



下载APP



29 | 消息驱动：如何集成 Stream 实现消息驱动？

2022-02-18 姚秋辰

《Spring Cloud 微服务项目实战》

课程介绍 >



讲述：姚秋辰

时长 16:53 大小 15.47M



你好，我是姚秋辰。

在上节课中，我们通过一些实际案例了解到了消息驱动技术的应用场景，这节课我们就使用 Spring Cloud Stream 技术来一场演练，基于 RabbitMQ 消息中间件来落地实践场景。

领资料

以往我们在项目中使用 Stream 时，大都是使用经典的 @Input、@Output 和 @StreamListener 等注解来注册消息生产者和消费者，而 Stream 在 3.1 版本之后在这个注解上打了一个 @Deprecated 标记，意思是这种对接方式已经被淘汰了，不推荐使用。取而代之的是更为流行的 Functional Programming 风格，也就是我们俗称的函数式编程。



从近几年的技术发展趋势就可以看出来，函数式编程成了一种技术演进的趋势，它能让你以更少的代码和精简化的配置实现自己的业务逻辑。函数式编程和约定大于配置相结合的编程风格比较放飞自我，在接下来的实战环节中，你就会体会到这种 less code style 的快感了。

因为函数式消息驱动在同一个应用包含多个 Event Topic 的情况下有一些特殊配置，所以为了方便演示这个场景，我选择了 Customer 服务中的两个具有关联性的业务，分别是用户领取优惠券和删除优惠券，这节课我们就将这两个服务改造成基于消息驱动的实现方式。


实现消息驱动

我把业务场景里的消息生产者和消费者都定义在了 Customer 服务中，可能你会以为，在真实项目里，生产者和消费者应该分别定义在不同的应用中，大多数情况下确实如此。比如在 [🔗 上节课](#) 的消息广播场景里，一个订单完成之后，通过广播消息触发下游各个服务的业务流程，这里的生产者和消费者是分在不同应用中的。

但是呢，我们也有把生产者和消费者定义在同一个应用中的场景，我叫它自产自销。比如在一些削峰填谷的例子中，为了平滑处理用户流量并降低负载，我们可以将高 QPS 但时效性要求不高的请求堆积到消息组件里，让当前应用的消费者慢慢去处理。比如我曾经实现的批量发布商品就是这么个自产自销的例子，商品服务接收请求后丢到 MQ，让同一个应用内部的消费者慢慢消化。

我们接下来就分三步走，用这个自产自销的路子来实现消息驱动业务。先添加生产者代码，再定义消费者逻辑，最后添加配置文件。

按照惯例，集成之前你需要先把下面这个 Stream 依赖项添加到 coupon-customer-impl 项目的 pom 文件中。由于我们底层使用的中间件是 RabbitMQ，所以我们引入的是 stream-rabbit 组件，如果你使用的是不同的中间件，那么需要引入对口的依赖项。

 复制代码

```
1 <dependency>
2     <groupId>org.springframework.cloud</groupId>
3     <artifactId>spring-cloud-starter-stream-rabbit</artifactId>
4 </dependency>
```

添加好依赖项之后，我们先来动手编写生产者逻辑。

添加生产者


生产者只做一件事，就是生产一个消息事件，并将这个事件发送到 RabbitMQ。我在 Customer 服务下创建了一个叫做 CouponProducer 的类，添加了 sendCoupon 和 deleteCoupon 这两个生产者方法，分别对应了领取优惠券和删除优惠券。在这两个方法内，我使用了 StreamBridge 这个 Stream 的原生组件，将信息发送给 RabbitMQ。

[复制代码](#)

```
1 @Service
2 @Slf4j
3 public class CouponProducer {
4
5     @Autowired
6     private StreamBridge streamBridge;
7
8     public void sendCoupon(RequestCoupon coupon) {
9         log.info("sent: {}", coupon);
10        streamBridge.send(EventConstant.ADD_COUPON_EVENT, coupon);
11    }
12
13    public void deleteCoupon(Long userId, Long couponId) {
14        log.info("sent delete coupon event: userId={}, couponId={}", userId, c
15        streamBridge.send(EventConstant.DELETE_COUPON_EVENT, userId + "," + co
16    }
17
18 }
```

