Internet, Intranet and Web — Lecture III

CSS, and Server Side Web Technologies

Marco Solieri marco.solieri@lipn.univ-paris13.fr
Info et Réseaux en Apprentissage, Sup Galilée, Université Paris 13

November 7th, 2014

Outline

Contents

1 CSS 3: presentational markup 1
1.1 History . . . 1
1.2 Syntax . . . 2
1.3 Selectors . . . 2
1.4 Properties . . . 3
1.5 Style and XHTML . . . 3
1.6 Cascade . . . 4

2 Web server 4

3 PHP language 5
3.1 Introduction . . . 5
3.2 Scripts and HTML documents . . . 5
3.3 Base syntax and semantics . . . 6
3.4 Flow control . . . 7
3.5 Modularization and object-orientation . . . 8

4 PHP and HTTP 9
4.1 HTTP request access . . . 9
4.2 HTTP trasmission management . . . 9
4.3 HTTP reply management . . . 9
4.4 HTTP state management . . . 10

5 PHP and storage 10
5.1 MySQL interface . . . 10
5.2 PDO interface . . . 11

1 CSS 3: presentational markup

1.1 History

Birth of presentation markup for Web
Presentation inside HTML

- Mosaic browser protype: font size and name
- Netscape 1.0: first tags for font settings, centering . . .
- HTML 2.0 - HTML 3.2: many tags for font style, colors . . .

Separation between other markup

- Bos and Lie proposed a language for styling HTML pages: CSS
- Idea: multiple style sheet acting in chain (cascading!)
- W3C working group for standardization

Killer features:

- better control of HTML presentation for authors and users,
- independence from the specific (X)HTML version.

CSS versions

Level 1
- W3C Recommendation in 1996;
- visual formatting: presentation for browser rendering on screen.

Level 2
- W3C Recommendations in 1998 and 2011;
- multiple media presentation, more complex layout management.

Level 3
- W3C Recommendation in 2011;
- extensions to layout, colors, borders, animations, transformations . . . ;
- support by browser: still not complete, quite heterogeneous; need to be checked:
  - http://caniuse.com/
  - http://www.w3schools.com/css3/

Level 4
- still in development, not supported at all.

1.2 Syntax

Declarations

Definition 1 (Property). Style trait assignable to an element.

Definition 2 (Declaration). Statement defining the value of a property, in the form:

```
property: value
```

Example 3 (Style declarations).

```
        color: green;
        font-family: Arial;
        font-size: 12 pt;
        margin-left: 15 pt;
```

Rules

Definition 4 (Selector). Statement defining a set of elements in the (X)HTML tree.

Definition 5 (Rule). Selector(s) followed by a block of declarations, in the form:

```
selector , ... {
    declaration ;
    ... }
```

Example 6 (Rules).

```
p { font-family: Arial; font-size: 12 pt; }
h1, h2, h3 { color: red; margin-left: 15 pt; }
```

1.3 Selectors

Base and structural selectors

Base selectors:

Universal `*`

Type `Element`

Class `Element.classname`

ID `Element#id`

Structural combinators:

Child `Element1 > Element2`

Descendent `Element1 Element2`

Adjacent `Element1 ~ Element2`

Successor `Element1 + Element2`

Pseudo classes selectors: structure and links

Document structure:

Cardinality `Element:nth-child(n)`, `Element:nth-of-child(n)`

Uniqueness `Element:only-child(n)`, `Element:only-of-type(n)`

Emptiness `Element:empty`,

Link:

Unvisited link `Element:link`

Visited hyperlink `Element:visited`

Pseudo classes selectors: user and interface

User actions:

Activation `Element:active`

Pointing `Element:hover`

Focusing `Element:focus`

Interface behaviour:

Enabling `Element:enabled`

Disabling `Element:disabled`

Attributes and pseudo element selectors

Attributes:

Attribute ownership `Element[attribute]`

Value is `Element[attribute="value"]`

Value list contains `Element[attribute~="value"]`

Value contains `Element[attribute*="value"]`

Generation time orderings:

Successor `Element::before`

Predecessor `Element::after`

Pseudo elements:

First line `Element::first-line`

First letter `Element::first-letter`

Meta selectors (the so-called at rules)

Seven selectors for defining meta rules

1. `@charset` character set encoding,
2. `@font-face` customized font, automagically downloaded,
3. `@import` rule inclusion from other stylesheets
4. `@media` media type targeting: screen, print, projection, aural ...
5. `@namespace` definition of xml namespaces
6. `@page` paged media styling: margins, padding ...
7. `@phonetic-alpabet` alphabet to be used when specifying pronounces of words

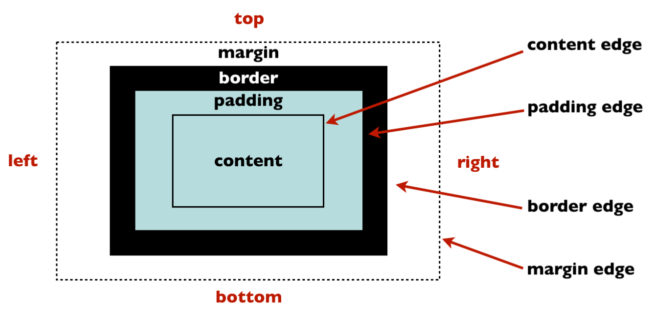
Brand new from CSS3

1.4 Properties

Visualization

Base visualization concepts:

- Every element is visualized by a box.
- Boxes contains boxes of all contained elements.
- Multiple boxes are placed according to their properties:



Text properties

Visualization properties:

- `display`: `block` for vertical order, `inline` for horizontal order.
- `float`: top-left or top-right corner, with other boxes all around.
- `position`:
  - `static` for default;
  - `absolute` or `relative` positioning, possibly with hiding overlay;
  - `fixed` for non scrolling.
- `z-index`: boxing level in the overlaying stack.
- `visibility`: hidden status, for caching hacks.

Text properties

Multicolumn layout properties (new from CSS3)

- text naturally flows, with adaptive column number,
- some properties: `column-width`, `column-gap`, `column-rule`

Text properties:

- `font-family`: name of the font (e.g. Dejavu Sans, Tahoma)
- `font-style`: normal, italic, oblique
- `font-weight`: normal, bold, bolder, lighter
- `font-stretch`: normal, wider, narrower, condensed, expanded
- `text-indent`, `text-align`, `text-height`: indentation, alignment, interline
- `text-decoration`: none, underline, overline, blink
- `letter-spacing`, `word-spacing`
- `text-tranform`: none, capitalize, uppercase, lowercase

1.5 Style and XHTML

Specifying the style

Four styles of choosing the style:

1. declarations inside a `<style>` attribute in the HTML element,
2. rules inside a `<style>` element,
3. importing stylesheet from a `<style>` element,
4. linking stylesheet from a `<link>` element.

Three granularity levels of styling:

1. type of element, e.g. all the paragraphs;
2. category, e.g. all the paragraphs with some `class` attribute;
3. element, e.g. the paragraph with some unique `id` attribute.

Examples of internal style

Example 7 (Declaration inside paragraph element (1)).

```
    <p id="p1" style="color: red;">
      ...
```

Example 8 (Rule inside the document (2)).

```
  ...
  <style type="text/css">
    #p1 { color: red; }
    ...
  </style>
</head>
...
    <p id="p1">
      ...
```

Examples of external style

Example 9 (Importing from a style element (3)).

```
  ...
  <style type="text/css">
    @import url(/style/external.css)
  </style>
</head>
...
    <p id="p1"> ...
```

Example 10 (Linking from a a style element (4)).

```
  ...
  <link type="text/css" rel="stylesheet"
    href="/style/external.css"/>
</head>
...
    <p id="p1"> ...
```

1.6 Cascade

Cascade

Stylesheet composition is

- sequential: there is a specified order in the chain,
- possibly overlapping: attributes can be rewritten.

