css-modules / **css-modules** Public

Documentation about css-modules

16.5k stars **550** forks

Star Watch

Code Issues 108 Pull requests 13 Actions Projects 2 Wiki ···

master

juanca and TrySound Documentation on `from global` (#270) … on Oct 27, 2017 69

View code

README.md

Releases

No releases published

Packages

No packages published

Contributors 20

+ 9 contributors

CSS Modules

A CSS Module is a CSS file in which all class names and animation names are scoped locally by default. All URLs (`url(...)`) and `@imports` are in module request format (`./xxx` and `../xxx` means relative, `xxx` and `xxx/yyy` means in modules folder, i. e. in `node_modules`).

CSS Modules compile to a low-level interchange format called ICSS or Interoperable CSS, but are written like normal CSS files:

```
/* style.css */
.className {
  color: green;
}
```

When importing the **CSS Module** from a JS Module, it exports an object with all mappings from local names to global names.

```
import styles from "./style.css";
// import { className } from "./style.css";
```

```
element.innerHTML = '<div class="' + styles.className + '">';
```

Naming

For local class names camelCase naming is recommended, but not enforced.

> This is recommended because the common alternative, kebab-casing may cause unexpected behavior when trying to access style.class-name as a dot notation. You can still work around kebab-case with bracket notation (eg. `style['class-name']`) but `style.className` is cleaner.

Exceptions

`:global` switches to global scope for the current selector respective identifier. `:global(.xxx)` respective `@keyframes :global(xxx)` declares the stuff in parenthesis in the global scope.

Similarly, `:local` and `:local(...)` for local scope.

If the selector is switched into global mode, global mode is also activated for the rules. (This allows us to make `animation: abc;` local.)

Example: `.localA :global .global-b .global-c :local(.localD.localE) .global-d`

Composition

It's possible to compose selectors.

```
.className {
  color: green;
  background: red;
}

.otherClassName {
  composes: className;
  color: yellow;
}
```

There can be multiple `composes` rules, but `composes` rules must be before other rules. Extending works only for local-scoped selectors and only if the selector is a single class name. When a class name composes another class name, the **CSS Module** exports both class names for the local class. This can add up to multiple class names.

It's possible to compose multiple classes with `composes: classNameA classNameB;`.

Dependencies

Composing from other files

It's possible to compose class names from other **CSS Modules**.

```
.otherClassName {
  composes: className from "./style.css";
}
```

Note that when composing multiple classes from different files the order of appliance is *undefined*. Make sure to not define different values for the same property in multiple class names from different files when they are composed in a single class.

Note that composing should not form a circular dependency. Elsewise it's *undefined* whether properties of a rule override properties of a composed rule. The module system may emit an error.

Best if classes do a single thing and dependencies are hierarchic.

Composing from global class names

It's possible to compose from **global** class names.

```
.otherClassName {
  composes: globalClassName from global;
}
```

Usage with preprocessors

Preprocessors can make it easy to define a block global or local.

i. e. with less.js

```
:global {
  .global-class-name {
    color: green;
  }
}
```

Why?

modular and **reusable** CSS!

- No more conflicts.
- Explicit dependencies.
- No global scope.

Examples

- css-modules/webpack-demo
- outpunk/postcss-modules-example
- Theming
- css-modules/browserify-demo
- x-team/starting-css-modules

History

- 04/2015: `placeholders` feature in css-loader (webpack) allows local scoped selectors (later renamed to `local scope`) by @sokra
- 05/2015: `postcss-local-scope` enables `local scope` by default (see blog post) by @markdalgleish
- 05/2015: `extends` feature in css-loader allow to compose local or imported class names by @sokra
- 05/2015: First CSS Modules spec document and github organization with @sokra, @markdalgleish and @geelen
- 06/2015: `extends` renamed to `composes`
- 06/2015: PostCSS transformations to transform CSS Modules into an intermediate format (ICSS)
- 06/2015: Spec for ICSS as common implementation format for multiple module systems by @geelen
- 06/2015: Implementation for jspm by @geelen and @guybedford
- 06/2015: Implementation for browserify by @joshwnj, @joshgillies and @markdalgleish
- 06/2015: webpack's css-loader implementation updated to latest spec by @sokra

Implementations

webpack

Webpack's css-loader in module mode replaces every local-scoped identifier with a global unique name (hashed from module name and local identifier by default) and exports the used identifier.

Extending adds the source class name(s) to the exports.

Extending from other modules first imports the other module and then adds the class name(s) to the exports.

Server-side and static websites

PostCSS-Modules allows to use CSS Modules for static builds and the server side with Ruby, PHP or any other language or framework.