A PHP5 újdonságai

Újraírt Zend-motor, gyorsabb futás és továbbfejlesztett objektumkezelés – mindez év vége táján várható.

Nyár van, mégsincs uborkaszezon a PHP háza táján. Június végén a nagyközönség elé tárták a PHP 5.0.0-s próbaváltozatát. A fejlesztés tehát eljutott első nyilvános megjelenéséig (eddig csak CVS segítségével lehetett beszerezni), s már látni lehet az alagút végét. Számos jelentős változás lesz a 4-es vonalhoz képest, de semmi olyan, ami miatt meglévő PHP-programjainkat újra kellene írnunk, legalábbis nem nagyobb mértékben, mint az a 3-asról a 4-esre váltás során szükségessé vált. Nagyszámú olyan újdonság jön viszont az 5-ös változattal, amit érdemes megismerni.

Névterek

Hasznos újdonság lesz a PHP életében a névterek bevezetése. Ez az apró kis adalék nagymértékben segítheti a nagyobb lélegzetű munkák szerkezetének a kialakítását, ráadásul külső osztálykönyvtárak és egyéb kódok anélkül könnyedén együtt használhatók, hogy bármiféle osztálynév-, állandó- (constans) vagy változónév-ütközés veszélye állna fenn. Márpedig ilyen jellegű galiba akár akkor is előfordulhat, ha csak egyszerűen többen dolgoznak ugyanannak a projektnek más-más részein, és ugyanazt a változónevet eltérő célból veszik használatba.
A névterekkel némi logikai csoportosításra is lehetőségünk nyílik, az összetartozó osztályokat, állandókat egy térbe vonhatjuk. Névtereket létrehozni a `namespace` kulcsszó segítségével tudunk. Lássunk egy egyszerű példát (lásd az *1. listát*)!
A példakód természetesen csak szemléltetésre alkalmas, de a lényeg látható. A namespace „kulcsszót” egy név követi, amellyel a továbbiakban hivatkozni tudunk rá. A terület körülhatárolására hagyományosan a kapcsos zárójelek hivatottak. A névtér egy osztályhoz vagy függvényhez hasonlóan saját hatókörrel rendelkezik. Ez annyit tesz, hogy minden, a névtérben létrehozott függvény, változó, állandó a névtéren belül hozzáférhető, míg azon kívül közvetlenül nem. Viszont elérhetők, ha egy kis előtaggal fejeljük meg, azaz a névtér nevével és egy kettőzött kettősponttal. A névtér előtagot és az utána következő változó-, állandó-, függvény-, illetve osztálynevet hívjuk az objektumközpontú világban „minősített névnek”. Ilyen formában nemcsak a névtereken kívül tudunk hozzáférni az azokon belül létrehozott elemekhez, de ilyen módon az egyik névtérből át is nézhetünk a másikba.
A PHP-értelmezőnek az sem okoz gondot , ha névterünket esetlegesen több darabban hozzuk létre, azaz ugyanazt a névteret akár több állományban is megnyithatjuk, hogy elemeket pakoljunk bele. A névterek egymásba is ágyazhatók, azaz névtereken belül is létrehozhatunk külön névtereket. A kívülről való hivatkozáshoz a minősített névben a szülőnévtértől kezdve a teljes névtércsaládfát le kell írnunk. A névtérhivatkozásokat azonban egymástól csak egyszeres kettősponttal szükséges elválasztanunk (lásd az 1. listát).

1. lista A namespace.txt

```
?php

namespace Mail {
     const FELADO = 'kosar@hajo.hu';
     $level_darab = 0;
     function jelentes($cimzett, $szoveg) {
          mail($cimzett, 'Jelentes', $szoveg,
               ➥'From: '.FELADO);
     }
}

echo Mail::FELADO;
echo Mail::$level_darab;
Mail::jelentes('kapitany@hajo.hu', 'Szárazföld!');

?>
```

3. lista Az object-overloading.txt

```
<?php

class Gyenge_pelda_tulterhelesre {

     __call($fuggveny, $parameterek) {
          if ($fuggveny = 'tukroz') {
               if
(is_numeric($parameterek[0])) {
                    return $this->
                    ➥tukroz_szamot
                    ➥($parameterek[0]); } else {
                    return $this->
                    ➥tukroz_szoveget
                    ➥($parameterek[0]);
               }

          }
     }

     private function tukroz_szamot($szam) {
          return 0 - $szam;
     }

     private function
tukroz_szoveget($szoveg) {
          return sttrev($szoveg);
     }

}

?>
```

Egyelőre még semmi sem biztos...

