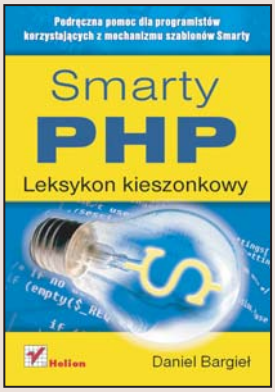
Smarty PHP. Leksykon kieszonkowy



Autor: Daniel Bargieł
ISBN: 83-246-0676-9
Format: B6, stron: 112

Smarty to obiektowa biblioteka dla języka PHP służąca do tworzenia witryn internetowych z wykorzystaniem szablonów. Dzięki wbudowanemu systemowi buforowania Smarty jest niezwykle wydajna i szybka, co staje się szczególnie ważne przy rozbudowanych witrynach WWW. Wykorzystanie biblioteki Smarty pozwala twórcom witryn i aplikacji WWW znacznie przyśpieszyć prace nad warstwą prezentacyjną i późniejszymi modyfikacjami swoich produktów.

Książka „Smarty PHP. Leksykon kieszonkowy" to zestawienie najważniejszych informacji dotyczących tej biblioteki. Znajdziesz w nim wszystko, co może okazać się przydatne podczas tworzenia witryny WWW z wykorzystaniem szablonów Smarty. W kolejnych rozdziałach opisano zagadnienia dotyczące konfigurowania Smarty, pracy ze zmiennymi, obiektów Smarty oraz obsługi pamięci podręcznej.

- Komentarze w szablonach
- Definiowanie zmiennych
- Konfiguracja
- Przetwarzanie danych
- Metody obiektów klasy Smarty
- Funkcje Smarty
- Korzystanie z mechanizmu buforowania

Jeśli korzystasz z szablonów Smarty, ta książka powinna znaleźć się w Twojej bibliotece

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

1. Podstawy 5
Renderowanie szablonu 5
Komentarze w szablonach 8
Zmienne szablonów Smarty 9
Stałe 13

2. Właściwości konfiguracyjne systemu Smarty 14
Kompilacja 14
Debuger 17
Pamięć podręczna 20

3. Modyfikatory zmiennych Smarty 24
Modyfikatory podstawowe 24
Kombinacja modyfikatorów 39

4. Metody obiektów klasy Smarty 41
Operacje na zmiennych 41
Obsługa szablonów TPL 47
Zgłaszanie błędów — metoda trigger_error() 50
Obsługa plików konfiguracyjnych 51

5. Funkcje Smarty 53
Funkcje iteracyjne 53
Funkcje warunkowe if 62
Funkcje dołączające 63
Funkcje HTML i Mail 68
Pozostałe funkcje 77

6. Rozszerzenia systemu szablonów 88
Nazewnictwo rozszerzeń 88
Funkcje szablonów 90
Funkcje blokowe szablonów 92
Modyfikatory 95

7. Obsługa pamięci podręcznej 97
Włączanie obsługi pamięci podręcznej 97
Testowanie kopii szablonu w pamięci podręcznej 98
Tworzenie wielu kopii dla jednego szablonu TPL 100
Usuwanie kopii szablonu z pamięci podręcznej 101
Kontrolowanie rozszerzeń Smarty 104

Skorowidz 105

Rozdział 4. Metody obiektów klasy Smarty

Obiekty klasy `Smarty` posiadają szereg metod, dzięki którym można wykonywać określone operacje dotyczące zmiennych, szablonów czy też plików konfiguracyjnych.

Operacje na zmiennych

Metoda append() oraz append_by_ref()

Metoda `append()` (definicja 4.1) umożliwia dodanie nowej wartości do zmiennej *Smarty*. Metoda dodaje nową zmienną, tworząc jej kopię.

Definicja 4.1. Metoda append()

```
void append(string nazwa_zmiennej, mixed wartosc_zmiennej
[, bool polaczenie])
```

Metoda przyjmuje następujące parametry:

- `nazwa_zmiennej` (*wymagany*) — nazwa zmiennej, do której ma zostać przypisana wartość. Jeżeli nowa wartość jest dodawana do zmiennej typu *string*, to następuje konwersja zmiennej `string` na tablicę, a następnie dodanie do niej kolejnego elementu;
- `wartosc_zmiennej` (*wymagany*) — wartość zmiennej, która zostanie dodana do zmiennej istniejącej;
- `polaczenie` — jeżeli parametr będzie miał wartość *true*, `wartosc_zmiennej` zostanie połączona z aktualną zmienną `nazwa_zmiennej`. W przeciwnym wypadku nowa wartość zostanie dołączona.

Na listingu 4.1 został przedstawiony przykład wykorzystania metody `append()`.

Listing 4.1. Przykład wykorzystania metody append()

```
[Skrypt PHP]
<?php
...
$smarty_obj->append(array('zmienna_1'=>'zmienna 1a',
'zmienna_2'=>'zmienna 2a'));
$smarty_obj->append('zmienna_1', 'zmienna 1b');
...
?>
[Szablon TPL]
{$zmienna_1[1]}

[Wynik wykonania skompilowanego szablonu TPL]
zmienna 1b
```

Metoda `append_by_ref()` (definicja 4.2) w działaniu jest podobna do metody `append()` z tą różnicą, że wartość przekazywana jest przez referencję, a nie przez wartość, jak w przypadku metody `append()`.

Definicja 4.2. Metoda append_by_ref()

```
void append_by_ref(string nazwa_zmiennej, mixed
wartosc_zmiennej [, bool polaczenie])
```

Na listingu 4.2 został przedstawiony przykład wykorzystania metody `append_by_ref()`.

Listing 4.2. Przykład wykorzystania metody append_by_ref()

```
[Skrypt PHP]
<?php
...
$wartosc = 'Zmienna 1b';
$smarty_obj->append(array('zmienna_1'=>'zmienna 1a'));
$smarty_obj->append('zmienna_1', $wartosc);
$smarty_obj->append_by_ref('zmienna_1', $wartosc);
$wartosc = 'Zmienna 1c';
...
?>
```

```
[Szablon TPL]
{$zmienna_1[0]}
{$zmienna_1[1]}
{$zmienna_1[2]}

[Wynik wykonania skompilowanego szablonu TPL]
zmienna 1a
Zmienna 1b
Zmienna 1c
```

Metoda assign() oraz assign_by_ref()

Metoda `assign()` (definicja 4.3) pozwala na utworzenie zmiennej `Smarty` dostępnej w szablonie TPL.

Definicja 4.3. Metoda assign()

```
void assign(array lista_zmiennych)
[lub]
void assign(string nazwa_zmiennej, mixed wartosc_zmiennej)
```

Jako parametr metody może zostać użyta tablica asocjacyjna, w której nazwą przyszłej zmiennej jest klucz asocjacyjny elementu tablicy, a jej wartością — wartość elementu tablicy (listing 4.3).

Listing 4.3. Tablica asocjacyjna zawierająca listę elementów

```
[Skrypt PHP]
<?php
...

//Przypisanie zmiennych
$smarty_obj->assign(array('zmienna_1'=>'Wartośc zmiennej 1',
'zmienna_2'=>'Wartośc zmiennej 2'));

//Rendering szablonu
$smarty_obj->display('szablon.tpl');
?>

[Szablon TPL]
Wartość zmiennej 1 to: {$zmienna_1}
Wartość zmiennej 2 to: {$zmienna_2}

[Wynik wykonania skompilowanego szablonu TPL]
Wartość zmiennej 1 to: Wartośc zmiennej 1
Wartość zmiennej 2 to: Wartośc zmiennej 2
```

Drugim sposobem przypisania zmiennych do szablonu TPL jest podanie do metody `assign()` następujących parametrów:

- `nazwa_zmiennej` (*wymagany*) — nazwa, pod którą wartość `wartosc_zmiennej` będzie dostępna w szablonie TPL;
- `wartosc_zmiennej` (*wymagany*) — wartość zmiennej `nazwa_zmiennej` dostępna w szablonie TPL.

Na listingu 4.4 został przedstawiony kod, który przypisuje takie same zmienne jak kod z listingu 4.3.

Listing 4.4. Inny sposób przypisania zmiennych

```
<?php
...
//Przypisanie zmiennych Smarty
$smarty_obj->assign('zmienna_1', 'Wartośc zmiennej 1');
$smarty_obj->assign('zmienna_2', 'Wartośc zmiennej 2');

//Rendering szablonu
$smarty_obj->display('szablon.tpl');
?>
```

Metoda `assign_by_ref()` (definicja 4.4) działa w sposób podobny do metody `assign()` z tą różnicą, że zmienna przypisywana jest przez referencję a nie przez wartość.

Definicja 4.4. Metoda assign_by_ref()

```
void assign_by_ref(string nazwa_zmiennej, mixed
wartosc_zmienej)
```

Przykład wykorzystania metody `assign_by_ref()` został przedstawiony na listingu 4.5.

Listing 4.5. Przykład wykorzystania metody assign_by_ref()

```
[Skrypt PHP]
<?php
...
$imie = 'Daniel';
$nazwisko = 'Bargieł';
$smarty_obj->assign('imie', $imie);
$smarty_obj->assign_by_ref('nazwisko', $nazwisko);
```

```
$imie = 'Ewa';
$nazwisko = 'Gelner';

//Rendering szablonu
$smarty_obj->display('szablon.tpl');
?>

[Szablon TPL]
Witaj {$imie} {$nazwisko}

[Wynik wykonania skompilowanego szablonu TPL]
Witaj Daniel Gelner
```

Metoda clear_all_assign() oraz clear_assign()

Metoda `clear_all_assign()` (definicja 4.5) usuwa wszystkie zmienne *Smarty* dostępne w szablonach TPL.

Definicja 4.5. Metoda clear_all_assign()

```
void clear_all_assign()
```

Metoda `clear_assign()` (definicja 4.6) usuwa określoną zmienną lub grupę zmiennych.

Definicja 4.6. Metoda clear_assign()

```
void clear_assign(mixed zmienne)
```

Metoda `clear_assign()` przyjmuje jeden parametr, który zawiera nazwę zmiennej przeznaczonej do usunięcia lub tablicę zawierającą listę nazw zmiennych do usunięcia. Przykład wykorzystania metody `clear_assign()` został przedstawiony na listingu 4.6.

Listing 4.6. Przykład wykorzystania metody clear_assign()

```
<?php
...
// Usunięcie pojedynczej zmiennej
$smarty_obj->clear_assign('Name');

// Usunięcie listy zmiennych
$smarty_obj->clear_assign(array('Name', 'Address', 'Zip'));
...
?>
```

Metoda get_template_vars()

Metoda `get_template_vars()` (definicja 4.7) obiektów klasy `Smarty` zwraca tablicę zawierającą wszystkie zmienne *Smarty*.

Definicja 4.7. Metoda get_template_vars()

```
array get_template_vars([string nazwa_zmiennej])
```

Jeżeli zostanie podany opcjonalny parametr `nazwa_zmiennej`, metoda `get_template_vars()` zwróci wartość wybranej zmiennej. W przeciwnym wypadku zostanie zwrócona tablica asocjacyjna, zawierająca wartość wszystkich zmiennych *Smarty*. Przykład wykorzystania metody `get_template_vars()` został przedstawiony na listingu 4.7.

Listing 4.7. Przykład wykorzystania metody get_template_vars()

```
[Skrypt PHP]
<?php
...
$imie = 'Daniel';
$nazwisko = 'Bargieł';

$smarty_obj->assign('imie', $imie);
$smarty_obj->assign('nazwisko', $nazwisko);

var_dump($smarty_obj->get_template_vars('imie'));
var_dump($smarty_obj->get_template_vars());
?>

[Wynik wykonania skryptu PHP]
string(6) "Daniel"
array(3) {
  ["SCRIPT_NAME"]=>
  string(10) "/index.php"
  ["imie"]=>
  &string(6) "Daniel"
  ["nazwisko"]=>
  string(7) "Bargieł"
}
```

Obsługa szablonów TPL

Obiekty klasy `Smarty` mają dwie metody, dzięki którym można zrenderować szablon TPL. Pierwsza z nich to metoda `display()` (definicja 4.8), która wysyła treść zrenderowanego szablonu do bufora wyjściowego. Metoda `fetch()` (definicja 4.9) zwraca wartość typu *string* zawierającą treść zrenderowanego szablonu TPL.

