Annotations

Friday, 2 December 2022 12:46 PM

Lombok's ***@Slf4j*** generates a logger instance with default name log using the SLF4J API.

```
@Configuration
@EnableBatchProcessing
public class BatchConfig {

   @Autowired
   private JobBuilderFactory jobBuilderFactory;

   @Autowired
   private StepBuilderFactory stepBuilderFactory;

   @Autowired
   private DataSource dataSource;

   @Bean
   public ItemReader<String> reader() {
      // Implementation of your ItemReader logic
      return null;
   }

   @Bean
   public ItemProcessor<String, String> processor() {
      // Implementation of your ItemProcessor logic
      return null;
   }

   @Bean
   public ItemWriter<String> writer() {
      // Implementation of your ItemWriter logic
      return null;
   }

   @Bean
   public Tasklet myTasklet() {
      return (contribution, chunkContext) -> {
         try {
            // Tasklet logic goes here

            return RepeatStatus.FINISHED;
         } catch (Exception e) {
            // Handle the exception as per your requirement
            throw new IllegalStateException("Failed to execute tasklet", e);
         }
      };
   }

   @Bean
   public Partitioner myPartitioner() {
      return gridSize -> {
         try {
            // Partitioner logic goes here

            return partitions;
```

```
            } catch (Exception e) {
                // Handle the exception as per your requirement
                throw new IllegalStateException("Failed to create partitions", e);
            }
        };
    }

    @Bean
    public Step myStep(ItemReader<String> reader, ItemProcessor<String, String> processor, ItemWriter<String>
writer) {
        return stepBuilderFactory.get("myStep")
                .<String, String>chunk(10)
                .reader(reader)
                .processor(processor)
                .writer(writer)
                .build();
    }

    @Bean
    public Job myJob(Step myStep, Partitioner myPartitioner) {
        return jobBuilderFactory.get("myJob")
                .start(myStep)
                .partitioner("myStep", myPartitioner)
                .build();
    }
}

@Component
public class MyItemReader implements ItemReader<String> {

    @Override
    public String read() throws Exception {
        // Implement your read logic here
        return null;
    }
}

@Component
public class MyItemProcessor implements ItemProcessor<String, String> {

    @Override
    public String process(String item) throws Exception {
        // Implement your processing logic here
        return null;
    }
}

@Component
public class MyItemWriter implements ItemWriter<String> {

    @Override
    public void write(List<? extends String> items) throws Exception {
        // Implement your write logic here
    }
}



import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;
```

```
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.partition.PartitionHandler;
import org.springframework.batch.core.partition.support.TaskExecutorPartitionHandler;
import org.springframework.batch.core.step.builder.StepBuilder;
import org.springframework.batch.core.step.builder.StepBuilderFactory;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.task.SimpleAsyncTaskExecutor;

@Configuration
@EnableBatchProcessing
public class BatchConfig {

   @Autowired
   private JobBuilderFactory jobBuilderFactory;

   @Autowired
   private StepBuilderFactory stepBuilderFactory;

   @Bean
   public Step step1() {
      return stepBuilderFactory.get("step1")
            .tasklet((contribution, chunkContext) -> {
               // Step logic goes here
               System.out.println("Step 1 executed");
               return RepeatStatus.FINISHED;
            })
            .build();
   }

   @Bean
   public Step step2() {
      return stepBuilderFactory.get("step2")
            .tasklet((contribution, chunkContext) -> {
               // Step logic goes here
               System.out.println("Step 2 executed");
               return RepeatStatus.FINISHED;
            })
            .build();
   }

   @Bean
   public PartitionHandler partitionHandler() {
      TaskExecutorPartitionHandler partitionHandler = new TaskExecutorPartitionHandler();
      partitionHandler.setStep(step1());
      partitionHandler.setGridSize(2);
      partitionHandler.setTaskExecutor(new SimpleAsyncTaskExecutor());
      return partitionHandler;
   }

   @Bean
   public Job myJob() {
      return jobBuilderFactory.get("myJob")
            .incrementer(new RunIdIncrementer())
            .start(step2())
            .next(partitionStep())
            .build();
   }
```

```
    @Bean
    public Step partitionStep() {
        StepBuilder stepBuilder = stepBuilderFactory.get("partitionStep");
        return stepBuilder.partitioner("step1", partitioner())
                .partitionHandler(partitionHandler())
                .build();
    }

    @Bean
    public MyPartitioner partitioner() {
        return new MyPartitioner();
    }

    // Other bean definitions and configuration

}
```

```
11
4
[5,9,6,10,-1,8,9,1,9,3,4]
[0,213,0,253,686,170,975,0,261,309,337]
```