Lymnee

« Attributs de données avec… style. »

Sommaire

<details>

<summary>

Les attributs de données ↗ constituent une alternative aux classes Cascading Styles Sheets (Css) pour mettre en forme une page Web.

Les attributs de données ↗ sont flexibles : leur valeur peut contenir n'importe quel caractère ! Aussi est-il loisible de suivre la grammaire Css pour déclarer un couple « sélecteur - valeur ». Par exemple…

```
<html data-ym-font-size="medium">
```

… où « data-ym-font-size » représente un espace de nom suivi du sélecteur.

Les relations Hypertext Markup Language (Html) - Css deviennent plus faciles à concevoir et à maintenir.

Certes, l'outil de validation Css Lint ↗ émet des avertissements libellés avec le titre « Disallow unqualified attribute selectors » et la description « Unqualified attribute selectors are known to be slow » quand lui sont soumis des attributs de données ↗.

Cependant, en dépit d'une légende tenace, les attributs de données ↗ ne sont pas plus lents à interpréter que les classes Css… L'article « How performant are data attributes as selectors » ↗ propose un test, disponible dans le fichier « lymnee.benchmark.html » ↗ afin de comparer les performances des deux sélecteurs : 10 000 itérations montrent que les attributs de données ↗ sont légèrement plus rapides !

Cette proposition de syntaxe, baptisée **« Lymnee »**, est simple, mais pas simpliste : pseudo-éléments ↗, Media queries ↗, sélecteurs contextuels, règles @supports ↗ sont pris en charge grâce à des opérateurs booléens.

Lymnee s'accompagne de deux outils pour générer le code Css : un pa-

sont disponibles en JavaScript : un fichier de visualisation, d'une part, et, d'autre part, un fichier de production : est écrite à la volée la feuille de style.

</summary>

Prolégomènes

Écrivons trois lignes de HTML et trois classes CSS correspondantes…

```
<h1 class="foo">Title</h1>
<h2 class="else">Title</h2>
<p class="bar">Paragraph</p>

.foo {
    color: red;
    font-size: large
}
.else {
    color: blue;
    font-size: large
}
.bar {
    color: blue;
    font-size: medium
}
```

Trois difficultés émergent…

1. Comment nommer les classes ? En utilisant la sémantique grâce à un incommensurable effort d'imagination (songer aux esprits curieux qui, en ligne, ausculteront notre savoir-faire !) ? En recourant à un nom générique et réutilisable ?
2. Comment ordonner les classes ? En les triant par ordre alphabétique au prix d'un travail fastidieux ? À défaut de tri, comment retrouver l'information dans plusieurs centaines, voire milliers de classes ? [Selon CSS Stats ↗, Facebook ↗ utilise 15 982 règles, le 26 août 2020.]

1. Comment appliquer le principe Don’t Repeat Yourself (Dry), c’est-à-dire factoriser le code afin d’obvier à la redondance ?

Avec un tri et un code factorisé (manuellement !), nos trois classes s’établissent ainsi…

```
.bar {
    font-size: medium
}
.bar,
.else {
    color: blue
}
.else,
.foo {
    font-size: large
}
.foo {
    color: red
}
```

ou…

```
.bar,
.else {
    color: blue
}
.foo {
    color: red
}
.bar {
    font-size: medium
}
.else,
.foo {
    font-size: large
}
```

Avec **Lymnee**, nous écrivons…

```
<h1 data-ym-color="red" data-ym-font-size="large">Title</h1>
<h2 data-ym-color="blue" data-ym-font-size="large">Title</h2>
<p data-ym-color="blue" data-ym-font-
size="medium">Paragraph</p>
```

Nous obtenons…

```
[data-ym-color="blue"] {
    color: blue
}
[data-ym-color="red"] {
    color: red
}
[data-ym-font-size="large"] {
    font-size: large
}
[data-ym-font-size="medium"] {
    font-size: medium
}
```

Un codage classique nécessite six sélecteurs et quatre règles. **Lymnee** utilise quatre sélecteurs et quatre règles.

Le code Html, valide ! est verbeux (lu communément par un logiciel…), mais la solution **Lymnee** présente plusieurs avantages…

1. La lecture du code Html évoque le rendu visuel.
2. La quête du nom de classe idéal n'existe plus.
3. Le code Css est factorisé, il n'y a plus de redondance. Les règles sont triées. (Deux différences, et non des moindres, avec les styles en ligne !)
4. La syntaxe est élémentaire. Après ajout d'un préfixe comme « data-ym- », le couple sélecteur - value est utilisé : il suffit de suivre la grammaire de Css.
5. Le code Css superflu devient évident et peut être éliminé sans effort.
6. Seuls sont réinitialisés les styles des balises utilisées, et cette réinitialisation est optionnelle.
7. La facilité de lecture de **Lymnee** par un humain facilite son traitement logique. C'est pourquoi, sont proposées deux librairies, sous forme de paquet Node.js ↗ avec dépendances, d'une part, et, d'autre part, d'un fi-

chier PHP. Ces librairies, paramétrables, analysent tous les fichiers d'un répertoire contenant des fragments au format HTML et extraient les attributs de données ↗ afin de produire le code CSS, compressé ou pas au choix de l'auteur. Deux fichiers JavaScript sont aussi disponible pour mettre à l'épreuve la méthode en quelques instants. Le présent document utilise JavaScript pour générer le code CSS à la volée ; le résultat non compressé est visible dans la console du navigateur : quelques sophistications superflues montrent la polyvalence de Lymnee.
2. Écrire le code CSS à la main demeure envisageable : l'exercice ne réclame pas un effort inouï !

