

JavaScript と jQuery

目次

1. JavaScript とは	2
1 JavaScript 概要	2
2 JavaScript プログラミング	2
2. JavaScript コードの記述方法	3
1 JavaScript コードの記述方法	3
2 コメント	5
3 セミコロン	6
4 大文字と小文字	6
5 空白スペース	8
6 script 要素	8
7 外部ファイル化	9
8 イベントハンドラ	10
3. 変数	12
1 変数宣言	12
2 変数名の命名規則	12
3 予約語	13
4. データ型	14
1 数値	15
2 文字列	15
3 論理値	16
4 null	16
5 未定義値	16

6	オブジェクト	17
7	配列	19
8	関数	22
5.	式と演算式	24
1	算術演算子	24
2	文字列結合演算子	24
3	比較演算子	26
4	論理演算子	27
5	代入演算子	28
6	その他の演算子	28
6.	if 文	29
7.	for 文	32
8.	while 文	33
9.	関数	34
1	function 文による関数定義	34
2	関数名を指定して呼び出す	35
10.	String オブジェクト	36
11.	イベント駆動型プログラミング	39
12.	Window オブジェクト	41
13.	フォーム	45
1	JavaScript から入力フォームへのアクセス	45
2	フォームの入力項目	46
3	入力値の取得と設定	46
4	入力項目の状態を変更する	47
14.	DOM	48
1	DOM と DOM API	48
2	W3C DOM	48
3	Document オブジェクト	49
15.	jQuery とは	53

1. JavaScript とは

1 JavaScript 概要

JavaScript は、Web ページを動的に制御するためのオブジェクト指向のスクリプトとして 1995 年に登場以来、様々な Web ページで利用されています。当初、JavaScript は手軽に動的な Web ページを作成できることから人気を博しましたが、Web ブラウザ間の互換性問題やセキュリティ問題などにより、次第に人気伸び悩むようになりました。しかし、近年 Google をはじめとした各種サービスで JavaScript が利用されたことから再び脚光を浴び、現在の Web アプリケーション開発において欠かせない技術となっています。ここでは JavaScript の概要として、JavaScript の動作例を交えながら、Web ページに JavaScript を組み込む方法を説明します。

JavaScript は、Internet Explorer、Chrome、Firefox、Safari など様々な Web ブラウザで利用可能なインタプリタ型のプログラミング言語です。JavaScript を使用することで、例えばリンク上にマウスポインタを移動させると画像が変化したり、説明文を追加したりといった動的な Web ページが作成できます。

JavaScript は、初心者でもとっつきやすいプログラミング言語ですが、様々なことができる本格的なプログラミング言語でもあります。主に Web ブラウザで利用されていますが、その用途は Web ブラウザだけではなく、最近では、Windows 上で動作するガジェットや Adobe AIR でのデスクトップアプリケーションを開発する際に使用するなど、様々なところで利用されています。

2 JavaScript プログラミング

JavaScript は、Web ブラウザとテキストエディタを準備できればプログラムを作成し、その動作を確認することができます。しかし、現実的にはメモ帳では機能が少なく、JavaScript 開発を快適に進めることはできません。例えば、HTML ファイルの中に JavaScript を記述する場合、HTML と JavaScript を色分けして表示してくれるテキストエディタを利用すると便利です。最近では、フリーソフトでも多機能なテキストエディタが多数あります。さらに本格的に JavaScript にて開発をする場合は、Web アプリケーションの統合開発環境を利用することで、効率よくアプリケーション開発ができます。本講義では Eclipse を使用します。

開発環境	説明
Aptana Studio	Eclipse ベースで作られた Web アプリケーションの統合開発環境。HTML、CSS、JavaScript、PHP、Python などでの開発をサポートしている
Eclipse	Java の開発環境として有名な統合開発環境。JavaScript 開発用のプラグインを導入することで JavaScript の開発が可能になる
Visual Studio	Microsoft が提供する Windows の開発環境。ASP.NET 開発向けの Visual Studio Express for Web で JavaScript 開発がサポートされている
WebStorm	JetBrains が提供する JavaScript 向けの統合開発環境。強力なコード解析機能や補完機能がサポートされている

2. JavaScript コードの記述方法

続いて、JavaScriptコードの記述方法について見ていきましょう。JavaScriptコードは、HTMLドキュメントの中に埋め込んで記述することで実行できます。HTMLドキュメントに埋め込む方法としては次の3つがあります。

- ① script 要素内に JavaScript コードを記述する
- ② <script>タグの src 属性から、JavaScript コードを記述した外部ファイルを読み込む
(「JavaScript コードの外部ファイル化」)
- ③ HTML のタグの属性としてイベントハンドラを設定し、属性値に JavaScript コードを記述する(「HTML タグのイベントハンドラでの記述」)

ここでは、はじめに①の実行方法をもとに、JavaScriptプログラミングの基本的な記述方法を説明していきます。そして、その後に残りの2つの実行方法について説明します。

1 JavaScript コードの記述方法

JavaScript では、HTML ファイルに埋め込んで Web ブラウザ上で動作(実行)させますが、その方法の1つに、「HTML ファイル内で<script>タグを利用し、その要素に JavaScript コードを記述する」方法があります。次のように、HTML ファイル内で<script>タグを利用し、<script>タグと</script>タグで囲まれた部分に JavaScript コードを記述します。

```
<script type="text/javascript">  
…この部分に JavaScript のコードを記述します。  
</script>
```

<script>タグは JavaScript 以外でも利用できるため、type 属性で利用するスクリプト言語の種類を設定します。JavaScript の場合、「text/javascript」を設定します。なお、HTML4.01 では<script>タグにおいて type 属性の指定が必須でしたが、HTML5 では type 属性が「text/javascript」の場合には省略可能となっています。本講座では、基本的に HTML5 表記でコード例を記載します。この記述方法に従って、文字列「Hello world!」を出力する JavaScript を HTML に組み込んだ記述例が以下になります。

01_JavaScript を組み込んだ例.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>サンプル</title>
  </head>
  <body>
    <script>
      document.write('<h1>');
      document.write('Hello world!');
      document.write('</h1>');
    </script>
  </body>
</html>
```

script 要素内で利用されている「(document.write(……))」という文は、丸括弧内の内容を HTML ドキュメントに出力する命令です。ここでは、シングルクォーテーション(')で囲まれた文字列を HTML ドキュメントに出力します。つまり、次の HTML を Web ブラウザで読み込んだ場合と同じ動作となります。

JavaScript 未使用の場合

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>サンプル</title>
  </head>
  <body>
    <h1>Hello world!</h1>
  </body>
</html>
```

2 コメント

コメントとは、プログラムの内容を自分自身や他のプログラマに対して伝えるために、プログラム内に記述する注釈のことです。コメントは後からソースコードの内容を見直す際の助けとなります。なお、プログラム実行時、コメント部分はすべて無視され、実行されません。

JavaScript のコメントの記述方法は以下の 2 種類があります。

- 単一行コメント
- 複数行コメント

単一行コメント

プログラム内に「//」を記述すると、その行以降の文字はコメント行として解釈されます。「//」を利用した記述例は、次の通りです。コメントは行頭からでも途中からでも記述可能です。

```
//コメント行です。自由に記述できます。  
var price = 100; // ここもコメントを記述できます
```

複数行コメント

「/*」と「*/」で囲まれた文字はコメント行として解釈されます。「/*」「*/」を利用した記述例は次の通りです。

```
/* コメント行です*/  
/* 複数行に渡って  
コメントを記述できます。*/
```

なお、コメントは入れ子状にした状態で利用できません。例えば、次の場合はエラーとなります。

```
/* /* 入れ子になったコメントです。*/ */
```


3 セミコロン

JavaScript では文の終わりにセミコロン(;)を記述します。また、次の例のように 1 行以上を波括弧({ })で囲む場合もあります。波括弧で囲まれた部分は「文ブロック」と呼び、複数文をまとめて 1 つの文として扱います。なお、この文ブロックの終わりにはセミコロンを記述しません。

```
for (var i = 0 ; i < 10; i++) {  
    document.write('Hello world!');  
    document.write('<br>');  
}
```

4 大文字と小文字

HTML のタグでは大文字と小文字を区別しませんが、JavaScript では大文字と小文字を区別します。例えば、前述の文字列「Hello world!」を出力する JavaScript を組み込んだ HTML にある「document.write('Hello world!');」の先頭文字「d」を大文字で記述すると実行時にエラーになります。



なお、上図のような JavaScript 実行時のエラー情報の表示方法は、Web ブラウザによって異なります。主要 Web ブラウザのエラー表示の表示方法は以下の通りです。


●Internet Explorer の場合

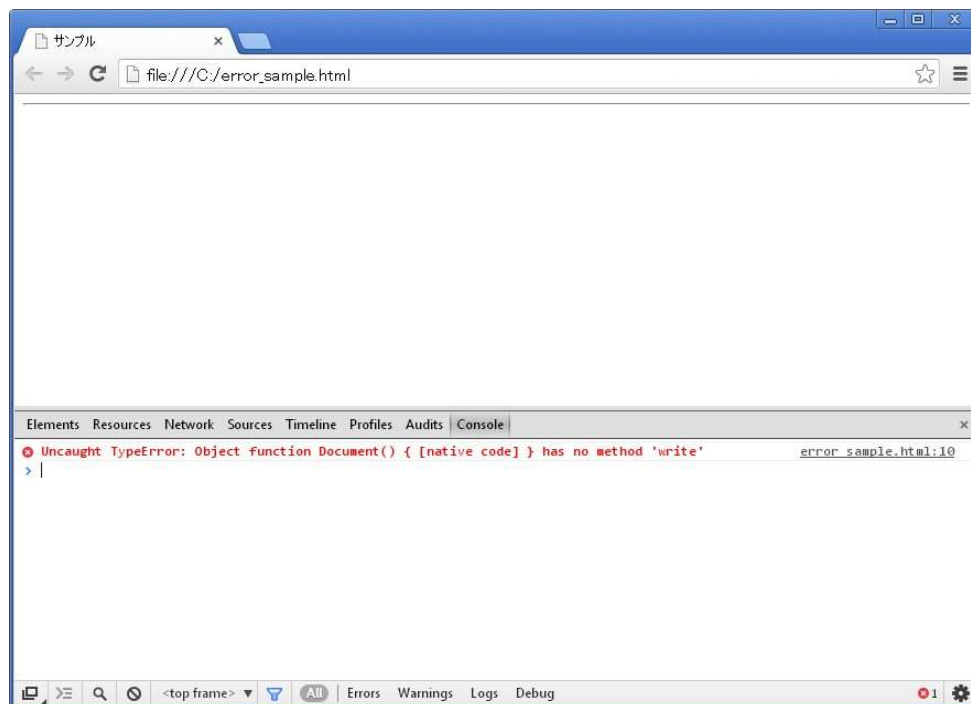
Internet Explorer では、F12 開発者ツールのコンソールにエラー情報が表示されます。

