

「え？！それ今ではCSSだけでできるの！？」
驚きの進化を遂げたモダンCSS

自己紹介

- 名前：西 悠太
- 所属：株式会社ダイニー
- TypeScriptが大好きです



今日のゴール

「へー！今のCSSってそんなことができるんだー」

と思って帰ってもらう

ちょっと質問です 🙋

以下の機能、どうやって実装しますか？

- 親要素のサイズに応じてレイアウトを変える
- フォームの入力状態に応じて親要素のスタイルを変更
- スクロールに連動したプログレスバー
- ポップアップやツールチップ

今こう思ってますか？

- めんどくさい処理だなあ...
- JavaScriptでしかできないなあ...
- まさかCSSでできるの？！

実は...全部CSSだけでできます

2019-2025年でCSSは劇的に進化しました

今日は「昔はJavaScript必須だったけど、今はCSSだけでできる」機能を紹介します

(＊一部ブラウザAPIやHTMLを含む＊)

1. レスポンシブデザインの新常識

Container Queries（コンテナクエリ）

こんな経験ありませんか？

サイドバーとメインエリアで同じカードコンポーネントを使いたい

でも...

- サイドバーは幅300px
- メインエリアは幅800px

同じブレイクポイントじゃ対応できない！

メディアクエリは画面全体のサイズしか見れないから、配置場所によって異なるレイアウトが必要な時に困る...

従来の解決方法：JavaScript

ResizeObserverで親要素のサイズを監視して クラスを付け替える...

```
const resizeObserver = new ResizeObserver(entries => {  
  entries.forEach(entry => {  
    const width = entry.contentRect.width;  
    entry.target.classList.toggle('small', width < 400);  
  });  
});
```

でもこれって：

- パフォーマンスへの影響
- 複雑なイベント管理
- CSSとJSの依存関係

もっとシンプルにできないの？

今はCSSだけで解決

Container Queries：親要素のサイズでスタイルを変える

```
/* 親要素をコンテナとして定義 */
.card-container {
  container-type: inline-size;
}

/* コンテナのサイズに応じてスタイルを変更 */
@container (max-width: 400px) {
  .card {
    flex-direction: column;
  }
}
```

これだけ！

サイドバーでは縦並び、メインでは横並び 場所に応じて自動的にレイアウトが変わる！

2. 親要素を操る魔法のセレクタ

:has() セレクタ

フォームでこんな実装してませんか？

「入力エラーがあるフィールドの 親要素を赤く表示したい」

よくある要件ですね？

でもCSSでは子要素は選択できても 親要素は選択できなかった...

従来の解決方法：JavaScript

入力が変わるたびに親要素を探してクラスを付ける

```
input.addEventListener('blur', () => {  
  if (input.invalid) {  
    input.closest('.form-group').classList.add('error');  
  }  
});
```

面倒だし、動的なフォームだと管理が大変...

今はCSSだけで解決

:has()セレクトタ：子要素の状態で親を選択

```
/* 無効な入力を含むフォームグループ */
.form-group:has(input:invalid) {
  border-left: 3px solid red;
  background: #ffebee;
}

/* チェックされたチェックボックスを含むラベル */
label:has(input:checked) {
  font-weight: bold;
  color: blue;
}
```

入力が無効になった瞬間に自動的に反映！

JavaScriptのイベント管理が不要に

3. z-index地獄からの解放

Popover API

モーダルやツールチップの実装で困ること

z-indexの値がどんどん大きくなっていく...

```
.dropdown { z-index: 100; }  
.modal { z-index: 1000; }  
.tooltip { z-index: 9999; }  
.super-modal { z-index: 999999; }
```

どれが一番上に来るか分からない！ しかもフォーカス管理やESCキーの処理も必要...

従来の解決方法：JavaScript

ライブラリを使うか、大量のコードを書く

- z-indexの動的計算
- フォーカストラップ
- キーボードイベント
- 外側クリックの検知

結果：Popper.jsなどに頼ることに

今はHTMLとCSSだけで解決

Popover API：ネイティブなポップオーバー機能

```
<button popovertarget="menu">開く </button>
<div id="menu" popover>
  <h3>メニュー</h3>
  <ul>
    <li>項目1</li>
    <li>項目2</li>
  </ul>
</div>
```

自動的に全て対応

```
/* ポップオーバーのスタイリング */  
[popover]:popover-open {  
  animation: slideIn 0.2s;  
}
```

自動的に：

- トップレイヤーに表示（z-index不要）
- ESCで閉じる
- 外側クリックで閉じる
- フォーカス管理

JavaScriptゼロ行！

4. 詳細度の戦争を終わらせる

@layer (カスケードレイヤー)

CSSあるある：詳細度の戦い

外部ライブラリのスタイルを上書きしたい時...

```
/* ライブラリ */  
.btn-primary { background: blue; }  
  
/* 上書きしたい... */  
.my-component .btn-primary {} /* 詳細度を上げる */  
body .my-component .btn-primary {} /* もっと上げる */  
.btn-primary { background: red !important; } /* 最終手段 */
```

!important地獄の始まり...

今はCSSだけで解決

@layer : 優先順位を明示的に制御

```
/* レイヤーの定義（左から右に優先度が上がる） */  
@layer library, components, utilities;  
  
@layer library {  
  .btn { background: blue; }  
}  
  
@layer components {  
  .btn { background: red; } /* 詳細度が低くても勝つ！ */  
}
```

レイヤーの順番で優先度が決まる

詳細度の計算から解放される！

5. スタイルに境界線を引く

@scope

グローバルCSSの永遠の課題

「このスタイル、どこまで影響するの？」

BEMやCSS Modulesで名前空間を作ってきたけど...

