Въведение в PHP

Низове

Георги Гроздев
g.grozdev@viscomp.bg

Атанас Василев
a.vasilev@viscomp.bg

www.viscomp.bg

About us

- Лектори
  - Георги Гроздев – g.grozdev@viscomp.bg
  - Атанас Василев – a.vasilev@viscomp.bg

- Инфо и слайдовете ще намерите на:
  - http://phplab.viscomp.bg

viscomp
we develop the web

Масиви

- Подредена колекция от елементи
- Всеки елемент има стойност и се идентифицира с ключ, който е уникален за масива
- Ключовете могат да са както целочислени, така и низове

Масиви

- Индексирани масиви
  - Целочислени ключове с основа 0
  - Елементите се идентифицират по позиция

```
$a = array('apple', 'banana', 301, array());

$b = array(
    0 => 'apple',
    1 => 'banana',
    2 => 301,
    3 => array(),
);
```

  - Отрицателните числа са валидни индекси и не обозначават относителна позиция от края на масива (както е при Perl)

Масиви

- Асоциативни масиви
  - Ключовете са низове
  - Низови стойности, идентични с целочислени такива (без водеща нула) се третират като целочислени:
    - $array[3] и $array['3'] сочат към един и същи елемент, докато $array['03'] - към друг.
  - Елементите се достъпват по ключ

Масиви

```
$employee = array(
    'name' => 'Jane',
    'gender' => 'female',
    'birthdate' => '1980-11-01',
    'subordinates' => array(
        0 => array(
            'name' => 'Peter',
            'gender' => 'male',
            'birthdate' => '1985-01-23',
            'subordinates' => array(),
        ),
        1 => array(
            'name' => 'Marry',
            'gender' => 'female',
            'birthdate' => '1982-07-12',
            'subordinates' => array(),
        ),
    ),
);
```

Масиви

- На ниско ниво PHP съхранява всички масиви като асоциативни
- Всички елементи в масив имат вътрешна поредност, независима от индексите или ключовете му. Обикновено тази поредност се обуславя от последователността, в която се добавят елементите.

Създаване на масиви

- Чрез конструкцията array(), на която могат да се подадат серия от стойности и евентуално ключове

```
$a = array (10, 20, 30);
print $a[0]; // 10
print $a[1]; // 20

$a = array ('a' => 10, 'b' => 20, 'cee' => 30);
print $a['a']; // 10
print $a['b']; // 20
print $a['cee']; // 30

$a = array (5 => 1, 3 => 2, 1 => 3,);
print $a[5]; // 1
print $a[3]; // 2
print $a[1]; // 3

$a = array();
```

Създаване на масиви

- Чрез два от операторите за масиви (квадратни скоби '[]')
- Промяна на стойност на елемент от масив става чрез директно присвояване на новата стойност

```
$x[] = 10;
$x['aa'] = 11;
print $x[0]; // 10
```

- Ако масивът $x не съществува, той ще бъде създаден автоматично.
- Ако не се укаже ключ за дадена стойност, такъв ще се генерира автоматично:
  - Ако текущият максимален целочислен индекс е положителен, генерираният ключ ще бъде максималния индекс плюс 1
  - Ако текущия максимален индекс е отрицателно число:
    - (PHP 4.3.0): генерираният индекс ще бъде нула (0)
    - (Преди PHP 4.3.0) генерираният индекс ще бъде максималния индекс плюс 1 (както е случая при положителен максимален индекс)

Създаване на масиви

- Използването на число с плаваща запетая ще се изчисли до целочислената стойност, без дробната част
- Използването на TRUE като ключ ще се изчисли като целочислената стойност 1
- Използването на FALSE като ключ ще се изчисли като целочислената стойност 0
- Използването на NULL като ключ ще се изчисли като празен низ ('')
- Масиви или обекти не могат да се ползват като ключове
- Не само литерали могат да се използват като ключове - всякакви изрази, които се изчисляват до стойности от скаларен тип (int, float, boolean, string), плюс специалния тип NULL, са валидни идентификатори за ключ
  - Променливи

```
$person = array();
$name = 'name';
$age = 'age';
$person[$name] = 'John';
$person[$age] = '23';
```

Създаване на масиви

- Променливи

```
$person = array();
$name = 'name';
$age = 'age';
$person[$name] = 'John';
$person[$age] = '23';
```

- Константи

```
define('index',5);
echo $array[index];
// Връща $array[5], а не $array['index'];
```

  - Ако не е дефинирана константа с името, което се подава като ключ на елемент от масив и ключът не е ограден с кавички или апострофи, PHP ще замести ключа с низов литерал със същата стойност.

