Course - Laravel Framework

Redis

Redis là một kho lưu trữ dữ liệu khóa-giá trị nâng cao, mã nguồn mở. Nó thường được gọi là máy chủ cấu trúc dữ liệu vì các khóa có thể chứa chuỗi, mã băm, danh sách, tập hợp và tập hợp được xếp thứ tự.

Tags: Laravel, Redis

Giới thiệu

Redis là một kho lưu trữ dữ liệu khóa-giá trị nâng cao, mã nguồn mở. Nó thường được gọi là máy chủ cấu trúc dữ liệu vì các khóa có thể chứa chuỗi, mã băm, danh sách, tập hợp và tập hợp được xếp thứ tự.

Trước khi sử dụng Redis với Laravel, chúng tôi khuyến khích bạn cài đặt và sử dụng phần mở rộng phpredis PHP thông qua PECL. Phần mở rộng sẽ phức tạp khi cài đặt hơn so với các gói PHP "user land" nhưng có thể mang lại hiệu suất tốt hơn cho các ứng dụng sử dụng Redis. Nếu bạn đang sử dụng Laravel Sail, tiện ích mở rộng này đã được cài đặt trong container Docker của ứng dụng của bạn.

Nếu bạn không thể cài đặt phần mở rộng phpredis, bạn có thể cài đặt gói *predis/predis* thông qua Composer. Predis là một ứng dụng khách Redis được viết hoàn toàn bằng PHP và không yêu cầu bất kỳ phần mở rộng bổ sung nào:

```
composer require predis/predis
```

Cấu hình

Bạn có thể cấu hình cài đặt Redis của ứng dụng của mình thông qua tập tin cấu hình *config/database.php*. Trong tập tin này, bạn sẽ thấy một mảng redis chứa các máy chủ Redis được ứng dụng của bạn sử dụng:

```
'redis' => [

    'client' => env('REDIS_CLIENT', 'phpredis'),

    'default' => [
        'host' => env('REDIS_HOST', '127.0.0.1'),
        'password' => env('REDIS_PASSWORD', null),
        'port' => env('REDIS_PORT', 6379),
        'database' => env('REDIS_DB', 0),
    ],

    'cache' => [
        'host' => env('REDIS_HOST', '127.0.0.1'),
        'password' => env('REDIS_PASSWORD', null),
        'port' => env('REDIS_PORT', 6379),
```

```
    'database' => env('REDIS_CACHE_DB', 1),
  ],
],
```

Mỗi máy chủ Redis được xác định trong tập tin cấu hình của bạn bắt buộc phải có tên, máy chủ lưu trữ và cổng trừ khi bạn xác định một URL duy nhất để đại diện cho kết nối Redis:

```
'redis' => [
  'client' => env('REDIS_CLIENT', 'phpredis'),

  'default' => [
    'url' => 'tcp://127.0.0.1:6379?database=0',
  ],

  'cache' => [
    'url' => 'tls://user:password@127.0.0.1:6380?database=1',
  ],
],
```

Cấu hình schema

Theo mặc định, các máy khách Redis sẽ sử dụng schema `tcp` khi kết nối với máy chủ Redis của bạn; tuy nhiên, bạn có thể sử dụng mã hóa TLS/SSL bằng cách chỉ định tùy chọn cấu hình `scheme` trong mảng cấu hình máy chủ Redis của bạn:

```
'redis' => [
  'client' => env('REDIS_CLIENT', 'phpredis'),

  'default' => [
    'scheme' => 'tls',
    'host' => env('REDIS_HOST', '127.0.0.1'),
    'password' => env('REDIS_PASSWORD', null),
    'port' => env('REDIS_PORT', 6379),
    'database' => env('REDIS_DB', 0),
  ],
],
```

Cluster

Nếu ứng dụng của bạn đang sử dụng một cụm máy chủ Redis, bạn nên xác định các cụm này trong một khóa cụm của cấu hình Redis của bạn. Khóa cấu hình này không tồn tại theo mặc định, vì vậy bạn sẽ cần phải tạo nó trong tập tin cấu hình *config/database.php* của ứng dụng:

```
'redis' => [
  'client' => env('REDIS_CLIENT', 'phpredis'),

  'clusters' => [
    'default' => [
      [
        'host' => env('REDIS_HOST', 'localhost'),
        'password' => env('REDIS_PASSWORD', null),
        'port' => env('REDIS_PORT', 6379),
        'database' => 0,
      ],
    ],
  ],
],
```

Theo mặc định, cluster sẽ thực hiện sharding phía máy khách trên các node của bạn, cho phép bạn gộp các node và tạo ra một lượng lớn RAM có sẵn. Tuy nhiên, sharding phía máy khách không xử lý chuyển đổi dự phòng; do đó, nó chủ yếu phù hợp với dữ liệu được lưu trong bộ nhớ cache tạm thời có sẵn từ một kho dữ liệu chính khác.

