GREEを支えるSassのしくみ

CSS Nite LP, Disk26 「CSS Preprocessor Shootout」
2013/1/12

グリー株式会社

開発本部　山田 あかね

自己紹介 - 山田 あかね

グリー株式会社　開発本部

- GREE Platformのフロントエンド管理
- SNSの各種プロジェクトにおける企画開発

```
staff   544  1 10 18:59 .git
staff   101 12  4 14:48 .gitignore
staff  5560 12  4 14:48 README.md
staff    57 12  4 14:48 compass.sh
staff  1003  1 10 18:41 config.rb
staff  7174  1 10 18:41 css
staff  7174  1 10 18:41 css_debug
staff   170 12  4 14:48 img
staff  7340 12  4 14:48 index.html
staff   986  1 10 18:41 js
staff  1360  1 11 21:05 pages
staff  2584 12  4 14:48 parts
staff  8636  1 10 18:41 sass
```

自己紹介 - 山田 あかね　主な著書

- 現場のプロから学ぶXHTML+CSS(共著)
- Facebookページプロフェッショナルガイド(共著)
- ほか、Webデザイン専門誌に不定期で寄稿

Agenda

Agenda

- はじめに
  - GREE Platformの課題
- なぜSassなのか
- 課題を解決するために、Sassが支えてきたこと
  - 1.開発効率
  - 2.コード品質
  - 3.高速化
- おわりに

はじめに

GREE Platformの課題

はじめに　GREE Platformの課題

- 1. 開発効率の課題
- 2. グローバル対応の課題

- 1. 開発効率の課題
  - 多くのエンジニアでCSSを触らなければならないのにルールが明確でなかった
  - CSSの中で似たソースがコピペの嵐になっており、開発効率が悪くなっていた
  - 無駄にファイルサイズが増えていた

- 2. グローバル対応の課題
  - 海外の通信環境は日本と比べて遅いケースが多い
    - 高速化が重要な課題となる
  - 多言語化を想定しなければならない

はじめに　GREE Platformの課題

- こんな大きな課題があってワクワク
- やっぱりわたしはこれを解決するためにグリーにきたんだ！と確信

なぜSassなのか

なぜSassなのか

- 構築当初から、何らかのCSS拡張メタ言語を使うことを検討
- LessとSassを比較して、どちらにするか判断
  - 当初Lessで開発を始めた
  - Twitterもつかってるし
  - 記法がほとんどCSSと同じで簡単だし
  - わたしにとってSassがよくても、あとから学ぶ人にとって敷居が高すぎてはよくない
- という理由でLessを選択

なぜSassなのか

- しかし、途中でSass+Compassに乗り換え
- このときの判断基準
  - Compassの機能萌え
    - ベンダー分岐イカス！
    - 便利mixinがいっぱいある！
  - SassだってSCSS形式で書けば、CSS知っている人がすぐに覚えやすいじゃないか
  - 当時のLessにextendがなかった
  - 技術を学ぶ楽しさがないとやってられないし！

なぜSassなのか

Jan 28, 2012

LessからSassに切り替え

akane-yamada authored a year ago

Jan 28, 2012

LessからSass

akane-yamada

create sass dir

akane-yamada au

delete old imag

akane-yamada au

moved close-b

akane-yamada au

renamed registration pages
akane-yamada authored a year ago
f94aefdcb8
Browse code

edit navigation-bar's tap-area
akane-yamada authored a year ago
ceb76fc7e9
Browse code

move sample files
akane-yamada authored a year ago
515af42ef1
Browse code

amend sub-nav html structure
akane-yamada authored a year ago
5391d1f71b
Browse code

delete old Less settings
akane-yamada authored a year ago
50ca6e9b3c
Browse code

なぜSassなのか

- 乗り換えてみて、満足！
- クラス名に接頭辞つけてたおかげで
  Less>Sass変換に1日もかからず
- 私の後に入ってきた方々にも受け入れてもらえた
- 大切なことは、今ベストと判断したものを使ってみること
- 使ってみないことには、良さがわからない
- 参考「Sass vs Less」
  - http://wrangl.com/sass-v-less