なお、JavaScript 実行時には自動的にエラー情報を表示させることも可能となっており、先程の図は、設定がオンの場合のエラー表示になります。表示させたい場合は次の手順で設定します。

1. メニューバーから [ツール] → [インターネットオプション] を選択します。
2. 「インターネットオプション」ダイアログが表示されるので、「詳細設定」タブを選択します。
3. 「詳細設定」タブ内で「設定」の「ブラウズ」→「スクリプトエラーごとに通知を表示する」項目を変更することで、JavaScript 実行時のエラー情報表示のオン/オフを設定します。

●Chrome の場合

Chrome では、JavaScript コンソールを表示させることで、エラー情報を表示できます。メニューバーの設定ボタン  から [ツール] → [JavaScript コンソール] を選択すると、コンソールが表示されます。なお、コンソールの表示には、Ctrl+Shift+J キーがショートカットキーとして割り当てられています。コンソールが表示された状態で JavaScript が実行すると、エラー情報が表示されます。



また、エラーコンソールで通知されたエラー箇所をクリックすると、該当箇所のコードが表示されます。


```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>サンプル</title>
6 </head>
7 <body>
8 <script>
9   document.write('<hr>');
10  Document.write('Hello world!');
11  document.write('<hr>');
12 </script>
13 </body>
14 </html>
```

Uncaught TypeError: Object function Document() { [native code] } has no method 'write'

5 空白スペース

JavaScript では、字句の間の空白スペース、タブ、改行コードを無視します。例えば、次のような記述も可能です。

```
document.write(
  'Hello world');
```

この性質により、1 文が長く複雑になった場合、処理のまとまりごとに改行を行い、JavaScript コードを整形しておくと、コードの可読性が向上します。

6 script 要素

HTML ドキュメント内に script 要素を複数記述できます。JavaScript は HTML ドキュメントに script 要素が現われる順序に従って実行されるため、後に記述された JavaScript は前に記述された JavaScript を参照できますが、逆の場合は参照できません。そのため、最初に読み込ませたい部分は<head>～</head>タグ間に記述するとよいでしょう。

なお、Web ブラウザのデフォルト状態では JavaScript は実行可能ですが、設定を変更することで Web ブラウザでの JavaScript 実行を不可にすることができます。この場合、JavaScript による処理が実行される場所では何も処理されないため、「何も表示(出力)されない」といったことが起こりえます。そのため、JavaScript が利用できない場合には、代替のメッセージなどを用意しておくのが望ましいでしょう。このような場合に活躍するのが<noscript>タグです。<script>タグの後に、次のように<noscript>～</noscript>タグを利用して、JavaScript が利用できない場合の HTML を記述しておきましょう。

JavaScript を無効にした場合の出力

```
<noscript>
  この Web サイトの全ての機能を利用するためには JavaScript を有効にする必要があります。
  <a href="http://www.enable-javascript.com/ja/" target="_blank">
  あなたの Web ブラウザーで JavaScript を有効にする方法</a>
  を参照してください。
</noscript>
```

7 外部ファイル化

JavaScript コードを HTML と分離して別ファイル化し、それを HTML に読み込むことで JavaScript を実行させることもできます。この場合、`<script>~</script>`などの HTML タグを含まないコードを、拡張子「.js」である JavaScript ファイルとして記述します。外部ファイル化の例として、サンプル「JavaScript を組み込んだ HTML」の JavaScript 部分を外部ファイル化したものを以下に示します。なお、外部ファイル名は「sample.js」とします。

```
document.write('<hr>');
document.write('Hello world!');
document.write('</hr>');
```

次に、HTML ドキュメントの JavaScript ファイルを読み込ませたい位置に`<script>`タグを記述し、その`<script>`タグの`src`属性に JavaScript コードの記述ファイルの URL を設定します。

```
<script type="text/javascript" src="ファイルの URL" charset="文字コード"></script>
```

また、HTML ファイルと JavaScript ファイルの文字コードが異なる場合は、JavaScript ファイルの文字コードを`charset`属性に設定します。例えば、HTML ファイルと JavaScript ファイルを同じフォルダに配置した場合、サンプルを外部ファイル化すると以下のように記述できます。

02_01 を sample.js ファイルを利用して記述した例.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>サンプル</title>
  </head>
  <body>
    <script src="sample.js"></script>
  </body>
</html>
```

このように、JavaScript コードを外部ファイル化し、HTML から JavaScript を分離することで以下のメリットが生まれます。

- HTML と JavaScript コードを分離できるため、HTML ファイルの可読性が高まる
- 異なる HTML ファイルで同じ JavaScript コードを利用する場合、その共通部分を外部ファイル化することで、コードの保守性が高まる

以上のメリットから、JavaScript コードはなるべく外部ファイル化して利用するのが望ましいでしょう。

8 イベントハンドラ

JavaScript はユーザの操作、すなわち Web ページのボタンをクリックする、テキストを選択するなどのイベントに対応した処理をさせたいときにも利用できます。発生したイベントをもとに、特定の処理を実施するコードのことを「イベントハンドラ」と呼びます。

イベントハンドラを設定するには、次の形式で、HTML タグのイベントハンドラに対応する属性に JavaScript の処理を記述します。

イベントハンドラ名 = "JavaScript コード"

利用可能なイベントハンドラは多数用意されています。例えば、フォーム内のボタンなどをクリック時に発生するイベントに対応するイベントハンドラ名は「onclick」となります。

次のコードは、ボタンをクリックした際にアラートダイアログを表示する JavaScript を組み込んだ HTML の記述例です。

イベントハンドラの利用例.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>サンプル</title>
  </head>
  <body>
    <form>
      <input type="button" value="アラート" onclick="alert('Hello World!')">
    </form>
  </body>
</html>
```



実行結果



前述のサンプルの form 要素に記述された<input>タグに注目すると、ボタンクリック時のイベントハンドラである onclick 属性に JavaScript の属性が記述されています。

```
<input type="button" value="アラート" onclick="alert('Hello World!')">
```

なお、onclick 属性に記述された「alert('Hello World!')」は、丸括弧内の内容をアラートダイアログに表示させる命令文です。

3. 変数

1 変数宣言

JavaScript で変数を利用する場合、まず変数を宣言する必要があります。変数を宣言することで、JavaScript で利用するデータの格納領域を確保します。変数を宣言する場合、次のように「var」キーワードを利用します。

```
var 変数名 ;
```

変数宣言時に値を設定することもできます。

```
var 変数名 = 値 ;
```

次のコードでは、初期値 20 で変数名「age」が宣言されます。このように JavaScript ではイコール(=)は等しいという意味ではなく、変数への「代入」を意味します。

```
var age = 20 ;
```

また、カンマ(,)を利用することで、複数の変数をまとめて宣言できます。例えば、変数名「name」と「age」の2つの変数をまとめて宣言する場合は、次のようになります。

```
var name, age ;
```

また、変数を宣言しただけで初期値を設定していない変数は、値が設定されるまで未定義値「undefined」という特殊な値となります。これについては後述します。

2 変数名の命名規則

JavaScript プログラミングでは、変数や関数などに対してユーザが定義する名称を「識別子」と呼びます。識別子は以下の命名規則に従う必要があります。

- 英字、アンダースコア(_)、ドル記号(\$)が利用できる。ただし、数字は先頭文字としては利用できない
- 予約語は利用してはならない。ただし、予約語を変数名の一部に含めることはできる

例えば、「address」「_name」「\$age」は命名規則に従っているため、識別子として利用できます。それに対して、「1st」や「2nd」は1文字目に数字が利用されているため、識別子として利用できません。また、予約語「for」は識別子として利用できませんが、「force」は利用できます。

3 予約語

予約語とは、制御構文や演算子など JavaScript の言語仕様であらかじめ定義されている特別なキーワードのことです。予約語は識別子(変数名や後述の関数名、ループラベルなど)として利用できません。

JavaScript の予約語を示します。

break	case	catch	continue	default	delete	do
else	debugger	finally	for	function	if	in
instanceof	new	with	return	switch	this	throw
while	try	typeof	var	void		

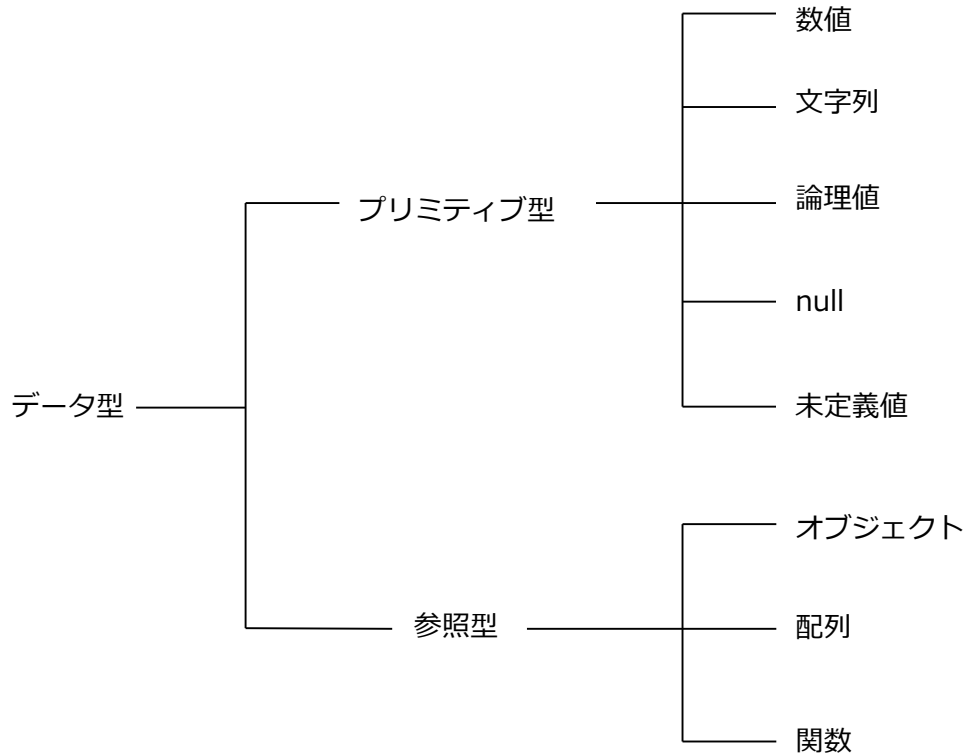
また、将来の使用を見越した予約語があります。現在は特別な機能を持っていませんが、将来機能を持つための、識別子として使用できません。

JavaScript の将来の使用を見越した予約語を示します。

class	extends	super	enum	import	export
-------	---------	-------	------	--------	--------