Chain ordering idea:

- (most of) undeclared properties are defaulted;
- ordering by media type;
- ordering by importance flag;
- ordering by source: user, author, user-agent;
- ordering by specificity: minimum inheritance;
- ordering by writing placement.

Cascading style

Example 11 (Style overwriting). Three stylesheets state the following rules, already ordered as:

```
p { font-family: Arial; font-size: 12 pt; }
p { color: red; font-size: 11 pt; }
p { margin-left: 15 pt; color: green;}
```

then resulting properties are:

```
p { font-family: Arial;
    font-size: 11 pt;
    margin-left: 15 pt;
    color: green;
}
```

2 Web server

Web server's duties, basics

File serving:

- map URL path into local directories,
- receive request, send appropriate reply
- redirection.

Access control for resources:

- set permissions: who can do what?
- realize authentication.

SSL/TLS underlay:

- session establishing,
- certificate selection.

Logging

- requests and replies statuses,
- errors.

Web server's duties, with dynamic web

Server-side processing

- interface between resources and script execution
- programming interface to HTTP protocol
- invoking script interpretation

URI management

- multiple domain handling,
- path and query rewriting.

Example 12 (URL rewriting). Before:

```
http://blog.fr/phpblog/index.php?
  id=20120531-2&type=post&user=antoine
```

After: `http://blog.fr/antoine/post/2012/05/31/2`

CGI: Common Gateway Interface

The server as a gateway:

1. receives a HTTP request
2. sets up environment variables, i.e. the various parts of request
3. spawns the interpreter on the script
4. waits for HTTP reply and forwards it

Spread of CGI:

- Described in RFC 3875
- Most common interface implementation (it's Common, indeed)

Performance:

- CGI can be slow: a new interpreter process for every request
- FastCGI can do better:
  - the web server keep some interpreter processes ready to execute,
  - when a request is received, interpretation instantaneously starts.

3 PHP language

3.1 Introduction

Introduction

PHP:

- PHP: Hypertext Processor.
- General-purpose server-side scripting language.
- Platform support: GNU/Linux, Windows, Unixes.
- Paradigm: procedural and object-oriented.
- Web server support: Apache, Lighttpd, IIS ...
- Database system support: MySQL, PostgreSQL, Oracle ...
- Licence: free (as in freedom).
- Spread: 44M websites and 2.1M web servers

History

Birth Rasmus Lerdorf creates a language for its own web pages Personal Home Page Tools (1994)

Publication First version released (1995)

v3 New parser; community driven project (1998-2000)

v4 Increased speed, modularity, Zend Engine parser (2000-2008)

v5 New object model Zend Engine II, DOM for XML and web services, namespaces, lambda functions, late static binding, generators, JSON (2005-)

Latest Stable releses v5.6.2 (released twenty days ago)

3.2 Scripts and HTML documents

Styles of scripting

Code embedding inside HTML

- Scripts inserted as HTML elements into the documents,
- server recognition and on-the-fly elaboration.
- Good for programming in the small.

Standalone scripting

- Scripts in separated files finally producing HTML documents,
- server triggers elaboration of the requested script.
- Good for programming in the large.

Code embedding

HTML special elements:

- `<?php ... ?>` compliant to XML,
- `<script language="php"> ... </script>` not compliant to XML.

Example 13.

```
<html>
  <body>
    <?php
      echo "Hello World"
    ?>
  </body>
</html>
```

Standalone
PHP script files:

- begin with `<?php` and end with `?>`,
- everything else outputted into standard output.
- File extension `.php`.

Example 14.

```
<?php
echo ("<html>
        <body>
          Hello World
        </body>
      </html>")
?>
```

3.3 Base syntax and semantics

Basics
Basics

- Statement separation with `;`
- Comments:
  - inline, preceded by `//` or `#`
  - block, enclosed between `/*` and `*/`

Variables:

- implicit declaration, name prefix `$`
- call: by value
- assignment operator `=`
- reference operator `&=`

Example 15.

```
<?php
$var1 = "foo";
$var2 = $var1;
$var3 &= $var1;
$var1 = "bar";
echo($var2);
echo($var3); ?>
```

(Prints "foobar".)