État des lieux

Depuis des temps immémoriaux (!), les auteurs.autrices sont en quête de la structure idéale pour organiser le code CSS.

Les méthodes comme Object-Oriented CSS ↗, BEM — Block Element Modifier ↗, Atomic Design ↗ ajoutent une couche d'abstraction. Comment reconnaître un premier et un deuxième objet, un « bloc » et un « élement », un « organisme » et un « modèle » ?… Collective et individuelle sont les variabilités… Le travail collaboratif, dans l'espace ou dans le temps, devient chaotique. Un.e auteur.autrice (le substantif « développeur.se » constitue un abus de langage, car CSS comme HTML n'utilisent pas la programmation) interprète différemment les règles au fil du temps, voire oublie le raisonnement qui l'a conduit à préférer une « molécule » à un « atome ».

Les préprocesseurs, comme Sass ↗ ou Less ↗, visent notamment à rationaliser le code CSS. Le contenu est souvent divisé en fragments gigognes qui suivent une logique aléatoire. La recherche d'une règle impose de parcourir plusieurs fichiers. Comme la plupart des navigateurs interprète maintenant les variables Css ↗ [Can I use # CSS Variables (Custom Properties) ↗] et utilise la fonction (hélas limitée aux opérations arithmétiques traditionnelles !) Calc ↗ [Can I use calc() as CSS unit value ↗], l'intérêt des préprocesseurs est moindre.

D'Attributes Modules for CSS…

En 2014, Glen Maddern et Ben Schwarz commettent l'article « Attributes Modules for CSS » ↗. Ils y proposent une syntaxe comme…

```
<a am-Button="primary large">Large primary button</a>
<a am-Button>
<a am-Button="info small">
<a am-Button="danger extra-small">
```

(L'exemple est repris de la documentation.)

Il s'agit d'un usage des attributs de données ↗, qui est aussi préconisé par Object-Oriented CSS ↗.

… Via Atomic CSS

Atomic CSS ↗, conçu par Thierry Koblentz, style directement les composants. Cette méthodologie évite de concevoir des classes difficiles à maintenir. Ainsi, le code HTML…

```
<div class="D(b) Va(t) Fz(20px)">Hello World!</div>
```

… est traduit comme suit en CSS…

```
.D(b) {
    display: block
}
.Va(t) {
    vertical-align: top
}
.Fz(20px) {
    font-size: 20px
}
```

(L'exemple est repris de la documentation.)

Cette méthode est captivante ! Atomic CSS ↗ n'utilise pas les styles en ligne. Le code CSS est mis en cache et factorisé. La syntaxe alourdit le poids d'un fichier HTML, mais la compression est performante.

Cependant, Atomic CSS ↗ présente deux écueils…

α. Les classes sont difficiles à mémoriser et à relire. Elles s'appuient sur la syntaxe Emmet ↗, dont les abréviations ne sont pas intuitives. Emmet ↗ n'est pas maintenu à jour : ainsi, comment recourir à Css Grid Layout ↗ ?
β. L'échappement des caractères spéciaux est pénible !

Par exemple, en utilisant les Media queries ↗, le code HTML…

```
<div class="bdrs@large"></div>
```

… est traduit comme suit en CSS…

```
@media only screen and (min-width: 768px) {
    .bdrs\@large{
        border-radius:.5rem
    }
}
```

JavaScript échappe les caractères spéciaux avec la méthode { CSS.escape }.

Autrement dit, les concepts d'Atomic CSS ↗ méritent quelque intérêt, mais sont difficiles à appréhender au quotidien.

… À Lymnee

Or, la philosophie d'Atomic CSS ↗ est applicable aux attributs de données ↗ : les codes HTML et CSS deviennent simples à écrire et à lire ! Par un humain. Par un logiciel.

Les **valeurs** des attributs de données ↗ peuvent contenir n'importe quel caractère : « (…) attributes on HTML elements may have any string value, including the empty string. (…) ».

Attributs de données

Les attributs de données ↗ permettent d'ajouter des informations à un élément HTML.

Par exemple, pour stocker le nombre d'habitants d'une ville sans recou-

à une base de données, peut être écrit… { Paris }. L'information est facile à extraire en JavaScript :
{ console.log(document.querySelector('#paris').dataset.population); }, voire { console.log(document.querySelector('#paris').getAttribute('data-population')); }.

Les attributs de données ↗ doivent être préfixés { data- } pour être valides. Une balise HTML peut en recevoir plusieurs.

