Hochschule Darmstadt
Fachbereich Informatik

Entwicklung webbasierter Anwendungen

h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi

FACHBEREICH INFORMATIK

Hochschule Darmstadt
Fachbereich Informatik

PHP 7

h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi

FACHBEREICH INFORMATIK

Releasekalender

Currently Supported Versions

Branch	Initial Release	Active Support Until	Security Support Until
5.5	20 Jun 2013	10 Jul 2015	10 Jul 2016
5.6	28 Aug 2014	28 Aug 2016	28 Aug 2017
7.0	3 Dec 2015	3 Dec 2017	3 Dec 2018

http://php.net/supported-versions.php

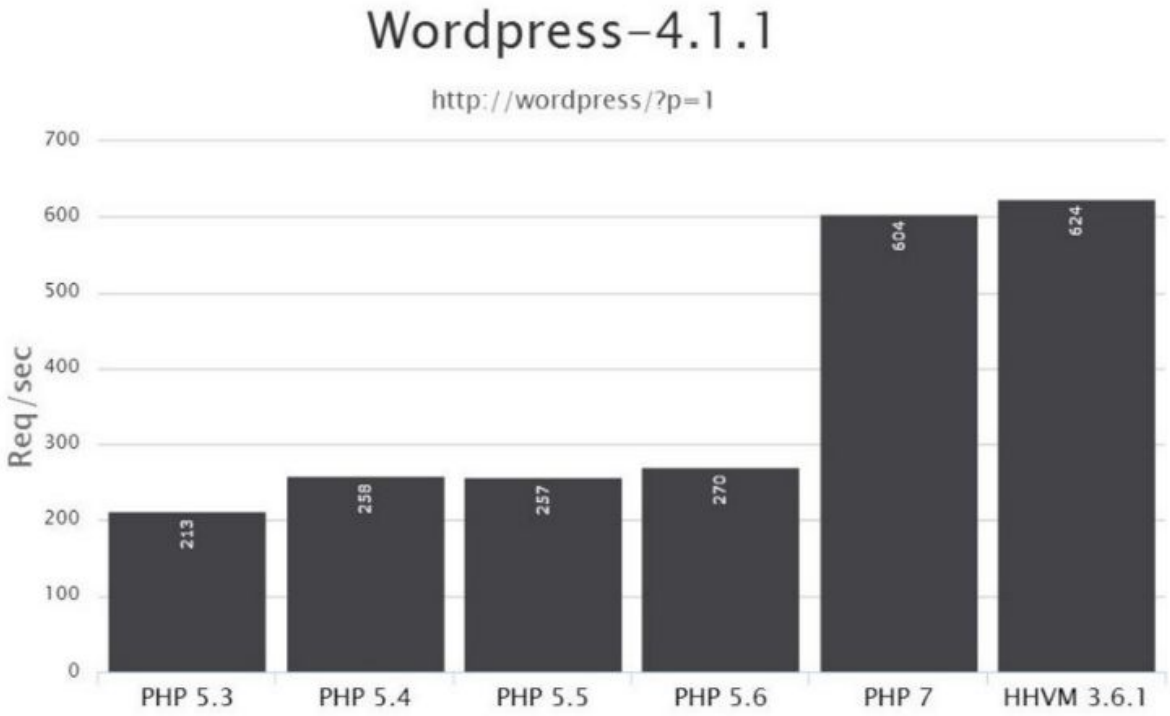
Historie

- Urpsrünglicher Plan für PHP **6**: Die PHP Engine sollte auf Unicode (UTF-16) portiert werden.
- Im März 2010 nach mehreren Jahren Diskussion PHP 6 Entwicklung eingestellt
- Direkter Sprung auf PHP 7

https://schlueters.de/blog/archives/128-Future-of-PHP-6.html

Big Picture

- PHP typisch: Kaum Änderungen, die die Abwärtskompatibilität brechen
- guter PHP 5 Code läuft in aller Regel auch unter PHP 7
- Performance:



https://kinsta.com/blog/hhvm-vs-php-7/

Neue Features: Null coalescing Operator

```php
<?php
$test = $not_there ?? "bar";
echo $test; //bar
```

VS

```php
<?php
$there = "foo";
$test = $there ?? "bar";
echo $test; //foo
```

Äquivalent zu

```php
<?php
$test = isset($there) ? $there : "bar";
```

Neue Features: “Spaceship” Operator

```php
<?php
// Numbers
echo 5 <=> 5;      // 0
echo 6 <=> 5;      // 1
echo 5 <=> 6;      // -1

//even with type juggling:
echo "5.0" <=> 5; // 0

// Strings
echo "a" <=> "a"; // 0
echo "a" <=> "b"; // -1
echo "b" <=> "a"; // 1
```

Drei-Wege-Vergleich, ähnlich zu “strcmp” in C

http://www.cplusplus.com/reference/cstring/strcmp/

Neue Features: Array Konstanten

```php
<?php
define('ANIMALS', [
    'dog',
    'cat',
    'bird'
]);

echo ANIMALS[1]; // cat
```