在这段代码里，streamBridge.send 方法的第一个参数是 Binding Name，它指定了这条消息要被发到哪一个信道中，其中 ADD_COUPON_EVENT=addCoupon-out-0，而 deleteCoupon=deleteCoupon-out-0。你先不要管这两个奇怪的值是什么，你只要把 Binding Name 理解成一条消息从 Stream 到达 RabbitMQ 之间的“通道”，待会儿看到配置文件的时候，你就会清楚这条通道是怎么与 RabbitMQ 中定义的消息队列名称关联起来的了。

消息的生产者已经定义好了，接下来我在 CouponCustomerController 中新添加了两个方法，单独用来测试我们定义的两个生产者服务。这两个 Controller 方法接收的参数和现有的领券、删除券的接口是一致的，唯二的区别是请求路径后面多了个 Event，以及方法的返回值变成了 void。

 复制代码

```
1 @PostMapping("requestCouponEvent")
2 public void requestCouponEvent(@Valid @RequestBody RequestCoupon request) {
3     couponProducer.sendCoupon(request);
4 }
5
6 // 用户删除优惠券
7 @DeleteMapping("deleteCouponEvent")
8 public void deleteCouponEvent(@RequestParam("userId") Long userId,
9                               @RequestParam("couponId") Long couponId) {
10     couponProducer.deleteCoupon(userId, couponId);
11 }
```


到这里，我们生产者端的配置就完成了，接下来我们就去编写消息的消费者。

添加消息消费者

我在 CouponProducer 的同级目录下创建了一个 CouponConsumer 类，它作为消息的消费者，从 RabbitMQ 处消费由生产者发布的消息事件，方法底层仍然是调用 CustomerService 服务来完成业务逻辑。

在这段代码中，有一个“**约定大于配置**”的规矩你一定要遵守，那就是不要乱起方法名。我这里定义的 addCoupon、deleteCoupon 两个方法名是有来头的，你要确保消费者方法的名称和配置文件中所定义的 Function Name 以及 Binding Name 保持一致，这是 function event 的一条潜规则。因为在默认情况下，框架会使用消费者方法的 method name 作为当前消费者的标识，如果消费者标识和配置文件中的名称不一致，那么 Spring 应用就不知道该把当前的消费者绑定到哪一个 Stream 信道上。

另外有一点需要提醒你，我在代码中采用了 Consumer 的实现方式，它是函数式编程的一种方式，你也可以根据自己的编程习惯，采用其它函数式编程方式来编写这段逻辑。

 复制代码

```
1 @Slf4j
2 @Service
3 public class CouponConsumer {
4
5     @Autowired
6     private CouponCustomerService customerService;
7
8     @Bean
9     public Consumer<RequestCoupon> addCoupon() {
```

```
10         return request -> {
11             log.info("received: {}", request);
12             customerService.requestCoupon(request);
13         };
14     }
15
16     @Bean
17     public Consumer<String> deleteCoupon() {
18         return request -> {
19             log.info("received: {}", request);
20             List<Long> params = Arrays.stream(request.split(","))
21                 .map(Long::valueOf)
22                 .collect(Collectors.toList());
23             customerService.deleteCoupon(params.get(0), params.get(1));
24         };
25     }
26
27 }
```

到这里消费者的定义也完成了。在定义生产者和消费者的过程中我多次提到了配置文件，下面我们就来看一下 Stream 的配置项都有哪些内容。

添加配置文件

Stream 的配置项比较多，我打算分 Binder 和 Binding 两部分来讲。我们先来看 Binder 部分，Binder 中配置了对接外部消息中间件所需要的连接信息。如果你的程序中只使用了单一的中间件，比如只接入了 RabbitMQ，那么你可以直接在 spring.rabbitmq 节点下配置连接串，不需要特别指定 binders 配置。

如果你在 Stream 中需要同时对接多个不同类型，或多个同类型但地址端口各不相同的消息中间件，那么你可以把这些中间件的信息配置在 spring.cloud.stream.binders 节点下。其中 type 属性指定了当前消息中间件的类型，而 environment 则指定了连接信息。

 复制代码

```
1 spring:
2   cloud:
3     stream:
4       # 如果你项目里只对接一个中间件，那么不用定义binders
5       # 当系统要定义多个不同消息中间件的时候，使用binders定义
6       binders:
7         my-rabbit:
8           type: rabbit # 消息中间件类型
9           environment: # 连接信息
```

```
10         spring:
11             rabbitmq:
12                 host: localhost
13                 port: 5672
14                 username: guest
15                 password: guest
16
```

配置完了 binders，我们接下来看看如何定义 spring.cloud.stream.bindings 节点，这个节点保存了生产者、消费者、binder 和 RabbitMQ 四方的关联关系。

