RWTH AACHEN UNIVERSITY

Skriptprogrammierung

PHP, MySQL, JavaScript

Marius Politze

WS 2012, 03.09. – 07.09.

Stand: 16.07.2012

Version 1.5

Verlauf der Veranstaltung

- 5 Vorlesungen
- 3 Übungsblätter
- Projekt in Kleingruppen
- Alle Aufgaben müssen funktionsfähig abgegeben werden und bis Freitag den 14.9. als .zip Datei ins BSCW
- Die Abgabe der Aufgaben und des Projekts sind Bedingung für die Klausurzulassung
- Klausur ist schriftlich am 21.09. um 8:30 im Hörsaal AH IV

Was haben wir vor?

- **Vorlesung**
  - Grundlagen für Skriptsprachen (PHP, JavaScript)
  - Grundlagen für dynamische Webprogrammierung
- **Übungsaufgaben**
  - PHP, JavaScript (jQuery)
  - Werden alleine bearbeitet
- **Projekt: Webprogramierung**
  - LAMP (Linux, Apache, MySql, PHP)
  - Ein Projekt in Kleingruppen

Gruppeneinteilung

- **Eine Projektgruppe besteht aus**
  - 1 Projektleiter
  - 5-6 Programmierer
- **Projektleiter Gruppe**
  - 10 Personen, 1 pro Gruppe
  - Kommen gleich mit mir und bekommen eine Aufgabe
- **Programmierer Gruppe**
  - Alle anderen
  - Bleiben hier und bekommen eine Aufgabe

Mehr Organisatorisches…

Zeit	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
08:15 – 09:45	Vorlesung Organisatorisches Grundl. Skriptsprachen	Vorlesung Formulare JavaScript (jQuery)	Übungen	Vorlesung Sicherheitsaspekte Regular Expressions	Wiederholung für Klausur
10:00 – 11:30	Übungen	Übungen	Übungen	Projektarbeit	Projektarbeit
11:30 – 12:30	Mittagspause	Mittagspause	Mittagspause	Mittagspause	Mittagspause
12:30 – 14:00	Vorlesung Grundlagen PHP Dynamische Webseiten	Vorlesung Datenbanken mit PHP Sessions	Projektarbeit	Projektarbeit	Vorstellung Projekte
14:15 – 15:45	Projektstart	Projektarbeit	Projektarbeit	Projektarbeit	Vorstellung Projekte
15:45 – 16:00	Feedback	Feedback	Feedback	Feedback	Feedback

RWTH AACHEN UNIVERSITY

Skriptsprachen

PHP und JavaScript

Skriptsprachen - Übersicht

- **Warum Skriptsprachen?**
- **Grundlagen**
- **Verschiedene Skriptsprachen**
  - PHP, JavaScript
- **Skriptsprachen „Internals“**

Warum Skriptsprachen?

- **Automatisierung von Abläufen**
- **Änderungen „zur Laufzeit“**
  - Änderungen des Verhaltens des Programms
  - Häufig nicht vom Programmierer, sondern vom Benutzer
- **Anwendungsgebiete**
  - Robotik / Künstliche Intelligenz
  - Webseiten
  - (Server) Administration
  - Makros z.B. MS Office

- **Klassische Merkmale der Skriptsprachen**
  - Deklaration von Variablen i.d.R. unnötig
  - Dynamische Typisierung / Duck Typing
  - Automatische Speicherverwaltung
  - Werden nicht kompiliert (der Quellcode wird „direkt“ ausgeführt)
  - Keine Objektorientierung

- **Viele dieser Merkmale sind inzwischen überholt!**

PHP – PHP Hypertext Processor

- Entwickelt 1994/95 von Rasmus Lerdorf: „Personal Home Page“
- Später umbenannt in „PHP Hypertext Processor“
- Objektorientiert seit Version 5.0
- Aktuelle Version 5.3
- Haupteinsatzgebiet: Programmierung von Webseiten (zusammen mit HTML und JavaScript)

- **Entwickelt 1995 von Brendan Eich**
- **Objektorientiert (aber Klassenlos)**
- **Erstellung dynamischer Inhalte für Webbrowser**
  - Extrem weit verbreitet
  - So gut, wie jeder Browser unterstützt JavaScript (wirklich?)
- **Bibliotheken vereinfachen cross-Browser Einsatz**
  - qooxdoo
  - jQuery

- **Klassische Merkmale der Skriptsprachen**
  - Deklaration von Variablen i.d.R. unnötig
  - Dynamische Typisierung / Duck Typing
  - Automatische Speicherverwaltung
  - Werden nicht kompiliert (der Quellcode wird „direkt“ ausgeführt)
  - Keine Objektorientierung

- **Viele dieser Merkmale sind inzwischen überholt!**

Unterschiedliche Skriptsprachen

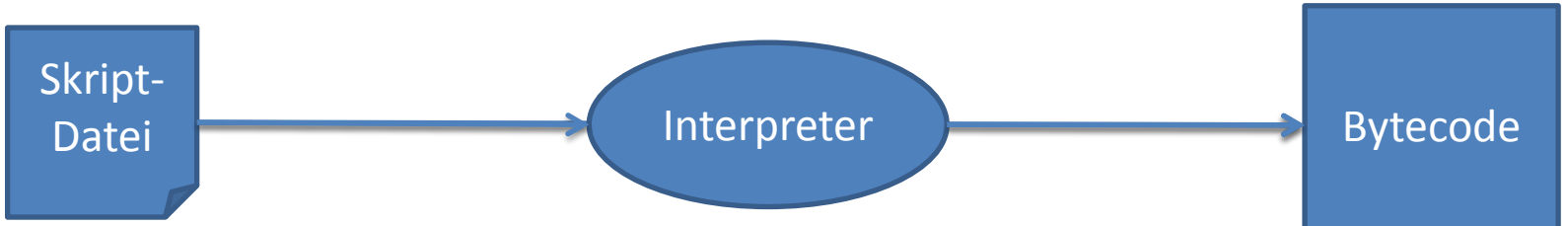
- **Stapelverarbeitungs-, Batch- und Shellskripte**
  - Betriebssystemabhängig
  - Abfolge von Aufrufen
  - Keine Bibliotheken oder vorgefertigte Funktionen
- **High-Level Skriptsprachen**
  - Interpreter muss i.d.R. nachinstalliert werden
  - Komplexe Kontrollstrukturen und Arrays
  - „Batteries Included“

Unterschiedliche Skriptsprachen

- **Stapelverarbeitungs-, Batch- und Shellskripte**
  - Bourne Script
  - DOS-Batch
- **High-Level Skriptsprachen**
  - JavaScript
  - PHP
  - Python

- **Batch- / Shellskripte**
  - http://de.wikibooks.org/wiki/Linux-Kompendium:_Shellprogrammierung
  - http://en.wikipedia.org/wiki/List_of_DOS_commands
- **High Level**
  - http://wiki.python.org/moin/SimplePrograms
  - http://www.w3schools.com/js/js_examples.asp
  - http://www.w3schools.com
  - http://www.coli.uni-saarland.de/~regneri/courses/Python1-08/page.php?id=slides

Wie funktioniert das nun?

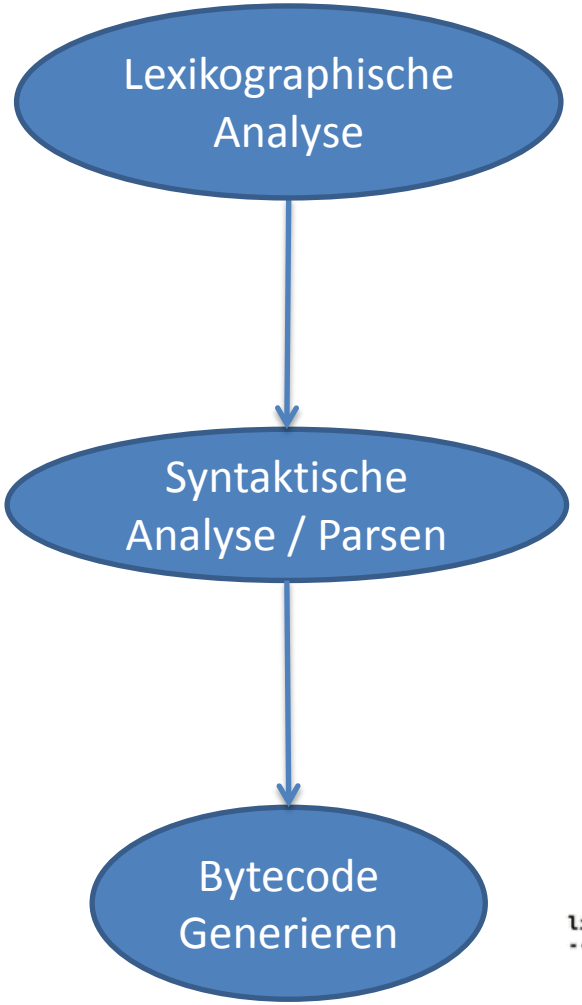


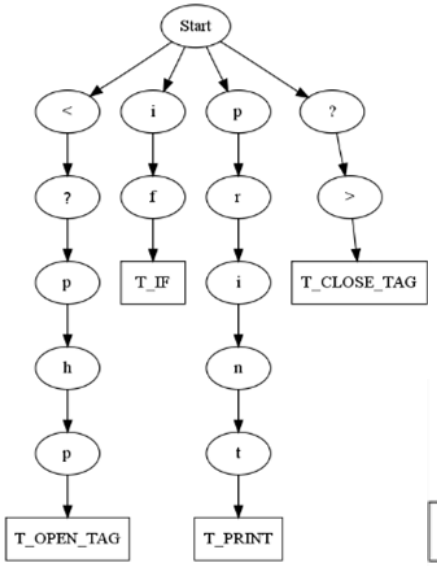

```
1 <?php
2 if (TRUE) {
3     print '*';
4 }
5 ?>
```

```
line     #  op                           fetch          ext  return  operands
---------------------------------------------------------------------------------
   2     0  EXT_STMT
         1  JMPZ                                                    true, ->6
   3     2  EXT_STMT
         3  PRINT                                        ~0         '%2A'
         4  FREE                                                    ~0
   4     5  JMP                                                     ->6
   6     6  EXT_STMT
         7  RETURN                                                  1
```

