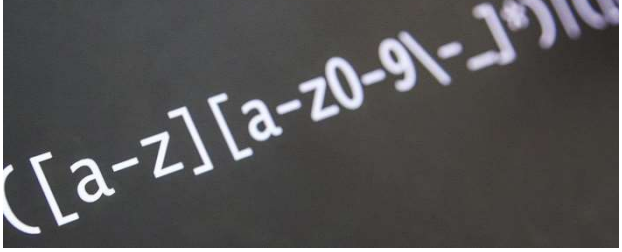

Les Expressions Régulières en JS

Formation
Développeur Web et Web Mobile
Module : 03
Développer la partie front-end d'une application web
Séquence : 03
Développer une interface utilisateur web dynamique
Séance : 03
Développer des scripts clients dans une page web

Libellé réduit :	Les RegEx en JS
Type de document :	Support de cours
Version :	1
Auteur :	Didier Bonneau
Centre afpa :	Créteil
Date de création :	29/01/2018
Date de mise à jour :	24/10/2022

Sommaire

Contenu

I. INTRODUCTION

Les expressions régulières sont aujourd'hui utilisées pour programmer des fonctionnalités de lecture, de contrôle, de modification, et d'analyse de textes. On les retrouve dans tous les langages.
Elles sont décrites par des formules ou motifs (en anglais patterns) à partir d'une suite de caractères.

A. CONSTRUCTION D'EXPRESSIONS REGULIERES EN JS:

en ligne:

```
var regex = /motif/igm;    //igm sont des marqueurs
```

à l'aide d'un constructeur avec 2 paramètres (motif, marqueur):

```
var regex = new RegExp("test","igm");
```

B. UTILISATION DANS LES SCRIPTS JS

En pratique 6 méthodes JavaScript permettent d'utiliser les expressions régulières :

- 2 sont des méthodes de l'objet Regexp : exec() et test()

```
regex.exec("chaine à tester");  //renvoi un tableau ou null
regex.test("chaine à tester");  //renvoi true or false
```

- 4 sont des méthodes de l'objet String: match(), search(), replace(), split()

```
str.match(regex);  //renvoi un tableau ou null
str.search(regex);// retourne un entier correspondant à la position ou -1
str.replace(regex, replacement ); // replacement un entier ou une fonction..
str.split(regex);// renvoi un tableau
```

II. SYNTAXE D'UNE EXPRESSION REGULIERE

Le motif d'une expression est composé de caractères simples (comme /toto/), ou de caractères simples et spéciaux, comme /to+to/ ou /prénom (to){0,2}/.

A. LES EXPRESSIONS SIMPLES

Les motifs simples sont construits à partir de caractères pour lesquels on souhaite avoir une correspondance directe.

Ex : /cdi/ sera trouvé dans la chaine "la formation cdi de l'afpa"

Ces caractères peuvent être une partie d'un mot et apparaître n'importe où dans la chaine à tester.

B. LES ENSEMBLES DE CARACTERES

Un ensemble de caractères peut être définie en énumérant tous les caractères ou définie par un intervalle. Dans les deux cas les caractères sont mis entre [ ] :

Ex : [abcdef] équivalent à [a-f] désigne tous les caractères de a à f.

Il est possible d'utiliser un caractère spécial « ^ » signifiant la complémentation (il se trouve entre les []).

Ex : [^abc] signifie tout caractères sauf les caractères a, b, c

Il est aussi possible de faire des unions : [0-9a-f] tout caractères hexadécimal

Les caractères spéciaux (voir C) mis à l'intérieur des crochets sont considérés comme des caractères normaux.