Les attributs de données ↗ peuvent aussi servir à styler un élément, comme le rappelle Chris Coyier dans son article « A Complete Guide to Data Attributes » ↗.

Classes

En CSS, chaque sélecteur a une spécificité. La hiérarchie s'organise comme suit en schématisant : balise (priorité 1), classe (priorité 10), identifiant (priorité 100). En cas de conflit, la dernière règle s'applique grâce à la cascade. (La règle « !important » a une priorité de 10 000 ; elle signe un travail bâclé.)

Les attributs de données ↗ ont la même spécificité que les classes.

Afin d'éviter les conflits, une bonne pratique consiste à n'utiliser que des classes pour la mise en forme. Les identifiants sont réservés aux interactions avec JavaScript.

Balises

Les balises servent exclusivement à structurer le contenu.

Souvenir des balbutiements du Web, chaque navigateur attribue aux balises un style par défaut, qui compromet la **sémantique**, d'une part, et, d'autre part, parasite la mise en forme. Aussi convient-il de réinitialiser les styles et d'obtenir le rendu visuel souhaité exclusivement au moyen de classes (ou des attributs de données ↗).

Privé.e des éléments visuels imposés par le navigateur, l'auteur.autrice emploie la **sémantique** à bon escient… Pourquoi recourir à { } pour afficher du gras ? Ce contenu a-t-il une haute impor-

utiliser { } pour obtenir de l'italique ? Ce texte mérite-t-il une emphase ? (Une portion de texte en italique parmi du romain créée une rupture visuelle incongrue comme le soulignement des liens et autres acronymes, d'ailleurs !) Ne vaut-il mieux utiliser un élement générique { } aux fins de décoration ?

Le respect de la **sémantique** favorise le référencement par les moteurs de recherche et la compréhension du contenu par les agents utilisateurs, notamment ceux qui n'offrent pas de restitution visuelle.

Un lien peut servir à la compréhension du contenu par les moteurs de recherche mais ne pas être heureux à signaler au.à la lecteur.lectrice. Cette pratique est néanmoins déconseillée.

Assigner un style à une balise oblige à modifier son apparence dans les cas spécifiques. Par exemple, afin de maintenir le rythme vertical nécessaire à une composition harmonieuse, est attribuée par défaut une marge supérieure aux paragraphes avec { p {margin-top: 1.5 rem} } ; si cette marge devient indésirable, par exemple pour un paragraphe inclus dans un item de liste, l'emploi d'une classe comme { .unwelcome {margin-top: 0} } s'impose. Approche guère rationnelle…

En revanche, il est commode de styler directement (« en dur ») les balises qui reçoivent des pseudo-éléments ↗, par exemple guillemets ou ornements textuels, si leur apparence dans la page ne varie pas.

Plutôt que recourir à une librairie comme Normalize.css ↗ pour citer la plus connue, l'usage de la propriété raccourcie { all: unset } semble préférable.

Ainsi…

> a, abbr, address, area, article, aside, audio, b, base, bdi, bdo, blockquote, body, br, button, canvas, caption, cite, code, col, colgroup, data, datalist, dd, del, details, dfn, dialog, div, dl, dt, em, embed, fieldset, figcaption, figure, footer, form, h1, h2, h3, h4, h5, h6, head, header, hgroup, hr, html, i, iframe, img, input, ins, kbd, label, legend, li, link, main, map, mark, math, menu, menuitem, meta, meter, nav, noscript, object, ol, optgroup, option, output, p, param, picture, pre, progress, q, rb, rp, rt, rtc, ruby, s, samp, script, section, select, slot, small, source, span, strong, style, sub, summary, sup, svg,

> table, tbody, td, template, textarea, tfoot, th, thead, time, title, tr, track, u, ul, var, video, wbr

… ou, en ne ciblant, que les élements visuels…

> a, abbr, address, area, article, aside, audio, b, base, bdi, bdo, blockquote, body, br, button, canvas, caption, cite, code, col, colgroup, data, datalist, dd, del, details, dfn, dialog, div, dl, dt, em, embed, fieldset, figcaption, figure, footer, form, h1, h2, h3, h4, h5, h6, header, hgroup, hr, html, i, iframe, img, input, ins, kbd, label, legend, li, main, map, mark, math, menu, menuitem, meter, nav, object, ol, optgroup, option, output, p, param, picture, pre, progress, q, rb, rp, rt, rtc, ruby, s, samp, section, select, slot, small, source, span, strong, sub, summary, sup, svg, table, tbody, td, template, textarea, tfoot, th, thead, time, tr, track, u, ul, var, video, wbr {
> all: unset
> }

… réinitialise chaque balise.

Un travail rigoureux commande de réinitialiser les seules balises pertinentes : la ressource est rare ! n'en déplaise aux amateurs de vastes librairies Css prêtes à l'emploi.

CSS Default Values Reference ↗ indique les valeurs Css par défaut de chaque sélecteur. La réinitialisation proposée peut paraître déconcertante quand les classes (ou les attributs de données ↗ !) n'existent pas encore.

