Web情報システム

— マルチメディア情報通信ソフトウェア

The World Wide Web information distribution system

— Multimedia and communication softwares

工学院大学情報学部情報通信工学科 小林亜樹

2018. 9.10 (Mon.) 版

目次

第I部

Web情報流通システム

第1章

World Wide Web
—マルチメディア情報流通システム

1.1 デジタル情報流通

1.1.1 デジタル

デジタル（digital）は、アナログ（analog）の対語である。analog は、analogy（類似、比喩、相似）と同語源である。電気電子情報系の用語としては「相似形、相似の」という訳語が充てられ、ある信号波形に対して、拡大、縮小の関係にあるような状況を指すといえる。電気回路で、入力信号波形を増幅するような回路をアナログ回路と呼ぶわけである。すなわち、入力信号をそのまま保ったまま変換するような回路を差し、そのため、連続的な量で表現することをアナログと呼ぶ。

一方、デジタルは、「digit の」という意味合いで、元の言葉である digit は、指という意味がある。指で数を数えるときは、必然に整数、0、1、2、となることから、このような数を指す。したがって、「digit の」という意味である digital は、「（整）数の」という意味になる。電気電子情報系の用語としては「計数（型、形）の」という訳語もある。仮に 1 次元の数値データ（スカラ値）を表現することを考えると、アナログでは、どんな数値も表現可能だが、デジタルでは、その系で数えられる数値しか表現することが出来ない。例えば、整数しか扱えないのであれば、小数 2.7 は、桁を丸めて 2、または 3 と表すよりほかない。小数点以下第 2 位まで表す、と決めても、0.625 のような数値を精確に表現することはできないし、$\sqrt{2}, \pi$ のような無理数は絶対に表現できない。

このように、理論的な表現能力はアナログとデジタルでは、アナログが無限の精度を持つため、絶対的に優位である。しかし、世の中はデジタル花盛りである。

- デジタル時計
- デジタル計算機
- デジタル通信網
- デジタル画像
- デジタルカメラ

今や、あえて「デジタル」を接頭しないものも多いほどである。ここまで普及した理由は、工業上、製造上にある。先ほど、アナログは無限の精度を表現しうる、と書いたが、そのとき「理論的な」という注釈を忘れな

かった。理論的な数学上の話であれば、アナログの絶対優位であるが、実際のモノとして扱うには難しい問題があるのである。

ノイズである。

S/N 比という用語を聞いたこともあろう。これは、その系で扱いたいと考えているデータ、すなわち（多くの場合）、電気的信号と、それ以外のノイズ、雑音との比を表す。実際のモノでは、データを電気的（あるいは光）信号で表現するが、どうしても取り扱おうと考えている以外の信号、すなわち、雑音も必然的に混入してしまう。アナログでは、この雑音の取り扱いが難点となり、精度を高めようとすると製造コストも高騰する点が問題である。

これに対して、そもそも一定以下の量を切り捨てて考えるデジタルでは、表現する水準に達しない雑音を考えなくても良いため、回路などの精度追求を一定程度に留めてモノを作っても、設計通りの性能を出しやすい。そのため、大量生産する工業製品の製造コストを下げることが出来るため、デジタル化が進展した。

1.1.2 デジタルメディア

文字それぞれに数値で表現される符号を割り当て、この符号列、すなわち、数値の並びとして文章を表現すれば、それは文章のデジタル化である。メディアとはこの場合、情報の容れ物となる媒体を指す。一般には、光学ディスク（CD、DVD、Blu-ray）、光磁気ディスク、磁気ディスク、磁気テープ、などの蓄積系のモノを指したり、通信ケーブルやシステムとしての通信路（インターネット）、放送、出版（新聞、雑誌）などの社会的存在を指す。しかし、語義に立ち戻れば、文章情報という情報概念そのものをデジタル符号で表現しているということ自体が、情報の容れ物として機能しているため、デジタル符号はデジタルメディアであるといえる。

もう少しわかりやすい例としては、画像情報が挙げられる。本来、光を 2 次元に投影した状態が情報そのものであり、これはアナログ情報である。これを、2 次元平面上をピクセルと呼ばれる区画に分け（空間離散化、量子化）、さらに、ピクセルの光（色）を有限の色番号で表現（色情報の量子化）したものがデジタル画像である。デジタルカメラで撮影すると得られる画像が、現在では最も一般的だろう。画像情報を数値データの並びで表現しており、写真では通常 JPEG 形式と呼ばれる画像圧縮符号化規格が使われている。

動画像情報は、DVD やデジタル放送に用いられる Mpeg2 形式、BD やネットビデオに用いられる H.264（または、MPEG4/AVC）形式、さらには、次世代の動画像符号化形式として知られる H.265（HEVC）形式などが主に用いられている。これらは、動画像の特徴を捉えて、効率的に圧縮を行う動画像圧縮符号化規格である。

このように、情報を表現する論理的主体として符号化方式や規格を考えれば、それもまたメディアである。そこで、デジタル情報を表現する符号化をデジタルメディアと表現することにする。デジタルメディアの特質は、前記のように、テキストだけでなく、画像や動画像（もちろん、音声、音楽なども）など、いわゆるマルチメディアと呼ばれる情報形式のいずれもが、同一のデジタル表現によって表現される点にある。すなわち、0/1 の bit の並びとしてあらゆる情報が表現されるのである。

このデジタルメディア表現では、すべてが 0/1 の bit 列であるがために、これを伝送する通信路や、これを蓄積する蓄積媒体など、情報流通に必要な部品をすべて bit 列を取り扱えるようにさえ設計しておけば、どのような情報でも取り扱うことができることになる。旧来のアナログ表現では、音声と画像は異なる性質を持つ信号であったため、通信ケーブルも記録媒体もそれぞれ専用のものが用いられてきた。これが、デジタルメディアが社会、産業に大きな衝撃を与えている最大の要因であると言えよう。

本稿では、このような影響の大きなデジタルメディアを取り扱う技術のうち、情報通信分野で非常にポピュ

ラーとなっている技術を取り上げる。まず、情報流通させるための通信システムとして、最も普及した情報流通システムとして、World Wide Web (WWW) をとりあげ、そのモデルについて説明する。その後、情報の表現媒体として、HTML、CSS と同時に、それらを制御するプログラミング言語として典型的な JavaScript について順に学んでいく。

1.1.3 マルチメディアによる情報流通

デジタル信号を遠隔地間でやりとりする通信網がデジタル通信網である。送受信端末としては、デジタル計算機（いわゆる普通のコンピュータ）が用いられる。インターネットは、デジタル通信網の一つで、パケット交換技術を用いてマルチメディア情報を転送する。

デジタル通信網を通じた情報伝送では、送受信する双方の端末間で、送受信のための手順をあらかじめ定めておくことで実現される。この手順はプロトコル（protocol）と呼ばれ、その大まかな手順形態に基づき通信モデルが規定される。本稿では、WWW で用いられるプロトコルである HTTP についてその基本モデルについて説明する。

WWW は、事実上インターネット標準である TCP/IP を用いて通信を行う枠組みを用いて利用されており、HTTP と呼ばれるプロトコルを利用する。HTTP は、WWW システムのために開発されたプロトコルであり、通信プロトコルという観点から WWW を説明する場合には、HTTP こそが WWW にほかならない。

1.2 オペレーティングシステム

1.2.1 オペレーティングシステムとは

オペレーティングシステム（Operating system）は、OS と略される場合もあるが、日本語では基本ソフトと訳される。直訳では、操作や運転、演算する系となり、すなわち、コンピュータを運用するための全般を司るものという意味である。実際には、情報処理系の構成要素のうち、多くの処理（を実行するソフトウェア）で共通に利用される機能要素を提供するソフトウェアである。このような共通機能としては、主にデータの入出力や、演算実行順の制御などが挙げられる。ここでは、前者について説明する。

データの入出力には、キーボード、マウス、タブレット、タッチパッドなどの直接利用者による入力機器や、ディスプレイやプリンタなどの直接利用者による出力機器がまず挙げられる。また、情報処理系の基本要素であるデータの蓄積も入出力機能の一環である。現在では、おおむねハードディスク（HDD）、ソリッドステートディスク（SSD）や、同一の管理を行うフラッシュメモリ（USB メモリ、CF カード、SD カード）など、ランダムアクセスでの読み書きが可能な媒体が対象である。

さらに、遠隔地との入出力を司るのがネットワーク入出力である。一般的なものは、Gigabit ethernet と呼ばれる LAN 用インタフェースを介した入出力と、このハードウェアを利用するために実装された、TCP/IP スタックと呼ばれるソフトウェアである。インターネットは、ここで実装された TCP/IP と呼ばれる通信プロトコル（通信手順）を用いて、コンピュータ間をつなぐネットワークとなっている。現代の OS は、これらを用いたインターネットへの接続性を OS 上で実行される各アプリケーションソフトウェアに提供する。これら自体は、本稿の対象外であるので説明は割愛する。

1.2.2 木構造 — Tree structure

木構造は非常に重要なデータ構造である。コンピュータ上で比較的複雑な構造を持った情報を定式化するために、木構造が用いられている例が多い。これは、リスト構造のような平坦（flat）な構造よりも複雑な構造を持っている割に、終端が存在するために、コンピュータ上で扱いやすいためである。

日本文「私は急いで話を聞いた。」は、図 1.1 木構造でその句構造を表現される。

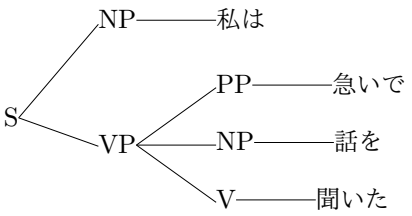


図 1.1　句構造木の例

木構造は、節点（vertex, node）と枝（edge）とから構成される。全体を束ねる節点を根（root）と呼ぶ。1 本の枝の両端には 1 つずつ節点が接続され、この 2 節点の関係を親子関係と呼ぶ。根に近い節点を親（parent）、根から遠い節点を子（child）と呼ぶ。子を一つも持たない節点を葉（leaf）と呼ぶ。

図 1.1 での根は、左端の S [*1]である。S は、NP[*2]、VP[*3]の 2 つから構成されることが読み取れる。それぞれの節点は、さらに右方の部分木（sub-tree）のような構造を持つことになる。句構造では、木構造の葉のみに実際の文字列が割り当てられ、それ以外は、文の構造を表す「句」の種類によって、構造が示されていることがわかる。

木構造は、住所などの名称、場合の数を数える際の樹形図、ファイルシステム、意志決定木など、極めて多様な局面での構造を示すことに用いられている。HTML、XHTML も木構造を表現するためのマルチメディア符号化規格であるといえる。

1.3 World Wide Web

1.3.1 歴史

1991 年、イギリス人研究者 Tim Berner-Lee は、スイスの CERN で研究を行っていた。研究では、世界中の研究者らと実験データをやりとりしたり、実験結果の議論などを行う必要があった。しかし、そのための適切なシステム（プロトコル）がないと感じた Tim は、自分で開発した。それが、プロトコル HTTP とその通信手順上で通信すべき内容を記述する言語 HTML である。

WWW は、HTML に画像を貼り込む能力と、それを手軽に見ることの出来るソフトウェアである Web browser を得て、急速に普及していく。HTML は、機能の拡張などに伴いバージョンアップを繰り返し、現在の最新バージョンは HTML 5 である。HTML5 の直近のバージョンに対する主な更新点は、よりインタラ

[*1] ちなみにこれは、sentence の頭文字である。sentence の意味は調べよ。
[*2] noun phrase
[*3] verb phrase

クティブなコンテンツを作成できる規格であることで、Web 上で動作するアプリケーションを構成できることを目指して設計されている。

この発展の過程で、木構造データを表現する方式の一つで、拡張性の高さを特徴とする XML が登場し、HTML をその上で再定義しようとする動きもあった。HTML4.01 の標準化の後、XHTML1.0 と呼ばれるバージョンの普及が進んだが、これも現在では HTML5 に取って代わられている。XHTML としてのバージョンは、1.1 が最新であるが、バージョン 1.0 からの置き換えが進んだというようなことはない。現在でも XHTML が選択されることはあるが、これは、XML の特徴である機械解釈されることを要件とするような文書などを目的とするためと考えられる。

2014 年 10 月 28 日、HTML5[1] が標準規格となるべく “Recommendation”（勧告）と呼ばれる状態として公開された。この時点での WWW での標準は、名実ともに HTML5 にあると言って良い。HTML5 では、動画や音楽といったマルチメディア情報の取り扱いも標準化に含められ、また、文書構造上の表現力もこれまで以上に強化されたものとなっており、商用 Web サイトの構築を意識したものとなっている。

以上をまとめると、現在の Web においては、過去の標準にも目配りをしつつも、HTML5 を採用することが望ましい。

1.4 HTML の表現する内容

1.4.1 概要

HTML(HyperText Markup Language)[*4]は、WWW のためのコンテンツ記述言語である。また、XHTML(Extensible HyperText Markup Language)[3] は、XML(Extensible Markup Language) というコンピュータ情報通信用に設計された、情報通信のための言語を定義するための言語上で、HTML とほぼ同内容を定義して規定された言語である。XML[*5]の規則に従わなければならないため、HTML とは一部異なる点も含まれている。

これらは WWW システムにおいては、HTTP により転送される。

Markup とは、「注釈」の意で、単純なテキストファイルの部分毎に、決められた「注釈」を示すための文法が用意されていると理解すると良い。

HTML は、SGML に含まれることになっていて、XML と共に、木構造となるデータ構造を表現するための符号化法の一つであり、現在では広く利用されるに至った標準技術である。XHTML は XML によって定義されており、同様に木構造のデータ構造を表現する。

1.5 Web 情報システム

Web 情報システムというときは、情報配信システムとしての WWW（world wide web）とそれを支える TCP/IP 網としてのインターネット（The internet）、また、実際の情報処理を司る、Web サーバやクライアント（Web ブラウザソフトウェア）を含めた全体を指す。したがって、この動作全体を知るためには、関連する規格、標準ばかりでなく、原理、文化（作法、しきたり）など、幅広い知識が要求される。ここでは、全体の概要をつかむために、配信情報を示すコンテンツ、配送のためのプロトコル、それらを扱う通信の両端に位

[*4] ちなみに HTML は、SGML ベースの言語である。

[*5] ちなみに XML は、SGML のサブセットになるよう規定されているが、SGML のサブセットとは規定されていない。

置する、サーバ、クライアントについて説明する。その後、これらを結ぶ上で重要な役割を果たす URL について説明する。

なお、通信システムの説明であり、多くはインターネット上で運用され、また、TCP/IP 上で用いられることが一般的なシステムである。そのため、IP、TCP、DNS などについては理解していることを前提とする。

1.5.1 コンテンツ

配信したい情報を HTML、css、Javascript、Adobe flash、Portable document format などで記述した電子データのことである。画像データであれば、Jpeg、PNG など、動画像データであれば、Mpeg2、Mpeg4 といった圧縮データ形式が用いられることが多い。

情報配信をしたい側（情報発信者）が、自身の管理下にある Web サーバに配信したいコンテンツを保存し、配信できる状態にしておくと、情報を受けたい側（情報受信者）が、Web ブラウザを用いて、通信プロトコルを通じてそのコンテンツファイルにアクセスする。その結果、情報受信者の Web ブラウザにコンテンツデータが転送され、情報受信者は見たい情報を閲覧することができる。

Web ブラウザにおいて、ひとまとまりのコンテンツ（一般に 1 ページ）として見えるコンテンツは、一般に多数のデータファイルによって構成されている。したがって、全部のデータを Web ブラウザが受け取らないと、その内容を完全に表示することはできない。

多数のコンテンツデータを束ねる主たるデータ形式が、HTML である。HTML は、HyperText Markup Language の略である。

1.5.2 通信プロトコル

Web 情報配信システムで用いられる通信プロトコルは HTTP である。HTTP は、HyperText Transfer Protocol の略であり、無理に日本語訳すれば、「超越文書転送手順」となる。コンテンツとして主として HTML で記述されたハイパーテキストを想定し、これをコンピュータ間で転送するための通信プロトコルである。

HTTP では、Web ブラウザを実行しているコンピュータである、クライアント（コンピュータ）が、Web サーバを実行しているサーバ（コンピュータ）に対して、まず、TCP/IP プロトコルで接続する。サーバ側では、Web サーバプロセスがこの接続を待ち構えており、接続を受容する。

接続すると、この接続（connection）を通じてクライアントは、サーバに対して、欲しいコンテンツデータの識別子を送信し、そのコンテンツデータを転送してくるよう要求（request）する。この要求の応答（response）として、Web サーバは、クライアントに対して、同接続を通じてコンテンツデータを転送する。

通常、多数のコンテンツデータによってひとまとまりのコンテンツは構成されているから、これらを HTTP を通じて連続的に要求、転送することになる。HTTP のバージョン 1.0 では、一度の HTTP 接続、すなわち、TCP/IP による接続〜HTTP による要求-応答そして切断までを通じて 1 つのコンテンツデータしか転送できなかった。このため、主に TCP 接続のセットアップ時に必要なオーバヘッド（間接的な処理に係る負荷）が大きく、転送時間や、Web サーバ負荷の増大を招いていた。そこで、現在主として用いられている HTTP のバージョン 1.1 では、一度確立した TCP/IP のセッション（session; connection と同義）を使って、多数の連続的なコンテンツ要求と応答ができるように改良されている。しかし、論理的に別々のコンテンツデータを複数個組み合わせてひとまとまりのコンテンツになるという設計自体は変わることはなく、多数の部品の転送

を待つ間、部分的にしかコンテンツ全体の様子はわからない。

1.5.3 Web サーバ

HTTP による接続を待ち受けているコンピュータ上の実行主体であるプロセス、あるいは、このプロセスを実行しているコンピュータ自体を指す。HTTP におけるサーバ側機能性を提供する主体、と捉えると Web サーバプロセスを指すと解するとわかりやすい。

HTTP プロトコルによる接続を待ち受けている Web サーバは、接続が確立した後、クライアントからのコンテンツ要求を待つ。コンテンツ要求を受信すると、その内容を解析する。

コンテンツ要求には様々な情報が含まれるが、最も本質的なのは、どのコンテンツを要求するのか、を表している識別子である。これは、通常、サーバ上のコンテンツデータファイルへのパスとして表現される[*6]。例えば、`/dir1/dir2/name3.html` などと表記され、これは見慣れたファイルパスの記法と同一である。

この識別子に対応するデータファイルを同一の HTTP 接続を通じてクライアントへ転送する。転送は、TCP などの 8bit transparent な通信路を仮定しており、そのままの（binary な）形式で転送される。

Web サーバ自身は、要求されたコンテンツを返す、という単純な仕事しかしない点に注意しよう。複数のコンテンツ間に張られた hyperlink（Web のリンク）について、Web サーバは基本的に関知しない。

1.5.4 Web ブラウザ

Web 情報配信システムにおいて、Web ブラウザこそが、Web らしい外観や機能性を提供しているとも言える。Web ブラウザは、HTTP におけるクライアントとしての機能性を通信システムとして提供すると共に、利用者に対して、HTML や css などを解釈した画面提示や、Javascript などの制御実行系といった機能性を提供する。

Web ブラウザは、HTTP でのクライアントコンピュータ上で実行されるプロセスである。HTTP のクライアントとして、接続を主導し、利用者要求を解釈した結果としてのコンテンツ識別子をサーバ側へ TCP/IP 接続を通じて送信することで、コンテンツを要求する。サーバ側応答として返されたコンテンツを受信し、それぞれの記述形式に応じて画面上に描画することで、利用者に対してのインタフェースを提供する。

1.5.5 URL

インターネットに接続された多数のホスト（host）で稼働する Web サーバに大量の Web コンテンツが配信可能な状態で置かれている。利用者は、Web ブラウザの助けを借りるなどしながら、これらのコンテンツ要求を行うが、このとき、コンテンツ同士を識別する識別子（identifier）が必要である。WWW では、URL（Uniform Resource Locator; 統一資源位置指定子）が用いられる。

URL は、例えば、`http://www.example.com/path/to/file.html` のような形式となる。ここで、`http` の部分は、情報を取得するためのスキームを示す。この場合は HTTP プロトコルによってリソース（resource; （情報）資源）にアクセス（access）することを示している。

`://` は区切りで、その後の `/` までの部分がホスト部である。インターネット上のホストを示し、DNS 名

[*6] Web サーバ上の実体として、OS が提供するファイルシステムと関連づけなければならない事実はどこにもない点に注意。しかし、実装上の性能、管理上のわかりやすさなどから、少なくとも部分的に OS の提供するディレクトリツリーとこの識別子を対応付けて運用されている例がほとんどである。

(Full Qualified Domain Name; FQDN)、または、IP アドレスで記述する。

/ 以降の /path/to/file.html は、ホスト内での情報位置を示す path である。絶対パスと同様な書式であるため、ディレクトリツリー上の位置と対応付けて管理されることが多い。しかし、そのようにしなければならないというわけではなく、Web サーバで提供する情報やサービスの種類、運用方針などによって異なる管理方法が用いられている。

http://otherhost.example.com:80/path/to/file/ のような書式を見ることもある。ここで、ホスト名に :80 という表記が付け加わっている。これは、ホスト名の一部ではなく、そのホストに HTTP で接続する際に用いる、TCP のポート番号である。HTTP の場合、これを省略すると 80 番ポート（80/tcp）を指定したものと見なされる。このような省略時のポートを default port と呼ぶ。

path が、スラッシュ（/）で終わっている。これは、ディレクトリツリーでは、ディレクトリ自体を示す書式であった。Web では、転送すべきコンテンツ（情報）を指定することが URL の役割である。そこで、ディレクトリが指定された際には、Web サーバ側で適当なコンテンツが指定されたと解釈する仕組みが採用されている。

一般には、当該ディレクトリ内で Web サーバに設定されているファイル名を持つファイルを転送すべきコンテンツであるとして動作する。これは、URL の後に、そのファイル名を補遺しているかのように振る舞うということである。

補遺されるファイル名は、Web サーバ側の事情で自由に設定可能であるし、ディレクトリ毎に異なっていても問題ない。しかし、多くの場合、index.html、index.htm、default.htm などが補遺される設定で運用されている事例が多い。

ほかにも異なる書式の URL を見る機会もある。しかし、それは使われることが稀であったり、本稿で扱う範囲を超える高度な処理のための書式であるので、説明は他に譲る。

第2章

Webによる情報発信の体験

2.1 HTMLの基礎

2.1.1 Markup

HTML(HyperText Markup Language)、XHTML（Extensible HyperText Markup Language）は、マークアップ言語である。マークアップとは、文字列としての存在である単純なテキスト（plain text）に対して、その部分に特定の意味合いを持たせるための指示を指す。HTML は、そのような指示の規定集であり、指示毎に意味合いが定義されている。

2.1.2 書式

HTML は、SGML というマークアップ言語を定義するための言語に影響を受け、一部のバージョンでは、SGML 準拠となるように規定された[*1]。そのため、その書式は SGML から強い影響を受けている。本文テキストをマークアップするためには、マークアップ部分が本文テキストとは異なることを示す必要がある。

そこで HTML では、テキストのうち一部が HTML の要素であることを <> 括弧で示されるタグ (tag) を用いる。具体的には、HTML 要素の部分であることを、その始まりと終わりにそれぞれ、開始タグ、終了タグ、と呼ばれるタグを置くことで示す。マークアップはどのような種類の指示であるかを示す要素名によって識別される。要素名は、タグを表す山括弧内に記述し、これを開始タグと呼ぶ。要素の終了は、要素名の前に“/”(スラッシュ) をつけたタグ（終了タグ）である。開始タグと終了タグに挟まれた部分がその要素の内容である。

