

- 2 デザイナーガイド -



2. デザイナーガイド

2.1 CSSの基本

ウェブページをデザインする場合、ページのスタイル(見栄え)を指定するには2つの方法があります。1つはタグで直接指定する方法で、例えば<body bgcolor="#FF0000"> という指定をすればページの背景は赤くなりますし、~ で囲んだ部分は文字が2段階大きくなります。

ソース

```
<body bgcolor="#FF0000">
<h1>題名</h1>
<p>本文<font size="+2">大きなテキスト</font>本文</p>
</body>
```

表示結果

題名

本文大きなテキスト本文

例 2.1 タグを使ったスタイルの指定

もう1つの方法が、CSS(カスケーディング・スタイルシート)を使う方法です。CSSは、HTML文書の構造と、スタイルを分離するためのものです。平たく言えば、CSSはHTMLのタグごとに、プロパティ(特性): 値; という形式でスタイルを指定するものです。CSSを使うことで、段落(pタグ)や見出し(h1~h6)に対しても背景や文字色の指定、文字の大きさなどといった細かい指定をすることができます。

もっとも簡単な方法は、各タグのstyle属性にCSSを指定することです。例えば、前の例の背景と文字の大きさの指定はCSSを使って次のように書くことができます。

```
<body style="background-color: #FF0000;">
<h1>題名</h1>
<p>本文<span style="font-size: x-large;">大きなテキスト</span>本文</p>
</body>
```

例 2.2 CSSを使ったスタイルの指定

CSSでは、fontのような、一部の文字のスタイルを指定するタグの代わりに、単に範囲を指定するためにあるspanタグを使います。また、例えばh1~h6のような段落を表すタグの代わりに(見出しではないが、単に字を大きくしたいといった場合)divタグを使います。

CSSを使う利点は、細かくスタイルを指定できるだけではありません。「セレクタ」という機能を使うことで、HTML文書とCSSを完全に分離することができます。以下の例は、class="red"という属性のあるタグの背景を赤くし、class="big"という属性のあるタグの文字を大きくするというCSSを使って、前の例と同じスタイル付けを実現するものです。

```
<head>
<style type="text/css">
  .red {
    background-color: #FF0000;
  }
  .big {
    font-size: x-large;
  }
</style>
</head>
<body class="red">
<h1>題名</h1>
<p>本文<span class="big">大きなテキスト</span>本文</p>
</body>
```

例 2.3 CSSを分離する

styleタグで囲った部分がセレクタを使ったスタイルの指定です。これがスタイルシートと呼ばれるもので、この部分だけ完全に別のファイル(.cssファイル)に分けることもできます。HTMLにどのスタイルシート・ファイルが適用されるかを指定するには、次の3つの方法があります。

1. linkタグを使う
2. @import指示子を使う
3. xml-stylesheet処理命令を使う

最もよく使われるのがlinkタグを使う方法で、次のようなタグをhead内に書いておきます。

```
<link rel="StyleSheet" type="text/css" href="スタイルシート・ファイルのURL">
```

例 2.4 linkタグによるスタイルシートの指定

もう1つの方法がスタイルシートの中からスタイルシート・ファイルを「インポート」する方法で、styleタグを使って次のように記述します。

```
<style type="text/css">
  @import url(スタイルとのURL);
</style>
```

例 2.5 @import指示子によるスタイルシートの指定

最後の、xml-stylesheet処理命令を使う方法は、主にXMLファイルのスタイルを指定するために使うものです。処理命令とは、XMLファイルに対してアプリケーションがどのような処理を行うかを指定するもので、HTMLファイルでは通常使われません。

```
<?xml-stylesheet type="text/css" href="スタイルシート・ファイルのURL"?>
```

例 2.6 xml-stylesheet処理命令によるスタイルシートの指定

このドキュメントでは、CSSの詳細については解説しません。CSSは一般のブラウザ(Internet Explorer, Netscape, Opera, Firefoxなど)でもサポートされており、[解説書が市販されています](#)。CSSJのサポートするCSSもページ分割などの機能を除けば、それらとほとんど共通するものです。