Le support de la propriété « all » est universel, comme l'indique Can I use CSS unset value ↗.

Cependant, la réinitialisation n'est pas parfaite, par exemple avec les éléments { <details> } et { <summary> } : une recherche devient alors nécessaire afin de déterminer comment les navigateurs gèrent ces balises.

Les outils de **Lymnee** proposent une réinitialisation « intelligente ».

Chaque sélecteur commence par « data-ym- » ou un espace de nom équivalent. La valeur est encadrée par des guillemets.

Un attribut s'écrit { data-ym-selecteur="valeur" }. La valeur reçoit des guillemets doubles de part et d'autre… { "valeur" }. Le sélecteur et la valeur doivent être **conformes** aux spécifications Css ↗.

Connaissant la syntaxe Css…

```
.myclass {
    background-color: yellow;
    display: flex
}
```

… Il est facile d'en déduire par rétroaction les attributs de données ↗ Html…

```
<div data-ym-background-color="yellow" data-ym-
display="flex">Div</div>
```

La syntaxe des attributs de données utilise les opérateurs booléens : { || } (or), { && } (and), { !! } (yes). Les booléens, qui s'écrivent de gauche à droite, sont encadrés d'une espace et suivent cet ordre. L'arobase { @ } spécifie une requête média, comme en Css.

Le code-source de la présente page contient plusieurs dizaines d'exemples, tandis que la console affiche les styles. Les styles sont écrits à la volée.

Attribut simple

```
<p data-ym-color="blue">Paragraph</p>

[data-ym-color="blue"] {
    color: blue
}
```

Une balise peut recevoir un nombre infini d'attributs de données ↗…

```
<p data-ym-color="blue" data-ym-font-
size="medium">Paragraph</p>
```

Le nom d’un attribut n’est pas duplicable : l’opérateur { || } répond à cet usage.

Les pseudo-classes ↗ ne sont pas supportées. En revanche, les pseudo-éléments ↗ sont licites…

```
<p data-ym-content="'•'::before">Paragraph</p>

[data-ym-content="'•'::before"]::before {
        content: '•'
}
```

Le symbole littéral « • » est encadré par des guillemets simples droits « ' » dans la déclaration HTML.

Les Media queries ↗ (« requêtes media ») sont prises en charge.

```
<div data-ym-max-width="1024px@media screen and (min-width:
1024px)">Div</div>

@media screen and (min-width: 1024px) {
   [data-ym-max-width="1024px@media screen and (min-width:
1024px)"] {
        max-width: 1024px
   }
}
```

Les variables Css ↗ sont utilisables…

```
<p data-ym-font-weight="var(--font-weight-bold)">Paragraph</p>

:root {
   --font-weight-bold: 700;
}

[data-ym-font-weight="var(--font-weight-bold)"] {
   font-weight: var(--font-weight-bold)
}
```

Une valeur peut comprendre n’importe quel caractère, hormis le guille-

met « " »…

```
<img data-ym-filter="invert(13%) sepia(92%) saturate(7286%) hue-rotate(0deg) brightness(94%) contrast(117%)">

[data-ym-filter="invert(13%) sepia(92%) saturate(7286%) hue-rotate(0deg) brightness(94%) contrast(117%)"] {
    filter: invert(13%) sepia(92%) saturate(7286%) hue-rotate(0deg) brightness(94%) contrast(117%)
}
```

Dernière illustration…

```
<div data-ym-animation="sweep 1.5s ease-in-out"></div>

[data-ym-animation="sweep 1.5s ease-in-out"] {
    animation: sweep 1.5s ease-in-out
}
```

Attribut avec l'opérateur { || }

Avec le booléen « or » sont appliquées plusieurs valeurs au même attribut. C'est utile notamment pour les Media queries ↗ (« requêtes media »).

Premier exemple…

```
<p data-ym-font-family="sans-serif@media screen || monospace@media print">Paragraph</p>

@media screen {
    [data-ym-font-family="sans-serif@media screen || monospace@media print"] {
        font-family: sans-serif
    }
}
@media print {
    [data-ym-font-family="sans-serif@media screen || monospace@media print"] {
        font-family: monospace
    }
```

```
}
```

Deuxième illustration…

```
<div data-ym-display="flex@media screen and (min-width: 30em) and
(max-width: 50em) || inline@media screen and (min-width:
50em)">Div</div>

@media screen and (min-width: 30em) and (max-width: 50em) {
   [data-ym-display="flex@media screen and (min-width: 30em) and
(max-width: 50em) || inline@media screen and (min-width: 50em)"] {
      display: flex
   }
}
@media screen and (min-width: 50em) {
   [data-ym-display="flex@media screen and (min-width: 30em) and
(max-width: 50em) || inline@media screen and (min-width: 50em)"] {
      display: inline
   }
}
```

Le troisième exemple est plus complexe à appréhender…

```
<div data-ym-display="inline-block::before || block">Div</div>

[data-ym-display="inline-block::before || block"]::before {
   display: inline-block
}
[data-ym-display="inline-block::before || block"] {
   display: block
}
```

Attribut avec l’opérateur { && }

Le booléen « and » permet l’usage d’un sélecteur contextuel.

