**Spring Cloud Stream框架简介**

Spring Cloud Stream框架是一个用于构建**消息驱动微服务**的框架(在下面方便起见也叫它Stream框架)，该框架在Spring Boot的基础上整合了Spring Integration来连接消息代理中间件（RabbitMQ，Kafka等）。它支持多个消息中间件的自定义配置，同时吸收了这些消息中间件的部分概念，例如持久化订阅、消费者分组，和分区等概念。使用Stream框架，**我们不必关系如何连接各个消息代理中间件**，也**不必关系消息的发送与接收**，只需要**进行简单的配置**就可以实现这些功能了，可以让我们更敏捷的进行开发主体业务逻辑了。

Spring Cloud Stream框架的组成部分：

1. Stream框架自己的应用模型；

2. 绑定器抽象层，可以与消息代理中间件进行绑定，通过绑定器的API，可实现插件式的绑定器。

3. 持久化订阅的支持。

4. 消费者组的支持。

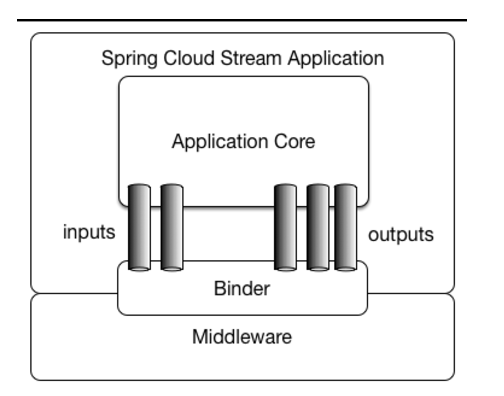
5. Topic分区的支持。

Spring Cloud Stream在生产者和消费者之间加入了一个类似代理的角色，它直接与消息代理中间件进行交互，消息生产者和消费者不需要直接调用各个消息代理中间件的API，它们甚至感觉不到消息中间件的存在，这样就降低了和消息中间件产品之间的依赖，我们可以很轻松的选择换用那个消息中间件。

**我们来看看Spring Cloud Stream官方文档上面描述的一些概念：**

1. 应用模型：

应用程序通过inputs和outputs来与Spring cloud stream中的Binder交互，通过我们的配置来绑定inputs和outputs，而Binder又和消息代理中间件进行交互。所以我们只需要搞清楚如何与Stream框架交互就可以很轻松的使用消息驱动的方式。

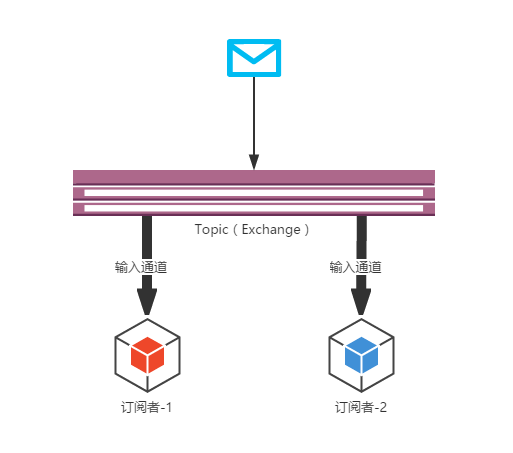


2. 抽象绑定器

Spring Cloud Stream通过抽象绑定器来绑定消息代理中间件，Stream目前提供了RabbitMQ和Kafka的Binder实现，也包括了一个TestSupportBinder，用于测试，你甚至可以根据这些API去写自己的Binder。所有Spring Cloud的框架都是基于Spring Boot的，所以Spring Cloud Stream通用使用Spring Boot的自动配置方式，我们可以再application.py 或者application.properties中指定使用哪个Binder实现，不需要修改我们的业务代码。

3. 发布-订阅

下图是一个经典的发布订阅模型，生产者消息发布在一个shared topic（主题）上，消费者通过订阅这个topic来获取消息。

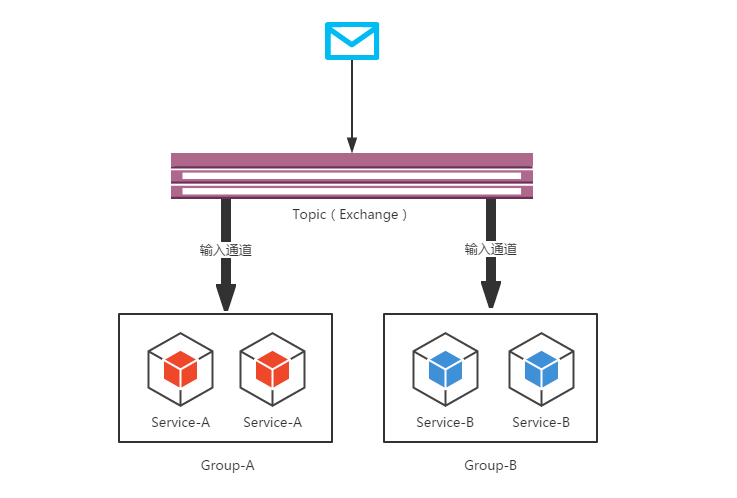


上图中的Topic对应Spring Cloud Stream中的destinations，这个在RabbitMQ中叫做Exchange, Kafka中叫Topic。

