

CSS-препроцессоры

Vira Huskova

CSS препроцессор — 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.

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

При помощи препроцессоров вы можете писать код, который нацелен на:

- Читабельность для человека
- Структурированность и логичность
- Производительность

Синтаксический сахар

Перед тем, как перейти к дальнейшему рассмотрению CSS-препроцессоров, рассмотрим понятие «синтаксический сахар».

Синтаксический сахар — это дополнения синтаксиса языка, 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.

Варианты CSS-препроцессоров



- Less



- Sass (SCSS)



- Stylus



- Closure Stylesheets



- CSS Crush

Смысл использования препроцессоров

Стандартный CSS — это сложно. Синтаксис без использования вложенности, которую предлагают CSS-препроцессоры, просто напросто сложен для зрительного восприятия.

Кроме того, нужно помнить имя родителя при вложенности. Отсутствие нормальных переменных и «функций» делает CSS-код узконаправленным.

Для использования препроцессоров нужно лишь установить программу, которая будет следить за файлами, предназначенными для препроцессора, и при их изменении будет компилировать содержимое этих файлов в чистый CSS-код.

Для более продвинутых пользователей есть специальные сборщики проектов.

Структура и логичность кода

Самым популярным предлагаемым функционалом любого CSS-препроцессора является возможность вкладывать селекторы друг в друга.

- 1. Родительский селектор
 - 1.1. Вложенный селектор
 - 1.2. Вложенный селектор
 - 1.2.1. Вложенный селектор
 - 1.3. Вложенный селектор

Примеси

Если говорить совсем кратко, то, используя **примеси** (Mixins), можно сделать код переиспользуемым. Это помогает избежать вспомогательных классов в разметке или дублирования свойств от селектора к селектору.

Less

Самый популярный препроцессор. Основан в 2009 году Алексис Сельер (Alexis Sellier) и написан на JavaScript (изначально был написан на Ruby, но Алексис вовремя сделал правильный шаг).

Имеет все базовые возможности препроцессоров и даже больше, но не имеет условных конструкций и циклов в привычном для нас понимании.

Основным плюсом является его простота, практически стандартный для CSS синтаксис и возможность расширения функционала за счёт системы плагинов.

Грань между Less & CSS

CSS

Фрагмент самого обычного CSS-кода, который используется в проектах.

Весь этот код хранится в файле `styles.css` (`main.css`, etc.), основное - синтаксис!

```
.area {  
    margin-right: auto;  
    margin-left: auto;  
}  
  
.area:before,  
.area:after {  
    display: table;  
    content: " "  
}  
  
.area:after {  
    clear: both;  
}  
  
@media (min-width: 768px) {  
    .area {  
        width: 750px;  
        padding-right: 15px;  
        padding-left: 15px;  
    }  
}
```


Возьмём стили и, ничего не меняя, поместим их в файл с таким же именем, но имеющим расширение `.less`.

Файл скомпилируется без ошибок и будет точно таким же по содержанию.

<code>_test.less</code>	<code>test.css</code>
<pre>1 .area { 2 margin-right: auto; 3 margin-left: auto; 4 } 5 6 .area:before, 7 .area:after { 8 display: table; 9 content: " "; 10 } 11 12 .area:after { 13 clear: both; 14 } 15 16 @media (min-width: 768px) { 17 .area { 18 width: 750px; 19 padding-right: 15px; 20 padding-left: 15px; 21 } 22 }</pre>	<pre>1 .area { 2 margin-right: auto; 3 margin-left: auto; 4 } 5 .area:before, 6 .area:after { 7 display: table; 8 content: " "; 9 } 10 .area:after { 11 clear: both; 12 } 13 @media (min-width: 768px) { 14 .area { 15 width: 750px; 16 padding-right: 15px; 17 padding-left: 15px; 18 } 19 } 20</pre>

На изображении выше представлено состояние less-файла до компиляции и после. Сразу видно, что они идентичны и различаются лишь наличием пустых строк после селекторов в Less-файле, а также расширением (если это не min версия).

Компиляция

Для того, чтобы браузер понимал код, написанный с использованием синтаксических конструкций препроцессора, его нужно компилировать в понятный для него язык.

CSS

Существует несколько вариантов того, как можно перейти от Less к CSS.

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.

Основан на идее подключения стилей с расширением `.less` к документу, используя стандартный тег `<link>`, но с изменённым атрибутом `rel`. А также осуществляется подключение файла библиотеки.

