PHP5 - The Next Generation

Alexander Merz

Einleitung

PHP4 ist mit mehr als 7 Millionen Installationen[1] das meistinstallierte Apache Webserver-Modul. Mit mehr als 350 vergebenen CVS-Accounts ist PHP fest in der OpenSource-Entwicklerszene etabliert. PHP ist leicht erlernbar und ist doch zugleich mächtig genug um weite Akzeptanz im Businessumfeld mit unternehmenkritischen Anforderungen zu finden.

In diesem Spannungsfeld zwischen ‚Low-End' und ‚Hig-End' bewegt sich die kontinuierliche Weiterentwicklung von PHP. Einerseits soll die Sprache weiter einfach zu begreifen und zu benutzen sein. Andererseits verstärkt sich der Druck seitens professioneller Programmierer, die parallel Java&Co nutzen und sich das ein oder andere Feature auch in PHP wünschen.

Aus diesem scheinbar widerstrebenden Anforderungen schält sich langsam die Liste neuer Merkmale für die Zend Engine 2 heraus. ‚Zend Engine 2' – war nicht von PHP5 die Rede? Ja und Nein.

Ein Blick zurück

Die neuen Features der ZE2 sind wahrscheinlich auch die neuen von PHP5[2]. Um diesen Dualismus zu verstehen, braucht es einen Blick in die PHP-Geschichte. Wer die Änderungen von PHP3 zu PHP4 studiert hat, fiel wohl der Umbau der internen Architektur von PHP ins Auge.

PHP bis einschließlich PHP3 war intern ein monolithische Interpreterprogramm. Der Interpreter nahm sich eine Skriptzeile vor, führte sie aus, kümmerte sich um die Ressourcenverwaltung etc. Wenn die Zeile abgearbeitet war, wurde mit der nächsten weitergemacht. Diese Art der Skriptverarbeitung resultierte noch aus den Anfängen von PHP. Das Vorgehen entsprach noch dem ursprünglichen Ansatz eines kleinen HTML-Template-Baukastens und weniger einer ausgewachsenen Programmiersprache. Das war aber PHP in der Version 3 schon längst geworden. Der Quellcode zu PHP3 war groß und aufgebläht, bei Optimierungen und Integration neuer Funktionen musste ständig auf Seiteneffekte an anderer Stelle geachtet werden. Die Luft für die effiziente Weiterentwicklung von PHP wurde dünn.

Aus diesem Grund präsentierten Zeev Suraski und Andi Gutsmann einen grundlegend neue interne Architektur für PHP. Beide hatten bereits erhebliche Anteil an der Weiterentwicklung von PHP, weshalb ihr Vorschlag positiv aufgenommen wurde.

Der Interpreter wurde aufgeteilt in Module[3] mit klar umrissenen Aufgaben (u.A. Script parsen, Ressourcenverwaltung, Scriptausführung). Auch wenn dieser Ansatz zunächst kompliziert wirkt, eröffnet er doch neue Möglichkeiten. Code- und Speicheroptimierungen

[1] Laut Netcraft.com
[2] deswegen werde ich im Text auch die Begriffe PHP5 und ZendEngine 2 synonym benutzen
[3] nicht zu verwechseln mit den PHP-Extensions, die es bereits mit PHP3 gab

können zielgerichtet durchgeführt werden. Der Einfluss durch Seiteneffekte auf andere Programmteile ist minimiert. Die Modularisierung macht die Pflege und Wartung einfacher.

Die Vorteile der Modul-Architektur zeigten sich z.B. im neuentwickelten Ressourcenmanager für PHP4.1 und in einer experimentellen Neuimplementierung des Language Scanners im Juli 2001. Erst die Abgrenzung der einzelnen Aufgabengebiete innerhalb des Interpreters ermöglicht die Entwicklung und den Einsatz von Binär-Caches wie dem Zend oder dem APC Cache.