要素の内容がない要素、すなわち、マークアップすべきテキストの存在しない要素も存在する。この場合、終了タグは存在できず、開始タグの書式のみをもって要素を構成する。例えば、テキストコンテンツ中の改行を表現する `br` 要素は、何らかのテキストに意味を持たせる指示をするような性質の要素ではない。したがって、終了タグは存在しないものとして定義されている。また、テキストの途中に画像を挿入する `img` 要素もまた、テキストに対する指示ではない。したがって、これもまた、終了タグは存在できない。

[*1] XHTML は、XML である。また、XHTML は、HTML と同様の記法となるように設計された。

```
<h1>見出しレベル 1 を示します。</h1>
<p>これは一つの段落を示します。</p>
<p>要素の終了は、/を要素名の前。<br>
改行は br 要素です。<br>
このように記述します。</p>
<p>本文中に画像を差し込むこともできます。
<img␣src="some_picture.jpg"␣alt="HTML 中の画像例">
このような要素をタグとして記述すると、
当該画像ファイルがブラウザ側で読み込まれ，表示されます。
</p>
```

2.1.3 空の HTML5 文書

厳密（規格上）な空の HTML5 文書は、

```
<!DOCTYPE html>
<title></title>
```

である。

実際には、言語や文字コード、ページタイトルなどと共に、基本的な要素を記述するべきであると考えられるので、これについては後述する

2.2 Web ページ作成体験

実際に Web ページを作成してみる。

2.2.1 空の HTML ファイル

内容をほぼ含まない HTML5 文書をリスト 2.1 に示す。

コード 2.1 空の HTML5

```
1  <!DOCTYPE html>
2  <html lang="ja">
3  <head>
4  <meta charset="utf-8">
5  <title></title>
6  </head>
7  <body>
8  <p></p>
9  </body>
10 </html>
```

このファイルを基に、必要な記述を足していくことで学習をすすめる。

たいが、単に Web ブラウザとして閲覧すると、真っ白なページとして見えてしまい、成功しているかわか

りにくいため、自分の名前を追記したリスト 2.2 を入力して、'ch02.html' というファイル名で適当なディレクトリに保存しておこう。このファイルに、必要な変更を加え、記述を足していくことで目的とする HTML ファイルを作成することにする。

コード 2.2　確認できる事実上空の HTML5

```
1  <!DOCTYPE html>
2  <html lang="ja">
3  <head>
4  <meta charset="utf-8">
5  <title>タイトルをここで指定します</title>
6  </head>
7  <body>
8  <p>自分の名前を書いてみましょう。</p>
9  </body>
10 </html>
```

正しく、設定された Web サーバ上の正しいディレクトリに、正しい URL を用いて、正しく設定された IP 網を経由すれば、使用した Web ブラウザ上に、自分の名前だけが表示された事実上、空のページ内容が表示（描画）される。

2.3　基本的な要素

2.3.1　HTML5 要素紹介

html　HTML の root 要素。
head　基本的情報。基本的に画面表示されない。
body　本文。基本的に画面表示される。
p　段落 (paragraph) を示す。
section　節（セクション）を示す。
h1　第 1 レベルの見出し要素。h2,h3,... と 6 レベルまである。
ul, ol　箇条書きを示す。ul は番号なし、ol は番号付き。block 内で li 要素により、項目を示す。
dl　定義リストを示し、block 内で、dt で項目を、dd で項目内容をしめす。
blockquote　引用を示す block。
q　引用を示す。
br　テキストコンテンツ中の改行を示す。

2.3.2　要素の記述 —タグ

要素は、テキスト上では、開始タグと終了タグとで囲まれた領域としてマークアップされる。タグは、<と> の両記号で囲まれた特定の文字列である。<記号直後に現れる文字列が要素の名前を示しており、これは開始タグである。</後に要素の名前が現れるのが終了タグである。

第II部

HTML
―マルチメディア符号化

第3章

HTMLによる情報の表現

3.1 HTML5

3.1.1 HTML5 規格策定

2014年10月28日、HTML5は、ワールド・ワイド・ウェブ・コンソーシアム（W3C）より、勧告として公開 [1] された。仕様策定が本格化してから6年あまり、当初の発端からは10年ほどかかったこととなるが、これによって名実ともにHTML5はWWWの世界での標準規格となった。

*1

2017年12月14日、現時点で最新の勧告であるHTML5.2が勧告された [1]。勧告文書はHTML5と同一のURL、すなわち、置き換え更新されている。HTML5と言えば、現時点ではHTML5.2のことを指し、文法の骨格をはじめ大部分は互換性が保たれていると言って良い。

多くのWebブラウザではHTML 5に基づく表示や動作を実装してきている。しかし、このように仕様自体が進化し続けていくHTML5という環境では、特定のバージョンに完全に準拠していることを期待できないのが、Web系の世界らしさである。

HTML5では、オーディオやビデオなどの典型的なマルチメディア情報を、HTML本体として正式に取り扱えるようになっている。また、gmailなどに代表される、Webブラウザを通じて使う、Webアプリケーションと呼ばれるタイプのアプリケーションソフトの実装環境のために、さまざまな機能拡張が施されている。さらに、HTML5は、これまでのHTML（XHTML）策定の際の泣き所であった、ブラウザ間の互換性がない、という点を低減するために、各ブラウザベンダーからもメンバーの参加を得て、細かな挙動に至るまで互換性を最大限に確保した規格とするべく議論を続けてきた。しかしながら、各ベンダー間の思惑や事業上の対立なども影響したと見られ、事前の宣伝ほどには互換性が得られていない状況ではある。

それでも、HTML5は今後のWebのスタンダードとなることが約束されているとは言え、一般的なWebコンテンツ開発においての第一選択肢はHTML5であろう。

*1 2012年12月17日、HTML5は規格策定の長い道のりにおける一里塚を迎えた。この日、HTML5は、“Candidate recommendtion” すなわち、正式な勧告の候補となる版を公開した。事実上、これで主要な仕様は確定したものとして扱っても実務上は大きな問題が生じないはずである。

3.1.2 HTML5.2

W3C と WHATWG（Web Hypertext Application Technology Working Group）[*2]とは 1 年に 1 回程度の頻度で、継続的に HTML5 を改善していくことを表明しており、2016 年 11 月 1 日に HTML5.1 の勧告 (recommendation)、さらに、2017 年 12 月 14 日に HTML5.2 の勧告を得た。先に述べたとおり、HTML5 の URL は、現行最新版の HTML5.2 の内容を指しており、順次仕様が更新されていく、という HTML5 の精神を体現している。

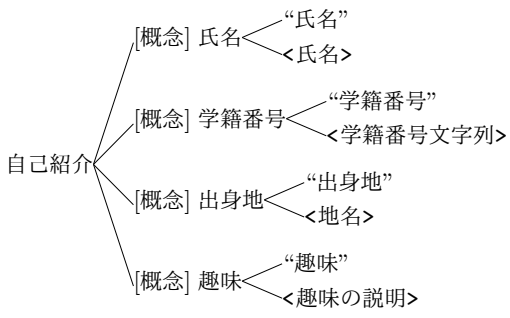
HTML5.1 では、HTML5 勧告前後に急速に普及したスマートフォンなどの小画面媒体、レスポンシブデザインへの対応強化や、Web アプリケーション構築で必要となる入力フォーム、コンテンツ操作のためのイベント対応強化などが図られた。HTML5.2 では、種々の細かい改訂がなされており、一部の要素の廃止も含まれている。

HTML だけでは実現できない機能は、HTML5 ファミリである css や Javascript を用いて、各 Web サイト開発者が実装してきた。それが HTML によって標準化がなされれば、利用者にとって挙動が統一化されわかりやすくなる効果と、実行プログラム（バイナリ）がブラウザ内で統合されることによる実行メモリの低減や実行速度の向上が見込まれる。

3.2 情報の表現

情報には、概念それ自体としての存在がある。ここでは、例として「自己紹介」を取り上げる。自己紹介は、氏名、学籍番号、性別、出身地、趣味、などの自己を他人に紹介するために適当と考えられる概念の集合を他人に提示することだと捉えられる。概念を他人に提示するためには、これを表現に変換しなければならない。私達は、通常の変換先表現として日本語を用いている。

情報通信網で扱うためには、コンピュータが認識でき、保存でき、通信路を介して送信できることが必要である。そこで、この一般的な表現である日本語を、コンピュータネットワークで扱えるように変換する。ここで、気をつけないといけないことは、日本語で表現された内容（概念）は、すでに、日本語での表現に変換された後であり、概念それ自体を最も適切に表現しているとは限らない点である。



例えば、このような木構造で表現される。この木構造データを通信路を介して転送しようとすると、通信路

[*2] Web Hypertext Application Technology Working Group: 2004 年の W3C ワークショップ後，Web 系企業の個人らによって設立された．当時 XHTML 推進の W3C とは立場を異とする現場の HTML ニーズに沿った規格策定を目指したグループ．結果的に，HTML5 では WHATWG での議論を受け入れた形で W3C が標準化した．

は 0/1 列、すなわち、1 次元のディジタルデータしか扱えないから、ディジタルによる表現に変換する必要がある。

HTML は、このような木構造でデータを取り扱う形になっている。これは、HTML が考案される過程で、木構造によるデータ管理を行っていた SGML の影響を強く受けたからとも考えられるし、多くの複雑なデータ構造を木構造ならばそれなりに表現でき、かつコンピュータによる取り扱いも容易であるからとも考えられる。HTML の規格策定において、一時主流となりかけた XHTML の基となっている XML もまた、データ構造として木構造を表現している。

情報（概念）を木構造というデータ構造で表現した HTML を、情報通信網で取り扱うためには、さらに符号化を行わなければならない。なぜなら、この段階では、HTML を抽象的な “文字列” として記述できるだけだからである。

3.2.1 テキストの符号化

文字列はテキストと呼ばれ、最も基本的なデータ表現である。その符号化手法は多々あるが、Windows 上では、cp932 と呼ばれる Shift JIS [*3]を拡張した規格が用いられてきた。マイクロソフト社のオペレーションシステムである Microsoft Windows 3.1 において成立したため、IANA での登録名は、Windows-31J である。

e-mail では、いわゆる JIS コードと呼ばれる、ISO-2022-JP が用いられる。UNIX では、EUC-JP(Extende UNIX Code for Japanese) が多く用いられてきた。これは、符号語に割り当てるコードの関係上、日本語文字がプログラミング言語の処理系の実装と干渉しにくいためである。

しかし、現在の標準的な符号化は utf-8 である。これは、UNICODE 系の符号化方式[*4] であり、多くの国々で用いられる。Windows においても、内部的には utf-8 を基本とするようになったし、現在の MAC は linux ベースであるが、これも utf-8 を用いている。

テキスト情報の符号化では、もう一つ重大な問題がある。それは、改行コードである。改行という概念も、何らかのコードで表現しなければならないことは自明だが、この表現にも流儀が存在する。MS-DOS 時代を含め、Windows 系は `CR`[*5]`LF`[*6]の 2 つの制御文字の連続で改行を表現する。UNIX 系は、`LF` の制御文字が改行を表現する。Mac 系は、旧来は `CR` であったが、現在の実装は、UNIX の上に設計されているため `LF` である。

インターネットは、UNIX 系と共に普及したが、通信路上の改行コードは、`CR LF` の 2 文字を用いる例が多い。しかし、サーバ内でのプログラミング言語処理系では、`LF` でないと誤動作や意図しない動作を行うこともある。したがって、そのファイルがどのようなプログラムによって取り扱われるのかを、厳密に意識して改行コードを決定する必要がある。

3.2.2 画像の符号化

WWW でテキストと共に最も良く使われるデータに、静止画がある。通常単に画像と呼ぶ。画像もまた符号化の必要があり、最も良く使われている符号化方式は、JPEG、PNG、GIF と呼ばれる方式である。

動画像は、MPEG が用いられることが多い。現在主流となっているのは、MPEG4/AVC 形式である。H.265/HEVC と呼ばれる、より新しい符号化方式も徐々に普及が始まっている。一般に新しく規格化された

[*3] Shift JIS は、NTT docomo が開発し、世界の携帯端末での情報化の牽引役となった i-mode での標準文字符号化法でもあった。
[*4] ここでは、符号化文字集合、文字符号化方式などの煩雑な議論は他の成書に譲る。
[*5] Carriage return: カーソル位置を行頭に戻す役割を表す制御文字の一種。
[*6] Line feed: 1 行分送ること、すなわち、カーソルを次行に移動させる制御文字の一種。

符号化方式の方が圧縮率などで勝っているが、古い機材等では再生できない問題がある。そのため、利用者として想定する範囲を考慮して符号化方式を選択する。

いずれも、（動）画像データというマルチメディア情報をディジタルデータとして表現しているという意味において、他の符号化と同様に位置付けられる。

3.3 要素

HTML5.2 での要素は、コンテンツモデルによって 7 種類に大分類される。これらの分類にしたがって、要素の大まかな振る舞い規則が存在する。一つの HTML 要素は複数の分類に属することができる。図 3.1 [*7]に見るような包含関係で、各要素は定義されている。

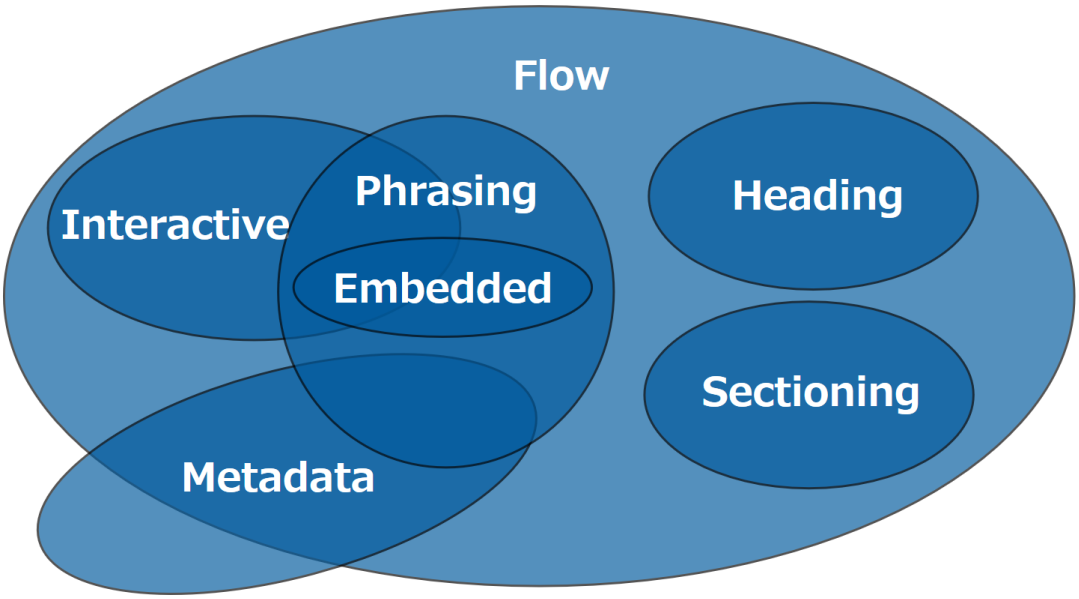


図 3.1 HTML5 コンテントモデルの分類

3.3.1 コンテンツモデル

なお、通常のテキストはなんらかの要素の内容であるが、phrasing content に属すると見做される。

Metadata content 文書のメタデータを表現するための要素が属する。

代表的な要素：*link*, *meta*, *script*, *style*, *title*

Flow content 行組されていく一般的なテキストやオブジェクトを表現するための要素が属する。

代表的な要素： *a*, *aside*, *br*, *canvas*, *div*, *dl*, *h1*, *img*, *input*, *ol*, *p*, *q*, *span*, *table*, *ul*

Sectioning content 論理的な文書の部分を表現するための要素が属する。

要素： *article*, *aside*, *nav*, *section*

Heading content 見出しを表現する要素が属する。

[*7] W3c の HTML5 勧告書内 `https://www.w3.org/TR/html52/dom.html#kinds-of-content` で、インタラクティブな図を見ることができる。

代表的な要素：*h1*, *hgroup*

Phrasing content　段落の一部分を構成する、句を表現する要素が属する。一般のテキストはこの phrasing content として取り扱われる。

代表的な要素： *a*, *br*, *em*, *img*, *input*, *q*, *span*, *strong*

Embedded content　埋め込まれたコンテンツを表現する要素が属する。

代表的な要素： *canvas*, *img*, *math*, *object*, *video*

Interactive content　利用者とやり取りするための対話コンテンツを表現する要素が属する。

代表的な要素： *a*, *buton*, *select*, *textarea*

ほとんどの要素は `body` 要素内に配置可能な flow コンテンツカテゴリに属する。そのうち、一部の要素は phrasing コンテンツに属しており、この phrasing コンテンツに属する要素のみを自身の内側に配置可能な要素が定義されており、これが要素間の入れ子関係の可能性を規程する第 1 の原則となっている。

要素の振る舞いに影響するこれらのコンテンツモデルは、Palpable content となる要素の条件のように、組版上の実務に合わせて特定の条件下でのみ特定のモデルに属するよう規定されている場合があるなど、かなり複雑である。ここでは、コンテンツモデルの存在と要素の分類、振る舞いに影響することの理解までを目的として、概要と代表的な要素についてのみ記した。

3.3.2　要素の入れ子

HTML は要素の中に別の要素を入れる入れ子構造をとる。このとき、ある要素の中に入れられる要素は要素ごとに定められている。おおむねの規則としては、phrasing content に属する要素内には、phrasing content を配置することができる。一方、phrasing content でない flog content に属する要素内には、flow content に属する要素を配置することができる。

要素の役割を文脈として典型的なものは想起できるが、すべての組み合わせを記憶しておくことは現実的ではないといえる。

`Body` 要素は、最上位区分要素（sectioning root）に位置付けられ、`html` 要素直下の 2 番目の要素（1 番目は `head` 要素）として使用可能であると定義されている。`Body` 要素内には、flow content を配置できる。flow content に属する代表的な要素には、

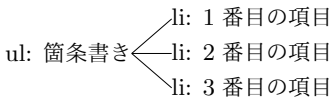
> p, section, h1, h2, ..., h6, div, ul, ol, dl, pre, hr, blockquote, table, form など

があった。

3.3.3　入れ子構造

全ての文書要素は、（木構造的に）`html` 要素の下に位置づけられるため、要素間の入れ子構造になっていることがわかる。しかし、このような形式的な入れ子ではなく、実際の論理構造として入れ子構造が利用される典型例として箇条書きを見てみる。

箇条書きは、番号なしの `ul` と番号付きの `ol`、また、類似の構造として `dl` が用意されている。もっとも単純な `ul` について見ると、箇条書き要素 `ul` は複数の項目を並べる。そこで、箇条書き要素の下に複数の項目がぶら下がる、次のような木構造が考えられる。



各項目は、項目内容として文字列を含むため、それらを項目としてマークアップする。HTML では、li 要素が項目を示すマークアップである。

したがって、

```
<li>1 番目の項目。</li>
<li>2 番目の項目。</li>
<li>3 番目の項目。</li>
```

のように記述することになる。

しかし、このままでは箇条書きとしてのまとまりを示すことができない[*8]。そこで、これら全体を内容として含む箇条書きを表す要素が必要であり、それが ul 要素などである。すると、全体としてこの箇条書きは、

```
<ul>
<li>1 番目の項目。</li>
<li>2 番目の項目。</li>
<li>3 番目の項目。</li>
</ul>
```

のように記述することになる。

箇条書きの項目の中に別の箇条書きを入れることもできる。木構造の段数（深さ）が増え、入れ子になる様子を描いた上で、実際に HTML で記述してみよう。

3.3.4 HTML のマークアップ

このように、HTML では、文書の論理的構造をマークアップで示す。したがって、自己紹介の場合、紹介の各要素（氏名、性別、学籍番号など）を h2 でそれぞれマークアップすれば、見出しの一覧だけで、文書の概要を知ることができる。これが HTML での「良い」マークアップとされる。

3.4 Validation

HTML が HTML 文書として妥当であるかどうか、検証することができる。このような妥当性検証を validation と呼ぶ。コンピュータ（計算機）が解釈するための規格（書式）である HTML であるから、コンピュータが解釈するのに規格通りでないと解釈ができなくなる。そのため、妥当な状態であるということは、HTML が成り立つかどうかの条件であるといえる。すなわち、妥当な状態でなければ、その文書は HTML としては存在していないとされても仕方がない[*9]。

[*8] 試しに 2 つの独立した箇条書きが連続している様子を考えてみよう。

[*9] 実際には HTML を取り扱う主な主体である Web ブラウザは、歴史的経緯もあり、相当間違った HTML でもそれなりに解釈してくれる。

HTML5 の規格に沿って、かつ、XML としても規格に沿った状態で記述することもできる。これを XHTML5 と呼ぶと、これは XML 文書でなくてはならない。XML は、HTML よりも厳密に解釈されるための規則を持っている。これは、XML がコンピュータ間のデータ転送を主目的として開発された経緯による。XML 文書で、ある文書が XML として妥当である、すなわち、XML の文法規則を完全に満たしているとき、その文書のことを well-formed である、と呼び、日本語では「整形式」と言う。XHTML validator を使うと、このような検証をコンピュータが行ってくれ、不正な部分を指摘してくれる。HTML でもこの validator を利用することで、利便性を考慮した解釈を行う Web ブラウザに任せるよりも適切に規格に則っているかを判定できる。

The W3C Markup Validation Service[5] は、ASP(Application service provider) 型のサービスであり、Saas(Service as a Service) でもある。インターネットアクセス可能な Web サーバではない、社内、学内や実験的なネットワーク上のサーバで運用している場合には、URI を入力しての検証はできない[*10]。直接入力などを用いれば確認できる。整形式であることは、XML 文書として最低限の検証である。通信網で扱うデータ形式であることを考えれば、XML 文書として妥当でなければ、通信先でどのように扱って良いかわからないためである。

気をつけなければいけないのは、この検証は文法上の規則を検証しているだけであり、先に述べたような、見出しが適切に設けられているか、などは解釈不可能である点である。

3.5 HTML の代表的要素

HTML5 の要素名などは、すべて英小文字である。

3.5.1 `html` 要素

`html` 要素は、HTML 文書に直接書く唯一の HTML の要素である。属性[*11]に指定するものは事実上決まっており、常に同一の文字列を記述する。

`html` 要素の直接下に位置することができる要素は、次の 2 つだけである。

`head` 文書の基本的情報を記述する部分。
`body` 文書の本文内容を記述する部分。

また、この 2 要素は、必ずこの順序で記述されることが要請されている[*12]。

3.5.2 head 要素

`head` 要素では、文書の表題、文書間の関係、その他のリソースなどを指し示す。すなわち、文書本文ではなく、文書に関する情報を示す部分である。従って、`head` 要素下におかれる要素は、これら文書のメタデータを示すような要素が定義されている。

`title` 文書の表題を示す。

[*10] 理由を考えよ。
[*11] 属性は、タグ内で半角空白で分離して記述していく。
[*12] 勧告では、これら要素の記述自体の省略についても規定されている。本書では学習目的のため、これら省略については扱わない。

link　他の文書との関連を示す。詳細は 3.5.2 節を参照。
meta　文書に関するその他の情報を記述する。

link 要素

文書から見た、他の文書との関係を記述する。関係を rel 属性に、文書 URL を href 属性に、利用者に対する説明として title 属性を記述する。例えば、文書の次のページを示すためには、

```
<link rel="next" href="page002.html" title="次ページ">
```

