Carsten Möhrke

Inkl. PHP 5.4

Besser PHP programmieren

Handbuch professioneller PHP-Techniken

Design Patterns
PHPUnit
Ajax
Subversion
Debugging
Sicherheit
Errorhandling
Zend Studio
jQuery
CouchDB
Zend Framework
Performance-Optimierung
MVC-Architektur
u. v. m.

4., aktualisierte und erweiterte Auflage

Galileo Computing

Inhalt

If it doesn't fit, use a bigger hammer.
– Miles O'Brian, Star Trek

2 Datentypen und -konvertierung

2.1 Datentypen in PHP

Im Gegensatz zu vielen anderen Programmiersprachen unterstützt PHP leider keine Variablendeklaration, und der Programmierer hat auch nicht die Möglichkeit, einer Variablen einen Typ zuzuweisen. PHP ermittelt selbst, welchen Typ eine Variable haben muss, und führt ein automatisches »Type Casting« durch. Das heißt, eine Variable kann automatisch von einem Typ in einen anderen konvertiert werden. Das führt dazu, dass sich viele Entwickler keine Gedanken über den Datentyp machen. Frei nach dem Motto »PHP wird's schon richten« gehen sie davon aus, dass die Konvertierung automatisch erfolgt und keine unangenehmen Nebeneffekte haben wird. Leider ist das nicht immer so.

Aber ich möchte die implizite Konvertierung nicht nur verteufeln. Sie macht vieles einfacher. Und wenn Sie mal mit einer streng typisierten Sprache gearbeitet haben, dann wissen Sie, wie nervig es sein kann, wenn man Datentypen ständig konvertieren muss. Bitte vergessen Sie aber nie, dass die implizite Konvertierung schnell zu einem Problem werden kann.

PHP kennt vier skalare Datentypen:

- Boolean
- Integer
- Float (Double)
- String

2.1.1 Boolean

Bei Boolean handelt es sich um einen Datentyp, der nur zwei Werte unterstützt, und zwar `true` (wahr) und `false` (falsch). Häufig wird `true` mit dem Wert 1 und `false` mit dem Wert 0 gleichgesetzt, was so aber nicht korrekt ist. Wird ein Boolescher Wert in einer Berechnung genutzt, konvertiert PHP ein `true` allerdings in 1

und `false` in 0. Das funktioniert natürlich auch in die andere Richtung. Die Zahl 0 kann in ein `false` konvertiert werden, wohingegen alle anderen Werte als `true` interpretiert werden. Aber Achtung: eine Besonderheit bei Booleschen Werten führt schnell zu Irritationen. Stellen Sie sich vor, Sie wollen zum Debuggen »mal gerade schnell« den Wert einer Variablen auslesen. Nichts einfacher als das:

```
$a = true;
$b = false;
echo '$a: '.$a;
echo "<br>";
echo '$b: '.$b;
/* Ausgabe im Browser:
$a: 1
$b:
*/
```

Lassen Sie Boolesche Werte direkt per `echo` ausgeben, oder konvertieren Sie sie in einen String, dann wird also nur das `true` in eine `1` konvertiert. Das `false` hingegen resultiert in einem Leer-String.

Boolesche Werte werden häufig als Rückgabewerte von Funktionen genutzt.

2.1.2 Integer

Bei dem Datentyp Integer handelt es sich um ganzzahlige Werte, also Zahlen ohne einen Nachkommaanteil. Wie groß ein Integer-Wert sein darf, ist heutzutage nicht mehr ganz so einfach zu beantworten. Das hängt nämlich von dem Betriebssystem ab, auf dem PHP ausgeführt wird. Nutzen Sie ein 32-Bit-System, dann kann der Wert zwischen –2.147.483.648 und 2.147.483.647 liegen. Nutzen Sie allerdings ein 64-Bit-System, dann steht Ihnen der Wertebereich von –9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807 zur Verfügung. Wie groß der Wertebereich ist, den Ihr System abdeckt, können Sie der Konstante `PHP_INT_MAX` entnehmen. Sie enthält den maximalen Wert. Der minimale Wert entspricht dem maximalen mit negativem Vorzeichen minus 1.

