

Web 演習 (ver. 3.0d)

東京電機大学 工学部

情報通信工学科

坂本 直志

2020 年 9 月 15 日

目 次

1	予習	4
1.1	準備	4
1.1.1	大学のサーバを利用する場合	4
1.1.2	自分のパソコンにサーバを立てる	4
1.2	1 週目	4
1.3	2 週目	4
1.4	その他	4
2	目的	5
3	原理	5
3.1	URL	5
3.2	HTTP	6
3.3	HTML	7
3.3.1	HTML 文書の基本構造	8
3.3.2	head 要素が内容として含める要素	10
3.3.3	body 要素の内容として含むことのできる要素	10
3.3.4	インライン要素	11
3.3.5	リスト (ol, ul, dl) の内容	13
3.3.6	表	13
3.3.7	マークアップの実際	14
3.4	Form	16
3.4.1	Form によるサーバへの情報伝達	16
3.4.2	HTML の form 要素	16
3.4.3	フォームの例	18
3.5	CSS	20

Web 演習 2

3.5.1	背景	20
3.5.2	CSS の書き方	20
3.5.3	セレクタの書式	21
3.5.4	表示のされ方	22
3.5.5	プロパティ	23
3.5.6	値の指定	25
3.5.7	CSS の例	26
3.6	DOM	28
3.6.1	基本概念	28
3.6.2	JavaScript の API	28
3.6.3	例	28
3.7	Node.js	29
3.7.1	Node.js の概要	29
3.7.2	express	29
3.7.3	セッション	29
3.8	EJS	30
3.8.1	EJS の基礎	30
3.8.2	include	31
4	実験	31
4.1	実験装置の説明と原理	31
4.2	実験の準備	32
4.3	実験上の注意	33
4.4	ネットワークの接続	33
4.4.1	実験室内でのネットワーク接続	34
4.4.2	VPN を利用した学外からの大学のサーバ利用	34
4.4.3	自分のパソコンにサーバをインストール	35
4.5	実験 1	35
4.6	実験 2	37
4.7	実験 3	38
4.8	実験 4	39
4.9	実験 5	40
4.10	実験 6	41
4.11	実験 7	42
4.12	レポートをまとめる上での注意点	43
5	検討事項	43
5.1	必須事項	43
5.2	発展事項	44

A	Form の例と集計プログラム	45
A.1	text を使った例	45
A.2	radio を使った例	46
A.3	select を使った例	47
A.4	button を使った例	48
A.5	shukei.ejs	49
B	DOM の利用例	51
B.1	見出し要素の内容の出力	51
B.2	商品リストの入力	52
C	商品登録	52
C.1	toroku.html	52
C.2	toroku.ejs	53
C.3	torokubody.ejs	54
D	ショッピングモールの設計	55
D.1	画面の設計	55
D.1.1	login	56
D.1.2	mall	56
D.1.3	confirm	56
D.1.4	logout	56
D.2	JSP に対する設計 1	57
D.2.1	login	57
D.2.2	mall	57
D.2.3	confirm	57
D.2.4	logout	58
D.3	技術的検討	58
D.3.1	データベースの実現	58
D.4	プログラム	59
D.4.1	login.ejs	59
D.4.2	mall.ejs	60
D.4.3	mallbody.jsp	61
D.4.4	confirm.ejs	63
D.4.5	confirmbody.ejs	64
D.4.6	logout.ejs	66
E	サーバ	67
E.0.1	server.js	67

1 予習

1.1 準備

HTML や CSS を編集するエディタ (TeraPad[1]、emacs など) と、telnet クライアント (Tera Term など) のソフトウェアが必要です。Windows でも Macintosh でも、何もダウンロードしてこなくても使えるようにはできますが、好みがあるものですので、自由に準備して下さい。

また、複数のブラウザでの見栄えの比較をしますので、Firefox、Opera、Safari など Edge 以外のブラウザを一つ以上利用できるようにしておいて下さい。

1.1.1 大学のサーバを利用する場合

VPN を利用しますので、総合メディアセンターのページ (<https://www.mrcl.dendai.ac.jp/>) から VPN の接続ができるように確認しておく。

1.1.2 自分のパソコンにサーバを立てる

坂本の授業ページ (<http://edu.net.c.dendai.ac.jp/>) よりサーバーキットをダウンロードし、パソコンにサーバをインストールする。

1.2 1 週目

実験 2 の 1,2, 実験 3、実験 4 のプログラムによる確認はサーバが無くてもできますので、時間内に実験 4 まで終了するように、できる範囲で事前にやっておいてください。

1.3 2 週目

実験 6 ではショッピングサイトを作りますので、あらかじめ数件のショッピングサイトを見て、画面のレイアウトや表現の特徴を捉えておいて下さい。ショッピングサイトのデザインの設計はサーバが無くても作れますので、事前に考えておいて下さい。実験中は外部のサイトにアクセスできません。

1.4 その他

ソースネクスト社のウィルスセキュリティ Zero はファイアーウォールを無効にしないと実験ができないかも知れません。ファイル共有フォルダにアクセス可能な設定方法がわかる人は教えてください。

トレンドマイクロ社のウィルスバスターでは Teraterm でサーバにアクセスできないことがあります。

また、ノートンやカスペルスキーでもトラブルが生じることがあります。

2 目的

本実験では Web 技術に関するさまざまな演習を行い、理解を深めます。プロトコルの理解、HTML によるマークアップ、スタイルシート、DOM、JavaScript、テンプレートによるドキュメントに関連する演習を行います。

概要を大雑把に体験して理解することが目標なので、完璧な理解は必要ありませんが、わからないところはこの実験書を精読したり、他のドキュメントを調べる必要があります。

またネットワークの実験を経験し、手順やトラブル対処等の理解度を深めます。

3 原理

始めに WWW を構成する基本的な 3 つのプロトコル URL, HTTP, HTML を説明します。さらに Web 技術として CSS, DOM について説明します。

URI とは、ネットワーク上の資源を一意に表す方法です。HTTP は Web の通信方法です。HTML は Web のドキュメントの清書の書式です。CSS は Web ドキュメントの見栄えの与え方です。DOM は XML ドキュメントをプログラムから利用する方法です。

最後に EJS という、Web ドキュメントにサーバサイドの JavaScript のプログラムを埋め込む方法を説明します。

3.1 URL

ネットワーク上の資源 (リソース) を文字列により一意に指し示す方法として URI (Universal Resource Identifier) があります。これは、アクセスの手法とリソースの所在を `:(コロン)` で区切って表示します。WWW のアクセス手法は `http` (Hyper Text Transfer Protocol) です。一方、WWW でのリソースの所在は、WWW サーバ名の前に `//` を書き、サーバ名の後にサーバ上のリソースの位置を絶対パス名 (`/` で始まるパス名) による表現でつなげます。したがって、東京電機大学工学部情報通信工学科のホームページ (ドキュメントの最初のページ) の URI は、アクセス手法が `http`、サーバ名が `www.c.dendai.ac.jp`、ホームページのサーバ上のリソース位置は `/(ルート)` なので、`http://www.c.dendai.ac.jp/` と表します。

またこれらのあとに `#` を付け、そのあとに HTML ドキュメントで `id` オプションや `a` 要素の `name` 属性で指定した名前を指定すると、文書の内部の特定の位置を示すことができます。

HTML ドキュメント上では相対 URI を使うことができます。これは、そのドキュメントからの相対的な位置だけを示します。`http://a/b/c.html` というドキュメントと同じディレクトリにある `d.html` というファイル (つまり `http://a/b/d.html`) は、`http://a/b/c.html` からは `http://a/b/d.html` と書く代わりに単純に `d.html` で表せます。また、親ディレクトリにある `e.html` (つまり `http://a/e.html`) は `../e.html` で表すことができます。

```
field1: xxxx
field2: yyyy
:
fieldn: zzzz

Here begins contents after one blank line....
The contents continue until the end.
```

図 1: RFC822[2] で定められた電子メールのメッセージ形式

3.2 HTTP

WWW サーバからリソースを取り出すには HTTP(*Hyper Text Transfer Protocol*) というプロトコル(通信手順)を使用します。HTTP は電子メールのプロトコルを元に作られました。

電子メールのプロトコル(SMTP RFC822[2], RFC2822[3], RFC5822[4])では、通信に使う情報であるヘッダと通信したいメッセージ部分のボディの二つの部分に分けられます。ヘッダ部分は行が改行記号で区切られ、各行はレコード名と内容が: (コロン)で区切られています。そして、ヘッダ部分とボディ部分は空行で区切られます(図 1)。

HTTP バージョン 1.0 では、次のような単純なプロトコルを使用します。利用者はサーバに対して TCP ポート 80 番にアクセスします。そして、一行目に HTTP のメッセージを送り、その後に電子メールと同様のメッセージを付けます。メッセージの書式は「コマンド サーバ上のリソースの位置 HTTP のバージョン名」となります。コマンドには GET、HEAD、POST があります。その際、単純にリソースを取り出すだけなら、空のメッセージを送るという意味でヘッダもボディも空を表す空行二つを送ります。例えば、`http://www.c.dendai.ac.jp/` の情報を取り出すには、`www.c.dendai.ac.jp` という名前のサーバの TCP 80 番のポートにアクセスし、次のメッセージを送ります。「GET / HTTP/1.0 『改行』『改行』」すると図 2 のような返答が返ってきます。この一行目が状態を表す行で、「プロトコルバージョン名 状態番号 状態」がひとつの空白で区切られてきます。

そして、二行目から空行までがヘッダ部分で、日付、サーバ名、更新日時などドキュメントのメタ情報が書かれています。そして、最初の空行以降がドキュメントの内容で、この場合は HTML 文書になっています。

状態番号には、正常を表す 200 番台、利用者側のエラーを示す 400 番台、サーバー側のエラーを示す 500 番台などがあります。

HTTP1.1 は以下の機能を実現したもので、1997 年に規格化され、しばらくは標準プロトコルとして、運用されました。

1. 一台のサーバーで複数の Web サーバを構築する
2. 接続を継続する

```
HTTP/1.1 200 OK
Date: Sat, 12 Sep 2020 11:12:38 GMT
Server: Apache/2.4.29 (Ubuntu)
Last-Modified: Fri, 19 Jun 2020 17:53:07 GMT
ETag: "9c4-5a87390cd51a3"
Accept-Ranges: bytes
Content-Length: 2500
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

<!DOCTYPE html>
<html lang="ja">
...
```

図 2: HTTP によるサーバの返答

3. 予め総送信バイト数を知らなくても送信できるチャンク転送
4. サーバーメッセージに 100 番台の情報提供が追加された

HTTP/2 は Google が開発しました。2015 年に RFC7540 として規格化されました。

1. HTTP 1.1 と互換性あり
2. 100 番台のメッセージを使い、ネゴシエーションを行う
3. ストリーミングの概念を導入し、複数のドキュメントを並列に送信する

あまり普及していませんが、高負荷の通信に耐えるため、採用が進んでいます。

なお、現在 Google は HTTP3.0 も開発しています。

3.3 HTML

Hyper Text Markup Language(HTML) ははもともとは SGML という汎用の「マークアップ言語」作成言語を使って作られた Hyper Text 用のマークアップ言語です。マークアップとは文書をコンピュータで処理するため、文書に印(タグ)をつけることです。

Tim Barnard Lee により開発された Web システムは多くの人の手により改良されました。そして、俗に言うブラウザ戦争により様々な規格が乱立し、規格を統一管理する必要が出てきました。始めに IETF が HTML 2.0 と HTML 2.x という規格をまとめました。

この後、Web 技術だけを取り扱う World Wide Web Consortium(W3C) ができました。1997 年 1 月に HTML 3.2 がまとめられた後、1997 年 12 月に HTML 4.0 Transitional,

Web 演習 8

Strict, Frameset がまとめられました。その後、W3C は SGML の非効率さを取り除き、拡張可能なメタマークアップ言語 XML(eXtensible Markup Language) を決めました。そして、HTML も XML 化を進め(未完)

SGML は 1986 年に提案されました。ところが、SGML はコンピュータで活用するには複雑である一方で、拡張性がないため、文法上は SGML と互換性を持たせた上で、拡張機能をもたせたのが XML です(1998 年)。

