


[Inicio](#) » [Java](#) » [Enterprise Java](#) » [Spring Batch: Escritor de salida de formato múltiple](#)

ACERCA DE JONNY HACKETT



Jonny es un ingeniero de software senior y mentor con más de 15 años de experiencia en TI. Como desarrollador Java, ávido fanático de SportingKC y fotógrafo.

Spring Batch: Escritor de salida de formato múltiple

Publicado por: Jonny Hackett en Enterprise Java 17 de agosto de 2016 0 153 Vistas



Al ser un firme defensor de Spring Batch, siempre he hablado sobre la noción de que Spring Batch proporciona a los desarrolladores un marco que les permite enfocarse en resolver las necesidades de negocios. Al hacerlo, permite a los desarrolladores no dedicar una cantidad excesiva de tiempo a resolver todos los aspectos técnicos para respaldar la solución.

Para ilustrar lo que quiero decir con esto, vamos a tomar uno de los ejemplos anteriores de Spring Batch que he escrito y lo mejoraré un poco para un requisito comercial adicional que se necesitaba.

El nuevo problema

En la Parte Tres de mi serie Spring Batch, presentamos un tutorial para manejar la

salida de archivos de Excel grandes.

Más tarde se determinó que una unidad de negocios adicional necesitaba los mismos datos, sin embargo, necesitaban la salida de datos en el formato de un archivo de texto delimitado por tuberías con solo tres de los campos.

Hay un par de formas diferentes de hacer esto, pero para este ejemplo, le mostraré cómo implementar rápidamente el suyo propio

```
ItemStreamReader
```

que delega la escritura a sus escritores individuales.

Lo primero que debemos hacer es crear el shell de nuestro

```
ItemStreamReader
```

. Lo estoy llamando el

```
MultiFormatItemWriter
```

. Aquí es cómo se ve la concha:

```
01 package com.keyhole.example;
02
03 import java.io.IOException;
04 import java.util.List;
05
06 import org.springframework.batch.core.StepExecution;
07 import org.springframework.batch.core.annotation.AfterStep;
08 import org.springframework.batch.core.annotation.BeforeStep;
09 import org.springframework.batch.item.ExecutionContext;
10 import org.springframework.batch.item.ItemStreamException;
11 import org.springframework.batch.item.ItemStreamWriter;
12 import org.springframework.batch.item.file.FlatFileItemWriter;
13 import org.springframework.beans.factory.annotation.Autowired;
14 import org.springframework.beans.factory.annotation.Qualifier;
15 import org.springframework.context.annotation.Scope;
16 import org.springframework.stereotype.Component;
17
```

HOJA INFORMATIVA

¡Los iniciados ya están disfrutando actualizaciones semanales y libros de cortésia! Unase a ellos ahora para obtener a las últimas noticias en el mundo como también información sobre Groovy y otras tecnologías relacionadas.

Enter your e-mail...

☐ Estoy de acuerdo con los Términos y Condiciones y la Política de Privacidad

[Regístrate](#)

ÚNETE A NOSOTROS



Con 1,500+ miembros únicos en 500+ ubicaciones, somos el sitio de Java. Códigos de búsqueda de animación de nosotros.

un blog con contenido único e interesante para consultar nuestro programa de escritura. También puedes ser un escritor.

```

25     @Override
26     public void write(List<? extends StockData> items) throws Exception {
27     }
28
29     @BeforeStep
30     public void beforeStep(StepExecution stepExecution) {
31     }
32
33     @AfterStep
34     public void afterStep(StepExecution stepExecution) throws IOException {
35     }
36
37     @Override
38     public void open(ExecutionContext executionContext) throws ItemStreamException {
39     }
40
41     @Override
42     public void update(ExecutionContext executionContext) throws ItemStreamException {
43     }
44
45     @Override
46     public void close() throws ItemStreamException {
47     }
48 }

```

A continuación, tendremos que hacer algunos ajustes a nuestro existente

```
StockDataExcelWriter
```

del ejemplo anterior para que funcione como un delegado en nuestro nuevo

```
MultiFormatItemWriter
```

. También encontré que hubo algunos problemas con el ejemplo anterior relacionado con el flujo de datos proveniente de Nasdaq. El formato de uno de los campos había cambiado y el ejemplo ya no funcionaba, por lo que había que corregirlo antes de poder continuar.

