RabbitMQ学习

1 RabbitMQ相关概念介绍

1.1 组件介绍

RabbitMQ中主要包含生产者（producer）、消费者（consumer）、消息中间件服务节点(Broker)。

模型如下图所示:

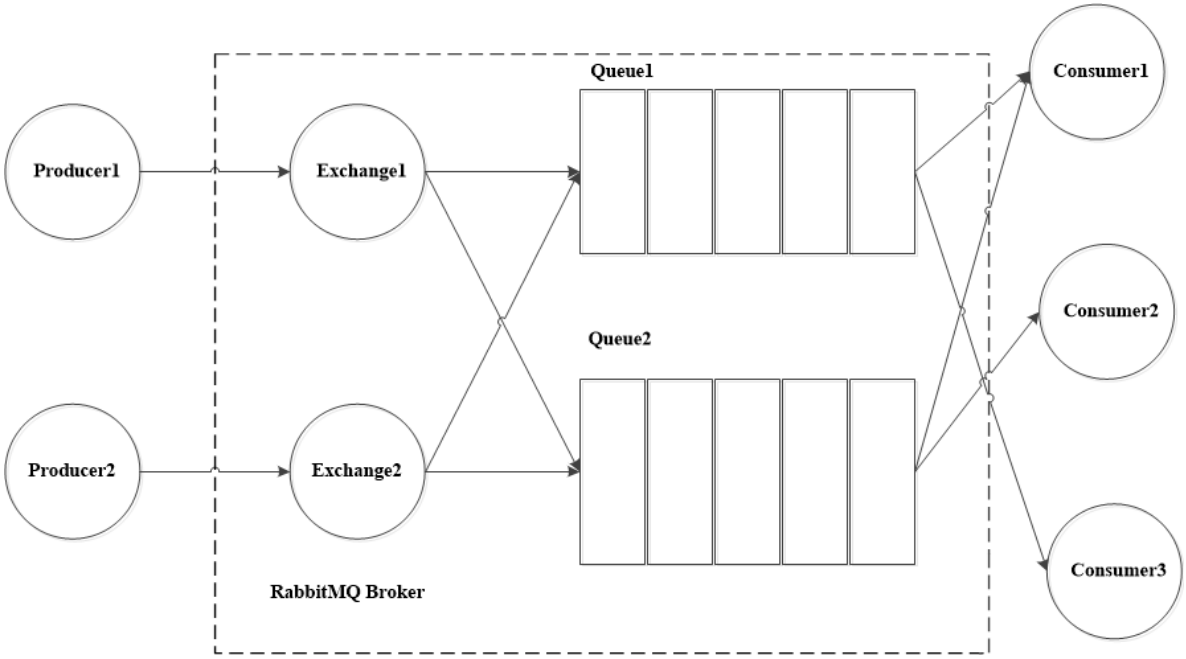


图1.1 RabbitMQ的模型架构

生产者，就是投递消息的一方。生产者创建消息，然后发布到RabbitMQ 中。

消费者， 就是接收消息的一方。消费者连接到RabbitMQ 服务器，并订阅到队列上。

Broker， 消息中间件的服务节点。

Queue: 队列，是RabbitMQ 的内部对象，用于存储消息。

1.2 交换器、路由键、绑定

交换器（Exchange）：

生产者将消息发送到Exchange (交换器，通常也可以用大写的"X" 来表示)，由交换器将消息路由到一个或者多个队列中。如果路由不到，或许会返回给生产者，或许直接丢弃。如图1.1所所示的从生产者—交换机—队列的路线。

路由键（RoutingKey）：

生产者将消息发给交换器的时候， 一般会指定一个RoutingKey ，用
来指定这个消息的路由规则，而这个RoutingKey 需要与交换器类型和绑定键(BindingKey) 联合使用才能最终生效。

