Technologie Internetu

PHP

Aleksander Denisiuk

denisjuk@pjwstk.edu.pl

Polsko-Japońska Wyższa Szkoła Technik Komputerowych

zamiejscowy ośrodek dydaktyczny w Gdańsku

ul. Brzegi 55

80-045 Gdańsk

PHP

Najnowsza wersja tego dokumentu dostępna jest pod adresem http://users.pjwstk.edu.pl/~denisjuk/

Wprowadzenie do PHP

- PHP: Hypertext Preprocessor
- `http://www.php.net/manual/pl`

Hello, World!

```
<html><head>
  <title>PHP Test</title>
</head><body>
  <?php echo '<p>Hello World</p>'; ?>
</body></html>
```

```
<html><head>
  <title>PHP Test</title>
</head><body>
  <p>Hello World</p>
</body></html>
```

Model Obiektowy

```php
<?php
    include('include/main.php');
    $main = new Main();
    $main->printHeader();
    $main->printBody();
    $main->printFooter();
?>
```

Oddzielenie logiki od prezentacji

- Szablony
  - Savant
  - Open Power Template
  - PHPTAL
  - Smarty

  (np. Smarty)

Smarty. Przykład

```
<?php $smarty->assign_by_ref('colors',
  array(1=>'red',2=>'blue',3=>'green')?>;
  $smarty->assign('wybrano',2);

<p>Masz {$colors|@count} opcji do wyboru:</p>
  {html_options name=kolorki
    options=$colors selected=$wybrano}
{* komentarz  *}
```

Typy Danych

- Proste
  - `boolean`
  - `integer`
  - `float, double`
  - `string`
- Złożone
  - `array`
  - `object`
- Specjalne
  - `resource`
  - `NULL`

Typizacja dynamiczna

```
$x=0.1;
$x=array();
```

Tablice

- sekwencyjne

```php
<?php
$numerki[0] = "Pierwszy";
$numerki[1] = "Drugi";
$numerki[] = "Trzeci";
echo $numerki[2];
?>
```

- assocjacyjne

```php
<?php
$osoba["imie"] = "Aleksander";
$osoba["adres"] = "denisjuk@pjwstk.edu.pl";
echo "<a href='mailto:".$osoba['adres'].
^^I       "'>$osoba[nazwisko]</a>"
?>
```

Łańcuchy znaków

```php
<?php
echo 'this is $Sstring\n';
echo "this is $Sstring\n";
echo <<<EOT
   $tablica[1]
   $tablica[prim]
     EOT;
echo <<<'EOT'
   $tablica[1]
   $tablica[prim]
     EOT;
?>
```

Zasięg zmiennych

```php
<?php
$a = 1;
include 'b.inc';
function test(){
    echo $a;
}

test();
?>
```

Zmienne globalne (global)

```
<?php
$a = 1; $b = 2;
function Suma1(){
    global $a, $b;
    $b = $a + $b;
}
function Suma2(){
    $GLOBALS['b'] =
^^I$GLOBALS['a'] + $GLOBALS['b'];
}

?>
```

Zmienne superglobalne

- $GLOBALS
- $_SERVER
- $_GET
- $_POST
- $_FILES
- $_COOKIE
- $_SESSION
- $_REQUEST
- $_ENV

Zmienne statyczne (`static`)

```php
<?php
function licznik(){
    static $a = 0;
    return $a++;
}

for($i=1;$i<10;$i++){
   echo licznik();
}
?>
```

- brak możliwości zresetowania licznika

Referencje

```php
<?php
$a =& $b;
$tygrys =& new Kot();

function zaczipuj (&$zwierze){...}
zaczipuj ($tygrys);

function &druh ($osoba){
   ...
   return $friend;
}
$pies =& druh($kot);
?>
```

Czym nie są referencje

```
<?php
function zaczipuj (&$zwierze){
    $zwierze =& $GLOBALS["kot"];
}
zaczipuj($kot);
?>
```

- Kot nie zostanie zaczipowany!

`$this`

- W metodach obiektowych `$this` jest zawsze referencją do obiektu wywołującego daną metodę.
- W metodach statycznych `$this` nie jest określony.

