

Web制作者のための [サス] Sass の教科書

Webデザインの現場で
必須のCSSプリプロセッサ

改訂
2版

平澤 隆(Latele)、森田 壮 著

CSSをより便利に、効率的に書ける!

最新のコーディング
環境にあわせて

大幅!
刷新!

基本から実践テクニックまで、
この一冊で完全網羅

タスクランナー「gulp」での導入方法から、GUIでの導入方法、
著者が実際に仕事の現場で使っているテクニックまで徹底解説!

インプレス

CONTENTS 目次

著者プロフィール	2
はじめに	3

第 1 章 Sass のキホン

11

1-1 まずは Sass って何なのかを知ろう	12
CSS を覚え始めたワクワク感や楽しさがよみがえる Sass	12
Sass とは？	14
Sass だけと SCSS ? sass ファイルと scss ファイルの違い	15
scss ファイルではブラウザは認識できない	17
魅力的な Sass の機能	18
Sass はなぜ誕生したのか	20
Sass の普及率	22
1-2 Sass を導入する前の疑問や不安	23
Ruby の知識は必要？ 黒い画面も使わないとダメなの？	23
エディタやオーサリングツールはそのまま使えるの？	25
コラム：著者が使っているエディタ	25
運用時に Sass を使うのは難しいから、Sass は導入できない？	26
自分以外の関係者が Sass を使えないから、覚えても使えない？	27
1-3 何はともあれ Sass を触ってみよう	28
Sass に対応しているソースコード共有サービス	32

第 2 章 Sass の利用環境を整えよう

33

2-1 本書で使用する環境について	34
node-sass (LibSass) について	34
Node.js について	36
gulp について	36
2-2 Node.js をインストールする	37
Node.js のインストール	37
コラム：Node.js のバージョン管理について	38
2-3 黒い画面を使ってみよう	39
ターミナルの起動方法 (Mac)	39
コマンドプロンプトの起動方法 (Windows)	40
コマンドを入力してみよう	41
現在地 (カレントディレクトリ) に移動する	42

2-4	セットアップ済みの環境をインストールする	46
	サンプルファイルをコピーする	46
	gulp-cli をインストール	47
	npm install で一括インストール	48
2-5	Sass をコンパイルする	49
	gulp タスクを実行して Sass をコンパイル	50
	gulp コマンドと gulpfile.js について	51
	Sass と CSS のディレクトリを指定する	52
	アウトプットスタイルを指定する	52
	ファイルの更新を監視する	55
2-6	セットアップ済みの環境を作成する方法	56
	package.json の作成	56
	gulp と gulp-sass をローカルインストール	57
	コラム：npm install コマンドの小ネタ	58
	gulpfile.js の作成	59
2-7	GUI (Prepros) で Sass を使う	61
	インストール	61
	プロジェクトの登録	62
	Sass ファイルの設定	63
	設定項目	63
	コンパイルする	64
	プロジェクトの設定	64
	フレームワークの使用	65
	コラム：GUI コンパイラのデメリット	65
2-8	Dreamweaver で Sass を使う	66
	サイトの定義	66
	CSS プリプロセッサ設定	67
	コンパイルする	69
	フレームワークの使用	70

第3章 これだけはマスターしたい Sass の基本機能

71

3-1	ルールのネスト (Nested Rules)	72
	ネストの基本	72
	子孫セクタ以外のセクタを使うには	74
	@media のネスト	75
3-2	親セクタの参照 & (アンパサンド)	76
3-3	プロパティのネスト (Nested Properties)	79
	コラム：- (ハイフン) があるプロパティはすべてネストできる	80
3-4	Sass で使えるコメント	81

1 行コメント	81
通常のコメント	81
3-5 変数 (Variables)	83
変数の基本	83
変数名で使える文字と使えない文字	84
ルールセット内で変数を宣言する	85
変数の参照範囲 (スコープ)	85
変数を参照できる場所	87
3-6 演算	88
演算の基本	88
別々の単位で演算する	89
変数と演算を同時に利用する	90
各演算子の注意点や条件など	91
色の演算	92
3-7 Sass の @import	93
@import の概要	93
CSS ファイルを生成しない、パーシャル	95
インポートファイルの複数指定	96
@import の指定位置	96

第4章 高度な機能を覚えて Sass を使いこなそう

97

4-1 スタイルの継承ができるエクステンド (@extend)	98
エクステンドの基本	98
同じルールセット内で、複数継承する	99
エクステンドの連鎖	100
エクステンドが使えるセレクタ	101
エクステンド専用のプレースホルダーセレクタ	102
@media 内ではエクステンドは使用できない	103
警告を抑止する !optional フラグ	105
4-2 柔軟なスタイルの定義が可能なミックスイン (@mixin)	106
ミックスインの基本	106
引数を使ったミックスイン	107
引数に初期値を定義する	109
引数を複数指定する	110
, (カンマ) を使うプロパティには可変長引数を利用する	111
複数の引数があるミックスインを読み込む際に可変長引数を使う	113
ミックスインのスコープ (利用できる範囲) を制限する	114
ミックスインにコンテンツブロックを渡す @content	115
ミックスイン名で使える文字と使えない文字	117
4-3 ネストしているセレクタをルートに戻せる @at-root	118
@at-root の基本的な使い方	118

複数のルールセットに @at-root を適用する	118
@at-root をメディアクエリ内で使った場合	119
@at-root のオプション @at-root (without: ...)	120
@at-root のオプション @at-root (with: ...)	121
4-4 Sass のデータタイプについて	122
データタイプの種類	122
データタイプを判別する	124
4-5 制御構文で条件分岐や繰り返し処理を行う	127
@if を使って条件分岐をする	127
@for で繰り返し処理を行う	130
@while でより柔軟な繰り返し処理を行う	132
@each でリスト（配列）の要素に対して繰り返し処理を実行する	133
4-6 関数を使ってさまざまな処理を実行する	135
関数とは？	135
数値の絶対値を取得する abs()	136
数値の小数点以下を四捨五入する round()	136
数値の小数点以下を切り上げる ceil() と数値の小数点以下を切り捨てる floor()	137
16 進数の RGB 値に透明度を指定して、RGBA 形式に変換できる rgba()	138
明るい色を簡単に作れる lighten() と暗い色を簡単に作れる darken()	139
2 つのカラーコードの中間色を作れる mix()	140
リストの N 番目の値を取得できる nth()	141
指定したキーの値を取得する map-get()	141
セレクタを置換する selector-replace()	143
4-7 自作関数を定義する @function	144
@function とは	144
オリジナル関数の例	145
ネイティブ関数と組み合わせる	145
値を変数に入れる	146
引数に初期値を設定する	146
4-8 テストやデバッグで使える @debug、@warn、@error	147
@debug で結果を確認する	147
@warn で警告を表示する	148
@error でエラーを出力し処理を中断する	150
4-9 使いどころに合わせて	
補完（インターポレーション）してくれる #{ }	151
インターポレーションとは	151
演算しないようにする	152
演算できない場所で演算する	152
アンクォートもしてくれるインターポレーション	153
4-10 変数の振る舞いをコントロールする !default と !global	154
!default フラグ	154
!global フラグ	155

