Expressions régulières en PHP

Hugo Etiévant

Dernière mise à jour : 23 février 2003

Présentation

Tout programmeur s'est déjà vu obligé de traiter des chaînes de caractères. Sur le web, les pages elles-mêmes, les données transmises aux scripts et celles provenant des bases de données sont des chaînes de caractères qu'il faut traiter, analyser, corriger… Chose ardue et quasi-impossible sans l'utilisation d'un formidable outil que sont les *expressions régulières* (dites aussi *expressions rationnelles*).

Nous n'expliquerons pas l'algorithmique sous jacente très complexe (machines à états, automates, graphes, récursivité…) mais illustrerons seulement son utilisation via les fonctions de PHP.

On utilisera par la suite le terme **regex** – terme emprunté à l'anglais – pour désigner une expression régulière.

A noter que les fonctions PHP dont il est question ici sont conformes à la norme POSIX et hérité du langage Perl.

Motifs

Une regex s'apparente à une expression mathématique, car on y trouve des opérateurs, des valeurs et des variables.

Les regex permettent de se lancer à la recherche de motifs décrits par la combinaison d'opérateurs et de valeurs.

Les fonctions de recherche de motifs du PHP retournent vrai si le motif a été trouvé dans une chaîne de caractères, elles permettent aussi d'extraire de cette chaîne la sous chaîne qui correspond au motif et de la modifier.

Une utilisation récurrente des regex consiste en la recherche de mots clés dans des fichiers ou dans une base de données ou encore en la vérification des données saisies par l'utilisateur afin de s'assurer qu'elles respectent un format prédéfini, ou même d'opérer des conversions de format.

Exemple

Par exemple on peut se lancer à la recherche du mot **‘voiture’** dans la chaîne **$str** :

```
if(ereg(‘voiture’, $str)) {
        echo ‘ok’;
} else {
        echo ‘invalide’;
}
```

Le motif ici est réduit à sa plus simple expression : **‘voiture’** est le motif de recherche, il consiste juste en une valeur (chaîne de caractères).

On peut le compliquer pour accepter une majuscule en début de mot : **‘[Vv]oiture’**. On pourra également interdire que ce motif soit inclus dans un mot plus grand comme **‘voiturette’** : **‘[Vv]oiture([^[:alpha:]]|$)’**. Mais autoriser son pluriel : **‘[Vv]oiture(s)?([^[:alpha:]]|$)’**.

Vous voyez, ça devient vite du charabia !

Les fonctions PHP

ereg($motif, $chaîne [, $vars]) : retourne VRAI si le motif **$motif** est trouvé dans la chaîne **$chaîne**. Le tableau **$vars** contiendra les sous chaînes de **$chaîne** vérifiant le motif.

ereg_replace($motif, $nouvelle, $chaîne) : retourne la chaîne **$chaîne** dont les sous chaînes vérifiant le motif **$motif** sont remplacées par la chaîne **$nouvelle**.

split($motif, $chaîne [, $num]) : retourne un tableau – d'au maximum **$num** éléments – des sous chaînes de **$chaîne** qui se trouvent séparées par des délimiteurs vérifiant le motif **$motif**.

Les fonctions **eregi()**, **eregi_replace()**, **spliti()** sont identiques aux précédentes mais insensibles à la casse.

Note : la **casse** est la différence majuscules/minuscules.

Description des motifs

Les motifs sont décrits par ces trois caractéristiques :

- les caractères, chaînes ou classes de caractères qui les compose
- leur nombre d'apparition
- leur position
- les alternatives

Chaîne de caractères

Un motif peut être constitué d'une simple chaîne.

Exemple : **ereg(’Paris’, ‘Je vis à Paris.’)**

Cet exemple renvoie VRAI car le motif **‘Paris’** a été trouvé dans la chaîne **‘Je vis à Paris.’**.

