
[ANDROID](#)
[CORE JAVA](#)
[DESKTOP JAVA](#)
[ENTERPRISE JAVA](#)
[JAVA BASICS](#)
[JVM LANGUAGES](#)
[SOFTWARE DEVELOPMENT](#)
[DEVOPS](#)
[Home](#) » [Enterprise Java](#) » [spring](#) » [Batch](#) » [Spring Batch Hibernate Example](#)

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](#) on [March 15th, 2018](#) with [1 Comment](#) and [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.

Want to master Spring Framework ?

Subscribe to our newsletter and download the Spring Framework Cookbook [right now!](#)

In order to help you master the leading and innovative Java framework, we have compiled a kick-ass guide with all its major features and use cases! Besides studying them online you may download the eBook in PDF format!

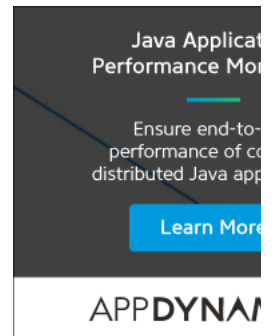
Email address:

[Sign up](#)

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.

Ahora mismo, la información que está buscando no está disponible como el tu



HOJA INFORMATIVA

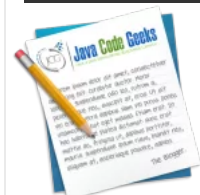
¡188,315 personas que ya han actualizado su información actualizada disfrutando de actualizaciones semanales y **¡188,315** blancos de cortesía! **Únete a ellos ahora** **acceso exclusivo** a las noticias más recientes en el mundo de Java, así como sobre Android, Scala, Groovy tecnologías relacionadas.

Dirección de correo electrónico:

☒ Reciba alertas de trabajo de desarrollador en su área

[Regístrate](#)

ÚNETE A NOSOTROS



Con **1,2** mensuales **500** artículos ubicados en sitios de Java. Con buscador animado nosotros un blog con contenido único e in entonces debería consultar nues



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>
```

```

02 <font></font>
03 import org.springframework.boot.SpringApplication;<font></font>
04 import org.springframework.boot.autoconfigure.SpringBootApplication;<font></font>
05 <font></font>
06 @SpringBootApplication<font></font>
07 public class Application {<font></font>
08 <font></font>
09     public static void main(String[] args) {<font></font>
10         SpringApplication.run(Application.class, args);<font></font>
11     }<font></font>
12 }<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

```

001 package com.JCG.config;<font></font>
002 <font></font>
003 import com.JCG.model.Person;<font></font>
004 import com.JCG.model.PersonRepository;<font></font>
005 import org.slf4j.Logger;<font></font>
006 import org.slf4j.LoggerFactory;<font></font>
007 import org.springframework.batch.core.*;<font></font>
008 import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;<font></font>
009 import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;<font></font>
010 import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;<font></font>
011 import org.springframework.batch.core.launch.support.RunIdIncrementer;<font></font>
012 import org.springframework.batch.item.ItemProcessor;<font></font>
013 import org.springframework.batch.item.database.JpaItemWriter;<font></font>
014 import org.springframework.batch.item.file.FlatFileItemReader;<font></font>
015 import org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper;<font></font>
016 import org.springframework.batch.item.file.mapping.DefaultLineMapper;<font></font>
017 import org.springframework.batch.item.file.transform.DelimitedLineTokenizer;<font></font>
018 import org.springframework.beans.factory.annotation.Autowired;<font></font>
019 import org.springframework.context.annotation.Bean;<font></font>
020 import org.springframework.context.annotation.Configuration;<font></font>
021 import org.springframework.core.io.ClassPathResource;<font></font>
022 <font></font>
023 import javax.persistence.EntityManagerFactory;<font></font>
024 <font></font>
025 @Configuration<font></font>
026 @EnableBatchProcessing<font></font>
027 public class BatchConfiguration {<font></font>
028     @Autowired<font></font>
029     public JobBuilderFactory jobBuilderFactory;<font></font>
030 <font></font>
031     @Autowired<font></font>
032     public StepBuilderFactory stepBuilderFactory;<font></font>
033 <font></font>
034     @Autowired<font></font>
035     EntityManagerFactory emf;<font></font>
036 <font></font>
037     @Autowired<font></font>
038     PersonRepository personRepository;<font></font>
039 <font></font>
040     private static final Logger log = LoggerFactory.getLogger(BatchConfiguration.class);<font></font>
041 <font></font>
042     @Bean<font></font>
043     public FlatFileItemReader reader() {<font></font>
044         FlatFileItemReader reader = new FlatFileItemReader<>();<font></font>
045         reader.setResource(new ClassPathResource("sample-data.csv"));<font></font>
046         reader.setLinesToSkip(1);<font></font>
047 <font></font>
048         DefaultLineMapper lineMapper = new DefaultLineMapper<>();<font></font>
049         DelimitedLineTokenizer tokenizer = new DelimitedLineTokenizer();<font></font>
050         tokenizer.setNames("firstName", "lastName");<font></font>
051 <font></font>
052         BeanWrapperFieldSetMapper fieldSetMapper = new BeanWrapperFieldSetMapper<>();<font></font>
053         fieldSetMapper.setTargetType(Person.class);<font></font>
054 <font></font>
055         lineMapper.setFieldSetMapper(fieldSetMapper);<font></font>
056         lineMapper.setLineTokenizer(tokenizer);<font></font>
057         reader.setLineMapper(lineMapper);<font></font>
058 <font></font>
059         return reader;<font></font>
060     }<font></font>
061 <font></font>
062 <font></font>
063     @Bean<font></font>
064     public JpaItemWriter writer() {<font></font>
065         JpaItemWriter writer = new JpaItemWriter();<font></font>
066         writer.setEntityManagerFactory(emf);<font></font>
067         return writer;<font></font>
068     }<font></font>
069 <font></font>
070     @Bean<font></font>
071     public ItemProcessor<Person, Person> processor() {<font></font>
072         return item -> {<font></font>
073             item.concatenarName();<font></font>
074             return item;<font></font>
075         };<font></font>
076     }<font></font>
077 <font></font>
078     @Bean<font></font>
079     public Job importUserJob(JobExecutionListener listener) {<font></font>
080         return jobBuilderFactory.get("importUserJob")<font></font>
081             .incrementer(new RunIdIncrementer())<font></font>
082             .listener(listener)<font></font>
083             .flow(step1())<font></font>
084             .end()<font></font>
085             .build();<font></font>
086     }<font></font>
087 <font></font>
088     @Bean<font></font>
089     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>
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

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 saltamos 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

