ホームページの基礎

山下弘隆

2008-02-25

目次

図目次

表目次

第 1 章

HTML の基本

1.1 お決まりの入門（言葉編）

このページでは、お決まりの言葉、要するに用語の簡単な説明をしておく。

1.1.1 ホームページ

以下のような意味で用いられることが多い。[*1]

1. Web ブラウザを起動した時や、 Web ブラウザのホームボタンを押した時に表示されるウェブ（Web）ページのこと。「**スタートページ**」とも言う。
2. Web サイトの入り口となるページで、 その Web サイトにとってのホームの意味。「**トップページ**」、あるいは、スタートページとも言う。
3. Web ページのこと。
4. Web サイトのこと。

1.1.2 Web サイト

ウェブページ群のこと。単に「**サイト**」と言うこともある。 例えば、企業のサイトと言った場合、その企業の Web ページ群を意味する。 また、Web サイトの管理者を「**Web マスター**」と呼ぶ。

1.1.3 Web ページ

“World Wide Web” 上にある個々の文書のことである。 単に「**ページ**」あるいは「**ホームページ**」と呼ばれることもある。 通常の紙媒体のページと異なり、 複数の Web ページ同士を相互に参照（これを「**ハイパーリンク**」と言う） する仕組みを持っている（この仕組みを「**ハイパーテキスト**」と呼ぶ)。

1.1.4 WWW（World Wide Web）

「**ワールド・ワイド・ウェブ**」は、 インターネット等で提供されるハイパーテキストシステム。単に「**Web（ウェブ）**」とも呼ぶ。 WWW にアクセスするためのソフトウェアを「**ユーザーエージェントあるいは WWW クライアント**」と呼ぶ。 特に閲覧を目的としたものは Web ブラウザ（「**WWW ブラウザ**」、あるいは単に「**ブラウザ**」）と呼ばれる。 通信の取り決めのことをプロトコル（通信規約）と言うが、 Web ページを閲覧するた

[*1] 本書では、“ホームページ” という言葉を用いる場合、“Web ページ” の意味で使用している。

めのプロトコルとしては、主に HTTP が使用される。 またページの記述には HTML や XHTML などのハイパーテキスト記述言語が使用され、 別の文書を参照するためのハイパーリンクとして URI（昔の URL のこと）を記述する。 この「**HTTP、HTML（XHTML）、URI（URL）**」の 3 つの技術によって WWW は広く普及した。

1.1.5　HTTP（Hypertext Transfer Protocol）

Web ブラウザと Web サーバの間で HTML などのコンテンツをやり取りするときの「**通信プロトコル**」である。「**ハイパーテキスト転送プロトコル**」とも呼ぶ。

1.1.6　HTML（HyperText Markup Language）

ハイパーテキストを利用して WWW 上で情報発信するための「**マークアップ言語**」（「**ページ記述言語**」とも呼ぶ）。「**XHTML（eXtensible HyperText Markup Language）**」とは、 SGML（マークアップ言語を記述するための言語。メタ言語[*2]とも呼ぶ。） で定義された HTML を「**XML**」（SGML の欠点を補った新しいメタ言語）で再定義したマークアップ言語。 HTML を習得すれば、XHTML の習得はそれほど難しくない。

1.1.7　URI（Uniform Resource Identifier）

一定の書式によってリソースを指し示す識別子のこと。「**URL（Uniform Resource Locator）**」を拡張したものである。 URL は、インターネット上の Web ページや電子メールの宛先を特定するための記号の並びのことであり、 俗語的表現としては「**アドレス**」とも呼ばれる。

1.2　とにかく作ろう

このページでは、小難しい説明はなし。とにかくホームページを作ってみよう。

まず、最初に使用するソフトウェアは、「**テキストエディタ**」と呼ばれるソフトウェアを使用する。 Windows なら “メモ帳”、Mac なら “テキストエディット” が、このソフトウェアになる。

まずは、下のコード（アルファベットの羅列：意味は今わからなくても大丈夫）をテキストエディタでタイプしよう。 タイプが終わったら、「**ファイル名を “test.html” として保存**」する。「**拡張子に “.html”**」を付けることを忘れないようにしよう。

初めての Web ページ（test.html）

```
<html>
<head>
<title>とにかく作ろう</title>
</head>
<body>
<p>初めての Web ページ</p>
</body>
</html>
```

小難しい説明はなし、なのだが、ついでなので、言葉をひとつ覚えておこう。 記号<と>で囲まれた<html>や<head>を「**タグ**」（荷札）と呼ぶ。

[*2] メタ言語とは、他の言語を構築するための言語のことである。

また、</で始まるタグを「**終了タグ**」（例えば、</html>、</body>など）、 /のないタグを「**開始タグ**」（例えば、<html>、<body>など）と呼ぶ。

さあ、タイプできたろうか？。タイプが完了すれば、 それが Web ページとなる「**HTML ファイル**」と呼ばれるものの完成だ。

演習 1

各自の環境で実際に “test.html” を作成しよう。 ここでは、各自の「**エディタ**」の使い方をマスターすることを目指そう。

Windows を使ってる人なら「**メモ帳**」、Mac を使ってる人なら「**テキストエディット**」を使いこなそう。 ただ、どちらも “おまけ” のソフトだから、機能的には不足するかもしれない。 そんなときは、フリーのエディタもたくさんあるから、インターネットに接続できる人は、 自分のお気に入りのエディタを見つけるのもよいかもしれない。

1.3　できたら確認しよう

このページでは、“とにかく作ろう” で作成した HTML ファイル “test.html” が、 きちんと表示されるか、確認してみよう。

確認には、まず Web ブラウザを使うんだ。 下のイメージは “Firefox”（図 1.1）というブラウザだが、Windows 標準の “Internet Explorer”[*3]でもいいし、Mac 標準の “Safari” でもいい。

図 1.1　Firefox の画面

つぎに示す方法は、“Firefox” での方法だが、他の Web ブラウザもほぼ同じ操作で確認できるので、 各自の環境で試してみるといいだろう。

[*3] ただし、これから先のことを考えると（Web 標準というのが出て来る） 現在の “Internet Explorer 6” はやめておいた方が無難かもしれない。

1. メニューから「**ファイルーファイルを開く**」を選択する。(図 1.2)

図 1.2　Firefox（メニューバー）

2.「**ファイル選択ダイアログ**」(図 1.3) が表示されるので、 作成した HTML ファイル “test.html” が存在する「**フォルダを選び**」、 その中から「**ファイル “test.html” を選択した後、開くボタンをクリックする。**」

図 1.3　Firefox（ファイルを開くダイアログ）

3. ブラウザに、作成した HTML ファイル “test.html” によって文字列が表示される。(図 1.4)

図 1.4　Firefox による表示の確認

このように画面にうまく表示されれば、一応出来上がりだ。 もし、うまく表示されない場合は、もう一度 HTML ファイルの内容に間違いが無いかどうか確かめてみよう。

演習 2

各自のブラウザで実際に “test.html” を表示してみよう。 ここでは、各自の「**ブラウザ**」の使い方をマスターすることを目指そう。

もし、複数のブラウザがインストールされているなら、それぞれの表示のされ方の違いを見るのもよい。

1.4 少し勉強しよう

1.4.1 タグで囲まれた全体を「要素」と言う。

タグで囲まれた部分を「**要素**」と呼ぶ。例えば、

```
<title>とにかく作ろう</title>
```

の<title>は開始タグ、</title>は終了タグ、となり全体を「**title 要素**」と呼ぶ。

1.4.2 要素は「入れ子構造（階層構造）」になる。

HTML ファイル “test.html” を見ると、最も大きな要素は開始タグ<html>、 終了タグ</html>で囲まれた「**html 要素**」だ。

そして、その中は「**head 要素**」と「**body 要素**」から構成されている。 さらに、head 要素の内容は title 要素、body 要素の内容は「**p 要素**」となっている。(図 1.5)

図 1.5 要素の階層構造図

仮に ABCD 要素と EFGH 要素というものがあったとき、

```
<ABCD>*****<EFGH>*****</EFGH>*****</ABCD>  正しい
```

という記述ならば、EFGH 要素が ABCD 要素の入れ子となっているので正しいが、

```
<ABCD>*****<EFGH>*****</ABCD>*****</EFGH>  誤り
```

という記述の場合は、EFGH 要素が ABCD 要素の入れ子になっていないので誤った記述となる

このように要素が「**完全な入れ子**」になった状態で記述する必要があるが、これを「**整形式**」と呼ぶ。 つま

り、HTML は整形式で記述しなければならないということだ。[*4]

