

CSS食わず嫌いに 効く本

TOMOKO HIRATA
@10tomokO



CSS 食わず嫌いに効く本

Tomoko Hirata 著

2018-10-08 版 ひよこ 10 発行

はじめに

はじめまして、tomoko (twitter: @10tomok0) です。今回は「CSS 食わず嫌いに効く本」をお手に取っていただきありがとうございます。

かなり挑戦的なタイトルになってしましましたが、筆者もつい3,4ヶ月前は CSS といえば Bootstrap などフレームワークを使ったり、CSS 職人さんが実装するものだと思っていたました。特にサーバーサイドエンジニアとしてずっとやってきている身としては、CSS ってどこからやっていいかわからないし、基本的な書き方は分かるけどそれ以上はセンスが必要なのでは（いつもダサくなるし）。。。上下中央に配置する方法をいつもググらないとわからないし。。。ちょっと遠慮したい。。。とかなり苦手意識を持っていました。そんな折、お仕事の関係でひとつのサイトをまるまるフルスクラッチで実装することになり、（苦手とかいってらんねえ...!）と腹をくくって半月ほど CSS を書いて書いて書きまったくところ、あれ、自分 CSS チョットデキルかも？ と錯覚するくらいには CSS のことを好きになっていました。もちろん、デザイナーさんが作ったデザインが元々素敵だったということもあるのですが、段々自分の思うとおりに画面が形になっていく感動は、サーバーサイドで無駄のないロジックが書けた時のとはまた違ったよろこびがあったのです...！

そんなわけで、筆者が開発した中でこれはよく使うな、この考え方を覚えておいた方がいいな、と思ったことを今回の本ではまとめてみました。サンプルも用意したので実際に色々触ってみてください。

最近は個人開発をして自分の Web サービスを作っている人や、個人サイトやポートフォリオでちょっとデザインにこだわりをもって作成したいという人も増えているかと思います。そういったときにこの本が CSS（再）入門のきっかけになったらいいなと思っています！

対象読者

- プログラミング言語を何かやったことがある
- なんとなく CSS に苦手意識がある
- CSS 職人レベルまでは必要ないけどいい感じの Web ページを作ってみたい

目次

はじめに	2
第 1 章 始める前に	5
1.1 免責事項	5
1.2 Sass を使う	5
1.3 (自分) ルールを決めておく	6
第 2 章 最初はボタンから	7
2.1 基本的なプロパティ	7
2.2 display:block/inline/inline-block	8
2.3 margin と padding	9
2.4 その他：リンクが不要な場合	10
第 3 章 文字をいろいろ装飾してみよう	12
3.1 基本的なプロパティ	12
3.2 フォントを変えてみる	12
3.3 position と疑似要素を組み合わせる	14
3.4 Flex と擬似要素を組み合わせる	16
第 4 章 好きな配置ができるようになろう	18
4.1 Flex と Grid	18
4.1.1 display:flex	18
4.1.2 display:grid	19
あとがき	22
著者紹介	23

第1章

始める前に

ここでは、実際に CSS を書いていく前に免責事項と開発を進める上ではじめにやっておいた方がいいことを書きました。

1.1 免責事項

この本では基本的な HTML や CSS の書き方やプロパティの説明はしていませんが、最低限の書き方や見方がわかれれば理解できるようにしています。

また、サンプルも CodePen 上になるべく作成するようにしていますので、実際の動きはそちらでもご確認ください。(サンプルの内容は本文と多少変わる場合があります)

1.2 Sass を使う

CSS を書く時に、そのまま生の CSS を書くよりも Sass を使うことをお勧めします。ここで Sass の詳細な説明は省きますが、Sass を使うことで CSS をプログラミングに近い書き方でかけるようになるため、JS や他のプログラミング言語を触ったことがある人にとってはとっつきやすくなるからです。

Sass の特徴についてざっくり説明します。

- プログラミングっぽくかける
 - 入れ子による構造化、共通化、変数の定義、演算ができます。
- 2種類の記法がある
 - Sass 記法と SCSS 記法がありますが、本書では CSS に近い書き方である SCSS を使います。
- コンパイルが必要
 - Sass/SCSS を CSS に変換できる環境が必要です。