绑定（Binding）

RabbitMQ 中通过绑定将交换器与队列关联起来，在绑定的时候一般会指定一个绑定键(BindingKey)，这样RabbitMQ 就知道如何正确地将消息路由到队列了。生产者将消息发送给交换器时， 需要一个RoutingKey， 当BindingKey 和RoutingKey 相匹配时， 消息会被路由到对应的队列中。BindingKey 并不是在所有的情况下都生效，它依赖于交换器类型， 比如fanout 类型的交换器就会无视BindingKey ， 而是将消息路由到所有绑定到该交换器的队列中。

1.3 交换器类型

RabbitMQ 常用的交换器有fanout 、direct、topic 、headers 这四种类型。

fanout:它会把所有发送到该交换器的消息路由到所有与该交换器绑定的队列中。

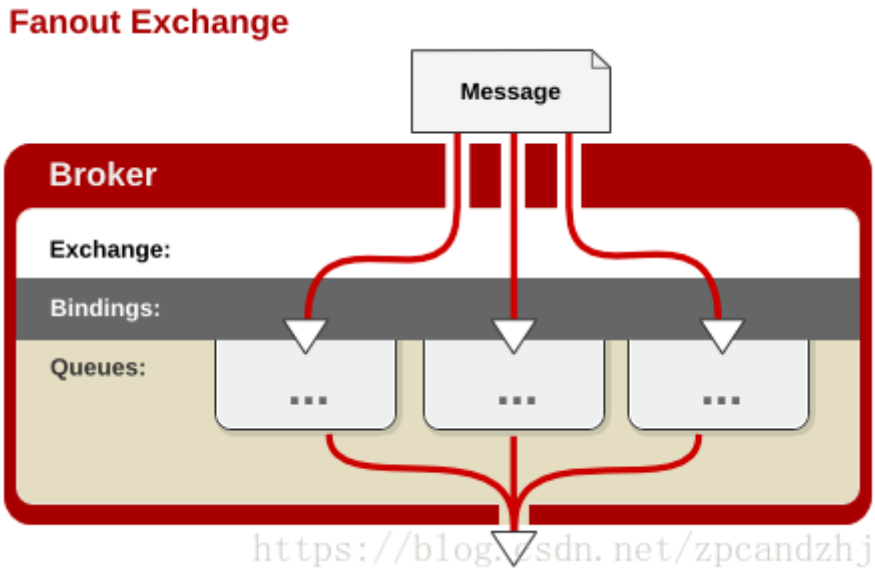


图1.2 fanout交换器

channel.basicPublish(EXCHANGE_NAME,"", null, message.getBytes()); 路由键此时无作用。消息会被发送到与交换器绑定的所有队列中。

direct:会把消息路由到那些BindingKey 和RoutingKey完全匹配的队列中。

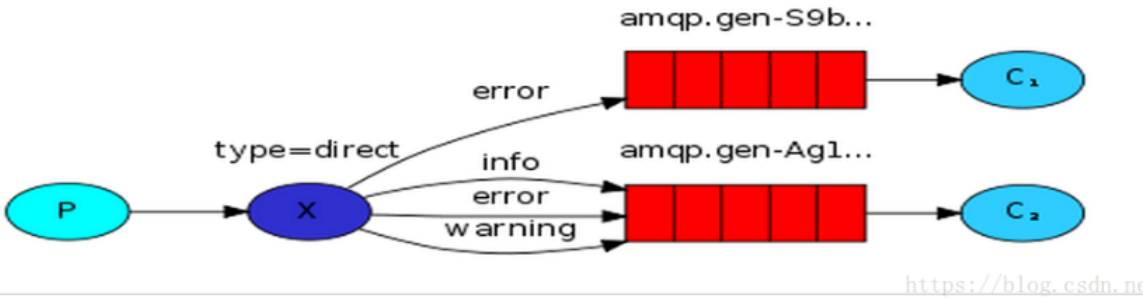


图1.3 direct交换器

交换器的类型为direct，如果我们发送一条消息，并在发送消息的时候设置路由键为" error" ，则消息会路由Queue1和Queue2 中，对应的示例代码为：

channel.basicPublish(EXCHANGE_NAME, "warning" ,message.getBytes()) ;