+ node 8 version

```
npm install less -g
```

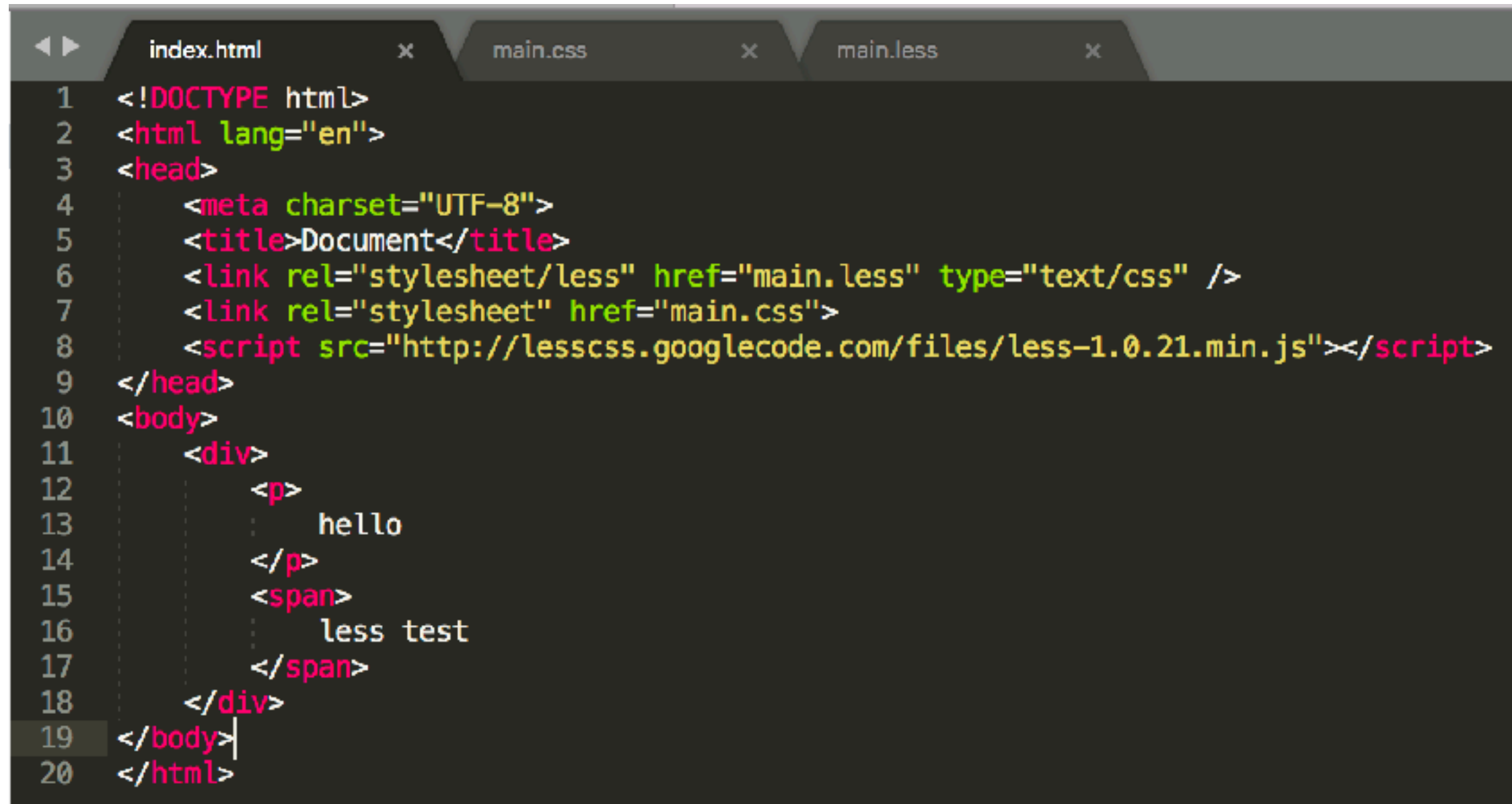
```
npm i less --save-dev
```

После компиляции скрипт проводит инъекцию полученного CSS-кода в секцию `head` документа посредством тега `<style>`.

Способ не желателен к применению на «продакшене» в виду того, что имеет серьёзные 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.

Пример демонстрирует использование CSS-препроцессора Less прямоком в браузере, без предварительной компиляции CSS.



The image shows a web browser window with three tabs: 'index.html', 'main.css', and 'main.less'. The 'index.html' tab is active, displaying the following HTML code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Document</title>
6     <link rel="stylesheet/less" href="main.less" type="text/css" />
7     <link rel="stylesheet" href="main.css">
8     <script src="http://lesscss.googlecode.com/files/less-1.0.21.min.js"></script>
9 </head>
10 <body>
11     <div>
12         <p>
13             hello
14         </p>
15         <span>
16             less test
17         </span>
18     </div>
19 </body>
20 </html>
```

