Web制作者のための Sass［サス］の教科書

これからのWebデザインの現場で必須のCSSメタ言語

平澤 隆（Latele）、森田 壮 著

知りたいことがすぐ引ける！
全機能リファレンス付き

「Sass」を導入する企業や、
求人情報に歓迎スキルとして掲載する企業も
増加中！

基本から実践テクニックまで、
この一冊で網羅

OS別の丁寧なインストール方法、人気のフレームワーク「Compass」の活用方法など、
著者が実際に仕事の現場で使っているテクニックを徹底解説！

インプレスジャパン

著者プロフィール

平澤 隆（ひらさわ・たかし）

ゲーム・マンガが好きで、絵描きの夢を追って上京。東京ゲームデザイナー学院を卒業後、Web関係の会社、デジパ株式会社のフロントエンドエンジニア、フリーランスとしての活動を経て、2013年1月に株式会社ラテールを設立。Webサイト制作全般およびコワーキングスペース（CSS Space）運営を行っている。趣味は、ボルダリングとスノーボード。犬も好きだけど、ねこ派。

●株式会社ラテール http://latele.co.jp/
●ブログ http://css-happylife.com/
●ねこブログ http://nekonekocube.com/

森田 壮（もりた・そう）

ソウラボの屋号で活動するフリーランスWebデザイナー。アパレル会社のEC担当からWeb制作の世界へ。その後、デジタルハリウッドを卒業し、制作会社でデザイナー、ディレクターを経てフリーランスへ。企画からデザイン、コーディング、構築までサイト制作全般を担当。制作の他にも、講師業や株式会社フィフティフォレストでEC業務など。
趣味はマンガとラーメン。猫も好きだけど、いぬ派。

●Sou-Lab http://sou-lab.com/
●ブログ http://blog.sou-lab.com/
●いぬブログ http://bebe-log.com/

謝辞

本書の執筆をするにあたって、翻訳などのサポートを引き受けてくれた宮内 耕治さん、レビューを引き受けてくれた三宅 葉子（Latele）さん、酒井 優（WEBCRE8.jp）さん、松田 清香さんに心より感謝いたします。
また、インプレスジャパンの柳沼さんには、本書の企画をいただいてから、わがままの多かった著者の意見も聞き入れていただき、最後までお付き合いいただきました。
本当にありがとうございました。

筆者による公式サポートサイト

書籍の内容に関するサポートや、書籍内で掲載されているソースコードの一覧などが提供されています。

URL **http://book.scss.jp/**

はじめに

本書は、Webサイト制作に必須のCSS（Cascading Style Sheets）を、より便利に効率的に書けるようにパワーアップさせた「Sass（サス）」に関してのアレコレを書いた、教科書的な本です。

・Sassって聞いたことはあるけど、導入が面倒そうという方
・Sassを始めてみたいと思っているが、今一歩踏み出せない方
・勉強コストとの天秤に掛けて、Sassのメリットがイマイチ見えてこない方
・周りでSassを使い出した人がいて、焦りを感じている方
・CSSを今よりも効率的に書きたいと思っている方

これらの方がSassを導入するきっかけとなり、ひと通りSassの機能を使いこなせるようになるのが本書の目標です。
Sassは、今までCSSを使っていた状況なら、数ページの小規模サイトから、数千ページの複数人でコーディングをするような大規模サイトまで、あらゆるシーンでより便利に効率的に制作を進めることができます。そのため、コーディングを主業務にしている方はもちろんですが、少しでもCSSに触れたことがある方すべてに読んでいただきたいという思いがあります。

本書では、Sassにまったく触れたことがない方を主な対象としているので、Sassの機能だけでなく、Sassの概要から利用環境の整え方まで丁寧に解説しています。また、せっかく環境を整えても、その後実際に使われなければ意味がないので、より実践的な内容やフレームワークなど、Sassを使いこなすために必要な内容を網羅しています。
なお、SassはCSSを拡張した言語なので、HTML＋CSSの知識が必須です。本書では、ひと通りCSSを使ったコーディングができる方を対象としており、HTMLやCSSに関してはほとんど説明していません。その点はご留意ください。もし、CSSの知識や理解が不足していると感じられる方は、CSSをある程度勉強してから、あらためて本書を手に取っていただくと、Sassに関する理解も早くなると思います。

最初は少し面倒ですが、一度初めてしまえばSassの魅力に取り憑かれ、今までのCSSには戻れなくなると思います。本書がきっかけでSassの使い方を覚え、もうCSSには戻れないカラダになっていただければ、著者としてそれ以上にうれしいことはありません。

2013年8月の雨の日
平澤 隆＆森田 壮

CONTENTS 目次

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。

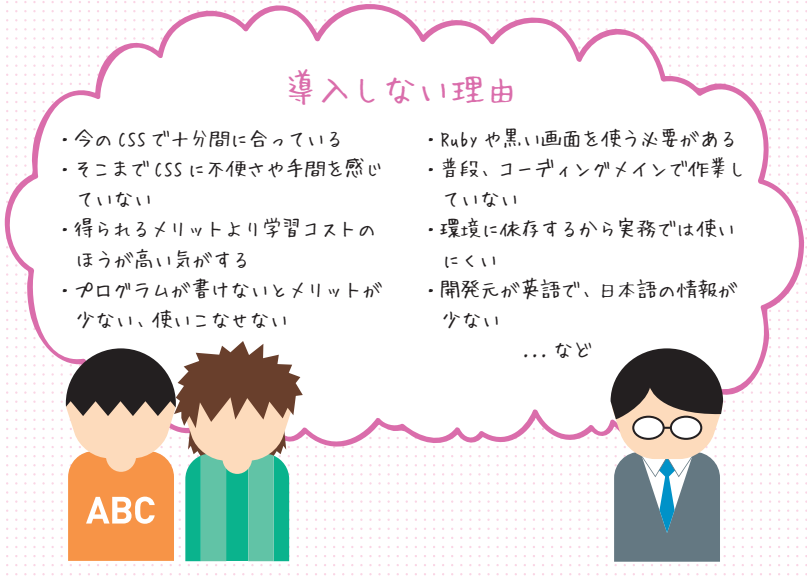
①-1 まずはSassって何なのかを知ろう

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.35) 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、著者の二人も初めて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はない方が多いのではないでしょうか？

