
Table of Contents

Introduction	1.1
SASS 環境安裝	1.2
從 CSS 到 SASS	1.3
SASS 秘辛	1.4
用 Gulp 編譯 SASS	1.5
實作：用 Sass 來抓寶吧！	1.6
學習資源	1.7

SASS Dojo

CSS 是用於網頁的一種靜態裝飾表達式，至今 CSS3 擁有許多表達式，讓網頁更加豐富、精彩。但 CSS 無法程式化，也缺乏再用性與易讀性，Sass 因而誕生。

Sass 基於 CSS 並補足 CSS 原生的缺點。針對未曾嘗試 Sass 開發的網頁開發者，本實作將帶來全新的體驗。藉由 Sass 與 CSS 的比較，並透過範例讓開發者瞭解 Sass 的好處，進而優化開發流程、加速 CSS 開發成果。

Let's write CSS with Superpower!

Sass 環境安裝

Sass 由 Ruby 開發而成，針對不同作業系統有不同的安裝方式：

MAC

OS X 內建 Ruby，你可以直接在終端機輸入指令來安裝：

```
gem install sass
```

如果你收到錯誤訊息，嘗試透過 `sudo` 來安裝：

```
sudo gem install sass
```

安裝完成後，可以透過 `sass -v` 來檢查目前 Sass 的安裝版本

Linux

你需要先安裝 Ruby，再透過以下指令安裝：

```
sudo su -c "gem install sass"
```

Windows

你可以透過 [Ruby Installer](#) 快速安裝 Ruby 與 Sass。

LibSass

LibSass 讓你可以用不同程式語言來編寫 Sass，你可以使用自己熟悉的語言與環境來增加開發效率。

更多資訊參考：<http://sass-lang.com/libsass>

從 CSS 到 SASS

Sass 並不會取代 CSS，而是透過編譯轉換成 CSS 檔案，再引入到 HTML 裡面。CSS 不像一般程式語言擁有定義變數或模組的方法，只能一行一行編寫。當有很多元素需要修改時，就會耗費很多時間逐行修正，不利維護與重置。

這是一般常見的 CSS：

```
.menu{  
    position: relative;  
}  
  
.menu .submenu{  
    float: left;  
}  
  
.menu a{  
    display: block;  
}  
  
.menu a:hover{  
    color: red;  
}  
  
.menu span{  
    color: #fff;  
}
```

以上範例可以發現 CSS 不管在新增、調整或除錯上都無法有效率的進行。

當我們採用 **Sass** 來編寫：

```
.menu
  position: relative

  .submenu
    float: left

    a
      display: block

      &:hover
        color: red

    span
      color: #fff
```

Sass 省略大括號與分號，並透過縮排來定義子元素，語法十分精簡。

如果你不適應 Sass 過於精簡的寫法，也可以試試 **Scss**：

```
.menu {  
  position: relative;  
  
  .submenu {  
    float: left;  
  }  
  
  a {  
    display: block;  
  
    &:hover{  
      color: red;  
    }  
  }  
  
  span {  
    color: #fff;  
  }  
}
```

Scss 寫法類似 CSS，保留大括號與分號，但多了巢狀結構。

Sass 秘辛

Sass 還有許多厲害的功能，例如變數、算數、函示……等，以下將依依介紹：

變數

將 CSS 屬性用變數儲存起來，便能有效地重複使用，也能快速地進行修改。

Sass 使用 `$` 來命名變數：

```
$font-stack: Helvetica, sans-serif;  
$primary-color: #333;  
  
h1 {  
    font-family: $font-stack;  
    color: $primary-color;  
}  
  
p {  
    color: $primary-color;  
}
```

當需要同時修改 `h1` 和 `p` 的顏色時，只要修改 `$primary-color` 就能一次套用。

巢狀結構

善用巢狀結構來避免重複編寫母元素的麻煩，同時提高易讀性和修改的效率。

```
nav {  
  
  ul {  
    margin: 0;  
    padding: 0;  
    list-style: none;  
  }  
  
  li { display: inline-block; }  
  
  a {  
    display: block;  
    padding: 6px 12px;  
    text-decoration: none;  
  }  
}
```

