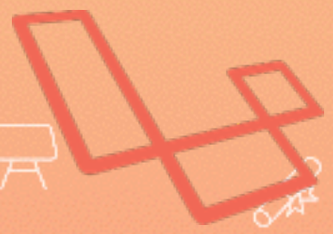
適用 5.1 版



PHP 語法基礎與物件導向

范聖佑 Shengyou Fan

新北市樹林國小 (2015/07/06)

```php
/**
 * Display the specified resource.
 *
 * @param  int  $id
 * @return Response
 */
public function show($id)
{
    $post = Post::with('comments')->where('id', $id)->first();

    return View::make('post.show')->with('post', $post);
}
```

單元主題

- 複習 PHP 基礎語法
- 複習物件導向觀念，及如何使用 PHP 實作物件導向程式設計
- PHP 5.3+ 新語法及功能介紹

本單元將以後續工作坊 (PHP 5.3+ 及 Laravel 5.1) 會使用到的語法為主，更多語法詳細內容請參考 PHP 官方文件及 Laravel 官方文件。

PHP 基礎語法

PHP

- 於 1995 年由 Rasmus Lerdorf 創始，並以開放源始碼釋出，至今已 20 年
- Personal Home Page、Hypertext Preprocessor
- 手稿型程式語言 (scripting language)
- 以直譯器 (interpreter) 動態編譯
- 動態、弱型別 (dynamic, weak typing)
- 當前最新穩定版為 5.6.10
- PHP 7 預定於今年 10 月釋出

兩種執行方式

- 一般常見 PHP 的執行方式，都是將 *.php 檔案放置在網頁伺服器的文件根目錄底下，再透過瀏覽器瀏覽執行後看結果 (在本工作坊就是使用 uwamp 這個整合包)
- 由於 PHP 也有實作指令列模式，因此也可以在終端機底下，以 `php {filename}.php` 的方式執行某個 php 檔案，並在終端機底下看到執行後的輸出結果
  (在本工作坊就是使用 cmder 這個整合包)
- 換句話說，其實可以拿 PHP 來寫指令列工具

P.S 要特別注意的是，要使用指令列模式，除了 PHP 本身要掛載正確的設定外，也要將 PHP 載入至終端機本身運行的 PATH 內

宣告 PHP 區塊

- PHP 天生的特性就是一個樣板語言，可以混合 HTML 直接輸出成網頁
- 純文字檔案，以 .php 結尾
- 撰寫 PHP 語法時，必須將語法以 <?php ?> 宣告
- 直譯式語言，每一句以分號 ; 結尾

```
<?php
    // php 語法寫在區塊內
    $var = 'some strings';
?>
<html>
<head>
<!-- 可以混合與 HTML 輸出 -->
```

輸出語法

- PHP 常見輸出變數內容的方式有兩種
  - 直接使用 echo 語法
  - 使用 <?= ?> 輸出變數

```
// 2-declarations/index.php
<?php
    $pageTitle = 'Hello, world';
    $pageContent 'Long long time ago...';
?>
<html>
<head><!-- 略 --></head>
<body>
    <h1><?php echo $pageTitle; ?></h1>
    <p><?=$pageContent?></p>
</body>
</html>
```

註解

- PHP 的註解型式有兩種：
  - 單行/行內式
  - 多行/區塊式

```
// 3-comments/index.php
<?php
    // 單行式註解
    $var = 'hello, world'; // 行內式註解
    # 也可以用 # 號

    /*
     * 多行區塊式註解
     */

    /* 想這樣使用也可以 */
?>
```

宣告變數

- 所有的 PHP 變數宣告皆以 $ 字號做為開頭
- 變數名稱以 _ 和 a-z 及 A-Z 開頭 (不可以數字開頭)
- 變數以 = 賦予其值
- PHP 是弱型別，賦予值時會自動判別資料型別

```
// 4-variables/index.php
// 宣告變數
$var = 'Bob';  // 合法，小寫開頭
$Var = 'Joe';  // 合法，大寫開頭
$_4site = 'hello, world'; // 合法，底線開頭

$4site = 'hello, world';  // 不合法，數字開頭
```

