CHAPTER#3: SPRING BOOT EMAIL/Artemis Project

1. **Introduction:--**

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

</dependency>

<dependency>

<groupId>com.github.librepdf</groupId>

<artifactId>openpdf</artifactId>

<version>1.3.8</version>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

=>Spring Boot mail API is defined on top of JAVA Mail API which reduces common lines of code for Email Application.

=> spring-boot-starter-activemq dependency gives the API defined to consume the values from Artemis Queue using JMS Template

=> Open PDF dependency helps us to create a PDF using data

=>To implement Email Service,JMS Consumer we need to provide below keys in application.properties file a

=>Example given with Outlook Server Details.

**Application.properties:--**

server.port=1220

artemis.brokerUrl=tcp://localhost:61618

artemis.user=admin

artemis.password=admin

jms.out.queue.destination=out-queue

#jms.in.queue.destination=in-queue

spring.artemis.mode=native

artemis.enable.ssl=false

artemis.connection.cache.size=5

##email Properties##

spring.mail.host=smtp.office365.com

spring.mail.password=

spring.mail.port=587

spring.mail.username=

spring.mail.properties.mail.smtp.starttls.enable=true

security.require-ssl=true

spring.mail.properties.mail.smpt.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

enable: true

# UML Diagram:--

Spring Boot AutoConfiguration does, 1> Get all Required Jars to Project. 2> Find for keys properties/yml

(auto-load based on prefix =spring.mail)

3> \*\*\*Writes Configuration for JavaMailSenderImpl Bean (i.e @Bean) 4> Creates Email session with Mail sender.

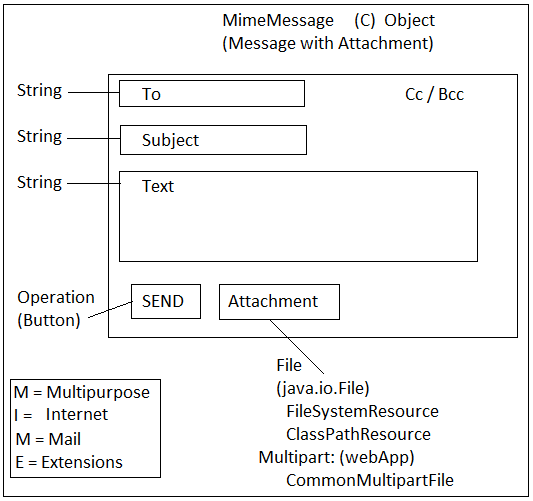
=Programmer not required defining bean for JavaMailSenderImpl in AppConfig (like Spring f/w).

=>But need to provide all inputs like host, port.. using Properties file.

=>We need to construct one object i.e “**MimeMessage**” which is equals to “New Message in Mail Application”.

=>Spring f/w has provided one helper class “**MimeMessageHelper**”, to construct object for “**MimeMessage**”.

=>To Load Files (Attachments) in case of stand-alone mode application in spring or Boot use “**Resource**” impl classes. For Example



* 1. **FileSystemResource :--** This is used to load one file from System drives. (ex:- d:/images/mydata).
  2. **ClassPathResource:--** If file is available in src/main/resources folder then use this concept.

**Step#1:--** Create Spring Boot starter Application using all dependencies: Java Mail Sender

**Step#2:--** Provide details in application.properties/yml host, port = To identify Mail service Provider.

user, pwd = To do login from boot Application. mail properties = Extra information to server.

**Step#3:--** Define Config and utility classes

# #28. Folder Structure of Spring Boot Mail Service:--



1. **ArtemisConfig.java:--**

@Configuration

**public** **class** ArtemisConfig {

**private** **static** **final** Logger LOGGER = LoggerFactory.getLogger(ArtemisConfig.**class**);

@Value("${artemis.enable.ssl}")

**private** String sslEnabled;

@Value("${artemis.brokerUrl}")

**private** String brokerUrl;

@Value("${artemis.user}")

**private** String user;

@Value("${artemis.password}")

**private** String password;

@Value("${artemis.connection.cache.size}")

**private** **int** sessionCacheSize;

@Bean(name = "jmsConnectionFactory")

