Kapitel 8
Neuerungen in PHP 5

von Hartmut Holzgraefe

Die Entwicklungsarbeiten an PHP 5, der nächsten »großen« PHP-Version, laufen bereits seit einiger Zeit. Seit kurzem werden erste Vorab-Versionen als Snapshots auch öffentlich zur Verfügung gestellt. Die Entwicklung ist zwar noch nicht abgeschlossen und auch die Testphase wird noch einmal einige Zeit in Anspruch nehmen, eine Veröffentlichung noch in diesem Jahr (2003) ist aber fest vorgesehen. Dieses Kapitel gibt daher schon einmal einen Überblick über die bereits umgesetzten und noch geplanten Änderungen.

Es macht übrigens trotz allem Sinn, sich jetzt noch mit PHP 4 zu beschäftigen. Erfahrungsgemäß stellen z. B. Provider nur sehr zögerlich auf neuere Versionen um, sodass sie hier auch weit über das Release-Datum von PHP 5 hinaus diese Version noch nicht voraussetzen dürfen. Ebenso kann sich der Release-Zeitpunkt selbst noch weiter nach hinten verschieben.

Kernstück von PHP 5 wird die komplett überarbeitete ZendEngine 2 sein. Dies ist auch in den ersten PHP 5-Snapshots der einzig wesentliche Unterschied zu PHP 4, das restliche System ist zunächst weiter aus den gleichen Quellen wie PHP 4.3 erstellt und daraus weiterentwickelt.

Eine weitere deutliche Änderung wird sein, dass PHP 5 mit wesentlich weniger integrierten Erweiterungsmodulen ausgeliefert wird. In der Vergangenheit wurden alle Extensions direkt in den PHP Quellcode aufgenommen. Mit mittlerweile etwa 100 Extensions, die insgesamt etwa 4.000 Funktionen zur Verfügung stellen, macht dieses Verfahren keinen Sinn mehr. PHP 5 wird daher nur noch eine Grundausstattung von Extensions enthalten, die auch tatsächlich in der Mehrheit der Installationen verwendet werden (wie z. B. Sessions, XML und MySQL). Im Wesentlichen wird es sich dabei um die Extensions handeln, die in PHP 4.3 bereits per Default aktiviert sind.

Alle anderen Extensions werden in das PECL Repository überführt, das ein Teil des PEAR-Systems ist (siehe vorangegangenes Kapitel). PECL steht dabei für *PHP Extension C Library*. Unter PECL verwaltete Extensions können dann über den PEAR-Installer heruntergeladen und installiert werden. Mehr zu PEAR später in einem weiteren Kapitel.

Weitere angestrebte Änderungen sind unter anderem eine einheitliche Namenskonvention für interne Funktionen, mehr thread-feste Extensions und Möglichkeiten zur Eingabefilterung noch vor der Script-Ausführung.

Die folgenden Abschnitte sind bewusst relativ kurz gefasst. Viele der beschriebenen neuen Features sind noch nicht final, d. h. es können sich bis zum tatsächlichen Release durchaus noch Änderungen ergeben. Es lohnt sich daher an dieser Stelle noch nicht, zu sehr ins Detail zu gehen, vielmehr geht es uns hier um eine kurze Vorstellung der neuen Funktionalitäten. Seien sie daher bitte auch nicht allzusehr überrascht, wenn einzelne Dinge in der Zukunft nicht mehr so funktionieren wie hier beschrieben.

8.1 ZendEngine 2

Die neue ZendEngine bietet neben verbesserter Performance und kleineren Korrekturen und Verbesserungen vor allem eine deutlich verbesserte Unterstützung für objektorientierte Entwicklung. In diesem Abschnitt stellen wir die neuen Möglichkeiten kurz vor, wobei wir davon ausgehen, dass sie bereits mit den vergleichbaren Eigenschaften anderer objektorientierter Sprachen wie C++, Java oder C# vertraut sind. Dieser Abschnitt versteht sich noch weniger als allgemeines Lehrbuch, als dies bereits bei der PHP 4-Sprachbeschreibung der Fall war.

