Препроцессоры

Что такое препроцессор и css-препроцессор

Препроцессор (от англ. Preprocessor) — это инструмент, преобразующий код из одного синтаксиса в другой.

Обычно, на вход препроцессора поступает код, написанный с использованием синтаксических конструкций, понятных этому препроцессору.

Стоит сказать, что препроцессор может полностью замещать синтаксические конструкции языка или частично расширять их, то есть добавлять новые конструкции.

Как правило, на выходе ожидается код с более низким уровнем. То есть код, лишённый синтаксических конструкций, вносимых препроцессором.

CSS препроцессор (от англ. CSS preprocessor) — это надстройка над CSS, которая добавляет ранее недоступные возможности для CSS, с помощью новых синтаксических конструкций.

Основная задача препроцессора — это предоставление удобных синтаксических конструкций для разработчика, чтобы упростить, и тем самым, ускорить разработку и поддержку стилей в проектах.

+

“Синтаксический сахар”

CSS препроцессоры преобразуют код, написанный с использованием препроцессорного языка, в чистый и валидный CSS-код.

Синтаксический сахар (от англ. syntactic sugar) — это дополнения синтаксиса языка программирования, которые не вносят каких-то существенных изменений или новых возможностей, но делают этот язык более читабельным для человека.

Синтаксический сахар вводит в язык альтернативные варианты записи заложенных в этот язык конструкций.

Под альтернативными вариантами записи стоит понимать более короткие или удобные конструкции для человека, которые в конечном итоге будут преобразовываться препроцессором в исходный язык, без синтаксического сахара.

Если попытаться применить это понятие к CSS-препроцессорам, то оно, в общем случае, полностью описывает их суть. Ещё раз напомню, что основной задачей препроцессоров является упрощение и ускорение разработки, а как это ещё можно сделать, если не ввести альтернативные варианты записи?

Обзор препроцессоров

можно выделить три популярных препроцессора:

- Less
- Sass (SCSS)
- Stylus

Какой препроцессор выбрать? На самом деле это… не имеет значения.

Все варианты предоставляют примерно одинаковые возможности, просто синтаксис у каждого разный.

Рекомендуем действовать следующим образом:

- если вы – программист и хотите работать со стилями как с кодом, используйте Sass;
- если вы – верстальщик и хотите работать со стилями как с обычной верстkой, обратите внимание на Less;
- если вы любите минимализм, воспользуйтесь Stylus.

Сначала мы рассмотрим препроцессор Less, а потом посмотрим чес он отличается от Sass и Slylus.

Препроцессор Less. Краткая история.

В 2009 году Алексис Селье представляет миру новый инструмент для более гибкой работы со стилями под названием Less. Первая реализация была разработана Алексисом на Ruby.

В это время Node.js начал набирать обороты. Алексис, чувствуя тенденцию, решает переписать Less на JavaScript.

Less довольно быстро начал обретать популярность, Twitter начал использовать его в своем фреймворке Bootstrap, звездочки и форки в GitHub росли.

В мае 2012 года Алексис передает контроль над разработкой и развитием проекта команде, что дало очередной рывок в развитии Less. Также появился новый сайт с хорошей документацией.

Сегодня Less может работать как на сервере под Node.js или Rhino, так и на клиенте.

Процесс разработки с помощью Less

1. Пишем код CSS с использованием синтаксиса и возможностей препроцессора
2. Компилируем получившийся CSS, т.к. браузер не понимает синтаксис препроцессора и мы не можем просто взять и подключить файл .less как .css. Варианты компиляции:
    a. с помощью приложения-компилятора или сборщика пакетов
    b. преобразовать файл вручную (например, использовать он-лайн компилятор)
    c. компиляция из командной строки (lessc)
    d. компилировать на стороне клиента “на ходу” во время обновления страницы (в браузере). Для этого нужно подключить на страницу less.js

Компиляция в браузере (less.js)

Наиболее простой способ использования CSS-препроцессора, но в тоже время малопопулярный. Альтернативные решения удобнее и предоставляют наиболее интересный функционал. Применяется на этапе разработки или отладки проекта, когда важен результат компиляции, а не её скорость.

```
<link rel="stylesheet/less" href="...">
<script src="less.min.js"></script>
```

Способ не желателен к применению на так называемом «продакшене» в виду того, что имеет серьезные проблемы со скоростью и сильно зависит от производительности устройства, а также скорости интернет-соединения.