課題を解決するために
Sassが支えてきたこと

課題を解決するために、Sassが支えてきたこと

- 1.開発効率
- 2.コード品質
- 3.ページ表示高速化

Sassが支えてきたこと
1. 開発効率

Sassが支えてきたこと 1.開発効率

- 黒い画面のポテンシャルを引き出す決意
  - Sass+Compassに変更したと同時に、思い切って黒い画面にシフト
  - シフトする前まではLess.appを使っていました
  - 黒い画面アレルギーの人が入ってきたらどうしよう、と考えるのではなく、黒い画面をアレルギーと思わずに、新しい技術に挑戦する人と働きたいと考える
- 黒い画面は、最初の設定の敷居が高く感じる
- 実際に使うのはものすごく簡単！

```
staff    544  1 10 18:59 .git
staff    101 12  4 14:48 .gitignore
staff   5560 12  4 14:48 README.md
staff     57 12  4 14:48 compass.sh
staff   1003  1 10 18:41 config.rb
staff   7174  1 10 18:41 css
staff   7174  1 10 18:41 css_debug
staff    170 12  4 14:48 img
staff   7340 12  4 14:48 index.html
staff    986  1 10 18:41 js
staff   1360  1 11 21:05 pages
staff   2584 12  4 14:48 parts
staff   8636  1 10 18:41 sass
```

Sassが支えてきたこと 1.開発効率

- コンパイルのための実行命令

```
./compass.sh
```

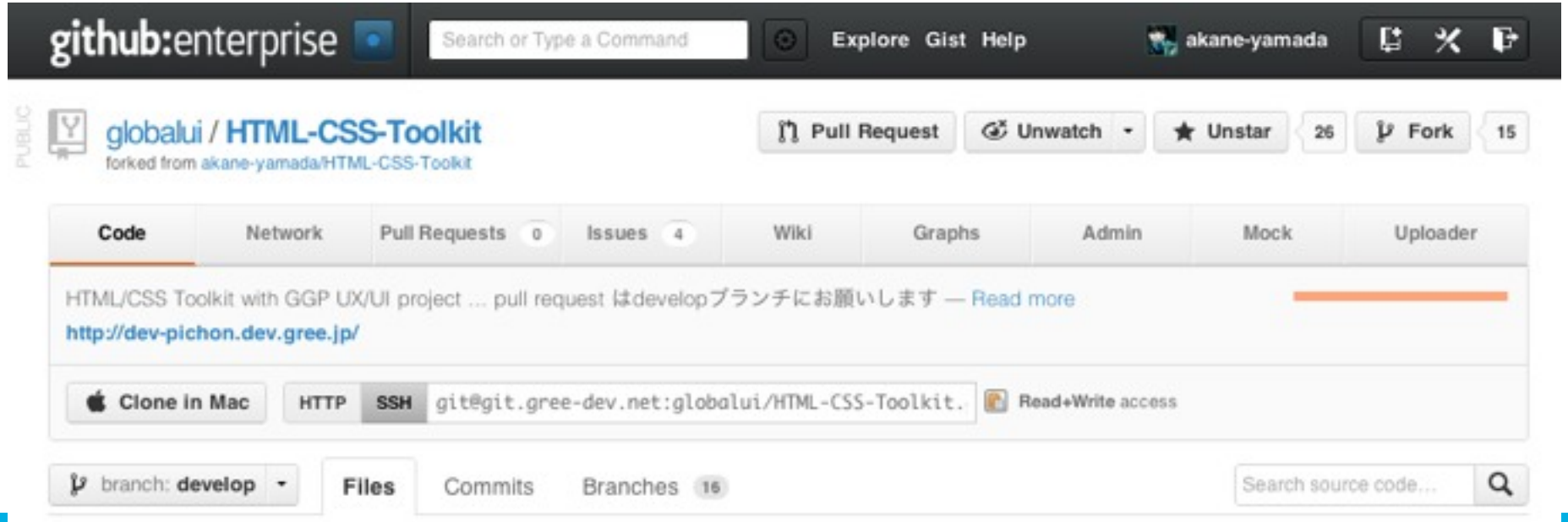
- Sassディレクトリ内の変更を監視できる

```
compass watch
```