**public** CachingConnectionFactory cachingConnectionFactory() {

ActiveMQConnectionFactory activeMQConnectionFactory = **new** ActiveMQConnectionFactory();

activeMQConnectionFactory.setBrokerURL(brokerUrl);

activeMQConnectionFactory.setUserName(user);

activeMQConnectionFactory.setPassword(password);

activeMQConnectionFactory.setTrustedPackages(Collections.singletonList("com.rama.artemis"));

CachingConnectionFactory cachingConnectionFactory = **new** CachingConnectionFactory(activeMQConnectionFactory);

cachingConnectionFactory.setSessionCacheSize(sessionCacheSize);

**return** cachingConnectionFactory;

}

}

**ArtemisConsumer.java**

@Component

**public** **class** ArtemisConsumer {

@Autowired

**private** JavaMailSender mailSender;

@Value("${spring.mail.username}")

**private** String user;

@JmsListener(destination = "${jms.out.queue.destination}")

**public** **void** receive(String msg) **throws** Exception{

String[] arrOfStr = msg.split(":");

System.out.println("Consumer Got Message: " + msg);

String from = user;

String to = arrOfStr[0];

/\*SimpleMailMessage message = new SimpleMailMessage();

message.setFrom(from);

message.setTo(to);

message.setSubject("Approved Loan Details");

message.setText(arrOfStr[1]);

mailSender.send(message);\*/

List<String> listUsers = **new** ArrayList<>();

listUsers.add("Hey");

listUsers.add("Kumar");

listUsers.add("Welcom");

listUsers.add("to Lender");

listUsers.add("Thanks for taking loan");

listUsers.add(arrOfStr[1]);

MimeMessage message = mailSender.createMimeMessage();

MimeMessageHelper helper = **new** MimeMessageHelper(message, **true**);

UserPDFExporter exporter = **new** UserPDFExporter(listUsers);

//now write the PDF content to the output stream

ByteArrayOutputStream outputStream = **new** ByteArrayOutputStream();

exporter.writePdf(outputStream);

**byte**[] bytes = outputStream.toByteArray();

OutputStream out = **new** FileOutputStream("out.pdf");

out.write(bytes);

out.close();

//send off the email

helper.setSubject("Here's your e-book");

helper.setFrom(from);

helper.setTo(to);

helper.setText("<b>Dear friend</b>,<br><i>Please find the book attached.</i>", **true**);

FileSystemResource file = **new** FileSystemResource(**new** File("out.pdf"));

helper.addAttachment("FreelanceSuccess.pdf", file);

mailSender.send(message);

//producer.send("Consumer send back:"+msg);

}

**User PDF Exporter.java**

**public** **class** ArtemisConfig {

package com.rama.artemis.consumer;

import java.awt.Color;

import java.io.IOException;

import java.io.OutputStream;

import java.util.List;

import javax.servlet.http.HttpServletResponse;

import com.lowagie.text.\*;

import com.lowagie.text.pdf.\*;

public class UserPDFExporter {

private List<String> listUsers;

public UserPDFExporter(List<String> listUsers) {

this.listUsers = listUsers;

}

private void writeTableHeader(PdfPTable table) {

PdfPCell cell = new PdfPCell();

cell.setBackgroundColor(Color.BLUE);

cell.setPadding(5);

Font font = FontFactory.getFont(FontFactory.HELVETICA);

font.setColor(Color.WHITE);

cell.setPhrase(new Phrase("User ID", font));

table.addCell(cell);

cell.setPhrase(new Phrase("E-mail", font));

table.addCell(cell);

cell.setPhrase(new Phrase("Full Name", font));

table.addCell(cell);

cell.setPhrase(new Phrase("Roles", font));

table.addCell(cell);

cell.setPhrase(new Phrase("Enabled", font));

table.addCell(cell);

}

private void writeTableData(PdfPTable table) {

for (String user : listUsers) {

table.addCell(String.valueOf(user));

table.addCell(user);

table.addCell(user);

table.addCell(user);

table.addCell(String.valueOf(user));

}

}

public void export(HttpServletResponse response) throws DocumentException, IOException {

Document document = new Document(PageSize.A4);

PdfWriter.getInstance(document, response.getOutputStream());

document.open();

Font font = FontFactory.getFont(FontFactory.HELVETICA\_BOLD);

font.setSize(18);

font.setColor(Color.BLUE);

Paragraph p = new Paragraph("List of Users", font);

p.setAlignment(Paragraph.ALIGN\_CENTER);

document.add(p);

