



Concepteur Développeur en Informatique

Développer des composants d'interface

S'initier aux expressions régulières

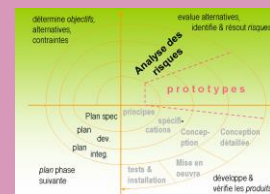
Accueil

Apprentissage

PAE

Evaluation

Impossible d'afficher l'image.



Localisation : U03-E05-S02

1.	Introduction	3
2.	Rédaction d'une expression régulière.....	3
2.1.	Définition du pattern ou motif	3
2.2.	Les classes de caractères	5
2.3.	Les quantificateurs	5
2.4.	L'alternative	6
2.5.	Les génériques.....	6
2.6.	Les méta-caractères	6
3.	Test d'une expression.....	7
4.	Extraire ou remplacer des valeurs d'une chaîne	7
4.1.	Utilisation de la méthode split	7
4.2.	Utilisation de la méthode match	8
4.1.	Utilisation de la méthode replace	9

1. Introduction

Les expressions régulières permettent de définir des modèles qui sont le plus souvent utilisés pour vérifier si une valeur est conforme à l'ensemble des règles définies dans ces modèles.

Les expressions régulières sont donc principalement utilisées pour effectuer des contrôles de validation. Elles peuvent être aussi mises en œuvre pour effectuer des recherches et des remplacements de texte ou partie de texte.

Les expressions régulières ne sont pas réservées à un langage mais constitue un langage propre disponible dans de nombreuses architectures logicielles (java, dot net, javascript,php,...) sous une forme très similaire.

Il y a bien entendu quelques différences de dialecte...

Elles peuvent simplifier la programmation des contrôles de données.

Tout au moins, l'expression du contrôle de validité sera exprimée dans une forme beaucoup plus concise que le même contrôle dans un langage de haut niveau tel que C# ou JavaScript.

Les expressions régulières peuvent être utilisées notamment pour valider les formats des données saisies (N° téléphone, N° carte bleue, e-mails, etc....).

Les expressions régulières sont créées de la manière suivante (manière la plus simple, sachant qu'il existe d'autres méthodes et d'autres paramètres) :

```
var nomExpression = /chaîne/;
```

Le nom de l'expression régulière créée sera le nom utilisé lors des contrôles de validité. La chaîne va représenter le format de l'expression que l'on souhaite obtenir.

La syntaxe des expressions régulières en javascript trouve ses origines dans le langage de script pour composants serveur perl.

Les expressions régulières peuvent s'utiliser avec des méthodes de l'objet string ou à partir d'un modèle spécifique RegExp.

2. Rédaction d'une expression régulière

2.1. Définition du pattern ou motif

Nous allons tout d'abord utiliser des formes simples et une méthode test qui permet de vérifier si la chaîne passée en argument vérifie la règle.

Nous avons besoin de deux objets, un objet de type RegExp dont la méthode test permettra de vérifier l'existence de l'expression dans un deuxième objet de type string contenant un littéral.

Expressions régulières

Nous allons utiliser dans notre premier exemple deux caractères de l'expression régulière qui permettent de rechercher une occurrence de caractères en début ou en fin de chaîne :

- Le symbole ^ pour déterminer si la chaîne commence par
- Le symbole \$ pour déterminer si la chaîne se termine par

Pour instancier un objet de type RegExp, il suffit de lui fournir une valeur entre / / ou d'utiliser un constructeur du type RegExp

```
var ExpReg = / /
```

ou

```
var ExpReg = new RegExp();
```

L'intérêt d'utiliser un objet RegExp dans la deuxième forme est d'instancier l'expression régulière avec une variable de type chaîne, ce qui n'est pas envisageable avec la première expression.

```
var expressionReg = new RegExp(chaine);
```

Le constructeur de l'expression régulière peut prendre un ou deux arguments :

- Le premier l'expression
- Le deuxième précise l'option de recherche (i pour ignorer la casse, g pour global, m pour une recherche sur plusieurs lignes).
Ces options peuvent être utilisées individuellement ou associées dans n'importe quel ordre : ig, gim, g indique que nous souhaitons traiter toutes les occurrences. m, moins usité, permet de préciser que les symboles ^et \$ s'appliquent aux bornes d'une ligne et non aux bornes du texte. Voir ci-dessous.

Par exemple, si nous souhaitons rechercher si l'expression 'car' débute une chaîne sans tenir compte de la casse, nous pouvons instancier nos objets de type expression régulière de ces deux manières :

- var ExpReg = /^car/i sous la forme /expression/ option de recherche
- var ExpReg = new RegExp('^car','i')

Symbole	Signification	Exemple Expression	Commentaire
^	Début de chaîne	/^ab/	Recherche si la chaîne commence par ab
\$	Fin de chaîne	/ab\$/	Recherche si la chaîne finit par ab
.	Un caractère	/.../	Recherche si la chaîne possède au moins 3 caractères
Aucun	Contient	/aca/	Recherche si la chaîne contient aca

2.2. Les classes de caractères

Ecrites en crochet [], les classes permettent de préciser un ensemble de caractères ou type de caractères.

