

# Einstieg in Profiling mit XHProf

In diesem kleinen Tutorial möchte ich zeigen, wie man das Profiling-Tool XHProf installiert und einsetzt, um Geschwindigkeitsbremsen im eigenen PHP-Code zu finden.

== Installation ==

Die Installation bezieht sich auf Ubuntu (UNIX). E sei angemerkt, dass es XHProf zurzeit nur für Linux/FreeBSD und MAC OS gibt.

Um nun XHProf zu installieren, geben wir folgendes in unsere Konsole ein:

## Quellcode

1. `cd /opt/`
2. `sudo wget http://pecl.php.net/get/xhprof-0.9.2.tgz`
3. `sudo tar xvf xhprof-0.9.2.tgz`
4. `cd xhprof-0.9.2/extension/`
5. `sudo phpize`
6. `sudo ./configure --with-php-config=/usr/bin/php-config`
7. `sudo make`
8. `sudo make install`
9. `sudo make test`

Als nächstes öffnen wir unsere PHP-Konfigurationsdatei:

## Quellcode

1. `sudo gedit /etc/php5/apache2/php.ini`

Folgende Zeilen werden wir nun hinzufügen:

## Quellcode

1. `[xhprof]`
2. `extension = xhprof.so`
3. `xhprof.output_dir="/tmp"`

Nun noch Apache neustarten:

## Quellcode

1. `sudo /etc/init.d/apache2 restart`

Vereinzelnt hatte ich Probleme, wenn ich nicht den absoluten Pfad zur Extension angegeben habe. Ob die Extension richtig geladen wurde, überprüft ihr einfach über die `phpinfo()` Ausgabe. Falls also die Extension nicht geladen wurde, dann nimmt einfach den absoluten Pfad, den Ihr wie folgt ermitteln könnt:

## Quellcode

1. `sudo find / -name xhprof.so`

== XHProf im Quellcode ==

Der nächste Code-Ausschnitt zeigt die grundsätzliche Verwendung von XHProf.

## Quellcode

```
1. <?php
2. function bar($x) {
3.     if ($x > 0) {
4.         bar($x - 1);
5.     }
6. }
7. }
8. function foo() {
9.     for ($idx = 0; $idx < 2; $idx++) {
10.        bar($idx);
11.    }
12.    $x = strlen("abc");
13. }
14. }
15. // Profiling starten
16. // xhprof_enable() - normaler Aufruf
17. // die Argumente geben an, dass wir CPU und Memory-Verbrauch auch ermitteln wollen
18. xhprof_enable(XHPROF_FLAGS_CPU + XHPROF_FLAGS_MEMORY);
19. // der zu profilende Code
20. foo();
21. // Profiling stoppen
22. $xhprof_data = xhprof_disable();
23. // zeigt die gesammelten Daten an
24. echo "<pre>"; print_r($xhprof_data); echo "</pre>";
25. // einbinden der XHProf-GUI Dateien
26. include_once "../xhprof_lib/utils/xhprof_lib.php";
27. include_once "../xhprof_lib/utils/xhprof_runs.php";
28. $xhprof_runs = new XHProfRuns_Default();
29. // speichern der Daten - gibt die dazugehörige ID zurück
30. $run_id = $xhprof_runs->save_run($xhprof_data, "xhprof_foo");
31. // Link zur GUI-Seite
32. echo '<a href="http://localhost/xhprof_html/index.php?run='.$run_id.'&source=xhprof_foo">Profiling Data</a>';
33. ?>
```

Alles anzeigen

Die erzeugte GUI-Seite die über unsern erstellten Link erreichbar ist, stellt uns die Daten übersichtlich in Tabellenform da. Ebenso kann man sich auch einen Graphen anzeigen lassen.

Screenshots siehe [XHProf-GUI - Tabellenübersicht](#) und [XHProf-GUI - Callgraph](#)

== Ergänzungen ==

Das interessante an XHProf ist im Gegensatz zum Profiling mit XDebug, dass man auch Diffs zweier Durchläufe einsehen kann. Dazu benötigen wir nur die entsprechenden "Run-ID's" und rufen die GUI-Seite wie folgt auf:

## Quellcode

```
1. http://<xhprof-ui-address>/index.php?run1=<run_id1>&run2=<run_id2>&source=<namespace>
```

In unserem Beispiel könnte es so aussehen:

## Quellcode

```
1. http://localhost/xhprof_html/index.php?run1=4d03a2382f6c6&run2=4d03a23fb5dca&source=xhprof_foo
```

Die Unterschiede der beiden Run's werden farblich hervorgehoben.

Screenshot siehe: [XHPProf-GUI - Diff](#)

Des Weiteren kann man z.B. auch die Standard PHP-Funktionen grundsätzlich vom Profiling ausschließen:

### Quellcode

1. `xhprof_enable(XHPROF_FLAGS_NO_BUILTINS);`

oder man definiert nur gewisse Funktionen, die nicht beachtet werden sollen:

### Quellcode

1. `xhprof_enable(0,`
2. `array('ignored_functions' => array('call_user_func', 'call_user_func_array'))`
3. `);`

Weitere Infos zu XHPProf unter: [XHPProf-Dokumentation](#)