Организация кода для CSS препроцессоров

CSS препроцессоры, возможно, одни из самых полезных инструментов во фронтенде. Неважно выбрали ли вы Sass, LESS, или какой-то другой препроцессор, все они прекрасны, потому что добавляют множество новых возможностей, которых никогда не было в CSS. Но я не пытаюсь убедить вас использовать препроцессоры, я полагаю вы уже это делаете.

Вместо этого, мы исследуем несколько различных вариантов архитектуры, которые можно использовать с вашим любимым препроцессором, чтобы найди подходящую именно вам. Хотя во всех примерах используются Sass файлы (т.е. `.scss`) они легко адаптируются под любой препроцессор.

Архитектура?

Одна из самых привлекательных возможностей CSS препроцессоров — подключение файлов. Это позволяет нам разделить код на несколько частей, которые впоследствии будут объединены препроцессором. Таким образом, можно работать с отдельными файлами и не беспокоиться о длинном списке HTTP запросов.

Сперва вы будете сомневаться в полезности данной возможности. Но лишь до тех пор, пока не начнете работать с большим проектом. Именно тогда ваш файл со стилями превратится в неаппетитное месиво. В моем случае произошло именно так, и я остро нуждался в структуризации кода.

Итак, что же мы имеем? CSS с некоторой натяжкой можно назвать языком программирования, но мы не можем просто взять и скопировать свойственные ему архитектурные шаблоны. Как же нам тогда организовать различные части стилей проекты? Пакуйте чемоданы, впрыгивайте в ботинки и добро пожаловать в чудный новый мир, который я для вас открою (в смысле — просто расслабьтесь и позвольте мне сделать всю тяжелую исследовательскую работу за вас).

Создание архитектуры я всегда буду начинать с папки base. В ней будут файлы, которые не следует трогать (например, reset или стили библиотек).

Функциональное распределение

Это структура, которая рано или поздно приходит в голову, если вы новичок в препроцессорах. Распределить функциональность на отдельные файлы и вправду кажется очень логичным. Один файл для всех переменных, один

для примесей и т.д. и, наконец, один для текущих стилей. Давайте добавим еще `normalize.scss` ради реализма.

- `/base`
- `_mixins.scss`
- `_variables.scss`
- `screen.scss`

Достоинства

- Красивый список примесей и переменных

Недостатки

- Все стили в одном файле

Вывод

Плохо. В итоге мы получили огромный файл стилей. С тем же успехом мы могли вообще не использовать препроцессор. Хотя, результат успешно сочетается с другими архитектурами. По факту, именно это нам и нужно. Но разве только это? Не лучшая идея.

Распределение «Катана»

Ну, а если, скажем, разделить страницы на части и обозначить стили для каждой части индивидуально?

- `/base`
- `/sections`
  - `_header.scss`
  - `_content.scss`
  - `_footer.scss`
  - `_sidebar.scss`
  - `_modals.scss`
- `_mixins.scss`
- `_variables.scss`
- `screen.scss`

Достоинства

- Имеет смысл
- Вряд ли будут повторяющиеся селекторы (отдельные блоки правил для одного и того же селектора)

Недостатки

- Можно запутаться, особенно если у вас много форм представления
- Файлы стилей могут стать довольно объемными, в частности `_content.scss`

Вывод

К сожалению, с помощью подобной структуры, вы мотивируете себя использовать стили исключительно в определенных областях страниц, и в результате получаете статичный CSS и множество повторяющихся свойств (например `border-radius`). Вы можете частично компенсировать этот недостаток используя примеси и переменные. Тем не менее, данный подход может подходит только для малых и средних проектов.

Шаблонное или страничное распределение

Если описывать отдельные файлы стилей для каждого шаблона или страницы, то легко найти какой-либо объект, особенно если он носит похожее название.

- `/base`
- `/templates`
    - `_category.scss`
    - `_footer.scss`
    - `_header.scss`
    - `_index.scss`
    - `_page.scss`
    - `_single.scss`
- `/pages`
- `_mixins.scss`
- `_variables.scss`
- `screen.scss`

Вы можете заметить, что имена шаблонов выглядят очень похожими на имена шаблонов WordPress. Это сделано специально.

Достоинства

- Хорошее распределение
- Легкий поиск
- Кастомизация каждого шаблона

Недостатки

- Мы опять слишком сильно ушли в детализацию

Вывод

Это еще одно возможное решение для отдельных проектов. На самом деле, эта и предыдущая архитектуры образуют прекрасную пару, которая может неплохо работать. Вариант подходит для многих средних и крупных проектов.

В терминах веб-дизайна

Я слышал, как Крис Койер (Chris Coyier) описывал эту архитектуру в ролике «Рабочий процесс современного веб-дизайнера». Как и предполагает название, она предназначена для веб-дизайнеров. То есть комфортно ей

воспользоваться смогут только люди знакомые как с дизайном, так и с профессиональным сленгом.

Чего Крис не сделал, так это не использовал папки, а они все же могут понадобиться.

