

1. 消息驱动概述

1.1. 是什么

1.1.1. 一句话

1.1.2. 什么是 Spring Cloud Stream

1.1.3. 官网

1.2. 设计思想

1.2.1. 标准MQ

1.2.2. 为什么用SpringCloud Stream

1.2.2.1. stream凭什么可以统一底层差异

1.2.2.2. Binder

1.2.3. Stream中的消息通信方式遵循了发布-订阅模式

1.3. SpringCloud Stream标准流程

1.3.1. Binder

1.3.2. Channel

1.3.3. Source和Sink

1.4. 编码API和常用注解

2. 案例说明

3. 消息驱动之生产者

3.1. 新建Module

3.2. POM

3.3. YML

3.4. 主启动

3.5. 业务类

3.5.1. 发送消息接口

3.5.2. 发送消息接口实现类

3.5.3. Controller

3.6. 测试

4. 消息驱动之消费者

4.1. 新建Module

4.2. POM

4.3. YML

4.4. 主启动

4.5. 业务类

4.5.1. Controller

4.6. 测试

5. 分组消费与持久化

5.1. 依照8802， clone出来一份运行8803

5.1.1. 新建Module

5.1.2. POM

5.1.3. YML

5.1.4. 主启动

5.1.5. 业务类

5.1.5.1. Controller

5.2. 启动

5.3. 运行后的两个问题

5.4. 消费

5.4.1. 现象

5.4.2. 如何解决

5.4.3. 生产实际案例

5.5. 分组

5.5.1. 原理

5.5.2. 案例演示1(8802/8803都变成不同组， group两个不同)

5.5.2.1. 8002修改YML

5.5.2.2. 8803修改YML

5.5.2.3. 测试

5.5.3. 结论1(8802/8803都变成不同组， group两个不同)

5.5.4. 案例演示2(8802/8803都变成相同组， group两个相同)

5.5.4.1. 8002修改YML

5.5.4.2. 8803修改YML

5.5.4.3. 测试

5.5.5. 结论2(8802/8803都变成相同组, group两个相同)

5.6. 持久化

5.6.1. 案例演示

1. 消息驱动概述

1.1. 是什么

1.1.1. 一句话

屏蔽底层消息中间件的差异, 降低切换版本, 统一消息的编程模型


1.1.2. 什么是 Spring Cloud Stream

- 官方定义 Spring Cloud Stream是个构建消息驱动微服务的框架。
- 应用程序通过 Inputs或者 outputs来与 Spring Cloud Stream中binder对象交互.通过我们配置来 binding (绑定), 而 Spring Cloud Stream的 binder对象负责与消息中间件交互. 所以, 我们只需要搞清楚如何与 Spring Cloud Stream交互就可以方便使用消息驱动的方式
- 通过使用 Spring Integration来连接消息代理中间件以实现消息事件驱动。
- Spring Cloud Stream为些供应商的消息中间件产品提供了个性化的自动化配置实现, 引用了发布-订阅、消费组、分区的三个核心概念
- 目前仅支持 RabbitMQ、Kafka

1.1.3. 官网

官网: <https://spring.io/projects/spring-cloud-stream#overview>

Spring Cloud Stream是用于构建与共享消息传递系统连接的高度可伸缩的事件驱动微服务框架, 该框架提供了一个灵活的编程模型, 它建立在已经建立和熟悉的 Spring术语和最佳实践上, 包括支持持久化的发布/订阅、消费组以及消息分区这三个核心概念



[Why Spring](#)
[Learn](#)
[Projects](#)
[Training](#)
[Support](#)
[Community](#)

[Spring Boot](#)
[Spring Framework](#)
[Spring Data](#)
[Spring Cloud](#)

[Spring Cloud Azure](#)
[Spring Cloud Alibaba](#)
[Spring Cloud for Amazon Web Services](#)
[Spring Cloud Bus](#)
[Spring Cloud CLI](#)
[Spring Cloud for Cloud](#)

Spring Cloud Stream


Horsham.SR6 [3.0.6 RELEASE]

[OVERVIEW](#)
[LEARN](#)
[SAMPLES](#)

Spring Cloud Stream is a framework for building highly scalable event-driven microservices connected with shared messaging systems.

The framework provides a flexible programming model built on already established and familiar Spring idioms and best practices, including support for persistent pub/sub semantics, consumer groups, and stateful partitions.

文档: <https://cloud.spring.io/spring-cloud-static/spring-cloud-stream/3.0.1.RELEASE/reference/html/>



Spring Cloud Stream Reference Documentation

Sabby Anandan · Marius Bogoevici · Eric Bottard · Mark Fisher · Ilayaperumal Gopinathan · Gunnar Hillert · Mark Pollack · Patrick Peralta · Glenn Renfro · Thomas Risberg · Dave Syer · David Turanski · Janne Valkealahti · Benjamin Klein · Vinicius Carvalho · Gary Russell · Oleg Zhurakousky · Jay Bryant · Soby Chacko

3.0.1.RELEASE

The reference documentation consists of the following sections:

Overview	History, Quick Start, Concepts, Architecture Overview, Binder Abstraction, and Core Features
Rabbit MQ Binder	Spring Cloud Stream binder reference for Rabbit MQ
Apache Kafka Binder	Spring Cloud Stream binder reference for Apache Kafka
Apache Kafka Streams Binder	Spring Cloud Stream binder reference for Apache Kafka Streams
Additional Binders	A collection of Partner maintained binder implementations for Spring Cloud Stream (e.g., Azure Event Hubs, Google PubSub, Solace PubSub+)
Spring Cloud Stream	A curated collection of repeatable Spring Cloud Stream samples to walk through the features

Spring Cloud Stream中文指导手册:

<https://m.wang1314.com/doc/webapp/topic/20971999.html>

<

Spring Cloud Stream中文指导手册

♡

Spring Cloud Stream中文指导手册

凌云 收藏于2019-04-17 转藏3次

举报

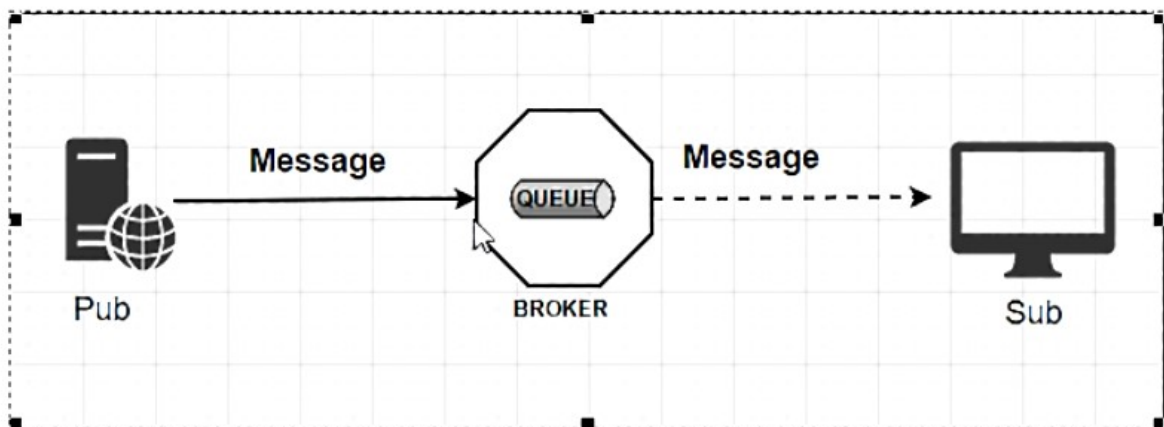
Spring Cloud Stream中文指导手册

source

- Spring Cloud Stream中文指导手册
 - Spring Cloud Stream 核心
 - 1.简介
 - 2.主要概念
 - 2.1.应用程序模型
 - 2.1.1.“胖” JAR
 - 2.2.Binder抽象
 - 2.3.支持持久的发布-订阅模式
 - 2.4.消费者组

1.2. 设计思想

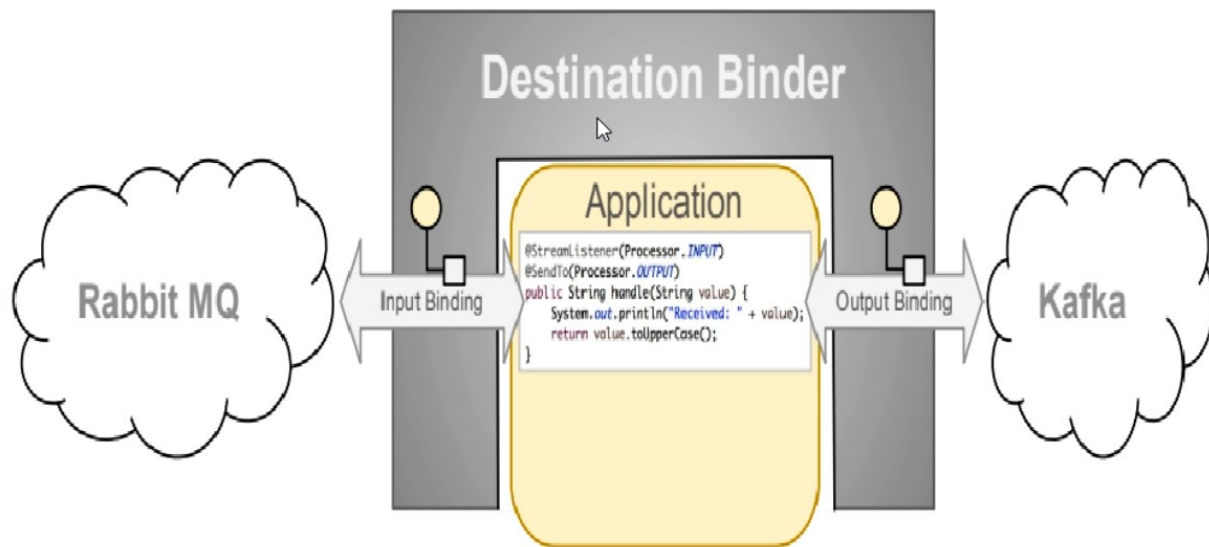
1.2.1. 标准MQ



- 生产者/消费者之间靠消息媒介传递信息内容: Message
- 消息必须走特定的通道: 消息通道MessageChannel
- 消息通道里的消息如何被消费呢，谁负责收发处理: 消息通道MessageChannel的子接口SubscribableChannel,由MessageHandler消息处理器订阅

