

PHP – Reflection

Reflection Nedir ?

Reflection (Yansıma) kısaca açıklanabilecek bir konu olmasa da olabildiğinde örneksel anlatmaya çalışacağım. Reflection bize çalışma anında hiç bilmediğimiz bir yerden gelebilecek bir objeyi ,bir sınıfı veya sınıf elemanını inceleyip hakkında bilgi sahibi olabilme olanağı tanıyor.

Nerde İhtiyaç Duyarız ?

Reflection bize Plugin tabanlı bir sistem hazırlama konusunda çok faydalı olabilir. Modül ve ya Plugin sistemlerinde kullanıcılardan istediğimiz sabit şeyler olabilir.

- Modülün adı ne ?
- Nerde çalışmalı ?
- Hangi şartlarda çalışmalı ?

Bu bilgileri bize modülün kendisi verilmeli değil mi ? verdiği taktirde biz de bunu formatımıza uygun olarak modülün istediği gibi işleyebiliriz. Elimizde modülün kendisine özel sınıflarından birinden türetilmiş bir nesne olduğunu düşünelim. Bu elemanın “name” adında bir propertysi var mı ? “calistir()” adında bir metodu var mı? Hataları yapımçıya gösterebilmek ve ya Modülü çalıştırabilmek için bunlara ihtiyacımız var . Reflection olayını kısaca kafamızda oluşturmuş olduk.

PHP, 5 ten sonra Reflection adına pek çok yenilik yaptı ve 5.3 ile Reflection gerçek adını aldı (Namespace ile beraber ihtiyaç duyuldu diye tahmin ediyorum.)

PHP’de (script dillerinin genelinde) Reflection olayı özel bir çaba (ekstra sınıflar) gerektirmeden yapılabilir. Bunu en iyi yapan script dili tahminimce PHP . Fakat bu olayı adlandırmak ve bir sınıfa toplamak güzel olmuş. İlk önce PHP ‘nin bize Reflector sınıflardan önce Reflection adına sunduğu ve daha önce [Lambda](#) konusunda kısaca değindim fonksiyonları incelemek istiyorum.

<code>get_included_files();</code>	<code>get_called_class();</code>
<code>get_defined_constants();</code>	<code>get_class_methods();</code>
<code>get_defined_functions();</code>	<code>get_class_vars();</code>
<code>get_defined_vars();</code>	<code>get_parent_class();</code>
<code>get_declared_classes();</code>	<code>get_object_vars();</code>
<code>get_declared_interfaces();</code>	<code>get_extension_funcs();</code>
<code>gettype();</code>	<code>get_loaded_extensions();</code>

Reflection sınıfı olmadan da bu metotlarla Reflector sınıfımızı yazıp tek tek ReflectionObject ten tutun Reflection sınıfına kadar yazabiliriz.

PHP Reflection sınıfına neden ihtiyaç duydu ?

Bu soruyu kendimize sormamız doğal. Hem bunların hepsini yapabildiğimizi söyledik hem de PHP nin 5.3 ten sonra Reflection olayına isim verdiğini söyledik. Bu işi kendi başımıza halledebiliyorsak neden bize reflection sınıfı yazıldı ? Bunun cevabını yine 5.3 ile beraber gelen özelliklerde aramalıydık ki bu analiz sonucunda çıkarabileceğimiz tek sonuç [Namespaces](#)

Namespaceler ile çalışınca sınıflarımıza reflekte olaylarında belki metot belki property hatta bu modifier incelemesi bile yapabiliriz ancak namespace incelemesi yapamayacağımızı verdiğimiz fonksiyonların isimlerinden bile anlayabiliriz.

Reflection Classes.

The Reflector interface

```
Reflector {  
    abstract public static string export ( void )  
    abstract public string __toString ( void )  
}
```

Tüm export edilebilir Reflection sınıfları bu arayüze sahip olmalıdır. Olmasa dahi sınıflarımız için reflekte işlemleri yapabiliriz (Bir daha bu cümleyi kurmamaya özen göstereceğim fakat unutmayın ki Reflection olayını zaten yapabiliyorduk). Fakat bu arayüz sayesinde diğer reflection sınıfları içinde kullanabileceğimiz objeler oluşturabilir hatasız işlem yapabiliriz.

Reflection sınıflarının 2 temel metodu da parametre almıyor __toString sihirbazını tanıtmaya fırsatım olmasa da bunları [OOP magics](#) sayfasından öğrenebilirsiniz. Export aslında __toString işlevi görüyor diyebiliriz. Fakat biz __toString metodunu sınıfın kendisini explain etmek için kullanırdık Reflection sınıflarında da exportu Reflekte edilmesi istenen sınıfı explain etmek için kullanacağız.

