

CT216 Software Engineering Tutorial

Eoin O Fiachain

18th/25th November 2004

1 Advanced HTTP and PHP Concepts

1.1 Web Proxy Server

Many computer networks employ a *web proxy server*. This is an intermediate server that sits between web browsers (clients) and World Wide Web servers.

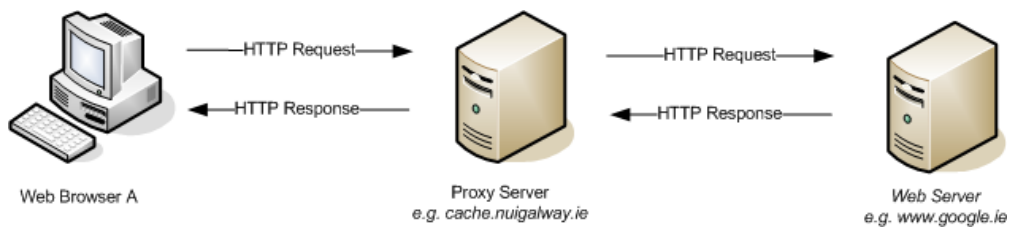


Figure 1: Operation of a web proxy server

Each time the client requests a document, the request is first sent to the proxy server. The proxy server then sends HTTP packets to the web server. When the HTTP response is returned, it is sent to proxy server, which in turn sends it back to the client.

The advantages of a proxy server include:

- **Enhanced performance by caching**

The web proxy can make temporary copies of frequently accessed web-pages and files. This can save network bandwidth and processing by preventing multiple downloads of the same files. This is called *caching*.

- **Restricted access to the WWW**

The web proxy can selectively choose which web sites a user can access or not. This is known as *filtering* requests.

- **Monitoring of WWW access**

A log file can be kept of web-sites that users access.

Web proxy servers are employed in NUI, Galway. Computer Services use a proxy server on `cache.nuigalway.ie` for their college-wide network. The IT Department network uses a proxy server on `cache.it.nuigalway.ie` for their internal IT Department network.

1.2 Cookies

1.2.1 Motivation for Cookies

A web server can handle HTTP requests and responses from potentially thousands of different clients. These requests could originate from any host on the entire Internet, situated in locations spanning the globe.

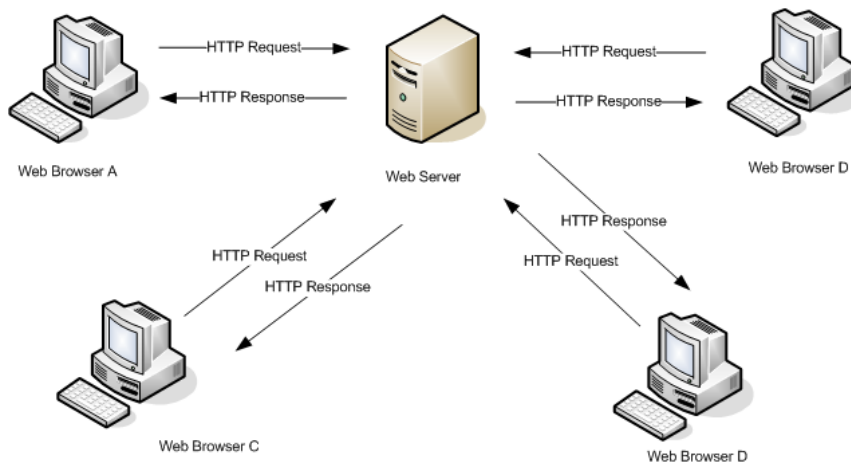


Figure 2: Web Server responding to requests from multiple clients

When the web server processes a request, it has no knowledge of any previous requests that came from the same user. In other words, all HTTP requests are independent of each other. Because of this, we call HTTP a *stateless* protocol.

While this approach works well for basic web-sites which don't need to distinguish between the source of requests, a major problem arises when we need to store separate 'personal' data for each user of the system. This separate data for each user/client is called *state* data.

Although we could attempt to distinguish users from each other based upon the IP address of the PC that sends the HTTP request, this will not always work.

Some computers (e.g. UNIX servers) may have many different users logged in simultaneously each making their own requests. All these requests would each have the same source IP address.

If a proxy server is employed in the company, all requests originating from that company would appear to originate from the proxy server's IP address. By relying on IP address alone, it would be impossible to tell each separate web surfer apart.

Accordingly we need to enhance the HTTP protocol in some way so as to allow for us to store *state data* for each user.