- Corrección de errores: se cambió el tipo de campo de marketCap en StockData de BigDecimal a String. Los valores ahora aparecían en la fuente de datos como "\$ 14.5M" y similares.
- Corrección de errores: dado que el formato de los datos había cambiado y estos artículos de blog usaban en su mayoría ejemplos estáticos, he creado un archivo de entrada con los datos de stock mencionados

```
companylist.csv
```

en el

```
data.stock
```

paquete en src / test / resources.

- Corrección de errores: modifiqué el lector de datos de archivo para usar este archivo de datos en lugar del feed Nasdaq en vivo.
- Se eliminó la

```
@Scope
```

anotación ("paso") de

```
StockDataExcelWriter
```

. Esto es necesario ya que el

```
MultiFormatItemWriter
```

alcance será en el nivel de paso.

- Se eliminaron las anotaciones

```
@BeforeStep
```

y las

```
@AfterStep
```

anotaciones

```
StockDataExcelWriter
```

ya que estos métodos se llamarán directamente desde MultiFormatItemWriter.

- Comentó el bucle for dentro del método de escritura que estaba escribiendo cada registro 300 veces en el archivo excel. Esto se usó para la demostración de archivos de Excel grandes, por lo que tendría que revertirse para que el ejemplo vuelva a funcionar.

Ahora que hemos abordado el problema

```
StockDataExcelWriter
```

, debemos abordar la salida de formato adicional que necesita la empresa. La segunda salida debe estar en un archivo de texto delimitado por tuberías y solo debe contener el símbolo, el nombre y los campos de la última venta.

Para este escritor delegado, vamos a utilizar el

```
FlatFileItemWriter
```

```

03     <property name="lineAggregator">
04         <bean class="org.springframework.batch.item.file.transform.DelimitedLineAggregator">
05             <property name="delimiter" value="|" />
06             <property name="fieldExtractor">
07                 <bean class="org.springframework.batch.item.file.transform.BeanWrapperFieldExtractor">
08                     <property name="names" value="symbol,name,lastSale" />
09                 </bean>
10             </property>
11         </bean>
12     </property>
13 </bean>

```

Gracias a que Spring Batch tiene su base en el marco de Spring, es sencillo configurar

```
FlatFileItemWriter
```

el bean proporcionado y conectar el código al código de la aplicación. En este caso, estamos creando el

```
FlatFileItemWriter
```

con el proporcionado

```
DelimitedLineAggregator
```

, especificando el carácter de canalización como delimitador y configurando el

```
fieldExtractor
```

uso del

```
BeanWrapperFieldExtractor
```

.

El

```
BeanWrapperFieldExtractor
```

toma la lista de registros de StockData que se envía a

```
ItemStreamWriter
```

y extrae los campos especificados por la lista delimitada por comas de los nombres de campo que se encuentran en el bean StockData. Finalmente, especificando el recurso para la salida que en este caso es el archivo extract-example.txt y se escribe en el directorio / data / example / excel.