PHP nutzt hierbei einen sogenannten »signed Integer«. Das heißt, dass sowohl der negative als auch der positive Zahlenbereich abgebildet werden kann. Ein »unsigned Integer«, also eine Integer-Darstellung, die nur positive Werte unterstützt, kann in PHP nicht genutzt werden.

Bei der Wertzuweisung sollten Sie beachten, dass PHP auch das hexadezimale und das oktale Zahlensystem unterstützt. Eine Zahl, die mit einer 0 beginnt, wird als oktal gewertet und ein Wert, der mit einem 0x beginnt, als hexadezimale Zahl. Dies kann schnell zu einem unerwarteten Verhalten führen:

```
$oktal = 0815; // beliebte Zahl zum Testen
echo $oktal; // gibt 0 aus, da 8 keine gültige oktale Zahl ist

$oktal2 = 042; // auch eine beliebte Zahl
echo $oktal2; // gibt 34 aus

$hex = 0x11;
echo $hex; // gibt 17 aus
```

Achten Sie also auch bei Werten, die Sie »nur mal gerade zum Testen« eingeben, sehr genau darauf, dass Sie nicht mit einer führenden Null arbeiten.

Ab PHP 5.4 haben Sie übrigens auch die Möglichkeit, Zahlen binär anzugeben. Das heißt, Sie können Integer-Werte mit Nullen und Einsen angeben, wenn Sie der Zahl 0b voranstellen.

```
<?php
$int = 0b10;
echo $int; // gibt 2 aus
?>
```

Hier wird 2 ausgegeben, weil das Bit an der Stelle 0 nicht gesetzt und das Bit an der Stelle 1 gesetzt ist. Das heißt, die Zahl wird von rechts nach links interpretiert, und die 0 steht dafür, dass das Bit 2^0 nicht gesetzt ist. Die 1 bedeutet, dass das Bit an der Stelle 2^1 gesetzt ist. Hätte ich 0b110 angegeben, dann würde sich die Zahl *6* ($2^2 + 2^1 + 0$) ergeben.

Jetzt fragen Sie sich sicher, wozu das gut ist. Wenn Sie gerade einfach mal nicht an Zahlen denken, sondern an Schalter, dann kann man mit einer 1 »eingeschaltet« (oder zulässig) symbolisieren und mit einer 0 »ausgeschaltet« (oder unzulässig) darstellen. Wollten Sie beispielsweise ein kleines Rechtesystem aufbauen, dann könnten Sie mit dem Bit an der Stelle 1 symbolisieren, dass jemand das Recht hat, einen Artikel zu schreiben, wohingegen das Bit an der Stelle 0 dafür stünde, dass jemand lesen darf. Das heißt, wenn jemand 0b11 hat, dann dürfte er lesen und schreiben. Das hört sich zunächst umständlich an. Wenn Sie nun herausfinden wollten, was jemand darf, dann müssten Sie immer mit Integer-Werten vergleichen. Wenn jemand ein Recht `>1` und `<4` hätte, dann dürfte er lesen.