4. データ型

JavaScript で利用可能なデータ型はデータ値の格納方法の違いにより、大きく「プリミティブ型」と「参照型」の2つに分けられます。



プリミティブ型には「数値」「文字列」「論理値」に加え、特殊なデータ型である「null」「未定義値」が含まれます。

参照型に含まれる「オブジェクト」とは、複数のプリミティブ型や参照型のデータの集合体です。また、特殊なオブジェクトとして、データにインデックスを割り当ててまとめて取り扱う「配列」や特定の処理を取りまとめた「関数」があります。

以上の通り、JavaScript には様々なデータ型がありますが、JavaScript はデータ型をあまり意識しない言語仕様となっています。例えば、数値を格納した変数に対して文字列を代入できます。逆に、文字列を格納した変数に対して数値を格納できます。ただし、まったくデータ型を意識しなくてもよいというわけではありません。ここではまず JavaScript で扱うデータ型について説明し、その後「プリミティブ型」と「参照型」の違いを説明します。

1 数値

任意の数値データを取り扱うデータ型です。JavaScript では整数と浮動小数点数の区別をせず、すべての数値を浮動小数点数で取り扱います。

また、数値リテラルは整数リテラルと浮動小数点数リテラルに分類されます。

2 文字列

任意の文字列を取り扱うデータ型です。文字列リテラルはシングルクォーテーション(')もしくはダブルクォーテーション(")で囲むことによって定義できます。

文字列リテラルの例を次に示します。

```
'Shoeisya'
"JavaScript"
'100'
""
"I'm a developer."
```

この例にある通り、文字列には 0 文字で構成される文字("" もしくは "")も含まれます。文字列リテラルを定義する際、シングルクォーテーションとダブルクォーテーションのどちらを利用してもかまいませんが、上記の例にある「 "I'm a developer. " 」の場合、シングルクォーテーションを利用すると「 'I'm a developer.' 」となり、シングルクォーテーションで全体を囲めないで文字列リテラルになりません。このような場合、上記の例のようにダブルクォーテーションで囲むか、エスケープシーケンスを利用して対処します。

エスケープシーケンス	説明
¥0	NULL 文字
¥b	バックスペース
¥t	タブ
¥n	改行(LF : Line Feed)
¥v	垂直タブ
¥f	フォームフィード
¥r	復帰(CR : Carriage Return)
¥"	ダブルクォーテーション「 " 」
¥'	シングルクォーテーション「 ' 」
¥¥	バックスラッシュ「¥」

JavaScript を HTML に埋め込んで利用する場合、シングルクォーテーションとダブルクォーテーションの囲み方によっては、プログラムが動作しなくなってしまう。そのため、例えば HTML 部分はダブルクォーテーションを、JavaScript 部分はシングルクォーテーションを利用して文字列を囲んで記述するなどの工夫が必要です。

3 論理値

真(true)もしくは偽(false)のどちらかを取り扱うデータ型です。何かが真であることを true、何かが偽であることを false で定義します。論理値は制御文の条件式で 2 つの値を比較する際によく利用されます。例えば、次の式は変数 x が 0 であるかどうかを評価する式です。

```
x == 0
```

変数 x が 0 に等しい場合、論理値は true となり、等しくない場合は false となります。

4 null

null は値がないことを示す特殊な値で、「null」というリテラルのみ利用できます。オブジェクト型においてオブジェクトがないことを表す特殊な値として扱われます。

5 未定義値

未定義値は変数宣言しただけで値が初期化されていないことを示す特殊な値です。値が代入されていない変数や存在しないオブジェクトのプロパティを利用すると、未定義値として「undefined」が返却されます。

未定義値を出力した例

```
var x ;  
alert(x) ;
```



実行結果

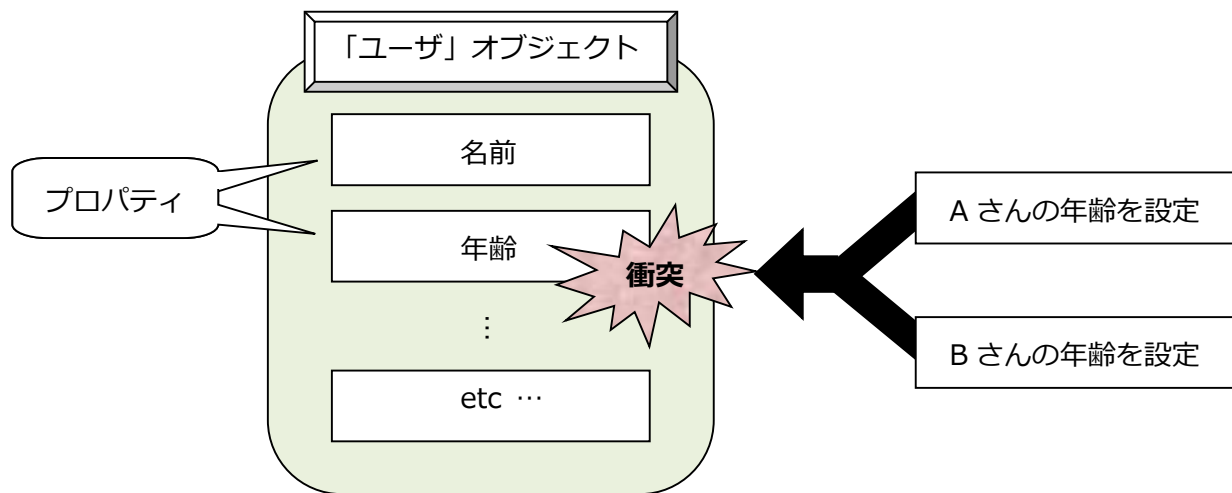


6 オブジェクト

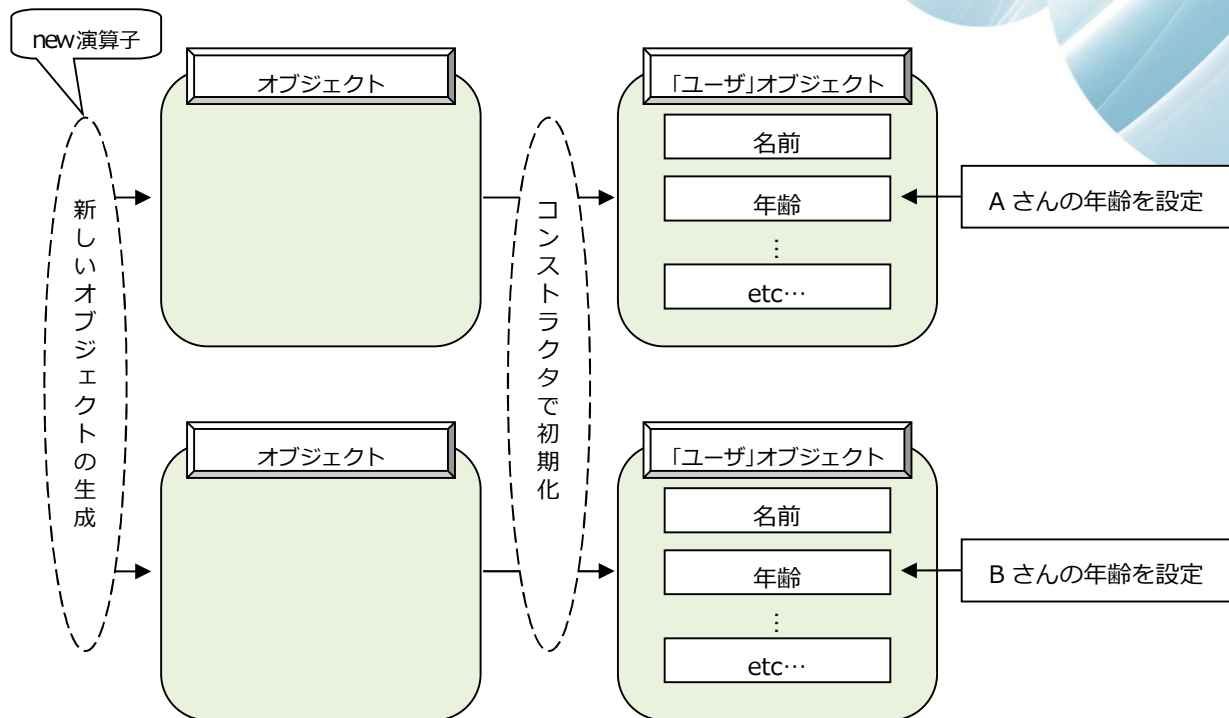
序列のない様々なデータ型の集合体を「オブジェクト」と呼びます。オブジェクトはこれらのデータ値を名前によって操作できます。この各データ値を「プロパティ」と呼び、プロパティにはオブジェクトの状態や特性を表す値が格納されます。また、オブジェクトに関連付いた関数を「メソッド」と呼びます。なお、関数とは一連の処理をまとめたものです。

例えば、Web ページの入力フォームを表す Form オブジェクトはフォームの送信やクリアといったフォームの処理をメソッドとして持っています。このように、オブジェクトはデータと処理を取りまとめて扱うことができるものといえます。

オブジェクトを利用するとデータ値の集合を共通の名前で利用できますが、どのようにオブジェクトを利用するのでしょうか。例えば、名前、年齢などのユーザ情報をプロパティに持つ「ユーザ」という名前のオブジェクトが定義されているとします。A さんのユーザ情報と B さんのユーザ情報を「ユーザ」オブジェクトに格納する場合、このオブジェクトが 1 つだけしか存在しないと、複数ユーザを扱うことができません。



そのため、オブジェクトを利用する場合、まず new 演算子を利用してプロパティが定義されていない新しいオブジェクトを生成します。次に、生成したオブジェクトに対して、「コンストラクタ」と呼ばれるオブジェクトを初期化する関数を呼び出し、プロパティを設定します。このように、新しいオブジェクトを作成し、保存領域を確保してからオブジェクトを定義するため、データを独立して扱えます。



JavaScript でのオブジェクトの生成は次のように記述します。

```
var 変数名 = new コンストラクタ名(引数, 引数, ...)
```

new 演算子とコンストラクタを利用することで、オブジェクトを生成できます。オブジェクトの生成時、コンストラクタに引き渡すデータを「引数」と呼びます。渡すべきデータがない場合は省略します。以下にオブジェクト生成の例を示します。

03_コンストラクタの利用例.js

```
//コンストラクタの定義
function user(name, age, gender) {
  this.name = name;
  this.age = age;
  this.gender = gender;
  this.showDetail = showDetail;
}
//メソッドの定義
function showDetail() {
  var detail = this.name + "¥n" + this.age + " years old¥n" + this.gender;
  alert(detail)
}
//オブジェクトの生成
var tom = new user("Tom", 21, "male");
tom.showDetail();
```



実行結果



オブジェクトのプロパティやメソッドを利用する場合はドット(.)を利用します。また、メソッドを呼び出す際、渡すべきデータがない場合は、引数を省略します。

変数名. プロパティ名
変数名. メソッド名(引数, 引数, ...)

