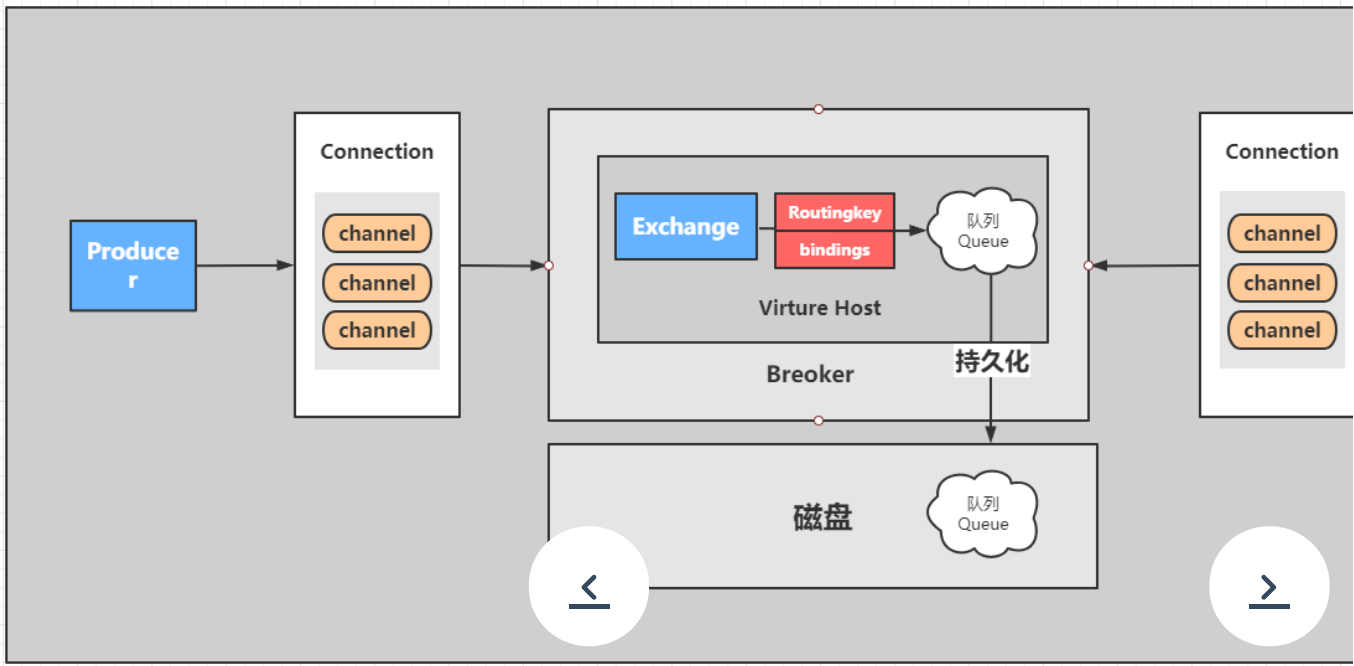


RabbitMQ-SpringBoot案例 -direct模式

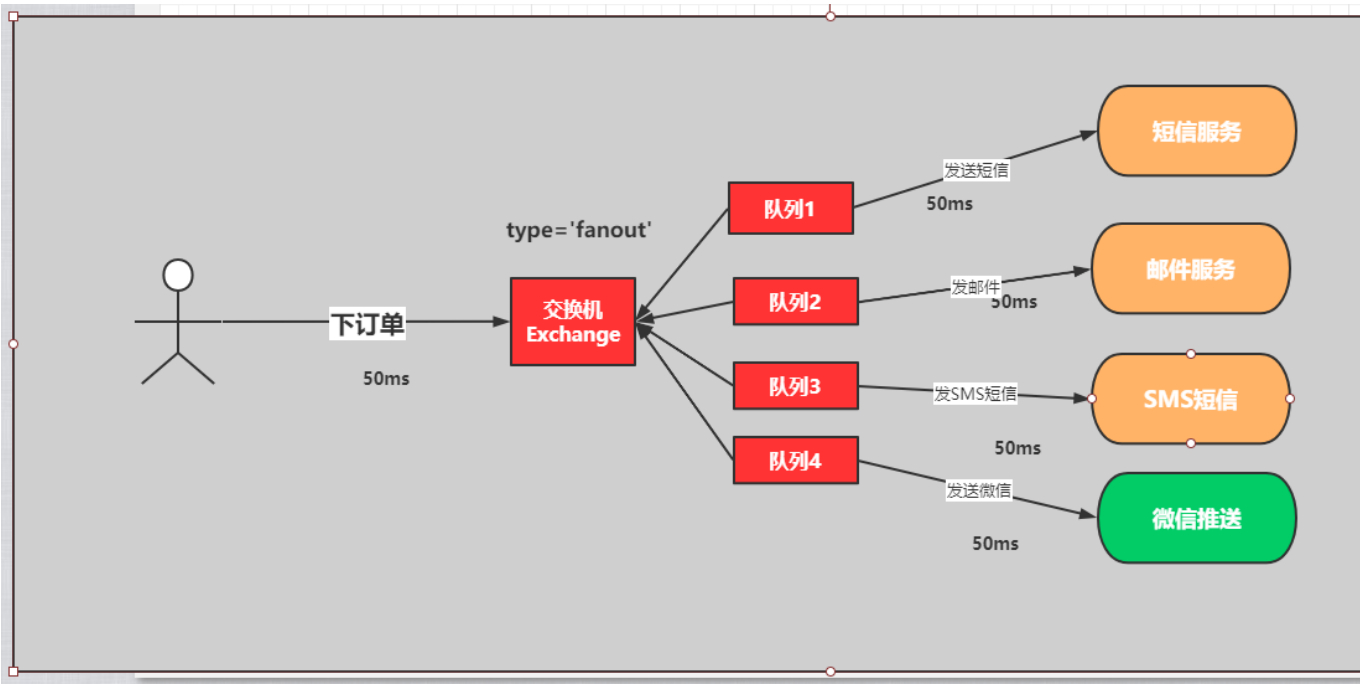
飞哥 VIP 分类: 学习笔记 创建时间: 2021/03/02 20:07 字体 皮肤

整体核心



01、目标

使用springboot完成rabbitmq的消费模式-Fanout



02、实现步骤

- 1: 创建生产者工程: sspringboot-rabbitmq-direct-producer
- 2: 创建消费者工程: springboot-rabbitmq-direct-consumer
- 3: 引入spring-boot-rabbitmq的依赖
- 4: 进行消息的分发和测试
- 5: 查看和观察web控制台的情况

