ABOUT RAJAGOPAL PARTHASARATHI

Rajagopal works in software industry solving enterprise-scale problems for customers across geographies specializing in open source distributed platforms. He currently holds masters in computer science with focus on cloud computing from Illinois Institute of Technology. His current interests include but not limited to machine learning and big data engineering.

Spring Batch Hibernate Example

Posted by: Rajagopal ParthaSarathi in Batch March 15th, 2018 1 Comment 65 Views

This article is a tutorial about Spring Batch with Hibernate. We will use Spring Boot to speed our development process.

1. Introduction

Spring Batch is a lightweight, scale-able and comprehensive batch framework to handle data at massive scale. Spring Batch builds upon the spring framework to provide intuitive and easy configuration for executing batch applications. Spring Batch provides reusable functions essential for processing large volumes of records, including cross-cutting concerns such as logging/tracing, transaction management, job processing statistics, job restart, skip and resource management.

Spring Batch has a layered architecture consisting of three components:

- Application – Contains custom code written by developers.
- Batch Core – Classes to launch and control batch job.
- Batch Infrastructure – Reusable code for common functionalities needed by core and Application.

Let us dive into spring batch with a simple example of reading persons from a CSV file and loading them into embedded HSQL Database. Since we use the embedded database, data will not be persisted across sessions.

2. Technologies Used

- Java 1.8.101 (1.8.x will do fine)
- Gradle 4.4.1 (4.x will do fine)
- IntelliJ Idea (Any Java IDE would work)
- Rest will be part of the Gradle configuration.

3. Spring Batch Project

Spring Boot Starters provides more than 30 starters to ease the dependency management for your project. The easiest way to generate a Spring Boot project is via Spring starter tool with the steps below:

- Navigate to https://start.spring.io/.
- Select Gradle Project with Java and Spring Boot version 2.0.0.
- Add Batch, JPA and HSqlDB in the "search for dependencies".
- Enter the group name as com.JCG and artifact as SpringBatchHibernate.
- Click the Generate Project button.

A Gradle Project will be generated. If you prefer Maven, use Maven instead of Gradle before generating the project. Import the project into your Java IDE.

3.1 Gradle File

Below we can see the generated build file for our project.

build.gradle

```
01 buildscript {<font></font>
02 ext {<font></font>
03 springBootVersion = '2.0.0.RELEASE'<font></font>
04 }<font></font>
05 repositories {<font></font>
06 mavenCentral()<font></font>
07 }<font></font>
08 dependencies {<font></font>
09 classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")<font></font>
10 }<font></font>
11 }<font></font>
12 apply plugin: 'java'<font></font>
13 apply plugin: 'eclipse'<font></font>
14 apply plugin: 'idea'<font></font>
15 apply plugin: 'org.springframework.boot'<font></font>
16 apply plugin: 'io.spring.dependency-management'<font></font>
17 group = 'com.JCG'<font></font>
18 version = '0.0.1-SNAPSHOT'<font></font>
19 sourceCompatibility = 1.8<font></font>
20 repositories {<font></font>
21 mavenCentral()<font></font>
22 }<font></font>
23 dependencies {<font></font>
24 compile('org.springframework.boot:spring-boot-starter-batch')<font></font>
25 compile('org.springframework.boot:spring-boot-starter-data-jpa')<font></font>
26 runtime('org.hsqldb:hsqldb')<font></font>
27 compile "org.projectlombok:lombok:1.16.8"<font></font>
28 testCompile('org.springframework.boot:spring-boot-starter-test')<font></font>
29 testCompile('org.springframework.batch:spring-batch-test')<font></font>
30 }<font></font>
```

- Spring Boot Version 2.0 se especifica en la línea 3.
- El complemento Idea se ha aplicado para admitir Idea IDE en la línea 14.
- Las líneas 23-29 declaran las dependencias necesarias para el proyecto con cada descarga de la última versión de spring.io.
- La línea 27 declara la

  ```
  Lombok
  ```

  dependencia que se utiliza para reducir el código repetitivo de mecanografía.

