

HỌC VIỆN KỸ THUẬT MẬT MÃ
KHOA ATTT

NGÔN NGỮ LẬP TRÌNH PHÍA SERVER (PHP)

NỘI DUNG

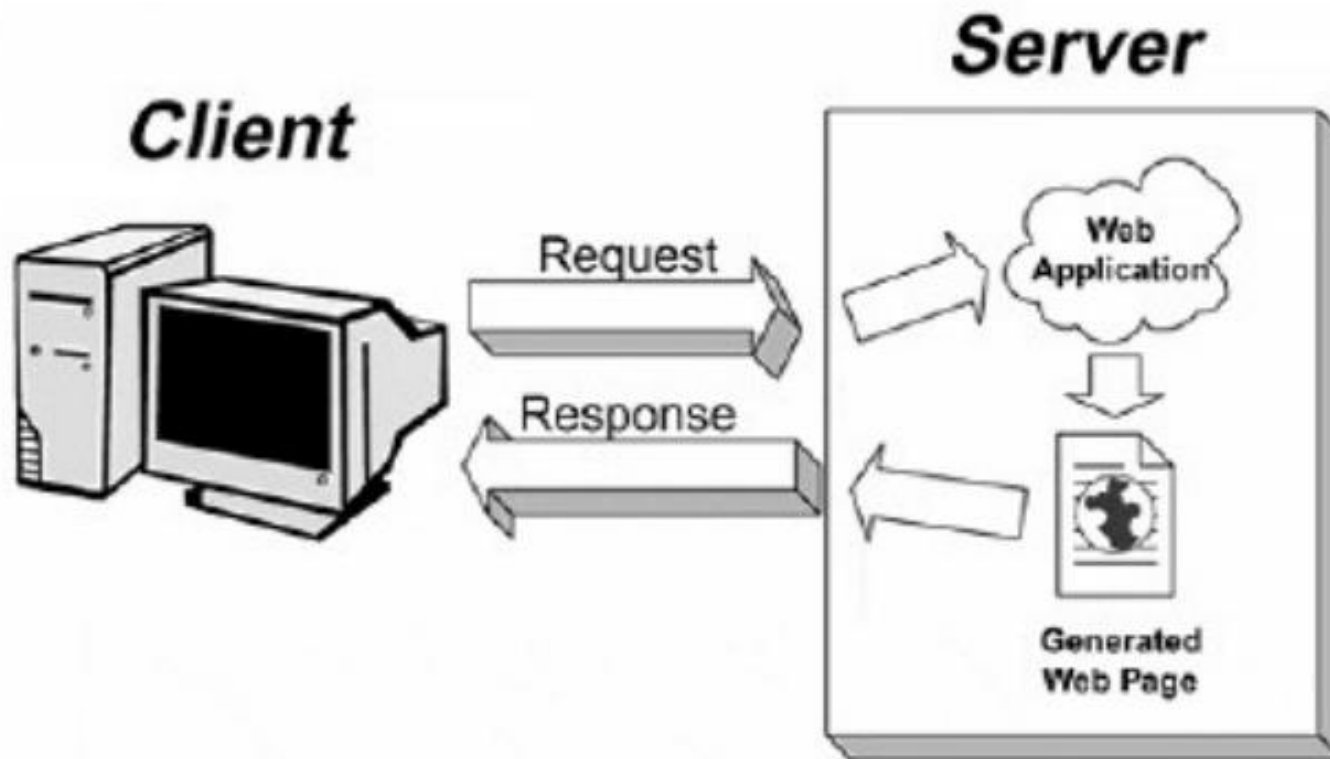
1. Kiến trúc của một ứng dụng web động
2. Ngôn ngữ lập trình web phía server
3. PHP
4. Mô hình MVC



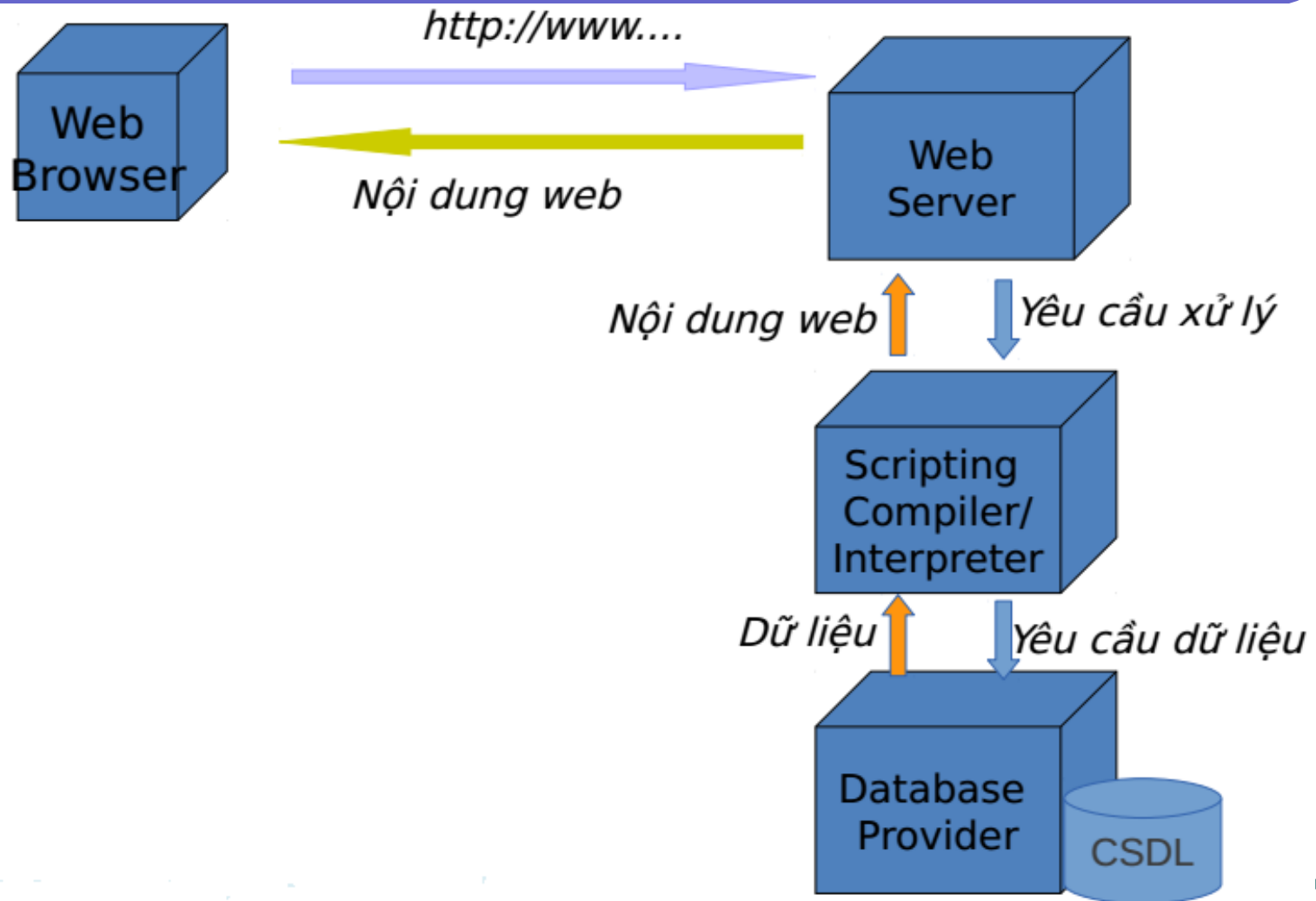
Web động

- Nội dung trang web (HTML + CSS + JavaScript) được Web Server sinh ra khi có yêu cầu từ Client.
- Rất phổ dụng: Hầu hết các trang web thương mại đều là web động.
- Sử dụng ngôn ngữ lập trình đa năng để sinh ra nội dung web.
- Sử dụng CSDL.

Kiến trúc web động

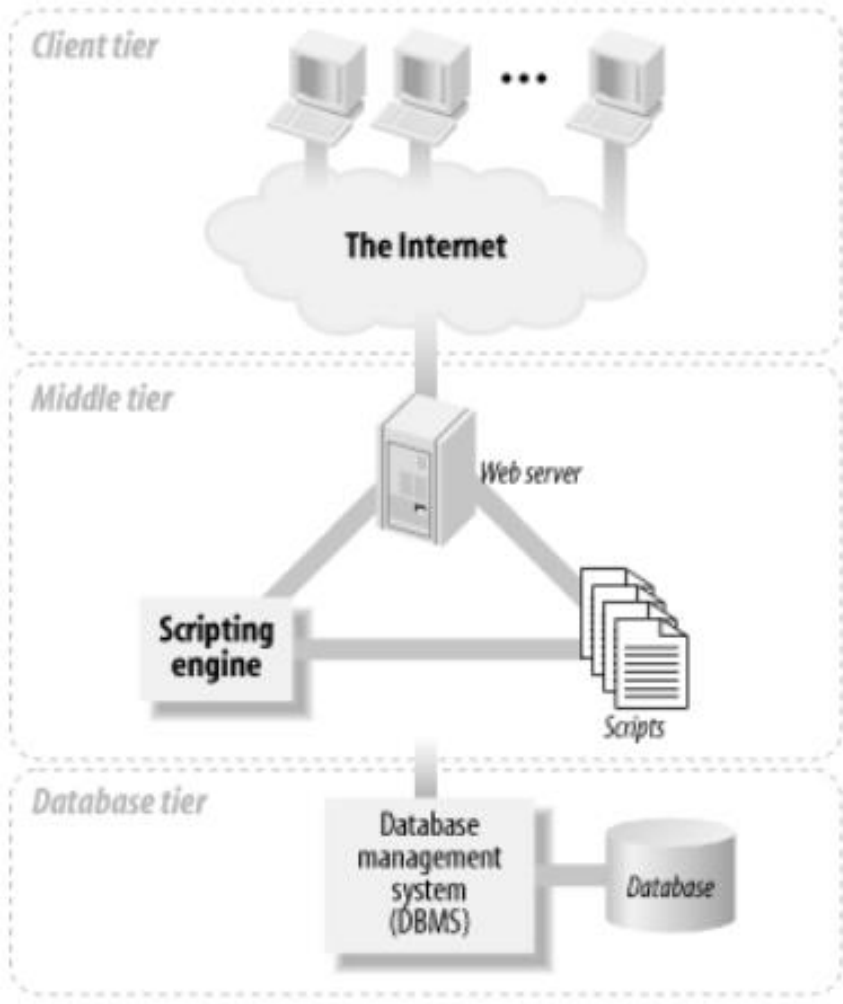


Web động với CSDL



Mô hình ba tầng

- **Tầng khách:** trình diễn và tương tác với người dùng
- **Tầng giữa:** thực hiện các logic của ứng dụng
- **Tầng CSDL:** bao gồm hệ quản trị CSDL, CSDL của ứng dụng