また、CSSJで帳票などの印刷物をレイアウトする際は、(Dreamweaverのような)CSSに対応した普通のウェブデザイン・ツールで構いません。

2.1.1 標準への準拠

CSSJがサポートするCSSのバージョンは2.1(CSS level 2 revision 1)です。ただし、双方向テキストなど、一部サポートしない機能があります。詳細は資料集を参照してください。

2.2 改ページ

2.2.1 改ページ機能の概要

CSSJが一般のブラウザと大きく異なるのは、改ページ機能です。CSSJはテーブルを使ってデザインされたようなウェブページを機械的に分割するのには適しません。ドキュメントや帳票類に対しては、可能な限り読みやすい分割を行います。

また、デザイナーが適切な改ページのタイミングを指定できるように、CSSで規定されている強制改ページ、改ページの抑制およびwindows, orphans(改ページの前後に必ず表示する行数)の制御を行うことができます。これにより、例えば見出しの直後でページが途切れてしまったり、1行だけしか印刷されないページができてしまったりといった、みっともない改ページのされ方を防ぐことができます。

2.2.2 強制改ページ

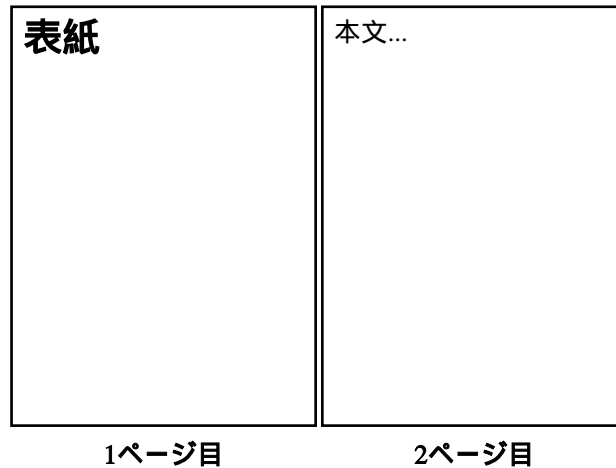
強制改ページは、文字通りデザイナーが指定した場所で強制的に改ページを発生させる機能です。用途としては、表紙の直後や、大見出しの前で改ページさせたり、納品書と領収証を別々のページで1つのPDFにまとめるといった具合です。

強制改ページはブロックの前か後に指定できます。ブロックの前の強制改ページの指定はpage-break-before: always;です。また、ブロックの後の強制改ページの指定はpage-break-after: always;です。

以下は強制改ページを使って表紙を作る例です。

```
<html>
<head>
  <title>ドキュメント</title>
</head>
<body>
  <h1 style="page-break-after: always;">表紙</h1>
  <p>本文...</p>
</body>
</html>
```

例 2.7 強制改ページ(ソース)



例 2.8 強制改ページ(表示結果)

また、単純に改ページするためではなく、改ページした直後のページが右になるか、左になるかを指定することができます。例えば、ドキュメント中の大見出しは必ず左ページに表示し、そのとき右ページは真っ白になることがある、といった具合です。

強制改ページの後のページが右になるか、左になるかを指定するには、page-break-beforeおよびpage-break-afterプロパティの値として、alwaysの代わりにleft(左ページにする場合)またはright(右ページにする場合)を指定します。

以下の例では、必ず右側になる中表紙を生成しています。

```
<html>
<head>
  <title>ドキュメント</title>
</head>
<body>
  <h1 style="page-break-after: always;">表紙</h1>
  <p style="page-break-after: always;">本文1...</p>
  <p>本文2...</p>
  <h1 style="page-break-before: right;">中表紙</h1>
</body>
</html>
```

例 2.9 空ページが生じるケース(ソース)

表紙	本文1...	本文2...
-----------	--------	--------

1ページ目

2ページ目

3ページ目

	中表紙
--	------------

4ページ目

5ページ目

例 2.10 空ページが生じるケース(表示結果)

2.2.3 改ページの抑制

CSSJでは、浮動体(フローティングボックス)、絶対位置ボックス、画像、テーブルのセルの中身は必ず改ページが抑制されます。改ページはそれに加えて、任意のブロック中での改ページを抑制することができます。例えば、重要なことが書かれた囲み記事や、複数の画像や文字で1つの図を構成しているといった箇所に適用すると便利です。