その後、HTML は XML 化された規格が出る一方で、産業的に様々な

2016 年現在、有効な HTML のバージョンには 4.01 Strict, 4.01 Transitional, 4.01 Frameset, HTML 5 があります。さらに、後継の XHTML 1.0 Strict, 1.0 Transitional, 1.0 Frameset, XHTML 1.1, XHTML Basic があります。これらは全て HTML 4.01 を基礎として作られています。4.01 Transitional と 4.01 Frameset は、従来のブラウザ戦争の名残りとして、行儀の悪いさまざまな要素を取り込んだ移行用の仕様です。そこで、この実験では将来性のある 4.01 Strict を使用します。

なお、本資料では実用上最小限の事項のみしか説明してません。HTML 4.01 Strict を使いこなすためには詳しい資料を別途御覧下さい。

なお、2014 年に仕様が勧告された HTML5 には、現行の HTML 4.01 に比べて次のような特徴を持ちます。

1. 従来は SGML のアプリケーションという立場を取っていたが、HTML 5 は HTML 4.01 と互換性を持ちながら、独自の文法を持つ
2. DOM の機能を重視している
3. MathML など也可以使用できる

HTML5 の資料は実は膨大です。HTML5 は HTML 4.01 と互換性を考えて策定されていて、さらに HTML4.01 は廃止されないため、HTML 4.01 を覚えても無駄にはならず、基礎になり得ます。

また、さらに HTML5 で積み残した仕様を組み込むため HTML5.1 が現在策定されています。

3.3.1 HTML 文書の基本構造

SGML 関連のマークアップはドキュメントにタグをつけることで情報を構造化します。このとき、要素(Element) は開始タグと内容(Content) と終了タグの三要素からなっています。開始タグは < 要素名 > で表します。内容は要素ごとに定められます。終了タグは </ 要素名 > で表します。また、属性値を指定する時は、開始タグ内に <要素名 属性名="値"> という形で指定します。以上をまとめると、図 3 のようになります。

HTML ドキュメントも SGML ドキュメントです。SGML ドキュメントはタグにどのような定義があるかの定義部と、実際にドキュメントにタグを付けて要素として列挙した部分に分かれます。但し、タグの定義は HTML 4.01 など仕様が決定している場合は一意

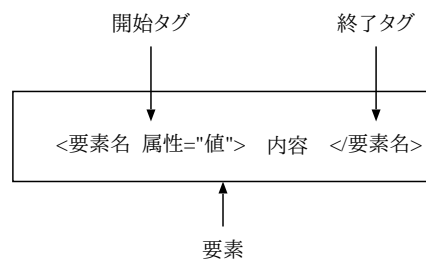


図 3: 要素とタグ

```

<!DOCTYPE html>
<html>
...
</html>

```

図 4: HTML 文書の基本

に定められています。それを示すため、文頭に SGML の DTD 宣言を書き、その後に要素を並べます。

HTML 文書は DTD 宣言とひとつの html 要素から構成されます (図 4)。

なお、`<!--` と `-->` の間にコメントを書くことができます。

各要素の含むことのできる内容は各要素ごとに決められています。html 要素は内容としてただひとつの head 要素とただひとつの body 要素を含みます。(図 5)。

なお、“ ” は通常の間隔を表します。漢字と同じ幅の間隔ではありませんので注意して下さい。

次節以降で基本的な要素を紹介します。なお、html ドキュメントが日本語で書かれて

```

<!DOCTYPE html>
<html lang="ja">
<head>
...
</head>
<body>
...
</body>
</html>

```

図 5: HTML 文書の基本 2

要素名	終了タグ	内容
meta	省略可	指定しない
title	必要	必須
link	省略可	指定しない
style	必要	CSS

図 6: head 要素が内容として含むことのできる要素 (ブロック要素)

いる場合は lang オプションに ja を指定します。

3.3.2 head 要素が内容として含める要素

meta meta 要素は body 要素の内容に書かれているドキュメントに関する情報 (メタ情報) を設定します。例えば、本文が使用している漢字コードなどはメタ情報になります。meta 要素は内容を含みません。また終了タグは省略できます。

meta 要素は文字コードの設定に必要です。UTF-8 を指定するには次のように設定します。このように内容、終了タグは指定しません。

```
<meta charset="utf-8">
```

title title 要素はドキュメントの題名を指定します。HTML 文書では必ずひとつの title 要素が必要となります。title 要素のない HTML 文書は文法上誤りと言うことになります。

link link 要素は他の情報源とのつながりを記述します。内容は指定しません。また終了タグは省略できます。スタイルシートを指定するには次のようにします。

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

style スタイルシートを HTML 文書の head 要素に含めるために使用します。type オプションで text/css を必ず指定します。本資料では、CSS の説明の都合上一つのファイルで説明を完結するために使用しますが、style 要素の代わり link 要素を使って CSS は別ファイルに書くことを勧めます。

3.3.3 body 要素の内容として含むことのできる要素

body 要素の内容には、複数のブロック要素を含むことができます。ブロック要素として代表的なものを説明します。

p p 要素は段落を意味します。内容にはインライン (テキスト) 要素を複数含むことができます。終了タグは省略可能です。

h1,h2,h3,h4,h5,h6 h_n ($n = 1 \dots 6$) 要素は見出しを意味します。数字はレベルを表し、h1 が最上位、h6 が最下位を表します。h1 のすぐ下のレベルには h2、h2 のすぐ下のレベルには h3 となるよう、レベルを連続して使うよう推奨されています。内容はインライン要素を複数含むことができます。終了タグは必須です。

blockquote **blockquote** 要素は引用を表します。内容としては複数のブロック要素を含むことができます。

ul, ol **ul, ol** 要素は箇条書を表します。ul は単なる列挙、ol は番号付の列挙になります。内容は li 要素のみを複数含むことができます。

dl **dl** 要素は要素名と定義の記述のリスト (定義リスト) を表します。内容には項目名を表す dt 要素と定義の記述を表す dd 要素のみを複数含むことができます。

div **div** 要素は意味がありません。これは文書の構造には意味づけをせず、CSS でスタイルを与えたい時などに使用します。内容にはインライン要素あるいはブロック要素を複数含むことができます。

hr **hr** 要素は水平線を表します。内容を含むことはできません。終了タグを省略できます。

pre **pre** 要素は、与えられた内容の改行や空白を無視せずに、文字を等幅に表示します。これは空白などで整形されたテキストを表示するのに使います。内容にはインライン要素のうち img, big, small, sub, sup など等幅フォントの指定に沿わない要素以外を使用できます。

address **address** 要素はドキュメントに対する連絡先を示すものです。内容はインライン要素で、終了タグは省略できません。

table **table** 要素は表を表すものです。終了タグは省略できません。内容には 0 個または 1 つの caption 要素、0 個以上の col または colgroup 要素、0 個または 1 つの thead 要素と tfoot 要素、そして、1 個以上の tbody 要素を含みます。これらの要素に関しては節を改めて説明します。

table 要素には表の見栄えに関してさまざまなオプションを設定できます。border オプションは外枠の太さを指定するオプションで、border="1" などと指定すると表の枠が表示されます。

3.3.4 インライン要素

文字 文字はインライン要素です。但し、ここで言う文字とは SGML で #PCDATA と呼ばれる文字のことです。基本的には、通常 of 文字はそのまま書いて構いません。しかし、タグを文章に入れるために、特殊な記号はそのまま書けない決まりになっています。<, >, &, " など is そのまま書かず、<, >, &, " と書く

要素名	終了タグ	内容
p	省略可	インライン
h <i>n</i>	必要	インライン
blockquote	必要	ブロック
ul, ol	必要	li 要素
dl	必要	dt, dd 要素
div	必要	ブロック、インライン
hr	禁止	指定しない
pre	必要	フォントに影響しないインライン
address	必要	インライン
table	必要	caption, col, colgroup, thead, tfoot, tbody

図 7: ブロック要素

必要があります。また、特定の文字を `&キーワード;` や `&数字;` などで表すことができます。著作権を表す「まるシーマーク」© は `©` や `©` で表します。

また、改行、タブ、空白文字の連続は、基本的にはひとつの空白とみなされます。但し、タグの直後の改行のみ無視されます。

em, strong, dfn `em, strong` 要素は強調を表します。 `dfn` は用語の定義を表します。内容はインライン要素です。終了タグは省略できません。

sup, sub `sup` 要素は上付、 `sub` 要素は下付の文字を意味しています。内容はインライン要素です。終了タグは省略できません。

span `span` 要素には意味がありません。スタイルシートなどと組み合わせて使います。内容はインライン要素です。終了タグは省略できません。

a `a` 要素はハイパーリンクを作るのに使います。内容は `a` 要素以外のインライン要素です。ですから、`a` 要素の内容に `a` 要素を入れること、すなわち入れ子はできません。また終了タグは省略できません。

オプションの `href` に URI を指定します。

img `img` 要素は画像を読み込みます。内容を含むことはできません。終了タグは禁止されています。 `src` オプションに画像の URI を指定します。また、 `width` 、 `height` オプションに画像の横、縦のピクセル数を与えます。 `alt` オプションには、画像が表示できない時のための文章を与えます (必須)。

br `br` 要素は強制改行を意味します。内容を含むことはできません。終了タグは禁止されています。

big, small, b, i, tt これらはフォントの表示に関わるものです。廃止はされてませんが、本来はスタイルシートで指定するものです。したがって、使うことは推奨されてい

要素名	終了タグ	内容
文字データ	なし	なし
em, strong, dfn	必要	インライン
sup, sub	必要	インライン
span	必要	インライン
a	必要	a 以外のインライン
img	禁止	指定しない
br	禁止	指定しない
big, small, b, i tt	必要	インライン

図 8: インライン要素

要素名	終了タグ	内容
li	省略可能	インラインまたはブロック
dt	省略可能	インライン
dd	省略可能	インラインまたはブロック

図 9: リストの内容

ません。big, small 要素は内容を大きく、または、小さくする意味があります。b, i, tt はそれぞれ、ボールド体、イタリック体、テレタイプの字体で表示することを意味しています。

3.3.5 リスト (ol, ul, dl) の内容

li li 要素はリストの要素を表します。ul では要素を表すため・などが補われ、ol では自動的に番号が付けられます。内容はブロック、または、インラインを含むことができます。終了タグは省略できます。

dt dt 要素は定義リスト dl の内容として、定義語を示します。内容はインラインを含むことができます。終了タグは省略できます。

dd dd 要素は定義リストでの定義の意味を表す要素です。内容はブロック、または、インラインを含むことができます。終了タグは省略できます。

3.3.6 表

caption caption 要素は表に題名を与えます。内容はインラインを含むことができます。終了タグは必須です。

要素名	終了タグ	内容
caption	必要	インライン
colgroup	省略可能	col 要素
col	禁止	指定しない
thead, tfoot, tbody	省略可能	tr 要素
tr	省略可能	th, td 要素
th, td	省略可能	インラインまたはブロック

図 10: テーブルの内容

colgroup, col colgroup, col は縦列のグループ化に使う要素です。colgroup は col 要素のみを含むことができます。col 要素は内容を含まず終了タグは禁止されています。col ひとつで縦列ひとつに対応します。col や colgroup ごとに class などを指定できます。

thead, tfoot, tbody thead は表の最初の部分、tfoot は表の最後の部分、tbody は表の本体部分を意味する要素です。表示される時は、tfoot 部分が最後に表示されますが、HTML 文書では tfoot は tbody より先に書く必要があることに注意して下さい。内容は tr 要素を複数含むことができます。終了タグは省略できます。

tr tr 要素は表の行を表します。内容は th または td 要素を含むことができます。終了タグは省略できます。

th, td th 要素は表の見出しを表し、td 要素は表の内容を表します。内容はインラインまたはブロックを含むことができます。終了タグは省略できます。

3.3.7 マークアップの実際

文書を実際に HTML を使ってマークアップすることを考えましょう。

まず、HTML で必ず書かなければならない DTD 宣言などはあらかじめファイルに保存しておくといいです。(図 11)

さて、HTML 文書では body 要素にはブロックしか含むことができません。つまり、単純に body 要素に文字を入れていくことができません。そこで、実際の文書をブロックに分割していくを考えます。見出し、段落、箇条書、表などです(なお、画像はブロックでないのに要注意です)。そして、その構造を示すようにマークアップしていきます。ここで注意しなければならないのは、構造を考えるにあたって、見栄えや見かけ上の改行などに気をとられてはいけないことです。強制改行を意味する br 要素が必要なことは滅多にありません。