整体核心

01、目标

02、实现步骤

具体实现

03、生产者

1、创建生产者工程: sspringboot-rabbit

2、在pom.xml中引入依赖

3、在application.yml进行配置

4: 定义订单的生产者

4、绑定关系

5、进行测试

04、定义消费者

1、创建消费者工程: springboot-rabbitn

2、引入依赖pom.xml

3、在application.yml进行配置

4、消费者 - 邮件服务

5、消费者 - 短信服务

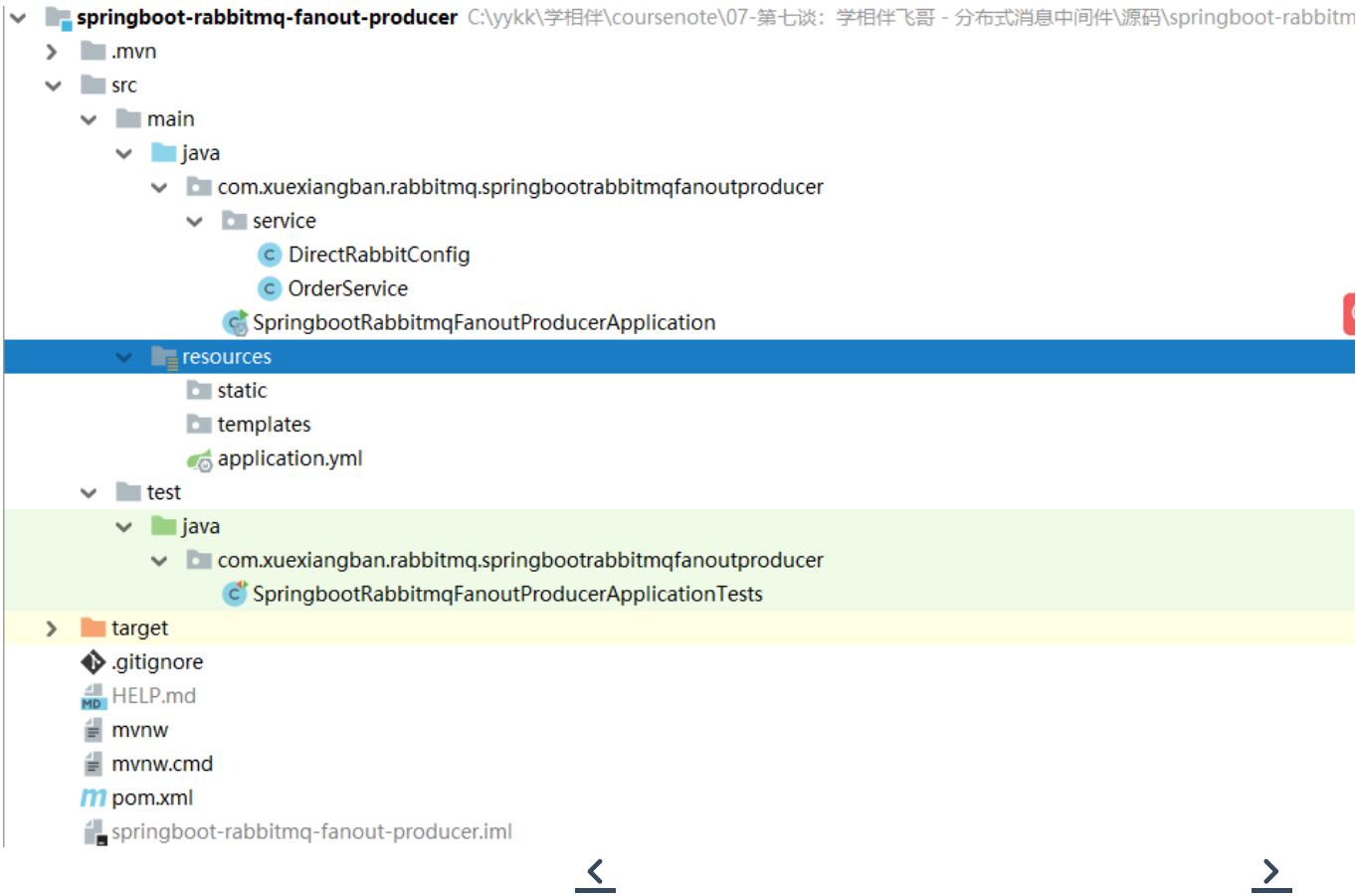
6、消费者 - 微信服务

7、启动服务SpringbootRabbitmqFanou

具体实现

03、生产者

1、创建生产者工程：sspringboot-rabbitmq-direct-producer



2、在pom.xml中引入依赖

```
1. <dependency>
2.     <groupId>org.springframework.boot</groupId>
3.     <artifactId>spring-boot-starter-amqp</artifactId>
4. </dependency>
5. <dependency>
6.     <groupId>org.springframework.boot</groupId>
7.     <artifactId>spring-boot-starter-web</artifactId>
8. </dependency>
```

3、在application.yml进行配置

```
1. # 服务端口
2. server:
3.     port: 8080
4.
5. # 配置rabbitmq服务
6. spring:
7.     rabbitmq:
8.         username: admin
9.         password: admin
10.        virtual-host: /
11.        host: 47.104.141.27
12.        port: 5672
```

