限流

springboot
https://www.jianshu.com/p/e02f0014f417

```
    @Bean
    public ConnectionFactory connectionFactory() {
        CachingConnectionFactory connectionFactory = new CachingConnectionFactory();
        connectionFactory.setHost("localhost");
        connectionFactory.setPort(5672);
        connectionFactory.setPassword("guest");
        connectionFactory.setUsername("guest");

connectionFactory.setPublisherConfirmType(CachingConnectionFactory.ConfirmType.CORRELATED);
        connectionFactory.setPublisherReturns(true);
        connectionFactory.createConnection();
        return connectionFactory;
    }
```

一.消费端限流

假设一个场景，比如，由于之前Rabbitmq服务器积压了许多之前未被处理的上万条消息，当我随便打开其中一个消费者客户端，会出现这种问题，信息一涌而进，当数据量特别大的时候可能会导致服务器卡顿或者直接崩溃，于是我们应该对消费端限流，用于保持的稳定。

1.1 basicQos

RabbitMQ 提供了一种 qos （服务质量保证）功能，即在非自动确认消息的前提下，如果一定数目的消息（通过基于 consume 或者 channel 设置 Qos 的值）未被确认前，不进行消费新的消息。

> 自动签收要设置成false, 建议实际工作中也设置成false
> void basicQos(int prefetchSize, int prefetchCount, boolean global) throws IOException;
> prefetchSize：消息大小限制, 一般设置为0, 消费端不做限制
> prefetchCount：会告诉RabbitMQ不要同时给一个消费者推送多于N个消息, 即一旦有N个消息还没有ack, 则该consumer将block(阻塞), 直到有消息ack
> global：true/false 是否将上面设置应用于channel, 简单来说就是上面的限制是channel级别的还是consumer级别
> 注意：
> prefetchSize和global这两项, RabbitMQ没有实现, 暂且不关注, prefetchCount在autoAck设置false的情况下

> 生效, 即在自动确认的情况下这两个值是不生效的

1.2 对消费端进行限流

1)我们既然要使用消费端限流，我们需要关闭自动 ack，将 autoAck 设置为 false

```
channel.basicConsume(queueName, false, consumer);
```

2)我们来设置具体的限流大小以及数量。

```
// 0,15 从0到15限制15条
// 设置 false 应该于consumer级别
channel.basicQos(0, 15, false);
```

1. 在消费者的 handleDelivery 消费方法中手动 ack，并且设置批量处理 ack 回应为 true

```
channel.basicAck(envelope.getDeliveryTag(), true);
```

2 生产端和消费端工程配置

省略配置....可翻之前的Rabbitmq工程配置

3.生产端工程

config配置

3.1 config配置

声明队列，此处使用direct队列

```
@Component
@Slf4j
public class RabbitListenerConfig {

    @Bean
    public ConnectionFactory connectionFactory() {
        CachingConnectionFactory connectionFactory = new CachingConnectionFactory();
        connectionFactory.setHost("localhost");
        connectionFactory.setPort(5672);
        connectionFactory.setPassword("guest");
        connectionFactory.setUsername("guest");

 connectionFactory.setPublisherConfirmType(CachingConnectionFactory.ConfirmType.CORRELATED);
```

```
        connectionFactory.setPublisherReturns(true);
        connectionFactory.createConnection();
        return connectionFactory;
    }

    @Bean
    public RabbitListenerContainerFactory rabbitListenerContainerFactory(ConnectionFactory
connectionFactory){
        SimpleRabbitListenerContainerFactory factory = new
SimpleRabbitListenerContainerFactory();
        factory.setConnectionFactory(connectionFactory);
        return factory;
    }
    @Bean
    @Qualifier("rabbitTemplate")
    public RabbitTemplate rabbitTemplate(ConnectionFactory connectionFactory) {
        RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);
        //开启mandatory模式（开启失败回调）
        rabbitTemplate.setMandatory(true);
        //添加失败回调方法
        rabbitTemplate.setReturnCallback((message, replyCode, replyText, exchange, routingKey) -
> {
            log.info("message:{}, replyCode:{}, replyText:{}, exchange:{}, routingKey:{}",
                    message, replyCode, replyText, exchange, routingKey);
        });
        // 添加发送方确认模式方法
        rabbitTemplate.setConfirmCallback((correlationData, ack, cause) ->
                log.info("correlationData:{}, ack:{}, cause:{}",
                        correlationData.getId(), ack, cause));
        return rabbitTemplate;
    }


    /***声明 direct 队列  一对一***/
    @Bean
    public Exchange directExchange(){
        return new DirectExchange("dircet.exchange.test");
    }

    @Bean
    public Queue directQueue(){
        return new Queue("direct.queue.test");
    }

    @Bean
    public Binding directBinding(){
        return new Binding("direct.queue.test",
                                Binding.DestinationType.QUEUE,
                                "dircet.exchange.test",
                                    "direct.key",null);
    }
}
```

dto

```
@Getter
@Setter
@ToString
public class OrderMessageDTO implements Serializable{
    private Integer orderId;
    private BigDecimal price;
    private Integer productId;
}
```

service

```
public interface DirectService {
public void sendMessage();

//  使用限流Qos
public void sendQosMessage() throws JsonProcessingException;
```

