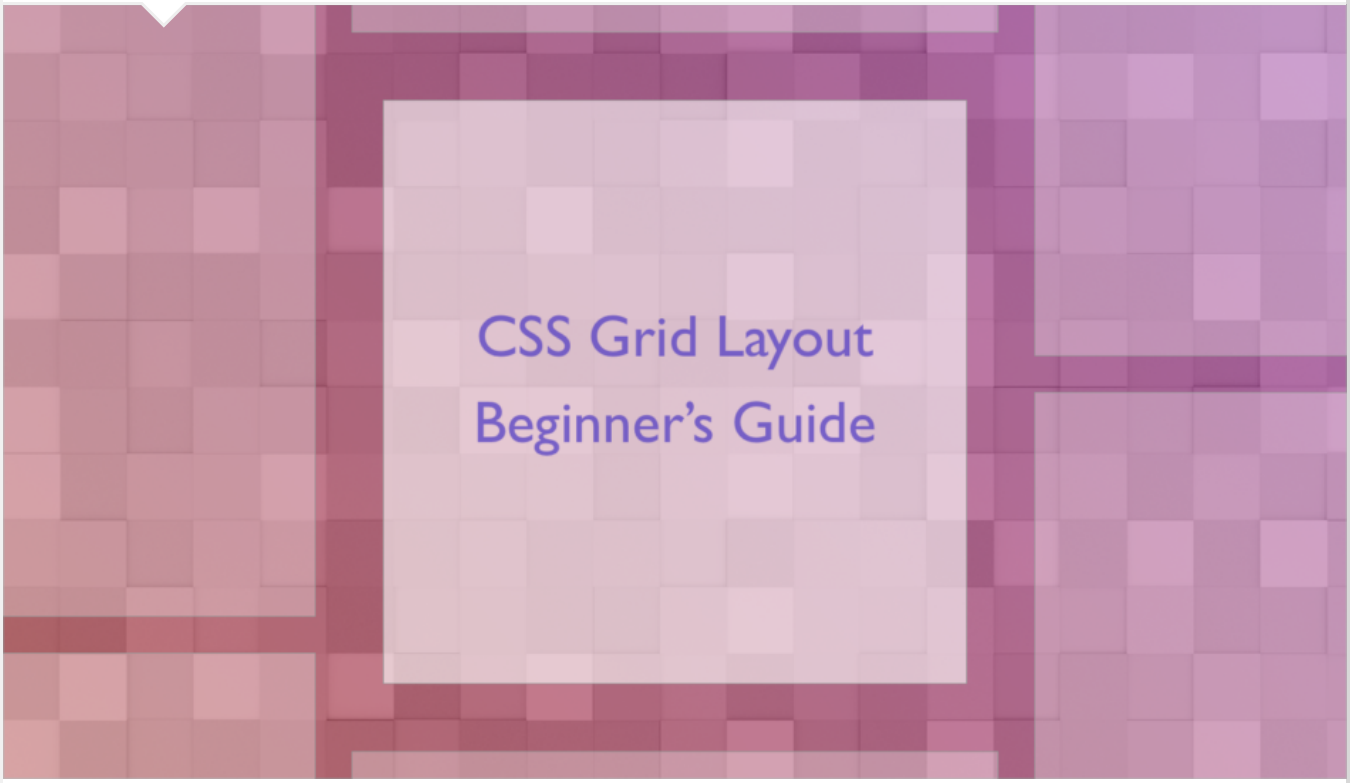
一番分かりやすいCSS Grid Layoutの使い方ガイド



CSS Grid Layoutは、CSSによるレイアウト手法の1つです。様々なレイアウトをこれ1つで実装することができます。

ただ、覚えなければならないことも多く、初心者の方にとっては少し難しく感じるかもしれません。

そこで、CSS Grid Layoutの重要なポイントに絞って、分かりやすく解説したいと思います。

もう迷わない！CSS Flexboxの使い方を徹底解説

CSS Flexboxは、CSSによるレイアウト作成でよく使われるCSSのレイアウト手法です。 レイアウトを作成する方法は他にもCSS Grid Layoutや、inline-blockを使用する方法...

Web Design Trends

INDEX

CSS Grid Layoutとは？

CSS Grid Layoutとは、**CSSでレイアウトを組む時に使われる主要な方法の1つ**です。

現在、CSSでレイアウトを組むときの方法は下記の2つが主要な方法です。

- CSS Grid Layout
- CSS Flexbox

最近では、CSS Grid LayoutもCSS Flexboxもそれぞれのブラウザである程度問題なく表示できるようになっており、一般的に使われるように

なっています。

どちらか片方を使えれば問題ないという訳ではなく、それぞれにメリットとデメリットがあります。2つのレイアウト手法の特徴を理解した上で、場面によって使い分けられるようどちらも身に付けておくようにした方がいいでしょう。

Grid LayoutとFlexboxの違い

Grid LayoutとFlexboxは、よく比較される存在ではあるものの、実際には使うべきタイミングや得意なことが異なります。

2つのレイアウト手法のおおまかかな違いは、下記の通りです。

Grid Layout	Flexbox
HTMLがシンプルに記述できる	HTMLの入れ子構造が複雑になりがち
アイテムのサイズは「コンテナ」で指定する	アイテムのサイズは「各アイテム」で指定する
複雑なレイアウトに向いている （2次元のレイアウト）	シンプルな横並びやタイルレイアウトに向いている （1次元のレイアウト）
グリッド間の幅を指定するプロパティがある	グリッド間の幅を指定するプロパティが無い ※marginなどで幅を作ることは可能

上記で紹介したのはほんの一例ですが、大雑把に説明するとこのようになります。

Grid Layoutを使用した場合は、**HTMLの記述が短くなる**というのは1つの特徴ですね。また、Grid Layoutは二次元、Flexboxは一次元というのもよく言われている言葉なので覚えておいた方がいいでしょう。

