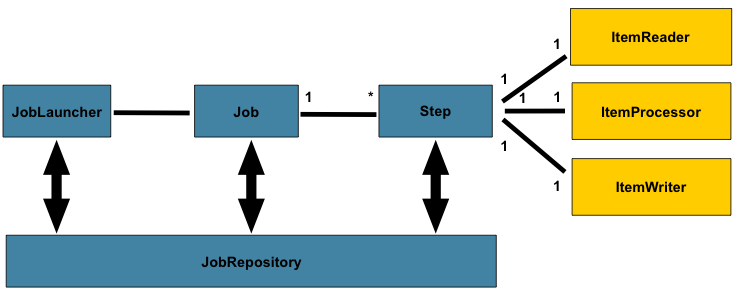
**Spring Batch**

1. **Tổng thể:**
2. *Định nghĩa:*

* Spring Batch là một phần của spring framework. Chúng giúp ta xử lý một chuỗi công việc trong một lúc, giảm overhead so với việc xử lý từng request một.
* Spring Batch cung cấp các chức năng có thể tái sử dụng, cần thiết cho quá trình xử lí lượng lớn dữ liệu.

1. *Cấu trúc:*



* **JobLaucher**: interface cho phép thực thi Job với tham số truyền vào.
* **Job:** nhận tham số và thực thi các Step được định nghĩa sẵn.
* **Step**: ta sẽ qui định các business ở đây. Mỗi Step sẽ gồm 3 bước: Item Reader, Item Processor, Item Writer. Hai bước Reader và Processor sẽ được xử lý đồng thời tuỳ vào lượng worker, riêng bước Writer sẽ xử lý theo chunk(ví dụ set chunk = 5 thì sau khi Reader và Processor xử lí đủ 5 record thì Writer mới bắt đầu chạy và xử lí).
* **JobRepository**: cung cấp CRUD cho JobLaucher, Job, Step.

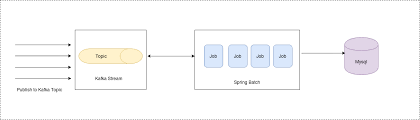
1. *Flow:*

* Khởi chạy JobLauncher với tham số truyền vào là một Job => Step bắt đầu khởi chạy phần đầu tiên là ItemReader để đọc và lấy dữ liệu(từ file csv, xml, kafka topic hoặc database…) => ItemProcessor lấy dữ liệu từ ItemReader và tiến hành xử lí dữ liệu nếu có =>

+ TH1: Nếu lượng dữ liệu đã được lấy chưa bằng với số chunk thì sẽ quay lại chạy ItemReader để đọc dữ liệu tiếp theo,

+ TH2: Nếu lượng dữ liệu đã lấy được bằng với số chunk thì tiến hành khởi chạy ItemWriter và lưu trữ dữ liệu xuống database => Sau đó lại quay lại chạy tiếp ItemReader.

1. **Demo:**
2. *Giới thiệu:*

* Phần tiếp theo, để dễ hình dung cách hoạt động, ý nghĩa của việc sử dụng spring batch cũng như cách code, chúng ta sẽ đi vào thực hiện một demo nhỏ về quản lí lượng order sản phẩm.
* Demo sử dụng Spring batch + kafka.
* Khách hàng sẽ gửi request order => Kafka sẽ tiếp nhận và lưu trữ vào kafka topic => Spring batch sẽ tiến hành đọc dữ liệu từ kafka topic => lưu trữ vào database. 

1. *Chuẩn bị:*

* IDE.
* Java 8 trở lên.
* MySQL server.