第5章 現場で使える 実践 Sass テクニック

157

5-1 管理 / 運用・設計で使えるテクニック	158
ネストが深すぎると生じる問題を把握して、バランスを見ながら利用する	158
コラム：ネストは何階層までがよいか	160
CSS とは違うパーシャルによる Sass ファイルの分割	161
サイトの基本設定を変数にして一元管理する	163
複数人で制作する場合は、各自の Sass ファイルを用意する	164
コメントを活用してソースをわかりやすくする	165
大規模サイトで活用できる @import のネスト	166
SASS 記法も使ってみよう	168
&（アンパサンド）を活用して BEM 的な設計を快適に	170
@keyframes をルールセット内に書いて関係性をわかりやすくする	173
エクステンドはスコープを決めて利用する	174
EditorConfig と stylelint でコーディングルールを統一する	176
stylelint でコードを解析しエラーを表示する	177
コラム：他の人を思いやって Sass 設計をしよう	178
5-2 レイアウト・パーツで使えるテクニック	179
clearfix をミックスインで活用する	179
変数を使って、サイドバーの幅を自動的に計算する	181
null で簡単に条件分岐をしてレイアウトをする	182
calc と Sass を組み合わせて四則演算を便利に使う	184
@for を使って余白調整用の class を生成する	186
リストマーカー用の連番を使った class 名を作成する	188
連番を使った class 名のゼロパディング（0 埋め）をする	189
文字リンクカラーのミックスインを作る	190
複数の値を @each でループし、ページによって背景を変更する	192
シンプルなグラデーションのミックスインを作る	194
Map 型と @each を使って SNS アイコンを管理する	196
値が比較しづらい z-index を Map 型で一括管理する	201
@function を使って px 指定する感覚でフォントサイズを rem 指定する	202
ネストしたセレクタを selector-replace() を使って書き換える	203
selector-extend() を使い、親セレクタだけ変更して同スタイルを適用させる	206
5-3 スマホ・マルチデバイス、 ブラウザ対応で使えるテクニック	208
スマホサイトでよく見る、リストの矢印をミックスインで管理する	208
埋め込み動画のアスペクト比計算に percentage() を使って楽をする	210
メディアクエリ用のミックスインを作成して楽々レスポンス対応	212
マップのキーの有無を map-has-key() で判定してわかりやすいエラー表示にする	214
CSS ハックをミックスインにして便利に使う	216
Retina ディスプレイなど、高解像度端末の対応を楽にする	218
5-4 gulp のタスクを追加してもっと便利な環境にする	219
glob でパーシャルファイルを一括で読み込む	219
ソースマップでコンパイル前のソースの場所を知る	221
エラー時に Watch を停止させずに自動コンパイルを継続させる	223



エラーに気付きやすくするために通知を出す	224
5-5 PostCSS で Sass をさらに便利にする	226
PostCSS とは	226
ベンダープレフィックスを自動付与する	229
画像名だけで画像のパスやサイズを取得する	232
CSS プロパティの記述順を自動でソートする	238
バラバラになったメディアクエリをまとめてコード量を削減してスッキリさせる	240



第 6 章 もっと Sass を便利にする フレームワークやツール 243

6-1 Sass のフレームワーク紹介	244
総合フレームワーク	244
機能拡張系	246
パーツ系	247
レイアウト系	248
6-2 Sass の GUI コンパイラ	249
Windows/Mac 両対応	249
Mac のみ対応	250



第 7 章 Sass 全機能リファレンス 251

7-1 Sass の基本と高度な機能	252
CSS の拡張機能 (CSS Extensions)	252
Sass のコメント	253
Sass スクリプト	253
@ ルールとディレクティブ	255
制御構文	257
ミックスイン	258
関数の定義	258
7-2 Sass の関数一覧	259
RGB 形式の色を操作する関数	259
HSL 形式の色を操作する関数	260
透明度を操作する関数	263
その他の色を調整する関数	264
文字列を操作する関数	265
数値を操作する関数	267
リストを操作する関数	268
Map 型を操作する関数	270
セレクタを操作する関数	272
チェック関係の関数	274
内部的な値を確認する関数	276
その他の関数	277



7-3 Sass の拡張	278
自作関数を Ruby で定義する	278
キャッシュの保存場所	278
インポートをカスタム	278
付録：コマンド一覧	280
付録：用語集	282
索引	290

第1章

Sassのキホン

第1章では、Sassの魅力や概要などに関して説明していきます。まずは、Sassがどんなものかを知り、導入する前の疑問や不安などを解決しましょう。本章を読み終えるころには、Sassの魅力を知り「今すぐにでも使いたい！」と思っただけでしょう。

- 1-1 まずは Sass って何なのかを知ろう 12
- 1-2 Sass を導入する前の疑問や不安 23
- 1-3 何はともあれ Sass を触ってみよう 28

①-1

まずはSassって何なのか を知ろう

最初に「そもそもSassとはどういったもので、どんなことができるのか」といったSassの魅力をお伝えしていきます。すでにSassのことを知っていて「早く導入したい!」と思っている方は、本章は読み飛ばして第2章(P.33)から読み始めましょう。

CSSを覚え始めたワクワク感や 楽しさがよみがえるSass

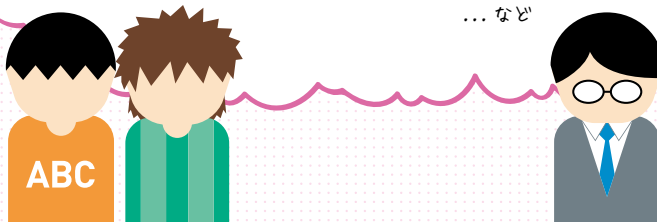
Sass(サス)を利用することで、いくらCSSをより便利に効率的に書けるといっても、普段のCSSによるコーディングで問題なく業務がこなせていれば、慣れの問題や、「CSSをプログラムのように書けたら便利になる!」などとはあまり考えられず、CSSが不便だとは思わないかもしれません。実際、著者の二人もはじめてSassの存在を知ったときは、必要性をあまり感じず、すぐには導入しませんでした。

それは、主に次のような理由からでした。

導入しない理由

- ・今のCSSで十分間に合っている
- ・Rubyや黒い画面を使う必要がある
- ・そこまでCSSに不便さや手間を感じていない
- ・普段、コーディングメインで作業していない
- ・得られるメリットより学習コストのほうが高い気がする
- ・環境に依存するから実務では使いにくい
- ・プログラムが書けないとメリットが少ない、使いこなせない
- ・開発元が英語で、日本語の情報が少ない

... など



こういった理由から、アンテナの高い一部の人たちが導入していることは知っていても、自分たちにはあまり関係ないというイメージでした。本書を手にとっていただいている皆さんも、Sassという言葉聞いたことがあっても、実際に試してみたことのある方は少ないのではないのでしょうか？

