

目次

1. CSS(Cascading Style Sheets)の利用について
 - 1-1 CSS を取り巻く環境
 - 1-2 CSS の利用に当たって気をつけること
 2. CSS を記述・適用してみよう
 - 2-1 どこに書けばいいのか？
 - 2-2 記述スタイルは？
 - 2-3 HTML との関連性は？
 - 2-4 どこまで可能なの？
 3. CSS の基本テクニック
 - 3-1 基本的な2カラムレイアウト
 - 3-2 ボックスの中央寄せ
 - 3-3 クロスブラウザ対策
- 付録 : よく使われるプロパティ一覧
- : レンダリングモードについて
4. 最後に

1. CSS(Cascading Style Sheets)の利用について

1-1CSS を取り巻く環境

皆さんが普段 Web 制作に携わっていると、「CSS」という言葉を耳にする機会は決して少なくないと思います。

「CSS」とは、(Cascading Style Sheets: カスケーディング・スタイルシート)の略称で、名前が長いので頭文字を読んで CSS (シー・エス・エス) と呼ばれています。

数年前の Web ページは、「テーブルレイアウト」という手法(<table>タグを使って視覚的にレイアウトを行う手法)で製作するのが主流で、CSS はフォントサイズの指定程度にしか使われていませんでした。

その大きな理由として、当時のブラウザは CSS を殆どサポートしていなかった事が挙げられます。この為、「わざわざ使うまでもない」という認識の下、CSS は表立って評価されるものではありませんでした。

しかし近年、その認識は完全に覆りました。その原因の一つとしては「SEO」があります。Yahoo! や Google といった検索エンジンが普及するにつれて、「何とか検索結果の上位に持って来れないか」と考えられたのが SEO です。

SEO 対策として重要なウエイトを占めるものが、「構造化され・適切に意味付けされた HTML」、つまり「検索ロボットが巡回しやすい HTML」であり（その他にも SEO の技術はありますが、ここでは割愛します）、それに伴って CSS の重要性が改められてきました。HTML において、<table>タグは「表」という意味を持ちます。テーブルレイアウトが主流であった頃、「スペーサーgif(空白の gif 画像)」という言葉が良く使われていました。「これは、表のセル同士の間隙間を作りたい場合、スペーサーgif という画像を挿入して無理矢理にレイアウトを調整していたのです。これだけでも無理のある方法なのですが（当時はその方法しかなかった、とも言えますが）、「表」というのは本来、並べるべきデータと、その値を整理するためにあります。しかしそこにデータとは関連性の無い空白の画像やテキストがいくつも入っていれば、当然それは表としての意味から外れてしまいます。そしてそれは、検索ロボットが巡回する際に正しく情報を読み取っていくことが困難になる事に繋がります。

検索ロボットは見た目を気にしません。つまり、「見た目がどうか」というより、「このページにはどんな情報が書かれているのか」に注目します。視覚的に綺麗なデザインを望むのはページを閲覧する私たちです。

そこで、Web ページにおいて「情報」と「見た目」を分離する手法として CSS が注目され始めました。また、各種ブラウザのバージョンが上がるにつれて CSS のサポートも充実してきたこともあり、CSS で Web ページの見た目を調整する方法が現在の主流となって

きました。

このため、適切な HTML を組む人、デザイン・レイアウトを行う人との分担作業も可能になるだけでなく、より自由なデザインが可能になる、作業・メンテナンス効率が格段に上がる、HTML は綺麗なままなので検索ロボットも巡回しやすくなる、といったように、CSS を用いることで様々なメリットが得られるようにもなりました。

これから登場する・開発する Web サイトでは、CSS は無くってはならない技術と言っても過言ではないでしょう。今回は CSS の基本的な扱い方、テクニックを学んでいただきたく思います。

1-2. CSS の利用に当たって気をつけること

CSS にもバージョンがあり、現在 W3C(※注1)が勧告しているのは CSS2.1 です。CSS3 も策定中のようなのですが、現在最も一般的な CSS2.1 を取り上げます。

近年のブラウザが CSS をサポートしてきたとはいえ、それは完全ではありません。

しかし、ページを見る人が使うブラウザは多種多様であり、「Opera だとちゃんと表示されているのに、IE だとどうしてもレイアウトが崩れてしまう」という事は頻繁にあり、またその逆も然りです。さらに、同じブラウザでも Windows と Mac では表示が違う場合もあります。かといって、そのままにはできません。見た目が悪い Web ページは印象も悪くなってしまいがちです。せっかく素晴らしいデザインをしても、レイアウトが出来ていなければそれは上手く伝わりません。

全てのブラウザで全く同じように見せる CSS というものは現時点では難しいと思います（可能ではありますが、使えるプロパティ[後述]が限定されてしまいます）。どのブラウザがどのプロパティをサポートしているか、という事まで覚える必要はありませんが、ブラウザによる癖を把握し、それと上手く付き合いながら、「なるべくどのブラウザでも同じように表示される CSS」を作成するのが望ましいと私は考えます。少しくらい表示が崩れてしまっても、それを補うほどのメリットは十分に得られるはずです。

それでは、実際に CSS を記述してみましょう。

※本講座で取り上げる HTML は、「正しく構造化された HTML」ということで、XHTML 1.0 Transitional を扱うこととします。XHTML の記述ルールの詳細は割愛いたしますが、文中に出てくる物は、そのルールに沿って書かれたものとします。

※1 World Wide Web Consortium (ワールド・ワイド・ウェブ・コンソーシアム) の略称。Web で使用される各種技術の標準化を推進する為に設立された標準化団体、非営利団体 (Wikipedia より抜粋)。

2. CSS を記述・適用してみよう

2-1 どこに書けばいいのか？

CSS の記述法として、「外部 CSS ファイル」、「<head>スタイル」、「インラインスタイル」という 3 つの記述方法があります。現在最も一般的なのが外部 CSS ファイルですが、まずは最も直感的に CSS の効果が見えるインラインスタイルから始めてみましょう。

以下のような HTML を記述してみましょう。

```
<div style = " width : 100px; height : 100px; background-color : #000000;"></div>
```

これは HTML タグですね。これを Web ブラウザで確認すると、真っ黒な正方形が描かれています。これは<div>タグにインラインスタイルが適用されたからです。

このように、HTML タグの style 属性に CSS を書き込むものをインラインスタイルと呼びます。これはとても直感的でよいのですが、冒頭で紹介した「デザインと HTML を分離できる」という CSS のメリットから外れてしまうので、あまり使われていません (Dreamweaver (以下、DW) でスタイル調整を行うと、このような記述になる時が多いです)。

それでは、もう少し HTML と CSS を分離してみましょう。今度は、HTML の<head></head>内に記述してみます。

```
<head>
<style type="text/css">
    div {
        width : 100px;
        height : 100px;
        background-color : #000000;
    }
</style>
</head>
```

このように記述しても、先ほどと同じ結果となります。<style>タグを使うことによって、CSS をその中に一括して指定できます。これにより、<body>内の<div>タグにはインラインスタイルのような余計な属性がつかず、綺麗になります。