Exemple : **ereg(’hugo’, ‘Hugo Pratt fut un grand dessinateur de BD.’)**

Cet exemple renvoie FAUX car le motif **‘hugo’** n'a pas été trouvé dans la chaîne **‘Hugo Pratt fut un grand dessinateur de BD.’**.

Attention à la casse des caractères !

Alternative

Un peut décider d'imposer la présence d'une chaîne parmi plusieurs grâce au caractère spécial de signification OU booléen : **|** .

Exemple : **ereg('hugo|Hugo|HUGO', 'Hugo Pratt fut un grand dessinateur de BD.')**

Cet exemple renvoie VRAI car le motif a été trouvé dans la chaîne **'Hugo Pratt fut un grand dessinateur de BD.'**. Et c'est en particulier **'Hugo'** qui a été trouvé.

Exemple :

```
$motif = 'hugo|Hugo|HUGO';
$str = 'Hugo Pratt fut un grand dessinateur de BD.';
if(ereg($motif, $str, $regs))
        foreach($regs as $elem)
                echo $elem.'<br />';
```

Cet exemple recherche et affiche le motif trouvé. Ici ce sera **'Hugo'**.

Ensemble de caractères (I)

Un peut vouloir recherche une chaîne complète : **‘voiture’** ou bien seulement un caractère parmi un ensemble. Les ensembles sont définis entre crochets **[ ]**. Pour rechercher l’une des voyelles dans un mot, on utilisera le motif suivent : **‘[aeiouy]’**.

Exemple : **ereg(’[aeiouy]’, ‘voiture’)**

Cet exemple renvoie VRAI puisque le mot **‘voiture’** contient au moins une des voyelles définies dans le motif.

Pour rechercher une plage de caractères, on indiquera le premier et le dernier caractères séparés par un tiret pour demander de rechercher un caractère parmi ceux de l’alphabet situés entre ces deux caractères. Pour rechercher les caractères entre **‘a’** et **‘d’** dans le mot **‘voiture’** :

Exemple : **ereg(’[a-d]’, ‘voiture’)**

Cet exemple renvoie FAUX car le mot **‘voiture’** ne contient aucune des lettres de l’alphabet comprises entre **‘a’** et **‘b’**.

Ensemble de caractères (II)

On peut étendre notre logique aux chiffres. Pour rechercher un chiffre entre **‘0’** et **‘9’**, le motif sera le suivant : **‘[0-9]’**.

Exemple : **ereg(’[0-9]’, ‘voiture’)**

Cet exemple renvoie FAUX car la chaîne **‘voiture’** ne contient aucun des chiffres parmi ceux de l’ensemble du motif.

Un peu ajouter à notre ensemble l’opérateur de négation **^**. Cet opérateur ne peut apparaître qu’en début d’ensemble et s’applique à tout l’ensemble.

Exemple : **ereg(’[^0-9]’, ‘voiture’)**

Cet exemple renvoie VRAI car effectivement, la chaîne **‘voiture’** ne contient aucun des chiffres parmi ceux de l’ensemble du motif.

Autre exemple : **ereg(’[^aeiouy]’, ‘voiture’)**

Cet exemple renvoie FAUX puisque le mot **‘voiture’** contient au moins une des voyelles définies dans le motif.

Ensemble de caractères (III)

Il est possible de combiner ensembles et plages de caractères.

Exemple : **ereg(’[a-zA-Z]’, ‘voiture’)**

Cet exemple renvoie VRAI car la chaîne **‘voiture’** contient au moins un des caractères définis par le motif. Le motif défini tous les caractères minuscules entre **‘a’** et **‘z’** ainsi que tous les caractères majuscules entre **‘A’** et **‘Z’**.

Exemple : **ereg(’[^a-zA-Z]’, ‘voiture’)**

Ici on se demande si notre chaîne vérifie le motif suivant : ne pas trouver de lettres qu’elles soient minuscules ou majuscules. Cet exemple renvoie FAUX puisque **‘voiture’** contient des éléments du motifs : des caractères minuscules.

