cosenary / **Simple-PHP-Cache**

Watch 24 Star 182 Fork 58

Code Issues 6 Pull requests 8 Pulse Graphs

Branch: **master** **Simple-PHP-Cache** / **cache.class.php** Find file Copy path

cosenary Version 1.6 ce840c6 Jan 4, 2014

3 contributors

314 lines (288 sloc) 6.81 KB Raw Blame History

```php
<?php

/**
 * Simple Cache class
 * API Documentation: https://github.com/cosenary/Simple-PHP-Cache
 *
 * @author Christian Metz
 * @since 22.12.2011
 * @copyright Christian Metz - MetzWeb Networks
 * @version 1.6
 * @license BSD http://www.opensource.org/licenses/bsd-license.php
 */

class Cache {

  /**
   * The path to the cache file folder
   *
   * @var string
   */
  private $_cachepath = 'cache/';

  /**
   * The name of the default cache file
   *
   * @var string
   */
  private $_cachename = 'default';

  /**
   * The cache file extension
   *
   * @var string
   */
  private $_extension = '.cache';

  /**
   * Default constructor
   *
   * @param string|array [optional] $config
   * @return void
   */
  public function __construct($config = null) {
    if (true === isset($config)) {
      if (is_string($config)) {
        $this->setCache($config);
      } else if (is_array($config)) {
        $this->setCache($config['name']);
        $this->setCachePath($config['path']);
        $this->setExtension($config['extension']);
      }
    }
  }

  /**
   * Check whether data accociated with a key
   *
   * @param string $key
   * @return boolean
   */
  public function isCached($key) {
    if (false != $this->_loadCache()) {
      $cachedData = $this->_loadCache();
      return isset($cachedData[$key]['data']);
    }
  }

  /**
   * Store data in the cache
   *
   * @param string $key
   * @param mixed $data
   * @param integer [optional] $expiration
   * @return object
   */
  public function store($key, $data, $expiration = 0) {
    $storeData = array(
      'time'   => time(),
      'expire' => $expiration,
      'data'   => serialize($data)
    );
    $dataArray = $this->_loadCache();
    if (true === is_array($dataArray)) {
      $dataArray[$key] = $storeData;
    } else {
      $dataArray = array($key => $storeData);
    }
    $cacheData = json_encode($dataArray);
    file_put_contents($this->getCacheDir(), $cacheData);
    return $this;
  }

  /**
   * Retrieve cached data by its key
   *
   * @param string $key
   * @param boolean [optional] $timestamp
   * @return string
   */
  public function retrieve($key, $timestamp = false) {
    $cachedData = $this->_loadCache();
    (false === $timestamp) ? $type = 'data' : $type = 'time';
    if (!isset($cachedData[$key][$type])) return null;
    return unserialize($cachedData[$key][$type]);
  }

  /**
   * Retrieve all cached data
   *
   * @param boolean [optional] $meta
   * @return array
   */
  public function retrieveAll($meta = false) {
    if ($meta === false) {
      $results = array();
      $cachedData = $this->_loadCache();
      if ($cachedData) {
        foreach ($cachedData as $k => $v) {
          $results[$k] = unserialize($v['data']);
        }
      }
      return $results;
    } else {
      return $this->_loadCache();
    }
  }

  /**
```

```php
   * Erase cached entry by its key
   *
   * @param string $key
   * @return object
   */
  public function erase($key) {
    $cacheData = $this->_loadCache();
    if (true === is_array($cacheData)) {
      if (true === isset($cacheData[$key])) {
        unset($cacheData[$key]);
        $cacheData = json_encode($cacheData);
        file_put_contents($this->getCacheDir(), $cacheData);
      } else {
        throw new Exception("Error: erase() - Key '{$key}' not found.");
      }
    }
    return $this;
  }

  /**
   * Erase all expired entries
   *
   * @return integer
   */
  public function eraseExpired() {
    $cacheData = $this->_loadCache();
    if (true === is_array($cacheData)) {
      $counter = 0;
      foreach ($cacheData as $key => $entry) {
        if (true === $this->_checkExpired($entry['time'], $entry['expire'])) {
          unset($cacheData[$key]);
          $counter++;
        }
      }
      if ($counter > 0) {
        $cacheData = json_encode($cacheData);
        file_put_contents($this->getCacheDir(), $cacheData);
      }
      return $counter;
    }
  }

  /**
   * Erase all cached entries
   *
   * @return object
   */
  public function eraseAll() {
    $cacheDir = $this->getCacheDir();
    if (true === file_exists($cacheDir)) {
      $cacheFile = fopen($cacheDir, 'w');
      fclose($cacheFile);
    }
    return $this;
  }

  /**
   * Load appointed cache
   *
   * @return mixed
   */
  private function _loadCache() {
    if (true === file_exists($this->getCacheDir())) {
      $file = file_get_contents($this->getCacheDir());
      return json_decode($file, true);
    } else {
      return false;
    }
  }

  /**
   * Get the cache directory path
   *

   * @return string
   */
  public function getCacheDir() {
    if (true === $this->_checkCacheDir()) {
      $filename = $this->getCache();
      $filename = preg_replace('/[^0-9a-z\.\_\-]/i', '', strtolower($filename));
      return $this->getCachePath() . $this->_getHash($filename) . $this->getExtension();
    }
  }

  /**
   * Get the filename hash
   *
   * @return string
   */
  private function _getHash($filename) {
    return sha1($filename);
  }

  /**
   * Check whether a timestamp is still in the duration
   *
   * @param integer $timestamp
   * @param integer $expiration
   * @return boolean
   */
  private function _checkExpired($timestamp, $expiration) {
    $result = false;
    if ($expiration !== 0) {
      $timeDiff = time() - $timestamp;
      ($timeDiff > $expiration) ? $result = true : $result = false;
    }
    return $result;
  }

  /**
   * Check if a writable cache directory exists and if not create a new one
   *
   * @return boolean
   */
  private function _checkCacheDir() {
    if (!is_dir($this->getCachePath()) && !mkdir($this->getCachePath(), 0775, true)) {
      throw new Exception('Unable to create cache directory ' . $this->getCachePath());
    } elseif (!is_readable($this->getCachePath()) || !is_writable($this->getCachePath())) {
      if (!chmod($this->getCachePath(), 0775)) {
        throw new Exception($this->getCachePath() . ' must be readable and writeable');
      }
    }
    return true;
  }

  /**
   * Cache path Setter
   *
   * @param string $path
   * @return object
   */
  public function setCachePath($path) {
    $this->_cachepath = $path;
    return $this;
  }

  /**
   * Cache path Getter
   *
   * @return string
   */
  public function getCachePath() {
    return $this->_cachepath;
  }

  /**
   * Cache name Setter
   *

   * @param string $name
```

```
     * @return object
     */
    public function setCache($name) {
      $this->_cachename = $name;
      return $this;
    }

    /**
     * Cache name Getter
     *
     * @return void
     */
    public function getCache() {
      return $this->_cachename;
    }

    /**
     * Cache file extension Setter
     *
     * @param string $ext
     * @return object
     */
    public function setExtension($ext) {
      $this->_extension = $ext;
      return $this;
    }

    /**
     * Cache file extension Getter
     *
     * @return string
     */
    public function getExtension() {
      return $this->_extension;
    }

  }
```

© 2016 GitHub, Inc. Terms Privacy Security Contact Help Status API Training Shop Blog About