3.2 Archivo de datos

- Crea un archivo de muestra sample-data.csv.
- Consta de dos columnas: nombre y apellido.
- El archivo debe estar en la ruta

  ```
  src/main/resources
  ```

  .

Muestra de CSV

```
1 FirstName,LastName<font></font>
2 Jill,Doe<font></font>
3 Joe,Doe<font></font>
4 Justin,Doe<font></font>
5 Jane,Doe<font></font>
6 John,Doe<font></font>
```

- Line1 indica el encabezado del archivo CSV. Será ignorado por lotes de primavera mientras lee el archivo.

3.3 Spring Batch Configuration

A continuación, cubriremos la configuración de Java para Spring Boot, Batch e Hibernate. Discutiremos cada parte de la configuración a continuación.

Clase de aplicación

```
01 package  com.JCG;<font></font>
```

```
<font></font>
import org.springframework.boot.SpringApplication;<font></font>
import org.springframework.boot.autoconfigure.SpringBootApplication;<font></font>
<font></font>
@SpringBootApplication<font></font>
public class Application {<font></font>
<font></font>
    public static void main(String[] args) {<font></font>
        SpringApplication.run(Application.class, args);<font></font>
    }<font></font>
}<font></font>
```

- Especificamos nuestra aplicación como la aplicación Springboot en la Línea 6. Se encarga de toda la magia de Configuración automática. Spring boot trabaja en la filosofía de la convención sobre la configuración. Proporciona valores predeterminados razonables y permite anular con la configuración adecuada.
- La línea 10 inicia nuestra aplicación con la configuración especificada en la sección siguiente.