Première illustration…

```
<div class="ispage">
   <div data-ym-background-color="red && ispage">Div</div>
</div>
```

```
.ispage [data-ym-background-color="red && ispage"] {
   background-color: red
}
```

Deuxième exemple, avec deux sélecteurs contextuels différents…

```
<div class="ishome">
   <div data-ym-font-weight="600@media screen and (min-width: 30em) and (max-width: 50em) && ishome || 700@media screen and (min-width: 50em) && ispage">Div</div>
</div>

<div class="ispage">
   <div data-ym-font-weight="600@media screen and (min-width: 30em) and (max-width: 50em) && ishome || 700@media screen and (min-width: 50em) && ispage">Div</div>
</div>

@media screen and (min-width: 30em) and (max-width: 50em) {
      .ishome [data-ym-font-weight="600@media screen and (min-width: 30em) and (max-width: 50em) && ishome || 700@media screen and (min-width: 50em) && ispage"] {
            font-weight: 600
      }
}
@media screen and (min-width: 50em) {
      .ispage [data-ym-font-weight="600@media screen and (min-width: 30em) and (max-width: 50em) && ishome || 700@media screen and (min-width: 50em) && ispage"] {
            font-weight: 700
      }
}
```

Attribut avec l’opérateur { !! }

Le booléen « yes » permet l’utilisation des règles @supports ↗, sous la forme du nom de la propriété suivi par deux points puis par la valeur…

```
<div data-ym-display="block || grid !! display: grid">Div</div>
```

```
[data-ym-display="block || flex !! display: grid"] {
	display: block
}
@supports (display: flex) {
	[data-ym-display="block || flex !! display: grid"] {
		display: grid
	}
}
```

Ici, s’applique la cascade.

Conjugaison de tous les opérateurs

À titre d’exemple !…

```
<div class="ishome">
   <div data-ym-font-weight="600@media screen and (min-width:
30em) and (max-width: 50em) && ishome !! display: block ||
700@media screen and (min-width: 50em) && ispage !! display:
flex">Div</div>
</div>

<div class="ishome">
   <div data-ym-font-weight="600@media screen and (min-width:
30em) and (max-width: 50em) && ishome !! display: block ||
700@media screen and (min-width: 50em) && ispage !! display:
flex">Div</div>
</div>

@supports (display: block) {
	@media screen and (min-width: 30em) and (max-width: 50em) {
		.ishome [data-ym-font-weight="600@media screen and
(min-width: 30em) and (max-width: 50em) && ishome !! display:
block || 700@media screen and (min-width: 50em) && ispage !!
display: flex"] {
			font-weight: 600
		}
	}
}
@supports (display: flex) {
```

```
@media screen and (min-width: 50em) {
        .ispage [data-ym-font-weight="600@media screen and (min-
width: 30em) and (max-width: 50em) && ishome !! display: block ||
700@media screen and (min-width: 50em) && ispage !! display: flex"]
{
            font-weight: 700
        }
    }
}
```

Le codage manuel du style de ces attributs de données ↗ est facile, mais il peut être automatisé.

Librairies

Deux librairies sont disponibles pour écrire les styles correspondant aux attributs de données ↗ qui respectent la syntaxe **Lymnee** :

a. un paquet Node.js ↗ avec dépendances ;
b. un fichier Php.

Deux librairies sont disponibles en JavaScript :

a. un fichier pour voir les styles produits ;
b. un fichier pour écrire à la volée, en production, une feuille de style.

La simplicité de lecture de **Lymnee** par un humain facilite son traitement logique…

À propos de XPath

… Surtout que le langage de requête XML Path Language ↗ localise les nœuds.

Avec le paquet xpath ↗ dans Node.js ↗, la syntaxe pour identifier ces nœuds s’écrit…

```
var prefixDataAttributes = 'data-ym-';
var nodesYm = xpath.select('//@*[starts-with(name(), "' +
```

```
prefixDataAttributes.slice(0, -1) + '")]', doc);
```

En JavaScript, la syntaxe pour identifier ces nœuds s'écrit…

```
var prefixDataAttributes = 'data-ym-';
var nodesYm = document.evaluate('//@*[starts-with(name(), "' +
prefixDataAttributes.slice(0, -1) + '")]', document, null,
XPathResult.ANY_TYPE, null);
```

La méthode iterateNext() ↗ effectue une itération sur chaque nœud.

En PHP, la syntaxe pour identifier ces nœuds s'écrit…

```
$prefixDataAttributes = 'data-ym-';
$nodesYm = $xpath->query('//@*[starts-with(name(),
"'.substr($prefixDataAttributes, 0, -1).'")]');
```

Les trois prédicats sont identiques, bien évidemment !