```
1 spring.jpa.hibernate.ddl-auto=create-drop<font></font>
2 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)

```
01 package com.JCG.model;<font></font>
02 <font></font>
03 import lombok.*;<font></font>
04 <font></font>
05 import javax.persistence.Entity;<font></font>
06 import javax.persistence.GeneratedValue;<font></font>
07 import javax.persistence.Id;<font></font>
08 import javax.persistence.Transient;<font></font>
09 <font></font>
10 @Entity<font></font>
11 @Getter<font></font>
12 @Setter<font></font>
13 @NoArgsConstructor<font></font>
14 @RequiredArgsConstructor<font></font>
15 @ToString(exclude={"firstName","lastName"})<font></font>
16 public class Person {<font></font>
17 <font></font>
18     @Id<font></font>
19     @GeneratedValue<font></font>
20     private int id;<font></font>
21 <font></font>
22     @Transient<font></font>
23     @NonNull<font></font>
24     private String lastName;<font></font>
25 <font></font>
26     @Transient<font></font>
27     @NonNull<font></font>
28     private String firstName;<font></font>
29 <font></font>
30     @NonNull<font></font>
31     private String name;<font></font>
32 <font></font>
33     public void concatenateName(){<font></font>
34         this.setName(this.firstName+" "+this.lastName);<font></font>
35     }<font></font>
36 }<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)

