Comodojo cache documentation

Release 2.0.0

Marco Giovinazzi

Jun 14, 2018

Contents

This library provides a fast, PSR-6 and PSR-16 compliant, enhanced data caching layer for PHP applications.

It also includes a *Cache Manager* to handle different cache providers at the same time.

Main features:

- PSR-6 and PSR-16 full compliance
- support for cache namespaces
- comes with a complete *Cache Manager*

> **Warning:** This documentation refers to comodojo/cache version 2.0.
>
> Documentation for version 1.0 (not PSR-6 or PSR-16 compliant) is available here.

CHAPTER 1

Installation

First install composer, then:

```
composer require comodojo/cache
```

1.1 Requirements

To work properly, comodojo/cache requires PHP >=5.6.0.

Some packages are optional but reccomended:

- ext-xattr: Fastest cache files handling via extended attributes
- ext-redis: Enable redis provider
- ext-memcached: Enable Memcached provider
- ext-apc: Enable Apc provider (apcu_bc also supported)
- ext-apcu: Enable Apcu provider

CHAPTER 2

Cache providers

Actually this library supports cache over:

- Apc
- Apcu
- Filesystem
- Memcached
- Memory
- Redis
- Vacuum

Each provider offers same functionalities and methods, but class constructors may accept different parameters.

Note: Both PSR-6 (Cache) and PSR-16 (SimpleCache) providers share same constructors.

Following examples can, therefore, be applied in both cases.

2.1 Apc & Apcu

Cache items using Alternative PHP Cache or APC User Cache.

Note: To enable these providers, apc (apcu_bc) or apcu extensions should be installed and enabled.

In case of CLI SAPI, remember to add configuration param: *apc.enable_cli => On*

These providers do not accept parameters.

Code example:

```
1 <?php
2
3 use \Comodojo\Cache\Providers\Apc;
```

(continues on next page)

(continued from previous page)

```php
use \Comodojo\Cache\Providers\Apcu;

$apcu_cache = new Apcu();
$apc_cache = new Apc();
```

2.2 Filesystem

This provider will store cached items on filesystem, using ghost-files or xattr (preferred) drivers to persist metadata.

Following parameters are expected:

- *cache_folder*: location (local or remote) to store file to.

Code example:

```php
<?php

use \Comodojo\Cache\Providers\Filesystem;

$fs_cache = new Filesystem([
    'cache_folder' => "/my/cache/folder"
]);
```

About filesystem drivers

This library offers two different filesystem-cache drivers, managed seamlessly from Filesystem provider. Both driver save cached data into single filesystem files, but each one uses a different strategy to save ttl information.

If available, Filesystem provider will try to save ttl information inside file's Extended Attributes. If not, a ghost file (.expire) will be created near data file (.cache).

This duplication is made for performance reasons: xattr open/read/close a single file handler, ghost files duplicate the effort saving or reading iformations.

Using 10k key-value pairs inside a docker container:

```
Runtime:       PHP 7.2.3

> (GHOST) set 10k data: 1.3727879524231 secs
> (GHOST) check 10k keys: 0.11777091026306 secs
> (GHOST) get 10k data: 0.18857002258301 secs
> (GHOST) total test time: 1.6791288852692 secs

> (XATTR) set 10k data: 0.76364898681641 secs
> (XATTR) check 10k keys: 0.048287868499756 secs
> (XATTR) get 10k data: 0.12987494468689 secs
> (XATTR) total test time: 0.94181180000305 secs
```

Using 100k key-value pairs inside a docker container:

```
Runtime:       PHP 7.2.3

> (GHOST) set 10k data: 15.756072998047 secs
> (GHOST) check 10k keys: 16.93918800354 secs
> (GHOST) get 10k data: 53.536478996277 secs
> (GHOST) total test time: 86.231739997864 secs

> (XATTR) set 10k data: 9.375433921814 secs
> (XATTR) check 10k keys: 0.55717587471008 secs
> (XATTR) get 10k data: 1.9446270465851 secs
> (XATTR) total test time: 11.877236843109 secs
```

To recap: in case of ghost file, two files will be created into cache folder for each item:

- MYITEM-MYNAMESPACE.cache
- MYITEM-MYNAMESPACE.expire

The first one will hold data, the second one will mark the ttl.