- 長いクラス名
- 命名規則の学習コスト
- それでも完璧じゃない

もっとシンプルに範囲を限定できないの？

今はCSSだけで解決

@scope : スタイルの適用範囲を限定

```
/* .article内だけに適用 */
@scope (.article) {
  h2 { font-size: 2rem; }
  p { line-height: 1.8; }
}

/* .sidebar内は完全に独立 */
@scope (.sidebar) {
  h2 { font-size: 1.2rem; } /* .articleのh2と競合しない！ */
}
```

シンプルなクラス名でOK

範囲が明確で安心！

@scopeのインラインスタイル構文

```
<section class="article-body">
  <style>
    @scope {
      /* ここに書いたスタイルは、自動的にこの<style>タグの親要素
       (この場合はsection.article-body) だけに適用されます */
      img {
        border: 5px solid black;
        background-color: goldenrod;
      }
    }
  </style>

  <!-- セクションの内容 -->
</section>
```

HTML内にインラインで@scopeを書くことで、そのHTML要素のみにスタイルを適用できます

6. CSS Nesting

Sassがいらなくなる日

ネストした構造を書きたい時

従来：Sassなどのプリプロセッサが必要

ビルド環境の構築、package.jsonの設定...

ちょっとしたプロジェクトには大げさすぎる

今はCSSだけで解決

ネイティブCSSでネスト記法が可能に

```
.card {  
  padding: 1rem;  
  
  & .title {  
    font-size: 1.5rem;  
  
    &:hover {  
      color: blue;  
    }  
  }  
}
```

ビルド不要でネストが書ける！

7. スクロール駆動アニメーション

Scroll-driven Animations

スクロールに連動した演出

パララックス、進捗バー、要素の出現...

従来はscrollイベントでゴリゴリ計算

- パフォーマンスの問題
- スムーズさに欠ける
- 複雑な計算ロジック

今はCSSだけで解決

animation-timeline: scroll()

```
/* スクロール進捗バー */
.progress-bar {
  animation: grow auto linear;
  animation-timeline: scroll(root); /* ページ全体のスクロール */
}

@keyframes grow {
  from { transform: scaleX(0); }
  to { transform: scaleX(1); }
}
```

たったこれだけで、スクロールに連動したアニメーションが実現

8. 初期表示アニメーション

@starting-style

display: noneからのアニメーション問題

モーダルやアコーディオンを スムーズに表示したいけど...

display: noneからdisplay: blockへは トランジションが効かない！

JavaScriptで2段階の処理が必要だった

今はCSSだけで解決

@starting-style : 初期状態を定義

```
dialog {  
  opacity: 1;  
  transition: opacity 0.3s;  
}  
  
/* 表示開始時の初期状態 */  
@starting-style {  
  dialog[open] {  
    opacity: 0;  
  }  
}
```

表示時に自動的にフェードイン！

9. 美しいテキスト折り返し

text-wrap: balance

見出しの最後の単語問題

「最後の単語だけ次の行になっちゃう...」

デザイナーさんからよく指摘される問題 でも動的なテキストだと制御が難しい

今はCSSだけで解決

text-wrap: balance

```
h1, h2, h3 {  
  text-wrap: balance;  
  max-width: 65ch; /* 読みやすい幅に */  
}
```

ブラウザが自動的にバランスの良い改行位置を計算！

10. 自動ダークモード対応

light-dark()関数

ダークモード対応の面倒さ

2セットの色を管理して JavaScriptで切り替え...

メディアクエリで分岐したり クラスで切り替えたり 管理が煩雑に

今はCSSだけで解決

light-dark() : 自動色切り替え

```
:root {  
  color-scheme: light dark;  
}  
  
/* light-dark(ライトモード, ダークモード) */  
body {  
  background: light-dark(white, #1a1a1a);  
  color: light-dark(black, #e5e5e5);  
}
```

OSの設定に自動追従！

ちなみに

一般的にはすべきではないですが、`color-scheme` を使って強制的にモードを切り替えることもできます。

```
:root {  
  color-scheme: light; /* 強制的にライトモードにする */  
}
```

11. 自動リサイズフォーム

field-sizing

テキストエリアの高さ調整

入力内容に応じて高さを変えたい

従来はJavaScriptでscrollHeightを計算して調整...

今はCSSだけで解決

field-sizing: content

```
textarea {  
  field-sizing: content;  
  min-height: 3lh; /* 最小3行分 */  
  max-height: 10lh; /* 最大10行分 */  
}
```

入力に応じて自動的にリサイズ！

まとめ

最後に今回紹介した機能をまとめます

レイアウト・レスポンス

- Container Queries - 親要素ベースのレスポンス
- `:has()` - 子要素の状態で親を選択

UI・インタラクション

- Popover API - ネイティブなポップオーバー
- Scroll-driven Animations - スクロール連動アニメ
- `@starting-style` - `display:none` からのアニメ

スタイル管理

- `@layer` - 詳細度の制御
- `@scope` - スタイルの適用範囲
- CSS Nesting - ネイティブなネスト

その他の便利機能

- `text-wrap: balance` - 自動改行調整
- `light-dark()` - 自動テーマ切り替え
- `field-sizing` - フォームの自動リサイズ

この他にも様々な機能があり、CSSは進化し続けています。ぜひ気になったら調べてみてください。

ご清聴ありがとうございました

このスライドは以下のリポジトリで公開しています。

https://github.com/riya-amemiya/amemiya_riya_slide_data/tree/main/frontend_conf_hokkaido_2025

ご質問はお気軽に

- XやGitHubなど: <https://riya-amemiya-links.oshaburikitchin.com/>