CSS Grid Layoutのブラウザ対応状況

CSS Grid Layoutは、最新のChromeやSafariでは動作しますが、**IEをサポート対象とする際は注意が必要**です。

IE11	ベンダープレフィックスが必要
IE10以前	一部プロパティが非対応

上記のようにIE11以降のみがサポート対象とする場合はベンダープレフィックスを付けるようにして、IE10以前のバージョンも保証するのであればGrid Layoutは使わない方がいいでしょう。

また、IE11ではベンダープレフィックス付きであれば基本的に問題ありませんが、一部特定の場合に不具合が発生することもあるため、必ず表示の確認をするようにしましょう。

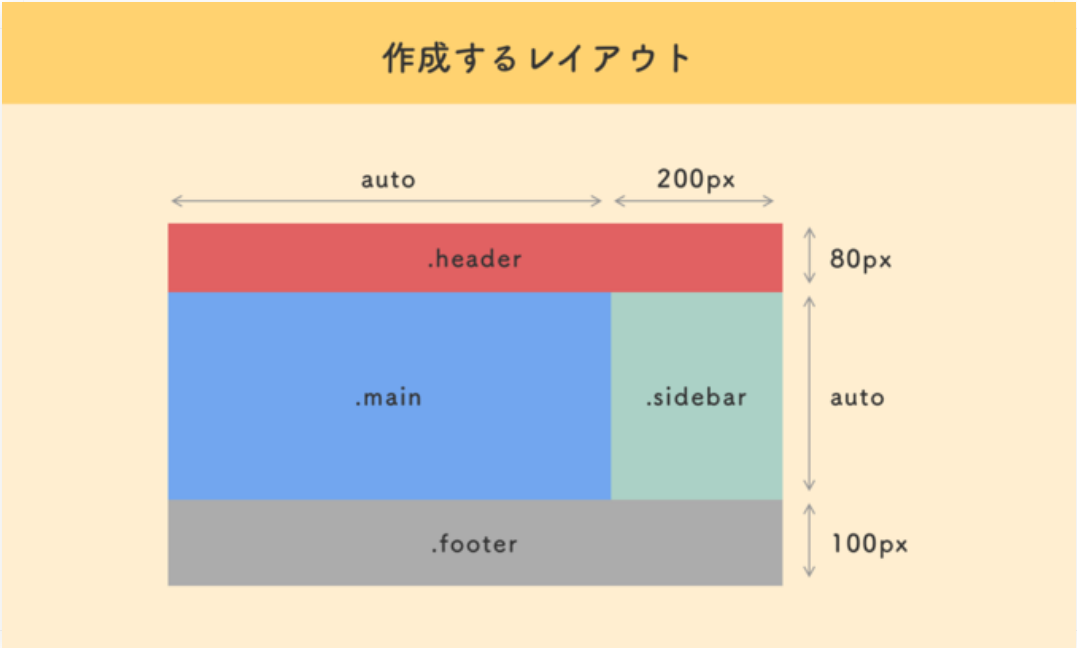
EdgeはGrid Layoutに対応しているので、問題なく使用することができます。

Grid Layoutの使い方

では、ここからGrid Layoutの使い方についてご説明していきます。

Grid Layoutを組む際は、それぞれの要素を囲む「Gridコンテナ」と、グリッド内に配置される「Gridアイテム」によって構成されます。この呼び方は一般的に使われているので、ぜひ覚えておいてください。また、単に「コンテナ」と「アイテム」と呼ばれることもあります。

例えば、下記のようなレイアウトを組む場合を考えます。



このとき、HTMLのコードは下記のようになります。

```
<div class="container">
  <header class="header">...</header>
  <main class="main">...</main>
  <aside class="sidebar">...</aside>
  <footer class="footer">...</footer>
</div>
```

上記のように、ベースとなるcontainerクラスの「Gridコンテナ」の中に、header, main, sidebar, footerクラスの「Gridアイテム」が配置さ

れています。

Grid Layoutを実装する場合は、主に下記の3つのステップで記述を行います。

1. コンテナにdisplay: gridを指定する
2. コンテナにgrid-template-rows、grid-template-columnsを指定してグリッドのサイズを決める
3. 各アイテムにgrid-row、grid-columnで表示する位置を決める

下記で、それぞれのステップの記述方法をご紹介していきます。

基本プロパティ（コンテナ編）

まずは、コンテナに記述する下記ステップの1番目と2番目についてご説明していきます。

1. **コンテナにdisplay: gridを指定**
2. **コンテナにgrid-template-rows、grid-template-columnsを指定してグリッドのサイズを決める**
3. 各アイテムにgrid-row、grid-columnで表示する位置を決める

Grid Layoutの適用 | display: grid;

まずは、コンテナに`display: grid;`を指定してGrid Layoutを適用させましょう。

CSSのコードは、下記のようになります。

```
.container {
  display: grid;
}
```

グリッドのサイズ | grid-template-rows, grid-template-columns

次に、グリッドのサイズを指定するプロパティです。`grid-template-rows`と`grid-template-columns`の2つがありますが、それぞれ下記

のとおりです。

grid-template-rows	グリッドの横方向のサイズを指定するプロパティ
grid-template-columns	グリッドの縦方向のサイズを指定するプロパティ

上記に掲載したレイアウトを作りたい場合、CSSのコードは下記のようになります。

```
.container {
  display: grid;
  grid-template-rows: 80px auto 100px;
  grid-template-columns: auto 200px;
}
```

それぞれ、横方向にグリッドが3つ、縦方向にグリッドが2つありますが、`gird-template-rows`と`grid-template-columns`によってそれぞれのグリッドのサイズを定義しています。

また、autoを指定すると状況に応じてコンテンツ量やその他のグリッドのサイズに合わせてグリッドのサイズが調整されます。

ここまででコンテナ側のCSSのコードは完了です。

基本プロパティ（アイテム編）

コンテナの基本プロパティで、下記の3つのステップのうち2つご説明し

ました。

1. コンテナにdisplay: gridを指定
2. コンテナにgrid-template-rows、grid-template-columnsを指定してグリッドのサイズを決める
3. **各アイテムにgrid-row、grid-columnで表示する位置を決める**