資料型別

- 整數 (integer)
- 浮點數 (float)
- 布林值 (boolean)
- 字串 (string)

```php
// 5-types/index.php
$int = 1234;
$int = -1234;

$float = 1234.56;

$boolean = true;

$string = 'hello, world';
```

資料型別

- 陣列 (array)
- 物件 (object)
- 資源 (resource)

```php
// 5-types/index.php
$array53 = array(1, 2, 3, 4, 5); // 5.3 以前的陣列寫法
$array54 = [ // 5.4 以後的陣列簡寫
    ['id' => 1, 'name' => 'Tom'],
    ['id' => 2, 'name' => 'Simon'],
];

$object = new MyObj;
$object->method();

// $handle 即為一種 resource
$handle = fopen("/* file path */", "r");
```

弱型別與強制轉型

- PHP 的變數是弱型別，在宣告時不需指定型別，直譯器會自動判定後給予型別。在操作過程中若有需要也會自動轉型
- 若需要強制變更變數的型別，可以用語法強制轉型

```php
// 6-weaktypes-and-casts/index.php
$var = "0"; // $var 是字串 0
$var = $var + 2; // $var 變成整數 2
$var = $var + 1.3 // $var 變成浮點數 3.3

$cast = (int) $var; // 強制轉型成整數
$cast = (bool) $var; // 強制轉型成布林值
$cast = (float) $var; // 強制轉型成浮點數
$cast = (string) $var; // 強制轉型成字串
$cast = (array) $var; // 強制轉型成陣列
$cast = (object) $var; // 強制轉型成物件
```

內建函式 (functions)

- PHP 本身內建了大量的函式可直接呼叫使用
- 可以在官方文件上查詢這些函式的使用方式

```
// 7-buildin-functions/index.php

// http://php.net/manual/en/function.strlen.php
$string = "hello, world";
echo strlen($string);

// http://php.net/manual/en/function.str-replace.php
$search = 'world';
$replace = 'dolly';
echo str_replace($search, $replace, $string);
```

自定函式

- PHP 可以自行定義函式，包括函式名稱、參數、參數預設值、函式回傳
- 呼叫時，直接使用函式名稱加上指定參數並以括號包起來即可

```php
// 8-userdefined-functions/index.php
// 定義函式
function greeting($name = 'Simon', $words = ['Hi']) {
    $index = rand(0, count($words) - 1);
    $sentence = $words[$index].', '.$name;

    return $sentence;
}

// 呼叫函式
echo greeting('Tom', ['Hola', 'Hello', 'Good Morning']);
```

算數操作子

- 加、減、乘、除

```
// 9-operators/index.php
$x = 10;
$y = 2;

$result = $x + $y;
echo $result;

$result = $x - $y;
echo $result;

$result = $x * $y;
echo $result;

$result = $x / $y;
echo $result;
```

賦值操作/遞增遞減

- 算數操作與賦值同時進行

```php
// 9-operators/index.php
$x = 20;

$x += 2;
$x -= 6;
$x *= 5;
$x /= 2;
$x .= 'txt';
```

- 遞增、遞減

```php
// 9-operators/index.php
$x = 20;

$x++;
$x--;
```

邏輯判斷

- if
- elseif
- else

```php
// conditions-1.php
$x = 21;

if ($x < 20) {
    echo '小於 20';
} elseif ($x > 20) {
     echo '大於 20’;
} else {
     echo '等於 20’;
}
```

比較運算子

- `>`、`<`、`>=`、`<=`
- `==`、`!=`
- `!`、`and` (`&&`)、`or` (`||`)

```php
// 11-comparisions/index.php

$x = 20;
var_dump($x > 20);
var_dump($x >= 20);
var_dump($x < 20);
var_dump($x <= 20);

var_dump($x == 20);
var_dump($x != 20);
```

```php
$int = 22;
$bool = false;

var_dump(!$bool);

var_dump($int > 0 && $int < 50);
var_dump($int > 0 || $int < 20);
```

迴圈 (loop)