```
index.html x main.css x main.less x
1  div {
2      background-color: pink;
3
4      p {
5          color: blue;
6      }
7
8      span {
9          color: green;
10     }
11 }
```

FOLDERS

- ▼ Less_test
 - <> index.html
 - /* main.css
 - /* main.less

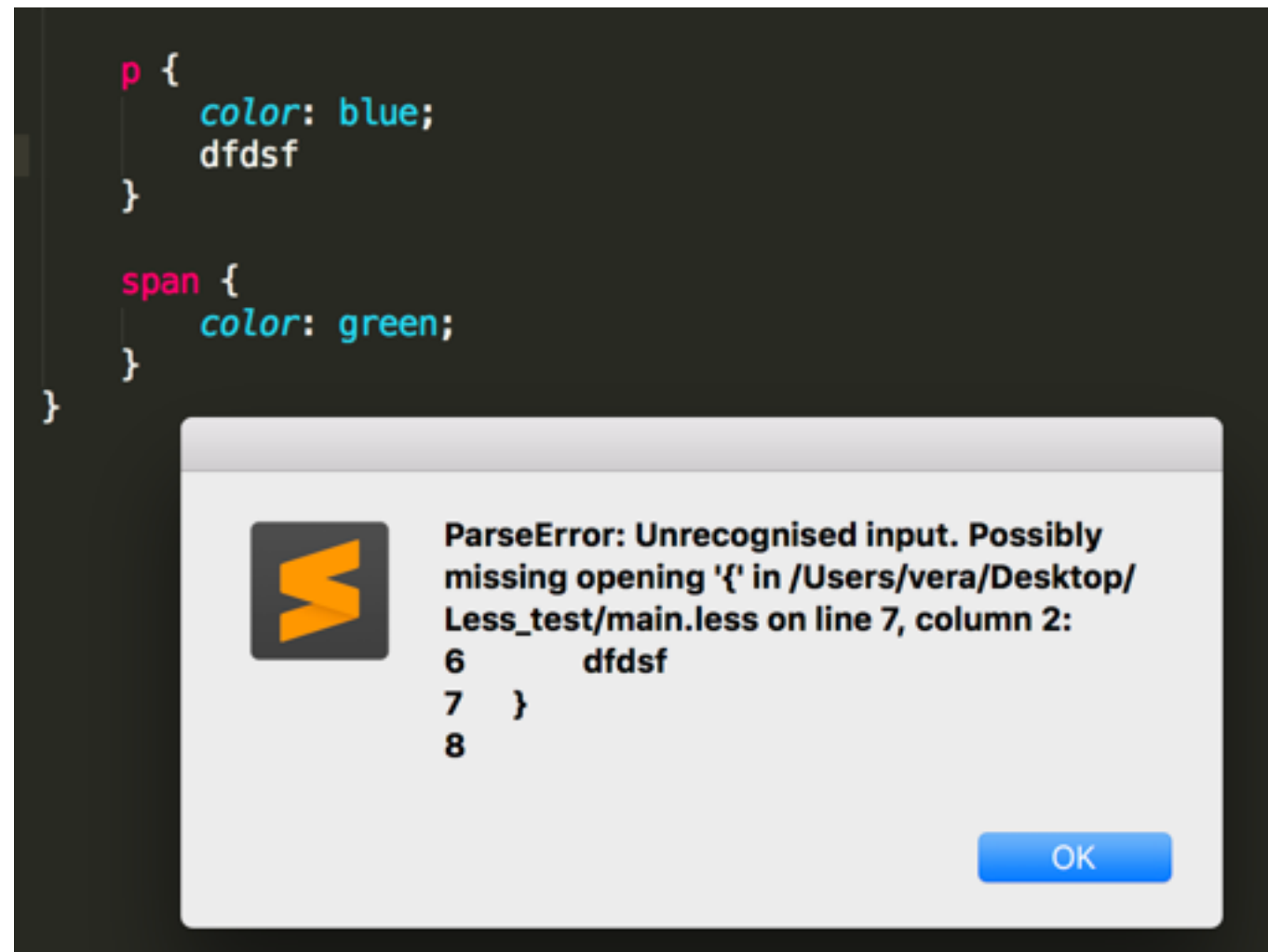
```
index.html x main.css x main.less x
1  div{background-color:pink}div p{color:#00f}div span{color:green}
```

hello

less test

Если в файле стилей допущена ошибка, то отобразится сообщение, в котором будет присутствовать следующая информация:

- Описание ошибки;
- Название файла, в котором обнаружена ошибка;
- Номер строки и столбца, где допущена ошибка;
- Код, порождающий ошибку;



Такой подробный отчёт позволит разработчику в удобной форме получить полную информацию о допущенной ошибке и в максимально короткие сроки приступить к её устранению.

При необходимости доступны некоторые настройки для управления преобразованиями, происходящими в препроцессоре и, как следствие, получаемыми данными на выходе. Причём настройки можно менять, используя JavaScript, так и с помощью атрибутов `data` у тега `link`.

В этом примере добавляется возможность автообновления страницы при изменении подключённого файла стилей. Для активации механизма слежения за изменением ресурсов, необходимо добавить к адресу страницы маркер `#!watch`. При изменении less-файлов в проекте, которые подключены к активной странице, будет происходить автоматическая компиляция этого файла.

```
<title>Document</title>
<link rel="stylesheet/less" href="main.less" type="text/css" />
<link rel="stylesheet" href="main.css">
<script src="http://lesscss.googlecode.com/files/less-1.0.21.min.js"></script>
<script>
  less = {
    env: 'development'
  };
</script>
<script src="less.min.js"></script>

<script>less.watch();</script>
</head>
```

```
ul {
  li {
    list-style-type: none;
    color: red;
  }
}
```

hello

less test

test

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
```

Помимо двух базовых команд можно передавать параметры скрипту, в зависимости от которых будет выполняться то или иное действие с вашим кодом.

