PHP 5 TABLICE INFORMATYCZNE

Marcin Lis

Helion

UMIESZCZANIE SKRYPTÓW W KODZIE HTML

Skrypty PHP są najczęściej wykonywane przez serwer WWW. Do wyodrębnienia kodu skryptu od innych elementów (np. kodu HTML) niezbędne jest umieszczenie go wewnątrz znaczników. Parser (analizator składniowy) PHP będzie przetwarzał jedynie ten fragment kodu, który znajdzie się pomiędzy znacznikiem otwierającym i zamykającym. Do dyspozycji są cztery typy znaczników:

- znaczniki kanoniczne,
- znaczniki typu SGML (skrócone),
- znaczniki typu ASP,
- znaczniki skryptów HTML.

Znaczniki kanoniczne

Znaczniki kanoniczne to standardowe i najczęściej spotykane znaczniki PHP w postaci:

```
<?php
  //kod skryptu
?>
```

Są one rozpoznawane zawsze, niezależnie od tego, jakie opcje włączy się w pliku konfiguracyjnym. Zaleca się stosowanie wyłącznie takich znaczników.

Znaczniki typu SGML

Znaczniki typu SGML mają następującą postać:

```
<?
  //kod skryptu
?>
```

Jest to najkrótsza forma znaczników bloku PHP, jaką można zastosować. By skorzystać z tego sposobu zagnieżdżania kodu PHP, należy w pliku konfiguracyjnym *php.ini* umieścić linię `short_open_tag = On` lub włączyć opcję `enable-short-tags` podczas kompilacji pakietu. Nie należy ich stosować w przypadku zagnieżdżania kodu PHP w plikach XHTML.

Znaczniki typu ASP

Znaczniki tego typu wywodzą się z techniki ASP. Mają one postać:

```
<%
  //kod skryptu
%>
```

Aby móc korzystać z tego typu wyróżnienia bloków PHP, należy w pliku konfiguracyjnym włączyć opcję `asp_tags = On`. Możliwość stosowania znaczników ASP została wprowadzona w PHP w wersji 3.0.4. Nie zaleca się wykorzystywania tej opcji.

Znaczniki skryptów HTML

Jest to typowy znacznik `<script>`, w którym parametrowi `language` została przypisana wartość `PHP`. Konstrukcja taka ma postać:

```
<script language="php">
  //kod skryptu
</script>
```

Postać ta, podobnie jak postać kanoniczna, jest rozpoznawana standardowo i nie wymaga włączania dodatkowych opcji konfiguracyjnych. W przypadku zagnieżdżania skryptów w kodzie HTML lub XHTML nie należy stosować takich znaczników, ponieważ są niezgodne z aktualnymi standardami.

Przykład skryptu

Najprostszy skrypt po prostu wyświetla napis na ekranie. Aby go napisać, można użyć instrukcji `echo("tekst");` lub `print("tekst");`. Po umieszczeniu jej między znacznikami PHP i zagnieżdżeniu w kodzie HTML (4.01 strict) uzyskamy:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
↪4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Moja strona WWW</title>
</head>
<body>
<div>
<?php echo ("Witam na stronie!"); ?>
</div>
</body>
</html>
```

Zarówno w przypadku `echo`, jak i `print` można pominąć nawias okrągły, np.: `echo "tekst";` lub `print "tekst";`.

KOMENTARZE

W kodzie PHP można zastosować trzy rodzaje komentarzy, dwa zapożyczone ze składni języków takich jak C i C++ i jeden stosowany w powłokach uniksowych:

- komentarz blokowy,
- komentarz jednowierszowy,
- komentarz jednowierszowy uniksowy.

Komentarz blokowy

Ten typ komentarza zaczyna się od sekwencji znaków `/*`, a kończy sekwencją `*/`. Wszystko to, co znajduje się pomiędzy nimi, zostanie zignorowane przez analizator składniowy PHP:

```
<?php
  /*
  To jest komentarz blokowy
  */
?>
```

Należy pamiętać, że komentarz ten koniecznie musi znaleźć się w bloku PHP. Nie wolno też dokonywać jego zagnieżdżania. Bez problemu można natomiast zastosować wewnątrz komentarza blokowego komentarze jednowierszowe.

Komentarz jednowierszowy

Komentarz tego typu rozpoczyna się od znaków `//` i obowiązuje do końca bieżącego wiersza. Można go umieszczać jedynie w bloku PHP.