算數

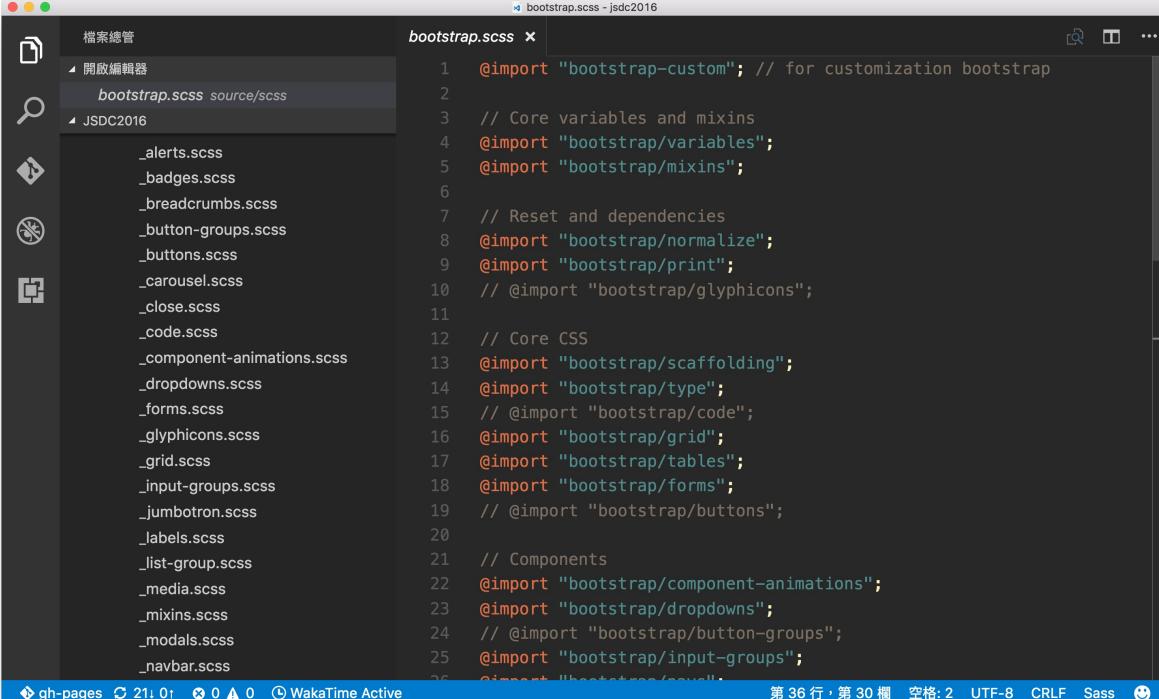
Sass 支援 `+` , `-` , `*` , `/` , `%` , 能夠快速地改變數值。

```
$font-size: 16px;  
$color: #333;  
  
.font-big {  
  font-size: $font-size * 1.2;  
  color: $color - #111;  
}  
  
.font-small {  
  font-size: $font-size / 2;  
  color: $color + #111;  
}
```

Import

Sass 能夠實現 CSS 的模組化，將不同功能的 CSS 檔案拆開，再引入到需要的檔案裡。除了方便管理，也能讓程式碼更精簡、易讀。

以底線 `_` 開頭的檔案名稱不會被編譯，這些檔案可以透過 `@import '檔案名稱 (可省略副檔名)' ;` 引入到其他檔案裡使用。



```

bootstrap.scss x
1  @import "bootstrap-custom"; // for customization bootstrap
2
3  // Core variables and mixins
4  @import "bootstrap/variables";
5  @import "bootstrap/mixins";
6
7  // Reset and dependencies
8  @import "bootstrap/normalize";
9  @import "bootstrap/print";
10 // @import "bootstrap/glyphicons";
11
12 // Core CSS
13 @import "bootstrap/scaffolding";
14 @import "bootstrap/type";
15 // @import "bootstrap/code";
16 @import "bootstrap/grid";
17 @import "bootstrap/tables";
18 @import "bootstrap/forms";
19 // @import "bootstrap/buttons";
20
21 // Components
22 @import "bootstrap/component-animations";
23 @import "bootstrap/dropdowns";
24 // @import "bootstrap/button-groups";
25 @import "bootstrap/input-groups";
26 @import "bootstrap/navs";

