

Инструменты разработчика (часть 2)

LESS

Динамический язык стилевой разметки.

LESS расширяет CSS динамическими возможностями, такими как переменные, примешивания, операции и функции.

LESS может использоваться как на стороне клиента (IE 6+, Webkit, Firefox), так и на стороне сервера, с Node.js и Rhino.

Пример:

```
@base: #f938ab;

.box-shadow(@style, @c) when (iscolor(@c)) {
  box-shadow:      @style @c;
  -webkit-box-shadow: @style @c;
  -moz-box-shadow:  @style @c;
}
.box-shadow(@style, @alpha: 50%) when (isnumber(@alpha)) {
  .box-shadow(@style, rgba(0, 0, 0, @alpha));
}
.box {
  color: saturate(@base, 5%);
  border-color: lighten(@base, 30%);
  div { .box-shadow(0 0 5px, 30%) }
}
```

Переменные

Переменные позволяют определить постоянно используемые значения в одном месте, а затем повторно использовать их в любом месте таблицы стилей, что облегчает внесение глобальных изменений буквально до изменения одной строки кода.

Пример:

```
@nice-blue: #5B83AD;
@light-blue: @nice-blue + #111;

#header { color: @light-blue; }
```

что дает на выходе:

```
#header { color: #6c94be; }
```

Можно также определять переменную, ссылаясь на другую переменную по ее имени:

Пример:

```
@fnord: "I am fnord.";
@var: 'fnord';
content: @@var;
```

Что компилируется в:

```
content: "I am fnord.";
```

Заметьте, что то, что мы называем переменными — на самом деле ‘константы’, поскольку они могут быть определены только раз.

Интерполяция строк

Переменные могут быть включены внутрь строк, как в ruby или в PHP, при помощи конструкции `@{name}`:

```
@base-url: "http://assets.fnord.com";
background-image: url("@{base-url}/images/bg.png");
```

Примешивания

В LESS возможно включать целый набор свойств из одного набора правил в другой набор.

Скажем, мы имеем следующий класс:

```
.bordered {
  border-top: dotted 1px black;
  border-bottom: solid 2px black;
}
```

И мы хотим использовать эти свойства внутри другого набора правил. Что же, мы просто указываем имя класса в любом другом наборе правил, куда нужно включить свойства, вот так:

```
#menu a {
  color: #111;
  .bordered;
}
.post a {
  color: red;
  .bordered;
```

```
}
```

Свойства класса `.bordered` теперь появятся и в `#menu a`, и в `.post a`.

Результатом компиляции окажется:

```
#menu a {  
  color: #111;  
  border-top: dotted 1px black;  
  border-bottom: solid 2px black;  
}  
.post a {  
  color: red;  
  border-top: dotted 1px black;  
  border-bottom: solid 2px black;  
}
```

Любой класс CSS, а также набор правил `id` или элемента может быть смешан точно так же.

Параметризированные примеси

В LESS существует особый вид наборов правил, которые можно смешивать, точно как описываемые выше классы, но которые принимают параметры.

Вот канонический пример:

```
.border-radius (@radius) {  
  border-radius: @radius;  
  -moz-border-radius: @radius;  
  -webkit-border-radius: @radius;  
}
```

А вот как мы можем подмешивать этот набор правил в другие наборы:

```
#header {  
  .border-radius(4px);  
}  
.button {  
  .border-radius(6px);  
}
```

Параметризированные примеси могут также иметь значения по-умолчанию для своих параметров:

```
.border-radius (@radius: 5px) {  
  border-radius: @radius;  
  -moz-border-radius: @radius;  
  -webkit-border-radius: @radius;  
}
```

И, если мы используем такой вызов:

```
#header {  
  .border-radius;
```

```
}
```

то получим на выходе значение border-radius, равное 5px.

Переменная с именем @arguments

Название переменной @arguments внутри примешивания имеет особое значение. В процессе компиляции заменяется на все переданные аргументы.

Это удобно, если нет желания возиться с отдельными параметрами:

```
.box-shadow (@x: 0, @y: 0, @blur: 1px, @color: #000) {  
  box-shadow: @arguments;  
  -moz-box-shadow: @arguments;  
  -webkit-box-shadow: @arguments;  
}  
  
.box-shadow(2px, 5px);
```

Импортирование

Вы можете импортировать файлы .less, при этом все переменные и примеси в них становятся доступными в главном файле. Расширение .less необязательно, так что обе следующие записи работают одинаково:

```
@import "lib.less";  
@import "lib";
```

Если Вы хотите импортировать файл CSS, и не хотите, чтобы LESS обрабатывал его содержимое, просто используйте расширение .css:

```
@import "lib.css";
```

Директива будет оставлена как есть, и в этом же виде попадет в вывод CSS.

