INSTITUT AFRICAIN D'INFORMATIQUE

Représentation du Cameroun

Centre d'Excellence Technologique Paul BIYA

B.P. : 13719 Yaoundé

OUTILS DE PROGRAMMATION WEB

Les Processus Et Les Files d'attente Dans Laravel

Par : MBOUOBOUO Abdel (50%) et EYENGA OTIE (50%)

Année Académique

2024/2025

Sous la supervision de :

M. NZE FOMEKONG

SOMMAIRE

Introduction

Dans le développement d'applications modernes avec Laravel, la gestion efficace des tâches intensives et des opérations en arrière-plan est devenue cruciale. Deux fonctionnalités essentielles de Laravel répondent à ces besoins : les processus (Process) et les files d'attente (Queues). Ces mécanismes, bien que distincts, sont complémentaires et offrent des solutions robustes pour différents cas d'utilisation.

Les processus Laravel permettent d'interagir directement avec les processus système de manière sécurisée et orientée objet. Ils excellent dans l'exécution de commandes système, la gestion de scripts externes et l'interaction avec des outils en ligne de commande. Cette fonctionnalité est particulièrement utile pour des tâches comme la compilation d'assets, l'exécution de commandes git, ou le traitement de fichiers système.

Les files d'attente, quant à elles, constituent le système de traitement asynchrone de Laravel. Elles permettent de différer l'exécution de tâches intensives, améliorant ainsi la réactivité des applications. Ce système est idéal pour des opérations comme l'envoi d'emails, le traitement d'images, la génération de rapports, ou toute autre tâche pouvant être exécutée en arrière-plan.

La compréhension et la maîtrise de ces deux fonctionnalités sont essentielles pour développer des applications Laravel performantes et scalables.

I. Les Processus dans Laravel

Les processus de Laravel permettent d'exécuter et de gérer des processus système directement depuis une application Laravel. Ils reposent sur le composant Symfony Process et offrent une API fluide et intuitive pour interagir avec le système d'exploitation. Grâce aux processus, Laravel facilite l'exécution de commandes shell, de scripts externes et d'autres processus système, tout en permettant de capturer leurs sorties et d'en gérer le comportement.

1. Utilisation des processus

Les processus permettent d'exécuter des commandes système de manière synchrone ou asynchrone. Ils peuvent être utilisés pour :

- Exécuter des scripts Bash, Python, ou autres depuis Laravel.
- Automatiser des tâches système, comme le nettoyage des fichiers temporaires.
- Interagir avec des services externes, par exemple, pour appeler un service CLI.
- Manipuler des fichiers, générer des rapports, ou compresser des données via des commandes shell.

L'utilisation des processus dans Laravel apporte plusieurs avantages :

- Automatisation : Exécution de tâches répétitives sans intervention manuelle.
- Optimisation : Déléguer certaines opérations au système pour de meilleures performances.
- Asynchronisme : Lancer des processus en arrière-plan pour ne pas bloquer l'application.

2. Exécution et appel des Processus

Laravel offre deux types d'exécution des processus : exécution synchrone (processus synchrone) et exécution asynchrone (processus asynchrone).

a. Les processus synchrones

L'exécution est bloquante : Laravel attend que la commande se termine avant de continuer. C'est la méthode `run()` qui est utilisée pour l'exécution du processus après son appel.

Exemple : Appel d'un processus pour lister les éléments du dossier public :

```
$result = Process::run('ls -la'); // appel du processus

echo $result->output(); // affichage du resultat grâce à la méthode output().
```

b. Les processus asynchrones

L'exécution est non bloquante : le script continue son exécution pendant que le processus tourne en arrière-plan. Ici, c'est la méthode `start()` qui est utilisée pour l'exécution du processus.

