目录

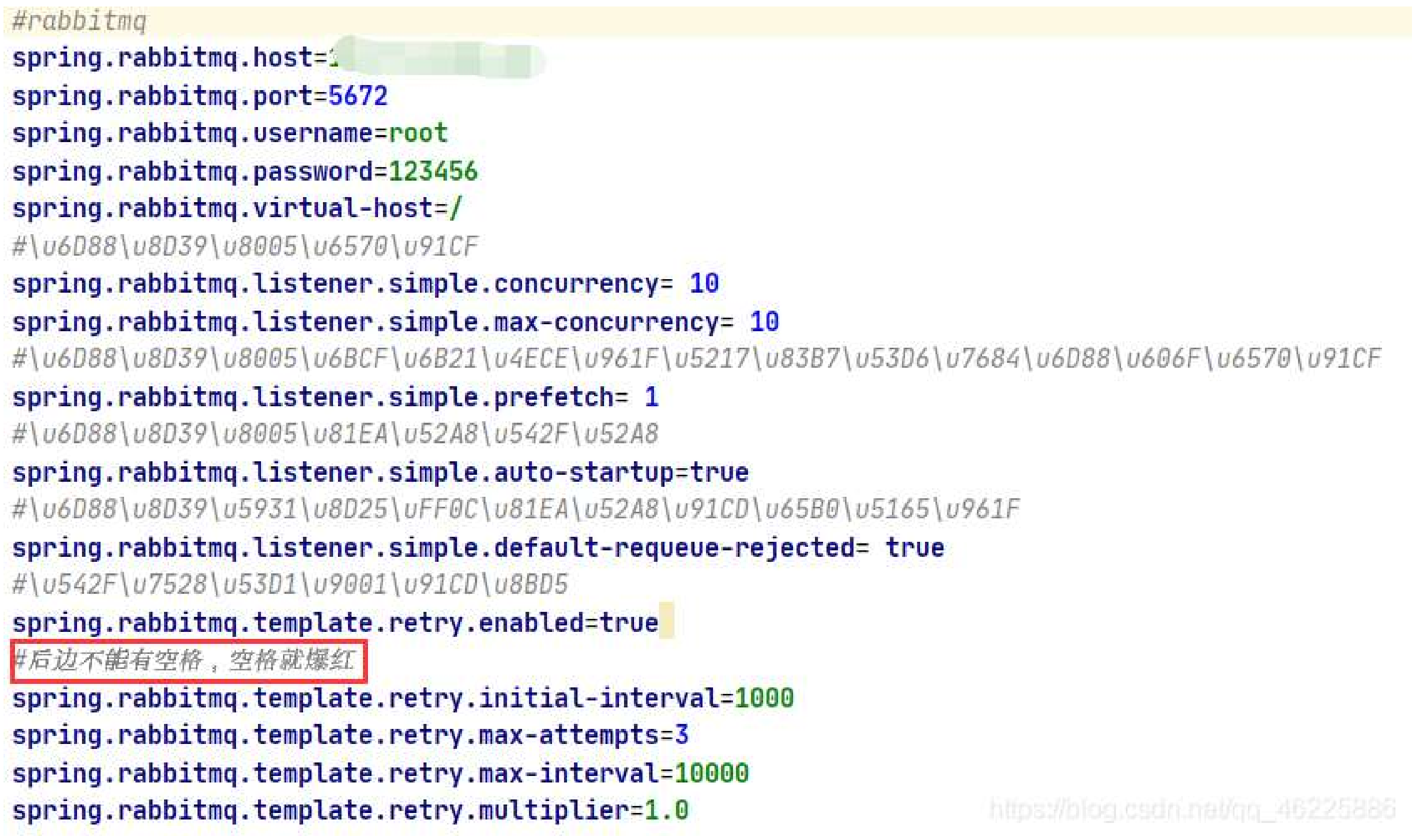
---

1. 集成RabbitMQ

1.1 添加依赖

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

1.2 添加配置信息



```properties
#rabbitmq
spring.rabbitmq.host=[redacted]
spring.rabbitmq.port=5672
spring.rabbitmq.username=root
spring.rabbitmq.password=123456
spring.rabbitmq.virtual-host=/
#\u6D88\u8D39\u8005\u6570\u91CF
spring.rabbitmq.listener.simple.concurrency= 10
spring.rabbitmq.listener.simple.max-concurrency= 10
#\u6D88\u8D39\u8005\u6BCF\u6B21\u4ECE\u961F\u5217\u83B7\u53D6\u7684\u6D88\u606F\u6570\u91CF
spring.rabbitmq.listener.simple.prefetch= 1
#\u6D88\u8D39\u8005\u81EA\u52A8\u542F\u52A8
spring.rabbitmq.listener.simple.auto-startup=true
#\u6D88\u8D39\u5931\u8D25\uFF0C\u81EA\u52A8\u91CD\u65B0\u5165\u961F
spring.rabbitmq.listener.simple.default-requeue-rejected= true
#\u542F\u7528\u53D1\u9001\u91CD\u8BD5
spring.rabbitmq.template.retry.enabled=true
#后边不能有空格，空格就爆红
spring.rabbitmq.template.retry.initial-interval=1000
spring.rabbitmq.template.retry.max-attempts=3
spring.rabbitmq.template.retry.max-interval=10000
spring.rabbitmq.template.retry.multiplier=1.0
```

---

2. 进行简单测试（Direct Exchange)

- **任何发送到Direct Exchange的消息都会被转发到RouteKey中指定的Queue**

2.1 创建一个配置类

```java
@Configuration
public class MQConfig {

    public static final String QUEUE_NAME = "queue";

    @Bean
    public Queue queue(){
        return new Queue(QUEUE_NAME,true);
    }
}
```

2.1.1 @Bean注解

- @Bean注解就是要告诉 `方法` ，产生一个 `Bean对象` ，并将这个Bean由Spring容器管理。产生这个Bean对象的方法Spring只会调用一次，随后这个Bean将放在IOC容器中。
- SpringIOC容器管理一个或者多个Bean，这些Bean都需要在 `@Configuration` 注解下进行创建