Kısaca `__toString()` Sihirbazı

Örnek üzerinden açıklamak istiyorum öncelikle bir sınıfımız olsun.

```
class Yazilimci
{
    public $name,$surname;
}
```

Örnek bir de kullanım yapalım

```
$tufan=new Yazilimci();
$tufan->name="Tufan Barış";
$tufan->surname="YILDIRIM";
```

bunu biz "Tufan Barış YILDIRIM" şeklinde yazdırmak istediğimizde

```
Echo $tufan->name." ". $tufan->surname;
```

Bu kullanım bize ad soyadı verir. Ancak. Buna bir metod hazırlamak istediğimizi varsayarsak

```
class Yazilimci
{
    public $name,$surname;

    public function isimYaz()
    {
        return $tufan->name." ". $tufan->surname;
    }
}
```

Şimdiki kullanımı ve bu metodun bize sağladığı yararı inceleyelim

```
Echo $tufan->isimYaz();
```

Evet metod bize isim yaz dediğimizde çok güzel bir şekilde ad ve soyadı yazdırıyor .

Peki objeye string gibi davranıp ekrana yazmaya kalkarsak ne olur ?

```
Echo $tufan;
```

5

Catchable fatal error: **Object of class Yazilimci could not be converted to string**

Yazilimci türündeki bir obje string e dönüştürülemedi. **\$tufan** değişkeni bizim için bu sınıfın verilerini tutan bloğun işaretçisini (Pointer) tutuyor değil mi ? O zaman bunu ekrana basabilmek için bu bloğun bir string olması gerekiyor ama değil. Şimdi __toString() sihirbazının yaptığı işi görmek için sınıfı tekrar yazalım.

```
class Yazilimci
{
    public $name,$surname;

    public function __toString()
    {
        return $tufan->name." ". $tufan->surname;
    }
}
```

Şimdi, hata aldığımız kullanımı tekrar denediğimizde ekrana çıkan sonuç "Tufan Barış YILDIRIM" obje stringe dönüştürülmek istendiğinde tetiklenen metodumuz. String dönüştürme ya da string gibi kullanmaya kalkmaya birkaç örnek verelim.

```
Echo $tufan;

Print $tufan;

Echo "Yazanın adı : ". $tufan;

$tamadi=(string)$tufan;
```

Bunlara bir örnek daha ekleyebiliriz. String parametre isteyen bir fonksiyona direk bu objeyi vermek gibi. Fakat fonksiyon içerisine girene kadar bu obje olarak kalacaktır. İçerikteki kullanım yine bunlardan biri olacağı için örnek verme gereği duymadım , değinmeden de geçmek istemedim.

```
Class ReflectionProperty implements Reflector
{
    /* Constants */
    const integer ReflectionProperty::IS_STATIC = 1 ;
    const integer ReflectionProperty::IS_PUBLIC = 256 ;
    const integer ReflectionProperty::IS_PROTECTED = 512 ;
    const integer ReflectionProperty::IS_PRIVATE = 1024 ;
    /* Properties */
    public $name ;
    public $class ;
    /* Methods */
    final private void __clone ( void )
    __construct ( mixed $class , string $name )
    public static string export ( mixed $class , string $name [, bool $return ] )
    public ReflectionClass getDeclaringClass ( void )
    public string getDocComment ( void )
    public int getModifiers ( void )
    public string getName ( void )
    public mixed getValue ( [ string $object ] )
    public bool isDefault ( void )
    public bool isPrivate ( void )
    public bool isProtected ( void )
    public bool isPublic ( void )
    public bool isStatic ( void )
    public void setAccessible ( bool $accessible )
    public void setValue ( object $object , mixed $value )
    public string __toString ( void )
}
```

ReflectionProperty sınıfı Reflection işlemlerinde elde ettiğimiz her property nin türüdür. Elemanlarını bir bir inceleyelim ;

Öncelikle üzerinde örneklendirme yapacağımız sınıfımızı tekrar yazalım

```
class Yazilimci implements Reflector
{
    public $name, $surname;
    private $girlFriend;

    public function __toString()
    {
        return $this->name." ".$this->surname;
    }

    public static function export($class,$name,$return=false)
    {
        return 'Requested Property Name is <b>'.$name.'</b> and Value is <b>'.$class->$name.'</b>';
    }
}
```

\$name - Property nin adı . örneğin Yazilimci türünden bir öğemiz olsaydı \$surname property'si için alacağımız değer " surname " olacaktı.

\$class- Property iyesi sınıfın adına denktir.Yine elimizdeki sınıf için incelersek bunun karşılığı "Yazilimci" olacaktı.

Metotlar

public static string export (mixed \$class , string \$name [, bool \$return])

Abstract interface (override edilmesi zorunlu) metotlardan biri olan bu metot bize reflector bir sınıf yazarken çıktıyı verebilecek metoddur. Private property'lere ulaşmamızı da sağlamış olacak ve reflection adına bize kolaylık sağlamış olacaktır.