Nếu bạn muốn sử dụng cluster Redis gốc thay vì sharding phía máy khách, bạn có thể chỉ định điều này bằng cách đặt giá trị cấu hình `options.cluster` thành redis trong tập tin cấu hình *config/database.php* của ứng dụng của bạn:

```
'redis' => [
  'client' => env('REDIS_CLIENT', 'phpredis'),

  'options' => [
    'cluster' => env('REDIS_CLUSTER', 'redis'),
  ],
```

```
  'clusters' => [
    // ...
  ],
],
```

Predis

Nếu bạn muốn ứng dụng của mình tương tác với *Redis* thông qua gói *Predis*, bạn nên đảm bảo giá trị của biến môi trường `REDIS_CLIENT` là giá trị `predis`:

```
'redis' => [
  'client' => env('REDIS_CLIENT', 'predis'),

  // ...
],
```

Ngoài các tùy chọn cấu hình mặc định như host, port, database và mật khẩu, Predis hỗ trợ các tham số kết nối bổ sung có thể được xác định cho từng máy chủ Redis của bạn. Để sử dụng các tùy chọn cấu hình bổ sung này, hãy thêm chúng vào cấu hình máy chủ Redis của bạn trong tập tin cấu hình *config/database.php* của ứng dụng:

```
'default' => [
  'host' => env('REDIS_HOST', 'localhost'),
  'password' => env('REDIS_PASSWORD', null),
  'port' => env('REDIS_PORT', 6379),
  'database' => 0,
  'read_write_timeout' => 60,
],
```

Bí danh của facade Redis

Tập tin cấu hình *config/app.php* của Laravel chứa một mảng bí danh xác định tất cả các bí danh class sẽ được đăng ký bởi khung công tác. Để thuận tiện, một mục bí danh được bao gồm cho mỗi facade do Laravel cung cấp; tuy nhiên, bí danh Redis bị vô hiệu hóa vì nó xung đột với tên class Redis do phần mở rộng phpredis cung cấp. Nếu bạn đang sử dụng ứng dụng Predis và muốn bật bí danh này, bạn có thể bỏ nhận xét bí danh đó trong tập tin

cấu hình *config/app.php* của ứng dụng.

phpredis

Theo mặc định, Laravel sẽ sử dụng phần mở rộng `phpredis` để giao tiếp với Redis. Máy khách mà Laravel sẽ sử dụng để giao tiếp với Redis được quy định bởi giá trị của tùy chọn cấu hình `redis.client`, thường phản ánh giá trị của biến môi trường `REDIS_CLIENT`:

```
'redis' => [
  'client' => env('REDIS_CLIENT', 'phpredis'),

  // Rest of Redis configuration...
],
```

Ngoài các tùy chọn cấu hình mặc định như `scheme`, `host`, `port`, `database` và `password`, phpredis hỗ trợ các tham số kết nối bổ sung sau: `name`, `persistent`, `persistent_id`, `prefix`, `read_timeout`, `retry_interval`, `timeout` và `context`. Bạn có thể thêm bất kỳ tùy chọn nào trong số này vào cấu hình máy chủ Redis của mình trong tập tin cấu hình *config/database.php*:

```
'default' => [
  'host' => env('REDIS_HOST', 'localhost'),
  'password' => env('REDIS_PASSWORD', null),
  'port' => env('REDIS_PORT', 6379),
  'database' => 0,
  'read_timeout' => 60,
  'context' => [
    // 'auth' => ['username', 'secret'],
    // 'stream' => ['verify_peer' => false],
  ],
],
```

Các thuật toán nén trong phpredis

Phần mở rộng `phpredis` cũng có thể được cấu hình để sử dụng nhiều thuật toán nén. Các thuật toán này có thể được cấu hình thông qua mảng `options` của cấu hình Redis của bạn:

```
use Redis;

'redis' => [
  'client' => env('REDIS_CLIENT', 'phpredis'),

  'options' => [
    'serializer' => Redis::SERIALIZER_MSGPACK,
    'compression' => Redis::COMPRESSION_LZ4,
  ],

  // Rest of Redis configuration...
],
```

Các thuật toán serialization được hỗ trợ bao gồm:

- `Redis::SERIALIZER_NONE` (mặc định),
- `Redis::SERIALIZER_PHP`,
- `Redis::SERIALIZER_JSON`,
- `Redis::SERIALIZER_IGBINARY`,
- `Redis::SERIALIZER_MSGPACK`.

