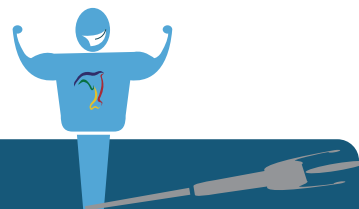


¿Qué ofrece Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

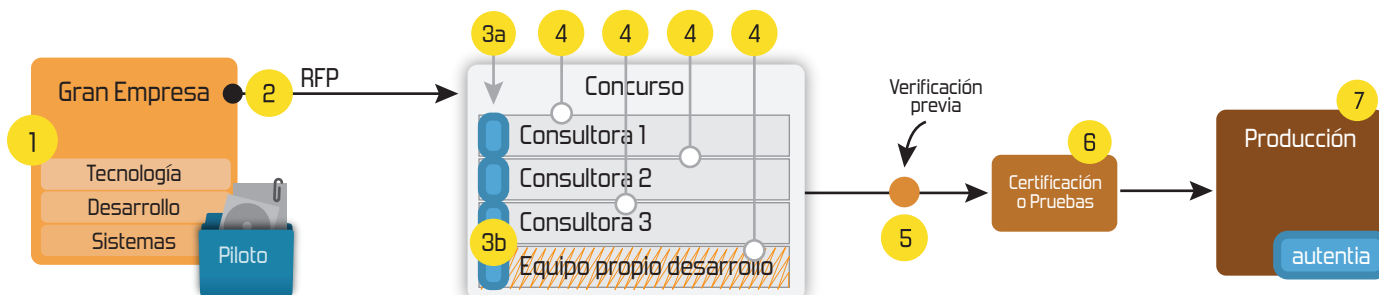
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces,
HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

Control de autenticación y
acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

» Estás en: Inicio » Tutoriales » Paradigma publish/subscribe con Spring Data Redis



Juan Alonso Ramos

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero en Informática, especialidad en Ingeniería del Software

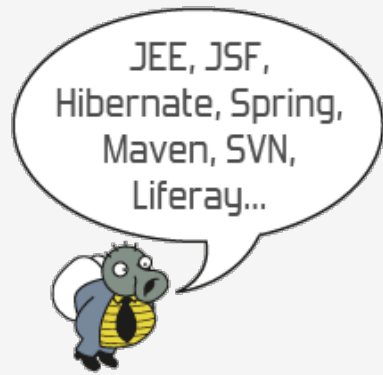
Puedes encontrarme en [Autentia](#): Ofrecemos de servicios soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE



[Ver todos los tutoriales del autor](#)

Catálogo de servicios Autentia



Síguenos a través de:



Últimas Noticias

» [Curso JBoss de Red Hat](#)

» Si eres el responsable o líder técnico, considérate desafortunado. No puedes culpar a nadie por ser gris

» [Portales, gestores de contenidos documentales y desarrollos a medida](#)

» [Comentando el libro Start-up Nation, La historia del milagro económico de Israel, de Dan Senor & Salu Singer](#)

» [Screencasts de programación narrados en Español](#)

[Histórico de noticias](#)

Últimos Tutoriales

» [Creación paso a paso de un webscript Alfresco](#)

» [Integración de MonkeyTalk en iOS](#)

» [Soporte de Redis con Spring: RedisTemplate](#)

Fecha de publicación del tutorial: 2014-11-14

Tutorial visitado 24 veces [Descargar en PDF](#)

Paradigma publish/subscribe con Spring Data Redis.

0. Índice de contenidos.

- 1. Introducción.
- 2. Entorno.
- 3. Publicar en Redis.
- 4. Suscripción a un canal de Redis.
- 5. Referencias.
- 6. Conclusiones.

1. Introducción.

Ya vimos en un [tutorial](#) anterior una pequeña introducción al proyecto Spring Data y el soporte de Redis a través de la clase **RedisTemplate**. En aquel tutorial vimos un ejemplo práctico donde recuperábamos tweets a través del API de Twitter4j y los persistíamos en Redis añadiendo una clave común a todos por la que posteriormente los recuperábamos.

Ese código se quedaba un poco cojo y no nos valía para cubrir un caso de uso más complejo. El problema tenía que ver con la sincronización de los datos almacenados. La aplicación levantaba dos beans de Spring, uno que guardaba los tweets en Redis y otro que los recuperaba. El segundo realizaba un único acceso a Redis recuperando todos los tweets que cumplían con los criterios de búsqueda, pero ¿qué pasaba con los nuevos tweets introducidos posteriores a esa búsqueda?. Esos datos no los recuperábamos a menos que hiciéramos sucesivas consultas a Redis. Al buscar mediante un patrón que tenían todas las claves (tweet_) cada vez que hacemos una nueva búsqueda Redis nos devuelve todos los registros por lo que estaríamos repitiendo constantemente los tweets recuperados en sucesivas consultas.

