QueueService

September 4, 2026

QueueService

Kind: Service

Source: `atloria-monorepo/apps/parser-orchestrator/src/queue/queue.service.ts`

`QueueService` is a NestJS service that manages parse jobs in the parser orchestrator queue. It provides methods to enqueue individual or bulk parsing work, inspect job state and results, wait for completion, cancel jobs, and retrieve queue metrics or job lists for operational monitoring.

Methods

Method	Signature	Returns	Description
`addParseJob`	`addParseJob(payload: Omit<ParseJobPayload, 'jobId'	'createdAt'>, priority: unknown)`	`Promise<string>`
`addParseJobsBulk`	`addParseJobsBulk(payloads: Array<Omit<ParseJobPayload, 'jobId'	'createdAt'>>)`	`Promise<string[]>`
`getJobStatus`	`getJobStatus(jobId: string)`	`Promise<JobStatus	null>`
`getJobResult`	`getJobResult(jobId: string)`	`Promise<T	null>`
`waitForJob`	`waitForJob(jobId: string, timeout: unknown)`	`Promise<T>`	Wait for job to complete
`cancelJob`	`cancelJob(jobId: string)`	`Promise<boolean>`	Cancel a job
`getMetrics`	`getMetrics()`	`Promise<QueueMetrics>`	Get queue metrics
`getWaitingJobs`	`getWaitingJobs(start: unknown, end: unknown)`	`Promise<Job<ParseJobPayload>[]>`	Get waiting jobs
`getActiveJobs`	`getActiveJobs()`	`Promise<Job<ParseJobPayload>[]>`	Get active jobs
`getFailedJobs`	`getFailedJobs(start: unknown, end: unknown)`	`Promise<Job<ParseJobPayload>[]>`	Get failed jobs

Method	Signature	Returns	Description
`retryJob`	`retryJob(jobId: string)`	`Promise<void>`	Retry a failed job
`cleanCompleted`	`cleanCompleted(grace: unknown)`	`Promise<void>`	Clean completed jobs
`cleanFailed`	`cleanFailed(grace: unknown)`	`Promise<void>`	Clean failed jobs
`pause`	`pause()`	`Promise<void>`	Pause the queue
`resume`	`resume()`	`Promise<void>`	Resume the queue
`isPaused`	`isPaused()`	`Promise<boolean>`	Check if queue is paused

Dependencies

- `Queue`

Where it refuses work

- `QueueService` stops the work with `Error` when `!job`, in 2 places.
- `QueueService` stops the work with an early return when `!job`, in 3 places.

Diagram

```
sequenceDiagram
    participant Client
    participant QueueService
    participant Queue
    participant Worker

    Client->>QueueService: addParseJob(payload)
    QueueService->>Queue: add(parse job)
    Queue-->>QueueService: jobId
    QueueService-->>Client: jobId

    Queue->>Worker: deliver queued job
    Worker->>Worker: parse payload
    Worker->>Queue: store result / status

    Client->>QueueService: waitForJob(jobId)
    QueueService->>Queue: await job completion
    Queue-->>QueueService: result or failure
    QueueService-->>Client: parsed result
```

Usage

```typescript
import { Injectable } from '@nestjs/common';
import { QueueService } from './queue/queue.service';

@Injectable()
export class ParseRequestService {
  constructor(private readonly queueService: QueueService) {}

  async submitDocumentParse(documentId: string, sourceUrl: string) {
    const jobId = await this.queueService.addParseJob({
      documentId,
      sourceUrl,
    });

    return {
      jobId,
      status: 'queued',
    };
  }

  async getParseResult<T>(jobId: string): Promise<T | null> {
    const status = await this.queueService.getJobStatus(jobId);

    if (status === 'completed') {
      return this.queueService.getJobResult<T>(jobId);
    }

    return null;
  }

  async parseAndWait<T>(documentId: string, sourceUrl: string): Promise<T> {
    const jobId = await this.queueService.addParseJob({
      documentId,
      sourceUrl,
    });

    return this.queueService.waitForJob<T>(jobId);
  }
}
```

AI Coding Instructions

- Use `addParseJob()` for a single parse request and `addParseJobsBulk()` when submitting multiple independent payloads.
- Prefer returning the job ID to API callers and querying status/results asynchronously instead of blocking request handlers with `waitForJob()`.
- Treat `getJobStatus()` and `getJobResult()` as nullable; jobs may not exist, may have expired, or may not have completed yet.
- Use `cancelJob()` only for jobs that are still eligible for cancellation, and handle a `false` result as a normal outcome.

- Use `getMetrics()`, `getWaitingJobs()`, `getActiveJobs()`, and `getFailedJobs()` for operational endpoints or diagnostics rather than exposing queue implementation details directly.

Relationships

- DEPENDS_ON → `queue`

Referenced By

- `OrchestratorService` (DEPENDS_ON)
- `QueueModule` (MODULE_PROVIDES)
- `QueueModule` (MODULE_EXPORTS)