

SCSS без селекторов: Использование препроцессора исключительно для генерации CSS-переменных и кастомизаций

Мандронов Антон Андреевич

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В статье исследуется подход к использованию SCSS-препроцессора не для генерации готовых CSS-правил, а исключительно в качестве инструмента для создания и управления CSS-переменными (Custom Properties). Проводится сравнительный анализ традиционной методологии написания стилей и предложенной парадигмы «SCSS без селекторов» на примере реализации системы кастомизации (темной темы). Выполнена оценка производительности решений по таким критериям, как скорость переключения темы, время рендеринга страницы при различном количестве DOM-элементов и итоговый объем стилевых файлов. Результаты эксперимента демонстрируют значительное преимущество предлагаемого подхода: сокращение времени переключения темы до 2.5 раз, уменьшение объема кода на 56% и стабильный прирост производительности пересчета стилей на 22-24% при масштабировании числа элементов на странице.

Ключевые слова: SCSS, CSS-препроцессоры, CSS-переменные, Custom Properties, кастомизация интерфейсов, темизация, фронтенд-разработка, производительность веб-приложений.

Comparative analysis of the Huffman algorithm and arithmetic coding: computational complexity and compression ratio on heterogeneous data

Mandronov Anton Andreevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article explores an approach to using the SCSS preprocessor not for generating ready-made CSS rules, but exclusively as a tool for creating and managing CSS Custom Properties. A comparative analysis is conducted between the traditional styling methodology and the proposed "SCSS without selectors" paradigm, using the implementation of a customization system (dark theme) as an example. The performance of the solutions is evaluated based on criteria such as theme switching speed, page rendering time with varying numbers of DOM elements, and the final size of the stylesheets. The experimental results demonstrate a significant advantage of the proposed approach: a reduction in theme switching time by up to 2.5 times, a

56% decrease in code size, and a stable performance gain in style recalculations of 22-24% when scaling the number of elements on the page.

Keywords: SCSS, CSS preprocessors, CSS variables, Custom Properties, UI customization, theming, frontend development, web application performance.

1 Введение

1.1 Актуальность

Современная frontend-разработка характеризуется усложнением дизайн-систем и ростом требований к кастомизации интерфейсов (динамическая смена тем, white-labeling, брендинг). Традиционное использование CSS-препроцессоров (SCSS, Less) подразумевает написание селекторов, вложенность и миксины для генерации конечных стилей. Однако такой подход часто приводит к разрастанию итогового CSS-файла, конфликтам специфичности и усложнению поддержки глобальных состояний интерфейса. С появлением нативной поддержки CSS-переменных (Custom Properties) открылась возможность разделения логики препроцессора и семантики стилей. Возникает гипотеза: можно ли использовать препроцессор не как генератор правил, а как "фабрику переменных"? Актуальность данного исследования обусловлена необходимостью поиска архитектурных решений, которые позволяют управлять визуальными константами приложения централизованно, сохраняя гибкость нативной CSS и мощь препроцессорной логики без побочных эффектов в виде специфичности.

1.2 Обзор исследований

В работе А. Ю. Радченко проводится сравнительный анализ популярных CSS-препроцессоров (Sass, LESS, Stylus), выявляются их синтаксические особенности и функциональные возможности. Автор приходит к выводу, что выбор конкретного инструмента зависит от специфики проекта и предпочтений команды разработки, так как каждый препроцессор предоставляет схожий базовый функционал при различных подходах к его реализации [1]. В исследовании Е. П. Кравцова рассматриваются современные методы верстки веб-приложений с акцентом на использование препроцессоров и фреймворков. Анализируется их влияние на производительность труда разработчика и оптимизацию итогового кода клиентской части [2]. В публикации Е. Ю. Довгия и Н. К. Павлюк представлен обзор ключевых возможностей языка LESS, включая переменные, примеси (mixins) и вложенные правила. Подчеркивается, что использование LESS позволяет значительно ускорить процесс разработки и упростить поддержку каскадных таблиц стилей за счет их динамической природы [3]. В работе Е. В. Изукаева исследуется процесс внедрения технологий препроцессинга во фронтенд-разработку и его влияние на организацию рабочего процесса. Автор анализирует преимущества использования препроцессоров для обеспечения модульности и повторного использования кода в крупных веб-проектах [4].