著者の二人も、最初は簡単な機能を使うだけで、「導入コストに比べて実務レベルで使えるほどの利点があるの？」と思っていました。しかし、いまやSassの魅力に取り憑かれてしまったので、通常のCSSが不便に感じてしまうほどです。実際にSassでどんなことができるのかは、本節の「魅力的なSassの機能」(P.18)で紹介しています。

Sassは、その魅力よりも先に、導入のハードルの高さや開発環境の依存、学習コストのほうに目が行ってしまうため、ちょっと見ただけでは魅力が伝わりにくい気がしています。

確かに、決して学習コストは低いとはいえませんし、他のライブラリやツール、ソフトに比べて導入のハードルが高いのは事実です。しかし、昨今ではユーザーも増え、日本語の情報もとても増えてきたことで、導入のハードルも下がり、Sassは通常のCSSよりも多く利用されているほど普及しています。詳しくは本節最後の「Sassの普及率」(P.22)を参照してください。

中小企業に比べてガイドラインの変更が容易ではない大手企業(LINE株式会社やクックパッド株式会社が有名)でもSassの導入が進んでおり、Sassを扱えることで、転職や就職に有利になることは間違いないでしょう。求人情報サイトなどでSassを「必要なスキル」や「歓迎スキル」として掲載している企業も多数存在しています。

このような就職や仕事上有利になるなどのメリットもありますが、Sassが与えてくれる一番の恩恵は、「CSSを覚え始めたころの、ワクワク感や楽しさを思い出させてくれること」だと思っています。覚えることは少なくないので、最初は慣れない書き方に戸惑ったり、試行錯誤を繰り返して、効率が落ちてしまったりすることもあると思いますが、そこで諦めず、ほんの少しがんばるだけで、今までのCSSとは違った世界が見えてきます。

次項からは、そんなSassの魅力について触れていきます。まずはSassがどういったものか、どんなことができるのかを見ていきましょう。

ヒント*1

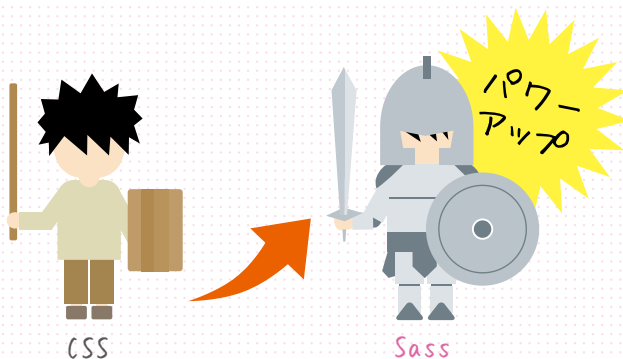
CSSを拡張したメタ言語をCSSメタ言語と表記しています。また、「CSSプリプロセッサ」や「Alt CSS」などと呼ばれることもあります。これらはほぼ同様の意味で扱われます。

昨今では、CSSメタ言語よりCSSプリプロセッサと呼ぶことが多くなっています。

Sass とは？

Sassは魅力的と書きましたが、そもそもSassって何？と思われるかもしれません。Sassとは、CSSを拡張したメタ言語^{*1}です。

メタ言語と聞いてもあまりなじみがないと思いますが、メタ言語とは「ある言語について何らかの記述をするための言語」で、Sassの場合は「CSSに対して機能を拡張した言語」ということになります。小難しい話を抜きにすれば、SassはCSSをより便利に効率的に書けるように大幅にパワーアップさせた言語です。



Sassは「Syntactically Awesome Stylesheets」の略で、日本語に訳すと「構文的にイケてるスタイルシート」という意味になります。構文的にイケてるといわれても、CSSってそんなにイケてないの？といった疑問を持つ方もいると思います。CSSは広く普及させる目的もあって、書式自体は非常にシンプルになっており、プロパティなどを1つ1つ覚えていけば誰にでも習得できるようになっています。しかし、それゆえに複雑なことができないという側面もあり、コードの再利用や、変数、演算、条件分岐などのプログラムでは当たり前のように使える機能がありませんでした。そのCSSの弱点を補う目的で誕生したのがSassなのです  1。



 1 Sassの公式サイト(2017年8月現在)
<http://sass-lang.com/>

第2章

Sassの利用環境を整えよう

第2章では、Sassのインストールから実際に動作させるまでを説明します。Sassを使うにあたって、1番の障壁ともいえるのがこのインストール作業ですが、インストール自体は手順を追っていけば誰でも簡単にできます。慣れてしまえば数分で終わる作業です。

一度インストールが完了してしまえば、その後の作業は大して面倒なことはありません。最初はちょっと面倒ですが、焦らずに進めていきましょう。

2-1	本書で使用する環境について	34
2-2	Node.js をインストールする	37
2-3	黒い画面を使ってみよう	39
2-4	セットアップ済みの環境をインストールする ...	46
2-5	Sass をコンパイルする	49
2-6	セットアップ済みの環境を作成する方法	56
2-7	GUI (Prepros) で Sass を使う	61
2-8	Dreamweaver で Sass を使う	66

② - 1

本書で使用する環境について

本書ではnode-sassでSassをコンパイルします。そのためにNode.jsをインストールします。また、タスクランナーにはgulpを使います。ここでは、それぞれの簡単な紹介と、選定理由について解説します。

ヒント*1

正式にはCUI (Character User Interface) と呼ばれ、キーボードだけで操作をするインターフェースのことです。それに対しマウスなどを使って操作するインターフェースはGUI (Graphical User Interface) と呼ばれます。

Sassのインストールおよび使用には、コマンドラインツールを使用します。いわゆる"黒い画面"と呼ばれているものです*1。Sassの実行そのものは1行のコマンドを入力するだけなので、難しいことはありません。しかし、「どうしても黒い画面を使いたくない！ GUIがいい！」という方もいるかもしれません。その場合はここは軽く目を通していただくにとどめ、「GUI (Prepros) でSassを使う」(P.61) または「DreamweaverでSassを使う」(P.66) から始めていただいても構いません。

node-sass (LibSass) について

LibSass



現在、Sassは大きく分けて2つのSassが存在します。それがRuby SassとLibSassです。Ruby Sassは、その名の通りRubyで開発・動作する従来からあるSassです。LibSassとは、C/C++で開発されたSassです。

LibSassが作られた理由は、Rubyに依存せずさまざまな言語でSassが使えるようにするためです。本書ではNode.jsで使いますが他にもPHPやPython、

ヒント*2

68項目のテスト結果なので、一部新機能などが動作しなかったり、Ruby Sass とコンパイル結果が異なったりする場合があります。

Scalaなどで使えます。そして、RubyでもLibSassを使うことができます。LibSassを使うメリットとして1番に挙げたいのが、コンパイルの速度です。Ruby Sassより圧倒的に速いです。

数年前からLibSassは存在しましたが、Ruby Sassとの互換性が不足しており、使えない機能などがありました。しかしLibSassバージョン3.3以降、互換性がほぼ100%^{*2}となり図1、多くの方がRuby SassからLibSassへ移行しています。

TESTS STATISTICS

Caution! Those stats are computed based on [the few tests](#) we run here on Sass-Compatibility. They do not reflect the current status of support for each engine. Please do not communicate those stats to compare different engines, it would be unfair.

