Laravel Queue

Tổng quan về Queue

- Hàng đợi cho phép bạn trì hoãn một công việc mất nhiều thời gian đến một thời điểm nào nó mới xử lý.
- Laravel cung cấp một API thống nhất cho rất nhiều các hàng đợi ở backend khác nhau, đó là chính là Laravel Queue

Ví dụ về Queue

Website của bạn có nhiều người vào xem và có thể đăng ký tài khoản, người đó đang ở trang đăng ký và nhập các thông tin vào form đăng ký. Khi đó chúng ta muốn thực hiện các công việc sau:

- Kiểm tra thông tin nhập và lưu vào CSDL.
- Gửi một email Chào mừng đến thành viên mới này.
- Trả về trang Thank you.

Sau khi kiểm tra và lưu thông tin vào CSDL, tiếp tục đến phần việc gửi email do mã PHP thực hiện tuần tự từ trên xuống. Người dùng sẽ thấy trang Thank you nếu email đã được gửi đi, quá trình gửi email có thể nhanh nhưng cũng có thể mất thời gian, vậy tại sao phải bắt người dùng chờ? Laravel Queue sẽ là cứu cánh cho tình huống này

Laravel Queue là gì?

- Một hàng đợi (queue) là một danh sách những việc cần làm (job) được quản lý theo thứ tự
- Khi chúng ta muốn thêm một công việc (job) vào hàng đợi, job phải implement interface `Illuminate\Contracts\Queue\ShouldQueue`
- Laravel Queue driver được sử dụng để quản lý các job như thêm job vào hàng đợi, lấy job ra khỏi hàng đợi
- Laravel có thể làm việc với nhiều các driver khác nhau như database, Redis, Amazon SQS... và bạn có thể tự tạo riêng một driver nếu muốn

Trong phạm vi của khoá học này, tôi hướng dẫn lưu trữ queue ở Database

Laravel Queue Migrate

Để Laravel tự động tạo các table lưu trữ queue, bạn chạy các câu lệnh sau

```
php artisan queue:table
```

```
php artisan queue:failed-table
```

```
php artisan migrate
```

Các câu lệnh này sẽ tạo ra bảng jobs và failed_jobs trong cơ sở dữ liệu. Ngoài ra, cần phải thiết lập trong file `.env`

```
QUEUE_DRIVER=database
```

Tạo và thêm Job vào queue

Mặc định, các job được lưu trong app\Jobs, nếu thư mục app\Jobs không có trong project bạn cũng đừng lo, câu lệnh tạo job sẽ tự động tạo ra thư mục này nếu chưa có. Thực hiện tạo một job mới bằng câu lệnh:

```
php artisan make:job SendWelcomeEmail
```

Nó sẽ tự động sinh ra job SendWelcomeEmail được implement interface `Illuminate\Contracts\Queue\ShouldQueue`.

Class này chứa phương thức `handle()` sẽ được gọi đến khi job được xử lý trong hàng đợi.

Chúng ta hãy xem khung của một job:

```php
<?php

namespace App\Jobs;

use App\User;
use App\AudioProcessor;
use Illuminate\Bus\Queueable;
use Illuminate\Queue\SerializesModels;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Contracts\Queue\ShouldQueue;

class SendWelcome Email implements ShouldQueue
{
    use InteractsWithQueue, Queueable, SerializesModels;

    protected $user;

    /**
     * Create a new job instance.
     *
     * @param  User  $user
     * @return void
     */
    public function __construct(User $user)
    {
        $this->user = $user;
    }

    /**
     * Execute the job.
     *
     * @param  AudioProcessor  $processor
     * @return void
     */
    public function handle(AudioProcessor $processor)
    {
        // Process uploaded user...
    }
}
```

Trong ví dụ trên chúng ta truyền một Eloquent Model User vào phương thức `__construct` của job. Để thêm job vào một queue, sử dụng phương thức dispatch():

```php
<?php
namespace App\Http\Controllers\Auth;
use App\User;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Validator;
use Illuminate\Foundation\Auth\RegistersUsers;
use App\Jobs\SendWelcomeEmail;
class RegisterController extends Controller
{
    ...
```

```
    protected function create(array $data)
    {
        $user = User::create([
            'name' => $data['name'],
            'email' => $data['email'],
            'password' => bcrypt($data['password']),
        ]);
        $job = (new SendWelcomeEmail($user))->delay(Carbon::now()->addMinutes(10));
        dispatch($job);
        return $user;
    }
}
```

Phương thức `delay()` sẽ dừng lại trước khi thực hiện job trong queue.

