JavaScript
Regex - partie 3*

IFNTI Sokodé – L3

05 Mai 2021

Nous avons vu comment créé des classes, les ancres, les quantificateurs... il est temps de voir les sous-masques et les assertions

1 Les sous-masques

Pour l'instant les quantificateurs et l'alternative ne peuvent cibler que le caractère (métacaractère ou classe) précédent. Nous aimerions bien pouvoir les utiliser sur des sous-motifs.

Pour délimiter des sous-motifs (ou sous-masque) nous allons utiliser les parenthèses « (» et «) ». Il faut les considérer comme en mathématiques. Elles délimitent des groupement qui sont prioritaire vis à vis de l'extérieur.

ATTENTION Les parenthèses ne fonctionnent pas dans les classes.
Exemples :

```
let masque1 = /er|t/g;
let masque2 = /e(r|t)/g;
let masque3 = /(ab)*c/g;
let masque4 = /(ab)|(bc)d/
```

2 Capture et réutilisation

Les parenthèses ont aussi un rôle de capture des sous-motifs. Capture, cela veut dire qu'en plus de la chaîne qui correspond à la regex, les chaînes correspondants à la capture seront aussi retournée.

Attention, cela ne fonctionne pas avec la méthode `match` et le drapeau « g » qui effacera les captures.

Outre la partie sélection, les captures permettent d'être réutilisées dans l'expression régulière. C'est ce que l'on appelle les `backreferences`. Pour rappeler la référence nous utiliserons le caractère d'échappement « » et un numéro correspondant à l'ordre des captures. (ex :`\1`)

ATTENTION Ce que l'on réutilise, c'est ce qui est capturé, pas le motif.
Exemple :

```
const text = "<a><b><a><c><b><c>";
let masque = /(<.>).*\1/g;
```

Exercice Faire une Expression régulière qui trouve les balises HTML et retourne le tag de la balise.

3 assertion

On appelle « assertion » un test qui va se dérouler sur le ou les caractères suivants ou précédent celui qui est à l'étude actuellement. Par exemple, le métacaractère $ est une assertion puisque l'idée ici est de vérifier qu'il n'y a plus aucun caractère après le caractère ou la séquence écrite avant $.

*Repris des cours de Pierre Giraud

Ce premier exemple correspond à une assertion dite simple. Il est également possible d'utiliser des assertions complexes qui vont prendre la forme de sous masques.

Il existe à nouveau deux grands types d'assertions complexes : celles qui vont porter sur les caractères suivants celui à l'étude qu'on appellera également « assertion avant » et celles qui vont porter sur les caractères précédents celui à l'étude qu'on appellera également « assertion arrière ».

Les assertions avant et arrière vont encore pouvoir être « positives » ou « négatives ». Une assertion « positive » est une assertion qui va chercher la présence d'un caractère après ou avant le caractère à l'étude tandis qu'une assertion « négative » va au contraire vérifier qu'un caractère n'est pas présent après ou avant le caractère à l'étude.

Notez que les assertions, à la différence des sous masques, ne sont pas capturantes par défaut et ne peuvent pas être répétées.

Voici les assertions complexes qu'on va pouvoir utiliser ainsi que leur description rapide :
— `a(?=b)` Cherche « a » suivi de « b » (assertion avant positive)
— `a(?!b)` Cherche « a » non suivi de « b » (assertion avant négative)
— `(?<=b)a` Cherche « a » précédé de « b » (assertion arrière positive)
— `(?<!b)a` Cherche « a » non précédé de « b » (assertion arrière négative)
Example :

```
let chaine = 'Bonjour, je suis Pierre et mon no. est le [06-36-65-65-65]';
let masque1 = /e(?=r)/g;
let masque2 = /e(?!r)/g;
let masque3 = /(?<=i)s/g;
let masque4 = /(?<!i)s/g;
```

4 options

Les options, encore appelées modificateurs, sont des caractères qui vont nous permettre d'ajouter des options à nos expressions régulières.

Les options ne vont pas à proprement parler nous permettre de chercher tel ou tel caractère mais vont agir à un niveau plus élevé en modifiant le comportement par défaut des expressions régulières. Elles vont par exemple nous permettre de rendre une recherche insensible à la casse.

On va pouvoir facilement différencier une option d'un caractère normal ou d'un métacaractère dans une expression régulière puisque les options sont les seuls caractères qui peuvent et doivent obligatoirement être placés en dehors des délimiteurs du masque, après le délimiteur final.

— « g » permet d'effectuer une recherche globale
— « i » rend la recherche insensible à la casse
— « s » permet au métacaractère « . » de remplacer n'importe quel caractère y compris un caractère de nouvelle ligne.
— « m » par défaut la chaine traité est considérée comme une seule ligne, les métacaractères `^` et `$` ne sont utilisés qu'une seule fois. L'option « m » permet de considérée chaque nouvelle ligne comme une nouvelle chaîne et de réappliquer `^` et `$` en début et fin de ligne.
Examples :

```
let chaine = 'Bonjour, je suis Pierre\n et mon no. est le [06-36-65-65-65]';

let masque1 = /pierre/;
let masque2 = /pierre/i;
let masque3 = /e$/;
let masque4 = /e$/gm;
let masque5 = /./gs;
```