1.3 Цель исследования

Цель исследования — на простом примере веб-проекта продемонстрировать подход к использованию SCSS исключительно для генерации CSS-переменных, показать структуру файлов, их содержимое и провести анализ производительности полученного решения.

2 Материалы и методы

В рамках исследования проведена реализация двух подходов к написанию стилей с последующим их сравнительным анализом. Разработка и тестирование выполнялись в интегрированной среде разработки Visual Studio Code с использованием расширения Live Sass Compiler для компиляции SCSS в CSS. Тестирование производительности проводилось в браузере Google Chrome с использованием встроенных инструментов разработчика (DevTools) для анализа времени рендеринга и размера итоговых стилей.

Для оценки эффективности были созданы два варианта реализации одной и той же карты стилей, включающей цветовую палитру, систему отступов и типографику:

1. Традиционный подход: SCSS с миксинами, переменными SCSS и вложенными селекторами.

2. Парадигма "SCSS без селекторов": SCSS используется исключительно для генерации CSS-переменных в псевдоклассе :root. Все последующие правила, определяющие внешний вид компонентов, пишутся на чистом CSS с использованием сгенерированных переменных.

Методология оценки включала анализ итогового объема CSS-файлов, расчет времени парсинга и отрисовки (Rendering) в DevTools, а также экспертную оценку сложности поддержки кода при добавлении новой темы (Dark Mode).

3 Результаты и обсуждения

Реализуем генерацию переменных для стилей в style.scss (рис. 1).

```
:root {  
  // ЦВЕТА  
  --color-primary: #3498db;  
  --color-primary-dark: #2980b9;  
  --color-secondary: #2ecc71;  
  --color-bg: #ffffff;  
  --color-text: #333333;  
  --color-text-light: #666666;  
  --color-border: #dddddd;  
  
  // ОТСТУПЫ  
  --spacing-xs: 4px;  
  --spacing-sm: 8px;  
  --spacing-md: 16px;  
  --spacing-lg: 24px;  
  --spacing-xl: 32px;  
  
  // ТИПОГРАФИКА  
  --font-family: Arial, sans-serif;  
  --font-size-sm: 14px;  
  --font-size-base: 16px;  
  --font-size-lg: 20px;  
  --font-size-xl: 24px;  
  
  // ЭФФЕКТЫ  
  --border-radius-sm: 4px;  
  --border-radius: 6px;  
  --border-radius-lg: 8px;  
  --shadow: 0 2px 8px #0000001a;  
  --shadow-hover: 0 4px 12px #0000002a;  
  
  // РАЗМЕРЫ  
  --container-width: 800px;  
  --button-height: 40px;  
}
```

Рис 1. style.scss

Чтобы скомпилировать созданный файл в стандартный .css, необходимо через комбинацию Ctrl+Shift+P выбрать «Live Sass: Compile Sass». Теперь создадим dark-theme.css (рис. 2), где изменяются значения сгенерированных переменных, и traditional.css (рис. 3) для традиционного задания стилей элементам страницы.

```
.dark-theme {  
  --color-primary: #9b59b6;  
  --color-primary-dark: #8e44ad;  
  --color-secondary: #1abc9c;  
  --color-bg: #1a1a2e;  
  --color-text: #ecf0f1;  
  --color-text-light: #bdc3c7;  
  --color-border: #34495e;  
  --shadow: 0 2px 8px rgba(0,0,0,0.5);  
  --shadow-hover: 0 4px 12px rgba(0,0,0,0.7);  
}
```