Совпадение с шаблоном и предохранительные выражения

Иногда, Вы можете захотеть изменить поведение примешивания, в зависимости от переданный в него параметров.

Предположим, что мы хотим, чтобы .mixin вела себя по-разному, в зависимости от величины @switch.

Определяем .mixin таким образом:

```
.mixin (dark, @color) {  
  color: darken(@color, 10%);  
}
```

```
.mixin (light, @color) {
  color: lighten(@color, 10%);
}
.mixin (@_, @color) {
  display: block;
}
```

Теперь, если мы используем:

```
@switch: light;
```

```
.class {
  .mixin(@switch, #888);
}
```

Мы получим такой CSS:

```
.class {
  color: #a2a2a2;
  display: block;
}
```

Т.е. величина цвета (@color), переданная в .mixin, оказалась осветлена. Если бы значение @switch было задано как dark, результатом стал бы более темный цвет.

Мы можем также сравнивать количество параметров, например:

```
.mixin (@a) {
  color: @a;
}
.mixin (@a, @b) {
  color: fade(@a, @b);
}
```

Если мы вызовем .mixin с одним аргументом, мы получим вывод первого определения, но если мы вызовем примешивание с двумя аргументами, мы получим второе определение, а именно @a, приглушенное на @b.

Предохранители

Предохранители полезны, когда мы хотим проверять совпадение с выражениями, а не с простыми величинами или с количеством параметров.

LESS использует условное исполнение при помощи предохраняемых примешиваний вместо выражение if/else, в стиле спецификации указания @media.

Пример:

```
.mixin (@a) when (lightness(@a) >= 50%) {
  background-color: black;
}
```

```
.mixin (@a) when (lightness(@a) < 50%) {
  background-color: white;
}
.mixin (@a) {
  color: @a;
}
```

Ключевым являемся указание when, которое описывает предохранительное выражение (здесь всего один предохранитель).

Теперь, если запустить код:

```
.class1 { .mixin(#ddd) }
.class2 { .mixin(#555) }
```

Мы получим:

```
.class1 {
  background-color: black;
  color: #ddd;
}
.class2 {
  background-color: white;
  color: #555;
}
```

Полный список операторов сравнения в предохраняющих выражениях: > >= = =< <. Кроме того, ключевое слово true - всегда дающее истину выражение, делающее две следующие примеси эквивалентными:

```
.truth (@a) when (@a) { ... }
.truth (@a) when (@a = true) { ... }
```

Любые выражения, кроме true, являются ложью:

```
.class {
  .truth(40); // Не совпадет ни с одним из выражений.
}
```

Предохранители могут разделяться запятыми ',' — если один из них срабатывает, весь набор считается совпадшим:

```
.mixin (@a) when (@a > 10), (@a < -10) { ... }
```

Можно сравнивать аргументы друг с другом, либо с не-аргументами:

```
@media: mobile;
.mixin (@a) when (@media = mobile) { ... }
.mixin (@a) when (@media = desktop) { ... }
.max (@a, @b) when (@a > @b) { width: @a }
.max (@a, @b) when (@a < @b) { width: @b }
```

Наконец, если Вы хотите сравнивать примешивания по типу значения, Вы можете использовать функции вида is:

```
.mixin (@a, @b: 0) when (isnumber(@b)) { ... }  
.mixin (@a, @b: black) when (iscolor(@b)) { ... }
```

Функции для сравнения основных типов:

```
iscolor  
isnumber  
isstring  
iskeyword  
isurl
```

Если необходимо проверить, чтобы величина была не только числом, но и имела бы значение определенного типа:

```
ispixel  
ispercentage  
isem
```

Можно использовать ключевое слово **and** для указания дополнительных условий, например:

```
.mixin (@a) when (isnumber(@a)) and (@a > 0) { ... }
```

И ключевое слово **not** для отрицания условия:

```
.mixin (@b) when not (@b > 0) { ... }
```

Вложенные правила

LESS дает возможность вкладывать определения.

