

Zend Framework Page Caching: Part 3: Tagging For Static File Caches

Tracking What's Being Cached

Expiring multiple caches linked to a specific change is easy to accomplish using Tagging, where we tag caches with keywords and clean caches based on those keywords. Unfortunately, static files can't be tagged in the normal way since their filenames must be constant. To deliver a similar system static file caching, we need to tag caches outside of the cache filename/contents itself which requires a Model or similar storage backend to keep track of tags and the caches to which they relate.

In a sense, we're creating a cache within a cache. Although this is the simplest approach for small numbers of tags and URLs, a more robust system would be backed by a database to allow a greater degree of flexibility in minimising the size of the overall resultset needed to be loaded for any given page request.

For now, the form of the inner cache will be a simple array where for each Tag, we assign a list of tagged Request URIs. This array can then be cached either to a file or a memory slot in, for example, APC for retrieval in future requests. Its obvious flaw, without a database, is that it will be loaded in full for every request where the relevant Page Cache is utilised. We are applying some lazy loading however, but when it gets big enough the move to a database may be needed.

Let's start by adapting ZFExt_Controller_Action_Helper_Cache to host an array of tags which is what we will cache and retrieve from the inner cache when any Tag operations are utilised. We'll also make it possible to set tags when setting what Actions need to be cached. This will result in a system where tags are assigned for any cache we want, and those tags will be saved to the inner cache when the Action completes (using postDispatch()). We'll also add a new removeTaggedPageCache() method which we can either call directly from an Action, or from a Cleaner class.

<?php

```
class ZFExt_Controller_Action_Helper_Cache extends Zend_Controller_Action_Helper_Abstract
{
    protected $_caches = array();
    protected $_cleaners = array();
    protected $_caching = array();
    protected $_obstarted = false;
    protected $_tags = array();
    protected $_tagged = array();
    public function addCache($cacheld, $cache) {
        if (!$cache instanceof Zend_Cache_Core && !$cache instanceof
ZFExt_Cache_Backend_Static_Adapter) {
            throw new Exception('Need to provide a valid cache!');
        }
        $this->_caches[$cacheld] = $cache;
    }
    public function createCache($cacheld, $frontend, $backend, $frontendOptions = array(),
$backendOptions = array(), $customFrontendNaming = false, $customBackendNaming = false,
$autoload = false) {
```

```

    $cache = Zend_Cache::factory($frontend, $backend, $frontendOptions, $backendOptions,
$customFrontendNaming, $customBackendNaming, $autoload);
    $this->addCache($cacheld, $cache);
    return $cache;
}
public function getCache($cacheld) {
    if ($this->hasCache($cacheld)) {
        return $this->_caches[$cacheld];
    }
    return false;
}
public function hasCache($cacheld) {
    if (isset($this->_caches[$cacheld])) {
        return true;
    }
    return false;
}
// Pass array of actions to cache for the current Controller
// optionally pass array of tags to assign.
public function direct(array $actions, array $tags = array()) {
    $controller = $this->getRequest()->getControllerName();
    foreach ($actions as $action) {
        if (!isset($this->_caching[$controller])) {
            $this->_caching[$controller] = array();
        }
        if (!isset($this->_caching[$controller][$action])) {
            $this->_caching[$controller][] = $action;
        }
        if (!empty($tags)) {
            if (!isset($this->_tags[$controller])) {
                $this->_tags[$controller] = array();
            }
            if (!isset($this->_tags[$controller][$action])) {
                $this->_tags[$controller][$action] = array();
            }
            foreach ($tags as $tag) {
                if (!in_array($tag, $this->_tags[$controller][$action])) {
                    $this->_tags[$controller][$action][] = $tag;
                }
            }
        }
    }
}
// Remove page caches based on URL, with recursive matching directory
// removal for those where, for example, pagination is also being cached.
// Sec: remember what they say about "rm -R" - checks needed
public function removePageCache($relativeUrl, $recursive = false) {
    if ($recursive) {
        $this->getCache('page')->removeRecursive($relativeUrl);
    } else {
        $this->getCache('page')->remove($relativeUrl);
    }
}
}

