

gulp + sass で目指せ倍速コーディング

東区フロントエンド勉強会 2015年 第1回

本編



エクスコード株式会社

<http://excode.jp>

フロントエンド エンジニア

末包 俊道 (すえかね)

第1章 Sass

第1章 Sass

1. 概要
2. 公式サイト
3. 導入のメリット・デメリット
4. Sass の記法 (SCSS / SASS)
5. 主な機能

第2章 gulp

1. 概要
2. 公式サイト
3. ハンズオン

第3章 応用

1. いろんなプラグインを使ってみよう
2. gulpfile.js を分割して管理しやすくしよう



1. 概要

CSS を効率的に書くためのメタ言語と、その言語で記述したファイルをCSSファイルへ変換するソフトウェア。

CSSにはない変数や配列を扱うことができ、セレクタのネスト（入れ子）により見通しの良いコーディングが可能になります。

2. 公式サイト



<http://sass-lang.com>

3. 導入のメリット・デメリット

メリット

- ・ コードの見通しが良くなる
- ・ 変数を使える
- ・ 簡易な関数を使える
- ・ よく使うコードを定義し再利用できる
- ・ header, body, side, footer など個別ファイルに分割して管理できる

デメリット

- ・ Sass の流儀に合わせてマークアップする必要がある
- ・ コンパイルする環境を整えておく必要がある
- ・ Windows 環境においては標準で Ruby がインストールされていないため導入のハードルがやや高い
- ・ Mac OSX のバージョンによっては旧バージョンの Ruby がインストールされており、Ruby のバージョンアップに躊躇する

4. Sass の記法 (SCSS記法)

SCSS 記法 (.scss)

```
@mixin opacity($string) {
  $opacityIE: $string * 100;
  opacity: $string;
  filter: alpha(opacity=$opacityIE);
}

.block {
  a {
    display: block;
    background-color: rgba(black, .1);
    color: blue;
    text-decoration: none;
    &:hover {
      color: red;
      text-decoration: underline;
      img {
        @include opacity(.8);
      }
    }
  }
}
```

- ・ ネットでよく見かける記法
- ・ CSS を入れ子にした構造
- ・ CSS を混在しても大丈夫

4. Sass の記法 (SASS記法)

SASS 記法 (.sass)

```
=opacity($string)
  $opacityIE: $string * 100
  opacity: $string
  filter: alpha(opacity=$opacityIE)

.block
  a
    display: block
    background-color: rgba(black, .1)
    color: blue
    text-decoration: none
    &:hover
      color: red
      text-decoration: underline
  img
    +opacity(0.8)
```

- ・ Sass本来の記法
- ・ セミコロン不要
- ・ {} 波カッコ不要
- ・ @mixin は =
- ・ @include は +
- ・ インデントで親子関係を示す
- ・ コロン「:」の後に必ず半角スペースが必要

4. Sass の記法 (SCSS/SASSの比較)

SCSS 記法 (.scss)

```
@mixin opacity($string) {
  $opacityIE: $string * 100;
  opacity: $string;
  filter: alpha(opacity=$opacityIE);
}

.block {
  a {
    display: block;
    background-color: rgba(black, .1);
    color: blue;
    text-decoration: none;
    &:hover {
      color: red;
      text-decoration: underline;
      img {
        @include opacity(.8);
      }
    }
  }
}
```

SASS 記法 (.sass)

```
=opacity($string)
  $opacityIE: $string * 100
  opacity: $string
  filter: alpha(opacity=$opacityIE)

.block
  a
    display: block
    background-color: rgba(black, .1)
    color: blue
    text-decoration: none
    &:hover
      color: red
      text-decoration: underline
      img
        +opacity(0.8)
```

本勉強会では SASS 記法で進めます

SCSS 記法 (.scss)

```
@mixin opacity($string) {
  $opacityIE: $string * 100;
  opacity: $string;
  filter: alpha(opacity=$opacityIE);
}

.block {
  a {
    display: block;
    background-color: rgba(black, .1);
    color: blue;
    text-decoration: none;
    &:hover {
      color: red;
      text-decoration: underline;
      img {
        @include opacity(.8);
      }
    }
  }
}
```

SASS 記法 (.sass)

```
=opacity($string)
  $opacityIE: $string * 100
  opacity: $string
  filter: alpha(opacity=$opacityIE)

.block
  a
    display: block
    background-color: rgba(black, .1)
    color: blue
    text-decoration: none
    &:hover
      color: red
      text-decoration: underline
      img
        +opacity(0.8)
```

5. 主な機能

1. コメントアウト
2. 変数と計算式
3. ネスト（入れ子）
4. & 親セレクタ参照
5. @mixin (=) と @include (+)
6. #{…} 変数に入った値を文字列をして出力
7. @extend 定義されたスタイルの読込
8. @import 外部ファイル読込
9. @if @else 条件分岐
10. @for ループ
11. @each ループ
12. 組み込み関数（一部紹介）

5-1. コメントアウト

SASS (.sass)

```
/* text */ ← CSSにも反映されるコメント
// text    ← この行はCSSに反映されない
//         ← この行以降のネストはCSSに反映されない
  text

h1
  /* コメント */
  font-size: 24px
  a
    //color: red
    color: blue

h2
  font-size: 21px
//
  a
    color: blue
```

CSS (.css)

```
h1 {
  /* コメント */
  font-size: 24px;
}

h1 a {
  color: blue;
}

h2 {
  font-size: 21px;
}
```

5-2. 変数と計算式

SASS (.sass)

```
$size: 13px
$base-color: white

h1
  font-size: $size * 2
  color: $base-color

p
  font-size: $size

small
  font-size: $size / 2
  color: $base-color * 0.3

small.floor
  font-size: floor($size / 2)
```

- ・ 加算, 減算 ... $x + n$, $x - n$
- ・ 積算, 除算 ... $x * n$, x / n
- ・ 切り上げ ... `ceil()`
- ・ 切り捨て ... `floor()`
- ・ 四捨五入 ... `round()`

CSS (.css)

```
h2 {
  font-size: 26px;
  color: white;
}

p {
  font-size: 13px;
}

small {
  font-size: 6.5px;
  color: #4d4d4d;
}

small.floor {
  font-size: 6px;
}
```

5-3. ネスト (入れ子)

SASS (.sass)

```
.header
  position: relative
  max-width: 360px
  height: 44px
  margin: 0 auto
```

1階層目

```
.logo
  display: inline
```

2階層目

```
  a
    position: absolute
    top: 0
    left: 0
    display: block
    width: 60px
    height: 44px
```

3階層目

CSS (.css)

```
.header {
  position: relative;
  max-width: 360px;
  height: 44px;
  margin: 0 auto;
}

.header .logo {
  display: inline;
}

.header .logo a {
  position: absolute;
  top: 0;
  left: 0;
  display: block;
  width: 60px;
  height: 44px;
  text-decoration: none;
}
```

5-4. 親セクタ参照

SASS (.sass)

```
a
  &:hover
  ...

  &:before
  ...

section
  &>div      子セクタ
  ...

  &~div      間接セクタ
  ...

  &+div      隣接セクタ
  ...

p
  div &
    &.class
  ...
```

CSS (.css)

```
a:hover {
  ...
}

a:before {
  ...
}

section > div {
  ...
}

section ~ div {
  ...
}

section + div {
  ...
}

div section p.class {
  ...
}
```

5-5. @mixin (=) と @include (+)

SASS (.sass)

```
$baseFontSize: 10px
=font-size($string) @mixin
  font-size: $baseFontSize * $string
a
  +font-size(1.2) @include
```

【参考】 SCSS の場合は以下の様に記述します

```
$baseFontSize: 10px;
@mixin font-size($string) {
  font-size: $baseFontSize * $string;
}
a {
  @include font-size(1.2);
}
```

CSS (.css)

```
a {
  font-size: 12px;
}
```


5-6. #{…} 変数に入った値を文字列をして出力

SASS (.sass)

```
=opacity($string) @mixin
  $opacityIE: $string * 100

  opacity: #{ $string }
  filter: alpha(opacity=#{ $opacityIE })

a
  &:hover
    +opacity(0.8) @include
```

CSS (.css)

```
a:hover {
  opacity: 0.8;
  filter: alpha(opacity=80);
}
```

5-7. @extend 定義されたスタイルの読込

SASS (.sass)

```
.buttonClass  
  display: block  
  color: white
```

クラス

```
%buttonPlaceholder  
  display: block  
  color: red
```

プレースホルダー

```
a  
  &.searchButton  
    @extend .buttonClass  
    width: 50%
```

