Proyecto Spring Batch

Descripción del uso e implementación de spring batch.

Spring Batch

Juan Carlos Bernal Sandoval

Introducción.

Spring batch es una rama del framework spring el cual nos ayuda a realizar trabajo por lotes de una forma sencilla y eficaz, haciendo uso de chunks o tasklets según se requiera.

Batch proporciona una serie de clases y anotaciones las cuales nos permiten tener un código limpio y sencillo de entender al momento de realizar trabajos por lotes, entiéndase como trabajo por lotes la manipulación de una gran cantidad de datos, por ejemplo, la lectura, tratamiento e inserción de una serie de registros desde un archivo csv a una base de datos.

DESARROLLO.

Como parte de la academia java en nuestra segunda semana exploramos el uso de esta util herramienta, para ello realizamos la implementación de un proyecto batch, el cual realiza la lectura de una serie de registros en un archivo csv el cual cuenta con 50000 registros.

Estos son extraídos directamente desde el archivo csv y filtrados por los campos que se desean resaltar, y solo estos registros serán tomados en cuenta para su inserción en base de datos.

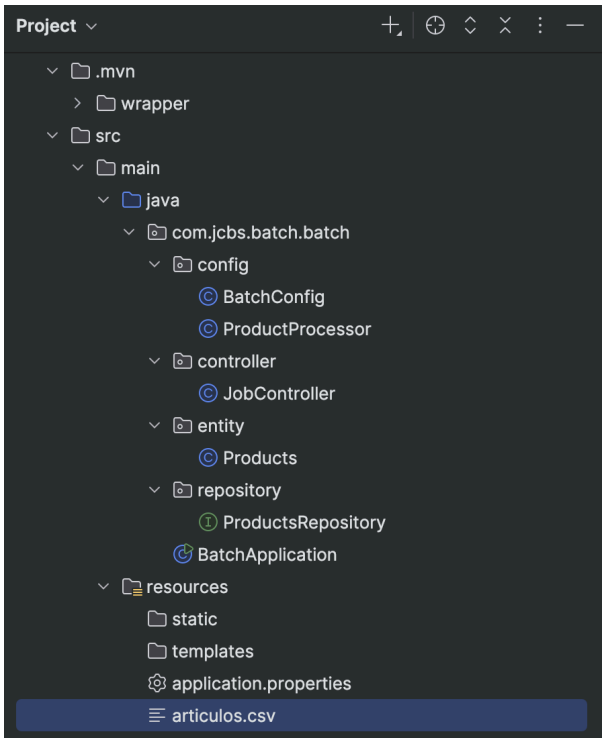
Para este proyecto en particular se utilizó una base de datos postgre, la cual la levantamos mediante docker, a través de un archivo docker compose.

Posterior a ser filtrados los registros fueron insertados en la base de datos, para trabajar con la base de datos se utilizó hibernate.

A continuación se presenta como se realizó la construcción y la implementación del código de nuestro proyecto batch.

Para el proyecto se utilizó una estructura por capas.

- Config: en él se guarda la configuración de nuestro proyecto batch
- Controller: aquí se encuentra nuestro restcontroller, el cual nos será de ayuda para interactuar con nuestra aplicación.
- Entity: en este paquete tendremos las clases que hacen referencia a nuestras tablas de nuestra base de datos.
- Repository: aquí podremos definir aquellas clases e interfaces con las que podemos interactuar con nuestra base de datos.



En nuestra clase JobController podremos definir los endpoints que nos ayudaran a interactuar con nuestra aplicación.

Para ellos hacemos uso de la anotación @RestController, la cual nos indica el tipo de bean con que le queremos asignar a nuestra clase y @RequestMapping, será la base de nuestros endpoints.

Además nos encontramos con la anotación @PostMapping, la cual nos sirve para indicar el verbo http y el endpoint mediante el cual accederemos a esa función al ser llamado.

