Spring boot2.0集成开源框架

**1spring boot 集成rabbitmq**

**主要作用：异步、解耦与消峰。**

**主要有两种模式：**

**简单模式**：一个生产者，一个消费者。队列对应消息全部由改消费者消费。应用场景:聊天，还有需要保证消费顺序时使用。

**work模式**：一个生产者，多个消费者。每个消费者获取到的**消息唯一（资源竞争）**。应用场景:红包;大项目中的资源调度(任务分配系统不需知道哪一个任务执行系统在空闲,直接将任务扔到消息队列中,空闲的系统自动争抢)。

以下几个关于rabbitmq测试都使用这个pom.xml

配置参数见<https://www.cnblogs.com/qts-hope/p/11242559.html>

pom.xml引入rabbitmq相关jar包

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.3.0.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.zlk</groupId>  
 <artifactId>mq</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>mq</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-amqp</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 <exclusions>  
 <exclusion>  
 <groupId>org.junit.vintage</groupId>  
 <artifactId>junit-vintage-engine</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.amqp</groupId>  
 <artifactId>spring-rabbit-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

## 1.1集成rabbitMQ(直接使用队列名称发送)

默认的direct交换器，用队列名称当做路由键。

(1)application.properties

spring.application.name=springboot\_mq  
server.port=8011  
  
# rabbitmq配置文件  
spring.rabbitmq.host=192.168.120.201  
spring.rabbitmq.port=5672  
spring.rabbitmq.username=guest  
spring.rabbitmq.password=guest

1. 编码（入门例子，**直接使用队列来接收**。先启动消费者，再启动生产者）

队列hello-mq-queue的配置类。

package com.zlk.mq.rabbitmq;  
  
import org.springframework.amqp.core.Queue;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;

@Configuration  
public class SenderConfig {  
 @Bean  
 public Queue queue() {  
 //hello-mq-queue为队列名称  
 return new Queue("hello-mq-queue");  
 }  
}

生产者，向队列hello-mq-queue发送数据。使用AmqpTemplate

package com.zlk.mq.rabbitmq;

import org.springframework.amqp.core.AmqpTemplate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 发送者:发送消息到队列  
 \*/*@Component  
public class Sender {  
  
 */\*\*  
 \* AmqpTemplate由spring提供  
 \*/* @Autowired  
 private AmqpTemplate rabbitTemplate;  
  
 */\*\*  
 \* 发送消息  
 \*/* public void send() {  
 String msg = "你好，这是一条消息";  
 //将消息发送到队列hello-mq-queue  
 rabbitTemplate.convertAndSend("hello-mq-queue", msg);  
 }  
  
}

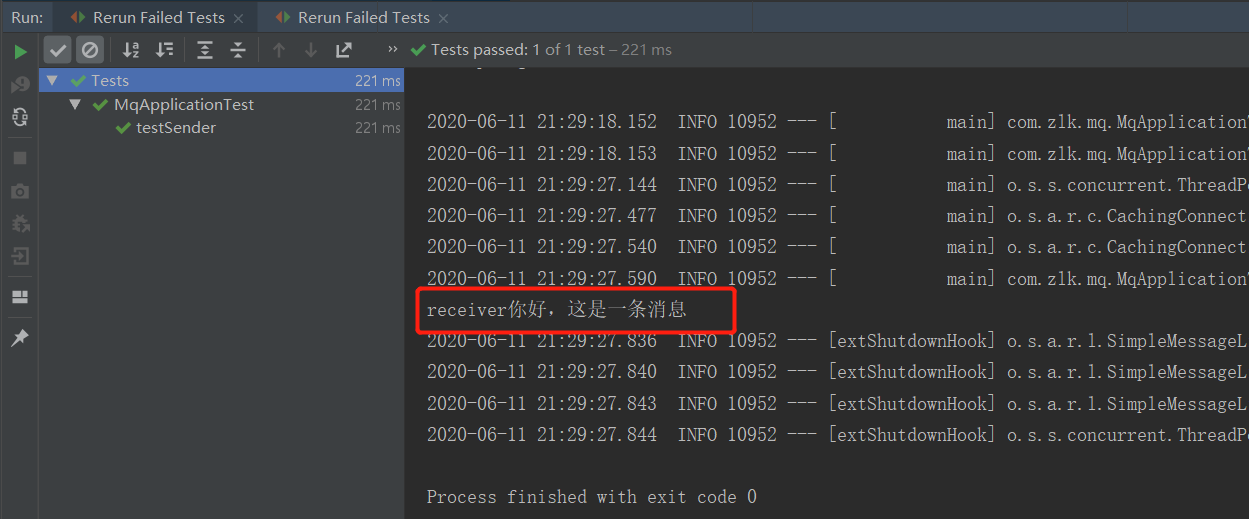
消费者，使用@RabbitListener(queues = "hello-mq-queue")来监听并消费hello-mq-queue队列中的消息。