A PHP eljövendő új változatáról még nem sok biztosat lehet elmondani, bármi és bármikor megváltozhat a végleges kiadásig. Néhol igen hosszúra nyúlt viták következményeképpen a dolgok megerősödnek, eltűnnek vagy megváltoznak. A cikk írása közben például kiderült, hogy egyre kevésbé valószínű, hogy a névterek bekerülnek a végleges ötös változatba. Még ez a játszma sincs lejátszva, sokan kardoskodnak a dolog mellett, de vannak erős érvek az elhagyása mellett is. Jelenleg az utóbbi áll nyerésre. A nehézség a nem teljesen tökéletes megvalósíthatóságban rejlik. Az egyik ellenérv szerint: írásmódját tekintve a „(feltétel) ? ha_igaz : ha_hamis" megoldással is ütközik a névtereket egymástól elválasztó kettőspont használata. Ez a kódba történő kisebb belenyúlással a megfelelő helyeken kiküszöbölhető. De természetesen nem csak ez a gond vele. Csekély sebességcsökkenés is jár a dologgal, és a sok hátrány mellett többek szemében egyre inkább eltörpülnek a névterek által nyújtott előnyök. Mindenesetre most sem tartják valószínűtlennek, hogy a névtértámogatás bekerüljön az új kiadásba, de ehhez egyelőre hiányzik a vállalkozó szellem: nincs aki választ tudna adni minden egyes felmerülő nehézségre. Reménykedjünk… Egy másik változás a MySQL-támogatásban lesz. A PHP és a MySQL együttműködésének nagy hagyományai vannak. Egyes híresztelésekkel szemben az igazság az, hogy ez így is fog maradni. Mindössze annyi módosul, hogy a PHP a MySQL-támogatást illetően egy régebbi állapotba kerül vissza, azaz mindenképp külső *mysql* függvénykönyvtárra lesz szükség. Erre a MySQL és a PHP felhasználási szerződésbeli különbségei miatt került sor. Nagy nehézséget a dolog nem jelent. Egyedül annyi pluszmunkával fog járni, hogy fordításkor a `-with-mysql=<mysql_könyvtár_elérhetősége>` kapcsoló beállítására is szükség lesz.

Változások az objektumkezelésben

Az PHP 5-ös változatának megjelenésével nagy lépést tesz előre az objektumtámogatásban is. Jelenleg elég kevés és tökéletlen megvalósítás áll csak rendelkezésre, ami azért így is sok mindenre elég. A jelenlegi kódoknak az új értelmezővel is lehetőleg ugyanolyan jól kell futniuk, így az sem mindegy, hogy milyen módon változnak meg a dolgok.

Az egyik sokak által és régóta várt módosítás az objektum értékként való átadásának megvalósításában történt. A korábbi (jelenlegi) változatokban ugyanis a függvényértékként átadott objektumból egy teljes másolat készült, amely önálló életet élt, majd a függvénnyel együtt megszűnt létezni. Több okból sem jó ez így: egyrészt a függvény által kezelt objektum változásai nem hatnak vissza az eredetire (márpedig ez lenne az üdvözítő), valamint felesleges processzor- és memóriaterhelést is jelent ez egyben. Ez a gond eleddig is megoldható volt, ha magunk gondoskodtunk a referenciahivatkozás megfelelő helyekre történő biggyesztésével, de a nagy többségre elmondható, hogy fogalma sem volt a kérdésről (csak szép csöndben több memóriát evett a PHP, mint kellett volna).

