Lab-4
Cookies and Sessions in php

Owing to the fact that HTTP is stateless - that is, any data you have stored is forgotten about when the page has been sent to the client and the connection is closed - it took a little work to find a solution to the problem. Eventually, Netscape put a solution into their browser known as "cookies" - tiny bits of information that a web site could store on the client's machine that were sent back to the web site each time a new page was requested. Each cookie could only be read by the web site that had written it, meaning that it was a secure way to store information across pages.

Cookies earned a bad name at first because they allowed people to track how often a visitor came to their site, what they did on the site, and such, and many people believed that cookies signalled the end of privacy on the web. Urban myths popped up in many places saying that cookies can read any information from your hard drive, and people were encouraged to disable cookies across the board. The reality is, of course, that cookies are relatively harmless, and are now commonly accepted.

Sessions grew up from cookies as a way of storing data on the server side, because the inherent problem of storing anything sensitive on clients' machines is that they are able to tamper with it if they wish. In order to set up a unique identifier on the client, sessions still use a small cookie - this cookie simply holds a value that uniquely identifies the client to the server, and corresponds to a data file on the server.

Cookies vs. Sessions

Both cookies and sessions are available to you as a PHP developer, and both accomplish much the same task of storing data across pages on your site. However, there are differences between the two that will make each favourable in their own circumstance.

Cookies can be set to a long lifespan, which means that data stored in a cookie can be stored for months if not years. Cookies, having their data stored on the client, work smoothly when you have a cluster of web servers, whereas sessions are stored on the server, meaning in one of your web servers handles the first request, and the other web servers in your cluster will not have the stored information.

Sessions are stored on the server, which means clients do not have access to the information you store about them - this is particularly important if you store shopping baskets or other information you do not want you visitors to be able to edit by hand by hacking their cookies. Session data, being stored on your server, does not need to be transmitted with each page; clients just need to send an ID and the data is loaded from the local file. Finally, sessions can be any size you want because they are held on your server; whereas many web browsers have a limit on how big cookies can be to stop rogue web sites chewing up gigabytes of data with meaningless cookie information.

So, as you can see, each has their own advantages, but at the end of the day it usually comes down one choice: do you want your data to work when you visitor comes back the next day? If so, then your only choice is cookies - if you have any particularly sensitive information, your best bet is to store it in a database, then use the cookie to store an ID number to reference the data. If you do not need semi-permanent data, then sessions are generally preferred, as they are a little easier to use, do not require their data to be sent in entirety with each page, and are also cleaned up as soon as your visitor closes their web browser.

Cookies

A cookie, as already mentioned, is a tiny little file on your client's hard drive which contains data you have asked to be stored. Some clients specifically configure their browser to reject cookies, believing for one reason or another that they are malicious, and there is nothing you can do about this - that person's browser will not be able to store your data. When creating cookies, you specify how long you want it to be valid for, and, once done, the cookie remains in place until that date, when it "expires".

Author's Note: Are cookies dangerous? No, not at all - a web-site can only read data it stored, and it can only store a small amount of data. The only possible danger to cookies is that they can store information about you without you realising it - a web-site can track how often you visit, what times

you visit at, what banners you clicked, etc. However, they cannot read your credit card number, raid your fridge, or anything of the sort!

Cookies are automatically sent to the web server (and received/parsed by PHP) each time a user visits you. That means that once we place our cookie, our visitors' browsers will automatically send the contents of that cookie across to us each time they view our message board index, and PHP will read the value into the $_COOKIE super global array. As cookies are sent each time, it is incredibly important not to store too much information there - they can really waste a lot of bandwidth.

The nice thing about cookies is that they are decentralised - you do not need to worry about creating databases to hold information or adding and removing rows, you just store the data and check whether it is set. As such, cookies are good for any pages where you have got a small amount of information to handle - usually this involves user preferences. For example, use cookies to store how users want their message board index sorting, what order they like their news printed, etc.

