

PHP – Obsługa wyrażeń regularnych, filtry

Beata Pańczyk - PHP (Wykład 3)

Plan wykładu

- Rozszerzenie POSIX
- Rozszerzenie PCRE
- Różnice składni
- Funkcje PCRE
- Filtry typu VALIDATE i SANITIZE
- Przykłady walidacji danych za pomocą filtrów

POSIX Regular Expressions

- PHP obsługuje wyrażenia regularne w stylu POSIX za pomocą funkcji `ereg_replace`, `ereg`, `eregi_replace`, `eregi`, `split`, `spliti`
- Funkcje te nie są jednak bezpieczne w przeciwieństwie do zalecanych obecnie funkcji PCRE

PCRE Regular Expressions (Perl-Compatible)

- Od wersji PHP 5.3.0 rozszerzenie POSIX RegExp jest usunięte
- Istnieją różnice w sposobie definiowania i pracy z wyrażeniami regularnymi w stylu POSIX regex i PCRE regex

POSIX a PCRE

- Podstawowe różnice definiowania wyrażeń w stylu PCRE w stosunku do POSIX to:
 - funkcje PCRE wymagają aby wyrażenie regularne było ujęte w specjalne ograniczniki (delimiters)
 - w przeciwieństwie do POSIX, rozszerzenie PCRE nie posiada dedykowanych funkcji nie uwzględniających wielkości liter – zamiast tego należy używać modyfikatora *i* (*PCRE_CASELESS*) (dostępne są również inne modyfikatory zmian w wyszukiwaniu)
 - funkcje POSIX znajdują najdłuższy (od lewej) łańcuch pasujący do wzorca podczas gdy funkcje PCRE przerywają szukanie po znalezieniu pierwszego łańcucha pasującego do wzorca

Przykładowe modyfikatory

- *i* (*PCRE_CASELESS*) – brak rozróżniania dużych i małych liter
- *m* (*PCRE_MULTILINE*) – domyślnie PCRE traktuje rozważany łańcuch jako jeden wiersz (nawet jeśli składa się z kilku); przy ustawionym modyfikatorze, jeśli w łańcuchu nie występuje znak "\n" w rozważanym ciągu lub nie występują znaki ^ lub \$ we wzorcu, ustawienie tego modyfikatora nie ma znaczenia
- *s* (*PCRE_DOTALL*) – przy ustawionym modyfikatorze, kropka we wzorcu pasuje do wszystkich znaków, łącznie ze znakiem nowej linii (normalnie jest on wykluczony)
- *x* (*PCRE_EXTENDED*) – przy ustawionym modyfikatorze, białe znaki we wzorcu są ignorowane z pewnymi wyjątkami

Ograniczniki (delimiters)

- Delimiter może być dowolnym nie alfanumerycznym, nie białym znakiem i nie znakiem \
- Najczęściej stosowane są znaki: / # ~ @ ! <>
- Np.:
 - /kot/
 - #^[^0-9]\$#
 - +php+
 - %[a-zA-Z0-9_-]%
- Jeśli ogranicznik ma wystąpić jako znak we wzorcu musi być poprzedzony \ (dobrym zwyczajem jest wtedy wybór innego znaku ograniczającego), np.:
 - /http:\\\\/
 - #http://#
- Po końcowym ograniczniku można dodać modyfikator np.:
 - #[a-z]#i

Funkcje POSIX a PCRE

POSIX	PCRE
ereg_replace()	preg_replace()
ereg()	preg_match()
eregi_replace()	preg_replace()
eregi()	preg_match()
split()	preg_split()
spliti()	preg_split()
sql_regcase()	<i>No equivalent</i>

Funkcje PCRE

- `preg_filter` — wyszukuje wyrażenie i zastępuje podanym
- `preg_grep` — zwraca tablicę ciągów pasujących do wzorca
- `reg_last_error` — zwraca kod błędu ostatniego wykonania funkcji PCRE
- `preg_match_all` — globalne dopasowanie wyrażenia regularnego
- `preg_match` — dopasowanie wyrażenia regularnego
- `preg_quote` — cytuje znaki wyrażenia regularnego
- `preg_replace_callback` — wyszukuje wyrażenie i zastępuje
- `preg_replace` — wyszukuje RegExp i zamienia podanym
- `preg_split` — dzieli ciąg względem RegExp

Funkcja preg_replace

- mixed **preg_replace** (mixed \$pattern , mixed \$replacement , mixed \$subject [, int \$limit = -1 [, int &\$count]])

gdzie:

- \$pattern - łańcuch lub tablica łańcuchów
- \$replacement - łańcuch lub tablica łańcuchów do zamiany
- \$subject - łańcuch lub tablica łańcuchów do wyszukania i zamiany
- \$limit - maksymalna liczba zamian (domyślnie -1 b.o.) w każdym łańcuchu
- \$count - jeśli podana - będzie zawierać liczbę zamian