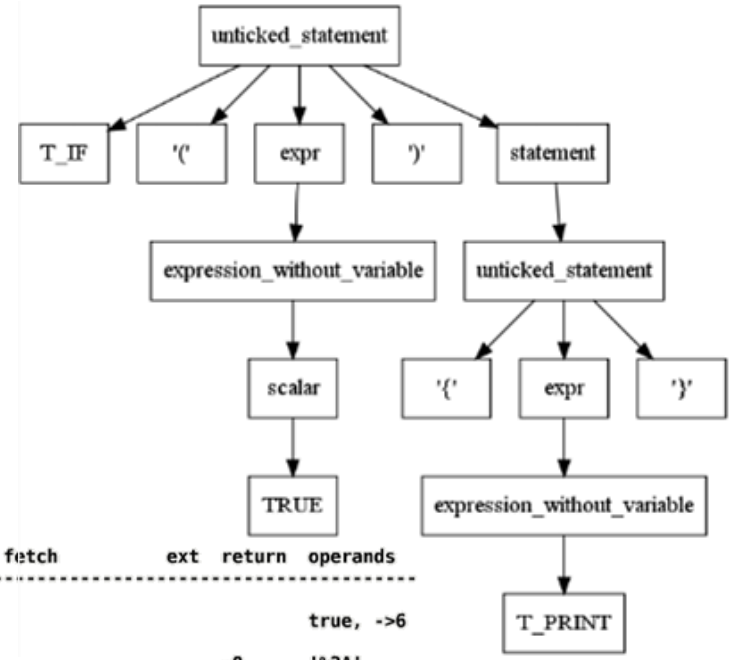
Basierend auf Folien (PHP Compiler Internals) von Sebastian Bergmann http://sebastian-bergmann.de/

Der Interpreter






```
1 <?php                          T_OPEN_TAG
2 if (TRUE) {                    T_IF
                                 T_WHITESPACE
                                 (
                                 T_STRING
                                 )
                                 T_WHITESPACE
                                 {
                                 T_WHITESPACE
3     print '*';                 T_PRINT
                                 T_WHITESPACE
                                 T_CONSTANT_ENCAPSED_STRING
                                 ;
                                 T_WHITESPACE
4 }                              }
                                 T_WHITESPACE
5 ?>                             T_CLOSE_TAG
```




```
line     #  op                                fetch          ext  return  operands
-------------------------------------------------------------------------------
   2     0  EXT_STMT
         1  JMPZ                                                      true, ->6
   3     2  EXT_STMT
         3  PRINT                                             ~0      '%2A'
         4  FREE                                                      ~0
   4     5  JMP                                                       ->6
   6     6  EXT_STMT
         7  RETURN                                                    1
```

Was hat das jetzt mit mir zu tun?

Fallstricke beim Parsing

- **Funktionen, Variablen und Klassen werden nur unter ihrem Namen abgelegt**
  - Kein überladen von Funktionen möglich
  - Ermöglicht das nachträgliche ändern von Klassen (zur Laufzeit)
  - Verknüpfung Name <-> Funktion muss schnell sein (O(1), O(log(n)), O(n))
- **Schnelle Dictionary Implementierungen nötig!**
  - Ermöglicht schnellen Zugriff auf Daten über den Namen
  - Kann i.d.R. auch als Datenstruktur für Skripte benutzt werden

ADT Dictionary

ADT Dictionary

	List	Tree	Hash Map
Space	O(n)	O(n)	O(n)
Search	O(n)	O(log n)	O(1)
Insert	O(1)	O(log n)	O(1)
Delete	O(n)	O(log n)	O(1)

PHP

Übersicht

- PHP grundlegende Syntax
- Wichtige Funktionen
- Do s und Don‘t s
- Funktionsreferenz unter: http://de3.php.net/manual/de/funcref.php

- **Weitestgehend gleich zu Java!**
- **Unterschiede liegen im Detail**
  - Benutzung von Variablen
  - Schreiben von Funktionen
  - Einbinden von Bibliotheken (Funktionen aus anderen Dateien)
- `<?php phpinfo() ?>`

PHP Syntax - Variablen

- Achtung! Alle Variablennamen beginnen mit $
- Variablen müssen nicht deklariert werden
- Variablen haben keinen festen Typ
- Beispiel:

```
<?php
    $variable = "Hallo Welt";
    $variable = 21;
    $variable *= 2;

    echo $variable;
?>
```

PHP Syntax - Datentypen

- int – ganzzahl
- bool – logische Werte (true, false)
- float / double – 64 Bit Fließkommazahl (IEEE 754)
- string – Zeichenkette
- array – (assoziatives) Array
- object – Objekt (im Sinne der OOP)

- **Logik**
  - Leere werte (null, 0) → false
  - Alle anderen true
- **Array: array()**
  - Erzeugt ein leeres Array
- **Objekttypen: null**
  - Leerer Wert

PHP Syntax - Typumwandlung

- **Häufig: Automatische Typumwandlung**

- **Datenbanken sind häufig nicht so flexibel**
  - Eine explizite Umwandlung ist ggf. Ratsam

```
$zahl = 42;
$text = (string) $zahl;
```

- **Dabei sind folgende Konvertierungen möglich:**
  - double, string → int
  - Int, string → double
  - int, double → string

- **Wird keine Zahl in einem String erkannt, wir 0 zurückgegeben**

PHP Syntax – Externe Dateien einbinden

- **Die Schlüsselworte `include` und `require` binden externe Dateien ein**

- **Sie werden benutzt um**
  - Funktionen in andere Dateien auszulagern
  - HTML Code oder anderen, langen und wiederkehrenden Code in Dateien auszulagern.

- **`include` verhält sich so, also würde die eingebundene Datei an der Stelle eingefügt werden.**
  - Variablen aus der Hauptdatei können benutzt werden
  - Variablen können verändert werden und sind nach dem `include` verfügbar

```
void include ( string $filename )
```

- Lädt eine Datei und führt sie separat aus
- Anwendung in Schleifen zulässig
- Datei wird je Befehlsaufruf jedes Mal neu geladen und interpretiert
- `include` erfolgt aber nur dann, wenn die Abarbeitung des Quellcodes an dem entsprechenden Funktionsaufruf angelangt ist
- Kann die Datei nicht eingebunden werden, wird eine Warnung ausgegeben, das Skript läuft aber weiter!

```
void require ( string $filename )
```

- Ausführung vor dem Parsen des gesamten Skriptes
- Nicht in Schleifen verwendbar
- Kann die Datei nicht eingebunden werden, wird die Ausführung des Skripts abgebrochen

PHP Syntax – `include`, `require` Varianten

RWTH AACHEN UNIVERSITY

```
void include_once ( string $filename )
```

- Entspricht der Funktion von include
- Eine Datei, die bereits eingebunden wurde, wird nicht noch mal eingebunden
- Praktisch für Funktionsdefinitionen!

```
void require_once ( string $filename )
```

- Analog zu `include_once`

PHP Syntax – Spezielle Befehle

- `exit`
  - Bricht die Ausführung des Skripts an der Stelle ab
- `header`
  - Weiterleitung auf eine andere Seite
  - Festlegen des Seitentyps. z.B. Download oder Webseite
- `echo, print_r`
  - Ausgabe

```
void header ( string $string )
```

- Sendet HTTP-Anfangsinformationen (Header) im Rohformat

  → so als würde vom Browser eine neue Seite aufgerufen

  ```
  header("Location: http://www.google.de");
  ```

- Muss vor jeglicher HTML-Ausgabe ausgeführt werden
  - Achtung: Auch ein Leerzeichen außerhalb von <?php … ?> ist schon

    eine HTML-Ausgabe

- Die weitere Ausführung von PHP-Code des aktuellen Skriptes sollte abgebrochen werden (`exit`)

PHP Syntax - `exit`

```
void exit ( void )
```

- Beendet das aktuelle Skript (Angabe der Klammern ist optional)
- Jeder weitere PHP-Code wird ignoriert

PHP Syntax – `echo`, `print_r`

```
void echo ( string $arg1 [, string $... ] )
```

- Gibt alle Parameter aus

```
mixed print_r ( mixed $expression [, bool $return = false ] )
```

- Gibt eine lesbare Form des Ausdrucks aus, Arrays werden rekursiv durchlaufen.
- Wenn `return true` ist, wird nicht ausgegeben, aber der Text wird zurückgegeben.