If you are storing information such as their email address, ICQ number, etc, you should probably use a database - data like that is generally stored for long periods of time, whereas cookies are usually more throwaway information. That said, if you are storing personal information in cookies, please take the time to encrypt it.

Sessions

Sessions are a combination of a server-side cookie and a client-side cookie, with the client-side cookie containing nothing other than a reference to the correct data on the server. Thus, when the user visits the site, their browser sends the reference code to the server, which loads the corresponding data.

This may seem a bit clumsier than just having a client-side cookie with all your data in, but there are a few advantages:

- Your server-side cookie can contain very large amounts of data with no hassle - client-side cookies are limited in size
- Your client-side cookie contains nothing other than a small reference code - as this cookie is passed each time someone visits a page on your site, you are saving a lot of bandwidth by not transferring large client-side cookies around
- Session data is much more secure - only you are able to manipulate it, as opposed to client-side cookies which are editable by all

It is also important to note that sessions only last till the user closes their browser, whereas cookies can be configured to last longer. However, other than the above, there is not much difference between session data and cookie data for most purposes.

Choosing the appropriate option

In its default implementation (as above), you should not use sessions in very large projects which are likely to be deployed on multiple load-balancing servers. The reason behind this may not be apparent at first, but if you recall, session data is actually stored in a file on a server's computer. If a user visits your site, and your load-balancer redirects them to one server for a request, that server has their session data. Next time they read a page, your load-balancer may well shift them to another server, leaving their original session data on the original server. The second server might also save some session data of its own.

The end result? They lose their information part-way through their visit, and what data you ***do*** have is fragmented across your servers. There are ways around this problem, never fear:

- Use a networked file system (NFS on Unix or the equivalent). Pretty much all operating systems allow you to connect to other computers and read/write their data. If you have a shared session data source, you would be able to bypass the above problem
- Write your own session implementation that stores data in a medium you can handle and share between all computers. This is tricky, and error-prone.
- Use a database to store your sessions.

The rumours are true: HTTP is a stateless protocol. This description recognizes the lack of association between any two HTTP requests. Because the protocol does not provide any method that the client can use to identify itself, the server cannot distinguish between clients.

While the stateless nature of HTTP has some important benefits—after all, maintaining state requires some overhead—it presents a unique challenge to developers who need to create stateful web applications. With no way to identify the client, it is impossible to determine whether the user is already logged in, has items in a shopping cart, or needs to register.

An elegant solution to this problem, originally conceived by Netscape, is a state management mechanism called cookies. Cookies are an extension of the HTTP protocol. More precisely, they consist of two HTTP headers: theSet-Cookieresponse header and theSet-Cookierequest header.

When a client sends a request for a particular URL, the server can opt to include aSet-Cookieheader in the response. This is a request for the client to include a corresponding Cookie header in its future requests. Figure 4-1 illustrates this basic exchange.

If you use this concept to allow a unique identifier to be included in each request (in a Cookie header), you can begin to uniquely identify clients and associate their requests together. This is all that is required for state, and this is the primary use of the mechanism.

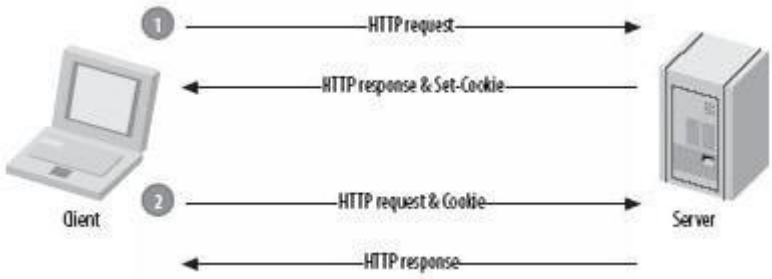

Figure 4-1. A complete cookie exchange that involves two HTTP transactions

The concept of session management builds upon the ability to maintain state by maintaining data associated with each unique client. This data is kept in a session data store, and it is updated on each request. Because the unique identifier specifies a particular record in the session data store, it's most often called the session identifier.

---

setcookie() function.