[复制代码](#)

```
1  spring:
2    cloud:
3      stream:
4        bindings:
5          # 添加coupon - Producer
6          addCoupon-out-0:
7            destination: request-coupon-topic
8            content-type: application/json
9            binder: my-rabbit
10         # 添加coupon - Consumer
11         addCoupon-in-0:
12           destination: request-coupon-topic
13           content-type: application/json
14           # 消费组，同一个组内只能被消费一次
15           group: add-coupon-group
16           binder: my-rabbit
17         # 删除coupon - Producer
18         deleteCoupon-out-0:
19           destination: delete-coupon-topic
20           content-type: text/plain
21           binder: my-rabbit
22         # 删除coupon - Consumer
23         deleteCoupon-in-0:
24           destination: delete-coupon-topic
25           content-type: text/plain
26           group: delete-coupon-group
27           binder: my-rabbit
28         function:
29           definition: addCoupon;deleteCoupon
```

我们以 addCoupon 为例，你会看到我定义了 addCoupon-out-0 和 addCoupon-in-0 这两个节点，节点名称中的 out 代表当前配置的是一个生产者，而 in 则代表这是一个消费者，这便是 spring-function 中约定的命名关系：

Input 信道（消费者）：< functionName > - in - < index > ；

Output 信道（生产者）：< functionName > - out - < index >。


你可能注意到了，在命名规则的最后还有一个 index，它是 input 和 output 的序列，如果同一个 function name 只有一个 output 和一个 input，那么这个 index 永远都是 0。而如果你需要为一个 function 添加多个 input 和 output，就需要使用 index 变量来区分每个生产者消费者了。如果你对 index 的使用场景感兴趣，可以参考文稿中的 [官方社区文档](#)。

现在你已经了解了生产者和消费者的信道是如何定义的，但是，至于这个信道和 RabbitMQ 里定义的消息队列之间的关系，你知道是怎么指定的吗？

信道和 RabbitMQ 的绑定关系是通过 binder 属性指定的。如果当前配置文件的上下文中只有一个消息中间件（比如使用默认的 MQ），你并不需要声明 binder 属性。但如果你配置了多个 binder，那就需要为每个信道声明对应的 binder 是谁。addCoupon-out-0 对应的 binder 名称是 my-rabbit，这个 binder 就是我刚才在 spring.cloud.stream.binders 里声明的配置。通过这种方式，生产者消费者信道到消息中间件（binder）的联系就建立起来了。

信道和消息队列的关系是通过 destination 属性指定的。以 addCoupon 为例，我在 addCoupon-out-0 生产者配置项中指定了 destination=request-coupon-topic，意思是将消息发送到名为 request-coupon-topic 的 Topic 中。我又在 addCoupon-in-0 消费者里添加了同样的配置，意思是让当前消费者从 request-coupon-topic 消费新的消息。

RabbitMQ 消息组件内部是通过交换机（Exchange）和队列（Queue）来做消息投递的，如果你登录 RabbitMQ 的控制台，就可以在 Exchanges 下看到我声明的 delete-coupon-topic 和 request-coupon-topic。

 RabbitMQ™

RabbitMQ 3.9.8Erlang 24.1.2

Overview

Connections

Channels

Exchanges

Queues

Admin

Exchanges

▼ All exchanges (10)

Pagination

Page 1 ▼ of 1 - Filter: ☐ Regex ?

Name	Type	Features	Message rate in	Message rate out	+/-
(AMQP default)	direct	D	0.00/s	0.00/s	
addCoupondeleteCoupon-in-0	topic	D			
amq.direct	direct	D			
amq.fanout	fanout	D			
amq.headers	headers	D			
amq.match	headers	D			
amq.rabbitmq.trace	topic	D I			
amq.topic	topic	D			
delete-coupon-topic	topic	D	0.00/s	0.00/s	
request-coupon-topic	topic	D	0.00/s	0.00/s	

切换到 Queues 面板，你还会看到这两个交换机所绑定的队列名称。这里的队列名称后面还跟了一个 group name，这就是我在消费者这一侧设置的消息分组，我在配置项中为 add-coupon-in 设置了 group=add-coupon-group，即当前分组内只有一台机器可以去消费队列中的消息，这就是所谓的“消息分组单播”的场景。如果你不设置 group 属性，那么这个消息就会成为一条“广播消息”。

RabbitMQ 3.9.8 Erlang 24.1.2

Overview Connections Channels Exchanges **Queues** Admin

Queues

▼ All queues (3)

Pagination

Page 1 of 1 - Filter: ☐ Regex ?