Pour identifier toutes les balises en JavaScript, la syntaxe est…

```
var tags = document.evaluate('//*', document, null,
XPathResult.ANY_TYPE, null);
```

Pour identifier uniquement les balises comportant l'attribut « data-ym- », la requête devient…

```
var prefixDataAttributes = 'data-ym-';
var tags = document.evaluate('//*[starts-with(name(@*), "' +
prefixDataAttributes.slice(0, -1) + '")]', document, null,
XPathResult.ANY_TYPE, null);
```

Paquet node.js

Au préalable, le script doit être installé : {npm install -g lymnee.js}. Lors du processus d'installation, les dépendances sont ajoutées.

Le code-source de « lymnee.js » ↗ est disponible.

Le script s'exécute avec la commande {node lymnee.js}. Dans ce cas, elle est utilisée sans paramètre.

Sont recherchés par défaut les fichiers au format HTML dans le répertoire courant et les sous-répertoires. Les attributs de données ↗ sous la forme « data-ym- » sont traités. La feuille de style n'est pas compressée (elle est facile à lire par un humain).

La commande peut invoquer plusieurs paramètres, par exemple {node lymnee.js cssNameFile=style excludeDirectories=/home/admin /web/foo.com/public_html/bar excludeFiles=draft.html includeExtensions=html,php path=current prefixDataAttributes=data-some reset=lymnee uncompress=false}.

Chaque paramètre est facultatif. L'ordre est sans importance.

Paramètres

{cssNameFile}
C'est le nom du fichier au format CSS à enregistrer. Par exemple, « style » produit « style.css ».
La feuille de style est écrite dans le répertoire défini avec le paramètre { path } ou, à défaut, dans le répertoire courant.

{excludeDirectories}
Un sous-répertoire peut être exclu de la recherche en indiquant son chemin relatif. Par exemple « a/b/c » voire, s'il y en a plusieurs, « a/b/c,d/ » ou « a/b/c, d/ ». Dans le paramètre, les virgules peuvent être suivies d'une espace.

{excludeFiles}
Certains fichiers peuvent être exclus de la recherche. Si le fichier est dans le répertoire courant, son nom suivi de son extension est requis. Si le fichier est dans un sous-répertoire, son chemin relatif doit être précisé suivi de son nom et de son extension. Par exemple, « draft.html », voire « draft.html,a/b/c/draft.js » ou « draft.html, a/b/c/draft.js ». Dans le paramètre, les virgules peuvent être suivies d'une espace.

{includeExtensions}
Par défaut sont parcourus les fichiers au format HTML, mais la recherche peut porter sur d'autres fichiers. Par exemple, pour lire les fichiers au format JavaScript ou PHP, est indiqué « js,php » ou « js, php ». Dans le paramètre, les virgules peuvent être suivies d'une espace. Pour

seulement les fichiers JavaScript, deuxième exemple, le paramètre est « js ».

{path}
Avec la valeur « current », sont recherchés les fichiers dans le répertoire courant et ses sous-répertoires. Un chemin absolu permet la recherche dans un autre répertoire et ses sous-répertoires, par exemple « /home /admin/web/foo.com/public_html/project ».

{prefixDataAttributes}
Le préfixe par défaut des attributs de données ↗ est « data-my- ». Cependant, il peut être remplacé dans le code HTML : le préfixe doit débuter par « data- » et être suivi d'un nom arbitraire sous la forme « nom- » : « data-nom- ». Par exemple, si le préfixe est « data-my- », cet espace de nom est déclaré lors de la compilation.

{reset}
Avec la valeur « lymnee » sont réinitialisés les styles par défaut des navigateurs des éléments qui reçoivent l'attribut « data-ym- » ou un équivalent. Avec la valeur « all » sont réinitialisés les styles par défaut des navigateurs de **tous** les éléments visibles. Par exemple, la balise { <head> } n'est pas modifiée. Dans les deux cas, en raison de la cascade, la réinitialisation s'affiche au début de la feuille de style. Si le paramètre est omis ou vaut « none », aucune réinitialisation n'est proposée.

{uncompress}
La valeur « false » produit un fichier au format CSS compressé, c'est-à-dire sans saut de ligne et espace inutile. Il est toujours judicieux d'économiser de la bande passante, même quelques octets, en production !

À l'issue du traitement, la console confirme le succès de l'opération, affiche les règles et indique si le code CSS est valide.

Fichier Php

Le script s'exécute grâce à « lymnee.php » ↗. Il a été développé avec la version 7.4.8. Avant utilisation, le fichier doit recevoir la permission d'écrire, par exemple avec un script sous la forme…

… Ou en modifiant directement la valeur de « chmod » avec le gestionnaire de fichiers ou en tapant la commande « ad hoc ».

Le fichier peut être exécuté sans paramètre.

Sont recherchés par défaut les fichiers au format HTML dans le répertoire courant et les sous-répertoires. Les attributs de données ↗ sous la forme « data-ym- » sont traités. Une feuille de style intitulée par défaut « lymnee.css » est produite dans le même répertoire que « lymnee.php ». Elle n'est pas compressée (elle est facile à lire par un humain).

Le fichier peut invoquer plusieurs paramètres, par exemple
{lymnee.php?cssNameFile=styles…excludeFiles=else.html,sub/foo.js&
includeExtensions=html,js…path=/home/admin/web/foo.com
/public_html/lymnee…prefixDataAttributes=data-some…reset=lymnee…
uncompress=true}.