Para solucionar este problema vamos a modificar la forma de introducir y recuperar los tweets en Redis ya que esta base de datos admite el paradigma de mensajería [publicador/suscriptor](#). En nuestro ejemplo la clase encargada de recuperar los tweets los publicará en redis y construiremos una clase con Spring que se suscribirá para recibirlos inmediatamente en cuanto son publicados.

2. Entorno.

El tutorial se ha realizado con el siguiente entorno:

- MacBook Pro 15' (2.4 GHz Intel Core i5, 8GB DDR3 SDRAM).
- Sistema Operativo: Mac OS Mavericks 10.9.5
- Oracle Java SDK 1.7.0_60
- Redis 2.8.11

3. Publicar en Redis

A través de nuestra clase Repository disponemos del método **send** que publica el contenido en un determinado canal para

que suscriptores al mismo lo riciban. Vamos a modificar el ejemplo que utilizamos en el tutorial de [Redis](#) para publicar los tweets. Simplemente modificaremos la forma de almacenar los tweets en redis (línea 53) utilizando el método send que se encarga de publicarlo en el canal 'tweets'.

```
1 package com.autentia.tutoriales.tweets;
2
3 import javax.annotation.PostConstruct;
4
5 import org.slf4j.Logger;
6 import org.slf4j.LoggerFactory;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.stereotype.Component;
9
10 import twitter4j.Query;
11 import twitter4j.QueryResult;
12 import twitter4j.Status;
13 import twitter4j.Twitter;
14 import twitter4j.TwitterException;
15 import twitter4j.TwitterFactory;
16 import twitter4j.TwitterObjectFactory;
17
18 import com.autentia.tutoriales.redis.StringRedisRepository;
19
20 @Component
21 public class TweetIngestor {
22
23     private static final Logger logger = LoggerFactory.getLogger(TweetIngestor.class)
24
25     private final Twitter twitter;
26
27     public static final String TWEETS_SUBSCRIPTION_CHANNEL = "tweets";
28
29     @Autowired
30     private StringRedisRepository redisRepository;
31
32     public TweetIngestor() {
33         this.twitter = new TwitterFactory().getInstance();
34     }
35
36     @PostConstruct
37     public void searchByHashtag() {
38         new Thread() {
39             @Override
40             public void run() {
41                 try {
42                     Query query = new Query("redis");
43
44                     int numTweets = 0;
45                     long init = System.currentTimeMillis();
46                     try {
47                         QueryResult result;
48
49                         do {
50                             result = twitter.search(query);
51                             for (Status status : result.getTweets()) {
52                                 if (!status.isRetweet()) {
53                                     redisRepository.send(TWEETS_SUBSCRIPTION_CHANNEL,
54                                                             numTweets++);
55                                 }
56                             }
57
58                             } while ((query = result.nextQuery()) != null);
59
60                             logger.info(String.format("%s tweets received in %s millis",
61                                                         } catch (TwitterException e) {
62                                     throw new RuntimeException("Something was wrong retrieving tw
63                                 }
64
65                             } catch (Exception e) {
66                                 logger.error("Error", e);
67                             }
68                         }
69                     }.start();
70                 }
71             }
```

El método **send** del StringRedisRepository llama al método **convertAndSend** del **StringRedisTemplate**

```
1 public void send(String key, String value) {
2     template.convertAndSend(key, value);
3 }
```

4. Suscripción a un canal de Redis.

Con Spring Data Redis es muy sencillo configurar la suscripción a un canal de Redis, bastará con crear una clase suscriptora de los mensajes asociados a un topic y un poco de configuración. Lo haremos todo en nuestra clase **Application** encargada de levantar el contexto de Spring y de configurar los Beans que necesitamos.

Configuramos un **RedisMessageListenerContainer** al que añadimos mediante el método **addMessageListener** un **MessageListenerAdapter** que recibe una clase suscriptora de los mensajes.