Помимо этого увеличивается объем загружаемых данных, так как браузеру пользователя приходится загружать less-файлы и файл библиотеки.

Только после полной загрузки необходимых ресурсов начинается процесс компиляции less-кода в CSS.

Компиляция из командной строки (lessc)

Работа из командной строки предполагает наличие установленного Node.js. Помимо этого, необходимо глобально установить пакет `less` — это можно сделать командой:

```
$ npm install -g less
```

Рассмотрим синтаксис команд npm:

- `npm` - пакетный менеджер;
- `i` - сокращение от `install`, то есть «установить»;
- `-g` - флаг, который указывает на то, что пакет будет установлен глобально;
- `less` - имя устанавливаемого пакета;

Компилирование файла с именем `_styles.less` без сохранения результата:

```
$ lessc _styles.less
```

Компилирование файла `_styles.less` с сохранением результата в файл `_main.css`:

```
$ lessc _styles.less > _main.css
```

Существует около двух десятков различных других специфических параметров, благодаря которым ваша работа с препроцессором в командной строке станет куда гибче.

Компиляция, используя системы сборки

Система сборки — это инструмент для автоматической, гибкой и удобной сборки проектов из командной строки.

Первым популярным сборщиком был Grunt, позднее появился Gulp и самыми молодыми сейчас являются Brunch и Broccoli.

Не рассматриваем, т.к. это выходит за рамки курса

Приложения для компиляции

Самый простой и удобный способ для проектов, использующих препроцессоры, причём не только CSS-препроцессоры, но и JS и HTML.

Существуют такие приложения, которые позволяют управлять проектами без написания кода, использования командной строки и систем сборок. Они написаны для людей, желающих делать своё дело и не вникать в некоторые тонкости,

Такие приложения имеют довольно обширный функционал и, как правило, умеют:

- Компилировать файлы различных препроцессоров (Less, Stylus, Jade, CoffeeScript и т.д.);
- Проверять файлы на ошибки и соответствие правилам (общим или проекта);
- Обрабатывать файлы (минификация, расстановка префиксов в CSS и т.д.);
- Автоматизировать некоторые часто используемые действия;
- Локальный сервер для тестирования проектов на этапе разработки;

Примеры приложений: Prepros (платно), SimpLESS, Koala(больше не поддерживается), WinLess

Онлайн LESS компиляторы

http://lesscss.org/less-preview/

http://winless.org/online-less-compiler (много примеров)

ИТОГО

1. Создаем папку для проекта (если еще нет)
2. Создаем файл с расширением .less.
3. Добавляем папку с проектом в приложение. После этого приложение само будет следить за обновлением .less файлов и автоматически компилировать их в .css (можно отключить автоматический мониторинг и компилировать по нажатию на кнопку). Если добавлен новый файл .less, то нужно просто нажать кнопку “обновить” и приложение добавит его в мониторинг.
4. Редактируем проект и смотрим, что появился файл с расширением .css и мы видим обновления на странице

Далее посмотрим, какие возможности нам предоставляет Less

Переменные

В Less переменные динамические, переопределяемые и требующие инициализации.

Если переменным можно присваивать значения любого типа, то их называют *динамическими*.

Процесс объявления (определения) переменных и присваивания им значения называют *инициализацией*.

Обычно, переменные *переопределяемые*, то есть при инициализации в них можно записать значение и, по ходу работы программы, изменять его столько раз, сколько нужно разработчику.

```
@header-font: Georgia;
h1, h2, h3, h4 {
    font-family: @header-font;
}
.large {
    font-family:@header-font;
}
```

Хранение данных в переменных

```
@var-color: #ffff00;
@var-string-0: header;
@var-string-1: 0 auto;
@var-number: 4768;
@var-value: 14px;
@var-rule: {
  color: red;
};
@var-url-0: "../images/nichosi-meme.png";
@var-url-1: url("../images/nichosi-meme.png");
```

Less - вложенные правила

```less
// Variables
@header-background: #181e21;
@header-color: #fff;

.global-header {
  position: relative;
  background-color: @header-background;
  color: @header-color;

  h1 {
    font-size: 44px;
    line-height: 50px;

    small {
      font-size: 24px;
      line-height: 36px;
    }
  }
}
```

```css
.global-header {
  position: relative;
  background-color: #181e21;
  color: #ffffff;
}
.global-header h1 {
  font-size: 44px;
  line-height: 50px;
}
.global-header h1 small {
  font-size: 24px;
  line-height: 36px;
}
```