A PHP ötös változata már másképp fog bánni az objektum típusú értékekkel, ugyanis önműködően referenciaként adja át őket. Hasonlóan fontos változás történik az objektumok létrehozó, illetve megszüntető (constructor/destructor) függvényeivel kapcsolatban. Létrehozó most is létezik a C++ nyelvben is ismert formában. Az osztály nevével azonos elnevezésű függvény hivatott az új objektum születésekor az előkészületeket megtenni, a változóknak alapértelmezett értéket adni. Viszont nem létezik destruktor, pedig az is hasznos eszköz lehet a tarsolyunkban. Mielőtt egy objektum megsemmisül, esetleg érdemes lehet még egy-két rutinfeladatot végrehajtatni, ilyen például a naplózás, esetleg a hibakeresésben segítő adatok kiíratása. Erre jelenleg nincs lehetőség. A C++-féle `~Osztálynév()` függvény létrehozása sem segít. A fenti feladatokat ugyan egy `register_shutdown_function()` függvényhívás segítségével is elvégezhetjük, de ekkor a záró műveletsor semmiféle közvetlen kapcsolatban nem lesz objektumunkkal.

Mit hoz a jövő? Egy új létrehozó-, illetve megszüntető-megadási lehetőséget. Az 5-ös változattól felfele a `__construct()` és a `__destruct()` tagfüggvények fognak gondoskodni az induláskori és a megsemmisülés pillanatában elvégzendő feladatok biztosításáról. Értelemszerűen a `__construct()` tagfüggvényben tudjuk megadni az éledéskori alapállapotot, a `__destruct()`-ban pedig az objektum megszűnésekori feladatokat tudjuk meghatározni.

Miért jobb ez az elnevezésmód? Tegyük fel, hogy egyre növekvő munkánkban meg szeretnénk változtatni az egyik osztályunk nevét. Ekkor hibalehetőséget jelenthet az, ha elfelejtjük a létrehozót is átnevezni, követve a névváltozást. Másrészt előfordulhat olyan eset is, hogy leszármaztatott osztályunk egyik tagfüggvényéből akarjuk meghívni a szülőobjektum létrehozóját. Ez a jelenlegi felállás szerint csak akkor oldható meg, ha a szülőobjektum neve ismert. Ez viszont nem feltétlenül egyértelmű. Az új elnevezés használatával viszont minden esetben meg tudjuk célozni a szülő létrehozóját. Természetesen nem kell megijedni, a PHP5 telepítése után nem kell a létrehozó függvényeket az összes osztályunk forrásaiban azonnal átnevezni. A visszafelé való csereszabatosság szem előtt tartásával a „régi", osztálynévvel megegyező létrehozó kerül (létezése esetén) végrehajtásra, ha a PHP nem lel a `__construct()` tagfüggvény nyomára.

Újdonságok az objektumkezelésben

- A `__clone()`
  Azzal a lépéssel, hogy az 5-ös változattól kezdve – ha tetszik, ha nem – másolat helyett hivatkozás adódik át a függvények számára értékként, elesünk attól a néha azért szükséges lehetőségtől, hogy mégiscsak egy teljes, az eredetivel azonos, de innentől önálló életet élő objektumot hozzunk létre. A feladat megoldására szolgál a minden objektummal együtt járó tartozék, a `__clone()` tagfüggvény. Példának okáért eztán függvényértéknek adott igény esetén nem a `$Dolly` objektumváltozót adjuk meg, hanem a `$Dolly->__clone()` által létrehozott másolatra való hivatkozást.
  No, azért ne elégedjünk meg ennyivel! A `__clone()` függvény az osztályban ugyanis egy saját, hasonnevű függvénynyel felülírható, ami átveszi a szerepét. Így teljes vezérlést nyerhetünk afelett, hogy mi és milyen módon kerüljön átadásra a klónozás során. A dolog hogyanja: a függvényen belül két objektumhivatkozás is rendelkezésre fog állni ehhez. Egyrészt a megszokott `$this`, amely már a klónozás termékére fog mutatni, míg a másolás forrását a `$that` objektumváltozóban kapjuk meg. Innentől ránk van bízva, mit és hogy teszünk át `$that`-ból `$this`-be, valamint miféle egyéb tevékenységeket folytatunk még eközben.
- A `__call()`
  Az új Zend-motor további újdonságokat is hoz. A `__call()` névre hallgató tagfüggvény segítségével minden olyan eljáráshívást elkaphatunk, ami mögött nem áll a névnek megfelelő tagfüggvény, esetleg csupán nem érhető el