クラスを読み込む

```
a  
  &.cartButton  
    @extend %buttonPlaceholder  
    width: 80%
```

プレースホルダーを読み込む

CSS (.css)

```
.buttonClass {  
  display: block;  
  color: white;  
}
```

クラス

プレースホルダーはCSSに書き出されません

```
a.searchButton {  
  display: block;  
  color: white;  
  width: 50%;  
}
```

読み込まれたクラス

```
a.cartButton {  
  display: block;  
  color: red;  
  width: 80%;  
}
```

読み込まれたプレースホルダー

5-8. @import 外部ファイル読込

SASS (style.sass)

```
@import header 外部ファイル化したヘッダーを読み込む

.body
  p
    margin: 1em auto
  a
    color: blue
```

SASS (_header.sass) ヘッダー

```
.header
  p
    &.logo
      width: 160px
      float: left
    a
      img
        width: 100%
```

CSS (style.css)

```
.header p.logo {
  float: left;
  width: 160px;
}

.header p.logo a img {
  width: 100%;
}

.body p {
  margin: 1em auto;
}

.body p a {
  color: blue;
}
```

5-9. @if @else 条件分岐

SASS (.sass)

```
$type: sample  
  
p  
  @if $type == sample  
    color: red  
  
  @else if $type == test  
    color: blue  
  
  @else  
    color: black
```

CSS (.css)

```
p {  
  color: red;  
}
```

5-10. @for ループ

SASS (.sass)

```
li
  position: absolute
  @for $i from 1 through 3
    &:nth-child(#{ $i })
    left: 40px * $i
```

CSS (.css)

```
li {
  position: absolute;
}

li:nth-child(1) {
  left: 40px;
}

li:nth-child(2) {
  left: 80px;
}

li:nth-child(3) {
  left: 120px;
}
```

5-11. @each ループ

SASS (.sass)

```
$images: apple banana orange
```

配列を定義 (スペースまたはカンマ区切り)

```
p
  @each $img in $images
    &.#{$img}
      background-image: url(/img/#{$img}.png)
```

CSS (.css)

```
p.apple {
  background-image: url(/img/apple.png);
}

p.banana {
  background-image: url(/img/banana.png);
}

p.orange {
  background-image: url(/img/orange.png);
}
```

5-12. 組み込み関数

SASS (.sass)

```
$color: #3366FF

p
  color: $color

  // 明度 lighten, darken
  &.lighten
    color: lighten($color, 20%)


  // 彩度 saturate, desaturate
  &.saturate
    color: saturate($color, 20%)


  // RGB+アルファ
  &.rgba
    color: rgba($color, 0.8)


$colors: red green blue

p
  &.color2
    color: #{nth($colors,2)}
```

CSS (.css)

```
p {
  color: #36f; 
}

p.lighten {
  color: #99b3ff; 
}

p.saturate {
  color: #1f5cff; 
}

p.rgba {
  color: rgba(51, 102, 255, .8);
}

p.color2 {
  color: green;
}
```

おまけ① BEM (Block, Element, Modifier) 記法を使いたい

SASS (.sass)

```
.hoge
  &__element1
    width: 100px
  &_modifier
    color: red
  &__element2
    width: 50%
  &_modifier
    color: blue
```

CSS (.css)

```
.hoge__element1 {
  width: 100px;
}

.hoge__element1_modifier {
  color: red;
}

.hoge__element2 {
  width: 50%;
}

.hoge__element2_modifier {
  color: blue;
}
```


おまけ② タイプ数を少なくしたい

SASS (.sass)

```
=w($s: auto)
  width: unit($s)

@function unit($s)
  ...
  @return $rs

.container
  +relative
  +pl(2em)
  +left
  &:after
    +absolute
    +t(0)
    +l(0)
    content: ''
    +block
    +w(20)
    +h(80%)
```

unit() という @function(独自関数)を作り
値に em や % が付いている場合はその単位、
値が数値の場合は px を返すものを用意。

unit() 関数は少し長いので、ここでは省略して
説明しています。

CSS (.css)

```
.container {
  position: relative
  padding-left: 2em;
  text-align: left;
}

.container:after {
  position: absolute;
  top: 0;
  left: 0;
  content: ''
  display: block;
  width: 20px;
  height: 80%;
}
```

第2章 gulp

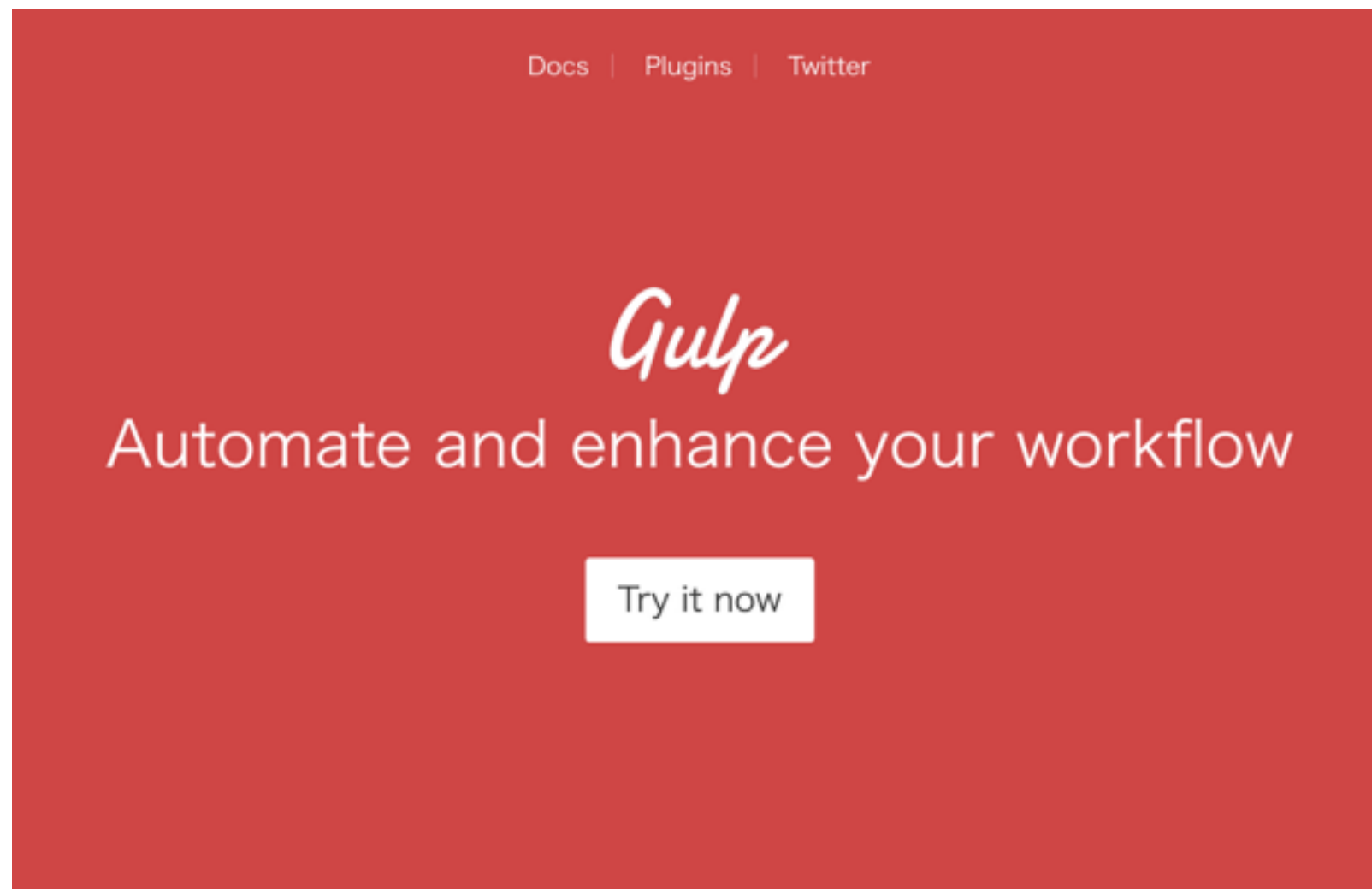


1. 概要

Node.js も用いたタスクの自動化ツール。

本勉強会では、ファイルを監視し、予め設定ファイルで定義したタスクを自動的に実行させます。

2. 公式サイト



<http://gulpjs.com/>

3. ハンズオン

<下準備>

1. 今回のディレクトリ構成
2. ディレクトリを準備
3. package.json を生成

<gulp設定>

4. 必要なプラグインを追加
5. gulpfile.js を作成

<gulp実行>

6. gulp sass:watch でファイルの変更を監視

3-1. 今回のディレクトリ構成

```
lesson
| <開発用ディレクトリ>
├ develop
|   | <.sass ディレクトリ>
|   └ sass
|       └ style.sass
|
| <納品ファイル用ディレクトリ>
├ html
|   | <.css ディレクトリ(自動生成)>
|   └ css
|       └ style.css
|
|   └ index.html
|
└ 上記以外でgulpが必要とするファイル
```

