博学谷
www.boxuegu.com

第五章 Spring Cloud Alibaba RocketMQ 消息驱动

一样的在线教育，不一样的教学品质

目录 Contents

RocketMQ 核心概念

小节导学

RocketMQ 由哪些部分组成?

各部分的作用是什么?

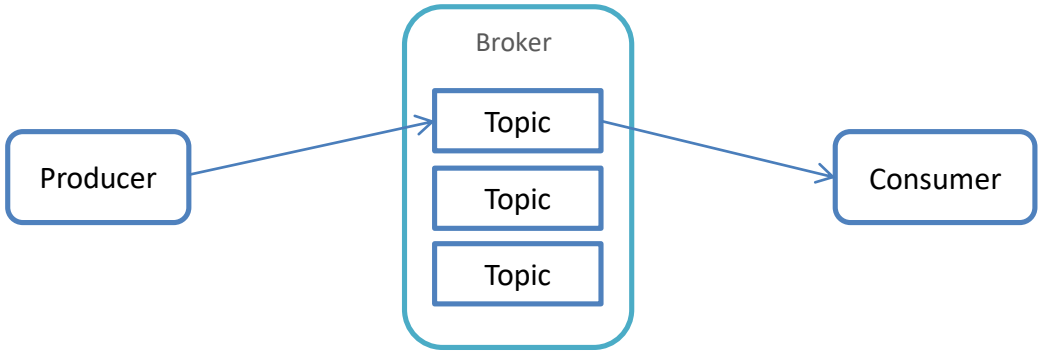
整体结构是怎样的?

本节我们就要探讨这些问题，对 RocketMQ 有个全局性的认识。

- 消息模型
- RocketMQ 架构

1. 消息模型

消息系统通用模型



- **Producer**

  生产者，生产消息

- **Consumer**

  消费者，消费消息

- **Topic**

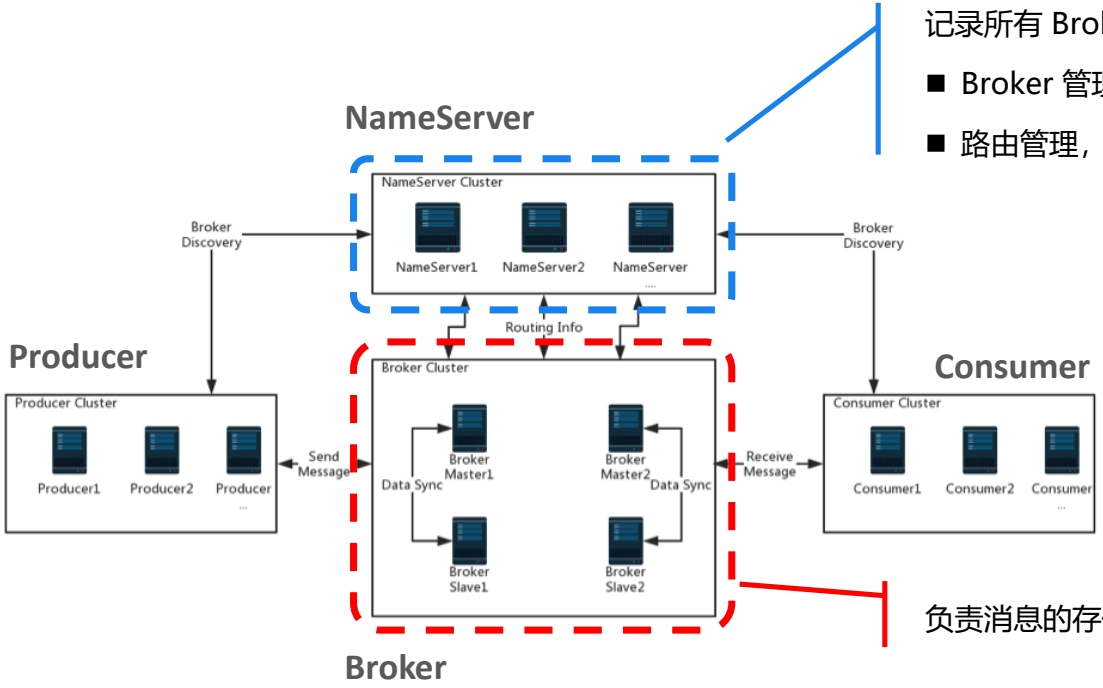
  主题，消息的集合，基本订阅单位

- **Broker**

  消息代理，存储消息，转发消息

2. RocketMQ 架构

RocketMQ 架构



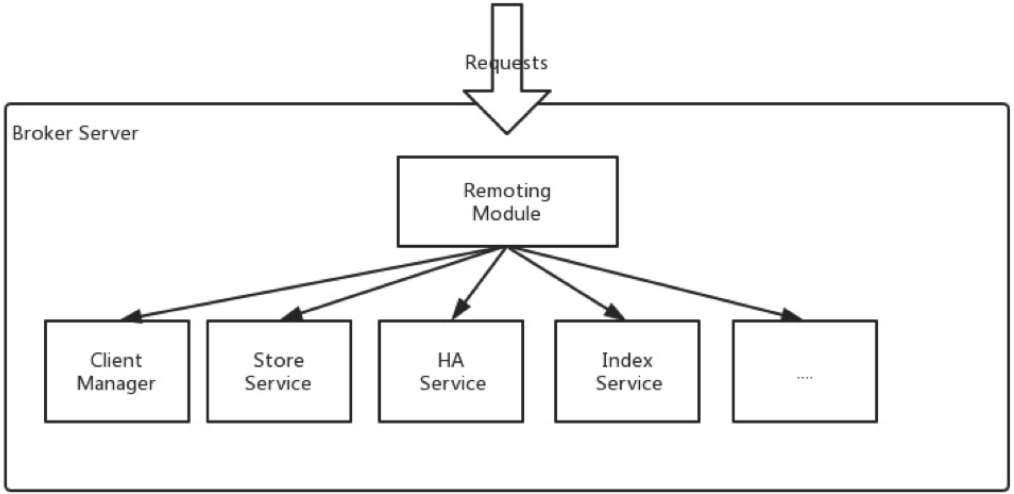
记录所有 Broker 的信息，提供服务发现、路由服务。

- Broker 管理，Broker 向 NameServer 注册
- 路由管理，维护所有 Broker 和队列的信息，供 查询

负责消息的存储、传递、查询，以及 HA 保障

2. RocketMQ 架构

Broker 核心子模块



- **Remoting**

  Broker 入口

- **Client Manager**

  管理 Producer/Consumer

- **Store Service**

  消息的存储、查询接口

- **HA Service**

  主从 Broker 数据同步

- **Index Service**

  构建消息索引

RocketMQ 核心概念-总结

重难点

1. 理解消息系统中的通用模型
2. 理解 RocketMQ 宏观架构图
3. 理解 RocketMQ Broker 核心子模块的作用

博学谷
www.boxuegu.com

目录 Contents

RocketMQ 环境搭建

小节导学

RocketMQ 部署结构中主要包括：

1. NameServer - Producer 和 Consumer 通过 NameServer 查找 Topic 所在的 Broker。
2. Broker - 负责消息的存储、转发。

部署完 NameServer、Broker 之后，RocketMQ 就可以正常工作了，但所有操作都是通过命令行，不太方便，所以我们还需要部署一个扩展项目 rocketmq-console，可以通过web界面来管理 RocketMQ。

本节我们的目标就是完成这3项的部署。

- 部署 RocketMQ（NameServer、Broker）
- rocketmq-console

1. 部署 RocketMQ

部署步骤

1. 下载

http://rocketmq.apache.org/dowloading/releases/

2. 解压编译

```
> unzip rocketmq-all-4.7.0-source-release.zip
> cd rocketmq-all-4.7.0-source-release
> mvn -Prelease-all -DskipTests clean install -U
```