4. lista A static.txt

```
<?php

class Math {
    static public $PI = 3.1415926536;
    static public abs($szam) {
        if ($szam >= 0) {
            return $szam;
        } else {
            return 0 - $szam;
        }
    }
}

$pi_erteke = Math::$PI;
$abszolut_ertek = Math::abs($valamennyi);

?>
```

5. lista Az abstract.txt

```
<?php

abstract class Alakzat {
    abstract function rajzol();
}

class Haromszog extends Alakzat {
    function rajzol() {
        // rajzolunk egy haromszoget
    }
}

class Kor extends Alakzat {
    function rajzol() {
        // rajzolunk egy kort
    }
}

?>
```

kívülről. Amennyiben létrehozunk egy ilyen `__call()` névre hallgató tagfüggvényt, az minden olyan esetben működésbe lép, ha a meghívott eljárás nem létezik vagy nem hozzáférhető. A függvénynek két értéket kell befogadnia: az elsőben érkezik a próbára tett eljárás neve, a másodikban pedig tömb formájában a függvényértékként megadott adatok futnak be. A dolog több mindenre is jó. Egyrészt így magunk határozhatjuk meg, hogy mi történjen olyankor, ha valaki nem létező függvényt próbál meghívni. Ennél viszont sokkal hasznosabb alkalmazási lehetősége is létezik ennek az eszköznek.

Ha nem is olyan egyszerű és önműködő formában, de létrehozhatjuk saját túlterhelt függvényeinket. Mit is jelent ez? A túlterhelt függvénynevek olyan nevek, amelyek mögött több megvalósítás is áll, és annak függvényében hívódik meg egyik vagy másik függvény, hogy épp milyen típusú és számosságú értéksort kapott. C++-ban, mivel az fordított és erősen típusos nyelv, a dolog viszonylag egyszerűen kivitelezhető. Ott egy adott függvényt nemcsak a neve, de a paraméterezhetőségének a miképtje is azonosít, így nem gond két különböző függvényt azonos néven létrehozni. A PHP viszont egy fordítás közben futó parancsnyelv, így a C++-féle módszer megvalósíthatatlan. A `__call()` függvényt mankónak használva azonban a dolog mégiscsak megoldható osztályokon belül, hiszen abból – az értékek típusának és számának megvizsgálása után – tetszőleges függvény meghívható. Szemléltetésül egy példa, ahol a hosszú nevű osztályunk `tukroz()` tagfüggvénye a valóságban nem létezik, azt két másik tagfüggvény helyettesíti: a `tukroz_szamot()` és a `tukroz_szoveget()` saját, belső (private) függvény (hogy jelen esetben mit jelent a belső szó, arról egy kicsit később lesz szó). A két függvény közül a megfelelő meghívásáról – a kapott érték megvizsgálása után – a `__call()` gondoskodik.

- A `__get()`, `__set()`

  Az objektumoknak a tagfüggvények mellett léteznek egyéb sajátjai is, például a saját változói. És léteznek bizony változók, értékek, amelyekkel az adott objektum nem rendelkezik. Az ilyenekhez való sikertelen hozzáférések esetére vethetjük be a `__get()` és a `__set()` függvényt. Az előbbi a nem létező változók olvasásakor, a másik pedig ezek írásakor lép működésbe, csakis akkor, ha létrehozunk ilyen függvényeket. A dolog tehát kísértetiesen hasonlít a `__call()` esetére. A `__set()` meghívásakor két értéket is fogadnunk kell. Az elsőben a nem talált vagy nem nyilvános változó nevét kapjuk meg, amibe adatot próbáltak kívülről adagolni. A másodikban pedig a beállítani kívánt értéket. Hogy mit kezdünk ezekkel az adatokkal, az már képzelőerő kérdése. A `__get()` esetén egyetlen fogadnivaló értékünk van: az olvasni próbált változó neve.