7 配列

データ値の集合を「配列」と呼び、データ値の集合を共通の名前で利用できます。データ値をまとめて取り扱う点ではオブジェクトと一緒にですが、配列ではインデックスを付け、順序付けされた形で取り扱います。

配列は次の形式で定義(生成)できます。配列の生成にはオブジェクトの生成に利用する new 演算子と Array コンストラクタを利用します。

```
var 変数名 = new Array(引数, 引数, ...);
```

オブジェクトの生成時にコンストラクタに引き渡すデータを「引数」を呼びます。渡すべきデータのない場合は省略します。

また、配列リテラルは次のように記述します。

```
var 変数名 = [要素, 要素, 要素, ...];
```

配列内の各データ値は「配列要素」と呼び、配列要素の先頭から順に 0、1、2...と「インデックス番号(添字)」が割り当てられます。配列要素はデータ型の制約がないため、文字列と数値のような別々のデータ型を同時に格納できます。また、配列要素に何も定義しない場合(例えば [1, , , 4] のような場合)は、何も定義していない配列要素は未定義値(undefined)として扱われます。

配列内のデータ値を利用する場合、次のように変数名の後に角括弧([])を利用して要素の「インデックス番号」を指定します。

```
変数名[インデックス番号]
```

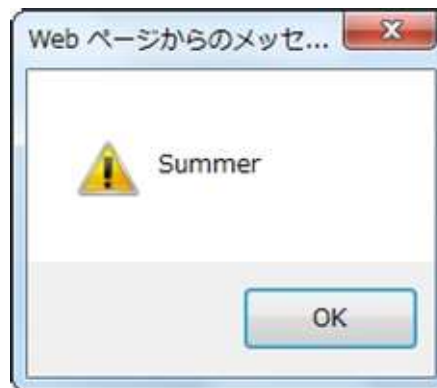

例えば、季節を表した 4 種類の文字列を配列として定義し、利用すると以下のようになります。このプログラムでは、配列のインデックスの「1」であるため、配列の 2 番目に格納している「Summer」が画面に表示されます。

配列を利用した出力例

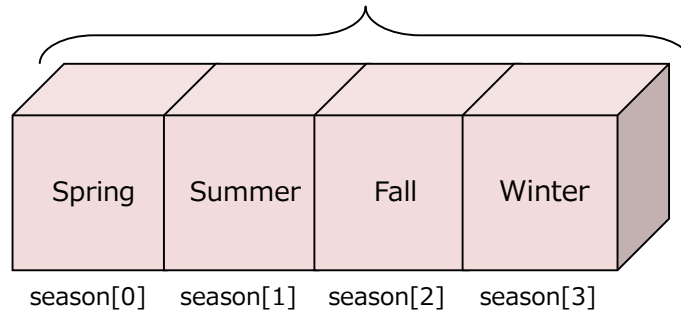
```
var season = ['Spring', 'Summer', 'Fall', 'Winter'];  
alert(season[1]);
```



実行結果



配列 season



また、配列の各要素には数字や文字列だけでなく、配列を定義できます。配列の各要素に配列を定義した場合、入れ子として定義された配列内のデータ値を参照するには、以下のように通常のインデックス番号の指定に続けて、入れ子として定義された配列内でのインデックス番号を指定します。

```
変数名[インデックス番号][インデックス番号]
```

以下のプログラムでは、配列のインデックスは「1」「0」であるため、配列の2番目に入れ子で格納している配列(['2', '3', '4'])の1番目のデータ値「2」が画面に表示されます。

多次元配列を利用した出力例

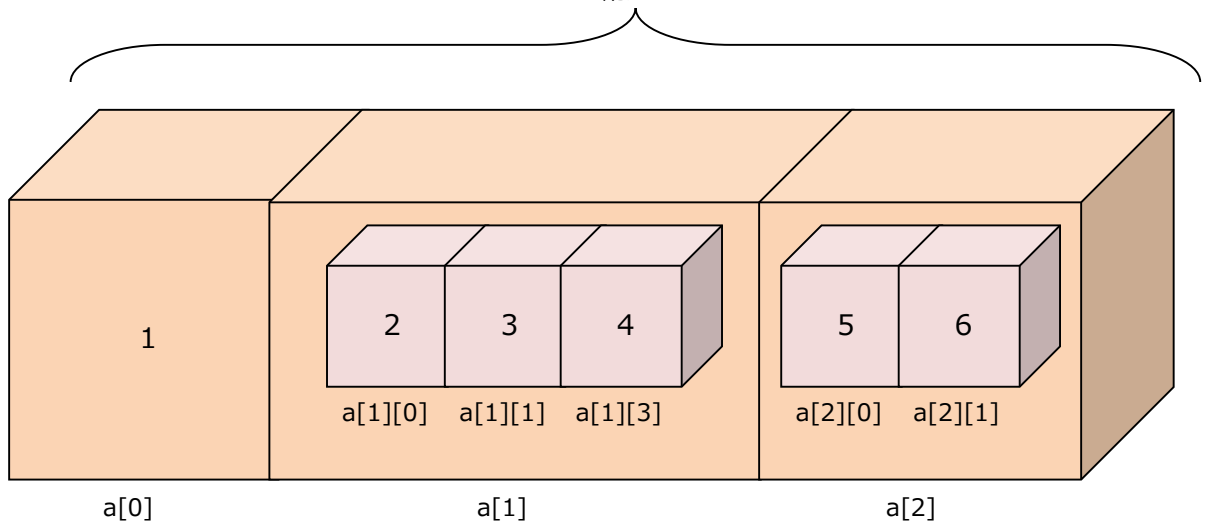
```
var a = ['1', ['2', '3', '4'], ['5', '6']];  
alert(a[1][0]);
```



実行結果



配列 a



8 関数

一連の処理をまとめたものを「関数」と呼びます。JavaScript の関数は「function」キーワードを使い次のように定義します。

```
function 関数名(引数, 引数, …){  
  複数の文  
  return 戻り値;  
}
```

関数に渡すデータを「引数」と呼びます。渡すべきデータがない場合は省略します。また、関数の呼び出しによって何らかの処理結果を呼び出し元に返却する場合、「return」の後にその値(戻り値)を記述します。

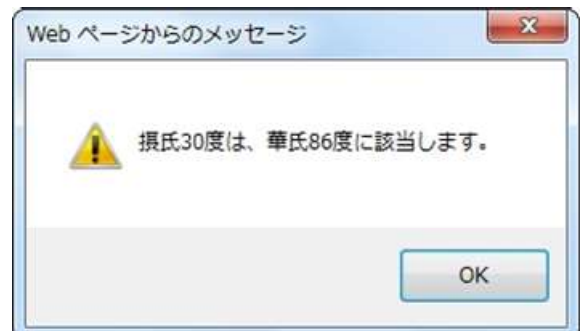
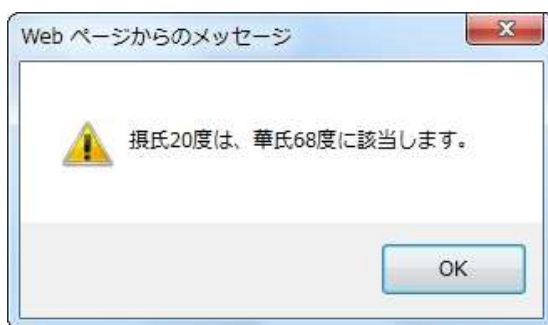
関数の定義例として、摂氏(C)から華氏(F)への変換処理を関数にしたものを次に示します。なお、摂氏から華氏への変換式は「 $F = 9 / 5 \times C + 32$ 」です。

関数の利用例

```
function convertCtoF(c) {  
  var f = 9 / 5 * c + 32;  
  return f;  
}  
alert('摂氏 20 度は、華氏' + convertCtoF(20) + '度に該当します。');  
alert('摂氏 30 度は、華氏' + convertCtoF(30) + '度に該当します。');
```



実行結果



上記サンプルでは、摂氏 20 度と 30 度を華氏に変換した結果を順番にアラート画面に表示しています。

JavaScript では function キーワードを使用して関数を定義しますが、関数式を利用した定義方法もあります。関数式を利用する場合も function による関数定義と同様の記述形式となりますが、関数名は省略できます。関数式を利用すると、次のように記述できます。

```
var 変数名 = function(引数, 引数, ...){  
  複数の文  
  return 戻り値  
}
```

このように、JavaScript では関数もデータ型として扱われるのが特徴です。また、関数リテラルの表記のように、関数名のないものを「無名関数」や「匿名関数」と呼びます。無名関数は関数の利用箇所であわせて関数を定義するため、呼び出し箇所とその処理の対応関係が理解しやすくなります。さらに、識別子が不要となり、識別子の重複に気を配る必要もありません。

5. 式と演算式

1 算術演算子

算術演算子はオペランドの数値をもとに加算(+)、減算(-)、乗算(*)、除算(/)といった四則演算などの算術演算をする演算子です。JavaScript で利用可能な算術演算子を示します。

演算子	説明	利用例	処理内容
+	加算の二項演算子	<code>x = y + 2;</code>	変数 <code>y</code> と 2 を加算した結果を変数 <code>x</code> に代入する
-	減算の二項演算子	<code>x = y - 1;</code>	変数 <code>y</code> から 1 を減算した結果を変数 <code>x</code> に代入する
*	乗算の二項演算子	<code>x = y * 10;</code>	変数 <code>y</code> と 10 を乗算した結果を変数 <code>x</code> に代入する
/	除算の二項演算子	<code>x = y / 2;</code>	変数 <code>y</code> を 2 で除算した結果を変数 <code>x</code> に代入する
%	剰余算の二項演算子	<code>x = y % 10;</code>	変数 <code>y</code> を 10 で除算した余りを変数 <code>x</code> に代入する。
+(単項)	単項演算子	<code>x = +y;</code>	変数 <code>y</code> の値をそのまま変数 <code>x</code> に代入する
-(単項)	符号反転の単項演算子	<code>x = -y;</code>	変数 <code>y</code> の値を符号反転して変数 <code>x</code> に代入
++	インクリメント演算子	<code>x++</code>	変数 <code>x</code> の 1 値を 1 増やす
--	デクリメント演算子	<code>x--</code>	変数の値を 1 減らす

2 文字列結合演算子

オペランドに文字列やオブジェクトの含まれる場合、「+」演算子は文字列結合演算子として扱われます。文字列結合演算子はオペランドを文字列として処理するため、オペランドが文字列以外の場合、文字列に変換してから各文字列が連結されます。