著者の平澤がSassに初めて触れたのは2010年の11月ごろでしたが、軽く触ってみたものの実際に仕事で使うこともなく時間が流れていました。本格的に覚え始めたのは、その1年後の2011年の冬です。著者の森田も2010年ごろに導入はしましたが、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、いまや著者の二人は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、昨今では日本語の情報も増えてきたことで、導入のハードルも下がり、Sassの普及が進んでいます。

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、今後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。すでに、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。

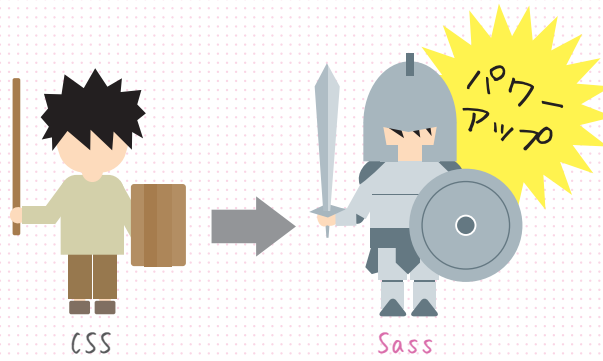
Sassとは？

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。

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」と「CSSメタ言語」は同様の意味になります。



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

図1 Sassの公式サイト（2013年8月現在）
http://sass-lang.com/

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。

Sassの詳しい歴史に関しては追って触れていきますが、現在主流となっているSassはCSSと互換性があるので、今までのCSS＋αとして使える設計になっています。 そのおかげで、初めてSassを使っても今までのCSSと変わらない感覚で使うことができるので、無理にすべてを覚えようとせずに、必要な機能を使うだけでも十分にSassの恩恵を受けることができます。

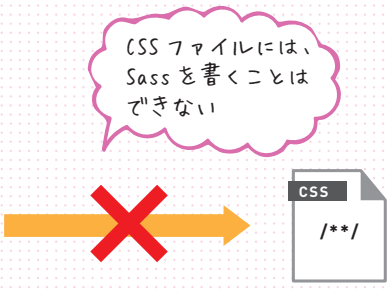
Sass だけど SCSS ？
sass ファイルと scss ファイルの違い

「構文的にイケてるスタイルシート」ということは、既存のCSSとは構文が異なっているため、CSSファイルにはSassを記述できないということになります。そのため、CSSでは拡張子が「css」のファイルになりますが、Sassの場合「scss」という専用の拡張子のファイルに記述していくことになります。

Sassソース

```
$width: 260px;

#side {
    float: right;
    width: $width;
    nav {
        margin: 0 0 20px;
    }
}
```



Sassは、CSSと互換性があるため、CSSファイルの拡張子を「scss」に変更するだけでも立派なSassファイルになります（Sassの機能を使わないとあまり意味はありませんが……）。

この拡張子ですが、Sassなら拡張子も「sass」のほうがわかりやすいのに「scss」という拡張子になっています。実はSassには記法が2つあり、最初に作られたのがSASS記法で拡張子は「sass」、後から作られたのがSCSS記法で拡張子は「scss」となっています。 この2つの記法には大きな違いがあり、最初に作られたSASS記法は、セレクタの後の{〜}（波括弧）の代わりにインデントで書き、値の後の;（セミコロン）は省略できるといった、非常に簡素化された記法

でした。その反面、通常のCSSとは互換性がなく書式も異なっていたため、それがネックとなって広く普及するには至りませんでした。そこで、CSSとの互換性を高めたSCSS記法が作られました。

ちなみに、SCSSは「Sassy CSS」の略で、翻訳すると「カッコいいCSS」や「イカしたCSS」という意味になります。

2つの記法の違い

CSS

```
ul {
    margin: 0 0 1em;
}
ul li {
    margin-bottom: 5px;
}
```

このCSSを、SCSS記法とSASS記法で書いた場合、次のようになります。

SCSS記法のSass

```
ul {
    margin: 0 0 1em;
    li {
        margin-bottom: 5px;
    }
}
```

SASS記法のSass

```
ul
  margin: 0 0 1em
  li
    margin-bottom: 5px
```

SASS記法では、記述量が減って簡素化していることがわかります。しかし、SASS記法はCSSと互換性がなく、インデントの深さや改行の位置など、細かい書式が決まっており、CSSの書式で書くとエラーになってしまいます。

SCSS記法では、ネスト[*2]という機能を使って書いているので、書式が異なっていますが、CSSと互換性があるため、CSSと同じ書式で書いても問題ありません。現在、Sassを指す場合はSCSS記法が一般的になっており、Sassの公式サイト（http://sass-lang.com/）でもSCSS記法のサンプルがデフォルト表示になっているため、本書でも特に言及がない限り、SCSS記法で説明しています。

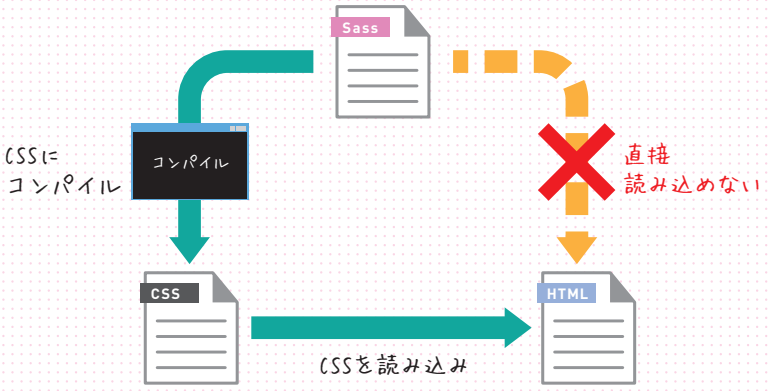
ヒント*2

ネストに関しては、第3章の「ルールのネスト（Nested Rules）」で詳しく説明しています。

詳しくは → P.88

SCSSファイルでは ブラウザは認識できない

Sassには、SCSS記法とSASS記法の2つの記法があるという説明をしましたが、CSSとは拡張子が違うため、そのままではブラウザが認識できません。そのため、SassファイルをCSSファイルにコンパイル[*3]する必要があります。



コンパイルをするには、Ruby[*4]とSassをインストールする必要があります。Rubyに関してはインストールするだけで、Rubyの知識が必要になることはありません。

