Intermediate PHP

Bradley Holt & Matthew Weier O'Phinney

Arrays

Photo by AJ Cann

- Associate values to keys
- Keys can be integers (enumerative arrays) or strings (associative arrays)
- Values can be primitive types, objects, or other arrays (multidimensional arrays)

```php
<?php
$post = array(
  'id'          => 1234,
  'title'       => 'Intermediate PHP',
  'updated'     => new DateTime(
    '2010-12-16T18:00:00-05:00'
  ),
  'draft'       => false,
  'priority'    => 0.8,
  'categories'  => array('PHP', 'BTV')
);
```

Reading, Writing, and Appending Arrays

```php
<?php
$post = array(
   'id'                => 1234,
   'title'             => 'Intermediate PHP',
   // …
);
echo $post['title']; // Intermediate PHP
```

Reading from an associative array

```php
<?php
$post = array(
  'id'                => 1234,
  'title'             => 'Intermediate PHP',
  // …
);
$post['title'] = 'PHP 201';
```

Writing to an associative array

```php
<?php
$post = array(
  'id'          => 1234,
  'title'       => 'Intermediate PHP',
  'updated'     => new DateTime(
    '2010-12-16T18:00:00-05:00'
  ),
  'draft'       => false,
  'priority'    => 0.8,
  'categories'  => array('PHP', 'BTV')
);
$post['summary'] = 'Arrays, functions, and
objects';
```

Appending to an associative array

```php
<?php
$post = array(
    // …
    'categories'   => array('PHP', 'BTV')
);
$post['categories'][] = 'Programming';
print_r($post['categories']);
/*
Array
(
        [0] => PHP
        [1] => BTV
        [2] => Programming
)
 */
```

Appending to an enumerative array

```php
<?php
$post = array(
   // …
   'categories'  => array('PHP', 'BTV')
);
$post['categories'][1] = 'Burlington';
print_r($post['categories']);
/*
Array
(
    [0] => PHP
    [1] => Burlington
)
 */
```

Writing to an enumerative array

Iterating Over Arrays

```php
<?php
$post = array(
  // …
  'categories'  => array('PHP', 'BTV')
);
foreach ($post['categories'] as $v) {
  echo $v . PHP_EOL;
}
/*
PHP
BTV
 */
```

Iterating values in an enumerative array

```php
<?php
$post = array(
  // …
  'categories'  => array('PHP', 'BTV')
);
foreach ($post['categories'] as $k => $v) {
  echo $k . ': ' . $v . PHP_EOL;
}
/*
0: PHP
1: BTV
 */
```

Iterating key value pairs in an enumerative array

```php
<?php
$post = array(
  'id'                => 1234,
  'title'             => 'Intermediate PHP',
  // …
);
foreach ($post as $k => $v) {
  echo $k . ': ' . $v . PHP_EOL;
}
/*
id: 1234
title: Intermediate PHP

…
 */
```

Iterating key value pairs in an associative array

Implode and Explode

```php
<?php
$post = array(
    // …
    'categories'   => array('PHP', 'BTV')
);
echo implode(', ', $post['categories']);
// PHP, BTV
```

Implode returns a string, not an array.

```php
<?php
$post = array(
   // …
   'categories'   =>
     explode(', ', 'PHP, BTV')
);
print_r($post['categories']);
/*
Array
(
     [0] => PHP
     [1] => BTV
)
 */
```

Explode returns an array.

Array Key Exists, In Array, and Array Keys

```php
<?php
$post = array(
  'id'              => 1234,
  // …
  'categories'  => array('PHP', 'BTV')
);
if (array_key_exists('categories', $post))
{
  echo implode(', ', $post['categories']);
} else {
  echo 'Uncategorized';
}
```

```php
<?php
$post = array(
    // …
    'categories'   => array('PHP', 'BTV')
);
if (in_array('PHP', $post['categories'])) {
     echo 'PHP: Hypertext Preprocessor';
}
```

```php
<?php
$posts = array(
  1233  => array(/* … */),
  1234  => array(/* … */),
);
print_r(array_keys($posts));
/*
Array
(
    [0] => 1233
    [1] => 1234
)
 */
```

Sorting Arrays

```php
<?php
$post = array(
   // …
   'categories'   => array('PHP', 'BTV')
);
sort($post['categories']);
print_r($post['categories']);
/*
Array
(
      [0] => BTV
      [1] => PHP
)
 */
```