各データ型による「+」演算子での文字列結合の例を以下に示します。

01_「+」演算子による文字列結合.js

```
var result1 = 'Hello' + 'World';  
var result2 = '1' + '2';  
var result3 = '1' + 2;  
var result4 = 1 + 2;  
alert(result1 + '¥n' + result2 + '¥n' + result3 + '¥n' + result4);
```



実行結果



3 比較演算子

比較演算子は、2つのオペランドの値を比較し、その比較結果に応じて論理値 `true` もしくは `false` を返却する演算子です。JavaScript で利用可能な比較演算子を示します。

演算子	説明	利用例	処理内容
<code>==</code>	値が等しいかを評価する二項演算子	<code>x == 10</code>	x の値が 10 と等しい場合 <code>true</code> 、等しくない場合 <code>false</code> を返す
<code>===</code>	同一であるか(値と型が等しいか)を評価する二項演算子	<code>x === 10</code>	x の値と型が数値の 10 と等しい場合 <code>true</code> 、等しくない場合 <code>false</code> を返す
<code>!=</code>	値が等しくないかを評価する二項演算子	<code>x != 10</code>	x の値が 10 と等しくない場合 <code>true</code> 、等しい場合 <code>false</code> を返す
<code>!==</code>	同一でないか(値と型が等しくないか)を評価する二項演算子	<code>x !== 10</code>	x の値と型が数値の 10 と等しくない場合 <code>true</code> 、等しい場合 <code>false</code> を返す
<code><</code>	左辺 < 右辺であるかを評価する二項演算子	<code>x < 10</code>	x の値が 10 未満の場合 <code>true</code> 、10 以上の場合 <code>false</code> を返す
<code>></code>	左辺 > 右辺であるかを評価する二項演算子	<code>x > 10</code>	x の値が 10 超の場合 <code>true</code> 、10 以下の場合 <code>false</code> を返す
<code><=</code>	左辺 <= 右辺であるかを評価する二項演算子	<code>x <= 10</code>	x の値が 10 以下の場合 <code>true</code> 、10 超の場合 <code>false</code> を返す
<code>>=</code>	左辺 >= 右辺であるかを評価する二項演算子	<code>x >= 10</code>	x の値が 10 以上の場合 <code>true</code> 、10 未満の場合 <code>false</code> を返す

==演算子、===演算子

等価演算子「`==`」と厳密等価演算子「`===`」は、ともにオペランドに任意のデータ型を指定でき、2つのオペランドの値が等しいかどうかを比較する演算子ですが、「等しい」と評価する基準が異なります。

`===`演算子はデータ型を含めて等しいかどうかを評価します。`==`演算子はオペランドに異なるデータ型が指定された場合、オペランドは次のように比較されます。

- オペランドの組み合わせが数値と文字列の場合、文字列を数値に変換して比較する
- オペランドの組み合わせに論理値が含まれる場合、数値に変換して比較する
- オペランドの組み合わせがオブジェクトと数値または文字列の場合、オブジェクトを `valueOf()` メソッドなどを利用してプリミティブ型の数値に変換してから、比較する
- オペランドの組み合わせが `null` と `undefined` の場合、等しい値として判定される

また、等価演算子「==」と厳密等価演算子「===」では、オペランドに数値や文字列などのプリミティブ型が指定された場合は値そのものを比較しますが、オブジェクトや配列などの参照型が指定された場合は参照先で比較します。例えば、以下のようなコードの場合、配列 a と配列 b は参照先がそれぞれ別となるため、等しくないと判断されます。

参照型を比較した例

```
var a = ['1', '2', '3'];
var b = ['1', '2', '3'];
alert(a == b);
```



実行結果



4 論理演算子

論理演算子は、理論値のオペランドに対して、否定(NOT)、論理積(AND)、論理和(OR)などの論理演算をする演算子です。JavaScript で利用可能な論理演算子を示します。

演算子	説明	利用例	処理内容
!	否定(NOT)を演算する単項演算子	!x	x の値が false の場合 true、true の場合 false を返す
&&	論理積(AND)を演算する二項演算子	x>0 && y>0	「x>0」かつ「y>0」の場合 true、それ以外の場合 false を返す
	論理和(OR)を演算する二項演算子	x>0 y>0	「x>0」または「y>0」の場合 true、それ以外の場合 false を返す

論理演算式のうち、「&&」と「||」は左オペランド(式)から順番に評価します。「&」と「|」は必ず評価しますが、「&&」と「||」は、まず左オペランド(式)を評価し、次に必要な場合のみ右のオペランドを評価します。例えば、次の式は変数 x が 0 以下の場合、右のオペランド(式)の「y > 0」は評価されません。

```
var result = (y > 0) && (y > 0);
```

これは、x が 0 以下だと「&&」の左側の式が false となり、右側の式の評価結果にかかわらず結果が false と確定するためです。

5 代入演算子

代入演算子は、変数にデータ値を設定する演算子で「=」を利用します。また、代入演算子である「=」以外に、代入演算と算術演算やビット演算を組み合わせた「複合代入演算子」があります。例えば、変数 x に対して「100」を加算したい場合、次のような記述になります。

```
x = x + 100 ;
```

上記のコードは演算子「+=」を利用して、次のような記述も可能です。

```
x += 100;
```

6 その他の演算子

JavaScript では、ここまでで紹介してきたもの以外にも次の演算子をサポートしています。

演算子	説明
delete	オブジェクトのプロパティなどを削除する単項演算子
in	オブジェクトにプロパティが含まれているかを確認するための二項演算子
instanceof	オブジェクトの種類を確認するための二項演算子
new	新しいオブジェクトを生成する単項演算子
typeof	データ型を調べるための単項演算子
void	未定義値を返却する単項演算子
カンマ(,)	左側、右側の順にオペランドを評価し、右側の式の結果を返却する二項演算子

6. if 文

if 文は条件によって処理を分岐させる場合に利用します。はじめに if 文の基本的な書式を見てみましょう。

```
if (条件式) { 文 }
```

if 文では「条件式」を評価し、その結果が true の場合、「文」を実行します。結果が false の場合、「文」は実行されず if 文の次の文が実行されます。「条件式」には比較演算子がよく利用されます。なお、条件式の評価結果が true もしくは false でない場合は、論理値に変換されて評価されます。

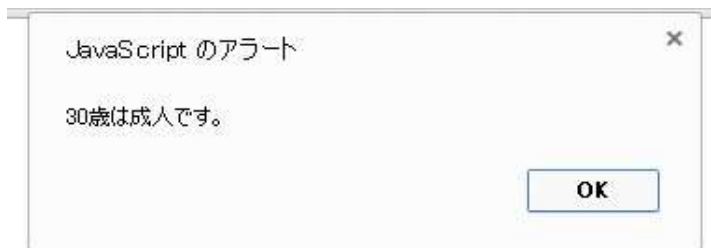
if 文の利用例として、変数 age が 20 以上の場合、if 文の内容が実行されアラートダイアログに「成人」と出力するプログラムを以下に示します。

01_if 文による成人の判定.js

```
var age = 30;  
if (age >= 20) {  
    alert(age + '歳は成人です。');  
}
```



実行結果



しかし、このサンプルの変数 age を 20 未満の値に修正して実行した場合、if 文の内容が実行されないため、何も起きません。

if 文での条件式の評価結果が true の場合だけでなく、false になった場合にも何らかの処理を実行したい場合があります。この場合、次のように if 文には else 文を追加した書式(if/else 文)があります。

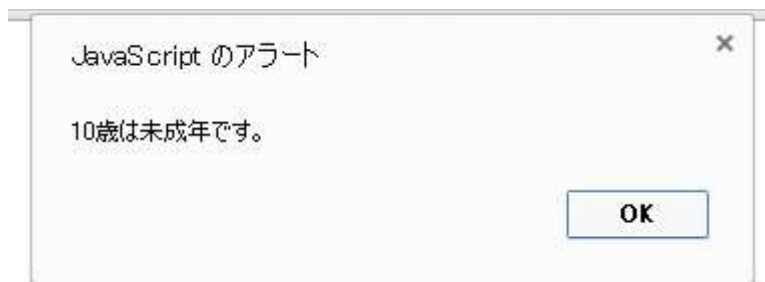
```
if (条件式) {  
  文 1  
} else {  
  文 2  
}
```

02_if/else 文による成人／未成年の判定.js

```
var age = 10;  
if (age >= 20) {  
  alert(age + '歳は成人です。');  
} else {  
  alert(age + '歳は未成年です。');  
}
```



実行結果



3 つ以上の分岐処理の場合、if/else 文のみではネストする必要があります。このような場合、else 文の代わりに else if 文を利用します。else if 文を利用した if 文の書式は次の通りです。

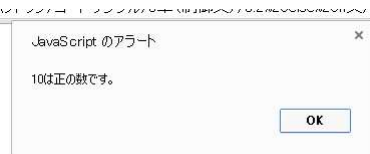
```
if (条件式 1) {  
  文 1  
} else if (条件式 2) {  
  文 2  
  .  
  .  
} else if (条件式 m) {  
  文 m  
} else {  
  文 n  
}
```

03_else if 文による数値の判定.js

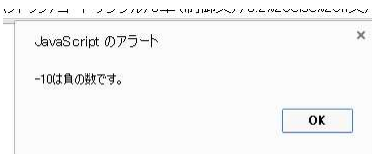
```
var num = 10;  
if (num > 0) {  
    alert(num + 'は正の数です。');  
} else if (num < 0) {  
    alert(num + 'は負の数です。');  
} else {  
    alert('0 です。');  
}
```



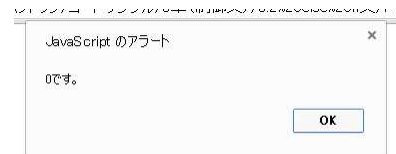
実行結果



変数 num が
10(正の数)の場合



変数 num が
-10(負の数)の場合



変数 num が
0 の場合

7. for 文

for 文を利用すると、文を繰り返し実行できます。for 文の書式は次の通りです。

```
for (初期化式; 条件式; 更新式) { 文 }
```

「初期化式」は最初に 1 回実行されます。「初期化式」ではループを制御する変数に初期値を与えます。この変数を「ループ制御変数」と呼びます。「初期化式」が実行されると、ループが開始されます。ループ本体を実行する前に、まず「条件式」が評価されます。「条件式」が true である場合、ループ本体の文を実行します。最後に「更新式」を実行します。「更新式」はループ制御変数の値を一定量増減させる式となります。

for 文の利用例として、0~9 までの数値を HTML ドキュメントに出力するプログラムを以下に示します。このプログラムを実行すると、0~9 までの数値が順番に出力されます。

01_for 文による数値の出力.js

```
for (var i = 0; i < 10; i++) {  
    document.write(i + '<br/>');  
}
```