4：定义订单的生产者

整体核心

01、目标

02、实现步骤

具体实现

03、生产者

1、创建生产者工程：sspringboot-rabbitmq-direct-producer

2、在pom.xml中引入依赖

3、在application.yml进行配置

4：定义订单的生产者

4、绑定关系

5、进行测试

04、定义消费者

1、创建消费者工程：springboot-rabbitmq-direct-consumer

2、引入依赖pom.xml

3、在application.yml进行配置

4、消费者 - 邮件服务

5、消费者 - 短信服务

6、消费者 - 微信服务

7、启动服务SpringbootRabbitmqFanoutProducerApplication



```
1. package com.xuexiangban.rabbitmq.springbootrabbitmqfanoutproducer.service;
2.
3. import org.springframework.amqp.rabbit.core.RabbitTemplate;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import org.springframework.stereotype.Component;
6.
7. import java.util.UUID;
8.
9. /**
10.  * @author: 学相伴-飞哥
11.  * @description: OrderService
12.  * @Date : 2021/3/4
13.  */
14. @Component
15. public class OrderService {
16.
17.     @Autowired
18.     private RabbitTemplate rabbitTemplate;
19.     // 1: 定义交换机
20.     private String exchangeName = "direct_order_exchange";
21.     // 2: 路由key
22.     private String routeKey = "";
23.
24.
25.     public void makeOrder(Long userId, Long productId, int num) {
26.         // 1: 模拟用户下单
27.         String orderNumer = UUID.randomUUID().toString();
28.         // 2: 根据商品id productId 去查询商品的库存
29.         // int numstore = productService.getProductNum(productId);
30.         // 3:判断库存是否充足
31.         // if(num > numstore ){ return "商品库存不足..."; }
32.         // 4: 下单逻辑
33.         // orderService.saveOrder(order);
34.         // 5: 下单成功要扣减库存
35.         // 6: 下单完成以后
36.         System.out.println("用户 " + userId + ",订单编号是:" + orderNumer);
37.         // 发送订单信息给RabbitMQ fanout
38.         rabbitTemplate.convertAndSend(exchangeName, routeKey, orderNumer);
39.     }
40. }
```

整体核心

01、目标

02、实现步骤



具体实现

03、生产者

1、创建生产者工程: sspringboot-rabbit

2、在pom.xml中引入依赖

3、在application.yml进行配置

4: 定义订单的生产者

4、绑定关系

5、进行测试

04、定义消费者

1、创建消费者工程: springboot-rabbitn

2、引入依赖pom.xml

3、在application.yml进行配置

4、消费者 - 邮件服务

5、消费者 - 短信服务

6、消费者 - 微信服务

7、启动服务SpringbootRabbitmqFanou

4、绑定关系



```
1. package com.xuexiangban.rabbitmq.springbootrabbitmqfanoutproducer.service;
2.
3. import org.springframework.amqp.core.Binding;
4. import org.springframework.amqp.core.BindingBuilder;
5. import org.springframework.amqp.core.DirectExchange;
6. import org.springframework.amqp.core.Queue;
7. import org.springframework.context.annotation.Bean;
8. import org.springframework.context.annotation.Configuration;
9.
10. /**
11.  * @Author : JCccc
12.  * @CreateTime : 2019/9/3
13.  * @Description :
14.  */
15. @Configuration
16. public class DirectRabbitConfig {
17.
18.     //队列 起名 : TestDirectQueue
19.     @Bean
20.     public Queue emailQueue() {
21.         // durable:是否持久化,默认是false,持久化队列:会被存储在磁盘上,当消息代理重启时
22.         // exclusive:默认也是false,只能被当前创建的连接使用,而且当连接关闭后队列即被删除
23.         // autoDelete:是否自动删除,当没有生产者或者消费者使用此队列,该队列会自动删除。
24.         // return new Queue("TestDirectQueue",true,true,false);
25.
26.         //一般设置一下队列的持久化就好,其余两个就是默认false
27.         return new Queue("email.fanout.queue", true);
28.     }
29.
30.     @Bean
31.     public Queue smsQueue() {
32.         // durable:是否持久化,默认是false,持久化队列:会被存储在磁盘上,当消息代理重启时
33.         // exclusive:默认也是false,只能被当前创建的连接使用,而且当连接关闭后队列即被删除
34.         // autoDelete:是否自动删除,当没有生产者或者消费者使用此队列,该队列会自动删除。
35.         // return new Queue("TestDirectQueue",true,true,false);
36.
37.         //一般设置一下队列的持久化就好,其余两个就是默认false
38.         return new Queue("sms.fanout.queue", true);
39.     }
40.
41.     @Bean
42.     public Queue weixinQueue() {
43.         // durable:是否持久化,默认是false,持久化队列:会被存储在磁盘上,当消息代理重启时
44.         // exclusive:默认也是false,只能被当前创建的连接使用,而且当连接关闭后队列即被删除
45.         // autoDelete:是否自动删除,当没有生产者或者消费者使用此队列,该队列会自动删除。
46.         // return new Queue("TestDirectQueue",true,true,false);
47.
48.         //一般设置一下队列的持久化就好,其余两个就是默认false
49.         return new Queue("weixin.fanout.queue", true);
50.     }
51.
52.     //Direct交换机 起名 : TestDirectExchange
53.     @Bean
54.     public DirectExchange directOrderExchange() {
55.         // return new DirectExchange("TestDirectExchange",true,true);
56.         return new DirectExchange("direct_order_exchange", true, false);
57.     }
58.
59.     //绑定 将队列和交换机绑定,并设置用于匹配键 : TestDirectRouting
60.     @Bean
61.     public Binding bindingDirect1() {
62.         return BindingBuilder.bind(weixinQueue()).to(directOrderExchange()).with
```

