PHP chemotox

Papp Győző
papp.gyozo@phpconf.hu

Második Magyarországi PHP Konferencia
2004. március 27.

Elméleti bevezető

`--dry-run`

Mit takar a cím?

- chemotox = gyermekkorom „legaktívabb” rovarirtója
- PHP programjainkban is vannak „kártevők”
- A hiba definíciója (≡):
  - „a program ...
  - nem kívánt viselkedése, helytelen reakciója vagy a kívánttól eltérő kimenet produkálása
  - valamilyen bemenet és belső állapot kombinációjára.”

✗ Ez az elmélet (nehéz megfogni a lényegét, hasznosság?)

✔ Mit nyújt mégis mindez a gyakorlati ember számára?

„A program …”

- Néhány közhely bemelegítésnek:
  - „*Az a jó program, ami nincs is.*”
  - „*Nincs hibátlan program.*”
  - „*No feature, no bug.*” ≈ „*No woman, no cry*”
- Megint egy definíció (≡):
  - „ ... utasítások sorozata, amelyeket a számítógép értelmezni és végrehajtani képes.”
- I. posztulátum: A gépnek átadott utasításokat kell szemügyre vennünk, és nem koncepciókat felállítanunk!

„... nem kívánt viselkedése ...” [és a többiek]

- *nem kívánt ~>*
  - feltételezi valamilyen terv meglétét („*is it a feature or bug?*”)
  - a jó tervezés jelentősen csökkenti a „beépülő” hibák számát
  - tesztadatokból képzett teszthalmazok
- *viselkedés ~>*
  - tehát létezik és működik: `` `php -l parse.php´ ``
  - ez nem jelent mindig „szemmel közvetlenül látható” eltérést! (*kívánttól eltérő kimenet*)
- *helytelen reakció ~>* másodlagos jelekből következtetünk a rendellenességre (figyelünk-e ezekre?)

„… bemenet és belső állapot kombinációjára”

- mindig van valamiféle *bemenet,*
  még ha az nem a felhasználótól érkezik is
- *belső állapot:*
  - normál PHP változókon – **`get_defined_vars()`** – kívül:
  - **`$_SERVER`**, **`$_ENV`**
  - **`get_defined_constants()`**
  - **`get_defined_functions()`**
- **`$_COOKIE`**, **`$_SESSION`** – állapot vagy bemenet?
- II. posztulátum: A változókat együtt kell figyelni a program utasításaival, amelyek ezeket módosítják!

„Kell egy terv!”

– Mi?
– Minek?

A terv hasznosítható eredményei

[ismétlés: „*nem kívánt*” + „*bemenet*”]

✗ „létezhetetlen” (fizikailag nem fordulhat elő) és

✔ lehetséges <u>bemeneti értékek</u> megállapítása: (értéktartomány)

- ✔ érvényes ~> normál működés folytatása
- ✗ érvénytelen ~> működés megtagadása (újrakérés)

- teszthalmazok kialakítása, (PEAR PHPUnit):

✔ jó teszt-telefonszám: 435-22-43, 435-2w-43, 32-456

✗ rossz teszt-telefonszám: „dasgdsfs”, „bocimobil”

A terv során születő kritériumok

[ismétlés: „*belső állapot*”]

- A kikötött feltételezések ne a dokumentációt hízlalják!
- Építsük be azokat a programokba!
- megjegyzésként:
  - ✔ emlékeztetőnek kiváló, de
  - ✗ csak formális segítség
  - ✗ nem „jelzi”, ha az implementáció eltér a tervtől
- aktív kódként: `assert();`

assert()

```php
<?php
function norm_whitespace( $value, $type )
{
  assert ( 'is_scalar($value) || is_string($type)' );

  switch ( $type = strtolower( $type ) )
  {
   case 'preserve':
        return $value;
   case 'replace' :
        return preg_replace('/\s/', ' ', $value);
   case 'collapse':
        return trim(preg_replace('/\s/+', ' ', $value));
   default:
        assert ('$type != "$type" '); // mindig HAMIS
  }
}
```

assert() – hol, mikor?

