Spring Batch POC

# Introduction

Spring Batch is a processing framework designed for robust execution of jobs.

Spring Batch builds upon the characteristics of the Spring while making it easy for developers to access and leverage more advance enterprise services when necessary. Spring Batch is intended to work in conjunction with a scheduler, not replace a scheduler.

Spring Batch provides reusable functions that are essential in processing large volumes of records, including logging/tracing, transaction management, job processing statistics, job restart, skip, and resource management. It also provides more advanced technical services and features that enable extremely high-volume and high performance batch jobs though optimization and partitioning techniques. Spring Batch can be used in both simple use cases (such as reading a file into a database or running a stored procedure) as well as complex, high volume use cases (such as moving high volumes of data between databases, transforming it, and so on). High-volume batch jobs can leverage the framework in a highly scalable manner to process significant volumes of information. (Ref :<https://docs.spring.io/>)

Features

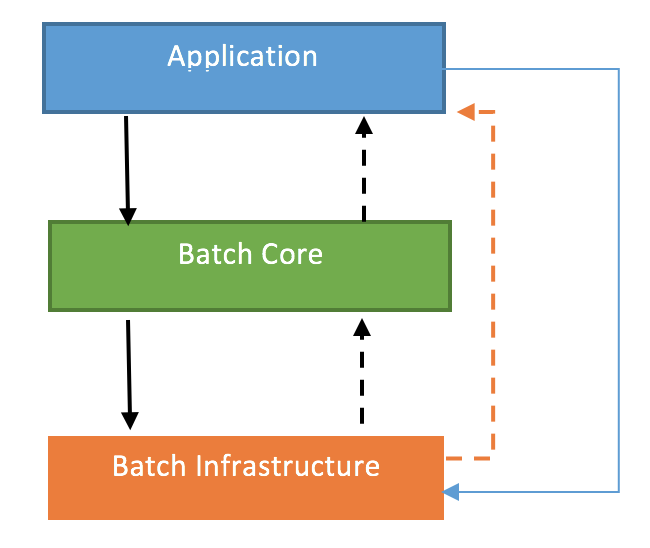
* Job Flow State Machine
* Transaction Management
* Declarative I/O
* Robust Error Handling
* Scalability

A typical batch program generally:

* Reads a large number of records from a database, file, or queue.
* Processes the data in some fashion.
* Writes back data in a modified form

It also accommodates JSR-352, which is new java specification for batch processing.

# Architecture and Workflow



* **Application**: it has all the batch jobs and code
* **Batch Core**: It has all the runtime classes needed to launch and control a batch job.
* **Batch Infrastructure**: This contains the reader and writer services(ItemReader and ItemWriter) which are used by developer and the framework itself.

Spring Batch is a state machine. A Spring Batch workflow looks like this:

A close up of text on a black background

Description automatically generated

* **Job** – Defines a flow, processing which will take the program through those states
* **Step** – It represents the state in the state machine. It is an object that encapsulates sequential phase of a job and holds all the necessary information to define and control processing.
* **Job Repository –** It maintains the state of the job.
* **Job Launcher –** Creates the state of the job before launching it.

A screenshot of a social media post

Description automatically generated

* The ***ItemReader*** reads the input data and provides the found items one by one. An *ItemReader* belongs to one step and each step must have only one *ItemReader*.
* The ***ItemProcessor*** transforms items into a form that is understood by the *ItemWriter* one item at a time. An *ItemProcessor* belongs to one step and each step can have one *ItemProcessor*.
* The ***ItemWriter*** writes an information of an item to the output one item at a time. An *ItemWriter* belongs to one step and a step must have only one *ItemWriter*

(Ref: <https://www.petrikainulainen.net/programming/spring-framework/spring-batch-tutorial-introduction/>)

The input for entire step is divided into smaller chunks and that input is then read, processed and written for each chunk. For each chunk, reader will read the entire chunk, until the chunk limit is reached, and the entire chunk is then processed and passed on to ItemWriter, which will write all the items in the chunk at once. This process is repeated until all the input is exhausted.