実行結果

0
1
2
3
4
5
6
7
8
9



8. while 文

for 文の他に繰り返し処理するための制御文として while 文があります。for 文は繰り返しの回数があらかじめ決まっている場合は便利な制御文です。しかし、繰り返し回数をあらかじめ決められない場合があります。このような場合に有効な制御文が while 文となります。while 文の書式は次の通りです。

```
while (条件式) { 文 }
```

while 文はループ処理の最初に「条件式」を評価します。「条件式」が真の場合、「文」を実行し、「条件式」が偽の場合、次の処理に進みます。

while 文の処理の流れを見ると、処理を繰り返すかどうかの判定を繰り返しの先頭部分で実施します。そのため、while 文は「条件式」が満たされなくなるまで処理を繰り返す文といえます。while 文の利用例として、0~9 までの数字を順番に HTML ドキュメントに出力するプログラムを以下に示します。

01_while 文による数値の出力.js

```
var i = 0;
while (i < 10) {
  document.write(i + '<br/>');
  i++;
}
```



実行結果

0
1
2
3
4
5
6
7
8
9



9. 関数

関数は大きく以下の2種類に分けられます。

- JavaScript に用意されている関数(ビルトイン関数)
- プログラム内で新たに定義する関数(ユーザ定義関数)

ここでは、ユーザ定義関数(自作関数)の定義の仕方について説明します。

1 function 文による関数定義

関数を定義するには、「function」キーワードを使用します。function 文では function キーワードの後に関数名を記述し、その後ろに()で囲んで引数を定義します。引数が複数ある場合は、カンマ(,)で区切ります。

関数の内容は、引数の後ろを { } で囲み、その中に文を記述します。文には、すでに説明した変数の定義や、別の関数の呼び出しなどを記述できます。また、戻り値がある場合は「return」キーワードの後に式や値を指定することで、関数の呼び出し元に結果を返却できます。

```
function 関数名(引数、引数、……) {  
  複数の文  
  return 戻り値;  
}
```

01 体積によってサイズを判定する処理(通常関数).html

```
// 円錐の体積を求める関数  
function calcCone(radius, height) {  
  // 底面の面積  
  var base = radius * radius * Math.PI;  
  // 体積 = 1/3 * 底面積 * 高さ  
  return base * height / 3;  
}
```

なお、JavaScript では、戻り値のデータ型を指定できません。そのため、関数を呼び出した結果どのようなデータ型が返却された、関数を呼び出した側で判断する必要があります。プログラムがわかりにくくなるためあまり利用はされませんが、関数の実行結果によって、異なるデータ型の戻り値を返すことも可能です。

2 関数名を指定して呼び出す

続いて、ここでは関数の呼び出しについて説明します。定義した関数名(または、関数を代入した変数)の後ろに()を付け、この中に引数を指定して関数を呼び出します。引数が1つもない場合は単に()だけを記述します。

関数名([引数、引数、……]);

関数の呼び出し例

```
calcCone(10, 5);
```

関数を右辺にして左辺に変数を指定することで、関数の戻り値を変数に代入できます。

02 関数の戻り値を変数に代入.html

```
var base = calcCone(100, 50);  
if (base >= 50) {  
  alert('計算結果を表示します');  
}
```

また、関数の引数やif文などの条件文に、関数の戻り値をそのまま使用することも可能です。

03 if文の条件に関数の戻り値を直接利用する.html

```
if (calcCone(100, 50) >= 50) {  
  alert('計算結果を表示します');  
}
```

10. String オブジェクト

JavaScript での文字列に関する処理には String オブジェクトを利用します。文字列を変数に代入するには文字列リテラル「`" "`」を使用する方法もありますが、String オブジェクトでも同様のことが可能です。String オブジェクトには、文字列を格納するだけでなく、文字列の加工や特定の文字の検索など、文字列を扱う際に便利ないくつかのメソッドが用意されています。

String オブジェクトを生成する構文は以下の通りです。引数に指定した文字列から String オブジェクトを生成します。

```
var 変数名 = new String(文字列);
```

String オブジェクトで利用できるプロパティ、メソッドを示します。

プロパティ名	説明
length	文字列の長さ(文字数)を示す

メソッド名／構文名	説明
charAt(index)	引数に指定した位置(index)にある文字を返す
charCodeAt(index)	引数に指定した位置(index)にある文字コードを返す
concat(str1[,str2...])	String オブジェクトが保持している文字列から、引数に指定した文字列(str)を追加した内容を返す
indexOf(searchStr[,start])	String オブジェクトが保持している文字列から、引数に指定した文字列(searchStr)を検索し、その位置を返す(第 2 引数に開始位置(start)が指定されている場合はその位置から検索を開始する)
lastIndexOf(searchStr[,start])	String オブジェクトが保持している文字列の後ろから前へ、引数に指定した文字列(searchStr)を検索し、その位置を返す(引数に開始位置(start)が指定されている場合はその位置から前に向かって検索する)
search(regex)	引数に指定した正規表現のパターン(regex)に一致した場合、文字列内で一致した箇所のインデックスを返し、一致しなかった場合は-1 を返す
match(regex)	引数に指定した正規表現のパターン(regex)に一致した場合、一致したパターンの全情報を配列で返し、一致しなかった場合は null を返す
replace(condition,replacement)	引数に指定した検索条件(condition)に一致する文字列を、別の文字列(replacement)に置換する(検索条件には文字列または正規表現が指定できる)
toLowerCase()	文字列をすべて小文字に変換した文字列を返す
toUpperCase()	文字列をすべて大文字に変換した文字列を返す
toString()	String オブジェクトが保持している文字列を返す
trim()	文字の両端のスペース(半角、全角、タブ、改行)を取り除く

valueOf()	String オブジェクトが保持している文字列を返す
slice(start[,end])	引数に指定した開始位置(start)と終了位置(end)で、文字列を切り出して返す(引数 end を省略した場合は文字列の最後まで切り出す)
split(delimiter[,limit])	String オブジェクトが保持する文字列を、引数に指定した区切り文字(delimiter)で分割した配列を返す(引数に上限要素数(limit)が指定されている場合は、この数を越えた分は無視される)
substring(start[,end])	引数に指定した開始位置(start)と終了位置(end)で文字列を切り出して返す(引数 end を省略した場合は文字列の最後まで切り出す)
localeCompare(target)	String オブジェクトが保持している文字列と、引数に指定した文字列(target)を、ロケール固有の順序付けに従って比較し、結果を数値で返す

● 文字列検索

match()メソッドを用いると、指定した正規表現のパターンに一致する文字が、文字列中に存在するか否かを調べることができます。例を次に示します。

02 文字列の検索.html

```
var target1 = '123-4567';
// 正規表現を使用して任意の数字を検索する
var result1 = target1.match(/^[0-9]{3}-[0-9]{4}$/);

document.write('検索 1: ' + result1 + '<br>');

var target2 = '1234567';
// 正規表現を使用して任意の数字を検索する
var result2 = target2.match(/^[0-9]{3}-[0-9]{4}$/);

document.write('検索 2: ' + result2 + '<br>');
```



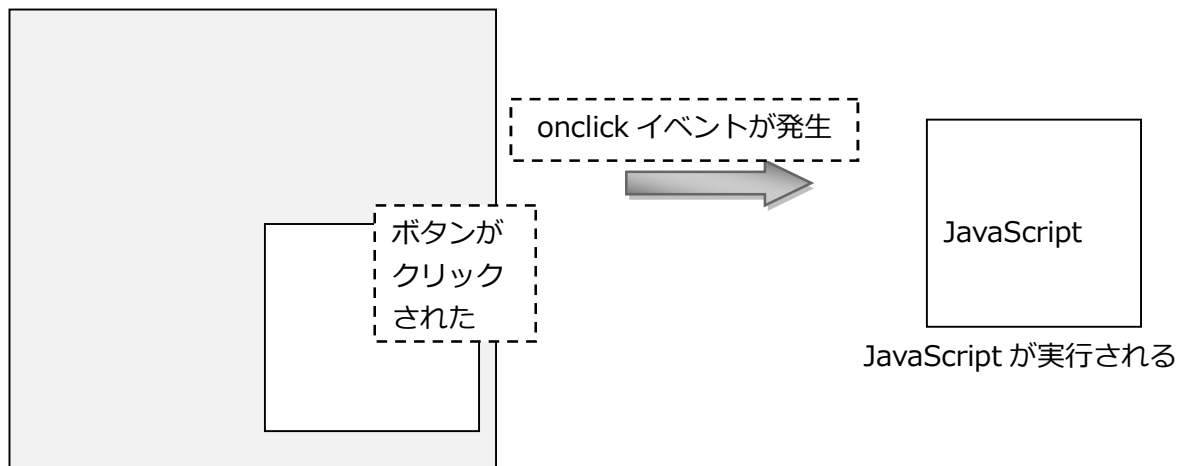
実行結果



`match()`メソッドは、引数に指定した正規表現のパターンのうち、どのパターンが一致したのかを配列で返します。ここでは、「/[^][0-9]{3}-[0-9]{4}\$/[/]」という正規表現を使用して、指定した文字列が郵便番号の形式であるか否かを調べています。正規表現のパターンに一致しなかった場合は `null` が返されます。

11. イベント駆動型プログラミング

イベントとは「ボタンがクリックされた」「入力項目からフォーカスが外れた」「マウスが特定の要素に重なった」など、ユーザの操作に応じて発生する信号のようなものです。このようなイベントに対して任意の JavaScript を実行できます。これは JavaScript に限らず、GUI を伴うアプリケーションでは一般的な手法で、「イベント駆動型」のプログラミングモデルと呼ばれています。



JavaScript でイベント処理を記述するには、HTML タグのイベントハンドラ属性を用います。イベントハンドラとは、「on イベント名」という名称の属性で、イベントの種類に応じて様々なものが存在します。

```
<input type="button" value="クリック" onclick="alert('ボタンが押されました')"/>
```

イベントハンドラ属性にはセミコロン(;)で区切ることで、複数の文を記述することもできます。

```
<input type="button" value="クリック"
onclick="alert('セミコロンで区切って'); alert('複数の文を記述できます')"/>
```

しかし、属性値に多くの JavaScript を記述すると HTML が見づらくなってしまいます。そのような場合には、イベント処理を script 要素や外部ファイルに関数として定義しておき、イベントハンドラ属性にはそれらの関数を呼び出すコードを記述します。

01_イベントハンドラ属性からの関数の呼び出し.html

```
<script>
  function showDialog () {
    alert(document.form.name.value + 'さん、こんにちは!');
  }
</script>
.
.
.

<form name="form">
  <input type="text" name="name">
</form>
<input type="button" value="クリック" onclick="showDialog()"/>
```

