# iranphp articles

| | |
|---|---|
| عنوان مقاله : | PHPدر مسیر شئ گرایي(قسمت اول) |
| نگارنده : | شیلان کلهر |
| آدرس پست الکترونیك : | ..... |
| تاریخ نگارش : | ..... |

## PHPدر مسیر شئ گرایی(قسمت اول) :

اغلب از مفهوم شئ گرایی در زبان مورد علاقه ما چشم پوشی می شود یا به غلط تفسیر می شود ولی اگر به طور صحیح بکار گرفته می شود خیلی هم قوی به نظر می رسد. آیندهPHP درباره شئ گرایی بسیار روشن است این از خصوصیات جدیدی که درPHP5 گنجانده شده کاملاً مشخص است. با ابزار مناسب تنها چیزی که لازم داریم دانش است.

در این مقاله سعی شده که مفهوم واقعی شئ را بررسی کنیم و اینکه چه جوری اونها را شناسایی کنیم همچنین سعی شده ما را با سه رکن اصلی OOP یعنی کپسوله کردن، ارث بری و پلی مورفیسم بیشتر آشنا کند .

برنامه نویسی شئ گرا همانطور که از اسمش هم پیداست برنامه نویسی با اشیاء است. ولی خوب معنی دقیق شئ چیه؟ بذارید اول بگم شئ چی نیست! شئ فقط یه کلاس نیست که یه مشت تابع داخلش ریخته باشن. شاید به نظر بدیهی بیاد ولی واقعیت اینه که وقتی آدم مقوله OOPرو در یه زبان رویه ای مثل PHP کشف می کنه هیجان زده میشه که زودتر استفادش کنه بدون اینکه از تئوری اولیه اون خبر داشته باشه !

### حالا شئ چیه؟

شئ موجودیتی است که خصوصیات (properties) و رفتارهایی را (behavior) در خودش کپسوله می کند که مخصوص همان موجودیت است . شاید به نظر عجیب بیاد ولی اشیاء دور و ور ما هستند. بعضی خودشون از اشیاء دیگری تشکیل شدن. روزنامه دستتون، پنجره اتاق خود اتاق در دیوار . . . ولی درباره OOP بعنوان یک مفهوم چیزای زیادی هست که باید بدونیم. در طول این سالها (کدوم سالها؟!) متدلوژیهای مختلفی برای نزدیک شدن به شئ گرایی توسعه یافتند، حتی می شه به سه مرحله تقسیمش کرد: تحلیل شئ گرا (OOA) ، طراحی شئ گرا (OOD) و برنامه نویسی شئ گرا (OOP) که برای تبحر در هر کدومشون وقت زیادی لازم است

مثلا مدلهای طراحی هنوز بدجوری رویه ای هستند. می دونید فقط syntax نیست که مهم هست در واقع قسمت سخت ماجرا شئ گرایانه فکر کردن و تمرین کردن و در نهایت بکارگیری هست. نگذارید این واقعیات شما رو دلسرد کنه چون یادگیری مفاهیم قسمت اعظم موضوعه (که لابد تو این مقاله قراره یاد بگیریم شئ گرایی مزایای زیادی داره از جمله: استفاده دوباره، توسعه پذیری و نگهداشت پذیری که مهمترین اونها هستن .

استفاده دوباره: اشیاء می توانند رو پای خوشون بایستند یعنی مجردند؛ و نشاندهنده یک چیز هستند. به این معنی که می توانند به گونه های مختلف ترکیب شوند، که این همون خاصیت استفاده دوباره را ایجاد میکند. استفاده دوباره از اشیاء کلی توی وقت ما صرفه جویی میکنه چون مجبور نیستیم همه چیز و از اول بسازیم .

توسعه پذیری: بجای نوشتن یک شئ از اول ما می تونیم یک شئ را گسترش بدیم. یه شئ می تواند از یک شئ دیگر مشتق شود و فقط کارایی هایی را که لازماست به شئ جدید بیفزاییم .

نگهداشت پذیری: طبیعتاً اشیاء چون خوانایی بالایی دارند، خیلی راحتتر تحلیل میشوند و خیلی بهتر میتوان از برنامه های موجود توسعه اشون داد و اینکه طبیعت "pluggable" دارند کمتری برای ویرایش اونها لازم است .