Exemple : Appel d'un processus pour créer un nouveau projet Laravel :

```
$process = Process::start('composer create-project laravel/laravel test-app'); // appel du processus asynchrone

// $process->running() === true si le processus est toujours en train d'être exécuté

if ($process->running()) {

echo "Le processus est en cours...";

}

echo $process->output(); // affiche le resultat à la fin de l'exécution.
```

3. Méthodes et options des processus

Les processus possèdent plusieurs méthodes et options. Dans cette partie, nous allons présenter quelques méthodes.

- `run()` : Elle est utilisée pour l'exécution des processus synchrones. `run()` prend en argument une chaine de caractère qui représente la commande a exécuté.

- `start()` : Elle est utilisée pour l'exécution des processus asynchrone et prend également une chaine de caractère en argument qui représente la commande a exécuté.

- `successful()` : Vérifie si le processus s'est bien terminé.

- `failed()` : Vérifie si le processus a échoué.

- `output()` : Récupère la sortie standard.

- `errorOutput()` : Récupère la sortie d'erreur.

- `exitCode()` : Récupère le code de sortie.

- `running()` : Vérifie si le processus est en cours.

Laravel permet également de modifier diverses fonctionnalités de processus, telles que le répertoire de travail, une entrée et le délai d'expiration.

- Répertoire de travail : Vous pouvez utiliser la méthode `path()` pour spécifier le répertoire de travail du processus. Si cette méthode n'est pas invoquée, le processus héritera du répertoire de travail du script PHP en cours d'exécution :
  ```
  $result = Process::path('C:\Users\Public')->run('ls -la');
  ```

- Entrée : Vous pouvez fournir des informations via la « saisie standard » du processus en utilisant la méthode `input()` :
  ```
  Process::input('print(5+5)')->run('python')->output();
  ```

- Délais : Par défaut, les processus lèveront une exception de `ProcessTimedOutException` après l'exécution pendant plus de 60 secondes. Cependant, vous pouvez personnaliser ce comportement via la méthode `timeout()` :
  ```
  Process::timeout(120)->run('ls -la')->output();
  ```

II. Les files d'attentes dans Laravel

Les files d'attente (queues) en Laravel permettent de différer l'exécution de tâches longues ou gourmandes en ressources en les traitant en arrière-plan. Cela améliore les performances et la réactivité des applications en évitant de bloquer l'exécution du code principal. Laravel fournit une API fluide pour gérer ces files d'attente, et supporte plusieurs drivers pour stocker et traiter les tâches.

Une file d'attente est un mécanisme qui permet de stocker des tâches à exécuter de manière asynchrone. Les tâches sont ajoutées à la file d'attente et sont exécutées par un worker (un processus qui exécute les tâches) lorsque celui-ci est disponible.

Les files d'attente en Laravel offrent plusieurs avantages, notamment :

- Amélioration de la performance : Les files d'attente permettent de décharger les tâches longues et intensives en ressources du serveur web, ce qui améliore la performance de l'application.

- Scalabilité : Les files d'attente permettent de gérer un grand nombre de tâches simultanément, ce qui rend l'application plus scalable.

- Fiabilité : Les files d'attente permettent de garantir que les tâches sont exécutées même en cas de panne du serveur web.

- Gestion des échecs : Possibilité de réessayer les tâches échouées.

1. Utilisation des files d'attente

Les files d'attente sont utilisées pour exécuter des tâches de manière asynchrone. Par exemple :

- Envoi d'e-mails après inscription d'un utilisateur.

- Traitement de fichiers volumineux (conversion, compression…).

- Génération de rapports sans ralentir l'interface utilisateur.
- Importation/exportation de données massives.
- Interaction avec des API externes pour éviter des temps d'attente longs.