Private, Protected, Public

A fenti szavak sem csengenek ismeretlenül a más nyelvektől érkezők számára. Ilyen előtagokkal ruházhatunk fel minden osztályon belüli változó- és függvénymeghatározást. A legutóbbi, a `public` ezek közül a PHP számára eddig is követett módszert jelenti, azaz jelenleg minden osztályon belül létrehozott tagfüggvény és változó elérhető kívülről. A PHP5 használatba vételétől kezdve az osztályon belüli elemek hozzáférését is finomszabályozhatjuk. A `private` előtag használatával (ezt mind változómegadás, mind függvénymeghatározás elé odabiggyeszthetjük) levédhetünk bizonyos elemeket a külső meghívástól, olvasástól, illetve módosítástól. Az osztályon belüli tagfüggvények ettől függetlenül továbbra is hozzáférnek mindenhez. A belső (private) függvények használatára volt példa a többértelmű, túlterhelt tagfüggvény megvalósítását bemutató listában, ahol minden, az értékektől függő megvalósításért felelő függvény védett a külső, közvetlen meghívástól. A teljes külvilágtól való elzárás és a teljes nyilvánosság között akad még egy átmeneti lehetőség. Előfordulhat, hogy bár a közvetlen külső meghívás nem kívánatos, az adott osztályt kibővítő gyermekosztályokból erre mégiscsak szükség lehet. Az ilyen hozzáférési forma a védett, azaz `protected` típusú elérési mód. Ha ilyen előtagot biggyesztünk egy érték vagy függvény meghatározása elé, az az osztályból származtatott összes gyermekosztályból (de csakis onnan) ugyanolyan jól hozzáférhető lesz.

Static

Megeshet, hogy egy osztály olyan elemekkel is rendelkezik – legyen az függvény vagy éppen egy változó – amelyhez

6. lista Az exception.txt

```
<?php

try {
    if (!mail($cim, $tema, $szoveg)) {
        throw new Exception('A levél
elküldése sikertelen');
    }
} catch (Exception $e) {
    echo '<b>Sulyos kivétel:</b>' . $e-
>getMessage() . "\n";
    echo 'A hibás sor: ' . $e->getFile() .
', ' $e->getLine() . ". sor\n";
}

?>
```

anélkül hozzá lehet férni vagy esetlegesen meg lehet hívni, hogy az adott osztályt egy objektumban példányosítanánk. Ez olyankor lehetséges, amikor például egy adott függvény kimenetére az adott objektum pillanatnyi állapota egyáltalán nincs hatással. Szemléltetésül tekintsük meg a *4. listát*.

Objektumhivatkozások önműködő feloldása (dereferencing)

Objektumaink éppen más objektumokat is tartalmazhatnak, és függvényértékként adhatják vissza őket. Ilyenkor a jelen lehetőségek között egy objektumtól visszakapott objektum egy tagfüggvényét meghívni két lépésben megy csupán. A PHP5 ebben is hoz némi változást. Az eddig csak ilyenformán működő programsorok

```
$csap = $Leny->csap_noveszt();
$csap->kapaszkodik();
```

az ötös változattól kezdve ezzel a sorral lesznek helyettesíthetők:

```
$Leny->csap_noveszt()->kapaszkodik();
```

