Search

Dutch PHP Conference 2024

Keyboard Shortcuts

?
: This help

j
: Next menu item

k
: Previous menu item

g p
: Previous man page

g n
: Next man page

G
: Scroll to bottom

g g
: Scroll to top

g h
: Goto homepage

g s
: Goto search
(current page)

/
: Focus search box

Числа с плавающей точкой »
« Логические значения

- Руководство по PHP
- Справочник языка
- Типы

Change language: Russian

Целые числа

int — это число из множества ℤ = {..., -2, -1, 0, 1, 2, ...}.

Смотрите также

- Числа с плавающей точкой
- Вычисления над числами с произвольной точностью BCMath
- Вычисления над целыми числами с произвольной точностью (GNU Multiple Precision)

Синтаксис

Целые числа (int) могут быть указаны в десятичной (основание 10), шестнадцатеричной (основание 16), восьмеричной (основание 8) или двоичной (основание 2) системе счисления. Для задания отрицательных целых (int) используют оператор отрицания

Для записи в восьмеричной системе счисления перед числом ставят `0` (ноль). Начиная с PHP 8.1.0 восьмеричной нотации также может предшествовать `0o` или `0O`. Для записи в шестнадцатеричной системе счисления необходимо поставить перед числом `0x`. Для записи в двоичной системе счисления необходимо поставить перед числом `0b`

Начиная с PHP 7.4.0 целочисленные литералы могут содержать подчёркивания (_) между цифрами для лучшей читаемости литералов. Эти подчёркивания удаляются сканером PHP.

Пример #1 Целые числа

```php
<?php

$a = 1234; // десятичное число
$a = 0123; // восьмеричное число (эквивалентно 83 в десятичной системе)
$a = 0o123; // восьмеричное число (начиная с PHP 8.1.0)
$a = 0x1A; // шестнадцатеричное число (эквивалентно 26 в десятичной системе)
$a = 0b11111111; // двоичное число (эквивалентно 255 в десятичной системе)
$a = 1_234_567; // десятичное число (с PHP 7.4.0)
?>
```

Формально, структура целых чисел int принята в PHP 8.1.0 (ранее не допускались восьмеричные префиксы `0o` или `0O`, а до PHP 7.4.0 не допускалось подчёркивание):

```
десятичные         : [1-9][0-9]*(_[0-9]+)*
                   | 0

шестнадцатеричные  : 0[xX][0-9a-fA-F]+(_[0-9a-fA-F]+)*

восьмеричные       : 0[oO]?[0-7]+(_[0-7]+)*

двоичные           : 0[bB][01]+(_[01]+)*

целые              : десятичные
                   | шестнадцатеричные
                   | восьмеричные
                   | двоичные
```

Размер типа int зависит от платформы, хотя, как правило, максимальное значение примерно равно 2 миллиардам (это 32-битное знаковое). 64-битные платформы обычно имеют максимальное значение около 9E18. PHP не поддерживает беззнаковые целые числа (int). Размер int может быть определён с помощью константы **`PHP_INT_SIZE`**, максимальное значение — с помощью константы **`PHP_INT_MAX`**, а с помощью константы **`PHP_INT_MIN`** можно определить минимальное значение.

Переполнение целых чисел

Если PHP обнаружил, что число превышает размер типа int, он будет интерпретировать его в качестве float. Аналогично, если результат операции лежит за границами типа int, он будет преобразован во float.

Пример #2 Переполнение целых на 32-битных системах

```php
<?php

$large_number = 2147483647;
var_dump($large_number); // int(2147483647)

$large_number = 2147483648;
var_dump($large_number); // float(2147483648)

$million = 1000000;
$large_number = 50000 * $million;
var_dump($large_number); // float(50000000000)
?>
```

Пример #3 Переполнение целых на 64-битных системах

```php
<?php
$large_number = 9223372036854775807;
var_dump($large_number); // int(9223372036854775807)

$large_number = 9223372036854775808;
var_dump($large_number); // float(9.2233720368548E+18)

$million = 1000000;
$large_number = 50000000000000 * $million;
var_dump($large_number); // float(5.0E+19)
?>
```

В PHP нет оператора целочисленного (int) деления, для этого существует функция intdiv(). Результатом `1/2` будет float `0.5`. Если привести значение к int, оно будет округлено вниз, то есть будет отброшена дробная часть числа. Для большего контроля над округлением используют функцию round().

```php
<?php

var_dump(25/7); // float(3.5714285714286)
var_dump((int) (25/7)); // int(3)
var_dump(round(25/7)); // float(4)
?>
```

Преобразование в целое

Для явного преобразования в тип int используют приведение `(int)` или `(integer)`. Однако в большинстве случаев в приведении типа нет необходимости, так как значение будет автоматически преобразовано, если оператор, функция или управляющая структура требует аргумент типа int. Значение также может быть преобразовано в тип int функцией intval().

