

Springboot整合rabbitmq集群配置详解

 飞哥 VIP

分类: 学习笔记

创建时间: 2021/03/06 21:59

☒ 字体

☐ 皮肤

最后修改于: 2021/03/06 22:07

pringboot整合rabbitmq
集群创建方式这里省略
整合开始
1 引入starter

1. <parent>

2. <groupId>org.springframework.boot</groupId>

3. <artifactId>spring-boot-starter-parent</artifactId>

4. <version>2.2.6.RELEASE</version>

5. <relativePath/> <!-- lookup parent from repository -->

6. </parent>

7. <dependency>

8. <groupId>org.springframework.boot</groupId>

9. <artifactId>spring-boot-starter-amqp</artifactId>

10. </dependency>



2: 详细配置如下



1. rabbitmq:2. addresses: 127.0.0.1:6605,127.0.0.1:6606,127.0.0.1:6705 #指定client连接到的server的地址，多个以逗号分隔(优先取addresses，然后再取host)3. # port:4. ##集群配置 addresses之间用逗号隔开5. # addresses: ip:port,ip:port6. password: admin7. username: 1234568. virtual-host: / # 连接到rabbitMQ的vhost9. requested-heartbeat: #指定心跳超时，单位秒，0为不指定；默认60s10. publisher-confirms: #是否启用 发布确认11. publisher-reurns: # 是否启用发布返回12. connection-timeout: #连接超时，单位毫秒，0表示无穷大，不超时13. cache:14. channel.size: # 缓存中保持的channel数量15. channel.checkout-timeout: # 当缓存数量被设置时，从缓存中获取一个channel的超时时间，单位毫秒；如果为0，则总是创建一个新channel16. connection.size: # 缓存的连接数，只有是CONNECTION模式时生效17. connection.mode: # 连接工厂缓存模式：CHANNEL 和 CONNECTION18. listener:19. simple.auto-startup: # 是否启动时自动启动容器20. simple.acknowledge-mode: # 表示消息确认方式，其有三种配置方式，分别是none、manual和auto；默认auto21. simple.concurrency: # 最小的消费者数量22. simple.max-concurrency: # 最大的消费者数量23. simple.prefetch: # 指定一个请求能处理多少个消息，如果有事务的话，必须大于等于transaction数量。24. simple.transaction-size: # 指定一个事务处理的消息数量，最好是小于等于prefetch的数量。25. simple.default-requeue-rejected: # 决定被拒绝的消息是否重新入队；默认是true（与参数acknowledge-mode有关系）26. simple.idle-event-interval: # 多少长时间发布空闲容器时间，单位毫秒27. simple.retry.enabled: # 是否可用28. simple.retry.max-attempts: # 最大重试次数29. simple.retry.initial-interval: # 第一次和第二次尝试发布或传递消息之间的间隔30. simple.retry.multiplier: # 应用于上一重试间隔的乘数31. simple.retry.max-interval: # 最大重试时间间隔32. simple.retry.stateless: # 重试是有状态or无状态33. template:34. mandatory: # 启用强制信息；默认false35. receive-timeout: # receive() 操作的超时时间36. reply-timeout: # sendAndReceive() 操作的超时时间37. retry.enabled: # 发送重试是否可用38. retry.max-attempts: # 最大重试次数39. retry.initial-interval: # 第一次和第二次尝试发布或传递消息之间的间隔40. retry.multiplier: # 应用于上一重试间隔的乘数41. retry.max-interval: #最大重试时间间隔

注：相关配置很多，大家只需要关注一些常用的配置即可

对于发送方而言，需要做以下配置：

- 1 配置CachingConnectionFactory
- 2 配置Exchange/Queue/Binding
- 3 配置RabbitAdmin创建上一步的Exchange/Queue/Binding
- 4 配置RabbitTemplate用于发送消息，RabbitTemplate通过CachingConnectionFactory获取到Connection，然后想指定Exchange发送