Ahora todo lo que tenemos que hacer es conectar a los dos escritores delegados a nuestro

```
MultiFormatItemWriter
```

. Asegúrese de llamar a los escritores delegados en los métodos apropiados y ¡habremos terminado! Aquí es cómo se verá el listado final de códigos para

```
MultiFormatItemWriter
```

:

```

01 package com.keyhole.example;
02
03 import java.io.IOException;
04 import java.util.List;
05
06 import org.springframework.batch.core.StepExecution;
07 import org.springframework.batch.core.annotation.AfterStep;
08 import org.springframework.batch.core.annotation.BeforeStep;
09 import org.springframework.batch.item.ExecutionContext;
10 import org.springframework.batch.item.ItemStreamException;
11 import org.springframework.batch.item.ItemStreamWriter;
12 import org.springframework.batch.item.file.FlatFileItemWriter;
13 import org.springframework.beans.factory.annotation.Autowired;
14 import org.springframework.beans.factory.annotation.Qualifier;
15 import org.springframework.context.annotation.Scope;
16 import org.springframework.stereotype.Component;
17
18 import com.keyhole.example.poi.StockData;
19 import com.keyhole.example.poi.StockDataExcelWriter;
20
21 @Component("multiFormatItemWriter")
22 @Scope("step")
23 public class MultiFormatItemWriter implements ItemStreamWriter<StockData> {
24
25     @Autowired
26     private StockDataExcelWriter stockDataExcelWriter;
27
28     @Autowired
29     @Qualifier("pipeDelimitedExtractFile")
30     private FlatFileItemWriter<StockData> extractWriter;
31
32     @Override
33     public void write(List<? extends StockData> items) throws Exception {
34         stockDataExcelWriter.write(items);
35         extractWriter.write(items);
36     }
37
38     @BeforeStep
39     public void beforeStep(StepExecution stepExecution) {

```

```

47     }
48     @Override
49     public void open(ExecutionContext executionContext) throws ItemStreamException {
50         extractWriter.open(executionContext);
51     }
52
53     @Override
54     public void update(ExecutionContext executionContext) throws ItemStreamException {
55         extractWriter.update(executionContext);
56     }
57
58     @Override
59     public void close() throws ItemStreamException {
60         extractWriter.close();
61     }
62
63 }

```

Como puede ver, realmente no hay mucho trabajo que hacer aquí y eso es lo que quería señalar. Realmente no había mostrado cuán simples pueden ser algunas soluciones de negocios utilizando algunos de los lectores y escritores integrados.

Pensamientos finales

Ahora sí mencioné que había un par de maneras de abordar esto. El segundo utilizaría el

CompositeItemWriter

que viene con Spring Batch. Hace casi lo mismo que he hecho aquí, solo que toma una lista de ellos

ItemWriters

y los recorre en cada método que se implementa.

En ese caso, habría convertido mi

StockDataExcelWriter

para implementar la

ItemStreamReader

interfaz y

MultiFormatOutputWriter

se reemplazaría con el

CompositeItemWriter

, que se configuraría en el xml de configuración del trabajo. Incluso menos código.

Así que mi punto con este artículo de hoy es expresar lo fácil que se pueden resolver las tareas más comunes y las soluciones empresariales con varios de los componentes ya implementados que se proporcionan con Spring Batch.

Este y los otros ejemplos se pueden encontrar en GitHub en la siguiente ubicación: <https://github.com/jonny-hackett/batch-example>.

Referencia: Spring Batch: escritor de salida de formato múltiple de nuestro socio de JCG, Jonny Hackett, en el blog Keyhole Software.

Etiquetado con: SPRING SPRING BATCH



(0 rating, 0 votos)

Debes ser un miembro registrado para calificar esto. [Iniciar la discusión](#) [153 Vistas](#) [Tweet it!](#)

¿Quieres saber cómo desarrollar tu conjunto de habilidades para convertirte en un Java Rockstar?