Операции с переменными (операторы)

@a	@b	@a + @b	@a - @b	@a * @b	@a / @b
1	1	2	0	1	1
1px	2	3px	-1px	2px	0.5px
5%	4	9%	1%	20%	1.25%
2%	3px	5%	-1%	6%	0.6666..66%
0.33	11%	11.33%	-10.67%	3.6300...03%	0.0300...02%
#6699cc	25%	#6b9ed1	#6194c7	#ffffff	#141f29
#666	#333	#999999	#333	#ffffff	#020202

Прямое объединение селекторов

Символ `&` играет особую и важную роль в Less. С помощью этого символа можно обращаться к родителю текущего селектора, а также сшивать, объединять их и создавать внутри них области видимости.

```
a {
  color: #777;
  text-decoration: none;

  &:hover {
    color: #a31515;
  }
}
```

```
a {
  color: #777;
  text-decoration: none;
}

a:hover {
  color: #a31515;
}
```

Обратное объединение селекторов

Представим, что нашему проекту требуется поддержка IE7, который не умеет работать со свойством `border-image`. Вместо того, чтобы писать новый класс `.ie7 .item-card {}` и плодить сложно поддерживаемую структуру, можно использовать принцип обратного объединения.

```
.main {
  .item-card {
    background-color: #f5f5f5;
    border-image: url("images/bg-image.png") 30
round round;

    .ie7 & {
      border: 1px solid #a31515;
    }
  }
}
```

```
.main .item-card {
  background-color: #f5f5f5;
  border-image: url("images/bg-image.png") 30 round
round;
}
.ie7 .main .item-card {
  border: 1px solid #a31515;
}
```

Сшивание (склеивание) селекторов

Стоит задача стилизации кнопок для проекта. Кнопки имеют различные цвета, в зависимости от действия и контекста использования. Изначально кнопки имеют серый цвет. Кнопка, предназначенная для добавления новой записи имеет зелёный цвет, а для удаления — красный.

Если опустить основные свойства, рассматривая лишь те, что необходимы для определения цвета кнопок, то получается следующий код:

```
.button {
  background-color: #ddd;
  color: #000;
}

.button-add {
  background-color: green;
  color: #fff;
}

.button-remove {
  background-color: red;
  color: #fff;
}
```

```
.button {
  background-color: #ddd;
  color: #000;

  &-add {
    background-color: green;
    color: #fff;
  }

  &-remove {
    background-color: red;
    color: #fff;
  }
}
```

Использование примесей

Примесь (от англ. mix-in) — набор свойств и селекторов, расширяющий поведение другой сущности (селектора).

```
.bordered() {
  border-top: dotted 1px #333;
  border-bottom: solid 2px #333;
}

.article {
  .bordered;
  color: #443d3d;
}
```

```
.article {
  border-top: dotted 1px #333;
  border-bottom: solid 2px #333;
  color: #443d3d;
}
```

В этом случае при компиляции не будет создан класс `.bordered`, так как у него указаны скобки после имени. Такая конструкция говорит компилятору, что она чистейшая примесь, которая не хочет быть скомпилирована без явных на то причин.

Примеси могут иметь параметры.

Переменные и примеси

Разделять параметры при объявлении примеси и её вызове можно запятыми `(@a, @b)`, либо точкой с запятой `(@a; @b)`.

```
.bordered(@_color) {
  border-top: dotted 1px @_color;
  border-bottom: solid 2px @_color;
}

.article {
  .bordered(#ccc);
  color: #443d3d;
}
```

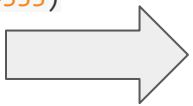
```
.article {
  border-top: dotted 1px #cccccc;
  border-bottom: solid 2px #cccccc;
  color: #443d3d;
}
```

Переменная @arguments

Кроме обычных локальных переменных для примеси, можно использовать специальные параметры, которые имеют несколько другое предназначение.
Переменная `@arguments` представляет собой синоним для всех переданных в примесь значений при её вызове. Такое поведение может быть полезно при использовании примеси с большим количеством переменных, предназначающихся для одного свойства.
Если примесь вызвана, а значения для параметров не были переданы или переданы частично, то ошибки компилятор не выдаст, а возьмёт значение, указанное по умолчанию. Если значения по умолчанию нет, то компилятор выдаст ошибку.

```
.box-shadow(@x: 0, @y: 0, @blur: 1px, @color: #333)
{
  -webkit-box-shadow: @arguments;
     -moz-box-shadow: @arguments;
          box-shadow: @arguments;
}

.big-block {
  .box-shadow(2px, 5px);
}
```



```
.big-block {
  -webkit-box-shadow: 2px 5px 1px #333;
     -moz-box-shadow: 2px 5px 1px #333;
          box-shadow: 2px 5px 1px #333;
}
```

