(/)

Mensajería con Spring AMQP

Última modificación: 12 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/) +

Mensajería (https://www.baeldung.com/tag/messaging/)

1. Información general

En este tutorial, exploraremos la comunicación basada en mensajes a través de AMQP utilizando el marco Spring AMQP. Primero, cubriremos algunos de los conceptos clave de la mensajería. Luego, pasaremos a un ejemplo práctico.

2. Comunicación basada en mensajes

La mensajería es una técnica para comunicarse entre aplicaciones. Se basa en el paso de mensajes asíncrono en lugar de la arquitectura basada en la respuesta de solicitud síncrona. **Los productores y consumidores de mensajes están desacoplados por una capa de mensajería intermedia conocida como intermediario de mensajes** . Un intermediario de mensajes proporciona características como el almacenamiento persistente de mensajes, el filtrado de mensajes y la transformación de mensajes.

En un caso c usa
comúnment **s utilizar**
clientes y c

3. AMQP - Protocolo avanzado de colas de mensajes

AMQP es una especificación de cable estándar abierto para la comunicación de mensajes asíncronos. Proporciona una descripción de cómo se debe construir un mensaje.

3.1. Cómo Amqp es diferente de Jms

Dado que AMQP es un estándar de protocolo binario neutral para la plataforma, las bibliotecas se pueden escribir en diferentes lenguajes de programación y ejecutarse en diferentes entornos.

No existe un bloqueo de protocolo basado en el proveedor, como es el caso cuando se migra de un agente JMS a otro. Para obtener más detalles, consulte JMS vs AMQP (https://www.linkedin.com/pulse/jms-vs-amqp-eran-shaham) y Comprensión de AMQP (https://docs.spring.io/spring-amqp/reference/html/) . Algunos de los corredores AMQP ampliamente utilizados son RabbitMQ , OpenAMQ y StormMQ.

3.2. Entidades AMQP

Brevemente, AMQP se compone de intercambios, colas y enlaces:

- *Los intercambios* son como oficinas de correos o buzones y los clientes publican un mensaje en un intercambio AMQP. Hay cuatro tipos de intercambio integrados.
  - Intercambio directo: enruta los mensajes a una cola haciendo coincidir una clave de enrutamiento completa
  - Fanout Exchange: enruta mensajes a todas las colas vinculadas
  - Intercambio de temas: enruta mensajes a varias colas haciendo coincidir una clave de enrutamiento con un patrón
  - Intercambio de encabezados: enruta mensajes basados en encabezados de mensajes
- *Las colas* están vinculadas a un intercambio utilizando una clave de enrutamiento
- *Los mensajes* se envían a un intercambio con una clave de enrutamiento. El intercambio luego distribuye copias de los mensajes a las colas.

Para obtener más detalles, eche un vistazo a Conceptos AMQP (https://www.rabbitmq.com/tutorials/amqp-concepts.html) y topologías de enrutamiento. (https://spring.io/blog/2011/04/01/routing-topologies-for-performance-and-scalability-with-rabbitmq/)

3.3. Spring AMQP

Spring AMQP consta de dos módulos: *spring-amqp* y *spring-rabbit* . Juntos, estos módulos proporcionan abstracciones para:

- Entidades AMQP: creamos entidades con las clases *Mensaje, Cola, Enlace e Intercambio*
- Gestión
  Caching
- Publica
- Consumo de mensajes: utilizamos un *@RabbitListener* para leer mensajes de una cola

4. Configurar un corredor de Rabbitmq

Necesitamos un corredor RabbitMQ disponible para que podamos conectarnos. La forma más sencilla de hacer esto es usando Docker para buscar y ejecutar una imagen RabbitMQ para nosotros:

```
docker run -d -p 5672:5672 -p 15672:15672 --name my-rabbit rabbitmq:3-management
```

Exponemos el puerto 5672 para que nuestra aplicación pueda conectarse a RabbitMQ.

Y exponemos el puerto 15672 para que podamos ver qué está haciendo nuestro agente RabbitMQ a través de la IU de administración: *http: // localhost: 15672 (http://localhost:15672)* o la API HTTP: *http: // localhost: 15672 / api / index.html (http://localhost:15672/api/index.html)* .

5. Crear nuestra aplicación Spring Amqp

Entonces, ahora creemos nuestra aplicación para enviar y recibir un simple mensaje "¡Hola, mundo!" Usando Spring AMQP.