Das geht aber auch einfacher, wenn Sie den Bit-Operator `&` nutzen. Verknüpfen Sie mit ihm zwei Bitfolgen, dann gibt er an den Stellen, an denen Sie bei Bitfolgen eine 1 hatten, eine 1 zurück. Die folgenden Beispiele verdeutlichen das System:

0010 & 0011 = 0010
1111 & 1000 = 1000
0000 & 1111 = 0000

Mit diesem System können Sie jetzt also erkennen, an welcher Stelle ein Bit gesetzt ist. Konvertieren Sie das Ergebnis einer solchen Operation jetzt in einen Booleschen Wert, dann ergibt sich für 0 ein `false` und für alle Werte ungleich 0 ein `true`:

```
$recht_schreiben = 0b10;
$recht_lesen     = 0b01;

$user_rechte     = 0b01;

// Diese Operation ergibt 01 und wird nach true konvertiert
if (true == (bool) ($user_rechte & $recht_lesen))
{
   echo "Darf lesen";
}

// Diese Operation ergibt 00 und wird nach false konvertiert
if (true == (bool) ($user_rechte & $recht_schreiben))
{
   echo "Darf schreiben";
}
```

Dieses kleine Beispiel gibt `Darf lesen` aus. Auf diesem Weg kann man eine ganze Menge interessanter Spielereien basteln. Beachten Sie aber immer, dass eine solche Implementation schon etwas speziell und nicht für jeden verständlich ist.

Insbesondere möchte ich noch darauf hinweisen, dass »Zahlen« wie Telefonnummern oder Postleitzahlen auf keinen Fall als Integer gespeichert werden sollten, da sie oft mit einer Null beginnen. Auch wenn das Wort Postleitzahl vermuten lässt, dass es sich um eine Zahl handelt, so sollten Sie solche Werte immer als String ablegen. Legen Sie nur dann einen Wert als Zahl ab, wenn Sie damit rechnen wollen.

2.1.3 Float

Float-Werte sind Fließkommazahlen, sie unterstützen also einen Nachkommaanteil. Eine Unterscheidung zwischen Fließkommazahlen mit einfacher und doppelter Genauigkeit ist in PHP nicht vorgesehen. Auch wenn dieser Datentyp in PHP als `float` bezeichnet wird, so basiert er doch auch auf dem Datentyp `double` des verwendeten C-Compilers. Der verfügbare Wertebereich entspricht somit

auch dem, den das Betriebssystem bzw. der C-Compiler für `double` unterstützt. Typischerweise ist das der Bereich von 1,8e-308 bis 1,8e+308. Die Genauigkeit beträgt hierbei 14 Stellen. Auch wenn dieser Wertebereich schon recht groß ist, so kann es natürlich passieren, dass das Ergebnis einer Berechnung den gültigen Fließkomma-Wertebereich verlässt. Um dies prüfen zu können, wurden mit PHP 4.2.0 die Funktionen `is_finite()` und `is_infinite()` eingeführt. Die erste liefert ein `true` zurück, wenn die übergebene Fließkommazahl ein endlicher Wert ist. Wurde einer Fließkommavariablen ein Wert zugewiesen, der größer ist als der zulässige Wertebereich, gibt sie ein `false` zurück.

```
$ok = 1e + 200;
$zu_gross = $ok * $ok;

if (true == is_finite($ok))
{
   echo "Der Wert ist OK";//Wird ausgegeben
}

if (true == is_infinite($zu_gross))
{
   echo "Der Wert ist zu groß";//Wird auch ausgegeben
}
```

Vergleicht man die hier beschriebenen Eigenschaften der Datentypen Integer und Float, könnte man auf die Idee kommen, immer nur mit Fließkommazahlen zu arbeiten. Leider sind diese aber sehr ungenau, so dass man sie wenn möglich vermeiden sollte.

```
$float = 0.3;
$float = $float + 0.3;
$float = $float + 0.3;
$float = $float + 0.1;
echo '$float: '.$float. '<br />';
echo 'floor() liefert: '.floor($float);
```

Listing 2.1 Ungenauigkeit bei Fließkommaoperationen

So generiert dieses kleine Programm nicht, wie erwartet, `1` und `1` als Ausgabe, sondern:

```
$float:1
floor() liefert:0
```

Dieses Verhalten ist üblicherweise auch recht einfach mit einer `for`-Schleife nachzuweisen. So rechnet diese Schleife grundsätzlich korrekt:

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z