Scripting: PHP-Performance-Tipps

Problem:

PHP ist zwar eine einfache und schnell zu erlehrnende Sprache. Aber sobald man größere Projekte angeht, hat man schnell das Problem das die Geschwindigkeit sinkt.

Hier will ich einige Vorschläge zur Beschleunigung der Compiletime (also das übersetzen vom PHP-Script in Binary-Code) und der Runtime (also optimiertere Laufzeit) machen:

Die Tipps im Einzelnen:
1. PHP-Code innerhalb von HTML

In PHP sieht man häufig solche Konstrukte:

```
‹a href="‹?php echo $_SERVER['PHP_SELF'] ?›"?command=action›Action‹/a›
```

Ein Verschachteln von HTML und PHP hat zwei wesentliche Nachteile:

1.) Man verliert deutlich die Übersicht.

2.) Die Zend-Engine braucht mehr Zeit zum Parsen.

Der deutlich bessere Weg ist PHP-Scripte wirklich wie Scripte zu behandeln und keine Mischung aus PHP-&HTML-Code zu zu lassen.

D.h. Ein PHP-Script beginnt in der ersten Zeile mit `‹?php` und endet in der letzten Zeile mit `?›`.

Alle HTML-Ausgaben erfolgen mit `echo`, `print` oder `printf`.

2. Strings

Strings sind beliebige Zeichenketten. Und doch gibt es wesentliche Unterschiede in der Notation. Man kann in PHP String mit einfachen (`'`) oder doppelten Hochkomma (`"`) schreiben.

Der Unterschied ist den meisten PHP-Programmierer sogar geläufig:

- Ein `'string
'` wird genau so übernommen.

- Ein `"string
"` hingegen erhält einen Zeilenumbruch.

- Ein `"string=$str"` löst sogar die Variable auf.

Aber warum nutzt man dann immer wieder `"string"` in doppelten Hochkommata, wenn man die Features des Parsings gar nicht nutzt?

Wenn man bedenkt, daß jeder String in `"` nochmals durch einen Parser läuft, so sollte man in Zukunft nur noch `'` einsetzen und den doppelten lediglich wenn man Zeilenumbrüche braucht.

3. Laufzeit

Zur Optimierung der Laufzeit möchte ich folgende zwei Beispiele untersuchen:

```
for ($j=0; $j‹sizeof($arr); $j++)
  echo $arr[$j]."‹br /›";
```

```
for ($j=0, $max = sizeof($arr), $s = ''; $j‹$max; $j++)
  $s .= $arr[$j].'‹br /›';
echo $s;
```

Grundsätzlich scheint das 2te Beispiel deutlich länger und vor allem umständlicher zu sein. Zuerst vergleichen wir den Schleifen-Konstrukt:

Im Ersten wird `sizeof($arr)` bei jedem Schleifendurchlauf aufgerufen. Dies macht i.d.R. nur Sinn, wenn sich das Array innerhalb der Schleife ändern könnte. Ansonsten ist es schneller die Größe des Arrays nur einmal zu bestimmen und dann eine feste Schleife zu fahren wie es im Zweiten der Fall ist.

Dies gilt natürlich für alle Schleifen-Arten und Funktionen. Ebenfalls für den Durchlauf eines Datenbank-Recordset's und ähnlichem.

Nun noch ein Wort zu `echo`: Jede Funktion die HTML-Code an den Apache und den Browser zurück liefern soll ist aufwendiger und kostet Zeit. Denn bei jeder Ausgabe wird ein Aufruf in die Apache-API gestartet und evtl. TCP/IP-Pakete gebaut, etc.

Schneller geht es, wenn man die Ausgabe eine Weile in einer Variable zwischen lagert und dann in einem Rutsch an den Apache weiterleitet.

Zu guter Letzt noch ein Blick auf die verschiedenen Schleifen-Konstrukte:

Wir Vergleichen eine `for`-Schleife mit `while`:

```
for ($i=0; $i‹$max; $i++) {
    ...
}

$i = 0;
while ($i++‹$max) {
    ...
}
```

Beide Schleifen liefern das selbe Ergebnis. Erstaunlicherweise ist die `while`-Schleife aber deutlich schneller.

(Eine `foreach`-Schleife will ich erst gar nicht in den Vergleich aufnehmen, da jedem bereits klar sein dürfte, daß dies die unperformanteste Art ist ein Array zu durchlaufen.)

4. Modularisierung

Modularisierung ist ein sehr guten Paradigma der modernen Softwareentwicklung. Aber bei Interpreter-Sprachen wie z.B. PHP benötigt jede Einbindung von weiteren Dateien (z.B. mit `include()`, `require()`, `require_once()`) Zugriff auf die Platte und damit Compiler-Zeit.

Ein `include()` ist sowieso etwas besonderes, da er erst zur Laufzeit ausgeführt wird. D.h. während des Script-Ablaufes kann/wird die Datei erst mit in das Script eingefügt. Es wird also nachträglich nochmals der Parser (Zend-Engine) angeschmissen um diese Datei zu verarbeiten. Eine absolut Zeit intensive Aktion.

Beschleunigung messen

Um die eigenen Optimierungen zu messen kann man einfach mit der Funktion `getmicrotime()` arbeiten:

```
‹?php
$start_time = getmicrotime();

// code
// ...

print (getmicrotime()-$start_time);
?›
```

Weiterführende Links:

- PHP Performance ist ein Blog mit regelmäßig erscheinenden Artikeln.

Scripting: PHP-Performance-Tipps

- A HOWTO on Optimizing PHP with tips and methodologies (bereits etwas älter)
- PHP: Strings / Zeichenketten

Eindeutige ID: #1239
huschi
2007-06-02 10:50