Передача параметра `-x`, который говорит скрипту (компилятору), что на выходе пользователь ожидает увидеть **минифицированный файл**.

Приложения для компиляции

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

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);
- Обработать файлы (минификация, расстановка префиксов в CSS и т.д.);
- Автоматизировать некоторые часто используемые действия;
- Локальный сервер для тестирования проектов на этапе разработки;

Среди всех подобных приложений можно выделить следующие решения:

- Prepros
- CodeKit (только OS X)
- Mixture
- Koala


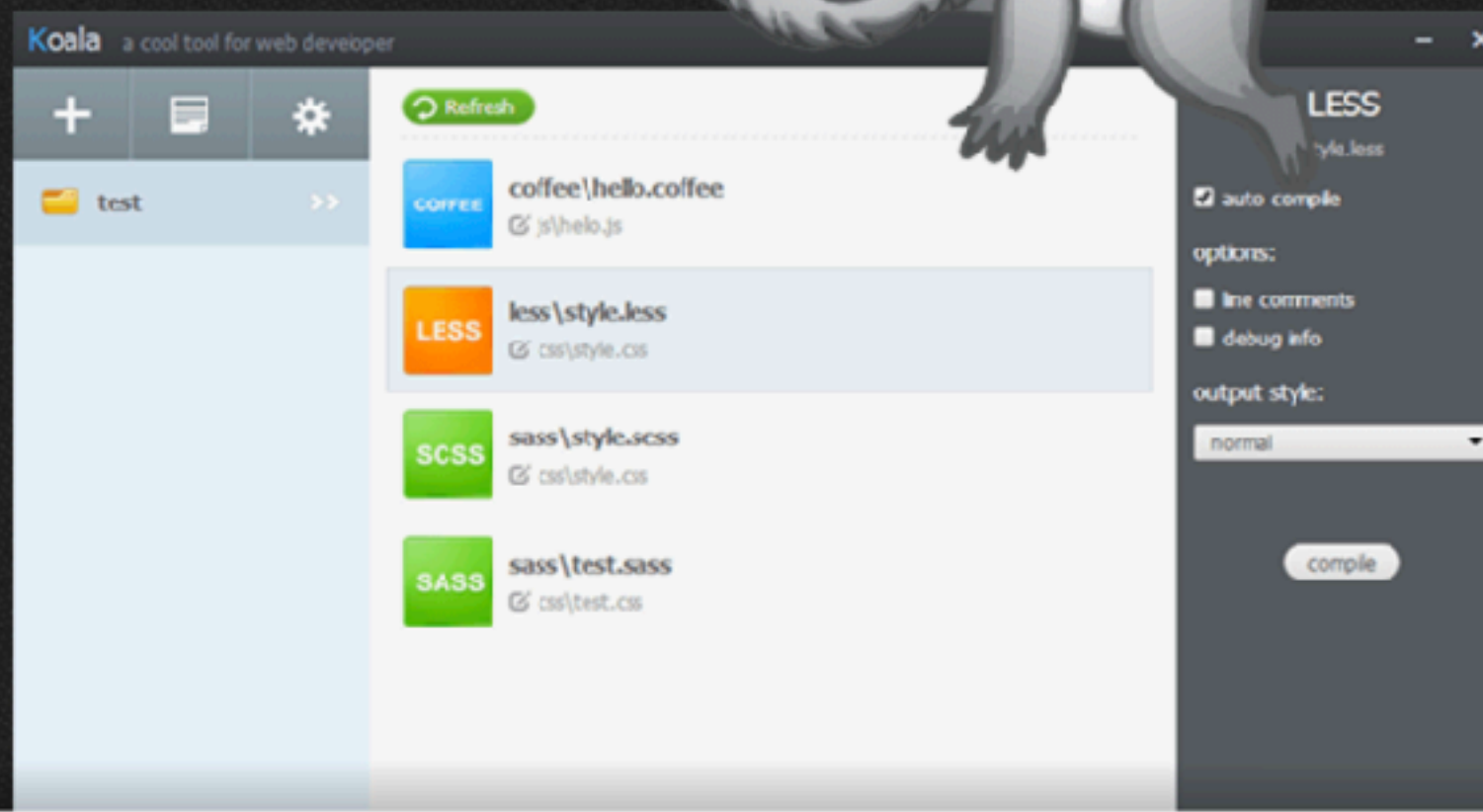
Koala

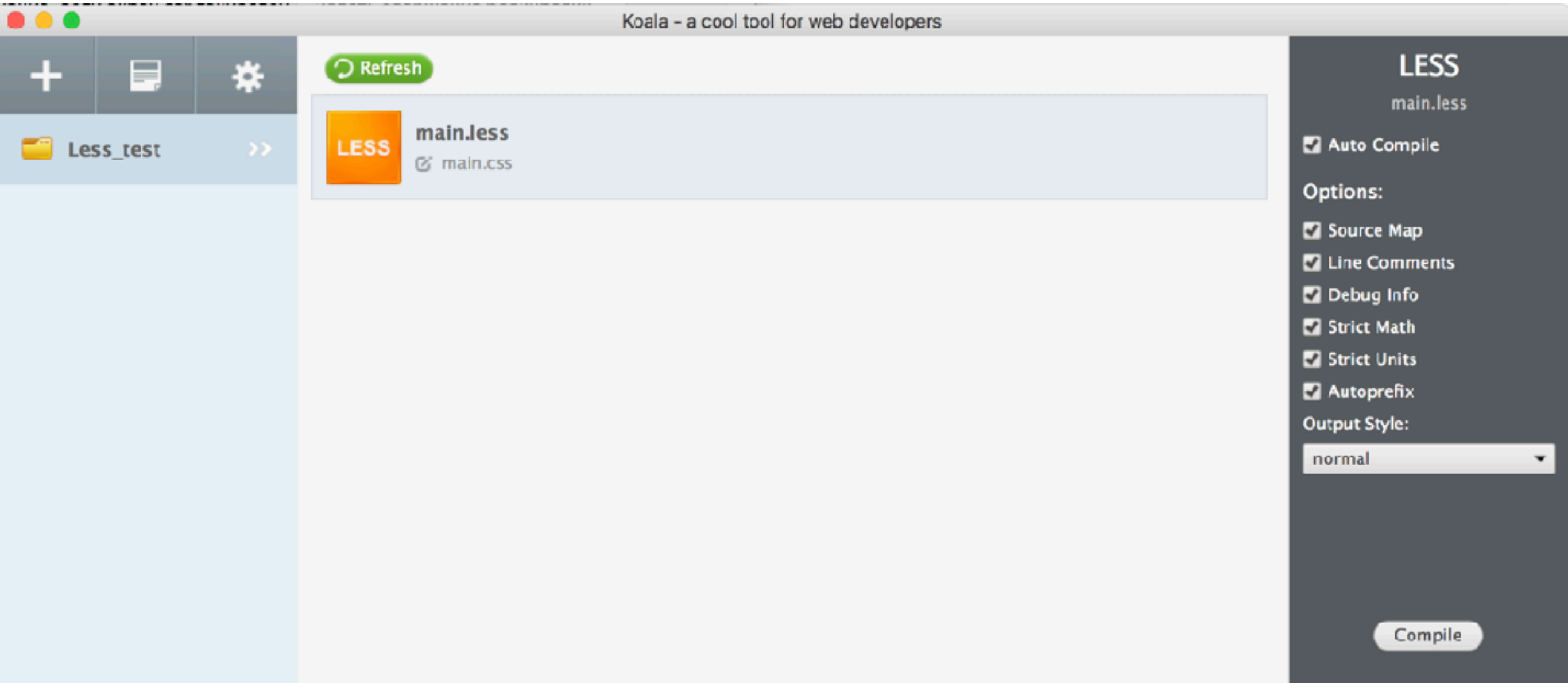
[Home](#)[Docs](#)[FAQ](#)[Changelog](#)[English](#)[简体中文](#)

Koala is a GUI application for Less, Sass, Compass and CoffeeScript compilation, to help web developers to use them more efficiently. Koala can run in windows, linux and mac.

 **Download**

Version 2.3.0, 2017-11-19

[Other system versions](#) [Donate](#) **Star** 3,671 **G+** **Tweet** **Нравится** 1,9 тыс



Альтернативные методы

Существуют решения для отдельно взятых сред (редакторов, IDE и т.д.), позволяющие использовать CSS-препроцессоры. В некоторых IDE есть встроенные средства для использования препроцессоров, а в тех, где нет, в общем случае, можно установить необходимые плагины, добавляющие такую возможность. Список таких плагинов.

<http://lesscss.org/usage/#editors-and-plugins>

Если для использования препроцессора Less на сервере, построенном с применением Node.js, требуется лишь официальный пакет доступный в npm, то на других платформах необходимы специальные библиотеки, а иногда и несколько.

Такой подход обеспечивает обмен переменными между Less и использующим его языком, что позволяет добиться компиляции файлов, в зависимости от контекста действий пользователя в приложении.

Отладка

Отладка — это процесс обнаружения, локализации и исправления возникающих ошибок в работе приложения.

В случае с CSS-препроцессором, приложением будут являться препроцессорные файлы, так как именно в них могут возникать ошибки или «неточности». И если с ошибками справляется компилятор, то исправлять «неточности» будет сложнее, из-за некоторых особенностей препроцессоров.

Карта кода (Source Maps)

Во время разработки и после неё, скомпилированные файлы стилей и исходные файлы могут сильно различаться. Происходит это из-за компиляции и обработки файлов. Если в процессе компиляции происходит раскрытие конструкций, написанных на препроцессорном языке в «чистый» CSS-код, то на выходе получается, как правило, большее количество строк кода. Выражается это в том, что в инспекторе браузера у тега стилей элемента указан один номер строки, а на самом деле он совсем другой.