```

第 36 行, 第 30 欄 空格: 2 UTF-8 CRLF Sass ☺

如圖，Bootstrap 將每個元件拆開，再用 `@import` 的方式集合起來，方便使用者有效地管理樣式。

Mixin

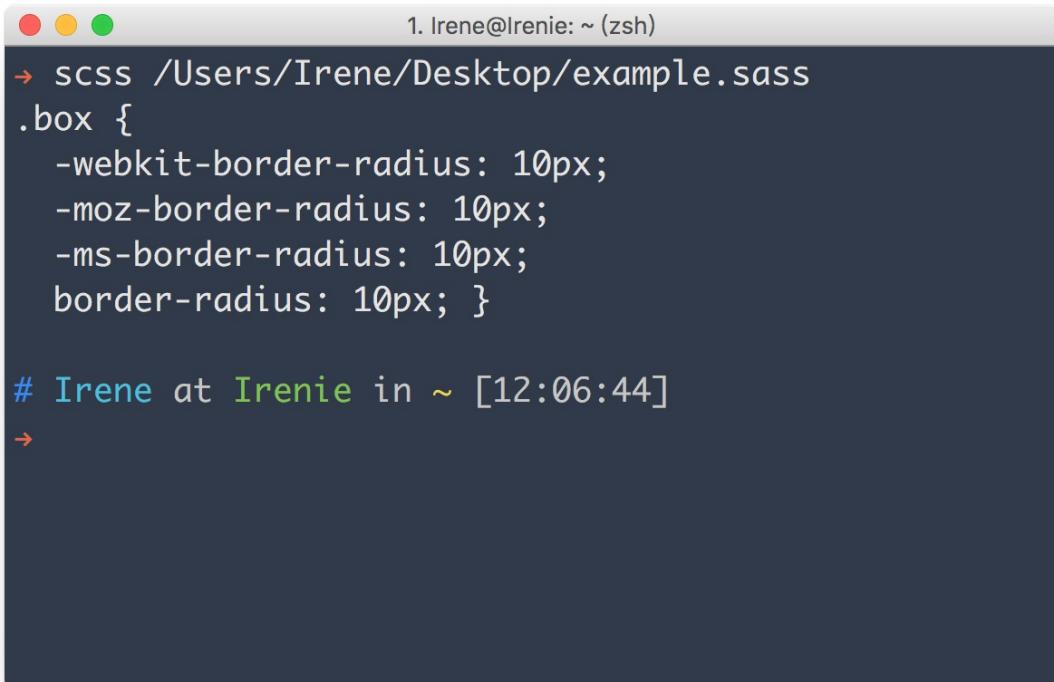
Mixin 能夠儲存數種樣式，也可以傳入數值，如同程式語言中常見的函式。

```
@mixin border-radius($radius) {  
  -webkit-border-radius: $radius;  
  -moz-border-radius: $radius;  
  -ms-border-radius: $radius;  
  border-radius: $radius;  
}  
  
.box { @include border-radius(10px); }
```

上述範例使用 `@mixin` 定義一個名叫 `border-radius` 的函示，且需要傳入一個參數。定義完成後，就能使用 `@include` 來取用。

使用 Gulp 編譯 Sass

Sass 需要編譯成 CSS 才能在瀏覽器上看到結果。在終端機輸入 `sass` 或 `scss` 指令，就能在終端機上看到編譯後的結果：



1. Irene@Irenie: ~ (zsh)
→ scss /Users/Irene/Desktop/example.sass
.box {
-webkit-border-radius: 10px;
-moz-border-radius: 10px;
-ms-border-radius: 10px;
border-radius: 10px; }

Irene at Irenie in ~ [12:06:44]
→

Gulp

Gulp 是一套基於 Node.js 的任務自動化管理工具，能夠將各種任務（編譯、壓縮、重新命名……等）串接起來並一次執行。

使用 Gulp 前需要先安裝 Node.js (<https://nodejs.org/>)，再透過以下指令安裝 Gulp：

```
npm install -g gulp
```

