SPRING BATCH

JAVA ACADEMY - XIDERAL

SEPTEMBER 6, 2024 **AUTHOR: REGINA RODRIGUEZ CAMPO GARRIDO**

Introduction

In this project, I utilized Spring Batch to manage and process data related to books from a library. Spring Batch is a robust framework for batch processing, which allows for the handling of large volumes of data efficiently. It provides essential features like transaction management, chunk processing, and job scheduling, making it ideal for tasks that require high-performance data handling.

Entity

The Customer class represents a book record with attributes such as ID, title, author, year, category, language, and origin country. It uses JPA annotations to map the class to a database table named CUSTOMERS_INFO.

```
package com.javatechie.spring.batch.entity;
3© import lombok.AllArgsConstructor;∏
2 @Entity
3 @Table(name = "CUSTOMERS_INFO")
4 @Data
5 @AllArgsConstructor
6 @NoArgsConstructor
90
       @Column(name = "CUSTOMER_ID")
       private int id;
      @Column(name = "title")
20
      private String title;
      @Column(name = "author")
       private String author;
      @Column(name = "year")
private int year;
@Column(name = "category")
60
      private String category;
       @Column(name = "languaje")
96
       private String languaje;
       @Column(name = "originCountry")
       private String originCountry;
```

Repository

bookrepository extends JpaRepository

```
package com.javatechie.spring.batch.repository;

import com.javatechie.spring.batch.entity.bookEntity;

public interface bookrepository extends JpaRepository<bookEntity,Integer> {

}
```

Spring Batch Configuration

This configuration class sets up the Spring Batch job to read book data from a CSV file and write it into a database.

```
package com.javatechie.spring.batch.config;

  import com.javatechie.spring.batch.entity.bookEntity;

  @Configuration
  @AllArgsConstructor
       private JobBuilderFactory jobBuilderFactory;
       private StepBuilderFactory stepBuilderFactory;
       private bookrepository customerRepository;
           FlatFileItemReader<>bookEntity> itemReader = new FlatFileItemReader<>>();
itemReader.setResource(new FileSystemResource("src/main/resources/books_list.csv"));
itemReader.setName("csvReader");
            itemReader.setLinesToSkip(1);
itemReader.setLineMapper(lineMapper());
            return itemReader;
       private LineMapper<bookEntity> lineMapper() { //mapped
    DefaultLineMapper<bookEntity> lineMapper = new DefaultLineMapper<>();
            DelimitedLineTokenizer lineTokenizer = new DelimitedLineTokenizer();
            lineTokenizer.setDelimiter(",");
lineTokenizer.setStrict(false);
lineTokenizer.setNames("id", "tilte", "author", "year", "category", "languaje", "originCountry");
            BeanWrapperFieldSetMapper<bookEntity> fieldSetMapper = new BeanWrapperFieldSetMapper<>();
            fieldSetMapper.setTargetType(bookEntity.class);
            lineMapper.setLineTokenizer(lineTokenizer);
            lineMapper.setFieldSetMapper(fieldSetMapper);
return lineMapper;
```

```
@Bean
public Step step1() {
    Supplier<RepositoryItemWriter<bookEntity>> supplier = () -> {
         RepositoryItemWriter<bookEntity> writer = new RepositoryItemWriter<>();
         writer.setRepository(customerRepository);
         writer.setMethodName("save");
         return writer;
    return stepBuilderFactory.get("csv-step")
              .<bookEntity, bookEntity>chunk(10)
              .reader(reader())
              .processor((ItemProcessor<bookEntity, bookEntity>) cliente -> {
                  if(cliente.getLanguaje().equals("French"))
                  return cliente; return null;
             .writer(supplier.get())
              .taskExecutor(taskExecutor())
              .build();
@Bean
public Job runJob() { //import csx
    return jobBuilderFactory
             .get("importBooks")
.flow(step1())
              .end()
              .build();
@Bean
public TaskExecutor taskExecutor() {
    SimpleAsyncTaskExecutor asyncTaskExecutor = new SimpleAsyncTaskExecutor();
    asyncTaskExecutor.setConcurrencyLimit(10);
    return asyncTaskExecutor;
```

Controller

The JobController exposes a REST that, when accessed, initiates the batch job responsible for importing book data.

```
package com.javatechie.spring.batch.controller;
import org.springframework.batch.core.Job;
 @RestController
 @RequestMapping("/jobs")
     @Autowired
     private JobLauncher jobLauncher;
     @Autowired
     private Job job;
     @PostMapping("/importBooks")
     public void importCsvToDBJob() {
         .toJobParameters();
             jobLauncher.run(job, jobParameters);
         } catch (JobExecutionAlreadyRunningException |
                 JobInstanceAlreadyCompleteException |
JobParametersInvalidException e) {
             e.printStackTrace();
     }
```

Outputs Empty table

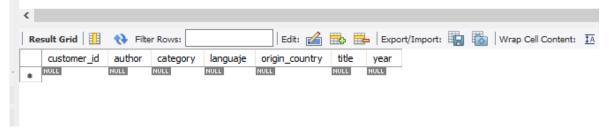


Table of books in English.



Table of books in French.