3-2. ディレクトリを準備 (1/2)

今回は説明の都合上、デスクトップにハンズオン用のディレクトリを作成していただきます。

Windows の方

デスクトップに移動

```
> cd Desktop  
または  
> cd C:\Users\ユーザー名\Desktop
```

新しくディレクトリ（フォルダ）を作成

```
> mkdir lesson
```

作成したディレクトリに移動

```
> cd lesson
```

Mac の方

デスクトップに移動

```
$ cd ~/Desktop
```

新しくディレクトリ（フォルダ）を作成

```
$ mkdir lesson
```

作成したディレクトリに移動

```
$ cd lesson
```

3-2. ディレクトリを準備 (2/2)

Windows の方

ディレクトリを作成、css用ディレクトリは自動生成です

```
> mkdir develop\sass  
> mkdir html
```

index.html と style.sass も作っておきましょう

```
> type nul develop\sass\style.sass  
> type nul html\index.html
```

Mac の方

ディレクトリを作成、css用ディレクトリは自動生成です

```
$ mkdir develop  
$ mkdir develop/sass  
$ mkdir html
```

index.html と style.sass も作っておきましょう

```
$ touch develop/sass/style.sass  
$ touch html/index.html
```


3-3. package.json を生成 (1/2)

Windows の方

package.json を生成

```
> npm init
```

対話形式で設定 (enterを押すと省略して事項へ進む)

```
name (lesson)
...
{
  ...
}
```

これで良いか確認されるので enter

```
Is this ok? (yes)
```

package.jsonが生成されていることを確認

```
> dir
<DIR>      .
<DIR>      ..
<DIR>      develop
<DIR>      html
          256 package.json
```

Mac の方

package.json を生成

```
$ npm init
```

対話形式で設定 (enterを押すと省略して事項へ進む)

```
name (lesson)
...
{
  ...
}
```

これで良いか確認されるので enter

```
Is this ok? (yes)
```

package.jsonが生成されていることを確認

```
$ ls
develop      html      package.json
```

3-3. package.json を生成 (2/2)

package.json の中身を見てみよう

```
{
  "name": "lesson",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" &&
exit 1"
  },
  "author": "",
  "license": "ISC",
}
```

現在のディレクトリ構成

```
lesson
| <開発用ディレクトリ>
├ develop
|   | <.sass ディレクトリ>
|   └ sass
|       └ style.sass
|
| <納品ファイル用ディレクトリ>
├ html
|   |
|   └ index.html
└ package.json
```

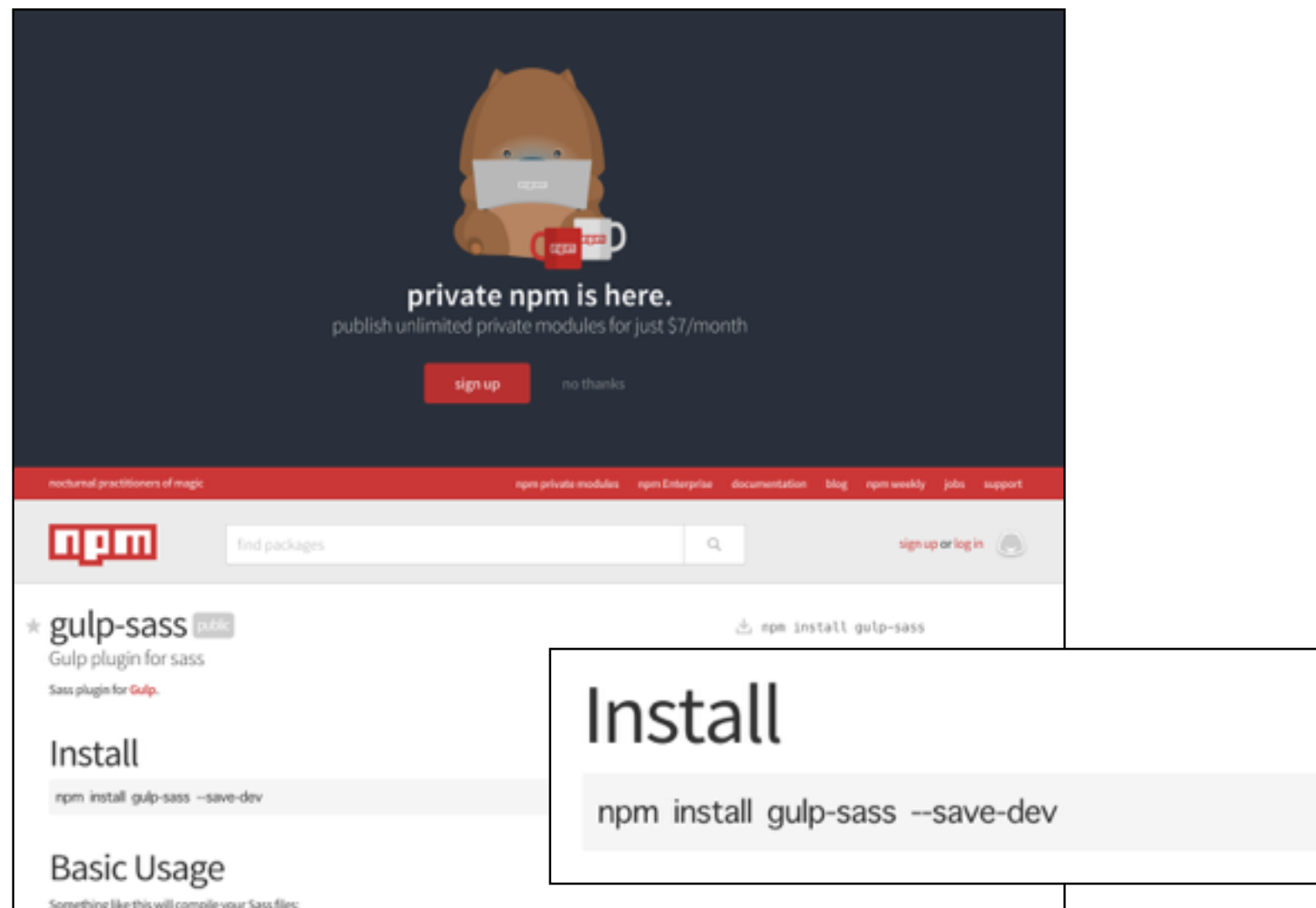
これで下準備は完了です

3-4. 必要なプラグインを追加



<http://gulpjs.com/plugins/>

3-4. 必要なプラグインを追加



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

後ほどこのコマンドを実行します

3-4. 必要なプラグインを追加

Windows の方

gulp を追加

```
> npm install gulp --save-dev
```

gulp-sass を追加（先ほどのページ）

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

Mac の方

gulp を追加

```
$ npm install gulp --save-dev
```

gulp-sass を追加（先ほどのページ）

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

追加された gulp プラグインは、node_module ディレクトリに格納され、package.json に追記されます

node_module ディレクトリが生成されていることを確認

```
> dir
<DIR> .
<DIR> ..
<DIR>    develop
<DIR>    html
<DIR>    node_modules
        256 package.json
```

node_module ディレクトリが生成されていることを確認

```
$ ls
develop      html      node_modules
package.json
```

3-4. 必要なプラグインを追加

package.json の中に gulp-sass が追記されています

```
{
  "name": "lesson",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" &&
exit 1"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "gulp-sass": "^2.0.4"
  }
}
```