Mittelpunkt und Mittler all dieser Module ist die Zend Engine. Sie stellt die grundlegenden PHP-Sprachelemente bereit und kümmert sich um die korrekte Zusammenarbeit der einzelnen Module und PHP-Extensions.

Auch wenn die Änderungen gravierend erscheinen, hat sich für den PHP-Programmier beim Umstieg von PHP3 auf PHP4 nicht viel geändert. Es mussten kaum Skripte umgeschrieben werden. Es gab auch keine wirklich neuen Sprachelemente (‚foreach' und die Array-Befehle). PHP 4 vergleicht sich zu PHP 3 wie ein getunter Golf zum Serienmodell. Tiefergelegt, Breitreifen, 10 Lautsprecher und 200 PS, aber trotzdem nur 5 Sitzplätze und der Kofferraum ist auch nicht größer geworden.

Die Überarbeitungen hatten erheblich zum Erfolg und zur Etablierung von PHP beigetragen. Damit vergrößerte sich das Anwendungsgebietes. Aufgrund der gestiegenen Anforderungen an PHP entschied man sich die Sprache um neue Elemente zu bereichern und zu beschleunigen. Das erzwingt eine Überarbeitung der Zend Engine, die in die Zend Engine 2 münden. Gemäß der Versionierungsregeln von PHP erzwingt eine solche Änderung einen großen Sprung auf PHP 5[4].

Soviel zum Hintergrund, aber nun zu den Neuerungen. Ich warne Sie allerdings! Die nachfolgenden Informationen sind eine Zusammenfassung aus den Diskussionen und Texten rund um die Zend Engine 2. Und die Diskussionen sind noch im Gange, es kann sich noch einiges ändern.

Neue vordefinierte Variablen

Einer der komfortabelsten Eigenschaften von PHP ist die automatische Bereitstellungen von Formulardaten oder URL-Parametern, wodurch diese ohne explizite Import zur Verfügung stehen. Das ist ein seit langem proklamiertes Sicherheitsproblem. Vorallem Anfänger kennen die dadurch möglichen Sicherheitsrisiken nicht, was zu einer großen Zahl unsicherer Scripte führt.

Aus diesem Grund arbeiten sicherheitsorientierte Programmierer grundsätzlich mit den vordefinierten $HTTP_*_VARS-Arrays. Vielen war es aber zuviel Tipparbeit. Seit PHP 4.1 stehen deshalb die neuen Variablen $_* ($_GET, $_POST, etc) bereit. Sie sind aber nicht nur kürzer, sondern auch ‚superglobal'. Sie stehen in Funktionen und Klassen ohne dem vorherigen Import mit global bereit. Auf diese Weise wird ein weiterer typischer Anfängerfehler beseitigt.

[4] Die ZE2 war einer der Gründe warum man die Regeln im Verlauf von PHP4 geändert hat – Die ZE2 wäre sonst trotz der großen Änderungen wohl in PHP 4.1 releast wurden.

In PHP 4.* wird man noch parallel auf beide Array-Arten zugreifen können. Unter PHP 5 wird $HTTP_*_VARS wohl wegfallen.

Funktionsnamen

In PHP sind Funktionsnamen im Gegensatz zu Variablenbezeichnungen ‚case-insensitiv'. Die Verwendung von Klein- und Großbuchstaben spielt keine Rolle[5]. So ist es in PHP derzeit nicht möglich Funktionen zu definieren, die sich nur durch die Schreibweise unterscheiden.

Folgendes klappt also nicht

```
<?php
function FooBar() {
}
function foobar() {
}
?>
```

Wie dies unter PHP 5 gehandhabt wird, stand noch nicht fest. Derzeit existieren drei Varianten.

