

Web講習会2021 ワールドワイドウェブ基礎

第5回-①: HTMLとCSS



Arthur

この回の目標

- HTML・CSSのコーディングで発生する諸問題と、それを解決する設計について理解する

良いマークアップ

case1: 色とクラス名

h1要素をteal(ブランドカラー)にするには？



case1の実装例①

style属性を利用

```
<h1 style="color: teal;">TitechApp</h1>
```

メリット

- HTMLを見るだけでどんな装飾かわかる

デメリット

- この装飾を使うすべてのh1に同様の記述が必要
- 色の一括変更に弱い
- 詳細度が強すぎる

case1の実装例②

h1要素に直接装飾をあてる

```
h1 {  
  color: teal;  
}
```

```
<h1>TitechApp</h1>
```

メリット

- CSS、HTMLともに記述量が少ない

デメリット

- この装飾を適用したくないh1が必要になったときに上書きの必要あり

case1の実装例③

文字色を変えるためのクラスを作成
現在のTitech App Project Webはこれ

```
.text-teal {  
  color: teal;  
}
```

```
<h1 class="text-teal">TitechApp</h1>
```

メリット

- h1以外でもこのクラスを利用できる
- h1の装飾を汚染しない

デメリット

- h1の色を変えたくなったときにHTMLの編集が必要
 - ブランドカラーが変わったら…

実装例③の行き着く先

1要素に指定するクラスが増えていく

実態が「HTMLは文書構造のみに責務を持つ」という原則から遠くなる

```
<h1 class="text-teal fontsize-30 mb-20 fontweight-bold">TitechApp</h1>
```


case1の実装例④

プロパティ値に依存しないクラス名

```
.text-brand-color {  
  color: teal;  
}
```

```
<h1 class="text-brand-color">TitechApp</h1>
```

メリット

- ブランドカラーが変わってもCSSの編集だけで済む

デメリット

- 1要素に指定するクラスが増えていく問題は解消されない

case1の実装例⑤

カスタムプロパティの利用 セマンティクスなクラス名にできる

```
:root {  
  --brand-color: teal;  
}  
.headline1 {  
  color: var(--brand-color);  
}
```

```
<h1 class="headline1">TitechApp</h1>
```