整体核心

01、目标

02、实现步骤

> 具体实现

03、生产者

1、创建生产者工程: sspringboot-rabbit

2、在pom.xml中引入依赖

3、在application.yml进行配置

4: 定义订单的生产者

4、绑定关系

5、进行测试

04、定义消费者

1、创建消费者工程: springboot-rabbitn

2、引入依赖pom.xml

3、在application.yml进行配置

4、消费者 - 邮件服务

5、消费者 - 短信服务

6、消费者 - 微信服务

7、启动服务SpringbootRabbitmqFanou

```
63.     }
64.
65.     @Bean
66.     public Binding bindingDirect2() {
67.         return BindingBuilder.bind(smsQueue()).to(directOrderExchange()).with("
68.     }
69.
70.     @Bean
71.     public Binding bindingDirect3() {
72.         return BindingBuilder.bind(emailQueue()).to(directOrderExchange()).with(
73.     }
74.
75. }
```

整体核心

01、目标

02、实现步骤

具体实现

03、生产者

1、创建生产者工程：sspringboot-rabbit

2、在pom.xml中引入依赖

3、在application.yml进行配置

4、定义订单的生产者

4、绑定关系

5、进行测试

04、定义消费者

1、创建消费者工程：springboot-rabbitn

2、引入依赖pom.xml

3、在application.yml进行配置

4、消费者 - 邮件服务

5、消费者 - 短信服务

6、消费者 - 微信服务

7、启动服务SpringbootRabbitmqFanou

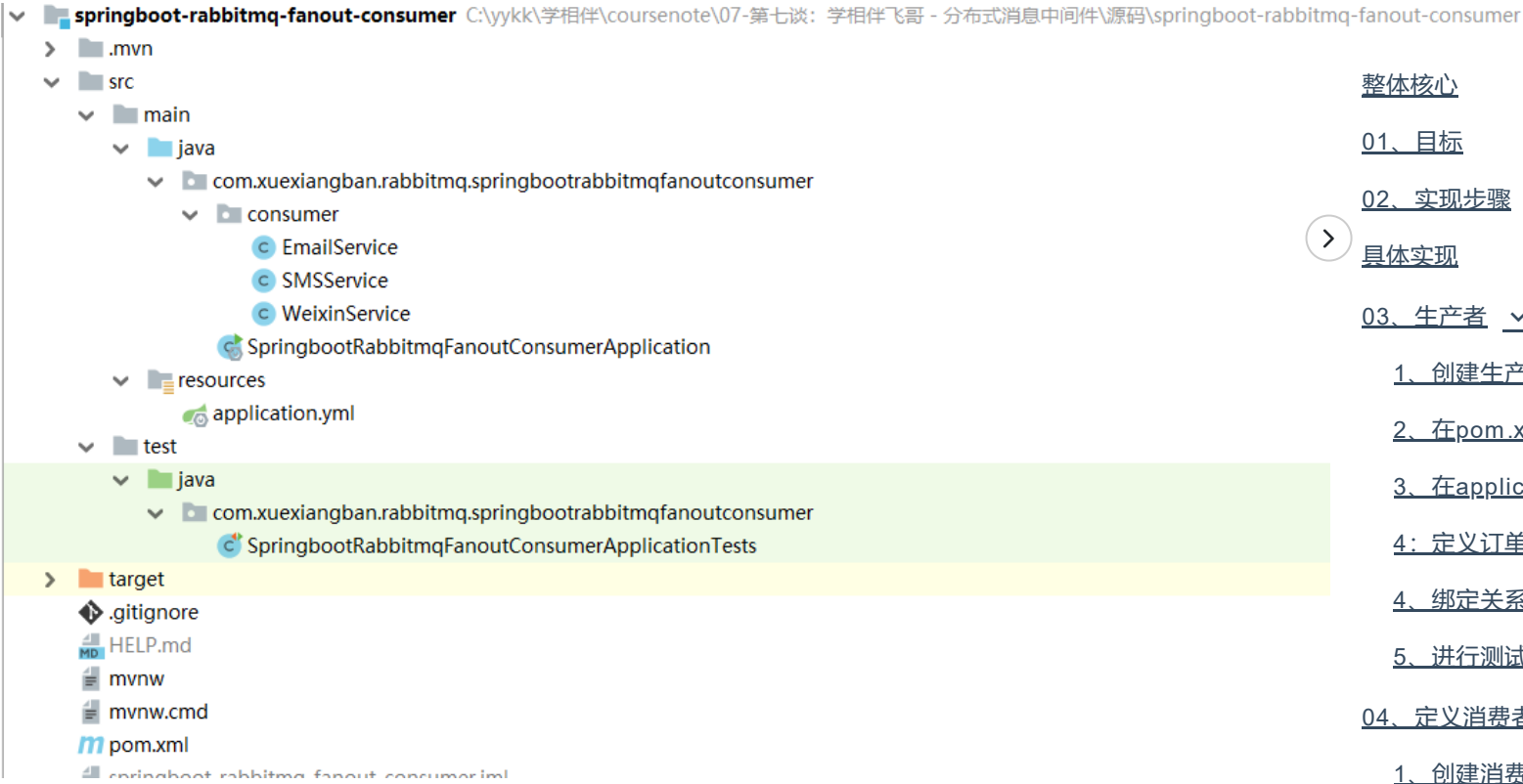
5、进行测试

```
1. package com.xuexiangban.rabbitmq.springbootrabbitmqfanoutproducer;
2.
3. import com.xuexiangban.rabbitmq.springbootrabbitmqfanoutproducer.service.OrderSe
4. import org.junit.jupiter.api.Test;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.boot.test.context.SpringBootTest;
7.
8. @SpringBootTest
9. class SpringbootRabbitmqFanoutProducerApplicationTests {
10.
11.     @Autowired
12.     OrderService orderService;
13.
14.     @Test
15.     public void contextLoads() throws Exception {
16.
17.         for (int i = 0; i < 10; i++) {
18.             Thread.sleep(1000);
19.             Long userId = 100L + i;
20.             Long productId = 10001L + i;
21.             int num = 10;
22.             orderService.makeOrder(userId, productId, num);
23.         }
24.     }
25.
26. }
```

