Web-разработка на PHP-технологиях

Курс лекций и семинаров для студентов,
желающих научиться основам Web-разработки на PHP

Осень-Зима 2014

Лекция №7

PHP — Основы ООП

Автор: Дмитрий Левин, Senior PHP Developer, Sibers

Sibers®

Основы ООП

Стратегию ООП лучше всего описать как смещение приоритетов в процессе программирования от функциональности приложения к структурам данных. Это позволяет программисту моделировать в создаваемых приложениях реальные объекты и ситуации.

Технология ООП обладает тремя главными преимуществами:

- она проста для понимания: ООП позволяет мыслить категориями повседневных объектов;
- повышенно надежна и проста для сопровождения — правильное проектирование обеспечивает простоту расширения и модификации объектно-ориентированных программ. Модульная структура позволяет вносить независимые изменения в разные части программы, сводя к минимуму риск ошибок программирования;
- ускоряет цикл разработки — модульность и здесь играет важную роль, поскольку различные компоненты объектно-ориентированных программ можно легко использовать в других программах, что уменьшает избыточность кода и снижает риск внесения ошибок при копировании.

Основы ООП

Объектно-ориентированное программирование основано на:

- Инкапсуляции
- Полиморфизме
- Наследовании

Основы ООП

Инкапсуляция — свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента (что у него внутри?), а взаимодействовать с ним посредством предоставляемого интерфейса (публичных методов и членов), а также объединить и защитить жизненно важные для компонента данные.

При этом пользователю предоставляется только спецификация (интерфейс) объекта. Пользователь может взаимодействовать с объектом только через этот интерфейс. Реализуется с помощью ключевого слова: public.

Пользователь не может использовать закрытые данные и методы. Реализуется с помощью ключевых слов: private, protected.

Сокрытие реализации целесообразно применять в следующих случаях: предельная локализация изменений при необходимости таких изменений, прогнозируемость изменений (какие изменения в коде надо сделать для заданного изменения функциональности) и прогнозируемость последствий изменений.

Основы ООП

Наследование — это механизм позволяющий описать новый класс на основе уже существующего (родительского),при этом свойства и функциональность родительского класса заимствуются новым классом.

Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс).Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса. Простое наследование Класс, от которого произошло наследование, называется базовым или родительским (англ. base class).

Классы, которые произошли от базового, называются потомками, наследниками или производными классами (derived class)В некоторых языках используются абстрактные классы. Абстрактный класс — это класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. То есть от абстрактного класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного. Например, абстрактным классом может быть базовый класс «сотрудник вуза», от которого наследуются классы «аспирант», «профессор» и т. д. Так как производные классы имеют общие поля и функции (например, поле «год рождения»), то эти члены класса могут быть описаны в базовом классе. В программе создаются объекты на основе классов «аспирант», «профессор», но нет смысла создавать объект на основе класса «сотрудник вуза».

Основы ООП

Множественное наследование

При множественном наследовании у класса может быть более одного предка. В этом случае класс наследует методы всех предков. Достоинства такого подхода в большей гибкости.

Множественное наследование — потенциальный источник ошибок, которые могут возникнуть из-за наличия одинаковых имен методов в предках.

Основы ООП

Полиморфизм — возможность объектов с одинаковой спецификацией иметь различную реализацию.

Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию — например, реализация класса может быть изменена в процессе наследования [1]. Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций».

Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода. Общие свойства объектов объединяются в систему, которую могут называть по-разному — интерфейс, класс. Общность имеет внешнее и внутреннее выражение: внешняя общность проявляется как одинаковый набор методов с одинаковыми именами и сигнатурами (именем методов и типами аргументов и их количеством); внутренняя общность — одинаковая функциональность методов. Её можно описать интуитивно или выразить в виде строгих законов, правил, которым должны подчиняться методы. Возможность приписывать разную функциональность одному методу (функции, операции) называется перегрузкой метода (перегрузкой функций, перегрузкой операций).

