PHP - Les expressions régulières

Date de dernière mise à jour : 27/06/2007 à 19:36

Présentation

Les expressions régulières sont souvent utilisées en PHP quand on doit faire des traitements conditionnels très poussés sur les chaînes de caractères. Ce chapitre est malheureusement compliqué à comprendre, et nous aborderons plein d'exemples simples qui je l'espère vous permettront de faire des choses plus compliquées ensuite.

Une expression régulière (aussi appelée regex) va par exemple vous permettre de savoir si une chaîne est composée de 6 caractères numériques, de savoir si elle commence par un "a" tout en finissant par un "w" ou encore de savoir si il s'agit d'une date valide, etc ...

Toutes ces conditions que l'on fixe se présentent sous la forme d'une chaîne de caractères qui peut être immense en fonction de ce que l'on souhaite savoir ou récupérer.

Les fonctions PCRE

En PHP, il existe deux "langages" pour les expressions régulières. Ces deux langages sont le POSIX et le PCRE. Nous étudierons les fonctions utilisant le langage PCRE, plus performant que le POSIX (que ce soit en terme de puissance ou de rapidité de traitement). Les fonctions utilisant le PCRE commencent généralement par le préfixe "preg_".

Voici une liste des fonctions PCRE que nous serons amenées à utiliser (il en existe d'autres) :

-preg_match_all
-preg_match
-preg_replace
-preg_split

Ces fonctions s'utilisent généralement de la manière suivante (pour les fonctions de remplacement) :

preg_X('masque', 'chaine de remplacement', 'chaine de caractères dans laquelle on appliquera le masque');

Les fonctions de comparaison sont plutôt de ce style là :

preg_Y('masque', 'chaine de caractères dans laquelle on appliquera le masque');

La chaîne "masque" correspond à une expression régulière. Nous allons voir plus bas comment créer des expressions régulières. Avant toute chose, il faut savoir qu'une regex est toujours délimitée par deux caractères identiques appelés délimiteurs. Ces délimiteurs sont importants car ils vont vous permettre d'ajouter des options à votre expression régulière (comme par exemple l'insensibilité aux majuscules/minuscules).

Voici l'exemple général d'utilisation d'une regex :`**masque`options**
Ce qui peut donner par exemple :`**[a-z]*`i**
Ici, vous ne le savez pas encore mais ceci désigne une chaîne composée de lettres uniquement, majuscules ou minuscules (le i dans l'option signifie que la recherche est insensible aux minuscules ou majuscules)

Recherche d'une chaîne - bases

La recherche d'une chaîne de caractères simple peut se faire avec la fonction strpos() de PHP, qui sera bien plus rapide qu'une expression régulière. Toutefois, ses fonctionnalités sont très limitées car elle ne peut disposer en paramètre que d'une chaîne fixe. Avec une expression régulière vous allez pouvoir imposer des conditions performantes sur la recherche.

Pour rechercher une chaîne suivant des conditions précises, on utilise la fonction preg_match(). Elle s'utilise comme ceci :

```
<?php     $chaine
=
'Un exemple de chaîne simple'
;
    if
(
preg_match
(
'`exemple`'
,
$chaine
))

{
        echo
'Vrai, la chaîne correspond au masque'
;

}    else    {         echo
'Faux, la chaîne ne correspond pas au masque'
;    }
?>
```

Ici, la chaîne correspond au masque car nous cherchons la chaîne "exemple" qui se situe bien dans la phrase "Un exemple de chaîne simple". Bon, vous vous imaginez que les fonctions de preg_match() ne se limitent pas à ça, sinon utiliser strpos() ...