のようにする。主な、rel 属性値について表 3.1 に列挙する。

表 3.1　代表的な rel の属性値と意味

rel 属性値	意味
start	開始ページ
next	次ページ
prev	前ページ
contents	目次ページ
index	索引ページ
stylesheet	適用スタイルシート

href 属性には、相対 URL 記述する。

```
<link rel="start" href="page001.html">
<link rel="next" href="page012.html">
<link rel="prev" href="page010.html">
<link rel="contents" href="contents.html">
<link rel="index" href="index01.html">
<link rel="stylesheet" href="halfmoon.css">
```

図 3.2　link 要素の例

meta 要素

meta 要素では、文書に関する様々な情報を記述する。代表的なのは、文書の検索用キーワードと、要約文である。

説明文　`<meta name="description" content="説明文">`
キーワード　`<meta name="keywords" content="word1, word2, ...">`

3.5.3　body 要素

文書本文を記述する部分である。body 直下に直接置けるのは、block 的な要素だけである。block 要素は、その名の通り、文書中で大きなブロックを構成するような意味づけの論理構造部品である。

これに対して、inline 要素は、block 要素内にのみ出現可能である。inline とは、行内、という意味であり、文書の行の一部分に対してマークアップするような意味づけの論理構造部品である。

代表的な flow 要素

p　段落 (paragraph) を示す。

h1　第 1 レベルの見出し要素。h2,h3,... と 6 レベルまである。

section　入れ子にして使うことで、セクション構成を示す。h1 などとの違いは、単に見出しを表現するのではなく、セクションとしての範囲を示す点にある。

ul, ol　箇条書きを示す。ul は番号なし、ol は番号付き。block 内で li 要素により、項目を示す。

dl　定義リストを示し、block 内で、dt で項目を、dd で項目内容をしめす。

blockquote　引用を示す block。

代表的な phrasing 要素

br　コンテンツとしての改行を示す。

em　強調（発声）する箇所。

strong　重要箇所。

img　画像を挿入する。src 属性に画像ファイルの URI を、alt 属性に代替テキストを指定する。

a　テキストにハイパーリンクを設定する。href 属性にリンク先 URI を指定する。

3.5.4　img 要素による画像の挿入

画像要素を示すのは、img 要素である。img 要素には、様々な属性が指定可能であるが、画像を文書中に挿入するにあたって、本質的に必要となるのは、src、alt の両属性のみである。

```
<img src="myphoto.jpg" alt="私の顔画像">
```

図 3.3　img 要素の例

img 要素による画像挿入例を図 3.3 に示す。img 要素は、flow contet, phrasing content, embedded content などに属する。多くの要素内に配置できる phrasing content の中でも、埋め込みコンテントである embedded content として配置可能な場所に配置できることになる。

この役割は、マークアップ箇所にコンテンツとして画像の挿入を指示することである。挿入するべき画像は、src 属性値に URL で示す。主 HTML 文書と同一サーバ内であれば、単に相対パス、あるいは絶対パスを示せば良い。

重要なのは、画像挿入の指示を直接に解決できないユーザーエージェント (UA) への対応である。テキストのみを表示するブラウザや、視覚障害者向け音声読み上げエージェントでは、挿入された画像を表示することは不可能である。そのため、挿入すべき画像の代わり、すなわち代替となるテキストを alt 属性値として示す。UA は、状況に応じてこの代替テキストを利用者に提示することができ、利用者は画像を直接見ることができなくとも、文書の内容を把握することができる。

したがって、alt 属性に指定する代替テキストには、画像がない場合に、文書の意味を指し示すようなテキストを指定しなければならない。例えば、単に「画像 1」のようなテキストでは不十分なことは明らかである。一方、装飾的に用いられる画像であるなら、代替テキストとして空文字列を指定する方が望ましい。例えば、ページデザイン上必要な、マスコットキャラクタの画像があったとして、文書内容に影響しないなら、「マスコット画像」などと代替テキストを提示されても、利用者は煩く感じるだけであろう。

3.5.5 a 要素によるハイパーリンク

HTML のハイパーテキストたる性格を主として体現しているのが a 要素である。この要素は、他の（あるいは、同一文書中の別部分への）ハイパーリンクをマークアップする。文書中の一部分が、他の文書を指し示すことになるため、マークアップすべき内容が存在することがわかる。内容は、他のマークアップを含んでも良い。例えば、画像（img 要素）を含むことも可能である。

ここでは、最も基本的なテキストに対するマークアップ例を示す。

```
<a href="http://www.kogakuin.ac.jp">工学院大学</a>
<a href="page002.html">次のページ</a>です。
```

図 3.4 a 要素の例

a 要素では、リンク先文書を href 属性で示す。href 属性値には、URL を指定する。上の例では、工学院大学のトップページを指定している。そのため、文字列「工学院大学」にハイパーリンクが設定され、[*13]このリンクを機能させると[*14]、リンク先の工学院大学トップページが読み込まれ表示される。

この URL は、画像の src 属性と同様、同一サーバ内の相対、または絶対パスであっても良い。下の例は、同一ディレクトリ内の page002.html ファイルへの相対パスでの指定となっている。

3.5.6 文書構造を示す要素

HTML5 では、文書構造を示す要素が強化されている。表 3.2 には代表的な構造化要素とその役割をまとめておく。これらは、Sectioning content とそれによって定義される headings, footers のうち、header、footer を取り上げてある。下位構造要素は、その要素に対して入れ子とできる構造化要素などを示してある。

3.6 マークアップの「正しさ」と「良さ」

3.6.1 論理的な正しさ

HTML では、文書の論理的構造をマークアップで示す。したがって、自己紹介の場合、紹介の各要素（氏名、性別、趣味など）を h2 でそれぞれマークアップすれば、見出しの一覧だけで、文書の概要を知ることが

[*13] 一般的なブラウザとスタイルシートでは、文字列に下線が引かれ、文字の色が変わり、ハイパーリンクが設定されていることが視覚的にわかるようになっている。

[*14] 通常のブラウザでは、クリックすると

できる。これが HTML での「良い」マークアップとされる。[*15]

3.6.2 形式的な正しさ

XHTML が XML 文書としてまた、XHTML 文書として妥当であるかどうか、検証することができる。このような妥当性検証を validation と呼ぶ。HTML においても、同様に HTML の規約を満足しているか判定することは重要である。

XML 文書が XML の文法規則を満たしているとき、**well-formed（整形式）**であると言う。さらに、XHTML として適切であるとき、XHTML として **valid（妥当）**であると言う。

The W3C Markup Validation Service[5] のような HTML/XHTML validator は、このような検証を行い、不正な部分を指摘してくれる。

多くの場合 Web ブラウザによる閲覧が予想される HTML 文書では、過去の経緯によって、規格から相当外れた文書であっても、Web ブラウザ側は適当な解釈ができるよう実装の努力が傾けられている。これは、多数の人々の手によって各 HTML 文書が作成されることからは当然と言える。

一方、HTML 文書の書き手（作成、生成者）としては、規格に合致した正当な文書を作成するべきである。これは、規格外の書き方はあくまで、受信者側のソフトウェアの実装に依存し、なんらコントロールする手立てを持たないためである。文書を書いた以上、正しく読んでもらうことに価値があるのではないだろうか？

[*15] さらに、画像が表示できない UA(user agent) で画像を説明したり、文字の読めない人や環境で、音声で読み上げることを考えたときに意味のわかるように記述するなど、「アクセシビリティ (accesibility)」を考えた記述が必須である。

表 3.2　構造化要素

要素名	役割	下位構造要素
article	一つの独立したコンテンツ。	article, section
section	一般的なコンテンツによるセクション。	h1, section, article
nav	サイト内の主ナビゲーション用	N/A
aside	補足的内容。	N/A
header	文書、コンテンツのヘッダ	
footer	文書、コンテンツのフッタ	

第4章

関係を表すHTML要素

4.1 表組み

table は表組を表現するための要素である。Flow content に属する。表は、長方形の駒と言うべき「セル（cell）」の並びとして表現され、それは横方向の「行（row）」と、縦方向の「列（column）」によって位置が決まる。複数の行や列を占有するような大きなセルの定義も可能である。記述は行単位で行われる。

4.1.1 簡単な表

```
<table>
<caption>受講している講義一覧</caption>
<tr><th>授業名</th><th>曜日</th><th>時間</th></tr>
<tr><td>理解のためのマルチメディア</td><td>火曜日</td><td>5, 6, 7</td></tr>
<tr><td>通信ソフトウェア論 I</td><td>水曜日</td><td>2</td></tr>
</table>
```

図 4.1　簡単な表組み

図 4.1 は、単純な表組みの例である。Table 要素下での最初の子要素には、caption 要素を置けることが規定されているため、ここでは caption 要素が置いている。Caption 要素は、その名の通り、表のキャプション（表の表題となる短い見出し文）を示す。

その後、表組の構造を表す tr 要素が続く。table 要素が表組み全体を表し、tr 要素が 1 行を表現する。行内の各セルは、th または、td 要素による。th は見出しセルを、td は通常のデータセルを表す。

ここでは、tr 要素が 3 要素並んでおり、3 行からなる表であることがわかる。Tr 要素下には、th 要素または、td 要素のみを置くことができ、いずれも、表のセルを表す。Th 要素は、表のセルとしての見出しセルを表す。項目のデータそのものではなく、データのカテゴリを示す文字列などが想定されている。Td 要素は、表中のデータセルを表す。

Th、td いずれの要素も、HTML の配置規則は同様で、tr 要素の子要素として配置される。また、これらの要素の子要素として配置できるコンテントモデルは、flow content であるが、th 要素では一部の要素は除外されている。

図 4.1 の表現している木構造を図 4.2 に示す。各項目を行単位でまとめて扱っている様子がわかる。

注意すべきは、table 要素もまた文章の構造のマークアップであり、罫線を用いた表形式とするかどうかなどは見映えの問題であるため、後に学ぶスタイルシートで指定する点である。上記の table は、おそらく図 4.3 のようにレンダリングされる。

ただし、過去との互換性もあり、セルを囲む枠線をとりあえず表示させるために、table 要素に border 属性[*1]を指定できる（図 4.4）。

[*1] Border 属性の値には、1 のみが許容されている。すなわち、<table border="1">である。この属性の値指定は省略できると規定されているため、<table border>でも同一の表現となる。

4.1.2 Table 要素

Table 要素のコンテントモデルは、

- caption（省略可）
- colgroup（省略可）（複数可）
- thead（省略可）
- tbody（省略可）（複数可）または、tr（複数可）
- tfoot（省略可）

という順序による構造を持つ。

4.1.3 複雑な表

```
<table border>
<caption>表の表題</caption>
<colgroup span="2">
  <col class="col1"><col class="col2">
</colgroup>
<thead>
 <tr><th colspan="2">見出し</th></tr>
</thead>
<tfoot><tr><td>フッタ 1</td><td>フッタ 2</td></tr></tfoot>
<tbody>
 <tr><td rowspan="2">項目 1</td><td>項目 1-2</td></tr>
 <tr><td>項目 2-2</td></tr>
</tbody>
</table>
```

図 4.5　複雑な表組み

表の表題は、table 要素内の先頭で、caption 要素により指定する。th、td 要素は、colspan、rowspan 属性により、行、または列を跨ぐセルを構成させることが出来る。いずれの属性も、行、または列を拡張（統合）するセルの数を属性値として持つ。

colspan 属性では、属性値の数の分だけ続くセルを統合する。統合された分のスペースを占めるはずのセルの td や th 要素は記述しない。

一方、rowspan 属性も、属性値の数の分だけ、続く行のセルを統合するが、この場合、続く tr 要素内に記述されるべきセルの記述をしないことになるため、直観的に理解が難しい。方眼に区切って表をデザインして、確認しながら記述することが必要である。

表中の列のグループ化要素として、colgroup 要素があり、構造としてまとめられる。その中で、さらに細かく見映え等の制御を行うために、col 要素がある。いずれも、span 属性で複数の列を指定できる。列は、実体となるデータでは複数の tr 要素内に散在する格好になるため、事前に、colgroup でグループ化を行うように記述する。

表中の行のグループ化要素として、thead、tfoot、tbody 要素があり、それぞれ、ヘッダ行、フッタ行、本

体行であることを示す。tr 要素を内部持つ形でグループ化を行う。

4.2 文書のグループ化

HTML 文書の一部（または全体）について、

- 文字色や大きさなどを変えたい
- 背景色や画像を入れたい
- 組み方（2 段組など）を制御したい
- 動的な（クリックされたら、など）変化

ということがある。これらは文書の論理的構造ではないため、HTML で指示することはできない。しかし、これらのことを行いたい文書の一部分をグループ化することは、何らかの指示対象をまとめておく、という意味において文書構造であると言え、このための記法が HTML にはある。

それは、div 要素と span 要素である。両者ともに、HTML 要素として特段の意味づけはなされていない。子要素に置いた要素を取り纏める目的に使用される。勧告内では、濫用への注意がなされており、これらの要素は、いわば「その他」を示すような論理構造である場合にのみ使用するべきである。

4.2.1 Div 要素

Div 要素は flow content に属する。原則として[*2]子要素には他の flog content に属する要素を置けるため、大きな構造を取り纏めるのに使用できる。

4.2.2 Span 要素

一方、span 要素は、phrasing content に属し、子要素にも phrasing contet のみを置くことができる。文中でいくつかの要素を取り纏めたいときにしようできる。

4.3 class 属性と id 属性

すべての要素には、class 属性と id 属性を付与することが出来る。これは、HTML の要素を識別、指定するための識別子として利用される。

例えば、css(cascading style sheet) でのスタイル（画面上の見映え）指定や、Javascript における操作対象要素としてである。具体的には‘chumoku’ class の p 要素の背景色を黄色、文字色を青、という指定ができる。表の一要素を正解がクリックされたら表示する、という操作も可能である。

4.3.1 div と class の例

図 4.6 は、class 属性をそれぞれの段落に付与した様子である。段落を 2 つのクラスに分類している。このような指定をすれば、奇数段落目と偶数段落目とで、文字の色を変えたり、背景色を変えるなどの処理が容易となる。

[*2] dt 要素の子要素としておかれている場合には、dt 要素によるコンテントモデルを強制され、dd 要素しか子要素に持てなくなる。

第 4 段落では、id 属性も指定されている。この id は、文書内で一意でなければならない。したがって、この id 属性値を指定することで、この第 4 段落を指定することがわかる。この段落だけ表示しないようにするなどの個別の処理が行える。

図 4.7 では、div 要素によるグループ化を行っている。図 4.8 と図 4.9 の文章としての内容は同一である。しかし、グループ化が異なる。この様子を、それぞれの構文木を図 4.8、図 4.9 に示す。

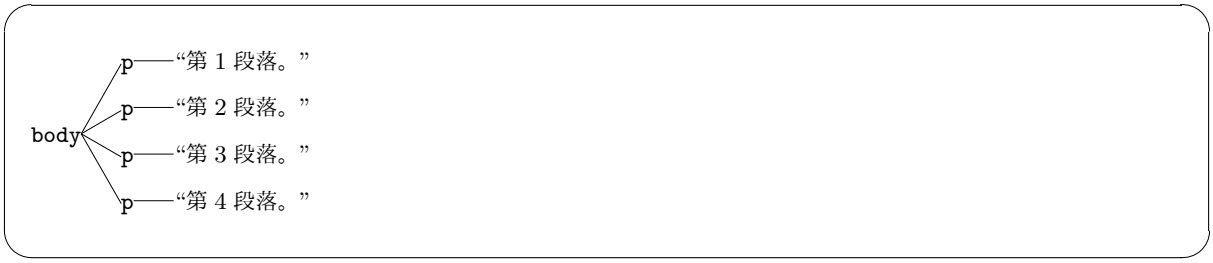


図 4.8　図 4.6 の構文木

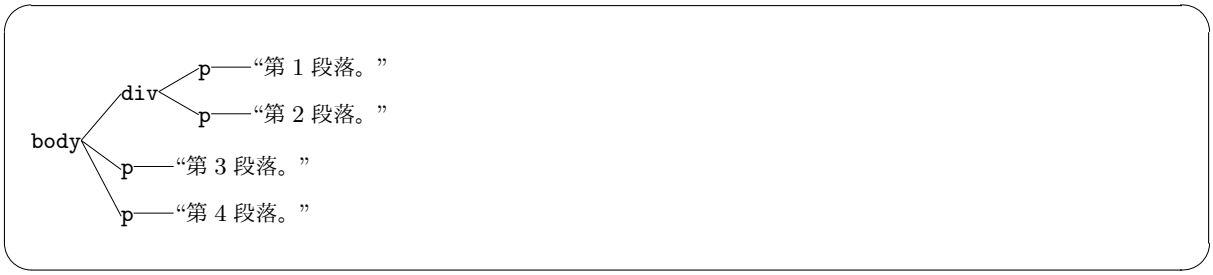


図 4.9　図 4.7 の構文木

例えば、第 1 段落と第 2 段落をまとめた外枠線を引きたい、というような場合には、図 4.9 のようなグループ化を行う必要がある。

具体例は、css の記述法を学んでから見ることとする。

4.3.2 Id と class 属性の書式

Id 属性や、class 属性には日本語文字も利用可能とされてはいるが、歴史的経緯により動作に不具合を生じるブラウザなどもあるほか、文字コードの解釈を、コンテントファイル、サーバ設定、ブラウザ設定で完全に一致するようにしておかないと、認識されなくなるため、実務的には（いわゆる）半角英数字のみを用いることが推奨されている。

ただし、先頭文字に数字は利用できない。必ず英文字ではじめ、英数字のみで構成されるようにすると問題が生じにくい。大文字小文字は区別されるが、区別されない場合も想定して、大文字小文字のみの違いで異なる id 名や class 名をつけるのは避けた方が良いだろう。

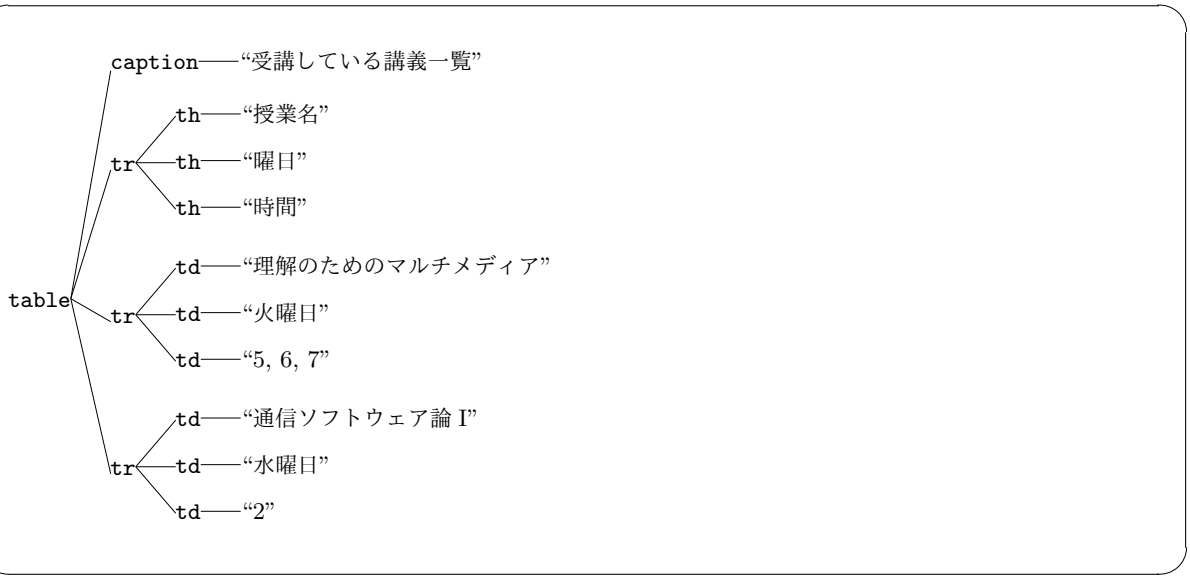


図 4.2　図 4.1 の構文木

受講している講義一覧

授業名	曜日	時間
理解のためのマルチメディア	火曜日	5, 6, 7
通信ソフトウェア論 I	水曜日	2

図 4.3　図 4.1 のレンダリング例

```
<table border>
<caption>受講している講義一覧</caption>
<tr><th>授業名</th><th>曜日</th><th>時間</th></tr>
<tr><td>理解のためのマルチメディア</td><td>火曜日</td><td>5, 6, 7</td></tr>
<tr><td>通信ソフトウェア論 I</td><td>水曜日</td><td>2</td></tr>
</table>
```

図 4.4　Border 属性を指定した場合

```
<body>
<p class="odd">第 1 段落。</p>
<p class="even">第 2 段落。</p>
<p class="odd">第 3 段落。</p>
<p class="even" id="fourth">第 4 段落</p>
</body>
```

図 4.6　class 属性の例

```
<body>
<div class="first_and_second">
<p class="odd">第 1 段落。</p>
<p class="even">第 2 段落。</p>
</div>
<p class="odd">第 3 段落。</p>
<p class="even" id="fourth">第 4 段落</p>
</body>
```

図 4.7　div によるグループ化

第III部

スタイルシート
―レイアウトとデザイン

第5章

Cascading Style Sheet

5.1 概要

Cascading Style Sheet(CSS) は、Web ページの視覚的な表現のための言語である。HTML が文書の論理的構造をマークアップするのに対して、CSS は、文書の「部分」に対して「見映え」を指定することができる。文書の「部分」の指定は、HTML における木構造の任意の枝を指すことによって行うことができる。また、「見映え」としては、文字の色、大きさ、フォント、背景色、枠囲いなどを指定できる。

5.1.1 スタイルシートの位置づけ

HTML は純粋に文書の論理的構造を記述することを目指して策定された過程の規格である。そのため、原理的には見映えとの関連はない。しかし、全く見映えの指定がない、ということは考えられない。なぜなら、例えば、ある要素を表示するのか、しないのか、ということ自体見映えの指定に他ならない。

そのため、各 Web ブラウザはブラウザ固有の初期スタイルシートが設定されている。Firefox では、`resource://gre-resources/html.css` とアドレスバーに入力することで、この標準のスタイルシートを見ることができる[*1]。

5.1.2 例に見るスタイルシート

```
h1 {
  display: block;
  font-size: 2em;
  font-weight: bold;
  margin: .67em 0;
}
```

図 5.1 Firefox の初期スタイルシート (h1)

図 5.1 は、Firefox の初期スタイルシートでの `h1` の宣言である。スタイルシートの基本的な記述ルールは、

[*1] ほかにも、`resource://gre-resources/forms.css`, `resource://gre-resources/quirk.css` などで各種の設定（利用者、情報発信者双方が指定していないときに利用されるもの）を確認することが出来る。

このようにスタイルの指定を行う要素名などを記述し、その後、具体的なスタイルを記述していく。ここでは、h1 要素に対して、次の 4 つの指定を行っている。

disyplay　block ボックスとして表示
font-size　文字の大きさを 2 倍に
font-weight　文字を太字に（bold）
margin　上下の余白を文字の 0.67 倍に、左右は 0

具体的な説明は 5.4 節で行う。

5.2　css の指定

Web ページにどのようなスタイルシートを関連づけて表示するのか、そのやり方は何通りかある。ここでは、Web ページ内での指定方法のひとつを紹介する。

それは、link 要素の rel 属性に stylesheet を設定し、href 属性として、適用するスタイルシートの URL を示す方法である。この方法は、Web ページの HTML ファイルとは異なる別のファイルにスタイルシートの情報を記述しておくため、外部スタイルシートとも呼ばれる。複数の Web ページに同一の見映えを指定したいときに、同一のスタイルシートの適用が容易にできるため、Web サイト全体でのデザインの統一を図るのにも適切だと言える。