In case of xattr support, only one file (.cache) will be created; ttl will be stored into file's attributes and filesystem cache will perform better.

2.3 Memcached

Cache items using a memcached instance.

Note: To enable this provider, memcached extension should be installed and enabled.

This provider accepts following parameters:

- *server*: (default '127.0.0.1')
- *port*: (default 11211)
- *weight*: (default 0)
- *persistent_id*: (default null)
- *username*: (default null)
- *password*: (default null)

Code example:

```
<?php

use \Comodojo\Cache\Providers\Memcached;

$memcached_cache = new Memcached([
    "server" => "memcached.example.com",
    "port" => 11212
]);
```

2.4 Memory

This provider will hold an array containing cached key value pairs; it does not accept parameters.

Code example:

```
<?php

use \Comodojo\Cache\Providers\Memory;

$memory_cache = new Memory();
```

2.5 PhpRedis

Cache items using a redis instance.

Note: To enable this provider, redis extension should be installed and enabled.

This provider accepts following parameters:

- *server*: (default '127.0.0.1')
- *port*: (default 6379)
- *timeout*: (default 0)
- *password*: (default null)

Code example:

```
<?php

use \Comodojo\Cache\Providers\PhpRedis;

$memcached_cache = new PhpRedis([
    "server" => "redis.example.com",
    "port" => 6378
]);
```

2.6 Vacuum

This provider will offer a handy way to discard any cached data; in other words, every key-value pair that is cached inside a vacuum provider will be trashed.

This provider does not accept parameters.

Code example:

```
<?php

use \Comodojo\Cache\Providers\Vacuum;

$vacuum_cache = new Vacuum();
```

CHAPTER 3

Using cache providers

Cache providers can be used as a standalone cache interface to most common cache engine.

Note: For an updated list of supported engines, please refer to *Cache providers*.

Each provider is available in two different namespace:

- *Comodojo\Cache\Providers* provides PSR-6-compatible classes
- *Comodojo\SimpleCache\Providers* provides PSR-16-compatible classes

3.1 PSR-6 (Caching Interface) usage

Following a list of common methods offered by each provider. For a detailed description of each method, please refer to the PSR-6 standard.

3.1.1 CRUD operations

```php
<?php

use \Comodojo\Cache\Providers\Memory;
use \Comodojo\Cache\Item;

// init provider
$cache = new Memory();

// create a 'foo' cache item,
// set its value to "Ford Perfect",
// declare a ttl of 600 secs
$item = new Item('foo');
$item->set('Ford Perfect')
    ->expiresAfter(600);

// persist item 'foo'
$cache->save($item);
```

(continues on next page)

(continued from previous page)

```php

// retrieve item 'foo'
$retrieved = $cache->getItem('foo');
$hit = $retrieved->isHit(); // returns true

// update item with value 'Marvin'
$retrieved->set('Marvin');
$cache->save($retrieved);

// delete 'foo'
$cache->deleteItem('foo');
```

3.1.2 Write-deferred

```php
<?php

use \Comodojo\Cache\Providers\Memory;
use \Comodojo\Cache\Item;

// init provider
$cache = new Memory();

// create a 'foo' cache item,
// set its value to "Ford Perfect",
// declare a ttl of 600 secs
$item = new Item('foo');
$item->set('Ford Perfect')
    ->expiresAfter(600);

// send item 'foo' to cache provider for deferred commit
$cache->saveDeferred($item);

// do some other stuff...

// commit item 'foo'
$deferred = $cache->commit(); // returns true
```

3.1.3 Batch operations

```php
<?php

use \Comodojo\Cache\Providers\Memory;
use \Comodojo\Cache\Item;

// init provider
$cache = new Memory();

// create two cache items 'foo' and 'boo'
$foo = new Item('foo');
$boo = new Item('boo');
$foo->set('Ford Perfect');
$boo->set('Marvin');

// send items to cache provider for deferred commit
$cache->saveDeferred($foo);
$cache->saveDeferred($foo);

```

(continues on next page)

(continued from previous page)

```php
// commit items 'foo' and 'boo'
$deferred = $cache->commit(); // returns true

// retrieve 'foo' and 'boo'
$items = $cache->getItems(['foo', 'boo']);
```

Note: *tests/Comodojo/Cache* folder contains several practical examples to learn from.