では、例として図 12 のドキュメントを御覧下さい。このドキュメントの構造を考えます。まず、一行目は表題です。そして、次の行は空行ですが、これは見栄え上あるもの

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

</body>
</html>
```

図 11: HTML 文書の必須部分

商品申し込みフォーム

以下の手順に沿って注文して下さい。

1. お客様のお名前、性別、年齢を入力して下さい。
2. 支払方法をお選び下さい。(0:現金 1:visa 2:mc 3:amex)
3. 商品名、単価、個数を入力して下さい。
4. 最後に submit ボタンを押して下さい。

図 12: サンプルテキスト

であって、この空行が無くなくてもドキュメントの意味や構造は変わりません。従って、HTML ドキュメントを作る上では特にマークアップの対象ではありません。3 行目は文章になってます。ここでは文章は一行のみですが、これで一つの段落を形成しています。そして、4 行目以降は番号付の箇条書になっています。構造の分析結果を図 13 に示します。

さて、この分析をもとに HTML ドキュメントを作ります。まず、HTML ドキュメントに不可欠な title ですが、これは見出しをそのまま利用して「商品申し込みフォーム」とすることにします。一方、上の分析により見出し、段落、箇条書という構造を見出しましたが、これは HTML では h1 要素、p 要素、ol 要素にあたります(図 14)。また、箇条書の各要素は li 要素になります。これをもとに作った HTML ドキュメントは図 15 になります。

商品申し込みフォーム (見出し)

以下の手順に沿って注文して下さい。 (本文、段落)

1. お客様のお名前、性別、年齢を入力して下さい。 (番号付箇条書)
2. 支払方法をお選び下さい。(0:現金 1:visa 2:mc 3:amex)
3. 商品名、単価、個数を入力して下さい。
4. 最後に submit ボタンを押して下さい。

図 13: 文章構造の分析

商品申し込みフォーム (*h1* 要素)

以下の手順に沿って注文して下さい。 (*p* 要素)

1. お客様のお名前、性別、年齢を入力して下さい。 (*ol* 要素、各項目は *li* 要素)
2. 支払方法をお選び下さい。(0:現金 1:visa 2:mc 3:amex)
3. 商品名、単価、個数を入力して下さい。
4. 最後に submit ボタンを押して下さい。

図 14: HTML 要素の明示

3.4 Form

3.4.1 Form によるサーバへの情報伝達

Form によりユーザから入力された情報は、submit によりサーバへ伝達されます。情報は、「名前=値」という形でサーバ側に渡されますが、その際、伝達方法には GET と POST の二種類があります。GET では URI の後に ? マークで情報が付加され、全体で一つの長い URI になります。一方、POST では HTTP のリクエストのボディ部に情報が付加されます。

GET は送る情報まで全てを一つの URI として記述できるため、検索エンジンのキーワードの送信などに利用されています。これはつまり URI にキーワードまで記録できるため、第三者に検索キーワードまで指定した情報を URI で送ったり、ブラウザの URI の保存機能により記憶できたりするからです。一方、POST は URI に存在する文字数の制限や、文字コードの制限はありませんし、やりとりされる情報がアクセスログに記憶されることがありません。GET である必要がない限りは POST を使うべきです。

3.4.2 HTML の form 要素

ブラウザからサーバーに情報を送るには HTML の form 要素を使用します。form 要素で使用する input 要素は様々な入力方法によりブラウザからの情報の入力を可能にします。


```

<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="utf-8">
<title>商品申し込みフォーム</title>
</head>
<body>
<h1>商品申し込みフォーム</h1>
<p>
以下の手順に沿って注文して下さい。
</p>
<ol>
<li>お客様のお名前、性別、年齢を入力して下さい。</li>
<li>支払方法をお選び下さい。(0:現金 1:visa 2:mc 3:amex)</li>
<li>商品名、単価、個数を入力して下さい。</li>
<li>最後に submit ボタンを押して下さい。</li>
</ol>
</body>
</html>

```

図 15: 例文のマークアップ

form form 要素はブロック要素です。オプションとして action にはサーバ側のプログラムを示す URI、method には post か get を指定します。内容には form 要素を除くブロック要素を含むことができます。また終了タグは省略できません。

input input はインライン要素です。input 要素はブラウザの利用者に入力を促します。内容は指定できません。また終了タグは禁止されています。name オプションで送る情報の項目名を指定します。また type オプションには次のものが指定できます。

text 一行の入力領域を作ります。

password text と同様ですが、入力した文字はそのまま表示されず、星印 * のような記号が代わりに表示されます。

checkbox チェック可能なボタンが表示されます。チェックすると value オプションで与えられた値が設定されます。

radio チェック可能なボタンが表示されます。チェックすると value オプションで与えられた値が設定されます。但し、name オプションで指定した項目名が同じ input 要素が複数あると、一番最後にチェックしたものだけが有効になります。

Web 演習 18

submit 提出用のボタンが作られます。これにより入力した情報がサーバ側のプログラムに送られます。value を設定するとボタンのラベルになります。

reset このボタンを押すと、ブラウザは全ての入力欄を初期値に戻します。

file ファイルを選ぶ画面が出て、指定したファイルが送られます。method="post" で行うべきです。

hidden 画面には何も出ず、value オプションで設定した値がそのまま送られます。

image 画像による提出ボタンを作ります。src オプションで画像の URI を示し、alt オプションで画像が表示できない時に表示する文字を指定します。

button value オプションが埋め込まれたボタンが作られます。

button button 要素はインライン要素です。この働きは input 要素の type オプションに button を指定したものと同様ですが、内容がそのままボタンとして使われることが違います。name、value オプションは input と同様です。type オプションには、submit(送信)、reset(リセット)、button(汎用のボタン) のいずれかを指定します。

内容は form 関連の要素と a 要素を除く、インライン、またはブロック要素を含むことができます。終了タグは省略できません。

textarea textarea 要素はインライン要素です。これは複数行の入力が可能なフィールドを作成します。row オプションで行数を、cols オプションで幅を指定します (必須)。name オプションで項目名を指定します。内容には文字列を含むことができ、初期値として表示されます。終了タグは省略できません。

select select 要素はインライン要素です。これはメニューを作る要素です。内容には option 要素や optgroup 要素を複数含むことができます。終了タグは省略できません。name オプションで指定した項目名に対して、メニューで選択したオプションの値が送られます。

option option 要素は select 要素に含まれ、メニューを構成します。内容には文字列を含むことができ、それがメニューの見出しを作ります。value オプションを指定すると、選ばれた時、その値が送信されますが、value オプションを指定していない場合は内容がそのまま送られます。終了タグは省略できます。

optgroup optgroup 要素は option 要素をグループ化するための要素です。内容には option 要素を複数含むことができます。グループ名として label オプションを必ず指定する必要があります。終了タグは省略できません。

3.4.3 フォームの例

ここでは簡単なフォームの設計の例を示します。付録 A の JSP プログラム shukei.jsp は簡単な集計プログラムです。これにデータを送るフォームを作りましょう。このプログ

要素名	終了タグ	内容
form	必要	form 以外のブロック要素
input	禁止	指定しない
button	必要	form 関連と a 要素以外のブロック要素とインライン要素
textarea	必要	文字列
select	必要	option 要素と optgroup 要素
option	省略可能	文字列
optgroup	必要	option 要素

図 16: form 関連の要素

集計結果
和食 5
洋食 3
中華 7

図 17: shukei.ejs の出力イメージ

ラムは `ryori` という名前で 0 なら和食、1 なら洋食、2 なら中華を選択したものとして集計します (図 17)。

例 1 図 A.1 は、`input` 要素の `type` オプションに `text` を使った例です。基本的には `type` オプションに `text` を指定すれば何でも入力できます。しかし、今回の例のように値として 0, 1, 2 のみしか受け付けられない場合、それ以外の入力が可能なら、範囲外の入力に対してエラー処理をしなければなりません。こういう場合は入力の範囲が強制される方式に劣ります。

例 2 図 A.2 は `input` 要素の `type` オプションに `radio` を指定したものです。このようにラジオボタンを使うと、択一で選択できます。

例 3 図 A.3 は `select` 要素を使用した例です。`select` を使うと、画面上に選択していない情報を隠すことができます。印刷する場合などに有効です。反面、選択肢の一覧性が失われます。また、マウス操作に神経を使う必要があります。さらに、ブラウザによっては画面の日本語化はできていても、`select` メニューでは文字化けしてしまうものもあります。ラジオボタンとの使い分けは、画面のレイアウトのデザインの考え方で決めれば良いでしょう。

例 4 図 A.4 は `button` 要素を使用した例です。選ぶものが単一の場合は、このようにそれぞれの値をもつ送信ボタンを用意する手もあります。

但し、これは Internet Explorer では正常に動作しません。

3.5 CSS

3.5.1 背景

本来 HTML はコンピュータに文書処理をさせるためにマークアップする言語ですが、ブラウザの登場により、本来の意味を無視して見栄えを良くするためにタグが誤用されるようになりました。たとえば、h1 要素は本来大見出しを意味します。そのため、多くのブラウザでは目立つように大きい字を使って表示するようになっています。これを誤用し、単に大きい字を表示したい場合に <h1>と </h1>で囲まれた文書が現れるようになりました。しかし、このように誤用が行われると大見出しの処理(目次の作成など)ができなくなります。本来字を大きくするしないは文書の構造には関係ないため、HTML の守備範囲ではありませんが、このような誤解はあとを絶ちません。さらに、過去において、HTML 2.0 規格を無視して一部のブラウザが勝手に仕様に存在しない要素、例えば font 要素を利用できるようにしてしまいました。この不法な拡張によって一部のブラウザだけは文書が表示される際のフォントを限定できるようになってしまいました。そこで、HTML 3.2 ではブラウザの機種による混乱や避けるため、一部のブラウザのために実際に使用されていた要素を取り込み、その結果 HTML で文書の見栄えを良くできるようになりました。しかし、これは HTML の本来の目的ではない異常な拡張でした。

そこで、HTML 4 からは、文書の見栄えを良くする機能を HTML とは別に定め、それを利用できるようになっています。この仕組みが CSS(Cascading Style Sheet) と呼ばれるものです。また、それにともない、HTML 4.01 Strict からは見栄えの指定が原則廃止されました。

CSS による見栄えの指定は、ドキュメントの作成者の他、利用者がブラウザにあらかじめ指定することもできます。

3.5.2 CSS の書き方

次は CSS の簡単な例です。

```
h1 {color: blue}
```

このように CSS は「セレクト { 宣言 }」という簡単な構文からなっています。また、宣言は「プロパティ: 値」という形になっています。この例では、これを指定した HTML ドキュメントにおいて、h1 要素の内容が青文字で表示されるようになります。

スタイルシートを HTML ドキュメントで指定するには、いくつか方法がありますが、次のように link 要素を用いることができます。

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

HTML ドキュメントでは、特定の要素の内容に別の要素が入ります。例えば、body 要素の内容に p 要素が入るなど。これを親子関係呼んだり、body 要素が親で、p 要素が子などと言うことがあります。そして CSS では、親の要素に対する宣言は子の要素に継承されます。例えば、body 要素で指定した文字の色は p 要素に継承されます。もし、p 要

素で色を指定しない場合、`body` 要素の文字の色が使われます。一方、`p` 要素で色を指定すれば、その色が優先されます。

但し、宣言されるプロパティの種類によっては継承されないものもあります。背景画像の指定 `background` などは継承されません。仮に継承されるとすると、背景画像が、子の要素が始まる度に、内容表示の左上から表示されることになり、全体として、親要素を覆う一つの背景画像の上に、子要素の部分だけいくつも親要素と同じ画像が表示されることになります。このような表示はデフォルトの指定としては好ましくありません。むしろ、子要素の背景画像が指定されなければ、親要素の一枚の背景画像の上に、子要素が背景画像を持たずに表示されるべきです。そのため、基本的には背景画像の指定は継承せず、もし背景画像を指定しなければ背景は透明になるようになってます。このようにすると、子の要素が始まっても、単純に親の要素の背景画像が透けて見えるようになり、画面全体では一枚の背景が連続して表示されることになります。

3.5.3 セレクタの書式

セレクタにはいろいろな書式があります。HTML 要素はそのままセレクタになります。複数のセレクタに対して同じ宣言を行うには次のようにカンマ「`,`」で区切ります。

```
h1, h2, h3 {font-family: helvetica}
```

一方、複数の宣言を行うには、次のようにセミコロンで「`;`」区切ります。