スタイルシートは複数ファイルに分割しておくこともできる。また、視覚障害者への対応やデザイン上の方針により、Web ページに適用するスタイルシートを利用者が切り替えられるようにする仕組みも用意されている[*2]。

```
<link rel="stylesheet" href="<指定するファイルの URL、またはパス名>">
```

図 5.2　HTML5 における link 要素によるスタイルシートの指定（省略を活用した場合）

5.3　CSS のボックスモデル

HTML の要素は、表示される際には、ボックスモデルに基づく矩形の領域としてレンダリングされる。要素毎にひとかたまりの箱として並べられるイメージである。

ボックスモデルは、外側から、マージン（margin）、ボーダー（border）、パディング（padding）、内容領域、というように定義されている。

content　内容として文書に記述した文字列（テキスト）や画像が格納される領域。文字列自体の太さの指定や、表示するための幅や高さなどは、この内容領域に対する指定である。

padding　内容領域の外側、border までの領域。border が主に、枠線などとして表示されるように使われるのにたいして、その空隙領域として用いられる。したがって、背景画像などの指定は、この padding と内容領域を合わせた矩形内に表示される。

[*2] Internet Explorer などごく一部のブラウザではこの標準機能は実装されていない。

border　ボックスモデルにおいて枠線と考えて良い。背景画像を隠すように表示される。

margin　隣接するボックスとの間隔を指定するための余白である。当該ボックスが包含される親要素の背景は表示される。

領域間の境界辺は 4 辺の上下左右の位置に関して、それぞれ、top、bottom、left、right と名付けられている。

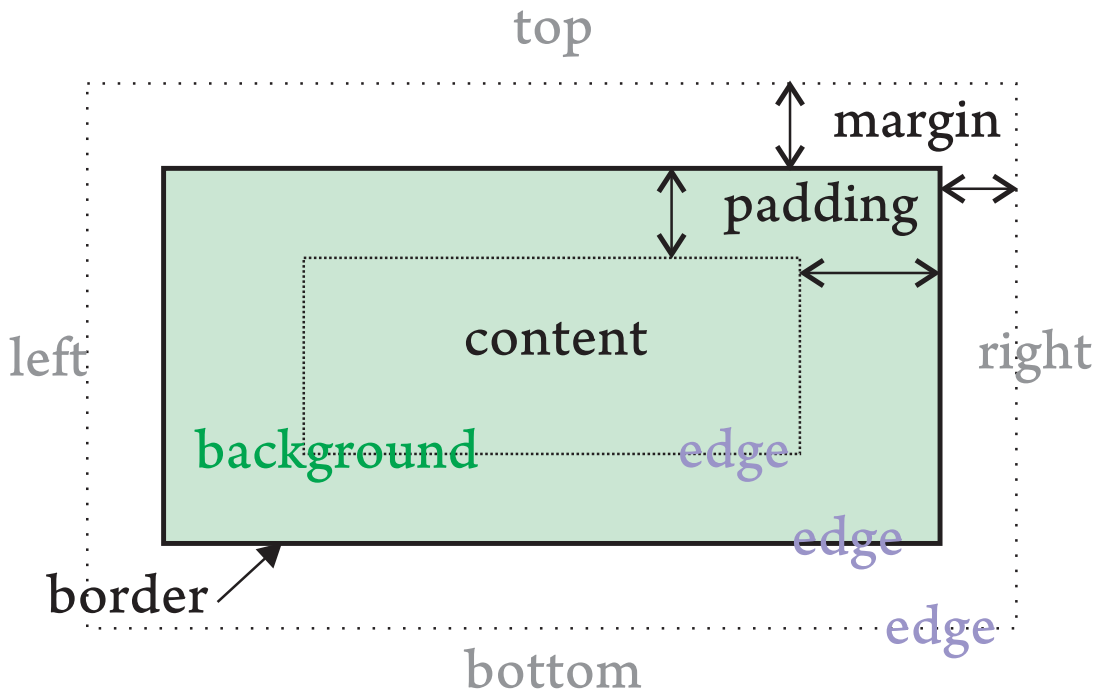


図 5.3　CSS のボックスモデル

5.4　CSS の書式

CSS によるスタイルの指定は、規則と呼ぶ。規則は、

```
<セレクタ> {<プロパティ>: <値>;
    <プロパティ>: <値>; ...}
```

のような書式を取る。

プロパティと値の組を宣言と呼び、セミコロン（;）で区切って宣言を複数並べ、中括弧（{ }）で括った部分を宣言ブロック（declaration-block）と呼ぶ。宣言ブロック内に宣言が複数並ぶ可能性があるため、セレクタからの一連の記述を規則集合（rule set）と呼ぶ。

5.4.1　セレクタ Selector

セレクタは、スタイルを適用する対象となる HTML 構文木の部分木を指す。例えば、p ならば全ての p 要素、td ならば全ての td 要素、といった具合である。セレクタとして指定できるものは多岐にわたるが、本書

では代表的なタイプセレクタ、ユニバーサルセレクタ、クラスセレクタ、ID セレクタについて扱う。詳細は後述する。

5.4.2 宣言

プロパティと値の組を宣言と呼び、セレクタ指定に対して、実際に適用するスタイルを指定する。セミコロンで区切ることで、複数の宣言を同時に指定することができる。CSS では、ボックスモデルに基づく見映えの指定が行え、モデルの各部分について個別に宣言によって色や大きさなどを指定する格好になる。

5.4.3 スタイル指定の例

```
* {
  color: #5555ff;
  border: thin solid #ffffff;
  background-color: #dddddd;
}

body {
  background-color: #90b090;
}

td.absentee {
  font-size: x-small;
  background-color: #f0a0a0;
  text-align: center;
}
```

図 5.4　スタイルシートの例

図 5.4 では、セレクタとして、'*'、'body'、'td.absentee' の 3 つを指定した、規則が宣言されている。

'*' は、ユニバーサルセレクタ (universal selector) と呼ばれ、あらゆる要素に対してスタイルを指定できる。この例では、文字色を RGB: 5555ff（16 進数）という青っぽい色にし、border を細線、実線、白色としている。さらに、背景色を灰色と指定している。

'body' は、body 要素の指定である。緑色っぽい色の背景色を指定している。

'textttd.absentee' は、class セレクタの例である。この場合、td 要素のうち、class="absentee"と属性値を指定した要素のみへのスタイル規則となる。宣言内容は、文字の大きさを小さくし、背景色を赤っぽい色に、また、文字をセンタリングしている。

5.5 CSS の体験

1. ch04/ch04.html を、ch05/ch05.html にコピーする。
2. ch05/ch05.html の head 要素内に、次の link 要素を追加する。

```
<link rel="stylesheet" type="text/css" href="ch05.css">
```

3. ch05/ch05.css を新規作成する。

まず、簡単に、通常はタイトルバーにしか表示されない title 要素を通常の描画エリアに表示させてみよう。

```
head {
  display: block;
}

title {
  display: block;
  color: #004000;
  text-align: right;
  margin: 2em;
  border: 8px solid #FF0000;
  padding: 1em;
  background-color: blue;
  width: 400px;
}
```

この内容を ch05/ch05.css ファイルに記述する。その上で、ch05/ch05.html ファイルへ Web アクセスするとどのように表示されるか確認してみよう。

5.6 Web 上の resource

http://firefox.geckodev.org/index.php?usercontent.css では、firefox へユーザが指定したスタイルシートを適用することで出来ることの可能性についてまとめている。良く読み、実際に試してみると、スタイルシートの効用について理解出来るだろう。

第6章

cssの書式

6.1 規則集合

css で、(X)HTML の（class や id で指定される名前を含めた）要素に対する書式を指定する一連の記述を規則集合（rule set）と呼ぶ。規則集合は、セレクタ（selector）で始まり、中括弧（{ }）で括られた宣言ブロック（declaration block）によって記述される。

宣言ブロックは、プロパティ（property）と対応させる値（value）とをコロン（: colon）で区切って並べた宣言（declaration）を、必要に応じて複数、セミコロン（; semi-colon）で区切って記述したものである（図 6.1）。

```
<セレクタ> {<プロパティ>: <値>;
    <プロパティ>: <値>; ...}
```

図 6.1 規則集合の書式

6.2 Selector

セレクタは、スタイルを適用する対象となる (X)HTML 構文木の部分木を指す。これによって、宣言ブロック（declaration block）で指定するデザインを適用する範囲を選定する。

例えば、`p` ならば全ての `p` 要素、`td` ならば全ての `td` 要素、といった具合である。このような、(X)HTML の要素名のみをセレクタとして指定するようなセレクタを type selector という。各種のセレクタのうち、代表的なタイプセレクタ、ユニバーサルセレクタ、クラスセレクタ、ID セレクタについて示す。

6.2.1 Type selector

HTML の要素名によって指定するセレクタである。特定の要素のデザインを一律に設定する際などに指定される。

6.2.2 Universal selector

universal selector は、あらゆる HTML 要素を対象とする。これは、アスタリスク (*) で示す。

6.2.3 Class selector

class セレクタは、文書内の複数の要素に同一の `class` 属性を持たせることで、複数の要素に同一のスタイルを適用することができる。class セレクタは、先頭のピリオド (`.`) に続けて `class` 属性名を記述する。

6.2.4 ID selector

ID セレクタは、`id` 属性値が文書内で唯一であるため、特定の要素にのみ当該スタイルを適用させるときに用いる。ID セレクタは、先頭がハッシュ記号 (`#`) である。

6.2.5 複雑な指定

セレクタは、複数のセレクタをカンマで区切って並べることで、複数の要素に同一のスタイルを適用することもできる。また、他の指定方法も多種あるが、やや複雑な用法であるため、本授業内では扱わない[*1]。必要なら適当な文献 [10][11][14] を参照せよ。

6.3 ボックスモデル

HTML の要素は、表示される際には、ボックスモデルに基づく矩形の領域としてレンダリングされる。要素毎にひとかたまりの箱として並べられるイメージである。

ボックスモデルは、外側から、マージン（margin)、ボーダー（border)、パディング（padding)、内容領域、というように定義されている。

content　内容として文書に記述した文字列（テキスト）や画像が格納される領域。文字列自体の太さの指定や、表示するための幅や高さなどは、この内容領域に対する指定である。

padding　内容領域の外側、border までの領域。border が主に、枠線などとして表示されるように使われるのにたいして、その空隙領域として用いられる。したがって、背景画像などの指定は、この padding と内容領域を合わせた矩形内に表示される。

border　ボックスモデルにおいて枠線と考えて良い。背景画像を隠すように表示される。

margin　隣接するボックスとの間隔を指定するための余白である。当該ボックスが包含される親要素の背景は表示される。

領域間の境界辺は 4 辺の上下左右の位置に関して、それぞれ、top、bottom、left、right と名付けられている。

[*1] 特定の要素の木構造であることを前提とした指定や、link 文字列などを表す疑似クラスなどが存在する。

6.4　スタイル指定の例

```
p {
  background-color: #90b090;
}

*.newsrelease {
  color: #000000;
  background-color: #80ffff;
}

.ad {
  font-size: small;
}

td#special01 {
  background-color: #f0a0a0;
  text-align: center;
}
```

図 6.2　スタイルシートの例 2

図 6.2 では、セレクタとして、‘p’、‘*.newsrelease’、‘.ad’、‘td#special01’ の 4 つによる、規則集合が定義されている。

‘p’ は、最も一般的なセレクタで、文書内の全 p 要素へ適用するスタイルを宣言している。‘*.newsrelease’ は、ユニバーサルセレクタ ‘*’ を前置して、ドット（.）により、適用するクラス名を指定している。この場合、文書内の要素の種類に関わらず、クラス名として ‘newsrelease’ を指定された全ての要素を対象としている。‘.ad’ は、このユニバーサルセレクタを省略した書き方である。‘*.ad’ と全く同一のセレクタである。‘td#special01’ は、ハッシュ（#）を区切りとして、id 名を指定したセレクタである。この場合、td 要素のうち、special01 なる id 名を付与された要素を指定している。ただし、id 名は、文書中において一意であるため、(X)HTML 要素名を付与せず指定することが一般的[*2]である。

6.5　css の指定

主文書たる (X)HTML に対して、対応する css 文書を指示する方法のうち、最も良いとされるのが、(X)HTML の link 要素を用いて外部ファイルを指定する方法（外部スタイルシート）である。これは、link 要素の rel 属性に stylesheet を設定し、type 属性に"text/css"、href 属性として、適用するスタイルシートの URL を示す（図 5.2 参照）。複数の Web ページに同一の見映えを指定したいときに、同一のスタイルシートの適用が容易にできるため、Web サイト全体でのデザインの統一を図るのにも適切だと言える。

link 要素は、(X)HTML では、head 要素下に入れる。

[*2] 理由を考えよ。

スタイルシートは複数ファイルに分割しておくこともできる。また、視覚障害者への対応やデザイン上の方針により、Web ページに適用するスタイルシートを利用者が切り替えられるようにする仕組みも用意されている[*3]。

```
<link rel="stylesheet" href="<指定するファイルの URL、またはパス名>" />
```

図 6.3　HTML5 における link 要素によるスタイルシートの指定（省略を活用した場合）

```
<link rel="stylesheet" type="text/css" media="all"
  href="<指定するファイルの URL、またはパス名>" />
```

図 6.4　link 要素によるスタイルシートの指定

6.6　value

6.6.1　長さ

em　font-size property を 1 とする単位
ex　英小文字 'x' の高さを 1 とする単位
px　画面の（論理的な）ピクセル
pt　ポイント (1/72 inch)
mm　ミリメートル
cm　センチメートル

6.6.2　色

6 桁 16 進 RGB 値　#に続けた 6 桁 (rrggbb) の 16 進数 (0, ⋯, 9, a, b, c, d, e, f)。
　　ex) `#7f55ff`
rgb()　rgb() 内に、r,g,b の順で 10 進数値を指定。
　　ex) `rgb(128, 255, 255)`

6.6.3　URI

url()　css ファイルのディレクトリからの相対パスで示す URL。
　　ex) `url("../img/image002.jpg")`

[*3] Internet Explorer などごく一部のブラウザではこの標準機能は実装されていない。

6.6.4 相対値

% パーセンテージで相対的な長さや大きさを示す。
ex) 150%

6.7 property

css2.1 での代表的なプロパティを種類別に紹介する。

6.7.1 テキスト

text-align テキストの揃え。
`left`, `center`, `right`, `justify` など
text-decoration テキストの装飾。下線など。
`underline`, `overline`, `line-through`, `blink` など
font-size フォントの大きさ。長さ、相対値で指示する。
font-style フォントの変形。
`normal`, `italic`, `oblique` など
font-weight フォントの太さ。
`bold`, `normal` など

6.7.2 背景

background-color 背景色を色で指示する。
background-image 背景に表示する画像を URI で指示する。
background-position 背景画像の表示位置を、画像左上の window 内相対座標で指示する。長さで指示する。
background-repeat 背景画像をタイルのように敷き詰めるか否か。
`repeat`, `repeat-x`, `repeat-y`, `no-repeat`

6.7.3 囲み

margin や padding など、content を囲む領域の大きさを指定する。値は 1〜4 つ指定でき、個数によって、上下左右の指定箇所か異なる。

1 つ 上下左右（全部共通）
2 つ 上下、左右
3 つ 上、左右、下
4 つ 上、右、下、左

margin 上下左右の margin を長さ、相対値で指示する。

padding　上下左右の padding を長さ、相対値で指示する。
border-width　境界線の太さ。
　　`thin`, `medium`, `thick`, 長さ など
border-style　境界線の形状。ブラウザによって対応が異なる。
　　`none`, `hidden`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset` など
border-color　境界線の色。色で指示する。

6.8　pseudo-class

セレクタとして指定できる、特殊なクラスが存在する。ここでは、その中でも分かりやすい 2 分類 5 種類のクラスについて紹介する。

6.8.1　リンク疑似クラス

:link　未訪問リンク（a:link）
:visited　既訪問リンク（a:visited）

6.8.2　ダイナミック疑似クラス

:hover　指示されているがアクティブでないとき（ポイント時）
:active　アクティブ状態（ボタン押下時）
:focus　フォーカス時（要素選択時）

6.9　演習問題

1. `ch05/ch05.html` を、`ch06/ch06.html` にコピーする。
2. `ch06/ch06.html` 内に `h2` 要素がなければ、適当な内容で追加する。
3. この `h2` 要素の色を変化させる。
   HTML ファイルと、css ファイルの 2 つのファイルを変更する必要がある。
   HTML ファイル `ch06/ch06.html` には、
   - `h2` 要素に `id` 属性を設定し、適当な属性名をつける。
   - `head` 要素内に `link` 要素を追加して、適当な外部スタイルシートを参照するようにする。（cf. 5.5 節）

   上記で設定した名称の css ファイルを作成して、
   - セレクタとしては id セレクタを用いる。#記号で始める。
   - 文字色を指定するプロパティ名は `color` である。
   - 色指定書式に従い、記述する。
4. `table` 要素を装飾してみよ。
   （a）ex.）`border` 属性を取り除いて書き換え。罫線の工夫。

（b）ex.) セルの背景色や文字色などを調整して見やすく。

5. css によって、どんな指定をしたかを HTML ファイル内で説明せよ。
   （a）Rule set の記述内容。
   （b）その具体的な効果など。

第IV部

JavaScript
―ページの動的制御

第7章

JavaScript入門

7.1 Javascript

7.1.1 言語

Javascript は、Web ブラウザ上での処理を目的として開発されたインタプリタ型のオブジェクト指向言語である。多彩な機能を言語組み込みで持ち、比較的単純な記述で一定の仕事をさせられることからスクリプト言語[*1]にも分類される。

Web 情報システムにおいては、Web サーバ上に配置されたプログラムファイルを、クライアントである Web ブラウザに転送してきた後、ブラウザ上で実行されるのが典型的処理である。近年は、サーバサイド（サーバコンピュータ上で実行する、の意）Javascript も注目されている。本章では、Web ブラウザ上で動作する Javascript について説明する。

Web ブラウザ組み込みの実行環境であるため、追加の実行用ファイルなどを用意する必要はないが、ブラウザ間での互換性に問題があり、実務上の大きな障害となっている。互換性問題に対処するため、ECMA[*2]が標準化を行ったものを ECMAScript(ECMA-262) と呼ぶが、実務面の問題を解消するには至っていない。

なお、Sun microsystems が開発した Java 言語とは、名称が似ているだけで全く無関係[*3]である。

7.1.2 処理モデル

実行環境が Web ブラウザ上にある場合、Window や、HTML document などを組み込みオブジェクトとして持つ。(X)HTML ドキュメントは、木構造で表現され、DOM[*4]に基づいた表現として取り扱われる。また、CSS 要素にもアクセス可能である。

そこで、document tree や CSS tree にアクセスし、要素の追加、削除、更新や属性の追加、属性値の変更などを行うことで、Web コンテンツに対して非常に広範囲の変更を加えることができる。言い換えれば、あるページの状態 A を、別のページ状態 B にしたい、ということが実現可能である。

利用者とのインタラクション目的に利用されることを企図して設計されている。そのため、後に述べるよう

[*1] スクリプト言語とは、容易にプログラミングが始められるようなプログラミング言語を指すものとされるが、厳密な定義は存在しない用語である。

[*2] Europian computer manufactures association

[*3] 興味のある人はそのあたりの経緯を調べると良い。

[*4] Document Object Model

な、Web コンテンツの閲覧モデル、また、利用者の操作に基づく、プログラムの起動 trigger としてのイベント駆動モデルで実行される。

インターネット環境での利用を前提としているため、セキュリティの観点から、プログラムを取得したドメインのサーバとの通信しか許されず、実行中の local マシンに対するファイルアクセスも許されない。厳密に言えば、cookie ファイルへのアクセスは実務上可能である。また、HTML5 においては相応の local storage へのアクセス機能が付与された。しかしながら、入門者にとっては難解であるため、このような厳密な解説は他書に譲る。

本書では、このように初学者向けに一部を省略した説明をする場合がある。これは、一度に複雑かつ厳密な理解と知識を持つことが難しいため、まずはプログラムがどのように構成されるのか、一般的な記法や機能に絞って説明するためである。また、プログラム要素や文法の説明を先に行う伝統的なプログラミング教科書の構成はとらない。これは、自然言語においても単語や文法だけを覚えても文章を構成できないのに似ていると考えるからである。単純なサンプルのプログラム記述に対し、必要に応じて説明を行う。このため、他人の書いたソースコードの理解や、実務的なプログラミングのために必要となる網羅的な理解には不足である。他書を参照されたい。

7.2 HTML との関連づけ

JavaScript プログラムを、HTML ファイルと関連づけるためには、(X)HTML の `script` 要素を `head` 要素下に配置し、外部ファイルを指定することが一般的である。

```
<script src="<指定したい URL>" type="text/javascript" charset="utf-8"></script>
```

図 7.1 `script` 要素による Javascript リソースの関連づけ

```
<script src="<指定したい URL>"></script>
```

図 7.2 Javascript リソースの関連づけ

JavaScript プログラムファイルへの URL を用いて、図図 7.1 のように記述する。ここで、プログラムファイルの文字コードとしても utf-8 を指定している。したがって、プログラム自体も utf-8 で記述する。

`script` 要素は、要素内容（content）としてスクリプトプログラム（コード）を記述することも可能である。しかし、XML や HTML の書式要件と、使用するプログラム言語の書式要件とを両方とも満たすよう記述する必要があるため、取り扱いが難しい。また、プログラムコードは、多くを複数の page で共通して使うため、個々の HTML ページに埋め込んでしまうと、更新時（機能追加や誤りの修正など）に支障を来す。すなわち、文書内容自体（論理的意味構造）と、閲覧時の追加機能としての動的要素であるプログラムコードとは、担当者が異なることが一般的でもあり、分離した方が望ましい。

`src` 属性には、実際に読み込むべき Javascript ファイルへのフル URL、絶対 path、相対 path のいずれかを記述する。通常は、同一サイト内のファイルを読み込むため、絶対 path か相対 path で指定する。

また、script 要素は、歴史的経緯[*5]から、空要素として記述することができない。そのため、閉じタグが必要である点に注意しよう。

HTML5 では、type="text/javascript"が default とされ、この場合、type 属性は省略できる。また、HTML 本文も utf-8 で記述されていれば、charset="utf-8"も省略できる。その結果、図図 7.2 のように記述しても構わない。

script 要素は、読み込みたいリソースの数に応じて任意の個数を記述できる。Javascript では、これらは読み込まれた順に実行（評価）されていく。すなわち、すべてのスクリプトファイルをすべてつなげた状態で実行する順序である。

ここで指定した Javascript プログラムでは、特定の (X)HTML 要素を操作（イベント取得を含む）対象とするため、操作したい当該 (X)HTML 要素には class、id 属性により識別子を割り当てておく。

7.3 JavaScript 入門

C 言語類似の文、制御文を持つと同時に、オブジェクト指向の文法を取り入れている。

7.3.1 Hello, world!

あらゆるプログラミング言語の恒例に従い、最初のプログラムは、その言語において最も単純な手法で、'Hello, world!' を出力する、通称 Hello, world プログラムを書いてみる。

```
// Sample 1
function sample1(event)
{
  window.alert('Hello, world!');
}
```

図 7.3　Hello world プログラム

処理本体は上記のみである。ほかに、HTML から呼び出せるようにする仕掛けが必要であるので、それは後で説明する。

コメント

'//' は、C++ スタイルのコメント記号で、これ以後、行末までをコメントとしてインタプリタは読み飛ばす。'/*' で始まり、'*/' で終わる C スタイルのコメントも使用できる。