移動到專案目錄底下，輸入 `npm init` 來初始化，並為這個專案安裝 Gulp 套件：

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

接著安裝 `gulp-sass` 這個套件來編譯 SASS：

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

gulpfile.js

在專案根目錄之下，新增 `gulpfile.js`：

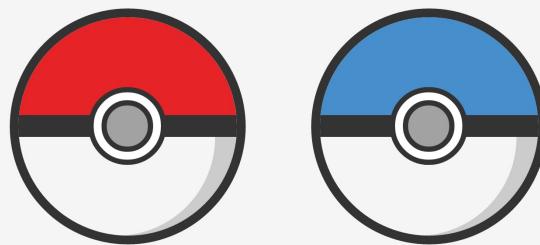
```
var gulp = require('gulp'), // 載入 gulp
    gulpSass = require('gulp-sass'); // 載入 gulp-sass，並指定給 gulpSass 這個變數

gulp.task('styles', function () { // 'styles' 是這個任務的名稱，可以任意命名
    gulp.src('./src/style/*.scss') // 指定要處理的 Sass 檔案目錄
        .pipe(gulpSass()) // 編譯 Sass
        .pipe(gulp.dest('./dist/style')); // 指定編譯後的 CSS 檔案目錄
});

gulp.task('default', ['styles']);
```

在終端機輸入 `gulp` 指令來執行上述程式碼，結束後就能在指定的輸出目錄之下找到編譯完成的 CSS 檔案。

用 Sass 來抓寶吧！



上圖即是用 Scss 繪製而成的神奇寶貝球，動手試試看吧：

Step 1: 建立專案目錄

為專案建立一個新資料夾，並開啟終端機移動到該目錄之下，輸入 `npm init` 來初始化專案。

接著輸入以下指令來安裝 Gulp：

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

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

Gulp 安裝完成後，在根目錄新增 `gulpfile.js` 來編譯 Scss：

```
var gulp = require('gulp'),  
  gulpSass = require('gulp-sass');  
  
gulp.task('styles', function () {  
  gulp.src('./src/*.scss')  
    .pipe(gulpSass())  
    .pipe(gulp.dest('./css/'));  
});  
  
gulp.task('default', ['styles']);
```

接著繼續在根目錄新增 `index.html` 與 `src/style.scss`。

完成後的目錄結構：

```
|- gulpfile.js  
|- index.html  
|- node_modules  
|- package.json  
|- src  
  |- style.scss
```

Step 2: 在頁面上展示你的神奇寶貝球

編輯 `index.html` 來調整神奇寶貝球在頁面上的位置：

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title></title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="css/style.css" rel="stylesheet">
  </head>
  <body>
    <div class="container">
      <div class="pokeball red"><div class="pokeball-btn"></div></div>
      <div class="pokeball blue"><div class="pokeball-btn"></div>
    </div>
  </body>
</html>
```

記得根據 `gulpfile.js` 的設定來引入編譯後的 CSS 檔案：

```
<link href="css/style.css" rel="stylesheet">
```

Step 3: 定義變數與函示

專案通常會將定義變數與函式的檔案獨立出來，再用 `@import` 的方式進行管理。

分別新增 `_variables.scss` 與 `_mixin.scss` 至 `src` 目錄底下：

```
// Variables
$color-red: #e62327;
$color-blue: #478ecc;
$color-gray: #333;
```

- 顏色或間距常會被定義成變數，能夠確保專案的統一性，以及避免繁複的撰寫。

```
// Mixins
@mixin border-radius($radius){
  -webkit-border-radius: $radius;
  -moz-border-radius: $radius;
  border-radius: $radius;
}