Sorting Function Attributes

- Some sort by value, others by key
- Some maintain key association, others do not
- Some sort low to high, others high to low
- Some are case sensitive, some are not
- See: http://www.php.net/manual/en/array.sorting.php

Sorting Functions

- array_multisort()
- asort()
- arsort()
- krsort()
- ksort()
- natcasesort()
- natsort()
- rsort()
- shuffle()
- sort()
- uasort()
- uksort()
- usort()

Stacks and Queues

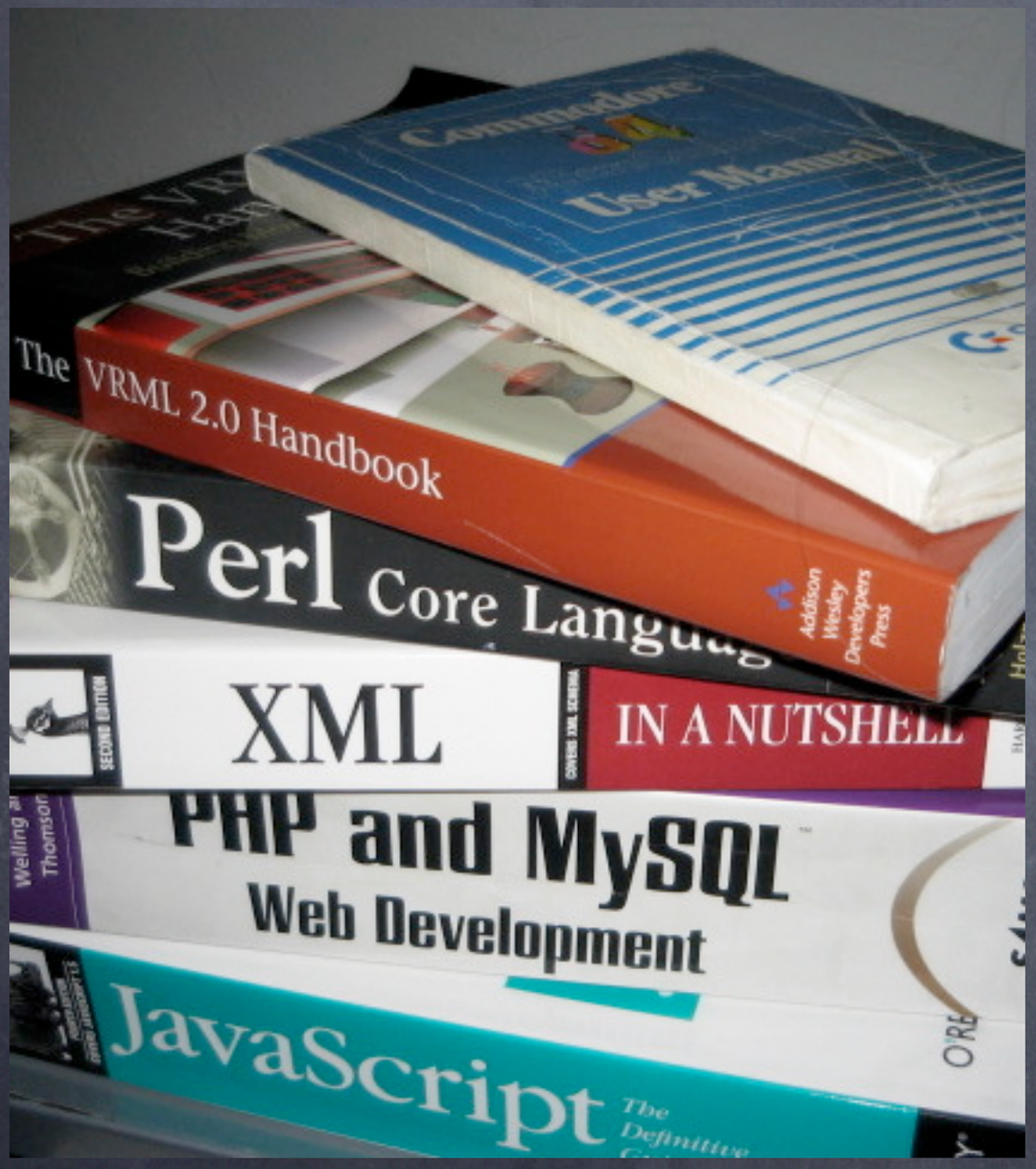

Photo by Phil Cooper

Photo by Thomas W

The SPL in PHP 5.3 has new data structures that can make stacks and queues more performant and memory efficient.

