Spring BAtch App To convert csv file data to MongoDB Documents

==========

writer : MongoItemWriter<T> reader :: FlatFileItemReder<T>

TopBrains.csv

7777

<T>:: Document class (java bean with @Document) <T>:: Model class (java bean) FlatFileItemReader<Exam Result>

1.specify csv file location setResource(....)

2. specify LineMapper with LineTokenizer setLineMapper(...)

Exam Result obj

Document object

id:

chunk size:3

dob:

percentage: semester.

ExamResultItemProcessor<Exam Result,

Exam Result>

MongoItemWriter<Exam Result> batch of Document objs

1. specify collection name writer.setCollection("SuperBrains")

3. specify FieldSetMapper to covert

each Line of CSV file to Model class object

Exam Result object

id: dob:

(Model class object)

percentage: semester:

required staters ::

X Lombok

X Spring Batch

3.x stepup

X Spring Data MongoDB

X h2

(InMemory DB acting

=>make sure that MongoDB, Studio3T(GUI DB tool) softwares are installled and the logical DB is created.

as JobRepository)

example app1 (writing date value to mongodb collection as Stirng value)

2. specify the MongoTemplate class obj to writer which is created through AutoConfiguration process.

@Autowired

private MongoTemplate template writer.setTemplate(template);

3. MongoTemplate internally compltes batch of documents data wrting to

MongoDB s/w to the specified collection

In MongoDB NoSQL DB s/w

Collection -----> db table

document ---> db table record

collection name

in mongo DB

MongoDB Db/w SuperBrains (collection)

//documents

✓ BatchApp04-CSVtoMongoDB [boot]

Spring Elements

#src/main/java

✓ com.nt

> BatchApp04CsVtoMongoDbApplication.java

com.nt.config

> BatchConfig.java

com.nt.document

> ExamResult.java

com.nt.listener

> JobMonitoringListener.java

com.nt.processor

> ExamResultitemProcessor.java

com.nt.runner

> SpringBatchRunner.java

#src/main/resources

application.properties

> # src/test/java

JRE System Library [JavaSE-11]

>

>

>

src

Maven Dependencies

> target

WHELP.md

mvnw

mvnw.cmd

M pom.xml

=>for Java bean class properties, always prefer taking wrapper data types becoz they hold "null" value when Model cum Document class no value is given.. where as simple data type properties holds 0,0.0 as the values which will inserted to db table while inserting the record.

======

```
//Model and Document class
package com.nt.document;
import java.time.LocalDate;
import org.springframework.data.annotation.Id; import
org.springframework.data.mongodb.core.mapping.Doc
ument;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Document
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Exam Result {
@Id
private Integer id;
private String dob; private Float percentage; private Integer semester;
Exam ResultProcessor.java
package com.nt.processor;
import org.springframework.batch.item.Item Processor; import org.springframework.stereotype.Component;
import com.nt.model.Exam Result;
@Component
public class Exam ResultProcessor implements ItemProcessor<Exam Result, Exam Result> {
@Override
public Exam Result process (Exam Result item) throws Exception {
if(item.getPercentage()>=95.0)
return item;
else
return null;
```

}

}

BatchConfig.java

}

package com.nt.config;

import org.springframework.batch.core.Job;

import org.springframework.batch.core.Job ExecutionListener;

import org.springframework.batch.core.Step;

import org.springframework.batch.core.configuration.annotation.EnableBatch Processing; import org.springframework.batch.core.configuration.annotation.JobBuilderFactory; import org.springframework.batch.core.configuration.annotation.StepBuilderFactory; import org.springframework.batch.core.launch.support.RunIdIncrementer;

import org.springframework.batch.item.data.MongoItemWriter;

import org.springframework.batch.item.file.FlatFileItemReader;

import org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper;

import org.springframework.batch.item.file.mapping.DefaultLineMapper;

import org.springframework.batch.item.file.transform.DelimitedLineTokenizer;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.io.FileSystem Resource;

import org.springframework.data.mongodb.core.MongoTemplate;

import com.nt.document.Exam Result;

import com.nt.listener.JobMonitoringListener;

import com.nt.processor.Exam ResultItemProcessor;

@Configuration

@EnableBatchProcessing

public class BatchConfig {

@Autowired

private JobBuilderFactory jobFactory; @Autowired

private StepBuilderFactory stepFactory; @Autowired

private MongoTemplate template;

//listener

//ExamResultProcessor.java

package com.nt.processor;

application.properties

spring batch settings

spring.batch.job.enabled=true

spring.batch.jdbc.initialize-schema-always

#MongoDB settings

spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

spring.data.mongodb.database=NTSPBMS714DB1 spring.data.mongodb.username=testuser
spring.data.mongodb.password=testuser

import org.springframework.batch.item.ItemProcessor; import org.springframework.stereotype.Component;

import com.nt.document.Exam Result;

@Component

public class Exam ResultProcessor implements ItemProcessor<Exam Result, Exam Result> {