package com.zlk.mq.rabbitmq;  
  
import org.springframework.amqp.core.AmqpTemplate;  
import org.springframework.amqp.rabbit.annotation.RabbitListener;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 接受者：从队列消费消息  
 \*/*@Component  
public class Receiver {  
  
 @Autowired  
 private AmqpTemplate rabbitTemplate;  
  
 */\*\*  
 \* 从消息队列接收消息  
 \*/* @RabbitListener(queues = "hello-mq-queue")  
 public void process(String msg) {  
 System.*out*.println("receiver" + msg);  
 }  
}

测试类：

package com.zlk.mq;   
import com.zlk.mq.rabbitmq.Sender;  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.test.context.junit4.SpringRunner;  
  
@RunWith(SpringRunner.class)  
@SpringBootTest  
public class MqApplicationTest {  
 @Autowired  
 private Sender sender;  
  
 @Test  
 public void testSender() {  
 sender.send();  
 }  
}

测试结果



## 1.2 交换器实战（exchange）

发布-订阅（direct）：路由键完全匹配，路由键绑定队列与交换器。

主题（topic）：也是通过路由键（模糊匹配，\*一个，#一个或者多个）规则绑定交换器与队列。

广播（fanout） ：消息将群发到当前交换器下绑定的所有队列上。该交换器队列与交换器不通过路由器绑定。

headers：通过headers来决定把消息发给哪些queue，用的比较少。

### 1.2.1 发布-订阅（direct）

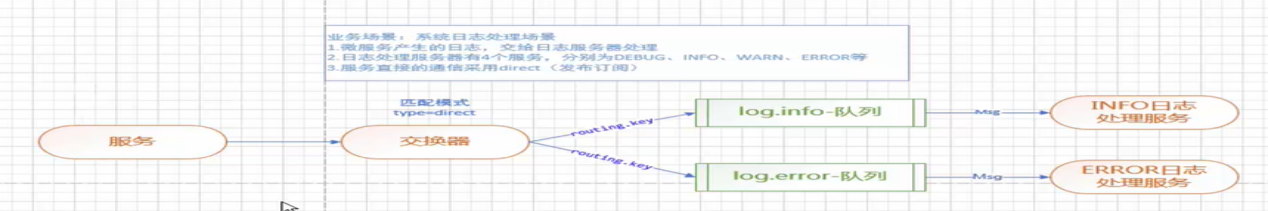
发布-订阅（direct）：路由键完全匹配，**路由键绑定队列与交换器**。

  处理路由键。需要将一个队列绑定到交换器（通过路由键）上，要求该消息与一个特定的路由键完全匹配。这是一个完整的匹配。如果一个队列绑定到该交换器上要求路由键 “test”，则只有被标记为“test”的消息才被转发，不会转发test.aaa，也不会转发dog.123，只会转发test。

当接收端订阅者有多个的时候，direct会轮询公平的分发给每个订阅者（订阅者消息确认正常）

应用场景：用户通知，当用户充值成功或转账完成系统通知用户，通知方式有短信、邮件多种方法；队列少的时候比较适用。

多个消费者时，队列消费消息也会保证唯一消费（注：不含重复消息情况。指一个队列，多个消费者时，消息不会被队列重复消费）。但是**与路由键和交换器绑定的队列可以有多个（都会接受到对应的完整信息），就是说可以有多个队列（消费者：路由键和交换器绑定的队列）来接收完整消息（生产者：路由键和交换器绑定的消息）**。