Chaque paramètre est facultatif. L'ordre est sans importance.

Le premier paramètre est toujours précédé de « ? » : il s'agit d'une requête « GET ». Bien entendu, ce script non sécurisé n'est pas utilisable en production !

Paramètres

{cssNameFile}
C'est le nom du fichier au format CSS à enregistrer. Par exemple, « style » produit « style.css ».

{excludeDirectories}
Un sous-répertoire peut être exclu de la recherche en indiquant son chemin relatif. Par exemple « a/b/c » voire, s'il y en a plusieurs, « a/b/c,d/ » ou « a/b/c, d/ ». Dans le paramètre, les virgules peuvent être suivies d'une espace.

{excludeFiles}
Certains fichiers peuvent être exclus de la recherche. Si le fichier est dans le répertoire courant, son nom suivi de son extension est requis. Si le fichier est dans un sous-répertoire, son chemin relatif doit être précisé suivi de son nom et de son extension. Par exemple, « draft.html »,

« draft.html,a/b/c/draft.js » ou « draft.html, a/b/c/draft.js ». Dans le paramètre, les virgules peuvent être suivies d'une espace.

{includeExtensions}
Par défaut sont parcourus les fichiers au format HTML, mais la recherche peut porter sur d'autres fichiers. Par exemple, pour lire les fichiers au format JavaScript ou PHP, est indiqué « js,php » ou « js, php ». Dans le paramètre, les virgules peuvent être suivies d'une espace. Pour parcourir seulement les fichiers JavaScript, deuxième exemple, le paramètre est « js ».

{path}
Avec la valeur « current », sont recherchés les fichiers dans le répertoire courant et ses sous-répertoires. Un chemin absolu permet la recherche dans un autre répertoire et ses sous-répertoires, par exemple « /home /admin/web/foo.com/public_html/project ».

{prefixDataAttributes}
Le préfixe par défaut des attributs de données ↗ est « data-ym- ». Cependant, il peut être remplacé dans le code HTML : le préfixe doit débuter par « data- » et être suivi d'un nom arbitraire sous la forme « nom- » : « data-nom- ». Par exemple, si le préfixe est « data-my- », cet espace de nom est déclaré lors de la compilation.

{reset}
Avec la valeur « lymnee » sont réinitialisés les styles par défaut des navigateurs des éléments qui reçoivent l'attribut « data-ym- » ou un espace de nom équivalent. Avec la valeur « all » sont réinitialisés les styles par défaut des navigateurs de **tous** les éléments visibles. Par exemple, la balise { <head> } n'est pas modifiée. Dans les deux cas, en raison de la cascade, la réinitialisation s'affiche au début de la feuille de style. Si le paramètre est omis ou vaut « none », aucune réinitialisation n'est proposée.

{uncompress}
La valeur « false » produit un fichier au format CSS compressé, c'est-à-dire sans saut de ligne et espace inutile. Il est toujours judicieux d'économiser de la bande passante, même quelques octets, en production !

À l'issue du traitement, la page confirme le succès de l'opération et af-

fiche les règles.

Fichier de visualisation en JavaScript

Le fichier en JavaScript se limite à afficher dans la console du navigateur le code Css des éléments de la page en cours. Pour mémoire, il est impossible avec ce langage côté client de lire le contenu d'une autre page ou d'un répertoire. Cependant, cette solution rudimentaire permet de l'éprouver en quelques secondes.

Deux fichiers sont disponibles :

- « lymnee.vanilla.html » ↗, dont le code Html est à compléter, mais qui inclut le script ;
- « lymnee.vanilla.js » ↗.

Doivent être définies au préalable, au début du script, les variables…

{prefixDataAttributes}
Le préfixe par défaut des attributs de données ↗ est « data-ym- ».
Cependant, il peut être remplacé dans le code Html : le préfixe doit débuter par « data- » et être suivi d'un nom arbitraire sous la forme « nom- » : « data-nom- ». Par exemple, si le préfixe est « data-my- », cet espace de nom doit être déclaré.

{reset}
Avec la valeur « lymnee » sont réinitialisés les styles par défaut des navigateurs des éléments qui reçoivent l'attribut « data-ym- » ou un espace de nom équivalent. Avec la valeur « all » sont réinitialisés les styles par défaut des navigateurs de **tous** les éléments visibles. Par exemple, la balise { <head> } n'est pas modifiée. Dans les deux cas, en raison de la cascade, la réinitialisation s'affiche avant les autres règles. Si le paramètre est omis ou vaut « none », aucune réinitialisation n'est proposée.

{uncompress}
La valeur « false » produit un code Css compressé, c'est-à-dire sans saut de ligne et espace inutile. Il est toujours judicieux d'économiser de la bande passante, même quelques octets, en production !

Fichier de production en JavaScript

Le fichier en JavaScript injecte une feuille de style dans le Document Object Model (Dom) à partir des attributs de données ↗ qui respectent la syntaxe **Lymnee**.