Además podemos observar que hemos realizado la inyección de dos dependencias, JobLauncher y Job, las cuales nos sirven para realizar la ejecución de nuestro proceso batch,

debemos recordar que JobLauncher es desde donde se lanzan cualquier proceso batch.

```java
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobParametersBuilder;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController  no usages
@RequestMapping("/jobs")
public class JobController {

    @Autowired  1 usage
    private JobLauncher jobLauncher;
    @Autowired  1 usage
    private Job job;

    @PostMapping("/importProducts")  no usages
    public void importCsvToDBJob() {

        try {
            jobLauncher.run(job, new JobParametersBuilder()
                    .addLong( key: "startAt", System.currentTimeMillis())
                    .toJobParameters());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

En la clase BatchConfig realizamos como su nombre lo indica la configuración de nuestro proceso batch, aquí definimos nuestros métodos reader, processor y writer, además de definir el orden en el cual serían ejecutados nuestros procesos, así mismo definiremos los steps por los cuales estará compuesto.

En esta clase destacan los métodos reader() mediante el cual definimos la ruta del archivo desde el cual queremos extraer la información así como la configuración previa a su lectura.

LineMapper() nos ayuda a tokenizar la información extraída de nuestro archivo csv, el cual al ser un archivo separado por comas, le damos un nombre a cada valor extraído desde este separador.

processor() definimos la clase processor, y la mandamos a ejecutar su código, mediante esta procesaremos o en este caso filtraremos los registros extraídos del archivo csv, según nuestro criterio.

writer() este método es de utilidad para realizar una llamada a nuestro repositorio el cual es utilizado para realizar la inserción de nuestros registros en la base de datos.

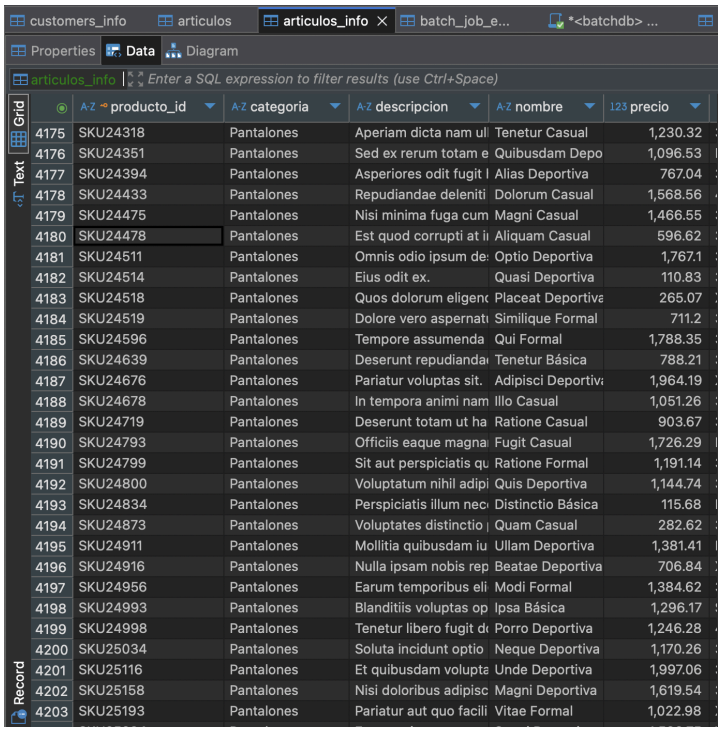
step1() aqui definimos el orden en el cual será ejecutados los métodos anteriormente mencionados.

```java
@Configuration no usages
@EnableBatchProcessing
@AllArgsConstructor
public class BatchConfig {

    private JobBuilderFactory jobBuilderFactory;
    private StepBuilderFactory stepBuilderFactory;
    private ProductsRepository productsRepository;

    @Bean 1 usage
    public FlatFileItemReader<Products> reader() {
        FlatFileItemReader<Products> itemReader = new FlatFileItemReader<>();
        itemReader.setResource(new FileSystemResource( path: "src/main/resources/articulos.csv"));
        itemReader.setName("csvReader");
        itemReader.setLinesToSkip(1);
        itemReader.setLineMapper(lineMapper());
        return itemReader;
    }

    private LineMapper<Products> lineMapper() { 1 usage
        DefaultLineMapper<Products> lineMapper = new DefaultLineMapper<>();

        DelimitedLineTokenizer lineTokenizer = new DelimitedLineTokenizer();
        lineTokenizer.setDelimiter(",");
        lineTokenizer.setStrict(false);
        lineTokenizer.setNames("id", "name", "description", "price","size","category", "stock");

        BeanWrapperFieldSetMapper<Products> f
        fieldSetMapper.setTargetType(Products

        lineMapper.setLineTokenizer(lineToken
        lineMapper.setFieldSetMapper(fieldSet

        return lineMapper;
    }
```

```
org.springframework.batch.item.file.transform.AbstractLineTokenize
public void setNames(
    String... names
)
batch
```

```java
public class BatchConfig {
    @Bean 1 usage
    public ProductProcessor processor(){
        return new ProductProcessor();
    }

