

## Miképp növelhetjük PHP-programjaink sebességét?

Bruno rámutat néhány problémára, amelyektől PHP-programjaink lelassulhatnak, és egyúttal megoldást is közöl az elkerülésükre.

Számos oka lehet PHP-programjaink lelassulásának. Dugót okozhat az adatbázis-hozzáférés sebessége, a weboldalak elérése, vagy akár egy rosszul megtervezett, lassú algoritmus. Ha a sebesség csökken, a felhasználó feszültté válik, míg a kérésre várakozik. Kevés felhasználó kisebb hasznot hoz, és a webhelyed elveszíti népszerűségét. Legfőképpen a rossz tervezés és fejlesztés okolható a lassúsáért. Gyakran hoznak létre és indítanak el honlapokat mindenféle teljesítménypróbák nélkül. Az adatbázisok sokszor kevesebb adat kiszolgálására képesek, mint amennyit várnak tőlük. Az algoritmusok pedig számtalanszor rosszul tervezettek, és sebesség tekintetében sem hatékonyak.

Ha a webhely felgyorsításához nem áll módodban újratervezni az egészet, a teljesítmény fenntartása érdekében statikus oldalak kiszolgálására lesz szükséged, így a PHP-kódot nem kell minden egyes letöltéskor újrafordítani. Nézzük meg, milyen lehetőségeink vannak ennek a kivitelezésére!

### A hagyományos megközelítés

Az első dolog, amit tehetsz, hogy megvizsgálod, mely programokra vagy programrészekre kell a legtöbbet várakozni. Ezt a legegyszerűbben a PHP-héj (shell) segítségével oldhatod meg. Tegyük fel, hogy létezik egy *index.php* nevű parancsfájlod, és ennek a kimenetét szeretnéd statikussá tenni. Feltéve, hogy a PHP-héj neve *phpsh*, a parancssor a következőképpen néz ki:

```
phpsh --q /egy/k nyvtar/index.php >
  ↳ /egy/k nyvtar/index.html
```

Az *index.html* fájl most egy teljesen egyszerű HTML-fájl, amely semmiféle PHP-feldolgozást nem igényel, és az ügyfél változatlan formában töltheti le. Vajon mi történik olyankor, amikor a PHP-program kimenete időről-időre változik? Minden egyes alkalommal, amikor a kimenet megváltozik, újra el kell készítenünk egy új, statikus változatot.

### Rendszeres újrafeldolgozás

Linux alatt az időnkénti futtatás legegyszerűbb módja a *crontab*. A következő *crontab*-bejegyzés a megadott parancsot minden negyedórában lefuttatja:

```
*/15 * * * * root phpsh q
  ↳ /egy/konyvtar/index.php >
  ↳ /egy/konyvtar/index.html
```

Előfordulhat, hogy az adat frissentartásához ez az időzítés nem megfelelő. Léteznek olyan parancsfájlok, amelyeket nagyon ritkán kérnek le, azonban akadnak olyanok is, amelyeket folyamatosan próbálnak elérni – ebben az esetben ez a módszer értelmetlen, helyette más parancsfájlalapú lehetőség szükséges.

### Előfeldolgozás csak szükség esetén

Ez a módszer a parancsfájlokat bizonyos időközönként újrafeldolgozza, de csak akkor, ha valamilyen kérés érkezik rá. Ez

a működés nagyon hasonló egy webproxy működéséhez. Két fogást mutatok be e módszer megvalósítására: a kimenet átmeneti tárazását és az időn túli feldolgozást.

A kimenet átmeneti tárazása esetén a parancsfájl ellenőrzi a tárolt fájl dátumát és idejét, és csak szükség esetén dolgozza fel őket. A feldolgozás úgy működik, hogy mielőtt elküldené az ügyfélnek, a parancsfájl kimenetét egy gyorsfájlban tárolja. PHP-ben ezt az *ob\_start()* függvény segítségével lehet elvégezni. Ez a függvény bekapcsolja a kimenet átmeneti tárazását, és elküldi egy visszahívó függvénynek. Létezik egy másik függvény is, mellyel a kimenetet elküldhetjük a böngészőnek: *ob\_end\_flush()*. Vessünk egy pillantást a példánkra:

```
<?php
// a fejlöcfajl csatolása
include("header.php");
?>
<?php
// 10 másodperc pihenı
sleep(10);
?>
Test:
<?php
// A lábölöc csatolása
include("footer.php");
?>
```

A *header.php* fájlban található minden gyorsfájl tárazással kapcsolatos szolgáltatás. Ez először ellenőrzi, hogy szükség van-e új gyorsfájl-változat tárolására a *needscache()* függvény segítségével. Ez az ellenőrzés történhet idő- vagy fájl módosítás alapján, vagy ahogyan szeretnéd. Írásunkban idő szerinti példát veszünk alapul.

Ha az előző (gyorsfájl tárazott) példány elavult, a parancsfájl elindít egy *ob\_ciklust*, és a kimenetét kiírja a fájlba. Ugyanakkor ha a fájl még aktuális, annak tartalmát egyszerűen elküldi a böngészőnek (1. lista, 30. CD. Magazin/PHP).