- Sassの部品の量が多いためあえてwatchは使わず
  - 必要なときに./compass.sh を実行する
  - とはいえ急いでるときや影響範囲が少ない変更のときはwatchしてリアルタイムに開発もできる

Sassが支えてきたこと 1.開発効率

- こうして、Sass+Compass導入と同時に
  - 黒い画面スキルが自然に身につくようになりました
  - 結果的に、Githubを使った開発手法を教える際に手間取らなくなりました
  - 結果、開発効率が向上しました



Sassが支えてきたこと
2. コード品質

Sassが支えてきたこと 2.コード品質

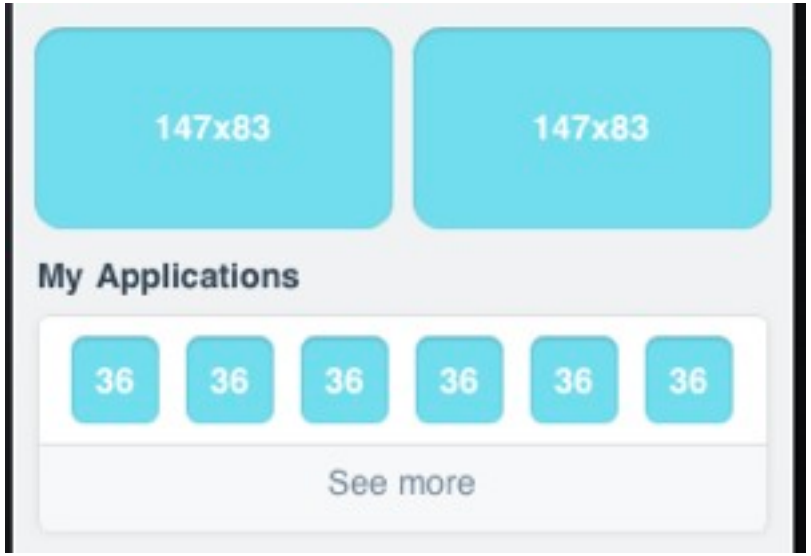
- 私が考える、
  GREE Platformにとって良いソースコード
  - 名前・ルールに一貫性がある
  - 大人数の開発でも破綻しない
- そのために必要な設計
  - Sassの利点を活用し、必要なときに独立した部品を
    呼び出せる仕組み（相互に依存しないモジュール設計）

Sassが支えてきたこと 2.コード品質

- モジュール化+OOCSSの考え方を
  多く取り入れている
- 言い換えるなら
  「積み木のように独立した部品を組み立てる」
- 積み木の部品ひとつひとつは独立して存在
- 極力相互に依存しない＝疎結合
  - http://www.sophia-it.com/content/%E7%96%8E%E7%B5%90%E5%90%88

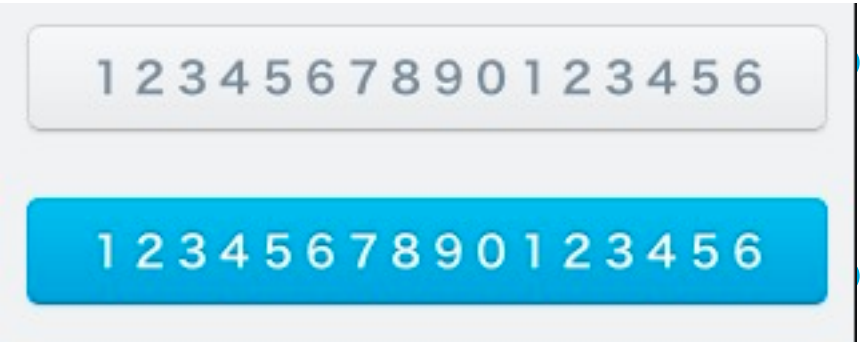
Sassが支えてきたこと 2.コード品質

- 良いコードの下地は良いデザインルールから
- とはいえ、デザインルールをガチガチに
  つくってもらうのも難しい
- デザインの意図・パーツの意味を理解した上で
  モジュールの設計を行うことが大切



