Web制作者のための Sass［サス］の教科書

改訂2版

Webデザインの現場で必須のCSSプリプロセッサ

平澤 隆（Latele）、森田 壮 著

最新のコーディング環境にあわせて大幅刷新！

CSSをより便利に、効率的に書ける！

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

第5章 現場で使える実践Sassテクニック

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！」と思っていただけるでしょう。

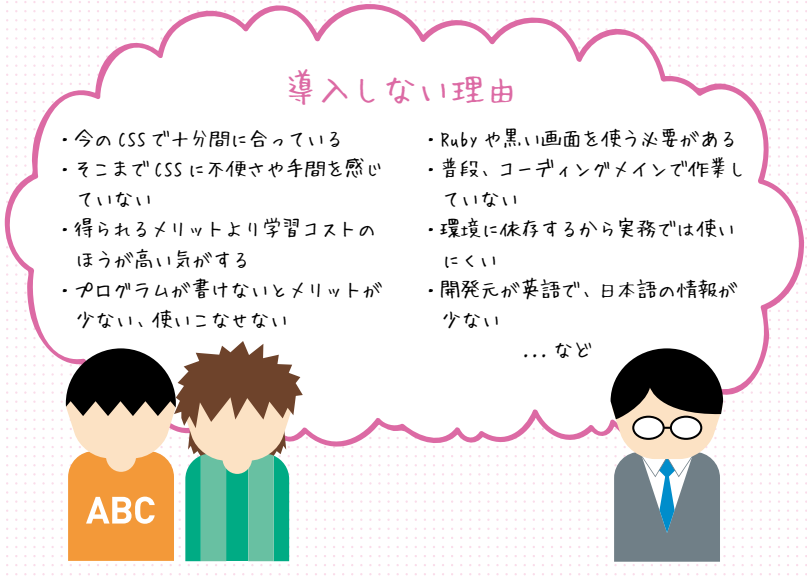
①-1 まずはSassって何なのかを知ろう

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。



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。本書を手に取っていただいている皆さんも、Sassという言葉を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。

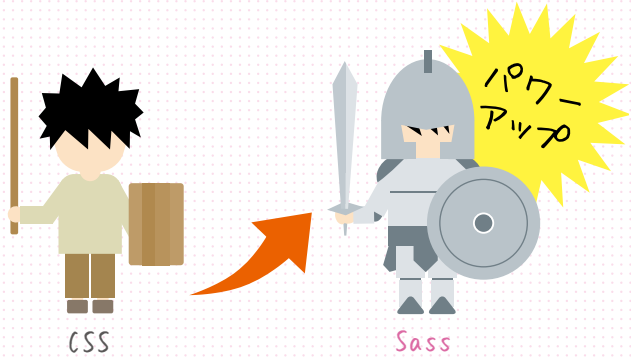
このような就職や仕事上有利になるなどのメリットもありますが、Sassが与えてくれる一番の恩恵は、「CSSを覚え始めたころの、ワクワク感や楽しさを思い出させてくれること」だと思っています。覚えることは少なくないので、最初は慣れない書き方に戸惑ったり、試行錯誤を繰り返して、効率が落ちてしまったりすることもあると思いますが、そこで諦めず、ほんの少しがんばるだけで、今までのCSSとは違った世界が見えてきます。

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。

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 図1 。

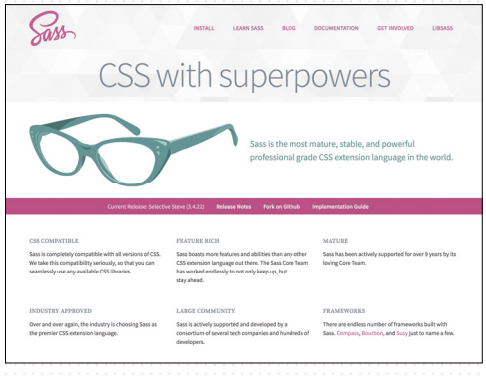


図1 Sassの公式サイト（2017年8月現在）
http://sass-lang.com/

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が、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。