Types

- Base types

  Scalar null, boolean, integer, float, string.

  Composite array, object.

  Special resource.

- Constructors: functions and classes.
- Type system: implicit, dynamic, weak (duck typing)
- Casting operator: `(type) $variable`

Example 16.

```
<?php
$var = "Hello";    // var : string
$var = 2;          // var : integer
?>
```

Strings
Definitions:

Literal single-quoted text.

Interpreted double-quoted text, with:

- evaluation of variables
- escape sequences: `\n \r \t \\ \$ \"` ...

Main constructs (they are immutable):

- concatenation operator: `string . string`
- concatenating assignement operator: `var .= string`
- print functions: `echo($string)` and `print($string)`

Example 17.

```
<?php
$version = 'version ' . phpversion();
echo("Server running PHP $version");
?>
```

Arrays
Indexed data as pairs $\langle key, value \rangle$

Numeric *key* is an integer.

Associative *key* is a string.

Multidimensional *value* is an array.

Definition:

- by listing

```
<?php
$numArray = array(val1, val2 /* ... */)
$assArray = array("key1"=>val1,
    "key2"=>val2 /* ... */); ?>
```

- by single values

```
<?php
$numArray[index] = val;
$assArray['key'] = val; ?>
```

Functions

Syntax of a function statement:

```
<?php
// declaration and parameters
// (possibly empty)
function functionName($arg1, /* ... */) {
  // body
  statement;
  /* ... */
  // possible output
  return $var
}
?>
```

- Defaulting of parameter, via direct assignment into declaration.
- No need for outer `;` statement separator.
- Function nesting allowed.

Scoping

- Scoping style: trivial lexical.
- Variables and their scope:

Global variable in the top-level: accessible only outside functions.

Local variable within a function: accessible only within it not in inner functions.

Parameter as local variable.

Static variable declared static within a function: as local variable, but preserved between invocations.

3.4 Flow control

Conditional execution: single guard

- Simple, with true branch only:

```
<?php
if (/* boolean expression */) {
    /* block */ } ?>
```

- Complete, with false branch too:

```
<?php
if (/* boolean expression */) {
    /* block */
} else {
    /* block */ } ?>
```

Conditional execution: multiple guards I

- Nested branching:

```
<?php
if (/* boolean expression */) {
    /* block */
} elseif  (/* boolean expression */) {
    /* block */
// ... other cases
} else {
    /* block */ } ?>
```

Conditional execution: multiple guards II

- Cases:

```
<?php
switch (/* variable */) {
  case /* value */ : /* block */
  // ... other cases
  default: /* block */ } ?>
```

Loops I

- Free condition

```
<?php
while (/* boolean expression */) {
  /* block */ } ?>
```

- Free condition, late test

```
<?php
do {
  /* block */
} while (/* boolean expression */) ?>
```

Loops II

- Variable condition

```
<?php
for (/* init */ ; /* test*/ ; /*incr */) {
  /* block */ } ?>
```

- Array condition

```
<?php
foreach ($array as $value) {
  /* block */ } ?>
```

(Break and goto statements available for writing ugly programs.)

3.5 Modularization and object-orientation

Modularization

File inclusion

- `include_path` directive in php.ini: predefined directories where to search for inclusion;
- `include()` function: file inclusion, with actual scope of the call.
- `require()` function: as include, but errors are fatal, not warning.
- Good for separating configurations and procedural programming.

Object orientation

- `class` declaration and `new` object instantiation.
- Inheritance to have class extensions.
- Good for large project structuring.

Classes and objects

- Class definition:

```
<?php
class ClassName {
  $field = value;
  function method(/* args */) {
    /* body */
  }
} ?>
```

- Object instantiation:

```
<?php $object = new ClassName; ?>
```

- Field access (notice only one dollar sign is used):

```
<?php $object->field; ?>
```

- Method invocation:

```
<?php $object->method(); ?>
```

Objects' life

Construction

- constructor method in class definition: `__construct()`,
- invoked by instantiation with `new`.

