コンポーネントとCSSのアーキテクチャ

自己紹介

- Name : **azu**
- Twitter : @azu_re
- Website: Web scratch, JSer.info

CSSについて

CSSとはレイアウトとコンポーネントのデザインをするもの

場所

- src/component 以下に置く
  - component/<ジャンル>/<コンポーネント>/
    - コンポーネント.js
    - コンポーネント.css
    - 子コンポーネント

分類

コンポーネントに関連するCSSはコンポーネント単位でJavaScriptと同居するので同じルールが適応される

- src/component/container/
  - ページから見たレイアウトを扱える
  - Containerコンポーネントのレイアウト
- src/component/project/
  - プロジェクトコンポーネントに対するスタイル
- src/component/ui-kit/
  - ui-kitに対するスタイル

コンポーネント以外のCSSは `src/css` に配置する

- src/css/base
  - 要素や*に対して適応するベースのCSS
  - normalize.cssのようなもの
- css/utils/
  - `u-` prefixの`!important`のクラス
  - SUIT CSS utilities
- src/css/variable
  - グローバルなCSS Custom Property変数
  - CSS Custom Property変数は上書き禁止(Transpilerでは正しく依存を解決できないため)

note: 全てのCSSは `src/index.css` から読み込まれ適応される。
そのため、コンポーネントに対するCSSは読み込み順序に依存しては行けない。

原則

- 各コンポーネントは 分類/<ComponentName>/<ComponentName>.css のように配置する
  - ComponentName は大文字開始のキャメルケール
  - <ComponentName>.cssにはそのコンポーネントと子コンポーネント(<ComponentName>-<childName>)を含んで良い
  - 対となるJavaScript(React)に書かれていないクラスはできるだけ扱わない(コンポーネント間の独立性を保つ)
- 各コンポーネントのStateは 分類/<ComponentName>/<ComponentName>.is-<stateName>.css のように配置する
  - StateごとにCSSファイルを分ける
- 各コンポーネントのディレクトリの中にはJavaScriptとCSSと同居して存在する

命名規則

命名規則はSUIT CSSの規約に準拠する。

- SUIT CSS documentation
- 命名規則: SUIT CSS naming conventions

ただしSUIT CSSのライブラリなどはそのまま使わなくても良い

- コンポーネント(Container/Project/ui-kit)はそのコンポーネントより上の要素/クラスのスタイルを扱わない

例) MyComponent

MyComponent/MyComponent.js というコンポーネントがあった場合に

```
import React from "react";
export default class MyComponent extends React.Component {
    render() {
        return <div className="MyComponent">
            <h1 className="MyComponent-title">{this.props.title}</h1>
        </div>;
    }
}
```

コンポーネントのCSS

CSSは以下のような形で書く

MyComponent/MyComponent.css:

```
.MyComponent {
}
.MyComponent-title {
}
```

State と CSS

MyComponentにstateがある場合は、SUIT CSSの規約に基づき `is-*` というクラスが追加される。

この場合は`is-active`というstateが追加されている。

```
<div className="MyComponent is-active">
    <h1 className="MyComponent-title">{this.props.title}</h1>
</div>
```

StateのCSSファイル

このstateに対するCSSは、MyComponent/MyComponent.cssとは別ファイルとして作成する。

MyComponent/MyComponent.is-active.css:

```
MyComponent.is-active {
    /* is-active の時のスタイル */
}
```

FAQ

Container ComponentとProject Componentの使い分け

Containerは次の事ができる

- `project/<ComponentName>`の実装を知っていても良い
- レイアウトに関するスタイルをして良い
- `margin`, `width`, `position:fixed`, `z-index`, `!important` のようなスタイルを使って良い

Containerは次の事ができない

- 親のContainer Componentに触っていは行けない

Projectは次の事ができる

- コンポーネントの内側のスタイルを設定して良い

Projectは次の事ができない

- 他のコンポーネントの実装を知ってはいけない
  - nvkitのUIコンポーネントは例外としてあり
- `margin`, `width`, `position:fixed`, `z-index`, `!important` のようなスタイルを使ってはいけない
  - `margin: 0;` のような打ち消しはありだが、親を意識したマージンを設定してはいけない

SUIT CSSの`.is-state`は`.is-state Component`と書いてはいけない?

- 推奨: `Component.is-state`
- 非推奨: `.is-state ChildComponent`

TODO:

- presentation-annotator/css.md at master · azu/presentation-annotator
- states on descendents · Issue #96 · suitcss/suit

変数と定数

- PostCSSを使い、CSS Custom Property(`--variable: <値>;`)を変数として利用できる。
- CSS Custom Propertyの仕様的には、変数の上書きはできるが読み込み順序に依存するため、
- 基本的に変数は再定義しないでグローバルな定数として扱う。

変数の名前

--<変数名> ： <値>:

という形で変数を定義できる。

変数の使い道

PostCSSで扱えるCSS Custom Propertyの変数はグローバル変数であるため、
変数を多用しすぎるとグローバルに必要な変数とコンポーネントに紐づく変数を命名規則で分離する。

グローバルな変数はページ全体から見た時に、認識として共通であるものをまとめるために利用する。
現在が同じ色だからという理由で、一つの変数を使いまわすと、
変更する際の変更箇所が増えてしまいまとめた意味がなくなってしまうことに注意する。

- 色
- フォント
- `z-index`
- コンポーネント間の幅(できればコンポーネントに紐づくものとして管理したい)
- 幅や高さ
- など

変数を使わなくても問題ない部分を無理やり変数にまとめようとしない。
(あくまでグローバル変数なので、グローバル変数を増やしすぎるのも逆に問題が起きやすいため)

コンポーネントに紐づく変数

コンポーネントのそれぞれの要素には一意なクラスがあるので、
それに準拠した変数名を使う

`--<ComponentName>--<PropertyName>: 値:`

という形式を利用する。

例): ComponentName-childName の width に対する変数

```
.ComponentName-childName {
    width: var(ComponentName-childName--width, 100px);
}
```

コンポーネントに紐付かない変数についてはこの限りではない。

CSS Custom Propertyの仕様は下記を参照する

- CSS カスケード変数のためのカスタムプロパティ – CSS Custom Properties for Cascading Variables Module Level 1 （日本語訳）

(仮)コンポーネントの外からコンポーネントの高さを決めたい

注記) 現実的には結構扱いが難しいでの、親がはっきりしているなら親で子の高さを決定しても良い。
表示が壊れた時に、どこを見ればわかるかというのを意識して書けば良い。

CSS Custom Propertyの変数使い、高さ(height)を外から指定できるようにすることで、
コンポーネントの高さを外から指定できるようにする。

DOM構造的に解決できるならそちらで解決したほうが良い

Component-inner の高さがComponentからの相対値で出せないようなケース

Component.js

```
<div class="Component">
    <div class="Component-inner">
        <p>hi</p>
    </div>
</div>
```

Component.css

```
.Component-inner {
    /* 300px は --Component-inner--height の値が定義されてない場合のFallback) */
    height: var(--Component-inner--height, 300px);
}
```

としておき、外のCSSから--Component-inner--heightの値を設定する。

```
:root{
    --Component-inner--height: 500px;
}
```

こうすることで、Component.cssには具体的な値(Fallbackのみ)がなくなり、

具体的な値は外のCSSに出すことができる。

結論

- 設計なんて人、目的次第 つまり #bikeshedjs