Cache types :

* [APC](http://php.net/manual/en/book.apc.php): that tool have some caching feature, plus PHP precompilation
* [memcached](http://memcached.org/): a distibuted caching system
* [eAccelator](http://eaccelerator.net/): pre compilation
* [xcache](http://xcache.lighttpd.net/): pre compilation

**PHP Caching to Speed up Dynamically Generated Sites**

Related Articles

* [PHP](http://theukwebdesigncompany.com/articles/articles.php?subject=11)

This entire site, like many, is built in PHP. PHP provides the power to simply 'pull' content from an external source, in the case of my site this is flat files but it could just as easily be an MySQL database or an XML file etc..

The downside to this is processing time, each request for one page can trigger multiple database queries, processing of the output, and formatting it for display... This can be quite slow on complex sites (or slower servers)

Ironically, these so-called 'dynamic' sites probably have very little changing content, this page will almost never be updated after the day it is written - yet each time someone requests it the scripts goes and fetches the content, applies various functions and filters to it, then outputs it to you...

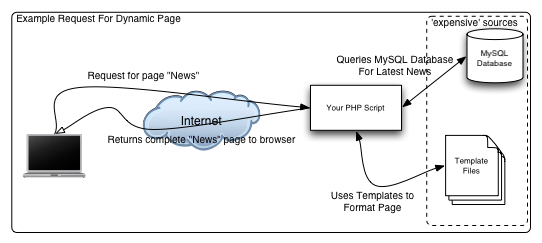
**Enter Caching**

This is where **caching** can help us out, instead of regenerating the page every time, the scripts running this site generate it the first time they're asked to, then store a copy of what they send back to your browser. The next time a visitor requests the same page, the script will know it'd already generated one recently, and simply send that to the browser without all the hassle of re-running database queries or searches.

**An Illustration**

This example shows a request for a "News" page on a website, the News changes daily so it makes sense to have it in a database rather than as a static file so it can be easily updated and searched, The News page is a PHP script which does the following;

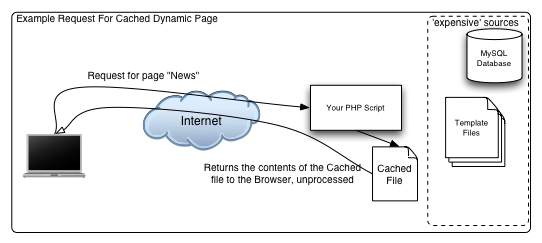
* Connect to an MySQL Database
* Request 5 most recent news items
* Sort news items from most recent to oldest
* Read a template file and substitute variables for content
* Output the finished page to the user



This takes a considerable amount of time, it's negligable if you get one or two visitors an hour, but if you get 500 visitors an hour it makes a *big* difference.

Consider the difference between this, and a straight forward request for a normal .html file. The web server doesnt have to do any hard work to serve up a .html file, it just finds the file and dumps it's contents to the browser... using caching allows you to experience this speed gain even with dynamic sites.

Continuing the same example, but where caching is in place, the first user to request the News page would cause the script to do exactly as above, and in addition actually*increase* the load by making it write the result to a file, as well as to the browser. However, subsequent requests would work something like this:



As you can see, the MySQL database and Templates aren't touched, the web server just sends back the contents of a plain .html file to the browser. The request is completed in a fraction of the time, the user gets their page faster, and your server has less load on it - everyone's happy.

**Implementing a Cache in PHP**

There are various ways of implementing a cache to do this, but the easiest to implement (if maybe not the most efficient) is to use a bit of extra PHP code in your scripts. Most of this example is based on this site, but could easily be applied to any site.

For the purposes of this example it helps to have a small understanding of my website. Basically each page location (e.g. "site/caching") has each / replaced by a . and that file (which contains all the content) is included into the template (so includes/design.caching in this case). The actual filename ends up in a variable called $reqfilename.

**The Output Buffer**

The Output Buffer, introduced in recent versions of PHP, is ideal for this. Basically if you call ob\_start() at the start of your program, it supresses all output until you specifically flush the output buffer. You can therefore easily get at the output of any PHP script.

**A Simple Cache**

Lets look at the most basic, and rather useless, cache. This little snippet of code will save the output of a call for the "home" page into a file called home.html

<?php  
// start the output buffer  
ob\_start(); ?>  
  
//Your usual PHP script and HTML here ...   
  
<?php  
$cachefile = "cache/home.html";  
// open the cache file "cache/home.html" for writing  
$fp = fopen($cachefile, 'w');   
// save the contents of output buffer to the file  
fwrite($fp, ob\_get\_contents());   
// close the file  
fclose($fp);   
// Send the output to the browser  
ob\_end\_flush();   
?>

Not tremendously useful, because now all we have is a script that generates a file called "**cache/home.html**" each time it is ran. But it's a good basis for a cache, it saves the content generated by the PHP script to a file. If you were to visit **cache/home.html** in a web browser you would see exactly the same page as if you visited the script the generated it, but that's no use unless the user knows where to look for it.

**Using the cache files**

Now we have our code to generate a cache file, we need to find a way of using these files constructively. There are two types of request a '**MISS**' and a '**HIT**'.

If a user requests a page that has not been requested before, or that was requested long enough ago that it might be out of date, that is considered a MISS, in this situation the script should regenerate the page from it's database (or whatever) sources, and save a new cache file.