Classes de caractères (I)

Il existe des ensembles prédéfinis de caractères, chacun portant un nom particulier. Ainsi, l'ensemble des chiffres : **'[0-9]'** s'appelle **'[[:digit:]]'**.

Les exemples suivants sont équivalents :

```
ereg('0|1|2|3|4|5|6|7|8|9', $chaine)
ereg('[0-9]', $chaine)
ereg('[[:digit:]]', $chaine)
```

Les exemples suivants sont équivalents :

```
ereg('[^[:alnum:]]', $chaine)
ereg('[^[:alpha:][:digit:]]', $chaine)
ereg('[^A-Za-z0-9]', $chaine)
```

Les exemples suivants sont équivalents :

```
ereg('[[:alpha:]]', $chaine)
ereg('[[:upper:][:lower:]]', $chaine)
ereg('[A-Za-z]', $chaine)
```

Classes de caractères (II)

Séquence	Equivalent	Description
[[:alnum:]]	**[A-Za-z0-9]**	Caractères alphanumériques
[[:alpha:]]	**[A-Za-z]**	Caractères alphabétiques
[[:digit:]]	**[0-9]**	Caractères numériques
[[:blank:]]	**[\x09]**	Espaces ou tabulations
[[:xdigit:]]	**[0-9a-fA-F]**	Caractères hexadécimaux
[[:graph:]]	**[!-~]**	Caractères affichables et imprimables
[[:lower:]]	**[a-z]**	Caractères en minuscule
[[:upper:]]	**[A-Z]**	Caractères en majuscule
[[:punct:]]	**[!-/:-@[-'{-~]**	Caractères de ponctuation
[[:space:]]	**[\t\v\f]**	Tout type d'espace
[[:cntrl:]]	**[\x00-\x19\x7F]**	Caractères d'échappement
[[:print:]]	**[-~]**	Caractères imprimables, exceptés ceux de contrôle

Caractères spéciaux

Les caractères spéciaux sont ceux qui possèdent une signification particulière aux yeux des règles de construction des motifs des regex. Ces caractères ne peuvent pas être utilisés comme n'importe quel autre, sauf à le précéder d'un antislash ****.

Il sont les suivants : **^ . [ ] $ ( ) | * + ? { } **

Toutefois ces caractères (sauf **]** et **-**) perdent leur caractère spécial lorsqu'il sont utilisés entre crochets. Comme en C, pour déspécialiser un caractère, il faut le faire précéder d'un antislash ****. A noter qu'en dehors des crochets, le tiret **-** n'a pas signification particulière. Pour utiliser malgré tout les caractères **]** et **-** entre crochets, il faudra ruser, et les placer respectivement en début et en fin d'ensemble.

Exemple : **ereg('[][(){}]', $chaine)**

Étudions le comportement de PHP face à ce motif : il rencontre un premier crochet ouvrant qu'il considère comme spécial et précédant la définition d'un ensemble de caractères. Puis vient le crochet fermant, comme l'ensemble vide n'est pas connu par les regex PHP, il considère ce crochet comme n'importe lequel des caractères normaux. Ensuite vient le crochet ouvrant, comme le mode *ensemble* est déjà actif, il ne vas en ouvrir un autre et considère ce crochet comme un caractère normal. … Et vient enfin le dernier caractère – le crochet fermant – qui clos le mode *ensemble*.