Przykład 1 – preg_replace

```
■ <?php
    $ciag = 'Ala ma kota i Ala ma psa';
    $wzor1 = '/A/';
    $wzor2 = '/A/i';
    $zamiana = 'U';
    echo preg_replace($wzor1, $zamiana, $ciag);
    echo '<br />';
    echo preg_replace($wzor2, $zamiana, $ciag);
?>
```

Ula ma kota i Ula ma psa
UIU mU kotU i UIU mU psU

Przykład 2 – preg_replace

```
<?php
    $ciag = 'Szybki, rudy lis.';
    echo $ciag.'<br />';
    $wzor = array();
    $wzor[0] = '/szybki/i';
    $wzor[1] = '/rudy/';
    $wzor[2] = '/lis/';
    $zamiana = array();
    $zamiana[0] = 'Powolny';
    $zamiana[1] = 'bury';
    $zamiana[2] = 'niedźwiedź';
    echo preg_replace($wzor, $zamiana, $ciag);
?>
```

Szybki, rudy lis.
Powolny, bury niedźwiedź.

Funkcja preg_match

- `int preg_match (string $pattern , string $subject [, array &$matches [, int $flags = 0 [, int $offset = 0]]])`

gdzie:

- `$pattern`, `$subject` - jak poprzednio
- `$matches` - tablica z wynikami wyszukiwania:
`$matches[0]` zawiera tekst wzorca, `$matches[1]` zawiera pierwszy pasujący tekst itp.
- flagą może być **PREG_OFFSET_CAPTURE**
 - Przy jej ustawieniu, każdy pasujący podciąg będzie umieszczony w tablicy `$matches` (każdy jej element będzie tablicą składającą się z pasujących podciągów z odpowiednim offsetem)
- `$offset` - normalnie wyszukiwanie zaczyna się od początku ciągu, opcjonalnie można podać inną pozycję startu (w bajtach)

Funkcja preg_match

- **preg_match()** - zwraca liczbę **znalezionych dopasowań**:
 - albo 0 (0 dopasowań)
 - albo 1 (**preg_match()** kończy szukanie po pierwszym znalezieniu ciągu pasującego do wzorca - w przeciwieństwie do funkcji **preg_match_all()**, która przeszukuje ciąg do końca)
 - zwraca **FALSE** w przypadku wystąpienia błędu

Przykład 3 – funkcja preg_match

```
<?php
// i - a case-insensitive search
if (preg_match("/^(W3C)$/i", "Skrót W3C")) {
    echo "<br />1. Wzór został znaleziony.";
} else {
    echo "<br />1. Wzór nie został znaleziony.";
}
if (preg_match("/(W3C)$/i", "Skrót W3C")) {
    echo "<br />2. Wzór został znaleziony.";
} else {
    echo "<br />2. Wzór nie został znaleziony.";
}
if (preg_match("/^(W3C)$/i", "W3C")) {
    echo "<br />3. Wzór został znaleziony.";
} else {
    echo "<br />3. Wzór nie został znaleziony.";
}
?>
```

1. Wzór nie został znaleziony.
2. Wzór został znaleziony.
3. Wzór został znaleziony.

Przykład 4 – walidacja adresu e-mail

```
<?php
if (isset($_GET['submit'])) //jeśli naciśnięto przycisk submit
{ if (!preg_match("/^[a-zA-Z0-9_]+@[a-zA-Z0-9\-.]+\.[a-zA-Z0-9\-.]+$/", $_GET['mail']))
    { echo "<h2> Nie poprawny adres e-mail </h2><br/>";
      echo "Spróbuj jeszcze raz: <a href='reg1.php'>Wstecz</a>" ; }
    else echo "<h2> Dziękujemy za e-mail!</h2>"; }
else //jeśli nie to wyświetl formularz
{ ?>
    <h4>Podaj e-mail :</h4>
    <form method="get" action="reg1.php">
    <input name="mail" width="25"/><br /><br />
    <input type="submit" value="Wyślij" name="submit"/>
    <input type="reset" value="Anuluj"/>
<?php }
?>
```


Przykład 4 - wynik

Podaj e-mail :

**Nie poprawny
adres e-mail**

Spróbuj jeszcze raz:

[Wstecz](#)

**Dziękujemy za
e-mail!**

filter_input()

- Funkcja pobiera zmienne zewnętrzne (np. z formularzy) i opcjonalnie je filtruje
- Składnia:
filter_input(typ, nazwa, filtr, opcje)
gdzie:
 - typ - typ we np. INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER
 - nazwa - nazwa zmiennej do sprawdzenia
 - filtr - opcjonalnie rodzaj filtra (domyślnie FILTER_DEFAULT – brak filtrowania)
 - opcje – opcjonalnie, flagi/opcje zależne od rodzaju filtra
- Wynik - przefiltrowane dane lub false