对于消费方而言，需要做以下配置：

- 1 配置CachingConnectionFactory
- 2 配置Exchange/Queue/Binding
- 3 配置RabbitAdmin创建上一步的Exchange/Queue/Binding
- 4 配置RabbitListenerContainerFactory
- 5 配置@RabbitListener/@RabbitHandler用于接收消息

在默认情况下主要的配置如下：

配置项	默认值	作用
host	localhost	RabbitMQ服务器地址
port	5672	RabbitMQ服务器端口
username	账户名	guest
password	密码	guest
virtualHost	RabbitMQ 虚拟主机 名	/
publisherConfirms	false	设置是否启用生产方确认
publisherReturns	false	设置是否启用生产方消息返回
ssl	对象	配置SSL，默认停用
template	对象	设置RabbitTemplate
template.retry	默认停用	设置RabbitTemplate发送消息时的重试，主要用于RabbitTemplate与RabbitMQ之间的网络连接
template.mandatory	false	设置发送消息失败时（无接收queue）是否return 消息，与return callback一并使用
template.exchange	""	默认发送的exchange
template.routingKey	""	默认发送消息时的routing key
template.defaultReceiveQueue	null	默认接收消息的queue
listener.simple	对象	设置SimpleRabbitListenerContainerFactory
listener.direct	对象	设置DirectRabbitListenerContainerFactory
listener.simple.concurrency	null	并发消费方数量
listener.simple.acknowledgeMode	AUTO	设置消费方确认模式，这里的AUTO与RabbitMQ的自动确认不是一回事
listener.simple.prefetch	250	设置消费方一次性接收消息的条数
listener.simple.defaultRequeueRejected	true	当Listener发生异常时是否requeue
listener.simple.retry	对象	设置Listener的重试机制，默认停用，当启用时，Listener对于消息处理过程中的异常将进行requeue重试，超过重试次数再抛弃，此时AmqpRejectAndDontRequeueException异常也会被重试

3 Spring AMQP的主要对象
注：如果不了解AMQP请前往官网了解.


类	作用
Queue	对应RabbitMQ中Queue
AmqpTemplate	接口，用于向RabbitMQ发送和接收Message
RabbitTemplate	AmqpTemplate的实现类
@RabbitListener	指定消息接收方，可以配置在类和方法上
@RabbitHandler	指定消息接收方，只能配置在方法上，可以与@RabbitListener一起使用
Message	对RabbitMQ消息的封装
Exchange	对RabbitMQ的Exchange的封装，子类有TopicExchange、FanoutExchange和DirectExchange等
Binding	将一个Queue绑定到某个Exchange，本身只是一个声明，并不做实际绑定操作
AmqpAdmin	接口，用于Exchange和Queue的管理，比如创建/删除/绑定等，自动检查Binding类并完成绑定操作
RabbitAdmin	AmqpAdmin的实现类
ConnectionFactory	创建Connection的工厂类，RabbitMQ也有一个名为ConnectionFactory的类但二者没有继承关系, Spring ConnectionFactory可以认为是对RabbitMQ ConnectionFactory的封装
CachingConnectionFactory	Spring ConnectionFactory的实现类，可以用于缓存Channel和Connection
Connection	Spring中用于创建Channel的连接类，RabbitMQ也有一个名为Connection的类，但二者没有继承关系， Spring Connection是对RabbitMQ Connection的封装
SimpleConnection	Spring Connection的实现类，将实际工作代理给RabbitMQ的Connection类
MessageListenerContainer	接口，消费端负责与RabbitMQ服务器保持连接并将Message传递给实际的@RabbitListener/@RabbitHandler处理
RabbitListenerContainerFactory	接口，用于创建MessageListenerContainer
SimpleMessageListenerContainer	MessageListenerContainer的实现类
SimpleRabbitListenerContainerFactory	RabbitListenerContainerFactory的实现类
RabbitProperties	用于配置Spring AMQP的Property类