4. 消费组（Consumer Group）

尽管通过上面的Topic，可以直接实现生产到消费这一过程，但是有很多应用场景我们需要进一步细分，比如我们想一个应用中只有一个实例来消费某条消息，这个时候就可以使用消费组。**当多个消费者放置在一个消费组中，这个组里面只有一个消费者可以消费消息。当多个消费者放在不同的消费组中，每个消费组都会收到一条消息，但每个消费组都只有一个消费者可以消费这个消息。**

设置消费组的配置为**spring.cloud.stream.bindings.<channelName>.group**



如上图所示，所有订阅Topic的消费组都会收到消息的一个备份，每个组中只有一个成员会收到消息。如果没有指定消费组，那么默认会为应用分配一个匿名消费组。 如果配置了多个组， 并且没有为input指定使用哪个组，那么默认会为该应用分配一个匿名消费者组，与所有其它组处于 订阅-发布 关系中。ps:也就是说如果管道没有指定消费组，那么这个匿名消费组会与其它组一起消费消息，出现了重复消费的问题。

*NOTE：destinations有点类似于RabbitMQ中的交换器Exchange，而消费组有点类似消息队列Queue。*

5. 消费者类型

1）支持有两种消费者类型：

Message-driven (消息驱动型，有时简称为异步)

Polled (轮询型，有时简称为 同步)

在Spring Cloud 2.0版本前只支持 Message-driven这种异步类型的消费者，消息一旦可用就会传递，并且有一个线程可以处理它；当你想控制消息的处理速度时，可能需要用到同步消费者类型。

2）持久化

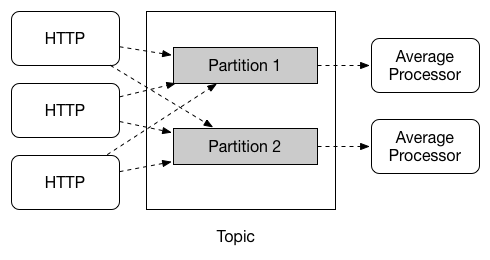
一般来说所有拥有订阅主题的消费组都是持久化的，除了匿名消费组。 Binder的实现确保了所有订阅关系的消费订阅是持久的，一个消费组中至少有一个订阅了主题，那么被订阅主题的消息就会进入这个组中，无论组内是否停止。

注意： 匿名订阅本身是非持久化的，但是有一些Binder的实现（比如RabbitMQ）则可以创建非持久化的组订阅

通常情况下，当有一个应用绑定到目的地的时候，最好指定消费消费组。扩展Spring Cloud Stream应用程序时，必须为每个输入绑定指定一个使用者组。这样做可以防止应用程序的实例接收重复的消息（除非需要这种行为，这是不寻常的）。

6. 分区（Partitioning）

在消费组中我们可以保证消息不会被重复消费，但是在同组下有多个实例的时候，我们无法确定每次处理消息的是不是被同一消费者消费，分区的作用就是为了确保具有共同特征标识的数据由同一个消费者实例进行处理，当然前边的例子是狭义的，通信代理（broken topic）也可以被理解为进行了同样的分区划分。Spring Cloud Stream 的分区概念是抽象的，可以为不支持分区Binder实现（例如RabbitMQ）也可以使用分区。



注意：要使用分区处理，你必须同时对生产者和消费者进行配置。

代码实战

只需⑤步，轻松带你走进Spring Cloud Stream和RabbitMQ的集成。

1. 打开IDE，**导入或创建Spring Cloud项目**，需要源代码的盆友请前往github:

<https://github.com/aharddreamer/chendong/tree/master/springcloud/stream-rabbitmq-CSDN>/

PS：Github上面这个演示项目里面会有些其他与stream-rabbit无关的配置或代码，请忽略。

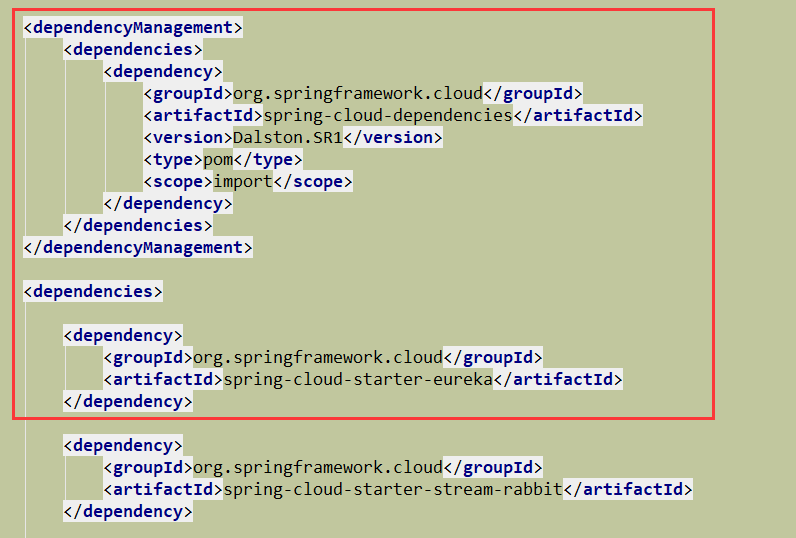
启动的时候，先启动eureka-server，再启动first-service-provider即可。这里我们只需要修改first-service-provider这个模块。

