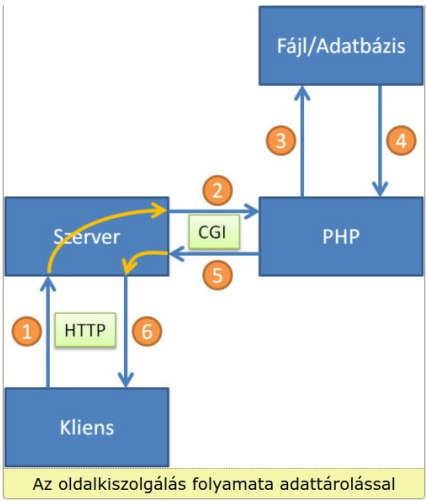
Adattárolás

Az oldalaink attól válnak dinamikussá, hogy adatokat jelenítenek meg HTML formában, és ezek az adatok változhatnak.

Az oldalkiszolgálás folyamatában a PHP szkriptek csupán a HTTP kérés kiszolgálásának idejére futnak. Ennek az a következménye, hogy minden bennük létrehozott változó a HTTP kérés végén megszűnik. A következő kérés kiszolgálásakor a szkript újra lefut, és benne újra létrejönnek a megfelelő változók, majd meg is szűnnek. Erre mutat példát az alábbi PHP szkript, ahol egy számláló értékét szeretnénk kérésenként növelni. Mivel a `$szamlalo` változó minden kérés végén megszűnik, így a következő kérésnél nem létezik, így értéke mindig 0-ról indul.

```php
<?php
if (isset($szamlalo))
{
	$szamlalo += 1;
}
else
{
	$szamlalo = 0;
}
var_dump($szamlalo);
?>
```

Két kérés kiszolgálása tehát az adatokat, a változókat illetően teljesen független. Ha valamilyen változó adatot szerver oldalon szeretnénk több kérés esetén is elérhetővé tenni, akkor azt az adatot nem szabad PHP változóban tárolni, hanem valamilyen módon a PHP állományon kívül kell elhelyezni. Kérés kiszolgálásakor a PHP szkript ebből a külső forrásból olvashatja be az adatokat, és még a szkript befejezése előtt ide is kell kimentenie.



Az oldalkiszolgálás folyamata adattárolással

Bármelyik módszert is választjuk minden esetben az a cél, hogy a külső forrásban valamilyen struktúrában tárolt adatot a feldolgozó függvény számára elfogadható formátumban előállítsuk. Az adatok tárolása tehát tulajdonképpen csak és kizárólag a beolvasás és kiírás folyamatába tartozik, nem szabad a feldolgozási logikában ilyen jellegű műveleteknek megjelennie.

Fájlban tárolás előnyei:

- bármikor elérhető, nincs szükség plusz szoftverekre;
- könnyen értelmezhető emberek vagy más programok számára;
- könnyen használható;
- viszonylag kis méretű.

A PHP sokféle nyelvi lehetőséget biztosít a szerveren elérhető fájlok kezelésére. Az általános, de viszonylag alacsony szintű műveletek mellett magasabb szintű függvények segítenek a tipikus feladatok egyszerű elvégzésében. Röviden tekintsük át a fontosabb függvényeket! A fájlkezeléssel kapcsolatos függvények teljes listáját és részletes leírását a dokumentáció tartalmazza.

Alacsony szintű, általános függvények:

```
$f = fopen($fájlnév, $mód): a fopen parancs az első paraméterében megadott
fájl vagy URL megnyitására szolgál. Sikeres megnyitás esetén a visszatérési
értékként megadott logikai fájlkezelővel lehet a további műveletekben a
fájlra hivatkozni. A $mód paraméter a megnyitás módját határozza meg:

r: csak olvasás
r+: olvasás, írás
w: csak írás, üres fájlt hoz létre
w+: olvasás, írás, üres fájlt hoz létre
a: hozzáfűzés
a+: hozzáfűzés, olvasás
x: csak írás, üres fájlt hoz létre, viszont ha létezik a fájl, akkor hiba
x+: olvasás, írás, üres fájlt hoz létre, viszont ha létezik, akkor hiba
c: csak írás, ha létezik a fájl, akkor elejére áll
c+: írás, olvasás, ha létezik a fájl, akkor elejére áll

fclose($f): a fájl bezárása.
feof($f): megadja, hogy elértük-e a fájl végét.
```

Alacsony szintű beolvasó utasítások:

```
$s = fread($f, $hossz): $hossz byte beolvasása.
$s = fscanf($f, $formátum): formátum szerinti beolvasás.
$s = fgets($f[, $hossz]): sor beolvasása a sor végéig vagy opcionálisan
adott hosszig.
```

$tömb = fgetcsv($f, $hossz[, $elválasztó]): egy sorban CSV formátumban tárolt elemek beolvasása.

