# 一站式微服务架构Spring Cloud

主讲：Cat 老师

北京动力节点教育科技有限公司  
2019 • 北京

动力节点•版权所有•禁止传播

# 快速回顾

1. 分布式与微服务架构的理论梳理；
2. 什么是Spring Cloud？
3. Spring Cloud的整体架构 （与Dubbo比较）
4. 服务消费者Controller直连调用服务提供者Controller（http协议的restful）
5. Spring Cloud的注册中心Eureka
6. Spring Cloud Eureka与Zookeeper比较
7. Spring Cloud Eureka高可用集群
8. Spring Cloud Eureka自我保护机制
9. Spring Cloud Ribbon负载均衡
10. Spring Cloud Feign声明式服务调用（与Dubbo接口层比较）
11. Spring Cloud Hystrix 服务熔断降级、服务限流
12. Spring Cloud Hystrix DashBoard仪表盘监控
13. Spring Cloud Hystrix Turbine聚合监控
14. Spring Cloud Zuul网关（路由、过滤、异常、降级）
15. Spring Cloud Config配置中心（用途、使用、加解密）
16. Spring Cloud Config 配置中心（自动刷新、高可用、安全认证）
17. Spring Cloud Sleuth 分布式链路跟踪
18. Spring Cloud Sleuth 整合Zipkin 分布式链路跟踪

# 本次课主题大纲

1. Spring Cloud Stream 概述
2. Spring Cloud Stream 基础
3. Spring Cloud Stream 消息通道
4. Spring Cloud Stream 分组
5. Spring Cloud Stream 消息路由

# Spring Cloud Stream

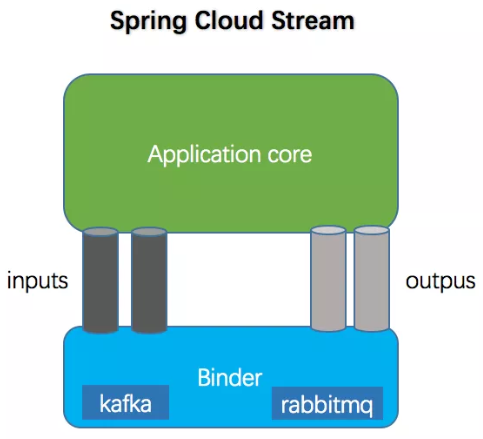
## Spring Cloud Stream概述

在微服务的开发过程中，可能会经常用到消息中间件，通过消息中间件在服务与服务之间传递消息，不管你使用的是哪款消息中间件，比如RabbitMQ还是Kafka，那么消息中间件和服务之间都有一点耦合性，这个耦合性就是指如果我原来使用的RabbitMQ，现在要替换为Kafka，那么我们的微服务都需要修改，变动会比较大，因为这两款消息中间件有一些区别，如果我们使用Spring Cloud Stream来整合我们的消息中间件，那么这样就可以降低微服务和消息中间件的耦合性，做到轻松在不同消息中间件间切换，当然目前Spring Cloud Stream只支持rabbitmq 和 kafka；

按照官方的定义，Spring Cloud Stream 是一个构建消息驱动微服务的框架。

Spring Cloud Stream解决了开发人员无感知的使用消息中间件的问题，因为Spring Cloud Stream对消息中间件的进一步封装，可以做到代码层面对消息中间件的无感知，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；

## Spring Cloud Stream 重要概念



应用程序通过input（相当于消费者consumer）、output（相当于生产者producer）来与Spring Cloud Stream中Binder交互，而Binder负责与消息中间件交互，因此，我们只需关注如何与Binder交互即可，而无需关注与具体消息中间件的交互。

|  |  |
| --- | --- |
| **组成** | **说明** |
| Binder | Binder是应用与消息中间件之间的封装，目前实现了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 | 该注解标识输入通道，通过该输入通道接收消息进入应用程序 |
| @Output | 该注解标识输出通道，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 | 将信道channel和exchange绑定在一起 |

下面我们看一个具体案例，来理解上面的几个概念；

## Spring Cloud Stream应用

消息生产者

1. 创建SpringBoot应用34-springcloud-service-stream；
2. 添加依赖：

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

3、配置文件

*#对接具体的消息中间件***spring.cloud.stream.binders.rabbitmq.type**=**rabbit  
spring.cloud.stream.binders.rabbitmq.environment.spring.rabbitmq.host**=**192.168.10.128  
spring.cloud.stream.binders.rabbitmq.environment.spring.rabbitmq.port**=**5672  
spring.cloud.stream.binders.rabbitmq.environment.spring.rabbitmq.username**=**guest  
spring.cloud.stream.binders.rabbitmq.environment.spring.rabbitmq.password**=**guest  
spring.cloud.stream.binders.rabbitmq.environment.spring.rabbitmq.virtual-host**=**/**

*#消息生产者  
#其中utput是一个key，这个名字是一个通道的名称，在代码中会用到  
#destination表示要使用的Exchange名称定义***spring.cloud.stream.bindings.output.destination**=**spring.cloud.stream.exchange***#设置要绑定的消息服务的binder***spring.cloud.stream.bindings.output.binder**=**rabbitmq**