4 使用：
通过配置类加载的方式：





```

1. package com.yd.demo.config;
2. import org.slf4j.Logger;
3. import org.slf4j.LoggerFactory;
4. import org.springframework.amqp.core.AcknowledgeMode;
5. import org.springframework.amqp.core.Binding;
6. import org.springframework.amqp.core.BindingBuilder;
7. import org.springframework.amqp.core.DirectExchange;
8. import org.springframework.amqp.core.Queue;
9. import org.springframework.amqp.core.TopicExchange;
10. import org.springframework.amqp.rabbit.config.SimpleRabbitListenerContainerFactory;
11. import org.springframework.amqp.rabbit.connection.CachingConnectionFactory;
12. import org.springframework.amqp.rabbit.connection.ConnectionFactory;
13. import org.springframework.amqp.rabbit.core.RabbitAdmin;
14. import org.springframework.amqp.rabbit.core.RabbitTemplate;
15. import org.springframework.amqp.rabbit.listener.RabbitListenerContainerFactory;
16. import org.springframework.amqp.support.converter.Jackson2JsonMessageConverter;
17. import org.springframework.beans.factory.annotation.Autowired;
18. import org.springframework.beans.factory.annotation.Value;
19. import org.springframework.context.annotation.Bean;
20. import org.springframework.context.annotation.Configuration;
21.
22. import java.util.HashMap;
23. import java.util.Map;
24.
25. @Configuration
26. public class RabbitConfig {
27.     private static final Logger logger = LoggerFactory.getLogger(RabbitConfig.class);
28.
29.     public static final String PUBLISHINGEXCHANGE="spring-ex";
30.     public static final String  EDLXQUEUE="spring-qu1";
31.
32.
33.     public static final String RECEIVEDLXROUTINGKEY="aa";
34.
35.     public static final String DIRECTEXCHANGE="spring-ex";
36.
37.     public static final String MDMQUEUE="mdmQueue";
38.
39.     public static final String TOPICEXCHANGE="spring-top";
40.
41.     @Value("${spring.rabbitmq.addresses}")
42.     private String hosts;
43.
44.     @Value("${spring.rabbitmq.username}")
45.     private String userName;
46.
47.     @Value("${spring.rabbitmq.password}")
48.     private String password;
49.
50.     @Value("${spring.rabbitmq.virtual-host}")
51.     private String virtualHost;
52.
53.     /* @Value("${rabbit.channelCacheSize}")
54.     private int channelCacheSize;*/
55.
56.     // @Value("${rabbit.port}")
57.     // private int port;
58.
59.
60.     /* @Autowired
61.     private ConfirmCallbackListener confirmCallbackListener;
62.
63.     @Autowired
64.     private ReturnCallbackListener returnCallbackListener;*/
65.     @Bean
66.     public ConnectionFactory connectionFactory(){
67.         CachingConnectionFactory cachingConnectionFactory = new CachingConnectionFactory();
68.         cachingConnectionFactory.setAddresses(hosts);