Cardinalité

Un caractère ou un ensemble de caractères peut être interdit, facultatif, obligatoire ou répété un certain nombre de fois selon la syntaxe qui l'accompagne.

syntaxe	description
?	Facultatif : apparaît une ou zéro fois
*****	Facultatif : apparaît zéro, une ou plusieurs fois
+	Obligatoire : apparaît une ou plusieurs fois
{n}	Doit apparaître exactement **n** fois
{n,}	Doit apparaître au moins **n** fois
{n,m}	Doit apparaître entre **n** et **m** fois avec **n<m**

Exemple : **ereg('[[:lower:]]?[[:digit:]]{4}', 'la voiture K2000 est intelligente')**

Retourne VRAI car la sous chaîne **'K2000'** contient un caractère minuscule optionnel (**[[:lower:]]?**) suivi de quatre chiffres obligatoires (**[[:digit:]]{4}**).

Caractère

Pour chercher un caractère n'importe lequel (y compris les caractères de fonction) : **.** (point).

Exemple : **ereg('http://.+\.com', 'http://cyberzoide.developpez.com')**

Cet exemple retourne VRAI car **'http://cyberzoide.developpez.com'** commence par **'http://'** suivi de n'importe quel caractère **.** présent une ou plusieurs fois et finissant par **'.com'** (dont le point est despécialisé).

Position

On peut insérer dans le motif des contraintes de positions dans la chaîne : début et fin.

syntaxe	description
^	Début de chaîne
$	Fin de chaîne

Exemple : **ereg('^[[:upper:]].+\.$', 'Les Misérables.')**

Retourne VRAI car **'Les Misérables.'** commence par une majuscule et fini par un point (avec entre les deux un nombre indéfini de caractères).

Exemple : **ereg('^\$', '$5.000')**

Retourne VRAI car **'$5.000'** commence par le symbole de l'unité monétaire américaine, le dollars. Le caractère de position de fin de chaîne a été despécialisé par un antislash.

Exercice – format monétaire

Ex. 1 : Construire un motif permettant de vérifier la validité d'une chaîne comportant un prix en Euros. Le prix pourra comporter de 0 à 2 décimales. La virgule sera le séparateur de décimales. Les milliers (groupes de 3 chiffres) seront séparés par un point. Le prix se terminera par « EUR ». La chaîne ne devra rien comporter d'autre.

bon	**mauvais**
0 EUR	25, EUR
20,5 EUR	1 500 EUR
1.500 EUR	30.5 EUR
5.299.138.25 EUR	100,555 EUR
5.000,00 EUR	

Solution : **ereg('^[0-9]{1,3}(\.[0-9]{3})*(,[0-9]{0,2})? EUR$', $str)**

^xxx EUR$: la chaîne contient seulement le nombre xxx suivi d'un espace et de l'unité « EUR ».

(,[0-9]{0,2})? : le nombre contient optionnellement des décimales introduites par une virgule, le nombre de décimales varie de 0 à 2

(\.[0-9]{3})* : il y a 0 ou plusieurs groupes de milliers séparés par un point

[0-9]{1,3} : il y a 1 ou 3 chiffres au minimum dans notre nombre

Remplacement

Les expressions ne se limitent pas à la recherche de motifs mais permettent aussi de remplacer les sous-chaînes satisfaisant un motif par une autre chaîne via la fonction **ereg_replace()**.

Exemple 1 :

```
$str = ''Je roule en voiture.'';
$str = ereg_replace('voiture', 'automobile', $str);
echo $str ;          // affiche : ''Je roule en automobile.''
```

Exemple 2 :

```
$str = ''cyberzoide@yahoo.fr'';
$str = ereg_replace('@(.+)\.fr', 'wanadoo', $str);
echo $str ;          // affiche : ''cyberzoide@wanadoo.fr''
```

Fractionnement

Il est possible de fractionner une chaîne en plusieurs sous-chaînes séparées par un délimiteurs satisfaisant un motif, en utilisant la fonction **split()**.

Exemple 1 :

```
$str = ''Hugo:Etiévant:cyberzoide@yahoo.fr'';          // affiche :
$tab = split(':'', $str, 3);                           Hugo
foreach($tab as $elem) {                               Etiévant
        echo $elem, ''<br />'';                        cyberzoide@yahoo.fr
}
```

Cet exemple sépare les 3 premières sous-chaînes délimitées par le caractère deux points comme cela se fait pour l'analyse d'une ligne du fichier **.passwd**.