- ✔ hibás belső állapotok felderítésére,
- ✔ érvénytelen belső(!) inicializálások elkerülésére,
- ✔ megbízható (modul-)paraméterek ellenőrzésére (interfészeken vagy belső modulfüggvényekben)

NEM szabad:

- ✗ felhasználói adatok érvényesítésére,
- ✗ mellékhatást kiváltó feltétellel vagy utasítással,
- ✗ rendeltetésszerű működés során fellépő hibákra, pl:

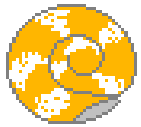
```
assert ( '$conn = mysql_connect ($server)' );
```

assert() – hogyan?

- PHP.ini-ből: kód: assert_options($option, $value);
- assert.active = ASSERT_ACTIVE
- assert.warning = ASSERT_WARNING
- assert.bail = ASSERT_BAIL
- assert.quiet_eval = ASSERT_QUIET_EVAL
- assert.callback = ASSERT_CALLBACK
- lásd: http://hu.php.net/assert-options
- assert() paraméter szintaktika: '… ', "…", … (natív kód)

www.phpconf.hu

„Csak egy kis pánik!”



„Előre láthatólag előfordulnak
előre nem látható események
előre nem látható időpontokban.”

A váratlan hibák

- ✔ érvényes adat ~> érdemi munka (programlogika)
- ✗ érvénytelen adat ~> hibajelzés a felhasználó felé
- ✗ érvénytelen állapot ~> hibakezelés + fejlesztők riasztása

- felhasználói szintű hibák:

```
if ($invalid) {
    header('Location: form.php'); exit;
}
```

- külső rendszerhibák és erőforrás–kiesések / leállások

```
if ( !$eroforras ) {
    readfile($tarolt_tartalom); exit;
} // ???
```

Hatékony hibakezelés ismérvei

- Felhasználói szintű hibák kezelésére:
  `if (!$siker ) { /*...*/ }` – nincs más ...
- Hatékony a rendszerhibák kezelése, ha:

✔ minden lehetséges hiba feldolgozásra kerül

✔ a teljesítmény nem romlik normál működés esetén

✔ a kód olvashatósága nem csökken
a kezelt hibalehetőségek számával

✔ az implementáció nem aránylik a hibalehetőségek számához (nincs állandó méretű többletkód: "if "-ezés)

Központosított hibakezelés

- PHP 4 saját hibakezelő függvény használata: **`set_error_handler()`**
- Atomi, ‘mindent vagy semmit’ kódrészek kiemelése és együttes hibakezelése, pl:
  - hibakezelő függvényben futás-megszakítás,
  - utasítás összevonás logikai operátorokkal (~shell)

```
((($f = fopen( BARKOCHBA, 'a' ))
  &&  fwrite( $f, $_POST['kerdes'] )
  &&  fclose( $f ));
```

- PHP 5 kivételkezelés = egyesíti az előző kettő előnyeit (erőforrás-igényes, de tisztább kód)

PHP rejtett tartalékai

- `handle ( $level, $msg, $file, $line, &$vars ) {`
  `// ...`
  `$lines = file($file, 1);`
  `print ( $lines[$line] );`
- `var_export ( $vars );`
- `debug_backtrace ();`
- `error_log ();`
- `set_error_handler( array(&$this,'handle'));`
- `get_defined_func();`
  `get_defined_constants();`
- `register_shutdown_function();`...

- minden futási hiba (`E_*`) lekezelhető (`@` nem számít!)
- hibás sor kiírása
- aktuális változók értékei
- függvényhívási lánc
- külön hibanaplózás (fájlba vagy e-mailen)
- objektummetódus hibakezelőként (4.3.0)

„Még egy kis pánik”


A bogarak támadása

„Ahogy mindig is szoktuk”

- `var_dump($mi_ez_magikus_szam);`
- `echo 'még futok';`

✔ a **leggyorsabb**, ha tudjuk, hol kell keresni a hibát

✔ elméletileg **minden telepítésnél** használható

✗ próbálkozások – „*trial and error*” ciklusok futtatása közbeavatkozás lehetősége nélkül

✗ átfogó vizsgálathoz sok helyre kell beszúrni (veszélyes lehet, ha elfelejtjük kiszedni őket)