PHP Syntax – String Operationen

- Zeichenketten können in " " oder in ' ' angegeben werden
- Steuerzeichen (z.B. \n) wird nur in Double-Quotes erkannt
- In Double-Quotes werden Variablen sofort ersetzt
  - `$sieben=5+2; echo "5+2= $sieben";` → 5+2=7
  - `$sieben=5+2; echo '5+2= $sieben';` → 5+2=$sieben
- Verbinden von Zeichenketten mit .
  - `$sieben=5+2; echo '5+2='.$sieben;` → 5+2=7
  - `$text .= $neuer_text;`

PHP Syntax – Mehr zu Strings

- PHP Anwendungen verwenden für fast alles Strings
- PHP hat eine Vielzahl von String Operationen
- http://www.php.net/manual/en/ref.strings.php liefert eine erste Übersicht
- Bei Fragen: Ein Blick in die PHP Dokumentation ist häufig sehr hilfreich, da es sehr viele Beispiele gibt!
- Einige wichtige Funktionen:

  `substr`, `nl2br`, `explode`, `implode`, `trim`,

  `strlen`, `date`, `number_format`, `preg_match` ...

PHP Syntax - Arrays

- **Spezielles Sprachkonstrukt in PHP**
- **Index des Arrays wird in [ ] angegeben**
- **Achtung!**
  Arrays in PHP sind assoziativ und haben dynamische Größe
- **Initialisierung:**
  - `$array = array();` oder
  - `$werte=array(1,2,3,4);` oder
  - `$room = array("Peter" => 13);` oder
  - `$colors = array("red" => array(255,0,0));`

PHP Syntax – Zugriff auf Arrays

- **`$array = array();`**
  - `$array[] = "neuer Eintrag";`

- **`$werte=array(1,2,3,4);`**
  - `$werte[0] = 5; // 5, 2, 3, 4`
  - `$werte["test"] = 42; // 0=>5, 1=>2, 2=>3, 3=>4, "test"=>42`

- **`$room = array("Peter" => 13);`**
  - `$peters_raum = $room["Peter"]; // 13`

PHP Syntax – Besonderheiten von Arrays

- Der Schlüssel von Arrays kann numerisch oder ein Name sein
- Jeder Schlüssel ist nur ein Mal vorhanden (keine Duplikate)
- Numerische Arrays können Lücken haben
- Achtung! Viele Funktionen erwarten allerdings lückenlose Arrays
- Arrays wachsen mit ihrem Inhalt. Mit `$array[] = $value` können werte angehängt werden
  - Das Element wird mit dem niedrigsten numerischen Index angefügt

PHP Syntax – Arrays sortieren

```
bool sort ( array &$array )
```

- Sortiert ein Array aufsteigend vom niedrigsten zum höchsten Wert

```
bool asort ( array &$array  )
```

- Sortiert ohne den Zusammenhang zwischen Schlüssel und Wert zu verändern

  → besonders für assoziative Arrays verwendet

```
bool ksort ( array &$array )
```

- Sortiert nach Schlüssel, Zusammenhang zwischen Schüssel und Wert bleibt unverändert

PHP Syntax - Schleifen

- `while`, `do while` und `for` wie in Java
- Sepzialfall: foreach schleife
  - ```
    foreach ($array as $key => $value){
    …
    }
    ```
- Oder einfacher:
  - ```
    foreach ($array as $value){
    …
    }
    ```
- Achtung! Änderung von `$value` übertragt sich nicht ins Array

PHP Syntax - Funktionen

- Wie Variablen sind auch Funktionen nicht Typsicher!
- Funktionen werden mit dem Schlüsselwort `function` definiert, ein Rückgabewert wird nicht definiert.
- Funktionsargumente in Klammern da hinter. Optional die Angabe von Standardwerten.
- Beispiel:

```
<?php
function test($p1, $p2=42){
    if($p1 == $p2) return true;
    else return $p1;
}

$ergebnis = test(1,2);
?>
```

PHP Syntax – Spezialfälle Funktionen

- **Innerhalb einer Funktion kann die Anzahl der tatsächlich übergebenen Parameter abgefragt werden:**

- **Beispiel:**

```
myfunk("Heute", "beginnt", "der", "PHP-Kurs");

function myfunk() {

$anzahl = func_num_args();

$tmp =  func_get_arg(2); // liefert „der„

$args = func_get_args();

...

}
```

- **Seit PHP5 werden auch Klassen unterstützt (Schlüsselwort `class`)**
  - Alles sehr ähnlich zu Java!
- **Vererbung durch `extends`**
- **Zugriffsmodifizierer**
  - public – Jeder kann auf das Attribut zugreifen
  - protected – Erbende Klassen können zugreifen
  - private – Zugriff nur innerhalb der Klasse möglich

- **Member Variablen**
  - Innerhalb der Klasse: `$this->name`
  - Außerhalb der Klasse: `$objekt->name`
  - Achtung! kein $ vor dem Namen der Membervariablen!
- **Erzeugen von Objekten**
  - $user = new User();
- **Konstruktor: __construct()**
- **Destruktor: __destruct()**
  - Je zwei führende Unterstriche
  - Nur EIN Konstruktor und Destuktor pro Klasse!

PHP Syntax – Objektorientierung Beispiel

```
class SimpleClass {
  // Member-Variablen
  private $var = "Vorgabewert";
  // Methoden
  public function __construct($new_var = ""){
    $this->var = $new_var;
  }
  public function displayVar() {
    echo $this->var;
  }
}
$obj = new SimpleClass("Testwert"); // neue Instanz der
                                       Klasse
$obj->displayVar();                 // Aufruf der Methode
```

Achtung! PHP verarbeitet Dateien anders als Java!

```
resource fopen ( string $filename , string $mode )
```

- Öffnet die Datei `$filename`
- Rückgabe `false` im Fehlerfall
- `$mode` gibt den Zugriffstyp an: `"r"` zum Lesen, `"w"` zum Schreiben, `"a"` zum Anhängen

```
bool fclose ( resource $handle )
```

- Schließt eine Datei
- Rückgabe `true` im Erfolgsfall, sonst `false`

```
int filesize ( string $filename )
```

- Rückgabe: Größe der Datei in Bytes, `false` im Fehlerfall

PHP Syntax – Dateioperationen 2

```
bool file_exists ( string $filename )
```

- Rückgabe `true`, falls die Datei oder das Verzeichnis existiert,
  sonst `false`

```
string fread ( resource $handle , int $length )
```

- Liest von einer vorher geöffneten Datei `$handle`
- Das Lesen endet nach `$length` Bytes oder bei Erreichen von EOF

```
int fwrite ( resource $handle , string $string [, int $length ] )
```

- Schreibt die Zeichenkette `$string` in die zuvor geöffnete Datei `$handle`
- Rückgabe: Anzahl der geschriebenen Bytes im Erfolgsfall,
  sonst `false`

Grundlagen der Webseitenprogrammierung

Themenübersicht

- **Grundlagen der Skriptprogrammierung**
  - Fallstudie: Webseitenprogrammierung
  - Statische und Dynamische Webseiten
  - Client- und Serverseitige Programmierung
- **Übertragung der Daten von und zum Webserver**
  - JSON
  - GET und POST
  - Auswertung von Formularen
- **Datenbankzugriff**
  - Datenbanken Grundlagen
  - PHP Datenbankanbindung
- **Cookies und Sessions**

Statische Webseiten

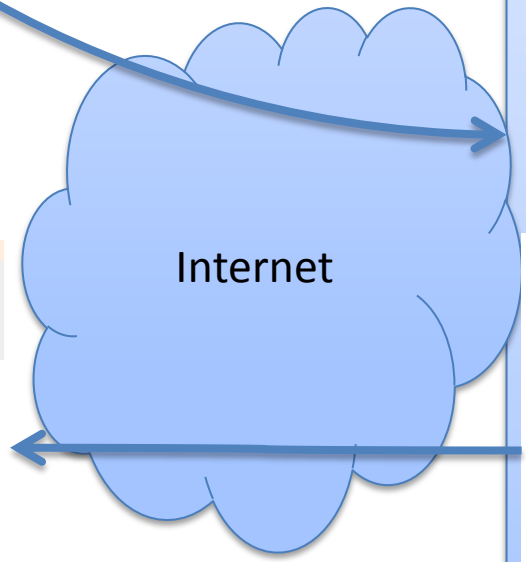
- Statische Webseiten sind unveränderlich. Häufig nur HTML und CSS
- Stellen immer den gleichen Inhalt dar

Dynamische Webseiten

- Werden erst beim Aufruf der Seite erzeugt
- Inhalt abhängig von Aktionen des Benutzers
- Einfügen von Daten aus Datenbanken
- Reaktionen auf Formularfelder