@mixin size($width, $height) {
  width: $width;
  height: $height;
}

@mixin ball-wrapper($color) {
  @include border-radius(100px 100px 0 0);
  @include size(200px, 90px);
  position: absolute;
  background-color: $color;
  border-bottom: 20px solid $color-gray;
}
```

- 函示常用來解決 prefix 的問題如 `@mixin border-radius($radius)` 。
- `@mixin size($width, $height)` 用來避免反覆撰寫元素的大小。
- `@mixin ball-wrapper($color)` 用來繪製不同寶貝球的上蓋顏色。

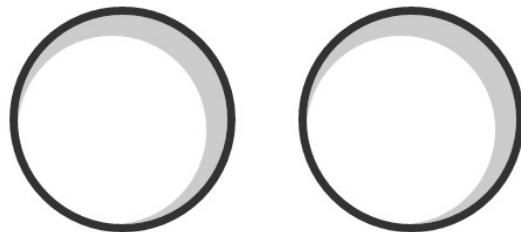
最後記得在 `style.scss` 開頭引入這兩支檔案：

```
@import 'variables';
@import 'mixin';
```

Step 4: 製作你專屬的神奇寶貝球

繼續編輯你的 `style.scss` 來完成你的神奇寶貝球吧：

A. 繪出寶貝球的輪廓

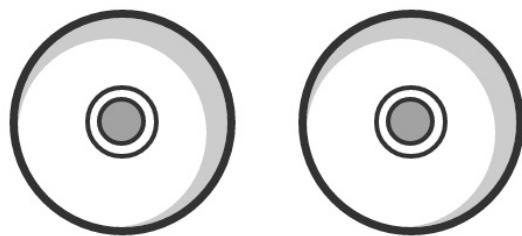


```
.pokeball {  
  @include border-radius(50%);  
  @include size(200px, 200px);  
  
  float: left;  
  overflow: hidden;  
  position: relative;  
  margin: 30px;  
  border: 8px solid $color-gray;  
  box-shadow: inset -10px 10px 0 10px #ccc;  
}
```

使用 `@include` 來使用我們已經定義好的函式，函式能根據傳入的參數不同而有不同結果。接著加上邊框與陰影。

B. 加上中間的按鈕

實作：用 Sass 來抓寶吧！



```
.pokeball {  
  @include border-radius(50%);  
  @include size(200px, 200px);  
  
  float: left;  
  overflow: hidden;  
  position: relative;  
  margin: 30px;  
  border: 8px solid $color-gray;  
  box-shadow: inset -10px 10px 0 10px #ccc;  
  
  .pokeball-btn{  
    @include border-radius(50%);  
    @include size(62px, 62px);  
    position: absolute;  
    border: 5px solid $color-gray;  
    background-color: white;  
    margin: 64px;  
  
    &::after {  
      @include border-radius(50%);  
      @include size(38px, 38px);  
      content:"";  
      position: absolute;  
      border: 5px solid $color-gray;  
      background-color: $color-gray + 111;  
      margin: 7px;  
    }  
  }  
}
```

這邊使用 `::after` 擬元素來畫出中間的灰色圓形，擬元素（pseudo-elements）是一種特別的 CSS Selector，用來指向指定元素的最前或最後，這樣就能避免在 HTML 裡寫入太多元素。

實作：用 Sass 來抓寶吧！

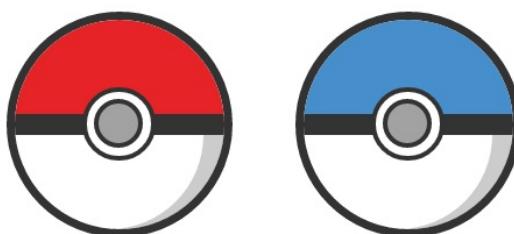
注意到 `::after` 前方的 `&` 符號了嗎？`&` 指向母元素，會將接在後方的 Selector 拉到與母元素同一個層級。例如：

```
.pokeball-btn {  
  &::after {  
    ...  
  }  
}
```

在 CSS 中等同於：

```
.pokeball-btn::after {  
  ...  
}
```

C. 你掉的是紅球還是藍球呢？



```
.pokeball {  
  @include border-radius(50%);  
  @include size(200px, 200px);  
  
  float: left;
```

實作：用 Sass 來抓寶吧！

```
overflow: hidden;
position: relative;
margin: 30px;
border: 8px solid $color-gray;
box-shadow: inset -10px 10px 0 10px #ccc;

