

PHP5

Einstieg und Änderungen in PHP5

10.11.2005

Alexander Lais

alexander.lais@gmx.de

Fahrplan

1. *Was ist PHP?*
2. Grundlegende Syntax
3. Ein- und Ausgabe
4. Objektorientierung PHP4 vs. PHP5
5. Neues MySQL Interface
6. Sicherheitsaspekte
7. (PHP || !PHP)?
8. Lesenswertes

Was ist PHP?

Abkürzung **PHP**

PHP: Hypertext Preprocessor

War mal:

Personal HomePage Tools

→ stark auf dynamische
Webanwendungen ausgerichtet

Was ist PHP?

- **Programmiersprache**

- Scriptsprache
- „Interkompiliert“
- Einfach zu verstehen
- Gewöhnungsbedürftig (Funktionsnamen ohne Konvention)
- Universell
- Erweiterbar
- Ansammlung von Bibliotheken

- **Laufzeitumgebung**

- Portabel (Unix, Windows, etc.)
- Läuft auf Console, als CGI und Modul
- Durch kommerzielle Plugins schneller
- mit Encodern sind closed-source Scripts möglich

Lebenszyklus eines PHP Scripts

1. Anfrage erhalten

2. Scripts vorkompilieren

3. Arbeit machen

- Session deserialisieren
- Objekte und DB-Verbindungen aufbauen
- Ausgaben an HTTP Server

4. Script beenden

- Speicher freigeben, Verbindungen trennen
- Session serialisieren

Fahrplan

1. Was ist PHP?
2. *Grundlegende Syntax*
3. Ein- und Ausgabe
4. Objektorientierung PHP4 vs. PHP5
5. Neues MySQL Interface
6. Sicherheitsaspekte
7. (PHP || !PHP)?
8. Lesenswertes

Grundlegende Syntax

PHP-Block (immer unterstützt):

```
<?php
    // php code here
?>
```

Kurzformen (kann aber per config deaktiviert werden):

```
<?=$var?> entspricht <?php echo $var; ?>
```

"Short Tag"

```
<?
    // php code here
?>
```

ASP Style (kann per config aktiviert werden):

```
<%
    // php code here
%>
```

PHP in HTML

```
<html>
<head><title>PHP5</title></head>
<body>
Der Server sagt, es ist gerade <?php echo time()?>
    Sekunden nach dem 01.01.1970.<br />
<?php
    if(isset($_GET['username'])) {
        echo "Hallo ", htmlentities($_GET['username']);
    }
?>
</body>
</html>
```

Strings

```
<?php
// Einfache Variablen
$hello = 'Hello';
$php = strtoupper('php');
$five = 4 + '1 (eins)';
$world = 'World';
echo 'Hello PHP5 World';
echo $hello, $php, $five, $world;
// Unterschied "" und ''
echo "$hello $php" . $five . ' $world';
?>
```

Ausgabe:

```
Hello PHP5 World
HelloPHP5World
Hello PHP5 $world
```

Arrays

```
<?php
// Arrays
$arr = Array();
$arr[0] = 'foo';
$arr['associative'] = 'bar';
$arr[] = 'kommt an index 1';
var_dump($arr);
?>
```

Ausgabe von `var_dump()`:

```
array(3) {
    [0]=> string(3) "foo"
    ["associative"]=> string(3) "bar"
    [1]=> string(16) "kommt an index 1"
}
```

Simple Objekte

```
<?php
/* Objekte...
   in PHP4 anders geschriebene Arrays
   ggf. mit Methoden */
$obj->attribute = 'php5';
$obj->{$obj->attribute} = 'evil';
echo $obj->php5;
?>
```

Ausgabe: evil

Flusskontrolle: if

```
<?php
# Flusskontrolle, if
if(!$expression) {
    // do something
} elseif($anotherexpression) {
    // do something else
} else {
    // do the default
}
?>
```

Flusskontrolle: switch

```
<?php
# Flusskontrolle, switch
switch($expr) {
    case 1: // do 1
        break;
    case 'einszwo': // do 'einszwo'
        break;
    default: // do default
}
?>
```

Flusskontrolle: Schleifen

```
<?php
# Flusskontrolle, Schleifen
$i = 10;
while($i > 2) {
    echo --$i . "\n"; // nicht '\n'
}
for($j = 2; $j < 10; $j++) {
    echo $j;
}
do {
    // something
} while(true);
?>
```