C. LES CARACTERES SPECIAUX

Certains caractères ont des significations bien particulières :

Caractère	*Signification*
****	Un *backslash* précédant un caractère normal indique que ce caractère doit être interprété comme spécial. Un *backslash* précédant un caractère spécial indique que ce caractère doit être interprété comme normal.
.	(Le point) correspond à n'importe quel caractère excepté un caractère de saut de ligne.
\d	Correspond à un chiffre et est équivalent à [0-9].
\D	Correspond à tout caractère qui n'est pas un chiffre et est équivalent à [^0-9].
\w	Correspond à n'importe quel caractère alphanumérique, y compris le tiret bas. C'est équivalent à [A-Za-z0-9_].
\W	Correspond à n'importe quel caractère n'étant pas un caractère de mot. Cela est équivalent à [^A-Za-z0-9_].
\f	Correspond à un saut de page.
\n	Correspond à un saut de ligne.
\r	Correspond à un retour chariot.
\s	Correspond à un blanc (cela comprend les espace, tabulation, saut de ligne ou saut de page).
\S	Correspond à un caractère qui n'est pas un blanc.

D. LES CARACTERES SPECIAUX DE REPETITION

Ce sont des caractères dont la fonction est de prévoir dans le modèle un nombre d'occurrences successives du littéral, du groupement ou d'élément d'ensemble sur lequel ils portent. Ils se placent immédiatement après.

Caractère	*Signification*
*****	Un nombre indéfini de fois : de 0 à n fois
+	Au moins une fois : de 1 à n fois
?	Eventuellement une fois : 0 ou 1 fois
{n}	Exactement n fois
{n,m}	Au moins n et au plus m fois
{n,}	Au moins n fois et un nombre indéfini

E. LE CARACTERE DE CHOIX

Le caractère de choix permet de prévoir, dans l'expression régulière, le choix entre différents sous-motifs, séparés par le caractère |. Comme pour les caractères de répétitions, celui-ci peut porter sur un ou plusieurs littéraux, ensembles ou groupements.

Par exemple, pour analyser une adresse, on peut imaginer l'expression suivante :

Dans cette expression apparaissent plusieurs groupements.

Parmi eux, certains constituent des choix (statut du destinataire, de la voie), d'autres permettent de faire porter un caractère de répétition sur plusieurs sous-motifs successifs.

Par exemple, 'identité du destinataire est composée d'au moins un mot, son nom (d'où le caractère de répétition + qui impose au moins 1) ; ce nom peut être composé ; il peut être précédé ou suivi d'un prénom, lui-même pouvant être composé chacun de ces éléments répondant au même modèle.

Avec une telle expression, on peut analyser une adresse du type :

M. Alain Parfait
9, Rue Marc Seguin
94000 CRETEIL

F. LES CARACTERES DE POSITIONNEMENT

Jusqu'ici, des appariements de caractères se produisaient n'importe où dans la chaine. Il existe aussi la possibilité d'apparier et donc, de contraindre encore, selon la position dans la chaîne. L'intérêt est de pouvoir mettre en place des ancrages de motifs.

Exemple : identificateur du login des comptes Afpa qui doit commencer forcément par « afpa » suivie de quatre chiffres.

proposition : l'expression régulière suivante : /afpa\d{4}/ devrait marcher
jeu d'essai : la chaîne « afpa1578 » match
la chaine « le login afpa1578 marche » match aussi

En fait pour toutes les expressions régulières que l'on a vues jusqu'ici, la vérification se contentait de tester si dans la chaîne fournie, une partie de celle-ci répondait au modèle décrit par l'expression régulière.

Il faudrait donc contraindre l'analyse à appliquer le modèle dès le début de la chaîne afin que « afpa » soit les premiers caractères de la chaine.

Voici ces caractères de positionnement :

Caractère	Signification
^	Ancre en début de chaîne
$	Ancre en fin de chaîne ou sous-chaine avec le marqueur m (voir les marqueurs)
\b	Ancre sur limite de mot
\B	Ancre sur non limite de mot

proposition corrigée : l'expression régulière : /^afpa\d{4}/ devrait marcher
jeu d'essai : la chaîne « afpa1578 » match
la chaine « le login afpa1578 marche » ne match pas
la chaine « afpa1578 à moi » match aussi

Si on veut exactement afpa suivi de 4 chiffres et rien d'autre :

proposition correcte : l'expression régulière : /^afpa\d{4}$/ devrait marcher
jeu d'essai : la chaîne « afpa1578 » match
la chaine « afpa1578 à moi » ne match pas

Ne pas confondre le caractère d'ancrage en début avec le caractère de complémentation. Même s'ils ont une représentation similaire, il n'y a aucune ambiguïté car l'un se situe obligatoirement en début d'expression (après /) tandis que l'autre ne peut apparaître qu'en début de définition d'ensemble (après [).