1.4.3　html 要素を「ルート（根）要素」と言う。

最も大きな要素である html 要素のことを「**ルート要素**」と呼び、 その html 要素のことを「**HTML 文書（ドキュメント）**」とも呼ぶ。

1.4.4　html 要素の中には「head 要素と body 要素をそれぞれこの順序で一つ記述」しなければならない。

各要素には、内側に記述できる要素に制限がある。また、必ず記述する必要がある要素、 必要に応じて省略できる要素もある。これらのルールを、ここでは「**文法**」と呼ぶことにする。

1.4.5　head 要素の中には「title 要素を必ず一つ記述」しなければならない。

title 要素は文書全体に付ける文書名のようなもので、ブラウザのウィンドウタイトルとして表示されることが多い。

1.4.6　文法は「DTD（Document Type Definition：文書型定義）」によって規定されている。

文法は、DTD と呼ばれるファイルで規定されている。 この DTD にもいくつかの種類があり、それによって HTML の種類が異なる。

また、SGML のルールによって記述された DTD によるものを HTML、XML のルールによって記述された DTD によるものを XHTML と呼ぶ。 本ページでは、XHTML を原則として表現するが、これを HTML[*5]に直すのは非常に簡単である。

整形式で、なおかつ、文法[*6]に従って正しく記述されたものを「**検証済み文書**」あるいは「**妥当な文書**」と呼ぶ。

1.4.7　HTML、XHTML それぞれ３種類の DTD（図 1.6）が存在する。

HTML、XHTML それぞれに「**Strict**」、「**Transitional**」、「**Frameset**」と呼ばれる 3 種類の DTD[*7]が存在する。

Strict　もっとも厳格な文法を規定したもの。非推奨要素を若干含む。

Transitional　柔軟な文法を規定したもの。非推奨要素を数多く含む。

Frameset　Transitional にフレーム関係の要素を付加したもの。

[*4] 厳密に言うと、本書で扱う XHTML 1.0 の場合、整形式でなければならない。

[*5] HTML 4.01 のこと。本書で “HTML” という場合、多くは XHTML のことである。

[*6] 正確には DTD に適合しているということ。

[*7] 本ページでは Strict を中心として記述する。

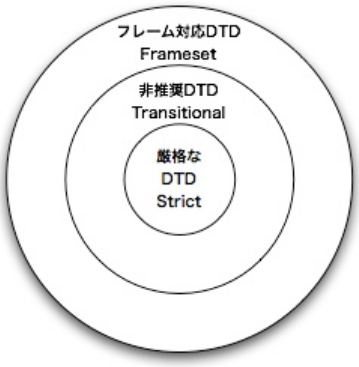

図 1.6 DTD の関係図

1.5 DTD を意識しよう

まずは、XHTML の DTD を意識してみよう。XHTML にもいくつかのバージョンが存在するが、 本書で使用するのは、「**XHTML 1.0**」と呼ばれるものだ。

先の “test.html” を “XHTML 1.0 Strict DTD” に対応させるためには、 1 行目から 4 行目までを追加し、html 開始タグを修正する。

XHTML 1.0 Strict に適合した test.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ja" lang="ja">
<head>
<title>とにかく作ろう</title>
</head>
<body>
<p>初めての Web ページ</p>
</body>
</html>
```

1.5.1 XML 宣言

XHTML は、XML をベースとした HTML である。 そのため、1 行目に “これは XML 文書ですよ” という宣言を記述する。[*8]

```
<?xml version="1.0" encoding="UTF-8"?>
```

このとき注意することは ecoding=”**UTF-8**”の “UTF-8” の部分である。 ここには、「**エンコーディング形式**」を指定する。 エンコーディングは、テキストエディタで作成したときの保存文字コードのことだ。 つまり、テキストエディタで HTML ファイルを作成する場合は、 どのような「**文字コード**」を使用しているのか、

[*8] XML 宣言は、必須ではないが、記述した方がよい。詳しくは、他の書籍、Web ページを参考にしてほしい。

常に意識する必要がある。[*9]以下に、代表的な文字コードと、それを表すエンコーディング文字列を上げておく。

■UTF-8（Unicode）コード

1 文字を可変長のバイト列に変換する方式。「**UTF-8**」あるいは「**utf-8**」と記述する。

■UTF-16（Unicode）コード

2 バイト文字の範囲はそのまま表現し、2 バイトでは足りない部分は 4 バイトで表現する方式。「**UTF-16**」あるいは「**utf-16**」と記述する。

■JIS コード

JIS 規格によって規定されている文字コード方式。「**ISO-2022-JP**」あるいは「**iso-2022-jp**」と記述する。

■Shift（シフト）JIS コード

Microsoft 社によって策定された文字コード方式。「SHIFT_JIS」あるいは「Shift_JIS」、「shift_jis」と記述する。

■日本語 EUC コード

日本語 UNIX システム諮問委員会の提案に基づいて 1985 年に AT&T 社が定めた方式。「**EUC-JP**」あるいは「**euc-jp**」と記述する。

1.5.2　DOCTYPE 宣言

修正した “test.html” の 2 行目から 4 行目が使用する文法、すなわち DTD の指定をする部分になる。

```
<!DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

この部分を「**DOCTYPE 宣言**」と言い、ここで使用する DTD を宣言する。 DTD が 3 種類あることは前に言ったが、それぞれ記述する形式は決まっている。

■Strict DTD

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

■Transitional DTD

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

■Frameset DTD

[*9] 勘違いしてはいけないことは、文字コードの記述は保存形式を明示するだけであるということだ。 文字コードの変換は行わないので、テキストエディタでは、記述した文字コード形式で確実に保存するようにしよう。

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

1.5.3 属性の指定

要素を表現するためにタグを使用するわけだが、その要素に対する補助情報を与えるものが「**属性**」である。属性は、開始タグにのみ記述できる。また、記述できる属性の種類、必ず書かなければならない属性、省略できる属性などの区別もすべて DTD に基づいている。

タグへの属性の書き方は次のようになる。

```
<要素名 属性１ 属性２・・・>
```

このとき要素名と属性の間、属性を複数記述する場合は、その間を「**１つ以上の空白で区切る**」。

次に属性そのものは次のような書き方になる。

```
属性名="属性値"
```

注意することは、「**属性名と属性値は等号（=）で結び、属性値は必ず引用符（”）で囲まれる**」ことだ。

1.5.4 html 要素の属性

html 要素にも複数の属性があるが、“test.html” では、３種類の属性を使用している。

■xmlns=”http://www.w3.org/1999/xhtml”

xmlns 属性は、名前空間と呼ばれるもので、XML 固有のものである。XHTML の場合には記述するが、省略も可能だ。なお、HTML 4.01 の場合には、記述しない。

■xml:lang=”ja”

xml:lang 属性は、言語コードを指定する。我々は日本語を使用するので、“ja” を指定する。これも XML 固有のもので、HTML 4.01 の場合には記述しない。

■lang=”ja”

この属性も xml:lang 属性と同じ働きをする。通常、XHTML の場合には記述しなくても良いのだが、現状では、HTML 4.01 との互換性を考慮して記述する。

演習 3

XHTML 1.0 の Web ページを作成しよう。つまり、

- テキストエディタで “test.html” を作成すること
- ブラウザで表示ができるかどうか確認すること

この 2 点について、自分で実際に試してみよう。

1.6　XHTML の文法チェックをしよう

DTD を意識した HTML ファイル “test.html” を作成したが、本当に DTD に適合しているかどうか、どうやって調べるか？

まず、注意することは、「**チェックにブラウザが使えない**」ことだ。ブラウザは、Web ページの閲覧を行うことを最大の目的として作成されたソフトウェアであり、DTD への適合はチェックしていない。つまり、「**ブラウザで表示されたとしても、それは、DTD 適合をまったく保証していない**」、ということだ。

では、どうするか？答えは簡単だ。DTD 適合をチェックしてくれるツールを使えばいい。これにはいろいろなものがあるが、もっともよく使われ、標準的なツールである「**W3C**」[*10]の妥当性チェッカーを使ってみよう。それでは、手順を見てみよう。[*11]

1. ブラウザのアドレスに URL“http://validator.w3.org/” を入力し、「**チェック用ページを開く**」。
2. テキストエディタで “test.html” を呼び出し、「**すべてを選択し、コピーする**」。(図 1.7)

