Spring Batch

Process the bulk data without any interruption of end-user.

Use-cases:

1) generate the report.

2) send notification to end-user.

3) Populate the huge data from one environment to another environment.

4) Normally a typical batch process reads the large number of records from DB, file and queue and then write back in a modified form and that will be automated using the spring batch.

Example: In bank a customer having the dues amount on a particular date. An email alert will automatically triggered to that person. Then there can be a chance there might be other dues amounts as well for other customers. At same time mails will be triggered to the respective users.

In bank application everyday we generate the transaction report,

Let’s discuss few components used for batch process job.

There are three:

1) Item reader (FileReader, InputStream)

2) Processor (for validation, after reading update or delete from data and create data)

3) Item writer (OutputStream)

Parts of Batch process

1) Input source (File, Database, static data)

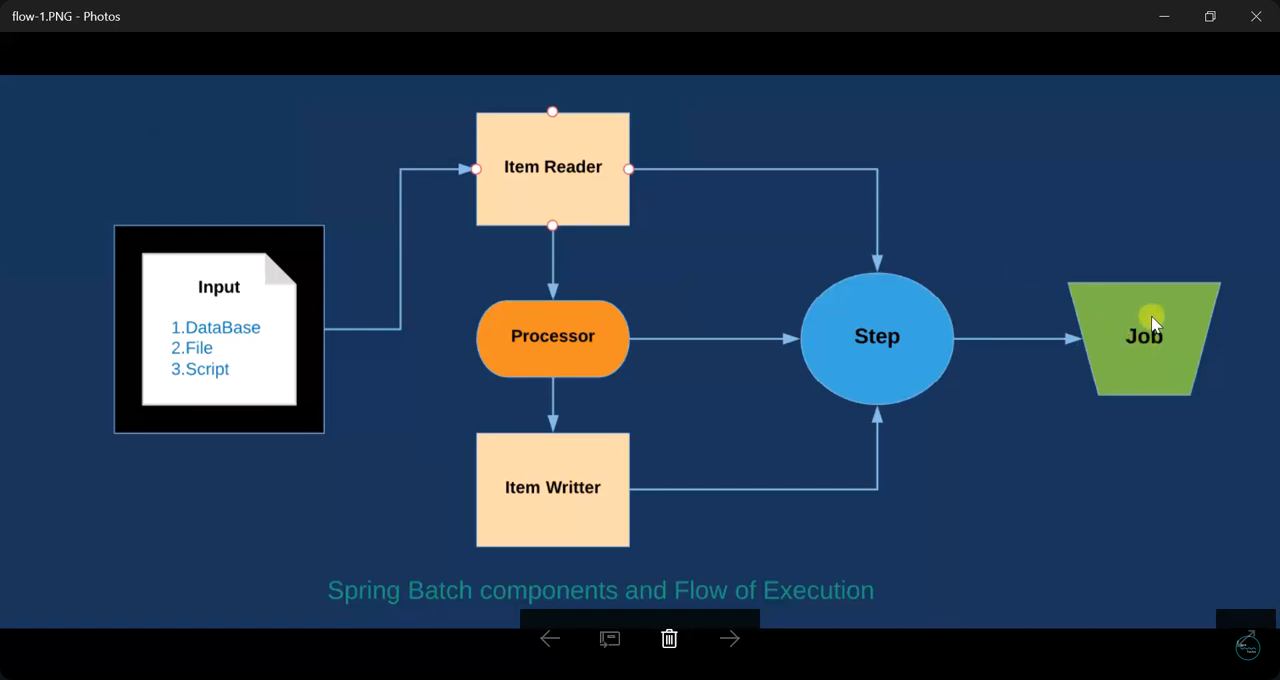
2) Item reader (FileReader, InputStream)

3) Processor (for validation, after reading update or delete from data and create data)

4) Item writer (OutputStream)

5) Step (used to process job in chunkf)

6) Job (that contains many jobs)



Create project with following dependencies:

1) Lombok

2) Mail

3) MongoDB

4) Batch

5) Web

Ad following two dependencies as well in your project:

To parse and to write xml:

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-oxm</artifactId>

</dependency>

<dependency>

<groupId>com.thoughtworks.xstream</groupId>

<artifactId>xstream</artifactId>

<version>1.4.9</version>

</dependency>

create packages:

>data:

@Getter

@Setter

@Document(collection= “BankUsers”)

Person.java

@Id

id

name

email

contactNo

dob

status

outstandingAmount

lastDuesDate

>application.properties

#DB

spring.data.mongodb.database=Person

spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

#Mail Properties

spring.mail.protocol=smtp

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=Enter ur email

spring.mail.password=Enter ur pwd

spring.mail.properties.mail.smtp.auth = true

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

>config

@Configuration

public class ApplicationConfig {

@Autowired

private JobBuilderFactory jobBuilderFactory;

@Autowired

private StepBuilderFactory stepBuilderFactory;

@Autowired

private MongoTemplate template;

@Autowired

private MailUtil util;

//The @SuppressWarnings("serial") annotation is used in Java to tell the compiler to suppress warnings related to a class that implements the Serializable interface but does not declare a serialVersionUID field.

@SuppressWarnings("serial") //pending to read

@Bean

public MongoItemReader<Person> reader() {

MongoItemReader<Person> reader = new MongoItemReader<Person>();

reader.setTemplate(template);

reader.setQuery("{}");

reader.setTargetType(Person.class);

reader.setSort(new HashMap<String, Sort.Direction>() {

{

put("\_custId", Direction.DESC);

}

});

return reader;

}

//We need to write output in xml format. So we need to convert object to xml. That is called marshaling. For it we use Jaxb api. But we will use Stax.

@Bean

public StaxEventItemWriter<Person> writter() {

StaxEventItemWriter<Person> writter = new StaxEventItemWriter<Person>();

writter.setRootTagName("Persons");

writter.setResource(new FileSystemResource("xml/bank.xml")); //it will create folder xml and place bank.xml in it.

writter.setMarshaller(marshaller());

return writter;

}

private XStreamMarshaller marshaller() {

XStreamMarshaller marshaller = new XStreamMarshaller();

Map<String, Class> map = new HashMap<>();

map.put("Person", Person.class);

marshaller.setAliases(map);

return marshaller;

}

@Bean

public Step step1() {

return stepBuilderFactory.get("step1").<Person, Person>chunk(10).reader(reader()).processor(process())

.writer(writter()).build();

//it can work without processor() as well if you don’t have any process()

}

@Bean

public Job runJob() {

return jobBuilderFactory.get("report generation").flow(step1()).end().build();

}

public ItemProcessor<Person, Person> process() {

ItemProcessor<Person, Person> process = new ItemProcessor<Person, Person>() {

@Override

public Person process(Person person) throws Exception {

if (person.getStatus().equalsIgnoreCase("Pending")) {

String msg = util.sendEmail(person.getEmail(), buildMessage(person));

System.out.println(msg);

return person;

}

return null;

}

private String buildMessage(Person person) {

String mailBody = "Dear " + person.getName() + "," + "\n" + "statement of your credit card ending with "

+ person.hashCode() + "\*\*" + " has been generated" + "\n" + "dues amount :"

+ person.getOutstandingAmount() + "\n" + "last payment date : "

+ new SimpleDateFormat("yyyy/MM/dd HH:mm:ss a").format(person.getLastDueDate()) + "\n" + "\n"

+ "If you already paid please ignore this email" + "\n" + "Thanks for using our credit card ";

return mailBody;

}

};

return process;

}

}

>main

SpringBatchMongoApplication

@SpringBootApplication**(exclude = DataSourceAutoConfiguration.class)** //when we provide db config in properties.

**@EnableBatchProcessing**

public class SpringBatchMongoApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBatchMongoApplication.class, args);

}

}

>util

MailUtil.java

@Component

public class MailUtil {

@Autowired

private JavaMailSender sender;

public String sendEmail(String to, String TextBody) {

String msg = "";

SimpleMailMessage message = new SimpleMailMessage();

try {

message.setTo(to);

message.setText(TextBody);

message.setSubject("Payment Dues Alert");

message.setFrom("javatechie4u@gmail.com");

sender.send(message);

msg = "mail triggered successfully to : " + to;

} catch (Exception e) {

msg = e.getMessage();

}

return msg;

}

}