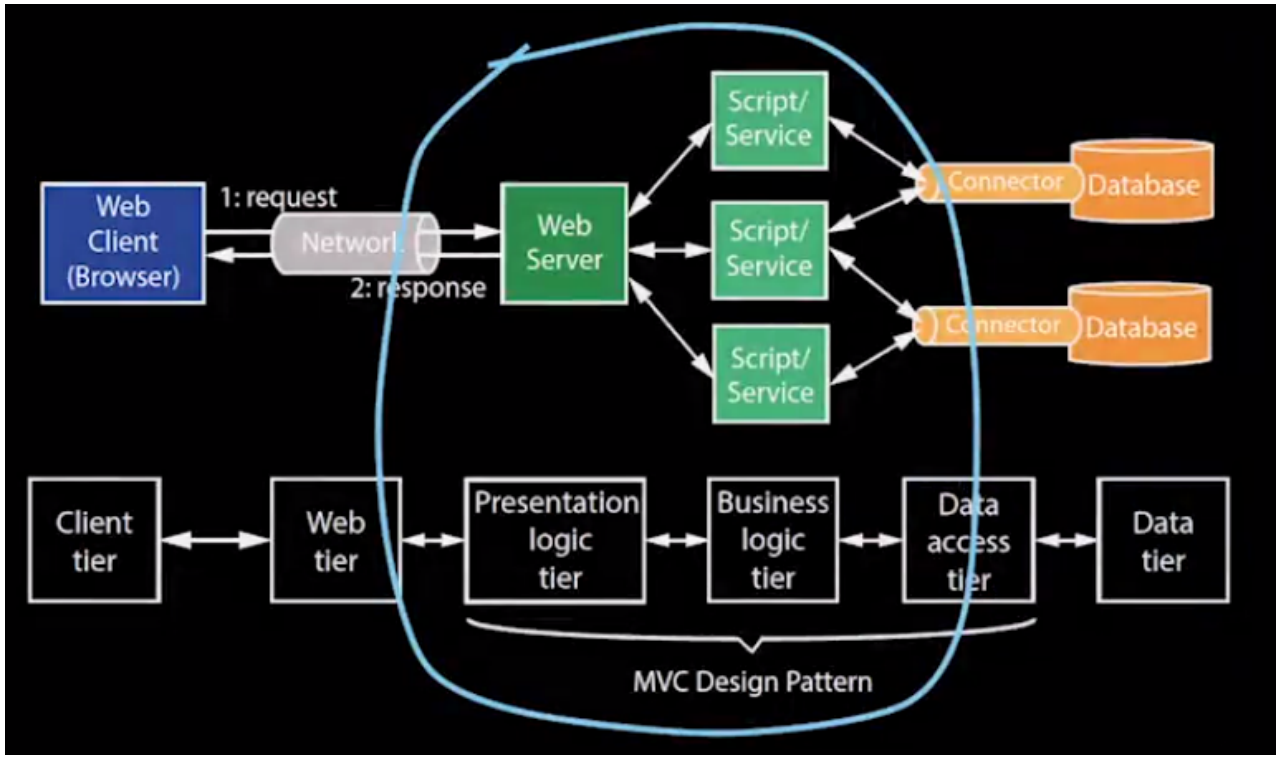
Overview

Focus of Attention

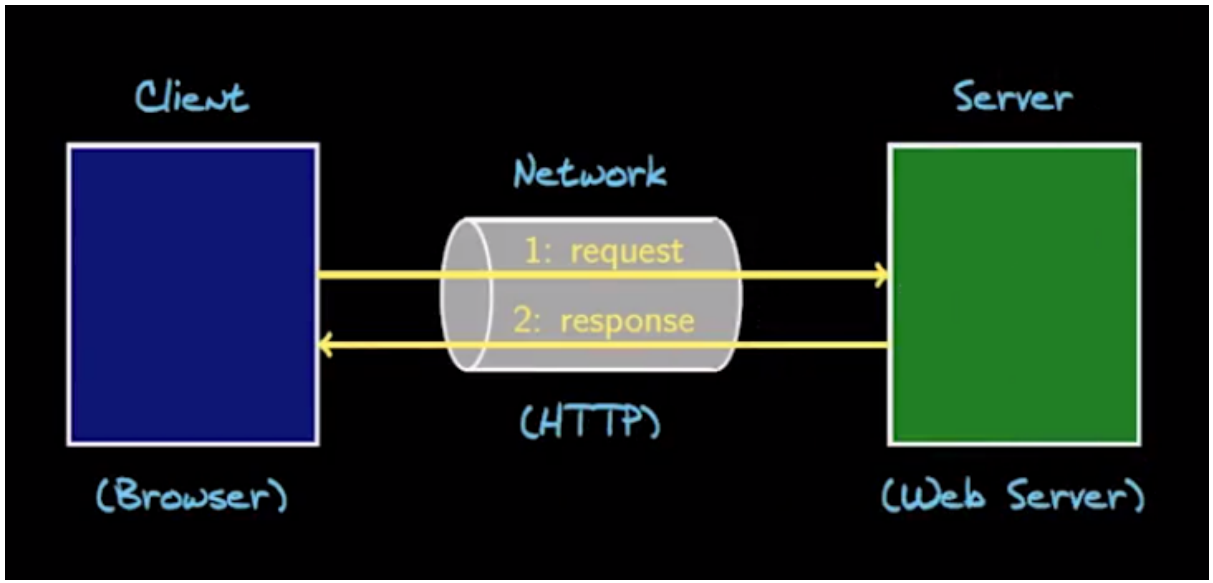


What is Middleware

- Middleware is the "software glue" between the operating system and application on each side of a client-server architecture, some refer to is as the "dash" in client-server
- Middleware as the software that provides services to applications beyone those available from the underlying operating system -it connects applications running on the server side, and passes data between them
- Thus, middleware allows multiple processes running on different machines to interact (where natively they would not be able to).

The Hypertext Transfer Protocol (HTTP)

- HTTP is the foundation for data communication on the web, and that it involves request/response inertactions:



- HTTP is an application layer protocol used to deliver resources in distributed hypermedia information systems. In a web application, the request initiates activities that are implemented over the middleware , and the response typically involves returning resources to the browser.
- In order to build and debug web applications, it's vital to have a good understanding of how HTTP works.

HTTP - Resources

- The resources delivered as part of this protocol typically include hypertext, marked up using the HyperText Markup Language(HTML), cascading style sheet (CSS), hypermedia and scripts
  - **Hypertext** - text that can be displayed on a computer, or other display device, possibly styled with CSS, and containing references(i.e., hyperlinks) to other hypertext that the reader is able to immediately access, usually via a mouse click.
  - **Hypermedia** - the logical extension of hypertext to graphics, audio and video.
  - **Hyperlinks** - define a structure over the web.Indeed, this is the structure that Google uses to determine the relevance of hyperlinks that are returned to you by a search
  - **Scripts** - code that can be executed on the client side.

HTTP - Background

- The HTTP protocol is extremely lightweight and simple - indeed, that's one of the main reasons for its sucess.
