lodev09 / **php-cache-class**

Watch 1 Star 1 Fork 0

Code Issues 0 Pull requests 0 Pulse Graphs

Branch: **master** php-cache-class / lib / **class.cache.php** Find file Copy path

lodev09 Version 1.0 8cf974e on Apr 9, 2014

1 contributor

251 lines (229 sloc) 7.85 KB Raw Blame History

```php
<?php

/**
 * @package Cache - A simple file based cache (based from Erik Giberti's FileCache class. http://af-design.com/blog/2010/07/30/simple-file-
 * @link http://www.lodev09.com
 * @author Jovanni Lo
 * @copyright 2014 Jovanni Lo, all rights reserved
 * @license
 * The MIT License (MIT)
 * Copyright (c) 2014 Jovanni Lo
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 *
 * Class to implement a file based cache. This is useful for caching large objects such as
 * API/Curl responses or HTML results that aren't well suited to storing in small memory caches
 * or are infrequently accessed but are still expensive to generate.
 *
 * For security reasons, it's *strongly* recommended you set your cache directory to be outside
 * of your web root and on a drive independent of your operating system.
 *
 * Uses JSON, PHP native serialization and encryption/decryption
 *
 * Sample usage:
 *
 * $cache = new Cache('/var/www/cache/');
 * $data = $cache->get('sampledata');
 * if(!$data){
 *   $data = array('a'=>1,'b'=>2,'c'=>3);
 *   $cache->set('sampledata', $data, 3600);
 * }
 * print $data['a'];
 *
 */

class Cache {

    /**
     * Value is pre-pended to the cache, should be the full path to the directory
     * @var string

     */
    protected $root = '/tmp/';

    /**
     * For holding any error messages that may have been raised
     * @var string
     */
    protected $error = null;

    /**
     * The encryption key. This is private! set this inside this class
     * @var string
     */
    private $_encryption_key = "Fil3C@ch33ncryptionK3y";

    /**
     * @param string $root The root of the file cache.
     */
    function __construct($root = '/tmp/') {
        $this->root = $root;
        // Requires the native JSON library
        if (!function_exists('json_decode') || !function_exists('json_encode')) {
            throw new Exception('Cache needs the JSON PHP extensions.');
        }
    }

    /**
     * Saves data to the cache. Anything that evaluates to false, null, '', boolean false, 0 will
     * not be saved.
     * @param string $key An identifier for the data
     * @param mixed $data The data to save
     * @param int $ttl Seconds to store the data
     * @returns boolean True if the save was successful, false if it failed
     */
    public function set($key, $data = false, $ttl = 3600) {
        if (!$key) {
            $this->error = "Invalid key";
            return false;
        }
        if (!$data) {
            $this->error = "Invalid data";
            return false;
        }

        $key = $this->_make_file_key($key);
        $store = array(
            'data' => serialize($data),
            'ttl' => time() + $ttl,
            );
        $status = false;
        try {
            $fh = fopen($key, "w+");
            if (flock($fh, LOCK_EX)) {
                ftruncate($fh, 0);
                fwrite($fh, $this->_encrypt(json_encode($store)));
                flock($fh, LOCK_UN);
                $status = true;
            }
            fclose($fh);
        }
        catch (exception $e) {
            $this->error = "Exception caught: ".$e->getMessage();
            return false;
        }
        return $status;
    }

    /**
     * Reads the data from the cache
     * @param string $key An identifier for the data
     * @returns mixed Data that was stored
     */
    public function get($key) {
        if (!$key) {
            $this->error = "Invalid key";
            return false;
        }

        $key = $this->_make_file_key($key);
        $file_content = null;

        if (file_exists($key) !== true) {
            return false;
        }

        // Get the data from the file
        try {
            $fh = fopen($key, "r");
            if (flock($fh, LOCK_SH)) {
                $file_content = trim($this->_decrypt(fread($fh, filesize($key))));
            }
            fclose($fh);
        }
        catch (exception $e) {
            $this->error = "Exception caught: ".$e->getMessage();
            return false;
        }

        // Assuming we got something back...
        if ($file_content) {
            $store = json_decode($file_content, true);
            if ($store['ttl'] < time()) {
                unlink($key); // remove the file
                $this->error = "Data expired";
                return false;
            } else return unserialize($store['data']);
        } else return false;
    }

    /**
     * Remove a key, regardless of it's expire time
     * @param string $key An identifier for the data
     */
    public function delete($key) {
        if (!$key) {
            $this->error = "Invalid key";
            return false;
        }

        $key = $this->_make_file_key($key);

        try {
            unlink($key); // remove the file
        }
        catch (exception $e) {
            $this->error = "Exception caught: ".$e->getMessage();
            return false;
        }

        return true;
    }

    /**
     * Reads and clears the internal error
     * @returns string Text of the error raised by the last process
     */
    public function get_error() {
        $message = $this->error;
        $this->error = null;
        return $message;
    }

    /**
     * Can be used to inspect internal error
     * @returns boolean True if we have an error, false if we don't
     */
    public function have_error() {
        return ($this->error !== null) ? true : false;
    }

    /**
     * returns an encrypted string
     * @param  string $pure_string source string to encrypt
     * @return string                   decrypted string
     */
    private function _encrypt($pure_string) {
        $iv_size = mcrypt_get_iv_size(MCRYPT_BLOWFISH, MCRYPT_MODE_ECB);
        $iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);
        $encrypted_string = mcrypt_encrypt(MCRYPT_BLOWFISH, $this->_encryption_key, utf8_encode($pure_string),
            MCRYPT_MODE_ECB, $iv);
        return $encrypted_string;
    }

    /**
     * returns a decrypted string
     * @param  string $encrypted_string ecrypted string
     * @return string                   decrypted string
     */
    private function _decrypt($encrypted_string) {
        $iv_size = mcrypt_get_iv_size(MCRYPT_BLOWFISH, MCRYPT_MODE_ECB);
        $iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);
        $decrypted_string = mcrypt_decrypt(MCRYPT_BLOWFISH, $this->_encryption_key, $encrypted_string,
            MCRYPT_MODE_ECB, $iv);
        return $decrypted_string;
    }

    /**
     * Create a key for the cache
     * @todo Beef up the cleansing of the file.
     * @param string $key The key to create
     * @returns string The full path and filename to access
     */
    private function _make_file_key($key) {
        $safe_key = str_replace(array(
            '.',
            '/',
            ':',
            '\''), array(
            '_',
            '-',
            '-',
            '-'), trim($key));
        return $this->root.$safe_key.".cache";
    }
}
?>
```