PdfPTable table = new PdfPTable(5);

table.setWidthPercentage(100f);

table.setWidths(new float[] {1.5f, 3.5f, 3.0f, 3.0f, 1.5f});

table.setSpacingBefore(10);

writeTableHeader(table);

writeTableData(table);

document.add(table);

document.close();

}

public void writePdf(OutputStream outputStream) throws Exception {

Document document = new Document();

PdfWriter.getInstance(document, outputStream);

document.open();

Paragraph paragraph = new Paragraph();

paragraph.add(new Chunk("hello!"));

document.add(paragraph);

PdfPTable table = new PdfPTable(5);

table.setWidthPercentage(100f);

table.setWidths(new float[] {1.5f, 3.5f, 3.0f, 3.0f, 1.5f});

table.setSpacingBefore(10);

writeTableHeader(table);

writeTableData(table);

document.add(table);

document.close();

}

**CHAPTER#4 SPRING BOOT BATCH AND SWAGGER**

1. **Introduction:--**

**Batch:--** Multiple Operations are executed as one Task or one large/Big Task executed step by Step.

=>Every task is called as “**JOB”** and sub task is called as **“STEP”**.

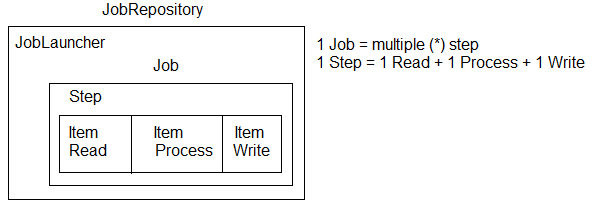
=>One JOB can contain one or more Steps (Step can also called as Sub Task).

=>One Step contains.

* 1. Item Reader (Read data from Source).
  2. Item Process (Do calculations and logics/operations etc…).
  3. Item writer (Provide output to next Step or final output).

=>JOBs are invoked by JobLauncher also known as JobStarter.

=>JOB, Steps and Lanuncher details must be stored in JobRepository (Config file).



# Step Implementation:--

=>In a Job (work) we can define one or multiple steps which are executed in order (step by step).

=>Job may contain 1 step, job may contain 2 step…… job may contains many Steps so, finally 1-job = \* (many) Step.

=>Every step having 3 execution stages

1. ItemReader<T>
2. ItemProcessor<I, O>
3. ItemWriter<T>.

**a>ItemReader<T> :--** It is used to read data **(Item)** from source **(Input location)** as input to current Step. A source can be File, DB, MQ, (Message Queues), Networks, XML, JSON… etc.

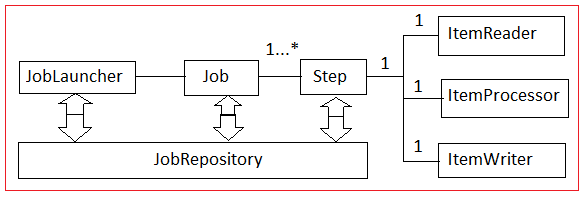
**b>ItemProcessor<I, O> :--** It is used to process input data given by Reader and returns in Modifier (or same) format of data.

**c>ItemWriter<T> :--** It will read bulk of Items from Processor at a time and writes to one destination. Even Destination can be a File (ex: DB, Text File, CSV, EXCEL, XML…etc).

=>To start Job, we need one **JobLauncher**… which works in general way or Scheduling based.

=>Here details like: Job, Steps, Launcher… are store in one memory which is called as

**JobRepository** [Ex: H2DB or MySQL, DB … any one DB].



# NOTE:--

1. An Item can be String, Array, Object of any class, Collection (List/Set….).
2. ItemReader will read one by one Item from Source. For example Source has 10 items then 10 times ItemReader will be executed.
3. ItemReader GenericType must match with ItemProcessor Input GenericType.
4. ItemProcess will read item by item from Reader and does some process (calculate, convert, check conditions, and convert to any class Object etc…).
5. ItemProcessor will provide output (may be same as Input type) which is called as Transformed Type.
6. ItemWriter will collect all output Items into one List type from Processor at a time (Only one time).
7. ItemWriter writes data to any destination.
8. Source/Destinatation can be Text, DB, Network, MessageQueue, Excel, CSV, JSON data, XML etc…

# Step Execution Flow :--

=>Step contains 3 interfaces (impls classes) given with generic types.