NỘI DUNG

1. Kiến trúc của một ứng dụng web động
2. Ngôn ngữ lập trình web phía server
3. PHP
4. Mô hình MVC

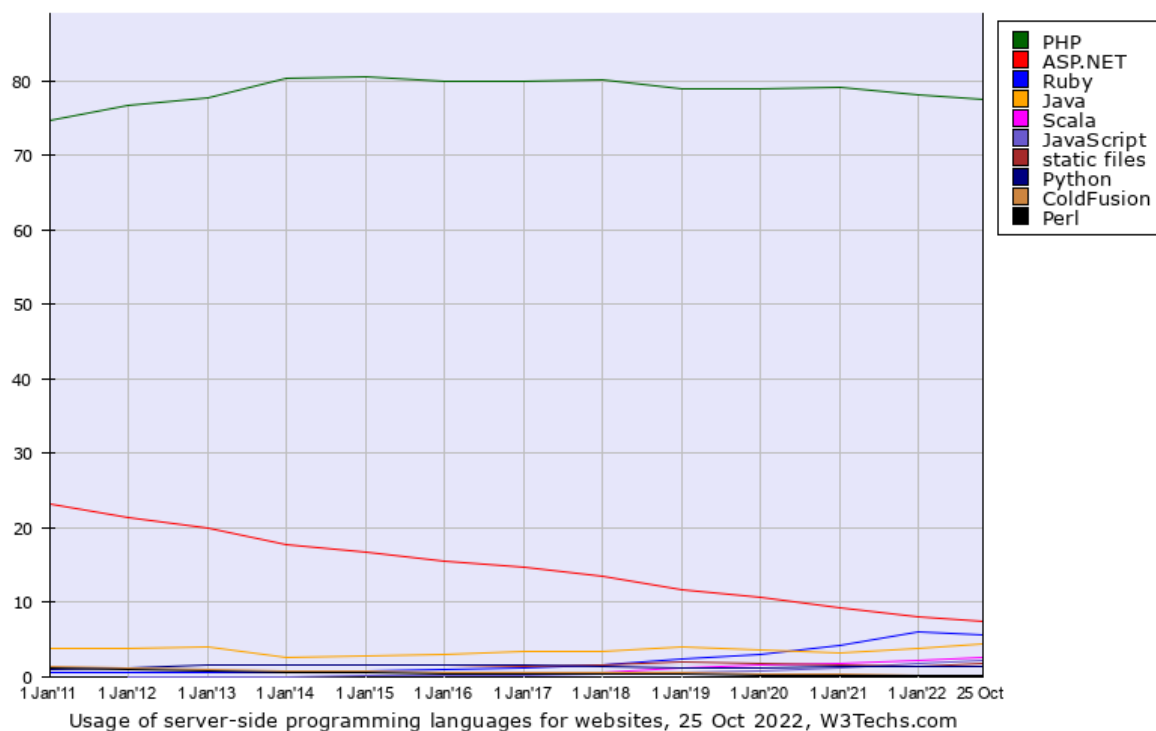


Historical yearly trends in the usage statistics of server-side programming languages for websites

This report shows the historical trends in the usage of the top server-side programming languages since January 2011.

	2011 1 Jan	2012 1 Jan	2013 1 Jan	2014 1 Jan	2015 1 Jan	2016 1 Jan	2017 1 Jan	2018 1 Jan	2019 1 Jan	2020 1 Jan	2021 1 Jan	2022 1 Jan	2022 25 Oct
PHP	74.8%	76.6%	77.7%	80.3%	80.6%	80.0%	80.0%	80.2%	78.9%	78.9%	79.1%	78.1%	77.4%
ASP.NET	23.2%	21.4%	19.9%	17.8%	16.7%	15.6%	14.8%	13.5%	11.8%	10.6%	9.3%	8.0%	7.6%
Ruby	0.5%	0.6%	0.5%	0.6%	0.9%	1.1%	1.3%	1.6%	2.4%	3.0%	4.3%	6.0%	5.6%
Java	3.8%	3.9%	4.0%	2.6%	2.8%	3.1%	3.3%	3.4%	4.0%	3.7%	3.2%	3.7%	4.5%
Scala					0.2%	0.2%	0.3%	0.5%	1.2%	1.6%	1.8%	2.3%	2.7%
JavaScript	<0.1%	<0.1%	0.1%	0.1%	0.1%	0.2%	0.3%	0.4%	0.7%	0.8%	1.2%	1.8%	2.2%
static files						1.5%	1.5%	1.6%	2.1%	1.8%	1.6%	1.5%	1.8%
Python	1.0%	1.3%	1.5%	1.7%	1.6%	1.7%	1.6%	1.3%	1.1%	1.3%	1.4%	1.4%	1.4%
ColdFusion	1.3%	1.2%	1.1%	0.8%	0.7%	0.7%	0.6%	0.6%	0.5%	0.5%	0.3%	0.3%	0.3%
Perl	1.1%	1.0%	0.8%	0.6%	0.5%	0.5%	0.4%	0.3%	0.3%	0.2%	0.2%	0.1%	0.1%
Erlang					0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%
Miva Script						0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%	<0.1%

The diagram shows only server-side programming languages with more than 1% usage.



Find more details in our extensive server-side programming languages market reports.

[Learn more](#)

NỘI DUNG

1. Kiến trúc của một ứng dụng web động
2. Ngôn ngữ lập trình web phía server
3. **PHP – Hypertext Preprocessor**
4. Mô hình MVC



PHP là gì?

- PHP viết tắt của từ Hypertext Preprocessor.
- Là một ngôn ngữ lập trình kịch bản (script) mã nguồn mở
- Chủ yếu được dùng để phát triển các ứng dụng viết cho máy chủ, cho phát triển web và có thể được nhúng vào HTML
- PHP chạy trên nhiều nền tảng khác nhau: Windows, Linux, Unix, Mac OS X, Android, ...
- PHP tương thích với hầu như tất cả các máy chủ sử dụng hiện nay: Apache, IIS, ...
- PHP Hỗ trợ rất nhiều cơ sở dữ liệu: MySQL, MS SQL server, Redis, MongoDB, Oracle, ...

PHP – Đặc điểm

- Tựa C/C++, trừ các điểm sau:
 - Định kiểu không rõ ràng
 - Tên biến bắt đầu bằng \$
 - Mảng là ánh xạ
 - Định nghĩa hàm bằng từ khóa **function**

Trang PHP

- Các trang có tên mở rộng **.php**
- Mã PHP được để trong cặp thẻ **<?php** và **?>** - được gọi là các **phân đoạn PHP**. Có thể nhúng các phân đoạn PHP vào bất kỳ vị trí nào trong trang. Bên ngoài các phân đoạn PHP có thể chứa mã HTML, CSS, javascript.
- Phần mã PHP được thực thi để sinh ra phần **động** của trang web.
- Sử dụng hàm **echo** để đưa nội dung (HTML, CSS, javascript) vào thân gói HTTP Response.
- Sử dụng hàm **header** để thay đổi giá trị các trường tiêu đề gói HTTP Response

Trang php

Mã HTML, CSS, javascript

```
<?php
```

```
//Mã php được thực thi bên server để  
sinh ra phần động của trang web
```

```
?>
```

Mã HTML, CSS, javascript

```
<?php
```

```
/*Mã php được thực thi bên server để  
sinh ra phần động của trang web */
```

```
?>
```

Mã HTML, CSS, javascript

Ví dụ trang php

```
1 <!DOCTYPE html>
2 <html>
3     <body>
4         <h1>Tài liệu học Lập Trình Web</h1>
5         <?php
6             echo "<hr>";
7         ?>
8         <p>Tài liệu học HTML</p>
9         <p>Tài liệu học CSS</p>
10        <?php
11            echo "<h2>Tài liệu học JavaScript</h2>";
12            echo "<h3>Tài liệu học MySQL</h3>";
13            echo "<h4>Tài liệu học PHP</h4>";
14        ?>
15        <hr>
16        <?php
17            $text = "Từ cơ bản" . " " . "đến nâng cao";
18            echo $text;
19        ?>
20    </body>
21 </html>
```

Tài liệu học Lập Trình Web

Tài liệu học HTML

Tài liệu học CSS

Tài liệu học JavaScript

Tài liệu học MySQL

Tài liệu học PHP

Từ cơ bản đến nâng cao

Khi nào thì cần có mã HTML, CSS, javascript trong trang php?

- Những trang chỉ bao gồm mã xử lý nghiệp vụ thì không cần mã HTML, CSS, javascript.
- Những trang tạo giao diện
 - có thể chứa mã HTML, CSS, javascript
 - hoặc dùng hàm echo của php để sinh ra mã HTML, CSS, javascript.

Cách ghi chú thích

- Cú pháp 1: Đặt cặp dấu gạch chéo // phía trước nội dung chú thích. Chỉ áp dụng cho trường hợp nội dung chú thích nằm gọn trên một dòng.
- Cú pháp 2: Dùng cặp dấu /* */ để bao bọc nội dung chú thích.