Flusskontrolle: foreach

```
<?php
# Flusskontrolle, foreach
# Jedes Tupel (Schlüssel/Wertepaar)
# eines assoziativen Arrays durchlaufen

foreach($array as $key => $value) {
    echo "Key: '$key', Value: '$value'\n";
}

/* seit PHP5 auch für Klassen, die Iterator
implementieren */

foreach($object as $key => $value) {
    // iterate here
}

?>
```

Böse Sachen

```
<?php
// Ausführen dynamischen Codes
// Langsam und potentiell gefährlich
eval("echo 'hallo';");

// Systemaufrufe (Shell)
$output = `ls -l`;

// Variable Variablen
$var = 'blubberdiblubb';
$blubberdiblubb = 'Hallo';
echo $$var; // Ausgabe: Hallo
echo "${$var}"; // Ausgabe: Hallo
?>
```

Fahrplan

1. Was ist PHP?
2. Grundlegende Syntax
3. *Ein- und Ausgabe*
4. Objektorientierung PHP4 vs. PHP5
5. Neues MySQL Interface
6. Sicherheitsaspekte
7. (PHP || !PHP)?
8. Lesenswertes

Ein- und Ausgabe

- HTTP Spezifisch (superglobals)
 - `$_GET`, URL Argumente
 - `$_POST`, Formular-Argumente
 - `$_COOKIE`, Cookie-Variablen
 - `$_FILES`, Hochgeladene Dateien
 - `$_SESSION`, Sitzungsdaten, serverseitig
 - `$_REQUEST`, GET + POST + COOKIE
- Umgebung
 - `$_SERVER`, Informationen vom HTTP Server
 - `$_ENV`, Umgebungsvariablen

\$_GET

HTTP GET Request:

Übergabe von Variablen über die URL

```
http://example.com/script.php?var1=value1&var2=value2
```

```
$_GET['var1'] == 'value1'
```

```
$_GET['var2'] == 'value2'
```

Achtung: Übergebene Werte validieren!

```
http://example.com/load.php?file=../../../../etc/passwd
```

HTTP POST Request:

Aufruf mit Formulardaten

```
<form action="script.php" method="post">  
  <input type="text" name="userid" />  
  <input type="password" name="password" />  
  <input type="submit" name="login"  
    value="Log in" />  
</form>
```

Daten:

```
$_POST['userid'] == 'hanswurst'  
$_POST['password'] == 'hrglbrmf'  
$_POST['login'] == 'Log in'
```

\$_SESSION

- Aufrufübergreifende Variablen
- Identifizierung der Session über Session-ID
- Session ID als GET/POST/COOKIE-Variable mit definiertem Namen
- (Re)aktivieren mit `session_start()`
- Automatische (De-)Serialisierung der Daten zwischen Aufrufen
- Seit PHP5: Objekte werden vollständig wiederbelebt, nicht nur die Daten

Register Globals

- `register_globals`
 - PHP registriert GET, POST, COOKIE und SESSION-Daten in lokalen Variablen
 - Quelle der Daten unbestimmt
 - Kann zu sehr schweren Lücken führen
 - falsche Annahmen über Datenherkunft
 - Vertrauen in dynamisch initialisierte Variablen
 - seit 4.2.0 standardmäßig "off"

Immer für `register_globals=off`
programmieren!

Magic Quotes

- Magic Quotes

- In GET-, POST- und COOKIE-Daten werden Anführungsstriche automatisch maskiert
- Grundgedanke: Direktes einfügen z.B. in Queries oder HTML-Attribute:

```
INSERT INTO bla VALUES ('$blubb')
```

Bsp, `example.php?var=beck's` wird zu:

```
INSERT INTO bla VALUES ('beck\'s')
```

Nicht nur auf Magic Quotes vertrauen.
Eingaben immer validieren!

Magic Quotes

Mit Magic Quotes umgehen:

- `get_magic_quotes_gpc()`: prüfen
- `addslashes()`: manuell escapen
- `stripslashes()`: manuell unescapen

Magic Quotes für Daten aus allen externen Quellen:

- `set_magic_quotes_runtime()`: an/aus
- `get_magic_quotes_runtime()`: prüfen

Ausgabe

- Generell nur durch echo / print
- Für die Trennung von Logik und Darstellung gibt es Templatesysteme
 - Smarty
 - PHPTemplate

Fahrplan

1. Was ist PHP?
2. Grundlegende Syntax
3. Ein- und Ausgabe
4. *Objektorientierung PHP4 vs. PHP5*
5. Neues MySQL Interface
6. Sicherheitsaspekte
7. (PHP || !PHP)?
8. Lesenswertes