Ainsi l'expression `/car[rt]e/` permet de préciser que la chaîne contient car et le caractère r ou t et le caractère e. Les mots carte, carre sont donc validés.

Le tiret permet de représenter l'intervalle dans la classe

Symbole	Signification	Expression	Commentaire
[]	Ensemble de caractères	<code>/car[rt]e/</code>	Contient car suivi de r ou t suivi de e
[.-.]	Caractère compris entre	<code>/car[r-t]e/</code>	Validera carte, carse et carte
		<code>/[0-9]/</code>	Validera une chaîne qui contient au moins un chiffre

Nous venons de préciser que la chaîne valide doit inclure un caractère ou une classe de caractères mais nous pouvons aussi exclure une classe de caractères en préfixant la classe du symbole ^.

Ainsi l'expression `/^car[^\r-t]/` validera les chaînes qui commencent par car et ne se poursuivent pas par r, s ou t ; Ainsi, carte et carré ne sont pas valides, mais carotte est valide.

2.3. Les quantificateurs

Certains symboles permettent de préciser combien de fois un caractère ou une classe de caractères qui précède le symbole doit être présent dans la chaîne.

Ils sont au nombre de trois, le point d'interrogation (?), l'astérisque (*) et le plus (+).

Ils sont donc associés aux classes de caractères et permettent d'en préciser le nombre d'occurrences.

Symbole	Signification	Expression	Commentaire
?	0 ou une occurrence	<code>/^1[a-z]?2/</code>	Commence par 1 suivi de 0 ou un caractère compris entre a et z suivi de 2 1a2 et 12 sont valides mais pas 1aa2 Equivalent à {0,1}
+	1 ou plusieurs occurrences	<code>/^1[a-z]+2/</code>	1a2 et 1aa2 sont valides mais pas 12
*	0,1 ou plusieurs occurrences	<code>/^1[a-z]*2/</code>	1a2, 12 et 1aa2 sont valides

Il est possible de préciser le nombre d'occurrences minimum et maximum d'une classe ou d'un caractère en utilisant les accolades {}. 3 formes possibles :

{n} : n occurrences du caractère ou de la classe de caractères qui précède. L'expression `/^a{2}b/` valide aab, mais pas ab ni aaab.

{n,m} : n occurrences au moins, m au plus. L'expression `/^a{2,3}b/` valide aab, aaab mais pas aaaab.

{n,} : n caractères sans limite de nombres d'occurrences maxi. L'expression `/^a{2,}b/` valide aaaab.

2.4. L'alternative

Il est possible de préciser des conditions de présence alternatives avec le symbole | et les () qui encadre le groupe conditionnel.

Ainsi, nous pouvons exprimer que la chaîne sera valide si elle commence par 1 ou 2 ainsi :

/^(1|2)/. l'expression /^(1|2)/ validera les chaînes 1ab, 2bc mais pas 3ab.

Nous pouvons bien entendu utiliser ce principe avec les classes et préciser le nombre d'occurrences nécessaire.

L'expression /[a-c]{2,3}1[d]{2}1/ valide les chaînes abc1 et dd1 mais pas ab1 ou ddd1

2.5. Les génériques

Il est possible de substituer des types génériques aux classes pour alléger l'écriture des expressions régulières. Ils sont représentés sous la forme d'un caractère alpha en minuscule ou en MAJUSCULE (opposé) précédé du symbole d'échappement anti-slash \.

Générique	Signification	Expression	Commentaire
\d	un caractère dans la classe des digits 0 à 9	/^d/	Commence par un digit
\D	un caractère non digit	/^D/	Ne commence pas par un chiffre
\s	un espace	/^s/	Commence par un espace
\S	un caractère qui n'est pas un espace	/^S/	Ne commence pas par un espace
\w	un caractère de la classe alpha numérique	/^w/	Commence par un caractère de la classe comprise entre 0-9 et a-Z. Underscore compris Valide _a mais invalide ?a
\W	un caractère autre que alpha numérique	/^W/	Ne commence pas par un caractère alphanumérique. Valide ?a mais pas _a

2.6. Les méta-caractères

Si vous devez rechercher des caractères spéciaux désignés aussi sous le vocable de méta caractères qui sont utilisés à des fins particulières pour préciser les motifs des expressions régulières, vous devez les échapper en les faisant précéder du caractère anti-slash \.