1.3 (自分) ルールを決めておく

ふたつ目は、ルールを決めておく、ことです。

ここでいうルールとは、プロパティの記述の順番、単位 (px,em)、クラスの命名規則などです。

調べれば色々出てくるのですが、どれもこれが絶対正しい！ というものがなく、どの方針に沿って進めばいいんだろうと最初とても迷ってしまいました。ですが、ルールを考えることに意識を向けるあまり実装が進まないのは本末転倒です。そのため、ある程度自分ルールを決めておくと、途中で迷いがなくなりスムーズに進めることができます。(もちろん、複数人で開発するときは共通のルール付けが必要だと思います)

ちなみに私の場合はプロパティの記述についてはおそらく一番有名な mozilla.org Base Styles を参考にしました。ページは現在見ることができないのでアーカイブされている URL を載せておきます。

<https://web.archive.org/web/20160123184648/https://www.mozilla.org/css/base/content.css>

このルールは最新のプロパティについて乗っていないので、使うときは+aでざっくりと順番を決めていました。(よく使うものを書いてます)

- 大枠 (display 系 block/inline , flex, grid)
- 見た目 (overflow, visibility)
- 要素を形成するもの (width/height/margin/padding/border)
- 背景 (background/opacity)
- 色 (color)
- テキスト (font/text-decoration/text-alien/vertical-align)
- リスト・コンテンツ (list-style/content)
- 動きをつける (transition/animation)

また、単位については基本的に絶対指定 (px) で記述していました。相対指定 (em/rem) も方法もありますが、ベースのサイズを意識する必要があるので、避けました。(もちろんレスポンシブ等で便利な側面もいっぱいあるので、次のステップでは使いこなしていくたいと思っています)

第2章

最初はボタンから

なんで最初はボタン？ と思うかもしれません、個人的にボタンを作る過程で CSS のプロパティで重要な部分の理解が深まったので、なにからやっていいかわからない！ という人にはひとつ自分のお気に入りのボタンを作ってみることをお勧めします。

ここでいうボタンは a タグで作るリンクボタンを想定しています。

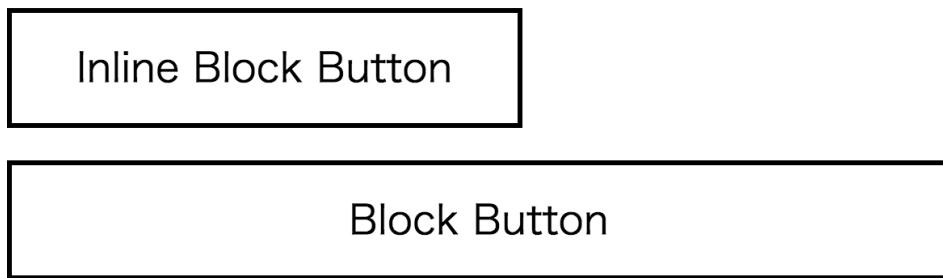
Web ページの大体のボタンはリンクボタンになっていることが多いので、今回は a タグをベースに作成しています。

もしリンク先のないボタンを作成したい場合は [リンクがいらない場合] を参照ください。

2.1 基本的なプロパティ

今回のボタンを作るときに使った主なプロパティです。

- display: block; 要素をブロック要素にします。
- display: inline-block; 要素をインラインブロック要素にします。
- box-sizing: border-box; width に padding や border の長さを含めます。
- width
- padding
- border



▲図 2.1 ボタン

▼ button.html

```
1: <a href="" class="link btn">Inline Block Button</a>
2: <a href="" class="link btn-block">Block Button</a>
```

▼ button.scss

```
1: .link {
2:   margin: 10px;
3:   padding: 15px 40px;
4:   border: 3px solid;
5:   background-color: #FFFFFF;
6:   color: #000000;
7:   font-size: 25px;
8:   letter-spacing: 0.03px;
9:   text-decoration: none;
10:
11:  &.btn {
12:    display: inline-block;
13:  }
14:
15:  &.btn-block {
16:    display: block;
17:    text-align: center;
18:  }
19: }
```

サンプルはこちら <https://codepen.io/tomoko523/pen/mGNraM>

いくつか CSS で詰まりがちなポイントがあるので、それについて説明していきます。

2.2 display:block/inline/inline-block

要素によって初期値の display は、block か inline が設定されています。
display プロパティを直接指定することで性質を変えることができます。（サンプルも元々

display:inline の a タグを、inline-block と block に変更しています。)

