

(/)

Repositorios con múltiples módulos de datos de resorte

Última modificación: 18 de septiembre de 2019

por baeldung (<https://www.baeldung.com/author/baeldung/>)
(<https://www.baeldung.com/author/baeldung/>)

Primavera (<https://www.baeldung.com/category/spring/>) +

Datos de primavera (<https://www.baeldung.com/category/persistence/spring-persistence/spring-data/>)

Acabo de anunciar el nuevo curso *Learn Spring*, centrado en los fundamentos de Spring 5 y Spring Boot 2:

>> **VER EL CURSO** (</ls-course-start>)

1. Introducción

A veces necesitamos conectarnos a más de una tecnología de base de datos en la misma aplicación.

En este tutorial, exploraremos las diversas opciones de configuración cuando se trata de **usar múltiples módulos Spring Data en la misma aplicación**.

Usemos una tienda de juguetes Spring Boot para explorar el tema.

2. Dependencias requeridas

Primero, necesitamos agregar nuestras dependencias en el archivo *pom.xml*, para que podamos usar los enlaces *spring-boot-starter-data-mongodb* (<https://spring.io/projects/spring-data-mongodb>) y *spring-boot-starter-data-cassandra* (<https://spring.io/projects/spring-data-cassandra>) Spring Boot Data:

```

1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-data-cassandra</artifactId>
4   <version>2.1.8.RELEASE</version>
5 </dependency>
6 <dependency>
7   <groupId>org.springframework.boot</groupId>
8   <artifactId>spring-boot-starter-data-mongodb</artifactId>
9   <version>2.1.8.RELEASE</version>
10 </dependency>

```

3. Configuración de la base de datos

A continuación, **necesitamos configurar las bases de datos reales** utilizando imágenes Docker (<https://docs.docker.com/install/>) preconstruidas de Cassandra (https://hub.docker.com/_/cassandra) y Mongo (https://hub.docker.com/_/mongo):

```

1 $ docker run --name mongo-db -d -p 27017:27017 mongo:latest
2 $ docker run --name cassandra-db -d -p 9042:9042 cassandra:latest

```

Estos dos comandos descargarán automáticamente las últimas imágenes de Cassandra y MongoDB Docker y ejecutarán los contenedores reales.

También debemos reenviar los puertos (con la opción *-p*) desde el interior de los contenedores en el entorno real del sistema operativo, para que nuestra aplicación pueda acceder a las bases de datos.

Debemos crear la estructura de la base de datos para Cassandra utilizando la utilidad *cqlsh*. **Creación de espacio de claves no se puede hacer automáticamente por *CassandraDataAutoConfiguration***, por lo que necesitamos para declarar que el uso de CQL (<https://cassandra.apache.org/doc/latest/cql/>) sintaxis.

Entonces, primero, lo conectamos a la carcasa de *bash* del contenedor Cassandra:

```

1 $ docker exec -it cassandra-db /bin/bash
2 root@419acd18891e:/# cqlsh
3 Connected to Test Cluster at 127.0.0.1:9042.
4 [cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
5 Use HELP for help.
6 cqlsh> CREATE KEYSPACE IF NOT EXISTS baeldung
7 WITH replication = {'class': 'SimpleStrategy', 'replication_factor':1};
8 cqlsh> USE baeldung;
9 cqlsh> CREATE TABLE bookaudit(
10     bookid VARCHAR,
11     rentalrecno VARCHAR,
12     loandate VARCHAR,
13     loaner VARCHAR,
14     primary key(bookid, rentalrecno)
15 );

```

En las líneas 6 y 9, creamos el espacio clave y la tabla relevantes .

Sin embargo, podríamos omitir la creación de la tabla y simplemente confiar en *spring-boot-starter-data-cassandra* para inicializar el esquema, ya que queremos explorar las configuraciones del marco individualmente, este es un paso necesario.

Por defecto, **Mongo no impone ningún tipo de validación de esquema, por lo que no es necesario configurar nada más para ello** .

Finalmente, configuramos la información relevante sobre las bases de datos en nuestra *aplicación* .

```

1 spring.data.cassandra.username=cassandra
2 spring.data.cassandra.password=cassandra
3 spring.data.cassandra.keyspaceName=baeldung
4 spring.data.cassandra.contactPoints=localhost
5 spring.data.cassandra.port=9042
6 spring.data.mongodb.host=localhost
7 spring.data.mongodb.port=27017
8 spring.data.mongodb.database=baeldung

```

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#)

Ok

4. Mecanismos de detección del almacén de datos

Cuando se detectan múltiples módulos Spring Data en el classpath, Spring Framework entra en modo de configuración estricta del repositorio. Esto significa que utiliza diferentes mecanismos de detección debajo, para identificar qué repositorio pertenece a qué tecnología de persistencia.