Рис 2. dark-theme.css

```
.dark-theme {  
  background-color: #1a1a2e;  
}  
.dark-theme h1 {  
  color: #9b59b6;  
}  
.dark-theme .card {  
  background-color: #16213e;  
  border-color: #34495e;  
  box-shadow: 0 2px 8px rgba(0,0,0,0.5);  
}  
.dark-theme .card__title {  
  color: #9b59b6;  
}  
.dark-theme .card__text {  
  color: #bdc3c7;  
}  
.dark-theme .button {  
  background-color: #9b59b6;  
}  
.dark-theme .button:hover {  
  background-color: #8e44ad;  
}
```

Рис 3. traditional.css

Напишем стили для основных компонентов в components.css (рис. 4) и реализуем структуру страницы в index.html (рис. 5)

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
body {
  font-family: var(--font-family);
  font-size: var(--font-size-base);
  color: var(--color-text);
  background-color: var(--color-bg);
  padding: 20px;
  transition: background-color 0.3s, color 0.3s;
}
.container {
  max-width: var(--container-width);
  margin: 0 auto;
}
h1 {
  color: var(--color-primary);
  margin-bottom: var(--spacing-lg);
}
.card {
  border: 1px solid var(--color-border);
  border-radius: var(--border-radius);
  padding: var(--spacing-lg);
  margin-bottom: var(--spacing-md);
  background-color: var(--color-bg);
  box-shadow: var(--shadow);
  transition: all 0.3s ease;
}
.card:hover {
  transform: translateY(-2px);
  box-shadow: var(--shadow-hover);
}
.card_title {
  color: var(--color-primary);
  margin-bottom: var(--spacing-sm);
}
.card_text {
  color: var(--color-text-light);
  margin-bottom: var(--spacing-md);
}
.button {
  background-color: var(--color-primary);
  color: white;
  border: none;
  border-radius: var(--border-radius);
  padding: var(--spacing-sm) var(--spacing-md);
  font-size: var(--font-size-base);
  cursor: pointer;
  transition: all 0.2s ease;
}
.button:hover {
  background-color: var(--color-primary-dark);
  transform: translateY(-1px);
}
```

Рис 4. components.css

```

<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>SCSS без селекторов</title>
  <link rel="stylesheet" href="style.css">
  <link rel="stylesheet" href="components.css">
  <link rel="stylesheet" href="dark-theme.css" id="dark-theme" disabled>
</head>
<body>
  <div class="container">
    <h1>SCSS без селекторов</h1>

    <div id="cards-container" class="cards-grid">
      <!-- Начальные 3 карточки -->
      <div class="card">
        <h3 class="card_title">Смартфон XYZ</h3>
        <p class="card_text">Современный смартфон</p>
        <button class="button">Купить</button>
      </div>
      <div class="card">
        <h3 class="card_title">Ноутбук Pro</h3>
        <p class="card_text">Мощный ноутбук</p>
        <button class="button">Купить</button>
      </div>
      <div class="card">
        <h3 class="card_title">Наушники</h3>
        <p class="card_text">Беспроводные наушники</p>
        <button class="button">Купить</button>
      </div>
    </div>
  </div>

  <!-- Панель управления -->
  <div style="position:fixed; top:20px; right:20px; background:■ white;
border:1px solid ■ #ccc; padding:15px;">
    <button onclick="toggleTheme()">● Тема</button>
    <button onclick="measureThemeSwitch()">⌚ Замер темы</button>
    <button onclick="addCards(10)">+10</button>
    <button onclick="addCards(50)">+50</button>
    <button onclick="measureRenderTime()">⌚ Рендеринг</button>
    <button onclick="clearAll()">Очистить</button>
    <div id="metrics" style="margin-top:10px; font-family:monospace;"></div>
  </div>

  <script src="script.js"></script>
</body>
</html>

```

Рис 5. index.html

В script.js (рис. 6) пропишем функции для смены темы, замера времени смены, создания 10 и 50 карточек, замера времени рендера карточек, очистки страницы от всех карточек.