如果在发送消息的时候设置路由键为" info" 或者"warning" ，消息只会路由到Queue2 。

topic:与direct 类型的交换器相似，也是将消息路由到BindingKey 和RoutingKey 相匹配的队列中，但这里的匹配规则有些不同，它约定:
(1)RoutingKey 为一个点号". "分隔的字符串(被点号". "分隔开的每一段独立的字符串称为一个单词)，如"com.rabbitmq.client"
(2)BindingKey 和RoutingKey 一样也是点号" ."分隔的字符串;
(3)BindingKey 中可以存在两种特殊字符串"*"和"#"，用于做模糊匹配，其中"**"用于匹配一个单词，"#"用于匹配多规格单词(可以是零个)。

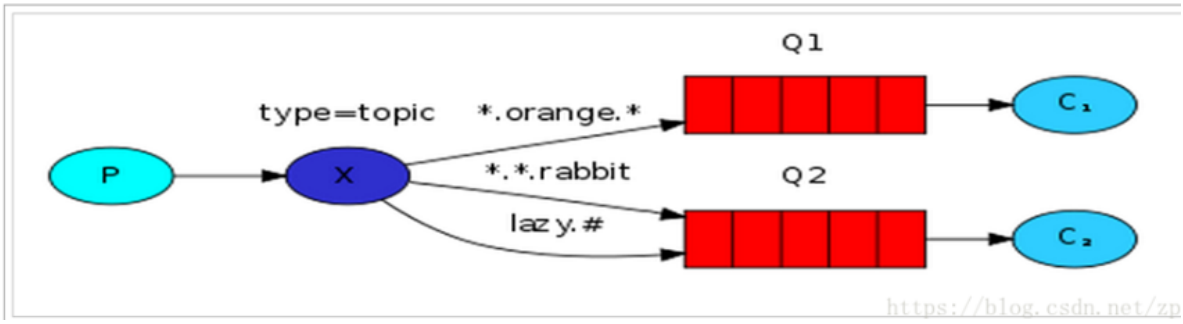


图1.3 topic交换器

如图1.3所示:

com.orange.lala会匹配到队列1中；com.orange.rabbit会匹配到队列1和队列2中。

2 RabbitMQ安装与配置

2.1 安装包下载

由于RabbitMQ是用Erlang语言编写的，因此需要先安装Erlang：http://www.erlang.org/downloads

RabbitMQ下载路径：http://www.rabbitmq.com/install-windows-manual.html

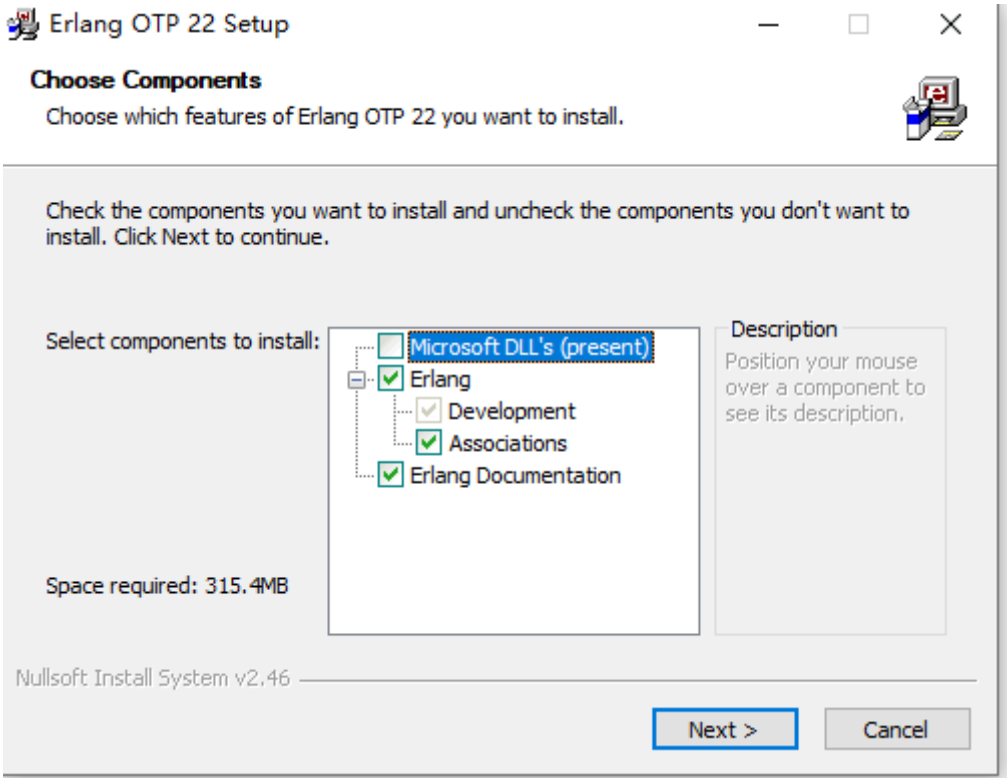
安装包展示:

otp_win64_22.0	2019/9/1 8:39	应用程序	91,890 KB
rabbitmq-server-3.7.17	2019/8/31 21:52	应用程序	9,876 KB

2.2 RabbitMQ安装

2.2.1 Erlang安装

1、双击应用程序进行安装，一路默认选择（直接选择Next）直至安装成功。

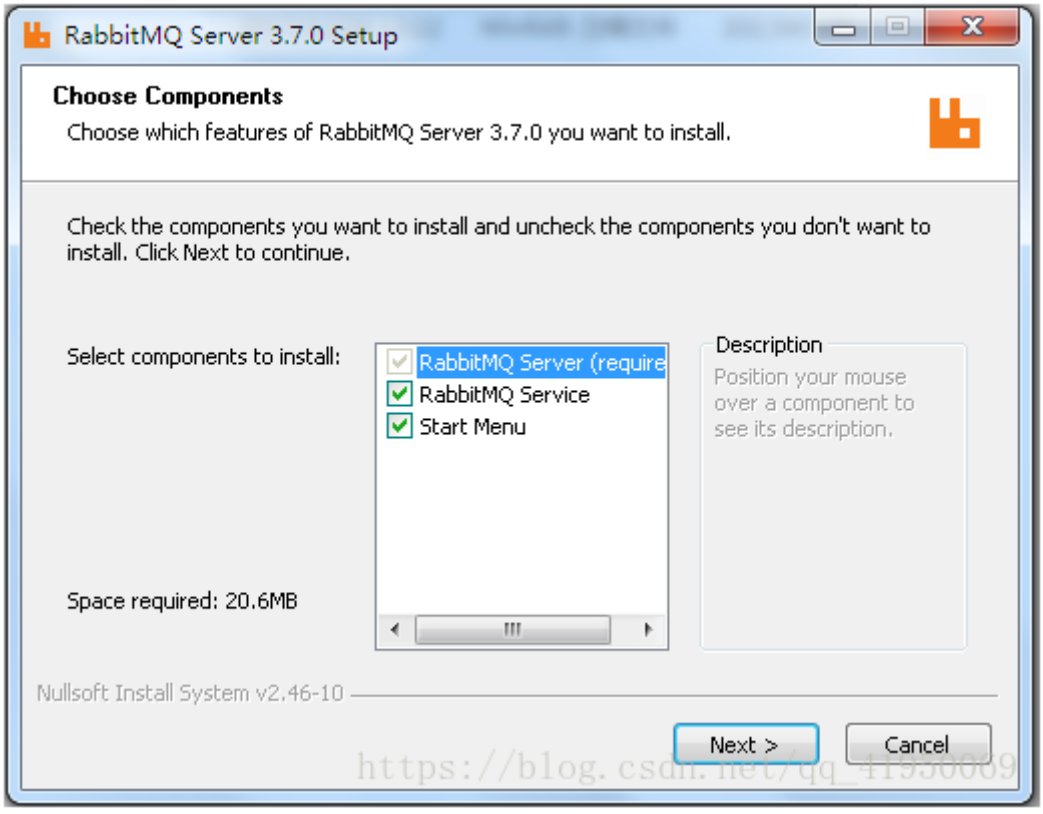


2、增加环境变量ERLANG_HOME=D:\JavaSoftware\rabbitmq\erl10.4（选择自己的安装目录）

3、修改环境变量Path，在原来的值后面加上“%ERLANG_HOME%\bin”

2.2.2 RabbitMQ安装

1、使用默认配置，一路点击Next直至安装完成。



2、增加环境变量RABBITMQ_HOEM=D:\JavaSoftware\rabbitmq\rabbitmq-server-3.7.17\rabbitmq_server-3.7.17

3、修改环境变量Path，在原来的值后面加上“%RABBITMQ_HOEM%\sbin”