```
h1 {
    font-weight: bold;
    font-size: 12pt;
    line-height: 14pt;
    font-family: helvetica;
    font-variant: normal;
    font-style: normal;
}
```

クラスという考え方があります。次のような CSS を考えます。

```
.pastral { color: green }
```

この `.pastral` はクラスの名前を意味します。その時、HTML ドキュメントで次のようにクラスを指定すると、クラスへの宣言が有効になります。

```
<h1 class="pastral">Way too green</h1>
```

なお、同時にはひとつのクラスしか指定できません。

また、セレクタとして特定の親子関係を指定することもできます。

```
h1 em { color: red }
```

このように指定すると、`h1` 要素に含まれる `em` 要素のみに宣言が有効になります。なお、`h1` の `em` と `h2` の `em` に有効なセレクタは `h1` , `h2 em` ではなく、`h1 em` , `h2 em` と空

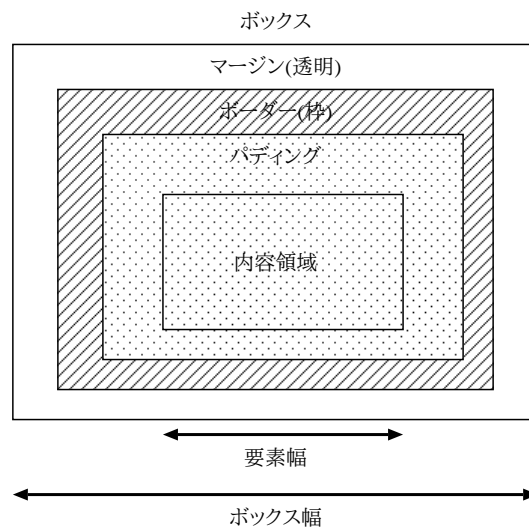


図 18: ブロック要素

白の結合の方が、,(カンマ)よりも強いことに注意して下さい。つまり、h1 と h2 の中に含まれる em を赤で強調するには次のようにします。

```
h1 em, h2 em { color: red }
```

一方、疑似クラスというものが用意されています。多くのブラウザではハイパーリンクが色付で用意されています。この色を疑似クラスを使って指定することができます。次のように疑似クラス link, visited, active を使うと、通常のリンク、既に訪れたリンク先、指定された状態のリンクの色を指定できます。

```
a:link { color: red }
a:visited { color: blue }
a:active { color: lime }
```

また、「一行目」「一文字目」を表す疑似クラスも用意され、文章の始めだけ表現を変えることができます(一部ブラウザでは実現されていません)。

```
p:first-letter { font-size: 200%; float: left }
p:first-line { padding-left: 1em }
```

3.5.4 表示のされ方

CSS での宣言が、どのように HTML ドキュメントの内容を表示されるかを説明します。

ブロック要素では、図 18 のようなモデルが仮定されています。マージンの大きさ、枠の太さや種類、パディングの大きさが指定できます。また、パディング領域の背景は background プロパティで指定できます。これはリスト内の要素 li, dt, dd も同じです。

また、CSS のプロパティには float があります。float プロパティが left または right という値に宣言されると、その要素はブロック要素として取り扱われます。例えば、下記

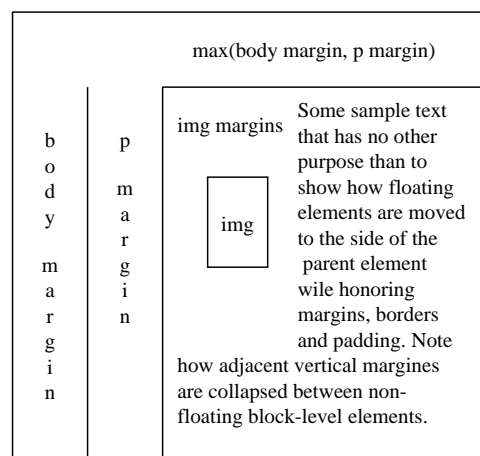


図 19: float(流し込み)

のように `img` 要素に `float: left` が指定されると表示は図 19 のように、`img` が左寄せでブロック要素として表示され、続くインライン要素が回り込むようになります。

```
<head>
<style type="text/css">
  img { float: left }
  body, p, img { margin : 2em }
</style>
</head>
<body>
<p>

Some sample text that has no other purpose than to show how
floating elements are moved to the side of the parent element
while honoring margins, borders and padding. Note how adjacent
vertical margins are collapsed between non-floating block-level
elements.
</p>
</body>
```

インライン要素も途中で改行されなければブロック要素と同じになります。但し、改行される場合、その位置での `margin`, `border`, `padding`, `text-decoration` の効果は無くなります。

3.5.5 プロパティ

ここでは CSS で指定可能な代表的なプロパティを紹介します。

継承するプロパティ

font-style, font-weight, font-size font-style はフォントスタイルを指定します。normal(立体的), italic(デザインが考慮された斜字体), oblique(単なる斜字体) が指定できます。

font-weight はフォントの太さを指定します。normal, bold, lighter, bolder の他、100, 200, 300, 400(normal と同じ), 500, 600, 700(bold と同じ), 800, 900 を指定できます。

font-size はフォントの大きさを指定します。値としては、xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger の他、ポイント数、パーセント表示なども利用できます。

color color は文字の色を指定します。なお、color を指定する時は必ず background-color を指定します。そうでない場合、ブラウザのデフォルトの色と color が一致する可能性があり、最悪文字が全く読めなくなってしまうです。

text-indent text-indent は最初の行の字下げを指定します。単位付数値で字下げの幅を指定します。

text-align text-align は行そろえを指定します。left で左寄せ、right で右寄せ、center で中央揃え、justify で両端揃えを指定します。

line-height line-height は行の高さを指定します。normal でブラウザに任せます。単位付数値で実際の高さを指定します。単位なし数値では使っているフォントの高さにその数値を掛けた高さにします。パーセント値も同様です。

list-style-type 箇条書(li 要素)のマークを指定します。disc(黒丸), circle(白丸), square(四角), decimal(十進数), lower-roman(英字の小文字), upper-roman(英字の大文字), lower-greek(小文字のギリシャ文字), hiragana(ひらがなのいろは), none(なし)などを指定できます。

継承しないプロパティ

background-color, background-image, background-repeat background-color は背景色を指定します。color と一緒に指定します。指定しない場合は、透明になります。

background-image は背景に表示する画像を指定します。url 指定を使います。なお、背景画像は瞬時には表示されないため、背景画像が表示されていない時点でも文字が読めるように、background-color で文字が読めるような色を指定すべきです。

background-repeat は背景画像を繰り返し表示するかどうかを指定します。repeat-x は横方向だけ繰り返します。repeat-y は縦方向だけ繰り返します。repeat は縦横両方とも繰り返します。no-repeat は画像を一枚だけ表示します。

text-decoration text-decoration は文字の下線などを指定します。none で無指定、underline で下線、overline で上線、line-through で取消線、blink で点滅を指定できます。

vertical-align vertical-align は同じ行の中で縦方向の位置揃えを指定します。baseline はベースライン同士を揃えます。top はボックスの上を揃えます。middle はボックスの中心を揃えます。bottom はボックスの下を揃えます。

margin-top, margin-right, margin-bottom, margin-left マージンの長さを設定するプロパティです。実数値+単位で実サイズを指定できます。パーセント値を設定すると親ボックスの横幅に対する割合で指定できます (margin-top でも横幅を参照する)。auto は状況に応じて自動的に設定されます。なお、四方向同じ指定で良ければ margin というプロパティが使えます。

padding-top, padding-right, padding-bottom, padding-left パディングの長さを設定するプロパティです。実数値+単位で実サイズを指定できます。パーセント値を設定すると親ボックスの横幅に対する割合で指定できます (padding-top でも横幅を参照する)。なお四方向同じ指定で良ければ padding というプロパティが使えます。

border-top, border-right, border-bottom, border-left 三つの値を空白で区切って指定します。最初の値は枠の色を指定します。次の値は枠の幅を指定します。幅の指定は、実数値+単位か、thin, medium, thick のキーワードになります。三つ目の値は枠の形式を指定します。none で透明、hidden は枠なし、solid は実線、double は二重線、groove は枠が立体的にへこんで見えるようなデザイン、ridge は枠が立体的に突き出ているように見えるようなデザイン、inset は枠の内側がへこんで見えるデザイン、outset は枠の内側が突き出て見えるデザイン、dashed は破線、dotted は点線を表します。なお、四方向同じ指定で良ければ border というプロパティが使えます。

width, height ボックスの内容領域の幅を指定するのが width、高さを指定するのが height です。実数値+単位、パーセント値、キーワード auto が使えます。

float, clear float は要素をボックスとして扱い、右、または左に寄せ、その反対側に続く要素の内容を回り込ませます。left, right, none を指定できます。なお、float を指定する要素には、画像を除き width プロパティを指定する必要があります。

clear は float で指定された回り込みを解除します。右を解除させたいければ right、左を解除させたいければ left、両側を解除させたいければ both、解除しなければ none を指定します。

3.5.6 値の指定

実数値+単位 CSS で使用できる単位には、cm や mm の他、in (インチ)、pt (ポイント 1/72 インチ)、pc (パイカ 12 ポイント) と実際の長さを表すものが使えます。さら

に、コンピュータ毎に長さが異なる、画面上の画素を表す px や、実際に表示されるフォントのサイズを表す em、表示される文字 x の高さを表す ex を使うことができます。

パーセント値 50% などパーセントを付けた指定は、現在の値 (継承された値) に対しての割合を示します。オプションが継承するものでも、この指定自体は継承せず、変更された値 (新しく決まったフォントの太さなど) が継承されます。

比較級 bolder や smaller は現在の値に対して、さらに太くや小さくするという指定を表します。オプションが継承するものでも、この指定自体は継承せず、変更された値 (新しく決まったフォントの太さなど) が継承されます。

url 指定 画像の指定などは url 指定を使います。url(http://www.c.dendai.ac.jp/a.png) や、url(../b.png) などと書きます。

色 色の指定法は #aabbcc のように # 記号の後に二桁の十六進数を三つ指定します。これで RGB(赤、緑、青) の光の強さの度合を指定します。#ffffff で白、#000000 で黒、#ff0000 で赤、#00ff00 で緑、#0000ff で青を表します。この他、プロパティの値の指定には次のキーワードも使用できます。black(#000000 と同じ), Silver(#c0c0c0 と同じ), gray(#808080 と同じ), white(#ffffff と同じ), maroon(#800000 と同じ), red(#ff0000 と同じ), purple(#800080 と同じ), fuchsia(#ff00ff と同じ), green(#008000 と同じ), lime(#00ff00 と同じ), olive(#808000 と同じ), yellow(#ffff00 と同じ), navy(#000080 と同じ), blue(#0000ff と同じ), teal(#008080 と同じ), aqua(#00ffff と同じ)。

3.5.7 CSS の例

1. 背景を赤、文字を黒にします。

```
body {color: black;
      background-color: red}
```

このスタイルシートを default.css というファイル名で保存し、次の HTML 文書と同じ場所に保存 (例えば test.html など) すると、スタイルシートが正しく作用しているかどうか確かめられます。

```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="utf-8">
<title>test</title>
<link rel="stylesheet" type="text/css" href="default.css">
</head>
<body>
```

```
<p>テスト</p>
</body>
</html>
```

2. em 要素を赤く表示するには次のように CSS ファイルに書きます (但し、背景が赤くないと言う前提です)。

```
em {color: red}
```

3. h1 要素を中央寄せで黒字に白文字にし、赤の二重線で囲みます。

```
h1 {text-align: center;
    color: white;
    background-color: black;
    border: red medium double
}
```

4. h2 要素を左寄せにしますが、内容の左側に 50px × 50px の画像 (a.png) をアクセントで表示します。

```
h2 {text-align: left;
    padding-left: 50px;
    background-image: url(a.png);
    background-repeat: no-repeat
}
```

5. h3 要素で文字の両側に黄色い正方形を表示します。

```
h3 {border-left: yellow 1em solid;
    border-right: yellow 1em solid;
    padding-left: 1em;
    padding-right: 1em
}
```

6. p 要素で行間を広めにとり、枠で囲む。上下の間隔もあける。

```
p { margin-top: 6ex;
    margin-bottom: 6ex;
    line-height: 6ex;
    border: aqua thin solid;
}
```

7. p 要素の最初の行を一段下げる