ブロックの改ページを抑制するには、`page-break-inside: avoid;`という指定を行います。

改ページ抑制されたブロックがページをはみ出す場合、ブロックごと次のページに送られます。また、段落自体がページの大きさより大きい場合、ブロックの途中では改ページされずページをはみ出します。はみ出しを防ぐためには、そのブロックが必ず1つのページに収まるようにしてください。

また、改ページの抑制はブロックの直前と直後に対しても指定することができます。例えば、見出しの直後や箇条書きの直前で改ページされるのは不恰好なので、これらの改ページを抑制したい場合があります。

ブロックの前後での改ページを抑制するには、`page-break-before`(ブロックの前)あるいは`page-break-after`(ブロックの後)プロパティの値に`avoid`を指定します。CSSJは改ページが抑制された箇所での改ページを避け、前後の何行かを必ず1つのページに含めるようにします。改ページが抑制されたポイントの前後の何行目まで改ページの抑制の効果があるかは、後で説明する`widows`,`orphans`プロパティに依存します。

なお、強制改ページや改ページの抑制をしない場合は`auto`と指定します。以下の例では、見出しの後の改ページを避けないように、`h1~h6`に対して`page-break-after: auto;`という指定をしています。

```
<html>
<head>
  <title>ドキュメント</title>
  <style type="text/css">
    h1,h2,h3,h4,h5,h6 {
      page-break-after: auto;
    }
  </style>
```

```

</head>
<body>
  <h2>見出し1</h2>
  <p">本文1...</p>
  ...省略...
  <h2>見出し2</h2>
  <p">本文2...</p>
</body>
</html>

```

例 2.11 見出しの後の改ページを避けなかったとき(ソース)

結果として、以下のように見出しの直後で改ページされるという好ましくない結果が起こりえます。

<p>見出し1 本文1... 見出し2</p>	<p>本文2...</p>
1ページ目	2ページ目

例 2.12 見出しの後の改ページを避けなかったとき(表示結果)

先のソースのautoの指定をavoidに変えると、改ページ直前の見出しを強制的に次のページに持ってゆきます。結果として、表示結果は以下ようになります。

<p>見出し1 本文1...</p>	<p>見出し2 本文2...</p>
1ページ目	2ページ目

例 2.13 見出しの後の改ページを避けたとき(表示結果)

なお、CSSJではHTMLのh1～h6要素がデフォルトで直後の改ページが抑制されています。

また、これらの指定は強制改ページと競合する場合があります。例えば、page-break-after: always;と指定された段落の直後に、page-break-before: avoid;と指定された段落がある場合です。このような競合が起こった場合、常に強制改ページが優先されます。つまり、このケースではpage-break-before: avoid;は無視されて改ページが発生します。

2.2.4 windows,orphans

ドキュメントや帳票が改ページされる時、最後のページに1行だけ印刷されて終わってしまうことがあります。これは好ましいことではありませんし、だからといってページのスペースが足りない以上、改ページしないわけにはいきません。このようなもどかしい状況を解決するためにあるのが、段落に対するwindowsとorphansの指定です。

windowsは改ページされた後のページに最低限表示されなければならない行数です。例えば、windows

が2の場合、以下の例では2ページ目に2行存在するので要件を満たしています。

(段落1)1行目... 2行目... 3行目... 4行目... 5行目... 6行目... 7行目... 8行目... (段落2)1行目... 2行目... 3行目...	4行目... 5行目...
1ページ目	2ページ目

例 2.14 windowsが2の場合

同じ文書でwindowsが3に指定されていればどうでしょうか？ この場合、以下のように前のページから次のページへ行を移動して、windowsを満たすようにします。

(段落1)1行目... 2行目... 3行目... 4行目... 5行目... 6行目... 7行目... 8行目... (段落2)1行目... 2行目...	3行目... 4行目... 5行目...
1ページ目	2ページ目

例 2.15 windowsが3の場合

orphansは逆に、改ページされる前のページに最低限残さなければならない行数です。行数がorphansに達していないのに、改ページをしなければならぬ場合、CSSJはいつそのことブロックを丸ごと次のページに移動しようとするでしょう。

また、windowsとorphansは競合することがあります。すなわち、windowsを満たすために前のページを行を移動するとorphansが満たされない場合です。このときは、ブロックごと次の行に移動することになります。