ソフトウェアをインストールするだけなら、通常はサイトなどからファイルを手に入れれば、後はインストーラーがやってくれるのでインストールは容易ですが、Sassに関してはCUI[*5]を使って行う必要があります。

Sassの導入にあたってハードルが高く感じてしまう要因の1つがSassのインストール作業だと思いますので、インストールに関しては、第2章「Sassの利用環境を整えよう」(P.35)で詳しく説明しています。

また、現在ではコマンドプロンプトやターミナルを一切使わなくともSassが利用できる、さまざまなGUI[*6]ツールがあるので、気軽に導入することが可能となっています。GUIツールのインストールや使い方に関しても第2章の「GUI(Koala)でSassを使用する」(P.73)で説明しています。

ヒント*3

コンパイルとは「変換」のことで、本書ではSassをCSSに変換することをコンパイルといっています。また「ビルド」と表記されている場合も同じ意味になります。

ヒント*4

Rubyのインストールは、Windowsのみ必要です。

ヒント*5

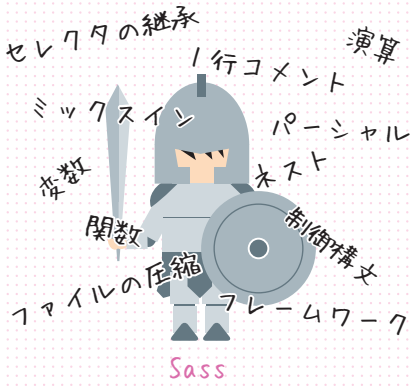
CUI(Character User Interface)は、テキストベースでキーボードからコマンドで操作を行います。コマンドプロンプトやターミナルなどのことです。

ヒント*6

GUI(Graphical User Interface)は、表示にグラフィックを多用しマウスでの操作が可能なため、操作性に優れ視認性もいいことから、主流となっているユーザーインターフェースです。

魅力的な Sass の機能

Sassは、CSSをより便利に効率的にするためのさまざまな機能があります。その分学習コストはかかりますが、一度覚えてしまえばコスト以上のメリットが得られます。ここでは、Sassによってどんなことができるようになるかを軽く紹介します。



記述の簡略化ができる

親子関係にあるセレクタを入れ子（ネスト）にして書くことができます。CSSでは、親の要素から対象要素までのセレクタを何度も書く必要がありますが、Sassはネストさせることで、同じ親のセレクタをまとめることができます[*7]。

専用のコメントが使える

Sassでは、CSSのコメント（/* ～ */）の他にも、JavaScriptなどでなじみがある、1行コメント（// ～）を使うことができます[*8]。

同じ値を使いまわすことができる

CSSでは、例えば複数の要素に同じ色を適用させる場合、何度も同じスタイルを書かなければなりませんが、Sassでは「変数」を使うことによって、同じ値を使いまわせます[*9]。

四則演算が使える

あらゆる値に対して、四則演算を使うことができます。これにより、widthからpaddingの値を引いたり、画像のサイズを半分に割ってRetinaディスプレイなどの高解像度端末に対応させたりといった使い方ができます[*10]。

ヒント*7
第3章の「ルールのネスト（Nested Rules）」
詳しくは → P.88

ヒント*8
第3章の「Sassで使えるコメント」
詳しくは → P.96

ヒント*9
第3章の「変数（Variables）」
詳しくは → P.98

ヒント*10
第3章の「演算」
詳しくは → P.105

ヒント*11

第4章の「スタイルの継承ができる@extend」

詳しくは → P.118

一度使ったセレクタを使いまわせる

@extend という機能を使えば、一度使ったセレクタのスタイルを、別のセレクタでも使うことができます。同じスタイルを何度も書く手間から開放され、コンパイル後のCSSはグループ化されるので、非常に合理的なソースになります*11。

ヒント*12

第4章の「柔軟なスタイルの定義が可能なミックスイン(@mixin)」

詳しくは → P.128

コードの再利用が可能

スタイルをまとめてテンプレートやモジュールのように定義し、それらを簡単に読み込んで使うことができます。また、引数を指定することで部分的に値を変えるといった、複雑な処理をすることも可能となっています。この機能は、ミックスインと呼ばれており、Sassの中でも最も強力な機能の1つです*12。

ヒント*13

第3章の「CSSファイルを生成しない、パーシャル」

詳しくは → P.113

1つのCSSファイルにまとめることができる

「パーシャル」という機能を使うことで、複数のSassファイルをコンパイル時に1つのCSSファイルとしてまとめることが可能です。これにより、HTTPリクエストを減らしつつ、Sassファイルを分割して管理しやすくできます*13。

ヒント*14

第4章の「制御構文で条件分岐や繰り返し処理を行う」

詳しくは → P.140

条件分岐などのプログラム的な処理ができる

条件分岐や繰り返し処理などの制御構文が使えます。各制御構文を使うことで、複雑な処理が可能になり、ミックスインなどと組み合わせることで、非常に強力な機能になります*14。

ヒント*15

第2章の「アウトプットスタイルを指定する(Styleオプション)」

詳しくは → P.58

CSSファイルを圧縮できる

Sassは、CSSファイルにコンパイルする際に、圧縮された状態にすることができます。これにより、Sassを使っているときは可読性を重視してコメントなどをしっかり使い、コンパイルされたCSSは圧縮して軽量化することが可能です*15。

ヒント*16

第6章の「Sassをさらに便利にするCompass」

詳しくは → P.245

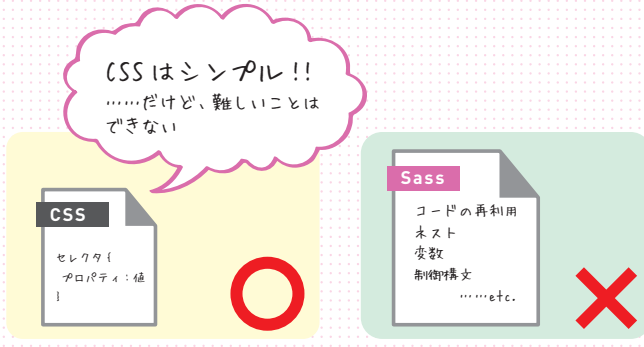
便利なフレームワークが使える

SassではCompass(コンパス)を代表とする、さまざまなフレームワークが開発されています。これにより、Sassをさらに便利に使うことができます*16。

Sassはなぜ誕生したのか

Sassの概要や魅力について説明してきましたが、Sassはどういった経緯で誕生したのか、そんな歴史的なことにも少し触れておきたいと思います。特に知らないと困るようなことではないので、退屈だったら読み飛ばしていただいて構いません。

