Job Management Service

Draft technical specification
v. 1.0

Author: Zakhar Izverov
Created: 12.09.2021
Updated: 18.09.2021

Overview

The purpose of this document is to describe the technical design of a simple job management service (JMS). This system should be capable of handling the execution of multiple types of jobs.

Requirements

The following requirements should be met:

1. The service is capable to perform any job, without restrictions (e.g., loading data into a DWH, indexing of some file-based content or sending emails)
2. Types of executed jobs are not known to the service. Running any new job type does not require the service redevelopment
3. Failed jobs do not create side-effects
4. A job has one of the following statuses, which reflect the job state:
    a. QUEUED
    b. RUNNING
    c. SUCCESS
    d. FAILED
5. Each job is executed based on its priority in relation to other jobs (HIGH, MEDIUM, or LOW)
6. There are two types of job execution: immediate and scheduled

Implementation

The JMS implementation has three key components:

1. *Job* class
2. *JobScheduler* class
3. *PropertiesReader* class

1. Job

A *Job* represents a job that can be executed by JMS. The *Job* object contains the following fields:

- *jobRunnable* – a runnable task
- *jobName* – common *Job* name
- *jobId* – unique *Job* ID, generated automatically
- *jobStatus* – *Job* status (*CREATED*, *QUEUED*, *RUNNING*, *SUCCESS* or *FAILED*)
- *jobPriority* – *Job* priority (*HIGH*, *MEDIUM* or *LOW*)
- *jobSchedule* – type of *Job* scheduling to execution (*IMMEDIATE*, *DELAYED*, *PERIODIC*)

Job itself implements *Runnable* interface and thus could be directly executed by the *JobScheduler*.

Jobs are constructed through static factory method *newJob*, which has four overloaded implementations for the different *Job* scheduling types.

2. **JobScheduler**

JobScheduler operates based on the producer-consumer model:

- When a *Job* is scheduled (using the *scheduleJob* method), it is added (produced) to the scheduler's *PriorityBlockingQueue*
- If the *JobScheduler* instance is started, it uses a single-threaded executor to take (consume) Jobs from the queue one by one according to their priority
- Jobs, taken from the queue, are scheduled to execute in a separate scheduled thread pool

JobScheduler is configurable through the jms.properties file located in the project root. One can set up:

- *Job* execution thread pool size (default - 10)
- *Job* queue size (default - 100)
- *JobScheduler* shutdown timeout in seconds (default - 10)

JobScheduler is constructed through static factory method *newJobScheduler*, which can take path to the properties file as a parameter. After constructing the scheduler instance, *Jobs* can be added to its queue. Actual *Job* execution, however, will only start after the *JobScheduler* is started using *start* method. *JobScheduler* should be shut down after usage with the *stop* method.

3. **PropertiesReader**

PropertiesReader has one public static method - *readProperties*. This method takes a path to the properties file as its argument (nullable *String*) and tries to read the properties from this file.

If no file path is specified, the file could not be read or contains invalid values, the method returns default values. If some parameters are not set in the file, they are set to default values. Configuration parameters are transferred to the *JobScheduler* in a special *Props* object.

Assumptions and extensions

- Status *CREATED* was added to the *Job* status model to represent the *Job* state before queuing