PHP

The PHP Hypertext Processor (version 5.0 and above)

Aplicações para a Internet

Engenharia Informática, 2014/2015

Copyright

Author(s):

- **Vítor Carreira (**vitor.carreira@ipleiria.pt**)**

Contributor(s):

- **Fernando Silva (**fernando.silva@ipleiria.pt**)**
- **Luís Marcelino (**luis.marcelino@ipleiria.pt**)**
- **Alexandrino Gonçalves (**alex@ipleiria.pt**)**
- **Ana Nogueira**
- **João Real**
- **Michael Pinheiro**
- **Norberto Henriques (**norberto.henriques@ipleiria.pt**)**

Revised on: March, 2015

- **Fernando Silva (**fernando.silva@ipleiria.pt**)**

Syllabus

- Dynamic web page
- Server-side languages
- PHP
- Server-side Web frameworks

Dynamic web page

A dynamic web page is a kind of web page where information is prepared (fetched/created/aggregated) in real time according to information available at the moment, context, user preferences or a combination of all.

Server-side languages

- Common Gateway Interface
  - Generic Languages (e.g. C/C++, etc)
  - Scripting languages (e.g. Perl, Python, etc)
- Languages/frameworks design to produce HTML: PHP, ASP.NET, JSP/JSF, etc

Web frameworks

- Web applications are successfully replacing traditional client-server desktop applications
- Most popular web development frameworks (no order implied):
  - PHP
  - ASP.NET (integrated with Visual Web Developer for RAD)
  - JEE (Java Enterprise Edition) - multi-tier framework:
    - JSP (JavaServer Pages) / JSF (JavaServer Faces) – HTML generation (view tier)
    - Servlets - application control tier
    - EJBs (Enterprise Java Beans) – business rules tier
    - JPA (Java Persistence API) - persistence tier
  - Ruby on rails - Model-View-Controller framework based on the popular Ruby language

ASP.NET

ASP.NET (Active Server Pages) is a web application framework developed by Microsoft that allows to build dynamic web sites, web applications and web services.

```
<%@ Master Language="C#" CodeFile="mainPage.cs” Inherits="MainPage" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN“
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server“><title>Sample Page</title></head>
    <body>
        <form id="form1" runat="server“>
          <div class="templateMenu“>
           <asp:Menu ID="mainMenu" runat="server”
                  DataSourceID="siteMapDataSource”></asp:Menu>
          </div>
        </form>
    </body>
</html>
```

JSP (JEE)