生产者发送消息时绑定路由键与交换器。

在当前**交换器下**，只有完全**匹配上路由键的队列**才能接收到消息。

1. **消费者服务**

application.properties

spring.application.name=mq\_direct\_consumer  
server.port=8013  
# rabbitmq配置文件  
spring.rabbitmq.host=192.168.120.201  
spring.rabbitmq.port=5672  
spring.rabbitmq.username=guest  
spring.rabbitmq.password=guest  
  
# rabbitmq交换器  
mq.config.exchange=log.direct  
  
# 队列名称  
mq.config.queue.info=log.info  
# 路由键  
mq.config.queue.info.routing.key=log.info.routing.key  
  
mq.config.queue.error=log.error  
mq.config.queue.error.routing.key=log.error.routing.key

消费者通交换器与路由键，完全匹配的方式去匹配队列。

消费者1:接收info日志

package com.zlk.mq.direct;  
  
import org.springframework.amqp.core.ExchangeTypes;  
import org.springframework.amqp.rabbit.annotation.\*;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 接受者：从队列消费消息(Info日志处理)  
 \** ***@RabbitListener*** *绑定队列  
 \*  
 \* autoDelete = "true" 临时队列，autoDelete = "false" 则开启持久化  
 \* type = ExchangeTypes.DIRECT 交换器类型 消息-订阅  
 \* key 路由键  
 \*/*@Component  
@RabbitListener(  
 bindings = @QueueBinding(  
 value = @Queue(value = "${mq.config.queue.info}",autoDelete = "true"),  
 exchange = @Exchange(value = "${mq.config.exchange}",type = ExchangeTypes.*DIRECT*),  
 key = "${mq.config.queue.info.routing.key}"  
 )  
)  
public class InfoReceiver {  
  
 */\*\*  
 \* 从消息队列接收info日志  
 \*/* @RabbitHandler  
 public void process(String msg) {  
 System.*out*.println("从消息队列接收info日志：" + msg);  
 }  
}

消费者2:接收error日志

package com.zlk.mq.direct;  
  
import org.springframework.amqp.core.ExchangeTypes;  
import org.springframework.amqp.rabbit.annotation.\*;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 接受者：从队列消费消息(error日志处理)  
 \** ***@RabbitListener*** *绑定队列  
 \*  
 \* autoDelete = "true" 临时队列，autoDelete = "false" 则开启持久化  
 \* type = ExchangeTypes.DIRECT 交换器类型 消息-订阅  
 \* key 路由键  
 \*  
 \*/*@Component  
@RabbitListener(  
 bindings = @QueueBinding(  
 value = @Queue(value = "${mq.config.queue.error}",autoDelete = "true"),  
 exchange = @Exchange(value = "${mq.config.exchange}",type = ExchangeTypes.*DIRECT*),  
 key = "${mq.config.queue.error.routing.key}"  
 )  
)  
public class ErrorReceiver {  
  
 */\*\*  
 \* 从消息队列接收error日志  
 \*/* @RabbitHandler  
 public void process(String msg) {  
 System.*out*.println("从消息队列接收error日志：" + msg);  
 }  
}

1. **生产者服务**

application.properties

spring.application.name=mq\_direct\_provider  
server.port=8026  
# rabbitmq配置文件  
spring.rabbitmq.host=192.168.120.201  
spring.rabbitmq.port=5672  
spring.rabbitmq.username=guest  
spring.rabbitmq.password=guest  
  
# rabbitmq交换器  
mq.config.exchange=log.direct

生产者，消息订阅方式需要交换器，路由键。

package com.zlk.mq;  
  
import org.springframework.amqp.core.AmqpTemplate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 发送者:发送消息到队列  
 \*/*@Component  
public class Sender {  
  
 */\*\*  
 \* AmqpTemplate由spring提供  
 \*/* @Autowired  
 private AmqpTemplate rabbitTemplate;  
  
 @Value("${mq.config.exchange}")  
 private String exchange;  
 */\*\*  
 \* 发送消息  
 \*/* public void send() {  
 //通过交换器与路由键 匹配 交换器与路由键绑定的消息队列  
 rabbitTemplate.convertAndSend(this.exchange,"log.error.routing.key", "错误日志........");  
 rabbitTemplate.convertAndSend(this.exchange,"log.info.routing.key", "普通日志.......");  
 }  
  