Suscríbete a nuestro boletín para comenzar Rocking

Para comenzar, te regalamos nuestros libros electrónicos más

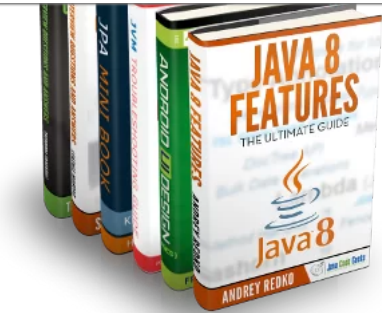
vendidos **GRATIS!**

1. Mini libro de JPA
2. Guía de resolución de problemas de JVM
3. Tutorial JUnit para Pruebas Unitarias
4. Tutorial Anotaciones Java
5. Preguntas de la entrevista de Java
6. Preguntas de la entrevista de primavera
7. Diseño de la interfaz de usuario de Android

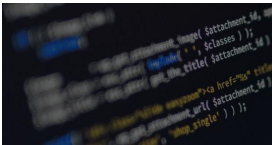
y muchos más

Enter your e-mail...

☐ Estoy de acuerdo con los Términos y Política de Privacidad



¿TE GUSTA ESTE ARTÍCULO? LEER MÁS DE JAVA CODE GEEKS



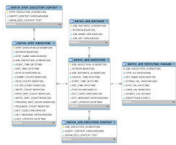
Web Hosting para
JAVA™

Ad ANW® Hosting JAVA™ Tomcat



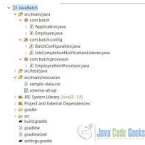
Scaling Spring Batch -
Step Partitioning

javacodegeeks.com

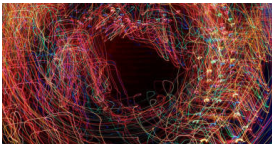


Spring Batch Tutorial – Java Batch Tutorial
The ULTIMATE Guide
(PDF Download)

javacodegeeks.com



javacodegeeks.com



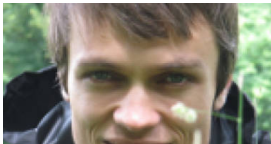
CQRS and Event
Sourcing for dummies

javacodegeeks.com



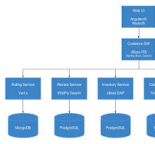
Query Databases
Using Java Streams

javacodegeeks.com



Spring Batch Tutorial
with Spring Boot and
Java Configuration


javacodegeeks.com



Writing end tc
for a microsef
architecture

javacodegeeks.com

Deja una respuesta

 Start the discussion...

Este sitio utiliza Akismet para reducir el spam. [Aprenda cómo se procesan los datos de sus comentarios](#) .

☒ Suscribirse ▼

BASE DE CONOCIMIENTOS

- Los cursos
 - Ejemplos
 - Minibooks
 - Recursos
 - Tutoriales
- FOGONADURA

SALÓN DE LA FAMA

- Serie "Tutorial completo de aplicaciones para Android"
- 11 sitios web de aprendizaje en línea que deberías visitar
- Ventajas y desventajas de la computación en la nube - Pros y contras de la computación en la nube
- Tutorial de Android de Google Maps
- Tutorial de Android JSON con Gson

ACERCA DE JAVA CODE GEEKS

JCGs (Java Code Geeks) es una comunidad en línea independiente enfocada en ser el centro de recursos para desarrolladores de Java a Java; dirigido al arquero líder del equipo técnico (desarrollador senior), jefe de proyecto y desarrollador por igual. Los JCG sirven a las comunidades Java, SOA, Agile y Telecomunicaciones diarias escritas por expertos en dominios, artículos, tutoriales, revisiones de fragmentos de código y proyectos de código abierto.

RENUNCIA

Todas las marcas comerciales y marcas registradas que aparecen en Java son propiedad de sus respectivos dueños. Java es una marca comercial registrada de Oracle Corporation en los Estados Unidos y en otros países.



Código .NET Geeks	Diferencia entre Comparator y Comparable en Java
Java Code Geeks	GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial
Código de sistema Geeks	Mejores Prácticas de Java - Vector vs ArrayList vs HashSet
Código web Geeks	

Java Code Geeks y todo el contenido es copyright © 2010-2018, Exelixis Media PC | Términos de uso | Política de privacidad | Contacto