@Override

public ExamResult process (Exam Result item) throws Exception {

if(item.getPercentage()>=95.0f)

}

@Bean

public JobExecutionListener createListener() {

return new JobMonitoringListener();

}

//processor

@Bean

public ExamResultItemProcessor createProcessor() {

return new Exam ResultItemProcessor();

}

@Bean

return item;

else

return null;

}

}

public FlatFileItemReader<Exam Result> createReader(){

FlatFileItemReader<Exam Result> reader=new FlatFileItemReader<>(); reader.setResource(new FileSystemResource("e:/csvs/TopBrains.csv"));

@Bean(name="ffiReader")

public FlatFileItemReader<Exam Result> createReder(){ return new FlatFileItemReaderBuilder<Exam Result>() .name("file-reader")

.resource(new ClassPathResource("TopStudents.csv")) .delimited().delimiter(",")

.names("id", "dob","percentage","semester") .targetType(Exam Result.class) .build();

reader.setLineMapper(new DefaultLineMapper<Exam Result>() {{

```
setLineTokenizer(new DelimitedLineTokenizer() {{
setDelimiter(",");
setNames("id", "dob","percentage","semester");
}});
setFieldSetMapper(new BeanWrapperFieldSetMapper<Exam Result>() {{ setTargetType(Exam Result.class);
}
}});
}});
return reader;
}
//writer
@Bean
public MongoItemWriter<Exam Result> createWriter(){
MongoItemWriter<Exam Result> writer=new MongoItemWriter<>();
writer.setCollection("SuperBrains");
writer.setTemplate(template);
return writer;
}
//step
@Bean(name="step1")
public Step createStep1() {
return stepFactory.get("step1")
}
.<Exam Result, Exam Result>chunk(3)
.reader(createReader())
.writer(createWriter())
.processor(createProcessor())
.build();
@Bean(name="job1")
public Job createJob1() {
return jobFactory.get("job1")
.incrementer(new RunIdIncrementer())
.listener(createListener())
.start(createStep1())
.build();
}
Time
```

Example App2 (date value as java.time.LocalDate in @Document class)

H

```
@Bean(name="job1")
public Job createJob1() {
}
return jobFactory.get("job1")
.incrementer(new RunIdIncrementer())
.listener(listener)
.start(createStep1())
.build();
```

BatchApp05-CSVtoMongoDB1 [boot] Spring Elements #src/main/java

com.nt

> BatchApp04CsVtoMongoDbApplication.java

com.nt.config

BatchConfig.java

com.nt.document

> DOExamResult.java

>com.nt.listener

com.nt.model

> DIExamResult.java

com.nt.processor

> ExamResultitemProcessor.java

com.nt.runner

> SpringBatchRunner.java

#src/main/resources

application.properties

>

src/test/java

> JRE System Library [JavaSE-11]

>

Maven Dependencies

> src

>

target

WHELP.md

mvnw

mvnw.cmd Mpom.xml

//Model class

==========

//Model class package com.nt.model;

import java.util.Date;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document; import org.springframework.format.annotation.DateTimeFormat;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor @AllArgsConstructor

public class IExam Result { @Id

private Integer id;

private String dob;

private Float percentage;

private Integer semester;

Document class

//Document class

package com.nt.document;

import java.time.LocalDate;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Document @Data

@NoArgsConstructor

@AllArgsConstructor

public class OExam Result {

@Id

private Integer id;

Time

private LocalDate dob;

private Float percentage; private Integer semester;

BatchConfig.java

package com.nt.config;

}

import org.springframework.batch.core.Job;

}

Processor

import org.springframework.batch.core.JobExecutionListener;

import org.springframework.batch.core.Step;

import org.springframework.batch.core.configuration.annotation.Enable Batch Processing; import org.springframework.batch.core.configuration.annotation.JobBuilderFactory; import org.springframework.batch.core.configuration.annotation.StepBuilderFactory; import org.springframework.batch.core.launch.support.RunIdIncrementer; import org.springframework.batch.item.data.MongoItemWriter;

import org.springframework.batch.item.file.FlatFileItemReader; import org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper; import org.springframework.batch.item.file.mapping.DefaultLineMapper; import org.springframework.batch.item.file.transform.DelimitedLineTokenizer;

import org.springframework.beans.factory.annotation.Autowired;

//ExamResultProcessor.java package com.nt.processor;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import org.springframework.batch.item.ItemProcessor; import org.springframework.stereotype.Component;

import com.nt.document.OExam Result;

import com.nt.model.IExamResult;

@Component

import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration; import org.springframework.core.io.FileSystemResource;

import org.springframework.data.mongodb.core.MongoTemplate;

import com.nt.document.OExam Result;

import com.nt.listener.JobMonitoringListener;

import com.nt.model.IExam Result;

import com.nt.processor.Exam ResultitemProcessor;

@Configuration

@EnableBatch Processing

public class BatchConfig {

@Autowired

private JobBuilderFactory jobFactory;