しかし、これにもやはり問題があります。この方式だと、「その HTML ファイルにしか適用

できない」というデメリットがあります。これは同じく「一括したデザイン変更ができる」というメリットから外れてしまいますし、何より適用する CSS が膨大になると、<head> タグがとても長くなってしまいます。

これらの問題を解決する手段として、「外部 CSS ファイル」という方法が用いられます。その名のとおりに、CSS ファイルを外部のファイルとして、それを HTML から参照する方法です。まずはやってみましょう。

[ファイル] → [新規]と進むと、項目の中に「CSS」があります。それを選択すれば、DW なら新規に CSS ファイルを作成することが可能です。

それでは、新規 CSS ファイルを作成し、先ほどの<style>タグの中をコピーしましょう。そして「style.css」という名前をつけて、HTML ファイルを同じ階層に保存しましょう。あとは、この CSS ファイルを参照する記述が必要です。<head>タグ内に以下のように記述してください。

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

この<link>タグについて少しだけ説明します

まず、「rel」属性です。これは、この HTML ファイルと参照先のファイルの関係を定義します。この場合、参照するのは CSS ですので、値は「stylesheet」となります。

次に、「type」属性です。これは、参照先のファイルの MIME タイプ（どのようなファイル形式をとっているか）を定義します。CSS の場合は「text/css」と指定します。

「href」属性は、その参照先のファイルへのパスを指定します。

この他にも「media」属性というものもあり、この属性によって出力用の CSS を切り替えることが出来ます。例えば、「print」と指定すれば印刷用の CSS、「handheld」と指定すればモバイル用に参照できます。何も指定しなければ通常の PC 用に参照します。

これで参照は完了です。ブラウザで確認すると、やはり同じ結果が得られます。

外部 CSS ファイルを用いると、以下の全ての HTML ファイルに上記の<link>タグを記述すれば、全てのページに同じ CSS を適用することができます。インラインスタイルや<head>スタイルのようなソースは無くなり、HTML はクリーンになり、かつ外部 CSS を修正する事で全てのページデザインを一括して変更することが出来るようになります。また、修正を行う場合も CSS だけを分離して確認できますので、これから先は全ての外部 CSS の「style.css」を修正していくようにします。

2-2 記述スタイルは？

今、style.css には、

```
div {  
    width : 100px;  
    height : 100px;  
    background-color : #000000;  
}
```

とだけ書かれていますね。一見して分かりにくいですが、これが基本的な CSS の記述スタイルです。

以下に図を示します。



まず、先頭の「div」。これは「セレクトタ」と呼ばれ、HTML 中のタグを示すものです。HTML のタグであれば、殆どのタグに対してセレクトタを指定することが出来ます（例外も有り）。

次に、「{ }」で示される部分がブロックです。指定されたセレクトタは、このブロック内に書かれた CSS を適用します。

「width」と書かれているのはプロパティです。沢山のプロパティがありますが、これらを指定し、「: (コロン)」に続いて値を入れることにより、CSS が適用されます。プロパティには把握できないほどの数がありますが、一般的に良く使うプロパティは決まっていますので、それらを覚えてしまえばそれほど迷うことはないでしょう。また、一つプロパティを指定するたびに「; (セミコロン)」で区切ります。

以上により、CSS の一般的な書式は以下のようになります。

```
セレクトタ {  
  
    プロパティ    :   値 ;  
  
    プロパティ    :   値 ;  
  
    .....  
  
}
```

これらを必要な分だけ繰り返して、CSS は適用されていきます。それでは試しに、先ほどの<div>タグの横幅を 2 倍にして長方形にしてみてください。

・
・
・
・
・

この場合は、width プロパティを 2 倍にすれば良いですね。

```
div {  
    width : 200px;  
    height : 100px;  
    background-color : #000000  
}
```

これで横幅が 2 倍の長方形になりました。

2-3 HTML との関連性は？

CSS は HTML があって初めて有効となるものです。そのため、CSS を深く知るには、まずは HTML についての知識が必要となります。ここでは少しだけ CSS から離れて、HTML

について学んで行きたいと思います。

- ・ (X)HTML

Web に携わる人ならおそらく誰もが知っており、ある程度書ける方が多いのが HTML です。Web ページには勿論、プログラマでも出力結果を書き出すために HTML の知識は必須であると言えます。しかしながら、その使用する HTML タグがどんな意味を持つか、構造化されている、とはどういうことなのかを知っている人は少ないと思います。それらを専門としている人を「マークアップエンジニア」と呼ぶことがあるくらいに奥の深いものです。誰もが簡単に習得できる反面、難しい一面も併せ持っていると言えるでしょう。

- ・ 簡単な理解度チェック

以下の HTML ソースを見てください。一見して問題無いように見えますが、現在の XHTML においては、実は細かい点での間違いや不適切な表記があります。

```
<body>
  <div id="box">
    <table border="1" class="test_table">
      <tr>
        <td>カラム 1 とは？</td>
        <td><b>その説明。</b></td>
      </tr>
    </table>
    <ul>
      <li><a href="test.html">リンクテキスト</a></li>
      <li><a href="test2.html">リンクテキスト 2</a></li>
    </ul>
    <p>リンクテキストについての説明文</p>
    <form action="test.cgi" method="post">
      <input type="text" name="name" id="form" />テキスト 1 <br />
      <input type="radio" name="radio" id="form" checked />ラジオボタ
ン 1
      <input type="image" src="buttonimage.jpg" />
    </form>
```

```
</div>
</body>
```

上記の間違いを指摘できれば HTML についての知識は問題ありません。(明らかな間違いが 2 つ、適切でない表記・方法が 5 つあります。探してみてください。)

- ・ インライン要素とブロックレベル要素

HTML にはインライン要素とブロックレベル要素という物があるのはご存知でしょうか？ インライン要素とは、その名のとおり、「行中に現れることの出来る要素」であり、ブロックレベル要素は、「段落等の区切りとしての要素」として意味合いがもたれています。(なお、<td>,<th>等は例外として「テーブルセル要素」とされていますが、ここでは同じブロックレベル要素として考えていただいてもかまいません。)

これらの使い分けが CSS には重要となってきます。(具体的には、インライン要素には、margin[後述]上下、padding[後述]上下方向の値が無効になります。)簡単に言うと、「インライン要素の中にブロックレベル要素は入れない」という事になります。行中に段落区切りが来るのは構造化された HTML としては文法違反となります。よくある例を挙げます。

```
<body>
  <a href="hogehoge.html">
    <p>
      リンクテキストです。
    </p>
  </a>
</body>
```

このように書かれた HTML は意外とよく目にします。一見ブラウザでちゃんとテキストにリンクが付けられていますが、実は HTML では文法違反なのです。

その理由は簡単です。<a>はインライン要素であるのに対し、<p>タグはブロックレベル要素なのです。リンクの中に段落があるのはおかしいですね。正しくは、

```
<body>

  <p>

    <a href="hogehoge.html">
      リンクテキストです。
    </a>

  </p>

</body>
```