Если значением с типом resource преобразовывается в значением с типом int, то результатом будет уникальный номер ресурса, привязанный к resource во время исполнения PHP программы.

Смотрите также: Манипуляции с типами.

Из логического типа

Логическое значение **`false`** преобразовывается в `0` (ноль), а **`true`** — в `1` (единицу).

Из чисел с плавающей точкой

При преобразовании из float в int число будет округлено *вниз*. Начиная с PHP 8.1.0 при неявном преобразовании неинтегрального числа с плавающей точкой (float) в целое число (int), которое теряет точность, выдаётся уведомление об устаревании.

```php
<?php

function foo($value): int {
return $value;
```

```
}

var_dump(foo(8.1)); // "Deprecated: Implicit conversion from float 8.1 to int loses precision" начиная с PHP 8.1.0
var_dump(foo(8.1)); // 8 до PHP 8.1.0
var_dump(foo(8.0)); // 8 в обоих случаях

var_dump((int)8.1); // 8 в обоих случаях
var_dump(intval(8.1)); // 8 в обоих случаях
?>
```

Если число с плавающей точкой превышает размеры int (обычно `± 2.15e+9 = 2^31` на 32-битных системах и `± 9.22e+18 = 2^63` на 64-битных системах, результат будет неопределённым, так как float не имеет достаточной точности, чтобы вернуть верный результат в виде целого числа (int). В этом случае не будет выведено ни предупреждения, ни даже замечания!

> **Замечание**:
>
> Значения NaN и Infinity при приведении к int становятся равными нулю, вместо неопределённого значения в зависимости от платформы.

Внимание

Нельзя приводить неизвестную дробь к типу int, так как это иногда может дать неожиданные результаты.

```
<?php
echo (int) ( (0.1+0.7) * 10 ); // выводит 7!
?>
```

Подробнее об этом рассказано в предупреждении о точности чисел с плавающей точкой.

Из строк

Если строка содержит число или ведущую числовую последовательность, тогда она будет преобразована в соответствующее целое число, в противном случае она преобразуется в ноль (`0`).

Из NULL

Значение **`null`** всегда преобразовывается в ноль (`0`).

Из других типов

Предостережение

Для других типов поведение преобразования в int не определено. *Не нужно* полагаться на наблюдаемое ранее поведение, так как оно может измениться без предупреждения.

＋add a note

User Contributed Notes 10 notes

up
down
138
php at richardneill dot org ¶
10 years ago
A leading zero in a numeric literal means "this is octal". But don't be confused: a leading zero in a string does not.
Thus:
```
$x = 0123; // 83
$y = "0123" + 0 // 123
```
up
down
55
d_n at NOSPAM dot Loryx dot com ¶
16 years ago
Here are some tricks to convert from a "dotted" IP address to a LONG int, and backwards. This is very useful because

accessing an IP addy in a database table is very much faster if it's stored as a BIGINT rather than in characters.

IP to BIGINT:

```
<?php
$ipArr = explode('.',$_SERVER['REMOTE_ADDR']);
$ip = $ipArr[0] * 0x1000000
+ $ipArr[1] * 0x10000
+ $ipArr[2] * 0x100
+ $ipArr[3]
;
?>
```

IP as BIGINT read from db back to dotted form:

Keep in mind, PHP integer operators are INTEGER -- not long. Also, since there is no integer divide in PHP, we save a couple of S-L-O-W floor (<division>)'s by doing bitshifts. We must use floor(/) for $ipArr[0] because though $ipVal is stored as a long value, $ipVal >> 24 will operate on a truncated, integer value of $ipVal! $ipVint is, however, a nice integer, so
we can enjoy the bitshifts.

```
<?php
$ipVal = $row['client_IP'];
$ipArr = array(0 =>
floor( $ipVal / 0x1000000) );
$ipVint = $ipVal-($ipArr[0]*0x1000000); // for clarity
$ipArr[1] = ($ipVint & 0xFF0000) >> 16;
$ipArr[2] = ($ipVint & 0xFF00 ) >> 8;
$ipArr[3] = $ipVint & 0xFF;
$ipDotted = implode('.', $ipArr);
?>
```

up
down
25
dhairya lakhera ¶
6 years ago

Question :

```
var_dump((int) 010); //Output 8

var_dump((int) "010"); //output 10
```

First one is octal notation so the output is correct. But what about the when converting "010" to integer. it should be also output 8 ?

Answer :

Casting to an integer using (int) will always cast to the default base, which is 10.

Casting a string to a number this way does not take into account the many ways of formatting an integer value in PHP (leading zero for base 8, leading "0x" for base 16, leading "0b" for base 2). It will simply look at the first characters in a string and convert them to a base 10 integer. Leading zeroes will be stripped off because they have no meaning in numerical values, so you will end up with the decimal value 10 for (int)"010".

Converting an integer value between bases using (int)010 will take into account the various ways of formatting an integer. A leading zero like in 010 means the number is in octal notation, using (int)010 will convert it to the decimal value 8 in base 10.