Statische Webseiten

http://de.selfhtml.org/index.htm

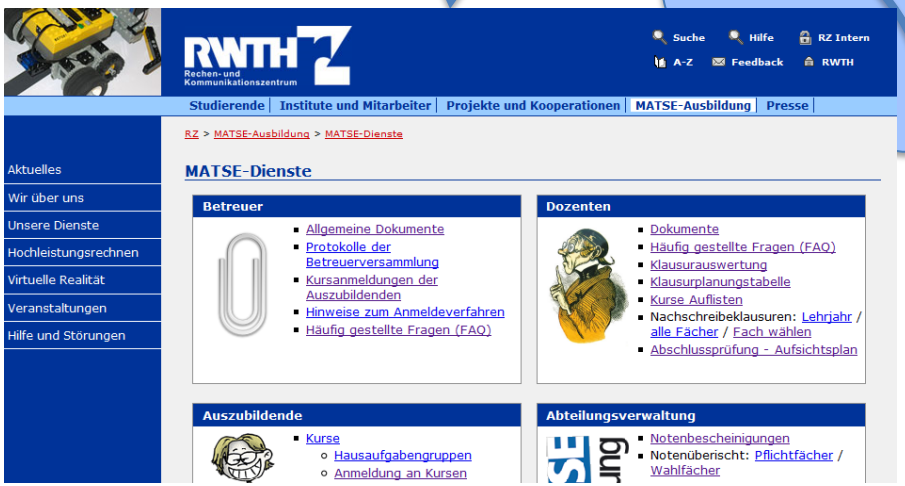


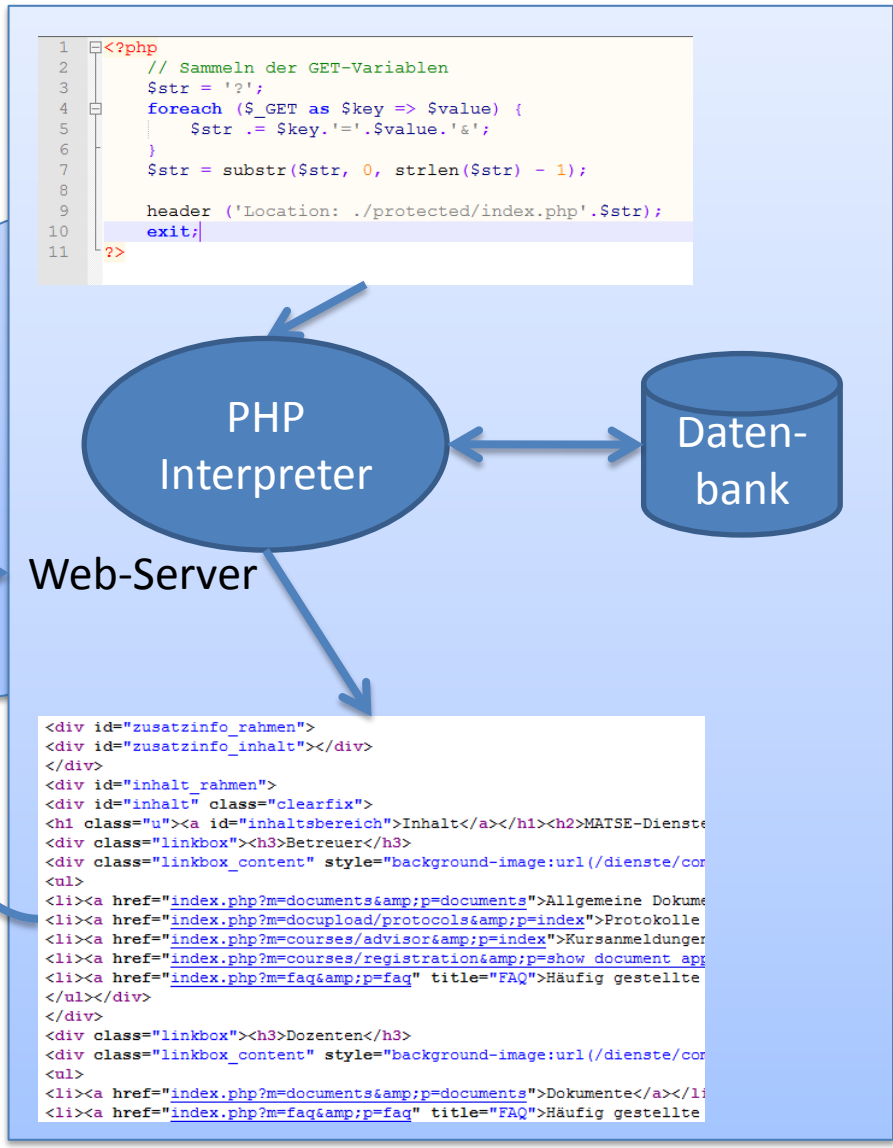
Web-Server

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859
<meta http-equiv="Content-Style-Type" content="text/css">
<title>SELFHTML 8.1.2 (HTML-Dateien selbst erstellen)</title>
<meta name="description"    content="SELFHTML 8.1.2 - Die bekannte D
<meta name="keywords"       content="SELFHTML, HTML, Dynamic HTML, J
<meta name="author"         content="Redaktion SELFHTML, selfhtml81@
<meta name="DC.Publisher"   content="SELFHTML e. V.">
<meta name="DC.Date"        content="2005-11-11T12:48:29+01:00">
<meta name="DC.Identifier"  content="http://de.selfhtml.org/">
<meta name="DC.Language"    content="de">
<meta name="DC.Rights"      content="editorial/copyright.htm">
<meta name="DC.Date.created" content="2001-10-27T08:00+01:00">
<meta name="SELF.Pagetype"  content="chapter">
<link rel="stylesheet" type="text/css" href="src/selfhtml.css">
<link rel="alternate" type="application/atom+xml" title="SELFHTML-We
<link rel="alternate" type="application/rss+xml" title="SELFHTML-Web
<link rel="shortcut icon" type="image/x-icon" href="src/favicon.ico"
```

Dynamische Webseiten

http://www.matse.rz.rwth-aachen.de/dienste/protected/index.php





Erster Kontakt mit PHP

- PHP wird verwendet um auf dem Webserver dynamische Webseiten zu erstellen
- Dafür wird das PHP-Skript vom Webserver ausgeführt und generiert den Inhalt der Webseite
- Ob (und was) der Webserver kann, kann mit dem Befehl
  `<?php phpinfo(); ?>`
  herausgefunden werden

PHP-Beispieldatei

```
<?php

$titel = "Entwicklung dynamischer Webseiten";

$gruss = "Willkommen im PHP/MySQL-Kurs";

?>

<html>

<head><title><?php echo $titel; ?></title></head>

<body><?php echo $gruss; ?></body>

</html>
```

Ausgabe der PHP-Beispieldatei

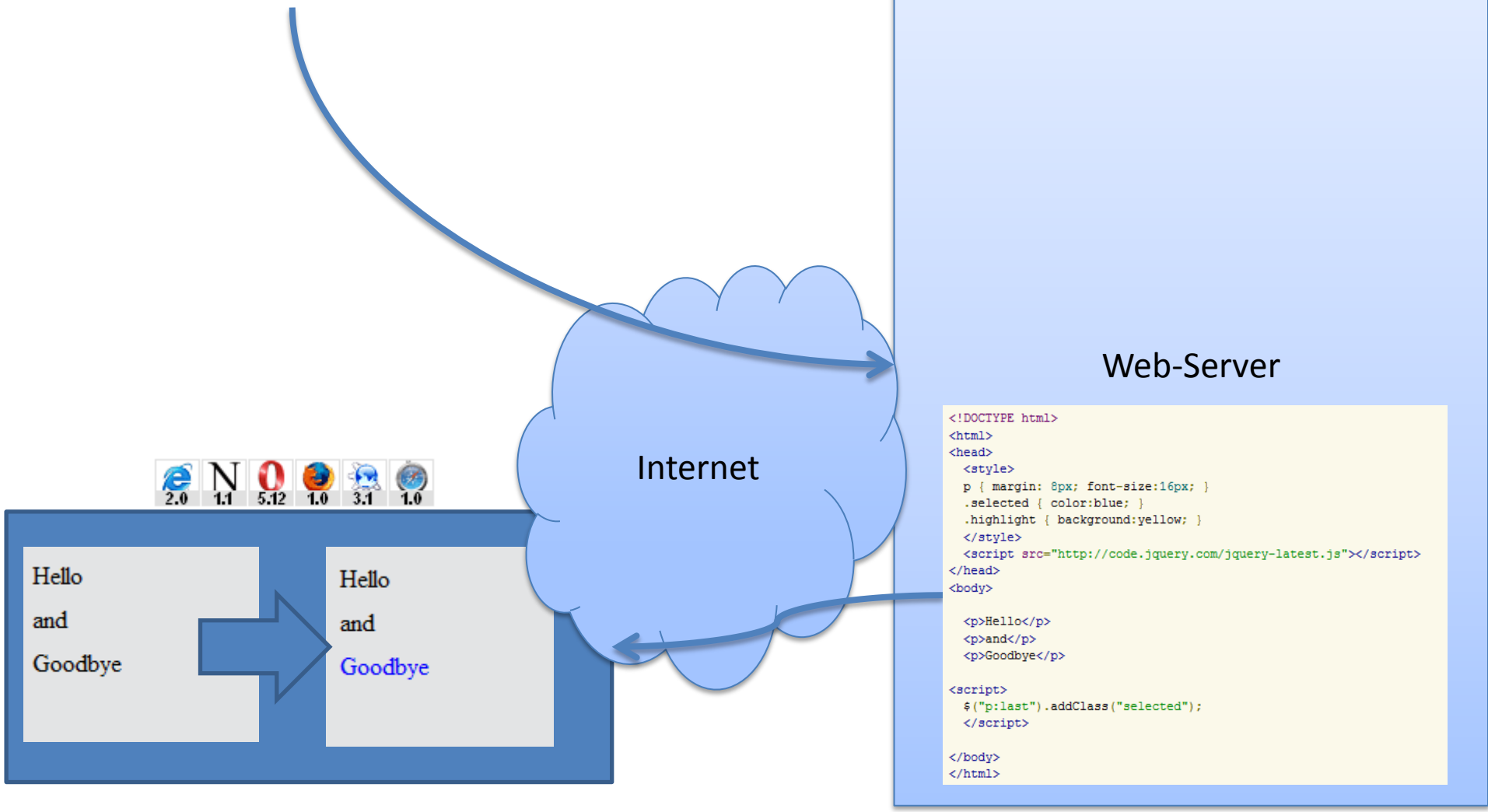
```
<html>