```
p:first-line { padding-left: 1em }
```

3.6 DOM

DOM(Document Object Model)[5] は、HTML や XML のドキュメントをオブジェクトとして取り扱うための概念です。これはドキュメントに対するプログラミング言語のインターフェイスを提供します。

これはもともと、Dynamic HTML や DHTML という概念として生まれ、ブラウザからドキュメントのコントロールをするために Javascript 言語で使用されました。その後、XML が誕生して、XML ドキュメントの標準的な API として位置づけられました。

3.6.1 基本概念

HTML 文書や XHTML などの XML 文書は、タグで表現された要素の集まりとして表現されます。図 3 において、タグの付けられた要素において、情報は、「要素名」、「属性」、「内容」の部分に分けられます。属性は複数指定できますので、一つの要素は複数の属性を含んでいます。また、内容には他の要素や文書などが含まれます。例えば、HTML 文書は一つの html 要素であり、その要素は head 要素と body 要素を含んでいます。

さて、DOM はマークアップされた文書を Node の階層構造として表現します。Node として Element の他、Document, Attribute, Text があります。また、Node の集まりとして NodeList というデータ構造を持ちます。NodeList は length という値と、item(int) という Node をインデックスで取り出すメソッドがあります。

Node オブジェクトは、基本的に名前を持ち、また、属性の取得、生成、削除と、子 Node に対する、取得、生成、検索、削除ができます。

3.6.2 JavaScript の API

(未完) document 変数などについて書く

3.6.3 例

HTML のアプリケーション ここでは HTML ドキュメントの文書のアプリケーションを考えてみましょう。付録 B.1 は目次を作るアプリケーションです。

HTML のドキュメントにおいて、h2 などの見出しの内容だけを抽出して出力すれば目次として利用できます。データ構造とアルゴリズム I の講義資料は XHTML で書かれていますので、ページを保存してこのプログラムに入れると目次を生成します。

プログラムへのデータ入力 一方、プログラムに対して構造を持ったデータの入力について考えてみましょう。従来は単純にデータを列挙し、その羅列されたデータに対してプログラムが構造の解釈を与えてデータを受け取ってました。従来の入力方法ではデータの構造に対応した入力プログラムを作らなければならなかったので、複雑な構造を持つデータの入力部を単純化することは難しかったです。

```
[  
  { "no": 123, "name": "aaa", "price": 100 },  
  { "no": 456, "name": "bbb", "price": 200 },  
  { "no": 789, "name": "ccc", "price": 300 }  
]
```

図 20: データ例

しかし、JSON を使うことによって、プログラム中で使用するオブジェクトのフィールドに対応したデータをデータ側で作ることができるようになります。

図 20 は商品リストを表す JSON ドキュメントです。このような形式で入力データを用意すると、JavaScript で読み込んで要素を一つ一つ処理するだけで済むので扱いは単純です。

3.7 Node.js

JavaScript は、当初ブラウザ側から HTML を操作するために開発されました。

しかし、近年、一般的なプログラミング言語としての市民権を得つつあります。さらに、非同期通信機能を取り込み、Web サーバ Node.js が作られました。Javascript 自体はインタプリタなので、コンパイル言語よりも原理的には速度が劣りますが、非同期通信機能により高負荷に耐えうる（高負荷になっても、処理効率が落ちない）ところに注目されています。

本講義では、サーバに Node.js とそれに使用するライブラリとして、express と ejs を使用します。

3.7.1 Node.js の概要

3.7.2 express

3.7.3 セッション

Web の用途にはドキュメントの公開の他に、Form 入力を処理しながら、ユーザからデータを処理するものもあります。

オンラインショッピングモールなどでは、次のような処理が行われます。

1. ユーザの認証
2. 購入商品の登録の繰り返し
3. 清算と確認

ここで重要なことは、複数のページが組み合わされ、処理の順番が与えられて使われるということです。このように複数のページをひとまとまりにした管理単位を セッション といいます。セッションはユーザのクライアントごとに管理される必要があります。ところが、Web サーバは前述したように要求したページを単に送り返すだけの仕事しかしません。そのため、Web サーバの基本機能ではユーザごとのセッション管理を行うことができません。そのため、クライアントとサーバでセッション情報を協調して共有する必要があります。

このため、セッションを管理するため、ブラウザに送るデータの中にセッション情報を埋め込んでおき、ブラウザがサーバにデータを送る際にいっしょに再度回収するような手順をとります。インターネットで端末を区別するのに固有の IP アドレスが必要なように、異なるクライアントを区別するには固有のセッション情報が必要になります。セッション管理の手法がまずいと単に動作が不良になるばかりでなく、一見正常の動作しているようでも第三者に不正にセッションに介入されてしまうという重大なセキュリティの欠陥が生じることがあります。すると、第三者にセッションを乗っ取られてしまうと不正取引などが生じ多大な損害が生じる可能性があります。

セッション管理には様々な常識があり、この常識を守らないとセキュリティ上問題が生じます。このため、JSP などのアプリケーションサーバではセッション管理を自動的にできる仕組みを提供しています。

3.8 EJS

プログラミングにおいて、プログラムとデータを分離することが重要です。

データの構造がある程度簡単で、プログラムによって複雑な処理を行う場合は、データはファイルなど一カ所に納めておき、プログラムからハンドルという形で参照を行って処理を行います。

しかし、Web アプリケーションなどでは逆にプログラムの処理は単純ですが、ページのレイアウトなど、表示されるデータが複雑になっています。このような場合は逆にデータが主になり、プログラムがパラメータとして与えられるべきです。この場合、データ(ドキュメント)にプログラムが埋め込まれるような形態をとるべきです。但し、コンピュータはデータをそのまま実行できないので、プログラムが埋め込まれたデータは変換プログラムにより、意図したデータを出力するようなプログラムに変換されます。

EJS は JavaServer Pages(JSP) に似せて作られた JavaScript 用のテンプレートです。これは、HTML 内に指定した形のタグでプログラムを埋め込み、Node.js というサーバ側で解釈しプログラムをじっこうした結果を埋め込んで HTML に変更します。

3.8.1 EJS の基礎

EJS は基本的には HTML に `<%` と `%>` で囲まれた部分に JavaScript プログラムを埋め込んだものです。式の値をそのまま出力するために式を `<%=` と `%>` で囲んだ表現があります。(図 22)。

```

<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="utf-8">
<title>test</title>
<link rel="stylesheet" type="text/css" href="default.css">
</head>
<body>
<p>
<% var s="テスト" %>
<%= s %>
</p>
</body>
</html>

```

図 21: スクリプトレット

例を示します (図 21)。

3.8.2 include

EJP では include 関数により、他のファイルを含めることができます。

これは<%- include ファイル名 (, 引き継ぐ変数オブジェクト)%> で読み込みます。

なお、現行の EJS だけでは、処理内容によってページを分岐することができません。また、このタグを使用してもブラウザのアドレス欄の URI は変わらないので注意が必要です。

4 実験

4.1 実験装置の説明と原理

実験には EJS を解釈する Node.js による Web サーバを使用します。Web サーバは実験室では Linux OS の上で動かしていますが、パソコンにも簡単にインストールできるものです。基本的には実験室のサーバにファイルを転送し、Web ブラウザでアクセスします。

一方、実験者の環境はノートパソコンにドキュメントを作成することを想定しています。作成したドキュメントを Web サーバに配送するのに Windows のファイル共有機能を使います。

