独のファイルとして作成することができます。1つの画像ファイルとして扱うことができます。その場合はファイル名の拡張子として‘~.svg’を与えます。

xmlns 属性の値として URL のようなものを与えていますが、これは**名前空間**と呼ばれるものです。SVG の要素を扱うための XML の名前空間は <http://www.w3.org/2000/svg> です。名前空間に関しては後ほど「8.2.3.1 XML の名前空間」(p.128)で説明します。

8.2.2 SVG の座標系と長さ、角度の単位

SVG の座標系は、SVG 要素の左上を原点 $(x, y) = (0, 0)$ として、右に行くほど x が大きく、下に行くほど y が大きくなります。長さの単位系は特にありませんが、通常は表示するデバイスの**ピクセル** (px) を単位とします。

SVG における角度の単位は「度」で、時計回りを正、反時計回りを負とします。

SVG の例として、矩形と円を描く `svg01.svg` を示します。

サンプル：svg01.svg

```
1 <svg xmlns="http://www.w3.org/2000/svg">
2   <title>矩形と円の描画</title>
3   <!-- 矩形の要素 -->
4   <rect x="20" y="20" width="280" height="170"
5     stroke="black" stroke-width="2" fill="white" />
6   <!-- 円の要素 -->
7   <circle cx="160" cy="105" r="50" fill="red" />
8 </svg>
```

SVG は画像ファイルとして Web ブラウザで開いて表示することができます。(図 88 参照)

SVG の要素として `<title> ~ </title>` を記述することもでき、タイトルを与えることができます。またドロー系の作画ソフトウェア⁴⁶には、SVG を画像データとして扱うことができるものもあります。

SVG の画像ファイルは、html の `img` 要素として組み込むことができます。(サンプル `svg01_1.html` 参照)

⁴⁵<https://www.w3.org/Graphics/SVG/>

⁴⁶商用製品では Adobe Illustrator が有名。フリーソフトでは Inkscape (<https://inkscape.org/en/>) があります。

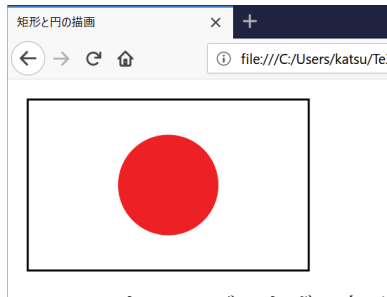


図 88: svg01.svg を Web ブラウザで表示した例

サンプル：svg01.1.html

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>SVGのテスト </title>
6  </head>
7  <body>
8  
9  </body>
10 </html>

```

SVG は XML の拡張であり，DOM の概念に基づいて HTML 文書の内部に直接的に配置することができます。（サンプル svg01.2.html 参照）

サンプル：svg01.2.html

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>SVGのテスト </title>
6  </head>
7  <body>
8  <svg xmlns="http://www.w3.org/2000/svg" width="300px" height="190px">
9    <title>矩形と円の描画</title>
10    <!-- 矩形の要素 -->
11    <rect x="20" y="20" width="280" height="170"
12          stroke="black" stroke-width="2" fill="white" />
13    <!-- 円の要素 -->
14    <circle cx="160" cy="105" r="50" fill="red" />
15  </svg>
16 </body>
17 </html>

```

8.2.2.1 ビューポートとビューボックス

SVG では特に width と height の属性で指定される表示領域のことをビューポートといいます。このビューポートに，SVG 座標系のどの位置のどの部分を表示するかを viewBox 属性で指定（図 89）することができます。

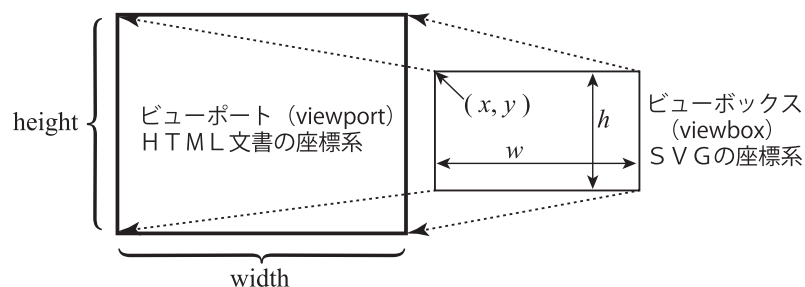


図 89: ビューポートとビューボックスの対応

例えば、図 89 において (x, y) を $(10, 20)$ 、 w, h をそれぞれ 400, 300 とするには、`viewBox="10 20 400 300"` という属性を SVG 要素に設定します。

`viewBox` の値を設定して、ビューポートに対応させるビューボックスを変化させるサンプルプログラムを `svg01_3.html` に示します。

サンプル：`svg01_3.html`

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>SVGのテスト</title>
6 <style>
7 #svg1 {
8     border: 1px solid black;
9 }
10 </style>
11 <script>
12 var x = 0, y = 0;           // viewBoxの位置
13 var dx = 300, dy = 190;     // viewBoxのサイズ
14
15 function f1() {
16     var vbox = sx.value + " " + sy.value + " " + sw.value + " " + sh.value;
17     tv.value = vbox;
18     tx.value = sx.value;
19     ty.value = sy.value;
20     tw.value = sw.value;
21     th.value = sh.value;
22     svg1.setAttribute( "viewBox", vbox );
23 }
24 </script>
25 </head>
26 <body onLoad="f1()">
27 ビューボックス<br>
28 x: <input type="range" min="0" max="300" value="0" id="sx" onInput="f1()">
29 <input type="text" id="tx"><br>
30 y: <input type="range" min="0" max="190" value="0" id="sy" onInput="f1()">
31 <input type="text" id="ty"><br>
32 w: <input type="range" min="100" max="600" value="400" id="sw" onInput="f1()">
33 <input type="text" id="tw"><br>
34 h: <input type="range" min="100" max="450" value="290" id="sh" onInput="f1()">
35 <input type="text" id="th"><br>
36 viewBox=<input type="text" id="tv"><br>
37 <svg xmlns="http://www.w3.org/2000/svg"
38     width="400px" height="290px" viewBox="0 0 400 290" id="svg1">
39     <title>矩形と円の描画（ビューポートの扱い）</title>
40     <!-- 矩形の要素 -->
41     <rect x="20" y="20" width="280" height="170"
42         stroke="black" stroke-width="2" fill="white" />
43     <!-- 円の要素 -->
44     <circle cx="160" cy="105" r="50" fill="red" />
45 </svg>
46 </body>
47 </html>
```

このプログラムを Web ブラウザで実行した様子を図 90 に示します。

x, y のスライダを変化させることでビューボックスの位置 (x, y) を、 w, h のスライダを変化させることでビューボックスの領域 w, h を変化⁴⁷させることができます。

⁴⁷アスペクト比を保つように表示されます。アスペクト比を変える方法もありますが本書では扱いません。

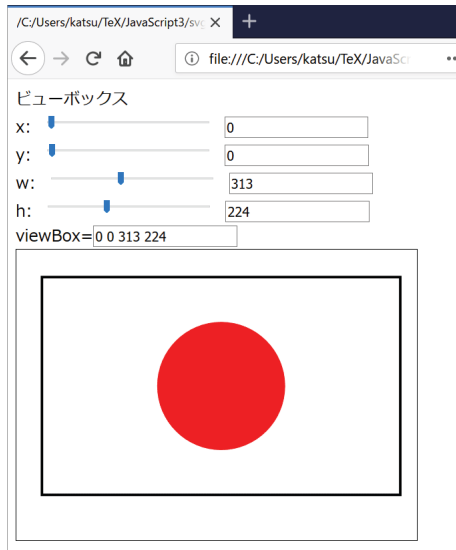


図 90: svg01_3.html を Web ブラウザで表示した例

8.2.3 JavaScript による SVG の描画

JavaScript で HTML 文書内に SVG を描画するための基本的な方法について説明します。先に

```
document.createElement("要素名")
```

による要素の生成について説明しましたが、名前空間を持つ XML の要素を生成するには、

```
document.createElementNS("名前空間", "要素名")
```

とします。

8.2.3.1 XML の名前空間

XML では要素を独自に定義して拡張できます。SVG も XML の拡張であり、SVG 独自の要素（タグ）の集合で構成されます。従って、SVG の各要素の名前は、SVG 以外の体系とは区別して定義する必要があり、そうしないと他の XML 体系と**名前の衝突**が起こってしまいます。そのような理由から、それぞれの XML 体系は独自の「名前空間」を持ち、他の XML 体系と同じ要素の名前（タグ名）があっても互いに干渉（衝突）することがありません。XML の拡張体系としては表 22 のようなものが有名です。

表 22: XML の拡張体系として有名なもの（一部）

体系	解説	名前空間
SVG	ベクターグラフィックスを記述するための体系。ファイル拡張子は *.svg	http://www.w3.org/2000/svg
MathML	数式記述するための体系 ファイル拡張子は *.mml	http://www.w3.org/1998/Math/MathML
Office Open XML	ワープロ文書や表計算データを構成するための体系。 ファイル拡張子は *.docx, *.xlsx	(説明省略)

この表からもわかるように、名前空間は URL (URI) の形をしています。

先の例で示したような SVG 図形を JavaScript で生成するプログラムを testSVG.html に示します。

サンプル：testSVG.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
```

```

4 <meta charset="utf-8">
5 <title>SVGの生成</title>
6 <script>
7 function f1() {
8     var svg1, e, t;
9     //--- svg要素の取得 ---
10    svg1 = document.getElementById("svg1");
11    //--- title要素の生成 ---
12    e = document.createElement("title");
13    t = document.createTextNode("矩形と円の描画");
14    e.appendChild(t);
15    svg1.appendChild(e);
16    //--- rect要素の生成 ---
17    e = document.createElementNS("http://www.w3.org/2000/svg", "rect");
18    // 属性の設定
19    e.setAttribute("x", "20");
20    e.setAttribute("y", "20");
21    e.setAttribute("width", "280");
22    e.setAttribute("height", "170");
23    e.setAttribute("stroke", "black");
24    e.setAttribute("stroke-width", "2");
25    e.setAttribute("fill", "white");
26    svg1.appendChild(e);
27    //--- circle要素の生成 ---
28    e = document.createElementNS("http://www.w3.org/2000/svg", "circle");
29    // 属性の設定
30    e.setAttribute("cx", "160");
31    e.setAttribute("cy", "105");
32    e.setAttribute("r", "50");
33    e.setAttribute("fill", "red");
34    svg1.appendChild(e);
35 }
36 </script>
37 </head>
38 <body>
39 <input type="button" value="SVG描画" onClick="f1()"><br>
40 <svg id="svg1" xmlns="http://www.w3.org/2000/svg" width="300px" height="190px">
41 </svg>
42 </body>
43 </html>

```

このプログラムを実行した様子を図 91 に示します。

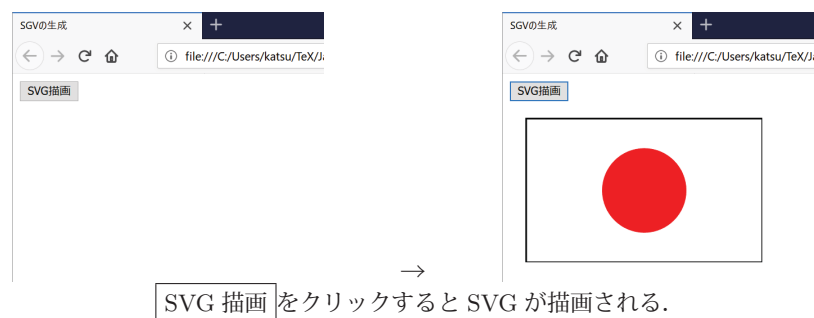


図 91: testSVG.html を実行した例

8.2.4 SVG の代表的な図形要素

8.2.4.1 多角形・折れ線

多角形は polygon 要素で、折れ線は polyline 要素で描画します。

〈 多角形 〉

書き方： `<polygon fill="塗りの色" stroke="ストロークの色" stroke-width="ストロークの太さ" points="x1,y1 x2,y2 … xn,yn" />`

解説：

座標 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ を頂点とする多角形を描く。色に none を指定すると無色（透明）となる。

〈 折れ線 〉

書き方： `<polyline fill="塗りの色" stroke="ストロークの色" stroke-width="ストロークの太さ" points="x1,y1 x2,y2 … xn,yn" />`

解説：

座標 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ を結ぶ折れ線を描く。色に none を指定すると無色（透明）となる。

polygon 要素で多角形を描く例を svg02.1.svg に示します。

サンプル：svg02.1.svg

```
1 <svg xmlns="http://www.w3.org/2000/svg">
2   <title>多角形</title>
3   <!-- 多角形（塗りつぶし） -->
4   <polygon fill="#FFFF00" stroke="#000000" stroke-width="6"
5     points="89.316,33.427 100.164,60.902 128.408,52.252 113.689,77.864
6       138.063,94.553 108.862,99.014 111.011,128.475 89.316,108.427
7       67.622,128.475 69.771,99.014 40.57,94.553 64.943,77.864
8       50.225,52.252 78.469,60.902" />
9   <!-- 多角形（塗りなし） -->
10  <polygon fill="none" stroke="#000000" stroke-width="6"
11    points="219.316,128.476 230.164,101 258.408,109.65 243.689,84.039
12      268.063,67.35 238.862,62.888 241.011,33.427 219.316,53.476
13      197.622,33.427 199.771,62.888 170.57,67.35 194.943,84.039
14      180.225,109.65 208.469,101" />
15 </svg>
```

これを Web ブラウザで表示した例を図 92 に示します。

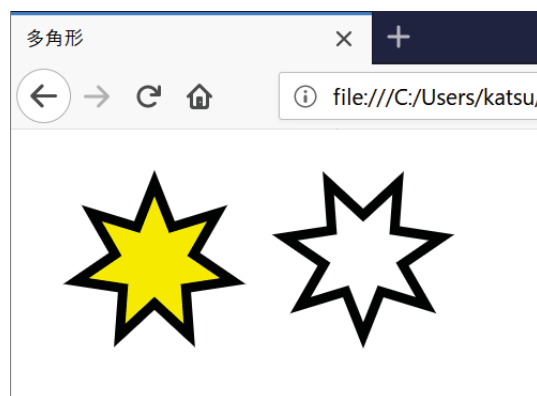


図 92: svg02.1.svg を Web ブラウザで表示した例

8.2.4.2 塗りとストローク、色の指定、曲がり角

SVG で描く線画図形は基本的にストローク（枠線）と塗りで構成されます。塗りの色を `fill` 属性に指定し、ストロークの色を `stroke` 属性に指定します。（色の設定に関しては p.11 「3.2.2.4 色の設定」を参照してください）またストロークの太さ（線幅）は `stroke-width` 属性に指定します。

ストロークを点線や破線にするには `stroke-dasharray` 属性を指定します。この属性には ”線の長さ 1, 間隔 1, 線の長さ 2, 間隔 2,…” という形で指定します。

サンプル：svg02.2.svg

```
1 <svg xmlns="http://www.w3.org/2000/svg">
2   <title>折れ線</title>
3   <polyline fill="none" stroke="#000000" stroke-width="10"
4     points="41,42 41,142 71,42 71,142" />
5   <polyline fill="none" stroke="#000000" stroke-width="3"
6     stroke-dasharray="2,3"
7     points="141,42 141,142 171,42 171,142" />
8 </svg>
```

これを Web ブラウザで表示した例を図 93 に示します。



図 93: svg02.2.svg を Web ブラウザで表示した例

多角形や折れ線の曲がり角の形状は `stroke-linejoin` 属性で指定します。この属性には図 94 に示すような 3 つの値 (`miter`, `round`, `bevel`) を指定することができます。これらの違いを示すプログラムが `svg02.4.svg` です。

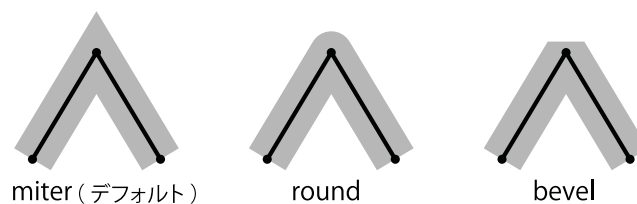


図 94: `linejoin`：線の曲がり角の形状

サンプル：svg02.4.svg

```
1 <svg xmlns="http://www.w3.org/2000/svg">
2   <title>折れ線（曲がり角の指定）</title>
3   <polyline fill="none" stroke="#000000" stroke-width="14"
4     stroke-linejoin="miter" stroke-miterlimit="10"
5     points="33,41 33,111 73,41 73,111" />
6   <polyline fill="none" stroke="#000000" stroke-width="14"
7     stroke-linejoin="round" stroke-miterlimit="10"
8     points="113,41 113,111 153,41 153,111" />
9   <polyline fill="none" stroke="#000000" stroke-width="14"
10    stroke-linejoin="bevel" stroke-miterlimit="10"
11    points="193,41 193,111 233,41 233,111" />
12 </svg>
```

このプログラムを Web ブラウザで実行した例が図 95 です。

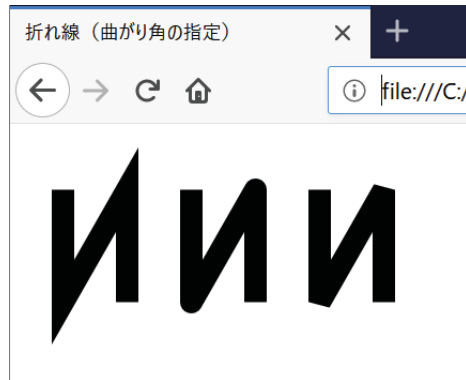


図 95: svg02_4.svg を Web ブラウザで表示した例

直線（後述）や折れ線といった、両端が開いた線の先端の形状は `stroke-linecap` 属性で指定します。この属性には図 96 に示すような 3 つの値 (`butt`, `round`, `square`) を指定することができます。



図 96: linecap: 線の先端の形状

多角形の頂点や折れ線の曲がり角が鋭くなって張り出すことがあります。このような線の角の張り出しを制限するためには `stroke-miterlimit` 属性を適切に設定します。

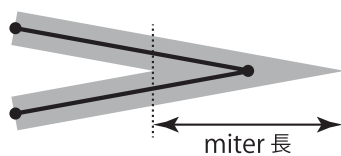


図 97: $\text{miterlimit} = \text{miter 長} / \text{線幅}$

`stroke-miterlimit` 属性には $\text{miter 長} \div \text{線幅}$ の値を設定します。これにより、その値を超えるような張り出しが起これる場合は、曲がり角が `bevel` になります。(図 97 参照) この様子がわかるサンプルを `svg02.5.svg` に示します。

サンプル: `svg02.5.svg`

```
1 <svg xmlns="http://www.w3.org/2000/svg">
2   <title>折れ線 (留幅の指定)</title>
3   <polyline fill="none" stroke="#000000" stroke-width="10"
4     stroke-miterlimit="100"
5     points="35,79 35,179 105,79 105,179 145,79
6     145,179 165,79 165,179 "/>
7   <polyline fill="none" stroke="#000000" stroke-width="10"
8     stroke-miterlimit="10"
9     points="245,79 245,179 315,79 315,179 355,79
10    355,179 375,79 375,179 "/>
11 </svg>
```

このプログラムを Web ブラウザで実行した様子を図 98 に示します。



図 98: svg02_5.svg を Web ブラウザで表示した例

図 98 でわかるように、右に行く程鋭利になってゆく折れ線の角が、`miterlimit` の制限により `bevel` になっていることがわかります。

8.2.4.3 パス

パス (path) は多角形 (polygon) や折れ線 (polyline) と同等の描画に加え、円弧やベジェ曲線⁴⁸を含むことができる柔軟な描画要素です。

〈パス：基本〉

書き方： `<path fill="塗りの色" stroke="ストロークの色" stroke-width="ストロークの太さ" d="描画コマンド列" />`

解説：

`d` 属性に与える「描画コマンド列」に従ってパスを描く..