そもそもブロックに含まれる行が、windowsやorphansに満たない場合はどうでしょうか。このような場合、ブロックは前のページに残るか、全て次のページに移動されるかのどちらかです。

以下のような表示になるには、複数の場合が考えられます。windows、orphansが共に2であるなら、以下のような表示にはならないでしょう。なぜなら、段落2の4,5行目を次ページに移動するだけで要件を満たせるからです。orphansが4以上の場合、windowsが5以上の場合、あるいはwindows、orphansが共に3以上であれば段落2を丸ごと次ページに移動する以外に要件を満たす方法がありません。

(段落1)1行目...	(段落2)1行目...
2行目...	2行目...
3行目...	3行目...
4行目...	4行目...
5行目...	5行目...
6行目...	
7行目...	
8行目...	

1ページ目

2ページ目

例 2.16 段落2が丸ごと移動される

windows,orphansは、ブロック要素に対するwindows,orphansプロパティで指定することができます。このドキュメント(デザイナーガイド)のPDF版を注意深く読めば、灰色のブロックで表される「HTMLソース」が、内容が短い場合は改ページされず、内容が長い場合は改ページされていることに気づかれるでしょう。実は、灰色のコード・サンプルにはwindows: 10; orphans: 10;という指定がされています。「改ページされて欲しくないけど、全く改ページされないのは困る」というもどかしい状況に直面したら、windows,orphansを大きめに指定してみることをおすすめします。

2.2.5 ページ番号の表示

position: fixed;と指定されたブロックは、一般のブラウザでは画面上に固定されますが、CSSJでは各ページの同じ位置に再表示されます。さらに、拡張機能として、contentプロパティによる内容の生成と、カウンタの操作がページ毎に実行されます。これを利用して、ページ番号を振ることができます。

```
<style type="text/css">
#page-number {
  position: fixed;
  bottom: -1cm;
  text-align: center;
  width: 100%;
}
#page-number:before {
  counter-increment: page;
  content: counter(page);
}
</style>
<div id="page-number"></div>
```

例 2.17 各ページの下部中央にページ番号を振る

詳細は「[資料集](#)」の「[固定位置決めされるブロックの内容の再生成](#)」を参照してください。

2.3 テーブル

2.3.1 自動テーブルと固定テーブル

CSSJはCSS2.1の自動レイアウトテーブルと、固定レイアウトテーブルの両方をサポートしています。自動レイアウトテーブルは、テーブルの内容に合わせてテーブルや列の幅を調整するもので、一般のブラウザの多くが昔からサポートしているものです。自動レイアウトテーブルは、テーブルや列の幅を指定しなくても、大体丁度良いレイアウトに調整される反面、自動で調整された結果が、必ずしもデザイナーのイメージ通りにならない場合があります。また、テーブル全体の情報をもとにレイアウトが調整されるため、CSSJはテーブルの全ての内容を一旦読み込まなければならず、何十ページにも渡る大きなテーブルでは、パフォーマンスの問題が生じます。

```

<html>
  <head>
    <title>商品一覧</title>
  </head>
  <body>
    <table>
      <tr>
        <th>品名</th>
        <th>単価</th>
        <th>説明</th>
      </tr>
      <tr>
        <td>チェンソー</td>
        <td>38,900</td>
        <td>排気量50ccの強力エンジン搭載で重量3.5kgという軽量を実現。なおかつこの価格！
      </td>
      </tr>
      <tr>
        <td>卓上ボール盤</td>
        <td>28,000</td>
        <td>100Vの家庭用電源で使用可能。連続5時間の長期稼動にも耐えられるプロ仕様の逸品
        です。</td>
      </tr>
      <tr>
        <td>丸ノコ</td>
        <td>16,800</td>
        <td>ニッカド電池を使った充電式丸ノコです。</td>
      </tr>
    </table>
  </body>
</html>

```

例 2.18 テーブルの例

品名	単価	説明
チェンソー	38,900	排気量50ccの強力エンジン搭載で重量3.5kgという軽量を実現。なおかつこの価格！
卓上ボール盤	28,000	100Vの家庭用電源で使用可能。連続5時間の長期稼動にも耐えられるプロ仕様の逸品です。
丸ノコ	16,800	ニッカド電池を使った充電式丸ノコです。

例 2.19 自動レイアウトテーブル(表示結果)