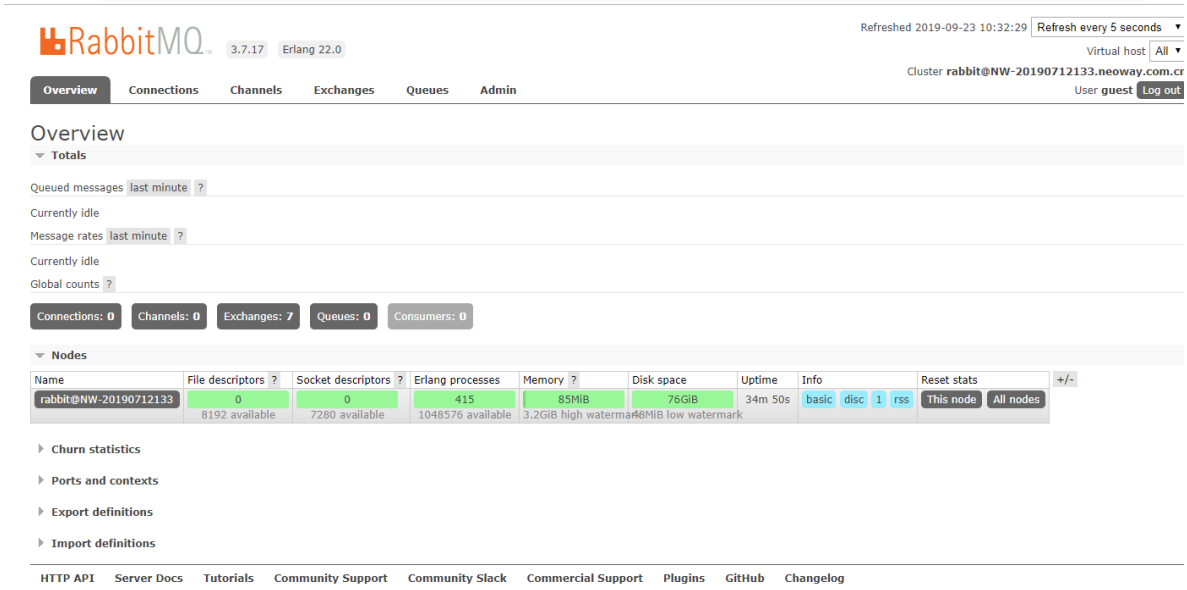
2.3 启动管理页面

1、运行命令rabbitmq-plugins enable rabbitmq_management 开启Web管理插件

```
C:\Users\20190712133>rabbitmq-plugins enable rabbitmq_management
Enabling plugins on node rabbit@NW-20190712133:
rabbitmq_management
The following plugins have been configured:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch
Applying plugin configuration to rabbit@NW-20190712133...
The following plugins have been enabled:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch

started 3 plugins.
```