	RUBY SASS 3.2	RUBY SASS 3.3	RUBY SASS 3.4	LIBSASS 3.1	LIBSASS 3.2	LIBSASS 3.3
PASSED	8	54	68	33	67	68
FAILED	60	14	0	35	1	0
PERCENTAGE	11.76%	79.41%	100.0%	48.53%	98.53%	100.0%

図1 Ruby SassとLibSassの互換性を比較するサイトSass Compatibility(<http://sass-compatibility.github.io/>)によると、LibSass 3.3の互換性は100%となっている。

node-sass

node-sassは、LibSassをNode.jsで動作できるようにしたライブラリです。つまり、Node.js環境があればSassを使うことができます。

本書では基本的にnode-sassを使用してSassをコンパイルしています(一部新機能などはRuby Sassで検証しています)。

node-sassをインストールすると「node-sass」コマンドが使えるようになり、黒い画面からSassをコンパイルすることができます。

フロントエンド界隈では、LibSassといえばnode-sassを指すことが多いです。



Node.js について

node-sassを動作させるために必要なものがNode.jsです。Node.jsは、JavaScriptで作られたサーバーサイド環境で、現在のモダンフロントエンド開発においてなくてはならない存在となっています。



Node.jsをインストールすることで使えるようになるパッケージ管理マネージャのnpm (Node Packaged Modules)は、世界で最も大きなライブラリエコシステムで、npmコマンドで簡単に必要なパッケージをインストールしたり、環境の共有を行うことができます。

本書で紹介するnode-sassもgulpもPostCSS^{*3}もnpmコマンドでインストールをします。なので、まずはNode.jsをマシンにインストールしてみましょう。

ヒント*3

PostCSSについては第5章で紹介しています。

詳しくは → P.226

ヒント*4

厳密にはストリーミングビルドシステムと名乗っています。

ヒント*5

gulpのタスクを追加し、より便利に活用する方法は第5章にて紹介しています。

詳しくは → P.219

gulp について

gulpはタスクランナーです^{*4}。タスクランナーとは、さまざまな処理(タスク)を自動化してくれるツールで、Sassのコンパイルも処理の1つとして指定できます。例えば、Sassをコンパイルし、ベンダープレフィックスを付与、CSSを圧縮、ローカルサーバーを立ち上げる。といったことをコマンド1つで実行できます^{*5}。



gulpは、gulpfile.jsというファイルにJavaScriptでタスクを書き込むことで、その通りに処理を実行してくれます。つまりgulpfile.jsがあればタスクを共有することも可能です。

本書がgulpを採用した理由は、gulpfile.jsの書きやすさとタスクの数、そして安定した人気です。

このようなタスクランナーは、以前はGruntが主流でした。最近はWebpackというビルドツールも人気です。フロントエンドのトレンドの移り変わりは早いので今後どのようなツールが主流になるかわかりませんが、使い方に大きな差はありません。まずはgulpでタスクランナーに慣れてみましょう。

Node.js をインストールする

node-sass を使えるようにするために、まずは Node.js をインストールしましょう。

Node.js のインストール

Node.js の公式サイトよりインストーラーをダウンロードしましょう 図 2。



図 2 Node.js 公式サイト (https://nodejs.org/)

ダウンロードボタンが2つあり、左が安定版である LTS^{*6}、右が開発最新版となっています。今回は、LTS を選択します。

執筆時 (2017 年 8 月) は v6.11.0 が LTS の最新バージョンでしたので、そちらをダウンロードしました。本を読んでいる時期でバージョンは変わるとはいますが、最新の LTS をダウンロードしてください。

それぞれの OS のインストーラーがダウンロードされるので実行しましょう

図 3 図 4。

ヒント*6

Long-term Support。長期サポートバージョンのことです。Node.js は偶数バージョンが LTS、奇数バージョンが最新機能版となります。



図 3 Mac インストーラー画面



図 4 Windows インストーラー画面

基本的には「次へ」を進めていけばインストールは完了です。

Column

Node.js のバージョン管理について

本書では簡単に Node.js 環境を構築するために、本家 Node.js サイトからインストーラーを使って、LTS の最新バージョンをインストールしました。もちろんそれで問題はありませんし、新しいバージョンを再度インストールすれば更新されます。

しかし、Node.js は約半年に 1 回のペースでバージョンが更新されるので、複数のプロジェクトが平行している場合、同時に複数のバージョンを使う場合があるかもしれません。

そんなときはバージョン管理ツールを使いましょう。Windows では「Nodist^{*7}」、Mac では「ndenv^{*8}」がオススメです。コマンドでバージョン切り替えもできますし、「.node-version」というバージョンナンバーを書いたテキストファイルがカレントディレクトリにあるとバージョンを自動切り替えしてくれます。

ヒント*7

<https://github.com/marcelklehr/nodist>

ヒント*8

<https://github.com/riywo/ndenv>

第3章

これだけは マスターしたい Sassの基本機能

Sassのインストールが完了して環境が整ったところで、第3章ではSassの基本機能に関して説明していきます。本章を読みながら実際にSassを書いてみることで、より理解が深まってSassの魅力に気付けるでしょう。

3-1	ルールのネスト (Nested Rules)	72
3-2	親セレクタの参照 & (アンバサンド)	76
3-3	プロパティのネスト (Nested Properties)	79
3-4	Sass で使えるコメント	81
3-5	変数 (Variables)	83
3-6	演算	88
3-7	Sass の @import	93

3 - 1

本節のサンプルコード

[http://book2.scss.jp/
code/c3/01.html](http://book2.scss.jp/code/c3/01.html)

ルールのネスト (Nested Rules)

ネストは、Sassの中でも一番よく使う機能で、CSSをHTMLの構造に合わせて入れ子で書いていくことができます。慣れてくれば何も考えずに使えるようになるでしょう。

ネストの基本

ネストの記述方法を知る前に、まずは、次のHTMLにスタイルを適用する場合を見てみましょう。

HTML

```
<div id="main">
  <section>
    <h1>見出し</h1>
    <p>段落</p>
    <p class="notes">注意書き</p>
    <ul>
      <li>リスト</li>
    </ul>
  </section>
  <section>
    <h1>見出し</h1>
    <p>段落</p>
  </section>
<!-- / #main --></div>
```

通常のCSSの場合は、次のようにそれぞれの要素にスタイルを指定するために、子孫セクタを使い、HTMLの階層をたどって書いていくことが多いと思います。

CSS

```
#main section {
  margin-bottom: 50px;
}
#main section h1 {
  font-size: 140%;
}
#main section p, #main section ul {
  margin-bottom: 1.5em;
}
#main section p.notes {
  color: red;
}
```

