# [Laravel 的缓存源码解析](https://segmentfault.com/a/1190000019136728)

# 前言

Laravel 支持多种缓存系统, 并提供了统一的api接口.

(Laravel 5.5)默认支持的存储驱动包括如下:

* file (默认使用)
* apc
* array (数组, 测试用)
* database (关系型数据库)
* memcached
* redis

默认的缓存配置文件在 config/cache.php

# 使用

直接使用Laravel为我们提供的Facade

**useIlluminate**\**Support**\**Facades**\**Cache**;

$cache = Cache::get('key');

支持的大部分方法:

Cache::put('key', 'value', $minutes);

Cache::add('key', 'value', $minutes);

Cache::forever('key', 'value');

Cache::remember('key', $minutes, **function**(){ **return**'value' });

Cache::rememberForever('key', **function**(){ **return**'value' });

Cache::forget('key');

Cache::has('key');

Cache::get('key');

Cache::get('key', 'default');

Cache::get('key', **function**(){ **return**'default'; });

Cache::tags('my-tag')->put('key','value', $minutes);

Cache::tags('my-tag')->has('key');

Cache::tags('my-tag')->get('key');

Cache::tags('my-tag')->forget('key');

Cache::tags('my-tag')->flush();

Cache::increment('key');

Cache::increment('key', $amount);

Cache::decrement('key');

Cache::decrement('key', $amount);

Cache::tags('group')->put('key', $value);

Cache::tags('group')->get('key');

Cache::tags('group')->flush();

# 源码

Laravel 中常用 Cache Facade 来操作缓存, 对应的实际类是 Illuminate\Cache\CacheManager 缓存管理类(工厂).

Cache::xxx()

我们通过 CacheManager 类获取持有不同存储驱动的 Illuminate\Cache\Repository 类

CacheManager::store($name = null)

Repository 仓库类代理了实现存储驱动接口 Illuminate\Contracts\Cache\Store 的类实例.

## Cache Facade

首先从 Cache Facade 开始分析, 先看一下其源码:

**<?php**

**namespaceIlluminate**\**Support**\**Facades**;

/\*\*

\* @see \Illuminate\Cache\CacheManager

\* @see \Illuminate\Cache\Repository

\*/**classCacheextendsFacade**{

/\*\*

\* Get the registered name of the component.

\*

\* @return string

\*/

**protectedstaticfunctiongetFacadeAccessor**()

{

**return**'cache';

}

}

在配置文件 config\app.php 中定义了 Cache 服务提供者

//...'providers' => [

// ......

Illuminate\Cache\CacheServiceProvider::class,

// ......

],//...

Illuminate\Cache\CacheServiceProvider 源文件:

**<?php**

**namespaceIlluminate**\**Cache**;

**useIlluminate**\**Support**\**ServiceProvider**;

**classCacheServiceProviderextendsServiceProvider**{

// ......

**publicfunctionregister**()

{

**$this**->app->singleton('cache', **function** ($app) {

**returnnew** CacheManager($app);

});

**$this**->app->singleton('cache.store', **function** ($app) {

**return** $app['cache']->driver();

});

**$this**->app->singleton('memcached.connector', **function** () {

**returnnew** MemcachedConnector;

});

}

// ......

}

通过上面源码可知, Cache Facade 关联的项是 Illuminate\Cache\CacheManager, 也就是我们通过 Cache Facade 实际调用的是 CacheManager实例的方法.

## CacheManager

**<?php**

**namespaceIlluminate**\**Contracts**\**Cache**;

**interfaceFactory**{

/\*\*

\* Get a cache store instance by name.

\*

\* @param string|null $name

\* @return \Illuminate\Contracts\Cache\Repository

\*/

**publicfunctionstore**($name = null);

}

CacheManager 实现了 Illuminate\Contracts\Cache\Factory 接口(↑), 即实现了一个简单工厂, 传入存储驱动名, 返回对应的驱动实例.

CacheManager实现的简单工厂接口方法:

**<?php**

**namespaceIlluminate**\**Cache**;

**useClosure**;**useInvalidArgumentException**;**useIlluminate**\**Contracts**\**Cache**\**Store**;**useIlluminate**\**Contracts**\**Cache**\**FactoryasFactoryContract**;**useIlluminate**\**Contracts**\**Events**\**DispatcherasDispatcherContract**;

/\*\*

\* @mixin \Illuminate\Contracts\Cache\Repository

\*/**classCacheManagerimplementsFactoryContract**{

/\*\*

\* Get a cache store instance by name.

