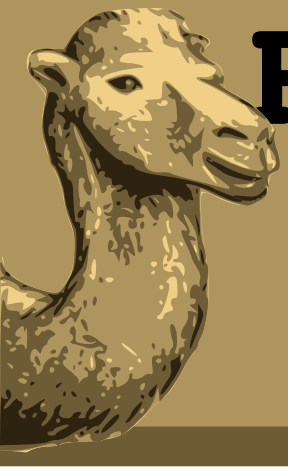
Perl Mongers im Ruhrgebiet



Ruhr.pm

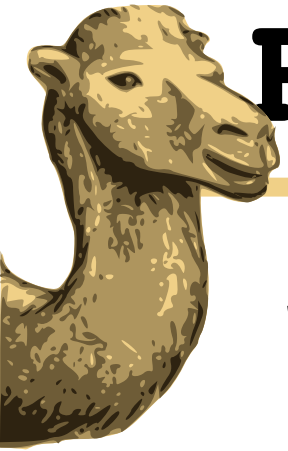
Web-Applikationen beschleunigen mit SpeedyCGI/PersistentPerl

Autor: Veit Wahlich

EMail: veit AT ruhr.pm.org

Datum: 21. April 2009

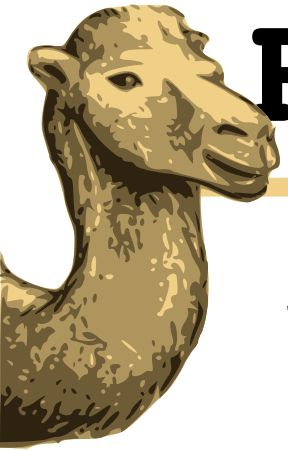
http://ruhr.pm.org/



Ruhr.pm

Was ist SpeedyCGI?

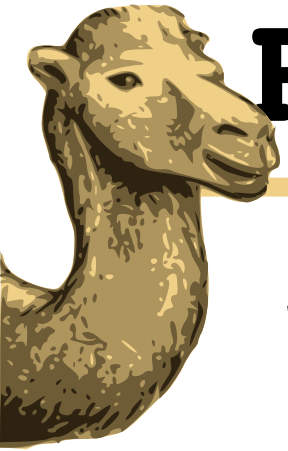
- persistenter, cachender Interpreter fuer in Perl geschriebene Web-Applikationen
- Funktion aehnlich mod_perl, aber
  - Nutzung mit z.B. Apache httpds mod_suexec problemlos moeglich
  - keine moegliche Beeinflussung der Integritaet oder Sicherheit des httpds durch Trennung in Frontend- und Backend-Prozesse
  - je nach Anwendung tendenziell groesserer Arbeitsspeicherbedarf



Ruhr.pm

Was ist SpeedyCGI?

- beschleunigte Zugriffszeiten
  - Perl-Interpreter muss nicht fuer jeden Zugriff auf ein Perl-CGI-Programm neu gestartet werden
  - Bytecode-Kompilate werden vorgehalten
- persistente Datenstrukturen
  - Daten und Objekte koennen an spaetere Iterationen von Programmen weitergegeben werden



Ruhr.pm

Was ist PersistentPerl?