2. 按照Spring Cloud项目的通性，加入stream-rabbit的dependency：

在first-service-provider的POM.xml中加入这个dependency：

<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-stream-rabbit</**artifactId**>  
</**dependency**>

你可以看到POM里面还有下面这些dependencies，这都是Spring Cloud需要的环境，这次我不演示这些玩意儿，需要了解的哥们儿请看之前的博客。



3. 在application.properties或py中加入炒鸡简单的rabbitmq配置：

**spring.rabbitmq.host**=**localhost  
spring.rabbitmq.port**=**5672  
spring.rabbitmq.username**=**guest  
spring.rabbitmq.password**=**guest**

（这几乎都是默认配置了，你甚至可以试试不用配。当然，要往更深层次的发展，肯定要定义很多东西，比如使用多少个消费者，定义消费组，定义destinations等）

4. 编写一些代码，使用生产者发送消息，然后使用消费者接收并处理消息。

首先创建一个接口，在里面定义生产者和消费者：

**public interface** MyStreamClient {  
  
 String ***INPUT*** = **"input"**;  
 String ***OUTPUT*** = **"output"**;  
  
 @Input(***INPUT***)  
 SubscribableChannel input();  
  
 @Output(***OUTPUT***)  
 MessageChannel output();  
}

注意看这两个家伙，input很明显是输入的意思，这里的输入不是你向rabbitMQ输入，而是rabbitMQ的消息向你输入，所以这个input肯定是消费者的角色，Subscribe就是订阅的意思，这是订阅通道。那么对应的output就是你向rabbitMQ输出消息，那么你肯定就是消息生产者。

接下来，再创建一个类，来监听和处理@Input通道进来的消息 （消费消息）。

@Component  
@EnableBinding(MyStreamClient.**class**)  
**public class** MyStreamReceiver {  
  
 @StreamListener(MyStreamClient.***INPUT***)  
 **public void** receiver(String message) {  
 System.***out***.println(**"接收到的消息："** + message);  
 }  
}

那么，我们哪来的消息消费呢，答对了，我们要创建一个测试的入口，在里面去生产一条消息，丢给生产者。

@RestController  
**public class** TestController {  
  
 @Autowired  
 **private** MyStreamClient **myStreamClient**;  
  
 @RequestMapping(**"/sendmsg"**)  
 **public** String sendMessage() {  
 Message message = MessageBuilder.*withPayload*(**"This is a test message"**).build();  
 **myStreamClient**.output().send(message);  
 **return "SEND SUCCESSFUL"**;  
 }  
}

5. 写完之后你是不是迫不及待想跑一把？别急，这样你肯定就犯了我之前犯的错误，还搞了半天才明白。这样生产者发出去，消费者却一直收不到，相反，如果把@StreamListener里面的参数换成output,倒是可以收到了。这是为啥？因为生产者和消费者没有连通，之所以监听output能收到消息是因为这个路线是直接生产者的一端到另一端，中间没有配置任何其他路线。

现在我们要把生产者和消费者用最简单的路线连接起来，那么就是配置一下destinations，直接指定output这个管道的目的地为input（消费端），当然肯定在他们中间可以规划更多的路线，甚至配置消费组，毕竟这是AMQP与身俱来的特性。

在application.properties或py中加入以下配置：

**spring.cloud.stream.bindings.output.destination**=**input**

这样监听@Input通道就能获取到生产者（@Output）发送过来的消息啦！

赶紧跑起来走一个！

。。。。。。

。。。。。

。。。。

。。。

。。

。

噢噢！！对了，很遗憾的提醒您跑不起来的…….别打我……赶紧在电脑上先安装RabbitMQ（下载页面：<https://www.rabbitmq.com/download.html> ），安装完成之后看看服务里面是不是启动了RabbitMQ Server，没启动的话手动吧它启动一下，然后你可以在浏览器输入localhost:15672 看看是否能进入rabbitMQ的控制平台页面。

参考书籍或链接：

《疯狂Spring Cloud微服务架构实战》

“Spring Cloud （十五）Stream 入门、主要概念与自定义消息发送与接收”

<https://www.cnblogs.com/hellxz/p/9396282.html>