İlk parametresi export etmek istediğimiz sınıf türünden bir obje name parametresi üzerinde işlem yapmak istediğimiz property'nin adı ve son return parametresi de çıktının geri döneceğini ya da ekrana direk basılacağını belirler.

public string __toString (void)

__toString() elemanını zorunlu kılması belki de Reflection sınıfının içerde kendisi için yaptığı bir kolaylıktır bilemiyorum. Lakin "Reflection::export" metodunun kullandığı da budur. Bize sadece Export metodunu sadece __toString() görevi yapmak zorunda olmama ve __toString() i de yine amacı dışında kullanabilme şansı tanımışlar.

Açıklamasını yaptığımız kullanımlara sınıfımız üzerinde bir örnek göstermeden önce ReflectionClass i tanıyalım:

The ReflectionClass class

ReflectionClass Sınıf reflekte edebileceğimiz yine Reflector kökenli bir sınıf. Propertyleri

```

$reflectMember=new Yazilimci();
$reflectMember->name="Tufan Barış";
$reflectMember->surname="YILDIRIM";

echo '<pre>';
echo Reflection::export($reflectMember);
echo Yazilimci::export($reflectMember,'name');
echo new ReflectionClass($reflectMember);

```

İlk echo bloğumuzda export() 'un __toString() i çalıştırdığını tekrar görüyoruz ekrana Tufan Barış YILDIRIM yazıyor. İkincisinde Yazilimci için özelleştirilen export metodundaki kod bloğunu çalıştıracaktır doğal olarak ve bizim sınıf, yazdığımız "Requested Property Name is..." ile başlayan cümleyi verecektir. Son Blokta ise ReflectionClass oluşturup __toString() metodunun tetiklenmesini sağladık işte genel çıktı

```

Tufan Barış YILDIRIM
Requested Property Name is <b>girlFriend</b> and Value is <b></b>Class [ <user> class Yazilimci implements Reflector ] {
  @@ C:\AppServ\www\test\enc.php 2-17

- Constants [0] {
}

- Static properties [0] {
}

- Static methods [1] {
  Method [ <user, prototype Reflector> static public method export ] {
    @@ C:\AppServ\www\test\enc.php 12 - 16

- Parameters [3] {
  Parameter #0 [ <required> $class ]
  Parameter #1 [ <required> $name ]
  Parameter #2 [ <optional> $return = false ]
}
}

- Properties [3] {
  Property [ <default> public $name ]
  Property [ <default> public $surname ]
  Property [ <default> private $girlFriend ]
}

- Methods [1] {
  Method [ <user, prototype Reflector> public method __toString ] {
    @@ C:\AppServ\www\test\enc.php 7 - 10
  }
}
}

```

Bu çıktı incelenerek oluşan obje hakkında bilgi edinilebilir. Objenin türetildiği sınıfın tanımlandığı dosyanın satır ve sütun numarası dahi var. Metot ve property listesi de modifierlarıyla beraber listeleniyor. Reflection sınıfını bu şekilde ekrana basmak için kullanmayacağız tabi ki. Programda veri niteliği taşıyabilmesi için New ReflectionClass ı bir değişkene atayıp istediğimiz gibi kullanacağız.

Not :

Reflection Sınıfları sadece derleme sırasında (Compile Time) oluşan reflekte bilgisini dikkate alır. Çalışma zamanında (Runtime) oluşan property ve ya metotlar için Reflekte işlemi yapılamaz.

Notunumuzu bir örnekle açıklayacak olursak

```

Class Reflekte
{
    public function Reflekte()
    {
        $this->yeniProperty="Yeni değer"
    }
}

```

Böyle bir sınıfımız olduğunu varsayarsak.” yeniProperty “ adındaki propertyimiz Constructor içinde oluşmuş olacak ve değer ataması yapılacak . Fakat reflekte işlemine geldiğimizde

```

$reflekte=new Reflekte();
$Reflection=new ReflectionProperty($reflekte,'yeniProperty'); // Hata Fırlatır.

```

Reflection Sınıfları

Reflector sınıflarının temeli aynı olduğu için Sadece *ReflectionProperty* ve *ReflectionClass* üzerinde durduk. Diğer sınıflarımızda temel aynı sadece kullanım alanı farklı.

Reflector (Interface) – Sınıflarımız reflekte edilebilmesi için bu arayüze sahip olmalıdır.

ReflectionClass – Class Reflekte etmek için tasarlanmış Reflector.

ReflectionExtension – Extension reflekte etmek için tasarlanmış Reflector.

ReflectionFunction - Fonksiyon reflekte etmek için tasarlanmış Reflector.

ReflectionFunctionAbstract- ReflectionFunction için yazılmış base Reflector.

ReflectionMethod –Metot reflekte etmek için tasarlanmış Reflector.

ReflectionObject - Object Reflekte etmek için tasarlanmış Reflector.

ReflectionParameter –ReflectionFunction içinden ayrıştırılmış Parametreleri reflekte etmek için tasarlanmış Reflector.

ReflectionProperty - ReflectionClass içinden ayrıştırılmış Propertyleri reflekte etmek için tasarlanmıştır aynı zamanda kendi başına da oluşturulup 1 property üzerinde çalışılabilir.