Примеси

Ещё одна интересная возможность Less — примеси. Мы можем «примешивать» содержимое одного CSS-правила в другое. Для этого нужно написать имя «примешиваемого» правила внутри другого правила. Рассмотрим пример:

```
Less.white { color: white; } /* объявление примеси */

.text { .white; }    /* применение примеси */
```

Этот Less-код скомпилируется в такой CSS:

```
CSS.white { color: white; }

.text { color: white; }
```

Как мы видим, в правиле, где была «вызвана» примесь `.white`, появилось её содержимое.

Чтобы не выводить саму примесь в CSS, нужно после объявления примеси поставить скобки:

```
Less.white() { color: white; } /* объявление невыводимой примеси */

.text { .white; }      /* применение примеси */

CSS.text { color: white; }
```

При применении примеси скобки писать необязательно.

```
Less/* Эти выражения дают один и тот же результат: */

.mixin();

.mixin;
```

Несколько примесей

К одному правилу можно применить несколько примесей одновременно. В таком случае примеси просто «вызываются» по очереди.

Например:

```
Less.big() {

  width: 100500px;

}



.white() {
```

```
  color: white;
}


.block {
  .big();
  .white();
}
```

```
CSS.block {
  width: 100500px;
  color: white;
}
```

Примесь с параметром

В примесь можно передавать параметры. Они указываются внутри скобок объявления примесей. Названия параметров начинаются с символа `@`. Рассмотрим пример:

```
Less.margin(@value) {
  margin-top: @value;
  margin-bottom: @value;
}


.block {
  .margin(10px);
}
```

```
CSS.block {
  margin-top: 10px;
  margin-bottom: 10px;
}
```

Параметры позволяют сделать примеси более универсальными.

Параметризованные примеси удобны, когда над разными элементами нужно провести однотипные действия с отличающимися результатами.

Значение параметра примеси по умолчанию

Параметризованные примеси можно сделать ещё универсальнее.

Параметру примеси можно задать значение по умолчанию. Оно указывается через двоеточие после названия параметра. Если в примесь при применении параметр не передаётся, то используется значение по умолчанию.

Рассмотрим пример, в котором значение параметра по умолчанию указано, но не задействуется, так как в примесь передаётся явный параметр.

```
.big(@size: 100500px) {

  width: @size;

}



.block {

  .big(10px);

}

.block {

  width: 10px;

}
```

В следующем примере примесь применяется без параметров, поэтому используется значение по умолчанию:

```
Less.big(@size: 100500px) {

  width: @size;

}



.block {

  .big();

}

CSS.block {
```

```
  width: 100500px;
}
```

Примесь с несколькими параметрами

В примесь можно передавать несколько параметров. Параметры перечисляются через запятую `,` или через точку с запятой `;`. Рекомендуется использовать точку с запятой.

Пример:

```
Less.offset(@padding; @margin) {
  padding: @padding;
  margin: @margin;
}

.block {
  .offset(5px; 10px);
}
CSS.block {
  padding: 5px;
  margin: 10px;
}
```

Шаблоны примесей

Иногда бывает полезным изменить поведение примеси в зависимости от передаваемых параметров. Например, у нас есть примесь, задающая размер шрифта:

```
Less.set-font-size(@size) {
  font-size: @size;
}
```

Мы можем создать ещё одну примесь с таким же названием, но передать дополнительно первым параметром название шаблона этой примеси. Имя шаблона указывается первым перед параметрами самой примеси. Добавим примеси дополнительный первый параметр `smaller` и немного изменим принцип её работы:

```
Less.set-font-size(smaller; @size) {

  font-size: @size / 2;

}
```

Теперь можно вызывать эту примесь с названием шаблона и без него и, в зависимости от этого, получать соответствующие результаты:

```
Less.text {

  .set-font-size(100px);

}



.small-text {

  .set-font-size(smaller; 100px);

}

CSS.text {

  font-size: 100px;

}



.small-text {

  font-size: 50px;

}
```

Таким образом можно для схожих действий не создавать несколько примесей с разными названиями. Лучше делать шаблоны одной примеси и просто вызывать её с дополнительным параметром.

Шаблоны примесей, часть 2

Если нужно задать общие свойства для нескольких шаблонов одной и той же примеси, можно создать универсальный шаблон:

```
.font-size(bigger; @size)  {

  font-size: @size * 2;

}
```

```
.font-size(smaller; @size) {
  font-size: @size / 2;
}
.font-size(@_; @size) {
  color: #000000;
} // универсальный шаблон

.content-bigger {
  .font-size(bigger; 20px);
}

.content-smaller {
  .font-size(smaller; 20px);
}
```

Универсальный шаблон применяется вместе с остальными шаблонами:

```
CSS.content-bigger {
  font-size: 40px;
  color: #000000;
}

.content-smaller {
  font-size: 10px;
  color: #000000;
}
```

В качестве имени в универсальный шаблон передаётся специальная переменная `@_`, за ней следуют параметры. Важно, чтобы универсальный шаблон принимал те же параметры, что и все остальные шаблоны.