次に、3番目のグリッドの表示位置を指定する方法をご説明します。

グリッドの位置の指定 | grid-row, grid-column

grid-row	縦方向のグリッドの位置を指定するプロパティ
grid-column	横方向のグリッドの位置を指定するプロパティ

上記のようなレイアウトを組む場合、CSSのコードは下記のようになります。

ここで、box4は下記のようなコードになります。

```
.header {
  grid-row: 1 / 2;
```

```
  grid-column: 1 / 3;
}

.main {
  grid-row: 2 / 3;
  grid-column: 1 / 2;
}

.sidebar {
  grid-row: 2 / 3;
  grid-column: 2 / 3;
}

.footer {
  grid-row: 3 / 4;
  grid-column: 1 / 3;
}
```

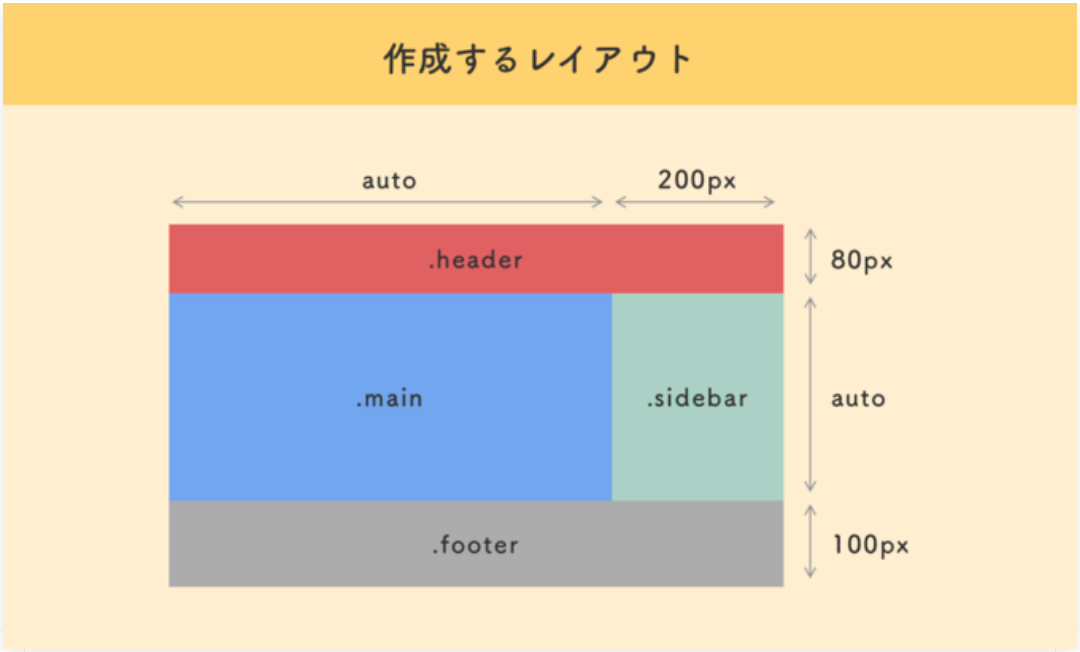
例えば、`.main`は、「縦方向には2番目の線から3番目の線まで、横方向には1番目の線から2番目の線までの領域に配置する」というコードになります。

`.header`や`.footer`のように、複数のグリッドに対して要素を配置することも可能です。

最初はこの記述方法が少し気になるかもしれませんが、Grid Layoutを使っていく内に慣れてきます。

コンテナでサイズを指定するプロパティが`grid-template-rows`、`grid-template-columns`と複数形であるのに対し、アイテムの位置を指定するプロパティは`grid-row`、`grid-column`と単数形となっているので間違えないようにしましょう。

これで、下記のようなレイアウトを作ることができます。



改めて、下記の3つのステップはGrid Layoutを実装する時のの基本的な流れなので覚えておきましょう。

1. コンテナにdisplay: gridを指定
2. コンテナにgrid-template-rows、grid-template-columnsを指定してグリッドのサイズを決める
3. 各アイテムにgrid-row、grid-columnで表示する位置を決める

その他のよく使うプロパティ（コンテナ編）

上記でご説明したのは、必須となるようなプロパティです。次に、細かなデザインの調整を行うためのプロパティをご説明します。

まずは、コンテナに適用するCSSプロパティです。

名前付きのグリッド定義 | grid-template-areas

上記では、`grid-column`と`grid-row`を使ってアイテムが配置される場所を指定しましたが、`grid-template-areas`を使えば、グリッド領域に名前を付けて配置することも可能です。

上記の例で紹介したレイアウトを作りたい場合、コンテナのCSSに下記のように記述します。

```
.container {
  display: grid;
  grid-template-rows: 80px auto 100px;
  grid-template-columns: auto 200px;
  grid-template-areas:
    "box1 box1"
    "box2 box3"
    "box4 box4";
}
```

ここで、新しく`grid-template-areas`というプロパティが登場しましたが、グリッドのそれぞれの領域に対して名前を定義しています。複数の領域にまたがってグリッドを配置したい場合は、同じ名前を定義することで配置することができます。

そして、それぞれのアイテムのCSSは下記のように記述します。

```
.header {
```

```
  grid-area: box1;
}

.main {
  grid-area: box2;
}

.sidebar {
  grid-area: box3;
}

.footer {
  grid-area: box4;
}
```

このように、grid-templateで名前を指定しておけば、`grid-row`と`grid-column`の2つの記述を簡略化することができるため、直感的なコードを書くことができます。

上記では分かりやすいように「box1」などの名前を付けていますが、実際は「header」のように何が配置されるか分かるように名前を付けるのがおすすめです。

また、`grid-template-rows`、`grid-template-columns`、`grid-template-areas`の3つはショートハンドを使って1つにまとめることも可能です。