```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="utf-8" >
<title>test</title>
<link rel="stylesheet" type="text/css" href="default.css">
</head>
<body>
  <p>
    <%
      function plusone(x){return x+1;}
    %>
    <%= plusone(1) %>
    <% var num=3; %>
    <%= plusone(num) %>
  </p>
</body>
</html>
```

図 22: 定義と出力

MacOS で実験のサーバにアクセスする場合は Finder のサーバメニューにアドレスを入れてアクセスして下さい。

4.2 実験の準備

また、実験の際に必要なソフトウェアとして下記のものが持参するノートパソコンにインストールされ使えるようになっている必要があります。

1. Tera Term (Windows の telnet がを使用する場合は不要)
2. Tera Pad のような HTML を書くことができるテキストエディタ (メモ帳などでも可能)
3. 2 つ以上の Web ブラウザ (Edge を含めても良い)

さらにプログラムを理解するためには JavaScript に関するドキュメントを用意しておく必要があります。

1. JavaScript のドキュメント
2. Node.js のドキュメント

3. Express のドキュメント
4. Express-session のドキュメント
5. EJS のドキュメント

実験は2通りの方法で実施できます。

1. VPN などを使って大学に用意しているサーバにアクセスする
2. 自分のパソコンにサーバをインストールする

さらに、発展項目の実験を行うには次のソフトウェアが必要です。

1. Wireshark

4.3 実験上の注意

1. 個人実験で、かつ、物理的な実験ではありません。そのため、レポートにおいて他の班員名と気温と湿度は不要です。その代わり、パソコンの機種名や OS やブラウザなどの使用したハードウェアとソフトウェアの諸元はレポートに記載しなさい。
2. サーバーのホームページに最新の情報が提供されてますので、実験を始めてネットワークが開通したら、始めに参照しなさい。
3. ファイルの修正は必ずローカルのパソコン上で行い、修正を終えたファイルをサーバーにコピーするようにしなさい。

4.4 ネットワークの接続

以下、3種類のサーバの接続方式を示します。それぞれのサーバのアドレスは次のように異なります

実験室内 <http://172.31.1.1/>

VPN <http://jikken1.net.c.dendai.ac.jp/>

自分の PC <http://localhost:8000/>

以後の実験書では実験室内での表記になりますが、適宜上記のアドレスに読み替えて下さい。

Web 演習 34

4.4.1 実験室内でのネットワーク接続

1. ネットワークケーブルを接続します。
2. コマンドプロンプトまたは Power Shell を表示します。
3. `ipconfig | more` コマンドで IP アドレスに 172.31.1.x というアドレスが割り当てられていることを確認します (次のページに行くにはスペースキーを押します)。ここで通信できない場合は HUB のランプ、ケーブルなど物理的な接続を確認します。169.254.x.x というアドレスが割り当てられているときは `ipconfig /renew` コマンドを実行して下さい。
4. `ping 172.31.1.1` でサーバと通信できることを確認します。ここで通信ができない場合は、`ping` コマンドが使用可能か？ファイアウォールが適切に設定されているかを確認します。
5. スタートボタンを右クリックして、エクスプローラを起動します (インターネットエクスプローラではありません)。
6. アドレスを打ち込むところがあるか確認してください。なければ ALT キーを一回押した後、「表示 ツールバー」でアドレスバーを有効にしてください。但し、バーの右端に「アドレス」とだけ表示される場合は、「表示 ツールバー」の「ツールバーを固定する」を無効になっていることを確認し、「アドレス」という部分を左に移動してアドレスの入力欄を画面に出します。
7. アドレスの入力欄に「\\172.31.1.1\」と入れて下さい。ID とパスワードが要求される場合は、ID を「guest」にし、パスワードは「」(なし)を指定します。そして、「jikken」というフォルダが表示されれば接続成功です。なお、バックスラッシュ記号\は、表示フォントにより円記号となることに注意すること。
また、セキュリティの設定上、ログイン名とパスワードを求められた場合、ログイン名は「guest」、パスワードには何も入力せずに進めて下さい。

4.4.2 VPN を利用した学外からの大学のサーバ利用

1. VPN で大学に接続します。
2. コマンドプロンプトを表示します。
3. スタートボタンを右クリックして、エクスプローラを起動します (インターネットエクスプローラではありません)。
4. アドレスを打ち込むところがあるか確認してください。なければ ALT キーを一回押した後、「表示 ツールバー」でアドレスバーを有効にしてください。但し、バーの右端に「アドレス」とだけ表示される場合は、「表示 ツールバー」の「ツール

バーを固定する」を無効になっていることを確認し、「アドレス」という部分を左に移動してアドレスの入力欄を画面に出します。

5. アドレスの入力欄に「`\\jikken1.net.c.dendai.ac.jp\`」と入れて下さい。ID とパスワードが要求される場合は、ID を「`guest`」にし、パスワードは「」(なし)を指定します。パスワードだけが求められたら、そのまま Enter を入力します。そして、「`jikken`」というフォルダが表示されれば接続成功です。なお、バックスラッシュ記号`\`は、表示フォントにより円記号となることに注意すること。

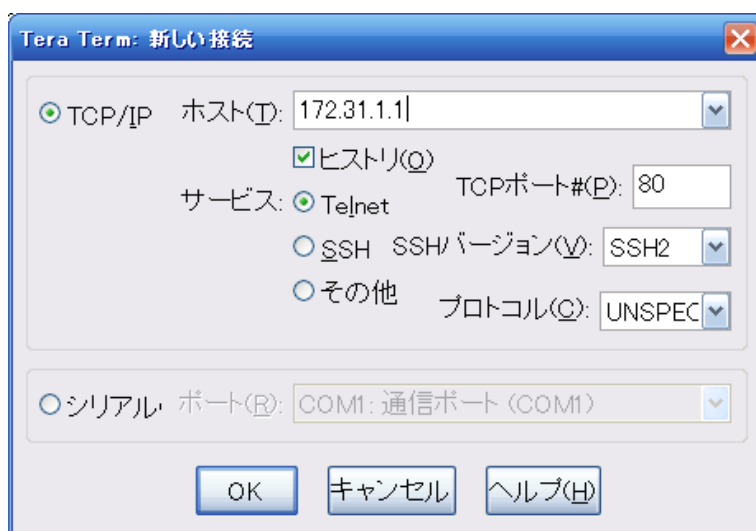
4.4.3 自分のパソコンにサーバをインストール

1. `http://edu.net.c.dendai.ac.jp/` の授業資料ページより、情報通信基礎実験のページに行き、サーバキット `serverkit.zip` をダウンロードします。
2. サーバキットのファイルを解凍します。
3. ドキュメントに従ってインストールします
4. サーバを起動します
5. ブラウザで `http://localhost:8000/` にアクセスしてホームページが出ることを確認する
6. ブラウザで `http://localhost:8000/jikken/mall/touroku.html` にアクセスして、商品登録画面を表示し、そのままボタンを押して、ダミーの商品が登録されることを確認する。
7. なお、実験でドキュメントを入れるフォルダは、解答したフォルダ内の `jikken` フォルダ内にさらに学籍番号のフォルダを作って入れて下さい。

4.5 実験 1

HTTP の演習を行います。Tera Term、あるいは、Windows に付属してくる `telnet.exe` を使って、Web サーバに TCP 接続を試みます。なお、`telnet.exe` は Windows Vista 以降は標準では使用できません。標準で使用できない場合は Tera Term を使用するか、あらかじめ Windows のコントロールパネルの「プログラム」の中の「Windows の機能の有効化または無効化」メニューで Telnet クライアントを有効にして、使用できるようにしてください。

以下の手順により WWW サーバから情報を取り出さない。



(telnet をマークしてから、TCP ポートを入力する)

図 23: 接続

Tera Term

1. Tera Term を起動します。起動後の接続の画面では図 23 のようにサーバのアドレスとポート番号 80 を指定します。

接続方法	サーバ	ポート
実験室	172.31.1.1	80
VPN	jikken1.net.c.dendai.ac.jp	80
PC	localhost	8000

2. 次に、設定メニューから端末を選び、出力改行コードを CR+LF に、漢字コードを SJIS に、ローカルエコーに設定します (図 24)。
3. さらに、設定メニューから TCP/IP を選び、「自動的にウィンドウを閉じる」のチェックを外します (図 25)。
4. 空のウィンドウに「GET /index.html HTTP/1.0 」と打ちます (は Enter キーを表します)。大文字、小文字は区別されますので注意して下さい。なお、入力した文字は画面に表示されません。
5. サーバからメッセージが送られてきて接続が切られます。

telnet の場合

1. コマンドプロンプトを動かします。
2. 「telnet 172.31.1.1 80 」など「telnet サーバアドレス ポート番号 」と打ちます (は Enter キーを表します)。



図 24: 端末

3. 「GET /index.html HTTP/1.0」と打ちます。大文字、小文字は区別されますので注意して下さい。なお、このコマンドでは「ローカルエコー」つまり押した文字を画面に表示させる機能はありません。キーボードを押しても画面に何も表示されませんが、それで正常です。
4. サーバからメッセージが送られてきて接続が切られます。

この手順で `http://サーバ/index.html` へアクセスしたことになります (図 26)。

この実験により、ブラウザに URI を入力して、HTML のドキュメントが得られるまでに、ブラウザが URI をどのように解釈してサーバにアクセスしているか、流れを説明しなさい。

なお、最初のステータスコードが 200 で無い場合は入力ミスなどを起こしていますので、正しい結果が得られるまでやり直して下さい。

4.6 実験 2

ここでは基本的な HTML のマークアップの実習を行います。図 27 の文書をマークアップしなさい。そして、Another HTML Lint 5[6] を使用して採点を受けなさい。なお、図のドキュメントを HTML にする際、`br` 要素を使ってはいけません。

1. 自分の学籍番号の名前を持つフォルダを作ります。
2. Terapad や meadow やメモ帳などで HTML ドキュメントを作ります (仮にファイル名を `ex2.html` とします)。1 に作成したフォルダに保存します。

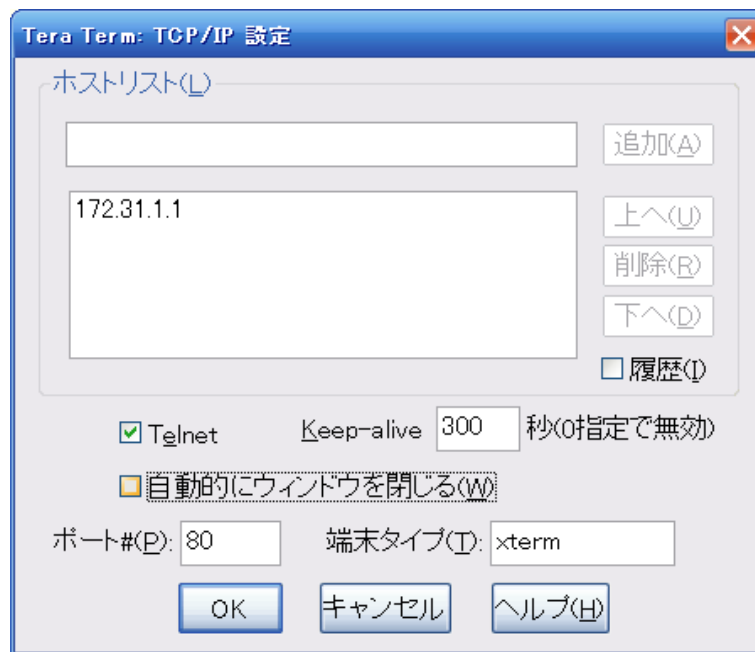


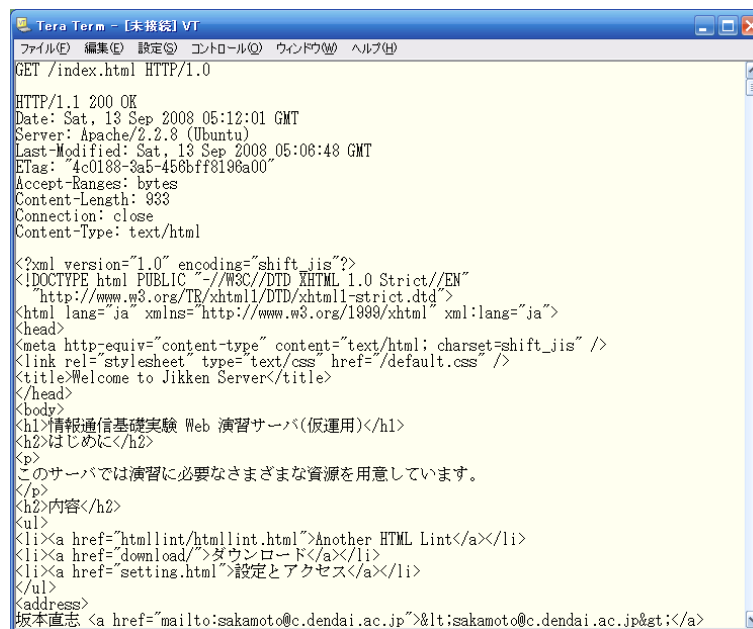
図 25: TCP/IP の設定

3. エクスプローラのアドレス欄に \\サーバ名 を打ち込み、jikken フォルダの中に、1 で作成した学籍番号のフォルダをコピーします。(PC の場合は jikken フォルダ内で行って下さい)
4. WWW ブラウザで <http://サーバアドレス/jikken/学籍番号/ex2.html> にアクセスして、マークアップに対して適度に整形された文書が出力されるのを確認下さい(このテキストに印刷してあるイメージ通りにはなりません)。
5. Another HTML Lint 5[6] にアクセスし、データを指定して、作成した HTML ファイルをチェックし、100 点になるようにします。以後、HTML Lint の出力は必ず 100 点になるようにします。

なお、ネットワークフォルダ内の HTML ファイルをエディタで直接開かないこと。必ずローカルで作成し、完成してからネットワークフォルダにコピーしてください。これは、ネットワークフォルダ内のファイルを直接修正している際にネットワークのトラブルが生じると、ファイルが消えてしまうなどの不具合が生じることがあるからです。

4.7 実験 3

実験 2 で作成した HTML ドキュメントにスタイルシートを与え、見栄えを良くしなさい。スタイルシートは別のファイルで与え、実験 2 で作成した HTML ドキュメントに link 要素のみを加えて読み込むようにしなさい。つまり、body 要素内は実験 2 と同一である必要があります。



```

Tera Term - [未接続] VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Sat, 13 Sep 2008 05:12:01 GMT
Server: Apache/2.2.8 (Ubuntu)
Last-Modified: Sat, 13 Sep 2008 05:06:48 GMT
ETag: "4c0188-3a5-456bff8196a00"
Accept-Ranges: bytes
Content-Length: 933
Connection: close
Content-Type: text/html

<?xml version="1.0" encoding="shift_jis"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="ja" xmlns="http://www.w3.org/1999/xhtml" xml:lang="ja">
<head>
<meta http-equiv="content-type" content="text/html; charset=shift_jis" />
<link rel="stylesheet" type="text/css" href="/default.css" />
<title>Welcome to Jikken Server</title>
</head>
<body>
<h1>情報通信基礎実験 Web 演習サーバ(仮運用)</h1>
<h2>はじめに</h2>
<p>
このサーバでは演習に必要なさまざまな資源を用意しています。
</p>
<h2>内容</h2>
<ul>
<li><a href="htmlint/htmlint.html">Another HTML Lint</a></li>
<li><a href="download/">ダウンロード</a></li>
<li><a href="setting.html">設定とアクセス</a></li>
</ul>
<address>
坂本直志 <a href="mailto:sakamoto@c.dendai.ac.jp">&lt;sakamoto@c.dendai.ac.jp&gt;</a>

```

図 26: 接続例

Edge の他、Firefox、Opera、Safari、Internet Explorer など複数のブラウザで見栄えの指定がどう反映されるか確認しなさい。レポートでは、どのような目的でスタイルを設計したかを記しなさい。なお、段落 (p 要素) は枠で囲み、箇条書は背景色を変更して下さい。この時、スタイルが他の要素と排他的になるようにしてください。つまり、箇条書きに枠をつけたり、段落の背景を変えてはいけません。

なお、画面で鮮明に見えていても、レポートで報告する際に表示がつぶれて見えなくなってしまうことがあるので、コントラストをつけるように注意すること。例えば、黒地に青字や、白地に黄色字などは画面で確認できても、印刷すると読めなくなります。濃い色と薄い色を組み合わせるように心がけること。

なお、ここで、実験 2 に CSS を指定するために link 要素を付け加えただけの HTML ドキュメントを「実験 3 の HTML」と呼ぶことにします。

さらに、この「実験 3 の HTML」も Another HTML Lint でチェックし 100 点であることを確認しなさい。

4.8 実験 4

1. ex3.html をコピーし ex4.html を作りなさい。そして、head の閉じタグ付近を図 28 のように改変しなさい
2. 付録 B.1 をコンパイルしなさい。なお、実験サーバより newmallsrc.zip をダウンロードすると、この実験を含め、付録にあるプログラムなど必要なファイルがすべて入手できます。ダウンロード、解凍し、得られたファイルやフォルダを ex4.html と同じフォルダに置きます。

情報通信ショッピングモールによるこそ

学籍番号 19ec999 の坂本様。現在のお取り引き金額は 12345 円です。

商品注文

下記の商品の注文数を指定することにより、商品の注文ができます。

- 商品番号 999001 商品名 りんご 単価 300 円 数量 0 個
- 商品番号 999002 商品名 ぶどう 単価 400 円 数量 0 個
- 商品番号 999003 商品名 みかん 単価 200 円 数量 0 個
- 商品番号 999004 商品名 もも 単価 200 円 数量 0 個
- 商品番号 999005 商品名 いちご 単価 500 円 数量 0 個

注文の流れ

1. 学籍番号と名前の入力
2. 商品の注文
3. 商品の注文状況と合計金額の確認
4. 発注総金額の確認

図 27: 文例

そして、ブラウザで表示し、冒頭に下記が表示されることを確認しなさい

- (a) 情報通信ショッピングモールによるこそ
- (b) 商品注文
- (c) 注文の流れ

なお、演習中にファイルの記述に矛盾が生じたときは実験 2 からやり直すこと。実験 2, 3, 4 で一貫したファイルになっていないと不合格です。

4.9 実験 5

1. 下記の書式で商品番号、商品名、単価の入った JSON ファイルを作ります。商品番号の上三桁は学籍番号の下三桁として下さい。商品名、単価は自由に考えて下さい。