#main section内の各要素にスタイルを指定するために、毎回同じセレクトアをコピー＆ペーストして最後の要素だけ変更しています。CSSに慣れている方なら、当たり前のように手が動くと思いますので、セレクトアを親から書くことやコピー＆ペーストが気にならないかもしれません。しかし、次のようにSassのネストを使うと、これらのコピー＆ペーストから開放されて階層構造も把握しやすくなります。

Sass

```
#main {
  section {
    margin-bottom: 50px;
    h1 {
      font-size: 140%;
    }
    p, ul {
      margin-bottom: 1.5em;
    }
    p.notes {
      color: red;
    }
  }
}
```

CSSと比べ、#main sectionを何度も書かなくていいので、すっきりしたことがわかると思います。ネストは、このようにセレクトアの後の{ ~ } (波括弧)の中に次のルールセットを書くといった具合に、入れ子にして書いていくことができます。また、インデントすることでHTMLのツリー構造と同じ形式になるので、構造の把握が容易になります。なお、インデントや改行にSass独自のルールはないので、自分が見やすいようにインデントや改行をしても問題ありません。

③-2

本節のサンプルコード

[http://book2.scss.jp/
code/c3/02.html](http://book2.scss.jp/code/c3/02.html)

親セレクトアの参照 & (アンパサンド)

本節では、親のセレクトアの参照ができる & (アンパサンド) に関して説明します。

ネストを使って書いていると、さらに親のセレクトアからスタイルを指定するにはどうするかという疑問が生じます。例えば、サイドバーの横幅をトップページだけ広くしたい場合、CSSでは次のように書くことがあります。

CSS

```
#side {  
  width: 240px;  
}  
body.top #side {  
  width: 300px;  
}  
#side ul.bnr {  
  margin-bottom: 10px;  
}
```

これをSassのネストを使って書く場合、ネストの基本だけで書くとおそらく次のようになるでしょう。

Sass

```
#side {  
  width: 240px;  
  ul.bnr {  
    margin-bottom: 10px;  
  }  
}  
body.top #side {  
  width: 300px;  
}
```

この書き方でも期待した結果は得られますが、#side内のネストやルールセットが増えれば増えるほど、body.top #sideのスタイルが下に行ってしまい可読性が悪くなってしまいます。そういった場合に使えるのが& (アンパサンド) を使った親セレクトアの参照です。

第4章

高度な機能を覚えて Sassを使いこなそう

ここからはより高度なSassの機能を説明していきます。プログラムの内容も出てくるので、プログラムになじみのない方には、少し難しく感じる部分もあるかもしれませんが、簡単なところから順番に学んでいけばちゃんと使えるようになります。覚えることも多いですが、Sassの最も魅力的な機能が詰まったところですよ。得られるメリットも非常に大きいので、ぜひ習得して使いこなせるようになりましょう。

4-1	スタイルの継承ができるエクステンド (@extend)	98
4-2	柔軟なスタイルの定義が可能なミックスイン (@mixin)	106
4-3	ネストしているセレクタをルートに戻せる @at-root	118
4-4	Sass のデータタイプについて	122
4-5	制御構文で条件分岐や繰り返し処理を行う	127
4-6	関数を使ってさまざまな処理を実行する	135
4-7	自作関数を定義する @function	144
4-8	テストやデバッグで使える @debug、@warn、@error	147
4-9	使いどころに合わせて 補完 (インターポレーション) してくれる #{}	151
4-10	変数の振る舞いをコントロールする !default と !global	154

4 - 1

スタイルの継承ができる エクステンド (@extend)

本節のサンプルコード

[http://book2.scss.jp/
code/c4/01.html](http://book2.scss.jp/code/c4/01.html)

エクステンドは高度な機能として紹介していますが、使い方や書き方は難しくありませんし、Sassを扱う上では重要な機能の1つなので、ぜひマスターしましょう。

エクステンドの基本

エクステンドとは、ひと言で説明すると、指定したセレクタのスタイルを継承することができる機能です。「継承」と聞いても少しわかりにくいと思いますので、まずは次の例を見てみましょう。

Sass		CSS (コンパイル後)
<pre>.box { margin: 0 0 30px; padding: 15px; border: 1px solid #ccc; } // .box で使ったスタイルを継承 .item { @extend .box; }</pre>		<pre>.box, .item { margin: 0 0 30px; padding: 15px; border: 1px solid #ccc; }</pre>

.boxで使ったスタイルを.itemにも継承するために、「@extend セレクタ;」というルールで記述しています。

コンパイル後のCSSを見ていただくと、2つのセレクタがグループ化されたのが確認できると思います。エクステンドは、このように一度使ったスタイルを継承して使いまわすことができるので、継承する数が増えれば増えるほど記述量を減らすことができ、同じスタイルを何度も書かずに済むようになります。

エクステンドの基本的な使い方はこれだけなので、思ったより簡単に使えることがわかるでしょうか。

同じルールセット内で、複数継承する

先ほどの例では、1つのルールセット内では1つのエクステンドしか使っていませんでしたが、エクステンドは複数指定することもできます。

指定方法はいたってシンプルで、単純にエクステンドを複数回書けば、それぞれのスタイルを継承してくれます。

Sass

```
// エクステンド
.notes {
  color: #d92c25;
  font-weight: bold;
  text-align: center;
}

.bd {
  border-top: 1px solid #900;
  border-bottom: 1px solid #900;
}

// 複数継承
.item {
  small {
    display: block;
    padding: 10px;
    @extend .notes;
    @extend .bd;
  }
}
```

CSS (コンパイル後)

```
.notes, .item small {
  color: #d92c25;
  font-weight: bold;
  text-align: center;
}

.bd, .item small {
  border-top: 1px solid #900;
  border-bottom: 1px solid #900;
}

.item small {
  display: block;
  padding: 10px;
}
```

.notes と .bd のスタイルを継承するため、.item small に対して、エクステンドを複数指定しています。

CSS (コンパイル後) を見ていただくと、.notes と .bd それぞれに、.item small がグループ化され、.item small に指定したスタイルだけが別で生成されたのが確認できます。

このように複数の継承も問題なく行えますが、継承する数が増えてプロパティがバッチングした場合、CSSの個性の計算通りに、後から書かれたスタイルが優先されるので注意しましょう。

4-2

本節のサンプルコード

[http://book2.scss.jp/
code/c4/02.html](http://book2.scss.jp/code/c4/02.html)

柔軟なスタイルの定義が 可能なミックスイン (@mixin)

本節では、Sassの中でも一番強力な機能として取り上げられることが多いミックスインに関して説明します。

ミックスインの基本

ミックスインの機能を簡単に説明すると、スタイルの集まりを定義しておき、それを他の場所で呼び出して使うことができるというものです。また、引数を指定することで、定義したミックスインの値を一部変更して使うといった、非常に柔軟で強力な処理が可能です。