3.2 PSR-16 (Common Interface for Caching Libraries) usage

Following a list of common methods offered by each provider. For a detailed description of each method, please refer to the PSR-16 standard.

3.2.1 CRUD operations

```php
<?php

use \Comodojo\SimpleCache\Providers\Memory;

// init provider
$cache = new Memory();

// create a 'foo' cache item,
// set its value to "Ford Perfect",
// declare a ttl of 600 secs
$cache->set('foo', 'Ford Perfect', 600);

// retrieve item 'foo'
$retrieved = $cache->get('foo');

// update item with value 'Marvin'
$cache->set('foo', 'Marvin', 600);

// delete 'foo'
$cache->delete('foo');
```

3.2.2 Managing multiple items

```php
<?php

use \Comodojo\SimpleCache\Providers\Memory;

// init provider
$cache = new Memory();

// create 'foo' and 'boo' cache items
$cache->setMultiple([
    'foo' => 'Ford Perfect',
    'boo' => 'Marvin'
], 600);

// retrieve items
$retrieved = $cache->getMultiple(['foo', 'boo']);
```

Note: *tests/Comodojo/SimpleCache* folder contains several practical examples to learn from.

3.3 Extended cache features

In both flavours providers offer some extended functions that may be handy in some cases, mantaining compatibility with standards.

3.3.1 State-aware provider implementation

To handle failure of underlying cache engines, each provider offer a set of methods to know the provider's status.

Status updates are managed seamlessly by provider itself.

```php
<?php

use \Comodojo\SimpleCache\Providers\Memcached;

// init provider
$cache = new Memcached();

// get the provider state
$cache->getState(); //return 0 if everything ok, 1 otherwise
$cache->getStateTime(); //return a DateTime object containing the reference to the time of state definition

// test the pool
$cache->test(); // returns a bool indicating how the test ends and sets the state according to test result
```

3.3.2 Namespaces support

Each item in cache is placed into a namespace ('GLOBAL' is the default one) and providers can switch from one namespace to another.

In other words, the entire cache space is partitioned by default, and different items can belong to a single partition at a time.

```php
<?php

use \Comodojo\SimpleCache\Providers\Memory;

// init provider
$cache = new Memory();

// set (a new) namespace to "CUSTOM"
$cache->setNamespace('CUSTOM');

// get the current namespace
$cache->getNamespace(); //return 'CUSTOM'

// save an item into 'CUSTOM' namespace
$cache->set('foo', 'Ford Perfect', 600);

// move to 'ANOTHER' namespace
$cache->setNamespace('ANOTHER');

```

(continues on next page)

(continued from previous page)

```php
// try to get back the 'foo' item
$cache->get('foo'); // returns null: 'foo' is not in 'ANOTHER' namespace!

// clear the 'ANOTHER' namespace
$cache->clearNamespace();

// since 'foo' belongs to 'CUSTOM' namespace, it was not deleted
$cache->setNamespace('CUSTOM');
$foo = $cache->get('foo'); // returns 'Ford Perfect'
```

3.3.3 Cache statistics

Stats about current provider can be accessed using the *$provider::getStats* method. It returns a *EnhancedCacheItemPoolStats* object.

```php
<?php

use \Comodojo\SimpleCache\Providers\Memory;

// init provider
$cache = new Memory();

// do some stuff with $cache...

// get statistics about $cache
$stats = $cache->getStats();

// get n. of objects in pool
$num = $stats->getObjects();
```

CHAPTER 4

Cache Manager

The Cache Manager component is a state-aware container that can use one or more cache provider at the same time.

In other words, Cache Manager can be configured to use one or more cache providers with a flexible selection strategy (pick algorithm).

