

Apr 20, 11 3:00

perl-true-false-HOWTO.txt

Page 1/2

HOWTO zu den Booleschen Wahrheitswerten in Perl
 (C) 2006 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
 OSTC GmbH, <http://www.ostc.de>
 \$Id: perl-true-false-HOWTO.txt,v 1.8 2011-04-19 15:03:59 tsbirn Exp \$

Dieses Dokument beschreibt die Logik der Booleschen WAHR/FALSCH-Werte in Perl.

Inhaltsverzeichnis

- 1) Boolesche Werte
- 2) Logische Operatoren
- 3) Short cut/circuit evaluation
- 4) "undef" versus definierte Werte

1) Boolesche Werte

Perl kennt keinen Booleschen Datentyp mit den Werten "TRUE" und "FALSE", sondern interpretiert JEDEN Wert (Skalar, Array, Hash, Referenz) als logischen Wert. Dabei ist exakt definiert, welche Werte als logisch FALSCH und welche als logisch WAHR interpretiert werden. Logisch FALSCH sind GENAU die folgenden 5 Werte:

Wert	Beschreibung
undef	Undefinierter Wert
" "	Leere Zeichenkette (natürlich auch ' ', qq//)
0	Zahl Null (auch -0 und 0.0)
"0"	Zeichenkette bestehend aus Zeichen Null (auch '0', Sonderfall!)
()	Leere Liste (natürlich auch qw//)

Sonderfall: Die Zeichenkette "0" wird auch als logisch FALSCH interpretiert, weil sie nicht von der Zahl 0 unterscheidbar ist, wenn Perl automatisch zwischen Strings und Zahlen hin- und herkonvertiert.

ALLE ANDEREN Werte werden als logisch WAHR interpretiert, insbesondere die folgenden:

Wert	Bedeutung
" "	Zeichenkette bestehend aus einem Leerzeichen
"\n"	Zeichenkette bestehend aus einem Newline
"\r"	Zeichenkette bestehend aus einem Carriage Return
" +0"	Zeichenkette bestehend aus zwei Zeichen
" -0"	Zeichenkette bestehend aus zwei Zeichen
"00"	Zeichenkette bestehend aus zwei Null-Zeichen
"0.0"	Zeichenkette bestehend aus drei Zeichen
-1	Negative Zahl
{}	Referenz auf leeren Hash
[]	Referenz auf leeres Array

2) Logische Operatoren

Die logischen Operatoren &&, and, ||, or, !, not und xor liefern angewendet auf Wahrheitswerte (oder andere Werte, die dann als Wahrheitswerte interpretiert werden) folgende Resultate:

Op	Alternativ	Bedeutung
A && B	A and B	Resultat WAHR wenn beide Operanden WAHR
A B	A or B	Resultat WAHR wenn einer der beiden Operanden WAHR
! A	not A	Resultat WAHR wenn Operand FALSCH (negieren)
	A xor B	Resultat WAHR wenn ein Operand WAHR und einer FALSCH

WAHR wird meist als "1" und FALSCH wird meist als "" (leere Zeichenkette) dargestellt. Letztlich liefern die logischen Operatoren aber immer den LETZTEN ausgewerteten Wert als Ergebnis zurück.

```
0 or "abc"      # -> "abc" = WAHR
1 and 234      # -> 234  = WAHR
9 and undef    # -> undef = FALSCH
0 or ()        # -> ()   = FALSCH
```

Der Vorrang von && und || liegt nach den Zuweisungsoperatoren (d.h. Klammern sind notwendig, wenn das Ergebnis zugewiesen werden soll). Der Vorrang von "and", "or", "not" und "xor" liegt NACH allen anderen Operatoren (auch nach ",", d.h. ist der geringstmögliche. Einfaches Austauschen von && und

"and" bzw. `||` und "or" bzw. `!` und "not" führt daher oft zu einer falschen Auswertungsreihenfolge!

3) Short cut/circuit evaluation

Logische Operatoren werden von links nach rechts ausgewertet (unter Berücksichtigung des Vorrangs). Ergibt sich dabei ein Zwischenergebnis, bei dem das Endergebnis des gesamten logischen Ausdrucks sofort feststeht, wird die Auswertung an dieser Stelle abgebrochen und der Rest des logischen Ausdrucks nicht mehr ausgewertet (sogenannte "Short cut evaluation", "Short circuit evaluation", verkürzte Auswertung). Dieses Verhalten ist keine Perl-Besonderheit, sondern ist in vielen Programmiersprachen (z.B. C, C++, PHP, Awk) zu beobachten.

Insbesondere wird eine Folge von UND-Verknüpfungen ("and", "&&") mit Gesamtergebnis FALSCH abgebrochen, sobald lx FALSCH als Zwischenergebnis vorkommt. Eine Folge von ODER-Verknüpfungen ("or", "||") wird mit Gesamtergebnis WAHR abgebrochen, sobald lx WAHR als Zwischenergebnis vorkommt.

Beispiel:

```
0 and print "Nicht ausgegeben";      # -> 0 und nichts ausgegeben
1 or  print "Auch nicht ausgegeben";  # -> 1 und nichts ausgegeben
0 or "abc" or 3.14                    # -> "abc"
1 and "" and 3.14                      # -> ""
```

4) "undef" versus definierte Werte

Nicht initialisierte Variablen haben in Perl den Wert "undef" und sind somit UNDEFINIERT (analog SQL-Wert "NULL"). ALLE anderen Werte sind DEFINIERT, diese können mit dem Operator "defined" vom Wert "undef" unterschieden werden.

Wert	WAHR	FALSCH	defined	!defined
undef	Nein	Ja	Nein	Ja
" "	Nein	Ja	Ja	Nein
0	Nein	Ja	Ja	Nein
"0"	Nein	Ja	Ja	Nein
()	Nein	Ja	Ja	Nein
" "	Ja	Nein	Ja	Nein
"\n"	Ja	Nein	Ja	Nein
"\r"	Ja	Nein	Ja	Nein
" +0"	Ja	Nein	Ja	Nein
" -0"	Ja	Nein	Ja	Nein
"00"	Ja	Nein	Ja	Nein
"0.0"	Ja	Nein	Ja	Nein
-1	Ja	Nein	Ja	Nein
{ }	Ja	Nein	Ja	Nein
[]	Ja	Nein	Ja	Nein

Folgende Tests prüfen, ob der Wert einer Variablen \$var DEFINIERT bzw. UNDEFINIERT ist:

```
if (defined $var) { ... }
if (not defined $var) { ... }
if (! defined $var) { ... }
```

Folgende Tests prüfen, ob der Wert einer Variablen \$var WAHR bzw. FALSCH ist:

```
if ($var) { ... }
if (not $var) { ... }
if (! $var) { ... }
```