Các thuật toán nén được hỗ trợ bao gồm:

- `Redis::COMPRESSION_NONE` (mặc định),
- `Redis::COMPRESSION_LZF`,
- `Redis::COMPRESSION_ZSTD`
- `Redis::COMPRESSION_LZ4`.

Tương tự với Redis

Bạn có thể tương tác với Redis bằng cách gọi nhiều phương thức khác nhau trên facade `Redis`. Facade `Redis` hỗ trợ các phương thức động, có nghĩa là bạn có thể gọi bất kỳ lệnh Redis nào trên facade và lệnh sẽ được truyền trực tiếp đến Redis. Trong ví dụ này, chúng ta sẽ gọi lệnh Redis `GET` bằng cách gọi phương thức `get` của facade `Redis`:

```
<?php

namespace App\Http\Controllers;
```

```php
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Redis;

class UserController extends Controller
{
  /**
   * Show the profile for the given user.
   *
   * @param  int  $id
   * @return \Illuminate\Http\Response
   */
  public function show($id)
  {
    return view('user.profile', [
      'user' => Redis::get('user:profile:'.$id)
    ]);
  }
}
```

Như đã đề cập ở trên, bạn có thể gọi bất kỳ lệnh nào của *Redis* trên facade `Redis`. Laravel sử dụng các phương thức magic để truyền các lệnh đến máy chủ Redis. Nếu một lệnh Redis yêu cầu các đối số, bạn nên truyền các đối số đó đến phương thức tương ứng của facade:

```php
use Illuminate\Support\Facades\Redis;

Redis::set('name', 'Taylor');

$values = Redis::lrange('names', 5, 10);
```

Ngoài ra, bạn có thể truyền các lệnh đến máy chủ bằng cách sử dụng phương thức `command` của `Redis`, phương thức này chấp nhận tên của lệnh làm đối số đầu tiên và một mảng giá trị làm đối số thứ hai của nó:

```php
$values = Redis::command('lrange', ['name', 5, 10]);
```

Sử dụng nhiều kết nối Redis

Tập tin cấu hình *config/database.php* của ứng dụng cho phép bạn xác định nhiều kết nối hoặc máy chủ Redis. Bạn có thể nhận được kết nối tới một kết nối Redis cụ thể bằng cách sử dụng phương thức `connection` của `Redis`:

```
$redis = Redis::connection('connection-name');
```

Để có được một đối tượng của kết nối Redis mặc định, bạn có thể gọi phương thức connection mà không có bất kỳ đối số bổ sung nào:

```
$redis = Redis::connection();
```

Các transaction

Phương thức `transaction` của facade Redis cung cấp một wrapper thuận tiện bao quanh các lệnh `MULTI` và `EXEC` gốc của Redis. Phương thức `transaction` chấp nhận một hàm xử lý làm đối số duy nhất của nó. Hàm này sẽ nhận được một đối tượng kết nối Redis và có thể đưa ra bất kỳ lệnh nào nó muốn cho đối tượng này. Tất cả các lệnh Redis được đưa ra trong hàm này sẽ được thực hiện trong một giao dịch nguyên tử duy nhất:

```
use Illuminate\Support\Facades\Redis;

Redis::transaction(function ($redis) {
  $redis->incr('user_visits', 1);
  $redis->incr('total_visits', 1);
});
```

> **Chú ý:** Khi tạo một transaction Redis, bạn không thể truy xuất bất kỳ giá trị nào từ kết nối Redis. Hãy nhớ rằng, giao dịch transaction của bạn được thực hiện dưới dạng một hoạt động nguyên tử và hoạt động đó không được thực hiện cho đến khi toàn bộ hàm xử lý của bạn hoàn tất việc thực hiện các lệnh của nó.

Các tập lệnh *Lua*

Phương thức `eval` cung cấp một phương thức khác để thực hiện nhiều lệnh Redis trong

một hoạt động đơn lẻ, nguyên tử. Tuy nhiên, phương thức `eval` có lợi ích là có thể tương tác và kiểm tra các giá trị khóa của Redis trong quá trình hoạt động đó. Các tập lệnh Redis được viết bằng ngôn ngữ lập trình Lua.