\*

\* @param string|null $name

\* @return \Illuminate\Contracts\Cache\Repository

\*/

**publicfunctionstore**($name = null)

{

$name = $name ?: **$this**->getDefaultDriver();

**return$this**->stores[$name] = **$this**->get($name);

}

/\*\*

\* Get the default cache driver name.

\*

\* @return string

\*/

**publicfunctiongetDefaultDriver**()

{

**return$this**->app['config']['cache.default'];

}

/\*\*

\* Attempt to get the store from the local cache.

\*

\* @param string $name

\* @return \Illuminate\Contracts\Cache\Repository

\*/

**protectedfunctionget**($name)

{

**return$this**->stores[$name] ?? **$this**->resolve($name);

}

/\*\*

\* Resolve the given store.

\*

\* @param string $name

\* @return \Illuminate\Contracts\Cache\Repository

\*

\* @throws \InvalidArgumentException

\*/

**protectedfunctionresolve**($name)

{

$config = **$this**->getConfig($name);

**if** (is\_null($config)) {

**thrownew** InvalidArgumentException("Cache store [{$name}] is not defined.");

}

**if** (**isset**(**$this**->customCreators[$config['driver']])) {

**return$this**->callCustomCreator($config);

} **else** {

$driverMethod = 'create'.ucfirst($config['driver']).'Driver';

**if** (method\_exists(**$this**, $driverMethod)) {

**return$this**->{$driverMethod}($config);

} **else** {

**thrownew** InvalidArgumentException("Driver [{$config['driver']}] is not supported.");

}

}

}

/\*\*

\* Dynamically call the default driver instance.

\*

\* @param string $method

\* @param array $parameters

\* @return mixed

\*/

**publicfunction\_\_call**($method, $parameters)

{

**return$this**->store()->$method(...$parameters);

}

}

可以看到 CacheManager 提供了**会话级别**的实例缓存, 当解析驱动名时, 它会按如下顺序解析:

1. 自定义驱动: 查看是否有通过 CacheManager::extend(...)自定义的驱动
2. Laravel提供的驱动: 查看是否存在 CacheManager::createXxxDriver(...)方法

这些方法返回的实例必须是实现了 Illuminate\Contracts\Cache\Repository 接口

本质上, CacheManager 就是一个提供了**会话级别缓存**的 Repository 实例工厂, 同时它提供了一个 \_\_call 魔术方法, 以便快速调用默认缓存驱动.

$value = Cache::store('file')->get('foo');

// 通过 \_call, 调用默认缓存驱动的 get 方法

$value = Cache::get('key');

## Repository

Illuminate\Contracts\Cache\Repository 接口

**<?php**

**namespaceIlluminate**\**Contracts**\**Cache**;

**useClosure**;**usePsr**\**SimpleCache**\**CacheInterface**;

**interfaceRepositoryextendsCacheInterface**{

**publicfunctionhas**($key);

**publicfunctionget**($key, $default = null);

**publicfunctionpull**($key, $default = null);

**publicfunctionput**($key, $value, $minutes);

**publicfunctionadd**($key, $value, $minutes);

**publicfunctionincrement**($key, $value = 1);

**publicfunctiondecrement**($key, $value = 1);

**publicfunctionforever**($key, $value);

**publicfunctionremember**($key, $minutes, Closure $callback);

**publicfunctionsear**($key, Closure $callback);

**publicfunctionrememberForever**($key, Closure $callback);

**publicfunctionforget**($key);

**publicfunctiongetStore**();

}

Repository 是一个符合 [PSR-16: Common Interface for Caching Libraries](http://www.baidu.com/link?url=9LkUzilH74QMr6SuXLs5wqzeJAXlxKxOoz7qJLQSMmy8JHcYUiJj_wtVBslGK2dB) 规范的缓存仓库类, 其在Laravel相应的实现类: Illuminate\Cache\Repository

Illuminate\Cache\Repository 部分代码如下:

**<?php**

**namespaceIlluminate**\**Cache**;

**useClosure**;**useArrayAccess**;**useDateTimeInterface**;**useBadMethodCallException**;**useIlluminate**\**Support**\**Carbon**;**useIlluminate**\**Cache**\**Events**\**CacheHit**;**useIlluminate**\**Contracts**\**Cache**\**Store**;**useIlluminate**\**Cache**\**Events**\**KeyWritten**;**useIlluminate**\**Cache**\**Events**\**CacheMissed**;**useIlluminate**\**Support**\**Traits**\**Macroable**;**useIlluminate**\**Cache**\**Events**\**KeyForgotten**;**useIlluminate**\**Support**\**InteractsWithTime**;**useIlluminate**\**Contracts**\**Events**\**Dispatcher**;**useIlluminate**\**Contracts**\**Cache**\**RepositoryasCacheContract**;

/\*\*

\* @mixin \Illuminate\Contracts\Cache\Store

\*/**classRepositoryimplementsCacheContract**, **ArrayAccess**{

**useInteractsWithTime**;

**useMacroable** {

**\_\_callasmacroCall**;

}

/\*\*

\* The cache store implementation.

\*

\* @var \Illuminate\Contracts\Cache\Store

\*/

**protected** $store;

/\*\*

\* The event dispatcher implementation.

\*

\* @var \Illuminate\Contracts\Events\Dispatcher

\*/

**protected** $events;

**protected** $default = 60;

/\*\*

\* Create a new cache repository instance.

\*

\* @param \Illuminate\Contracts\Cache\Store $store

\* @return void

\*/

**publicfunction\_\_construct**(Store $store)

{

**$this**->store = $store;

}

**publicfunctionhas**($key)

{

**return** ! is\_null(**$this**->get($key));

}

**publicfunctionget**($key, $default = null)

{

**if** (is\_array($key)) {

**return$this**->many($key);

}

$value = **$this**->store->get(**$this**->itemKey($key));

// If we could not find the cache value, we will fire the missed event and get

// the default value for this cache value. This default could be a callback

// so we will execute the value function which will resolve it if needed.

**if** (is\_null($value)) {

**$this**->event(**new** CacheMissed($key));

$value = value($default);

} **else** {

**$this**->event(**new** CacheHit($key, $value));

}

**return** $value;

}

**publicfunctionpull**($key, $default = null)

{

**return** tap(**$this**->get($key, $default), **function** ($value) **use** ($key) {

**$this**->forget($key);

});

}

**protectedfunctionevent**($event)

{

**if** (**isset**(**$this**->events)) {

**$this**->events->dispatch($event);

}

}

/\*\*

\* Set the event dispatcher instance.

\*

\* @param \Illuminate\Contracts\Events\Dispatcher $events

\* @return void

\*/

**publicfunctionsetEventDispatcher**(Dispatcher $events)

{

**$this**->events = $events;

}

**publicfunction\_\_call**($method, $parameters)

{

**if** (**static**::hasMacro($method)) {

**return$this**->macroCall($method, $parameters);

}

**return$this**->store->$method(...$parameters);

}

**publicfunction\_\_clone**()

{

**$this**->store = **clone$this**->store;

}

}

从源码可以看出, Illuminate\Cache\Repository 实现了代理模式, 具体的实现是交由 Illuminate\Contracts\Cache\Store来处理, Repository 主要作用是

1. 提供一些便捷操作(可以理解为语法糖)
2. Event 事件触发, 包括缓存命中/未命中、写入/删除键值

## Store

Illuminate\Contracts\Cache 缓存驱动是实际处理缓存如何写入/读取/删除的类, 接口内容如下:

**<?php**

**namespaceIlluminate**\**Contracts**\**Cache**;

**interfaceStore**{

**publicfunctionget**($key);

**publicfunctionmany**(array $keys);

**publicfunctionput**($key, $value, $minutes);

**publicfunctionputMany**(array $values, $minutes);

**publicfunctionincrement**($key, $value = 1);

**publicfunctiondecrement**($key, $value = 1);

**publicfunctionforever**($key, $value);

**publicfunctionforget**($key);

**publicfunctionflush**();

**publicfunctiongetPrefix**();

}

具体的实现类有:

* ApcStore
* ArrayStore
* NullStore
* DatabaseStore
* FileStore
* MemcachedStore
* RedisStore