この行が追加されました

※ gulp は `--save-dev` オプションを付けていないので、package.jsonには追記されません

3-5. gulpfile.js を作成

package.json と同じ階層に gulpfile.js を作成します

Windows の方

gulpfile.js を作成

```
> type nul > gulpfile.js
```

gulpfile.js が生成されていることを確認

```
> dir
<DIR>      .
<DIR>      ..
<DIR>      develop
            0  gulpfile.js
<DIR>      html
<DIR>      node_modules
            256 package.json
```

Mac の方

gulpfile.js を作成

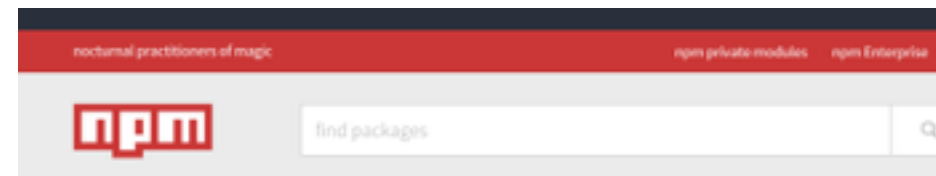
```
$ touch gulpfile.js
```

gulpfile.js が生成されていることを確認

```
$ ls
develop      gulpfile.js  html
node_modules package.json
```

3-5. gulpfile.js を作成

先ほどの gulp-sass のページを見て gulp-sass の動作に必要な記述を書いていきます



Install

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

Basic Usage

Something like this will compile your Sass files:

```
'use strict';

var gulp = require('gulp');
var sass = require('gulp-sass');

gulp.task('sass', function () {
  gulp.src('./sass/**/*.scss')
    .pipe(sass().on('error', sass.logError))
    .pipe(gulp.dest('./css'));
});

gulp.task('sass:watch', function () {
  gulp.watch('./sass/**/*.scss', ['sass']);
});
```

この部分を参考に
書いていきます

You can also compile synchronously, doing something like this:

```
'use strict';

var sass = require('sass');
```

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

3-5. gulpfile.js を作成

先ほどのページの内容を gulpfile.js にそのままコピペします

gulpfile.js

```
'use strict';

var gulp = require('gulp');
var sass = require('gulp-sass');

gulp.task('sass', function () {
  gulp.src('./sass/**/*.scss')
    .pipe(sass().on('error', sass.logError))
    .pipe(gulp.dest('./css'));
});

gulp.task('sass:watch', function () {
  gulp.watch('./sass/**/*.scss', ['sass']);
});
```

3-5. gulpfile.js を作成

今回のディレクトリ構成に合わせて編集します

gulpfile.js

```
'use strict';

var gulp = require('gulp');
var sass = require('gulp-sass');

gulp.task('sass', function () {
  gulp.src('./sass/**/*.scss')
    .pipe(sass().on('error', sass.logError))
    .pipe(gulp.dest('./css'));
});

gulp.task('sass:watch', function () {
  gulp.watch('./sass/**/*.scss', ['sass']);
});
```

- ・ Sass の拡張子は **.sass**
- ・ Sass のディレクトリは、gulpfile.js からの相対パス **./develop/sass**
- ・ CSS のディレクトリは、gulpfile.js からの相対パス **./html/css**

3-5. gulpfile.js を作成

編集後はこのようになります

gulpfile.js

```
'use strict';

var gulp = require('gulp');
var sass = require('gulp-sass');

gulp.task('sass', function () {
  gulp.src('./develop/sass/**/*.sass')
    .pipe(sass().on('error', sass.logError))
    .pipe(gulp.dest('./html/css'));
});

gulp.task('sass:watch', function () {
  gulp.watch('./develop/sass/**/*.sass',
    ['sass']);
});
```

現在のディレクトリ構成

```
lesson
| <開発用ディレクトリ>
├ develop
|   | <.sass ディレクトリ>
|   └ sass
|       └ style.sass
|
├ gulpfile.js
|
| <納品ファイル用ディレクトリ>
├ html
|   |
|   └ index.html
|
├ node_modules
└ package.json
```

gulpfile.jsの設定も完了しました

3-6. gulp sass:watch を実行してファイルを監視

Windows の方

gulp sass:watch タスクを実行してみます

```
> gulp sass:watch  
[15:38:32] Starting 'sass:watch'...  
[15:38:32] Finished 'sass:watch' after 10  
ms
```

Mac の方

gulp sass:watch タスクを実行してみます

```
$ gulp sass:watch  
[15:38:32] Starting 'sass:watch'...  
[15:38:32] Finished 'sass:watch' after  
10 ms
```

watch モードに入りました

3-6. gulp sass:watch を実行してファイルを監視

develop/sass/style.sass を編集・保存してみよう

style.sass

```
h1
  margin: 0 auto
a
  color: red
```

以下のように、sassが実行されていれば成功です

Windows の方

gulp sass:watch タスク

```
> gulp sass:watch
[15:38:32] Starting 'sass:watch'...
[15:38:32] Finished 'sass:watch' after 10 ms
[15:38:40] Starting 'sass'...
[15:38:40] Finished 'sass' after 9.06 ms
```

Mac の方

gulp sass:watch タスク

```
$ gulp sass:watch
[15:38:32] Starting 'sass:watch'...
[15:38:32] Finished 'sass:watch' after 10 ms
[15:38:40] Starting 'sass'...
[15:38:40] Finished 'sass' after 9.06 ms
```

3-6. gulp sass:watch を実行してファイルを監視

html/css/style.css を確認してみよう

style.css

```
h1 {  
  margin: 0 auto; }  
h1 a {  
  color: red; }
```

書式が気になります

現在のディレクトリ構成

```
lesson
| <開発用ディレクトリ>
├ develop
|   | <.sass ディレクトリ>
|   └ sass
|       └ style.sass
|
├ gulpfile.js
|
| <納品ファイル用ディレクトリ>
├ html
|   | <.css ディレクトリ(自動生成)>
|   └ css
|       └ style.css
|       |
|       └ index.html
|
├ node_modules
└ package.json
```

ひとまず gulp + Sass の環境は整いました

第3章 応用

1. プラグインを追加してみよう

1. gulp-csscomb で出力したCSSを整形
2. gulp-autoprefixer でベンダー・プレフィクスを追加
3. gulp-plumber でエラー時に処理が中断されないように

2. Sass コーディングに役立つプラグイン

1-1. gulp-csscomb で出力したCSSを整形

html/css/style.css の書式が気になります

style.css

```
h1 {  
  margin: 0 auto; }  
h1 a {  
  color: red; }
```

求める形はこっち



style.css

```
h1 {  
  margin: 0 auto;  
}  
  
h1 a {  
  color: red;  
}
```

1-1. gulp-csscomb で出力したCSSを整形

一旦、watch から抜けてプラグインと設定を追加をします

Windows の方

ctrl + c を押し、一旦タスクを終了します

```
[13:09:14] Starting 'sass'...  
[13:09:14] Finished 'sass' after 980 μs  
^C  
>
```

Mac の方

ctrl + c を押し、一旦タスクを終了します

```
[13:09:14] Starting 'sass'...  
[13:09:14] Finished 'sass' after 980 μs  
^C  
$
```