固定レイアウトテーブルは、テーブルや列の幅を最初から指定するものです。テーブルの内容に関わらずレイアウトは固定されるため、デザイナーがテーブルのレイアウトを把握することが容易です。また、複数ページに渡るテーブルの場合も全体を読み込む必要がないため、大きなテーブルになるほど、自動レイアウトより高速に処理することができます。ただし、デザイナーはテーブルの内容がはみ出さないように注意して、テーブルのレイアウトを指定する必要があります。

固定レイアウトテーブルでは、各列の幅を指定しない限りは、内容とは無関係に均等に幅が割り振られます。

品名	単価	説明
チェンソー	38,900	排気量50ccの強力エンジン搭載で重量3.5kgという軽量を実現。なおかつこの価格！
卓上ボール盤	28,000	100Vの家庭用電源で使用可能。連続5時間の長期稼動にも耐えられるプロ仕様の逸品です。

丸ノコ	16,800	ニッカド電池を使った充電式丸ノコです。
-----	--------	---------------------

例 2.20 固定レイアウトテーブル(表示結果)

デフォルトでは自動レイアウトテーブルになります。固定レイアウトテーブルを作成するには、テーブルに対して `table-layout: fixed;` を適用してください。また、各列の幅を指定するには、最初の行のセルあるいはHTMLの `col` 要素の `width` 属性がCSSの `width` プロパティを使ってください。

2.3.2 テーブル中での改ページ

テーブル中で改ページできる位置は、テーブルの行間です。テーブルセル内では改ページされないの
で、テーブルセルは常にページの大きさより十分に小さくなるように気をつける必要があります。

テーブルに表題(caption)、ヘッダ(thead)、フッタ(tfoot)がある場合、それらはページ毎に再表示
されます。

```
<html>
<head>
  <title>テーブル</title>
</head>
<body>
  <table>
    <caption>表題</caption>
    <thead>
      <tr><th>ヘッダ1</th><th>ヘッダ2</th></tr>
    </thead>
    <tfoot>
      <tr><th>フッタ1</th><th>フッタ2</th></tr>
    </tfoot>
    <tbody>
      <tr><td>内容1-1</td><td>内容1-2</td></tr>
      <tr><td>内容2-1</td><td>内容2-2</td></tr>
      <tr><td>内容3-1</td><td>内容3-2</td></tr>
      <tr><td>内容4-1</td><td>内容4-2</td></tr>
      <tr><td>内容5-1</td><td>内容5-2</td></tr>
    </tbody>
  </table>
</body>
</html>
```

例 2.21 複数ページに渡るテーブル(ソース)

<table border="1"> <tr><th colspan="2">表題</th></tr> <tr><th>ヘッダ1</th><th>ヘッダ2</th></tr> <tr><td>内容1-1</td><td>内容1-2</td></tr> <tr><td>内容2-1</td><td>内容2-2</td></tr> <tr><td>内容3-1</td><td>内容3-2</td></tr> <tr><th>フッタ1</th><th>フッタ2</th></tr> </table>	表題		ヘッダ1	ヘッダ2	内容1-1	内容1-2	内容2-1	内容2-2	内容3-1	内容3-2	フッタ1	フッタ2	<table border="1"> <tr><th colspan="2">表題</th></tr> <tr><th>ヘッダ1</th><th>ヘッダ2</th></tr> <tr><td>内容4-1</td><td>内容4-2</td></tr> <tr><td>内容5-1</td><td>内容5-2</td></tr> <tr><th>フッタ1</th><th>フッタ2</th></tr> </table>	表題		ヘッダ1	ヘッダ2	内容4-1	内容4-2	内容5-1	内容5-2	フッタ1	フッタ2
表題																							
ヘッダ1	ヘッダ2																						
内容1-1	内容1-2																						
内容2-1	内容2-2																						
内容3-1	内容3-2																						
フッタ1	フッタ2																						
表題																							
ヘッダ1	ヘッダ2																						
内容4-1	内容4-2																						
内容5-1	内容5-2																						
フッタ1	フッタ2																						

1ページ目

2ページ目

例 2.22 複数ページに渡るテーブル(表示結果)

