Course - Laravel Framework

Cache

Một số tác vụ truy xuất hoặc xử lý dữ liệu do ứng dụng của bạn thực hiện có thể tốn nhiều CPU hoặc mất vài giây để hoàn thành. Khi trường hợp này xảy ra, thông thường dữ liệu đã truy xuất được lưu vào bộ nhớ cache trong một thời gian để có thể truy xuất nhanh trong các request tiếp theo đối với cùng một dữ liệu.

Tags: laravel cache, laravel

Giới thiệu

Một số tác vụ truy xuất hoặc xử lý dữ liệu do ứng dụng của bạn thực hiện có thể tốn nhiều CPU hoặc mất vài giây để hoàn thành. Khi trường hợp này xảy ra, thông thường dữ liệu đã truy xuất được lưu vào bộ nhớ cache trong một thời gian để có thể truy xuất nhanh trong các request tiếp theo đối với cùng một dữ liệu. Dữ liệu đã lưu trong bộ nhớ cache thường được lưu trữ trong một kho dữ liệu rất nhanh như Memcached hoặc Redis.

Rất may, Laravel cung cấp một API thống nhất, rõ ràng cho các phần phụ trợ bộ nhớ cache khác nhau, cho phép bạn tận dụng khả năng truy xuất dữ liệu nhanh như chớp của chúng và tăng tốc ứng dụng web của bạn.

Cấu hình

Tập tin cấu hình bộ nhớ cache của ứng dụng của bạn được đặt tại *config/cache.php*. Trong tập tin này, bạn có thể chỉ định driver bộ nhớ cache nào bạn muốn được sử dụng theo mặc định trong toàn bộ ứng dụng của mình. Laravel hỗ trợ các driver bộ nhớ đệm phổ biến như Memcached, Redis, DynamoDB và cơ sở dữ liệu quan hệ. Ngoài ra, còn có driver bộ nhớ cache dựa trên tập tin có sẵn, trong khi `array` và driver bộ nhớ cache "null" cung cấp driver bộ nhớ cache thuận tiện cho các bài auto test của bạn.

Tập tin cấu hình bộ nhớ cache cũng chứa nhiều tùy chọn khác, được ghi lại trong tập tin, vì vậy hãy đảm bảo đọc qua các tùy chọn này. Mặc định, Laravel được định cấu hình để sử dụng `file` làm driver bộ nhớ cache, driver cache này lưu trữ các đối tượng được nén, và được lưu trong bộ nhớ cache trên hệ thống tập tin của máy chủ. Đối với các ứng dụng lớn hơn, bạn nên sử dụng driver mạnh mẽ hơn như Memcached hoặc Redis. Bạn thậm chí có thể cài đặt nhiều cấu hình bộ nhớ cache cho cùng một driver.

Các yêu cầu đối với driver

Database

Khi sử dụng `database` làm driver bộ đệm, bạn sẽ cần một bảng để chứa các mục bộ đệm. Bạn sẽ tìm thấy một `Schema` khai báo cho bảng như ví dụ dưới đây:

```
Schema::create('cache', function ($table) {
  $table->string('key')->unique();
  $table->text('value');
  $table->integer('expiration');
```

```
});
```

> Bạn cũng có thể sử dụng lệnh Artisan `php artisan cache:table` để tạo quá trình migrate với schema thích hợp.

Memcached

Sử dụng Memcached làm driver, bạn sẽ được yêu cầu cài đặt gói Memcached PECL. Bạn có thể liệt kê tất cả các máy chủ Memcached của mình trong tập tin cấu hình *config/cache.php*. Tập tin này có chứa một mục là memcached.servers để giúp bạn bắt đầu:

```
'memcached' => [
  'servers' => [
    [
      'host' => env('MEMCACHED_HOST', '127.0.0.1'),
      'port' => env('MEMCACHED_PORT', 11211),
      'weight' => 100,
    ],
  ],
],
```

Nếu cần, bạn có thể đặt `host` tùy chọn thành đường dẫn ổ cắm UNIX. Nếu bạn làm điều này, `port` tùy chọn sẽ được đặt thành `0`:

```
'memcached' => [
  [
    'host' => '/var/run/memcached/memcached.sock',
    'port' => 0,
    'weight' => 100
  ],
],
```

Redis

Trước khi sử dụng bộ đệm Redis với Laravel, bạn cần cài đặt phần mở rộng PhpRedis PHP qua PECL hoặc cài đặt gói `predis/predis` (~ 1.0) qua Composer. Laravel Sail đã có sẵn

phần mở rộng này. Ngoài ra, các nền tảng triển khai Laravel chính thức như Laravel Forge và Laravel Vapor có phần mở rộng PhpRedis được cài đặt sẵn theo mặc định.