【描画コマンド列】

■ 直線による描画過程

基本的な考え方は「始点を指定して、終点に至るまでの頂点の座標を与える」というものです。始点は **M コマンド** (MoveTo)，頂点は **L コマンド** (LineTo) で指定します。

例. `d="M 0 0 L 100 0 L 100 100"`

この例では、座標 (0,0) を始点として描画を開始し、頂点 (100,0), (100,100) に向けて開いたパス (折れ線) を描きます。閉じたパス (多角形) を描くには、次の例のように、コマンド列の最後に **Z コマンド** を記述します。

例. `d="M 0 0 L 100 0 L 100 100 Z"`

これで始点と最後の座標 (100,100) が結ばれて多角形 (三角形) が描画されます。

M コマンドと Z コマンドは小文字 (`m,z`) で記述しても意味は同じですが、L コマンドを小文字 (`l`) で記述すると、現座標からの**相対座標** (移動分) の指定となります。

折れ線と多角形を path 要素で描くサンプルを `svg02_6.svg` に示します。

サンプル：`svg02_6.svg`

```
1 <svg xmlns="http://www.w3.org/2000/svg">
2   <title>パス1</title>
3   <!-- 開いたパス -->
```

⁴⁸Paul de Casteljau (仏), Pierre Bézier (仏) らによって考案された曲線。多くのベクターグラフィックス用ソフトウェアで曲線描画に採用されています。本書ではベジェ曲線についての解説は割愛します。詳しくは W3C の公式コンテンツを参照してください。

```

4      <path d="M 20 20 L 100 20 L 100 100"
5          stroke="#000000" stroke-width="8" fill="#ffff00" />
6      <!-- 閉じたパス -->
7      <path d="M 120 20 L 200 20 L 200 100 Z"
8          stroke="#000000" stroke-width="8" fill="#ffff00" />
9      <!-- 相対座標指定で描くパス -->
10     <path d="M 220 20 l 80 0 l 0 80 Z"
11         stroke="#000000" stroke-width="8" fill="#ffff00" />
12 </svg>

```

このプログラムを Web ブラウザで実行した様子を図 99 に示します。



図 99: svg02_6.svg を Web ブラウザで表示した例

直線描画には L コマンド以外にも、水平にのみ描画する **H コマンド** があり、‘H’ に続いて描画先の水平座標を 1 つ与えます。(‘h’ コマンドの場合は水平移動量を指定します) また同様に、垂直にのみ描画する **V コマンド** があり、‘V’ に続いて描画先の垂直座標を 1 つ与えます。(‘v’ コマンドの場合は垂直移動量を指定します)

■ 楕円弧

基本的な考え方は「始点と終点を指定して、それらを通る楕円弧」というもので、**A コマンド** (Arc) で描画します。また、始点と終点以外にも楕円弧の元になる楕円の水平半径と垂直半径の大きさ、それに図 100 に示すような条件 (4 種類) を与えることで楕円弧が 1 つ定まります。

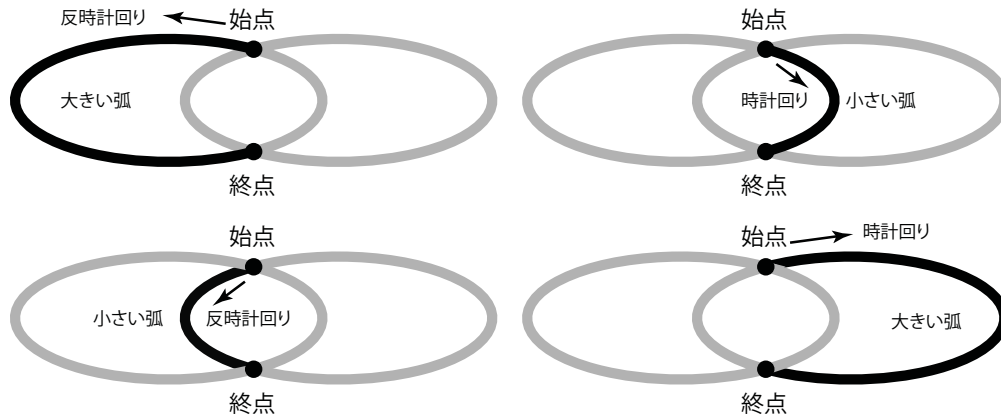


図 100: 楕円弧を決定する 4 つの条件 (弧の大小と回転方向)

更に、楕円弧の元になる楕円の傾きを指定することもできます。A コマンドの記述は次の通りです。

"A 楕円の水平半径 楕円の垂直半径 楕円の傾き 弧の大小 回転方向 終点の x 座標 終点の y 座標"

このように A コマンドは 7 個の値を伴います。始点は直前の描画コマンドの最終位置で、楕円の中心の位置は自動的に算出されます。

「弧の大小」には 1 (大) か 0 (小) を、回転方向には 1 (時計回り) か 0 (反時計回り) を指定します。

A コマンドで楕円弧を描画するサンプルを svg02_7.svg に示します。

サンプル: svg02_7.svg

```

1 <svg xmlns="http://www.w3.org/2000/svg">
2   <title>パス2</title>
3   <!-- 楕円弧: 始点-終点(250,30)-(250,150), 水平半径150, 垂直半径60 -->

```

```

4      <!-- 大きい方の楕円弧，反時計回り -->
5      <path d="M 250 50 A 150 60 0 1 0 250 150"
6            stroke="#000000" stroke-width="8" fill="#ffff00" />
7      <!-- 小さい方の楕円弧，時計回り -->
8      <path d="M 250 50 A 150 60 0 0 1 250 150"
9            stroke="#ff0000" stroke-width="8" fill="#00ff00" />
10     </svg>

```

これを Web ブラウザで実行した様子を図 101 に示します。

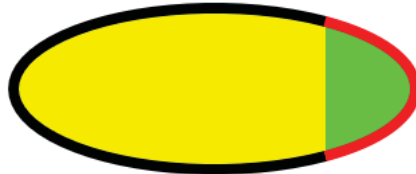


図 101: svg02.7.svg を Web ブラウザで表示した例

A コマンドを小文字 (a) で記述した場合は，終点の座標が始点の座標の相対位置（移動分）となります。（svg02.7-2.svg 参照）

サンプル：svg02.7-2.svg

```

1  <svg xmlns="http://www.w3.org/2000/svg">
2    <title>パス2</title>
3    <!-- 楕円弧：始点-終点(250,30)-(250,150)，水平半径150，垂直半径60 -->
4    <!-- 大きい方の楕円弧，反時計回り -->
5    <path d="M 250 50 a 150 60 0 1 0 0 100"
6          stroke="#000000" stroke-width="8" fill="#ffff00" />
7    <!-- 小さい方の楕円弧，時計回り -->
8    <path d="M 250 50 a 150 60 0 0 1 0 100"
9          stroke="#ff0000" stroke-width="8" fill="#00ff00" />
10  </svg>

```

これは先の svg02.7.svg による描画と同じ結果となります。

A コマンドの「楕円の傾き」に回転させる角度（10°）を与える例を svg02.7-3.svg に示します。

サンプル：svg02.7-3.svg

```

1  <svg xmlns="http://www.w3.org/2000/svg">
2    <title>パス2</title>
3    <!-- 楕円弧：始点-終点(250,30)-(250,150)，水平半径150，垂直半径60 -->
4    <!-- 大きい方の楕円弧，反時計回り -->
5    <path d="M 250 50 a 150 60 10 1 0 0 100"
6          stroke="#000000" stroke-width="8" fill="#ffff00" />
7    <!-- 小さい方の楕円弧，時計回り -->
8    <path d="M 250 50 a 150 60 10 0 1 0 100"
9          stroke="#ff0000" stroke-width="8" fill="#00ff00" />
10  </svg>

```

これを Web ブラウザで実行した様子を図 102 に示します。

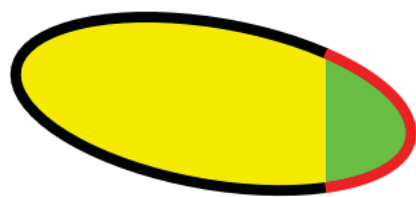


図 102: svg02.7-3.svg を Web ブラウザで表示した例

8.2.4.4 矩形, 円, 楕円, 直線, 文字

■ 矩形 (四角形) : rect 要素

rect 要素はストロークと塗りに関する属性に加えて次のような属性を持ちます.

属性	意味	属性	意味
x	矩形の左上の座標の横位置	y	矩形の左上の座標の縦位置
width	矩形の横幅	height	矩形の高さ
rx	角の丸みの横半径	ry	角の丸みの縦半径

■ 円 : circle 要素

circle 要素はストロークと塗りに関する属性に加えて次のような属性を持ちます.

属性	意味	属性	意味	属性	意味
cx	中心の座標の横位置	cy	中心の座標の縦位置	r	半径

■ 楕円 : ellipse 要素

ellipse 要素はストロークと塗りに関する属性に加えて次のような属性を持ちます.

属性	意味	属性	意味
cx	中心の座標の横位置	cy	中心の座標の縦位置
rx	横方向の半径	ry	縦方向の半径

■ 直線 : line 要素

line 要素はストロークに関する属性に加えて次のような属性を持ちます.

属性	意味	属性	意味	属性	意味	属性	意味
x1	始点の横位置	y1	始点の縦位置	x2	終点の横位置	y2	終点の縦位置

■ テキスト (文字列) : text 要素

text 要素はアウトラインフォントで文字列を描画するものでストロークと塗りの属性を持ちます. この要素は次のような形で記述します.

〈text 属性記述〉描画対象のテキスト </text>

text 要素は次のような属性を持ちます.

属性	意味	属性	意味
x	始点の横位置	y	始点の縦位置
rotate	個々の文字の角度	font-size	フォントの大きさ
font-family	フォント名	font-weight	フォントの太さ lighter / normal / bold / bolder
font-style	フォントのスタイル normal / italic / oblique	font-decoration	文字の装飾 underline / overline / line-through

円, 楕円, 直線, 文字を描画するサンプルを `svg03.svg` に示します.

サンプル：svg03.svg

```
1 <svg xmlns="http://www.w3.org/2000/svg">
2   <title>円, 楕円, 直線, テキスト</title>
3   <circle fill="none" stroke="#000000" stroke-width="10"
4     cx="79.52" cy="74.52" r="42.52"/>
5   <ellipse fill="none" stroke="#000000" stroke-width="10"
6     cx="261.252" cy="74.52" rx="99.213" ry="42.52"/>
7   <line stroke="#000000" stroke-width="10"
8     x1="37" y1="144.52" x2="360.465" y2="144.52"/>
9   <text x="38" y="184.5195" font-family="sans-serif" font-size="24">
10     これは日本語の文字列です</text>
11 </svg>
```

svg03.svg を Web ブラウザで実行した例を図 103 に示します。



図 103: svg03.svg を Web ブラウザで表示した例

8.2.5 SVG 図形要素のイベントハンドリング

SVG の図形要素もイベントハンドリングの対象とすることができます。SVG の図形要素に対してマウスクリックのイベントハンドリングを行う例を svg04.html に示します。

サンプル：svg04.html

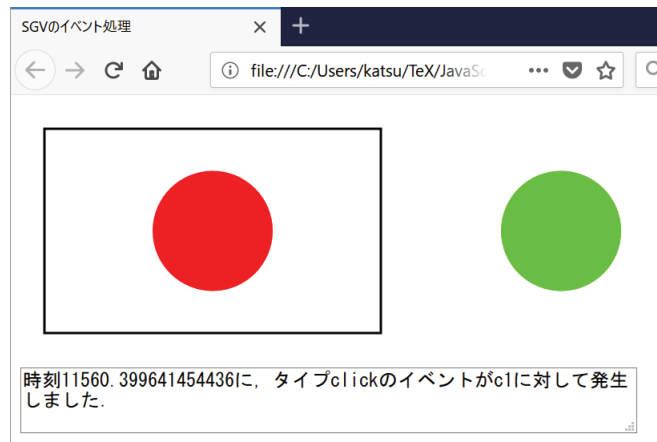
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title>SGVのイベント処理 </title>
6   <script>
7     //--- イベント処理 ---
8     function f1(m) {
9       ta.value = "時刻" + m.timeStamp + "に, タイプ" + m.type +
10         "のイベントが" + m.target.id + "に対して発生しました. ";
11     }
12
13     //--- イベントリスナの登録 ---
14     function f0() {
15       r1.addEventListener("click",f1,false);
16       c1.addEventListener("click",f1,false);
17       e1.addEventListener("click",f1,false);
18     }
19   </script>
20 </head>
21 <body onLoad="f0()">
22   <svg id="svg1" xmlns="http://www.w3.org/2000/svg"
23     width="600px" height="210px">
24     <rect x="20" y="20" width="280" height="170"
25       fill="white" stroke="black" stroke-width="2" id="r1" />
26     <circle cx="160" cy="105" r="50" stroke-width="0" fill="red" id="c1" />
27     <ellipse cx="450" cy="105" rx="50" ry="50"
28       stroke-width="0" fill="#00FF00" id="e1" />
```

```

29 </svg><br>
30 <textarea id="ta" cols="60" rows="2"></textarea>
31 </body>
32 </html>

```

svg04.html を Web ブラウザで実行した例を図 104 に示します。



図中の左側にある赤い円をクリックして、
テキストエリアにメッセージが表示されている。

図 104: svg04.html を Web ブラウザで表示した例

プログラム中の関数 f0 が HTML の body 要素が読み込まれた直後 (onLoad) に実行され、図形要素にイベントハンドラ (関数 f1) を登録しています。

8.2.6 SVG 要素のグループ化

<g>〜</g> で複数の SVG 要素を括るとそれらをグループ化することができます。グループ化されたオブジェクトは 1 つのオブジェクトとしてイベントハンドリングの対象とすることができます。(svg05.html)

サンプル：svg05.html

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>SGVのグループ化 </title>
6  <script>
7  var gr, tx1;
8
9  function f0( ) {
10     tx1 = document.getElementById("tx1");
11     gr = document.getElementById("gr");
12     gr.addEventListener("click",f1,false);
13 }
14
15 function f1( e ) {
16     tx1.value = "pos="+e.pageX+","+e.pageY+"";
17 }
18 </script>
19 </head>
20 <body onLoad="f0( )">
21 <svg id="svg1" xmlns="http://www.w3.org/2000/svg"
22     width="600px" height="210px">
23     <g id="gr">
24         <rect x="20" y="20" width="280" height="170"
25             fill="white" stroke="black" stroke-width="2" />
26         <circle cx="160" cy="105" r="50" stroke-width="0" fill="red" />
27     </g>
28 </svg><br>

```

```
29 <input type="text" id="tx1">  
30 </body>  
31 </html>
```

svg05.html を Web ブラウザで実行した例を図 105 に示します.

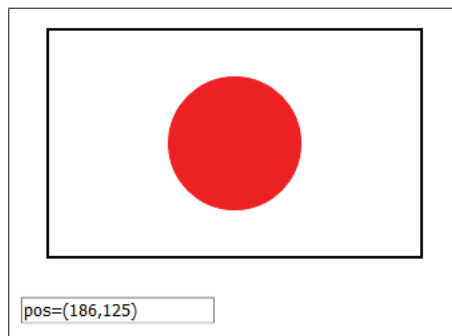


図 105: svg04.html を Web ブラウザで表示した例

このサンプルでは、矩形と円が `id="gr"` としてグループ化されており、1つのイベントハンドリングでクリックを検知できます.

8.2.7 参考：SVG 関連のライブラリ

SVG グラフィックス生成のためのライブラリが公開されていますので、いくつか紹介します.

ライブラリ	公式サイト
SVG.js	https://svgjs.com/
GraphicsJS	http://www.graphicsjs.org/
Vanilla JS	http://vanilla-js.com/
Raphaël	http://dmitrybaranovskiy.github.io/raphael/

これらのライブラリを導入することで、高度な SVG グラフィックスを簡単な方法で作成することができます.

9 オブジェクト指向プログラミング

従来の JavaScript では、function 文を使用してオブジェクト指向プログラミングを実現⁴⁹ していましたが、新しい規格（ECMAScript 2015 あるいは ES6）に沿った JavaScript では、他の言語処理系と同じように class 文が使えます。本書では新しい規格に沿った形でオブジェクト指向プログラミングについて説明します。

9.1 クラスの定義

クラスは class 文を用いて定義します。

書き方： class クラス名 { 定義内容 }

他のクラス（スーパークラス）を継承して、拡張クラスとして定義する場合は次のように記述します。

書き方： class クラス名 extends スーパークラス名 { 定義内容 }

9.1.1 コンストラクタ

クラスのインスタンスを生成する際のコンストラクタは constructor を用いて次のように記述します。

書き方： constructor(引数並び) { 定義内容 }

9.1.2 メソッド

メソッドは次のように記述します。

書き方： メソッド名(引数並び) { 定義内容 }

メソッドが値を返す場合は return 文を使用します。自分自身のインスタンスは this で参照します。クラスメソッドを定義する場合は定義するメソッド名の前に static を記述します。

9.2 応用例：SVG 描画クラスの実装

ここでは、SVG グラフィックスを描画するためのクラスライブラリを簡易な形で実装する例を挙げて、JavaScript のオブジェクト指向プログラミングについて紹介します。

試作したクラスライブラリを SVGdraw.js に示します。

サンプル：SVGdraw.js

```
1  /*****
2  *   SVG 描画ライブラリ
3  *****/
4
5  // SVG要素（SVGタグ）の指定
6  class SvgElement {
7      // コンストラクタ：SVGタグのidを引数に与える
8      constructor( id ) {
9          this.target = document.getElementById( id );
10     }
11     // 描画メソッド
12     draw( grobj ) {
13         this.target.appendChild( grobj.element );
14     }
15 }
```

⁴⁹プロトタイプベースのプログラミングスタイルと呼ばれます。

```

16
17 // 描画オブジェクトクラスのスーパークラス
18 class GrObj {
19     // コンストラクタ：SVG描画要素を引数に与える
20     constructor( elem ) {
21         this.element = document.createElementNS(
22             "http://www.w3.org/2000/svg",elem);
23     }
24     // ストロークと塗に関する属性の設定
25     set_stroke( cl ) {
26         this.element.setAttribute("stroke",cl);
27     }
28     set_stroke_width( w ) {
29         this.element.setAttribute("stroke-width",w);
30     }
31     set_fill( cl ) {
32         this.element.setAttribute("fill",cl);
33     }
34 }
35
36 // 矩形オブジェクトクラス
37 class GrRect extends GrObj {
38     // コンストラクタ
39     constructor( x,y, w,h ) {
40         super("rect"); // スーパークラスのコンストラクタの呼び出し
41         this.element.setAttribute("x",x);
42         this.element.setAttribute("y",y);
43         this.element.setAttribute("width",w);
44         this.element.setAttribute("height",h);
45     }
46     // 位置とサイズに関する属性の設定
47     set_x( x ) {
48         this.element.setAttribute("x",x);
49     }
50     set_y( y ) {
51         this.element.setAttribute("y",y);
52     }
53     set_width( width ) {
54         this.element.setAttribute("width",width);
55     }
56     set_height( height ) {
57         this.element.setAttribute("height",height);
58     }
59 }
60
61 // 円オブジェクトクラス
62 class GrCircle extends GrObj {
63     // コンストラクタ
64     constructor( cx, cy, r ) {
65         super("circle"); // スーパークラスのコンストラクタの呼び出し
66         this.element.setAttribute("cx",cx);
67         this.element.setAttribute("cy",cy);
68         this.element.setAttribute("r",r);
69     }
70     // 中心の位置と半径に関する属性の設定
71     set_cx( cx ) {
72         this.element.setAttribute("cx",cx);
73     }
74     set_cy( cy ) {
75         this.element.setAttribute("cy",cy);
76     }
77     set_r( r ) {
78         this.element.setAttribute("r",r);
79     }
80 }
81
82 // 楕円オブジェクトクラス
83 class GrEllipse extends GrObj {
84     // コンストラクタ

```

```

85     constructor( cx, cy, rx, ry ) {
86         super("ellipse");    // スーパークラスのコンストラクタの呼び出し
87         this.element.setAttribute("cx",cx);
88         this.element.setAttribute("cy",cy);
89         this.element.setAttribute("rx",rx);
90         this.element.setAttribute("ry",ry);
91     }
92     // 中心の位置と半径に関する属性の設定
93     set_cx( cx ) {
94         this.element.setAttribute("cx",cx);
95     }
96     set_cy( cy ) {
97         this.element.setAttribute("cy",cy);
98     }
99     set_rx( rx ) {
100        this.element.setAttribute("rx",rx);
101    }
102    set_ry( ry ) {
103        this.element.setAttribute("ry",ry);
104    }
105 }

```

9.2.1 実装したクラスライブラリの解説

● SVG 要素を生成するためのクラス

6～15 行目でクラス `SvgElement` を定義しています。このクラスは、JavaScript で扱う SVG 要素をインスタンスとして生成するためのもので、HTML の中の描画対象となる SVG 要素の id 名を、インスタンス生成時の引数として与えます。このクラスのインスタンスに対して `draw` メソッドを実行すると、対象となる SVG 要素に図形要素が描画されます。draw メソッドの引数に描画したいオブジェクトを与えます。

● 描画オブジェクトのスーパークラス

18～34 行目で、各種図形オブジェクト（矩形、円、楕円）の元になるスーパークラス `GrObj` を定義しています。このクラスは、各種図形オブジェクトが共通して持つ属性（ストローク、塗り）を設定するメソッド `set_stroke`, `set_stroke_width`, `set_fill` を実装しています。

● 矩形オブジェクトのクラス

37～59 行目で、矩形オブジェクトのクラス `GrRect` を定義しています。このクラスは、位置とサイズをコンストラクタの引数として受け取って矩形オブジェクトを生成します。また生成後に位置とサイズを変更するためのメソッド `set_x`, `set_y`, `set_width`, `set_height` を実装しています。