1. *Cấu trúc thư mục:*

Ảnh có chứa văn bản

Mô tả được tạo tự động

1. *Config maven:*

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
</dependency>  
<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <scope>runtime</scope>  
</dependency>  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-batch</artifactId>  
</dependency>  
<dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
</dependency>  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
</dependency>  
<!-- https://mvnrepository.com/artifact/org.springframework.kafka/spring-kafka -->  
<dependency>  
 <groupId>org.springframework.kafka</groupId>  
 <artifactId>spring-kafka</artifactId>  
 <version>2.9.1</version>  
</dependency>

1. *Config Kafka:*
   1. *Kafka constants:*

* Tạo class KafkaConstants nhằm chứa các hằng số chung liên quan đến kafka như topic, group id, kafka broker.

public class KafkaConstants {  
 public static final String *KAFKA\_TOPIC* = "kafka-product-mangement-3";  
 public static final String *GROUP\_ID* = "product-management-sandbox";  
 public static final String *KAFKA\_BROKER* = "localhost:9092";  
}

* 1. *Config Producer:*
* Đầu tiên, cần cấu hình ProducerFactory nhằm tạo Producer instance.
* Tiếp đó, chúng ta cũng cần một KafkaTemplate nhằm cung cấp các phương thức cho việc gửi message đến Kafka topic.

@EnableKafka  
@Configuration  
public class KafkaProducerConfiguration {  
 @Bean  
 public ProducerFactory<String, OrderEntity> producerFactory() {  
 return new DefaultKafkaProducerFactory<>(producerConfigurations());  
 }  
  
 @Bean  
 public Map<String, Object> producerConfigurations() {  
 Map<String, Object> configurations = new HashMap<>();  
 configurations.put(ProducerConfig.*BOOTSTRAP\_SERVERS\_CONFIG*, KafkaConstants.*KAFKA\_BROKER*);  
 configurations.put(ProducerConfig.*KEY\_SERIALIZER\_CLASS\_CONFIG*, StringSerializer.class);  
 configurations.put(ProducerConfig.*VALUE\_SERIALIZER\_CLASS\_CONFIG*, JsonSerializer.class);  
 return configurations;  
 }  
  
 @Bean  
 public KafkaTemplate<String, OrderEntity> kafkaTemplate() {  
 return new KafkaTemplate<>(producerFactory());  
 }  
}

* 1. *Config Consumer:*
* Cũng như Producer, chúng ta cũng cần phải config một ConsumerFactory và một KafkaListenerContainerFactory để lắng nghe message từ Producer.

@EnableKafka  
@Configuration  
public class KafkaConsumerConfig {  
 @Bean  
 ConcurrentKafkaListenerContainerFactory<String, OrderEntity> kafkaListenerContainerFactory() {  
 ConcurrentKafkaListenerContainerFactory<String, OrderEntity> factory = new ConcurrentKafkaListenerContainerFactory<>();  
 factory.setConsumerFactory(consumerFactory());  
 return factory;  
 }  
  
 @Bean  
 public ConsumerFactory<String, OrderEntity> consumerFactory() {  
 return new DefaultKafkaConsumerFactory<>(consumerConfigurations(), new StringDeserializer(), new JsonDeserializer<>(OrderEntity.class));  
 }  
  
 @Bean  
 public Map<String, Object> consumerConfigurations() {  
 Map<String, Object> configurations = new HashMap<>();  
 configurations.put(ConsumerConfig.*BOOTSTRAP\_SERVERS\_CONFIG*, KafkaConstants.*KAFKA\_BROKER*);  
 configurations.put(ConsumerConfig.*GROUP\_ID\_CONFIG*, KafkaConstants.*GROUP\_ID*);  
 configurations.put(ConsumerConfig.*KEY\_DESERIALIZER\_CLASS\_CONFIG*, StringDeserializer.class);  
 configurations.put(ConsumerConfig.*VALUE\_DESERIALIZER\_CLASS\_CONFIG*, JsonDeserializer.class);  
 configurations.put(ConsumerConfig.*AUTO\_OFFSET\_RESET\_CONFIG*, "latest");  
 return configurations;  
 }  
}

* 1. *Tạo Listener:*
* Tạo Class OrderListener nhằm tiếp nhận dữ liệu và gọi JobLauncher nhằm khởi động Spring Batch.
* Autowired JobLauncer và Job nhằm khởi chạy Spring Batch sau khi lắng nghe được message từ Producer.