8.1.1 Referenzen statt Kopien

In PHP 4 werden Objekte bei jeder Zuweisung und bei jeder Übergabe an und Rückgabe aus Funktionen kopiert, wenn nicht ausdrücklich eine Weitergabe per Referenz erzwungen wurde. Dies führt zu Performance-Verlusten und entspricht auch nicht dem üblichen Verhalten objektorientierter Sprachen. Insbesondere der Umgang mit untereinander verknüpften Objekten gestaltet sich so schwierig.

In PHP 5 werden dagegen Objekte automatisch per Referenz übergeben und zugewiesen. Dies steigert nicht nur die Performance im Umgang mit Objekten (dies hätte auch in PHP 4 durch konsequenten Einsatz des Referenz-Operators `&` erreicht werden können), sondern macht auch das Objekt-Handling mit Funktionen wesentlich einfacher (in PHP 4 reicht oft schon ein vergessenes `&` bei einer Zuweisung oder Übergabe, um schwer nachvollziehbare Fehler zu produzieren).

Soll tatsächlich eine Kopie eines Objekts erzeugt werden, so kommt die neue Methode `__clone()` zum Einsatz. Eine Klasse kann eine eigene `__clone()`-Methode definieren und so das Verhalten von Objekten beim Kopieren selbst bestimmen. Ist keine `__clone()`-Methode vorhanden, so werden wie in PHP 4 einfach alle Properties 1 zu 1 kopiert.

8.1.2 Konstruktoren und Destruktoren

In PHP 4 muss der Name der Konstruktor-Methode mit dem Klassennamen übereinstimmen. Dies macht sich spätestens dann unangenehm bemerkbar, wenn man den Namen einer Klasse ändern muss und bei dieser Gelegenheit

auch in allen davon abgeleiteten Klassen den Aufruf des Konstruktors der Basisklasse anpassen muss. Destruktoren sind in PHP 4 nicht vorhanden.

In PHP 5 werden Konstruktor und Destruktor einer Klasse über die speziellen Methoden `__construct()` und `__destruct()` implementiert. Wenn keine `__construct()`-Methode vorhanden ist, wird aus Gründen der Kompatibilität weiter eine Methode, deren Name mit dem Klassennamen übereinstimmt, als Konstruktor genutzt.

Innerhalb des Konstruktors einer abgeleiteten Klasse kann der Konstruktor der Basisklasse nun mit `parent::__construct(...)` aufgerufen werden, damit sind hier auch bei Änderungen der Klassennamen keine Anpassungen mehr nötig.

Der Destruktor einer Klasse wird ausgeführt, sobald ein Objekt nicht mehr benutzt wird. Dies wird über den internen Referenz-Zähler erkannt, über den jede PHP-Variable verfügt. Die Destruktor-Aufrufe erfolgen dabei wie in C++, sobald ein Objekt nicht mehr benutzt wird und nicht erst verzögert wie bei der `finalize`-Methode in Java oder C#, die erst beim nächsten Lauf des Garbage-Collectors angestossen wird.

8.1.3 Zugriffsschutz über private, protected und public

In PHP 4 waren alle Member-Variablen und Methoden einer Klasse grundsätzlich `public`, d. h. auch in allen abgeleiteten Klassen sowie von außerhalb der Klasse sichtbar und benutzbar.

Mit PHP 5 werden die Schlüsselworte `private`, `protected` und `public` eingeführt, über die sich der Zugriff auf Member-Variablen und Methoden beschränken lässt. Mit `private` deklarierte Variablen und Methoden sind dabei nur innerhalb der Klasse selbst verfügbar. `protected` gestattet den Zugriff auch für abgeleitete Klassen, dabei muss allerdings die `protected` Deklaration für Member-Variablen in jeder abgeleiteten Klasse wiederholt werden. `public` entspricht dem alten Default-Verhalten und erlaubt Zugriffe auch von außerhalb der Klasse.

