A PHP5 Reflection API

A sorozat előző részeiben megismerkedhettünk a PHP5 objektumközpontú újításaival, megtanulhattuk azok rendeltetésszerű használatát, illetve elmélyedhettünk a programnyelv által rendelkezésünkre bocsátott különleges tagfüggvények adta lehetőségekben. A cikksorozat záró akkordjaként az újdonságnak számító Reflection API-t szeretném bemutatni, amellyel lehetőségünk nyílik a már elkészített függvények, osztályok, metódusok, paraméterek, objektumok, stb. elemzésére futásidőben.

Mi is ez valójában?

Ez az ***API (alkalmazás programozási felület)*** nem más, mint egy osztálygyűjtemény, amelyet a ***PHP*** bocsát rendelkezésünkre – hasonlóan az ***Exception*** kivétel osztályhoz. Az egyes programelemek elemzéséhez a megfelelő elemző osztályt kell kiválasztanunk, amelyek mindegyike a ***Reflection*** ősosztályból öröklődnek. A szolgáltatás igénybe vételéhez nincs is más dolgunk, mint a megfelelő konstruktor-paraméterekkel példányosítani az osztályokat, majd az adott példány metódusain keresztül elérni a paraméterek szerinti programelemek összes létező tulajdonságát. A továbbiakban megnézzük, hogy milyen osztályokkal van dolgunk, ezek mire használhatók – néhány rövid példával illusztrálva őket.

Sorban az első: ReflectionFunction

A nevéből már kitalálhattuk, hogy itt bizony az egyes függvények nyilvános és rejtett tulajdonságait nézhetjük meg közelebbről. Az itt leírt használati módszer egyébként az összes többi osztály esetében is hasonló, sőt, alapjaiban ugyanez, csak a későbbi osztályokban más és más (általában egyre nagyobb) az elérhető elemző tagfüggvények köre. Néhány fontosabb metódust az *1. táblázat* tartalmaz.

Mivel ez így még mindig homályosan hangzik kissé, lássunk egy példát az általános használatra, amelyben minden fenti metódusra szerepel egy-egy példa. Bár ez konkrétan a ***ReflectionFunction*** osztály hétköznapi viselkedését szemlélteti, a többi osztálynál sem mutatkozik jelentős eltérés, így hát valamiféle általános igazságként tekintsünk erre a programra.

1. táblázat *A ReflectionFunction néhány fontosabb metódusa*

`getName()`	Visszaadja a függvény nevét.
`getFileName()`	Megmondja, hogy mi a neve annak a fájlnak, amelyben szerepel.
`getStartLine()`	Melyik sortól kezdődik a függvény implementációja.
`getEndLine()`	Melyik sorban végződik a függvény implementációja.
`isInternal()`	Igaz, ha belső, *PHP*-s függvényről van szó.
`isUserDefined()`	Igaz, ha felhasználói függvényről van szó.
`getDocComment()`	Visszaadja a függvényhez tartozó *PHPDoc/JavaDoc*-nak megfelelő dokumentáció jellegű megjegyzést.
`invoke()`	Meghívja a függvényt a paraméterben szereplő paraméterekkel, majd visszatér a függvény visszatérési értékével.
`export()`	Statikus függvény, amely megjeleníti az alapértelmezett kimeneten az elemzett függvény szerkezetét, felépítését, általános adatait

```
<?php
//----A deklarációs rész-------------------------
/**
 * Szabványos módon kommentezett függvény,
 * amely négyzetre emeli a paramétert.
 *
 * Ez a megjegyzés phpdoc/javadoc alapú
 * (megfelelő), s a Reflection API-n
 * keresztül is lekérdezhető
 * @param      int     Bementő paraméter
 * @return     int     Visszatérési érték
 */
function negyzet($i=0) {
    return $i*$i;
}
```

```
//----Az elemző/kiíratórész----------------------
// létrehozzuk az elemző objektumot, amelyen
// keresztül a kívánt paraméterek
// lekérdezhetők – EZ ITT A LENYEG
$func = new ReflectionFunction('negyzet');

echo "<pre>";

//alap információ kiíratása
echo
"==========================================\n";
echo "Általános információk\n";
echo
"==========================================\n";
printf(
    "A függvény neve: %s\n".
    "A függvény típusa: %s\n".
    "A függvény deklarálásának helye:\n".
    "   fájl: %s\n".
    "   deklaráció kezdete: %d. sor\n".
    "   deklaráció vége: %d. sor\n",
   $func->getName(),
   $func->isInternal() ? 'belső' : 'fehasználói',
   $func->getFileName(),
   $func->getStartLine(),
   $func->getEndline()
);

// A dokumentáció jellegű komment kiíratása
echo
"\n==========================================\n";
echo "PHPDoc/JavaDoc stílusú komment:\n";
echo
"==========================================\n";
printf("%s\n", var_export($func->getDocComment(),
➥1));

//A függvény hívása
echo
"\n==========================================\n";
echo "Függvényhívás eredménye:\n";
echo
"==========================================\n";
var_dump($func->invoke(1));
var_dump($func->invoke(2));
var_dump($func->invoke(3));

//A paraméterek
echo
"\n==========================================\n";
echo "A függvény paraméterei:\n";
echo
"==========================================\n";
var_dump($func->getParameters(3));

//Az export metódus eredménye
echo
"\n==========================================\n";
echo "A függvény szerkezete:\n";
echo
```