Biến

- Cách khai báo biến:
 - \$tên biến = giá trị mà bạn muốn gán cho biến;

```
1 <?php
2     $name = "Nguyễn Thành Nhân";
3     $year = 1993;
4     $city = 'Cần Thơ';
5     echo "<p>Giá trị của biến name là chuỗi $name";
6     echo "<p>Giá trị của biến year là số $year";
7     echo "<p>Giá trị của biến city là chuỗi $city";
8 ?>
```

- Các loại biến khác nhau như sau:
 - global (toàn cầu, toàn cục)
 - local (cục bộ)
 - static (tĩnh)

Phạm vi của biến (tiếp)

```
<?php
    $a = 1;
    include 'lib.inc';
    $b = $a;
?>
```

- \$a có phạm vi trong cả tệp lib.inc
- \$b được định nghĩa trong lib.inc cũng có phạm vi trong cả tệp đang định nghĩa

Phạm vi của biến (tiếp)

```
<?php
$a = 1; /* global scope */
function test() {
    echo $a; /* reference to local scope
             variable */
}
test();
?>
```

- Hàm test() không in ra gì cả vì \$a trong hàm test là biến cục bộ, không phải là \$a toàn cục.

Phạm vi của biến

```
<?php
$a = 1; /* global scope */
function test() {
    global $a;
    echo $a; /* reference to global scope variable */
    echo " ";
    echo $GLOBALS['a']; /* reference to global
    scope variable */
}
test();
?>
```

- Hàm test() in ra 1 vì nó dùng biến \$a toàn cục.

Biến tĩnh

- Biến tĩnh chỉ có phạm vi truy cập cục bộ trong hàm, nhưng giá trị của nó không bị mất khi thực thi của chương trình thoát khỏi hàm

```
function test() {  
    static $count = 0;  
    $count++;  
    echo $count;  
    if ($count < 10) {  
        test();  
    }  
    $count--;  
}
```

Kiểm tra kiểu của biến

- `is_int($v)` - \$v là số nguyên?
- `is_float($v)` - \$v là số thực?
- `is_bool($v)` - \$v là biến bool?
- `is_string($v)` - \$v là nội xâu?
- `is_array($v)` - \$v là một mảng?
- `is_object($v)` - \$v là một đối tượng?
- `is_numeric($v)` - \$v là một số?
- `is_null($v)` - \$v = NULL?

Kiểm tra sử dụng biến

- `isset($v)` - `$v` đã được thiết lập hay chưa
- `empty($v)` - `$v` có giá trị null
- `unset($v)` – hủy `$v`

Chuyển kiểu

- `strval($v)` - chuyển giá trị của \$v thành một chuỗi
- `intval($v)` - chuyển giá trị của \$v thành một số nguyên
- `floatval($v)` - chuyển giá trị của \$v thành một số thực

Hằng

- Cú pháp:
 - `define(name, value, case-insensitive)`

Tham số	Mô tả
name !	- Tên hằng.
value !	- Giá trị của hằng.
case-insensitive ✓	<ul style="list-style-type: none">- Xác định việc có phân biệt trường hợp chữ IN HOA hay chữ thường đối với tên hằng hay không.- Tham số này có thể nhận một trong hai giá trị:<ul style="list-style-type: none">◦ TRUE - Không phân biệt.◦ FALSE - Có phân biệt (đây là giá trị mặc định)

Kiểu dữ liệu

- String (chuỗi)
- Integer (số nguyên)
- Float (số thực, số dấu phẩy động)
- Boolean (đúng/sai)
- Array (mảng)
- Object (đối tượng)
- NULL (giá trị NULL)
- Resource (tài nguyên)

Mảng (...)

- Mảng trong PHP là ánh xạ có thứ tự chứa các phần tử <key, value>
 - key chỉ nhận giá trị kiểu nguyên, chuỗi
 - value nhận giá trị kiểu bất kỳ
 - Các phần tử có thể sử dụng key tăng tự động

Ví dụ:

```
$thu = array(  
    "CN" => "Chủ nhật",  
    2 => "Thứ hai",  
    "Ba", "Tư", "Năm", "Sáu", "Bảy"  
);
```

Mảng (tiếp)

- Truy cập các phần tử mảng
 - array[key]
- Ví dụ

```
echo $thu["CN"];  
$tmp = $thu[2];
```

Mảng (tiếp)

- Thêm/sửa đổi giá trị phần tử mảng
 - `$arr[key] = value; //sửa đổi nếu đã tồn tại phần tử có khóa là key, thêm phần tử mới nếu ngược lại`
 - `$arr[] = value; //thêm phần tử mới với khóa tăng tự động`
- Ví dụ:
`$thu["CN"] = "Sunday"; //sửa đổi`
`$thu[2] = "Monday"; //sửa đổi`
`$thu[1] = "Sunday"; //thêm mới`
`$thu["Sun"] = "Sunday"; //thêm mới`

Mảng (tiếp)

- Xóa phần tử mảng
 - `unset(array[key]);`
- Ví dụ
`unset($thu["Sun"]);`

Mảng (tiếp)

- Duyệt các phần tử mảng
 - foreach(array as key=>value)
 - foreach(array as value)
- Ví dụ

```
$colors = array('red', 'blue', 'green', 'yellow');
foreach ($colors as $color) {
    echo "Do you like $color?\n";
}
foreach ($colors as $key => $color) {
    $colors[$key] = strtoupper($color);
}
```

Mảng "nhiều chiều"

```
$poems = array(  
    array("name" => "Nguyễn A",  
        "titles" => array("Gió thu", "Sóng sánh", "Chiều hồng")),  
    array("name" => "Trần B",  
        "titles" => array("Ra trận", "Hồng quân")),  
    array("name" => "Trịnh C", "titles" => array("Sông quê"))  
);  
  
foreach ($poems as $p) {  
    echo $p["name"]." có ". count($p["titles"]). " tác phẩm là:";  
    foreach ($p["titles"] as $t) echo " ".$t;  
    echo ". ";  
}
```

//Nguyễn A có 3 tác phẩm là: Gió thu Sóng sánh Chiều hồng. //Trần B có 2 tác phẩm là: Ra trận Hồng quân. Trịnh C có 1 tác phẩm là: Sông quê.

Mảng

- Đếm số phần tử mảng
 - `count(array);`
- Sắp xếp các phần tử mảng
 - `sort(array);`
- Gán mảng
 - `array1 = array2;`
- ...
- Tham khảo: <http://php.net/ref.array>

Xâu

- Xâu được đánh dấu bởi dấu nháy đơn hoặc nháy kép
 - Ví dụ, “Đây là một xâu ký tự”, ‘Đây là một xâu khác’
- Xâu có thể chứa biến bên trong

`$number = 45;`

`$vehicle = "bus";`

`$message = "This $vehicle holds $number people";`

- Nối xâu bằng dấu chấm (.)

`$message = "This ". $vehicle "." holds ". $number ."
people";`

- Các ký tự đặc biệt cần được để sau \,

- Ví dụ: \\, \', \", \\$, ...

Xâu (tiếp)

- `strlen($s)` – độ dài xâu `$s`
- `strtolower($s)` - xâu viết thường của `$s`
- `strtoupper($s)` - xâu viết hoa của `$s`
- `ucfirst($s)` – xâu viết hoa ký tự đầu của `$s`
- `ucwords($s)` – xâu viết hoa ký tự đầu các từ `$s`
- `ltrim($s)` – xâu được bỏ dấu trắng đầu xâu của `$s`
- `rtrim($s)` – xâu được bỏ dấu trắng cuối xâu của `$s`
- `trim($s)` – xâu được bỏ dấu trắng đầu và cuối xâu của `$s`
- Tạo dữ liệu định dạng:
 - `string sprintf (string format [, mixed args...])`
 - `printf (string format [, mixed args...])`

So sánh xâu

- `strcmp($str1, $str2)`
- `strncmp($str1, $str2, $length)`
 - 0 – nếu hai xâu bằng nhau
 - -1 – nếu $\$str1 < \$str2$
 - 1 – nếu $\$str1 > \$str2$

Tìm và thay thế chuỗi con

- `substr($s, $start [, $length])` – lấy chuỗi con của \$s, bao gồm các ký tự bắt đầu từ chỉ mục \$start và có \$length ký tự (hoặc đến hết nếu vắng \$length)
- `strpos($s, $f [, $offset])` – trả về chỉ mục của xuất hiện đầu tiên của \$f trong \$s, [bắt đầu tìm từ \$offset]
- `strstr($s, $f)` - tìm \$f trong \$s và trả về chuỗi con bắt đầu từ điểm xuất hiện đầu tiên của \$f đến hết \$s
- `stristr(string haystack, string needle)` – tương tự `strstr()` nhưng không phân biệt hoa thường
- `explode($sep, $s [, $limit])` – trả về mảng là kết quả của tách \$s bằng chuỗi phân cách \$sep
- `implode($glue, $array)` – trả về chuỗi là kết quả nối các phần tử mảng \$array, sử dụng \$glue để nối

Tìm và thay thế chuỗi con

- `substr_replace($s, $r, $start [, $length])`:
Trả về chuỗi là \$s được thay [\$length] ký tự bắt đầu từ chỉ mục \$start bằng [\$length] ký tự đầu của \$r
- Ví dụ
 - `substr_replace("chào cháu", "chú", 5);`
cho kết quả *"chào chú"*
 - `substr_replace("", "chiều", 10, 1);` cho kết quả *"chào buổi sáng"*

Thay thế chuỗi con

- **strtr(\$s, \$from, \$to)** – Trả về chuỗi là kết quả của thay thế các ký tự của \$s xuất hiện trong \$from bằng ký tự cùng chỉ mục trong \$to, Ví dụ
*\$mischief = strtr("command.com", "aeiou", "äëïöü");
print \$mischief; // prints cömmänd.cöm*
- **strtr(\$s, \$map)** – Trả về chuỗi là kết quả của thay thế các chuỗi con của \$s xuất hiện trong mảng \$map. Ví dụ:
*\$glossary = array("BTW"=>"by the way",
"IMHO"=>"in my humble
opinion", "IOW"=>"in other words", "OTOH"=>"on
the other hand");
print strtr(\$geekMail, \$glossary);*

Gỡ lỗi với kiểu và giá trị

- In ra kiểu, giá trị và biểu diễn của biểu thức
 - `print_r(bieu_thuc)`
 - `var_dump(bieu_thuc1, bieu_thuc2, ...)`