- Извикване на функция

```
$a = foobar;
echo $array[substr($a, 0,3)];
// Ще върне стойността на $array['foo']
```

- Израз, комбиниращ горните

Създаване на масиви

- Чрез функции, които приемат параметри от друг тип и връщат масив
  - `compact()`
  - `explode()`
  - `range()`
  - `array_fill()`
  - `split()`
  - `preg_split()`
  - `chunk_split()`

Преобразуване в масив

- int, float, string, boolean, resource се преобразуват до масив с един елемент, чийто ключ е нула (0) и стойност равна на скаларната стойност на първоначалния тип

```
$a = array();

$a['int']          = (array) 4;
$a['float']        = (array) 2.5;
$a['string']       = (array) 'a string';
$a['boolean']      = (array) true;
$a['resource']     = (array) fopen('/tmp/filename.txt', 'w');
print_r ($a);
```

- NULL се преобразува в празен масив
- Обект се преобразува в масив като свойствата му стават елементи в масива - имената на стойствата стават ключове в масива при следните ограничения:
  - всички private свойства ще имат добавено името на класа пред ключа в масива
  - всички protected свойства ще имат добавено '*' пред ключа в масива
  - и при двата случая допълнително-добавените низове ще бъдат оградени с т. нар. zero-bite (\0)

Работа с масиви

- Обединяване на масиви
  - array_merge()
    - Стойностите на втория масив се добавят към края на първия
    - Ако двата масива имат еднакви низови ключове, стойността от втория масив ще припокрие тази от първия
    - Цифровите ключове не се припокриват - стойностите на елементите с цифрови ключове от втория масив се добавят в края на резултатния масив

```
$array1 = array("color" => "red", 2, 4);
$array2 = array(
      "a",
      "b",
      "color" => "green",
      "shape" => "trapezoid",
      4
  );
  $result = array_merge($array1, $array2);
  print_r($result);
```

Работа с масиви

- Обединяване на масиви
  - Операторът '+'
    - Стойностите от втория масив се добавят към края на първия
    - Елементи с дублиращи ключове не се презаписват, нито се добавят

```
$a = array (1, 2, 3);
$b = array ('a' => 1, 'b' => 2, 'c' => 3);
$result = ($a + $b);
print_r($result);
$a = array (1, 2, 3, 'a' => 'A');
$b = array ('a' => 1, 2, 3, 'b' => 'B');
print_r($a + $b);
```

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [a] => 1
    [b] => 2
    [c] => 3
)
```

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [a] => A
    [b] => B
)
```

Работа с масиви

- Сравняване на масиви
  - Операторът за равенство (==) връща TRUE ако масивите имат равен брой елементи, с еднакви ключове и стойности, без значение на реда, в който се появяват в масива
  - Операторът за идентичност (===) връща TRUE само ако масивите имат еднакви ключове и стойности в същата последователност.

```
$a = array (1, 2, 3);
$b = array (1 => 2, 2 => 3, 0 => 1);
$c = array ('a' => 1, 'b' => 2, 'c' => 3);

var_dump ($a == $b);   // true
var_dump ($a === $b);  // false
var_dump ($a == $c);   // false
var_dump ($a === $c);  // false

$a = array (1, 2, 3);
$b = array (1 => 2, 2 => 3, 0 => 1);
var_dump ($a != $b);   // false
var_dump ($a !== $b);  // frue
```

Работа с масиви

- Пребройване на елементи
  - **count() и sizeof()**
    - Ако параметърът не е масив, ще бъде върната челочислената стойност '1' с две изключения
      - Ако параметърът е обект, имплементиращ интерфейса Countable, ще бъде върнат броя на свойствата му.
      - Ако параметърът е NULL (или неинициализирана променлива) ще бъде върната стойност '0'

```
echo count(array (1, 2, 4));  // 3

echo count(array());          // 0
echo count(10);               // 1
echo count(@$nonexistent);    // 0
echo count(NULL);             // 0
```

      - Не е коректно count() и sizeof() да се ползват за определяне, дали стойността на дадена променлива е масив:

        `count(10) == 1`.