Configuración de lote

```
package com.JCG.config;<font></font>
<font></font>
import com.JCG.model.Person;<font></font>
import com.JCG.model.PersonRepository;<font></font>
import org.slf4j.Logger;<font></font>
import org.slf4j.LoggerFactory;<font></font>
import org.springframework.batch.core.*;<font></font>
import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;<font></font>
import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;<font></font>
import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;<font></font>
import org.springframework.batch.core.launch.support.RunIdIncrementer;<font></font>
import org.springframework.batch.item.ItemProcessor;<font></font>
import org.springframework.batch.item.database.JpaItemWriter;<font></font>
import org.springframework.batch.item.file.FlatFileItemReader;<font></font>
import org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper;<font></font>
import org.springframework.batch.item.file.mapping.DefaultLineMapper;<font></font>
import org.springframework.batch.item.file.transform.DelimitedLineTokenizer;<font></font>
import org.springframework.beans.factory.annotation.Autowired;<font></font>
import org.springframework.context.annotation.Bean;<font></font>
import org.springframework.context.annotation.Configuration;<font></font>
import org.springframework.core.io.ClassPathResource;<font></font>
<font></font>
import javax.persistence.EntityManagerFactory;<font></font>
<font></font>
@Configuration<font></font>
@EnableBatchProcessing<font></font>
public class BatchConfiguration {<font></font>
    @Autowired<font></font>
    public JobBuilderFactory jobBuilderFactory;<font></font>
<font></font>
    @Autowired<font></font>
    public StepBuilderFactory stepBuilderFactory;<font></font>
<font></font>
    @Autowired<font></font>
    EntityManagerFactory emf;<font></font>
<font></font>
    @Autowired<font></font>
    PersonRepository personRepository;<font></font>
<font></font>
    private static final Logger log = LoggerFactory.getLogger(BatchConfiguration.class);<font></font>
<font></font>
    @Bean<font></font>
    public FlatFileItemReader reader() {<font></font>
        FlatFileItemReader reader = new FlatFileItemReader<>();<font></font>
        reader.setResource(new ClassPathResource("sample-data.csv"));<font></font>
        reader.setLinesToSkip(1);<font></font>
<font></font>
        DefaultLineMapper lineMapper = new DefaultLineMapper<>();<font></font>
        DelimitedLineTokenizer tokenizer = new DelimitedLineTokenizer();<font></font>
        tokenizer.setNames("firstName", "lastName");<font></font>
<font></font>
        BeanWrapperFieldSetMapper fieldSetMapper = new BeanWrapperFieldSetMapper<>();<font></font>
        fieldSetMapper.setTargetType(Person.class);<font></font>
<font></font>
        lineMapper.setFieldSetMapper(fieldSetMapper);<font></font>
        lineMapper.setLineTokenizer(tokenizer);<font></font>
        reader.setLineMapper(lineMapper);<font></font>
<font></font>
        return reader;<font></font>
    }<font></font>
<font></font>
<font></font>
    @Bean<font></font>
    public JpaItemWriter writer() {<font></font>
        JpaItemWriter writer = new JpaItemWriter();<font></font>
        writer.setEntityManagerFactory(emf);<font></font>
        return writer;<font></font>
    }<font></font>
<font></font>
    @Bean<font></font>
    public ItemProcessor<Person, Person> processor() {<font></font>
        return (item) -> {<font></font>
            item.concatenateName();<font></font>
            return item;<font></font>
        };<font></font>
    }<font></font>
<font></font>
    @Bean<font></font>
    public Job importUserJob(JobExecutionListener listener) {<font></font>
        return jobBuilderFactory.get("importUserJob")<font></font>
                .incrementer(new RunIdIncrementer())<font></font>
                .listener(listener)<font></font>
                .flow(step1())<font></font>
                .end()<font></font>
                .build();<font></font>
    }<font></font>
<font></font>
    @Bean<font></font>
    public Step step1() {<font></font>
```

```
090         return stepBuilderFactory.get("step1")<font></font>
091                 .<Person, Person>chunk(10)<font></font>
092                 .reader(reader())<font></font>
093                 .processor(processor())<font></font>
094                 .writer(writer())<font></font>
095                 .build();<font></font>
096     }<font></font>
097 <font></font>
098     @Bean<font></font>
099     public JobExecutionListener listener() {<font></font>
100         return new JobExecutionListener() {<font></font>
101 <font></font>
102 <font></font>
103             @Override<font></font>
104             public void beforeJob(JobExecution jobExecution) {<font></font>
105                 /**<font></font>
106                  * As of now empty but can add some before job conditions<font></font>
107                  */<font></font>
108             }<font></font>
109 <font></font>
110             @Override<font></font>
111             public void afterJob(JobExecution jobExecution) {<font></font>
112                 if (jobExecution.getStatus() == BatchStatus.COMPLETED) {<font></font>
113                     log.info("!!! JOB FINISHED! Time to verify the results");<font></font>
114                     personRepository.findAll().<font></font>
115                             forEach(person -> log.info("Found <" + person + "> in the database."));<font>
    </font>
116                 }<font></font>
117             }<font></font>
118         };<font></font>
119     }<font></font>
120 }<font></font>
```

La línea 25 indica que es una clase de configuración y debe recogerse con un arranque de resorte para conectar los beans y las dependencias. La línea 26 se usa para habilitar el soporte por lotes para nuestra aplicación. Spring define a

```
Job
```

que contiene múltiples

```
Step
```

para ser ejecutado. En nuestro ejemplo, usamos solo un paso para nuestro

```
importUserJob
```

. Usamos una

```
JobExecutionListener
```

para rastrear la ejecución del trabajo que trataremos a continuación. A

```
Step
```

podría ser a

```
TaskletStep
```

(contiene una sola función para la ejecución) o

```
Step
```

que incluye a

```
Reader
```

,

```
Processor
```

y

```
Writer
```

. En el ejemplo anterior, lo hemos usado

```
Step
```

.