1.2.2. 为什么用SpringCloud Stream

比方说我们用到了 RabbitMQ和 Kafka，由于这两个消息中间件的架构上的不同，像 RabbitMQ有 exchange，kafka有 Topic和 Partitions分区，



这些中间件的差异性导致我们实际项目开发给我们造成了一定的困扰，我们如果用了两个消息队列的其中一种，后面的业务需求，我想往另外一种消息队列进行迁移，这时候无疑就是一个灾难性的，一大堆东西都要重新推倒重新做，因为它跟我们的系统耦合了，这时候 springcloud Stream给我们提供了一种解耦合的方式。

1.2.2.1. stream凭什么可以统一底层差异

在没有绑定器这个概念的情况下，我们的 SpringBoot应用要直接与消息中间件进行信息交互的时候，由于各消息中间件构建的初衷不同，它们的实现细节上会有较大的差异性

通过定义绑定器作为中间层，完美地实现了应用程序与消息中间件细节之间的隔离。

通过向应用程序暴露统一的 Channel通道，使得应用程序不需要再考虑各种不同的消息中间件实现通过定义绑定器 Binder作为中间层，实现了应用程序与消息中间件细节之间的隔离。

https://cloud.spring.io/spring-cloud-static/spring-cloud-stream-binder-rabbit/3.0.1.RELEASE/reference/html/spring-cloud-stream-binder-rabbit.html#_rabbitmq_binder_overview

[Back to index](#)

Reference Guide

- Usage
- RabbitMQ Binder Overview**
- Configuration Options
- Using Existing Queues/Exchanges
- Retry With the RabbitMQ Binder
- Error Channels
- Dead-Letter Queue Processing
- Partitioning with the RabbitMQ Binder

Appendices

2. RabbitMQ Binder Overview

The following simplified diagram shows how the RabbitMQ binder operates:

Figure 1. RabbitMQ Binder

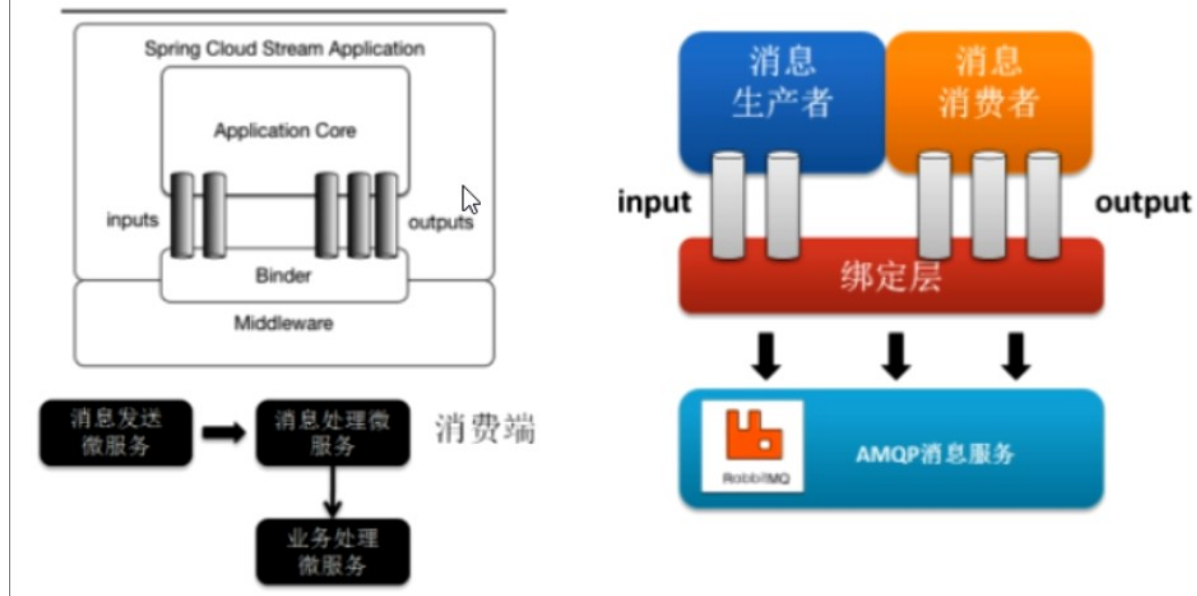
By default, the RabbitMQ Binder implementation maps each destination to a `TopicExchange`. For each consumer group, a `Queue` is bound to that `TopicExchange`. Each consumer instance has a corresponding RabbitMQ `Consumer` instance for its group's `Queue`. For partitioned producers and consumers, the queues are suffixed with the partition index and use the partition index as the routing key. For anonymous consumers (those with no `group` property), an auto-delete queue (with a randomized unique name) is used.

By using the optional `autoBindDl` option, you can configure the binder to create and configure dead-letter

1.2.2.2. Binder

在没有绑定器这个概念的情况下，我们的 Spring Boot应用要直接与消息中间件进行信息交互的时候，由于各消息中间件构建的初衷不同，它们的实现细节上会有较大的差异性，通过定义绑定器作为中间层，完美地实现了应用程序与消息中间件细节之间的隔离。Stream对消息中间件的进一步封装可以做到代码层面对中间件的无感知，甚至于动态的切换中间件（rabbitmq切换为kafka），使得微服务开发的高度解耦，服务可以关注更多自己的业务流程

SpringCloudStream处理架构



通过定义绑定器 Binder 作为中间层，实现了应用程序与消息中间件细节之间的隔离。

INPUT 对应于消费者

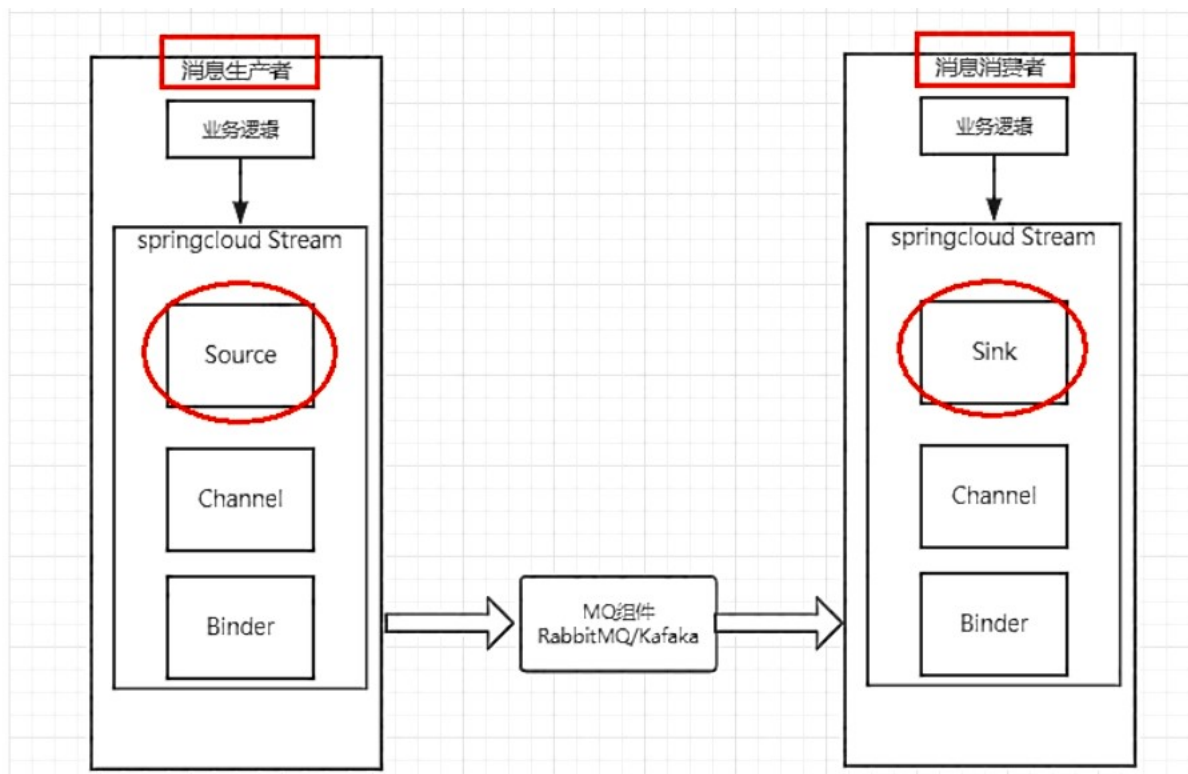
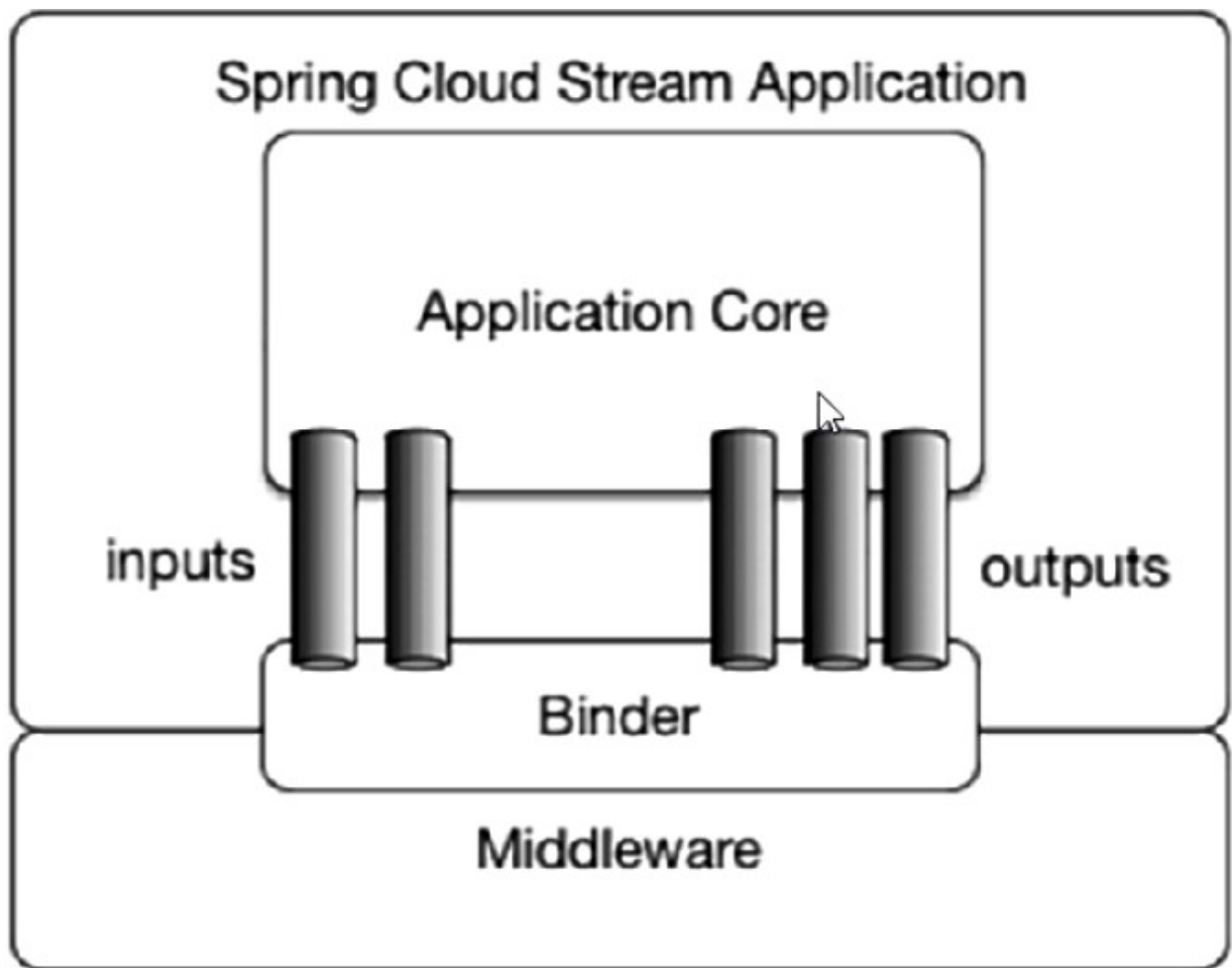
OUTPUT 对应于生产者