Пусть, например, у нас есть такой CSS:

```
#header { color: black; }  
#header .navigation {  
    font-size: 12px;  
}  
#header .logo {  
    width: 300px;  
}  
#header .logo:hover {  
    text-decoration: none;  
}
```

При использовании LESS, мы также можем записать его в виде:

```
#header {  
    color: black;  
  
    .navigation {  
        font-size: 12px;  
    }  
    .logo {
```

```
    width: 300px;
    &:hover { text-decoration: none }
  }
}
```

либо так:

```
#header      { color: black;
  .navigation { font-size: 12px }
  .logo       { width: 300px;
    &:hover    { text-decoration: none }
  }
}
```

Итоговый код более понятен, и повторяет структуру вашего дерева DOM. Обратите внимание на элемент `&` — он используется, когда указанный селектор нужно приписать к селектору-родителю, а не использовать как вложенный элемент. Это особенно удобно для псевдо-классов типа `:hover` или `:focus`.

Например, запись:

```
.bordered {
  &.float {
    float: left;
  }
  .top {
    margin: 5px;
  }
}
```

даст на выходе

```
.bordered.float {
  float: left;
}
.bordered .top [
  margin: 5px;
}
```

Операции

С любыми числовыми и цветовыми значениями можно производить математические операции. Вот несколько примеров:

```
@base: 5%;
@filler: @base * 2;
@other: @base + @filler;
```

```
color: #888 / 4;
background-color: @base-color + #111;
height: 100% / 2 + @filler;
```


Итоговый код будет вполне логичным — LESS распознает разницу между указаниями цветов и другими единицами измерений.

Если единицы измерений используются в операции, скажем, так:

```
@var: 1px + 5;
```

LESS использует указанную единицу измерения для итогового кода — что даст, в данном случае, 6px.

Скобки также разрешены в операциях:

```
width: (@var + 5) * 2;
```

И даже обязательны в составных значениях:

```
border: (@width * 2) solid black;
```

Функции работы с цветом

LESS предлагает множество функций, изменяющих указание цвета. Цвета сначала преобразуются в цветовую модель HSL (от англ. Hue, Saturation, Lightness — цветовая модель, в которой цветовыми координатами являются тон, насыщенность и светлота), а затем манипуляции ведутся с каждым каналом по отдельности:

```
lighten(@color, 10%); // цвет, который на 10% *светлее*, чем @color
```

```
darken(@color, 10%); // цвет, который на 10% *темнее*, чем @color
```

```
saturate(@color, 10%); // цвет, на 10% *более* насыщенный, чем @color
```

```
desaturate(@color, 10%); // цвет, на 10% *менее* насыщенный, чем @color
```

```
fadein(@color, 10%); // цвет, на 10% *менее* прозрачный, чем @color
```

```
fadeout(@color, 10%); // цвет, на 10% *более* прозрачный, чем @color
```

```
spin(@color, 10); //цвет, на 10 градусов больший по оттенку, чем @color
```

```
spin(@color, -10); //цвет, на 10 градусов меньший по оттенку, чем @color
```

Пример:

```
@base: #f04615;
```

```
.class {  
  color: saturate(@base, 5%);  
  background-color: lighten(spin(@base, 8), 25%);  
}
```

Вы также можете извлекать информацию о цвете:

```
hue(@color); // возвращает канал тона (`hue`) в указанном @color
```

```
saturation(@color); // возвращает канал насыщенности (`saturation`) в указанном @color
```

lightness(@color); // возвращает канал освещенности (`lightness`) в указанном @color

Это полезно, если Вы хотите создать новый цвет, основываясь на значениях каналов другого цвета, например:

```
@new: hsl(hue(@old), 45%, 90%);
```

в результате @new получит тон (hue), равный тону цвета @old, но при этом будет иметь свои собственные насыщенность и светлоту.

Математические функции

LESS предлагает пару удобных математических функций, которые можно применять к числовым значениям:

```
round(1.67); // вернет `2`
```

```
ceil(2.4); // вернет `3`
```

```
floor(2.6); // вернет `2`
```

Если Вам нужно преобразовать значение в проценты, это можно сделать при помощи функции `percentage`:

```
percentage(0.5); // вернет `50%`
```

Пространства имен

Иногда появляется необходимость сгруппировать переменные или примеси, либо для удобства организации кода, либо просто для некоторой инкапсуляции. Вы можете сделать это весьма интуитивно понятно в LESS — скажем, чтобы собрать несколько примесей и переменных под #bundle, для дальнейшего использования, достаточно указать:

```
#bundle {  
  .button () {  
    display: block;  
    border: 1px solid black;  
    background-color: grey;  
    &:hover { background-color: white }  
  }  
  .tab { ... }  
  .citation { ... }  
}
```

Теперь, чтобы примешать .button к нашему классу #header a, указываем:

```
#header a {  
  color: orange;  
  #bundle > .button;  
}
```