1. 部署 RocketMQ

部署步骤

3. 启动

创建配置文件： `conf/broker.properties`

写入：`brokerIP1=【你的IP】`

- 启动 NameServer

```
> cd distribution/target/rocketmq-4.6.0/rocketmq-4.6.0
> nohup sh bin/mqnamesrv &
```

- 启动 Broker

```
> nohup sh bin/mqbroker -n IP:9876 -c conf/broker.properties &
```

1. 部署 RocketMQ

常见问题

在启动 broker 时遇到报错：

```
Java HotSpot(TM) 64-Bit Server VM warning: INFO: os::commit_memory(0x00000005c0000000, 8589934592,
0) failed; error='Cannot allocate memory' (errno=12)
...
```

原因是**内存不足。**

解决方法：

修改 `bin/runbroker.sh`，把内存参数改小一点，例如：

```
JAVA_OPT="${JAVA_OPT} -server -Xms512m -Xmx512m -Xmn256m"
```

1. 部署 RocketMQ

测试

开2个终端，都进入到 rocketmq 目录

终端窗口 1 用于生产消息，执行：

```
> export NAMESRV_ADDR=localhost:9876
> sh bin/tools.sh org.apache.rocketmq.example.quickstart.Producer
```

终端窗口 2 用于消费消息，执行：

```
> export NAMESRV_ADDR=localhost:9876
> sh bin/tools.sh org.apache.rocketmq.example.quickstart.Consumer
```

2. 部署 rocketmq-console

部署步骤

- 下载

一个 rocketmq 的扩展项目，其中的 rocketmq-console 是控制台，下载项目：

```
https://github.com/apache/rocketmq-externals
```

- 配置

```
> cd rocketmq-console
> vim src/main/resources/application.properties
```

1. 设置 console 的端口
2. 找到 `rocketmq.config.namesrvAddr`，填上自己的地址端口

- 运行

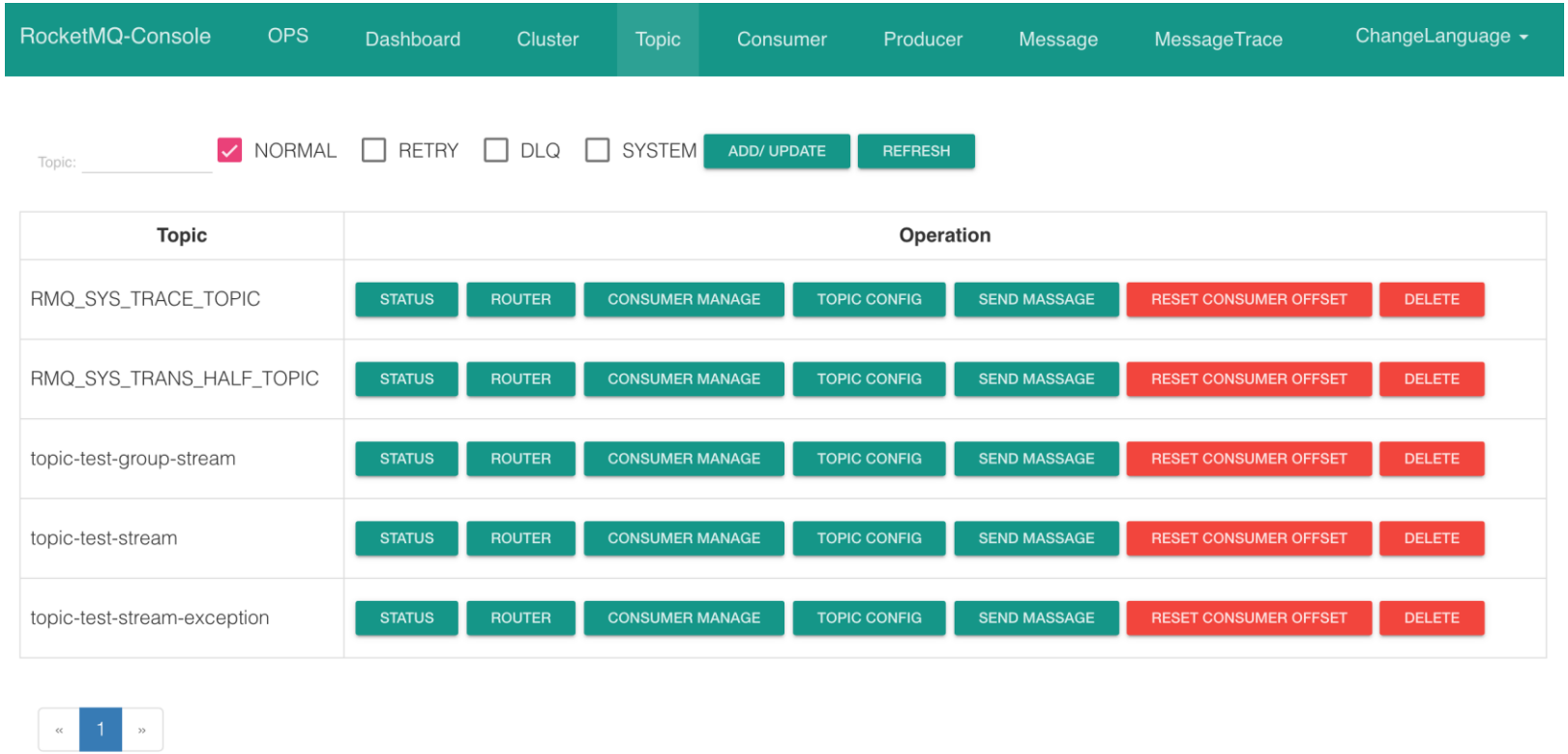
```
> mvn spring-boot:run
```

或打包：

```
> mvn clean package -Dmaven.test.skip=true
```

2. 部署 rocketmq-console

控制台效果



RocketMQ 环境搭建-总结

重难点

1. 理解 RocketMQ 的部署结构
2. 掌握 RocketMQ（NameServer、Broker）搭建方式
3. 掌握 RocketMQ Console 的搭建方式

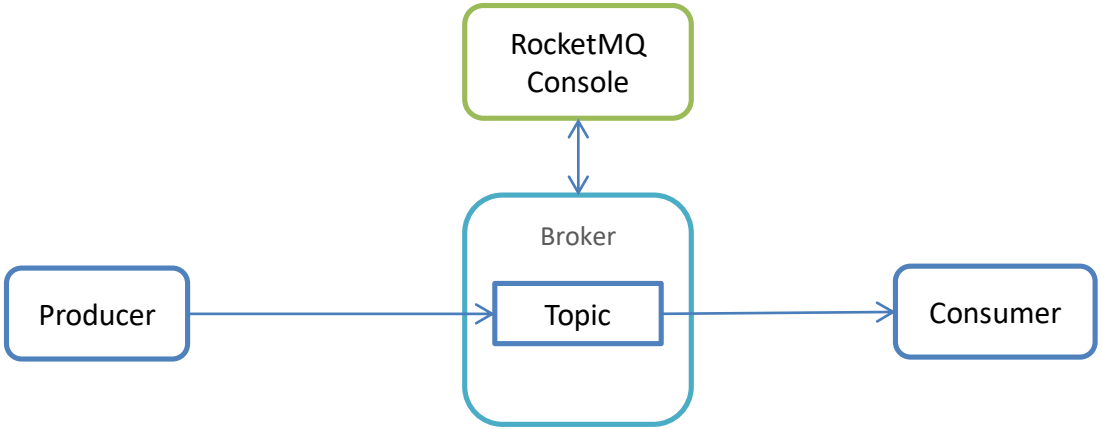
目录 Contents

RocketMQ 生产者与消费者开发

小节导学

本节目标：

- 开发一个 Producer，向 RocketMQ 发送消息，通过 RocketMQ Console 验证发送成功
- 开发一个 Consumer，从 RocketMQ 成功接收消息