1. Alles bleibt wie es ist. Das scheint auf den ersten Blick eigentlich nicht wünschenswert. Die Zend Engine 2 wird aber Namensräume unterstützen, weshalb die Überschneidung von Funktionsnamen nicht in diesem Maße ins Gewicht fallen muß. Ausserdem ist dieser Ansatz kompatibel zu früheren PHP-Versionen

2. Funktionsnamen sind case-sensitiv, es findet eine Unterscheidung von Gross- und Kleinschreibung statt. Diese Lösung wird von der Entwicklergemeinde favorisiert und ist auch die sauberste Lösung. Nachteilig ist der unter Umständen erhebliche Bruch mit der Kompatibilität.

3. Die „Zend Engine" schaut in die Kristallkugel. Die Idee wurde kurz aufgeworfen, fand aber keine großen Anklang, da er weiter unsaubere Programmierung erlaubt und mehrere Probleme auftreten können. Bei dieser Idee wird der Funktionsname intern gemäß der Definition gespeichert. Findet ein Funktionsaufruf statt, wird kontrolliert, ob eine entsprechende Funktion existiert, wenn nicht wird versucht eine Funktion zu ermitteln, die zu mindest den gleichen Namen hat. Diese Lösung würde weitestgehende Kompatibilität ermöglichen.

In diesem Zusammenhang stellt sich u.a. die Frage, warum man nicht einen neue Variable in der php.ini einfügt, die die Entscheidung dem Nutzer überlässt. Entsprechende Gedanken gab es, nicht nur bei dieser Diskussion. Aber man hat sich entschieden, grundsätzlich keine Variablen mehr einzuführen, die unmittelbar Einfluss auf Syntax und Semantik von PHP haben[6].

Try/Catch/Throw

Zwar geistern diese Schlüsselworte schon seit geraumer Zeit durch die Programmierszene . Wirklich populär sind sie aber erst durch Java geworden. Dort stellen sie einen integralen Teil der Fehlerbehandlung im Umgang mit den Klassenbibliotheken dar.

---

[5] PHP speichert intern alle Funktionsnamen in Kleinbuchstaben und wandelt Funktionsaufrufe wie z.B. isSet() automatisch in isset() um.

[6] Schon jetzt ist es schwierig genug, PHP-Skripte zu schreiben, die auf allen PHP-Installationen laufen

Die Idee ist die Vereinheitlichung der Fehlerbehandlung einhergehend mit eleganterem und besser lesbarem Code.

Typischerweise erfolgt die Fehlerbehandlung auf folgende Weise

```
<?php
if( $connect = mysql_connect()) {
	$query = "...";
	if($result = mysql_query ( $query)) {
		while( $row = mysql_fetch_row ( $result)) {
			//...
		}
	} else {
		echo mysql_error();
	}
} else {
	echo mysql_error();
}
?>
```

So etwas führt zwangsläufig zu vielen unübersichtlichen Schachtelkonstruktionen. Weiterhin muss man ständig im Kopf behalten, was eine Funktion überhaupt im Fehlerfall zurückliefert.

Der try/catch-Mechanismus vereinfacht das Ganze zu

```
try {
	$connect = mysql_connect();
	$query = "...";
	$result = mysql_query ( $connect, $query);
	while( $row = mysql_fetch_row ( $result)) {
		//...
	}
}
catch (DB_Exception $e) {
	switch ( $e) {
		case DB_NOCONNECT :
			//...
		break;
		case DB_WRONGQUERY :
			//...
		break;
	}
}
?>
```

Der Codeabschnitt in dem ein Fehler auftreten kann, wird mit try{} gekapselt. Tritt innerhalb einer Funktion ein Fehler auf, generiert die Funktion mit Hilfe von throw eine Instanz einer Fehlerklasse[7] (‚Execption') und die Funktion wird vom PHP-Interpreter beendet[8]. Die Ausführung des try-Blocks wird ebenfalls beendet und es wird in den catch-Block

[7] Allgemein spricht man dann vom ‚Werfen' eines Fehlers

[8] Ohne irgendwelche return-Anweisungen zu beachten! Eine Funktion, in der thrown ausgeführt wird, gibt nichts zurück.

gesprungen. Dort wird geprüft, ob das catch-Statment tatsächlich die geworfene Exception behandelt und führt den catch-Block gegebenenfalls aus.