04、定义消费者

1、创建消费者工程：springboot-rabbitmq-fanout-consumer





整体核心

01、目标

02、实现步骤

具体实现

03、生产者

1、创建生产者工程：[sspringboot-rabbit](#)

2、在pom.xml中引入依赖

3、在application.yml进行配置

4、定义订单的生产者

4、绑定关系

5、进行测试

04、定义消费者

1、创建消费者工程：[springboot-rabbitn](#)

2、引入依赖pom.xml

3、在application.yml进行配置

4、消费者 - 邮件服务

5、消费者 - 短信服务

6、消费者 - 微信服务

7、启动服务[SpringbootRabbitmqFanou](#)

2、引入依赖pom.xml

```
1. <dependency>
2.     <groupId>org.springframework.boot</groupId>
3.     <artifactId>spring-boot-starter-amqp</artifactId>
4. </dependency>
5. <dependency>
6.     <groupId>org.springframework.boot</groupId>
7.     <artifactId>spring-boot-starter-web</artifactId>
8. </dependency>
```

3、在application.yml进行配置

```
1. # 服务端口
2. server:
3.     port: 8081
4.
5. # 配置rabbitmq服务
6. spring:
7.     rabbitmq:
8.         username: admin
9.         password: admin
10.        virtual-host: /
11.        host: 47.104.141.27
12.        port: 5672
```

4、消费者 - 邮件服务





```
1. package com.xuexiangban.rabbitmq.springbootrabbitmqfanoutconsumer.consumer;
2.
3. import org.springframework.amqp.core.ExchangeTypes;
4. import org.springframework.amqp.rabbit.annotation.*;
5. import org.springframework.stereotype.Component;
6.
7. // bindings其实就是用来确定队列和交换机绑定关系
8. @RabbitListener(bindings = @QueueBinding(
9.     // email.fanout.queue 是队列名字, 这个名字你可以自定随便定义。
10.    value = @Queue(value = "email.fanout.queue",autoDelete = "false"),
11.    // order.fanout 交换机的名字 必须和生产者保持一致
12.    exchange = @Exchange(value = "fanout_order_exchange",
13.        // 这里是确定的rabbitmq模式是: fanout 是以广播模式 、 发布订阅模式
14.        type = ExchangeTypes.FANOUT)
15. ))
16. @Component
17. public class EmailService {
18.     // @RabbitHandler 代表此方法是一个消息接收的方法。该不要有返回值
19.     @RabbitHandler
20.     public void messagerevice(String message){
21.         // 此处省略发邮件的逻辑
22.         System.out.println("email----->" + message);
23.     }
24. }
```

