## Developing the Spring Boot Batch Application using sprong boot 3.x setup

**=> In spring boot batch 5.x (part of spring boot 3.x) we are not getting the following objects through AutoConfiguration**

**a) StepBuilderFactory obj**

**b) JobBuilderFactory obj**

**=> As alternate, we need to use 2 -param constructors of**

**StepBuilder, JobBuilder classes to create the objects**

```
// Sample with v4 (Spring boot 2.x)
@Configuration
@EnableBatch Processing
public class MyStepConfig {
@Autowired
private StepBuilderFactory stepBuilderFactory;
@Bean
public Step myStep() {
return this.stepBuilderFactory.get("myStep")
.tasklet(..) // or .chunk() .build();
}
}
```

**=>Spring boot 3.x incorporates spring batch 5.x**

**=>Spring boot 2.x incorporates spring batch 4.x**

```
// Sample with v5 (Spring boot 3.x code)
@Configuration
// @EnableBatch Processing (Not required in 3.x code)
public class MyStepConfig {
@Bean
public Tasklet myTasklet() {
}
return new MyTasklet();
@Bean
public Step myStep(Job Repository jobRepository, Tasklet myTasklet, PlatformTransactionManager transactionManager) { return new StepBuilder("myStep", jobRepository)
.tasklet(myTasklet, transactionManager) // or .chunk(chunkSize, transactionManager)
.build();
}
}
// Sample with v4
```

```
@Configuration
@EnableBatchProcessing
public class MyJobConfig {
@Autowired
private JobBuilderFactory jobBuilderFactory;
@Bean
public Job myJob(Step step) {
return this.jobBuilderFactory.get("myJob")
.start(step)
.build();
}
}
// Sample with v5
@Configuration
// @EnableBatch Processing
public class MyJobConfig {
@Bean
(not required)
public Job myJob(Job Repository jobRepository, Step step) {
return new JobBuilder("myJob", jobRepository)
.start(step)
.build();
}
}
```

(Excel file data)

(spring batch 5.x)

Example App on converting CSV file data to DB table data using spring boot 3.x setup

note:: In spring boot 3.x (spring batch 5.x) we get following objects through AutoConfiguration process a) JobLauncher obj b) JobRepository obj c) Platform TransactionManager object

|--->To run the job

|--->keep track

job execution activities

|---> for commit and rollback activities

employeesInfo.csv 101,raja,9000,hyd 102,ramesh,9000, vizag

FlatFileItemReader<Employee>

1. specify Source location (csv file location)

2. speficy LineMapper to

**get each Line from csv file**

**101,raja,9000,hyd**

**3. specify Line Tokenizer to**

get tokens/value from the line

**based on given delimiter (,)**

**101 raja 9000**

**hyd**

**4. Specify FieldSetMapper to convert Line content into Model class obj by**

**(output)**

**Employee class obj eno:101 ename:raja eadd:hyd**

**chunksize:3**

**EmployeeInfoItemProcessor<Employee,**

**Employee>**

**=>calculates grossSalary and netSalary only when basicSalry is>=5000**

**basicSalary: 90000 gressSalary: 12100 netSalary: 10800 (comes as the List<Employee> obj**

**giving names to line tokens and by mapping line tokens**

**to Model cass obj properties**

**(input)**

**Employee class obj**

**eno:101**

ename:raja

**eadd:hyd**

**basicSalary: 90000**

gressSalary: null

**netSalary: null**

**Reader ::: FlatFileItemReader<T>**

**Writer ::: RepositoryItem Writer<T>**

**Example App**

=====

**to**

**usecase1 :: census info comes Central ministry form every village/town/city in the form of csv files (excel files) So the Batch app of central ministry should process data by categoringizing and validating before writing to oracle db tables**

**usecase2: While conducting election survery the filed workers gets people's pluse in the form of excel sheets so we should process and store in db table to generate certains report or graphs (Survery report or graph)**

**step1) create spring boot starter Project adding following starters.. spring batch, oracle driver, lombok api, Spring data jpa**

**step2) place csv file (source file) in main/src/resources folder**

**we can genernate csv file with mock /example data using https://www.mockaroo.com/**

**step3) create the fllowing pkgs in src/main/java folder**

#src/main/java

>com.nt

com.nt.config

com.nt.listener

>com.nt.model

com.nt.processor

com.nt.runner

#src/main/resources

application.properties

Employee_Info.csv

:::

eno

Field Name

Type

**Options**

**number**

blank:

0% Σ min:

**max:**

**1000**

decimals:

0 blank: 0%

**Σ X**

ename

**First Name**

blank: 0%

Σ X

M

X

Number

min:

10000

**max:**

100000 decimals:

0 blank:

0% ΣX

eadd

City

salary

ADD ANOTHER FIELD

blank:

0%

---> select csv format, exclude headers, select rowscount as 1000 -->

--> preview and download data..

step4) add the following entries in application.properties file

spring.application.name=BootBatch Proj03-CSVFileToOracle DB

# Oracle DataSource configuration

spring.datasource.driver-class-name-oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system

spring.datasource.password=tiger

# jpa hubernate properties

spring.jpa.database-platform=org.hibernate.dialect.Oracle Dialect

spring.jpa.hibernate.ddl-auto-update

spring.jpa.show-sql=true

#BatchProcessing properties

spring.batch.jdbc.initialize-schema-always spring.batch.job.enabled=false

step5) Develop the Listener class (same previous App)