本章の「Sassとは？」でも軽く触れましたが、CSSは広く普及させる目的などから、できる限りシンプルな書式で、多くの人にとってわかりやすい仕様になっています。 もちろん、CSSでもセレクタやプロパティ、ブラウザの対応などを覚えなくてはいけないので、習得が容易だとはいえませんが、書式だけを見ればセレクタ（pや#mainなど）から始まり、波括弧（}）で終わるルールセットを繰り返し書いていくだけのシンプルなものでした。これは、CSSのメリットでもありますが、同時に複雑なことはできないというデメリットも併せ持っています。



Sassの誕生

特にここ数年で、マルチデバイスの対応や、Retinaディスプレイなどの高解像度ディスプレイへの対応、CSS3、レスポンシブWebデザインなど、CSSに求められる要件も上がっています。そういった中で、CSSの仕様にさまざまな機能が追加されるのを待っていても、それからブラウザへの実装が進んで実用レベルに達するには相当な年月がかかってしまいます。

Sassの開発者であるハンプトン・カトリン（Hampton Catlin）氏とネイサン・ワイゼンバウム（Nathan Weizenbaum）氏は、CSSの仕様策定やブラウザの実装を待つのではなく、サーバーやローカル環境で動作するプログラムによって、既存のCSSの仕様に合うように変換すればいいことに気付き、Sassの開発に着

手しました。 そして、最初に開発したSass（SASS記法）が2006年に公開されました。 公開当初、SassはHaml[*17]とセットで誕生したため、インデントを使ってCSSをシンプルに書けるのが特徴でした。

SCSS 記法の誕生

しかし、いくらCSSがパワーアップしたといっても、既存のCSSの書き方とは大きく異なっており、CSSとの互換性もなかったため、Webデザイナーやプログラムに触れない人たちにとっては非常にハードルが高い存在でした。 そこで、Sass 3.0より、CSSの記法と似ている Sassy CSS（SCSS）が導入されました。これによりCSSと完全互換となり、既存の環境からもSassの導入が簡単になったことで利用者が増え、Sassの主流はSCSS記法になりました。

国内での Sass

日本では、Sassの誕生時点ではRubyユーザーがHamlと一緒に使う程度で、あまりWeb業界の人たちに知られている言語ではありませんでした。 しかし、SCSS記法の誕生により、2010年の秋ごろから「Sassは素晴らしいよ！」という声を聞くようになりました。当時はまだまだ日本語の情報は少なかったのですが、Sassの利便性に感動したクリエイターたちのブログやSNS、口コミにより、徐々にユーザーを増やしていきました。メジャーなブログや動画解説サイトでも取り上げられるようになり、Sassの話題を見かける機会がとても増えました。2012年5月には、Sassでは国内初となる電子書籍も発売され、Sassを導入している人はさらに増えたと思います。

そして、昨今では、Sassを取り扱ったセミナーなども開催されて多くの人が集まるようになり、注目度は非常に高まっているといっていいでしょう。 開発環境側でも動きがあり、オーサリングツールの代表である「Adobe Dreamweaver CC」がSassとLESS（レス）に対応しました。 こういった動きからも、今後のWebサイト制作において、Sassを代表とするメタ言語の習得が必須となりうるレベルになってきています。

ヒント*17

Haml（ハムル）はHTML/XHTMLを生成するためのマークアップ言語で、インデントや簡略構文によって簡潔な記述が行えます。

さまざまなCSSメタ言語とSass

Sassは CSSを拡張したメタ言語という話をしましたが、このメタ言語はSass以外にもたくさんあります。 Sassをご存じの方なら名前を聞いたことがあるかもしれませんが、Sassと同じくらい有名な「LESS」や、SASS記法に近い「Stylus（スタイラス）」が代表的なCSSのメタ言語です。

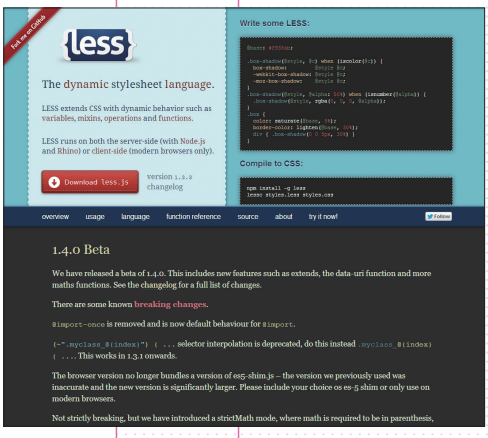


図2　LESS　http://lesscss.org/

LESS

「LESS」は、Bootstrapで採用されていたため、Sassと同じくらい有名なCSSメタ言語です 図2 。Sassとの比較記事も多く見かけます。書き方はSassと似ており、機能としてはSassより少ないですが、JavaScriptで実装しているため、Rubyのインストールなどの作業が必要なく、コンパイルもJavaScriptを読み込むことでクライアントサイドでの動的なCSSの生成ができるなどの特徴があります。 JavaScriptベースだったり気軽に使いやすかったりという点で使っている方が多いように見受けられます。

図3　Stylus
http://learnboost.github.io/stylus/

Stylus

「Stylus」は、SassやLESSよりも後発のCSSメタ言語で、Sassの最初の記法であるSASS記法に似ており、記述を極限まで簡素化して書くことができるのが特徴です 図3 。ただ、SASS記法との大きな違いとして、CSSの書式も使うことができます。 また、Stylusはもともと Node.jsのモジュールの1つとして提供されたので、Node.jsベースのフレームワークでも使われています。