G. LES CARACTERES DE GROUPEMENT (ET DE REFERENCEMENT)

Les caractères de groupement « () » ont pour fonction de regrouper plusieurs éléments constituant un sous-motif du modèle en un seul élément. Le groupement s'opère simplement en l'encadrant d'une paire de parenthèses. Il y a plusieurs utilités à pouvoir regrouper des éléments :

- on peut ainsi faire porter les caractères de répétition que nous venons de voir sur l'ensemble des éléments du modèle ainsi regroupés. Ainsi, c'est cet ensemble qui sera dupliqué et non les éléments constitutifs individuellement ;
- on peut aussi référencer ce groupement de façon à prévoir une autre occurrence des éléments de la chaîne analysés avec lesquels il a été apparié, plus loin dans le modèle.

Le référencement dont il vient d'être question s'opère par l'apparition dans l'expression régulière du caractère « \num » où num désigne le n° de parenthésage auquel il fait référence (comme à 1).

Exemple : supposons que l'on veuille analyser la chaîne « JavaScript » encadrée de simples ou doubles guillemets (les deux devant être identiques bien entendu).

proposition : l'expression régulière : /['"]JavaScript['"]/ devrait marcher
jeu d'essai : la chaine "JavaScript" match
la chaine 'JavaScript' match
la chaîne 'JavaScript" match
la chaine "JavaScript' match

Si l'on veut une analyse correcte, il faut utiliser le regroupement mémorisé (['"]) et l'apparieller avec \1 :/(['"])JavaScript\1/.

proposition : l'expression régulière : /(['"])JavaScript\1/ devrait marcher
jeu d'essai : la chaine "JavaScript" match
la chaine 'JavaScript' match
la chaîne 'JavaScript" ne match pas
la chaine "JavaScript' ne match pas

Grâce à cet essai, on constate que cette expression autorise en début de chaîne le caractère ' ou le caractère ", mais une fois le premier analysé, il contraint le second à lui être identique.

H. LES MARQUEURS

Les marqueurs d'appariement sont au nombre de trois. Ils se placent après l'expression régulière sur laquelle ils portent.

On dispose de marqueur « i » qui indique s'il est présent que l'analyse de la chaîne doit se faire sans tenir compte de la casse des caractères.

Exemple : pour les nombres hexadécimaux : /^[A-F0-9]+$/i.

Le marqueur « g » signifie que l'appariement doit se faire d'une façon globale sur l'ensemble de la chaine.

Le marqueur « m » signifie que la chaine est multi ligne. Elle possède des caractères de retour à la ligne. Important pour les ancres de fin de chaine où $ matchera les sous-chaines.

III. LES PROPRIETES & METHODES DE LA CLASSE REGEXP

A. LA PROPRIETE INDEX

Cette propriété index contient la position du caractère de la chaîne à partir duquel l'appariement a eu lieu.

B. LA PROPRIETE SOURCE

Accessible en lecture seulement, cette propriété est une chaîne de caractères contenant le texte de l'expression régulière référencée. Elle est mise à jour à la création de l'expression régulière.

C. LA PROPRIETE LASTINDEX

C'est une propriété accessible en lecture/écriture dans laquelle est enregistré, dans le cas de recherches globales (utilisation du marqqueur g), l'indice dans la chaîne source, à partir duquel la recherche devra reprendre. Cette propriété est en particulier utilisée par les méthodes test() et exec().

D. LES PROPRIETES GLOBAL ET IGNORECASE

Ces deux propriétés sont elles aussi accessibles seulement en lecture sont deux booléens permettant de récupérer la valeur des attributs de l'expression régulière référencée. Si global

est vrai, cela signifie que la recherche d'appariements est globale à toute la chaîne ; si ignoreCase est vrai il n'est pas tenu compte de la casse des caractères.