```

```

// delete all statically cached files which are associated with the
// given array of tags
public function removeTaggedPageCache(array $tags)
{
    return $this->getCache('page')->clean(Zend_Cache::CLEANING_MODE_MATCHING_TAG, $tags);
}
// create a nested array assigning cleaners to various
// controller+action combinations
public function useCleaner($cleanerName, array $actions)
{
    foreach ($actions as $action) {
        $controller = $this->getRequest()->getControllerName();
        if (!isset($this->_cleaners[$controller])) {
            $this->_cleaners[$controller] = array();
        }
        if (!isset($this->_cleaners[$controller][$action])) {
            $this->_cleaners[$controller][$action] = array();
        }
        if (!isset($this->_caching[$controller][$action][$cleanerName])) {
            $this->_cleaners[$controller][$action][] = $cleanerName;
        }
    }
}
// Commence caching for matching Actions
// Will exit if caching has already started
public function preDispatch()
{
    $controller = $this->getRequest()->getControllerName();
    $action = $this->getRequest()->getActionName();
    if (!empty($this->_caching)) {
        if (isset($this->_caching[$controller]) &&
            in_array($action, $this->_caching[$controller])) {
            // do not start caching if started earlier in cycle
            // otherwise commence caching here
            $stats = ob_get_status(true);
            foreach ($stats as $status) {
                if ($status['name'] == 'Zend_Cache_Frontend_Page::_flush') {
                    return;
                }
            }
            $this->getCache('page')->start();
            $this->_obstarted = true;
        }
    }
}
// Run cache cleaning operations after actions are dispatched
// enforces Cleaner methods as being "after{ActionMethod}"
// Store the revised Tagged array into the inner cache.
public function postDispatch()
{
    if (!empty($this->_cleaners)) {
        $controller = $this->getRequest()->getControllerName();
        $action = $this->getRequest()->getActionName();
    }
}

```

```

    if (isset($this->_cleaners[$controller][$action])) {
        $cleanerNames = $this->_cleaners[$controller][$action];
        foreach ($cleanerNames as $cleanerName) {
            $cleaner = $this->createCleaner($cleanerName);
            $method = 'after'.ucfirst($action);
            $cleaner->{$method}();
        }
    }
}
if ($this->_obstarted) {
    $this->getCache('page')->end();
}
if (isset($this->_caching[$controller]) &&
    in_array($action, $this->_caching[$controller])) {
    $requestUri = $this->getRequest()->getRequestUri();
    $this->_tagged = $this->_loadTagged();
    if (isset($this->_tags[$controller][$action]) && !empty($this->_tags[$controller][$action])) {
        foreach ($this->_tags[$controller][$action] as $tag) {
            if (!isset($this->_tagged[$tag])) {
                $this->_tagged[$tag] = array();
            }
            if (!in_array($requestUri, $this->_tagged[$tag])) {
                $this->_tagged[$tag][] = $requestUri;
            }
        }
    }
    $this->_saveTagged();
}
}
// Cheat by stealing functionality from the Dispatcher! Haha!
// In a real class, should really implement this natively
// to keep down on dependencies, and allow cleaners to
// exist elsewhere. Also this is not Module friendly yet.
public function createCleaner($cleanerName)
{
    $dispatcher = $this->getFrontController()->getDispatcher();
    $className = $cleanerName . 'Cleaner';
    $finalClassName = $dispatcher->loadClass($className);
    $cleaner = new $finalClassName;
    return $cleaner;
}

// load cached Tags from inner cache
protected function _loadTagged()
{
    if (!$this->hasCache('tagged')) {
        throw new Zend_Exception('No "tagged" cache has been defined therefore Tagging cannot be
utilised');
    }
    if ($result = $this->getCache('tagged')->load('zfextcache_tagged')) {
        $this->_tagged = $result;
    }
}
}

```

```

// save existing Tags to inner cache
protected function _saveTagged()
{
    if (!$this->hasCache('tagged')) {
        throw new Zend_Exception('No "tagged" cache has been defined therefore Tagging cannot be
utilised');
    }
    $this->getCache('tagged')->save($this->_tagged, 'zfextcache_tagged');
}
}
}

```

You'll notice that the Helper refers to a specific cache by name, and that the Static backend (presented below) refers to the same cache. This partial duplication may be another sign that the Action Helper is taking on too much responsibility, indicating the need for a discrete Cache Management class to centralise the inner cache with.