This is similar to how you use 0x10 to write in hexadecimal (base 16) notation. Using (int)0x10 will convert that to the base 10 decimal value 16, whereas using (int)"0x10" will end up with the decimal value 0: since the "x" is not a numerical value, anything after that will be ignored.

If you want to interpret the string "010" as an octal value, you need to instruct PHP to do so. intval("010", 8) will

interpret the number in base 8 instead of the default base 10, and you will end up with the decimal value 8. You could also use octdec("010") to convert the octal string to the decimal value 8. Another option is to use base_convert("010", 8, 10) to explicitly convert the number "010" from base 8 to base 10, however this function will return the string "8" instead of the integer 8.

Casting a string to an integer follows the same the logic used by the intval function:

Returns the integer value of var, using the specified base for the conversion (the default is base 10).
intval allows specifying a different base as the second argument, whereas a straight cast operation does not, so using (int) will always treat a string as being in base 10.

```
php > var_export((int) "010");
10
php > var_export(intval("010"));
10
php > var_export(intval("010", 8));
8
```

up
down
10

ganlvtech at qq dot com ¶
3 years ago

Be aware of float to int cast overflow

```
<?php

// You may expected these
var_dump(0x7fffffffffffffff); // int(9223372036854775807)
var_dump(0x7fffffffffffffff + 1); // float(9.2233720368548E+18)
var_dump((int)(0x7fffffffffffffff + 1)); // int(9223372036854775807)
var_dump(0x7fffffffffffffff + 1 > 0); // bool(true)
var_dump((int)(0x7fffffffffffffff + 1) > 0); // bool(true)
var_dump((int)'9223372036854775807'); // int(9223372036854775807)
var_dump(9223372036854775808); // float(9.2233720368548E+18)
var_dump((int)'9223372036854775808'); // int(9223372036854775807)
var_dump((int)9223372036854775808); // int(9223372036854775807)

// But actually, it likes these
var_dump(0x7fffffffffffffff); // int(9223372036854775807)
var_dump(0x7fffffffffffffff + 1); // float(9.2233720368548E+18)
var_dump((int)(0x7fffffffffffffff + 1)); // int(-9223372036854775808) <-----
var_dump(0x7fffffffffffffff + 1 > 0); // bool(true)
var_dump((int)(0x7fffffffffffffff + 1) > 0); // bool(false) <-----
var_dump((int)'9223372036854775807'); // int(9223372036854775807)
var_dump(9223372036854775808); // float(9.2233720368548E+18)
var_dump((int)'9223372036854775808'); // int(9223372036854775807)
var_dump((int)9223372036854775808); // int(-9223372036854775808) <-----

?>
```

These overflows are dangerous when you try to compare it with zero, or substract it from another value (e.g. money).

up
down
18

egwayjen at gmail dot com ¶
6 years ago

"There is no integer division operator in PHP". But since PHP 7, there is the intdiv function.

up
down
10

litbai ¶
7 years ago

```
<?php
$ipArr = explode('.', $ipString);
$ipVal = ($ipArr[0] << 24)
+ ($ipArr[1] << 16)
+ ($ipArr[2] << 8)
+ $ipArr[3]
;
?>
```

1. the priority of bit op is lower than '+',so there should be brackets.
2. there is no unsighed int in PHP, if you use 32 bit version, the code above will get negative result when the first position of IP string greater than 127.
3. what the code actually do is calculate the integer value of transformed 32 binary bit from IP string.

up
down
13

Anonymous ¶

9 years ago

Converting to an integer works only if the input begins with a number

```
(int) "5txt" // will output the integer 5
(int) "before5txt" // will output the integer 0
(int) "53txt" // will output the integer 53
(int) "53txt534text" // will output the integer 53
```

up
down
17

rustamabd@gmail-you-know-what ¶

17 years ago

Be careful with using the modulo operation on big numbers, it will cast a float argument to an int and may return wrong results. For example:

```
<?php
$i = 6887129852;
echo "i=$i\n";
echo "i%36=".($i%36)."\n";
echo "alternative i%36=".($i-floor($i/36)*36)."\n";
?>
```

Will output:

```
i=6.88713E+009
i%36=-24
alternative i%36=20
```

up
down
14

Anonymous ¶

16 years ago

To force the correct usage of 32-bit unsigned integer in some functions, just add '+0' just before processing them.

```
for example
echo(dechex("2724838310"));
will print '7FFFFFFF'
but it should print 'A269BBA6'

When adding '+0' php will handle the 32bit unsigned integer
correctly
echo(dechex("2724838310"+0));
will print 'A269BBA6'
```

up
down
4

Jacek ¶

16 years ago

On 64 bits machines max integer value is 0x7fffffffffffffff (9 223 372 036 854 775 807).

＋add a note

- Copyright © 2001-2024 The PHP Group
- My PHP.net
- Contact
- Other PHP.net sites
- Privacy policy