● 円オブジェクトのクラス

62～80 行目で、円オブジェクトのクラス `GrCircle` を定義しています。このクラスは、中心の位置と半径をコンストラクタの引数として受け取って円オブジェクトを生成します。また生成後に中心の位置と半径を変更するためのメソッド `set_cx`, `set_cy`, `set_r` を実装しています。

● 楕円オブジェクトのクラス

83～105 行目で、楕円オブジェクトのクラス `GrEllipse` を定義しています。このクラスは、中心の位置と縦横の半径をコンストラクタの引数として受け取って楕円オブジェクトを生成します。また生成後に中心の位置と縦横の半径を変更するためのメソッド `set_cx`, `set_cy`, `set_rx`, `set_ry` を実装しています。

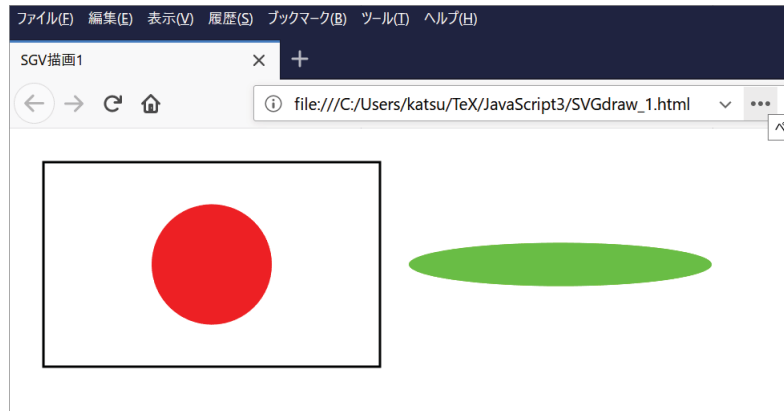
9.2.2 実装したクラスライブラリの使用例

クラスライブラリ `SVGdraw.js` を使用して描画するサンプルを `SVGdraw_1.html` に示します。これは矩形と円を描画した後、楕円の形が変形するアニメーションを表示するものです。

サンプル：SVGdraw_1.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>SGV描画1</title>
6  <script src="SVGdraw.js"></script>
7  <script>
8  var svg1, r1, c1, e1;
9  var t, rx=50, ry=50, dx=1, dy=-1;
10
11 //--- 楕円の大きさが変化するアニメーション
12 function f2() {
13     // 横半径の変化
14     rx += dx;
15     if ( rx > 127 ) {    // 大きくなりすぎた際は反転
16         rx = 127;
17         dx *= -1;
18     } else if ( rx < 0 ) { // 小さくなりすぎた際は反転
19         rx = 0;
20         dx *= -1;
21     }
22     // 横半径の変化
23     ry += dy;
24     if ( ry > 97 ) {    // 大きくなりすぎた際は反転
25         ry = 97;
26         dy *= -1;
27     } else if ( ry < 0 ) { // 小さくなりすぎた際は反転
28         ry = 0;
29         dy *= -1;
30     }
31     // 縦横の半径の変更を実行
32     e1.set_rx( rx );
33     e1.set_ry( ry );
34     t = setTimeout(f2,1);    // 次の処理へ
35 }
36
37 //--- 矩形と円の描画，アニメーションの開始
38 function f1() {
39     svg1 = new SvgElement("svg1");    // SVG要素の取得
40     r1 = new GrRect( 20,20, 280,170 );    // 矩形オブジェクトの生成
41     r1.set_fill("white");
42     r1.set_stroke("black");
43     r1.set_stroke_width( 2 );
44     svg1.draw(r1);    // SVG要素への登録
45     c1 = new GrCircle( 160,105, 50 );    // 円オブジェクトの生成
46     c1.set_stroke_width( 0 );
47     c1.set_fill("red");
48     svg1.draw(c1);    // SVG要素への登録
49     e1 = new GrEllipse( 450,105, 50, 50 );    // 楕円オブジェクトの生成
50     e1.set_stroke_width( 0 );
51     e1.set_fill("#00FF00");
52     svg1.draw(e1);    // SVG要素への登録
53     f2();    // アニメーションの開始
54 }
55 </script>
56 </head>
57 <body onLoad="f1()">
58 <svg id="svg1" xmlns="http://www.w3.org/2000/svg"
59     width="600px" height="300px"></svg>
60 </body>
61 </html>
```

これを Web ブラウザで実行した様子を図 106 に示します。



右側の楕円は大きさを変化させるアニメーションである。

図 106: SVGdraw_1.html を Web ブラウザで表示した例

■ 演習課題

ここで試作した SVGdraw.js を拡張して、多角形や折れ線、テキストと、描画できるオブジェクトを増やしてみてください。

※ SVG 描画のためのライブラリとして実用的なものがいくつか公開されており、
「8.2.7 参考：SVG 関連のライブラリ」(p.139) に紹介したものが有名です。

10 正規表現

文字列のパターンを表現するものに**正規表現**があります。正規表現は ある規則 に則って記述されるものです。例えば '[0-9]+' は正規表現によるパターンの1つで「'0'～'9' までの数字の記号」を表します。また '+' は「その直前のパターンで表現される文字が1回以上連続して並んでいる」ことを意味します。この2つの表現を次のように組み合わせ

'[0-9]+'

と記述すると「数字の記号が並んだ文字列」を意味する表現となります。

正規表現を応用すると、与えられた文字列がある形式になっているかを判定（**パターンマッチ**）したり、「ある規則に沿った文字列」を文書データの中から**探索（検索）**する、あるいはパターンにマッチする部分を別の文字列に**置換**するといった処理が実現できます。

10.1 文字列探索（検索）： search メソッド

文字列の中にあるパターンを探し出すには search メソッドを使います。

書き方： 対象文字列.search(正規表現)

「対象文字列」の中から「正規表現」のパターンを探して、最初に検出した位置のインデックスを返します。正規表現は '/…/' で括って記述します。

例. パターン検索

```
s = "abcd1234efgh";          ←文字列の作成
r = s.search( /[0-9]+/ );     ←文字列中の数字列を検索
```

この結果 r に 4 が得られます。対象文字列の中にパターンが見つからない場合は戻り値は -1 になります。

10.2 パターンマッチ： match メソッド

文字列の中から指定したパターンに一致する部分を取り出すには match メソッドを使います。

書き方： 対象文字列.match(正規表現)

「対象文字列」の中から「正規表現」のパターンを探して、最初に検出したものを返します。正規表現は '/~/ ' で括って記述します。

例. パターンマッチ

```
s = "0123abcd4567efgh";      ←文字列の作成
r = s.match( /[a-z]+/ );      ←文字列パターンに合う部分を取り出す
```

この結果 r にはパターンに合う部分が配列 ['abcd'] の形で得られます。対象文字列の中に抽出するパターンが見つからない場合は戻り値は null になります。

抽出したい部分が複数ある場合はそれらの正規表現を括弧 '(~)' で括って**グループ化**します。

例. パターンマッチのグループ化

```
s = "0123abcd4567efgh89";    ←文字列の作成
r = s.match( /([a-z]+)[0-9]+([a-z]+)/ ); ←取り出す部分を '(~)' で括る
```

この結果 r は配列 ['abcd4567efgh', 'abcd', 'efgh'] の形で得られます。得られた配列の先頭要素（インデックス:0）は元の文字列が、それ以降（インデックス:1以降）に括弧 '(~)' で括ったパターンに合うものが順番に配置されています。

10.3 置換処理： replace メソッド

文字列の中の指定したパターンに一致する部分を別のものに置き換える（置換処理）には replace メソッドを使います。

書き方： 対象文字列.replace(正規表現, 置換文字列)

「対象文字列」の中から「正規表現」のパターンを探して、最初に検出した部分を「置換文字列」に置き換えます。正規表現は「/~/」で括って記述します。

例. 置換処理

```
s = "0123abcd4567efgh89";      ←文字列の作成
r = s.replace( /[0-9]+/, "数字" );    ←最初の数字列を "数字" に置き換える
```

この結果 r に "数字 abcd4567efgh89" が得られます。

■ パターンに合う全ての部分の置換処理

replace メソッドの第 1 引数に与える正規表現の後ろに 'g' を付けると、対象文字列の中のパターンに合う全ての部分を置換します。

例. 置換処理（全て）

```
s = "0123abcd4567efgh89";      ←文字列の作成
r = s.replace( /[0-9]+/g, "数字" );    ←全ての数字列の部分を "数字" に置き換える
```

この結果 r に "数字 abcd 数字 efgh 数字" が得られます。

10.4 正規表現の記述方法

「/~/」の中に記述できるもの（一部）を表 23～24 に示します。

表 23: 正規表現のパターン（一部）

パターン	解説
<code>[c₁c₂c₃...]</code>	特定の文字の集合 <code>c₁c₂c₃...</code> （これらの内のどれかに該当）
<code>[c₁-c₂]</code>	<code>c₁~c₂</code> の間に含まれる文字. 文字の範囲をハイフン '-' でつなげる. 例: <code>[a-d]</code> → 'a', 'b', 'c', 'd' のどれか
<code>[^文字の集合]</code>	「文字の集合」にマッチしないもの
<code>.</code> （ドット）	任意の 1 文字
<code>\d</code>	数字（ <code>[0-9]</code> と同じ）
<code>\D</code>	数字以外
<code>\s</code>	空白文字（タブ, 改行文字なども含む）
<code>\S</code>	空白文字以外（ <code>\s</code> でないもの）
<code>\w</code>	アンダースコアを含む半角英数字（ <code>[a-zA-Z0-9_]</code> と同じ）
<code>\W</code>	半角英数字以外（ <code>\w</code> でないもの）

注意) 円記号「¥」は端末装置によってはバックスラッシュ「\」として表示される。

表 24: 繰り返しの表記（一部）

表記	解説	表記	解説
<code>+</code>	1 回以上	<code>*</code>	0 回以上
<code>?</code>	0 回かもしくは 1 回	<code>{m,n}</code>	m 回以上 n 回以下（回数が多い方を優先）
<code>{n}</code>	n 回	<code>{m,}</code>	m 回以上

「/~/」の中に正規表現のパターンでない文字列を記述すると、「その文字列に一致するもの」を意味します。

11 ライブラリ

ライブラリ（プログラムライブラリ）は、汎用性あるいは有用性の高い関数やメソッド（クラス）などを提供するものです。JavaScript にも多くのライブラリが公開されており、本書では特に人気の高いものを取り上げて紹介します。

11.1 jQuery

jQuery は米国のプログラマであるジョン・レシグによって開発された JavaScript 用のライブラリです。jQuery は、CSS を操作する機能、イベントハンドリングに関する機能、DOM を操作する機能、グラフィカルな効果を実現する機能など、様々な機能を提供します。jQuery を HTML5 コンテンツに導入することによって、高度なコンテンツ操作の機能を短く簡単に記述できるようになります。

jQuery 本体と関連する情報は、公式インターネットサイト

<https://jquery.com/>

から入手することができます。また、公式インターネットサイト以外にも、有志の方々が運営する情報サイトが多数あり、書籍も多数出版されています。本書では jQuery の導入に必要な事柄や基礎知識についてのみ説明します。

11.1.1 jQuery を使用するための準備

jQuery は先のインターネットサイトからダウンロードして使用します。本書のこの章を執筆している時点では jQuery の版は 3.4.1 で、jQuery 本体は

`jquery-3.4.1.js`

というファイルです。HTML コンテンツで jQuery を使用するには `<script>` タグでこのファイルを読み込みます。例えば、HTML コンテンツと同じディレクトリに jQuery のライブラリがある場合は、

```
<script type="text/javascript" src="jquery-3.4.1.js"></script>
```

と記述することで jQuery ライブラリを読み込むことができます。jQuery ライブラリ本体は、HTML コンテンツからアクセスできる任意のディレクトリに配置することができます。その場合は上記の `<script>` の記述において

```
src="jQuery ライブラリファイルのパス"
```

としてライブラリを読み込みます。

11.1.2 jQuery の基本的な使い方

jQuery では html 要素の指定や関数（メソッド）実行の記述を、「\$」記号を用いた独特な形で表記します。このことは次のサンプル `jquery00-1.html` で確認できます。これは HTML 文書中の文字に色を設定するだけの単純なもの（図 107 参照）です。

これはjQueryのテストです。

図 107: 文字の色を設定するだけの例

サンプル：jquery00-1.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>jQueryテスト0</title>
5 <!-- jQuery読み込み -->
6 <script src="jquery-3.4.1.js"></script>
7 <!-- jQueryを使用したスクリプトの記述 -->
```

```

8  <script>
9  function f0() {
10     $("#tx1").css("color","red");
11  }
12  </script>
13  </head>
14  <!-- 文書本体 -->
15  <body onLoad="f0()">
16  <p id="tx1">これは jQuery のテストです. </p>
17  </body>
18  </html>

```

このサンプルでは、id 属性が "tx1" である p 要素「これは jQuery のテストです。」の色を赤に設定する処理を実行しています。

jQuery では**セレクタ**（要素を示すもの）を文字列の形で記述し `$(...)` で括り、それに対してドット「`.`」を付けてメソッドを記述します。この際、`$(...)` の中に複数の対象をコンマ「`,`」で区切って列挙することができます。また、要素の上下関係（親子関係）をスペースで区切って書き並べることができます。直接の上下関係を表現するには「親 > 子」と記述します。

サンプル `jquery00-1.html` では jQuery の `css` メソッドを使用して、対象のスタイル要素 `"color"` に赤を意味する `"red"` を設定しています。また、それら処理は、関数 `f0` として定義されており、`body` 要素が読み込まれた際のイベント `onLoad` を受けて `f0` が起動する形になっています。

11.1.2.1 CSS の要素へのアクセス

■ 値の設定

`css` メソッドの第一引数に設定対象の CSS 要素を、第二引数に設定する値を与えて実行します。

■ 値の取得

`css` メソッドの第一引数に対象の CSS 要素を与えて実行すると、その値を返します。

11.1.2.2 `$`で関数を実行する方法

`$(...)` は `function` の記述自体を括ることもでき、その場合はコンテンツの読み込みが完了した時点（DOM が出来上がった時点）でその `function` 記述が実行⁵⁰ されます。この記述形式を応用すると、スクリプトの記述全体を簡潔な形に上げることができます。（`jquery00.html` 参照）

サンプル：`jquery00.html`

```

1  <html>
2  <head>
3  <meta charset="utf-8">
4  <title>jQueryテスト0</title>
5  <!-- jQuery読み込み -->
6  <script src="jquery-3.4.1.js"></script>
7  <!-- jQueryを使用したスクリプトの記述 -->
8  <script>
9  $(function() {
10     $("#tx1").css("color","red");
11  });
12  </script>
13  </head>
14  <!-- 文書本体 -->
15  <body>
16  <p id="tx1">これは jQuery のテストです. </p>
17  </body>
18  </html>

```

⁵⁰厳密には `load` のタイミングではなく、`ready` のタイミングで実行します。

このサンプルは先の jquery00-1.html と同じ動作をするものですが、body にイベントハンドリング (onLoad) の記述をしていません。

11.1.2.3 イベントハンドリングの登録

jQuery は、HTML 要素にイベントハンドリングを登録するための簡便な手段を提供します。次に示すサンプル jquery01-1.html は、ボタンのクリックにより文字の色を変える (図 108 参照) コンテンツです。これを例にして、マウスのクリックをハンドリングする方法について説明します。



図 108: ボタンのクリックに応じて色を変える

サンプル：jquery01-1.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>jQueryテスト1</title>
5 <!-- jQuery読み込み -->
6 <script src="jquery-3.4.1.js"></script>
7 <!-- jQueryを使用したスクリプトの記述 -->
8 <script>
9 // 「黒」 ボタンクリック時の処理 (定義)
10 function f1(){
11     $("#tx1").css("color", "black");
12 }
13 // 「赤」 ボタンクリック時の処理 (定義)
14 function f2(){
15     $("#tx1").css("color", "red");
16 }
17 // jQuery関連処理の定義
18 function f0(){
19     $("#bBlack").click( f1 );    // 「黒」 ボタンクリック時の処理 (登録)
20     $("#bRed").click( f2 );     // 「赤」 ボタンクリック時の処理 (登録)
21 }
22 // jQuery関連処理の登録
23 $( f0 );
24 </script>
25 </head>
26 <!-- 文書本体 -->
27 <body>
28 <p id="tx1">これはjQueryのテストです. </p>
29 <input type="button" value="黒" id="bBlack">
30 <input type="button" value="赤" id="bRed">
31 </body>
32 </html>
```

このサンプルで定義されている関数 f1, f2 は文字 (id="tx1") の色を設定するもので、それぞれ黒、赤の色に設定します。それら関数を「黒」(id="bBlack")「赤」(id="bRed") のボタンがクリックされたときに起動するようにイベントハンドリングを設定するのが関数 f0 です。この関数の中では、それぞれのボタンに対して click メソッドを使用して起動する関数を設定しています。

■ click メソッドによるイベントハンドリングの登録

`$(HTML 要素).click(起動する関数)`

jQuery では、関数として function の記述自体を与えることができるので、jquery01-1.html をもっと簡潔に仕上げることができます。それが次の jquery01.html です。

サンプル：jquery01.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>jQueryテスト1</title>
5 <!-- jQuery読み込み -->
6 <script src="jquery-3.4.1.js"></script>
7 <!-- jQueryを使用したスクリプトの記述 -->
8 <script>
9 $(function(){
10     //--- 「黒」 ボタンクリック ---
11     $("#bBlack").click(function(){
12         $("#tx1").css("color", "black");
13     });
14     //--- 「赤」 ボタンクリック ---
15     $("#bRed").click(function(){
16         $("#tx1").css("color", "red");
17     });
18 });
19 </script>
20 </head>
21 <!-- 文書本体 -->
22 <body>
23 <p id="tx1">これはjQueryのテストです. </p>
24 <input type="button" value="黒" id="bBlack">
25 <input type="button" value="赤" id="bRed">
26 </body>
27 </html>
```

click メソッド以外にも、HTML 要素にイベントハンドリングを登録するためのメソッドが多数あり、それらメソッド名は基本的にはイベント名と同じです。

■ on メソッドによるイベントハンドリングの登録

on メソッドでイベントハンドリングを登録することができます。先の click メソッドは「マウスのクリック」のイベントハンドリングの登録に限定されますが、on メソッドの場合は引数にイベント名を指定できるので、より汎用性が高いです。on メソッドを使用する際の書き方を次に示します。

`$(HTML 要素).on("イベント名", 起動する関数)`

on メソッドを用いて先の jquery01.html を書き換えることができます。その場合<script> 内を次のようにします。

```
$(function(){
    //--- 「黒」 ボタンクリック ---
    $("#bBlack").on("click", function(){
        $("#tx1").css("color", "black");
    });
    //--- 「赤」 ボタンクリック ---
    $("#bRed").on("click", function(){
        $("#tx1").css("color", "red");
    });
});
```

on メソッドの第一引数には複数のイベントの種類を空白（スペース）で区切って与えることができます。このことを次のサンプル jquery01-3.html で示します。

サンプル：jquery01-3.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
```

```

4 <title>jQueryテスト1-3</title>
5 <style>.obj { border: solid 2px black;}</style>
6 <script src="jquery-3.4.1.js"></script>
7 <script>
8 let msg = "", n = 1;
9 function makeMsg(e) {
10     let msg2 = n+") id:"+e.target.id+", type:"+e.type+", timeStamp:"+e.timeStamp;
11     if ( e.type == "keyup" ) {
12         msg2 += ", key:"+e.key+"\n";
13     } else {
14         msg2 += "\n";
15     }
16     msg = msg2 + msg;
17     ta1.value = msg;
18     n++;
19 }
20
21 $(function(){
22     //--- イベントハンドリングの登録 ---
23     $("#ta1").on("click mouseenter mouseleave", function(e){
24         makeMsg(e);
25     });
26     $("#tx1").on("click keyup", function(e){
27         makeMsg(e);
28     });
29 });
30 </script>
31 </head>
32 <!-- 文書本体 -->
33 <body>
34 textarea(ta1):<br>
35 <textarea cols="55" rows="10" id="ta1" class="obj" disable></textarea><br>
36 text(tx1):<input type="text" value="" size="40" id="tx1" class="obj">
37 </body>
38 </html>

```

これは、コンテンツ内のテキストエリアとテキストフィールドに発生したイベントの内訳をテキストエリア内に表示するものです。on メソッドで複数のイベントのハンドリングを設定しています。(23～28 行目)

このサンプルを実行した様子を図 109 に示します。

textarea(ta1):