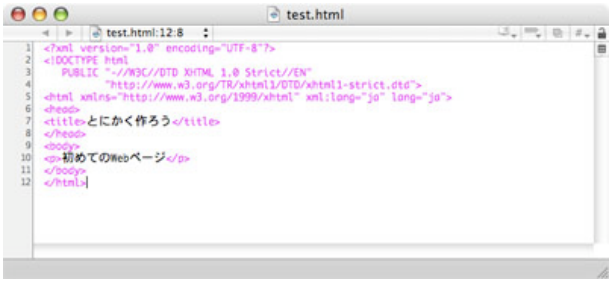

図 1.7　エディタでコピー

3. W3C Validator のページにある「**Validate by Direct Input**」タブをクリックし、テキストエリアに「**コピーしたテキストをペーストする**」。その後、テキストエリア下にある「**Check ボタンをクリック**」する。(図 1.8)
4. DTD に適合していれば、次のような画面（図 1.9）が表示される。
   もし、適合していない場合は、次のような画面（図 1.10）が表示され、エラー箇所も合わせて表示される。

さあ、実際に作成した “test.html” をチェックしてみよう。もし、不適合の場合は、どこかに間違いがあるので修正しよう。[*12]

> 演習 4
>
> 演習 3 で作成した “test.html” が本当に XHTML 1.0 Strict DTD に適合しているかどうか、確認しよう。

[*10] W3C“World Wide Web Consortium” は、Web に関する技術仕様を策定している団体で、XML、HTML、XHTML などの仕様も勧告している。

[*11] このツールは、2007-12-06 現在のものである。しばしばバージョンアップされるので、実際の画面とは異なる場合がある。

[*12] 最初は、多くの間違いがあり、途方に暮れるが、まずは、最初の方から順番に 1 つずつ修正しよう。間違いの多さに圧倒される必要はない。1 箇所の間違いで、20 も 30 もエラーが出ることは日常茶飯事だから。そして、この修正がホームページを作成する一番の訓練になる。頑張ってほしい。

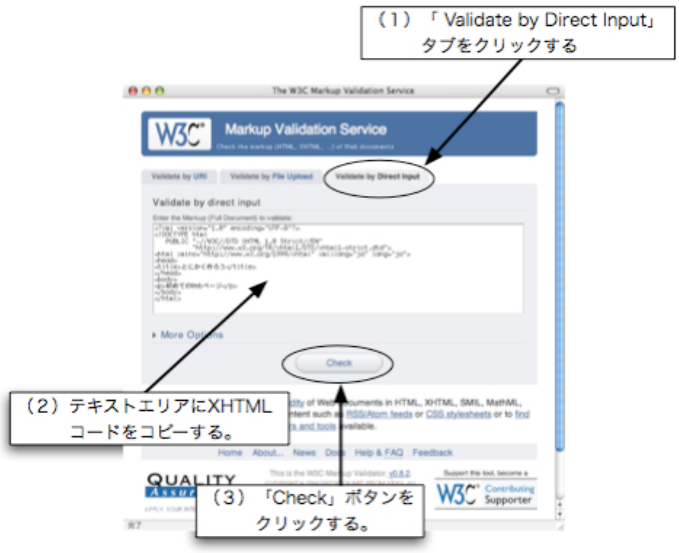

図 1.8　W3C Validator page

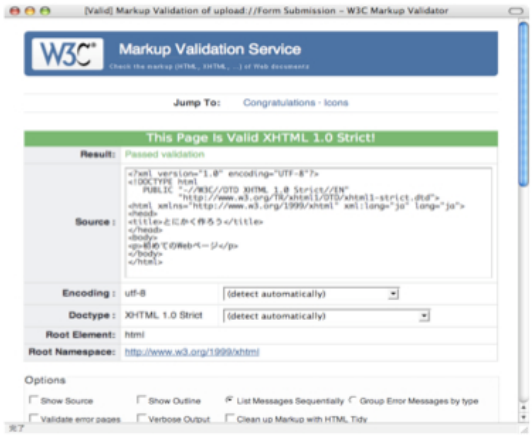

図 1.9　DTD 適合画面

図 1.10　DTD 不適合画面

1.7　補助情報を付加しよう

せっかく Web ページを作成しても、そのページをできる限り多くの人に見てもらい、 また、できる限り正しく見てもらうことが重要だ。

特にサーチエンジン[*13]の上位に自分の Web ページが表示されるようにする技術を「**サーチエンジン最適化（SEO：Search Engine Optimization）**」といい、 非常に重要視されている。

これを一から学ぶことも大事だが、まずは、どうすればできるのか？その具体的方法を見よう。 これらのことを指定するためには「**meta 要素**」を使用する。

1.7.1　meta 要素

ページ作成者、エンコーディング方式、検索用キーワードなど補助情報のことを「**メタ情報**」と呼ぶ。 このメタ情報は、meta 要素によって記述される。具体的には meta タグを使って記述される。 この meta タグには大きく分けて 2 種類の書き方がある。

```
書き方１：<meta http-equiv="属性値" content="属性値" />
書き方２：<meta name="属性値" content="属性値" />
```

また、この meta 要素は、これまでの要素（html 要素、head 要素など）と異なり、 開始、終了タグが他の要素を囲む形式ではなく、タグ単独で要素を表す。 すなわち、入れ子となる内側要素を持たない要素である。このような要素を「**空要素**」と呼ぶ。空要素は、以下の形式で記述する。[*14]

```
<要素名　属性 />
```

1.7.2　文字エンコーディングを指定する

XHTML の場合、第 1 行目の XML 宣言で文字エンコーディングを指定したが、 ブラウザによってはこれを無視するものもある。 そこで HTML 文書内に meta 要素を使って、文字エンコーディングを明示的に指定する必要がある。 具体的な書き方としては meta 要素の “書き方 1 ” を使用し、「**http-equiv 属性の属性値に “Content-Type”**」を指定し、「**content 属性の属性値に “ファイル形式” 及び “エンコーディング方式”**」を指定する。

■完全な XHTML の場合

XHTML を完全な XML 文書として扱う場合、本来、拡張子を “.xhtml” とし、 以下のような meta 要素で文字エンコーディングを指定する。

```
<meta http-equiv="Content-Type"
      content="application/xhtml+xml; charset=UTF-8" />
```

■互換性を考慮した XHTML の場合

XHTML に対応していないブラウザもあるので、XHTML を HTML と互換が保てる文書として扱う場

[*13] google とか Yahoo など

[*14] タグの終了を表す “/>” の前に空白を記述しているが、 これは XHTML に対応していないブラウザへの互換を保つための工夫である。 XHTML の文法的には空白を記述する必要はない。

合、 拡張子を “.html” とし、以下のような meta 要素で文字エンコーディングを指定する。[*15]

```
<meta http-equiv="Content-Type"
      content="text/html; charset=UTF-8" />
```

1.7.3 ページ概要を記述する

ページの概要を記述する。この場合、「**name 属性の属性値に “description”**」を指定し、「**content 属性の属性値としてページ概要**」を記述する。また、通常は、日本語で記述するので、「**言語コードを意味する xml:lang 属性、lang 属性**」を指定する。

```
<meta name="description"
      xml:lang="ja" lang="ja"
      content="ここにページの概要を記述する" />
```

1.7.4 検索用キーワードを記述する

ロボット型検索エンジンに探してもらえるように検索用キーワードを記述する。 この場合は、「**name 属性に “keywords”**」、「**content 属性に複数の検索用キーワードをカンマで区切って記述**」する。 また、日本語を使用している場合は、言語コードも合わせて指定する。

```
<meta name="keywords"
      xml:lang="ja" lang="ja"
      content="キーワード１, キーワード２, ・・・, キーワード n" />
```

1.7.5 作成者を記述する

Web ページの作成者を記述します。この場合は、「**name 属性に “author”**」、「**content 属性に作成者名を記述**」する。 また、日本語を使用している場合は、言語コードも合わせて指定する。

```
<meta name="author"
      xml:lang="ja" lang="ja"
      content="作成者名" />
```

1.7.6 著作権情報を記述する

Web ページの著作権情報を記述します。この場合は、「**name 属性に “copyright”**」、「**content 属性に著作権情報を記述**」する。 また、日本語を使用している場合は、言語コードも合わせて指定する。

```
<meta name="copyright"
      xml:lang="ja" lang="ja"
      content="著作権情報" />
```