Cách khai báo & gọi một hàm

- Khai báo:
 - *function TênHàm(không có tham số / tham số 1, tham số 2, tham số 3){
 //Some code
}*
- Lệnh return dùng để trả về cho hàm một giá trị (sau khi hàm thực thi xong, hàm sẽ có một giá trị, lúc đó nó có thể được sử dụng giống như một biến)

Các hàm số thực và nguyên

- Trị tuyệt đối
 - `integer abs(integer number)`
 - `float abs(float number)`
- Trần và sàn
 - `float ceil(float value)` - Giá trị nguyên thấp nhất cao hơn
 - `float floor(float value)` – Giá trị nguyên cao nhất thấp hơn
- Làm tròn
 - `float round(float value [, integer precision])` - $\geq .5$ chuyển thành trần, $< .5$ chuyển thành sàn

Các hàm số thực và nguyên

- Chuyển đổi thập phân – hệ số khác (xâu)
 - `string decbin(integer number)` - Thập phân -> Nhị phân
 - `integer bindec(string binarystring)` – Nhị phân -> thập phân
 - `string dechex(integer number)` - Thập phân -> hexa (hệ 16)
 - `integer hexdec(string hexstring)` - Hexa -> thập phân
 - `string decoct(integer number)` - Thập phân -> Octan (hệ 8)
 - `integer octdec(string octalstring)` – Octan -> thập phân

Các hàm lượng giác

- `float sin(float arg)` – sin
- `float cos(float arg)` – cos
- `float tan(float arg)` – tang
- `float asin(float arg)` – asin
- `float acos(float arg)` – acos
- `float atan(float arg)` – atang
- `float atan2(float y, float x)` - atang2
- `float pi()` - 3.1415926535898
- `float deg2rad(float arg)` – (Đo góc) độ -> radian
- `float rad2deg(float arg)` – (Đo góc) radian -> độ

Các hàm mũ và logarit

- `float exp(float arg)` – e^{arg}
- `float pow(float base, number exp)` - base^{exp}
- `float sqrt(float arg)` – Căn bậc hai của arg
- `float log(float arg)` – Logarit cơ số 2 của arg
- `float log10(float arg)` – Logarit cơ số 10 của arg

Các hàm sinh số ngẫu nhiên

- `void srand(integer seed)` - Được gọi trước khi gọi các hàm tiếp sau
- `integer rand()` – Tạo một số tự nhiên ngẫu nhiên
- `integer rand(integer min, integer max)` - Tạo một số tự nhiên ngẫu nhiên trong đoạn `[min, max]`

Lệnh điều kiện if ... elseif ... else trong PHP

```
if(điều kiện 1){
```

```
    //đoạn mã này sẽ được thực thi nếu điều kiện 1 là đúng
```

```
}elseif(điều kiện 2){
```

```
    //đoạn mã này sẽ được thực thi nếu điều kiện 1 sai và điều kiện 2 là đúng
```

```
}else{
```

```
    //đoạn mã này sẽ được thực thi nếu tất cả những điều kiện trên là sai
```

```
}
```

Cách sử dụng lệnh "switch case" trong PHP

```
switch (giá trị){  
    case trường hợp 1:  
        //đoạn mã này sẽ được thực thi khi giá trị trùng khớp với trường hợp 1  
        break;  
    case trường hợp 2:  
        //đoạn mã này sẽ được thực thi khi giá trị trùng khớp với trường hợp 2  
        break;  
    case trường hợp 3:  
        //đoạn mã này sẽ được thực thi khi giá trị trùng khớp với trường hợp 3  
        break;  
    ...  
    ...  
    ...  
    default:  
        //đoạn mã này sẽ được thực thi khi giá trị KHÔNG trùng khớp trường hợp nào cả  
        break;  
}
```


Vòng lặp while & do while

- Vòng lặp while

```
while(điều kiện){  
    //Đoạn mã mà bạn muốn thực thi  
}
```

- Vòng lặp do while

```
do{  
    //Đoạn mã mà bạn muốn thực thi  
}while(điều kiện);
```

Vòng lặp for

- Vòng lặp for

```
for(biểu thức 1; biểu thức 2; biểu thức 3){  
    //Đoạn mã mà bạn muốn được thực thi  
}
```

· Trong đó:

- Biểu thức 1 thường là một câu lệnh khai báo biến
(biến này dùng để tham gia vào biểu thức 2)
- Biểu thức 2 là một biểu thức điều kiện
(nếu điều kiện đúng thì đoạn mã sẽ được thực thi, còn nếu điều kiện sai thì vòng lặp kết thúc)
- Biểu thức 3 thường là một biểu thức làm thay đổi giá trị của biến được khai báo trong biểu thức 1

Vòng lặp foreach

- Cú pháp:

```
foreach ($array as $value) {  
    //Đoạn mã mà bạn muốn được thực thi  
}
```

```
1 <?php  
2  
3     $data = array("HTML", "CSS", "JavaScript", "MySQL", "PHP");  
4  
5     foreach ($data as $value) {  
6         echo "<p>Lập Trình Web</p>";  
7     }  
8  
9 ?>
```

Lớp và đối tượng

- Lớp (class) là một khuôn mẫu cho các đối tượng, một đối tượng (object) là một thể hiện của một lớp.
- Khi các đối tượng được tạo thì chúng sẽ kế thừa tất cả thuộc tính & phương thức từ lớp, nhưng mỗi đối tượng sẽ có các giá trị khác nhau cho các thuộc tính.

Class	Object
Mobile	Nokia
	Samsung
	LG

Lớp và đối tượng

```
class SimpleClass {  
    //định nghĩa hằng  
    const constant = 'constant value';  
    // định nghĩa biến/thuộc tính  
    public $var = 'a default value';  
    // định nghĩa hàm/phương thức  
    public function displayVar() {  
        echo $this->var;  
    }  
}
```

class tên_lớp{ //các thuộc tính & phương thức được khai báo tại đây }

Để tạo một đối tượng từ một lớp thì ta sử dụng từ khóa **new**

```
$obj = new SimpleClass();  
$obj->displayVar();  
echo $obj->var;  
echo SimpleClass::constant;
```

- *\$this* được dùng để chỉ đối tượng gọi, chỉ có thể được sử dụng bên trong các phương thức
- Ba từ khóa dùng để xác định phạm vi truy cập vào các thuộc tính & phương thức của lớp:
 - public (mặc định) - có thể được truy cập ở bất cứ đâu.
 - protected - chỉ có thể được truy cập bên trong lớp, hoặc bên trong các lớp được kế thừa từ lớp này.
 - private - chỉ có thể được truy cập bên trong lớp.

Thay đổi giá trị thuộc tính

- Cách 1: Bên trong lớp (sử dụng \$this)

```
<?php
class Fruit {
    public $name;
    function set_name($name) {
        $this->name = $name;
    }
}
$banana = new Fruit();
$banana->set_name("Banana");
?>
```

- Cách 2: Bên ngoài lớp (thay đổi trực tiếp giá trị của thuộc tính)

```
<?php
class Fruit {
    public $name;
}
$banana = new Fruit();
$banana->name = "Banana";
?>
```

Thuộc tính/phương thức tĩnh

```
class A {  
    public static $foo = 'I am foo';  
    public $bar = 'I am bar';  
    public static function getFoo() { echo self::$foo; }  
    public static function setFoo() { self::$foo = 'I am a new foo'; }  
    public function getBar() { echo $this->bar; }  
}  
$ob = new A();  
A::getFoo(); // output: I am foo  
$ob->getFoo(); // output: I am foo  
A::getBar(); // output: fatal error: using $this not in object  
context  
$ob->getBar(); // output: I am bar
```

Kế thừa

- “kế thừa” dùng để chỉ việc một lớp được dẫn xuất từ một lớp khác.
- Lớp được dẫn xuất từ lớp khác thì được gọi là “lớp con”, nó sẽ thừa hưởng tất cả các thuộc tính và phương thức (thuộc loại public & protected) của lớp cha, ngoài ra thì lớp con có thể sở hữu các thuộc tính & phương thức của riêng nó.
- Khai báo:

```
class TênLớpCon extends TênLớpCha{  
    //some code  
}
```


Kế thừa

- Các phương thức được kế thừa từ lớp cha có thể được ghi đè bằng cách khai báo lại bên trong lớp con.

```
<?php
class CongDan{
    public $name;
    public $year;
    public function __construct($input_name, $input_year){
        $this->name = $input_name;
        $this->year = $input_year;
    }
    protected function intro(){
        echo "Tôi tên là {$this->name}, sinh năm {$this->year}";
    }
}

class SinhVien extends CongDan{
    public $gender;
    public function __construct($input_name, $input_year, $input_gender){
        $this->name = $input_name;
        $this->year = $input_year;
        $this->gender = $input_gender;
    }
    public function intro(){
        echo "Tôi tên là {$this->name}, sinh năm {$this->year}, giới tính {$this->gender}";
    }
}

$nhân = new SinhVien("Nguyễn Thành Nhân",1993,"Nam");
$nhân->intro();
?>
```

Kế thừa

- Từ khóa **final** dùng để ngăn chặn việc kế thừa lớp, hoặc ngăn chặn việc ghi đè lên phương thức.

```
<?php
final class CongDan{
    public $name;
    public function __construct($input_name){
        $this->name = $input_name;
    }
    public function intro(){
        echo "Tôi tên là {$this->name}";
    }
}
class SinhVien extends CongDan{ //ERROR
    //some code
}
?>
```

Không lớp nào có thể kế thừa lớp CongDan (việc định nghĩa lớp SinhVien kế thừa lớp CongDan sẽ xảy ra lỗi)

Kế thừa

- Từ khóa **final** dùng để ngăn chặn việc kế thừa lớp, hoặc ngăn chặn việc ghi đè lên phương thức.

```
<?php
class CongDan{
    public $name;
    public function __construct($input_name){
        $this->name = $input_name;
    }
    final public function intro(){
        echo "Tôi tên là {$this->name}";
    }
}
class SinhVien extends CongDan{
    public function intro(){ //ERROR
        echo "Chào các bạn, tôi tên là {$this->name}";
    }
}
?>
```