こうなります。これなら正しく構造化されています。また、<p>タグの中に<p>タグを入れる（ネストすると言います）ことも実は文法違反なのです。段落の中に段落があるのは文としておかしいですね。さらに言うなら、中身を持たないタグを頻繁に使用するのも避けたいところです。例えば、行間を調整するのに
タグを何度も使うケースがありますが、これは文法違反、とまでは行きませんが、使用は推奨されていません。

これらを回避するために、<div>タグという汎用区切り要素がよく用いられますが、あまり頻繁に使用すると、今度はソースが読みにくくなり、結果として構造が分かりにくくなるので注意が必要です。

以上で簡単な HTML の説明は終わります。それでは、話を CSS に戻しましょう。

2-4 どこまで可能なの？

CSS はページを「装飾」するために書くものです。近年のモダンブラウザであれば使い方によっては簡単な動きのある装飾も可能ですが、やはり CSS はページのレイアウトに用いることが第一です。具体的な動きをつけるには JavaScript 等のクライアントサイドスクリプトを用いますが、これらのスクリプトもまた、CSS と非常に強い関連性を持っているので、CSS を学んでおくことは JavaScript 等を学ぶ際にとっても役立ちます。それでは、CSS の一歩進んだ使い方を見てみましょう。

- ・ 特定の id 名、クラス名に対してのみ適用させる

HTML のみでコーディングをする際には、id や class 属性はそれほど重視されません。しかし、CSS を適用する際に、これらはとても大きな意味を持つようになります。以下のコードを見てください。

```
<body>
  <div>
    <p>
      長方形
    </p>
  </div>
  <div>
    <p>
      正方形
      <span>実際の表示は？</span>
    </p>
  </div>
</body>
```

このようなコードに、先ほどの style.css を<link>タグで参照させるとします。すると、黒い長方形が二つできてしまいます。これは、div のセレクトアが、HTML 上の全ての<div>タグに対して適用されるからなのです。しかし、二つ目の<div>タグは正方形であって欲しいのです。このような場合には、二つ目の<div>タグに id 名、または class 名をつけてやります。

```
<div id="square">
  <p>
    正方形
    <span>実際の表示は？</span>
  </p>
</div>
```

これにより、CSS 側で id により<div>タグを個別に識別することができます。CSS で id 名を識別させるには、セレクトアに続けて「#id 名」と書きます。先ほどの CSS ファイルに追記しましょう。

```
div#square {
    width : 100px;
    height : 100px;
    background-color : #000000;
}
```

これで二つ目の<div>タグには、それ専用のスタイルが付けられました。

また、これらの処理は、class 名をつけたときもほぼ同様です。実際に id ではなく class をつけたときは、セレクトタに続けて「.(ドット) class 名」と書きます。実際にやってみましょう。

・
・
・
・

・ HTML

```
<div class="square">
    <p>
        正方形
        <span>実際の表示は？</span>
    </p>
</div>
```

・ CSS

```
div.square {
    width : 100px;
    height : 100px;
    background-color : #000000;
}
```

※ id と class 名の違いは？

CSS を始めた方はよくこの疑問に辿り着きます。

簡単な違いは、「id は HTML 中に一度しか使ってはいけない、class は何度でも使える」という違いがあります。ですから、同じ要素にまとめてスタイルを適用したい場合は class

名を、全ての要素を内包するボックスのような一度しか書かないようなものには id 名をつけるといった使い分けが出来るでしょう。

・ テキスト色を変えてみる ～ プロパティの継承 ～

それでは、次に文字の色を変えてみましょう。今のままですと真っ黒なボックスなので中に書かれた文字が読めないですね。テキストの色を変えるには、「color」プロパティを使います。値には RGB の 16 進数値が入ります。今回は良く見えるように白に変えてみましょう。最初の div セレクタのブロック内に、

```
color : #ffffff ;
```

と追記しましょう。白は 16 進数では #ffffff です。これを上書き保存してブラウザで確認すると、少し期待とは違う結果が得られます。下の正方形のボックスのテキストの色も変わっていますね。

実は、CSS には「継承」という仕組みが用意されています。これはとても便利で（このために苦労することもあるのですが）、予めベースのスタイルをセットしておき、細かい部分を別途追記・上書きしていく方法が可能となります。

これらは、セレクタの優先度と、記述された順番によって決定付けられていきます。例えば、id と class とが両方付けられ、それぞれに CSS が記述されていた場合、優先されるのは id の CSS なのです。

（優先順位の詳しい説明は今回は省略させていただきます。）

これらを加味すると、div.square のセレクタは、直前に書かれた div セレクタの内容を継承していますね（厳密には継承ではありませんが、ここでは継承していると思っていただいたほうが分かりやすいです）。なので、実際は、

```
div.square {  
    width : 100px;  
}
```

と書くだけで同じ表現ができます。height プロパティと background-color プロパティは継承していますので、width プロパティのみを上書きすれば期待通りの結果となります。この継承の仕組みを上手く使えば、少ないコードで効率的に CSS を適用させることができます

す。覚えておいてください。

- ・ 子セレクタ、子孫セレクタ

では次に、正方形の中にある「」タグに CSS を適用しましょう。これも前と同様に、タグに適切な id、class をつけてもよいのですが、CSS には「子セレクタ」という便利な記述方法があります。

「子セレクタ」とは、HTML の階層を辿って指定したタグを装飾するセレクタです。

例を挙げましょう。今回は、<div class="square">の中の<p>タグ、その中のタグというように階層を辿っていきます。

```
div.square p span {  
    font-size : 1.3em ;  
}
```

このように記述すると、タグ内のフォントサイズが大きくなります。

セレクタの部分を見ると、上から順に div, p, span と階層を降りていくのが分かります。

これらセレクタの間は半角スペースでつなぎます。全角を入れると上手く動作しないので注意してください。また、これは、

```
div.square > p > span {  
    font-size : 1.3em ;  
}
```

と書くことも出来ます。後者の方が階層を降りていくのが分かりやすいですが、Internet Explorer 6 では対応していないので、実際の使用では前者を使用する方が良いでしょう。

一見してこれらのセレクタは同じように見えますが、前者 (>をつけない記述) は「子孫セレクタ」、後者 (>をつける記述) は「子セレクタ」と分類されます。

今回は同じ表現となりましたが、これらには違いがあるのです。ではその違いとはなんのでしょうか？

HTML において、ある要素の中に記述する要素はそれぞれ関係を持つこととなります。

例えば、

```
<div>
  <p>
    <a>リンク</a>
  </p>
  <p>
    <a>リンク 2</a>
  </p>
</div>
```

このような HTML があるとします。この時、先頭の<div>要素は中の<p>、<a>要素を包みます。この時、<p>要素は<div>要素の直下にあるので、<p>要素は<div>要素の「子要素」と呼ばれます（逆に、<div>要素は<p>要素の親要素と呼ばれます）。さらにその中の<a>要素は<div>要素から見て子供の子供となるので、「孫要素」となります。私たちの家族の関係と同じ考え方です。ということは、2 目目の<p>要素もまた<div>要素の子要素となり、それぞれ2つの<p>要素は「兄弟要素」となります。