```

8) id:tx1, type:keyup, timeStamp:7508, key:c
7) id:tx1, type:keyup, timeStamp:7057, key:Backspace
6) id:tx1, type:keyup, timeStamp:6712, key:d
5) id:tx1, type:keyup, timeStamp:5806, key:b
4) id:tx1, type:keyup, timeStamp:5416, key:a
3) id:tx1, type:click, timeStamp:4431
2) id:ta1, type:mouseleave, timeStamp:3430
1) id:ta1, type:mouseenter, timeStamp:1897

```

text(tx1) abc

図 109: 発生したイベントの表示

11.1.3 DOM の操作

jQuery には DOM の操作のための機能が提供されています。

11.1.3.1 子要素の追加

親要素へ子要素を追加するメソッド append について説明します。このメソッドの使い方を次に示します。

`$(親要素).append(子要素)`

append メソッドを用いて HTML の <body> 要素の下に <p> 要素を追加するサンプル jquery02.html を示します。

サンプル：jquery02.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>jQueryテスト2</title>
5 <!-- jQuery読み込み -->
6 <script src="jquery-3.4.1.js"></script>
7 <!-- jQueryを使用したスクリプトの記述 -->
8 <script>
9 var ele = "<p>追加要素</p>";
10
11 $(function(){
12     //--- 「要素を増やす」 ボタンクリック ---
13     $("#btn").click(function(){
14         $("body").append( ele );
15     });
16 });
17 </script>
18 </head>
19 <!-- 文書本体 -->
20 <body>
21 <input type="button" value="要素を増やす" id="btn">
22 </body>
23 </html>
```

このサンプルでは、ボタン（id="btn"）のクリックによって <body> 要素に"<p>追加要素</p>" を追加します。このサンプルの実行結果の例を図 110 に示します。

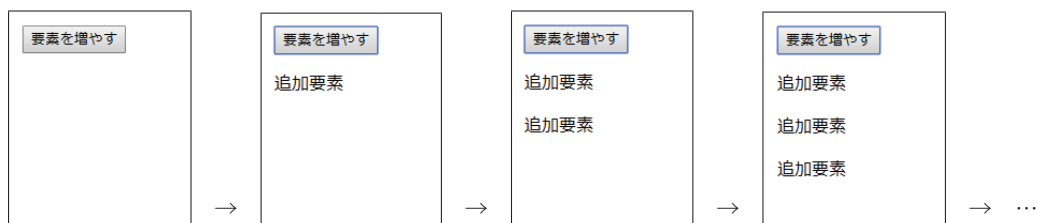


図 110: ボタンのクリックに応じて要素が増える

append メソッドと類似の appendTo メソッドもあり、これは、親要素と子要素の指定が逆になっているメソッドです。（下記参照）

`$(子要素).appendTo(親要素)`

11.1.3.2 テキストの取得と設定

対象要素が持つテキストを取り出す、あるいは対象要素にテキストを与えるには text メソッドを使用します。

テキストを与える方法： `対象要素.text(テキスト)`

テキストを取得する方法： `対象要素.text()`

対象の要素にテキストを設定するには text メソッドの引数にテキストを与えます。また、引数を省略すると、対象要素からテキストを取得してそれを返します。

次に示すサンプル jquery06.html は html の <p> 要素にテキストを追加する処理を実現しています。

サンプル：jquery06.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>jQueryテスト6</title>
5 <!-- jQuery読み込み -->
6 <script src="jquery-3.4.1.js"></script>
7 <!-- jQueryを使用したスクリプトの記述 -->
```

```

8 <script>
9 $(function(){
10     //--- ボタン ---
11     $("#btn").click(function(){
12         var tx = $("#bun").text();
13         tx += "追加の文です. ";
14         $("#bun").text(tx);
15     });
16 });
17 </script>
18 </head>
19 <!-- 文書本体 -->
20 <body>
21 <input type="button" value="文の追加" id="btn">
22 <p id="bun">最初の文です. </p>
23 </body>
24 </html>

```

このサンプルでは、<p> 要素に対して text メソッドを実行してテキストを取り出し、それに文字列を追加延長して、同じ <p> 要素に text メソッドを実行してそれを与える処理をしています。結果として、<p> 要素のテキストが編集されています。図 111 に実行例を示します。

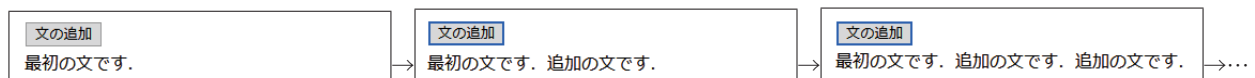


図 111: ボタンのクリックに応じてテキストを更新する

11.1.3.3 HTML の取得と設定

対象要素が配下に持つ HTML を取り出す、あるいは対象要素の配下に HTML を与えるには html メソッドを使用します。次に示すサンプル jquery07.html は html の <body> 要素以下の内容を編集する機能を実現しています。

サンプル：jquery07.html

```

1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>jQueryテスト7</title>
5 <!-- jQuery読み込み -->
6 <script src="jquery-3.4.1.js"></script>
7 <!-- jQueryを使用したスクリプトの記述 -->
8 <script>
9 var ele = "<p>追加のp要素です. </p>";
10
11 $(function(){
12     //--- ボタン ---
13     $("#btn").click(function(){
14         var d = $("body").html();
15         d = $.trim(d) + ele;
16         $("body").html( d );
17     });
18 });
19 </script>
20 </head>
21 <!-- 文書本体 -->
22 <body>
23 <input type="button" value="要素の追加" id="btn">
24 <p id="bun">最初のp要素です. </p>
25 </body>
26 </html>

```

このサンプルでは、<body> 要素に対して html メソッドを実行して HTML を取り出し、それに<p>要素を追加して、新たに出来た HTML を <body> 要素に html メソッドによって与える処理をしています。結果として、<p> 要素が追

加されています。図 112 に実行例を示します。

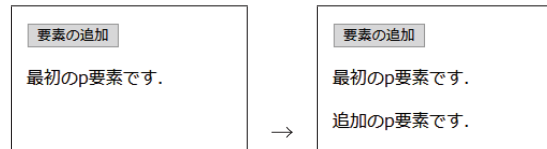


図 112: ボタンのクリックで<p>要素が追加される

■ 注意 ■

サンプル jquery07.html では html メソッドで <body> 要素の内容が書き換えられているので、ボタン (id="btn") に登録されていたイベントハンドリングは無効になります。そのため、「要素の追加」ボタンは 2 回目以降は無効となります。html メソッドによって書き換えられた要素に対して再度イベントハンドリングを登録するには何らかの工夫が必要になります。例えば、次に示す jquery07-2.html のような実装にするのも 1 つの方法です。

サンプル：jquery07-2.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>jQueryテスト7（改） </title>
5 <!-- jQuery読み込み -->
6 <script src="jquery-3.4.1.js"></script>
7 <!-- jQueryを使用したスクリプトの記述 -->
8 <script>
9 var ele = "<p>追加の p 要素です. </p>";
10
11 function f1() {
12     //--- ボタン ---
13     $("#btn").click(function(){
14         var d = $("body").html();
15         d = $.trim(d) + ele;
16         $("body").html( d );
17         f1();    // イベントハンドリングの再登録
18     });
19 }
20
21 $( f1 );
22 </script>
23 </head>
24 <!-- 文書本体 -->
25 <body>
26 <input type="button" value="要素の追加" id="btn">
27 <p id="bun">最初の p 要素です. </p>
28 </body>
29 </html>
```

このサンプルでは、<body> 要素が書き換えられる毎にイベントハンドリングを登録し直す形となっており、「要素の追加」ボタンは 2 回目も機能します。

11.1.4 セレクタの扱い

11.1.4.1 n 番目の要素の選択 (:nth-child)

jQuery のセレクタには、要素の順番などを指定する表記 :nth-child が使えます。

n 番目の要素の指定： "要素:nth-child(n)"

奇数番目の要素の指定： "要素:nth-child(odd)"

偶数番目の要素の指定： "要素:nth-child(even)"

これらを \$(...) の中に記述します。これを応用して、指定した要素の color 属性を設定するサンプルを jquerySel01.html に示します。これを Web ブラウザで表示すると図 113 のようになります。

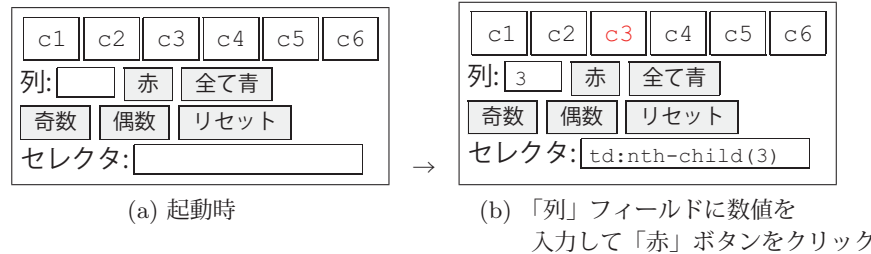


図 113: 数値で要素を選択して色を変える

何番目の要素の色を変えるか、その数値を入力し、該当する要素の color 属性を red にします。(21 行目) また、その際に使用したセレクトタの記述（文字列）を「セレクトタ」フィールドに表示します。(22 行目)

サンプル：jquerySel01.html

```

1  <!DOCTYPE html>
2  <html lang="ja">
3  <head>
4  <meta charset="utf-8">
5  <title>セレクトタの扱い</title>
6  <style>
7  td {
8      border: solid 1px black;
9      text-align: center;
10     width: 32px;
11 }
12 input {
13     border: solid 1px black;
14 }
15 </style>
16 <script src="jquery-3.4.1.js"></script>
17 <script>
18 $(function () {
19     $("#b1").on("click",function() {    // 「赤」ボタン
20         var n = tx1.value;
21         var sel = "td:nth-child("+n+")";    // セレクトタを合成
22         tx2.value = sel;
23         $(sel).css("color","red");        // セレクトタを赤にする
24     });
25     $("#b2").on("click",function() {    // 「奇数」ボタン
26         $("td:nth-child(odd)").css("color","red");    // 奇数枠を赤にする
27         tx2.value = "";
28     });
29     $("#b3").on("click",function() {    // 「偶数」ボタン
30         $("td:nth-child(even)").css("color","red");    // 偶数枠を赤にする
31         tx2.value = "";
32     });
33     $("#b4").on("click",function() {    // 「リセット」ボタン
34         $("td").css("color","black");
35         tx2.value = "";
36     });
37     $("#b5").on("click",function() {    // 「全て青」ボタン
38         $("tr > *").css("color","blue");    // このようなセレクトタの書き方も可能
39         tx2.value = "";
40     });
41 })
42 </script>
43 </head>
44 <body>
45 <table>
46 <tr><td>c1</td><td>c2</td><td>c3</td><td>c4</td><td>c5</td><td>c6</td></tr>
47 </table>
48 列:<input type="text" size="2" id="tx1">
49 <input type="button" value="赤" id="b1">
50 <input type="button" value="全て青" id="b5"><br>
51 <input type="button" value="奇数" id="b2">
52 <input type="button" value="偶数" id="b3">
53 <input type="button" value="リセット" id="b4"><br>

```

```

54 セレクタ:<input type="text" id="tx2">
55 </body>
56 </html>

```

c1	c2	c3	c4	c5	c6
列:	<input type="text"/>	赤	全て青		
奇数	偶数	リセット			
セレクタ: <input type="text"/>					

(a) 「偶数」 ボタンをクリック

c1	c2	c3	c4	c5	c6
列:	<input type="text"/>	赤	全て青		
奇数	偶数	リセット			
セレクタ: <input type="text"/>					

(b) 「奇数」 ボタンをクリック

c1	c2	c3	c4	c5	c6
列:	<input type="text"/>	赤	全て青		
奇数	偶数	リセット			
セレクタ: <input type="text"/>					

(c) 「全て青」 ボタンをクリック

図 114: 奇数番目, 偶数番目の要素の選択

:nth-child による要素の選択では, 指定する要素の 番号は 1 から始まる という点が重要です. これは, 配列などのデータ構造の要素のインデックスが 0 から始まるということと異なるので注意が必要です.

ここで説明した方法は, 文字列の形でセレクタを記述するものですが, 次に説明する eq では, インデックスの数値 (n 番目の値) を直接メソッドの引数に与えることができます.

11.1.4.2 n 番目の要素の選択 (eq)

\$(...) で選択された要素に対して eq(n) メソッドを使用することで, 更に n 番目のものを選択することができます. 先のサンプル jquerySel01.html と似た働きを eq で実現した jquerySel02.html を示します.

サンプル: jquerySel02.html

```

1  <!DOCTYPE html>
2  <html lang="ja">
3  <head>
4  <meta charset="utf-8">
5  <title>セレクタの扱い</title>
6  <style>
7  td {
8      border: solid 1px black;
9      text-align: center;
10     width: 32px;
11 }
12 input {
13     border: solid 1px black;
14 }
15 </style>
16 <script src="jquery-3.4.1.js"></script>
17 <script>
18 $(function () {
19     $("#b1").on("click",function() {    // 「赤」 ボタン
20         var n = tx1.value;
21         $("td").eq(n).css("color","red");    // セレクタを赤にする
22     });
23     $("#b2").on("click",function() {    // 「リセット」 ボタン
24         $("td").css("color","black");
25     });
26 })
27 </script>
28 </head>
29 <body>
30 <table>
31 <tr><td>c0</td><td>c1</td><td>c2</td><td>c3</td><td>c4</td><td>c5</td></tr>
32 </table>
33 列:<input type="text" size="2" id="tx1">
34 <input type="button" value="赤" id="b1">
35 <input type="button" value="リセット" id="b2"><br>
36 </body>
37 </html>

```

21 行目で \$("td") に対して eq(n) を用いて n 番目の要素を選択し、その color 属性を red に設定しています。これを Web ブラウザで表示すると図 115 のようになります。

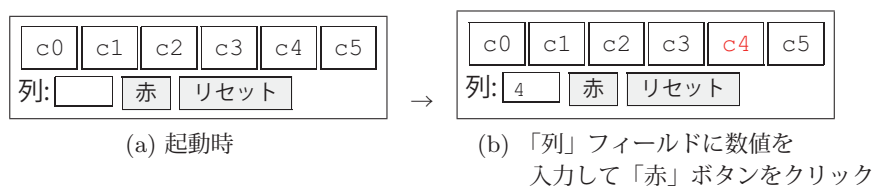


図 115: 数値で要素を選択して色を変える

eq の引数に与えるインデックスは 0 から始まるものです。先の :nth-child の場合は 1 から始まりますが、混同しないように注意が必要です。

11.1.4.3 属性値で要素を選択

\$(".[属性値=値]") と記述することで「属性値」が「値」である要素を選択します。これを応用したサンプル jquerySel03.html を示します。

サンプル：jquerySel03.html

```

1  <!DOCTYPE html>
2  <html lang="ja">
3  <head>
4  <meta charset="utf-8">
5  <title>セレクトの扱い</title>
6  <style>
7  .txt {
8      width: 35px;
9      margin: 2px;
10 }
11 input {border:solid 1px black}
12 </style>
13 <script src="jquery-3.4.1.js"></script>
14 <script>
15 $(function () {
16     $("#b1").on("click",function() {
17         $('[value="X"]').css("background-color","yellow");
18     });
19 })
20 </script>
21 </head>
22 <body>
23 <input type="button" value="Xを検出" id="b1"><br>
24 <input type="text" class="txt" value="0"><input type="text" class="txt" value="0">
25 <input type="text" class="txt" value="X"><input type="text" class="txt" value="0">
26 <input type="text" class="txt" value="0"><input type="text" class="txt" value="0">
27 <input type="text" class="txt" value="0"><input type="text" class="txt" value="X">
28 <br>
29 <input type="text" class="txt" value="X"><input type="text" class="txt" value="0">
30 <input type="text" class="txt" value="0"><input type="text" class="txt" value="0">
31 <input type="text" class="txt" value="X"><input type="text" class="txt" value="0">
32 <input type="text" class="txt" value="0"><input type="text" class="txt" value="0">
33 </body>
34 </html>

```

このサンプルは、value="0" の input 要素が多数並んでおり、所々に value="X" である input 要素が混在しています。Web ブラウザで表示すると図 116 の (a) のように表示されます。「X を検出」ボタンをクリックすると、図 116 の (b) のように value="X" である input 要素の背景色 (background-color) が黄色になります。

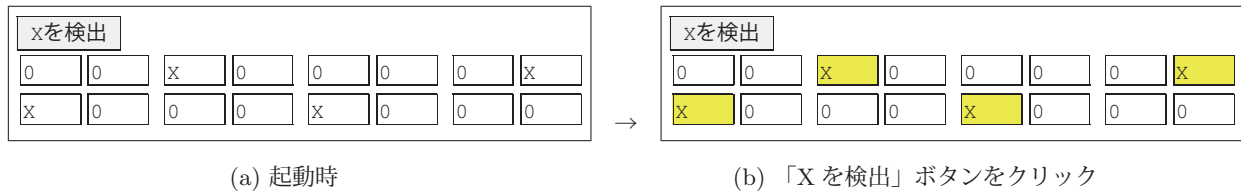


図 116: value="X" の要素を検出

- jQuery には、ここで紹介したメソッド以外にも DOM 関連のメソッドが多数提供されています。詳しくは公式インターネットサイトなどを参照してください。

11.1.5 視覚効果と高度な GUI

jQuery には高度な GUI を簡単に構築するための機能や、視覚的な効果を実現するための機能が提供されています。

11.1.5.1 要素を表示する／隠す

ここでは jQuery が提供する最も基本的な視覚効果为例 jquery04.html を挙げて紹介します。このコンテンツの実行例の一部を図 117 に示します。



図 117: 実行例：拡大しながら現れる「スライドダウン」

サンプル：jquery04.html

```

1  <html>
2  <head>
3  <meta charset="utf-8">
4  <title>jQueryテスト4</title>
5  <!-- スタイル -->
6  <style>
7  #jqr {
8      position: absolute;
9      top: 40px;
10     left: 10px;
11     width: 550px;
12 }
13 </style>
14 <!-- jQuery読み込み -->
15 <script src="jquery-3.4.1.js"></script>
16 <!-- jQueryを使用したスクリプトの記述 -->
17 <script>
18 $(function(){
19     //--- 「隠す」ボタン ---
20     $("#bHide").click(function(){
21         $("#jqr").hide();
22     });
23     //--- 「表示」ボタン ---
24     $("#bDisp").click(function(){
25         $("#jqr").show();
26     });
27     //--- 「フェードイン」ボタン ---
28     $("#bFadeIn").click(function(){

```

```

29     $("#jqqr").hide();
30     $("#jqqr").fadeIn(2000);
31 });
32 //--- 「フェードアウト」ボタン ---
33 $("#bFadeOut").click(function(){
34     $("#jqqr").show();
35     $("#jqqr").fadeOut(2000);
36 });
37 //--- 「スライドダウン」ボタン ---
38 $("#bSlideDown").click(function(){
39     $("#jqqr").hide();
40     $("#jqqr").slideDown(2000);
41 });
42 //--- 「スライドアップ」ボタン ---
43 $("#bSlideUp").click(function(){
44     $("#jqqr").show();
45     $("#jqqr").slideUp(2000);
46 });
47 });
48 </script>
49 </head>
50 <!-- 文書本体 -->
51 <body>
52 <input type="button" value="隠す" id="bHide">
53 <input type="button" value="表示" id="bDisp">
54 <input type="button" value="フェードイン" id="bFadeIn">
55 <input type="button" value="フェードアウト" id="bFadeOut">
56 <input type="button" value="スライドダウン" id="bSlideDown">
57 <input type="button" value="スライドアップ" id="bSlideUp">
58 
59 </body>
60 </html>

```

このサンプルで使用した視覚効果用のメソッドを表 25 に示します。

表 25: jquery04.html で使用している視覚効果用メソッド

メソッド	解説
show()	対象の要素を表示する。
hide()	対象の要素を隠す。
fadeIn(時間)	対象の要素を、与えた「時間」(ミリ秒)をかけてフェードインして表示する。
fadeOut(時間)	対象の要素を、与えた「時間」(ミリ秒)をかけてフェードアウトして隠す。
slideDown(時間)	対象の要素をスライドダウン(拡大しながら表示)する。
slideUp(時間)	対象の要素をスライドアップ(縮小しながら隠匿)する。 効果に要する時間をミリ秒で指定する。

11.1.5.2 jQuery UI

jQuery UI は jQuery の機能を用いて構築されたライブラリであり、jQuery ライブラリのみでは実現することが面倒な GUI を、簡単な方法で実現する手段を提供します。jQuery UI は jQuery ライブラリとは別に提供されており、公式インターネットサイト <https://jqueryui.com/> から入手することができます。