This function in PHP Language is used to work with HTTP cookies. setcookie() defines a cookie to be sent along with the rest of the HTTP headers. On successful it will return TRUE. But this does not mean that client browser has accepted cookie.

setcookie() syntax.

bool setcookie (string name [, string value [, int expire [, string path [, string domain [, int secure]]]]])

name: This argument sets the name of the cookie.

for example *setcookie('mycookie', ...)* will set mycookie and is called $_COOKIE['mycookie'] at server side.

value: This will set the value of the cookie. Since this values is stored on the client browser extra care must be taken that it does not store some secure information e.g passwords. The values is accessed by $_COOKIE['mycookie'] at the web server.

expire: Sets the expire time of cookie. It is Unix timestamp so generally it is used with time() function. For example time()+60*30. This will set the cookie to expire in 30 minutes. If not set the cookie is not persistent and will expire when the browser closes.
path: The path of cookies are used to organise cookies based on the path at web server. If set to '/' this cookie is availabe to all directories. If set to '/dir1/' this cookie is available to dir1 only and all sub directories of /dir1 i.e /dir1/sub1. Note that the default value is the current directory so if the current directory is '/dir1/' and you want to set it for all directories it must be '/'
domain: This argument will decide in which domain cookie is accessible. Value 'www.mydomin.com' makes it accessible to www mydomain only. To make it accessible to all sub domains of mydomin.com a value '.mydomin.com' must be set.
secure: Value 1 indicates the cookie must be used on secure (https) connection. Default value is 0.

setcookie() Examples.

setcookie('mycookie', 'Test mycookie'); This will set 'mycookie' with value 'Test mycookie' will expire when browser closes.

*setcookie('mycookie', 'Test mycookie', time()+3600*24);* This will expire in 1 day.

*setcookie('mycookie', 'Test mycookie', time()+3600*24, "/dir1/");*Available to /dir1 directory and all subdirectories under it.

Accessing cookie values at server.

At server in PHP script, cookies sent from the client browser will be turned into PHP variables. After PHP 4.1.0 the global array variable $_COOKIE is set for cookies from the client. $HTTP_COOKIE_VARS is also present which is availabe before PHP 4.1.0. See example below.

echo $_COOKIE["mycookie"]; This will output "Test mycookie" in our example.

Testing cookie.

On successful return of setcookie() does not mean that client browser has accepted the cookie or cookie is set successfully. It must be checked on next loading of the page if a cookie was successfully set or not. This can be done by using PHP function print_r($_COOKIE) function. This will show weather the cookie is set or not.

Deleting a cookie.

Cookies can be deleted by setting its value to "" and all other parameters must be the same as they were set at the time of sending the cookie. We must ensure that the expiration date is in the past when deleting the cookie. See examples below.

setcookie ("mycookie", "", time() – 3600);
setcookie ("mycookie", "", time() – 3600, "/dir1/");

Multiple cookies.

Multiple cookies can be set using following.

setcookie('mycookie1', 'Test mycookie1');
setcookie('mycookie2', 'Test mycookie2');
setcookie('mycookie3', 'Test mycookie3');
setcookie('mycookie4', 'Test mycookie4');

Set Session

```
<?php

echo "<b>". "Hello" . "</b>";
session_start();
// Use $HTTP_SESSION_VARS with PHP 4.0.6 or less
if (!isset($_SESSION['count'])) {
   $_SESSION['count'] = 0;
} else {
   $_SESSION['count']++;
}
echo $_SESSION['count'] . '<br>';
echo "Session Name: " . session_name() . '<br>';
echo "Session ID: " . session_id() . '<br>';
echo "Session Module Name: " . session_module_name() . '<br>';
echo "Session Save Path: " . session_save_path() . '<br>';
echo "Encoded Session: " . session_encode() . '<br>' . '<br>';
?>
```

Unset Session

```
<?php
session_start();
// Use $HTTP_SESSION_VARS with PHP 4.0.6 or less
unset($_SESSION['count']);
echo $_SESSION['count'];
?>
```