- for

```php
// 12-loops/index.php
for ($i = 0; $i < 10; $i++) {
    echo $i;
}
```

- foreach

```php
// 12-loops/index.php
foreach (range(1, 10) as $index => $value) {
    echo 'index: '.$index.', value: '.$value;
}
```

引入

- require
- include

```
// 13-require-and-include/index.php
require 'require-sample.php';

include 'include-sample.php';
```

PHP 物件導向

物件導向程式設計

- 雖然使用 PHP 內建的函式就可以寫出動態網站，但若使用物件導向程式設計 (OOP)，可以讓大量網站的程式碼更加有組織、讓更多人可以參與並讓程式碼重複利用
- 物件導向程式設計的價值在於封裝 (encapsulation)。它讓我們把一系列有關係的變數、函式等集合成一個方便操作的物件，除了方便使用外，也可以讓程式碼易於維護和擴充

定義物件

```php
// 14-defined-object/Animal.php
class Animal // 物件名稱
{
    public $name;  // 物件屬性
    protected $axis;

    public function __construct($name) // 建構式
    {
        $this->name = $name;
        $this->axis = ['x' => 0, 'y' => 0];
    }

    public function move($x, $y) // 物件方法
    {
        $this->axis['x'] += $x;
        $this->axis['y'] += $y;

        return $this->axis;
    }
}
```

實體化並使用它

- 物件定義只是一個藍圖，要使用它之前，需要先實體化並儲存在變數內，接著就可以使用物件本身的所有功能
- 要實體化一個物件要用 new 關鍵字
- 要存取物件本身的屬性或方法時，要用 -> 語法

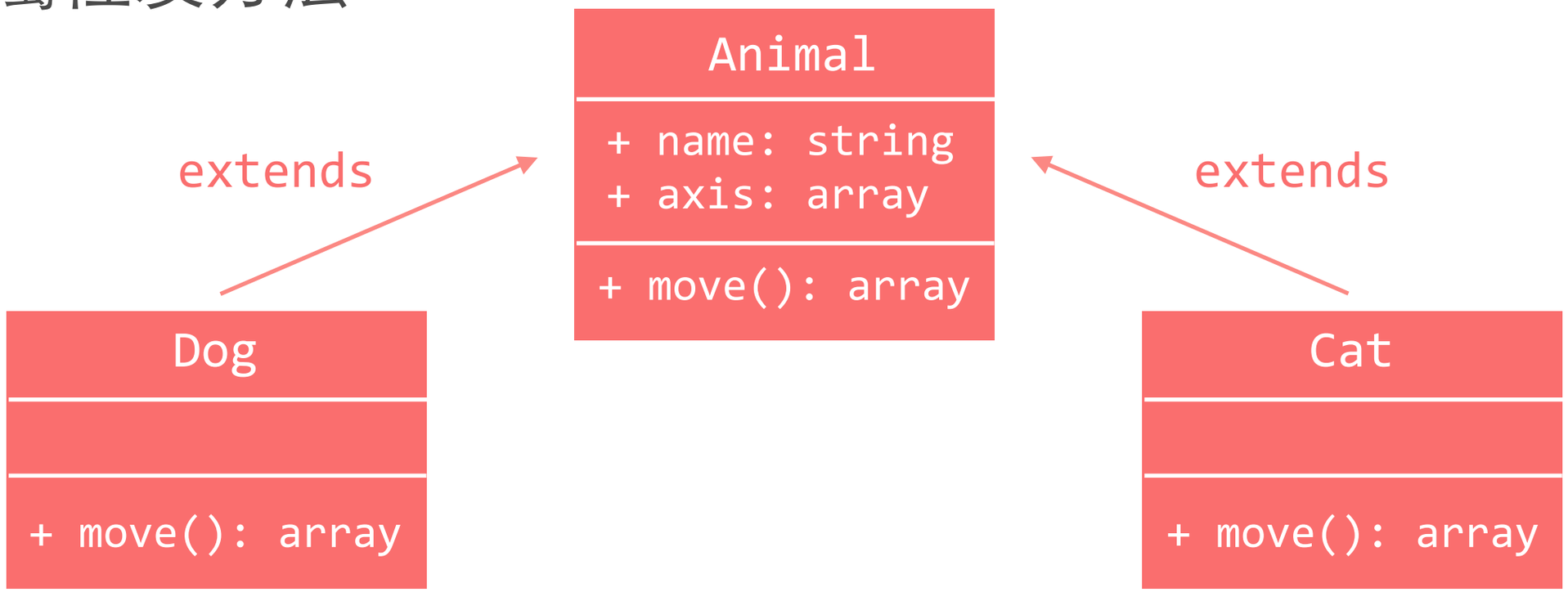
```
// 15-instantiate/index.php
$myPet = new Animal('Lucky');

$name = $myPet->name;
$axis = $myPet->move(10, 20);

echo 'My pet is '.$name;
echo 'It axis is ('.$axis['x'].', '.$axis['y'].')';
```