1.2.3. Stream 中的消息通信方式遵循了发布-订阅模式

Topic 主题进行广播

- 在 RabbitMQ 就是 Exchange
- 在 kafka 中就是 Topic

1.3. SpringCloud Stream 标准流程



1.3.1. Binder

很方便的连接中间件，屏蔽差异

1.3.2. Channel

通道，是队列Queue的一种抽象，在消息通讯系统中就是实现存储和转发的媒介，通过对Channel对队列进行配置

1.3.3. Source和Sink

简单的可理解为参照对象是Spring Cloud Stream自身，从Stream发布消息就是输出，接受消息就是输入

1.4. 编码API和常用注解



The diagram illustrates the Spring Cloud Stream Application architecture. It shows a central 'Spring Cloud Stream Application' box containing an 'Application Core' and a 'Binder'. The 'Application Core' has 'inputs' and 'outputs' connected to the 'Binder'. The 'Binder' is connected to a 'Middleware' box below it.

组成	说明
Middleware	中间件，目前只支持RabbitMQ和Kafka
Binder	Binder是应用与消息中间件之间的封装，目前实行了Kafka和RabbitMQ的Binder，通过Binder可以很方便的连接中间件，可以动态的改变消息类型(对应于Kafka的topic, RabbitMQ的exchange)，这些都可以通过配置文件来实现
@Input	注解标识输入通道，通过该输入通道接收到的消息进入应用程序
@Output	注解标识输出通道，发布的消息将通过该通道离开应用程序
@StreamListener	监听队列，用于消费者的队列的消息接收
@EnableBinding	指信道channel和exchange绑定在一起

2. 案例说明

前提: RabbitMQ环境已经OK

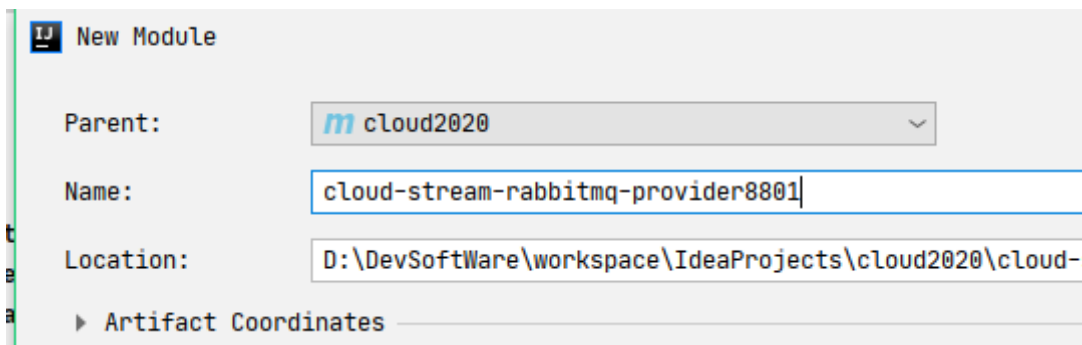
工程中新建三个子模块

- cloud-stream-rabbitmq-provider8801,作为生产者进行发消息模块
- cloud-stream-rabbitmq-consumer8802,作为消息接收模块
- cloud-stream-rabbitmq-consumer8803,作为消息接收模块

3. 消息驱动之生产者

3.1. 新建Module

新建cloud-stream-rabbitmq-provider8801



3.2. POM

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.c
5     <parent>
6         <artifactId>cloud2020</artifactId>
7         <groupId>cn.sitedev.springcloud</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>cloud-stream-rabbitmq-provider8801</artifactId>
13
14    <dependencies>
15
16        <dependency>
17            <groupId>org.springframework.cloud</groupId>
18            <artifactId>spring-cloud-starter-stream-rabbit</artifactId>
19        </dependency>
20
21        <!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cl
22        <dependency>
23            <groupId>org.springframework.cloud</groupId>
24            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
25        </dependency>
26
27        <dependency>
28            <groupId>cn.sitedev.springcloud</groupId>
29            <artifactId>cloud-api-commons</artifactId>
30            <version>1.0-SNAPSHOT</version>
31        </dependency>
32
33        <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot
34        <dependency>
35            <groupId>org.springframework.boot</groupId>
36            <artifactId>spring-boot-starter-web</artifactId>
37        </dependency>
38
39        <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot
40        <dependency>
```

```

41         <groupId>org.springframework.boot</groupId>
42         <artifactId>spring-boot-starter-actuator</artifactId>
43     </dependency>
44
45
46     <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot
47     <dependency>
48         <groupId>org.springframework.boot</groupId>
49         <artifactId>spring-boot-devtools</artifactId>
50         <scope>runtime</scope>
51         <optional>true</optional>
52     </dependency>
53
54     <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
55     <dependency>
56         <groupId>org.projectlombok</groupId>
57         <artifactId>lombok</artifactId>
58         <optional>true</optional>
59     </dependency>
60
61     <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot
62     <dependency>
63         <groupId>org.springframework.boot</groupId>
64         <artifactId>spring-boot-starter-test</artifactId>
65         <scope>test</scope>
66     </dependency>
67
68 </dependencies>
69
70 </project>

```

3.3. YML

```

1 server:
2   port: 8801
3
4 spring:
5   application:
6     name: cloud-stream-provider
7   cloud:
8     stream:

```

```

9     binders: # 在此处配置要绑定的rabbitmq的服务信息;
10     defaultRabbit: # 表示定义的名称, 用于于binding整合
11         type: rabbit # 消息组件类型
12     environment: # 设置rabbitmq的相关的环境配置
13         spring:
14             rabbitmq:
15                 host: localhost
16                 port: 5672
17                 username: guest
18                 password: guest
19     bindings: # 服务的整合处理
20     output: # 这个名字是一个通道的名称
21         destination: studyExchange # 表示要使用的Exchange名称定义
22         content-type: application/json # 设置消息类型, 本次为json, 文本则设置"text
23         binder: defaultRabbit # 设置要绑定的消息服务的具体设置
24
25 eureka:
26     client: # 客户端进行Eureka注册的配置
27         service-url:
28             defaultZone: http://localhost:7001/eureka
29     instance:
30         lease-renewal-interval-in-seconds: 2 # 设置心跳的时间间隔 (默认是30秒)
31         lease-expiration-duration-in-seconds: 5 # 如果现在超过了5秒的间隔 (默认是90秒)
32         instance-id: send-8801.com # 在信息列表时显示主机名称
33         prefer-ip-address: true # 访问的路径变为IP地址

```

3.4. 主启动

```

1 package cn.sitedev.springcloud;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class StreamMQMain8801 {
8     public static void main(String[] args) {
9         SpringApplication.run(StreamMQMain8801.class, args);
10    }
11 }

```

3.5. 业务类

3.5.1. 发送消息接口

```
1 package cn.sitedev.springcloud.service;
2
3 public interface IMessageProvider {
4     String send();
5 }
```

3.5.2. 发送消息接口实现类

```
1 package cn.sitedev.springcloud.service.impl;
2
3 import cn.sitedev.springcloud.service.IMessageProvider;
4 import org.springframework.cloud.stream.annotation.EnableBinding;
5 import org.springframework.cloud.stream.messaging.Source;
6 import org.springframework.integration.support.MessageBuilder;
7 import org.springframework.messaging.MessageChannel;
8
9 import javax.annotation.Resource;
10 import java.util.UUID;
11
12 @EnableBinding(Source.class) //定义消息的推送管道
13 public class MessageProviderImpl implements IMessageProvider {
14     @Resource
15     private MessageChannel output; // 消息发送管道
16
17     @Override
18     public String send() {
19         String serial = UUID.randomUUID().toString();
20         output.send(MessageBuilder.withPayload(serial).build());
21         System.out.println("*****serial: " + serial);
22         return null;
23     }
24 }
```

3.5.3. Controller

```

1 package cn.sitedev.springcloud.controller;
2
3 import cn.sitedev.springcloud.service.IMessageProvider;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 import javax.annotation.Resource;
8
9 @RestController
10 public class SendMessageController {
11     @Resource
12     private IMessageProvider messageProvider;
13
14     @GetMapping(value = "/sendMessage")
15     public String sendMessage() {
16         return messageProvider.send();
17     }
18 }