1.2.2 Cookies

A cookie is a small file which is stored by the web browser on a client computer. The file contains *state data* for this particular client for accessing a particular web server.

Each time the web browser sends a request to a particular web server, the corresponding cookie is sent as part of the HTTP request to the server. This way each client can present different information to the web server when requesting the same file.

A cookie is initially created through a special *Set-Cookie* header directive in the HTTP response from a web server. This is usually generated through a server-side script.

Each cookie can have a number of characteristics:

- **name**
A unique name identifying the cookie for a particular host
- **expires**
A date string which defines a date and time after which a cookie is no

longer valid. The web browser will delete the cookie once this date is reached.

If unspecified, the cookie will expire as soon as the web browser application is next closed. This is useful for temporarily storing sensitive data like passwords, so that they aren't stored on a computer's hard-disk for long periods of time.

- **domain**

A string defining what web servers to send this cookie to. If the latter part of a fully qualified host name matches the domain value, then the cookie will be sent. If they don't match then a cookie will not be sent.

This prevents other web-servers from reading the *state data* associated with a particular web server.

By default, *domain* is set to the host name of the server which sets the cookie.

e.g. If the cookie's *domain* value is `it.nuigalway.ie` and the web browser requests a page from the server `walrus.it.nuigalway.ie`. As the latter part of server's fully-qualified host name matches the *domain* value, the cookie will be sent to the server as part of the HTTP request.

- **path**

A string providing for a further matching against the path name in a URL. If the *domain* value matches, a further comparison then takes place with the *path* value.

By default, *path* is set to `'/'` which matches all paths.

e.g. Assume the path value of a cookie is `/members/` and the domain values match. If the web browser is accessing `/nonmembers/somefile.html` then the cookie will not be sent. If the web browser is accessing `/members/somefile.html` the cookie will be sent.

- **secure**

Indicates whether to only send cookies if a connection to the web-server is made using a secure encrypted protocol such as https.

Cookies also have data associated with them. Each cookie can typically hold up to 4kb of data. A server can set up to 20 cookies on a particular client.

We term a cookie that expires as soon as we close our web browser as *non-persistent*. A cookie that expires at some later fixed date is termed *persistent*.

We call cookies a means of providing for *client-side state* as they provide a means for storing state data on the web client.

1.2.3 Cookies in PHP

The `setcookie()` function can be used to create a cookie in PHP, as in the following example:

```
<?php
$value = 'something from somewhere';

/* Non-persistent cookie */
setcookie("TestCookie", $value);

/* Persistent cookie that expires in 1 hour */
setcookie("TestCookie2", $value, time()+3600);

/* Cookie with domain and path constraints */
setcookie("TestCookie3", $value, time()+3600,
    "~/ct216/", ".example.com", 1);
?>
```

Note that this function must be called before any output is generated from the script e.g. calls to `echo` or `print`

The super-global array `$_COOKIE` can be used to retrieve the cookie data passed to a particular page, in a similar manner to `$_GET` and `$_POST`:

```
echo $_COOKIE['TestCookie'];
```

1.3 PHP Sessions

PHP sessions are a form of *server-side state*. Like cookies, sessions provide a means for storing state data. However, instead of storing the data on the web client, sessions store the state data on the web server instead.

Sessions allow for server-side data objects to be preserved across subsequent accesses by the same user.

PHP sessions normally function by assigning each user a unique ID number called a *session-id*. This *session-id* number is stored in a cookie for each user. However, no other session data is stored in the cookie. The *session-id* is used to reference session data stored on the server.

1.3.1 Sessions vs. Cookies

The advantages of using sessions instead of cookies are:

- Data objects can be automatically stored with sessions, whereas they must be converted explicitly to and from a string format when using cookies.
- Unlike cookies, there are no size limitations on the amount of data stored.
- HTTP traffic may be reduced. This is because superfluous cookie data is not sent with each HTTP request to the server. Instead, all data (apart from the session-id) is stored on the server.
- The server has greater control over the state data. With cookies a user could potentially change any part of the cookie data stored on his system. With sessions, session data can only be changed in a manner enabled by server-side programs.
- When cookies are disabled by a user, sessions can still operate in certain circumstances using alternative techniques like URL rewriting.

The disadvantages of using sessions are:

- As data is stored on the web server, web server disk-space may be reduced.
- When using multiple web-servers collectively in what is known as a *web server clustering*, session data needs to be synchronised or managed across all web servers.