Nous allons voir à présent les propriétés statiques de la classe RegExp. Rappelons qu'à la différence des propriétés d'instance qui s'applique à tout objet instance de la classe, chaque propriété statique, est unique dans la classe et donc commune à tous les objets de cette classe.

E. LES PROPRIETES REGEXP.LEFTCONTEXT & REGEXP.RIGHTCONTEXT

A chaque fois qu'une recherche d'appariement entre une expression régulière et une chaîne de caractères est opérée, par des méthodes que nous verrons plus loin, qu'elles soient de la classe RegExp ou de la classe String, ces indicateurs seront positionnés. Le premier contiendra la partie de la chaîne située à gauche du dernier appariement effectué, tandis que le second contiendra la partie droite.

F. LES PROPRIETES REGEXP.LASTMATCH & REGEXP.LASTPAREN

Les deux précédentes propriétés permettent de récupérer les contextes gauche et droit du dernier appariement opéré. Ce qui se trouve entre les deux, c'est à dire la partie de la chaîne source mise en concordance avec l'expression régulière sera disponible dans la propriété lastMatch.
A l'intérieur de la sous-chaîne appariée contenue dans lastMatch, on peut de plus dégager la partie qui s'est appariée au dernier groupement du précédent appariement. Celle-ci est contenue dans lastParen. Il est intéressant de noter que l'on peut obtenir les sous-chaînes mises en appariement avec les neuf premiers groupements par l'intermédiaire des propriétés RegExp.$1, RegExp.$2, ..., RegExp.$9.

G. LA PROPRIETE REGEXP.MULTILINE

Ce booléen permet de préciser si la chaîne sur laquelle agit l'expression régulière contient une seule ou plusieurs ligne. Selon le cas, le comportement pourra être différent et c'est en particulier le cas pour les ancrages de début et de fin. Dans le cas où le texte n'est pas précisé multiligne, ^ représente le début de la chaîne et $, la fin de la chaîne, même s'il contient des littéraux \n. Dans le cas où ce même texte est déclaré multiligne, ^ représente le début et $ la fin de chaque ligne.

H. LA METHODE TEST(<CHAINE>)

Cette méthode essaie d'opérer un appariement entre l'expression régulière référencée et la chaîne de caractères donnée en paramètre à partir de l'indice spécifié par la propriété lastIndex dans le cas d'un appariement global. Si un appariement est possible, lastIndex est réactualisé et la valeur booléenne renvoyée est true. Dans le cas contraire, la valeur renvoyée est false et lastIndex est remis à 0.

I. LA METHODE EXEC(<CHAINE>)

Cette méthode essaie de procéder à l'appariement de l'expression régulière sur la chaine. En cas d'échec, la valeur de retour est null. Dans le cas contraire, la valeur retournée est un tableau contenant à l'indice 0, la sous-chaîne appariée avec l'expression régulière et dans les éléments suivants les sous-chaînes appariées avec les éventuels groupements parenthésés. Par ailleurs, en cas d'appariement, cette méthode actualise lastIndex, si bien que si elle est rappelée avec la même référence, elle procède à une nouvelle recherche à

partir de cet emplacement. Comme pour la méthode précédente, en cas d'échec, lastIndex est remis à 0.

IV. LES METHODES DE LA CLASSE STRING METTANT EN JEU LES EXPRESSIONS REGULIERES

A. LA METHODE MATCH(<EXPR. REGUL.>)

Cette méthode permet de rechercher dans la chaîne référencée la ou les portions de celle-ci qui répondent à un modèle (expression régulière) fourni en paramètre. Dans le cas où aucune portion de la chaîne ne répond au modèle, la valeur retournée est null. Dans le cas contraire, la valeur rendue est un tableau construit de façon différente selon que l'on ne recherche qu'une (la première à gauche) mise en concordance ou chacune d'elles. Dans le premier cas, le premier élément du tableau comporte la sous chaîne répondant au modèle et les éléments suivants contiennent les sous-parties correspondant aux éventuelles sous expressions parenthésées du modèle. Dans le cas où toutes les concordances sont recherchées, le tableau rendu se limite à chacune des parties de la chaîne source répondant au modèle.

