**Spring Batch**

Spring Batch is a lightweight, comprehensive batch framework designed to enable the development of robust batch applications.

**Spring Batch Application**

The application which will process bulk records of data is called Batch Application.

Ex:

Sending salary to all Employees

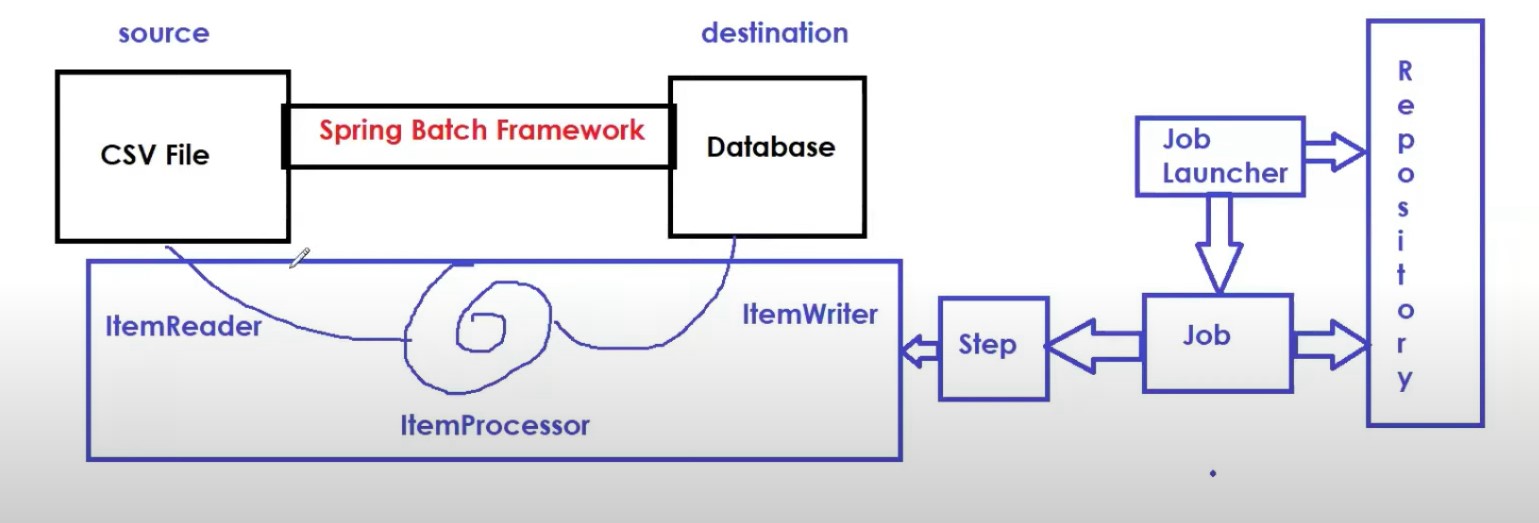
Sending greetings to all Employees during festive seasons

Sending salary slips to all Employees.

**Spring Batch Terminology**

* JobLauncher- launches the job
* Job – Execute the Step
* Step – Task to be done
* ItemReader – Reads from the source file
* ItemProcessor – Executes the conversion
* ItemWriter – Writes into destination file

**Reading data from CSV file to Database using Spring Batch**



Each component has a different functionality in the SpringBatch. ItemReader reads the CSV file, ItemProcessor will process the obtained data using the suitable logic. Item Writer will write the data into database.All these are included in the Step. To execute the step, we should execute the Job. To launch the job, we need to execute the job launcher. The function of joblauncher is to launch the job. The job may have many tasks to execute. The job repository contains will contain the job execution metadata.

**Demo Project was done to illustrate the reading of CSV file to Database.**

**Dependencies Used:**

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.30</version>

<scope>provided</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>spring-batch-test</artifactId>

<scope>test</scope>

</dependency>

**If u use the 3.2.1 Spring version**

* customerReader Method:

Purpose: Configures a FlatFileItemReader for reading data from a CSV file.

Details:

Sets the resource to the CSV file using setResource.

Skips the first line (header) using setLinesToSkip(1).