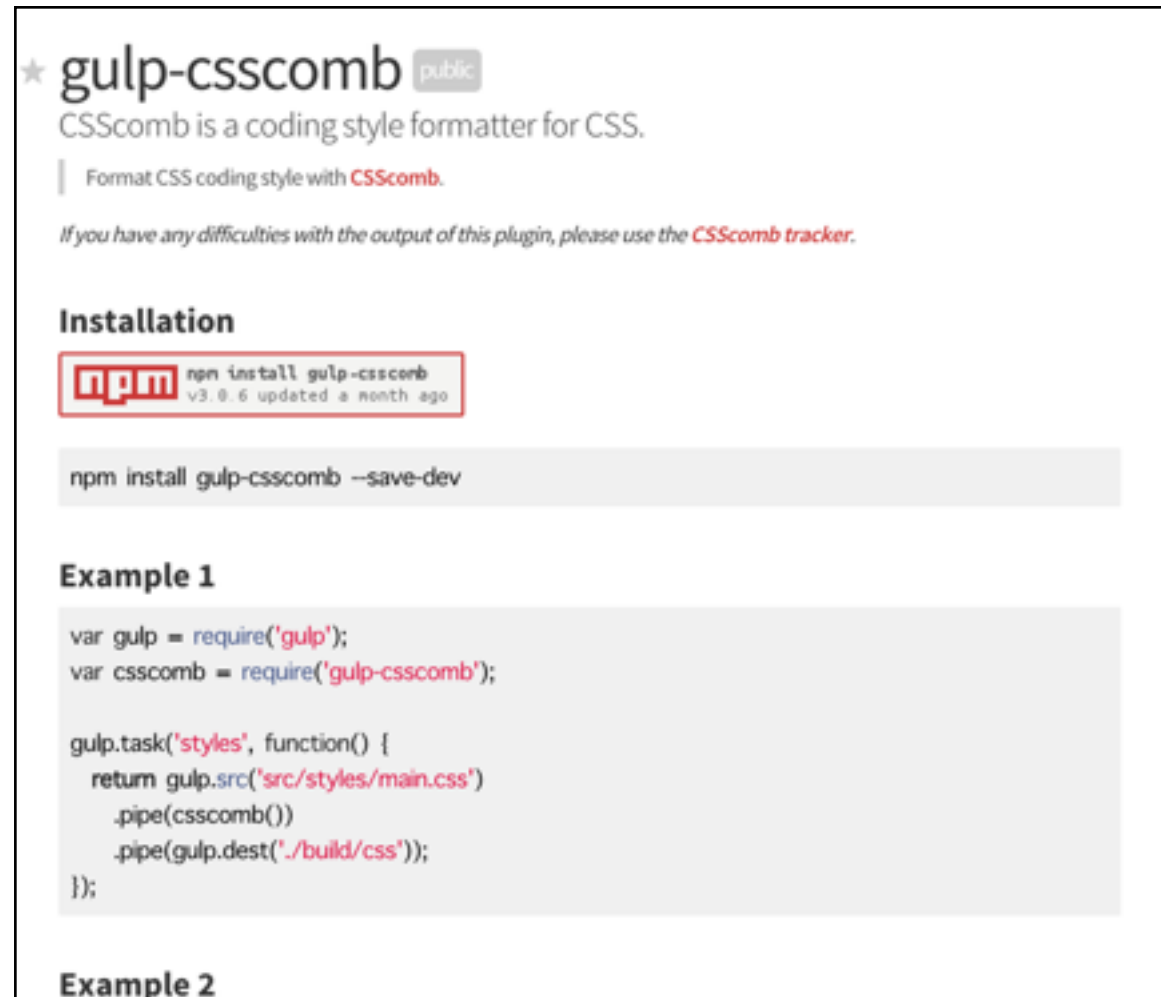

1-1. gulp-csscomb で出力したCSSを整形

gulp-csscomb

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

出力する CSS を好みの書式に整形します

1-1. gulp-csscomb で出力したCSSを整形



The screenshot shows the npm package page for **gulp-csscomb**. It includes the package name, a description "CSScomb is a coding style formatter for CSS.", and a note to use the CSScomb tracker for any difficulties. The **Installation** section shows the command `npm install gulp-csscomb --save-dev`. The **Example 1** section shows a Gulp task configuration that uses `csscomb` to format CSS files. The **Example 2** section is partially visible.

```
var gulp = require('gulp');
var csscomb = require('gulp-csscomb');

gulp.task('styles', function() {
  return gulp.src('src/styles/main.css')
    .pipe(csscomb())
    .pipe(gulp.dest('./build/css'));
});
```

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

プラグインページの内容を参考に追加しましょう。

1. インストール

```
npm install gulp-csscomb --save-dev
```

2. タスクを追記

```
var csscomb = require('gulp-csscomb');
```

3. タスクを `.pipe()` で繋げる

```
.pipe(csscomb())
```

1-1. gulp-csscomb で出力したCSSを整形

Windows の方

gulp-csscomb を追加

```
> npm install gulp-csscomb --save-dev
```

Mac の方

gulp-csscomb を追加

```
$ npm install gulp-csscomb --save-dev
```

1-1. gulp-csscomb で出力したCSSを整形

gulpfile.js

```
'use strict';

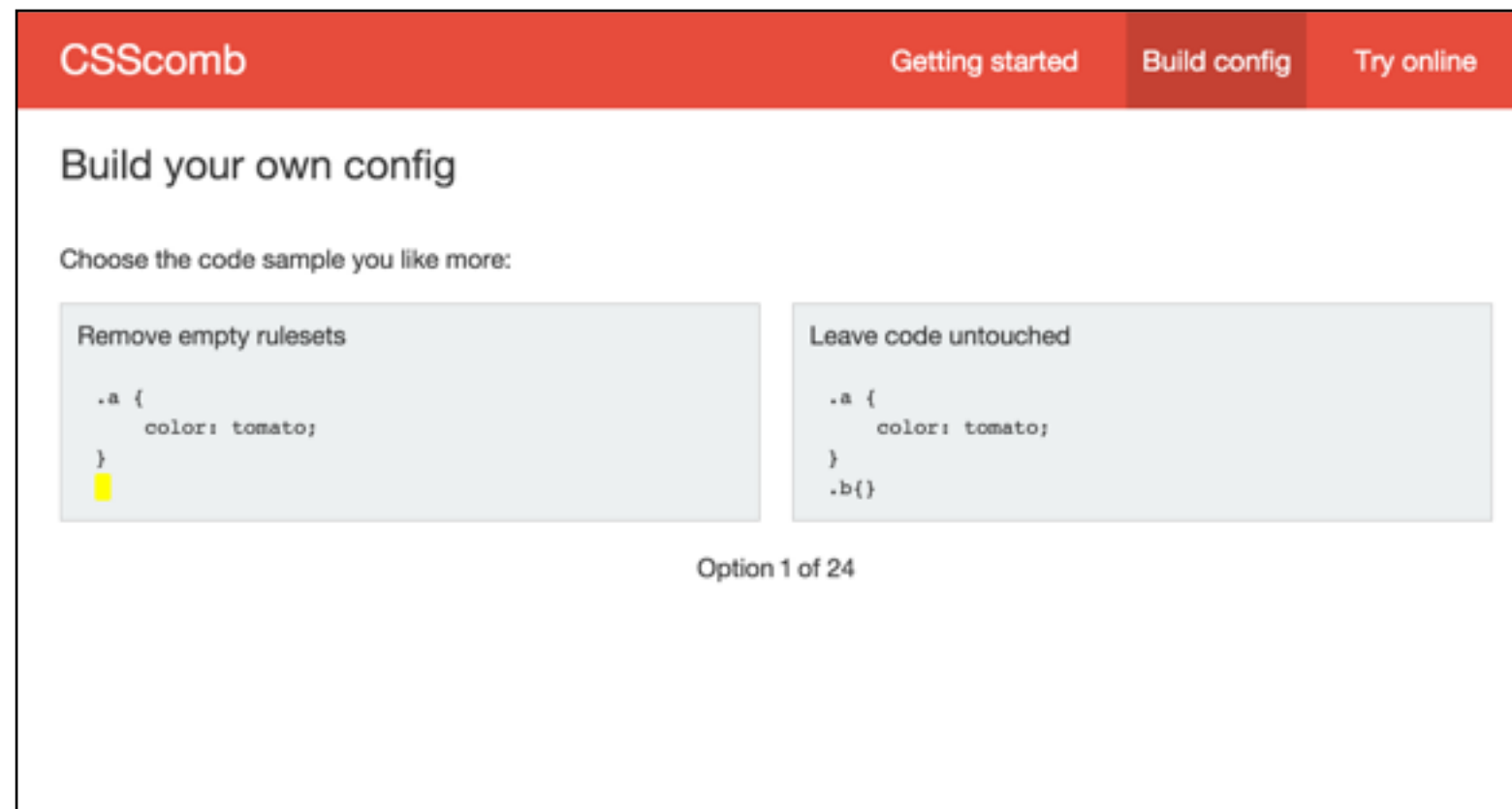
var gulp = require('gulp');
var sass = require('gulp-sass');
var csscomb = require('gulp-csscomb');

gulp.task('sass', function () {
  gulp.src('./develop/sass/**/*.sass')
    .pipe(sass().on('error', sass.logError))
    .pipe(csscomb())
    .pipe(gulp.dest('./html/css'));
});

gulp.task('sass:watch', function () {
  gulp.watch('./develop/sass/**/*.sass', ['sass']);
});
```

1-1. gulp-csscomb で出力したCSSを整形

gulp-csscomb ジェネレーターでお好みの設定を作成します



<http://csscomb.com/config>

1-1. gulp-csscomb で出力したCSSを整形

CSScomb[Getting started](#)[Build config](#)[Try online](#)