Objektorientierung PHP4 vs. PHP5

Neu in PHP5

- Zugriffskontrolle (abstract, private, ...)
- Interfaces
- Type Hinting
- Einheitliche Konstruktoren
- Neue "Magic Functions"
- Objektdereferenzierung
- PHP Standard Library
- Exception Handling

Zugriffskontrolle

```
<?php
class Testclass {
    private $var;
    protected function printme() {
        echo $var;
    }
}

$tc = new Testclass;

// Zugriff nicht gestattet
$tc->printme();
echo $tc->var;
?>
```

Interfaces

```
<?php
interface Printable {
    function printme();
}

class PrintTest implements Printable {
    function printme() {
        echo "Hallo Printable";
    }
}

$pt = new PrintTest;
if($pt instanceof Printable)
    $pt->printme();
?>
```

Type Hinting

```
<?php
class PrintTest implements Printable {
    function printme() {
        echo "Hallo Printable";
    }
}

class PrintablePrinter {
    function printme(Printable $printable) {
        $printable->printme();
    }
}

$pt = new PrintablePrinter;
$pt->printme(new PrintTest);

?>
```

Einheitliche Konstruktoren

```
<?php
public class Baseclass {
    function __construct() {
        echo "Baseclass::__construct()";
    }
}

public class Childclass extends Baseclass {
    function __construct() {
        parent::__construct();
        echo "Childclass::__construct()";
    }
}

$cc = new Childclass;

?>
```

Neue "Magic Functions"

Neue Magic Function:

- `__autoload($classname)`
 - Unbekannter Klassenname angefordert
 - kann in dieser Funktion nachgeladen werden

Methoden in Klassen:

- `__sleep()` und `__wakeup()`
 - Objekt-(De-)Serialisierung

Neue "Magic Functions"

- `__toString()`
 - Aufruf bei explizitem oder implizitem cast nach `string`
- `__get($var)` **und** `__set($var, $value)`
 - Zugriff auf nicht deklarierte Objektattribute
 - Sehr dynamische Klassen möglich
- `__call($method)`
 - Aufruf nicht im Objekt definierter Methoden (z.B. Webservices)

Objektdereferenzierung

```
<?php
/** Zurückgegebene Objekte können direkt angesprochen
werden. Bei PHP4 musste man diese zwischenspeichern .
Die Klassen im Beispiel sind fiktiv.*/

// PHP4:
$foo = FooFactory::getInstance();
$bar = $foo->getFooFilter();
$bar->filter("Hello World");

// PHP5, dank Objektdereferenzierung:
$foo = FooFactory::getInstance();
$foo->getFooFilter()->filter("Hello World");
?>
```

Standard Interfaces für PHP5

- Iterator
 - DirectoryIterator
 - SimpleXMLIterator
 - ArrayIterator ...
 - Anwendung z.B. für `foreach()` Schleifen
- Countable
 - für `count()`
- Exceptions
- Observer

Exception Handling

```
<?php
function exceptional($arg){
    if($arg) {
        throw new Exception('$arg was true...');
    }
}
try{ exceptional(true); }
catch (Exception $e) { echo $e;}
?>
exception 'Exception' with message '$arg was true...'
  in ...\\test.php:4
Stack trace:
#0 ...\\test.php(4): exceptional()
#1 ...\\test.php(9): exceptional(true)
#2 {main}
```

Fahrplan

1. Was ist PHP?
2. Grundlegende Syntax
3. Ein- und Ausgabe
4. Objektorientierung PHP4 vs. PHP5
5. *Neues MySQL Interface*
6. Sicherheitsaspekte
7. (PHP || !PHP)?
8. Lesenswertes

Neues MySQL Interface

- Objektorientierte Connection, ResultSets etc.
- Erweiterungen der Standardklassen möglich
 - neue Methoden
 - andere Arten, an Serverdaten zu kommen
- Objektdereferenzierung = weniger Code
 - Beispiel:

```
echo $mysqli->query("SELECT * FROM blubb")  
->fetch_object()->foo;
```
- Prepared Statements

Beispiel MySQLi

```
$mysqli = new mysqli('server', 'user', 'pw', 'db');  
if (mysqli_connect_errno()) {  
    echo "Connect failed: " . mysqli_connect_error();  
    exit();  
}  
  
$stmt = $mysqli->prepare(  
    "INSERT INTO mytable VALUES (?,?)");  
$stmt->bind_param('sd', $code, $percent);  
$code = 'DEU';  
$percent = 11.2;  
  
$stmt->execute();  
$stmt->close();  
$mysqli->close();
```