RocketMQ 生产者与消费者开发

开发步骤 - Producer

- 添加 RocketMQ 依赖

```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
    <version>2.1.0</version>
</dependency>
```

- RocketMQ 配置

```
rocketmq:
  name-server: 49.235.54.112:9876
  producer:
    group: test-group
```

RocketMQ 生产者与消费者开发

开发步骤- Producer

- 创建消息实体

```
public class User {
    Long id;
    String name;
    public User(){}
    public User(Long id, String name) {
        this.id = id;  this.name = name;
    }
    // setter/getter
    @Override
    public String toString() {
        return "User{id=" + id +", name='" + name + "'}";
    }
}
```

RocketMQ 生产者与消费者开发

开发步骤 - Producer

- 使用 RocketMQTemplate 发送消息

```
@RestController
public class TestController {
    @Autowired
    RocketMQTemplate rocketMQTemplate;

    @GetMapping("/sendmsg")
    public String sendmsg(Long id, String name){
        rocketMQTemplate.convertAndSend("topic-test", new User(id, name));
        return "ok";
    }
}
```

RocketMQ 生产者与消费者开发

开发步骤- Consumer

- 使用 RocketMQListener 接收消息

```
@Service
@RocketMQMessageListener(consumerGroup = "group-consumer", topic = "topic-test")
public class MyMQConsumer implements RocketMQListener<User> {
    @Override
    public void onMessage(User user) {
        // consume logic
        System.out.println(user);
    }
}
```

RocketMQ 生产者与消费者开发

Spring 消息模型编程模板

生产者 Template

- RocketMQ : RocketMQTemplate
- ActiveMQ/Artemis : JmsTemplate
- RabbitMQ : AmqpTemplate
- Kafka : KafkaTemplate

消费者 Listener

- RocketMQ : RocketMQMessageListener
- ActiveMQ/Artemis : JmsListener
- RabbitMQ : RabbitListener
- Kafka : KafkaListener

RocketMQ 生产者与消费者开发-总结

重难点

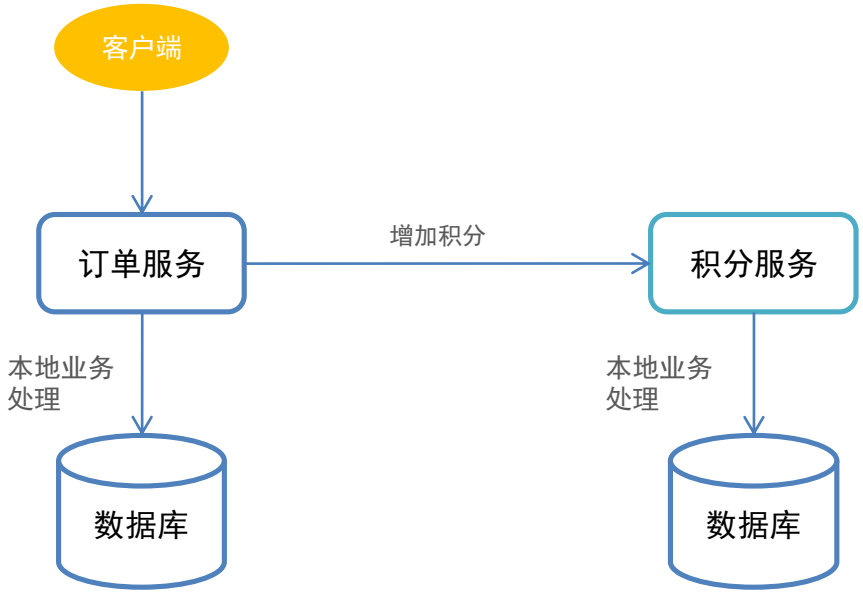
1. 理解 SpringBoot RocketMQ 的开发思路
2. 掌握 SpringBoot RocketMQ 开发 Producer、Consumer 的流程
3. 掌握 rocketmq-console 的基本操作

目录 Contents

RocketMQ 实现分布式事务

小节导学

分布式事务问题

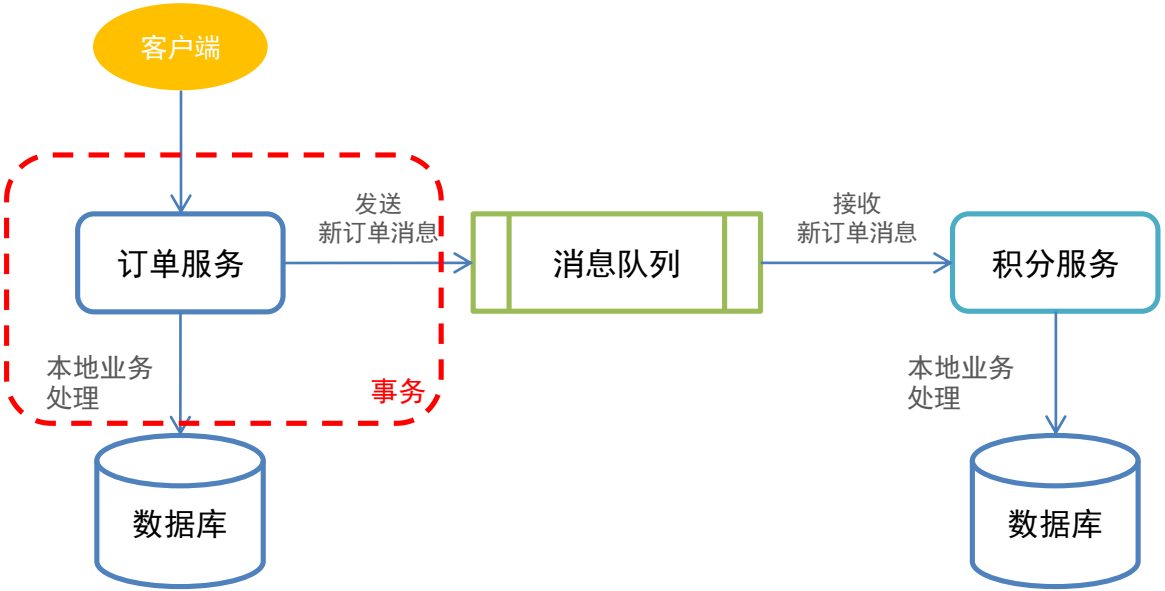


RocketMQ 实现分布式事务

小节导学

分布式事务的解决方案中，有一个可靠消息模式，就是使用消息队列来实现的。

这个方案的关键点：怎么保证本地事务与发送消息保持一直，本地成功 & 发送成功 || 本地失败 & 发送失败



RocketMQ 实现分布式事务

小节导学

RocketMQ 独有的事务消息机制可以方便的解决这个问题。

RocketMQ 的事务消息是怎么工作的?

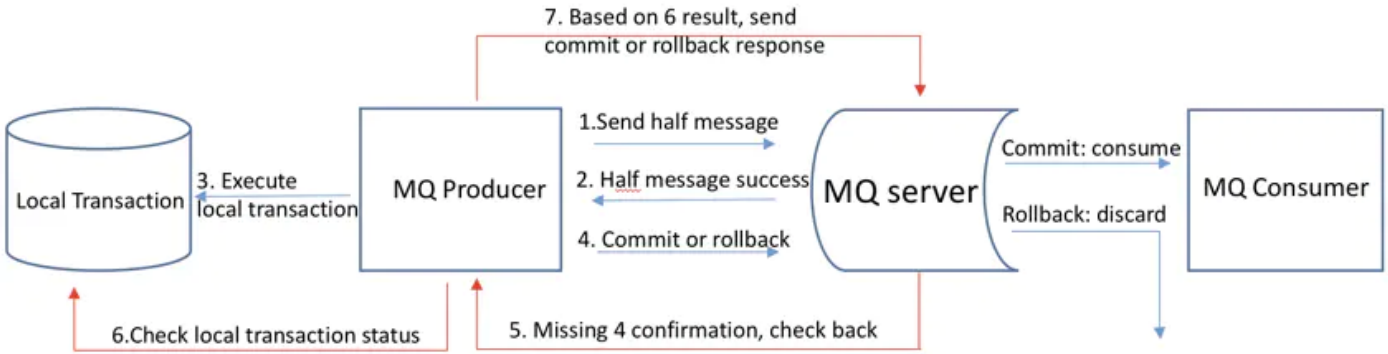
怎么实现事务消息?