Build your own config

Copy the config from here ↓ and save it as `.csscomb.json` in your project's dir:

```
{
  "color-case": "upper",
  "block-indent": " ",
  "color-shorthand": true,
  "element-case": "upper",
  "eof-newline": false,
  "leading-zero": false,
  "quotes": "double",
  "space-before-colon": "",
  "space-after-colon": "",
  "space-before-combinator": "",
  "space-after-combinator": "",
  "space-between-declarations": " ",
  "space-before-opening-brace": " ",
  "space-after-opening-brace": " ",
  "space-after-selector-delimiter": " ",
  "space-before-selector-delimiter": "",
  "space-before-closing-brace": " ",
  "tab-size": true
}
```

出来上がった設定を `.csscomb.json` というファイル名で
`gulpfile.js` と同じ階層に保存します

1-1. gulp-csscomb で出力したCSSを整形

```
lesson
├── .csscomb.json ※MacのFinder上では不可視ファイルになります
│
│   <開発用ディレクトリ>
├── develop
│   │   <.sass ディレクトリ>
│   │   └── sass
│   │       └── style.sass
│   │
│   └── gulpfile.js
│   │
│   <納品ファイル用ディレクトリ>
├── html
│   │   <.css ディレクトリ(自動生成)>
│   │   └── css
│   │       └── style.css
│   │
│   └── index.html
│
├── node_modules
└── package.json
```

1-1. gulp-csscomb で出力したCSSを整形

gulp sass: watch でファイルを監視します

Windows の方

あらためて gulp sass:watch タスクを実行

```
> gulp sass:watch  
[15:38:32] Starting 'sass:watch'...  
[15:38:32] Finished 'sass:watch' after 10  
ms
```

Mac の方

あらためて gulp sass:watch タスクを実行

```
$ gulp sass:watch  
[15:38:32] Starting 'sass:watch'...  
[15:38:32] Finished 'sass:watch' after  
10 ms
```

watch モードに入りました

1-1. gulp-csscomb で出力したCSSを整形

先ほどと同じように develop/sass/style.sass を保存してみてください

style.sass

```
h1
  margin: 0 auto
a
  color: red
```

Windows の方

gulp sass:watch タスク

```
> gulp sass:watch
[15:38:32] Starting 'sass:watch'...
[15:38:32] Finished 'sass:watch' after 10 ms
[15:38:40] Starting 'sass'...
[15:38:40] Finished 'sass' after 9.06 ms
```

Mac の方

gulp sass:watch タスク

```
$ gulp sass:watch
[15:38:32] Starting 'sass:watch'...
[15:38:32] Finished 'sass:watch' after 10 ms
[15:38:40] Starting 'sass'...
[15:38:40] Finished 'sass' after 9.06 ms
```

1-1. gulp-csscomb で出力したCSSを整形

[html/css/style.css](#)を確認してみよう

style.css

```
h1 {  
  margin: 0 auto;  
}  
  
h1 a {  
  color: red;  
}
```

設定した通りの書式になっていれば完成です

美しい CSS は保証されました

1-2. gulp-autoprefixer でベンダー・プレフィクスを追加

style.sass

```
h1.span
  transform: rotate(-45deg)
  background-size: 20px 20px
```

求める形はこっち



style.css

```
h1.span {
  -webkit-transform: rotate(-45deg);
  -moz-transform: rotate(-45deg);
  -ms-transform: rotate(-45deg);
  transform: rotate(-45deg);
  -webkit-background-size: 20px 20px;
  background-size: 20px 20px;
}
```

1-2. gulp-autoprefixer でベンダー・プレフィックスを追加

一旦、watch から抜けてプラグインと設定を追加をします

Windows の方

ctrl + c を押し、一旦タスクを終了します

```
[13:09:14] Starting 'sass'...  
[13:09:14] Finished 'sass' after 980 μs  
^C  
>
```

Mac の方

ctrl + c を押し、一旦タスクを終了します

```
[13:09:14] Starting 'sass'...  
[13:09:14] Finished 'sass' after 980 μs  
^C  
$
```

1-2. gulp-autoprefixer でベンダー・プレフィクスを追加

gulp-autoprefixer

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

出力する CSS に要件に沿ったベンダー・プレフィクスを追加します

1-2. gulp-autoprefixer でベンダー・プレフィクスを追加



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

プラグインページの内容を参考に追加しましょう。

1. インストール

```
npm install gulp-autoprefixer --save-dev
```

2. タスクを追記

```
var autoprefixer = require('gulp-autoprefixer');
```

3. タスクを `.pipe()` で繋げる

```
.pipe(autoprefixer({
  browsers: ['last 2 versions'],
  cascade: false
}))
```

1-2. gulp-autoprefixer でベンダー・プレフィックスを追加

Windows の方

gulp-csscomb を追加

```
> npm install gulp-autoprefixer --save-dev
```

Mac の方

gulp-csscomb を追加

```
$ npm install gulp-autoprefixer --save-dev
```


1-2. gulp-autoprefixer でベンダー・プレフィクスを追加

gulpfile.js

```
'use strict';

var gulp = require('gulp');
var sass = require('gulp-sass');
var csscomb = require('gulp-csscomb');
var autoprefixer = require('gulp-autoprefixer');

gulp.task('sass', function () {
  gulp.src('./develop/sass/**/*.sass')
    .pipe(sass().on('error', sass.logError))
    .pipe(csscomb())
    .pipe(autoprefixer({
      browsers: ['last 2 versions'],
      cascade: false
    }))
    .pipe(gulp.dest('./html/css'));
});

gulp.task('sass:watch', function () {
  gulp.watch('./develop/sass/**/*.sass', ['sass']);
});
```

1-2. gulp-autoprefixer でベンダー・プレフィックスを追加

gulp sass: watch でファイルを監視します

Windows の方

あらためて gulp sass:watch タスクを実行

```
> gulp sass:watch  
[15:38:32] Starting 'sass:watch'...  
[15:38:32] Finished 'sass:watch' after 10 ms
```

Mac の方

あらためて gulp sass:watch タスクを実行

```
$ gulp sass:watch  
[15:38:32] Starting 'sass:watch'...  
[15:38:32] Finished 'sass:watch' after 10 ms
```

watch モードに入りました

1-2. gulp-autoprefixer でベンダー・プレフィクスを追加

先ほどと同じように develop/sass/style.sass を保存してみてください

style.sass

```
h1 span
  transform: rotate(-45deg)
  background-size: 20px 20px
```

Windows の方

gulp sass:watch タスク

```
> gulp sass:watch
[15:38:32] Starting 'sass:watch'...
[15:38:32] Finished 'sass:watch' after 10 ms
[15:38:40] Starting 'sass'...
[15:38:40] Finished 'sass' after 9.06 ms
```

Mac の方

gulp sass:watch タスク

```
$ gulp sass:watch
[15:38:32] Starting 'sass:watch'...
[15:38:32] Finished 'sass:watch' after 10 ms
[15:38:40] Starting 'sass'...
[15:38:40] Finished 'sass' after 9.06 ms
```

1-2. gulp-autoprefixer でベンダー・プレフィクスを追加

[html/css/style.css](#)を確認してみよう

style.css

```
h1 span {  
  -webkit-transform: rotate(-45deg);  
    transform: rotate(-45deg);  
  background-size: 20px 20px;  
}
```