少し長くなりましたが、これらの関係をたどる上で、先ほどの「子セクタ」と「子孫セクタ」の用途の違いが分かってきます。

「子セクタ」の場合、「左の要素の直接の指定子要素のみ」に CSS を適用し、「子孫セクタ」の場合は、「左の要素内に存在する全ての指定要素（つまり、孫も含む）要素」に CSS を適用します。違いが分かりにくいので、下の例を見てください。

```
<div>
  <p>
    <span>
      テキスト
    </span>
  </p>
</div>
```

上記のような HTML があったとします。この時、最下層の要素に CSS を適用した場合には、

```
div span {
  CSS ルール
}
```

このように書くことで CSS が適用されます。この場合、「子孫セレクタ」を使用したので、先頭の<div>要素の中に含まれる全ての要素に CSS が適用されるので、結果として希望する要素を CSS で装飾することが出来ます。

では、これを子セレクタで書くとどうなるでしょうか。CSS を、

```
div > span {  
    CSS ルール  
}
```

と書くと、希望する要素には CSS が適用されません。これは、先頭の<div>要素から見て、要素は「孫要素」に当たるからです（実際に、<div>の子要素は次の<p>となります）。この記述は一見して同じ様に見えますので、使用する際には注意が必要です。また、子孫セレクタを使用する場合も、「div span」ではなく、「div p span」と記述したほうが間違いが少なく、望まない要素に CSS が適用されることも少なくなります。

このように、階層を下ってセレクタを作成することもできます。余計な id や class をつける必要も無く、より綺麗な HTML のまま保つこともできます。ただし、このように階層を下る場合、やはりきちんと階層・構造化された HTML が前提であることも忘れないでください。

- ・ 疑似クラス

CSS のセレクタの中で、少し特殊なものとして「疑似クラス」という物があります。これは主にユーザーのアクションに対して適用する CSS を変えたいときに使います。しかしながら、Internet Explorer では対応していない物もありますので、使用には注意が必要です。ここでは疑似クラスの中で代表的な「:hover」疑似クラスを使ってみましょう。この疑似クラスは、Internet Explorer では<a>タグにのみ使用できます。

リンクにマウスが乗ったときにテキストの色を変えたりする際に使います。以下のコードを見てください。

- ・ HTML

```
<a href="test.html">リンクテキスト</a>
```

・ CSS

```
a {  
    color : #0000ff;  
}  
a:hover {  
    color : #cc0000;  
}
```

HTML の方は何の変哲も無いリンクタグです。あとはこれにプロパティを適用させるのですが、二つ目のブロックには、「a:hover」と疑似クラスが使われています。適用させたいセレクトに繋げて「:hover」と記述するだけです。これをブラウザで確認すると、マウスを合わせるとテキストの色が変わりますね。これはマウスが乗った時に「a:hover」の方のCSSが適用されるからです。そして、マウスが外れると、元通り a の CSS が適用されます。

これは「ロールオーバー」という手法でよく使われています。背景画像を指定するプロパティがありますが、それを: hover で違う画像に差し替えれば、画像のロールオーバーは実現できます。但し、先述のように Internet Explorer では<a>タグにしか効果がありません（エラーが出たりはしませんが）。もしも Firefox や Opera で見れば良い、ということであれば、<div>などの要素にも: hover は適用可能です。他にもある疑似クラスの代表的なものを下表に示しておきます。

※:after.:before については、「疑似要素」と呼ばれていますが、使い方は基本的に疑似クラスと同じです。

:active 疑似クラス	マウスがクリックされた際の反応
:after 疑似要素	指定した要素の最後に擬似的に要素を入れる
:before 疑似要素	指定した要素の最初に擬似的な要素を入れる
:focus 疑似クラス	カーソルがフォーカスされた時の反応

※:active については、Internet Explorer は<a>タグにしか効果が無い。

※その他3つの疑似クラス・要素は Internet Explorer は対応していない。

また、頻繁に使用されるプロパティは末尾の「付録」にてご紹介します。そちらも参照し

てください。

3. CSS の基本テクニック

3-1 基本的な2カラムレイアウト

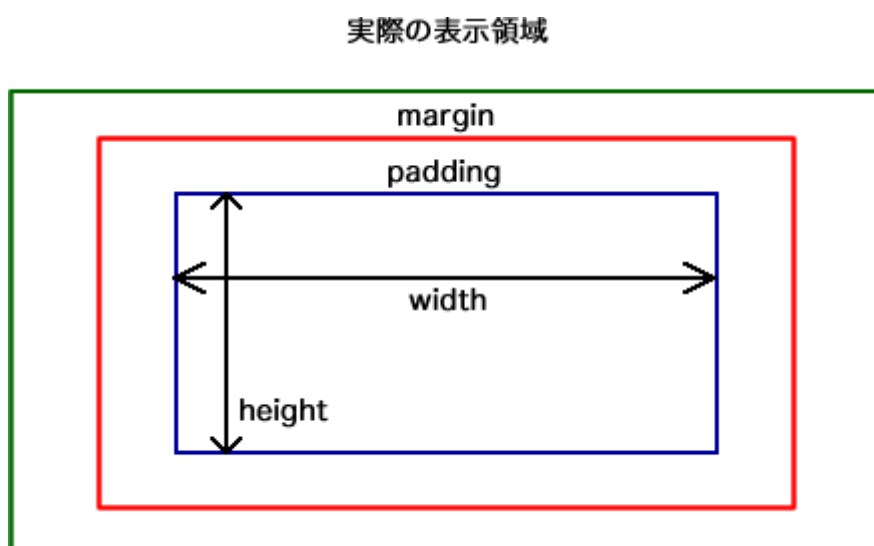
さて、ここまで基本的な CSS の記述の方法を学びましたので、ここからはより実践的な CSS でのレイアウトについて勉強していきましょう。ここから先、初めて出てくるプロパティばかりとなりますが、末尾の付録と合わせて随時説明を行いますので、覚えながらゆっくり進めていきましょう。

- ・ ボックスモデルという解釈

CSS でレイアウトを行う前に、「ボックスモデル」という概念について知っておく必要があります。簡単に言えば、ボックスモデルとは、「<div>のようなボックスを組み合わせるレイアウト」ということで、言わばパズルのようなものです。

<div>といったブロック要素を自由自在に組み上げて、Web ページをレイアウトしていくのですが、それに当たって数点気をつけておかなければいけないことがあります。

- ・ ボックスモデルの図



上の図を見てください。これはボックスモデルを説明するときに良く使われるものですが、ここで margin と padding という新しいプロパティが出てきましたね。先にそれらを説明

します。

- ・ margin と padding

margin(マージン)はその名のとおり、余白を意味します。上下左右それぞれに指定できます。指定した分だけ余白を取ってボックスの位置を決めます。

逆に padding(詰め)は、指定した部分に空の内容を「詰め」ます。これらの二つは同じように見えますが、「背景色(画像)が表示されるか、されないか」とで分類することが出来ます。

margin の場合、余白を取るなので、その部分には背景は存在しません。しかし、padding の場合、空けたスペースには空の内容が詰まっていますので、背景は表示されるのです。