本机我们就解决这些问题。

- RocketMQ 事务消息机制
- RocketMQ 分布式事务实践
- 分布式事务问题扩展

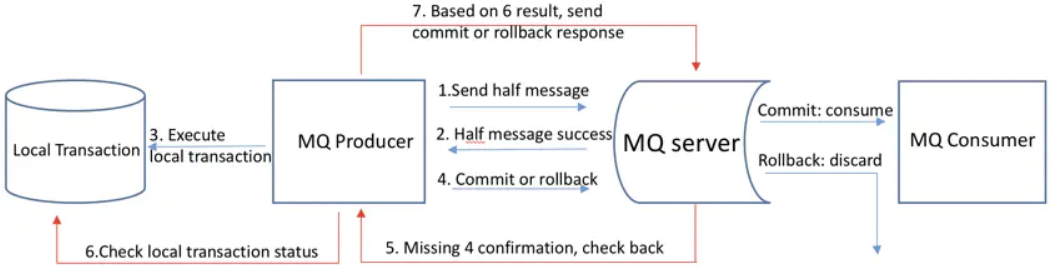
1. RocketMQ 事务消息机制

事务消息流程图

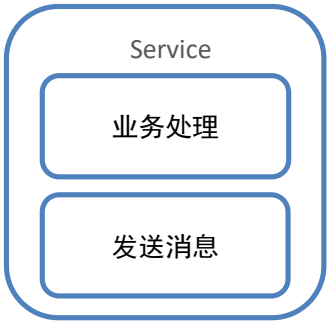


2. RocketMQ 分布式事务实践

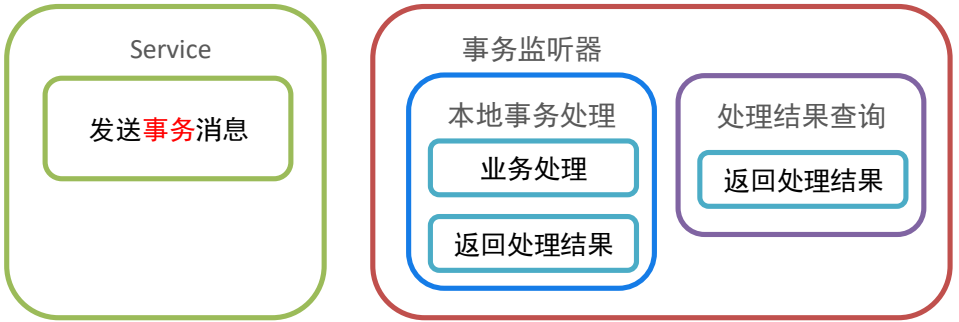
事务消息开发流程



非事务消息开发模式



事务消息开发模式

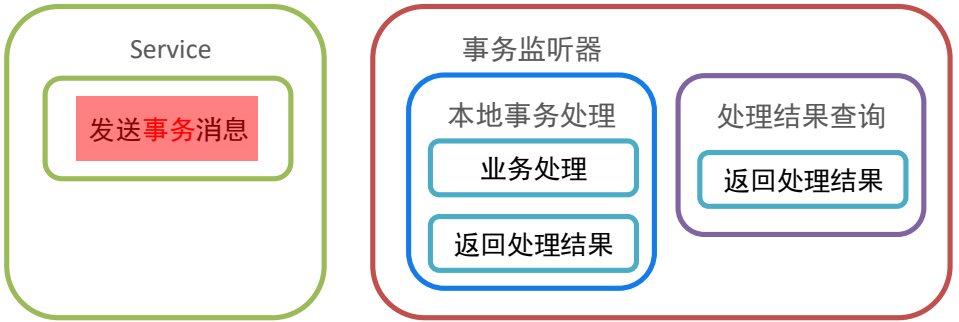


2. RocketMQ 分布式事务实践

实现步骤

1. Producer 发送事务消息

```
@GetMapping("/tx/test")
public String sendTxMsg() {
    rocketMQTemplate.sendMessageInTransaction("topic-tx",
            MessageBuilder.withPayload("hi").build(), null);
    return "ok";
}
```

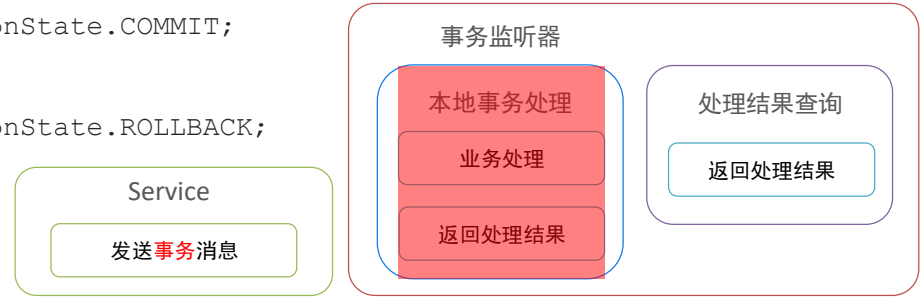


2. RocketMQ 分布式事务实践

实现步骤

2. Producer 事务消息监听器

```
@Component
@RocketMQTransactionListener
public class TxmsgListener implements RocketMQLocalTransactionListener {
    @Override
    public RocketMQLocalTransactionState executeLocalTransaction(Message message, Object o) {
        RocketMQLocalTransactionState state = RocketMQLocalTransactionState.ROLLBACK;
        try {
            // 本地业务逻辑
            return RocketMQLocalTransactionState.COMMIT;
        }catch (Exception e){
            return RocketMQLocalTransactionState.ROLLBACK;
        }
    }
    ....
}
```

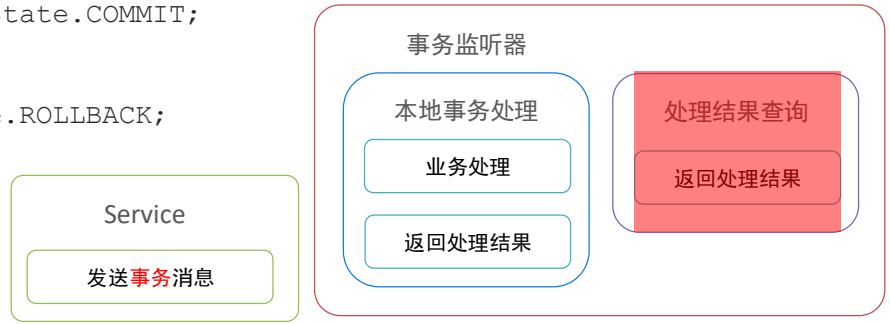


2. RocketMQ 分布式事务实践

实现步骤

2. Producer 事务消息监听器

```
@Component
@RocketMQTransactionListener
public class TxmsgListener implements RocketMQLocalTransactionListener {
        @Override
    public RocketMQLocalTransactionState checkLocalTransaction(Message message) {
        // 检查本地执行状态 ...
        if(state > 0){
            return RocketMQLocalTransactionState.COMMIT;
        }
        return RocketMQLocalTransactionState.ROLLBACK;
    }
    ....
}
```



2. RocketMQ 分布式事务实践

实现步骤

3. Consumer 接收消息

```
@Service
@RocketMQMessageListener(consumerGroup = "tx-consumer", topic = "topic-tx")
public class MyTxConsumer implements RocketMQListener<String> {
    @Override
    public void onMessage(String msg) {
        System.out.println("MyTxConsumer receive:" + msg);
    }
}
```