2. táblázat *A ReflectionFunction néhány fontosabb metódusa*

`getClass()`	Visszaadja, hogy mely osztály példányát várjuk paraméterként (`NULL`, ha alaptípusról van szó) *ReflectionClass* típus formájában, amely azonnal további elemzést tesz lehetővé.
`getDefaultValue()`	Visszaadja a paraméter alapértelmezett értékét.
`allowsNull()`	Igaz, ha null érték megengedett a paraméter használata során.
`isPassedByReference()`	Igaz, ha referencia szerinti paraméterátadás formájában használjuk a paramétert (ha objektumot adunk át, ez mindig igaz lesz).
`isOptional()`	Igaz, ha opcionális paraméterről van szó.

3. táblázat *A ReflectionFunction osztálytól különböző néhány fontosabb metódus*

`getConstructor()`	Visszaadja az osztály konstruktorát a *ReflectionMethod* osztály példányaként
`getMethotds()`	Visszaadja az osztály metódusait a *ReflectionMethod* osztály példányaiként egy tömbben – a konstruktort is beleértve
`getProperties()`	Visszaadja az osztálytulajdonságokat a *ReflectionProperty* osztály példányaiként egy tömbben
`getConstants()`	Visszaadja az osztály konstansait egy tömb értékeiként
`isInterface()`	Igaz, ha felületet elemzünk
`isAbstract()`	Igaz, ha az osztály elvont
`isFinal()`	Igaz, ha nem örökíthető az osztály tovább
`isInstantiable()`	Igaz, ha példányosítható az adott osztály (pl. elvont osztály nem példányosítható)
`getParentClass()`	Visszaadja a szülő osztályt a *ReflectionClass* osztály példányaként
`isSubClassOf()`	Igaz, ha a paraméterben szereplő osztály alosztája a vizsgált osztály
`isIterateable()`	Igaz, ha az osztály iterálható (*PHP5*-ben ugyanis az osztályokon, objektumokon végigmehetünk egy **foreach**-el, mintha mondjuk tömbök lennének, ahol a tömb elemei az egyes tulajdonságok, tagfüggvények, konstansok, stb.)

```
"==========================================\n";
echo ReflectionFunction::export('negyzet');

echo "</pre>";
?>
```

A program futási eredményét terjedelmi okokból nincs értelme közölni, de mint az magából a kódból látható, egyfajta összefoglaló elemzést készít.