    @Bean 1 usage
    public RepositoryItemWriter<Products> writer() {
        RepositoryItemWriter<Products> itemWriter = new RepositoryItemWriter<>();
        itemWriter.setRepository(productsRepository);
        itemWriter.setMethodName("save");
        return itemWriter;
    }

    @Bean 1 usage
    public Step step1(){
        return stepBuilderFactory.get("step1") StepBuilder
                .<Products, Products>chunk( chunkSize: 10) SimpleStepBuilder<Products, Products>
                .reader(reader())
                .processor(processor())
                .writer(writer())
                .taskExecutor(taskExecutor()) AbstractTaskletStepBuilder<SimpleStepBuilder<...>>
                .build();
    }

    @Bean no usages
    public Job runJob() {
        return jobBuilderFactory.get("importProducts") JobBuilder
                .start(step1()) SimpleJobBuilder
                .build();
    }

    @Bean 1 usage
    public TaskExecutor taskExecutor(){
```

Una vez realizada la lectura, y el filtrado de los registros, podemos consultar el resultado de nuestros procesos batch, comprobando la inserción en base de datos de los registros extraídos.



customers_info | articulos | articulos_info | batch_job_e... | *<batchdb> ...

Properties | Data | Diagram

articulos_info | Enter a SQL expression to filter results (use Ctrl+Space)

	producto_id	categoria	descripcion	nombre	precio
4175	SKU24318	Pantalones	Aperiam dicta nam ul	Tenetur Casual	1,230.32
4176	SKU24351	Pantalones	Sed ex rerum totam e	Quibusdam Depo	1,096.53
4177	SKU24394	Pantalones	Asperiores odit fugit	Alias Deportiva	767.04
4178	SKU24433	Pantalones	Repudiandae deleniti	Dolorum Casual	1,568.56
4179	SKU24475	Pantalones	Nisi minima fuga cum	Magni Casual	1,466.55
4180	SKU24478	Pantalones	Est quod corrupti at i	Aliquam Casual	596.62
4181	SKU24511	Pantalones	Omnis odio ipsum de	Optio Deportiva	1,767.1
4182	SKU24514	Pantalones	Eius odit ex.	Quasi Deportiva	110.83
4183	SKU24518	Pantalones	Quos dolorum eligenc	Placeat Deportiva	265.07
4184	SKU24519	Pantalones	Dolore vero aspernatı	Similique Formal	711.2
4185	SKU24596	Pantalones	Tempore assumenda	Qui Formal	1,788.35
4186	SKU24639	Pantalones	Deserunt repudianda	Tenetur Básica	788.21
4187	SKU24676	Pantalones	Pariatur voluptas sit.	Adipisci Deportiva	1,964.19
4188	SKU24678	Pantalones	In tempora animi nam	Illo Casual	1,051.26
4189	SKU24719	Pantalones	Deserunt totam ut ha	Ratione Casual	903.67
4190	SKU24793	Pantalones	Officiis eaque magna	Fugit Casual	1,726.29
4191	SKU24799	Pantalones	Sit aut perspiciatis qu	Ratione Formal	1,191.14
4192	SKU24800	Pantalones	Voluptatum nihil adipi	Quis Deportiva	1,144.74
4193	SKU24834	Pantalones	Perspiciatis illum nec	Distinctio Básica	115.68
4194	SKU24873	Pantalones	Voluptates distinctio	Quam Casual	282.62
4195	SKU24911	Pantalones	Mollitia quibusdam iu	Ullam Deportiva	1,381.41
4196	SKU24916	Pantalones	Nulla ipsam nobis rep	Beatae Deportiva	706.84
4197	SKU24956	Pantalones	Earum temporibus eli	Modi Formal	1,384.62
4198	SKU24993	Pantalones	Blanditiis voluptas op	Ipsa Básica	1,296.17
4199	SKU24998	Pantalones	Tenetur libero fugit d	Porro Deportiva	1,246.28
4200	SKU25034	Pantalones	Soluta incidunt optio	Neque Deportiva	1,170.26
4201	SKU25116	Pantalones	Et quibusdam volupta	Unde Deportiva	1,997.06
4202	SKU25158	Pantalones	Nisi doloribus adipisc	Magni Deportiva	1,619.54
4203	SKU25193	Pantalones	Pariatur aut quo facili	Vitae Formal	1,022.98

Grid | Text | Record

De acuerdo a nuestras necesidades podemos agregar tantos steps y métodos como sean

necesarios.