Alacsony szintű kiíró utasítások:

fwrite($f, $s): $s kiírása.
fputs($f, $s): ugyanaz, mint az fwrite().
fputcsv($f, $tömb[, $elválasztó]): tömb elemeinek kiírása CSV formátumban.

* Magas szintű beolvasó utasítások:

$tömb = file($fájlnév[, módosítók]): a megadott fájl tartalmát soronként beolvassa egy tömbbe. FILE_SKIP_EMPTY_LINES tulajdonsággal a függvény figyelmen kívül hagyja az üres sorokat, így azok nem töltődnek be a tömbbe, a FILE_IGNORE_NEW_LINES utasítással elhagyásra kerülnek a sorvégi \n karakterek.

$s = file_get_contents($fájlnév): az egész fájlt szövegként beolvassa.

readfile($fájlnév): a fájl tartalmát beolvassa és automatikusan a kimenetre írja. Hasznos pl. képállományok kiírásánál.

fpassthru($f): a már megnyitott és valameddig feldolgozott fájl maradék részét beolvassa és a kimenetre írja.

Magas szintű kiíró utasítások:

file_put_contents($fájlnév, $s): a megadott szöveget a fájlba írja.

Néhány fontos fájlkezelő függvény:

mkdir($útvonal): könyvtár létrehozása
rmdir($könyvtárnév): könyvtár törlése
copy($forrás, $cél): másolás
rename($mit, $mire): átnevezés, mozgatás
unlink($fájlnév): törlés
is_dir($fájlnév): könyvtár-e
is_file($fájlnév): fájl-e
is_readable($fájlnév): olvasható-e
is_writable($fájlnév): írható-e
is_link($fájlnév): link-e
is_executable($fájlnév): futtatható-e
basename($útvonal): fájlnevet adja vissza
chown($fájl, $user): fájl tulajdonosának beállítása
chmod($fájl, $mód): fájl jogainak beállítása
chgrp($fájl, $group): fájl csoportjának beállítása
stat($fájl): fájl adatai tömbben
fseek($f, $offset): fájlmutató mozgatása

Hibakezelés

A fájlműveletek hiba esetén hamis értékkel térnek vissza. Így például annak ellenőrzése, hogy sikeresen megnyílt-e a fájl, a következőképpen történhet:

```
<?php
$f = @fopen('nem_letezik.txt', 'r');
if ($f)
{
      /* ... */
      fclose($f);
}
?>
```

Bár egy nem létező állomány esetén a vezérlés nem lép bele az elágazásba, a PHP bizonyos beállítások mellett egy figyelmeztető üzenetet ír ki a képernyőre jelezvén a fájl hiányát. Ha ilyen beállítások mellett szeretnénk elkerülni ezt a hibaüzenetet, akkor erre a @ operátor ad lehetőséget. A fenti kódrészletben a fopen által generált hibaüzenet nem kerül kiírásra, ha a @ jelet elé írjuk:

Adatszerkezet-vezérelt tárolás

Ahogy a bevezetőben írtuk a fájlkezelésre most olyan szemmel nézünk, mint a feldolgozás szempontjából fontos adatszerkezet tárolásának egyik lehetséges formájára. Ez azt jelenti, hogy ha szkriptünkben adott egy adatszerkezet, akkor azt egy megfelelő utasítással szeretnénk kimenteni egy fájlba, és következő alkalommal ebből a fájlból ugyanezt a struktúrát visszaolvasni. Ebben a folyamatban a fájl tartalmának felépítése nem érdekes. Csak az érdekel minket, hogy az adat a megfelelő formátumban rendelkezésre álljon.

Célszerű lenne tehát a PHP-s adatszerkezetet valahogy egy az egyben fájlba írni. Ebben segít nekünk az ún. **sorosítás**, ami egy adatszerkezet visszaalakítható szöveges megfelelője. A sorosítást elsősorban tároláskor és továbbításkor szokták alkalmazni.

