



链滴

springboot 2.1.6 + rabbitmq 整合之道

作者: [zsm110110](#)

原文链接: <https://ld246.com/article/1571655032304>

来源网站: 链滴

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

1. 导入rabbitmq

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

2. 在application.yml中添加rabbitmq配置

```
spring:
  rabbitmq:
    host: localhost      # rabbitmq的IP地址或对应的域名
    username: admin     # rabbitmq登录账号
    password: 123456    # rabbitmq 登录密码
    virtual-host: /
    publisher-returns: true
    connection-timeout: 15000
    template:
      mandatory: true
    listener:
      simple:
        concurrency: 5 #监听最小为5
        max-concurrency: 10 #监听最大为10
```

3.注入队列

```
@Configuration
public class RabbitConfig {
    @Bean
    public Queue Queue() {
        return new Queue("zsm-rabbitmq");
    }
}
```

注意：zsm-rabbitmq为队列名称

4.创建消息生产者

```
@Component
public class RabbitProducer {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    @Scheduled(fixedDelay = 10000L)
    public void send() {
        rabbitTemplate.convertAndSend("zsm-rabbitmq", "Hello rabbitmq !~");
    }
}
```

```
}
```

注意：通过zsm-rabbitmq发送一个字符串也可以发送一个object,在生成环境中，绝大多数是一个object

5.创建消费者

```
@Component
public class RabbitConsumer {

    @RabbitHandler
    @RabbitListener(queues = "zsm-rabbitmq")
    public void process(@Payload String foo) {
        System.out.println(new Date() + ": " + foo);
    }
}
```

@RabbitListener(queues = "zsm-rabbitmq")这里里面要注意一下，这里要写你刚才在Queue里面那个队列名称

6.启动主类

1. 在启动类上面加入@EnableScheduling，表示启动定时任务
2. 修改Producer生产者类中的 @Scheduled(fixedDelay = 10000L)里面的数值
3. 控制台打印：Mon Oct 21 16:55:43 GMT+08:00 2019: Hello rabbitmq !~ 表示成功
4. 到此，一个简单的springboot + rabbitmq整合应用就弄好了；
5. 由于rabbitmq引用了Exchange概念，Exchange有四种类型：Direct、Topic、Headers、Fanout；其中Headers使用是最少的了，Direct是最简单的；
6. 下面我们来介绍一下topic和Fanout。

6.1 简单说一下Exchange中的四种类型

direct: exchange在和queue进行binding时会设置routingkey，将消息发送到exchange时会设置应的routingkey，只有这两个routingkey完全相同，exchange才会选择对应的binding进行消息路由。

fanout: 直接将消息路由到所有绑定的队列中，无须对消息的routingkey进行匹配操作。（广播）

topic: 此类型exchange和direct类型差不多，但direct类型要求routingkey完全相等，这里的routingkey可以有通配符：'*','#'。

其中'*'表示匹配一个单词，'#'则表示匹配没有或者多个单词。

header: 其路由的规则是根据header来判断，其中的header就是binding时的arguments参数：

7. Topic Exchange

7.1 配置Topic规则

```
import com.example.amqbrabbitmq.common.MQConst;
import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.Queue;
import org.springframework.amqp.core.TopicExchange;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

@Configuration

```
public class TopicRabbitConfig {
```

```
    @Bean
```

```
    public Queue queueMessage1() {
        return new Queue(MQConst.TOPIC_QUEUE_NAME1);
    }
```

```
    @Bean
```

```
    public Queue queueMessage2() {
        return new Queue(MQConst.TOPIC_QUEUE_NAME2);
    }
```

```
    @Bean
```

```
    TopicExchange exchange() {
        return new TopicExchange(MQConst.TOPIC_EXCHANGE);
    }
```

```
    @Bean
```

```
    Binding bindingExchangeMessage(Queue queueMessage1, TopicExchange exchange) {
        // 将队列1绑定到名为topicKey.A的routingKey
        return BindingBuilder.bind(queueMessage1).to(exchange).with(MQConst.TOPIC_KEY1);
    }
```

```
    @Bean
```

```
    Binding bindingExchangeMessages(Queue queueMessage2, TopicExchange exchange) {
        // 将队列2绑定到所有topicKey开头的routingKey
        return BindingBuilder.bind(queueMessage2).to(exchange).with(MQConst.TOPIC_KEYS);
    }
}
```

7.2 配置消费者

```
import com.example.amqbrabbitmq.common.MQConst;
import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;
```

@Component

```

public class TopicConsumer {

    @RabbitHandler
    @RabbitListener(queues = MQConst.TOPIC_QUEUE_NAME1)
    public void process1(Object message) {
        System.out.println("queue:topic.message1,message:" + message);
    }

    @RabbitHandler
    @RabbitListener(queues = MQConst.TOPIC_QUEUE_NAME2)
    public void process2(Object message) {
        System.out.println("queue:topic.message2,message:" + message);
    }
}

```

7.3 配置生产者

```

import com.example.amqbrabbitmq.common.MQConst;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.Scheduled;

@Configuration
public class TopicProducer {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    @Scheduled(fixedDelay = 10000L)
    public void send() {
        rabbitTemplate.convertAndSend(MQConst.TOPIC_EXCHANGE, MQConst.TOPIC_KEYS, "是TOPIC_KEYS");
        rabbitTemplate.convertAndSend(MQConst.TOPIC_EXCHANGE, MQConst.TOPIC_KEY1, "是TOPIC_KEY1");
    }
}

```

7.4 启动主类，控制台输出：