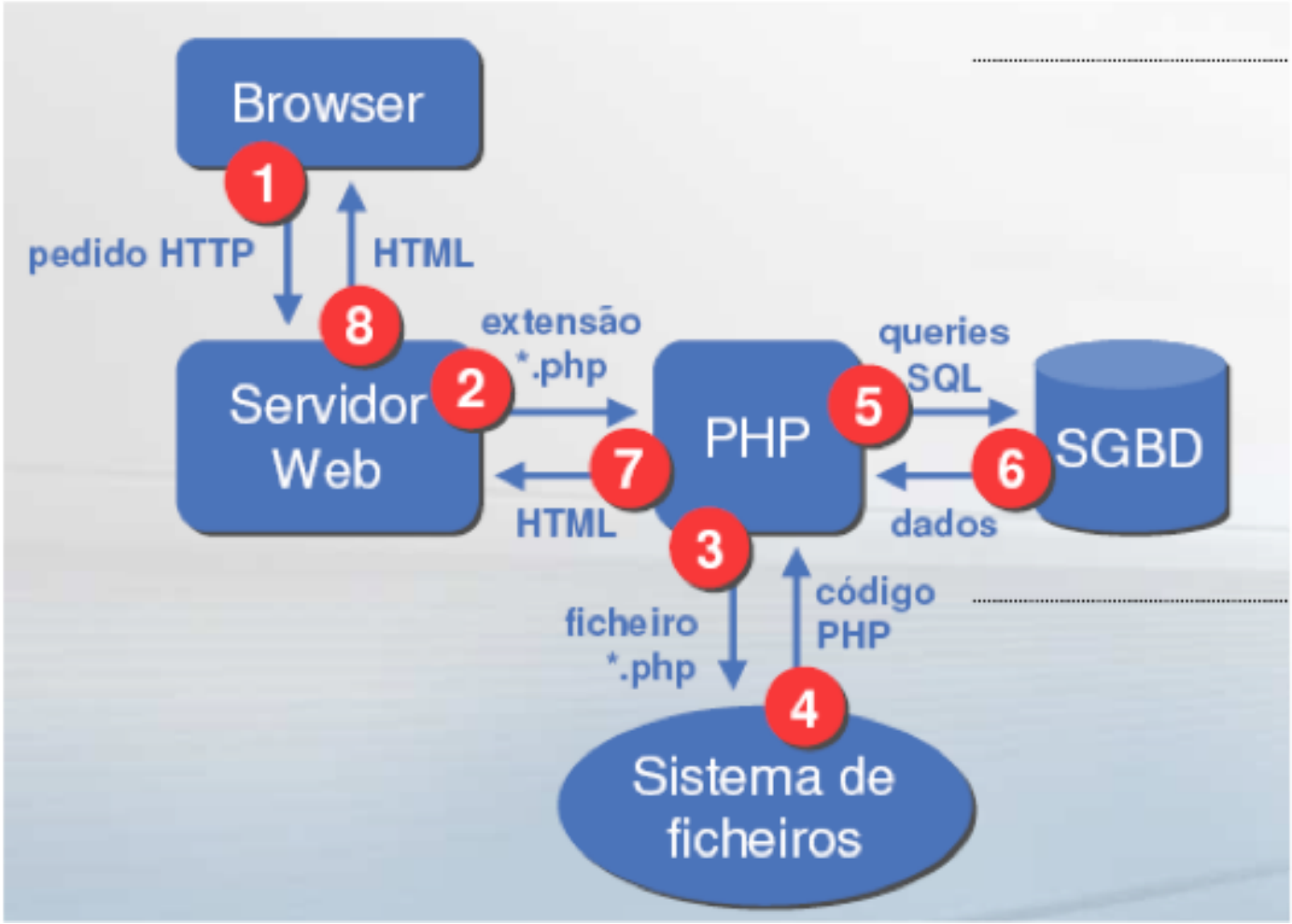
JSP (JavaServer Pages) is a Java technology developed by former Sun Microsystems for building dynamic web sites. JSP is just a component of a more broader framework for enterprise applications called JEE.

```
<%@ page import="java.util.*,MyLocales" %>
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server"><title>Sample Page</title></head>
    <body>
        <jsp:include page="mycommon.jsp" >
        <jsp:param name="extraparam" value="myvalue" />
          </jsp:include>name:<%=request.getParameter("extraparam")%>
    </body>
</html>
```

PHP: PHP Hypertext Preprocessor

- Server-side scripting language (can also be used for desktop applications)
- Supports both procedural and object-oriented paradigms
- Specially designed for dynamic web page creation
- Cross operating systems (windows, linux, osx)
- Supported on a diversity of web servers (apache, IIS, etc)
- Support for multiple DBMS (MySql, Oracle, SQLServer, etc)

PHP: Architecture



PHP: Hello World

- PHP tags and comments

```
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
      <?php
          error_reporting(E_ALL);
          //This is a comment
          echo '<p>Hello World!</p>';
          # This is another comment
          /* This is a comment block */
          phpinfo();
      ?>
  </body>
</html>
```

PSR-1 (Basic Coding Standard):

- Files MUST **use only <?php** and **<?= tags**

- PHP code MUST use **only UTF-8 without BOM**.

PSR-2 (Coding Style Guide)

Code MUST use **4 spaces for indenting, not tabs**.

PHP: Variables 1/2

- No need to declare variables (just assign a value)
- Weakly typed
- Variable's naming rules:
  - MUST start with $
  - Can include letters, numbers and _
  - Cannot start with digits
  - Case sensitive
- Use unset(<varname>) to explicit destroy a variable (e.g. unset($age);)

```
$age = 12;
$price = 2.55;
$number = -2;
$name = "Jones";
$logic = true;
```

PHP: Variables 2/2

```
$a = "hello";
$$a = "world";
echo "$a ${$a} <br />";
echo "$a $hello <br />";
// They have the same output
```

PHP: Constants

- Constants are specified using keyword **define**

```
define("COMPANY_NAME","ABC Pet Store");
define("AGE",29);
...
echo COMPANY_NAME;
echo AGE;
```

PSR-1 (Basic Coding Standard):

Constants MUST be declared **in all upper case with underscore separators**

PHP: Operators

- Arithmetic operators: +, -, *, /, %
- Comparison operators: ==, >, <, >=, <=, != (or <>), ===, !==
- Logical operators: && (and), || (or), !
  - and/or have a lower precedence than &&/||

```
$result = (1 + 2) * 4 + 1
$weather == "raining"
$age < 13
```