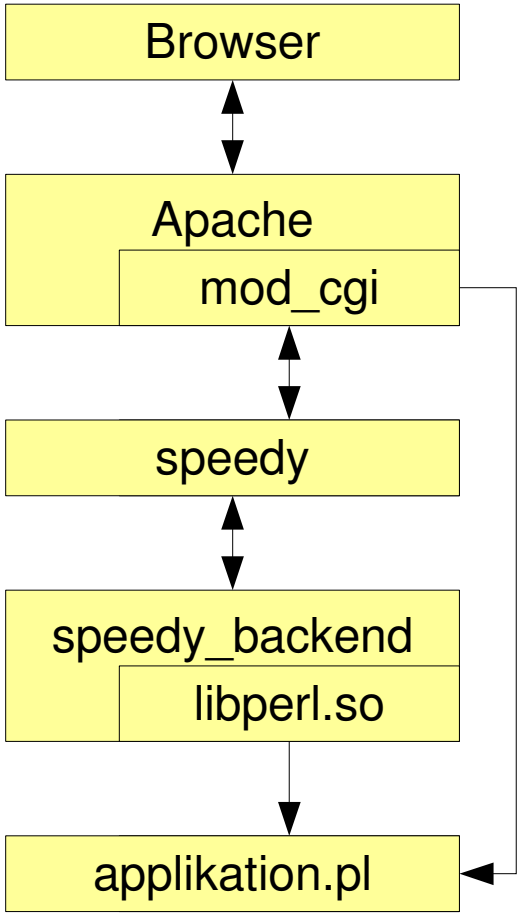
- zukuenftiger Name fuer SpeedyCGI
  - SpeedyCGI derzeit noch bekannter
  - SpeedyCGI bleibt Teil von PersistentPerl
- Namensaenderung, da SpeedyCGI prinzipiell fuer beliebige Perl-Programme verwendet werden kann
  - nicht nur CGI-/Web-Applikationen
  - daher oft auch nur “Speedy” genannt
- bislang gleiche Code-Basis wie SpeedyCGI

Ruhr.pm

Aufbau von SpeedyCGI

- Das SpeedyCGI-System teilt sich in 3 Teile auf:
  - Frontend
    - speedy-Binary fuer Aufruf per CGI
    - mod_speedycgi fuer Apache httpd
  - Backend
    - speedy_backend-Binary
  - Kontroll- und Statusmodul
    - CGI::SpeedyCGI
    - optional

SpeedyCGI per klassischem CGI



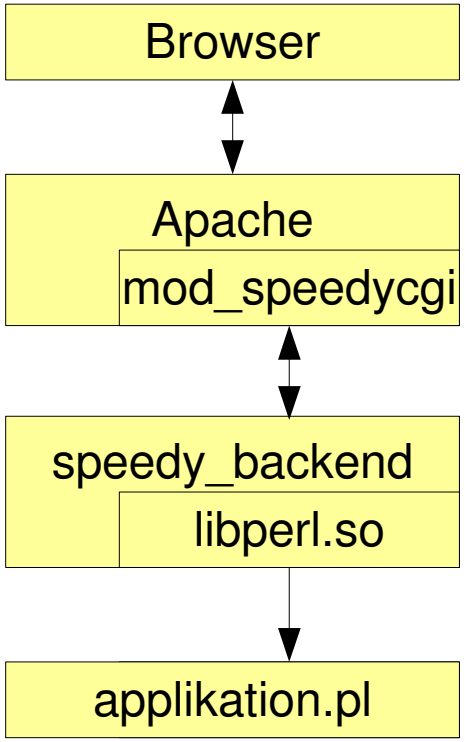
Der Browser setzt eine Anfrage an applikation.pl per HTTP ueber den Webserver (hier Apache) ab.

mod_cgi liest aus der Shebang von applikation.pl, dass die Ausfuehrung ueber speedy stattfinden soll.

speedy sucht eine verfuegbare Instanz von speedy_backend und startet ggf. eine neue.

speedy_backend laedt applikation.pl und fuehrt die Applikation aus.

SpeedyCGI mit mod_speedycgi



Der Browser setzt eine Anfrage an applikation.pl per HTTP ueber den Webserver (hier Apache) ab.

Apache uebergibt anhand eines Kriteriums (Location, Extension, ...) die Interpretation von applikation.pl an mod_speedycgi.

mod_speedycgi sucht eine Instanz von speedy_backend und startet ggf. eine neue.

speedy_backend laedt applikation.pl und fuehrt die Applikation aus.