3.4 impl
此处使用了另一种传递消息，使用了MessageProperties，将MessageProperties传递的对象转换成message

```
@Slf4j
@Service
public class DirectServiceImpl implements DirectService {

    @Autowired
    private RabbitTemplate rabbitTemplate;


    ObjectMapper objectMapper = new ObjectMapper();

    /**
     * 发送Qos 限流消息
     */
    @Override
    public void sendQosMessage() throws JsonProcessingException {
        /* 使用MessageProperties传递的对象转换成message*/
        MessageProperties messageProperties = new MessageProperties();
        OrderMessageDTO orderMessageDTO = new OrderMessageDTO();
        orderMessageDTO.setProductId(100);
        orderMessageDTO.setPrice(new BigDecimal("20"));
        orderMessageDTO.setOrderId(1);
```

```
        String messageToSend = objectMapper.writeValueAsString(orderMessageDTO);
        Message message = new Message(messageToSend.getBytes(),messageProperties);
        // 发送端确认是否确认消费
        CorrelationData correlationData = new CorrelationData();
        // 唯一ID
        correlationData.setId(orderMessageDTO.getOrderId().toString());

 rabbitTemplate.convertAndSend("dircet.exchange.test","direct.queue.test",message,correlationDat
a);
    }
}
```

controller

```
@RestController
@Slf4j
@RequestMapping("/api")
public class SendController {
    @Autowired
    private DirectService directService;

    @GetMapping("/qosDirect")
    public void qosDirect() throws JsonProcessingException {
        for (int i = 0; i < 200; i++) {
            directService.sendQosMessage();
        }
    }
}
```

消费端工程

config配置

```
@Component
@Slf4j
public class RabbitListenerConfig {

    @Bean
    public ConnectionFactory connectionFactory() {
        CachingConnectionFactory connectionFactory = new CachingConnectionFactory();
        connectionFactory.setHost("localhost");
        connectionFactory.setPort(5672);
        connectionFactory.setPassword("guest");
        connectionFactory.setUsername("guest");

 connectionFactory.setPublisherConfirmType(CachingConnectionFactory.ConfirmType.CORRELATED);
```

```
        connectionFactory.setPublisherReturns(true);
        connectionFactory.createConnection();
        return connectionFactory;
    }

    @Bean
    public RabbitListenerContainerFactory rabbitListenerContainerFactory(ConnectionFactory
connectionFactory){
        SimpleRabbitListenerContainerFactory factory = new
SimpleRabbitListenerContainerFactory();
        factory.setConnectionFactory(connectionFactory);
        return factory;
    }

}
```

impl

```
    @RabbitListener(
            containerFactory = "rabbitListenerContainerFactory",
            bindings = {
                    @QueueBinding(
                            value = @Queue(name = "direct.queue.test"),
                            exchange = @Exchange(name = "dircet.exchange.test",
                                    type = ExchangeTypes.DIRECT),
                            key = "direct.queue.test"
                    )
            }
    )
    @Override
    public void QosDirectRecive(@Payload Message message, Channel channel) throws IOException {
        log.info("========directQos接受消息===========");
        channel.basicQos(0,10,false);
        DefaultConsumer consumer = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                try {
                    Thread.sleep(5000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                String message = new String(body, "UTF-8");
                log.info("[x] Received '" + message + "'");

                channel.basicAck(envelope.getDeliveryTag(), true);
            }
        };
        //. 设置 Channel 消费者绑定队列
        channel.basicConsume("direct.queue.test",false,consumer);
    }
```

spring版

customer

rabbitmq.properties

```
rabbitmq.host=172.16.98.133
rabbitmq.port=5672
rabbitmq.username=guest
rabbitmq.password=guest
rabbitmq.virtual-host=/
```

spring-rabbitmq-consumer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:rabbit="http://www.springframework.org/schema/rabbit"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       https://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/rabbit
       http://www.springframework.org/schema/rabbit/spring-rabbit.xsd">
    <!--加载配置文件-->
    <context:property-placeholder location="classpath:rabbitmq.properties"/>

    <!-- 定义rabbitmq connectionFactory -->
    <rabbit:connection-factory id="connectionFactory" host="${rabbitmq.host}"
                               port="${rabbitmq.port}"
                               username="${rabbitmq.username}"
                               password="${rabbitmq.password}"
                               virtual-host="${rabbitmq.virtual-host}"/>


    <context:component-scan base-package="com.itheima.listener" />

    <!--定义监听器容器-->
    <rabbit:listener-container connection-factory="connectionFactory" acknowledge="manual"
prefetch="1" >
<!--    <rabbit:listener-container connection-factory="connectionFactory" acknowledge="manual"
>-->
<!--        <rabbit:listener ref="ackListener" queue-names="test_queue_confirm">
</rabbit:listener>-->
        <rabbit:listener ref="qosListener" queue-names="test_queue_confirm"></rabbit:listener>
        <!--定义监听器，监听正常队列-->
```

```
<!--        <rabbit:listener ref="dlxListener" queue-names="test_queue_dlx"></rabbit:listener>--
>

        <!--延迟队列效果实现:  一定要监听的是 死信队列!!! -->
<!--        <rabbit:listener ref="orderListener" queue-names="order_queue_dlx">
</rabbit:listener>-->
    </rabbit:listener-container>

</beans>
```

QosListener

```
package com.itheima.listener;

import com.rabbitmq.client.Channel;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.listener.api.ChannelAwareMessageListener;
import org.springframework.stereotype.Component;

/**
 * Consumer 限流机制
 *  1. 确保ack机制为手动确认。
 *  2. listener-container配置属性
 *      perfetch = 1,表示消费端每次从mq拉去一条消息来消费，直到手动确认消费完毕后，才会继续拉去下一条消息。
 */

@Component
public class QosListener implements ChannelAwareMessageListener {

    @Override
    public void onMessage(Message message, Channel channel) throws Exception {

        Thread.sleep(1000);
        //1.获取消息
        System.out.println(new String(message.getBody()));

        //2. 处理业务逻辑

        //3. 签收
        channel.basicAck(message.getMessageProperties().getDeliveryTag(),true);

    }
}
```