ややこしいものですが、使いながら表示を確認していけば分かりやすいと思います。

さて、話をボックスモデルに戻します。上図の「実際に表示される領域」とは、横幅の場合は、CSS で指定された $width + margin(左右) + padding(左右)$ で算出されるのです(縦幅の場合は同じく $height + margin(上下) + padding(上下)$ で算出されます)。また、枠線を表示する border プロパティを指定した場合、その線の太さも加算されます。

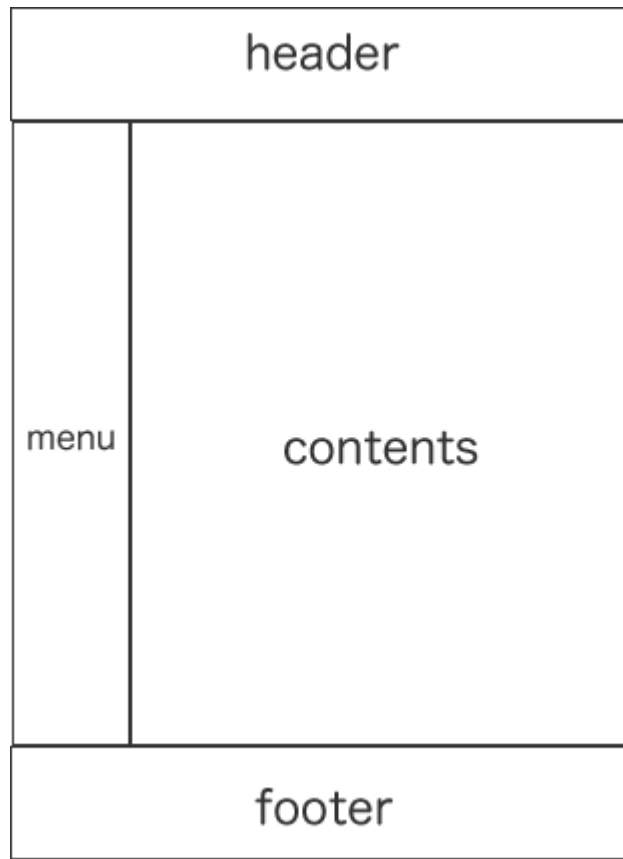
つまり、実際に 500px のボックスの表示を期待する場合、余白を 10px ずつ取るのであれば、width プロパティは $(500 - 10 \times 2)$ で 480px と指定しなければなりません。また、枠線を 2px 取るのであれば、476px となります。このように、ぴったりとしたレイアウトを行うには、少しだけ計算を行う必要があります。

問題はまだまだあります。Internet Explorer には「レンダリングモード」というモードがあり、このモードによって実際の表示領域の計算が違ってしまう場合もあるのです(詳しくは付録を参照下さい。ここでは省略します。)

さて、これらを踏まえて実際にレイアウトを行いながら説明していきましょう。

- ・ 2カラムレイアウトを作ってみる

昨今のブログや企業サイトでも見られるごく一般的なレイアウトで、メニューとコンテンツ(カラム)を横に2つ並べたものです(下図)。

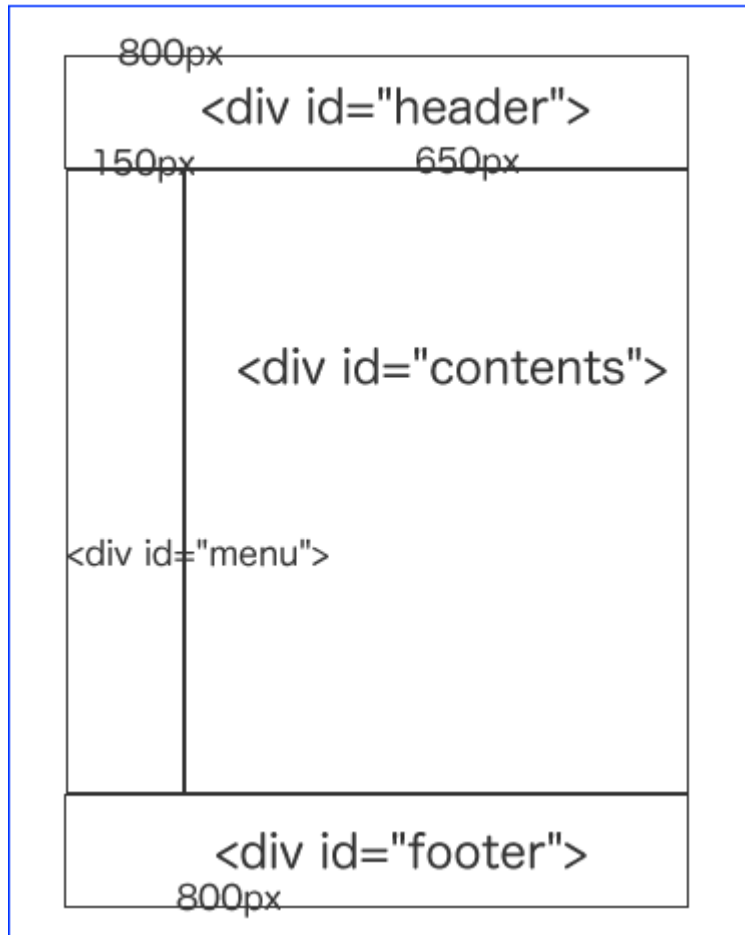


基本的な2カラムレイアウト

上部にヘッダ、左にメニュー（左カラム）・右にコンテンツ（右カラム）、そして下部にフッタがあるレイアウトですね。

これを CSS で作成するには、まずは HTML から順序良くマークアップしていく必要があります。この図にマークアップを行った後の図を下図に示します。

これらを内包する<div id=box">



次のようにマークアップします。

- ・ HTML

<body>

```
<div id="box">  
  <div id="header">  
    ヘッダ領域  
  </div>  
  <div id="contents">  
    コンテンツ領域  
  </div>  
  <div id="menu">
```

```
                メニュー領域
            </div>
            <div id="footer">
                フッタ領域
            </div>
        </div>
</body>
```

各領域の<div>はそれぞれ一度しか使用しないので、id 属性を付けました。これをブラウザで確認すると、ただ単に各領域の文字が縦に並んでいるだけです。分かりにくいので CSS で装飾していきましょう。

まずは、全ての要素を内包する<div id="box">の横幅を決めましょう。今回は 800px とします。横幅は width プロパティですね。

```
div#box {
    width : 800px ;
}
```

次に、その中の<div id="header">に CSS を適用します。横幅は box と同じにしておきます。ヘッダにはある程度高さが必要なので、ここでは 100px にしましょう（好きな数値で構いません）。

```
div#header {
    width : 800px ;
    height : 100px ;
}
```

- ・ 二つのボックスを横に並べるには？

ここまでは簡単でした。しかし、ここからが大事なポイントとなります。通常、このようなボックスを2つ並べるには2つの手法があり、「float レイアウト」・「position レイアウト」と呼ばれます。どちらでも実現は可能ですが、今回は最も一般的に使われている「float レイアウト」を用いて横に並べてみましょう。