Note: This library provides two different implementation of cache manager:

- *Comodojo\Cache\Manager* (PSR-6)
- *Comodojo\SimpleCache\Manager* (PSR-16)

Let's consider this example:

```php
<?php

use \Comodojo\Cache\Manager;
use \Comodojo\Cache\Providers\Memcached;
use \Comodojo\Cache\Providers\Memory;

$manager = new Manager(Manager::PICK_FIRST);

$memcached_cache = new Memcached();
$memory_cache = new Memory();

$manager->addProvider($memcached_cache);
$manager->addProvider($memory_cache);

$item = $this->manager->getItem('Ford');
```

In this example, manager was feeded with two different providers (memcached and memory); according to pick algorithm (PICK_FIRST), the item Ford is retrieved from the first provider in stack (memcached). In case of memcached failure, first provider will be suspended and memory will be used instead.

This is particularly useful to ensure that application will continue to have an active cache layer also if preferred one is failing.

4.1 Selection Strategy (Pick Algorithm)

Providers are organized placed on a stack and picked according to the selected strategy.

Currently the manager supports six different pick algorithms.

4.1.1 Manager::PICK_FIRST

Select the first (enabled) provider in stack; do not traverse the stack if value is missing.

Note: this is the default algorithm.

4.1.2 Manager::PICK_LAST

Select the last (enabled) provider in stack; do not traverse the stack if value is missing.

4.1.3 Manager::PICK_RANDOM

Select a random (enabled) provider in stack; do not traverse the stack if value is missing.

4.1.4 Manager::PICK_BYWEIGHT

Select a provider by weight, stop at first enabled one.

Weight is an integer (tipically 1 to 100); selection is made considering the greather weight of available (and enabled) providers.

4.1.5 Manager::PICK_ALL

Ask to all (enabled) providers and match responses.

This is useful during tests but not really convenient in production because of the latency introduced that increase linearly with number of providers into the stack.

4.1.6 Manager::PICK_TRAVERSE

Select the first (enabled) provider, in case of null response traverse the stack.

4.2 Align cache between providers

By default manager will try to set/update/delete cache items in any active provider. This beaviour is particularly convenient to ensure availability of cache information also in case the master provider fails.

On the other side, cache performances can really get worse: the total number of iteration for a single, atomic transaction will increase linearly with the number of providers defined into the stack.

This feature can be disabled during class init:

```php
<?php

use \Comodojo\Cache\Manager;
use \Comodojo\Cache\Providers\Memcached;
use \Comodojo\Cache\Providers\Memory;

// init the manager
// PICK_FIRST strategy
// null logger
// do not align cache between providers
$manager = new Manager(Manager::PICK_FIRST, null, false);
```

4.3 Using the Manager

The manager is itself a provider, therefore can be used like any other PSR-6 or PSR-16 provider. It also supports *Extended cache features*.

Just to make a working example:

```php
<?php

use \Comodojo\Cache\Manager;
use \Comodojo\Cache\Providers\Memcached;
use \Comodojo\Cache\Providers\Memory;

// init the manager
// PICK_FIRST strategy
$manager = new Manager(Manager::PICK_BYWEIGHT);

// push two providers to manager's stack
// memcached will be the preferred provider due to its weight
$memcached_cache = new Memcached();
$memory_cache = new Memory();
$manager->addProvider($memcached_cache, 100);
$manager->addProvider($memory_cache, 10);

// create a 'foo' cache item,
// set its value to "Ford Perfect",
// declare a ttl of 600 secs
$item = new Item('foo');
$item->set('Ford Perfect')
    ->expiresAfter(600);

// item 'foo' will be saved in both providers
$manager->save($item);

// retrieve item 'foo' from preferred provider
$retrieved = $manager->getItem('foo');
$hit = $retrieved->isHit(); // returns true

// update item with value 'Marvin'
// since the align_cache flag was leaved to default (true), the update operation
↪will be performed into both providers
$retrieved->set('Marvin');
$manager->save($retrieved);

// delete 'foo'
$manager->deleteItem('foo');
// item is deleted from both providers
```