Przestrzenie imion

- Problemy:
  - Kolizje imion
  - Zbyt dlugie i nieczytelne nazwy zmiennych (by zapobiec kolizjom)

Przestrzenie imion. Deklaracja

```php
<?php
namespace OpenDocument;
const Encoding = 'UTF-8';
class Document{ /* ... */ }
function toPDF() { /* ... */  }
?>

<?php
namespace OpenDocument\Calc;
const Encoding = 'UTF-8';
class Document{ /* ... */ }
function toPDF() { /* ... */  }
?>
```

Przestrzenie imion. Rozwiązanie

```php
<?php
namespace OpenDocument;

toPDF(); //  OpenDocument\toPDF();
Document::method(); // metoda method() klasy OpenDocument\Document
echo Encoding; // OpenDocument\Encoding

Calc\toPDF(); //  OpenDocument\Calc\toPDF();
Calc\Document::method(); // metoda method()
^^I    // klasy OpenDocument\Document\Calc
echo Calc\Encoding; // OpenDocument\Calc\Encoding

\OpenDocument\Calc\toPDF(); //  OpenDocument\Calc\toPDF();
\OpenDocument\Calc\Document::method(); // metoda method()
^^I    // klasy OpenDocument\Document\Calc
echo \OpenDocument\Calc\Encoding; // OpenDocument\Calc\Encoding
?>
```

Formularze

```
<form action="action.php?sid=12092371" method="post">
 <p>Login: <input type="text" name="name" /></p>
 <p>Password:
    <input type="password" name="pswrd" /></p>
 <p><input type="submit" name="sbmt" value="ok"/></p>
</form>

<?php
    $_GET['sid']; //12092371
    $_POST['name'];
    $_POST['pswrd'];
    $_POST['sbmt']; //ok
?>
```

Sesje

```php
<?php session_start();
  if (empty($_SESSION['count'])) {
 $_SESSION['count'] = 1;
} else {
 $_SESSION['count']++;
}
  echo SID; //$

  session_destroy();
?>
```

- BOM

Obiektowy model PHP5. Deklaracja klasy

```php
<?php
class Zwierze{
    // property declaration
    private $lap = 4;

    // method declaration
    public function getIloscLap() {
        return $this->lap;
    }
}

$kot = new Zwierze;
?>
```

Konstruktor i destruktor

```php
<?php
class Zwierze{
  private $lap=4;
  function __construct($x) {
    $this->lap = $x;
  }
  function __destruct() {
    echo "brak dobrego przykladu";
  }
}
$tukan = new Zwierze(2);
?>
```

- Dla wstecznej kompatybilności możliwy jest konstruktor `Zwierze`

Obiekty a referencje

- Obiekty zawsze są przekazywane przez referencję
- Nadanie wartości obiektu też przez referencję
- Utworzenie kopii obiektu:

```
$dolly=clone $owca
```

- Jeżeli określona, zostanie wywołana na kopii metoda

```
__clone()
```

Dziedziczenie

```php
<?php
class Zwierze{
  protected $lap;
  public function getIloscLap(){
    return $this->lap = $x;
  }
}

class Kot extends Zwierze{
  public function getIloscLap() {
     return parent::getIloscLap();
  }
  public function __construct(){
    $lap=4;
  }
}
```

Stałe klas

```php
<?php
class OpenDocument{
    const encoding = 'UTF-8';
    function showEncoding() {
        echo  self::encoding;
    }
}

echo OpenDocument::encoding;
$classname = "OpenDocument";
echo $classname::encoding; // od PHP 5.3.0
$class = new OpenDocument();
$class->showConstant();
echo $class::encoding; // od PHP 5.3.0
?>
```

Static

```php
<?php
class OpenDocument{
     static encoding = 'UTF-8';
     static function toEPS($picture) {...}
     public function showEncoding(){
       echo $self::encoding
     }
}

echo OpenDocument::encoding;
OpenDocument::encoding='iso8859-2'
OpenDocument::toEPS=('obraz.pdf');
?>
```

Klasy abstrakcyjne

```php
<?php
abstract class Figura{
    abstract protected function pole();
    public function printPole() {
        print $this->pole();
    }
}
class Kolo extends Figura{
    const Pi=3.1415926;
    private $promien;
    protected function pole() {
        return Pi*$promien;
    }
    public function __construct($r) {
       $this->promien=$r;
    }
}
?>
```

Interfejsy

- Pozwalają na definicję funkcji, które klasa będzie implementowała, bez implementacji tych funkcji
- Interfejsy się deklaruje podobno do klas, za pomocą słowa `interface`
- Wszystkie metody powinny być publiczne (`public`)
- Dopuczszalne jest dziedziczenie interfejsów (`extends`)
- Interfejsy mogą mieć stałe (`const`)
- Klasa implementująca deklaruję się za pomocą operatora `implements`
- Klasa może implementować kilka interfejsów (przecinek)
- Wszystkie metody powinny być zaimplementowane w implementującej klasie

