APLIKACJE INTERNETOWE
PHP

TEMAT 5-01: Obsługa błędów i debugowanie oraz wyjątki.

Autor dokumentu: Wojciech Galiński	środa, 7 września 2016 r.	351203 Technik informatyk

ŹRÓDŁA WIEDZY: http://www.php.net/manual/pl, http://webmaster.helion.pl/index.php/kurs-php, http://pl.wikipedia.org/, http://pl.wikibooks.org/wiki/PHP, http://phpkurs.pl/, http://kursphp.com/.

Zagadnienia obowiązkowe

1. **Debugowanie** – (źródło: Wikipedia) „(ang. debugging – odrobaczanie) proces systematycznego redukowania liczby błędów w oprogramowaniu bądź systemie mikroprocesorowym, który zazwyczaj polega na kontrolowanym wykonaniu programu pod nadzorem debuggera”.
2. **Etapy debugowania** – wyróżniamy:
   1) ***Reprodukcja błędu*** – spróbuj wywołać błąd w sposób kontrolowany.
   2) ***Wyizolowanie źródła błędu*** – wyeliminuj czynniki nie przyczyniające się do powstania błędu.
   3) ***Identyfikacja przyczyny awarii*** – wyświetl dodatkowe wartości kontrolne, aby wykryć miejsce występowania błędu.
   4) ***Usunięcie defektu*** – zmodyfikuj kod w taki sposób, żeby usunąć błąd.
   5) ***Weryfikacja powodzenia naprawy*** – sprawdź, czy po usunięciu defektu skrypt działa prawidłowo (nie generuje już tego błędu, albo nowych błędów).
3. **Narzędzia do debugowania w języku PHP** – wyróżniamy:
   - ➔ **ini_set('display_errors', 1);** – funkcja ta włącza wyświetlanie błędów (włączenie tej opcji może być kontrolowane w pliku konfiguracyjnym o nazwie „php.ini” na serwerze);
   - ➔ ***error_reporting(E_ALL | E_NOTICE | E_STRICT);*** – funkcja ta aktywuje wyświetlanie wszystkich błędów (opcję tą można włączać także w pliku konfiguracyjnym o nazwie „php.ini” na serwerze), np.
     ```
     $poprzedni = error_reporting(E_ALL ^ E_NOTICE);
     echo 'Teraz komunikaty Notice nie działają: '. $nieistniejacaZmienna;
     error_reporting($poprzedni);
     ```
   - ➔ ***echo*** – najprostsza metoda debugowania poprzez wyświetlenie wartości w przeglądarce, np.
     ```
     $a = 5;   echo '['. $a. ']<br />';
     ```
   - ➔ ***print_r*** – instrukcja „echo” wzbogacona o czytelne wyświetlanie zawartości tablic oraz obiektów, np.
     ```
     $tab = array(array(false, 1), array(2.0, '3'));
     echo '<pre>'; print_r($tab); echo '</pre>';
     ```
   - ➔ ***var_dump*** – instrukcja „print_r” wzbogacona o wyświetlanie typów zmiennych;
     PRZYKŁADY:
     ```
     echo '<pre>'; print_r($tab); echo '</pre>';
     $a = 1; $b = 2.0; $c = array('3', false); var_dump($a, $b, $c);
     ```
   - ➔ ***mysql_query($zapytanie) or die(mysq_error());*** – ułatwia lokalizację błędów w zapytaniach SQL;
   - ➔ ***assert*** – służy do alarmowania o niespodziewanych wynikach (błąd jest generowany w przypadku przekazania do funkcji wartości „false”), np.
     ```
     assert_options(ASSERT_ACTIVE, 1);   assert_options(ASSERT_WARNING, 1);
     $zmienna = 5;   assert($zmienna == 6);
     ```
   - ➔ ***debug_backtrace(), debug_print_backtrace()*** – funkcje te wyświetlają na różne sposoby, które funkcje były wywoływane i z jakimi parametrami, żeby dotrzeć do funkcji, w której znajduje się funkcja „debug_backtrace”.
     PRZYKŁAD:
     ```
     function a() { b(); }   function b() { c(); }
     function c() { echo '<pre>'; print_r(debug_backtrace()); echo '</pre>'; }
      a();
     ```
4. **Systemy debugowania dla języka PHP** – oto przykładowe systemy:
   - ➔ ***XDebug 2*** (bezpłatny);
   - ➔ ***Zend Studio*** (komercyjny).
5. **Informacje o bieżącym pliku i numerze wiersza dla informacji debuggera** – można wykorzystać do tego stałe magiczne.
   - ➔ **__LINE__** – numer linii w pliku;
   - ➔ **__FILE__** – nazwa bieżącego pliku;
   - ➔ **__DIR__** – nazwa bieżącego katalogu;
   - ➔ **__FUNCTION__** – nazwa bieżącej funkcji;
   - ➔ **__CLASS__** – nazwa bieżącej klasy;
   - ➔ **__METHOD__** – nazwa bieżącej metody.