```
1 package com.JCG.model;<font></font>
2 <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.

```

2018-03-11 12:45:39.518 INFO 3638 --- [main] o.h.t.schema.internal.SchemaCreatorImpl : ###000476: Executing import script 'org.hibernate.tool.schema.inte
2018-03-11 12:45:39.521 INFO 3638 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default
2018-03-11 12:45:40.103 WARN 3638 --- [main] o.s.b.a.batch.JpaBatchConfigurer : JPA does not support custom isolation levels, so locks may not be
2018-03-11 12:45:40.104 INFO 3638 --- [main] o.s.b.c.r.s.JobRepositoryFactoryBean : No database type set, using meta data indicating: HSQL
2018-03-11 12:45:40.202 INFO 3638 --- [main] o.s.b.c.l.support.SimpleJobLauncher : No TaskExecutor has been set, defaulting to synchronous executor.
2018-03-11 12:45:40.217 INFO 3638 --- [main] o.s.jdbc.datasource.init.ScriptUtils : Executing SQL script from class path resource [org/springframework
2018-03-11 12:45:40.223 INFO 3638 --- [main] o.s.jdbc.datasource.init.ScriptUtils : Executed SQL script from class path resource [org/springframework/
2018-03-11 12:45:40.320 INFO 3638 --- [main] o.s.j.e.s.AnnotationMBeanExporter : Registering beans for JMX exposure on startup
2018-03-11 12:45:40.321 INFO 3638 --- [main] o.s.j.e.s.AnnotationMBeanExporter : Bean with name 'dataSource' has been autodetected for JMX exposure
2018-03-11 12:45:40.325 INFO 3638 --- [main] o.s.j.e.s.AnnotationMBeanExporter : Located MBean 'dataSource': registering with JMX server as MBean
2018-03-11 12:45:40.335 INFO 3638 --- [main] com.JCG.Application : Started Application in 2.982 seconds (JVM running for 3.653)
2018-03-11 12:45:40.337 INFO 3638 --- [main] o.s.b.a.b.JobLauncherCommandLineRunner : Running default command line with: []
2018-03-11 12:45:40.403 INFO 3638 --- [main] o.s.b.c.l.support.SimpleJobLauncher : Job: [FlowJob: [name=importUserJob]] launched with the following p
2018-03-11 12:45:40.417 INFO 3638 --- [main] o.s.batch.core.job.SimpleStepHandler : Executing step: [step1]

Hibernate: call next value for hibernate_sequence
Hibernate: call next value for hibernate_sequence
Hibernate: call next value for hibernate_sequence
Hibernate: call next value for hibernate_sequence
Hibernate: insert into person (name, id) values (?, ?)
Hibernate: insert into person (name, id) values (?, ?)
Hibernate: insert into person (name, id) values (?, ?)
Hibernate: insert into person (name, id) values (?, ?)
2018-03-11 12:45:40.495 INFO 3638 --- [main] com.JCG.config.BatchConfiguration : !!! JOB FINISHED! Time to verify the results
2018-03-11 12:45:40.536 INFO 3638 --- [main] o.h.h.i.QueryTranslatorFactoryInitiator : ###000397: Using ASTQueryTranslatorFactory
Hibernate: select person0_.id as id1_0, person0_.name as name2_0_ from person person0_
2018-03-11 12:45:40.633 INFO 3638 --- [main] com.JCG.config.BatchConfiguration : Found <Person(id=1, name=Jill Doe)> in the database.
2018-03-11 12:45:40.633 INFO 3638 --- [main] com.JCG.config.BatchConfiguration : Found <Person(id=2, name=Joe Doe)> in the database.
2018-03-11 12:45:40.633 INFO 3638 --- [main] com.JCG.config.BatchConfiguration : Found <Person(id=3, name=Justin Doe)> in the database.
2018-03-11 12:45:40.633 INFO 3638 --- [main] com.JCG.config.BatchConfiguration : Found <Person(id=4, name=Jane Doe)> in the database.
2018-03-11 12:45:40.634 INFO 3638 --- [main] com.JCG.config.BatchConfiguration : Found <Person(id=5, name=John Doe)> in the database.
2018-03-11 12:45:40.636 INFO 3638 --- [main] o.s.b.c.l.support.SimpleJobLauncher : Job: [FlowJob: [name=importUserJob]] completed with the following
2018-03-11 12:45:40.639 INFO 3638 --- [Thread-9] s.c.a.AnnotationConfigApplicationContext : Closing org.springframework.context.annotation.AnnotationConfigApp
2018-03-11 12:45:40.641 INFO 3638 --- [Thread-9] o.s.j.e.s.AnnotationMBeanExporter : Unregistering JMX-exposed beans on shutdown
2018-03-11 12:45:40.641 INFO 3638 --- [Thread-9] o.s.j.e.s.AnnotationMBeanExporter : Unregistering JMX-exposed beans
2018-03-11 12:45:40.642 INFO 3638 --- [Thread-9] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2018-03-11 12:45:40.642 INFO 3638 --- [Thread-9] .SchemaDropperImpl$DelayedDropActionImpl : ###000477: Starting delayed drop of schema as part of SessionFacto
Hibernate: drop table person if exists
Hibernate: drop sequence hibernate_sequence if exists
2018-03-11 12:45:40.644 INFO 3638 --- [Thread-9] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2018-03-11 12:45:40.645 INFO 3638 --- [Thread-9] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.

Process finished with exit code 0

```

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

¿Desea saber cómo desarrollar sus habilidades para convertirse en **Java Rockstar**?

iSuscríbete a nuestro boletín para comenzar a rocking ahora mismo!

iPara comenzar, te ofrecemos nuestros eBooks más vendidos **GRATIS!**

1. Mini libro de JPA
2. Guía de solución de problemas de JVM
3. JUnit Tutorial para pruebas unitarias
4. Tutorial de anotaciones en 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

Dirección de correo electrónico:

☒ Reciba alertas de trabajo de Java y desarrollador en su área

Registrate



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(AbstractNestablePropertyAcc
essor.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

BASE DE CONOCIMIENTOS

Cursos
Minilibros
Noticias
Recursos
Tutoriales

LA RED CODE GEEKS

.NET Code Geeks
Java Code Geeks
System Code Geeks
Web Code Geeks

HALL OF FAME

Android Alert Dialog Example
Android OnClickListener Example
How to convert Character to String and a String to Character Array in Java
Java Inheritance example
Java write to File Example
java.io.FileNotFoundException – How to solve File Not Found Exception
java.lang.arrayindexoutofboundsexception – How to handle Array Index Out Of Bounds Exception
java.lang.NoClassDefFoundError – How to solve 'No Class Def Found' Error
JSON Example With Jersey + Jackson
Spring JdbcTemplate Example

ABOUT JAVA CODE GEEKS

JCGs (Java Code Geeks) is an independent online community focused o ultimate Java to Java developers resource center; targeted at the techn technical team lead (senior developer), project manager and junior dev JCGs serve the Java, SOA, Agile and Telecom communities with daily is domain experts, articles, tutorials, reviews, announcements, code snipp source projects.

DISCLAIMER

All trademarks and registered trademarks appearing on Java Code Gees property of their respective owners. Java is a trademark or registered t Oracle Corporation in the United States and other countries. Examples i is not connected to Oracle Corporation and is not sponsored by Oracle.