<head><title> Entwicklung dynamischer Webseiten

  </title></head>

<body> Willkommen im PHP/MySQL-Kurs </body>

</html>
```

PHP – Der klassische Weg

- **Die Beispieldatei zeigt, wie PHP benutzt wird um eine HTML Seite direkt mit Inhalt zu füllen**
- **Die Variante vermischt die Darstellung und die Berechnung der Daten (Business Intelligence)**
- **Bei einer Änderung an der Darstellung muss daher häufig Programmcode, der gar nicht betroffen ist mit geändert werden.**
- **Eine (moderne) Alternative:**
  - Generieren der Daten mit PHP und dann
  - Nutzen von JavaScript zur Darstellung

Dynamische Webseiten: Clientseitig

http://api.jquery.com/addClass/



Erster Kontakt mit JavaScript

- JavaScript wird verwendet um dem dynamische Seiten im Webbrowser zu erstellen
- JavaScript wird um Webbrowser ausgeführt und verändert i.d.R. nur die Darstellung, aber nicht den Inhalt der Daten
- Erster JavaScript Code:

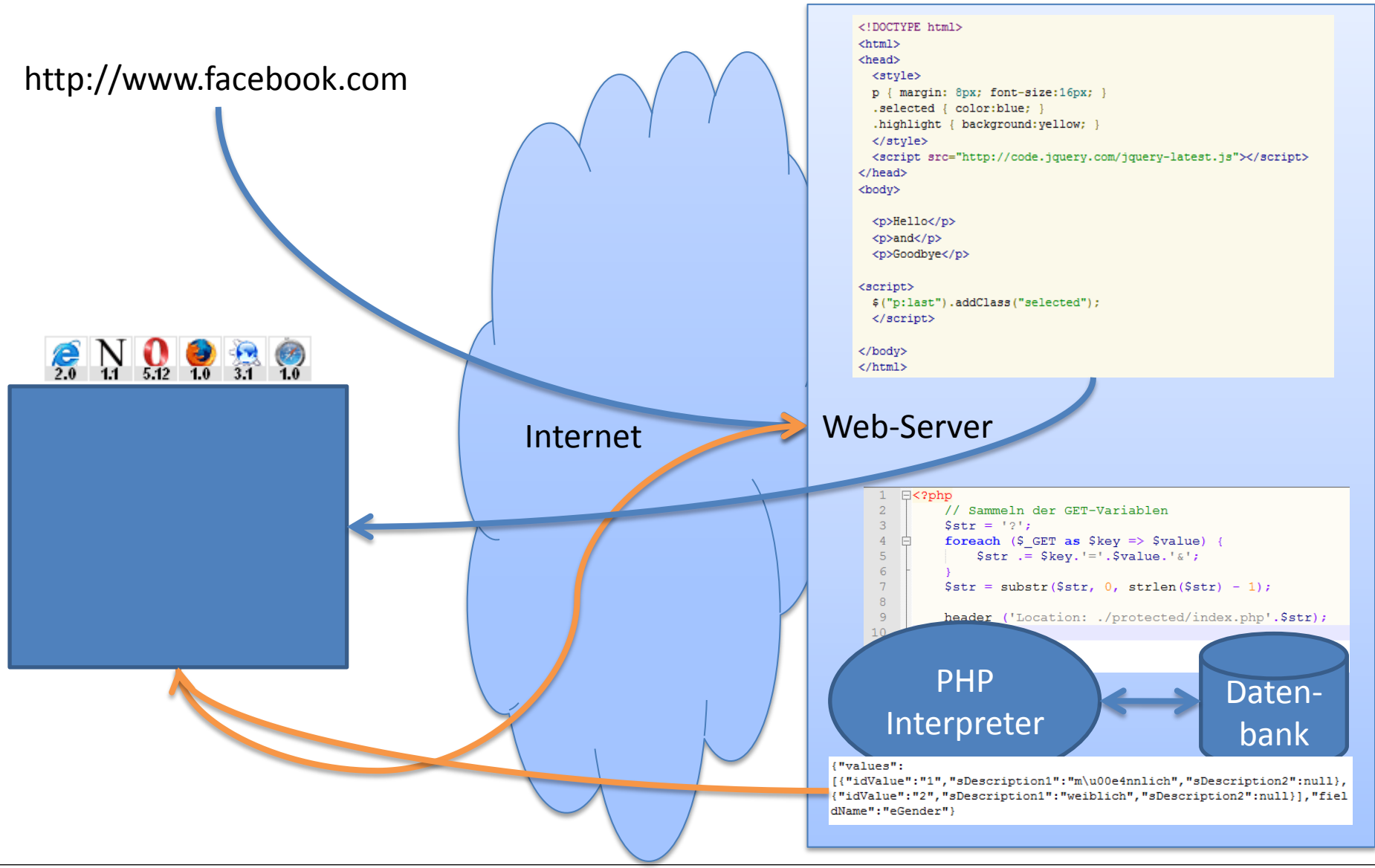
```
<html><body><script>
alert('Hello World!');
</script></body></html>
```

- Funktionsreferenz (nicht ganz so eindeutig, wie in PHP): http://de.selfhtml.org/

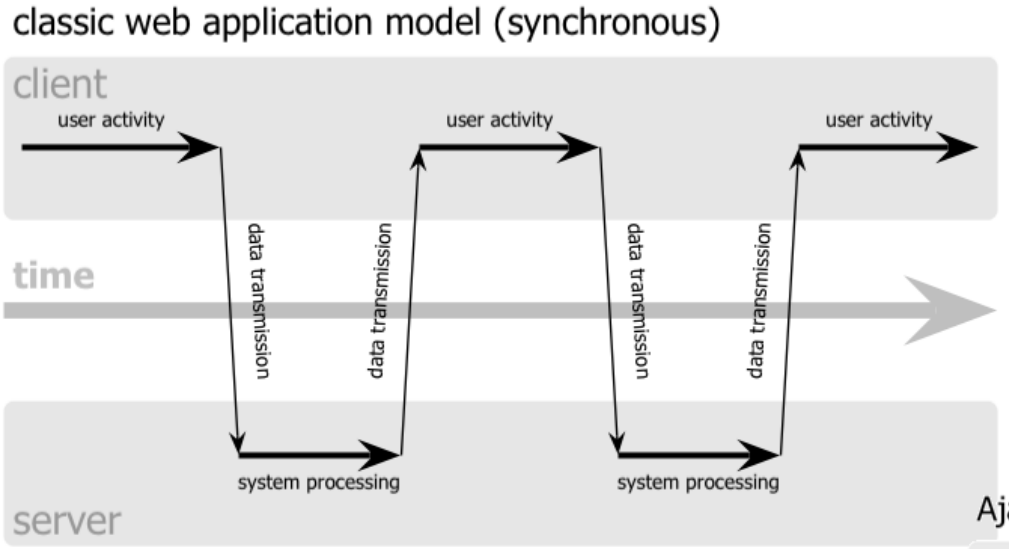
Kommunikation PHP - JavaScript

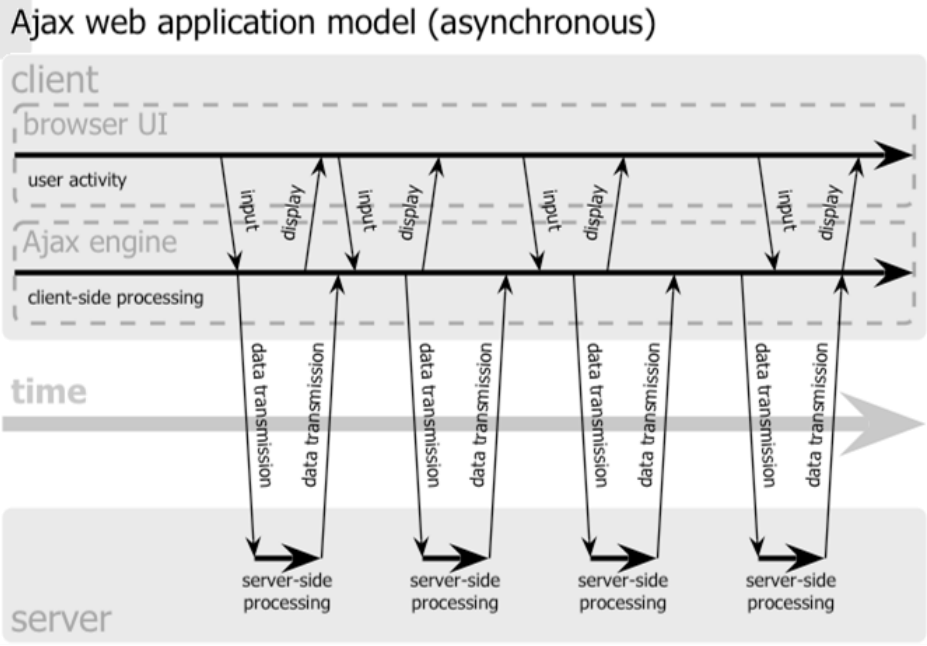
- **Damit PHP und JavaScript miteinander Kommunizieren können wird eine Schnittstelle benötigt**
- **Beide Skriptsprachen können gut mit Strings umgehen, daher**
  - Nutzen eines einheitlichen Formats um Objekte zu serialisieren (in einen String zu schreiben)
  - Häufig XML
  - Etwas einfacher zu Lesen JavaScript Object Notation (JSON)
- **Webseiten können mit JavaScript einzelne Seitenteile nachladen, ohne komplett neu geladen zu werden → das Klassische Modell wird erweitert**

AJAX: asynchronous JavaScript and XML



Klassisches Modell vs. AJAX





JSON - Beispiel