2.2 创建消息的接受器

```
@Service
@Slf4j
public class MQReceiver {

    @RabbitListener(queues = MQConfig.QUEUE_NAME)
    public void receive(String message){
        log.info("receive message:" + message);
    }
}
```

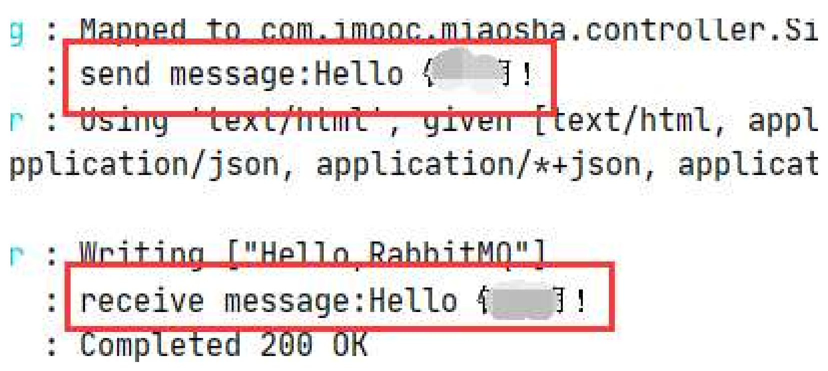
2.2.1 @RabbitListener注解

- `@RabbitListener` ，其中queues属性通过识别队列的名字来接受消息进行消费

2.3 创建消息的发送器

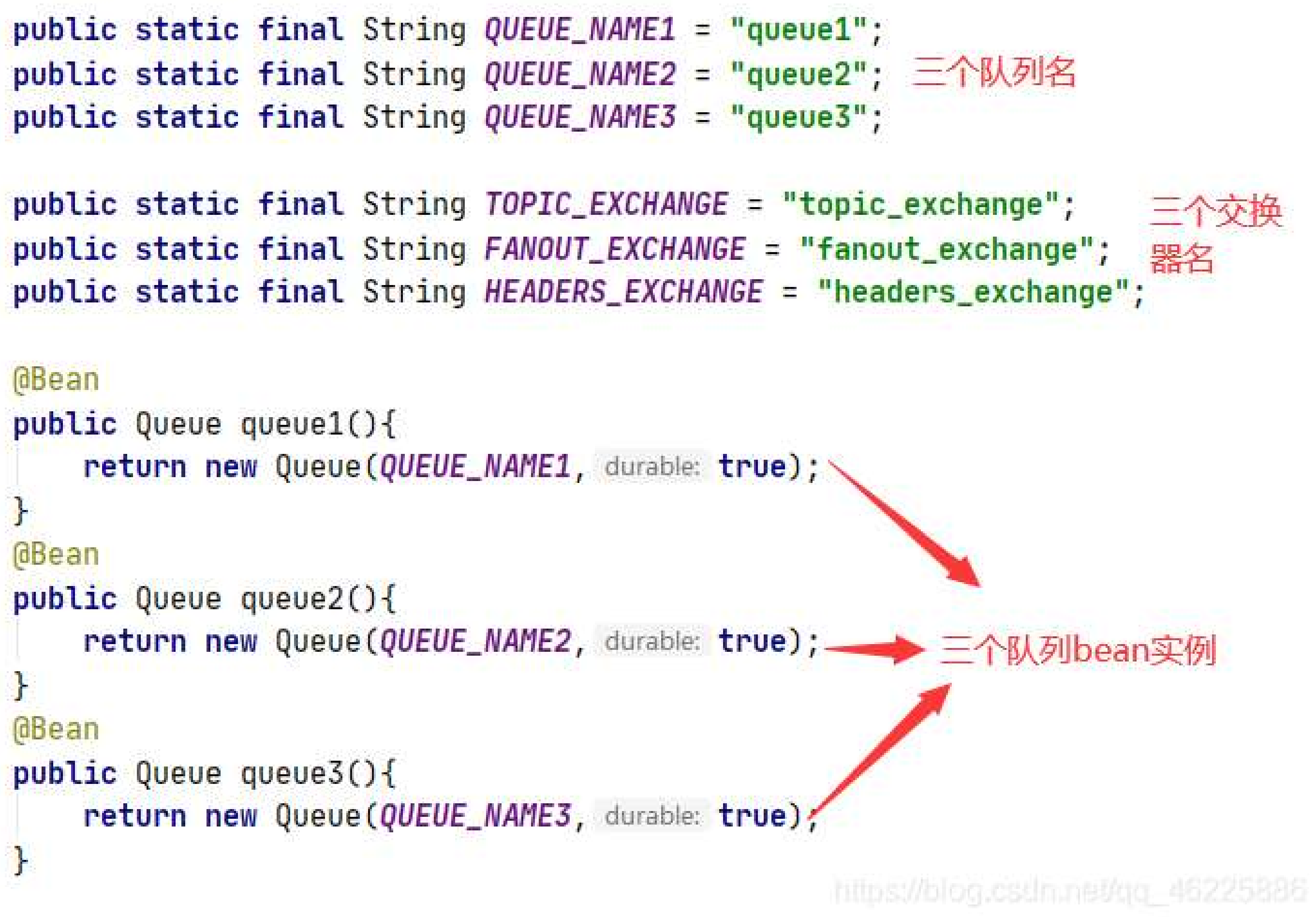
```
@Service
@Slf4j
public class MQSender {

    @Autowired
    //AmqpTemplate接口定义了发送和接收消息的基本操作
    AmqpTemplate amqpTemplate;

    public void send(Object message){
        String msg = RedisService.beanToString(message);
        log.info("send message:" + msg);
        amqpTemplate.convertAndSend(MQConfig.QUEUE_NAME,msg);
        }
    }
```