Основы ООП

Пример наследования:

```php
<?php
class Parent {
  function parent_funct() { echo "<h1>Это родительская функция</h1>"; }
  function test () { echo "<h1>Это родительский класс</h1>"; }
}

class Child extends Parent {
  function child_funct() { echo "<h2>Это дочерняя функция</h2>"; }
  function test () { echo "<h2>Это дочерний класс</h2>"; }
}

$object = new Parent;
$object = new Child;

$object->parent_funct(); // Выводит 'Это родительская функция'
$object->child_funct(); // Выводит 'Это дочерняя функция'
$object->test(); // Выводит 'Это дочерний класс'
?>
```

Основы ООП

Полиморфизм классов в PHP

```php
<?php
class A {
// Выводит, функция какого класса была вызвана
function Test() { echo "Test from A\n"; }
// Тестовая функция – просто переадресует на Test()
function Call() { Test(); }
}
class B extends A {
// Функция Test() для класса B
function Test() { echo "Test from B\n"; }
}
$a=new A();
$b=new B();
?>
```

```
$a->Call(); // выводит "Test from A"
$b->Test(); // выводит "Test from B"
$b->Call(); // Внимание! Выводит "Test from B"!
```

Основы ООП

Полиморфизм классов в PHP

```php
<?php
class Base {
 function funct() {
 echo "<h2>Функция базового класса</h2>";
 }
 function base_funct() {
 $this->funct();
 }
}

class Derivative extends Base {
 function funct() {
 echo "<h3>Функция производного класса</h3>";
 }
}
```

```php
$b = new Base();
$d = new Derivative();

$b->base_funct();
$d->funct();
$d->base_funct();
// Скрипт выводит:

// Функция базового класса
// Функция производного класса
// Функция производного класса
?>
```

Основы ООП

public/private/protected — модификаторы доступа для методов и свойств

```php
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Works
echo $obj->protected; // Fatal Error
echo $obj->private; // Fatal Error
$obj->printHello(); // Shows Public, Protected and Private
```

Основы ООП

PHP 5 позволяет объявлять **методы-конструкторы**. Классы, в которых объявлен метод-констуктор, будут вызывать этот метод при каждом создании нового объекта, так что это может оказаться полезным, чтобы, например, инициализировать какое-либо состояние объекта перед его использованием.

Конструктор, ранее совпадавший с названием класса, теперь необходимо объявлять как **__construct()**, что позволит легче перемещать классы в иерархиях.

Конструкторы в классах-родителях не вызываются автоматически. Чтобы вызвать конструктор, объявленный в родительском классе, следует обратиться к методу **parent::__construct()**.

Основы ООП

```
class BaseClass {
     function __construct() {
         print "Конструктор класса BaseClass\n";
     }
}

class SubClass extends BaseClass {
     function __construct() {
         parent::__construct();
         print "Конструктор класса SubClass\n";
     }
}

$obj = new BaseClass();
$obj = new SubClass();
```

Основы ООП

PHP 5 предоставляет концепцию деструкторов, сходную с теми, что применяются в других ОО языках, таких, как Java: когда освобождается последняя ссылка на объект, перед высвобождением памяти, занимаемой этим объектом, вызывается метод __destruct(), не принимающий параметров.

```php
class MyDestructableClass {
    function __construct() {
        print "Конструктор\n";
        $this->name = "MyDestructableClass";
    }

    function __destruct() {
        print "Уничтожается " . $this->name . "\n";
    }
}

$obj = new MyDestructableClass();
```

Siberis®

Функции для работы с классами и объектами

Функции для работы с классами и объектами

get_class_methods()

Функция get_class_methods() возвращает массив имен методов класса с заданным именем

```php
<?php
...
class Airplane extends Vehicle {
  var $wingspan;
  function setWingSpan($wingspan) {
    $this->wingspan = $wingspan;
  }

  function getWingSpan() {
    return $this->wingspan;
  }
}

$cls_methods = get_class_methods(Airplane);
// Массив $cls_methods содержит имена всех методов,
// объявленных в классах "Airplane" и "Vehicle"
?>
```