Phương thức intro() sẽ không thể bị ghi đè bên trong các lớp con

Kế thừa

- Một lớp chỉ có thể kế thừa từ một lớp khác
- Lớp con có thể ghi đè/che phương thức lớp cha
 - Để tuy cập phương thức được kế thừa bị che, sử dụng *parent::*
 - Để định nghĩa một phương thức không thể che, thêm từ khóa *final* vào định nghĩa phương thức

```
class ExtendClass extends SimpleClass {  
    // Redefine the parent method  
    function displayVar() {  
        echo "Extending class\n";  
        parent::displayVar();  
    }  
}  
  
$extended = new ExtendClass();  
$extended->displayVar();
```

Hàm tạo

- void **__construct** ([\$args [, \$...]])

```
class BaseClass {  
    function __construct() {  
        print "In BaseClass constructor\n";  
    }  
}  
  
class SubClass extends BaseClass {  
    function __construct() {  
        parent::__construct();  
        print "In SubClass constructor\n";  
    }  
}  
  
$obj = new BaseClass();  
$obj = new SubClass();
```

Hàm tạo

- Hàm tạo - `__construct()` thường được khai báo bên trong lớp, khi tạo một đối tượng từ lớp đó thì hàm tạo sẽ tự động được gọi đến.
- Được sử dụng để thiết lập việc gán giá trị cho các “thuộc tính” của đối tượng thông qua việc tạo một đối tượng mới.

```
• void __construct ([ $args [, $... ] ] )  
class BaseClass {  
    function __construct() {  
        print "In BaseClass  
constructor\n";  
    }  
}  
class SubClass extends BaseClass {  
    function __construct() {  
        parent::__construct();  
        print "In SubClass  
constructor\n";  
    }  
}  
$obj = new BaseClass();  
$obj = new SubClass();
```

Hàm tạo

```
<?php
class Mobile{
    public $model;
    public $color;
    public $price;
    function set_model($input_model){
        $this->model = $input_model;
    }
    function set_color($input_color){
        $this->color = $input_color;
    }
    function set_price($input_price){
        $this->price = $input_price;
    }
    function get_model(){
        return $this->model;
    }
    function get_color(){
        return $this->color;
    }
    function get_price(){
        return $this->price;
    }
}
$nokia = new Mobile();
$nokia->set_model("Nokia 8.1");
$nokia->set_color("Black");
$nokia->set_price("5.000.000");
echo "Model: " . $nokia->model . "<br>";
echo "Màu sắc: " . $nokia->color . "<br>";
echo "Giá tiền: " . $nokia->price;
```

```
<?php
class Mobile{
    public $model;
    public $color;
    public $price;
    function __construct($input_model, $input_color, $input_price){
        $this->model = $input_model;
        $this->color = $input_color;
        $this->price = $input_price;
    }
    function get_model(){
        return $this->model;
    }
    function get_color(){
        return $this->color;
    }
    function get_price(){
        return $this->price;
    }
}
$nokia = new Mobile("Nokia 8.1", "Black", "5.000.000");
echo "Model: " . $nokia->model . "<br>";
echo "Màu sắc: " . $nokia->color . "<br>";
echo "Giá tiền: " . $nokia->price;
```

Hàm hủy

- void **__destruct** (void)

```
class MyDestructableClass {  
    function __construct() {  
        print "In constructor\n";  
        $this->name = "MyDestructableClass";  
    }  
    function __destruct() {  
        print "Destroying " . $this->name . "\n";  
    }  
}  
$obj = new MyDestructableClass();
```


Toán tử chỉ phạm vi (::)

- :: được sử dụng để
 - truy cập hằng
 - truy cập thuộc tính, phương thức tĩnh
 - truy cập phương thức của lớp cha bị ghi đè

Lớp ảo, phương thức ảo (trừu tượng)

- Phương thức ảo

- Được định nghĩa với từ khóa **abstract**
- Chỉ có tên, không có thân

- Lớp ảo

- Được định nghĩa với từ khóa **abstract**
- Không có thể hiện
- **Lớp có phương thức ảo phải được định nghĩa là lớp ảo**

Lớp ảo, phương thức ảo

```
abstract class AbstractClass {  
    // Force Extending class to define this method  
    abstract protected function getValue();  
    abstract protected function prefixValue($prefix);  
  
    // Common method  
    public function printOut() { print $this->getValue() . "\n"; }  
}  
  
class ConcreteClass extends AbstractClass {  
    protected function getValue() { return "ConcreteClass"; }  
    public function prefixValue($prefix) { return  
    "{$prefix}ConcreteClass"; }  
}
```

Interface

- Interface cho phép chỉ định những phương thức mà một lớp nên triển khai.
 - Tất cả các phương thức phải public
- Interface giúp dễ dàng sử dụng nhiều lớp khác nhau theo cùng một cách.
- Khai báo interface:

```
<?php
    interface InterfaceName{
        public function method1();
        public function method2($name, $color);
        public function method3() : string;
    }
?>
```

Interface

- Để triển khai một Interface cho lớp phải sử dụng từ khóa implements.

```
interface iTemplate {  
    public function setVariable($name, $var);  
    public function getHtml($template);  
}  
// Implement the interface  
class Template implements iTemplate {  
    private $vars = array();  
    public function setVariable($name, $var) { $this->vars[$name] = $var; }  
    public function getHtml($template) {  
        foreach($this->vars as $name => $value) {  
            $template = str_replace('{ ' . $name . '}', $value, $template);  
        }  
        return $template;  
    }  
}
```

Interface

- Interface có thể kế thừa từ **nhiều** interface khác
- Một lớp có thể cài đặt nhiều interface

```
interface a {  
    public function foo();  
}  
interface b {  
    public function bar();  
}  
interface c extends a, b {  
    public function baz();  
}  
class D implements a, b {  
}
```

Trait

- Trong PHP, một lớp chỉ được phép kế thừa từ một lớp duy nhất, không thể cho một lớp kế thừa nhiều lớp được.
- ⇒ sử dụng Trait: cho phép kế thừa các thuộc tính & phương thức từ nhiều lớp khác nhau
- ⇒ Để sử dụng một Trait trong lớp thì dùng từ khóa **use**: ***use TraitName;***

Trait

- Trait
 - cung cấp một cơ chế sử dụng lại mã khác kế thừa
 - được định nghĩa tương tự lớp
 - không có thể hiện
- Một lớp hoặc trait có thể sử dụng nhiều trait khác
 - Hữu ích cho những NNLT đơn kế thừa như PHP

```
trait Hello {  
    public function sayHello() {  
        echo 'Hello';  
    }  
}  
  
trait World {  
    public function sayWorld() {  
        echo 'World';  
    }  
}  
  
trait HelloWorld {  
    use Hello, World;  
}
```

```
class MyHelloWorld1 {  
    use HelloWorld;  
}  
  
$o1 = new MyHelloWorld1();  
$o1->sayHello();  
$o1->sayWorld();
```

Hello World

```
class MyHelloWorld2 {  
    use Hello, World;  
    public function sayExclamationMark() {  
        echo '!';  
    }  
}  
  
$o2 = new MyHelloWorld2();  
$o2->sayHello();  
$o2->sayWorld();  
$o2->sayExclamationMark();
```

Hello World!

Giải quyết xung đột

- Xung đột xảy ra khi sử dụng các trait khác nhau có phương thức giống nhau trong cùng một lớp hoặc một trait khác
- Giải quyết
 - Chỉ định sử dụng phương thức của trait nào
 - Đổi tên phương thức trong một trait

```
trait A {  
    public function smallTalk() {  
        echo 'a';  
    }  
    public function bigTalk() {  
        echo 'A';  
    }  
}
```

```
trait B {  
    public function smallTalk() {  
        echo 'b';  
    }  
    public function bigTalk() {  
        echo 'B';  
    }  
}  
  
class Talker {  
    use A, B {  
        B::smallTalk insteadof A;  
        A::bigTalk insteadof B;  
        B::bigTalk as talk;  
    }  
}
```

Có thể thay đổi tính khả kiến với toán tử **as**

```
B::bigTalk as protected;  
B::bigTalk as protected talk;
```

Sao chép đối tượng

- Các biến trong PHP chỉ lưu tham chiếu đến đối tượng
- Nếu cần sao chép đối tượng (thành một thể hiện khác), sử dụng hàm

`$copy_of_object = clone $object;`

* Hàm clone sao chép tất cả các thuộc tính => các thuộc tính tham chiếu vẫn giữ nguyên giá trị tham chiếu

* Sau khi hoàn thành hàm clone, nếu hàm `void __clone(void)` được định nghĩa, hàm `_clone()` của đối tượng mới được tạo được gọi cho phép những thay đổi cần thiết giá trị các thuộc tính

Ví dụ sao chép đối tượng

```
class Class1 {  
    private $var_ = 1;  
    public function printMe() {  
        echo " var_ = $this->var_";  
    }  
    public function changeValue($v_) {  
        $this->var_ = $v_;  
    }  
}
```

```
$obj1 = new Class2();  
$obj2 = clone $obj1;  
$obj1->printMe();  
$obj2->printMe();  
$obj2->changeValues(5, 6);  
$obj1->printMe();  
$obj2->printMe();
```

```
class Class2 {  
    private $var = 2;  
    private $obj;  
  
    public function __construct() {  
        $this->obj = new Class1();  
    }  
  
    public function printMe() {  
        echo "<br>var = $this->var";  
        $this->obj->printMe();  
    }  
  
    public function changeValues($v, $v_) {  
        $this->var = $v;  
        $this->obj->changeValue($v_);  
    }  
}
```

```
var = 2 var_ = 1  
var = 2 var_ = 1  
var = 2 var_ = 6  
var = 5 var_ = 6
```

Ví dụ sao chép đối tượng

```
class Class1 {  
    private $var_ = 1;  
    public function printMe() {  
        echo " var_ = $this->var_";  
    }  
    public function changeValue($v_) {  
        $this->var_ = $v_;  
    }  
}
```

```
$obj1 = new Class2();  
$obj2 = clone $obj1;  
$obj1->printMe();  
$obj2->printMe();  
$obj2->changeValues(5, 6);  
$obj1->printMe();  
$obj2->printMe();
```

```
class Class2 {  
    private $var = 2;  
    private $obj;  
  
    public function __construct() {  
        $this->obj = new Class1();  
    }  
  
    public function __clone() {  
        $this->obj = new Class1();  
    }  
    public function printMe() {  
        echo "<br>var = $this->var";  
        $this->obj->printMe();  
    }  
  
    public function changeValues($v, $v_) {  
        $this->var = $v;  
        $this->obj->changeValue($v_);  
    }  
}
```

```
var = 2 var_ = 1  
var = 2 var_ = 1  
var = 2 var_ = 1  
var = 5 var_ = 6
```

