Spring Batch

Introducción

Spring Batch es un framework ligero y comprensivo diseñado para facilitar el desarrollo de aplicaciones de procesamiento por lotes. Este tipo de aplicaciones ejecuta tareas repetitivas en un gran volumen de datos de manera eficiente y confiable. Spring Batch proporciona características esenciales como procesamiento de transacciones, logging/tracing, administración de trabajos, estadísticas de trabajos, manejo de excepciones y reinicio.

Las aplicaciones de procesamiento por lotes son comunes en diversas industrias y se utilizan para tareas como la migración de datos, la generación de informes, el procesamiento de archivos, y la integración de sistemas. Spring Batch ofrece una estructura bien definida para desarrollar, probar, y mantener estas aplicaciones, garantizando que sean robustas y escalables.

Elementos de Spring Batch

1. Job

Un Job en Spring Batch representa el proceso por lotes completo. Está compuesto por uno o más Steps y define la secuencia en la que estos pasos se ejecutan. Un Job puede ser configurado para manejar la lógica de control de flujo, como la ejecución condicional y la repetición de pasos fallidos.

```
@Bean

public Job exampleJob(JobBuilderFactory jobBuilderFactory, Step step1) {

   return jobBuilderFactory.get("exampleJob")

      .start(step1)

      .build();

}
```

Este bloque de código define un Job llamado exampleJob. Un Job en Spring Batch es una tarea completa que puede incluir múltiples Step. Aquí, se utiliza el JobBuilderFactory para construir el Job, y se configura para que comience con el step1.

2. Step

Un Step es una etapa independiente de un Job que encapsula una etapa del procesamiento por lotes. Un Step puede incluir tareas como leer datos de una fuente, procesar estos datos y escribir el resultado en un destino. Cada Step puede configurarse con transacciones y manejo de errores.

```
@Bean
public Step step1(StepBuilderFactory stepBuilderFactory, ItemReader<String> reader,
            ItemProcessor<String, String> processor, ItemWriter<String> writer) {
   return stepBuilderFactory.get("step1")
      .<String, String>chunk(10)
      .reader(reader)
      .processor(processor)
      .writer(writer)
      .build();
}
```

Este bloque de código define un Step llamado step1. Un Step es una unidad de trabajo dentro de un Job. Aquí, se utiliza el StepBuilderFactory para construir el Step, y se configura para procesar datos en fragmentos (chunk) de 10 elementos. Se especifican un ItemReader, un ItemProcessor y un ItemWriter para leer, procesar y escribir los datos, respectivamente.

3. ItemReader

El ItemReader es responsable de leer los datos de la fuente. Puede ser un archivo, una base de datos, una cola de mensajes, etc. El ItemReader lee los datos uno por uno y los pasa al ItemProcessor.

```
@Bean
public FlatFileItemReader<Person> reader() {
   return new FlatFileItemReaderBuilder<Person>()
      .name("personItemReader")
      .resource(new ClassPathResource("sample-data.csv"))
      .delimited()
      .names(new String[]{"firstName", "lastName"})
      .fieldSetMapper(new BeanWrapperFieldSetMapper<Person>() {{
         setTargetType(Person.class);
      }})
      .build();
```

```
}
```

Este bloque de código define un ItemReader que lee datos de un archivo CSV. Utiliza FlatFileItemReaderBuilder para construir el lector, especificando el nombre del lector, el recurso (archivo CSV), el delimitador, los nombres de los campos y un FieldSetMapper para mapear los datos leídos a una clase Person.

4. ItemProcessor

El ItemProcessor se encarga de procesar cada ítem leído por el ItemReader. Aquí se puede implementar la lógica de negocio necesaria para transformar los datos antes de escribirlos.

```
public class PersonItemProcessor implements ItemProcessor<Person, ProcessedPerson> {

   @Override

   public ProcessedPerson process(Person person) throws Exception {

      // Processing logic

      return new ProcessedPerson(person.getFirstName().toUpperCase(),
person.getLastName().toUpperCase());

   }

}
```

Este bloque de código define un ItemProcessor que transforma los datos leídos. Implementa la interfaz ItemProcessor y sobrescribe el método process para convertir los nombres a mayúsculas y crear una instancia de ProcessedPerson.

5. ItemWriter

El ItemWriter es responsable de escribir los datos procesados en un destino. Puede ser un archivo, una base de datos, etc.

```
@Bean

public JdbcBatchItemWriter<ProcessedPerson> writer(DataSource dataSource) {

   return new JdbcBatchItemWriterBuilder<ProcessedPerson>()

      .itemSqlParameterSourceProvider(new BeanPropertyItemSqlParameterSourceProvider<>())

      .sql("INSERT INTO processed_people (first_name, last_name) VALUES (:firstName,
:lastName)")

      .dataSource(dataSource)

      .build();
```

}

Este bloque de código define un ItemWriter que escribe los datos procesados en una base de datos. Utiliza JdbcBatchItemWriterBuilder para construir el escritor, especificando un itemSqlParameterSourceProvider para mapear los parámetros de los objetos a SQL, la consulta SQL y la fuente de datos (dataSource).

6. JobRepository

El JobRepository es el componente que almacena el estado de los Jobs y Steps. Es crucial para el manejo de transacciones y reinicio de trabajos fallidos.

```
@Bean
public JobRepository jobRepository(DataSource dataSource, PlatformTransactionManager transactionManager) throws Exception {
    JobRepositoryFactoryBean factory = new JobRepositoryFactoryBean();
    factory.setDataSource(dataSource);
    factory.setTransactionManager(transactionManager);
    factory.setIsolationLevelForCreate("ISOLATION_SERIALIZABLE");
    factory.setTablePrefix("BATCH_");
    factory.afterPropertiesSet();
    return factory.getObject();
}
```

Este bloque de código configura un JobRepository, que maneja la persistencia del estado del trabajo. Utiliza JobRepositoryFactoryBean para crear el repositorio, especificando la fuente de datos (dataSource), el gestor de transacciones (transactionManager), el nivel de aislamiento (ISOLATION_SERIALIZABLE) y el prefijo de las tablas (BATCH_).

7. JobLauncher

El JobLauncher es el componente que se utiliza para iniciar un Job.

```
@Bean
public JobLauncher jobLauncher(JobRepository jobRepository) {
    SimpleJobLauncher jobLauncher = new SimpleJobLauncher();
    jobLauncher.setJobRepository(jobRepository);
```

```
    return jobLauncher;
}
```

Este bloque de código define un JobLauncher, que se encarga de iniciar la ejecución de trabajos. Utiliza SimpleJobLauncher y establece el JobRepository creado anteriormente.

8. JobParameters

Los JobParameters son parámetros que se pasan al Job en el momento de su ejecución. Estos parámetros pueden ser utilizados para controlar el comportamiento del Job.

```
JobParameters jobParameters = new JobParametersBuilder()
        .addString("input.file.name", "input.csv")
        .toJobParameters();
jobLauncher.run(job, jobParameters);
```

Conclusión

Spring Batch es una herramienta poderosa para desarrollar aplicaciones de procesamiento por lotes robustas y escalables. Su estructura modular y configurable permite manejar de manera eficiente grandes volúmenes de datos, y su integración con otros componentes de Spring facilita su adopción y uso en aplicaciones empresariales.