2. Création des files d'attentes et de jobs

a. Drivers

Pour créer une file d'attente dans Laravel, il faut choisir un driver de file d'attente parmi lesquels nous avons :

- `sync` : Il exécute directement les tâches sans les mettre dans une file d'attente. Il est utile pour les tests et ne convient pas aux applications nécessitant un traitement en arrière-plan.
- `database` : Le driver `database` stocke les jobs (tâches) dans une base données et les exécute via le worker Laravel. Pour l'utiliser, il faudra créer la table des jobs avec les commandes : `php artisan queue:table` et ensuite `php artisan migrate`. Pour lancer le worker, utiliser la commande `php artisan queue:work.`
- `redis` : le driver `redis` utilise Redis comme système de file d'attente et est l'un des plus performants. Il est rapide et évolutif et compatible avec Laravel Horizon pour la getion avancée des jobs.
- `sqs` : le driver `sqs` utilise le service Amazon SQS (Simple Queue Service). `sqs` gère automatiquement les files et peut être utilisé avec AWS Lambda.

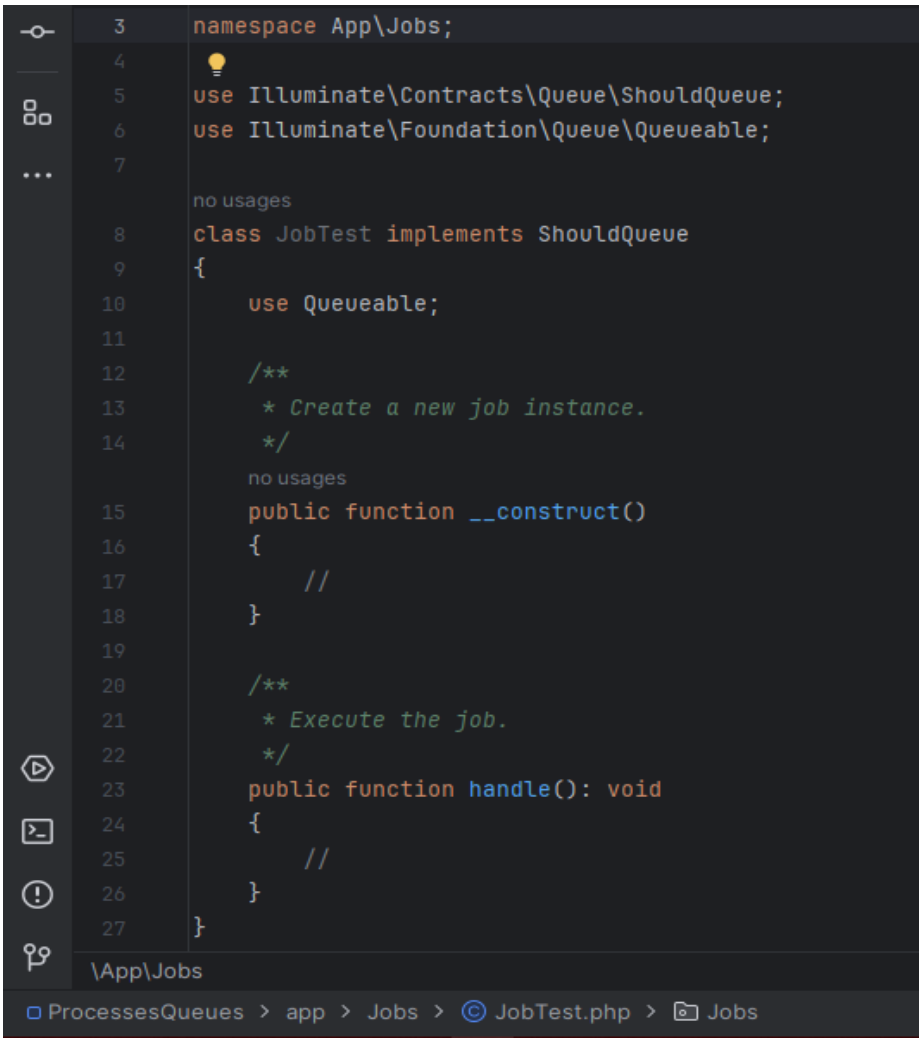
Il faudra ensuite modifier la variable `QUEUE_CONNECTION` des fichiers `.env` et `config/queue.php` pour que sa valeur correspondent au driver choisit.