* 1. ItemReader<T> : (T= Input Data Type)
  2. ItemProcessor<I, O> : ( **I** must match with Reader **T** type and **O** must match with

writer **T** type).

* 1. ItemWriter<T> : (T = Type after processing)

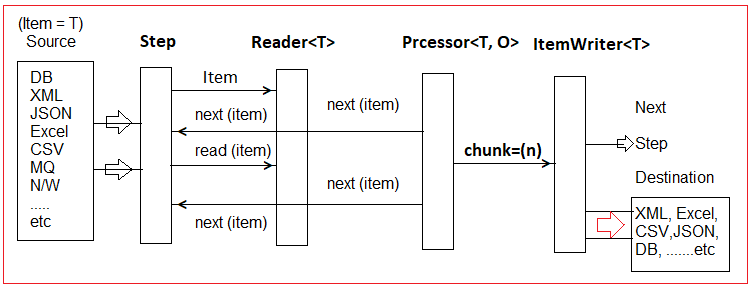
=>Here T/I/O can be String Object of any class or even collection (List, Set…).

=>Here ItemReader reads data from source with the helps of steps.

=>Reader and Processor are executed for every item, but writer is called based on chunk size.

Ex:-- No. of Items = 200, chunk=50 then Reader and Processor are executed 200 times but writer is called one time after 50 items are processed (Here 4 times executed).

=>ItemWriter writes data to destination with the help of step.



# Spring Boot Batch coding Files and Steps:-

Batch programming is implemented in 3 stages.

1. Step Creation
2. Job Creation
3. Job Executions
4. **Step Creation:-** This process is used to construct one (or more) Step(s). Ex:-- Step1, Stpe2, Step3..

=>here, Step is interface, It is constructed using **“StepBuilderFactory** (C)” with 3 inputs taken as (interfaces Impl class object) :

a>ItemReader<T> b>ItemProcessor<I, O> c>ItemWriter<T>

=>Programmer can define above interfaces Impl classes or can be also use exist (pre-defined) classes.

=>Reader, Writer, Processor is functional interfaces. We can define them using Lambda Expression and methods references even.

**2>Job Creation:--** One job is collection of Steps executed in order (one by one).

=>Even job may contain one Step also. Here, job is interface which is constructed using

**“JobBuilderFactory <C>”** and Step (I) instances.

=>To execute any logic before or after job, define Impl class for **“JobExecutionListener (I)”** having method

Like: beforeJob(…) : void and afterJob(…) : void

=>Here Listener is optional, It may be used to find current status of Batch (Ex: COMPLETED, STOPTES, FAILED…) start date and time, end date and time etc.

**3>Job Execution:--** Once Steps and job are configured, then we need to start them using **“JobLauncher (I)”** run(…) method.

=>This run(…) method takes 2 parameters

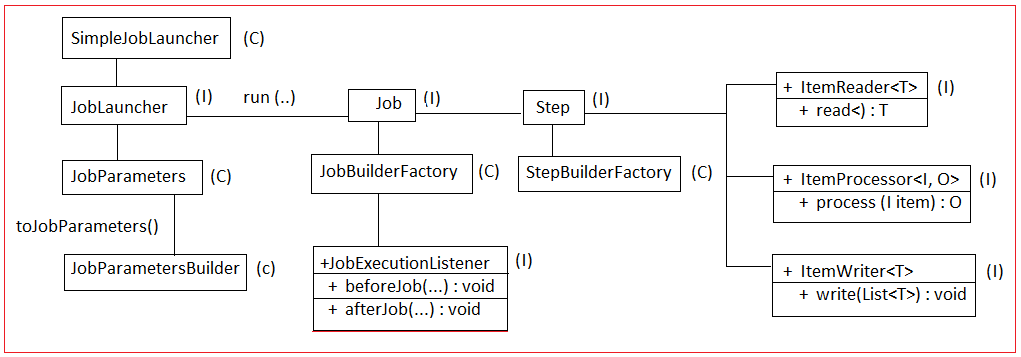
* 1. Job (I) and
  2. JobParameter (C)

=>Here, JobParameters (C) are inputs given to Job While starting this one.

Ex:- Parameters are : Server Data and Time, Customer Name flags (true/false), Task Name… etc.