テーブル内では、行に対する `page-break-before` および `page-break-after` の指定ができます。これは
段落に対する指定と同様に、強制改ページと改ページの抑制を指定することが出来ます。例えば、帳票
なので明細の後に合計が表示される場合、合計欄の直前で改ページされてしまうことは避けたいもので

す。このような場合、合計欄の直前に改ページの抑制を指定することで見やすい改ページをすることができます。

```
<html>
<head>
  <title>テーブル</title>
</head>
<body>
  <table>
    <caption>表題</caption>
    <thead>
      <tr><th>ヘッダ1</th><th>ヘッダ2</th></tr>
    </thead>
    <tfoot>
      <tr><th>フッタ1</th><th>フッタ2</th></tr>
    </tfoot>
    <tbody>
      <tr><td>内容1-1</td><td>内容1-2</td></tr>
      <tr><td>内容2-1</td><td>内容2-2</td></tr>
      <tr><td>内容3-1</td><td>内容3-2</td></tr>
      <tr style="page-break-before: avoid;"><td>内容4-1</td><td>内容4-2</td></tr>
      <tr><td>内容5-1</td><td>内容5-2</td></tr>
    </tbody>
  </table>
</body>
</html>
```

例 2.23 4行目の直前の改ページを抑制(ソース)

表題	
ヘッダ1	ヘッダ2
内容1-1	内容1-2
内容2-1	内容2-2
フッタ1	フッタ2

1ページ目

表題	
ヘッダ1	ヘッダ2
内容3-1	内容3-2
内容4-1	内容4-2
内容5-1	内容5-2
フッタ1	フッタ2

2ページ目

例 2.24 4行目の直前の改ページを抑制(表示結果)

2.4 PDFへの変換

2.4.1 フォント

CSSJがサポートするフォントには、次のものがあります。

- 欧文基本14フォント
- 等幅明朝体、等幅ゴシック体
- TrueType外部フォント
- CFF/Type2埋め込みフォント

フォントは、font-family、font-weight、font-styleおよびfontプロパティによって指定されます。CSSJはこれらの指定の組み合わせに最も近いフォントを選択して利用します。

font-familyは大文字小文字を区別しません。また、空白文字やハイフンは無視されます。従って、font-family: "Times-Roman";という指定とfont-family: "TIMESROMAN";という指定は同じです。フォント・ファミリには正式なフォント名、大まかなファミリ名、あるいは別名(エイリアス)を指定することができます。大まかなファミリ名が指定された場合は、さらにfont-weight, font-styleを手がかりにフォントを選択します。

欧文基本14フォントのCourier, Helvetica, Timesには太字(bold)と斜体(italic)と両者を組み合わせたものが用意されています。font-weight: bold;およびfont-style: italic;という指定をすることでそれらのフォントを選ぶことができます。font-weightにさらに太い指定をした場合(font-weight: 900;など)、フォントの輪郭をプログラムで拡張し、太いフォントを表現します。また、font-styleにitalicあるいはobliqueと指定したにもかかわらず斜体のフォントが見つからない場合は、プログラムでフォントを傾けます。

欧文基本14フォント

欧文基本14フォントは、PDFにより最低限利用可能な14の書体です。これらのフォントは、外部フォントとして利用可能です。以下がフォントの一覧です。

表 2.1 欧文フォント

フォント名	ファミリ名	別名	太さ	書体
Courier-Bold	Courier	Courier-New	bold	normal
Courier-BoldOblique	Courier	Courier-New	bold	italic
Courier-Oblique	Courier	Courier-New	normal	italic
Courier	Courier	Courier-New	normal	normal
Helvetica-Bold	Helvetica	Arial	bold	normal
Helvetica-BoldOblique	Helvetica	Arial	bold	italic
Helvetica-Oblique	Helvetica	Arial	normal	italic
Helvetica	Helvetica	Arial	normal	normal
Symbol	Symbol		normal	normal
Times-Bold	Times	Times-New-Roman	bold	normal
Times-BoldItalic	Times	Times-New-Roman	bold	italic
Times-Italic	Times	Times-New-Roman	normal	italic
Times-Roman	Times	Times-New-Roman	normal	normal
ZapfDingbats	ZapfDingbats	ITC Zapf Dingbats	normal	normal