```
{
   "sNameTitle":null, "sNameAcadTitle":"B.Sc.",
   "sNameFirst":"Marius", "sNameMiddle":null,
   "sNameLast":"Politze","eGender":"1",
   "sTimId":"mp285453", "sPhone1":"0241 80 29720",
   "sPhone2":null, "sPhone3":null,
   "sFax1":null,
   "sMail1":"marius.politze@rwth-aachen.de",
   "sMail2":"politze@rz.rwth-aachen.de"
}
```

JSON – Benutzen…

- **PHP**
  - JSON ist seit PHP 5.2.0 fester Bestandteil von PHP
  - Objekte, Arrays können als JSON-String ausgegeben werden
  - JSON-Strings können in Objekte und Strings umgewandelt werden
- **JavaScript**
  - Viele Browser unterstützen JSON von Hause aus
  - Einfacher ist die Verwendung von json2.js

    https://github.com/douglascrockford/JSON-js

Datenübertragung von und zum Webserver

Datenübertragung - Übersicht

- **HTTP: Die Schnittstelle zum Webserver**
  - GET
  - POST
- **HTML-Formulare**
- **Erster Exkurs: Sicherheit von Formulardaten**

- Die Ausgabe des PHP Skripts wird an den Browser gesendet
- Mit HTML und CSS kann die Ausgabe formatiert werden
- Wie bekommt man jetzt Daten zum Webserver?
- Das HTTP Protokoll überträgt nur Daten beim Seitenaufruf
- Zwei Möglichkeiten
  - In der Adresse
  - In einem „versteckten“ Feld bei der HTTP Übertragung

Datenübertragung - GET

- **GET: Kodieren der Daten in der Adresse**
  - Anhängen der Parameter mit ? an die Adresse
  - Es folgen Wertepaare der Form: `name=wert`
  - Trennung der Wertepaare durch &
- **Beispiel: http://www.google.com/?q=php**
- **Die Parameter werden in PHP in einem Array gespeichert:**
  - `$_GET['q'] // liefert GET-Parameter q`
- **Begrenzung der URL ca. 2048 Zeichen – daher nicht für lange Übertragungen geeignet**

Datenübertragung - POST

- **POST überträgt die Daten im Body der HTTP Anfrage**
  - Besser für große Datenmengen
  - Sicherer (vom Benutzer nicht direkt einsehbar)
- **Wie auch bei GET werden die Parameter in einem Array gespeichert:**
  - `$_POST['q'] // liefert POST-Parameter q`
- **Formularfelder in HTML können per POST und per GET übergeben werden.**

- **Formulare zur Eingabe und Absenden von Daten:**

```
<form action="post.php" method="post" name="post">
  <input name="name" type="text" size="40">
  <input name="Submit" type="submit" value="Abschicken">
</form>
```

- **Eine Ausführliche Beschreibung gibt es unter: http://de.selfhtml.org/html/formulare/index.htm**

- **Sowohl POST als auch GET**

- **Beim Drücken des „Abschicken“ Buttons wird die Seite geladen, die unter `action` definiert wird. Mit JavaScript kann das Neu laden der Seite unterbunden werden.**

Datenübertragung – HTML Formulare

http://de.selfhtml.org/html/formulare/formatieren.htm

Datenübertragung – Werte

- Betreffende Attribute der Formular-Tags: name und value
- Checkbox: Übertragung des Checkbox-Namens und des Wertes z.B. „on“ im aktivierten Fall, sonst keine Wertübertragung
- Radio-Buttons: Übertragung des Namens und des ausgewählten Wertes im aktivierten Fall, wurde kein Radio-Button ausgewählt wird nichts übertragen

Datenübertragung – Werte

- **Textfeld (text, hidden, textarea, password): Immer Übertragung des Namens, bei ausgefülltem Feld der Wert, sonst leere Zeichenkette**
- **Select-Element: Analog zu Checkbox und Radio-Button**
  → ohne Auswahl nicht mal Übertragung des Namens
- **Buttons: Wenn das Attribut value gesetzt ist, wird dieser Wert übertragen, andernfalls nichts**

Dateiübertragung – Achtung!

- **Obwohl HTML spezielle Funktionen zur Überprüfung der werte auf der Client Seite bieten, müssen immer alle übertragenen Daten im PHP Skript auf Korrektheit geprüft werden**

- **Besonders Gefährlich**
  - Nicht veränderbare Textfelder:

    http://de.selfhtml.org/html/formulare/anzeige/readonly.htm
  - Hidden Felder:

    http://de.selfhtml.org/html/formulare/anzeige/input_hidden.htm

- **Insbesondere JavaScript und PHP Code in Textfeldern muss behandelt werden (Cross-Site Scripting)**
  - `string htmlentities ( string $string [, int $flags = ENT_COMPAT [, string $charset ]] )`

JavaScript / jQuery

Übersicht JavaScript / jQuery

- **Wozu JavaScript?**
- **JavaScript Syntax**
- **Benutzung von jQuery**
  - Eventhandling
  - Ajax

Wozu JavaScript?

- **Manipulation der Webseite ohne neu Laden**
  - Spart traffic und damit Geld
  - Gibt dem Benutzer das Gefühl eine Desktop Anwendung zu benutzen
  - Auslagern von Berechnungen auf den Client möglich
- **Browser nutzen JavaScript unterschiedlich**

```
function getMouseXY(e) // works on IE6,FF,Moz,Opera7 {
    if (!e) e = window.event;
    // works on IE, but not NS (we rely on NS passing us the event)
    if (e) {
        if (e.pageX || e.pageY) {
        // this doesn't work on IE6!! (works on FF,Moz,Opera7)
            mousex = e.pageX;
            mousey = e.pageY;
            algor = '[e.pageX]';
            if (e.clientX || e.clientY) algor += ' [e.clientX] '
        }
        else if (e.clientX || e.clientY) {
        // works on IE6,FF,Moz,Opera7
            mousex = e.clientX + document.body.scrollLeft;
            mousey = e.clientY + document.body.scrollTop;
            algor = '[e.clientX]';
            if (e.pageX || e.pageY) algor += ' [e.pageX] '
        }
    }
}
```

JavaScript – Syntax

- **Wie der Name schon sagt *Java*Script**
- **Im Wesentlichen prozedural zu programmieren**
- **Wie in PHP:**
  - Variablen ohne festen Typ
  - Funktionen ohne festen Rückgabe-Typ
  - Keine Überladung
- **Nicht wie in PHP**
  - Objektorientierung: Prototypisch
  - Keine „echten“ assoziativen Arrays

JavaScript – Syntax Beispiele

```
function factorial(n) {
    if (n === 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

```
function displayClosure() {
    var count = 0;
    return function () {
        return ++count;
    };
}
var inc = displayClosure();
inc(); // returns 1
inc(); // returns 2
inc(); // returns 3
```

```
function sum() {
    var i, x = 0;
    for (i = 0; i < arguments.length; ++i) {
        x += arguments[i];
    }
    return x;
}

sum(1, 2, 3); // returns 6
```

- Achtung! In JavaScript gibt es nur numerische Arrays!
- Auch assoziative Zuordnung ist möglich: Objekteigenschaften

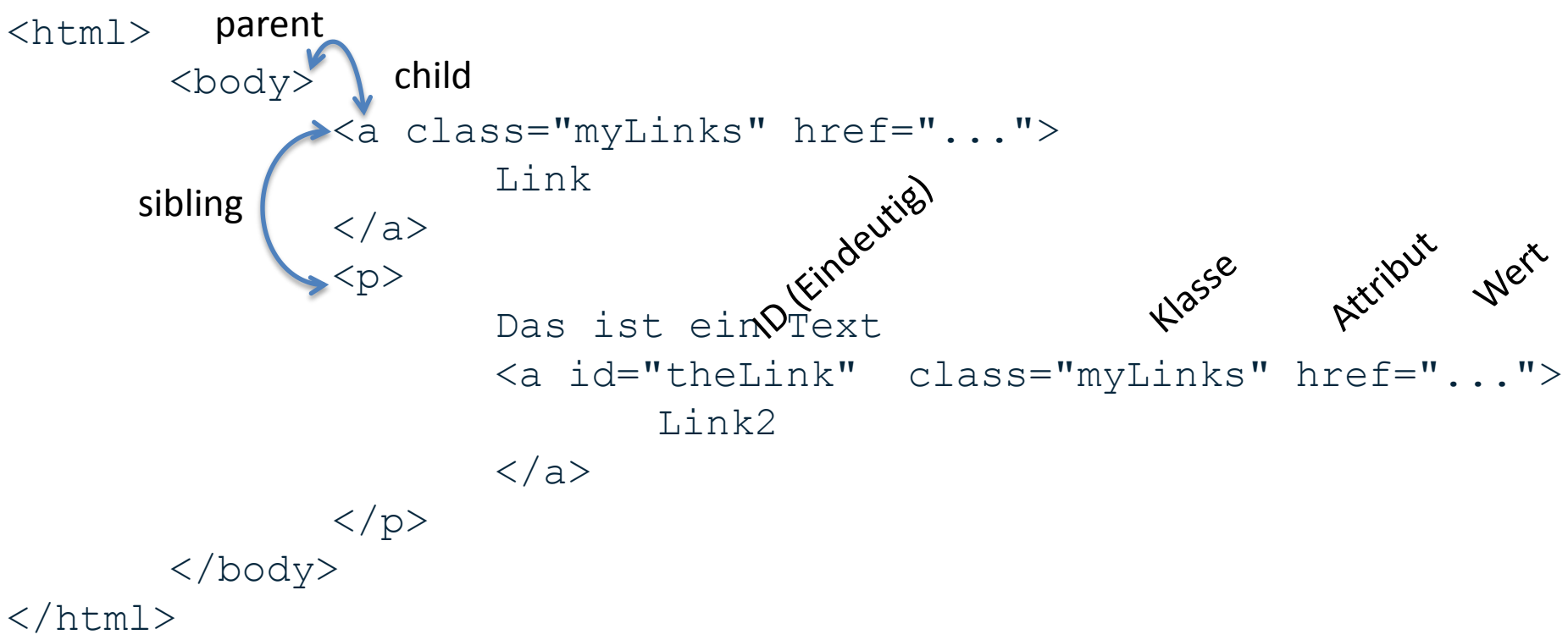
```
var k = new Object();

k['hallo'] = "test";

alert(k.hallo);
```