        Правилният начин е с is_array()

Работа с масиви

- Търсене на елемент в масив
  - По ключ
    - За проверка дали елемент с определен ключ съществува често погрешно се ползва **isset()**:

```
$a = array ('a' => 1, 'b' => 2, 'c' => NULL);

isset ($a['a']);                // true
isset ($a['d']);                // false
isset ($a['c']) ;               // false
array_key_exists('c', $a)       // true
```

    - **array_keys()** - връща масив с всички ключове на даден масив

```
$array = array(0 => 100, "color" => "red");
print_r(array_keys($array));
```

Работа с масиви

- Търсене по стойност
  - **in_array()** - връща TRUE ако елемент с дадена стойност съществува в масива

```
a = array ('a' => NULL, 'b' => 2);
echo in_array (2, $a); // true
```

  - **array_search()** - връща ключа на елемент с дадена стойност.
    - Ако има повече елементи с дадената стойност, само ключа на първия ще бъде върнат
    - Тази функция може да върне стойност '0', която в определен контекст да се преобразува до FALSE, така че трябва да проверяваме с оператор за идентичност (===), ако искаме само да проверим дали дадената стойност съществува

```
$array = array(
    0 => 'blue',
    1 => 'red',
    2 => 'green',
    3 => 'red',
);

$key = array_search('green', $array); // $key = 2;
$key = array_search('blue', $array);   // $key = 0;
```

Работа с масиви

- Търсене по стойност (2)
  - За да получим всички ключове на елементи, които имат дадена стойност използваме **array_keys()** с допълнителния параметър за търсене по стойност

```
$array = array('blue', 'red', 'green', 'blue', 'blue');
print_r(array_keys($array, 'blue'));

Array
(
    [0] => 0
    [1] => 3
    [2] => 4
)
```

  - array_values() връща всички стойности на даден масив

```
$array = array("size" => "XL", "color" => "gold");
print_r(array_values($array));
```

Работа с масиви

- Изтриване на елемент от масив
  - Елемент на масив премахваме с функцията **unset()**. Забележете, че елементът се премахва, но масива не се ре-индексира, така че новодобавен елемент без ключ да заеме мястото на премахнатия ключ (ако е бил цифров)

```
$a = array (1 => 1, 2 => 'two');
unset ($a[2]);
echo in_array ('two', $a); // false
$a[] = 'three';
print_r($a);
```

```
Array
(
    [1] => 1
    [3] => three
)
```

Работа с масиви

- Flip vs. Reverse
  - **array_flip()** - разменя местата на ключовете и стойностите в даден масив
    - Стойностите на входния масив трябва да са валидни ключове (цифрови или низови) - в противен случай ще бъде изведена грешка и въпросните елементи няма да разменят ключовете и стойностите си
    - Ако дадена стойност се появява повече от веднъж, за стойност ще се използва последния ключ, а всички останали ще бъдат изгубени.

```
$trans = array(
    "a" => 1,
    "b" => 1,
    "c" => 2
);
$trans = array_flip($trans);
print_r($trans);
```

```
Array
(
    [1] => b
    [2] => c
)
```

Работа с масиви

- **array_reverse()** - връща масив с елементите в обратен ред
  - Ако незадължителният втори параметър не е подаден като TRUE цифровите ключове ще бъдат ре-индексирани

```
$a = array (true, 'x' => 'a', 10 => 'b', 'c');
$b = array (true, 'x' => 'a', 10 => 'b', 'c');
```

```
print_r (array_reverse ($a));
Array
(
    [0] => c
    [1] => b
    [x] => a
    [2] => 1
)
```

```
print_r (array_reverse (
    $b, true
));

Array
(
    [11] => c
    [10] => b
    [x] => a
    [0] => 1
)
```