Fahrplan

1. Was ist PHP?
2. Grundlegende Syntax
3. Ein- und Ausgabe
4. Objektorientierung PHP4 vs. PHP5
5. Neues MySQL Interface
6. *Sicherheitsaspekte*
7. (PHP || !PHP)?
8. Lesenswertes

Sicherheitsaspekte

- Der User ist grundsätzlich böse
- Potentiellen HTML Code, der keiner sein sollte maskieren (`htmlspecialchars()`)
- Benutzereingaben nie in eine Query einfügen ohne zu prüfen/maskieren
- `register_globals` nie auf "on" setzen
 - bzw. global registrierte Vars nicht nutzen
- `magic_quotes` beachten

Cross Site Scripting (XSS)

Über eine Lücke kann jemand
(Client-)Code einschleusen

z.B.

```
http://example.com/test.php?user=  
<script>alert("Hallo");</script>
```

Server schreibt: User \$user unbekannt.

Das JavaScript läuft los...

Richtige Exploits nicht so harmlos!

Cross Site Scripting (XSS)

Folge:

- Anzeige falscher Information auf der eigenen Seite / Domain, auch **mit SSL**
→ z.B. Phishing durch geschickte Platzierung von JavaScript-Code

Lösungsansatz XSS:

- Eingaben immer validieren
 - Fehlerausgaben in HTML maskieren
- Vertraue niemandem, nicht einmal dir

SQL Injection

Ungültige Eingaben in Query einfügen

Beispiel, löscht alle User:

```
$inject = "1 OR 1 = 1";
```

```
mysql_query("DELETE FROM USERS WHERE user_id =  
$inject");
```

Lösungsansatz, z.B. Regular Expressions:

```
if(preg_match('/^\d+$/',$inject)) {  
    mysql_query(...);  
}
```

Besser: Prepared Statements

– Daten werden unbehandelt in DB gepackt

Fahrplan

1. Was ist PHP?
2. Grundlegende Syntax
3. Ein- und Ausgabe
4. Objektorientierung PHP4 vs. PHP5
5. Neues MySQL Interface
6. Sicherheitsaspekte
7. (*PHP* || *!PHP*)?
8. Lesenswertes

(PHP || !PHP)?

PHP lohnt sich für:

- den schnellen Einstieg in Webprogrammierung
- kleinere Projekte wie eine persönliche Homepage
- Intranets mit geringer Last

Nicht für:

- Sehr komplexe Systeme
- Anwendungen mit großer Last

Komplexe Systeme mit PHP

Erfordern:

- Extreme Disziplin
- Gute Nerven
- Code Cache (eAccelerator, Produkte von Zend)
- Debugger (z.B. Zend Studio)

Idealerweise:

- Unit Tests mit PHPUnit
- phpDoc (wie JavaDoc)

Fahrplan

1. Was ist PHP?
2. Grundlegende Syntax
3. Ein- und Ausgabe
4. Objektorientierung PHP4 vs. PHP5
5. Neues MySQL Interface
6. Sicherheitsaspekte
7. (PHP || !PHP)?
8. *Lesenswertes*

Lesenswertes

- PHP allgemein

- <http://de.php.net/manual/de/>
- <http://www.php-faq.de>

- Kein stabiles mod_php für Apache 2?

- <http://marc.theaimsgroup.com/?l=php-dev&m=108736540021355>
- <http://groups.google.com/group/php.general/msg/9031e5112f534413?fwc=1>

- Softwareentwicklung mit PHP5

- <http://www.professionelle-softwareentwicklung-mit-php5.de>

- Artikel von Zend.com

- <http://zend.com/php5/>
- <http://zend.com/php5/articles/engine2-php5-changes.php>
- <http://zend.com/php5/articles/php5-sqlite.php>
- <http://zend.com/php5/articles/php5-xmlphp.php>

Lesenswertes

- Template Engines

- <http://smarty.php.net>
- <http://sourceforge.net/projects/php-templates>

- Standard PHP Library

- <http://de.php.net/spl>

- Cross Site Scripting

- <http://www.cgisecurity.com/articles/xss-faq.shtml>
- <http://www.phpnuke.org/>

- SQL Injection

- <http://dev.mysql.com/tech-resources/articles/guide-to-php-security-ch3.pdf>
- <http://www.sqlsecurity.com/DesktopDefault.aspx?tabid=23>