以上のようなメタ情報を付加したサンプルのページ “test.html” のコードを下に掲載しておく。

[*15] 本書では、この互換性を考慮した書き方を使用する。

meta 要素を追加した test.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ja" lang="ja">
<head>
<meta http-equiv="Content-Type"
      content="text/html; charset=UTF-8" />
<title>とにかく作ろう</title>
<meta name="description"
      xml:lang="ja" lang="ja"
      content="御隠居のホームページ作成法で使用するサンプルページです。" />
<meta name="keywords"
      xml:lang="ja" lang="ja"
      content="御隠居, ホームページ,Web ページ,HTML,XHTML,CSS,W3C" />
<meta name="author"
      xml:lang="ja" lang="ja"
      content="御隠居" />
<meta name="copyright"
      xml:lang="ja" lang="ja"
      content="Copyright (C) 御隠居" />
</head>
<body>
<p>初めての Web ページ</p>
</body>
</html>
```

演習 5

演習 3 で作成した “test.html” に、メタ情報として文字エンコーディングのみ付加しよう。 メタ情報を追加したら、ブラウザによる表示確認および DTD 適合検査を忘れずに行うこと。

1.8　XHTML を HTML にしてみよう

これまで XHTML を中心に説明してきたが、無料ホームページスペースなどでは、 広告表示の部分が勝手に追加されて表示されてしまうことが多い。 このとき追加部分が XHTML の記述になっていないものが大半である。 そこで、ここでは XHTML を HTML に修正する方法を示す。[*16]

HTML にもいろいろなバージョンが存在するが、 ここでは「**HTML 4.01**」の最も厳格な DTD である「**Strict**」に修正することとする。 なお、Strict で DTD のチェックを行い、 実際にアップロードして表示したページをもう一度チェックし、 そこで不具合が生じるようであれば、 DOCTYPE 宣言を「**Transitional**」に変更するだけでほとんどの場合、 対応できるので覚えておこう。

[*16] これはあくまでも後方互換（過去に対する互換)、 つまり、古い形式で記述されたページへの書き直しということには、注意してほしい。

1.8.1 変更する XHTML ファイルのサンプル

XHTML 1.0 Strict で記述された test.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ja" lang="ja">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>とにかく作ろう</title>
<meta name="description" xml:lang="ja" lang="ja"
      content="御隠居のホームページ作成法で使用するサンプルページです。" />
<meta name="keywords" xml:lang="ja" lang="ja"
      content="御隠居, ホームページ,Web ページ,HTML,XHTML,CSS,W3C" />
<meta name="author" xml:lang="ja" lang="ja" content="御隠居" />
<meta name="copyright" xml:lang="ja" lang="ja"
      content="Copyright (C) 御隠居" />
</head>
<body>
<p>初めての Web ページ</p>
</body>
</html>
```

1.8.2 コメント

XHTML、HTML 共通で「**コメント**」を記述することができる。 コメントは、注釈のことで、単なるただし書きでありブラウザ上には表示されない。 コメントは、以下の書式で記述する。

```
<!-- ここにコメントを記述する -->
```

このとき注意することは、コメント部分に “--” を記述しないことだ。 要は、「**2つ以上の連続したマイナス記号を記述しない**」ように注意しよう。

1.8.3 XML 宣言は不要

HTML は、当然ながら XML 文書ではないので、XML 宣言は不要となる。 また、DTD も HTML のものを使用するので、 ここでは XML 宣言と DOCTYPE 宣言をコメントアウト[*17]しておこう。

```
<!-- ?xml version="1.0" encoding="UTF-8"? -->
<!-- DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" -->
```

[*17] コメントにすることによって、その記述を無効化し、かつ、変更前のコードも残すことができる。

1.8.4 HTML 用の DTD を追加する

HTML 4.01 にも XHTML と同様に 3 種類の DOCTYPE 宣言がある。

■Strict DTD

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
```

■Transitional DTD

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
            "http://www.w3.org/TR/html4/loose.dtd">
```

■Frameset DTD

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
            "http://www.w3.org/TR/html4/frameset.dtd">
```

ここでは、“Strict” を使用するので、以下のように新しい “DOCTYPE 宣言” を追加する。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
<!--      上の 2 行が、新しく追加された DOCTYPE 宣言 -->

<!-- ?xml version="1.0" encoding="UTF-8"? -->
<!-- DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" -->
```

1.8.5 XML 固有の属性を削除する

XML 固有の属性を削除する。対象となるものは、「**xmlns 属性**」と「**xml:lang 属性**」だ。 サンプルの関連する部分をコメントアウトして、新しく書き直してみると以下のようになる。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
<!-- ?xml version="1.0" encoding="UTF-8"? -->
<!-- DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" -->

<!-- 新しく書き直された html 開始タグ -->
<html lang="ja">
<!-- html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ja" lang="ja" -->

<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>とにかく作ろう</title>
```

```
<!-- 新しく書き直された meta タグ -->
<meta name="description" lang="ja"
      content="御隠居のホームページ作成法で使用するサンプルページです。" />
<meta name="keywords" lang="ja"
      content="御隠居, ホームページ,Web ページ,HTML,XHTML,CSS,W3C" />
<meta name="author" lang="ja" content="御隠居" />
<meta name="copyright" lang="ja"
      content="Copyright (C) 御隠居" />
<!-- meta name="description" xml:lang="ja" lang="ja"
      content="御隠居のホームページ作成法で使用するサンプルページです。" / -->
<!-- meta name="keywords" xml:lang="ja" lang="ja"
      content="御隠居, ホームページ,Web ページ,HTML,XHTML,CSS,W3C" / -->
<!-- meta name="author" xml:lang="ja" lang="ja" content="御隠居" / -->
<!-- meta name="copyright" xml:lang="ja" lang="ja"
      content="Copyright (C) 御隠居" / -->

</head>
<body>
<p>初めての Web ページ</p>
</body>
</html>
```

1.8.6 空要素のタグ修正

XML では空要素のタグの終わりが “/>” で終わるが、HTML では、「**単に > のみで終わる**」。つまり XML 的な解釈をすれば、 開始タグのみを記述していることになる。

これですべての修正が終了し、“HTML4.01 Strict” に基づいた HTML ファイルになった。

HTML 4.01 Strict に適合した test.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
<!-- ?xml version="1.0" encoding="UTF-8"? -->
<!-- DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" -->
<html lang="ja">
<!-- html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ja" lang="ja" -->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" >
<title>とにかく作ろう</title>
<meta name="description" lang="ja"
      content="御隠居のホームページ作成法で使用するサンプルページです。" >
<meta name="keywords" lang="ja"
      content="御隠居, ホームページ,Web ページ,HTML,XHTML,CSS,W3C" >
<meta name="author" lang="ja" content="御隠居" >
<meta name="copyright" lang="ja"
      content="Copyright (C) 御隠居" >
<!-- meta name="description" xml:lang="ja" lang="ja"
      content="御隠居のホームページ作成法で使用するサンプルページです。" / -->
<!-- meta name="keywords" xml:lang="ja" lang="ja"
```

```
      content="御隠居, ホームページ,Web ページ,HTML,XHTML,CSS,W3C" / -->
<!-- meta name="author" xml:lang="ja" lang="ja" content="御隠居" / -->
<!-- meta name="copyright" xml:lang="ja" lang="ja"
      content="Copyright (C) 御隠居" / -->
</head>
<body>
<p>初めての Web ページ</p>
</body>
</html>
```

演習 6

演習 7 で作成した “test.html” を “HTML 4.01 Strict DTD” に変更しよう。 ただし、ファイル名は、“test40.html” とする。

作成ができたならば、ブラウザによる表示確認および DTD 適合検査を忘れずに行うこと。 また、コメントアウト部分は、削除してもよい。

1.9 段落と見出し

ここでは、使う頻度の多い「**段落（p 要素）**」と「**見出し（h1 要素、h2 要素、h3 要素、h4 要素、h5 要素、h6 要素）**」を中心に説明する。

1.9.1 body 要素

これまでサンプルをブラウザで何度か表示させてきた。 実際にブラウザに表示される部分は、 すべて開始タグ “<body>” と終了タグ “</body>” の間に記述されたものだ。

つまり、「**Web ページとして表示させたい部分は、body 要素として記述する**」ことになる。 body 要素については、これくらいの理解で、今は先に進もう。

1.9.2 段落（p 要素）

テキストを表示させる場合、通常はひとつのかたまりの文章を「**段落**」として扱う。 この段落を表現する要素が「**p 要素**」だ。まず、その書式は、次のようになる。

```
<p>・・・テキスト・・・</p>
```

段落で注意しなければならないことは改行位置だ。改行位置を強制的に制御することは可能である。 ただし、本当に制御する必要があるかどうかを再検討しないといけない。 つまり、作成した Web ページを見ているユーザー（読者）の環境を統一することは不可能なため、 ユーザーによって、表示幅が異なり、それによって段落の改行位置も変化するからだ。