Sorban a következő: ReflectionParameter

Szorosan kapcsolódik az előző osztályhoz ez a függvény és tagfüggvény paramétereket elemezni tudó osztály. A kapcsolat valójában abban áll, hogy a paramétereket, mint osztályokat a `ReflectionFunction::getParameters()` metódusával kaphatjuk meg, nem példányosítással, mint az előző esetben. A fenti függvény tehát ***ReflectionParameter*** típusú elemek tömbjével tér vissza, amelyek sorban a függvény vagy tagfüggvény egyes paramétereit képviselik. A teljesség igénye a *2. táblázatban* felsoroltam néhány fontosabb tagfüggvényt.
A fenti példánál maradva elemezzük a függvény egyetlen paraméterét a szabványos módon:

```
//tovabbi elemzes
echo
"\n==========================================\n";
echo "A függvény paramétereinek elemzése:\n";
echo
"==========================================\n";
foreach ($func->getParameters() as $i => $param) {
    printf(
        "Paraméter: #%d\n".
        "----------------------\n".
        "       Neve: %s\n".
        "       Osztálya: %s\n".
        "       Alapértelmezett érték: %s\n".
        "       Null érték megengedett?: %s\n".
        "       Referenciaként átadva?: %s\n".
        "       Opcionális?: %s\n".
        "\n",
        $i,
        $param->getName(),
        var_export($param->getClass(), 1),
        var_dump($param->getDefaultValue()),
        var_export($param->allowsNull(), 1),
        var_export($param->isPassedByReference(),
        ➥1),
        var_export($param->isOptional(), 1)
   );
}
```

Jól látható, hogy a paraméter elemzés nem áll többől, mint végig menni a visszakapott paramétereken, s mindegyiknél valami hasonlót kell eljátszani, mint a függvény elemzésénél.

Osztályok vizsgálata a ReflectionClass segítségével

Felépítése többnyire a metódusok számában különbözik a ***ReflectionFunction*** osztályétól, s a használat során is leginkább a gazdagabb funkciók körében kell keresnünk a különbségeket. Igazából nem is csak osztályok, de ***felületek (interface)*** vizsgálatára is alkalmas. Az analízis tényleg teljes körű: Megmondja, hogy melyik osztályból öröklődik, melyik felületet valósítja meg, elvont-e (abstract), milyen metódusai vannak - ha vannak, példányosítható-e az osztály, visszakaphatjuk a konstruktort, destruktort (természetesen a ***ReflectionMethod*** osztály példányaként a további elemzés céljából), az osztálytulajdonságokat (osztályváltozókat) hasonló elemezhető formában, és még sorolhatnám. Néhány, a ***ReflectionFunction*** osztálytól különböző lényeges tagfüggvény leírását a *3. táblázat* tartalmazza.
Példaként álljon itt egy a ***ReflectionFunction*** osztálynál szereplő példához hasonló osztály-elemző program:

```
<?php
//----A deklarációs rész--------------------------
class Negyzet {
    protected $oldal = 0;

    public function __construct($oldal) {
        $this->oldalHossztBeallit($oldal);
    }

    public function oldalHossztBeallit($ertek) {
        $this->oldal=$ertek;
    }
}

class Rombusz extends Negyzet {

    protected $szog = 0;

    public function __construct($oldal,$szog) {
        parent::__construct($oldal);
        $this->szogetBeallit($szog);
    }

    public function szogetBeallit($szog) {
        $this->szog=$szog;
    }

    public function terulete() {
        echo $this->oldal*sin($this->szog)
        ➥*$this->oldal;
    }
}
$class = new ReflectionClass('Rombusz');

echo "<pre>";

//alap információ kiíratása
echo
"==========================================\n";
echo "Általános információk\n";
echo
"==========================================\n";
printf(
    "Az osztály neve: %s\n".
    "A osztály jellemzői: %s, %s, %s, %s\n".
    "A függvény deklarálásának helye:\n".
```

```
    “   fájl: %s\n”.
    “   deklaráció kezdete: %d. sor\n”.
    “   deklaráció vége: %d. sor\n”,
    $class->getName(),
    $class->isInternal() ? ‘belső’ :
    ➥‘felhasználói’,
    $class->isAbstract() ? ‘elvont’ : ‘nem elvont’,
    $class->isFinal() ? ‘ final’ : ‘gyermekből
    ➥felülírható’,
    $class->isInterface() ? ‘felület’ : ‘osztály’,
    $class->getFileName(),
    $class->getStartLine(),
    $class->getEndline()
);

// Szülő osztály dolgai
echo
“\n=========================================\n”;
echo “Szülő osztály szerkezete\n”;
echo
“=========================================\n”;
Reflection::export($class->getParentClass());

// Megvalósított felületek
echo
“\n=========================================\n”;
echo “Megvalósított felületek\n”;
echo
“=========================================\n”;
printf(“%s\n”, var_export($class->getInterfaces(),
➥1));

// Osztálytulajdonságok
echo
“\n=========================================\n”;
echo “Osztálytulajdonságok\n”;
echo
“=========================================\n”;
printf(“%s\n”, var_export($class->getProperties(),
➥1));

// Metódusok
echo
“\n=========================================\n”;
echo “Tagfüggvények\n”;
echo
“=========================================\n”;
printf(“%s\n”, var_export($class->getMethods(),
➥1));

echo “</pre>”;
?>
```