.pokeball-btn{
  @include border-radius(50%);
  @include size(62px, 62px);
  position: absolute;
  border: 5px solid $color-gray;
  background-color: white;
  margin: 64px;

  &::after {
    @include border-radius(50%);
    @include size(38px, 38px);
    border: 5px solid $color-gray;
    background-color: $color-gray + 111;
    margin: 7px;
  }
}

&.red {
  &::before{
    @include ball-wrapper($color-red)
    content:"";
    position: absolute;
  }
}

&.blue {
  &::before{
    @include ball-wrapper($color-blue)
    content:"";
    position: absolute;
  }
}
```

```
}
```

還記得在 HTML 中，我們如何定義紅色與藍色的寶貝球嗎？我們將顏色與 `.pokeball` 放在一起。

```
<div class="pokeball red"></div>
<div class="pokeball blue"></div>
```

因此再次使用 `&` 符號，便能將 `.red` 與 `.blue` 拉到與 `.pokeball` 同一層級。這樣不僅能夠分別定義不同的顏色，也能保有Sass 特有的巢狀結構。

Step 5: 我要成為神奇寶貝大師！

只差最後一步，就能帶著你的寶貝球踏上神奇寶貝大師之旅了！

打開終端機，在專案根目錄下輸入 `gulp`，Gulp 就會將 `./src/style.scss` 編譯成 `./css/style.css`，編譯完成後，就能在瀏覽器看到成果囉！

```
1. Irene@Irene: ~/Github/sass-in-real/exercise (zsh)
→ gulp
(node:10260) fs: re-evaluating native module sources is not supported. If you are
e using the graceful-fs module, please update it to a more recent version.
[17:48:13] Using gulpfile ~/Github/sass-in-real/exercise/gulpfile.js
[17:48:13] Starting 'styles'...
[17:48:13] Finished 'styles' after 9.29 ms
[17:48:13] Starting 'default'...
[17:48:13] Finished 'default' after 14 µs

events.js:160
    throw er; // Unhandled 'error' event
    ^
Error: src/style.scss
Error: Invalid CSS after "i": expected 1 selector or at-rule, was "import 'varia
bles';"
    on line 1 of src/style.scss
>> import 'variables';
    ^
    at options.error (/Users/Irene/Github/sass-in-real/exercise/node_modules/nod
e-sass/lib/index.js:272:32)

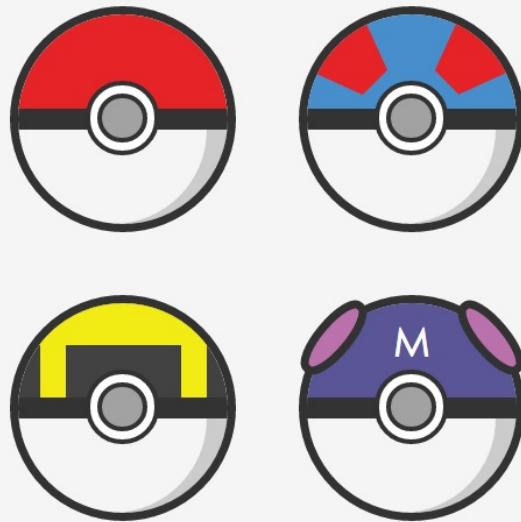
# Irene at Irene.local in ~/Github/sass-in-real/exercise on git:irene_demo_code
✖ [17:48:13]
→ |
```

當你的程式碼有誤導至編譯不成功時，Gulp 會在終端機指出問題出處來幫助你除錯。

精益求精

試著改變範例的程式碼，看看能不能作出更多自己想要的效果。你也可以挑戰更高的目標，試試做出各種不同的神奇寶貝球：

實作：用 Sass 來抓寶吧！



唯有不斷練習才能更加熟練，很快你就會說：「我再也回不去了！」

本範例完整程式碼：(待捕)

學習資源

Sass

- [Sass 官方網站](#)
- [Sass & Susy教學手冊](#)

Gulp

- [gulp 學習筆記](#)
- [gulp 入門指南](#)