Примесь с условием

В примесях можно использовать полноценные условия, которые могут изменять поведение примеси в зависимости от значений входных параметров.

Чтобы создать условие, нужно после названия примеси поставить ключевое слово `when`, за которым в скобках написать условную конструкцию. Пример:

```
Less.mixin(@variable) when (@variable = 1) {

  // сделать что-то

}
```

Такая примесь применится, если «вызвать» её с параметром `1`. В противном случае примесь не применится.

```
Less.some-class {

  .mixin(1);

}
```

В условной конструкции допускаются следующие операторы: `>`, `>=`, `=`, `=<`, `<`. Также допустимо использовать встроенные функции Less, которые возвращают конкретные значения.

К примеру, в Less есть встроенная функция `lightness`, которая принимает в качестве параметра значение цвета и возвращает степень его светлоты. Чёрный цвет обладает `0%` светлоты, а белый — `100%`. Вот пример её использования:

```
.mixin(@color) when (lightness(@color) > 50%) {

  // сделать что-то, когда цвет светлее серого

}



.mixin(@color) when (lightness(@color) = 100%) {

  // сделать что-то, когда цвет полностью белый

}
```

Условия и внешние переменные

Условия в примесях могут работать не только с параметрами, с которыми «вызвана» примесь, но и с переменными, объявленными вне примесей. Например, если создать примесь с условием, но без параметров:

```
Less.text-color() when (@theme = light) {
```

```
  color: white;
}
```

А потом создать внешнюю переменную и вызвать где-то примесь:

```less
@theme: light;

.content {
  .text-color();
}
```

То условие выполнится, созданная примесь отработает:

```css
.content {
  color: white;
}
```

То есть можно управлять условиями примесей с помощью внешних переменных.

Условия и типы параметров

В Less есть встроенные функции для проверки типа значения. Их можно применять в условиях примесей для проверки типа переданного параметра. Пример:

```less
// проверка: значение — цвет
.mixin(@param) when (iscolor(@param)) { … }

// проверка: значение — число
.mixin(@param) when (isnumber(@param)) { … }

// проверка: значение — строка
.mixin(@param) when (isstring(@param)) { … }

// проверка: значение — ключевое слово
.mixin(@param) when (iskeyword(@param)) { … }
```

```
// проверка: значение — url

.mixin(@param) when (isurl(@param)) { … }
```

Все эти функции возвращают `true`, если переданный в них параметр соответствует проверяемому типу.

Таким образом можно создать универсальную примесь, которая в зависимости от типа переданных параметров будет работать по-разному.

Переменные-вставки

Переменные можно использовать не только в *значениях* CSS-свойств, но и как составные части селекторов, названий свойств или как «кусочки» значений свойств. С помощью такой *подстановки переменных*, или *Variable Interpolation*, можно динамически формировать разные части CSS-правил.

Чтобы сделать подстановку значения переменной, нужно использовать фигурные скобки вокруг её имени:

```
@state: success;

@property: color;

@icon: "question";

@pixels: 2;



.btn-@{state} {

  background-color: green;

}

.btn-error {

  background-@{property}: red;

}

.btn-help {

  background-image: url("/img/icons/@{icon}.png");

}

.btn-info {

  border: ~"@{pixels}px" solid blue;

}
```

Из примеров выше скомпилируется такой CSS:

```
CSS.btn-success {
  background-color: green;
}

.btn-error {
  background-color: red;
}
.btn-help {
  background-image: url("/img/icons/question.png");
}
.btn-info {
  border: 2px solid blue;
}
```

Кстати, тильда `~` в примере выше нужна для хитрого механизма экранирования Less. Без неё «склеивание» переменной и единицы измерения не сработает.

С помощью «переменных-вставок» можно формировать имена селекторов динамически в зависимости от определённых условий или в цикле.

Цикл

В Less нет специального синтаксиса для циклов. Но есть возможность вызывать примеси внутри самих себя. Так с помощью рекурсии и реализуются циклы. Рассмотрим пример:

```
Less.mixin(@n) {
  .mixin(@n + 1);
}

.mixin(1);
```

В примере создаётся «бесконечный» цикл с увеличивающейся переменной-счётчиком, который никогда не закончится. Чтобы рекурсия всё-таки когда-нибудь прекращалась, к примеси добавляется условие выполнения:

```
Less.mixin(@n) when (@n > 0) {
```

```
   .mixin(@n - 1);

}



.mixin(2);
```

Теперь цикл выполнится два раза, сработает условие выполнения примеси и произойдёт выход из рекурсии.

Для чего можно применять циклы? Например, для генерирования целых CSS-правил. Если в цикле в имени селектора использовать переменную-вставку то можно на выходе получить набор правил с разными селекторами. В примере ниже цикл исполняется три раза, в каждой итерации создастся правило с переменной-счётчиком `@n` в качестве суффикса селектора:

```
.mixin(@n) when (@n > 0) {

  .text-@{n} {

  }



  .mixin(@n - 1);

}



.mixin(3);

CSS.text-3 {}

.text-2 {}

.text-1 {}
```

Внутри «цикла» переменную-счётчик можно использовать не только в условиях или в селекторах, но и в значениях свойств.