Overview				Messages			Message rates			+/-
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
delete-coupon-topic.delete-coupon-group	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
request-coupon-topic.add-coupon-group	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
zipkin	classic	D	idle	262	0	262	0.00/s			

► Add a new queue

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

最后的最后，有一个最为重要的配置项，我专门把它放到最后来讲，那就是 `spring.cloud.stream.function`。如果你的项目中只有一组消费者，那么你完全不用搭理这个配置项，只要确保消费者代码中的 `method name` 和 `bindings` 下声明的消费者信道名称相对应就好了；如果你的项目中有多组消费者（比如我声明了 `addCoupon` 和 `deleteCoupon` 两个消费者），在这种情况下，你需要将消费者所对应的 `function name` 添加到 `spring.cloud.stream.function`，否则消费者无法被绑定到正确的信道。

[复制代码](#)

```
1 spring:
2   cloud:
3     stream:
4       function:
5         definition: addCoupon;deleteCoupon
```

到这里，我们就完整搭建了一套消息驱动的方案。下面让我来带你回顾下本节重点吧。

总结

在今天的课程里，我们使用了 Stream 在新版本中推荐的 `functional event` 风格实现了用户领券和删除券。在这个环节里，你需要注意的是遵循 `spring function` 的约定，在下面的几个地方使用一致的命名规则。

1. 生产者端代码中的 binding name (addCoupon-out-0) 和配置文件中的生产者名称一致；
2. 同一对生产者和消费者，在配置文件中要使用一样的 Topic Name；
3. 如果项目中存在多个消费者，使用 spring.cloud.stream.function 或者 spring.cloud.function 把所有消费者的 function name 写出来。

约定大于配置这种风格，对初学者来说，是需要一些上手门槛的，但当你熟悉了这里面的门道之后，你就能利用这种 less code 开发风格大幅提高开发效率。

思考题

如果 Consumer 在消费消息的时候发生了异常，你知道有哪些异常处理的方式吗？

好啦，这节课就结束啦。欢迎你把这节课分享给更多对 Spring Cloud 感兴趣的朋友。我是姚秋辰，我们下节课再见！

分享给需要的人，Ta购买本课程，你将得 20 元

 生成海报并分享

 赞 1  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 28 | 消息驱动：谁说消息队列只能削峰填谷？

下一篇 30 | 消息驱动：如何高效处理 Stream 中的异常？

精选留言 (4)

 写留言



奔跑的蚂蚁

2022-02-24

试了下spring-cloud-starter-stream-rocketmq 的 function binding
第一次发送消息会报日志

DefaultBinderFactory : Retrieving cached binder: rocketmq

DirectWithAttributesChannel : Channel 'unknown.channel.name' has 1 subscriber(s).

BeanFactoryAwareFunctionRegistry : Looking up function 'streamBridge' with accepted...

展开 ∨



wake

2022-02-22

老师如果我其它方法名也叫addCoupon会被误认为消费者吗？或者说只有返回值为Consumer才会被认定为消费者，那如果有两个同样的返回Consumer的addCoupon方法呢

作者回复: 其实和返回值关系不大，Consumer只是函数式编程的一种实现方式，还有其他方式比如Flux、Supplier。如果方法重名，我印象中是遵循先来后到，后初始化的bean对应的consumer优先生效。同学可以本地试一下，回头告诉我们答案哈



奔跑的蚂蚁

2022-02-18

我们项目用了spirng.cloud.stream.rocketMq 出了异常是自动重试，老师能讲讲原理嘛 能自定义重试间隔是次数嘛（是不是mq里面配置的）

作者回复: 重试策略（次数+每次重试间隔）会在后面的一节课程里单独讲到，应该快上线了



peter

2022-02-18

请教老师几个问题：

Q1：消费者标识是否可以不用方法名，另外指定一个？

Q2：“采用了 Consumer 的实现方式”，“添加消息消费者”部分的这句话中，consumer的实现方式是指什么？笔误吗？

Q3：“return request ->”，request从哪里来？...

展开 ∨

作者回复: Q1：不走寻常路是可以，但是失去了约定大约配置的意义

Q2：Consumer是函数式编程的一种方式

Q3：函数式编程语法，建议先找一些开源资料熟悉下语法

Q4: 任意名字

Q5：交换机和队列之间可以建立绑定关系，官网的帮助文档里有详细信息<https://www.rabbitmq.com/documentation.html>