Metoda display()

Metoda `display()` (definicja 4.8) wyświetla zrenderowany szablon (wysyła jego treść do bufora wyjściowego)

Definicja 4.8. Metoda display()

```
void display(string nazwa_szablonu [, string
identyfikator_cache [,string identyfikator_kompilacji]])
```

Metoda przyjmuje następujące parametry:

- `nazwa_szablonu` (*wymagany*) — ścieżka dostępu do szablonu TPL, który ma zostać wyświetlony;
- `identyfikator_cache` — parametr opcjonalny, pozwalający na określenie identyfikatora kopii szablonu w pamięci podręcznej. Więcej informacji na ten temat w rozdziale „Obsługa pamięci podręcznej”;
- `identyfikator_kompilacji` — parametr ten pozwala na utworzenie kilku skompilowanych wersji jednego szablonu TPL. Może to być przydatne np. wtedy, gdy szablon powinien być skompilowany oddzielnie dla każdej z wersji językowych obsługiwanych w aplikacji.

Przykład wykorzystania metody `display()` został przedstawiony na listingu 4.8.

Listing 4.8. Przykład wykorzystania metody display()

```
<?php
...
// Ścieżka absolutna
$smarty->display('/usr/local/include/templates/header.tpl');
$smarty->display('file:/usr/local/include/templates/
header.tpl');

// Ścieżka absolutna w systemie Windows
$smarty->display('file:c:/www/pub/templates/header.tpl');

// Ładowanie szablonu z zasobu o nazwie db
$smarty->display('db:header.tpl');
?>
```

Metoda fetch()

Metoda `fetch()` (definicja 4.9) obiektów klasy `Smarty` działa bardzo podobnie do metody `display()`, z tym że nie wysyła zrenderowanego szablonu do bufora wyjściowego, lecz zwraca go w postaci łańcucha tekstowego.

Definicja 4.9. Metoda fetch()

```
string fetch(string nazwa_szablonu[, string
identyfikator_cache [, string identyfikator kompilacji]])
```

Znaczenie parametrów metody `fetch()` jest identyczne jak w przypadku metody `display()` (definicja 4.8).

Na listingu 4.9 został przedstawiony przykład wykorzystania metody `fetch()` obiektu klasy `Smarty`.

Listing 4.9. Przykład wykorzystania metody fetch()

```
<?php
...
echo $smarty_obj->fetch('szablon.tpl');
?>
```

Metoda template_exists()

Metoda `template_exists()` (definicja 4.10) sprawdza, czy określony szablon TPL istnieje.

Definicja 4.10. Metoda template_exists()

```
bool template_exists(string nazwa_szablonu)
```

Jeżeli szablon `nazwa_szablonu` istnieje, metoda zwraca wartość *true*. W przeciwnym wypadku zwracana jest wartość *false*.

Przykład wykorzystania metody `template_exists()` został przedstawiony na listingu 4.10.

Listing 4.10. Przykład wykorzystania metody template_exists()

```
<?php
...
$szablon_tpl = basename($_GET['strona']).'.tpl';
if ($smarty_obj->template_exists($szablon_tpl)) {
   $smarty_obj->display($szablon_tpl);
} else $smarty_obj->display('strona_nie_istnieje.tpl');
?>
```

Metoda clear_compiled_tpl()

Metoda `clear_compiled_tpl()` (definicja 4.11) czyści skompilowaną wersję szablonu TPL lub cały katalog kompilacji.