4.1. Ampliación de las interfaces de repositorio específicas del módulo

El primer mecanismo intenta determinar si un repositorio extiende un tipo de repositorio específico del módulo Spring Data:

```
1 public interface BookAuditRepository extends CassandraRepository<BookAudit, String> {  
2  
3 }
```

Para el propósito de nuestro ejemplo, *BookAudit.java* contiene alguna estructura básica de almacenamiento para rastrear usuarios que prestaron un libro:

```
1 public class BookAudit {  
2     private String bookId;  
3     private String rentalRecNo;  
4     private String loaner;  
5     private String loanDate;  
6  
7     // standard getters and setters  
8 }
```

Lo mismo se aplica a la definición de repositorio relacionada con MongoDB:

```
1 public interface BookDocumentRepository extends MongoRepository<BookDocument, String> {  
2  
3 }
```

Éste almacena el contenido del libro y algunos metadatos relevantes al respecto:

```
1 public class BookDocument {  
2     private String bookId;  
3     private String bookName;  
4     private String bookAuthor;  
5     private String content;  
6  
7     // standard getters and setters  
8 }
```

Cuando se carga el contexto de la aplicación, el marco coincidirá con cada tipo de repositorio utilizando la clase base de la que deriva :

```

1 | @Test
2 | public void givenBookAudit_whenPersistWithBookAuditRepository_thenSuccess() {
3 |
4 |     // given
5 |     BookAudit bookAudit =
6 |         new BookAudit("lorem", "ipsum", "Baeldung", "19:30/20.08.2017");
7 |
8 |     // when
9 |     bookAuditRepository.save(bookAudit);
10 |
11 |    // then
12 |    List<BookAudit> result = bookAuditRepository.findAll();
13 |    assertThat(result.isEmpty(), is(false));
14 |    assertThat(result.contains(bookAudit), is(true));
15 | }

```

Puedes notar que nuestras clases de dominio son simples objetos Java. En esta situación particular, el esquema de la base de datos Cassandra debe crearse externamente con *CQL* como lo hemos hecho en la sección 3.

4.2. Uso de anotaciones específicas de módulo en los objetos de dominio

La segunda estrategia **determina la tecnología de persistencia basada en anotaciones específicas del módulo en los objetos de dominio**.

Extendamos un *repositorio* genérico de *Crud* y confiemos ahora en las anotaciones de objetos administrados para la detección:

```

1 | public interface BookAuditCrudRepository extends CrudRepository<BookAudit, String> {
2 |
3 | }

```

```

1 | public interface BookDocumentCrudRepository extends CrudRepository<BookDocument, String> {
2 |
3 | }

```

El *BookAudit.java* ahora se convierte anotado con la específica Cassandra *@Table* y requiere una clave principal compuesto de:

```

1 | @Table
2 | public class BookAudit {
3 |
4 |     @PrimaryKeyColumn(type = PrimaryKeyType.PARTITIONED)
5 |     private String bookId;
6 |     @PrimaryKeyColumn
7 |     private String rentalRecNo;
8 |     private String loaner;
9 |     private String loanDate;
10 |
11 |    // standard getters and setters
12 | }

```

Elegimos la combinación de *bookId* y *rentalRecNo* como nuestro criterio único, ya que nuestra aplicación simplemente registra un nuevo registro de alquiler cada vez que alguien presta un libro.

Para *BookDocument.java* usamos la anotación *@Document* que es específica de MongoDB:

```

1 | @Document
2 | public class BookDocument {
3 |
4 |     private String bookId;
5 |     private String bookName;
6 |     private String bookAuthor;
7 |     private String content;
8 |
9 |
10 | }

```

La activación de un guardado de *BookDocument* con *CrudRepository* sigue siendo exitosa, pero el tipo devuelto en la línea 11 ahora es *Iterable* en lugar de una *Lista*:

```

1  @Test
2  public void givenBookAudit_whenPersistWithBookDocumentCrudRepository_thenSuccess() {
3
4      // given
5      BookDocument bookDocument =
6          new BookDocument("lorem", "Foundation", "Isaac Asimov", "Once upon a time ...");
7
8      // when
9      bookDocumentCrudRepository.save(bookDocument);
10
11     // then
12     Iterable<BookDocument> resultIterable = bookDocumentCrudRepository.findAll();
13     List<BookDocument> result = StreamSupport.stream(resultIterable.spliterator(), false)
14         .collect(Collectors.toList());
15     assertThat(result.isEmpty(), is(false));
16     assertThat(result.contains(bookDocument), is(true));
17 }