If a user requests a page that has been requested recently, and is in the cache, the script just needs to pass that file to the user and doesnt need to do anything else. This is known as a HIT.

Checking to see if a page has already been cached is easy:

<?php

$cachefile = "cache/home.html";

if (file\_exists($cachefile)) {

// the page has been cached from an earlier request

// output the contents of the cache file

include($cachefile);

// exit the script, so that the rest isnt executed

exit;

}

?>

Placing that code at the start of your script will cause it to use the cached file if it exists, and then exit from the script (so the rest of it will never run). If you have a site that never changes then that's enough, but very few sites never change. The other time when this snippet along would be enough is if you had a site that only changed every day or so, then you could use cron to empty the cache directory each day. This wouldn't be suitable for many sites, we need a way of expiring content in the cache so that it isnt use idefinitely.

**Expiring Cache Data**

There are numerous ways to check if a cache file should be updated, we will look at the two most common here;

**Simple Time Expiry**

This is probably the best option for most sites, you give the cache files a life e.g. 5mins, 20mins, 1hour after which they will expire and the page be regenerated. The following example shows how this would work and when changes would be visible to the user if a 2 hour expiry time was used; The first visit of the day was at 12:00, there was no valid cache so the page was generated, this is valid until 1400. So although the database (and therefore the content of the generated page) was updated at 1320, any requests recieved between then and 1400, when the cache expires would contain the out of date information. The next request at 1400 will finally call on the database sources again, and the user will see the information added at 1320.

The database is then updated again at 1500, but these changes wont be visible until after 1600, one hour after they were made.

While this approach is suitable for most sites, it's obviously not appropriate for up-to-the-minute news sites, or sites with regularly changing content

To implement this we simply have to expand the:if (file\_exists($cachefile))statement above to include a check of the cache file's modification time:

<?php

// 5 minutes

$cachetime = 5 \* 60;

// Serve from the cache if it is younger than $cachetime

if (file\_exists($cachefile) &&

(time() - $cachetime < filemtime($cachefile)))

{

include($cachefile);

echo "<!-- From cache generated ".date('H:i',

filemtime($cachefile))."

-->n";

exit;

}

?>

Putting this together with the previous code we get a basic structure that will cache the output of a page for 5 minutes:

<?php

$cachefile = "cache/".$reqfilename.".html";

$cachetime = 5 \* 60; // 5 minutes

// Serve from the cache if it is younger than $cachetime

if (file\_exists($cachefile) && (time() - $cachetime

< filemtime($cachefile)))

{

include($cachefile);

echo "<!-- Cached ".date('jS F Y H:i', filemtime($cachefile))."

-->n";

exit;

}

ob\_start(); // start the output buffer

?>

.. Your usual PHP script and HTML here ...

<?php  
 // open the cache file for writing  
 $fp = fopen($cachefile, 'w');

// save the contents of output buffer to the file  
 fwrite($fp, ob\_get\_contents());

// close the file

fclose($fp);

// Send the output to the browser

ob\_end\_flush();   
?>

**Regenerate only When Necessary**

An alternative method involves checking to see if the data sources have been modified, this increases the load of each request slightly, because it requires a database connection in the case of DB-based sites, or a query of the file modification time of potentially a few files, it also makes the script slightly more complicated. However, this method prevents unecessary LARGE queries, such as those required to retrieve data for inclusion in a page, and prevents regenerating pages regularly even when nothing has changed. This is the approach used on this site.

All that is involved here is changing theif()clause, for example:

<?php  
 $cachefile = "cache/".$reqfilename.".html";

// Serve from the cache if it is the same age or younger than the last

// modification time of the included file (includes/$reqfilename)

if (file\_exists($cachefile) && (filemtime("includes/".$reqfilename)

< filemtime($cachefile))) {

include($cachefile);

echo "<!-- Cached ".date('H:i', filemtime($cachefile))."

-->n";

exit;  
 }

// start the output buffer  
 ob\_start();   
?>

.. Your usual PHP script and HTML here ...

<?php  
 // open the cache file for writing

$fp = fopen($cachefile, 'w');

// save the contents of output buffer to the file  
 fwrite($fp, ob\_get\_contents());

// close the file  
 fclose($fp);

// Send the output to the browser  
 ob\_end\_flush();  
?>

This could be easily adapted to query a database containing a column for 'datemodified' or something similar.

**Where not to use Caching**

Caching should not be used for some things, the most obvious being search results, forums etc... where the content has to be up-to-the-minute and changes depending on user's input. It's also advisable to avoid using this method for things like a "Latest News" page, in general dont use it on any page that you wouldn't want the end users browser or proxy to cache.

This page modified: 27th March 2004.

**Copyright 2004 Ross S. McKillop**

*Publication Date:* Friday 26th March, 2004  
*Author:* Ross Mckillop [View profile](http://theukwebdesigncompany.com/authors/author-profile.php?author=95)

<?php  
  
ob\_start();  
  
echo "Hello ";  
  
$out1 = ob\_get\_contents();  
  
echo "World";  
  
$out2 = ob\_get\_contents();  
  
ob\_end\_clean();  
  
var\_dump($out1, $out2);

//output

// string 'Hello ' *(length=6)*

// string 'Hello World' *(length=11)*

?>