



Gain Proficiency in Batch Processing with Spring Batch




Mastering Batch Processing:

A Guide to Spring Batch Proficiency



Visit Us for more
Information

www.inexture.com 

In the ever-evolving landscape of enterprise applications, efficiently handling large-scale data processing tasks is a common challenge. Spring Batch, is a robust batch application development framework that is lightweight and all-inclusive, facilitating the creation of batch applications essential to enterprise system everyday operations.

Spring Batch provides reusable functions essential in processing large volumes of records, including logging/tracing, transaction management, job processing statistics, job restart, skip, and resource management.

Spring Batch Architecture

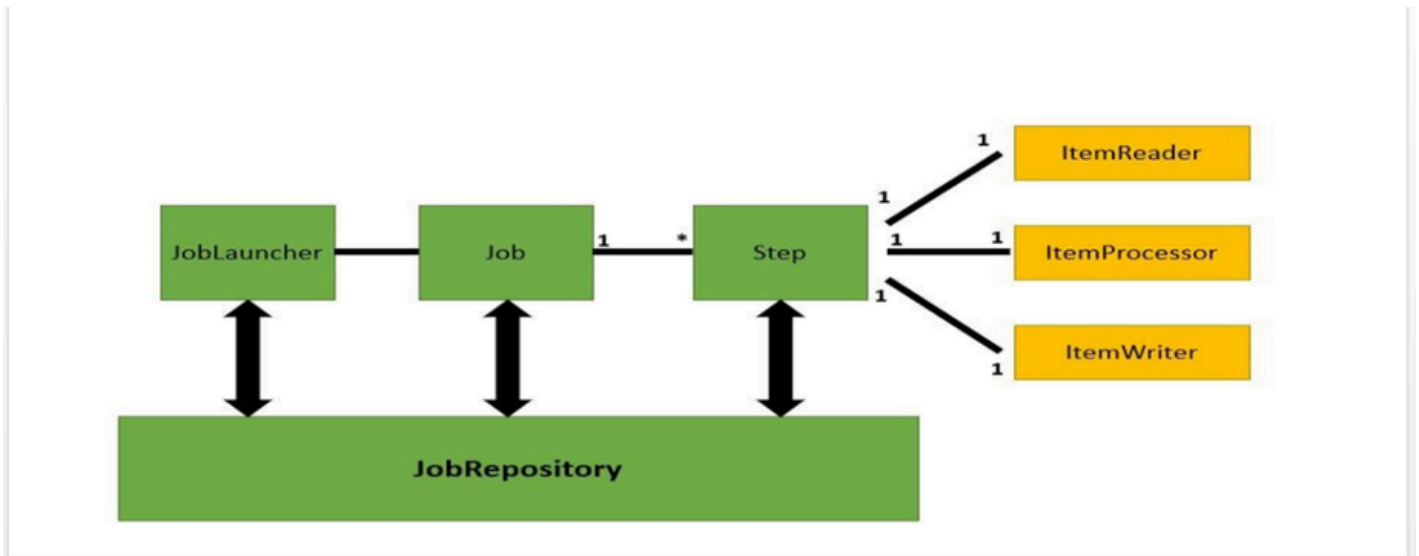
A typical batch application is roughly as follows.

- Read a large number of records from a database, file, or queue.
- Process the data in some way.
- Write back the data in a modified form.

The corresponding schematic is as follows.



The general architecture of the spring batch is as follows.



Core Concepts

Job: In Spring Batch, a job is an executable task that can be divided into smaller, more manageable jobs.

Step: Within a job, a step is a self-contained, executable unit. It performs a certain function, like reading, processing, and writing data.

Item: An item is a data point that has undergone step processing. It could be any kind of data, such as a line from a file or a record from a database.

Building Blocks of Spring Batch

- Readers, Processors, and Writers

ItemReader

Reads information from a file or database, among other sources. An ItemReader reads one item at a time. Spring Batch provides an Interface **ItemReader**. All the **readers** implement this interface.

Here are some of the predefined ItemReader classes provided by Spring Batch to read from various sources.

Reader

FlatFileItemReader
StaxEventItemReader

Purpose

To read data from flat files.
To read data from XML files.

StoredProcedureItemReader	To read data from the stored procedures of a database.
JDBCPagingItemReader	To read data from relational databases database.
MongoItemReader	To read data from MongoDB.
Neo4jItemReader	To read data from Neo4jItemReader.

ItemProcessor

Reads data from the ItemReader and processes it before sending it to the ItemWriter. An ItemWriter writes one item at a time. Spring Batch provides an Interface **ItemWriter**. All the writers implement this interface. Here are some of the predefined ItemWriter classes provided by Spring Batch to read from various sources.

Writer	Purpose
FlatFileItemWriter	To write data into flat files.
StaxEventItemWriter	To write data into XML files.
StoredProcedureItemWriter	To write data into the stored procedures of a database.
JDBCPagingItemWriter	To write data into relational databases database.
MongoItemWriter	To write data into MongoDB.
Neo4jItemWriter	To write data into Neo4j.

ItemWriter

An ItemProcessor is used to process the data. When the given item is not valid it returns **null**, else it processes the given item and returns the processed result.

Tasklet

When no reader and writer are given, a **tasklet** acts as a processor for SpringBatch. It processes only a single task.