### تصور غلط درباره بکارگیری OOP در PHP

قبل از ادامه بحث دوست دارم یه مقدار درباره این موضوع که PHP 4 شئ گرایی را پشتیبانی نمی کند صحبت کنم. بریم سر اصل مطلب :

فقدان وجود کنترلهای دستیابی: گرچه این در php5 تغییر میکنه (private, public و protected اضافه خواهند شد) به عقیده بنده این پارامتری نیست که جلوی ما را در بکارگیری OOP بگیره، کنترلهای دستیابی به هیچ وجه به منظور ایجاد امنیت بوجود نیامدند، بلکه برای کمک کردن به برنامه نویس درست شدند که برنامه نویس لازم نباشه نگران باشه که تصادفی به اعضایی دسترسی پیدا شود که نباید دست بخورند. فقط باید حواسمون رو بیشتر جمع کنیم. اینکه مفسر برنامه این اجازه را از شما نمی گیره به این معنا نیست که شما تئوری oop رو نمی تونید بکار بگیرید .

فقدان کلاسهای مجرد: بازهم این نمی تونه شما رو از کدنویسی شئ گرا باز نگه داره حتی در تئوری! اساساً یک کلاس مجرد به این معناست که شما باید در

هنگام ایجاد زیرکلاسها متدهای مجرد پیاده سازی کنید، و قادر نباشید از آن instance بسازید(خدا وکیلی اینو خودم هم نفهمیدم چی گفت). بعضی از زبانها مثل جاوا بصورت built-inاین موضوع را پشتیبانی می کنند که خیلی خوبه ولی بدون اون هم زندگی غیر ممکن نیست! بسادگی میشه یک متد خالی در superclass رو در زیر کلاسها overrideکنید.(که در قسمت پلی مورفیسم نشان خواهم داد چگونه).

عدم پشتیبانی از ارث بری چندگانه: گرچه بعضی اعتقاد دارند که ارث بری چندگانه مفید هست ولی به عقیده من فقط ایجاد ابهام میکند. شما چندتا مثال سراغ دارین که در آن یک شئ از دو یا چند superclass ارث ببره و هنوز بتونه رابطه is-a را حفظ کنه (اگر معنی این رابطه را نمی دانید من پایینتر توضیح دادم).
گرچه عقاید متفاوتند ولی من فکر نمی کنم این محدودیتها کاملا قابل چشم پوشی هستند و اصلاً دلیل خوبی برای ترک شئ گرایی در phpنیستند.

## درک اشیاء

وقتی داریم تکنیک های جدید برنامه نویسی را یاد می گیریم، معمولاً خوبه که از نمونه های دنیای واقعی کمک بگیریم. اینجا می تونیم یک ماشین را در نظر بگیریم. فکر کنید یک ماشین چه چیزهایی دارد (properties) و چه کارهایی می تواند انجام دهد. (behaviour)
چیزهایی که ماشین دارد (properties) :
پنجره
در
چرخ
موتور
در اینجا لازم به ذکر است که که properties خودشون می توانند شئ باشند با properties و behaviour خودشان. در شئ گرایی این ترکیب نام دارد. شما می توانید بگویید که ماشین تشکیل شده از... و حتی پا را فراتر بگذارید و بگویید که هر شئ هم از اشیاء دیگر تشکیل شده و .... ولی در موقع برنامه نویسی سعی کنید فقط چیزهایی را تعریف کنید که بدرد میخورند و بیخودی وارد جزئیات نشوید.

خوب بریم ببینیم یک ماشین چه کار می تواند بکند :
شتاب بگیرد
ترمز کند
درها باز شوند
این رفتارها مخصوص خود ماشین هستند. شما باید قادر به تشخیص اشیاء در واقعیت باشید تا بتوانید خوب از معماری شئ گرا استفاده کنید واقعاً احمقانه نیست که در زندگی عادی شئ گرا فکر کنیم. کل عالم از اشیاء تشکیل شده، سعی کنید اشیاء را شناسایی کنید!