```
[{"no": "999001", "name": "dummy1", "price": 100},  
{"no": "999002", "name": "dummy2", "price": 200},  
{"no": "999003", "name": "dummy3", "price": 300},
```



```
<script src="mokuji.js"></script>
</head>
<body onload="mokuji()">
<nav id="mokuji">
</nav>
```

図 28: mokuji.js の読み込み

```
{ "no": "999004", "name": "dummy4", "price": 400 },
{ "no": "999005", "name": "dummy5", "price": 500 }]
```

2. <http://サーバアドレス/jikken/mall/toroku.html> にアクセスして登録して下さい。

なお、toroku 関連のプログラムは付録 C にあります。

4.10 実験 6

付録 D.4 は基礎的な電子ショッピングサイトのプログラムリストです。6 つの ejc ファイルから作られています。

1. 始めにこれらのファイルをサーバの自分のフォルダにコピーし、<http://サーバアドレス/jikken/学籍番号/login.jsp> にアクセスします。そして、正常に動作することを確認下さい(これは報告する必要はありません)。
2. 次に、このファイルを変更して自分の独自のデザインに下さい。最低でも次の点を実現下さい。
 - (a) ショッピングモールの名称を与える。また問い合わせ先として自分のメールアドレスを入れる。
 - (b) 顧客の名前を表示する際に必ず敬称を入れる。金額を表示する際に必ず通貨単位も表示する。
 - (c) 各ページに統一したスタイルを与える(実験 3 で作成したスタイルシートを拡張しても良い)
3. 発展項目として余裕があれば下記の課題も行いなさい。
 - (a) 表示を日本語にする
 - (b) 商品などのリストをテーブル形式で表示する
 - (c) 自分が登録した商品以外は表示しない

Web 演習 42

報告書を作成するために、実験中に完成した画面のハードコピーをファイルに保存すること。

また、さらに変更後の文法が正しいかどうかを HTML Lint で確認しなさい。URL の入力ではうまくいかないはずなので、次のようにしてチェックしてください (Chrome ブラウザではうまくできないかもしれません)。

1. ブラウザにおいてチェックしたい画面を表示します
2. ソースを表示し、全選択し、コピーします (Microsoft Edge では一旦 F12 を押すと開発者モードになり、右クリックのメニューに「ソースを表示」が追加されます)
3. HTML Lint で DATA の欄にコピーし、DATA を選択してチェックします

4.11 実験 7

(発展学習) 余裕がある場合は下記のセキュリティ関連の実験を行いなさい。これは必須ではありません。なお、以下で対象とするサーバーのプログラムは実験 6 で使用した EJS ファイルです。

1. 各ページで、ブラウザの戻るボタンを押した後、処理が続けられるページとそうでないページを分類しなさい。
2. 各入力項目でエラーが起きるような例を見付けなさい。無い場合は無いで構いません。
3. WireShark プログラムを使い、各ページにおいてどのようなリクエストがサーバに送られるか調べなさい。
4. 下記の記述を login 画面に入れると何が起きるか調べなさい。

id

```
</p><script language="javascript">window.alert(  
document.cookie);</script><p>
```

name a

5. 次のような URL をブラウザに打つとどうなるか調べなさい。

```
http://サーバアドレス/jikken/mall/mall.jsp?id=%3C%2Fp%3E%3Cscript%20  
language%3D%22javascript%22%3Ewindow.alert%28document.cookie%29%3B%3C  
%2Fscript%3E%3Cp%3E&name=a
```

4.12 レポートをまとめる上での注意点

1. 数学の問題で「下記の問題を解きなさい」に対して、「解きました」という解答がありえないのと同様に、「確かめなさい」「チェックしなさい」に対して「確かめました」や「チェックしました」などは解答としてありえません。必ず、方法、経過、結果の証拠の提示、考察などを行って下さい。
2. レポートはワープロでも構いません。また、本文は基本的に白黒で書きなさい。
3. 画面の出力はハードコピーをとり、カラー印刷にすること。他の部分は白黒でも良い。また、HTML や CSS などのソースファイルの出力は、ハードコピーではなくワープロを使用して成形してもよい。
4. レポートは必要な事項のみを簡潔に述べること。不要な記述は減点の対象になる。また、なるべく両面印刷を使用すること。但し、カラー印刷などで出力が汚くなってしまう場合は片面とする。
5. 気温、湿度などの実環境情報は不要であるが、実験に使用したハードウェア、ソフトウェアの仕様、バージョンなどは記述すること。

5 検討事項

5.1 必須事項

1. HTTP/1.0 では GET 要求に対して、リソースの返信を行うという一往復のみのプロトコルです。一方、HTTP/1.1 では基本的に接続を切らず、何往復もサーバとのやりとりが可能です。なぜ、このように改良されたのか、理由を考えなさい。

ヒント: TCP 接続、img 要素

2. 見栄えを HTML で指定せず、CSS で指定する利点を考えなさい。
3. HTML や CSS などは規格が定められているのに、異なるブラウザで見栄えが異なることの理由を説明しなさい。

ヒント 1 実験中に CSS のパラメータを全部指定しましたか?

ヒント 2 性能の悪いブラウザが存在することは理由にはなりません

4. 実験 6 のショッピングモールは不十分なプログラムである。これに一つ機能を付け足すとしたら、どのような機能を付け足すか? アイディアを説明し、実際にどのように追加するか検討しなさい。なお、個々の商品に対して情報を付加する場合、商品ごとに異なる情報を表示する必要があることに注意すること。
5. 各ドキュメントの作成にあたって、デザインする上で心がけたことを実際に作成した自分のページなどを使いながら具体的に説明しなさい。

Web 演習 44

6. 文書を HTML Lint にかけると、100 点でない場合もブラウザでは正常に表示される場合がある。このような時、HTML Lint で 100 点を目指すのはどのような意味があるか。

5.2 発展事項

1. 実験 6 のプログラムにはどのようなセキュリティ上の問題があるか、一つだけ取り上げて説明しなさい。
2. セッションの情報が他人に洩れると具体的に何ができるのか仕組みを説明しなさい。
3. 複数の画面構成をとるシステムで、個々の画面に直接アクセスが可能なき、どのような不都合が考えられますか？また、これを防ぐアイデアがあれば示しなさい。

A Form の例と集計プログラム

A.1 text を使った例

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <title>料理選択</title>
</head>
<body>
  <h1>料理選択</h1>
  <form action="shukei.ejs" method="post">
    <p>
      料理番号を選択して、submit ボタンを押して下さい(0:和食、1:洋食、2:中華
    )。
    <input name="ryori" type="text" value="">
    <input type="submit">
  </p>
</form>
</body>
</html>
```

A.2 radio を使った例

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <title>料理選択</title>
</head>
<body>
  <h1>料理選択</h1>
  <form action="shukei.ejs" method="post">
    <p>
      料理番号を選択して、submit ボタンを押して下さい
      <input name="ryori" type="radio" value="0">
和食、
      <input name="ryori" type="radio" value="1">
洋食、
      <input name="ryori" type="radio" value="2">
中華。
      <input type="submit">
    </p>
  </form>
</body>
</html>
```

A.3 select を使った例

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <title>料理選択</title>
</head>
<body>
  <h1>料理選択</h1>
  <form action="shukei.ejs" method="post">
    <p>
      料理番号を選択して、submit ボタンを押して下さい
    <select name="ryori">
      <option value="0">和食</option>
      <option value="1">洋食</option>
      <option value="2">中華</option>
    </select>
    <input type="submit">
  </p>
</form>
</body>
</html>
```

A.4 button を使った例

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <title>料理選択</title>
</head>
<body>
  <h1>料理選択</h1>
  <form action="shukei.ejs" method="post">
    <p>
      料理番号を選択して、submit ボタンを押して下さい
      <button name="ryori" type="submit" value="0">和食</button>
      <button name="ryori" type="submit" value="1">洋食</button>
      <button name="ryori" type="submit" value="2">中華</button>
    </p>
  </form>
</body>
</html>
```


A.5 shukei.ejs

```

<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <title>集計結果</title>
</head>
<body>
  <h1>集計結果</h1>
  <form action="shukei.ejs" method="post">
    <p>
      <%= req.ryori %>
    </p>
    <%
      ryorime=[ '和食', '洋食', '中華' ]
      if(typeof application.ryori == 'undefined'){
        application.ryori=[0,0,0];
      }
      if(typeof session.ryori == 'undefined'){
        session.ryori=[0,0,0];
      }
      application.ryori[req.ryori]++
      session.ryori[req.ryori]++
    %>
  <dl>
    <% for(var i=0; i<ryorime.length; i++){%>
      <dt>
        <%= ryorime[i] %>
      </dt>
      <dd>
        <%= application.ryori[i] %>
      </dd>
    <% }%>
    <% for(var i=0; i<ryorime.length; i++){%>
      <dt>
        <%= ryorime[i] %>
      </dt>
      <dd>
        <%= session.ryori[i] %>

```

Web 演習 50

```
        </dd>
        <% }%>
    </dl>
</body>
</html>
```

B DOM の利用例

B.1 見出し要素の内容の出力

```
function mokuji(){
    const nav = document.getElementById( 'mokuji' );
    const elem =document.getElementsByTagName( "*" );
    const regex = new RegExp( '^H[1-6]' );
    const mokuji = document.createElement( 'ol' );
    const m=elem.length;
    for (var i=0; i< m; i++){
        if(regex.test(elem[i].nodeName)){
            elem[i].setAttribute( 'id', i );
            var item = document.createElement( 'li' );
            var anker = document.createElement( 'a' );
            anker.setAttribute( 'href', "#"+i );
            var c=elem[i].childNodes;
            for(var j=0; j< c.length; j++){
                anker.appendChild(c[j].cloneNode(true));
            }
            item.appendChild( anker );
            mokuji.appendChild( item );
        }
    }
    nav.appendChild( mokuji );
}
```

B.2 商品リストの入力

C 商品登録

JSON データをサーバのオブジェクト `application.itemmap` に追加します。

プログラムは、入力、処理、出力に分割するため、`toroku.html` は入力、`toroku.ejs` は登録処理を、`torokubody.ejs` は登録後の登録状況の表示を行うようになっています。

C.1 `toroku.html`

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="utf-8">
    <title>商品登録</title>
  </head>
  <body>
    <h1>商品登録</h1>
    <form action="toroku.ejs" method="post">
      <p>
        商品のリストを JSON で入力して下さい。
      </p>
      <textarea name="jsondata" cols="80" rows="10">
        [{ "no": "999001", "name": "dummy1", "price": 100 },
          { "no": "999002", "name": "dummy2", "price": 200 },
          { "no": "999003", "name": "dummy3", "price": 300 },
          { "no": "999004", "name": "dummy4", "price": 400 },
          { "no": "999005", "name": "dummy5", "price": 500 } ]
      </textarea>
      <p>
        <input type="submit">
      </p>
    </form>
  </body>
</html>
```

C.2 toroku.ejs

```
<%  
  if(typeof application.itemmap == 'undefined'){  
    application.itemmap={};  
  }  
  var j=JSON.parse(req.jsondata);  
  for(var i = 0; i<j.length; i++){  
    application.itemmap[j[i].no]=j[i];  
  }  
  %>  
<%- include( 'torokubody.ejs ',{itemmap: application.itemmap}) %>
```