```
<?php
  //To jest komentarz jednowierszowy
?>
```

Komentarz jednowierszowy uniksowy

Komentarz tego typu rozpoczyna się od znaku `#` i obowiązuje do końca bieżącego wiersza. Jego składnia pochodzi z systemu Unix. Można go umieszczać jedynie w bloku PHP.

```
<?php
  #To jest komentarz jednowierszowy
?>
```

ELEMENTY JĘZYKA

Typy danych

Występujące w PHP typy danych można podzielić na:

- typy proste (skalarne, ang. *primitive types*, *scalar types*),
- typy złożone (ang. *compound types*, *complex types*),
- typy specjalne (ang. *special types*).

Typy proste

Typ boolean

Typ logiczny przyjmujący jedną z dwóch wartości: `true` (prawda) lub `false` (fałsz). W przypadku konwersji innych typów na typ `boolean` obowiązuje zasada, że wartość `false` powstaje z przekształcenia:

- typu `integer` o wartości `0`,
- typu `double` o wartości `0`,
- typu łańcuchowego o wartości pustej `""`,
- typu łańcuchowego o wartości `"0"`,
- tablicy o zerowej liczbie elementów,
- typu obiektowego o zerowej liczbie elementów,
- typu specjalnego `null`,
- obiektu `SimpleXML`, utworzonego z pustych znaczników.

W każdym innym przypadku w wyniku konwersji otrzymywana jest wartość `true`.

Typ integer

Typ całkowitoliczbowy (oznaczany jako `int`), reprezentujący zarówno dodatnie, jak i ujemne liczby całkowite. Liczby te mogą być zapisane w trzech różnych formatach: dziesiętnym, ósemkowym (oktalnym) i szesnastkowym (heksadecymalnym). Domyślnie stosowany jest format dziesiętny. Aby zapisać liczbę ósemkową, należy poprzedzić ją znakiem `0` (zero), liczbę szesnastkową należy poprzedzić znakami `0x` lub `0X`:

- literał `16` to liczba w systemie dziesiętnym o wartości 16,
- literał `020` to liczba ósemkowa o wartości 16 dziesiętnie,
- literał `0x10` to liczba szesnastkowa o wartości 16 dziesiętnie.

Maksymalny zakres typu całkowitego zależy od platformy sprzętowo-systemowej, na jakiej uruchamiane jest PHP. Typowe zakresy wartości:

System	Liczba bajtów	Min.	Maks.
32-bitowy	4	-2^{31}	$2^{31}-1$
64-bitowy	8	-2^{63}	$2^{63}-1$

Rozmiar typu w danej implementacji może być odczytany ze stałej `PHP_INT_SIZE`, natomiast maksymalna dopuszczalna wartość — ze stałej `PHP_INT_MAX`. W przypadku przekroczenia zakresu wartość jest konwertowana na typ `double`.

Przy konwersji z typów `boolean` i `double` na typ `integer` obowiązują następujące zasady:

- Konwersja z typu `boolean` o wartości `true` daje w wyniku 1.
- Konwersja z typu `boolean` o wartości `false` daje w wyniku 0.
- Konwersja z typu `double` powoduje zaokrąglenie w dół do najbliższej liczby całkowitej.

Uwaga: jeśli wartość typu `double` przekracza zakres liczb typu `integer`, wartość, jaka powstaje w wyniku konwersji jest nieokreślona.

W przypadku konwersji z typu `string` obowiązują następujące zasady:

- Jeśli ciąg znaków rozpoczyna się od prawidłowej liczby całkowitej nieprzekraczającej dopuszczalnego zakresu (np.: `"256"`, `"-512"`, `"64xyz"`), wynikiem jest wartość reprezentowana przez ten ciąg.
- Jeśli ciąg znaków rozpoczyna się od prawidłowej liczby zmiennopozycyjnej (np.: `"1.2"`, `"-2.4"`, `"3.6e2"`, `"2.8e4xyz"`) lub przekracza dopuszczalny zakres, najpierw wykonywana jest konwersja do typu `double`.
- Jeśli ciąg znaków nie rozpoczyna się od prawidłowej liczby, wynikiem jest wartość zero.

Konwersja z innych typów niż wymienione nie została zdefiniowana i takiej konwersji nie należy wykonywać.

Uwaga: konwersja pojedynczego znaku do typu `integer` nie daje w rezultacie kodu tego znaku. W celu uzyskania kodu znaku należy korzystać z funkcji `ord` i `chr`.

Typ double

Typ `double` (oznaczany również jako `float`) reprezentuje liczby zmiennopozycyjne (ang. *floating point*). Ich zakres, podobnie jak dla typu `integer`, zależy od platformy sprzętowo-systemowej (z reguły maksymalna wartość jest zbliżona do 1.8e308). Typową reprezentacją jest zapis z kropką dziesiętną, czyli np. `1.5`. Można również używać notacji wykładniczej, w postaci `XeY`, gdzie `X` jest liczbą w notacji z kropką dziesiętną, a `Y` to żądana potęga liczby 10, np.:

- `1.2e1` to 12,
- `4e2` to 400,
- `0.12e3` to 120.

W przypadku konwersji do typu zmiennoprzecinkowego ze wszystkich typów, z wyjątkiem łańcuchowego, najpierw wykonywana jest konwersja do typu `integer`, a następnie z `integer` do `double` (przy próbie wykonania konwersji z typu obiektowego zostanie wygenerowane ostrzeżenie). Przy konwersji z typu `string` obowiązują następujące zasady:

- Jeśli ciąg znaków rozpoczyna się od prawidłowej liczby całkowitej (np.: `"256"`, `"-512"`, `"64xyz"`), której wartość nie przekracza dopuszczalnego zakresu dla typu `integer`, najpierw jest wykonywana konwersja do typu `integer`.
- Jeśli ciąg znaków rozpoczyna się od prawidłowej liczby zmiennopozycyjnej (np.: `"1.2"`, `"-2.4"`, `"3.6e2"`, `"2.8e4xyz"`) bądź też zaczyna się od liczby całkowitej przekraczającej dopuszczalny zakres dla typu `integer`, wynikiem jest wartość reprezentowana przez ciąg.
- Jeśli ciąg znaków nie rozpoczyna się od prawidłowej liczby, wynikiem jest wartość `0`.

Typ string

Typ `string`, to typ łańcuchowy, który służy do zapamiętywania sekwencji znaków. Nie ma ograniczenia długości ciągu. Pojedynczy znak zapamiętywany jest na jednym bajcie, nie ma więc bezpośredniej obsługi standardu Unicode. Łańcuch znaków można utworzyć w jednym z czterech sposobów zapisu:

- wykorzystując znaki apostrofu,
- wykorzystując znaki cudzysłowu,
- wykorzystując składnię `heredoc`,
- wykorzystując składnię `nowdoc` (od PHP w wersji 5.3.0).

Znaki apostrofu

Najprostszym sposobem deklaracji łańcucha znakowego jest ujęcie go w znaki apostrofu. PHP nie dokonuje interpretacji takiego ciągu, wyświetlany jest on zatem w niezmienionej postaci:

```
'Przykładowy łańcuch znaków'
```

Wyjątkiem jest jedynie sam znak apostrofu. Aby go uzyskać, należy poprzedzić go znakiem lewego ukośnika (backslash):

```
'Znak apostrofu wygląda tak: \''
```

Znaki cudzysłowu

Ciągi znaków ujęte w cudzysłów są przetwarzane przez PHP. Występujące w nich zmienne są zamieniane na wartości tych zmiennych. Obowiązują przy tym następujące zasady:

- Jeżeli zmienna jest typu znakowego, zawarty w niej ciąg znaków jest wklejany do ciągu bieżącego.
- Jeżeli zmienna jest innego typu niż znakowy, najpierw następuje jej konwersja na typ `string`, a następnie zostaje ona wklejona do ciągu bieżącego.
- Jeżeli zmienna nie zawiera żadnej wartości, jest traktowana jak pusty ciąg znaków.

W ciągach tego typu można stosować sekwencje znaków specjalnych:

- `\n` — nowy wiersz,
- `\r` — powrót karetki,
- `\t` — tabulator poziomy,
- `\v` — tabulator pionowy,
- `\f` — przewinięcie strony,
- `\\` — ukośnik,
- `\$` — znak dolara,
- `\"` — znak cudzysłowu.

Jeśli ukośnikiem zostanie poprzedzona liczba, zostanie ona potraktowana jako kod znaku w notacji oktalnej (ósemkowej). Przykładowy ciąg:

```
"\145\143\150\157"
```

zostanie potraktowany tak samo, jak napis:

```
"echo"
```

Jeśli przed liczbą pojawi się sekwencja `\x`, zostanie ona potraktowana jako kod znaku w notacji heksadecymalnej (szesnastkowej). Przykładowy ciąg:

```
"\x65\x63\x68\x6F"
```

zostanie potraktowany tak samo, jak napis:

```
"echo"
```

Składnia heredoc

W przypadku składni `heredoc` łańcuch znakowy rozpoczyna się od sekwencji `<<<`, po której następuje identyfikator. Następnie wykorzystuje się ten identyfikator, by zasygnalizować koniec łańcucha znakowego.

```
<<<ID1
Przykładowy ciąg znaków
ID1;
```

Nazwa identyfikatora musi się zaczynać od znaku podkreślenia lub litery oraz może zawierać dowolną kombinację liter, cyfr i znaków podkreślenia.

Linia kończąca nie może zawierać żadnych innych znaków oprócz identyfikatora i średnika. Uwaga ta dotyczy wszystkich znaków, również spacji, tabulatorów itp.

Składnia nowdoc

Składnia typu `nowdoc` została wprowadzona w PHP 5.3.5. Ciągi tego typu nie są przetwarzane przez PHP, podobnie jak ciągi utworzone wyłącznie za pomocą znaków apostrofu. Zasady konstrukcji są podobne do zasad `heredoc`. Należy użyć sekwencji `<<<`, po której następuje identyfikator, z tym że identyfikator musi być ujęty w znaki apostrofu prostego. Identyfikator jest następnie wykorzystywany w celu zasygnalizowania końca łańcucha znakowego, jednak bez znaków apostrofu:

```
<<<'ID1'
Przykładowy ciąg znaków
ID1;
```

Dostęp do pojedynczych znaków

Dostęp do pojedynczego znaku w ciągu odbywa się przez zastosowanie nawiasu kwadratowego (sposób zalecany) lub klamrowego, w którym umieszcza się indeks znaku (pierwszy znak ciągu ma indeks zero), np.:

```
<?php
$napis = "Przykład";
$litera = $napis[2];
echo $litera;
?>
```

W analogiczny sposób można wymienić znak w ciągu:

```
<?php
$napis = "Przykład";
$napis[2] = 'z';
echo "$napis";
?>
```

Wybrane funkcje operujące na ciągach znaków

Funkcja addcslashes (dostępna od PHP 4)

Deklaracja: `string addcslashes (string str, string charlist)`

Zwraca ciąg, w którym znaki wymienione w `charlist` zostały poprzedzone znakiem backslash (`\`). Znaki o kodach mniejszych od 32 i większych od 126 zostaną zamienione na postać oktalną. Przykładowo: wykonanie `addcslashes('abcde', 'ab')` da w wyniku ciąg `\a\bcdefgh`.

Funkcja addslashes (dostępna od PHP 3)

Deklaracja: `string addslashes (string str)`

Zwraca ciąg, w którym znaki specjalne (apostrof, cudzysłów, backslash oraz bajt zerowy NUL) zostaną poprzedzone znakiem ukośnika. Funkcja jest najczęściej wykorzystywana do przetwarzania zapytań do baz danych.

Funkcja bin2hex (dostępna od PHP 3.0.9)

Deklaracja: `string bin2hex (string str)`

Zwraca ciąg, w którym wszystkie znaki w `str` zostały przedstawione w postaci heksadecymalnej. Przykładowo: wywołanie `bin2hex('abcde')` da w wyniku ciąg `6162636465`.

Funkcja chr (dostępna od PHP 3)

Deklaracja: `string chr (int ascii)`

Zwraca ciąg zawierający znak o kodzie przekazanym w argumencie `ascii`.

Funkcja chunk_split (dostępna od PHP 3.0.6)

Deklaracja: `string chunk_split (string body[, int chunklen [, string end]])`

Dzieli ciąg `body` na mniejsze części o długości `chunklen` (domyślnie 76 znaków), dodając na końcu każdej linii ciąg `end` (domyślnie znaki końca linii `\r\n`).

Funkcja convert_uudecode (dostępna od PHP 5)

Deklaracja: `string convert_uudecode (string data)`

Dekoduje dane zakodowane za pomocą algorytmu `uuencode` (np. przetworzone przez funkcję `convert_uuencode`). Zwraca rozkodowany ciąg znaków.

Funkcja convert_uuencode (dostępna od PHP 5)

Deklaracja: `string convert_uuencode (string str)`

Koduje ciąg znaków zapisany w argumencie `str` za pomocą algorytmu `uuencode`. Zwraca zakodowany ciąg znaków.

Funkcja count_chars (dostępna od PHP 4)

Deklaracja: `mixed count_chars (string str [, int mode])`

Zlicza wystąpienia każdego bajta w ciągu `str` i zwraca wynik obliczeń w postaci wskazanej przez argument `mode` (domyślnie `mode` ma wartość 0):

- 0 — tablica, w której kluczami są bajty, a wartościami kluczy liczba wystąpień danego bajta w ciągu;
- 1 — tak jak parametr 0, z tą różnicą, że tablica nie uwzględnia kluczy, których wartość wynosi 0;
- 2 — tak jak parametr 0, z tą różnicą, że tablica uwzględnia wyłącznie klucze, których wartość wynosi 0;
- 3 — ciąg znaków zawierający znaki występujące w ciągu `str`;
- 4 — ciąg znaków, zawierający znaki niewystępujące w ciągu `str`.

Funkcja crc32 (dostępna od PHP 4.0.1)

Deklaracja: `int crc32 (string str)`

Oblicza 32-bitową sumę kontrolną ciągu `str`.

Funkcja crypt (dostępna od PHP 3)

Deklaracja: `string crypt (string str [, string salt])`

Koduje ciąg znaków `str` na podstawie funkcji szyfrujących, udostępnionych przez system operacyjny (typowo DES). Od PHP 5.3.0 zawiera własne implementacje algorytmów szyfrujących i generujących funkcje skrótu.

Funkcja explode (dostępna od PHP 3)

Deklaracja: `array explode(string separator, string str[, int limit])`

Dzieli ciąg znaków `str` na części rozdzielane znakami separatora. Opcjonalny argument `limit` (dostępny od PHP 4.0.1) wskazuje maksymalną liczbę elementów tablicy wynikowej (od wersji 5.1.0 może mieć wartość ujemną). Jeśli argument `separator` będzie pustym ciągiem znaków, funkcja zwraca wartość `false`. Jeżeli znaki separatora nie będą występowały w ciągu `str`, zwrócony zostanie cały ciąg.

Funkcja fprintf (dostępna od PHP 5)

Deklaracja: `int fprintf (resource handle, string format [, mixed args [, mixed ...]])`

Zapisuje w strumieniu wskazywanym przez `handle` dane znajdujące się w argumentach `args`, sformatowane według argumentu `format`. Zwraca liczbę bajtów ciągu wynikowego. Argument `format` może zawierać znaczniki opisane przy funkcji `sprintf`.

Funkcja html_entity_decode (dostępna od PHP 4.3.0)

Deklaracja: `string html_entity_decode (string str [, int quote_style [, string charset]])`

Wykonuje czynność odwrotną do funkcji `htmlentities`. Przykładowo ciąg:

```
&lt;b&gt;Napis&lt;/b&gt;
```

zostanie zamieniony na:

```
<b>Napis</b>
```

Opcjonalny argument `quot_style` może przyjmować wartości (domyślnie `ENT_COMPAT`):

- `ENT_COMPAT` — konwersji zostaną poddane cudzysłowy;
- `ENT_QUOTES` — konwersji zostaną poddane apostrofy i cudzysłowy;
- `ENT_NOQUOTES` — apostrofy i cudzysłowy nie będą konwertowane.

Ciąg dalszy na stronie 2 →