「ミックスインはエクステンドとの違いがイマイチわからない」という声をよく聞きます。確かに、スタイルを任意の場所で呼び出して使うという点では似ていますが、両者の機能は明確に違います。はじめは違いがわかりにくいかもしれませんが、少しずつ理解していきましょう。

まずは、基本的なミックスインの使い方を見てみましょう。

Sass

```
// ミックスインを定義
@mixin boxSet {
  padding: 15px;
  background: #999;
  color: white;
}
```

ミックスインは、@mixinの後に半角スペースを空けて任意のミックスイン名を定義します。そして、{ ~ } (波括弧) 内にスタイルを書いていきます。

エクステンドは、一度使ったスタイルを継承するために使いましたが、ミックスインの場合は、スタイルを定義しているだけなので、呼び出さない限りは何も起こりません。定義したミックスインを呼び出す際は、次のように「@include ミックスイン名;」と書きます。

Sass	CSS (コンパイル後)
<pre>// 定義したミックスインを呼び出し .relatedArea { @include boxSet; }</pre>	<pre>.relatedArea { padding: 15px; background: #999; color: white; }</pre>

ミックスインはこのように「@mixin」と「@include」をセットで使います。

あらかじめ定義したミックスインが.relatedAreaに展開されたことがわかります。しかしこの例を見ただけだと、エクステンドとの違いがわからないと思います。実際にこれをエクステンドで書き直しても同じCSSが生成されます。では、エクステンドとの違いがわかるように定義したミックスインは最初の例と同じとして、ルールセットが1つ増えた場合の例を見てみましょう。

Sass	CSS (コンパイル後)
<pre>// 定義したミックスインを呼び出し .relatedArea { @include boxSet; } // 別のルールセットでも呼び出し .pickupArea { @include boxSet; }</pre>	<pre>.relatedArea { padding: 15px; background: #999; color: white; } .pickupArea { padding: 15px; background: #999; color: white; }</pre>

エクステンドでは、(カンマ)区切りでグループ化されたのに対して、ミックスインでは、まったく同じスタイルが.relatedAreaと.pickupAreaに展開されました。これで生成されるCSSの違いはわかったかと思います。

しかし、現時点ではエクステンドの方が合理的なソースになるため、ミックスインのメリットが見えてこないと思います。ですので、次項からは、エクステンドとの明確な違いやメリットを見ていきましょう。

引数を使ったミックスイン

先ほどまでの例では、エクステンドとの違いやメリットがイマイチわからなかったと思いますが、冒頭で軽く触れたように「引数」を使うことで、エクステンドとの違いやミックスインの強力さが見えてきます。引数は、ミックスイン

ヒント*1

例として使用していますが、現在のブラウザには border-radius プロパティにベンダープレフィックスは不要です。

で定義した値の一部を変更する機能です。

まずは、引数を使った簡単な例を見てみましょう*1。

Sass

```
// 引数を使ったミックスインを定義
@mixin kadomaru($value) {
  -moz-border-radius: $value;
  -webkit-border-radius: $value;
  border-radius: $value;
}
```

引数を使う場合、ミックスイン名の直後に () (丸括弧) を書き、括弧内に引数 (変数) を書きます。

Sass

```
.box {
  @include kadomaru(3px);
  background: #eee;
}

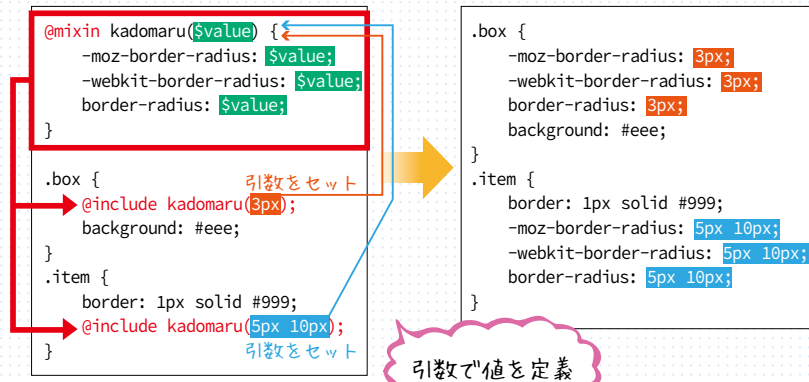
.item {
  border: 1px solid #999;
  @include kadomaru(5px 10px);
}
```

CSS (コンパイル後)

```
.box {
  -moz-border-radius: 3px;
  -webkit-border-radius: 3px;
  border-radius: 3px;
  background: #eee;
}

.item {
  border: 1px solid #999;
  -moz-border-radius: 5px 10px;
  -webkit-border-radius: 5px 10px;
  border-radius: 5px 10px;
}
```

どちらも同じミックスインから呼び出していますが、.boxと.itemでそれぞれ値が異なっていることがわかります。エクステンドでは同じスタイルを継承して使いましたが、ミックスインはこのように、値の一部を変えて使いまわすことができます。



④ - 3

ネストしているセレクトタを
ルートに戻せる @at-root

@at-root は記述した場所より親のセレクトタや @media など除外し、ルートに戻ることができる機能です。

本節のサンプルコード

[http://book2.scss.jp/
code/c4/03.html](http://book2.scss.jp/code/c4/03.html)

@at-root の基本的な使い方

@at-root はルールセットの中で次のように記述して使います。

Sass

```
.block {  
  .element-A {  
    width: 80%;  
  }  
  @at-root .element-B {  
    width: 100%;  
  }  
}
```

CSS (コンパイル後)

```
.block .element-A {  
  width: 80%;  
}  
  
.element-B {  
  width: 100%;  
}
```

.element-A は、ネストしているのでコンパイル後のCSSでは、.block .element-A となっていますが、@at-root を使った .element-B はルートから書き出されているのが確認できます。

複数のルールセットに
@at-root を適用する

@at-root は複数のルールセットに対しても使うことができます。


```

Sass
.block {
  .element-A {
    width: 80%;
  }
  @at-root {
    .element-B {
      width: 100%;
    }
    .element-C {
      width: 50%;
    }
  }
}

```

```

CSS (コンパイル後)
.block .element-A {
  width: 80%;
}

.element-B {
  width: 100%;
}

.element-C {
  width: 50%;
}

```

.element-B と .element-C がルートから書き出されました。

@at-root をメディアクエリ内で使った場合

@media 内で使った場合、該当のルールセットは @media の外ではなく中に書き出されます。

```

Sass
.block {
  width: 50%;
  @media (max-width: 640px) {
    width: 100%;
    @at-root {
      .item {
        margin-bottom: 30px;
      }
    }
  }
}

```

```

CSS (コンパイル後)
.block {
  width: 50%;
}
@media (max-width: 640px) {
  .block {
    width: 100%;
  }
  .item {
    margin-bottom: 30px;
  }
}