メソッド

function キーワードは、JavaScript の予約語[*6]の一つで、関数 (関数オブジェクト; メソッド) の定義を開

[*5] Web ブラウザの実装上の問題である。

[*6] プログラム言語、ここでは Javascript、の規格の上で特定の機能を指し示すため、利用者であるプログラマが自由に使えないよう「予約」してある語を指す。Javascript の予約語は現状で約 30 個、将来的な拡張のため予約語となりそうな語はその約倍程度考えられる。予約語に重なる語は使用できない場面があるため、一般的な語を避けたプログラミングが安全である。

始する宣言である。ここでは、関数名 sample1 を定義しようとしている。丸括弧内でこの関数の仮引数を指定している。ここでは、event という仮引数を指定しているが、関数本体中では使用していない。

関数定義は、波括弧で括ったブロックで記述する。定義本体は、window.alert() の呼び出しのみである。これは、window オブジェクトの alert メソッドの呼び出しを表す。

また、このように末尾がセミコロン（;）で終わっているような 1 行を、ひとまず「文」と呼ぶ。

メソッドとは、オブジェクト指向プログラミングの用語で、オブジェクトに関連づけられた関数だと理解すれば良い。言い換えると、オブジェクトを操作対象とする関数であることを明示している記法である。オブジェクトとは、プログラム内での何らかの実体を表す「モノ」である。Javascript では、オブジェクト毎に処理をさせるメソッド群が定義されており、必要に応じて、オブジェクトと処理内容のメソッドを指定し、詳細を定めるパラメータを引数を介して渡すことで、意図した動作をさせる。

オブジェクトに対しては、プログラマが自身で新規のメソッドを追加で定義することもできる。function キーワードは、そのための定義を始める、という宣言であるといえる。

メソッドは、プログラマが処理させたい仕事内容をまとめたものであり、プログラムの各所からその仕事内容をひとまとまりとして実行させたい単位として記述する。どの程度の処理をメソッドとして定義するべきかは、理論上も実務上も興味深い問題であるが、ここでは議論しない。

はじめてプログラムというものに接する人のために、比喩表現で説明しよう。

メソッド（関数）

一定の仕事の手順を示す手順書がメソッドである。この手順書には表題をつけられ、これがメソッド名（関数名）である。
手順中で参照すべきデータを渡すことができ、これは引数と呼ばれる。
各手順は、中括弧で始めと終わりを明示した上で、その中に、順に文として並べて書く。
メソッドは、仕事が完了（または中途で終了）したときにどのように仕事を行ったかを報告するため、戻り値を持つことができる。
引数を与えると、一定の処理を行い、戻り値を返す、という特性から関数とも呼ばれる。

window オブジェクト

window は、Web ブラウザのウインドウ自体を示すオブジェクトで、Web ブラウザ上で実行される際には、実行環境が事前に準備して利用可能な状態として提供される。

オブジェクトは、状態を表す属性情報を保持する容れ物としてプロパティと呼ばれるものを任意の個数持てる。プロパティには、数値や文字列、他のオブジェクト、メソッドなどを保存できる。各プロパティは識別のため名付けることができて、これをプロパティ名などと呼び、通常はこの名前で保存容器としてのプロパティを指定する。

たとえば、abc と名付けられたオブジェクトに def と ghi という 2 つのプロパティがあったとすると、それぞれ、abc.def、abc.ghi のようにオブジェクト名とプロパティ名をドット（.; ピリオド）でつないで記述する。

プロパティは空の場合もあり、数値や文字列などの単純なデータらしいものが入っていることもある。また、他のオブジェクトやメソッドも入れられる。このような容れ物（箱）として見た場合、プロパティは変数と呼ばれる。中身が変わることのあるデータ（数値だけに限らない）、という意味合いである。

気をつけなければならないのは、他のオブジェクトやメソッドは、大きすぎて直接プロパティには入らない

点である。たとえばメソッドは、手順書のようなものであり、長い紙に多数の手順を連ねたようなものであると言える。これは、小さな容器であるプロパティ（変数）にはそのまま入らない。そこで、このようなものは、本体（ここではメソッド定義；手順書）は棚にしまっておき、変数には、「棚のどこどこの位置に置いてあるもの」というような小さな紙片だけを入れておくような処理が行われる。この格納位置を指し示していることを「参照」と呼ぶ。

オブジェクトのプロパティにメソッド定義を保存（代入）してやると、そのプロパティ名を使って、このメソッドを呼び出す（実行）することができる。Javascript の処理系は、プロパティに保存された参照を使って、メソッド定義の本体を見つけ、それをプログラムとして実行するのである。

alert() メソッド

alert メソッドは、window オブジェクトに Javascript 処理系[*7]が提供しているメソッドである。このメソッドは、呼び出すと、引数の文字列と、「OK」ボタンのついた警告ダイアログボックスを表示する。

文

オブジェクト名とメソッド名を.（ピリオド）でつなぐと、そのメソッドが呼び出される。このような単位を文と呼び、;（セミコロン）で区切ることで複数の文を並べることができ、上から順番に実行させることができる。

Javascript の文は、広範囲に渡ってのセミコロンの省略を許容する。しかし、文法を正確に理解していないと誤った解釈となる恐れがあるため、文の末尾にはセミコロンを置き、また、途中では改行しないようにした方が良い。

文には、条件判断を行わせて、処理内容を分岐させることのできる if 文や、一定の条件下で、記述した処理内容を繰り返し処理させる for 文なども用意されている。基本的な書式は、C 言語に類似しているが、Javascript 独自の拡張も見られる。

7.3.2 HTML からの呼び出し

ここでは、第 06 章でのファイルを用いて試してみよう。

1. ch06/ch06.html, ch06/ch06.css を、ch07/ch07.html, ch07/ch07.css にそれぞれコピーする。
2. 新規ファイル ch07/ch07.js に、図 7.3 のコードを記述。
3. ch07/ch07.html に、script 要素を適切に記述。

ここまでが、処理本体のための準備である。次に、この HTML ファイル表示時に sample1() が呼び出されるような記述を追加する。

1. 06 章の演習問題で、一つだけ色を変えた h2 要素の id 属性値を記録する。
   - ここでは、その id 属性値が sample1 であるとして説明する。
   - 例) <h2 id="sample1">...</h2>
2. 図 7.4 のコードを、ch07/ch07.js の JavaScript コードに追加する。<id 属性値>は、sample1 である

[*7] 正確には、Web ブラウザ上の処理系が提供するのが、window オブジェクトなどである。alert メソッドは、window オブジェクト中のプロパティの一つである。

が、自分の指定した属性値に変更すること。

```
window.onload = function()
{
  /* sample1 イベントハンドラ設定 */
  var node_sample1
    = document.getElementById('<id 属性値>');
  node_sample1.onclick = sample1;
};
```

図 7.4　HTML 要素との関連付け

間違いがなければ、ch07/ch07.html をブラウザで表示させ、該当する h2 要素をクリックすると、先のメソッド sample1() が呼び出され、'Hello, world!' と書かれたダイアログウィンドウが表示される。[*8]

7.3.3　解説

window.onload は、window オブジェクトのイベントハンドラ[*9]の一つで、Web ページの読み込み (load) が完了した時に呼び出される一連の処理を設定する。ここでは、function() として、以下に定義する処理を呼び出させるようにしている。

波括弧ブロック内は、まず、C 言語スタイルのコメント文がある。

次の var は変数の宣言を行っている。変数（オブジェクト）node_sample1 を宣言し、引き続き代入演算子によって、値を割り当てている。割り当てられるべき値は代入演算子の右辺にある、メソッド呼び出しの結果（戻り値）である。

document.getElementById('<id 属性値>') は、<id 属性値>を属性 id に持つ、文書内で唯一（それが、id 属性の意味であった）の HTML の要素を取り出す。したがって、変数 node_sample1 には、その要素が代入される。[*10]この要素は、Javascript 上ではオブジェクトとして取り扱われる。すなわち node は、その HTML 要素データを含むような Javascript オブジェクトである。

node_sample1.onclick は、その取り出した要素の onclick プロパティを指す。すなわち、node オブジェクトが保持する onclick プロパティである。

オブジェクトとプロパティの関係は、包含するものと包含されるものである。オブジェクトは多数のプロパティを持ちうる。オブジェクトの保持するプロパティは、Javascript では変数と同義である。すなわち、プロパティ（変数）は、数値や文字列、他のオブジェクトや関数（メソッド）を保持できる。

オブジェクトの種類によって、特定のプロパティに特別な役割を持たせてあることがある。HTML 要素を

[*8] 実際にどのような形で表示されるかは、実行を担う Web ブラウザ（実行環境とも呼ぶ）によって異なる。以前は、小さなダイアログボックスとして表示される実行環境が多かったが、利用者に確実に注意を喚起するために、呼び出し元の window 全体を暗転させ、警告メッセージを表示するようなインタフェースも見られるようになった。

[*9] 厳密に言えば、window オブジェクトの特別なプロパティの一つで、ここに保存されたメソッドが特定のタイミング（イベント発生時）で呼び出されるというものである。このようなイベント時に実行されるメソッドのことをイベントハンドラと呼ぶ。

[*10] 気をつけるべきは、要素単位で取り出される点で、例えば、body 要素を取り出したとすると、そこにふくまれる本文全体を指すことになる。

示すオブジェクトの onclick プロパティも、そのような特別な役割を割り当てられたプロパティの一つである。onclick プロパティは、その要素がマウスでクリックしたときに呼び出される関数を保持する役割を持つ。逆に言えば、このプロパティの中が空なら、この HTML 要素はクリックされてもなにも起こらないが、このプロパティに何らかの処理手順を記述した関数が保持されていると、クリックされたときにその関数が呼び出される（実行される）。

したがって、ここでは、sample1 という関数が呼び出し先として登録される。onclick はマウスクリック時のイベントハンドラ（を保持する箱）というわけである。

7.3.4 イベント一覧

どのようなイベントが取り扱えるのか、主な一覧表で示す。設定可能な要素や制御手順に制約があるが、それらについては割愛する。

7.4 演習問題

1. ch07/ch07.html の Hello, world プログラムの出力メッセージを適当な日本語に変えたものを設定せよ。
2. 設定した h2 要素は、Web ブラウザで見てわかるよう、説明を加えておくこと。
3. Valid[5] であり、Web ブラウザを通じて HTTP でアクセスできることを確認。

表 7.1　主なイベントハンドラを設定するためのプロパティ

イベント名	動作
onchange	フォーム状態が変化した
onclick	オブジェクトがクリックされた
ondblclick	オブジェクトがダブルクリックされた
onfocus	オブジェクトフォーカスされた
onblur	オブジェクトのフォーカスが外れた
onkeydown	key が押された
onkeyup	key が離された
onkeypress	key が打たれた
onload	page/画像が読み込まれた
onunload	page 移動、閉じた、再読込
onmousedown	オブジェクト上でマウスボタンが押された
onmouseup	オブジェクト上でマウスボタンが離された
onmouseover	オブジェクトにマウスポインタが重なった
onmouseout	オブジェクトからマウスポインタが離れた
onmousemove	オブジェクト上でマウスポインタが移動した
onsubmit	フォームの送信ボタンが押された
onreset	フォームの取消ボタンが押された

表中でのオブジェクトは、Javascript 上の当該オブジェクトであり、すなわち、HTML の要素を指す。

第8章

Javascriptの文法

C 言語類似の文、制御文を持つと同時に、オブジェクト指向の文法を採り入れている。

8.1 再掲：Hello, world!

前章で見た、Hello, world! プログラムをじっくり読んでみよう。

```
// Sample 1
function sample1(event)
{
  window.alert('Hello, world!');
}
```

図 8.1 Hello world プログラム

処理本体は上記のみである。HTML から呼び出せるようにする仕掛けの部分については後で説明する。

8.1.1 コメント

'//' は、C++ スタイルのコメント記号で、これ以後、行末までをコメントとしてインタプリタは読み飛ばす。'/*' で始まり、'*/' で終わる C スタイルのコメントも使用できる。

コメントは、プログラムを他人[*1]が読む際に非常に重要な役割を果たすため、適切に付与しておくことが望ましい。プログラムコードを読むだけですぐにわかることではなく、そのようなコードを書いた理由や、処理の高度な意味づけなどについての説明が一般には適切であるとされる。

8.1.2 メソッド

function キーワードは、JavaScript の予約語[*2]の一つで、関数 (関数オブジェクト; メソッド) の定義を開

[*1] 3 ヶ月後の自分自身も含まれる。

[*2] プログラム言語、ここでは Javascript、の規格の上で特定の機能を指し示すため、利用者であるプログラマが自由に使えないよう「予約」してある語を指す。Javascript の予約語は現状で約 30 個、将来的な拡張のため予約語となりそうな語はその約倍程度考

始する宣言である。ここでは、関数名 sample1 を定義しようとしている。丸括弧内でこの関数の仮引数を指定している。ここでは、event という仮引数を指定しているが、関数本体中では使用していない。

関数定義は、波括弧で括ったブロックで記述する。定義本体は、window.alert() の呼び出しのみである。これは、window オブジェクトの alert メソッドの呼び出しを表す。

また、このように末尾がセミコロン（;）で終わっているような 1 行を、ひとまず「文」と呼ぶ。

メソッドとは、オブジェクト指向プログラミングの用語で、オブジェクトに関連づけられた関数だと理解すれば良い。言い換えると、オブジェクトを操作対象とする関数であることを明示している記法である。オブジェクトとは、プログラム内での何らかの実体を表す「モノ」である。Javascript では、オブジェクト毎に処理をさせるメソッド群が定義されており、必要に応じて、オブジェクトと処理内容のメソッドを指定し、詳細を定めるパラメータを引数を介して渡すことで、意図した動作をさせる。

オブジェクトに対しては、プログラマが自身で新規のメソッドを追加で定義することもできる。function キーワードは、そのための定義を始める、という宣言であるといえる。

8.1.3 オブジェクト

オブジェクトを正面から説明することは難しい。直訳で object は「もの」などとされているが、これでは不明確である。「操作対象（物）」という程度に捉えると良いかも知れない。プログラミングにおける「操作」とは、データの取得、記憶、計算などの処理であるから、オブジェクトはこれらの「操作」を行うことができる「もの」と捉えれば良い。

例えば、Web ブラウザは通常 window 表示されている。この window 自体への「操作」を考えると、window を開く、閉じる、位置を変える、などが考えられる。もう少し、window への操作というものを考えてみると、window 内の表示位置を変える（スクロール）や、この window からの警告ダイアログの表示、ユーザからの入力待ちダイアログの表示、なども考えられる。これらは、実際に Javascript で window に対して操作可能な操作の例である。

Web ブラウザ上で動作する Javascript では、組み込みのオブジェクトとして実行されている window を指し示す、window オブジェクトを持つ。

8.1.4 window オブジェクト

window オブジェクトは、プログラム中で window として参照可能な window オブジェクトである。実際の Web ブラウザの window をプログラムから操作するための仕組みが組み込まれている。これを、window は、Web ブラウザのウインドウ自体を示すオブジェクトで、Web ブラウザ上で実行される際には、実行環境が事前に準備して利用可能な状態として提供される、のように言う。

例えば、window.close() とすると、window を閉じることができる。試してみよう。

このように、オブジェクト名（ここでは、window）に続けて、ドット（.; ピリオド）を打ち、その後指示する操作名を記述することで、その操作を行うことを指示することになる。このときの操作を行うプログラム上のものを「メソッド」と呼び、このオブジェクトに関連づけたれた関数と同義といって良い。組み込みオブジェクトでは、ブラウザ提供者が定義した操作の内容である関数定義が別にあり、その関数をメソッドとして「呼び出す」、すなわち、その関数に記述された処理手順を順に実行することで、そこでの操作を実現する形に

えられる。予約語に重なる語は使用できない場面があるため、一般的な語を避けたプログラミングが安全である。

なる。

8.1.5 オブジェクトのプロパティ

オブジェクトは、状態を表す属性情報を保持する容れ物としてプロパティと呼ばれるものを任意の個数持てる。プロパティには、数値や文字列、他のオブジェクト、メソッドなどを保存できる。各プロパティは識別のため名付けることができて、これをプロパティ名などと呼び、通常はこの名前で保存容器としてのプロパティを指定する。

たとえば、abc と名付けられたオブジェクトに def と ghi という 2 つのプロパティがあったとすると、それぞれ、abc.def、abc.ghi のようにオブジェクト名とプロパティ名をドット（.; ピリオド）でつないで記述する。

プロパティは空の場合もあり、数値や文字列などの単純なデータらしいものが入っていることもある。また、他のオブジェクトやメソッドも入れられる。このような容れ物（箱）として見た場合、プロパティは変数と呼ばれる。中身が変わることのあるデータ（数値だけに限らない)、という意味合いである。

気をつけなければならないのは、他のオブジェクトやメソッドは、大きすぎて直接プロパティには入らない点である。たとえばメソッドは、手順書のようなものであり、長い紙に多数の手順を連ねたようなものであると言える。これは、小さな容器であるプロパティ（変数）にはそのまま入らない。そこで、このようなものは、本体（ここではメソッド定義；手順書）は棚にしまっておき、変数には、「棚のどこどこの位置に置いてあるもの」というような小さな紙片だけを入れておくような処理が行われる。この格納位置を指し示していることを「参照」と呼ぶ。

オブジェクトのプロパティにメソッド定義を保存（代入）してやると、そのプロパティ名を使って、このメソッドを呼び出す（実行）することができる。Javascript の処理系は、プロパティに保存された参照を使って、メソッド定義の本体を見つけ、それをプログラムとして実行するのである。

このことを模式的に示したのが図 8.2 である。オブジェクトは、多数のプロパティを持つことができ、それぞれのプロパティは名前（プロパティ名）によって識別できる。プロパティは、単純な数値や文字列、メソッドなどを持つことができるが、ごく簡単な数値などのほかは、オブジェクト外部のメモリ領域の場所（address）を指し示すコンパクトな情報だけを暗黙に持っている。プログラムから利用するときは、このメモリへの参照は自動的に解決されて、参照先メモリ内容自体をあたかもプロパティとして持っているかのように振る舞う。

8.1.6 alert() メソッド

alert メソッドは、window オブジェクトに Javascript 処理系[*3]が提供しているメソッドである。このメソッドは、呼び出すと、引数の文字列と、「OK」ボタンのついた警告ダイアログボックスを表示する。

8.1.7 文

オブジェクト名とメソッド名を.（ピリオド）でつなぐと、そのメソッドが呼び出される。このような単位を文と呼び、;（セミコロン）で区切ることで複数の文を並べることができ、上から順番に実行させることができる。

[*3] 正確には、Web ブラウザ上の処理系が提供するのが、window オブジェクトなどである。alert メソッドは、window オブジェクト中のプロパティの一つである。

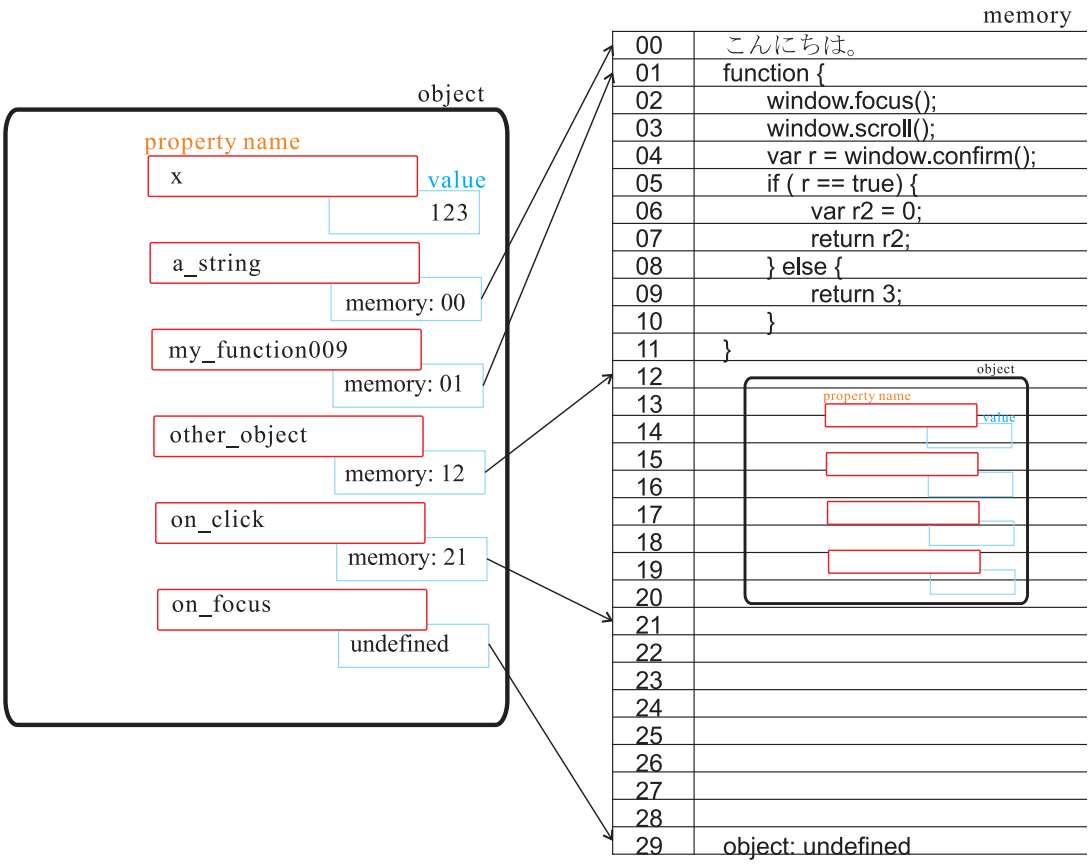


図 8.2　Javascript のオブジェクトの概念図

Javascript の文は、広範囲に渡ってのセミコロンの省略を許容する。しかし、文法を正確に理解していないと誤った解釈となる恐れがあるため、文の末尾にはセミコロンを置き、また、途中では改行しないようにした方が良い。

文には、条件判断を行わせて、処理内容を分岐させることのできる `if` 文や、一定の条件下で、記述した処理内容を繰り返し処理させる `for` 文なども用意されている。基本的な書式は、C 言語に類似しているが、Javascript 独自の拡張も見られる。

8.2 HTML からの呼び出し

第 07 章での設定を思い出してみよう。

1. 図 8.1 のコードで、処理本体を記述した。
2. HTML 側に、その Javascript ファイルを読み込むよう、`script` 要素を記述した。
3. 特定の HTML 要素をクリックしたときに処理が実行されるように、HTML 要素の `id` 属性で Javascript 側から参照できるように名前をつけ、Javascript 側で、その名前の要素について処理されるよう設定した。

この最後の処理は、次のようなものであった。

1. 6 章の演習問題で、一つだけ色を変えた `h2` 要素の `id` 属性値を記録する。
   - ここでは、その `id` 属性値が `sample1` であるとして説明する。
   - 例) `<h2 id="sample1">...</h2>`
2. 図 8.3 のコードを、`ch08/ch08.js` の JavaScript コードに追加する。

```
window.onload = function()
{
  /* sample1 イベントハンドラ設定 */
  var node_sample1
    = document.getElementById('<id 属性値>');
  node_sample1.onclick = sample1;
}
```

図 8.3　HTML 要素との関連付け

`ch08/ch08.html` をブラウザで表示させ、該当する `h2` 要素をクリックすると、先のメソッド `sample1()` が呼び出され、'Hello, world!' と書かれたダイアログウィンドウが表示される。

8.2.1 解説

`window.onload` は、window オブジェクトのイベントハンドラ[*4]の一つで、Web ページの読み込み (load)