Exemple 2 :

```
$str = ''23-03-2003'';
list($jour, $moi, $an) = split(''[./-]'', $str);
```

Cet exemple sépare les sous-chaînes d'une date dont le séparateur peut être l'un de la plage suivante : point, slash et tiret.

Parenthèses capturantes

Un autre atout des regex est de pouvoir capturer la sous-chaîne satisfaisant le motif afin de l'inclure dans une chaîne de remplacement. Pour cela, on va employer les parenthèses afin d'encadrer la sous-chaîne du motif qu'il conviendra de capturer (que l'on appellera « instance »). Puis, on fera référence à cette instance du motif via la syntaxe suivante : « \\X » où X est le numéro de la parenthèse que l'on souhaite capturer. Car il est possible de capturer 9 instances. L'instance \\0 faisant référence à la chaîne en entier et \\1 à la première parenthèse capturante.

Exemple :

```
$date = "21-02-2003";
$date = ereg_replace("([[:digit:]]{2})-([[:digit:]]{2})-([[:digit:]]{4})",
"\\2/\\1/\\3", $date);                    // affiche 02/21/2003
```

Cet exemple converti une date MySQL au format francophone.

Le même résultat aurait pu être obtenu par l'utilisation successive des fonctions : **split()**, **list()** et **ereg_replace()**. Mais grâce aux parenthèses capturantes, on a tout fait en une seule commande !

Autres fonctions de traitement de chaînes

Les expressions régulières sont un outil puissant pour traiter des chaînes de caractère dont on connaît le schéma, c'est-à-dire la manière générale dont elles sont « grammaticalement » composées.

Malgré leurs qualités, elles souffre d'un défaut majeur : la lenteur d'exécution du moteur de traitement de ces chaînes.

Ainsi, il est recommandé d'utiliser le plus souvent possible les fonctions standard de traitement des chaînes de caractères afin d'accélérer le temps de traitement de vos scripts.

On peut organiser ces fonctions en 3 classes :

- *recherche, comparaison* (similar_text, strcmp, strnatcmp, strcasecmp, strncmp, substr, strstr, strspn, strpos)

- *remplacement* (AddSlashes, AddCSlashes, htmlentities, htmlspecialchars, QuoteMeta, trim, nl2br, strip_tags, StripSlashes, str_pad, str_repeat, str_replace, strtr, ucfirst, ucword, wordwrap)

- *fractionnement* (explode, implode, join, chunk_split, strtok)

Tableau comparatif

Certaines fonctions standards sont équivalentes à une expression régulière, il faudra alors privilégier ces premières :

Fonction standard	Fonction regex
nl2br($str)	ereg_replace(‘\n’, ‘ ’,$str)
ltrim($str)	ereg_replace(‘^( )+’,’’, $str)
strcmp($str1, $str2)	ereg(‘’^$str1$’’, $str2);
strip_tags($str)	ereg_replace(‘<.+>’, ‘’, $str)
stripslashes($str)	ereg_replace(‘\[^\]’, ‘’, $str)
strtok($str, $op)	split($op, $str)
strtr($str, ’a’, ‘@’)	ereg_replace(‘a’, ‘@’, $str)

Les fonctions standards (I)

strcmp($str1, $str2) : compare en binaire les 2 chaînes, retourne un entier négatif si **$str1**<**$str2**, positif si **$str1**>**$str2**, nul si **$str1**=**$str2**.

strncmp($str1, $str2, $i) : comme **strcmp()** mais sur les $i premiers caractères.

strcasecmp($str1, $str2) : comme **strcmp()** mais insensible à la case.

strncasecmp($str1, $str2) : comme **strncmp()** mais insensible à la case.