それぞれのざっくりとした特徴です。

- **block**

- div/p/h1~6... など
- 要素は横いっぱいに広がる（並べると縦に重なる、改行される）
- 高さ、幅が指定できる
- 余白が上下左右に指定できる

- **inline**

- span/a/img... など
- 要素の幅で留まる（並べると横に並ぶ、改行されない）
- 高さ、幅は指定できない
- 余白が左右のみ指定できる

そして inline-block は block と inline の mix のような位置付けです。

- **inline-block**

- 要素の幅で留まる（並べると横に並ぶ、改行されない）
- 高さ、幅が指定できる
- 余白が上下左右に指定できる

サンプルのボタンは、上の「Inline Block Button」は inline-block を指定しているので要素幅で収まっていますが、下の「Block Button」は block を指定しているので画面幅いっぱいに表示されています。また、どちらも後述の余白 (margin,padding) を指定できるようになっています。

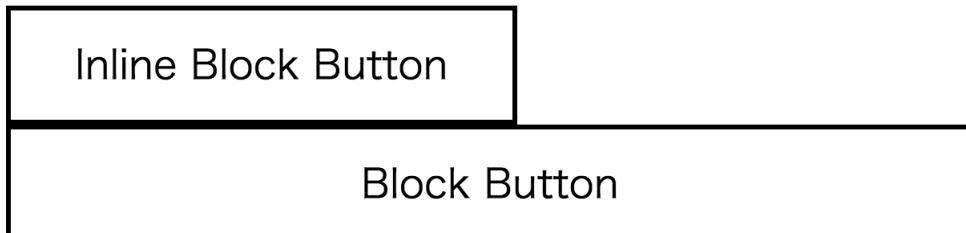
2.3 margin と padding

margin と padding、どちらも余白を指定するプロパティですが、どっちを使つたらいいのかわからなくなることはありませんか？ そういうときは、ボタンを作つてイメージを掴むのとわかりやすいのです。

言葉で説明すると、margin は要素の外側、padding は要素の内側の余白を作るプロパティです。

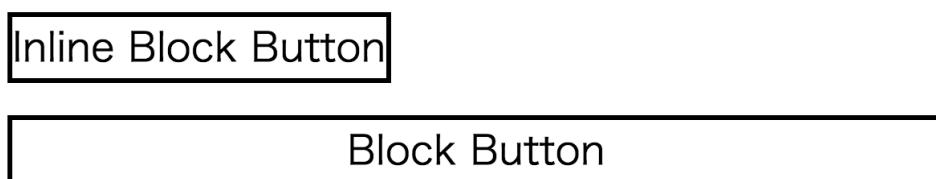
今回のボタンの場合、要素の外側（border の外側）に余白を持たせるために margin を、要素の内側（テキスト要素と border の間）に領域を持たせるために padding を指定しています。

ではまず、margin がないとどうなるのでしょうか…？



▲図 2.2 margin なし

ボタンの外側の余白が無くなってしまったので、くっついてしまいました。
では次に padding がないとどうなるのでしょうか…？



▲図 2.3 padding なし

内側の余白が無くなってしまったので、窮屈な感じになってしまいました。

margin と padding が作るそれぞれの余白のイメージはついたでしょうか？
margin と padding の違いは最初に引っかかる人が多い部分だと思いますので早めにイメージでしっかりとらえておくと役に立ちます！

2.4 その他：リンクが不要な場合

(イベントを発火させる等) リンクが不要な場合は a タグを span タグに置き換えてください。

ただこの場合、a タグが持っていた cursor : pointer; のプロパティが使えなくなるので、

プロパティを追加する必要があります。またリンクの装飾を消していた text-decoration: none; も不要になります。

- cursor: pointer; ホバーした時にカーソルマークを表示する
- text-decoration: none; 文字の装飾を無効にする (a タグの装飾を無効にする)

第3章

文字をいろいろ装飾してみよう

次は文字です！

文字は、大きさやフォントを変えるだけでもいろんな加工ができるのでぜひサンプルを色々動かして慣れてみてください。

3.1 基本的なプロパティ

文字を扱うときによく使うプロパティです。

- font-size 文字の大きさ
- font-style 文字のスタイル (normal 通常、italic 斜め、等)
- font-weight 文字の太さ
- line-height 文字の高さ
- letter-spacing 文字の間隔
- writing-mode: vertical-rl; 縦書き

