**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。

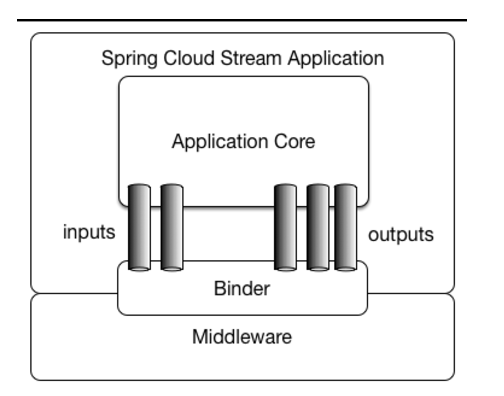
5. Topic分区的支持。

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。

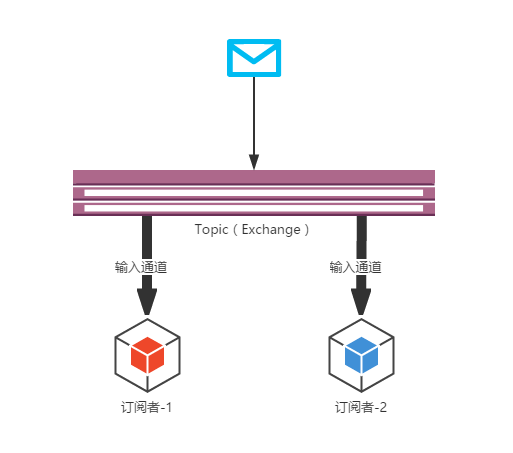


2. 抽象绑定器

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。

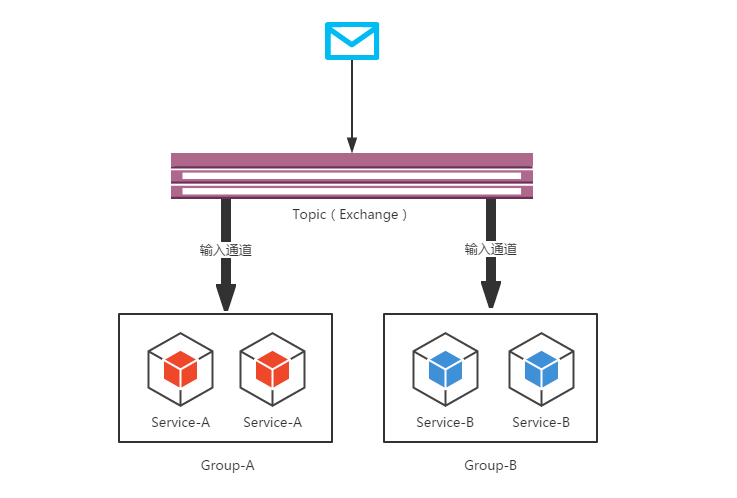


上图中的Topic对应Spring Cloud Stream中的destinations，这个在RabbitMQ中叫做Exchange, Kafka中叫Topic。

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

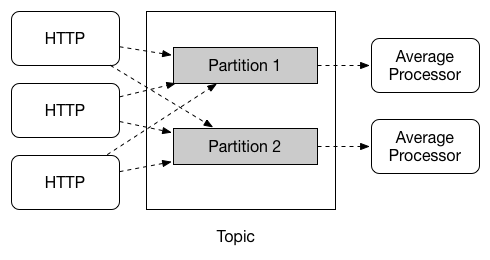
一般来说所有拥有订阅主题的消费组都是持久化的，除了匿名消费组。 Binder的实现确保了所有订阅关系的消费订阅是持久的，一个消费组中至少有一个订阅了主题，那么被订阅主题的消息就会进入这个组中，无论组内是否停止。

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。

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>