- Benutzung der „Foreach-Schleife“ zum Durchlaufen von Objekteigenschaften

```
for(p in k){
    alert(k[p]);
}
```

- **Vereinfacht den Umgang mit HTML Dokumenten**
- **Funktioniert in allen (aktuellen) Browsern gleich (IE 6.0+, FF 3.6+, Safari 5.0+, Opera, Chrome)**
- **Weit verbreitet bei Firmen und Projekten**
  - Google, Dell, Mozilla.org
  - Wordpress, Drupal

Enge Anbindung an das DOM-Modell der Webseite



Erste Schritte mit jQuery I

```
<html>
<head>
        <script type="text/javascript„ src="jquery.js"></script>
        <script type="text/javascript">
        $(document).ready(function() {
                //JavaScript hier...
        });
        </script>
</head>
        <body>
        <!-- Inhalt der Webseite hier... -->
        </body>
</html>
```

Erste Schritte mit jQuery II

```
<!DOCTYPE html>
<html>
<head>
  <style>
    body { font-size:14px; }
  </style>
  <script src="http://code.jquery.com/jquery-latest.js"></script>
  <script type="text/javascript">
    $(document).ready(function() {
      $("ul.topnav > li").css("border", "3px double red");
    });
  </script>

</head>
<body>

<ul class="topnav">
    <li>Item 1</li>
    <li class="special">Item 2
        <ul>
          <li class="special">Nested item 1</li>
          <li id="verySpecial">Nested item 2</li>
          <li>Nested item 3</li>
        </ul>
    </li>
    <li>Item 3</li>
</ul>
</body>
</html>
```

Events mit jQuery

```
<!DOCTYPE html>
<html>
<head>
  <style>
  p { color:red; margin:5px; cursor:pointer; }
  p.hilite { background:yellow; }
  </style>
  <script src="http://code.jquery.com/jquery-latest.js"></script>
</head>
<body>
  <p>First Paragraph</p>

  <p>Second Paragraph</p>
  <p>Yet one more Paragraph</p>
<script>
    $("p").click(function () {
      $(this).slideUp();
    });
    $("p").hover(function () {
      $(this).addClass("hilite");
    }, function () {
      $(this).removeClass("hilite");
    });
</script>

</body>
</html>
```

AJAX mit jQuery

```
<!DOCTYPE html>
<html>
<head>
  <style>img{ height: 100px; float: left; }</style>
  <script src="http://code.jquery.com/jquery-latest.js"></script>
</head>
<body>
  <div id="images">

</div>
<script>
  $.getJSON("http://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=?",
  {
    tags: "cat",
    tagmode: "any",
    format: "json"
  },
  function(data) {
    $.each(data.items, function(i,item){
      $("<img/>").attr("src", item.media.m).appendTo("#images");
      if ( i == 3 ) return false;
    });
  });
</script>

</body>
</html>
```

Persistente PHP Skripte

Persistente PHP Skripte

- Problem: PHP Skripte „leben“ immer nur für einen Aufruf der Seite
- Angelegte Objekte werden danach aus dem Speicher entfernt
- Um Daten zwischen zwei Seitenaufrufen zu übertragen können diese immer wieder per GET oder POST zwischen der Webseite und dem Webserver übertragen werden
- Damit nicht immer viele Hidden Felder übertragen werden müssen, gibt es die Möglichkeit Cookies an den Browser zu übertragen.
- Der Inhalt der Cookies wird dann bei jedem Seitenaufruf vom Browser an den Webserver übertragen

Cookies

- Cookies bieten die Möglichkeit clientseitig Informationen zu speichern
- So können Daten gespeichert und wieder gelesen werden
- Früher reine Textdateien, inzwischen in SQLite Datenbanken
- Auf Anfrage des Servers sendet der Browser den Wert des Cookies zurück
- Zum Cookie wird gespeichert, von welcher URL das Cookie angelegt wurde
  → nur diese URL kann das Cookie auslesen

Cookie – Aufbau

- **Name: Darüber kann das Cookie angesprochen werden**
- **Value: Wert des Cookies in Form einer Zeichenkette**
- **Expires: Lebensdauer des Cookies**
  - Ohne Angabe → bis zum Schließen des Browsers
  - Mit Zeitangabe → bis zum Ablauf der vorgegebenen Zeit
  - Vorzeitiges Löschen → Cookie neu speichern mit einer Zeitangabe, die in der Vergangenheit liegt

Cookies lesen in PHP

Abruf eines Cookies analog zur POST- bzw. GET-Methode

- `$_COOKIE[string cookiename]`
- Kann nicht in ein und demselben Skript gespeichert und wieder gelesen werden (in einem Aufruf)
- Ein Cookie wird erst mit Senden der Datei vom Browser gespeichert
  → erst das nächste Skript kann dieses Cookie lesen

Cookies setzen in PHP

```
bool setcookie (string $name, string $value [, int $expire])
```

- Speichert ein Cookie unter dem Namen `$name`
- Wert des Cookies in der Textdatei ist `$value`
- `$expire` ist optional
  - Unix Timestamp und ein Offset in Sekunden
  - Ohne Angabe → Cookie wird bei Schließen des Browsers gelöscht
  - Hilfreiche Funktion: `time`

Sessions

- Sessions werden vom Server verwaltet. Jeder Client bekommt eine Session-ID
- Die Session-ID wird vom Browser (in einem Cookie) verwaltet
- Das Übertragen und Erkennen der Session-ID wird von PHP übernommen
- Im gegensatz zu Cookies werden die Werte von Sessions nur auf dem Server gespeichert
- Sie sind nicht vom Benutzer einsehbar oder veränderbar
- Es können komplexe Variablen, wie Arrays oder Objekte gespeichert werden

Session in PHP

- **`session_start()` startet eine Session**
  - Muss auf jeder Seite als erstes ausgeführt werden
  - Falls bereits eine Session-ID besteht, wird die aktuelle ID verwendet und die zugehörige Session geöffnet
  - Lebenszeit: in der Regel 1440 Sekunden (24 Minuten)
- **Eine Session kann vorzeitig durch `session_destroy()` beendet werden**

Verwendung von Sessions

- **Variablen werden mittels Zuweisung an `$_SESSION["name"]` gespeichert:**
  - `$_SESSION["name"] = array("W" => "Walter„, "M" => "Monika");`
- **Zugriff auf eine Session-Variable erfolgt analog zu POST, GET oder einem Cookie-Abruf:**
  - `$array = $_SESSION["name"];`
    `// Oder auf alle Variablen über`
    `$sessionVars = $_SESSION; // liefert assoziatives Array`

Beenden der Session

- `session_unset()` löscht alle Sessionvariablen
- Einzelne Session-Variable löschen: `unset($_SESSION["name"]);`
  - Achtung! Nicht die Registrierung der gesamten `$_SESSION` mit `unset($_SESSION)` löschen!
- Um eine Session richtig zu beenden muss zudem noch der Session Cookie gelöscht werden:

```
$params = session_get_cookie_params();
setcookie(session_name(), '', time() - 42000,
        $params["path"], $params["domain"],
        $params["secure"], $params["httponly"]);
```

Datenbankanbindung

Früher mysql_* funktionen

- Nachteil: funktioniert nur mit MySql, Fehleranfällig (SQL Injection)

PDO: Wrapper Klasse für Datenbankzugriffe

- Verbindungsaufbau

```
$dsn = 'mysql:dbname=testdb;host=127.0.0.1';
$user = 'dbuser';
$password = 'dbpass';
try {
    $dbh = new PDO($dsn, $user, $password);
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
}
```

PDO - Verbindung

- **Zunächst Verbindungsaufbau: Die Kommunikation mit dem Datenbankserver erfolgt übers Netzwerk**
- **PDO benötigt daher im Konstruktor Informationen über**
  - Art der Datenbank (MySQL, Oracle, MSSQL…)
  - IP des Datenbankservers (Häufig: localhost)
  - Name der Datenbank
  - Benutzername und Passwort
- **Die Ersten drei Informationen werden in einem Parameter (Data Source Name, DSN) zusammengefasst:**

```
mysql:dbname=testdb;host=127.0.0.1
```

PDO - Konstruktoraufruf

```
PDO::__construct() ( string $dsn [, string $username [,
  string $password [, array $driver_options ]]] )
```