测试通过↓



```
g : Mapped to com.imooc.miaosha.controller.Si
  : send message:Hello [redacted]!
r : Using 'text/html', given [text/html, appl
pplication/json, application/*+json, applicat

r : Writing ["Hello RabbitMQ"]
  : receive message:Hello [redacted]!
  : Completed 200 OK
```

3. 预先配置

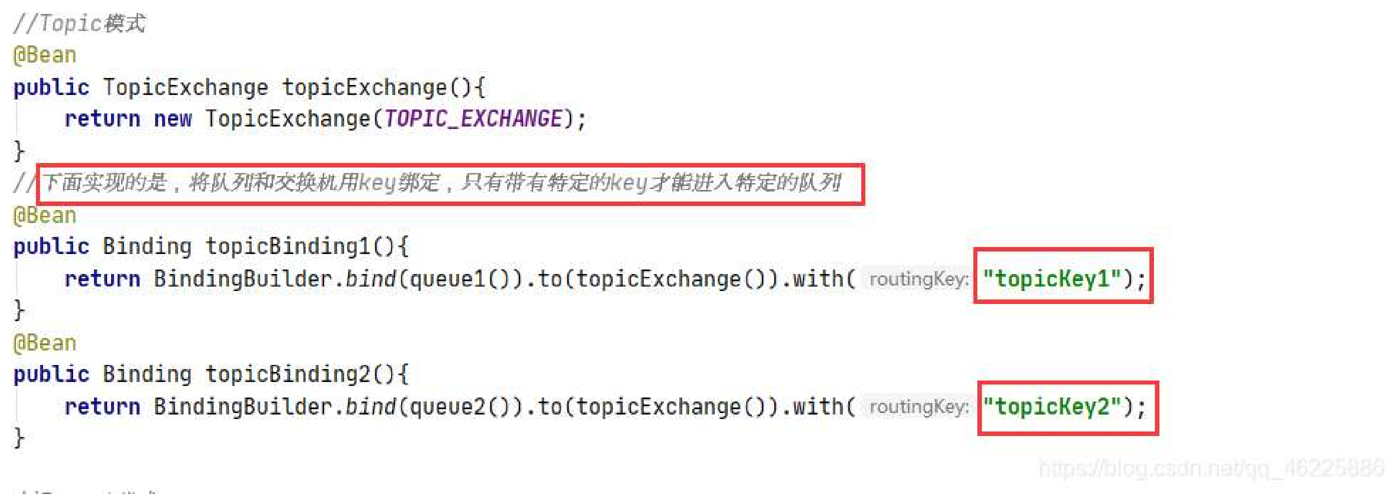


```java
public static final String QUEUE_NAME1 = "queue1";
public static final String QUEUE_NAME2 = "queue2";
public static final String QUEUE_NAME3 = "queue3";

public static final String TOPIC_EXCHANGE = "topic_exchange";
public static final String FANOUT_EXCHANGE = "fanout_exchange";
public static final String HEADERS_EXCHANGE = "headers_exchange";

@Bean
public Queue queue1(){
    return new Queue(QUEUE_NAME1, durable: true);
}
@Bean
public Queue queue2(){
    return new Queue(QUEUE_NAME2, durable: true);
}
@Bean
public Queue queue3(){
    return new Queue(QUEUE_NAME3, durable: true);
}
```

4. Topic Exchange

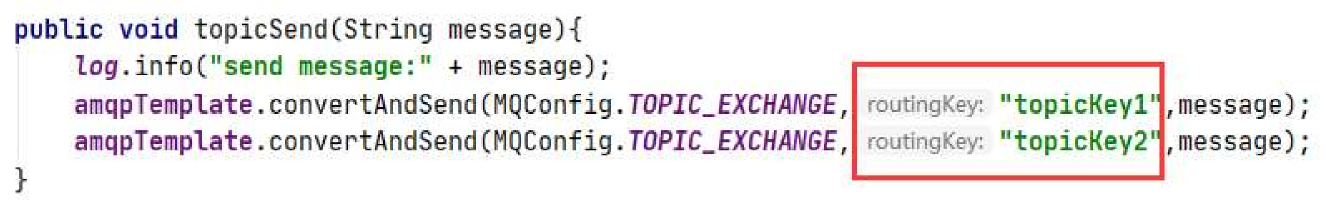
- **任何发送到Topic Exchange的消息都会被转发到与routingKey匹配的队列上**