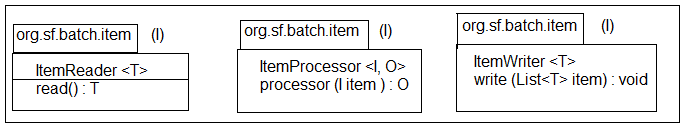
=>JobParameters (C) object is created using **“JobParametesBuilder”** and its method toJobParameters().

**Step :--** One **Step** can be constructed using StepBuilderFactory (sf) class by providing name, chunk size, reader, processor and writer.



# StepBuilderFactory (sf):--

|  |  |
| --- | --- |
| sf.get(“step1”) | =>Provide name of Step. |
| .<String, String> chunk(1) | => No. of Items to be read at a time. |
| .reader (readerObj) | =>Any Impl class of IteamReader<T> (I) |
| .processor(processorObj) | =>Any Impl class of itemProcessor <I, O> |
| .writer(writerObje | =>Any Impl class of ItemWriter <T> (I) |
| .build(); | =>Convert to step (impl class) Object. |

**UML Notation:--**

# JobBuilderFactory (C) :--

=>This class is used to create one or more Jobs using **Steps, listener, Incrementer**….

=>Job (I) Construction flow

# JobBuilderFactory jf :--

|  |  |
| --- | --- |
| Jf.get(“jobA”) | =>Job Name |
| .incremental(runIdIncrementar) | =>Incrementer |
| .listener (jobExListener) | =>Job Execution Listener |
| .start (stepA | =>First Step |

|  |  |
| --- | --- |
| .next (stepB)  .next (stepC)  .next (stepD) | =>Steps in Order |
| .build(); | =>Create job Type |

**JobExecutionListener (I):--**

=>This Interface is provided Spring Batch f/w, which gets called automatically for our Job.

=>For one job – one Listener can be configured.

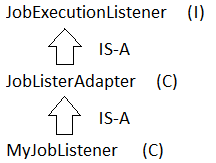
=>It is an Interface which has provided two abstract methods.

1. beforeJob (JobExecution) : void
2. afterJob (JobExecution) : void

=>If we write any impl class for above Listener (I) we need to implement two abstract method in our class.

=>Some times we need only one method in our class file afterJob() method is required. Then go for JobListenerAdapter(C) which has provided default impl logic for both **beforeJob; and afterJob();** methods.

=>JobExecution is a class which is used to find current job details like jobParameters, BatchStatus, stepExecutions etc…



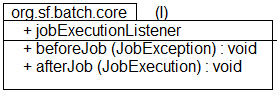
=>\*\*\*BatchStatus is an enum having possible values : COMPLETED, STARTING, STARTED, STOPPING, STOPPED, FAILED, ABANDONED, UNKNOWN.

# Q>What are possible Job Status/Batch Status in Batch Programming?

A> All possible status in Batch Execution are given as enum **“BatchStatus”**. Those are

|  |  |  |
| --- | --- | --- |
| 1 | COMPLETED | =>Successfully done. |
| 2 | STARTING | =>About to call run(…). |
| 3 | STARTED | =>Entered into run(…). |

|  |  |  |
| --- | --- | --- |
| 4 | STOPPED | =>Abort & come out of run(…). |
| 5 | STOPING | =>Run(…) method finished. |
| 6 | FAILED | =>Exception in run(…). |
| 7 | ABANDONED | =>Stopped by External service. |
| 8 | UNKNOWN | =>Unable to provide. |



# \*\*JobLauncher (I) :--

=>This interface is used to invoke our Job with Parameters (input) like creationTime, uniqueId (name), programmerData etc.

=>This Interface is having method run (Job, JobParameters).

=>This Interface object is created by JobParametersBuilder which holds data in Map

<key, Value> style.

# Final Spring Boot Batch Implementation Diagram:--

**Step#1:-** Define one Spring Starter Project and select **Spring Batch** or else add below dependencies.

<dependency>

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

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

</dependency>

**Step#2:-** Define Impl classes for IteamReader, ItemProcessor and IteamWriter Ex: Reader (C), Processor (C), Writer (C).