Voyons d'autres cas de figure :

```
<?php     $chaine
=
'Un exemple de chaîne simple'
;
    if
(
preg_match
(
'`Exemple`'
,
$chaine
))

{
        echo
'Vrai, la chaîne correspond au masque'
;

}    else    {
echo

'Faux, la chaîne ne correspond pas au masque'
;    }
```

```
?>
```

Ici, vous aurez un résultat faux car le E de exemple est en majuscule. Pour remédier à ce problème, on utilise l'option "i" qui signifie que le masque ne tient pas compte des minuscules et majuscules. Les options se placent après le deuxième délimiteur (ici le caractère `) :

```
<?php     $chaine
=
'Un exemple de chaîne simple'
;
    if
(
preg_match
(
'`Exemple`i'
,
$chaine
))

{
        echo
'Vrai, la chaîne correspond au masque'
;

}    else    {
echo

'Faux, la chaîne ne correspond pas au masque'
;    }
?>
```

Ici, preg_match() renvoie TRUE. Voyons d'autres cas :

Chaîne	Regex	Résultat
'Un exemple de chaîne simple'	`exemple`	Vrai (le mot "exemple" se trouve bien dans la chaîne)
'Un exemple de chaîne simple'	`compliquée`	Faux (le mot "compliquée" ne se trouvant pas dans la chaîne)
'Un exemple de chaîne simple'	`Exemple`	Faux car le E est en majuscule dans la regex et pas dans la chaîne

'Un exemple de chaîne simple'
`Exemple`i
Vrai car le i rend insensible à la casse (majuscules/minuscules) la regex

Recherche d'une chaîne : condition "ou"
Le symbole "ou" est désigné en PCRE par une seule barre verticale (contrairement au PHP qui lui en demande deux dans les conditions habituelles).

Mettons que nous recherchions dans une chaine le mot "site" ou le mot "web". La regex pourra s'écrire comme ceci :

`site|web`

Chaîne
Regex
Résultat

'Bienvenue sur mon site web'
`site|web`
Vrai (on a bien soit site soit web dans la chaîne)

'BIENVENUE SUR MON SITE WEB'
`site|web`
Faux (aucun des deux mots n'est en minuscules)

'BIENVENUE SUR MON SITE WEB'
`site|web`i
Vrai, on a rendu l'expression régulière insensible à la casse

'Bienvenue sur mon site'
`site|web`
Vrai car le mot "site" figure dans la chaîne à analyser

'Bienvenue sur mon espace web'
`site|web`
Vrai car le mot "web" figure dans la chaîne à analyser

'Bienvenue sur mon espace'
`site|web`
Faux car ni le mot "site" ni le mot "web" ne figurent dans la chaîne

Le début et la fin d'une chaîne
Bon, nous allons maintenant voir quelque chose d'un peu plus précis. Il s'agit du début et de la fin d'une chaîne. Par exemple nous allons pouvoir dire "est-ce que cette chaîne commence par la lettre A majuscule et se termine par un b minuscule ?"

Les deux symboles utilisés pour désigner cela sont ceux-ci :

-^ qui désigne le début d'une chaîne.
-$ qui désigne la fin d'une chaîne.

Chaîne

Regex
Résultat

'Anthony'
`^A`
Vrai (la chaîne commence bien par un A)

'Anthony'
`y$`
Vrai (la chaîne se termine par un y)

'Début de phrase'
`^Dt$`
Faux car la chaîne est différente de "Dt"

'Bienvenue sur mon espace web'
`^Bienvenue sur mon espace web$`
Vrai car la chaîne est identique des deux côtés

'Bienvenue sur mon espace web'
`^Bienvenue web$`
Faux car il y a des mots entre le mot "Bienvenue" et le mot "Web"

Les quantificateurs
Les quantificateurs sont très utiles, grâce à eux vous allez pouvoir dire "je veux savoir si cette chaîne commence par 7 fois la lettre A", ou encore "il me faut un A, trois o et deux z".