4.1 进行配置



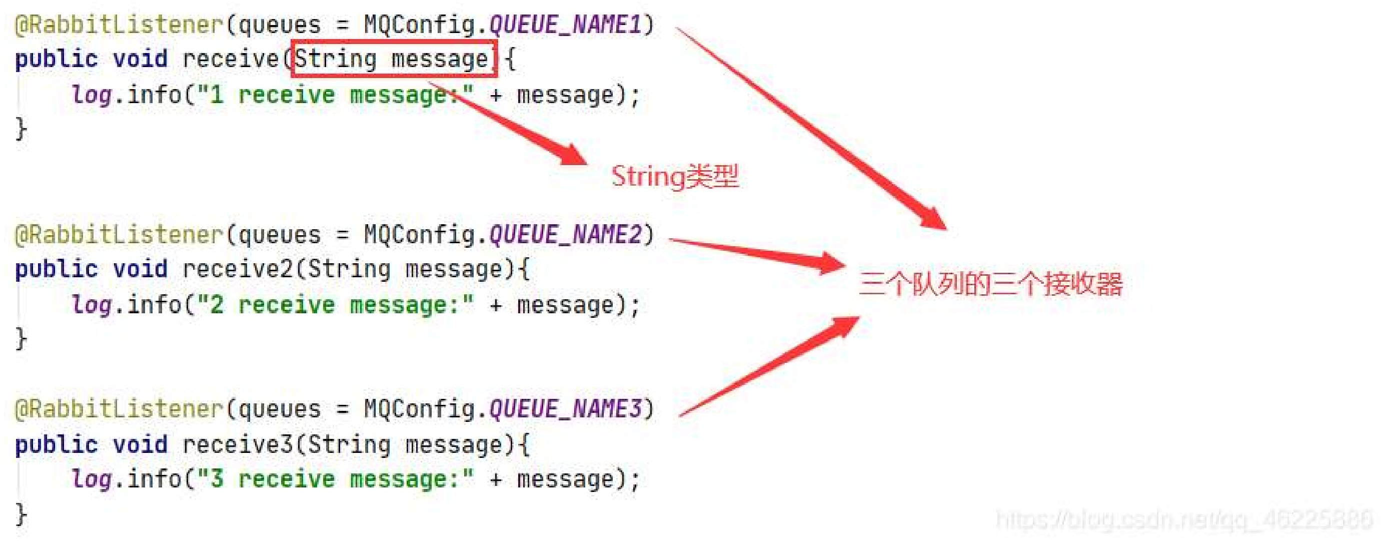
```
//Topic模式
@Bean
public TopicExchange topicExchange(){
    return new TopicExchange(TOPIC_EXCHANGE);
}
//下面实现的是，将队列和交换机用key绑定，只有带有特定的key才能进入特定的队列
@Bean
public Binding topicBinding1(){
    return BindingBuilder.bind(queue1()).to(topicExchange()).with( routingKey: "topicKey1");
}
@Bean
public Binding topicBinding2(){
    return BindingBuilder.bind(queue2()).to(topicExchange()).with( routingKey: "topicKey2");
}
```

4.2 编写消息发送者



```
public void topicSend(String message){
    log.info("send message:" + message);
    amqpTemplate.convertAndSend(MQConfig.TOPIC_EXCHANGE, routingKey: "topicKey1",message);
    amqpTemplate.convertAndSend(MQConfig.TOPIC_EXCHANGE, routingKey: "topicKey2",message);
}
```

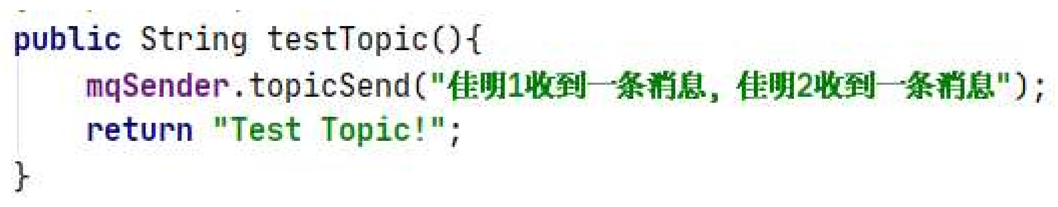
4.3 编写消息接收器



```
@RabbitListener(queues = MQConfig.QUEUE_NAME1)
public void receive(String message){
    log.info("1 receive message:" + message);
}

@RabbitListener(queues = MQConfig.QUEUE_NAME2)
public void receive2(String message){
    log.info("2 receive message:" + message);
}

@RabbitListener(queues = MQConfig.QUEUE_NAME3)
public void receive3(String message){
    log.info("3 receive message:" + message);
}
```

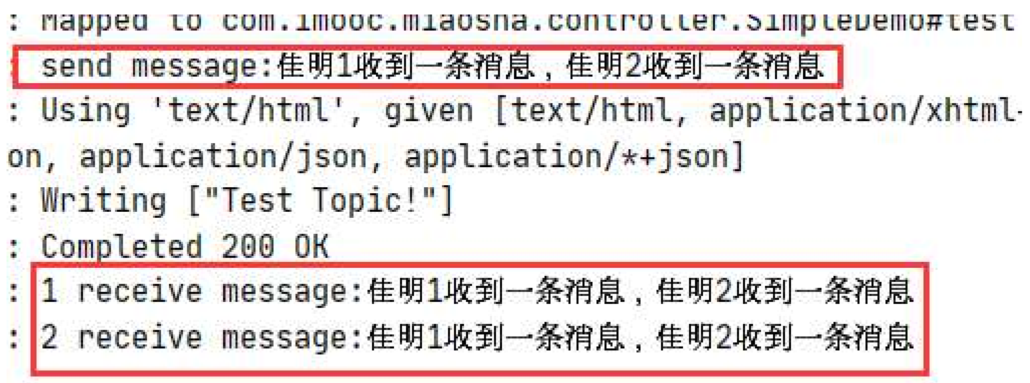
4.4 测试结果

- 我们只绑定了队列1和队列2，根据消息发送者，会为队列1和队列2各发送一条消息，队列1和队列2各收到一条消息
- 测试内容



```
public String testTopic(){
    mqSender.topicSend("佳明1收到一条消息，佳明2收到一条消息");
    return "Test Topic!";
}
```