Szeretném felhívni a figyelmet a következő sorra: `Reflection::export($class->getParentClass());`. A *Reflection* ősosztály rendelkezik egy olyan, mindegyik gyermekéből elérhető statikus metódussal, amely előre meghatározott formában a szabványos kimenetre küldi a paraméterként átadott osztály, függvény, paraméter, stb. szerkezetét. A fenti programban ezt arra használtuk, hogy megmutassuk az ősosztály felépítését.

Ha tovább szeretnénk menni, némi rekurzió alkalmazásával elérhetjük, hogy nem csak a közvetlen őst, de az osztály összes felmenőjét megvizsgáljuk, amelyet alaposan megkönnyít, hogy az ősosztály is egy ilyen *ReflectionClass* osztály példányaként érkezik, aminek szintén van `getParentClass()` metódusa, tehát nekünk szinte már semmi dolgunk.

4. táblázat *A ReflectionMethod néhány fontosabb tagfüggvénye*

`isFinal()`	Igaz, ha a tagfüggvény a gyermekekben nem írható felül.
`isAbstract()`	Igaz, ha a tagfüggvény elvont.
`isPrivate()`	Igaz, ha a tagfüggvény csak az osztályon belül érhető el (hasonlóan: `isProtected()`, `isPublic()`).
`isStatic()`	Igaz, ha a metódusunk statikus.
`isConstructor()`	Igaz, ha az aktuális metódus az osztály konstruktora (hasonlóan: `isDestructor()`).

A ReflectionMethod osztály

Ez az előző bekezdésben már emlegetett osztály az egyes tagfüggvények elemzésére szolgál. Keletkezését tekintve tudni kell róla, hogy a *ReflectionFunction* osztályból öröklődik, így mind működésében, mind feladatában igen hasonlít az ősére, azzal az apró különbséggel, hogy tartalmaz néhány tagfüggvényt a metódusok függvényekhez képest plusz tulajdonságainak lekérdezésére. A fontosabb tagfüggvények leírását a *4. táblázat* tartalmazza.

Egészítsük ki az előző példánkat az alábbi tagfüggvényelemző kódrészlettel:

```
//Metódusinformáció
echo
“\n=========================================\n”;
echo “Metódusok adatai\n”;
echo
“=========================================\n”;
foreach ($class->getMethods() as $i => $method) {
    printf(
        “Metódus: %d\n”.
        “----------------------\n”.
        “       Neve: %s\n”.
        “       Tulajdonságai: %s, %s, %s, %s\n”.
        “       Típusa: %s\n”.
        “       Deklarálásának helye:\n”.
        “               fájl: %s\n”.
        “               deklaráció kezdete: %d.
                        ➥sor\n”.
        “               deklaráció vége: %d.
                        ➥sor\n”.
      “\n”,
```