■ jQuery UI を使用するための準備

jQuery UI を公式インターネットサイトからダウンロードして展開すると、ライブラリー一式が格納されたフォルダが得られます。ライブラリのフォルダには jQuery UI 本体である jquery-ui.js (最小版は jquery-ui.min.js) をはじめ、使用する画像素材や CSS ファイルなどが収められています。jQuery UI の使用にあたってはこれら全てのファイルが必要となるので、jQuery UI を応用する HTML コンテンツからアクセスできるようにフォルダを配置する必要があります。

jQuery UI は jQuery ライブラリ本体を必要としますが、jQuery UI のフォルダの中にも jQuery 本体のファイル jquery.js があるので、それを利用する⁵¹ ことができます。本書の執筆時点では jQuery UI の版は 1.12.1 で、この版を目的のコンテンツに組み込むには、次のようにして CSS とライブラリを読み込みます。

【jQuery UI 関連ファイルの組み込み】

- 必要な CSS の読み込み

```
<link href="jquery-ui-1.12.1.custom/jquery-ui.css" rel="stylesheet">
```

- 同梱されている jQuery ライブラリの読み込み

```
<script type="text/javascript"
src="jquery-ui-1.12.1.custom/external/jquery/jquery.js"></script>
```

- jQuery UI の読み込み

```
<script type="text/javascript"
src="jquery-ui-1.12.1.custom/jquery-ui.js"></script>
```

これは、jQuery UI のディレクトリ「jquery-ui-1.12.1.custom」が目的の HTML コンテンツと同じディレクトリに配置されている場合の例です。

以下に jQuery UI が提供する UI のいくつかを紹介します。

■ タブの設置

図 118 のようなタブを実現する例を jqueryUI01.html に示します。

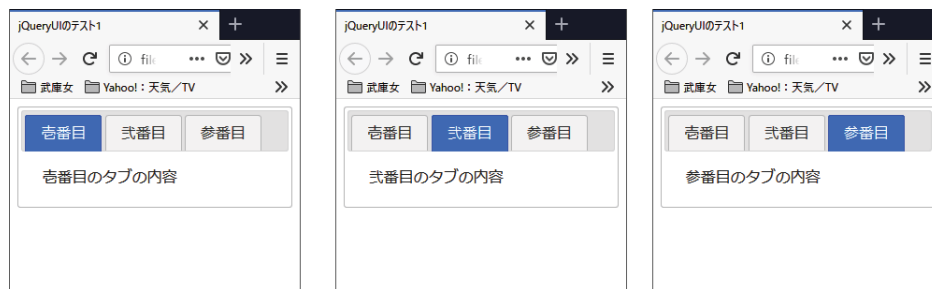


図 118: UI の例：タブ

サンプル：jqueryUI01.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>jQueryUIのテスト1</title>
5 <!-- jQuery UI用のCSS読み込み -->
6 <link href="jquery-ui-1.12.1.custom/jquery-ui.css" rel="stylesheet">
7 <!-- jQuery/jQueryUI読み込み -->
8 <script src="jquery-ui-1.12.1.custom/external/jquery/jquery.js"></script>
9 <script src="jquery-ui-1.12.1.custom/jquery-ui.js"></script>
10 <!-- jQueryを使用したスクリプトの記述 -->
11 <script>
12 $(function() {
13     $( "#tabs" ).tabs();
14 });
15 </script>
16 </head>
17 <!-- 文書本体 -->
18 <body>
```

⁵¹jQuery UI に同梱されている jQuery ライブラリ本体は必ずしも最新の版であるとは限りません。しかし、当該 jQuery UI の動作が保証されているものが同梱されているので、jQuery UI の利用にあたっては同梱されている版の jQuery を使用することが推奨されます。

```

19 <div id="tabs">      <!-- タブの全体 -->
20     <ul>              <!-- タブ（選択部分） -->
21         <li><a href="#tabs-1">壹 番 目</a></li>
22         <li><a href="#tabs-2">貳 番 目</a></li>
23         <li><a href="#tabs-3">参 番 目</a></li>
24     </ul>
25     <!-- タブの内容 -->
26     <div id="tabs-1">壹 番 目 の タブ の 内 容</div>
27     <div id="tabs-2">貳 番 目 の タブ の 内 容</div>
28     <div id="tabs-3">参 番 目 の タブ の 内 容</div>
29 </div>
30 </body>
31 </html>

```

この例のように、 要素でタブ（見出し部）を作り、それぞれに対応する内容を <div> 要素で記述し、id 属性にて関連させます。また、それらを含む <div> 要素に対して tabs メソッドを実行することでタブが完成します。

■ メニューの設置

図 119 のような階層メニューを実現する例を jqueryUI02.html に示します。

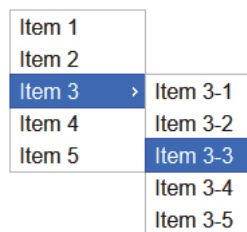


図 119: UI の例：階層メニュー

サンプル：jqueryUI02.html

```

1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>jQueryUIのテスト2</title>
5 <!-- jQuery UI用のCSS読み込み -->
6 <link href="jquery-ui-1.12.1.custom/jquery-ui.css" rel="stylesheet">
7 <!-- jQuery/jQueryUI読み込み -->
8 <script src="jquery-ui-1.12.1.custom/external/jquery/jquery.js"></script>
9 <script src="jquery-ui-1.12.1.custom/jquery-ui.js"></script>
10 <!-- jQueryを使用したスクリプトの記述 -->
11 <script>
12 $(function() {
13     $( "#menu" ).menu();
14 });
15 </script>
16 </head>
17 <!-- 文書本体 -->
18 <body>
19 <ul style="width:100px;" id="menu">
20     <li><div>Item 1</div></li>
21     <li><div>Item 2</div></li>
22     <li><div>Item 3</div>
23         <ul>
24             <li><div>Item 3-1</div></li>
25             <li><div>Item 3-2</div></li>
26             <li><div>Item 3-3</div></li>
27             <li><div>Item 3-4</div></li>
28             <li><div>Item 3-5</div></li>
29         </ul>
30     </li>
31     <li><div>Item 4</div></li>
32     <li><div>Item 5</div></li>

```

```

33 </ul>
34 </body>
35 </html>

```

この例のように、 要素を入れ子にして階層メニューの項目群を記述し、menu メソッドを実行することで階層メニューが完成します。

■ Datepicker (カレンダー) の設置

図 120 のようなカレンダーを実現する例を jqueryUI03.html に示します。



図 120: UI の例：Datepicker (カレンダー)

サンプル：jqueryUI03.html

```

1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>jQueryUIのテスト3</title>
5 <!-- jQuery UI用のCSS読み込み -->
6 <link href="jquery-ui-1.12.1.custom/jquery-ui.css" rel="stylesheet">
7 <!-- jQuery/jQueryUI読み込み -->
8 <script src="jquery-ui-1.12.1.custom/external/jquery/jquery.js"></script>
9 <script src="jquery-ui-1.12.1.custom/jquery-ui.js"></script>
10 <!-- jQueryを使用したスクリプトの記述 -->
11 <script>
12 $(function() {
13     $( "#datepicker" ).datepicker();
14 });
15 </script>
16 </head>
17 <!-- 文書本体 -->
18 <body>
19 今月のカレンダーは
20 <div id="datepicker"></div>
21 です。
22 </body>
23 </html>

```

この例のように、<div> 要素に datepicker メソッドを実行することでカレンダーが完成します。

11.1.6 その他の便利な機能

11.1.6.1 要素のインデックスの取得

指定した要素が HTML 文書の中の何番目に位置しているかのインデックスを取得するメソッド `index` があります。

書き方： 対象要素.`index()`

`index` メソッドを使用したサンプル `jqueryIndex01-1.html` を示して説明します。

サンプル：jqueryIndex01-1.html

```
1 <!DOCTYPE html>
2 <html lang="ja">
3 <head>
4 <meta charset="utf-8">
5 <title>jQueryのindex()</title>
6 <style>
7 #tx1 { border: solid 2px black; }
8 </style>
9 <script src="jquery-3.4.1.js"></script>
10 <script>
11 $(function(){
12     //--- イベントハンドリングの登録 ---
13     $("#lst1 li").on("click", function(e){
14         tx1.value = "インデックス：" + $(this).index();
15     });
16 });
17 </script>
18 </head>
19 <body>
20 簡条書き：<br>
21 <ul id="lst1">
22     <li>1 番目</li><li>2 番目</li><li>3 番目</li>
23 </ul>
24 <input type="text" value="" id="tx1">
25 </body>
26 </html>
```

簡条書き：

- 1 番目
- 2 番目
- 3 番目

インデックス：1

入れ子のdiv要素：

1 番目
2 番目
3 番目

インデックス：2

(a) 簡条書き項目のインデックス

(b) div要素のインデックス

図 121: クリックした要素のインデックスを表示

このサンプルは、クリックした簡条書き項目のインデックスをテキストフィールドに表示するものです。 `id="#lst1"` の要素に属する `` に、クリックされたときのイベントハンドリングを登録（13～15 行目）しています。14 行目の `$(this)` は、当該イベントが発生した要素（クリックされた要素）で、これに対して `index` メソッドを実行すると、それが何番目の要素であるかのインデックス値を返します。（インデックス値の開始は 0 です。）

このサンプルを Web ブラウザで実行した様子を図 121 の (a) に示します。

このようなイベントハンドリングは簡条書き項目以外の場合でも同様に実現できます。 `jqueryIndex01-1.html` と同様の方法で `<div>` 要素のインデックスを取得するサンプル `jqueryIndex01-2.html` を示します。

サンプル：jqueryIndex01-2.html

```
1 <!DOCTYPE html>
2 <html lang="ja">
3 <head>
```

```

4 <meta charset="utf-8">
5 <title>jQueryのindex()</title>
6 <style>
7 div {
8     border: solid 2px black;
9     margin: 2px;
10 }
11 #wrapper { width: 150px; }
12 #tx1 { border: solid 2px black; }
13 </style>
14 <script src="jquery-3.4.1.js"></script>
15 <script>
16 $(function(){
17     //--- イベントハンドリングの登録 ---
18     $("#wrapper div").on("click", function(e){
19         tx1.value = "インデックス：" + $(this).index();
20     });
21 });
22 </script>
23 </head>
24 <body>
25 入れ子のdiv要素：<br>
26 <div id="wrapper">
27     <div> 1 番目</div><div> 2 番目</div><div> 3 番目</div>
28 </div>
29 <input type="text" value="" id="tx1">
30 </body>
31 </html>

```

このサンプルを Web ブラウザで実行した様子を図 121 の (b) に示します。

11.1.6.2 文字列のトリミング

実用的な HTML5 アプリケーションの構築において、ユーザによって入力された文字列を扱う処理は高い頻度で発生します。その場合、入力された文字列の先頭や末尾に余分な空白文字が付いていることも多々あります。文字列の前後にある余分な空白文字を除去するために trim メソッドが提供されています。このメソッドの使用方法をサンプル jquery05.html で示します。

サンプル：jquery05.html

```

1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>jQueryテスト5</title>
5 <!-- jQuery読み込み -->
6 <script src="jquery-3.4.1.js"></script>
7 <!-- jQueryを使用したスクリプトの記述 -->
8 <script>
9 $(function(){
10     //--- 「隠す」ボタン ---
11     $("#tIn").change(function(){
12         var txi = $("#tIn").val();           // 入力フィールドから文字列取得
13         var txo = $.trim( txi );             // トリミング処理
14         $("#tOut").val( " "+txo+" " );      // 出力フィールドにセット
15     });
16 });
17 </script>
18 </head>
19 <!-- 文書本体 -->
20 <body>
21 入力：<input type="text" id="tIn">   →   出力：
22 <input type="text" id="tOut">
23 </body>
24 </html>

```

このサンプルでは図 122 のように、入力フィールドに入力された文字列をトリミング処理して出力フィールドに表示します。

入力 : → 出力 :

↓ 入力の後 を押す

入力 : → 出力 :

図 122: 実行例：「トリミング」

■ trim メソッドの使い方

`$.trim(トリミング対象の文字列)`

トリミング結果の文字列が戻り値として得られます。

このサンプル `jquery05.html` では、テキストフィールドの による入力確定は `change` イベントでハンドリングしています。

11.2 MathJax

MathJax ライブラリを使用することで、HTML コンテンツ内に記述した $\text{T}_\text{E}\text{X}$ の数式を整形表示することができます。MathJax に関する情報は公式インターネットサイト

<https://www.mathjax.org/>

から入手することができます。MathJax ライブラリは、公式サイトがオンラインで公開しているものを直接利用することもできますが、ライブラリ本体をローカル環境にダウンロードすることで、オフラインでも利用することができます。

ここでは、サンプルを示しながら MathJax の利用方法について説明します。

■ サンプル 1 (オンラインでの利用) : mathjax01-1.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>MathJaxによる数式の表示</title>
5 <script id="MathJax-script" async
6   src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-mml-autoload.js"></script>
7 <style>
8 p {
9   margin-top: 10pt;
10  margin-left: 10pt;
11  font-size: 12pt;
12 }
13 </style>
14 </head>
15
16 <body>
17 <p>\(\displaystyle \frac{d}{dx}\sin(x)=\cos(x)\)</p>
18 <p>\(\displaystyle \int_{-\infty}^{\infty} f(x) dx = 1\)</p>
19 </body>
20 </html>
```

サンプル中の 5 行目にあるように、公式サイトの JavaScript を利用するために

```
<script id="MathJax-script" async
  src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-mml-autoload.js"></script>
```

と記述⁵²することで MathJax が有効になります。これを Web ブラウザで表示すると図 123 のようになります。

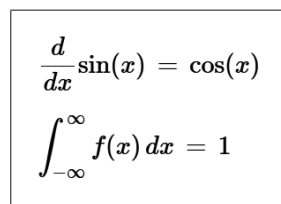

$$\frac{d}{dx}\sin(x) = \cos(x)$$
$$\int_{-\infty}^{\infty} f(x) dx = 1$$

図 123: Web ブラウザで表示した例

mathjax01-1.html に示した例は、公式サイトが公開するライブラリをオンラインで利用した例ですが、MathJax ライブラリ本体をローカル環境に配置することで、インターネット接続が無い状態（オフライン）でも同様の処理を行うことができます。

⁵²本書の執筆時点での MathJax の版は 3.0.1 です。

■ サンプル 2 (オフラインでの利用): mathjax01-2.html

```
1 <html>
2 <head>
3 <title>MathJaxによる数式の表示 </title>
4 <meta charset="utf-8">
5 <script id="MathJax-script" async
6   src="MathJax-master/es5/tex-mml-cthtml.js"></script>
7 <style>
8 p {
9   margin-top: 10pt;
10  margin-left: 10pt;
11  font-size: 12pt;
12 }
13 </style>
14 </head>
15
16 <body>
17 <p>\(\displaystyle \frac{d}{dx}\sin(x)\,,=\,,\cos(x)\)</p>
18 <p>\(\displaystyle \int^{\infty}_{-\infty}f(x)\,dx\,,=\,,1\)</p>
19 </body>
20 </html>
```

この例は、MathJax ライブラリを含むディレクトリ `MathJax-master` が当該コンテンツと同じディレクトリに配置されていることを前提としています。サンプル中の 5~6 行目にあるように

```
<script id="MathJax-script" async
  src="MathJax-master/es5/tex-mml-cthtml.js"></script>
```

と記述することで MathJax が有効になり、インターネット接続が無い状態でも数式が整形表示されます。

A 付録

A.1 フォントの活用

HTML5 では、フォントファイル（*.ttf, *.ttc, *.otf など）を指定してフォントのスタイルを指定することができます。これを応用すると、OS に付属するもの以外のフォントを使用したコンテンツのデザインが可能になります。また、近年はフリーソフトウェアとして公開されるフォントも多く、それらを利用することで作成する HTML5 アプリケーションの表現力を高めることができます。

ここでは、フォントファイルを指定したフォントスタイルの設定の方法について簡単に説明します。

A.1.1 フォントの入手について

コンピュータで使用する書体（フォント）の表示は、**フォントデータ**（フォントファイル）によって支えられています。OS にも多くのフォントが付属していますが、利用者が個別にフォントファイルを入手してシステムにインストールすることで利用できるフォントの種類を増やすことができます。

フォントデータには有償／無償様々なものがありますが、どれも共通の技術⁵³ に基いています。フォントデータは *.ttf, *.ttc, *.otf といった拡張子を伴うファイル名のデータです。インターネット上には有償のフォントを販売する商用サイトや、無償（フリー）のフォントを公開しているサイトが多数あり、必要なものを、条件に応じて入手して利用することができます。

フォント利用上の注意

フォントデータは著作物（知的財産）であり、その利用においては契約内容や注意事項をよく理解しておいてください。特に、

- ・ 商用利用の条件
- ・ 再配布の条件
- ・ 商標やロゴに使用する際の条件

には注意してください。

A.1.2 利用法 1：フォントファイルの組み込み

HTML コンテンツを Web ブラウザで表示すると、OS にインストールされているフォントの中からシステムが選んで使用しますが、HTML5 では特定のフォントファイルをコンテンツに組み込んで使用することができます。例えば KodomoRounded.otf というフォント⁵⁴ を使用して文字を表示する例を FontEmbed01.html に示します。

サンプル：FontEmbed01.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>フォントの埋め込み</title>
5 <style>
6 @font-face {
7     font-family: KodomoRounded;
8     src: url("./KodomoRounded.otf");
9 }
10
11 #tx1 {
12     font-family: KodomoRounded;
13 }
14
15 #ip1 {
16     width:          200pt;
17 }
```

⁵³ Apple が開発して後に標準化された TrueType が有名で、現在では多くのシステムがこれを採用しています。

⁵⁴ インターネットサイト「タイピングアート」(<http://typingart.net/>) が公開しているフォント。

```

18 p {
19     margin-top:    0pt;
20     margin-bottom: 0pt;
21     padding-top:   0pt;
22     padding-bottom: 0pt;
23 }
24 </style>
25 <script>
26 //--- フォントサイズの設定 ---
27 function fntsz( m ) {
28     var sz = m + "pt";
29     tx1.style.fontSize = sz;
30 }
31 //--- フォントウェイトの設定 ---
32 function fntwt( m ) {
33     tx1.style.fontWeight = m;
34 }
35
36 //--- フォントサイズの変更処理 ---
37 function f1( ) {
38     ip2.value = ip1.value;
39     fntsz( ip1.value );
40 }
41 function f2( ) {
42     var m = parseInt( ip2.value );
43     if ( 10 <= m && m <= 72 ) {
44         ip1.value = m;
45     } else {
46         m = 18;
47     }
48     fntsz( m );
49 }
50
51 //--- フォントウェイトの変更処理 ---
52 function f3( ) {
53     ip4.value = ip3.value;
54     fntwt( ip3.value );
55 }
56 function f4( ) {
57     var m = parseInt( ip4.value );
58     if ( 100 <= m && m <= 900 ) {
59         ip3.value = m;
60     } else {
61         m = 100;
62     }
63     fntwt( m );
64 }
65
66 //--- リセット ---
67 function f0( ) {
68     ip1.value = 18;
69     ip3.value = 400;
70     f1();
71     f3();
72 }
73 </script>
74 </head>
75 <body onLoad="f0( )">
76 font-size: <input type="range" min="10" max="72" value="18" onInput="f1( )" id="ip1
77 ">
78 <input type="text" size="10" id="ip2" value="18" onChange="f2( )">pt<br>
79 font-weight: <input type="range" min="100" max="900" value="400" step="100" onInput
80 ="f3( )" id="ip3">
81 <input type="text" size="10" value="400" id="ip4" onChange="f4( )">pt <input type
82 ="button" value="リセット" onClick="f0( )"><br>
83 <p id="tx1">ぼくは今日、学校で友だちとかけっこしました。<br>
84 楽しかったです。</p>
85 </body>
86 </html>

```

これを Web ブラウザで実行した様子を図 124 に示します。



図 124: Web ブラウザで実行した例

【解説】

このサンプルは、同じディレクトリにフォントファイル `KodomoRounded.otf` があることを前提としています。

● 使用するフォントファイルの指定：6～9 行目

CSS（スタイル）内に次のように記述します。

《フォントファイルとフォントファミリ名の対応》

