JEKYLL: Les fichiers statiques

Table of Contents

Un fichier statique est un fichier qui ne contient aucun élément permettant de construire dynamiquement les pages. Cela inclut les images, les PDF et tout autre contenu non rendu.

Méthode pour accéder aux fichiers statiques

Les fichiers statiques sont accessibles dans Liquid via **site.static_files**

Métadonnées par défaut

Variable	Description
file.path	chemin relatif au fichier, par exemple /assets/img/image.jpg
file.modified_time	date ou le fichier a été modifié pour la dernière fois, par exemple 2016-04-01 16:35:26 +0200
file.name	nom du fichier, par exemple image.jpg
file.basename	nom de base du fichier, par exemple image pour image.jpg
file.extname	nom de l'extension du fichier, par exemple .jpg pour image.jpg

Note: le 'file' peut être n'importe quoi. C'est une variable définie de manière arbitraire utilisée dans une logique personnelle (comme dans une boucle for). Ce n'est pas une globale du site ou une variable de page.

Ajouter une métadonnée aux fichiers statiques

Bien que l'on ne puisse pas ajouter directement de valeurs de premier plan à des fichiers statiques, on peut définir ces valeurs via la propriété par défaut dans le fichier de configuration. Lorsque Jekyll construit le site, il utilisera les valeurs ainsi définies.

Par exemple: Dans le fichier _config.yml on ajouter les valeurs suivantes à la propriété par defaults

pour identifier les fichiers images.

```
defaults : - scope : path : " assets/img" values : image : true
```

Cela suppose que les images (fichiers statiques) du site sont stockées dans un dossier assets/img où . Lorsque Jekyll construit le site, chaque image est traitée comme si elle avait la valeur **image: true** .

Maintenant on peut répertorier toutes les images contenues dans assets/img en utilisant une boucle for pour rechercher dans l'objet static_files et obtenir tous les fichiers statiques ayant cette propriété:

```
{% assign  image_files  =  site.static_files | where: "image", true  %}
{% for  myimage in image_files  %}
  {{ myimage.path }}
{% endfor %}
```

Exemple : menu dynamique de fichiers statiques

On souhate construire un menu dynamique à partir enutilisant l'arborescence des dossiers dans lesquels sont stockées les pages html préconstruites.

Organisation des fichiers statiques

Par exemple les pages html sont stockées dans des sous dossiers du dossier **_pages**:

```
├── _pages/
│       ├── rubrique1/
│       │      ├── page1-1.html
│       │      └── page1-2.html
│       └── rubrique2/
│              ├── page2-1.html
│              └── page2-2.html
```

Le menu final doit ressembler à cela

- rubrique1
    - page1-1
    - page1-2
- rubrique2
    - page2-1
    - page2-2

Utilisation d'une boucle

En utilisant Liquid 1, on peut parcourir en boucle **site.static_files**

```
  {% for node in site.static_files %}
```

... on ne traite que les objets dans _pages

```
    {% if node.path contains '_pages' %}
```

... on découpe le path de chaque objet en array contenant en [1] le dossier racine, [2] les sous dossiers, [3] le nom de l'objet

```
      {% assign node_url_parts = node.path | split: '/' %}
```

... pour chaque élément d'index [2] dans le path on construit l'entrée des rubriques

```
      {% if node_url_parts[2] != prev_url_parts %}
        {% if prev_url_parts != ""  %}
           </ul>
           </li>
        {% endif %}
        {% assign prev_url_parts = node_url_parts[2] %}
        <li>
        <a href="#" >{{ node_url_parts[2] }} </a>
        <ul>
      {% endif %}
```

... pour chaque page .html on créé une entrée dans la rubrique concernée

```
      {% if node.extname contains 'html' %}
        {% assign node_url_parts = node.path | split: '/' %}
        {% assign filename = node_url_parts | last %}
        <li><a href='{{node.path}}'>{{node.basename}}</a></li>
      {% endif %}
```

... fin du if contains “_pages”

```
    {% endif %}
```

... fin de boucle

```
  {% endfor %}
```

Pour aller plus loin

Metro UI 4 Jekyll 2.0 fournit un ensemble de ressources permettant de créer facilement des sites Web

statiques avecune interface similaire à METRO de Windows 8.

L'utilisation des composants MENU du package permet de créer des menus déroulants dans la barre de navigation du site:

- Télacharger le package Metro UI 4 Jekyll 2.0

```
$ wget https://github.com/A-G-F/metro-ui-jekyll/archive/master.zip
```

- Inclure les fichiers css et js du package

```
$ unzip master.zi
$ cp metro-ui-jekyll-master/css . -r
$ cp metro-ui-jekyll-master/js . -r
```

- Importer les feuilles de styles et les scripts dans le fichier _layouts/default.html

```
  <link href="{{ site.baseurl }}/css/metro-bootstrap.min.css"
rel="stylesheet">
  <link href="{{ site.baseurl }}/css/metro-bootstrap-responsive.min.css"
rel="stylesheet">
  <link href="{{ site.baseurl }}/css/iconFont.min.css" rel="stylesheet">
  <link href="{{ site.baseurl }}/css/style.css" rel="stylesheet">
  <!-- Load JavaScript Libraries -->
  <script src="{{ site.baseurl }}/js/jquery/jquery.min.js"></script>
  <script src="{{ site.baseurl }}/js/jquery/jquery.widget.min.js"></script>
  <script src="{{ site.baseurl }}/js/metro.min.js"></script>
```

Il est alors possible d'utilser les classes de **dropdown** pour produire des menus déeoulants:

```
  {% for node in site.static_files %}
    {% if node.path contains '_pages' %}
      {% assign node_url_parts = node.path | split: '/' %}
      {% if node_url_parts[2] != prev_url_parts %}
        {% if prev_url_parts != ""  %}
           </ul>
           </li>
        {% endif %}
        {% assign prev_url_parts = node_url_parts[2] %}
        <li>
        <a class="dropdown-toggle" href="#" >{{ node_url_parts[2] }} <span
class="caret"></span></a>
        <ul class="dropdown-menu" data-role="dropdown">
      {% endif %}
      {% if node.extname contains 'html' %}
        {% assign node_url_parts = node.path | split: '/' %}
        {% assign filename = node_url_parts | last %}
        <li><a href='{{node.path}}'>{{node.basename}}</a></li>
```

```
      {% endif %}
    {% endif %}
  {% endfor %}
```