PHP: Strings 1/2

- Strings are a sequence of characters enclosed by “ or ‘
- String concatenation operator: .
- String functions: strlen, strpos, implode, etc

```
$string = 'Hello World!';
$string = 'It is Tom\'s house';
$string1 = 'Hello';
$string2 = 'World!';
$stringall = $string1.' '.$string2;
echo strlen("Hello world!");
echo strpos("Hello world!","world");
```

PHP: Strings 2/2

- Strings enclosed by “ are interpreted
- Strings enclosed by ‘ are not interpreted
- Special chars: \n, \t, etc

```
$age = 12;

echo 'The age is $age';
// Prints “The age is $age”

echo "The age is $age";
// Prints “The age is 12”
```

PHP: Date/Time 1/2

- Current time and date

```
$today = time();
```

- Date/time to string

```
$cdate = date("d/m/y", $today);
$ctime = date("G:i:s", $today);
```

PHP: Date/Time 2/2

- String to date/time

```
$importantDate = strtotime("tomorrow");
$importantDate = strtotime("now + 24 hours");
$importantDate = strtotime("last Saturday");
$importantDate = strtotime("8pm + 3 days");
$importantDate = strtotime("2 weeks ago");
$importantDate = strtotime("this 4am");
```

- Date operations

```
// Difference between dates in seconds
$timeSpan = $today - $importantDate;
```

PHP: Data types 1/4

- Like Javascript, PHP is weakly typed
- Data type is inferred by the value assigned to the variable
- Internal data types: Integer, Float, String, Boolean, Array and Object
- Special data types:
  - null (no value assigned)
  - resource (represents an handler to external resources like opened files, database connections, etc…)

PSR-2 (Coding Style Guide)

- PHP **keywords MUST be in lower case**.
- The PHP **constants true, false, and null MUST be in lower case**.

PHP: Data types 2/4

- Data type conversion and test functions:
  - string **gettype**(mixed var) - get var's data type;
  - bool **settype**(mixed var, string type) - change var's data type;
  - **is_array**() - Checks whether the variable is an array;
  - **is_double**(), **is_float**(), **is_real**() (identical functions) - Checks whether the variable is a float;
  - **is_long**(), **is_int**(), **is_integer**() (identical functions) - Checks whether the variable is an integer;
  - **is_string**() - Checks whether the variable is a string;

PHP: Data types 3/4

- Data type conversion and test functions:
  - **is_bool**() - Checks whether the variable is a boolean;
  - **is_object**() - Checks whether the variable is an object;
  - **is_resource**() - Checks whether the variable is a resource;
  - **is_null**() - Checks whether the variable is null;
  - **is_scalar**() - Checks whether the variable is a scalar, that is, an integer, boolean, string, or float;
  - **is_numeric**() - Checks whether the variable is any kind of number or a numeric string;
  - **is_callable**() - Checks whether the variable is the name of a valid function;

PHP: Data types 4/4

- Test/change variable state:
  - bool **isset**(mixed var) – returns true if the variable var is defined;
  - void **unset**(mixed var) – destroys the variable var;
  - bool **empty**(mixed var) – returns true if the variable var does not exist or is not initialized;
- Conversion functions:
  - int **intval**(mixed var[, int base]) - converts var to an int value;
  - float **floatval**(mixed var) - converts var to a float value;
  - string **strval**(mixed var) - converts var to a string value;

PHP: Control structures

According with PSR-2 – Coding Style Guide, the general style rules for **control structures** are as follows:

- There MUST be one space after the control structure keyword
- There MUST NOT be a space after the opening parenthesis
- There MUST NOT be a space before the closing parenthesis
- There MUST be one space between the closing parenthesis and the opening brace
  - Opening braces MUST go on the same line
- The structure body MUST be indented once
- The closing brace MUST be on the next line after the body

- The body of each structure MUST be enclosed by braces. This standardizes how the structures look, and reduces the likelihood of introducing errors as new lines get added to the body.

http://www.php-fig.org/psr/psr-2/

PHP: Control structures

- If statement

```
if ($country == "Germany") {
    $message = "Willkommen!";
} elseif ($country == "France"){
    $message = "Bienvenue!";
} else{
    $message = "Welcome!";
}
echo "$message<br />";
```

PSR-2 (Coding Style Guide)

The keyword **elseif SHOULD be used instead of else if** so that all control keywords look like single words.