1-2. gulp-autoprefixer でベンダー・プレフィクスを追加

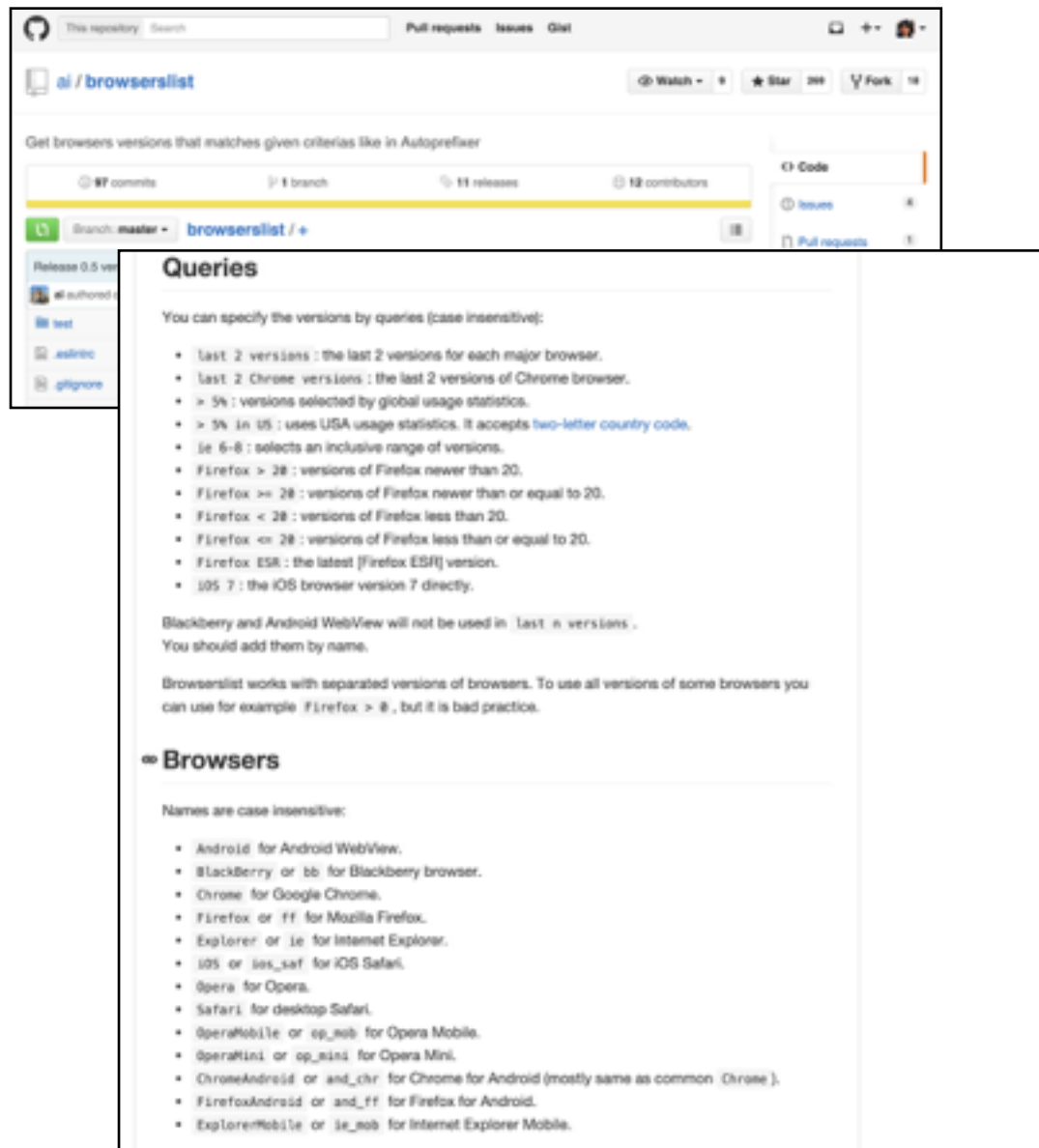
[html/css/style.css](#)を確認してみよう

style.css

```
h1 span {  
  -webkit-transform: rotate(-45deg);  
  -moz-transform: rotate(-45deg);  
  -ms-transform: rotate(-45deg);  
  transform: rotate(-45deg);  
  -webkit-background-size: 20px 20px;  
  background-size: 20px 20px;  
}
```

求めているものと少し違います！

1-2. gulp-autoprefixer でベンダー・プレフィックスを追加



GitHub ai/browserslist に設定方法が記載されています。

これを参考に、設定を変更します。
※プロジェクトに合わせ適宜調整してください

今回のプロジェクトは以下の設定を使用します。

```
browsers:['safari 5', 'ie 8', 'ie 9', 'ie 10', 'ie 11', 'opera 12.1', 'firefox 14', 'ios 6', 'android 2.1'],
```

<https://github.com/ai/browserslist>

1-2. gulp-autoprefixer でベンダー・プレフィクスを追加

gulpfile.js

```
'use strict';

var gulp = require('gulp');
var sass = require('gulp-sass');
var csscomb = require('gulp-csscomb');
var autoprefixer = require('gulp-autoprefixer');

gulp.task('sass', function () {
  gulp.src('./develop/sass/**/*.sass')
    .pipe(sass().on('error', sass.logError))
    .pipe(csscomb())
    .pipe(autoprefixer({
      browsers: ['safari 5', 'ie 8', 'ie 9', 'ie 10', 'ie 11', 'opera 12.1',
        'firefox 14', 'ios 6', 'android 2.1'],
      cascade: false
    })));
  .pipe(gulp.dest('./html/css'));
});

gulp.task('sass:watch', function () {
  gulp.watch('./develop/sass/**/*.sass', ['sass']);
});
```

1-2. gulp-autoprefixer でベンダー・プレフィクスを追加

設定を読み直すため一旦、watch から抜けて、

Windows の方

ctrl + c を押し、一旦タスクを終了します

```
[13:09:14] Starting 'sass'...  
[13:09:14] Finished 'sass' after 980 μs  
^C  
>
```

Mac の方

ctrl + c を押し、一旦タスクを終了します

```
[13:09:14] Starting 'sass'...  
[13:09:14] Finished 'sass' after 980 μs  
^C  
$
```

再度 gulp sass: watch でファイルを監視します

あらためて gulp sass:watch タスクを実行

```
> gulp sass:watch  
[15:38:32] Starting 'sass:watch'...  
[15:38:32] Finished 'sass:watch' after 10 ms
```

あらためて gulp sass:watch タスクを実行

```
$ gulp sass:watch  
[15:38:32] Starting 'sass:watch'...  
[15:38:32] Finished 'sass:watch' after 10 ms
```

watch モードに入りました

1-2. gulp-autoprefixer でベンダー・プレフィクスを追加

先ほどと同じように develop/sass/style.sass を保存してみてください

style.sass

```
h1 span
  transform: rotate(-45deg)
  background-size: 20px 20px
```

Windows の方

gulp sass:watch タスク

```
> gulp sass:watch
[15:38:32] Starting 'sass:watch'...
[15:38:32] Finished 'sass:watch' after 10 ms
[15:38:40] Starting 'sass'...
[15:38:40] Finished 'sass' after 9.06 ms
```

Mac の方

gulp sass:watch タスク

```
$ gulp sass:watch
[15:38:32] Starting 'sass:watch'...
[15:38:32] Finished 'sass:watch' after 10 ms
[15:38:40] Starting 'sass'...
[15:38:40] Finished 'sass' after 9.06 ms
```

1-2. gulp-autoprefixer でベンダー・プレフィックスを追加

[html/css/style.css](#)を確認してみよう

style.css

```
h1 span {  
  -webkit-transform: rotate(-45deg);  
  -moz-transform: rotate(-45deg);  
  -ms-transform: rotate(-45deg);  
  transform: rotate(-45deg);  
  -webkit-background-size: 20px 20px;  
  background-size: 20px 20px;  
}
```

求めている仕上がりになりました

自動でベンダー・プレフィクスが追加されます

1-3. gulp-plumber でエラー時に処理が中断されないように

文法違反などエラー発生時、gulp は処理を中断し停止してしまいます。

style.sass

```
h1
  &.span
    transform:
```

←値がない

この状態で保存すると処理が停止



```
[09:58:54] Starting 'sass'...
[09:58:54] Finished 'sass' after 12 ms

events.js:85
    throw er; // Unhandled 'error' event
          ^
Error: develop/sass/demo.sass
  2:21  error reading values after -
       at options.error (/Users/.../index.js:276:32)
$
```

←処理が停止

1-3. gulp-plumber でエラー時に処理が中断されないように

一旦、watch から抜けてプラグインと設定を追加をします

Windows の方

ctrl + c を押し、一旦タスクを終了します

```
[13:09:14] Starting 'sass'...  
[13:09:14] Finished 'sass' after 980 μs  
^C  
>
```

Mac の方

ctrl + c を押し、一旦タスクを終了します

```
[13:09:14] Starting 'sass'...  
[13:09:14] Finished 'sass' after 980 μs  
^C  
$
```

