# 15. Spring Cloud Stream

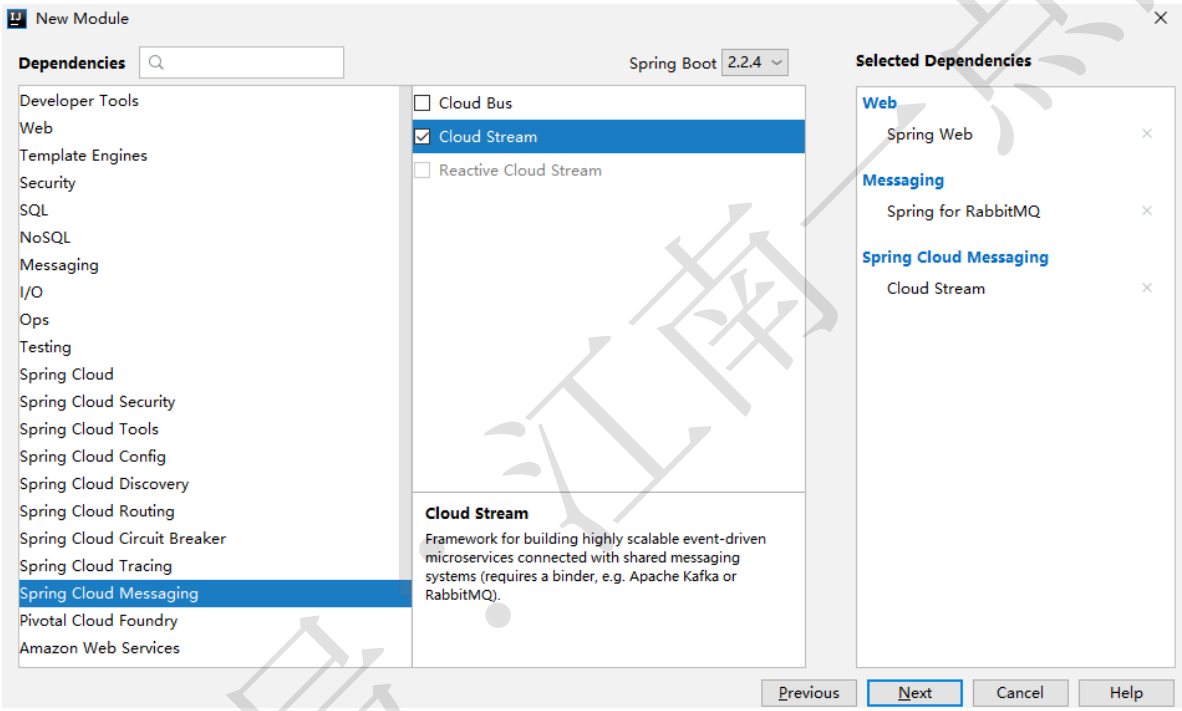
## 15.1 概念

Spring Cloud Stream 用来构建消息驱动的微服务。

Spring Cloud Stream 中，提供了一个微服务和消息中间件之间的一个粘合剂，这个粘合剂叫做 Binder，Binder 负责与消息中间件进行交互。而我们开发者则通过 inputs 或者 outputs 这样的消息通道与 Binder 进行交互。