Функции для работы с классами и объектами

get_class_vars()

Функция get_class_vars() возвращает массив имен атрибутов класса с заданным именем

```php
<?php

class Vehicle {
var $model;
var $current_speed;
}

 class Airplane extends Vehicle {
 var $wingspan;
 }

$a_class = "Airplane";
$attribs = get_class_vars($a_class);
// $attribs = array ("wingspan", "model", "current_speed")
?>
```

Функции для работы с классами и объектами

get_object_vars()

Функция get_object_vars() возвращает ассоциативный массив с информацией обо всех атрибутах объекта с заданным именем

```
class Vehicle {
var $wheels;
}

 class Land extends Vehicle {
 var $engine;
 }

  class car extends Land {
  var $doors;
   function car($doors, $eng, $wheels) {
   $this->doors = $doors;
   $this->engine = $eng;
   $this->wheels = $wheels;
   }

   function get_wheels() {
   return $this->wheels;
   }
  }
```

```
$toyota = new car(2,400,4);
$vars = get_object_vars($toyota);
while (list($key, $value) = each($vars)) :
print "$key ==> $value <br>";
endwhile;
// Выходные данные:
// wheels ==> 4
// engine ==> 400
// doors ==> 2
```

Sibers®

Функции для работы с классами и объектами

method_exists()

Функция method_exists() проверяет, поддерживается ли объектом метод с заданным именем. Если метод поддерживается, функция возвращает TRUE, в противном случае возвращается FALSE.

```
class Vehicle {
// ...
}

 class Land extends Vehicle {
 var $fourWheel;

  function setFourWheelDrive() {
  $this->fourWeel = 1;
  }

 }
// Создать объект с именем $car
$car = new Land;
```

```
// Если метод "fourWheelDrive" поддерживается классом "Land"
// или "Vehicle", вызов method_exists возвращает TRUE;
// в противном случае возвращается FALSE.
// В данном примере method_exists() возвращает TRUE.
if (method_exists($car, "setfourWheelDrive")) :
print "This car is equipped with 4-wheel drive";
else :
print "This car is not equipped with 4-wheel drive";
endif;
```

Sibers®

Функции для работы с классами и объектами

```php
<?php

class Vehicle {
}

 class Land extends Vehicle {
 }

// Создаем объект с именем $car:
$car = new Land;
// Переменной $class_a присваивается строка "Land":
$class_a = get_class($car);
echo $class_a;
?>
```

Функции для работы с классами и объектами

get_parent_class()

Функция get_parent_class() возвращает имя родительского класса (если он есть) для объекта с заданным именем

```php
<?php

class Vehicle {
//...
}

 class Land extends Vehicle {
 //...
 }

// Создаем объект с именем $car:
$car = new Land;
// Переменной $parent присваивается строка "Vehicle":
$parent = get_parent_class($car);
?>
```

Функции для работы с классами и объектами

is_subclass_of()

Функция is_subclass_of() проверяет, был ли объект создан на базе класса, имеющего родительский класс с заданным именем. Функция возвращает TRUE, если проверка дает положительный результат, и FALSE в противном случае.

```php
<?php

class Vehicle {
//...
}

 class Land extends Vehicle {
 //...
 }
$auto = new Land;
// Переменной $is_subclass присваивается TRUE
$is_subclass = is_subclass_of($auto, "Vehicle");
?>
```

Функции для работы с классами и объектами

get_declared_classes()

Функция get_declared_classes() возвращает массив с именами всех определенных классов

```php
<?php

 class Vehicle {
 //...
 }

  class Land extends Vehicle {
  //...
  }

$declared_classes = get_declared_classes();
// $declared_classes = array("Vehicle", "Land")
?>
```

Заключение:

В седьмой лекции мы рассмотрели преимущества и основы технологии ООП, обозначили семь функций для работы с классами и объектами

В следующей лекции:

Паттерны ООП

Полезные ссылки:

http://www.php.su/learnphp/phpoo/

http://www.firststeps.ru/theory/oop/r.php?1

http://habrahabr.ru/post/87119/