

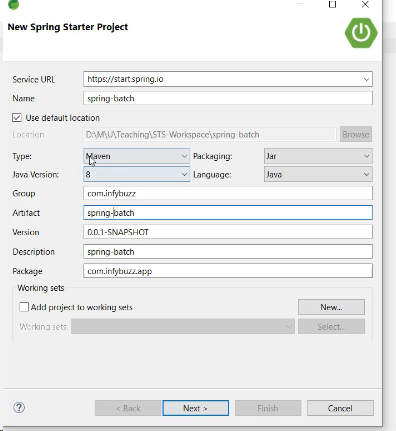
Concept about create springbatch project from STS:

Starting with springbatch in STS

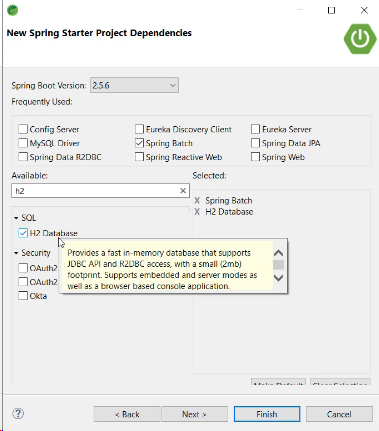
Add 2 dependencies 1. Springbatch 2.h2 db

To create springboot project in STS

1.file->new->choose spring starter project.

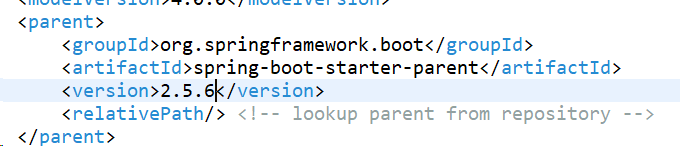


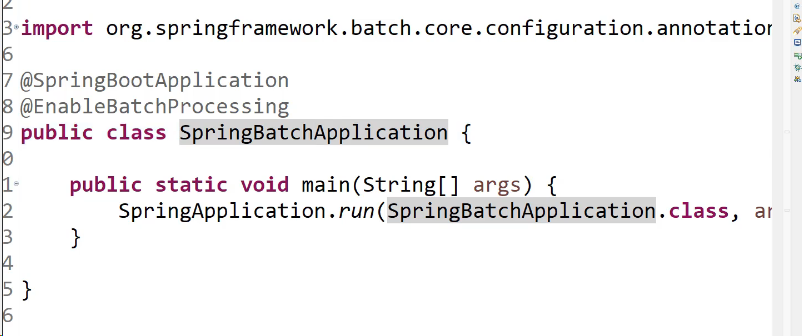
After click on next choose h2 and springbatch at available option shown in below



Click on finish. Add @enableBatchProcessing in main method.

Change parent version as 2.5.6 otherwise you get jobbuildfactory bean error.



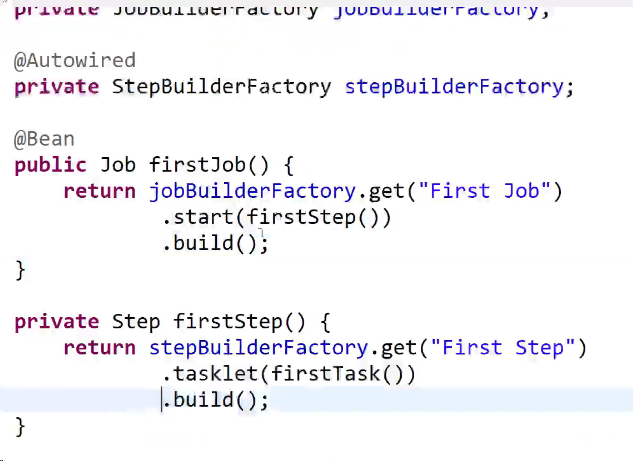


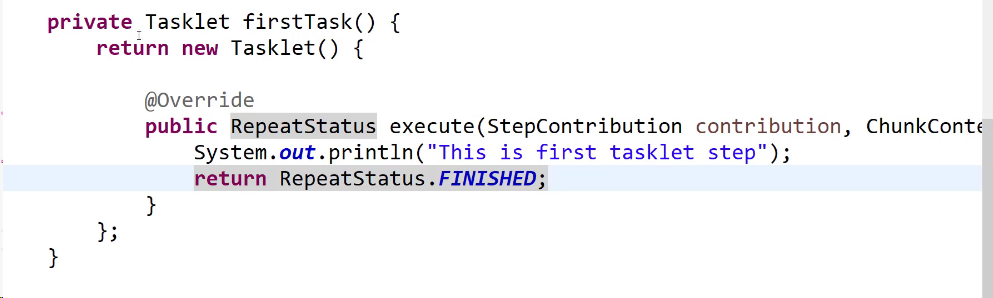
Concept about just start a sample job and test:

Create a simplejob class and write below code to start a job

First create a job bean and jobbean requires step so create step step requires tasklet.

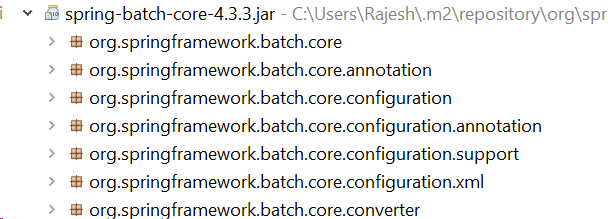
Tasklet means 🡪perform a task for requirement.





All classes of springbatch is existed on mavendependencies🡪spring-batch-core-4.3.3.jar

Go down to that last of this jar its having mysql related info also.



Concept about JobTables automatically created based on below configuration:

Job tables are not generated automatically getting mysql server version issue to avoid on this

Use mysql connector dependency::

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

Use below parent dependency ::

=========================================

<parent>

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

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

<version>2.5.6</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

================================

Use java version as 1.8::

<properties>

<java.version>1.8</java.version>

</properties>

=======================================================

Install mysql server 8.3 download and install.

https://dev.mysql.com/downloads/mysql/

Use application.properties :

spring.datasource.url=jdbc:mysql://localhost:3307/spring\_batch

spring.datasource.username=root

spring.datasource.password=root

spring.batch.jdbc.initialize-schema=always

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

====================================================

Concept about ItemRead,Process,Write :

ItemProcessor🡪 input is output from item reader and what we return from itemprocessor is the output for itemprocessor.

List<Integer>itemList=Arrays.asList(1,2,3,4,5,6,7,8,9,10);

Chunk 🡪 means chunk(3) it will read at a time 3 records from list.

**private** Step firstChunkStep() {

**return** stepBuilderFactory

.get("First Chunk Step")

.<Integer,Long>chunk(3).build();

}

.<Integer,Long>chunk(3).build()🡪means input from itemreader output is from itemwriter. Don’t focus on about itemProcessor on here.

ItemWriter🡪 target to write the data.

Spring.batch.job.enabled=false -🡪 it will stop all jobs in application.

Concept about async job:

@enableAsync add in mainmethod @async add in service layer method so that secondjob will not wait to complete the first job.

@Async

**public** **void** startJob(String jobName) {

JobExecution JobExecution = **null**;

HashMap<String, JobParameter> jobParams = **new** HashMap<>();

jobParams.put("currentTime", **new** JobParameter(System.*currentTimeMillis*()));

JobParameters jobParameters = **new** JobParameters(jobParams);

**try** {

**if** (jobName.equals("First Job")) {

JobExecution = JobLauncher.run(firstJob, jobParameters);

} **else** **if** (jobName.equals("second Job")) {

JobExecution = JobLauncher.run(secondJob, jobParameters);

}

System.***out***.println("job execution ID" + JobExecution.getId());

} **catch** (Exception e) {

System.***out***.println("exception while starting job" + e.getMessage());

}

}

Cronmaker.com 🡪 to get the cron expression. If we want to get 1 mint expression

0 0/1 \* 1/1 \* ? \* but we remove last \* provide up to ?

Concept about schedule job:

Add @enablescheduling in main method

//every 1 mint below method is called.

Add @scheduled(cron=”0 0/1 \* 1/1 \* ?”)

Public void startJob(){

}

Concept about stopJob:

If we want to stop job scheduling we get JobOperator object

**private** JobOperator jobOperator;

jobOperator.stop(executionId);

based on executionId we stop the job execution.

exectionId we get batch\_job\_execution table.

=======================================