# Configuration

## Maven Dependency

<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring-jdbc</artifactId>

    <version>5.2.0.RELEASE</version>

</dependency>

<dependency>

    <groupId>org.springframework.batch</groupId>

    <artifactId>spring-batch-core</artifactId>

    <version>4.2.0.RELEASE</version>

</dependency>

***For Spring boot***

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-batch</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

## Batch Config

<bean id=*"jobRepository"*

class=*"org.springframework.batch.core.repository.support.JobRepositoryFactoryBean"*>

<property name=*"dataSource"* ref=*"dataSource"* />

<property name=*"transactionManager"* ref=*"transactionManager"* />

<property name=*"databaseType"* value=*"mysql"* />

</bean>

<bean id=*"jobLauncher"* class=*"org.springframework.batch.core.launch.support.SimpleJobLauncher"*>

<property name=*"jobRepository"* ref=*"jobRepository"* />

</bean>

<bean id=*"transactionManager"*

class=*"org.springframework.batch.support.transaction.ResourcelessTransactionManager"* />

<bean id=*"dataSource"*

class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"* />

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/mysql"* />

<property name=*"username"* value=*"root"* />

<property name=*"password"* value=*""* />

</bean>

<!-- Create meta-tables -->

<jdbc:initialize-database data-source=*"dataSource"*>

<jdbc:script location=*"org/springframework/batch/core/schema-drop-mysql.sql"* />

<jdbc:script location=*"org/springframework/batch/core/schema-mysql.sql"* />

</jdbc:initialize-database>

## Application.properties

spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/mysql

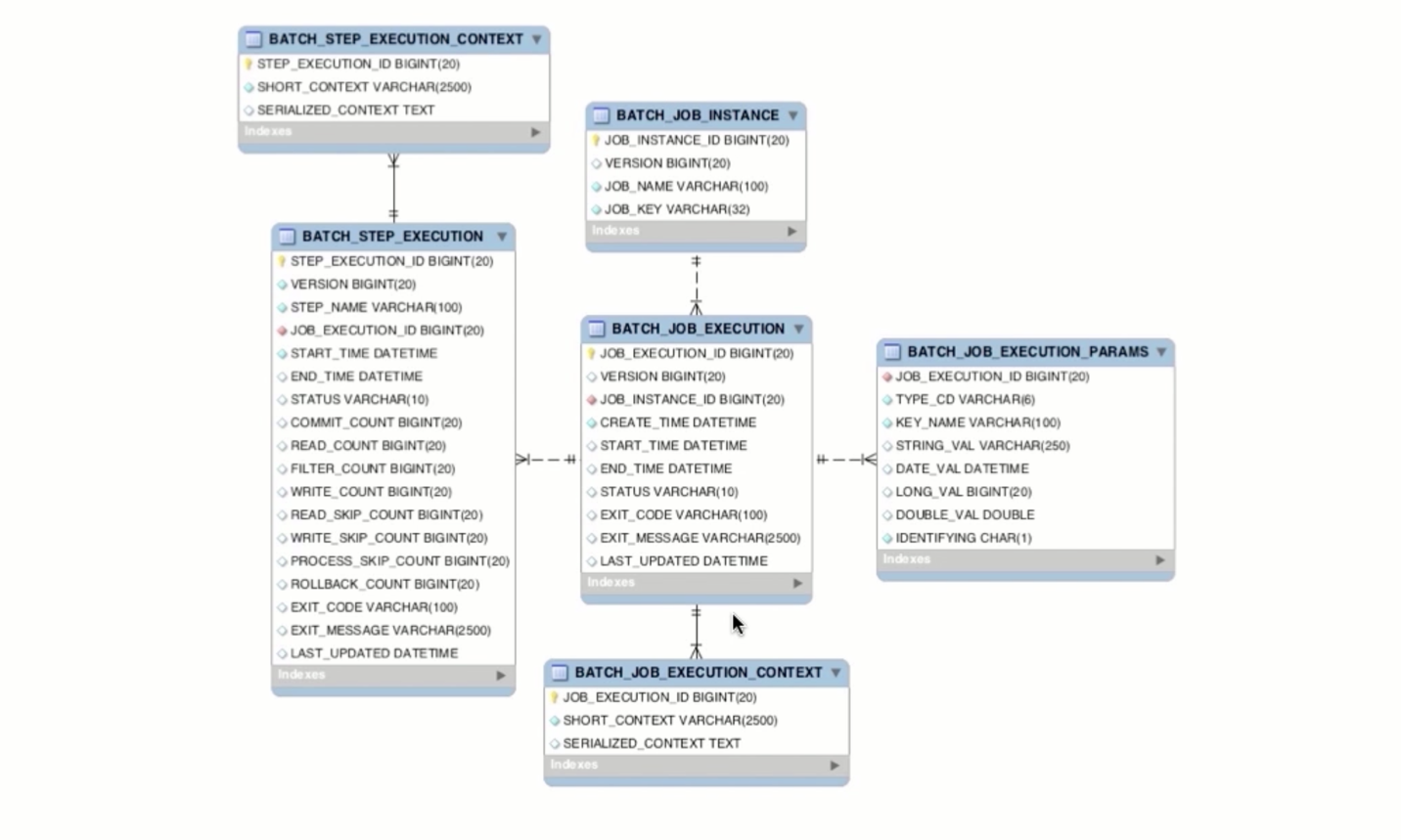
spring.datasource.username=root

spring.datasource.password=passw0rd

spring.batch.initialize-schema=ALWAYS

spring.datasource.schema=org/springframework/batch/core/schema-mysql.sql

The structure of default schema create using schema-mysql.sql looks like



## Java Configuration Sample

@Configuration

@EnableBatchProcessing

public class SpringConfig {

    @Value("org/springframework/batch/core/*schema-drop-mysql.sql*")