整体核心

01、目标

02、实现步骤



具体实现

03、生产者

1、创建生产者工程: sspringboot-rabbit

2、在pom.xml中引入依赖

3、在application.yml进行配置

4: 定义订单的生产者

4、绑定关系

5、进行测试

04、定义消费者

1、创建消费者工程: springboot-rabbitn

2、引入依赖pom.xml

3、在application.yml进行配置

4、消费者 - 邮件服务

5、消费者 - 短信服务

6、消费者 - 微信服务

7、启动服务SpringbootRabbitmqFanou

5、消费者 - 短信服务

```
1. package com.xuexiangban.rabbitmq.springbootrabbitmqfanoutconsumer.consum
2.
3. import org.springframework.amqp.core.ExchangeTypes;
4. import org.springframework.amqp.rabbit.annotation.*;
5. import org.springframework.stereotype.Component;
6.
7. // bindings其实就是用来确定队列和交换机绑定关系
8. @RabbitListener(bindings = @QueueBinding(
9.     // email.fanout.queue 是队列名字, 这个名字你可以自定随便定义。
10.    value = @Queue(value = "sms.fanout.queue",autoDelete = "false"),
11.    // order.fanout 交换机的名字 必须和生产者保持一致
12.    exchange = @Exchange(value = "fanout_order_exchange",
13.        // 这里是确定的rabbitmq模式是: fanout 是以广播模式 、 发布订阅模式
14.        type = ExchangeTypes.FANOUT)
15. ))
16. @Component
17. public class SMSService {
18.
19.     // @RabbitHandler 代表此方法是一个消息接收的方法。该不要有返回值
20.     @RabbitHandler
21.     public void messagerevice(String message){
22.         // 此处省略发邮件的逻辑
23.         System.out.println("sms----->" + message);
24.     }
25. }
```

6、消费者 - 微信服务



```
1. package com.xuexiangban.rabbitmq.springbootrabbitmqfanoutconsumer.consumer;
2.
3. import org.springframework.amqp.core.ExchangeTypes;
4. import org.springframework.amqp.rabbit.annotation.*;
5. import org.springframework.stereotype.Component;
6.
7. // bindings其实就是用来确定队列和交换机绑定关系
8. @RabbitListener(bindings = @QueueBinding(
9.     // email.fanout.queue 是队列名字, 这个名字你可以自定随便定义。
10.    value = @Queue(value = "weixin.fanout.queue",autoDelete = "false"),
11.    // order.fanout 交换机的名字 必须和生产者保持一致
12.    exchange = @Exchange(value = "fanout_order_exchange",
13.        // 这里是确定的rabbitmq模式是 : fanout 是以广播模式 、 发布订阅模式
14.        type = ExchangeTypes.FANOUT)
15. ))
16. @Component
17. public class WeixinService {
18.
19.    // @RabbitHandler 代表此方法是一个消息接收的方法。该不要有返回值
20.    @RabbitHandler
21.    public void messagerevice(String message){
22.        // 此处省略发邮件的逻辑
23.        System.out.println("weixin----->" + message);
24.    }
25. }
```

整体核心

01、目标

02、实现步骤

具体实现

03、生产者

1、创建生产者工程: sspringboot-rabbit

2、在pom.xml中引入依赖

3、在application.yml进行配置

4: 定义订单的生产者

4、绑定关系

5、进行测试

04、定义消费者

1、创建消费者工程: springboot-rabbitn

2、引入依赖pom.xml

3、在application.yml进行配置

4、消费者 - 邮件服务

5、消费者 - 短信服务

6、消费者 - 微信服务

7、启动服务SpringbootRabbitmqFanou

7、启动服务SpringbootRabbitmqFanoutConsumerApplication, 查看效果