Neue Features: Exception Handling

- Aufruf einer Methode auf einem Null-Objekt, ohne Exception Handling

```
1 <?php
2 $foo = null;
3 $foo->methodCall();
```

Crash mit Fatal Error, Abarbeitung des Skripts stoppt:

C:\WINDOWS\system32\cmd.exe

```
D:\xampp\htdocs\php7>php exc.php
PHP Fatal error:  Uncaught Error: Call to a member function methodCall() on null in D:\xampp\htdocs\php7\exc.php:3
Stack trace:
#0 {main}
  thrown in D:\xampp\htdocs\php7\exc.php on line 3
```

Neue Features: Exception Handling

- In PHP 7 möglich:

```php
<?php
$foo = null;

try {
    $foo->methodCall();
}
catch (Error $ex)
{
    echo "Fehler: " . $ex->getMessage();
}
```

C:\WINDOWS\system32\cmd.exe

```
D:\xampp\htdocs\php7>php exc.php
Fehler: Call to a member function methodCall() on null
D:\xampp\htdocs\php7>
```

Neue Features: Exception Handling

- Achtung: “Error” != “Exception”.
- Fangen als “Exception” nicht möglich, um Abwärtskompatibilität zu erhalten

```php
<?php
$foo = null;

try {
    $foo->methodCall();
}
catch (Exception $ex)
{
    echo "Fehler: " . $ex->getMessage();
}
```

C:\WINDOWS\system32\cmd.exe

```
D:\xampp\htdocs\php7>php exc.php
PHP Fatal error:  Uncaught Error: Call to a member function methodCall() on null in D:\xampp\htdocs\php7\exc.php:5
Stack trace:
#0 {main}
  thrown in D:\xampp\htdocs\php7\exc.php on line 5
```

Neue Features: Exception Handling

- Neue Exception-Struktur: “Error” und “Exception” leiten beide von “Throwable” ab

```
\Throwable
├── \Exception (implements \Throwable)
│   ├── \LogicException (extends \Exception)
│   │   ├── \BadFunctionCallException (extends \LogicException)
│   │   │   └── \BadMethodCallException (extends \BadFunctionCallException)
│   │   ├── \DomainException (extends \LogicException)
│   │   ├── \InvalidArgumentException (extends \LogicException)
│   │   ├── \LengthException (extends \LogicException)
│   │   └── \OutOfRangeException (extends \LogicException)
│   └── \RuntimeException (extends \Exception)
│       ├── \OutOfBoundsException (extends \RuntimeException)
│       ├── \OverflowException (extends \RuntimeException)
│       ├── \RangeException (extends \RuntimeException)
│       ├── \UnderflowException (extends \RuntimeException)
│       └── \UnexpectedValueException (extends \RuntimeException)
└── \Error (implements \Throwable)
    ├── \AssertionError (extends \Error)
    ├── \ParseError (extends \Error)
    └── \TypeError (extends \Error)
```

https://www.daveyshafik.com/archives/69237-an-exceptional-change-in-php-7-0.html

Neue Features: Exception Handling

- Mit catch (Throwable) lassen sich also Exceptions und Errors fangen

```php
<?php
$foo = null;

try {
    $foo->methodCall();
}
catch (Throwable $ex)
{
    echo "Fehler: " . $ex->getMessage() . "\n";
}

try {
    throw new Exception("test");
}
catch (Throwable $ex)
{
    echo "Caught: " . $ex->getMessage() . "\n";
}
```

C:\WINDOWS\system32\cmd.exe

```
D:\xampp\htdocs\php7>php exc.php
Fehler: Call to a member function methodCall() on null
Caught: test
```

Neue Features: Anonyme Klassen

```php
<?php
interface Logger {
    public function log($msg);
}

class Application {
    public $logger;

    public function __construct(Logger $logger) {
        $this->logger = $logger;
    }
}

$app = new Application(new class implements Logger {
    public function log($msg) {
        echo $msg;
    }
});

$app->logger->log("Test!");
```

C:\WINDOWS\system32\cmd.exe

```
D:\xampp\htdocs\php7>php logger.php
Test!
D:\xampp\htdocs\php7>
```

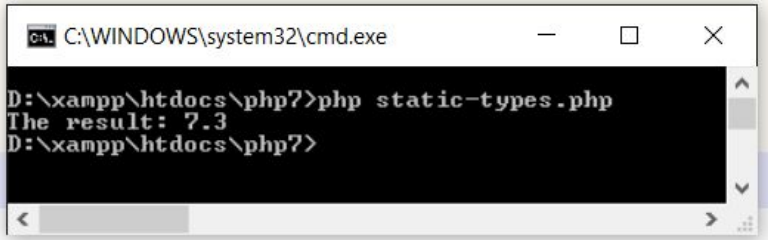
Neue Features: Strict Types

Meistdiskutiertes und wahrscheinlich relevantes PHP 7 Feature.