    private Resource dropReopsitoryTables;

    @Value("org/springframework/batch/core/*schema-mysql.sql*")

    private Resource dataReopsitorySchema;

    @Bean

    public DataSource dataSource() {

        DriverManagerDataSource dataSource = new DriverManagerDataSource();

        dataSource.setDriverClassName("*com.mysql.jdbc.Driver*");

        dataSource.setUrl("jdbc:mysql://localhost:3306/mysql");

        return dataSource;

    }

    @Bean

    public DataSourceInitializer dataSourceInitializer(DataSource dataSource)

      throws MalformedURLException {

        ResourceDatabasePopulator databasePopulator =

          new ResourceDatabasePopulator();

        databasePopulator.addScript(dropReopsitoryTables);

        databasePopulator.addScript(dataReopsitorySchema);

        databasePopulator.setIgnoreFailedDrops(true);

        DataSourceInitializer initializer = new DataSourceInitializer();

        initializer.setDataSource(dataSource);

        initializer.setDatabasePopulator(databasePopulator);

        return initializer;

    }

    private JobRepository getJobRepository() throws Exception {

        JobRepositoryFactoryBean factory = new JobRepositoryFactoryBean();

        factory.setDataSource(dataSource());

        factory.setTransactionManager(getTransactionManager());

        factory.afterPropertiesSet();

        return (JobRepository) factory.getObject();

    }

    private PlatformTransactionManager getTransactionManager() {

        return new ResourcelessTransactionManager();

    }

    public JobLauncher getJobLauncher() throws Exception {

        SimpleJobLauncher jobLauncher = new SimpleJobLauncher();

        jobLauncher.setJobRepository(getJobRepository());

        jobLauncher.afterPropertiesSet();

        return jobLauncher;

    }

}

## Spring Batch Job Config

<import resource=*"../config/context.xml"* />

<import resource=*"../config/database.xml"* />

<bean id=*"report"* class=*"com.springBatch.model.Report"* scope=*"prototype"* />

<bean id=*"itemProcessor"* class=*"com.springBatch.CustomItemProcessor"* />

<batch:job id=*"myJob"*>

<batch:step id=*"step1"*>

<batch:tasklet>

<batch:chunk reader=*"cvsFileItemReader"* writer=*"xmlItemWriter"* processor=*"itemProcessor"*

commit-interval=*"10"*>

</batch:chunk>

</batch:tasklet>

</batch:step>

</batch:job>

<bean id=*"cvsFileItemReader"* class=*"org.springframework.batch.item.file.FlatFileItemReader"*>

<property name=*"resource"* value=*"classpath:cvs/input/report.csv"* />

<property name=*"lineMapper"*>

<bean class=*"org.springframework.batch.item.file.mapping.DefaultLineMapper"*>

<property name=*"lineTokenizer"*>

<bean

class=*"org.springframework.batch.item.file.transform.DelimitedLineTokenizer"*>

<property name=*"names"* value=*"id,sales,qty,staffName,date"* />

</bean>

</property>

<property name=*"fieldSetMapper"*>

<bean class=*"com.springBatch.ReportFieldSetMapper"* />

<!-- if no data type conversion, use BeanWrapperFieldSetMapper to map by name

<bean

class="org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper">

<property name="prototypeBeanName" value="report" />

</bean>

-->

</property>

</bean>

</property>

</bean>

<bean id=*"xmlItemWriter"* class=*"org.springframework.batch.item.xml.StaxEventItemWriter"*>

<property name=*"resource"* value=*"file:xml/outputs/report.xml"* />

<property name=*"marshaller"* ref=*"reportMarshaller"* />

<property name=*"rootTagName"* value=*"report"* />

</bean>

<bean id=*"reportMarshaller"* class=*"org.springframework.oxm.jaxb.Jaxb2Marshaller"*>

<property name=*"classesToBeBound"*>

<list>

<value>com.springBatch.model.Report</value>

</list>

</property>

</bean>