Le code Css, compressé, s'inscrit juste avant la fermeture de la balise { <head> }.

L'analyse puis la génération ne ralentissent pas significativement l'affichage du document.

Cette option permet, à la marge, de tester une règle de style, instantanément, c'est-à-dire à la volée.

Le fichier doit être inclus à la fin de chaque page…

```
<script src="lymnee.live.js"></script>
```

Deux fichiers sont disponibles :

- « lymnee.live.js » ↗ ;
- « lymnee.live.min.js » ↗, qui est compressé.

Doivent être définies au préalable les variables…

{prefixDataAttributes}
Le préfixe par défaut des attributs de données ↗ est « data-ym- ».
Cependant, il peut être remplacé dans le code Html : le préfixe doit débuter par « data- » et être suivi d'un nom arbitraire sous la forme « nom- » : « data-nom- ». Par exemple, si le préfixe est « data-my- », cet espace de nom doit être déclaré.

{reset}
Avec la valeur « lymnee » sont réinitialisés les styles par défaut des navigateurs des éléments qui reçoivent l'attribut « data-ym- » ou un espace de nom équivalent. Avec la valeur « all » sont réinitialisés les styles par défaut des navigateurs de **tous** les éléments visibles. Par exemple, la balise { <head> } n'est pas modifiée. Dans les deux cas, en raison de la cascade, la réinitialisation s'affiche avant les autres règles. Si le para-

mètre est omis ou vaut « none », aucune réinitialisation n'est proposée.

Du code Css peut être ajouté avant et après les styles qui correspondent aux attributs de données ↗ qui respectent la syntaxe **Lymnee**… Les crochets { cssAfter } et { cssBefore } sont disponibles ; leur code est encadré par les caractères { ` } pour permettre l'emploi des guillemets simples et doubles.

L'avantage d'injecter ainsi le code Css est de permettre aux navigateurs dont JavaScript est désactivé de servir le contenu avec les styles par défaut : il reste accessible. De même, le contenu reste visible durant l'exécution du script qui met définitivement en forme la page.

Les caractères dans le code Css incorporé à JavaScript doivent être échappés. Ainsi…

```
em::before {
    content: '«\202F'
}
kbd::before {
    content: '\7B'
}
```

… ne fonctionne pas, contrairement à…

```
em::before {
    content: '«\u202F'
}
kbd::before {
    content: '\x7B'
}
```

Les caractères Unicode compris entre 0 et 255 doivent être échappés avec le préfixe { \x }. Les caractères Unicode compris entre 0 et 65 535 doivent être échappés avec le préfixe { \u }. Sous Ecma 6 ↗, il est loisible d'écrire { \u{7B}\u{202F} }. L'article « Escape sequence types » ↗ constitue une précieuse documentation.

Limitations

Attributs

Avec le paquet « lymnee.js » ↗ sous Node.js ↗, la création d'attributs de données ↗ supplémentaires avec JavaScript ou Css nécessite de ne pas utiliser le préfixe « data-ym- » ou un espace de nom équivalent afin d'éviter leur localisation par XML Path Language ↗.

A contrario, le script Php et les fichiers en JavaScript ne reconnaissent pas les attributs de données ↗ contenus dans le code JavaScript.

L'usage des entités au format décimal ou hexadécimal est impossible dans un attribut. Par exemple, ne fonctionne pas…

```
<ul data-my-list-style-type="'\1F34E'">
<!-- <ul data-my-list-style-type="'\1F34E'"> -->
```

… Contrairement à…

```
<ul data-my-list-style-type="'🍎'">
```

Noter que le symbole littéral est encadré par des guillemets simples droits, { ' }.

Les espaces ne sont pas pris en compte, par exemple { data-ym-content="' ”'::after }. En pareil cas, la création d'une classe s'impose.

Pseudo-classes

Les pseudo-classes ↗ ne sont pas prises en charge. Les pseudo-éléments ↗ sont supportés.

Règles @supports

Les règles @supports ↗ ne prennent pas en charge les conditions « not » et « and ».

Interaction avec JavaScript

Les méthodes Element.setAttribute() ↗ et Element.removeAttribute() ↗ permettent d'ajouter ou de supprimer un attribut de données…

> document.querySelector('#foo').setAttribute('data-ym-color', 'blue');
> document.querySelector('#bar').removeAttribute('data-ym-color');

La méthode HTMLElement.dataset ↗, qui nécessite de supprimer le tiret et de changer la casse de la première lettre suivante, ne présente pas un intérêt supplémentaire…

> document.querySelector('#foo').dataset.ymColor = 'blue';
> document.querySelector('#bar').dataset.ymColor = '';

Bogues

- Les tabulations des attributs de données ↗ les plus complexes sont incorrectes quand le code Css n’est pas compressé.
- ~~Le tri est imparfait quand les sélecteurs sont identiques : les valeurs ne sont pas ordonnées.~~
- ~~Les règles de style pour imprimer la présente page ne sont pas disponibles.~~

Document à parfaire mis à jour le 26 septembre 2020.

</details>