```




```

69.         cachingConnectionFactory.setUsername(userName);
70.         cachingConnectionFactory.setPassword(password);
71.         //         cachingConnectionFactory.setChannelCacheSize(channelCacheSize);
72.         //cachingConnectionFactory.setPort(port);
73.         cachingConnectionFactory.setVirtualHost(virtualHost);
74.         //设置连接工厂缓存模式：
75.         cachingConnectionFactory.setCacheMode(CachingConnectionFactory.CacheMode.CONNECTION);
76.         //缓存连接数
77.         cachingConnectionFactory.setConnectionCacheSize(3);
78.         //设置连接限制
79.         cachingConnectionFactory.setConnectionLimit(6);
80.         logger.info("连接工厂设置完成，连接地址{}"+hosts);
81.         logger.info("连接工厂设置完成，连接用户{}"+userName);
82.         return cachingConnectionFactory;
83.     }
84.
85.     @Bean
86.     public RabbitAdmin rabbitAdmin(){
87.         RabbitAdmin rabbitAdmin = new RabbitAdmin(connectionFactory());
88.         rabbitAdmin.setAutoStartup(true);
89.         rabbitAdmin.setIgnoreDeclarationExceptions(true);
90.         rabbitAdmin.declareBinding(bindingMdmQueue());
91.         //声明topic交换器
92.         rabbitAdmin.declareExchange(directExchange());
93.         logger.info("管理员设置完成");
94.         return rabbitAdmin;
95.     }
96.
97.
98.     @Bean
99.     public RabbitListenerContainerFactory listenerContainerFactory() {
100.         SimpleRabbitListenerContainerFactory factory = new SimpleRabbitListenerContainerFactory();
101.         factory.setConnectionFactory(connectionFactory());
102.         factory.setMessageConverter(new Jackson2JsonMessageConverter());
103.         //最小消费者数量
104.         factory.setConcurrentConsumers(10);
105.         //最大消费者数量
106.         factory.setMaxConcurrentConsumers(10);
107.         //一个请求最大处理的消息数量
108.         factory.setPrefetchCount(10);
109.         //
110.         factory.setChannelTransacted(true);
111.         //默认不排队
112.         factory.setDefaultRequeueRejected(true);
113.         //手动确认接收到了消息
114.         factory.setAcknowledgeMode(AcknowledgeMode.MANUAL);
115.         logger.info("监听者设置完成");
116.         return factory;
117.     }
118.
119.     @Bean
120.     public DirectExchange directExchange(){
121.         return new DirectExchange(DIRECTEXCHANGE,true,false);
122.     }
123.
124.     @Bean
125.     public Queue mdmQueue(){
126.         Map arguments = new HashMap<>();
127.         // 绑定该队列到私信交换机
128.         arguments.put("x-dead-letter-exchange",RECEIVEDLXEXCHANGE);
129.         arguments.put("x-dead-letter-routing-key",RECEIVEDLXROUTINGKEY);
130.         logger.info("队列交换机绑定完成");
131.         return new Queue(RECEIVEDLXQUEUE,true,false,false,arguments);
132.     }
133.
134.     @Bean
135.     Binding bindingMdmQueue() {
136.         return BindingBuilder.bind(mdmQueue()).to(directExchange()).with("");

```

```
137.     }
138.
139.     @Bean
140.     public RabbitTemplate rabbitTemplate(){
141.         RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory());
142.         rabbitTemplate.setMandatory(true);
143.         //发布确认
144.         //      rabbitTemplate.setConfirmCallback(confirmCallbackListener);
145.         // 启用发布返回
146.         //      rabbitTemplate.setReturnCallback(returnCallbackListener);
147.         logger.info("连接模板设置完成");
148.         return rabbitTemplate;
149.     }
150.
151.     /* @Bean
152.     public TopicExchange topicExchange(){
153.         return new TopicExchange(TOPICEXCHANGE,true,false);
154.     }*/
155.
156.     /*
157.
158.
159.     */
160.     * @return DirectExchange
161.     */
162.
163.     @Bean
164.     public DirectExchange dlxExchange() {
165.         return new DirectExchange(RECEIVEDLXEXCHANGE,true,false);
166.     }
167.
168.     */
169.     *
170.     * @return Queue
171.     */
172.     @Bean
173.     public Queue dlxQueue() {
174.         return new Queue(RECEIVEDLXQUEUE,true);
175.     }
176.     */
177.     * @return Binding
178.     */
179.     @Bean
180.     public Binding binding() {
181.         return BindingBuilder.bind(dlxQueue()).to(dlxExchange()).with(RECEIVEDLXROUTINGKEY);
182.     }*/
183. }
```

通过两种方式加载

- 1 通过配置文件
- 2 通过配置类

说明：上面是通过配置文件与配置类的方式去加载,常用的配置如上所示。实际使用中要生产方与消费方要分开配置，相关配置也会有小变动，大体配置不变。更多信息可查看官网配置。