```

4.3. Uso de alcance basado en paquetes

Finalmente, **podemos especificar los paquetes base donde se definen nuestros repositorios**, utilizando las anotaciones *@EnableCassandraRepositories* y *@EnableMongoRepositories*:

```

1  @EnableCassandraRepositories(basePackages="com.baeldung.multipledatamodules.cassandra")
2  @EnableMongoRepositories(basePackages="com.baeldung.multipledatamodules.mongo")
3  public class SpringDataMultipleModules {
4
5      public static void main(String[] args) {
6          SpringApplication.run(SpringDataMultipleModules.class, args);
7      }
8  }

```

Como podemos ver en las líneas 1 y 2, **este modo de configuración supone que usamos diferentes paquetes para los repositorios Cassandra y MongoDB**.

5. Conclusión

En este tutorial, hemos configurado una aplicación Spring Boot simple para usar dos módulos Spring Data diferentes de tres maneras.

Como primer ejemplo, *extendimos CassandraRepository y MongoRepository* y utilizamos clases simples para los objetos de dominio.

En nuestro segundo enfoque, ampliamos la interfaz genérica *CrudRepository* y confiamos en las anotaciones específicas del módulo como *@Table* y *@Document* en nuestros objetos administrados.

Al final, utilizamos la detección basada en paquetes cuando configuramos la aplicación con la ayuda de *@EnableCassandraRepositories* y *@EnableMongoRepositories*.

Como siempre, el código completo está disponible en GitHub (<https://github.com/eugenp/tutorials/tree/master/spring-boot-data>).

Acabo de anunciar el nuevo curso *Learn Spring*, centrado en los fundamentos de Spring 5 y Spring Boot 2:

>> VER EL CURSO (/ls-course-end)



¿Estás aprendiendo a construir tu API
con Spring ?

Enter your email address

>> Obtenga el libro electrónico

Asia desde 395€*

p.ej. Delhi desde 395€*

* Precio final (Iv, tasas e impuestos incluidos).

Reserva ahora

Deja una respuesta




Start the discussion...

✉ Suscribir ▼

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy/\)](/privacy-policy/)

Ok

 **ezoic** (<https://www.ezoic.com/what-is-ezoic/>)

reportar este anun

CATEGORÍAS

PRIMAVERA ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/](https://www.baeldung.com/category/spring/))
DESCANSO ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/](https://www.baeldung.com/category/rest/))
JAVA ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/](https://www.baeldung.com/category/java/))
SEGURIDAD ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/](https://www.baeldung.com/category/security-2/))
PERSISTENCIA ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/](https://www.baeldung.com/category/persistence/))
JACKSON ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/](https://www.baeldung.com/category/json/jackson/))
HTTP DEL LADO DEL CLIENTE ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/](https://www.baeldung.com/category/http/))
KOTLIN ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/](https://www.baeldung.com/category/kotlin/))

SERIE

TUTORIAL DE JAVA "VOLVER A LO BÁSICO" ([/JAVA-TUTORIAL](#))
JACKSON JSON TUTORIAL ([/JACKSON](#))
HTTPCLIENT 4 TUTORIAL ([/HTTPCLIENT-GUIDE](#))
RESTO CON SPRING TUTORIAL ([/REST-WITH-SPRING-SERIES](#))
TUTORIAL SPRING PERSISTENCE ([/PERSISTENCE-WITH-SPRING-SERIES](#))
SEGURIDAD CON PRIMAVERA ([/SECURITY-SPRING](#))

ACERCA DE

SOBRE BAELDUNG ([/ABOUT](#))
LOS CURSOS ([HTTPS://COURSES.BAELDUNG.COM](https://courses.baeldung.com))
TRABAJO DE CONSULTORÍA ([/CONSULTING](#))
META BAELDUNG ([HTTP://META.BAELDUNG.COM/](http://meta.baeldung.com/))
EL ARCHIVO COMPLETO ([/FULL_ARCHIVE](#))
ESCRIBIR PARA BAELDUNG ([/CONTRIBUTION-GUIDELINES](#))
EDITORES ([/EDITORS](#))
NUESTROS COMPAÑEROS ([/PARTNERS](#))

ANUNCIE EN BAELDUNG ([/ADVERTISE](#))

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#)

Ok

[TÉRMINOS DE SERVICIO \(/TERMS-OF-SERVICE\)](#)
[POLÍTICA DE PRIVACIDAD \(/PRIVACY-POLICY\)](#)
[INFORMACIÓN DE LA COMPAÑÍA \(/BAELDUNG-COMPANY-INFO\)](#)
[CONTACTO \(/CONTACT\)](#)