Il faut tout d'abord retenir trois symboles :

-? qui désigne "0 ou une occurrence". Si on prend par exemple le masque **`v?`**-, la châine 'v' sera valide, 'vv' ne sera pas valide et '' sera valide
-+ qui désigne "1 ou plusieurs occurrences". Si on prend par exemple le masque **`v+`**- la chaîne 'v' sera valide, 'vv' le sera aussi, 'vvvvvv' aussi mais par contre '' ne sera pas valide.
-* qui désigne "0, 1 ou plusieurs occurrences". Si on prend par exemple le masque **`v+`**-, la chaîne 'v' sera valide, 'vv' le sera aussi, 'vvvvvv' aussi ainsi que ''

Ces symboles s'appliquent uniquement à la lettre (ou au mot si vous placez des parenthèses) qui est situé(e) à gauche du symbole.

Chaîne
Regex
Résultat

'Anthony'
`An?thony`
Vrai (Il y a bien 0 ou 1 "n" après le A)

'Annnnnnthony'
`An?thony`
Faux (il y a plus de 1 "n" après le A)

'Annnnnnthony'

`An+thony`
Vrai (il y a bien au moins 1 "n" après le A)

'Chat'
`Chats?$`
Vrai car le mot "chat" peut ici aussi être au pluriel (0 ou 1 "s"), ensuite on marque la fin de la chaîne

'Chat'
`Chats*$`
Vrai car * signifie "0", "1" ou "plusieurs" ici il n'y a pas de s, la condition reste quand même vraie.

Il existe un autre moyen de quantifier, cette fois plus précisément, une chaîne ou une lettre, ou encore une classe de caractères (on verra ce que c'est juste après). On utilise pour cela les accolades. Voici comment ça fonctionne :

-{X} : la chaîne doit être répétée X fois. Par exemple : **`A{2}`**- fonctionnera pour AA mais pas pour AAA ni pour A.
-{X,Y} : la chaîne peut être répétée de X fois à Y fois. Par exemple : **`A{2,4}`**- fonctionnera pour AA, pour AAA et pour AAAA mais pas pour A ni pour AAAAA.
-{X,} : la chaîne doit être répétée au moins X fois mais il n'y a pas de limite de nombre. Par exemple : **`A{2,}`**- fonctionnera pour AA, pour AAA, pour AAAA... mais pas pour A.

Chaîne
Regex
Résultat

'Anthony'
`An{0,1}thony`
Vrai (Il y a bien 0 ou 1 "n" après le A)

'Annnnnnthony'
`An{2}thony`
Faux (il n'y a pas deux "n" après le A)

'Annnnnnthony'
`An{2,}thony`
Vrai (il y a bien au moins 2 "n" après le A)

Les classes de caractères
Les classes de caractères vont vous permettre d'effectuer des recherches précises tout en limitant le nombre de caractères de vos expressions régulières. Les classes de caractères sont toujours contenues entre des crochets simples ou doubles (nous verrons certaines classes de caractères spéciales après).

Voici un exemple d'expression régulière incluant une classe de caractères : **`[nv]ous`**

le **[nv]** signifie "n" OU "v". Les mots "nous" et "vous" seront donc acceptés, il s'agit donc de l'équivalent de **`(n|v)ous`**

Chaîne
Regex
Résultat

'Nous'
`[nv]ous`i
Vrai (le i rend la regex insensible à la casse)

'Nous'
`[vs]$`
Vrai (la chaîne se termine bien par un v ou un s)

Voyons maintenant quelques classes un peu spéciales, elles ont de spécial le fait que l'on utilise un tiret entre plusieurs valeurs, il s'agit donc d'intervalles de caractères. Nous allons voir ici quelques exemples rapides, car je pense que vous allez vite comprendre l'intérêt de cette syntaxe :

`[a-z]` : indique une lettre minuscule allant de a à z

`[0-3]` : indique un chiffre allant de 0 à 3

`[a-z0-9]` : indique des lettres minuscules OU des chiffres allant de 0 à 9

Chaîne
Regex
Résultat

'Anthony'
`[a-z]*`
Faux (le premier A est en majuscules)

'Annnnnnthony'
`[a-z]*`i
Vrai (les caractères sont tous des lettres minuscules ou majuscules)

'Annnnnnthony1'
`[0-9]$`
Vrai (la chaîne se termine bien par un chiffre allant de 0 à 9)

Vous pouvez également interdire un ou une plage de caractères, cette fois-ci on utilise le symbole ^ placé juste après le premier crochet ouvrant. Il ne signifie plus "début de la chaîne" mais une interdiction.