Exemple : `QUEUE_CONNECTION=database` dans le fichier `.env` et `'default' => env('QUEUE_CONNECTION','database'),` dans `config/queue.php`.

b. Création des jobs

Les jobs sont les tâches exécutées dans une file d'attente. La commande `php artisan make:job nom_du_job` permet de créer un nouveau job stocké dans `app/Jobs/ nom_du_job.php`. Le job créer est une classe qui implémente l'interface `ShouldQueue` qui permet d'exécuter le job en arrière-plan.

A sa création, job un possède un constructeur pour créer une instance du job et une méthode `handle()` qui contiendra le code qui sera exécuter.



Structre d'un job à sa création

3. Quelques méthodes et commandes sur les files d'attente

- `php artisan queue:work` : Permet de lancer la file d'attente
- `php artisan queue:failed` : Surveille les jobs en échec

- `php artisan queue:retry {id}` : Relance le job dont l'id est spécifié
- `php artisan queue:flush` : Supprime les jobs échoués
- `dispatch()` : Dispatch basique, il envoie le job dans la file d'attente pour une exécution asynchrone. Exemple : `nom_du_job::dispatch();`
- `dispatchSync()` : Il exécute le job directement de façon synchrone.

III. Utilisation des processus et des files d'attente dans un projet Laravel

Dans ce projet, nous avons utilisé Laravel pour exécuter du code Python de manière asynchrone en exploitant le système de files d'attente et processus. L'objectif est de permettre aux utilisateurs d'envoyer du code Python à exécuter et de récupérer les résultats une fois le traitement terminé.

L'application se compose de :

- Une interface utilisateur (Vue Blade) où l'utilisateur peut écrire et soumettre du code Python.
- Un contrôleur Laravel (PythonExecutorController) qui reçoit le code et déclenche un job en file d'attente.
- Un job Laravel (ExecutePythonCode) qui exécute le code Python dans un processus isolé.
- Une file d'attente pour traiter les exécutions de code Python en arrière-plan.

- Un cache temporaire pour stocker et récupérer les résultats d'exécution.

1. Détails techniques

- Le contrôleur (PythonExecutorController.php) :
    - Il gère les requêtes entrantes et dispatch le job ExecutePythonCode en file d'attente avec le code reçu et l'identifiant.
    - Il génère un identifiant unique qui est accessible à la vue pour chaque exécution.
    - Il permet aussi de récupérer les résultats depuis le cache.

```php
4 usages  Asad16001
class PythonExecutorController extends Controller
{
    Asad16001
    public function index()
    {
        return view( view: 'python.index'); // affiche la vue dans /v
    }

    1 usage  Asad16001
    public function execute(Request $request)
    {
        $executionId = uniqid( prefix: 'py_'); // génération d'un id p
        ExecutePythonCode::dispatch($request->code, $executionId);
        return response()->json(['execution_id' => $executionId]);
    }
```

Méthode index() et execute() du contrôleur

La méthode `index()` permet d'afficher la vue `index.blade.php` qui se trouve dans `resources/views/python.`

La méthode `execute()` récupère le code saisie par l'utilisateur et lui donne un identifiant unique (py_######) pour suivre le traitement. Ensuite, elle lance le job `ExecutePythonCode` en utilisant la méthode `dispatch()` qui dans ce cas prend deux paramètres le code saisie et l'identifiant généré. Ces paramètres représente les paramètres du constructeur du job `ExecutePythonCode.`

```
    public function getResult($executionId)
    {
        $result = cache()->get("python_result_{$executionId}");
        if ($result) {
            return response()->json($result); // retourne le resu
        }
        return response()->json(['status' => 'pending']); // si
    }
}
```

Méthode getResult() du contrôleur

Après l'exécution du code dans la file d'attente, le résultat est conservé dans le cache avec une clé constituée de l'identifiant de l'exécution en cours. La méthode `getResult()` accède au cache en utilisant cette clé ensuite, elle récupère et met le résultat au format json.