}

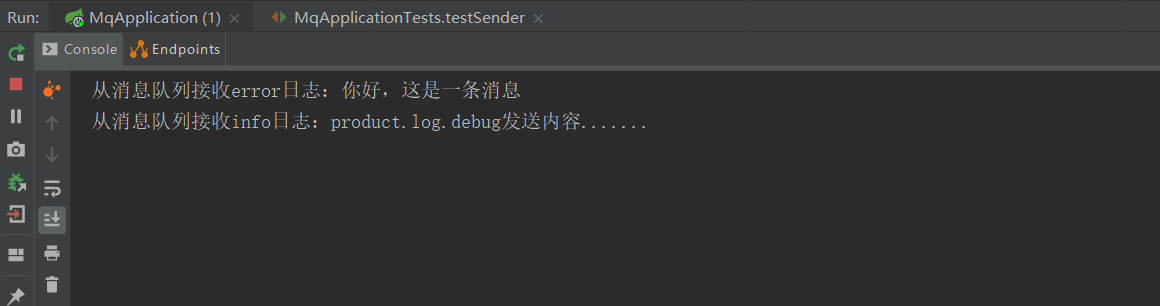
测试

package com.zlk.mq;  
  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.test.context.junit4.SpringRunner;  
  
@RunWith(SpringRunner.class)  
@SpringBootTest  
public class MqApplicationTests {  
  
 @Autowired  
 private Sender sender;  
  
 @Test  
 public void testSender() {  
 sender.send();  
 }  
}

测试结果

生产者发送消息时绑定路由键与交换器。

在当前**交换器下**，只有完全**匹配上路由键的队列**才能接收到消息。

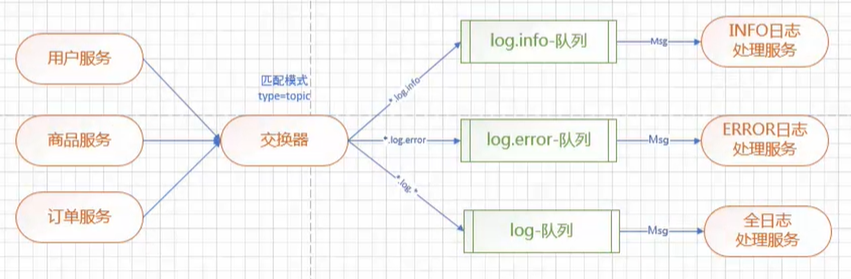


### **1.2.2主题（topic）**

主题（topic）：也是通过路由键（模糊匹配，\*一个，#一个或者多个）规则绑定交换器与队列。

将路由键和某模式进行匹配。此时队列需要绑定要一个模式上。符号“#”匹配一个或多个词，符号“\*”匹配不多不少一个词。因此“audit.#”能够匹配到“audit.irs.corporate”类型，但是“audit.\*” 只会匹配到“audit.irs”类型。

多个消费者时，队列消费消息也会保证唯一消费（注：不含重复消息情况。指一个队列，多个消费者时，消息不会被队列重复消费）。但是**与路由键和交换器绑定的队列可以有多个（都会接受到对应的完整信息），就是说可以有多个队列（消费者：路由键（模糊匹配，\*一个，#一个或者多个）和交换器绑定的队列）来接收完整消息（生产者：路由键和交换器绑定的消息，消费者队列模糊匹配上规则才行）。**



1. **消费者**

application.properties

spring.application.name=mq\_topic\_consumer  
server.port=8013  
# rabbitmq配置文件  
spring.rabbitmq.host=192.168.120.201  
spring.rabbitmq.port=5672  
spring.rabbitmq.username=guest  
spring.rabbitmq.password=guest  
  
# rabbitmq交换器  
mq.config.exchange=log.topic  
  
# 队列名称  
mq.config.queue.info=log.info  
mq.config.queue.error=log.error  
# 全日志队列  
mq.config.queue.logs=log.msg

Error日志消费

package com.zlk.topic.mq;  
  
import org.springframework.amqp.core.ExchangeTypes;  
import org.springframework.amqp.rabbit.annotation.\*;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 接受者：从队列消费消息(error日志处理)  
 \** ***@RabbitListener*** *绑定队列  
 \* autoDelete = "true" 临时队列，autoDelete = "false" 则开启持久化  
 \* type = ExchangeTypes.TOPIC 交换器类型 主题  
 \* key = "\*.log.error" 路由键模糊匹配  
 \*  
 \*/*@Component  
@RabbitListener(  
 bindings = @QueueBinding(  
 value = @Queue(value = "${mq.config.queue.error}",autoDelete = "true"),  
 exchange = @Exchange(value = "${mq.config.exchange}",type = ExchangeTypes.*TOPIC*),  
 key = "\*.log.error"  
 )  
)  
public class ErrorReceiver {  
  
