第 19 章 面向对象的特性

学习要点：

1.OOP 的封装

2.OOP 的继承

3.OOP 的多态

主讲教师：李炎恢

合作网站：http://www.ibeifeng.com

讲师博客：http://hi.baidu.com/李炎恢

面向对象的三个主要特性是封装、继承和多态。

一. OOP的封装

隐藏对象的字段和实现细节，仅对外公开接口,控制在程序中字段的读和修改的访问级别；将抽象得到的数据和行为（或功能）相结合，形成 一个有机的整体，也就是将数据与操作数据的源代码进行有机的结合，形成“类”，其中数据和函数都是类的成员。

字段的作用域

1.public 公共的 (类外可以访问)

2.private 私有的 (类内可以访问)

3.protected 受保护的 (类内和子类可以访问，类外不可访问)

创建使用了私有的字段，这样外部就无法访问了

```
class Computer {
    //类的字段(成员)
    private $_name = '联想120';
    private $_model = 'LX';
}
```

通过一个公共方法作为入口，访问私有字段，而必须使用$this关键字。

```
class Computer {
    //类的字段(成员)
    private $_name = '联想120';
    private $_model = 'LX';
    //通过公共方法来访问私有字段
    function run () {
        echo $this->_name;
    }
}
$computer->run();
```

属性操作(私有字段的赋值与取值)

可以设计两个公共方法，一个方法为setName()，用于赋值；一个方法为getName()，用于取值。

```
class Computer {
    //类的字段(成员)
    private $_name;
    private $_model;
    //赋值
    function setName($_name) {
        $this->_name = $_name;
    }
    //取值
    function getName() {
        return $this->_name;
    }
}
$computer = new Computer();
$computer->setName('IBM');
echo $computer->getName();
```

如果有十个字段那么就必须要二十个方法才能够赋值和取值，那么有没有更简便的方法呢？PHP内置两个方法(拦截器)专门用于取值与赋值：__set()，__get()。

```
class Computer {
    //类的字段(成员)
    private $_name;
    private $_model;
    //所有字段的赋值都在这里进行
    function __set($_key,$_value) {
        $this->$_key = $_value;
    }
    //所有字段的取值都在这里进行
    function __get($_key) {
        return $this->$_key;
    }
}
$computer = new Computer();
$computer->_model =  'LX';
echo $computer->_model;
```

方法私有：有些使用类里面的方法并不需要对外公开，只是里面运作的一部分，这个时候可以将方法也封装起来。

```
class Computer {
    //类的字段(成员)
    private $_name;
    private $_model;
```

```
        //私有方法
        private function getEcho() {
            echo '我是私有化的方法';
        }
        //公共方法一般是对外的入口
        public    function run() {
            $this->getEcho();
        }
    }
    $computer = new Computer();
    $computer->run();
```

建议：方法前面如果没有修饰符，那么就是外部可访问的公共方法，但为了让程序更加的清晰，建议在前面加上public。

常量（constant）

在类中可以定义常量，用来表示不会改变的值。对于从该类实例化的任何对象来说，常量值在这些对象的整个生命周期中都保持不变。

```
    class Computer {
        const PI = 3.1415926;
    }
    echo Computer::PI;
```

静态类成员

有时候，可能需要创建供所有类实例共享的字段和方法，这些字段和方法与所有的类实例有关，但不能由任何特定对象调用。

```
    class Computer {
        public static $_count = 0;
    }

    echo Computer::$_count;
```

一般来说，必须将字段做成私有化。所以可能需要这么做：

```
    class Computer {
        private static $_count = 0;
        public static function setRun() {
            self::$_count++;
        }
        public static function getRun() {
            return self::$_count;
        }
    }
```

```
Computer::setRun();
echo Computer::getRun();
```

Instanceof关键字

PHP5有一个instanceof关键字，使用这个关键字可以确定一个对象是类的实例、类的子类，还是实现了某个特定接口，并进行相应的操作。

```
class Computer {
    //
}
$computer = new Computer();
echo ($computer instanceof Computer);
```

二．OOP继承

继承是从一个基类得到一个或多个类的机制。

继承自另一个类的类被称为该类的子类。这种关系通常用父类和孩子来比喻。子类将继承父类的特性。这些特性由属性和方法组成。子类可以增加父类之外的新功能，因此子类也被称为父类的“扩展”。