Egy PHP-s adatszerkezet szöveges reprezentánsát többféleképpen előállíthatjuk. Használhatjuk a PHP sorosító függvényeit:

- $s = serialize($adat): a paraméterként megadott adat szöveges megfelelőjének előállítása
- $adat = unserialize($s): a szöveges megfelelő visszaalakítása adattá.

Például:

```php
<?php
$filmek = array(
	array(
		'cim' => 'Passió',
		'rendezo' => 'Mel Gibson',
		'ev' => '2004',
		'szereplok'=> array(
			'Jim Caviezel',
			'Maia Morgenstern',
			'Christo Jivkov',
		),
	),
	array(
		'cim' => 'Pio atya - A csodák embere',
		'rendezo' => 'Carlo Carlei',
		'ev' => '2000',
		'szereplok'=> array(
			'Sergio Castellitto',
			'Sergio Albelli',
		),
	),
);
?>
```

Az előállított formátum pedig:

```php
<?php
echo serialize($filmek);
//a:2:{i:0;a:4:{s:3:"cim";s:7:"Passió";s:7:"rendezo";s:10:"Mel
Gibson";s:2:"ev";s:4:"2004";s:9:"szereplok";a:3:{i:0;s:12:"Jim
Caviezel";i:1;s:16:"Maia Morgenstern";i:2;s:14:"Christo Jivkov";}}i:1;a:4:
{s:3:"cim";s:27:"Pio atya - A csodák embere";s:7:"rendezo";s:12:"Carlo
Carlei";s:2:"ev";s:4:"2000";s:9:"szereplok";a:2:{i:0;s:18:"Sergio
Castellitto";i:1;s:14:"Sergio Albelli";}}}
?>
```

Egy másik lehetőséget a szöveges megfelelő előállítására a JSON formátum adja, aminek kezelésére a PHP a következő két függvényt adja:

- `$s = json_encode($adat)`: az adat JSON szövegként való előállítása.
- `$adat = json_decode($s[, $asszoc])`: JSON szöveg adattá alakítása. Második paraméterével azt lehet beállítani, hogy a JavaScript objektumokat PHP objektumként vagy asszociatív tömbként reprezentálja.

A fenti példa esetében a json_encode eredménye a következő:

```
<?php
echo json_encode($filmek);
//[{"cim":"Passi\u00f3","rendezo":"Mel Gibson","ev":"2004","szereplok":
["Jim Caviezel","Maia Morgenstern","Christo Jivkov"]},{"cim":"Pio atya - A
csod\u00e1k embere","rendezo":"Carlo Carlei","ev":"2000","szereplok":
["Sergio Castellitto","Sergio Albelli"]}]


//Vagy formázva
/*
[
     {
          "cim":"Passi\u00f3",
          "rendezo":"Mel Gibson",
          "ev":"2004",
          "szereplok":[
               "Jim Caviezel",
               "Maia Morgenstern",
               "Christo Jivkov"
          ]
     },
     {
          "cim":"Pio atya - A csod\u00e1k embere",
          "rendezo":"Carlo Carlei",
          "ev":"2000",
          "szereplok":[
               "Sergio Castellitto",
               "Sergio Albelli"
          ]
     }
]
*/
?>
```

A JSON formátumnak többek között megvan az az előnye, hogy egyrészt nagyon elterjedt formátum az adatleírás és kommunikáció terén, másrészt formázott állapotában szövegesen is áttekinthető és akár szerkeszthető.
Feladatok

1. Adott filmcímek listája egy fájlban, soronként egy filmmel. Olvassuk ezt be egy tömbbe!

2. Készítsük el az előző példa párját, azaz oldjuk meg egy filmcímeket tartalmazó tömb fájlba mentését is (egy cím egy sor)!

3. Adott egy rekordokból álló tömb. Végezzük el a kiírását úgy, hogy egy sorban egy rekordnyi információ legyen, az egyes értékeket soron belül tabulátorral válasszuk el!

4. Az előző példában kapott fájlt olvassuk be rekordok tömbjeként!

5. Az előző feladatbeli rekordok tömbjét tároljuk úgy a fájlban, hogy a rekord minden egyes mezeje külön sorba kerüljön, és az egyes rekordokat üres sor válassza el egymástól. Oldjuk meg a tömb beolvasását is!