

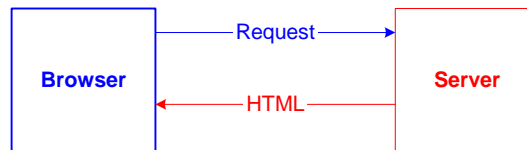
Some Basics on Web Design

Loi Ngoc Nguyen, Duy-Ky Nguyen PhD
© All Rights Reserved

2007-04-30

| | |
|---|----------|
| SOME BASICS ON WEB DESIGN..... | 1 |
| 1. HTML | 1 |
| 2. CASCADING STYLE SHEETS (CSS)..... | 2 |
| 2.1. INLINE STYLES | 2 |
| 2.2. EMBEDDED STYLES..... | 2 |
| 2.3. EXTERNAL STYLE SHEETS STYLES | 3 |
| 3. JAVASCRIPT | 3 |
| 4. COMMON GATEWAY INTERFACE (CGI)..... | 4 |
| 4.1. OS SHELL | 4 |
| 4.2. C LANGUAGE..... | 4 |
| 4.3. PERL LANGUAGE | 5 |
| 4.4. PHP LANGUAGE | 6 |
| 4.5. SMARTY | 6 |

1. HTML



When we read news from Internet, we'll use a Web browser, like Internet Explorer or FireFox as a client, to request some news from a Web server. Those data are static as the server does nothing but just simply get the data and send to us in HTML format and displayed on the browser.

HTML stands for HyperText Markup Language. We use an editor in creating a HTML file. But what we see on the editor is NOT what you see on the browser. In a HTML file, we have to use tags (as markups) for browser to display, for example the tag <p> for a new paragraph.

Example 1

```
html >
<body>

Hello!

This     i s     m y   f i r s t
       H T M L   f i l e.

</body>
</html >
```

Hello! This is my first HTML file.

Example 2

```
html >
<body>

Hello!

<p>
This is my first
HTML file.

</body>

</html >
```

Hello!
This is my first HTML file.

2. Cascading Style Sheets (CSS)

HTML tags were originally designed to define the content of a document. They were supposed to say "This is a header", "This is a paragraph", "This is a table", by using tags like <h1>, <p>, <table>, and so on.

CSS adds new HTML tags and attributes (like the tag and the color attribute) to the original HTML specification. There are 3 ways of applying CSS to a Web Page

2.1. Inline Styles

This applies within the anchor only.

Example 3

```
<H1 STYLE="color: red">This Heading will be Red</H1>
<P STYLE="font-size: 12pt; font-family: Verdana, sans-serif">This is the content of the
paragraph. </P>
```

This Heading will be Red
This is the content of the paragraph.

2.2. Embedded Styles

This applies to the whole page. It's right on top of the file, so it's easy to modify and to maintain.

Example 4

```
<HTML>
<HEAD>
<STYLE TYPE="text/css">
<!--
H1 { color: red; }
P { font-size: 12pt;
Font-family: Verdana, sans-serif;
}
-->
</STYLE>
<TITLE> This Heading will be Red </TITLE>
</HEAD>
This is the content of the paragraph.
</HTML>
```

2.3. External Style Sheets Styles

This applies to the many pages with this external style sheet. All styles are in a single file, so it's easy to modify and to maintain the whole web page with many files.

Example 5

```
<HTML>
<HEAD>

<LINK REL=STYLESHEET" HREF="pathname/styl esheet.css" TYPE="text/css">

<TITLE> Thi s Heading wi ll be Red </TITLE>
</HEAD>
This is the content of the paragraph.
</HTML>
```

3. JavaScript

JavaScript is a simple scripting language invented specifically for use in web browsers to make websites more dynamic. It's also known as Client-Side Scripting (CSJS) as it is run by client web browser. This is different from Server-Side Scripting, like CGI, PHP, ..., which is run on a web server to generate dynamic [HTML](#) pages.

Example 6

```
<html >
<head>

<scri pt type="text/j avascri pt">

functi on mouseOver()
{
document. b1. src ="ptr_l eft. gi f";
}

functi on mouseOut()
{
document. b1. src ="ptr_ri ght. gi f";
}
</scri pt>
</head>

<body>
<a href="page1. htm">

<i mg border="0" src="ptr_ri ght. gi f" name="b1"
onMouseOver="mouseOver()"
onMouseOut="mouseOut()" />
</a>

</body>
</html >
```

onMouseOver



onMouseOut



4. Common Gateway Interface (CGI)

When we have to provide some info for the server to process before sending the results back to us, like converting for some amount of US dollars to Euros based on the foreign rate at the instant of request, the server has to use a programming language called server-side script, well-known as Server-Side Includes (SSI), to process the request., and send back to client browser in HTML format.

For that very reason, any programming language can be used as SSI to generate HTML file.

Remark 1

All CGI must be run on Web Server and put in a special executable directory CGI-BIN. A web browser is used to get that file and display on user screen.

Let's say we want to send to the browser an HTML file below

Example 7 : Hello.htm

```
<html >
<body>

Hel I o Worl d!

</body>
</html >
```

we can use OS shell (DOS, Linux, ...), C language, Perl, PHP, ...

4.1. OS Shell

Example 8 : Hello.cgi

```
echo "<html >"
echo "<body>"
echo ""
echo "Hel I o Worl d! "
echo ""
echo "</body>"
echo ""
echo "</html >"
```

Remark 2

Extension CGI is optional, just for tracking purpose.