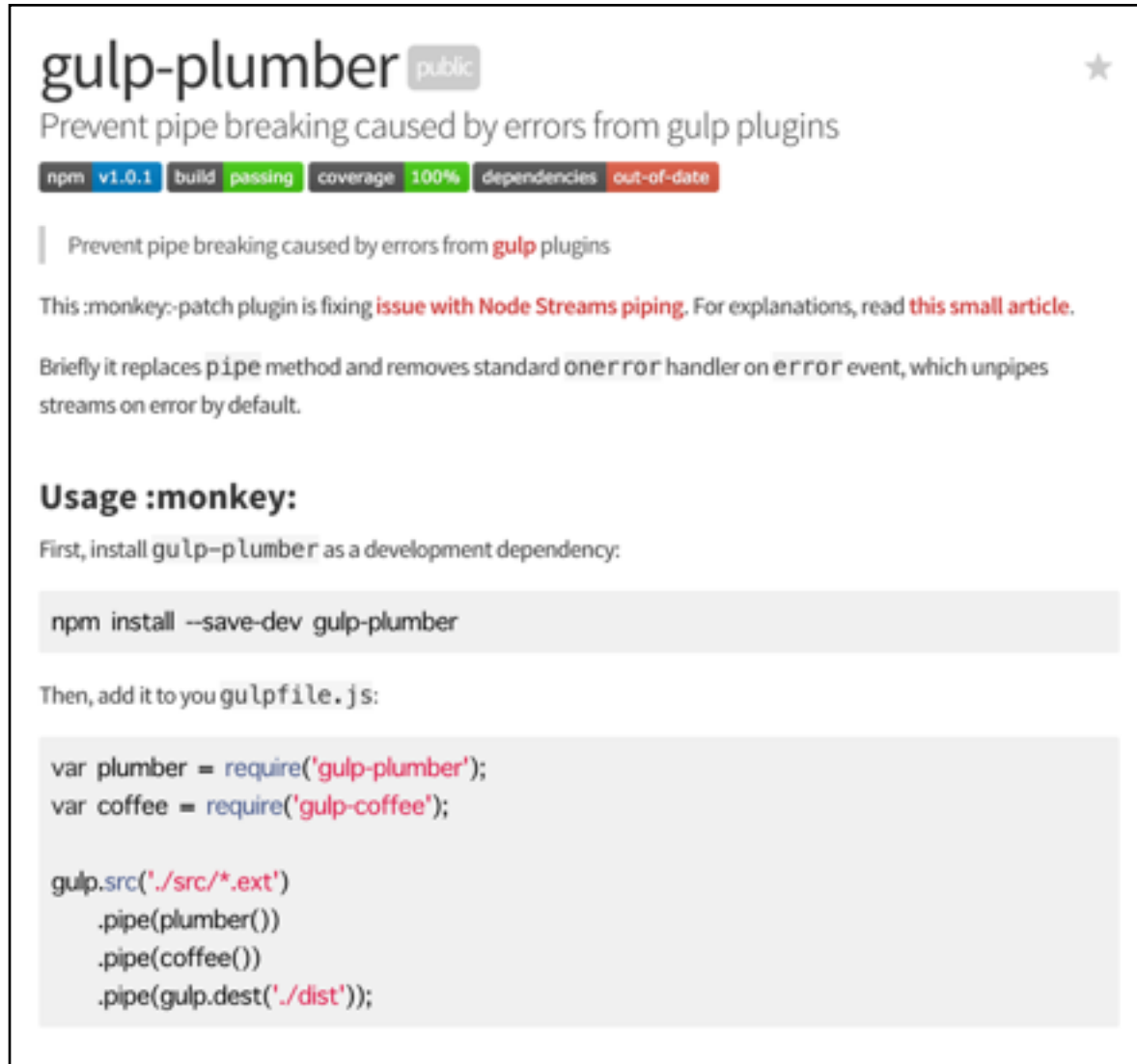
1-3. gulp-plumber でエラー時に処理が中断されないように

gulp-plumber

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

エラー発生時に処理を中断しないようにします

1-3. gulp-plumber でエラー時に処理が中断されないように



The screenshot shows the npm package page for `gulp-plumber`. It includes the package name, version `v1.0.1`, and status indicators for build, coverage, and dependencies. The description states it prevents pipe breaking caused by errors from gulp plugins. It mentions a patch for Node Streams piping and provides a link to a small article. The usage section instructs to install it as a development dependency and shows a code snippet for integrating it into a gulpfile.js.

gulp-plumber public

Prevent pipe breaking caused by errors from gulp plugins

npm v1.0.1 build passing coverage 100% dependencies out-of-date

Prevent pipe breaking caused by errors from **gulp** plugins

This `:monkey:` patch plugin is fixing **issue with Node Streams piping**. For explanations, read [this small article](#).

Briefly it replaces `pipe` method and removes standard `onError` handler on `error` event, which unpipes streams on error by default.

Usage :monkey:

First, install `gulp-plumber` as a development dependency:

```
npm install --save-dev gulp-plumber
```

Then, add it to you `gulpfile.js`:

```
var plumber = require('gulp-plumber');
var coffee = require('gulp-coffee');

gulp.src('./src/*.ext')
  .pipe(plumber())
  .pipe(coffee())
  .pipe(gulp.dest('./dist'));
```

プラグインページの内容を参考に追加しましょう。

1. インストール

```
npm install gulp-plumber --save-dev
```

2. タスクを追記

```
var plumber = require('gulp-plumber');
```

3. タスクを `.pipe()` で繋げる

```
.pipe(plumber())
```

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

1-3. gulp-plumber でエラー時に処理が中断されないように

Windows の方

gulp-csscomb を追加

```
> npm install gulp-plumber --save-dev
```

Mac の方

gulp-csscomb を追加

```
$ npm install gulp-plumber --save-dev
```


1-3. gulp-plumber でエラー時に処理が中断されないように

gulpfile.js

```
'use strict';

var gulp = require('gulp');
var sass = require('gulp-sass');
var csscomb = require('gulp-csscomb');
var autoprefixer = require('gulp-autoprefixer');
var plumber = require('gulp-plumber');

gulp.task('sass', function () {
  gulp.src('./develop/sass/**/*.sass')
    .pipe(plumber())
    .pipe(sass().on('error', sass.logError))
    .pipe(csscomb())
    .pipe(autoprefixer({
      browsers: ['last 2 versions'],
      cascade: false
    }))
    .pipe(gulp.dest('./html/css'));
});

gulp.task('sass:watch', function () {
  gulp.watch('./develop/sass/**/*.sass', ['sass']);
});
```

1-3. gulp-plumber でエラー時に処理が中断されないように

gulp sass: watch でファイルを監視します

Windows の方

あたためて gulp sass:watch タスクを実行

```
> gulp sass:watch  
[15:38:32] Starting 'sass:watch'...  
[15:38:32] Finished 'sass:watch' after 10  
ms
```

Mac の方

あたためて gulp sass:watch タスクを実行

```
$ gulp sass:watch  
[15:38:32] Starting 'sass:watch'...  
[15:38:32] Finished 'sass:watch' after  
10 ms
```

watch モードに入りました

1-3. gulp-plumber でエラー時に処理が中断されないように

style.sass

```
h1
  &.span
    transform:
```

←値がない

先ほどと同じように保存



```
[10:31:18] Starting 'sass'...
[10:31:18] Finished 'sass' after 16 ms
[10:31:18] Plumber found unhandled error:
Error in plugin 'gulp-sass'
Message:
  develop/sass/demo.sass
  2:21  error reading values after -
Details:
  column: 21
  line: 2
  file: stdin
  status: 1
  messageFormatted: develop/sass/demo.sass
  2:21  error reading values after -
```

←処理が継続されています

処理が継続されるので作業に集中できます

2. Sass コーディングに役立つプラグイン

del

<https://www.npmjs.com/package/del>

ファイルやディレクトリを削除

2. Sass コーディングに役立つプラグイン

gulp-combine-media-queries

<https://www.npmjs.com/package/gulp-combine-media-queries>

メディアクエリ @media をまとめてくれます

2. Sass コーディングに役立つプラグイン

gulp-replace

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

正規表現で文字列を置き換えできます

2. Sass コーディングに役立つプラグイン

gulp-concat

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

複数のファイルを結合

2. Sass コーディングに役立つプラグイン

gulp-rename

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

ファイル名を変更

2. Sass コーディングに役立つプラグイン

gulp-minify-css

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

CSS をミニファイ化します

2. Sass コーディングに役立つプラグイン

gulp-gzip

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

CSS を gzip 圧縮します

2. Sass コーディングに役立つプラグイン

gulp-webserver

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

ファイルの更新を検知して自動的にブラウザを更新します

2. Sass コーディングに役立つプラグイン

gulp-sftp

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

ファイルを sftp で転送 (gulp-ftp もあります)

などなど、いろいろありますので試してみましょう