Der Versuch, unberechtigt eine `private` oder `protected` Methode auszuführen hat eine Fehlermeldung zur Folge und die Programmasuführung wird abgebrochen. Ein unberechtigter Zugriff auf eine geschützte Member-Variable dagegen wird aus Performance-Gründen zwar verhindert, aber nur wie ein Zugriff auf eine nicht definierte Variable behandelt.

8.1.4 Überladene Member-Variablen und Methoden

PHP 5 gestattet es Klassen, die Funktionsweise von Zugriffen auf Member-Variablen und Methoden selbst zu verwalten. Hierzu dienen bei Member-Variablen die speziellen Methoden `__get()` und `__set()` sowie bei Methoden die spezielle Methode `__call()`.

Bei einem Zugriff auf eine nicht deklarierte Member-Variable wird die `__get()`-Methode aufgerufen, falls vorhanden. Diese erhält als einzigen Parameter den Namen der gewünschten Member-Variablen und liefert deren Wert zurück.

Wie dieser Wert bestimmt wird und wie mit ungültigen Namen umzugehen ist inclusive eventueller Fehlermeldungen, ist nun vollständig die Aufgabe dieser Methode.

Bei einer Zuweisung an eine nicht deklarierte Member-Variable wird entsprechend die Methode `__set()`, wenn vorhanden, mit Name und zugewiesenem Wert der Variablen der Zuweisung aufgerufen. Auch hier sind der Umgang mit diesen Werten und eventuelle Fehlermeldungen nun ausschließlich Aufgabe dieser Methode.

Ist die spezielle Methode `__call()` implementiert, so führen Aufrufe nicht implementierter Methoden einer Klasse nicht mehr zu Fehlermeldungen, vielmehr wird der Aufruf an `__call()` weitergeleitet. `__call()` erhält dabei als ersten Parameter den Namen der gewünschten Funktion und als zweiten Parameter ein Array mit den ursprünglichen Aufrufparametern.

8.1.5 Direkte Nutzung von zurückgegebenen Objekten

In PHP 4 ist es nicht möglich, direkt auf Member-Variablen oder Methoden eines Objektes zuzugreifen, das von einer Funktion zurückgegeben wurde. Sie müssen dieses vielmehr zunächst einer Variablen zuweisen, bevor sie es benutzen können.

In PHP 5 entfällt dieser oft lästige Zwischenschritt, sie können nun ein von einer Funktion zurückgegebenes Objekt direkt weiterbenutzen. Leider fehlt ein entsprechendes Feature für Arrays bisher noch.

```
<?php
  class foo {
    public $bar="member bar";
    function bar() { echo "methode bar";}
  }

  function f() {
    return new foo;
  }

  echo f()->bar;
  f()->bar();

  // leider noch nicht möglich (1.2.2003):
  function a() { return array("hund","katze","maus"); }

  echo a()[1]; // syntax error statt "katze"
?>
```

8.1.6 instanceof

Bereits in PHP 4 sind die Funktionen `is_a()` und `is_subclass_of()` vorhanden, mit denen die Abstammungsverhältnisse einer Klasse überprüft werden können.

PHP 5 bietet darüber hinaus den neuen Vergleichs-Operator `instanceof`.

```
<?php
class a {}
class b extends a {}
class c {}
$a = new b;

// PHP 4
echo is_($a, "a") ? "ja":"nein"; // ja
echo is_($a, "b") ? "ja":"nein"; // ja
echo is_($a, "c") ?  "ja":"nein"; // nein

// neu in PHP 5
echo $a instanceof a ? "ja":"nein"; // ja
echo $a instanceof b ? "ja":"nein"; // ja
echo $a instanceof c ? "ja":"nein"; // nein
?>
```

8.1.7 Statische Member-Variablen und Konstanten

Bereits in PHP 4 besteht die Möglichkeit, über den :: Operator Methoden einer Klasse auch ohne eine Instanziierung derselben aufzurufen. In diesem Fall steht in der Methode dann natürlich keine Objekt-Referenz auf `$this` zur Verfügung.