2. RocketMQ 分布式事务实践

测试

实验场景

1. 本地事务正常，提交事务消息，Consumer 接收

2. 本地事务失败，回滚事务消息，Consumer 未接收

3. 本地事务没返回，mq 回查，Consumer 接收

3. 分布式事务问题扩展

1. 幂等

上面测试第 3 个场景的时候，Consumer 会收到 2 次消息，可能导致重复增加积分。

保证消息不被重复处理，就是“幂等”

幂等是一个数学概念，可以理解为：

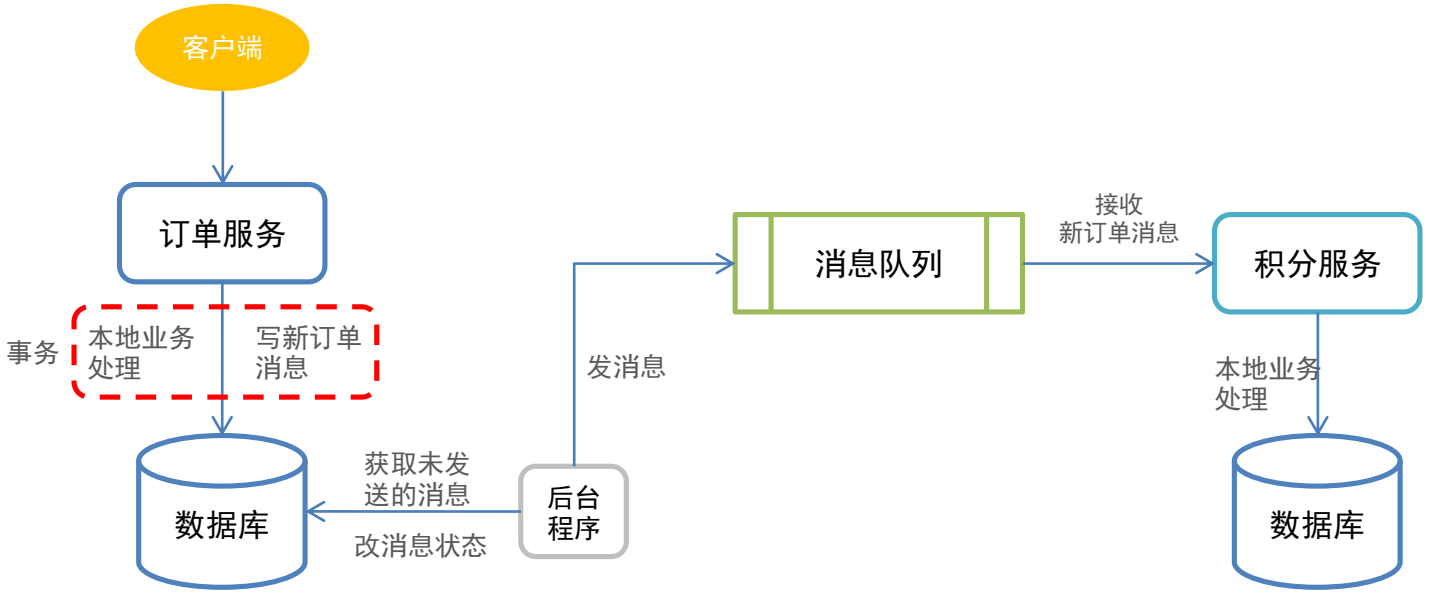
同一个函数，参数相同的情况下，多次执行后的结果一致

解决方法：

Consumer 端建立一个判重表，每次收到消息后先，先到判重表中看一下，看这条消息是否处理过。

3. 分布式事务问题扩展

2. 非 RocketMQ 的实现方案



RocketMQ 实现分布式事务-总结

重难点

1. 理解分布式事务的可靠消息模式
2. 理解 RocketMQ 事务消息机制的处理流程
3. 掌握 RocketMQ 事务消息的开发流程
4. 掌握幂等的概念及处理方式

目录 Contents

SpringCloud Stream 开发模型

小节导学

SpringCloud Stream 对 MQ 的操作做了**高度抽象**，使我们开发时可以对底层的 MQ **无感知**，更换 MQ 时只需要更换相应的 Binder 即可。

这个 Binder 是什么?

Stream 怎么做的抽象?

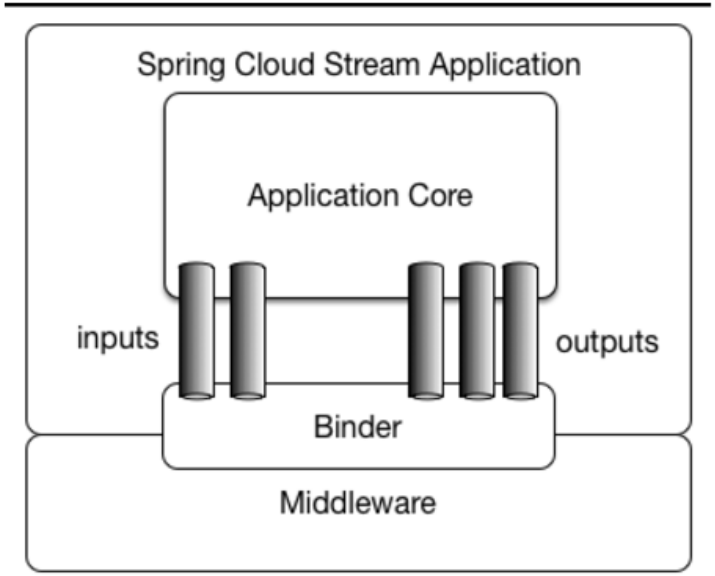
如何使用 Stream 做消息开发?

这些就是本节学习的内容，包括：

- SpringCloud Stream 模型
- SpringCloud Stream 生产与消费开发实践

1. SpringCloud Stream 模型

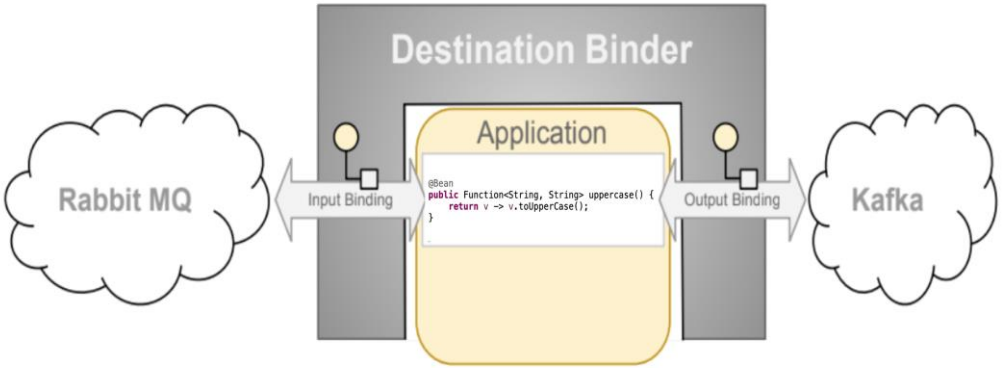
应用模型



- 应用程序集成 SpringCloud Stream
- 应用程序无需直接操作底层的消息中间件
- SpringCloud Stream 的 Binder 连接消息中间件
- 应用通过 input 与 output 连接 Binder