その他のCSSメタ言語

大半のCSSメタ言語は、Sassを覚えていればある程度すんなり使えるものが多いため、気になったCSSメタ言語があれば、実際に試してみるといいでしょう。

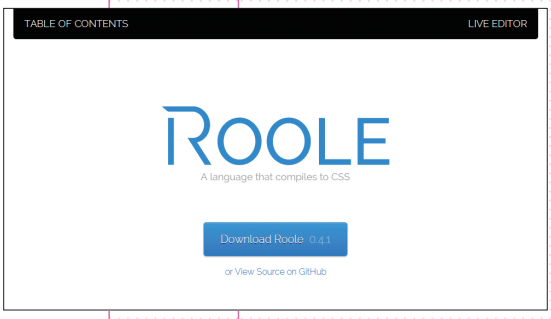


図 4　Roole　http://roole.org/

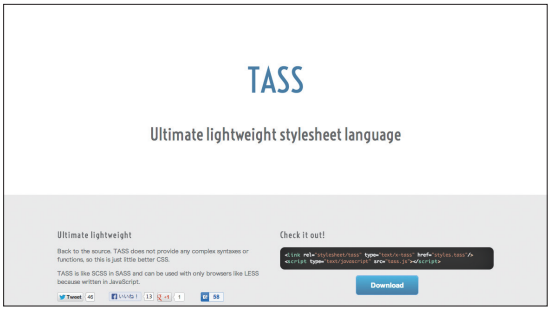


図 5　TASS　http://cho45.github.io/tasscss/

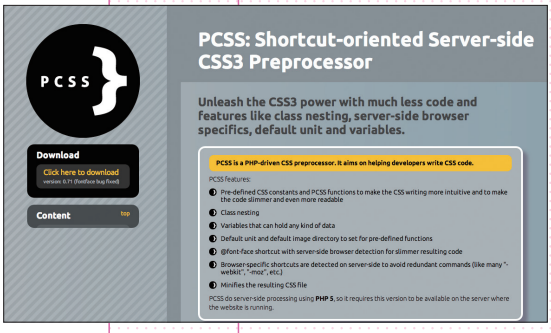


図 6　PCSS　http://pcss.wiq.com.br/

図 7　CSS Crush　http://the-echoplex.net/csscrush/

それでも Sass

ここで紹介したCSSメタ言語は、Sassよりも後発のものが多いため、単純な機能だけ見たらSassよりも優れている部分があったり、インストールが簡単だったりしますが、本書でSassを取り上げたのには理由があります。 第一にSassはユーザー数が多い点が挙げられます。ユーザー数が多いということはそれだけ支持されている証拠ですし、困ったときに相談したり、Web上を検索して解決策が見つかる可能性も高くなります。 第二に、開発が活発に進んでいます。せっかく覚えても開発が止まってしまっては先行きが怪しくなってしまいますから、勉強する上では重要なポイントでしょう。仮に開発元が開発をやめてしまっても、人気が高いものは有志の方々が引き継いでくれてプロジェクト自体が存続しやすいはずです。 他にも、優秀なGUIツールや、Compassを代表とするフレームワークの充実もSassを選ぶ大きな理由の1つといえます。本書では、Compassに関しても、基本的な使い方や便利な機能に関して説明しているので、フレームワークの魅力にも触れていただけます(P.245)。

他にもSassの魅力はたくさんありますが、こういった理由からSassを選んでいます。

①-2 Sassを導入する前の疑問や不安

ここでは、導入のハードルが高く感じられたり、「いざ導入してもその後実務では使えないのでは？」といった、導入する前の不安や疑問などを解決していきます。

Rubyの知識は必要？黒い画面も使わないとダメなの？

Rubyの知識は不要です！

CUIでコンパイルするには、RubyとSassを自身のパソコンにインストール[*18]する必要がありますが、Rubyのコードを書くことはありません。Rubyはインストーラーが用意されているので、通常のソフトウェア同様インストーラーをダウンロードしてから、数回クリックするだけでインストールが完了します。また、インストール後の作業も特にないので、一度インストールしてしまえば、その後は特にすることはありません。

ヒント*18

Rubyは、Macにはもともとインストールされているので、Windowsのみ必要です。

黒い画面も使わなくとも大丈夫です！

GUIツールを使えば、黒い画面（Windowsではコマンドプロンプト、Macではターミナル）を一切使わずに、Sassを利用することも可能です。

また、GUIツールを使わない場合は、必ず黒い画面を使う場面が出てきますが、一度環境を構築してしまえば、その後に面倒なことや難しいことはまったくありません。

黒い画面が必要となるのは、主に次の場面です。

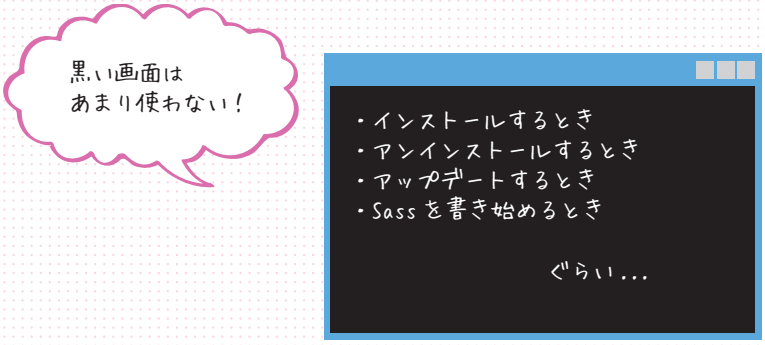
黒い画面を使う主な場面

- Sassのインストール作業
- Sassのアンインストール作業
- Sassのアップデート、gemのアップデート
- Sassを書き始めるとき

当然、インストール作業は初回のみなので、以後は必要ありません。アップデート作業は、そこまで頻繁にやる必要はないので、数カ月に一度行えば基本的には十分ですが、可能な限り最新版を使いたい場合は、こまめにアップデートを行いましょう。

Sassを書き始めるときは、必ず黒い画面を使う必要がありますが、Sassを書き始めるたびに黒い画面を立ち上げてコマンドを入力するのは非常に面倒な作業です。そこで、本書ではコマンドプロンプト用のバッチファイル、ターミナル用のシェルスクリプトを用意する方法をお勧めしています。 この、バッチファイル／シェルスクリプトをダブルクリックするだけで、Sassを書く準備が完了するため便利です。その方法に関しては第2章の「バッチファイル／シェルスクリプトで簡単にコマンドを実行する」(P.68) で説明しています。

普段から黒い画面に触れる機会はあまりないと思いますので、GUIが主流な昨今では、黒い画面には抵抗や苦手意識を感じる方も多いと思います。しかし、本書を読み進めていただければ、黒い画面に関してはほとんど覚える必要がないことがわかると思います。