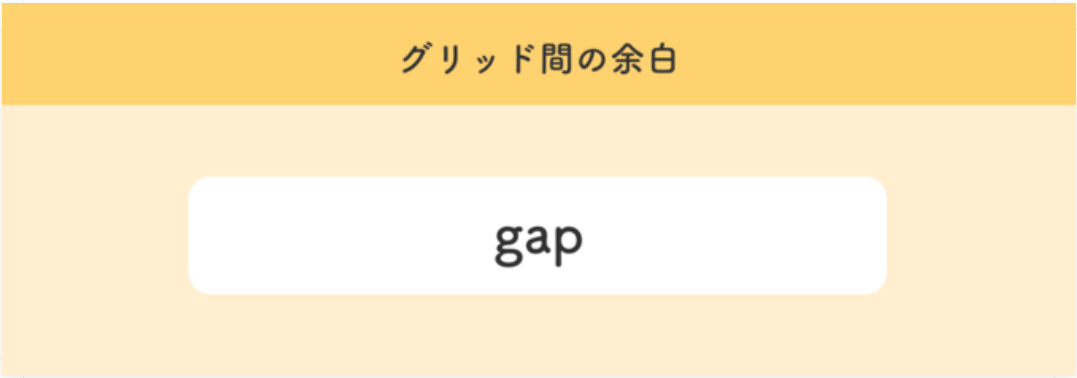
```
grid-template:
    "box1 box1" 80px
    "box2 box3" auto
    "box4 box4" 100px /
    auto 200px;
```

grid-templateを使えばCSS Gridが簡単に扱える

CSS Gridの使い方はよく分からないけど、Flexboxの方が簡単そうだからFlexboxを使っているという方も多いのではないでしょうか。 確かに、CSS GridはFlexboxと比べると少し扱...

Web Design Trends

グリッド間の余白 | gap



最初にFlexboxにはグリッド間の余白を定義するプロパティが無いが、Grid Layoutにはあるとご説明したのはこれらのプロパティのことです。

gap	縦方向と横方向のグリッド間の余白を定義するプロパティ
row-gap	縦方向のグリッド間の余白を定義するプロパティ
column-gap	横方向のグリッド間の余白を定義するプロパティ

縦方向には20px、横方向には10pxの余白を取りたい場合、コードは下記のようになります。

```
.container {
  display: grid;
  row-gap: 20px;
```

```
  column-gap: 10px;
}
```

縦横どちらも10pxの余白を取りたい場合は、下記のように記述できます。

```
.container {
  display: grid;
  gap: 10px;
}
```

横方向の位置 | justify-content

`justify-content` は、**コンテナ内におけるグリッドの横方向（インライン軸）の位置を指定するプロパティ**です。

`justify-content` に指定できる値には、下記のようなものがあります。

start	左寄せ
center	中央寄せ
end	右寄せ
stretch	軸の領域いっぱいにグリッドを広げる

space-between	均等配置 / 左右の余白無し
space-around	均等配置 / 左右の余白ありはアイテム間の半分
space-evenly	均等配置 / 左右の余白はアイテム間と同様

それぞれの表示がどのようになるのかは、MDNのページで確認できるので、気になる方は下記のページでチェックしてみてください。

> justify-content | MDN web docs

縦方向の位置 | align-content

`align-content` は、**コンテナ内におけるグリッドの縦方向（ブロック軸）の位置を指定するプロパティ**です。

`align-content` に指定できる値には、下記のようなものがあります。

start	上寄せ
center	中央寄せ
end	下寄せ
stretch	軸の領域いっぱいにグリッドを広げる
space-between	均等配置 / 上下の余白無し
space-around	均等配置 / 上下の余白ありはアイテム間の半分

space-evenly	均等配置 / 上下の余白はアイテム間と同様

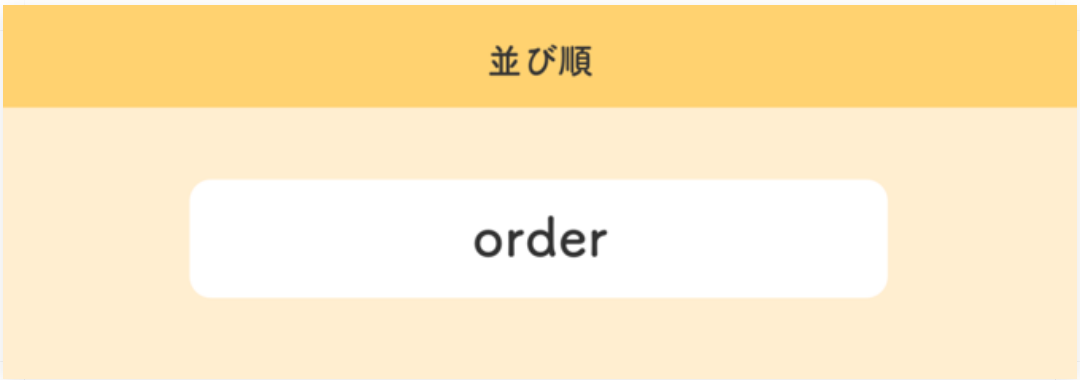
こちらも、MDNのページでそれぞれの値を適用した時の表示を確認することができます。

> align-content | MDN web docs

その他のよく使うプロパティ（アイテム編）

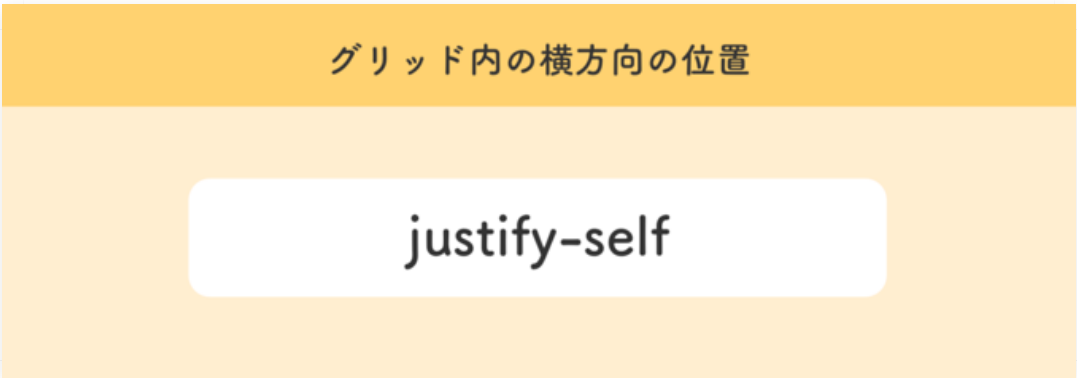
次に、アイテムに使用するプロパティのうち、覚えておいた方がいいプロパティをご紹介します。