1. SpringCloud Stream 模型

开发模型



- Destination Binder

  目标绑定器，与消息中间件通信的组件

- Destination Binding

  目标绑定，是连接应用和消息中间件的桥梁，用于消息的消费和生产，由 Binder 创建

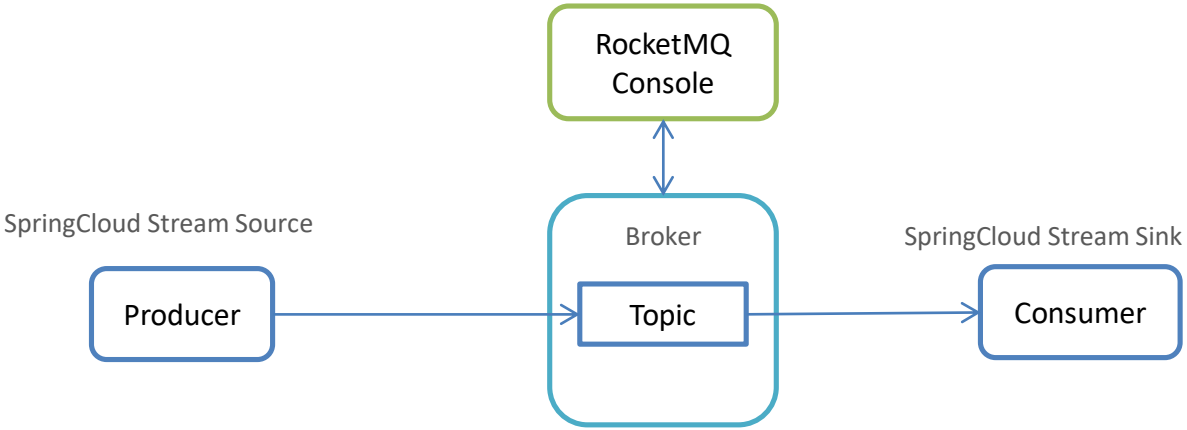
- Message

  消息，用于 Producer、Consumer 通过 Binder 沟通的规范数据。

2. SpringCloud Stream 生产与消费开发实践

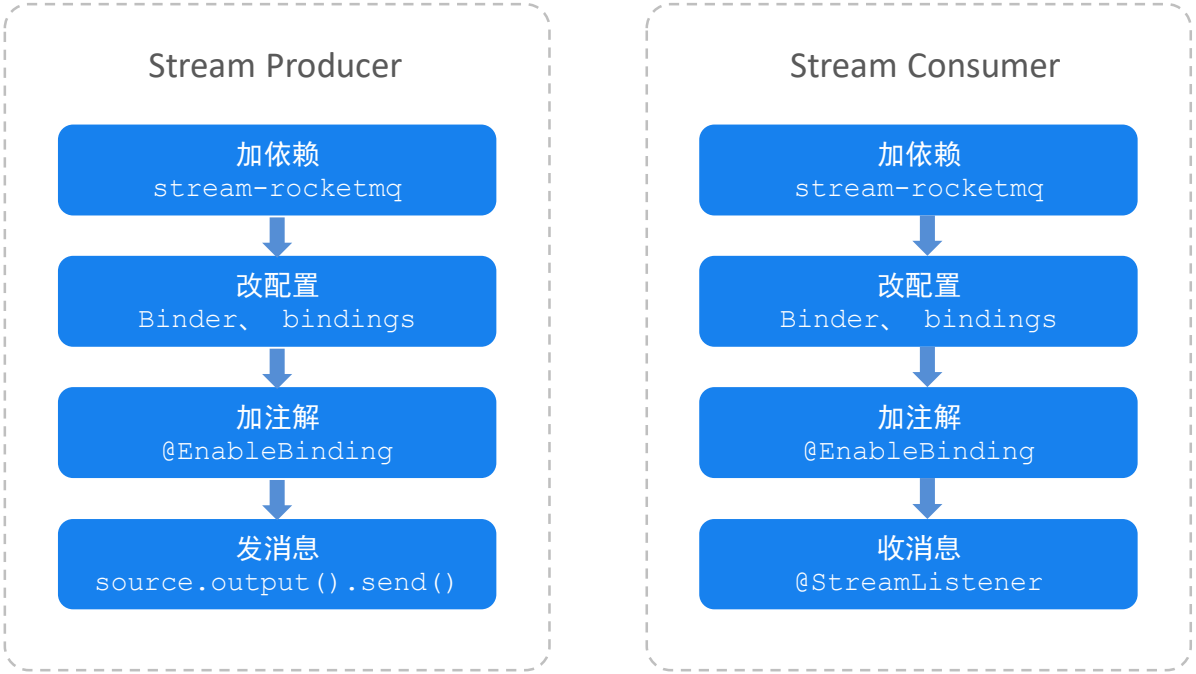
目标

1. 创建一个 stream-producer，集成 SpringCloud Stream，绑定 RocketMQ，发送消息
2. 创建一个 stream-Consumer，集成 SpringCloud Stream，绑定 RocketMQ，接收消息



2. SpringCloud Stream 生产与消费开发实践

流程



2. SpringCloud Stream 生产与消费开发实践

步骤

1. stream-producer

添加 stream-rocketmq 依赖：

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-rocketmq</artifactId>
</dependency>
```

2. SpringCloud Stream 生产与消费开发实践

步骤

1. stream-producer

rocketmq binder、binding destination 属性配置：

```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          name-server: 49.235.54.12:9876
      bindings:
        output:
          destination: topic-test-stream
          group: stream-consumer-group
```

2. SpringCloud Stream 生产与消费开发实践

步骤

1. stream-producer

开启 Binding：

```
@SpringBootApplication
@EnableBinding(Source.class)
public class StreamproducerApplication {
            public static void main(String[] args) {
                        SpringApplication.run(StreamproducerApplication.class, args);
            }
}
```

2. SpringCloud Stream 生产与消费开发实践

步骤

1. stream-producer

发送消息：

```
@Autowired
Source source;

@GetMapping("teststream")
public String testStream(){
    source.output().send(
            MessageBuilder.withPayload("msg").build()
    );
    return "ok";
}
```

2. SpringCloud Stream 生产与消费开发实践

步骤

2. stream-consumer

添加 stream-rocketmq 依赖：

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-rocketmq</artifactId>
</dependency>
```

2. SpringCloud Stream 生产与消费开发实践

步骤

2. stream-consumer

rocketmq binder、binding destination 属性配置：

```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          name-server: 49.235.54.12:9876
      bindings:
        input:
          destination: topic-test-stream
          group: stream-consumer-group
```

2. SpringCloud Stream 生产与消费开发实践

步骤

2. stream-consumer

开启 Binding：

```
@SpringBootApplication
@EnableBinding(Sink.class)
public class StreamconsumerApplication {
            public static void main(String[] args) {
                        SpringApplication.run(StreamconsumerApplication.class, args);
            }
}
```

2. SpringCloud Stream 生产与消费开发实践

步骤

2. stream-consumer

接收消息：

```
@Service
public class MyStreamConsumer {
    @StreamListener(Sink.INPUT)
    public void receive(String msg){
        // consume logic
        System.out.println("receive: " + msg);
    }
}
```

2. SpringCloud Stream 生产与消费开发实践

消息过滤

Consumer 可能希望处理具有某些特征的消息，这就需要对消息进行过滤。
最简单的方法就是收到消息后自己判断一下 `if ... else ...`
为了简化开发，Stream 提供了消息过滤的方式，在 Listener 注解中加一个判断条件即可：

```
@Service
public class MyStreamConsumer {
    @StreamListener(value = Sink.INPUT,
            condition = "headers['test-header']=='my test'")
    public void receive(String msg){
        System.out.println("receive: " + msg);
    }
}
```

2. SpringCloud Stream 生产与消费开发实践

消息监控

收发消息不正常时怎么办？可以查看监控信息
actuator 中有 binding 信息、健康检查信息，为我们提供排错依据

```
/actuator/bindings
/actuator/health
/actuator/channels
```

SpringCloud Stream 开发模型-总结

重难点

1. 理解 SpringCloud Stream 的应用模型与开发模型
2. 掌握 SpringCloud Stream 生产消息、消费消息的开发方法
3. 掌握监控消息生产、消费问题的方法

目录 Contents

SpringCloud Stream 自定义接口

小节导学

回顾一下上节通过 Stream 发送消息的方式：

配置文件中指定了“bindings.output”

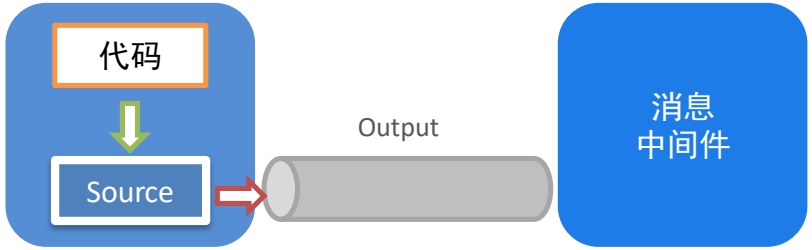
使用注解开启了 binding “@EnableBinding(Source.class)”

就可以使用“Source”发送消息了。

这种默认的自动化方式非常便利，但是，如果想再加一个“output”通道怎么办?