PHP: Control structures

- Switch statement

```
switch ($country) {
    case "Germany":
        $salestaxrate = 0.19;
        break;
    case "France":
        $salestaxrate = 0;
        // no break;
    case "Great Britain":
        $salestaxrate = 0.20;
        break;
    case "Greece":
    case "Portugal":
        $salestaxrate = 0.23;
        break;
    default:
        $salestaxrate = 0.19;
        break;
}
$salestax = $orderTotalCost * $salestaxrate;
```

PSR-2 (Coding Style Guide)

The case statement MUST be indented once from switch, and the break keyword (or other terminating keyword) MUST be indented at the same level as the case body. There MUST be a comment such as *// no break* when fall-through is intentional in a non-empty case body.

PHP: Control structures

- While statement

```
while ($testvar != "yes") {
    if ($customers[$k] == "Smith") {
        $testvar = "yes";
        echo "Smith<br />";
    } else {
        echo "$customers[$k], not Smith<br />";
    }
    $k++;
}
// break and continue statements behave like in C
```

PHP: Control structures

- Do...while statement

```
do {
    if ($customers[$k] == "Smith") {
        $testvar = true;
        echo "Smith<br />";
    } else {
        echo "$customers[$k], not Smith<br />";
    }
    $k++;
} while ($testvar);

// break and continue statements behave like in C
```

PHP: Control structures

- For statement

```
for ($i = 0; $i < count($customerNames); $i++) {
    echo "$customerNames[$i]<br />";
}

for ($i = 0, $j = 1; $t <= 4; $i++, $j++) {
    $t = $i + $j;
    echo "$t<br />";
}

// The count function returns the number of elements in an array
```

PHP: Arrays 1/8

- Simple arrays

```
$animals[0] = "cat";
$animals[1] = "tiger";
$animals[2] = "elephant";
$animals = array("cat","tiger","elephant");
for ($i = 0; $i < count($animals); $i++) {
    echo $animals[$i]." ";
}
foreach ($animals as $animal) {
    echo $animal. ' ';
}
```

NOTE:
Since PHP 5.4 it is possible to define an array in a abreviated and more practical way:

```
$animals = ['cat','tiger','elephant'];
```

```
$animals = array("cat","tiger","elephant");
$animals[20] = 'dog';
count($animals); // Returns 4
end($animals); // returns "dog"
key($animals); // Returns 20
```

PHP: Arrays 2/8

- Associative arrays (keys can be strings or integers)

```
$airlines['BA'] = "British Airways";
$airlines['LH'] = "Lufthansa";
$airlines['AF'] = "Air France";


$airlines = array("BA" => "British Airways",
"LH" => "Lufthansa", "AF" => "Air France");
```

NOTE:
Since PHP 5.4 it is possible to define an associative array in a abreviated and more practical way:

```
$airlines= [‘BA’ => “British Airways”, ‘LH’ => “Lufthansa”,
‘AF’ = “Air France”];
```

PHP: Arrays 3/8

- Foreach and associative arrays

```
$airlines = array("BA" => "British Airways","LH" =>
"Lufthansa", "AF" => "Air France");

foreach ($airlines as $symbol => $name) {
    echo "$name ($symbol)<br />";
}
```

PHP: Arrays 4/8

- Sort operations on simple arrays

```
sort($animals);

rsort($animals); // Reverse sort
```

- Sort operations on associative arrays:
- By value: `asort, arsort` (reverse order)
- By key: `ksort, krsort` (reverse order)
- Custom sort: `usort(<array>, <sort function>)`

PHP: Arrays 5/8

- Arrays support iterators: `reset()`, `current()`, `prev()`, `next()`, `end()`, `sizeof()`

```
reset($airlines); // moves to the first element

$value = current($airlines);

echo "$value<br />"; // current array element

$value = next($airlines);

echo "$value<br />";

$value = next($airlines);

echo "$value<br />";
```

PHP: Arrays 6/8

Table 3.1 **PHP's Array Operators**

Operator	Name	Example	Result
`+`	Union	`$a + $b`	Union of `$a` and `$b`. The array `$b` is appended to `$a`, but any key clashes are not added.
`==`	Equality	`$a == $b`	True if `$a` and `$b` contain the same elements.
`===`	Identity	`$a === $b`	True if `$a` and `$b` contain the same elements, with the same types, in the same order.
`!=`	Inequality	`$a != $b`	True if `$a` and `$b` do not contain the same elements.
`<>`	Inequality	`$a <> $b`	Same as `!=`.
`!==`	Non-identity	`$a !== $b`	True if `$a` and `$b` do not contain the same elements, with the same types, in the same order.