Esta clase debemos implementarla y crear el método **receiveTweet** que será el que se invocará cuando se publique un mensaje en Redis que cumpla el patrón indicado en la definición del listener: **new PatternTopic(TweetIngestor.TWEETS_SUBSCRIPTION_CHANNEL)**, concretamente el topic es el mismo con el que se publicaron los tweets.

```
1 package com.autentia.tutoriales;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
```

» Embeber vídeo en MailChimp

» Tutorial VIPER en Swift

Últimos Tutoriales del Autor

» Soporte de Redis con Spring: RedisTemplate

» Monitorización de Apache Kafka

» Monta fácilmente tu proyecto con Spring Boot Starter POMs

» Primeros pasos con Apache Kafka

» Trident, un compañero de viaje para tratar con Storm

Categorías del Tutorial

Spring

BBDD


```
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.ComponentScan;
7 import org.springframework.context.annotation.Configuration;
8 import org.springframework.data.redis.connection.RedisConnectionFactory;
9 import org.springframework.data.redis.core.StringRedisTemplate;
10 import org.springframework.data.redis.listener.PatternTopic;
11 import org.springframework.data.redis.listener.RedisMessageListenerContainer;
12 import org.springframework.data.redis.listener.adapter.MessageListenerAdapter;
13
14 import com.autentia.tutoriales.redis.StringRedisRepository;
15 import com.autentia.tutoriales.tweets.TweetIngestor;
16 import com.autentia.tutoriales.tweets.TweetsReceiver;
17
18 @Configuration
19 @EnableAutoConfiguration
20 @ComponentScan
21 public class Application {
22
23     @Bean
24     RedisMessageListenerContainer container(RedisConnectionFactory connectionFactory,
25         final RedisMessageListenerContainer container = new RedisMessageListenerConta
26         container.setConnectionFactory(connectionFactory);
27         container.addMessageListener(listenerAdapter, new PatternTopic(TweetIngestor.
28
29         return container;
30     }
31
32     @Bean
33     MessageListenerAdapter messageListenerAdapter(TweetsReceiver tweetsReceiver) {
34         return new MessageListenerAdapter(tweetsReceiver, "receiveTweet");
35     }
36
37     @Bean
38     TweetsReceiver receiver() {
39         return new TweetsReceiver();
40     }
41
42     @Bean
43     StringRedisTemplate template(RedisConnectionFactory connectionFactory) {
44         return new StringRedisTemplate(connectionFactory);
45     }
46
47     @Bean
48     StringRedisRepository stringRedisRepository(StringRedisTemplate template) {
49         return new StringRedisRepository(template);
50     }
51
52     public static void main(String[] args) throws InterruptedException {
53         SpringApplication.run(Application.class, args);
54     }
55 }
```

La clase suscriptora define el método receiveTweets en formato JSON que es como los publicamos en Redis. Posteriormente se parsea y únicamente recuperamos el usuario y el texto del tweet que imprimimos por consola.

```
1 package com.autentia.tutoriales.tweets;
2
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5
6 import twitter4j.Status;
7 import twitter4j.TwitterException;
8 import twitter4j.TwitterObjectFactory;
9
10 public class TweetsReceiver {
11     private static final Logger LOGGER = LoggerFactory.getLogger(TweetsReceiver.class)
12
13     public void receiveTweet(String rawTweet) {
14         final Status tweet = parse(rawTweet);
15
16         LOGGER.info(String.format("@%s : %s", tweet.getUser().getName(), tweet.getTex
17     }
18
19     private Status parse(String rawJson) {
20         try {
21             return TwitterObjectFactory.createStatus(rawJson);
22         } catch (TwitterException e) {
23             LOGGER.warn("Invalid tweet" + rawJson, e);
24             return null;
25         }
26     }
27 }
```

Para probar nuestro código levantamos el servidor de redis (**redis-server**) y el ejemplo con **mvn spring-boot:run** y pasados unos segundos aparecerán por consola los tweets que previamente han pasado por Redis. Si queremos suscribirnos al topic desde el cliente de Redis basta con lanzar el comando desde la consola del **redis-cli**:

```
1 | PSUBSCRIBE tweets
```

5. Referencias

- Comandos de Redis
- Código fuente del tutorial

6. Conclusiones.

Hemos visto un ejemplo de uso de la funcionalidad para mensajería publish/subscribe de Redis una más de las muchas que soporta.

A través de las APIs que nos proporcionan un Driver de conexión (como Jedis) ya se nos facilita mucho la tarea para trabajar con Redis pero junto al soporte de Spring Data el uso de Redis parece un juego de niños.

Espero que te haya sido de ayuda.

Un saludo.

Juan

A continuación puedes evaluarlo:

[Regístrate para evaluarlo](#)



Por favor, vota +1 o compártelo si te pareció interesante

+

Share

|

f

t

in

e

★

g

g+1

0

g+1

0

Anímate y coméntanos lo que pienses sobre este **TUTORIAL**:

» **Regístrate** y accede a esta y otras ventajas «



Esta obra está licenciada bajo [licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)

PUSH THIS

Page Pushers

Community

Help?

no clicks

+

+

+

+

+

+

+

+

0 people

brought clicks to this page

powered by [karmacracy](#)