5.1. Dependencias de Maven

Para agregar los módulos *spring-amqp* y *spring-rabbit* a nuestro proyecto, agregamos la dependencia *spring-boot-starter-amqp* a nuestro *pom.xml* :

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-amqp</artifactId>
        <version>2.1.6.RELEASE</version>
    </dependency>
</dependencies>
```

Podemos encontrar la última versión en Maven Central (https://search.maven.org/classic/#search%7Cga%7C1%7C(g%3A%22org.springframework.boot%22%20AND%20a%3A%22spring-boot-starter-amqp%22)) .

5.2. Conectando con nuestro corredor Rabbitmq

Vamos a utilizar la configuración automática de la primavera de arranque para crear nuestros *ConnectionFactory*, *RabbitTemplate* y *RabbitAdmin* granos. Como resultado, obtenemos una conexión con nuestro corredor RabbitMQ en el puerto 5672 utilizando el nombre de usuario y la contraseña predeterminados de "invitado". Entonces, simplemente anotamos nuestra aplicación con *@SpringBootApplication*:

```
@SpringBootApplication
public class HelloWorldMessageApp {
   // ...
}
```

5.3. Crea nuestra cola

Para crear nuestra cola, simplemente definimos un bean de tipo *cola*. *RabbitAdmin* encontrará esto y lo vinculará al intercambio predeterminado con una clave de enrutamiento de "myQueue":

```
@Bean
public
    return new Queue("myQueue", false);
}
```

Configuramos la cola para que no sea duradera, de modo que la cola y cualquier mensaje en ella se eliminen cuando se detenga RabbitMQ. Sin embargo, tenga en cuenta que reiniciar nuestra aplicación no tendrá ningún efecto en la cola.

5.4. Envía nuestro mensaje

Vamos a **utilizar el *RabbitTemplate* para enviar** nuestro mensaje "Hola, mundo!":

```
rabbitTemplate.convertAndSend("myQueue", "Hello, world!");
```

5.5. Consume nuestro mensaje

Implementaremos un mensaje de consumidor anotando un método con *@RabbitListener*:

```
@RabbitListener(queues = "myQueue")
public void listen(String in) {
    System.out.println("Message read from myQueue : " + in);
}
```

6. Ejecutando nuestra aplicación

Primero, comenzamos el corredor RabbitMQ:

```
docker run -d -p 5672:5672 -p 15672:15672 --name my-rabbit rabbitmq:3-management
```

Luego, ejecutamos la aplicación Spring Boot ejecutando *HelloWorldMessage.java* , ejecutando el método *main ()* :

```
mvn spring-boot:run -Dstart-class=com.baeldung.springamqp.simple.HelloWorldMessageApp
```

Mientras se ejecuta la aplicación, veremos que:

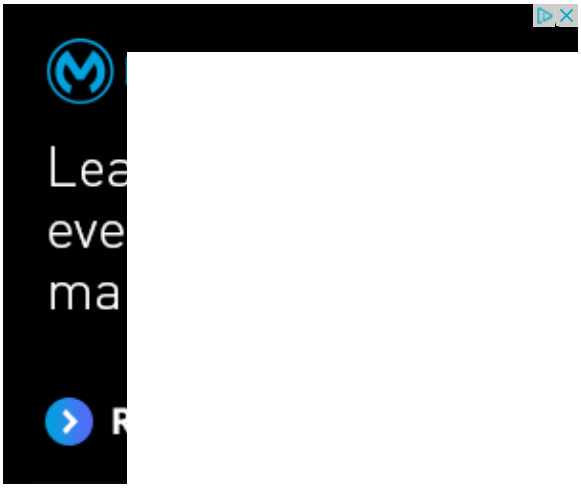
- La aplicación envía un mensaje al intercambio predeterminado con "myQueue" como clave de enrutamiento
- Luego, la cola "myQueue" recibe el mensaje
- Finalmente, el método *listen* consume el mensaje de "myQueue" y lo imprime en la consola

También podemos usar la página de administración de RabbitMQ en *http: // localhost: 15672 (http://localhost:15672)* para ver que nuestro mensaje ha sido enviado y consumido.

7. Conclusión

En este tutorial, cubrimos la arquitectura basada en mensajería sobre el protocolo AMQP utilizando Spring AMQP para la comunicación entre aplicaciones.

El código fuente completo y todos los fragmentos de código para este tutorial están disponibles en el proyecto GitHub (https://github.com/eugenp/tutorials/tree/master/spring-amqp) .



Amit Patel

Invitado

If I want synchronous communication then how to use sendAndRecieve() method ,I want to send message to queue and that message is consumed by consumer and consumer should return the response to caller point ,, can you provide code example .

+ 0 −

2 years ago

Grzegorz Piwowarek

Guest

Amit, everything would stay the same just like in this example. Only the dispatch method changes to sendAndReceive() which returns a response and not a void.

+ 0 −

2 years ago

Comments are closed on this article!

CATEGORIES

SPRING (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
HTTP CLIENT-SIDE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)