Was passiert, wenn die Execption nicht in catch() erwartet wird oder try/catch überhaupt nicht verhanden ist, obwohl thrown benutzt wird, lies sich nicht ermitteln.

Die Implementierung orientiert sich deutlich an Java, ein Blick in die Java Dokumentation empfiehlt sich bei mehr Interesse.

Objektorientierung in PHP

Die Objektorientierung in PHP war stets ein großer Kritikpunkt, da sie allenfalls rudimentär vorhanden war. Das wird sich in PHP 5 grundlegend ändern und trotz neuer Features wird die Objektnutzung schneller sein als jemals zuvor[9].

Private Variablen

Schon seit PHP 3 gewünscht, wird es nun endlich private Klassenvariablen geben

```
<?php
class myClass {
      private $myVar;
      …
}
?>
```

$myVar ist nur innerhalb der Klasse sichtbar.

Konstruktoren und Destruktoren

Die Zend Engine 2 unterstützt zukünftig neben Konstruktoren auch Destruktoren. In diesem Zusammenhang werden auch zwei neue Schlüsselwörter eingeführt: __construct() und __deconstruct(). Mit ihnen erfolg die Deklaration des Konstruktors und des Dekonstruktors.

```
<?php
class myClass {
// Konstruktorfunktion
function __construct(){}
// Dekonstruktorfunktion
function __construct(){}
?>
```

Als Tribut an die Rückwärtskompatiblität akzeptiert PHP 5 noch die alte Form der Konstruktordeklaration. Findet die Zend Engine 2 keine Funktion __construct(), schaut es zusätzlich noch ob eine Funktion mit dem Namen der Klasse definiert wurde[10].

Übergabe per Reference

Ein typischer Tip erfahrener PHP-Programmierer ist die Übergabe von Objekte grundsätzlich per Reference durchzuführen

```
<?php
$klasse1 = &new Klasse();
```

[9] Erste einfache Test zeigten eine Beschleunigung von 30% bis 50%!

[10] Das kostet zwangsläufig Performance, deshalb gehe ich davon aus, dass dieses Feature früher oder später aus PHP herausfliegt.

```
$klasse2 = &$klasse1;
?>
```

Einerseits beschleunigt dies Anwendungen andererseits ist es oftmals erforderlich um ein korrektes Verhalten zu erreichen[11]. In PHP 5 wird die Übergabe und Zuweisung von Objekten grundsätzlich ‚by Reference' erfolgen.

Verwendet man unter PHP 5 obiges Codebeispiel, erhält man dann allerdings nicht eine Referenz auf eine Referenz, sondern nur eine einzelne Referenz. Die beiden Zeilen sind unter PHP5 identisch und führen zum gleichen Ergebnis[12]:

```
<?php
$klasse2 = &$klasse1;
$klasse2 = $klasse1;
?>
```

Echte Kopien - __clone()

Im Zusammenhang mit der Übergabe per Referenz stellt sich die Frage, wie man denn eine Kopie eines Objektes erhalten kann. Auch dafür gibt es eine Antwort und ein neues Schlüsselwort __clone().

Die __clone()-Funktion steht in allen Klasse automatisch zur Verfügung.

```
<?php
$klasse2 = $klasse1.__clone();
?>
```

Die Funktion kann aber auch in der eigenen Klasse überschrieben werden. Zum Beispiel soll ein Objekt immer seine Erstellungszeit als Attribut besitzen. In diesem Fall muss das Attribut der Kopie dem Zeitpunkt des Kopierens entsprechen.

```
<?php
class MyClass {
     var $createtime;
     var $irgendwas;
     …
     function __clone() {
         $this -> irgendwas = $clone -> irgendwas;
         $this -> createtime = time();
     }
}
$klasse1 = new MyClass();
$klasse2 = $klasse1.__clone();
?>
```

`$klasse2` besitzt außer dem Erzeugungsdatum die selben Eigenschaften wie `$klasse1.` `$clone` ist ein nur innerhalb der __clone()-Funktion vorhandenes Schlüsselwort analog zu $this. Es verweist auf die Instanz des zu kopierenden Objektes.