3.2 フォントを変えてみる

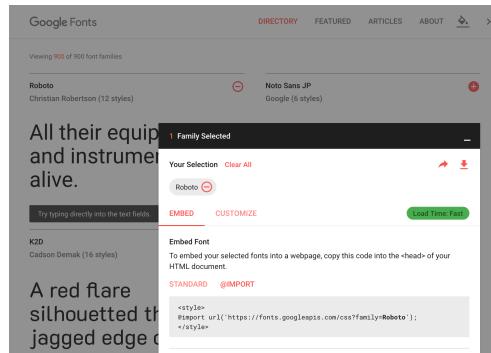
文字のフォントを変えてみましょう。もともとあるサイトやページのフォントを変えるだけでガラッとイメージが変わることが多いですし、フォントの印象はデザインにも多く影響するのですが、まずはフォントにこだわってみるのもいいと思います。

ちょっと試す場合は Google Fonts^{*1}がお勧めです。

色々なフォントを試すだけでも楽しい気分になってきますよ～！

^{*1} <https://fonts.googleapis.com/>

使い方も簡単で、各フォントの右上の+マークを押した後、下部に出てくる「Family Selected」を選択すると@IMPORTでCSS上で必要な記述が表示されます。



▲図 3.1 GoogleFonts

次の例では class="font"をつけたタグに対してフォントを設定します。

▼ use_font.scss

```

1: /* Google Font */
2: @import url('https://fonts.googleapis.com/css?family=Roboto');
3:
4: @mixin font-Roboto {
5:   font-family: Roboto;
6:   font-style: normal;
7: }
8:
9: .font {
10:   @include font-Roboto;
11: }
```

サンプルはこちら <https://codepen.io/tomoko523/pen/qJdYdV>
こちらはフォントを変えて背景をつけただけですが、ステッカー風な文字になりました。



▲図 3.2 ステッカー風

サンプルはこちら <https://codepen.io/tomoko523/pen/WavJpb>

3.3 position と疑似要素を組み合わせる

次はちょっと応用編です。:before,:after 擬似要素を使います。
:before は要素の直前、:after は要素の直後に内容（content）を追加します。これらが使えるようになるとぐっと表現の幅が広がりますので順番にさわってみましょう！
まずは、:before,:after 擬似要素の基本的な使い方です。



▲図 3.3 擬似要素

ここでは、テキストの前後にそれぞれ:before,:after で指定した内容が差し込まれていることがわかります。

▼ pseudo_elements.html

```
1: <span class="before-content">New Books</span>
```

▼ pseudo_elements.scss

```
1: .before-content {  
2:   &:before {  
3:     margin-right: 10px;  
4:     border: 4px solid #EE2020;  
5:     color: #FFFFFF;  
6:     background-color: #EE2020;  
7:     font-size: 14px;  
8:     content: 'NEW';  
9:   }  
10:  &:after {  
11:    padding-left: 4px;  
12:    content: '★';  
13:  }  
14: }
```

サンプルはこちら <https://codepen.io/tomoko523/pen/jePrRN>

では、これを応用してテキストの真下に内容を追加するにはどうしたらいいでしょうか？ これには position: relative,absolute; を使います。

Events

▲図 3.4 下線を追加

position は要素の配置する基準を示すプロパティです。relative は通常の表示位置からの配置、absolute は親要素からの配置を示します。そのため、absolute を指定したい要素の親要素に relative を指定すると、親要素を基準に動かすことができます (top:0; だと親要素とぴったり重なる)。

ここでは、本来要素の直後につくはずの after 要素を、親要素であるテキストを基準 (position:relative;) にして絶対指定 (position: absolute;) で場所を指定しています (left: 0;bottom: 0;)。(ちなみに:before を使っても結果は同じです)