- 测试结果



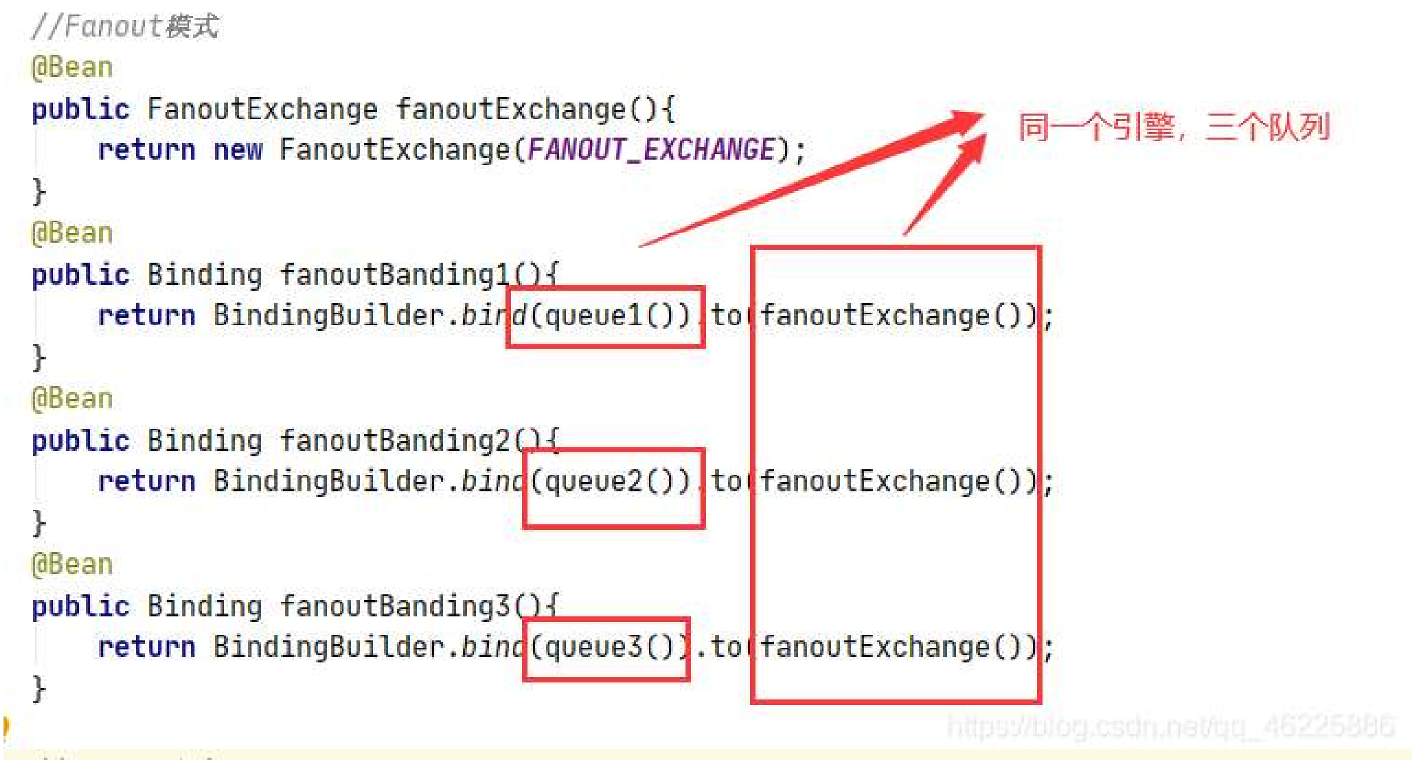
```
: Mapped to com.imooc.miaosha.controller.SimpleDemo#test
: send message:佳明1收到一条消息，佳明2收到一条消息
: Using 'text/html', given [text/html, application/xhtml-
on, application/json, application/*+json]
: Writing ["Test Topic!"]
: Completed 200 OK
: 1 receive message:佳明1收到一条消息，佳明2收到一条消息
: 2 receive message:佳明1收到一条消息，佳明2收到一条消息
```

---

5. Fanout Exchange

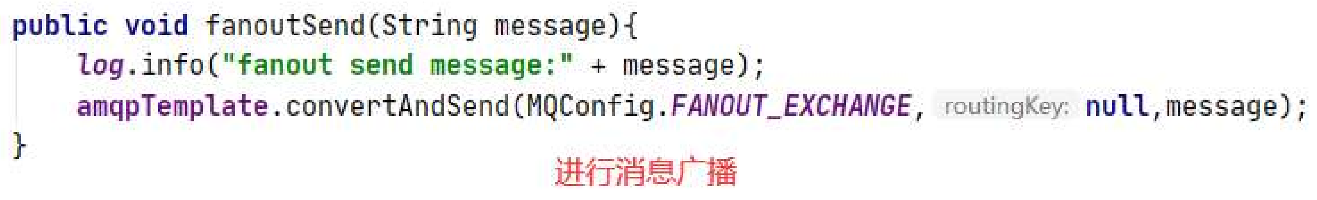
- **任何发送到Fanout Exchange的消息都会被转发到与之绑定的队列上**

5.1 进行配置



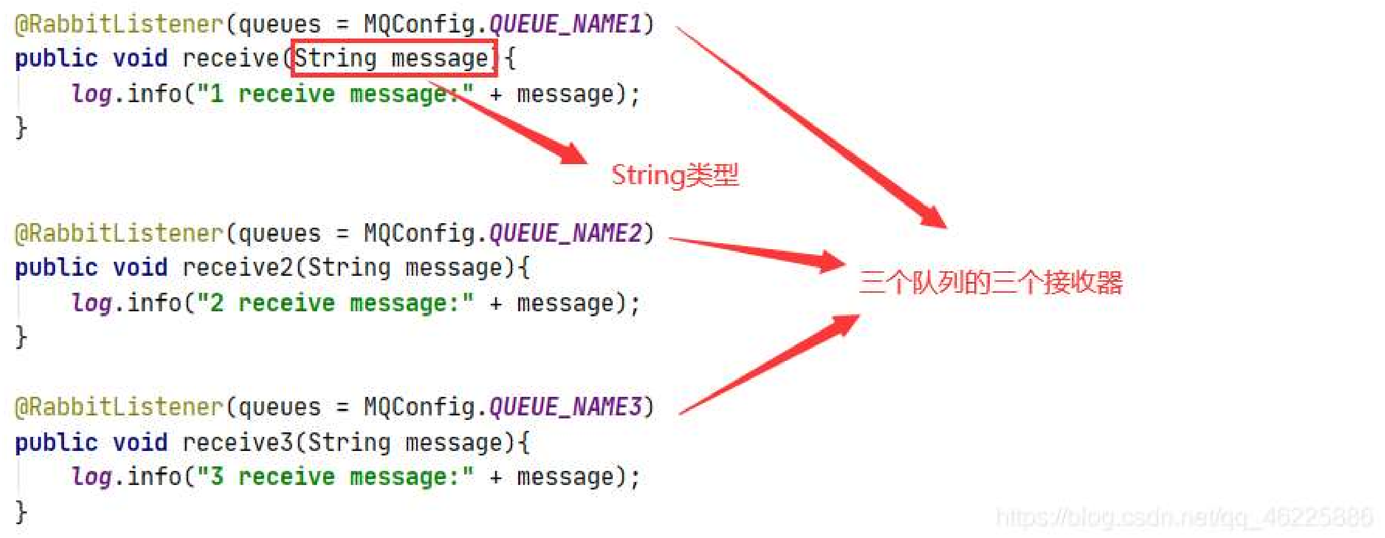
```
//Fanout模式
@Bean
public FanoutExchange fanoutExchange(){
    return new FanoutExchange(FANOUT_EXCHANGE);
}
@Bean
public Binding fanoutBanding1(){
    return BindingBuilder.bind(queue1()).to(fanoutExchange());
}
@Bean
public Binding fanoutBanding2(){
    return BindingBuilder.bind(queue2()).to(fanoutExchange());
}
@Bean
public Binding fanoutBanding3(){
    return BindingBuilder.bind(queue3()).to(fanoutExchange());
}
```