OOP 術語

- Object (物件) - 類比於現實生活中的一個東西
- Class (類別) - 創造物件的藍圖定義
- Instantiate (實體化) - 依照 Class 實做出物件的過程
- Property (屬性) - 物件裡的變數
- Method (方法) - 物件裡的函式
- Constructor (建構式) - 在實體化物件時會執行的一段函式，可用於物件初始化的設定。使用 PHP 的魔術方法 (magic method) `__construct()` 定義

繼承

- 在實作上為了讓類別之間具有親屬關係，並讓物件可依需求擴充其功能，所以在物件導向程式設計裡很重要的語法就是繼承。
- 簡而言之就是子類別可以繼承 (擴充 `extends`) 一個父類別，透過繼承，子類別就自動具有所有父類別的屬性及方法



繼承範例

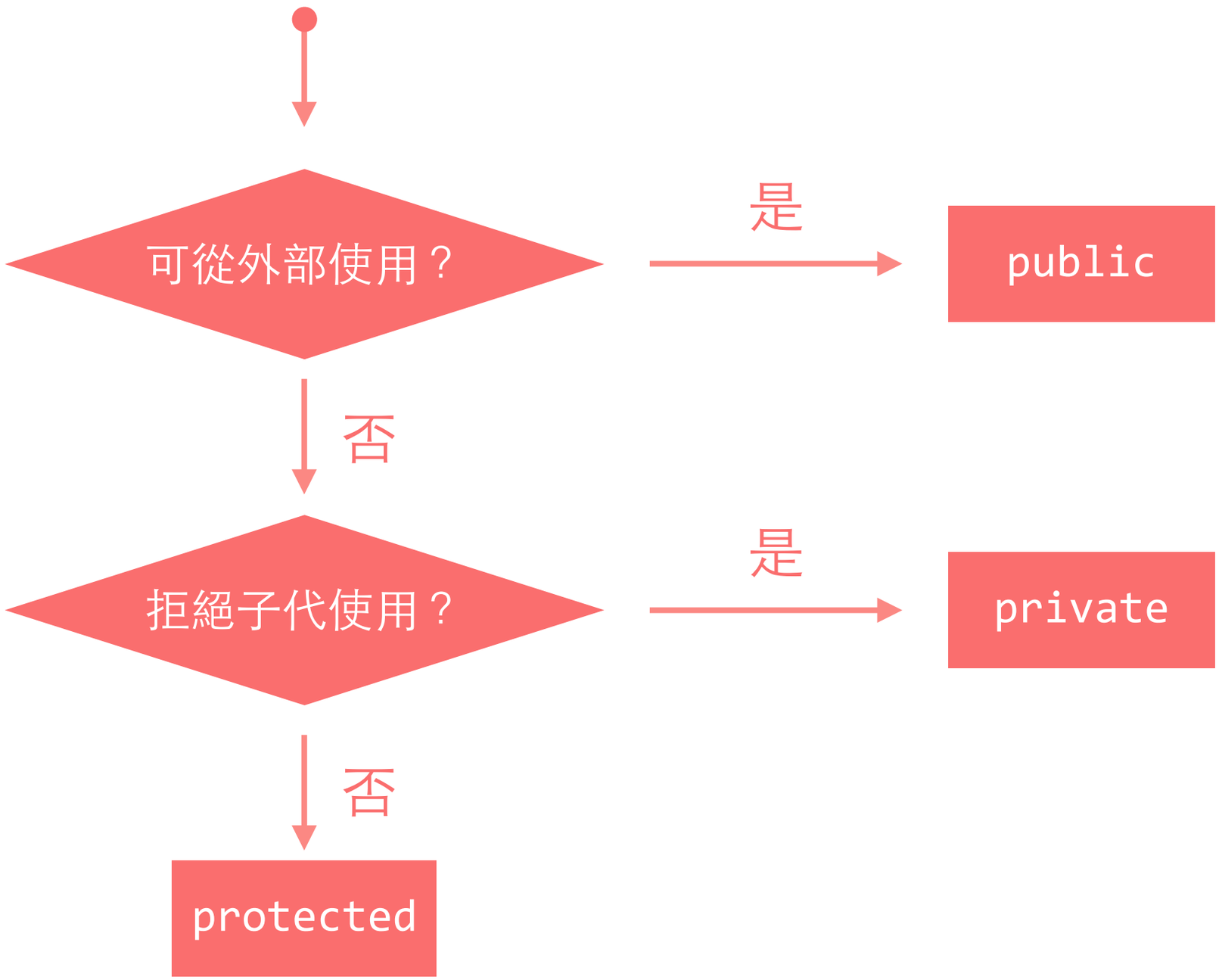
```php
// 16-inheritance/Dog.php
class Dog extends Animal
{
    // overwrite
    public function move($x, $y)
    {
        $x = $x + 20;
        $y = $y + 20;
        $this->axis['x'] += $x;
        $this->axis['y'] += $y;

        return $this->axis;
    }
}
```

```php
// 16-inheritance/Cat.php
class Cat extends Animal
{
    // overwrite
    public function move($x, $y)
    {
        $x = $x - 10;
        $y = $y - 10;
        $this->axis['x'] += $x;
        $this->axis['y'] += $y;

        return $this->axis;
    }
}
```