並び順 | order



`order`は、アイテムの並び順を指定するプロパティです。レスポンシブデザインで画面サイズによって要素の表示順を変更したいような場合は、`order`プロパティを使って並び順を変えることができます。

横方向の位置 | justify-self



`justify-self`は、**各グリッド内の横方向の位置を指定するプロパティ**です。正確には軸方向の位置を指定するプロパティなので、`grid-auto-flow: column`を指定した場合は、縦方向の位置を指定するプロパティとなります。

コンテナ側で紹介した`justify-content`は、**グリッド間の位置を指定**するプロパティであるのに対し、`justify-self`は、**グリッド内における要素の位置**を指定するプロパティなので混同しないように注意してください。

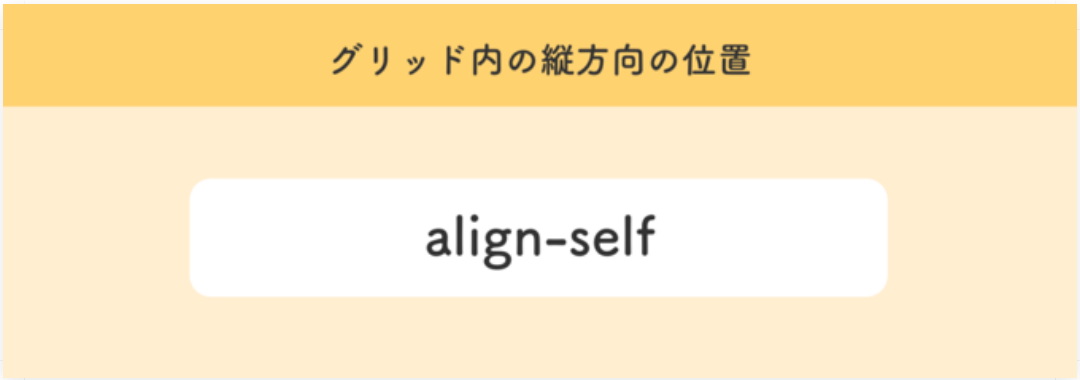
start	左寄せ
center	中央寄せ
end	右寄せ
stretch	軸の領域いっぱいにグリッドを広げる

なお、コンテナ側で`justify-items`プロパティを指定することで、すべてのアイテムに`justify-self`を記述した時と同じ挙動になります。

下記のページで、それぞれの値の挙動を確認することができます。

> **justify-self | MDN web docs**

縦方向の位置 | align-self



`align-self` は、各グリッド内の縦方向の位置を指定するプロパティです。正確には軸に対して垂直方向の位置を指定するプロパティなので、`grid-auto-flow: row` を指定した場合は、横方向の位置を指定するプロパティとなります。

コンテナ側で紹介した `align-content` は、グリッド間の位置を指定するプロパティであるのに対し、`align-self` は、グリッド内における要素の位置を指定するプロパティなので混同しないように注意してください。

start	上寄せ
center	中央寄せ
end	下寄せ
stretch	軸の領域いっぱいにグリッドを広げる

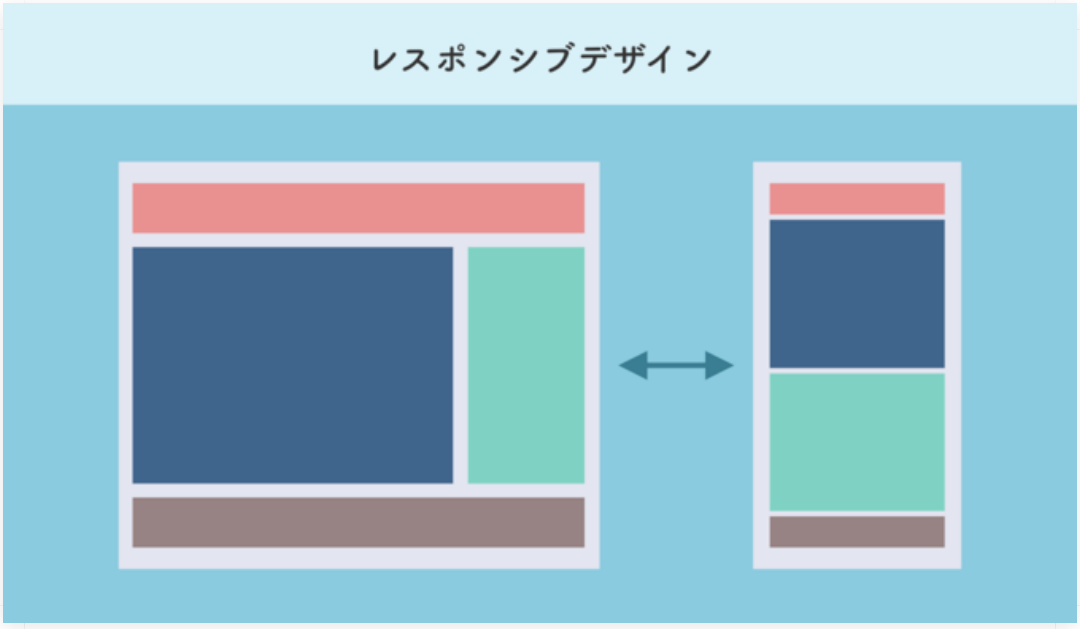
なお、コンテナ側で `align-items` プロパティを指定することで、すべてのアイテムに `align-self` を記述した時と同じ挙動になります。

> align-self | MDN web docs

実践テクニック

ここまでよく使うプロパティについてご説明してきました。次に、実際にCSS Grid Layoutを適用する場合によく使われるコードを紹介していきます。

レスポンシブデザインにする



まずは、Grid Layoutを使って上記のようなレスポンシブデザインを実装する方法をご説明します。

レスポンシブデザインを実装する方法はいくつかありますが、個人的には`grid-template`を使う方法がコードの記述量が少なく済むためおすすめです。

まずは、下記のようなHTMLを用意します。

```
<div class="container">
    <header class="header"></header>
    <main class="main"></main>
```

```
    <aside class="sidebar"></aside>
    <footer class="footer"></footer></div>
</div>
```