HTML タグのイベントハンドラ属性には次のようなものがあります。

属性名	説明(呼び出されるタイミング)
onclick	クリックされたとき
ondblclick	ダブルクリックされたとき
onmousedown	マウスのボタンが押下されたとき
onmouseup	マウスのボタンを放したとき
onmouseover	マウスが要素上に移動したとき
onmouseout	マウスが要素上から外れたとき
onmousemove	マウスが要素内で移動したとき
onchange	フォーム内の要素について値が変更されたとき
onblur	フォーム内の要素についてフォーカスが外れたとき
onfocus	フォーム内の要素についてフォーカスが当たったとき
onkeydown	キーを押したとき
onkeypress	キーを押して放したとき
onkeyup	キーを放したとき
onselect	テキストフィールドやテキストエリアでテキストが選択されたとき
onsubmit	フォームが送信される前。属性値に記述した JavaScript が true を返す場合はフォームが送信され、false を返す場合はフォームが送信されない。
onreset	フォームがリセットされる前。属性値に記述した JavaScript が true を返す場合はフォームがリセットされ、false を返す場合はフォームがリセットされない。
onload	Web ページがロードされたとき
onunload	Web ページがアンロードされる時

12. Window オブジェクト

Window オブジェクトは Web ブラウザ環境で動作する JavaScript のグローバルオブジェクトです。Window オブジェクトは「window」という変数で参照できます。Window オブジェクトは Web ブラウザを制御したり、Web ページのコンテンツにアクセスしたりするための様々なプロパティやメソッドを提供しています。例えば、アラートダイアログを表示する alert() 関数は、実は Window オブジェクトのメソッドとして定義されています。つまり、以下の 2 つのコードは等価です。

```
alert('Hello JavaScript ! ');
window.alert('Hello JavaScript ! ');
```

Window オブジェクトのプロパティとメソッドを示します。

プロパティ名	説明
closed	Window が閉じているかどうか
document	Document オブジェクト
frames[]	このウィンドウ内のフレーム
history	History オブジェクト
innerHeight,innerWidth	ウィンドウの表示領域の高さと幅(ピクセル単位)。Internet Explorer 8 以前はこれらのプロパティをサポートしていない。
localStorage	永続的なクライアントサイドストレージ
location	Location オブジェクト
name	ウィンドウの名前。open()メソッドの引数や、frame 要素の name 属性で指定された値。この名前は a 要素や form 要素の target 属性の値として指定できる。
navigator	Navigator オブジェクト
opener	open()メソッドでこのウィンドウを開いたウィンドウ
outerHeight,outerWidth	ウィンドウの高さと幅(ピクセル単位)。Internet Explorer 8 以前はこれらのプロパティをサポートしていない。
pageXOffset,pageYOffset	水平方向、垂直方向へのスクロール量(ピクセル単位)。Internet Explorer 8 以前はこれらのプロパティをサポートしていない。
parent	このウィンドウがフレーム内のウィンドウである場合、親ウィンドウへの参照。このウィンドウ自身がトップレベルウィンドウの場合、自分自身の参照
screen	Screen オブジェクト
screenLeft,screenTop	画面上でのウィンドウの左上の座標。Firefox はこのプロパティをサポートしていない。
screenX,screenY	画面上でのウィンドウの左上の座標。Internet Explorer 8 以前はこれらのプロパティをサポートしていない。
self	window プロパティと同じくこのウィンドウ自身への参照
sessionStorage	ウィンドウやタブを閉じるまで有効なクライアントサイドストレージ

status	Web ブラウザのステータスバーに表示する文字列
top	このウィンドウがフレーム内のウィンドウである場合、トップレベルウィンドウへの参照。このウィンドウ自身がトップレベルウィンドウの場合、自分自身の参照
window	self プロパティと同じくこのウィンドウ自身への参照

メソッド名／構文	説明
alert(message)	引数に指定したメッセージ(message)で警告ダイアログを表示する
blur()	ウィンドウからフォーカスを外す
clearInterval(intervalID)	引数に指定した ID(intervalID)の繰り返し処理をキャンセルする
clearTimeout(timeoutID)	引数に指定した ID(timeoutID)のタイマー処理をキャンセルする
close()	ウィンドウを閉じる
confirm(question)	引数に指定したメッセージ(question)で確認したダイアログを表示する
focus()	ウィンドウにフォーカスを与える
moveBy(x,y)	ウィンドウを、引数に指定したピクセル分(x,y)移動する
moveTo(x,y)	ウィンドウを、引数に指定した座標(x,y)に移動する
open()	新しいウィンドウを開く
print()	表示しているドキュメントを印刷する
prompt(message,default)	ユーザからの入力値を取得するためのダイアログを表示する。第 1 引数にはダイアログに表示する文字列(message)、第 2 引数には入力エリアの初期値(default)を指定する
resizeBy(width,height)	ウィンドウを、引数に指定したピクセル数(width,height)分広げる
resizeTo(width,height)	ウィンドウを、引数に指定したサイズ(width,height)にリサイズする
scrollBy(x,y)	ウィンドウを、引数に指定したピクセル数(x,y)分スクロールする
scrollTo(x,y)	ウィンドウを、引数に指定した座標(x,y)にスクロールする
setInterval(code,interval)	引数に指定したコード(code)を一定間隔(interval)で実行する
setTimeout(code,delay)	引数に指定したコード(code)を指定時間(delay)経過後に実行する
postMessage(message,targetOrigin)	異なるサーバのコンテンツを表示しているウィンドウ間で通信を行う

以降ではこれらのうち、alert()と confirm()の詳しい利用方法について説明していきます。

●警告ダイアログ

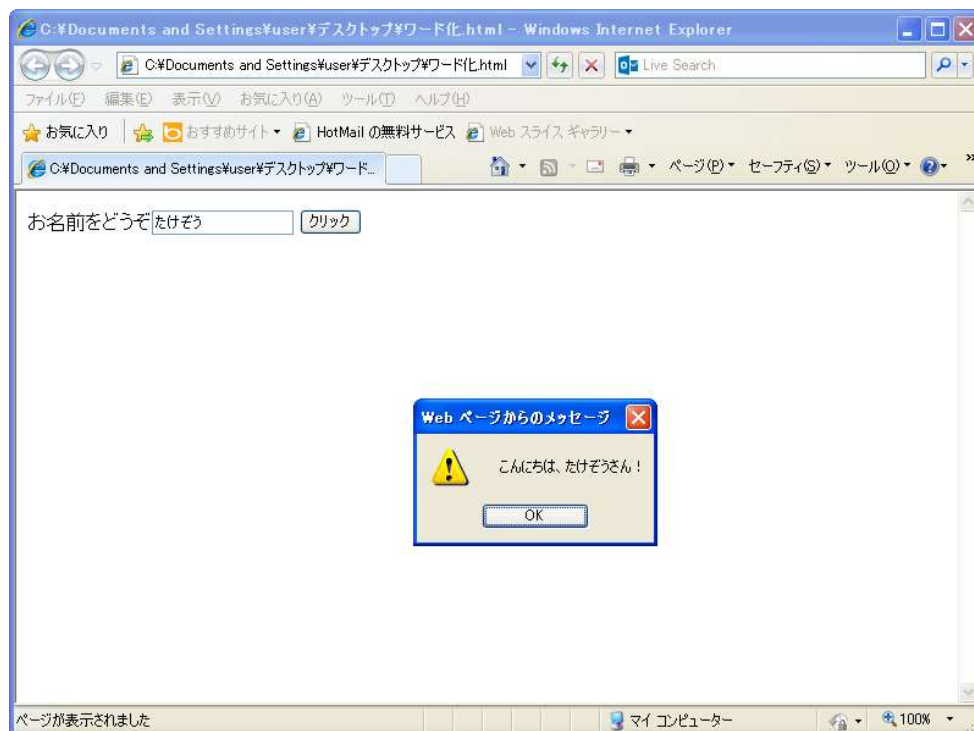
alert()メソッドは警告ダイアログを表示します。作成中のスクリプト内で変数の値を表示させるなど、簡単なデバッグ用途に利用することもあります。

alert()メソッドの利用例を次に示します。

01_alert()メソッドで警告ダイアログを表示.html

```
<!DOCTYPE html>
<html>
  <body>
    <form name = "form">
      お名前をどうぞ<input type= "text" name= "name"/>
      <input type= "button" value= "クリック"
        onclick= "alert('こんにちは、 ' + document.form.name.value + 'さん !')"/>
    </form>
  </body>
</html>
```

実行画面



●確認ダイアログ

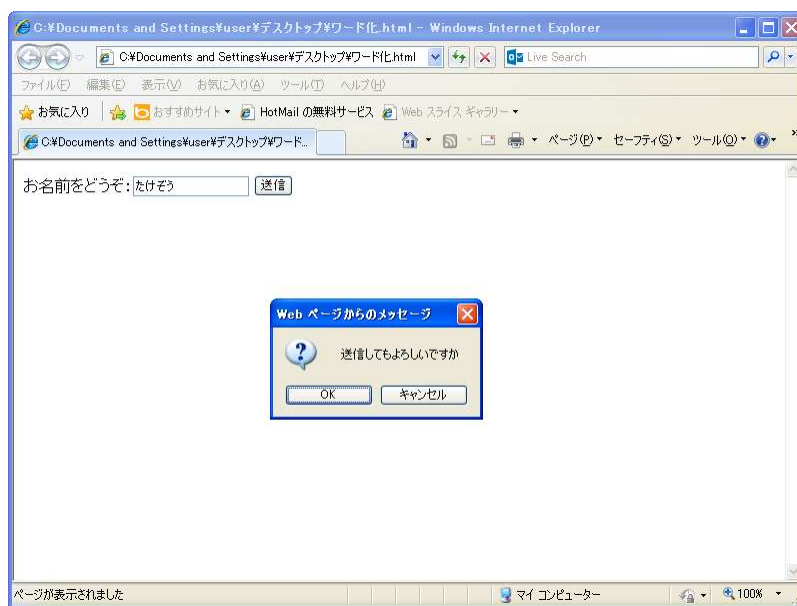
confirm()メソッドは確認用のダイアログを表示します。[OK]が選択された場合は戻り値がtrue、[キャンセル]が選択された場合は戻り値がfalse になります。

confirm()メソッドの利用例を次に示します。form 要素の onsubmit 属性はスクリプトが true を返す場合のみフォームを送信します。したがって、この例では確認ダイアログで[OK]ボタンが押された場合のみフォームが送信されます。

02_confirm()メソッドで確認ダイアログを表示.html

```
<!DOCTYPE html>
<html>
  <body>
    <form name="form" onsubmit="return confirm('送信してもよろしいですか')">
      お名前をどうぞ: <input type="text" name="name"/>
      <input type="submit" value="送信"/>
    </form>
  </body>
</html>
```

実行画面



13. フォーム

HTML では form 要素によって入力フォームを作成できます。入力フォームは Web アプリケーションにおいてユーザからの入力値を取得するために欠かせません。JavaScript を利用すると HTML の入力フォームに以下のような機能を付け加え、Web アプリケーションの使い勝手を向上させることができます。