在PHP中，类继承通过extends关键字实现。继承自其他类的类成为子类或派生类，子类所继承的类成为父类或基类。(PHP只支持单继承，PHP不支持方法重载)。

```
class Computer {
    private $_name = '联想120';
    private function __get($_key) {
        return $this->$_key;
    }
    public function run() {
        echo '我是父类';
    }
}
class NoteBookComputer extends Computer {

}
$notebookcomputer = new NoteBookComputer();
$notebookcomputer->run();
echo $notebookcomputer->_name;
```

字段和方法的重写（覆盖）

有些时候，并不是特别需要父类的字段和方法，那么可以通过子类的重写来修改父类的字段和方法。

```
class Computer {
    public $_name = '联想120';
    protected function run() {
        echo '我是父类';
    }
}

class NoteBookComputer extends Computer {
    public $_name = 'IBM';
    public function run() {
        echo '我是子类';
    }
}
```

子类调用父类的字段或方法

为了安全，我们一般将父类的方法封装了起来，这样，外部就无法调用，只能被继承它的子类所看到。这个时候，就需要通过子类操作来调用父类了。

```
class Computer {
    protected $_name = '联想120';
    protected function run() {
        echo '我是父类';
    }
}

class NoteBookComputer extends Computer {
    public function getName() {
        echo $this->_name;
    }
    public function getRun() {
        echo $this->run();
    }
}
```

通过重写调用父类的方法

有的时候，我们需要通过重写的方法里能够调用父类的方法内容，这个时候就必须使用语法：父类名::方法() 或者parent::方法()即可调用。

```
class Computer {
    protected function run() {
        echo '我是父类';
    }
}
```

```
class NoteBookComputer extends Computer {
    public function run() {
        echo Computer::run(); //
    }
}
```

final关键字可以防止类被继承，有些时候只想做个独立的类，不想被其他类继承使用，那么就必须使用这个关键字。建议只要是单独的类，都加上这个关键字。

```
final class Computer {
    //无法继承的类
    final public function run() {} //无法被继承的方法
}
class NoteBookComputer extends Computer {
    //会报错
}
```

抽象类和方法（abstract）

抽象方法很特殊，只在父类中声明，但在子类中实现。只有声明为abstract的类可以声明抽象方法。

规则：

1.抽象类不能被实例化，只能被继承。

2.抽象方法必须被子类方法重写。

```
abstract class Computer {
    abstract function run();
}
final class NotebookComputer extends Computer {
    public function run() {
        echo '我实现了';
    }
}
```

接口（interface）

接口定义了实现某种服务的一般规范，声明了所需的函数和常量，但不指定如何实现。之所以不给出实现的细节，是因为不同的实体可能需要用不同的方式来实现公共的方法定义。关键是要建立必须实现的一组一般原则，只要满足了这些原则才能说实现了这个接口。

规则：

1.类全部为抽象方法（不需要声明abstract）

2.接口抽象方法必须是public

3.成员（字段）必须是常量

```
interface Computer {
    const NAME = '联想120';
    public function run();
}
final class NotebookComputer implements Computer {
    public function run() {
        echo '实现了接口的方法';
    }
}
$notebookcomputer = new NoteBookComputer();
$notebookcomputer->run();
echo Computer::NAME;
```

子类可以实现多个接口

```
interface Computer {
    const NAME = '联想120';
    public function run();
}
interface Notebook {
    public function book();
}
final class NotebookComputer implements Computer,Notebook {
    public function run() {
        echo '实现了接口的方法';
    }
    public function book() {
        echo '实现了接口的方法';
    }
}
```

三．多态

多态是指 OOP 能够根据使用类的上下文来重新定义或改变类的性质或行为，或者说接口的多种不同的实现方式即为多态。把不同的子类对象都当作父类来看，可以屏蔽不同子类对象之间的差异，写出通用的代码，做出通用的编程，以适应需求的不断变化。

```
interface Computer {
    public function version();
    public function work();
}

class NotebookComputer implements Computer {
    public function version() {
```

```
            echo '联想120';
        }
        public function work() {
            echo '笔记本正在随时携带运行！';
        }
}

class desktopComputer implements Computer {
        public function version() {
            echo 'IBM';
        }
        public function work() {
            echo '台式电脑正在工作站运行！';
        }
}

class Person {
        public function run($type) {
            $type->version();
            $type->work();
        }
}

$person = new Person();
$desktopcomputer = new desktopComputer();
$notebookcomputer = new NoteBookComputer();
$person->run($notebookcomputer);
```

感谢收看本次教程！

本课程是由北风网**(ibeifeng.com)**

瓢城 **Web** 俱乐部**(yc60.com)**联合提供：

本次主讲老师：李炎恢

我的博客：**hi.baidu.com/李炎恢/**

我的邮件：**yc60.com@gmail.com**