Spring Batch Integration with Detailed Code and Comments

Overview

This document provides a step-by-step guide to implementing a Spring Batch Job that waits for the application to start, runs a batch job using the latest versions, and integrates Spring Integration for message flow.

Prerequisites

- JDK 17 or higher
- Spring Boot 3.x
- Dependencies: spring-boot-starter-batch, spring-integration
- A properly configured database (e.g., H2, MySQL)

Step 1: Batch Configuration

```
@Configuration
public class BatchConfig {
    @Bean
    public Job job(JobBuilderFactory jobBuilderFactory, Step step) {
        return jobBuilderFactory.get("myJob")
                .start(step)
                .build();
    }

    @Bean
    public Step step(StepBuilderFactory stepBuilderFactory) {
        return stepBuilderFactory.get("myStep")
                .tasklet((contribution, chunkContext) -> {
```

```
                System.out.println("Executing step...");
                return RepeatStatus.FINISHED;
            }).build();
    }
}
```

Step 2: Integration Flow Configuration

```
@Configuration
public class IntegrationFlowConfig {

    @Bean
    public IntegrationFlow integrationFlow(JobLaunchingGateway jobLaunchingGateway) {
        return IntegrationFlows.from("inputChannel")
                .transform(new JobLaunchRequestTransformer())
                .handle(jobLaunchingGateway)
                .channel("outputChannel")
                .get();
    }

    @Bean
    public JobLaunchingGateway jobLaunchingGateway(JobLauncher jobLauncher) {
        return new JobLaunchingGateway(jobLauncher);
    }

    @Bean
    public MessageChannel inputChannel() {
```

```
        return new DirectChannel();
    }

    @Bean
    public MessageChannel outputChannel() {
        return new DirectChannel();
    }
}
```

Step 3: Application Startup Listener

```
@SpringBootApplication
public class BatchApplication implements ApplicationListener<ApplicationReadyEvent> {

    private final MessageChannel inputChannel;

    public BatchApplication(MessageChannel inputChannel) {
        this.inputChannel = inputChannel;
    }

    public static void main(String[] args) {
        SpringApplication.run(BatchApplication.class, args);
    }

    @Override
    public void onApplicationEvent(ApplicationReadyEvent event) {
        System.out.println("Application started. Sending job launch request...");
```

```
        JobLaunchRequest request = new JobLaunchRequest(new Job(), new JobParameters());
        inputChannel.send(MessageBuilder.withPayload(request).build());
    }
}
```

Step 4: JobLaunchRequest Transformer

```
public class JobLaunchRequestTransformer implements GenericTransformer<String, JobLaunchRequest> {

    @Override
    public JobLaunchRequest transform(String filePath) {
        JobParameters params = new JobParametersBuilder()
                .addString("filePath", filePath)
                .toJobParameters();
        return new JobLaunchRequest(new Job(), params);
    }
}
```

Explanation

- The `BatchConfig` class defines the batch job and its steps.
- The `IntegrationFlowConfig` class creates the input and output channels, and configures a flow that transforms a
  file path into a `JobLaunchRequest`, which is handled by the `JobLaunchingGateway`.
- The `BatchApplication` class sends a message to the `inputChannel` after the application has started.
- The `JobLaunchRequestTransformer` transforms an input message (e.g., file path) into a

`JobLaunchRequest`

with necessary parameters.