//Listener class

==========

```
package com.nt.listener;
import java.util.Date;
import org.springframework.batch.core.Job Execution;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.stereotype.Component;
@Component("jobListener")
public class JobMonitoringListener implements JobExecutionListener {
private long start, end;
@Override
public void beforeJob(JobExecution jobExecution) {
System.out.println("JobMonitoringListener:: Job Started at::"+new Date());
```

```
start=System.currentTimeMillis();
}
@Override
public void afterJob(JobExecution jobExecution) {
System.out.println("Job Exit status ::"+jobExecution.getExitStatus()+" at-->"+ new Date());
end=System.currentTimeMillis();
System.out.println("Job Excution time is ::"+(end-start)+" ms");
}
}
```

step5) Develop the Processor class

```
package com.nt.processor;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.stereotype.Component;
import com.nt.model.Employee;
@Component("empProcessor")
public class EmployeeItem Processor implements ItemProcessor<Employee, Employee> {
@Override
public Employee process(Employee item) throws Exception {
if(item.getSalary()>=50000.0) {
}
return null; }//method
item.setGrossSalary(item.getSalary()+(item.getSalary()*0.4));
item.setNetSalary(item.getGrossSalary()-(item.getGrossSalary()*0.2));
return item;
}//class
```

step6) Develop the Entity class

```
package com.nt.model;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table; import lombok.Data;
@Data
@Entity
@Table(name="BATCH_EMPLOYEE")
public class Employee {
@Id
private Integer empno; @Column(length = 40)
```

```
private String empname; @Column(length = 40) private String empaddrs;
private Double salary;
private Double grossSalary;
private Double netSalary;
}
step7) Develop the Repository Interface
package com.nt.repository;
import org.springframework.data.repository.Crud Repository;
import com.nt.model.Employee;
public interface IEmployee Repository extends Crud Repository<Employee, Integer> {
}
step8) Develop the BatchConfig.java as Configuration class
//BatchConfig.java
package com.nt.config;
import javax.sql.DataSource;
import org.aspectj.apache.bcel.Repository; import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.Enable Batch Processing;
import org.springframework.batch.core.job.builder.JobBuilder;
import org.springframework.batch.core.launch.support.RunId Incrementer;
import org.springframework.batch.core.repository.Job Repository;
import org.springframework.batch.core.step.builder.StepBuilder;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.item.data.RepositoryItemWriter;
import org.springframework.batch.item.data.builder.RepositoryItemWriterBuilder;
import org.springframework.batch.item.database.JdbcBatchItemWriter;
import org.springframework.batch.item.database.builder.JdbcBatchItemWriterBuilder;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPath Resource;
import org.springframework.transaction.PlatformTransactionManager;
import com.nt.listener.JobMonitoringListener;
import com.nt.model.Employee;
import com.nt.processor.EmployeeItem Processor;
```

```
import com.nt.repository.IEmployee Repository;
@Configuration //@EnableBatch Processing
public class BatchConfig {
@Autowired
(This annotation is not required)
private JobMonitoringListener listener;
@Autowired
private EmployeeItem Processor processor;
@Autowired
private DataSource ds;
@Autowired
private IEmployee Repository empRepo;
@Bean(name="reader") //using Builder class + method chaining
public FlatFileItemReader<Employee> createReader(){
return new FlatFileItemReaderBuilder<Employee>()
.name("file-reader")
.resource(new ClassPathResource("Employee_Info.csv")) .delimited()
.names("empno","empname", "empaddrs","salary") .targetType(Employee.class)
.build();
@Bean(name="writer")
public RepositoryItemWriter<Employee> createWriter(){
return new RepositoryItemWriterBuilder<Employee>()
}
.repository(empRepo)
.methodName("save")
.build();
//create the Step obj
@Bean(name="step1")
public Step myStep(JobRepository jobRepository, Platform TransactionManager transactionManager) {
return new StepBuilder("step1",jobRepository)
.<Employee, Employee>chunk(3,transactionManager)
.reader(createReader())
.processor(processor)
.writer(createWriter())
.build();
@Bean(name="job1")
public Job createJob(JobRepository jobRepository,Step step1) {
```

```
return new JobBuilder("job1",jobRepository)
.incrementer(new RunIdIncrementer())
.listener(listener)
.start(step1)
.build();
}
}
```

step8) Develope the Runner class

```
package com.nt.runners;
import java.util.Date;
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.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
@Component
public class JobLaunch TestRunner implements CommandLineRunner {
}
@Autowired
private JobLauncher launcher;
@Autowired
private Job job;
@Override
public void run(String... args) throws Exception {
try
{
JobParameters params=new JobParametersBuilder().addDate("startDate", new Date()).toJobParameters();
JobExecution execution=launcher.run(job, params);
System.out.println("Job Execution Status ::"+execution.getExitStatus());
}
catch(Exception e) {
e.printStackTrace();
}
```

step9) Run the Application

RepositoryItemWrier

List<Employee > object

**(List of Employee objs as inpu to writer)**

**=> create Object for RepositoryItemWriter class**

**=> specify the custom repository obj name (indirectly datasource obj) => specify the repository method name ("save") method note:: This save(-) inserts the batch of records to db table**

**each time**

**Oracle DB table batch_employee**

:

...

... ... ....

...