@Autowired

private StepBuilderFactory stepFactory; @Autowired

private MongoTemplate template;

```
//listener
@Bean
public JobExecutionListener createListener() {
return new JobMonitoringListener();
}
//processor
@Bean
public Exam ResultItemProcessor createProcessor() {
return new Exam ResultItemProcessor();
}
@Bean
public FlatFileItemReader<IExam Result> createReader(){
FlatFileItemReader<IExam Result> reader=new FlatFileItemReader<>();
reader.setResource(new FileSystemResource("e:/csvs/TopBrains.csv"));
reader.setLineMapper(new DefaultLineMapper<IExam Result>() {{
setLineTokenizer(new DelimitedLineTokenizer() {{
setDelimiter(",");
setNames("id", "dob","percentage","semester");
}});
setFieldSetMapper(new BeanWrapperFieldSetMapper<IExam Result>() {{ setTargetType(IExam Result.class);
}});
}});
return reader;
}
//writer
@Bean
public MongoItemWriter<OExam Result> createWriter(){
}
//step
MongoItemWriter<OExam Result> writer=new MongoItemWriter<>();
writer.setCollection("SuperBrains1");
writer.setTemplate(template);
return writer;
@Bean(name="step1")
public Step createStep1(JobRepository jobRepository, Platform Transaction Manager transaction Manager) {
return new StepBuilder("step1",jobRepository)
}
```

```
<Exam Result,Exam Result>chunk(3, transactionManager)
.reader(createReader())
.processor(processor)
.writer(createWriter())
.build();
}
//Job obj
@Bean(name="job1")
public Job createJob(Job Repository jobRepository, Step step1) {
return new JobBuilder("job1",jobRepository)
.incrementer(new RunIdIncrementer())
.listener(listener)
.start(step1)
.build();
```

Flow of execution (For any Spring Boot batch application)

==========

=>Run the application ---> main(-) of main class executes ---> SpringApplication.run(-) method internally creates the

```
public class Exam ResultProcessor implements ItemProcessor<IExam Result, OExam Result> {
@Override
public OExamResult process(IExam Result item) throws Exception {
if(item.getPercentage()>=95.0f) {
OExam Result result=new OExamResult();
result.setId(item.getId());
result.setDob(LocalDateTime.parse(item.getDob(),DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss.S")));
result.setPercentage (item.getPercentage());
result.setSemester(item.getSemester());
return result;
}
else
return null;
}
}
```

IOC container and bootstraps (starts app's execution)

=> The @ComponentScan annotation of @SpringBootApplication scans the current pkg and sub pkgs classes

for stereo type annotations and finds in Processor class, runner class (@Component) and BatchConfig class

(@Configuration) (all these are singleton

=> The Object of Configuration class will be created automatically..

=> @EnableAutoConfiguration annotation of @Configuration class creates the following objects

as spring beans based on jar files that are added as part AutoConfiguration process

JobBuilderFactory

StepBuilderFactory JobLauncher

JobReposutory

3.x

2.x

Platform Transaction MAnager

In both Versions

and etc..

to

MongoTemplate

by default)

=> Pre-instantation of singlescope spring beans takes place and injections on spring beans will be completed in this process.

scope

-> @Bean methods having singleton will be executed automatically

-> All @Autowired Injections will be completed.

=> Keeps the singleton scope spring bean class objects in the Internal cache IOC container

=> Runner Ruins run() method executes and this method calls lancher.run(job,params) -->

In

This process job object is taken ---> from job Step obj is taken ---> from Step object reader, writer and processor

will be taken will be executed as per chunksize to complete the batch jobs.

and

In application.properties

spring.batch.job.enabled=true

on

true :: Indicates run the job the app's startup irrespective of launcher.run(-) method is called or not

false ::Indicates run the job only when launcher.run(-) is called.

To enable scheduling on Batch Processing

==========

step1) place @EnableScheduling on the top of main class

step2) place @Scheduled annotation on the top no param method in any spring bean class

//service class

package com.nt.runner;

```
import java.util.Random;

import org.springframework.batch.core.Job;

import org.springframework.batch.core.JobExecution; import
org.springframework.batch.core.JobParameters; import
org.springframework.batch.core.JobParametersBuilder; import
org.springframework.batch.core.launch.JobLauncher; import
org.springframework.beans.factory.annotation.Autowired; import
org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Service;

@Service

public class Batch TestService {

@Autowired

private JobLauncher launcher;

@Autowired

private Job job;

}

@Scheduled(cron = "0 23/28 ***")

public void run() throws Exception {

System.out.println("BatchTestRunner.run()");

//create JobParameters

JobParameters params= new JobParametersBuilder().addLong("run.id",new
Random().nextLong(1000)).toJobParameters();

}

// run the job

JobExecution execution=launcher.run(job, params);

System.out.println("launch method is called");

System.out.println("Job Execution status ::"+execution.getExitStatus());
```