余談ですが、プログラマやシステムエンジニアに使われていた黒い画面は、現状フロントエンドの開発環境にも必要不可欠になってきています。 Sassはもちろんのこと、Haml、Git、Node.js、Stylus、Grunt、Bowerなど……これらのツールは基本的に黒い画面を使います。 今後も優秀なツールはたくさん出てくると思いますが、最初はCUIでのコマンドベースの操作がほとんどです。 その後一定の人気を獲得するとGUIのツールが作られるパターンが多いです (Sassは優秀なGUIツールがすでに多数あります)。

Sassは、簡単なコマンドを覚えるだけで使えるので、これから始めるのにとても適しています。黒い画面が苦手で使用を敬遠していた方も、本書をきっかけに少しだけ黒い画面と仲良くなりましょう。

公式サイトは英語だし、日本語の情報が少ないのでは？

最近では、日本語の情報もかなり増えてきています！ 確かに2、3年前までは、日本語の情報も少なく、まともなGUIツールなどもほとんどなかったため、せっかく覚えようと思っても途中で挫折してしまうケースも多かったと思いますが、今では、多くのブログやサイトでSassを含むCSSメタ言語の紹介記事があるので、公式サイトを見なくとも特に困ることはないでしょう。

しかし、それでも公式サイトの情報は欠かせません。英語のドキュメントを読むのは、多少英語ができてもなかなかつらいものがありますが、現在、GitHub上のグループで、公式サイトの翻訳も進められています 図8 。すでにかなりの内容が翻訳されているので、公式サイトの情報に興味がある場合は、一読してみるといいでしょう。

図8 GitHubで、日本人の複数の翻訳者により、公式サイトの翻訳が進められている
https://github.com/enja-oss/Sass

エディタやオーサリングツールはそのまま使えるの？

Sassは、CSSと同じテキストデータなので、使い慣れているテキストエディタやDreamweaverなどを使うことができます。ただ、何の設定もしないと認識されなかったり、シンタックスハイライトやコードヒントが表示されなくなったりしてしまうので、多少設定を変更する必要があります。

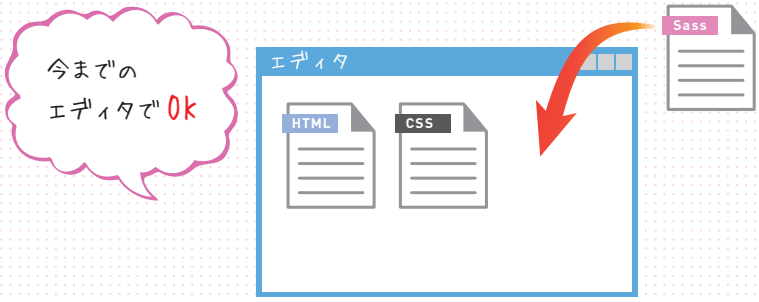
テキストエディタ

多くのテキストエディタの場合、拡張子ごとに設定を変更できたりすると思いますので、CSSで使っていた設定をそのままSassで使えば基本的に問題なく使うことが可能です。

Adobe Dreamweaver

Dreamweaverの場合は、拡張子が変わってしまうとCSSファイルとして認識されなくなります。そのため、最初に拡張子を追加するなどの作業をする必要があります。

なお、Dreamweaver CC からは、Sassに対応したため、そのまま利用することができます。



Sassで使えるテキストエディタやIDE（統合開発環境）に関しては、第7章の「Sassが使えるテキストエディタ」（P.304）にて紹介しています。

運用時にSassを使うのは難しいから、Sassは導入できない？

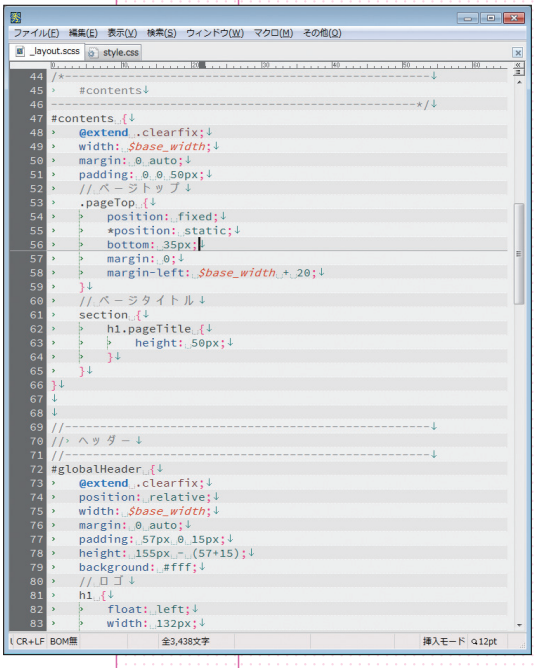
Sassのハードルを上げている理由として、「社内で自分だけがSassを使うことはできない」または、「フリーランスやSOHOでやっていて、Sassファイルを納品するわけにいかない」といった事情が挙げられます。確かに、社内のガイドラインが決まっている場合は、会社全体のコーディングルールから見直さなければならないケースもありますが、そこまでルールが厳しくない場合は、新規でコーディングをするときだけSassを使い、運用フェーズに入ってからは、Sassを使わないで従来通りCSSファイルを編集して使うことも可能です。

Sassはコンパイルする際に、CSSファイルのフォーマットを選ぶことができます*19。このフォーマットの1つにexpandedというものがあり、この指定をすると、普通に書いたCSSのように書き出されます。

ヒント*19

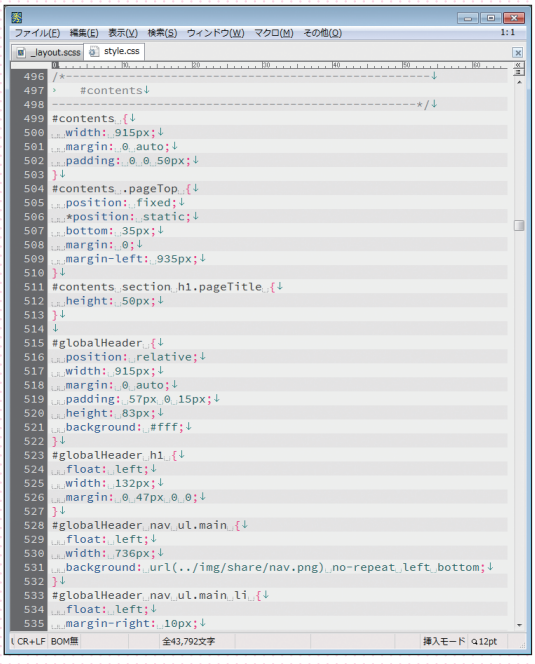
書き出されるCSSのフォーマット（アウトプットスタイル）は4種類あり、自由に選ぶことができます。