On voit donc que la similitude que l'on trouvait au début entre exec() et match() n'est pas totale puisque alors que exec() n'évalue qu'un seul appariement, match() effectue tout les appariements que l'expression autorise sur la chaîne passée en paramètre. En particulier, le marqueur g prendra pleinement sa signification dans match() ce qui n'est pas le cas dans exec().
Dans l'exemple qui suit, nous allons comparer les comportements de match() et exec() dans un contexte de recherche globale ou pas.

Le programme...

```
var Tab;
var Texte="Un1,\nDeux2,Trois3,\nQuatre4,Cinq5.";
```

`var Model=/([A-Z]([a-z]+)\d([,.]))+/g;`	`var Model=/([A-Z]([a-z]+)\d([,.]))+/;`
`Tab = Model.exec(Texte);`	`Tab = Texte.match(Model);`

```
if (Tab==null) alert("Aucun appariement n'est possible !");
else {
  S = "";
  for(var I = 0; i < Tab.length; i++)
    S += "En " + i + " : " + Tab[i] + "\n";
  alert(S);
}
```

B. LA METHODE REPLACE(<EXPR. REGUL.>,<REMPLACEMENT>)

Cette méthode référence une chaîne de caractères dans laquelle une sous-chaîne appariée avec l'expression régulière donnée dans le premier paramètre va être remplacée par le second paramètre. Généralement,, le remplacement s'opère par une chaîne de caractères. Mais il peut advenir que ce soit la sous-chaîne extraite de l'appariement que l'on veuille modifier tout en gardant tout ou partie des éléments qui la composent. Dans ces conditions il est souhaitable de pouvoir accéder à ces parties sous forme paramétrée afin de les

répercuter dans le paramètre de remplacement sous la forme et dans l'ordre désirés. Cela est possible si l'on prévoit dans l'expression régulière des groupements parenthèsés que l'on peut ensuite référencer dans le paramètre remplacement par les caractères $1, $2,...,$9. Plusieurs caractères faisant référence à diverses portions de la chaîne sont ainsi disponibles

Caractère	*Signification*
$1, $2,..., $9	sous-chaînes appariées aux 9 premiers groupements
$&	sous-chaîne appariée à l'expression régulière
$'	sous-chaîne à droite de l'appariement (idem rightContext)
$`	sous-chaîne à gauche de l'appariement (idem leftContext)
$+	dernier groupement apparié

Exemple : Les dates sont codées différemment dans les pays anglo-saxons et en France. Sur la base de deux chiffres pour le jour (jj), le mois(mm) et l'année (aa), une date sera codée en France jjmmaa, alors qu'en Angleterre, par exemple, elle sera codée mmjjaa. Quant au séparateur, qu'il soit " ", "/" ou "-", il demeurera identique. En supposant que le texte sur lequel porte l'opération soit référencé par la variable Txt, le problème sera résolu par :

```
Txt.replace(/\s*(\d{1,2})([ \/-])(\d{1,2})\2(\d{1,2})\s*/,"$3$2$1$2$4")
```

ATTENTION : Ces caractères n'ont de validité qu'à l'intérieur de l'appel à la méthode replace(). Si vous essayez de les utiliser à l'extérieur vous aurez de grosses surprises !... A moins que vous preniez en compte qu'ils sont aussi des propriétés statiques de la classe RegExp et que vous les utilisiez sous la forme qui convient.
Par ailleurs, de façon logique, lorsque aucun appariement n'est possible, la chaîne référencée reste inchangée.