Слияние свойств

Допустим, что имеется код, который генерирует одно и тоже свойство, но с разными значениями. В первом случае это будет тень сверху блока, а во-втором — снизу.
С таким подходом можно встретиться в модном сейчас направлении — материальный дизайн.
Проблема в том, что второе свойство переопределит первое. Это приведёт к тому, что тень не будет состоять из верхней и нижней частей — браузер покажет лишь нижнюю, так как она стоит ниже в селекторе и заменяет собой верхнюю.

```
.depth-top() {
  box-shadow: 0 2px 5px 0 rgba(0, 0, 0, 0.16);
}

.depth-bottom() {
  box-shadow: 0 2px 10px 0 rgba(0, 0, 0, 0.12);
}

.block {
  .depth-top();
  .depth-bottom();
}
```

Благодаря Less значения свойств можно конкатенировать, и делается это двумя способами:

- Через запятую (свойства `box-shadow`, `font-family` и т.д.)
- Через пробел (свойства `transform`, `text-overflow` и т.д.)

Слияние свойств через запятую

Для того, чтобы решить проблему, просто добавим плюс (+) перед двоеточием свойства. Это скажет компилятору, что при встречи двух одинаковых свойств в одном селекторе — его целью будет их объединение.

```
.depth-top() {
  box-shadow+: 0 2px 5px 0 rgba(0, 0, 0, 0.16);
}

.depth-bottom() {
  box-shadow+: 0 2px 10px 0 rgba(0, 0, 0, 0.12);
}
```

После компиляции результат оправдает ожидание. Свойство одно и оно включает в себя конкатенированное значение двух примесей, записанных через запятую, что и соответствует синтаксису этого свойства.

```
.block {
  box-shadow: 0 2px 5px 0 rgba(0, 0, 0, 0.16), 0 2px 10px 0 rgba(0, 0, 0, 0.12);
}
```

Слияние свойств через пробел

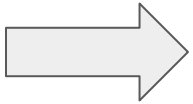
По сути своей, никакого отличия от слияния свойств через запятую здесь нет — добавляется лишь нижнее подчёркивание после плюса перед двоеточием.

```
.scale(@scale) {
  transform+_: scale(@scale);
}

.rotate(@angle) {
  transform+_: rotate(@angle);
}

.translate(@px) {
  transform+_: translateX(@px);
}

.block {
  .scale(1.75);
  .rotate(45deg);
  .translate(10px);
}
```



В результате, после компиляции получим следующее валидное для этого свойства значение:

```
.block {
  transform: scale(1.75) rotate(45deg) translateX(10px);
}
```

Что еще?

Строки и списки

Работа с изображениями

Математические функции

Условные конструкции

Циклические конструкции

http://lesscss.org/

Sass. Чем отличается SASS от SCSS

Самый мощный из CSS-препроцессоров. Имеет довольно большое сообщество разработчиков. Основан в 2007 году как модуль для HAML и написан на Ruby (есть порт на C++).

Имеет куда больший ассортимент возможностей в сравнении с Less.

Возможности самого препроцессора расширяются за счёт многофункциональной библиотеки Compass, которая позволяет выйти за рамки CSS и работать, например, со спрайтами в автоматическом режиме.

Имеет два синтаксиса:

- Sass (Syntactically Awesome Style Sheets) — упрощённый синтаксис CSS, который основан на идентации. Считается устаревшим.
- SCSS (Sassy CSS) — основан на стандартном для CSS синтаксисе.