La liste de ces symboles est ^ . [] \$ () * + ? | { } \

3. Test d'une expression

La classe RegExp propose deux méthodes utiles :

- La méthode Test qui renvoie true si la chaîne est conforme à l'expression.

```
var ExpReg = new RegExp(document.getElementById('txtExpression').value,'i');  
if (ExpReg.test(document.getElementById('txtChaine').value))
```

- La méthode Exec qui retourne un tableau avec la portion de chaîne correspondant à l'expression ou null

4. Extraire ou remplacer des valeurs d'une chaîne

Les expressions régulières permettent de vérifier le format d'une chaîne mais pas seulement.

Elles peuvent être aussi rédigées pour extraire des valeurs présentes dans une chaîne ou remplacer un ensemble de caractères par d'autres.

L'objet String possède 4 méthodes qui peuvent prendre en argument une expression régulière :

- La méthode split
- La méthode search
- La méthode match
- La méthode replace

Vous trouverez dans ce document quelques illustrations d'usage de ces méthodes.

Rappel : Le terme g passé en deuxième argument de l'expression régulière indique qu'il est nécessaire de traiter toutes les occurrences. Sans précision, seule la première occurrence correspondant au motif est retenue.

4.1. Utilisation de la méthode split

Par exemple, nous pouvons extraire les mots d'une liste dont nous fournissons les séparateurs au sein d'une expression régulière qui sera utilisée avec split.

```
function ExtraireMot(chaine) {  
var expSeparateurs = new RegExp("[ ;.?!:]","gi");  
var tabMots = chaine.split(expSeparateurs);  
var liste = document.getElementById('selMots');  
  
while (liste[0]) {  
    liste.remove(liste[0]);  
}  
for (var index = 0; index < tabMots.length; index++) {  
    var option = document.createElement("option");  
    option.text = tabMots[index];  
    liste.add(option, null);  
}  
}
```

Dans l'exemple ci-dessus, les mots sont insérés dans un objet combo box.

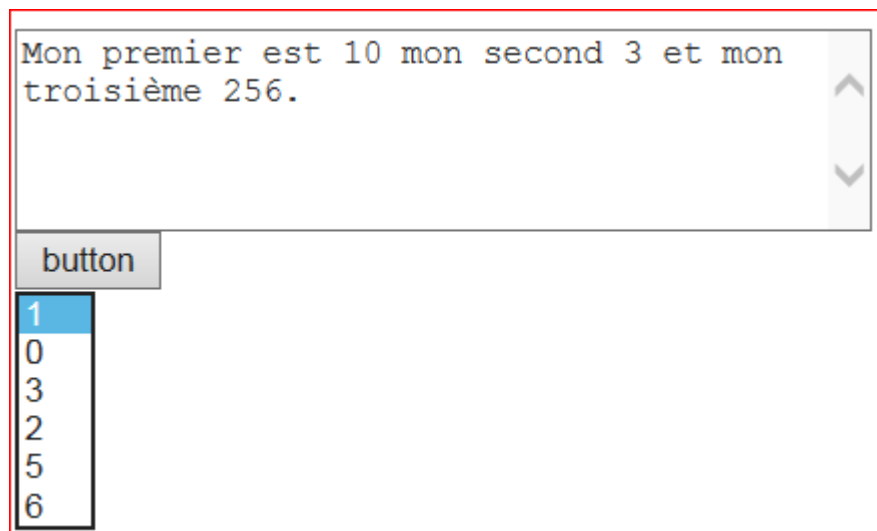
4.2. Utilisation de la méthode match

Dans ce deuxième exemple, je souhaite récupérer dans un tableau tous les éléments qui peuvent être validés par l'expression régulière fournie.

J'ai choisi ici de récupérer les chiffres présents dans la chaîne.

```
function ExtraireNombre(chaine) {  
    var expChiffre = /[0-9]/g  
    var tabNombres=chaine.match(expChiffre);  
    var liste = document.getElementById('selMots');
```

Les éléments du tableau sont ensuite insérés dans l'objet combo box. Résultat affiché ci-dessous.



Nous pourrions exprimer les choses de manière légèrement différente pour récupérer les nombres (expression de 1 à 10 chiffres).

Expression régulière est modifiée dans ce but. On recherche alors des expressions constituées de 1 à 10 chiffres.

```
function ExtraireNombre(chaine) {  
  
    var expNombre = /[0-9]{1,10}/g;  
    var tabNombres=chaine.match(expNombre );  
    var liste = document.getElementById('selMots');
```


Mon premier est 10 mon second 3 et mon troisième 256.

button

10

3

256

4.1. Utilisation de la méthode replace

Nous pouvons aussi remplacer une occurrence trouvée par une autre valeur avec la méthode replace.

A partir des mêmes valeurs que l'exemple précédent, nous remplaçons ici les nombres par NNN.

```
function RemplacerNombre(chaine) {  
  
    var expNombre = /[0-9]{1,10}/g;  
  
    chaine=chaine.replace(expNombre,'NNN');  
    alert(chaine);  
  
}
```

Nous obtenons alors le résultat souhaité.