visibility (可見度)

- 物件內的屬性和方法都可以設定其 visibility，由此管制屬性和方法是否能從外部或子代來變更？
- visibility 的設定共有三種：
  - `public`：該屬性和方法可從外部使用
  - `private`：該屬性和方法只有本身可存取，外部或子代都無法使用
  - `protected`：與 `private` 類似，但其子代可以覆寫

選擇正確的 visibility



建立 getter 及 setter

- 透過 public 讓外界直接改變物件的屬性太過自由，比較好的作法是將屬性設定為 protected 後，建立 getter 及 setter 來存取它們
- 這樣做除了可以管控存取外，也可以在取值和設值的過程中依需求多做一些事 (Log、變更格式、驗證…等)

getter 及 setter 範例

```php
// 17-getter-and-setter/Animal.php
class Animal
{
    protected $name;
    /* 略 */

    // getter
    public function getName()
    {
        return $this->name;
    }

    // setter
    public function setName($name)
    {
        $this->name = $name;
        return true;
    }
}
```

```php
// 17-getter-and-setter/
// index.php
$animal = new Animal;
$animal->setName('Lucky');

echo $animal->getName();
```

靜態屬性和方法

- 在物件還沒有實體化之前，所有的屬性和方法都是不可使用的
- 但若在定義時就設定該屬性或方法為 `static` 的話，則可以在未實體化前就使用
- 使用時要用 `::` 來取用

```php
// 18-static/Dog.php
class Dog
{
    // 略
    public static function bark()
    {
        echo 'wang wang';
    }
}
```

```php
// 18-static/index.php

// 使用靜態方法
Dog::bark();
```

介面

- 雖然物件之間可以透過繼承取得父類別能力的實作，但在某些狀況下，能力的實作其實不符合物件繼承的概念
- 介面是描述物件能力的一種方式，其只指定了方法的名稱和參數，但不實作內容
- 在類別宣告的時候，需要使用 `implements` 關鍵字來宣告實作介面，而一旦類別實作了該介面，就一定要實作該介面定義的方法內容
- 透過介面，讓物件在實作上有了合約 (contract) 的締結，讓物件導向程式設計在實作上更為嚴謹