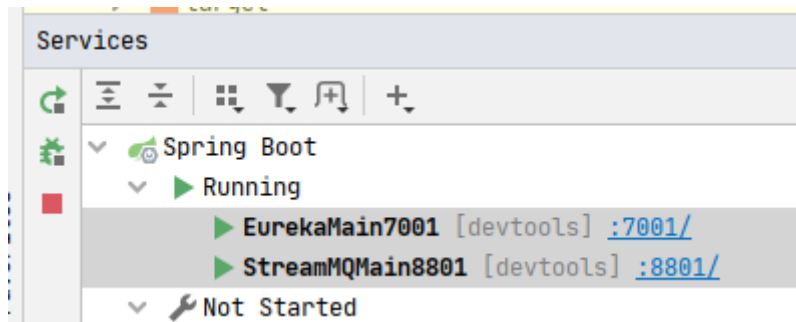
```

3.6. 测试

启动RabbitMQ(如果没有安装rabbitmq_management插件请先安装)

启动EurekaMain7001

启动StreamMQMain8801



浏览器访问RabbitMQ管理页面<http://localhost:15672> , 查看Exchanges

Exchanges

▼ All exchanges (9)

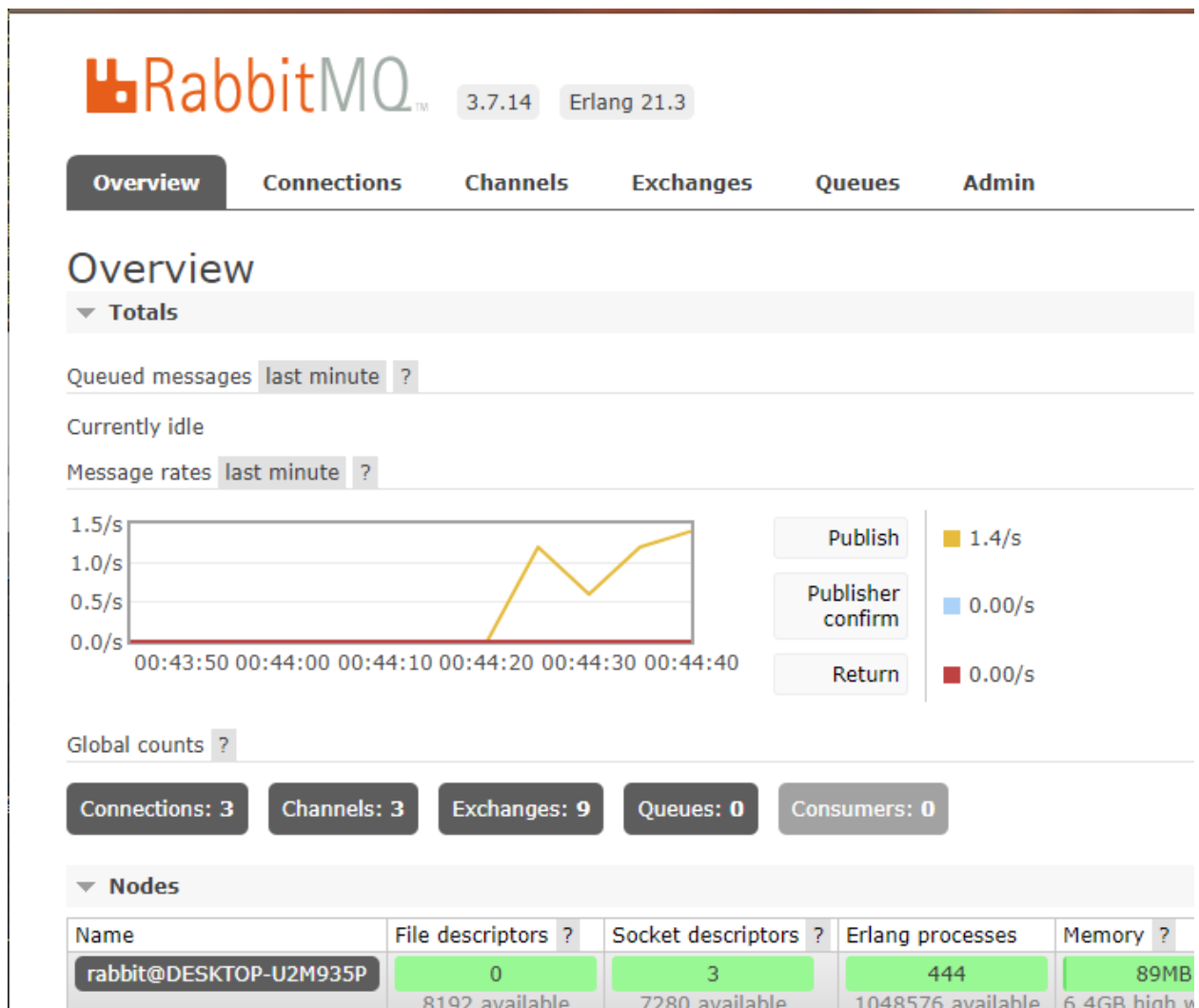
Pagination

Page 1 of 1 - Filter: ☐ Regex ?

Name	Type	Features	Message rate in	Message rate out	+/-
(AMQP default)	direct	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			
springCloudBus	topic	D			
studyExchange	topic	D			

► Add a new exchange

浏览器多次访问<http://localhost:8801/sendMessage> , 查看RabbitMQ管理页面的Overview



4. 消息驱动之消费者

4.1. 新建Module

新建cloud-stream-rabbitmq-consumer8802

New Module

Parent:

Name:

Location:

► Artifact Coordinates

4.2. POM

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
```



```
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.c
5  <parent>
6      <artifactId>cloud2020</artifactId>
7      <groupId>cn.sitedev.springcloud</groupId>
8      <version>1.0-SNAPSHOT</version>
9  </parent>
10 <modelVersion>4.0.0</modelVersion>
11
12 <artifactId>cloud-stream-rabbitmq-consumer8802</artifactId>
13
14 <dependencies>
15
16     <dependency>
17         <groupId>org.springframework.cloud</groupId>
18         <artifactId>spring-cloud-starter-stream-rabbit</artifactId>
19     </dependency>
20
21     <!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cl
22     <dependency>
23         <groupId>org.springframework.cloud</groupId>
24         <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
25     </dependency>
26
27     <dependency>
28         <groupId>cn.sitedev.springcloud</groupId>
29         <artifactId>cloud-api-commons</artifactId>
30         <version>${project.version}</version>
31     </dependency>
32
33
34     <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot
35     <dependency>
36         <groupId>org.springframework.boot</groupId>
37         <artifactId>spring-boot-starter-web</artifactId>
38     </dependency>
39
40     <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot
41     <dependency>
42         <groupId>org.springframework.boot</groupId>
43         <artifactId>spring-boot-starter-actuator</artifactId>
44     </dependency>
45
```

```

46
47     <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot
48     <dependency>
49         <groupId>org.springframework.boot</groupId>
50         <artifactId>spring-boot-devtools</artifactId>
51         <scope>runtime</scope>
52         <optional>true</optional>
53     </dependency>
54
55     <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
56     <dependency>
57         <groupId>org.projectlombok</groupId>
58         <artifactId>lombok</artifactId>
59         <optional>true</optional>
60     </dependency>
61
62     <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot
63     <dependency>
64         <groupId>org.springframework.boot</groupId>
65         <artifactId>spring-boot-starter-test</artifactId>
66         <scope>test</scope>
67     </dependency>
68
69 </dependencies>
70
71 </project>

```

4.3. YML

```

1 server:
2   port: 8802
3
4 spring:
5   application:
6     name: cloud-stream-consumer
7   cloud:
8     stream:
9       binders: # 在此处配置要绑定的rabbitmq的服务信息；
10      defaultRabbit: # 表示定义的名称，用于于binding整合
11      type: rabbit # 消息组件类型
12      environment: # 设置rabbitmq的相关的环境配置

```

```

13         spring:
14             rabbitmq:
15                 host: localhost
16                 port: 5672
17                 username: guest
18                 password: guest
19     bindings: # 服务的整合处理
20     input: # 这个名字是一个通道的名称
21         destination: studyExchange # 表示要使用的Exchange名称定义
22         content-type: application/json # 设置消息类型，本次为json，文本则设置"text
23         binder: defaultRabbit # 设置要绑定的消息服务的具体设置
24
25 eureka:
26     client: # 客户端进行Eureka注册的配置
27         service-url:
28             defaultZone: http://localhost:7001/eureka
29     instance:
30         lease-renewal-interval-in-seconds: 2 # 设置心跳的时间间隔（默认是30秒）
31         lease-expiration-duration-in-seconds: 5 # 如果现在超过了5秒的间隔（默认是90秒）
32         instance-id: receive-8802.com # 在信息列表时显示主机名称
33         prefer-ip-address: true # 访问的路径变为IP地址

```

4.4. 主启动

```

1 package cn.sitedev.springcloud;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class StreamMQMain8802 {
8
9     public static void main(String[] args) {
10         SpringApplication.run(StreamMQMain8802.class, args);
11     }
12 }

```

4.5. 业务类

4.5.1. Controller

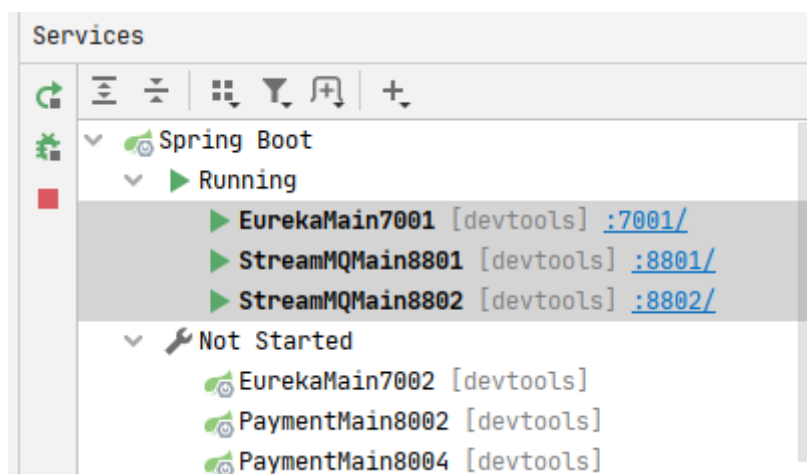
```
1 package cn.sitedev.springcloud.controller;
2
3 import org.springframework.cloud.stream.annotation.StreamListener;
4 import org.springframework.messaging.Message;
5 import org.springframework.beans.factory.annotation.Value;
6 import org.springframework.cloud.stream.annotation.EnableBinding;
7 import org.springframework.cloud.stream.messaging.Sink;
8 import org.springframework.stereotype.Component;
9
10 @Component
11 @EnableBinding(Sink.class)
12 public class ReceiveMessageListenerController {
13     @Value("${server.port}")
14     private String serverPort;
15
16     @StreamListener(Sink.INPUT)
17     public void input(Message<String> message) {
18         System.out.println("消费者1号，接受: " + message.getPayload() + "\t port:" +
19     }
20
21 }
```