5. táblázat *A ReflectionProperty néhány fontosabb metódusa*

`isPrivate()`	Igaz, ha az osztálytulajdonság csak az osztályon belül érhető el (hasonlóan: `isProtected()`, `isPublic()`).
`isStatic()`	Igaz, ha az az osztálytulajdonság statikus.
`isDefault()`	Igaz, ha már a fordítás során deklarált a paraméter, hamis, ha csak futásidőben tesszük ezt meg.
`getValue()`	Lekérdezhetjük az értékét.
`setValue()`	Beállíthatjuk a tulajdonság értékét, amely akkor lehet hasznos, ha a `ReflectionMethod::invoke()` metódusával szeretnénk meghívni a függvényt, és az esetleg használja valamelyik osztálytulajdonság értékét is.

```
            $i,
            $method->getName(),
            $method->isAbstract() ? 'elvont' :
            ➥'nem elvont',
            $method->isFinal() ? 'final' : 'gyermekből
            ➥felülírható',
            $method->isStatic() ? 'statikus' : 'nem
            ➥statikus',
            $method->isPublic() ? 'public' : $method->
            ➥isPrivate() ? 'private' : 'protected',
            $method->isInternal() ? 'belső' :
            ➥'fehasználói',
            $method->getFileName(),
            $method->getStartLine(),
            $method->getEndline()
        );
}
```

Eredményül megkapjuk az osztály jellemzői alatt a hozzá tartozó metódusok (örökölt és nem örökölt egyaránt) tulajdonságait.
Ennél természetesen még tovább is mehetnénk.
A *ReflectionParameter* osztály esetében már láttunk arra példát, hogy a *ReflectionFunction* osztály által visszaadott paramétereket hogyan kell 'bejárni', ezáltal elemezni. Ezt itt is megtehetjük, ha másért nem, hát gyakorlásképpen, és hozzáírhatjuk az egyes metódusokhoz, hogy milyen tulajdonságú paraméterekkel rendelkeznek. Kezd körvonalazódni, hogy ezek az osztályok egymással egész kis láncolatot alkotnak, jelentősen egyszerűsítve nekünk, fejlesztőknek a munkáját.

A ReflectionProperty osztály

Mint tudjuk, egy osztálynak nem csak tagfüggvényei, hanem tulajdonságai is vannak. Az osztály-elemző példában ezt elintéztük annyival, hogy rázúdítottuk a kimenetre a `ReflectionClass::getProperties()` metódusának

visszatérési értékét. A kimenetből láthattuk is, hogy ezek ***ReflectionProperty*** típusúak, így a már megszokott módon végigmehetnénk a paramétereken, hogy kicsit azokat is elemezzük. Mielőtt ezt mentenénk, nézzük, milyen tulajdonságokat kérdezhetünk le egy ilyen osztálytól (*5. táblázat*). Most pedig cseréljük ki az osztálytulajdonságok elemzését végző részt az alábbi kódrészletre:

```
// Osztálytulajdonságok
echo
"\n=========================================\n";
echo "Osztálytulajdonságok\n";
echo
"=========================================\n";
foreach ($class->getProperties() as $i => $method)
{
    try {
        $value=var_export($method->getValue(),1);
    } catch (Exception $ex) {
        $value='private vagy protected,
        ➥nem lekérdezhető';
    }

    printf(
        "Osztálytulajdonság: %d\n".
        "------------------------\n".
        "       Neve: %s\n".
        "       Tulajdonságai: %s, %s, %s\n".
        "       Értéke: %s".
       "\n",
        $i,
        $method->getName(),
        $method->isStatic() ? 'statikus' :
        ➥'nem statikus',
        $method->isPublic() ? 'public' : $method->
        ➥isPrivate() ? 'private' : 'protected',
        $method->isDefault() ? 'fordítási időben
        ➥deklarálva' : 'futásidőben megadva',
        $value
   );
}
```

Egyetlen említésre méltó elem a kódrészletben a kivételkezelési rész. Ha ugyanis nem publikus értékkel dolgozunk, akkor egy kivétellel jelzi az osztály, hogy ez bizony nem fog menni. Általában is igaz, hogy ha olyan műveletet szeretnénk végezni, amely valamilyen ok miatt nem lehetséges, s általános esetben programhibát okozna, arról kivételt generál a ***PHP***, amelyet nekünk kell lekezelnünk.

Mondhatjuk, hogy a végére értünk a legfőbb osztályoknak (természetesen akad még ilyen funkciójú ***Reflection*** osztály), s most nem árt feltennünk a kérdést, hogy mégis mire jó ez, mégis miért dolgoztunk ennyit?