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

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。

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。

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。

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。

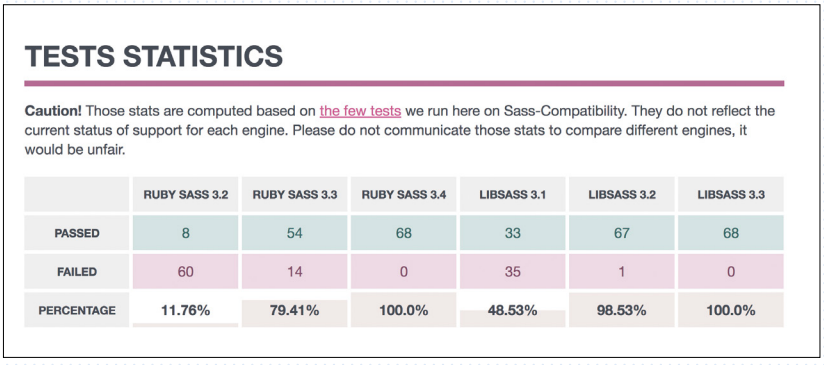
LibSassが作られた理由は、Rubyに依存せずさまざまな言語でSassが使えるようにするためです。本書ではNode.jsで使いますが他にもPHPやPython、

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。

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もあります。



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

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。

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

②-2 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思いますが、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

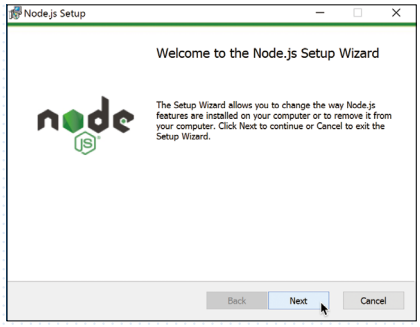


図4 Windowsインストーラー画面

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)

本節のサンプルコード
http://book2.scss.jp/code/c3/01.html

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、子孫セレクタを使い、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、毎回同じセレクタをコピー＆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、セレクタ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、このようにセレクタの後の { 〜 }（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 親セレクタの参照 &(アンパサンド)

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

本節のサンプルコード
http://book2.scss.jp/code/c3/02.html

ネストを使って書いていると、さらに親のセレクタ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)を使った親セレクタの参照です。

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。プログラム的な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。得られるメリットも非常に大きいですから、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)

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。

本節のサンプルコード
http://book2.scss.jp/code/c4/01.html

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

```
.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;
}
```

CSS（コンパイル後）

```
.box, .item {
  margin: 0 0 30px;
  padding: 15px;
  border: 1px solid #ccc;
}
```

.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ティングした場合、CSSの個別性の計算通りに、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 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。

本節のサンプルコード
http://book2.scss.jp/code/c4/02.html

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

```
// 定義したミックスインを呼び出し
.relatedArea {
    @include boxSet;
}
```

CSS（コンパイル後）

```
.relatedArea {
  padding: 15px;
  background: #999;
  color: white;
}
```

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

```
// 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;
}
```

CSS（コンパイル後）

```
.relatedArea {
  padding: 15px;
  background: #999;
  color: white;
}
.pickupArea {
  padding: 15px;
  background: #999;
  color: white;
}
```