4.6. 测试

启动EurekaMain7001

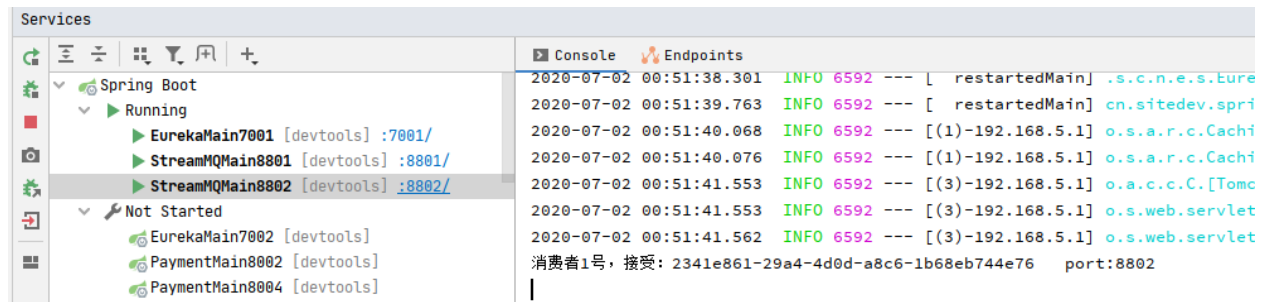
启动StreamMQMain8801

启动StreamMQMain8802



浏览器访问<http://localhost:8801/sendMessage> , 测试8801发送8802接收消息

查看8802控制台输出:

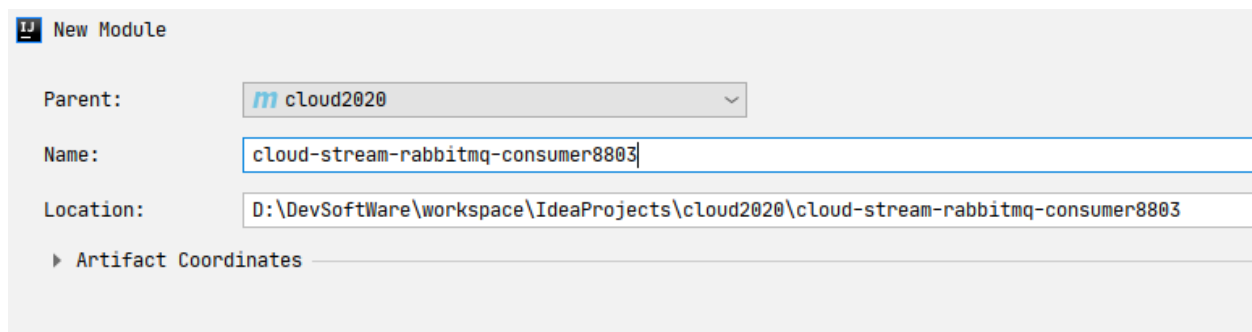


5. 分组消费与持久化

5.1. 依照8802, clone出来一份运行8803

5.1.1. 新建Module

新建cloud-stream-rabbitmq-consumer8803



5.1.2. POM

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/POM/4.0.0"
5     <parent>
6         <artifactId>cloud2020</artifactId>
7         <groupId>cn.sitedev.springcloud</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>cloud-stream-rabbitmq-consumer8803</artifactId>
13
```

```

14     <dependencies>
15         <dependency>
16             <groupId>org.springframework.boot</groupId>
17             <artifactId>spring-boot-starter-web</artifactId>
18         </dependency>
19         <dependency>
20             <groupId>org.springframework.cloud</groupId>
21             <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
22         </dependency>
23         <dependency>
24             <groupId>org.springframework.cloud</groupId>
25             <artifactId>spring-cloud-starter-stream-rabbit</artifactId>
26         </dependency>
27         <dependency>
28             <groupId>org.springframework.boot</groupId>
29             <artifactId>spring-boot-starter-actuator</artifactId>
30         </dependency>
31         <!-- 基础配置 -->
32         <dependency>
33             <groupId>org.springframework.boot</groupId>
34             <artifactId>spring-boot-devtools</artifactId>
35             <scope>runtime</scope>
36             <optional>true</optional>
37         </dependency>
38         <dependency>
39             <groupId>org.projectlombok</groupId>
40             <artifactId>lombok</artifactId>
41             <optional>true</optional>
42         </dependency>
43         <dependency>
44             <groupId>org.springframework.boot</groupId>
45             <artifactId>spring-boot-starter-test</artifactId>
46             <scope>test</scope>
47         </dependency>
48     </dependencies>
49 </project>

```

5.1.3. YML

```

1 server:
2   port: 8803

```

```

3
4 spring:
5   application:
6     name: cloud-stream-consumer
7   cloud:
8     stream:
9       binders: # 在此处配置要绑定的rabbitmq的服务信息;
10      defaultRabbit: # 表示定义的名称, 用于于binding整合
11      type: rabbit # 消息组件类型
12      environment: # 设置rabbitmq的相关的环境配置
13      spring:
14        rabbitmq:
15          host: localhost
16          port: 5672
17          username: guest
18          password: guest
19      bindings: # 服务的整合处理
20      input: # 这个名字是一个通道的名称
21      destination: studyExchange # 表示要使用的Exchange名称定义
22      content-type: application/json # 设置消息类型, 本次为对象json, 如果是文本则
23      binder: defaultRabbit # 设置要绑定的消息服务的具体设置
24      group: sitedev
25
26 eureka:
27   client: # 客户端进行Eureka注册的配置
28     service-url:
29       defaultZone: http://localhost:7001/eureka
30   instance:
31     lease-renewal-interval-in-seconds: 2 # 设置心跳的时间间隔 (默认是30秒)
32     lease-expiration-duration-in-seconds: 5 # 如果现在超过了5秒的间隔 (默认是90秒)
33     instance-id: receive-8803.com # 在信息列表时显示主机名称
34     prefer-ip-address: true # 访问的路径变为IP地址

```

5.1.4. 主启动

```

1 package cn.sitedev.springcloud;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication

```

```

7 public class StreamMQMain8803 {
8     public static void main(String[] args) {
9         SpringApplication.run(StreamMQMain8803.class, args);
10    }
11 }

```

5.1.5. 业务类

5.1.5.1. Controller

```

1 package cn.sitedev.springcloud.controller;
2
3 import org.springframework.beans.factory.annotation.Value;
4 import org.springframework.cloud.stream.annotation.EnableBinding;
5 import org.springframework.cloud.stream.annotation.StreamListener;
6 import org.springframework.cloud.stream.messaging.Sink;
7 import org.springframework.messaging.Message;
8 import org.springframework.stereotype.Component;
9
10
11 @Component
12 @EnableBinding(Sink.class)
13 public class ReceiveMessageListenerController {
14     @Value("${server.port}")
15     private String serverPort;
16
17     @StreamListener(Sink.INPUT)
18     public void input(Message<String> message) {
19         System.out.println("消费者2号,----->接受到的消息: " + message.getPayload() + " ");
20     }
21 }

```

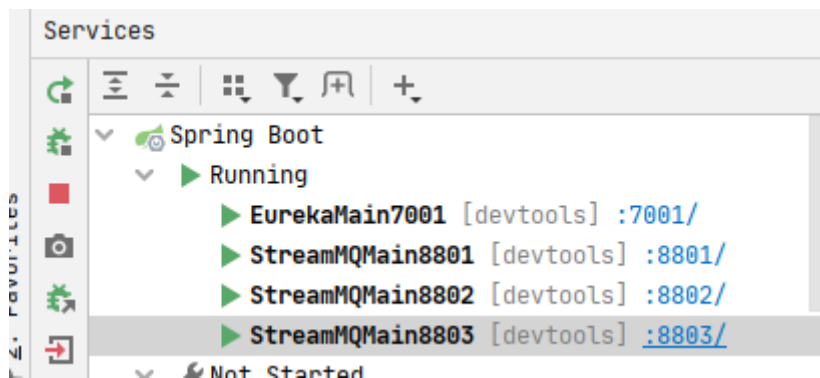
5.2. 启动

服务注册: EurekaMain7001

消息生产: StreamMQMain8801

消息消费: StreamMQMain8802

消息消费: StreamMQMain8803



5.3. 运行后的两个问题

有重复消费问题

消息持久化问题

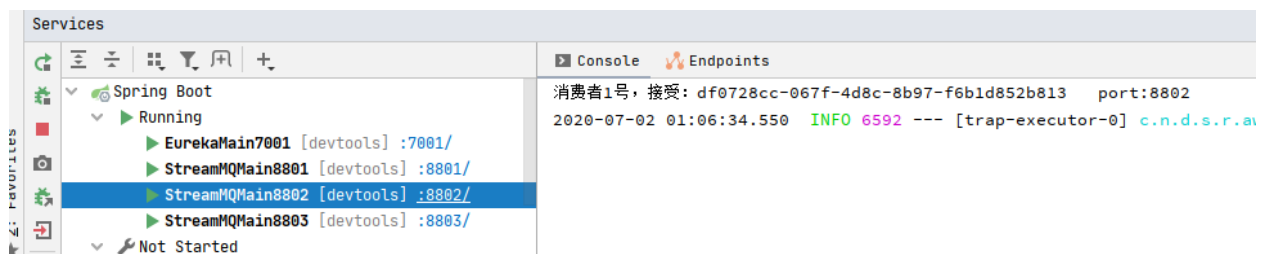
5.4. 消费

目前是8802/8803同时都收到了，存在重复消费问题

5.4.1. 现象

浏览器访问<http://localhost:8801/sendMessage>

查看8802控制台输出:



查看8803控制台输出:



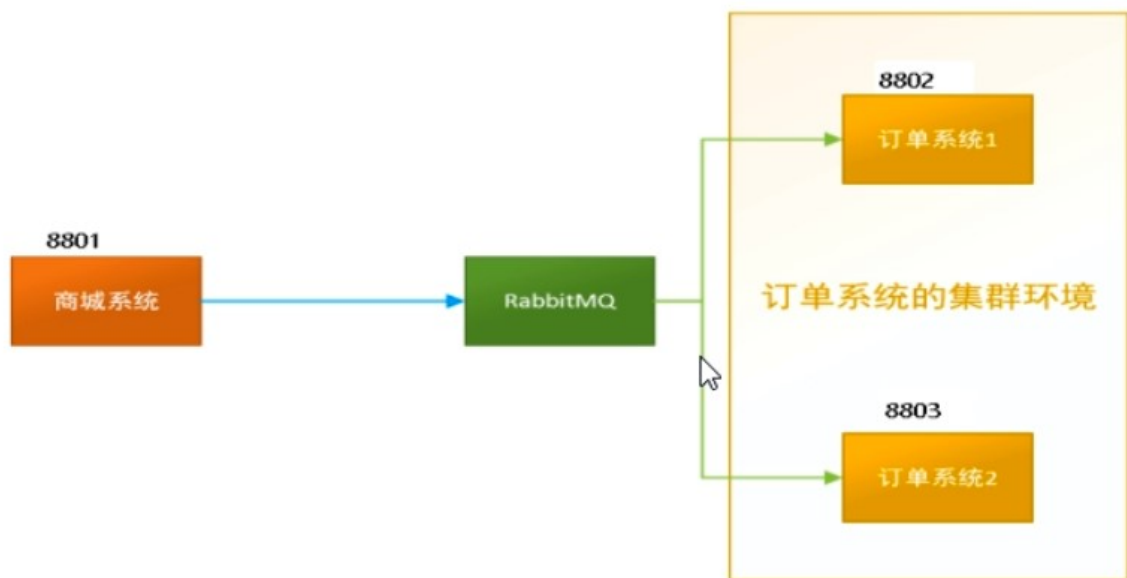
5.4.2. 如何解决

分组和持久化属性group

5.4.3. 生产实际案例

比如在如下场景中，订单系统我们做集群部署，都会从 RabbitMQ中获取订单信息，那如果一个订单同时被两个服务获取到，那么就会造成数据错误，我们得避免这种情况.这时我们就可以使用

Stream中的消息分组来解决



注意在 Stream中处于同一个group中的多个消费者是竞争关系，就能够保证消息只会被其中一个应用消费一次

不同组是可以全面消费的（重复消费）

5.5. 分组