Configures a line mapper using the lineMapper method.

Returns the configured FlatFileItemReader.

* lineMapper Method:

Purpose: Configures a line mapper for mapping lines from the CSV file to Customer objects.

Details:

Configures a DelimitedLineTokenizer to tokenize each line using commas as delimiters.

Sets the column names for the tokenizer.

Configures a BeanWrapperFieldSetMapper to map the tokenized values to a Customer object.

Returns a configured DefaultLineMapper with the tokenizer and field set mapper.

* customerProcessor Method:

Purpose: Configures a bean of type CustomerProcessor.

Details:

Returns a new instance of CustomerProcessor, indicating that this method is expected to return a bean of that type.

The actual implementation of CustomerProcessor is assumed to contain the custom processing logic for each Customer item.

* customerWriter Method:

Purpose: Configures a RepositoryItemWriter for writing Customer entities to the database.

Details:

Creates a new instance of RepositoryItemWriter.

Sets the repository (customerRepo) to be used for writing.

Sets the method name (save) of the repository to be invoked for writing.

Returns the configured RepositoryItemWriter.

* step1 Method:

Purpose: Configures a Spring Batch step (Step) for processing data in chunks.

Details:

Uses StepBuilder to create a step named "step-1" with a specified JobRepository.

Configures a chunk-oriented processing model (<Customer, Customer>) with a reader (customerReader), processor (customerProcessor), and writer (customerWriter).

Defines the chunk size as 10 and sets the transaction manager (manager) for handling transactions.

Returns the configured Step.

* job Method:

Purpose: Configures a Spring Batch job (Job) named "customers-job".

Details:

Uses JobBuilder to create a job with a specified JobRepository.

Configures the job to flow through step1.

Ends the job configuration.

Returns the configured Job.

@Configuration

//@EnableBatchProcessing

//@AllArgsConstructor

public class CsvBatchConfig {

@Autowired

private CustomerRepository customerRepo;

@Autowired

private JobRepository jobRepository;

@Autowired

private PlatformTransactionManager manager;

@Bean

public FlatFileItemReader<Customer> customerReader()

{

FlatFileItemReader<Customer> itemReader =new FlatFileItemReader<>();

itemReader.setResource(new FileSystemResource("src/main/resources/customers.csv"));

itemReader.setName("csv-reader");

itemReader.setLinesToSkip(1);

itemReader.setLineMapper(lineMapper());

return itemReader;

}

private LineMapper<Customer> lineMapper() {

DefaultLineMapper<Customer> lineMapper =new DefaultLineMapper<>();

DelimitedLineTokenizer lineTokenizer=new DelimitedLineTokenizer();

lineTokenizer.setDelimiter(",");

lineTokenizer.setStrict(false);

lineTokenizer.setNames("id","firstName","lastName","email","gender","contactNo","country","dob");

BeanWrapperFieldSetMapper<Customer> fieldSetMapper=new BeanWrapperFieldSetMapper<>();

fieldSetMapper.setTargetType(Customer.class);

lineMapper.setLineTokenizer(lineTokenizer);

lineMapper.setFieldSetMapper(fieldSetMapper);

return lineMapper;

}

//create processor

@Bean

public CustomerProcessor customerProcessor()

{

return new CustomerProcessor();

}

//create writer

@Bean

public RepositoryItemWriter<Customer> customerWriter()

{

RepositoryItemWriter<Customer> repositoryWriter =new RepositoryItemWriter<>();

repositoryWriter.setRepository(customerRepo);

repositoryWriter.setMethodName("save");

return repositoryWriter;

}

//create step

@Bean

public Step step1()

{

return new StepBuilder("step-1”, jobRepository).<Customer,Customer>chunk(10,manager)

. reader(customerReader())

. processor(customerProcessor())

. writer(customerWriter())

. build ();

}

//create job

@Bean

public Job job ()

{

return new JobBuilder("customers-job",jobRepository)

. flow(step1())

. end ()

. build ();

}

}

**If u use the 2.7.11 Spring version**

* customerReader Method:

Purpose: Configures a FlatFileItemReader for reading data from a CSV file.