[*4] 厳密に言えば、window オブジェクトの特別なプロパティの一つで、ここで指し示されるメソッドが特定のタイミング（イベント

が完了した時に呼び出される一連の処理を設定する。ここでは、function() として、以下に定義する処理を呼び出させるようにしている。

波括弧ブロック内は、まず、C 言語スタイルのコメント文がある。

次の var は変数の宣言を行っている。変数（オブジェクト）node_sample1 を宣言し、引き続き代入演算子によって、値を割り当てている。割り当てられるべき値は代入演算子の右辺にある、メソッド呼び出しの結果（戻り値）である。

document.getElementById('<id 属性値>') は、<id 属性値>を属性 id に持つ、文書内で唯一（それが、id 属性の意味であった）の HTML の要素を取り出す。したがって、変数 node_sample1 には、その要素が代入される。[*5]この要素は、Javascript 上ではオブジェクトとして取り扱われる。すなわち node は、その HTML 要素データを含むような Javascript オブジェクトである。

node_sample1.onclick は、その取り出した要素の onclick プロパティを指す。すなわち、node オブジェクトが保持する onclick プロパティである。

オブジェクトの種類によって、特定のプロパティに特別な役割を持たせてあることがある。HTML の各要素を示すオブジェクトの onclick プロパティも、そのような特別な役割を割り当てられたプロパティの一つである。onclick プロパティは、その要素がマウスでクリックしたときに呼び出される関数を保持する役割を持つ。逆に言えば、このプロパティの中が空（undefined）なら、この HTML 要素はクリックされてもなにも起こらないが、このプロパティが何らかの処理手順を記述したメソッドを指し示していると、クリックされたときにそのメソッドが呼び出される（実行される）。

したがって、ここでは、sample1 というメソッドが呼び出し先として登録される。onclick はマウスクリック時のイベントハンドラ（を保持する箱）というわけである。

8.2.2 イベント一覧

どのようなイベントが取り扱えるのか、主な一覧表を表 8.1 に示す。マウス、キーボードの操作や入力がイベントとして検知できることがわかる。設定可能な要素や制御手順に制約があるが、それらについては割愛する。

8.3 変数と演算子

Javascript の基礎的な文法として、変数と演算子について説明する。

8.3.1 変数

オブジェクト内のプロパティは、各種のデータを保持できる。このようなデータの容れ物は変数と呼ばれる。変数には、オブジェクトを格納できる。逆に言えば、オブジェクトを格納する容れ物に名前を付けて、その名前の変数として参照している、と言ってもよい。オブジェクトの実装上の定義としては、メモリ上の一定の領域を占めるデータ、と言い換えても良く、このオブジェクトをプログラム内で参照するための名前付き容れ物が変数であると言える。実務上は、変数とオブジェクトは同一視されていることも多い。

発生時）で呼び出されるというものである。このようなイベント時に実行されるメソッドのことをイベントハンドラと呼ぶ。

[*5] 気をつけるべきは、要素単位で取り出される点で、例えば、body 要素を取り出したとすると、そこにふくまれる本文全体を指すことになる。

オブジェクトに属していない（ように見える）変数を用いることもできる。[*6]

```
var x;
```

これは、単純な変数利用の例である。ある変数をはじめて使用するときは、var キーワードを用いる。var x で、‘x’ という名前のラベルを貼り付けたの変数（箱；容れ物）を用意しろ、という意味になる。これを、var キーワードによって変数を宣言する、という。[*7]

8.3.2 代入と演算

```
var y = 20;

var myString = 'This is a pen.';

var a = y + 20;
```

変数に値（データ）を設定するためには、代入演算子（=）を用いる。数学の等号の用法とは異なり、左辺の変数に右辺の計算結果を格納する、という意味になる。変数の宣言と併せて 1 文で書いてもよい。一般に変

[*6] 実際には、記述してある点で暗黙のオブジェクトの内部を操作していることになっているが、記述上、独立して存在するような変数が便利であり一般的であるため、このような記法がとられている。

[*7] 宣言しないで変数を使用し始める記述も規格上は許容されている。しかし、後述するスコープの問題から実務的には推奨されない。

数の初期化などと呼ばれる。

```
var x = 5;
x = 2;
```

変数には、複数の値を格納することはできない。このように複数回の代入を行うと、最後の代入の効果だけが最終結果となり、このとき x の値は 2 である。

8.3.3 算術演算子

```
var a = y + 20;
var m = h % 60;
```

算術演算子は算数で習った四則演算に加えて割り算の余りを計算する剰余演算子などがある。初歩的な演算子の一覧を表 8.2 に示す。これらは、数値に対して有効である。[*8]

表 8.2 初歩的な演算子

演算子	処理内容	記述例	意味
=	左辺に右辺を代入	a = b	
+	足し算	a + b	
-	引き算	a - b	
*	かけ算	a * b	$a \times b$
/	割り算	a / b	$a \div b$
%	剰余	a % b	a を b で割った余り
++	加算	a++	a に 1 を加えて新たな a の値にする
+=	代入加算	a += b	a = a + b に同じ

8.3.4 データ型と文字列演算子

データ型と演算子を意識する事例として、文字列について簡単に触れておく。

```
var s1 = 'Hello, world!';
var s2 = "Hello, world!\n";
var s3 = 'He said "Yes!".';

var concatinated = s1 + s2;
```

文字列は、シングルクォート（'）か、ダブルクォート（"）を用いて囲むことで記述できる。両者の違いは、エスケープ文字を用いて特殊な文字を記述できるか否かである。s2 では、文字列の最後の改行記号を挿入してある。特殊文字の記法ではこの改行記号（\n）が最もよく使われる。

引用符自身も同様にバックスラッシュ（日本語環境では円記号（¥）であることが多い）でエスケープすることで文字列中に含むことができる。しかし、単純な場合には、文字列定義で用いていない引用符記号を用いれば簡単に記述できる。変数 s3 の例を見よ。

[*8] 演算子の説明を厳密に行うと、非常に多種の演算子を導入することになる。また、データ型についての網羅的な説明も必須である。ここでは、基本的な四則演算に限って導入する。また、データ型については曖昧な書き方のままとする。容赦されたい。

文字列は、文字列として連結することができ、このとき、演算子（+）を用いる。+ 演算子は対象が文字列か数値かによって振る舞いが異なるので注意が必要である。[*9]

8.4 演習問題

1. `ch07/ch07.html` の Hello, world プログラムの出力メッセージを例えば、”2+3 は 5 です。”のように計算を含むようにしてみよ。このとき、計算結果の’5’ は’2+3’ のようなプログラム内での演算結果となるよう記述すること。
2. 設定した `h2` 要素は、Web ブラウザで見てわかるよう、説明を加えておくこと。
3. 他の要素に対しても、マウスクリック以外のイベントに反応する仕掛けをコーディングすること。説明文も忘れずに。
4. Valid[5] であり、Web ブラウザを通じて HTTP でアクセスできることを確認。

[*9] 適用規則は厳密に存在するが、人間が判断を間違えるような微妙な条件ではプログラムを書かないことが望ましい。なぜなら、他の人が読んだ際に、どのような演算が行われるか誤読を生むからである。このとき、「他の人」には書いた本人、あなたの数週間後の姿も含まれる点に注意！

表 8.1　主なイベントハンドラ

イベント名	動作
onchange	フォーム状態が変化した
onclick	オブジェクトがクリックされた
ondblclick	オブジェクトがダブルクリックされた
onfocus	オブジェクトフォーカスされた
onblur	オブジェクトのフォーカスが外れた
onkeydown	key が押された
onkeyup	key が離された
onkeypress	key が打たれた
onload	page/画像が読み込まれた
onunload	page 移動、閉じた、再読込
onmousedown	オブジェクト上でマウスボタンが押された
onmouseup	オブジェクト上でマウスボタンが離された
onmouseover	オブジェクトにマウスポインタが重なった
onmouseout	オブジェクトからマウスポインタが離れた
onmousemove	オブジェクト上でマウスポインタが移動した
onsubmit	フォームの送信ボタンが押された
onreset	フォームの取消ボタンが押された

表中でのオブジェクトは、Javascript 上の当該オブジェクトであり、すなわち、HTML の要素を指す。

第9章

WebでのjavaScript

9.1 Javascriptの実行モデル

9.1.1 オブジェクト指向

Javascript はオブジェクト指向言語である。これは、プログラムの主要構成要素がオブジェクトであり、このオブジェクトへの操作[*1]を記述することで、目的とする処理を実現する。

オブジェクト（object）とは、単にモノを指すような抽象的な用語である。オブジェクト指向言語では、実体（プログラムの外側）世界における何らかのモノ、コトを、モデル化した存在としてオブジェクトを定義する。このモデル化が、実体世界のモノの特徴だけを捉えて、何らかの実物と結びつけない状態であるならば、そのオブジェクトはプログラム中のみの存在であり、振る舞い（動作）がモデル化したモノのように振る舞うだけで、実物に影響が及ぶことはない。しかし、実行環境（実行システム）が何らかの実体と結びつけてオブジェクトを提供しているならば、プログラム中のオブジェクトに対する操作は、実物を操作することに等しい。

例えば、Web ブラウザで動作している Javascript の実行環境は、Web ブラウザの window や、読み込んだ document、その HTML の各要素などをオブジェクトとしてモデル化した上で、提供している。したがって、プログラマは、このオブジェクトに対して操作を行えば、Web 情報システムのクライアント側にある実物を操作することができる。すなわち、HTML を自在に操作し、各要素の style を操作し（したがって、css を操作するのに等しい)、window を操作できる。

9.1.2 Event driven

また、Javascript では、利用者による対話的な利用を目的として利用者の様々なアクション、すなわち、マウス操作（クリック、ダブルクリック、移動、オブジェクトへの重なりなど）や、それに伴うオブジェクトの状態変化に対して、あらかじめ定義した処理内容を実行させることができるようになっている。これは、利用者の行動とは無関係に、一連の処理内容を実行させる、いわゆるバッチ型処理の対極にある実行モデルである。

このような動作モデルを event driven(イベント駆動) と呼ぶ。GUI 画面はその代表的な実装例である。

画面上の各要素を映し出すプログラム内の各オブジェクトには、利用者からのアクションに対する処理内容が事前に設定されている。利用者が画面を操作、すなわち、マウスカーソルの移動や、特定の画面要素上に乗った、外れた、マウスボタンのクリックなどのアクションは、イベントと呼ばれる情報のかたまりとしてプ

[*1] オブジェクト指向の世界では、これをオブジェクトへのメッセージ送信などと呼ぶ。

ログラムに通知される。

プログラム側で、事前にどの種類のイベントに反応したいかを記述しておくと、実行環境側が、イベント内容を示すオブジェクトを用意し、そのイベントに反応するために記述した関数を呼び出していく。たとえば、画面上のボタンの画を押し込まれた状態に変更する関数を用意しておき、マウスクリックのイベント発生時にこの関数が呼び出されるように設定しおく。すると、利用者が画面上のそのボタン領域をクリックすると、画が押し込まれたものに変化するため、利用者はあたかも画面上での感覚、ここではボタンを押したかのような、を得られる仕組みである。

図 9.1 は、第 08 章でのサンプルプログラムがどのような状態で動作しているのかを示す概念図である。実行環境[*2]は、Web ブラウザとしての動作を提供するために各オブジェクトに対する、各種イベントへの処理内容を設定できるプロパティを持っている。そこで、目的に適ったプロパティへ、実行させたい処理内容を function キーワードを用いて生成した、関数オブジェクト[*3]を設定してやることで、そのイベントの発生時に、その関数が実行される。

イベント処理用の関数を設定するという特定用途を持たされたプロパティのことを、イベントハンドラと呼ぶ。各オブジェクトは、各種のイベントハンドラを持つため、複数のイベントを受信し、処理を実行することができる。

9.2 イベント駆動の事例

処理対象となる要素（オブジェクト) が、イベントを受信したオブジェクトとは異なっていても良い。

次の処理本体の sample2_over() は、受け取った event（文字列化した内容）を「サンプル」という文字列と一緒に、id='target2' なる別の HTML 要素の内容に設定している。

```
// Sample 2
function sample2_over(event)
{
  var target = document.
    getElementById('target2');
  target.innerHTML = ' サンプル ('+event+')';
}

function sample2_out(event)
{
  var target = document.
    getElementById('target2');
  target.innerHTML =
    'Javascript プログラミング入門
    (sample2 実行済み)';
}
```

innterHTML() メソッドは、HTML の各要素のオブジェクトに用意されたメソッドで、その要素内の (X)HTML テキストを設定し直すことができる。言い換えると、その要素より下に位置する部分木（sub-tree）を、HTML 文字列の形で与えることができる。

また、sample2_out() では、同じ要素の内容を別の文字列に設定している。

[*2] この場合 Web ブラウザ

[*3] 処理内容の命令書を示すオブジェクトと考えると良い。

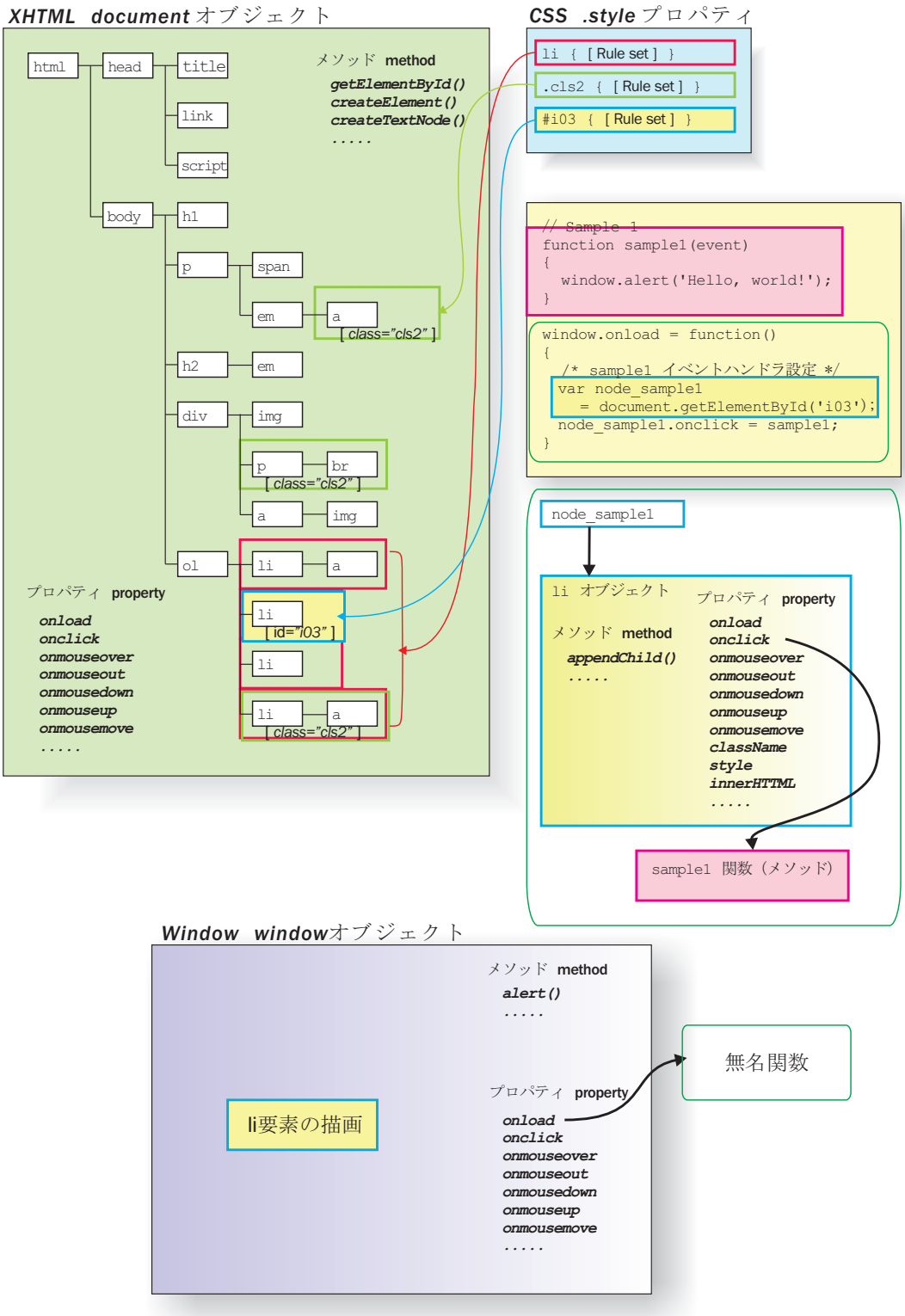


図 9.1　Javascript の動作概念図

この 2 つの関数が呼び出されるタイミングを決定するのがイベントである。逆に言うと、イベントが発生したときに関数で定義された処理が実行される。処理がイベントによって駆動されるイメージであり、したがって、イベント駆動と呼ばれる。

```
window.onload = function()
{
  /* sample2 イベントハンドラ設定 */
  var node_sample2 =
    document.getElementById('sample2');
  node_sample2.onmouseover = sample2_over;
  node_sample2.onmouseout  = sample2_out;
}
```

イベントと呼び出される関数を結びつけるのがイベントハンドラの役割であり、そのためのコードが上の例のようになる。ここでは、onmouseover と onmouseout の 2 つのイベントに対して、それぞれ別の処理本体関数を設定している。イベントハンドラの設定は、id="sample2"オブジェクトであり、操作対象の id="target2"とは異なるオブジェクトである。このように、イベント発生時に呼び出された関数では、どのようなオブジェクトを対象とした処理を行うこともできる。

9.3 メソッド、関数

Javascript では、メソッドと関数は同一のものを指す。オブジェクトとの関連性が高い状況での関数を指す場合に特にメソッドと呼ばれることが多い。

次の簡単な関数について、関数の記述や意味を説明する。

```
function add(a , b)
{
  var result = a + b;
  return result;
}
```

関数の記述は、function キーワードから始める。続いて、関数名を書く。ここでは、add である。関数には、処理対象とする値を渡すことができ、それらを引数と呼ぶ。

関数本体内で、この引数を参照して計算や処理を進めるため、関数内で参照するための名前を与えてやる必要がある。それが、関数名 add に続く丸括弧内の a, b である。ここでは、a と b の 2 つの変数名を用いて、関数への引数を処理することを宣言している。このような変数のことを仮引数と呼ぶ。

関数本体では、処理手続きを順に文として並べて記述する。後述する制御文を組み合わせて複雑な手順を構成することもできる。ここでは、新たに変数 result を宣言し、そこに a + b の結果を代入している。

最後の return キーワードは、関数からの戻り値を返すための return 文であることを示す。ここでは、変数 result を戻り値としている。戻り値は引数とは異なり、1 つしか返すことができない。[*4]

この関数 add は、次のようにして呼び出される（実行される）。

[*4] しかし、オブジェクトに多数のプロパティを格納できることを考えれば、一つのオブジェクトに多数の戻り値に相当する値を別々のプロパティとして格納して、この一つのオブジェクトだけを戻り値とすれば、事実上いくつでも戻り値を返せることに気づくだろう。また、配列と呼ばれる仕組みを用いることでも、同様のことができる。ただし、本書では、具体的な記法については触れない。

```
var x = 1;

var y = add(x, 2);
```

add の 2 つの引数について、最初の引数には、変数 x、2 番目の引数には、数値 2 を渡している。これらは、それぞれ add 内では、仮引数 a，b として参照される。戻り値は、代入文を通じて別の変数 y に代入されている。結果として、y の値は 3 となる。

```
var addFunction = add;
var answer = addFunction(3, 4);
```

関数は変数に格納することもできる。

9.3.1 識別子

変数名や、関数名には命名規則が決まっている。

1. 1 文字目は、英字（大文字、小文字）、アンダースコア（_）、ドル記号（$）
2. 2 文字目以降は、上記に加えて数字が使える
3. 大文字と小文字は区別される
4. Javascript の予約語ではないこと

予約語は、break，case，catch，continue，default，delete，do，else，finally，for，function，if，in，instanceof，new，return，switch，this，throw，try，typeof，var，void，while，with であるが、ほかにも null，undefined など避けなければならない語もある。予約語は Javascript の構文上意味が与えられている語である。その他避けなければいけない語は、特定の意味を与えられた識別子である。

9.4 Javascript の制御構造

javascript はオブジェクト指向の手続き型言語に属する。そのため、一つの関数（method）内では、記述したコードの上から下へと処理が進む。この処理の流れを変更する構文が制御構造である。

C 言語類似の制御構造を基本に持ち、while 系のループ処理や、switch 文なども持つが、ここでは、最も基本的といえる、C 言語型 for ループ制御文と、if 制御文の概要を説明する。

9.4.1 if

if の基本構文は、次のようになる。

```
if（条件）
{
  // 条件が真のときの処理内容
}
else
{
  // 条件が偽のときの処理内容
}
```

条件には様々なパターンを記述可能だが、最も基本的な数値の条件式は、`i == j` のように書く。ここで、==が条件演算子と呼ばれる、条件の意味を指定する部分で、この場合、変数 `i` と `j` が同じ値であるとき、条件は真となる。条件演算子には、数学と同様の <、> のほか、等号の条件を含む <=、>=、また、「等しくない」という条件を示す!=などがある。

```
var a = 1;
var b = 2;

if（a > b）
{
  // この値では実行されない
}
else
{
  // a の方が小さいため、ここが実行される
}
```

9.4.2 for ループ

C 言語由来の `for` ループを Javascript でも使うことができる。このループは多用される傾向があるが、理解することが難解なループでもある。その理由は、ほとんどあらゆる制御をこのループの形態を使って記述可能な点にある。すなわち、極めて柔軟性が高いが、故に複雑なのである。正しく理解しないと、意図したとおりに動作しないため十分注意することが必要である。

```
for(初期化部; 条件部; 更新部) 文;
```

`for` ループの構文は上記のとおりである。ここで、最後の文は for ループとは無関係に複数の文の並びからなるブロックでも良いことに注意すると、

```
for(初期化部; 条件部; 更新部)
{
  // 処理内容
}
```

のように書け、この書式が一般的である。

この制御構文がどのように実行されるかという順序を示すと

1. 初期化部
2. 条件部 → 偽なら次の文へ

3. 文[*5]
4. 更新部
5. 2 に戻る

のようになる。

固定回数だけ文の実行を繰り返す慣用句として、次のような構文が見られる。ここでは、固定回数を 3 回とした例を示すと、

```
for(i = 0; i < 3; ++i) 文;
```

のような構文である。この構文で、なぜ 3 回だけ文が実行されるのか、実行順序を順に追って確認してみると良い。ちなみに、`++i` というのは、`i = i + 1` の略記法で、インクリメント（increment）演算子 `++` を用いた記法である。

increment/decrement 演算子の前置と後置

increment 演算子 `++`、decrement 演算子`--`には、前置記法と後置記法がある。たとえば、`++i` と `i++` の両方の書き方ができる。両者の意味はほとんど同じで、単独で使われるならば違いはない。式の値として評価されるタイミングが両者の違いである。前置演算子では演算が行われた後の値が評価される。一方、後置演算子は演算が行われる前に評価される。
例を使って確かめるのが良い。ここで、`p = 3` であったとする。
`x = ++p` では、`x` の値は 4 である。`x = p++` では、`x` の値は 3 である。
この違いを用いたコーディングは一般的に行われているが、わかりにくいコーディングスタイルに属する。初学者のうちはこれらの違いに頼るようなコードは書かない方が良い。

9.5 オブジェクト

9.5.1 オブジェクトとハッシュ

すでに述べたように、JavaScript のオブジェクトとは多数のプロパティを一括りにする器であるといえる。各プロパティには、数値や文字列、他のオブジェクトを格納できる。オブジェクトは、このように複雑な構成を持つことも可能だが、データを保持する器という意味合いでは単に「変数」と呼ばれることも多い。