介面範例

```php
// 19-interface/Swimmable.php
interface Swimmable
{
    public function swim();
}
```

```php
// 19-interface/Dog.php
class Dog extends Animal implements Swimmable
{
    public function swim()
    {
        echo 'swimming...';
    }
}

// 19-interface/index.php
$dog = new Dog;
$dog->swim();
```

型別提示 (Type Hint)

- 雖然 PHP 本身是弱型別程式語言，但 function 接受參數時，我們可以透過 type hinting 的功能，要求傳入的參數一定是某種型別或物件，這在物件導向程式設計裡特別有用
- 可以做型別提示的只有 array、object、callable

```
// 20-typehinting/Human.php
public function greeting(array $words = null) {
    $speaks = ($words)? $words : $this->words;
    $index = rand(0, count($speaks) - 1);
    $sentence = $this->name.' say '.$speaks[$index].' to you!';
    return $sentence;
}

// 20-typehinting/index.php
echo $tom->greeting(['Hi', 'Hello', 'Good Morning']);
```

Polymorphism (多型)

- PHP 的物件導向支援 Polymorphism 的設計，也就是說子代也可以被識別為具有父代或介面特徵
- 因此在物件方法做 type hinting 時，可以指定介面或父類別，即便傳入的是子類別的實作或擴充，一樣都會通過
- 在這物件導向程式設計裡非常有用，讓物件間僅需依賴介面實作，就可以確保取得物件的能力，而不需要知道物件本身的實作

method chain (方法鏈結)

- 在實作物件方法時，若我們在方法內回傳物件本身，就可以實作出 Fluent Interface (可以在許多 PHP 元件上看到這種用法)

```php
// 21-method-chain/Dog.php
class Dog
{
    // 略
    public function bark()
    {
        echo 'wang wang';
        return $this;
    }
    public function swim()
    {
        echo 'swimming...';
        return $this;
    }
}
```

```php
// 21-method-chain/
// index.php
$dog = new Dog;
$dog->bark()->swim();
```

abstract 類別

- 當物件被宣告為 abstract 時，該物件本身就不能被實體化，而一定是被拿來做繼承
- 可以用來做 Base Class 使用，我們會在 Laravel 的 BaseController 看到這種設計

```php
// Illuminate\Routing\Controller.php
abstract class Controller
{
    protected $middleware = [];
    /* 略 */

    public function middleware($middleware, array $options = [])
    {
        $this->middleware[$middleware] = $options;
    }
    /* 略 */
}
```

PHP 5.3+ 新語法

namespace (命名空間)

- 在撰寫物件時，最常碰到的一個狀況就是：萬一我的物件名稱跟別人重複時，該怎麼辦？
- namespace 就是為了解決這個問題而存在，在定義類別時，可將類別放在命名空間內
- 在使用 namespace 時，要特別注意目前所在的 namespace 為何？
  - 可以透過 全名 實體化到該物件
  - 可以先用 use 關鍵字宣告

定義 namespace

```php
// 23-namespace/app/Helpers/StringHelper.php
<?php

namespace App\Helpers;

class StringHelper
{
    public function sanitize($string)
    {
        $result = trim($string);
        $result = strip_tags($result);

        return htmlentities($result, ENT_QUOTES, 'UTF-8');
    }
}
```

使用 namespace 的物件

```php
// 23-namespace/index.php
<?php

require 'app/Helpers/StringHelper.php';

$string = '   hello <h1>world</h1>   ';

$helper = new \App\Helpers\StringHelper;
echo $helper->sanitize($string);
```

```php
// 23-namespace/index.php
<?php

require 'app/Helpers/StringHelper.php';

use App\Helpers\StringHelper;

$string = '   hello <h1>world</h1>   ';
$helper = new StringHelper;
echo $helper->sanitize($string);
```

namespace alias

- 若覺得 Class 名稱太長、或是遇到兩個一樣名稱的 Class 時，可以使用 alias (別名) 來做區隔

```
// 23-namespace/index.php
<?php

require 'app/Helpers/StringHelper.php';

use App\Helpers\StringHelper as S;

$string = '   hello <h1>world</h1>   ';
$helper = new S;
echo $helper->sanitize($string);
```