Работа с масиви

- Обхождане на масив (Array iteration)
  - В PHP масивите не са подредени, както е в много други езици

```
$a = array ('a' => 10, 10 => 20, 'c' => 30);
```

  - Никоя от основните циклични структури (while, for) не позволява обхождането на елементите на масив - необходима е допълнителна функционалност, която да ни позволи да обхождаме и манипулираме елементите на толкова гъвкави масиви, каквито са те в PHP
  - Всеки масив има вътрешен указател (показалец, pointer, cursor, iterator). Той е удобен начин да се запази информация за текущата итерация на даден масив, без за това да ползваме допълнителна външна променлива
  - Указателят се ползва от множество конструкции, но може да се манипулира само чрез няколко функции, като това в никаква степен не пречи да достъпваме отделните елементи в него чрез стандартните функции за работа с масиви

Работа с масиви

- **reset()** - установява указателя на първия елемент в масива и връща стойността му
- **current()** - връща стойността на елемента, към който сочи указателя
- **next()** - премества указателя към следващия елемент и връща стойността му
- **prev()** - премества указателя към предишния елемент и връща стойността му
- **end()** - премества указателя към последния елемент и връща стойността му
- **each()** - връща масив с ключа и стойността на текущия елемент и премества указателя към следващия
- **key()** - връща ключа на текущия елемент в масива

```
function displayArray($array) {
    reset($array);
    while (key($array) !== null) {
        echo key($array) .": "
             .current($array) . PHP_EOL;
        next($array);
    }
}
displayArray(array('foo' => 'bar', 'baz', 'bat' => 2));
```

```
foo: bar
0: baz
bat: 2
```

Работа с масиви

- Важно е да се отбележи, че няма абсолютно никаква зависимост между указателя на масива и ключовете в него. Като го преместваме напред и назад ни получаваме достъп до елементите според мястото им в него, а не според ключовете им.
- Обхождане чрез **foreach()**
  - Ако просто искаме да обходим всички елементи на масив, можем да ползваме функцията foreach():

```
foreach ($array as $key => $value) {
    echo "$key: $value";
}
```

Работа с масиви

- Сортиране на масив
  - По стойност
    - **sort(), rsort()**
    - **asort(), arsort()**
    - Подаденият като параметър масив всъщност се подава по референция, така че функциите модифицират оригиналния масив, а връщат TRUE / FALSE
    - Поради тази специфика, функциите приемат само променливи като параметър

Работа с масиви

- **sort(), rsort()**
  - И при двете функции съществуващите ключове се премахват и ре-индексират наново
  - rsort() сортира в обратен ред

```
$array = array(
    'a' => 'foo',
    'b' => 'bar',
    'c' => 'baz'
);
sort($array);
print_r($array);
```

```
Array
(
    [0] => bar
    [1] => baz
    [2] => foo
)
```

Работа с масиви

- **asort(), arsort()**
  - И двете функции запазват ключовете на подредените стойности
  - arsort() сортира в обратен ред

```
$array = array(
    'a' => 'foo',
    'b' => 'bar',
    'c' => 'baz'
);
asort($array);
print_r($array);
```

```
Array
(
    [b] => bar
    [c] => baz
    [a] => foo
)
```

Работа с масиви

- Сортиране по ключ
  - **ksort(), krsort()**
    - Сортират масив по ключ във възходящ и низходящ ред респективно, като запазват връзката между ключ и стойност

```
$array['ksort'] = array(
    'a' => 'foo',
    'b' => 'bar',
    'c' => 'baz',
);
ksort($array['ksort']);
```

```
[ksort] => Array
    (
        [a] => foo
        [b] => bar
        [c] => baz
    )
```

```
$array['krsort'] = array(
    'a' => 'foo',
    'b' => 'bar',
    'c' => 'baz',
);
krsort($array['krsort']);
```

```
[krsort] => Array
    (
        [c] => baz
        [b] => bar
        [a] => foo
    )
```