Details:

Sets the resource to the CSV file using setResource.

Skips the first line (header) using setLinesToSkip(1).

Configures a line mapper using the lineMapper method.

Returns the configured FlatFileItemReader.

* lineMapper Method:

Purpose: Configures a line mapper for mapping lines from the CSV file to Customer objects.

Details:

Configures a DelimitedLineTokenizer to tokenize each line using commas as delimiters.

Sets the column names for the tokenizer.

Configures a BeanWrapperFieldSetMapper to map the tokenized values to a Customer object.

Returns a configured DefaultLineMapper with the tokenizer and field set mapper.

* customerProcessor Method:

Purpose: Configures a bean of type CustomerProcessor.

Details:

Returns a new instance of CustomerProcessor, indicating that this method is expected to return a bean of that type.

The actual implementation of CustomerProcessor is assumed to contain the custom processing logic for each Customer item.

* customerWriter Method:

Purpose: Configures a RepositoryItemWriter for writing Customer entities to the database.

Details:

Creates a new instance of RepositoryItemWriter.

Sets the repository (customerRepository) to be used for writing.

Sets the method name (save) of the repository to be invoked for writing.

Returns the configured RepositoryItemWriter.

* step Method:

Purpose: Configures a Spring Batch step (Step) for processing data in chunks.

Details:

Uses StepBuilderFactory to create a step named "step-1".

Configures a chunk-oriented processing model (<Customer, Customer>) with a reader (customerReader), processor (customerProcessor), and writer (customerWriter).

Defines the chunk size as 10 (number of items to be processed in each transaction).

Returns the configured Step.

* job Method:

Purpose: Configures a Spring Batch job (Job) named "customers-import".

Details:

Uses JobBuilderFactory to create a job with the name "customers-import".

Configures the job to flow through the single step (step()).

Ends the job configuration.

Returns the configured Job.

@Configuration

@EnableBatchProcessing

@AllArgsConstructor

public class SpringBatchConfig {

private JobBuilderFactory jobBuilderFactory;

private StepBuilderFactory stepBuilderFactory;

private CustomerRepository customerRepository;

@Bean

public FlatFileItemReader<Customer> customerReader() {

FlatFileItemReader<Customer> itemReader = new FlatFileItemReader<>();

itemReader.setResource(new FileSystemResource("src/main/resources/customers.csv"));

itemReader.setName("csv-reader");

itemReader.setLinesToSkip(1);

itemReader.setLineMapper(lineMapper());

return itemReader;

}

private LineMapper<Customer> lineMapper() {

DefaultLineMapper<Customer> lineMapper = new DefaultLineMapper<>();

DelimitedLineTokenizer lineTokenizer = new DelimitedLineTokenizer();

lineTokenizer.setDelimiter(",");

lineTokenizer.setStrict(false);

lineTokenizer.setNames("id", "firstName", "lastName", "email", "gender", "contactNo", "country", "dob");

BeanWrapperFieldSetMapper<Customer> fieldSetMapper = new BeanWrapperFieldSetMapper<>();

fieldSetMapper.setTargetType(Customer.class);

lineMapper.setLineTokenizer(lineTokenizer);

lineMapper.setFieldSetMapper(fieldSetMapper);

return lineMapper;

}

@Bean

public CustomerProcessor customerProcessor() {

return new CustomerProcessor();

}

@Bean

public RepositoryItemWriter<Customer> customerWriter() {

RepositoryItemWriter<Customer> writer = new RepositoryItemWriter<>();

writer.setRepository(customerRepository);

writer.setMethodName("save");

return writer;

}

@Bean

public Step step() {

return stepBuilderFactory.get("step1"). <Customer, Customer>chunk(10)

. reader(customerReader())

. processor(customerProcessor())

. writer(customerWriter())

. build ();

}

@Bean

public Job job () {

return jobBuilderFactory.get("customers-import")

. flow (step ())

. end ()

. build ();

} }

**Writing into CSV file by Multiple Threads**

1. This is done for writing into destination file asynchronously. Multiple threads will be executed making it faster.
2. We can Fastly store with the help of this rather than the manual entering of records and executing it.