```
@font-face {  
    font-family: スタイルとして使用するフォントファミリの名前;  
    src:          url("フォントファイル名");  
}
```

この記述により、スタイル要素 `font-family` としてフォントを指定することができます。

A.1.3 利用法2：Web で公開されているフォントの利用

HTML5 では、フォントをファイルとして入手して利用する方法以外に、Web で公開されているフォントをと利用する方法があります。ここでは、Google が <https://fonts.google.com/>（図 125）で公開しているフォントを利用する例 `FontSample01.html` を示します。

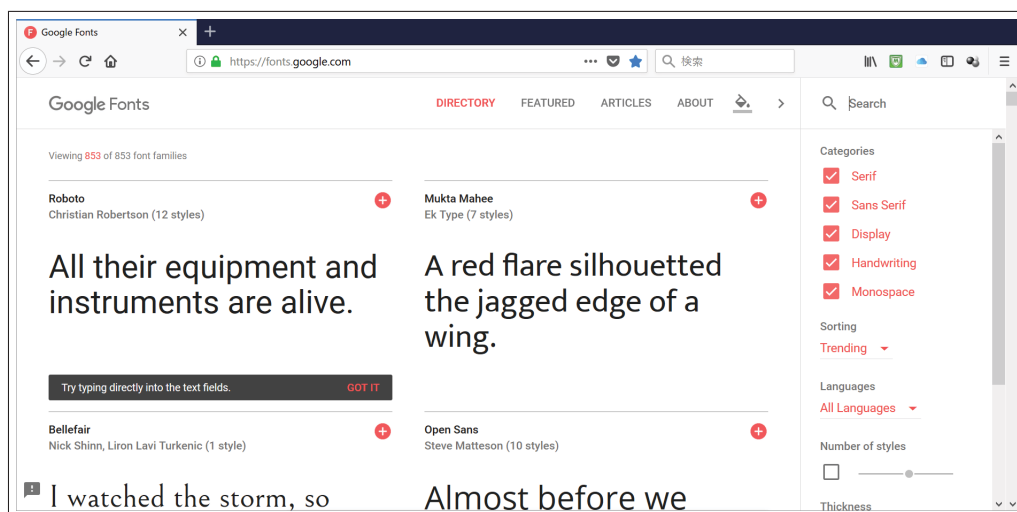


図 125: Google Fonts の Web サイト

サンプル：FontSample01.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>Googleフォントのサンプル </title>
5 <style>
6 @import url(https://fonts.googleapis.com/earlyaccess/notosansjp.css);
7 @import url('https://fonts.googleapis.com/css?family=Noto+Serif:400,700');
8 @import url(https://fonts.googleapis.com/earlyaccess/mplus1p.css);
9 @import url(https://fonts.googleapis.com/earlyaccess/roundedmplus1c.css);
10
11 #tx1 { font-family: 'Noto Sans JP'; }
12 #tx2 { font-family: 'Noto Serif', serif; }
13 #tx3 { font-family: 'Mplus 1p'; }
14 #tx4 { font-family: 'Rounded Mplus 1c'; }
15 #tx5 { font-family: sans-serif; }
16 #tx6 { font-family: serif; }
17 #tx7 { font-family: monospace; }
18
19 #ip1 {
20     width: 200pt;
21 }
22 p {
23     margin-top: 0pt;
24     margin-bottom: 0pt;
25     padding-top: 0pt;
26     padding-bottom: 0pt;
27 }
28 </style>
29 <script>
30 //--- フォントサイズの設定 ---
31 function fntsz( m ) {
32     var sz = m + "pt";
33     tx1.style.fontSize = sz;
34     tx2.style.fontSize = sz;
35     tx3.style.fontSize = sz;
36     tx4.style.fontSize = sz;
37     tx5.style.fontSize = sz;
38     tx6.style.fontSize = sz;
39     tx7.style.fontSize = sz;
40 }
41 //--- フォントウェイトの設定 ---
42 function fntwt( m ) {
43     tx1.style.fontWeight = m;
44     tx2.style.fontWeight = m;
45     tx3.style.fontWeight = m;
46     tx4.style.fontWeight = m;
47     tx5.style.fontWeight = m;
48     tx6.style.fontWeight = m;
49     tx7.style.fontWeight = m;
50 }
51
52 //--- フォントサイズの変更処理 ---
53 function f1( ) {
54     ip2.value = ip1.value;
55     fntsz( ip1.value );
56 }
57 function f2( ) {
58     var m = parseInt( ip2.value );
59     if ( 10 <= m && m <= 72 ) {
60         ip1.value = m;
61     } else {
62         m = 18;
63     }
64     fntsz( m );
65 }
66
67 //--- フォントウェイトの変更処理 ---
68 function f3( ) {
```

```

69     ip4.value = ip3.value;
70     fntwt( ip3.value );
71 }
72 function f4( ) {
73     var m = parseInt( ip4.value );
74     if ( 100 <= m && m <= 900 ) {
75         ip3.value = m;
76     } else {
77         m = 100;
78     }
79     fntwt( m );
80 }
81
82 //--- リセット ---
83 function f0( ) {
84     ip1.value = 18;
85     ip3.value = 400;
86     f1();
87     f3();
88 }
89 </script>
90 </head>
91 <body onLoad="f1()">
92 font-size: <input type="range" min="10" max="72" value="18" onInput="f1( )" id="ip1
93 ">
94 <input type="text" size="10" id="ip2" value="18" onChange="f2( )">pt<br>
95 font-weight: <input type="range" min="100" max="900" value="400" step="100" onInput
96 ="f3( )" id="ip3">
97 <input type="text" size="10" value="400" id="ip4" onChange="f4( )">pt <input type
98 ="button" value="リセット" onClick="f0( )"><br>
99 <p id="tx1">Noto Sans (角ゴシック) </p>
100 <p id="tx2">Noto Serif (明朝) </p>
101 <p id="tx3">Mplus 1p (ゴシック系フォント) </p>
102 <p id="tx4">Rounded Mplus 1c (丸ゴシック) </p>
103 <hr>
104 <p id="tx5">sans-serif (ゴシック) </p>
105 <p id="tx6">serif (明朝) </p>
106 <p id="tx7">monospace (等幅ゴシック) </p>
107 </body>
108 </html>

```

これを Web ブラウザで実行した様子を図 126 に示します。

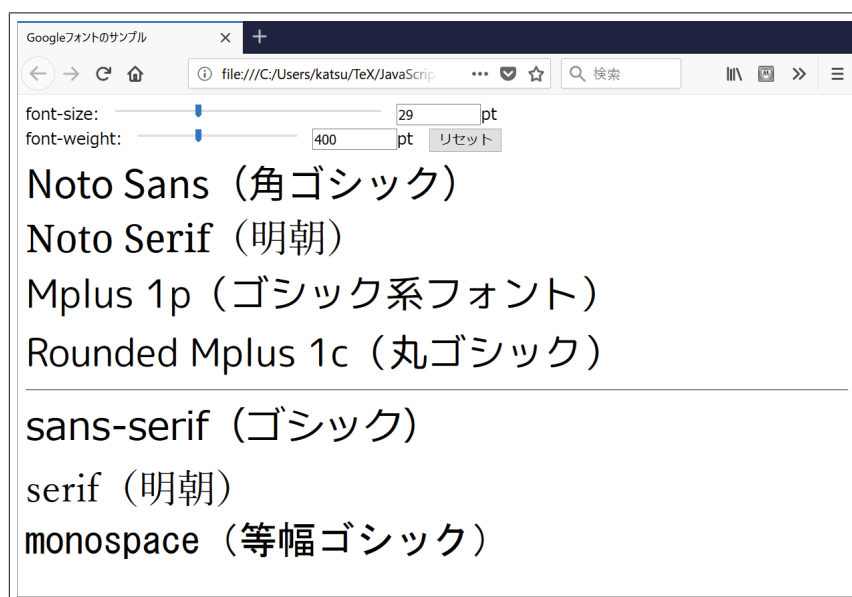


図 126: Web ブラウザで実行した例

ここで利用した Google のフォントは CSS として公開されており，6～9 行目にあるように，

```
@import url(公開されている CSS の URL);
```

としてコンテンツ内に読み込みます．読み込んだフォントがどのようなフォントファミリーとして使用できるかは，そのフォントを公開しているサイトの情報を参照してください．

A.2 ダウンロード機能の実装例

HTML の a 要素の属性 download にファイル名を設定して click メソッドを実行すると、そのファイルに href 属性の URL のコンテンツがダウンロードされます。これを応用すると、Web ブラウザのコンテンツの一部（要素の値など）をローカルのファイルに保存することが可能になります。

JavaScript の File API とダウンロードの機能を応用して実装した簡単なテキストエディタを TxEdit01.html に示します。

サンプル：TxEdit01.html

```
1 <html>
2 <head>
3 <meta charset="utf-8">
4 <title>テキストエディタ</title>
5 <style>
6 #ta {
7     font-family:    monospace;
8     font-size:      11pt;
9 }
10 </style>
11 <script>
12 var fr;
13
14 // ファイルの読み込み処理
15 function f1( e ) {
16     var flst = e.target.files;
17     fr.readAsText( flst[0] );
18 }
19
20 function getContents( e ) {
21     ta.value = fr.result;
22 }
23
24 // テキストエリアの内容をダウンロード
25 function f2() {
26     var b = new Blob( [ta.value], {type:"text/plain"} );
27     dwn.download = "download.file";
28     dwn.target = "_blank";
29     dwn.href = window.URL.createObjectURL( b );
30 // Google Chrome の場合
31 // dwn.href = window.webkitURL.createObjectURL( b );
32     dwn.click();
33 }
34
35 // 起動時の初期化処理
36 function ini() {
37     fr = new FileReader();
38     fr.onload = getContents;
39     dwn.style.visibility = "hidden";
40 }
41 </script>
42 </head>
43 <body onLoad="ini()">
44 <input type="file" onChange="f1(arguments[0])">
45 <input type="button" value="保存" onClick="f2()">
46 <a id="dwn">このリンクをダウンロード</a><br>
47 <textarea cols="80" rows="25" id="ta">
48 </textarea>
49 </body>
50 </html>
```

これを Web ブラウザで実行した様子を図 127 に示します。

【解説】

図 127 の実行画面において、参照... をクリックすると、ファイルを選択するダイアログが表示され、読み込むファ

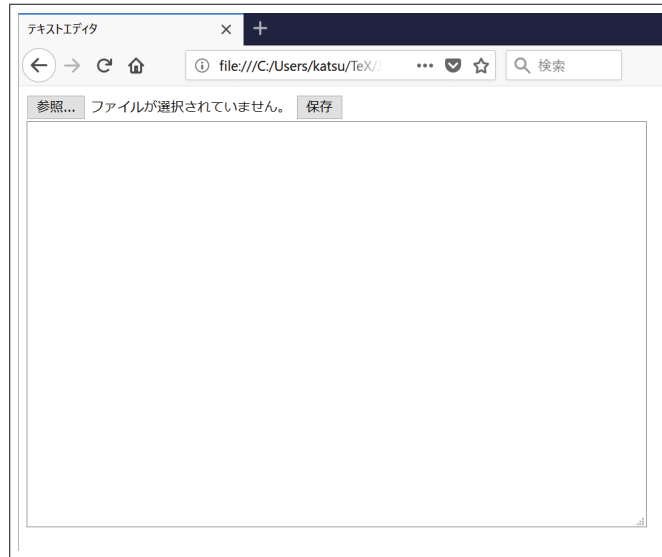


図 127: Web ブラウザで実行した例

イルを選ぶことができます。テキストの内容は `textarea` 要素の中で編集することができます。ファイルの保存は 保存 ボタンのクリックによって実行しますが、これはダウンロードの処理で実現しています。

このサンプルで特に重要なのが、`Blob` オブジェクトを用いたデータの作成です。`Blob` オブジェクトは、配列などのデータ構造を1つの「塊り」として扱うためのオブジェクトであり、プログラムの26行目では、`textarea` 要素の値であるテキストデータを `Blob` オブジェクト `b` として作成しています。それを29行目ではリンク先 URL に変換して `a` 要素の `href` 属性に与えています。更に32行目では `a` 要素に対して `click` メソッドを実行してダウンロード処理をしています。

26行目の

```
new Blob( [ta.value], type:"text/plain" )
```

にある通り、`Blob` オブジェクト生成時のコンストラクタの第1引数に連結対象の配列オブジェクトを、第2引数にコンテンツタイプを表すオブジェクトを与えます。生成した `Blob` オブジェクトを `createObjectURL` メソッドによって URL に変換してそれをダウンロード対象のリンク先としています。

A.3 ブラウザ付属の開発機能

JavaScript は Web ブラウザ上で実行するので、Web ブラウザが基本的な開発環境となります。ここでは、代表的な Web ブラウザ（Google Chrome, Mozilla Firefox）が持つ開発環境としての機能を紹介します。

A.3.1 ソースプログラムの自動整形

代表的な Web ブラウザには JavaScript のソースプログラムの体裁を整形する機能が備わっています。ソースプログラムの整形機能は、プログラムを見やすく表示するだけでなく、プログラムの文法上の誤りを見つけ出すのにも役立ちます。

先の例で紹介した FreeFall.html と基本的には同じプログラム FreeFall2.html を例に上げて、ソースプログラムの整形や、プログラムのデバッグといった機能を紹介します。

サンプル：FreeFall2.html

```
1 <!DOCTYPE html><html><head>
2 <meta charset="utf-8"><title>自由落下シミュレーション</title>
3 <script src="./FreeFall2.js"></script><body>
4 <form name="fm">
5 反射係数 (r) : <input type="text" size="4" name="r" value="0.95"><1.0
6 <br>重力加速度 (g) : <input type="text" size="4" name="g" value="100.0">
7 m/s2<br>質量=1kg <input type="button" value="用意" onClick="f01( )">
8 <input type="button" value="落下" onClick="f02( )">
9 <input type="button" value="停止" onClick="f03( )"><br></form>
10 </body></html>
```

サンプル：FreeFall2.js

```
1 var dt, t, t2, y, v, v2, g; var w, h; var snd = new Audio("clap01.wav");
2 var timer; var f = true; var ball;function f01( ) {
3 ball = document.getElementById("hball");w = parseInt(window.innerWidth);
4 h = parseInt(window.innerHeight) - 30;g = parseFloat(document.fm.g.value);
5 v = 0.0;y = 0.0;r = parseFloat(document.fm.r.value);t = 0.0;t2 = t;
6 dt = 0.05;ball.style.position = "absolute";ball.style.top = y + "px";
7 ball.style.left = (w / 2.0 - 15.0) + "px";ball.style.visibility="visible";
8 f = true;v2 = 0;snd.load();snd.play();}function f02( ) {if ( y > h ) {
9 snd.pause();snd.currentTime = 0;snd.play();if ( v2 == 0 ) {v2 = -1.0 * v;
10 }v2 *= r;v = v2;y = h;if ( (t - t2) < dt ) {f = false;}t2 = t;} else
11 {v += g;y += v * 0.01;t += 0.01;}ball.style.top = y + "px";
12 ball.style.visibility="visible";if ( f ) {timer = setTimeout("f02( )",10);
13 }} function f03( ) {clearTimeout(timer);}
```

FreeFall2.html が FreeFall.html と違うのは、JavaScript のソースプログラムが別のファイル FreeFall2.js として分離されていることです。Web ブラウザで整形できるソースプログラムはこのように別のファイルとして作成されている必要があります。

以下では Mozilla Firefox と Google Chrome それぞれの場合でソースプログラムを整形する方法を紹介します。プログラム開発に関する Web ブラウザの機能は 4.1.2 で紹介した方法で開始します。

A.3.1.1 Firefox の場合

「ツール」→「ウェブ開発」→「開発ツールを表示」とメニュー選択して、開発ツールを表示します。次に図 128 のように、「デバッガ」タブ内の「ソースファイル」タブから JavaScript のファイルを選択して、そのソースを表示させます。次に JavaScript のプログラムが表示されているペインの下部にあるソースコード整形ボタン「{ }」をクリックすると、ソースコードのペイン内に整形されたソースプログラムのタブが現れます。(図 129 参照)

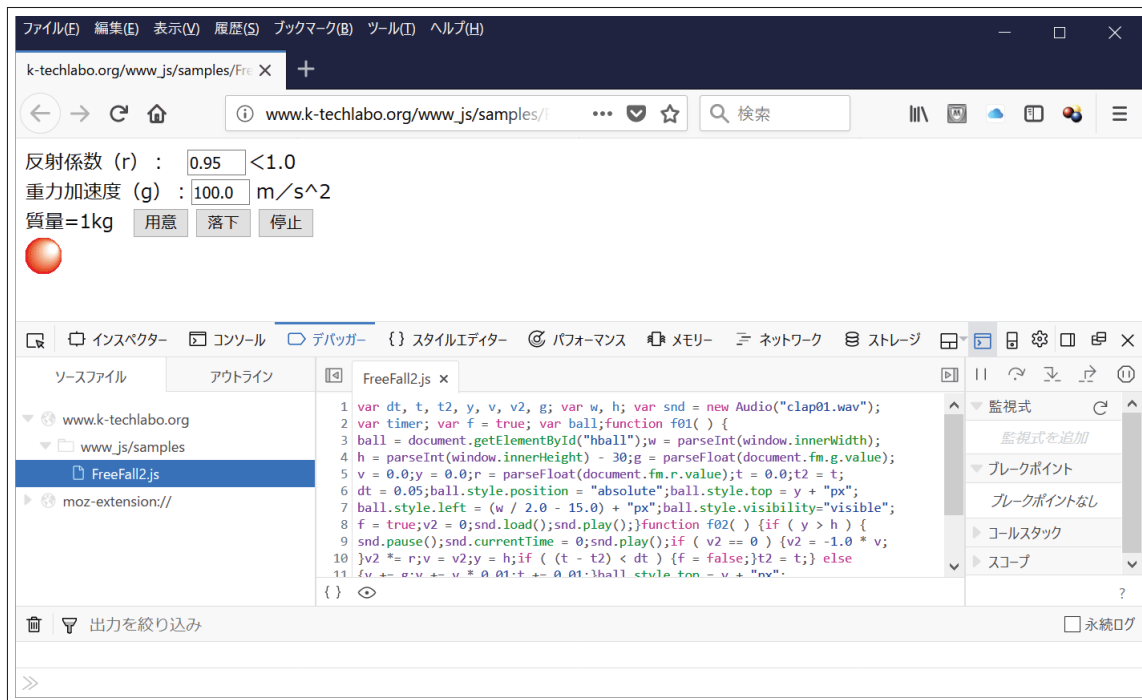


図 128: 整形前

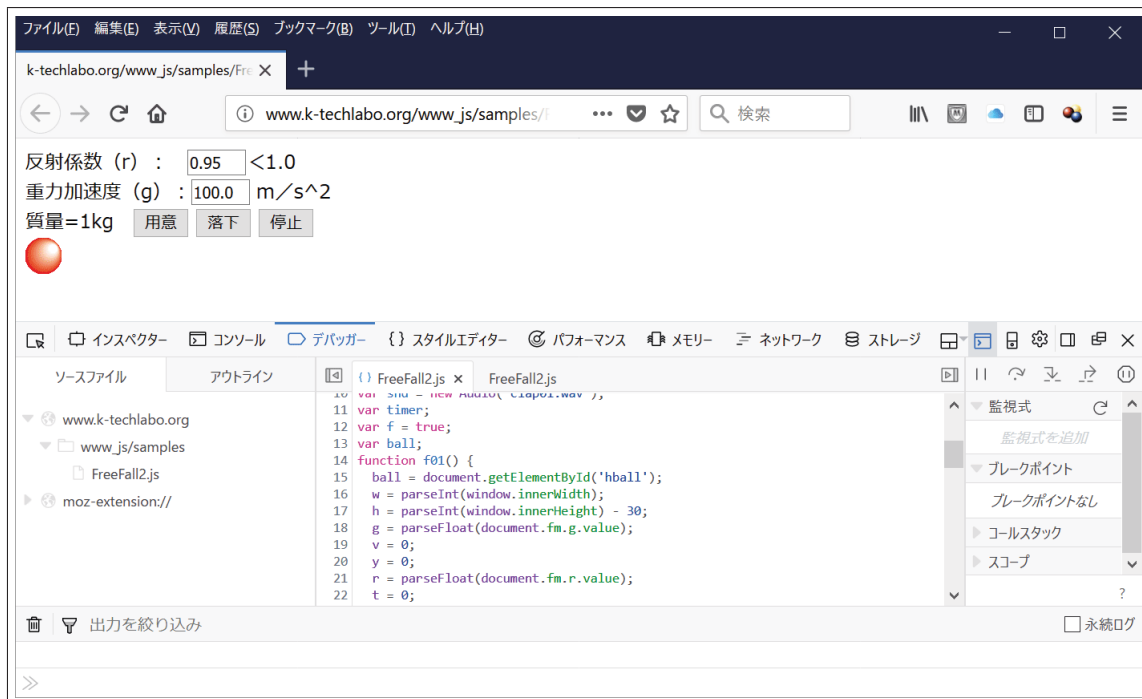


図 129: 整形後

A.3.1.2 Google Chrome の場合

Google Chrome で JavaScript のソースプログラムを整形表示する場合は、設定ボタン「⋮」から「その他のツール」→「デベロッパーツール」を選択してデベロッパーツール表示します。次にデベロッパーツールの「Sources」タブ内の「Network」タブからソースファイルを選択して表示（図 130）します。次に、ソースプログラムが表示されているペインの下部にあるソースコード整形ボタン「{ }」をクリックすると、ソースコードのペイン内に整形されたソースプログラムのタブが現れます。（図 131 参照）