本节就来解决自定义接口的问题：

- 自定义 source
- 自定义 sink



1. 自定义 source

步骤

1. 添加 output 配置

```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          name-server: 49.235.54.112:9876
      bindings:
        output:
          destination: topic-test-stream
        my-output:
          destination: topic-test-stream-myoutput
```

2. 创建 source 接口

```
public interface MySource {

    String my_output = "my-output";
    @Output(my_output)
    MessageChannel output();
}
```

1. 自定义 source

步骤

3. 启用自定义 source

```
@SpringBootApplication
@EnableBinding({Source.class, MySource.class})
public class StreamproducerApplication {
            public static void main(String[] args) {
                        SpringApplication.run(StreamproducerApplication.class, args);
            }
}
```

1. 自定义 source

步骤

4. 发送消息

```
@Autowired
MySource mysource;

@GetMapping("teststream-mysource")
public String testStream_mysource(){
    mysource.output().send(
            MessageBuilder.withPayload("msg mysource").build()
    );
    return "ok";
}
```

2. 自定义 sink

步骤

1. 添加 iutput 配置

```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          name-server: 49.235.54.112:9876
      bindings:
        input:
          destination: topic-test-stream
        my-input:
          destination: topic-test-stream-myoutput
          group: my-group
```

2. 创建 sink 接口

```
public interface MySink {

    String MY_INPUT = "my-input";
    @Input(MY_INPUT)
    SubscribableChannel input();
}
```

2. 自定义 sink

步骤

3. 启用自定义 sink

```
@SpringBootApplication
@EnableBinding({Sink.class, MySink.class})
public class StreamconsumerApplication {
            public static void main(String[] args) {
                        SpringApplication.run(StreamconsumerApplication.class, args);
            }
}
```

2. 自定义 sink

步骤

4. 接收消息

```
@Service
public class MyInputStreamConsumer {
    @StreamListener(MySink.MY_INPUT)
    public void receive(String msg){
        // consume logic
        System.out.println("myinput receive: " + msg);
    }
}
```

SpringCloud Stream 自定义接口-总结

重难点

1. 掌握 SpringCloud Stream 自定义 source、sink 的开发方法

目录 Contents

SpringCloud Stream 消费异常处理

小节导学

消费者在接收消息时，可能会发生异常，如果我们想处理这些异常，需要采取一些处理策略，可以分为：

1. 应用级处理 - 通用，与底层 MQ 无关
2. 系统级处理 - 根据不同的 MQ 特性进行处理，例如 RabbitMQ 可以放入死信队列
3. 重试 RetryTemplate - 配置消费失败后如何重试

本节我们学习最通用的“应用级处理”策略，此方式又分为：

- 局部处理方式
- 全局处理方式

SpringCloud Stream 消费异常处理

局部消费异常处理

```
@ServiceActivator(
        inputChannel = “主题.消费者组.errors"
)
public void handleError(ErrorMessage errorMessage) {
    // 异常处理逻辑
}
```

全局消费异常处理

```
@StreamListener("errorChannel")
public void handleError(ErrorMessage errorMessage) {
    // 异常处理逻辑
}
```