```

queue:topic.message2,message:(Body:'我是TOPIC_KEY1' MessageProperties [headers={}, contentType=text/plain, contentEncoding=UTF-8, contentLength=0, receivedDeliveryMode=PERSISTENT, priority=0, redelivered=false, receivedExchange=topic_exchange, receivedRoutingKey=opicKey.A, deliveryTag=1, consumerTag=amq.ctag-FFUIrKtXJAJI_ecGEo0NSA, consumerQueue=topic_queue_name2])
queue:topic.message2,message:(Body:'我是TOPIC_KEYS' MessageProperties [headers={}, contentType=text/plain, contentEncoding=UTF-8, contentLength=0, receivedDeliveryMode=PERSISTENT, priority=0, redelivered=false, receivedExchange=topic_exchange, receivedRoutingKey=opicKey.#, deliveryTag=1, consumerTag=amq.ctag-1VqDmcyXJi1_ZhD3GJcefA, consumerQueue=topic_queue_name2])
queue:topic.message1,message:(Body:'我是TOPIC_KEY1' MessageProperties [headers={}, cont

```

```
ntType=text/plain, contentType=UTF-8, contentLength=0, receivedDeliveryMode=PERSISTENT, priority=0, redelivered=false, receivedExchange=topic_exchange, receivedRoutingKey=opicKey.A, deliveryTag=1, consumerTag=amq.ctag-dDQ5i2_5YPcAYdCTHmjDxg, consumerQueue=topic_queueName1])
queue:topic.message1,message:(Body:'我是TOPIC_KEY1' MessageProperties [headers={}, contentType=text/plain, contentType=UTF-8, contentLength=0, receivedDeliveryMode=PERSISTENT, priority=0, redelivered=false, receivedExchange=topic_exchange, receivedRoutingKey=opicKey.A, deliveryTag=1, consumerTag=amq.ctag-fgs6LkKQo9x2dXw9EI6OEq, consumerQueue=topic_queueName1])
queue:topic.message2,message:(Body:'我是TOPIC_KEYS' MessageProperties [headers={}, contentType=text/plain, contentType=UTF-8, contentLength=0, receivedDeliveryMode=PERSISTENT, priority=0, redelivered=false, receivedExchange=topic_exchange, receivedRoutingKey=opicKey.#, deliveryTag=1, consumerTag=amq.ctag-rX6y-N4JGfg_FB1xzHsBiw, consumerQueue=topic_queueName2])
queue:topic.message2,message:(Body:'我是TOPIC_KEY1' MessageProperties [headers={}, contentType=text/plain, contentType=UTF-8, contentLength=0, receivedDeliveryMode=PERSISTENT, priority=0, redelivered=false, receivedExchange=topic_exchange, receivedRoutingKey=opicKey.A, deliveryTag=1, consumerTag=amq.ctag-cdPvOG38zLth8qBNxAaDug, consumerQueue=topic_queueName2])
```

8. Fanout Exchange

8.1 配置Fanout规则

```
import com.example.amqbrabbitmq.common.MQConst;
import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.FanoutExchange;
import org.springframework.amqp.core.Queue;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

@Configuration

```
public class FanoutRabbitConfig {
```

```
    @Bean
```

```
    public Queue MessageA() {
```

```
        return new Queue(MQConst.FANOUT_QUEUE_NAME1);
```

```
    }
```

```
    @Bean
```

```
    public Queue MessageB() {
```

```
        return new Queue(MQConst.FANOUT_QUEUE_NAME2);
```

```
    }
```

```
    @Bean
```

```
    FanoutExchange fanoutExchange() {
```

```
        return new FanoutExchange(MQConst.FANOUT_EXCHANGE);
```

```
    }
```

```
    @Bean
```

```
    Binding bindingExchangeA(Queue MessageA, FanoutExchange fanoutExchange) {
```

```
        return BindingBuilder.bind(MessageA).to(fanoutExchange);
```

```

    }

    @Bean
    Binding bindingExchangeB(Queue messageB, FanoutExchange fanoutExchange) {
        return BindingBuilder.bind(messageB).to(fanoutExchange);
    }
}

```

8.2 配置消费者

```

import com.example.amqbrabbitmq.common.MQConst;
import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
public class FanoutConsumer {

    @RabbitHandler
    @RabbitListener(queues = MQConst.FANOUT_QUEUE_NAME1)
    public void process1(String message){
        System.out.println("queue:fanout.message1,message:" + message);
    }

    @RabbitHandler
    @RabbitListener(queues = MQConst.FANOUT_QUEUE_NAME2)
    public void process2(String message){
        System.out.println("queue:fanout.message2,message:" + message);
    }
}

```

8.3 配置生产消息

```

import com.example.amqbrabbitmq.common.MQConst;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.Scheduled;

@Configuration
public class FanoutProducer {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    @Scheduled(fixedDelay = 10000L)
    public void send() {
        rabbitTemplate.convertAndSend(MQConst.FANOUT_EXCHANGE, "", "我是FANOUT_QUEUE_NAME2");
    }
}

```

8.4 启动主类 控制台输出:

```
queue:fanout.message2,message:我是FANOUT_QUEUENAME2  
queue:fanout.message1,message:我是FANOUT_QUEUENAME2  
queue:fanout.message2,message:我是FANOUT_QUEUENAME2
```

总结:

一般direct和topic用来具体的路由消息, 如果要用广播的消息一般用fanout的exchange。header型用的比较少, 但还是知道一点好。

源码链接:

<https://gitee.com/zzmedu/amqb-rabbitmq.git>