Препроцессорные и скомпилированные файлы. Отличие в количестве строк, работе препроцессора и её результат на выходе.

```
// 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;
    }
  }
}
```

```
.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;
}
```


Когда файл предназначен для браузера, то имеется в виду, что понять его человеку будет практически невозможно:

```
{
  "version": 3,
  "sources": [
    "_styles.less"
  ],
  "names": [],
  "mappings": "AAIA;EACE,kBAAA;EACA,yBAAA;EACA,cAAA;;AAHF,cAKE;EACE,eAAA;EACA,iBAAA;;AAPJ,
  "file": "undefined"
}
```

Но было бы здорово, если бы слитый в один файл и минифицированный код в production-окружении можно было удобно читать и даже отлаживать без ущерба производительности.

Source map

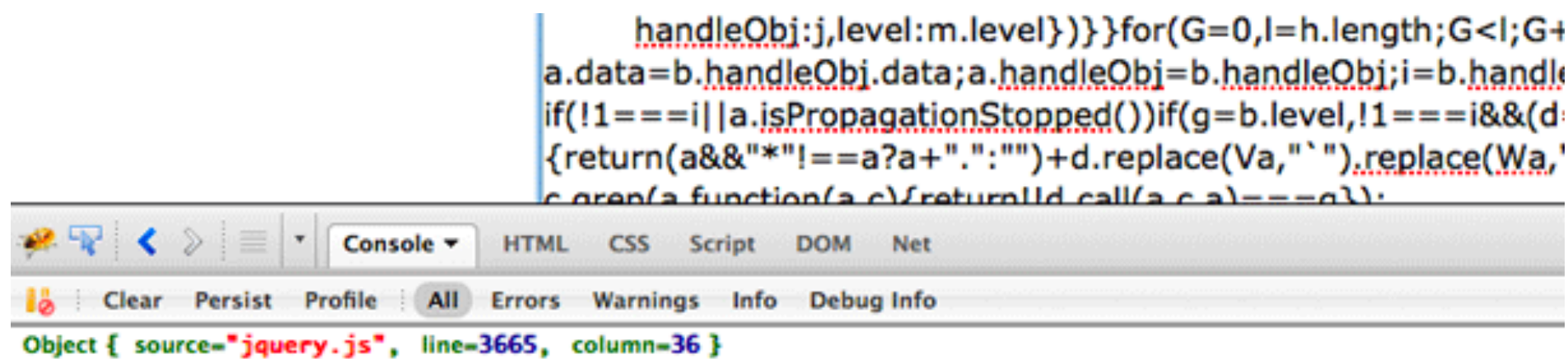
Это способ связать минифицированный/объединённый файл с файлами, из которых он получился.

Во время сборки для боевого окружения помимо минификации и объединения файлов также генерируется файл-маппер, который содержит информацию об исходных файлах.

Когда производится обращение к конкретному месту в минифицированном файле, то производится поиск в маппере, по которому вычисляется строка и символ в исходном файле.

Developer Tools (WebKit nightly builds или Google Chrome Canary) умеет парсить этот файл автоматически и прозрачно подменять файлы, как будто ведётся работа с исходными файлами.

https://docs.google.com/document/d/1U1RGAehQwRypUTovF1KRlpiOFze0b-_2gc6fAH0KY0k/edit?hl=en_US&pli=1&pli=1



Работа с селекторами, медиа-запросами и файлами

Вложенные правила

В хорошо структурированных таблицах стилей нет необходимости присваивать каждому элементу классы.

Достаточно лишь более подробно описывать стили элементов, используя возможность вкладывать селекторы в другие селекторы.

К слову, такие селекторы называются **вложенными** и представляют собой объёмную структуру.

Это хорошо пока имена классов короткие, глубина вложенности не велика.

```
.global-header {  
  background-color: #f5f5f5;  
  color: #443d3d;  
  border-bottom: 1px solid #ddd;  
}  
  
.global-header h1 {  
  margin-top: 0;  
  margin-bottom: 0;  
  font-size: 4rem;  
}  
  
.global-header h1 small {  
  font-size: 2rem;  
}  
  
.global-header .header-actions {  
  background-color: #fff;  
  padding-top: 10px;  
  padding-bottom: 10px;  
  text-align: center;  
}
```

```
.global-header {  
  background-color: #f5f5f5;  
  color: #443d3d;  
  border-bottom: 1px solid #ddd;  
  
  h1 {  
    margin-top: 0;  
    margin-bottom: 0;  
    font-size: 4rem;  
  
    small {  
      font-size: 2rem;  
    }  
  }  
  
  .header-actions {  
    background-color: #fff;  
    padding-top: 10px;  
    padding-bottom: 10px;  
    text-align: center;  
  }  
}
```