```
studyExchange.anonymous.HZSLT0iFRDyH7fLmfkqP1g 8802
studyExchange.anonymous.NkiopRZNRSa-qGn3Y_5AxA 8803
```

故障现象：重复消费

导致原因：默认分组group是不同的，组流水号不一样，被认为不同组，可以消费

自定义配置分组，

自定义配置分为同一个组，解决

重复消费问题

5.5.1. 原理

微服务应用放置于同一个group中，就能够保证消息只会被其中一个应用消费一次。不同的组是可以消费的，同一个组内会发生竞争关系，只有其中一个可以消费。

5.5.2. 案例演示1(8802/8803都变成不同组，group两个不同)

8802/8803都变成不同组，group两个不同

5.5.2.1. 8002修改YML

修改内容：

```

1 spring:
2   cloud:
3     stream:
4       bindings: # 服务的整合处理
5       input: # 这个名字是一个通道的名称
6       group: sitedevA

```

完整内容:

```

1 server:
2   port: 8802
3
4 spring:
5   application:
6     name: cloud-stream-consumer
7   cloud:
8     stream:
9       binders: # 在此处配置要绑定的rabbitmq的服务信息;
10      defaultRabbit: # 表示定义的名称, 用于于binding整合
11      type: rabbit # 消息组件类型
12      environment: # 设置rabbitmq的相关的环境配置
13      spring:
14        rabbitmq:
15          host: localhost
16          port: 5672
17          username: guest
18          password: guest
19      bindings: # 服务的整合处理
20      input: # 这个名字是一个通道的名称
21      destination: studyExchange # 表示要使用的Exchange名称定义
22      content-type: application/json # 设置消息类型, 本次为json, 文本则设置"text
23      binder: defaultRabbit # 设置要绑定的消息服务的具体设置
24      group: sitedevA
25
26 eureka:
27   client: # 客户端进行Eureka注册的配置
28     service-url:
29       defaultZone: http://localhost:7001/eureka
30   instance:
31     lease-renewal-interval-in-seconds: 2 # 设置心跳的时间间隔 (默认是30秒)
32     lease-expiration-duration-in-seconds: 5 # 如果现在超过了5秒的间隔 (默认是90秒)
33     instance-id: receive-8802.com # 在信息列表时显示主机名称

```

```
34     prefer-ip-address: true      # 访问的路径变为IP地址
```

5.5.2.2. 8803修改YML

修改内容:

```
1  spring:
2    cloud:
3      stream:
4        bindings: # 服务的整合处理
5        input: # 这个名字是一个通道的名称
6        group: sitedevB
```

完整内容:

```
1  server:
2    port: 8803
3
4  spring:
5    application:
6      name: cloud-stream-consumer
7    cloud:
8      stream:
9        binders: # 在此处配置要绑定的rabbitmq的服务信息;
10       defaultRabbit: # 表示定义的名称，用于于binding整合
11       type: rabbit # 消息组件类型
12       environment: # 设置rabbitmq的相关的环境配置
13       spring:
14         rabbitmq:
15           host: localhost
16           port: 5672
17           username: guest
18           password: guest
19       bindings: # 服务的整合处理
20       input: # 这个名字是一个通道的名称
21       destination: studyExchange # 表示要使用的Exchange名称定义
22       content-type: application/json # 设置消息类型，本次为对象json，如果是文本则用text/plain
23       binder: defaultRabbit # 设置要绑定的消息服务的具体设置
24       group: sitedevB
25
26  eureka:
```

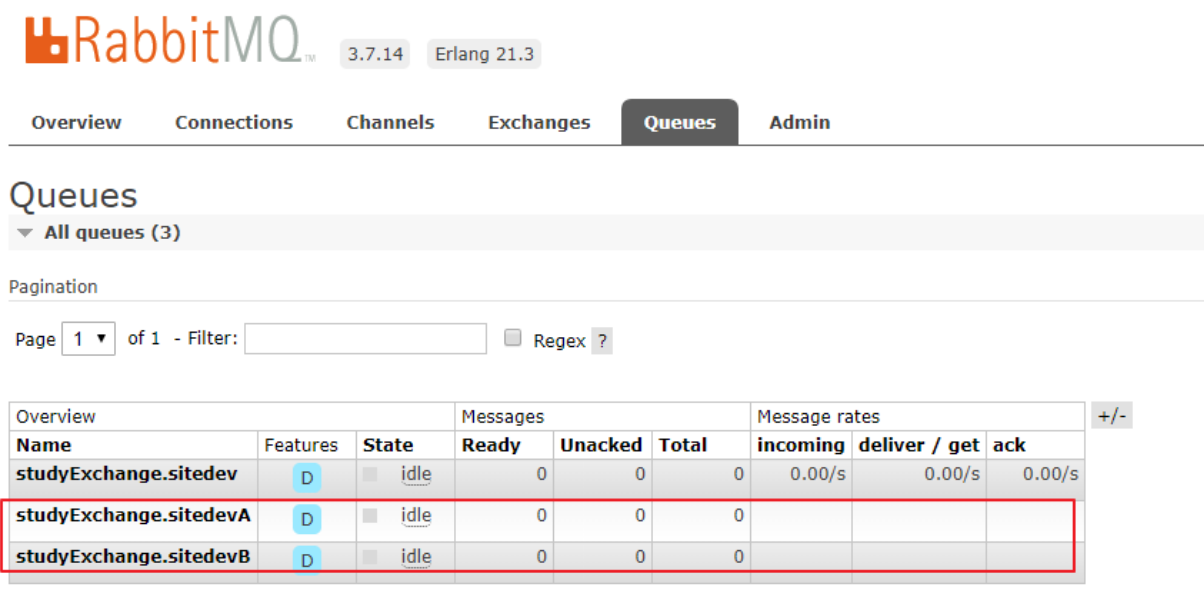
```

27 client: # 客户端进行Eureka注册的配置
28     service-url:
29         defaultZone: http://localhost:7001/eureka
30     instance:
31         lease-renewal-interval-in-seconds: 2 # 设置心跳的时间间隔（默认是30秒）
32         lease-expiration-duration-in-seconds: 5 # 如果现在超过了5秒的间隔（默认是90秒）
33         instance-id: receive-8803.com # 在信息列表时显示主机名称
34         prefer-ip-address: true # 访问的路径变为IP地址

```

5.5.2.3. 测试

浏览器访问RabbitMQ管理页面, 查看Queues: <http://localhost:15672/#/queues>



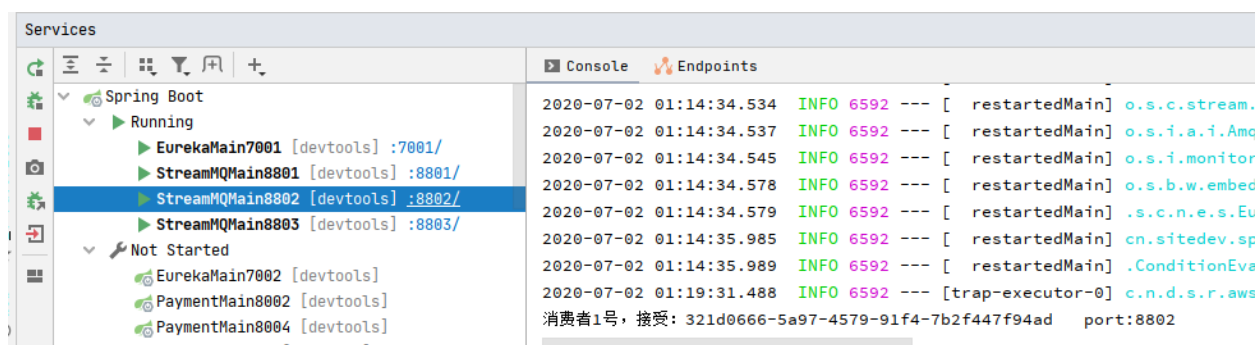
The screenshot shows the RabbitMQ Management UI. The 'Queues' tab is selected. Under 'All queues (3)', there is a table listing three queues. The first two queues, 'studyExchange.sitedevA' and 'studyExchange.sitedevB', are highlighted with a red border. The table has columns for Name, Features, State, Messages (Ready, Unacked, Total), and Message rates (incoming, deliver / get, ack).

Overview			Messages			Message rates			
Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
studyExchange.sitedev	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
studyExchange.sitedevA	D	idle	0	0	0				
studyExchange.sitedevB	D	idle	0	0	0				

可以看到有2个分组

浏览器访问<http://localhost:8801/sendMessage> 发送一条消息

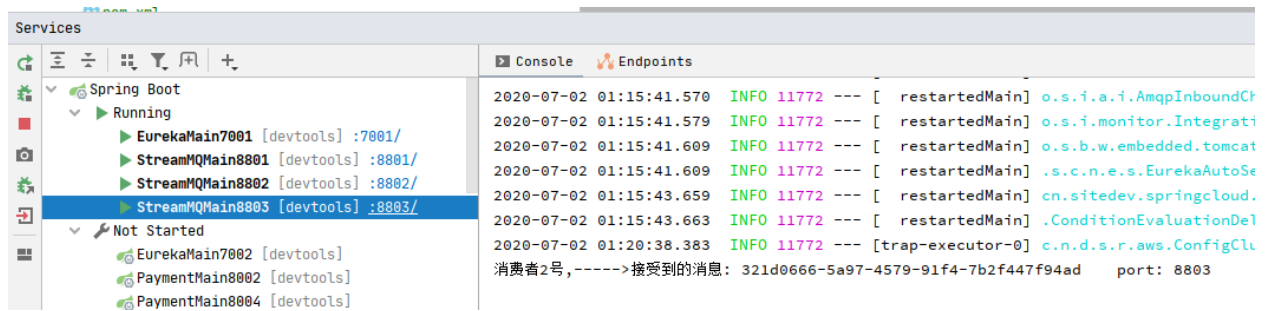
查看8802控制台输出:



The screenshot shows the Spring Boot Services console output. The 'StreamMQMain8802' service is selected. The console output shows several log messages indicating that the service has restarted and is now running. The messages include timestamps, log levels (INFO), and details about the service's state and endpoints.

Timestamp	Log Level	Message	Endpoint
2020-07-02 01:14:34.534	INFO 6592	[restartedMain]	o.s.c.stream.
2020-07-02 01:14:34.537	INFO 6592	[restartedMain]	o.s.i.a.i.Amq
2020-07-02 01:14:34.545	INFO 6592	[restartedMain]	o.s.i.monitor
2020-07-02 01:14:34.578	INFO 6592	[restartedMain]	o.s.b.w.embed
2020-07-02 01:14:34.579	INFO 6592	[restartedMain]	.s.c.n.e.s.Eu
2020-07-02 01:14:35.985	INFO 6592	[restartedMain]	cn.sitedev.sp
2020-07-02 01:14:35.989	INFO 6592	[restartedMain]	.ConditionEvs
2020-07-02 01:19:31.488	INFO 6592	[trap-executor-0]	c.n.d.s.r.aws

查看8803控制台输出:



可以看到, 8802和8803都收到了消息

5.5.3. 结论1(8802/8803都变成不同组, group两个不同)

8802/8803都变成不同组, group两个不同, 还是会存在重复消费的情况

5.5.4. 案例演示2(8802/8803都变成相同组, group两个相同)

5.5.4.1. 8002修改YML

修改内容:

```
1 spring:
2   cloud:
3     stream:
4       bindings: # 服务的整合处理
5       input: # 这个名字是一个通道的名称
6       group: sitedevC
```

完整内容:

```
1 server:
2   port: 8802
3
4 spring:
5   application:
6     name: cloud-stream-consumer
7   cloud:
8     stream:
9       binders: # 在此处配置要绑定的rabbitmq的服务信息;
10      defaultRabbit: # 表示定义的名称, 用于于binding整合
11      type: rabbit # 消息组件类型
12      environment: # 设置rabbitmq的相关的环境配置
13      spring:
14      rabbitmq:
```

```

15         host: localhost
16         port: 5672
17         username: guest
18         password: guest
19     bindings: # 服务的整合处理
20         input: # 这个名字是一个通道的名称
21             destination: studyExchange # 表示要使用的Exchange名称定义
22             content-type: application/json # 设置消息类型，本次为json，文本则设置"text
23             binder: defaultRabbit # 设置要绑定的消息服务的具体设置
24             group: sitedevC
25
26 eureka:
27     client: # 客户端进行Eureka注册的配置
28         service-url:
29             defaultZone: http://localhost:7001/eureka
30     instance:
31         lease-renewal-interval-in-seconds: 2 # 设置心跳的时间间隔（默认是30秒）
32         lease-expiration-duration-in-seconds: 5 # 如果现在超过了5秒的间隔（默认是90秒）
33         instance-id: receive-8802.com # 在信息列表时显示主机名称
34         prefer-ip-address: true # 访问的路径变为IP地址

```

5.5.4.2. 8803修改YML

修改内容:

```

1 spring:
2     cloud:
3         stream:
4             bindings: # 服务的整合处理
5                 input: # 这个名字是一个通道的名称
6                     group: sitedevC

```

完整内容:

```

1 server:
2     port: 8803
3
4 spring:
5     application:
6         name: cloud-stream-consumer
7     cloud:

```

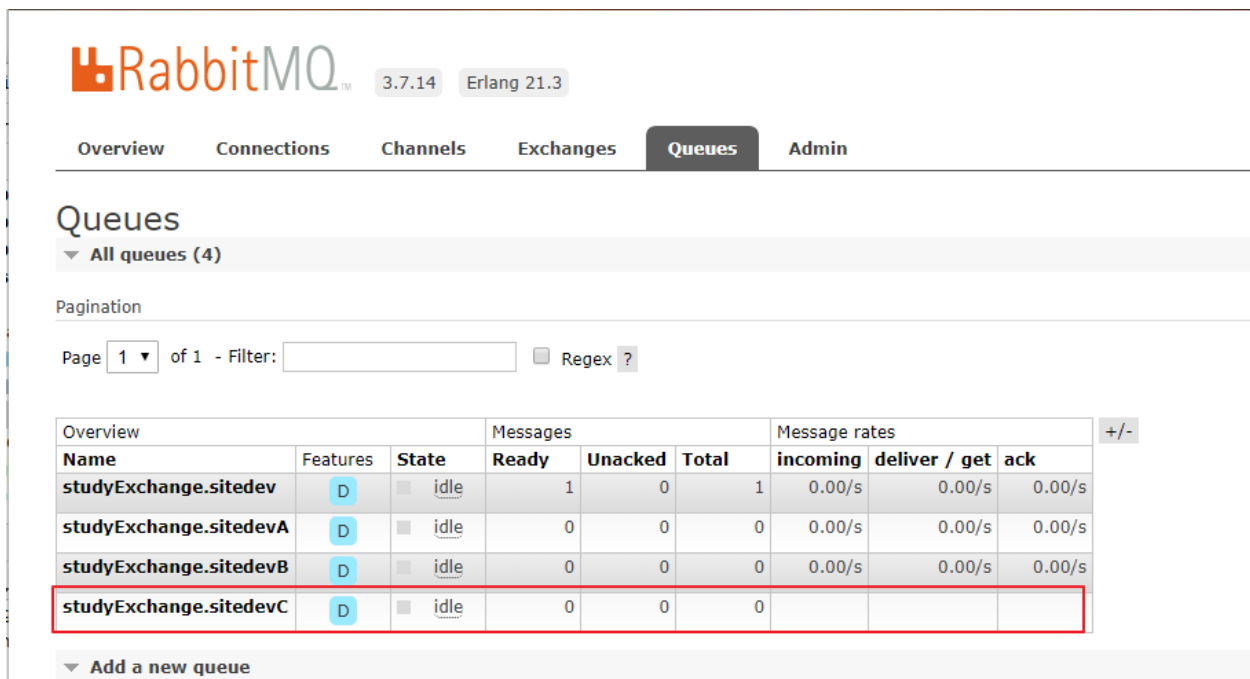
```

8      stream:
9          binders: # 在此处配置要绑定的rabbitmq的服务信息:
10             defaultRabbit: # 表示定义的名称, 用于于binding整合
11                 type: rabbit # 消息组件类型
12                 environment: # 设置rabbitmq的相关的环境配置
13                 spring:
14                     rabbitmq:
15                         host: localhost
16                         port: 5672
17                         username: guest
18                         password: guest
19             bindings: # 服务的整合处理
20                 input: # 这个名字是一个通道的名称
21                     destination: studyExchange # 表示要使用的Exchange名称定义
22                     content-type: application/json # 设置消息类型, 本次为对象json, 如果是文本则
23                     binder: defaultRabbit # 设置要绑定的消息服务的具体设置
24                     group: sitedevC
25
26 eureka:
27     client: # 客户端进行Eureka注册的配置
28         service-url:
29             defaultZone: http://localhost:7001/eureka
30     instance:
31         lease-renewal-interval-in-seconds: 2 # 设置心跳的时间间隔 (默认是30秒)
32         lease-expiration-duration-in-seconds: 5 # 如果现在超过了5秒的间隔 (默认是90秒)
33         instance-id: receive-8803.com # 在信息列表时显示主机名称
34         prefer-ip-address: true # 访问的路径变为IP地址

```

5.5.4.3. 测试

浏览器访问RabbitMQ管理页面, 查看Queues: <http://localhost:15672/#/queues>



RabbitMQ 3.7.14 Erlang 21.3

Overview Connections Channels Exchanges **Queues** Admin

Queues

▼ All queues (4)

Pagination

Page 1 of 1 - Filter: ☐ Regex ?

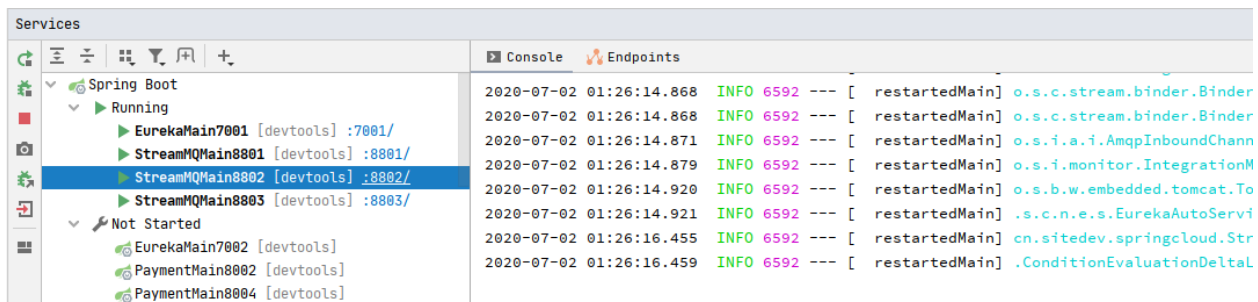
Overview			Messages			Message rates				+/-
Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack		
studyExchange.sitedev	D	idle	1	0	1	0.00/s	0.00/s	0.00/s		
studyExchange.sitedevA	D	idle	0	0	0	0.00/s	0.00/s	0.00/s		
studyExchange.sitedevB	D	idle	0	0	0	0.00/s	0.00/s	0.00/s		
studyExchange.sitedevC	D	idle	0	0	0					

▼ Add a new queue

可以看到有1个分组

浏览器访问<http://localhost:8801/sendMessage> 发送一条消息

查看8802控制台输出:



Services

Spring Boot

- Running
 - EurekaMain7001 [devtools] :7001/
 - StreamMQMain8801 [devtools] :8801/
 - StreamMQMain8802 [devtools] :8802/
 - StreamMQMain8803 [devtools] :8803/
- Not Started
 - EurekaMain7002 [devtools]
 - PaymentMain8802 [devtools]
 - PaymentMain8804 [devtools]

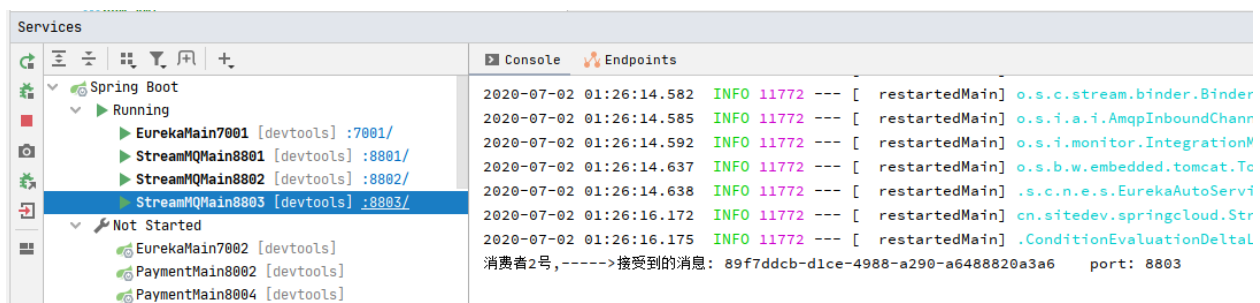
Console