Przykładowe filtry typu VALIDATE

- FILTER_VALIDATE_BOOLEAN
- FILTER_VALIDATE_EMAIL
- FILTER_VALIDATE_FLOAT
- FILTER_VALIDATE_INT
- FILTER_VALIDATE_IP
- FILTER_VALIDATE_REGEXP
- FILTER_VALIDATE_URL
- Więcej na stronie:
http://www.w3schools.com/php/php_ref_filter.asp

Przykład 4a – Walidacja adresu e-mail za pomocą filtra

```
<?php
if (filter_input(INPUT_GET, "submit"))
{ //jeśli naciśnięto submit
    if (!filter_input(INPUT_GET, "mail", FILTER_VALIDATE_EMAIL))
    {
        echo "<h2> Nie poprawny adres e-mail </h2><br/>";
        echo "Spróbuj jeszcze raz: <a href='reg1.php'>Wstecz</a>";
    } else
        echo "<h2> Dziękujemy za e-mail!</h2>";
    }
else ...
```

Inne przydatne funkcje filtrujące

- `filter_var` — filtruje zmienne zgodnie ze wskazanym rodzajem filtra ¶
- `filter_input_array` — pobiera dane zewnętrzne i opcjonalnie je filtruje
- `filter_var_array` — pobiera tablicę zmiennych i opcjonalnie je filtruje ¶

Przykładowe filtry typu SANITIZE

- `FILTER_SANITIZE_EMAIL` – usuwa wszystkie znaki za wyjątkiem liter, cyfr oraz znaków: `!#$%&'*+ -= ? ^_`{|}~@.[]`.
- `FILTER_SANITIZE_ENCODED` – usuwa lub koduje znaki specjalne (zgodnie z zadana flagą)
- `FILTER_SANITIZE_MAGIC_QUOTES` – tak jak `addslashes()`
- `FILTER_SANITIZE_NUMBER_INT` – usuwa wszystkie znaki poza cyframi, oraz znakami plus i minus.
- Więcej na stronie:
<http://php.net/manual/en/filter.filters.sanitize.php>

FILTER_VALIDATE_REGEX P

```
$str = "Ala ma kota i Ala ma psa";  
$opt = array( "options" => array("regexp" => "/Ale/") );  
if (filter_var($str, FILTER_VALIDATE_REGEXP, $opt ))  
    echo "Pasuje";  
else echo "Nie pasuje";  
  
$email = filter_input(INPUT_GET, "mail");  
if (!filter_var($email, FILTER_VALIDATE_EMAIL) === false) {  
    echo("$email poprawny");  
} else {  
    echo("$email nie poprawny");  
}
```

filter_input_array (1)

```
<?php
error_reporting(E_ALL | E_STRICT);
/* Przykładowe dane przesłane metodą POST:
$_POST = array(
    'product_id'    => 'libgd<script>',
    'component'     => '10',
    'versions'      => '2.0.33',
    'testscalar'    => array('2', '23', '10', '12'),
    'testarray'     => '2'
);
*/
```


filter_input_array (2)

//przygotowanie argumentów do walidacji:

```
$args = array(
    'product_id' => FILTER_SANITIZE_ENCODED,
    'component'  => array('filter' => FILTER_VALIDATE_INT,
                          'flags'  => FILTER_REQUIRE_ARRAY,
                          'options' => array('min_range' => 1,
                                             'max_range' => 10)    ),
    'versions'   => FILTER_SANITIZE_ENCODED,
    'doesnotexist' => FILTER_VALIDATE_INT,
    'testscalar' => array( 'filter' => FILTER_VALIDATE_INT,
                          'flags' => FILTER_REQUIRE_SCALAR ),
    'testarray'  => array( 'filter' => FILTER_VALIDATE_INT,
                          'flags' => FILTER_REQUIRE_ARRAY )
);
```

filter_input_array (2) - Wynik

```
$myinputs = filter_input_array(INPUT_POST, $args);  
var_dump($myinputs);  
echo "<br/>";;  
?>  
//Wynik:  
array(6) {  
    ["product_id"]=> string(17) "libgd%3Cscript%3E"  
    ["component"]=> array(1) {  [0]=>  int(10) }  
    ["versions"]=> string(6) "2.0.33"  
    ["doesnotexist"]=> NULL  
    ["testscalar"]=> bool(false)  
    ["testarray"]=> array(1) {  [0]=>  int(2) }  
}
```

filter_var_array

```
<?php
error_reporting(E_ALL | E_STRICT);
$data = array(
    'product_id' => 'libgd<script>',
    'component'  => '10',
    'versions'   => '2.0.33',
    'testscalar' => array('2', '23', '10', '12'),
    'testarray'  => '2'
);
// $args jak w przykładzie dla input_filter_array
$myinputs = filter_var_array($data, $args);
var_dump($myinputs);
echo "<br/>";
?>
```