Sassが支えてきたこと 2.コード品質

- GREE Platformでの名前のつけかた
  - 出力後の見本に対して名前をつける
  - UIドリブンかつ英語で破綻していない名前
  - 部品で定義できなかったものは便利クラスで調節



.button.block

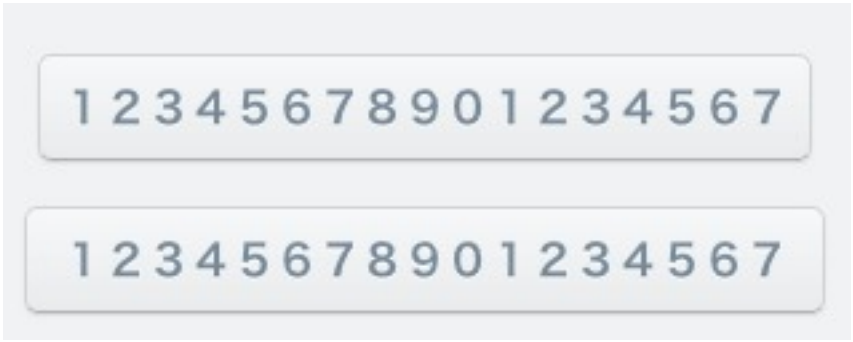
.button.primary.block

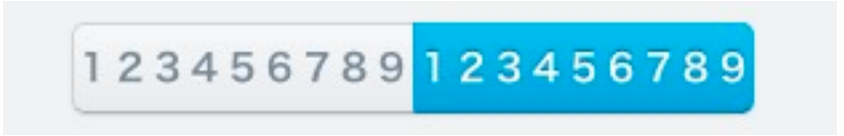
Sassが支えてきたこと 2.コード品質

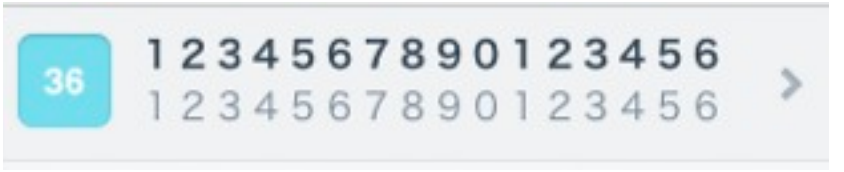
- セマンティックな観点でダメとされる
  class="red"とかわらないのではないか
  - UIパーツの積み木=HTMLをレイアウト用言語と
    みなしてあえて割り切り
  - 情報の意味にこだわりすぎて
    抽象度が詳細になってしまうと
    使い回しに向かなくなる

Sassが支えてきたこと 2.コード品質

- ここは余白15pxのバージョン、ここは下線がないバージョン・・のような亜種がでてくる







Sassが支えてきたこと 2.コード品質

- そこを便利クラスとするのか、モジュール(部品)として定義してしまうのかをしっかり考える
- モジュールになる基準：単独でどの場所でも使い回しできるデザインかどうか
  - リキッド（伸縮自在）な領域のどこに置いても破綻しないデザイン
  - たとえば、ナビゲーションの中に置く用のボタンと本文の中に置く用のボタンの見た目が同一で、マージンだけが少し異なるような場合：ボタンを示すソースコードは共通にして、マージン調節用便利クラスを追加して調節します

Sassが支えてきたこと 2.コード品質

- この考え方で設計することで、
  Sassで自由自在に必要な部品を呼び出す
  下地ができました

Sassが支えてきたこと
3. ページ表示高速化

Sassが支えてきたこと 3.ページ表示高速化

- グローバル化において、表示速度は重要課題
  - 海外の通信環境は日本と比べて遅いケースが多い
  - 国内でもスマホ利用時に長く待たされるのは致命的
- 表示高速化にあたり・・・
  - 画像減色
  - gzip圧縮
  - CDNにスタティックファイルを置く
  - 遅延ロード
- などの手法もお伝えしたいところですが、今回はSassで行える高速化の事例をご紹介します

Sassが支えてきたこと 3.ページ表示高速化

- 一般的に、ひとつのCSSを使ってキャッシュさせるのが好ましい
- しかし、初回に表示させる画面においてサイト全体のスタイルを全部読み込みする必要はない
- そこでSassを活用
  - ページに必要なモジュールだけを呼び出し、CSSをページ別に生成