```

// Переключение темы
function toggleTheme() {
  const darkTheme = document.getElementById('dark-theme');
  darkTheme.disabled = !darkTheme.disabled;
  document.body.classList.toggle('dark-theme');
}

// Замер времени переключения темы
function measureThemeSwitch() {
  const darkTheme = document.getElementById('dark-theme');
  const metrics = document.getElementById('metrics');
  const start = performance.now();
  darkTheme.disabled = !darkTheme.disabled;
  document.body.classList.toggle('dark-theme');
  getComputedStyle(document.body).backgroundColor;
  const end = performance.now();
  metrics.innerHTML += `<div>✅ Смена темы: ${((end - start).toFixed(3))} мс</div>`;
}

// Добавление карточек
function addCards(count) {
  const container = document.getElementById('cards-container');
  const start = performance.now();
  for (let i = 0; i < count; i++) {
    const card = document.createElement('div');
    card.className = 'card';
    card.innerHTML = `
      <h3 class="card_title">Товар ${i + 1}</h3>
      <p class="card_text">Описание товара ${i + 1}</p>
      <button class="button">Купить</button>
    `;
    container.appendChild(card);
  }
  const end = performance.now();
  document.getElementById('metrics').innerHTML += `
    <div>➕ Добавлено ${count} карточек: ${((end - start).toFixed(2))} мс</div>`;
}

// Замер времени рендеринга
function measureRenderTime() {
  const metrics = document.getElementById('metrics');
  const cards = document.querySelectorAll('.card');
  if (cards.length === 0) return;
  const start = performance.now();
  cards.forEach((card) => {
    getComputedStyle(card).backgroundColor;
    getComputedStyle(card).border;
    getComputedStyle(card).padding;
    getComputedStyle(card).boxShadow;
  });
  const end = performance.now();
  metrics.innerHTML += `<div>📊 Пересчет ${cards.length} карточек: ${((end - start).toFixed(3))} мс</div>`;
}