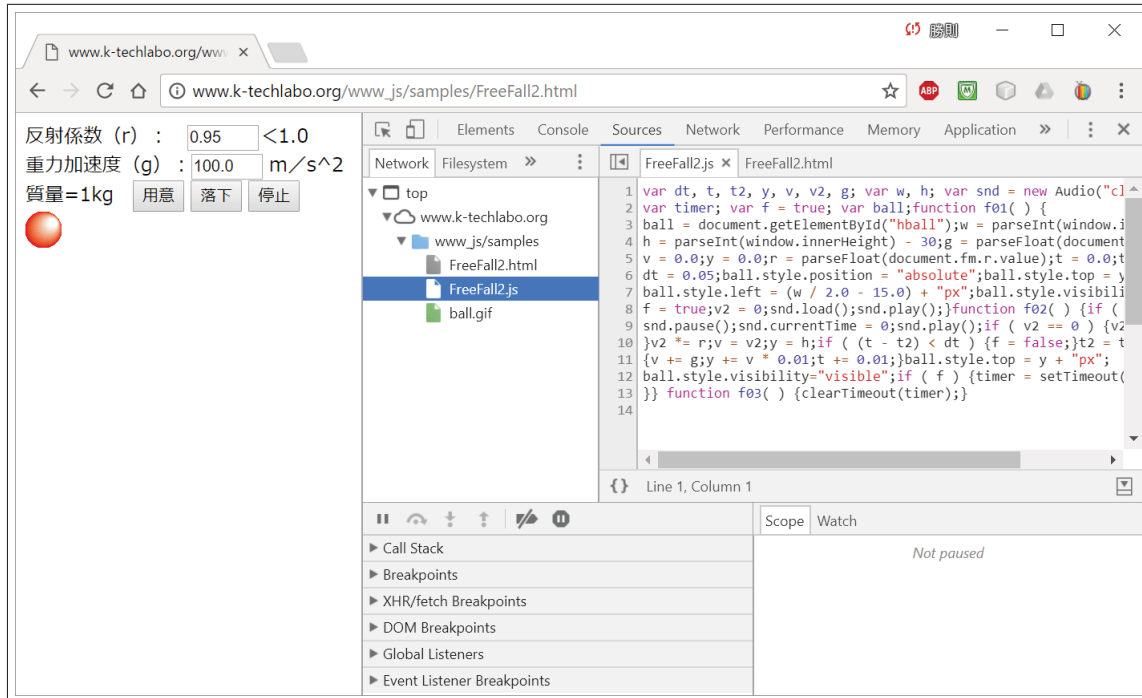


図 130: デベロッパーツールでソースを表示したところ（整形前）

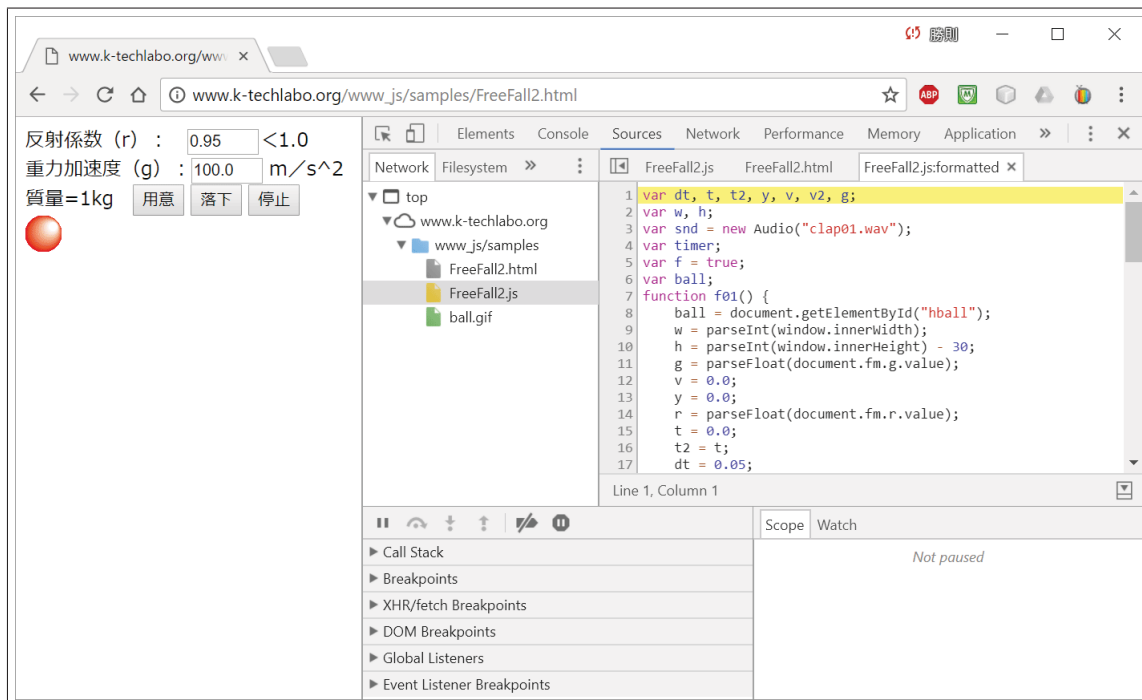


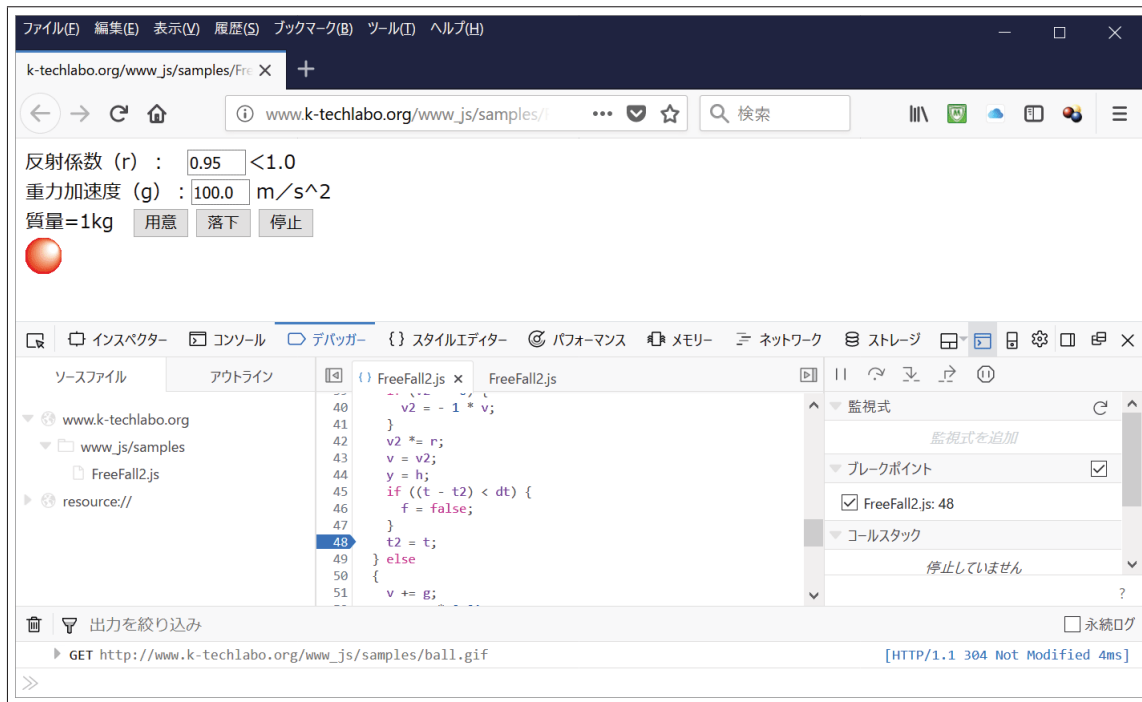
図 131: JavaScript のソースの表示（整形後）

A.3.2 プログラムのステップ実行

Mozilla Firefox と Google Chrome ブラウザでは、JavaScript のプログラムの指定したポイント（ブレークポイント）でプログラムの実行を一時停止して、その時点での変数の値などを調べることができます。

A.3.2.1 Firefox の場合

Mozilla Firefox の開発ツールでデバッガのタブを表示し、JavaScript のソースプログラムの行数が表示されている部分の左部分をクリックすると、ブレークポイントのマークが付きます。（図 132 参照）



↓
ブレークポイントを拡大
↓

```
40      v2 = - 1 * v;  
47    }  
48    t2 = t;  
49  } else
```

図 132: ブレークポイントの設定

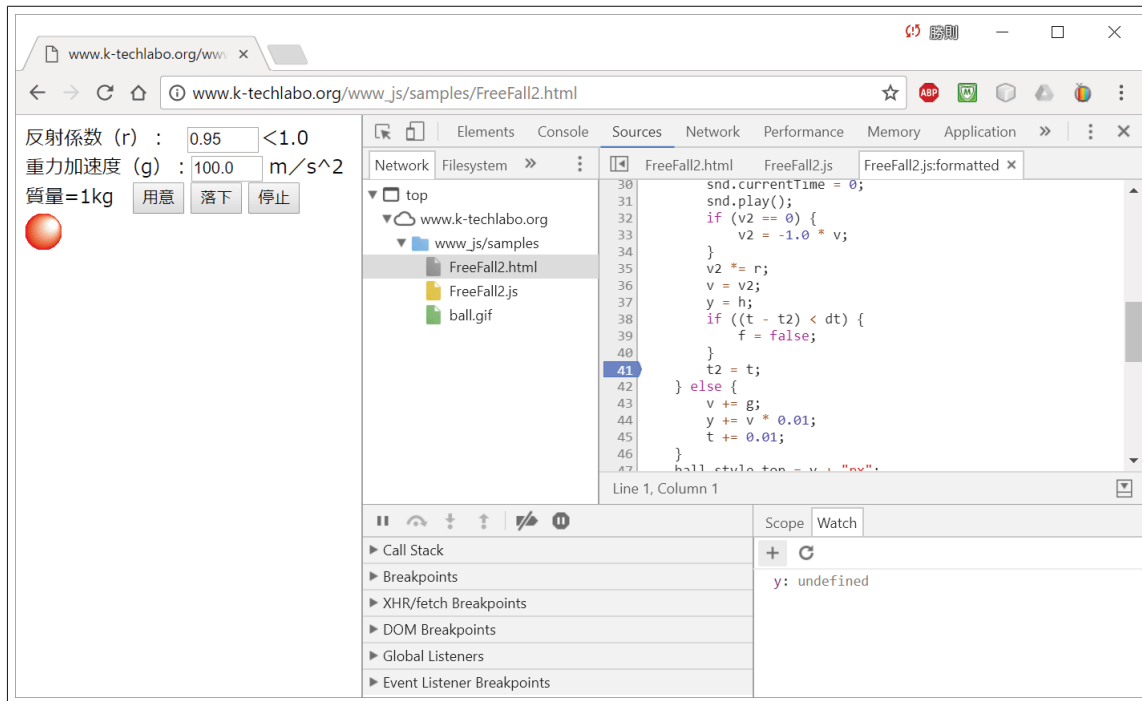
この状態で「監視式」のペインの中に変数名を入力しておくと、ブレークポイントでプログラムの実行が一時停止され、変数の値が表示されます。

プログラムにブレークポイントを設定すると、そのポイントで実行が一時停止状態になりますが、「▶」のボタンをクリックする度に実行が再開します。

A.3.2.2 Google Chrome の場合

Google Chrome で JavaScript プログラムをステップ実行するには、デバッガーツール内に表示されているソースプログラムの行番号の左部分をクリックしてブレークポイントを設定します。（図 133 参照）

プログラムを実行するとブレークポイントの位置で実行が一時停止します。また「Watch」ペインに、調査対象の変数の名前を入力しておくと、ブレークポイントで一時停止した状態での変数の値を表示することができます。（図 134 参照）



↓
 ブレークポイントを拡大
 ↓

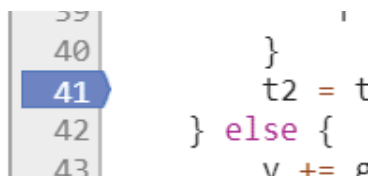


図 133: ブレークポイントの設定

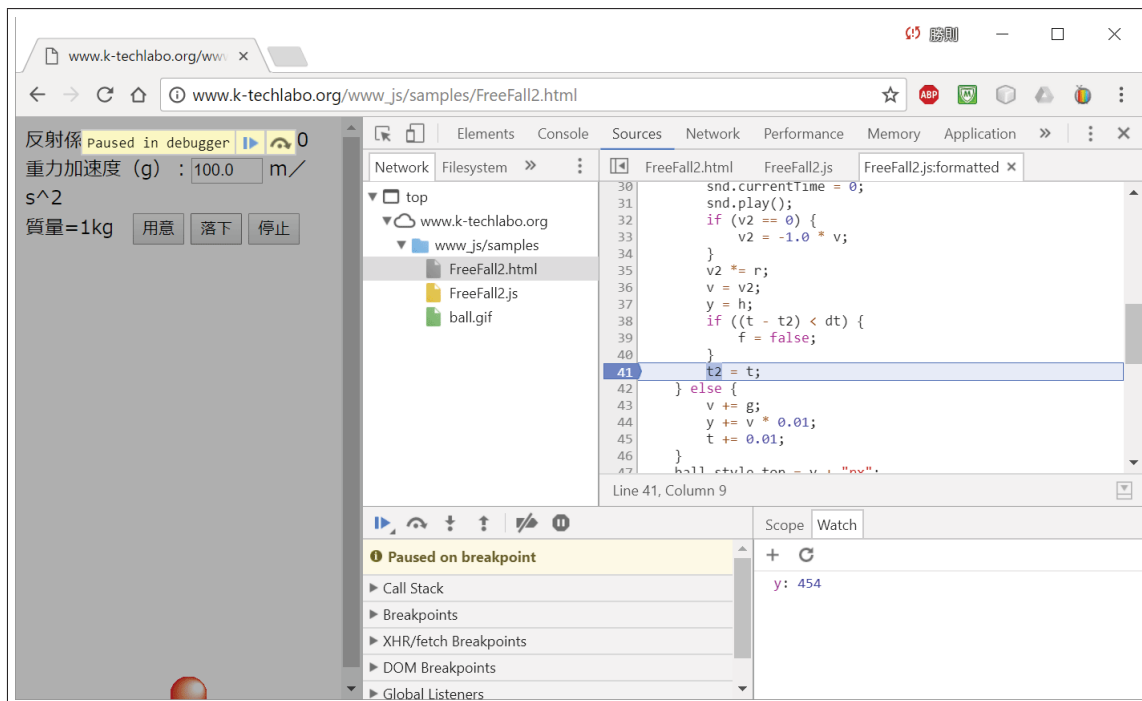


図 134: ブレークポイントで一時停止した状態

一時停止したプログラムの実行を再開するには図 135 のボタンをクリックすると実行が再開します。

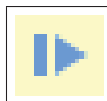


図 135: ステップ実行ボタン

A.3.3 プログラムのデバッグ

Mozilla Firefox と Google Chrome ブラウザでは、JavaScript を含む HTML5 コンテンツのデバッグのための機能が使えます。デバッグ作業の主なものは、

- HTML5 コンテンツ中にある文法上の誤りを探す
- JavaScript の `console.log()` 文で気になるオブジェクトの値を表示する

の 2 つで、Mozilla Firefox の場合は開発ツールで、Google Chrome の場合はデベロッパーツールでデバッグ作業を行います。これら 2 種類の Web ブラウザとも、デバッグ作業の流れはとても似ており、「コンソール」タブ（Firefox の場合）や「Console」タブ（Chrome の場合）で作業します。プログラム中の文法上の誤りはこれらのタブ内に表示され、プログラマはそれを手がかりにして誤りを訂正します。

プログラムの誤りは文法的なものばかりではなく、**論理上の誤り**（設計上の誤り）もありますが、それらの多くは検出が難しいです。そのような類の誤りを検出するには、気になる変数やオブジェクトの値を必要に応じて調べる方法が有効です。上記 2 種類のブラウザでは、JavaScript の `console.log()` 文を使うことができ、変数やオブジェクトの値をコンソールタブに表示することができます。

ここでは Google Chrome によるデバッグ作業の例を示します。

A.3.3.1 Google Chrome によるデバッグ作業の例

DebugSample01_1.html は、文法的な誤りを含むプログラムです。これを訂正して行く過程を例示します。

サンプル：DebugSample01_1.html （7 行目に文法誤りあり）

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>デバッグ用サンプル</title>
6 <script type="text/javascript">
7 function f1() {
8     t1.value = "ボタンがクリックされました。";
9 }
10 </script>
11 </head>
12 <body>
13 <input type="button" value="表示" class="btn" onClick="f1()">
14 <input type="button" value="消去" class="btn" onClick="f2()"><br>
15 <input type="text" size="30" id="t1">
16 </body>
17 </html>
```

これを Google Chrome で表示し、デベロッパーツールの Console タブを見ると図 136 のように表示されます。

Console タブのメッセージから、「function」なる語が誤りであることがわかります。これを DebugSample01_2.html のように修正します。

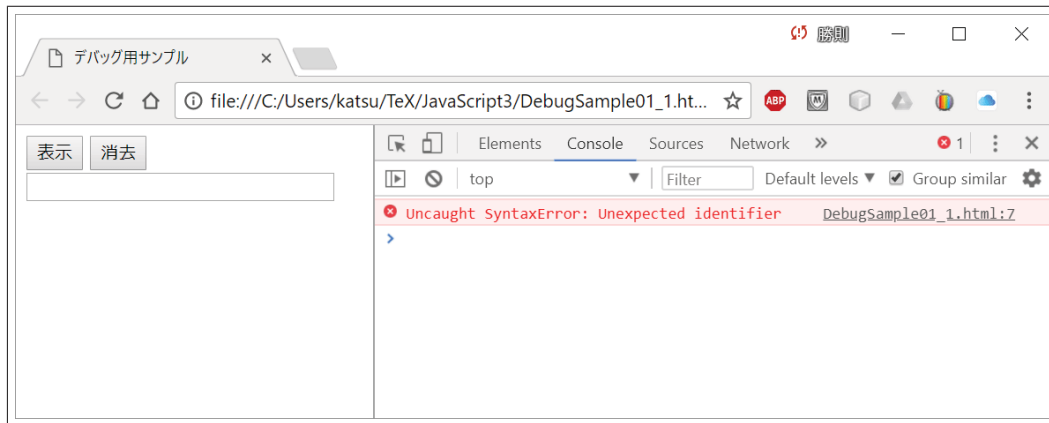


図 136: 誤りのある行番号が Console タブに表示されている。

サンプル：DebugSample01_2.html （文法誤りを修正：まだ怪しい…）

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>デバッグ用サンプル</title>
6 <script type="text/javascript">
7 function f1() {
8     t1.value = "ボタンがクリックされました。";
9 }
10 </script>
11 </head>
12 <body>
13 <input type="button" value="表示" class="btn" onClick="f1()">
14 <input type="button" value="消去" class="btn" onClick="f2()"><br>
15 <input type="text" size="30" id="t1">
16 </body>
17 </html>

```

しかし、「消去」ボタンのクリックをハンドリングする関数 `f2` が無いので、Web ブラウザで実行して「消去」ボタンをクリックすると図 137 のようなメッセージが Console タブに表示されます。

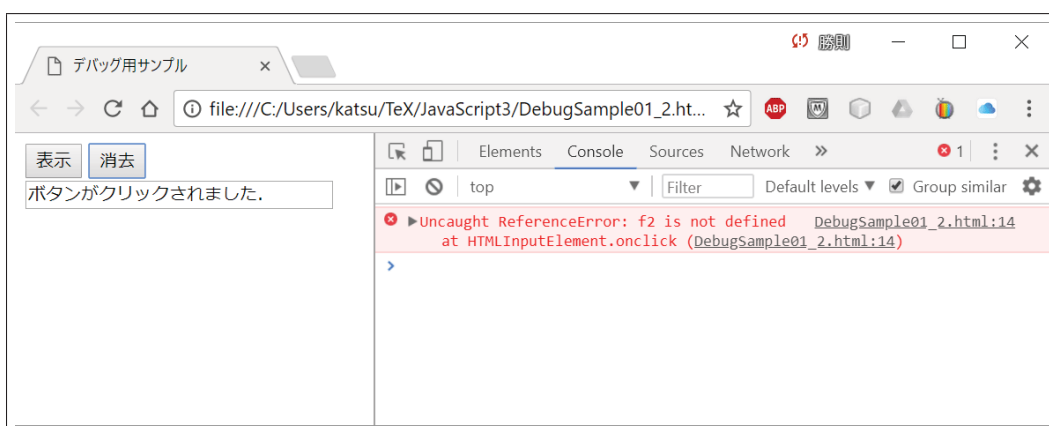


図 137: 「消去」ボタンをクリックしても、対応する関数 `f2` が無い…

関数 `f2` の定義を記述して、更に関数実行の開始と終了の時点で `console.log` 文を実行して、Console タブにメッセージを表示する形にしたものを DebugSample01.html に示します。

