Date : 16/04/2021
Spring Boot 9AM
Mr. RAGHU

Spring Boot : Technical View Of Batch API

ItemReader<T> (I): It will read data from source (Database,FileSystem)
For this Impl class is given by Spring batch API only.
we need not to write any new impl class.
Just configured existed class.

ItemProcessor<I,O> (I): For this we should define our own Impl class
with our logic.

ItemWriter<T> (I): It will write data to destination(DB,File)
For this Impl class is given by Spring batch API only.
we need not to write any new impl class.
Just configured existed class.

Step(I) : A Step a collection of
name + chunk + Reader Obj + Processor Object + writer object

StepBuilderFactory (C): Step is a Interface this is created using
StepBuilderFactory(C). This class object is auto-configured
by Batch API. So, directly use it(Autowired it).

Job : It is a collection of Steps
name + step caller(Incrementor) + listener object(optionla)
+ step objects in order

JobBuilderFactory(C) : This class is used to create Job object
This class object is auto-configured by Batch API.
So, directly use it(Autowired it).

JobExecutionListener (I):- This is optional. To execute any logic
(fine time, batch status..etc) before starting
and after finishing job processing.

*** We should define Impl class for this.

JobLauncher(I) and Impl class SimpleJobLauncher(C)
This is used to call/execute Job Object. Spring batch API
has provided Impl class and Auto-Configured, directly autowired
this.

JobParameters (C) created using JobParametersBuilder(C):-
To pass any input to Job (key=val, ex: clientId,AppVersion,
DB Location...etc) then we can use it. It can be empty also.

*) Generally in Spring f/w using properties file is not must.
But in Spring Boot almost all config inputs are given using
proeprties only. Batch API was designed using Spring. So,
JobParameters is added, now after Spring Boot, it is used
very less, directly pass using properties.

=======Batch API Coding Steps==========

1. Reader Class
2. Processor class
3. Writer class
4. Job Listener class
5. BatchConfig***
   a. Reader Object
   b. Processor Object
   c. Writer Object
   d. Listener Object
   e. Autowire StepBuilderFactory
   f. Step object
   g. Autowire JobBuilderFactory
   h. Job object
6. Job Runner

*) BatchStatus is a enum
  COMPLETED: Successfully done
  STARTING : About to Start Batch Processing
  STARTED : Job Runner is called
  STOPPING : About to finish last step
  STOPPED : Job Runner completed
  FAILED   : Exeception in Batch process
  ABANDONED: Job Execution stopped bocz of some problems
           in server/force stop of server/DB not responding.
  UNKNOWN   : Unable to find problem, check log files.

=======Example codes================================
*) Ignore Reader and Writer Impl class.

1. Reader
```
package in.nareshit.raghu.reader;

import org.springframework.batch.item.ItemReader;
import org.springframework.batch.item.NonTransientResourceException;
import org.springframework.batch.item.ParseException;
import org.springframework.batch.item.UnexpectedInputException;

public class MyReader implements ItemReader<String> {

        public String read() throws Exception,
UnexpectedInputException, ParseException,
NonTransientResourceException {
                System.out.println("FROM READER");
                return null;
        }
}
```
2. Processor
```
package in.nareshit.raghu.processor;

import org.springframework.batch.item.ItemProcessor;

public class MyProcessor implements ItemProcessor<String, String> {

        @Override
        public String process(String item) throws Exception {
                System.out.println("FROM PROCESSOR");
                return null;
```

```
	}
}
```

3. Writer

```
package in.nareshit.raghu.writer;

import java.util.List;

import org.springframework.batch.item.ItemWriter;

public class MyWriter implements ItemWriter<String> {

	public void write(List<? extends String> items)
			throws Exception {
		System.out.println("FROM WRITER");
	}
}
```

4. Listener

```
package in.nareshit.raghu.listener;

import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobExecutionListener;

public class MyJobListener implements JobExecutionListener {

	public void beforeJob(JobExecution je) {
		System.out.println("BEFORE JOB " + je.getStatus());
	}

	public void afterJob(JobExecution je) {
		System.out.println("AFTER JOB "  + je.getStatus());
	}

}
```

5. BatchConfig***

```
package in.nareshit.raghu.config;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.batch.core.Step;
import
org.springframework.batch.core.configuration.annotation.EnableBatchPro
cessing;
import
org.springframework.batch.core.configuration.annotation.JobBuilderFact
ory;
import
org.springframework.batch.core.configuration.annotation.StepBuilderFac
tory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.batch.item.ItemReader;
import org.springframework.batch.item.ItemWriter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;

import in.nareshit.raghu.listener.MyJobListener;
import in.nareshit.raghu.processor.MyProcessor;
import in.nareshit.raghu.reader.MyReader;
import in.nareshit.raghu.writer.MyWriter;

@EnableBatchProcessing
@Configuration
public class BatchConfig {

	//  a. Reader Object
	@Bean
	public ItemReader<String> reader() {
		return new MyReader();
	}
	//  b. Processor Object
	@Bean
	public ItemProcessor<String, String> processor() {
		return new MyProcessor();
	}
	//  c. Writer Object
	@Bean
	public ItemWriter<String> writer() {
		return new MyWriter();
	}
	//  d. Listener Object
	@Bean
	public JobExecutionListener listener() {
		return new MyJobListener();
	}
	//  e. Autowire StepBuilderFactory
	@Autowired
	private StepBuilderFactory sf;

	//  f. Step object
	@Bean
	public Step stepA() {
		return sf.get("stepA")//name
				.<String,String>chunk(10)
				.reader(reader()) //reader
				.processor(processor()) //processor
				.writer(writer()) //writer
				.build()
				;
	}
	//  g. Autowire JobBuilderFactory
	@Autowired
	private JobBuilderFactory jf;

	//  h. Job object
	@Bean
	public Job jobA() {
		return jf.get("jobA") //name
				.listener(listener()) //listener
				.incrementer(new RunIdIncrementer())
//call steps in order
```

```
                                .start(stepA()) //first step
                                //.next(stepB()) // next step
                                .build()
                                ;
        }
}
```

6. JobRunner

```
package in.nareshit.raghu.launcher;

import org.springframework.batch.core.Job;
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 MyJobLauncher implements CommandLineRunner {

        @Autowired
        private JobLauncher jobLauncher;
        @Autowired
        private Job jobA;

        public void run(String... args) throws Exception {
                jobLauncher.run(jobA,
                                new JobParametersBuilder()
                                .addLong("time",
System.currentTimeMillis())
                                .toJobParameters()
                                );
        }

}
```