**Step#3:-** Define one class for Batch Configuration. a>Create Beans for Reader, Processor, Writer. b>Create Bean for Step using StepBuilderFactory

(BatchConfig <>StepBuilderFactory)

c>Create Bean for job using JobBuilderFactory (BatchConfig <>JobBuilderFactory)

d>Define JobExecutionListener (I) Impl class (It is optional)

**Step#4:-** Create ConsoldeRunner to make job Execution using JobLauncher [run(…) method].

ConsoleRunner <>JobLauncher

ConsoleRunner <>Job

=>Provider JobParameters using its builder obj.

**Step#5:-** At Starter class level add annotation: @EnableBatchProcessing

**Step#6:-** add below key in application.properties spring.batch.job.enabled=false

=>By default Starter class executes Job one time on startup application and even JobLauncher is executing another time. To avoid starter execution by starter class add above key as false value.

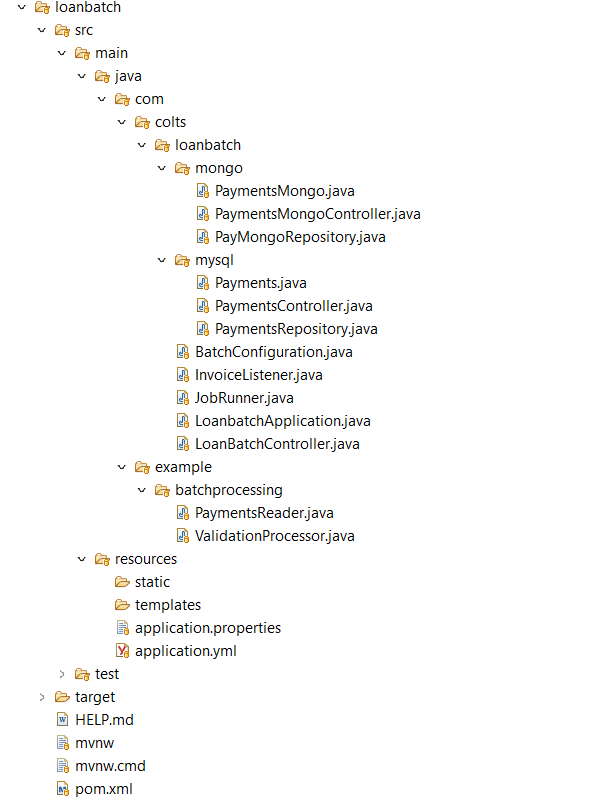
# Coding order:--

1. Reader
2. Processor
3. Writer
4. Step Configuration using StepBuilderFactory
5. JobExecutionListener
6. Job Config –job BuilderFactory
7. JobParameters using JobParametersBuilder
8. JobLauncher using CommandLineRunner
9. \*\* Add key in application.properties

Spring.batch.job.enabled=false

=>To avoid execution of job multiple times (by Starter class)

# #30. Folder Structure of SpringBoot batch Programing:--

****

**application.properties:--**

server.port=8084

# My SQL Database

spring.datasource.driver = com.mysql.jdbc.Driver

spring.datasource.url = jdbc:mysql://127.0.0.1:3306/db

spring.datasource.username = user

spring.datasource.password = password

#Mongo Db

spring.data.mongodb.host=127.0.0.1

spring.data.mongodb.port=27017

spring.data.mongodb.database=test\_db

spring.jpa.show-sql = true

spring.jpa.hibernate.ddl-auto = update

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

#Disabling Default while lable error

server.error.whitelabel.enabled=false

spring.batch.initialize-schema=ALWAYS

spring.batch.jdbc.initialize-schema=ALWAYS

# 1>DataReader:--

**public** **class** PaymentsReader **implements** ItemReader<Payments> {

@Autowired

PaymentsRepository paymentsRepository;

@Override

**public** Payments read() **throws** Exception, UnexpectedInputException, ParseException, NonTransientResourceException {

// **TODO** Auto-generated method stub

List<Payments> paymentList = **new** ArrayList<>();

paymentList= (List<Payments>) paymentsRepository.findAll();

Payments p = **new** Payments();

p=paymentList.stream().find().get();

paymentList.remove(paymentList.stream().findFirst().get());

**return** p;

}

}

# DataProcessor:--

package com.app.batch.processor;

public class ValidationProcessor implements ItemProcessor<Payments, PaymentsMongo> {

@Override

public PaymentsMongo process(Payments p) throws Exception {

PaymentsMongo pm = new PaymentsMongo(p.getPaymentId(), p.getLoanId(),LocalDate.now(), p.getUTR(), p.getAmount());

return pm;