## 15.2 HelloWorld

创建一个 Spring Cloud Stream 项目，添加三个依赖，web、rabbitmq、cloud stream：




项目创建成功后，添加 RabbitMQ 的基本配置信息：

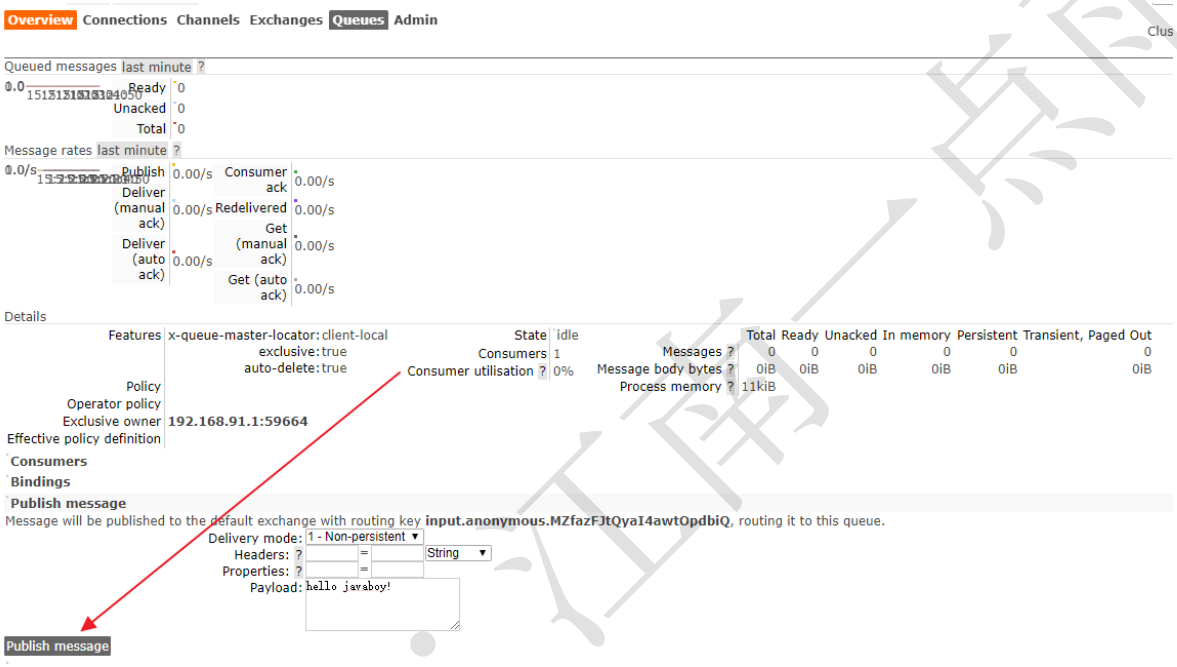
```
spring.rabbitmq.host=192.168.91.128
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

接下来，创建一个简单的消息接收器：

```
//@EnableBinding 表示绑定 Sink 消息通道
@EnableBinding(Sink.class)
public class MsgReceiver {
    public final static Logger logger =
LoggerFactory.getLogger(MsgReceiver.class);

    @StreamListener(Sink.INPUT)
    public void receive(Object payload) {
        logger.info("Received:" + payload);
    }
}
```

启动 stream 项目，然后，在 rabbitmq 后台管理页面去发送一条消息。




## 15.3 自定义消息通道

首先创建一个名为 MyChannel 的接口：

```
public interface MyChannel {
    String INPUT = "javaboy-input";
    String OUTPUT = "javaboy-output";

    @Output(OUTPUT)
    MessageChannel output();

    @Input(INPUT)
    SubscribableChannel input();
}
```

1. 注意，两个消息通道的名字是不一样的
2. 从 F 版开始，默认使用通道的名称作为实例命令，所以这里的通道名字不可以相同（早期版本可以相同），这样的话，为了能够正常收发消息，需要我们在 application.properties 中做一些额外配置。

接下来，自定义一个消息接收器，用来接收自己的消息通道里的消息：

```
@EnableBinding(MyChannel.class)
public class MsgReceiver2 {
    public final static Logger logger =
LoggerFactory.getLogger(MsgReceiver2.class);

    @StreamListener(MyChannel.INPUT)
    public void receive(Object payload) {
        logger.info("received2:" + payload);
    }
}
```

再定义一个 HelloController 进行测试：

```
@RestController
public class HelloController {
    @Autowired
    MyChannel myChannel;
    @GetMapping("/hello")
    public void hello() {
        myChannel.output().send(MessageBuilder.withPayload("hello spring cloud
stream!").build());
    }
}
```

同时，为了让消息输入输出通道对接上（因为现在这两个的通道名称不一样），再增加一点额外配置。