Array Push and Array Pop (stack)

```php
<?php
$post = array(
   // …
   'categories'  => array('PHP')
);
array_push($post['categories'], 'BTV');
echo array_pop($post['categories']);
// BTV
print_r($post['categories']);
/*
Array
(
    [0] => PHP
)
 */
```

Example of an array used as a stack (FILO)

Array Unshift and Array Pop (queue)

```php
<?php
$post = array(
   // …
   'categories'   => array('BTV')
);
array_unshift($post['categories'], 'PHP');
print_r($post['categories']);
/*
Array
(
      [0] => PHP
      [1] => BTV
)
 */
echo array_pop($post['categories']);
// BTV
```

Example of an array used as a queue (FIFO)

Functions

Internal Functions

```php
<?php
print_r(get_defined_functions());
/*
Array
(
    [internal] => Array
        (
            [0] => zend_version
            [1] => func_num_args
            // …
        )
    [user] => Array
        (
        )
)
 */
```

```php
<?php
$functions = get_defined_functions();
echo count($functions['internal']);
// 1857
```

The number of internal functions will depend on which extensions you have installed.

```php
<?php
$name = 'Marcus Börger';
if (function_exists('mb_strtolower')) {
   echo mb_strtolower($name, 'UTF-8');
   // marcus börger
} else {
   echo strtolower($name);
   // marcus b?rger
}
```

The multibyte string (mbstring) extension allows you to safely manipulate multibyte strings (e.g. UTF-8).

User-Defined Functions

Rules

- Any valid PHP code is allowed inside functions, including other functions and class definitions
- Function names are case-insensitive
- Once defined, a function cannot be undefined

```php
<?php
function nextId($posts)
{
  return max(array_keys($posts)) + 1;
}
$posts = array(
  1233  => array(/* … */),
  1234  => array(/* … */),
);
echo nextId($posts); // 1235
```

This function finds the highest array key and adds one.

Type Hinting

```php
<?php
function nextId(array $posts)
{
  return max(array_keys($posts)) + 1;
}
$posts = array(
  1233  => array(/* … */),
  1234  => array(/* … */),
);
echo nextId($posts); // 1235
echo nextId(1234);
// Argument 1 passed to nextId() must be an
array, integer given
```

You can type hint on array and class names, but not primitive data types like strings and integers.

Multiple Arguments

```php
<?php
function isPublished(DateTime $published,
$draft)
{
  if ($draft) { return false; }
  $now = new DateTime();
  return $now >= $published;
}
$published = new DateTime
('2010-12-16T18:00:00-05:00');
$draft = false;
var_dump(isPublished($published, $draft));
// bool(true)
```

This function determines if a post is published based on its published date and whether or not it’s a draft.

Default Arguments

```php
<?php
function isPublished(DateTime $published,
$draft, $now = false)
{
  if ($draft) { return false; }
  $now = $now ? $now : new DateTime();
  return $now >= $published;
}
$published = new DateTime
('2010-12-16T18:00:00-05:00');
$draft = false;
$now = new DateTime
('2010-12-16T17:59:59-05:00');
var_dump(isPublished($published, $draft,
$now));
// bool(false)
```

This is essentially the same as the previous function but allows you to specific when “now” is.

Function Overloading
(not really)

```php
<?php
function nextId($arg1)
{
  switch (true) {
    case is_array($arg1):
      return max(array_keys($arg1)) + 1;
    case is_int($arg1):
      return $arg1 + 1;
  }
}
$posts = array(
  1233  => array(/* … */),
  1234  => array(/* … */),
);
echo nextId($posts); // 1235
echo nextId(1234); // 1235
```

```
throw new InvalidArgumentException();
```

Variable Number of Arguments

```php
<?php
function mostRecent()
{
    $max = false;
    foreach (func_get_args() as $arg) {
        $max = $arg > $max ? $arg : $max;
    }
    return $max;
}
$mostRecent = mostRecent(
        new DateTime('2010-12-14T18:00:00'),
        new DateTime('2010-12-16T18:00:00'),
        new DateTime('2010-12-15T18:00:00')
);
// 2010-12-16T18:00:00
```

Objects

Basics

- Encapsulate metadata and behavior
- metadata => properties (variables, constants)
- behavior => methods (functions)

We already know about variables and functions now!