詳しくは → P.58



```
/*----------------------------------------------------------↓
  #contents↓
----------------------------------------------------------*/↓
#contents {↓
  @extend .clearfix;↓
  width: $base_width;↓
  margin: 0 auto;↓
  padding: 0 0 50px;↓
  // ページトップ↓
  .pageTop {↓
    position: fixed;↓
    *position: static;↓
    bottom: 35px;↓
    margin: 0;↓
    margin-left: $base_width + 20;↓
  }↓
  // ページタイトル↓
  section {↓
    h1.pageTitle {↓
      height: 50px;↓
    }↓
  }↓
}↓
↓
↓
//----------------------------------------------------------↓
// ヘッダー↓
//----------------------------------------------------------↓
#globalHeader {↓
  @extend .clearfix;↓
  position: relative;↓
  width: $base_width;↓
  margin: 0 auto;↓
  padding: 57px 0 15px;↓
  height: 155px - (57+15);↓
  background: #fff;↓
  // ロゴ↓
  h1 {↓
    float: left;↓
    width: 132px;↓
```

図9　コンパイル前のSassファイル



```
/*----------------------------------------------------------↓
  #contents↓
----------------------------------------------------------*/↓
#contents {↓
  width: 915px;↓
  margin: 0 auto;↓
  padding: 0 0 50px;↓
}↓
#contents .pageTop {↓
  position: fixed;↓
  *position: static;↓
  bottom: 35px;↓
  margin: 0;↓
  margin-left: 935px;↓
}↓
#contents section h1.pageTitle {↓
  height: 50px;↓
}↓
↓
#globalHeader {↓
  position: relative;↓
  width: 915px;↓
  margin: 0 auto;↓
  padding: 57px 0 15px;↓
  height: 83px;↓
  background: #fff;↓
}↓
#globalHeader h1 {↓
  float: left;↓
  width: 132px;↓
  margin: 0 47px 0 0;↓
}↓
#globalHeader nav ul.main {↓
  float: left;↓
  width: 736px;↓
  background: url(../img/share/nav.png) no-repeat left bottom;↓
}↓
#globalHeader nav ul.main li {↓
  float: left;↓
  margin-right: 10px;↓
```

図10　SassをexpandedスタイルでコンパイルしたCSSファイル

コンパイル前後のソースコード 図9 図10 を見比べていただくとわかるように、Sassをコンパイルしても通常のコメント（/* ～ */）は残りますし、CSSファイルの可読性も特別悪くなるということはありません。Sassの1行コメント（// ～）はコンパイルすると必ず消えてしまうのでできるだけ控えるようにし、CSS

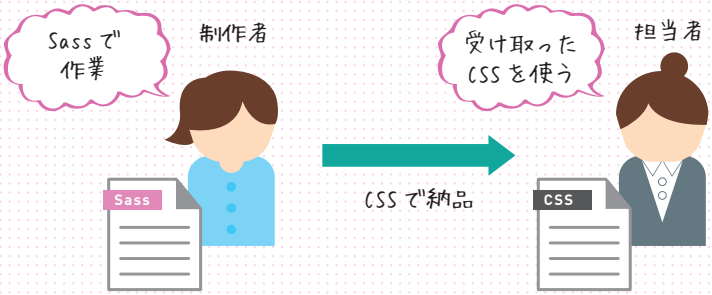
ファイルの分割ルールなども合わせておくなどの対応をすれば、後から別の人がメンテナンスできないという状態は避けられます。

Sassの魅力の1つであるメンテナンス性の向上が活用できなくなってしまいますが、その辺りは割り切ってしまってもいいでしょう。

また、フリーランスなどで多くの企業とかかわる場合、各社のガイドラインに従う必要がありますが、アウトプットスタイルをexpandedにすれば問題ないケースも多いです。納品時にも、Sassで書いてる旨は特に伝えず普通に納品してしまって問題ないと思います（もちろん、クライアントに応じてケース・バイ・ケースですが）。また、それとは逆に、SassやLESSなどのメタ言語でのコーディングを依頼されるケースも徐々に発生するかもしれません。メタ言語の経験がないからといって断ってしまうのは非常にもったいないので、覚えておいて損をすることはないでしょう。

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？

先ほどは、CSSファイルで運用するというやり方でしたが、納品時のCSSファイルにはクライアントや更新担当の方には触らないようにしてもらい、別途、担当者用のCSSファイルを用意するというやり方もあります。 この方法なら、普段の更新は担当者用のCSSファイルに記述してもらい、大き目な修正や追加などでCSSをガッツリ修正・追加する場合だけSassを使うようにして、場合によっては担当者用のCSSをその際にマージしてしまうことも可能です。



「Sassだと関係者に使えない人がいるから自分だけ覚えても使えない……」と決め込まず、担当者や関係者間で話し合ってうまく運用ルールを決めていければ、十分に使える場面はあるでしょう。

①-3 何はともあれSassを触ってみよう

実際に自身のパソコン上でSassを使うにはインストールなどの作業が必要ですが、その前に軽くWeb上でSassを触ってみて、感触をつかんでみましょう。

本節のサンプルコード

http://book.scss.jp/code/c1/03.html

ここまで読んで、Sassのメリットが多少は見えてきたと思います。次章以降は、実際にSassを使ってコーディングをするためにインストール作業などを説明していきますが、その前に少しだけSassに触れてみましょう。ちょっと試すだけなら特別な環境は必要なく、インターネットにつながっているPCならブラウザ上（IEは8以上）で簡単に試すことができます。