5.2 编写消息发送者



```
public void fanoutSend(String message){
    log.info("fanout send message:" + message);
    amqpTemplate.convertAndSend(MQConfig.FANOUT_EXCHANGE, routingKey: null,message);
}
```

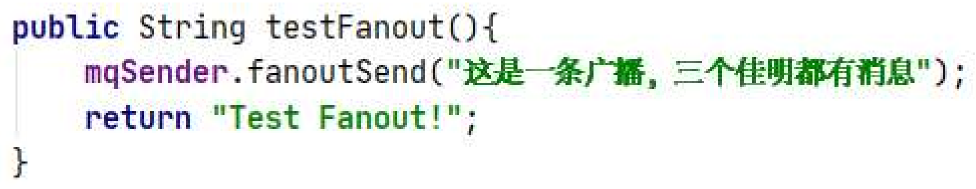
5.3 编写消息接受器



```
@RabbitListener(queues = MQConfig.QUEUE_NAME1)
public void receive(String message){
    log.info("1 receive message:" + message);
}

@RabbitListener(queues = MQConfig.QUEUE_NAME2)
public void receive2(String message){
    log.info("2 receive message:" + message);
}

@RabbitListener(queues = MQConfig.QUEUE_NAME3)
public void receive3(String message){
    log.info("3 receive message:" + message);
}
```

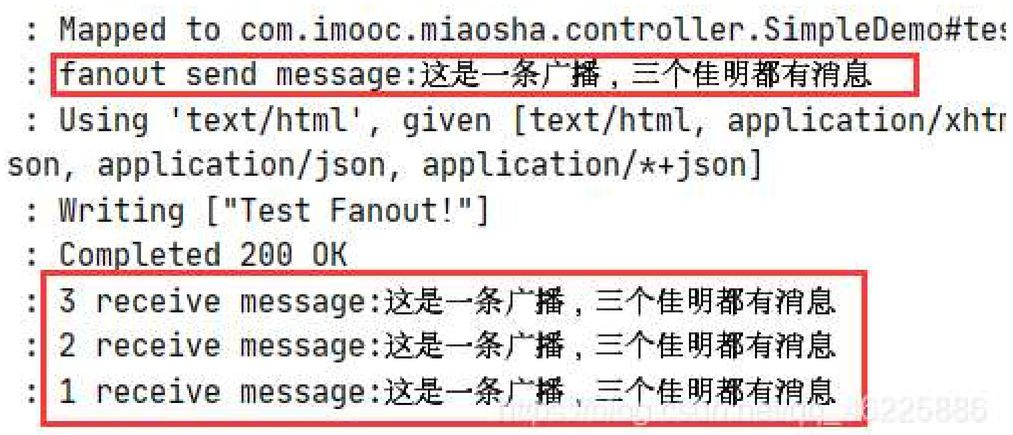
5.4 测试结果

- 根据条件，我们可以知道Fanout Exchange进行广播，每个队列都会收到消息



```
public String testFanout(){
    mqSender.fanoutSend("这是一条广播，三个佳明都有消息");
    return "Test Fanout!";
}
```