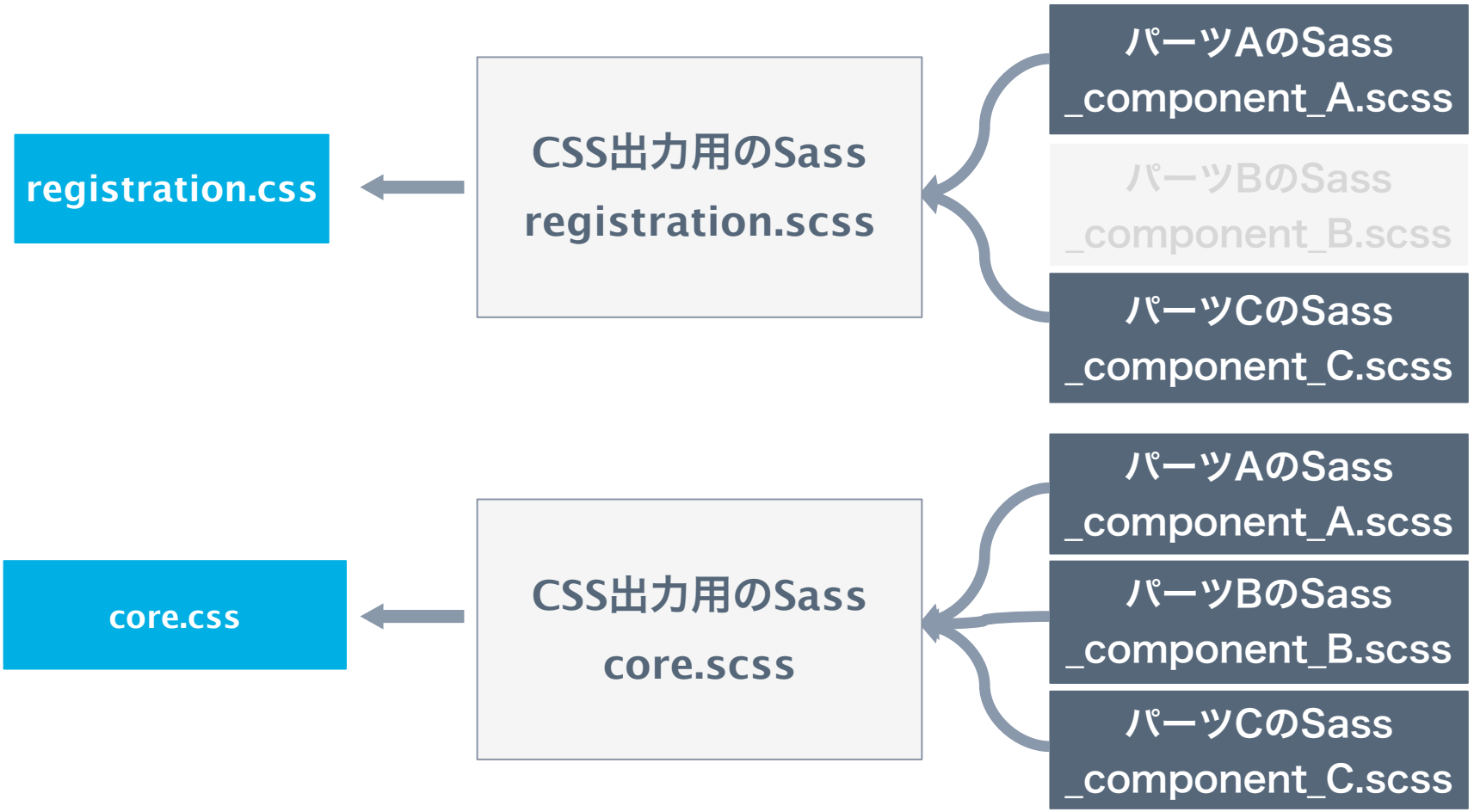
Sassが支えてきたこと 3.ページ表示高速化

- ページに必要なモジュールを呼び出すサンプル

```
// components（抜粋）
/* 基本デザインのテーブルを読み込む */
@import "./partial/component/component_table";

/* フラットなデザインのテーブルを読み込む */
@import "./partial/component/component_table_flat";

/* 角丸なデザインのテーブルはページに不要なので読み込まない */
//@import "./partial/component/component_table_round";

//不要なデザインの部品は読み込まないから無駄がない
```