文字だけではもしかしたら理解しにくい部分があるかもしれませんので、実際にサンプルの値を触って感覚を掴んでみてください。

▼ under_line.html

```
1: <span class="under-line">Events</span>
```

▼ under_line.scss

```
1: .under-line {
2:   position: relative; // ここを基準にする
3:   padding-bottom: 20px;
4:   font-size: 40px;
5:
6:   &:after {
7:     position: absolute; // 絶対指定で動かしたい要素
8:     left: 0;
9:     bottom: 0;
10:    width: 110%;
11:    height: 3px;
12:    background-color: #000000;
13:    content: "";
14:  }
15: }
```

サンプルはこちら <https://codepen.io/tomoko523/pen/Pyqgqx>

position を使いこなせるようになると（と言っても、基本的には上記の使い方で十分ですが）作れる文字の幅が広がります！

3.4 Flex と擬似要素を組み合わせる

最後に `display:flex` を使った例をみてみましょう。

—— そして物語は始まる

▲図 3.5 前に線を追加

テキストの前の線は before 要素です。

ここでは `position` で位置を変更するのではなく、`display:flex` で before 要素とテキスト要素を垂直方向に中央揃え (`align-items: center;`) で、水平方向に左揃え (`justify-content: flex-start;`) で指定しています。こうすることで、before 要素を高さ 2px の線としてテキストの前に配置することができます。

▼ before_line.html

```
1: <span class="before-line">そして物語は始まる</span>
```

▼ before_line.scss

```
1: .before-line {  
2:   display: flex;  
3:   align-items: center;  
4:   justify-content: flex-start;  
5:   letter-spacing: .09em;  
6:   font-style: italic;  
7:   font-size: 40px;  
8:   font-weight: 700;  
9:  
10:  &:before {  
11:    width: 10%;  
12:    height: 2px;  
13:    margin-right: 5px;  
14:    background-color: #000000;  
15:    content: "";  
16:  }  
17: }
```

サンプルはこちら <https://codepen.io/tomoko523/pen/VELxVg>

他にもいくつかサンプルを用意しました。

これ以外にも before,after 要素と position や flex を組み合わせると色々なアレンジができるので、ぜひ色々触ってこだわりの文字を作ってみてください。

サンプル

二重下線を作る

<https://codepen.io/tomoko523/pen/gBpexv>

縦書きと Flex

<https://codepen.io/tomoko523/pen/EdjRLM>

第4章

好きな配置ができるようになろう

4.1 Flex と Grid

折角要素がうまく作成できても配置がうまくできなかつたら困りますよね。

おそらく、複雑な配置をすることになったら、display:flex や display:grid が候補に挙がってくるかと思います。それぞれの細かい使い方については公式ドキュメントが一番かと思いますので、ポイントを絞って説明します。

4.1.1 display:flex

display:flex は 1 次元的な配置に向いているといわれています。

(縦や横) 一直線に要素を並べる時に有効です。文字をいろいろ装飾してみようの最後でも、複数の要素を並べるのに使っていましたね。

等間隔に並べるのも、justify-content: space-around でサクッと実装できます。

Story	Character	Cast/Staff	News	Release
-------	-----------	------------	------	---------

▲図 4.1 Flex

▼ flex_sample.html

```
1: <div class="links">
2:   <a href="">Story</a>
3:   <a href="">Character</a>
4:   <a href="">Cast/Staff</a>
5:   <a href="">News</a>
6:   <a href="">Release</a>
7: </div>
```

▼ flex_sample.scss

```

1: .links {
2:   display: flex;
3:   justify-content: space-around;
4:
5:   a {
6:     color: #000000;
7:     font-weight: 700;
8:     text-decoration: none;
9:   }
10: }

```

サンプルはこちら <https://codepen.io/tomoko523/pen/JmYPrz>

設定するプロパティについて簡単に説明します。

親要素

- display: flex;

子要素

- flex-direction 配置する方向（初期値 row）
- flex-wrap 子要素を折り返すかどうか（初期値 no-wrap）
- (flex-flow flex-direction と flex-wrap をまとめて指定）
- justify-content 水平方向の配置方法
- align-items 垂直方向の配置方法

使った感想としては flex-direction/flex-wrap は初期値で使うことが多く（左から右、折り返しなし）、justify-content/align-items をシーンによって使い分けるパターンが多い印象でした。

細かいプロパティの使い方は、最後の参考 URL をご確認ください。

4.1.2 display:grid

display:flex は 2 次元的な配置に向いているといわれています。

名前のとおり、Grid=格子を作成してそれぞれの要素をどこに配置するか指定することができます。

設定するプロパティについて簡単に説明します。

親要素

- display: grid;