CSSのコードは、下記のようになります。

```
.container {
  display: grid;
  grid-template-areas:
    "header header"
    "main sidebar"
    "footer footer";
  grid-template-rows: 80px auto 100px;
  grid-template-columns: auto 200px;
}

.header {
  grid-area: header;
}

.main {
  grid-area: main;
}

.sidebar {
  grid-area: sidebar;
}

.footer {
  grid-area: footer;
}
```

ここまでは上記でご説明したように`grid-template-areas`を使ったGrid Layoutのコードですが、ここに**メディアクエリを使って下記のようにコードを追記します**。

```
@media screen and (max-width: 600px) {
  .container {
    grid-template-areas:
      "header"
```

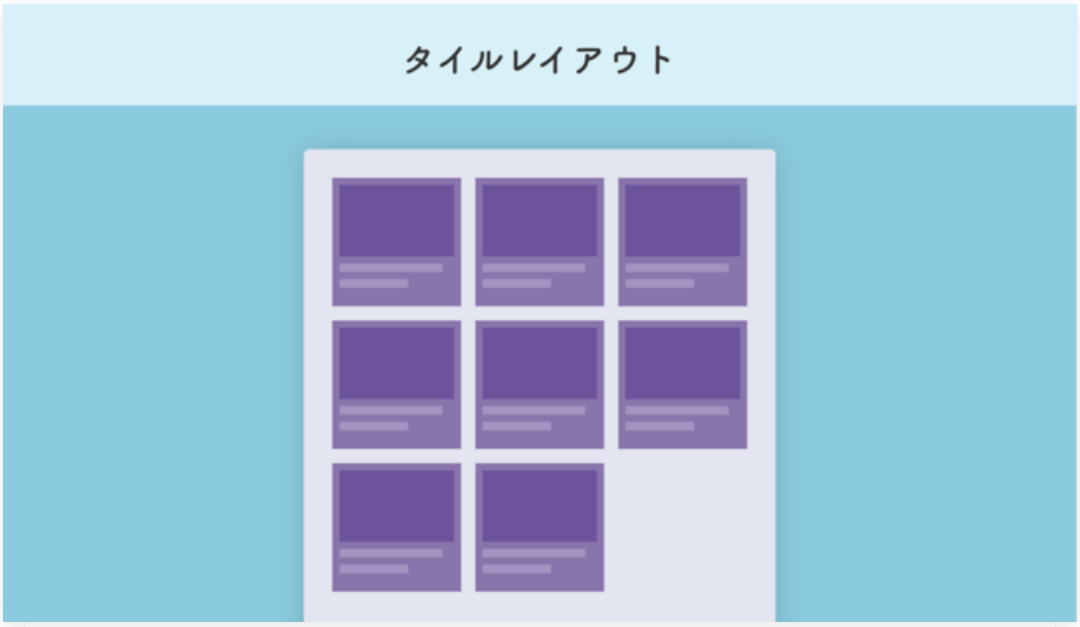
```
      "main"
      "sidebar"
      "footer";
    grid-template-rows: 80px auto auto 200px;
    grid-template-columns: auto;
  }
}
```

上記のコードにより、画面の横幅が600pxよりも小さい時にグリッドを縦に並べることができます。

このように、Grid Layoutでレスポンシブデザインを実装する場合には、メディアクエリを使って画面サイズごとにグリッドの並びを再定義することによって実現することができます。

また、`grid-template`を使ってコードを記述することによって、レスポンシブデザインを適用する時もコンテナ側のコードだけ追記すれば済むため、コード量を減らすことができるというメリットがあります。

タイルレイアウトを作る



タイルレイアウトは、デザインでよく見られるレイアウト手法です。スペースを有効活用することができ、画面の中で多くの情報を分かりやすく表示することができます。

まずは、下記のようなHTMLを用意します。

```
<div class="container">
  <div class="item">...</div>
  <div class="item">...</div>
  <div class="item">...</div>
  <div class="item">...</div>
  <div class="item">...</div>
</div>
```

CSSコードは下記のようになります。

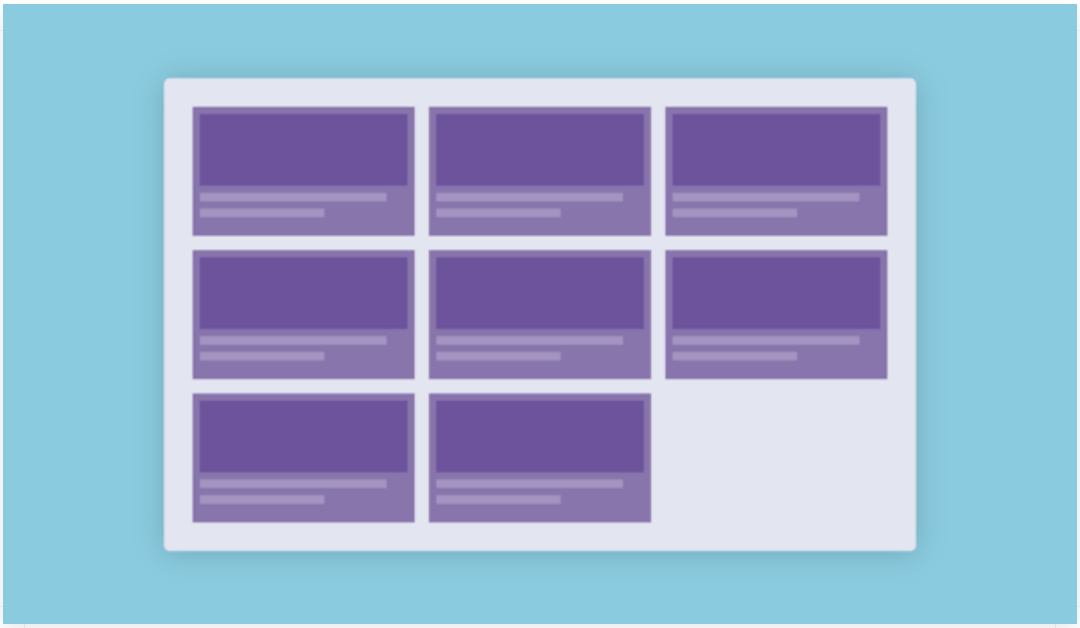
```
.container {
  display: grid;
  grid-template-columns: repeat(3, 180px);
  gap: 20px;
}
```