Ruhr.pm

Funktionsweise des Frontends

- das Frontend entscheidet, ob und welchem Backend die Ausfuehrung eines Perl-Programms uebertragen wird
  - die Kommunikation mit den Backends findet ueber Unix-Sockets /tmp/speedy.*.*.S statt
  - Tracking ueber Datenbank /tmp/speedy.*.*.F
- wird kein verfuegbares Backend gefunden, startet das Frontend ein neues
  - sind bereits zu viele Backends aktiv, kann das Frontend die Ausfuehrung verweigern

Ruhr.pm

Funktionsweise des Backends

- das Backend fuehrt das uebergebene Perl-Programm aus
- nach der Ausfuehrung verbleibt der Prozess im Arbeitsspeicher und wartet auf neue Aufgaben
  - globale Variablen bleiben erhalten
  - wird das Backend eine vorgegebene Zeit (i.d.R. 1 Stunde) nicht benutzt, beendet es sich selbst
  - optional beendet sich das Backend auch auf Anweisung des Perl-Programms oder nach einer maximalen Anzahl Programmaufrufe

Ruhr.pm

Funktionen von CGI::SpeedyCGI

- Abfrage, ob das Perl-Programm gerade unter SpeedyCGI ausgefuehrt wird
- Registrieren von Aufraeum-Funktionen
  - aufraeumen beim Beenden des Perl-Programms
  - aufraeumen beim Herunterfahren des Backends
- Auslesen/Setzen von Konfigurationsparametern
- Herunterfahren des Backends nach Ausfuehrung
- sofortiges Herunterfahren des Backends

Ruhr.pm

Ein einfaches Beispiel

```
#!/usr/bin/speedy -w -- -t10

use strict;
use English;
use CGI::SpeedyCGI;

my $sp = new CGI::SpeedyCGI;

print("Content-Type: text/plain\n\n");

printf("Ich bin ein %s Prozess mit PID "
  . "%d!\n", $sp->i_am_speedy
  ? 'persistenter' : 'fluechtiger', $PID);

sleep(1);
```

Dieses Programm startet mit dem CGI-Interface **speedy**. Dabei wird der Parameter **-w** an den Perl-Interpreter uebergeben , der Parameter **-t10** hingegen setzt die Inaktivitaetszeit, nach der sich **speedy_backend** beendet, auf 10 Sekunden. Getrennt werden Parameter fuer Perl und **speedy** durch den Spezialparameter **--**.

Die **CGI::SpeedyCGI**-Methode **i_am_speedy** gibt einen wahren Wert zurueck, wenn das Programm unter SpeedyCGI ausgefuehrt wird.

Ruhr.pm

Persistente Variablen

```
#!/usr/bin/speedy -- -r10

use strict;
use warnings;
use DBI;
use CGI::SpeedyCGI;

use vars qw($dbh $counter);

my $sp = new CGI::SpeedyCGI;

print("Content-Type: text/html\n\n");

unless(defined($dbh) && $dbh->ping){
  $dbh = DBI->connect('dbi:SQLite:dbname='
    . 'speedydbi.sqlite', '', '')
    || do{
      print("Kann Datenbankverbindung nicht'
        . ' aufbauen.\n");
      $sp->shutdown_now;
    };
}
```

Der **speedy**-Parameter **-r10** laesst das Backend nach zehnmaligem Ausfuehren des Programms sich selbst beenden.

use vars ist die bevorzugte Methode, globale Variablen zu deklarieren. Diese sind persistent.

Nach dem Start pruefen, ob bereits eine Datenbankverbindung besteht. Wenn nicht, eine aufbauen, oder mit der Methode **shutdown_now** das Backend beenden, falls keine Verbindung aufgebaut werden kann.

