Queues & Jobs In Laravel

- Queue drivers in Laravel
- Laravel Job Batching
- Difference between: Queue, Job, Event

Queue drivers in Laravel

Laravel offers several queue drivers for handling background jobs, which can be configured in your `config/queue.php` file. The primary drivers available include:

This driver executes jobs immediately within the current request. It's primarily used for local development and testing, as it doesn't offer true background processing.

This driver stores jobs in a database table. It's a simple and readily available option, particularly useful for smaller applications or when you want to leverage your existing database infrastructure.

A popular and high-performance in-memory data store, Redis is frequently used as a queue driver in production environments. It offers fast job processing and is well-suited for applications with a high volume of queued jobs. Laravel Horizon is a dedicated dashboard for monitoring and managing Redis queues.

A fully managed message queue service offered by AWS, SQS provides a scalable and reliable solution for queuing jobs, especially for applications deployed on AWS.

A lightweight, open-source work queue, Beanstalkd is another option for managing queues, offering a simple and efficient way to handle background tasks.

This driver discards queued jobs, which can be useful during development or testing when you want to prevent jobs from actually being processed.

You can select the appropriate driver based on your application's needs, scale, and deployment environment. For production applications, Redis or Amazon SQS are often preferred for their performance and scalability.

Laravel Job Batching

Laravel's **job batching** feature allows you to group multiple jobs into a batch and perform actions when the batch completes, fails, or progresses. This is particularly useful for tasks like processing large datasets or executing dependent jobs.

Defining Batchable Jobs

To make a job batchable, include the *Illuminate\Bus\Batchable* trait in your job class. This provides access to the *batch()* method, which allows the job to interact with its batch. For example:

```
namespace App\Jobs;

use Illuminate\Bus\Batchable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Queue\Queueable;

class ImportCsv implements ShouldQueue
{
    use Batchable, Queueable;

    public function handle(): void
    {
        if ($this->batch()->cancelled()) {
            // Stop processing if the batch is cancelled
            return;
        }

        // Process the CSV data
    }
}
```

Dispatching Job Batches

You can dispatch a batch of jobs using the *Bus::batch()* method. This method also supports completion callbacks like *then*, *catch*, and *finally*:

```
use App\Jobs\ImportCsv;
use Illuminate\Bus\Batch;
use Illuminate\Support\Facades\Bus;

$batch = Bus::batch([
    new ImportCsv(1, 100),
    new ImportCsv(101, 200),
    new ImportCsv(201, 300),
])->then(function (Batch $batch) {
```

```
    // All jobs completed successfully
})->catch(function (Batch $batch, Throwable $e) {
    // Handle the first job failure
})->finally(function (Batch $batch) {
    // The batch has finished executing
})->dispatch();
```

Monitoring and Managing Batches

Laravel provides tools to inspect and manage batches:

- *$batch->progress()* returns the completion percentage.
- *$batch->cancel()* cancels the batch.
- *$batch->failedJobs* retrieves the number of failed jobs.

You can also retrieve a batch by its ID using *Bus::findBatch($batchId)*.

Adding Jobs Dynamically

You can add jobs to an existing batch from within a batch job using the *add()* method:

```
$this->batch()->add([
    new ImportCsv(401, 500),
    new ImportCsv(501, 600),
]);
```

Batch Failures and Pruning

When a job in a batch fails, the *catch* callback is triggered. By default, a failed job cancels the entire batch, but you can override this behavior using the *allowFailures()* method. To clean up old batch records, schedule the *queue:prune-batches* Artisan command.

Important Considerations

- **Database Migration**: Ensure you have a *job_batches* table by running *php artisan make:queue-batches-table* and migrating it.
- **Callbacks**: Avoid using *$this* in batch callbacks as they are serialized.
- **Batch Naming**: Use the *name()* method to assign a name for better debugging in tools like Horizon.

Laravel's job batching simplifies handling complex workflows by grouping jobs and providing robust tools for monitoring and managing their execution.

Difference between: Queue, Job, Event

a `Queue` is a mechanism that holds items in waiting, while a `Job` is the specific task to be performed, and an `Event` is a notification that something has occurred.

Feature	Event	Job	Queue
Primary Role	A notification that something has happened.	A specific unit of work or task to be executed.	A data structure (usually FIFO) that manages waiting tasks.
Trigger	Fired when a specific occurrence or change of state takes place (e.g., `UserSignedUp`).	Explicitly dispatched by a controller, command, or event listener when work is needed.	The system continually checks the queue for items to process.
Execution	Can be synchronous (immediate) or asynchronous (queued).	Can be synchronous (run immediately) or asynchronous (queued).	Facilitates asynchronous processing by holding tasks to be run in the background.
Listeners/ Handlers	Can have multiple independent listeners reacting to a single event, promoting decoupling.	Typically a single, self-contained unit of work with a defined handler method.	The mechanism that a worker uses to retrieve the next job to process.
Purpose	Decouples different parts of an application by allowing them to react to occurrences without direct dependency.	Used for time-consuming or long-running tasks that shouldn't block the main application flow (e.g., sending emails, data processing).	Provides a reliable buffer and load-leveling system for processing tasks efficiently and at scale.

Summary of Differences

Event: An event is essentially a piece of data that describes "what happened" in the system. It acts as a signal that one or more other components in the system might be interested in. The power of events is that you can have multiple, independent listeners perform different actions in response to a single event without those listeners knowing about each other.

Job: A job is an actual, self-contained task or unit of work that needs to be performed. Jobs are explicitly created and dispatched to a system to "do something" specific, like generating a report or processing a video file.

Queue: A queue is the underlying infrastructure or data structure that holds the jobs (or asynchronous events) that are waiting to be processed. It is a waiting line, typically following the "First-In, First-Out" (FIFO) principle, where a worker process picks up items one by one. The queue itself does not define the task but manages its order and execution flow.