4.2. C Language

Example 9 : Hello.c

```
include <stdio.h>
int main(void) {
    printf("Content-Type: text/plai n; charset=us-asci i \n\n");
    printf("Hel I o worl d\n\n");
    return 0;
}
```

Using C compiler to create execute file, say hello.cgi, for this code

4.3. Perl Language

Example 10 : Hello.cgi

```
#!/usr/bin/perl

print "Content-type: text/html\n";
print "\n";

print "<html >\n";
print "<body>\n";
print "\n";
print "Hello World!\n";
print "\n";
print "</body>\n";
print "\n";
print "</html >\n";
```

or better version using here-document syntax

Example 11

```
#!/usr/bin/perl -wT

use strict;

print "Content-type: text/html\n\n";

print <<HTML_EOF;
<html >
<head>

</head>

<body>

Hello World!

</body>
</html >
HTML_EOF
```

Or even better version using perl module CGI

Example 12

```
#!/usr/bin/perl -wT

use CGI qw(:standard);

print header;

print "Hello, world!\n";

print end_html;
```

4.4. PHP Language

PHP stands for Hyper Text Pre-Processor, originally known as Personal Home Page.

PHP code is embedded within `<?php . . . ?>` in HTML, while other SSI languages are used to generate HTML.

Example 13: hello.php

```
<html >
<head>

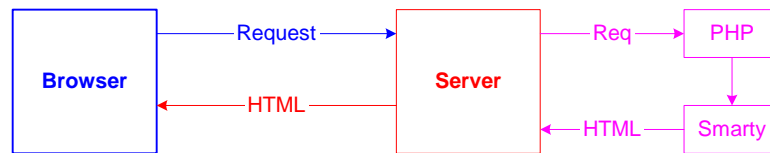
</head>

<body>
<?php
echo "Hel I o Worl d! ";
?>
</body>
</html >
```

4.5. Smarty

PHP is used to handle the logic while HTML for presentation. PHP codes are mixed with HTML code in a file, so the advantage is continuity of the logic, but the disadvantage is PHP and HTML so convoluted, hence difficult to maintain.

Smarty is a solution to this problem in to separating PHP and HTM.



Note that we cannot use a browser to open a HTML file embedded with PHP as PHP is server-side script (SSS), but we can with HTML embedded with Client-Side JavaScript. (CSJS)

When Smarty splits a HTM file into PHP and a HTML-like called Template (TPL), browser must request PHP file, not TPL even is HTML alike. The PHP file must have the statement `$smarty->display('template.tpl')`. That's why we use extension TPL instead of HTM. There is always a pair in Smarty implementation.

Therefore, for tool configuration, we have to

- Specify a path of PHP library in Server config file via "LoadModule"
- Specify a path of Smarty lib in PHP.ini file via "include_path"

A PHP code must be anchored within `<?php "PHP code" ?>` or span over multiple lines

In Smarty approach, every PHP file must

- specify 4 directories, no need if they are in the same doc_root
- template_dir to hold all TPL files
- config_dir to hold config file, if so required
- cache_dir : just for Smarty to hold temporary file during processing
- compile_dir : just for Smarty to hold temporary file during processing
- export results to TPL using “assign”
- evoke the corresponding TPL using “display”

Every PHP file requires has its own TPL file in form of HTM syntax, but embedded with Smarty codes within curly braces { ... }, including results exported from PHP using \$.

Example 14 : smrty-00.php

```
<?php
require 'Smarty.class.php';
$smarty = new Smarty;

$smarty->template_dir = 'C:\WWW\Apache2.2\htdocs\templates';
$smarty->config_dir = 'C:\WWW\Apache2.2\htdocs\configs';

$smarty->cache_dir = 'C:\WWW\Smarty\cache';
$smarty->compile_dir = 'C:\WWW\Apache2.2\htdocs\templates_c';

$smarty->assign('name', 'Ned');

$smarty->display('smrty-00.tpl');
?>
```

Example 15 : smrty-00.tpl

```
{* Smarty *}
Hello {$name}, welcome to Smarty!
```

Or even better version using setup file

Example 16: smarty-setup.php

```
<?php
// Load Smarty library
require 'Smarty.class.php';

// The smarty-setup.php file is a good place to load required application library files
class New_Smarty extends Smarty {
    function New_Smarty()
    {
        // Class Constructor.
        // These automatically get set with each new instance.
        $this->Smarty();
        $this->template_dir = 'C:\WWW\Apache2.2\htdocs\templates';
        $this->compile_dir = 'C:\WWW\Apache2.2\htdocs\templates_c';
        $this->config_dir = 'C:\WWW\Apache2.2\htdocs\configs';
        $this->cache_dir = 'C:\WWW\Smarty\cache';
        $this->caching = true;
        $this->assign('app_name', 'New Smarty');
    }
}
?>
```

Example 17 : smrty-01.php

```
<?php
require 'Smarty_Setup.php';
$smarty = new New_Smarty;
$smarty->assign('name', 'Ned');
$smarty->display('smrty-01.tpl');
?>
```

Example 18 : smrty-01.tpl

```
{* Smarty *}
Hello {$name}, welcome to Smarty!
```

Output

Hello **Ned**, welcome to Smarty!

Remark 3

- PHP display **template.TPL** using **\$smarty_display('template.TPL')**
- TPL display **\$var** exported by PHP using **\$smarty_assign('var', 'value')**