Destruction

- destroyer method in class definition: `__destruct()`,
- invoked at object death,
- automatic management with garbage collection.

Example 18.

```
<?php class MyClass {
  function __construct() {
    echo("Hello world!\n"); }
  function __destruct()  {
    echo("So soon? Aaargh.\n"); }
}
echo("new object\n");
$obj = new MyClass();
echo("another object\n");
$obj =new MyClass(); // ref. lost
$obj = null  //ref. lost again
?>
```

Inheritance and implementation

- Class inheritance from a single super-class:
  - `extends` modifier in declaration,
  - method overriding supported,
  - invocation of a superclass method with `parent::method()`.
- Class modifiers
  - `abstract`: instantiation prohibition
  - `final`: extension prohibition
- Interfaces
  - declaration:

    ```
    <?php interface ifaceName {
      method($arg1,$arg2); }  ?>
    ```

  - class implementation with `implements` modifier in class declaration

Modifiers

- Fields and methods visibility:

  `public` accessible outside the class (default)

  `private` not accessible outside the class

  `protected` accessible outside, by subclasses only

- Fields and methods staticity:
  - `static` modifier
  - belonging to class, not to instantiations
  - access with diffferent syntax:

    ```
    <?php
    $var = ClassName::$field;
    ClassName::method();
    ?>
    ```

4 PHP and HTTP

4.1 HTTP request access

HTTP request access: basics

Definition 19 (HTTP Request, recall)**. Request** MIME message:

```
Method URI HTTP_version
[ Header
]*

[ Body ]
```

URI `http[s]://auth/pa/t/h/[?query][#fragment]`

Query `parameter=value[&parameter=value]*`

Access to HTTP request

Server and client information: associative array `_SERVER[]`

- Method of the HTTP request`"REQUEST_METHOD"`
- URI of the HTTP request`"REQUEST_URI"`
- browser identification: `"HTTP_USER_AGENT"`
- browser TCP/IP info: `"REMOTE_ADDR"` and `"REMOTE_PORT"`
- ...

URL parsing:

- function `parse_url(str)` returns an associative array
- keys: scheme, host, port, user, pass, path, query, fragment

Specific information to query:

- associative arrays, with parameter of query as array key,
- `_GET[]` for GET method and `_POST[]` for POST method.

GET vs POST

Maximum length

- GET: at most 2000 characters
- POST: at most 8 megabytes

URI visibility

- GET: yes, browser change URI
- POST: no, browser does not change URI

Implementation

- GET and POST: `<form>` element, web scripting
- GET only: anchor element in HTML document, user input

4.2 HTTP trasmission management

Managing trasmission encodings

HTTP character sets

URL URL escaping, for simple strings (ASCII)

- encoding a string: `urlencode(str)`
- decoding in a string: `urldecode(str)`

Base64 6 bits text, for complex data (large charset or binary)

- encoding a string: `base64_encode(str)`
- decoding in a string: `base64_decode(str)`

4.3 HTTP reply management

Producing HTTP replies

Definition 20 (HTTP Reply: MIME syntax)**.**

```
Version Status_code Reason_phrase
[ Header: value
]*

Body
```

`Header` function

- set up the reply headers
- signature: `header(head[,replace[,code]])`

  `head` the header line (string)

  `replace` overwriting flag, default true (boolean)

  `reply_code` the HTTP reply code (int)

Body output functions: `echo(str)` and `print(str)`

Header transmission management

Alert

- PHP output transmission is a stream.
- Therefore: header output is forbidden after body output.

Interrogation about status of header transmission

- signature `header_list([&file [, &line ]])`

  `file` output: file path of transmission (string reference)

  `line` output: starting line of transmission (int reference)

  return headers transmitted? (bool)

Interrogation about current headers, sent or waiting

- function signature `headers_list()`

  return array of strings of header lines

Common header manipulation

Authentication process:

- reply codes: 401 unauthorized and 403 forbidden
- header: `WWW-Authenticate` for the challenge

Redirection:

- reply codes: 301 moved permanently and 303 see other
- header: `Location` for the target URL

File serving

- headers: `Content-type`, `Content-length`, `Content-disposition`

...