strnatcmp($str1, $str2) : comme **strcmp()** mais dans l'ordre « naturel ».

strnatcasecmp($str1, $str2) : comme **strcasecmp()** mais dans l'ordre « naturel ».

strstr($str1, $str2) : retourne le contenu de **$str1** depuis la première occurrence de **$str2** jusqu'à la fin.

stristr($str1, $str2) : comme **strstr()** mais insensible à la casse.

strrchr($str1, $str2) : comme **strstr()** mais à partir de la dernière occurrence.

substr($str, $i [, $n]) : retourne la sous-chaîne de **$str** débutant à la position **$i** jusqu'à **$n**.

Les fonctions standards (II)

addslashes($str) : retourne la chaîne **$str** dont les caractères **‘**, **’’** et **** sont protégés par un antislash.

stripslashes($str) : fonction réciproque de addslashes.

quotemeta($str) : ajoute un antislash devant les caractères suivants : **. \\ + * ? [^] ($)**.

htmlspecialchars($str) : convertit tous les caractères spéciaux en leur code HTML, par exemple **<** devient **<** Synonyme : **htmlentities()**.

ltrim($str) : supprime les espaces de début de chaîne

rtrim($str) : supprime les espaces de fin de chaîne, synonyme : **chop()**

trim($str) : **ltrim()** + **rtrim()**

explode($op, $str [, $n]) : scinde la chaîne **$str** en au plus **$n** morceaux en utilisant le séparateur **$op**. Retourne un tableau.

implode($op, $tab) : fonction réciproque de **explode()**.

strtok([$str,] $op) : morcelle la chaîne **$str** via le séparateur **$op**.

chunk_split($str1 [, $n [, $str2]]) : morcelle **$str1** par insertion de **$str2** tous les **$n** caractères.

Les fonctions standards (III)

ord($str) : retourne la valeur ASCII du caractère

chr($str) : fonction réciproque de **ord()**

strlen($str) : retourne la taille de $str (i.e. le nombre de caractères)

str_pad($str1, $n [, $str2 [, $type]]) : complète la chaîne **$str1** avec **$n** fois **$str2** (ou **$n** espaces par défaut) en fin de chaîne par défaut ou en début ($type = STR_PAD_LEFT) ou encore sur les deux côtés (STR_PAD_BOTH).

str_repeat($str, $n) : répète **$n** fois la chaîne **$str**.

strrev($str) : inverse une chaîne

strpos($str1, $str2 [, $n]) : recherche la première occurrence de **$str2** dans la chaîne **$str1** à partir de la position **$n** (au début par défaut).

strrpos($str1, $str2) : comme **strpos()** mais un caractère à partir de la fin

substr_count($str1, $str2) : compte le nombre d'occurrences de **$str2** dans **$str1**

Les fonctions standards (IV)

substr_replace($str1, $str2, $i [, $n]) : remplace **$n** caractères depuis **$i** dans **$str1** par **$str2**

strtolower($str) : conversion en minuscules

strtoupper($str) : conversion en majuscules

ucfirst($str) : conversion en majuscule du premier caractère de la chaîne

ucwords($str) : conversion en majuscule du premier caractère de chaque mot

wordwrap($str [, $n [, $op [, $test]]]) : ajoute la césure **$op** dans **$str** tous les **$n** caractères. Si **$test** vaut **1**, les mots trop long seront coupés. Par défaut découpe tous les **75** caractères par **\n**.

nl2br($str) : remplace les sauts de ligne **\n** par la balise XHTML **
**.

Historique

- **23 février 2003** : parenthèses capturantes, fonctions standards (28 diapos)
- **23 décembre 2002** : création du document (19 diapos)

Agissez sur la qualité de ce document en envoyant vos critiques et suggestions à l'auteur.

Pour toute question technique, se reporter au forum PHP de Developpez.com

Hugo Etiévant
cyberzoide@yahoo.fr
http://cyberzoide.developpez.com/