```

コンパイル後の .item は、.block が除外され @media 内に書き出されたのが確認できます。

第5章

現場で使える 実践Sassテクニック

第5章は、実際の現場で活用できそうなSassテクニックをアレコレ詰め込みました。比較的簡単なテクニックからミックスインを使った便利なテクニック、gulp や PostCSS といった現在のフロントエンド開発において必要と思われる情報なども紹介しています。実際に Sass を使う上でのヒントになる情報がいっぱい詰まった章です。

5-1	管理 / 運用・設計で使えるテクニック	158
5-2	レイアウト・パーツで使えるテクニック	179
5-3	スマホ・マルチデバイス、 ブラウザ対応で使えるテクニック	208
5-4	gulp のタスクを追加して もっと便利な環境にする	219
5-5	PostCSS で Sass をさらに便利にする	226

⑤ - 3

本節のサンプルコード

[http://book.scss.jp/
code/c5/03.html](http://book.scss.jp/code/c5/03.html)

スマホ・マルチデバイス、ブラウザ対応で使えるテクニック

スマートフォンや、タブレット端末などマルチデバイス対応で使えるテクニックや、ブラウザ対応に関するテクニックを紹介していきます。

スマホサイトでよく見る、リストの矢印をミックスインで管理する

スマートフォン向けサイトやタブレット向けのサイトの場合、右側に「>」などの矢印を付けたデザインがよく使われます [図 7](#)。

そこで、この矢印をミックスインとしてあらかじめ定義しておき、いつでも呼び出せるようにします。まずは、基本となるミックスインを用意します。

このミックスインを、呼び出したいルールセットで `@include` します。



図 7

スマホ向けサイトなどでよく見かける矢印が付いたリンク

Sass (基本となるミックスイン)

```
@mixin linkIcon($color: #333) {
  &::before {
    content: "";
    position: absolute;
    top: 50%;
    right: 15px;
    width: 10px;
    height: 10px;
    margin-top: -7px;
    border-top: 3px solid $color;
    border-right: 3px solid $color;
    transform: rotate(45deg);
  }
}
```

Sass (ミックスインを呼び出す)

```
ul.linkList {
  margin: 20px;
  li {
    list-style: none;
    margin: 0 0 1px;
    a {
      position: relative;
      display: block;
      padding: 15px {
        right: 27px;
      }
      background: #eee;
      color: #333;
      text-decoration: none;
      @include linkIcon();
    }
  }
}
```

CSS (コンパイル後)

```
ul.linkList {
  margin: 20px;
}
ul.linkList li {
  list-style: none;
  margin: 0 0 1px;
}
ul.linkList li a {
  ... (略) ...
}
ul.linkList li a::before {
  content: "";
  position: absolute;
  top: 50%;
  right: 15px;
  width: 10px;
  height: 10px;
  margin-top: -7px;
  border-top: 3px solid #333;
  border-right: 3px solid #333;
  transform: rotate(45deg);
}
```

このミックスインでは引数にborder-colorを渡しているので、「@include linkIcon(#383F9A);」などと書けば簡単に矢印の色を変更することができます

図 8。



図 8 引数の値を変えるだけで、矢印の色を変更することができます

埋め込み動画のアスペクト比計算に percentage() を使って楽をする

Youtube などの動画をサイトに表示させるために iframe の埋め込みコードを HTML に貼り付けると思いますが、そのままではレスポンシブ Web デザインになっていないのでスマホなどに対応していません。そのため、スマホなどで閲覧した際に、動画部分だけはみ出してしまうことがあります。

この問題を解決するために、埋め込み用の iframe を div で囲って、CSS でレスポンシブ対応を行います。

HTML と CSS は、次のような感じになります。

HTML

```
<div class="movie">
  <iframe width="640" height="360" src="https://...(略)..."
  frameborder="0" allowfullscreen></iframe>
</div>
```

CSS

```
@media all and (max-width: 768px) {
  .movie {
    position: relative;
    padding-top: 56.25%;
  }
  .movie > iframe {
    position: absolute;
    top: 0;
    left: 0;
    width: 100% !important;
    height: 100% !important;
  }
}
```

iframe に対して、「width: 100% !important;」を指定すれば、最低限レスポンシブ対応が可能ですが、高さが変わらないので、動画の上下に黒い背景の余白が出てしまったりして見栄えが悪くなってしまいます。そこで上記のような CSS を適用することで高さも可変するようにしています。

細かいCSSの解説は省きますが、この中でもポイントになってくるのが、`.movie` に指定している「`padding-top: 56.25%;`」です。この「56.25%」というのが、16:9のアスペクト比をパーセンテージにした場合の値になります。

埋め込みたい動画のアスペクト比が常に16:9なら56.25%を指定しておけば問題ありませんが、アスペクト比の異なる動画を埋め込みたい場合に、毎回計算し直すのは面倒です。

そこで、この計算をSassの`percentage()`関数を使って簡単に行う方法を紹介します。

```
Sass
// Responsive Movie
@mixin rwd-iframe($width: 16, $height: 9) {
  position: relative;
  padding-top: percentage($height / $width);
  >iframe {
    position: absolute;
    top: 0;
    left: 0;
    width: 100% !important;
    height: 100% !important;
  }
}
```

使い回しが楽にできるようにミックスインで定義しています。

ポイントになってくるのは「`percentage($height / $width)`」の部分で、このように書くだけで簡単にパーセンテージにしてくれます。

そして、先ほど用意したミックスインを使うには次のように書きます。

```
Sass
@media all and (max-width: 768px) {
  .movie {
    @include rwd-iframe(640, 360);
  }
}
```

埋め込みたい動画のアスペクト比が異なる場合は「640, 360」の部分を変更すればOKです。

これでアスペクト比の異なる動画の埋め込みが簡単に行えるようになりました。

5-4

gulpのタスクを追加して
もっと便利な環境にする

本節のサンプルコード

[http://book2.scss.jp/
code/c5/04.html](http://book2.scss.jp/code/c5/04.html)

ヒント*26

ダウンロードURL
[http://book2.scss.jp/dl/
c5/](http://book2.scss.jp/dl/c5/)

ヒント*27

カレントディレクトリの
移動やnpm installなど
については第2章を参照
してください

詳しくは → P.33

第2章でgulpを使ったSassをコンパイルする環境を作成しました。ここでは、gulpのタスクを追加して環境をもっと便利にしてみましょう。

まずは、第5章のサンプルファイル^{*26}をコピーしてください。第2章でgulpの設定を行っている場合は、第2章のデータをそのまま使ってもOKです。

第5章のサンプルファイルを使う場合は、下記コマンドでパッケージをインストールしましょう^{*27}。

```
npm install
```

globでパーシャルファイルを
一括で読み込む

昨今のCSSコンポーネント設計では、コンポーネントごとにファイルを細分化する手法も多く用いられます。その際に、コンポーネントが増えるごとに、パーシャルファイルを@importで指定するのは面倒です。

Sass

```
@import "components/headings";  
@import "components/texts";  
@import "components/lists";  
@import "components/buttons";  
@import "components/images";
```