メリット

- h1の色以外の装飾を増やしてもクラスが増えない
- 他の要素や文字色以外にも定義した変数を使える

デメリット

- Internet Explorerは非対応
 - CSSプリプロセッサ(SASSなど)の変数機能で解決できる

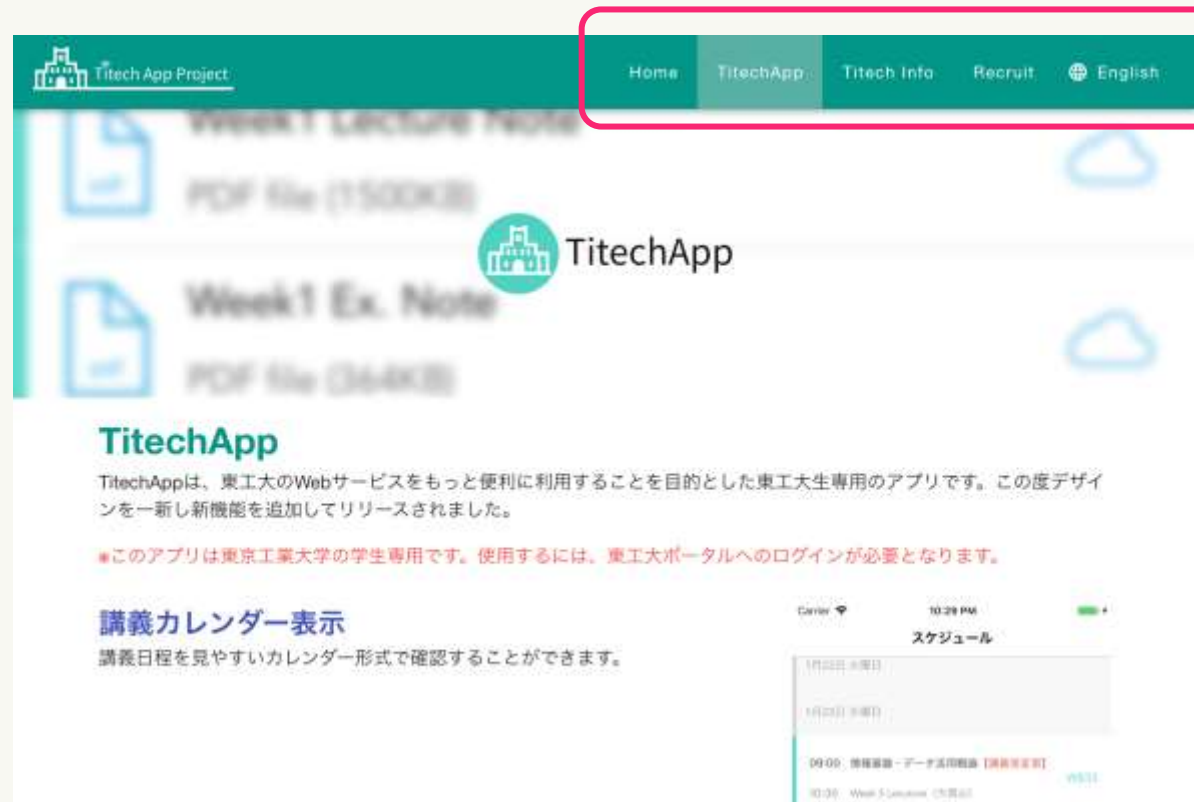
case1での大原則



見た目を変えたくなったときに
CSSを編集すれば解決するようにする

case2: メニュー

横並びのメニューを実装するには？



case2の解答例


```
* {  
  margin: 0;  
  padding: 0;  
}  
.navList {  
  display: flex;  
  list-style: none;  
}  
.navList_item {  
  display: block;  
  width: 100px;  
  text-align: center;  
}
```

```
<ul class="navList">  
  <li class="navList_item">  
    <a href="/hoge">Hoge</a>  
  </li>  
  <li class="navList_item">  
    <a href="/poyo">Poyo</a>  
  </li>  
</ul>
```

Hoge

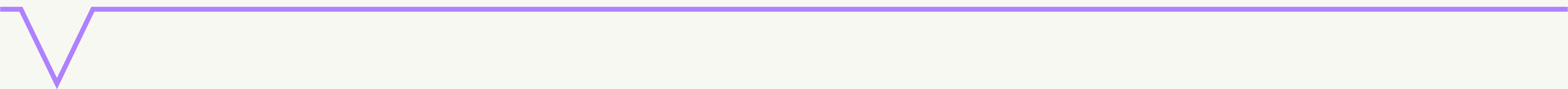
Poyo

case2での大原則



見た目でタグを選ばない
装飾はCSSでどうにでもなる

CSSアーキテクチャ



ソフトウェアコンポーネント

プログラムを機能について複数の機構に分割し、部分問題に帰着
→ 関心の分離 (Separation of Concerns: SoC)

これを実現するためにはモジュール性とカプセル化の機構が必要
外部からの影響を許可したもののみ限定
内部の変更による影響を外部に与えない

CSSの限界

CSSには名前空間がなく、すべてグローバル
クラス名が被ると、容易に他の要素の装飾を汚染する

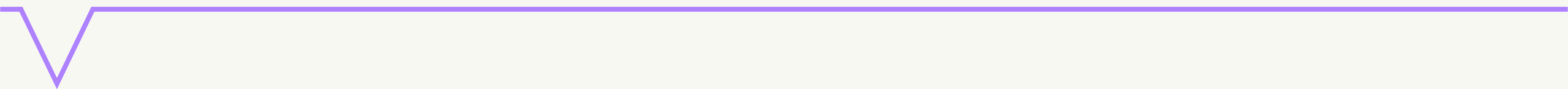
再利用性を維持しつつ、どのように保守性を高めるか

→CSSアーキテクチャの登場

BEM・OOCSS・SMACSS・FLOCSS…

ここでは、OOCSSというアーキテクチャについてゆるふわ解説します

ゆるふわOOCSS



OOCSS

OOCSS (Object-Oriented CSS)