_styles.less

```
1 .global-header {
2   background-color: #f5f5f5;
3   color: #443d3d;
4   border-bottom: 1px solid #ddd;
5
6   h1 {
7     margin-top: 0;
8     margin-bottom: 0;
9     font-size: 4rem;
10
11     small {
12       font-size: 2rem;
13     }
14   }
15
16   .header-actions {
17     background-color: #fff;
18     padding-top: 10px;
19     padding-bottom: 10px;
20     text-align: center;
21   }
22 }
```

styles.css

```
1 .global-header {
2   background-color: #f5f5f5;
3   color: #443d3d;
4   border-bottom: 1px solid #ddd;
5 }
6 .global-header h1 {
7   margin-top: 0;
8   margin-bottom: 0;
9   font-size: 4rem;
10 }
11 .global-header h1 small {
12   font-size: 2rem;
13 }
14 .global-header .header-actions {
15   background-color: #fff;
16   padding-top: 10px;
17   padding-bottom: 10px;
18   text-align: center;
19 }
20
```

Предостережение!!!

Вкладывать селекторы друг в друга можно бесконечно, но делать это строго **не рекомендуется!**

Рекомендуется, что бы структура, в общем случае, не должна превышать **трёх вложений**. Нет необходимости вкладывать селекторы, начиная от родительского, на такую глубину.

Максимально допустимый уровень, в крайних случаях, *пять вложений*. Старайтесь избегать крайних случаев, если это действительно не требуется.

Ссылка на родителя селектора

Работа с псевдоклассами, псевдоэлементами и комбинированными селекторами. Для этого в Less есть специальный символ — `&`(родительский селектор).

С помощью этого символа можно обращаться к родителю текущего селектора, а также сшивать, объединять их и создавать внутри них области видимости.

Наиболее часто символ `&` применяется для добавления псевдоклассов к селекторам. Например, с помощью него можно добавить эффект при наведении, используя псевдокласс `:hover`.

```
a {  
  color: #777;  
  text-decoration: none;  
}  
  
a:hover {  
  color: #a31515;  
}
```

```
a {  
  color: #777;  
  text-decoration: none;  
  
  &:hover {  
    color: #a31515;  
  }  
}
```

Объединение селекторов с другими селекторами

```
.class-1 {  
  background-color: #fff;  
  
  &.class-2 {  
    color: #000;  
  }  
}
```

```
.class-1 {  
  background-color: #fff;  
}  
.class-1.class-2 {  
  color: #000;  
}
```

препроцессор не запрещает использовать стандартный синтаксис `.class-1.class-2 {}`. Но при таком стиле записи теряется весь смысл, который нам дарит препроцессор.

Обратное объединение селекторов

Возможность использования обратного объединения позволяет писать более гибкий код, когда нужно изменить контекст применения селектора.

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

Во время компиляции, селектор, у которого имеется ссылка на родителя справа от имени, будет опущен ниже по дереву вложенности вне зависимости от глубины вложенности. То есть при обратном объединении родительский селектор всегда ссылается на корневой селектор всего объявления.

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

Внимание!

Если написать `.ie7` & без пробела, то после компиляции селектор будет комбинированным, а не вложенным:

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

Склеивание селекторов

Не редко требуется производить операцию склеивания имён текущего и родительского селектора. Такая практика применяется при создании новых классов на основе старого.

Такая операция необходима тем людям, кто использует методологию БЭМ (и ей подобные) при написании стилей.

```
.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;  
  }  
}
```

```
.button {  
  background-color: #ddd;  
  color: #000;  
  
  &-add,  
  &-remove {  
    color: #fff;  
  }  
  
  &-add { background-color: green; }  
  &-remove { background-color: red; }  
}
```

Многократное и комбинированное использование

```
.header {  
  .item {  
    & + & {  
      color: red;  
    }  
  
    & & {  
      color: green;  
    }  
  
    && {  
      color: blue;  
    }  
  
    &, &-box {  
      color: yellow;  
    }  
  }  
}
```

```
.header .item + .header .item {  
  color: red;  
}  
  
.header .item .header .item {  
  color: green;  
}  
  
.header .item.header .item {  
  color: blue;  
}  
  
.header .item,  
.header .item-box {  
  color: yellow;  
}
```

Группировка селекторов

Для того, чтобы уменьшить количество кода после компиляции, а также упростить работу с селекторами, в Less был введён специальный псевдокласс **:extend()**.

Этот псевдокласс позволяет производить группировку селекторов (объединение) за счёт перечисления нескольких классов в одном месте, при условии, что все эти селекторы имеют общие свойства. Проще говоря, псевдокласс **:extend()** автоматизирует следующий процесс:

- Найти селекторы, у которых есть одинаковые свойства.
- Выбрать базовый селектор.
- Перечислить все найденные селекторы в объявлении базового селектора.
- Все новые селекторы добавлять в список селекторов базового объявления.