C.3 torokubody.ejs

```

<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="utf-8">
    <title>商品リスト</title>
  </head>
  <body>
    <h1>商品リスト</h1>
    <table border="1">
      <thead>
        <tr>
          <th>商品番号</th><th>商品名</th><th>単価</th>
        </tr>
      </thead>
      <tbody>
        <%
          var keys = Object.keys(itemmap);
          keys.sort();
          for(var key of keys){
            %>
            <tr>
              <td><%= itemmap[key].no %> </td>
              <td><%= itemmap[key].name %> </td>
              <td><%= itemmap[key].price %> </td>
            </tr>
            <% } %>
          </tbody>
        </table>
      </body>
    </html>

```

D ショッピングモールの設計

本演習では簡単な電子ショッピングモールを作成します。但し、実験では Web デザイナー的な役割を演じるため、ビジネスロジックと呼ばれるプログラムの処理は全て提供します。本章では雛型となったビジネスロジックの説明を行います。

動作の基本方針は次の通りです。

1. ユーザごとに買い物ができる。
2. ユーザは商品を選んで数量を指定することで注文ができる。
3. 購入すると、購入した金額がユーザごとに積算される。

このような動作を実現するため、以下のように考察を行います。

まず、このショッピングモールにおいて管理するのは主に次のデータです。

1. ユーザ情報
2. 商品情報

ここでは、ユーザ情報として、学籍番号、ユーザ名、購入金額を、商品情報として、商品番号、商品名、単価を管理します。これらはアプリケーションの起動から終了まで一貫して管理することにします。

つまり、同一ユーザは何回でも買い物ができ、毎回同一の商品の注文もできます。また、複数回の買い物をした後、積算金額が計算されます。

この、ユーザ情報と商品情報はデータベース的な検索が可能なこととします。つまり、検索の鍵を持ち、それぞれ学籍番号と商品番号を指定することとします。

D.1 画面の設計

次に、ショッピングモールを実現するには複数の画面が必要になります。ここでは必要な画面を定義し、それらの関係を検討します。

ショッピングは、ユーザ認証をした後、商品を選んで、確認を行います。そのため、次のような画面が必要になります。

1. まず、ユーザ認証が必要なのでこれを login 画面と呼ぶこととします。
2. 次にユーザ認証が終了したら、商品を選ぶ画面が必要です。これを mall 画面と呼ぶことにします。
3. 商品を選択した後、確認が必要です。これを confirm 画面と呼ぶことにします。
4. 最後に商品を購入した旨を表示し、logout する logout 画面を用意します。

このように 4 画面が必要となります。次に、これらの 4 画面の機能を考えます。

D.1.1 login

login 画面ではユーザの入力により認証を行います。ここでは、学籍番号とユーザ名を入力させます。本来はパスワードを利用するのが普通ですが、簡単のために省略します。なお、次の画面に移行する前に以下の処理を行います。

1. 学籍番号が未登録の場合は新規として扱い、ユーザ名を登録し、購入金額を 0 としてデータベースにレコードを追加する。
2. 一方、既に学籍番号が登録されている場合は、ユーザ名を登録情報と比較して、異なる場合は login 画面に戻る。一致している場合は次の画面で購入金額を表示する。

D.1.2 mall

mall 画面では次の機能が必要です。

1. login が成功した旨の表示
2. 商品の表示
3. 注文数の入力

D.1.3 confirm

confirm 画面では次の機能が必要です。

1. ユーザの情報 (ユーザ名、購入金額) の表示
2. 購入品目、個数、小計、合計の表示
3. 確認の入力

確認の画面では承認か非承認かを入力します。承認であれば合計金額を加算します。

D.1.4 logout

logout 画面では、取引の詳細を表示し、一連の処理を終了する。

D.2 JSP に対する設計 1

EJS は Web のドキュメントに処理が従属する形態になる。そのため、ユーザへの入力の要求と、入力された値の処理は別ドキュメントで行うことになる。一方、入力された値の処理の後、次の画面を表示するが、これはビジネスロジックと画面表示という別の流れになるので、これは積極的に分離した方がよい。

したがって、EJS では次のような方針でプログラムを作成します。

1. Form によるデータ入力画面
2. Form の飛び先でのデータ集計 (ビジネスロジック) その後、表示画面へ飛ぶ
3. データ表示

ここで、EJSP で別ドキュメントを読み込む include はアドレスバー上のアドレスを変えないことに注意する。つまり、form の飛び先のビジネスロジックが本来の画面の持つべき URL になるようにするため、サーバのアドレスを変える必要がある。

D.2.1 login

login 画面では学籍番号とユーザ名の入力要求を行う。入力値は mall 画面に送る。

D.2.2 mall

login から送られてきたデータに対して次の処理を行う。

1. ユーザ入力が無ければ login 画面に移行する
2. ユーザ入力があったら、ユーザの認証を行う

その後、ユーザの番号、名前、購入総額を表示する。そして、商品の一覧を表示し、商品の発注個数を入力させる。入力値は confirm 画面に送る

D.2.3 confirm

confirm 画面では次のような処理が必要になる。

1. ユーザ入力が無ければ login 画面に移行する
2. 発注個数などにエラーがあったら mall 画面に移行する

その後、ユーザの番号、名前、購入総額を表示する。そして、発注状況の一覧と発注総額を表示する。確認ボタンを入力させる。

D.2.4 logout

logout 画面では次のような処理が必要になる。

1. ユーザ入力が無ければ login 画面に移行する

その後、ユーザの番号、名前、購入総額、発注状況の一覧を表示する。ログアウトした旨を表示する。

D.3 技術的検討

D.3.1 データベースの実現

データベースを実現するのに、簡単のために JavaScript の連想配列を使う。これはキーと値を保存し、キーの検索により順に値を取り出せるものである。これをユーザ情報、商品情報ともに使用する。

D.4 プログラム

D.4.1 login.ejs

```

<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="utf-8">
    <title>Login</title>
  </head>
  <body>
    <h1>Login</h1>
    <%
      if(typeof session.errorMessage != 'undefined'){
    %>
    <p>
      <em><%= session.errorMessage %></em>
    </p>
    <% } %>
    <form action="mall.ejs" method="post">
      <p>
        id: <input name="id" type="text" value="">
        name: <input name="name" type="text" value="">
        <input type="submit">
      </p>
    </form>
  </body>
</html>

```

D.4.2 mall.ejs

```
<%  
if(typeof application.usermap == 'undefined'){  
    application.usermap={};  
}  
var user;  
if(application.usermap[req.id]){  
    user = application.usermap[req.id];  
    if(req.name != user.name){  
        session.errorMessage = "Wrong_name";  
        return false;  
    }  
}else{  
    user={"id":req.id,"name":req.name,"amount":0};  
    application.usermap[user.id]=user;  
}  
delete session.errorMessage;  
session.user=user;  
return true;  
%>
```

D.4.3 mallbody.jsp

```

<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="utf-8">
    <title>Login</title>
  </head>
  <body>
    <h1>Mall</h1>
    <p>
      <%
        var user=session.user;
      %>
      <%= user.id %>
      <%= user.name %>
      <%= user.amount %>
    </p>
    <%
      var keys = Object.keys(application.itemmap).filter(function(x){
        return x>"0" && x<"zzzzzz";});
      keys.sort();
    %>
    <form action="confirm.ejs" method="post">
      <ul>
        <%
          for(var key of keys){
            var item = application.itemmap[key];
          %>
          <li>
            <%= item.no %>
            <%= item.name %>
            <%= item.price %>
            <input name="item<%=item.no_%>" type="text" value="0">
          </li>
          <% }%>
        </ul>
        <p><input type="submit"></p>
      </form>
    </body>

```

Web 演習 62

</html>

D.4.4 confirm.ejs

```
<%  
var keys = Object.keys(req);  
var chumon = {};  
for(var key of keys){  
    chumon[/item(.*)/.exec(key)[1]] = req[key];  
}  
session.chumon = chumon;  
%>  
<%- include('confirmbody.ejs',{chumon: chumon, session: session, application:
```

D.4.5 confirmbody.ejs

```

<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="utf-8">
    <title>Confirm</title>
  </head>
  <body>
    <h1>Confirm</h1>
    <p>
      <%
        var user=session.user;
      %>
      <%= user.id %>
      <%= user.name %>
      <%= user.amount %>
    </p>
    <%
      var keys = Object.keys(chumon);
      keys.sort();
    %>
    <ul>
      <%
        var total=0;
        for(var key of keys){
          var item = application.itemmap[key];
          if(chumon[key]>0){
            <li>
              <%= item.no %>
              <%= item.name %>
              <%= item.price %>
              <%= chumon[key] %>
              <%= total += item.price*chumon[key]; %>
            </li>
            <% }%>
          <% }%>
        }
      %>
    </ul>
    <hr>

```



```
<p>
  <%= total %>
  <%= session.total=total; %>
</p>
<form action="logout.ejs" method="post">
  <p><input type="submit"></p>
</form>
</body>
</html>
```

D.4.6 logout.ejs

```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="utf-8">
    <title>Logout</title>
  </head>
  <body>
    <h1>Logout</h1>
    <%
      var user=session.user;
      user.amount += session.total;
    %>
    <p>
      <%= user.id %>
      <%= user.name %>
      <%= user.amount %>
    </p>
    <% session.destroy((err)=>{}) %>
  </body>
</html>
```

E サーバ

サーバは Node.js + express + express-session + ejs を利用して作りました。

なお、login して成功したら、コンテンツに移動するという仕組みを作るのに express のプログラムの中で処理しました。一般的には passport というパッケージを使うそうです。

E.0.1 server.js

```
var path = require('path')
var express = require('express')
var session = require('express-session')
var home = express()
var work = express()
var mall = require('./jikken/mall/mall.js')

var application={}
work.use(session({
  secret: 'web_enshu.c.tdu',
  resave: true,
  saveUninitialized: false,
  cookie: { secure: false,
    maxage: 1000*60*30}
}))
work.engine('.ejs',require('ejs').renderFile)
work.set('views', './jikken')
work.set('view_engine', 'ejs')
work.get('/(.*\\.ejs)', function(req, res) {
  console.log("1_"+req.path)
  res.render(req.path.substr(1),{ application: application, req: req.query
});
work.use('/',express.static('jikken'))
work.use(express.json())
work.use(express.urlencoded({ extended: true }))
work.post('/(.*)/mall.ejs', function(req, res) {
  console.log("2_"+req.path)
  var p = /\/(.*\\\/)[^\\\/]*\/.exec(req.path)[1];
  if(mall.auth(application, req.body, res, req.session)){
    res.redirect("mallbody.ejs");
  }else{
    res.redirect("login.ejs");
  }
});
```

Web 演習 68

```
    }  
  });  
work.post( '/(.*)', function(req, res) {  
  console.log("3_"+req.path)  
  res.render(req.path.substr(1),{ application: application, req: req.body,  
});  
home.use(express.static( 'home'))  
home.use( '/jikken/',work)  
home.listen(8000)  
console.log( 'server_starts.');
```

参考文献

- [1] Susumu Terao. Tera Pad. <https://tera-net.com/>.
- [2] D. Crocker. STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES. RFC 822, IETF, August 1982.
- [3] P. Resnick. Internet Message Format. RFC 2822, IETF, April 2001.
- [4] P. Resnick. Internet Message Format. RFC 5322, IETF, October 2008.
- [5] Arnaud Le Hors, Philippe Le Hégarret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document object model(DOM) level 2 Core specification. Available at <http://www.w3.org/TR/DOM-Level-2-Core/>, November 2000.
- [6] XyXon. Another html-lint 5. <http://www.htmlint.net/>.
- [7] Håkon Wium Lie and Bert Bos. Cascading style sheets, level 1, April 3008. Available from <http://www.w3.org/TR/CSS1>.
- [8] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. Cascading style sheets level 2 revision 1 (CSS 2.1) specification, April 2016. Available from <http://www.w3.org/TR/CSS2>.
- [9] 大藤幹. 詳解 HTML & XHTML & CSS 辞典. 秀和システム, 2002.
- [10] HTML 4.01 specification, December 1999. Available from <http://www.w3.org/TR/html4/>.
- [11] T. Berners-Lee and D. Connolly. Hypertext Markup Language - 2.0. RFC 1866, IETF, November 1995.
- [12] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, IETF, May 1996.
- [13] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, IETF, June 1999.
- [14] ギャリー・バリンジャー, バラシィ・ナタラヤン. 独習 JSP. 翔泳社, 2002. 監修: 小野 哲, 訳: トップスタジオ.
- [15] 川崎克巳. サーブレット & JSP 逆引き大全 650 の極意. 秀和システム, 改訂第 3 版, 2007.