- ・ float プロパティを使うときの注意点

実は、この「float」プロパティは、CSS を始めたばかりの方には少し扱いづらい面があります（後述しますが、clear プロパティで解除しなければならなかったり、margin や padding を指定する際にカラム落ちが発生しやすいためです）。しかし、このプロパティを自在に扱えるようになれば、CSS の表現の幅が一気に広がるので、焦らずにゆっくりと覚えていきましょう。

float というプロパティの値には、「left」と「right」の値が指定できます。これを指定した要素は「浮動ボックス」と呼ばれるものになり、ボックスごとに右寄せ、左寄せが可能になります。

今回はメニューが左、コンテンツが右側なので、<div id="menu">は left、<div id="contents">は right に設定しましょう。このとき、横幅を指定することを忘れないで下さい。図に従えば、メニュー部分は 150px、コンテンツ部分は 650px ですね。次に高さも指定してみましよう。CSS は次のようになります。

```
div#menu {
    float : left ;
    width : 150px ;
    height : 600px ;
}
div#contents {
    float : right ;
    width : 650px ;
    height : 600px ;
}
```

本来ならば高さは指定しないのですが（内容が詰まっていれば高さは自然に増えていきます）、今回は内容が無いので、擬似的に高さを指定します。

ブラウザで確認すると、テキストが離れましたね。しかしこれでは分かりにくいので、枠線をつけて実際のボックスがどうなっているのか見てみましょう。

枠線の付与には border プロパティを使います。

```
border : solid 1px #000000;
```

今度は値が複数ありますね。これは CSS の値を一括指定するやり方です。

線の種類、太さ、色の順に指定できます。勿論個別にも設定できるのですが、

```
border-top-style : solid ;           ー上部の枠線の形式を指定
border-top-width : 1px ;             ー上部の枠線の太さを指定
border-top-color : #000000;         ー上部の枠線の色を指定
```

と、上側の線の指定だけで3つも書かないといけません。あと左右、下線も描くのはとても面倒なので、前述のような省略形で一括指定できます。是非この方法を覚えてください。

さて、各 div# のセレクトに枠線を付与します。

```
div#header {
    border : solid 1px #000000;
    width : 798px ;
    height : 100px ;
}
div#menu {
    border : solid 1px #000000;
    float : left ;
    width : 150px;
    height : 600px ;
}
div#contents {
    border : solid 1px #000000;
    float ; right ;
    width : 650px ;
    height : 600px ;
}
```

このように書き換えて Web ブラウザで確認すると、おかしな現象がおこっていることが確認できるはず。メニュー領域が下のほうにずれてしまっています。さらに、Firefox の方は「フッタ領域」というテキストがヘッダの下に来ていると思います。

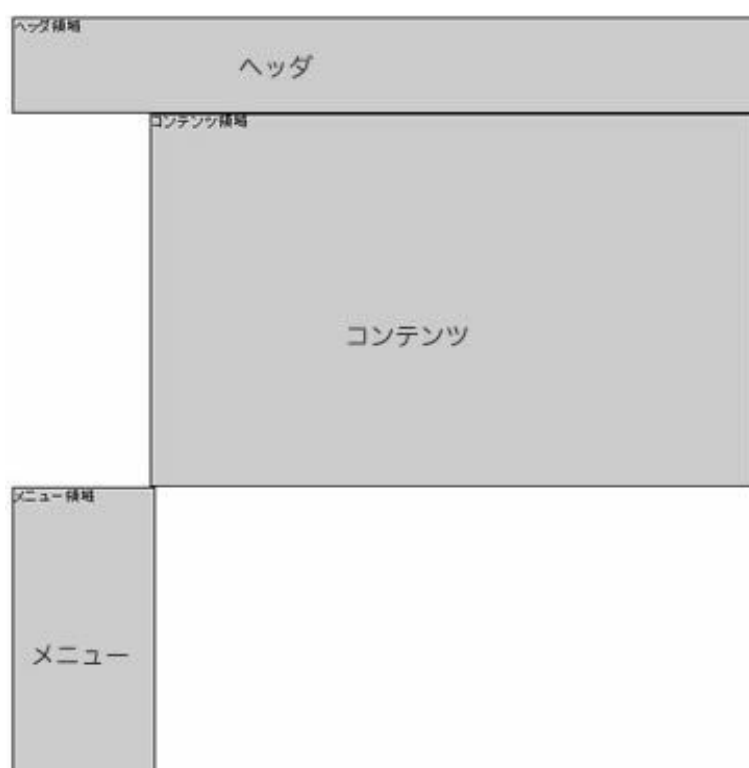
これは「カラム落ち」という現象(下図参照)で、float を使ったときによくある現象です。この現象は、横に並べた2つのボックスの横幅の合計が、包まれるボックスの横幅を越えてしまった時に起こる現象です。では、何故カラム落ちが起こってしまったのでしょうか？それは、少し前に見ていただいた「ボックスモデル」を覚えていれば分かります。

「枠線のみだけ実際の表示領域が増えてしまった」のです。つまり、メニュー領域は実際には 152px、コンテンツ領域は 652px となってしまう。この二つを包むボックスは

800px ですから、合計値が 800px よりも大きくなったので、カラム落ちが起きてしまいました。これを修正するには、各 width の値を枠線の分だけマイナスすることで解決します。実際にやってみてください。…解決しましたでしょうか？

menu の横幅が 148px、contents の横幅が 648px にすれば解決しますね。

このように、1px でも超えるとこのような現象が起きてしまいます。もしカラム落ちが起こった場合、内容領域の横幅に影響を及ぼすものを探してみてください。内容領域を超えるような画像を配置していないか、等です。



カラム落ちの例

- ・ float は後に続くボックスにも影響を与える

さて、これでおおまかな 2 カラムレイアウトは完成したように見えます。しかし、フッタはちゃんと表示されているように見えますが、実はたまたまピッタリと入っているので綺麗に見えるだけで、実際は不安定な状態にあります。試しに contents の高さを 400px にしてみてください。フッタがコンテンツの下に回り込んでしまいます (下図)。

これは、float が後に続く領域にも影響を与えているからです (厳密に言う「継承」とは、このことを指します)。これを回避するには、clear プロパティを使用し、float の解除をす

する必要があります。float を指定した場合、終了したい場所で clear を指定する。これを忘れないようにしてください。



フッタ領域の回り込み

それでは、float の解除を行います。今回解除したいのはフッタからなので、フッタのブロックに clear を指定します。ついでに横幅、縦幅も指定しておきましょう。

```
div#footer {  
    border : solid 1px #000000 ;  
    clear : both ;  
    width : 798px ;  
    height : 50px;  
}
```

長くなってしまいましたが、これでようやく 2 カラムレイアウトは完成です。あとは細かいプロパティを指定し、内容を装飾していただくだけです。内容を適宜追加し、CSS を調整してみてください。

3-2 ボックスを中央に寄せる

皆さんが普段見ている Web サイトは殆どが中央に配置されていることが多いでしょう。

これも CSS で簡単に実現できます。

- ・ margin を auto にする

先ほどの2カラムレイアウトを見てみましょう。これは左に配置されていますね。Web ブラウザは左上を基点としてレイアウトを行うので、これは自然な表示です。これを中央に配置してみましょう。