Работа с масиви

- Разбъркване на елементите в масив
  - **shuffle()**
    - Разбърква елементите на масива, като изцяло премахва и ре-индексира ключовете
    - Параметърът се предава по референция, т.е. функцията приема само променлива

```
$cards = array (
    1,
    'foo' => 'bar',
    2,
    3,
    4
);
shuffle ($cards);
print_r ($cards);
```

```
Array
(
    [0] => bar
    [1] => baz
    [2] => foo
)
```

Работа с масиви

- **array_keys()**
  - За да разбъркаме поредността на елементите в масив като запазим връзката ключ-стойност може да ползваме array_keys() и shuffle()

```
$cards = array (
    'a' => 10,
    'b' => 12,
    'c' => 13
);
$keys = array_keys ($cards);
shuffle($keys);
foreach ($keys as $v) {
    echo $v . "-" . $cards[$v]."\n";
}
```

```
b-12
c-13
a-10
```

Работа с масиви

- **array_rand()**
  - Връща или един случаен ключ или масив с няколко случайни ключа от масив

```
$random = array();
$cards = array (
    'a' => 10,
    'b' => 12,
    'c' => 13
);
$keys = array_rand ($cards, 2);

print_r ($keys);
print_r (array(
    $cards[$keys[0]],
    $cards[$keys[1]]
));
```

```
Array
(
    [0] => a
    [1] => c
)
Array
(
    [0] => 10
    [1] => 13
)
```

Мета-структури

- Стек (LIFO - Last In, First Out)
  - Абстрактна структура от данни, която често се използва за описване но състояние при постъпков процес. Например със стек може да се изведе списък с функции, които се изпълняват последователно – stack trace
  - **array_push()** - добавя елемент в края на масив и връща новата му дължина
  - **array_pop()** - връща последния елемент, като го премахва от масива, подаден като параметър

Мета-структури

- Опашка (FIFO - First In, First Out)
  - **array_shift()** - връща стойността на първия елемент от масива, като в същото време го премахва от него. Дължината на масива се намалява с единица, като всички цифрови индекси се ре-индексират. Низовите ключове не се променят
  - **array_unshift()** - добавя елемент в началото на масива. Връща новата му дължина.

```
$queue = array('qux', 'bar', 'baz');
$first_element = array_shift($queue);
print_r($queue);

array_unshift($queue, 'foo');
print_r($queue);
```

```
Array
(
    [0] => bar
    [1] => baz
)
Array
(
    [0] => foo
    [1] => bar
    [2] => baz
)
```

Мета-структури

- Множество (Set)
  - Трите основни операции в теорията за множествата (обединение, сечение, разлика) могат да бъдат реализирани с помощта на различни PHP функции, като множествата се представят от масиви
  - Обединение (Union) - всички елементи от две множества, без дубликатите. За да го реализираме ползваме две функции
    - **array_merge()** - описано по-горе
    - **array_unique()** - връща масив, без елементите с повтарящи се стойности в оригиналния масив

```
$input = array(
    'a' => 'green',
    'red',
    'b' => 'green',
    'blue',
    'red'
);
$result = array_unique($input);
print_r($result);
```

```
Array
(
    [a] => green
    [0] => red
    [1] => blue
)
```

Мета-структури

- Сечение (Intersection) - общите елементи в 2 или повече масива
  - **array_intersect()** - връща масив с всички стойности от първия параметър, които са налични и в останалите подадени параметри. Ключовете ще бъдат запазени
  - **array_intersect_key()** - връща масив с всички стойности от първия параметър, които имат общи ключове във всички останали подадени параметри
  - **array_intersect_assoc()** - връща масив с всички стойности от първия параметър, които имат еднакви двойки ключ-стойност и в останалите подадени параметри
- Разлика (Difference) - стойностите от един масив, които не съществуват в друг масив
  - **array_diff()** - връща масив с всички елементи от първия параметър, които не съществуват в никой от останалите параметри
  - **array_diff_key()** - връща масив с всички елементи от първия параметър, които нямат общи ключове с елементи от никой от останалите параметри
  - **array_diff_assoc()** - връща масив с всички елементи от първия параметър, които нямат общи двойки ключ-стойност в никой от останалите параметри

Въпроси?

Благодаря ви за вниманието!