まずは、ブラウザを立ち上げ、アドレスバーに次のURLを入力してください。

http://sass-lang.com/try.html *20

ヒント*20

本書公式サポートサイトにリンクを用意してあります。
http://book.scss.jp/link/

もしくはGoogleなどで「sass try」で検索して、「Try Online - Sass」というタイトルのページに移動します 図11。

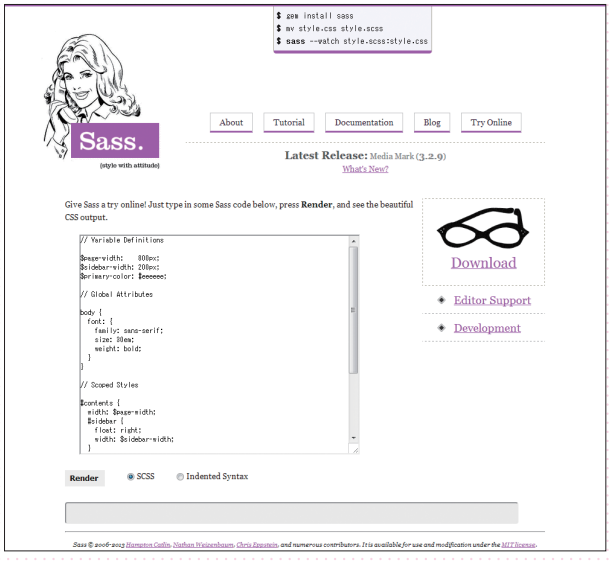


図11 Sass公式サイトのSassをオンライン上で試すことができるページ（2013年8月現在）

このページは、Sass 公式サイトのコンテンツの1つで、テキストエリアにSassを記述して［Render］ボタンをクリックするだけで、SassをCSSにコンパイルしてくれます。

ページに移動すると初期値でサンプルソースが入力されていますが、最初は何だかわからないと思いますので、いったん消去してから、簡単な下記のソースコードを書いてみましょう。 その際、改行やインデントなどは皆さんそれぞれが書きやすい方法で書いて問題ありません[*21]。

ヒント*21

手入力が面倒な方は、本書公式サポートサイトからコピー＆ペーストしてください。
http://book.scss.jp/code/c1/03.html

Sass

```
#main {
  width: 600px;
  p {
    margin: 0 0 1em;
    em {
      color: #f00;
    }
  }
}
```

このソースコードを見てもらうと、#main 内にp要素のスタイルが書かれており、さらにp要素内でem要素のスタイルが書かれているのがわかると思います。これは、CSSではできなかった書き方をしていて、Sassの機能では一番使う、ルールのネスト（入れ子）という機能を使って書いています。詳しくは第3章の「ルールのネスト」（P.88）で説明しているので、あまり難しく考えずサンプルソースのまま書いてから、［Render］ボタンをクリックしてみましょう。 そうすると、次のCSSが表示されると思います。

CSS（コンパイル後）

```
#main {
  width: 600px; }
  #main p {
    margin: 0 0 1em; }
    #main p em {
      color: #f00; }
```

CSSでも入れ子のような状態で書き出されてしまうので、ちょっと可読性が悪いかも知れませんが、次のCSSと同様です。

CSS

```
#main {
    width: 600px;
}
#main p {
    margin: 0 0 1em;
}
#main p em {
    color: #f00;
}
```

この簡単なサンプルだけでは、Sassの魅力はあまり見えないかもしれませんが、いつもだったら #main内のp要素とem要素にスタイルを当てるために、毎回親のセレクタから書く必要がありました。それが、Sassの機能の1つであるネストを使うことで、記述が簡略化され効率的に書いていくことができます。

次に、「変数」という機能を使ったソースコードを書いてみましょう。先ほどと同じように、テキストエリアに次のソースコードを書きます。

Sass

```
$red: #ff1122;

.notes {
  color: $red;
}
#main {
  .notesArea {
    border: 1px solid $red;
  }
}
```

$（ダラー）から始まるCSSでは見覚えがない記号を使っていますが、これがSassの変数という機能で、このようにあらかじめ変数に値を定義しておけば、変数の値を好きな場所から参照することができます。 同じように [Render] ボタンをクリックすると、次のようなCSSが表示されます。

CSS（コンパイル後）

```
.notes {
  color: #ff1122; }

#main .notesArea {
  border: 1px solid #ff1122; }
```

最初に書いた「$red: #ff1122;」が、それぞれのプロパティに適用されています。実際にコーディングする際には、同じ値を多くのプロパティで使うことがありますが、変数を使うと同じ値を参照してくれるので、後から変更が入っても1カ所直せば他もすべて同じ値に変更することができます。変数に関しては第3章の「変数（Variables）」（P.98）で詳しく説明しています。

ここでは公式サイトを利用しましたが、同様のWebサービスで「SassMeister」というサービスもあります 図12。

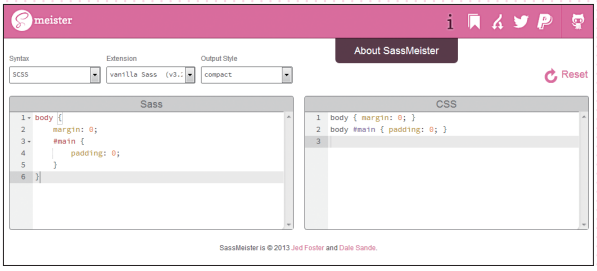


図12 SassMeister | The Sass Playground! http://sassmeister.com/

このSassMeisterは使い勝手がよく、リアルタイムにSassをCSSにコンパイルしてくれたり、フレームワークの選択やアウトプットスタイルも選ぶことができます。ちょっと試しに書いたり、フレームワークを利用してみたい場合など、公式サイトより便利に使うことができるので、気になった方は試してみてはいかがでしょうか。

Sassに対応しているソーシャルコーディングサービス

先ほど紹介した公式サイトやSassMeiste以外にも、ブラウザ上で簡単にSassを試すことができるソーシャルコーディングサービスもあります。これらのサービスは、Web上でHTML5、CSS、JavaScriptを書いてその場で実行することができるので、より実際の感じがつかめると思います。 中でもjsdo.it（http://jsdo.it/）は日本語サイトなので、ちょっと試してみるには一番とっつきやすいと思います 図13。

- **jsdo.it**
  http://jsdo.it/

図13 jsdo.it

- **Create a new Fiddle - jsFiddle**
  http://jsfiddle.net/

- **Pens picked by the Editors of CodePen**
  http://codepen.io/

- **HTML5, CSS3, JS Demos, Creations and Experiments | CSSDeck**
  http://cssdeck.com/

実際にSassに触れてみて、いかがでしたか？　ここで紹介した機能は、Sassの機能のほんの一部に過ぎませんが、CSSではできなかったことが、Sassを使うことでできるようになったのが、何となくおわかりいただけたかと思います。これらの便利な機能を使ってコーディングができるように、次章では、自身の環境にSassをインストールする方法を説明します。