オブジェクトには、イベントハンドラのためのプロパティのような特殊なプロパティもあるが、単にデータの容れ物として見た場合は、プロパティ名によって識別できる多数の変数の集まり、と考えることができる。これは、プロパティ名をキー（key）として値（value）を格納する器と言い換えることが出来、このような機能を実現する実装に由来してハッシュ（hash）とも呼ばれる。

JavaScript においては、「オブジェクト」と「ハッシュ」は等価である。[*6]オブジェクトの各プロパティ、すなわちハッシュの各キーに対応する値にアクセスする方法は、これまでに見た、

```
オブジェクト名 . プロパティ名
```

だけでなく、

[*5] または、ブロック
[*6] ハッシュは、歴史的経緯により「連想配列」とも呼ばれる。

```
オブジェクト名 ['<プロパティ名>']
```

という形式を使うこともできる。

また、オブジェクトの定義時には、オブジェクトリテラル形式という記法も用いることができる。

```
var my_obj1 = { prop1: 1, prop2: 234, prop3: 'hello, world!' }; // 右辺がオブジェクトリテラル形式
window.alert(my_obj1.prop1); // 1 が表示される
window.alert(my_obje1['prop3']); // hello, world! が表示される
```

この例では、prop1，prop2，prop3 の 3 つのプロパティ名を持つオブジェクトを定義している。それぞれのプロパティ名に対応する値は、コロンで区切って記述される。

9.5.2 配列

ハッシュを用いれば、多数のデータを一つの変数（オブジェクト）に格納して統一的に取り扱うことが可能である。しかし、ハッシュキーには任意の識別子名をつけることができ自由度が高いため、コンピュータの処理上効率が悪い。そこで、単に連番の自然数によって識別できれば良い場合のために、配列と呼ばれるものがある。

配列は、

```
配列名 [<インデックス番号>]
```

によって参照される。インデックス番号は 0 以上の自然数である。また、配列リテラルとして定義することも可能である。

```
var my_array = [200, 'OK', 'Created', 'Accepted'];
window.alert(my_array[1]); // OK が表示される
window.alert(my_array[2]); // Created が表示される
```

9.5.3 データ型

これまで述べてきた、数値、文字列、オブジェクト、配列はそれぞれ異なる性質を持ち、このような性質毎のもののことをデータ型と呼ぶ。この呼び方によれば、これらはそれぞれ、数値型、文字列型、オブジェクト型、配列型と呼ばれることとなる。

例えば、次のように特性を述べることができる。数値型は四則演算を行うことができる。文字列型は連結演算子を持つ。オブジェクト型はプロパティを持ち、ハッシュとして機能する。配列型はインデックス番号により識別される均質的な多数のデータを格納できる。

JavaScript ではここまでで明示的に触れていないデータ型も存在するが、データ型として初学者が意識すべきはこの程度で良い。

9.5.4 複雑な変数

これまでの記法を用いると、次のような複雑なデータ構造を持つ変数を定義することもできる。どのようになっているかわかるだろうか。

```
var complex_data = {
  my_string: 'This is a string.',
  two_dim_array: [ [0, 1, 2], ['zero', 'one', 'two' 'three'], 'end of array'],
  other_object: { x: 120, y: 30, color: '#40ff80'  }
};
```

また、具体的に各要素へアクセスしようとするにはどのような記法が必要だろうか。調べてみよう。

9.6 スコープ

次のようなコードを考えてみる

```
var a_variable = 'first';

function my_function()
{
  window.alert(a_variable);  // undefined が出力される
  var a_variable = 'second';
  return a_variable;
}

window.alert(my_function()); // second が出力される
window.alert(a_variable); // first が出力される
```

ここでは、変数 a_variable を 2 カ所で定義している（ように見える）。ここで、関数 my_function() 内で定義された a_variable は、この関数内のみで閉じて参照される。このように変数が参照されうる範囲のことを「スコープ（scope）」と呼ぶ。

これは次のように理解すると良い。まず、1 行目の a_variable は、「全体の a_variable」である。一方、6 行目は、「my_function() の a_variable」である。このように考えると、決して同じ a_variable という名前の変数ではなく、それぞれが別物であることがわかる。

ここで、「my_function() の a_variable」は、my_function() 関数の定義内でしか参照することができない。このように特定の関数内にスコープが限定されている変数のことをローカル変数と呼ぶ。ここでは、ローカル変数としての a_variable は、関数 my_function() 冒頭では、まだ値を割り当てられていない。そのため、undefined が出力される。[*7]この関数の戻り値は、return 文によってもたらされるが、このときの

[*7] このように関数ブロックの途中で定義しても、その冒頭からがスコープとなるのは JavaScript 特有の挙動である。

値はローカル変数として割り当てられた `second` である。しかし、関数の外側では、プログラム冒頭で定義された変数が依然として参照対象となり、`first` が出力される。

このように、プログラム全体から参照されうる変数のことをグローバル変数と呼ぶ。[*8]

9.7 演習問題

`http://intra.ns.kogakuin.ac.jp/~ct13213/cs1/starting-js.html` のプログラムを参照して答えよ。

1. Sample 3 がどのように動作するかの説明を HTML 上で記述せよ。
   - 1 行毎に説明すること。わからない行はその旨書いておくこと。
   - for ループを「〜繰り返し」「〜に戻る」などと単純化して説明しないこと。具体的な変数がどのように変化するかを解説せよ。
2. Sample 4 では、HTML 要素に適用する css の rule set を class 名を書き換えることで丸ごと切り替える方式を用いることで、表示を切り替えている。同様の方式を用いて、サンプルとは異なる見栄えを与える効果を適当な要素に与え、その説明分をわかりやすく書け。
3. その他独自の効果を工夫してみよ。

[*8] `var` キーワードを用いずに変数宣言もできるが、その場合グローバル変数と見なされる。この挙動は思わぬ動作を招きやすいため、`var` キーワードは省略しないことを習慣とすると良い。

第10章

イベント

10.1 Javascriptのイベント駆動モデル

Webブラウザ上のJavascriptは、主文書としての(X)HTML文書を持ち、この文書を操作することが主目的となる。主文書の(X)HTMLでは、`script`要素により、操作手順書となるJavascriptプログラムファイルを指し示すことで、Webブラウザは、両者を関連づけて実行することができることは既に学んだ。

このようなWebブラウザ上のJavascript実行環境では、ブラウザwindowを指し示すオブジェクト`window`や、主文書を指し示すオブジェクト`document`が、実行環境（Webブラウザ内のJavascript実行処理系）によって用意されている。そのため、`window`や`document`オブジェクトの各プロパティを用いて、主文書やそれを表示しているwindowの振る舞いに指示を与え、操作することができる。

10.1.1 イベントハンドラ

Javascriptでは、マウスクリックなどのイベント発生時に処理すべき内容を記述しておくことで、イベントに応じた動作を指示することができる。このようなイベントに対応して、実行する内容を変化させるようなプログラミングの考え方をイベント駆動（event driven）と呼ぶ。画面上で描画される領域に対応するオブジェクトには、マウスやキーの状態変化に応じたイベントハンドラが予め設定されている（表8.1）。

10.1.2 イベントハンドラの登録

実際にイベント発生時に行わせる処理を、各イベントハンドラに登録するためには、`window.onload`イベント発生時が適切である。このイベントは、`window`オブジェクトに全ドキュメントが読み込まれたときに発生するため、必要なファイルが読み込み途中で設定できない、といったことが起きない。

HTML要素へのイベントハンドラ登録

HTML要素のイベントハンドラを登録する際は、登録対象のHTML要素オブジェクトを取得する必要がある。HTML要素オブジェクトの取得で最も簡単なのは、(X)HTML文書で予め`id`属性により与えておいた文書中唯一の名前で検索する方法である。この方法のためには、`document`オブジェクトのメソッド`getElementById()`を用いる。与えるべき引数は、`id`属性名そのものである。

ここで得られたオブジェクトの、意図するイベントハンドラに実際に処理させる関数を設定すれば良い。

10.2　タイマイベント

JavaScript のイベントは、利用者が何らかの操作をしたときに発生するものが多かった。しかし、コンピュータ処理全般では、一定時間が経過したとき、あるいは、一定時間間隔で定期的に何らかの処理をさせたいことも多い。

例えば、アニメーション表現が挙げられる。これは、数分の 1 秒から数十分の 1 秒程度の間隔を以て、定期的に表示画像を描き変える処理をさせることで、人に動きのある映像として知覚させる技術である。また、一定の制限時間を設定しておき、ゲームのタイムオーバー処理に用いるなどの応用も考えられる。

JavaScript では、このような時間の経過によって発生を予約できるイベントとしてタイマイベントが用意されている。タイマは任意個数生成することができるので、固定のタイマイベントハンドラは用意されていない。その代わりに、タイマを生成する際に、タイマイベント発生時に呼び出すべきプログラムコードをそのタイマに登録する。好きな数だけ目覚まし時計を買ってきて、それぞれの目覚まし時計に目覚まし時刻を設定しておき、そこに実行手順書（関数）を貼り付けておくイメージである。

10.2.1　タイマ

JavaScript のタイマには、2 種類ある。

1. 設定時間経過後に処理させる
2. 設定時間毎に繰り返し処理させる

この 2 種類である。処理内容を登録しておき、設定時間経過後（毎）にその処理が呼び出されることになる。設定時間後（毎）に処理を開始するイベントが発生するイメージである。

通常のイベントハンドラでは、設定したオブジェクトのプロパティへの割り当てをやり直すことで、そのイベントハンドラの登録を取り消すことができた。タイマイベントの場合には、専用のメソッド[*1]が用意されている。

表 10.1　タイマ関連メソッド

いずれも、`window` オブジェクトのメソッドである。

method	動作
`setTimeout(<処理文字列（プログラム）>, t)`	t[ms] 後の処理を設定
`clearTimeout(tid)`	`tid` での処理の待機を取消
`setInterval(<処理文字列（プログラム）>, t)`	t[ms] 間隔で処理実行を設定
`clearInterval(tid)`	`tid` での処理の待機を取消

そこで、登録と、削除のためのメソッドを表 10.1 に一覧する。ここで、処理文字列には、JavaScript のプログラム文そのものを記述する[*2]。長大なプログラムを直接記述することも可能だが、処理の見通しを良くするために、呼び出すべき関数を用意しておき、その呼び出し文だけを記述しておく方が良い。

また、待機中のタイマ処理を取り消すためには、あらかじめ設定 method（`setTimeout()`, `setInterval()`）を呼び出した際の戻り値を適当な変数に割り当てておく。ここでの戻り値はタイマオブジェクトであり、設定

[*1] window オブジェクトの提供する method である。

[*2] プログラム文字列ではなく（コンパイル時に処理される）関数を記述しておくことも可能であり、性能面ではそのほうが有利である。

情報を持ったタイマであると考えれば良い。取り消し method に、取り消すべきこのタイマオブジェクトを引数として渡して呼び出すことで、タイマ処理を取り消すことができる。記述例を図 10.1 に示す。

```
// myfunc(p1, p2) という関数呼び出しを 1000ms 毎に呼び出すことを登録
var tid = setInterval('myfunc(p1, p2)', 1000);

// 上のタイマを取り消し
clearInterval(tid);
```

図 10.1　タイマの設定と取り消し

10.3　タイマを利用した時計

タイマを利用して、1 秒毎に現在時刻を表示すれば時計になる。この考え方で時計を作成してみよう。

10.3.1　設計

HTML 側に、文字列を表示できる要素を準備しておく。

Javascript 側では、1 秒おきに現在時刻文字列を表示するようタイマを使って設定しておく。現在時刻文字列を得るためには、実行中のコンピュータから現在時刻を示すような情報を取得する必要がある。

10.3.2　HTML 要素

利用者が情報を入力するフォームと呼ばれる一連の HTML 要素がある。ここでは、比較的短い文字列を入力するときに使われるテキスト入力要素を用いて、その内容として時計を表示してみる。

```
<input type="text" size="50" id="clock" />
```

図 10.2　input 要素

input 要素の記述例は図 10.2 のようになる。input 要素は、inline level 要素であり、マークアップすべきテキストのない空要素として記述される。input 要素には、ラジオボタンやチェックボックスなど様々な入力フォーム用の形式が用意されている。この形式を指定する属性が type であり、ここでは、短いテキスト入力に向いた text を指定している。

input type="text"は、テキスト入力欄であるため、表示の際の大きさ（長さ）を指定できる。ここでは、十分な文字数が表示できるよう、size="50"として、50 文字分を確保している。

さらに、Javascript から操作するために、id 属性名として clock を指定している。

input type="text"では、その内容を設定するために、value 属性を用いることができる。

Javascript から、この value 属性を書き換えれば、任意の文字列を表示できることになる。

10.3.3　日付時刻オブジェクト

Javascript には日付、時刻の情報を、実行中のコンピュータ*3から取得する方法が用意されている。これを用いて、現在時刻の文字列を得ることにする。なお、あくまで実行中のコンピュータの時刻であるため、必ずしも正確な時刻とは限らず、場合によっては 10 年以上も異なる時刻かも知れない点は注意しておく。

Javascript では、日付と時刻の情報を保持し、計算などができるオブジェクトとして、Date オブジェクトが用意されている。スケジュール管理等で日付情報を用いたいときには、例えば会議の開始日時の情報を設定して、このオブジェクトを生成したりする。

Date オブジェクトを引数無しで生成すると、現在時刻が得られることになっている。そこで、ここでは、その生成方法を利用する。

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

図 10.4　現在時刻の取得

Javascript では、オブジェクトを文字列が必要な場面で用いると、予め設定された方法に従って自動的に文字列に変換される。Date オブジェクトは、英語による日付時刻表示文字列に変換される仕様となっている。

10.3.4　イベントハンドラ

適切なタイミングで呼び出されたイベントハンドラでは、どのような処理を行えば良いだろうか。

input 要素に表示される文字列は、value 属性値であった。そこで、input 要素の value 属性に適切な時刻文字列を設定（代入）すればよい。すなわち、1 秒ごとに表示文字列を更新していく。そのためには、input 要素が node オブジェクトとして参照できるとき、

```
node.value = current_date;
```

図 10.5　時刻文字列の設定

のようにする。

ここで、図 10.4 と図 10.5 のコードは、1 行でまとめて書くことができる。イベント発生時に処理すべき全処理内容は、それだけであるため、一つの関数として定義すると図 10.6 のようになる。

*3 すなわち、アクセスしている Web ブラウザが実行中のクライアント側コンピュータである。

10.3.5　イベントハンドラの登録

```
var node_clock;
node_clock = document.getElementById('clock');
setInterval('put_clock_time(node_clock)', 1000);
```

図 10.7　イベントハンドラの中核コード

ここで定義した関数 put_clock_time() を 1 秒間隔で呼び出すよう設定できれば、現在時刻を表示し続けるように見える[*4]。一定時間間隔でプログラムを実行させるため、setInterval() 関数によるイベント登録が適切である。

図 10.7 では、setInterval() 関数を用いた設定で、put_clock_time(node_clock) という文が、1 秒おきに実行される。したがって、node_clock 変数が適切な処理対象 input 要素のオブジェクトを参照していなければならない。そのため、事前に、node_clock 変数を宣言し、getElementById() メソッドにより、処理対象要素オブジェクトを設定している。

setInterval() 関数によるイベントハンドラの登録は、主文書が読み込まれた後、1 回だけ行われることが適切である[*5]。すなわち、他のイベントハンドラ登録と同様、window.onload イベントハンドラ内にて設定することが適切である。

すると、設定の全体コードは図 10.8 のようになる。

```
var node_clock;

window.onload = function() {
  node_clock = document.getElementById('clock');
  setInterval('put_clock_time(node_clock)', 1000);
  // ... 他のイベント登録などの処理
}
```

図 10.8　イベントハンドラの登録

[*4] 現実には、プログラム実行のための処理遅れなどでずれることもあるため、表示が 1 秒分飛ぶこともある

[*5] 複数回設定すると、別々のタイマイベント設定とみなされ、1 秒おきに設定しただけの回数、同じ関数が異なる処理として呼び出されることになる。今回の目的ではこれは処理の無駄である。

10.3.6　時計実現の全コード

```
var node_clock;

window.onload = function() {
  node_clock = document.getElementById('clock');
  setInterval('put_clock_time(node_clock)', 1000);
  // ... 他のイベント登録などの処理
}

function put_clock_time(node)
{
  node.value = new Date();
}
```

図 10.9　時計実現の全 JavaScript コード

サンプルコードによる実装例は、10 章のサンプルコードページとして用意してあるので、Web ブラウザで試すと同時に、(X)HTML、Javascript のソースコードについて確認すること。なお、同サンプルコードには、演習問題で必要となるタイマーのコードも含まれている。

10.4　演習問題

空の HTML5 ファイルを基に、css ファイル、JavaScript ファイルを関連づけ、それぞれ、`ch10/ch10.html`, `ch10/ch10.css`, `ch10/ch10.js` として準備する。

1. サンプルの時計を組み込んでみる。
2. サンプルのタイマーを適当な場所に組み込んでみる。
3. サンプルのタイマーを改変した減算タイマーを組み込んでみる。
   （a）ボタンを押すと残り 1 分から始まり、1 秒ずつ減算すること。
   （b）残り 0 秒で止まること。
   （c）残り 10 秒以下となると、表示文字を赤色文字に変化させること。
4. 秒数が 3 の倍数ならば、表示内容を変える、開始時刻を設定できる、などの効果や機能を独自に追加してみよ。

```
<input type="text" size="50" id="clock" value="sample" />
```

図 10.3　input 要素の value 属性

```
function put_clock_time(node)
{
  node.value = new Date();
}
```

図 10.6　時刻文字列の設定メソッド

第11章

Canvas入門

11.1　HTML5 と canvas

HTML5 に至る以前に普及していた XHTML では文書の論理構造のみを表現することに注力されてきたが、Web 情報システムにおいて、いわゆるリッチなインタフェース、アプリケーション構築を目指して HTML5 の仕様が策定されつつある。現在は草案段階であり規格が固まりつつある段階で、現在は詳細なレンダリング方式などが議論されている。しかし、多くのブラウザにおいて、その主要機能が実装されている[*1]。

ここでは、HTML5 の canvas 要素を用いた絵の描画を行ってみる。canvas は、Javascript によって描画要素を制御することを前提とした HTML 要素で、ブラウザ画面内に一定の矩形領域を確保し、その中に自由に絵を描画できる。描画は Javascript 内の canvas と関連づけられたメソッドによって行う。

11.1.1　再掲：空の HTML5

内容のない空の書式をファイルとして用意しておき、これに必要な内容を追加するのが良い。空の HTML5 を次に示す。

```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="UTF-8" />
<title></title>
</head>
<body>
</body>
</html>
```

図 11.1　空の HTML5 ファイル

XHTML と比較すると、XML 文書であることを宣言する DOCTYPE 宣言や、文書のメタ情報表現が簡潔となっている。また、HTML5 では、かつての HTML のように、要素を構成するタグの大胆な省略を認めて

[*1] また、XHTML1.0 の文書であっても、Web ブラウザでの描画に限って言えば、後述の canvas 要素を描画することも可能である。ただし、XHTML としては不正となるため、適切な DTD(文書定義) を読み込ませるなどするのが正当である。

いる。特に終了タグを明示しなくとも構わない場合が広範に規定されており、記述時の簡略化や、リソースサイズの低減が狙える。反面、XML として直接解釈することはかなわないため、XML への変換プログラム（converter）などの普及までは、XML としてコンピュータが解釈することが容易なように、また人手によるミスを減らすためにも終了タグも明示したほうが良いだろう。

図 11.1 では、日本語であること、utf-8 形式で記述することを定義している点が目立つ程度である。

11.1.2 canvas 要素

Canvas は、HTML5 で導入が見込まれている要素である。Working draft（草案）は、W3C により公開 [12] されている。

canvas 要素は、画面上の矩形領域を描画用に予約する。従って、矩形領域の大きさを指定する属性として、width、height がある。単位は、css pixel であり、css で指定されたピクセルを単位とする長さである。現時点では、通常 1 物理 pixel であると考えても良い。

また、内容（content）は、画像での表現ができない場合などに代替表示される内容となる。ここでは、画面描画を試してみることが目的のため、内容は空にしておく。すると、次のようなタグによって canvas 要素を定義できる。

```
<canvas id="drawingarea" width="640" height="480"></canvas>
```

ここでは、横 640、縦 480 ピクセルの矩形領域を描画用に予約している。また、Javascript でこの要素を操作するために、id 属性を設定している。

```
<!DOCTYPE html>
<html lang="ja-jp">
<head>
<meta charset="UTF-8" />
<title>canvas 要素の描画</title>
<script src="canvas-sample.js"
            type="text/javascript"
            charset="utf-8"></script>
</head>
<body>
<h1>canvas に Javascript で描画してみる</h1>
<canvas id="drawingarea" width="640"
        height="480"></canvas>
</body>
</html>
```

図 11.2　canvas を組み込んだ例

上記の例では、canvas 要素の描画は、Javascript によって操作するため、操作プログラムを外部スクリプトとして読み込ませている。外部スクリプトの指定は script 要素で行う。src 属性で読み込むリソースを、type 属性で Javascript であることを示している。

11.2 canvas 要素への描画

Javascript で、canvas 要素への描画を行うための基本的なメソッドなどを紹介する。

11.2.1 window.onload に設定

ページ読み込み完了後に、描画プログラムの実際の処理をスタートさせるため、処理内容を記述した関数を呼び出すだけの、イベントハンドラを設定しておく。

```
window.onload = function() {
  draw();
};
```

図 11.3 描画本体処理の登録

11.2.2 描画コンテキストの取得

Canvas では、描画コンテキストオブジェクトに対して、用意された描画メソッドを呼び出すことで描画を行う。描画コンテキストオブジェクトは、canvas 要素を示すオブジェクトに用意された、getContext() メソッドにより行う。getContext() は、引数に contetId をとる。描画コンテキストとして取得するオブジェクトの仕様を指定する文字列を指定する。

現時点では、この文字列として仕様上有効なのは、'2d' だけであるため、事実上、getContext("2d") として呼び出すだけである。

11.2.3 線を引く

図 11.4 では、取得した描画コンテキストを変数 ctx に格納している。そこで、この ctx に対して、描画のためのメソッドを呼び出して、実際に描画させてみる。

描画の際には、描画する canvas 上の位置を指定する必要があり、このため、canvas には座標が用意されている。canvas の座標は、左上隅が $(0, 0)$ で、右に行くに従って x 座標が増加、下に行くに従って y 座標が増加する。したがって、右下隅が最も座標値が大きい組となる。

ここでは、線を引いてみる。線を引くためには、stroke() というメソッドがあるが、このメソッドは、事前に設定されたパス（path）を描画するためのメソッドである。そのため、まず、引こうとする線の形状となるパスを設定し、その後描画する。

beginPath() で、線の開始を宣言する。moveTo(x, y) で、x, y 座標へペン先を移動する。lineTo(x, y) で、x, y 座標までペンで線を引く。closePath() で、線の開始点まで図形を閉じるように線を引いて、Path を閉じる。stroke() で、実際に描画される。

図 11.3、図 11.4 を一つのファイルに納めて、HTML から呼び出されるように設定すると、ゆがんだ五角形が描画される。

11.2.4　四角を描く

context に対する描画メソッドには様々なものがあり、四角や円などを描くこともできる。ここでは、四角を描くメソッドを紹介する。

表 11.1　四角描画 methods

method	動作
`fillRect(x, y, w, h)`	塗りつぶした矩形
`strokeRect(x, y, w, h)`	枠線のみの矩形
`clearRect(x, y, w, h)`	矩形領域を削除して透明に