PHP 5 erweitertet dieses Konzept um statische Member-Variablen und Konstanten. Diese haben in allen Instanzen einer Klasse den gleichen Wert und sind auch unabhängig von einer Instanziierung mit :: ansprechbar. Wird der Wert einer statischen Member-Variablen in einer Instanz geändert, so ändert dieser sich automatisch in **allen** Instanzen analog.

```
<?php
  class foo {
     const c='foobar';
     static $bar = 1;
  }

  echo foo::c;    // -> foobar
  echo foo::$bar; // 1

  class instance_count {
    static $count = 0;
    public $id;

    function __construct() {
      $this->id = ++self::$count;
```

```
    }
  }

  $a = new instance_count;
  $b = new instance_count;

  echo "a is {$a->id} of ".instance_count::$count."\n"; // 1 of 2
  echo "b is {$b->id} of ".instance_count::$count."\n"; // 2 of 2
?>
```

8.1.8 Abstrakte Methoden und Klassen

Über das Schlüsselwort `abstract` können Methoden als abstrakt deklariert werden, damit wird auch die gesamte Klasse abstrakt. Damit ist die Instanzierung von Objekten dieser Klasse nicht mehr möglich, vielmehr dient die Klasse so nur als Vorlage für davon abgeleitete Klassen. Diese müssen die als abstrakt deklarierten Methoden dann zwingend implementieren, um instanzierbar zu sein, ansonsten bleibt auch die abgeleitete Klasse weiter abstrakt.

```
<?php
  // abstrakte Klasse
  class a {
    abstract function foo();
    abstract function bar();
  }

  // immer noch abstrakte Klasse
  class b extents a {
    function foo() { return false; }
  }

  // nicht mehr abstrakt, damit implementierbar
  class c extents b {
    function bar() { return false; }
  }

  $c = new c; // in Ordnung

  $b = new a; // PHP Fatal error:  Cannot instantiate abstract class a
?>
```

8.1.9 Exception-Handling

PHP 5 führt die als neues Feature Exception-Handling ein, wie es auch aus anderen Sprachen wie Java und C++ bekannt ist. Hiermit lässt sich übersichtlicher Code erzeugen, da die eigentliche Funktionalität und die Fehlerbehandlung so voneinander getrennt werden können. Für das Exception-Handling werden die neuen Schlüsselwörter `try`, `catch` und `throw` eingeführt. Wenn in

bestehendem PHP 4 Code Funktionen mit diesen Namen bereits definiert und verwendet werden, so müssen diese für den Einsatz unter PHP 5 leider umbenannt werden.

Ein Codeblock, in dem Exception-Handling stattfinden soll, wird mit `try` eingeleitet. Im Anschluss an den eigentlichen Code folgen ein oder mehrere `catch`-Blöcke, die jeweils eine mögliche Exception abfragen und Behandlungs-Code dafür enthalten. Es wird dabei nur der jeweils erste passende `catch`-Block ausgeführt.

Innerhalb eines `try`-Blocks wird eine Exception mit `throw` ausgelöst. `throw` erwartet als Argument ein Objekt, das an die `catch`-Blöcke weitergegeben werden kann. Die Ausführung eines `try`-Blocks wird durch ein `throw` sofort beendet. Anders als in Java existiert keine Einschränkung der als Exception verwendbaren Klassen.