Không gian tên

- Không gian tên (namespace) được sử dụng để nhóm các lớp, giao diện, hàm và hằng nhằm tránh đụng độ khi sử dụng lại mã do trùng tên
 - Ở các ngôn ngữ khác:
 - .NET: namespace
 - Java: package

Định nghĩa không gian tên

- Sử dụng từ khóa namespace để định nghĩa không gian tên

```
<?php
```

```
    namespace MyNameSpace {
```

```
        const CONNECT_OK = 1;
```

```
        interface Conn { /*...*/}
```

```
        class Connection { /* ... */ }
```

```
        function connect() { /* ... */ }
```

```
    }
```

```
?>
```

Không gian tên lồng nhau

- Sử dụng cú pháp biểu diễn thư mục
<?php

```
namespace Parent\Child\GrandChild {  
    const CONNECT_OK = 1;  
    interface Conn { /*...*/}  
    class Connection { /* ... */}  
    function connect() { /* ... */}  
}  
?>
```

Không gian tên toàn cục

- Các lớp, giao diện, hàm, hằng không được định nghĩa trong một không gian tên nào được coi nằm trong không gian tên toàn cục (\)
- Có thể sử dụng namespace không có tên để biểu thị không gian tên toàn cục

`namespace { /*Không gian tên toàn cục*/ }`

- Các không gian tên khác được xem như nằm trong không gian tên toàn cục.

`\`

`\namespace1`

`\namespace1\subnamespace1`

`\namespace2`

`\namespace2\subnamespace2`

`\namespace2\subnamespace2\subsubnamespace2`

Tên đầy đủ trong không gian tên

- Tên đầy đủ của lớp, giao diện, hàm, hằng bao gồm không gian tên phía trước

`\namespace\ClassName`

`\namespace\InterfaceName`

`\namespace\functionName`

`\namespace\CONSTANT_NAME`

Phân giải tên

- Khi lớp, giao diện không được viết với tên đầy đủ
 - Chúng được hiểu là thuộc không gian tên hiện tại
- Khi hàm, hằng không được viết với tên đầy đủ
 - Chúng được hiểu là thuộc không gian tên hiện tại
 - Hoặc thuộc không gian tên toàn cục nếu không tìm thấy trong không gian tên hiện tại

Ví dụ phân giải tên

```
namespace ns1 {  
    class A {  
        private $var = 1;  
        public function a() {  
            echo "<br>ns1\A->var: $this->var";  
        }  
    }  
}  
  
namespace ns2\{  
    class A {  
        private $var = 3;  
        public function a() {  
            echo "<br>ns3\A->var: $this->var";  
        }  
    }  
}
```

```
namespace ns2 {  
    class A {  
        private $var = 2;  
        public function a() {  
            echo "<br>ns2\A->var: $this->var";  
        }  
    }  
}  
  
$obj1 = new \ns1\A();  
$obj2 = new A();  
$obj3 = new ns3\A();  
  
$obj1->a();  
$obj2->a();  
$obj3->a();  
}
```

```
ns1\A->var: 1  
ns2\A->var: 2  
ns3\A->var: 3
```

Nhập và đặt bí danh

- Để không phải viết tên đầy đủ (dài), PHP cho phép nhập và đặt bí danh cho không gian tên, lớp, và giao diện
- `use ns\subns\Classname;` //sau đó chỉ cần sử dụng `Classname` thay cho tên đầy đủ
- `use ns\subns\Classname as Another;` //sau đó sử dụng `Another` thay cho tên đầy đủ
- `use ns\subns\NSname;` //sau đó sử dụng `NSname` thay cho tên đầy đủ

Xử lý ngoại lệ

- Ném ngoại lệ

```
throw new Exception("Mô tả ngoại lệ");
```

- Bắt ngoại lệ

```
try {
```

```
    //mã xử lý nghiệp vụ
```

```
} catch (Exception $e) {
```

```
    //nếu có ngoại lệ xảy ra ở khối try thì mã xử lý ngoại lệ ở khối catch được thực hiện. Sử dụng $e->getMessage() để lấy mô tả ngoại lệ
```

```
} [catch (OtherException $oe) {
```

```
    //Có thể nhiều khối catch sau khối try. Mỗi khối catch bắt một loại ngoại lệ
```

```
}]*
```

```
[finally {
```

```
    Mã được chạy bất kể ngoại lệ đã xảy ra hay không
```

```
} ]
```

Ví dụ

```
function inverse($x) {  
    if (!$x) {  
        throw new Exception('Division by zero.');    }  
    else return 1/$x;  
}  
  
try {  
    echo inverse(5) . "\n";  
    echo inverse(0) . "\n";  
} catch (Exception $e) {  
    echo 'Caught exception: ', $e->getMessage(), "\n";  
}
```

0.2

Caught exception: Division by
zero.

Các biến dựng sẵn

- Trong PHP có những biến toàn cục dạng mảng được định nghĩa trước. Các biến này có thể truy cập ở bất kỳ đâu:
 - `$GLOBALS` — Mảng các biến toàn cục
 - `$_SERVER` — Mảng các biến máy chủ
 - `$_GET` — Mảng các biến GET
 - `$_POST` — Mảng các biến POST
 - `$_FILES` — Mảng các tệp upload
 - `$_REQUEST` — Mảng các biến Request (cả GET và POST)
 - `$_SESSION` — Mảng các biến phiên
 - `$_ENV` — Mảng các biến môi trường
 - `$_COOKIE` — Mảng các biến Cookies

\$GLOBALS

- \$GLOBALS được sử dụng để truy cập các biến toàn cục từ bất kỳ đâu trong tập lệnh PHP
- \$GLOBALS là một mảng chứa tất cả các tham chiếu đến các biến toàn cục. Với key của mảng là tên biến.

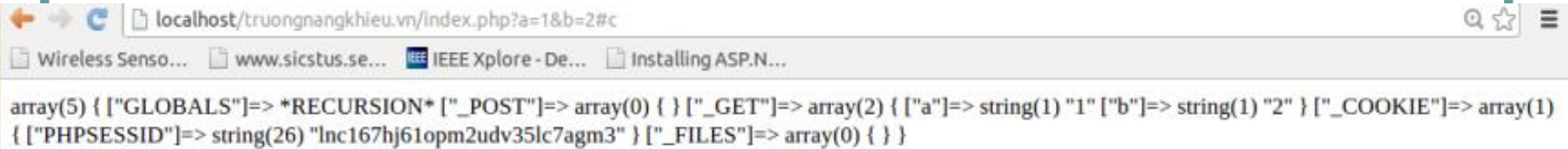
```
<?php
    $x = 75;
    $y = 25;

    function addition() {
        $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
    }

    addition();
    echo $z;
?>
```


\$GLOBALS

```
<?php var_dump($GLOBALS); ?>
```



The screenshot shows a web browser window with the address bar displaying `localhost/truongnangkhiu.vn/index.php?a=1&b=2#c`. The browser's tab bar shows several open tabs, including "Wireless Senso...", "www.sicstus.se...", "IEEE Xplore - De...", and "Installing ASP.N...". The main content area of the browser displays the output of the PHP script, which is a PHP array dump:

```
array(5) { ["GLOBALS"]=> *RECURSION* ["_POST"]=> array(0) { } ["_GET"]=> array(2) { ["a"]=> string(1) "1" ["b"]=> string(1) "2" } ["_COOKIE"]=> array(1) { ["PHPSESSID"]=> string(26) "lnc167hj61opm2udv35lc7agm3" } ["_FILES"]=> array(0) { } }
```

\$_SERVER

- \$_SERVER là một mảng chứa các thông tin do Server sinh ra như Header, Path, Location Script. Một số thông tin có thể lấy được qua biến này như:
 - \$_SERVER['HTTP_HOST']: Trả về Host của trang (Có thể hiểu là tên miền của trang)
 - \$_SERVER['DOCUMENT_ROOT']: Trả về Path của trang (Ví dụ: /Applications/AMPPS/www)
 - \$_SERVER['QUERY_STRING']: Lấy chuỗi truy vấn trên URL (Ví dụ: ?username=webat)
 - \$_SERVER['REQUEST_URI']: Trả về URI của trang (Ví dụ: /phpbasic/mang.php)

\$_SERVER

- Một số thông tin có thể lấy được qua biến `$_SERVER`:
 - `$_SERVER['HTTP_REFERER']`: Trả về URL trước khi truy cập tới trang hiện tại (Thường được dùng để thống kê nguồn truy cập)
 - `$_SERVER['SCRIPT_NAME']`: Trả về đường dẫn của file PHP hiện tại
 - `$_SERVER['REQUEST_METHOD']`: Trả về phương thức gửi Request đến Server (POST, GET, PUT,...)
 - `$_SERVER['REQUEST_TIME']`: Trả về timestamp lúc gửi yêu cầu đến Server

\$_SERVER

- Một số thông tin có thể lấy được qua biến `$_SERVER`:
 - `$_SERVER['HTTP_USER_AGENT']`: Trả về User-Agent là header của yêu cầu gửi đến, qua cái này có thể biết được thông tin như Trình duyệt, Hệ điều hành, Thiết bị ... đang truy cập
 - `$_SERVER['REMOTE_ADDR']`: Trả về IP đang truy cập