A válasz tömören az, hogy minden esetben hasznos lehet, ahol általános, vagy ismeretlen eredetű osztályok, függvények együttműködését kell garantálnunk, például ha több fejlesztő dolgozik együtt, vagy ha olyan átfogó „anyaosztályokat” készítünk, amelyekkel a programozói akadályok sokféleségét kezelni tudjuk.

A konkrét példáknál maradva előfordulhat, hogy olyan osztályokkal dolgozunk, amelyek többfélék lehetnek – az internetes környezetben úgyis gyakran fordulnak elő ún. félstrukturált adatok, amelyek vagy ilyenek, vagy olyanok, az mindig az adott körülményektől függ. Meglehetősen nagy annak a valószínűsége, hogy ugyanazt az általunk várt bemenetet többféle osztály is képviselheti, s nekünk a többféleség függvényében más és más metódusokat kell meghívnunk. Ekkor lehet hasznos vagy az osztály elemzése, vagy a lehetséges metódusok hívogatása, kezelése kivétellel, ezáltal annak kitapogatása, hogy vajon milyen bemenettel is van dolgunk. Enélkül elképzelhetetlen, hogy ne kapjunk programhiba-üzenetet, amit ugye senki sem szeret, ráadásul a programunkra sem fogható rá, hogy tökéletesen működik.

Ennél különlegesebb esett, ha szeretnénk egy automatikus dokumentáció-készítő programot csinálni, amely fejlesztői leírást készít minden általunk készített programelemre, méghozzá egyformán és gyorsan. Minden fejlesztő utál dokumentációt készíteni, pedig ha van, az a későbbiekben megoldást jelenthet egy rakás problémára. Normális esetben ez úgy lehetséges, ha elemezzük a ***php*** fájlokat, tehát írnunk kell egy kisebb fajta ***PHP*** motort – mondanom sem kell, mekkora emberi ráfordítással. Ha azonban kihasználjuk a ***Reflection API*** nyújtotta lehetőségeket, akkor már az itt szereplő kódrészlet nem túl jelentős átalakításával is kaphatunk például ***HTML*** alapú dokumentáció-készítő programocskát, amivel ugyan kézzel és egyenként kell végigmennünk az osztályokon, objektumokon, de legalább a dokumentációt nem nekünk kell megcsinálni.

Egyszerűsíthetjük is a dolgot, s annak a bizonyos `getDocComment()` metódusnak a visszatérési értékét elemezve készíthetünk egy egyszerű kis fejlesztői leírást, amelyet odaadhatunk a fejlesztői csapat többi tagjának, ezáltal csupán a kommentezéssel megússzuk a dokumentáció készítését, ha ügyesek és körültekintőek vagyunk. Megjegyezném, hogy ilyen programok már léteznek, a fenti elvárásokat teljesíti például a ***PHPDocumentor (http://www.phpdoc.org/)***, amely egyike a legjobbaknak

Láthatjuk, hogy nem épp hétköznapi igények megoldására használhatjuk igazán hatékonyan a ***Reflection API***-t, de azért számtalan olyan helyzetet találunk, amelynél szintén nagy könnyebbséget jelent a használata. Sajnos nem állunk túl jól a témáról fellelhető leírást illetően.

A ***Google*** által első helyeken felkínált találatok mind-mind a ***http://www.php.net*** weboldalon található dokumentációt követik, ami viszont nem valami bőbeszédű, sok helyen pedig messze nem teljes körű. Öröm az ürömben, hogy a ***Reflection API*** kiválóan alkalmas ám önmaga elemzésére is, ezáltal a hiányzó információkhoz viszonylag könnyedén hozzájuthatunk egy-két ***PHP*** parancs kiadásával – jelen cikk írása során is előfordult párszor, hogy a tárgyalt téma képviselőjére bíztam önmaga feltérképezését.

Komáromi Zoltán
(komi@kiskapu.hu)
23 éves, a BME hallgatója, mellette PHP-programozóként dolgozik. Kedvenc területe a multimédia.