Sassが支えてきたこと 3.ページ表示高速化

- 部品自体はSass上で共通だから、たとえ異なるCSSが生成されたとしても秩序が保たれる



Sassが支えてきたこと 3.ページ表示高速化

- 出力されるCSSの例
  - ベンダー別/主要ページ別
    - webkit_registration.css
    - moz_registration.css
    - などなど。PHPでベンダー判定してHTMLを返す
    - ベンダープレフィックスが撲滅されるまでは仕方ない
    - 将来的にプレフィックスが撲滅されたら、ベンダー無しのCSSを使えば良い
    - そして、部品自体はおなじ部品のままでいられる
      (部品を変更したときは再コンパイルすればすべてのCSSに新しい部品の変更を適用できる)

Sassが支えてきたこと 3.ページ表示高速化

- ベンダー別CSS出力用のSass

```
/* 出力用のSassでwebkit用の命令が入ったSassを呼び出す */
@import "./partial/vender/webkit";
```

- webkit対応に必要な変数をtrueに

```
/* Compassのベンダーサポートの変数を制御 */
$experimental-support-for-mozilla: false;
$experimental-support-for-webkit: true;
$official: true;
```

Sassが支えてきたこと 3.ページ表示高速化

- もし、このように「必要ないなら削ればいい」の発想を普通のCSSでやろうと思ったら問題だらけ
  - 部品ごとに書いたCSSをインポートした場合、部品を呼ぶたびにHTTPリクエストが発生してしまい、パフォーマンスに悪影響
  - １枚のCSSから不要な行を削除する手法の場合、どこからどこが部品なのか見通しが悪く、ミスの原因

Sassが支えてきたこと 3.ページ表示高速化

- ご紹介したSassによる必要な部品の読み込み手法でやっていること自体は、特に目新しいことではありません。
- しかし、高速化に命を注がなければならないサイトにとって、とても重要な機能のひとつです。

Sassが支えてきたこと 3.ページ表示高速化

- 仕上げに、プロダクション用のCSSと
  デバッグ用のCSSを同時出力
  - プロダクション用：
    余分なスペースやコメントを除外して１行にしたCSS
  - デバッグ用：
    コメントあり・ネスト整形ありのCSS
- Compassのconfigファイルとシェルスクリプト
  にじゅもんをかいておくだけで設定完了

Sassが支えてきたこと 3.ページ表示高速化

- config.rbで設定するじゅもん

```
# 出力したいcssのディレクトリ名を設定
css_dir = (environment == :production) ? "css" :
"css_debug"
# 出力形式を設定
output_style = (environment
== :production) ? :compressed : :expanded
```

- compass.sh(シェルスクリプト）のじゅもん

```
#!/bin/bash
compass compile –e production
compass compile
```

Sassが支えてきたこと 3.ページ表示高速化

- いずれのじゅもんも、最初だけ設定すれば
  あとは何も考えずに

```
./compass.sh
```

これだけでいっちょあがり！

(変更箇所によってはコンパイルに時間かかることもあります)

おわりに

おわりに

- HTML+CSSにCSS拡張メタ言語という技術が入ってきてよかった！
  - 子どもが夢中になって積み木で遊ぶようにCSSを設計、構築することができるようになった
  - これまでより柔軟で多様性をもった運用が可能に

おわりに

- 今回ご紹介した内容のほかに、
  グローバル対応など、Sassを活用する場面は今後
  ますます増えます
- グリーは、こんな楽しいお仕事を、自分で考えて
  提案でき、世界を動かしていける会社です
- グリーで働いてみたいと思った方は、是非私まで
  声をかけてくださいね！

GREE

ありがとうございました

- Twitter　@purprin
- Facebook　http://facebook.com/purprin
- purprin@gmail.com