- 测试内容
- 测试结果



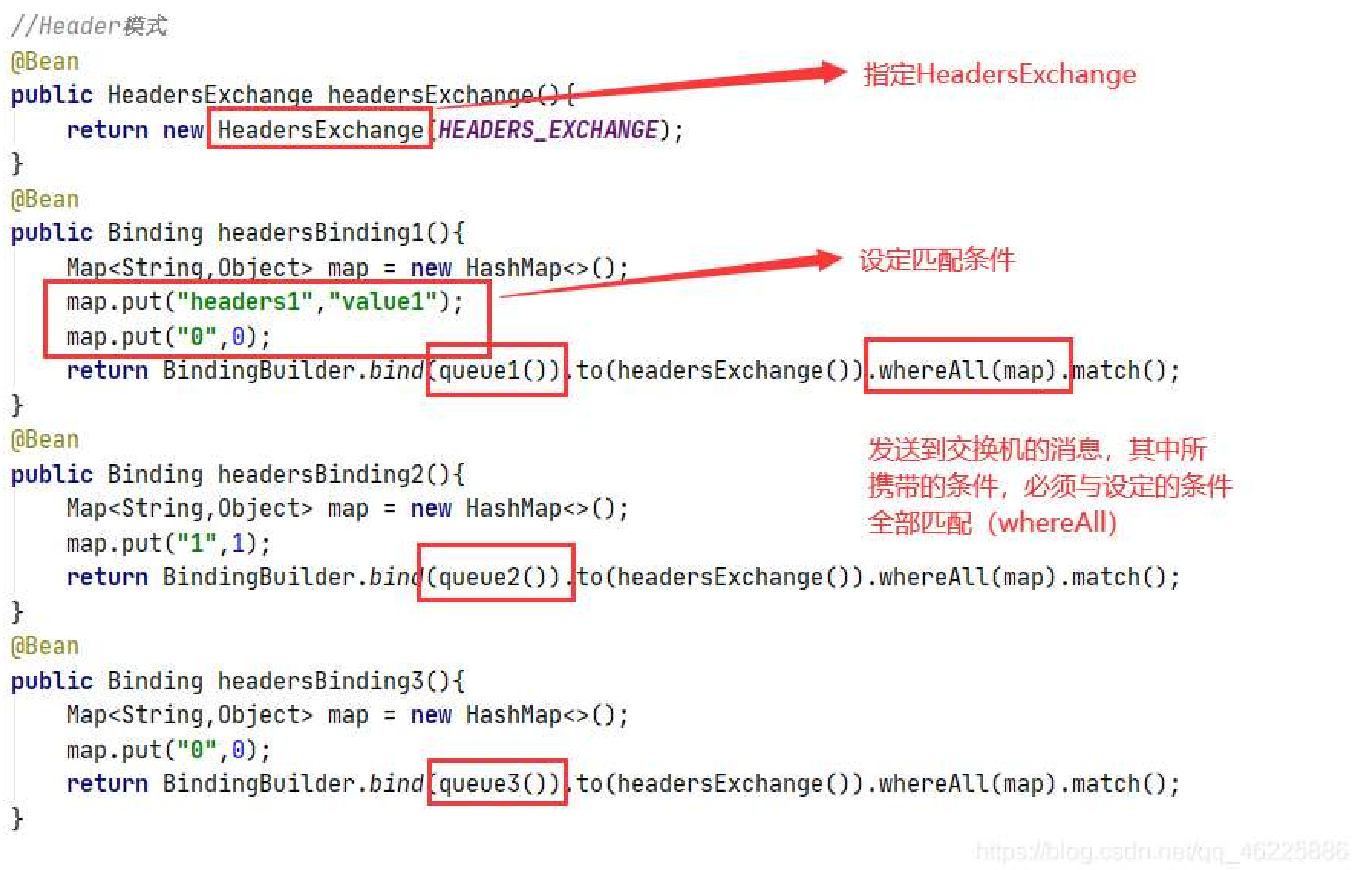
```
 : Mapped to com.imooc.miaosha.controller.SimpleDemo#te
 : fanout send message:这是一条广播，三个佳明都有消息
 : Using 'text/html', given [text/html, application/xht
son, application/json, application/*+json]
 : Writing ["Test Fanout!"]
 : Completed 200 OK
 : 3 receive message:这是一条广播，三个佳明都有消息
 : 2 receive message:这是一条广播，三个佳明都有消息
 : 1 receive message:这是一条广播，三个佳明都有消息
```

---

6. Headers Exchange

- **任何发送到Headers Exchange的消息，都会和其中存储的条件进行匹配，有whereall和whereAny的区别（全部匹配/任何匹配）**

6.1 进行配置



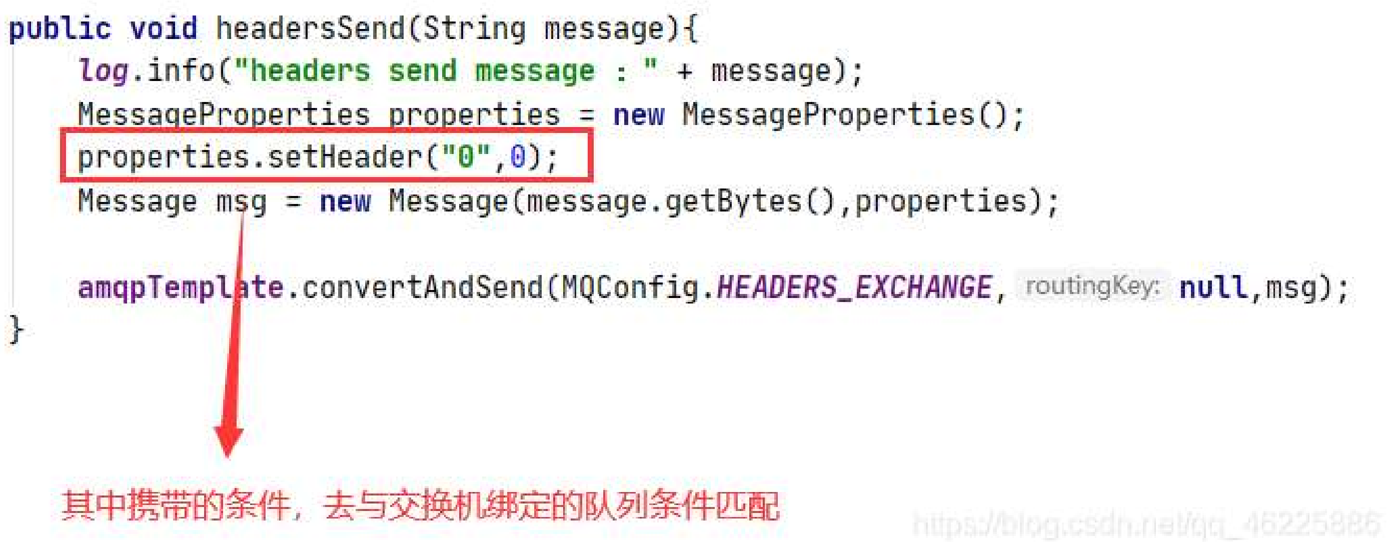
```java
//Header模式
@Bean
public HeadersExchange headersExchange(){
    return new HeadersExchange(HEADERS_EXCHANGE);
}
@Bean
public Binding headersBinding1(){
    Map<String,Object> map = new HashMap<>();
    map.put("headers1","value1");
    map.put("0",0);
    return BindingBuilder.bind(queue1()).to(headersExchange()).whereAll(map).match();
}
@Bean
public Binding headersBinding2(){
    Map<String,Object> map = new HashMap<>();
    map.put("1",1);
    return BindingBuilder.bind(queue2()).to(headersExchange()).whereAll(map).match();
}
@Bean
public Binding headersBinding3(){
    Map<String,Object> map = new HashMap<>();
    map.put("0",0);
    return BindingBuilder.bind(queue3()).to(headersExchange()).whereAll(map).match();
}
```