\$_SERVER

<?php var_dump(\$_SERVER); ?>



```
array(27) ( ["HTTP_HOST"]=> string(9) "localhost" ["HTTP_CONNECTION"]=> string(10) "keep-alive" ["HTTP_CACHE_CONTROL"]=> string(9) "max-age=0"
["HTTP_ACCEPT"]=> string(74) "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8" ["HTTP_USER_AGENT"]=> string(102)
"Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/30.0.1599.66 Safari/537.36" ["HTTP_ACCEPT_ENCODING"]=> string(17)
"gzip,deflate,sdch" ["HTTP_ACCEPT_LANGUAGE"]=> string(14) "en-US,en;q=0.8" ["HTTP_COOKIE"]=> string(36)
"PHPSESSID=inc167hj61opm2udv35lc7agm3" ["PATH"]=> string(28) "/usr/local/bin:/usr/bin:/bin" ["SERVER_SIGNATURE"]=> string(70) "
Apache/2.2.22 (Ubuntu) Server at localhost Port 80
" ["SERVER_SOFTWARE"]=> string(22) "Apache/2.2.22 (Ubuntu)" ["SERVER_NAME"]=> string(9) "localhost" ["SERVER_ADDR"]=> string(9) "127.0.0.1"
["SERVER_PORT"]=> string(2) "80" ["REMOTE_ADDR"]=> string(9) "127.0.0.1" ["DOCUMENT_ROOT"]=> string(8) "/var/www" ["SERVER_ADMIN"]=> =>
string(19) "webmaster@localhost" ["SCRIPT_FILENAME"]=> string(37) "/var/www/truongnangkhieu.vn/index.php" ["REMOTE_PORT"]=> string(5) "43736"
["GATEWAY_INTERFACE"]=> string(7) "CGI/1.1" ["SERVER_PROTOCOL"]=> string(8) "HTTP/1.1" ["REQUEST_METHOD"]=> string(3) "GET"
["QUERY_STRING"]=> string(7) "a=1&b=2" ["REQUEST_URI"]=> string(37) "/truongnangkhieu.vn/index.php?a=1&b=2" ["SCRIPT_NAME"]=> => string(29)
"/truongnangkhieu.vn/index.php" ["PHP_SELF"]=> string(29) "/truongnangkhieu.vn/index.php" ["REQUEST_TIME"]=> => int(1382279758) }
```

\$_GET và \$_POST

- \$_GET là một mảng kết hợp, lưu thông tin gửi đến bằng phương thức HTTP GET, hay nói cách khác lưu các thông tin chuyển đến thông qua tham số từ URL
- \$_POST là một mảng kết hợp, lưu thông tin gửi đến bằng phương thức HTTP POST.

\$_GET và \$_POST

- PHP `_GET` và `_POST` là hai phương thức dùng để thu thập dữ liệu form, đây cũng là phương thức dùng để chuyển dữ liệu từ máy client lên máy chủ (server).
- Cả hai `_GET` và `_POST` đều tạo một mảng với cặp key/value, với key chính là thuộc tính name của các thành phần form, còn value chính là giá trị của thành phần đó với name tương ứng.
- `_GET` thường dùng cho dữ liệu không quan trọng, không cần bảo mật, thể hiện rõ trên tham số khi submit hay click từ liên kết.
- `_POST` thường dùng cho dữ liệu quan trọng, cần bảo mật như thông tin login, payment, ... `_POST` nhận được thông qua phương thức HTTP.

\$_GET

- `_GET` có thể được nhận biết thông qua 2 cách:
 - `_GET` thu thập dữ liệu form sau khi được gửi (submit) thông qua thuộc tính `method="get"`
 - `_GET` cũng có thể thu thập dữ liệu được gửi thông qua đường dẫn URL trên thanh địa chỉ.

Điện thoại:

Gửi

Điền giá trị 0123456 và click nút Gửi, thông tin sẽ gửi tới trang xử lý `php_get_post.php` (hiện tại đang sử dụng chính trang hiện hành), khi này đường dẫn trên thanh địa chỉ có dạng `php_get_post.php?phone=0123456`

```
<form action="php_get_post.php" method="get">
    Điện thoại: <input type="text" name="phone">
    <button type="submit">Gửi</button>
</form>
```

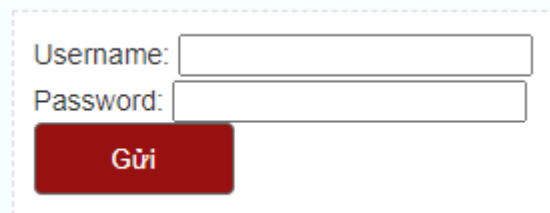
```
Thông tin nhận được <?php if(isset($_GET["phone"])) { echo $_GET["phone"]; } ?>
```

Xử lý giá trị

- Cách lấy dữ liệu `_GET`, ta sử dụng cấu trúc `$_GET[key]`.
- Dùng câu lệnh `if` để xác định xem có tồn tại phương thức `_GET` hay không, trước khi nhận giá trị.

\$_POST

- Với phương thức POST thì trình duyệt không thể hiện tham số trên thanh địa chỉ.



Username:

Password:

```
<form action="php_get_post.php" method="post">
    Username: <input type="text" name="user"><br>
    Password: <input type="password" name="password"><br>
    <button type="submit">Gửi</button>
</form>
```

```
Username vừa nhập: <?php if(isset($_POST["user"])) { echo $_POST["user"]; } ?> <br>
Password vừa nhập: <?php if(isset($_POST["password"])) { echo $_POST["password"]; } ?>
```

Xử lý giá trị

- Tương tự như cách lấy dữ liệu của `_GET`, đối với `_POST` cũng sử dụng cấu trúc tương tự `$_POST[key]`.
- Dùng câu lệnh `if` để xác định xem có tồn tại phương thức `_POST` hay không, trước khi nhận giá trị.

PHP xử lý form

- Các giá trị thành phần của form được thu thập thông qua phương thức `_GET` và `_POST`.
- Các giá trị thành phần của form được xử lý dựa theo thuộc tính *name* của từng thành phần tương ứng và thông qua thuộc tính *method* của form để xác định phương thức truyền dữ liệu.
- Các thành phần thường dùng của form:
 - PHP input type="text"
 - PHP input type="password"
 - PHP input type="checkbox"
 - PHP input type="radio"
 - PHP select option
 - PHP textarea

PHP xử lý form

Họ tên:

Password:

Đăng ký học: HTML ☐, CSS ☐

Giới tính: Nam ☐, Nữ ☐

Thành phố:

Tin nhắn:

```
<form action="" method="post">
    <p>Họ tên: <input type="text" name="fullname" value=""></p>
    <p>Password: <input type="password" name="password" value=""></p>
    <p>Đăng ký học: HTML <input type="checkbox" name="check_html" value="HTML">, CSS <input
type="checkbox" name="check_css" value="CSS"></p>
    <p>Giới tính: Nam <input type="radio" name="gender" value="Nam">, Nữ <input type="radio"
name="gender" value="Nữ"></p>
    <p>Thành phố: <select name="city">
        <option value="Hà Nội">Hà Nội</option>
        <option value="Hồ Chí Minh">Hồ Chí Minh</option>
        <option value="Đà Nẵng">Đà Nẵng</option>
        <option value="Cần Thơ">Cần Thơ</option>
    </select></p>
    <p>Tin nhắn:<br>
    <textarea cols="30" rows="7" name="message"></textarea></p>
    <button type="submit">Gửi</button>
</form>

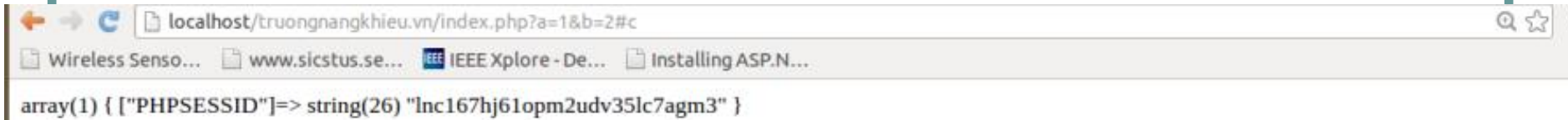
<p>Họ tên: <?php if(isset($_POST["fullname"])) { echo $_POST["fullname"]; } ?></p>
<p>Password: <?php if(isset($_POST["password"])) { echo $_POST["password"]; } ?></p>
<p>Đăng ký học: <?php if(isset($_POST["check_html"])) { echo $_POST["check_html"]; } ?> <?php
if(isset($_POST["check_css"])) { echo $_POST["check_css"]; } ?></p>
<p>Giới tính: <?php if(isset($_POST["gender"])) { echo $_POST["gender"]; } ?></p>
<p>Thành phố: <?php if(isset($_POST["city"])) { echo $_POST["city"]; } ?></p>
<p>Tin nhắn: <?php if(isset($_POST["message"])) { echo $_POST["message"]; } ?></p>
```

Các biến dựng sẵn

- Biến `$_REQUEST` là một mảng kết hợp, lưu trữ thông tin chứa trong biến `$_GET`, `$_POST` và `$_COOKIE`
- `$_COOKIE` là một mảng để truy cập thông tin cookie do trình duyệt gửi lên
- `$_SESSION` là một mảng để lưu trữ thông tin phiên làm việc
- Biến `$_FILES` lưu thông tin các file upload lên server

\$_COOKIE

```
<?php var_dump($_COOKIE); ?>
```



Tạo tiêu đề gói HTTP Response

```
void header ( string $string [, bool $re  
    place =  
    true [, int $http_response_code ] ] )
```

- Ví dụ

```
header("HTTP/1.0 404 Not Found");  
header("Location: http://www.example.co  
    m/"); /* Redirect browser */  
header('WWW-Authenticate: Negotiate');
```

Các hàm khác

- Các hàm để đặt cookie, session, chuyển đổi biểu diễn địa chỉ ip, ... (xem ở các bài sau và trong tài liệu tham khảo)