```

2020-07-02 01:26:14.868 INFO 6592 --- [ restartedMain] o.s.c.stream.binder.Binder
2020-07-02 01:26:14.868 INFO 6592 --- [ restartedMain] o.s.c.stream.binder.Binder
2020-07-02 01:26:14.871 INFO 6592 --- [ restartedMain] o.s.i.a.i.AmqpInboundChann
2020-07-02 01:26:14.879 INFO 6592 --- [ restartedMain] o.s.i.monitor.IntegrationM
2020-07-02 01:26:14.920 INFO 6592 --- [ restartedMain] o.s.b.w.embedded.tomcat.To
2020-07-02 01:26:14.921 INFO 6592 --- [ restartedMain] .s.c.n.e.s.EurekaAutoServi
2020-07-02 01:26:16.455 INFO 6592 --- [ restartedMain] cn.sitedev.springcloud.Str
2020-07-02 01:26:16.459 INFO 6592 --- [ restartedMain] .ConditionEvaluationDeltal

```

查看8803控制台输出:



Services

Spring Boot

- Running
 - EurekaMain7001 [devtools] :7001/
 - StreamMQMain8801 [devtools] :8801/
 - StreamMQMain8802 [devtools] :8802/
 - StreamMQMain8803 [devtools] :8803/
- Not Started
 - EurekaMain7002 [devtools]
 - PaymentMain8802 [devtools]
 - PaymentMain8804 [devtools]

Console

```

2020-07-02 01:26:14.582 INFO 11772 --- [ restartedMain] o.s.c.stream.binder.Binder
2020-07-02 01:26:14.585 INFO 11772 --- [ restartedMain] o.s.i.a.i.AmqpInboundChanr
2020-07-02 01:26:14.592 INFO 11772 --- [ restartedMain] o.s.i.monitor.IntegrationM
2020-07-02 01:26:14.637 INFO 11772 --- [ restartedMain] o.s.b.w.embedded.tomcat.Tc
2020-07-02 01:26:14.638 INFO 11772 --- [ restartedMain] .s.c.n.e.s.EurekaAutoServi
2020-07-02 01:26:16.172 INFO 11772 --- [ restartedMain] cn.sitedev.springcloud.Str
2020-07-02 01:26:16.175 INFO 11772 --- [ restartedMain] .ConditionEvaluationDeltal
消费者2号,---->接受到的消息: 89f7ddcb-d1ce-4988-a290-a6488820a3a6 port: 8803

```

可以看到, 只有8803收到了消息

5.5.5. 结论2(8802/8803都变成相同组, group两个相同)

同一个组的多个微服务实例, 每次只会有一个拿到

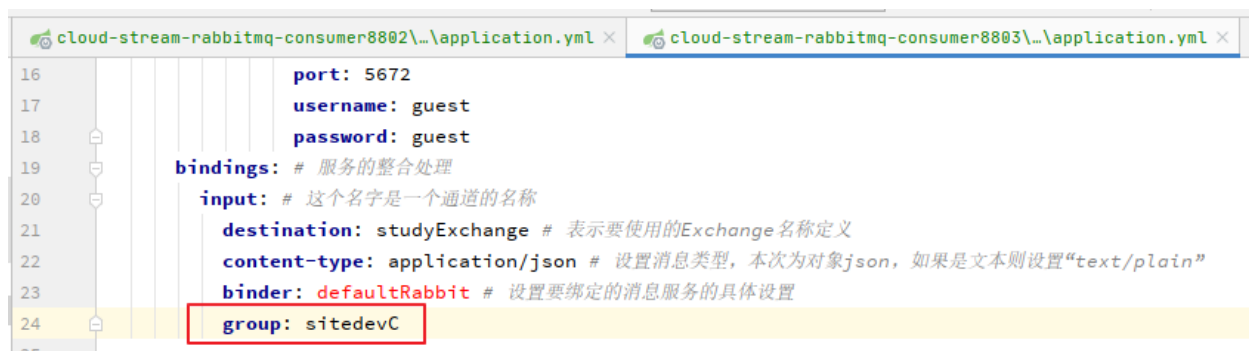
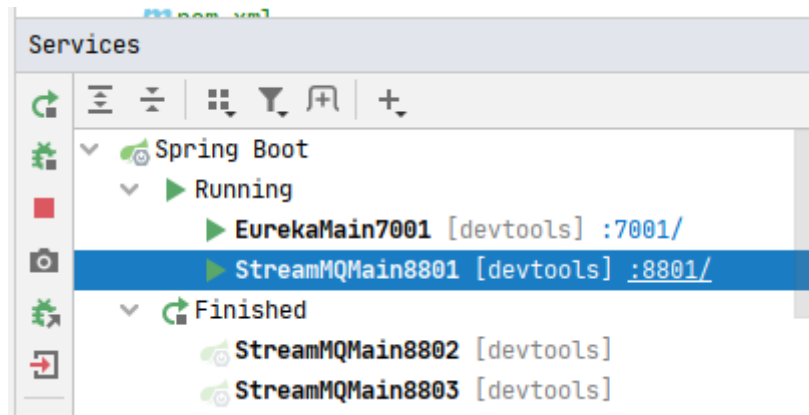
8802/8803实现了轮询分组, 每次只有一个消费者 8801模块的发的消息只能被8802或8803其中一个接收到, 这样避免了重复消费

5.6. 持久化

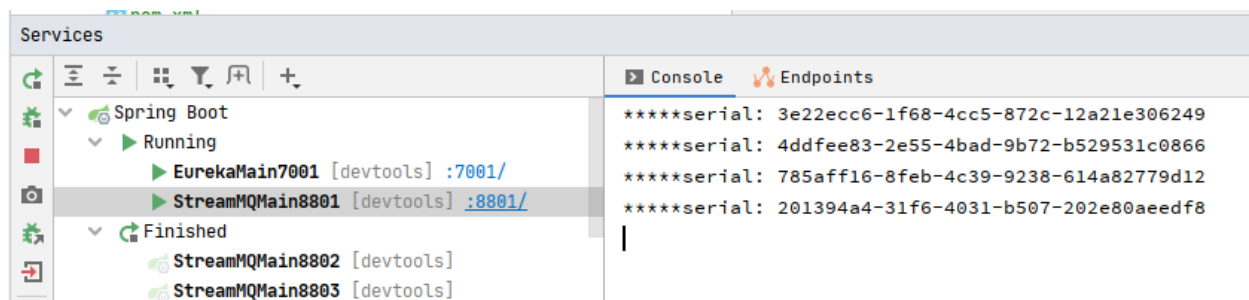
通过上述，解决了重复消费问题，再看看持久化

5.6.1. 案例演示

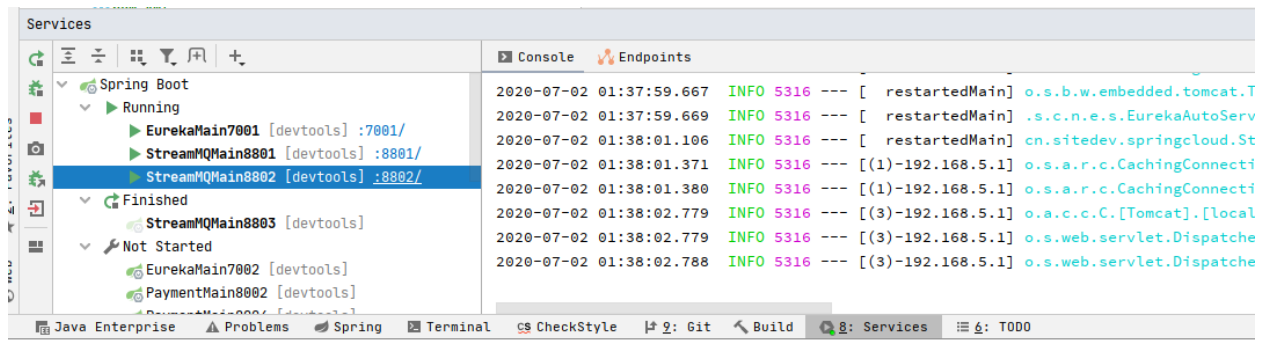
1) 停止8802/8803并去除掉8802的分组group: sitedevC, 但是不删除8803的分组group: sitedevC



2) 8801先发送4条信息到RabbitMQ (浏览器访问4次 <http://localhost:8801/sendMessage>)



3) 先启动8802，无分组属性配置，后台没有打出来消息



4) 再启动8803，有分组属性配置，后台打出来了MQ上的消息