すなわち、「**改行位置を気にするのではなく、意味的にひとつの段落かどうか、 それによってタグ付けをした方がよい**」。 つまり、この段階では見栄えは気にしない方がよい。

1.9.3 見出し要素

見出しを表現するためのものが見出し要素だが、これには 6 種類の要素が用意されている。 各見出し要素の書式は、次のようになる。

```
<h1>ここに見出しを書く</h1>
<h2>ここに見出しを書く</h2>
<h3>ここに見出しを書く</h3>
<h4>ここに見出しを書く</h4>
<h5>ここに見出しを書く</h5>
<h6>ここに見出しを書く</h6>
```

各見出し要素の番号の違いだが、 見出しにも大見出し、中見出し、小見出しのように階層がある訳で、 それを番号で区別するようになっている。 つまり、h1 から順番に h2、h3・・・と階層が下がって行く訳だ。 例えば、h1 が大見出し、h2 が中見出し、h3 が小見出しという具合に使って行く。

見出し要素で注意する点は、“h1 が大見出し、h3 が中見出し、h5 が小見出し” とか、“h3 が大見出し、h4 が中見出し、h5 が小見出し” のような使い方は避けるということだ。「**見出しは、h1 から順番に間を空けずに使おう。**」

1.9.4 HTML が表現するもの

実際に見出し要素などを指定して、ブラウザで表示すると文字の大きさが変化しているものが大部分だろう。しかし、これは本来の “HTML” の機能ではない。

文字の大きさや色などのいわゆる見栄えに関するものは、 すべて文書の「**物理構造（スタイル）**」と呼ばれるもので、HTML の範疇ではない。 HTML は文書の「**論理構造**」、つまり、ここは見出し、 ここは段落、ここは箇条書きといった「**文書の意味的構造を記述する**」ものだ。

このことはとても大切だ。「**Web ページでの見栄えはとても重要だが、それを担当するものは、 スタイルシートと呼ばれるもので記述する。HTML は、とにかく見栄えを気にすることなく、 ひたすら文書の意味を考えることに留意しよう。**」

演習 7

つぎの文章を Web ページとして作成しなさい。ファイル名は、“sample09.html” とする。 なお、記号 “<” と “>” は、全角文字を用いることとする。

■段落と見出し
ここでは、使う頻度の多い段落（p 要素）と見出し（h1 要素、h2 要素、h3 要素、h4 要素、h5 要素、h6 要素）を中心に説明する。

■ body 要素
これまでサンプルをブラウザで何度か表示させてきた。 実際にブラウザに表示される部分は、 すべて開始タグ＜ body ＞と終了タグ＜/body ＞の間に記述されたものだ。
つまり、Web ページとして表示させたい部分は、body 要素として記述することになる。 body 要素については、これくらいの理解で、今は先に進もう。

1.10　改行と段落

段落を表現するためには、p 要素を使用する。これは前に説明した。 また、見栄えは XHTML（HTML）の担当ではないこともやった。

しかし、見栄えとは関係なく、段落内で改行位置を指定したい場合もある。 そのときは「**br 要素**」を使用して文書構造としての改行を指定することができる。 まずは、サンプルで段落（p 要素）の性質を見てみよう。

```
<p>
これは段落のテストです。
あいうえお
かきくけこ
さしすせそ
たちつてと
なにぬねの
</p>
```

このサンプルを実際にブラウザで表示した場合の 2 通りの結果を見て欲しい。 最初の画面（図 1.11）は、ウィンドウの幅が広く、 2 番目の画面（図 1.12）は、ウィンドウの幅が狭い。

図 1.11　p 要素の表示サンプル画面 1

図 1.12　p 要素の表示サンプル画面 2

サンプルコードと 2 種類の表示結果から、次のことを理解して欲しい。

- XHTML（HTML）ファイルの内容で改行してもブラウザ上では改行されない。
- ブラウザの幅によって表示される長さが異なる。
- そのため幅が短い場合、自動的に改行される。
- ブラウザの幅を XHTML（HTML）で制御することはできない。

そこで意識的に改行位置を指定する場合、br 要素を使用する。 ただし、「**ブラウザの幅を XHTML（HTML）で制御することはできない。**」[*18]は同じなので、 見栄えとしての改行位置は思い通りにならないこ

[*18] 厳密に言えば、制御不可能ではないが、不必要にユーザ環境を制御すべきではない。

とがあるので注意がいる。

br 要素　br 要素は、改行を意味する空要素で、書き方は以下のようになる。

```
<br />
```

では、サンプルを変更して、ブラウザで表示した結果（図 1.13）を示しておく。

```
<p>
これは段落のテストです。<br />
あいうえお<br />
かきくけこ<br />
さしすせそ<br />
たちつてと<br />
なにぬねの
</p>
```

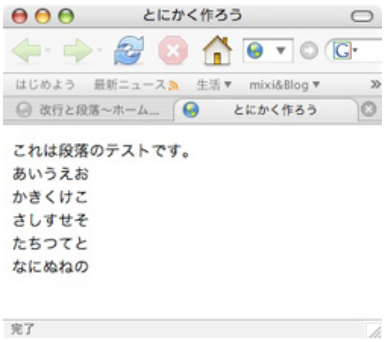

図 1.13　br 要素による改行のサンプル表示

演習 8

サンプル（図 1.13）を実際に表示する Web ページを作成しなさい。 ただし、ファイル名は “sample10.html” とする。

1.11　字句要素を見てみよう

文字列の強調や用語、上付き文字、下付き文字などの構造を表現するものが「**字句要素**」だ。 これには多くの要素が定義されている。それぞれの意味と要素の関係を見てみよう。[*19]

1.11.1　em 要素

em 要素は、「**強調（emphasis）**」を意味する。

```
<em> 強調する部分 </em>
```

[*19] 字句要素の構造的用途、つまり文書構造の意味合い（強調とか略語とか）に注意して欲しい。

1.11.2　strong 要素

strong 要素は、「**最強調（strong emphasis）**」を意味する。

```
<strong> 最強調する部分 </strong>
```

1.11.3　dfn 要素

dfn 要素は、「**定義用語（definitional）**」を意味する。

```
<dfn> 定義用語 </dfn>
```

1.11.4　code 要素

code 要素は、「**プログラムコード（program code）**」を意味する。

```
<code> プログラムコード </code>
```

1.11.5　samp 要素

samp 要素は、「**サンプル（sample）**」を意味する。通常は、コンピュータの出力例などに使用する。

```
<samp> 出力サンプル </samp>
```

1.11.6　kbd 要素

kbd 要素は、「**ユーザ入力文字列（something user would type）**」を意味する。

```
<kbd> ユーザ入力文字列 </kbd>
```

1.11.7　var 要素

var 要素は、「**変数・引数（variable）**」を意味する。

```
<var> 変数・引数 </var>
```

1.11.8　cite 要素

cite 要素は、「**出典先・参照先（citation）**」を意味する。

```
<cite> 出典先・参照先 </cite>
```

1.11.9 abbr 要素

abbr 要素は、「**略語（abbreviation）**」を意味する。例えば “Monday” を “Mon” と表現する場合。

```
<abbr> 略語 </abbr>
```

1.11.10 acronym 要素

acronym 要素は、「**頭文字略語（acronym）**」を意味する。例えば “Hyper Text Markup Language” を “HTML” と表現する場合。

```
<acronym> 頭文字略語 </acronym>
```

1.11.11 q 要素

q 要素は、「**引用句（quote）**」を意味する。

```
<q> 引用句 </q>
```

1.11.12 sub 要素

sub 要素は、「**下付き文字（subscript）**」を意味する。

```
<sub> 下付き文字 </sub>
```

1.11.13 sup 要素

sup 要素は、「**上付き文字（superscript）**」を意味する。

```
<sup> 上付き文字 </sup>
```

1.11.14 字句要素の使用

このように字句要素には、多くの要素が定義されているが、実際には、すべての要素を使用することは少ない。 主に使用するものは、strong 要素（２種類の強調を使用するときは em 要素を併用)、 sub 要素、sup 要素ぐらいだ。しかし、文書構造を正確に表現したいならば、 字句の文書構造を表現するために、なるべく字句要素を指定することが望ましい。

ここで一つ注意。書籍などでは、用語を太字で表現したり、重要箇所を下線で表現することが多いが、「**XHTML（HTML）では、太字とか下線を表現するのではなく、用語は定義された用語であるとか、強調したいとか、 そういった構造的な意味合いを考えてタグを指定する。**」つまり、見た目から要素を判別してはいけない。これは特に注意を必要とする。