Переменные в Sass

```
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
 font: 100% $font-stack;
 color: $primary-color;
}
```

CSS Output

```
body {
 font: 100% Helvetica, sans-serif;
 color: #333;
}
```

Примеси (Mixins)

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


.box { @include border-radius(10px); }
```

CSS Output

```
.box {
 -webkit-border-radius: 10px;
 -moz-border-radius: 10px;
 -ms-border-radius: 10px;
 border-radius: 10px;
}
```

Вложенные селекторы

```
nav {
 ul {
 margin: 0;
 padding: 0;
 list-style: none;
 }

 li { display: inline-block; }

 a {
 display: block;
 padding: 6px 12px;
 text-decoration: none;
 }
}
```

CSS Output

```
nav ul {
 margin: 0;
 padding: 0;
 list-style: none;
}

nav li {
 display: inline-block;
}

nav a {
 display: block;
 padding: 6px 12px;
 text-decoration: none;
}
```

Препроцессор Stylus

- так как Stylus основан на Node.js, то отпадает необходимость в использовании сторонней технологии (Sass требует для своей работы Ruby)
- Stylus предоставляет набор JavaScript API, что делает возможным дальнейшую настройку этого инструмента
- синтаксис Stylus не нуждается в скобках (brackets), запятых (comma), двоеточиях (colons), точках с запятыми (semicolon) - он целиком и полностью основан на использовании табуляции и пробелов; но если необходимо использовать любой из видов пунктуации, то его можно легко применить в Stylus - компиляция произойдет корректно
- под препроцессор Stylus имеется готовая библиотека миксинов (mixin) под названием Nib

Основы синтаксиса Stylus

```
/*  Простая переменная  */
  base-font-size = 12px

/* Инициализация переменной с помощью вызова миксина */
body-background = invert(#ccc)

/* Селектор и набор правил для него */
body
  color #333
  background #fff
```

Основы синтаксиса Stylus

```
/* Вложенность правил */
nav
  margin 10px
    ul
      list-style-type none
        > li
          display inline-block
          &.current
            background-color lightblue

/* Использование вычисляемых значений */
div.column
  margin-right (grid-spacing / 2)
  margin-top (grid-spacing * 2)
```

Основы синтаксиса Stylus

```
/* Использование ранее установленного значения */
div.center-column
  width 200px
  margin-left -(@width / 2)

/* Задание значений, полученных как результат вычислений миксинов */
.promo
  apply-promo-style()
  apply-width-center(400px)

/* Итерация в цикле */
table
  for row in 1 2 3 4 5
    tr:nth-child( row)
      height 10px * row

/* Другой вариант итерации в цикле */
for row in (1..5)
  tr:nth-child(row)
    height 10px * row
```

Сравнение. Переменные

Less

```
@mainColor: #0982c1;
@siteWidth: 1024px;
@borderStyle: dotted;

body {
  color: @mainColor;
  border: 1px @borderStyle
@mainColor;
  max-width: @siteWidth;
}
```

SaSS

```
$mainColor: #0982c1;
$siteWidth: 1024px;
$borderStyle: dotted;

body {
  color: $mainColor;
  border: 1px $borderStyle
$mainColor;
  max-width: $siteWidth;
}
```

Stylus

```
mainColor = #0982c1
siteWidth = 1024px
$borderStyle = dotted

body
  color mainColor
  border 1px $borderStyle
mainColor
  max-width siteWidth
```

Сравнение. Примеси

Less

```
/* LESS mixin error with (optional) argument
@borderWidth which defaults to 2px if not specified */

.error(@borderWidth: 2px) {
  border: @borderWidth solid #F00;
  color: #F00;
}

.generic-error {
  padding: 20px;
  margin: 4px;
  .error(); /* Applies styles from mixin error */
}
.login-error {
  left: 12px;
  position: absolute;
  top: 20px;
  .error(5px); /* Applies styles from mixin error with
argument @borderWidth equal to 5px */
}
```

SaSS

```
/* Sass mixin error with (optional) argument
$borderWidth which defaults to 2px if not specified */


@mixin error($borderWidth: 2px) {
  border: $borderWidth solid #F00;
  color: #F00;
}

.generic-error {
  padding: 20px;
  margin: 4px;
  @include error(); /* Applies styles from mixin error */
}
.login-error {
  left: 12px;
  position: absolute;
  top: 20px;
  @include error(5px); /* Applies styles from mixin
error with argument $borderWidth equal to 5px*/
}
```

Stylus

```
/* Stylus mixin error with (optional) argument
borderWidth which defaults to 2px if not specified
*/

error(borderWidth= 2px) {
  border: borderWidth solid #F00;
  color: #F00;
}

.generic-error {
  padding: 20px;
  margin: 4px;
  error(); /* Applies styles from mixin error */
}
.login-error {
  left: 12px;
  position: absolute;
  top: 20px;
  error(5px); /* Applies styles from mixin error with
argument borderWidth equal to 5px */
}
```