}

}

# BatchConfig.java:--

@Configuration

@EnableBatchProcessing

public class BatchConfiguration {

@Autowired

PaymentsRepository paymentsRepository;

@Autowired

PayMongoRepository payMongoRepository;

@Bean

public ItemReader<Payments> reader() {

return new PaymentsReader();

}

@Bean

public ItemProcessor<Payments, PaymentsMongo> processor() {

return new ValidationProcessor();

}

@Bean

public ItemWriter<PaymentsMongo> writer(){

// return new InvoiceItemWriter(); // Using lambda expression code instead of a separate implementation

return paymentMongoList -> {

payMongoRepository.saveAll(paymentMongoList);

};

}

@Bean

public JobExecutionListener listener() {

return new InvoiceListener();

}

@Autowired

private StepBuilderFactory sbf;

//Step Object

@Bean

public Step stepA() {

return sbf.get("stepA")

.<Payments,PaymentsMongo>chunk(2)

.reader(reader())

.processor(processor())

.writer(writer())

.build()

;

}

//Autowire JobBuilderFactory

@Autowired

private JobBuilderFactory jbf;

//Job Object

@Bean

public Job jobA(){

return jbf.get("jobA")

.incrementer(new RunIdIncrementer()).flow(stepA()).end().build()

//.listener(listener())

//.start(stepA())

// .next(stepB())

// .next(stepC())

// .build()

;

}

}

# MyJobListener.java:--

public class InvoiceListener implements JobExecutionListener{

@Override

public void beforeJob(JobExecution jobExecution) {

System.out.println("Job started at: "+ jobExecution.getStartTime());

System.out.println("Status of the Job: "+jobExecution.getStatus());

}

@Override

public void afterJob(JobExecution jobExecution) {

System.out.println("Job Ended at: "+ jobExecution.getEndTime());

System.out.println("Status of the Job: "+jobExecution.getStatus());

}

}

# MyJobLauncher.java:--

@Component

public class JobRunner implements CommandLineRunner {

@Autowired

private JobLauncher jobLauncher;

@Autowired

private Job jobA;

@Override

public void run(String... args) throws Exception {

JobParameters jobParameters =

new JobParametersBuilder()

.addLong("time", System.currentTimeMillis())

.toJobParameters();

jobLauncher.run(jobA, jobParameters);

System.out.println("JOB Execution completed!");

}}

**2. Spring Boot Batch Processing : Moving data from MySQL to Mongo:--**

=>Consider input data in Payments table of Mysql of ng payment id, loanid, paymentdate, UTR by using one item reader read data to Payment class object.

=>Define one Processor class to calculate of payment.

=>Finally payment should be inserted to Mongo in Chunks.

Check Loaa batch in project for more details on this.

**4. Enable Swagger UI in Spring Boot ReST Application:--**

=>Compared to all other tools Swagger is a RichSet API provides dynamic UI based on code written for Rest Controller with common Paths.

**Step#1:-** Add below dependencies in pom.xml

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.7.0</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger-ui</artifactId>

<version>2.7.0</version>

</dependency>

|  |  |
| --- | --- |
| **Flow** | **Meaning** |
| ->Docket () | =>Create Docket |
| ->select () | =>Choose Rest classes |
| ->apis(basepackage()) | =>Classes are in package |
| ->paths (regex()) | =>Having common path |
| ->build() | =>Create final output |

Step#2:- Define Swagger Configuration class in Application (SwaggerConfig.java):-- @Configuration

@EnableSwagger2

**public** **class** SwaggerConfig {

**public** Docket api() {

**return** **new** Docket(DocumentationType.SWAGGER\_2).select()

.apis(RequestHandlerSelectors.basePackage("com.colt.lbconnect.controller"))

.paths(PathSelectors.any())

.build();

}

}

\*\* basePackage () is a static method defined in RequestHandlerSelectors (C) and in same way regex() is a static method defned in PathSelectors (C).

**Step#3:-** Run starter class and enter URL : http://localhost:2019/swagger-ui.html

http://localhost:2019/rest/employee/save

{ "empId" : 10, "empName" : "Uday Kumar", "empSal" : 45.87}

# Output:--

**-------------------------------------------------------------------------------------------------------------------**

Localhost:8080/swagger-ui.html to open swagger