Под списком селекторов понимается последовательность селекторов, разделяемая с помощью запятой. Я более чем уверен, что вы уже встречали такие списки и даже их использовали в своих проектах.

```
.class-1,  
.class-2,  
.class-3 {  
  background-color: #fff;  
  color: #000;  
}
```

```
.class-1 {  
  background-color: #fff;  
  color: #000;  
}  
  
.class-2:extend(.class-1) {}  
  
.class-3 {  
  &:extend(.class-1);  
}
```

```
.class-1 {  
  background-color: #fff;  
  color: #000;  
}  
  
.class-2 {  
  background-color: #fff;  
  color: #000;  
}  
  
.class-3 {  
  background-color: #fff;  
  color: #000;  
}
```

Использование медиавыражений

```
@media (min-width: 992px) {  
  .class {  
    display: none;  
  }  
}
```

```
@media screen {  
  .one,  
  .test {  
    background-color: #fff;  
  }  
}  
  
@media screen and (min-width: 992px) {  
  .two,  
  .test {  
    color: #777;  
  }  
}  
  
.test {  
  border-right: 1px solid #000;  
}
```

```
@media (min-width: 768px) and (orientation: landscape) {  
  .class {  
    display: none;  
  }  
}  
  
@media tv and (min-width: 992px) and (orientation: landscape) {  
  .class {  
    display: block;  
  }  
}
```

```
.one {  
  @media (min-width: 768px) {  
    background-color: #f5f5f5;  
  
    .two {  
      @media (max-width: 992px) {  
        color: #000;  
      }  
    }  
  }  
}
```

Импорт стилей

Импорт стилей в CSS

В CSS директива `@import` позволяет импортировать стили из других таблиц.

Можно разбить одну большую таблицу стилей на несколько маленьких.

```
@import url("имя файла");  
@import "имя файла";
```

Самым важным минусом выступает тот факт, что такие подключения должны предшествовать другим стилям в таблице, где подключается дополнительный файл. То есть сделать так, как написано в коде ниже **нельзя**:

```
.class {  
    background-color: #fff;  
}  
  
@import "имя файла";
```

```
@import "имя файла";  
  
.class {  
    background-color: #fff;  
}
```


Импорт стилей в Less

В Less импорт стилей происходит с помощью всё той же директивы, но с расширенным функционалом. Перед именем файла можно указывать (необязательно) ключевое слово, которое указывает компилятору, как ему поступать с файлом.

```
@import (keyword) "имя файла";
```

Sr.No.	Import options & Description
1	reference ↗ It uses a LESS file only as reference and will not output it.
2	inline ↗ It enables you to copy your CSS into the output without being processed.
3	less ↗ It will treat the imported file as the regular LESS file, despite whatever may be the file extension.
4	css ↗ It will treat the imported file as the regular CSS file, despite whatever may be the file extension.
5	once ↗ It will import the file only one time.
6	multiple ↗ It will import the file multiple times.
7	optional ↗ It continues compiling even though the file to import is not found.

Опции импорта

Опция (less)

С помощью этого ключевого слова можно попросить компилятор рассматривать подключаемый файл как less-файл, то есть производить его компиляцию, а также конкатенацию с тем файлом, где происходит его подключение.

Опция (css)

Полная противоположность опции (less). На этот раз мы можем заставить любой файл подключаться стандартным для CSS способом.

```
@import (css) "import/_duckduckgo.less";

.canonium {
  color: #53599a;
}
```

```
@import "import/_duckduckgo.less";

.canonium {
  color: #53599a;
}
```

Опция (reference)

Замечательная опция, позволяющая использовать less-файлы, но не выводить их содержимое до тех пор, пока оно не будет явно вызвано. Пригодится в тех случаях, когда нужно использовать определённый селектор, а остальное содержимое файла не нужно. Такое поведение положительно сказывается на работе с библиотеками, которые имеют избыточный функционал в виде множества селекторов, а вам необходимы лишь некоторые из них.

Опция (inline)

Задача этой опции сказать компилятору, что разработчик ожидает на выходе подключённый файл, но без обработки компилятором. Такая опция может пригодиться при подключении CSS-файла, в котором присутствуют конструкции, которые в Less необходимо преобразовывать.

Опция (once) и (multiple)

Они представляют собой абсолютную дуальную пару, то есть они полностью противоположны по значению.

Ключевое слово (once) запрещает многократное подключение файла с таким именем. Эта опция включена в Less изначально, и прописывать её в директиве `@import` не нужно.

Ключевое слово (multiple) разрешает многократное подключение файла с таким именем.

Опция (optional)

Эта опция позволяет продолжать компиляцию, если подключаемый файл не найден. Если не использовать это ключевое слово, то при отсутствии файла компилятор будет бросаться в вас ошибкой `FileError`.

```
@import "import/_duckduckgo";  
@import (optional) "import/_yahoo";  
@import (optional) "import/_yandex";  
  
.canonium {  
    color: #53599a;  
}
```

```
.duckduckgo {  
    color: #de5833;  
}  
  
.yandex {  
    color: #ffcc00;  
}  
  
.canonium {  
    color: #53599a;  
}
```