`grid-template-columns: repeat(3, 180px);`は、「180pxのグリッドを横に3つ並べる」という記述になります。

グリッドの数が4つ以上ある場合は、自動的に折り返しされるため、それぞれのアイテムをタイル状に配置する事が可能です。

あとは、アイテムに対して適切なスタイルを適用してデザインを整えれば簡単にタイルレイアウトを作成することができます。

✔ グリッドの横幅を画面サイズにあわせて調整する

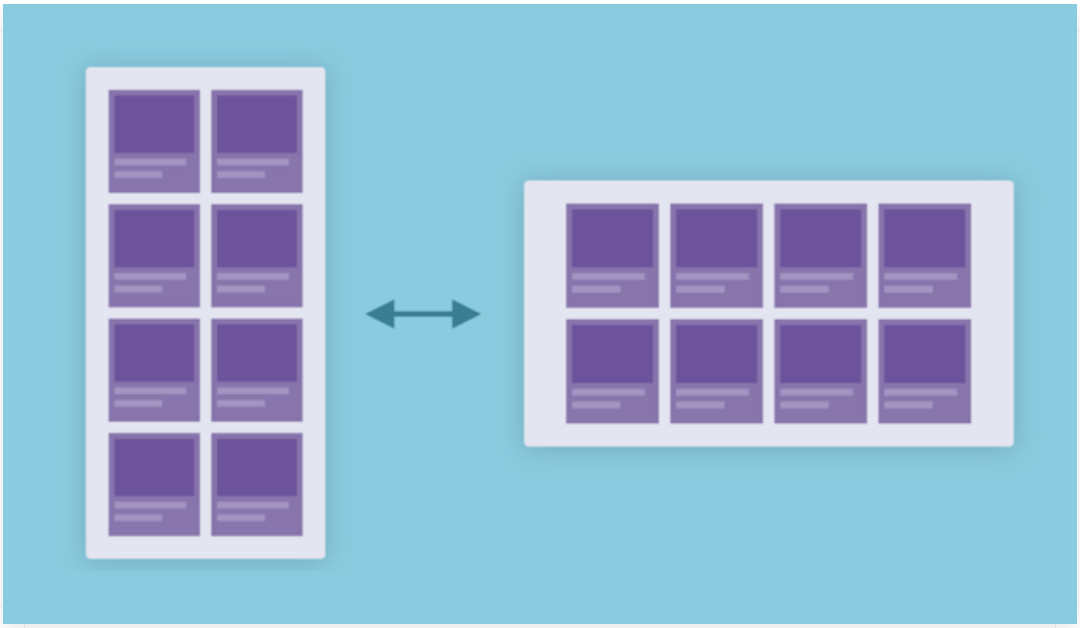


タイルレイアウトを組む場合、画面サイズに合わせて各グリッドの横幅を調整したいケースがよくあります。

その場合は、下記のようにグリッドのサイズを固定値で指定するのではなく、「1fr」や「auto」などの値を指定することで実装することが可能です。

```
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}
```

✔ 横並びのグリッドの数を画面サイズに合わせて変化させる



上記のように、画面サイズに合わせて横に並べるグリッドの数を変化させたい場合は、下記のように記述することで実現可能です。

```
.container {
  display: grid;
  grid-template: repeat(auto-fill, 200px);
}
```

repeatの1つめの値に`auto-fill`を指定することで、**画面サイズに合わせて横に並べるグリッドの数を変化させる**ことができます。

✔ グリッドの横幅に最小値と最大値を指定する

グリッドの横幅に最小値と最大値を指定したい場合、下記のように記述することで実現することができます。

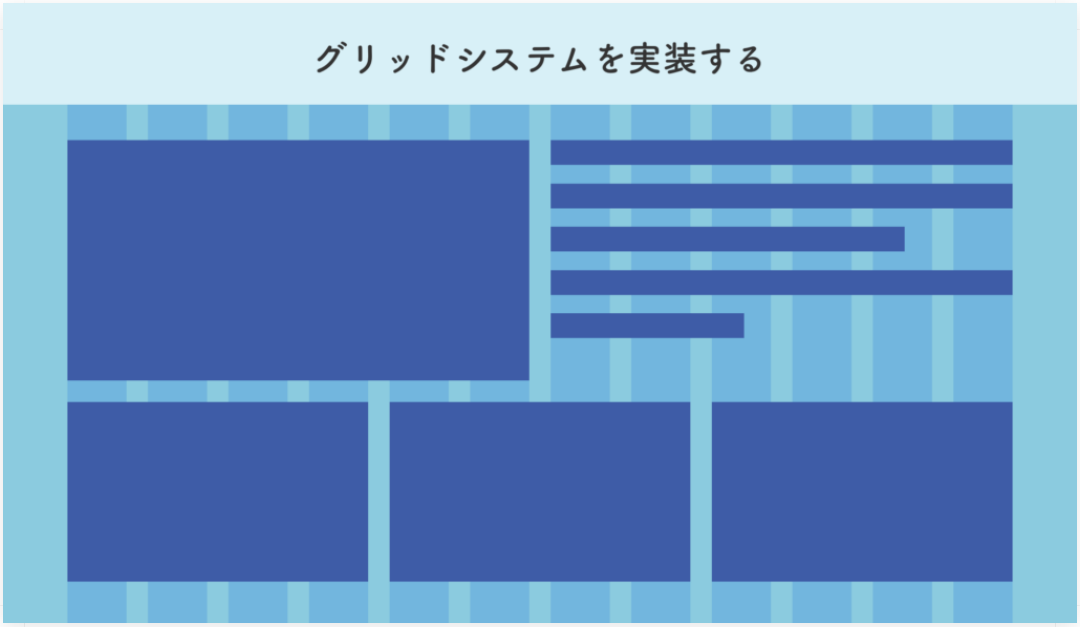
```
.container {
  display: grid;
  grid-template-columns: repeat(4, minmax(160px, 240px))
}
```