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

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]。

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。

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。

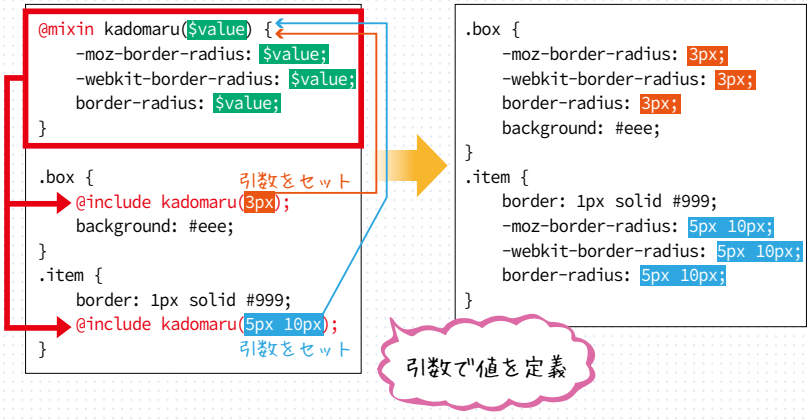
Sass

```
.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タや @media などを除外し、ルートに戻す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

@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

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

CSS（コンパイル後）

```css
.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

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

CSS（コンパイル後）

```css
.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。

5-3 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

本節のサンプルコード

http://book.scss.jp/code/c5/03.html

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る

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

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。

本節のサンプルコード

http://book2.scss.jp/code/c5/04.html

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
```

ヒント*26

ダウンロードURL
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

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";
```

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。

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。

ヒント*28

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

ヒント*29

Ctrl+Cで停止し、再度コマンドを実行しましょう。

⑤-5 PostCSSでSassをさらに便利にする

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。

本節のサンプルコード

http://book2.scss.jp/code/c5/05.html

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で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。

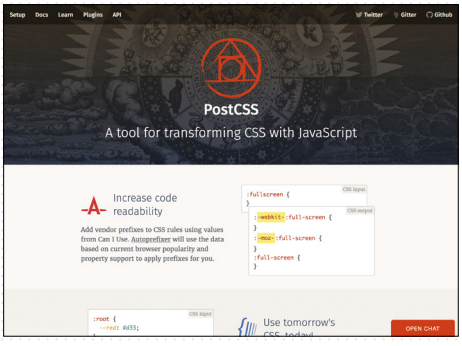


図17 PostCSS 公式サイト
(http://postcss.org/)

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。

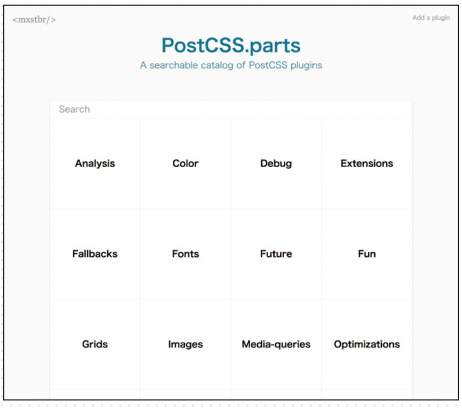


図18 PostCSS.parts
(https://www.postcss.parts/)

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。

ヒント*34

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

- **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

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。

ヒント*35

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

```
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

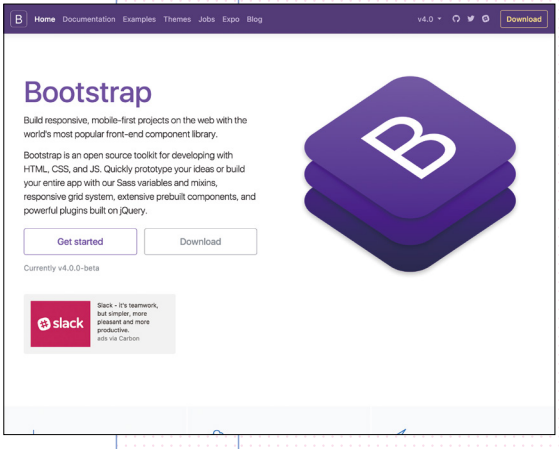
さまざまなフレームワークがあるのもSassの魅力の1つです。本節では、ひとまとめにフレームワークといっても用途はさまざまなので、総合・機能拡張・パーツ・レイアウトと分けて紹介します。

総合フレームワーク

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/

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]。



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。