オブジェクト指向プログラミングの雰囲気を採用したアーキテクチャ

構造とスキン(見た目)を分離してクラス定義

定義したクラスを組み合わせてスタイルを定義

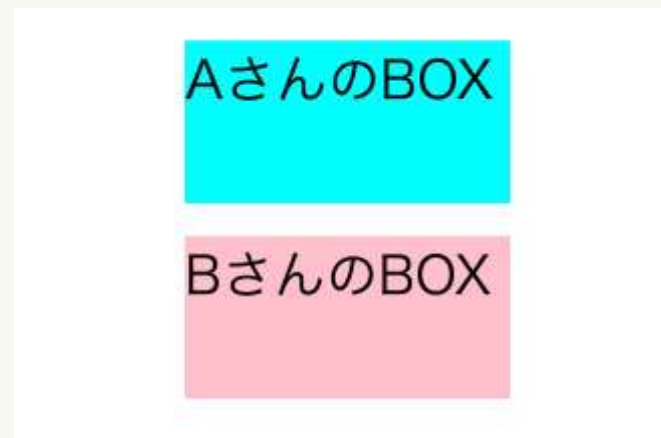
構造: クラス(スーパークラス)

スキン: インスタンス(サブクラス)

工夫なしのコード

```
#box-Asan {  
  width: 100px;  
  height: 50px;  
  margin: 10px auto;  
  background: aqua;  
}  
  
#box-Bsan {  
  width: 100px;  
  height: 50px;  
  margin: 10px auto;  
  background: pink;  
}
```

```
<div id="box-Asan">  
  AさんのBOX  
</div>  
<div id="box-Bsan">  
  BさんのBOX  
</div>
```



OOCSSで書くと

```
.box {  
  width: 100px;  
  height: 50px;  
  margin: 10px auto;  
}  
.box.aqua {  
  background: aqua;  
}  
.box.pink {  
  background: pink;  
}
```

```
<div id="box-Asan" class="box aqua">  
  AさんのBOX  
</div>  
<div id="box-Bsan" class="box pink">  
  BさんのBOX  
</div>
```

AさんのBOX

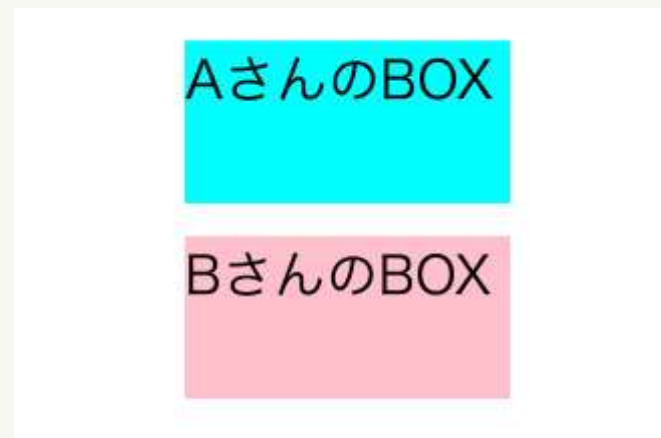
BさんのBOX

より冗長で堅牢な書き方

```
.box {  
  width: 100px;  
  height: 50px;  
  margin: 10px auto;  
}  
  
.box-aqua {  
  background: aqua;  
}  
  
.box-pink {  
  background: pink;  
}
```

ベースのコンポーネント名をprefixに持つ
→スタイルの競合と汚染を防ぐ

```
<div id="box-Asan" class="box box-aqua">  
  AさんのBOX  
</div>  
  
<div id="box-Bsan" class="box box-pink">  
  BさんのBOX  
</div>
```



おすすめの本

Web設計者のためのCSS設計の教科書

もう6年前の本なので多少古い内容が見られますが、
CSSアーキテクチャを学ぶのには分かりやすいと思います