- Le job (ExecutePythonCode.php) :
    - Il implémente l'interface ShouldQueue, ce qui permet son exécution en arrière-plan.
    - Il sauvegarde temporairement le code Python dans un fichier.
    - Il lance un processus système pour exécuter le code via Symfony\Component\Process\Process.
    - Il stocke les résultats en cache pour être récupérés plus tard.

```php
2 usages  Asad16001
class ExecutePythonCode implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    2 usages
    private $code; // Code à exécuter
    3 usages
    private $executionId; // id généré par le contrôleur

    // initialisation
    no usages  Asad16001
    public function __construct(string $code, string $executionId)
    {
        $this->code = $code;
        $this->executionId = $executionId;
    }
```

Constructeur du job

Le constructeur permet d'initialiser le job pour qu'il puisse être placé dans la file d'attente grâce à la méthode `dispatch()`.

```php
    public function handle()
    {
        $filename = "{$this->executionId}.py"; // création du fichier python (ex: "
        Storage::disk( name: 'local')->put($filename, $this->code); // sauvegarde du
        $filepath = Storage::disk( name: 'local')->path($filename); // chemin comple

        $process = new Process(['python', $filepath]); // création du processus
        $process->run(); // exécution du processus (python chemin/complet/du/fichie


        // stockage du resultat dans le cache
        cache()->put(
            key: "python_result_{$this->executionId}", // clé unique pour le résult
            [
                'output' => $process->getOutput() ?: $process->getErrorOutput() //
            ],
            now()->addMinutes( value: 5) //conserve le cache pendant 5 minutes
        );

        Storage::disk( name: 'local')->delete($filename); // suppréssion du fichier
    }
}
```

La méthode `handle()` exécute le code de l'utilisateur en démarrant le processus. Pour ce faire, on crée un fichier `$filename` qui est « l'identifiant.py » ensuite, le code de l'utilisateur est écrit dans le fichier et enfin, on récupère le chemin absolu du fichier `$filepath.`

Après cette opération, le processus est démarré en utilisant `$filepath` et `python` qui revient à faire `python chemin/du/fichier/identifiant.py` en invite de commandes.

Ensuite, le résultat est stocké dans le cache avec une clé qui sera également utilisée pour récupérer le résultat et l'afficher. On pourra alors stocké le résultat de l'exécution du code ou l'erreur si l'interpréteur python détecte des erreurs dans le code soumis.

- L'interface utilisateur (index.blade.php)
    - Elle permet aux utilisateurs d'écrire du code et de le soumettre.
    - Elle interroge ensuite le serveur pour récupérer les résultats et les afficher dynamiquement.

Ce projet illustre comment Laravel peut gérer efficacement des processus externes et des tâches asynchrones via un système de jobs en file d'attente. L'exécution de code Python en file d'attente permet une meilleure gestion des ressources et une amélioration de l'expérience utilisateur.

Conclusion

Les processus et les files d'attente dans Laravel représentent deux piliers fondamentaux pour la gestion des tâches complexes et des opérations en arrière-plan. Bien que servant des objectifs différents, ils se complètent parfaitement dans l'écosystème Laravel.

Les processus excellent dans :

- L'exécution directe de commandes système
- La gestion synchrone ou asynchrone des opérations système
- L'interaction avec des outils externes
- Le contrôle précis des opérations système

Les files d'attente brillent dans :

- Le traitement asynchrone des tâches
- La gestion des charges de travail importantes
- L'amélioration des performances applicatives
- La scalabilité des opérations

En pratique, ces deux fonctionnalités peuvent être utilisées ensemble : par exemple, un job dans une file d'attente peut exécuter un processus système, ou un processus peut déclencher l'ajout de tâches dans une file d'attente. Cette synergie offre une grande flexibilité dans la conception d'applications robustes.