2、通过浏览器访问http://localhost:15672，并通过默认用户guest进行登录，密码也是guest，登录后的页面：



3 客户端开发

3.1 RabbitMQ连接

```
<rabbit:connection-factory id="connectionFactory" host="127.0.0.1"
                                port="5672" username="admin" password="lj..0000"
                                 virtual-host="host"/>
```

```
AbstractApplicationContext context =
                new ClassPathXmlApplicationContext("classpath:rabbitmq-
context.xml");
```

3.2 交换器和队列

3.2.1 exchangeDeclare方法

```
Exchange.DeclareOk exchangeDeclare(String exchange ,String type , boolean
durable,
boolean autoDelete , boolean internal ,Map<String, Object> arguments) throws
IOException;
```

exchange : 交换器的名称

type : 交换器的类型，常见的有fanout、direct 、topic

durable: 设置是否持久化。durable 设置为true 表示持久化， 反之是非持久化。持久化可以将交换器存盘，在服务器重启的时候不会丢失相关信息。

autoDelete : 设置是否自动删除。autoDelete 设置为true 则表示自动删除。自动删除的前提是至少有一个队列或者交换器与这个交换器绑定， 之后所有与这个交换器绑定的队列或者交换器都与此解绑。注意不能错误地把这个参数理解为: "当与此交换器连接的客户端都断开时， RabbitMQ 会自动删除本交换器" 。

internal : 设置是否是内置的。如果设置为true ， 则表示是内置的交换器，客户端程序无法直接发送消息到这个交换器中，只能通过交换器路由到交换器这种方式。

argument : 其他一些结构化参数，比如a l ternate - exchange

删除交换器的方法

```
//如果isUnused 设置为true ， 则只有在此交换器没有被使用的情况下才会被删除;如果设置false，则无论如何这个    交换器都要被删除
Exchange.DeleteOk exchangeDelete(String exchange, boolean ifUnused) throws
IOException;
```

3.2.2 queueDeclare方法

```
Queue.DeclareOk queueDeclare (String queue,boolean durable,boolean
exclusive,boolean autoDelete, Map<String ,  Object> arguments) throws
IOException;
```

queue: 队列的名称

exclusive: 设置是否排他。为true 则设置队列为排他的。如果一个队列被声明为排他队列，该队列仅对首次声明它的连接可见，并在连接断开时自动删除。这里需要注意三点:排他队列是基于连接(Connection) 可见的，同一个连接的不同信道(Channel)是可以同时访问同一连接创建的排他队列; "首次"是指如果一个连接已经声明了一个排他队列，其他连接是不允许建立同名的排他队列的，这个与普通队列不同:即使该队列是持久化的，一旦连接关闭h或者客户端退出，该排他队列都会被自动删除，这种队列适用于一个客户端同时发送和读取消息的应用场景。

删除队列方法

```
Queue.DeleteOk queueDelete(String queue, boolean ifUnused, boolean ifEmpty )
throws IOException;
```

其中queue 表示队列的名称。ifEmpty 设置为true 表示在队列为空(队列里面没有任何消息堆积)的情况下才能够删除。

3.2.3 queueBind方法

将队列和交换器绑定:

```
//routingKey: 用来绑定队列和交换器的路由键;
Queue.BindOk queueBind(String queue, String exchange, String routingKey,
Map<String, Object>arguments) throws IOException;
```

解除绑定

```
Queue.UnbindOk queueUnbind (String queue, String exchange, String routingKey,
Map<String, Object> arguments) throws IOException;
```

3.2.4 exchangeBind

交换器与交换器绑定

```
Exchange.BindOk exchangeBind(String destination, String source, String
routingKey, Map<String, Object> arguments) throws IOException;
```

3.2.5 发送消息（basicPublish）

```
void basicPublish(String exchange, String routingKey, boolean mandatory,
             boolean immediate, BasicProperties props, byte[] body) throws
IOException ;
```

props : 消息的基本属性集，其包含14 个属性成员，分别有contentType 、content E ncoding 、headers (Map<String , Object>) 、deliveryMode 、priority 、correlationld 、replyTo 、expiration 、messageld、timestamp 、type 、userld 、appld、clusterld。
byte[] body : 消息体(payload) ，真正需要发送的消息.