A *footer.php* fájl az *ob*-feldolgozást egyszerűen csak lezárja:

```
<?php
ob_end_flush();
?>
```

Működését kipróbálhatod, ha a parancsfájlt többször meghívod, míg a gyorsfájlban tárolt fájl el nem évül. Tapasztalni fogod, hogy az első futás alkalmával a parancsfájl tíz másodpercet vár (mivel a parancsfájl egy *sleep()* hívás következtében a jobb szemléletesség céljából várakozik), míg a többi futás alkalmával a parancsfájl azonnal visszatér. Mindenesetre ha a tárolt fájl elévül, mindenképpen várnod kell, hogy a parancsfájl újat hozzon létre.

Lássuk, hogyan tudod ezt megkerülni, és azt az látszatot kelteni, mintha a parancsfájlod mindig gyors lenne!

Időn túli feldolgozás esetén a parancsfájl ellenőrzi a fájl dátumát és idejét, de a feldolgozás mindig csak azután történik meg, hogy a böngészőt már kiszolgálta – megoldva ezzel a fájl elévülését járó feldolgozási nehézséget. Ebben az esetben a gyorsított fájl csak a parancsfájl lefutása után lép működésbe. PHP-ben ez úgy oldható meg, hogy „a parancsfájl befejezése” eseményhez a `register_shutdown_function()` híváson keresztül egy függvényt rendelünk hozzá. Ebben az esetben csak a `header.php` fájlban szükséges módosítani (2. lista). Egyszerűen csak hozzáadtam a `doaftercache()` függvényt, amely akkor hívódik meg, ha a parancsfájl futása befejeződött. Ez a függvény meghívja a parancsfájlt – ahogyan a böngésző is tenné –, és a tartalmát tárolja. Csak egyetlen esetben várható a böngészőből, mégpedig ha a parancsfájl korábban még sosem futott. Próbáld ki, és az lesz az érzésed, hogy a program nagyon gyors!

## Összegzés

Bemutattuk, hogyan működik a PHP gyorsítótárási módszer, és a működését egy „csináld magad” példával szemléltettük. Ha a példákat kipróbáltad és tetszettek is, nyugodtan használd fel őket a saját programjaidban. Léteznek egyéb módok is a teljesítmény javítására, mint például a függvény-gyorsítás vagy a PHP-előfordítás. E feladatok megvalósítására a PHP további eszközöket kínál. Keresd meg a legjobb megoldást, és próbáld ki, hogyan állja a vizontagságokat!

*Bruno Pedro*

rendszerfejlesztő, tízévnnyi tapasztalattal rendelkezik az adatbázis-alkalmazások terén, ezenkívül társalapítója és vezetője az ethernet Ida projektnek.

2. lista Tetszőleges eljárás azonosítása a parancsfájl kilépési eseményével

```
<?php
// a gyorsítór elörösi átvonala
$CACHE_PATH="/tmp";

// a gyorsítór időtállópőse másodpercekben
$CACHE_TIMEOUT=10;

// a parancsfájl elöröse
$SCRIPT_URL="http://".$HTTP_HOST.$PHP_SELF;

// gyorsítór fájl létrehozása a parancsfájl
// névvel és a gyorsítór átvonallal
function cachefilename() {
global $PHP_SELF,$CACHE_PATH;

return($CACHE_PATH."/".md5($PHP_SELF) .
".cache");
}

// megvizsgáljuk, hogy a parancsfájlt
// szükséges-e gyorsítani
function needscache($timeout) {
clearstatcache();
if (time() -
filemtime(cachefilename())>$timeout)
return(true);
else
return(false);
}

// beolvassuk a gyorsítórát és kikérjük
// a böngészőnek
function outputcache() {
readfile(cachefilename());
}

// gyorsítór az a parancsfájlt
function docache($buffer) {

// kérjük az eredményt a
// gyorsítórba
$fp=fopen(cachefilename(),"w");

if ($fp)
fputs($fp,$buffer);

// adjuk a kimenetet
// a böngészőnek
return($buffer);
}

// gyorsítór az a eredményt
function doaftercache() {
global $SCRIPT_URL;

// beolvassuk a parancsfájl kimenetét
$fp=fopen($SCRIPT_URL,"r");
while (!feof($fp))
$buffer.=fgets($fp,4096);

// kérjük a parancsfájl kimenetét
// a gyorsítórba
$fp=fopen(cachefilename(),"w");
if ($fp)
fputs($fp,$buffer);
fclose($fp);
}

if (needscache($CACHE_TIMEOUT)) {
// a parancsfájlnak szükséges
// a gyorsítór
if (file_exists(cachefilename())) {

register_shutdown_function("doaftercache");
outputcache();
exit();
} else
ob_start("docache");
} else {
// a parancsfájl gyorsítórban van,
// olvassuk ki
outputcache();
exit();
}
?>
```