Absztrakt osztályok

A legtöbb osztály, amelyből további osztályokat származtatunk, önmagában is hasznos, példányosításra méltó típus. Létezik más nyelvekben, azonban a bázisosztálynak egy olyan fajtája, ami önmagában semmire sem jó, kizárólag más osztályok közös alapjaként szolgál, egyben megfogalmazva azok közös tulajdonságait. Az ilyen önállótlan bázisosztályokat absztrakt osztályoknak is szokás nevezni.
Egy osztályt meghatározásakor az `abstract` előtag használatával tudjuk megfosztani az önálló lét lehetőségétől. Az ilyen osztályok példányosítási kísérletei rendre kudarcot fognak vallani, csakis az ezekből származtatott alosztályok számára lehetséges az önálló lét. A teljes osztályon belül lehetnek helyben kifejtett tagfüggvények, de ezek külön is rendelkezhetnek az `abstract` előtaggal, ami annyit jelent, hogy az adott függvény kifejtése csak a származtatott osztályban fog megtörténni (de ott kötelező jelleggel, hiánya pedig hibaüzenethez vezet). Lássunk egy példát egy absztrakt osztályra és két bővítményére (*5. lista*)!

Közös felület

A PHP4-nek az a korlátja, hogy egyszeres öröklődésre nyílik csak lehetőség, a továbbiakban is fennmarad. Ez annyit jelent, hogy egy bázisosztályt kiegészítő gyermekosztály nem szolgálhat további gyermek bázisaként. Ily módon az absztrakciónak is megvannak a maga korlátai. A gond enyhítésére szolgál a közös felület (interface) létrehozásának a lehetősége. Egy ilyen felület egy olyan absztrakt osztálynak felel meg, amelynek minden tagfüggvénye is absztrakt formában (helyben nem kifejtve) létezik. A kulcsszavak, amikre a felületekkel kapcsolatban szükségünk van, az `interface` és az `implements`. Ezek rendre a hagyományos osztályok `class` és `extends` szavainak felelnek meg. A felületek nagy előnye, hogy a megvalósítás (implements) még nem számít származtatásnak, így a megvalósításra szánt osztályból még származtathatók gyermekek.

Több szülő közös gyermeke

Kellemes új lehetőség lesz az is, hogy egy származtatott osztályt több osztály közös gyermekeként is létrehozhatunk, vagy akár több felületet valósítunk meg egy osztályban.
A dolog trükkje mindössze annyi, hogy az `extends` és `implements` kulcsszavak után nem egy, hanem vesszővel elválasztva több bázisosztály/felület nevét adjuk meg.

Kivételkezelés

Általános nehézség az osztályokkal kapcsolatban, hogy a különféle osztálykönyvtárakat létrehozó programozók ugyan pontosan be tudnak határolni hibákat, de nem feltétlenül tudják, hogy mit kezdjenek vele. Ugyanígy az ilyen osztálykönyvtárakat felhasználó programozók már tudják, hogy melyik hibával mit kezdjenek, de nem tudják közvetlenül felderíteni őket. Sokszor szükség van rá tehát, hogy a hiba felderítésének és kezelésének helye (és szintje) ne okvetlenül legyen ugyanaz. Az ilyen hibák egységes kezelésére nagyszerű eszköz a kivételkezelés. És bizony, a kivételkezelés hármasa – a `try`, `throw`, `catch` csapat – is felüti a fejét a PHP következő nemzedékének kiadásában. Innentől pontosan meghatározhatjuk, hogy mit tartunk hibának, és hol és milyen módon óhajtjuk kezelni azt. Ráadásul azt az előnyét is élvezhetjük, hogy ilyen módon teljes mértékben elkülöníthető a „hasznos” kód és a hibák kezeléséért felelős kód.

A cikkhez tartozó listák megtalálhatóak az 51. CD Magazin/PHP könyvtárában.

Heiglig (Cece) Szabolcs (cece@phphost.hu)
Veszprémben él. Hobbija, kedvenc időtöltése és munkája is a programozás. Szabadidejében hajlamos kerékpárra pattanni, vagy baráti társaságban szerepjátékokkal foglalkozni.

KAPCSOLÓDÓ CÍMEK

A legfrissebb fejlesztői PHP5 a
➲ http://snaps.php.net címről gyűjthető be.
A Zend-motorról és változásairól több cikk és GYIK is szól
➲ http://zend.com
A PHP5 új fejlesztéseinek folyamatos és teljes leírását megkísérlő cikk a ➲ http://phpvolcano.com/articles/php5 címről tölthető le.