Зоны видимости

Зоны видимости в LESS похожи на одноименное понятие, применяемые в языках программирования. Переменные и примеси сначала ищутся локально, а затем, если компилятор не смог их найти, поиск продолжается в родительской зоне видимости, затем выше, и т.д.

```
@var: red;

#page {
  @var: white;
  #header {
    color: @var; // white
  }
}

#footer {
  color: @var; // red
}
```

Компиляция

Кроссплатформенные

SimpLESS <http://wearekiss.com/simpless>

Crunch! <http://crunchapp.net/> (на Adobe Air, с редактором)

Windows

WinLess <http://winless.org>

Mac OS X

Less.app <http://incident57.com/less/>

Online

<http://winless.org/online-less-compiler>

Использование на стороне клиента

Подключите таблицу стилей .less при помощи указания rel со значением “stylesheet/less”:

```
<link rel="stylesheet/less" type="text/css" href="styles.less">
```

Затем скачайте less.js по ссылке вверху этой страницы, и подключите его в секции `<head>` вашей страницы, вот так:

```
<script src="less.js" type="text/javascript"></script>
```

Обратите внимание, что важно подключить таблицу стилей до скрипта.

Режим слежения

Режим слежения — функция на стороне клиента, при использовании которой стили обновляются автоматически по мере изменения.

Для того, чтобы включить ее, допишите `'#!watch'` к адресу в строке браузера, а затем обновите страницу.

Bootstrap

Bootstrap — самый популярный фронтэнд фреймворк для разработки адаптивных веб-проектов.

<http://getbootstrap.com> (официальный сайт)

<https://github.com/twbs/bootstrap> (проект на github)

<http://www.oneskyapp.com/docs/bootstrap/ru> (русский перевод)

Основные инструменты Bootstrap:

- Сетка (grid) — заранее заданные размеры колонок, которые можно сразу же использовать, например ширина колонки 90px относится к классу `.span2`, который мы можем использовать в CSS описании документа
- Шаблоны - Фиксированный или резиновый шаблон документа
- Типографика - Описания шрифтов, определение некоторых классов для шрифтов таких как код, цитаты и т.п.
- Медиа - Представляет некоторое управление изображениями и Видео
- Таблицы - Средства оформления таблиц, вплоть до добавления функциональности сортировки
- Формы - Классы для оформления не только форм но и некоторых событий происходящих с ними.
- Навигация - Классы оформления для Табов, Вкладок, Страничности, Меню и Тулбара
- Алерты - Оформление диалоговых окон, Подсказок и Всплывающих окон

Начало работы

Скачайте и распакуйте архив с файлами Bootstrap с сайта <http://getbootstrap.com>.

Структура файлов каталога:

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.min.css
│   ├── bootstrap-theme.css
│   └── bootstrap-theme.min.css
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphs-halflings-regular.eot
    ├── glyphs-halflings-regular.svg
    ├── glyphs-halflings-regular.ttf
    └── glyphs-halflings-regular.woff
```

Базовый шаблон

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap 101 Template</title>

    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet">

    <!-- HTML5 Shim and Respond.js IE8 support of HTML5 elements and
media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the page via
file:// -->
    <!--[if lt IE 9]>
      <script src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/
html5shiv.js"></script>
      <script src="https://oss.maxcdn.com/libs/respond.js/1.4.2/
respond.min.js"></script>
    <![endif]-->
  </head>
  <body>
    <h1>Hello, world!</h1>

    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="https://code.jquery.com/jquery.js"></script>
    <!-- Include all compiled plugins (below), or include individual
files as needed -->
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>

```

Вам нужно подключить 2 файла — bootstrap.min.css в разедле <head></head> страницы и bootstrap.min.js в конце страницы перед закрывающим тегом </body>.

Обратите внимание, что также вам потребуется подключить jquery.js, который необходим для работы JavaScript плагинов Bootstrap.

Сетка дизайна

Сетка дизайна используются для создания макетов страниц с помощью разбивки страницы на ряд строк и столбцов. Каждая полученная в результате такого разбиения ячейка может содержать контент.

Основные правила работы с сеткой дизайна в Bootstrap:

- Строки должны быть помещены в блок `.container` (фиксированная ширина) или `.container-fluid` (вся ширина экрана) для правильного выравнивания и заполнения;
- Используйте строки для создания горизонтальных групп столбцов;
- Контент должен быть помещен в колонках, и только колонки могут быть непосредственными потомками строк;
-

Для того, чтобы выровнять ваш контент по центру экрана оберните его в элемент с классом `.container`.

```
<div class="container">
  ...
</div>
```

Основа Twitter Bootstrap — динамическая 12-колонная сетка:

