Linux, Apache, MySQL i PHP. Zaawansowane programowanie

Autorzy: Jason Gerner, Morgan L. Owens, Elizabeth Naramore, Matt Warden, Jeremy Stolz
Tłumaczenie: Wojciech Moch
ISBN: 83-246-0489-8
Tytuł oryginału: Professional LAMP: Linux, Apache, MySQL and PHP Web Development
Format: B5, stron: 432

Zaprojektuj profesjonalne witryny WWW

- Poznaj zasady programowania obiektowego w PHP5.
- Wykorzystaj pełnię możliwości bazy MySQL.
- Zabezpiecz witryny WWW przed atakami hakerów.

Połączenie możliwości systemu operacyjnego Linux, serwera WWW Apache, bazy danych MySQL i języka PHP pozwala na tworzenie i utrzymywanie rozbudowanych aplikacji internetowych. Popularność tych narzędzi, często określanych za pomocą akronimu LAMP (Linux, Apache, MySQL, PHP), jest efektem ich elastyczności, wydajności i doskonałej interakcji pomiędzy elementami tego zestawu. Nie bez znaczenia jest także fakt, że dostępne są one bezpłatnie, na licencji open source, co zwalnia zarówno twórców aplikacji, jak i ich użytkowników z konieczności ponoszenia jakichkolwiek opłat licencyjnych.

Książka „Linux, Apache, MySQL i PHP. Zaawansowane programowanie" opisuje metody tworzenia wydajnych aplikacji WWW. Czytając ją, poznasz możliwości języka PHP5 w zakresie programowania obiektowego i nauczysz się formułować złożone zapytania SQL. Dowiesz się, jak optymalnie skonfigurować serwer WWW Apache, zabezpieczać aplikacje WWW przed atakami i korzystać z modułów rozszerzających, takich jak PEAR i PECL. Znajdziesz tu również informacje o systemach zarządzania treścią, technologii AJAX i mechanizmach buforowania dostępnych dla PHP.

- Obsługa wyjątków w PHP
- Programowanie obiektowe
- Złożone zapytania SQL
- Konfiguracja PHP5
- Uwierzytelnianie użytkowników witryn WWW
- Korzystanie z pakietu PEAR
- Optymalizacja wydajności aplikacji
- Generowanie z poziomu PHP plików PDF, SWF i grafiki
- Tworzenie interfejsów użytkownika za pomocą AJAX
- Zarządzanie treścią serwisów WWW

Wykorzystuj w pracy nowoczesne narzędzia

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

Spis treści

1

Co nowego w PHP 5

O co chodzi z tym nowym PHP 5? Każdy doświadczony programista korzystający z języka PHP 4 najprawdopodobniej zna się na programowaniu obiektowym i zna też sposoby tworzenia takiego oprogramowania w PHP 4. Jeżeli ktoś nie korzystał jeszcze z języka PHP, ale zna inne języki programowania, to zapewne stwierdzi, że implementacja mechanizmów obiektowych zastosowana w PHP 5 nie jest mu obca. Na szczęście w języku PHP 5 obsługa obiektów została bardzo uproszczona. Wygoda ta okupiona została jednak różnymi zmianami i poprawkami, które pociągnęły za sobą zwiększenie ilości opcji w pliku konfiguracyjnym *php.ini*. Oczywiście oprócz „poprawionej wygody obsługi obiektów" w PHP 5 znajdziemy też wiele nowych funkcji związanych z obsługą tablic oraz funkcji przeznaczonych do innych zdań. W niniejszym rozdziale przyjrzymy się wszystkim tym zmianom.

Zmiany w obsłudze obiektów

Wszystkie opisywane tu zmiany dotyczą modelu obiektów oraz powiązanych z nim funkcji i zagadnień. Większość zmian omawiana jest w szczegółach w rozdziale 2., a tutaj prezentujemy tylko ich skrócony zarys.

Przekazywanie obiektów

Jedną z najważniejszych modyfikacji modelu obiektowego w języku PHP 5 jest sposób przekazywania parametrów do funkcji. W języku PHP 4 zmienne były domyślnie przekazywane przez wartość, a nie przez referencję, chyba że sami zapisaliśmy odpowiedni tryb przekazywania za pomocą specjalnej składni `&$nazwa_zmiennej`. W języku PHP 5 zmienne są domyślnie przekazywane przez referencję.

Wyjątki

W skrócie, wyjątki są procedurami wykonywanymi w momencie wystąpienia jakiegoś błędu. Dzięki temu program nie zatrzyma się po natknięciu się na nieoczekiwany błąd, ale wykonany zostanie dopisany przez nas kod, w którym możemy zapisać, co program ma zrobić w przypadku wystąpienia wspomnianego błędu. Zapewne wszyscy znają funkcję `set_error_handler()` dostępną w języku PHP 4. Jeżeli ktoś jej nie zna, to wyjaśniamy, że funkcja ta pozwala na zdefiniowanie funkcji użytkownika odpowiedzialnej za obsługę błędów. Niestety, implementacja tej funkcji miała wiele ograniczeń. Na przykład nie pozwalała na obsługę błędów typu `E_ERROR`, `E_PARSE`, `E_CORE_ERROR`, `E_CORE_WARNING`, `E_COMPILE_ERROR`, `E_COMPILE_WARNING` i większości błędów `E_STRICT`. Poza tym, jeżeli w skrypcie błąd pojawił się przed wywołaniem funkcji `set_error_handler()`, to funkcja obsługi błędu nigdy nie została wywołana. W PHP 5 otrzymujemy całkowicie nowy mechanizm obsługi błędów.

Try/Catch/Throw

Każdy kto miał już wcześniej kontakty z językami programowania takimi jak C++ lub Java, z całą pewnością zetknął się też ze słowami kluczowymi `try`, `catch` i `throw`. W momencie wystąpienia jakiegoś błędu *wywoływany* (ang. throw) jest wyjątek. Kod, który może powodować takie błędy, umieszczany jest wewnątrz bloku `try/catch`. W takim bloku następuje próba (ang. try) wykonania tego kodu, a jeżeli wywołany zostanie wyjątek, to jest on przechwytywany (ang. catch) i wykonywany jest kod umieszczony w sekcji `catch`. Składnia instrukcji `try/catch/throw` wygląda następująco:

```
<?php

try
{
  $error_message = 'Witam, nazywam się błąd';
  throw new Exception($error_message);
}
catch (Exception $e)
{
  echo 'Przechwycono wyjątek: ', $e->gerMessage(), "\n";
}

?>
```

Wbudowana klasa Exception

Wykorzystana w powyższym przykładzie klasa `Exception` również jest nowym elementem wprowadzonym w PHP 5 i jest klasą wbudowaną w język. W języku PHP 5 możemy teraz odczytywać różne informacje o przechwyconych wyjątkach i odpowiednio wykorzystywać je do podjęcia właściwych działań. Wbudowana klasa `Exception`, która zajmuje się przechowywaniem i przetwarzaniem tych danych, wygląda następująco:

```
<?php

class Exception
{
  protected $message = 'Komunikat wyjątku';
  protected $code = 0;
  protected $file;
  protected $line;

  function __construct($message = null, $code = 0);

  final function getMessage();
  final function getCode();
  final function getFile();
  final function getLine();
  final function getTrace();
  final function getTraceAsString();

  function __toString()
}

?>
```

Przyjrzyjmy się teraz dokładniej elementom klasy `Exception`:

Nazwa metody lub pola	Reprezentowane dane
`$message`	Komunikat wyjątku; można wpisać tutaj dowolny tekst.
`$code`	Kod zdefiniowany przez użytkownika.
`$file`	Nazwa pliku, w którym wystąpił błąd.
`$line`	Numer wiersza, w którym wystąpił błąd.
`getMessage()`	Zwraca wartość pola `$message`.
`getCode()`	Zwraca wartość pola `$code`.
`getFile()`	Zwraca wartość pola `$file`.
`getLine()`	Zwraca wartość pola `$line`.
`getTrace()`	Zwraca tablicę przechowującą informacje odebrane z funkcji `debug_backtrace()`.
`getTraceAsString()`	Zwraca te same informacje co funkcja `getTrace()`, ale sformatowane w postaci ciągu znaków.
`toString()`	Magiczna metoda języka PHP 5 pozwalająca na sformatowanie obiektu do postaci ciągu znaków, jeżeli obiekt używany jest w instrukcjach `print` lub `echo`.

W poprzednim przykładzie wykorzystującym konstrukcję `try/catch/throw` użyliśmy metody `getMessage()` z wbudowanej klasy `Exception` i z jej pomocą wypisaliśmy tekst komunikatu.

Rozbudowywanie wbudowanej klasy Exception

Wbudowana klasa wyjątku jest najogólniejszą klasą tego typu i w związku z tym doskonale sprawdza się w wielu sytuacjach. Czasami jednak możemy chcieć nieco większej szczegółowości obsługi wyjątków. Na przykład, gdy w czasie wysyłania e-maila zapomnimy wypełnić pola adresu odbiorcy, to nie pojawi się żaden błąd PHP. W takiej sytuacji jedyną metodą stwierdzenia, że e-mail nie został wysłany, jest przejrzenie dzienników serwera *sendmail*. Jest to jednak bardzo istotna informacja, a zatem dobrze byłoby rozbudować domyślną klasę wyjątku, wykorzystując przy tym słowo kluczowe `extends`, na przykład tak:

```
<?php

class emailException extends Exception
{
  function __construct($message)
  {
    echo "W czasie wysyłania e-maila wystąpił błąd: <br \>";
    echo $message;
  }
}

function sendEmail($to, $subject, $message)
{
  if ($to == NULL)
  {
     throw new emailException ("Brak adresu odbiorcy");
  }

  mail($to, $subject, $message);
}

try
{
  sendEmail("", "Test wyjątku",
            "Sprawdzamy działanie mechanizmu obsługi wyjątków w PHP 5");
}
catch (emailException $e)
{
  echo " w pliku <strong>" . $e->getFile() . "</strong>";
  echo " w wierszu <strong>" . $e->getLine() . "</strong>";
}

?>
```

W efekcie na stronie zobaczymy następujące komunikaty:

```
W czasie wysyłania e-maila wystąpił błąd:
Brak adresu odbiorcy w pliku /katalog/email_exception.php w wierszu 11.
```

Z wbudowanej klasy wyjątku mieliśmy możliwość odczytania informacji o nazwie pliku i wierszu, a jednocześnie mogliśmy przygotować znacznie dokładniejszy komunikat o błędzie, ułatwiający debugowanie programu.

Funkcja set_exception_handler()

Domyślnie każdy nieprzechwycony wyjątek powoduje błąd krytyczny, który zatrzymuje działanie skryptu. Język PHP 5 daje nam szansę obsłużenia każdego nieprzechwyconego wyjątku za pomocą funkcji `set_exception_handler()`. Do funkcji tej przekazywane są nazwa klasy, komunikat błędu i ślad wykonania (ang. backtrace), ale można w niej odczytywać również informacje zapisane we wbudowanej klasie wyjątku. Funkcji `set_exception_handler()` używać należy następująco:

```
<?php

function lastResort ($e)
{
  echo "Nieprzechwycony wyjątek pochodzi z klasy " . get_class($e);
  echo " z pliku " . $e->getFile() . " z wiersza " . $e->getLine();
}

set_exception_handler("lastResort");

throw new Exception;

?>
```

W efekcie zobaczymy następujący komunikat:

```
Nieprzechwycony wyjątek pochodzi z klasy Exception z pliku:
/katalog/set_exception_handler.php z wiersza 9
```

Proszę zauważyć, że podając funkcji `set_exception_handler()` nazwę funkcji przekazywaną w parametrze, trzeba umieścić ją w cudzysłowach. Jeżeli o tym zapomnimy, to otrzymamy komunikat, że próbujemy zastosować niezdefiniowaną jeszcze stałą.

Interfejsy

Dodane w PHP 5 interfejsy pozwalają na powiązanie klasy z więcej niż tylko jednym zestawem metod. Interfejsy składają się z pustych funkcji (ich definicje dodane zostaną w implementacji klasy) oraz stałych, jakich chcemy używać w implementacji klasy. Każda klasa implementująca dany interfejs musi tworzyć implementacje wszystkich jego metod. Dzięki temu użytkownicy naszego kodu będą zawsze wiedzieli, jakich metod mogą użyć.

Interfejsy są opisywane dokładniej w rozdziale 2.

Iteratory

Dzięki standardowej bibliotece publicznej (SPL — Standard Public Library) uzyskujemy dostęp do całego zestawu klas i interfejsów pozwalających na manipulowanie i używanie różnych iteratorów. W następnym podpunkcie omawiać będziemy podstawową klasę `Iterator`. Wiele funkcji iteratorów jest bardzo podobnych do funkcji tablicowych.

Interfejs głównego iteratora

Klasa `Iterator` jest wbudowanym interfejsem pozwalającym na iterowanie po wszystkim tym, co pozwala na iterowanie za pomocą instrukcji `foreach`, na przykład po plikach z danego katalogu, wynikach zapytania lub tablicy danych. Klasa ta udostępnia kilka bardzo ważnych funkcji:

- `Iterator::current()` — zwraca bieżący element.
- `Iterator::key()` — zwraca klucz bieżącego elementu.
- `Iterator::next()` — przenosi wskaźnik na następny element.
- `Iterator::valid()` — sprawdza, czy po wywołaniu metody `next()` lub `rewind()` wskaźnik wskazuje jakikolwiek element.

Przedstawione funkcje wstępnie definiują zachowania iteratora, przez co wspomagają używanie takich iteratorów w naszym kodzie. Jednak najważniejszą cechą interfejsu `Iterator` jest to, że jest on podstawowym elementem wszystkich klas iteratorów.

Inne klasy iteratorów

W kolejnych punktach przedstawiać będziemy najważniejsze klasy wywiedzione z klasy `Iterator`. Dokładniejsze opisy tych klas, a także ich bardziej rozbudowaną listę, znaleźć można na stronach dokumentacji biblioteki SPL: *http://www.php.net/~helly/php/ext/spl/main.html*.

DirectoryIterator

Jak sugeruje nazwa, klasa `DirectoryIterator` jest klasą implementującą interfejs `Iterator` i pozwalającą na iterowanie po plikach znajdujących się w danym katalogu. Klasa ta ukrywa zestaw przydatnych funkcji pozwalających na odczytanie różnych informacji o plikach, takich jak czas i data ich modyfikacji, rozmiar, typ, nazwa właściciela, pełna ścieżka do pliku, oraz o innych uprawnieniach związanych z plikiem (nie jest to oczywiście pełna lista).

RecursiveIterator

Klasa `RecursiveIterator` pozwala na iterację rekursywną, czyli automatyczne wywoływanie tej samej funkcji do momentu, aż zostanie spełniony pewien warunek. Funkcje oferowane przez tę klasę umożliwiają sprawdzenie, czy istnieją już iteratory potomne i określenie ich typów.

ArrayIterator

Za pomocą wbudowanej klasy `ArrayIterator` możemy modyfikować wartości i klucze tablicy w czasie iterowania po elementach obiektu. Funkcje takie jak `append()`, `copy()` lub `seek()` pozwalają programiście na rozszerzenie typowego zestawu funkcji tablicowych.

Iteratory są cały czas poprawiane i rozbudowywane, dlatego w języku PHP 5.1 można spodziewać się wprowadzenia w nich wielu zmian.

Konstruktory i destruktory

W PHP 4 można było wywołać pewną metodę klasy przy tworzeniu obiektu — wystarczyło nadać metodzie nazwę identyczną z nazwą klasy. W PHP 5 tę samą funkcję spełnia „magiczna metoda" o nazwie `__construct()`. Jeżeli zostanie ona dodana do klasy, to będzie automatycznie wywoływana na rzecz każdego nowo tworzonego obiektu.

Podobnie, obiekt jest niszczony przez wywołanie metody `__destruct()` dołączonej do definicji klasy. Metoda ta wywoływana jest też automatycznie w sytuacji, gdy nie będzie już żadnej referencji wskazującej na dany obiekt albo w momencie zakończenia pracy przez skrypt PHP.

W rozdziale 2. podamy więcej szczegółowych informacji na temat konstruktorów i destruktorów.

Modyfikatory dostępu

Teraz możemy kontrolować poziom widoczności poszczególnych elementów składowych klasy. Pozwalają na to trzy słowa kluczowe wprowadzone do PHP 5: `public`, `private` i `protected`. Za ich pomocą możemy oznaczać, że metody i pola klasy będą publiczne, prywatne lub chronione.

Poniżej przedstawiamy skrócony opis tych trzech słów kluczowych (więcej szczegółów na ich temat podamy w rozdziale 2.):

- `public` — tak oznaczone metody i pola dostępne są w całym skrypcie. Można z nich korzystać wewnątrz obiektu, ale i poza nim.
- `protected` — metody i pola oznaczone jako chronione dostępne są tylko wewnątrz obiektów danej klasy lub klas z niej wywiedzionych.
- `private` — metody i pola prywatne dostępne są wyłącznie w obiektach danej klasy, co oznacza, że nie można z nich korzystać w obiektach klas wywiedzionych.

Słowo kluczowe final

Słowo kluczowe `final` może być dodawane do całej klasy albo do metod tej klasy. Jeżeli zostanie dodane do klasy, to zablokuje możliwość dalszej rozbudowy tej klasy przez dziedziczenie. Jeżeli jednak zostanie dodane do metody klasy, to uniemożliwi klasom wywiedzionym pokrywanie tej metody. W ten sposób metoda jest zabezpieczana przed ewentualnymi zmianami wprowadzanymi przez innych programistów.

Słowo kluczowe static

Zadeklarowanie pola lub metody klasy z wykorzystaniem słowa kluczowego `static` powoduje połączenie tego pola lub metody z klasą i uniezależnienie od jakiegokolwiek obiektu lub egzemplarza klasy. Do statycznych pól i metod klasy można odwoływać się w całym skrypcie.

Wykorzystanie słowa kluczowego `static` wymusza zachowanie kilku ważnych konwencji. Po pierwsze, dostępu do zmiennych klasy z wnętrza klasy nie uzyskujemy już za pomocą składni `$this->nazwaPola`, ale używamy składni `self::$nazwaPola`. Po drugie, chcąc uzyskać dostęp do statycznego pola z poza klasy, należy użyć składni `nazwaKlasy::$nazwaPola`. Podobnie, dostęp do metody statycznej z wnętrza klasy uzyskamy za pomocą składni `self::nazwaMetody()`, a poza klasą należy używać składni `nazwaKlasy::nazwaMetody()`.

Więcej informacji na temat funkcjonowania słowa kluczowego `static` podamy w rozdziale 2.

Słowo kluczowe abstract

Słowo kluczowe `abstract` stosowane jest do definiowania klas wysokiego poziomu, które muszą zostać odziedziczone przez specjalizowane klasy niższego poziomu. Abstrakcyjnych klas i metod można użyć na przykład w ten sposób:

```
<?php

abstract class getToday {
  public $today = getdate();

  abstract function showCalendar();
}

class monthlyCalendar extends getToday {
  function showCalendar() {
    // wyświetla kalendarz bieżącego miesiąca
  }

class dailyCalendar extends getToday {
  function showCalendar() {
    // wyświetla kalendarz z dzisiejszym dniem
  }

class yearlyCalendar extends getToday {
  function showCalendar() {
    // wyświetla kalendarz bieżącego roku
  }
}

?>
```

Tak zdefiniowana klasa abstrakcyjna oznacza tylko, że „tworzymy kalendarz, ale jego dokładny typ zostanie określony w klasach wywiedzionych". Jak widać, metoda `showCalendar()` została celowo oznaczona jako abstrakcyjna, aby wymusić w klasach wywiedzionych zdefiniowanie tej metody i uzupełnienie jej kodem wyświetlającym określony typ kalendarza. Klasa `getToday` jest niekompletna i jako taka jest doskonałym kandydatem na klasę abstrakcyjną. Metody abstrakcyjne są bardzo podobne do metod interfejsów. One również nie mają żadnego kodu (są puste) i wymagają implementowania w klasach wywiedzionych.

W czasie używania klas abstrakcyjnych nie można też zapominać o następujących regułach:

- Jeżeli w klasie znajduje się metoda abstrakcyjna, to i cała klasa musi zostać zadeklarowana jako abstrakcyjna.
- Nie wolno tworzyć egzemplarzy klas abstrakcyjnych.
- Klasa może zostać oznaczona jako abstrakcyjna nawet wtedy, gdy nie ma w niej ani jednej metody abstrakcyjnej.
- Klasy wywiedzione implementujące metody abstrakcyjne muszą mieć tę samą (lub mniejszą) widoczność co ich klasy bazowe.

Na temat abstrakcji klas i metod będziemy mówić więcej w rozdziale 2.

Funkcje przeciążające metody wbudowane

W PHP 4 w celu wymuszenia kontroli przeciążenia trzeba było wywołać funkcję `overload()`. Język PHP 5 domyślnie pozwala na przeciążanie referencji dowolnej metody lub pola, jeżeli nie odpowiadają one naszym wymaganiom. W ten sposób otrzymujemy bezpośrednią kontrolę nad wykonywanymi operacjami. W czasie przeciążania wykorzystywane są nowe magiczne metody `__get()`, `__set()` i `__call()`.

```
<?php
class overLoad {
  function __get($property) {
    // Sprawdzamy, czy właściwość istnieje w klasie, a jeżeli nie,
    // to wykonujemy adekwatne operacje, na przykład wyświetlamy
    // komunikat o błędzie. Jeżeli jednak właściwość istnieje,
    // to zwracamy jej wartość.
  }

  function __set($property, $value) {
    // Sprawdzamy, czy właściwość istnieje w klasie, a jeżeli nie,
    // to wykonujemy adekwatne operacje, na przykład wyświetlamy
    // komunikat o błędzie. Jeżeli jednak właściwość istnieje,
    // to przypisujemy jej nową wartość.
  }

  function __call($method, $array_of_arguments) {
    // Sprawdzamy, czy metoda istnieje w klasie, a jeżeli nie,
    // to wykonujemy adekwatne operacje, na przykład przekazujemy
    // wywołanie do innej klasy oferującej tę metodę
    // albo wypisujemy komunikat o błędzie. Jeżeli jednak metoda
    // istnieje, to jest wywoływana.
  }

}
?>
```

W rozdziale 2. będziemy dokładniej opisywali mechanizm przeciążania.

Nowe funkcje

Oto wyczerpująca lista wszystkich nowych funkcji wprowadzonych do języka PHP 5, z wyjątkiem tych, które wymagają zastosowania określonych rozszerzeń:

- `array_combine()` — łączy dwie tablice w jedną, przy czym z jednej z tablic pobierane są wartości, a z drugiej klucze.
- `array_diff_uassoc()` — określa różnice między dwoma lub więcej tablicami, wykonując przy tym dodatkowe porównywanie kluczy wykonywane przez funkcję zewnętrzną.
- `array_udiff()` — określa różnice między dwoma lub więcej tablicami, przy czym porównywanie wartości wykonywane jest przez funkcję zewnętrzną.
- `array_udiff_assoc()` — określa różnice między dwoma lub więcej tablicami z dodatkową kontrolą kluczy, przy czym porównywanie wartości wykonywane jest przez funkcję zewnętrzną.
- `array_udiff_uassoc()` — określa różnice między dwoma lub więcej tablicami z dodatkową kontrolą kluczy, przy czym porównywanie wartości i kluczy wykonywane jest przez funkcje zewnętrzne.
- `array_uintersect()` — określa część wspólną dwóch lub więcej tablic, wykorzystując do porównywania danych funkcję zewnętrzną.
- `array_uintersect_assoc()` — określa część wspólną dwóch lub więcej tablic z dodatkową kontrolą kluczy, wykorzystując do porównywania danych funkcję zewnętrzną.
- `array_uintersect_uassoc()` — określa część wspólną dwóch lub więcej tablic z dodatkową kontrolą kluczy wykonywaną przez zewnętrzną funkcję, wykorzystując do porównywania danych funkcję zewnętrzną.
- `array_walk_recursive()` — dla każdego elementu tablicy rekursywnie wywołuje podaną funkcję.
- `convert_uudecode()` — dekoduje ciąg znaków w formacie uudecode.
- `convert_uuencode()` — koduje ciąg znaków do formatu uuencode.
- `curl_copy_handle()` — kopiuje uchwyt cURL wraz z jego wszystkimi parametrami.
- `date_sunrise()` — na podstawie długości i szerokości geograficznej oraz przesunięcia czasowego zwraca czas wschodu słońca.
- `date_sunset()` — na podstawie długości i szerokości geograficznej oraz przesunięcia czasowego zwraca czas zachodu słońca.
- `dba_key_split()` — dzieli klucz z reprezentacji ciągu znaków na reprezentację tablicową.
- `dbase_get_header_info()` — zwraca informacje o strukturze kolumny w bazie danych dBase.

- `dbx_fetch_row()` — pobiera wiersze z wyniku zapytania, ale jeżeli w zapytaniu nie ma parametru `DBX_RESULT_UNBUFFERED`, to funkcja musi poczekać.
- `fbsql_set_password()` — zmienia hasło podanego użytkownika.
- `file_put_contents()` — równoważna jest z otwarciem pliku, zapisaniem danych i zamknięciem pliku.
- `ftp_alloc()` — wysyła do serwera FTP polecenie `ALLO`, które przygotowuje miejsce na przesyłany plik.
- `get_declared_interfaces()` — zwraca dowolny interfejs zadeklarowany w skrypcie.
- `get_headers()` — zwraca nagłówki przesyłane przez serwer w odpowiedzi na zapytanie HTTP.
- `headers_list()` — zwraca listę wysłanych (lub gotowych do wysłania) nagłówków odpowiedzi.
- `http_build_query()` — na podstawie tablicy tworzy zapytania sformatowane zgodnie z wymogami adresów URL.
- `ibase_affected_rows()` — zwraca liczbę wierszy, na jakie miało wpływ ostatnie zapytanie (Interbase).
- `ibase_backup()` — inicjuje w menedżerze usług zadanie wykonywania kopii bezpieczeństwa i kończy pracę (Interbase).
- `ibase_commit_ret()` — zatwierdza transakcję i wraca bez jej zamykania (Interbase).
- `ibase_db_info()` — zwraca informacje o bazie danych (Interbase).
- `ibase_drop_db()` — usuwa bazę danych (Interbase).
- `ibase_errcode()` — zwraca kod błędu (Interbase).
- `ibase_free_event_handler()` — zwalnia zarejestrowaną procedurę obsługi błędów (Interbase).
- `ibase_gen_id()` — inkrementuje generator i zwraca wartość po inkrementacji (Interbase).
- `ibase_maintain_db()` — wykonuje na serwerze bazodanowym polecenie konserwacji (ang. maintenance) (Interbase).
- `ibase_name_result()` — nadaje nazwę zbiorowi wyników (Interbase).
- `ibase_num_params()` — zwraca liczbę parametrów podanego zapytania (Interbase).
- `ibase_param_info()` — zwraca informacje o parametrach podanego zapytania (Interbase).
- `ibase_restore()` — inicjuje w menedżerze usług zadanie odtwarzania danych z kopii bezpieczeństwa i kończy pracę (Interbase).
- `ibase_rollback_ret()` — cofa operację transakcji bez jej zamykania (Interbase).
- `ibase_server_info()` — zwraca informacje na temat bazy danych (Interbase).
- `ibase_service_attach()` — tworzy połączenie z menedżerem usług (Interbase).

- `ibase_service_detach()` — zamyka połączenie z menedżerem usług (Interbase).
- `ibase_set_event_handler()` — rejestruje funkcję użytkownika, która będzie wywoływana w związku z określonym zdarzeniem (Interbase).
- `ibase_wait_event()` — czeka na przesłanie z bazy danych informacji o wystąpieniu zdarzenia (Interbase).
- `iconv_mime_decode()` — dekoduje pole nagłówka MIME.
- `iconv_mime_decode_headers()` — dekoduje jednocześnie kilka pól nagłówków MIME.
- `iconv_mime_encode()` — tworzy pole nagłówka MIME.
- `iconv_strlen()` — zwraca liczbę znaków z ciągu znaków.
- `iconv_strpos()` — zwraca pozycję pierwszego wystąpienia podanego znaku w zadanym ciągu znaków.
- `iconv_strrpos()` — zwraca pozycję ostatniego wystąpienia podanego znaku w zadanym zakresie ciągu znaków.
- `iconv_substr()` — zwraca wycinek ciągu znaków.
- `idate()` — zapisuje lokalny czas i datę w postaci liczby całkowitej.
- `imagefilter()` — dodaje do obrazu określony filtr.
- `image_type_to_extension()` — zwraca rozszerzenia pliku z obrazem.
- `imap_getacl()` — zwraca wartość ACL zadanej skrzynki pocztowej.
- `ldap_sasl_bind()` — łączy zasób do katalogu LDAP, wykorzystując przy tym SASL.
- `mb_list_encodings()` — zwraca tablicę z obsługiwanymi rodzajami kodowania.
- `pcntl_getpriority()` — zwraca priorytet danego numeru PID.
- `pcntl_wait()` — wstrzymuje pracę bieżącego procesu do czasu zakończenia pracy procesu potomnego i zwraca identyfikator tego procesu.
- `pg_version()` — zwraca zapisane w tablicy nazwę klienta, protokół i numer wersji serwera.
- `php_strip_whitespace()` — usuwa z kodu źródłowego wszystkie komentarze, białe znaki i znaki nowego wiersza i zwraca wynik tych operacji.
- `proc_nice()` — zmienia priorytet bieżącego procesu, dodając do niego określoną liczbę.
- `pspell_config_data_dir()` — ustala umiejscowienie plików z danymi językowymi.
- `pspell_config_dict_dir()` — ustala umiejscowienie plików z główną listą słów.
- `setrawcookie()` — równoważna z funkcją `setcookie()`, ale dane ciasteczka nie są kodowane do formatu URL.
- `snmp_read_mib()` — odczytuje i parsuje plik MIB i wstawia go do aktywnego drzewa MIB.
- `sqlite_fetch_column_types()` — podaje typy kolumn określonej tabeli.
- `str_split()` — dzieli ciąg znaków, doprowadzając go do postaci tablicy znaków.

- `stream_copy_to_stream()` — kopiuje dane pomiędzy strumieniami i zwraca ilość skopiowanych danych.
- `stream_get_line()` — działa podobnie do funkcji `fgets()`, ale pozwala na podanie znaku rozdzielającego.
- `stream_socket_accept()` — akceptuje połączenie w gnieździe utworzone przez wcześniejsze wywołanie funkcji `stream_socket_server()`.
- `stream_socket_client()` — otwiera połączenie strumieniowe z komputerem w internecie lub w domenie uniksowej.
- `stream_socket_get_name()` — zwraca nazwę połączenia.
- `stream_socket_recvfrom()` — odbiera z gniazda dane do określonej długości.
- `stream_socket_sendto()` — wysyła dane do podanego gniazda niezależnie od tego, czy w tym gnieździe ustanowiono jakieś połączenie czy też nie.
- `stream_socket_server()` — tworzy strumień na podanym serwerze.
- `strpbrk()` — szuka w ciągu znaków dowolnego znaku z podanej listy znaków i zwraca pozostałą część ciągu, zaczynając od pierwszego znalezionego znaku.
- `substr_compare()` — porównuje dwa ciągi znaków. Przy porównaniu brane jest pod uwagę podane przesunięcie i ewentualne wyłączenie rozróżniania wielkości liter.
- `time_nanosleep()` — wstrzymuje wykonywanie skryptu na określoną liczbę sekund i nanosekund.

Więcej informacji na temat tych funkcji znaleźć można na stronie *http://www.php.net*.

Inne zmiany w PHP 5

Oprócz zmian w funkcjonowaniu mechanizmów obiektowych i całej armii nowych funkcji w języku PHP 5 wprowadzono jeszcze wiele innych poprawek.

Zmiany w konfiguracji

W języku PHP wprowadzono wiele zmian dotyczących pliku konfiguracyjnego *php.ini*. Zmiany te w szczegółach opisywane są w rozdziale 5., ale już tutaj podamy ich skróconą listę:

- `mail.force_extra_parameters`
- `register_long_arrays`
- `session.hash_function`
- `session.hash_bits_per_character`
- `zend.ze1_compatibility_mode`

Tych kilka nowych opcji pozwala nam uzyskać nieco większą kontrolę nad środowiskiem pracy interpretera PHP, a tym samym otrzymać nieco większą swobodę tworzenia kodu.

MySQLi

Jak zapewne wszyscy wiedzą, serwer MySQL doskonale współpracuje z językiem PHP w zakresie tworzenia dynamicznych witryn WWW korzystających z baz danych. W związku z tym pojawienie się rozszerzenia MySQLi (MySQL improved — usprawnione MySQL) bardzo ułatwiło pracę każdemu twórcy stron WWW.

Ustawienia konfiguracji

W pliku *php.ini* znajdziemy kilka ustawień dotyczących rozszerzenia MySQLi. Oto ich skrócony opis:

- `mysqli.max_links` — ustala maksymalną liczbę połączeń z serwerem MySQL przypadającą na jeden proces.
- `mysqli.default_port` — ustala domyślny port TCP/IP przeznaczony na połączenia z serwerem MySQL.
- `mysqli.default_socket` — ustala domyślną nazwę gniazda przeznaczonego na połączenia z serwerem MySQL.
- `mysqli.default_host` — ustala domyślną nazwę komputera, na którym pracuje z serwer MySQL.
- `mysqli.default_user` — ustala domyślną nazwę użytkownika używaną przy połączeniach z serwerem MySQL.
- `mysqli.default_pw` — ustala domyślne hasło użytkownika używane przy połączeniach z serwerem MySQL.

Wbudowane klasy i właściwości

Rozszerzenie `mysqli` dodaje do języka nowy zestaw klas i właściwości wbudowanych, których możemy użyć w swoich skryptach PHP. Jeżeli nasz skrypt ma naturę proceduralną i nie korzystamy w nim z obiektów, to nowe metody mogą być też używane jako zwykłe funkcje. Chcąc skorzystać z funkcji w programie proceduralnym, wystarczy przed nazwą metody dopisać przedrostek `mysqli_`. Na przykład w wersji obiektowej przygotowanie nowego połączenia może wyglądać następująco:

```
<?php

$connect = new mysqli("serwer", "uzytkownik", "haslo", "nazwa_bazy");

// wywołanie metody klasy mysqli

$connect->close();

?>
```

To samo zadanie można zrealizować też w tradycyjnym programie proceduralnym. Wystarczy zastosować poniższy kod:

```
<?php

$connect = mysqli_connect("serwer", "uzytkownik", "haslo", "nazwa_bazy");

// wywołanie funkcji z wykorzystanie przedrostka mysqli_

mysqli_close($connect);

?>
```

W kolejnych punktach opisywać będziemy poszczególne elementy rozszerzenia dodane do PHP 5.

Klasa mysqli

Ta klasa zajmuje się obsługą połączeń z serwerem MySQL. Konstruktor klasy tworzy nowe połączenie z serwerem i jednocześnie przygotowuje nowy obiekt typu `mysqli`. Klasy tej można użyć do manipulowania istniejącym połączeniem, pobierania o nim informacji albo wykonywania podstawowych funkcji zapytań.

Klasa ta udostępnia następujące metody:

- `autocommit` — ustala, czy transakcje mają być automatycznie zatwierdzane w bazie danych.
- `change_user` — przełącza połączenie na innego użytkownika.
- `character_set_name` — zwraca domyślny ciąg znaków.
- `close` — zamyka połączenie z bazą danych.
- `commit` — zatwierdza transakcję w bazie danych.
- `connect` — otwiera nowe połączenie do serwera MySQL (wywołuje konstruktor klasy `mysqli`).
- `debug` — do debugowania wykorzystuje bibliotekę Freda Fisha.
- `dump_debug_info` — zwraca wszystkie informacje debugowania uzyskane za pomocą biblioteki Freda Fisha.
- `get_client_info` — odczytuje informacje na temat klienta.
- `get_host_info` — odczytuje informacje na temat połączenia.
- `get_server_info` — odczytuje informacje na temat serwera MySQL.
- `get_server_version` — odczytuje wersję serwera MySQL.
- `info` — zwraca informacje na temat ostatnio wykonanego zapytania.
- `init` — inicjuje obiekt przed wywołaniem funkcji `real_connect`.
- `kill` — zabija wątek MySQL.

- more_results — wyszukuje wyników z wykonanej wcześniej funkcji multi_query.
- multi_query — wykonuje jedno lub więcej zapytań.
- next_result — zwraca następny wynik z wykonanej wcześniej funkcji multi_query.
- options — zmienia lub ustala opcje połączenia dla obiektu real_connect.
- ping — sprawdza (pinguje) połączenie z serwerem, a jeżeli zostało zerwane, to ustanawia je ponownie.
- prepare — przygotowuje do wykonania kod zapytania SQL.
- query — wykonuje zapytanie.
- real_connect — tworzy połączenie z serwerem MySQL i pozwala na ustalenie dodatkowych opcji lub parametrów.
- real_escape_string — zwraca ciąg znaków z oznaczonymi znakami specjalnymi, którego można użyć w instrukcji SQL.
- rollback — cofa operacje bieżącej transakcji.
- select_db — wybiera podaną bazę danych i oznacza ją jako aktywną.
- set_charset — ustala zestaw znaków, jaki ma być używany w połączeniu.
- ssl_set — ustala parametry SSL i włącza bezpieczne połączenie.
- stat — zwraca informacje na temat stanu systemu.
- stmt_init — inicjuje instrukcję do użycia z funkcją mysqli_stmt_prepare.
- store_result — przenosi wyniki ostatnio wykonanego zapytania.
- thread_safe — informuje, czy zapewnione jest bezpieczeństwo pracy w wątkach.
- use_result — przenosi niezbuforowany zbiór wyników ostatnio wykonanego zapytania.

Klasa ta udostępnia też poniższe właściwości:

- affected_rows — podaje liczbę wierszy wykorzystanych w ostatnio wykonanej operacji serwera MySQL.
- client_info — zwraca numer wersji klienta MySQL w postaci ciągu znaków.
- client_version — zwraca numer wersji klienta MySQL w postaci liczby całkowitej.
- errno — zwraca kod błędu ostatnio wywołanej funkcji.
- error — zwraca ciąg znaków błędu ostatnio wywołanej funkcji.
- field_count — zwraca liczbę kolumn ostatnio wykonanego zapytania.
- host_info — zwraca ciąg znaków reprezentujący typ ustanowionego połączenia.
- info — zwraca informacje o ostatnio wykonanym zapytaniu.
- insert_id — zwraca automatycznie wygenerowany identyfikator używany w ostatnim zapytaniu.
- protocol_version — zwraca wersję używanego protokołu MySQL.

- `sqlstate` — zwraca ciąg znaków zawierający kod błędu SQLSTATE związany z ostatnim błędem.
- `thread_id` — zwraca identyfikator wątku bieżącego połączenia.
- `warning_count` — zwraca liczbę ostrzeżeń wygenerowanych w czasie wykonywania ostatniej instrukcji SQL.

Klasa mysqli_stmt

Ta klasa zajmuje się obsługą zapytań przygotowanych, czyli instrukcji SQL tymczasowo przechowywanych na serwerze MySQL i wywoływanych w razie potrzeby. Przykładem zapytania przygotowanego może być instrukcja `SELECT * FROM customets WHERE lastname = ?`. W momencie, gdy będziemy chcieli wykonać to polecenie, wystarczy tylko dopisać do niego wartość parametru `lastname`. Jeżeli metod tej klasy będziemy chcieli użyć jako funkcji, to przed nazwą każdej metody dopisać trzeba przedrostek `mysql_stmt_` (na przykład `mysqli_stmt_close`).

Klasa `mysqli_stmt` udostępnia następujące metody:

- `bind_param` — wiąże zmienne z zapytaniem przygotowanym.
- `bind_result` — wiąże zmienne z elementami wyniku zapytania przygotowanego.
- `close` — zamyka zapytanie przygotowane.
- `data_seek` — przechodzi do podanego wiersza wyniku zapytania.
- `execute` — wykonuje zapytanie przygotowane.
- `fetch` — pobiera wynik zapytania przygotowanego i zapisuje go do powiązanych zmiennych.
- `free_result` — zwalnia wyniki zapytania przechowywane w pamięci.
- `prepare` — przygotowuje zapytanie SQL.
- `reset` — zeruje ustawienia zapytania przygotowanego.
- `result_metadata` — odczytuje metadane zbioru wyników zapytania przygotowanego.
- `send_long_data` — wysyła dane podzielone na kawałki.
- `store_result` — buforuje cały zbiór wyników zapytania przygotowanego.