 */\*\*  
 \* 从消息队列接收error日志  
 \*/* @RabbitHandler  
 public void process(String msg) {  
 System.*out*.println("----------error------------从消息队列接收error日志" + msg);  
 }  
}

Info日志消费

package com.zlk.topic.mq;  
  
import org.springframework.amqp.core.ExchangeTypes;  
import org.springframework.amqp.rabbit.annotation.\*;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 接受者：从队列消费消息(Info日志处理)  
 \** ***@RabbitListener*** *绑定队列  
 \* autoDelete = "true" 临时队列，autoDelete = "false" 则开启持久化  
 \* type = ExchangeTypes.TOPIC 交换器类型 主题  
 \* key = "\*\*.log.info" 路由键模糊匹配  
 \*  
 \*/*@Component  
@RabbitListener(  
 bindings = @QueueBinding(  
 value = @Queue(value = "${mq.config.queue.info}",autoDelete = "true"),  
 exchange = @Exchange(value = "${mq.config.exchange}",type = ExchangeTypes.*TOPIC*),  
 key = "\*.log.info"  
 )  
)  
public class InfoReceiver {  
  
 */\*\*  
 \* 从消息队列接收info日志  
 \*/* @RabbitHandler  
 public void process(String msg) {  
 System.*out*.println("----------info------------从消息队列接收info日志" + msg);  
 }  
}

所有日志消费

package com.zlk.topic.mq;  
  
import org.springframework.amqp.core.ExchangeTypes;  
import org.springframework.amqp.rabbit.annotation.\*;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 接受者：从队列消费消息(所有日志处理)  
 \** ***@RabbitListener*** *绑定队列  
 \*  
 \* autoDelete = "true" 临时队列，autoDelete = "false" 则开启持久化  
 \* type = ExchangeTypes.TOPIC 交换器类型 主题  
 \* key = "\*\*.log.\*" 路由键模糊匹配  
 \*  
 \*/*@Component  
@RabbitListener(  
 bindings = @QueueBinding(  
 value = @Queue(value = "${mq.config.queue.logs}",autoDelete = "true"),  
 exchange = @Exchange(value = "${mq.config.exchange}",type = ExchangeTypes.*TOPIC*),  
 key = "\*.log.\*"  
 )  
)  
public class LogsReceiver {  
  
 */\*\*  
 \* 从消息队列接收所有日志  
 \*/* @RabbitHandler  
 public void process(String msg) {  
 System.*out*.println("----------logs------------从消息队列接收logs日志" + msg);  
 }  
}

1. 生产者

application.properties

spring.application.name=mq\_topic\_provider  
server.port=8026  
# rabbitmq配置文件  
spring.rabbitmq.host=192.168.120.201  
spring.rabbitmq.port=5672  
spring.rabbitmq.username=guest  
spring.rabbitmq.password=guest  
  
# rabbitmq交换器  
mq.config.exchange=log.topic

生产者1

package com.zlk.topic;  
  
import org.springframework.amqp.core.AmqpTemplate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 订单发送者:发送消息到队列  
 \*/*@Component  
public class OrderSender {  
  
 */\*\*  
 \* AmqpTemplate由spring提供  
 \*/* @Autowired  
 private AmqpTemplate rabbitTemplate;  
  
 @Value("${mq.config.exchange}")  
 private String exchange;  
 */\*\*  
 \* 发送消息  
 \*/* public void send() {  
 //通过交换器与路由键 匹配 交换器与路由键绑定的消息队列  
 rabbitTemplate.convertAndSend(this.exchange,"order.log.info", "order.log.info发送内容.......");  
 rabbitTemplate.convertAndSend(this.exchange,"order.log.debug", "order.log.debug发送内容.......");  
 rabbitTemplate.convertAndSend(this.exchange,"order.log.warn", "order.log.warn发送内容.......");  
 rabbitTemplate.convertAndSend(this.exchange,"order.log.error", "order.log.error发送内容.......");  
 }  
  