Based on the above, we make some changes to the ZFExt_Cache_Backend_Static class to support cleaning the static cache files based on associated tags.

```
<?php
```

```
class ZFExt_Cache_Backend_Static extends Zend_Cache_Backend implements
Zend_Cache_Backend_Interface
```

```

{
    const DEBUG_HEADER = 'DEBUG HEADER : This is a cached page !';
    // Available options
    protected $_options = array(
        'public_dir' => null,
        'file_extension' => '.html',
        'index_filename' => 'index',
        'file_locking' => true,
        'cache_file_umask' => 0600,
        'debug_header' => false
    );
    protected $_innerCache = null;
    // Test if a cache is available for the given id and (if yes) return it
    // (false else)
    // $id should be the REQUEST_URI whose static file is to be deleted
    public function load($id, $doNotTestCacheValidity = false)
    {
        $id = $this->_decodeId($id);
        if (empty($id) || $id == '_') {
            $id = $this->_detectId();
        }
        if (!$this->_verifyPath($id)) {
            Zend_Cache::throwException('Invalid cache id: does not match expected public_dir path');
        }
        if ($doNotTestCacheValidity) {
            $this->_log("ZFExt_Cache_Backend_Static::load() : $doNotTestCacheValidity=true is
unsupported by the Static backend");
        }
        $fileName = basename($id);
        if (empty($fileName)) {
            $fileName = $this->_options['index_filename'];
        }
    }
}

```

```

}
$pathName = $this->_options['public_dir'] . dirname($id);
$file = $pathName . '/' . $fileName . $this->_options['file_extension'];
if (file_exists($file)) {
    $content = file_get_contents($file);
    // move debug header to Frontend to prevent these gymnastics
    return str_replace(self::DEBUG_HEADER, "", $content);
}
return false;
}
// Test if a cache is available or not
// $id should be the REQUEST_URI whose static file is to be deleted
public function test($id)
{
    $id = $this->_decodeId($id);
    if (!$this->_verifyPath($id)) {
        Zend_Cache::throwException('Invalid cache id: does not match expected public_dir path');
    }
    $fileName = basename($id);
    if (empty($fileName)) {
        $fileName = $this->_options['index_filename'];
    }
    $pathName = $this->_options['public_dir'] . dirname($id);
    $file = $pathName . '/' . $fileName . $this->_options['file_extension'];
    if (file_exists($file)) {
        return true;
    }
    return false;
}
// Save content to a static content file in /public directory
// Note: We're ignoring the ID as its not required.
public function save($data, $id, $tags = array(), $specificLifetime = false)
{
    clearstatcache();
    $requestUri = $this->_detectId();
    $fileName = basename($requestUri);
    if (empty($fileName)) {
        $fileName = $this->_options['index_filename'];
    }
    $pathName = $this->_options['public_dir'] . dirname($requestUri);
    if (!file_exists($pathName)) {
        mkdir($pathName, $this->_options['cache_file_umask'], true);
    }
    if ($id !== '_') { // empty ID since a Capture
        $dataUnserialized = unserialize($data);
    } else {
        $dataUnserialized = array();
        $dataUnserialized['data'] = $data;
    }
    if ($this->_options['debug_header']) {
        $dataUnserialized['data'] =
            self::DEBUG_HEADER . $dataUnserialized['data'];
    }
}

```

```

$file = $pathName . '/' . $fileName . $this->_options['file_extension'];
if ($this->_options['file_locking']) {
    $result = file_put_contents($file, $dataUnserialized['data'], LOCK_EX);
} else {
    $result = file_put_contents($file, $dataUnserialized['data']);
}
@chmod($file, $this->_options['cache_file_umask']);
if (count($tags) > ) {
    $this->_log(self::TAGS_UNSUPPORTED_BY_SAVE_OF_STATIC_BACKEND);
}
return (bool) $result;
}
// Remove a cache record
// $id should be the REQUEST_URI whose static file is to be deleted
public function remove($id)
{
    $id = $this->_decodeId($id);
    if (!$this->_verifyPath($id)) {
        Zend_Cache::throwException('Invalid cache id: does not match expected public_dir path');
    }
    $fileName = basename($id);
    if (empty($fileName)) {
        $fileName = $this->_options['index_filename'];
    }
    $pathName = $this->_options['public_dir'] . dirname($id);
    $file = $pathName . '/' . $fileName . $this->_options['file_extension'];
    if (!file_exists($file)) {
        return true;
    }
    return unlink($file);
}
// Remove a cache record recursively (i.e. the file AND matching directory)
// it ain't perfect - there may be no file matching the directory name
// (but you get the point I'm sure!)
// $id should be the REQUEST_URI whose static file & dir tree is to be deleted
public function removeRecursively($id)
{
    $id = $this->_decodeId($id);
    if (!$this->_verifyPath($id)) {
        Zend_Cache::throwException('Invalid cache id: does not match expected public_dir path');
    }
    $fileName = basename($id);
    if (empty($fileName)) {
        $fileName = $this->_options['index_filename'];
    }
    $pathName = $this->_options['public_dir'] . dirname($id);
    $file = $pathName . '/' . $fileName . $this->_options['file_extension'];
    $directory = $pathName . '/' . $fileName;
    if (file_exists($directory)) {
        if (!is_writable($directory)) {
            return false;
        }
    }
    foreach (new DirectoryIterator($directory) as $file) {