6. **Fazy przetwarzania skryptu PHP** – wyróżniamy:
   - ➔ ***parsowanie skryptu*** – wyszukiwanie błędów uniemożliwiających wykonanie skryptu (Parse error);
   - ➔ ***wykonanie skryptu*** – i w rezultacie (przeważnie) wygenerowanie dokumentu HTML (w tej fazie mogą pojawić się błędy wykonania skryptu (w tym: wyjątki), ostrzeżenia oraz informacje).
7. **Ukrywanie błędów** – służy do tego instrukcja „@”. Zalecane jest dla końcowych wersji skryptów PHP, gdy nie chcemy, żeby klientowi wyświetlały się niezrozumiałe dla niego treści, w dodatku nie związane z tematyką witryny.

   Tylko kod znajdujący się ZA instrukcją „@” objęty jest ukrywaniem przed błędami.

   Ma on jednak kilka poważnych wad, m. in.:
   - ➔ ukrywa on błędy parsowania oraz wykonania, przez co utrudnia znalezienie przyczyny błędnego działania skryptu;
   - ➔ jego stosowanie znacznie obniża wydajność skryptu.

   Z powyższych powodów, stosujemy go tylko wtedy, gdy jest to naprawdę konieczne.

   PRZYKŁAD: `echo @$a; // ale tak jest lepiej: if (isset($a)) echo $a;`
8. **Wyjątki PHP** – są to specjalne obiekty klasy „Exception” do obsługi błędów w skryptach PHP. Wyjątki można obsługiwać albo generować (rzucać). Oto instrukcje do obsługi wyjątków:
   - ➔ ***blok „try”*** – blok instrukcji, które mogą wygenerować wyjątek, który chcemy obsłużyć lub wygenerować (bloki „try” można zagnieżdżać);
   - ➔ ***blok „catch”*** – fragment kodu wykonujący się automatycznie po wygenerowaniu wyjątku;
   - ➔ **instrukcja „throw”** – służy do generowania (rzucania) wyjątków w sytuacji, którą uznajemy za wyjątkową.

   PRZYKŁAD:

```
if (!isset($_GET['where']))   $_GET['where'] = 0;
try
{ if (!$_GET['where'])  throw new Exception('Błąd 0');
  echo 'Dalsza część bloku...<br/>';
  try
  { if($_GET['where'] == 1)  throw new Exception('Błąd 1');
    echo 'Dalsza część bloku podrzędnego...<br/>'; }
  catch(Exception $exception)
  { echo 'Problem: '.$exception->getMessage().'<br/>';   }
  echo 'Dalsza część skryptu...<br/>';
}
catch(Exception $exception)
{ echo 'Błąd krytyczny: '.$exception->getMessage().'<br/>'; }
```

   Klasa „Exception” może być dziedziczona, dzięki czemu możemy tworzyć własne klasy wyjątków.
9. **Testowanie skryptów** – polega na przygotowaniu zestawu testów, których prawidłowe rezultaty znamy. Po przygotowaniu zestawów testów, przekazujemy je skryptowi i sprawdzamy, czy zwrócił spodziewane wyniki.
10. **Systemy testowania skryptów PHP** – oto przykładowe systemy:
    - ➔ ***Selenium*** – testowanie funkcjonalne (testowanie funkcjonalności gotowej aplikacji);
    - ➔ ***PHPUnit*** – testowanie jednostkowe (testowanie pojedynczych funkcji języka PHP).

Zadania

1. Zapoznaj się z przedstawionymi przykładami. Wykonaj je i przeanalizuj ich działanie.
2. Dowiedz się więcej w Internecie na temat wyżej wymienionych systemów do debugowania skryptów PHP oraz systemów testowania skryptów PHP. A może uda Ci się znaleźć inne tego typu systemy?