Trait

- Interface 解決了不同類別之間共享能力的問題，但 Interface 無法定義實作內容。若兩個類別的實作內容相同時，仍有部份的程式碼是重覆的
- Trait 可以把它想像成請 PHP Interpreter 幫我們做程式碼複製貼上的動作，讓不同類別間的實作內容可以共用
- 但要特別注意的是，PHP Interpreter 僅幫我們做複製貼上的動作，並不嚴格的管理 Class 裡的實作是否符合 Trait 裡假定，若中間有誤是會讓程式出錯的
- 在 Laravel 裡，SoftDelete 就是用 Trait 來實作

Trait 範例

```php
// 24-trait/Swimmable.php
trait Swimmable
{
    public function swim()
    {
        echo 'swimming...';
    }
}

// 24-trait/Dog.php
class Dog extends Animal
{
    use Swimmable;
}

// 24-trait/index.php
$dog = new Dog;
$dog->swim();
```

callbacks/callables

- 在看 PHP 官方文件時，不知道有沒有注意過有些函式的參數是 $callback？

```
array array_map ( callable $callback , array $array1 [, array $... ] )
```

- 透過 anonymous functions，讓我們可以直接將函式寫成 callback，不必在另外定義函式

```
// 25-callback/index.php
// 大部份的範例
function cube($n) {
    return ($n * $n * $n);
}

$result = array_map("cube", [
    1, 2, 3, 4, 5
]);
```

```
// 其實可以直接寫成這樣
$r = array_map(function($n) {
    return ($n * $n * $n);
}, [1, 2, 3, 4, 5]);
```

傳遞參數至 callback 內

- 使用 anonymous functions 時，函式內部是運作在其自己的空間內，換句話說，函式本身無法存取外部變數的。但可以透過 use 關鍵字將變數傳遞至函式內部使用

```php
// 25-callback/index.php
$base = 2;

$result = array_map(function($n) use ($base) {
    return $n * $base;
}, [1, 2, 3]);

// $result = [2, 4, 6];
```

PSR 規範

- PHP 界充滿了各式各樣的組件與框架。但在早期，所謂的現代 PHP 生態系還沒生成前，舊版框架在開發上的作法常常讓開發者無法容易地分享程式碼
- 2009 年在 php | tek 研討會上，數個 PHP 框架的開發者意識到這樣的問題，因此組成了 PHP-FIG 這樣的組織，目的在於讓框架之間能夠更有效的溝通與交流
- PSR 就是 PHP-FIG 提出的建議規範，依照編號目前有 PSR-1、PSR-2、PSR-3、PSR-4，每一個建議都是為了解決一些常見的特定問題