ヒント*28

gulp-sass-glob -
<https://www.npmjs.com/package/gulp-sass-glob>

npm から「gulp-sass-glob^{*28}」パッケージをインストールしましょう。

```
npm install --save-dev gulp-sass-glob
```

パッケージをインストールしたら、gulpfile.jsに読み込みの設定とsassGlobをsassの前に追記します。

```
gulpfile.js
var gulp = require('gulp');
var sass = require('gulp-sass');
var sassGlob = require('gulp-sass-glob'); //①「gulp-sass-glob」を読み込み

gulp.task('sass', function() {
  return gulp.src('./sass/**/*.scss')
    .pipe(sassGlob()) //②「sass」の前に指定
    .pipe(sass({outputStyle: 'expanded'}))
    .pipe(gulp.dest());
});
```

これで設定は完了です。gulpを起動していない場合は、起動し直しましょう^{*29}。globを使えば、下記のように「**」と指定できます。

ヒント*29

Ctrl+Cで停止し、再度コマンドを実行しましょう。

Sass

```
@import "components/**";
```

これだけで「components」ディレクトリの全ファイルがインポートされます。コンポーネントファイルが増えることがあっても、いちいちファイル名を指定しないでインポートすることが可能です。

globはファイル名が数字・アルファベット順にインポートされます。もしもコンポーネント間に依存関係があると、コンパイルエラーや、レイアウト崩れの可能性があります。依存関係がないコンポーネント設計をしましょう。

⑤ - 5

PostCSSで Sassをさらに便利にする

本節のサンプルコード

[http://book2.scss.jp/
code/c5/05.html](http://book2.scss.jp/code/c5/05.html)

PostCSSでSassにはできない機能を拡張し、より便利にSassを書けるようにしてみましょう。

PostCSS とは

PostCSSは、Node.js製のCSSの変換ツールです[図 17]。プラグインを組み合わせることができさまざまな処理を行うことができます。CSSだけではなくSassにも使うことができるので、Sassの機能拡張に使ってみましょう。



図 17 PostCSS 公式サイト
(<http://postcss.org/>)

PostCSS プラグイン

PostCSSの標準機能は解析とAPIの提供のみなので、CSSの処理にはプラグインが必要です。PostCSSのプラグインは、PostCSS.partsというプラグイン検索サイトが用意されているので、キーワード検索やカテゴリー検索で目的のプラグインが探せます。現在200以上のプラグインが登録されています

[図 18]。

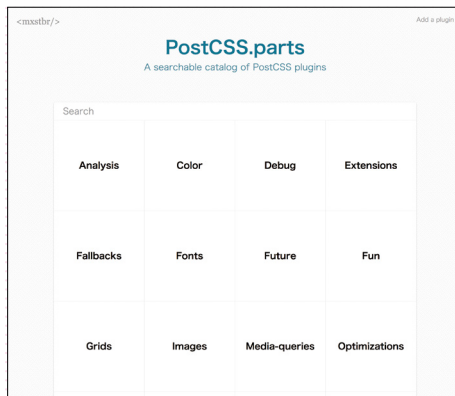


図 18 PostCSS.parts
(<https://www.postcss.parts/>)

PostCSS のユースケース

PostCSSはSassにも使えますが、SassをCSSにコンパイルすることではできません。あくまで、Sassを処理しSassを書き出します。ファイルの変換は行いません。なのでSassと併用する場合は、以下のパターンでの使用となります。

• Sassを処理

SCSS ファイルを SCSS ファイルに処理します^{*34} 図 19。その後、SassでCSSにコンパイルします。PostCSS独自のプロパティなど、Sassでコンパイルできない場合に事前に処理を行います。



図 19 SCSSをPostCSSでコンパイル、その後SassでCSSに処理

• コンパイルしたCSSを処理

SassでコンパイルしたCSSを処理します 図 20。ポストプロセッサと呼ばれていた方法です。



図 20 SassでCSSにコンパイルしてからCSSをPostCSSで処理

• サンドイッチで処理

Sass のコンパイルの前後で処理を行います 図 21。



図 21 PostCSSでSCSSを処理、SassでCSSにコンパイル、さらにCSSをPostCSSで処理

ヒント*34

SCSSの処理にはPostCSS SCSS Syntaxプラグインが必要です。

ヒント*35

gulp-postcss - <https://www.npmjs.com/package/gulp-postcss>

これら組み合わせの指定は、一見複雑そうに見えますが、gulpで簡単に順番を指定できます。

また、組み合わせ次第で複雑に処理を行いますが、PostCSSのコンパイルは非常に高速なのでSassのコンパイルと体感にあまり差は感じません。

PostCSS のインストール

まずはPostCSSを使う準備として、npmから「gulp-postcss^{*35}」パッケージをインストールしておきましょう。

```
npm install --save-dev gulp-postcss
```

パッケージをインストールしたら、gulpfile.jsに読み込みの設定をしましょう。

```
gulpfile.js
var gulp = require('gulp');
var sass = require('gulp-sass');
var sassGlob = require('gulp-sass-glob');
var sourcemaps = require('gulp-sourcemaps');
var plumber = require('gulp-plumber');
var notify = require("gulp-notify");
var postcss = require('gulp-postcss'); // ①「gulp-
postcss」を読み込み
```

gulp-postcssをインストールすれば、前説で紹介したgulpプラグインと同じようにPostCSSを設定することができます。

.pipeでpostcssを指定し、その引数の中でPostCSSのプラグインを指定していきます。

```
gulpfile.js
.pipe(postcss([autoprefixer()])))
```

このようにpostcssの引数でautoprefixerを指定しているのがわかります。

では、PostCSSプラグインを追加してみましょう。前節のgulpタスクに追加していきます。

第6章

もっと Sass を 便利にするフレーム ワークやツール

第6章では、Sassをもっと使いこなすためのフレームワークやコンパイラなどの情報を紹介します。ひと通り Sass が使いこなせるようになって、その魅力や面白さに引き込まれた方は、ぜひ本章を参考にしてより一層充実した Sass ライフをお楽しみください。

6-1	Sass のフレームワーク紹介	244
6-2	Sass の GUI コンパイラ	249

6-1

Sass のフレームワーク紹介

さまざまなフレームワークがあるのも Sass の魅力の1つです。本節では、ひとまとめにフレームワークといっても用途はさまざまなので、総合・機能拡張・パーツ・レイアウトと分けて紹介します。

総合フレームワーク

グリッドシステムにレスポンシブ対応、フォームやボタンなどの各パーツ、アイコンや JavaScript のカルーセルなど、サイトで使うものがほぼすべて用意されているフレームワークです^{*1}。

ヒント*1

本書公式サポートサイトで紹介しているサイトのリンクを掲載しています。
<http://book2.scss.jp/link/>

Bootstrap

<http://getbootstrap.com/>

最も人気のある Web アプリケーションフレームワーク。元々は LESS で作られていましたが、現在は Sass に移行しています。HTML にクラスを付けるだけで簡単にレイアウトすることができます。Bootstrap をベースにしたテンプレートも多数配布されており、拡張性も高いです。