- Zusammen mit den anderen Parametern kann dann ein neues PDO erstellt werden:

```
$dsn = 'mysql:dbname=testdb;host=127.0.0.1';
$user = 'dbuser';
$password = 'dbpass,;
$dbh = new PDO($dsn, $user, $password);
```

- Der Konstruktor wirft eine Exception, wenn die Verbindung fehlschlägt

PDO – Wichtige Funktionen

```
string PDO::lastInsertId ([ string $name = NULL ] )
```

- Gibt die Zeilennummer des letzten INSERTs zurück.
- Praktisch, wenn direkt eine Abfrage gemacht werden muss, die den Index benötigt.

```
PDOStatement PDO::prepare ( string $statement )
```

- Übertragt eine SQL-Abfrage an den Datenbankserver
- Dazu gleich mehr…

PDO – SQL Abfragen

- Bevor eine SQL Anweisung ausgeführt wird, wird diese an den Server übertragen
- In der SQL Anweisung können Variablen definiert werden, die Später ersetzt werden
- Nachdem die Variablen besetzt wurden, wird das Statement ausgeführt
- Das Ergebnis kann dann Zeilenweise abgerufen werden
- PDO unterstützt sowohl Abfragen als auch Datenmanipulationen

PDO – Statementhandler

- Eine wichtige Klasse im Zusammenhang mit dem PDO ist das `PDOStatement`
- `PDOStatement` repräsentiert eine an den Datenbankserver geschickte SQL Abfrage
- `PDOStatement PDO::prepare ( string $statement )`
- Das `PDOStatement` wird von PDO erstellt, wenn eine SQL-Abfrage an den Datenbankserver geschickt wird
- Für das Ersetzen der Variablen wird dann das `PDOStatement` verwendet

PDO – Statementhandler II

```
bool PDOStatement::bindValue ( mixed $parameter , mixed $value )
```

- Ersetzt für eine Variable in dem Statement durch den angegebenen Wert
- Variablen werden in den statements duch einen vorangestellten Doppelpunkt gekennzeichnet
- Als Weiterer Parameter (aus Platzgründen nicht aufgelistet) kann der Datentyp übergeben werden

```
bool PDOStatement::execute ([ array $input_parameters ] )
```

- Führt die SQL-Abfrage aus
- Variablen werden entsprechend den Aufrufen von bindValue ersetzt

PDO – Statementhandler III

RWTH AACHEN UNIVERSITY

`mixed PDOStatement::fetch (int $fetch_style)`

- Holt eine einzelne Reihe des Resultats
- Kann nur verwendet werden, wenn vorher `execute` genutzt wurde
- Funktioniert nur bei `SELECT` Abfragen
- Der Parameter `fetch_style` gibt an wie das Ergebnis geholt wird:
  - Als Array mit numerischem Index (in der Reihenfolge der Spalten)
  - Als Array mit assoziativem Index (Name der Spalte => Wert)
  - Als Objekt mit Parametern, entsprechend der Spalten

`array PDOStatement::fetchAll (int $fetch_style)`

- Wie `fetch`, jedoch werden alle Zeilen vom Datenbankserver abgefragt
- Achtung! Bei großen Abfrageergebnissen kann dies Sehr langsam sein.

PDO – Statementhandler IV

```
bool PDOStatement::closeCursor ( void )
```

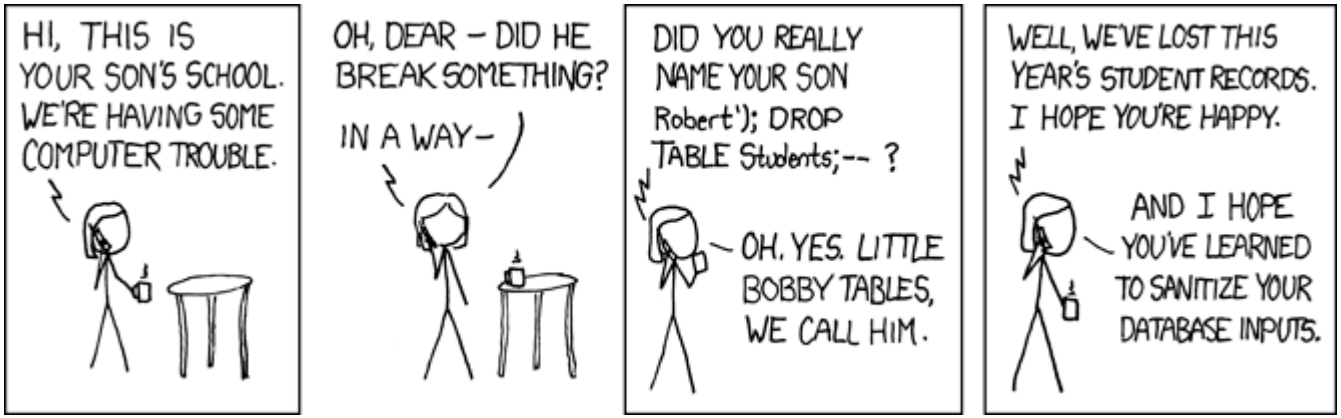
- Beendet die Anfrage am Datenbankserver
- Gibt angeforderte Ressourcen wieder frei
- Muss aufgerufen werden, bevor ein `PDOStatement` erneut ausgeführt werden kann. Achtung! Auch bei UPDATEs, INSERTs …

PDO – Beispiel

```
$calories = 150;
$colour = 'red';

$sth = $dbh->prepare(
    'SELECT name, colour, calories
     FROM fruit
     WHERE calories < :calories AND colour = :colour');

$sth->bindParam(':calories', $calories);
$sth->bindParam(':colour', $colour);

$sth->execute();
while($row = $sth->fetch()){
...
}
```

SQL-Injection

- **Top 1 der Gefährlichsten Programmierfehler**
  http://cwe.mitre.org/top25/



- **Kann aber einfach verhindert werden**

- **Bei der Benutzung von `bindParam` werden Inhalte, die das Statement verändern automatisch Ersetzt**

SQL-Injection – Was ist das?

- Bei Verarbeitung von z.B. Formulardaten in einer SQL-Abfrage können vom User Angriffe auf die Datenbank gestartet werden.
- Beispiel:

```
"SELECT id FROM users WHERE user='".$_GET["user"]."'
AND passwd='".$_GET["password"]."'LIMIT 0,1"

SELECT id FROM users WHERE user='admin'/*'
AND passwd='' LIMIT 0,1
```

Sicherheitsprobleme

SQL-Injection

- Beispiele bereits im Datenbankteil
- Kann zum Verlust aller Daten führen (DROP TABLE)
- Bei Benutzung des PDO kann dieser Fehler weitestgehend verhindert werden.

- **Einbettung von Fremden Skripten in Webseiten anderer**
- **Kann zum Abgreifen von Passworten verwendet werden**
- **Im einfachsten Fall wird ein `script`-Tag in einem Kommentarfeld gepostet:**
  - Beispiel: http://ha.ckers.org/xss.html

Passwortverschlüsselung

- Für den Login Vorgang müssen Passworte in der Datenbank abgelegt werden
- Passworte im Klartext liefern eine Angriffsfläche für Hacker
- Passworte sollten daher immer verschlüsselt gespeichert werden
- PHP liefert verschiedene „one-way Verschlüsselungsverfahren“
- `sha1`, `crypt`

Regular Expressions

Regular Expressions – Steuerzeichen

- **Suchen von bestimmten Zeichenketten in Strings**
- **Position im Text**
  - ^ Anfang des Texts „^MATSE“
  - $ Ende des Texts „Auszubildender$“
- **Platzhalter**
  - . Beliebiges Zeichen „.ATSE“
- **Escape Zeichen**
  - \ Escape Zeichen „\.“

Regular Expressions – Steuerzeichen

Anzahl

+	ein Mal oder öfter	„a+“
*	0-mal oder öfter	„a*“
?	0-Mal oder ein Mal	„hallo?“
{n}	n-Mal	„ab{2}“
{n,m}	zwischen n und m-Mal	„ab{3,5}“
{n,}	mindestens n-Mal	„ab{6,}“

Regular Expressions – Steuerzeichen

Gruppen

()	Konjunktion	„a(bc)“
		„a(bc){1,3}“
[ ]	Disjunktion	„[ab]“
		„[a-z]“
	nicht	„[^A-Z]“

Oder

|	Entweder Oder	„(abc)|(xyz)“
		„(a|b)*c“

Regular Expressions mit PHP

- `preg_split( string $pattern , string $subject , ...)`
  - Teilt den `string` in `$subject` an Zeichen, die mit `$pattern` übereinstimmen.
- `preg_match( string $pattern , string $subject ...)`
  - Überprüft, ob eine RegEx auf einen String zutrifft.
- **ACHTUNG! Bei Benutzung der PHP preg_* Funktionen muss die Regular Expression in Trennzeichen eingefasst werden:**
  - Aus „a(ab)*“ wird z.B. „|a(ab*)|“
  - Hinter dem letzten Trennzeichen können weitere Argumente übergeben werden: http://www.php.net/manual/en/reference.pcre.pattern.modifiers.php

Klasse RegExp

- Spezielles Literal (/pattern/flags):
- `var re = /a(ab)*/i;`
- `re.exec(string)`
- `re.test(string)`

```
var match = /s(amp)le/i.exec("Sample text")
//match == ["Sample","amp"]
```

Klasse String

- `String.match(pattern)`
- `String.search(pattern)`
- `String.replace(pattern)`
- `String.split(pattern)`

```
var idx = "Watch out for the rock!".search(/for/)
//idx == 10
```