PHP 物件導向

mike 陳南邦
chennanbang@gmail.com
line = telegram = 0933 596 597

可見度 Visibility

- 屬性，方法，都可設定可見度
  - public：可從外部存取，繼承後可存取
  - protected：不可從外部存取，繼承後可存取
  - private：不可從外部存取，繼承後不可存取
- 測試剛才的例子
  - 吃跟咀嚼的不同，外部可見 / 自己可見

內部函式

靜態 static

- 物件實體化之前，即可使用，已在記憶體中

```php
class Cat
{
    public $name;
    protected $position;
    public static $constant;

    public function __construct($nam
    {
        $this->name = $name;
        $this->position = ['x'=>0, '
        self::$constant = "last assi
    }
```

- static 屬性
  - 所有物件共用同樣的資料
- 使用方式
  - 用2個冒 —> ::。其中，在 static 方法中，不能用 $this。

```php
public static function test()
{
    self::$constant = "test static function";
    $this->name = self::$constant;
```

練習

- static 屬性，方法
  - 試試有什麼結果
- 子類別建構子呼叫父類別建構子

覆載 Overriding

- 繼承父類別的方法，並重新改寫當中的內容
- 執行子物件的該方法時，會以重新改寫的內容執行，而不是父類別的

```php
public function move($x, $y)
{
    $this->position['x'] += $x;
    $this->position['y'] += $y;

    return $this->position;
}
```

練習

- 繼承並改寫移動的內容
- 測試看看～

```php
public function move($x, $y)
{
    $this->position['x'] += $x;
    $this->position['y'] += $y;

    return $this->position;
}
```

```php
public function move($x, $y)
{
    $this->position['x'] += $x*2;
    $this->position['y'] += $y*2;

    return $this->position;
}
```

介面 Interface

- 不適合繼承情況，比如功能的 "實作"。
- 用來描述功能名稱，參數，不包括實作
- 用 implements 來宣告要實作該介面，且一定要實作。
- 好像是一個合約 (Contract) 讓實作方面更嚴謹

練習

- 將動物移動的能力都改由介面來做
- 測試看看～

```
interface Action
{
    public function move();
}
```

型別提示 Type Hint

- 可以要求傳入的參數的型別
- php 5.6.21 可用的型別：object，array，callable(callback)

```php
class TypeA
{
    private $typeName = "TypeA";
    public function getTypeName()
    {
        return $this->typeName;
    }
}
```

```php
function testTypeHint(TypeA $type)
{
    $type->getTypeName();
}

testTypeHint(new TypeA());
testTypeHint(new TypeB());
```

練習

- object，array，callable(callback) 的 TypeHint

```php
function testArray(array $array) {
    echo $array[0]."<br>";
}

testArray(array('直接輸入 array'));
testArray(array('k'=>'v'));
testArray(array('0'=>'solved?'));
//testArray('string');
```

```php
function tryThis()
{
    echo "function show<br>";
}

function testCallable(callable $c
    $callableFunction();
}

testCallable(
    function()
    {
        echo "閉包<br>";
    }
);
testCallable(array(new TypeC, 'ge
testCallable('tryThis');
```

多型 Polymorphism

- 想要使用某個類別的某個功能時，得到的是繼承該類別的子類別的那個功能所提供的結果。結果是因不同的子類別而不同。
- 也就是針對某個父類別所描述的某個功能，讓不同的子類別的的那個功能的實作來實現父類別所描述的那個功能。
  - 父類別中，動物的叫聲
  - 貓，狗，鯨魚的叫聲

應用

- 錄音機錄動物的叫聲
  - 錄音機不需要知道是什麼動物
  - 結果，不同的動物，錄到不同的叫聲

練習

- PHP 是由介面來實現多型

```php
interface Sounds
{
    public function sounds();
}
```

- 建立聲音介面，描述聲音功能
- 子類別實作聲音功能
- 建立錄音類別來錄音
- 測試看看～

```php
class Whale implements Sounds
{
    public function sounds()
    {
        return "鯨魚叫<br>";
    }
}
```

```php
class Recording
{
    public function addSound(Sounds $sounds)
    {
        $recordingSounc = $sounds->sounds();
        echo "錄到的聲音: ".$recordingSounc;
    }
}
```

多型的好處

- 只要物件有繼承到該類別即可
- 擴充時，不需太多修改
- 例如，遊戲增加新角色時，只要求能實現某個功能，但不需知道細節。

Q & A

mike 陳南邦
chennanbang@gmail.com
line = telegram = 0933 596 597