では、以下にサンプルコードとブラウザでの表示例（図 1.14）を示しておく。

```
<p>
<em>強調です（em 要素）</em><br />
<strong>最強調です（strong 要素）</strong><br />
<dfn>定義用語です（dfn 要素）</dfn><br />
<code>プログラムコードです（code 要素）</code><br />
<samp>出力サンプルです（samp 要素）</samp><br />
<kbd>ユーザ入力文字列です（kbd 要素）</kbd><br />
<var>変数・引数です（var 要素）</var><br />
<cite>出典先・参照先です（cite 要素）</cite><br />
<abbr>略語です（abbr 要素）</abbr><br />
<acronym>頭文字略語です（acronym 要素）</acronym><br />
<q>引用句（q 要素）</q><br />
<sub>下付き文字です（sub 要素）</sub><br />
<sup>上付き文字です（sup 要素）</sup>
</p>
```

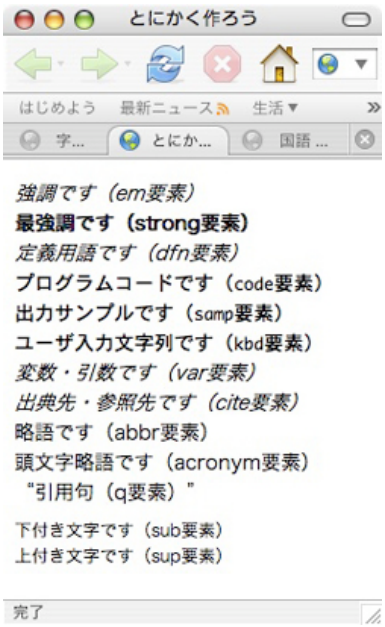

図 1.14　字句要素のサンプル表示

演習 9

サンプル（図 1.14）を実際に表示する Web ページを作成しなさい。 ただし、ファイル名は “sample11.html” とする。

また、各要素がブラウザで実際どのように表示されるか観察しよう。

1.12 ブロックとインライン

これまでいくつかの要素を見て来たが、Web ページとしてブラウザ上に表示される部分は、 body 要素の内容として記述した。この body 要素内にさまざまな要素を指定するが、 これまで出てきた要素は大きく分けて 2 種類に分類される。

ブロックレベル要素 ブロックレベル要素は、矩形（四角）として表現される文章を意味する要素群である。 これまでの要素では、見出し要素、段落要素が、このブロックレベル要素に分類される。 通常、ブロックレベル要素の前後は自動的に改行される。

インライン要素 インライン要素は、文章内の一部分の構造を表現する要素である。 これまでの要素では、改行要素、字句要素が、このインライン要素に分類される。

この要素群の分類は、とても重要である。なぜならば、「**インライン要素をブロックレベル要素の中に記述することは可能である。 しかし、その逆はできない。**」

```
<p><strong>正しい記述</strong></p>
<strong><p>誤った記述</p></strong>
```

このサンプルの場合、1 行目は正しいが、2 行目は誤った記述となっている。 しかし、ブラウザは、どちらも問題なく表示してしまう。「**多くのブラウザは、このような誤りをチェックすることができない。**」ブラウザは、ページ閲覧のために表示することが目的のソフトウェアであり、文法チェックをすることが目的ではないからだ。 この点は、十分に注意して欲しい。詳しくは、「XHTML の文法チェックをしよう」を参照して欲しい。

さて、前にも簡単な部分は説明したが、XHTML（HTML）の文法は、DTD によって規定されている。 言い換えれば、DTD を読むことが XHTML（HTML）の文法を理解することになる。

さて、DTD を読む前に、もう少し、例を見てみよう。まず、ブロックレベル要素同士の場合、その内容にブロックレベルを持つことはできない。[*20]つまり、段落要素の中に見出し要素を記述したり、その逆はできない。

```
<p>段落１－１<h1>見出し</h1>段落１－２</p>　誤った記述
```

一部の例外はあるが、インライン要素の中にインライン要素を記述することはできる。

```
<code><strong>字句</strong></code>　正しい記述
<strong><code>字句</code></strong>　正しい記述
```

さて、次はいよいよ DTD の読み方について見てみよう。

演習 10

正しい記述と誤った記述をいくつか作成し、ブラウザで表示した場合、どのように表示されるか実験しなさい。 また、その実験結果からどのようなことが分かるか、併せて考察しなさい。

[*20] ただし、例外もある。例えば、blockquote 要素のように、ブロックレベルしか記述できないものもある。

1.13　DTD の読み方（要素編）

DTD にもいろいろな種類があるが、ここでは XHTML 1.0 の strict.dtd を例に説明を進める。

1.13.1　要素の定義

DTD での「**要素定義**」は、「**¡!ELEMENT**」で始まる部分を探す。 その部分は、次の形式で記述されている。

```
<!ELEMENT 要素名 内容モデル>
```

以下に例を示す。

```
<!ELEMENT p %Inline;>

<!ELEMENT h1 %Inline;>
<!ELEMENT h2 %Inline;>
<!ELEMENT h3 %Inline;>
<!ELEMENT h4 %Inline;>
<!ELEMENT h5 %Inline;>
<!ELEMENT h6 %Inline;>

<!ELEMENT br EMPTY>

<!ELEMENT em %Inline;>
<!ELEMENT strong %Inline;>
<!ELEMENT dfn %Inline;>
<!ELEMENT code %Inline;>
<!ELEMENT samp %Inline;>
<!ELEMENT kbd %Inline;>
<!ELEMENT var %Inline;>
<!ELEMENT cite %Inline;>
<!ELEMENT abbr %Inline;>
<!ELEMENT acronym %Inline;>
<!ELEMENT q %Inline;>
<!ELEMENT sub %Inline;>
<!ELEMENT sup %Inline;>
```

1.13.2　要素名

要素名で指定されたものがタグを指定する場合の名前となる。 つまり、属性部分を無視すれば、タグの書き方は、つぎの 2 種類となる。

```
書き方１　<要素名>
書き方２　<要素名 />
```

1.13.3 内容モデル

「**内容モデル**」は、その要素が持つことができる他の要素を定義している。 ここに「**EMPTY**」と記述されている場合、その要素は、「**空要素**」を意味する。 つまり、EMPTY 以外の指定がある場合、その要素は、必ず開始タグと終了タグを持つことになる。

```
<!ELEMENT br EMPTY>
```

EMPTY 以外の内容モデルは、以下の書式（表 1.1）で定義される。

書式	意味
a	a 要素を必ず 1 個持つ（必須ということ）ことを意味する。
a?	a 要素を 0 個または 1 個持つ（オプションということ）ことを意味する。
a+	a 要素を必ず 1 個以上持つ（必須ということ）ことを意味する。
a*	a 要素を 0 個以上持つ（任意ということ）ことを意味する。
a , b	a 要素の次に b 要素が出現することを意味する。
a | b	a 要素と b 要素が任意の順序で出現することを意味する。
#PCDATA	解析対象となる文字列を意味する。

表 1.1 内容モデルの書式

「**解析対象文字列（#PCDATA）**」とは、タグその他の文字列を含んだもので、 もし、それがタグであれば要素と解釈し、そうでなければ単純な文字列と解釈される文字列を意味する。 例えば、

```
<p>abc<em>123</em>def</p>
```

の場合、最初に全体が解析対象となり p 要素と解析される。 次にその内容 “abc123def” が解析され、次のように解釈される。

```
abcem 要素 def
```

そのため、em 要素の部分を要素ではなく文字列 “123” と表示したい場合、「**実体参照**」（表 1.2）を使用する。

文字	実体参照
<	<
>	>
&	&
"	"
’	'（XHTML 1.0 のみ）

表 1.2 文字の実体参照

つまり、次のようなコードになる。

```
<p>abc&lt;em&gt;123&lt;/em&gt;def</p>
```

この場合は、“abc「**em 要素**」def” ではなく、“123” のようにタグと解釈せずに単純に表示される。

1.13.4 パラメータ実体参照

これまでの要素のうち br 要素以外の内容モデルは、%Inline; と指定されている。 これは「**パラメータ実体参照**」と呼ばれるものだ。これは以下のようなルールで記述されている。

```
% 実体名;
```

これによって実体名を参照し、そこで指定されている置換テキストに置き換えて定義される。 このパラメータ実体参照の置換テキスト定義は<!ENTITY を探す。

```
<!ENTITY % 実体名 "置換テキスト">
```

例えば、“%Inline;” は次のように定義されている。

```
<!ENTITY % Inline "(#PCDATA | %inline; | %misc;)*">
```