}

生产者2

package com.zlk.topic;  
  
import org.springframework.amqp.core.AmqpTemplate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 商品发送者:发送消息到队列  
 \*/*@Component  
public class ProductSender {  
  
 */\*\*  
 \* AmqpTemplate由spring提供  
 \*/* @Autowired  
 private AmqpTemplate rabbitTemplate;  
  
 @Value("${mq.config.exchange}")  
 private String exchange;  
 */\*\*  
 \* 发送消息  
 \*/* public void send() {  
 //通过交换器与路由键 匹配 交换器与路由键绑定的消息队列  
 rabbitTemplate.convertAndSend(this.exchange,"product.log.info", "product.log.info发送内容.......");  
 rabbitTemplate.convertAndSend(this.exchange,"product.log.debug", "product.log.debug发送内容.......");  
 rabbitTemplate.convertAndSend(this.exchange,"product.log.warn", "product.log.warn发送内容.......");  
 rabbitTemplate.convertAndSend(this.exchange,"product.log.error", "product.log.error发送内容.......");  
 }  
  
}

生产者3

package com.zlk.topic;  
  
import org.springframework.amqp.core.AmqpTemplate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 用户发送者:发送消息到队列  
 \*/*@Component  
public class UserSender {  
  
 */\*\*  
 \* AmqpTemplate由spring提供  
 \*/* @Autowired  
 private AmqpTemplate rabbitTemplate;  
  
 @Value("${mq.config.exchange}")  
 private String exchange;  
 */\*\*  
 \* 发送消息  
 \*/* public void send() {  
 //通过交换器与路由键 匹配 交换器与路由键绑定的消息队列  
 rabbitTemplate.convertAndSend(this.exchange,"user.log.info", "user.log.info发送内容.......");  
 rabbitTemplate.convertAndSend(this.exchange,"user.log.debug", "user.log.debug发送内容.......");  
 rabbitTemplate.convertAndSend(this.exchange,"user.log.warn", "user.log.warn发送内容.......");  
 rabbitTemplate.convertAndSend(this.exchange,"user.log.error", "user.log.error发送内容.......");  
 }  
  