C. LA METHODE SEARCH(<EXPR. REGUL.>)

Nous avons vu, dans le chapitre précédent, la méthode indexOf() qui permettait de déterminer l'emplacement d'une sous-chaîne dans une chaîne donnée en référence. En particulier, nous avons mis en évidence ses faiblesses en l'utilisant dans un exemple où l'on cherchait à compter le nombre d'occurrences d'un caractère car pour prendre en compte la casse de celui-ci, il nous aurait fallu procéder à deux appels, l'un pour rechercher les occurrences minuscules et l'autre pour les majuscules. La méthode présentée ici bénéficie des possibilités offertes par les expressions régulières pour rechercher une sous-chaîne vérifiant un modèle particulier. Reprenons l'exemple auquel il était fait référence à l'instant :

Le texte...

```
"Il est certain que JavaScript est un langage simple à apprendre. Il n'en est pas pour autant rudimentaire... "
```

Le programme (version search)...

```
var Compt = 0;
var Deb = Texte.search(/i/i);
while (Deb != -1) {
  Compt++;
  Deb = RegExp.rightContext.search(/i/i);
}
alert("Nbre d'occurrences de 'i' : " + Compt);
```

Les résultats à tester...

On constate qu'avec indexOf("i"), la valeur retournée correspond au nombre de caractères très précisément identiques à celui donné en paramètre. Tandis qu'avec search() on peut préciser, grâce à l'attribut i que la casse est indifférente. On voit aussi que malgré le fait que search() ne dispose pas de paramètre indiquant l'indice de début de recherche, comme c'était le cas pour indexOf(), grâce à la propriété statique rightContext, on peut balayer la totalité de la chaîne fournie. Ces propriétés de classe n'existant pas sous Internet Explorer, cet exemple ne fonctionnera pas de façon satisfaisante sous ce navigateur.

NB : Cette méthode met à jour les propriétés statiques de RegExp.

Pour terminer signalons une autre méthode de la classe String pouvant utiliser une expression régulière. Nous avons vu dans le chapitre précédent la méthode split(), qui, sur la base d'un séparateur fourni en paramètre, transformait la chaîne référencée en un tableau dont les éléments étaient la succession des sous-chaînes ainsi séparées.
Cette méthode, en l'état, atteint vite ses limites. Si, par exemple, nous voulons obtenir un tableau constitué des différents mots d'un texte sur la base d'un seul séparateur, on va se heurter à plusieurs problèmes. Tous les caractères de ponctuations vont soit faire partie de mots (c'est le cas de la virgule ou du point, toujours accolés au mot qui les précède), soit constituer des mots eux-mêmes (c'est le cas de ; ? ! qui réclament un espace de part et d'autre). Cela sans compter les espaces doublés ou oubliés après la virgule ou le point, les apostrophes, etc. Bref, les mots dégagés et donc le nombre de mots trouvés seraient entachés d'erreur.

En utilisant l'expression régulière /[',.;?!-]+/ en tant que séparateur, voire, beaucoup plus simplement /\W+/, le problème sera résolu !....

Considérons par exemple, le texte suivant :

Ho cà ! n'ai-je pas lieu de me plaindre de vous ? ***Et, pour n'en point mentir, n'êtes vous pas méchante*** ***De vous plaire à me dire une chose affligeante ?*** *(Tartuffe de Molère)*	split(/[',.;?!-]+/) split(/\W+/)

Pour les deux expressions régulières, le résultat est le même. Par contre, selon que vous serez sous Netscape ou Explorer, le résultat sera 33 pour l'un, ce qui est faux, ou 32 pour l'autre, ce qui est exact. Effectivement, regardez bien l'énumération des mots répertoriés. Celle-ci se termine par une virgule sous Netscape, preuve qu'il y a un mot vide comptabilisé, ce qui n'est pas le cas avec Explorer.

NB : Vous remarquerez aussi, sous Explorer, que dans la deuxième méthode, les caractères accentués ne sont pas reconnus comme appartenant à \w. Ils sont donc inclus dans \W et donc interprétés comme séparateurs. Si bien qu'un mot contenant n caractères accentués sera décompté pour n+1 mots. En particulier dans cet exemple, vous pouvez constater que "méchante" compte pour deux mots à cause du "é", alors que "à" (de "...plaire à me dire..."), disparaît. Par ailleurs dans le texte analysé, "çà" a en fait été orthographié "cà", ce qui

explique que le "c" subsiste et qu'un mot soit ainsi détecté. Si l'orthographe avait été correcte, il manquerait un mot au décompte final... Conclusion ?