Date : 18/02/2021
Spring Boot 6PM
Mr. RAGHU

Spring Boot : Batch Processing

*) Batch Processing : Working with large set of data.
(Data) (Data)
Source -> Read -> Process -> Write -> Destination

*) Source/Destination can be File System, Database, Streams(MQs)...etc
Few Examples:
a. CSV (Old Excel File) <---> SQL/NoSQL Database
b. XML <---> SQL/NoSQL Database
c. Database <---> Database (Backups).
[Production] [Maintains]

*) Source/Destination can have Data in any format : Text, JSON, XML, ...etc

==========

Job : Our Batch Processing /Big Task is indicates by Job.
Step : One Big Task is made into small tasks ie called as Steps.

=> One Job should have atleast one step and at most n steps.
=> One Step contains
One Reader/ItemReader [Read data from Source
One Processor/ItemProcessor [claculations/logics..etc
One Writer/ItemWriter [Give data to destination

JobLauncher : It is used to start/execute Job.
=> 1 time JobLauncher is called then 1 time Job is executed.
=> JobLauncher in realtime executed using schedulers.
For example after every 3 months take DB backup.

Job Repository : (it is memory/DB) It holds all executed Job inforamtion
used for next process(like backup revoke, re-execute...)
Details means : Job Execution Status, Execution Time (Start,End Times),
Steps in order...their details...

--FAQs-------

Q) Batch Processing?
bulk data/large data/more data.
source-->destination

Q) Data/Item exist in what format?
File(Text/XML/JSON..), Database, Streams(MQs)

Q) What are term in Batch processing?
A) Job, Step, Reader, Writer, Procesor,
JobLauncher (using Scheduler/Manual)

Q) What is Job Repository?
A) It is memory/db, executed job details
(time, status, order..etc)

---

*) Item means data that can be converted to one java object

ex: hello, welcome, to, nit, all

```
       employee table
    eid  ename   esal
    10    A       3.3
    11    B       3.3
    12    C       3.3
```

```
[
  { eid: 10, ename: 'A', esal:3.3 },
  { eid: 11, ename: 'B', esal:4.3 },
  { eid: 12, ename: 'C', esal:5.3 },
  { eid: 13, ename: 'D', esal:9.0 },
]
```

```
<employees>
  <emp>
      <eid>10</eid>
      <ename>A</ename>
      <esal>3.3</esal>
  </emp>
  <emp>
      <eid>11</eid>
      <ename>B</ename>
      <esal>3.3</esal>
  </emp>
  <emp>
      <eid>12</eid>
      <ename>C</ename>
      <esal>3.3</esal>
  </emp>
</employees>
```

-> Step executes Reader,Processor and Writer in order.
-> For every item from source 'Read and Processor' are executed one time.
   ie Source contains 40 items then 40 times reader and processor are executed.

-> After Processing, Writer reads all data/items into one List(size=chunk)
   which is limited to n number called as chunk.

-> Once Writer received items of size(chunk) then one time it will communicate to destination.
   ie Items =43, chunk =10, then 5 times writer communicates with Destination. (ie 5 network calls)

*) ItemReader<T>, ItemProcessor<I,O>, ItemWriter<T> are interfaces given by Spring Boot batch. (even impl class also given by BatchAPI).

-> Reader Generic Type(T) must match with Processor Input Type(I).
-> Processor Output Type (O) must match with Writer Generic Type(T).

```
------------------API Details-----------------------------------
Starter name : spring-boot-starter-batch

package name: org.springframework.batch.item

org.sf.batch.item
+ ItemReader<T>
+ read() :T

org.sf.batch.item
+ ItemProcessor<I, O>
+ process(I i):O

org.sf.batch.item
+ ItemWriter<T>
+ write(List<T> items):void

---------------------------------------------------------------
Q) What is the difference b/w below codes?
  List<T>
  List<? extends T>
  List<? super T>
 (with core java equal examples)
Ref Link:
https://docs.oracle.com/javase/tutorial/java/generics/bounded.html

A)


---------------------------------------------------------------
*)Note:
a. To create one step we need below details
   -> Step Name
   -> Chunk with <Input,Output>
   -> Reader Object
   -> Processor Object
   -> Writer Object

b. To create one Job we need below details
  -> Job Name
  -> Incrementor (call steps in given order)
  -> start(step)
  -> next(step)
  -> next(step)
      ....
  -> Job Execution Listener (optional)

*) JobExecutionListener(I) has 2 abstract methods
   beforeJob(...): void
   afterJob(...) : void

  -> for current batch details:
     To find time, batch status using logs, console..etc.

  -> It is optional.
-----------------------------------------------------------
```