CSS では簡単です。一番外側の<div id="box">に、

```
div#box {  
    margin : 0 auto ;  
}
```

と指定するだけです。この margin の指定の方法は省略形で、二つだけ値を指定した場合、一つ目が上下、二つ目が左右の値となります。この書き方もよく使われるので覚えておきましょう。

さてこの場合、二つ目の値は「auto」です。この値は特殊で、左右の横幅をブラウザ表示領域に合わせて自動的に算出してくれます。これにより、ボックスの中央寄せが可能となります。また、コンテンツ内等でブロックレベル要素を中央寄せしたときにも有効です（その場合、横幅が指定されていることが条件です）。

これもとても便利な手法ではありますが、やはり Internet Explorer のレンダリングモードによっては無効となる場合がある（付録参照）ので、注意が必要です。Internet Explorer 以外のブラウザであればこの手法で確実に中央寄せできるでしょう。

3-4 クロスブラウザ対策

冒頭でも述べましたが、クロスブラウザ（どの Web ブラウザで見ても全く同じ表示をさせる）を行うには少しハードルが高いのが現状です。使用するプロパティを決めてしまえばそれも可能ですが、やはりそれではデザインの表現力には限界があります。

- ・ CSS ハックを試してみる

「ハック」と聞くと悪い印象を持たれるかもしれませんが、決して悪意のあるものではありません。CSS のハックにおいては、「特定のブラウザしか解釈しない記述法で CSS を制御する」という意味で、決して悪意があって使用するわけではありません。CSS ハックは

技術や仕様として公に発表されているわけではありませんが、開発者たちがユーザビリティを損なわないために試行錯誤した結果、生み出された手法であると言えるでしょう。CSS Validator(CSS の記述に誤りがないかをチェックする機構)には通らないものが殆どですが、ユーザーが快適に見えるようにするのが最優先事項となるのです。しかしながら、表示が崩れるからといって、むやみに CSS ハックで解決しようとするのは好ましくありません。CSS ハックを使わずに解決できるのであれば、勿論それに越したことはありません。さらに、CSS ハックに頼ることは後々の新しいブラウザに対応していけなくなり、本当の意味での CSS の理解には遠くなる、というデメリットもあります。CSS ハックは「Web ブラウザ間のサポートの違いを埋める緊急策」という認識の下で使っていただきたいです。今後 Web ブラウザの開発が進み、どのブラウザでも同じ CSS をサポートするようになれば必要が無くなる技術なのです。

前置きが長くなりましたが、CSS ハックについて少しだけ説明します。本テキストでは何度も「Internet Explorer では～」と書いてきましたが、CSS において、Internet Explorer の CSS の解釈の方法、サポート状況は他のブラウザとは違うようです。今回は、Internet Explorer にのみ別の値をセットする方法をご紹介します。

現在、最も普及している Internet Explorer のバージョンは 7 です (8 もリリースされていますが、サポート状況は細かく公表されていない上、導入率は低いです)。困ったことに、Internet Explorer はバージョンによっても CSS のサポート状況が大きく違ってきます。ここでは主要ブラウザとして、バージョン 6 と 7 について説明します。

- ・ CSS ハックの使い方

CSS ハックは、まずは規定の CSS を指定し、後からその値を特定のブラウザだけ「上書き」する、というスタイルです。つまり、そのハックを解釈するブラウザだけが値を上書きし、その他のブラウザは解釈せず、規定の値を使う、という流れになります。上書きするので、当然 CSS ハックは規定の CSS よりも「後」に書く必要があります。

- ・ Internet Explorer バージョン 6 にのみ違う値を適用する

まずは色々解釈の違いがある Internet Explorer6 についてのみ特定のスタイルを記述する方法です。このよう書きます。

```
* html セレクタ {
    適用させたいプロパティ : 値 ;
}
```

子孫セレクタのような記述法ですね。ここで、先頭の「*(アスタリスク)」は「ユニバーサルセレクタ」と呼ばれ、これで指定されたものは全ての要素に対してCSSが適用されます。この意味は、「全ての要素の子孫要素である HTML 要素の子孫要素」となります。しかし、全ての要素とは HTML 要素も含まれますので、実質「* html」という指定はおかしい筈です。当然他のブラウザは解釈しないのですが、Internet Explorer 6はこのセレクタを解釈します。それが幸か不幸か、CSS ハックとして利用されるようになっていきます。このハックを先頭の「*」を「スター」とも呼ぶことから、「スターハック」呼ばれることもあります。CSS Validator には通りますが、これは Internet Explorer 7では無視されます。

- ・ Internet Explorer バージョン 7 にのみ違う値を適用する

続いてバージョン 7 にだけ有効な CSS ハックをご紹介します。まずはハックを見てください。

```
*:first-child+html セレクタ {
    適用させたいプロパティ : 値 ;
}
```

このようになります。このセレクタの意味は簡単に言うなら「全ての要素の一番初めに来る要素に隣接した html 要素」となりますが、前述のスターハックと同じく、この指定は本来はおかしいのです(html 要素はページ中に一つしかない筈です)。

ところが、この指定をすると Internet Explorer 7 だけが解釈するのです (DOCTYPE スイッチ[後述]が関係していると言われていきます)。バージョン 6 は無視します。このため、Internet Explorer 7 だけに有効な CSS の指定として利用されています。前述のスターハックと比較して、「ファーストスターハック」と私は呼んでいますが、実際にこれといった名前が付けられているわけではありません。

これら 2 つの CSS ハックを覚えておくだけでも、クロスブラウザでのデザインの再現性はかなり高まると思います。しかしながら、ワンポイントで使うようにする「緊急策」であることは忘れないようにしてください。

付録 : よく使われるプロパティ一覧

CSS を組む上で、頻繁に使用されるプロパティをまとめました。これだけ覚えれば良い、というものではありませんが、基本的なプロパティを理解し、その上で他のプロパティを覚えていく方が効率が良いと思います。ご参考にどうぞ。

プロパティ	効果	値 (単位等)
margin	上下左右に余白をつける。	一つの場合は上下左右、 二つの場合は上下/左右。 四つ指定の場合は左から 上/右/下/左。 単位は% / px / em 等。
padding	上下左右にスペースを作る。	一つの場合は上下左右、 二つの場合は上下/左右。 四つ指定の場合は左から 上/右/下/左。 単位は% / px / em 等。
font-size	フォントサイズの変更。	% / px / em / pt 等
font-weight	フォントの太さの変更。	300 ~ 900。 他には normal / bold 等。
color	フォントカラーの変更。	16進数表記 (推奨)、 または rgb 表記。
background-image	背景画像を指定。	url(画像へのパス)で指定。
background-color	背景色を指定。	16進数表記 (推奨)、 または rgb 表記。
line-height	行の高さを指定。	% / px / em 等
display	表示モードの切り替え。	block / inline 等。
width	ボックスの内容領域幅の指定。	% / px / em 等
height	ボックスの内容領域高さの指定。	% / px / em 等
position	ボックスの配置モードを変更。	relative(相対) / static (通常) / absolute(絶対) / fixed(固定)※
float	浮動ボックスに変換。	left または right
border	枠線の付与。	形式 太さ カラーの順 に指定できる (省略形)
text-align	テキストの揃え方の指定。	left / center / right /