- Initially, with HTTP/0.9 (the first documented HTTP protocol), a client could only issue GET requests, asking a server for a resource.
  Ex. GET /welcome.html
  will cause the server to return the contents of the requested file(the response was required to be HTML)
- The HTTP/1.0 protocol, introduced in 1996, extended HTTP/0.9 to include request headers along with additional request methods.

HTTP - Background

- The HTTP/1.1 extension followed soon thereafter, and included the following improvements:
  - Faster response, by allowing multiple transactions to take place over a single persistent connection.
  - Faster response and bandwidth savings, by adding cache support.
  - Faster response for dynamically-generated content, by supporting chunked encoding, which allows a response to be sent before its total length is known
  - Effient use of IP addresses, multiple domains can be served from a single IP address
  - Support for proxies.

HTTP - Basic

- HTTP has always been **a stateless protocol**
- This refers to the fact that the protocol does not require the server to retain information related to previous client requests.
- Thus, each client request is executed independently , without any knowledge of the client requests that preceded it.
- Cookies, sessions, URL encoded parameters and a few other technologies have been introduced to address this issue, thereby allowing for the emergence of Web 2.0 and 3.0 applications

HTTP - Sessions

- An HTTP session proceeds as follows:
  1. An HTTP client(e.g., a browser) establishes a TCP connection to a particular port on a host server(typically this is port 80) , an initiates a request. Establishing the TCP connection may first involve using DNS server in order to obtain an IP Address.
  2. An HTTP server listening on that prt waits for a client's request message.
  3. Upon receiving the request, the server processes it and sends back a status line, such as "HTTP/1.1 200 OK", along with a message of its own (i.e., a response), the body of which might be a requested resource, an error message, or some other information