3.3.1 Lector

Las líneas 42-60 incluyen nuestra configuración de lector. Usamos

```
FlatFileItemReader
```

para leer de nuestro archivo CSV. La ventaja de utilizar un lector incorporado es que maneja las fallas de la aplicación con elegancia y admite reinicios. También puede omitir líneas durante errores con un límite de omisión configurable.

Necesita los siguientes parámetros para leer correctamente el archivo línea por línea.

- Recurso: la aplicación lee de un recurso de ruta de clase como se especifica en la línea 45. Nos salteamos la línea de encabezado especificando

  ```
  setLinesToSkip
  ```

  .

- Line Mapper: se usa para mapear una línea leída del archivo en una representación utilizable por nuestra aplicación. Usamos `DefaultLineMapper`

  desde Spring Infrastructure. Esto, a su vez, usa dos clases para mapear la línea de nuestro modelo `Person`

  . Utiliza un `LineTokenizer`

  para dividir una sola línea en tokens en función de los criterios especificados y a `FieldSetMapper`

  para asignar los tokens en un fieldset utilizable por nuestra aplicación.
  - Tokenizador de línea: usamos `DelimitedLineTokenizer`

    para tokenizar las líneas dividiendo con una coma. Por defecto, la coma se usa como el tokenizador. También especificamos los nombres de token para que coincidan con los campos de nuestra clase de modelo.
  - `FieldSetMapper`

    - Aquí estamos usando `BeanWrapperFieldSetMapper`

    para asignar los datos a un bean por sus nombres de propiedad. Los nombres exactos de los campos se especifican en el tokenizer que se usará.

- Line Mapper está asignado al lector en la línea 57.

El lector lee los elementos en el `chunk(10)`

que se especifica mediante la `chunk`

configuración en la línea 91.

3.3.2 Procesador

Spring no ofrece un procesador incorporado y generalmente se deja a la implementación personalizada. Aquí, estamos usando una función lambda para transformar el `Person`

objeto entrante . Llamamos a la `concatenateName`

función para concatenar el primer nombre y apellido. Devolvemos el artículo modificado al escritor. El procesador hace su ejecución un elemento a la vez.

3.3.3 Escritor

Aquí, estamos usando `JpaItemWriter`

para escribir el objeto modelo en la base de datos. JPA utiliza hibernate como el proveedor de persistencia para persistir los datos. El escritor solo necesita que el modelo se escriba en la base de datos. Agrega los elementos recibidos del procesador y vacía los datos.

3.3.4 Oyente

`JobExecutionListener`

ofrece los métodos `beforeJob`

para ejecutar antes de que comience el trabajo y `afterJob`

que se ejecuta después de que se haya completado el trabajo. En general, estos métodos se utilizan para recopilar diversas métricas de trabajos y, en ocasiones, para inicializar constantes. Aquí, usamos `afterJob`

para verificar si los datos se conservaron. Utilizamos un método de repositorio `findAll`

para buscar a todas las personas de nuestra base de datos y mostrarlas.

3.4 Configuración del modelo / hibernación

application.properties

```
spring.jpa.hibernate.ddl-auto=create-drop<font></font>
spring.jpa.show-sql=true<font></font>
```

Aquí especificamos que las tablas deben crearse antes del uso y destruirse cuando la aplicación finaliza. Además, hemos especificado la configuración para mostrar SQL ejecutado por hibernación en la consola para la depuración. El resto de la configuración de cableado

```
Datasource
```

para hibernar y luego a JPA

```
EntityManagerfactory
```

es manejado por

```
JpaRepositoriesAutoConfiguration
```

y

```
HibernateJpaAutoConfiguration
```

.