---

[11] Z.B. bei der Umsetzung von Software-Design-Patterns

[12] Vorraussichtlich erhält man aber unter PHP 5 bei der ersten Zeile eine Warnung, dass dieser Syntax ‚deprecated' ist – daher nicht mehr verwendet werden sollte.

Namensräume

Namensräume oder Namespaces dienen dazu Überschneidungen von Funktionens-, Konstanten-, Klassen- und Variablenbezeichungen zu vermeiden. Auf diese Weise müssen sich die Programmierer innerhalb eines Projektes nicht auf mögliche Namensgleichheiten Rücksicht nehmen.

Derzeit ist geplant Namensräume auf Basis des class-Schlüsselwortes zu implementieren. Der Zugriff auf die Elemente im Namensraum ähnelt dem Zugriff auf statische Klassenfunktionen.

```
<?php
class Foo {

     const CONST_FOO = "Konstante";
     class Bar {
          function test() {
          }
     }
}
echo Foo::CONST_FOO;
Foo::Bar::test();
?>
```

Foo ist der Bezeichner des Namensraumes und nicht die Deklaration einer Klasse!

Packages

PHP konnte bereits seit PHP 3 durch in C geschriebene Extensions erweitert werden. Wenn die Erstellung mittlerweile auch für C-Laien machbar ist, ist diese Hürde einer zweiten Programmiersprache oder im Rahmen einer schnellen Programm-Entwicklung für viele zu hoch. In PHP 5 wird es möglich sein, in PHP selbst geschriebene Erweiterungen, sogenannte Packages einzubinden.

Ein Package besteht aus Funktionen und Klassen, die in einem Namensraum gekapselt sind. Der Namensraum kann sich über mehrere Dateien erstrecken, die zusammen mit einer speziellen Package-Datei in einem Archiv gebündelt werden. Das Archiv kann beliebig verteilt werden. Mithilfe spezieller Tools kann dieses Archiv auf Basis der Package-Datei auf dem eigenen System installiert und verwaltet werden.

Um die Package-Elemente im Programm verfügbar zu machen, muss das Package noch im Quelltext eingebunden werden. Derzeit kursieren verschiedene Varianten über die Art und Weise. Die erste Variante ist die Einführung eines neuen Schlüsselwortes. Bislang im Gespräch sind ‚import', ‚use' oder ‚share'. Die zweite Variante ist eine Erweitung von ‚include/require'. Und drittens eine Erweiterung des dl()-Befehls, der bereits jetzt zur Einbindung von C-Extensions dient. Ein Grossteil der Programmierer plädiert derzeit für die erste Variante.

Einen ersten Einblick im Umgang mit Packages gibt PEAR[13]. Das „PHP Extension and Add-On Repository“ verfolgt einen ähnlichen Ansatz wie CPAN für Perl. Es ist der zentrale Sammelpunkt für hochwertige, gut gepflegter Packages in PHP. PEAR stellt bereits einen Grossteil der Infrastruktur (Werkzeuge und Standards) zur Erzeugung und Verteilung von Packages bereit und besteht derzeit aus ca. 40 Packages. Die bestehenden Packages lassen sich bereits unter PHP 4 nutzen. Nachteilig wirkt sich hier die etwas problematische Installation und Einbindung ‚zu Fuss’ per include/require aus. Auch die Tools verlangen hin und wieder durch spartanischer Dokumentation und Beta-Status etwas Forscherfleiß.