指定HeadersExchange

设定匹配条件

发送到交换机的消息，其中所携带的条件，必须与设定的条件全部匹配（whereAll）

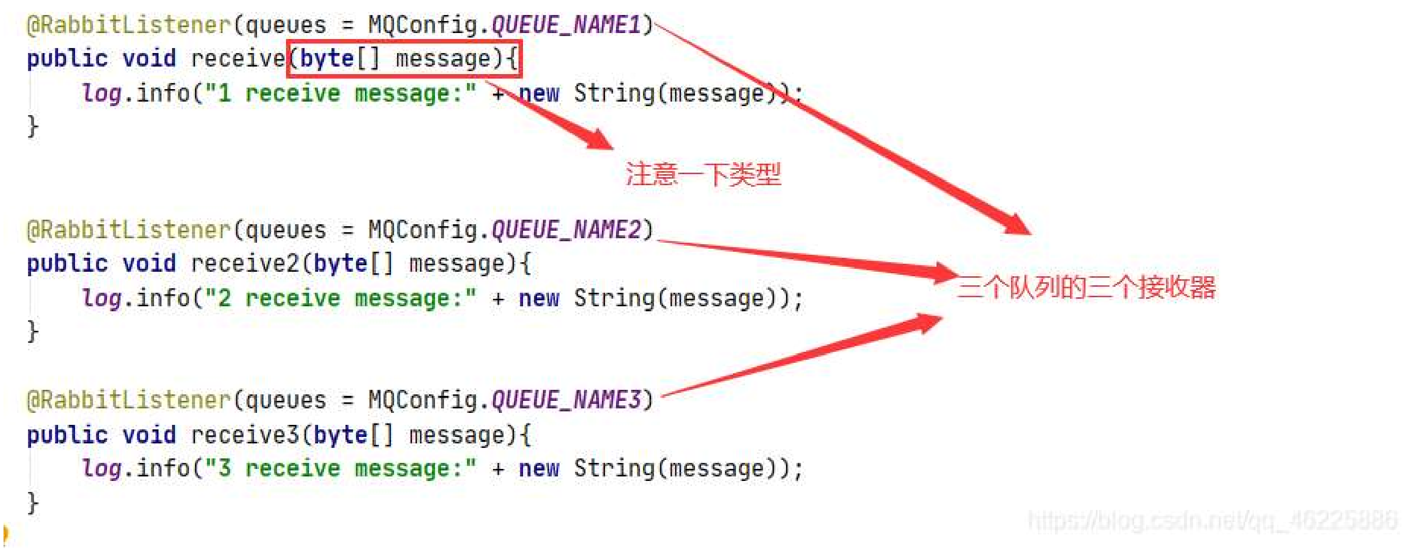
6.2 编写消息发送者



```java
public void headersSend(String message){
    log.info("headers send message : " + message);
    MessageProperties properties = new MessageProperties();
    properties.setHeader("0",0);
    Message msg = new Message(message.getBytes(),properties);

    amqpTemplate.convertAndSend(MQConfig.HEADERS_EXCHANGE, routingKey: null,msg);
}
```

其中携带的条件，去与交换机绑定的队列条件匹配

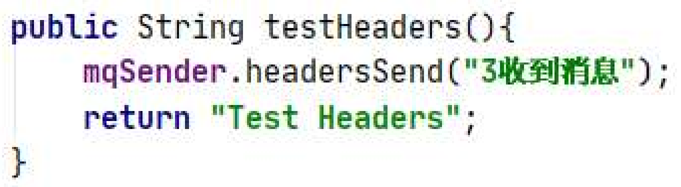
6.3 编写消息接收器



```
@RabbitListener(queues = MQConfig.QUEUE_NAME1)
public void receive(byte[] message){
    log.info("1 receive message:" + new String(message));
}

@RabbitListener(queues = MQConfig.QUEUE_NAME2)
public void receive2(byte[] message){
    log.info("2 receive message:" + new String(message));
}

@RabbitListener(queues = MQConfig.QUEUE_NAME3)
public void receive3(byte[] message){
    log.info("3 receive message:" + new String(message));
}
```

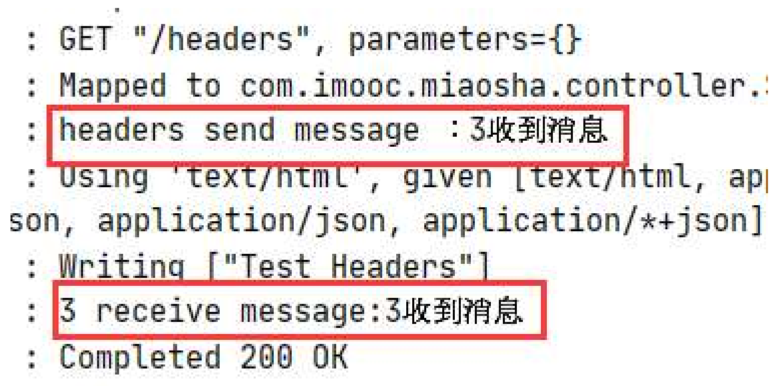
6.4 测试结果

- 根据匹配条件我们可以知道，只有3队列能接受到消息。



```
public String testHeaders(){
    mqSender.headersSend("3收到消息");
    return "Test Headers";
}
```

- 测试内容



```
: GET "/headers", parameters={}
: Mapped to com.imooc.miaosha.controller.
: headers send message  :3收到消息
: Using 'text/html', given [text/html, ap
son, application/json, application/*+json]
: Writing ["Test Headers"]
: 3 receive message:3收到消息
: Completed 200 OK
```

- 测试结果