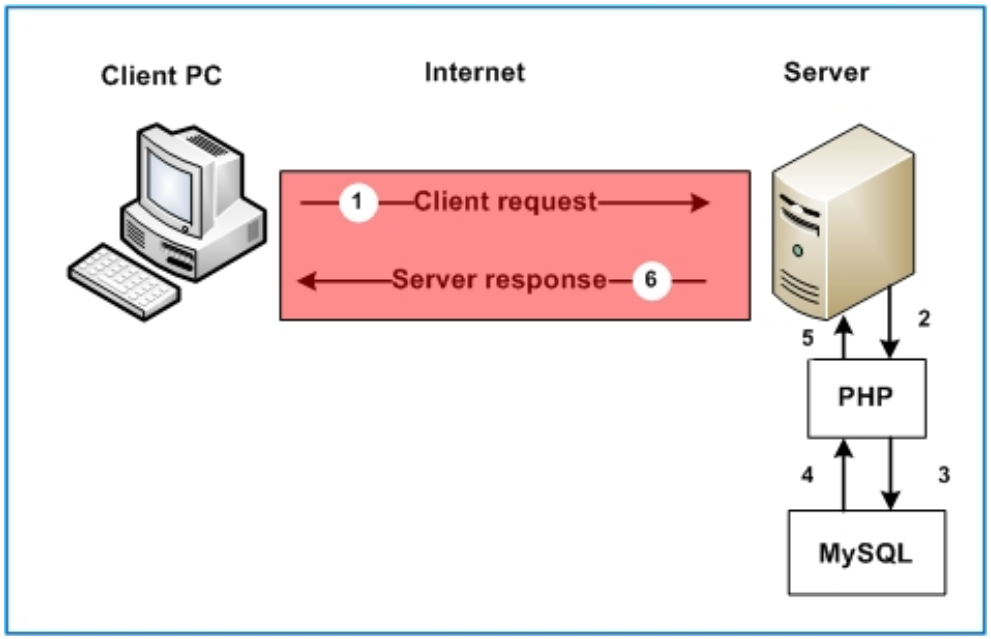
Server-Side Scripting Languages

- Often, dynamic pages are mostly static
  - A lot of static information: page headers, layout, etc.
  - A few really dynamic parts inside
- Let's design pages by writing static HTML documents with **pieces of code** inside
  - The code is **executed by the server** at request time
  - Clients only see "regular" pages, **without code statements**

PHP

- The most popular third party module for Apache
  - Code and extensive docs available from *http* : *//www.php.net/*
- PHP-enabled web pages use the *.php* suffix not *.html*
- PHP is part of what is commonly refereed to as the LAMP stack

PHP


Client PC
Internet
Server
1
Client request
Server response
6
5
2
PHP
4
3
MySQL

What do I need to run PHP

- Apache
- PHP
- MySQL.

PHP Syntax

- Syntax defines the rules for writing code using a specific programming language
- PHP syntax is very similar to $C++$, C, Java,C#
  - Like these other programming languages PHP marks the end of a statement with a ';'
  - As a beginner expect to get error messages from the PHP preprocessor for **"forgotten"** semicolons.

PHP Key Words

- The syntax (rules) define "key" words that have special meaning in the programming language

PHP Keywords

__halt_compiler()	abstract	and	array()	as
break	callable (as of PHP 5.4)	case	catch	class
clone	const	continue	declare	default
die()	do	echo	else	elseif
empty()	enddeclare	endfor	endforeach	endif
endswitch	endwhile	eval()	exit()	extends
final	finally (as of PHP 5.5)	for	foreach	function
global	goto (as of PHP 5.3)	if	implements	include
include_once	instanceof	insteadof (as of PHP 5.4)	interface	isset()
list()	namespace (as of PHP 5.3)	new	or	print
private	protected	public	require	require_once
return	static	switch	throw	trait (as of PHP 5.4)
try	unset()	use	var	while
xor	yield (as of PHP 5.5)			

PHP Tag

- PHP code may be contained within HTML code; therefore, we need a mechanism to identify the PHP code for the server's PHP preprocessor.
- We differentiate PHP code by enclosing it within the $<?php?>$ tags.
  - The short form $<?...?>$ is not considered good practice as it may limit the portability of our code.

Variables

- PHP variables are "loosely-typed".
  - We are not required to tell PHP the type of the data to be stored

```
<?php // The tage that begin the PHP code section
$Integer_variable = 42;
$String_variable = "I␣Love␣this␣course";
$Double_variable = 3.13159;
echo $Integer_variable."<br>";
echo $String_variable."<br>";
echo $Double_variable."<br/>";
echo "----------------------<br>";
$String_variable = 42;
$Integer_variable = "I␣really␣love";
$Double_variable = "this␣course";
echo $String_variable."<br>";
echo $Integer_variable.$Double_variable;
// The tag that ends the PHP code section
?>
```

Variables

- A variable is created by assigning a value to it.
  - $MyVariable = 42 creates a variable that stores the number 42.
- Variables have life spans; the area within the PHP code within which the variable is accessible commonly referred to as scope.
  - There are three variable scopes:local, global, and static.
- Variables with local scope can only be accessed within the function in which the variable was created.
- Variables with global scope are created outside of a function making them accessible by any function

Superglobals

- Superglobals are built-in variables populated by PHP that can be accessed from anywhere within a PHP application
  - $_SERVER - information about the web server and HTTP connection
  - $_COOKIE - information about any cookie data sent to the server
  - $_REQUEST - combination of information from $_COOKIE, $_POST, and $_GET
  - $_SESSION - information we stored about a user's interaction with the application
  - $_ENV - information about server environment

Single(') versus Double (") Quotes

- Single quotes will not interpret the value of variable enclosed within them. It is said that single quotes display values "as-is"
- Double quotes will interpret the value of variables and special characters.