Klasa ta udostępnia też poniższe właściwości:

- `affected_rows` — podaje liczbę wierszy wykorzystanych przy ostatnim uruchomieniu zapytania.
- `errno` — zwraca kod błędu ostatniego uruchomienia zapytania przygotowanego.
- `error` — zwraca tekst błędu ostatniego uruchomienia zapytania przygotowanego.
- `param_count` — zwraca liczbę parametrów zapytania przygotowanego.
- `sqlstate` — zwraca ciąg znaków zawierający kod błędu SQLSTATE związany z ostatnim błędem zapytania przygotowanego.

Klasa mysqli_result

Ta klasa reprezentuje pobrane z bazy danych wyniki zapytania. Klasy tej można użyć do manipulowania i wyświetlania wyników zapytania.

Klasa `mysqli_result` udostępnia następujące metody:

- `close` — zamyka zbiór wyników (w wersji proceduralnej nazywa się `mysqli_free_result`).
- `data_seek` — przenosi wewnętrzny wskaźnik wyniku.
- `fetch_field` — odczytuje z wyników informacje o kolumnie.
- `fetch_fields` — odczytuje z wyników informacje o wszystkich kolumnach.
- `fetch_field_direct` — odczytuje z wyników informacje o określonej kolumnie.
- `fetch_array` — odczytuje z wyników wiersz, traktując go jako tablicę asocjacyjną lub tablicę numeryczną.
- `fetch_assoc` — odczytuje z wyników wiersz, traktując go jako tablicę asocjacyjną.
- `fetch_object` — odczytuje z wyników wiersz, traktując go jako obiekt.
- `fetch_row` — odczytuje z wyników wiersz, traktując go jako tablicę wyliczeniową.
- `field_seek` — przenosi wskaźnik wyniku na określone pole.

Klasa ta udostępnia również następujące właściwości:

- `current_field` — zwraca wskazanie na bieżące pole.
- `field_count` — zwraca liczbę pól w zbiorze wyniku.
- `lengths` — zwraca tablicę długości komun.
- `num_rows` — zwraca liczbę wierszy ze zbioru wyników.

Jak widać, nowe klasy rozszerzenia `mysqli` pozwalają na tworzenie bardzo efektywnego kodu. Co więcej, dają nam dużo większe możliwości działania niż funkcje MySQL udostępniane przez PHP 4.

Obsługa XML

W PHP 5 zastosowano o wiele lepszą obsługę standardu XML niż ta oferowana przez PHP 4. Dodano kilka nowych rozszerzeń związanych z obsługą XML, które zostały napisane z wykorzystaniem biblioteki *libxml2*, co poprawiło zgodność ze standardem i ułatwiło zarządzanie kodem.

- **DOM** — nowy zestaw funkcji zastępujących funkcje DOMXML znane z PHP 4; rozszerzenie to zostało przygotowane tak, żeby możliwa była obsługa forsowanego przez konsorcjum W3C standardu DOM Level 2.
- **XSL** — wcześniej rozszerzenie to nazywało się XSLT; przeznaczone jest do automatyzacji przekształcenia jednego dokumentu XML na inny za pomocą standardu arkuszy stylów XSL.

- **SimpleXML** — ten zestaw funkcji pozwala na szybkie i proste pobranie danych z pliku XML; później można manipulować, wyświetlać i porównywać atrybuty i elementy, wykorzystując standardowe iteratory tablicowe i instrukcję `foreach`.
- **SOAP** — rozszerzenie SOAP pozwala na tworzenie serwerów i klientów SOAP; wymaga wcześniejszego zainstalowania biblioteki GNOME XML (*libxml*).

Rozszerzenia obsługi standardu XML omawiane będą dokładniej w rozdziale 8.

Rozszerzenie Tidy

PHP 5 obsługuje też bibliotekę *Tidy*, która dostępna jest pod adresem *http://tidy.sourceforge.net*. Biblioteka ta pomaga twórcy stron WWW w dopracowaniu i oczyszczeniu kodu HTML. Rozszerzenie to dostępne jest w ramach biblioteki PECL, którą znajdziemy pod adresem *http://pecl.php.net/package/tidy*. Pełna lista klas i funkcji udostępnianych przez bibliotekę *Tidy* wypisana została w podręczniku języka PHP na stronie *http://us2.php.net/tidy*.

SQLite

Bibliotekę *SQLite* wprowadzono już w późniejszych wersjach języka PHP 4, ale ostateczną postać przybrała ona dopiero w języku PHP 5. SQLite to swego rodzaju miniserwer SQL. Do języka PHP 5 dodanych zostało wiele klas i funkcji, a samo rozszerzenie instalowane jest wraz z interpreterem języka. Więcej informacji na temat rozszerzenia SQLite znaleźć można na stronie *http://sqlite.org*.

Podsumowanie

Oczywiście największym i najlepszym usprawnieniem w stosunku do języka PHP 4 jest wprowadzenie całkowicie nowego modelu obiektów, ale poprawionych zostało też wiele innych elementów, które razem bardzo podnoszą komfort pracy programisty.

Jedną z najwspanialszych cech języka PHP jest to, że cały czas jest zmieniany i rozbudowywany poprzez uzupełnianie go o kolejne usprawnienia (podobnie jak większość projektów o otwartych źródłach). Drobne usprawnienia składające się na wersję PHP 5 pozwalają na skuteczniejszą pracę z serwerem MySQL, poprawienie kodu i znacznie lepsze wykorzystanie możliwości, jakie daje technologia XML. Najlepszym usprawnieniem jest niewątpliwie nowy model obiektów. Jeżeli zamiast tradycyjnych programów proceduralnych zaczniemy korzystać z obiektów, to na pewno od razu zmieni się nasz sposób postrzegania kodu. W następnym rozdziale podamy więcej informacji na temat tego, jak można uzyskać najlepsze efekty za pomocą nowego modelu obiektów PHP.