@Component  
public class OrderListener {  
 @Autowired  
 private JobLauncher jobLauncher;  
 @Autowired  
 private Job job;  
  
 @KafkaListener(  
 topics = KafkaConstants.*KAFKA\_TOPIC*,  
 groupId = KafkaConstants.*GROUP\_ID* )  
 public void listen(OrderEntity order) {  
 System.*out*.println("sending via kafka listener..");  
 JobParameters jobParameters = new JobParametersBuilder()  
 .addLong("startAt", System.*currentTimeMillis*()).toJobParameters();  
 try {  
 jobLauncher.run(job, jobParameters);  
 } catch (JobExecutionAlreadyRunningException | JobRestartException  
 | JobInstanceAlreadyCompleteException | JobParametersInvalidException e) {  
  
 e.printStackTrace();  
 }  
 }  
}

1. *Config Spring Batch:*

* Tạo class BatchConfig chứa các config cho Spring Batch.
* Cụ thể: tiến hành config Joblauncher, Job, Step.
* Đối với Step: config KafkaItemReader nhằm đọc dữ liệu từ kafka topic, tiếp đến cấu hình Processor để tiếp nhận và xử lí dữ liệu nếu có. Sau cùng, config writer nhằm ghi dữ liệu vào database.

@Configuration  
@EnableBatchProcessing  
@RequiredArgsConstructor  
public class BatchConfig extends DefaultBatchConfigurer {  
 private static final Logger *logger* = LoggerFactory.*getLogger*(BatchConfig.class);  
 private final KafkaProperties properties;  
 private final StepBuilderFactory stepBuilderFactory;  
 private final JobBuilderFactory jobBuilderFactory;  
 private final OrderItemProcessor orderItemProcessor;  
 @Autowired  
 private JobRepository jobRepository;  
  
 @Override  
 public void setDataSource(DataSource dataSource) {  
 super.setDataSource(null);  
 }  
  
 @Bean(name = "myJobLauncher")  
 public JobLauncher simpleJobLauncher() throws Exception {  
 SimpleJobLauncher jobLauncher = new SimpleJobLauncher();  
 jobLauncher.setJobRepository(jobRepository);  
 jobLauncher.setTaskExecutor(new SimpleAsyncTaskExecutor());  
 jobLauncher.afterPropertiesSet();  
 return jobLauncher;  
 }  
  
 @Bean  
 public Job importUserJob(Step step1) {  
 return jobBuilderFactory.get("importUserJob")  
 .incrementer(new RunIdIncrementer())  
 .flow(step1)  
 .end()  
 .build();  
 }  
  
 @Bean(name = "reader")  
 KafkaItemReader<OrderEntity, OrderEntity> kafkaItemReader() {  
 Properties props = new Properties();  
 props.putAll(this.properties.buildConsumerProperties());  
 return new KafkaItemReaderBuilder<OrderEntity, OrderEntity>()  
 .partitions(0)  
 .consumerProperties(props)  
 .name("customer-reader")  
 .saveState(true)  
 .topic(KafkaConstants.*KAFKA\_TOPIC*)  
 .build();  
 }  
  
 @Bean  
 public JdbcBatchItemWriter<OrderEntity> writer(DataSource dataSource) {  
 return new JdbcBatchItemWriterBuilder<OrderEntity>().itemSqlParameterSourceProvider(new BeanPropertyItemSqlParameterSourceProvider<>())  
 .sql("INSERT INTO order\_entity (product\_name, numbers\_of\_product) VALUES (:productName, :numbersOfProduct)")  
 .dataSource(dataSource)  
 .build();  
 }  
 @Bean(name = "start")  
 Step start(JdbcBatchItemWriter<OrderEntity> writer) {  
 return stepBuilderFactory  
 .get("step1")  
 .<OrderEntity, OrderEntity>chunk(2)  
 .reader(kafkaItemReader())  
 .processor(orderItemProcessor)  
 .writer(writer)  
 .build();  
 }  
  
}