Clase de modelo (persona)

```
package com.JCG.model;<font></font>
<font></font>
import lombok.*;<font></font>
<font></font>
import javax.persistence.Entity;<font></font>
import javax.persistence.GeneratedValue;<font></font>
import javax.persistence.Id;<font></font>
import javax.persistence.Transient;<font></font>
<font></font>
@Entity<font></font>
@Getter<font></font>
@Setter<font></font>
@NoArgsConstructor<font></font>
@RequiredArgsConstructor<font></font>
@ToString(exclude={"firstName","lastName"})<font></font>
public class Person {<font></font>
<font></font>
    @Id<font></font>
    @GeneratedValue<font></font>
    private int id;<font></font>
<font></font>
    @Transient<font></font>
    @NonNull<font></font>
    private String lastName;<font></font>
<font></font>
    @Transient<font></font>
    @NonNull<font></font>
    private String firstName;<font></font>
<font></font>
    @NonNull<font></font>
    private String name;<font></font>
<font></font>
    public void concatenateName(){<font></font>
        this.setName(this.firstName+" "+this.lastName);<font></font>
    }<font></font>
}<font></font>
```

Una clase modelo debe ser anotada con

```
Entity
```

para ser utilizada por contenedor de primavera. Hemos usado

```
Lombok
```

anotaciones para generar getter, setter y Constructor de nuestros campos. Los campos

```
firstName
```

y

```
lastName
```

anotados como

```
Transient
```

para indicar que estos campos no se deben persistieron a la base de datos. Hay un

```
id
```

campo que se anota para generar la secuencia de hibernación mientras se guarda en la base de datos.

Clase de repositorio (PersonRepository)

```
package com.JCG.model;<font></font>
<font></font>
```

```
3 import org.springframework.data.jpa.repository.JpaRepository;<font></font>
4 import org.springframework.stereotype.Repository;<font></font>
5 <font></font>
6 @Repository<font></font>
7 public interface PersonRepository extends JpaRepository<Person,Integer> {<font></font>
8 }<font></font>
```

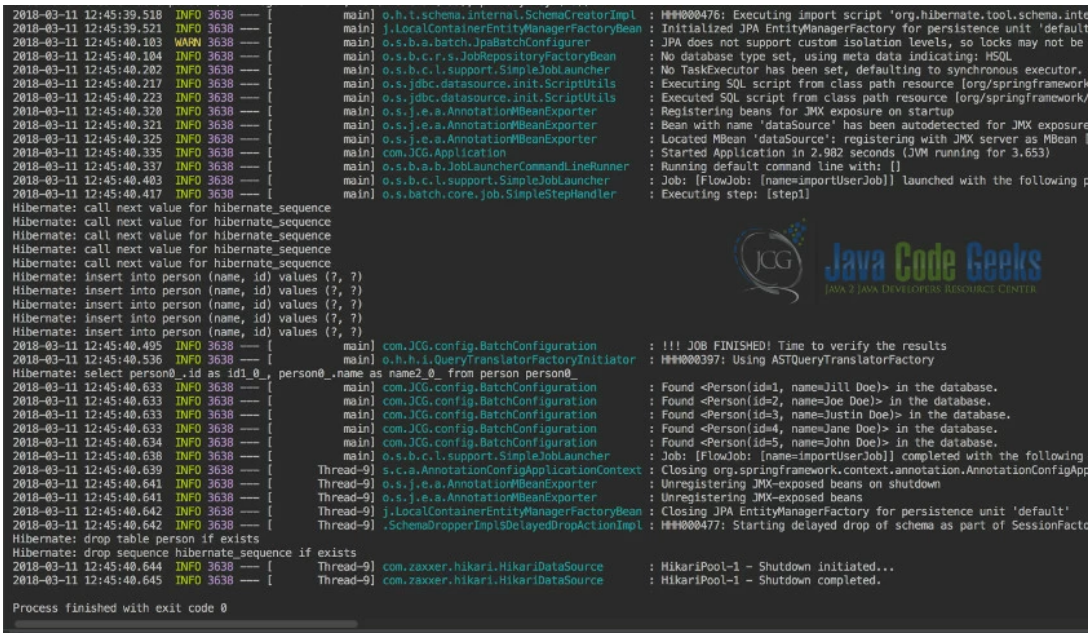
Esta es solo una implementación de repositorio de Spring JPA. Para ver un ejemplo detallado, consulte el ejemplo del Repositorio JPA .