IDE and Variable

Demonstration

Operators

- Operators give us the ability to manipulate the value of data within our code.
  - Operators can act on variables or literal values. A literal value is a value that is included in your code directly
    - echo 42 // 42 is a literal value
- PHP provides a plethora of operators.
  - Arithmetic
  - Comparison
  - String
  - Assignment
  - Combination

Arithmetic Operators

- The arithmetic operators perform standard mathematical operations on values.
  - Addition "+"
  - Subtraction "-"
  - Multiplication "*"
  - Division "/"
  - modulus "%"

Example Arithmetic Operators

- A loop that makes colored rows

```
for ($i=1;$i<=10;$i++)
{
        if (($i%2)==1) // odd
        {
            echo "<div class=\"dark\">$i</div>";
        }
        else
        {
         // even
            echo "<div class=\"light\">$i</div>";
        }
}
```

Comparison Operators

- Comparison operators return TRUE or FALSE when comparing two values.
  - Equal TO "=="
  - Not Equal To "!="
  - Identical "==="
  - Less Than "¡"
  - Greater Than "¿"
  - Less Than or Equal To "¡="
  - Greater Than or Equal To "¿="

Logical Operators

- Use the logical operators when you want to combine for common operations that involve two operators.
  - Plus Equals "+="
  - Minus Equals "-="
  - ...
- A very common task is to increase or decrease the value of an integer variable by one.
  - Use the Pre or Post Increment or Decrement Operator
    - $myVariable++ / $myVariable−−
    - ++$myVariable/−−$myVariable

Operator Precedence

- Operator precedence defines the order in which the operators are evaluated by PHP.

Operator	Precedence
++ −−	Evaluated first
!&	...
+ -	...
...	...
AND	...
OR	...
XOR	Evaluated last

Controlling the Flow

- Controlling the flow of execution allows us to execute a set of PHP statements if a certain condition is TRUE.
- The simplest form of the conditional statement.

```
if (<something is TRUE>) {
      //execute this code
      // otherwise skip this code
}
// This code will execute in either case
```

Controlling the Flow (continued)

- To test a condition and perform some action whether the condition is TRUE or FALSE use the if .. else.

```
if (<something is TRUE>) {
    //execute this code
    // otherwise skip this code
} else {
   // execute this code
}
// This code will execute in either case
```

Controlling the Flow (continued)

- To test several conditions use the if ... elseif statement.

```
if (<something is TRUE>) {
    //execute this code
    // otherwise skip this code
} elseif () {
    //execute this code
    // otherwise skip this code
} else {
  //execute this code
}
// This code will execute in either case
```

Controlling the Flow (continued)

- The if ... elseif and the switch (...) conditional statements are functionally identical

```
switch (<value>) {
    case <some possible value>:
        // execute this code
    case <some possible value>:
        // execute this code
    case <some possible value>:
        // execute this code
    case <some possible value>:
        // execute this code
    default:
       // execute this code if none of the comparisons yielded
       // a positive result

}
// This code will execute in either case
```

- Choosing the switch statement over the if ... elseif statement is a matther of style and readability.

Controlling the Flow

Demonstration

Repeating Code

- A loop allows us to repeat a collection of code statements some number of times referred to as iterations.
- PHP provides three looping constructs.
  - *while*(< *someconditionisTRUE* >)
    - Requires a condition that will change when we want the loop to terminate
    - We can use a value that we can control within the loop
  - *for*(< *startingvalue* >, < *endingvalue* >, < *increment* >)
    - Will execute from starting value to ending value incrementing the count variable each time through the loop by ¡increment¿ value
  - foreach

Repeating Code (continued)

- We can terminate either the current iteration or the entire loop.
  - Using break will terminate the loop completely

```
for($i=0;$i<100;$i++) {
// do something interesting
    if($i==42) {
        break;
    }
    echo "$i"; // the loop will print al
}
```

Repeating Code (continued)

- Using continue will terminate the current iteration of a loop

```
for($i=0;$i<100;$i++) {
// do something interesting
      if($i==42) {
        continue;
      }
      echo "$i";
      // the loop will print all values
      // from 0 to
      // 99 EXCEPT 42
}
```

Repeating Code

Demonstration