- `_normalize.scss`
- `_buttons.scss`
- `_footer.scss`
- `_grid.scss`
- `_header.scss`
- `_icons.scss`
- `_navigation.scss`
- `_typography.scss`
- `screen.scss`

Достоинства

- Имеет смысл (для дизайнеров)

Недостатки

- Пугает не-дизайнеров
- Довольно неряшлива
- Могут быть повторяющиеся селекторы

Вывод

Надо заметить, что список файлов, который показал Крис, был в три раза длиннее приведенного выше. Однако, эта архитектура вполне работоспособна, я нередко ею пользуюсь.

Коктейль «Хьюго»

Чуть ранее в этом году Хьюго (Hugo Giraudel) описал свою Sass архитектуру. Оказалось, это смесь всего того, что мы обсуждали ранее.

- `/base`
  - `_normalize.scss`
  - `_typography.scss`
- `/components`
  - `_buttons.scss`
  - `_navigation.scss`
- `/helpers`
  - `_mixins.scss`
  - `_variables.scss`
- `/layout`
  - `_grid.scss`
  - `_header.scss`
  - `_footer.scss`
- `/pages`
- `/themes`
- `/vendors`
  - `_bootstrap.scss`
  - `_jquery-ui.scss`

- `main.scss`

Достоинства

- Маленькие файлы
- Выглядит организовано

Недостатки

- Слишком много файлов и папок

Вывод

Архитектура хорошо работает на больших проектах. Как я уже заметил ранее, это смесь всего, что мы обсуждали выше. Иногда мне кажется, что данная архитектура несколько перегружена. Но с другой стороны я редко работал над масштабируемыми проектами.

Дополнение от Игоря Зенича

Если вы используете БЭМ-методологию, то логичным будет использовать один из БЭМ-вариантов организации файловой системы:

Обратная БЭМ-иерархия

В раннем БЭМ, когда Яндекс ещё писал код руками, использовался другой вариант, который за неимением официального названия, я называю «обратная БЭМ-иерархия»: исходники раскладываются на папки по технологиям, а внутри них — разделяются на файлы по блокам:

```
scss/
blocks/
  input/                          # Директория блока input
    _input_theme_forest.scss      # Реализация модификатора theme в значении forest в технологии CSS
    _input__clear.scss            # Реализация элемента clear в технологии CSS
    _input.scss                   # Блок input в технологии CSS
  button/                         # Директория блока button
    _button.scss
jade/
  _input.jade
  _button.jade
js/
  blocks/
    input.js                    # Блок input в технологии JavaScript
    button.js
xml/
xsl/
img/
  blocks/
    input/                        # Директория блока input
      input__clear.png          # Реализация элемента clear в технологии PNG
    button/                       # Директория блока button
```

```
      button.png
```

Достоинства

- Легко поддерживать вручную
- Разделяет файлы на отдельные блоки

Недостатки

- Нет четких критериев: когда создавать папку под блок, а когда — просто отдельный файл
- Нельзя использовать bem-tools

Прямая БЭМ-иерархия

Канонический Яндекс-вариант: разбирать вёрстку на блоки, складывая каждый в отдельную папку. В эту же папку добавляются все относящиеся к этому css-блоку изображения и js.

```
blocks/
  input/                        # Директория блока input
    _theme/                     # Директория опционального модификатора theme
      input_theme_forest.css    # Реализация модификатора theme в значении forest в технологии
CSS
    __clear/                    # Директория опционального элемента clear
      input__clear.css          # Реализация элемента clear в технологии CSS
      input__clear.png          # Реализация элемента clear в технологии PNG
    input.css                   # Блок input в технологии CSS
    input.js                    # Блок input в технологии JavaScript
  button/                       # Директория блока button
    button.css
    button.js
    button.png
```

Достоинства

- Легко добавлять и удалять блоки
- Можно использовать bem-tools — инструменты для сборки проекта из блоков

Недостатки

- Если вы, как и большинство разработчиков, не используете полный стек БЭМ-технологий, то поддерживать такую структуру вручную довольно утомительно
- Неудобно использовать популярные инструменты сборки (Grunt, Gulp) и плагины к ним, т.к. они предполагают что файлы сложены по папкам отдельно: css, js, графика, а не разбросаны вперемешку.

Вывод

Прямая БЭМ-иерархия подходит для проектов любого размера, если у вас есть опыт работы с полным стеком БЭМ-технологий, если же нет — о ней стоит задуматься на больших и очень больших проектах, группах проектов с

общим дизайном. Обратная БЭМ-иерархия — удобна на маленьких и средних проектах, где не стоит проблема частого переноса/замены блоков.

Резюмируя

Все это — не более чем примеры различных архитектур. Я все еще нахожусь в поиске «той самой», идеальной, архитектуры, если, конечно, она существует. Но, даже если я её и найду, это в любом случае будет моим субъективным мнением.

Тем не менее, это достаточно важный вопрос, и его стоит тщательно обдумать. Если у вас есть возможность избежать беспорядка, особенно в интерфейсах, когда вы начинаете новый проект, вы просто обязаны это сделать.

Оригинальная статья: Organizing Your CSS Code for PreprocessorsСтатьювычитывали: DEI, FMRobot, MrMoranXP, ihorzenich, SilentImp

Brian Rinaldi
http://remotesynthesis.com/
Twitter: https://twitter.com/remotesynth
GitHub: https://github.com/remotesynth

Руслан Каймаков
Twitter: https://twitter.com/MrMoranXP
GitHub: https://github.com/mrmoranxp