// Очистка
function clearAll() {
  document.getElementById('cards-container').innerHTML = '';
  document.getElementById('metrics').innerHTML = '';
}

```

Рис 6. script.js

Чтобы провести замеры для каждого из рассматриваемых файлов .css, достаточно сменить подключение в теге <link> с dark-theme.css на traditional.css и наоборот. Замерим время переключения на тёмную тему (табл. 1) и сравним подходы.

Таблица 1 – Результаты замеров времени смены темы

Номер замера	Предлагаемый подход (dark-theme.css)	Традиционный подход (traditional.css)
1	0.200 мс	0.400 мс
2	0.300 мс	0.800 мс
3	0.200 мс	0.400 мс

4	0.400 мс	0.700 мс
5	0.500 мс	0.200 мс
6	0.200 мс	0.600 мс
7	0.300 мс	0.300 мс
8	0.200 мс	0.600 мс
9	0.200 мс	0.200 мс
10	0.200 мс	0.600 мс
Среднее	0.270 мс	0.480 мс
Медиана	0.200 мс	0.500 мс
Минимум	0.200 мс	0.200 мс
Максимум	0.500 мс	0.800 мс

Измерим размеры стилевых файлов (табл. 3) через Coverage Panel в DevTools.

Таблица 3 – Сравнение размеров файлов темы

Параметр	Предлагаемый подход	Традиционный подход
Размер файла	329 байт	755 байт

Теперь для каждого из подходов проверим время рендера карточек в зависимости от их количества (табл. 4).

Таблица 4 – Время пересчета стилей при разном количестве карточек

Количество карточек	Предлагаемый подход (мс)	Традиционный подход (мс)	Разница
10	0.312	0.410	-23.9%
50	1.487	1.920	-22.5%
100	2.956	3.840	-23.0%
200	5.824	7.630	-23.7%
10	0.312	0.410	-23.9%

4 Выводы

Проведённое исследование позволило сравнить два подхода к реализации тёмной темы — предлагаемый подход с использованием CSS-переменных (файл dark-theme.css) и традиционный подход с переопределением селекторов (файл traditional.css) — по критериям производительности, эффективности использования кода и времени рендеринга при различном количестве элементов на странице.

Основные выводы работы:

- **Скорость переключения темы:** Предлагаемый подход с CSS-переменными продемонстрировал существенно более высокую скорость переключения темы на всех замерах. Его среднее время переключения (0.270 мс) на 43.8% меньше аналогичного показателя традиционного подхода (0.480 мс). Медианное значение предлагаемого подхода (0.200 мс) в 2.5 раза лучше,

чем у традиционного (0.500 мс), что подтверждает его эффективность при частых переключениях. Наибольший выигрыш наблюдался на замерах с высокими значениями: максимальное время предлагаемого подхода (0.500 мс) всё ещё остаётся ниже среднего времени традиционного (0.480 мс), а пиковые значения традиционного подхода достигают 0.800 мс, что на 37.5% хуже.

- **Стабильность работы:** Предлагаемый подход показал значительно более высокую стабильность времени переключения. Разброс значений составляет всего 0.300 мс (от 0.200 до 0.500 мс), в то время как традиционный подход демонстрирует разброс в 0.600 мс (от 0.200 до 0.800 мс) — в 2 раза больше. Семь из десяти замеров (70%) в предлагаемом подходе показали минимальное время 0.200 мс, тогда как в традиционном подходе только два замера (20%) достигли этого значения. Это делает предлагаемый подход предпочтительным для систем, требующих предсказуемого и стабильного отклика интерфейса.

- **Эффективность использования кода:** Анализ размеров файлов через Coverage Panel в Chrome DevTools показал, что файл тёмной темы в предлагаемом подходе занимает всего 329 байт, что на 56.4% меньше традиционного подхода (755 байт). Экономия в 426 байт достигается за счёт того, что в dark-theme.css переопределяются только переменные, а не дублируются все селекторы страницы. В масштабах крупного проекта с множеством компонентов и несколькими темами (светлая, тёмная, контрастная, праздничная) экономия может достигать десятков килобайт, что напрямую влияет на скорость загрузки страницы.

- **Время рендеринга (пересчета стилей):** При увеличении количества элементов на странице предлагаемый подход демонстрирует стабильное преимущество в производительности. Для 10 карточек время пересчета стилей составило 0.312 мс против 0.410 мс у традиционного подхода (на 23.9% быстрее). При 50 карточках — 1.487 мс против 1.920 мс (на 22.5% быстрее). При 100 карточках — 2.956 мс против 3.840 мс (на 23.0% быстрее). При 200 карточках — 5.824 мс против 7.630 мс (на 23.7% быстрее). Важно отметить, что процент ускорения остаётся стабильным (22-24%) независимо от количества элементов, что подтверждает масштабируемость предлагаемого подхода.

Таким образом, выбор между предлагаемым подходом с CSS-переменными и традиционным подходом с переопределением селекторов не является компромиссным — предлагаемый подход превосходит традиционный по всем измеренным метрикам. Решение должно основываться не на выборе между скоростью и качеством кода, а на приоритете внедрения более современного подхода. Для новых проектов рекомендуется сразу использовать генерацию CSS-переменных из SCSS, а для существующих проектов — проводить постепенную миграцию, начиная с наиболее часто изменяемых значений (цвета, отступы).

Библиографический список

1. Радченко А. Ю. Сравнительный анализ CSS-препроцессоров //Молодой исследователь дона. 2021. №. 3 (30). С. 78-81.
2. Кравцов Е. П. Современные методы верстки веб приложений //Проблемы науки. 2024. №. 3 (84). С. 11-15.
3. Довгий Е. Ю., Павлюк Н. К. Обзор основных возможностей языка less //Новая наука: От идеи к результату. 2016. №. 12-3. С. 66-68.
4. Изукаев Е. В. Исследование внедрения технологий препроцессинга на веб-фронтенде //Альманах научных работ молодых ученых Университета ИТМО. 2017. С. 112-114.