Chunks and Transactions

Spring Batch uses chunks to process data, which enables efficient handling of large datasets. The process of transactions ensures either all items in a chunk are processed successfully or none.

Job Execution Flow

A job is usually composed of one or more steps, and the execution process can be customized to meet specific requirements.

XML or Java-based configuration can be used to configure and orchestrate jobs and steps.

JobRepository

A Job repository in Spring Batch provides Create, Retrieve, Update, and Delete (CRUD) operations for the JobLauncher, Job, and Step implementations.

Real-World Applications

- **Data Migration**

The efficiency of transferring large amounts of data from one system to another is demonstrated by Spring Batch, which proves to be instrumental in data migration scenarios.

- **ETL Mastery**

The extraction, transformation, and loading of data is a fundamental use case for Spring Batch. Developers are empowered to create jobs that collect data from diverse sources, apply transformations, and archive the refined results.

- **Report Generation**

Batch processing has a specialization in producing complicated reports that involve complex calculations or aggregations.

Features of Spring Batch

- Maintainability
- Transaction management
- Flexibility
- Retry and Skip Mechanisms
- Chunk based processing

Implementation of Spring Batch

Step 1: Project Setup

To begin your Spring Batch project, establish a new Spring Boot project. You have the option to use either the Spring Initializer, which is a web-based tool or manually configure your project through your preferred IDE.

Method 1: Using Spring Initializr

Go to Spring Initializr and select the project settings you want, which include project type, language, and packaging.

Include the dependency for “Spring Batch” in your project.

To download the project structure as a ZIP file, click on “Generate”.

Method 2: Manual Configuration

Create a new Spring Boot project in your preferred IDE, ensuring that you have included the necessary Spring Batch dependencies.

Set up your project with the right directory structure to make batch-related components easy to find and organize.

Step 2: Define Job and Steps

Once your project is set up, define a job and its steps. Batch processing can be divided into phases with one or more steps per job. The steps can be configured using `ItemReader`, `ItemProcessor`, and `ItemWriter` implementations.

```
@Configuration
@EnableBatchProcessing
public class BatchConfiguration {

    @Autowired
    private JobBuilderFactory jobBuilderFactory;

    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Autowired
    private ItemReader<MyEntity> itemReader;

    @Autowired
    private ItemProcessor<MyEntity, MyProcessedEntity> itemProcessor;

    @Autowired
    private ItemWriter<MyProcessedEntity> itemWriter;

    @Bean
    public Job myBatchJob() {
        return jobBuilderFactory.get("myBatchJob")
            .start(myStep())
            .build();
    }
}
```

```
@Bean
public Step myStep() {
    return stepBuilderFactory.get("myStep")
        .<MyEntity, MyProcessedEntity>chunk(10)
        .reader(itemReader)
        .processor(itemProcessor)
        .writer(itemWriter)
        .build();
}
}
```

Step 3: Implement ItemReader, ItemProcessor, and ItemWriter

Develop customized implementations for `ItemReader`, `ItemProcessor`, and `ItemWriter` adapted to your specific use case. Spring Batch includes several preinstalled implementations, like `JdbcCursorItemReader` and `JpaItemWriter`, that can be modified to suit your requirements.

```
@Bean
public ItemReader<MyEntity> itemReader() {
    // Custom implementation of ItemReader, e.g., JdbcCursorItemReader
}

@Bean
public ItemProcessor<MyEntity, MyProcessedEntity> itemProcessor() {
    // Custom implementation of ItemProcessor
}

@Bean
public ItemWriter<MyProcessedEntity> itemWriter() {
    // Custom implementation of ItemWriter, e.g., JpaItemWriter
}
```

Step 4: Configure Batch Properties

Adjust your batch job by configuring batch-related properties. Modify settings including chunk size, retry policies, and transaction management to improve the performance and reliability of your batch jobs.

```
# application.properties or application.yml
spring.batch.job.names=myBatchJob
spring.batch.initialize-schema=always
spring.batch.job.enabled=true
# ... other batch-related properties
```

To make it clear, let's define each property:

spring.batch.job.names

Specifies the names of the batch jobs to be executed. Multiple job names can be provided as a comma-separated list.

spring.batch.initialize-schema

Controls the initialization of the batch schema in the underlying database. Setting it to **'always'** ensures that the schema is created every time the application starts.

spring.batch.job.enabled

Indicates whether the execution of batch jobs is enabled or disabled. When set to **'true'**, jobs will run as usual. Conversely, setting it to **'false'** prevents the execution of any configured batch jobs.

Step 5: Run the Batch Job

Run your batch job either programmatically or with Spring Boot's built-in command line support. Check the status of the job execution by monitoring it through the JobRepository.

```
@SpringBootApplication
public class BatchApplication {

    public static void main(String[] args) {
        SpringApplication.run(BatchApplication.class, args);
    }
}
```

Key-Takeaway

With Spring Batch, developers can easily handle complex batch processing scenarios. The framework's flexibility and scalability make it a top choice in the Java ecosystem for handling large datasets or orchestrating ETL workflows. By mastering Spring Batch's core concepts and building blocks, developers can unleash their full potential for efficient and reliable batch processing.

Explore more about the [latest updates in Software Technology](#) on our blog page for continuous insights and innovations.

Originally published by: [Gain Proficiency in Batch Processing with Spring Batch](#)