Để biết thêm thông tin về cách cấu hình Redis, hãy tham khảo trang tài liệu Laravel của nó.

DynamoDB

Trước khi sử dụng driver bộ nhớ cache với DynamoDB, bạn phải tạo một bảng DynamoDB để lưu trữ tất cả dữ liệu đã lưu trong bộ nhớ cache. Thông thường, bảng này nên được đặt tên `cache`. Tuy nhiên, bạn nên đặt tên bảng dựa trên giá trị của giá trị cấu hình trong tập tin cấu hình `stores.dynamodb.table` cùng với tập tin cấu hình `cache` của ứng dụng của bạn

Bảng này cũng phải có key phân vùng kiểu chuỗi với tên tương ứng với giá trị của cấu hình trong tập tin cấu hình `stores.dynamodb.attributes.key` cùng với tập tin cấu hình cache của ứng dụng của bạn. Mặc định, key phân vùng sẽ được đặt tên `key`.

Cách dùng Cache

Nhận đối tượng Cache

Để nhận được một đối tượng lưu trữ bộ nhớ cache, bạn có thể sử dụng facade `Cache`, đây là những gì chúng tôi sẽ sử dụng trong suốt tài liệu này. `Cache` cung cấp quyền truy cập thuận tiện, ngắn gọn vào các thực thi cơ bản của các hợp đồng bộ nhớ cache Laravel:

```php
<?php
namespace App\Http\Controllers;
use Illuminate\Support\Facades\Cache;

class UserController extends Controller
{
  /**
   * Show a list of all users of the application.
   *
   * @return Response
   */
  public function index()
  {
```

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

    //
  }
}
```

Truy cập vào nhiều kho lưu trữ cache

Sử dụng facade `Cache`, bạn có thể truy cập các kho lưu trữ bộ nhớ cache khác nhau thông qua phương thức `store`. Key được truyền cho phương thức `store` phải tương ứng với một trong các kho lưu trữ được liệt kê trong mảng cấu hình `stores` trong tập tin cấu hình `cache` của bạn:

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

Cache::store('redis')->put('bar', 'baz', 600); // 10 Minutes
```

Lấy các mục từ bộ nhớ cache

Phương thức `get` của facade `Cache` được sử dụng để truy xuất các mục từ bộ nhớ cache. Nếu mục không tồn tại trong bộ nhớ cache, `null` sẽ được trả lại. Nếu muốn, bạn có thể truyền đối số thứ hai vào phương thức `get` để chỉ định giá trị mặc định mà bạn muốn được trả lại nếu mục không tồn tại:

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

$value = Cache::get('key', 'default');
```

Bạn thậm chí có thể truyền một hàm nặc danh làm giá trị mặc định. Kết quả của hàm này sẽ được trả về nếu mục được chỉ định không tồn tại trong bộ đệm. Việc truyền một hàm nặc danh sẽ cho phép bạn trì hoãn việc truy xuất các giá trị mặc định từ cơ sở dữ liệu hoặc dịch vụ bên ngoài khác:

```php
$value = Cache::get('key', function () {
  return DB::table(...)->get();
});
```

Kiểm tra sự tồn tại của chỉ mục

Phương thức `has` có thể được sử dụng để xác định xem một mục có tồn tại trong bộ nhớ cache hay không. Phương thức này cũng sẽ trả về `false` nếu mục tồn tại nhưng giá trị của nó là `null`:

```php
if (Cache::has('key')) {
  //
}
```

Tăng/Giảm giá trị

Phương thức `increment` và `decrement` có thể được sử dụng để điều chỉnh giá trị của các mục số nguyên trong bộ nhớ cache. Cả hai phương pháp này đều chấp nhận đối số thứ hai tùy chọn cho biết số lượng để tăng hoặc giảm giá trị của mục:

```php
Cache::increment('key');
Cache::increment('key', $amount);
Cache::decrement('key');
Cache::decrement('key', $amount);
```

Lấy và lưu trữ

Đôi khi bạn có thể muốn truy xuất một mục từ bộ nhớ cache, nhưng cũng lưu trữ một giá trị mặc định nếu mục được yêu cầu không tồn tại. Ví dụ: bạn có thể muốn truy xuất tất cả người dùng từ bộ đệm hoặc nếu họ không tồn tại, thì sẽ truy xuất họ từ cơ sở dữ liệu và thêm họ vào bộ đệm. Bạn có thể thực hiện việc này bằng phương thức `Cache::remember`:

```php
$value = Cache::remember('users', $seconds, function () {
  return DB::table('users')->get();
});
```

Nếu mục không tồn tại trong bộ đệm, thì hàm nặc danh được truyền cho phương thức `remember` sẽ được thực thi và kết quả của nó sẽ được đặt trong bộ đệm.

Bạn có thể sử dụng phương thức `rememberForever` này để truy xuất một mục từ bộ nhớ cache hoặc lưu trữ nó mãi mãi nếu nó không tồn tại:

```php
$value = Cache::rememberForever('users', function () {
  return DB::table('users')->get();
});
```

Truy xuất và xóa

Nếu bạn cần truy xuất một mục từ bộ nhớ cache và sau đó xóa mục đó, bạn có thể sử dụng phương thức `pull`. Giống như phương thức `get`, `null` sẽ được trả về nếu mục không tồn tại trong bộ nhớ cache:

```php
$value = Cache::pull('key');
```

Lưu trữ các mục trong bộ nhớ cache

Bạn có thể sử dụng phương thức `put` trên facade `Cache` để lưu trữ các mục trong bộ nhớ cache:

```php
Cache::put('key', 'value', $seconds = 10);
```

Nếu thời gian lưu trữ không được truyền cho phương thức `put`, mục sẽ được lưu trữ vô thời hạn:

```php
Cache::put('key', 'value');
```

Thay vì truyền số giây dưới dạng số nguyên, bạn cũng có thể truyền một đối tượng `DateTime` đại diện cho thời gian hết hạn mong muốn của mục được lưu trong bộ nhớ cache:

```
Cache::put('key', 'value', now()->addMinutes(10));
```

Lưu trữ nếu không tồn tại

Phương thức `add` này sẽ chỉ thêm mục vào bộ đệm nếu nó chưa tồn tại trong bộ nhớ đệm. Phương thức sẽ trả về `true` nếu mục thực sự được thêm vào bộ nhớ cache. Nếu không, phương thức sẽ trả về `false`. Phương thức `add` này là một phép toán nguyên tử:

```
Cache::add('key', 'value', $seconds);
```

Lưu trữ vĩnh viễn các mục

Phương thức `forever` này có thể được sử dụng để lưu trữ vĩnh viễn một mục trong bộ nhớ cache. Vì các mục này sẽ không hết hạn, chúng phải được xóa thủ công khỏi bộ nhớ cache bằng phương pháp `forget`:

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

> Nếu bạn đang sử dụng driver Memcached, các mục được lưu trữ "vĩnh viễn" có thể bị xóa khi bộ nhớ cache đạt đến giới hạn kích thước.

Xóa các mục khỏi bộ nhớ cache

Bạn có thể xóa các mục khỏi bộ nhớ cache bằng phương thức `forget`:

```
Cache::forget('key');
```

Bạn cũng có thể xóa các mục bằng cách cung cấp số giây hết hạn bằng 0 hoặc âm:

```
Cache::put('key', 'value', 0);
```

```
Cache::put('key', 'value', -5);
```

Bạn có thể xóa toàn bộ bộ nhớ cache bằng phương thức `flush`:

```
Cache::flush();
```

> **Chú ý:** Việc xóa bộ nhớ cache không quan trọng "prefix" bộ nhớ cache đã cấu hình của bạn và nó sẽ xóa tất cả các mục nhập khỏi bộ nhớ cache. Hãy xem xét điều này cẩn thận khi xóa bộ nhớ cache được chia sẻ bởi các ứng dụng khác.

Hàm `cache`

Ngoài việc sử dụng facade `Cache`, bạn cũng có thể sử dụng hàm `cache` để truy xuất và lưu trữ dữ liệu qua bộ nhớ cache. Khi hàm `cache` được gọi với một đối số chuỗi, nó sẽ trả về giá trị của key đã cho:

```
$value = cache('key');
```

Nếu bạn cung cấp một mảng các cặp khóa/giá trị (key/value) và thời gian hết hạn cho hàm, nó sẽ lưu trữ các giá trị trong bộ nhớ cache trong khoảng thời gian được chỉ định:

```
cache(['key' => 'value'], $seconds);

cache(['key' => 'value'], now()->addMinutes(10));
```

Khi hàm `cache` được gọi mà không có bất kỳ đối số nào, nó sẽ trả về một đối tượng của `Illuminate\Contracts\Cache\Factory`, cho phép bạn gọi các phương thức lưu vào bộ nhớ đệm khác:

```
cache()->remember('users', $seconds, function () {
  return DB::table('users')->get();
});
```

> Khi việc thử nghiệm testing gọi đến hàm `cache`, thì bạn có thể sử dụng phương

`Cache::shouldReceive` này giống như khi bạn đang test facade.

Cache Tags

Chú ý: Tag bộ nhớ cache không được hỗ trợ khi sử dụng file, dynamodb hoặc database làm driver bộ nhớ cache. Hơn nữa, khi sử dụng nhiều tag với bộ nhớ đệm được lưu trữ "forever", hiệu suất sẽ tốt nhất với một driver như memcached, nó sẽ tự động xóa các record cũ.

Lưu trữ các mục được gắn thẻ trong bộ nhớ đệm

Thẻ bộ nhớ cache cho phép bạn gắn thẻ các mục có liên quan trong bộ nhớ cache và sau đó xóa tất cả các giá trị được lưu trong bộ nhớ cache đã được gán một thẻ nhất định. Bạn có thể truy cập bộ nhớ cache được gắn thẻ bằng cách truyền vào một dãy tên thẻ có thứ tự. Ví dụ: hãy truy cập vào một bộ nhớ cache được gắn thẻ và `put` một giá trị vào bộ nhớ cache:

```
Cache::tags(['people', 'artists'])->put('John', $john, $seconds);

Cache::tags(['people', 'authors'])->put('Anne', $anne, $seconds);
```

Truy cập các mục trong bộ nhớ đệm được gắn thẻ

Để truy xuất một mục bộ nhớ cache được gắn thẻ, hãy truyền một danh sách các thẻ được sắp xếp theo thứ tự cho phương thức `tags` và sau đó gọi phương thức `get` bằng key mà bạn muốn truy xuất:

```
$john = Cache::tags(['people', 'artists'])->get('John');

$anne = Cache::tags(['people', 'authors'])->get('Anne');
```

Xóa các mục trong bộ nhớ đệm được gắn thẻ

Bạn có thể xóa tất cả các mục được gán thẻ hoặc danh sách thẻ. Ví dụ: câu lệnh này sẽ xóa tất cả các bộ nhớ đệm được gắn thẻ hoặc hoặc cả hai `people` và `authors`. Vì vậy, cả hai `Anne` và `John` sẽ bị xóa khỏi bộ nhớ cache:

```
Cache::tags(['people', 'authors'])->flush();
```

Ngược lại, câu lệnh này sẽ chỉ xóa các giá trị được lưu trong bộ nhớ đệm được gắn thẻ `authors`, vì vậy `Anne` sẽ bị xóa, nhưng không xóa `John`:

```
Cache::tags('authors')->flush();
```

Lock nguyên tử

> **Chú ý:** Để sử dụng tính năng này, ứng dụng của bạn phải sử dụng trình điều khiển bộ nhớ cache `memcached`,,, hoặc làm trình điều khiển bộ nhớ cache mặc định của ứng dụng của bạn. Ngoài ra, tất cả các máy chủ phải được giao tiếp với cùng một máy chủ bộ đệm trung tâm. `redis dynamodb database file array`.

Điều kiện tiên quyết về trình điều khiển

Cơ sở dữ liệu

Khi sử dụng `database` làm driver cho bộ nhớ cache, bạn sẽ cần thiết lập một bảng để chứa các key bộ nhớ cache của ứng dụng. Bạn sẽ tìm thấy một `Schema` khai báo cho bảng như ví dụ dưới đây:

```
Schema::create('cache_locks', function ($table) {
  $table->string('key')->primary();
  $table->string('owner');
  $table->integer('expiration');
});
```

Quản lý lock

Lock nguyên tử cho phép thao tác với các lock phân tán mà không cần lo lắng về xung đột dữ liệu. Ví dụ, Laravel Forge sử dụng lock nguyên tử để đảm bảo rằng chỉ có một tác vụ từ xa đang được thực thi trên một máy chủ tại một thời điểm. Bạn có thể tạo và quản lý lock bằng phương thức `Cache::lock`:

```php
use Illuminate\Support\Facades\Cache;
$lock = Cache::lock('foo', 10);

if ($lock->get()) {
  // Lock acquired for 10 seconds...

  $lock->release();
}
```

Phương thức `get` cũng chấp nhận một hàm nặc danh. Sau khi hàm nặc danh được thực hiện, Laravel sẽ tự động giải phóng lock:

```php
Cache::lock('foo')->get(function () {
  // Lock acquired indefinitely and automatically released...
});
```

Nếu lock không khả dụng tại thời điểm bạn yêu cầu, bạn có thể cho Laravel đợi trong một số giây cụ thể. Nếu không thể nhận được lock trong thời hạn đã chỉ định, một exception như `Illuminate\Contracts\Cache\LockTimeoutException` sẽ được ném ra:

```php
use Illuminate\Contracts\Cache\LockTimeoutException;
$lock = Cache::lock('foo', 10);

try {
  $lock->block(5);

  // Lock acquired after waiting a maximum of 5 seconds...
} catch (LockTimeoutException $e) {
  // Unable to acquire lock...
} finally {
  optional($lock)->release();
}
```

Ví dụ trên có thể được đơn giản hóa bằng cách truyền một hàm nặc danh cho phương thức `block`. Khi một hàm nặc danh được truyền cho phương thức này, Laravel sẽ cố gắng lấy lock trong số giây đã chỉ định và sẽ tự động giải phóng lock khi quá trình đóng đã được

thực hiện:

```
Cache::lock('foo', 10)->block(5, function () {
  // Lock acquired after waiting a maximum of 5 seconds...
});
```

Quản lý lock xuyên xuốt quá trình

Đôi khi, bạn có thể muốn có được một lock trong một quy trình và giải phóng nó trong một quy trình khác. Ví dụ: bạn có thể nhận được một lock trong khi web request và muốn mở lock khi kết thúc công việc đã xếp hàng được kích hoạt bởi request đó. Trong trường hợp này, bạn nên truyền "owner token" trong phạm vi của lock cho công việc được xếp hàng đợi để công việc có thể khởi tạo lại lock bằng cách sử dụng mã đã cho.

Trong ví dụ dưới đây, chúng ta sẽ gửi một công việc được xếp hàng đợi nếu lock được đặt thành công. Ngoài ra, chúng ta sẽ truyền owner token của lock cho công việc được xếp hàng đợi thông qua phương thức `owner` của lock:

```
$podcast = Podcast::find($id);
$lock = Cache::lock('processing', 120);

if ($lock->get()) {
  ProcessPodcast::dispatch($podcast, $lock->owner());
}
```

Trong job có tên `ProcessPodcast` bên trong ứng dụng của chúng ta, chúng ta có thể khôi phục và giải phóng lock bằng cách sử dụng owner token:

```
Cache::restoreLock('processing', $this->owner)->release();
```

Nếu bạn muốn mở lock mà không quan trọng chủ sở hữu hiện tại của nó, bạn có thể sử dụng phương thức `forceRelease`:

```
Cache::lock('processing')->forceRelease();
```

Thêm driver tự tạo cho bộ nhớ cache

Viết driver

Để tạo driver tự tạo cho bộ nhớ cache của chúng ta, trước tiên chúng ta cần thực thi hợp đồng `Illuminate\Contracts\Cache\Store` contract. Vì vậy, triển khai bộ nhớ cache MongoDB có thể giống như sau:

```php
<?php
namespace App\Extensions;
use Illuminate\Contracts\Cache\Store;

class MongoStore implements Store
{
  public function get($key) {}
  public function many(array $keys) {}
  public function put($key, $value, $seconds) {}
  public function putMany(array $values, $seconds) {}
  public function increment($key, $value = 1) {}
  public function decrement($key, $value = 1) {}
  public function forever($key, $value) {}
  public function forget($key) {}
  public function flush() {}
  public function getPrefix() {}
}
```

Chúng ta chỉ cần triển khai từng phương pháp này bằng cách sử dụng kết nối MongoDB. Để biết ví dụ về cách triển khai từng phương thức này, hãy đọc mã nguồn của framework Laravel `Illuminate\Cache\MemcachedStore`. Khi quá trình triển khai của chúng ta hoàn tất, chúng ta có thể hoàn tất đăng ký driver tự tạo của mình bằng cách gọi phương thức của facade `Cache::extend`.

```php
Cache::extend('mongo', function ($app) {
  return Cache::repository(new MongoStore);
});
```

> Nếu bạn đang tự hỏi nơi đặt mã driver tự tạo cho cache của mình, thì bạn có thể tạo một namespace `Extensions` trong thư mục *app/* của mình. Tuy nhiên, hãy nhớ rằng Laravel không có cấu trúc ứng dụng cứng nhắc và bạn có thể tự do sắp xếp ứng dụng của mình theo sở thích của mình.

Đăng ký driver

Để đăng ký driver cache tự tạo với Laravel, chúng ta sẽ sử dụng phương thức `extend` trên facade `Cache`. Vì các nhà cung cấp dịch vụ khác có thể cố gắng đọc các giá trị được lưu trong bộ nhớ cache trong phương thức `boot` của chúng, chúng ta sẽ đăng ký driver tự tạo của mình trong một callback `booting`. Bằng cách sử dụng callback `booting`, chúng ta có thể đảm bảo rằng driver tự tạo được đăng ký ngay trước khi phương thức `boot` được gọi bởi service provider của ứng dụng của chúng ta nhưng sau khi phương thức `register` được gọi bởi tất cả các service provider. Chúng ta sẽ đăng ký callback `booting` của chúng ta trong phương thức `register` của class `App\Providers\AppServiceProvider` trong ứng dụng của chúng ta:

```php
<?php

namespace App\Providers;

use App\Extensions\MongoStore;
use Illuminate\Support\Facades\Cache;
use Illuminate\Support\ServiceProvider;

class CacheServiceProvider extends ServiceProvider
{
  /**
   * Register any application services.
   *
   * @return void
   */
  public function register()
  {
    $this->app->booting(function () {
      Cache::extend('mongo', function ($app) {
        return Cache::repository(new MongoStore);
      });
```

```
    });
  }

  /**
   * Bootstrap any application services.
   *
   * @return void
   */
  public function boot()
  {
    //
  }
}
```

Đối số đầu tiên được truyền cho phương thức `extend` là tên của driver. Điều này sẽ tương ứng với driver tự chọn của bạn trong tập tin cấu hình *config/cache.php*. Đối số thứ hai là một hàm nặc danh sẽ trả về một đối tượng `Illuminate\Cache\Repository`. Hàm nặc danh sẽ được truyền vào đối tượng `$app`, là một đối tượng của service container.

Sau khi tiện ích mở rộng của bạn được đăng ký, trong tập tin cấu hình *config/cache.php* hãy cập nhật tùy chọn `driver` thành tên của tiện ích mở rộng của bạn.

Các sự kiện

Để thực thi mã code trên mọi hoạt động của bộ đệm, bạn có thể theo dõi các sự kiện do bộ đệm kích hoạt. Thông thường, bạn nên đặt các chương trình theo dõi sự kiện này trong class `App\Providers\EventServiceProvider` của ứng dụng của mình:

```
/**
 * The event listener mappings for the application.
 *
 * @var array
 */
protected $listen = [
  'Illuminate\Cache\Events\CacheHit' => [
    'App\Listeners\LogCacheHit',
  ],
```

```
    'Illuminate\Cache\Events\CacheMissed' => [
        'App\Listeners\LogCacheMissed',
    ],

    'Illuminate\Cache\Events\KeyForgotten' => [
        'App\Listeners\LogKeyForgotten',
    ],

    'Illuminate\Cache\Events\KeyWritten' => [
        'App\Listeners\LogKeyWritten',
    ],
];
```