	インライン要素に適用される(標準レンダリングモード)。	justify※
list-style	リストタグのキャプションの指定。	none / decimal 等
text-decoration	テキスト装飾の指定。	underline / none 等
clear	浮動ボックスの解除。	left / right / both

※ は Internet Explorer 6 では未対応。

レンダリングモードについて

Internet Explorer(以下、IE)では、HTML の記述方法によって標準モード (Standard)、後方互換モード(Quirks)という二種類のレンダリングモードを自動的に決定します。

このモードの違いによって、CSS の書き方も変わってくるので注意が必要です。

Firefox、Opera といったモダンブラウザは基本的に標準モードとでレンダリングを行いますので、やはり IE でも標準モードでのレンダリングを期待するほうが望ましいです。一方、後方互換モードは、旧バージョン (IE5.0 等以前のバージョン) との互換性を保つための仕様のように、このモードでクロスブラウザを実現させるのはとても困難です。

・ DOCTYPE スイッチ

では、このモードの決定はどのようにして行われるのでしょうか？それは、HTML ファイルの先頭に記述する「DOCTYPE 宣言」が関与しています。今までこの宣言について詳しく調べたことはあまり無いと思いますが、実はこの宣言は IE のレンダリングを決定付けるものでもあるのです。代表的な DOCTYPE 宣言の例を見てみましょう。

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

これは XHTML Transitional で記述する宣言の時に記述するものです。

さて、参考書等を見ていると、「XHTML の場合は先頭に XML 宣言が必要である」と書かれているものがあります。つまり、以下のような記述を例にしています。

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

実はこの場合、IE 6 は後方互換モードになります。「先頭に XML 宣言を行うと後方互換モードになる」と書かれている書籍も見受けられますが、厳密に言うと、実は「DOCTYPE 宣言より前に半角スペース以外の文字があると後方互換モードになる」のです。本来ならば XML 宣言は必要なのですが、ブラウザ間の誤差を少なくするために、XML 宣言はしないのが現在の主流となっています。

また、DOCTYPE 宣言内での「DTD 宣言」にも要因があります。先述では XHTML を例に挙げましたが、HTML4.01 で記述する場合に DTD 宣言がないと、やはり後方互換モードとなります。例は以下の通りです。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" >
```

DOCTYPE 宣言そのものが無い場合も後方互換モードとなります。

Dreamweaver では自動的にこの宣言を行ってくれるので楽ですが、一からテキストエディタで書く場合は注意してください。

- ・ 後方互換モードでの IE の挙動

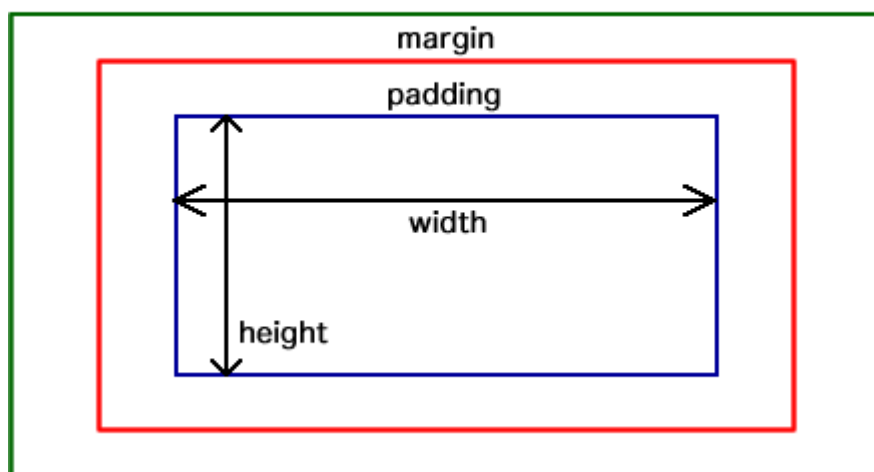
後方互換モードが悪者であるかの様には書きましたが、後方互換モード自体が悪いわけではありません。CSS のサポートが旧バージョンのレベルと互換を持ってしまうのが問題となります。では、主にどのような特徴があるのかを簡単に示します。

1. margin : auto は無効になる

「3-2 ボックスを中央に寄せる」での手法が使えなくなってしまいます。この場合、逆に text-align プロパティがブロックレベル要素にも有効になるので（これも本来ならば間違っただレンダリングです）、「text-align : center」と指定して中央寄せを行うこととなります。

2. 実際の表示領域の計算方法が変わってしまう

実際の表示領域



上図はボックスモデルの説明の時の図ですが、後方互換モードの場合、実際の表示領域の計算式が変わってしまいます。

標準モードの場合、横幅は、

$$\text{width} + \text{margin(左右)} + \text{padding(左右)} + \text{border(左右)}$$

で算出されます（これを content-box と言います）。しかし、後方互換モードの場合は、width で指定した値がそのまま実際の表示領域となります（これを border-box と言います）。つまり、内容領域の幅は

$$\text{width} - \text{margin(左右)} + \text{padding(左右)} + \text{border(左右)}$$

で逆算する必要があるので、クロスブラウザではとても扱いが困難になります。content-box の解釈が正しいので、やはり標準モードが推奨されます。

3 デフォルトの文字サイズが小さくなる

標準モードの場合、デフォルトの文字サイズは「medium」です。しかし、後方互換モードでは「small」となります。フォントサイズを%等で調整している場合は注意が必要です。

他にも色々解釈の違いはありますが、主に2の理由から後方互換モードは推奨されませ

ん。正しい CSS を適用させる場合は標準モードでレンダリングさせたほうが賢明です。

3. 最後に

CSS は、HTML やプログラムのように単体で動作するものではなく、どうしても Web ページ作成において重きを置かれないものである、と私は感じています。CSS は Web ページには今やなくてはならないものであるにも関わらず、専門家が居ない場合が多く、多くはコーダーやデザイナーが兼任したりして作成される場合が多いのではないかと思います（実際、専門家が必要なほど難しいものではないからかもしれません）。

しかし、もしも皆さんがこれから JavaScript 等を覚える機会を得て、さらに一步前進するようなら、CSS は絶対に知っておくべきものであると思います。HTML + CSS + JavaScript を学んだとき、皆さんはフロントエンドの専門家になれることでしょう。デザイナーの方であれば、あなた一人にフロントエンドを任せられますね。

さて、今後は CSS3 が勧告されようとしています。簡単に触れておきますと、角丸のボックスが作成可能になったり、モジュール化されたり、縦書きまで対応されたり、といったより高度な見た目が実現できるようです。しかしそれは、あくまで CSS2.1 が基盤となっており、それらを知って初めて CSS3 を学べるのではないかと思います。

これを機に、CSS は HTML の付加言語としてついでにやるものではなく、専門的なものとして興味を持っていただければ幸いです。

2009 年 5 月 23 日

杉本 吉章