- grid-template-rows/grid-template-columns グリッドの幅、高さ
- grid-row-gap/grid-column-gap グリッド間の余白
- grid-template-areas 後述

子要素

- grid-area 後述

個人的にこれは使うべき！と思ったのは `grid-template-areas` と `grid-area` です。これは、grid に配置する要素に名前をつけ、その配置方法を視覚的にわかりやすく指定することができます。これは、ぱっと見て配置がわかりやすくなるのもありますが、レスポンシブ対応等で画面サイズによって配置をガラッと変えたいときに非常に有効です。ここでは幅 767px までは、左上に text1、左下に text2、右半分に box の要素を配置しますが、幅が 767px 以下になると左上に text1、右上に text2、下半分に box の要素が配置されます。

… とても便利ですよね！！

私は、はじめてこれを知ってからレスポンシブ対応がとても楽になりました。ちょっと Grid 難しそうだなと思っていた方がいたら、ぜひこのプロパティを最初に使ってみてほしいです。

▼ grid_sample.scss

```
1: <div class="grid-sample">
2:   <span id="text1">Text1</span>
3:   <span id="text2">Text2</span>
4:   <span id="box"></span>
5: </div>
```

▼ grid_sample.scss

```
1: .grid-sample {
2:   display: grid;
3:   grid-template-areas: // 基本の配置
4:   "text1 box"
5:   "text2 box";
6:
7:   @media (max-width: 767px) {
8:     grid-template-areas: // 幅 767px 以下の時の配置
9:     "text1 text2"
10:    "box   box";
11:  }
12:
13:  #text1 {
14:    grid-area: text1;
15:  }
16:
```

```
17: #text2 {  
18:   grid-area: text2;  
19: }  
20:  
21: #box {  
22:   grid-area: box;  
23: }  
24: }
```

サンプルはこちら <https://codepen.io/tomoko523/pen/vV0qqR>
細かいプロパティの使い方は、最後の参考 URL をご確認ください。

あとがき

「CSS 食わず嫌いに効く本」いかがでしたでしょうか？

CSS ちょっと書いてみようかな...?という気持ちが少しでも生まれていたら嬉しいです。私自身まだまだ CSS に関しては学ぶことが多いため、今回はとにかく基本を押さえて昔の私のような初心者が挫折しそうなポイントを中心に執筆してみました。もし同じような立場の方がいて、少しでも助けとなれば幸いです！

苦手なものが得意になる瞬間ってちょっといいものです。

謝辞

多忙の中、私のふんわりした要望から素敵な表紙・裏表紙を作成いただきました kaori masuda さん、また、同時進行で執筆していた本の執筆者の masanori hori さん、お二方には大変感謝しております。また、事前のサークルチェックも大変執筆の励みになり（半分プレッシャーで）書き切ることができました。ありがとうございます。

参考 URL

CSS リファレンス

- <https://developer.mozilla.org/ja/docs/Web/CSS/Reference>
- https://developer.mozilla.org/ja/docs/Web/CSS/CSS_Flexible_Box_Layout
- https://developer.mozilla.org/ja/docs/Web/CSS/CSS_Grid_Layout

サルワカ HTML&CSS 入門 Web デザインをイチから学ぼう

- <https://saruwakakun.com/html-css/basic>

Codepip

- <https://flexboxfroggy.com/#ja>
- <http://cssgridgarden.com/#ja>

著者紹介

Tomoko Hirata

SIer で 5 年 + Web エンジニア もうすぐ 2 年目。

担当範囲はバックエンド、インフラをベースにフロントサイドもやるなんでも屋さんになります。

コーヒーと旅行と川と黄色くて丸いものが好きです。

ちなみに本書執筆中はレビュースタアライトにどハマリしてました。香子ちゃん推し。

Twitter @10tomok0

Github @tomoko523

blog <http://tomoko-tsubasa.hateblo.jp/>

CSS 食わず嫌いに効く本

2018年10月8日 技術書典5版 v1.0.0

著者 Tomoko Hirata

編集 Tomoko Hirata

発行所 ひよこ10

(C) 2018 ひよこ10



TOMOKO HIRATA
@1OtomokO