Jede `catch` Anweisung benötigt als Parameter den Namen einer Klasse und einer Variable. Der erste `catch`-Block, für den das mit `throw` übergebene Objekt von der angegeben Klasse ist bzw. von dieser abstammt, wird ausgeführt, die Exception wird dabei in der angegebenen Variable abgelegt. Passt das Exception-Objekt zu keinem `catch`-Block, so wird die Ausführung mit einer Fehlermeldung abgebrochen. Eine `finally`-Anweisung gibt es leider zur zeit noch nicht.

```
<?php
  class ex1  { public $type = "EX1"; }
  class ex2  { public $type = "EX2"; }
  class ex2b extends ex2 { public $type = "EX2b"; }

  try
  {
    throw new ex2b;
  }
  catch (ex1 $extension) {
    echo $extension->type;
  }
  catch (ex2 $extension) { // match
    echo $extension->type;
  }
  catch (ex2b $extension) {
    // überflüssig, exb2 ist Subklasse von ex2,
    // wird damit immer schon vom vorigen "catch" abgefangen
  }
?>
```

8.1.10 Verschachtelte Klassen und Namensräume

PHP 4 kennt nur drei Namensräume: einen für globale Variablen, Funktionen und Klassen, einen für Member-Variablen und Methoden innerhalb von Klassen und einen für lokale Variablen innerhalb von Funktionen und Methoden.

In PHP 5 lassen sich über die neu hinzugekommene Möglichkeit, Klassen innerhalb anderer Klassen zu deklarieren, beliebig weitere Namensräume schaffen.

Der bereits in PHP 4 vorhandene Zugriffs-Operator wird dabei in seinen Anwendungsmöglichkeiten erweitert, da über ihn nun auch der Zugriff auf verschachtelte Klassen geregelt wird.

```
<?php
  // verschachtelte Klassen
  class outer {
    function foo() { echo "outer";}

    class inner {
      function foo() { echo "inner";}
    }

    class inner2 {
      function bar() { echo "inner";}
    }

    class inner3 extends outer {
      function bar() { echo "inner";}
    }
  }

  // Zugriffe über ::
  outer::foo();   //     -> outer
  outer::inner::foo(); //-> inner

  // outer::inner2::foo(); -> fehler
  // inner2 ist zwar in outer enthalten, erweitert diese aber nicht

  // das folgende funktioniert, da inner3 outer erweitert
  // und so foo() von outer erbt
  outer::inner3::foo(); // -> outer

  // verschachtelte Klassen können auch nachträglich deklariert werden
  class outer::inner4 {
    function bar() { echo "inner";}
  }
  outer::inner4::bar();
?>
```

Damit erweitert sich unter anderem die schon in PHP 4 bestehende Möglichkeit, Funktionsbibliotheken als statische Methoden einer Klasse zu implementieren und so einen eindeutigen Namespace zu schaffen. Allerdings ist die ständige Eingabe von `klassenname::` bei Nutzung dieses Konzeptes auf die Dauer ermüdend. PHP 5 bietet alternativ die Möglichkeit, statische Methoden und Member-Variablen sowie Klassen aus anderen Namensräumen in den eigenen Namensraum zu importieren.

```
<?php
  // verschachtelte Klassen
  class outer {
    class inner {
      function foo() { echo "foo";}
      function bar() { echo "bar";}
    }
    class inner2 {
    }
  }

  import function * from outer::inner;
  echo foo();
  // entspricht echo outer::inner::foo();

  import class inner2 from outer;
  $obj = new inner2;
  // entspricht $obj = new outer::inner2;
?>
```

Die Namensräume `self::` und `parent::` sind bereits in PHP 4 reserviert, in PHP 5 kommt hier noch der neue Namensraum `main::` hinzu. Dieser dient dazu, explizit auf eine globale Funktion oder Variable zuzugreifen.

8.1.11 Optionale Referenz-Parameter

In PHP 4 können optionale Parameter und die Übergabe per Referenz nicht kombiniert werden, der Versuch führt zu einem Parse Error.

In PHP 5 ist die Zuweisung eines Default-Werts an einen Referenz-Parameter dagegen nun möglich. Dabei gelten die üblichen Regeln, d. h. es können nur konstante Werte zugewiesen werden. Wird beim Aufruf keine Variable per Referenz übergeben, so haben Zuweisungen an den optionalen Parameter keine Wirkung über die Funktion hinaus.

```
function optional_reference(&$param = false) { ... }
```