Definicja 4.11. Metoda clear_compiled_tpl()

```
void clear_compiled_tpl([string nazwa_szablonu [, string
identyfikator_kompilacji [, int czas_zycia]]])
```

Metoda przyjmuje następujące parametry:

- `nazwa_szablonu` — nazwa pliku szablonu TPL, którego skompilowana kopia powinna zostać usunięta;
- `identyfikator_kompilacji` — podanie identyfikatora kompilacji spowoduje, że zostaną usunięte tylko te kopie szablonu TPL, które posiadają określony identyfikator kompilacji;

- `czas_zycia` — podanie czasu życia skompilowanych kopii szablonów TPL spowoduje, że metoda `clear_compiled_tpl()` usunie tylko te kopie, które są starsze niż podany czas życia.

Na listingu 4.11 został podany przykład wykorzystania metody `clear_compiled_tpl()`.

Listing 4.11. Przykład wykorzystania metody clear_compiled_tpl()

```
<?php
...
//Usunięcie wszystkich kopii kompilacji szablonu TPL
$smarty_obj->clear_compiled_tpl('szablon.tpl');

//Wyczyszczenie całego katalogu kompilacji
$smarty_obj->clear_compiled_tpl();
?>
```

Zgłaszanie błędów — metoda trigger_error()

Metoda `trigger_error()` (definicja 4.12) umożliwia zgłoszenie błędu przez obiekt klasy `Smarty`. Metoda ta jest najczęściej wykorzystywana w kodzie funkcji rozszerzających możliwości *Smarty* (patrz rozdział „Rozszerzenia systemu szablonów").

Definicja 4.12. Metoda trigger_error()

```
void trigger_error(string tresc_bledu [, int poziom_bledu])
```

Metoda przyjmuje następujące parametry:

- `tresc_bledu` (*wymagany*) — komunikat błędu;
- `poziom_bledu` — możliwe poziomy błędu są takie same jak dla funkcji PHP `trigger_error()`. Domyślną wartością parametru jest *E_USER_WARNING*.

Przykład wykorzystania metody `trigger_error()` został przedstawiony na listingu 4.12.

Listing 4.12. Przykład wykorzystania metody trigger_error()

```
[Skrypt PHP]
<?php
...
$smarty_obj->trigger_error('Zgłaszam błąd obiektu klasy
Smarty', E_USER_WARNING);
?>

[Wynik wykonania skryptu PHP]
<b>Warning</b>:  Smarty error: Zgłaszam błąd obiektu klasy
Smarty in <b>c:\document_root\helion_smarty\smarty\
Smarty.class.php</b> on line <b>1095</b>
```

Obsługa plików konfiguracyjnych

Pliki zawierające zmienne konfiguracyjne *Smarty* mogą być obsługiwane z poziomu kodu PHP oraz szablonów TPL.

Metoda config_load()

Metoda `config_load()` (definicja 4.13) obiektu klasy `Smarty` ładuje plik konfiguracyjny i przypisuje zawarte w nim zmienne do szablonów.

Definicja 4.13. Metoda config_load()

```
void config_load(string nazwa_pliku [, string nazwa_sekcji])
```

Metoda przyjmuje następujące parametry:

- `nazwa_pliku` (*wymagany*) — nazwa pliku konfiguracyjnego, który ma zostać załadowany;
- `nazwa_sekcji` — nazwa sekcji w pliku konfiguracyjnym, z którego powinny zostać załadowane zmienne.

Działanie metody jest identyczne z działaniem funkcji `config_load` (patrz rozdział „Funkcje Smarty"), przy czym załadowane zmienne zawsze posiadają zasięg globalny, czyli są dostępne w każdym szablonie TPL.

Metoda clear_config()

Metoda `clear_config()` (definicja 4.14) usuwa z szablonów przypisane zmienne konfiguracyjne.

Definicja 4.14. Metoda clear_config()

```
void clear_config([string nazwa_zmiennej])
```

Jeżeli podany zostanie opcjonalny parametr `nazwa_zmiennej`, metoda usunie tylko podaną zmienną.

Metoda get_config_vars()

Metoda `get_config_vars()` (definicja 4.15) zwraca listę zmiennych konfiguracyjnych przypisanych do szablonu TPL.

Definicja 4.15. Metoda get_config_vars()

```
array get_config_vars([string nazwa_zmiennej])
```

Jeżeli podany zostanie opcjonalny parametr `nazwa_zmiennej`, metoda zwróci jedynie wartość podanej zmiennej.