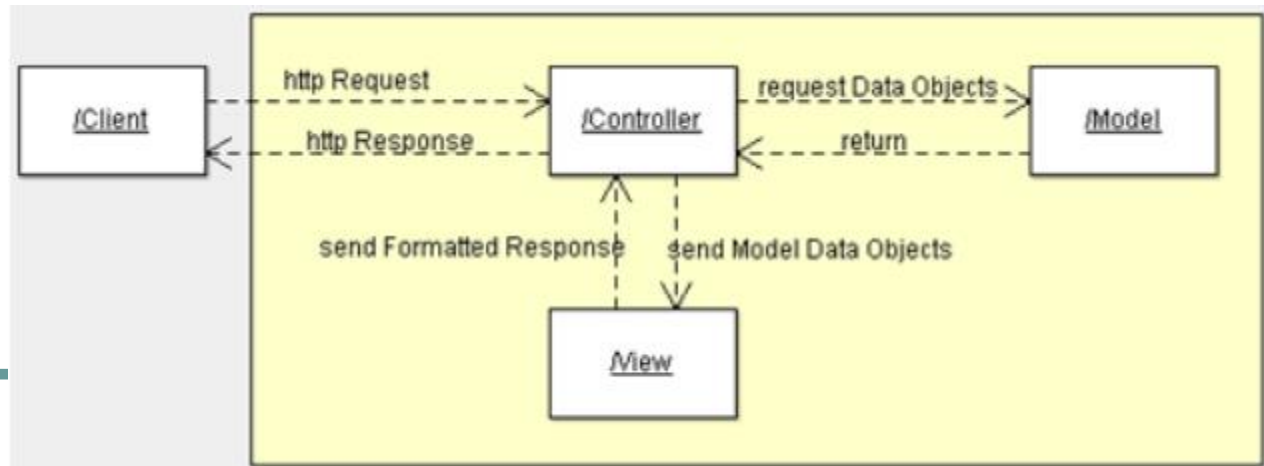
NỘI DUNG

1. Kiến trúc của một ứng dụng web động
2. Ngôn ngữ lập trình web phía server
3. PHP
4. **Mô hình MVC**

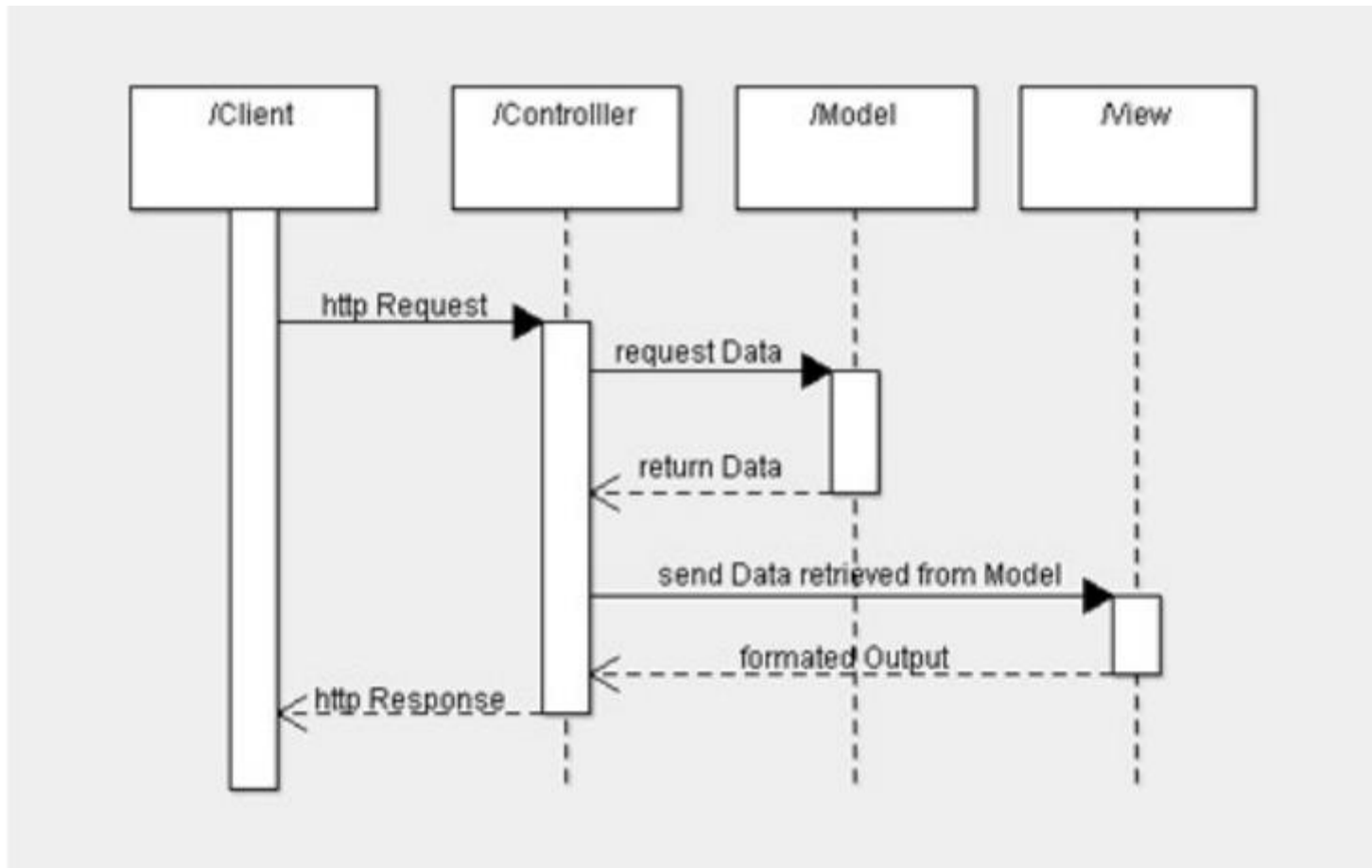


Model – View - Control

- MVC là mẫu thiết kế được sử dụng rộng rãi cho ứng dụng web
 - **Model:** Xử lý logic của ứng dụng, cung cấp dữ liệu, thường thao tác với CSDL
 - **View:** Hiển thị dữ liệu dưới định dạng cụ thể (với web là định dạng HTML)
 - **Control:** Giao tiếp với client, điều phối model và view làm việc



Model – View - Control



Model – View - Control

- MVC hỗ trợ phát việc phát triển nhanh chóng và song song. Nếu một mô hình MVC được dùng để phát triển bất kỳ ứng dụng web cụ thể nào, một lập trình viên có thể làm việc trên View và một developer khác có thể làm việc với Controller để tạo logic nghiệp vụ cho ứng dụng web đó.
- Do đó, ứng dụng mô hình MVC có thể được hoàn thành nhanh hơn ba lần so với các ứng dụng mô hình khác.
- MVC đa phần phù hợp với các dự án lớn; với các dự án nhỏ, lẽ vì khá là cồng kềnh và mất thời gian.

Ví dụ: Phương trình bậc 2

- Model: Phương trình $a*x*x + b*x + c = 0$
 - Thành viên dữ liệu
 - a, b, c: Các hệ số của phương trình
 - retArray: Nghiệm của phương trình
 - Thành viên hàm
 - solve(): Giải phương trình
 - getValues(): Trả về nghiệm của phương trình

Ví dụ: Phương trình bậc 2

```
class QuadraticModel {  
    private $a; //he so bac hai  
    private $b; //he so bac nhat  
    private $c; //he so tu do  
    private $retArray; //ket qua: [so nghiem, nghiem 1, nghiem 2]  
  
    public function __construct($a, $b, $c) {  
        $this->a = $a;  
        $this->b = $b;  
        $this->c = $c;  
        $this->retArray = array();  
        $this->solve();  
    }  
    public function getValues() { return $this->retArray; }
```

Ví dụ: Phương trình bậc 2

```
/**Giai phuong trinh**/  
private function solve() {  
    if ($this->a == 0) {  
        if ($this->b == 0) {  
            if ($this->c == 0) $this->retArray[] = -1;  
            else $this->retArray[] = 0;  
        } else {  
            $this->retArray[] = 1;  
            $this->retArray[] = -$this->c/$this->b;  
        }  
    } else {  
        $delta = $this->b*$this->b - 4*$this->a*$this->c;  
        if ($delta > 0) {  
            $this->retArray[] = 2;  
            $this->retArray[] = (-$this->b+sqrt($delta))/2/$this->a;  
            $this->retArray[] = (-$this->b-sqrt($delta))/2/$this->a;  
        } else if ($delta == 0) {  
            $this->retArray[] = 1;  
            $this->retArray[] = -$this->b/2/$this->a;  
        } else {  
            $this->retArray[] = 0;  
        }  
    }  
}
```

- View: Hiển thị kết quả
 - Thành viên dữ liệu
 - arr: Nghiệm của phương trình
 - Thành viên hàm
 - render(): Tạo nội dung web từ dữ liệu

```
class QuadraticView {
    private $arr; //du lieu tho, la ket qua cua xu ly nghiep vu, can gui ve cho client
    public function __construct($arr) {
        $this->arr = $arr;
    }

    /**Noi dung web se gui ve cho client*/
    public function render() {
        $html = "";
        if ($this->arr[0] == 0) $html .= "Phuong trinh vo nghiem";
        else if ($this->arr[0] == 1) $html .= "Phuong trinh co MOT nghiem x =
".$this->arr[1];
        else if ($this->arr[0] == 2) $html .= "Phuong trinh co HAI nghiem x1 =
".$this->arr[1].", x2 = ".$this->arr[2];
        return $html;
    }
}
```


- Control: Nhận các tham số của người dùng, điều phối model và view hoạt động, trả kết quả cho người dùng
 - Thành viên dữ liệu
 - Không
 - Thành viên hàm
 - act(): Thực hiện các công việc nêu trên

```

class QuadraticControl {
    public function act() {
        if (isset($_GET["a"]) && isset($_GET["b"]) && isset($_GET["c"])) {
//1. Nhan yeu cau, xu ly cac tham so
            $a = EscapeShellCmd($_GET["a"]);
            $b = EscapeShellCmd($_GET["b"]);
            $c = EscapeShellCmd($_GET["c"]);
//2. Goi model de xu ly nghiep vu
            require_once("model.php");
            $qmodel = new QuadraticModel($a, $b, $c);
            $sarr = $qmodel->getValues(); //ket qua xu ly nghiep vu
//3. Goi view de tao noi dung web
            require_once("view.php");
            $qview = new QuadraticView($sarr);
            $html = $qview->render();
//4. Tra loi client
            echo $html;
        } else {
            echo "Nhap a, b, c. Vi du ?a=3&b=-4&c=1";
        }
    }
}

$qcontrol = new QuadraticControl();
$qcontrol->act();

```

The slide features a solid purple header bar at the top. Below it, a large white rectangular area is framed by a thin teal border with rounded corners. The text "The End!" is centered within this white area.

The End!