PHP: Arrays 7/8

- Multi-dimensional arrays

```
$productPrices['clothing']['shirt'] = 20.00;
$productPrices['clothing']['pants'] = 22.50;
$productPrices['linens']['blanket'] = 25.00;
$productPrices['linens']['bedspread'] = 50.00;
$productPrices['furniture']['lamp'] = 44.00;
$productPrices['furniture']['rug'] = 75.00;...
$shirtPrice = $productPrices['clothing']['shirt'];
```

PHP: Arrays 8/8

- Multi-dimensional arrays

```
<?php
echo "<table border=1>";
foreach ($productPrices as $category) {
    foreach ($category as $product => $price) {
        $f_price = sprintf("%01.2f", $price);
        echo "<tr>";
        echo "<td>$product</td>";
        echo "<td>$f_price</td>";
        echo "</tr>";
    }
}
echo "</table>";
?>
```

PHP: Functions 1/4

- Syntax: `function <name>(args) { //code }`

```
function add_two_numbers($num1 = 1, $num2 = 1)
{
    $total = $num1 + $num2;
    return $total;
}
echo add_two_numbers().'<br />';
echo add_two_numbers(3).'<br />';
echo add_two_numbers(4,2).'<br />';
```

PSR-2 (Coding Style Guide)

When making a method or function call, there MUST NOT be a space between the method or function name and the opening parenthesis, there MUST NOT be a space after the opening parenthesis, and there MUST NOT be a space before the closing parenthesis.

In the argument list, there MUST NOT be a space before each comma, and there MUST be one space after each comma.

PHP: Functions 2/4

- Local vs Global vars

```
<?php
$VAT = 0.25; // Global var
function cost_with_vat($cost)
{
    global $VAT; // references global var
    $total_with_vat = $cost + $cost * $VAT;
    return $total_with_vat;
}

$total = cost_with_vat(199.99);
?>
```

PHP: Functions 3/4

- Pass by reference

```
<?php
function increment(&$value, $amount = 1)
{
    $value = $value + $amount;
}
$a = 10;
echo $a.'<br />';
increment($a, 5);
echo $a.'<br />';
?>
// In PHP arrays are not passed by reference by
default
```

PHP: Functions 4/4

- A function can return any value (even an array)

```
<?php
function mult_return()
{
    $a = 10;
    $b = $a * 5;
    $c = "AINet classes are cool";
    return array($a,$b,$c);
}
$arr = mult_return();
for ($i = 0; $i < count($arr); $i++) {
    echo $arr[$i] . "<br />";
}
// or
list($var1, $var2, $var3) = mult_return();
echo "$var3<br />$var2<br />$var1<br />";
?>
```

PHP: Code reuse 1/2

- PHP promotes code reuse:
  - **include**(<file>) – triggers a warning if the file doesn't exist
  - **require**(<file>) – throws an error if the file doesn't exist
  - **require_once**(<file>) – identical to *require()* except PHP will check if the file has already been included, and if so, it will not include (require) it again. Recommended for bootstrapping code.

PHP: Code reuse 2/2

```
<?php
require('header.php');
?>
<!-- page content -->
<p>Welcome to the home of AINET.</p>
<?php
require('footer.php');
?>
```

PSR-1 Notes on Side Effects

PSR-1 (Basic Coding Standard):

A file SHOULD declare new symbols (classes, functions, constants, etc.) and cause no other side effects, or it SHOULD execute logic with side effects, **but SHOULD NOT do both**.

- “Side effects" means execution of logic not directly related to declaring classes, functions, constants, etc., merely from including the file
- "Side effects" include but are not limited to:
  - Generating output, explicit use of require or include, connecting to external services, modifying ini settings, emitting errors or exceptions, modifying global or static variables, reading from or writing to a file, and so on

References

- PHP and MySQL Web Development (4th Edition)
  - Luke Welling and Laura Thomson, Addison-Wesley 2009
- http://www.w3schools.com/php/
- PHP Documentation
  - Manual: http://www.php.net/manual/en/
  - Function reference: http://www.php.net/manual/en/funcref.php
- PHP PSR Standards
  - http://www.php-fig.org/psr/psr-1/
  - http://www.php-fig.org/psr/psr-2/