表 11.1 に矩形描画 method を一覧する。`x, y` は、矩形左上端の座標で、`w, h` は、幅と高さである。

11.2.5　色を設定する

これまでのところ、描画色は黒だけであったが、もちろん、自由な色を指定できる[*2]。枠線描画用のペンと、塗りつぶし用の 2 種類のペンがあると考えると良い。これらは、描画コンテキストのプロパティとして定義されており、それぞれ、`strokeStyle` と `fillStyle` である。表 11.2 に両者を列挙しておく。

色の指定は、CSS の色指定と同様の文字列を設定する。例えば、`#80ff80` や、`rgb(128,255,128)` のような文字列である。プログラム内の事情に応じて、適切な方を用いれば良い。

[*2] 単純な色以外の指定もできる。canvas 関連のオブジェクトとして指定するが説明は割愛する。

11.3 HTML5 の情報源

Canvas に関わる仕様は、先述の W3C working draft[12] の公開内容が最新公開版である。しかし、ダイジェストではあるが、例の提示もあり、策定中の HTML5 に関する日本語情報源として、HTML5.JP[13] が利用できる。

11.4 演習問題

11.4.1 Canvas の準備

`ch11/ch11.html` として、図 11.2 のような canvas を含む HTML5 文書を用意しなさい。

11.4.2 三角形の描画

前期で用意した `ch11/ch11.html` に対して、制御用 Javascript のファイルは、`ch11/ch11.html` と同ディレクトリに、`ch11/ch11.js` のように配置すること。

適当な 3 点を結ぶ直線をそれぞれ異なる色で 3 本描く。

11.4.3 `Math.random()` 関数の活用

三角形の 3 辺の色を、毎回異なる色となるようにランダムに割り当てる。色指定には、'`rgb(rrr, ggg, bbb)`' の形式が利用しやすい。(`rrr, ggg, bbb` はそれぞれ、0〜255 の 10 進数値が使用できる。) 小数を整数化するための関数例としては `Math.floor` がある。

11.4.4 座標のランダム化

三角形の 3 頂点を毎回異なる座標となるようにランダムに割り当てる。

11.4.5 イベントとの関連づけ

画面内にボタンを追加して、click すると三角形の各辺を再描画するようにする。(click するたびに、色が変わるはずである。)

11.4.6 多数の描画

`for` ループを用いて、ランダムな三角形を 30 個描画するように書き換えてみる。

11.4.7　応用

四角形や円などを描画してみる。単純にランダムに配置するのではなく、法則性を持たせると幾何学模様を生み出すこともできる。

11.4.8　タイマイベントとの組み合わせ

描画を時間をおいて逐次的に行うと、アニメーションしているように見える。工夫してみよ。

```
function draw()
{
  // canvas 要素の取得
  var canvas =
    document.getElementById('drawingarea');
  if ( ! canvas || ! canvas.getContext )
    { return false; }
  // 描画用 context を取得
  var ctx = canvas.getContext('2d');
  // context を用いて線を引く
  ctx.beginPath();
  ctx.moveTo(50, 50);
  ctx.lineTo(100, 20);
  ctx.lineTo(200, 200);
  ctx.lineTo(90, 120);
  ctx.closePath();
  ctx.stroke();
}
```

図 11.4　線を引く

表 11.2　色設定 methods

property	設定されるペン
`strokeStyle`	枠線用
`fillStyle`	塗りつぶし用

第12章

Canvasの応用

HTML5 で導入が予定されている canvas 要素上に任意の線分を描画することができるようになった。これを用いれば、シミュレーション（simulation）結果を、Web ブラウザ上で表示させることなども可能である。

ここでは、確率過程の基礎である random walk について、その過程のシミュレーションと、結果の描画を併せて Javascript で行ってみる。また、結果の描画を、時間的なタイミングをうまく取りながら行えば、アニメーション表示も可能である。

12.1 Random walk

確率的にしか定まらない事象（確率事象）が次々に発生していく経過を表現したものとして、確率過程と呼ばれる考え方がある。ここでは、確率過程の最も単純な例として、Random walk を取り上げ、これの描画を試みる。

12.1.1 1 次元 Random walk

コイン投げを考える。表裏の出る確率は共に 1/2 とする。変数 y の初期値を 0 とするとき、t 回目に表が出たら +1、裏が出たら-1 を加えることにする。このとき、y は、t の関数となり、$y(t)$ はランダムウォークと呼ばれる。このような 1 次元の t に対する値として定義されるランダムウォークは、1 次元ランダムウォークである。

表 12.1　1 次元 random walk の一例

t	0	1	2	3	4	5	6	7	8	9
加算値	-	-1	-1	+1	+1	+1	-1	+1	+1	-1
$y(t)$	0	-1	-2	-1	0	+1	0	+1	+2	+1

12.1.2 シミュレーション

1 次元ランダムウォークをシミュレートし、その様子を画面に描画することを考える。t の関数として表現される $y(t)$ を描画するため、横軸に t を、縦軸に $y(t)$ をそれぞれとった、2 次元平面に描画すれば良い。

ここで、画面範囲が有限であるため、画面の端までシミュレーション結果を描画したら、シミュレーション

を打ち切る方針を採る[*1]。すると、このシミュレーション本体の擬似コードは、図 12.1 のようになる。

```
t := 0
y := 0
[指定回数の loop]
{
  // ここでの (t, y) が 1 次元 random walk
  [(t, y) を出力]

  t := t + 1;
  y := y + [表なら +1, 裏なら-1]
}
```

図 12.1　1 次元 random walk

一定の終了条件（例えば回数）を満たすまでの loop 中で、t の値を 1 ずつ加算し、併せて、y の値を乱数を用いて +1 または、-1 を加算していく。プログラム中では、変数 y が $y(t)$ の値を保持する。1 回の loop 毎に、t と y の値を出力すれば良い。

12.1.3　座標軸

t-y の関係を適当な座標に示せば、描画として表現することができる。例えば、x 座標で t を、y 方向で y を表現すれば、横に移動するに従い値が変化するぎざぎざの線が観察される。

12.1.4　ヒント

- `Math.random()` は、0 以上 1 未満の乱数を返す
- `Math.floor(x)` は、x を超えない最大の整数値を返す
- まず t-y の関係の計算と確認
- (x, y) 座標に変換
- 計算した座標に向けて線を引く
- 開始、終了点などを調整
- 描画を関数としてまとめる
- イベントハンドラとして描画関数を登録する

12.2　演習問題

canvas 上に折れ線グラフで random walk のシミュレーション結果を描画するようになるため、HTML5 として `ch12-1/ch12-1.html` ファイルを作成し、適切な `ch12-1/ch12-1.js` などと関連づけること。

[*1] 画面端までいったら、逆端から続きを描画する方針も考えられる。

12.2.1　1 次元 random walk の表

1. `ch12-1/ch12-1.html` をメインファイルとして、1 次元 random walk の表を完成しなさい。
2. `input`、`textarea`、`table`、その他、見やすいと思う HTML 要素で記述すれば良い。

工夫 正負の確率を偏らせてみる。

工夫 正負の確率を t の値によって変化させてみる。

工夫 $y(t)$ の値に応じて、背景色などを変化させてみる。

工夫 2 次元 random walk に拡張してみる。

工夫 その他、ここにないアイディアがオリジナリティである。

12.2.2　1 次元 random walk の折れ線グラフ

1. `ch12-1/ch12-1.html` をメインファイルとして、1 次元 random walk を `canvas` 上の折れ線グラフとして描画しなさい。

発展 2 次元 random walk などを定義できた人は、それだけを描画しても良い。

工夫 $y(t)$ の値に応じて、線の色を変えてみる。

工夫 ボタンによって再描画させてみる。

工夫 パラメータをフォーム（`input`）から入力できるようにする。

発展 その他、ここにないアイディアがオリジナリティである。

12.2.3　1 次元 random walk のアニメーション

1. `ch12-2/ch12-2.html` をメインファイルとして、1 次元 random walk を `canvas` 上の折れ線をアニメーションにして描画しなさい。

ヒント 折れ線毎に描画するタイミングをタイマイベント毎にすればよい。

工夫 折れ線の履歴を少し保持しておき、先端とその次、その次、と色を変えてやると、光っているように見せたりできる。

工夫 一部の処理を 1 度のタイマイベントでまとめて処理させると、見かけの実行速度が上がったように見える。

工夫 リアルタイムに条件を判断し、色などを変化させると派手な演出ができる。

発展 その他、ここにないアイディアがオリジナリティである。

付録 A

OS 操作の基礎

A.1　UNIX 系システム

UNIX 系システム（以下、単に UNIX）は、オペレーティングシステム（OS）の一種類である。OS は、コンピュータを利用するにあたって、多くのアプリケーションプログラム（応用プログラム）で共通に利用されるような機能を提供し、コンピュータ利用の便を図る目的で使用され、また、開発されている。日本語では基本ソフトウェアと訳されることもある。

UNIX は、複数の利用者で同時に、また、ネットワークを通じて遠隔地から利用できることを前提として開発されている。遠隔地への通信経路は通信帯域が狭かったり、遅延が大きかったりすることもある。また、遠隔地のシステムが十分に広いディスプレイを持っていないことも考えられる。これらの遠隔利用のために考えられる障害のため、（また、もちろん、UNIX 系のシステムの歴史が長いこともあり、）キャラクタ端末（文字だけを出力できる端末）で利用出来るようになっている。キャラクタ端末では、利用者との対話（interaction）全般を文字だけでやりとりするため、キーボードと文字ベースのディスプレイを用いる。

このような利用者インタフェースを Character-based user interface と呼び、CUI と略される。その逆で、マウスなどを用いて画面上のポインタを操作することでコンピュータを操作する利用者インタフェースを Graphical user interface と呼び、GUI と略される。UNIX では、CUI を基本として持ち、GUI による操作系も用意されているため、用途に応じて使い分ければよい。

A.2　ファイルとディレクトリ

A.2.1　ディレクトリツリー

OS が提供する最も基本的な機能にファイル管理がある。ファイル管理機能を提供する部分をファイルシステムと呼ぶ。アプリケーションプログラムがひとかたまりとして利用したいデータのかたまりを、扱いやすいように名前で識別し、いつでも利用可能な状態で格納しておく手段などを提供する。

近年の OS では、多数のファイルを整理しやすくする手段として、ディレクトリツリー（directory tree）と呼ばれる木構造が用意されている。システム全体を束ねる root node を起点とし、いくつかのシステムによる命名を除けば、利用者が任意の名前を識別子としてつけられる、多階層のディレクトリによって構成される。ファイルはこのディレクトリツリー内の特定のディレクトリ（ノード）内に存在することになる。Microsoft 社の最近の Windows OS では、ディレクトリのことをファイルを束ねる道具に見立ててフォルダ（folder）と呼び、最近では、この呼び名は UNIX にも移入されつつある。

ディレクトリとは、本来住所録のことを指し、地域ごとに整然と並べられ、分類整理されている電話帳のようなものである。これをファイルシステムにおける多数のファイルの整理メタファ（metaphor; 隠喩、暗喩）として用いたのが、OS におけるディレクトリである。GUI では、ディレクトリは多数のファイルを格納する箱などとして表現されることが多く、ディレクトリの中を覗くと（開くと）、その中にさらに下位層のディレクトリやファイルが格納されている。

ユーザ毎に固有の情報などを保存しておくべき個別のディレクトリとして設定されたディレクトリのことを、ホームディレクトリという。一般的な UNIX 系システムでは、login 後にはこのホームディレクトリを操作対象ディレクトリとしてシェルが起動しているように設定される。

ホームディレクトリは、個人毎に異なるように設定することが一般的であるため、一人一人異なる。そのため、各人のホームディレクトリを表す表現として、変数的な`$HOME` が用いられる[*1]。`$HOME` は、操作している人毎のホームディレクトリへの絶対パスに置き換えて読めばよい。

A.2.2 絶対パスと相対パス

ある特定のファイルは、（いくつかの例外的な取り扱いを除けば）ディレクトリツリー上の一つのノード（ディレクトリ）内に存在する。一つのディレクトリ内において、同一名のファイルの存在は通常許されない。したがって、root node から、ファイルの格納されているディレクトリを含むディレクトの名称を順に書き連ねることで、ファイルを特定することができる。すなわち、ファイルの識別子として機能する。これは、広域の単位から徐々に狭い地域を指す地名を書き連ねていき、最終的に地理上の一地点を指し示す住所の記法と同様である。

例えば、図 A.1 において、「root node に含まれる `directory 2` に含まれる
subdirectory 1 に含まれる
filename3」という名前のファイル、とすれば、一意にファイルを特定できる。図上では、順にその名前のディレクトリを辿っていけばよい。この図では、「directory」と含まれているノードがディレクトリであることとする。

OS 上では、このファイルの指定を簡潔に書くため、UNIX などでは、「に含まれる」の代わりに、区切りとなる文字として‘/’（スラッシュ）を用いる[*2]。また、root node は、明らかに全てのファイル（そして、もちろんディレクトリ）を含んでいるため、全てのファイル（そしてディレクトリ）を指し示す際に「root node に含まれる」からではじめることになる。このような自明な重複は冗長であるとされ、最低限の記述で済ますため、書かないことになっている。すると、先ほどのファイルを指し示す識別子は、`/directory 2/subdirectory 1/filename 3` と書ける。

このような識別子のことを絶対パス（absolute path）と呼ぶ。path とは、小径を指し、ディレクトリツリーを辿って特定する様子から名付けられた。絶対パスは、図 A.1 の全てのファイルにおいて必ず異なる文字列になることを確認しよう。例えば、「`filename 3`」という名前の付けられたファイルは多数存在するが、絶対パスによる命名ではそれぞれが完全に識別される。絶対パスはいわば、ファイルの本名のようなものである。

この考え方によると、root directory は、2 つのディレクトリを直接含む。また、`/directory 1/`は、さらに、2 つのディレクトリを含む。このように、一般にディレクトリは、含む-含まれるの関係にあり、含まれる

[*1] `sh` と呼ばれる基本的なシェルにおいて、実際に変数として設定されることが多いことから利用されている。

[*2] 歴史的経緯により、Windows OS では、‘\’（バックスラッシュ）が用いられる。そして日本語版では、これまた歴史的経緯により、バックスラッシュ文字は円記号で表現されている。

側のディレクトリを含む側のディレクトリから見て「子」、逆を「親」と言う。図 A.1 で言えば、右側へ辿ると子、左側がその親、である。子ディレクトリは、child であるが、「含まれる」という含意で sub directory と呼ばれることも多い。親ディレクトリは、parent directory である。

ちなみに、`/direcotory 1/subdirectory 2` のように、ディレクトリにひとつのファイルもないディレクトリのことを「空ディレクトリ」と呼ぶ。

絶対パスを用いれば、必ず一つ一つのファイルを識別できる。しかし、それでは都合の悪いこともある。単に識別子が長いこともあるが、それは小さな問題に過ぎない。図 A.1 において、`/directory 1/subdirectory 1/`ディレクトリに含まれるファイルの数はいくつだろうか。

答えはもちろん、2 である。

では、同様に、`/directory 2/subdirectory 1/`ディレクトリに含まれるファイルの数はいくつだろうか。答えは 3 である。

このような、あるディレクトリに含まれるファイルの数を数える、という簡単な仕事を考えると、これはコンピュータにやらせるべき筆頭の仕事であると言える。数えるべきディレクトリを絶対パスで示してやれば良い。

では、`subdirectory 1` という名前のついたディレクトリに含まれるファイルの数を数える、という仕事を考えた場合、どのようになるだろうか。root directory にはその名前のディレクトリはない。一方、`/directory 1/`と`/direcotry 2/`には、含まれる。したがって、`subdirectory 1` というディレクトリに含まれるファイル数を数えることができる。

このように、どこのディレクトリを起点として考えるか、によって、実際に指し示すディレクトリが異なるような処理をしたい場合、逆に言えば、一般的に同じ処理を特定の部分ディレクトリツリーに適用したい場合がある。このような場合に、絶対パスを用いるのは一般化ができないため、プログラムの再利用性に乏しい。

部分木を特定するためには、その部分木の最上位ノードを特定すれば必要十分である。すなわち、最上位ノードの絶対パスを記述すれば良い。例えば、`/directory 1/`とすれば、上側の部分木全体を指すように考

える。そうすれば、「その部分木に含まれる `subdirectory 1` に含まれる」ファイルの数を数える、と仕事を書くことができる。

「その部分木に含まれる」というところを最も簡潔に書く方法として、何も書かない、という方法が編み出された。すなわち、「`subdirecotory 1/`」である。このような書き方を相対パス（relative path）と呼ぶ。しかし、これではどこの `subdirectory 1` かがわからない。そこで、「その」を指し示すために、「カレントディレクトリ（current directory)」という概念が使われる。この場合のカレントディレクトリは、「`/directory 1/`」である。

カレントディレクトリと相対パスを連結すると絶対パスになることに注意しよう。したがって、カレントディレクトリと相対パスの組み合わせで、ファイルを一意に識別できることは自明である。

これは、一見すると絶対パスと同様であるように見えるかも知れない。しかし、カレントディレクトリを別のディレクトリにすると、例えば、「`/directory 2/`」とすると、全く別のディレクトリを指す。カレントディレクトリは、そこから仕事を始める起点のように振る舞うため、「working directory」とも呼ばれる。

A.3 UNIX コマンド

UNIX の CUI では、シェル（shell）と呼ばれるプログラム（ソフトウェア）を用いて、character-based なインタラクションを利用者と計算機とが行う。シェルは、利用者または、管理者が変更することが可能であるが、一般的には bash や tcsh、zsh などと呼ばれるプログラムが用いられることが多い。

これらのシェルは、コマンドシェルと呼ばれることもあるように、コマンドを入力することで計算機に指示を出す。コマンドの処理結果などは、画面上に文字列で表示される。コマンドは、シェル自体が解釈する内部コマンドと、別のプログラムを呼び出す外部コマンドとに分けられる。ごく基本的な操作を内部コマンドとして提供し、ほとんどの操作は外部コマンドを呼び出す実装になっている。このため、内部コマンドについてはシェルの種類によってコマンドが異なることになるが、ごく基本的な使い方に関しては、おおむね統一がとれていると考えて良い。

コマンドを多数連ねることで、一連の定型的な操作を行うことも多いため、これらをファイルとして記述しておくことで、一連のコマンドを自動的に連続して実行させる仕組みが用意されている。これは、一連の処理をさせるシェル上の記述であることから、シェル（言語）プログラムと呼ばれる。

コマンドは、カレントディレクトリの概念を持ち、仕事に便利なようにカレントディレクトリをプロンプトの一部として表示するよう設定することが多い。コマンドに渡す（コマンド名に続けて入力する）引数としてディレクトやファイルを書く際には、相対パスで示した場合、このカレントディレクトリと連結された絶対パスとして評価される。

A.3.1 基本的な UNIX コマンド

主として、ファイルとディレクトリの操作に関するコマンドが基本的なコマンドである。本授業で用いる基本的な UNIX コマンド一覧を表 A.1 に示す。これらは、きわめて多用されるため、その基本的な使い方を完全に習得しておくことが必要である。

詳細な使い方は、`man` コマンドを用いれば良い。`man man` と入力すれば、`man` コマンドの使用法自体が表示される。

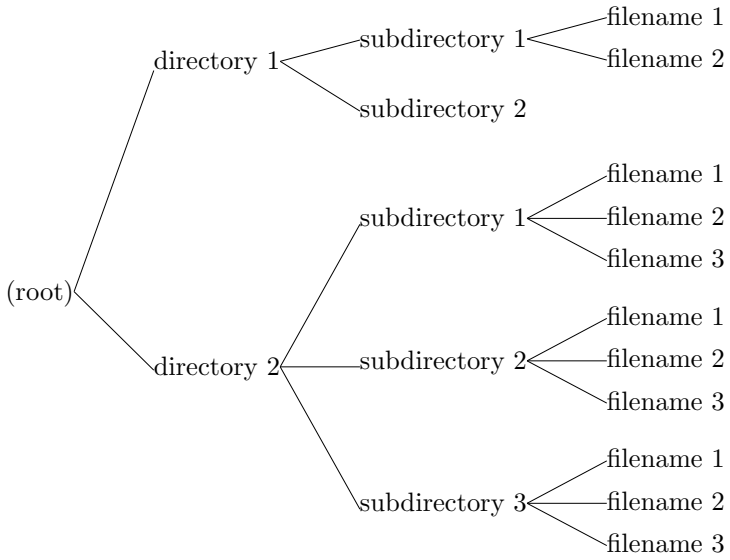


図 A.1　ディレクトリツリーの例

表 A.1　基本的な UNIX コマンド

コマンド	説明
`man`	コマンドの使い方の説明書を表示する。
`pwd`	現在の working directory を表示する。
`cd`	current (working) directory を変更する。
`ls`	ファイルのリストを表示する。
`cat`	（連結した）ファイル内容を表示する。
`cp`	ファイル、ディレクトリを複製する。
`mv`	ファイル、ディレクトリを移動する。
`touch`	ファイルの更新日時を現在にする。
`rm`	ファイル、ディレクトリを削除する。
`mkdir`	ディレクトリを作成する。
`chmod`	アクセス権を変更する。

参考文献

[1] “HTML5 A vocabulary and associated APIs for HTML and XHTML,” W3C Recommendation 28 October 2014, http://www.w3.org/TR/html5/, 2014.

[2] “HTML 5.1 W3C Recommendation, 1 November 2016,” W3C Recommendation, http://www.w3.org/TR/html51/, 2016.

[3] “XHTML 1.0 The Extensible HyperText Markup Language (Second Edition),” W3C Recommendation, (2002).

[4] PortableApps.com, http://portableapps.com.

[5] “The W3C Markup Validation Service,” Web resouce http://validator.w3.org/, Accessed 29th Sep 2009.

[6] 狩野 祐東， “いちばんよくわかる HTML5&CSS3 デザインきちんと入門”, SB クリエイティブ, 2016.

[7] 伊藤 庄平, 益子 貴寛, 久保 知己, 宮田 優希, 伊藤 由暁, “いちばんよくわかる Web デザインの基本きちんと入門　レイアウト／配色／写真／タイポグラフィ／最新テクニック”, SB クリエイティブ, 2017.

[8] エビスコム, “HTML5&CSS3 レッスンブック”, ソシム, 2013.

[9] エビスコム, “HTML5&CSS3 デザインブック”, ソシム, 2014.

[10] 益子貴寛, Web 標準の教科書, 秀和システム, 2005.

[11] 市瀬裕哉, 福島英児, 望月真琴, 実践 Web Standards Design, 技術評論社, 2009.

[12] “HTML Canvas 2D Context, W3C Working Draft 24 June 2010”, http://www.w3.org/TR/2dcontext/, 2010.

[13] HTML5.JP，http://www.html5.jp，Accessed 28th, June, 2010.

[14] エ・ビスコム・テック・ラボ，“CSS3 スタンダード・デザイン・ガイド”，マイナビ，2011.

[15] 山田祥寛，“JavaScript 本格入門”，技術評論社，2010.

[16] Sbelley Powers 著，武舎広幸，武舎るみ訳，初めての JavaScript 第 2 版，オライリージャパン，2009.

[17] 藤岡功, JavaScript ビジュアル・リファレンス, 初版, エムディエヌコーポレーション, 2004.

[18] 山崎大助, “JQuery レッスンブック”, ソシム, 2014.

[19] 増井 雄一郎, 深津 貴之, 川崎 有亮, 台場 圭一, Ajax 実装のための基礎テクニック, 技術評論社, 2006.

[20] 情報科学研究教育センター利用ガイド, 工学院大学情報科学研究教育センター, ～2017