Thực thi các jobs trong queue

Số lần thử thực hiện job

Mặc định job sẽ được thực hiện 1 lần, nếu lỗi sẽ được bỏ qua, để thiết lập số lần thử thực hiện lại một job chúng ta có hai cách: hoặc sử dụng câu lệnh artisan cho tất cả các job

```
php artisan queue:work --tries=3
```

hoặc đưa vào thuộc tính $tries của từng job

```
<?php
namespace App\Jobs;

class SendWelcome implements ShouldQueue
{
    /**
     * Số lần job sẽ thử thực hiện lại
     *
     * @var int
     */
    public $tries = 3;
}
```

Thiết lập thời gian timeout của job trong queue

Bạn có thể thiết lập thời gian timeout của các job bằng cách sử dụng câu lệnh artisan

```
php artisan queue:work --timeout=60
```

hoặc thiết lập trong thuộc tính `$timeout` của từng job

```
<?php
namespace App\Jobs;

class SendWelcome implements ShouldQueue
{
    /**
     * Số giây job có thể chạy trước khi timeout
     *
     * @var int
     */
    public $timeout = 60;
}
```

Queue worker - thực thi các job trong queue

Laravel có một queue worker để thực thi các job đang có trong hàng đợi, bạn có thể chạy worker này bằng câu lệnh artisan:

```
php artisan queue:work
```

Chú ý, câu lệnh này khi đã thực hiện sẽ chạy cho đến khi đóng cửa sổ dòng lệnh hoặc dừng nó bằng một câu lệnh. Queue worker là các tiến trình có thời gian sống dài do đó nó sẽ không cập nhật code khi có thay đổi, khi bạn thay đổi code chương trình, bạn cần khởi động lại queue worker bằng câu lệnh

```
php artisan queue:restart
```

Thiết lập thời gian nghỉ giữa các lần xử lý job

Các job trong hàng đợi được xử lý liên tục mà không có sự dừng lại nào, tùy chọn sleep sẽ xác định worker dừng lại sau bao lâu trước khi tiếp tục xử lý job tiếp theo:

```
php artisan queue:work --sleep=3
```

Sử dụng Supervisor giám sát xử lý hàng đợi trên Linux

Supervisor là một chương trình giám sát các xử lý trong hệ điều hành Linux, nó sẽ tự động khởi động lại xử lý queue:work nếu bị lỗi. Để cài đặt supervisor trên CentOS trước hết phải cài đặt python

```
yum install python-setuptools
easy_install supervisor
```

Cài đặt supervisor trên Ubuntu

```
sudo apt-get install supervisor
```

Supervisor có file cấu hình nằm trong thư mục `/etc/supervisor/conf.d`, trong này bạn có thể tạo nhiều file để bắt supervisor giám sát các xử lý.

Ví dụ tạo file `laravel-worker.conf` để giám sát cho `queue:worker` với nội dung:

```
[program:laravel-worker]
process_name=%(program_name)s_%(process_num)02d
command=php /home/forge/app.com/artisan queue:work sqs --sleep=3 --tries=3
autostart=true
autorestart=true
user=forge
numprocs=8
redirect_stderr=true
stdout_logfile=/home/forge/app.com/worker.log
```

numprocs = 8 tức là Supervisor sẽ chạy 8 xử lý queue:work cùng lúc và tự động khởi động lại queue:work khi gặp lỗi.

Sau khi thiết lập cấu hình Supervisor xong, bạn phải cập nhật cấu hình và chạy các xử lý bằng các câu lệnh như sau:

```
$supervisorctl reread
$supervisorctl update
$supervisorctl start laravel-worker:*
```

Giám sát xử lý hàng đợi trên Windows

Công cụ supervisor chỉ hoạt động trong môi trường Linux, vậy trên Windows chúng ta sẽ xử lý như thế nào. Chúng ta sẽ sử dụng gói Forever, cài đặt Forever bằng npm. npm được tích hợp sẵn trong bộ cài Node.js, bạn tải về và cài đặt. Sau đó cài đặt Forever:

```
npm install -g forever
```

Sau khi cài đặt Forever, chúng ta có thể dùng nó để giám sát các xử lý

```
forever -c php artisan --queue:work --tries=3 --timeout=60 --sleep=5
```

Laravel Queue trong bài toán xử lý song song

Trong lập trình, hàng đợi giúp cho việc thiết kế các ứng dụng tốt hơn trong hiệu năng xử lý.

Chúng ta đến với ví dụ thực tế sau, trong hệ thống corebanking của ngân hàng, cuối ngày sẽ thực hiện chạy khóa ngày COB (Close Of Bussiness), quá trình này là một tập hợp (các batch) rất nhiều các tác vụ khác nhau như cập nhật thông tin hệ thống, tính toán lãi tiền gửi, tiền vay, chuyển trạng thái các hợp đồng, gửi tin nhắn đến khách hàng và cập nhật thông tin các bảng cần thiết cho ngày mới như bảng lãi suất, tỉ giá...

Nếu tất cả các công việc này được thực hiện tuần tự, quá trình COB sẽ mất rất nhiều thời gian mà hệ thống corebanking chỉ được phép chạy trong khoảng 3-5 tiếng do cần phải có thời gian dự phòng.

Vậy giải pháp nào cho những vấn đề tương tự như vậy? Sử dụng hàng đợi sẽ giúp xử lý những vấn đề như trên. Chúng ta sẽ chia các công việc ra thành các Batch, trong mỗi Batch chứa các job khác nhau liên quan đến một bussiness logic nào đó. Khi đó, chúng ta thực hiện chạy nhiều queue:work sẽ giúp tăng tốc độ lên đáng kể.

Trong ví dụ về sử dụng Supervisor, giả sử cấu hình máy chủ có thể chạy được 30 session một lúc, nếu chúng ta để

```
numprocs=30
```

Tốc độ xử lý sẽ nhanh hơn rất nhiều lần. Laravel Queue rất thích hợp cho các bài toán xử lý nhiều việc song song.