Przykład interfejsów

```php
<?php
interface displayable{
   function display();
}

interface printable{
   function doprint();
}

class foo implements displayable, printable{
   function display() { ... }
   function doprint() { ... }
}
?>
```

Przeciążenie metod

- `mixed __call ( string $name , array $arguments )` — przy dostępie do nieokreślonej metody klasy
- `mixed __callStatic ( string $name , array $arguments )` — przy dostępie do nieokreślonej statycznej metody klasy

Przeciążenie metod. Przykład

```php
<?php class Magic {
  function __call($name,$arguments) {
    if($name=='foo') {
      if(is_int($arguments[0]))
        $this->foo_for_int($arguments[0]);
      if(is_string($arguments[0]))
        $this->foo_for_string($arguments[0]);
    }
  }
  private function foo_for_int($x) { ... }
  private function foo_for_string($x) { ... }
}
$x = new Magic();
$x->foo(3);
$x->foo("3"); ?>
```

Przeciążenie właściwości

- `__set()` — przy zapisywaniu danych do nieokreślonej właściwości.
- `__get()` — przy odczytywaniu danych z nieokreślonej właściwości.
- `__isset()` — przy wywołaniu `isset()` lub `empty()` do nieokreślonej właściwości.
- `__unset()` — przy wywołaniu `unset()` do nieokreślonej właściwości.

final

```php
<?php
class BaseClass {
    final public function nieprzykrywalna(){
        ...
    }
}
final class KlasaNiedziedziczalna{
    ...
}
?>
```

Late Static Bindings

```php
<?php
class A {
    public static function who() {
        echo __CLASS__;}
    public static function test() {
        static::who(); // static::
    }
}
class B extends A {
    public static function who() {
        echo __CLASS__;}
}

B::test();
?>
```

Serializacja

```php
<?php
// classa.inc:
   class A { ...   }

// page1.php:
   include("classa.inc");
   $a = new A;
   $s = serialize($a);
   file_put_contents('store', $s);

// page2.php:
   include("classa.inc");
   $s = file_get_contents('store');
   $a = unserialize($s);
?>
```

Wyjątki

```php
<?php
function readConf($cfgFile){
    $resultArr = array();
    if (file_exists($cfgFile)) {
        return parse_ini_file($cfgFile);
    } else {
        throw new Exception('Brak pliku: '.$cfgFile);
    }
}
try {
    $conf = readConf();
}
catch(Exception $e){//obsuga bledu, np.:
    echo $e->getMessage().'<br>';
    echo $e->getTraceAsString();
}//$
?>
```

Funkcje anonimowe. Motywacja

```php
<?php
function _compare($a, $b){
    return $b - $a;
} // end _compare();
 usort($tablica, '_compare');
?>
```

- funkcja `_compare` jest wywołana tylko jeden raz, ale musi mieć własną nazwę
- pomysł na funkcje anonimowe

Funkcje anonimowe

```php
<?php
$funkcja = function($x, $y){
    return $x - $y;};
echo $funkcja(5, 3);

usort($tablica, $funkcja);

usort($tablica, function($a, $b){
    return $a - $b;});
?>
```

Zmiennie wolne i domknięcia

- $f(x, y) = x + y$
- $f(x) = x + y$, y — zmienna *wolna*
- Zmienne wolne nie są ani agrumentami funkcji, ani jej lokalnymi zmiennymi
- Zmienne wolne otrzymują wartości z kontekstu nadrzędego, poprzez mechanizm *domknięć*
- Zmienne wolne definiuje się słowem kluczowym `use`

Domknięcia

```php
<?php
$products = array(
 array('name' => 'Gruszki', 'amount' => 23),
 array('name' => 'Rodzynki', 'amount' => 38)
);

$total = 0;

array_walk($products, function($element)
                                  use(&$total){
   $total += $element['amount'];
});

echo 'Razem: '.$total;
```

Domknięcia. Fabryka funkcj

```php
<?php
function returnClosure($argument)
{
    return function($x) use($argument)
    {
        return $x + $argument;
    };
} // end returnClosure();

$closureOne = returnClosure(7);
$closureTwo = returnClosure(5);

echo $closureOne(6).PHP_EOL;
echo $closureTwo(6).PHP_EOL;
```