Include this in Step Method and write the method block

.taskExecutor(taskExecutor())

@Bean

public TaskExecutor taskExecutor() {

SimpleAsyncTaskExecutor taskExecutor = new SimpleAsyncTaskExecutor();

taskExecutor.setConcurrencyLimit(10);

return taskExecutor;

}

**Additional Configurations in yaml file for data storage**

* The data will be stored in database only when we hit the API at the first time also.

**job:**

**enabled: false**

* The data will be stored in db automatically when we run the application. If we truncate the table and run the application, the data will not be stored. When we hit the API, it will be stored.

**job:**

**enabled: true**

**Writing data into CSV file from the Database**

1. reader Method:

Purpose: Configures a JdbcCursorItemReader to read data from a database.

Details:

Sets the data source for the reader using setDataSource.

Specifies the SQL query to execute using setSql.

Defines a custom RowMapper to map the result set to a Student object.

The mapRow method in the RowMapper implementation is responsible for creating a Student object and populating its fields.

Returns the configured JdbcCursorItemReader for reading data from the database.

1. writer Method:

Purpose: Configures a FlatFileItemWriter to write Student data to a CSV file.

Details:

Sets the resource (output file) for the writer using setResource.

Configures a DelimitedLineAggregator to format the output lines in the CSV file.

Specifies a BeanWrapperFieldExtractor to extract the fields from the Student object.

Sets the header line for the CSV file using setHeaderCallback.

Returns the configured FlatFileItemWriter for writing Student data to a CSV file.

1. executeStep Method:

Purpose: Configures a Spring Batch step (Step) for processing data in chunks.

Details:

Uses StepBuilderFactory to create a step named "executeStep."

Configures a chunk-oriented processing model (<Student, Student>) with a reader (reader()), processor (StuProcessor), and writer (writer()).

Defines the chunk size as 10 (number of items to be processed in each transaction).

Returns the configured Step.

1. processJob Method:

Purpose: Configures a Spring Batch job (Job) to execute the defined step.

Details:

Uses JobBuilderFactory to create a job named "processJob."

Configures the job to increment its run id using incrementer(new RunIdIncrementer()).

Specifies the flow of the job, including the defined step (executeStep) using flow(executeStep()).

Ends the job configuration.

Returns the configured Job.

@Bean

public JdbcCursorItemReader<Student> reader() {

JdbcCursorItemReader<Student> reader = new JdbcCursorItemReader<Student>();

reader.setDataSource(dataSource);

reader.setSql("select id,firstName,lastName,email from csvdbdata");

reader.setRowMapper(new RowMapper<Student>() {

@Override

public Student mapRow(ResultSet rs, int rowNum) throws SQLException {

Student s = new Student();

s.setId(rs.getInt("id"));

s.setFirstName(rs.getString("firstName"));

s.setLastName(rs.getString("lastName"));

s.setEmail(rs.getString("email"));

return s;

}

});

return reader;

}

@Bean

public FlatFileItemWriter<Student> writer() {

FlatFileItemWriter<Student> writer = new FlatFileItemWriter<Student>();

writer.setResource(new FileSystemResource("D://studentdata.csv"));

DelimitedLineAggregator<Student> aggregator = new DelimitedLineAggregator<>();

BeanWrapperFieldExtractor<Student> fieldExtractor = new BeanWrapperFieldExtractor<>();

fieldExtractor.setNames(new String[] { "id", "firstName", "lastName", "email" });

aggregator.setFieldExtractor(fieldExtractor);

writer.setHeaderCallback(writer1-> writer1.write("id,firstName,lastName,email"));

writer.setLineAggregator(aggregator);

return writer;

}

@Bean

public Step executeStep() {

return stepBuilderFactory.get("executeStep"). <Student, Student>chunk(10).reader(reader()).processor(new StuProcessor()).writer(writer()). build();

}

@Bean

public Job processJob()

{

return jobBuilderFactory.get("processJob").incrementer(new RunIdIncrementer()).flow(executeStep()).end().build();

}