Ruhr.pm

Persistente Variablen

```
$sp->add_shutdown_handler(sub{
    $dbh->disconnect;
  });

$counter = defined($counter)
  ? $counter + 1 : 0;

$dbh->do('INSERT INTO zugriffe '
  . '(pid, aufruf, aufrufer, zeitpunkt) '
  . 'VALUES (?, ?, ?, CURRENT_TIMESTAMP)',
  undef, ($$, $counter, $0));
```

Mit der **add_shutdown_handler**-Methode eine anonyme Funktion registrieren, welche die Datenbankverbindung beim Beenden des Backends sauber schliesst.

In **$counter** zaehlen wir, wie oft der Backend-Prozess bereits das Programm ausgefuehrt hat.

Eine Zeile wird in die Datenbanktabelle **zugriffe** eingefuegt, die den aktuellen Zugriff dokumentiert (PID, Aufrufszaehler, aufrufendes Script und Aufrufszeitpunkt).

Ruhr.pm

Persistente Variablen

```
print(<<'__EOF');
<html><body>
 <table>
  <tr>
   <th>PID</th><th>Aufruf Nr.</th>
   <th>Aufrufer</th><th>Zeitpunkt</th>
  </tr>
__EOF

foreach my $row (@{$dbh->selectall_arrayref(
    'SELECT pid, aufruf, aufrufer, ' .
    'zeitpunkt FROM zugriffe ' .
    'ORDER BY id DESC LIMIT 20'
  )}){
  printf(
     "  <tr>\n" .
    ("   <td>%s</td>\n" x 4) .
     "  </tr>\n",
    @{$row});
}

print(<<'__EOF');
 </table>
</body></html>
__EOF
```

Abschliessend erzeugen wir eine HTML-Seite mit einer Tabelle, die die 20 letzten Eintraege in der Zugriffstabelle anzeigt.

Ruhr.pm

Persistenzgruppen

- im regulaeren Betrieb erhaelt jede mit speedy gestartete Applikation pro Benutzer, der sie startet, ein eigenes Backend
  - hoher Arbeitsspeicherverbrauch, insbes. wenn Backends lange persistent im RAM verbleiben oder Applikationen aus vielen kleinen Programmen bestehen
- Persistenzgruppen erlauben es, verschiedene Perl-Programme mit den selben Backend-Prozessen auszufuehren

Ruhr.pm

Persistenzgruppen

- der Parameter **-g** weist speedy an, alle mit diesem Parameter gestarteten Programme des Benutzers mit gemeinsamen Backends auszufuehren
  - um unerwuenschte Nebeneffekte durch Gruppierung zu vermeiden, koennen mit **-g***name* Programme in einer Gruppe *name* zusammengefasst werden

Ruhr.pm

Persistenzgruppen

- globale Variablen werden nur jeweils zwischen den Aufrufen des selben Programms geteilt
  - zwei Programme, die nach einander das selbe Backend verwenden, koennen nicht auf die globalen Variablen des jeweils anderen zugreifen

Ruhr.pm

Persistenzgruppen

In speedydbi_member1.pl:

```
#!/usr/bin/speedy -- -r10 -gdatabase
...
```

In speedydbi_member2.pl:

```
#!/usr/bin/speedy -- -r10 -gdatabase
...
```

Das vorhergehende Beispiel “speedydbi.pl” wurde in die Dateien “speedydbi_member1.pl” und “speedydbi_member2.pl” kopiert und um den speedy-Parameter **`-gdatabase`** ergaenzt.

Dadurch werden beide Programme in der Gruppe “database” zusammengefasst und sie koennen sie selben Backend-Prozesse fuer die Ausfuehrung verwenden.

Ruhr.pm

Vielen Dank
fuer Eure Aufmerksamkeit

Ruhr.pm

Links

- SpeedyCGI (CGI::SpeedyCGI, speedy, speedy_backend, mod_speedycgi)
  - http://daemoninc.com/SpeedyCGI/
- PersistentPerl (PersistentPerl, perperl, perperl_backend, mod_persistentperl)
  - http://daemoninc.com/PersistentPerl/
- mod_perl
  - http://perl.apache.org/