Anonymous Objects

"Anonymous" as they are not of any specific class; they're basically equivalent to "bare objects" or "object literals" in JavaScript.

Cast an Associative Array to (object)

```php
<?php
$a = array(
   'foo' => 'bar',
);
$o = (object) $a;
echo $o->foo; // 'bar'
```

stdClass

```php
$o = new stdClass;
$o->foo = 'bar';
echo $o->foo; // 'bar'
```

Declaring a Class

```php
<?php
class Foo
{
}
```

Declaring Properties

```php
<?php
class Foo
{
  protected $bar = 'bar';
}
```

Definitions may not use computations or runtime values (other than constants); they must be concrete.

Extending Classes

```php
<?php
class FooBar extends Foo
{
}
```

Visibility

- **public**: accessible from instances or within methods of any visibility, and within extending classes

- **protected**: accessible only within instance methods of any visibility, and within extending classes

- **private**: accessible only within instance methods from the declaring class

Modifiers

- **final**: cannot be overridden in extending classes
- **abstract**: must be overridden in extending classes

Abstract Classes

```php
<?php
abstract class AbstractFoo
{
  abstract public function sayBar();
}

class Foo extends AbstractFoo
{
  public $bar;

  public function sayBar()
  {
    echo $this->bar;
  }
}
```

Interfaces

- "Blueprints" for classes
- Typically indicate **behaviors** found in implementing classes
- You can implement many interfaces, but only **extend** once
  - Allows you to compose multiple behaviors into a single class

Interfaces **can** extend other interfaces. Methods in interfaces **must** be public.

```php
<?php
interface Resource {
  public function getResource();
}
interface Dispatcher {
  public function dispatch();
}

abstract class DispatchableResource
implements Resource,Dispatcher
{
  // …
}
```

```php
<?php
interface Resource { /* … */ }
interface Dispatcher { /* … */ }

abstract class DispatchableResource
implements Resource,Dispatcher
{
  protected $resource;
  public function getResource() {
    return $this->resource;
  }
  public function dispatch() {
    return 'Dispatched ' . $this-
>getResource();
  }
}
```

```php
<?php
interface Resource { /* … */ }
interface Dispatcher { /* … */ }

abstract class DispatchableResource
implements Resource,Dispatcher { /* … */ }

class TrafficCop extends
DispatchableResource
{
  protected $resource = 'traffic cop';
}
$cop = new TrafficCop();
echo $cop->dispatch();
// 'Dispatched traffic cop'
```

Useful Tidbits

Type hinting: you can indicate an interface, abstract class, or class name "hint" with parameters:

```
public function doSomething(Dispatcher
$dispatcher) {/* … */}
```

Test for types:

```
if ($object instanceof Dispatcher) { }
```

Type hints look at the entire inheritance chain, including interfaces!

Statics

- Static properties and methods may be accessed without instantiation
- Requires a different token to access:
  - `ClassName::$varName` for variables
  - `Classname::methodName()` for methods
  - or use `self` or `parent` or `static` within a class, instead of the class name

Late Static Binding

- `static` is used for "Late Static Binding" (LSB)

```php
<?php
class BasicPerson
{
  public static $sex = 'unisex';
  public static function getSex()
  {
    return static::$sex;
  }
}
class MalePerson extends BasicPerson
{
  public static $sex = 'male'; // LSB
}
echo BasicPerson::getSex(); // 'unisex'
BasicPerson::$sex = 'female';
echo BasicPerson::getSex(); // 'female'
echo MalePerson::getSex();  // 'male'
```

Magic Methods

Tie into different states of an object

Construct, Destruct, and Invoke

- __construct – Constructor (used when "new Class" is called)
- __destruct – Destructor (called when object is destroyed)
- __invoke – Call an object instance like a function (echo $object();)

Sleep and Wakeup

- __sleep – Define what properties should be serialized, and how
- __wakeup – Determine how to initialize state from serialized instance

Call and Call Static

- Overload method access
- __call
- __callStatic

Get and Set

- Overload property access
  - __get
  - __set

Cardinal Rule

- Objects should model discrete concepts, and not just be a container for functions.

If you want the latter, use namespaces!

Questions?

Thank You

Bradley Holt & Matthew Weier O'Phinney

License