なお、プレビューの際は最も近い代替フォントが使われるため、表示されるフォントはPDFと一致しません。

等幅明朝体、等幅ゴシック体

これは、WindowsまたはMacOSの日本語環境ではほぼ同じように表示されるフォントです。どちらも外部フォントとなります。等幅明朝体は'Mincho'、等幅ゴシック体は'Gothic'というフォント・ファミリ名が割り当てられています。

TrueType外部フォント

TrueType外部フォントが利用可能かどうかは、プログラマによる[設定](#)により決まります。外部フォントが利用可能なときにfont-familyにインストール済みのフォントを指定すると、外部フォントとして出力PDFに反映されます。

外部フォントを使った場合、PDFにフォント自体は含まれないため、PDFを表示・印刷する環境に該当するフォントがインストールされている必要があります。また、使用できるフォントはTrueTypeだけです。

CFF/Type2埋め込みフォント

これは、CFF/Type2という形式に変換されてPDFに埋め込まれるフォントです。埋め込みフォントが利用可能かどうかは、プログラマによる設定により決まります。埋め込みフォントが利用可能なときにfont-familyにインストール済みのフォントを指定すると、出力PDFにフォントが埋め込まれます。埋め込みフォントを使った場合、出力されたPDFはどのような環境でも同じ事体で表示されます。

一般フォントファミリ

CSSの一般フォントファミリで指定する場合は、serifはMincho、sans-serifはGothicになります。その他の一般フォントファミリは全て和文ゴシック体になります。なお、互換性のために、和文明朝体には"MS-明朝"、和文ゴシック体には"MS-ゴシック"という別名が付けられています。

2.4.2 文書情報

PDFの文書情報のうち、作成時刻(CreationDate)、変更時刻(ModDate)はサーバーの時計を元に自動的に設定されます。生成プログラム(Producer)にはCSSJのプログラム名とバージョンが入ります。また、文書の題名(Title)はHTMLの<title>要素に記述されたものがそのまま使われます。

他の文書情報はHTMLの<meta name="名前" content="値">要素によって設定することができます。

表 2.2 meta要素による文書情報の指定

名前	PDFの属性名	説明
DESCRIPTION	Subject	文書に内容についての簡潔な説明。
KEYWORDS	Keywords	スペースまたはカンマ区切りで羅列した、文書の内容に関連するキーワード。
AUTHOR	Author	文書の作成者。
GENERATOR	Creator	HTML文書を生成したプログラム、エディタ、オーサリングツールなど。

なお、メタ要素のname属性は大文字小文字を区別しません。<meta name="KEYWORDS" content="ほうじ茶 Linux 旧ソ連">としても、<meta name="Keywords" content="ほうじ茶 Linux 旧ソ連">としても、PDF文書にキーワードが設定されます。

2.4.3 ブックマーク

ブックマーク生成オプションがtrueになっている場合、CSSJはHTMLのh1~h6(見出し)要素をもとにブックマークを生成します。

ブックマークの項目名には見出し要素の中に含まれるテキストが使われます。例えば、<h2>注意事項(死亡等重要な事故につながります)</h2> という見出しがあった場合、注意事項(死亡等重要な事故につながります)がブックマークの項目名となります。

見出し要素は、最も大きな見出しをh1とし、最も小さな見出しをh6として使用してください。抜けている見出しレベルがあっても(例えばh1を使わずh2から始めたり、h3を使わずいきなりh4とするなど)構いませんが、同じレベルに属する見出しは、必ず同じ見出し要素を使ってください。また、見出し要素を見出し以外の目的で(単に字を大きくするなど)使った場合も、ブックマークに加えられてしまうので注意してください。

2.4.4 ハイパーリンク

ハイパーリンク生成オプションがtrueになっている場合、ウェブへのハイパーリンクと、文書内リンクの両方をサポートします。HTMLのa(リンク)要素のhref属性が#で開始するか、あるいは"文書自身のURL#"で始まる場合、文書内リンクと見なされます。それ以外のリンクはウェブへのハイパーリンクとなり、おそらくユーザーがクリックするとブラウザが開きます。

普通のHTMLページと同じく、文書内リンクの#の後に続く部分はアンカー名となります。ターゲットとなるアンカーは、name属性付きのa要素で配置することができます。ユーザーが文書内リンクをクリックすると、ターゲットのアンカーがあるページが開き、所定の場所にスクロールします。