```
spring.cloud.stream.bindings.javaboy-input.destination=javaboy-topic
spring.cloud.stream.bindings.javaboy-output.destination=javaboy-topic
```

## 15.4 消息分组

默认情况下，如果消费者是一个集群，此时，一条消息会被多次消费。通过消息分组，我们可以解决这个问题。

只需要添加如下配置即可：

```
spring.cloud.stream.bindings.javaboy-input.group=g1
spring.cloud.stream.bindings.javaboy-output.group=g1
```

## 15.5 消息分区

通过消息分区可以实现相同特征的消息总是被同一个实例处理。只需要添加如下配置即可：

```
spring.cloud.stream.bindings.javaboy-input.group=g1
spring.cloud.stream.bindings.javaboy-output.group=g1
# 开启消息分区（消费者上配置
spring.cloud.stream.bindings.javaboy-input.consumer.partitioned=true
# 消费者实例个数（消费者上配置
spring.cloud.stream.instance-count=2
# 当前实例的下标（消费者上配置
spring.cloud.stream.instance-index=0
# （生产者上配置
spring.cloud.stream.bindings.javaboy-output.producer.partition-key-expression=1
# 消费端的节点数量（生产者上配置
spring.cloud.stream.bindings.javaboy-output.producer.partition-count=2
```

接下来，启动两个实例，注意，启动时，spring.cloud.stream.instance-index 要动态修改。

```
java -jar stream-0.0.1-SNAPSHOT.jar --server.port=8080 --
spring.cloud.stream.instance-index=0
java -jar stream-0.0.1-SNAPSHOT.jar --server.port=8081 --
spring.cloud.stream.instance-index=1
```

## 15.6 定时任务

每天定时执行的任务，可以使用 cron 表达式，有一种比较特殊的定时任务，例如几分钟后执行，这种可以结合 Spring Cloud Stream+RabbitMQ 来实现。

这个需要首先下载一个 rabbitmq 插件。：https://dl.bintray.com/rabbitmq/community-plugins/3.7.x/rabbitmq_delayed_message_exchange/rabbitmq_delayed_message_exchange-20171201-3.7.x.zip

执行如下命令：

```
# 解压下载的文件
unzip rabbitmq_delayed_message_exchange-20171201-3.7.x.zip
# 将解压后的文件，拷贝到 Docker 容器中
docker cp /root/rabbitmq_delayed_message_exchange-20171201-3.7.x.ez javaboy-
rabbit:/plugins
# 进入到容器中
docker exec -it javaboy-rabbit /bin/bash
# 启用插件
rabbitmq-plugins enable rabbitmq_delayed_message_exchange
# 查看是否启用成功
rabbitmq-plugins list
```

配置文件中，开启消息延迟功能：

```
# 开启消息延迟功能
spring.cloud.stream.rabbit.bindings.javaboy-input.consumer.delayed-exchange=true
spring.cloud.stream.rabbit.bindings.javaboy-output.producer.delayed-
exchange=true
```

同时注意，消息输入输出通道的 destination 定义：

```
spring.cloud.stream.bindings.javaboy-input.destination=delay_msg
spring.cloud.stream.bindings.javaboy-output.destination=delay_msg
```

然后在消息发送时，设置消息延迟时间为 3 秒：

```
@RestController
public class HelloController {
    public final static Logger logger =
LoggerFactory.getLogger(HelloController.class);
    @Autowired
    MyChannel myChannel;

    @GetMapping("/hello")
    public void hello() {
        logger.info("send msg:" + new Date());
        myChannel.output().send(MessageBuilder.withPayload("hello spring cloud
stream!").setHeader("x-delay", 3000).build());
    }
}
```

同时，在接收消息时，也打印出延迟时间：

```
@EnableBinding(MyChannel.class)
public class MsgReceiver2 {
    public final static Logger logger =
LoggerFactory.getLogger(MsgReceiver2.class);

    @StreamListener(MyChannel.INPUT)
    public void receive(Object payload) {
        logger.info("received2:" + payload+":"+new Date());
    }
}
```