これにより、内容モデルには、解析対象文字列 (#PCDATA)、パラメータ実体参照 (“%inline;” と “%misc”) が任意の順序で 0 回以上記述できることを意味する。 いま、“%misc;” を無視して残りの “%inline;” を探すと次のようになる。

```
<!ENTITY % inline "a | %special; | %fontstyle; | %phrase; | %inline.forms;">
```

同様に “%special;” と “%phrase;” について見て、残りを無視すると次のようになる。

```
<!ENTITY % special "br | span | bdo | object | img | map">
<!ENTITY % phrase "em | strong | dfn | code | q | sub | sup |
                   samp | kbd | var | cite | abbr | acronym">
```

これを逆に置き換えて行くと “%Inline;” は次のように置き換えられる[*21]。

```
<!ENTITY % Inline "(#PCDATA | br | span | bdo | object |
                   img | map | em | strong | dfn | code | q | sub | sup |
                   samp | kbd | var | cite | abbr | acronym)*">
```

このように DTD を読むことにより、p 要素、見出し要素のそれぞれの内側に p 要素、見出し要素が記述できないことも分かる。

1.14 DTD の読み方（要素編の復習）

これまで出て来た要素の DTD（要素部分のみ）を示す。頑張って読んでみよう。

```
<!--=================== Text Elements ====================================-->

<!ENTITY % special.pre "br | span | bdo | map">
<!ENTITY % special "%special.pre; | object | img ">
<!ENTITY % fontstyle "tt | i | b | big | small ">
<!ENTITY % phrase "em | strong | dfn | code | q |
                   samp | kbd | var | cite | abbr | acronym | sub | sup ">
<!ENTITY % inline "a | %special; | %fontstyle; | %phrase; | %inline.forms;">
<!ENTITY % Inline "(#PCDATA | %inline; | %misc.inline;)*">
```

[*21] なお、無視した部分は含んでいない

```
<!--================== Block level elements ==============================-->
<!ENTITY % heading "h1|h2|h3|h4|h5|h6">
<!ENTITY % block
     "p | %heading; | div | %lists; | %blocktext; | fieldset | table">
<!ENTITY % Block "(%block; | form | %misc;)*">

<!--================ Document Structure ==================================-->
<!ELEMENT html (head, body)>

<!--================ Document Head =======================================-->
<!ENTITY % head.misc "(script | style | meta | link | object)*">

<!ELEMENT head (%head.misc;,
                ((title, %head.misc;, (base, %head.misc;)?) |
                (base, %head.misc;, (title, %head.misc;))))>
<!ELEMENT title (#PCDATA)>
<!ELEMENT meta EMPTY>

<!--================== Document Body =====================================-->
<!ELEMENT body %Block;>

<!--================== Paragraphs ========================================-->
<!ELEMENT p %Inline;>

<!--================== Headings ==========================================-->
<!ELEMENT h1  %Inline;>
<!ELEMENT h2 %Inline;>
<!ELEMENT h3 %Inline;>
<!ELEMENT h4 %Inline;>
<!ELEMENT h5 %Inline;>
<!ELEMENT h6 %Inline;>

<!--==================== Inline Elements =================================-->
<!ELEMENT br EMPTY>   <!-- forced line break -->
<!ELEMENT em %Inline;>   <!-- emphasis -->
<!ELEMENT strong %Inline;>   <!-- strong emphasis -->
<!ELEMENT dfn %Inline;>   <!-- definitional -->
<!ELEMENT code %Inline;>   <!-- program code -->
<!ELEMENT samp %Inline;>   <!-- sample -->
<!ELEMENT kbd %Inline;>  <!-- something user would type -->
<!ELEMENT var %Inline;>   <!-- variable -->
<!ELEMENT cite %Inline;>   <!-- citation -->
<!ELEMENT abbr %Inline;>   <!-- abbreviation -->
<!ELEMENT acronym %Inline;>   <!-- acronym -->
<!ELEMENT q %Inline;>   <!-- inlined quote -->
<!ELEMENT sub %Inline;> <!-- subscript -->
<!ELEMENT sup %Inline;> <!-- superscript -->
```

1.15　箇条書き（リスト）の作り方

XHTML（HTML）では、3 種類の箇条書き（リスト）が用意されている。

- **無順序リスト**
- **順序リスト**
- **定義リスト**

無順序リストと順序リストは共通の文書構造を持ち、定義リストは若干異なる文書構造を持っている。(図 1.15)

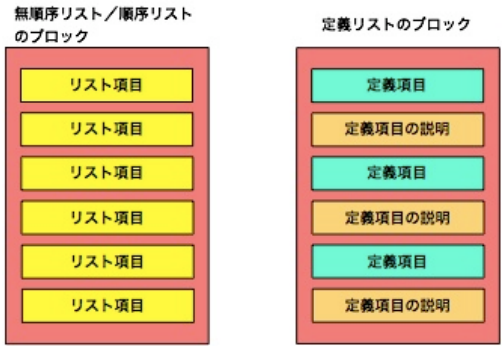

図 1.15　リスト構造の概要

また、「**３種類のリスト要素はいずれもブロックレベル要素**」に分類されている。 以下に関係する部分の DTD（一部）を抜粋しておく。

```
<!ENTITY % lists "ul | ol | dl">

<!ENTITY % block
     "p | %heading; | div | %lists; | %blocktext; | fieldset | table">
```

1.15.1　無順序リスト

「**無順序リスト**」は、順序関係のない、いわゆる通常の箇条書きの構造を表す要素である。 以下に無順序リストの DTD を示す。

```
<!ELEMENT ul (li)+>

<!ELEMENT li %Flow;>
```

無順序リストは「**ul 要素**」で表され、その内容モデルは、 1 つ以上の「**li 要素**」から構成される。つまり、ul 要素の子要素としては、 li 要素以外の要素は記述できない。

次に無順序リストを構成するリスト項目は、li 要素で表される。 その li 要素の内容モデルは、すべてのブロックレベル要素とインライン要素を記述することができる。

```
<!--================== Text Elements ====================================-->
<!ENTITY % special.pre
```

```
    "br | span | bdo | map">

<!ENTITY % special
   "%special.pre; | object | img ">

<!ENTITY % fontstyle "tt | i | b | big | small ">

<!ENTITY % phrase "em | strong | dfn | code | q |
                   samp | kbd | var | cite | abbr | acronym | sub | sup ">

<!ENTITY % inline.forms "input | select | textarea | label | button">

<!-- these can occur at block or inline level -->
<!ENTITY % misc.inline "ins | del | script">

<!-- these can only occur at block level -->
<!ENTITY % misc "noscript | %misc.inline;">

<!ENTITY % inline "a | %special; | %fontstyle; | %phrase; | %inline.forms;">

<!-- %Inline; covers inline or "text-level" elements -->
<!ENTITY % Inline "(#PCDATA | %inline; | %misc.inline;)*">

<!--================== Block level elements ==============================-->
<!ENTITY % heading "h1|h2|h3|h4|h5|h6">
<!ENTITY % lists "ul | ol | dl">
<!ENTITY % blocktext "pre | hr | blockquote | address">

<!ENTITY % block
     "p | %heading; | div | %lists; | %blocktext; | fieldset | table">

<!ENTITY % Block "(%block; | form | %misc;)*">

<!-- %Flow; mixes block and inline and is used for list items etc. -->
<!ENTITY % Flow "(#PCDATA | %block; | form | %inline; | %misc;)*">
```

では、いくつかのサンプルを見てみよう。

```
<ul>
    <li>リスト項目１</li>
    <li>リスト項目２</li>
    <li>リスト項目３</li>
</ul>
```

これは次のように表示（図 1.16）される。[*22]
次の例は、リストの入れ子（リストの中のリスト）のサンプルである。

```
<ul>
    <li>
        リスト項目１
        <ul>
```

[*22] 実際に表示されるマークはブラウザによって異なる。

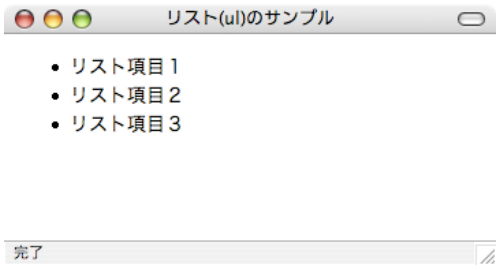

図 1.16　無順序リストの表示サンプル

```
            <li>リスト項目１－１</li>
            <li>リスト項目１－２</li>
        </ul>
    </li>
    <li>リスト項目２</li>
    <li>リスト項目３</li>
</ul>
```