Chaîne
Regex
Résultat

'ANTHONY'
`^[^a-z]*$`
Vrai (la chaîne ne contient pas de minuscules)

'Anthony'
`^[^a-z]*$`
Faux (la chaîne contient des minuscules)

Voici maintenant quelques classes encore spéciales, puisqu'elles sont contenues entre des doubles crochets. Ce sont des classes prédéfinies qui vont vous simplifier la syntaxe de vos regex :

-[[:alpha:]] : n'importe quelle lettre
-[[:lower:]] : n'importe quelle lettre en minuscule
-[[:upper:]] : n'importe quelle lettre en majuscule
-[[:alnum:]] : n'importe quelle lettre ou chiffre
-[[:digit:]] : n'importe quel chiffre
-[[:punct:]] : n'importe quel signe de ponctuation
-[[:space:]] : n'importe quel espace blanc
-[[:blank:]] : espace ou tabulation
-[[:graph:]] : caractères affichables et imprimables
-[[:cntrl:]] : caractères d'échappement
-[[:xdigit:]] : caractères héxadécimaux
-[[:print:]] : scaractères imprimables exceptés ceux de contrôle

Les métacaractères
Les métacaractères sont des caractères spéciaux effectuant des actions bien précises sur les chaînes. Voici la liste de ces caractères :

[ ] ! ( ) { } ^ $? . + * \ #

Le problème qu'ils génèrent est lorsque vous souhaitez les utiliser dans vos recherches ou remplacements de caractères. Il faut les faire précéder d'un antislash \. Voici un exemple :

Chaîne
Regex
Résultat

'ça va?'
`ça va?$`
Faux (la chaîne ne se termine pas par 0 ou 1 "a")

'ça va?'
`ça va\?$`
Vrai (on a échappé correctement le caractère)

Attention, cette règle ne s'applique pas à l'intérieur des classes de caractères. Ainsi, la classe [ab+?] signifie que vous pourrez avoir la lettre "a" ou "b" ou "+" ou "?".

Les classes abrégées
Les classes abrégées vont vous permettre de réduire encore la longueur de vos expressions régulières au mépris de leur compréhension directe. On utilise pour cela un antislash suivi d'un caractère, qui à eux deux signifient une classe de caractère plus complexe. Quand on connaît la signification des lettres par coeur, c'est assez facile de ne pas se tromper, mais sinon la confusion peut régner, voilà pourquoi personnellement je préfère utiliser les classes de caractères simples. Voici la liste des classes de caractères abrégées :

-. signifie que vous pouvez mettre n'importe quel caractère.
-\w indique les mots [_a-zA-Z0-9]
-\W indique ce qui ne correspond pas à un mot [^_a-zA-Z0-9]
-\d indique que vous souhaitez un chiffre [0-9]
-\D indique les caractères n'étant pas des chiffres [^0-9]
-\s correspond à un espace (correspond à \t \n \r)
-\S correspond à ce qui n'est pas un espace (\t \n \r)
-\t correspond à une tabulation
-\n correspond à un saut de ligne

-\r correspond à un retour chariot

Chapitre suivant

PHP et les images
Présentation
Activer la librairie GD
Principe de la création d'une image
Créer une image basique
Envoi de l'image au navigateur ou sauvegarde sur le disque
Les couleurs
Ecrire du texte
Les formes (rectangle, ligne ...)
Effets sur les images (rotations, transparence ...)