Bs_FileCache:

cache and cash can be synonyms

Sam Blume
Andrej Arn

BlueShoes

SOMETIMES IT IS NOT NEEDED TO PROCESS THE SAME DATA AGAIN AND AGAIN, SIMPLY BECAUSE THERE WAS NO CHANGE, AND THUS THERE WON'T BE A DIFFERENT RESULT.
THIS IS WHERE A CACHE COMES IN HANDY.

Typical Use Cases:

- You have just crunched data typically from a file using a lot of CPU and want to persist the crunched data for later use.
  Also you want to be notified if the persisted data has become 'out of date' because the 'origin-file' has changed AND/OR the lifetime you have set has passed. – (This is also referred as “Hot Update”).
- You have some data that needs to be recalculated if it’s ‘too old’. (No source file is involved that needs to be checked for changes).

Source Location:
core/file/Bs_FileCache.class.php
core/file/examples/fileCache1.php
core/file/examples/fileCache2.php
Note that there is the specialiced, subclassed core/net/http/Bs_UrlCache.class.php that works great as web cache for full pages.

Features :

- Checks if 'origin-file' has **changed** AND/OR the **lifetime** you have set has passed (cache then becomes 'out of date').
- Cache versioning to identify outdated/invalid cache data. All cache with older versions then $cacheVersion are considered 'out of date'. Very useful during development and updates, where the structure of the data you generate has changed and all cache data becomes invalid. Saves you the hassle of deleting all old cache files spread all over the disk.
- Exclusive cache writing to disk.
- Second level memory buffer that holds the cache data in memory for reuse (Currently only until script ends).
- setBufferSize(): Ability to set the memory buffer size in **bytes** OR in **%** of the available script memory.
- setDir(): Define where you want the cache-files to be stored. 2 options
    - In one central dir (tip: use a RAM-disk)
    - Under each 'origin-file' in a subdir.
- setVerboseCacheNames(): Define if the 'cache-files' should contain a verbose name or not. A verbose name contains the name of the 'origin-file' (max first 100 chars).

Example fileCache1.php:

```php
<?php
# basic example:
# imagine that we have a source file, in our case /tmp/toCrunchData.txt
# based on that file we create something. and the output will be cached.
# whenever the source file changes, we have to drop the cache and
# "crunch" our data again.

#include dependencies
require_once($_SERVER["DOCUMENT_ROOT"]    . "../global.conf.php");
include_once($APP['path']['core'] . 'file/Bs_FileCache.class.php');

# 1) create the instance
$cacher =& new Bs_FileCache();

# 2) Try to get the cached data using the path of the source file as key.
$data = $cacher->fetch('/tmp/toCrunchData.txt');

# 3) Test the return value
if ($data === FALSE) {       // ERROR
  echo "ERROR: " . $cacher->getLastError();
} elseif ($data === NULL){   // cache is "Out Of Date", or not cached yet.
  // Crunch the data (Replace with *YOUR* crunching function.)
  $crunchedData = myCruncher('/tmp/toCrunchData.txt');

  # 4) Cache the data as serialized data
  $cacher->store('/tmp/toCrunchData.txt', serialize($crunchedData));
} else {                     // SUCCESS
  $crunchedData = unserialize($data);
}

# 5) do something with $crunchedData
dump($crunchedData);


# this is your cruncher function. of course you would do more
# than just reading the data.
function myCruncher($fullPath) {
    return join('', file($fullPath));
}
?>
```

Example fileCache2.php:

```php
<?php
# example 2:
# Use with some property settings.
# This time there is no 'origin-file' file and we use a 'lifetime'
# for the cache.


#include dependencies
require_once($_SERVER["DOCUMENT_ROOT"]    . "../global.conf.php");
include_once($APP['path']['core'] . 'file/Bs_FileCache.class.php');

# 1) create the instance and set properties
$cacher =& new Bs_FileCache();
$cacher->setDir('r:/');               // tip: Use a RAM-Disk!
$cacher->setBufferSize('200k');       // % of memory is also possible
$cacher->setVerboseCacheNames(TRUE);  // Give cache-files a readable name

# 2) Try to get the cached data using any key *you* define.
$MY_CACHE_KEY = 'chacheKey_1';
$data = $cacher->fetch($MY_CACHE_KEY);

# 3) Test the return value
if ($data === FALSE) { // ERROR
  echo "ERROR: " . $cacher->getLastError();
} elseif ($data === NULL){ // cache is "Out Of Date", or not cached yet.
  // Crunch the data (Replaced with *YOUR* crunching function.)
  $crunchedData = myCruncher('/tmp/toCrunchData.txt');
  # 4) Cache the data serialized with a timeout of 600s
    #    and disabled 'origin-file' check
  $cacher->store($MY_CACHE_KEY, serialize($crunchedData), $originCheck=FALSE,
600);
} else { // SUCCESS
  $crunchedData = unserialize($data);
}

# 5) do something with $crunchedData
dump($crunchedData);


# this is your cruncher function. of course you would do more
# than just reading the data.
function myCruncher($fullPath) {
    return join('', file($fullPath));
}
?>
```