SpringCloud Stream 消费异常处理-总结

重难点

1. 掌握局部异常处理的开发方法
2. 掌握全局异常处理的开发方法

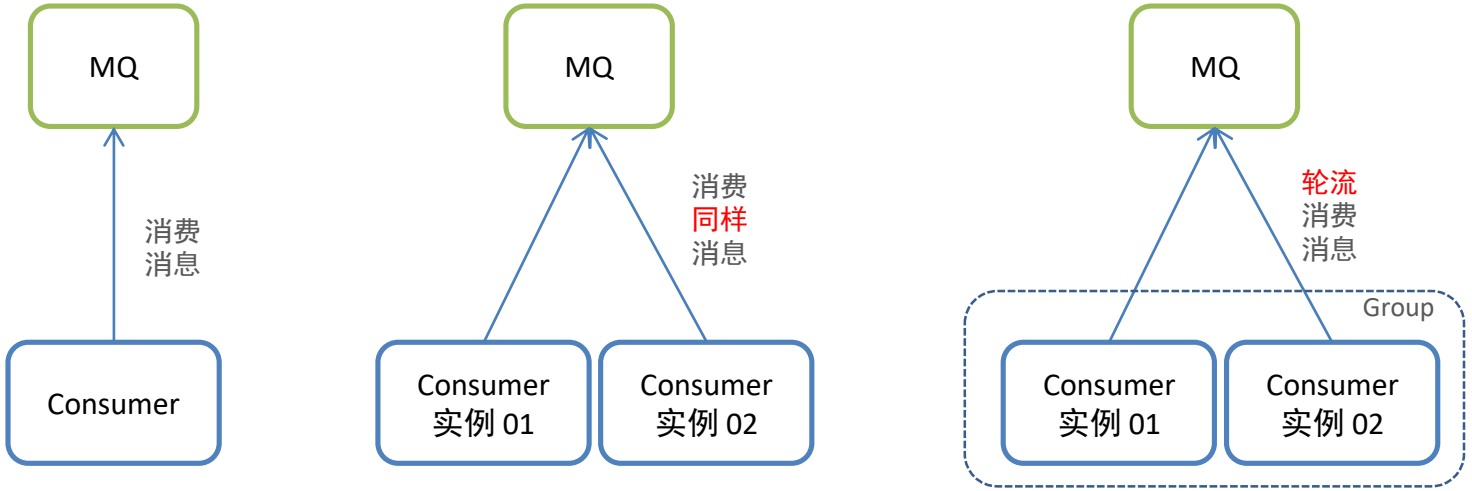
目录 Contents

SpringCloud Stream 消费组

小节导学

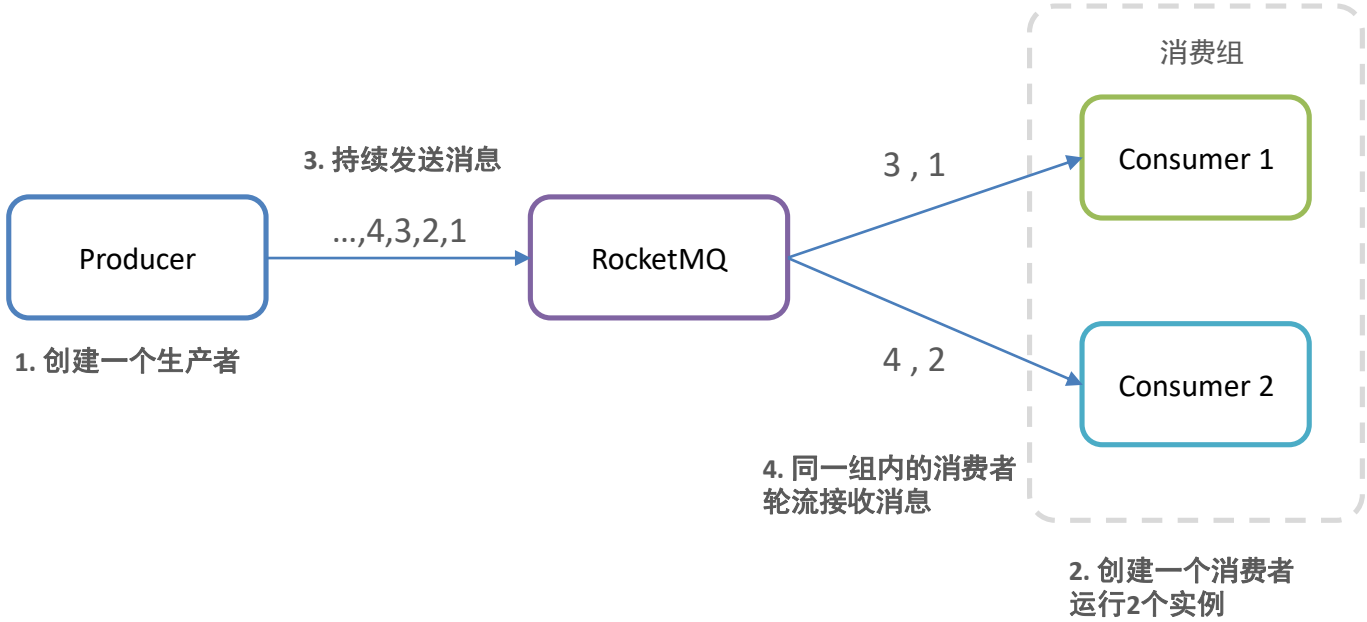
线上环境中，一个服务通常都会运行多个实例，以保证高可靠，对于消费服务，运行多个实例的时候，每个实例就都会去消费消息，造成重复消费，设置 Consumer Group（消费组）可以实现组内消费者均衡消费。

本节我们就学习消费组的设置，体验其效果。



SpringCloud Stream 消费组

实践流程



SpringCloud Stream 消费组-总结

重难点

1. 理解 SpringCloud Stream 中 Consumer Group 的作用
2. 掌握 Consumer Group 的配置使用方法

目录 Contents

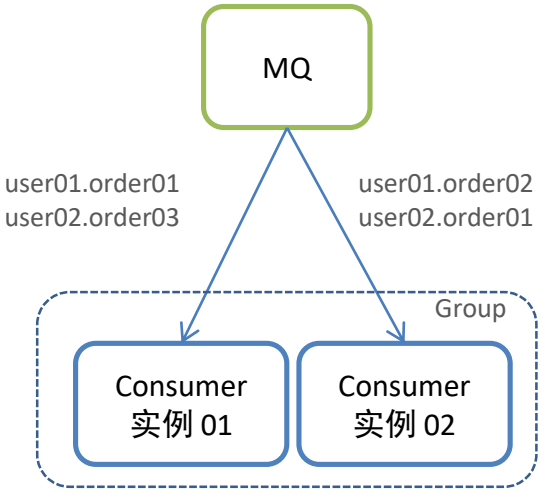
SpringCloud Stream 消息分区

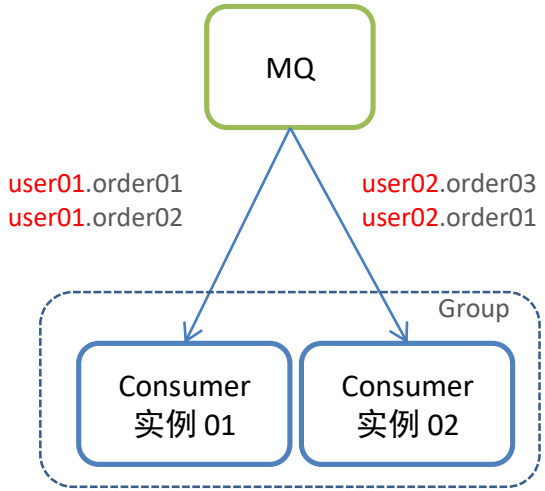
小节导学

消息被哪个实例消费是不一定的，但如果我们希望同一类的消息被同一个实例消费怎么办?

例如同一个用户的订单消息希望被同一个示例处理，这样更便于统计。

SpringCloud Stream 提供了消息分区的功能，可以满足这个场景的需求，本节我们就学习如何使用。

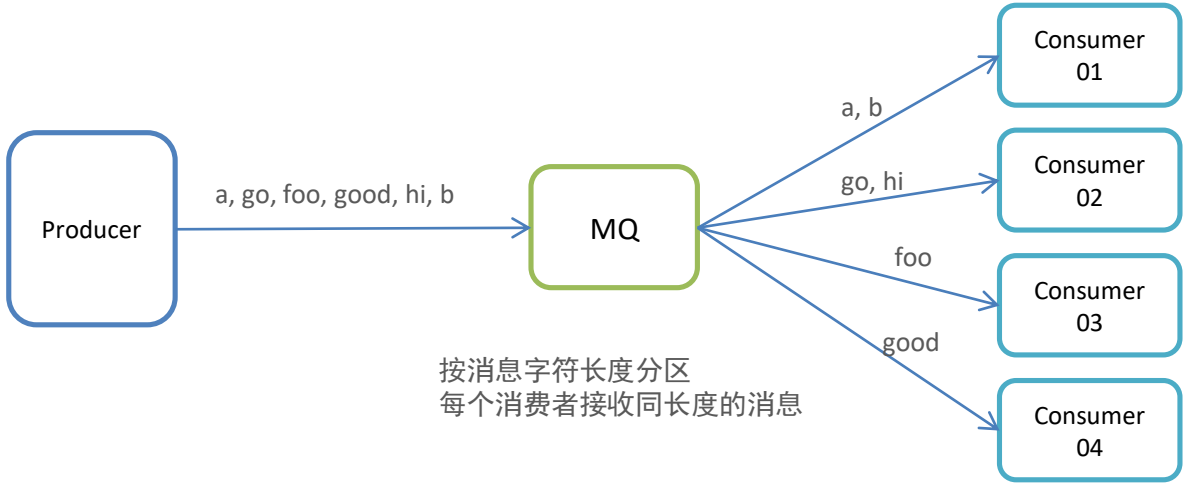




SpringCloud Stream 消息分区

目标

1. 创建1个 Producer 一直发送消息，设置消息如何分区
2. 创建1个 Consumer 接收消息，设置按分区接收消息
3. 启动4个 Consumer 实例，指定分区标识，同一分区的消息应被相同的 Consumer 实例接收



SpringCloud Stream 消息分区

重点步骤 - Producer 属性配置

```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          name-server: 49.235.54.112:9876
      bindings:
        output:
          destination: topic-test-stream-partition
          producer:
            partition-key-expression: headers['partitionKey'] - 1
            partition-count: 4
```

SpringCloud Stream 消息分区

重点步骤 - Consumer 属性配置

```
spring:
  cloud:
    stream:
      rocketmq:
        binder:
          name-server: 49.235.54.112:9876
      bindings:
        input:
          destination: topic-test-stream-partition
          group: stream-test-partition
          consumer:
            partitioned: true
      instance-index: 0
      instance-count: 4
```

SpringCloud Stream 消息分区

重点步骤 -验证方式

启动 4 个 Consumer 实例：

```
java -jar consumer.jar --server.port=9001 --spring.cloud.stream.instanceIndex=0
java -jar consumer.jar --server.port=9002 --spring.cloud.stream.instanceIndex=1
java -jar consumer.jar --server.port=9003 --spring.cloud.stream.instanceIndex=2
java -jar consumer.jar --server.port=9004 --spring.cloud.stream.instanceIndex=3
```

启动 Producer，访问 `http://localhost:8001/produce`，启动消息发送

查看 4 个 Consumer 的控制台，每个 Consumer 应接收到字符长度相同的消息。

SpringCloud Stream 消息分区-总结

重难点

1. 理解 SpringCloud Stream 中 消费分区 的作用、使用场景
2. 掌握 消费分区 的配置与开发方法
3. 了解不同 MQ 对于 SpringCloud Stream 支持的差异性

一样的在线教育，不一样的教学品质