これは次のように表示（図 1.17）される。[*23]

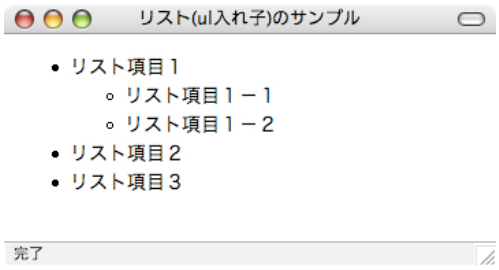

図 1.17　無順序リスト（入れ子構造）の表示サンプル

1.15.2　順序リスト

「**順序リスト**」は、順序関係のある、いわゆる番号付けされた箇条書きの構造を表す要素である。 以下に順序リストの DTD を示す。

```
<!ELEMENT ol (li)+>

<!ELEMENT li %Flow;>
```

順序リストは「**ol 要素**」で表され、その内容モデルは、 1 つ以上の「**li 要素**」から構成される。つまり、ol 要素の子要素としては、 li 要素以外の要素は記述できない。

次に順序リストを構成するリスト項目は、li 要素で表される。 これは ul 要素で使用したものとまったく同じものである。

では、いくつかのサンプルを見てみよう。

```
<ol>
```

[*23] 実際に表示されるマークはブラウザによって異なる。

```
    <li>リスト項目１</li>
    <li>リスト項目２</li>
    <li>リスト項目３</li>
</ol>
```

これは次のように表示（図 1.18）される。[*24]

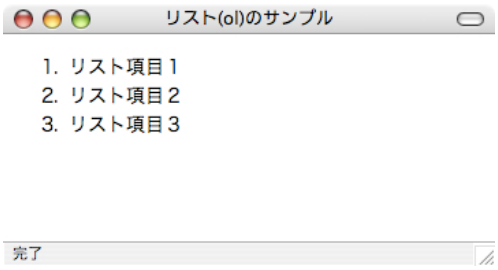

図 1.18　順序リストの表示サンプル

次の例は、リストの入れ子（リストの中のリスト）のサンプルである。

```
<ol>
    <li>
        リスト項目１
        <ol>
            <li>リスト項目１－１</li>
            <li>リスト項目１－２</li>
        </ol>
    </li>
    <li>リスト項目２</li>
    <li>リスト項目３</li>
</ol>
```

これは次のように表示（図 1.19）される。[*25]

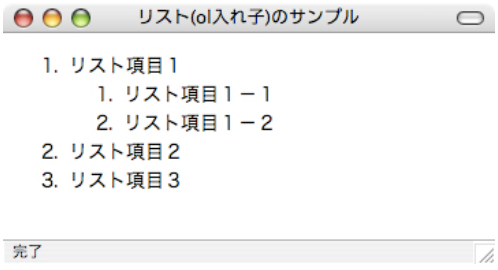

図 1.19　順序リスト（入れ子構造）の表示サンプル

また、各リストはお互いを入れ子にすることも可能である。

```
<ol>
    <li>
        リスト項目１
```

[*24] 実際に表示される番号の種類はブラウザによって異なる。
[*25] 実際に表示される番号の種類はブラウザによって異なる。

```
        <ul>
            <li>リスト項目１－１</li>
            <li>リスト項目１－２</li>
        </ul>
    </li>
    <li>リスト項目２</li>
    <li>リスト項目３</li>
</ol>
```

これは次のように表示（図 1.20）される。[*26]

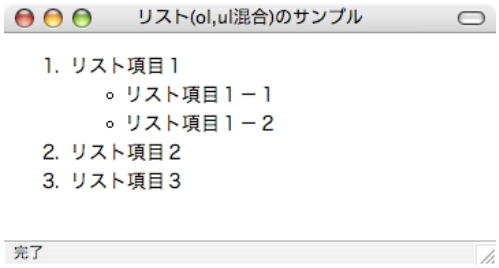

図 1.20　混合リスト（入れ子構造）の表示サンプル

1.15.3　定義リスト

「**定義リスト**」は、用語とその説明を表すなどの箇条書きの構造を表す要素である。 以下に定義リストの DTD を示す。

```
<!ELEMENT dl (dt|dd)+>

<!ELEMENT dt %Inline;>

<!ELEMENT dd %Flow;>
```

定義リストは「**dl 要素**」で表され、その内容モデルは、 1 つ以上の「**dt 要素**」と「**dd 要素**」から構成される。 通常、dt 要素と dd 要素が対で使用されるが、複数の dt 要素に対して 1 つの dd 要素、 あるいはその逆であっても文法的には正しい。また、その出現順序についても規定はないが、 一般的に dt 要素の次に dd 要素を書くことが多い。

定義リストを構成する項目は、dt 要素と dd 要素であるが、「**dt 要素が用語に相当し、その内容モデルは、インライン要素のみ**」である。 一方、「**dd 要素が説明に相当し、こちらの内容モデルはインライン要素、ブロックレベル要素いずれも記述できる**」。

では、代表的なサンプルを見てみよう。

```
<dl>
    <dt>用語１</dt>
    <dd>用語１の説明</dd>
    <dt>用語２</dt>
    <dd>用語２の説明</dd>
    <dt>用語３</dt>
```

[*26] 実際に表示される番号の種類、マークはブラウザによって異なる。

```
    <dd>用語３の説明</dd>
</dl>
```

これは次のように表示（図 1.21）される。

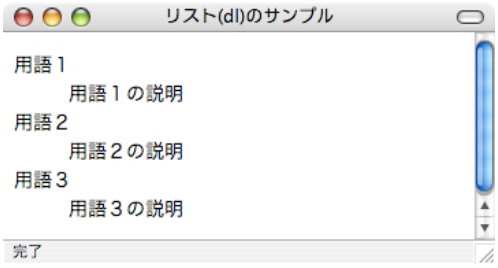

図 1.21　定義リストの表示サンプル

1.16　その他のブロックレベル要素

これまで勉強してきたブロックレベル要素としては、段落要素（p 要素）、見出し要素（h1〜h6 要素）、リスト要素（ul 要素、ol 要素、dl 要素）がある。 ここでは、それ以外の主なブロックレベル要素を勉強する。

1.16.1　address 要素

「**address 要素**」は、「**著作者情報**」を意味する要素である。内側にはインライン要素のみを記述することができる。 下に address 要素の DTD と使用例を示す。

```
<!ELEMENT address %Inline;>
```

```
<address>(C)Copyright 御隠居</address>
```

1.16.2　hr 要素

「**hr 要素**」は、「**水平区画線**」を意味する要素である。この要素は空要素である。 また、多くのブラウザでは水平線として描画するものが多い。 下に hr 要素の DTD と使用例を示す。

```
<!ELEMENT hr EMPTY>
```

```
<hr />
```

1.16.3　blockquote 要素

「**blockquote 要素**」は、「**ブロックレベルの引用文**」を意味する要素である。内側にはブロックレベル要素のみを記述することができる。 下に blockquote 要素の DTD と使用例を示す。

```
<!ELEMENT blockquote %Block;>
```

```
<blockquote><p>引用文の記述</p></blockquote>
```

1.16.4　pre 要素

「**pre 要素**」は、「**整形済みテキスト**」を意味する要素である。 内側には一部の要素（img、object、big、small、sub、sup）を除いたインライン要素のみを記述することができる。 段落等では、コード上での空白や改行が無視されるが、pre 要素内の空白や改行は有効となり、 その形式のまま表示する。下に pre 要素の DTD と使用例を示す。

```
<!ENTITY % special.pre "br | span | bdo | map">
<!ENTITY % fontstyle "tt | i | b | big | small ">
<!ENTITY % phrase "em | strong | dfn | code | q |
                   samp | kbd | var | cite | abbr | acronym | sub | sup ">
<!ENTITY % inline.forms "input | select | textarea | label | button">
<!ENTITY % misc.inline "ins | del | script">

<!-- pre uses %Inline excluding big, small, sup or sup -->
<!ENTITY % pre.content
   "(#PCDATA | a | %fontstyle; | %phrase; | %special.pre; | %misc.inline;
      | %inline.forms;)*">

<!ELEMENT pre %pre.content;>
```

```
<pre>Preformat1
   Preformat2
         Preformat3</pre>
```

このサンプルは次のように表示（図 1.22）される。実際、pre 要素を p 要素に書き換えて比較すれば違いが明確になるだろう。

図 1.22　pre 要素の表示サンプル

演習 11

1.2 節を Web ページとして作成しなさい。ただし、ファイル名は “sample16.html” とする。