✗ HTTP-fejléc küldés meghiúsítása (redirect, cookie-k)

Interaktív nyomkövetők előnyei

- ✔ kézben tartható a futtatás:
  - parancsról parancsra léptetés, blokkok átugrása vagy léptetése
  - feltételes futásfelfüggesztés,
  - töréspontok, stb.
- ✔ belső állapotváltozások lépésenkénti nyomonkövetése
- ✔ könnyen inicializálható bemenet (**`$_GET,$_POST,$argv`**)
- ✔ működés közben módosítható változókörnyezet
- ✔ hibaüzenetek és naplók a kimenettől elkülönülnek
- ✔ érintetlen kód – eredeti funkcionalitás (sehogy máshogy!)
- ✗ szerveren telepíteni kell (és kliens alkalmazás is kell)

www.weblabor.hu-n folytatjuk

Fejlesztőkörnyezetek izgalmas újdonságai:

- kódelemzés és optimalizálás (profiling)
- helyi (local), távoli (remote) …
- „incidens vezérelt” (just in time) …
- munkamenet alapú (per session) hibakeresés
- kódmódosítás futtatás közben – ez azért odébb lesz!

Előkészületek

- Előbb minden releváns hibát megszüntetni!
  – A részletes hibajelentés (napló) kincset ér
- webes környezet sajátosságai:
  - gyorsítótárak
  - kliens oldali kódok
  - bemeneti adatok visszavezetése a kezdetekhez
- PHP beállítások ellenőrzése és egyeztetése
- nyitott szellemiség, kritikus attitűd
- friss levegő, kóla, kávé, pizza ...

Oszd meg és uralkodj!

- Meghatározni:
  … a bemeneti adatok közül
  - a hibáért felelős paraméterek legszűkebb halmazát
  - ettől legkevésbé eltérő, de még jó eredményt adó tesztadatot
  - a kettő különbségét
  … a kódban
  - a „hibás” kimenetet előállító programsorokat
  - a különbségért felelős / azt okozó műveleteket
- igazolni a feltevésünket és javítani
- végül: bővíteni a teszthalmazt a hibát okozó adattal!

Összefoglalás

- Nincs semmi, ami felérne a tervezéssel!
- érvényes és hibás tesztadatokból teszthalmazok kialakítása
- `assert ();`
- `error_reporting = E_ALL; [php.ini]`
- `set_error_handler ();`
- `error_log ( ... ); // var_dump ($mi_ez);`
- **végül és utolsó sorban**: kell egy debugger (nem ezzel kell kezdeni!)

Bemutatott eszközök elérhetősége

A következő eszközöket érintettük ebben az előadásban :

- ErrorHandler PHP hibakezelő osztály:
  http://www.phpclasses.org/errorhandler
- xdebug:
  http://www.xdebug.org
- DBG:
  http://dd.cron.ru/dbg
- Zend Studio IDE (Personal Licence):
  http://www.zend.com/store/products/zend-studio.php
- NuSphere PhpED:
  http://www.nusphere.com

További hasznos eszközök

amelyek nem kerültek részletes bemutatásra:

- ErrorHandler PEAR-esített verziója:
  http://www.mojavelinux.com/forum/viewtopic.php?t=51
- NiceDebug:
  http://dev.izibox.isa-geek.org/NiceDebug/
- ActiveState Komodo IDE (Perl/PHP/Python):
  http://www.activestate.com/Komodo
- APD
  http://pear.php.net/package/apd

Idézett források – Hasznos irodalom

- A PHP Kézikönyv idevágó fejezetei:
  - Error Handling & Logging: http://hu.php.net/error_func
  - PHP Options &Informations: http://hu.php.net/info
  - Variables: http://hu.php.net/variables
- Derick Rethans hasznos tippjei: http://www.derickrethans.nl/errorhandling/talk.html
- PHP Builder – „Debugging PHP” (PHPUnit): http://www.phpbuilder.com/columns/oier20010406.php3
- SitePoint – „Effortless (or Better!) Bug Detection with ...” http://www.sitepoint.com/article/bug-detection-php-assertions
- minden bemutatott eszközhöz járó dokumentáció

Hasznos volt?