`grid-template-columns: repeat(4, minmax(160px, 240px));`の`minmax(160px, 240px)`は「最小160px、最大240pxのグリッドを作る」という意味になります。

repeatの1つ目の値に「4」が指定されているので、「グリッドを横に4つ並べて、画面サイズに合わせて各グリッドを最小160px、最大240pxの範囲で変化」することができます。

また、minmaxの最小値と最大値のいずれかに`1fr`などと指定することで、片方のみ指定することもできます。

グリッドシステムを実装する



デザインやCSSの記述では、画面を横向きにn分割してレイアウトを組むことがよくあります。CSSフレームワークの「Bootstrap」なども横向きに12分割してレイアウトを組みますね。

Grid Layoutを使用する場合、下記のように記述することでそれを実現することができます。

```
.container {
  display: grid;
  grid-template-columns: repeat(12, 1fr);
}
```

上記の`grid-template-columns: repeat(12, 1fr);`の部分が「グリッドを横向きに12分割する」という記述になります。10分割にしたい場合は、`grid-template-columns: repeat(10, 1fr);`となります。

あとは、コンテンツに合わせて`grid-template-rows`や、それぞれのグリッドを配置することでグリッドシステムを使ったレイアウト実装を行うことができます。

まとめ

最初はGrid Layoutを難しく感じるかもしれませんが、慣れてしまえば様々なレイアウトを簡単に記述することができるため、とても頼もしい存在です。

コーディングを勉強したての方は、Flexboxを使えるようになったら次にGrid Layoutを覚えることで、CSSでのレイアウトの実装で困ることはほとんどなくなるはずです。

もしFlexboxにまだ触れたことが無い方は、ぜひFlexboxにも挑戦してみてください。

もう迷わない！CSS Flexboxの使い方を徹底解説

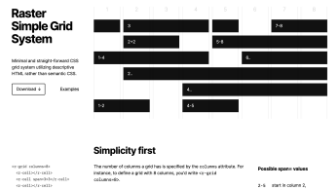
CSS Flexboxは、CSSによるレイアウト作成でよく使われるCSSのレイアウト手法です。 レイアウトを作成する方法は他にもCSS Grid Layoutや、inline-blockを使用する方法...

Web Design Trends

おすすめの記事

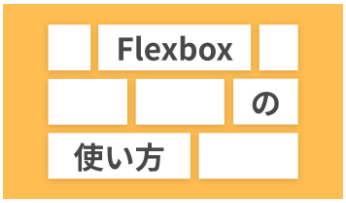
CSSで要素を横並びにする方法のメリット・デメリットまとめ

grid-templateを使えばCSS Gridが簡単に扱える



CSS Grid Layoutを使った便利なテクニックやツールなど

Flexboxが使えるおすすめのCSSフレームワークまとめ



もう迷わない！CSS Flexboxの使い方を徹底解説

2020年版：CSS Grid Layoutの対応ブラウザやベンダープレフィックスについて解説！

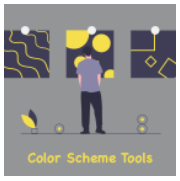
2020年5月13日

Web制作

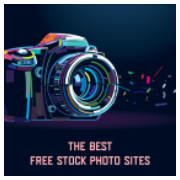
Previous post

Next post

POPULAR



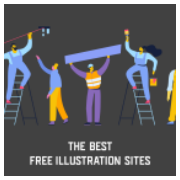
【2024】おしゃれな配色パターンが見つかる！カラーパレット人気ツール30個まとめ



【2024年版】無料で商用利用可能！フリー画像・写真素材サイトのおすすめ12選

【2023年版】おすすめのリセットCSSまとめ！基本知識と使い方を解説

【2024】無料で使えるフリーアイコン素材サイト30個まとめ【商用利用OK】



【2024】ほんとに無料？フリーイラスト素材サイト40個まとめ【商用利用OK】

各種SNSのシェアボタン設置用URLまとめ！サンプルコードも

もう迷わない！CSS Flexboxの使い方を徹底解説

Material Icons（Google Font Icons）の使い方を解説！表示されない時に確認すべきポイントも

2023年のトレンドカラーは「ビバ マゼンタ」。鮮やかで力強い華やかなレッドに

コピペで使える！CSSグラデーションジェネレーター20個まとめ

RECOMMENDED

【2023年最新】WordPressの使い方を徹底解説！初心者向けのおすすめ設定も

初心者でもOK！WordPressサイトをたった10分で公開する手順まとめ

NEW

【2024年】SEOに強いおすすめWordPressテーマまとめ

おしゃれなフリー素材サイトのジャンル別まとめ！登録不要で商用利用もOK

【WordPress】クラシックエディタとブロックエディタの違いや切り替え方法を解説！

AI搭載のUIデザインツールMotiffとは？プロンプトでUIデザインが生成可能

静的サイトジェネレーターminista（ミニスタ）とは？特徴や魅力を解説！

BACK TO HOME

WEB DESIGN TRENDS

Web Design Trends(ウェブデザイントレンド)は、Webに関わる全ての人のためのメディアです。ウェブサイトの制作やデザインに役立つ情報、最新トレンド、チェックすべきサービスなどを配信しています。

LATEST POSTS

【2024年】SEOに強いおすすめWordPressテーマまとめ

おしゃれなフリー素材サイトのジャンル別まとめ！登録不要で商用利用もOK

【WordPress】クラシックエディタとブロックエディタの違いや切り替え方法を解説！

AI搭載のUIデザインツールMotiffとは？プロンプトでUIデザインが生成可能

静的サイトジェネレーターminista（ミニスタ）とは？特徴や魅力を解説！

SOCIAL

Twitter（@TrendWebDesign）

LINK

プライバシーポリシー