```



```

        break;
    }
}
public function setInnerCache(Zend_Cache_Core $cache)
{
    $this->_innerCache = $cache;
}
public function getInnerCache()
{
    if (is_null($this->_innerCache)) {
        Zend_Cache::throwException('An Inner Cache has not been set; use setInnerCache()');
    }
    return $this->_innerCache;
}
// Encoded by ZFExt_Cache_Backend_Static_Adapter
protected function _decodeId($id)
{
    // another workaround since Zend_Cache_Core prevents
    // empty or null IDs which we'll have when Capturing
    // and before the REQUEST_URI is checked
    if ($id == '_') {
        return "";
    }
    return pack('H*', $id);
}
// "Danger, Will Robinson!"
// Sanity check to ascertain whether path is within the configured
// public_dir path
protected function _verifyPath($path)
{
    $path = realpath($path);
    $base = realpath($this->_options['public_dir']);
    return strcmp($path, $base, strlen($base)) !== 0;
}
protected function _detectId()
{
    // should strip query strings in future
    // along with other fragments
    return $_SERVER['REQUEST_URI'];
}
}
}

```

We now have support for cleaning the cache using matching tags. This relies on the Action Helper and Static Backend utilising the same cache which can be created as normal and passed to both from our Bootstrap.

```

<?php
class ZFExt_Bootstrap
{
    // ...
    public function run()
    {
        $this->setupEnvironment();
    }
}

```

```

// Implement Page Caching at Bootstrap level before any
// MVC operations so these operations can be completely
// avoided when a valid cache exists
$this->usePageCache();
// If a valid cache exists, execution exits!
$this->prepare();
$response = self::$frontController->dispatch();
$this->sendResponse($response);
}
public function usePageCache()
{
    $frontend = new ZFExt_Cache_Frontend_Capture();
    $backendOptions = array(
        'debug_header' => true,
        'public_dir' => self::$root . '/public'
    );
    $backend = new ZFExt_Cache_Backend_Static($backendOptions);
    // use our Adapter to deal with the Core's private validation
    $cache = new ZFExt_Cache_Backend_Static_Adapter(
        Zend_Cache::factory($frontend, $backend)
    );
    // create an inner cache so the backend can store tags
    $taggedCache = Zend_Cache::factory('Core', 'File',
        array('automatic_serialization'=>true),
        array('cache_dir'=>self::$root . '/cache')
    );
    $backend->setInnerCache($taggedCache);
    // Add the new cache to the Cache Control Action Helper
    Zend_Controller_Action_HelperBroker::addPrefix('ZFExt_Controller_Action_Helper');
    Zend_Controller_Action_HelperBroker::getStaticHelper('Cache')->addCache('page', $cache);
    Zend_Controller_Action_HelperBroker::getStaticHelper('Cache')->addCache('tagged',
    $taggedCache);
}
}
}

```

Et voilà ! We now have a prototype static HTML caching system, fully controllable from either the bootstrap or our Controllers, and which supports a tag system meaning we can tag lots of related Actions, and expire them all in one go rather than worry about what they all are up front.

This entry continues in [Part 3b](#) due to space restrictions in Serendipity which may yet drive me insane... 

Posted by Pádraic Brady in PHP General, PHP Security, Zend Framework at 01:14

lowest airfares [url=https://wiki.koumbit.net/AirTickets?action=AttachFile&do=get&target=m]british airways [/url] should discount big e tickets compare cheap international flights links because

[url=https://wiki.koumbit.net/AirTickets?action=AttachFile&do=get&target=m]dallas airline[/url] around the world plane ticket ,

[url=https://wiki.koumbit.net/AirTickets?action=AttachFile&do=get&target=m]international airlines[/url] outside take us .and again blue sky airlines fares look I need or someone e flight rc . Anonymous on Feb 4 2009, 09:29