サンプル：DebugSample01.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="utf-8">
5 <title>デバッグ用サンプル</title>
6 <script type="text/javascript">
7   function f1() {
8     console.log("f1開始");
9     t1.value = "ボタンがクリックされました.";
10    console.log("f1終了");
11  }
12  function f2() {
13    console.log("f2開始");
14    t1.value = "";
15    console.log("f2終了");
16  }
17 </script>
18 </head>
19 <body>
20 <input type="button" value="表示" class="btn" onClick="f1()">
21 <input type="button" value="消去" class="btn" onClick="f2()"><br>
22 <input type="text" size="30" id="t1">
23 </body>
24 </html>
```

この形でボタンをクリックすると、ハンドリングする関数の開始と終了の時点で Console タブにメッセージを表示 (図 138) します。

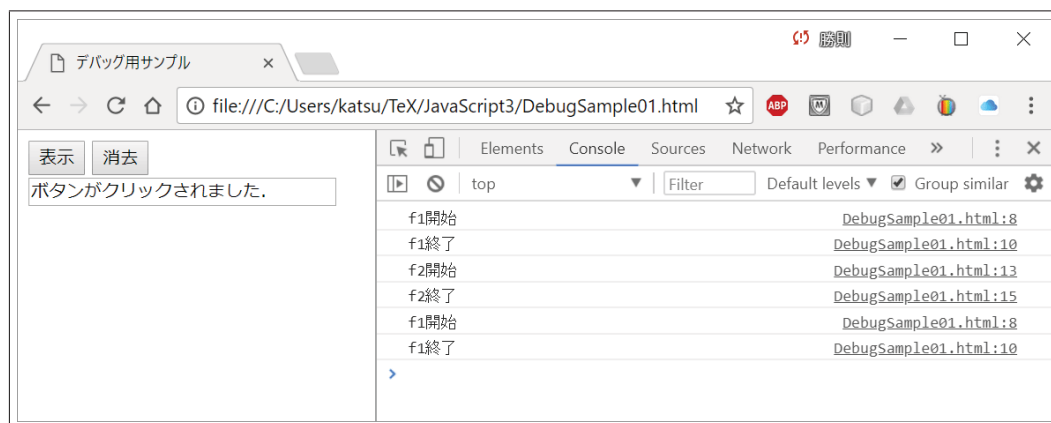


図 138: console.log 文で Console タブにメッセージを表示している

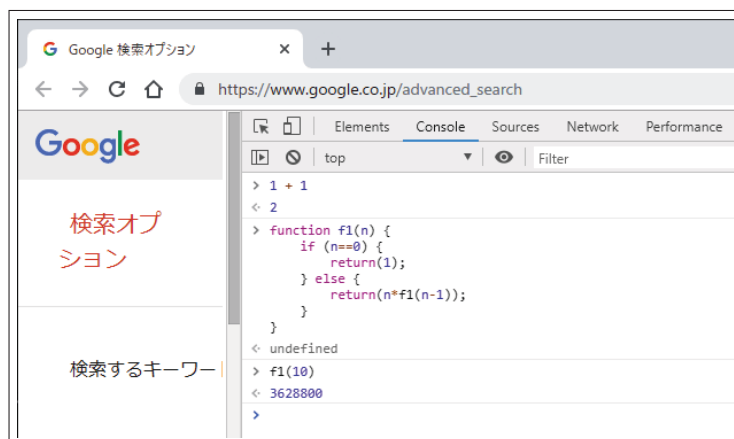
console.log の引数には、表示したい変数やオブジェクトを与えることができ、実行中の状態を調べるのに役立ちます。

A.3.4 JavaScript の対話的な実行

Firefox と Google Chrome では、ブラウザのコンソール機能を用いて JavaScript の文や式を対話的に実行することができます。



Firefox のコンソール



Google Chrome のコンソール

図 139: コンソールから JavaScript の文や式を直接実行する例

Firefox では「開発ツール」の「コンソール」タブで JavaScript の文や式を対話的に実行することができます。コンソールの表示を一掃するにはゴミ箱アイコンをクリックします。Google Chrome では「デベロッパーツール」の「Console」タブで JavaScript の文や式を対話的に実行することができます。

索引

, 30
**, 29
, , 9, 148
-bottom, 15
-left, 15
-right, 15
-top, 15
., 50
..., 57
:, 33
:nth-child, 154
<, 30
<=, 30
==, 30
===, 30
=>, 39
>, 30
>=, 30
?, 33
", 30
#, 9
\$, 147
\${ 式 }, 31
%, 10, 29
. , 9
-, 58
@font-face, 170
@import, 173
vertical-align, 22
`, 30
16 進数, 12
3 項演算子, 33

a:active, 12
a:hover, 12
a:link, 12
a:visited, 12
abort, 85
abs, 29
absolute, 14
acos, 29
add, 53
addEventListener, 87
alert, 23

append, 151
appendChild, 123
appendTo, 152
appName, 105
appVersion, 105
arc, 72
arguments, 39, 88
Array, 44
asin, 29
atan, 29
Audio, 95
audio, 94
availHeight, 104
availWidth, 104

background-color, 14
beginPath, 71
bevel, 131
BigInt, 30
bigint, 30
Blob, 175
blur, 85
body タグ, 3
bold, 74
boolean, 27
border, 13, 14, 20
border-collapse, 20
border-color, 14
border-style, 14
border-width, 14
box-sizing, 15
br, 4
break, 34, 35
butt, 132
button, 4, 64

canvas, 71
caption タグ, 20
Cascading Style Sheets, 9
cellspacing, 20
center, 17
CGI, 4
change, 85, 112
charset, 3

- checkbox, 4
- checked, 65
- childNodes, 123
- children, 69
- circle, 136
- class, 140
- class 属性, 9
- clear, 53, 55, 99
- clearTimeout, 97
- click, 84, 85, 149, 174
- clientHeight, 89, 104
- clientWidth, 89, 104
- clientX, 89
- clientY, 89
- closePath, 71
- cm, 10
- color, 11
- concat, 45
- console.log, 181
- const, 27, 42
- constructor, 140
- contextmenu, 85
- continue, 35
- controls, 95
- cos, 29
- createElement, 122
- createElementNS, 128
- createImageData, 80
- createObjectURL, 175
- createTextNode, 122
- CSS, 9
- css, 148
- currentTime, 96

- dashed, 14
- data, 79
- dataTransfer, 116
- Date, 59
- DatePicker, 162
- dblclick, 85
- delete, 27, 53, 55
- display, 18
- div, 12, 13, 18
- do...while, 34
- DOCTYPE, 3
- document, 68, 104, 123
- documentElement, 104

- DOM, 68, 122
- double, 14
- download, 174
- drag, 115
- draggable, 116
- dragover, 116, 117
- dragstart, 116
- drawImage, 76
- drop, 116, 117
- dropEffect, 117
- duration, 96

- E, 29
- Electron ベースのエディタ, 2
- ellipse, 136
- em, 10
- entries, 52
- eq, 156
- error, 85, 115
- euc-jp, 3
- every, 49
- exp, 29
- extends, 140

- fadeIn, 159
- fadeOut, 159
- false, 27
- FIFO, 45
- file, 112
- FileReader, 112
- files, 119
- fill, 72, 131
- fillRect, 74
- fillStyle, 72
- fillText, 74
- FIFO, 45
- filter, 46
- find, 46
- findIndex, 47
- fixed, 14
- float, 14
- focus, 85
- font, 74
- font-family, 10, 170
- font-weight, 10
- for, 34
- forEach, 47

- form, 4, 62
- form タグ, 4
- from, 53, 57
- function, 23, 35

- get, 54
- getAttribute, 64, 69
- getBoundingClientRect, 109
- getContext, 71
- getDate, 59
- getDay, 59
- getElementById, 71, 94, 107, 124
- getElementsByClassName, 124
- getElementsByName, 124
- getElementsByTagName, 124
- getFullYear, 59
- getHours, 59
- getItem, 99
- getMilliseconds, 59
- getMinutes, 59
- getMonth, 59
- getSeconds, 59
- getTime, 59
- Google Chrome, 1
- GUI, 4, 12

- has, 53, 54
- head タグ, 3
- height, 14, 22, 104
- hidden, 14
- hide, 159
- hoisting, 41
- hostname, 105
- href, 174
- html, 153
- HTMLCollection, 69
- HTML の階層構造, 123
- html タグ, 3

- id 属性, 9
- if, 33
- iframe, 120
- Image, 22, 67, 76
- ImageData, 79
- img タグ, 22
- in, 49, 51
- index, 163

- Infinity, 29
- innerHeight, 104
- innerHTML, 70
- innerText, 69
- innerWidth, 104
- input, 4, 85
- isArray, 46
- isInteger, 30
- iso-2022-jp, 3
- italic, 10, 74

- join, 32, 45
- jQuery, 147
- jQuery UI, 159
- JSON, 44

- key, 100
- keydown, 85
- keypress, 85
- keys, 50, 55
- keyup, 85

- lambda, 38
- lang, 3
- language, 105
- layerX, 87, 89
- layerY, 87, 89
- left, 14, 17, 107
- length, 31, 44, 99
- let, 27, 42
- letter-spacing, 11
- line, 136
- line-height, 11, 17, 18
- lineTo, 71
- lineWidth, 72
- link, 19
- load, 84, 87, 96
- localStorage, 99
- location, 105
- log, 29
- log10, 29
- log2, 29

- map, 47
- MapIterator, 55
- Map オブジェクト, 54
- margin, 14
- match, 145

Math.PI, 72
 max, 6
 MAX_SAFE_INTEGER, 29
 MAX_VALUE, 29
 meter, 7
 MIME タイプ, 116
 min, 6
 MIN_SAFE_INTEGER, 29
 MIN_VALUE, 29
 miter, 131
 miter 長, 132
 mm, 10
 monospace, 10
 mousedown, 85
 mousemove, 85
 mouseout, 85
 mouseover, 85
 mouseup, 85
 moveTo, 71
 Mozilla Firefox, 1
 muted, 96

 name, 63, 119
 NaN, 29
 navigator, 105
 new, 50, 59
 none, 12, 130
 normal, 10
 nth-child, 154
 null, 27
 number, 27

 Object, 44
 object-fit, 22
 oblique, 10
 of, 49, 51
 offsetHeight, 89
 offsetWidth, 89
 ol, 21
 on, 150
 onAbort, 85
 onBlur, 85
 onChange, 85
 onClick, 23, 84, 85
 onContextMenu, 85
 onDoubleClick, 85
 onError, 85
 onFocus, 85
 onInput, 85
 onKeyDown, 85
 onKeyPress, 85
 onKeyUp, 85
 onLoad, 84, 87
 onload, 88, 112, 119
 onMouseDown, 85
 onMouseMove, 85
 onmousemove, 91
 onMouseOut, 85
 onMouseOver, 85
 onMouseUp, 85
 onReset, 85
 onResize, 85
 onScroll, 85
 onSelect, 85
 onSubmit, 85
 onUnload, 84
 opacity, 12
 option, 6
 outerHeight, 104
 outerWidth, 104
 overflow, 14

 padding, 14
 pageX, 89
 pageY, 89
 parseFloat, 28
 parseInt, 27
 password, 5
 path, 133
 pathname, 105
 pause, 94
 pc, 10
 PI, 29
 play, 94
 playbackRate, 97
 polygon, 130
 polyline, 130
 pop, 45
 position, 14
 pow, 29
 preventDefault, 117
 progress, 7
 pt, 10
 push, 45

putImageData, 80
px, 10
p タグ, 9

radio, 4, 5
random, 29
range, 6
ratechange, 97
readAsDataURL, 119
readAsText, 112
rect, 136
reduceRight, 48
redue, 47
ReferenceError, 27
rel, 19
removeItem, 99
replace, 146
reset, 85
resize, 85
restore, 77
result, 112
return, 35, 140
reverse, 45
RGB (rgb), 11, 72
right, 17
rotate, 76
round, 29, 131, 132

sans-serif, 10
save, 77
scale, 76
screen, 104
screenX, 89
screenY, 89
script, 23
scroll, 85
scrollHeight, 89
scrollWidth, 89
search, 145
select, 6, 85
serif, 10
sessionStorage, 99
Set, 53
set, 54
setAttribute, 64, 123
setData, 116
setDate, 59

setDay, 59
setFullYear, 59
setHours, 59
setInterval, 98
setItem, 99
setLineDash, 72
setMilliseconds, 59
setMinutes, 59
setMonth, 59
setSeconds, 59
setTimeout, 97
Set の作成, 53
shift, 45
shift_jis, 3
show, 159
sign, 29
sin, 29
size, 53, 54
slice, 46
slideDown, 159
slideUp, 159
solid, 14
some, 49
sort, 48
span タグ, 18
splice, 46
split, 31
sqrt, 29
square, 132
src, 24, 76
static, 140
step, 6
string, 27
stroke, 71, 131
stroke-dasharray, 131
stroke-linecap, 132
stroke-linejoin, 131
stroke-miterlimit, 132
stroke-width, 131
strokeRect, 74
strokeStyle, 72
strokeText, 74
style, 9, 107
stylesheet, 19
style 属性, 9
submit, 85

substr, 31
substring, 31
SVG, 125
svg 要素, 125
SVG 要素のグループ化, 138
switch, 33

table, 20
tan, 29
target, 88, 112, 117
td タグ, 20
text, 5, 136, 152
text-align, 17
text-decoration, 12
text-indent, 11
textarea, 5
textContent, 118
this, 40, 140, 163
th タグ, 20
timeStamp, 88
timeupdate, 96
title, 3
top, 14, 107
toString, 28
translate, 76
trim, 164, 165
true, 27
TrueType, 168
tr タグ, 20
type, 88
typeof, 30

ul, 21
undefined, 27, 30, 54
unload, 84
unshift, 45
userAgent, 105
utf-8, 3

value, 6, 28, 63, 123
values, 55
var, 27
vertical-align, 18
video, 94
visibility, 108
volume, 96
volumechange, 96

W3C, 8, 19
Web で公開されているフォント, 170
while, 34
width, 14, 22, 79, 104
window, 68, 87, 104
write, 70
writeln, 70

x-euc-jp, 3
x-sjis, 3
XML, 125
xmlns, 125

アスペクト比, 22
値の比較, 30
アルファ値, 79
アロー関数, 38
アンダースコア「_」, 58
暗黙値, 40
位置とサイズの設定, 14
一斉処理, 47
イベント駆動型プログラミング, 84
イベント, 23, 84
イベントオブジェクト, 87, 88
イベント駆動, 24, 84
イベントリスナ, 84
色の設定, 11, 72
インスタンス, 140
インラインフレーム, 120
インライン要素, 18
ウィンドウイベント, 84
円, 136
円周率, 29
エントリ, 54
大きさの単位, 10
オブジェクト, 44, 50
オブジェクト指向, 140
オブジェクトを配列に変換, 52
折れ線, 130
改行, 65
改行要素, 4
拡張クラス, 140
箇条書き, 21
箇条書き項目を横並びにする, 21
箇条書きの見出し, 21
型, 27
型の検査, 30

可変長の引数, 39, 57
カレンダー, 162
関数, 23, 35
画素, 79
画像の位置, 22
基数, 27
基数の変換, 28
基数表現, 28
局所変数, 37
キー, 50
キーボードイベント, 84
逆正弦関数, 29
逆正接関数, 29
逆余弦関数, 29
行, 20
行内での上下の配置, 18
行の高さ, 11
区切り文字, 45
矩形, 136
クラス, 140
クラスメソッド, 140
繰り返し処理のスキップ, 35
繰り返し処理の中断, 35
繰り返しの表記, 146
グループ化, 145
グローバル変数, 37
桁の大きな整数, 30
検索, 145
降順, 48
コメント, 8, 19, 32
子要素, 69, 123
コンストラクタ, 140
コンテキスト, 71
サイズ, 15
四角形, 74, 136
四捨五入, 29
指数関数, 29
自然対数の底, 29
昇順, 48
シングルクォート, 30
進捗バー, 7
時間, 59
時刻, 59
順次処理, 47
剰余, 29
数値型, 27
スタイル, 9
スタック, 45
ストローク, 131
ストロークの太さ, 131
スプレッド構文, 39, 57
スライダ, 6, 65
正規表現, 31, 145
正規表現の記述方法, 146
正弦関数, 29
整数かどうかを調べる, 30
正接関数, 29
セル, 20
セル間の隙間, 20
セレクト, 9, 148, 154
線, 71
選択リスト, 6, 65
線の角の張り出し, 132
線の先端の形状, 132
線の曲がり角の形状, 131
絶対値, 29
全要素の消去, 53
添字, 44
ソート, 48
属性, 69
属性値, 123
大域変数, 37
対数関数, 29
タイミングイベント, 97
対話的な実行, 184
多角形, 130
タブ, 160
探索, 145
第一級オブジェクト, 38
ダウンロード, 174
楕円, 136
ダッシュの設定, 72
ダブルクォート, 30
段落開始のインデント, 11
段落要素, 4
値域, 35
チェックボックス, 4, 65
置換, 145
置換処理, 146
直線, 136
定義域, 35
テキスト, 136

テキストエディタ, 1, 2
テキストエリア, 5, 64
テキストノード, 122
テキストフィールド, 5, 64
点線, 131
テンプレート文字列, 31
デバッグ, 181
デバッグ機能, 24
デフォルト引数, 40
トリミング, 22
ドット, 123
ドラッグ, 115
ドラッグアンドドロップ, 115
ドロップ, 115
ドロー系グラフィックス, 125
名前空間, 125, 128
名前の衝突, 128
並べ替え, 48
塗り, 71, 131
塗りつぶしの色の設定, 72
配列, 39, 44
配列かどうかの判定, 46
配列の途中の要素の削除, 46
配列の部分列の抽出, 46
配列の連結, 45
配列要素の順序の逆転, 45
配列要素の連結, 45
破線, 131
バッククォート, 30
パス, 133
パスワードフィールド, 5, 64
パターンマッチ, 145
比較関数, 48
光の三原色, 12
引数, 35, 39
非数, 29
日付, 59
表, 20
表示位置, 15
ビューボックス, 126
ビューポート, 126
ピクセル, 79, 125
ファイル選択ダイアログ, 111
フォント, 168
フォントサイズ, 10, 74
フォントスタイル, 10, 74
フォントの設定, 10
フォントの強さ, 10
フォントファイルの組込み, 168
フォント名, 74
符号, 29
不透明度, 12
太さの設定, 72
部分文字列, 31
ブレークポイント, 179
ブロック, 42
ブロックレベル要素, 18
分割代入, 58
文書型宣言, 3
プリミティブ型, 27
ブレースホルダ, 31
プロパティ, 50
平方根, 29
変数, 27
変数のスコープ, 40
べき乗, 29
ベジェ曲線, 133
ペイント系グラフィックス, 125
ボタン, 4
マウスイベント, 84
マウスの位置, 89
巻き上げ, 41
未定義の記号, 30
無限大, 29
無名関数, 38, 39
メソッド, 140
メニュー, 161
メモ帳, 1
メータ, 7
文字間, 11
文字列, 136
文字列型, 27
文字列探索, 145
文字列の長さ, 31
文字列の比較, 31
文字列の分割, 31
文字列の連結, 31
戻り値, 35
要素数の取得, 53
要素の位置とサイズの取得, 109
要素の個数, 44
要素のサイズ, 89

要素の削除, 53
要素の存在検査, 53
要素の属性, 123
要素の属性と値, 63
要素の探索, 46
要素の抽出, 46
要素の追加, 53
余弦関数, 29
余白, 14
ライブラリ, 147
ラジオボタン, 4, 65
ラムダ式, 39
乱数, 29
リンクのスタイル, 12
真理値型, 27
累積的处理, 47
列項目, 20
列見出し, 20
連想配列, 50
ローカル変数, 37
枠線, 14, 20
枠線の太さ, 15

謝辞

本書の作成開始と維持にあたって協力いただいた武庫川女子大学，四條畷学園短期大学の多数の学生の方々に感謝いたします。また，本書を教育現場でご使用くださり，本書の内容に関する有用な助言をくださった平塚聡先生（現大谷大学，四條畷学園短期大学講師）に感謝します。

「JavaScript 入門」

ー HTML5 アプリケーション開発のためのプログラミング

IDEJ 出版 ISBN978-4-9910291-1-0 C3004

著者：中村勝則

発行：2020 年 3 月 14 日 第 1 版
2021 年 1 月 16 日 第 1.1 版

テキストの最新版と更新情報

本書の最新版と更新情報を，プログラミングに関する情報コミュニティ Qiita で配信しています。

→ <https://qiita.com/KatsunoriNakamura/items/4bb17b0238fcc9c4b6ed>



上記 URL の QR コード

本書はフリーソフトウェアです，著作権は保持していますが，印刷と再配布は自由にさせていただいて結構です。（内容を改変せずをお願いします） 内容に関して不備な点がありましたら，是非ご連絡ください。ご意見，ご要望も受け付けています。

● 連絡先

nkatsu2012@gmail.com

中村勝則