4. Resumen

Ejecute la

```
Application
```

clase desde un IDE de Java. Se mostrará un resultado similar a la captura de pantalla siguiente. En este ejemplo, vimos una manera simple de configurar una aplicación Spring Batch Project.



SpringBatchHibernate Logs

5. Descargue el código fuente

Descargar
Puede descargar el código fuente completo de este ejemplo aquí: **SpringBatchHibernate**

Etiquetado con: HIBERNATE JPA SPRING SPRING BATCH

ME GUSTA ESTE ARTICULO? LEER MÁS DE JAVA CODE GEEKS

Spring Batch Ejemplo paso a paso

15 de septiembre de 2017

Ejemplo de lote de muelles de cuarzo

25 de julio de 2016

Ejemplo de multiproceso de lotes de primavera

6 de junio de 2016

Deja una respuesta

1 comentario sobre "Spring Batch Hibernate Example"

Join the discussion

Subscribe ▾ ▲ newest ▲ oldest ▲ most voted

Guest

lubican

Caused by: org.springframework.beans.NotWritablePropertyException: Invalid property 'firstName' of bean class [com.JCG.model.Person]: Bean property 'firstName' is not writable or has an invalid setter method. Does the parameter type of the setter match the return type of the getter?
at org.springframework.beans.BeanWrapperImpl.createNotWritablePropertyException(BeanWrapperImpl.java:247) ~[spring-beans-5.0.4.RELEASE.jar:5.0.4.RELEASE]
at org.springframework.beans.AbstractNestablePropertyAccessor.processLocalProperty(AbstractNestablePropertyAccessor.java:426) ~[spring-beans-5.0.4.RELEASE.jar:5.0.4.RELEASE]
at org.springframework.beans.AbstractNestablePropertyAccessor.setPropertyValue(AbstractNestablePropertyAccessor.java:278) ~[spring-beans-5.0.4.RELEASE.jar:5.0.4.RELEASE]
at org.springframework.beans.AbstractNestablePropertyAccessor.setPropertyValue(AbstractNestablePropertyAccessor.java:266) ~[spring-beans-5.0.4.RELEASE.jar:5.0.4.RELEASE]
at org.springframework.beans.AbstractPropertyAccessor.setPropertyValues(AbstractPropertyAccessor.java:97) ~[spring-beans-5.0.4.RELEASE.jar:5.0.4.RELEASE]
at org.springframework.validation.DataBinder.applyPropertyValues(DataBinder.java:839) ~[spring-context-5.0.4.RELEASE.jar:5.0.4.RELEASE]
at org.springframework.validation.DataBinder.doBind(DataBinder.java:735) ~[spring-context-5.0.4.RELEASE.jar:5.0.4.RELEASE]
at org.springframework.validation.DataBinder.bind(DataBinder.java:720) ~[spring-context-5.0.4.RELEASE.jar:5.0.4.RELEASE]
en org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper.mapFieldSet (BeanWrapperFieldSetMapper.java:198) ~ [spring-batch-infrastructure-4.0.0.RELEASE.jar: 4.0.0.RELEASE]
en org.springframework. batch.item.file.mapping.DefaultLineMapper.mapLine (DefaultLineMapper.java:43) ~ [spring-batch-infrastructure-4.0.0.RELEASE.jar: 4.0.0.RELEASE]
en org.springframework.batch.item.file .FlatFileItemReader.doRead (FlatFileItemReader.java:180) ~ [spring-

batch-infrastructure-4.0.0.RELEASE.jar: 4.0.0.RELEASE]
... 50 marcos comunes omitidos

+ 0 − respuesta

Hace 2 horas