## رکن اول: کپسوله کردن

از اینجا به بعد باید از یک زاویه دیگه به قضیه نگاه کنیم. آیا یک ماشین می تواند خودش براند؟ البته که نه، وقتی شئ گرا نگاه کنیم خود راننده یک موجودیت جداست.
خیلی مهم است که این تفاوت را حس کنیم: اشیاء باید مسوولیت های خودشان را داشته باشند. اونها فقط باید قادر باشند کارهای خودشان را انجام بدهند و نه چیزی بیشتر در واقع property های یک شئ فقط باید تحت تأثیر رفتارهای همان شئ تغییر کنند. اینکه یک شئ مستقیماً بتواند روی property های شئ دیگر تأثیر بگذارد بکلی غلط است. یک شئ باید جزئیاتش را برای خودش حفظ کند و آنچه که بروز می دهد فقط interface اش باشد و یک شئ دیگر فقط به جزئیات آن شئ از طریق این interfaceمی تواند دسترسی داشته باشد. وقتشه که این تئوری را ازطریق یک کد بازگو کنیم: یک کارخانه اتومبیل می خواهد اتومبیل بسازد ولی فقط در سه رنگ :
قرمز، آبی و سبز . برای اینکه بدونید مسوولیت یعنی چی من اعتراف می کنم مثال اولم مثال خوبی نبود! در ضمن syntax برنامه ربطی به من نداره می تونید
php manual را تحت عنوان Class/object functions مطالعه کنید.

```php
<?php
        class Car
        {
        var $color;
        }

        $specificCar = new Car();
        $specificCar->color = "yellow";
        ?>
```

اشتباه! قانون ما این بود که هیچ رنگی غیر از قرمز و آبی و سبز ساخته نشود. خوب معلومه آخر عاقبت اداره کردن مستقیم colour property همینه دیگه! اتومبیل موند رو دستمون .

چون ما یک interface برای مراقبت از مسوولیت ها ایجاد نکردیم. در لیست ۱ مثال بهتری را خواهید یافت .

لیست ۱ :

```php
<?php
        class Car
        {
        var $color;
        var $possibleColors = array("red", "green", "blue");

        function setColor($color)
        {
        if (in_array($color, $this->possibleColors)) {
        $this->color = $color;
        }
        }
        }

        $specificCar = new Car();
        $specificCar->setColor("yellow"); //would not alter the object

        $specificCar->setColor("red"); //would alter the object
        ?>
```

سعی کنید که این کد را درک کنید و اینکه چرا این راه بهتر است، این بار اتومبیل ما یک interface دارد که ما از طریق این interface می توانیم مقادیر properties را تغییر دهیم، متد setColor مراقب هست که چه اتفاقاتی ممکن است برای properties اتومبیل بیفتد. اگر interface امان را خوب بنویسیم، می توان مطمئن بود که

که اگر به غلط هم مقداردهی شود اتفاق بدی برای properties نخواهد بود و interface مراقب اوضاع خواهد بود. متد setColor اصلاح کننده (modifier) نام دارد و

propertyشئ اتومبیل را اصلاح می کند. در کنار اصلاح کننده، accessorها را داریم . برای برگرداندن property بجای اصلاح آن بکار می رود .

لیست ۲ مثالی برای accessor را نشان می دهد .

لیست ۲ :

```php
<?php
class Car
{
var $color;

/*
* constructor, code inside this function is executed on object
```

```
* initialisation
* note that in PHP 5 the constructor will have to have the name
* __construct(), instead of the class name
* although this way of constructing will still work as long as no
*___construct() is found
*/
function Car()
{
$this->color = "red";
}

// Accessor
function getColor()
{
return $this->color;
}
}
?>
```

در لیست ۲ متد getColor همان accessor است accessor . ها همچنین می توانند داده را قبل از برگرداندن اداره کنند. همیشه بهتر است که بجای ارجاع مستقیم به property های داخلی یک شئ از accessor ها استفاده کنیم. حالا می دونم بعضی از شما می پرسید که "من واقعاً نمی تونم property های شئ را مستقیماً اداره کنم؟"

درسته. وقتی این مقاله نوشته می شد php هنوز فاقد اصلاح کننده هایی مثل private بود. هرچند این موضوع در php5 تغییر خواهد کرد شما می توانید محدودیت های سر سختی برای قسمت های خصوصی خودتان بگذارید تا خیالتان راحت باشد که بهیچ وجه دستکاری نمی شوند . خیلی مهم است که رفتار شئ شما بخوبی از طریق interface یا متدها تعریف شده باشد. چون خیلی مهم است که یک interface خوب در موقع استفاده دوباره هم خوبباشد، باید واضح باشد که یک شئ چکار می کند. همیشه سعی کنید اسامی برای کلاسها انتخاب کنید که کار اون کلاس را بطور خلاصه بیان کند. از نوشتن اسامی بلند نترسید

در مجموع کپسوله کردن یعنی :

اشیاء جزئیات را برای خودشون نگه دارند

یک شئ فقط باید چیزی را ارائه کند که لازم است

یک شئ مسوولیت های خودش را داره

یک شئ باید interface واضحی داشته باشد.