4.4 HTTP state management

Cookies management: setup

Setup function signature:

```
setcookie(name[, value[, expire[,
    path[, domain[, secure]] ]]])
```

`name` identificator (string)

`value` stored value, transparently URLencoded (string)

`expire` expiration date in seconds (int)

`path` path of validity, default '/' (string)

`domain` domain of validity, terminated with '.' (string)

`secure` need for SSL/TLS underlay? (int)

Example 21.

```
<?php setcookie("username", "Spiderman",
    time()+3600); ?>
```

Cookies management: access and deletion

Reading a cookie

- `$_COOKIE[]` associative array, with cookie's name as key

Example 22 (Welcoming piece of XHTML).

```
<?php
if (isset($_COOKIE["user"]))
    echo 'Welcome ' . $_COOKIE["user"] .
        '!<br />';
else
    echo 'Welcome anonymous!<br />'; ?>
```

Deleting a cookie

- no explicit way: make it expire

Memorandum: cookies are in HTTP headers ...

Session management

Fully-featured and automatic management

- session ID: in URI or in cookies,
- session data: in the server storage.

Session start

- manual, with function `session_start()`
- automatic, with PHP configuration `session.auto_start`

Session data access

- read from and write to `$_SESSION[]` associative array
- delete with `unset($_SESSION['key'])`

Session destroy

- with function `session_destroy()`

5 PHP and storage

Server side programming and DBMS

Separation of data from logic, the most common approach:

- logic with server-side programming
- data stored in a database management system

PHP programming interface to DBMS features:

- connection management,
- querying,
- transaction.

5.1 MySQL interface

MySQL: DBMS operations

MySQL functions:

- Connection opening and authentication

```
<?php connection mysql_connect(
    [host[,username[,password]]])?>
```

- Connection closing:

```
<?php mysql_close([conn]); ?>
```

- Database listing:

```
<?php mysql_list_dbs([conn]) ?>
```

- Database selection:

```
<?php mysql_select_db(database[, conn]) ?>
```

MySQL: querying

Asking a query:

- function

  ```
  <?php mysql_query(query [,conn]) ?>
  ```

- where query is the string containing the MySQL query, such as:
  - CREATE TABLE,
  - INSERT INTO,
  - SELECT FROM (WHERE),
  - UPDATE SET (WHERE),
  - DELETE FROM (WHERE).

Reading response

- row, in a numeric array

  ```
  <?php mysql_fetch_row(result) ?>
  ```

- row, in a associative/numeric array

  ```
  <?php mysql_fetch_array(result) ?>
  ```

MySQL: summarizing example

```
<?php
$con = mysql_connect('localhost',
    'antoine','$3cr37!');
if (!$con)
    die('Could not connect: '.
        mysql_error());
mysql_select_db("my_db", $con);
$res = mysql_query(
    "SELECT * FROM Persons");
while($row = mysql_fetch_array($res)) {
    echo $row['FirstName'] . ' '.
        $row['LastName'];
    echo "<br />";
}
mysql_close($con); ?>
```

5.2 PDO interface

PDO: PHP Data Object

- Lightweight API for unified data-access
- Not a full database abstraction
- DBMS supported with appropriate driver:
  - Firebird/Interbase
  - IBM DB2
  - IBM Informix
  - MS SQL Server
  - MySQL 3-5
  - PostgreSQL
  - ODBC (IBM DB2, unixODBC and win32 ODBC)
  - Oracle Call
  - SQLite 2-3

PDO security

Higher security:

- Parametrization of query strings variable,
- pameter passing as separated action.
- Result: SQL injection no more.

Example 23 (Secure querying with PDO).

```
<?php
$preparedStmt = $db->prepare(
    "SELECT * FROM users WHERE firstName = :name"
// ...
$name = $someUserInputCall;
$preparedStmt->bindParam(':name', $name,
    PDO::PARAM_STR);
?>
```