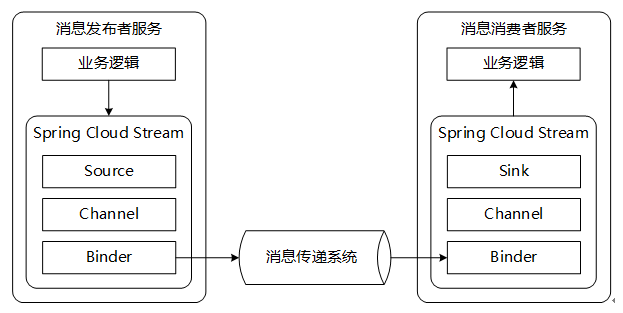
**消息发送：**

@EnableBinding(Source.**class**)  
**public class** MessageSender {@Autowired  
 **private** MessageChannel **output**; *// 消息的发送管道* **public void** publish(String msg) {  
 **output**.send(MessageBuilder.*withPayload*(msg).build());  
 System.***out***.println(**"消息发送：<"** + msg + **"> 完成，时间："** + **new** Date());  
 }  
}

**消息接收：**

*#配置消息消费者  
#指定交换机***spring.cloud.stream.bindings.input.destination**=**spring.cloud.stream.exchange***#设置要绑定的消息服务的binder***spring.cloud.stream.bindings.input.binder**=**rabbitmq**

@EnableBinding(Sink.**class**)  
**public class** MessageReceiver {  
  
 @StreamListener(Sink.***INPUT***)  
 **public void** input(Message message) {  
 System.***out***.println(**"消息接收：<"** + message.getPayload() + **"> 完成，时间："** + **new** Date());  
 }  
}



## Spring Cloud Stream自定义消息通道

在前面的案例中，我们已经实现了一个基础的 Spring Cloud Stream 消息传递处理操作，但在操作之中使用的是系统提供的 Source （output）、Sink（input），接下来我们来看一下自定义通道名称；

**public interface** MessageSource {  
  
 *//channel名称* String ***OUTPUT*** = **"myOutput"**;  
  
 @Output(***OUTPUT***)  
 MessageChannel output();  
}

**public interface** MessageSink {  
  
 String ***INPUT*** = **"myInput"**;  
  
 @Input(Sink.***INPUT***)  
 SubscribableChannel input();  
}

*#配置消息消费者  
#指定交换机***spring.cloud.stream.bindings.myInput.destination**=**spring.cloud.stream.exchange** *#设置要绑定的消息服务的binder***spring.cloud.stream.bindings.myInput.binder**=**rabbitmq**

*#消息生产者  
#其中utput是一个key，这个名字是一个通道的名称，在代码中会用到  
#destination表示要使用的Exchange名称定义***spring.cloud.stream.bindings.myOutput.destination**=**spring.cloud.stream.exchange** *#设置要绑定的消息服务的binder***spring.cloud.stream.bindings.myOutput.binder**=**rabbitmq**

## Spring Cloud Stream分组与持久化

前面的案例我们成功地实现了消息的发送与接收，但是所发送的消息在默认情况下都属于一种临时消息，也就是说如果没有消费者进行消费处理，那么该消息是不会被永久保留，可能会造成消息的丢失；

如果要实现持久化消息，需要在消息的消费端配置一个分组，有分组就表示该消息可以进行持久化；

*#指定分组，可以进行消息的持久化 applies to consumers only***spring.cloud.stream.bindings.myInput.group**=**rabbitmq-group**

在Spring Cloud Stream 中在消费者端如果将队列设置为持久化队列，则队列名称会变为为destination.group，此时消费端的微服务宕机或重启，该队列信息依然会被保留在 RabbitMQ中，后续依然可以进行消费；

分组的第一个作用是消息可以持久化；

**消息分组：（另一个作用，可以实现同一分组只有一个消费者能接收到消费）**

没有做分组时，一个消息可以被多个消费者接收，分组可以让一个消息只能被一个消费者接收，避免一个消息被多个消费者消费；

当项目集群部署了很多份，那么就会变成多个消费者，但是业务可能需要的是一个消息只消费一次，所以此时需要加个分组，就可以实现同一个分组里面的消费者只会有一个消费者能接收到消息；

注意：

1. 不分组的话，消费者要先启动起来，然后再用生产者发送消息，这样才可以接收到消息，否则发送的消息就丢失了，生产者先发了消息，消费者后面才启动的话是接收不到消息的；
2. 不分组的话，多个消费者都能接收消息，也就是一个消息可以被多个消费者接收；

## Spring Cloud Stream设置路由键

默认情况下Spring Cloud Stream传送消息属于广播消息，默认匹配方式是 #，表示所有消费者都可以匹配上，我们也可以通过指定路由键 RoutingKey实现按需求匹配消息消费端进行消息接收处理；

在消费端进行设置

*#设置一个RoutingKey路由key，默认是#，我们可以指定***spring.cloud.stream.rabbit.bindings.myInput.consumer.bindingRoutingKey**=**spring.cloud.stream.#**

**开发的时候有两种选择，**

一种就是 直接SpringBoot + rabbitmq整合实现消息传送；

一种就是 使用Spring Cloud Stream对消息中间件的包装，来实现消息传送；

# 完善Spring Cloud 项目

注册中心

配置中心

服务监控

等服务都可以部署到测试环境，开发的时候直接连接测试环境的这些服务即可；