1.3.2 Accessing sessions in PHP

Each script which uses sessions must call the *session_start* function before any output occurs. This either resumes an existing session or creates a new session if no session already exists.

After that the super-global array `$_SESSION` can be used to access session variables. Each variable is identified by the string identifier in the array.

The following example assigns values to a number of session variables:

```
<?php
    session_start();

    echo 'Welcome to page #1';

    $_SESSION['favcolor'] = 'green';
    $_SESSION['animal']   = 'cat';
    $_SESSION['time']     = time();
?>
```

The following script reads from a number of existing session variables:

```
<?php
    session_start();

    echo 'Welcome to page #2<br />';

    echo $_SESSION['favcolor']; // green
    echo $_SESSION['animal']; // cat
    echo date('Y m d H:i:s', $_SESSION['time']);
?>
```

1.3.3 User Authorisation using Session Variables

Many web sites involve multiple users, who need to login to the web site to access particular content. Different users are required to have access to different elements of the web site.

Session-variables are often used as a means of implementing the security on these web pages. A session variable determines whether a user can access a

particular page or not. This session variable can only ever be altered when a user logs-in or logs-out of the system.

At the beginning of each page, a function is called which checks the value of this session variable and determines whether to allow the script to continue executing or not.

2 Internationalisation and Localisation

Many applications and web-sites need to be accessible in many different languages. Furthermore, cultural preferences such date styles (dd/mm/yy or mm/dd/yyyy etc.) and currency types and symbols also vary across the world.

A *locale* is a group of settings that describe text formatting and language customs in a particular area in the world.

Internationalisation is the the process of taking a program designed for one locale and redesigning it so that it can be used in many different locales, without needing to change source code.

Localisation is the process of adding a particular locale or locales to an internationalised program.

2.1 Locale Names

Locates are identified by *locale names*. These are typically composed of a number of parts:

- A two-letter code identifying a specific language e.g. *en* for English, *fr* etc.for French
- Optionally, an underscore character followed by a two-letter country code e.g. *IE* for Ireland, *GB* for Great Britain etc.
- Optionally, a full stop character followed by a character-set specifier e.g. ISO-8859-1, Big5 etc.

Character sets vary across the world. For example, the Chinese alphabet is very different to the English alphabet. On a smaller scale, the Irish language has extra characters for *fadas* compared to the English language.

Many defined character sets exist. The most commonly used character set is ASCII which has 128 characters which can express most European languages.

Unicode is a more modern standard which aims to represent the characters needed to implement almost all the languages in the world. It can have up to 65,536 characters.

Some example locale names would be:

en	English
nl	Dutch
en_GB	English in Great Britain
en_IE	English in Ireland
en_US	English in the US
zh_TW.Big5	Taiwanese Chinese using the Big5 character set

2.2 Internationalisation for CT216 Software Engineering Project

With many applications, localisation takes place at *compile-time*. Different versions of applications are compiled for different languages.

With other applications, localisation must take place at *run-time* where a user opts for a particular language. The CT216 Software Engineering Project is an example of the latter.

Internationalisation may involve providing a variety of different types and styles of data, including data formatting, currency formatting, currency symbols, images etc.

We will focus primarily on internationalising text data for the CT216 Software Engineering Project. It may also be necessary to internationalise particular images if those images contain language-specific text.

In a typical web application there are two types of text data that may need to be internationalised:

- **Design-time strings**

These are the strings which would typically be found in source code in applications (or in resource files). They are defined by the designer / programmer of the system.

- **Run-time strings**

These are strings which are added to the web application by the users

as the system runs. These are considerably more difficult to handle, as the system will need to provide a means of accepting translated strings for all supported languages at run-time.

In the CT216 Software Engineering project we will focus on internationalising design-time strings. Strings inputted by users will not need to be translated. We will localise the application for two languages - English and French.

2.3 Internationalisation with PHP

There are a number of approaches which can be adopted for localisation each of which has advantages and disadvantages:

- Store strings in separate **user-defined files** to the PHP program. These strings are loaded by the application and read from memory when necessary.
- Store strings in an **external database**. These strings are loaded by the application and read from memory when necessary.
- Use the **GNU gettext** library together with the PHP *gettext* module to access specially-designed string files. *GNU gettext* is a set of utilities that are specifically designed to assist with the internationalisation of software.

We will focus on the latter *gettext* technique for the CT216 Software Engineering project.

Although it is possible to read strings directly from the database or from a user-defined file on each occasion that they are used, this isn't very efficient and doesn't work very well for large web-sites. It is much more efficient to read strings from memory.