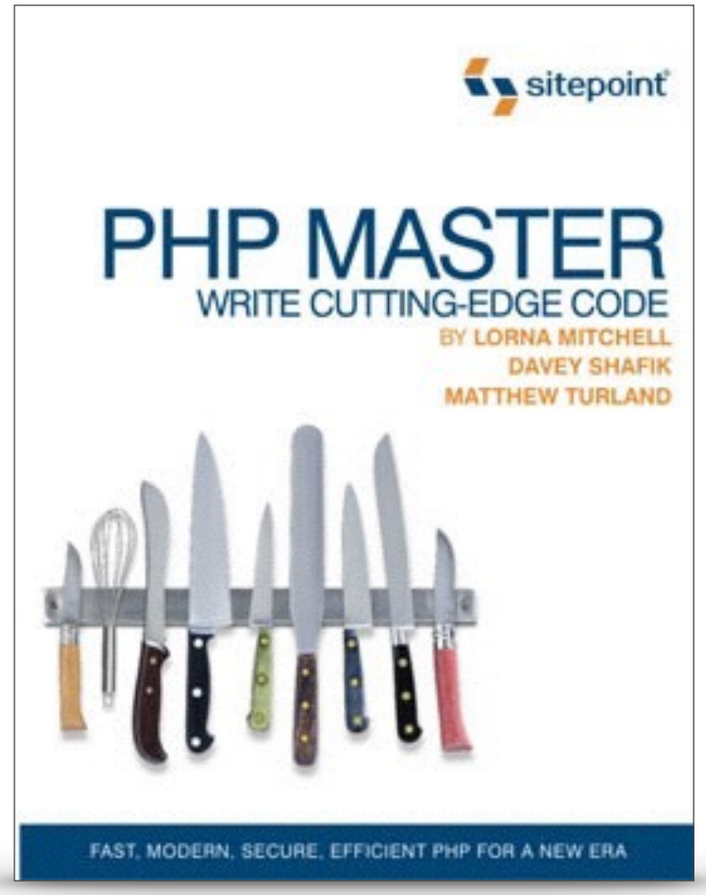
PSR-4 與 autoload

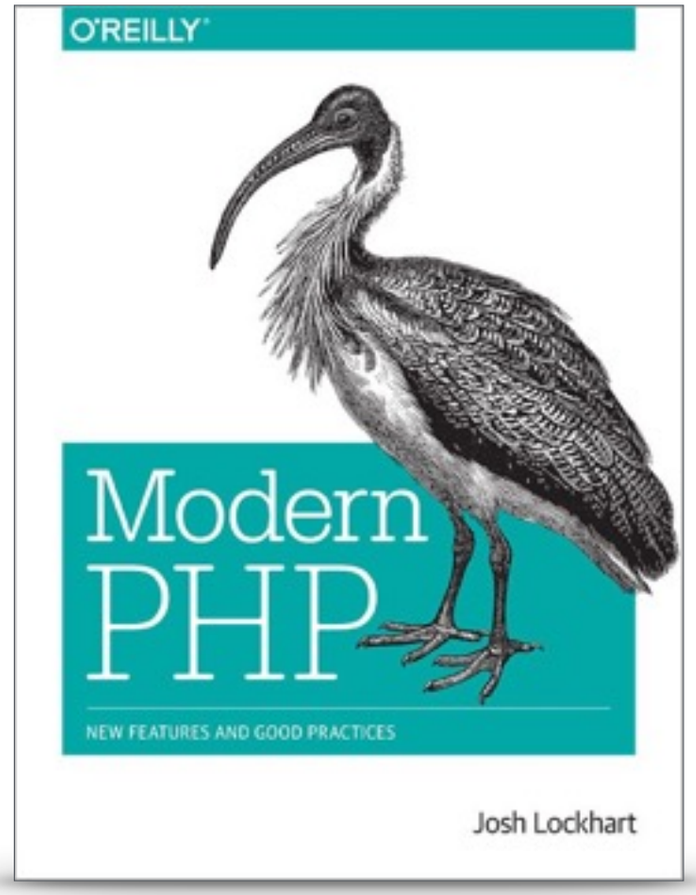
- 以往在使用類別前，都要透過 `require`、`require_once`、`include`、`include_once` 等方式將檔案引用後才可使用，造成檔案的開頭會有一堆引入的語法
- 為了解決這個問題，PHP 提供 `__autoload()` 及 `spl_autoload_register()` 可以讓開發者自行定義載入邏輯
- 但若每個開發者的載入邏輯都不同時，對於元件的使用者而言也是一樣的困擾。因此 PSR-4 規範裡定義了一套 namespace 與資料夾放置的統一作法，只要元件開發者依循其規範，所有的使用者就可以透過 Composer 處理自動載入的問題

單元總結

- 在這個單元裡我們學到了些什麼？
  - PHP 基礎觀念
  - 物件導向語法
  - PHP 5.3+ 新語法及功能

推薦閱讀





http://www.sitepoint.com/store/php-master-write-cutting-edge-code/

http://shop.oreilly.com/product/0636920033868.do

Q & A

歡迎提問討論