3.2.6 消费消息

```
String basicConsume(String queue, boolean autoAck, String consumerTag, boolean
noLocal, boolean exclusive, Map<String, Object> arguments, Consumer callback)
throws IOException;
```

queue : 队列的名称:
autoAck : 设置是否自动确认。建议设成false ， 即不自动确认:
consumerTag: 消费者标签，用来区分多个消费者:
noLocal : 设置为true 则表示不能将同一个Connectio口中生产者发送的消息传送给这个Connection 中的消费者:
exclusive : 设置是否排他:
arguments : 设置消费者的其他参数:
callback : 设置消费者的回调函数。用来处理Rabb itMQ 推送过来的消息，比如DefaultConsumer ,使用时需要客户端重写(override) 其中的方法。

4 springBoot结合RabbitMQ

4.1 引入依赖

```
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-amqp</artifactId>
            <version>2.1.5.RELEASE</version>
</dependency>
```

4.2 配置Rabbit连接

```
#rabbitMQ
spring.rabbitmq.host=127.0.0.1
spring.rabbitmq.port=5672
spring.rabbitmq.username=admin
spring.rabbitmq.password=lj..0000
spring.rabbitmq.virtual-host=host
#消息发送到交换机确认机制，是否确认回调
spring.rabbitmq.publisher-confirms=true
spring.rabbitmq.publisher-returns=true
server.port=8080
```

4.3 配置类

配置类中定义交换机、队列、交换机与队列绑定关系

```
@Configuration
public class TopicRabbitConfig {
    final static String message = "q_topic_message";
    final static String messsages = "q_topic_messages";
    @Bean(name = "message")
    public Queue queueMessage(){
        return new Queue(TopicRabbitConfig.message);
    }
    @Bean
    public Queue queueMessages(){
        return new Queue(TopicRabbitConfig.messsages);
    }
    /**
     * 声明Topic交换机
     * @return
     */
    @Bean
    TopicExchange exchange(){
        return new TopicExchange("myExchange");
    }
    /**
     * 将队列绑定到交换机，并指定routingKey
     * @param message
     * @param exchange
     * @return
     */
    @Bean
    Binding bindingExchangeMessage(Queue message,TopicExchange exchange){
        return BindingBuilder.bind(message).to(exchange).with("topic.message");
    }
    /**
     * 路由键中"#"可以匹配多个单词
     * @param queueMessages
     * @param exchange
     * @return
     */
    @Bean
    Binding bindingExchangeMessages(Queue queueMessages ,TopicExchange exchange)
{
        return BindingBuilder.bind(queueMessages).to(exchange).with("topic.#");
    }
}
```

4.4 编写生产者代码

```
@Component
public class TopicSender {
    //spring提供了AmqpTemplate，已建立好连接
    @Autowired
    private AmqpTemplate rabbitTemplate;
    public void send1(){
        String context = "message 1 send message";
        System.out.println("sender: " + context);

 this.rabbitTemplate.convertAndSend("myExchange","topic.message",context);
    }
    public void send2(){
        String context = "message 2 send message";
        System.out.println("sender: " + context);

 this.rabbitTemplate.convertAndSend("myExchange","topic.messages",context);
    }
}
```

4.5 编写消费者代码

```
@Component
@RabbitListener(queues = "q_topic_message")
public class TopicReceiver1 {
    @RabbitHandler
    public void receive(String msg){
        System.out.println("receiver1: " + msg);
    }
}
```

```
@Component
@RabbitListener(queues = "q_topic_messages")
public class TopicReceiver2 {
    @RabbitHandler
    public void receive(String  msg){
        System.out.println("receiver2: " + msg);
    }
}
```

4.6 测试

```
    @Test
    public void send1() throws Exception{
        topicSender .send1();
    }
    @Test
    public void send2() throws  Exception{
        topicSender .send2();
    }
```