Lúc đầu, phương thức `eval` có thể hơi đáng sợ, nhưng chúng ta sẽ khám phá một ví dụ cơ bản để phá băng. Phương thức `eval` mong đợi một số đối số. Đầu tiên, bạn nên truyền tập lệnh Lua (dưới dạng một chuỗi) vào phương thức. Thứ hai, bạn nên truyền số lượng khóa (dưới dạng số nguyên) mà tập lệnh sẽ tương tác. Thứ ba, bạn nên truyền tên của các khóa đó. Cuối cùng, bạn có thể truyền bất kỳ đối số bổ sung nào khác mà bạn cần truy cập trong tập lệnh của mình.

Trong ví dụ này, chúng ta sẽ tăng bộ đếm, kiểm tra giá trị mới của nó và tăng bộ đếm thứ hai nếu giá trị của bộ đếm đầu tiên lớn hơn năm. Cuối cùng, chúng ta sẽ trả về giá trị của bộ đếm đầu tiên:

```
$value = Redis::eval(<<<'LUA'
  local counter = redis.call("incr", KEYS[1])

  if counter > 5 then
    redis.call("incr", KEYS[2])
  end

  return counter
LUA, 2, 'first-counter', 'second-counter');
```

Chú ý: Vui lòng tham khảo tài liệu Redis để biết thêm thông tin về tập lệnh Redis.

Các lệnh Pipelining

Đôi khi bạn có thể cần thực hiện hàng chục lệnh Redis. Thay vì thực hiện một đường truyền mạng đến máy chủ Redis của bạn cho mỗi lệnh, bạn có thể sử dụng phương thức `pipeline`. Phương thức `pipeline` chấp nhận một đối số: một hàm nhận một đối tượng Redis. Bạn có thể đưa ra tất cả các lệnh của mình cho đối tượng Redis này và tất cả chúng sẽ được gửi đến máy chủ Redis cùng một lúc để giảm các đường truyền mạng đến máy chủ. Các lệnh sẽ vẫn được thực hiện theo thứ tự mà chúng đã được đưa ra:

```
use Illuminate\Support\Facades\Redis;
```

```
Redis::pipeline(function ($pipe) {
  for ($i = 0; $i < 1000; $i++) {
    $pipe->set("key:$i", $i);
  }
});
```

Pub/Sub

Laravel cung cấp một giao diện thuận tiện cho các lệnh `publish` và `subscribe` của Redis. Các lệnh Redis này cho phép bạn theo dõi tin nhắn trên một "channel" nào đó. Bạn có thể xuất bản các tin nhắn lên channel từ một ứng dụng khác hoặc thậm chí sử dụng một ngôn ngữ lập trình khác, cho phép giao tiếp dễ dàng giữa các ứng dụng và quy trình.

Trước tiên, hãy thiết lập chường trình theo dõi channel bằng phương thức `subscribe`. Chúng ta sẽ đặt lệnh gọi phương thức này trong một lệnh Artisan vì việc gọi phương thức `subscribe` bắt đầu một quá trình chạy lâu dài:

```
<?php

namespace App\Console\Commands;

use Illuminate\Console\Command;
use Illuminate\Support\Facades\Redis;

class RedisSubscribe extends Command
{
  /**
   * The name and signature of the console command.
   *
   * @var string
   */
  protected $signature = 'redis:subscribe';

  /**
   * The console command description.
   *
   * @var string
```

```
     */
    protected $description = 'Subscribe to a Redis channel';

    /**
     * Execute the console command.
     *
     * @return mixed
     */
    public function handle()
    {
        Redis::subscribe(['test-channel'], function ($message) {
            echo $message;
        });
    }
}
```

Giờ đây, chúng ta có thể xuất bản tin nhắn lên kênh bằng phương pháp `publish`:

```
use Illuminate\Support\Facades\Redis;

Route::get('/publish', function () {
    // ...

    Redis::publish('test-channel', json_encode([
        'name' => 'Adam Wathan'
    ]));
});
```

Đăng ký ký tự đại diện (wildcard)

Sử dụng phương pháp `psubscribe`, bạn có thể đăng ký một kênh ký tự đại diện wildcard, điều này có thể hữu ích để xem tất cả thông báo trên tất cả các channel. Tên channel sẽ được chuyển làm đối số thứ hai cho hàm xử lý được cung cấp:

```
Redis::psubscribe(['*'], function ($message, $channel) {
    echo $message;
```

```
});

Redis::psubscribe(['users.*'], function ($message, $channel) {
  echo $message;
});
```