}

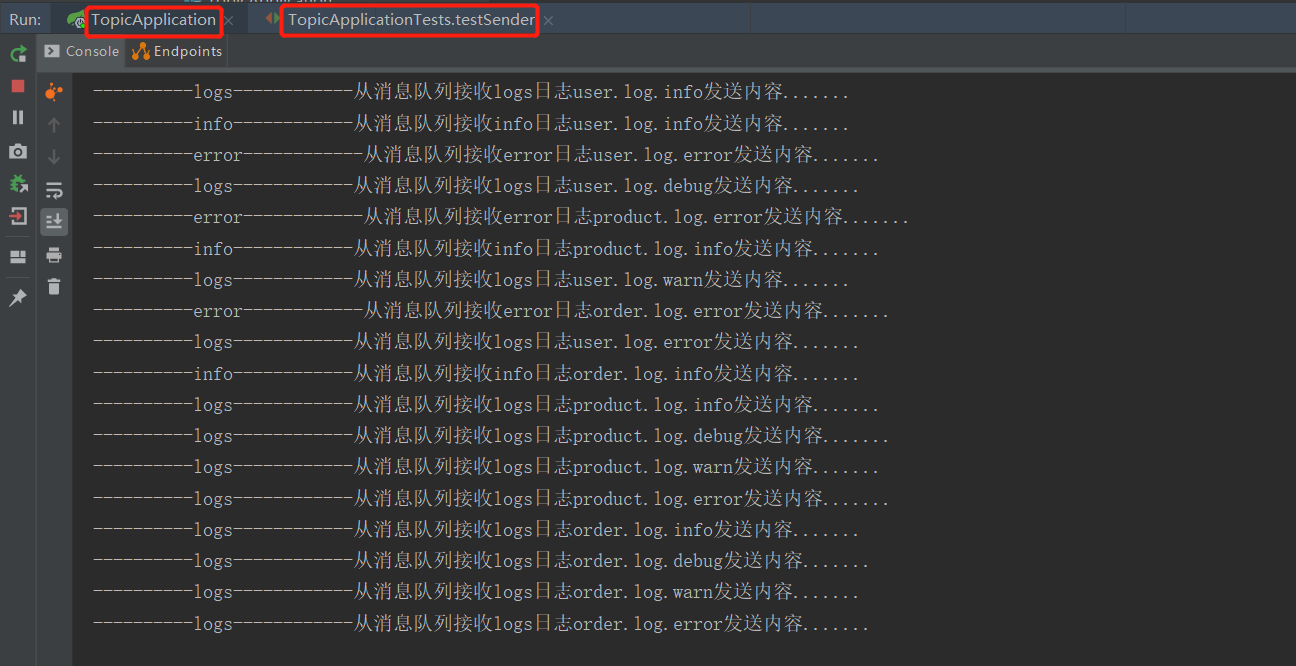
测试

package com.zlk.topic;  
  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.test.context.junit4.SpringRunner;  
  
  
@RunWith(SpringRunner.class)  
@SpringBootTest  
public class TopicApplicationTests {  
  
 @Autowired  
 private UserSender userSender;  
 @Autowired  
 private ProductSender productSender;  
 @Autowired  
 private OrderSender orderSender;  
  
 @Test  
 public void testSender() {  
 userSender.send();  
 productSender.send();  
 orderSender.send();  
 }  
}

测试结果：

生产者发送消息时绑定路由键与交换器。

在当前**交换器下**，只要**模糊匹配上路由键的队列**都将接收到消息。



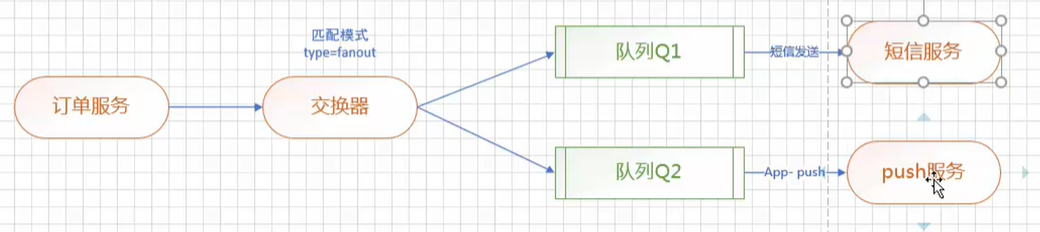
### **1.2.3 广播（fanout）**

广播（fanout） ：消息将群发到当前交换器下绑定的所有队列上。该交换器队列与交换器不通过路由器绑定。

  不处理路由键。你只需要简单的将队列绑定到交换器上。一个发送到交换器的消息都会被转发到与该交换器绑定的所有队列上。很像子网广播，每台子网内的主机都获得了一份复制的消息。Fanout交换器转发消息是最快的。

应用场景：用户通知，当用户充值成功或转账完成系统通知用户，通知方式有短信、邮件多种方法；

多个消费者时，队列消费消息也会保证唯一消费（注：不含重复消息情况。指一个队列，多个消费者时，消息不会被队列重复消费）。但是**交换器绑定的队列可以有多个（都会接受到对应的完整信息），就是说可以有多个队列（消费者：交换器绑定的队列）来接收完整消息（生产者：交换器绑定的消息）**。



1. **消费者**

application.properties

spring.application.name=mq\_topic\_consumer  
server.port=8028  
# rabbitmq配置文件  
spring.rabbitmq.host=192.168.120.201  
spring.rabbitmq.port=5672  
spring.rabbitmq.username=guest  
spring.rabbitmq.password=guest  
  
# rabbitmq交换器  
mq.config.exchange=order.fanout  
  
# 队列名称  
mq.config.queue.sms=order.sms  
mq.config.queue.push=order.push

消费者push处理。

package com.zlk.fanout.mq;  
  
import org.springframework.amqp.core.ExchangeTypes;  
import org.springframework.amqp.rabbit.annotation.\*;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 接受者：从队列消费消息(push日志处理)  
 \** ***@RabbitListener*** *绑定队列，路由键、交换器  
 \* \* autoDelete = "true" 临时队列，autoDelete = "false" 则开启持久化  
 \* type = ExchangeTypes.FANOUT 交换器类型 广播形式，  
 \* 不需要路由键key  
 \*  
 \*/*@Component  
@RabbitListener(  
 bindings = @QueueBinding(  
 value = @Queue(value = "${mq.config.queue.push}",autoDelete = "true"),  
 exchange = @Exchange(value = "${mq.config.exchange}",type = ExchangeTypes.*FANOUT*)  
 )  
)  
public class PushReceiver {  
  