Fazit

PHP5 ist ein großer Sprung in der Evolution von PHP. Nichtsdestotrotz bleibt PHP die einfach zu erlernende Sprache von früher, aber gepaart mit größerer Mächtigkeit. PHP5 wird verstärkt mit JSP konkurrieren. Einerseits durch seine alten und neuen Fähigkeiten, andererseits durch seine große Community und zunehmend verfügbaren kommerziellen Support.

Auf eins will ich aber nochmals deutlich hinweisen: PHP5 wird bewusst mit der Kompatibilität zu früheren Versionen gebrochen. Ein Schritt der manchen Anwender weh tun und Arbeit bereiten wird, sollen die bisherigen Applikationen mit unverminderter Kraft weiterlaufen. Aber die neuen Anwendungsfelder für PHP und eine zukünftig verkürzte Entwicklungs- und Ausführungszeit machen diesen Nachteil mehr als wett.

Zwei Bitten

Nun hätte ich noch zwei Bitten an Sie:

1. PHP ist ein OpenSource-Projekt/Produkt. Jeder ist aufgerufen zu helfen. Die Zend Engine2 ist immer noch im Entwicklungsprozess. Wenn Sie der Meinung sind, dass Sie ein Feature vermissen und/oder einen Geistesblitz für die Lösung des ein oder anderen Problems/ Diskussion haben, scheuen Sie sich nicht eine EMail an die Zend Engine 2-Mailingliste[14] zu senden! PHP lebt vom Mitmachen.

2. Dieses Dokument entstand aus einem Vortrag zu PHP5 und basiert auf dem Stand von Ende Februar. Wenn Sie Fehler finden oder neue Infos haben schreiben Sie mir[15]. Ich bin leider nicht immer in der Lage die Diskussion auf obiger Mailingliste zu verfolgen und verpasse eventuell das ein oder andere. Auch Fragen für die FAQ sind willkommen.

FAQ

1. **Wann wird PHP 5 erscheinen.**
   Keine Ahnung! Es gibt leider keine Roadmaps. Es scheint sich allerdings der Spätsommer/Herbst für die ersten Release Candides[16] herzukristallisieren.
2. **Kann ich die Zend Engine 2 bereits testen?**
   Ja, die Zend Engine 2 funktioniert auch mit PHP 4. Laden Sie sich aus dem CVS den PHP-Quellcode herunter[17]. Wichtig ist dabei, das nicht das Verzeichnis ‚Zend’, sondern das Verzeichnis ‚ZendEngine2’ heruntergeladen wird. Das Verzeichnis muss dann lokal in ‚Zend’ umbenannt werden. Danach kann man den angegeben Build-

[13] Das Package-Konzept ist durch Druck seitens der PEAR-Entwickler überhaupt erst in die Zend Engine 2 aufgenommen wurden.
[14] engine2(at)lists.php.net
[15] alexmerz(at)php.net
[16] RC = Vorabversionen zum Testen
[17] http://www.php.net/anoncvs.php

Prozess folgen. Man sollte aber bei der Installation aufpassen um nicht eine vorhandene Installation zu überschreiben. Auch darf man nicht vergessen, dass die Zend Engine 2 noch den einen oder anderen größeren Bug enthält.

3. **Wird PHP 5 auch parallel zu PHP 4 laufen können?**
   Bis jetzt gibt es keine Aussagen die das Gegenteil behaupten. Also kurz „ja“, wahrscheinlich gilt ähnliches wie für die Parallelinstallation von PHP 3 und 4. Allerdings wird man sich Gedanken wegen der Dateiendungen machen müssen (*.php5?).

Infos

Das Zend White Paper zur Zend Engine 2
http://www.zend.com/engine2/ZendEngine-2.0.pdf

Zend Engine 2 Mailingliste-Archiv
http://www.zend.com/lists.php

Zend Engine 2 Mailingliste – Wöchentliche Zusammenfassung (leider nur bis Oktober 2001)
http://www.zend.com/zend/zengine/