●入力チェック

Web アプリケーションでは通常サーバサイドで入力チェックを行いますが、JavaScript を利用することで、クライアントサイドでも入力チェックを行うことが可能です。

ただし、JavaScript は Web ブラウザの設定で無効にすることもできるため、JavaScript で入力チェックする場合でもサーバサイドでの入力チェックはあわせて実施する必要があります。

●入力補助

フォーカスが外れたタイミングで入力値をフォーマットしたり、ある項目の入力内容によって別の項目の値を自動的に設定するなど、JavaScript によってユーザの入力を補助します。

もちろん工夫次第で他にも様々な活用方法が考えられます。

1 JavaScript から入力フォームへのアクセス

JavaScript から入力フォームへアクセスするには Document オブジェクトを使用します。例として以下のような入力フォームの場合を考えてみましょう。

```
<form name="loginForm">
  ユーザ ID : <input type="text" name="userId"/><br>
  パスワード : <input type="password" name="password"/>
  <input type="submit" value="ログイン"/>
</form>
```

フォームには Document オブジェクトの forms プロパティ、フォームの入力項目には Form オブジェクトの elements プロパティを利用してアクセスできます。例えば、ユーザ ID のテキストフィールドに値を取得/設定するには次のようにします。

```
// テキストフィールドの値を取得
var userID = document.loginForm.userId.value;
// テキストフィールドに値を設定
document.loginForm.userId.value = 'たろう';
```


2 フォームの入力項目

フォーム内にはテキストフィールドやラジオボタン、チェックボックスなど、様々な入力項目を配置できます。HTML で利用可能なフォームの入力項目には次のようなものがあります。

それぞれの入力項目によってサポートされているプロパティは異なります。入力項目ごとにサポートされているプロパティを示します。

プロパティ	入力項目						
	text password textarea	checkbox	radio	select	option	file	submit reset button
checked		○	○				
defaultChecked		○	○				
defaultSelected					○		
defaultValue	○						
index					○		
length			○	○	○		
name	○	○	○	○		○	○
options				○			
selected					○		
selectedIndex				○			
text					○		
type	○	○	○	○		○	○
value	○	○	○		○	○	○

3 入力値の取得と設定

これまでの例でも見てきたように、JavaScript では入力項目の value プロパティで入力値の取得/設定ができます。

テキストフィールドの値を取得/設定する JavaScript コードの例を次に示します。

テキストフィールド値の取得と設定

```
<form name="form">
  <input type="text" name="name" value="">
</form>
<script>
  // テキストフィールドの値を取得
  var name = document.form.name.value;
  // テキストフィールドに値を設定
  document.form.name.value = 'たろう';
</script>
```

4 入力項目の状態を変更する

フォームの入力項目には状態を変更するためのプロパティが用意されています。disabled プロパティに true を設定すると、

```
<input type="text" name="userId" disabled>
```

HTML の input 要素に disable 属性を指定した場合と同様、その項目は無効となり、入力はもちろんフォーカスを当てることもできなくなります。フォームの送信時にも値は送信されなくなります。

入力項目の状態を変更

```
// userId フィールドを無効にする
document.form.userId.disabled = true;
// userId フィールドを有効にする
document.form.userId.disabled = false;
<中略>
<form name="form">
  <input type="text" name="userId">
</form>
```

14. DOM

DOM(ドキュメントオブジェクトモデル)とは、XML や HTML などのツリー構造を操作するための機能です。Web ブラウザ上で動作する JavaScript は、DOM を使用することで Web ブラウザ上の HTML のすべての要素にアクセスし、変更を加えることができます。

ここでは W3C によって標準化されている DOM(W3C DOM)の利用方法を中心に、DOM を使用した CSS の操作や、W3C DOM が提供するイベントモデルなどについても合わせて説明します。

1 DOM と DOM API

JavaScript から Web ページのコンテンツにアクセスするには Document オブジェクトを使用するという点については前章で説明しました。Document オブジェクトからは DOM という API を利用してドキュメントにアクセスし、変更を加えることができます。

DOM は XML や HTML などの階層構造を持ったドキュメントをツリー状に表現します。XML や HTML の要素間の親子関係をそのままツリーにしたものと考えれば分かりやすいでしょう。

DOM API とは、このツリーにアクセスするための API です。なお、DOM は JavaScript 専用の API というわけではなく、JavaScript 以外の言語でも XML など扱うための機能として DOM API が提供されています。

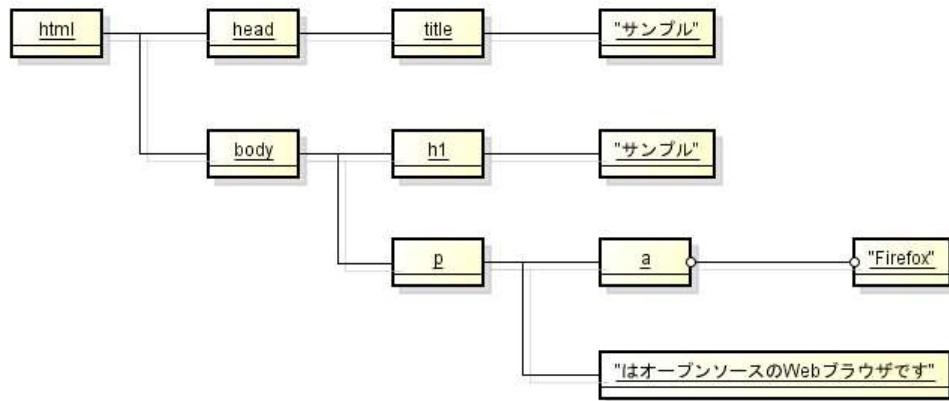
2 W3C DOM

W3C DOM は HTML のすべての要素にアクセスすることが可能となっています。
例えば、次のような HTML があったとします。

階層構造のサンプルコード

```
<!DOCTYPE html>
<html>
  <head>
    <title>サンプル</title>
  </head>
  <body>
    <h1>サンプル</h1>
    <p id="paragraph1">
      <a href="http://mozilla.jp/firefox/">Firefox</a>
      はオープンソースの Web ブラウザです。
    </p>
  </body>
</html>
```

W3C DOM ではこの HTML を次のような階層構造を持ったツリーで表します。



ツリーの各要素を「ノード」と呼びます。ノードは Document オブジェクトから階層構造をたどってアクセスできます

3 Document オブジェクト

ドキュメントの DOM ツリーには Document オブジェクトからアクセスできます。Document オブジェクトは Window オブジェクトの document プロパティとして定義されており、

```
window.document
```

もしくは単純に、

```
document
```

と記述することで参照できます。

Document オブジェクトには「要素名」を指定して要素を検索する `getElementsByTagName()` メソッド、要素の「id 属性」を指定して要素を取得する `getElementById()` メソッド、要素の「name 属性」を指定して要素を取得する `getElementsByName()` メソッドが用意されています。

```
// 要素名を指定し、配列として取得
var h1 = document.getElementsByTagName('h1')[0];
// id 属性の値を指定して要素を取得
var p = document.getElementById('paragraph1');
// name 属性の値を指定し、配列として取得
var empIdField = document.getElementsByName('empId')[0];
```

特に「id 属性」はドキュメント内で一意になることが保証されているため、要素を特定するのに便利です。

取得した要素の属性を取得するには `getAttribute()` メソッドを使います。

例えば、「`Google`」というリンクがある場合、`href` 属性の値を取得するには次のようになります

```
// id 属性が link の要素を取得
var link = document.getElementById('link');
// href 属性の値を取得
var href = link.getAttribute('href');
```

なお、属性は属性名と同名のプロパティでアクセスすることもできます。次のコードはどちらも同じ意味になります。

```
// getAttribute()メソッドで属性値を取得
var href = link.getAttribute('href');
// プロパティで属性値を取得
var href = link.href;
```

●要素の内容を変更する

DOM ツリーを操作することで、Web ページのコンテンツを動的に操作できます。

DOM ツリーを変更する最も簡単な例として、ドキュメント内の特定の `` タグの内容を変えるサンプルを次に示します。このサンプルでは、`p` 要素上でマウスをクリックすると `this.textContent` プロパティに「置換後のテキスト」という文字列が代入されます。イベントハンドラ属性の `this` はその要素自身への参照を表します。

要素の内容を変更

```
<p onclick="replaceText(this, '置換後のテキスト')">
  元のテキスト
</p>
<script>
  function replaceText(element, text) {
    element.textContent = text;
  }
</script>
```

実行結果



innerHTML プロパティを使用することで、任意の HTML を挿入することも可能です。

innerHTML の利用例

```
<p onclick="this.innerHTML = '<ins>置換後のテキスト</ins>'">  
  元のテキスト  
</p>
```

実行結果



属性値の変更には `setAttribute()` メソッドを使用します。以下のコードはボタンをクリックすることで、a 要素の `href` 属性の値を「`http://www.google.co.jp/`」から「`http://www.yahoo.co.jp/`」に切り替えます。

04_setAttribute.html

```
<a id="link" href="http://www.google.co.jp/">Google</a>
<input type="button" value="クリック" onclick="changeLinkTarget()"/>
<script>
  function changeLinkTarget(){
    var link = document.getElementById('link');
    link.setAttribute('href', 'http://www.yahoo.co.jp/');
  }
</script>
```

属性名と同じプロパティに値を代入することで属性値を変更することもできます。

属性値の変更

```
link.setAttribute('href', 'http://www.yahoo.co.jp/');
// 前述のコードと同じ意味
link.href = 'http://www.yahoo.co.jp/';
```


15. jQuery とは

ここまで JavaScript について説明してきましたが、じつは JavaScript は Internet Explorer や Chrome、Firefox などの Web ブラウザによって微妙に仕様が異なり、多くのブラウザをサポートしようとする、ブラウザの種類によって処理を分けるといったテクニックが必要となります。また、JavaScript からの DOM や CSS の操作は、少々煩雑な処理になりがちです。

このような問題を解決するために広く利用されているが、JavaScript ライブラリと呼ばれるものです。

jQuery は最も広く利用されている JavaScript ライブラリの 1 つで、MIT ライセンスで公開されています。CSS セレクタを使用した DOM の検索や更新、Ajax を簡単に実現するための機能を提供しているほか、様々なプラグインで機能を拡張することもできます。

jQuery のような JavaScript ライブラリを利用することで、Web ブラウザ間の差異を吸収しつつ、煩雑な処理を簡単に記述できるようになります。これまでに紹介したサンプルでは、JavaScript の基礎を学ぶことを目的としているためライブラリの使用はしていませんが、実際に JavaScript を用いた Web サイトや Web アプリケーションを開発する際には、これら JavaScript ライブラリを積極的に活用すべきでしょう。