 */\*\*  
 \* 从消息队列接收push日志  
 \*/* @RabbitHandler  
 public void process(String msg) {  
 System.*out*.println("push处理：" + msg);  
 }  
}

消费者短信处理

package com.zlk.fanout.mq;  
  
import org.springframework.amqp.core.ExchangeTypes;  
import org.springframework.amqp.rabbit.annotation.\*;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* 接受者：从队列消费消息(短信日志处理)  
 \** ***@RabbitListener*** *绑定队列，路由键、交换器  
 \* autoDelete = "true" 临时队列，autoDelete = "false" 则开启持久化  
 \* type = ExchangeTypes.FANOUT 交换器类型 广播形式，  
 \* 不需要路由键key  
 \*  
 \*/*@Component  
@RabbitListener(  
 bindings = @QueueBinding(  
 value = @Queue(value = "${mq.config.queue.sms}",autoDelete = "true"),  
 exchange = @Exchange(value = "${mq.config.exchange}",type = ExchangeTypes.*FANOUT*)  
 )  
)  
public class SmsReceiver {  
  
 */\*\*  
 \* 从消息队列接收sms日志  
 \*/* @RabbitHandler  
 public void process(String msg) {  
 System.*out*.println("sms处理:" + msg);  
 }  
}

1. **生产者**

application.properties

spring.application.name=mq\_topic\_provider  
server.port=8029  
# rabbitmq配置文件  
spring.rabbitmq.host=192.168.120.201  
spring.rabbitmq.port=5672  
spring.rabbitmq.username=guest  
spring.rabbitmq.password=guest  
  
# rabbitmq交换器  
mq.config.exchange=order.fanout

**生产者**

package com.zlk.fanout;  
  
import org.springframework.amqp.core.AmqpTemplate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
import java.util.ArrayList;  
  
*/\*\*  
 \* 订单发送者:发送消息到队列  
 \*/*@Component  
public class Sender {  
  
 */\*\*  
 \* AmqpTemplate由spring提供  
 \*/* @Autowired  
 private AmqpTemplate rabbitTemplate;  
  
 @Value("${mq.config.exchange}")  
 private String exchange;  
 */\*\*  
 \* 发送消息  
 \*/* public void send() {  
 //广播不要路由键，直接将消息绑定到交换器上  
 rabbitTemplate.convertAndSend(this.exchange,"","订单提交成功");  
 }  
}

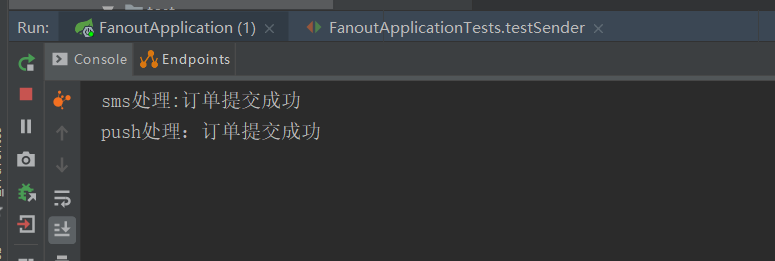
**测试**

package com.zlk.fanout;  
  
  
import org.junit.jupiter.api.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.test.context.junit4.SpringRunner;  
  
@RunWith(SpringRunner.class)  
@SpringBootTest  
class FanoutApplicationTests {  
  
 @Autowired  
 private Sender orderSender;  
  
 @Test  
 public void testSender() {  
 orderSender.send();  
 }  
}

**测试结果**

生产者发送消息时没有路由键，直接将消息绑定到交换器。

在当前**交换器下的所有队列**都将接收到**绑定到当前交换器消息**。



**单词备注**

## **fanout** [fænaut] n. 扇出；展开；分列（账户）；广播

**direct美 [dəˈrekt,daɪˈrekt]** adj. 直接的；直系的；亲身的；恰好的

* vt. 管理；指挥；导演；指向
* vi. 指导；指挥
* adv. 直接地；正好；按直系关系;
* 发布订阅。