Neue Features: Strict Types

```php
<?php
function calc (int $num1, float $num2, string $text) : string
{
    return $text . ": " . ($num1 + $num2);
}

echo calc(3, 4.3, "The result");
```



Vor PHP 7 waren Typehints nur für Objekte und Arrays möglich. Nun auch für primitive Datentypen.

Auch der Rückgabetyp von Funktionen / Methoden kann vorgegeben werden!

Neue Features: Strict Types

```php
<?php
function calc (int $num1, float $num2, string $text) : string
{
    return $text . ": " . ($num1 + $num2);
}

echo calc(4.7, 4, 123);
```

C:\WINDOWS\system32\cmd.exe

```
D:\xampp\htdocs\php7>php static-types.php
123: 8
D:\xampp\htdocs\php7>
```

Unerwartete Parameter: Casting am Beispiel:

- int $num1 -> float 4.7 wird zu int 4
- float $num2 -> int 4 wird zu float 4.0
- string $text -> int 123 wird zu string “123”

Neue Features: Strict Types

```
1  <?php
2  function calc (int $num1, float $num2, string $text) : int
3  {
4      return $text . ": " . ($num1 + $num2);
5  }
6
7  echo calc("4strings", 4.0, 123);
8
```

C:\WINDOWS\system32\cmd.exe

```
D:\xampp\htdocs\php7>php static-types.php
PHP Notice:  A non well formed numeric value encountered in D:\xampp\htdocs\php7\static-types.php on line 2

Notice: A non well formed numeric value encountered in D:\xampp\htdocs\php7\static-types.php on line 2
PHP Notice:  A non well formed numeric value encountered in D:\xampp\htdocs\php7\static-types.php on line 4

Notice: A non well formed numeric value encountered in D:\xampp\htdocs\php7\static-types.php on line 4
123
D:\xampp\htdocs\php7>
```

Ausgabe von “Notices”, bspw. bei Übergabe eines Strings statt eines Integers

Neue Features: Strict Types

Abhilfe schafft das Verwenden von “strict types”. Ausgabe eines Fatal Errors bei Typkonflikten.

```php
<?php
declare(strict_types=1);

function calc (int $num1, float $num2, string $text) : string
{
    return $text . ": " . ($num1 + $num2);
}

echo calc(4, 4.4, 1234);
```

C:\WINDOWS\system32\cmd.exe

```
D:\xampp\htdocs\php7>php static-types.php
PHP Fatal error:  Uncaught TypeError: Argument 3 passed to calc() must be of the type string, integer given,
types.php:4
```

Neue Features: Strict Types

Zulässige Konvertierung trotz “strict_types”: int kann statt float übergeben werden. Im Beispiel wird int 5 für float $num2 akzeptiert.

```php
<?php
declare(strict_types=1);

function calc (int $num1, float $num2, string $text) : string
{
    return $text . ": " . ($num1 + $num2);
}

echo calc(4, 5, "The result");
```

C:\WINDOWS\system32\cmd.exe

```
D:\xampp\htdocs\php7>php static-types.php
The result: 9
```

Backwards Incompatible Changes

- Auflistung: http://php.net/manual/de/migration70.incompatible.php
- Praktisch: Keine relevanten Brüche, nur relativ “exotische” Konstrukte betroffen
- Auswertungsreihenfolge:

Old and new evaluation of indirect expressions

Expression	PHP 5 interpretation	PHP 7 interpretation
`$$foo['bar']['baz']`	`${$foo['bar']['baz']}`	`($$foo)['bar']['baz']`
`$foo->$bar['baz']`	`$foo->{$bar['baz']}`	`($foo->$bar)['baz']`
`$foo->$bar['baz']()`	`$foo->{$bar['baz']}()`	`($foo->$bar)['baz']()`
`Foo::$bar['baz']()`	`Foo::{$bar['baz']}()`	`(Foo::$bar)['baz']()`

Deprecated Changes

- Auflistung: http://php.net/manual/de/migration70.deprecated.php
- “Deprecated Changes” werfen eine Deprecation Notice und werden in der nächsten “nicht minor” PHP Version entfernt
- Sollten jetzt bereits entfernt werden!

Deprecated Changes

- “PHP 4 Style” Konstruktoren sollten nicht mehr verwendet werden, nur noch “__construct”:

```php
<?php
class foo {
    function foo() {
        echo 'I am the constructor';
    }
}
```

Deprecated: Methods with the same name as their class will not be constructors in a future version of PHP; foo has a deprecated constructor in example.php on line 3

Deprecated Changes

- Nicht-statische Methoden sollten nicht mehr statisch aufgerufen werden

```
<?php
class foo {
    function bar() {
        echo 'I am not static!';
    }
}

foo::bar();
```

Deprecated: Non-static method foo::bar() should not be called statically in - on line 8

I am not static!